

Algoritmy – prezentace k přednáškám

doc. Mgr. Jiří Dvorský, Ph.D.

Stav prezentace ke dni 28. dubna 2024

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



1/776

Aktuální verze prezentací

Prezentace jsou průběžně, podle potřeb výuky, doplňovány a aktualizovány. Aktuální verzi prezentací najdete vždy na webu předmětu

www.cs.vsb.cz/dvorsky/Algorithms_Slides.html

2/776

Stručná osnova všech přednášek

Algoritmy I

Úvodní přednáška předmětu

Co je to algoritmus. Strategie řešení problémů pomocí algoritmů. Významné typy řešených problémů.

Analýza složitosti algoritmů

Strategie řešení problémů hrubou silou a úplným prohledáváním

3/776

Stručná osnova všech přednášek (pokrač.)

Strategie řešení sníží a vyřeš

Snížení o konstantní faktor

Snížení o proměnný faktor

Strategie řešení rozděl a panuj

Algoritmy II

Úvodní přednáška předmětu

4/776

Stručná osnova všech přednášek (pokrač.)

Strategie řešení transformuj a vyřeš

Záměna paměťové a časové složitosti

Dynamické programování

Hladové algoritmy

Strategie řešení iterativním zlepšováním

5/776

Stručná osnova všech přednášek (pokrač.)

Meze možností algoritmického řešení problémů. P, NP a NP-úplné problémy.

Zdolávání mezí možností algoritmického řešení problémů

Ostatní

Přílohy

6/776

Celková osnova všech přednášek

Algoritmy I

Úvodní přednáška předmětu

O předmětu Algoritmy I

Prezenční forma studia

Výuka

Úkoly a jejich hodnocení

Kombinovaná forma studia

Výuka

Úkoly a jejich hodnocení

Software pro výuku

7/776

Celková osnova všech přednášek (pokrač.)

Studijní literatura

Co je to algoritmus. Strategie řešení problémů pomocí algoritmů. Významné typy řešených problémů.

Co je to algoritmus?

Základy algoritmického řešení problémů

Důležité typy problémů

Základní datové struktury

Lineární datové struktury

Grafy

Stromy

8/776

Celková osnova všech přednášek (pokrač.)

Množiny a slovníky

Analýza složitosti algoritmů

Základy analýzy složitosti algoritmů

Nejhorsí, nejlepší a průměrný případ

Asymptotické notace složitosti

Analýza nerekurzivních algoritmů

Analýza rekurzivních algoritmů

Strategie řešení problémů hrubou silou a úplným prohledáváním

Třídící algoritmy

9/776

Celková osnova všech přednášek (pokrač.)

Třídění výběrem – SelectSort

Bublinové třídění – BubbleSort

Sekvenční vyhledávání

Vyhledávání podřetězce hrubou silou

Problém nejbližší dvojice bodů

Konvexní obal množiny

Úplné prohledávání

Problém obchodního cestujícího

Problém batohu

10/776

Celková osnova všech přednášek (pokrač.)

Průchody grafem

Průchod grafem do hloubky

Průchod grafem do šířky

Strategie řešení sniž a vyřeš

Třídění vkládáním – InsertSort

Topologické třídění

Generování kombinatorických objektů

Generování permutací

Generování podmnožin

Snižování o konstantní faktor

11/776

Celková osnova všech přednášek (pokrač.)

Snižování o proměnný faktor

Strategie řešení rozděl a panuj

Násobení velkých celých čísel

Strassenovo násobení matic

Problém nejbližší dvojice bodů

Konvexní obal množiny

Algoritmy II

Úvodní přednáška předmětu

O předmětu Algoritmy II

12/776

Celková osnova všech přednášek (pokrač.)

Prezenční forma studia

Výuka

Úkoly a jejich hodnocení

Kombinovaná forma studia

Výuka

Úkoly a jejich hodnocení

Software pro výuku

Studijní literatura

Strategie řešení transformuj a vyřeš

Předtřídění dat

13/776

Celková osnova všech přednášek (pokrač.)

Jedinečnost prvků v poli

Výpočet modu

Vyhledávání

Gaussova eliminační metoda

LU-rozklad matice

Inverzní matice

Determinant matice

Vyvážené vyhledávací stromy

AVL stromy

2-3 stromy

14/776

Celková osnova všech přednášek (pokrač.)

Halda a třídění haldou

Hornerovo schéma

Redukce problému

Záměna paměťové a časové složitosti

B-stromy

Vyhledávání klíče v B-stromu

Vkládání klíče do B-stromu

Smazání klíče z B-stromu

Dynamické programování

Hladové algoritmy

15/776

Celková osnova všech přednášek (pokrač.)

Minimální kostra grafu

Primův algoritmus

Kruskalův algoritmus

Dijkstrův algoritmus

Huffmanův kód

Strategie řešení iterativním zlepšováním

Meze možností algoritmického řešení problémů. P, NP a NP-úplné problémy.

Zdolávání mezí možností algoritmického řešení problémů

Ostatní

16/776

Přílohy

Zásobník

Možnosti implementace

Použití

Fronta

Možnosti implementace

Použití

Binární strom

Budování binárního stromu

Vkládání do binárního vyhledávacího stromu

17/776

Úvodní přednáška předmětu

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



18/776

Úvodní přednáška předmětu O předmětu Algoritmy I

O předmětu Algoritmy I

Upozornění

Všechny aktuální informace k předmětu naleznete na

<http://www.cs.vsb.cz/dvorsky/>

Tato prezentace slouží jen pro účely úvodní přednášky
a nebude dále aktualizována.

19/776

O předmětu Algoritmy I

- Náplní předmětu jsou základní strategie algoritmického řešení úloh (hrubá síla, rozděl a panuj atd.) a typické příklady jejich užití.
- Přednášky jsou zaměřeny na **teorii**.
- Cvičení jsou zaměřena na **implementaci** řešení problémů danou strategií v jazyce C resp. C++.
- Vazby na další předměty:
 - Úvod do programování – jazyk C,
 - Funkcionální programování – rekurze a
 - Objektivě orientované programování – asi není třeba komentáře.

20/776

Rozsah předmětu, způsob zakončení

Rozsah předmětu

- výuka probíhá v letním semestru prvního ročníku bakalářského studia
- hodinová dotace:
 - 2 hodiny přednášky a 2 hodiny cvičení týdně v prezenční formě a
 - 6 tutoriálů v kombinované formě studia

Zakončení – **klasifikovaný zápočet**

- klasifikovaný zápočet není zkouška, řídí jinými pravidly,
- prostudujte si proto **Studijní a zkušební řád pro studium v bakalářských a magisterských studijních programech**, článek 12.

21/776

Garant předmětu

doc. Mgr. Jiří Dvorský, Ph.D.

Kancelář: EA441

Email: jiri.dvorsky@vsb.cz

Web: www.cs.vsb.cz/dvorsky



K čemu je garant předmětu?

Garant předmětu zodpovídá za průběh výuky celého předmětu, průběh cvičení, plnění úkolů na cvičeních a za korektní hodnocení úkolů. Problémy spojené se cvičeními řešte primárně se svým cvičícím. Nepodaří-li dosáhnout řešení problému s cvičícím obraťte se na garanta předmětu.

22/776

Prerekvizity

- Prerekvizity jsou souhrnem požadavků, které je nutné splnit, aby si student mohl zapsat předmět. Prerekvizity jsou buď formální nebo věcné.
- Formální prerekvizity – žádné
- Věcné prerekvizity:
 - znalosti z předmětu Úvod do programování,
 - středoškolské matematiky a
 - obecná orientace ve výpočetní technice.
- Předmět **Algoritmy I** je ale **povinnou prerekvizitou** navazujícího předmětu **Algoritmy II**.

23/776

Přednášky

- účast na přednáškách je **výrazně doporučena**.

Cvičení

- jsou **povinná**,
- účast a aktivita na cvičeních jsou hodnoceny,
- je nutno získat dostatečné bodové hodnocení.

24/776

Centrum Slunečnice FEI

- <http://slunecnice-fei.vsb.cz/>,
- poskytuje podporu zpřístupňující studium i pro studenty se specifickými nároky,
- lze získat, mimo jiné, zvýšenou časovou dotaci na úkoly.

Výzva

Je vysoce žádoucí, aby studenti, kteří dostanou tuto zvýšenou časovou dotaci, neprodleně kontaktovali svého cvičícího a garanta předmětu, abychom předešli případným problémům!

25/776

Individuální studijní plán

Individuální studijní plán umožňuje, v odůvodněných případech, individuální termíny pro plnění studijních povinností.

Výzva

- Je vysoce žádoucí, aby studenti, kteří získají individuální studijní plán, neprodleně kontaktovali svého cvičícího a garanta předmětu a dohodli se na individuálních termínech plnění úkolů.
- Individuální studijní plán neznamena naprostou libovůli v termínech.
- Individuální studijní plán už vůbec neznamena možnost vybrat si úkoly, které budu plnit a které ne.

26/776

Konzultace

- Pokud nebudete ve výuce něčemu rozumět, potřebujete s čímkoliv poradit nebo vyřešit nějaký problém s přednáškou, cvičeními, testy, Vaší absencí na výuce atd. je možné využít si domluvit **individuální konzultaci**.
- Konzultaci je nutné si domluvit předem, například mailem.
- Pokud potřebujete poradit s učivem, připravte si materiály, které jste si k tématu prostudovali, vypište si co je Vám jasné a kde jste se „zasekli“ a potřebujete poradit.
- Konzultací s vyučujícím nic neriskujete – maximálně se dozvíte co potřebujete.
- Přijďte se zeptat rovnou ke zdroji informací – internetová fóra jsou zaplevelena různými polopravdami i naprostými nesmysly.

27/776

- Prezenční a kombinované studium má specifickou formu výuky.
- Obě formy studia mají specifické podmínky pro splnění předmětu.
- Podle formy Vašeho studia se Vás týká pouze jedna ze dvou následujících částí prezentace.

Úvodní přednáška předmětu Prezenční forma studia

Témata přednášek

1. Organizační informace k předmětu Algoritmy I
2. Úvod
 - 2.1 Co je to algoritmus
 - 2.2 Základy algoritmického řešení problémů
 - 2.3 Důležité typy problémů
 - 2.4 Fundamentální datové struktury
3. Analýza složitosti algoritmů
 - 3.1 Základy analýzy složitosti algoritmů
 - 3.2 Asymptotické notace složitosti
 - 3.3 Analýza nerekurzivních algoritmů
 - 3.4 Analýza rekurzivních algoritmů
4. Strategie řešení problémů hrubou silou a úplným prohledáváním

Témata přednášek (pokrač.)

- 4.1 Třídění výběrem a bublinové třídění
- 4.2 Sekvenční vyhledávání
- 4.3 Vyhledávání podřetězce hrubou silou
- 4.4 Problém nejbližší dvojice bodů
- 4.5 Konvexní obal množiny
- 4.6 Úplné prohledávání – problém obchodního cestujícího a problém batohu
- 4.7 Průchod grafem do šířky
- 4.8 Průchod grafem do hloubky
5. Strategie řešení problémů a výhled
 - 5.1 Třídění vkládáním
 - 5.2 Topologické třídění
 - 5.3 Generování permutací a podmnožin
 - 5.4 Vyhledávání půlením intervalu

Témata přednášek (pokrač.)

- 5.5 Hledání mediánu
- 5.6 Interpoláční vyhledávání
- 5.7 Vyhledávání a vkládání do binárního vyhledávacího stromu
- 6. Strategie řešení rozděl a panuj
 - 6.1 Třídění sléváním
 - 6.2 QuickSort
 - 6.3 Průchody binárním stromem
 - 6.4 Násobení velkých celých čísel a násobení matic
 - 6.5 Problém nejbližší dvojice bodů
 - 6.6 Konvexní obal množiny

Výše uvedená **témata přednášek** budou pochopitelně rozdělena do jednotlivých přednášek na celý semestr.

31/776

Cvičení

- Přímá výuka ve cvičeních odpovídá přednáškám.
- Ve cvičeních pracují studenti pod vedením cvičícího na konkrétní implementaci příkladů v jazyce C++.
- Dále je také možné konzultovat s cvičícím probírané učivo.
- Rozdělení do cvičení tak, jak je uvedeno v informačním systému Edison, je nutné respektovat.
- Není možné překračovat kapacitu cvičení.
- Veškeré přesuny je nutné mít zaznamenány v systému Edison.

32/776

Cvičení (pokrač.)

- **Cvičení nenahrazuje přednášku!**
 - Účelem cvičení není příprava na závěrečnou písemku.
 - Cvičení nejsou bleskovou přednáškou pro ty, kteří nechodí na přednášky.
 - Na cvičení je nutné být připraven.

33/776

Úkoly

- Hodnocení v předmětu Algoritmy I se skládá ze tří částí, úkolů:
 1. **průběžné aktivity na cvičeních,**
 2. **obhajoby projektu a**
 3. **závěrečné písemné práce.**
- Všechny úkoly jsou povinné.
- Z každého úkolu je nutné získat aspoň minimální počet bodů.

34/776

Úkoly – průběžná aktivita na cvičeních

- Tato část hodnocení probíhá **průběžně po celý semestr**.
- Na každém cvičení bude cvičícím ohodnocena Vaše aktivita. Aktivita je hodnocena pomocí barevného kódu:
 - **zelená** – student na cvičení pracoval aktivně, v látce se orientoval, dařilo se mu implementovat zadané úkoly,
 - **oranžová** – student na cvičení byl spíše pasivní, na cvičení nebyl příliš připraven (ve znalostech měl „mezery“), implementace úkolů se příliš nedařila a
 - **červená** – student na cvičení byl zcela pasivní, o výuku nejevil zájem, implementaci úkolů nezvládl. Do této kategorie spadá i neomluvená neúčast na cvičení.

35/776

Úkoly – průběžná aktivita na cvičeních (pokrač.)

- Každému barevnému kódu odpovídá určitá váha, která se projeví v celkovém hodnocení všech cvičení. Zelená aktivita má váhu 1, oranžová má váhu 0,5 a červená 0.
- Z takto získaných vah z jednotlivých cvičení se na konci semestru vypočte průměrná váha, která se vynásobí maximálním možným počtem bodů (30) a výsledek je Vámi získaný počet bodů.
- Je zřejmé, že všechny zelené kódy odpovídají maximálnímu počtu bodů (30), samé červené kódy odpovídají nulovému počtu bodů.
- Body za aktivitu nelze získat zpětně.

36/776

Úkoly – průběžná aktivita na cvičeních (pokrač.)

Příklad

Student Franta na pěti cvičeních získal zelené hodnocení, na třech oranžové a na dvou červené. Průměrnou váhu vypočtete jako:

$$\frac{5 \times 1 + 3 \times 0,5 + 2 \times 0}{5 + 3 + 2} = \frac{6,5}{10} = 0,65.$$

Výsledné bodové hodnocení je tedy $0,65 \times 30 = 19,5$ bodů.

37/776

Úkoly – obhajoba projektu

- Zadání projektu bude zveřejněno na webu předmětu začátkem dubna 2024.
- Deadline pro odevzdání je **12. května 2024 23:59**.
- Způsob odevzdání stanoví jednotliví cvičící.
- Obhajoby projektů proběhnou v zápočtovém týdnu a ve zkuškovém období. Harmonogram obhajob je v kompetenci cvičících.
- Bez ohledu na to, kdy proběhnou obhajoby projektů platí, že se obhajuje verze, která byla odevzdána do deadlinu.
- Na obhajobu projektu **není možný** opravný termín.

38/776

Úkoly – závěrečná písemná práce

- Závěrečná písemná práce se bude psát ve zkuškovém období.
- Všechny termíny budou vypsány v systému Edison.
- Opravný termín na závěrečnou písemnou práci je poskytován jen těm studentům, kteří na první pokus získali aspoň 10 bodů.

Počet bodů na prvním termínu	Opravný termín
0 až 9	NE
10 až 20	ANO
více než 21	není nutný, úspěch

39/776

Úkoly – závěrečná písemná práce (pokrač.)

- Závěrečnou písemnou práci máte možnost psát celkem **dvakrát**, jinak řečeno máte nárok na **jednu opravu**. Předmět je ukončen klasifikovaným zápočtem. Nevztahuje se tudíž na něj požadavek dvou oprav, jak to vyžaduje studijní řád u zkoušky.
- Žádné další opravy nejsou možné.

40/776

Hodnocení úkolů

- Je nutné splnit **všechny výše uvedené úkoly**,
- a zároveň u všech úkolů **aspoň minimální počet bodů**.

Úkol	Počet bodů	
	minimum	maximum
Průb. aktivita na cvičeních	15	30
Obhajoba projektu	15	30
Písemná práce	21	40
Celkový počet bodů	51	100

41/776

Úvodní přednáška předmětu

Kombinovaná forma studia

1. tutoriál – 23. února 2024 **povinný**

- Na tomto úvodním tutoriálu Vám budou sděleny informace o organizaci studia předmětu a informace o náplni předmětu.
- Konzultace k tématům: Algoritmus. Strategie řešení problémů pomocí algoritmů. Významné typy řešených problémů.

2. tutoriál – 8. března 2024

- Konzultace k tématům: Analýza složitosti algoritmů.

42/776

3. tutoriál – 22. března 2024

- Konzultace k tématům: Strategie řešení problémů hrubou silou. Třídění výběrem, bublinové třídění. Sekvenční vyhledávání. Konvexní obal množiny bodů. Nalezení nejbližší dvojice bodů.

4. tutoriál – 12. dubna 2024

- Konzultace k tématům: Strategie řešení úplným prohledáváním. Problém obchodního cestujícího. Problém batohu. Průchody grafem.

43/776

5. tutoriál – 26. dubna 2024

- Konzultace k tématům: Strategie řešení sníží a vyřeš. Třídění vkládáním. Generování permutací a podmnožin. Vyhledávání půlením intervalu. Nalezení mediánu. Interpoláční vyhledávání. Vyhledávání a vkládání do binárního vyhledávacího stromu.

6. tutoriál – 11. května 2024

- Konzultace k tématům: Strategie řešení rozděl a panuj. QuickSort. MergeSort. Konvexní obal množiny bodů. Nalezení nejbližší dvojice bodů.

44/776

- Hodnocení v předmětu Algoritmy I se skládá ze tří částí, úkolů:
 1. **průběžné aktivity na tutoriálech,**
 2. **obhajoby projektu a**
 3. **závěrečné písemné práce.**
- Všechny úkoly jsou povinné.
- Z každého úkolu je nutné získat aspoň minimální počet bodů.
- Další informace o jednotlivých úkolech budou k dispozici na webu tutora.

45/776

Úkoly – průběžná aktivita na tutoriálech

Průběžná aktivita na tutoriálech znamená:

- účast na tutoriálech a
- průběžné plnění úkolů zadaných na jednotlivých tutoriálech.

46/776

Úkoly – obhajoba projektu

- Zadání projektu bude zveřejněno na webu předmětu začátkem dubna 2024.
- Deadline pro odevzdání je **12. května 2024 ve 23:59**.
- Způsob odevzdání a další náležitosti budou upřesněny na webu tutora.
- Obhajoby projektů proběhnou ve zkuškovém období. Termíny budou vypsány v systému Edison.
- Bez ohledu na to, kdy proběhnou obhajoby projektů platí, že se obhajuje verze, která byla odevzdána do deadlinu.
- Na obhajobu projektu **není možný** opravný termín.

47/776

Úkoly – závěrečná písemná práce

- Závěrečná písemná práce je zaměřena na teoretické znalosti.
- Závěrečné písemná práce proběhne ve zkuškovém období.
- Termíny budou vypsány v systému Edison.
- Opravný termín na závěrečnou písemnou práci je poskytován jen těm studentům, kteří na první pokus získali aspoň 10 bodů.

Počet bodů na prvním termínu	Opravný termín
0 až 9	NE
10 až 20	ANO
více než 21	není nutný, úspěch

48/776

Úkoly – závěrečná písemná práce (pokrač.)

- Závěrečnou písemnou práci máte možnost psát celkem **dvakrát**, jinak řečeno máte nárok na **jednu opravu**. Předmět je ukončen klasifikovaným zápočtem. Nevztahuje se tudíž na něj požadavek dvou oprav, jak to vyžaduje studijní řád u zkoušky.
- Žádné další opravy nejsou možné.

49/776

- Je nutné splnit **všechny výše uvedené úkoly**,
- a zároveň u všech úkolů **aspoň minimální počet bodů**.

Úkol	Počet bodů	
	minimum	maximum
Průb. aktivita na tutoriálech	15	30
Obhajoba projektu	15	30
Písemná práce	21	40
Celkový počet bodů	51	100

Úvodní přednáška předmětu Software pro výuku

Primární software

- Vývojové prostředí pro C++
- Dokumentace k C++

Doplňkový software

- Dokumentační systém Doxygen, www.doxygen.org
- Typografický systém \LaTeX , www.ctan.org

- Na učebnách je pro výuku k dispozici Microsoft Visual Studio Community 2022.
- Toto vývojové prostředí doporučuji i pro domácí přípravu.
- Obecně lze použít jakékoliv vývojové prostředí s kompilátorem podporujícím minimálně specifikaci **C++17**.

Poznámky

1. Při hodnocení Vašich projektů bude používán překladač **Microsoft Visual C++** a specifikace jazyka **C++17**.
2. Pozor na nestandardní rozšíření jazyka C++ implementovaného v GNU C++ kompilátoru.
3. Jazyk C není totožný s jazykem C++!

53/776

Úvodní přednáška předmětu Studijní literatura

Studijní literatura

Studijní literaturu lze rozdělit do dvou skupin:

- **povinná literatura** – strategie algoritmického řešení problémů a
- **doporučená literatura** – programovací jazyk C++.

Níže uvedenou literaturu využijete v předmětu Algoritmy I i Algoritmy II.

54/776

Povinná literatura

1. LEVITIN, Anany. *Introduction to the Design and Analysis of Algorithms*. 3rd ed. Boston: Pearson, 2012. ISBN 978-0-13-231681-1.
2. CORMEN, Thomas H., Charles Eric LEISERSON, Ronald L. RIVEST a Clifford STEIN, [2022]. *Introduction to algorithms*. Fourth edition. Cambridge, Massachusetts: The MIT Press. ISBN 978-026-2046-305.
3. SEDGEWICK, Robert. *Algoritmy v C*. Praha: SoftPress, 2003. ISBN 80-864-9756-9.

55/776

4. MAREŠ, Martin a Tomáš VALLA, 2017. *Průvodce labyrintem algoritmů* [online]. Praha: CZ.NIC, z.s.p.o. [cit. 2020-10-03]. CZ.NIC. ISBN 978-80-88168-19-5. Dostupné z: <https://knihy.nic.cz/>
5. WRÓBLEWSKI, Piotr. *Algoritmy*. Brno: Computer Press, 2015. ISBN 978-80-251-4126-7.
6. WIRTH, N. *Algoritmy a struktury údajov*. Alfa, Bratislava 1989.

1. STROUSTRUP, Bjarne. *C++ programovací jazyk*. Praha: Softwarové Aplikace a Systémy, 1997. ISBN 80-901-5072-1.
2. VIRIUS, Miroslav. *Pasti a propasti jazyka C++*. 2., aktualiz. a rozš. vyd. Brno: CP Books, 2005. ISBN 80-251-0509-1.
3. SCHILDT, Herbert. *Nauč se sám C++: [poznej, vyzkoušej, použij]*. Praha: SoftPress, 2001. ISBN 80-864-9713-5.
4. ECKEL, Bruce. *Myslíme v jazyku C++*. Praha: Grada, 2000. Knihovna programátora (Grada). ISBN 80-247-9009-2.

Děkuji za pozornost

Co je to algoritmus. Strategie řešení problémů pomocí algoritmů. Významné typy řešených problémů.

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



Co je to algoritmus. Strategie řešení problémů pomocí algoritmů. Významné typy řešených problémů.

Co je to algoritmus?

Proč studovat algoritmy?

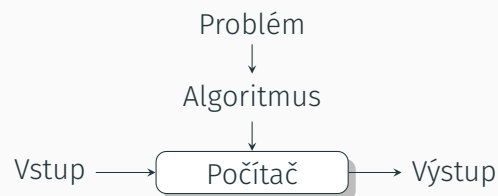
- Profesionální vývojář/informatik by měl znát standardní algoritmy pro řešení základních problémů, měl by umět navrhnout nové algoritmy a analyzovat efektivitu algoritmů.
- Algoritmy vedou k rozvoji analytického myšlení – jde o nalezení přesného a formálního postupu jak problém řešit.
- Jde o obecně použitelný mentální nástroj – **člověk problému dostatečně nerozumí do té doby, dokud ho nedokáže vysvětlit někomu jinému, neřkuli vysvětlit ho počítači.**
- Schopnost formalizovat řešení vede k daleko hlubšímu pochopení problematiky, než kdybychom se jednoduše pokusili řešit problém řekněme ad-hoc způsobem.

59/776

Co je to algoritmus?

Algoritmus

Algoritmus chápeme jako konečnou posloupnost jednoznačných instrukcí vedoucích k řešení problému, tj. vedoucích k získání požadovaného výstupu pro libovolný správný vstup v konečném čase.



60/776

Co je to algoritmus? (pokrač.)

- Předchozí popis pojmu algoritmus **není definicí** v matematickém slova smyslu.
- Předpokládáme, že existuje něco nebo někdo kdo je schopen porozumět „jednoznačným instrukcím“ a je schopen se jimi řídit.
- Pro korektní definici bychom museli nejdříve jasně definovat, **co** je to ta jednoznačná instrukce.
- Formální definice algoritmu **neexistuje!**

61/776

Co je to algoritmus? (pokrač.)

Poznámky

- Automatický předpoklad – algoritmus bude vykonávat elektronický počítač, computer.
- Překlad slova **computer**:
 1. dnes – počítač
 2. dříve – **počtář**, člověk zapojený do numerických výpočtů.
- Přestože budeme dále předpokládat, že algoritmy budeme implementovat na elektronickém počítači, pojem algoritmu samotný na elektronických počítačích nezávisí.

62/776

Ukázka algoritmu

- Tři algoritmy pro řešení téhož problému – nalezení největšího společného dělitele dvou celých čísel.
- Demonstrace několika důležitých skutečností:
 - dodržení požadavku jednoznačnosti instrukcí,
 - rozsah vstupních hodnot je nutné přesně specifikovat,
 - ten samý algoritmus můžeme reprezentovat několika různými způsoby,
 - pro řešení jednoho problému může existovat více algoritmů a
 - algoritmy řešící stejný problém mohou být založeny na zcela rozdílných myšlenkách, principech a mohou se výrazně lišit v rychlosti řešení daného problému.

63/776

Největší společný dělitel (NSD, GCD)

- Mějme dvě nezáporná celá čísla m a n , z nichž aspoň jedno je navíc různé od nuly.
- Největší společný dělitel $\gcd(m, n)$ definujeme jako největší celé číslo, které dělí obě čísla m a n beze zbytku.
- Algoritmus pro jeho nalezení popsal v knize „Základy“ Euklides z Alexandrie zhruba ve třetím století před naším letopočtem.

64/776

Největší společný dělitel – Euklidův algoritmus

Algoritmus je založen na opakovaném aplikování vztahu

$$\gcd(m, n) = \gcd(n, m \bmod n), \quad (1)$$

dokud zbytek $m \bmod n$ není roven 0.

Protože $\gcd(m, 0) = m$, je poslední hodnota m rovna hledanému největšímu společnému děliteli.

65/776

Příklad

$$\begin{aligned}\gcd(60, 24) &= \gcd(24, 12) = \gcd(12, 0) = 12 \\ \gcd(24, 60) &= \gcd(60, 24) = \gcd(24, 12) = \gcd(12, 0) = 12 \\ \gcd(7, 3) &= \gcd(3, 1) = \gcd(1, 1) = \gcd(1, 0) = 1 \\ \gcd(3, 7) &= \gcd(7, 3) = \gcd(3, 1) = \gcd(1, 1) = \gcd(1, 0) = 1 \\ \gcd(13, 0) &= 13 \\ \gcd(0, 13) &= \gcd(13, 0) = 13\end{aligned}$$

66/776

Strukturovanější forma zápisu – jednotlivé kroky výpočtu:

- Krok 1** Jestliže $n = 0$ potom vrať hodnotu m jako výsledek a skonči; jinak pokračuj Krokem 2.
- Krok 2** Vyděl číslo m číslem n , zbytek po dělení přiřaď do r .
- Krok 3** Přiřaď hodnotu čísla n do m , hodnotu čísla r do n . Pokračuj Krokem 1.

67/776

Zápis pomocí pseudokódu:

Vstup : Dvě nezáporná celá čísla m a n , z nichž aspoň jedno je nenulové

Výstup: Největší společný dělitel čísel m a n , $\gcd(m, n)$

```
1 while  $n \neq 0$  do
2    $r \leftarrow m \bmod n$ ;
3    $m \leftarrow n$ ;
4    $n \leftarrow r$ ;
5 end
6 return  $m$ ;
```

68/776

- Algoritmus založen přímo na definici NSD – NSD dělí obě zadaná čísla m a n beze zbytku.
- NSD nemůže být větší než menší ze zadaných čísel, takže můžeme psát $t = \min(m, n)$.
- Pokud t dělí obě čísla m a n beze zbytku, pak $\gcd(m, n) = t$, jinak číslo t snížíme o 1 a postup opakujeme.
- Kdy se algoritmus zastaví?

Příklad

Pro $m = 60$ a $n = 24$ je $t = \min(60, 24) = 24$.

Algoritmus nejprve vyzkouší $t = 24$, pak $t = 23$ a tak dále až se nakonec zastaví na čísle $t = 12$.

69/776

Největší společný dělitel – algoritmus postupným dělením (pokrač.)

Strukturovanější forma zápisu – jednotlivé kroky výpočtu:

- Krok 1** Přiřaď do t hodnotu $\min(m, n)$.
- Krok 2** Vyděl číslo m číslem t . Jestliže zbytek po dělení je roven 0, pokračuj Krokem 3; jinak pokračuj Krokem 4.
- Krok 3** Vyděl číslo n číslem t . Jestliže je zbytek po dělení roven 0, vrať číslo t jako výsledek a skonči; jinak pokračuj Krokem 4.
- Krok 4** Sniž hodnotu čísla t o 1 a pokračuj Krokem 2.

70/776

Největší společný dělitel – algoritmus postupným dělením (pokrač.)

Chyba v algoritmu

- Algoritmus v této podobě nefunguje správně, pokud je jedno z čísel m a n rovno 0. Číslo t by mělo hodnotu 0 a došlo by k dělení nulou.
- Požadavky na hodnoty vstupující do algoritmu je nutno pečlivě specifikovat!

71/776

Největší společný dělitel – algoritmus rozkladem na prvočinitele

- Krok 1** Proved' rozklad čísla m na prvočinitele.
- Krok 2** Proved' rozklad čísla n na prvočinitele.
- Krok 3** Najdi všechny společné prvočinitele v rozkladech získaných v Kroku 1 a Kroku 2.
Počet výskytů společného prvočinitele p je roven

$$\min(p_m, p_n),$$

kde p_m resp. p_n je počet výskytů p v rozkladu čísla m resp. n ,

- Krok 4** Spočítej součin všech společných prvočinitelů a tento součin vrať jako výsledek.

72/776

Největší společný dělitel – algoritmus rozkladem na prvočinitele (pokrač.)

Příklad

Pro $m = 60$ a $n = 24$ bude průběh algoritmu následovný:

$$\begin{aligned} 60 &= 2^2 \cdot 3^1 \cdot 5^1 \\ 24 &= 2^3 \cdot 3^1 \\ \gcd(60, 24) &= 2^2 \cdot 3^1 \\ &= 12 \end{aligned}$$

73/776

Největší společný dělitel – algoritmus rozkladem na prvočinitele (pokrač.)

Problémy

- Popsaný algoritmus je výpočetně mnohem náročnější než Euklidův algoritmus.
- Nalezení NSD pomocí rozkladu na prvočinitele **není algoritmus** – rozklad čísla není „jednoznačná instrukce“.
- Pro rozklad na prvočinitele je totiž nezbytný seznam prvočísel.
- Krok 3 také není zřejmý – jak nalézt společné prvky v prvočíselném rozkladu? Jak nalézt společné prvky ve dvou setříděných seznamech čísel?

74/776

Eratosthenovo síto

- Řešení problému nalezení všech prvočísel menších nebo rovných číslu n , kde $n > 1$.
- Původ v Řecku, cca 200 let před naším letopočtem.
- Nejprve si vytvoříme seznam všech přirozených čísel od 2 do n .
- Postupně bereme čísla, které zůstávají v seznamu a vylučujeme jeho násobky.
- Tímto způsobem pokračujeme dokud, nelze ze seznamu vyloučit žádné další číslo.
- Čísla, která zůstala v seznamu jsou hledaná prvočísla.

75/776

Eratosthenovo síto (pokrač.)

Příklad

Pro $n = 25$ dostáváme

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	
2	3	5	7	9	11	13	15	17	19	21	23	25												
2	3	5	7		11	13		17	19		23	25												
2	3	5	7		11	13		17	19		23													

76/776

Eratosthenovo síto (pokrač.)

Zastavení algoritmu

- V příkladu jsme jako poslední vylučovali násobky čísla 5.
- Jaké bude, pro dané n , největší číslo p jehož násobky budeme ze seznamu vylučovat?
- První násobek bude $p \cdot p$, tj. p^2 .
- Všechny nižší násobky $2p, 3p, \dots, (p-1)p$ již byly eliminovány jako násobky jiných čísel: $2p$ jako násobek 2, $3p$ jako násobek 3 a tak dále.
- Dále je zřejmé, že $p^2 \leq n$ a tudíž $p = \lfloor \sqrt{n} \rfloor$, kde $\lfloor x \rfloor$ značí nejbližší menší přirozené číslo k číslu x .

77/776

Eratosthenovo síto (pokrač.)

Vstup : Přirozené číslo $n > 1$

Výstup : Pole L obsahující všechna prvočísla $\leq n$

```
1 for  $p \leftarrow 2$  to  $n$  do
2   |  $A[p] \leftarrow p$ ;
3 end
4 for  $p \leftarrow 2$  to  $\lfloor \sqrt{n} \rfloor$  do
5   | if  $A[p] \neq 0$  then           //  $p$  nebylo dosud vyloučeno
6     |  $j \leftarrow p^2$ ;
7     | while  $j \leq n$  do
8       |  $A[j] \leftarrow 0$ ;
9       |  $j \leftarrow j + p$ ;
10    | end
11   | end
12 end
```

78/776

Eratosthenovo síto (pokrač.)

```
13 // Čísla, která nebyla z pole  $A$  vyloučena, okopírujeme
    do pole  $L$ 
```

```
14  $i \leftarrow 0$ ;
15 for  $p \leftarrow 2$  to  $n$  do
16   | if  $A[p] \neq 0$  then
17     |  $L[i] \leftarrow A[p]$ ;
18     |  $i \leftarrow i + 1$ ;
19   | end
20 end
```

79/776

Největší společný dělitel – algoritmus rozkladem na prvočinitele

- Začleněním Eratosthenova síta dostáváme pro výpočet největšího společného dělitele pomocí prvočíselného rozkladu regulární algoritmus.
- Zbývá vyřešit problém, kdy jedno nebo obě čísla, pro než počítáme největšího společného dělitele, je rovno 1...

80/776

Co je to algoritmus. Strategie řešení problémů pomocí algoritmů. Významné typy řešených problémů.

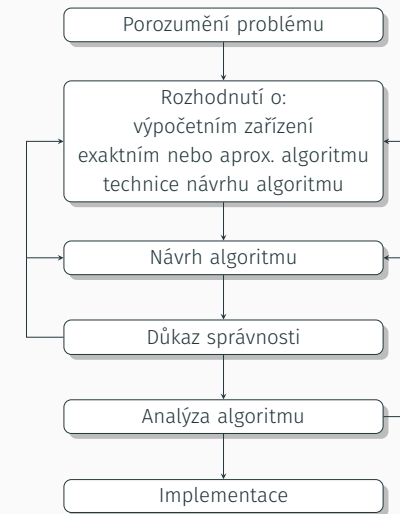
Základy algoritmického řešení problémů

Základy algoritmického řešení problémů

- Algoritmy považujeme za **procedurální, konstruktivní** způsob řešení daného problému.
- Algoritmy nejsou řešením problému samy o sobě, ale jsou návodem jak řešení získat.
- Informatika vs. matematika – neexistence „nekonečně malého ϵ “, „limity pro n jdoucího k nekonečnu“.
- Podobnost informatiky a starořeckého pojetí geometrie – řešení pomocí „pravítka a kružítka“, konečný počet kroků.

81/776

Proces návrhu a analýzy algoritmu



82/776

Porozumění problému

- Na první pohled banalita – nesprávné porozumění se může vymstít \Rightarrow nutnost algoritmus přepracovat.
- Řešení ukázkových případů, speciální případy řešení.
- Vstupní data definují **instanci problému**. Definice přípustných vstupních dat.
- **Správný algoritmus** musí **pracovat správně pro všechna** přípustná vstupní data, ne jen pro **většinu**.
- Znalost odborné literatury výhodou – typické problémy a jejich typická řešení.
- **Není** tedy vždy nutné „vynalézat znovu kolo“.
- Pro výběr vhodného algoritmu je dobré znát jejich silné i slabé stránky.

83/776

Výpočetní zařízení

- Výpočetní zařízení – počítač nemusí být jen „notebook“.
- Paralelní výpočetní zařízení – vícejádrové procesory, akcelerátory CUDA, paralelní superpočítače.
- Dosud převažuje **von Neumannova architektura** (John von Neumann 1946).
- V dalším výkladu se budeme zabývat **sekvenčními algoritmy** na von Neumannově architektuře.
- **Random Access Machine (RAM)** – teoretický model von Neumannovy architektury počítače.

84/776

- Pro návrh algoritmu a zkoumání jeho efektivity je vhodné využít RAM – HW a SW nezávislost.
- Praktická implementace – je nutné brát v úvahu HW a SW omezení konkrétního počítače.
- Předpoklad dostatečného výkonu použitého počítače. Počítačový „pravěk“.

Varování

„The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times;

premature optimization is the root of all evil

(or at least most of it) in programming.“

Donald Knuth, The Art of Computer Programming

- **Exaktní algoritmus** – poskytuje přesné řešení.
- **Aproximační algoritmus** – poskytuje přibližné řešení.
Využití aproximačních algoritmů:
 1. Existují důležité problémy, které neumíme přesně řešit, např. aerodynamické a hydrodynamické problémy.
 2. Exaktní algoritmy jsou z podstaty problému nepřijatelně pomalé. Pomalost je způsobena obrovským počtem možných řešení, ne nekvalitním algoritmem nebo implementací.
 3. Aproximační algoritmus je součástí sofistikovaného exaktního algoritmu.

Poznámka

Pokud nepotřebujeme striktně oddělit algoritmus pro přesné a pro přibližné řešení problému, přívlastek exaktní obvykle vynecháváme.

- Máme pohromadě vše potřebné: pochopili jsme zadaný problém, zvolili výpočetní zařízení a rozhodli zda použijeme exaktní nebo aproximační algoritmus.
- Jak postupovat při návrhu algoritmu? Jakou použít techniku návrhu algoritmu?

Definice

Technika návrhu algoritmu (strategie návrhu algoritmu či paradigma návrhu algoritmu) je obecný přístup k algoritmickému řešení problémů, který je aplikovatelný na množství problémů z různých oblastí informatiky.

Přínosnost technik návrhu algoritmů

1. poskytují návod, jak navrhovat algoritmy pro nové problémy, pro které není znám uspokojivý algoritmus, a
2. umožňují přehledně klasifikovat nejrůznější algoritmy podle jejich základní myšlenky.

89/776

Mějme však na paměti, že

- návrh konkrétního algoritmu pro řešení konkrétního problému může být velice náročným úkolem,
- ne všechny techniky návrhu algoritmu lze aplikovat na konkrétní problém, někdy je nutné techniky kombinovat,
- může být obtížné rozpoznat na jaké technice návrhu je algoritmus založen,
- i pokud je technika jasná, sestavení algoritmu často vyžaduje netriviální úsilí a vynalézavost, avšak
- s přibývajícím vývojářovou praxí se vše stává snazší a snazší, ale zřídka snadné.

90/776

Význam datových struktur

- vhodná datová struktura má zásadní význam pro navrhovaný algoritmus – Eratosthenovo síto versus spojový seznam
- některé z technik návrhu algoritmů silně závisí na struktuře nebo na restrukturalizaci dat určujících instanci řešeného problému,
- Niklaus Wirth: „Algorithms + Data Structures = Programs“

91/776

Přirozený jazyk

- nemusí jít o písemný záznam – ústně formulovaná myšlenka
- možné nejednoznačnosti – extrémní případ „Ženu holí stroj.“
- schopnost precizně formulovat myšlenky, formulovat je logicky správně, definovat pojmy popisujících problém, pojmy zařazovat do myšlenkového schématu atd.

92/776

Pseudokód

- směs přirozeného jazyka a konstrukcí podobných programovacím jazykům.
- obvykle přesnější a stručnější než přirozený jazyk
- stručnější zápis navrhovaného algoritmu
- existuje množství navzájem podobných „dialektů“ pseudokódu

93/776

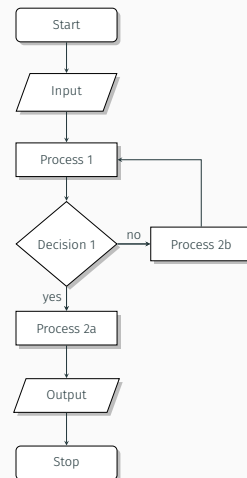
Programovací jazyk

- další možný způsob zápisu
- tento zápis považován spíše za implementaci

94/776

Vývojový diagram

- angl. flowchart
- grafická forma zápisu algoritmu
- dnes již nepoužíváno



95/776

Definice

Algoritmus považujeme za **správný**, pokud pro každý správný vstup poskytne v konečném čase správný výsledek. Pro nesprávný vstup není chování správného algoritmu definováno.

- Obvyklou metodou důkazu je matematická indukce.
- Důkaz správnosti vs. nesprávnosti algoritmu
 - Pro korektní důkaz správnosti algoritmu nestačí prokázat správnost pro **některou** instanci problému, správnost musíme umět prokázat pro **všechny** instance problému, a naopak

96/776

Důkaz správnosti algoritmu (pokrač.)

- jako důkaz nesprávnosti algoritmu stačí najít **jedinou** instanci problému, abychom mohli algoritmus prohlásit za chybný.
- Správnost aproximačního algoritmu – chyba výsledku algoritmu nepřesáhne předem definovanou mez.

97/776

Analýza algoritmu – zkoumané vlastnosti

Správnost – již vyřešeno

Časová složitost (angl. **time complexity**)

- „jak rychle algoritmus pracuje“
- rychlost neměříme časovými jednotkami, ale množstvím provedených instrukcí algoritmu (stejný algoritmus na rychlejším a pomalejším HW)

Prostorová složitost (angl. **space complexity**)

- „jak mnoho paměti algoritmus potřebuje“
- měříme v bytech a násobcích

98/776

Analýza algoritmu – zkoumané vlastnosti (pokrač.)

Jednoduchost (angl. **simplicity**)

- nelze exaktně definovat, na rozdíl od složitosti,
- spíše jde o subjektivní záležitost – krása, elegance (NSD Euklidovým algoritmem vs. rozklad na prvočinitele),
- jednodušší algoritmus – snazší pochopit, implementovat, patrně i menší množství chyb,
- jednodušší algoritmus – nemusí mít nutně nižší složitost,
- použití – typicky prototyp SW. Pokud nevyhovuje – přechod na algoritmus s nižší složitostí. Ale! „Předčasná optimalizace...“

99/776

Analýza algoritmu – zkoumané vlastnosti (pokrač.)

- terminologie – opak jednoduchosti není „složitost“ algoritmu, ale komplikovanost, nesrozumitelnost, nevhodnost návrhu.

Obecnost

1. obecnost navrženého algoritmického řešení – řešit problém velice obecně a nebo brát v úvahu možná zjednodušení v konkrétním případě?
 - řešení obecnějšího problému je lehčí než konkrétního – např. nesoudělnost dvou čísel, řešení přes NSD, NSD je obecnější problém

100/776

Analýza algoritmu – zkoumané vlastnosti (pokrač.)

- řešení obecnějšího a konkrétního problému na stejné úrovni – např. hledání mediánu, řešení přes třídění (obecnější) i konkrétní algoritmus
 - řešení obecnějšího je výrazně náročnější – např. kvadratická rovnice $ax^2 + bx + c = 0$ versus obecná algebraická rovnice n -tého stupně.
2. obecnost instance problému – návrh algoritmu by měl zpracovat všechny rozumně očekávatelné, přirozené instance problému
- U NSD není přirozené vyloučit číslo 1, ale
 - u kvadratické rovnice většinou předpokládáme, že a , b a c jsou reálná čísla – obecněji lze brát i čísla komplexní.

101/776

Analýza algoritmu – zkoumané vlastnosti (pokrač.)

Pokud nejsme spokojeni se složitostí nebo jednoduchostí návrhu či obecností algoritmu?

Nezbývá nic jiného než se vrátit na začátek, sednout si za stůl, vzít do ruky tužku a papír a přemýšlet, kreslit, hledat v literatuře a tak dále.

„konstruktor ví, kdy dosáhl dokonalosti. Ne v okamžiku, kdy již není co přidat, ale v okamžiku, kdy již není co odebrat.“

Antoine de Saint-Exupéry

„Keep It Simple, Stupid!“

Kelly Johnson

102/776

Kódování algoritmu

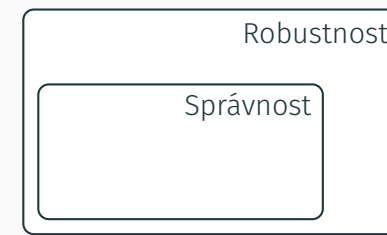
- Opět podceňovaná fáze – „Algoritmus máme vymyšlený, tak teď to jenom přepíšeme do počítače a máme hotovo.“
- Algoritmus implementujeme buď nesprávně nebo neefektivně nebo dokonce nastanou obě možnosti najednou.
- V praktickém životě – správnost programů je ověřována testováním.
- Testování programů je „umělecké řemeslo“.
- Další kritické místo – vstup dat.
 - Škola – vstupní data definují korektní instanci řešeného problému.
 - Praxe – otázku kontroly vstupních dat je nutno řešit.

103/776

Robustnost algoritmu

Definice

Algoritmus považujeme za **robustní**, pokud je správný a pro každý nesprávný vstup vydá hlášení o chybě a je schopen se z chyby zotavit.



104/776

Efektivita implementace

- Správnost implementace algoritmu je nezbytnost.
- Ale i správnou implementaci lze provést neefektivně, výkon počítače není využit tak, jak by mohl být.
- Optimalizace kódu:
 1. manuální – výpočet invariantu cyklu, nahrazení společných podvýrazů proměnnou.
 2. automatická – algoritmy optimalizace zabudované do kompilátorů, např. přidělování registrů.
- Optimalizací kódu lze zlepšit efektivitu programu o nějaký konstantní faktor, např. 10%.
- Pro radikální, řádové, zlepšení je nutné implementovat algoritmus s nižší složitostí.

105/776

Efektivita implementace (pokrač.)

- Hledání stále lepšího a lepšího algoritmu zajímavé mentální dobrodružství...
- Otázkou je kdy přestat. Dokonalost je drahý luxus. Inženýrský přístup – zdroje alokované pro projekt.
- Akademická otázka **optimality algoritmu**: „Jaká je nejmenší možná složitost jakéhokoliv algoritmu, který vyřeší daný problém?“
- Například sekvenční algoritmus pro třídění pole s n prvky – minimálně $n \log_2 n$ porovnání.
- Lze každý problém řešit algoritmem? Nerozhodnutelné problémy – **nelze** je řešit jakýmkoliv algoritmem.

106/776

Efektivita implementace (pokrač.)

- Naštěstí většinu problémů z praktického života lze algoritmicky řešit.
Dobrý algoritmus je výsledkem opakovaného úsilí a několikanásobného přepracování.

107/776

Co je to algoritmus. Strategie řešení problémů pomocí algoritmů. Významné typy řešených problémů.

Důležité typy problémů

Důležité typy problémů

- Třídění
- Vyhledávání
- Zpracování řetězců
- Grafové úlohy
- Kombinatorické úlohy
- Geometrické úlohy
- Numerické úlohy

108/776

Třídění

- Třídění v informatice – přeuspořádání prvků do neklesající posloupnosti. Srovnej s tříděním odpadu.
- Mezi prvky musí být definována **relace uspořádání** čili vztah „menší nebo rovno“, \leq .
- V praxi třídíme čísla, řetězce či strukturované záznamy.
- U záznamu musíme definovat **klíč** tj. část záznamu pole které třídíme pro kterou je definováno uspořádání. Klíč nemusí být definován explicitně, např. u čísel je jím číslo samo.

109/776

Uspořádání

Definice

Mějme binární homogenní relaci $\rho \subseteq A \times A$ na množině A .

- Relaci ρ nazýváme (neostré) **částečné uspořádání**, jestliže je současně reflexivní, antisymetrická a tranzitivní.
- Relaci ρ nazýváme (neostré) **úplné uspořádání**, jestliže je současně reflexivní, antisymetrická, tranzitivní a úplná.
- Relaci ρ nazýváme (částečné) **ostré uspořádání**, jestliže je současně asymetrická (a tedy i antisymetrická a ireflexivní) a tranzitivní.
- Relaci ρ nazýváme **úplné ostré uspořádání**, jestliže je současně asymetrická (a tedy i antisymetrická a ireflexivní), tranzitivní a souvislá.

110/776

Uspořádání – poznámky

- Neostré uspořádní standardně značíme \leq , ostré uspořádání pak $<$.
- Místo označení **částečné uspořádání** používáme někdy jen **uspořádání**.
- Místo termínu **úplné uspořádání** se používá i termín **totální** či **lineární** uspořádání.
- Je-li \leq uspořádání na množině A , pak relačnímu systému (A, \leq) říkáme **uspořádaná množina** (angl. **poset** – partially ordered set). Úplně uspořádané množině říkáme **řetězec** (angl. **chain**).

111/776

Uspořádání – poznámky (pokrač.)

- Dva různé prvky x, y jsou **porovnatelné** v uspořádání \leq , jestliže platí $(x \leq y) \vee (y \leq x)$. V opačném případě jsou prvky **neporovnatelné**. V úplném uspořádání jsou každé dva prvky porovnatelné.
- Průnik uspořádání je opět uspořádáním. Sjednocení uspořádání nemusí být obecně uspořádáním.
- Vztah mezi ostrým a neostrým uspořádáním je možno zapsat takto: " \leq " = " $<$ " \cup " $=$ ", tj. přidáním identické relace („rovnosti“) k ostrému uspořádání.

112/776

Použité vlastnosti binárních homogenních relací

Použité vlastnosti relace $\rho \subseteq A \times A \forall x, y, z \in A$:

- reflexivita: $x\rho x$,
- ireflexivita: $\neg(x\rho x)$,
- asymetrie: $x\rho y \Rightarrow y\not\rho x$,
- antisymetrie: $x\rho y \wedge y\rho x \Rightarrow x = y$,
- tranzitivita: $x\rho y \wedge y\rho z \Rightarrow x\rho z$,
- souvislost: $[x \neq y \Rightarrow x\rho y \vee y\rho x]$,
- úplnost: $x\rho y \vee y\rho x$.

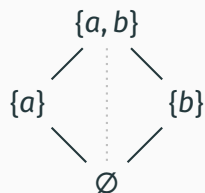
113/776

Hasseův diagram

Relaci uspořádání standardně znázorňujeme pomocí **Hasseova diagramu**, který

- reprezentuje relaci bezprostředního předcházení bez tranzitivních hran, která je stejná pro ostré i neostré uspořádání a který
- odpovídá orientovanému grafu, kde jsou všechny hrany orientovány zdola nahoru.

Příklad



Hasseův diagram pro relaci uspořádání „být podmnožinou“ na množině $\{a, b\}$. Tranzitivní hrana, která se běžně nezobrazuje, je zobrazena tečkovaně.

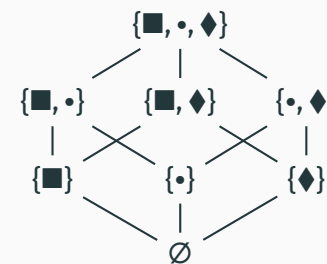
114/776

Částečné uspořádání – příklad

Pro libovolnou množinu A můžeme definovat na množině jejích podmnožin $P(A)$ uspořádání \leq inkluzí: $X \leq Y$ pokud $X \subseteq Y$, kde $X, Y \in P(A)$.

Takto definované uspořádání není úplné ale pouze částečné, protože v něm existují neporovnatelné prvky.

Příklad



V $A = \{\blacksquare, \bullet, \blacklozenge\}$ jsou neporovnatelnými prvky

- všechny jednoprvkové podmnožiny mezi sebou a
- všechny dvouprvkové podmnožiny mezi sebou.

115/776

Úplné uspořádání – příklad



- Obvyklá relace $<$ na množině přirozených, celých, racionálních a reálných číslech je úplné uspořádání.
- Abecední, lexikografické, uspořádání řetězců je také úplné uspořádání.
- Správně seskládané matřičky jsou úplně uspořádané pomocí relace „být uvnitř“. Ale pouze za předpokladu, že do žádné bábušky se nesmí vejít více menších vedle sebe – v tom případě dostáváme pouze částečné uspořádání.

116/776

Třídění – využití

- Setříděný seznam hodnot je požadovaným výstupem – výsledková listina závodu, výsledky vyhledávání na internetu.
- Pro některé úlohy se řeší lépe pro setříděný vstup – typicky **vyhledávání**. Telefonní seznam. Geometrické úlohy. Kompresí dat. Hladové algoritmy.

117/776

Definice třídícího problému

- Předpokládejme posloupnost prvků $A = a_1, a_2, \dots, a_n$. Úkolem třídění je nalézt permutaci $\pi : \mathbb{N} \rightarrow \mathbb{N}$ takovou, že $a_{\pi_i} \leq a_{\pi_{i+1}}$ pro všechna $1 \leq i < n$.
- Permutaci π nelze nalézt přímo, permutací n prvků je $n!$.
- Třídící algoritmy budeme chápat jako algoritmy, které postupně konstruuji permutaci π , například porovnáváním a výměnou prvků.

118/776

Definice třídícího problému – příklad

Mějme posloupnost $A = \mathbf{ebfcda}$ a obvyklé abecední uspořádání písmen. Hledaná permutace je

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 2 & 4 & 5 & 1 & 3 \end{pmatrix}$$

Potom

$$\begin{aligned} a_{\pi_1} &< a_{\pi_2} < a_{\pi_3} < a_{\pi_4} < a_{\pi_5} < a_{\pi_6} \\ a_6 &< a_2 < a_4 < a_5 < a_1 < a_3 \\ a &< b < c < d < e < f \end{aligned}$$

119/776

Třídění

- Třídících algoritmů byla vyvinuta celá řada. Neexistuje jeden univerzální algoritmus pro všechny situace.
 - jednoduché a pomalé vs. komplexní a rychlé,
 - náhodná vs. téměř setříděná posloupnost na vstupu
 - vnitřní paměť vs. vnější paměť.
- Máme-li dáno n prvků, minimální počet porovnání je $n \log_2 n$ pro sériové algoritmy založené na porovnání a výměně.

120/776

Třídění (pokrač.)

- **Stabilní třídění** – zachovává vzájemné polohy prvků. Jestliže máme ve tříděné posloupnosti dva prvky se shodným klíčem na pozici i a j , kde $i < j$, pak po setřídění budou tyto prvky na pozicích i' a j' , kde $i' < j'$.



Nápověda: sledujte vzájemné polohy oranžových a zelených čísel. Algoritmy třídící pomocí výměn na velkou vzdálenost jsou obvykle rychlejší, ale zase nejsou stabilní.

121/776

Třídění (pokrač.)

- Třídění **in-situ** – třídící algoritmus si vystačí jen se pamětí pro uložení prvků plus přídavná paměť konstantního rozsahu tj. tato paměť nezávisí na počtu tříděným prvků, typicky proměnné pro průchod cyklem, logické příznaky atd.
- **Přirozené** třídění – složitost třídícího algoritmu roste s mírou neseříděnosti vstupních dat.

122/776

Míra neseříděnosti posloupnosti dat

- Cílem je najít měřítko neseříděnosti, „rozházenosti“, posloupnosti n prvků, které máme třidit.
- Setříděné posloupnosti by měla odpovídat **nulová** neseříděnost.
- Posloupnosti setříděné v opačném pořadí by měla odpovídat **maximální** neseříděnost.
- Ostatní posloupnosti by se měly pohybovat mezi těmito krajními možnostmi

123/776

Míra nestříděnosti permutace

- Permutace čísel $1 \dots n$.
- Identická permutace – nulová nesetříděnost

$$\pi_{id} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$$

- Obrácená permutace – maximální nesetříděnost

$$\pi_{rev} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

- Nesetříděnost permutace budeme měřit **počtem inverzí** dané permutace.

124/776

Inverze v permutaci

Definice

Mějme permutaci $\pi : \mathbb{N} \rightarrow \mathbb{N}$. Inverze v permutaci π je dvojice prvků i, j taková, že $i < j$ a zároveň $\pi_i > \pi_j$.

Inverzi v permutaci můžeme volně interpretovat tak, že „na menším indexu je větší prvek a současně na větším indexu je menší prvek“.

Příklad

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}$$

Všechny permutace v π

$$\begin{array}{ll} 1 < 2 \wedge 4 > 1 & 1 < 4 \wedge 4 > 2 \\ 1 < 3 \wedge 4 > 3 & 3 < 4 \wedge 3 > 2 \end{array}$$

125/776

Počet inverzí v permutaci

- Identická permutace – celkový počet inverzí je nulový
- Obrácená permutace

Prvek	Inverze s prvky	Počet inverzí
n	$n-1, n-2, n-3, \dots, 1$	$n-1$
$n-1$	$n-2, n-3, \dots, 1$	$n-2$
\vdots	\vdots	\vdots
3	2, 1	2
2	1	1
1	-	0

Celkový počet inverzí je roven

$$(n-1) + (n-2) + \dots + 2 + 1 + 0 = \frac{1}{2}n(n-1)$$

126/776

Průměrný počet inverzí v permutaci

- Označme C_n celkový počet inverzí ve všech permutacích n prvků. Nejprve odvodíme vztah mezi C_n a C_{n-1} .
- Uvažujme všechny permutace $n-1$ prvků. Ke všem těmto permutacím přidáme n za poslední prvek permutace. Počet inverzí se nezvýší a bude roven C_{n-1} .
- K permutacím $n-1$ prvků přidáme n za předposlední prvek permutace. Počet inverzí se zvýší o jednu pro každou permutaci, tedy $C_{n-1} + 1 \cdot (n-1)!$
- Až nakonec k permutacím $n-1$ prvků přidáme n před první prvek permutace. Počet inverzí se zvýší o $n-1$ pro každou permutaci, tedy $C_{n-1} + (n-1)(n-1)!$

127/776

Průměrný počet inverzí v permutaci (pokrač.)

Proto

$$C_n = C_{n-1} + 0 \cdot (n-1)! + C_{n-1} + 1 \cdot (n-1)! + C_{n-1} + 2 \cdot (n-1)! + \dots + C_{n-1} + (n-1)(n-1)!$$

Odtud

$$\begin{aligned} C_n &= nC_{n-1} + [0+1+\dots+(n-1)](n-1)! \\ &= nC_{n-1} + \left[\frac{1}{2}n(n-1)\right](n-1)! \\ &= nC_{n-1} + \frac{1}{2}(n-1)n! \end{aligned}$$

128/776

Průměrný počet inverzí v permutaci (pokrač.)

Průměrný počet inverzí I_n je roven

$$I_n = \frac{C_n}{n!}.$$

Odtud dosadíme $C_n = n!I_n$ resp. $C_{n-1} = (n-1)!I_{n-1}$ a dostáváme

$$\begin{aligned} n!I_n &= n(n-1)!I_{n-1} + \frac{1}{2}(n-1)n! \\ &= n!I_{n-1} + \frac{1}{2}(n-1)n! \end{aligned}$$

Po vykrácení $n!$ dostáváme

$$I_n = I_{n-1} + \frac{1}{2}(n-1)$$

129/776

Průměrný počet inverzí v permutaci (pokrač.)

Rozepíšeme vztah pro I_n

$$\begin{aligned} I_n &= I_{n-2} + \frac{1}{2}(n-2) + \frac{1}{2}(n-1) \\ &= I_{n-3} + \frac{1}{2}(n-3) + \frac{1}{2}(n-2) + \frac{1}{2}(n-1) \\ &\vdots \\ &= I_{n-i} + \frac{1}{2}(n-i) + \dots + \frac{1}{2}(n-2) + \frac{1}{2}(n-1) \end{aligned}$$

Dále víme, že jednoprvková permutace nemůže mít inverzi, tudíž $I_1 = 0$.

130/776

Průměrný počet inverzí v permutaci (pokrač.)

Nyní hledáme takové i , aby výraz $n-i$ v indexu I_{n-i} byl roven 1. Zřejmě $i = n-1$ a proto

$$\begin{aligned} I_n &= I_{n-(n-1)} + \frac{1}{2}[n-(n-1)] + \frac{1}{2}[n-(n-2)] + \dots + \frac{1}{2}(n-2) + \frac{1}{2}(n-1) \\ &= I_1 + \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2 + \dots + \frac{1}{2}(n-2) + \frac{1}{2}(n-1) \\ &= I_1 + \frac{1}{2}[1+2+\dots+(n-2)+(n-1)] \\ &= I_1 + \frac{1}{2}\left[\frac{1}{2}n(n-1)\right] \\ &= I_1 + \frac{1}{4}n(n-1) \end{aligned}$$

131/776

Průměrný počet inverzí v permutaci (pokrač.)

A protože $I_1 = 0$ dostáváme konečně

$$I_n = \frac{1}{4}n(n-1)$$

Shrnutí – počet inverzí v permutaci n prvků

Minimum	0
Průměr	$\frac{1}{4}n(n-1)$
Maximum	$\frac{1}{2}n(n-1)$

132/776

Vyhledávání

- Základní úloha – nalezení prvku a v dané množině M , či multimnožině.
- Matematicky – platí $a \in M$ resp. $a \notin M$?
- Matematika neřeší složitost této operace.
- Algoritmů pro vyhledávání existuje celá řada – sekvenční, půlením intervalu, hašování...
- Neexistuje optimální algoritmus pro všechny situace, algoritmy mají různé předpoklady – více paměti pro rychlejší práci, setříděné pole...
- Důležité aspekty:
 - vzájemný poměr operací vyhledávání, vkládání a mazání prvku z množiny – převažuje vyhledávání nebo je poměr vyrovnaný?
 - organizace velmi velkých dat.

133/776

Zpracování řetězců

- Řetězec – posloupnost znaků z dané abecedy.
- Typické příklady řetězců:
 - textové řetězce, abeceda složená z písmen, číslic a interpunkce,
 - bitové řetězce složené z 0 a 1 nebo
 - genové řetězce složené ze znaků **A**, **C**, **G** a **T**
- Využití
 - zpracování textů,
 - komprese dat,
 - programovací jazyky a kompilátory nebo
 - vyhledávání v řetězcích (pattern matching) – hledání jednoho řetězce, vzorku, či vzorků v jiném řetězci. Triviální příklad – **Ctrl+F** v textovém editoru.

134/776

Vyhledávání v textu – Pattern Matching

- Při **vyhledávání v textu** zjišťujeme, zda daný **vzorek/vzorky** (pattern) odpovídá, shoduje se, s částí v daného **textu**. Můžeme také říci, že hledáme **výskyty vzorku v textu**.
- Využití:
 - v textových editorech (pohyb v editovaném textu, záměna řetězců),
 - v utilitách typu **grep**, které umožní najít všechny výskyty zadaných vzorků v množině textových souborů,
 - vyhledávání na webu,
 - při zkoumání DNA,
 - při analýze obrazu, zvuku apod.

135/776

Vyhledávání v textu – klasifikace vyhledávacích algoritmů

		Předzpracování textu	
		ne	ano
Předzpracování vzorku	ne	vyhledávání hrubou silou	indexové metody, typicky webové vyhledávače, obecně jsou to tzv. Information Retrieval Systems
	ano	pokročilé vyhledávací algoritmy	signaturové vyhledávací metody

136/776

Vyhledávání v textu – další kritéria rozdělení

Počet hledaných vzorků – jeden, konečný počet nebo nekonečný počet vzorků

Počet výskytů – první výskyt, všechny výskyty

Způsob porovnávání – přesné vyhledávání versus přibližné vyhledávání, kdy jsou povoleny odchylky mezi vzorkem a textem např. jeden znak se může lišit

Směr vyhledávání – v textu obvykle postupujeme od nižších indexů k vyšším, „zleva doprava“

- sousměrné algoritmy – vzorek je procházen stejným směrem
- protisměrné algoritmy – vzorek je procházen opačným směrem.

137/776

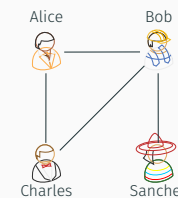
Vyhledávání v textu – označení

V dalším textu budeme používat následující označení:

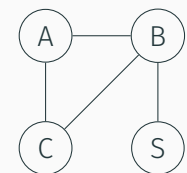
- p hledaný vzorek, $p = p_0p_1 \dots p_{m-1}$, kde $|p| = m$ je délka vzorku,
- t prohledávaný text, $t = t_0t_1 \dots t_{n-1}$, kde $|t| = n$ je délka vzorku,
- Σ – abeceda z níž je sestaven vzorek i text,
- σ – velikost abecedy Σ ($\sigma = |\Sigma|$),
- \bar{C}_n – očekávaný počet porovnání potřebných k vyhledání vzorku v textu délky n .

138/776

Grafové úlohy



Převzato z [1]



- **Graf** – neformálně množina bodů, **vrcholů**, z nichž některé jsou propojeny úsečkami, **hranami**.
- Využití – reprezentace dopravních sítí, projektového řízení, sociální sítě, elektrické sítě atd.
- Základní úlohy:
 - průchod grafem – lze se dostat do všech vrcholů grafu
 - nejkratší cesta – nejkratší cesta mezi dvěma městy

139/776

Grafové úlohy (pokrač.)

- topologické třídění – organizace projektu, činnosti musí na sebe navazovat, lze něco dělat souběžně?
- **Problém obchodního cestujícího** (traveling salesman problem, TSP) – úkolem je najít nejkratší cestu mezi n městy, přičemž každé lze navštívit právě jednou. Logistika, výroba mikročipů.
- **Problém barvení grafu** – úkolem je najít nejmenší počet barev vrcholů tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu. Plánování – události odpovídají vrcholů, hrany spojují události, které nelze vykonávat současně, řešení problému barvení grafu poskytuje optimální rozvrh.

140/776

Kombinatorické úlohy

- Podstata úloh – nalézt **permutaci, kombinaci** či **podmnožinu** z dané množiny objektů, která splňuje určitá **omezení** a případně mají nějakou další vlastnost např. minimalizace resp. maximalizace nějaké funkce.
- Problém obchodního cestujícího – pořadí navštívených měst je permutace, minimalizovaná funkce je celková vzdálenost.
- Patrně nejsložitější problémy informatiky z teoretického i praktického pohledu:
 - počet možných kandidátů řešení (např. permutací) roste velice rychle a dosahuje obrovských hodnot už pro středně velké problémy

141/776

Kombinatorické úlohy (pokrač.)

- není znám algoritmus pro nalezení přesného řešení v přijatelném čase a
- dokonce se neví jestli takový algoritmus existuje, předpokládá se že ne.
- Některé kombinatorické problémy ale lze řešit efektivně např. hledání nejkratší cesty

142/776

Geometrické úlohy

- Zpracovávají body, úsečky, mnohoúhelníky a podobné objekty.
- Jsou to vlastně první algoritmy – euklidovská geometrie, konstrukce „pravítkem a kružítkem“.
- Využití:
 - počítačová grafika,
 - počítačové hry,
 - robotika,
 - medicína.
- V našem předmětu:
 - problém nejbližší dvojice bodů – množina bodů v rovině, najít dva body s minimální vzdáleností,
 - konvexní obal množiny bodů – nalézt nejmenší konvexní mnohoúhelník obsahující dané body.

143/776

Numerické úlohy

- Řešení soustav rovnic, výpočet hodnot funkcí, určitých integrálů atd.
- Většina těchto úloh vyžaduje počítání s reálnými čísly.
Typické problémy:
 - počítač umí zachytit jen omezený rozsah čísel (ne ∞) a s omezenou přesností ($\frac{1}{3}, \pi$) a
 - kumulace zaokrouhlovacích chyb.
- Vědeckotechnické výpočty – klasická aplikace prvních počítačů. Inženýrské aplikace.
- Dnes – ukládání a analýza dat, navigace, logistika....
- V našem předmětu – několik typických úloh, řešení soustavy rovnic, matice.

144/776

Co je to algoritmus. Strategie řešení problémů pomocí algoritmů. Významné typy řešených problémů.

Základní datové struktury

Základní datové struktury

- **Datovou strukturu** můžeme definovat jako způsob organizace vzájemně souvisejících dat.
- Jakou použít datovou strukturu silně závisí na řešeném problému.
- Existuje několik zvláště důležitých datových struktur:
 - lineární datové struktury – **pole, spojový seznam, zásobník, fronta, prioritní fronta**
 - **graf**
 - **strom**
 - **množina**
 - **slovník**

145/776

Pole (Array)

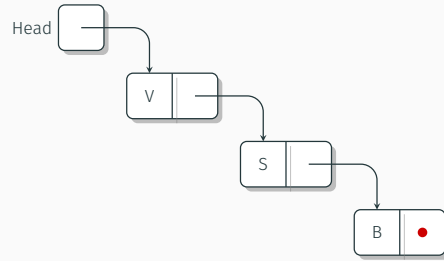
- Konečná posloupnost n hodnot uložených ve spojitém úseku paměti.
 - Přístup pomocí indexu, náhodný přístup s konstantní časovou složitostí.
 - Index:
 - nezáporné celé číslo,
 - pole s n hodnotami má rozsah indexů **vždy** $0, \dots, n - 1$
- | | | | |
|--------|--------|-----|----------|
| $a[0]$ | $a[1]$ | ... | $a[n-1]$ |
|--------|--------|-----|----------|
- Využití:
 - přímo – vektory, buffery,
 - základ pro další datové struktury – řetězce, matice atd.

146/776

Seznam (Linked list)

Charakteristika

- nejobecnější lineární datová struktura,
- operace nejsou striktně určeny,
- existuje mnoho variant.



Atributy

- atributy seznamu závisí na konkrétní implementaci,
- v nejjednodušším případě jen nutný odkaz na první prvek seznamu, tzv. **hlavu seznamu**.

147/776

Seznam (Linked list) (pokrač.)

Varianty seznamu

- **jednosměrný seznam** (singly linked list) – nejjednodušší varianta, odkaz pouze na následníka,
- **obousměrný seznam** (doubly linked list) – položka obsahuje odkaz na předchůdce i následníka,
- **kruhový seznam** (circular list) – hlava a ocas seznamu splývají.

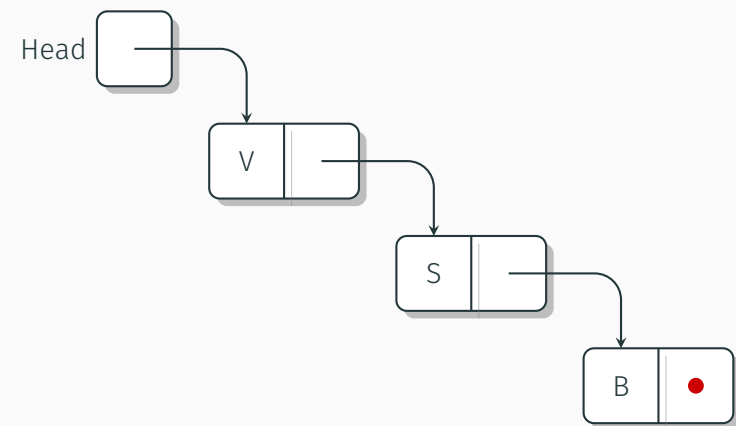
148/776

Seznam – jednosměrný seznam

- složen z položek (items),
- položky obsahují data a odkaz na další položku,
- každá položka odkazuje na následovníka,
- k položkám lze přistupovat sekvenčně,
- přímý přístup na základě indexu je složitý (průchod cyklem),
- pohyb seznamem dozadu je také problém,
- konec seznamu – speciální odkaz „nikam“, většinou pojmenován **nil**, **NULL**, **nullptr** či **Nothing**.

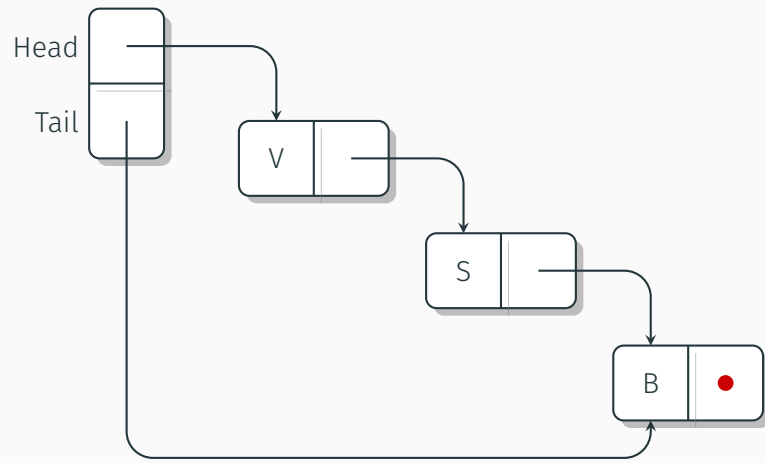
149/776

Seznam – jednosměrný, jen hlava seznamu



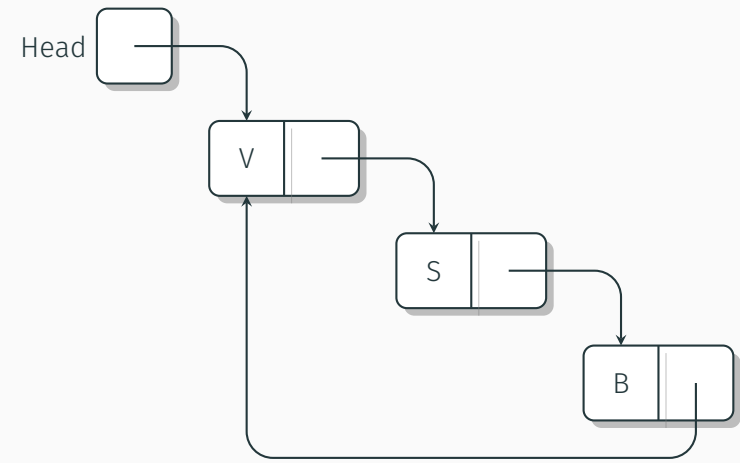
150/776

Seznam – jednosměrný, hlava i ocas seznamu



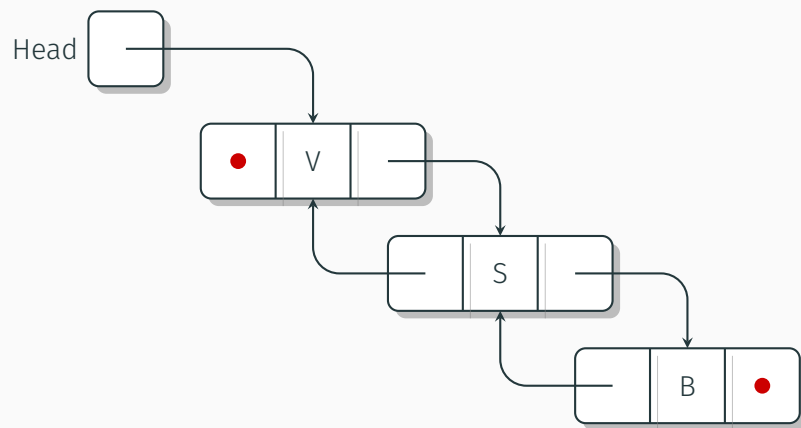
151/776

Seznam – jednosměrný, kruhový



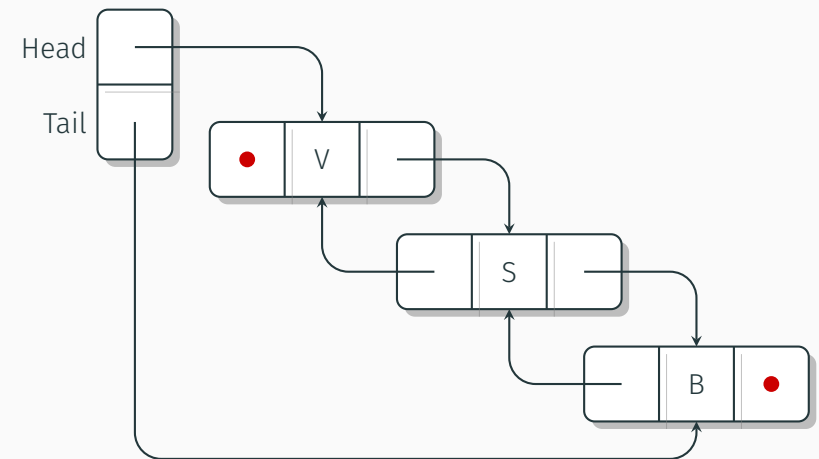
152/776

Seznam – obousměrný, jen hlava seznamu



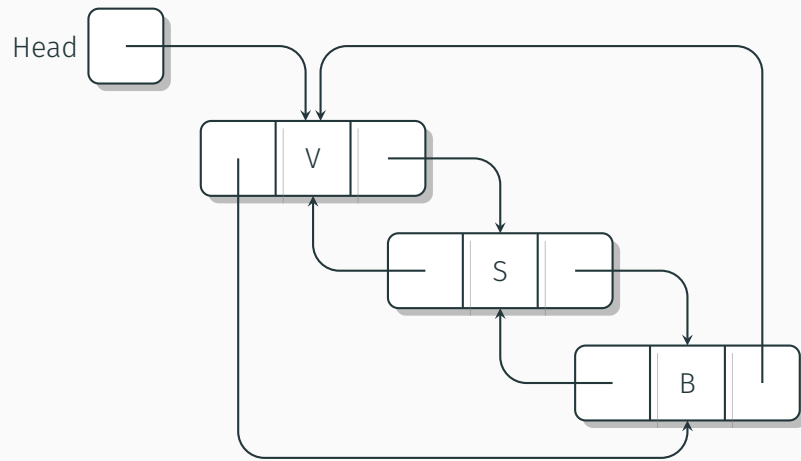
153/776

Seznam – obousměrný, hlava i ocas seznamu



154/776

Seznam – obousměrný, kruhový



155/776

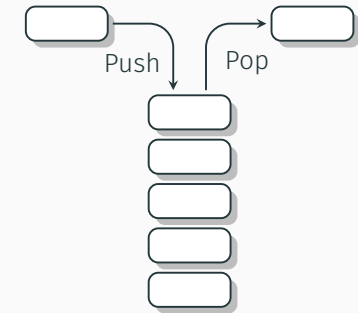
Zásobník (Stack)

Charakteristika

- princip **last-in, first-out, LIFO**
- prvek, který byl vložen poslední, je jako první ze zásobníku vyzvednut

Atributy

- prvky vkládáme na tzv. **vrchol zásobníku** (stack pointer).
- prvně vložený prvek se nazývá **dno zásobníku** (stack bottom)



156/776

Zásobník – operace

Základní operace

- Push – vložení prvku na vrchol zásobníku
- Pop – vyjmutí prvku z vrcholu zásobníku
- IsEmpty – test prázdnoty zásobníku
- Top – vrátí prvek z vrcholu zásobníku bez jeho vyjmutí

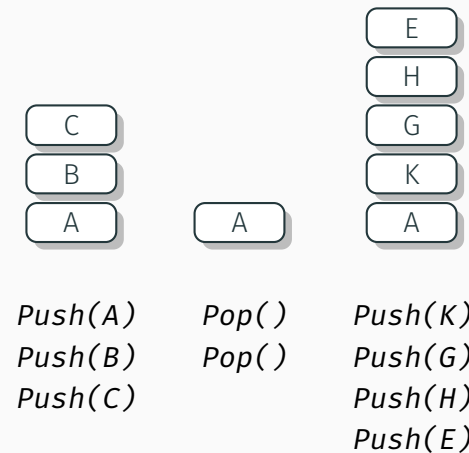
Další možné operace

- Init – inicializace zásobníku
- Clear – vyjmutí všech prvků ze zásobníku
- IsFull – test, zda je zásobník plný (pouze pro zásobník s omezenou kapacitou)

Správně implementované operace mají **konstantní** časovou složitost $O(1)$, tj. jejich časová složitost nezávisí na počtu prvků v zásobníku.

157/776

Zásobník – vizualizace



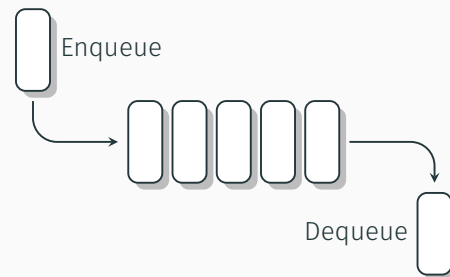
158/776

- Pokud provedeme operaci Pop na prázdném zásobníku nastává tzv. **podtečení** (stack underflow).
- Pokud není možné přidat další prvek, nastává tzv. **přetečení** (stack overflow).

- volání funkcí (metod)
- vyhodnocování aritmetických výrazů
- odstranění rekurze
- zásobníkove orientované jazyky, například PostScript, PDF
- testování parity závorek, HTML/XML značek

Charakteristika

- princip **first-in, first-out, FIFO**
- prvek, který byl vložen první, je také jako první z fronty vyzvednut



Atributy

- první prvek se nazývá **hlava** fronty (**head**),
- poslední prvek se nazývá **ocas** fronty (**tail**).

Základní operace

- Enqueue – vložení prvku na konec fronty
- Dequeue – vyjmutí prvku ze začátku fronty
- Peek – vrátí prvek ze začátku fronty bez jeho vyjmutí
- IsEmpty – test zda je fronta prázdná

Další možné operace

- Init – inicializace fronty
- Clear – vyjmutí všech prvků z fronty
- IsFull – test, zda je fronta zaplněna (pouze u fronty s omezenou kapacitou)

Správně implementované operace mají konstantní časovou složitost **O(1)**, tj. jejich časová složitost nezávisí na počtu prvků ve frontě.



Enqueue(A)
Enqueue(B)
Enqueue(C)



Dequeue()
Dequeue()



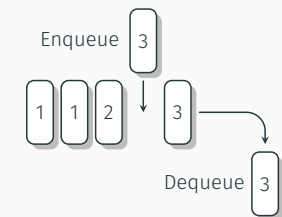
Enqueue(K)
Enqueue(G)
Enqueue(H)
Enqueue(E)

- Pokud provedeme operaci Dequeue na prázdné frontě nastává tzv. **podtečení** (queue underflow).
- Pokud není možné přidat další prvek, nastává tzv. **přetečení** (queue overflow).

- tisková fronta u sdílené tiskárny
- plánovač v operačním systému (více běžících procesů na jednoprocessorovém počítači ⇒ procesy se musí střídat)
- obsluha uživatelů na serverech obecně

Charakteristika

- řešení úlohy „Vyjmi z množiny největší prvek a zpracuj ho.“
- na rozdíl od obyčejné fronty se k prvkům váže ještě priorita,
- pro prvky se stejnou prioritou FIFO,
- prvek s vyšší prioritou předbíhá ty s nižší prioritou a odchází z fronty dříve



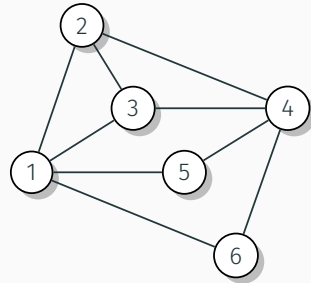
Implementace

- pomocí pole nebo setříděného pole,
- efektivněji pomocí datové struktury zvané **halda** (heap).

Neorientovaný graf

Definice

Neorientovaným grafem nazýváme dvojici $G = (V, E)$, kde V je konečná neprázdná množina **vrcholů**, E je množina jednoprvkových nebo dvouprvkových podmnožin V . Prvky množiny E se nazývají **hrany** grafu.



Příklad

$$G = (V, E)$$

$$V = \{1, 2, 3, 4, 5, 6\}$$

$$E = \{\{1, 2\}, \{1, 3\}, \{1, 5\}, \{1, 6\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}, \{4, 6\}\}$$

167/776

Hrany v neorientovaném grafu

Mějme hranu $e \in E$, kde $e = \{u, v\}$.

- O hraně e říkáme, že **spojuje** vrcholy u a v .
- Vrcholům u a v říkáme **krajní vrcholy** hrany e .
- Dále říkáme, že vrcholy u a v jsou **incidentní** (nebo, že **incidují**) s hranou e . A obdobně, že hrana e je incidentní s vrcholy u a v .
- Protože hrana e spojuje vrcholy u a v říkáme, o nich že jsou to **sousední** (sousedící) vrcholy.

Definice

Hranu spojující vrchol se sebou samým nazýváme **smyčkou**.

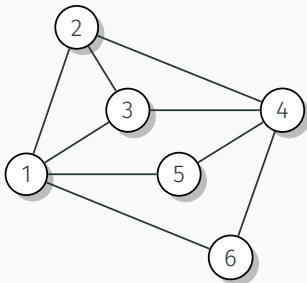
168/776

Neorientovaný graf – stupeň vrcholu

Definice

Stupněm vrcholu v neorientovaném grafu nazýváme počet hran s vrcholem incidentních, tj. $s(v) = |\{e \in E \mid v \in e\}|$.

Příklad



v	$s(v)$
1	4
2	3
3	3
4	4
5	2
6	2

169/776

Neorientovaný graf – stupeň vrcholu (pokrač.)

Věta

Součet stupňů vrcholů libovolného neorientovaného grafu $G = (V, E)$ je roven dvojnásobku počtu jeho hran.

$$\sum_{v \in V} s(v) = 2|E|$$

Důkaz.

Zřejmý (v sumě se každá hrana počítá dvakrát). □

170/776

Počet hran v neorientovaném grafu

Věta

Pro libovolný neorientovaný graf $G = (V, E)$ bez smyček platí:

$$0 \leq |E| \leq \frac{1}{2}|V|(|V| - 1)$$

Důkaz.

Maximálního počtu hran v grafu docílíme tak, že každý z $|V|$ vrcholů spojíme hranou se všemi ostatními vrcholy, kterých je $|V| - 1$. Součin $|V|(|V| - 1)$ musíme vydělit dvěma, protože jsme každou hranu započítali dvakrát. \square

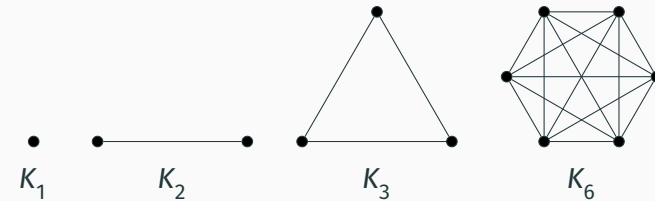
171/776

Úplný graf

Definice

Neorientovaný graf $G = (V, E)$ ve kterém pro každou dvojici vrcholů u a v existuje hrana nazýváme **úplným grafem** a označujeme je $K_{|V|}$.

Příklad



172/776

Hustý vs. řídký graf

- **Hustý graf** – „téměř“ kompletní graf, chybí jen „relativně“ malý počet hran do maximálního počtu
- **Řídký graf** – „velice malý“ počet hran, „relativně“ velký počet hran neexistuje.
- Přesná definice není, pojmy jako „téměř“, „relativně“ či „velice malý“ jsou subjektivní.
- Záleží vždy na konkrétní situaci.
- Při výběru reprezentace grafu v počítači je nutno brát zřetel na to zda je graf hustý nebo řídký. A s tím následně souvisí časová složitost implementovaných algoritmů.

173/776

Podgraf

Definice

Graf $H = (V_H, E_H)$ nazýváme podgrafem grafu $G = (V_G, E_G)$, jestliže platí následující podmínky:

1. $V_H \subseteq V_G$
2. $E_H \subseteq E_G$
3. Hrany grafu H mají oba vrcholy v H .

174/776

Podgraf (pokrač.)

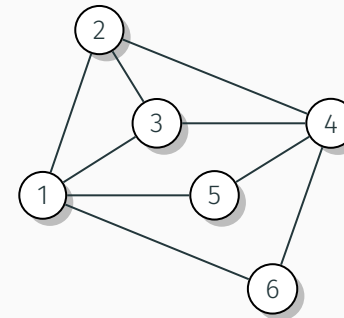
Poznámky

- Jinými slovy, podgraf vznikne vymazáním některých vrcholů původního grafu, všech hran do těchto vrcholů zasahujících a případně některých dalších hran.
- Termín podgraf se v teorii grafů používá jako jistá obdoba pojmu podmnožina.

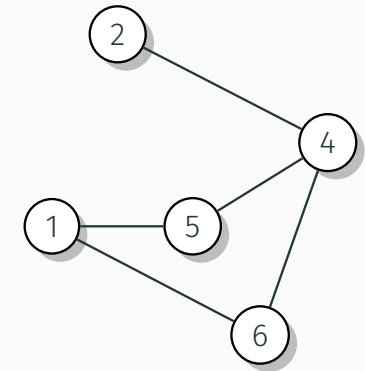
175/776

Podgraf (pokrač.)

Příklad



Graf G



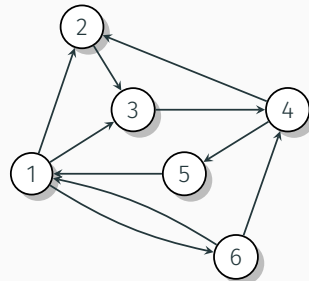
Podgraf H

176/776

Orientovaný graf

Definice

Orientovaným grafem nazýváme dvojici $G = (V, E)$, kde V je konečná neprázdná množina **vrcholů**, E je množina uspořádaných dvojic (u, v) , **hran**, z kartézského součinu $V \times V$, neboli $(u, v) \in V \times V$.



Příklad

$G = (V, E)$

$V = \{1, 2, 3, 4, 5, 6\}$

$E = \{(1, 2), (1, 3), (1, 6), (2, 3), (3, 4), (4, 2), (4, 5), (5, 1), (6, 1), (6, 4)\}$

177/776

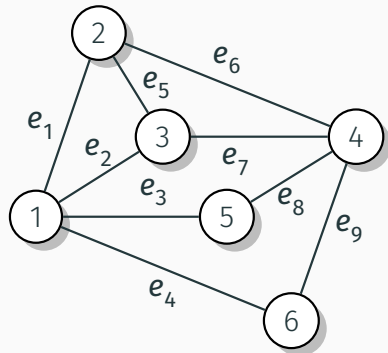
Způsoby reprezentace grafu

- grafickou formou:
 - prostě obrázkem,
 - asi nejsrozumitelnější forma pro člověka,
 - vhodně pro grafy s malým počtem vrcholů,
 - prakticky nemožnost zpracování počítačem.
- maticí,
- seznamem sousedících vrcholů.

178/776

Maticice incidence

- Počet řádků matice odpovídá počtu vrcholů, počet sloupců odpovídá počtu hran.
- Pokud je vrchol incidentní s hranou, je na dané pozici jednička, jinak nula.



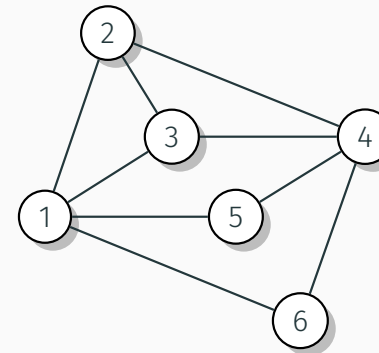
Maticice incidence

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

179/776

Maticice sousednosti

- Čtvercová matice, kde řád matice odpovídá počtu vrcholů v grafu.
- Pokud jsou dva vrcholy spojeny hranou, je na dané pozici jednička, jinak nula.

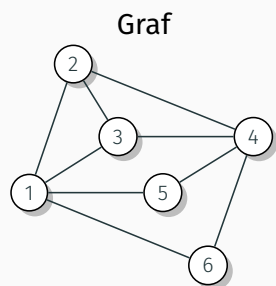


Maticice sousednosti

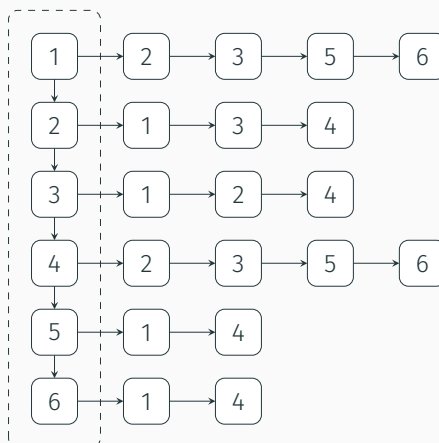
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

180/776

Seznam sousedících vrcholů



Seznam sousedících vrcholů



181/776

Hustý vs. řídký graf a jeho reprezentace

Seznam sousedících vrcholů

- ukazatele v seznamech zabírají paměť navíc,
- vhodný pro řídké grafy,
- pohodlnější změny struktury grafu (vlození smazání vrcholu, stejně tak hrany).

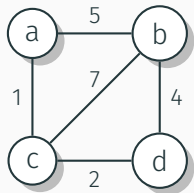
Maticová reprezentace

- vhodná pro husté grafy,
- vložení či smazání vrcholu je komplikované.

182/776

Vážené grafy

- Každé hraně je přiřazeno číslo označované jako **váha** či **cena** hrany.
- Motivace v reálném světě – délka cesty, kapacita datové linky atd.
- Vážené grafy mohou být orientované i neorientované.
- Reprezentace:
 - matice sousednosti – hodnota v matici udává váhu hrany nebo je tu speciální hodnota pro neexistující hranu, ∞
 - seznam sousedících vrcholů – v seznamu sousedů uložíme i váhu konkrétní hrany.



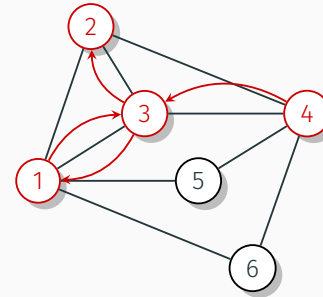
$$\begin{pmatrix} \infty & 5 & 1 & \infty \\ 5 & \infty & 7 & 4 \\ 1 & 7 & \infty & 2 \\ \infty & 4 & 2 & \infty \end{pmatrix}$$

183/776

Sled

Definice

Poslovnost navazujících vrcholů a hran $v_1, e_1, v_2, \dots, v_n, e_n, v_{n+1}$, kde $e_i = \{v_i, v_{i+1}\}$ pro $1 \leq i \leq n$ nazýváme (neorientovaným) **sledem**.



Sled

4 {4, 3} 3 {3, 1} 1 {1, 3} 3 {3, 2} 2

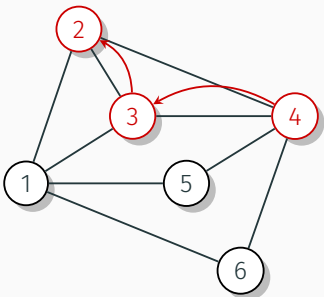
V orientovaném grafu mluvíme o orientovaném sledu.

184/776

Cesta

Definice

Sled, v němž se neopakuje žádný vrchol nazýváme **cestou**. Tedy $v_i \neq v_j, \forall 1 \leq i < j \leq n$. Číslo n pak nazýváme **délkou cesty**.



Cesta

4 3 2

Z faktu, že se v cestě neopakují vrcholy, vyplývá, že se v ní neopakují ani hrany. Každá cesta je tedy zároveň i sledem.

V orientovaném grafu mluvíme o orientované cestě.

185/776

Souvislost grafu

Definice

Graf se nazývá **souvislý**, jestliže mezi každými dvěma vrcholy existuje cesta.

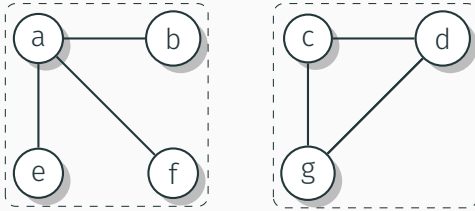
Nesouvislý graf se skládá z několika souvislých částí tzv. souvislých komponent.

Definice

Souvislá komponenta grafu je maximální souvislý podgraf daného grafu.

186/776

Souvislost grafu (pokrač.)



Věta

Nechť $G = (V, E)$ je souvislý graf. Pak platí $|E| \geq |V| - 1$.

Důkaz.

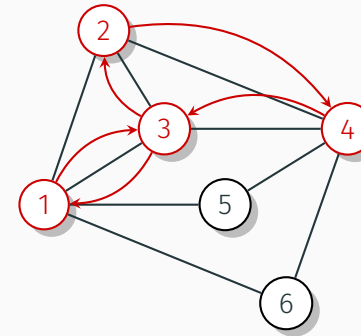
Zřejmý. □

187/776

Uzavřený sled

Definice

Sled, který má alespoň jednu hranu a jehož počáteční a koncový vrchol splývají, nazýváme **uzavřeným sledem**.



Uzavřený sled

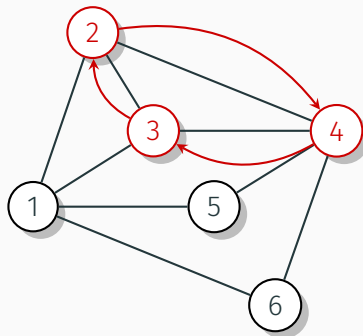
4 {4, 3} 3 {3, 1} 1 {1, 3} 3 {3, 2}
2 {2, 4} 4

188/776

Kružnice

Definice

Uzavřená cesta je uzavřený sled, v němž se neopakují vrcholy ani hrany. Uzavřená cesta se nazývá také **kružnicí**.



Kružnice

4 3 2

V definici kružnice jsme museli zakázat kromě opakování vrcholů i opakování hran proto, aby posloupnost v_1, e_1, v_2, e_2, v_1 nemohla být považována za kružnici.

189/776

Acykličnost grafu

Definice

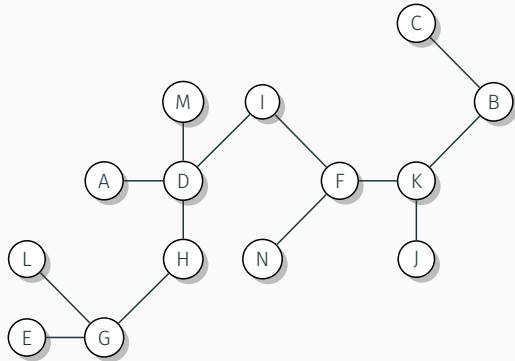
Graf se nazývá **acyklický**, jestliže neobsahuje kružnici.

190/776

Volný strom

Definice

Souvislý, acyklický, neorientovaný graf nazýváme **volným stromem** (angl. free tree).



Poznámka

Prázdný graf je možné považovat za strom, tzv. prázdný strom.

191/776

Volný strom (pokrač.)

Terminologie

- V teorii grafů se objekty, které propojujeme hranami nazývají obvykle vrcholy (angl. vertex, vertices).
- Pokud mluvíme o stromech lze pro vrchol používat i výraz **uzel** (angl. node).
- Označení vrchol a uzel je rovnocenné, jde spíše o zvyklost.

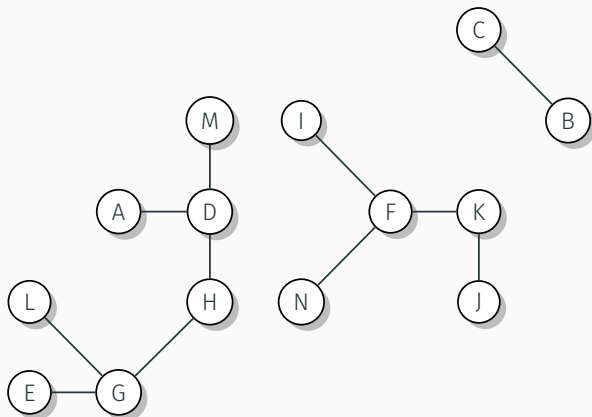
192/776

Les

Definice

Acyklický graf, který není spojitý se nazývá **les** (angl. forest).

Každá souvislá komponenta lesa je volný strom



193/776

Vlastnosti volného stromu

Věta

Nechť $G = (V, E)$ je neorientovaný graf, potom následující tvrzení jsou ekvivalentní:

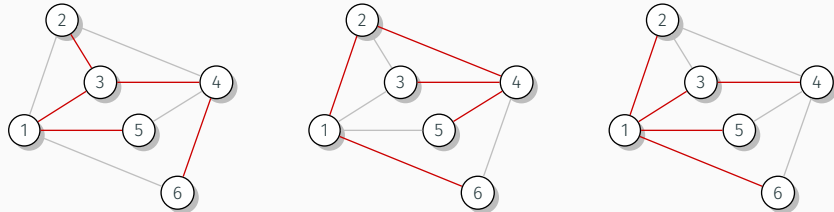
1. G je volný strom.
2. Každé dva vrcholy v G jsou spojeny právě jednou cestou.
3. G je souvislý, ale pokud odebereme libovolnou hranu, získáme nesouvislý graf.
4. G je souvislý, a $|E| = |V| - 1$.
5. G je acyklický, a $|E| = |V| - 1$.
6. G je acyklický. Přidáním jediné hrany do množiny hran E bude výsledný graf obsahovat kružnici.

194/776

Kostra grafu (angl. Spanning tree)

Definice

Kostrou souvislého grafu G nazýváme takový podgraf grafu G na množině všech jeho vrcholů, který je stromem.



Poznámky

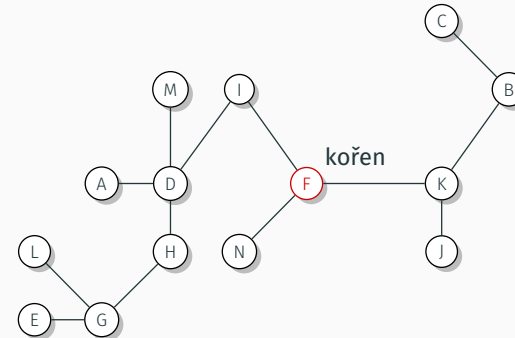
- Kostra musí obsahovat všechny vrcholy původního grafu G .
- Koster grafu může být více.

195/776

Kořenový strom

Definice

Volný strom, který obsahuje jeden odlišný vrchol, se nazývá **kořenový strom** (angl. rooted tree). Odlišný vrchol se nazývá **kořen** stromu.



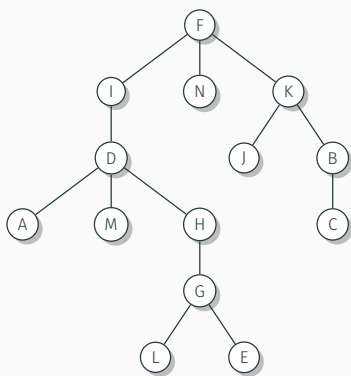
Poznámka

Někdy se používá také označení **zakořeněný strom**.

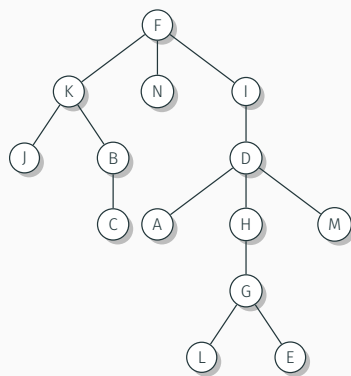
196/776

Kořenový strom – obvyklá vizualizace

Vizualizace 1



Vizualizace 2



Obě vizualizace jsou kořenový strom **rovnocenné!** Neexistuje „vlevo“ či „vpravo“.

197/776

Kořenový strom – základní pojmy

Uvažujme vrchol x v kořenovém stromu T s kořenem r .

- Libovolný vrchol y na jednoznačné cestě od kořene r do vrcholu x se nazývá **předchůdce** vrcholu x .
- Jestliže y je předchůdce x , potom x se nazývá **následovník** vrcholu y .
- Jestliže poslední hrana na cestě z kořene r do vrcholu x je hrana (y, x) , potom se vrchol y nazývá **rodič** vrcholu x a vrchol x je **potomek** vrcholu y .
- Dva vrcholy mající stejného rodiče se nazývají **sourozenci**.
- Vrchol bez potomků se nazývá vnější vrchol nebo-li **líst**.
- Nelistový vrchol se nazývá **vnitřním** vrcholem stromu.

198/776

Kořenový strom – základní pojmy (pokrač.)

Poznámky

- Každý vrchol je pochopitelně předchůdcem a následovníkem sama sebe.
- Jestliže y je předchůdce x a zároveň $x \neq y$, potom y je vlastní předchůdce vrcholu x a x je vlastní následovník vrcholu y .
- Kořen stromu je jediným vrcholem ve stromu bez rodiče.
- Vrchol je obecný pojem. Každý list a vnitřní vrchol je zároveň vrchol (bez přívlastku). Srovnej: člověk, žena, muž.

199/776

Stupeň vrcholu

Definice

Počet potomků vrcholu x v kořenovém stromu se nazývá **stupeň vrcholu x** .

Poznámky

- Metoda výpočtu stupně vrcholu se u kořenového stromu liší od výpočtu ve volném stromu.
- V kořenovém stromu se nepočítá rodič.
- Ve volném stromu pojem rodiče neexistuje, existují jen sousední vrcholy, počítají se tudíž všechny vrcholy.

200/776

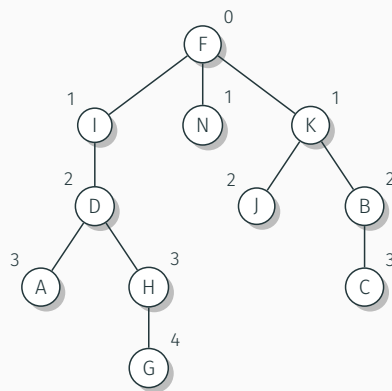
Hloubka vrcholu, výška stromu

Definice

Délka cesty od kořene stromu k vrcholu x se nazývá **hloubka vrcholu x** ve stromu T .

Definice

Největší hloubka libovolného vrcholu se nazývá **výška stromu T** .



Výška stromu je 4.

201/776

Seřazený strom

Definice

Kořenový strom ve kterém je určeno pořadí potomků se nazývá **seřazený strom** (angl. ordered tree).

Poznámky

- Tudíž, pokud vrchol má k potomků, lze určit prvního potomka, druhého potomka, až k -tého potomka.
- Pokud ale například prvního potomka zrušíme, ostatní potomci se posouvají! Druhý potomek se stane prvním, druhý třetím atd. Nelze mít mezi potomky „prázdnou pozici“.

202/776

Binární strom

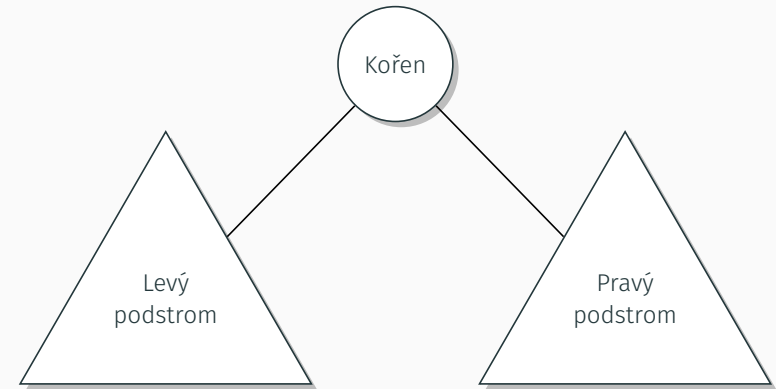
Definice

Binární strom je struktura definovaná nad konečnou množinou uzlů M , která:

- **Pravidlo 1**
neobsahuje žádný uzel, tj. $M = \emptyset$ nebo
- **Pravidlo 2**
je složena ze tří disjunktních množin uzlů L , R a $\{r\}$,
 $L \cup R \cup \{r\} = M$:
 - kořene stromu r ,
 - binárního stromu nad množinou L zvaného levý podstrom a
 - binárního stromu nad množinou R zvaného pravý podstrom.

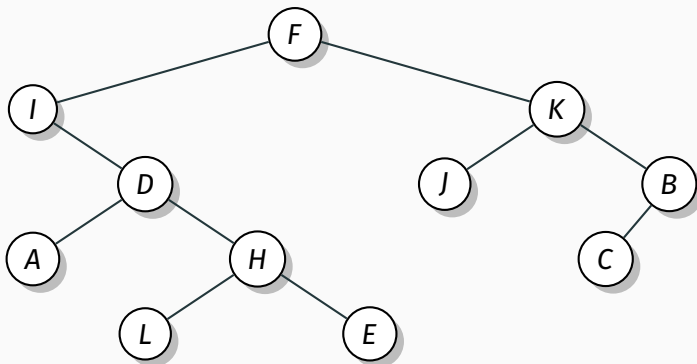
203/776

Binární strom – grafické znázornění rekurzivní definice



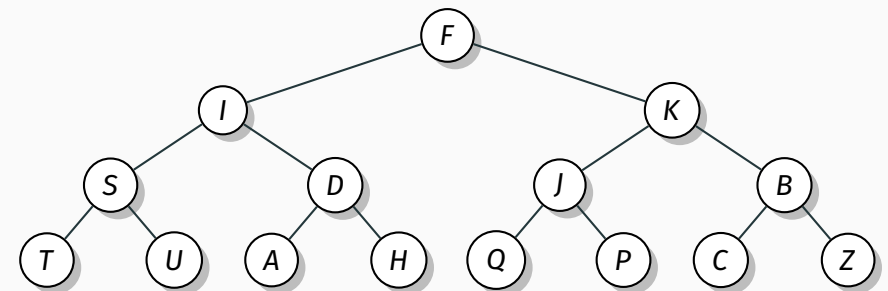
204/776

Binární strom, příklad



205/776

Úplný binární strom



Úplný binární strom – každý vnitřní uzel má právě dva potomky.

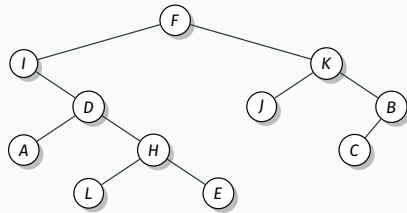
206/776

Binární vyhledávací strom

Jak využít binární stromy jako datovou strukturu? Jakým způsobem v nich organizovat data?

Libovolně? Nesmysl – jde o zbytečně komplikovaný seznam!

Řešením je využít vlastností stromu (souvislost a jedinečnost cesty z uzlu do uzlu) a doplnit je vhodným „navigačním pravidlem“.



207/776

Binární vyhledávací strom – „navigační pravidlo“

Nechť y je uzel v binárním stromu. Potom pro každý uzel x v levém podstromu uzlu y a každý uzel z v pravém podstromu uzlu y platí

$$x_{key} < y_{key} < z_{key}.$$

Binární strom, ve kterém pro všechny jeho uzly platí toto pravidlo nazýváme **binární vyhledávací strom** (angl. binary search tree).

208/776

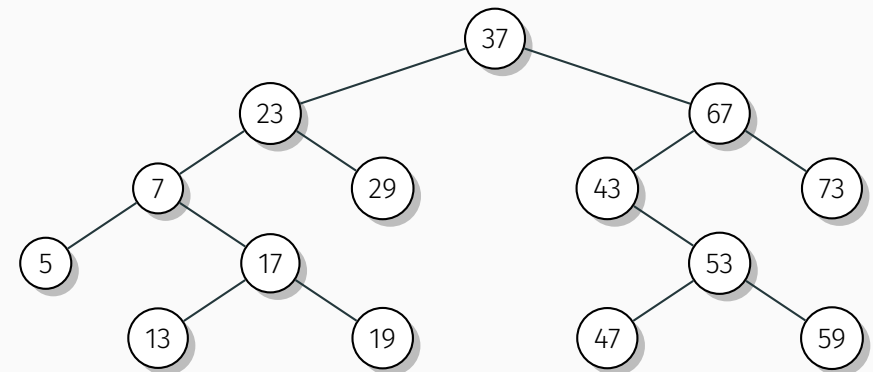
Binární vyhledávací strom – „navigační pravidlo“ (pokrač.)

Poznámky

- Navigační pravidlo tedy určuje, jak mají být data v binárním vyhledávacím stromu rozmístěna.
- Znalosti rozmístění dat ve stromu využijeme při jejich vyhledávání.
- Algoritmy pro vložení a vyjmutí ze stromu jsou navázány na algoritmus vyhledávání.
- Binární vyhledávací strom je tedy už od počátku budován s ohledem na toto pravidlo.

209/776

Binární vyhledávací strom



210/776

Binární vyhledávací strom – vyhledávání

Hledání hodnoty a zahájíme v kořeni stromu r . Potom mohou nastat tyto možnosti:

1. Strom s kořenem r je prázdný, potom tento strom nemůže obsahovat uzel s klíčem a a hledání končí neúspěchem.
2. V opačném případě srovnáme klíč a s klíčem kořene r .
V případě, že
 - 2.1 $a = r_{key}$ strom obsahuje uzel s klíčem a a hledání končí úspěšně;
 - 2.2 $a < r_{key}$ všechny uzly s klíči menšími než r_{key} jsou levém podstromu, pokračujeme rekurzivně v levém podstromu;
 - 2.3 $a > r_{key}$ všechny uzly s klíči většími než r_{key} jsou pravém podstromu, pokračujeme rekurzivně v pravém podstromu.

211/776

Binární vyhledávací strom

Efektivita mnoha algoritmů pracujících obecně s binárními stromy, např. vyhledávání v binárním vyhledávacím stromu závisí na výšce binárního stromu.

Pro výšku h binárního stromu s n uzly platí nerovnost

$$\lceil \log_2 n \rceil \leq h \leq n - 1$$

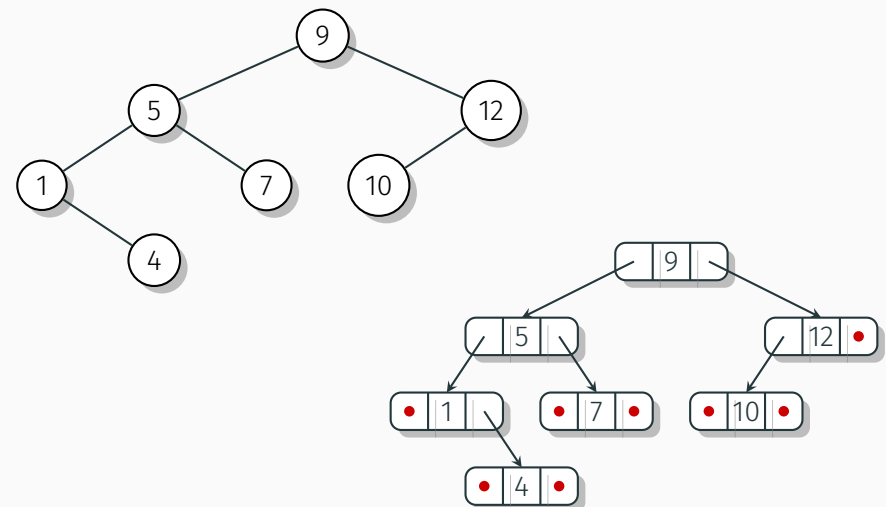
212/776

Binární vyhledávací strom – vkládání

- Vložení klíče musí korespondovat s vyhledávacím algoritmem.
- Nejdříve se musíme pokusit vkládaný klíč ve stromu najít.
- Pokud jej nenajdeme, tak místo, kde jsme hledání neúspěšně zakončili odpovídá místu ve stromu, kde by tento klíč měl být.
- Toto plyne z jednoznačnosti cesty mezi kořenem a kterýmkoliv uzlem.
- Nový uzel je připojen jako nový list ke stromu – strom roste prostřednictvím listů.
- Otázkou je co s duplicitami? Řešení závisí na povaze konkrétní řešení úlohy.

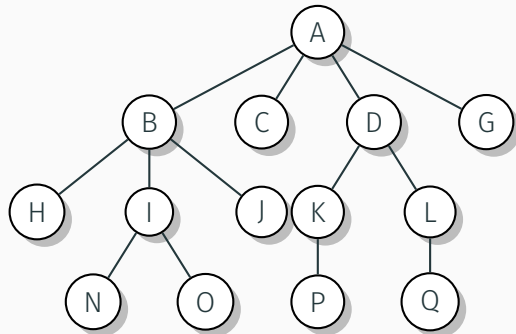
213/776

Binární strom – standardní implementace



214/776

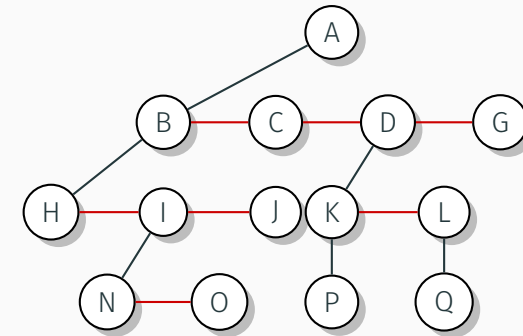
Reprezentace seřazeného stromu



- Každý uzel může mít libovolný počet potomků.
- Komplikovaná reprezentace uzlu – seznam potomků?

215/776

Reprezentace seřazeného stromu – first child – next sibling

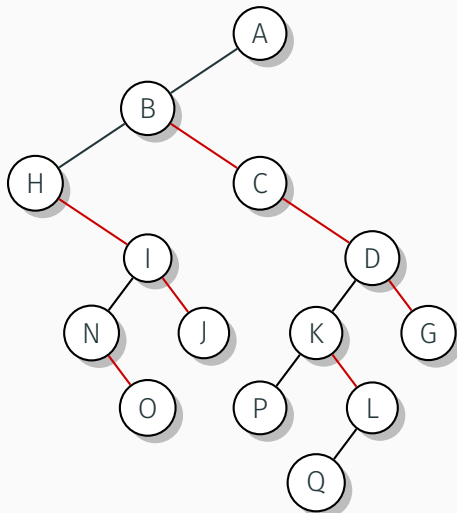


First child – next sibling reprezentace – každý uzel obsahuje dva ukazatele:

1. ukazatel na prvního potomka a
2. **ukazatel na sourozence.**

216/776

Reprezentace seřazeného stromu – Knuthova transformace



First child – next sibling reprezentace otočena o 45° po směru hodinových ručiček.

217/776

Množina

- Datovou strukturu množina budeme chápat jako nesetříděnou kolekci (i prázdnou) navzájem různých prvků.
- Množinu můžeme zadat dvěma způsoby:
 1. **výčtem prvků**, např. $M = \{2, 3, 5, 7\}$ nebo
 2. **vlastností**, které musí prvky splňovat, např. $M = \{n, \text{prvočísla menší než } 10\}$.
- Nejdůležitější množinové operace:
 - příslušnost prvku do množiny, čili dotaz „Je x prvkem M?“,
 - sjednocení dvou množin a
 - průnik dvou množin.

218/776

Bitový vektor

- univerzum $U = \{u_0, u_1, \dots, u_{n-1}\}$, $|U| = n$
- libovolnou množinu M považujeme za podmnožinu univerza U
- bitový vektor \vec{b} dimenze n , kde

$$\vec{b}_i = \begin{cases} 1 & u_i \in M \\ 0 & \text{jinak} \end{cases}$$

Příklad

$$U = \{0, 1, 2, \dots, 8, 9\}$$

$$M = \{2, 3, 5, 7\}$$

$$\vec{b} = (0, 0, 1, 1, 0, 1, 0, 1, 0, 0)$$

219/776

Výčet prvků

- množinu reprezentujeme výčtem, prvků v ní obsažených
- podle okolností můžeme pro uložení prvků využít pole, spojový seznam, binární vyhledávací strom, hašovací tabulku atd.
- vždy záleží na konkrétním problému, jaké operace jsou podstatné, je-li podstatné udržovat uspořádané a tak dále

220/776

Slovník

- Pokud je s prvkem množiny svázán nějaký další údaj, mluvíme pak o **slovníku** (angl. dictionary, associative array, map, symbol table).
- Slovník udržuje dvojice (klíč, hodnota), kde klíč musí být ve slovníku unikátní.
- Matematicky jde o **zobrazení**.
- Nejdůležitější operace:
 - vložení dvojice do slovníku
 - smazání dvojice ze slovníku
 - modifikace hodnoty ve slovníku
 - vyhledání hodnoty pro daný klíč
- Pro implementaci lze využít pole, spojový seznam, binární vyhledávací strom, hašovací tabulku atd.

221/776

Děkuji za pozornost

Analýza složitosti algoritmů

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



222/776

Analýza složitosti algoritmů Základy analýzy složitosti algoritmů

Analýza algoritmů

Co analyzovat?

- správnost
- časová složitost
- prostorová složitost
- optimalita

Možné přístupy

- empirický a
- teoretický

223/776

Časová a prostorová složitost algoritmu

- **Časová složitost** – jak dlouho algoritmus bude pracovat.
- **Prostorová složitost** – kolik paměti bude algoritmus potřebovat **navíc** než je samotné uložení dat.
- Dříve byly oba zdroje kritické.
- Díky pokroku ve výpočetní technice je paměti relativně dost.
- Budeme zkoumat časovou složitost – tady lze dosáhnout významného pokroku.
- Ukazuje se, že prostorovou složitost lze zkoumat stejným aparátem jako časovou

224/776

Měření velikosti vstupu

- Triviální pozorování – větší data algoritmus obvykle zpracovává déle.
- Zavedeme parametr n označující velikost vstupních dat, který představuje například:
 - hledání v seznamu, poli – délka pole
 - vyhodnocení polynomu $p(x) = a_n x^n + \dots + a_1 x + a_0$ v bodě x – stupeň polynomu
 - násobení matic typu $n \times n$ – rozměr matice. Skutečný počet čísel na vstupu je ale n^2 , což je ale pořad závislé na n
 - kontrola pravopisu – počet znaků nebo počet slov, podle toho s čím algoritmus pracuje

225/776

Měření velikosti vstupu (pokrač.)

- test prvočíselnosti – vstupem je vždy jedno číslo (!) a , doba běhu závisí na velikosti čísla (srovnej test 2^3 a 2^{64}), velikostí vstupu bude počet bitů nutných k zápisu čísla

$$n = \lfloor \log_2 a \rfloor + 1 \quad (2)$$

- grafové úlohy – počet vrcholů a/nebo počet hran – zde už jsou dva parametry

226/776

Empirické měření složitosti

- Zadáme vhodná (?) vstupní data a změříme dobu běhu programu v obvyklých jednotkách času.
- Nevýhody:
 - Závislost na konkrétním HW, způsobu implementace, kompilátoru.
 - Chceme měřit složitost algoritmů – nemáme prostředky pro zachycení výše uvedených vlivů.
 - Vývoj HW – znamená to, že se algoritmy zrychlují? Ne, ty zůstávají stejné.
 - Počet operací provedených programem lze obtížně zjistit.
 - Chceme se obejít bez implementace – zkoumáme přece algoritmy.

227/776

Časová složitost algoritmu

Časovou složitost algoritmu budeme vyjadřovat (měřit) **počtem vykonaných základních operací** vzhledem k (jako funkci) **velikosti vstupu n** :

$$T(n) \approx c_{op} C(n),$$

kde

- n je velikost vstupu,
- $T(n)$ je doba běhu algoritmu,
- c_{op} je doba vykonání jedné základní operace a
- $C(n)$ je počet základních operací.

228/776

Základní operace

Typické operace pro daný algoritmus, které významně přispívají ke celkové „době běhu“ algoritmu.

Problém	Měřítko vstupu	Základní operace
Hledání prvku v seznamu	Počet prvků v seznamu	Porovnání prvků
Násobení matic	Rozměry matic	Aritmetické operace (násobení)
Test prvočíselnosti	Počet bitů čísla	Dělení čísel
Grafové úlohy	Počet vrcholů a / nebo hran	Zpracování vrcholu či průchod hranou

229/776

Řádový růst složitosti

Ke vztahu

$$T(n) \approx c_{op} C(n),$$

je potřeba ale přistupovat „s rezervou“, protože

1. $C(n)$ nebere v úvahu vliv jiných operací než základních a
2. c_{op} nelze spolehlivě zjistit.

Vztah chápeme jako **rozumný odhad doby běhu algoritmu**, mimo extrémně malých n .

230/776

Řádový růst složitosti (pokrač.)

Problém: Kolikrát rychleji poběží můj algoritmus na počítači, který je **10×** rychlejší než můj současný počítač?

Řešení:

Samozřejmě **10×**, c_{op} je desetinové.

231/776

Řádový růst složitosti (pokrač.)

Problém: Kolikrát déle poběží můj algoritmus pro dvojnásobně velký vstup, když $C(n) = \frac{1}{2}n(n - 1)$?

Řešení: Aproximujeme shora počet operací $C(n)$

$$C(n) = \frac{1}{2}n(n - 1) = \frac{1}{2}n^2 - \frac{1}{2}n < \frac{1}{2}n^2$$

a odtud

$$\frac{T(2n)}{T(n)} \approx \frac{c_{op}C(2n)}{c_{op}C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = \frac{4n^2}{n^2} = 4$$

Zanedbáno

Lineární člen $\frac{1}{2}n$

Podstatné

Kvadratický růst složitosti – mluvíme o **řádovém** růstu.

232/776

Řádový růst složitosti (pokrač.)

Tabulka ukazuje, jak rychle, či pomalu, rostou hodnoty vybraných funkcí pro různá n .

n	Hodnota funkce tj. počet operací $C(n)$						
	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3,3	10^1	$3,3 \cdot 10^1$	10^2	10^3	10^3	$3,6 \cdot 10^6$
10^2	6,6	10^2	$6,6 \cdot 10^2$	10^4	10^6	$1,3 \cdot 10^{30}$	$9,3 \cdot 10^{157}$
10^3	10	10^3	$1,0 \cdot 10^4$	10^6	10^9	n/a	n/a
10^4	13	10^4	$1,3 \cdot 10^5$	10^8	10^{12}	n/a	n/a
10^5	17	10^5	$1,7 \cdot 10^6$	10^{10}	10^{15}	n/a	n/a
10^6	20	10^6	$2,0 \cdot 10^7$	10^{12}	10^{18}	n/a	n/a

Pro malé n je rozdíl mezi hodnotami funkcí vcelku nezajímavý, ale pro zvětšující se n může být rozdíl propastný.

233/776

Řádový růst složitosti (pokrač.)

Poznámky

- Základ logaritmu není podstatný: $\log_a n = \log_a b \cdot \log_b n$.
- Počítači s rychlostí 10^{12} (tisíc miliard) operací za sekundu trvá provedení $2^{100} \approx 1,3 \cdot 10^{30}$ operací cca 40 miliard let. Stáří Země je cca 4,4 miliard let.
- O provedení $100!$ operací nebudeme ani uvažovat...

Algoritmy s exponenciální nebo faktoriálovou řádovou složitostí jsou použitelné jen pro velice malé velikosti vstupu!

234/776

Řádový růst složitosti (pokrač.)

Problém: Kolikrát déle poběží můj algoritmus pro dvojnásobně velký vstup, pro algoritmy s různým řádovým růstem?

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
$2n$	+1	2x	$\approx 2x$	4x	8x	$(\dots)^2$	n/a

protože

$$\log_2(2n) = \log_2 2 + \log_2 n = 1 + \log_2 n$$

$$2^{2n} = (2^n)^2$$

235/776

Analýza složitosti algoritmů
Nejhorší, nejlepší a průměrný případ

Nejhorší, nejlepší a průměrný případ

- Počet základních operací udáváme jako funkci s jedním parametrem n , velikostí vstupu.
- Některé algoritmy mohou mít i pro stejné n různé počty zákl. operací, například algoritmus lineárního vyhledávání.

Vstup : Pole $A[0 \dots n - 1]$ a hledaný prvek x

Výstup: Index prvního výskytu prvku x v poli A , jinak -1

```
1 for  $i \leftarrow 0$  to  $n - 1$  do
2   | if  $A[i] = x$  then
3   |   | return  $i$ ;
4   | end
5 end
6 return -1;
```

236/776

Nejhorší, nejlepší a průměrný případ (pokrač.)

Významné počty základních operací:

- $C_{worst}(n)$ – nejhorší případ, nejvyšší počet operací
- $C_{best}(n)$ – nejlepší případ, nejnižší počet operací
- $C_{avg}(n)$ – průměrný případ, průměrný počet operací.

237/776

Nejhorší případ $C_{worst}(n)$

- Analyzujeme algoritmus a hledáme vstup velikosti n pro který nastane nejvyšší možný počet operací.
- Nejhorší případ poskytuje horní mez složitosti, všechny ostatní případy jsou buď stejné nebo lepší.
- Nízký počet operací v nejhorším případě – pozitivní zpráva.

Příklad

Lineární vyhledávání: prvek x v poli A není nebo je nalezen až na konci, tedy $C_{worst}(n) = n$.

238/776

Nejlepší případ $C_{best}(n)$

- Obecně hledáme vstup velikosti n , pro který algoritmus vykoná nejmenší počet operací.
- Většinově nejlepší případ není tak důležitý jako nejhorší případ.
- Vstupy „podobné“, „blízké“ nejlepšímu. Třídění téměř setříděných posloupností.
- Nejlepší případ s „děsivým“ počtem operací – obecně špatná zpráva a „konečná“ pro algoritmus. Ale pro šifrovací algoritmus je „děsivý“ počet operací kryptoanalýzy i nejlepším případě nezbytný.

Příklad

Lineární vyhledávání: prvek x je prvním prvkem v poli A , $C_{best}(n) = 1$.

239/776

Průměrný případ $C_{avg}(n)$

- Počet operací v průměrném, „typickém“, „náhodném“ případě (nejlepší a nejhorší případy jsou extrémní).
- **Nejedná se o průměr nejlepšího a nejhoršího případu!**
- Musíme brát v úvahu pravděpodobnosti jednotlivých možných vstupů velikosti n .
- Analýza průměrného případu je tudíž komplikovanější než předchozích dvou.
- Existují algoritmy, kde se nejhorší a průměrný počet operací značně liší, např. QuickSort.

240/776

Průměrný případ $C_{avg}(n)$ – lineární vyhledávání

Předpoklady

1. pravděpodobnost úspěšného vyhledání p , kde $0 \leq p \leq 1$
2. pravděpodobnost nalezení na všech pozicích v poli je shodná a je rovna $\frac{p}{n}$

241/776

Průměrný případ $C_{avg}(n)$ – lineární vyhledávání (pokrač.)

Úspěšné vyhledání

- nalezení na první pozici – jedno porovnání s pravděpodobností $\frac{p}{n}$,
- nalezení na druhé pozici – dvě porovnání s pravděpodobností $\frac{p}{n}$, a tak dále, tedy

$$1 \frac{p}{n} + 2 \frac{p}{n} + \dots + i \frac{p}{n} + \dots + n \frac{p}{n}$$

Neúspěšné vyhledání

- pravděpodobnost neúspěchu je $1 - p$ a provedeme n porovnání, tj. $n(1 - p)$

242/776

Průměrný případ $C_{avg}(n)$ – lineární vyhledávání (pokrač.)

Odtud

$$\begin{aligned} C_{avg}(n) &= \left(1 \frac{p}{n} + 2 \frac{p}{n} + \dots + i \frac{p}{n} + \dots + n \frac{p}{n} \right) + n(1 - p) \\ &= \frac{p}{n} (1 + 2 + \dots + i + \dots + n) + n(1 - p) \\ &= \frac{p}{n} \left[\frac{1}{2} n(n + 1) \right] + n(1 - p) \\ &= \frac{1}{2} p(n + 1) + n(1 - p) \end{aligned}$$

Rozbor

- vždy úspěšné hledání, $p = 1$ a tedy $C_{avg}(n) = \frac{1}{2}(n + 1)$
- neúspěšné hledání, $p = 0$ a tedy $C_{avg}(n) = n$

243/776

Amortizovaná složitost

- Nezkoumáme jeden, izolovaný, běh algoritmu, ale zkoumáme „sadu“ běhů s různými vstupy stejné velikosti.
- Zajímá nás celkový počet operací za sadu.
- Počet operací pro jeden vstup ze sady může být sice vysoký, ale je vyvážen, „amortizován“ výrazně menším počtem operací pro další vstupy ze sady.
- Například jeden ze vstupů způsobí rozsáhlou změnu v datové struktuře a díky tomu zpracování dalších vstupů proběhne snadněji.
- V průmyslu je například nákup drahého stroje amortizován levnější výrobou výrobku.

244/776

Zdroje pro samostatné studium

- Kniha [2], kapitola 2.1, strany 42 – 51
- Kniha [3], kapitola 2.2, strany 25 – 34 (částečně)

245/776

Analýza složitosti algoritmů

Asymptotické notace složitosti

O-notace

Definice

Mějme funkce $t(n)$ a $g(n)$, kde $t(n), g(n) : \mathbb{N} \rightarrow \mathbb{N}$. Říkáme, že funkce $t(n)$ patří do $O(g(n))$, jestliže existuje kladná nenulová reálná konstanta c a přirozené číslo $n_0 > 1$ takové, že

$$t(n) \leq cg(n)$$

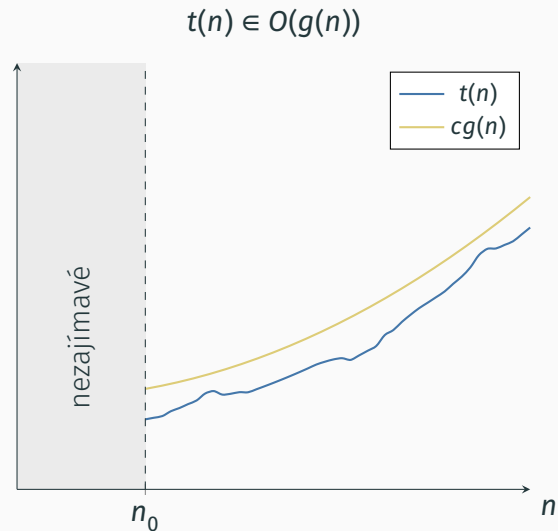
pro všechna $n \geq n_0$.

Poznámka

Místo „ $t(n)$ patří do $O(g(n))$ “ můžeme říkat, že „ $t(n)$ je řádu $O(g(n))$ “.

246/776

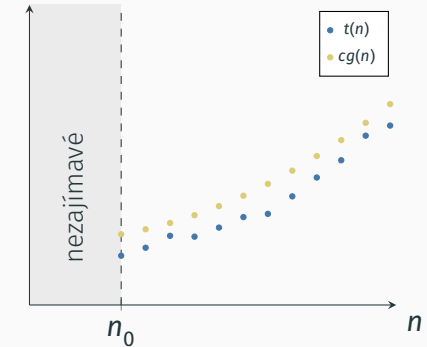
O-notace graficky



247/776

O-notace – formálně správný graf

Formálně jsou definičním oborem i oborem hodnot funkcí $t(n)$ i $g(n)$ přirozená čísla \Rightarrow graf by měl být složen pouze z bodů, nikoliv křivek.



Proložíme-li body křivky \Rightarrow dostáváme spojité funkce \Rightarrow můžeme použít k výpočtům matematickou analýzu (limity, derivace atd.).

248/776

O-notace – příklad 1

Dokažte, že $3n + 7 \in O(n)$.

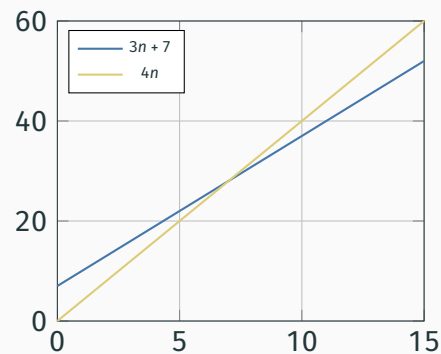
Řešení

1. Hledáme konstanty c a n_0 takové, aby platilo

$$3n + 7 \leq cn$$

pro všechna $n \geq n_0$.

2. Je zřejmé, že $c > 3$. Zvolíme-li např. $c = 4$, pak $n_0 = 7$.



249/776

O-notace – příklad 2

Dokažte, že $3n + 7 \in O(n^2)$

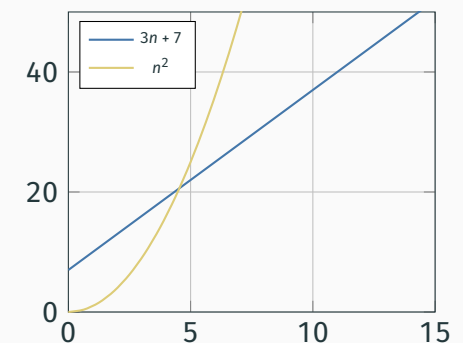
Řešení

1. Hledáme konstanty c a n_0 takové, aby platilo

$$3n + 7 \leq cn^2$$

pro všechna $n \geq n_0$.

2. Zvolíme-li $c = 1$, pak $n_0 = 5$.



250/776

O-notace – příklad 3

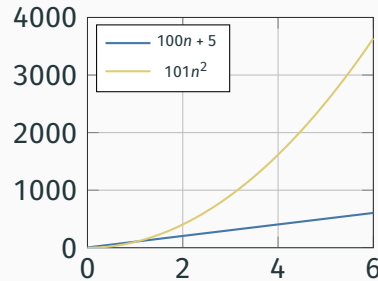
Dokažte, že $100n + 5 \in O(n^2)$.

Řešení

1. Platí, že $100n + 5 \leq 100n + n$ pro všechna $n \geq 5$.
2. Dále platí, že $101n \leq 101n^2$.
3. Odtud

$$100n + 5 \leq 101n \leq 101n^2$$

a tedy $c = 101$ a $n_0 = 5$.



251/776

O-notace – příklad 3 (pokrač.)

Ve výběru konstant c a n_0 máme ale mnoho možností – lze postupovat i takto:

$$100n + 5 \leq 100n + 5n = 105n$$

pro všechna $n \geq 1$. Z toho plyne, že

$$105n \leq 105n^2$$

a tudíž $c = 105$ a $n_0 = 1$.

252/776

Ω -notace

Definice

Mějme funkce $t(n)$ a $g(n)$, kde $t(n), g(n) : \mathbb{N} \rightarrow \mathbb{N}$. Říkáme, že funkce $t(n)$ patří do $\Omega(g(n))$, jestliže existuje kladná nenulová reálná konstanta c a přirozené číslo $n_0 > 1$ takové, že

$$t(n) \geq cg(n)$$

pro všechna $n \geq n_0$.

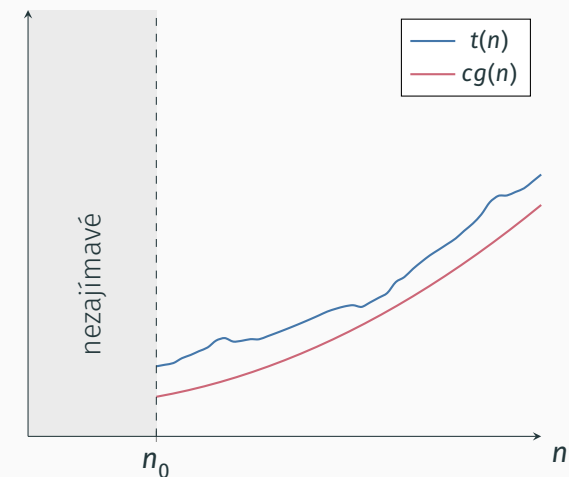
Příklad

Dokažte, že platí $n^3 \in \Omega(n^2)$. Zřejmě platí, že $n^3 \geq n^2$ pro všechna $n \geq 0$. Tudíž můžeme volit $c = 1$ a $n_0 = 1$.

253/776

Ω -notace graficky

$t(n) \in \Omega(g(n))$



254/776

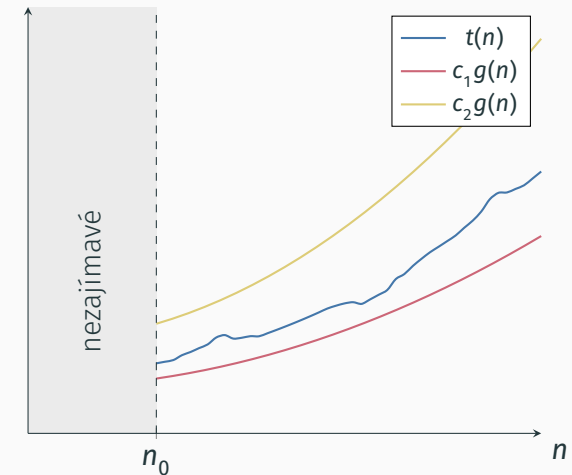
Definice

Mějme funkce $t(n)$ a $g(n)$, kde $t(n), g(n) : \mathbb{N} \rightarrow \mathbb{N}$. Říkáme, že funkce $t(n)$ patří do $\Theta(g(n))$, jestliže existují kladné nenulové reálné konstanty c_1, c_2 a přirozené číslo $n_0 > 1$ takové, že

$$c_1 g(n) \leq t(n) \leq c_2 g(n)$$

pro všechna $n \geq n_0$.

$$t(n) \in \Theta(g(n))$$



Zadání

Dokažte, že $\frac{1}{2}n(n-1) \in \Theta(n^2)$

Řešení

1. Nejprve dokážeme pravou nerovnost $t(n) \leq c_2 g(n)$ (omezení shora)

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2$$

pro všechna $n \geq 0$.

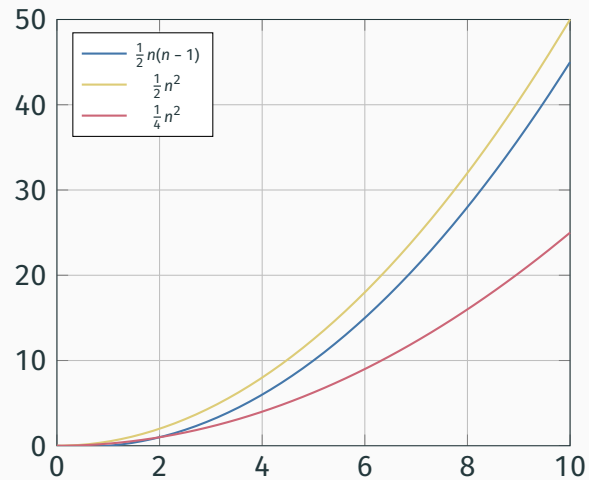
2. Levou nerovnost $c_1 g(n) \leq t(n)$ (omezení zdola) dokážeme takto:

$$\begin{aligned} t(n) = \frac{1}{2}n(n-1) &= \frac{1}{2}n^2 - \frac{1}{2}n \\ &\geq \frac{1}{2}n^2 - \frac{1}{2}n \cdot \frac{1}{2} \\ &\geq \frac{1}{2}n^2 - \frac{1}{4}n^2 \\ &\geq \frac{1}{4}n^2 \end{aligned}$$

Souhrnně tedy $\frac{1}{4}n^2 \leq \frac{1}{2}n(n-1)$ pro všechna $n \geq 2$.

3. Z předchozích nerovností plyne, že $c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$ a $n_0 = 2$.

Θ-notace – příklad (pokrač.)



259/776

Vlastnosti asymptotické notace

Základní vlastnosti:

1. $f(n) \in O(f(n))$
2. $f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n))$
3. $f(n) \in O(g(n)) \wedge g(n) \in O(h(n)) \implies f(n) \in O(h(n))$
tranzitivita, srovnej s $a \leq b \wedge b \leq c \implies a \leq c$

260/776

Vlastnosti asymptotické notace (pokrač.)

Problém

- Algoritmus A se skládá z částí A_1 a A_2 .
- Části algoritmu se vykonávají po sobě, tj. po dokončení A_1 se začne vykonávat A_2 .
- Složitost části A_1 je $t_1(n) \in O(g_1(n))$, složitost části A_2 je $t_2(n) \in O(g_2(n))$.
- Otázka zní – jaká je celková složitost algoritmu A ?

261/776

Vlastnosti asymptotické notace (pokrač.)

Věta

Pokud $t_1(n) \in O(g_1(n))$ a současně $t_2(n) \in O(g_2(n))$, potom

$$t_1(n) + t_2(n) \in O(\max(g_1(n), g_2(n))).$$

Poznámka

Shodně tvrzení můžeme vyslovit i pro Ω a Θ notaci.

262/776

Vlastnosti asymptotické notace (pokrač.)

Důkaz.

Protože $t_1(n) \in O(g_1(n))$, tak existuje kladná nenulová konstanta c_1 a nezáporná konstanta n_1 taková, že

$$t_1(n) \leq c_1 g_1(n) \quad \forall n \geq n_1.$$

Obdobně

$$t_2(n) \leq c_2 g_2(n) \quad \forall n \geq n_2.$$

263/776

Vlastnosti asymptotické notace (pokrač.)

Důkaz.

Označme $c_3 = \max(c_1, c_2)$ a $n_0 \geq \max(n_1, n_2)$. Potom platí

$$\begin{aligned} t_1(n) + t_2(n) &\leq c_1 g_1(n) + c_2 g_2(n) \\ &\leq c_3 g_1(n) + c_3 g_2(n) = c_3 [g_1(n) + g_2(n)] \\ &\leq 2c_3 \max(g_1(n), g_2(n)). \end{aligned}$$

Tudíž $t_1(n) + t_2(n) \in O(\max(g_1(n), g_2(n)))$, protože existují konstanty $c = 2c_3 = 2 \max(c_1, c_2)$ a $n_0 = \max(n_1, n_2)$. \square

Závěr: Celkovou složitost algoritmu určuje část algoritmu s **nejvyšší** složitostí.

264/776

Vlastnosti asymptotické notace (pokrač.)

Příklad

Zadání: Test, zda se v poli vyskytují dvě shodné hodnoty.

Řešení:

1. Setřídění pole nevyžaduje ne více než $\frac{1}{2}n(n-1)$ porovnání, tj. složitost třídy $O(n^2)$.
2. Porovnání všech dvojic sousedních prvků bude vyžadovat $n-1$ porovnání, tj. složitost třídy $O(n)$.

Celková složitost algoritmu je tedy $O(\max(n^2, n)) = O(n^2)$.

265/776

Využití limit k výpočtům

Rychlost růstu funkcí lze snadněji počítat pomocí limit:

$$\lim_{n \rightarrow \infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & t(n) \text{ roste } \text{pomaleji} \text{ než } g(n) \\ c & t(n) \text{ roste } \text{stejně rychle} \text{ jako } g(n) \\ \infty & t(n) \text{ roste } \text{rychleji} \text{ než } g(n) \end{cases}$$

Je zřejmé, že:

$$\begin{aligned} t(n) \in O(g(n)) &\Leftrightarrow t(n) \text{ roste pomaleji nebo stejně rychle než } g(n) \\ t(n) \in \Omega(g(n)) &\Leftrightarrow t(n) \text{ roste stejně rychle nebo rychleji než } g(n) \\ t(n) \in \Theta(g(n)) &\Leftrightarrow t(n) \text{ roste stejně rychle jako } g(n) \end{aligned}$$

266/776

Využití limit k výpočtům (pokrač.)

Některé užitečné vzorce

L'Hospitalovo pravidlo

$$\lim_{n \rightarrow \infty} \frac{t(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{t'(n)}{g'(n)}$$

Stirlingův vzorec

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

267/776

Využití limit k výpočtům – příklad I

Srovnejte rychlost růstu funkcí $\frac{1}{2}n(n-1)$ a n^2 .

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\frac{1}{2}n(n-1)}{n^2} &= \frac{1}{2} \lim_{n \rightarrow \infty} \frac{n^2 - n}{n^2} \\ &= \frac{1}{2} \lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right) \\ &= \frac{1}{2} \left(\lim_{n \rightarrow \infty} 1 - \lim_{n \rightarrow \infty} \frac{1}{n}\right) \\ &= \frac{1}{2}(1 - 0) = \frac{1}{2} > 0 \end{aligned}$$

Funkce $\frac{1}{2}n(n-1)$ a n^2 rostou stejně rychle, tedy

$$\frac{1}{2}n(n-1) \in \Theta(n^2)$$

268/776

Využití limit k výpočtům – příklad II

Srovnejte rychlost růstu funkcí $\log_2 n$ a \sqrt{n} .

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\log_2 n}{\sqrt{n}} &= \lim_{n \rightarrow \infty} \frac{(\log_2 n)'}{(\sqrt{n})'} \\ &= \lim_{n \rightarrow \infty} \frac{(\log_2 e) \frac{1}{n}}{\frac{1}{2\sqrt{n}}} = (\log_2 e) \lim_{n \rightarrow \infty} \frac{1}{\frac{1}{2\sqrt{n}}} \\ &= \log_2 e \lim_{n \rightarrow \infty} \frac{2\sqrt{n}}{n} \\ &= 2 \log_2 e \lim_{n \rightarrow \infty} \frac{1}{\sqrt{n}} = 0 \end{aligned}$$

Funkce $\log_2 n$ tedy **roste pomaleji** než \sqrt{n} .

269/776

Využití limit k výpočtům – příklad III

Srovnejte rychlost růstu funkcí $n!$ a 2^n .

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{n!}{2^n} &= \lim_{n \rightarrow \infty} \frac{\sqrt{2\pi n} \left(\frac{n}{e}\right)^n}{2^n} \\ &= \sqrt{2\pi} \lim_{n \rightarrow \infty} \sqrt{n} \frac{n^n}{2^n e^n} \\ &= \sqrt{2\pi} \lim_{n \rightarrow \infty} \sqrt{n} \left(\frac{n}{2e}\right)^n = \infty \end{aligned}$$

Poznámky

- Funkce $n!$ tedy roste rychleji než 2^n .
- Definice Θ -notace nevylučuje, že $n! \in \Omega(2^n)$, ale výpočet podle limity jasně říká, že $n!$ roste rychleji než 2^n

270/776

Základní třídy složitosti

Přestože teoreticky existuje nekonečně mnoho tříd složitosti, složitost většiny algoritmů padne do několika málo tříd.

Třída	Jméno	Poznámka
1	konstantní	složitost nezávisí na velikosti vstupu; jen velmi málo algoritmů
$\log n$	logaritmická	typicky algoritmy redukující velikost vstupu konstantním faktorem; vyhledávání půlením intervalu
n	lineární	algoritmy zpracovávající seznam o n prvcích; např. sekvenční vyhledávání
$n \log n$	lineárně-logaritmická	„rozděl a panuj“ algoritmy; průměrné složitosti QuickSortu, MergeSortu

271/776

Základní třídy složitosti (pokrač.)

Třída	Jméno	Poznámka
n^2	kvadratická	obecně algoritmy se dvěma vnořenými cykly; elementární metody třídění, sčítání matic typu $n \times n$
n^3	kubická	obecně algoritmy se třemi vnořenými cykly; násobení matic typu $n \times n$
2^n	exponenciální	typicky generování všech podmnožin n prvkové množiny
$n!$	faktoriál	typicky generování všech permutací n prvkové množiny

272/776

Vliv multiplikační konstanty

- Třída složitosti je dána až na multiplikační konstantu, která obvykle není přesně specifikována.
- Mohl by tedy algoritmus s vyšší třídy složitosti běžet, pro nějaké rozumné n , rychleji než algoritmus z lepší třídy? Například:

Algoritmus	Doba běhu	
A	n^3	A bude lepší než B pro $n < 10^6$.
B	$10^6 n^2$	

- Multiplikační konstanty obvykle nabývají podobných, relativně malých, hodnot.
- Lze očekávat, že algoritmy s nižší složitostí budou lepší než ty vyšší složitostí už pro středně velké vstupy.

273/776

Zdroje pro samostatné studium

- Kniha [2], kapitola 2.2, strany 52 – 61
- Kniha [3], kapitoly 3.1 a 3.2, strany 49 – 63

274/776

Analýza složitosti algoritmů

Analýza nerekurzivních algoritmů

Nalezení největšího prvku v poli n čísel

Vstup : Pole $A[0 \dots n - 1]$ celých čísel

Výstup: Největší prvek pole A

```
1  $max \leftarrow A[0]$ ;  
2 for  $i \leftarrow 1$  to  $n - 1$  do  
3   | if  $A[i] > max$  then  
4   |   |  $max \leftarrow A[i]$ ;  
5   |   end  
6 end  
7 return  $max$ ;
```

275/776

Nalezení největšího prvku v poli n čísel (pokrač.)

Pracovní postup

1. Velikost vstupu – velikost pole n
2. Základní operace:
 - nejčastěji vykonávané operace jsou uvnitř cyklu – porovnání $A[i] > max$ a přiřazení $max \leftarrow A[i]$
 - základní operací bude **porovnání**, protože se
 - provede v každém průchodu cyklem,
 - je to klíčová operace pro algoritmus, „Kolik dvojic prvků musím porovnat, abych našel maximum?“
3. Počet porovnání je stejný pro všechny vstupy velikosti n , není nutné rozlišovat mezi nejlepším, průměrným a nejhorším případem

276/776

Nalezení největšího prvku v poli n čísel (pokrač.)

4. Počet základních operací, porovnání, $C(n)$ bude roven

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$

5. **Závěr:** Nalezení největšího prvku v poli n čísel je **lineární algoritmus**.

277/776

Nalezení největšího prvku v poli n čísel, všechny operace

Počet operací	Popis
1	přiřazení $max \leftarrow A[0]$
1	přiřazení $i \leftarrow 1$
$n - 1$	porovnání $i \leq n - 1$
$n - 1$	zvýšení i o 1
$n - 1$	porovnání $A[i] > max$
$n - 1$	přiřazení $max \leftarrow A[i]$
1	vrácení výsledku $return max$
<hr/>	
$4(n - 1) + 3 = 4n - 1 \in \Theta(n)$	

Závěr: Nalezení největšího prvku v poli n čísel je **lineární** algoritmus.

278/776

Obecný postup určení časové složitosti nerek. algoritmů

1. Volba parametru, či parametrů, reprezentujícího velikost vstupu n .
2. Nalezení základních operací algoritmu (jsou to ty v nejvíce vnořeném cyklu!).
3. Závisí počet základních operací jen na velikosti vstupu? Pokud závisí i na něčem dalším, musíme zkoumat nejhorší, nejlepší a průměrný případ zvlášť.
4. Sestavení vztahu, resp. vztahů, („vzorečků“) vyjadřujících počet, resp. počty, provedení základních operací.
5. Zjednodušení sestavených vztahů a, nebo aspoň, stanovení řadového růstu.

279/776

Užitečné součtové vzorce

$$\sum (a_i \pm b_i) = \sum a_i \pm \sum b_i \quad (3)$$

$$\sum ca_i = c \sum a_i \quad (4)$$

$$\sum_{i=1}^n a_i = \sum_{i=1}^m a_i + \sum_{i=m+1}^n a_i \quad (5)$$

$$\sum_{i=l}^u 1 = 1 + 1 + \dots + 1 = u - l + 1 \quad (6)$$

Konkrétně

$$\sum_{i=1}^n 1 = n \in \Theta(n) \quad (7)$$

280/776

Užitečné součtové vzorce (pokrač.)

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{1}{2}n(n+1) \approx \frac{1}{2}n^2 \in \Theta(n^2) \quad (8)$$

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + \dots + n^2 = \frac{1}{6}n(n+1)(2n+1) \approx \frac{1}{3}n^3 \in \Theta(n^3) \quad (9)$$

$$\sum_{i=0}^n a^i = 1 + a + a^2 + \dots + a^n = \frac{a^{n+1} - 1}{a - 1}, \text{ pro } a \neq 1 \quad (10)$$

Konkrétně

$$\sum_{i=0}^n 2^i = 2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1 \in \Theta(2^n) \quad (11)$$

281/776

Unikátnost prvků v poli

Je dáno pole o n prvcích. Naším úkolem provést analýzu algoritmu, který zjistí, zda všechny prvky v poli jsou navzájem různé, čili unikátní.

Vstup : Pole $A[0 \dots n - 1]$

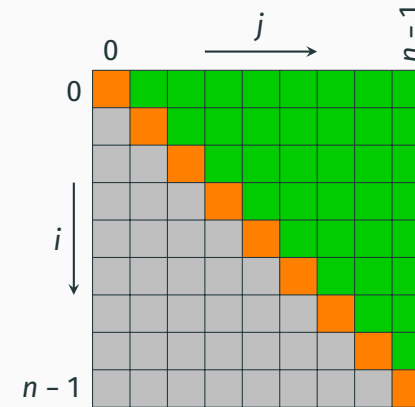
Výstup: Vrací true, pokud jsou všechny prvky unikátní, jinak vrací false

```
1 for  $i \leftarrow 0$  to  $n - 2$  do
2   for  $j \leftarrow i + 1$  to  $n - 1$  do
3     if  $A[i] = A[j]$  then
4       return false;
5     end
6   end
7 end
8 return true;
```

282/776

Unikátnost prvků v poli (pokrač.)

Vizualizace algoritmu



Legenda

- dvojice, které je nutně otestovat
- prvek sám se sebou není nutně testovat
- dvojice testované v předchozích průchodech cyklem

283/776

Unikátnost prvků v poli (pokrač.)

Pracovní postup

1. Velikost vstupu – velikost pole n
2. Základní operace – nejvíce vnořený cyklus obsahuje jedinou operaci, porovnání $A[i] = A[j]$
3. Závislost pouze na n ? Ne, počet zákl. operací závisí i na tom, zda se v poli objeví shodný prvek. Tudiž provádíme analýzu **nejhoršího**, **nejlepšího** a **průměrného** případu.
4. Sestavení vztahů. Pro nejhorší případ je z vnitřního cyklu je patrné, že nesmí dojít k předčasnému ukončení cyklu, a to buď:
 - 4.1 protože všechny prvky jsou unikátní nebo
 - 4.2 je shodná až poslední dvojice.

284/776

Unikátnost prvků v poli (pokrač.)

Tudiž provedeme

- jedno porovnání pro každý průchod vnitřním cyklem, tj. $j = i + 1, \dots, n - 1$
- vnější cyklus, v každém svém průchodu, zopakuje celý vnitřní cyklus, tj. $i = 0, \dots, n - 2$

285/776

Unikátnost prvků v poli (pokrač.)

$$\begin{aligned} C_{\text{worst}}(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 && \text{podle (6)} \\ &= \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i) \\ &= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i && \text{podle (3)} \\ &= (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} && \text{podle (4) a (3)} \\ &= (n-1)^2 - \frac{(n-2)(n-1)}{2} && \text{podle (6)} \\ &= \frac{1}{2}n(n-1) - \frac{1}{2}n^2 = \Theta(n^2) \end{aligned}$$

286/776

Násobení čtvercových matic

Naším úkolem je provést analýzu algoritmu pro výpočet součinu $C = AB$ dvou čtvercových matic A a B řádu n .

Z definice jsou prvky matice rovny skalárním součinům řádků matice A se sloupci matice B .

$$\begin{array}{c} \text{row } i \\ \left[\begin{array}{cccc} & & & \end{array} \right] \end{array} * \begin{array}{c} \text{col. } j \\ \left[\begin{array}{c} \\ \\ \\ \end{array} \right] \end{array} = \begin{array}{c} \text{C} \\ \left[\begin{array}{c} \\ \\ \\ \end{array} \right] \\ C[i,j] \end{array}$$
$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$$

pro všechna
 $0 \leq i, j \leq n-1$

$$C[i, j] = A[i, 0] \times B[0, j] + \dots + A[i, k] \times B[k, j] + \dots + A[i, n-1] \times B[n-1, j]$$

287/776

Násobení čtvercových matic (pokrač.)

Vstup : Dvě čtvercové matice A a B řádu n

Výstup: Čtvercová matice C řádu n , kde $C = AB$

```
1 for i ← 0 to n - 1 do
2   for j ← 0 to n - 1 do
3     C[i, j] ← 0;
4     for k ← 0 to n - 1 do
5       C[i, j] ← C[i, j] + A[i, k] × B[k, j];
6     end
7   end
8 end
9 return C;
```

288/776

Násobení čtvercových matic (pokrač.)

Pracovní postup

1. Velikost vstupu – řád matice n
2. Základní operace:
 - nejvíce vnořený cyklus obsahuje dvě operace – sčítání a násobení,
 - v každém průchodu se obě provedou přesně jedenkrát,
 - historická tradice velí počítat násobení (bývalo mnohokrát pomalejší než sčítání),
 - zavedeme $M(n)$ jako celkový počet násobení.
3. Počet operací závisí pouze na n – nejhorší, nejlepší a průměrný případ splývají

289/776

Násobení čtvercových matic (pokrač.)

4. Sestavení vztahů – počet násobení v nejnvnitřnějším cyklu

$$\sum_{k=0}^{n-1} 1$$

Celkový počet násobení je roven

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1$$

290/776

Násobení čtvercových matic (pokrač.)

Pomocí vztahů (4) a (6) postupně dostáváme

$$\begin{aligned} M(n) &= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (n - 1 - 0 + 1) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = n \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 \\ &= n \sum_{i=0}^{n-1} (n - 1 - 0 + 1) = n \sum_{i=0}^{n-1} n = n^2 \sum_{i=0}^{n-1} 1 \\ &= n^2(n - 1 - 0 + 1) \\ &= n^3 \end{aligned}$$

291/776

Násobení čtvercových matic (pokrač.)

Neformální postup:

1. algoritmus musí vypočítat $n \times n$ prvků matice C
2. každý prvek matice C je vypočten jako skalární součin i -tého řádku matice A a j -tého sloupce matice B
3. řádky i sloupce mají n prvků, které musíme vynásobit
4. celkem tedy $n^2 \times n = n^3$ násobení

292/776

Násobení čtvercových matic (pokrač.)

Doba běhu algoritmu na konkrétním počítači

$$T(n) \approx c_m M(n) = c_m n^3$$

započteme-li i sčítání

$$T(n) \approx c_m M(n) + c_a A(n) = c_m n^3 + c_a n^3 = (c_m + c_a) n^3,$$

kde c_m resp. c_a je doba trvání násobení resp. sčítání, $A(n)$ je počet operací sčítání, platí $A(n) = M(n)$.

Shrnutí

Doba běhu algoritmu se může, v závislosti na konkrétním počítači, měnit, ale řádová složitost algoritmu (n^3) zůstává.

293/776

Počet bitů v binárním zápisu čísla

Naším úkolem je analyzovat algoritmus, který pro dané přirozené číslo n vypočte počet bitů nutných pro zápis čísla n v binární soustavě.

Vstup : Přirozené číslo n

Výstup: Počet bitů v binárním zápisu čísla n

```
1 count ← 1;
2 while n > 1 do
3   | count ← count + 1;
4   | n ←  $\lfloor n/2 \rfloor$ ;
5 end
6 return count;
```

- Velikost vstupu – jedno číslo?

294/776

Počet bitů v binárním zápisu čísla (pokrač.)

- Základní operace – sčítání, dělení, porovnání s 1?
- Nejdůležitější je, v tomto případě, určit počet průchodů cyklem. Počet porovnání je o jedna větší než počet průchodů cyklem.
- Hodnota čísla n se v každém průchodu cyklem zmenšuje na polovinu, což vede ke vztahu

$$\lfloor \log_2 n \rfloor + 1$$

a což odpovídá vztahu (2).

- K odvození budeme potřebovat umět řešit rekurzivní rovnice...

295/776

Zdroje pro samostatné studium

- Kniha [2], kapitola 2.3, strany 61 – 70

296/776

Analýza složitosti algoritmů

Analýza rekurzivních algoritmů

Výpočet faktoriálu

Naším úkolem je analyzovat rekurzivní algoritmus, který pro dané přirozené číslo n vypočte jeho faktoriál $n!$.

$$n! = \begin{cases} 1 & \text{pro } n = 0 \\ n(n-1)! & \text{jinak} \end{cases}$$

```
1 Function F(n)
  | Vstup: Přirozené číslo n
  | Výsledek: Hodnota n!
2  if n = 0 then
3  |   return 1;
4  end
5  return n × F(n - 1);
6 end
```

297/776

Výpočet faktoriálu (pokrač.)

- Velikost vstupu – jedno číslo. Budeme brát v úvahu počet bitů? Ne, bylo by to komplikované.
- Základní operace – násobení. Alternativně lze uvažovat počet porovnání $n = 0$, které odpovídá počtu volání funkce F .
- Bude nás zajímat počet násobení $M(n)$ v závislosti na čísle n

298/776

Výpočet faktoriálu (pokrač.)

- Pro $n > 0$ se funkce $F(n)$ počítá jako

$$F(n) = F(n-1) \times n$$

a odtud

$$M(n) = M(n-1) + 1, \quad (12)$$

kde

$M(n-1)$ výpočet funkce $F(n-1)$ a
1 vynásobení výsledku $F(n-1)$ číslem n .

- Hodnota $M(n)$ není definována explicitně tj. jako funkce n například n^3 , ale **implicitně** pomocí vztahu založeném na hodnotě totožné funkce pro jiné přirozené číslo, konkrétně $n-1$. Jde o tzv. **rekurentní (rekurzivní) vztah**.

299/776

Výpočet faktoriálu (pokrač.)

Poznámka

Explicitní	Implicitní
výslovný, přímý, jasný, zřetelný	zahrnutý, obsažený, ale nevyjádřený přímo
otevřeně, přímo vyjádřený, nenechávací nic zamlčeného, skrytého	nikoli zjevný, samo sebou se rozumějící
významově navzájem opačná slova	

Příklad

Explicitně: Lžete.

Implicitně: O pravdivosti vašeho tvrzení by se dalo s úspěchem pochybovat.

300/776

Výpočet faktoriálu (pokrač.)

- Cílem je najít explicitní vyjádření $M(n)$.
- Vztah $M(n) = M(n - 1) + 1$ není jednoznačný, pro jednoznačné řešení je nutné definovat **počáteční podmínku**.

```
• Podmínka
  if  $n = 0$  then
  |   return 1;
  end
```

nám říká:

1. nejmenší n pro které se algoritmus provede je $n = 0$ a
2. v tomto případě algoritmus neprovede žádné násobení, tudíž $M(0) = 0$.

301/776

Výpočet faktoriálu (pokrač.)

- Celkově tedy pro výpočet $M(n)$ platí

$$M(n) = M(n - 1) + 1 \text{ pro } n > 0$$

$$M(0) = 0$$

- Soustavu budeme řešit **metodou zpětné substituce**. Do

$$M(n) = M(n - 1) + 1$$

dosadíme za $M(n - 1) = M(n - 2) + 1$

$$M(n) = [M(n - 2) + 1] + 1 = M(n - 2) + 2$$

302/776

Výpočet faktoriálu (pokrač.)

a opět dosadíme za $M(n - 2) = M(n - 3) + 1$

$$M(n) = [M(n - 3) + 1] + 2 = M(n - 3) + 3.$$

Je zřejmé, že

$$M(n) = M(n - i) + i$$

Poznámka

Správnost této formule lze dokázat pomocí matematické indukce.

303/776

Výpočet faktoriálu (pokrač.)

Počáteční podmínka je definována pro $n = 0$, takže musíme dosadit za $i = n$ a dostáváme

$$M(n) = M(n - 1) + 1 = \dots = M(n - i) + i = \dots = M(n - n) + n = n.$$

Shrnutí

1. Výsledek $M(n) = n$ byl víceméně očekávaný.
2. Iterativní algoritmus provede stejný počet násobení jak rekursivní navíc bez režie spojené se zásobníkem pro volání funkcí.
3. Důležitý je ale popsán přístup jak řešit rekurentní rovnice.

304/776

Obecný postup určení časové složitosti rekurzivních algoritmů

1. Volba parametru, či parametrů, reprezentujícího velikost vstupu n .
2. Nalezení základních operací algoritmu.
3. Závisí počet základních operací jen na velikosti vstupu? Pokud závisí i na něčem dalším, musíme zkoumat nejhorší, nejlepší a průměrný případ zvlášť.
4. Sestavení rekurentního vztahu a vhodné počáteční podmínky, vyjadřující počet provedení základních operací.
5. Zjednodušení sestavených vztahů a, nebo aspoň, stanovení řádového růstu.

305/776

Hanojské věže (Tower of Hanoi)

- Matematický hlavolam, autor Édouard Lucas, 1883.
- Hlavolam se skládá ze tří tyčí.
- Na začátku je na jedné tyči nasazeno několik kotoučů různých poloměrů, seřazených od největšího (vespod) po nejmenší (nahore).



- Úkolem je přemístit všechny kotouče z první tyče na třetí tyč za pomoci druhé tyče.
- Pravidla hry:
 - V jednom tahu lze přemístit jen jeden kotouč.

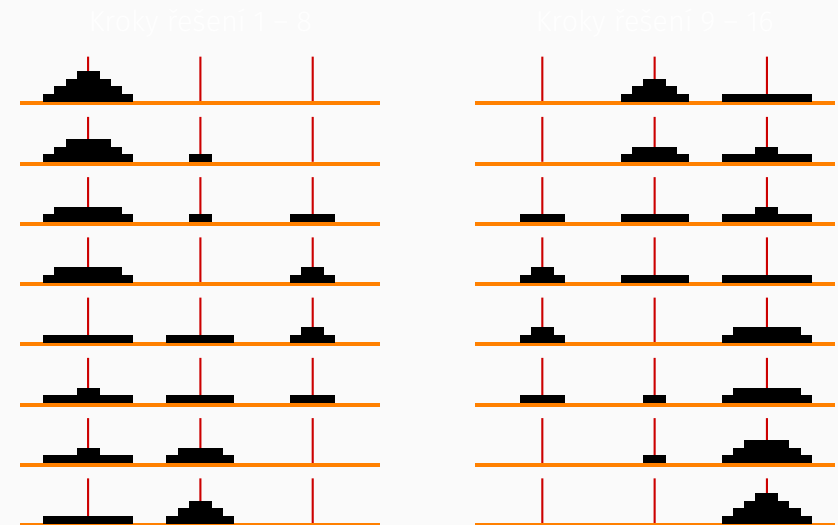
306/776

Hanojské věže (Tower of Hanoi) (pokrač.)

- Jeden tah se sestává ze sejmutí vrchního kotouče z některé tyče a jeho navlečení na jinou tyč.
- Je zakázáno položit větší kotouč na menší.
- Podle legendy stojí v Hanoji klášter, v němž jsou hanojské věže se 64 zlatými kotouči. Mnozí každý den v poledne přemístí jeden kotouč. V okamžiku, kdy bude přemístěn poslední kotouč, nastane konec světa.
- Hlavní klíč: Řešení tohoto hlavolamu pro 64 kotoučů vyžaduje $2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$ tahů. I kdyby přemístili každou sekundu jeden kotouč (a postupovali nejkratším možným způsobem), doba řešení je cca 600 miliard let.

307/776

Hanojské věže (Tower of Hanoi) (pokrač.)

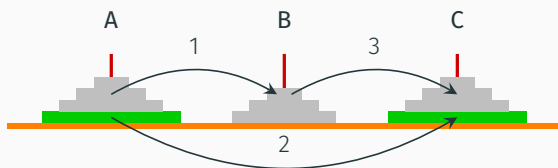


308/776

Hanojské věže (Tower of Hanoi) (pokrač.)

Řešení problému pro n disků

1. Přesun $n - 1$ disků z tyče A na tyče B (pomocí tyče C).
2. Přesun největšího disku z tyče A přímo na tyč C.
3. Přesun $n - 1$ disků z tyče B na tyč C (pomocí tyče A).



Pro $n = 1$ existuje triviální řešení...

309/776

Hanojské věže (Tower of Hanoi) (pokrač.)

Pracovní postup

1. Velikost vstupu – počet disků n .
2. Základní operace – přesun disku.
3. Závislost pouze na n ? Ano. Tudiž **nemusíme** zkoumat nejhorší, nejlepší a průměrný případ zvlášť.
4. Sestavení vztahů

$$M(n) = \begin{cases} 1 & \text{pro } n = 1 \\ 2M(n-1) + 1 & \text{jinak} \end{cases} \quad (13)$$

310/776

Hanojské věže (Tower of Hanoi) (pokrač.)

5. Řešení metodou zpětné substituce. Do vztahu

$$M(n) = 2M(n-1) + 1$$

dosadíme $M(n-1) = 2M(n-2) + 1$

$$M(n) = 2[2M(n-2) + 1] + 1 = 2^2M(n-2) + 2 + 1,$$

opět dosadíme $M(n-2) = 2M(n-3) + 1$

$$M(n) = 2^2[2M(n-3) + 1] + 2 + 1 = 2^3M(n-3) + 2^2 + 2 + 1,$$

po dalším dosazení dostaneme

$$M(n) = 2^4M(n-4) + 2^3 + 2^2 + 2 + 1$$

311/776

Hanojské věže (Tower of Hanoi) (pokrač.)

Po i -tém dosazení dostáváme

$$\begin{aligned} M(n) &= 2^i M(n-i) + \underbrace{2^{i-1} + 2^{i-2} + \dots + 2 + 1}_{\text{sečteme podle (11)}} \\ &= 2^i M(n-i) + 2^i - 1. \end{aligned}$$

Počáteční podmínky platné pro $n = 1$ dosáhneme při $i = n - 1$

$$\begin{aligned} M(n) &= 2^{n-1} M(n - (n-1)) + 2^{n-1} - 1 \\ &= 2^{n-1} M(1) + 2^{n-1} - 1 \\ &= 2^{n-1} \cdot 1 + 2^{n-1} - 1 = 2 \cdot 2^{n-1} - 1 = 2^{n-1+1} - 1 \\ &= 2^n - 1 \end{aligned}$$

312/776

Hanojské věže (Tower of Hanoi) (pokrač.)

6. Závěr:

- 6.1 Navržený rekurzivní algoritmus provede **exponenciální počet** základních operací vzhledem k velikosti vstupu.
- 6.2 Algoritmus je použitelný jen pro malá n , což **není způsobeno nevhodným návrhem**. Je to způsobeno **podstatou problému** – lze dokázat, že toto je nejlepší možný algoritmus.

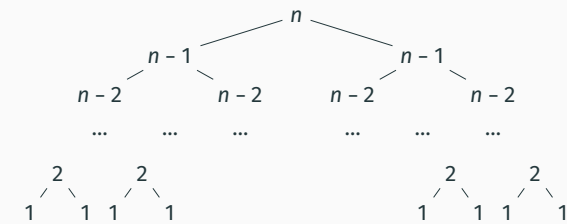
Opatrně s rekurzivními algoritmy

Obecně je nutné k rekurzivním algoritmům přistupovat velice opatrně, protože jejich stručnost může maskovat neefektivitu.

313/776

Hanojské věže (Tower of Hanoi) (pokrač.)

Vizualizace rekurzivního volání



Počet uzlů odpovídá počtu volání rekurzivní funkce

$$C(n) = \sum_{l=0}^{n-1} 2^l = 2^n - 1,$$

kde l je číslo úrovně ve stromu.

314/776

Zdroje pro samostatné studium

- Kniha [2], kapitola 2.4, strany 70 – 79

315/776

Děkuji za pozornost

Strategie řešení problémů hrubou silou a úplným prohledáváním

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



316/776

Strategie řešení problémů hrubou silou

Charakteristika

Strategie řešení problémů hrubou silou je založena na **přímočarém přístupu k řešení problému**, kdy algoritmus řešení vychází přímo ze zadání problému a pojmů obsažených v zadání.

317/776

Strategie řešení problémů hrubou silou – příklady

Výpočet mocniny

Máme vypočítat mocninu a^n pro nenulové a a n přirozené číslo. Z definice mocnění platí

$$a^n = \underbrace{a \times a \times \dots \times a}_{n \text{ krát}}$$

Řešení hrubou silou – vynásobíme číslo a mezi sebou $n - 1$ krát.

Další příklady řešení hrubou silou:

- výpočet NSD postupným dělením a
- násobení matic podle algoritmu z předchozí prezentace.

318/776

Strategie řešení problémů hrubou silou

1. obecná strategie – je obtížné najít problém, kde by „nezabrala“,
2. obecně sice nevede k efektivním algoritmům, ale pro některé problémy, např. násobení matic, pattern matching, jsou algoritmy založené na této strategii použitelné i pro větší vstupy,
3. přijatelná strategie v případě, kdy se nevyplatí zabývat se sofistikovanějším algoritmem – ad hoc řešení problému,
4. vždy použitelná strategie pro řešení problémů s malou velikostí vstupu a
5. význam i jako měřítko, kterým můžeme poměřovat efektivnější algoritmy řešící stejný problém.

319/776

Strategie řešení problémů hrubou silou a úplným prohledáváním

Třídící algoritmy

Třídící algoritmy

- Máme dáno pole n prvků pro které je definována relace uspořádání (čili vztah „menší než“), pro ukázkou vezměme celé čísla.
- Úkolem je přeuspořádat prvky pole do neklesající posloupnosti – prvek pole na nižším indexu musí být menší nebo roven prvku na vyšším indexu.
- Otázka zní: existuje třídící algoritmus řešící problém hrubou silou, zcela přímočaře?

320/776

Třídění výběrem – SelectSort

Výchozí úvaha

Otázka: O kterém prvku z pole víme přesně kam patří?

Odpověď: O nejmenším! Patří na začátek pole, na nejnižší index! (Pozn. Totéž platí pro největší prvek.)

Princip algoritmu

1. Vybereme z n prvků pole nejmenší prvek a zaměníme jej s prvním prvkem pole.
2. Vybereme ze zbylých $n - 1$ prvků pole nejmenší prvek a zaměníme jej s druhým prvkem pole.
3. Obecně v i -tém kroku vybereme ze zbylých $n - i$ nejmenší prvek a zaměníme ho s i -tým prvkem.
4. Po $n - 1$ krocích je pole setříděno.

321/776

Třídění výběrem – ukáзка

89	45	68	90	29	34	17
17	45	68	90	29	34	89
17	29	68	90	45	34	89
17	29	34	90	45	68	89
17	29	34	45	90	68	89
17	29	34	45	68	89	90

322/776

Třídění výběrem – pseudokód

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním na prvcích pole

Výstup: Setříděné pole A

```
1 for  $i \leftarrow 0$  to  $n - 2$  do
2    $min \leftarrow i$ ;
3   for  $j \leftarrow i + 1$  to  $n - 1$  do
4     if  $A[j] < A[min]$  then
5        $min \leftarrow j$ ;
6     end
7   end
8    $Swap(A[i], A[min])$ ;
9 end
```

323/776

Funkce pro výměnu hodnot dvou proměnných

1 Procedure $Swap(x, y)$

Vstup : Parametry x a y shodného datového typu

Výstup: Navzájem přehozené hodnoty x a y

```
2    $aux \leftarrow x$ ;
3    $x \leftarrow y$ ;
4    $y \leftarrow aux$ ;
5 end
```

324/776

Třídění výběrem – analýza

1. Velikost vstupu – počet prvků n .
2. Základní operace – porovnání prvků (někdy i počet výměn prvků).
3. Počet základních operací závisí pouze na n – nejhorší, nejlepší a průměrný případ splývají.
4. Sestavení vztahů

$$\begin{aligned} C(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] \\ &= \sum_{i=0}^{n-2} (n-1-i) = \frac{1}{2}n(n-1) \approx \frac{1}{2}n^2 = \Theta(n^2) \end{aligned}$$

325/776

Třídění výběrem – analýza (pokrač.)

Poznámky

- Podrobný výpočet sumy lze najít v příkladu „Unikátnost prvků“ v minulé lekci.
- Složitost algoritmu nijak nezávisí na míře neseříděnosti vstupního pole. Algoritmus tedy **není přirozený**.
- Algoritmus je ale **stabilní** – jako minimum je vždy brán první prvek z několika shodných.
- Algoritmus je **in situ** – vystačíme si s konstantním rozsahem paměti navíc.

326/776

Třídění výběrem

Animace

K algoritmu třídění výběrem je k dispozici animace v samostatném souboru.

327/776

Bublinové třídění – BubbleSort

- Vezmu dvojici sousedních prvků A_i a A_{i+1} .
- Pokud jsou v nesprávném pořadí, tak je vyměním.
- Zvýším i o jedna a pokračuji další dvojicí prvků.
- Po každém průchodu polem A se dostane jeden prvek určitě na své místo.

$$A_0, \dots, A_j \leftrightarrow A_{j+1}, \dots, A_{n-i-1} \mid \underbrace{A_{n-i} \leq \dots \leq A_{n-1}}_{\text{setříděno}}$$

V dalším průchodu polem už procházím o jeden prvek méně.

- Proč bublinové třídění – největší prvek z nesetříděné části takto „probublá“ směrem ke konci pole.

328/776

Bublinové třídění – ukázka

89	↔	45		68		90		29		34		17
45		89	↔	68		90		29		34		17
45		68		89	↔	90		29		34		17
45		68		89		90	↔	29		34		17
45		68		89		29		90	↔	34		17
45		68		89		29		34		90	↔	17
45		68		89		29		34		17		90
45	↔	68	↔	89	↔	29		34		17		90
45		68		29		89	↔	34		17		90
45		68		29		34		89	↔	17		90
45		68		29		34		17		89		90

a tak dále

329/776

Bublinové třídění – pseudokód

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním na prvcích pole

Výstup: Setříděné pole A

```
1 for  $i \leftarrow 0$  to  $n - 2$  do
2   for  $j \leftarrow 0$  to  $n - i - 2$  do
3     if  $A[j] > A[j + 1]$  then
4       |  $Swap(A[j], A[j + 1]);$ 
5     end
6   end
7 end
```

330/776

Bublinové třídění – analýza

1. Velikost vstupu – počet prvků n .
2. Základní operace – porovnání prvků.
3. Počet základních operací závisí pouze na n – nejhorší, nejlepší a průměrný případ splývají.
4. Sestavení vztahů

$$\begin{aligned} C(n) &= \sum_{i=0}^{n-2} \sum_{j=0}^{n-i-2} 1 = \sum_{i=0}^{n-2} [(n-i-2) - 0 + 1] \\ &= \sum_{i=0}^{n-2} (n-1-i) = \frac{1}{2}n(n-1) \approx \frac{1}{2}n^2 = \Theta(n^2) \end{aligned}$$

331/776

Bublinové třídění – analýza (pokrač.)

Počet základních operací $C(n)$ je shodný s tříděním výběrem.

Počet vzájemných výměn prvků $S(n)$ ale už na vstupu závisí a je v nejhorším případě roven

$$S_{\text{worst}}(n) = C(n) = \frac{1}{2}n(n-1) \in \Theta(n^2)$$

332/776

Bublinové třídění – další vlastnosti

- Algoritmus je **in situ** – vystačíme si s konstantním rozsahem paměti navíc.
- Algoritmus je **stabilní** – k výměně dojde jen pokud je $A[j] > A[j+1]$, jinak ne.
- Složitost této verze algoritmu nijak nezávisí na míře neseříděnosti vstupního pole. Algoritmus tedy **není přirozený**.
- Modifikované verze (viz dále) už ale přirozené jsou.

333/776

Bublinové třídění – alternativní konstrukce algoritmu I

- Počet opakování vnějšího cyklu by měl záviset na provedení či neprovedení výměny prvků.
- Pokud se během vnitřního cyklu neprovedla žádná výměna prvků není nutné toto pole dále procházet – pole je evidentně seříděno.
- Zavedeme si logickou proměnnou **AnySwap**, kam si budeme ukládat informaci o případné výměně.

334/776

Bublinové třídění – alternativní konstrukce algoritmu I

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním na prvcích pole

Výstup: Setříděné pole A

```
1 do
2   AnySwap ← false;
3   for  $i \leftarrow 0$  to  $n - 2$  do
4     if  $A[i] > A[i + 1]$  then
5       Swap ( $A[i], A[i + 1]$ );
6       AnySwap ← true;
7     end
8   end
9 while AnySwap;
```

335/776

Bublinové třídění – alternativní konstrukce algoritmu II

- Kombinovaná konstrukce algoritmu:
 - nedošlo k výměně prvků a
 - zkracování posloupnosti o jeden prvek zprava.
- Zavedeme si logickou proměnnou **AnySwap**, kam si budeme ukládat informaci o případné výměně.
- Proměnná **Right** bude označovat pravý okraj tříděné posloupnosti.

336/776

Bublinové třídění – alternativní konstrukce algoritmu II

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním na prvcích pole

Výstup: Setříděné pole A

```
1 Right ←  $n - 2$ ;
2 do
3   AnySwap ← false;
4   for  $i \leftarrow 0$  to Right do
5     if  $A[i] > A[i + 1]$  then
6       Swap ( $A[i], A[i + 1]$ );
7       AnySwap ← true;
8     end
9   end
10  Right ← Right - 1;
11 while AnySwap;
```

337/776

Bublinové třídění – alternativní konstrukce algoritmu III

Proměnná **LastSwapIndex**

- bude označovat index poslední výměny prvků,
- jinak řečeno pravý okraj tříděné posloupnosti v následujícím průchodu polem.

338/776

Bublinové třídění – alternativní konstrukce algoritmu III

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním na prvcích pole

Výstup: Setříděné pole A

```
1 Right ← n - 2;
2 do
3   LastSwapIndex ← 0;
4   for i ← 0 to Right do
5     if  $A[i] > A[i + 1]$  then
6       Swap ( $A[i], A[i + 1]$ );
7       LastSwapIndex ← i + 1;
8     end
9   end
10  Right ← LastSwapIndex;
11 while LastSwapIndex > 0;
```

339/776

Bublinové třídění

Animace

K algoritmu bublinového třídění je k dispozici animace v samostatném souboru. V animaci je znázorněn algoritmus podle snímku 339.

340/776

Třídění přetřásáním – ShakerSort

U BubbleSortu lze pozorovat dva efekty:

1. „velké“ prvky, **zajíci**, se velice rychle posunují ke konci pole, ale
2. „malé“ prvky, **želvy**, se posunují na začátek pole jen jako důsledek rychlého přesunu velkých prvků.

Modifikace BubbleSortu:

- pole budeme procházet střídavě z obou stran
- po každém průchodu pole se vymění role zajíců a želv, cimrmanovská „výměna mečů“ ze hry Blaník
- **ShakerSort** – pole je „přetřásáno, setřepáváno“ jako v barovém shakeru.

341/776

Třídění přetřásáním – ukázka

89	↔	45	68	90	29	34	17					
45		89	↔	68	90	29	34	17				
45		68		89	↔	90	29	34	17			
45		68		89		90	↔	29	34	17		
45		68		89		29		90	↔	34	17	
45		68		89		29		34		90	↔	17
45		68		89		29		34		17		90
45		68		89		29		34	↔	17		90
45		68		89		29	↔	17		34		90
45		68		89	↔	17		29		34		90
45		68	↔	17		89		29		34		90
45	↔	17		68		89		29		34		90
17		45		68		89		29		34		90

a tak dále

342/776

Třídění přetřásáním – pseudokód

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním
na prvcích pole

Výstup: Setříděné pole A

```
1  $Left \leftarrow 0$ ;  
2  $Right \leftarrow n - 1$ ;  
3 do  
4    $LeftToRight(A, Left, Right)$ ;  
5    $RightToLeft(A, Left, Right)$ ;  
6 while  $Left < Right$ ;
```

343/776

Třídění přetřásáním – pseudokód (pokrač.)

```
1 Procedure  $LeftToRight(A, Left, Right)$   
2    $j \leftarrow 0$ ;  
3   for  $i \leftarrow Left$  to  $Right - 1$  do  
4     if  $A[i] > A[i + 1]$  then  
5        $Swap(A[i], A[i + 1])$ ;  
6        $j \leftarrow i$ ;  
7     end  
8   end  
9    $Right \leftarrow j$ ;  
10 end
```

344/776

Třídění přetřásáním – pseudokód (pokrač.)

```
1 Procedure  $RightToLeft(A, Left, Right)$   
2    $j \leftarrow 0$ ;  
3   for  $i \leftarrow Right$  downto  $Left + 1$  do  
4     if  $A[i - 1] > A[i]$  then  
5        $Swap(A[i - 1], A[i])$ ;  
6        $j \leftarrow i$ ;  
7     end  
8   end  
9    $Left \leftarrow j$ ;  
10 end
```

345/776

Třídění přetřásáním

Animace

K algoritmu třídění přetřásáním je k dispozici animace
v samostatném souboru.

346/776

Strategie řešení problémů hrubou silou a úplným prohledáváním

Sekvenční vyhledávání

Sekvenční vyhledávání

- Typická ukázka strategie řešení hrubou silou.
- Silná stránka algoritmu – jednoduchost (simplicity).
- Slabá stránka algoritmu – vysoká složitost.

Vstup : Pole $A[0 \dots n - 1]$ a hledaný prvek x

Výstup: Index prvního výskytu prvku x v poli A , jinak -1

```
1 for  $i \leftarrow 0$  to  $n - 1$  do
2   | if  $A[i] = x$  then
3     |   return  $i$ ;
4   | end
5 end
6 return -1;
```

Algoritmus bývá také označován jako **lineární vyhledávání**.

347/776

Lineární vyhledávání – složitost

	Počet porovnání
Nejhorší případ	n
Nejlepší případ	1
Průměrný případ	$\frac{1}{2}p(n + 1) + n(1 - p)$

kde p je pravděpodobnost úspěšného vyhledání

348/776

Lineární vyhledávání – využití zářáčky (angl. sentinel)

Vstup : Pole $A[0 \dots n]$ a hledaný prvek x

Výstup: Index prvního výskytu prvku x v poli A , jinak -1

```
1  $A[n] \leftarrow x$ ;
2  $i \leftarrow 0$ ;
3 while  $A[i] \neq x$  do
4   |  $i \leftarrow i + 1$ ;
5 end
6 if  $i < n$  then return  $i$ ;
7 return -1;
```

349/776

Strategie řešení problémů hrubou silou a úplným prohledáváním

Vyhledávání podřetězce hrubou silou

Vyhledávání podřetězce hrubou silou

Zadání – najít vzorek p v textu t .

Řešení hrubou silou

1. Přiložíme vzorek na začátek textu.
2. Začneme porovnávat znaky ve vzorku a textu.
3. Pokud se shodují všechny znaky vzorku s textem – nalezeno.
4. Pokud nalezneme neshodu, posuneme vzorek o jednu pozici dopředu a pokračujeme bodem 2

$$\begin{array}{cccccccc} t_0 & \cdots & t_i & \cdots & t_{i+j} & \cdots & t_{i+m-1} & \cdots & t_{n-1} \\ & & \downarrow & & \downarrow & & \downarrow & & \\ & & p_0 & \cdots & p_j & \cdots & p_{m-1} & & \end{array}$$

350/776

Vyhledávání hrubou silou – pseudokód

Vstup : Vzorek p , text t a počáteční pozice s

Výstup: Pozice prvního výskytu p v textu t nebo -1

```
1 for  $i \leftarrow s$  to  $|t| - |p|$  do
2    $j \leftarrow 0$ ;
3   while  $j < |p|$  do
4     if  $p[j] \neq t[i + j]$  then break;
5      $j \leftarrow j + 1$ ;
6   end
7   if  $j = |p|$  then return  $i$ ;
8 end
9 return -1;
```

351/776

Vyhledávání hrubou silou – příklad

První pokus

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G	
1	2	3	4																					
G	C	A	G	A	G	A	G																	

Posun o 1 znak

Druhý pokus

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
1																							
G	C	A	G	A	G	A	G																

Posun o 1 znak

352/776

Vyhledávání hrubou silou – příklad (pokrač.)

Třetí pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
      1
G C A G A G A G
```

Posun o 1 znak

Čtvrtý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
      1
G C A G A G A G
```

Posun o 1 znak

353/776

Vyhledávání hrubou silou – příklad (pokrač.)

Pátý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
      1
G C A G A G A G
```

Posun o 1 znak

Šestý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
           1 2 3 4 5 6 7 8
G C A G A G A G
```

Posun o 1 znak

354/776

Vyhledávání hrubou silou – příklad (pokrač.)

Sedmý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
      1
G C A G A G A G
```

Posun o 1 znak

Osmý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
      1
G C A G A G A G
```

Posun o 1 znak

355/776

Vyhledávání hrubou silou – příklad (pokrač.)

Devátý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1 2
G C A G A G A G
```

Posun o 1 znak

Desátý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1
G C A G A G A G
```

Posun o 1 znak

356/776

Vyhledávání hrubou silou – příklad (pokrač.)

Jedenáctý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1 2
                G C A G A G A G
```

Posun o 1 znak

Dvanáctý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1
                G C A G A G A G
```

Posun o 1 znak

357/776

Vyhledávání hrubou silou – příklad (pokrač.)

Třináctý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1 2
                G C A G A G A G
```

Posun o 1 znak

Čtrnáctý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1
                G C A G A G A G
```

Posun o 1 znak

358/776

Vyhledávání hrubou silou – příklad (pokrač.)

Patnáctý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1
                G C A G A G A G
```

Posun o 1 znak

Šestnáctý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1
                G C A G A G A G
```

Posun o 1 znak

359/776

Vyhledávání hrubou silou – příklad (pokrač.)

Sedmnáctý pokus

```
G C A T C G C A G A G A G T A T A C A G T A C G
                1
                G C A G A G A G
```

Posun o 1 znak

Algoritmus provedl celkem 30 porovnání znaků.

360/776

Vyhledávání podřetězce hrubou silou – složitost algoritmu

- Velikost vstupu – délka textu n a délka vzorku m .
- Základní operace – porovnání znaku vzorku a textu.
- Závislost jen na velikosti vstupu – ne, záleží i na tom, kdy se najde první neshoda.
- Nejhorší případ – text $a^{n-1}b$, vzorek $a^{m-1}b$
 - v každém pokusu provedeme všech m porovnání vzorku s textem
 - současně provedeme všech $n - m + 1$ pokusů.
 - Celkem provedeme $m(n - m + 1)$ porovnání, algoritmus spadá do $O(mn)$.
- Nejlepší případ – vzorek je nalezen na začátku textu, složitost $O(m)$.
- Přirozené jazyky – posun nastane po „několika“ (k_L) porovnáních, nejhorší složitost $O(k_L n) = O(n)$.

361/776

Strategie řešení problémů hrubou silou a úplným prohledáváním

Problém nejbližší dvojice bodů

Problém nejbližší dvojice bodů

Zadání problému

Nalezněte dva navzájem nejbližší body z množiny n bodů.

- Jde o jeden z problémů **výpočetní geometrie**.
- Body mohou ležet na rovině nebo obecně v nějakém mnohodimenzionálním prostoru.
- Body mohou reprezentovat objekty reálného světa nebo záznamy v databázi, texty...
- Příklady aplikací:
 - Bezpečnost letového provozu – hledáme dvě nejbližší letadla ve vzdušném prostoru.
 - Shlukování – hierarchické shlukovací algoritmy postupně spojují sobě nejbližší shluky do jednoho, většího shluku.

362/776

Problém nejbližší dvojice bodů – předpoklady

Předpokládejme množinu n bodů $\{P_1, \dots, P_n\}$, každému bodu P_i odpovídá vektor \vec{p}_i se složkami

$$\vec{p}_i = (x_i, y_i)$$

v obvyklých kartézských souřadnicích.

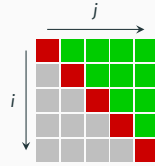
Vzdálenost bodů \vec{p}_i a \vec{p}_j budeme počítat pomocí Euklidovské vzdálenosti

$$d(\vec{p}_i, \vec{p}_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

363/776

Problém nejbližší dvojice bodů – řešení hrubou silou

- Vypočteme vzdálenost všech dvojic bodů \vec{p}_i a \vec{p}_j a nalezneme minimum.
- Stačí počítat jen dvojice bodů pro $j = i + 1, \dots, n$.



Vstup : Množina bodů $\{\vec{p}_1, \vec{p}_2, \dots, \vec{p}_n\}$

Výstup: Vzdálenost dvou nejbližších bodů

```
1 MinDist ← ∞;
2 for i ← 1 to n - 1 do
3   for j ← i + 1 to n do
4     | MinDist ← min(MinDist,  $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ );
5   end
6 end
7 return MinDist;
```

364/776

Problém nejbližší dvojice bodů – řešení hrubou silou, složitost

1. Velikost vstupu – počet bodů n
2. Základní operace
 - Výpočet odmocniny – nejde o triviální záležitost¹.
 - Výpočtu odmocniny se lze vyhnout – jde o rostoucí funkci, lze hledat minimum „čtverců“ vzdáleností.
 - Za základní operaci vezmeme umocňování rozdílů souřadnic.
3. Počet základních operací závisí pouze na n – nejhorší, nejlepší a průměrný případ splývají.

365/776

Problém nejbližší dvojice bodů – řešení hrubou silou, složitost (pokrač.)

4. Sestavení vztahů

$$\begin{aligned} C(n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n 2 = 2 \sum_{i=1}^{n-1} (n-i) \\ &= 2[(n-1) + (n-2) + \dots + 1] = (n-1)n \in \Theta(n^2) \end{aligned}$$

5. Odstranění odmocniny – snížení složitosti o konstantní faktor, nedošlo k zlepšení asymptotické složitosti, stále je to $\Theta(n^2)$ algoritmus.
6. Později si ukážeme algoritmus s lineárně logaritmickou složitostí.

¹https://en.wikipedia.org/wiki/Methods_of_computing_square_roots

366/776

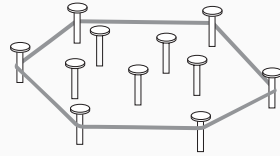
Strategie řešení problémů hrubou silou a úplným prohledáváním

Konvexní obal množiny

Konvexní obal – Convex Hull

Zadání problému

Úkolem je najít konvexní obal (obálku) množiny bodů v prostoru.



- Jde o jeden z problémů **výpočetní geometrie**.
- Body mohou ležet na rovině nebo obecně v nějakém mnohodomenzionálním prostoru.
- Příklady aplikací:
 - Detekce kolizí – počítačová grafika, autonomní vozidla,
 - GIS – bodové senzory, vytvoření oblasti z těchto dat,
 - Optimalizační úlohy – vrcholy konvexního obalu jsou jistým způsobem extrémní; konvexní mnohoúhelník vzniká jako průnik konečného množství polorovin; polorovina je definována nerovnicí...

367/776

Konvexní množina bodů

Definice

Množina bodů M v rovině se nazývá **konvexní**, jestliže pro libovolnou dvojici bodů $p, q \in M$ úsečka spojující body p a q náleží do množiny M .

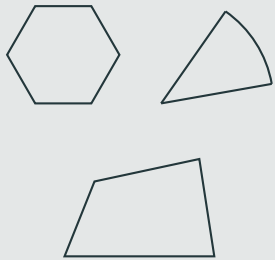
Množina, které není konvexní se nazývá **nekonvexní**.

Představíme-li si hranici množiny jako neprůhlednou, znamená konvexita množiny názorně to, že z každého jejího bodu je vidět každý její bod.

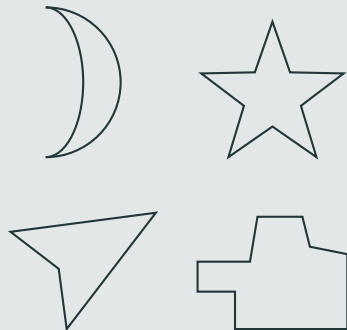
368/776

Konvexní množina bodů – příklady

Konvexní útvary



Nekonvexní útvary



369/776

Konvexní obal

Definice

Konvexním obalem množiny bodů M nazýváme nejmenší konvexní množinu, která obsahuje M .

Výraz „nejmenší“ znamená, že konvexní obal množiny M musí být podmnožinou jakékoliv jiné konvexní podmnožiny obsahující množinu M .

Konvexní obal

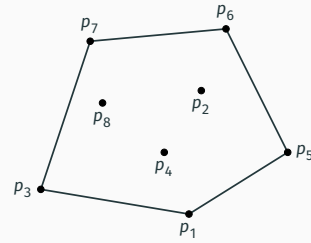
- dvouprvkové množiny – úsečka spojující oba body.
- tříprvkové množiny – trojúhelník, pokud body neleží na přímce, jinak je to úsečka spojující nejbližší body.

370/776

Konvexní obal (pokrač.)

Věta

Konvexní obal množiny bodů M s více než dvěma body, které neleží na jedné přímce, je konvexní mnohoúhelník, jehož vrcholy náležejí do M .



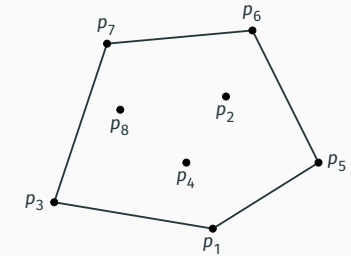
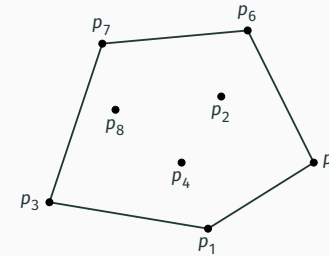
Body množiny M , které specifikují konvexní obal M se nazývají extrémní body.

Problém nalezení konvexního obalu množiny M redukuje se na nalezení extrémních bodů.

371/776

Konvexní obal – řešení hrubou silou

Úsečka $\vec{p}_i\vec{p}_j$ náleží do konvexního obalu množiny M , právě když všechny ostatní body z M leží v jedné z polorovin definovaných přímkou $\vec{p}_i\vec{p}_j$.



372/776

Konvexní obal – řešení hrubou silou (pokrač.)

Obecnou rovnici přímky procházející body \vec{p}_i a \vec{p}_j lze psát jako

$$ax + by + c = 0,$$

kde

$$a = y_j - y_i$$

$$b = x_i - x_j$$

$$c = y_i x_j - x_i y_j$$

373/776

Konvexní obal – řešení hrubou silou (pokrač.)

Přímka definuje dvě poloroviny:

$$ax + by + c < 0 \quad (14)$$

$$ax + by + c > 0 \quad (15)$$

Stačí tedy ověřit, zda pro zbývajících $n - 2$ bodů platí buď nerovnice (14) nebo (15).

374/776

- Musíme prověřit všech $\frac{1}{2}n(n-1)$ dvojic bodů a současně
- pro každou přímku definovanou jednou dvojicí bodů musíme ověřit platnost nerovnic (14) a (15) pro zbývajících $n-2$ bodů.
- Celkově tedy $\left[\frac{1}{2}n(n-1)\right](n-2) \in O(n^3)$.

375/776

Strategie řešení problémů hrubou silou a úplným prohledáváním

Úplné prohledávání

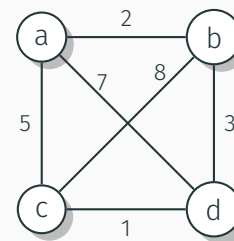
Úplné prohledávání (Exhaustive search)

- Součástí řešení mnoha problémů – nalezení **jednoho prvku**, s nějakou **specifickou vlastností**, z množiny prvků, tzv. domény, která **exponenciálně** nebo rychleji roste s velikostí problému.
- Hledaný element je typicky **kombinatorické povahy** – permutace, kombinace, podmnožina.
- Typicky jde o **optimalizační úlohy** – typicky hledáme maximum, minimum. Například minimalizujeme délku cesty, maximalizujeme zisk.

Úplné prohledávání je strategie řešení problému hrubou silou spočívající v otestování všech prvků uvažované domény.

376/776

Problém obchodního cestujícího – příklad



Trasa	Délka trasy l
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	$2 + 8 + 1 + 7 = 18$
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	$2 + 3 + 1 + 5 = 11$
$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$	$5 + 8 + 3 + 7 = 23$
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	$5 + 1 + 3 + 2 = 11$
$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$	$7 + 3 + 8 + 5 = 23$
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	$7 + 1 + 8 + 2 = 18$

377/776

Úplné prohledávání



378/776

Problém obchodního cestujícího (Traveling Salesman Problem)



379/776

Problém batohu



380/776

Strategie řešení problémů hrubou silou
a úplným prohledáváním

Průchody grafem

Průchod grafem do hloubky a do šířky



381/776

Algoritmus průchodu grafem do hloubky

```
Input : Graf  $G(V, E)$ , počáteční vrchol  $s \in V$ 
Output: DF-strom
1 Init( $V, s$ );
2 while stack  $\neq \emptyset$  do
3    $u \leftarrow \text{Top}(\text{stack})$ ;
4   switch state [ $u$ ] do
5     case discovered do
6       | ProcessDiscoveredVertex( $u$ );
7     end
8     case current do
9       | ProcessCurrentVertex( $u$ );
10    end
11  end
12 end
```

382/776

Algoritmus průchodu grafem do hloubky (pokrač.)

```
1 Procedure Init( $V, s$ )
   Input : Množina vrcholů  $V$ , počáteční vrchol  $s \in V$ 
2   foreach  $u \in V$  do
3     state [ $u$ ]  $\leftarrow$  unknown;
4     d [ $u$ ]  $\leftarrow$  f [ $u$ ]  $\leftarrow$   $\infty$ ;
5      $\pi$  [ $u$ ]  $\leftarrow$  nothing;
6   end
7   state [ $s$ ]  $\leftarrow$  discovered;
8   stack  $\leftarrow$   $\emptyset$ ;
9   Push(stack,  $s$ );
10  time  $\leftarrow$  0;
11 end
```

383/776

Algoritmus průchodu grafem do hloubky (pokrač.)

```
1 Procedure ProcessDiscoveredVertex( $u$ )
   Input : Vrchol  $u \in V$ 
2   state [ $u$ ]  $\leftarrow$  current;
3   d [ $u$ ]  $\leftarrow$  time  $\leftarrow$  time + 1;
4   foreach  $v \in \text{Adj}(G, u)$  do
5     if state [ $v$ ] = unknown then
6       | state [ $v$ ]  $\leftarrow$  discovered;
7       |  $\pi$  [ $v$ ]  $\leftarrow$   $u$ ;
8       | Push(stack,  $v$ );
9     end
10  end
11 end
```

384/776

Algoritmus průchodu grafem do hloubky (pokrač.)

```

1 Procedure ProcessCurrentVertex (u)
  Input : Vrchol  $u \in V$ 
2   state [u]  $\leftarrow$  finished;
3   f [u]  $\leftarrow$  time  $\leftarrow$  time + 1;
4   Pop(stack);
5 end
    
```

385/776

Animace průchodu grafem do hloubky – legenda

Vrcholy grafu:

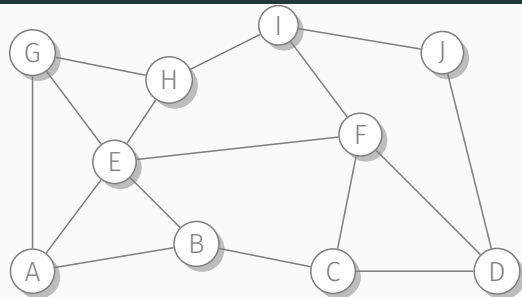
šedá vrchol ve stavu unknown
žlutá vrchol ve stavu discovered
červená vrchol ve stavu current
modrá vrchol ve stavu finished

Hrany grafu:

šedá hrana mezi vrcholy ve stavu unknown nebo hrana nepatřící do DF-stromu
žlutá hrana incidentní s vrcholy ve stavu discovered
červená hrana incidentní s vrcholem ve stavu current
modrá hrana mezi vrcholy ve stavu finished

386/776

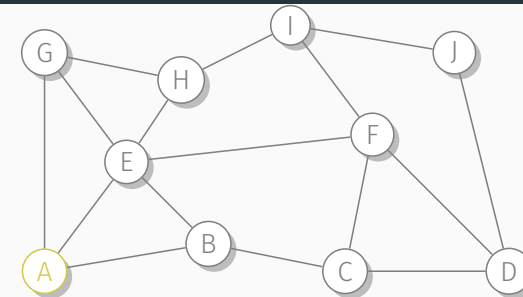
Průchod grafem do hloubky, krok 1



<i>v</i>	<i>d</i> [<i>v</i>]	<i>f</i> [<i>v</i>]	π [<i>v</i>]	<i>v</i>	<i>d</i> [<i>v</i>]	<i>f</i> [<i>v</i>]	π [<i>v</i>]
A	∞	∞	/	F	∞	∞	/
B	∞	∞	/	G	∞	∞	/
C	∞	∞	/	H	∞	∞	/
D	∞	∞	/	I	∞	∞	/
E	∞	∞	/	J	∞	∞	/

387/776

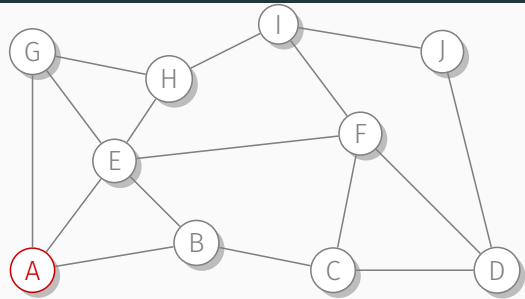
Průchod grafem do hloubky, krok 2



<i>v</i>	<i>d</i> [<i>v</i>]	<i>f</i> [<i>v</i>]	π [<i>v</i>]	<i>v</i>	<i>d</i> [<i>v</i>]	<i>f</i> [<i>v</i>]	π [<i>v</i>]
A	∞	∞	/	F	∞	∞	/
B	∞	∞	/	G	∞	∞	/
C	∞	∞	/	H	∞	∞	/
D	∞	∞	/	I	∞	∞	/
E	∞	∞	/	J	∞	∞	/

388/776

Průchod grafem do hloubky, krok 3

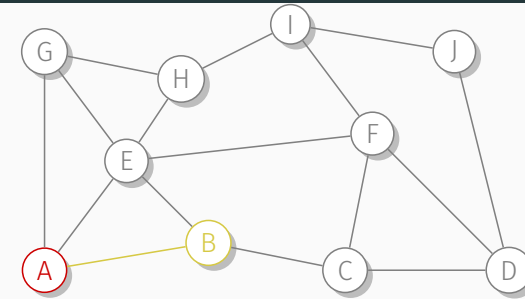


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	/	G	∞	∞	/
C	∞	∞	/	H	∞	∞	/
D	∞	∞	/	I	∞	∞	/
E	∞	∞	/	J	∞	∞	/

A
S

389/776

Průchod grafem do hloubky, krok 4

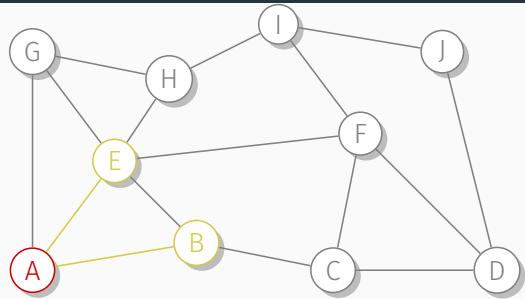


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	A	G	∞	∞	/
C	∞	∞	/	H	∞	∞	/
D	∞	∞	/	I	∞	∞	/
E	∞	∞	/	J	∞	∞	/

B
A
S

390/776

Průchod grafem do hloubky, krok 5

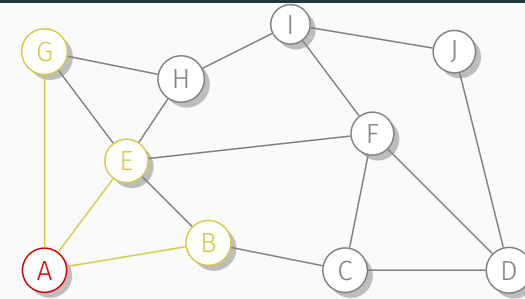


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	A	G	∞	∞	/
C	∞	∞	/	H	∞	∞	/
D	∞	∞	/	I	∞	∞	/
E	∞	∞	A	J	∞	∞	/

E
B
A
S

391/776

Průchod grafem do hloubky, krok 6

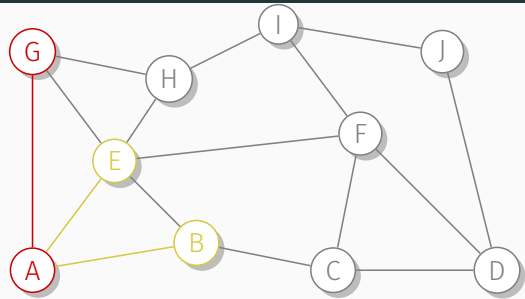


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	A	G	∞	∞	A
C	∞	∞	/	H	∞	∞	/
D	∞	∞	/	I	∞	∞	/
E	∞	∞	A	J	∞	∞	/

G
E
B
A
S

392/776

Průchod grafem do hloubky, krok 7

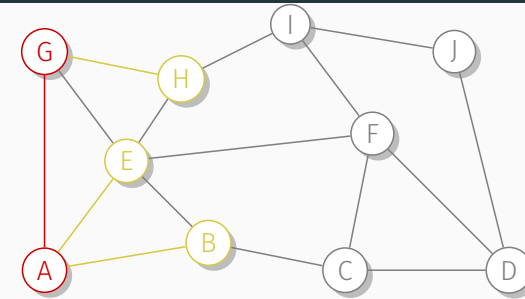


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	∞	∞	/
D	∞	∞	/	I	∞	∞	/
E	∞	∞	A	J	∞	∞	/



393/776

Průchod grafem do hloubky, krok 8

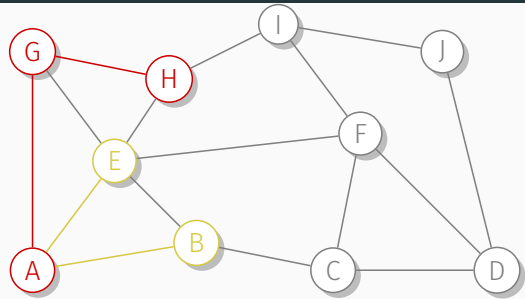


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	∞	∞	G
D	∞	∞	/	I	∞	∞	/
E	∞	∞	A	J	∞	∞	/

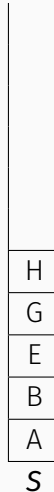


394/776

Průchod grafem do hloubky, krok 9

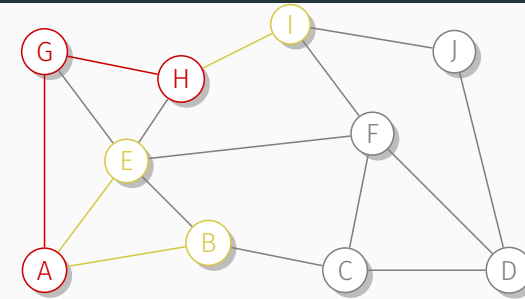


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	3	∞	G
D	∞	∞	/	I	∞	∞	/
E	∞	∞	A	J	∞	∞	/

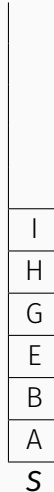


395/776

Průchod grafem do hloubky, krok 10

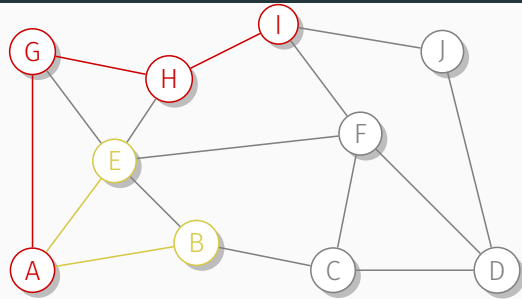


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	3	∞	G
D	∞	∞	/	I	∞	∞	H
E	∞	∞	A	J	∞	∞	/

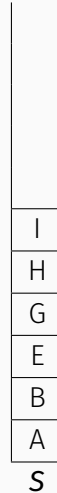


396/776

Průchod grafem do hloubky, krok 11

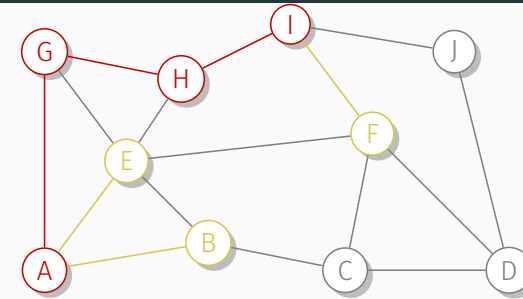


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	/
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	3	∞	G
D	∞	∞	/	I	4	∞	H
E	∞	∞	A	J	∞	∞	/

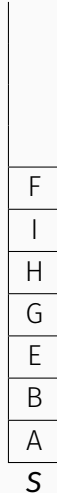


397/776

Průchod grafem do hloubky, krok 12

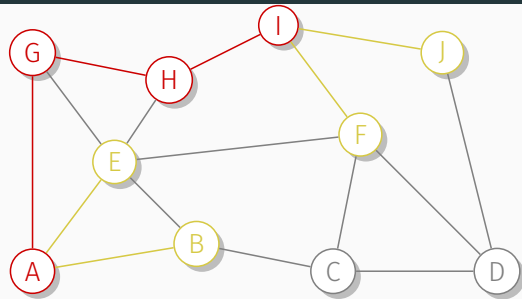


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	3	∞	G
D	∞	∞	/	I	4	∞	H
E	∞	∞	A	J	∞	∞	/

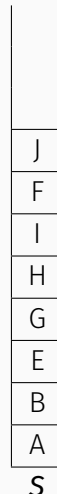


398/776

Průchod grafem do hloubky, krok 13

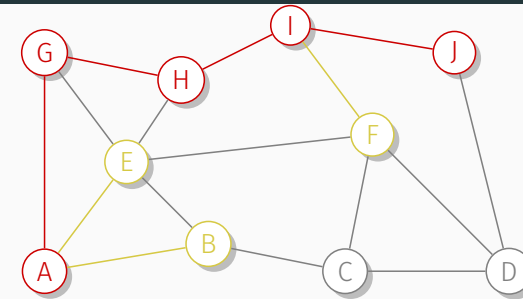


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	3	∞	G
D	∞	∞	/	I	4	∞	H
E	∞	∞	A	J	∞	∞	I



399/776

Průchod grafem do hloubky, krok 14

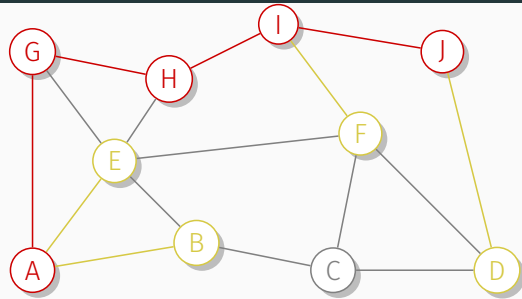


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	3	∞	G
D	∞	∞	/	I	4	∞	H
E	∞	∞	A	J	5	∞	I



400/776

Průchod grafem do hloubky, krok 15

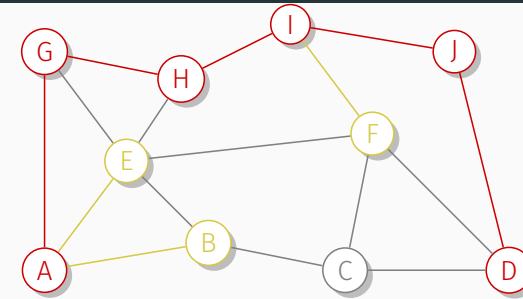


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	3	∞	G
D	∞	∞	J	I	4	∞	H
E	∞	∞	A	J	5	∞	I

D
J
F
I
H
G
E
B
A
S

401/776

Průchod grafem do hloubky, krok 16

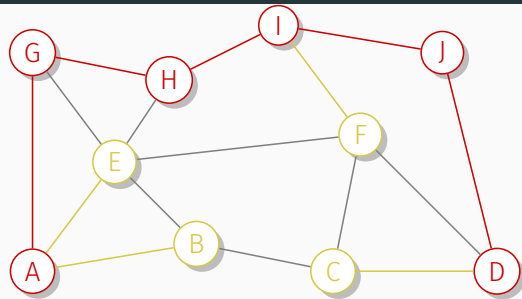


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	∞	∞	/	H	3	∞	G
D	6	∞	J	I	4	∞	H
E	∞	∞	A	J	5	∞	I

D
J
F
I
H
G
E
B
A
S

402/776

Průchod grafem do hloubky, krok 17

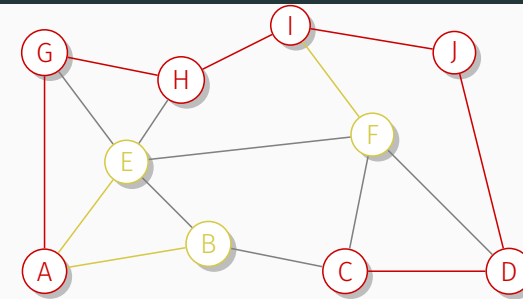


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	∞	∞	D	H	3	∞	G
D	6	∞	J	I	4	∞	H
E	∞	∞	A	J	5	∞	I

C
D
J
F
I
H
G
E
B
A
S

403/776

Průchod grafem do hloubky, krok 18

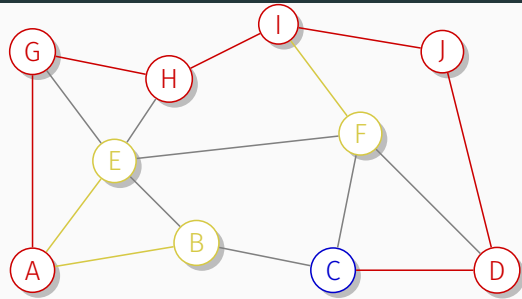


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	7	∞	D	H	3	∞	G
D	6	∞	J	I	4	∞	H
E	∞	∞	A	J	5	∞	I

C
D
J
F
I
H
G
E
B
A
S

404/776

Průchod grafem do hloubky, krok 19

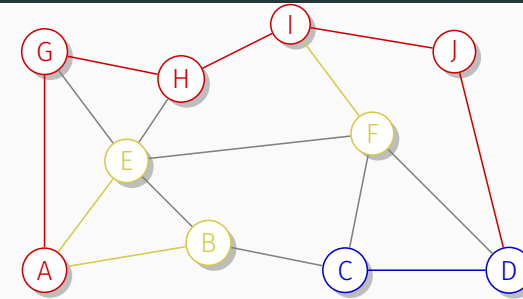


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	7	8	D	H	3	∞	G
D	6	∞	J	I	4	∞	H
E	∞	∞	A	J	5	∞	I

D
J
F
I
H
G
E
B
A
S

405/776

Průchod grafem do hloubky, krok 20

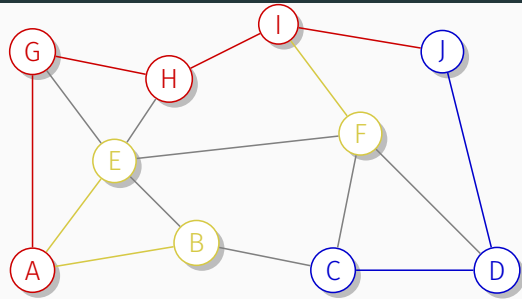


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	7	8	D	H	3	∞	G
D	6	9	J	I	4	∞	H
E	∞	∞	A	J	5	∞	I

J
F
I
H
G
E
B
A
S

406/776

Průchod grafem do hloubky, krok 21

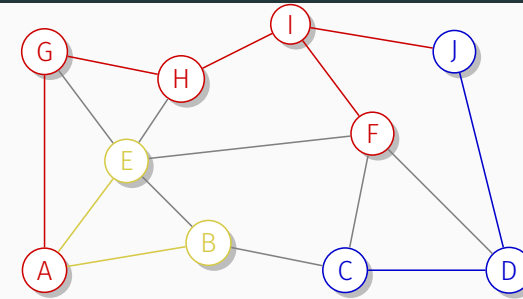


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	∞	∞	I
B	∞	∞	A	G	2	∞	A
C	7	8	D	H	3	∞	G
D	6	9	J	I	4	∞	H
E	∞	∞	A	J	5	10	I

F
I
H
G
E
B
A
S

407/776

Průchod grafem do hloubky, krok 22

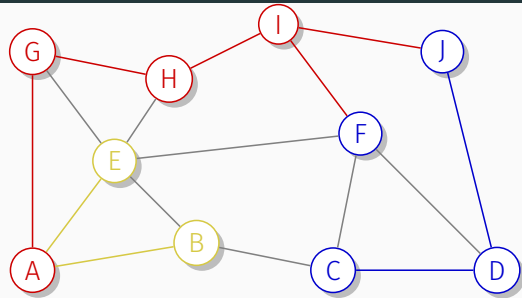


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	∞	I
B	∞	∞	A	G	2	∞	A
C	7	8	D	H	3	∞	G
D	6	9	J	I	4	∞	H
E	∞	∞	A	J	5	10	I

F
I
H
G
E
B
A
S

408/776

Průchod grafem do hloubky, krok 23

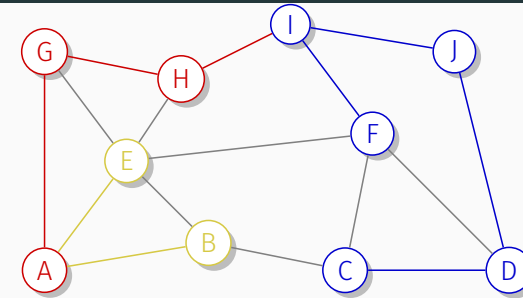


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	12	I
B	∞	∞	A	G	2	∞	A
C	7	8	D	H	3	∞	G
D	6	9	J	I	4	∞	H
E	∞	∞	A	J	5	10	I

I
H
G
E
B
A
S

409/776

Průchod grafem do hloubky, krok 24

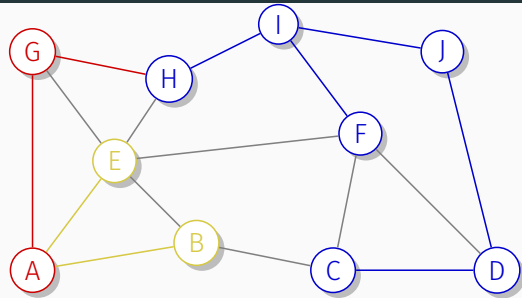


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	12	I
B	∞	∞	A	G	2	∞	A
C	7	8	D	H	3	∞	G
D	6	9	J	I	4	13	H
E	∞	∞	A	J	5	10	I

H
G
E
B
A
S

410/776

Průchod grafem do hloubky, krok 25

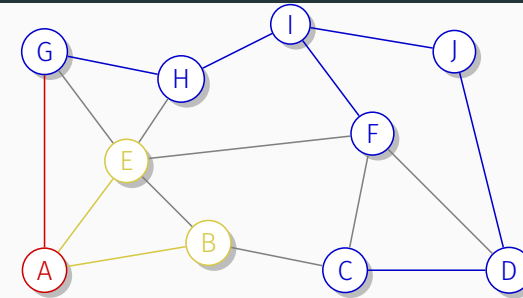


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	12	I
B	∞	∞	A	G	2	∞	A
C	7	8	D	H	3	14	G
D	6	9	J	I	4	13	H
E	∞	∞	A	J	5	10	I

G
E
B
A
S

411/776

Průchod grafem do hloubky, krok 26

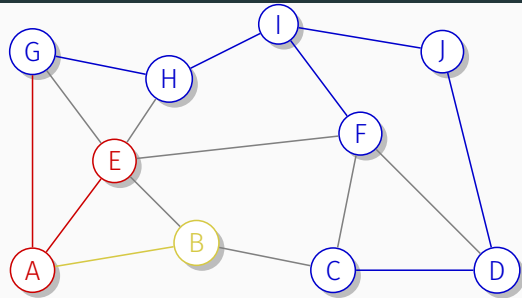


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	12	I
B	∞	∞	A	G	2	15	A
C	7	8	D	H	3	14	G
D	6	9	J	I	4	13	H
E	∞	∞	A	J	5	10	I

E
B
A
S

412/776

Průchod grafem do hloubky, krok 27

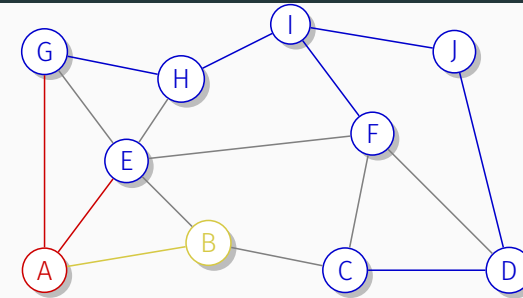


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	12	I
B	∞	∞	A	G	2	15	A
C	7	8	D	H	3	14	G
D	6	9	J	I	4	13	H
E	16	∞	A	J	5	10	I



413/776

Průchod grafem do hloubky, krok 28

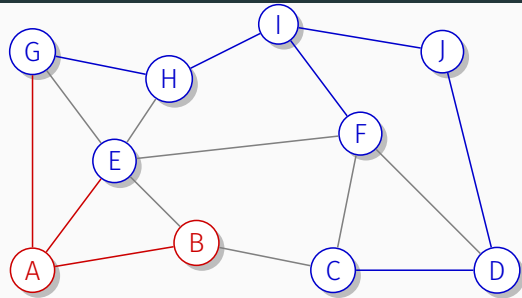


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	12	I
B	∞	∞	A	G	2	15	A
C	7	8	D	H	3	14	G
D	6	9	J	I	4	13	H
E	16	17	A	J	5	10	I



414/776

Průchod grafem do hloubky, krok 29

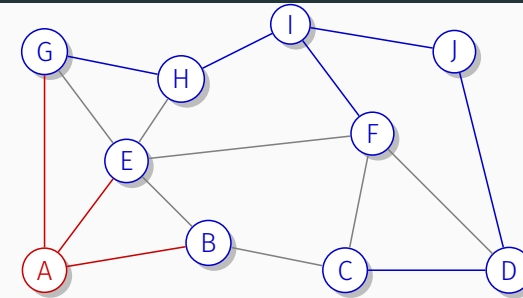


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	12	I
B	18	∞	A	G	2	15	A
C	7	8	D	H	3	14	G
D	6	9	J	I	4	13	H
E	16	17	A	J	5	10	I



415/776

Průchod grafem do hloubky, krok 30

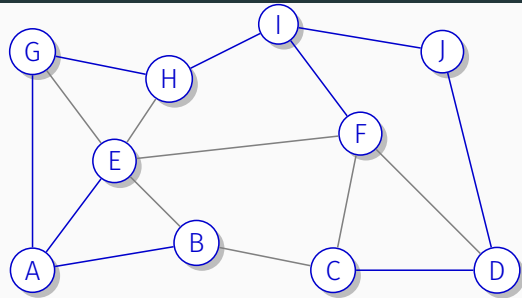


v	$d[v]$	$f[v]$	$\pi[v]$	v	$d[v]$	$f[v]$	$\pi[v]$
A	1	∞	/	F	11	12	I
B	18	19	A	G	2	15	A
C	7	8	D	H	3	14	G
D	6	9	J	I	4	13	H
E	16	17	A	J	5	10	I



416/776

Průchod grafem do hloubky, krok 31



v	d[v]	f[v]	$\pi[v]$	v	d[v]	f[v]	$\pi[v]$
A	1	20	/	F	11	12	I
B	18	19	A	G	2	15	A
C	7	8	D	H	3	14	G
D	6	9	J	I	4	13	H
E	16	17	A	J	5	10	I

417/776

Algoritmus průchodu grafem do šířky

Input : Graf $G(V, E)$, počáteční vrchol $s \in V$

Output: BF-strom

```

1 foreach  $u \in V \setminus \{s\}$  do
2   state [u]  $\leftarrow$  unknown;
3   d [u]  $\leftarrow$   $\infty$ ;
4    $\pi$  [u]  $\leftarrow$  nothing;
5 end
6 Q  $\leftarrow$   $\emptyset$ ;
7 state [s]  $\leftarrow$  discovered;
8 d [s]  $\leftarrow$  0;
9  $\pi$  [s]  $\leftarrow$  nothing;
10 Enqueue(Q, s);
    
```

418/776

Algoritmus průchodu grafem do šířky (pokrač.)

```

11 while Q  $\neq$   $\emptyset$  do
12    $u \leftarrow$  Dequeue(Q);
13   foreach  $v \in Adj(G, u)$  do
14     if state [v] = unknown then
15       state [v]  $\leftarrow$  discovered;
16       d [v]  $\leftarrow$  d [u] + 1;
17        $\pi$  [v]  $\leftarrow$  u;
18       Enqueue(Q, v);
19     end
20   end
21   state [u]  $\leftarrow$  finished;
22 end
    
```

419/776

Animace průchodu grafem do šířky – legenda

Vrcholy grafu

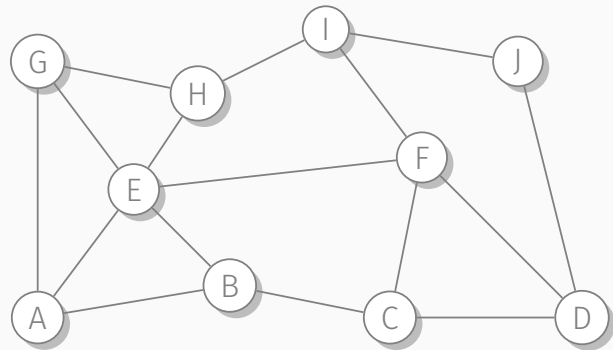
- šedá vrchol ve stavu unknown
- žlutá vrchol ve stavu discovered
- červená právě zpracovávaný vrchol
- modrá vrchol ve stavu finished

Hrany grafu

- šedá hrana mezi vrcholy ve stavu unknown nebo hrana nepatřící do BF-stromu
- žlutá hrana incidentní s vrcholy ve stavu discovered
- červená hrana incidentní s právě zpracovávaným vrcholem
- modrá hrana mezi vrcholy ve stavu finished

420/776

Průchod grafem do šířky, krok 1

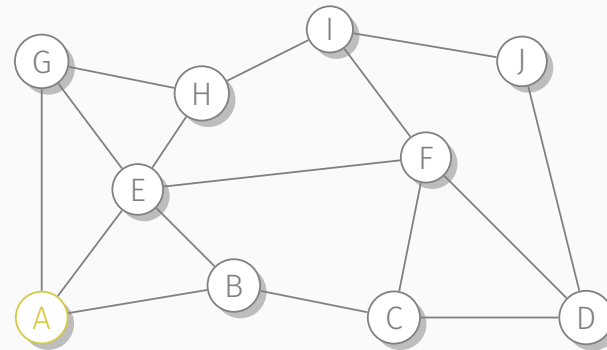


Q

v	d[v]	$\pi[v]$
A	∞	/
B	∞	/
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

421/776

Průchod grafem do šířky, krok 2

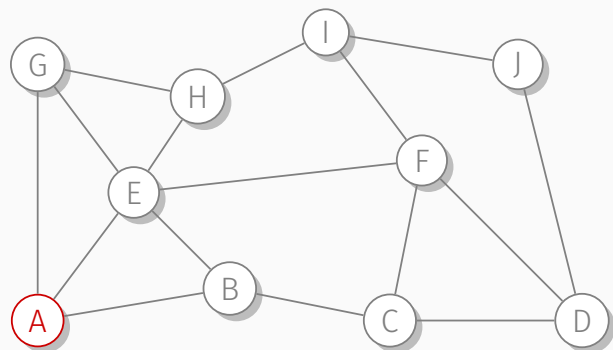


Q A

v	d[v]	$\pi[v]$
A	0	/
B	∞	/
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

422/776

Průchod grafem do šířky, krok 3

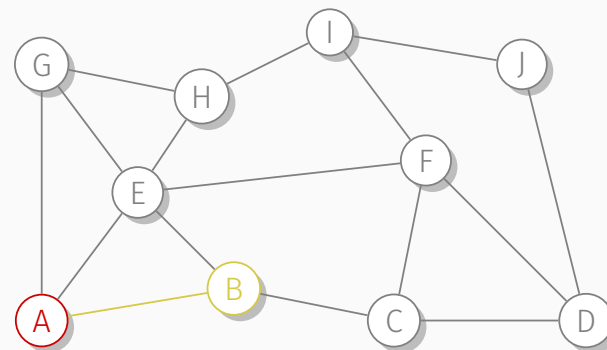


Q

v	d[v]	$\pi[v]$
A	0	/
B	∞	/
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

423/776

Průchod grafem do šířky, krok 4

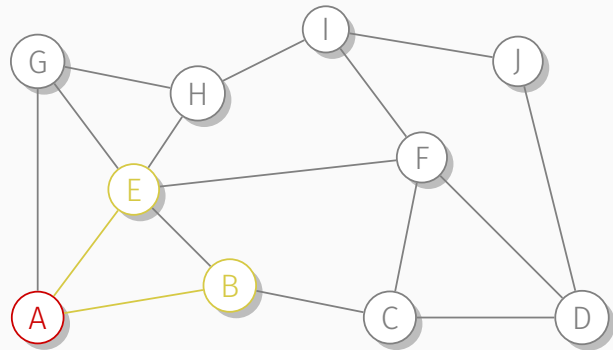


Q B

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	∞	/
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

424/776

Průchod grafem do šířky, krok 5

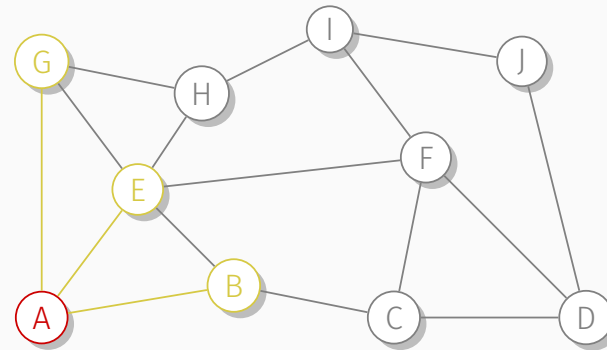


Q [B | E |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	1	A
F	∞	/
G	∞	/
H	∞	/
I	∞	/
J	∞	/

425/776

Průchod grafem do šířky, krok 6

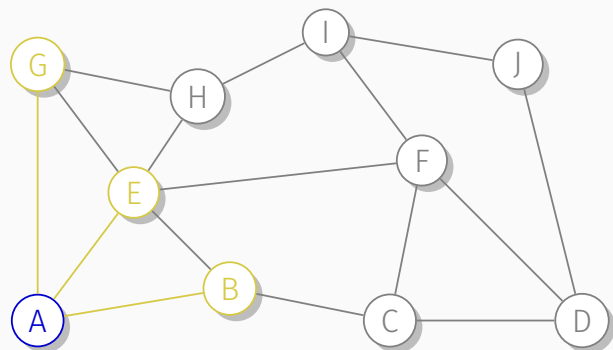


Q [B | E | G |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

426/776

Průchod grafem do šířky, krok 7

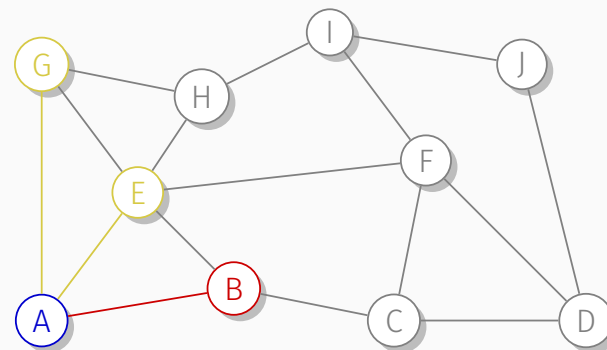


Q [B | E | G |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

427/776

Průchod grafem do šířky, krok 8

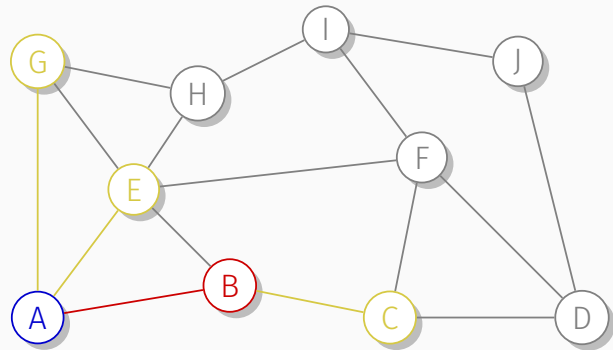


Q [E | G |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	∞	/
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

428/776

Průchod grafem do šířky, krok 9

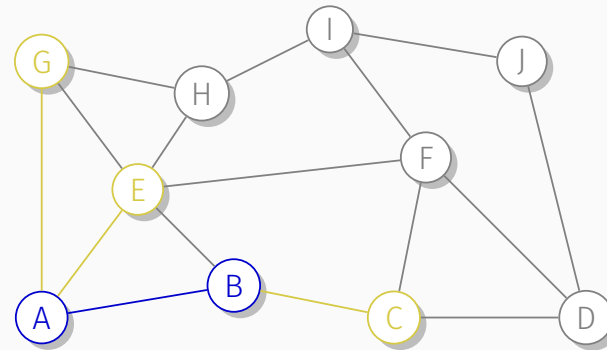


Q [E | G | C |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

429/776

Průchod grafem do šířky, krok 10

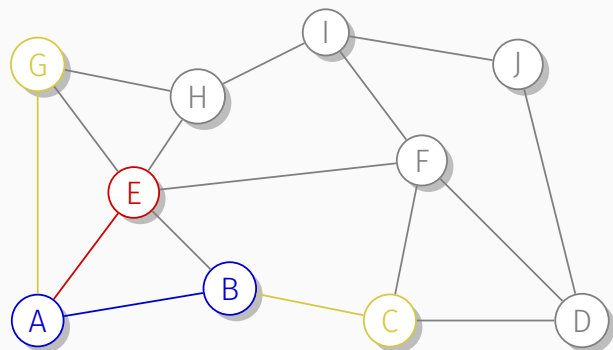


Q [E | G | C |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

430/776

Průchod grafem do šířky, krok 11

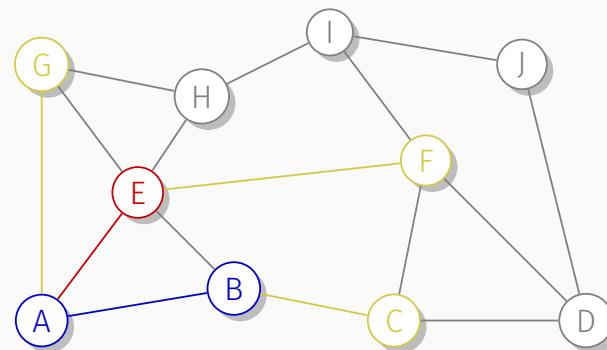


Q [G | C |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	∞	/
G	1	A
H	∞	/
I	∞	/
J	∞	/

431/776

Průchod grafem do šířky, krok 12

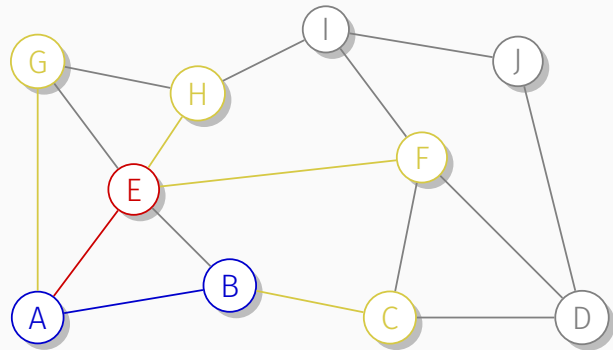


Q [G | C | F |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	∞	/
I	∞	/
J	∞	/

432/776

Průchod grafem do šířky, krok 13

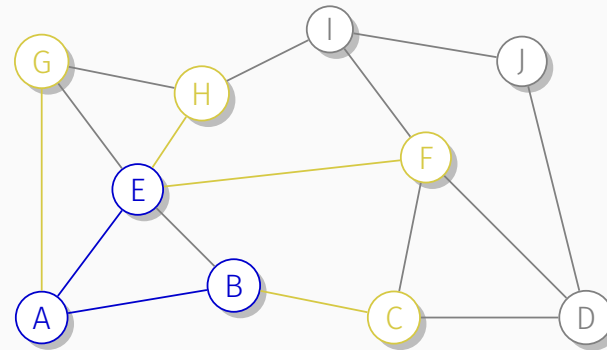


Q [G | C | F | H |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

433/776

Průchod grafem do šířky, krok 14

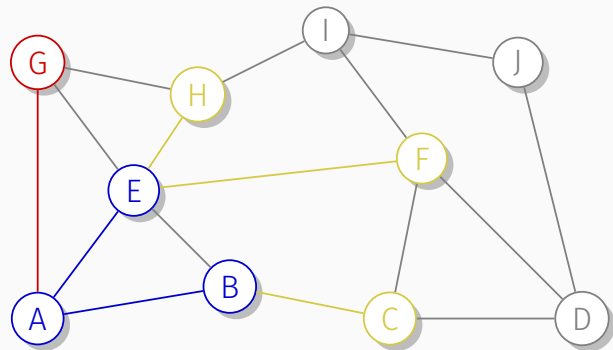


Q [G | C | F | H |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

434/776

Průchod grafem do šířky, krok 15

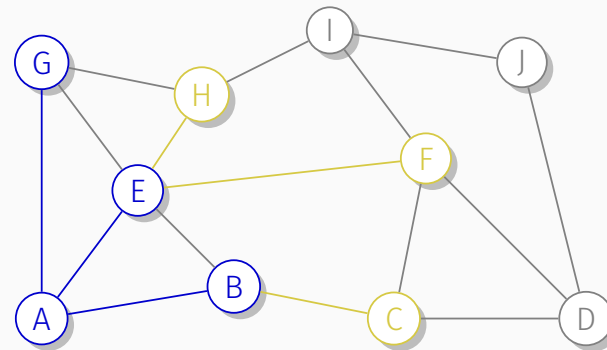


Q [C | F | H |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

435/776

Průchod grafem do šířky, krok 16

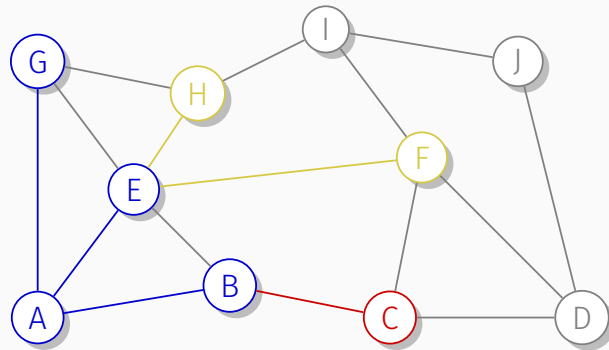


Q [C | F | H |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

436/776

Průchod grafem do šířky, krok 17

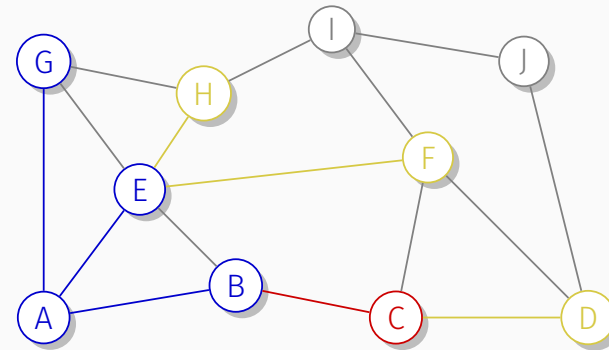


Q [F | H |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	∞	/
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

437/776

Průchod grafem do šířky, krok 18

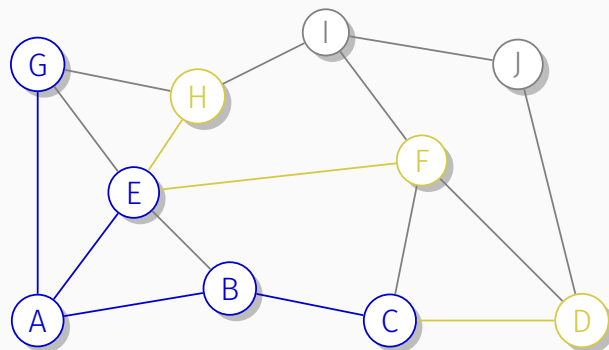


Q [F | H | D |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

438/776

Průchod grafem do šířky, krok 19

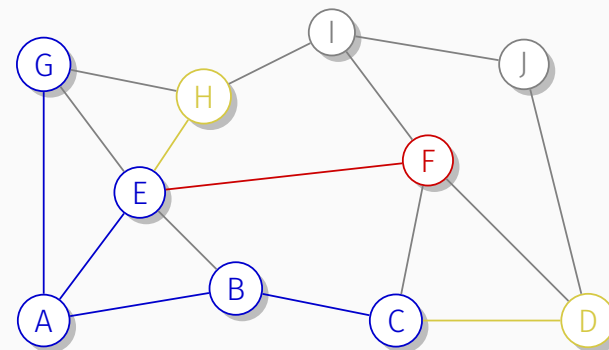


Q [F | H | D |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

439/776

Průchod grafem do šířky, krok 20

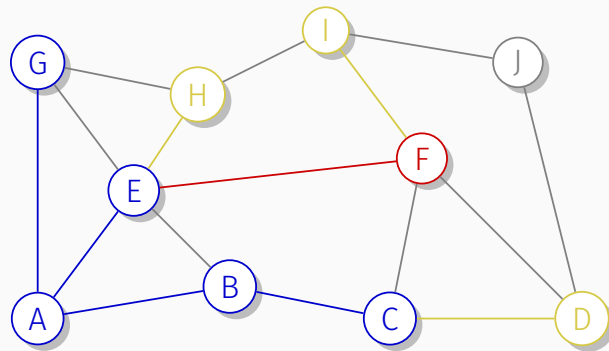


Q [H | D |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	∞	/
J	∞	/

440/776

Průchod grafem do šířky, krok 21

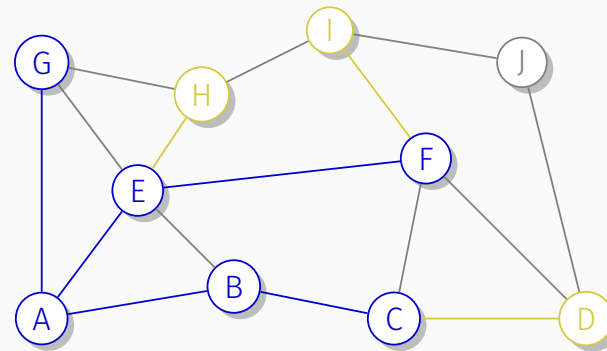


Q [H | D | I |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

441/776

Průchod grafem do šířky, krok 22

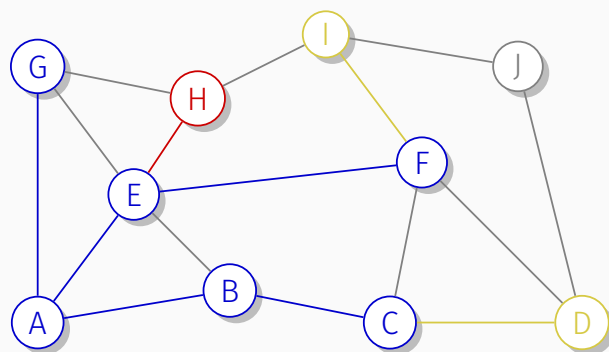


Q [H | D | I |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

442/776

Průchod grafem do šířky, krok 23

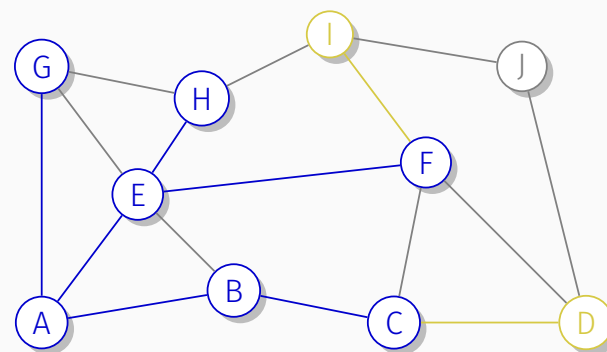


Q [D | I |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

443/776

Průchod grafem do šířky, krok 24

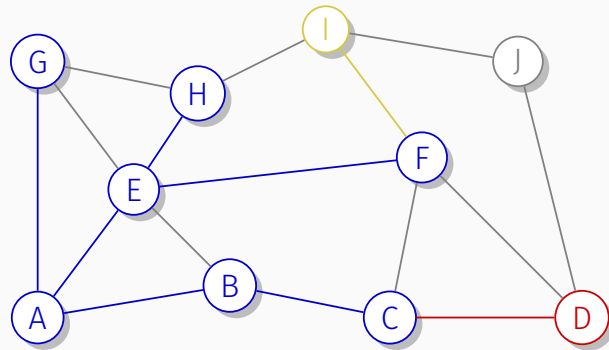


Q [D | I |]

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

444/776

Průchod grafem do šířky, krok 25

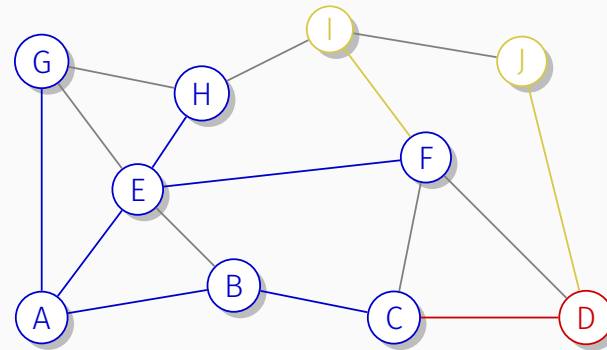


Q | I |

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	∞	/

445/776

Průchod grafem do šířky, krok 26

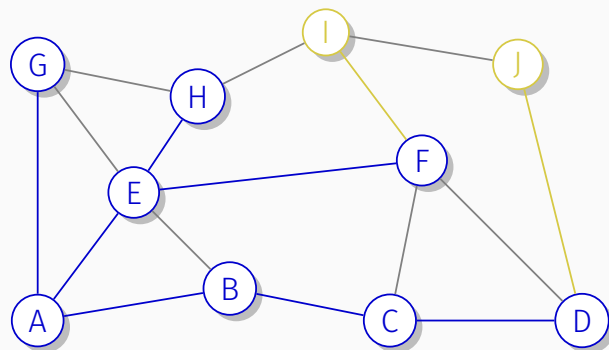


Q | I | J |

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

446/776

Průchod grafem do šířky, krok 27

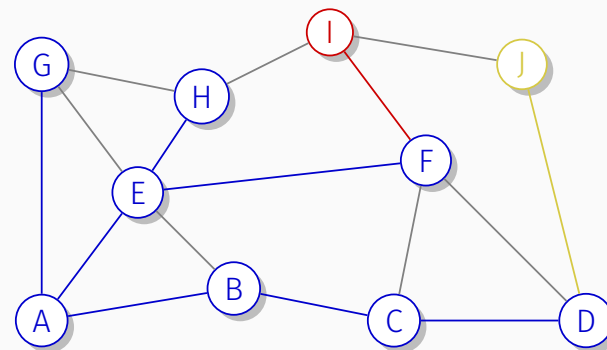


Q | I | J |

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

447/776

Průchod grafem do šířky, krok 28

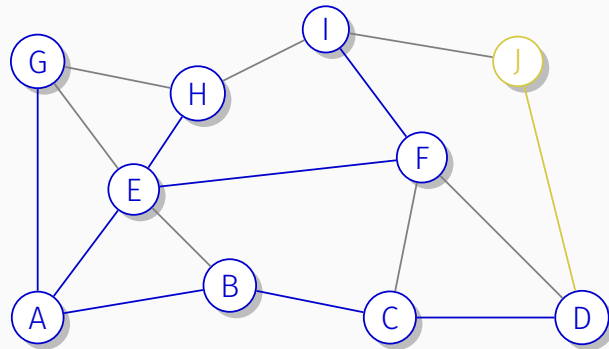


Q | J |

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

448/776

Průchod grafem do šířky, krok 29

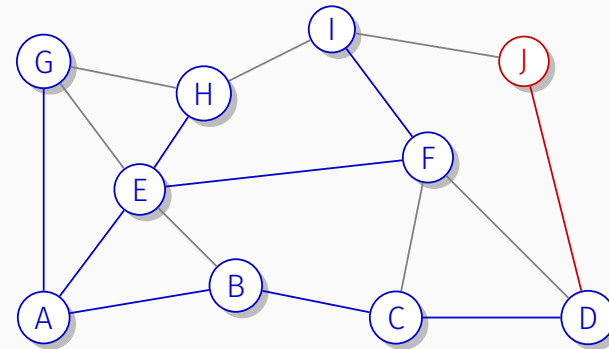


Q J

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

449/776

Průchod grafem do šířky, krok 30

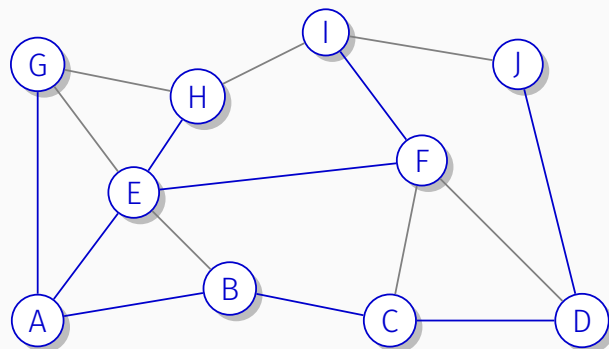


Q

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

450/776

Průchod grafem do šířky, krok 31



Q

v	d[v]	$\pi[v]$
A	0	/
B	1	A
C	2	B
D	3	C
E	1	A
F	2	E
G	1	A
H	2	E
I	3	F
J	4	D

451/776

Průchod grafem do hloubky a do šířky

Animace

K oběma algoritmům průchodu grafem jsou dostupné dvě animace:

- vyhledání cesty v bludišti a
- aplikace v počítačové grafice – vyplňování oblasti.

Animace jsou k dispozici animace v samostatných souborech.

452/776

Zdroje pro samostatné studium

- **Strategie řešení problémů hrubou silou**
 - kniha [2], kapitola 3, strany 97 – 98
- **Třídění výběrem**
 - kniha [2], kapitola 3.1, strany 98 – 100
- **Bublinové třídění**
 - kniha [2], kapitola 3.1, strany 100 – 101
- **Třídění přetřásáním**
 - kniha [4], kapitola 4, strany 78 – 79
- **Sekvenční vyhledávání**
 - kniha [2], kapitola 3.2, strany 104 – 104
- **Vyhledávání podřetězce hrubou silou**
 - kniha [2], kapitola 3.2, strany 105 – 106

453/776

Zdroje pro samostatné studium (pokrač.)

- **Problém nejbližší dvojice bodů**
 - kniha [2], kapitola 3.3, strany 108 – 109
- **Konvexní obal množiny**
 - kniha [2], kapitola 3.3, strany 109 – 113
- **Úplné prohledávání**
 - kniha [2], kapitola 3.4, strany 115
- **Problém obchodního cestujícího**
 - kniha [2], kapitola 3.4, strany 116
- **Problém batohu**
 - kniha [2], kapitola 3.4, strany 116 – 117
- **Průchod grafem do hloubky**
 - kniha [2], kapitola 3.5, strany 122 – 125
- **Průchod grafem do šířky**
 - kniha [2], kapitola 3.5, strany 125 – 128

454/776

Děkuji za pozornost

VŠB TECHNICKÁ | FAKULTA | KATEDRA
UNIVERZITA | ELEKTROTECHNIKY | INFORMATIKY
OSTRAVA | A INFORMATIKY

Strategie řešení sniž a vyřeš

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



455/776

Strategie řešení sniž a vyřeš

Třídění vkládáním – InsertSort

Strategie řešení sniž a vyřeš

Topologické třídění

Strategie řešení sniž a vyřeš

Generování kombinatorických objektů

Generování kombinatorických objektů

- Generování kombinací, variací, permutací, podmnožin je součástí různých algoritmů.
- Typicky jde o výběr nějaké alternativy, možnosti, nastavení parametrů...
- Příklady – Problém obchodního cestujícího, Problém batohu.
- Matematika se zajímá spíše počty těchto objektů, zatímco informatika hledá algoritmy jak je generovat.
- **Počet těchto objektů roste exponenciálně nebo i rychleji!**

Generování permutací

- Budeme generovat permutace celých čísel $1, 2, \dots, n$.
- Obecněji můžeme generovat permutací prvků $\{a_1, a_2, \dots, a_n\}$.
- Využití strategie sniž a vyřeš:
 1. Generování $n!$ permutací pro n prvků převedeme na generování $(n-1)!$ permutací $n-1$ prvků.
 2. Jakmile máme problém pro $n-1$ vyřešen, vložíme prvek n na všech n možných pozic do každé z $(n-1)!$ permutací.
 3. Jinak řečeno, máme $(n-1)!$ permutací, ke každé z nich vygenerujeme n dalších. Celkově dostáváme $n(n-1)! = n!$ permutací.

457/776

Generování permutací – příklad

permutace prvku 1	1		
vložení 2 do permutace 1 zprava doleva	12	21	
vložení 3 do permutace 12 zprava doleva	123	132	312
vložení 3 do permutace 21 zleva doprava	321	231	213

Co je z příkladu patrné?

- Všechny permutace jsou navzájem různé.
- Minimální změna mezi permutacemi – dvě po sobě jdoucí permutace se liší výměnou jediné dvojice prvků a to dokonce sousedních prvků.

458/776

Johnson-Trotterův algoritmus

- Existuje možnost generovat permutace n prvků? Bez nutnosti vygenerovat permutace pro $n-1$? Ano, existuje.
- Každému z n prvků permutace přiřadíme šipku (směr), buď vlevo nebo vpravo.
- O prvku k řekneme, že je v dané permutaci **mobilní**, jestliže sousední prvek ve směru šipky prvku k je menší než k .

Příklad

Permutace se šipkami

3 2 4 1

Prvky 3 a 4 jsou mobilní, prvky 2 a 1 nejsou mobilní.

459/776

Johnson-Trotterův algoritmus

Vstup : Přirozené číslo n

Výstup: Seznam všech permutací čísel $\{1, \dots, n\}$

```
1  $\pi \leftarrow 1 \ 2 \ \dots \ n$ ;  
2 while  $\pi$  obsahuje mobilní prvek do  
3    $k \leftarrow$  největší mobilní prvek v  $\pi$ ;  
4   přehod' v  $\pi$  prvek  $k$  se sousedem ve směru šipky;  
5   změň směr šipky u všech prvků větších než  $k$ ;  
6   vlož nově vytvořenou permutaci (krok 4)  $\pi$  do  
   výsledného seznamu;  
7 end  
8 return seznam všech permutací;
```

460/776

Johnson-Trotterův algoritmus – příklad

Ukázka generování permutací pro $n = 3$

```
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

O prvku k řekneme, že je v dané permutaci **mobilní**, jestliže sousední prvek ve směru šipky prvku k je menší než k .

461/776

Johnson-Trotterův algoritmus – příklad, $n = 4$

```
1 2 3 4    1 3 4 2    4 3 2 1    2 4 3 1
1 2 4 3    1 3 2 4    3 4 2 1    4 2 3 1
1 4 2 3    3 1 2 4    3 2 4 1    4 2 1 3
4 1 2 3    3 1 4 2    3 2 1 4    2 4 1 3
4 1 3 2    3 4 1 2    2 3 1 4    2 1 4 3
1 4 3 2    4 3 1 2    2 3 4 1    2 1 3 4
```

462/776

Johnson-Trotterův algoritmus

- Jeden z nejefektivnějších algoritmů pro generování permutací.
- Časová složitost algoritmu je $\Theta(n!)$.
- „Děsivá“ složitost algoritmu ale není způsobena algoritmem samotným, ten pracuje velice rychle. Je způsobena obrovským množstvím permutací, které musí vygenerovat...

463/776

Děkuji za pozornost

Strategie řešení rozděl a panuj

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



464/776

Strategie řešení rozděl a panuj



465/776

Master theorem



466/776

Strategie řešení rozděl a panuj

Násobení velkých celých čísel

Násobení velkých celých čísel

- Násobení „běžných“ celých čísel řeší procesor.
- Co násobení mnohem větších čísel, se stovkami číslic? Například v kryptografii.
- Určitě by šlo implementovat algoritmus podobný písemnému násobení.
- Jeho provedení vyžaduje n^2 násobení číslic, kde n je počet číslic.

$$\begin{array}{r} 2\ 3 \\ 1\ 4 \\ \hline 9\ 2 \\ 2\ 3\ 0 \\ \hline 3\ 2\ 2 \end{array}$$

Otázka k řešení

Lze to provést rychleji? Nebo je toto nejlepší možný algoritmus?

467/776

Násobení velkých celých čísel – násobení 23 a 14

Určíme dekadický rozvoj čísel

$$\begin{aligned} 23 &= 2 \cdot 10^1 + 3 \cdot 10^0 \\ 14 &= 1 \cdot 10^1 + 4 \cdot 10^0 \end{aligned}$$

A oba rozvoje mezi sebou vynásobíme

$$\begin{aligned} 23 \times 14 &= (2 \cdot 10^1 + 3 \cdot 10^0) \times (1 \cdot 10^1 + 4 \cdot 10^0) \\ &= (2 \times 1) \cdot 10^2 + (2 \times 4 + 3 \times 1) \cdot 10^1 + (3 \times 4) \cdot 10^0 \\ &= 322 \end{aligned}$$

Pro výpočet jsme potřebovali 4 násobení (označena \times), tj. n^2 násobení.

468/776

Násobení velkých celých čísel – násobení 23 a 14 (pokrač.)

Prostřední výraz (desítky) lze vyhodnotit i takto

$$2 \times 4 + 3 \times 1 = (2 + 3) \times (1 + 4) - 2 \times 1 - 3 \times 4$$

Neviděli jsme výrazy 2×1 a 3×4 už někde...?

Obecněji mějme $a = a_1 a_0$ a $b = b_1 b_0$ potom

$$c = a \times b = c_2 \cdot 10^2 + c_1 \cdot 10^1 + c_0,$$

kde

- $c_2 = a_1 \times b_1$ je násobek prvních číslic,
- $c_0 = a_0 \times b_0$ je násobek druhých číslic a
- $c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$ je násobek součtů číslic a a b minus číslice c_2 a c_0 .

469/776

Násobení velkých celých čísel – rozděl a panuj

Mějme dvě n -ciferná čísla a a b , kde n je sudé přirozené číslo.

Označme první polovinu číslic čísla a jako a_1 , druhou polovinu jako a_0 . Zápis $a = a_1 a_0$ budeme chápat jako

$$a = a_1 a_0 = a_1 \cdot 10^{n/2} + a_0$$

Pro $b = b_1 b_0$ platí obdobný vztah.

470/776

Násobení velkých celých čísel – rozděl a panuj

Součin $c = a \times b$ můžeme zapsat jako

$$\begin{aligned}c &= (a_1 \cdot 10^{n/2} + a_0) \times (b_1 \cdot 10^{n/2} + b_0) \\ &= (a_1 \times b_1) \cdot 10^n + (a_1 \times b_0 + a_0 \times b_1) \cdot 10^{n/2} + (a_0 \times b_0) \\ &= c_2 \cdot 10^n + c_1 \cdot 10^{n/2} + c_0,\end{aligned}$$

kde

- $c_2 = a_1 \times b_1$ je násobek prvních polovin,
- $c_0 = a_0 \times b_0$ je násobek druhých polovin a
- $c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$ je násobek součtů polovin čísel a a b minus součet c_2 a c_0 .

Čísla c_2, c_1 a c_0 vypočteme stejným způsobem – rekurzivní algoritmus.

Ukončení rekurze: $n = 1$ nebo čísla a, b lze násobit pomocí HW. 471/776

Násobení velkých celých čísel – počet násobení

- Počet násobení nutných pro výpočet součinu dvou n -ciferných čísel označíme jako $M(n)$.
- Výpočet součinu vyžaduje 3 násobení čísel poloviční velikosti. Násobení čísel pro $n = 1$ vyžaduje jedno násobení. Tedy

$$\begin{aligned}M(n) &= 3M\left(\frac{n}{2}\right) \text{ pro } n > 1 \\ M(1) &= 1\end{aligned}$$

472/776

Násobení velkých celých čísel – počet násobení (pokrač.)

- Metodou zpětné substituce pro $n = 2^k$ dostáváme

$$\begin{aligned}M(2^k) &= 3M(2^{k-1}) = 3[3M(2^{k-2})] = 3^2M(2^{k-2}) \\ &\vdots \\ &= 3^i M(2^{k-i}) \\ &\vdots \\ &= 3^k M(2^{k-k}) = 3^k\end{aligned}$$

473/776

Násobení velkých celých čísel – počet násobení (pokrač.)

- Vzhledem k tomu, že $k = \log_2 n$ dostáváme dále

$$M(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1,585}$$

Poznámky

1. Pro logaritmy platí $a^{\log_b c} = c^{\log_b a}$.
2. Rekurze nemusí nutně pokračovat až k $n = 1$, lze skončit dříve a pro malá n použít běžný algoritmus.

474/776

Násobení velkých celých čísel – počet sčítání a odčítání

- Jak je to ale se sčítáním a odčítáním? Není nižší počet násobení vykoupen vyšším počtem sčítání a násobení?
- Označme $A(n)$ počet sčítání a odčítání při násobení dvou n ciferných čísel.
- Kromě $3A\left(\frac{n}{2}\right)$ operací nutných pro rekurzivní výpočet c_2, c_1 a c_0 potřebujeme 5 součtů a 1 odečítání (na slajdu 471 označena barevně), tedy

$$A(n) = 3A\left(\frac{n}{2}\right) + cn \text{ pro } n > 1$$
$$A(1) = 1$$

475/776

Násobení velkých celých čísel – počet sčítání a odčítání (pokrač.)

- Podle vztahu (??), **Master theorem**, dostáváme

$$A(n) \in \Theta\left(n^{\log_2 3}\right)$$

- Celkový počet sčítání a odčítání roste asymptoticky stejnou rychlostí jako počet násobení.

476/776

Násobení velkých celých čísel – historie

- Autorem algoritmu je sovětský matematik Anatolij Alexejevič Karacuba (1937 – 2008), který jej představil v roce 1960.
- Do té doby převládal názor, že tradiční algoritmus je asymptoticky optimální.
- Takže má smysl se zabývat i již „vyřešenými“ problémy :-)
- Otázkou je kdy použít standardní algoritmus a kdy algoritmus založený na strategii rozděl a panuj.

477/776

Strategie řešení rozděl a panuj

Strassenovo násobení matic

Strassenovo násobení matic

- Je násobení matic pomocí strategie hrubou silou nejlepší možné?
- Složitost násobení hrubou silou je $\Theta(n^3)$.
- Asymptoticky lepší algoritmus představil Volker Strassen v roce 1969.
- Výchozí „objev“ – násobení čtvercových matic řádu 2 lze provést se 7 násobeními, na rozdíl od 8 u hrubé síly.

478/776

Strassenovo násobení matic řádu 2

$$\begin{pmatrix} c_{0,0} & c_{0,1} \\ c_{1,0} & c_{1,1} \end{pmatrix} = \begin{pmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{pmatrix} \times \begin{pmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{pmatrix} \\ = \begin{pmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{pmatrix}$$

$$\begin{aligned} m_1 &= (a_{0,0} + a_{1,1})(b_{0,0} + b_{1,1}) & m_5 &= (a_{0,0} + a_{0,1})b_{1,1} \\ m_2 &= (a_{1,0} + a_{1,1})b_{0,0} & m_6 &= (a_{1,0} - a_{0,0})(b_{0,0} + b_{0,1}) \\ m_3 &= a_{0,0}(b_{0,1} - b_{1,1}) & m_7 &= (a_{0,1} - a_{1,1})(b_{1,0} + b_{1,1}) \\ m_4 &= a_{1,1}(b_{1,0} - b_{0,0}) \end{aligned}$$

479/776

Strassenovo násobení matic

- Počty operací pro matice 2×2 :

	Hrubá síla	Strassen
Počet násobení	8	7
Počet sčítání a odčítání	4	18

- Násobit takto matice 2×2 je očividný nesmysl. Ale!

480/776

Strassenovo násobení matic (pokrač.)

- Vztahy můžeme přeformulovat na převod násobení matic $n \times n$ na podmatice řádu $\frac{n}{2} \times \frac{n}{2}$ takto:

$$\left(\begin{array}{c|c} C_{0,0} & C_{0,1} \\ \hline C_{1,0} & C_{1,1} \end{array} \right) = \left(\begin{array}{c|c} A_{0,0} & A_{0,1} \\ \hline A_{1,0} & A_{1,1} \end{array} \right) \times \left(\begin{array}{c|c} B_{0,0} & B_{0,1} \\ \hline B_{1,0} & B_{1,1} \end{array} \right)$$

- Podmatici $C_{0,0}$ můžeme vypočítat buď jako

$$C_{0,0} = A_{0,0} \times B_{0,0} + A_{0,1} \times B_{1,0}$$

nebo jako

$$C_{0,0} = M_1 + M_4 - M_5 + M_7$$

- Matice M_1, \dots, M_7 vypočteme stejným, rekurzivním, způsobem.

481/776

Strassenovo násobení matic – analýza složitosti

Počet násobení $M(n)$ pro matice $n \times n$ je dán rekurentní rovnicí:

$$\begin{aligned}M(n) &= 7M\left(\frac{n}{2}\right) \text{ pro } n > 1 \\M(1) &= 1\end{aligned}$$

Předpokládejme, že $n = 2^k$ a odtud dostáváme

$$\begin{aligned}M(2^k) &= 7M(2^{k-1}) = 7[7M(2^{k-2})] = 7^2M(2^{k-2}) \\&\vdots \\&= 7^iM(2^{k-i}) \\&\vdots \\&= 7^kM(2^{k-k}) = 7^k.\end{aligned}$$

Protože $k = \log_2 n$ a tudíž

$$M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807} < n^3$$

482/776

Strategie řešení rozděl a panuj

Problém nejbližší dvojice bodů

Strassenovo násobení matic – analýza složitosti, sčítání

- Neroste ale počet sčítání $A(n)$ pro matice $n \times n$ příliš rychle?
- Pro násobení matic $n \times n$ potřebujeme:
 1. vypočítat 7 podmatic řádu $\frac{n}{2} \times \frac{n}{2}$ a
 2. provést 18 sčítání/odečítání podmatic řádu $\frac{n}{2} \times \frac{n}{2}$.

Takže

$$\begin{aligned}A(n) &= 7A\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \text{ pro } n > 1 \\A(1) &= 0\end{aligned}$$

- Podle vztahu (??), **Master theorem**, dostáváme

$$A(n) \in \Theta(n^{\log_2 7})$$

- Z toho plyne, že Strassenovo násobení matic má asymptotickou složitost $\Theta(n^{\log_2 7})$, což je méně než řešení hrubou silou.

483/776

Strategie řešení rozděl a panuj

Konvexní obal množiny

Děkuji za pozornost

Úvodní přednáška předmětu

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



Úvodní přednáška předmětu O předmětu Algoritmy II

O předmětu Algoritmy II

Upozornění

Všechny aktuální informace k předmětu naleznete na

<http://www.cs.vsb.cz/dvorsky/>

Tato prezentace slouží jen pro účely úvodní přednášky
a nebude dále aktualizována.

O předmětu Algoritmy II

- Pokračování předmětu Algoritmy I, tj. naplní předmětu jsou další strategie algoritmického řešení úloh (dynamické programování, hladové algoritmy atd.) a typické příklady jejich užití.
- Přednášky – zaměřeny na **teorii**.
- Cvičení – zaměřena na **implementaci** řešení problémů danou strategií v jazyce C resp. C++.
- Vazby na další předměty:
 - Úvod do programování – jazyk C,
 - Funkcionální programování – rekurze a
 - Objektově orientované programování.

486/776

Rozsah předmětu, způsob zakončení

Rozsah předmětu

- výuka probíhá v zimním semestru druhého ročníku bakalářského studia
- hodinová dotace:
 - 2 hodiny přednášky a 2 hodiny cvičení týdně v prezenční formě a
 - 6 tutoriálů v kombinované formě studia
- předmět je ohodnocen 5 kredity

Způsob zakončení

- zakončení klasifikovaným zápočtem
- klasifikovaný zápočet není zkouška
- nejsou tři termíny jako u zkoušky, jiné dělení bodů v rámci celého předmětu

487/776

Garant předmětu, přednášející, tutor komb. formy

doc. Mgr. Jiří Dvorský, Ph.D.

Kancelář: EA441

Email: jiri.dvorsky@vsb.cz

Web: www.cs.vsb.cz/dvorsky



K čemu je garant předmětu?

Garant předmětu zodpovídá za průběh výuky celého předmětu, průběh cvičení, plnění úkolů na cvičeních a za korektní hodnocení úkolů. Problémy spojené se cvičeními řešte primárně se svým cvičícím. Nepodaří-li dosáhnout řešení problému s cvičícím obraťte se na garanta předmětu.

488/776

Prerekvizity

Povinná prerekvizita – úspěšné absolvování Algoritmů I.

Poznámka

- Algoritmy II jsou, jakožto povinný předmět, automaticky zapsány všem studentům.
- Nesplnění prerekvizity ale způsobí, že do Edisonu nejde zapsat jakýkoliv výsledek.
- Dokud nemáte splněné povinné prerekvizity, tak nemůžete absolvovat předmět, který je na nich závislý.

489/776

Přednášky

- jsou obecně nepovinné
- účast na nich je doporučena

Cvičení

- jsou naopak povinná,
- účast a aktivita na cvičeních jsou hodnoceny,
- je nutno získat dostatečné bodové hodnocení.

490/776

Centrum Slunečnice FEI

- <http://slunecnice-fei.vsb.cz/>,
- poskytuje podporu zpřístupňující studium i pro studenty se specifickými nároky,
- lze získat, mimo jiné, zvýšenou časovou dotaci na úkoly.

Výzva

- Pokud máte jakoukoliv úlevu, neprodleně kontaktujte svého cvičícího a garanta předmětu.
- Vyučující **nemají** k dispozici žádné seznamy studentů zahrnutých do tohoto programu.

491/776

- Individuální studijní plán umožňuje, v odůvodněných případech, individuální termíny pro plnění studijních povinností.
- Studenti, kteří mají individuální studijní plán, prosím neprodleně kontaktujte svého cvičícího a garanta předmětu a dohodněte se na individuálních termínech plnění úkolů.
- Individuální studijní plán neznamena naprostou libovůli v termínech.

492/776

- Pokud na přednášce nebudete něčemu rozumět, potřebujete poradit nebo vyřešit nějaký problém s přednáškou, cvičeními, testy, Vaší absencí na výuce atd. je možné využít **konzultační hodiny**.
- V tento čas jsem připraven věnovat se Vám osobně.
- Termín konzultačních hodin je uveřejněn mým webových stránkách.
- Žádám Vás ale o dodržení několika pravidel:
 1. Konzultaci je nutné si domluvit předem, nejlépe emailem.
 2. Pokud potřebujete poradit s učivem, přineste si s sebou materiály, které jste si k tématu prostudovali, vypište si co je Vám jasné a kde jste se „zasekli“ a potřebujete poradit.

493/776

- Konzultací s vyučujícím nic neriskujete – maximálně se dozvíte co potřebujete.
- Přijďte se zeptat rovnou ke zdroji informací – internetová fóra jsou zaplevelena různými polopravdami i naprostými nesmysly.

- Další možná forma konzultací.
- Určeno především pro kombinovanou formu studia.
- Termín konzultací je nutné individuálně dohodnout.
- Konzultace budou probíhat v MS Teams.

- Prezenční a kombinované studium má specifickou formu výuky.
- Obě formy studia mají specifické podmínky pro splnění předmětu.
- Podle formy Vašeho studia se Vás týká pouze jedna ze dvou následujících částí prezentace.

Úvodní přednáška předmětu Prezenční forma studia

Témata přednášek

1. Úvodní přednáška předmětu
2. Strategie řešení transformuj a vyřeš
3. Záměna paměťové a časové složitosti
4. Dynamické programování
5. Hladové algoritmy
6. Strategie řešení iterativním zlepšováním
7. Meze možností algoritmického řešení problémů. P, NP a NP-úplné problémy.
8. Zdolávání mezí možností algoritmického řešení problémů

497/776

Cvičení

- Přímá výuka ve cvičeních odpovídá přednáškám.
- Ve cvičeních pracují studenti pod vedením cvičícího na konkrétní implementaci příkladů v jazyce C++.
- Dále je také možné konzultovat s cvičícím probírané učivo.
- Rozdělení do cvičení tak, jak je uvedeno v informačním systému Edison, je nutné respektovat.
- Není možné překračovat kapacitu cvičení.
- Veškeré přesuny je nutné mít zaznamenány v systému Edison.

498/776

Cvičení (pokrač.)

- **Cvičení nenahrazuje přednášku!**
 - Účelem cvičení není příprava na závěrečnou písemku.
 - Cvičení nejsou bleskovou přednáškou pro ty, kteří nechodí na přednášky.
 - Na cvičení je nutné být připraven.

499/776

Úkoly

- Hodnocení v předmětu Algoritmy II se skládá ze tří částí, úkolů:
 1. průběžné aktivity na cvičeních,
 2. obhajoby projektu a
 3. závěrečné písemné práce.
- Všechny úkoly jsou povinné.
- Z každého úkolu je nutné získat aspoň minimální počet bodů.

500/776

Úkoly – průběžná aktivita na cvičeních

- Tato část hodnocení probíhá **průběžně po celý semestr**.
- Na každém cvičení bude cvičícím ohodnocena Vaše aktivita. Aktivita je hodnocena pomocí barevného kódu:
 - **zelená** – student na cvičení pracoval aktivně, v látce se orientoval, dařilo se mu implementovat zadané úkoly,
 - **oranžová** – student na cvičení byl spíše pasivní, na cvičení nebyl příliš připraven (ve znalostech měl „mezery“), implementace úkolů se příliš nedařila a
 - **červená** – student na cvičení byl zcela pasivní, o výuku nejevila zájem, implementaci úkolů nezvládl. Do této kategorie spadá i neomluvená neúčast na cvičení.

501/776

Úkoly – průběžná aktivita na cvičeních (pokrač.)

- Každému barevnému kódu odpovídá určitá váha, která se projeví v celkovém hodnocení všech cvičení. Zelená aktivita má váhu 1, oranžová má váhu 0,5 a červená 0.
- Z takto získaných vah z jednotlivých cvičení se na konci semestru vypočte průměrná váha, která se vynásobí maximálním možným počtem bodů (30) a výsledek je Vámi získaný počet bodů.
- Je zřejmé, že všechny zelené kódy odpovídají maximálnímu počtu bodů (30), samé červené kódy odpovídají nulovému počtu bodů.
- Body za aktivitu nelze získat zpětně.

502/776

Úkoly – průběžná aktivita na cvičeních (pokrač.)

Příklad

Student Franta na pěti cvičeních získal zelené hodnocení, na třech oranžové a na dvou červené. Průměrnou váhu vypočtete jako:

$$\frac{5 \times 1 + 3 \times 0,5 + 2 \times 0}{5 + 3 + 2} = \frac{6,5}{10} = 0,65.$$

Výsledné bodové hodnocení je tedy $0,65 \times 30 = 19,5$ bodů.

503/776

Úkoly – obhajoba projektu

- Zadání projektu bude zveřejněno na webu předmětu koncem října.
- Deadline odevzdání je 10. prosince 2023 23:59.
- Způsob odevzdání stanoví jednotliví cvičící.
- Obhajoby projektů proběhnou v zápočtovém týdnu a ve zkuškovém období. Harmonogram obhajob je v kompetenci cvičících.
- Bez ohledu na to, kdy proběhnou obhajoby projektů platí, že se obhajuje verze, která byla odevzdána do deadlinu.
- Na obhajobu projektu **není možný** opravný termín.

504/776

Úkoly – závěrečná písemná práce

- Závěrečná písemná práce proběhne ve zkouškovém období.
- Termíny budou zveřejněny v systému Edison.
- Opravný termín na závěrečnou písemnou práci je poskytován jen těm studentům, kteří u svého prvního pokusu získali aspoň 10 bodů.

Počet bodů na prvním termínu	Opravný termín
0 až 9	NE
10 až 20	ANO
více než 21	není nutný, úspěch

505/776

Úkoly – závěrečná písemná práce (pokrač.)

- Závěrečnou písemnou práci máte možnost psát celkem **dvakrát**, jinak řečeno máte nárok na **jednu opravu**. Předmět je ukončen klasifikovaným zápočtem. Nevztahuje se tudíž na něj požadavek dvou oprav, jak to vyžaduje studijní řád u zkoušky.
- Žádné další opravy nejsou možné.

506/776

Hodnocení úkolů

- Je nutné splnit **všechny výše uvedené úkoly**,
- a zároveň u všech úkolů **aspoň minimální počet bodů**.

Úkol	Počet bodů	
	minimum	maximum
Průb. aktivita na cvičeních	15	30
Obhajoba projektu	15	30
Písemná práce	21	40
Celkový počet bodů	51	100

507/776

Obecné pokyny ke všem úkolům

- U všech úkolů jste povinni se prokázat svou studentskou kartou nebo jiným oficiálním dokladem totožnosti. Bez prokázání totožnosti Vám nebude výsledek započítán.
- Každý **prohřešek** vůči studijnímu řádu u testů a písemné práce bude **nekompromisně postihován**. Jde především o **opisování, plagiátorství, a záměnu studentů**.
- Je zakázáno zadání testů, písemek atd. kopírovat, fotit mobily, fotoaparáty, skenovat či jakkoliv jinak kopírovat, rozmnožovat, sdílet elektronickým způsobem a podobně.

508/776

Úvodní přednáška předmětu

Kombinovaná forma studia

Tutoriály

1. tutoriál – 22. září 2023 **povinný**

- Na tomto úvodním tutoriálu Vám budou sděleny informace o organizaci studia předmětu a informace o náplni předmětu.
- Konzultace k tématům: Strategie řešení transformuj a vyřeš.

2. tutoriál – 7. října 2023

- Konzultace k tématům: Záměna paměťové a časové složitosti.

509/776

Tutoriály (pokrač.)

3. tutoriál – 20. října 2023

- Konzultace k tématům: Dynamické programování.

4. tutoriál – 3. listopadu 2023

- Konzultace k tématům: Hladové algoritmy.

510/776

Tutoriály (pokrač.)

5. tutoriál – 24. listopadu 2023

- Konzultace k tématům: Strategie řešení iterativním zlepšováním.

6. tutoriál – 8. prosince 2023

- Konzultace k tématům: Meze možností algoritmického řešení problémů. P, NP a NP-úplné problémy. Zdolávání mezi možností algoritmického řešení problémů.

511/776

Úkoly

- Hodnocení v předmětu Algoritmy II se skládá ze tří částí, úkolů:
 1. průběžné aktivity na tutoriálech,
 2. obhajoby projektu a
 3. závěrečné písemné práce.
- Všechny úkoly jsou povinné.
- Z každého úkolu je nutné získat aspoň minimální počet bodů.
- Další informace o jednotlivých úkolech budou k dispozici na webu tutora.

512/776

Úkoly – průběžná aktivita na tutoriálech

Průběžná aktivita na tutoriálech znamená:

- účast na tutoriálech a
- průběžné plnění úkolů zadaných na jednotlivých tutoriálech.

513/776

Úkoly – obhajoba projektu

- Zadání projektu bude zveřejněno na webu předmětu koncem října.
- Deadline odevzdání je 10. prosince 2023 23:59.
- Způsob odevzdání a další náležitosti budou upřesněny na webu tutora.
- Obhajoby projektů proběhnou ve zkuškovém období. Termíny budou vypsány v systému Edison.
- Bez ohledu na to, kdy proběhnou obhajoby projektů platí, že se obhajuje verze, která byla odevzdána do deadlinu.
- Na obhajobu projektu **není možný** opravný termín.

514/776

Úkoly – závěrečná písemná práce

- Závěrečná písemná práce je zaměřena na teoretické znalosti.
- Závěrečné písemná práce proběhne ve zkuškovém období.
- Termíny budou vypsány v systému Edison.
- Opravný termín na závěrečnou písemnou práci je poskytován jen těm studentům, kteří u svého prvního pokusu získali aspoň 10 bodů.

Počet bodů na prvním termínu	Opravný termín
0 až 9	NE
10 až 20	ANO
více než 21	není nutný, úspěch

515/776

Úkoly – závěrečná písemná práce (pokrač.)

- Závěrečnou písemnou práci máte možnost psát celkem **dvakrát**, jinak řečeno máte nárok na **jednu opravu**. Předmět je ukončen klasifikovaným zápočtem. Nevztahuje se tudíž na něj požadavek dvou oprav, jak to vyžaduje studijní řád u zkoušky.
- Žádné další opravy nejsou možné.

516/776

Hodnocení úkolů

- Je nutné splnit **všechny výše uvedené úkoly**,
- a zároveň u všech úkolů **aspoň minimální počet bodů**.

Úkol	Počet bodů	
	minimum	maximum
Průb. aktivita na tutoriálech	15	30
Obhajoba projektu	15	30
Písemná práce	21	40
Celkový počet bodů	51	100

517/776

Obecné pokyny ke všem úkolům

- U všech úkolů jste povinni se prokázat svou studentskou kartou nebo jiným oficiálním dokladem totožnosti. Bez prokázání totožnosti Vám nebude výsledek započítán.
- Každý **prohřešek** vůči studijnímu řádu u testů a písemné práce bude **nekompromisně postihován**. Jde především o **opisování, plagiátorství**, a **záměnu studentů**.
- Je zakázáno zadání testů, písemek atd. kopírovat, fotit mobily, fotoaparáty, skenovat či jakkoliv jinak kopírovat, rozmnožovat, sdílet elektronickým způsobem a podobně.

518/776

Úvodní přednáška předmětu Software pro výuku

Primární software

- Vývojové prostředí pro C++
- Dokumentace k C++

Doplňkový software

- Dokumentační systém Doxygen, www.doxygen.org
- Typografický systém \LaTeX , www.ctan.org

519/776

- Na učebnách je pro výuku k dispozici Microsoft Visual Studio Community 2022.
- Toto vývojové prostředí doporučuji i pro domácí přípravu.
- Obecně lze použít jakékoliv vývojové prostředí s kompilátorem podporujícím minimálně specifikaci **C++17**.

520/776

Poznámky

1. Při hodnocení Vašich projektů bude používán překladač **Microsoft Visual C++** a specifikace jazyka **C++17**.
2. Pozor na nestandardní rozšíření jazyka C++ implementovaného v GNU C++ kompilátoru.
3. Jazyk C není totožný s jazykem C++!

521/776

Úvodní přednáška předmětu
Studijní literatura

Studijní literatura

Studijní literaturu lze rozdělit do dvou skupin:

- **povinná literatura** – strategie algoritmického řešení problémů a
- **doporučená literatura** – programovací jazyk C++.

Níže uvedenou literaturu využijete v předmětu Algoritmy I i Algoritmy II.

522/776

Povinná literatura

1. LEVITIN, Anany. *Introduction to the Design and Analysis of Algorithms*. 3rd ed. Boston: Pearson, 2012. ISBN 978-0-13-231681-1.
2. CORMEN, Thomas H., Charles Eric LEISERSON, Ronald L. RIVEST a Clifford STEIN, [2022]. *Introduction to algorithms*. Fourth edition. Cambridge, Massachusetts: The MIT Press. ISBN 978-026-2046-305.
3. SEDGEWICK, Robert. *Algoritmy v C*. Praha: SoftPress, 2003. ISBN 80-864-9756-9.

523/776

Povinná literatura (pokrač.)

4. MAREŠ, Martin a Tomáš VALLA, 2017. *Průvodce labyrintem algoritmů* [online]. Praha: CZ.NIC, z.s.p.o. [cit. 2020-10-03]. CZ.NIC. ISBN 978-80-88168-19-5. Dostupné z: <https://knihy.nic.cz/>
5. WRÓBLEWSKI, Piotr. *Algoritmy*. Brno: Computer Press, 2015. ISBN 978-80-251-4126-7.
6. WIRTH, N. *Algoritmy a struktury údajov*. Alfa, Bratislava 1989.

524/776

Doporučená literatura

1. STROUSTRUP, Bjarne. *C++ programovací jazyk*. Praha: Softwarové Aplikace a Systémy, 1997. ISBN 80-901-5072-1.
2. VIRIUS, Miroslav. *Pasti a propasti jazyka C++*. 2., aktualiz. a rozš. vyd. Brno: CP Books, 2005. ISBN 80-251-0509-1.
3. SCHILDT, Herbert. *Nauč se sám C++: [poznej, vyzkoušej, použij]*. Praha: SoftPress, 2001. ISBN 80-864-9713-5.
4. ECKEL, Bruce. *Myslíme v jazyku C++*. Praha: Grada, 2000. Knihovna programátora (Grada). ISBN 80-247-9009-2.

525/776

Děkuji za pozornost

Strategie řešení transformuj a vyřeš

doc. Mgr. Jiří Dvorský, Ph.D.

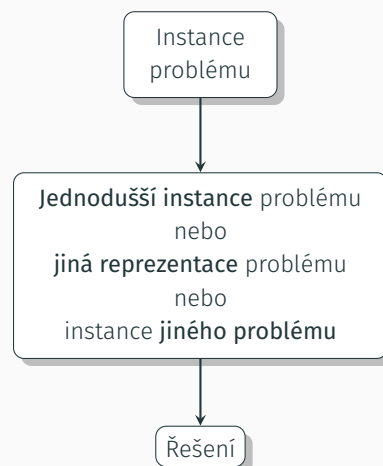
Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



Strategie řešení transformuj a vyřeš

Dvoufázová strategie

1. transformace
2. řešení



Strategie řešení transformuj a vyřeš

Předtřídění dat

Přetřídění dat

- Poměrně stará myšlenka, která mimo jiné motivovala výzkum třídících algoritmů.
- Setříděná data vedou na výrazně jednodušší algoritmy, „pořádek musí být“.
- Předpoklady:
 1. data jsou uložena v poli – třídění pole je snazší než třídění seznamu
 2. pro třídění použijeme algoritmus se složitostí $\Theta(n \log n)$ – typicky QuickSort, MergeSort.
- Využití: geometrické algoritmy, grafové algoritmy, žravé algoritmy.

528/776

Jedinečnost prvků v poli

Zadání

Máme dáno pole A s n prvky. Máme určit, zda se v poli A vyskytuje každý prvek právě jednou.

Řešení hrubou silou – porovnáváme všechny dvojice prvků dokud:

1. nenajdeme dvojici stejných prvků nebo
2. jsme otestovali všechny dvojice prvků.

Časová složitost je v nejhorším případě $\Theta(n^2)$.

529/776

Jedinečnost prvků v poli

ALGORITHM *PresortElementUniqueness*($A[0..n - 1]$)

```
//Solves the element uniqueness problem by sorting the array first
//Input: An array  $A[0..n - 1]$  of orderable elements
//Output: Returns “true” if  $A$  has no equal elements, “false” otherwise
sort the array  $A$ 
for  $i \leftarrow 0$  to  $n - 2$  do
    if  $A[i] = A[i + 1]$  return false
return true
```

Časová složitost algoritmu

$$T(n) = T_{\text{sort}}(n) + T_{\text{scan}}(n) \in \Theta(n \log n) + \Theta(n) = \Theta(n \log n)$$

530/776

Výpočet modu

Zadání

Máme dáno pole A s n prvky. Máme určit, který prvek se v poli vyskytuje nejčastěji. Tento prvek se nazývá **modus**.

Pro jednoduchost budeme předpokládat, že v poli A existuje jen jeden modus.

Řešení hrubou silou

Pro každý prvek $a_i \in A$ prohledáme pomocný seznam L :

1. pokud nalezneme shodu, inkrementujeme příslušnou četnost,
2. v opačném případě vložíme prvek a_i na konec seznamu s četností 1.

531/776

Výpočet modu – časová složitost řešení hrubou silou

- Nejhorší případ – všechny prvky v poli **A** jsou různé.
- Pro a_i musíme provést $i - 1$ porovnání s prvky v seznamu **L**, než přidáme nový prvek na jeho konec.
- Počet porovnání je tedy roven

$$C(n) = \sum_{i=1}^n (i - 1) = 0 + 1 + \dots + (n - 1) = \frac{1}{2}n(n - 1) \in \Theta(n^2)$$

- Nalezení maxima vyžaduje $n - 1$ porovnání, což neovlivní kvadratickou složitost algoritmu.

532/776

Výpočet modu – předtřídění dat

- Pokud pole **A** setřídíme, budou shodné prvky v poli **A** vedle sebe.
- Pro výpočet modu stačí nalézt nejdelší úsek (angl. run) shodných prvků v **A**.
- Časová složitost

$$T(n) = T_{\text{sort}}(n) + T_{\text{scan}}(n) \in \Theta(n \log n) + \Theta(n) = \Theta(n \log n)$$

533/776

Výpočet modu

ALGORITHM *PresortMode*($A[0..n - 1]$)

//Computes the mode of an array by sorting it first
//Input: An array $A[0..n - 1]$ of orderable elements
//Output: The array's mode

sort the array **A**

$i \leftarrow 0$ //current run begins at position i

$modefrequency \leftarrow 0$ //highest frequency seen so far

while $i \leq n - 1$ **do**

$runlength \leftarrow 1$; $runvalue \leftarrow A[i]$

while $i + runlength \leq n - 1$ **and** $A[i + runlength] = runvalue$
 $runlength \leftarrow runlength + 1$

if $runlength > modefrequency$

$modefrequency \leftarrow runlength$; $modevalue \leftarrow runvalue$

$i \leftarrow i + runlength$

return $modevalue$

534/776

Vyhledávání prvku x v poli **A** délky n

- Řešení hrubou silou vede na algoritmus vyžadující n porovnání v nejhorším případě.
- Po setřídění pole, lze použít algoritmus půlení intervalu, který vyžaduje $\lceil \log_2 n \rceil + 1$ porovnání v nejhorším případě.
- Časová složitost algoritmu potom bude

$$T(n) = T_{\text{sort}}(n) + T_{\text{search}}(n) = \Theta(n \log n) + \Theta(\log n) = \Theta(n \log n),$$

což je **více** než složitost sekvenčního vyhledávání!!!

- Ale pro **opakované** vyhledávání se již vyplatí pole **A** setřídít.

535/776

- Kniha [2], kapitola 6.1, strany 202 – 205

Strategie řešení transformuj a vyřeš Gaussova eliminační metoda

Gaussova eliminační metoda – motivace

Soustavu dvou rovnic o dvou neznámých

$$a_{11}x + a_{12}y = b_1$$

$$a_{21}x + a_{22}y = b_2$$

Lze řešit poměrně snadno – například proměnnou x vyjádříme jako funkci y , dosadíme do druhé rovnice a rovnici vyřešíme.

Problém

Jak řešit soustavu n rovnic o n neznámých? Stejným způsobem?

Gaussova eliminační metoda

Soustavu n lineárních rovnic o n neznámých

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

\vdots

$$a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n$$

transformujeme na ekvivalentní soustavu rovnic, kde všechny koeficienty pod hlavní diagonálou jsou nulové

$$a'_{11}x_1 + a'_{12}x_2 + \dots + a'_{1n}x_n = b'_1$$

$$a'_{22}x_2 + \dots + a'_{2n}x_n = b'_2$$

\vdots

$$a'_{nn}x_n = b'_n$$

Gaussova eliminační metoda – maticový zápis

$$\mathbf{A}\vec{x} = \vec{b} \implies \mathbf{A}'\vec{x} = \vec{b}'$$

kde

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \quad \vec{b} = \begin{pmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{pmatrix}$$

$$\mathbf{A}' = \begin{pmatrix} a'_{11} & a_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & a'_{nn} \end{pmatrix} \quad \vec{b}' = \begin{pmatrix} b'_{11} \\ b'_{21} \\ \vdots \\ b'_{n1} \end{pmatrix}$$

\mathbf{A}' se nazývá **horní trojúhelníková matice**.

539/776

Gaussova eliminační metoda – výhody změny reprezentace

Soustavu danou horní trojúhelníkovou maticí lze snadno řešit pomocí **zpětné substituce**:

1. Z rovnice

$$a'_{nn}x_n = b'_n$$

vypočteme neznámou x_n .

2. Hodnotu neznámé x_n dosadíme do rovnice

$$a'_{n-1\ n-1}x_{n-1} + a'_{n-1\ n}x_n = b'_{n-1}$$

a vypočteme neznámou x_{n-1} .

3. Takto postupujeme dále až k výpočtu neznámé x_1 .

Složitost tohoto algoritmu je $\Theta(n^2)$.

540/776

Gaussova eliminační metoda – elementární operace

Matici soustavy \mathbf{A} převedeme na horní trojúhelníkovou matici \mathbf{A}' pomocí **elementárních operací**:

- záměna dvou rovnic v soustavě,
- vynásobení rovnice nenulovým koeficientem a
- přičtení či odečtení násobku jiné rovnice k dané rovnici, tj. lineární kombinace s jinou rovnicí.

Elementární operace nemění řešení soustavy rovnic – transformovaná soustava má stejné řešení jako původní soustava.

541/776

Gaussova eliminační metoda – transformace matice

1. Zvolíme a_{11} jako **pivot** a „vynulujeme“ všechny koeficienty v prvním sloupci, kromě a_{11} .

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

„Vynulování“ – od druhé rovnice odečteme $\frac{a_{21}}{a_{11}}$ násobek první rovnice, od třetí rovnice odečteme $\frac{a_{31}}{a_{11}}$ násobek první rovnice,...

2. Zvolíme a_{22} jako pivot a opakujeme stejný postup.

Poznámka

Změny provádíme pochopitelně i pro vektor pravých stran \vec{b} .

542/776

Gaussova eliminační metoda – příklad

Mějme soustavu rovnic

$$2x_1 - x_2 + x_3 = 1$$

$$4x_1 + x_2 - x_3 = 5$$

$$x_1 + x_2 + x_3 = 0$$

Rozšířená matice soustavy

$$\left(\begin{array}{ccc|c} 2 & -1 & 1 & 1 \\ 4 & 1 & -1 & 5 \\ 1 & 1 & 1 & 0 \end{array} \right)$$

543/776

Gaussova eliminační metoda – příklad (pokrač.)

Dopředná eliminace

Od druhého řádku odečteme $\frac{4}{2}$ násobek prvního řádku, od třetího řádku odečteme $\frac{1}{2}$ násobek prvního řádku

$$\left(\begin{array}{ccc|c} 2 & -1 & 1 & 1 \\ 0 & 3 & -3 & 3 \\ 0 & \frac{3}{2} & \frac{1}{2} & -\frac{3}{2} \end{array} \right)$$

Od třetího řádku odečtem $\frac{1}{2}$ násobek druhého řádku

$$\left(\begin{array}{ccc|c} 2 & -1 & 1 & 1 \\ 0 & 3 & -3 & 3 \\ 0 & 0 & 2 & -2 \end{array} \right)$$

544/776

Gaussova eliminační metoda – příklad (pokrač.)

Zpětná substituce

$$x_3 = \frac{-2}{2} = -1$$

$$x_2 = \frac{3 - (-3)x_3}{3} = \frac{3 - (-3)(-1)}{3} = 0$$

$$x_1 = \frac{1 - x_3 - (-1)x_2}{2} = \frac{1 - (-1)}{2} = 1$$

545/776

Gaussova eliminační metoda – dopředná eliminace

ALGORITHM *ForwardElimination*($A[1..n, 1..n]$, $b[1..n]$)

//Applies Gaussian elimination to matrix A of a system's coefficients,
//augmented with vector b of the system's right-hand side values
//Input: Matrix $A[1..n, 1..n]$ and column-vector $b[1..n]$
//Output: An equivalent upper-triangular matrix in place of A with the
//corresponding right-hand side values in the $(n + 1)$ st column

for $i \leftarrow 1$ **to** n **do** $A[i, n + 1] \leftarrow b[i]$ //augments the matrix

for $i \leftarrow 1$ **to** $n - 1$ **do**

for $j \leftarrow i + 1$ **to** n **do**

for $k \leftarrow i$ **to** $n + 1$ **do**

$A[j, k] \leftarrow A[j, k] - A[i, k] * A[j, i] / A[i, i]$

546/776

Gaussova eliminační metoda – dopředná eliminace

Částečné pivotování

- V algoritmu dopředné eliminace je chyba. Pokud $a_{ii} = 0$, tak dojde k dělení nulou.
- Problém lze řešit výměnou rovnic (elementární operace) tak, aby $a_{ii} \neq 0$.
- Lze současně řešit i případné zaokrouhlovací chyby – pivot volíme tak, aby byl ze všech prvků a_{ij} až a_{ni} v absolutní hodnotě největší.

Opakované výpočty

V nejnvnitřnějším cyklu se podíl $\frac{a_{ji}}{a_{ii}}$ počítá opakovaně – stačí jej spočítat jednou mimo cyklus.

547/776

Gaussova eliminační metoda – částečné pivotování

ALGORITHM *BetterForwardElimination*($A[1..n, 1..n]$, $b[1..n]$)

```
//Implements Gaussian elimination with partial pivoting
//Input: Matrix  $A[1..n, 1..n]$  and column-vector  $b[1..n]$ 
//Output: An equivalent upper-triangular matrix in place of  $A$  and the
//corresponding right-hand side values in place of the  $(n + 1)$ st column
for  $i \leftarrow 1$  to  $n$  do  $A[i, n + 1] \leftarrow b[i]$  //appends  $b$  to  $A$  as the last column
for  $i \leftarrow 1$  to  $n - 1$  do
     $pivotrow \leftarrow i$ 
    for  $j \leftarrow i + 1$  to  $n$  do
        if  $|A[j, i]| > |A[pivotrow, i]|$   $pivotrow \leftarrow j$ 
    for  $k \leftarrow i$  to  $n + 1$  do
         $swap(A[i, k], A[pivotrow, k])$ 
    for  $j \leftarrow i + 1$  to  $n$  do
         $temp \leftarrow A[j, i] / A[i, i]$ 
        for  $k \leftarrow i$  to  $n + 1$  do
             $A[j, k] \leftarrow A[j, k] - A[i, k] * temp$ 
```

548/776

Gaussova eliminační metoda – časová složitost

- Velikost vstupu – počet rovnic v soustavě, tj. rozměr matice n .
- Základní operace – aritmetické operace, z historických důvodů násobení. V nejnvnitřnějším cyklu počet násobení odpovídá počtu odčítání, jde jen o násobek konstantou 2.
- Bude nás zajímat počet násobení $C(n)$ v závislosti na čísle n .

549/776

Gaussova eliminační metoda – časová složitost (pokrač.)

$$\begin{aligned} C(n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=i+1}^n 1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^n (n+1-i+1) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n (n+2-i) \\ &= \sum_{i=1}^{n-1} (n+2-i)[n-(i+1)+1] = \sum_{i=1}^{n-1} (n+2-i)(n-i) \\ &= (n+1)(n-1) + n(n-2) + \dots + 3 \cdot 1 \\ &= \sum_{i=1}^{n-1} (i+2)i = \sum_{i=1}^{n-1} i^2 + \sum_{i=1}^{n-1} 2i = \frac{(n-1)n(2n-1)}{6} + 2 \frac{(n-1)n}{2} \\ &= \frac{n(n-1)(2n+5)}{6} = \frac{2n^3 + 3n^2 - 5n}{6} = \frac{1}{3}n^3 \in \Theta(n^3) \end{aligned}$$

550/776

Gaussova eliminační metoda – časová složitost (pokrač.)

Protože složitost zpětné substituce je $\Theta(n^2)$, je složitost celé Gaussovy eliminační metody $\Theta(n^3)$.

551/776

LU-rozklad matice

Mějme matici \mathbf{A} soustavy lineárních rovnic z předchozího příkladu

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 1 \\ 4 & 1 & -1 \\ 1 & 1 & 1 \end{pmatrix}$$

Dále uvažujme dvě matice:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{pmatrix}$$

Koeficienty z Gaussovy eliminace

$$\mathbf{U} = \begin{pmatrix} 2 & -1 & 1 \\ 0 & 3 & -3 \\ 0 & 0 & 2 \end{pmatrix}$$

Výsledek Gaussovy eliminace

552/776

LU-rozklad matice

Definice

Mějme \mathbf{A} regulární čtvercovou matici s prvky z \mathbb{R} , u které není třeba při Gaussově eliminaci prohazovat řádky. Pak existují regulární matice \mathbf{L} a \mathbf{U} , které jsou určeny jednoznačně a platí pro ně následující tvrzení

$$\mathbf{A} = \mathbf{LU},$$

kde \mathbf{L} je dolní trojúhelníková matice s jedničkami na celé hlavní diagonále a \mathbf{U} horní trojúhelníková matice s nenulovými prvky na hlavní diagonále.

553/776

Řešení soustavy rovnic LU-rozkadem

Mějme soustavu lineárních rovnic

$$\mathbf{A}\vec{x} = \vec{b}$$

Matici \mathbf{A} nahradíme jejím LU rozkladem

$$\mathbf{LU}\vec{x} = \vec{b}$$

Dále označme součin $\mathbf{U}\vec{x} = \vec{y}$. Po dosazení dostáváme soustavu rovnic

$$\mathbf{L}\vec{y} = \vec{b}$$

Tuto soustavu můžeme snadno vyřešit, protože \mathbf{L} je dolní trojúhelníková matice. A nakonec můžeme lehce vyřešit i soustavu

$$\mathbf{U}\vec{x} = \vec{y},$$

protože \mathbf{U} je horní trojúhelníková matice.

554/776

Řešení soustavy rovnic LU -rozkladem, příklad

Mějme soustavu rovnic

$$2x_1 - x_2 + x_3 = 1$$

$$4x_1 + x_2 - x_3 = 5$$

$$x_1 + x_2 + x_3 = 0$$

Provedeme LU -rozklad matice soustavy \mathbf{A}

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & 1 \\ 4 & 1 & -1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 & 1 \\ 0 & 3 & -3 \\ 0 & 0 & 2 \end{pmatrix}$$

555/776

Řešení soustavy rovnic LU -rozkladem, příklad (pokrač.)

Nejprve budeme řešit soustavu $\mathbf{L}\vec{y} = \vec{b}$

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \\ 0 \end{pmatrix}$$

$$y_1 = 1$$

$$y_2 = 5 - 2y_1 = 3$$

$$y_3 = 0 - \frac{1}{2}y_1 - \frac{1}{2}y_2 = -2$$

556/776

Řešení soustavy rovnic LU -rozkladem, příklad (pokrač.)

Následně vyřešíme soustavu $\mathbf{U}\vec{x} = \vec{y}$

$$\begin{pmatrix} 2 & -1 & 1 \\ 0 & 3 & -3 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ -2 \end{pmatrix}$$

$$x_3 = \frac{-2}{2} = -1$$

$$x_2 = \frac{3 - (-3)x_3}{3} = \frac{3 - (-3)(-1)}{3} = 0$$

$$x_1 = \frac{1 - x_3 - (-1)x_2}{2} = \frac{1 - (-1)}{2} = 1$$

557/776

LU -rozklad matice, poznámky

- V praxi se pro řešení soustav lineárních rovnic používá právě LU -rozklad.
- Pomocí LU -rozkladu lze efektivně řešit více soustav rovnic se shodnou maticí soustavy.
- Matice \mathbf{L} a \mathbf{U} lze uložit společně v jedné „matici“ – z matice \mathbf{L} ukládáme jen prvky pod diagonálou. Proč?

558/776

Inverzní matice

Definice

Ke každé regulární čtvercové matici \mathbf{A} řádu n existuje právě jedna čtvercová matice \mathbf{A}^{-1} řádu n taková, že

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}.$$

Matice \mathbf{A}^{-1} se nazývá **inverzní matice** k matici \mathbf{A} .

Matice bez inverze se nazývají **singulární**.

Inverzní matice – obdoba převráceného čísla u racionálních či reálných čísel; srovnej řešení lineární rovnice a soustavy lineárních rovnic

$$\begin{aligned} \mathbf{a}\mathbf{x} &= \mathbf{b} & \mathbf{A}\vec{x} &= \vec{b} \\ \mathbf{x} &= \frac{1}{\mathbf{a}}\mathbf{b} = \mathbf{a}^{-1}\mathbf{b} & \vec{x} &= \mathbf{A}^{-1}\vec{b} \end{aligned}$$

559/776

Výpočet inverzní matice

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

Musíme řešit n soustav lineárních rovnic o n neznámých se stejnou maticí \mathbf{A}

$$\mathbf{A}\vec{x}^j = \vec{e}^j,$$

kde \vec{x}^j resp. \vec{e}^j je j -tý sloupcový vektor matice \mathbf{A} resp. \mathbf{I} .

Využijeme LU rozklad

$$\mathbf{L}\mathbf{U}\vec{x}^j = \vec{e}^j$$

560/776

Determinant matice

Definice

Determinant čtvercové matice \mathbf{A} řádu n definujeme předpisem

$$\det \mathbf{A} = \sum_{p \in P_n} \sigma(p) a_{1p_1} a_{2p_2} \dots a_{np_n},$$

kde

- P_n je množina všech permutací čísel $\{1, \dots, n\}$ a
- $\sigma(p) = (-1)^s$ je znaménko permutace, s označuje počet inverzí v permutaci p .

Determinant $\det \mathbf{A}$ zapisujeme i jako $|\mathbf{A}|$.

561/776

Výpočet determinantů matic řádu 1, 2 a 3

$$|a_{11}| = a_{11}$$

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12}$$

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{13}a_{21}a_{32} + a_{12}a_{23}a_{31} \\ - a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33}$$

Poslední vzorec je znám také jako tzv. Sarrusovo pravidlo.

562/776

Výpočet determinantů matic vyšších řádů

- Výpočet podle definice vyžaduje sečíst $n!$ součinů prvků matice.
- Opět lze využít, lehce modifikovanou, Gaussovu eliminaci a upravit matici na horní trojúhelníkovou. Platí věta, že determinant horní (dolní) trojúhelníkové matice je roven součinu prvků na její diagonále.
- Determinanty vyšších řádů tak, lze počítat s kubickou časovou složitostí.

563/776

Výpočet determinantů matic vyšších řádů – příklad

$$\begin{vmatrix} 3 & 6 & -9 \\ 4 & 8 & 10 \\ 2 & 7 & 5 \end{vmatrix} = 3 \begin{vmatrix} 1 & 2 & -3 \\ 4 & 8 & 10 \\ 2 & 7 & 5 \end{vmatrix} = 3 \begin{vmatrix} 1 & 2 & -3 \\ 0 & 0 & 22 \\ 0 & 3 & 11 \end{vmatrix} = -3 \begin{vmatrix} 1 & 2 & -3 \\ 0 & 3 & 11 \\ 0 & 0 & 22 \end{vmatrix} \\ = -3 \cdot 66 = -198$$

Poznámka

Tento výpočet slouží pouze pro ukázkou, determinant matice řádu 3 lze pochopitelně počítat přímo vzorcem.

564/776

Cramerovo pravidlo

Řešení soustavy rovnic $\mathbf{A}\vec{x} = \vec{b}$ lze vyjádřit jako

$$\vec{x} = \left(\frac{\det \mathbf{A}_1}{\det \mathbf{A}}, \dots, \frac{\det \mathbf{A}_i}{\det \mathbf{A}}, \dots, \frac{\det \mathbf{A}_n}{\det \mathbf{A}} \right)$$

kde \mathbf{A}_i je matice, která vznikne záměnou i -tého sloupce matice \mathbf{A} za vektor pravých stran \vec{b} .

Složitost algoritmu

- Výpočet determinantu řádu n lze zvládnout v čase $\Theta(n^3)$.
- Musíme vypočítat n determinantů matic \mathbf{A}_i a jeden determinant matice \mathbf{A} , celkem tedy $n + 1$ determinantů.
- Celková složitost výpočtu řešení soustavy rovnic Cramerovým pravidlem má složitost $\Theta(n^4)$.

565/776

Zdroje pro samostatné studium

- Kniha [2], kapitola 6.2, strany 208 – 216
- Kniha [3], kapitoly 28.1 a 28.2, strany 819 – 838
- Kniha [5], kapitoly 12 a 13, strany 133 – 163
- Kniha [6], kapitoly 1.3 a 1.4, strany 24 – 36

566/776

Strategie řešení transformuj a vyřeš

Vyvážené vyhledávací stromy

Binární vyhledávací stromy – připomenutí

- Fundamentální datová struktura pro implementaci množin, slovníků atd.
- Každý uzel obsahuje jeden klíč; nad klíči musí být definováno uspořádání.
- Pro každý uzel platí, že všechny klíče v levém podstromu jsou menší než klíč v daném uzlu a v pravém podstromu jsou všechny klíče větší.
- **Průměrná** časová složitost hledání, vkládání a mazání uzlů je $\Theta(\log_2 n)$.
- **Nejhorší** případ je ale stále $\Theta(n)$ – strom degeneruje na seznam.

567/776

Vyvážené vyhledávací stromy

Možná řešení nejhoršího případu:

Aktivní opatření

- transformace na vyvážený binární strom pomocí rotací
- různé definice vyváženosti
- AVL stromy, červeno-černé stromy, splay stromy.

Změna reprezentace

- více klíčů v jednom uzlu,
- 2-3 stromy, 2-3-4 stromy, B-stromy.

568/776

AVL stromy

Autoři

- Georgij Maximovič Adelson-Velskij a
- Jevgenij Michajlovič Landis

Poprvé publikováno v roce 1962.

Definice

Faktorem vyváženosti uzlu u nazýváme rozdíl výšek jeho levého a pravého podstromu. Výšku prázdného stromu definujeme jako -1 .

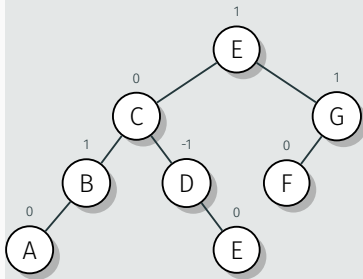
Definice

Binární vyhledávací strom nazýváme **AVL stromem** tehdy a jen tehdy, je-li faktor vyváženosti pro každý uzel ve stromu buď -1 , 0 nebo $+1$.

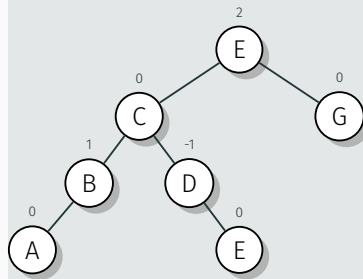
569/776

AVL stromy – příklad

AVL strom



Toto není AVL strom



570/776

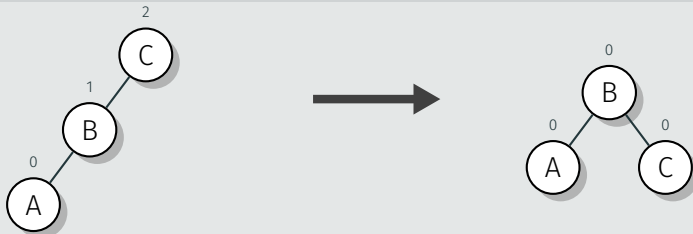
AVL stromy – udržování vyváženosti

- Vložení nového uzlu, resp. smazání exstujícího, může způsobit vyváženost AVL stromu.
- Vyváženost je nutné po každé takové operaci obnovit.
- Vyváženost se obnovuje pomocí **rotací**.
- Rotace je lokální transformace stromu v těch uzlech, kde faktor vyváženosti dosáhne hodnoty -2 nebo 2.
- Pokud je takových uzlů více, začínáme vždy uzlem na nejnižší úrovni (co nejbliže k listům stromu). A postupujeme vzhůru ke kořeni stromu.
- Existují celkem čtyři rotace – dvě dvojice navzájem zrcadlově symetrických rotací.

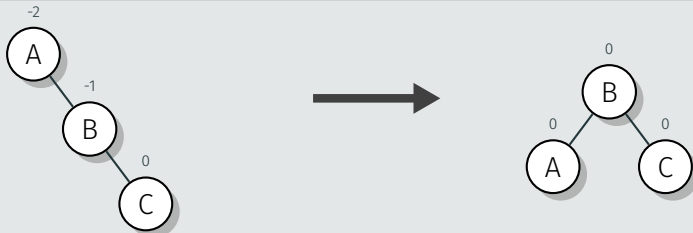
571/776

Jednoduché rotace

R-rotace



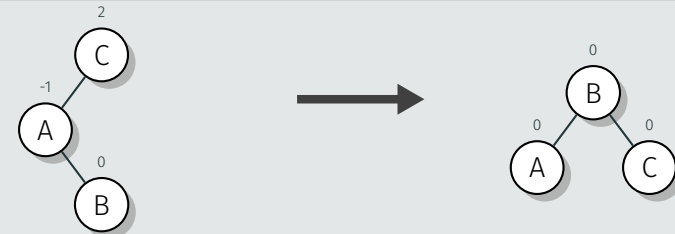
L-rotace



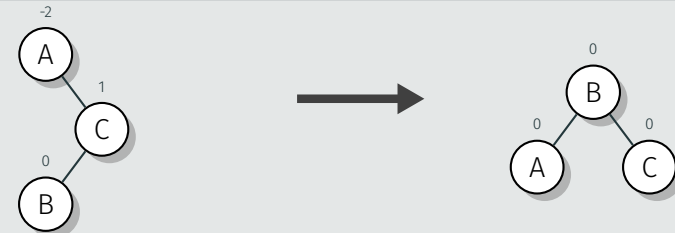
572/776

Dvojité rotace

LR-rotace

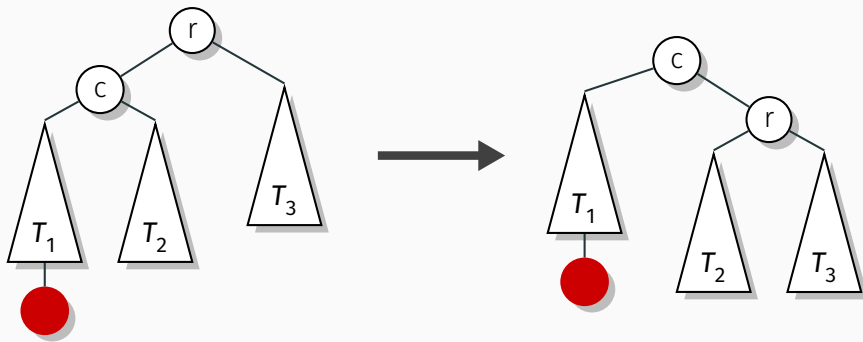


RL-rotace



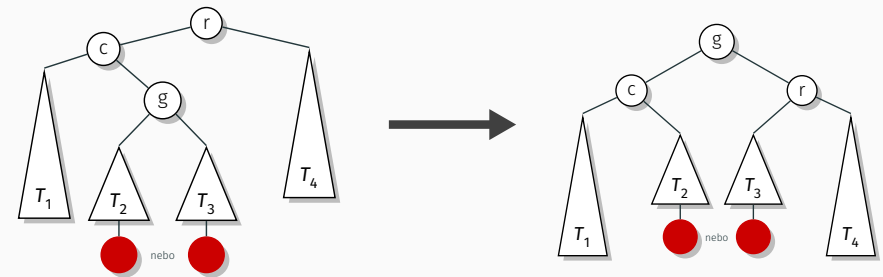
573/776

AVL stromy – obecné schéma R-rotace



574/776

AVL stromy – obecné schéma LR-rotace



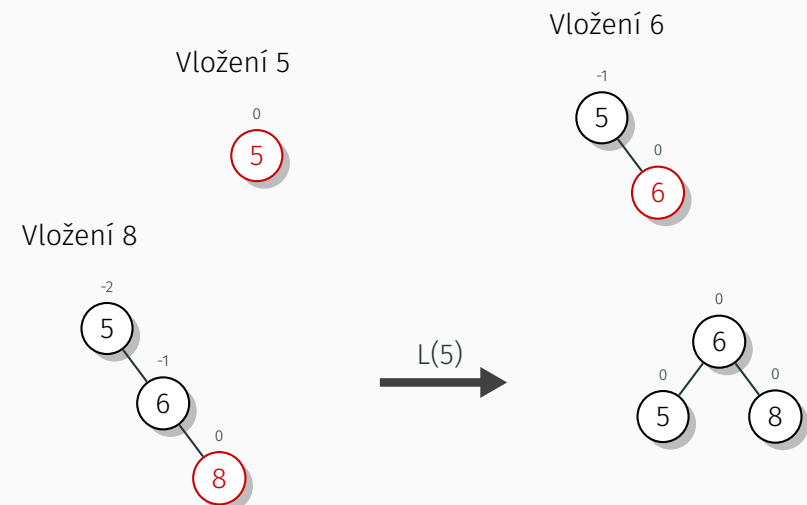
575/776

AVL stromy – vlastnosti rotací

- Konstantní časová složitost – přesunují se pouze ukazatele mezi uzly, ne data.
- Rotace zachovávají uspořádání klíčů ve stromu – po dokončení rotace jsou „vlevo“ vždy menší klíče, „vpravo“ vždy větší.

576/776

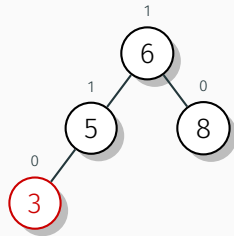
AVL stromy – postupná konstrukce stromu



577/776

AVL stromy – postupná konstrukce stromu (pokrač.)

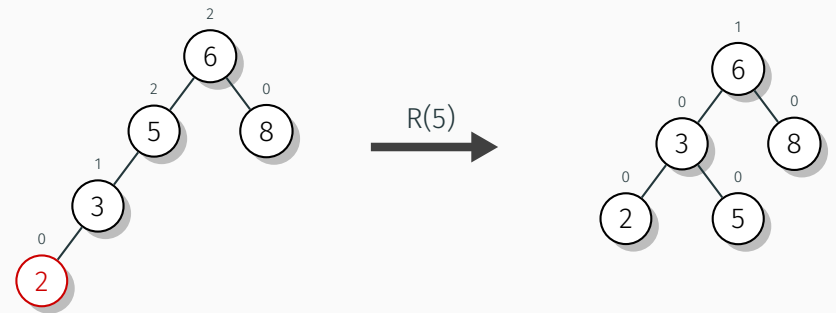
Vložení 3



578/776

AVL stromy – postupná konstrukce stromu (pokrač.)

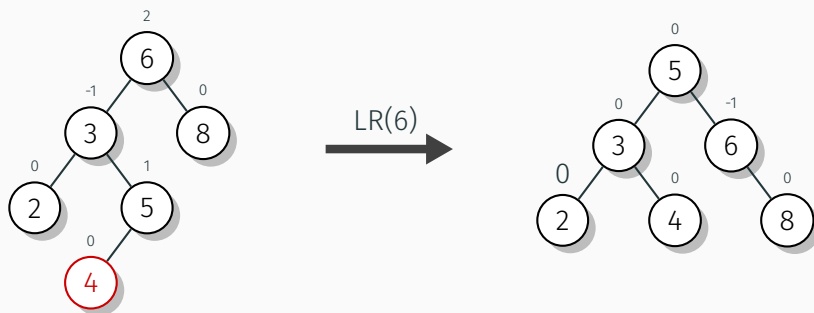
Vložení 2



579/776

AVL stromy – postupná konstrukce stromu (pokrač.)

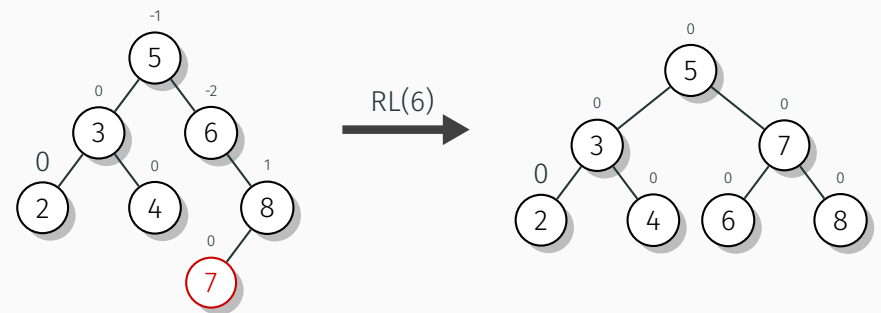
Vložení 4



580/776

AVL stromy – postupná konstrukce stromu (pokrač.)

Vložení 7



581/776

AVL stromy – vlastnosti

- Výška AVL stromu s n uzly je ohraničena

$$\lfloor \log_2 n \rfloor \leq h < 1.4405 \log_2(n + 2) - 1.3277$$

- Operace vyhledávání a vkládání tedy probíhají se složitostí $\Theta(\log_2 n)$ i v nejhorším případě.
- Průměrná výška AVL stromu sestaveného z náhodné posloupnosti n klíčů je $1.01 \log_2 n + 0.1$.
- Smazání uzlu je komplikovanější, ale stále spadá do logaritmické třídy složitosti.
- Nevýhody – velké množství rotací při vyvažování stromu.

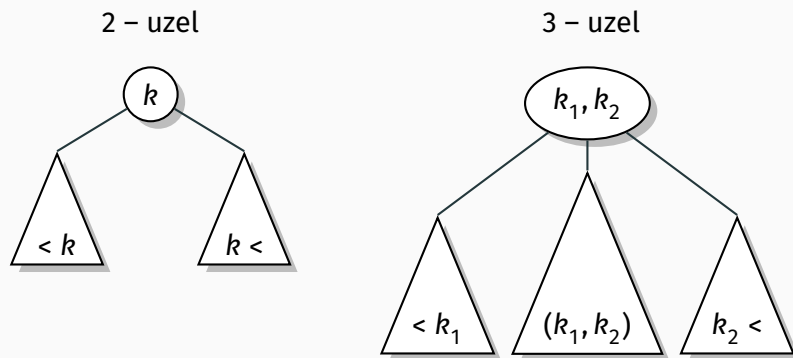
582/776

2-3 stromy



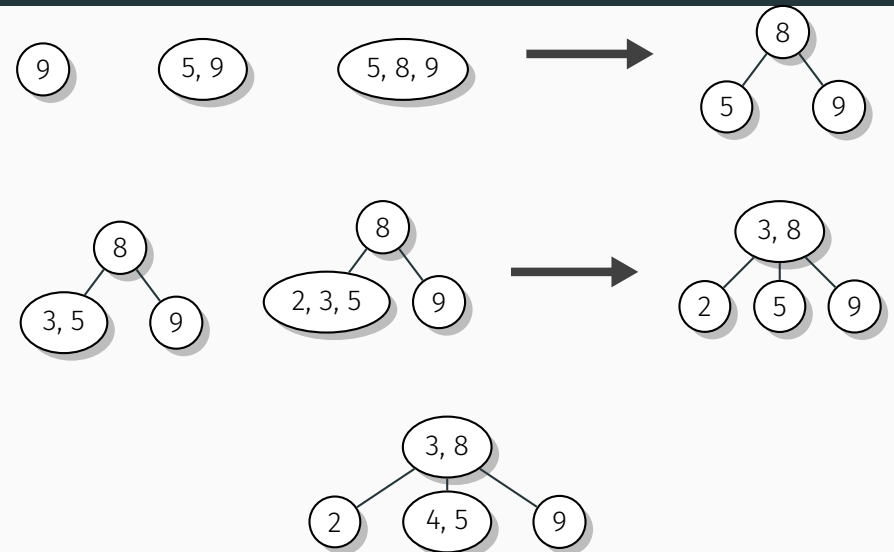
583/776

2-3 stromy – druhy uzlů



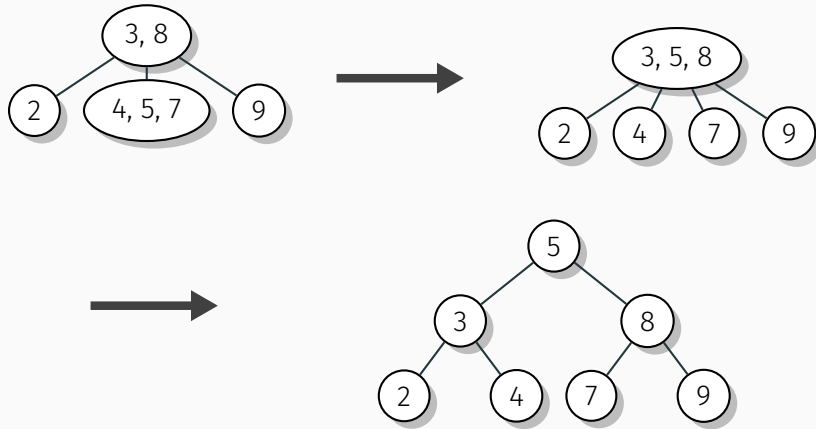
584/776

Konstrukce 2-3 stromu z posloupnosti 9, 5, 8, 3, 2, 4, 7



585/776

Konstrukce 2-3 stromu z posloupnosti 9, 5, 8, 3, 2, 4, 7 (pokrač.)



586/776

Zdroje pro samostatné studium

- Kniha [2], kapitola 6.3, strany 218 – 225
- Kniha [7], kapitoly 4.4.6, 4.4.7 a 4.4.8, strany 296 – 310

587/776

Strategie řešení transformuj a vyřeš

Halda a třídění haldou

Halda (Heap)

Halda – částečně setříděná datová struktura, zvláště vhodná pro implementaci prioritní fronty.

Prioritní fronta – datová struktura chápána jako multimnožina, kde prvky jsou řazeny podle **priority** a podporující operace:

- nalezení prvku s nejvyšší prioritou,
- odebrání prvku s nejvyšší prioritou a
- vložení nového prvku do fronty.

Využití prioritní fronty :

- plánování úloh v OS
- grafových algoritmech např. Primův, Dijkstrův atd.
- třídění haldou – **HeapSort**
- a dalších...

588/776

Halda – rozlišení terminologie

Pojem **halda** se v informatice používá pro označení:

- datové struktury a
- části operační paměti za běhu programu.

V dalším výkladu se budeme zabývat haldou výhradně jako **datovou strukturou**.

589/776

Halda

Definice

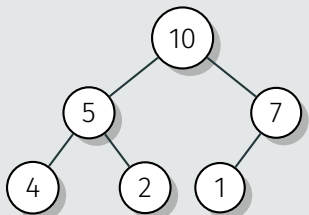
Haldu definujeme jako binární strom s jedním klíčem v každém uzlu, který splňuje následující dvě vlastnosti:

1. **kompletnost**, tj. všechna patra stromu jsou zaplněna, s výjimkou posledního. V posledním patře může zprava chybět několik listů a
2. **rodičovská dominance**, tj. klíč v každém uzlu je vždy větší nebo roven než klíče ve všech jeho potomcích. V listech je libovolný klíč vždy brán jako větší než klíče v neexistujících potomcích.

590/776

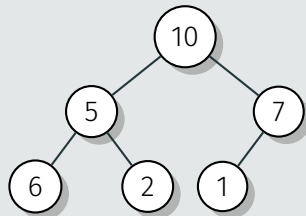
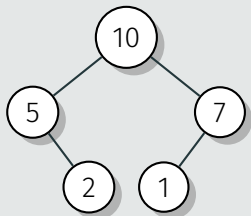
Halda – příklad

Halda



Ne každý binární strom je haldou!

Toto nejsou haldy – proč?



591/776

Halda – další vlastnosti

Pro všechny haldy lze dokázat, že:

1. Klíče na každé cestě z kořene do listu tvoří **nerostoucí** posloupnost. Jinak mezi klíči nejsou žádné vztahy, např. menší klíče v levém podstromu než v pravém atd.
2. Pro n klíčů existuje pouze jeden úplný binární strom. Jeho výška je $\lfloor \log_2 n \rfloor$.
3. Největší klíč je vždy v kořeni haldy.
4. Každý uzel v haldě je vždy kořenem haldy tvořené tímto uzlem a jeho potomky.

592/776

Halda – reprezentace v poli

V poli haldu ukládáme směrem od kořene k listům a zleva doprava: Potom:

1. vnitřní uzly – prvních $\lfloor \frac{n}{2} \rfloor$, listy zbývajících $\lceil \frac{n}{2} \rceil$,
2. potomci uzlu na pozici i , kde $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$, se nachází na pozicích $2i$ a $2i + 1$. A naopak rodič uzlu na pozici j , pro $2 \leq j \leq n$, se nachází na pozici $\lfloor \frac{j}{2} \rfloor$.

Poznámka

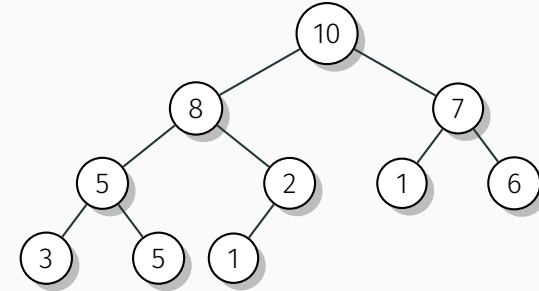
Haldu lze definovat jako pole $H[1 \dots n]$ ve kterém pro každý prvek na indexu i platí

$$H[i] \geq \max\{H[2i], H[2i + 1]\}$$

pro všechna $i = 1, \dots, \lfloor \frac{n}{2} \rfloor$.

593/776

Halda – reprezentace v poli, příklad



index	1	2	3	4	5	6	7	8	9	10
klíč	10	8	7	5	2	1	6	3	5	1
	vnitřní uzly					listy				

594/776

Konstrukce haldy

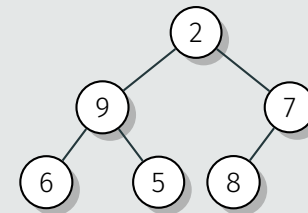
Haldu lze konstruovat dvěma způsoby:

1. zdola nahoru a
2. shora dolů.

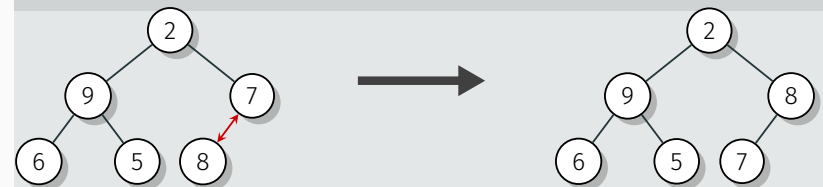
595/776

Konstrukce haldy zdola nahoru – příklad

Výchozí stav haldy



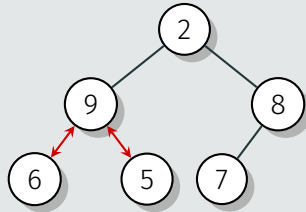
Krok 1



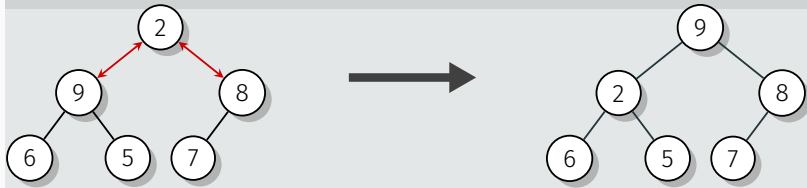
596/776

Konstrukce haldy zdola nahoru – příklad (pokrač.)

Krok 2



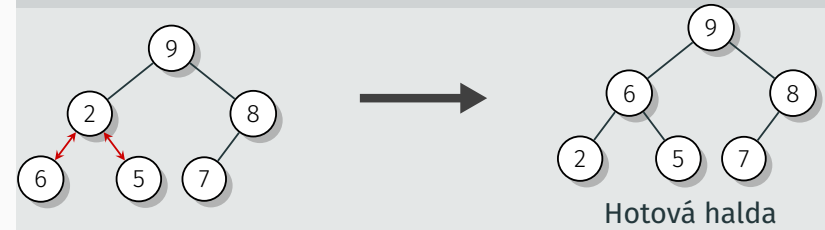
Krok 3a



597/776

Konstrukce haldy zdola nahoru – příklad (pokrač.)

Krok 3b



Hotová halda

598/776

Konstrukce haldy zdola nahoru

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním na prvcích pole, i kořen budované haldy

Výstup: Halda s kořenem na indexu i

```
1 procedure Heapify(A, n, i)
2   largest ← i;
3   l ← 2 * i + 1;
4   r ← 2 * i + 2;
5   if l < n ∧ A[l] > A[largest] then largest ← l;
6   if r < n ∧ A[r] > A[largest] then largest ← r;
7   if largest ≠ i then
8     Swap(A[i], A[largest]);
9     Heapify(A, n, largest);
10  end
11 end
```

599/776

Konstrukce haldy zdola nahoru

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním na prvcích pole

Výstup: Halda v poli A

```
1 procedure MakeHeap(A, n)
2   for i ← ⌊n/2⌋ - 1 downto 0 do
3     Heapify(A, n, i);
4   end
5 end
```

600/776

Konstrukce haldy zdola nahoru – časová složitost

Pro jednoduchost předpokládejme, že $n = 2^k - 1$, tj. halda tvoří úplný binární strom.

Výška haldy je pak $h = \lfloor \log_2 n \rfloor$, což lze psát jako

$$\begin{aligned} \lfloor \log_2(n+1) \rfloor - 1 &= \lfloor \log_2(2^k - 1 + 1) \rfloor - 1 \\ &= \lfloor \log_2(2^k) \rfloor - 1 \\ &= k - 1 \end{aligned}$$

601/776

Konstrukce haldy zdola nahoru – časová složitost (pokrač.)

Poznámka

Výraz $\lfloor \log_2(n+1) \rfloor$ lze interpretovat jako „výška haldy s $n+1$ prvky“. Předpokládali jsme úplný binární strom \Rightarrow strom s $n+1$ prvky má určitě o jednu úroveň více než strom s n prvky.

Každý klíč z úrovně i se bude při konstrukci haldy, v nejhorší případě, posunovat až do listu, tj. na úroveň h .

Posun o jednu úroveň vyžaduje dvě porovnání:

1. nalezení většího z obou potomků a
2. test, zda je nutná výměna s rodičem.

602/776

Konstrukce haldy zdola nahoru – časová složitost (pokrač.)

Počet porovnání je tedy $2(h-i)$.

Celkový počet porovnání bude v nejhorším případě roven

$$\begin{aligned} C(n) &= \sum_{i=0}^{h-1} \sum_{\text{klíče úrovně } i} 2(h-i) \\ &= \sum_{i=0}^{h-1} 2(h-i)2^i = 2h \sum_{i=0}^{h-1} 2^i - 2 \sum_{i=0}^{h-1} i2^i \\ &= 2n - 2 \log_2(n+1) \end{aligned}$$

Konstrukce haldy s n prvky vyžaduje, v nejhorším případě, méně než $2n$ porovnání.

603/776

Konstrukce haldy zdola nahoru – časová složitost (pokrač.)

Poznámka

V odvození jsme použili vzorce:

$$\begin{aligned} \sum_{i=0}^n 2^i &= 2^{n+1} - 1 \\ \sum_{i=1}^n i2^i &= 1 \cdot 2 + 2 \cdot 2^2 + \dots + n2^n = (n-1)2^{n+1} + 2 \end{aligned}$$

604/776

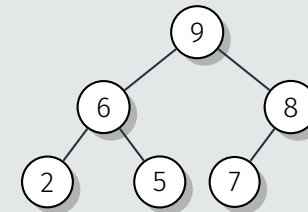
Konstrukce haldy shora dolů

- Opakované vkládání nového klíče do již existující haldy.
 1. Nový klíč vložíme na konec haldy.
 2. Nový klíč porovnáme s rodičem a případně nový klíč přesuneme o patro výš.
 3. Takto postupujeme dokud nenarazíme na většího rodiče nebo dojdeme do kořene haldy.
- Výška haldy s n prvky je $\approx \log_2 n$, tudíž složitost vložení klíče do haldy je $O(\log n)$.
- Konstrukce shora dolů je tedy složitější než zdola nahoru.

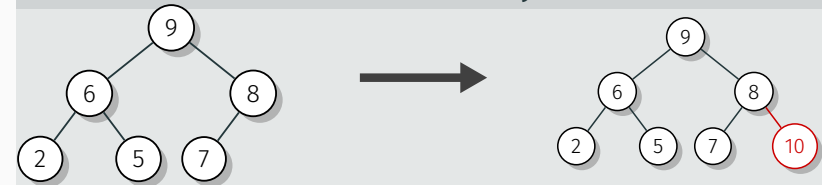
605/776

Konstrukce haldy shora dolů – příklad

Výchozí stav haldy



Krok 1 – vložení klíče 10 na konec haldy



606/776

Konstrukce haldy shora dolů – příklad (pokrač.)

Krok 2a – porovnání klíče 10 s rodičem



Krok 2b – porovnání klíče 10 s rodičem



607/776

Odstranění největšího klíče z haldy

Princip algoritmu:

1. Výměna klíče v kořeni s klíčem na konci haldy.
2. Zmenšení haldy o jedna.
3. Obnova haldy – otestovat, zda je klíč v rodiči větší než klíče v obou potomcích a případně provést výměnu. Postup opakovat dokud nebude rodičovský klíč větší než klíče v potomcích.

Poznámka

Principiálně, lze z haldy odebrat jakýkoliv klíč. Ale tato operace nemá praktický význam.

608/776

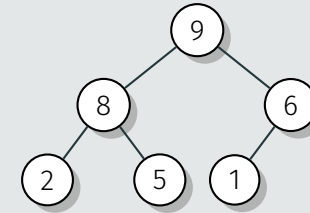
Odstranění největšího klíče z haldy – složitost algoritmu

- Počet porovnání nutných pro obnovení haldy je úměrný výšce haldy – „posunujeme“ klíč z kořene po patrech dolů.
- Porovnáváme vždy rodiče s oběma potomky – musíme najít největšího z dané trojice.
- Výška haldy je $h \approx \log_2 n$, počet porovnání nebude tedy větší než $2h$.
- Složitost algoritmu je tedy $O(\log n)$.

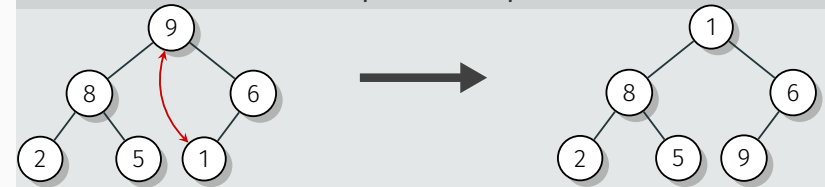
609/776

Odstranění největšího klíče z haldy – příklad

Výchozí stav haldy



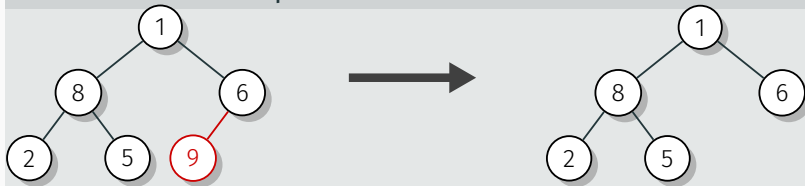
Krok 1 – záměna kořene s posledním prvkem



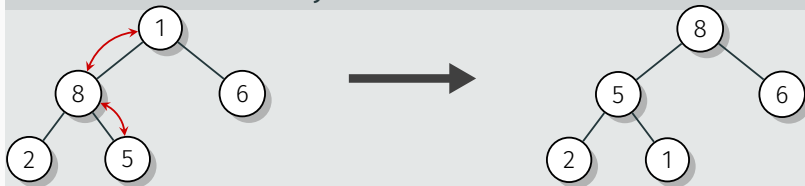
610/776

Odstranění největšího klíče z haldy – příklad (pokrač.)

Krok 2 – odstranění posledního uzlu



Krok 3 – obnovení haldy



611/776

Třídění haldou – HeapSort

Algoritmus pracuje ve dvou fázích:

Konstrukce haldy : pro dané pole je sestavena halda.

Odstranění maxima : $(n - 1)$ -krát je aplikován algoritmus pro odstranění největšího klíče z, postupně se zmenšující, haldy.

612/776

Třídění haldou – HeapSort

Vstup : Pole $A[0 \dots n - 1]$ s definovaným uspořádáním
na prvcích pole

Výstup: Setříděné pole A

```
1 procedure HeapSort (A, n)
2   MakeHeap (A, n);
3   for i ← n - 1 downto 0 do
4     Swap (A[0], A[i]);
5     Heapify (A, i, 0);
6   end
7 end
```

613/776

Třídění haldou – složitost algoritmu

- Složitost první fáze je $O(n)$.
- Ve druhé fázi postupně odstraňujeme největší klíč z haldy o klesající velikosti $n, n - 1, \dots, 2$. Počet porovnání $C(n)$ je

$$\begin{aligned} C(n) &\leq 2 \lfloor \log_2(n - 1) \rfloor + 2 \lfloor \log_2(n - 2) \rfloor + \dots + 2 \lfloor \log_2 1 \rfloor \\ &\leq 2 \sum_{i=1}^{n-1} \log_2 i \\ &\leq 2 \sum_{i=1}^{n-1} \log_2(n - 1) = 2(n - 1) \log_2(n - 1) \leq 2n \log_2 n \end{aligned}$$

Platí tedy $C(n) \in O(n \log n)$.

614/776

Třídění haldou – složitost algoritmu (pokrač.)

- Pro obě fáze dostáváme $O(n) + O(n \log n) = O(n \log n)$.
- Další analýzou složitosti lze dokázat, že stejná složitost platí i pro průměrný případ. Tedy $\Theta(n \log n)$.
- Třídění haldou je srovnatelné s tříděním sléváním.
- V praxi je však pomalejší než QuickSort.

615/776

Zdroje pro samostatné studium

- Kniha [2], kapitola 6.4, strany 226 – 232
- Kniha [3], kapitoly 6.1 až 6.4, strany 161 – 172

616/776

Strategie řešení transformuj a vyřeš

Hornerovo schéma

Hodnota polynomu v bodě

Zadání

Máme dán polynom

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0.$$

Naším úkolem je vypočítat hodnotu polynomu $p(x)$ v bodě x_0 .

Motivace

- Polynomy se využívají pro aproximaci funkcí aneb
 1. Jak procesor vypočte hodnotu funkce $\sin(x)$?
 2. Kde se vzaly hodnoty funkce $\sin(x)$ v matematických tabulkách?Pomocí Taylorova rozvoje funkce, což je polynom!
- Rychlá Fourierova transformace

617/776

Taylorův rozvoj funkce $y = f(x)$

Funkci $f(x)$, která má v bodě a konečné derivace do řádu $n + 1$ lze, v okolí bodu a , psát jako rozvoj

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_{n+1}^{f,a}(x)$$

Pro $a = 0$ se rozvoj nazývá Maclaurinův

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \dots + \frac{f^{(n)}(0)}{n!}x^n + R_{n+1}^{f,0}(x)$$

618/776

Taylorův rozvoj funkce $y = \sin(x)$ v bodě 0

$$\sin(x) = \sin(0) + \frac{\sin'(0)}{1!}x + \frac{\sin''(0)}{2!}x^2 + \dots + \frac{\sin^{(n)}(0)}{n!}x^n + R_{n+1}^{\sin,0}(x)$$

Derivace

$$\sin^{(1)} 0 = \cos 0 = 1 \quad \sin^{(2)} 0 = -\sin 0 = 0$$

$$\sin^{(3)} 0 = -\cos 0 = -1 \quad \sin^{(4)} 0 = \sin 0 = 0$$

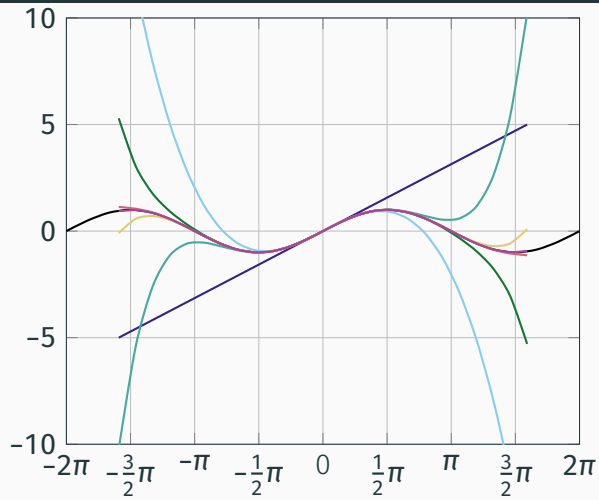
$$\sin(x) = 0 + \frac{1}{1!}x + \frac{0}{2!}x^2 + \frac{-1}{3!}x^3 + \frac{0}{4!}x^4 + \dots + R_{n+1}^{\sin,0}(x)$$

Aproximace polynomem 13-tého stupně

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!}$$

619/776

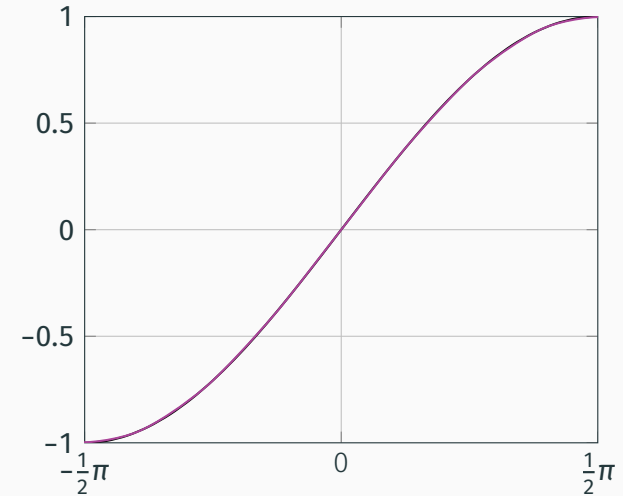
Taylorův rozvoj funkce $y = \sin(x)$ v bodě 0



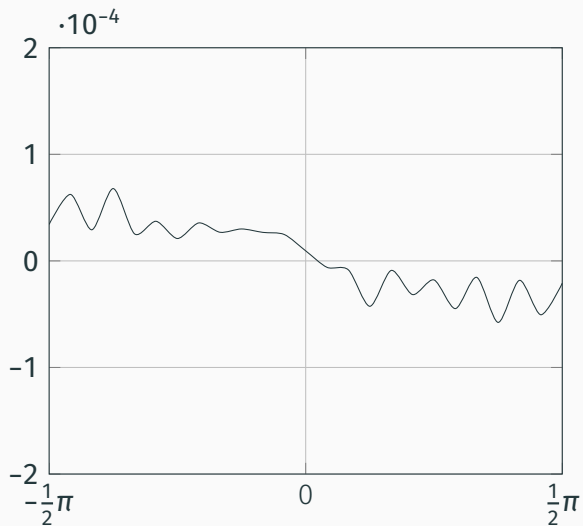
Taylorův rozvoj:
stupně 1
stupně 3
stupně 5
stupně 7
stupně 9
stupně 11
stupně 13

Funkce $y = \sin(x)$ je zobrazena černě.

Taylorův rozvoj funkce $y = \sin(x)$ stupně 13 v bodě 0



Taylorův rozvoj funkce $y = \sin(x)$ v bodě 0, chyba aproximace



Tabulky hodnot funkcí

- Taylorovým rozvojem lze aproximovat hodnotu požadované funkce a sestavit tabulky.
- Ruční výpočet – náročné a zatíženo obrovským množstvím chyb.
- Průlomová myšlenka – k numerickým výpočtům není nutná inteligence! Lze je provádět strojově!

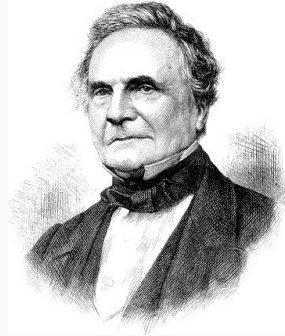
7.4 cos x (x v radiánech)										
x	0	1	2	3	4	5	6	7	8	9
0,0	1,0000	0,9998	9996	9992	9984	9972	9956	9936	9912	9884
0,1	0,9950	0,9900	9838	9776	9712	9648	9584	9520	9456	9392
0,2	0,9801	0,9701	9579	9457	9335	9213	9091	8969	8847	8725
0,3	0,9553	0,9412	9250	9088	8926	8764	8602	8440	8278	8116
0,4	0,9211	0,8930	8610	8330	8050	7770	7490	7210	6930	6650
0,5	0,8776	0,8355	7835	7415	6995	6575	6155	5735	5315	4895
0,6	0,8253	0,7683	7063	6443	5823	5203	4583	3963	3343	2723
0,7	0,7648	0,6928	6108	5288	4468	3648	2828	2008	1188	368
0,8	0,6967	0,6047	5007	3987	2967	1947	927	157	-1003	-2083
0,9	0,6216	0,5096	3787	2567	1347	127	-1013	-2193	-3373	-4553
1,0	0,5403	0,3993	2247	927	-1193	-2373	-3553	-4733	-5913	-7093
1,1	0,4534	0,2844	1027	-1343	-2623	-3903	-5183	-6463	-7743	-9023
1,2	0,3614	0,1944	137	-1583	-2963	-4343	-5723	-7103	-8483	-9863
1,3	0,2652	0,1042	-1283	-2763	-4243	-5723	-7203	-8683	-10163	-11643
1,4	0,1700	-0,0550	-2403	-4083	-5563	-7043	-8523	-10003	-11483	-12963
1,5	0,0777	-0,1592	-3303	-4983	-6563	-8143	-9723	-11303	-12883	-14463
1,6	-0,0202	-0,2992	-4083	-5763	-7343	-8923	-10503	-12083	-13663	-15243
1,7	-0,1288	-0,4488	-4643	-6323	-7903	-9483	-11063	-12643	-14223	-15803
1,8	-0,2422	-0,6042	-4983	-6763	-8343	-9923	-11503	-13083	-14663	-16243
1,9	-0,3613	-0,7692	-5083	-7063	-8643	-10223	-11803	-13383	-14963	-16583
2,0	-0,4881	-0,9442	-4943	-6963	-8543	-10123	-11703	-13283	-14863	-16463
2,1	-0,6248	-1,1308	-4663	-6883	-8463	-10043	-11623	-13203	-14783	-16363
2,2	-0,7715	-1,3278	-4143	-6603	-8263	-9843	-11423	-13003	-14583	-16163
2,3	-0,9282	-1,5358	-3383	-6423	-8063	-9643	-11223	-12803	-14383	-15963
2,4	-1,0949	-1,7548	-2403	-6243	-7863	-9443	-11023	-12603	-14183	-15763
2,5	-1,2716	-1,9848	-1323	-6063	-7663	-9243	-10823	-12403	-13983	-15563
2,6	-1,4583	-2,2248	137	-5883	-7463	-9043	-10623	-12203	-13783	-15363
2,7	-1,6550	-2,4748	1117	-5703	-7263	-8843	-10423	-12003	-13583	-15163
2,8	-1,8617	-2,7348	1857	-5523	-7063	-8643	-10223	-11803	-13383	-14963
2,9	-2,0784	-3,0048	2497	-5343	-6863	-8443	-10023	-11603	-13183	-14763
3,0	-2,3051	-3,2848	3137	-5163	-6663	-8243	-9823	-11403	-12983	-14563
3,1	-2,5418	-3,5748	3777	-4983	-6463	-8043	-9623	-11203	-12783	-14363
3,2	-2,7885	-3,8748	4417	-4803	-6263	-7843	-9423	-11003	-12583	-14163
3,3	-3,0452	-4,1848	5057	-4623	-6063	-7643	-9223	-10803	-12383	-13963
3,4	-3,3119	-4,5048	5697	-4443	-5863	-7443	-9023	-10603	-12183	-13763
3,5	-3,5886	-4,8348	6337	-4263	-5663	-7243	-8823	-10403	-11983	-13563
3,6	-3,8753	-5,1748	6977	-4083	-5463	-7043	-8623	-10203	-11783	-13363
3,7	-4,1720	-5,5248	7617	-3903	-5263	-6843	-8423	-10003	-11583	-13163
3,8	-4,4787	-5,8848	8257	-3723	-5063	-6643	-8223	-9803	-11383	-12963
3,9	-4,7954	-6,2548	8897	-3543	-4863	-6443	-8023	-9603	-11183	-12763
4,0	-5,1221	-6,6348	9537	-3363	-4663	-6243	-7823	-9403	-10983	-12563

Charles Babbage – Difference Engine

Difference Engine

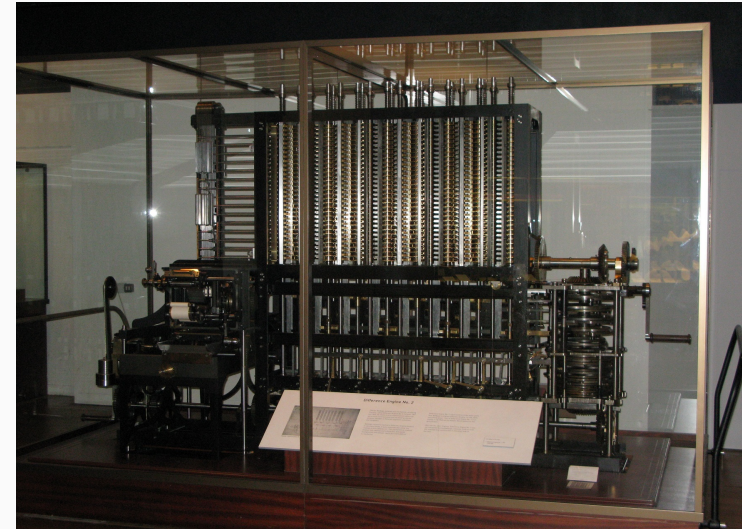
- první programovatelný počítač na světě
- 1819 – zahájení prací
- 1822 – dokončen prototyp
- 1823 – zahájeny práce na velkém stroji
- 1833 – přerušení prací
- 1842 – ukončení vládní podpory, na projekt vynaloženo 17 tisíc liber, stroj nebyl nikdy dokončen
- 1991 – funkční replika!

Charles Babbage
(1791 – 1871)



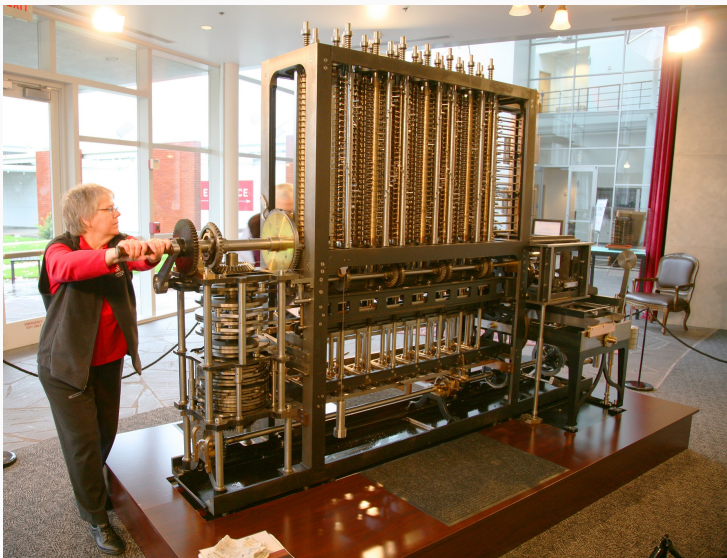
624/776

Difference Engine



625/776

Difference Engine



626/776

První programátor na světě?!

Augusta Ada King, hraběnka z Lovelace
(1815 – 1852)

Programátorka **Analytical Engine**,
(Babbage 1837), což byl první obecně
použitelný turingovsky úplný počítač.



627/776

Hornerovo schéma – transformace

Základní myšlenka:

- transformace polynomu na jiný tvar,
- postupně vytýkáme z částí polynomu proměnnou x .

$$\begin{aligned} p(x) &= a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} + a_nx^n \\ &= a_0 + x(a_1 + a_2x + \dots + a_{n-1}x^{n-2} + a_nx^{n-1}) \\ &= a_0 + x(a_1 + x(a_2 + \dots + a_{n-1}x^{n-3} + a_nx^{n-2})) \\ &\vdots \\ &= a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + a_nx) \dots)) \end{aligned}$$

Že tato rovnost platí, je snadno vidět postupným roznásobením všech závorek.

628/776

Hornerovo schéma – výpočet

Hodnotu $p(x_0)$ počítáme „z vnitřku“ závorek, postupně počítáme hodnoty b_i

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + b_nx_0 \\ b_{n-2} &= a_{n-2} + b_{n-1}x_0 \\ &\vdots \\ b_0 &= a_0 + b_1x_0 \end{aligned}$$

Hodnota b_0 je pak rovna $p(x_0)$, neboť

$$p(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \dots + x_0(a_{n-1} + a_nx_0) \dots))$$

629/776

Hornerovo schéma – výpočet (pokrač.)

a postupným dosazováním za b_i dostáváme

$$\begin{aligned} p(x_0) &= a_0 + x_0(a_1 + x_0(a_2 + \dots + x_0(a_{n-1} + b_nx_0) \dots)) \\ p(x_0) &= a_0 + x_0(a_1 + x_0(a_2 + \dots + x_0(b_{n-1}) \dots)) \\ p(x_0) &= a_0 + x_0(b_1) \\ p(x_0) &= b_0 \end{aligned}$$

630/776

Hornerovo schéma – ruční výpočet

Vypočítejte hodnotu polynomu $p(x) = 2x^3 - 6x^2 + 2x - 1$ v bodě $x_0 = 3$.

x_0	x^3	x^2	x^1	x^0
3	2	-6	2	-1
		6	0	6
	2	0	2	5

Běžný výpočet

$$\begin{aligned} p(3) &= 2 \times 3^3 - 6 \times 3^2 + 2 \times 3 - 1 \\ &= 2 \times 27 - 6 \times 9 + 2 \times 3 - 1 \\ &= 54 - 54 + 6 - 1 = 5 \end{aligned}$$

631/776

ALGORITHM *Horner*($P[0..n]$, x)

//Evaluates a polynomial at a given point by Horner's rule

//Input: An array $P[0..n]$ of coefficients of a polynomial of degree n ,

// stored from the lowest to the highest and a number x

//Output: The value of the polynomial at x

$p \leftarrow P[n]$

for $i \leftarrow n - 1$ **downto** 0 **do**

$p \leftarrow x * p + P[i]$

return p

632/776

Je zřejmé, že počet násobení $M(n)$ a počet sčítání $A(n)$ je roven

$$M(n) = A(n) = \sum_{i=0}^{n-1} 1 = n \in \Theta(n)$$

Výpočet hrubou silou

Jen pro výpočet $a_n x^n$ je zapotřebí:

- $n - 1$ násobení pro výpočet mocniny
- 1 násobení pro vynásobení a_n .

Za shodný počet násobení zvládne Hornerův algoritmus vypočítat i zbývajících $n - 1$ členů polynomu!!!

633/776

Zdroje pro samostatné studium

- Kniha [2], kapitola 6.5, strany 234 – 239
- Kniha [3], kapitola 30.1, strany 879 – 880

634/776

Strategie řešení transformuj a vyřeš

Redukce problému

Redukce problému

Smyslem redukce je převedení řešeného problému na problém jiný, který umíme vyřešit.

Postup redukce:

1. **Problém 1** – to, co chceme řešit
2. Redukce **Problému 1** na **Problém 2**
3. **Problém 2** – řešitelný algoritmem **A**
4. Provedení algoritmu **A**
5. **Řešení Problému 2**

635/776

Nejmenší společný násobek

Nejmenší společný násobek $\text{lcm}(m, n)$ dvou přirozených čísel m a n definujeme jako nejmenší přirozené číslo, které je dělitelné m a n zároveň.

Řešení pomocí prvočíselného rozkladu:

$$\begin{aligned}24 &= 2^3 \cdot 3^1 \\60 &= 2^2 \cdot 3^1 \cdot 5^1 \\ \text{lcm}(24, 60) &= 2^3 \cdot 3^1 \cdot 5^1 = 120\end{aligned}$$

Řešení pomocí největšího společného dělitele:

Lze dokázat, že

$$\text{lcm}(m, n) = \frac{mn}{\text{gcd}(m, n)}$$

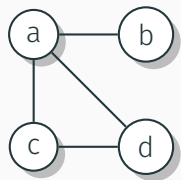
$\text{gcd}(m, n)$ lze vypočítat, efektivním, Euklidovým algoritmem.

636/776

Počet sledů v grafu

Zadání: Vypočítat počet sledů mezi dvojicemi vrcholů v daném grafu G .

Řešení: Lze dokázat, že počet různých sledů délky k mezi vrcholy i a j je roven prvku a_{ij} matice \mathbf{A}^k , kde \mathbf{A} je matice sousednosti grafu G .



$$\mathbf{A} = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{matrix} \quad \mathbf{A}^2 = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{pmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{pmatrix} \end{matrix}$$

Z a do a vedou tři sledy délky 2: $a - b - a$, $a - c - a$, $a - d - a$

Z a do c vede jeden sled délky 2: $a - d - c$

637/776

Redukce optimalizačních problémů

Maximalizační problém – nalezení maxima funkce $f(x)$

Minimalizační problém – nalezení minima funkce $f(x)$

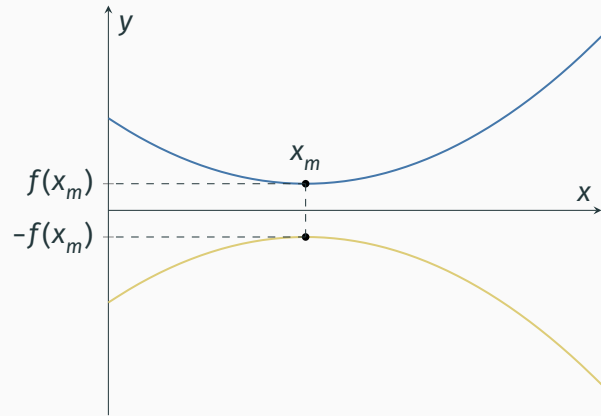
Jak řešit situaci?

- Máme minimalizovat funkci $f(x)$, ale
- k dispozici máme pouze maximalizační algoritmus.

Lze využít maximalizační algoritmus pro minimalizační problém? Případně naopak?

638/776

Redukce optimalizačních problémů



$$\min f(x) = -\max[-f(x)]$$

$$\max f(x) = -\min[-f(x)]$$

639/776

Koza, vlk a zelí

- Na břehu řeky je převozník, koza, vlk a zelí.
- Převozník má převést kozu, vlka a zelí na druhý břeh pomocí loďky.
- Na loďku se, mimo převozníka, vejde nejvýše jedna z převážených entit.
- Na stejném břehu se nesmí bez převozníkovy dozoru ocitnout dvojice (koza, zelí) a (vlk, koza).
- Úkolem je sestavit plán převozu nebo dokázat, že řešení neexistuje.

Nejstarší písemná podoba úlohy pochází z 9. století...

640/776

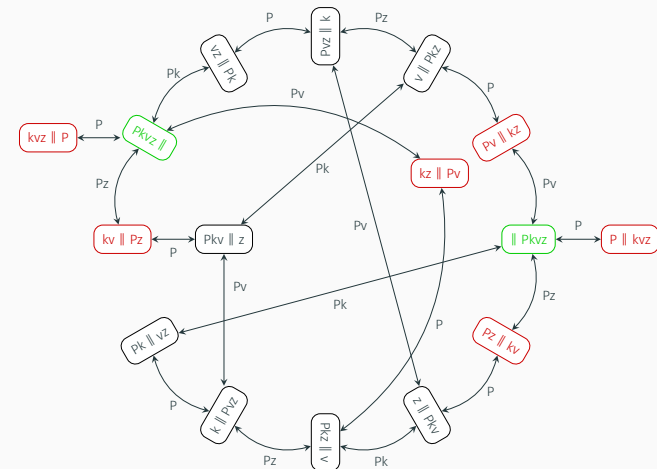
Koza, vlk a zelí – stavový prostor

Stav – reprezentuje obsazení obou břehů oddělených řekou, např. Pkv||z

Přechod mezi stavy – cesta z jednoho břehu řeky na druhý, s případným převozem

641/776

Koza, vlk a zelí – graf stavového prostoru



Řešení problému – nalezení orientované cesty z počátečního stavu do koncového stavu průchodem do šířky.

642/776

- Kniha [2], kapitola 6.6, strany 240 – 248

Děkuji za pozornost

Záměna paměťové a časové složitosti

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



Záměna paměťové a časové složitosti

B-stromy

B-stromy – motivace

- Zpracování velkého množství strukturovaných záznamů (lze je identifikovat jednoznačným klíčem), které přesahuje dostupnou operační paměť.
- Data musí být uložena ve vnější paměti, tzv. „na disku“.
- Disk nabízí jen sekvenční soubor.
- Hledáme datovou strukturu, která umožní v takovém souboru efektivně vyhledávat, vkládat a mazat záznamy.
- Odpovědí je záměna paměťové složitosti za časovou, jinak řečeno zvýšíme paměťovou složitost (obětujeme extra paměť navíc), abychom snížili časovou složitost operací.

645/776

B-stromy

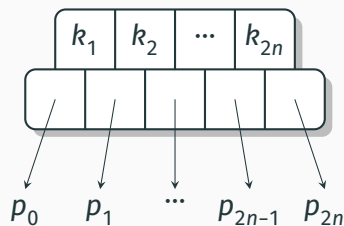
B-strom řádu n je $(2n + 1)$ -ární strom, který splňuje následující kritéria:

1. Každá stránka obsahuje nejvýše $2n$ klíčů.
2. Každá stránka, s výjimkou kořenové obsahuje alespoň n klíčů.
3. Každá stránka je buď listovou stránkou, tj. nemá žádné potomky, nebo má $m + 1$ potomků, kde m je aktuální počet klíčů ve stránce.
4. Všechny listové stránky jsou na stejné úrovni. Jinak řečeno strom je dokonale vyvážený.

Publikováno Rudolfem Bayerem v roce 1972 [8].

646/776

B-stromy – schéma stránky



- Uzly v B-stromu se tradičně nazývají **stránky** (pages).
- Počet klíčů ve stránce kolísá od n do $2n$, výjimku tvoří kořen stromu.
- Klíče ve stránce jsou setříděné, tedy $k_1 \leq k_2 \leq \dots \leq k_{2n}$.

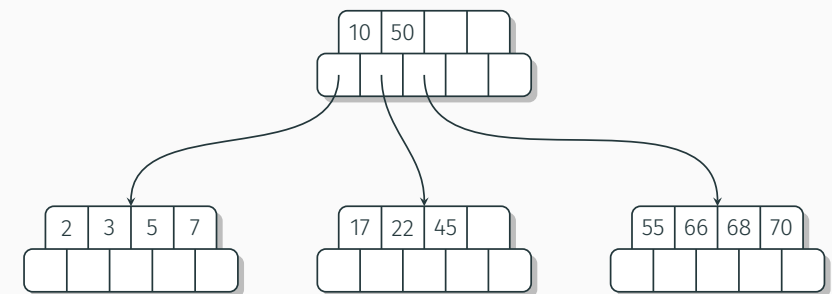
- Pro klíče v podstromech na které odkazují ukazatelé p_0, \dots, p_{2n} platí

$$K_{p_0} \leq k_1 \leq K_{p_1} \leq k_2 \leq \dots \leq K_{p_{2n-1}} \leq k_{2n} \leq K_{p_{2n}},$$

kde K_{p_i} je množina všech klíčů v podstromu s kořenem ve stránce na kterou ukazuje p_i .

647/776

B-stromy – ukázka



648/776

B-stromy – poznámky

- Z definice je zřejmé, že B-strom není vždy zcela zaplněn. Faktor zaplnění kolísá od 50 % do 100 %.
- Volné místo ve stromu umožňuje snadné vkládání dalších klíčů.
- Díky stromové struktuře lze operace vyhledávání, vkládání a mazání klíčů v B-stromu provádět s logaritmickou časovou složitostí.
- Algoritmy pro B-stromy jsou zobecněním algoritmů pro binární vyhledávací stromy.

649/776

B-stromy – alternativní definice, varianty

- Výše uvedená definice připouští jen B-stromy s max. kapacitou $2n$ klíčů, tj. sudým číslem.
- Max. kapacita může být ale libovolná, i lichá.
- Některé definice označují pomocí čísla n max. kapacitu.
- Počet klíčů ve stránce tak kolísá od $\lceil \frac{n}{2} \rceil$ do n .
- Cílem naší definice je snadná pochopitelnost principů fungování B-stromu a jednoduchý zápis.
- Někdy se lze setkat i s definicí, kde číslo n označuje max. počet potomků nikoliv počet klíčů ve stránce.

650/776

B-stromy – varianty

- B⁺-strom**
- všechny klíče jsou uloženy pouze v listech
 - listy jsou navzájem propojeny odkazy – rychlejší práce se souvislými úseky klíčů, „najdi všechny klíče mezi 100 a 200“
 - v knize od Levitina [2] je jako B-strom popsána právě tato varianta.
- B^{*}-strom**
- stránka musí být zaplněna minimálně ze dvou třetin,
 - při vkládání klíče do zaplněné stránky jsou klíče nejprve přesouvány mezi sourozenci,
 - dochází k menšímu počtu štěpení stránek.

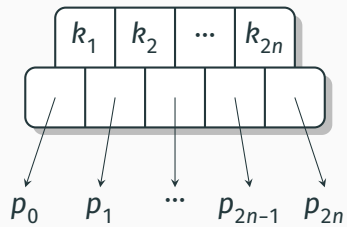
651/776

B-stromy – vyhledávání klíče x

1. Na počátku algoritmu označíme kořenovou stránku za aktuální stránku P .
2. Pokud stránka P neexistuje, vyhledávání končí neúspěchem.
3. Jinak předpokládejme, že ve stránce P je m klíčů k_1, \dots, k_m a odpovídající ukazatelé na potomky p_0, \dots, p_m . Potom:
 - 3.1 Pokud $x = k_i$, pro některé $1 \leq i \leq m$, pak vyhledání končí úspěchem.
 - 3.2 Jestliže $x < k_1$, potom $P = p_0$ a zpět k bodu 2.
 - 3.3 Jestliže $x > k_m$, potom $P = p_m$ a zpět k bodu 2.
 - 3.4 Jinak nalezneme takové i , $1 \leq i \leq m$, pro které platí, že $k_i < x < k_{i+1}$. Potom $P = p_i$ a a zpět k bodu 2.

652/776

Příklady operací s B-stromem

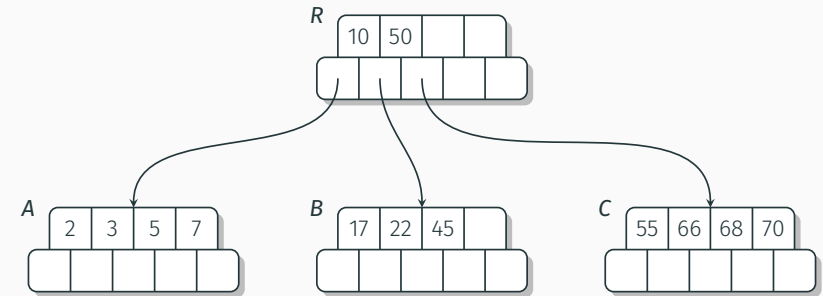


V ukázkách budeme používat B-strom pro $n = 2$, tzn. každá stránka obsahuje minimálně 2 a maximálně 4 klíče.

A dále každá stránka odkazuje na minimálně 3 a maximálně 5 potomků. Výjimku tvoří kořen stromu.

653/776

Příklad vyhledávání v B-stromu – vyhledání klíče 50

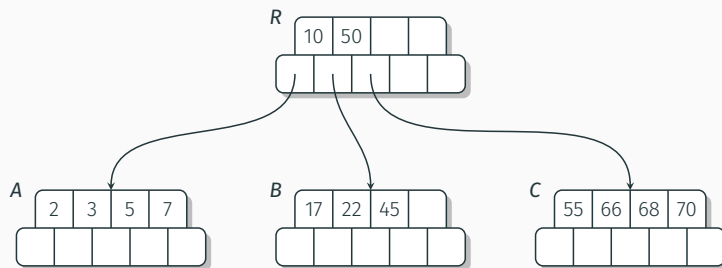


Postup

1. Vyhledávání zahájíme v kořeni R , tedy $P = R$.
2. Stránka P existuje, pokračujeme dalším bodem.
3. Protože $x = k_2$ hledání končí úspěchem.

654/776

Příklad vyhledávání v B-stromu – vyhledání 3

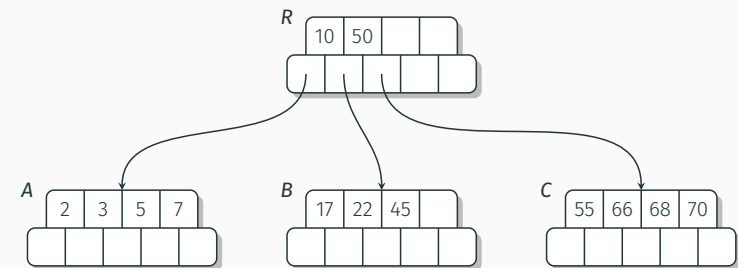


Postup

1. Vyhledávání zahájíme v kořeni R , tedy $P = R$.
2. Stránka P existuje, pokračujeme dalším bodem.
3. Protože $x < k_1$, pak $P = p_0 = A$.
4. Stránka P existuje, pokračujeme dalším bodem.
5. Protože $x = k_2$ hledání končí úspěchem.

655/776

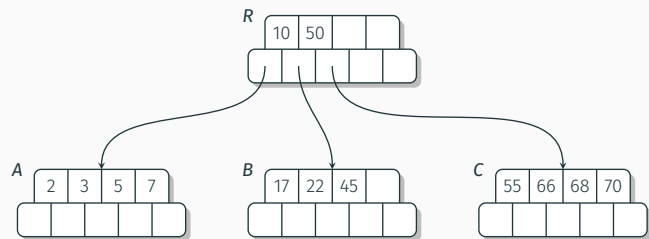
Příklad vyhledávání v B-stromu – vyhledání 45



Postup

1. Vyhledávání zahájíme v kořeni R , tedy $P = R$.
2. Stránka P existuje, pokračujeme dalším bodem.
3. Protože $k_1 < x < k_2$, pak $P = p_1 = B$.
4. Stránka P existuje, pokračujeme dalším bodem.
5. Protože $x = k_3$ hledání končí úspěchem.

656/776



Postup

1. Vyhledávání zahájíme v kořeni R , tedy $P = R$.
2. Stránka P existuje, pokračujeme dalším bodem.
3. Protože $x > k_2$, pak $P = p_2 = C$.
4. Stránka P existuje, pokračujeme dalším bodem.
5. Protože $k_1 < x < k_2$, pak $P = p_1 = null$.
6. Protože P neexistuje, hledání končí **neúspěchem**.

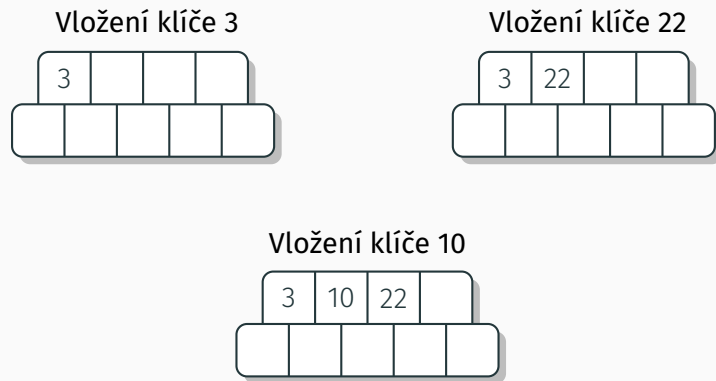
1. Nejprve je nutné určit, pomocí algoritmu vyhledávání, listovou stránku L kam bude klíč x vložen.
2. Mohou nastat dva případy:
 - Stránka L není zcela zaplněna – klíč x je vložen do stránky tak, aby zůstalo zachováno uspořádání klíčů.
 - Stránka L je zcela zaplněna, potom
 - 2.1 klíč x zařídíme (např. v pomocném poli) mezi klíče ze stránky L tak, aby zůstalo zachováno uspořádání klíčů. Dostaneme posloupnost $2n + 1$ klíčů $k'_1 < k'_2 < \dots < k'_{2n+1}$
 - 2.2 vytvoříme novou stránku P , se stejným rodičem R jako L
 - 2.3 distribuce klíčů do stránek

Klíče	Akce
k'_1, \dots, k'_n	ponechat ve stránce L
k'_n	vložit do rodičovské stránky R
$k'_{n+2}, \dots, k'_{2n+1}$	vložit do nové stránky P

- Proces vytvoření nové stránky a přerozdělení klíčů nazýváme **štěpení stránky**.
- Vložení klíče k'_n do rodičovské stránky R dojde ke zvýšení počtu klíčů v této stránce a tím i k nárůstu počtu odkazů na potomky této stránky. Bez přesunu k'_n by ve stránce R chyběl volný odkaz pro připojení stránky P .
- Vložení klíče k'_n do stránky R se realizuje stejným algoritmem jako vložení klíče x do L . Vložení k'_n může způsobit rozštěpení stránky R .
- Štěpení stránek může vést až k vytvoření nového kořene celého stromu, což je jediný způsob jak B-strom může zvýšit svoji výšku.

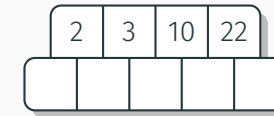
- V tomto rozsáhlejší ukázkovém příkladu budeme postupně budovat B-strom se stejnými parametry jako u příkladu vyhledávání.
- Do stromu postupně vložíme klíče 3, 22, 10, 2, 17, 5, 66, 68, 50, 7, 55, 45, 70, 44, 6, 21, 67, 1, 4, 8, 9, 12 a 15.

Příklad vkládání do B-stromu – vložení klíčů 3, 22 a 10



661/776

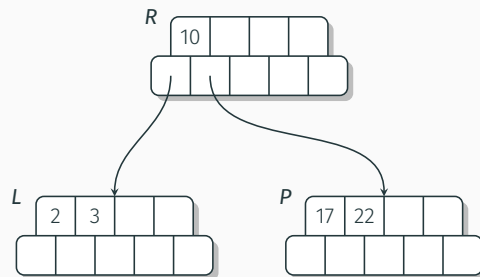
Příklad vkládání do B-stromu – vložení klíče 2



Stránka je zcela zaplněna, vložení libovolného dalšího klíče způsobí změnu struktury B-stromu.

662/776

Příklad vkládání do B-stromu – vložení klíče 17

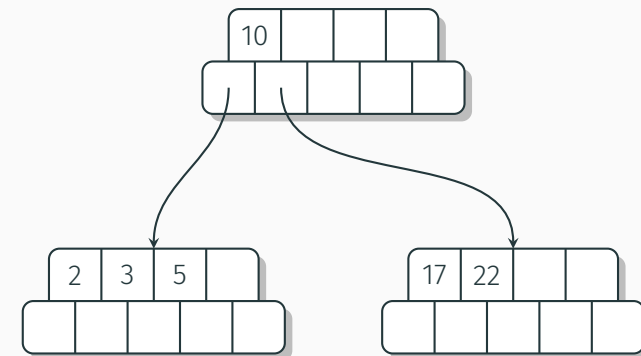


Vložním klíče 17 došlo:

1. k rozštěpení stránky *L* a k přesunu poloviny klíčů do nové stránky *P*,
2. ke vzniku nové kořenové stránky *R* a k přesunu klíče 10 do nového kořene.

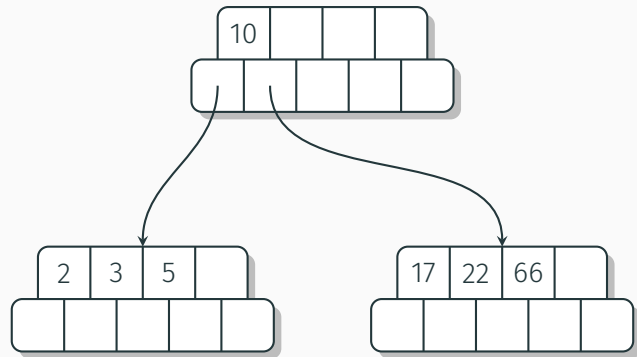
663/776

Příklad vkládání do B-stromu – vložení klíče 5



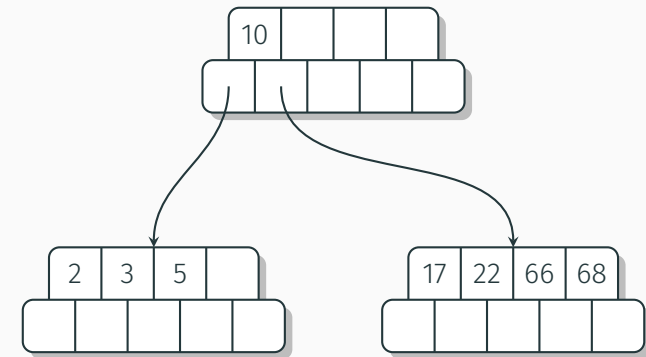
664/776

Příklad vkládání do B-stromu – vložení klíče 66



665/776

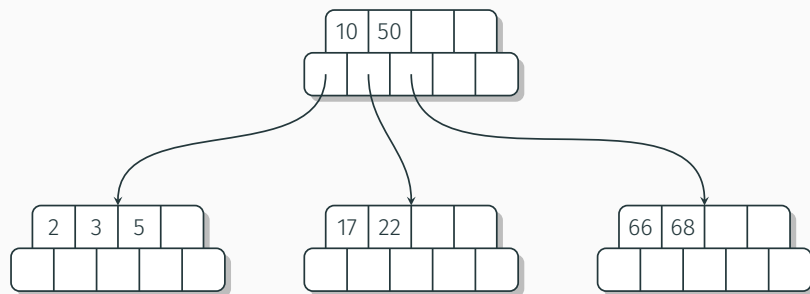
Příklad vkládání do B-stromu – vložení klíče 68



Stránka s klíči 17 až 68 je zcela zaplněna, vložení dalšího klíče do této stránky způsobí změnu struktury B-stromu.

666/776

Příklad vkládání do B-stromu – vložení klíče 50

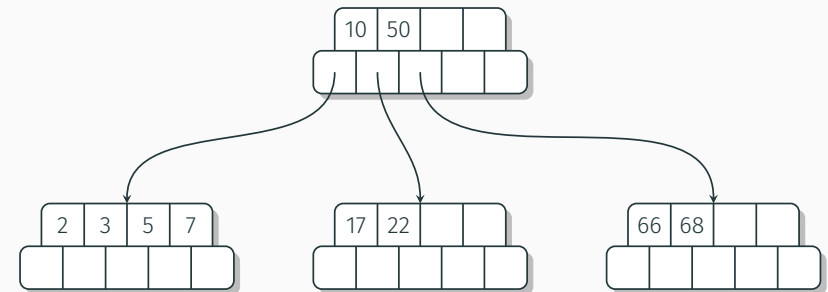


Vložení klíče 50 došlo:

1. k rozštěpení stránky a k přesunu poloviny klíčů do nové stránky a
2. současně byl nově vložený klíč 50, jakožto medián hodnot v původní stránce, přesunut do kořenové stránky.

667/776

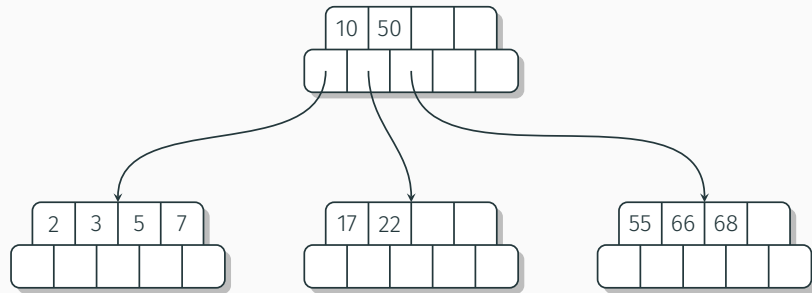
Příklad vkládání do B-stromu – vložení klíče 7



Stránka s klíči 2 až 7 je zcela zaplněna, vložení dalšího klíče do této stránky způsobí změnu struktury B-stromu.

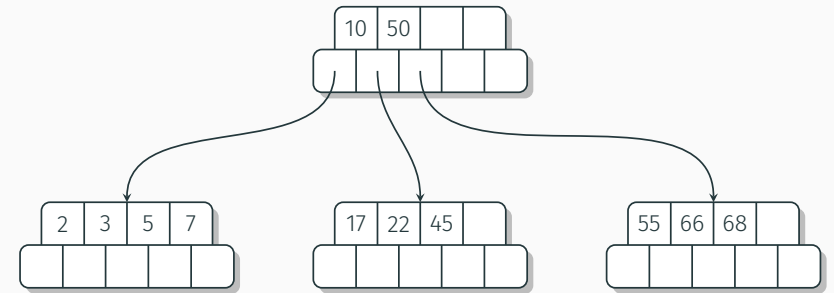
668/776

Příklad vkládání do B-stromu – vložení klíče 55



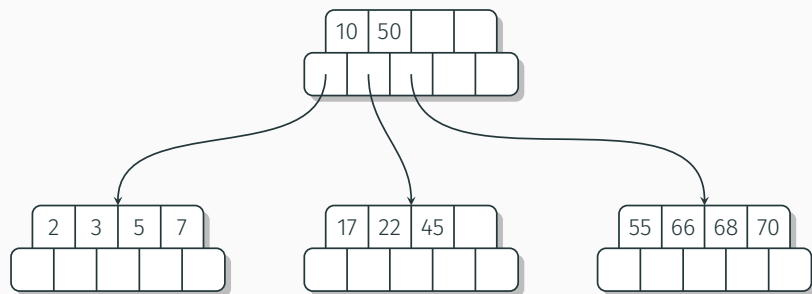
669/776

Příklad vkládání do B-stromu – vložení klíče 45



670/776

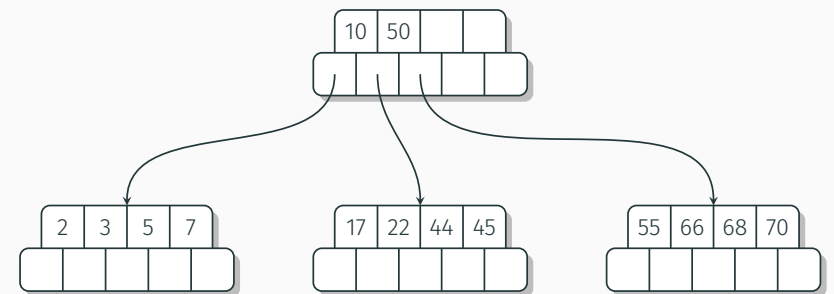
Příklad vkládání do B-stromu – vložení klíče 70



Stránka s klíči 55 až 70 je zcela zaplněna, vložení dalšího klíče do této stránky způsobí změnu struktury B-stromu.

671/776

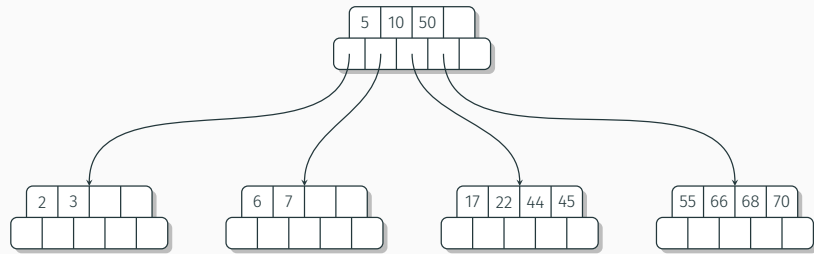
Příklad vkládání do B-stromu – vložení klíče 44



Všechny listové stránky jsou zcela zaplněny, vložení jakéhokoliv dalšího klíče způsobí změnu struktury B-stromu.

672/776

Příklad vkládání do B-stromu – vložení klíče 6

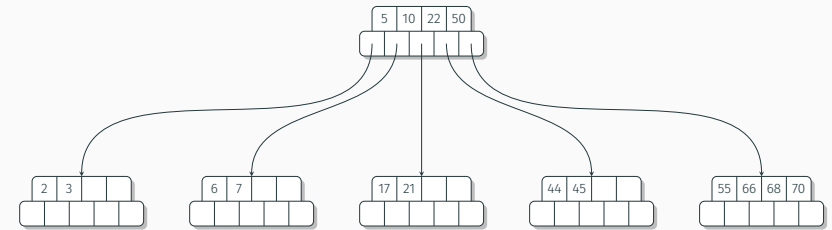


Vložení klíče 6 došlo:

1. k rozštěpení stránky a k přesunu poloviny klíčů do nové stránky a
2. současně k přesunu klíče 5 do kořenové stránky.

673/776

Příklad vkládání do B-stromu – vložení klíče 21

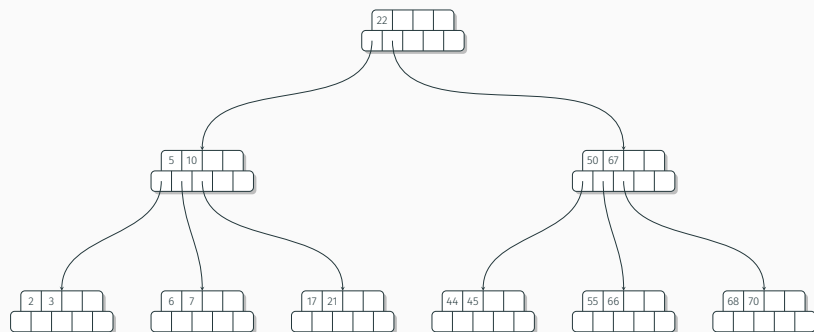


Vložení klíče 21 došlo:

1. k rozštěpení stránky a k přesunu poloviny klíčů do nové stránky a
2. k přesunu klíče 22 do rodičovské stránky.
3. Zároveň došlo k zaplnění kořenové stránky stromu.

674/776

Příklad vkládání do B-stromu – vložení klíče 67



675/776

Příklad vkládání do B-stromu – vložení klíče 67 (pokrač.)

Vložení klíče 67 došlo:

1. k rozštěpení stránky a k přesunu poloviny klíčů do nové stránky a
2. současně byl nově vložený klíč 67, jakožto medián hodnot v původní stránce, přesunut do rodičovské stránky.
3. Protože i tato stránka byla již zcela zaplněna, došlo k jejímu rozštěpení a tím pádem k vytvoření nové kořenové stránky stromu s jediným klíčem 22.

676/776

Příklad vkládání do B-stromu – vložení klíče 67 (pokrač.)

Poznámky

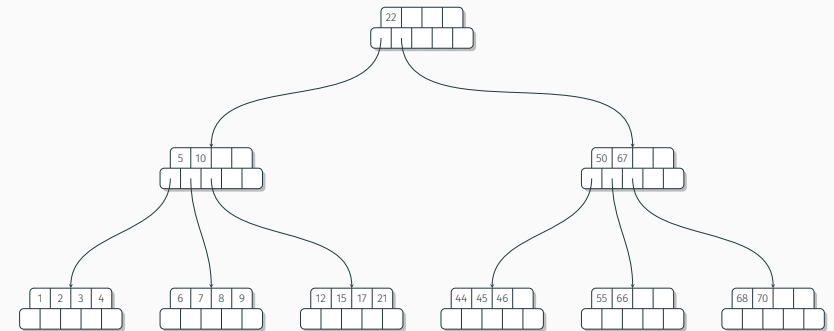
- V tomto okamžiku dosahuje faktor naplnění B-stromu minimální hodnoty $\approx 50\%$. V B-stromu je maximum volného místa pro vkládání dalších klíčů.
- Minimální zaplnění B-stromu ale naopak způsobí, že odebráním libovolného klíče dojde ke slučování stránek² B-stromu, včetně zrušení kořene a následného snížení výšky B-stromu.

²Viz další část prezentace

677/776

Příklad vkládání do B-stromu – vložení dalších klíčů

Do stromu byly dále vloženy klíče 1, 4, 8, 9, 12, 15 a 46. Na pořadí vložení klíčů, v tomto případě, nezáleží.



678/776

B-stromy – smazání klíče x

1. Nejprve je nutné klíč x ve stromu najít.
2. Označme stránku s klíčem x jako P .
3. Mohou nastat dva případy:
 - stránka P je **vnitřní stránka** stromu nebo
 - stránka P je **listová stránka**.

679/776

B-stromy – smazání klíče x z vnitřní stránky P

1. Klíč x ve stránce P nahradíme k němu nejbližším větším klíčem y .
2. Klíč y se musí nacházet v podstromu s klíči většími než x a zároveň je z těchto klíčů nejmenší, musí se tedy nacházet v listové stránce.
3. Smazání klíče x z vnitřní stránky stromu jsme tak převedli na smazání klíče y z listové stránky stromu.

680/776

B-stromy – smazání klíče x z listové stránky P

1. Klíč x vymažeme ze stránky P .
2. Pokud stránka P i potom obsahuje aspoň n klíčů, je proces smazání ukončen.
3. Pokud P potom obsahuje jen $n - 1$ klíčů, musíme chybějící jeden klíč doplnit.
 - 3.1 Zjistíme počet klíčů v sourozenci stránky P . Sourozence označíme jako S . Společného rodiče stránek P a S označíme jako R .
 - 3.2 Pokud je v S je **více než n** klíčů, potom
 - 3.2.1 nejbližší větší klíč než x přesuneme z R do P a
 - 3.2.2 nejmenší z klíčů v S přesuneme do R .
 - 3.3 Ve stránce S je **přesně n** klíčů, potom

681/776

B-stromy – smazání klíče x z listové stránky P (pokrač.)

- 3.3.1 klíče ze stránky S přesuneme do stránky P a dostáváme jednu stránku s $2n - 1$ klíči.
- 3.3.2 Stránku S zrušíme.
- 3.3.3 Ve stránce R je nyní jeden odkaz na stránku přebytečný. Nejbližší větší klíč než x přesuneme z R do stránky P , která nyní obsahuje přesně $2n$ klíčů.

682/776

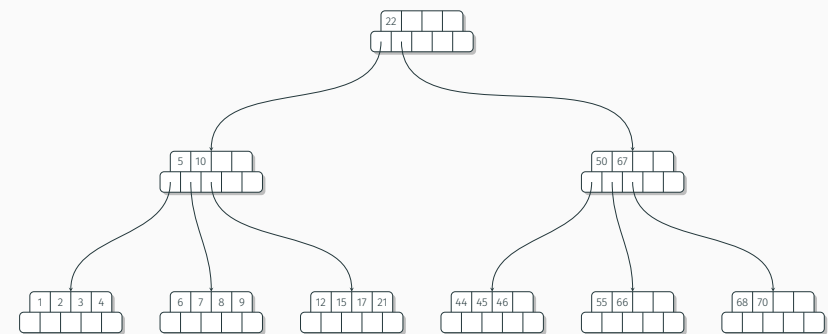
B-stromy – smazání klíče x z listové stránky P (pokrač.)

Poznámky

- Obvykle volíme sourozence s většími klíči než je x , tedy sourozence „vpravo“ od P . V předchozím výkladu jsme předpokládali tuto volbu.
- Lze však volit i sourozence s menšími klíči, čili „vlevo“ od P . Další postup je v tomto případě zrcadlovým obrazem k sourozenci „vpravo“.
- Proces přesunu klíčů z S do P a následné zrušení stránky S se nazývá **slučování stránek**.
- Proces slučování stránek může pokračovat postupně až ke kořeni stromu a může vést i k zániku současného kořene stromu. Novým kořenem stromu pak bude stránka vzešlá z procesu slučování. B-strom tak sníží svoji výšku.

683/776

Příklad mazání v B-stromu – výchozí B-strom

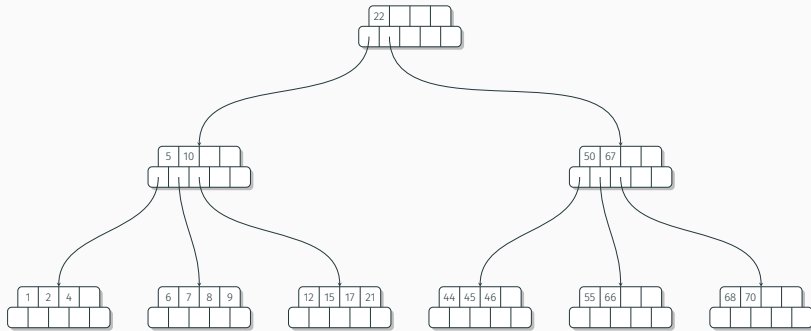


V tomto stavu jsme B-strom zanechali na konci příkladu vkládání klíčů do B-stromu. Nyní budeme klíče postupně z B-stromu mazat.

684/776

Příklad mazání v B-stromu – smazání klíče 3

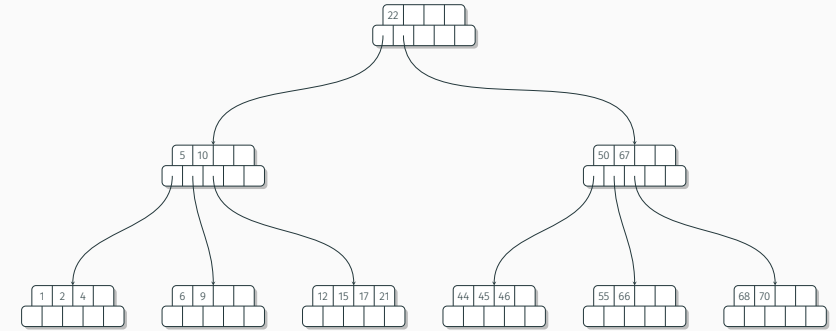
Klíč 3 se nachází v listové stránce, kde je dostatečný počet klíčů na to, abychom klíč 3 mohli jednoduše smazat. Tím dostáváme následující B-strom.



685/776

Příklad mazání v B-stromu – smazání klíčů 7 a 8

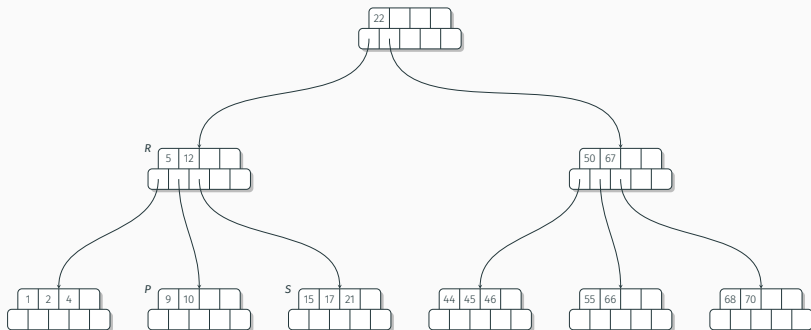
Stejným způsobem smažeme klíče 7, 8 a dostáváme



686/776

Příklad mazání v B-stromu – smazání klíče 6

1. Po smazání klíče 6 obsahuje stránka P $n - 1 = 1$ klíč, číslo 9.
2. Sourozenec S obsahuje více než n klíčů.
3. Nejbližší větší klíč než 6, tj. 10, přesuneme z R do P .
4. Nejmenší klíč z S , tj. 12, přesuneme do R .



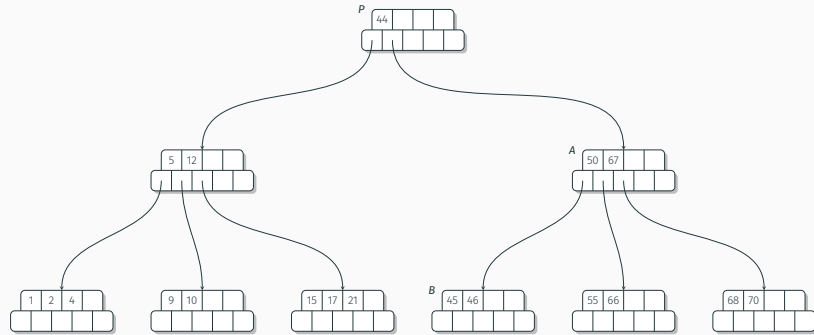
687/776

Příklad mazání v B-stromu – smazání klíče 22

1. Klíč 22 se nachází ve vnitřní stránce P , viz následující obrázek.
2. Nahradíme jej nejbližším větším klíčem – větší klíče než 22 jsou v podstromu s kořenem ve stránce A . Odtud pokračujeme do nejlevější listové stránky, v našem případě do B .
3. Vybereme nejmenší klíč v B , tj. 44.
4. Tím jsme převedli smazání klíče 22 na smazání klíče 44.
5. Po provedení všech operací, které odpovídají smazání 44 (v tomto případě se jedná jen o smazání 44 ze stránky B), nahradíme klíčem 44 klíč 22.

688/776

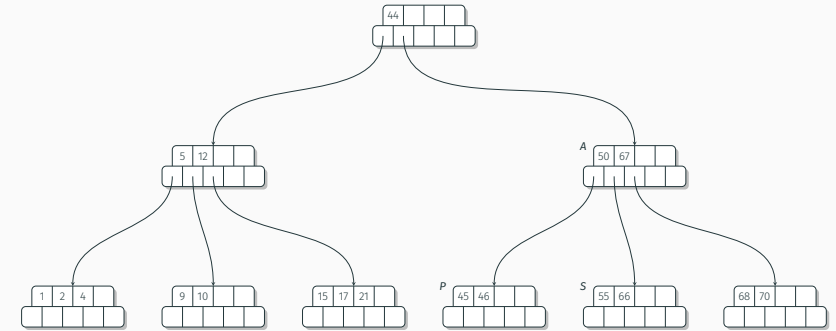
Příklad mazání v B-stromu – smazání klíče 22 (pokrač.)



689/776

Příklad mazání v B-stromu – smazání klíče 46, fáze I

Stav B-stromu před zahájením mazání



690/776

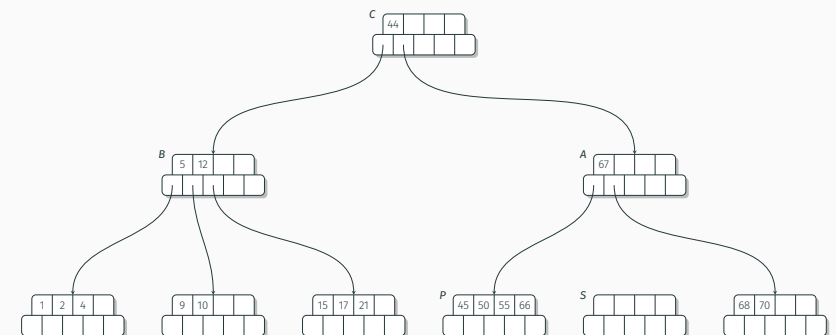
Příklad mazání v B-stromu – smazání klíče 46, fáze I (pokrač.)

- Po smazání klíče 46 obsahuje stránka **P** $n - 1$ klíčů, tj. pouze klíč 45.
- Sourozenec **S** obsahuje přesně než n klíčů, musíme slučovat stránky.
- Všechny klíče z **S** přesuneme do **P**.
- Stránka **P** je prvním potomkem stránky **A**, přesuneme tedy i první klíč z **A** do **P**.

691/776

Příklad mazání v B-stromu – smazání klíče 46, fáze I (pokrač.)

Výsledek fáze 1



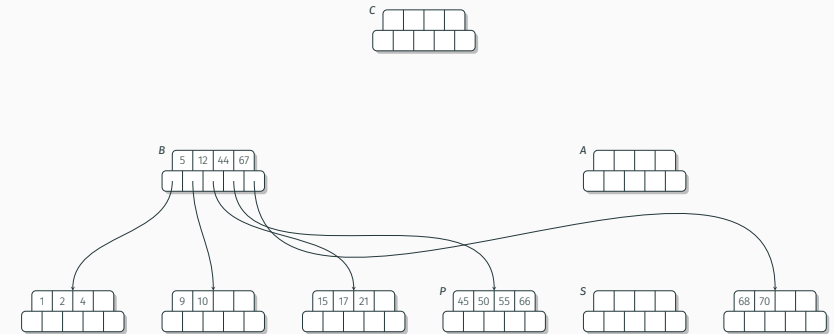
692/776

Příklad mazání v B-stromu – smazání klíče 46, fáze II

1. Ve stránce **A** zůstalo jen $n - 1$ klíčů, což odporuje definici B-stromu.
2. Sourozenec **B** obsahuje přesně n klíčů, musíme i na této úrovni provést slučování stránek.
3. Přesuneme klíč 67 ze stránky **A** do stránky **B**.
4. A stejně tak do **B** přesuneme i jeden klíč z rodičovské stránky **C**.
5. Tím došlo ke zrušení kořenové stránky a ke snížení výšky B-stromu.

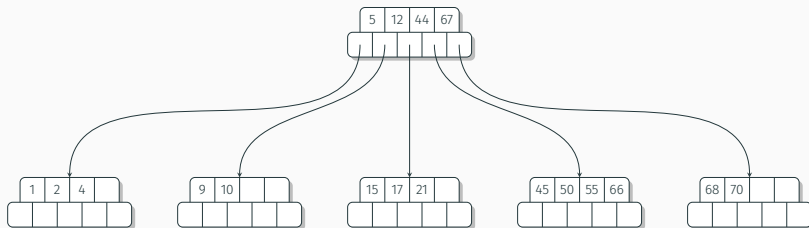
693/776

Příklad mazání v B-stromu – smazání klíče 46, fáze II (pokrač.)



694/776

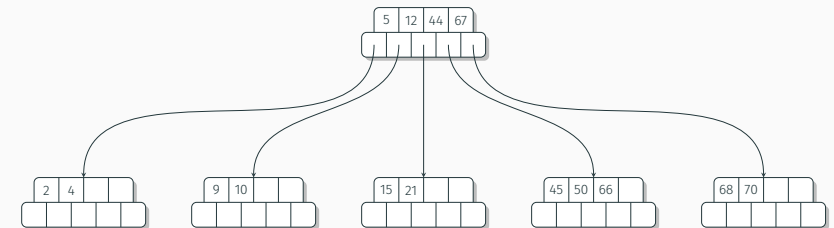
Příklad mazání v B-stromu – smazání klíče 46, výsledek



695/776

Příklad mazání v B-stromu – smazání klíčů 1, 17 a 55

Klíče 1, 17 a 55 se nachází v listových stránkách, kde je dostatečný počet klíčů na to, abychom je mohli jednoduše smazat.

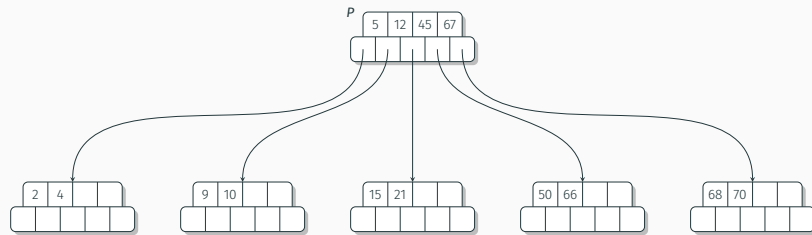


696/776

Příklad mazání v B-stromu – smazání klíče 44

1. Klíč 44 se nachází ve vnitřní stránce.
2. Nahradíme jej nejbližším větším klíčem, tj. klíčem 45.

Výsledný strom



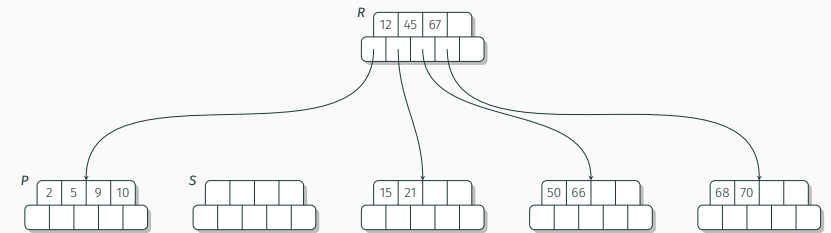
B-strom se nyní nachází ve stavu, kdy listové stránky jsou naplněny na minimální přípustnou úroveň. Smazání jakéhokoliv klíče způsobí sloučení stránek.

697/776

Příklad mazání v B-stromu – smazání klíče 4

1. Po smazání klíče 4 obsahuje stránka *P* pouze $n - 1$ klíčů.
2. Sourozenec *S* obsahuje přesně n klíčů.
3. Klíče z *S* přesuneme do *P*.
4. Do *P* přesuneme i klíč 5 z rodičovské stránky *R*, protože v *R* by jinak jeden odkaz na potomky přebýval.

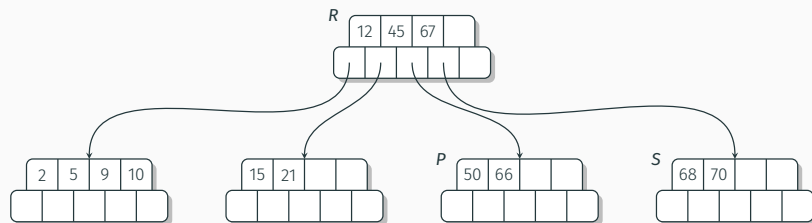
Výsledný strom



698/776

Příklad mazání v B-stromu – smazání klíče 45

Strom před smazáním klíče 45



1. Klíč 45 se nachází ve vnitřní stránce *R*.
2. Nahradíme jej nejbližším větším klíčem, tj. klíčem 50.
3. Tím jsme převedli smazání klíče 45 na smazání klíče 50.

699/776

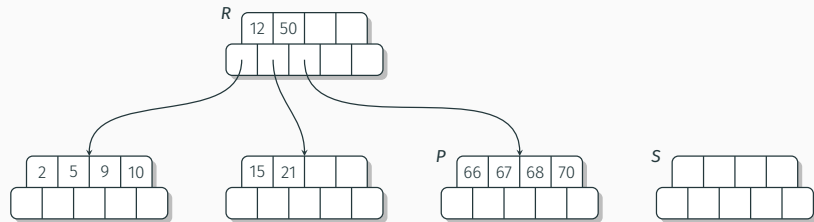
Příklad mazání v B-stromu – smazání klíče 45 (pokrač.)

4. Po smazání klíče 50 obsahuje stránka *P* pouze $n - 1$ klíčů.
5. Sourozenec *S* obsahuje přesně n klíčů.
6. Klíče z *S* přesuneme do *P*.
7. Do *P* přesuneme i nejbližší větší klíč než 45, tj. klíč 57, z rodičovské stránky *R*, protože v *R* by jinak jeden odkaz na potomky přebýval.

700/776

Příklad mazání v B-stromu – smazání klíče 45 (pokrač.)

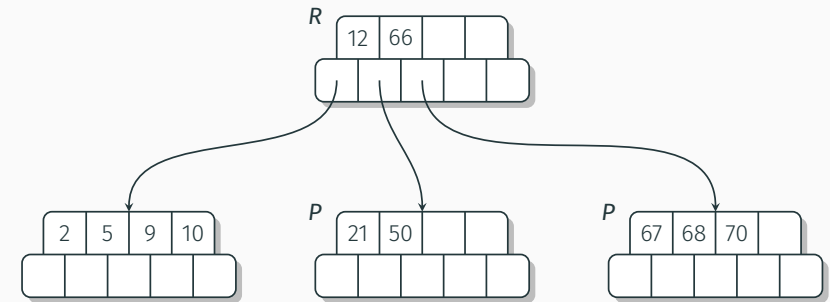
Výsledný strom



701/776

Příklad mazání v B-stromu – smazání klíče 15

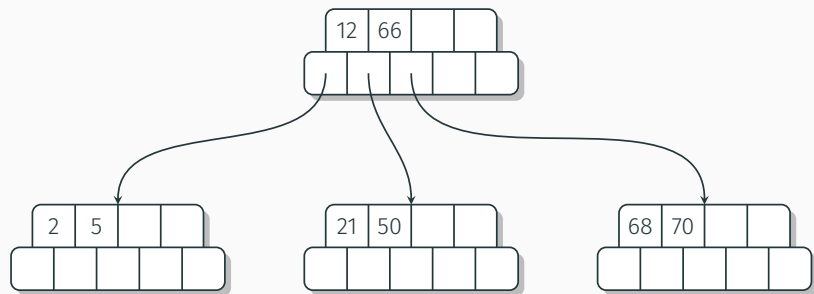
Po smazání 15 zůstalo ve stránce *P* pouze $n - 1$ klíčů. Musíme tedy přesunout přes stránku *R* jeden klíč ze stránky *S*.



702/776

Příklad mazání v B-stromu – smazání klíčů 9, 10 a 67

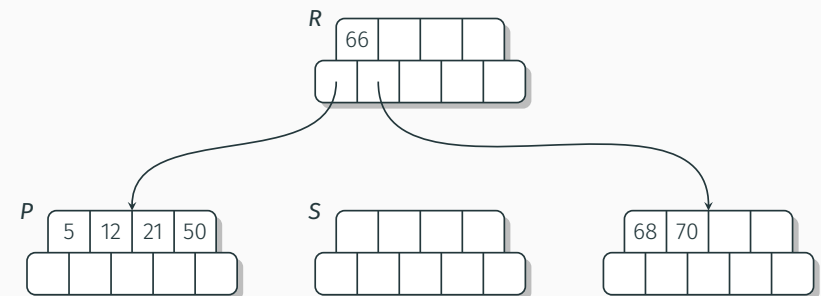
Smazání klíčů 9, 10 a 67 je velice jednoduché.



703/776

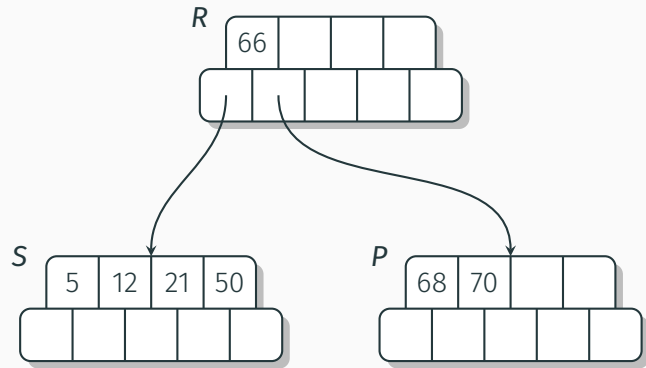
Příklad mazání v B-stromu – smazání klíče 2

Po smazání 2 zůstalo ve stránce *P* pouze $n - 1$ klíčů. Sourozenec *S* obsahuje n klíčů, dojde tedy ke slučování stránek.



704/776

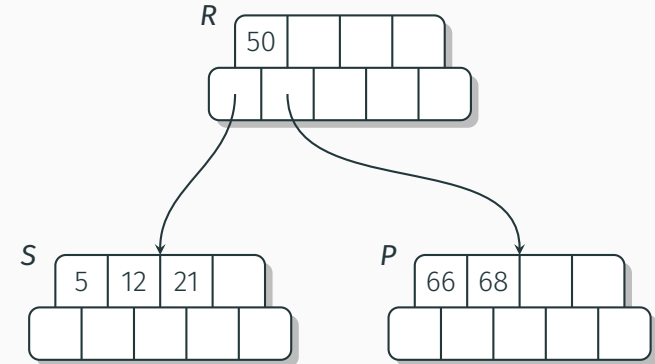
Příklad mazání v B-stromu – smazání klíče 70



705/776

Příklad mazání v B-stromu – smazání klíče 70 (pokrač.)

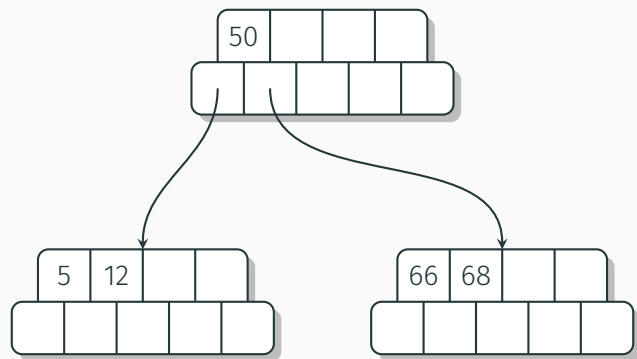
Po smazání 70 zůstalo ve stránce P pouze $n - 1$ klíčů. Sourozenec S obsahuje více n klíčů, dojde tedy k přesunu 66 z R do P a nejbližšího menšího klíče z S do R .



706/776

Příklad mazání v B-stromu – smazání klíče 21

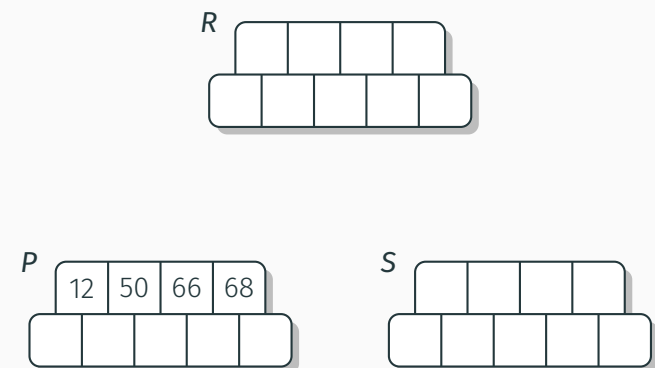
Smazání klíče 21 je velice jednoduché.



707/776

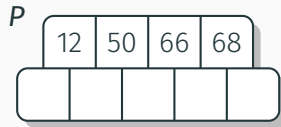
Příklad mazání v B-stromu – smazání klíče 5

Smazání klíče 5 je zřejmé.



708/776

Stránka **P** se stala novým kořenem, a současně jedinou stránkou, B-stromu.



Smazání klíčů 12, 50, 66 a 68 je už triviální záležitostí.

709/776

Děkuji za pozornost

Dynamické programování

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



710/776

Děkuji za pozornost

Hladové algoritmy

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



711/776

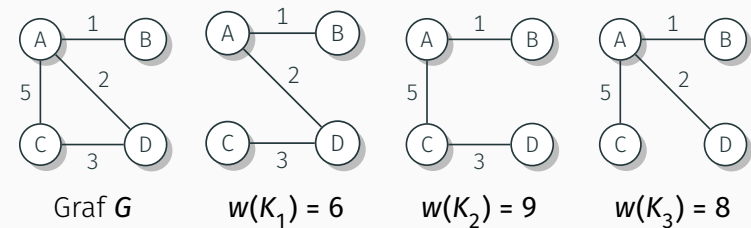
Minimální kostra grafu

712/776

Hladové algoritmy Minimální kostra grafu

Minimální kostra grafu – příklad

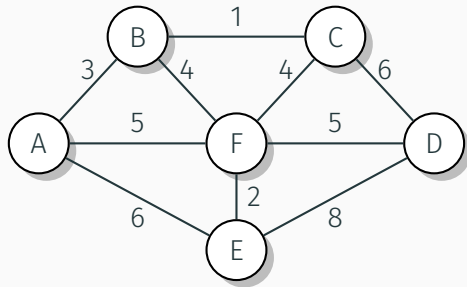
Graf G má celkem 3 kostry



Minimální kosterou je kostra K_1 .

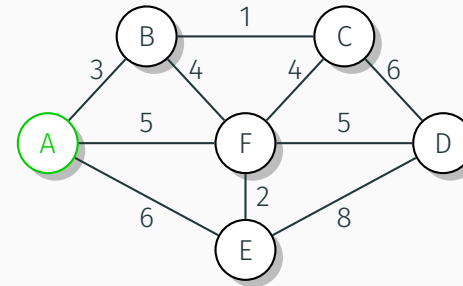
713/776

Minimální kostra grafu



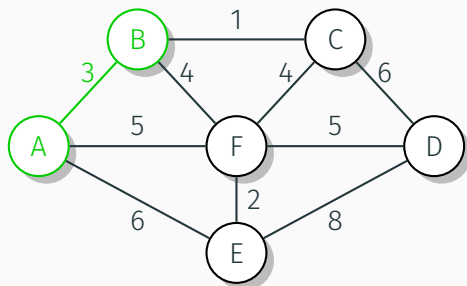
714/776

Minimální kostra grafu (pokrač.)



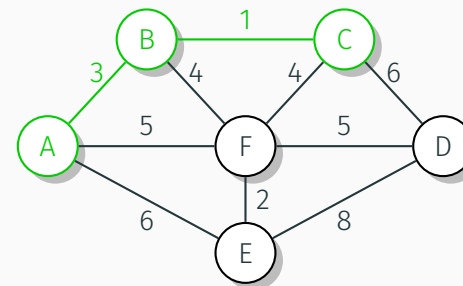
715/776

Minimální kostra grafu (pokrač.)



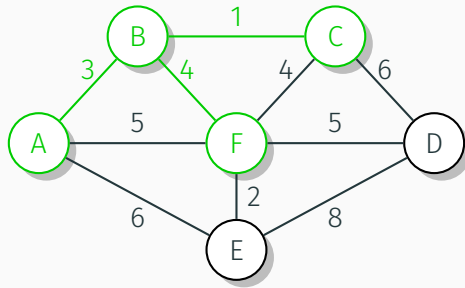
716/776

Minimální kostra grafu (pokrač.)



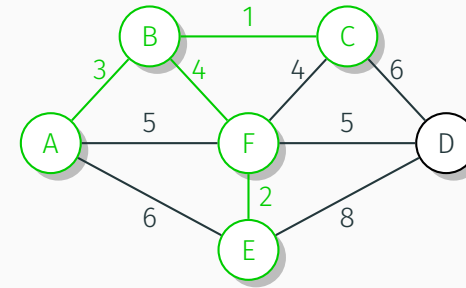
717/776

Minimální kostra grafu (pokrač.)



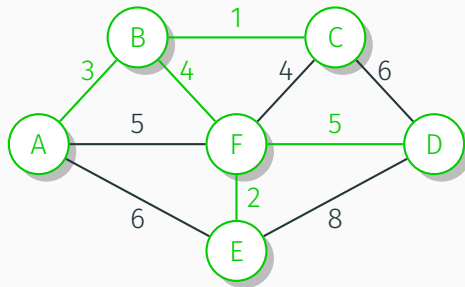
718/776

Minimální kostra grafu (pokrač.)



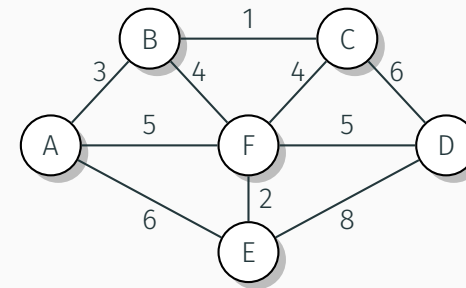
719/776

Minimální kostra grafu (pokrač.)



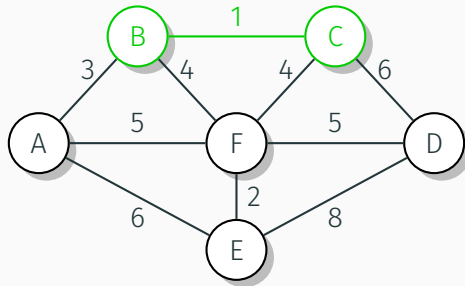
720/776

Minimální kostra grafu



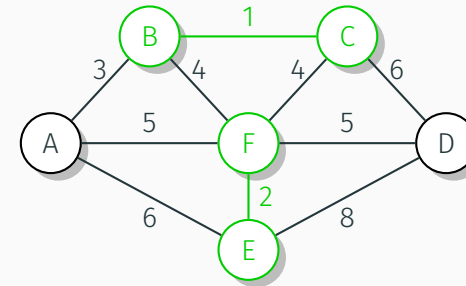
721/776

Minimální kostra grafu (pokrač.)



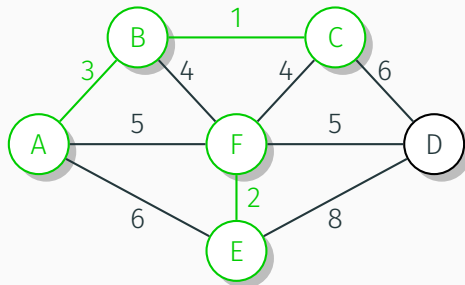
722/776

Minimální kostra grafu (pokrač.)



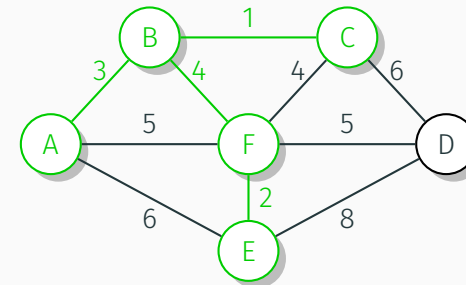
723/776

Minimální kostra grafu (pokrač.)



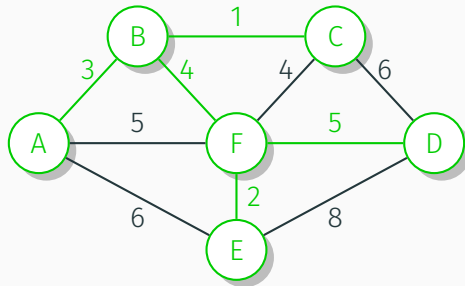
724/776

Minimální kostra grafu (pokrač.)



725/776

Minimální kostra grafu (pokrač.)



726/776

Hladové algoritmy
Dijkstrův algoritmus

Hladové algoritmy
Huffmanův kód

Děkuji za pozornost

Strategie řešení iterativním zlepšováním

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



727/776

Děkuji za pozornost

Meze možností algoritmického řešení problémů. P, NP a NP-úplné problémy.

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



728/776

Děkuji za pozornost

Zdolávání mezí možností algoritmického řešení problémů

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



729/776

Děkuji za pozornost

Literatura

1. **Železniční mapy ČR** [online]. Praha: SŽDC, 2019 [cit. 2019-11-15]. Dostupné z: <http://provoz.szdc.cz/porta1/Show.aspx?path=/Data/Mapy/kjr.pdf>.
2. LEVITIN, Anany. **Introduction to the Design and Analysis of Algorithms**. 3rd ed. Boston: Pearson, 2012. ISBN 978-0-13-231681-1.
3. CORMEN, Thomas H.; LEISERSON, Charles Eric; RIVEST, Ronald L.; STEIN, Clifford. **Introduction to algorithms**. Fourth edition. Cambridge, Massachusetts: The MIT Press, [2022]. ISBN 978-026-2046-305.

730/776

Literatura (pokrač.)

4. WRÓBLEWSKI, Piotr. **Algoritmy**. 1. vyd. Brno: Computer Press, 2015. ISBN 978-80-251-4126-7.
5. BEČVÁŘ, Jindřich. **Lineární algebra**. Vyd. 4. Praha: Matfyzpress, 2010. ISBN 978-80-7378-135-4.
6. ROHN, Jiří. **Lineární algebra a optimalizace**. Praha: Karolinum, 2004. ISBN 80-246-0932-0.
7. WIRTH, Niklaus. **Algoritmy a štruktúry údajov**. 1. vyd. Bratislava: Alfa, 1988. ISBN 063-030-87.

731/776

8. BAYER, Rudolf; MCCREIGHT, Edward Meyers. Organization and maintenance of large ordered indexes. *Acta Informatica*. 1972, roč. 1, č. 3, s. 173–189. ISSN 1432-0525. Dostupné z DOI: [10.1007/BF00288683](https://doi.org/10.1007/BF00288683).

732/776

Přílohy

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava



733/776

Přílohy Zásobník

Zásobník – možnosti implementace

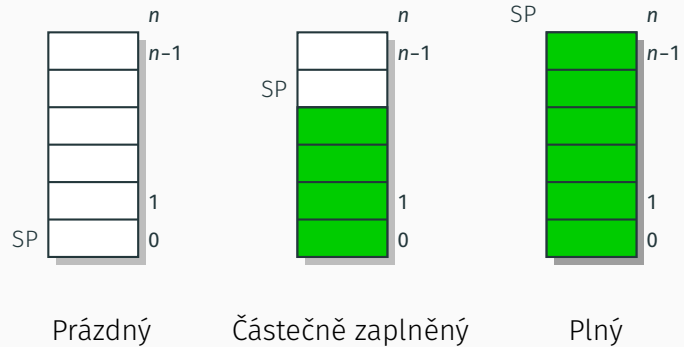
Pomocí pole

- pole délky n
- dno zásobníku na indexu 0
- stack pointer
 - index v poli, číslo typu *int*
 - neukazuje na poslední vložený prvek, nýbrž
 - ukazuje na **první volnou pozici** – dodržuje zvyklosti C++ o polích
 - prázdný zásobník – stack pointer roven 0
 - plný zásobník – stack pointer roven n

Pomocí spojových struktur – dynamická alokace paměti, pointerů

734/776

Zásobník – implementace pomocí pole



735/776

Zásobník – využití zásobníku

- volání funkcí (metod)
- vyhodnocování aritmetických výrazů
- odstranění rekurze
- zásobníkove orientované jazyky, například PostScript, PDF
- testování parity závorek, HTML/XML značek

736/776

Zásobník – testování parity závorek

Zadání

Máme dán matematický výraz *Expr*, který obsahuje kulaté, hranaté a složené závorky. Úkolem je implementovat funkci, která bude vracet *true*, pokud je výraz správně uzávorkován, jinak bude vracet *false*.

737/776

Zásobník – testování parity závorek

Ukázka

Výraz <i>Expr</i>	Výsledek
<code>()</code>	<i>true</i>
<code>)(</code>	<i>false</i>
<code>((()))</code>	<i>true</i>
<code>[(())]</code>	<i>true</i>
<code>{()()()[(())]}</code>	<i>true</i>
<code>[(())</code>	<i>false</i>
<code>[(]()</code>	<i>false</i>
<code>[[()]</code>	<i>false</i>

738/776

Princip řešení

- Při průchodu řetězcem se musí pravé závorky objevovat v přesně opačném pořadí než levé závorky.
- Levé závorky je nutné si pamatovat tak, aby aktuální, posledně načtená, levá závorka byla zpracována jako první; zatímco první načtená levá závorka musí přijít na řadu až jako poslední.
- Zkontrolovaný pár závorek vyřadíme z dalšího zpracování.
- Způsob pamatování levých závorek přesně odpovídá mechanismu práce zásobníku – LIFO.

739/776

```
Input : Výraz Expr obsahující závorky {}[]
Output: true pokud je Expr správně uzavřeno, jinak
        false
1 Init (S);
2 foreach ei ∈ Expr do
3   switch ei do
4     case (, [, { do
5       | Push (S, ei);
6     end
7     case ), ], } do
8       | if ¬IsEmpty(S) then
9         | if Pop (S) „není správná závorka“ then
10        | | return false
11        end
12        end
13        else
14        | return false
15        end
16        end
17  end
18 end
19 return IsEmpty (S)
```

740/776

Přílohy

Fronta

Fronta – možnosti implementace

Pomocí spojových struktur – dynamická alokace paměti, pointerů

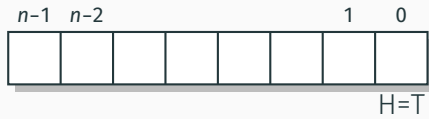
Pomocí pole

- hlava a ocas fronty jsou indexy do pole,
- jednoduché přidávání a odebrání – jak hlava, tak ocas se posunují dozadu.

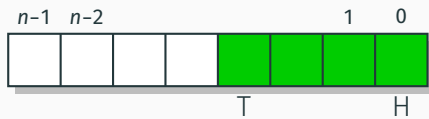
741/776

Fronta – implementace v poli

Prázdná fronta



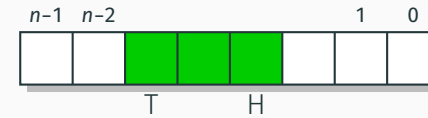
Vloženy 4 prvky



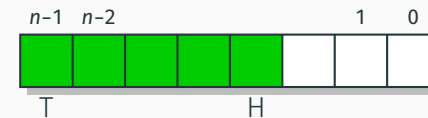
Vyjmuty 3 prvky, vloženy 2 prvky

742/776

Fronta – implementace v poli (pokrač.)



Vloženy 2 prvky



Výsledný stav:

- do fronty již nelze přidat další prvky,
- ale současně fronta není plná.

743/776

Fronta – implementace v poli (pokrač.)

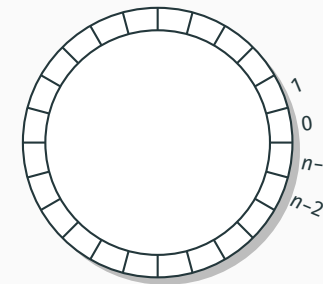
Možná řešení:

- přesouvat prvky na začátek pole – vyjmutí prvku již nebude operace s časovou složitostí $O(1)$ ale $O(n)$, bude docházet k masivnímu kopírování dat,

744/776

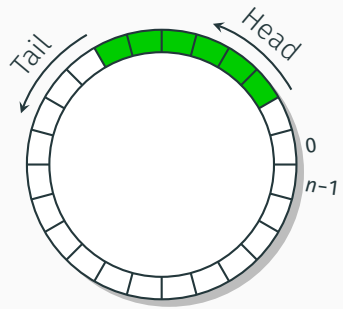
Fronta – kruhový buffer

Řešením je **kruhový buffer** (circular buffer) – začátek a konec pole se navzájem dotýkají.



745/776

Fronta – kruhový buffer



Charakteristika

- při vložení prvku se ocas fronty zvyšuje o jedničku,
- při odebírání prvku se hlava fronty zvyšuje o jedničku
- při překročení délky bufferu se hodnoty hlavy či ocasu vrací na první pozici tj. 0 – **modulární aritmetika**

Problém: Jak rozlišit prázdnou a zaplněnou frontu?

Řešení: V buffer je udržováno jedno prázdné místo. Ocas fronty ukazuje na první volnou pozici, ne na konec fronty.

746/776

Fronta – využití

- tisková fronta u sdílené tiskárny
- plánovač v operačním systému (více běžících procesů na jednoprocessorovém počítači ⇒ procesy se musí střídat)
- obsluha uživatelů na serverech obecně

747/776

Fronta – plánovač OS, cyklická obsluha (round robin)

Tato část OS přiděluje jednotlivým procesům sdílený procesor:

- vybere z fronty čekajících procesů první proces,
- obnoví registry procesoru,
- spustí obnovený proces,
- po uplynutí vymezeného časového kvanta, zahájí odebírání procesoru,
- uloží obsah registrů
- zařadí tento proces na konec fronty,
- vybere z fronty čekajících procesů první proces ...

748/776

Přílohy
Binární strom

Budování binárního stromu podle rekurzivní definice

- Výchozí množina uzlů $M = \{A, B, C, D, E, F, H, I, J, K, L\}$
- Postupně budeme aplikovat pravidla z definice binárního stromu.
- Uzly ve tvaru kruhu odpovídají již vytvořeným uzlům binárního stromu, viz **Pravidlo 2**.
- Uzly ve formě množin odpovídají dosud nevytvořeným podstromům, viz **Pravidlo 2**. Zde budeme rekurzivně pokračovat v tvorbě binárního stromu.
- Prázdné množiny odpovídají prázdným podstromům, viz **Pravidlo 1**.

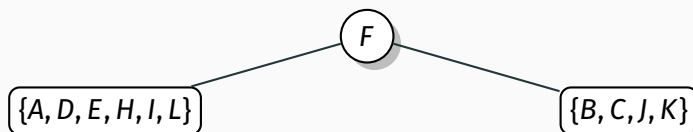
749/776

Budování binárního stromu podle rekurzivní definice

$\{A, B, C, D, E, F, H, I, J, K, L\}$

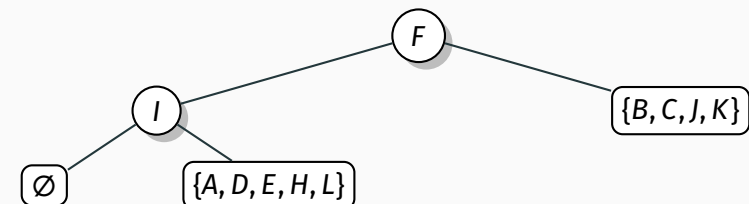
750/776

Budování binárního stromu podle rekurzivní definice



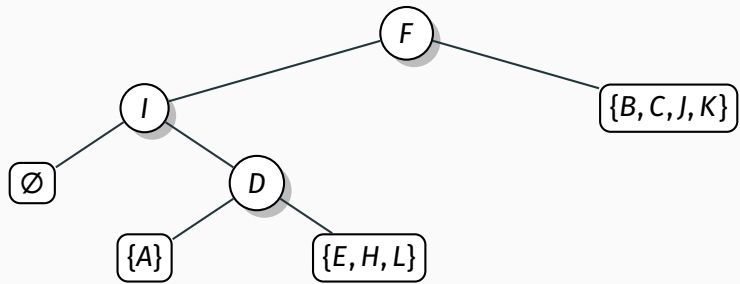
751/776

Budování binárního stromu podle rekurzivní definice



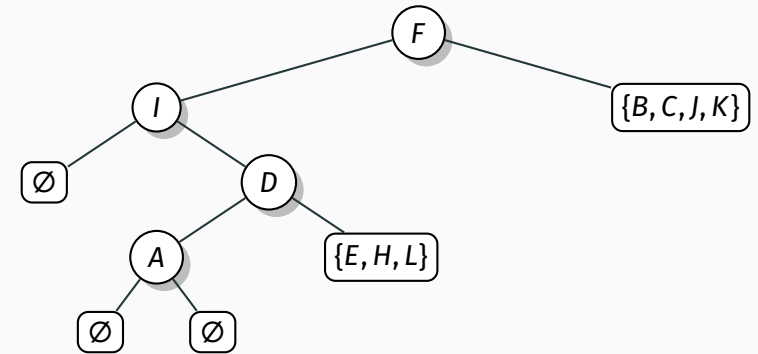
752/776

Budování binárního stromu podle rekurzivní definice



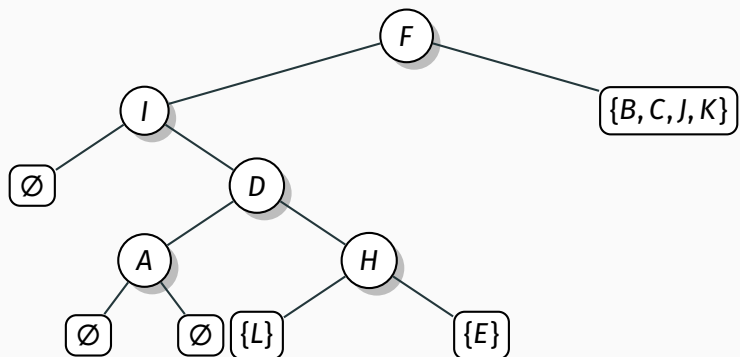
753/776

Budování binárního stromu podle rekurzivní definice



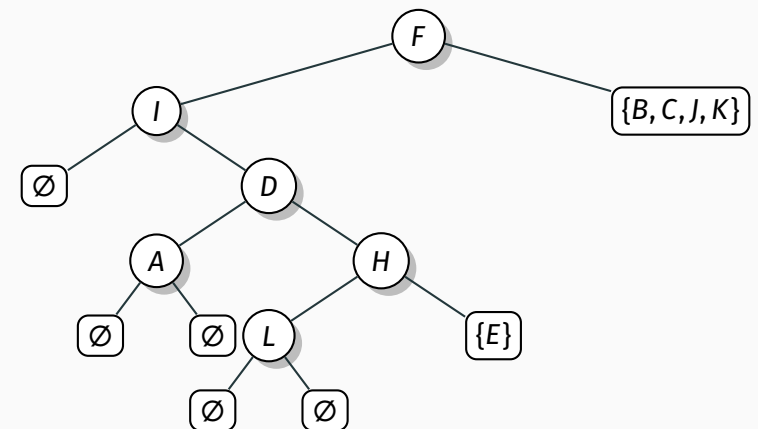
754/776

Budování binárního stromu podle rekurzivní definice



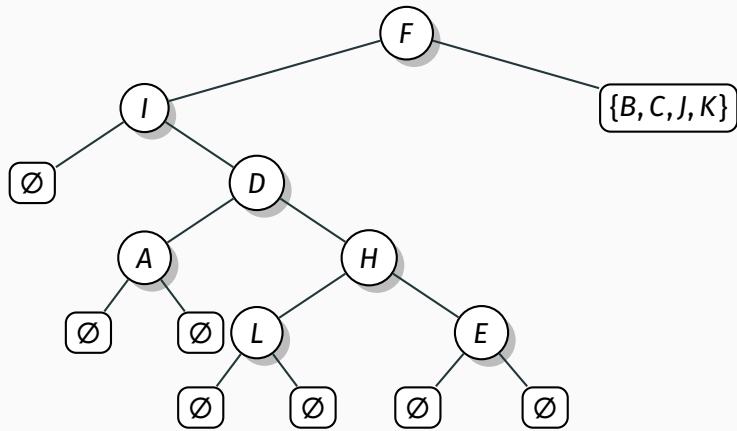
755/776

Budování binárního stromu podle rekurzivní definice



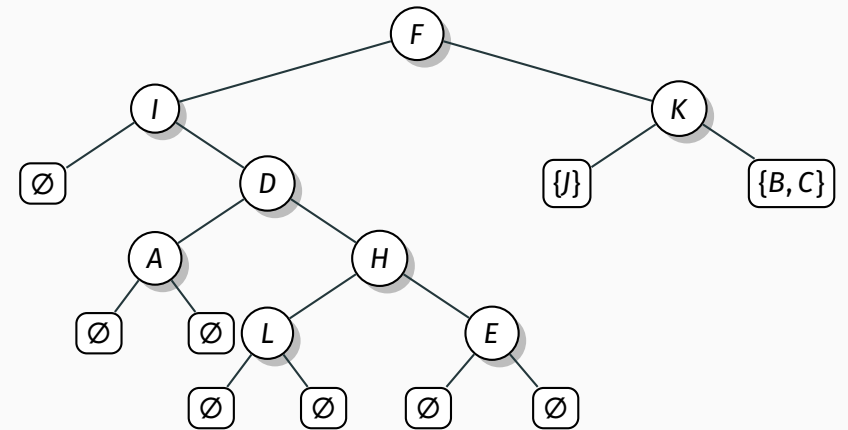
756/776

Budování binárního stromu podle rekurzivní definice



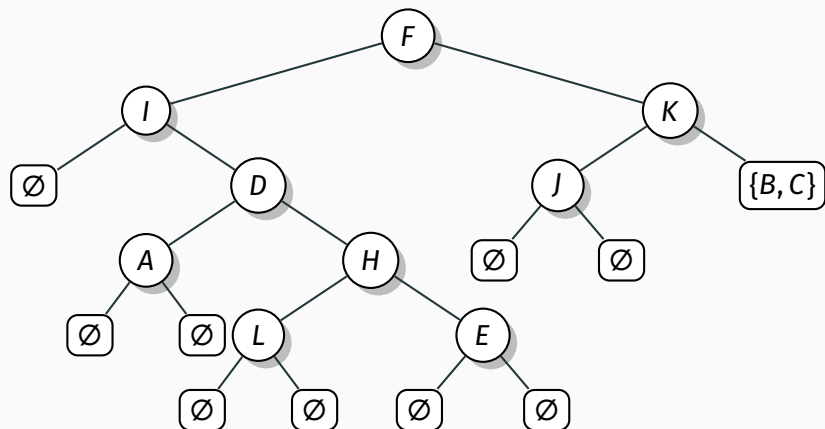
757/776

Budování binárního stromu podle rekurzivní definice



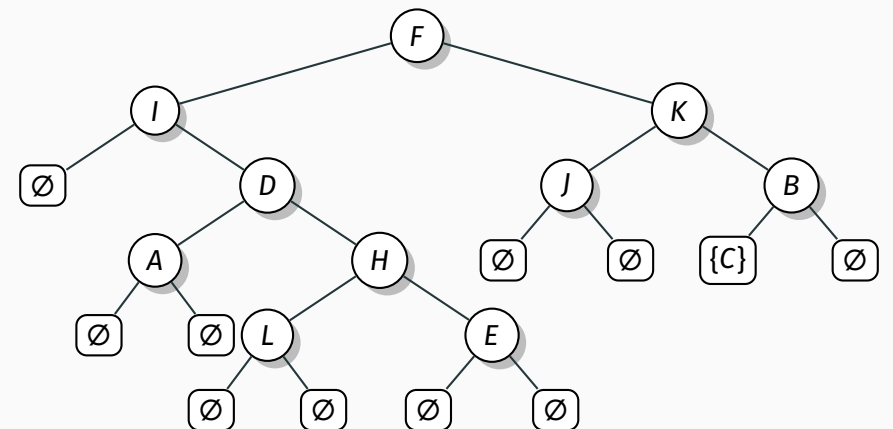
758/776

Budování binárního stromu podle rekurzivní definice



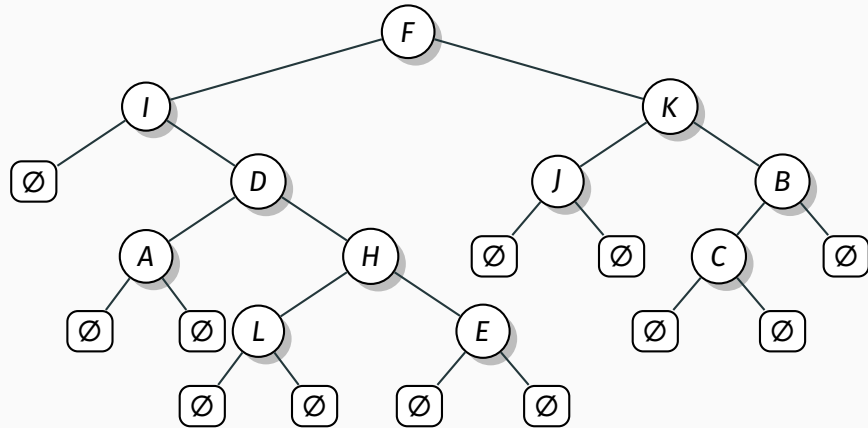
759/776

Budování binárního stromu podle rekurzivní definice



760/776

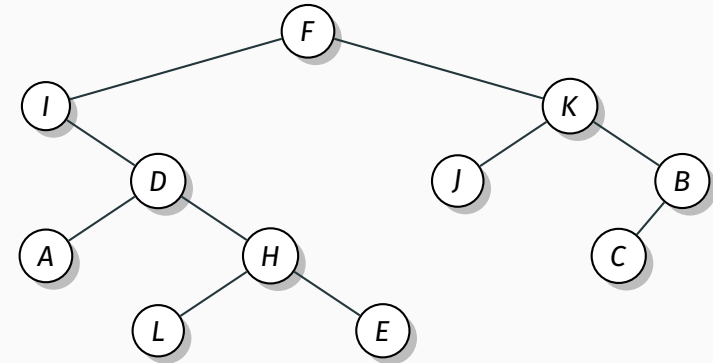
Budování binárního stromu podle rekurzivní definice



761/776

Budování binárního stromu podle rekurzivní definice

Odstraníme-li vizualizaci prázdných podstromů, zobrazených jako prázdné množiny, dostáváme výsledný binární strom v obvyklé podobě.



762/776

Binární vyhledávací strom – vložení klíče 37



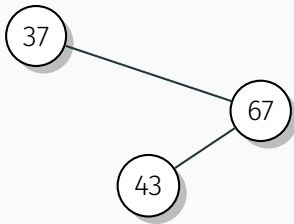
763/776

Binární vyhledávací strom – vložení klíče 67



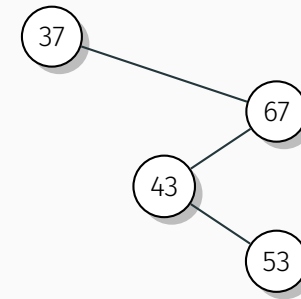
764/776

Binární vyhledávací strom – vložení klíče 43



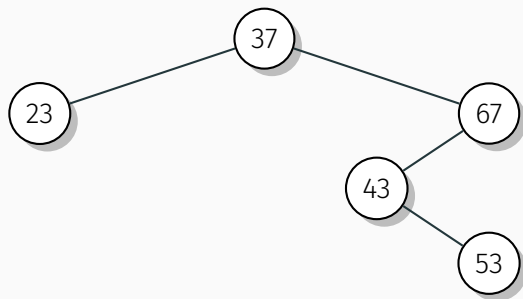
765/776

Binární vyhledávací strom – vložení klíče 53



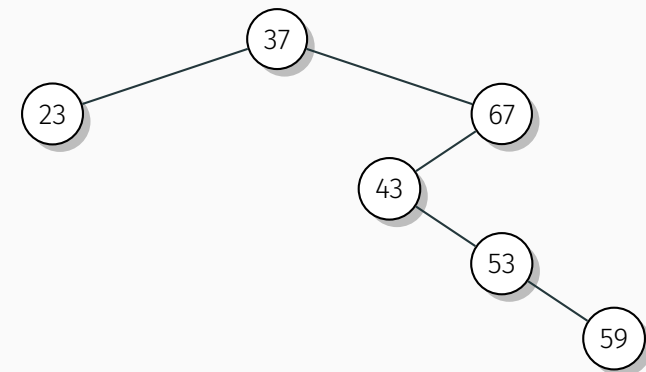
766/776

Binární vyhledávací strom – vložení klíče 23



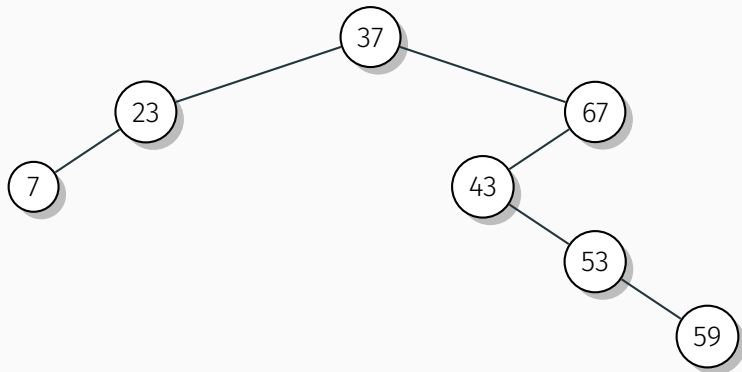
767/776

Binární vyhledávací strom – vložení klíče 59



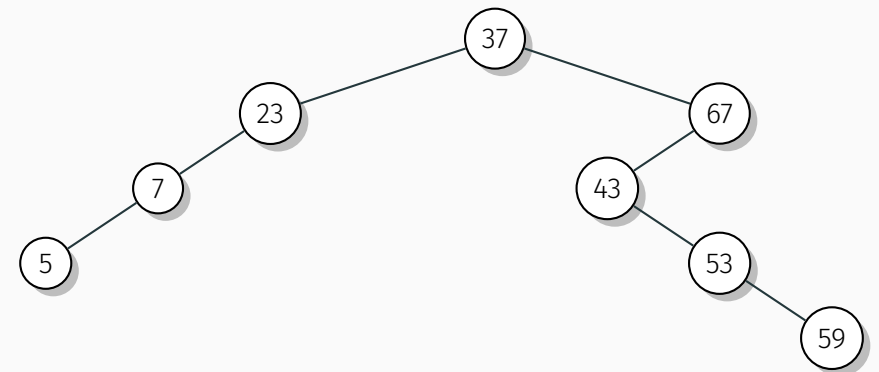
768/776

Binární vyhledávací strom – vložení klíče 7



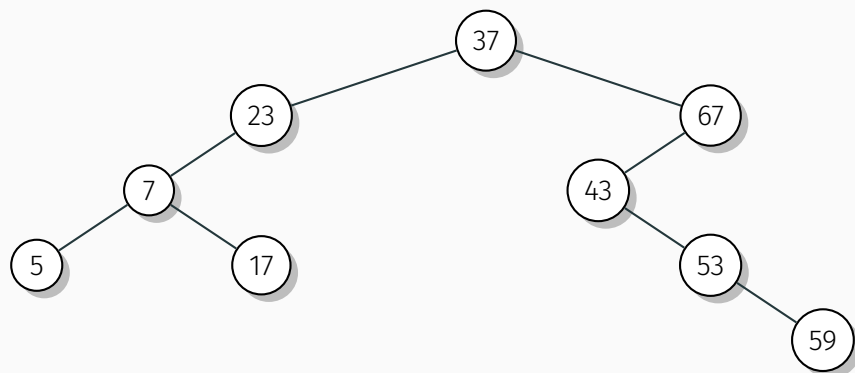
769/776

Binární vyhledávací strom – vložení klíče 5



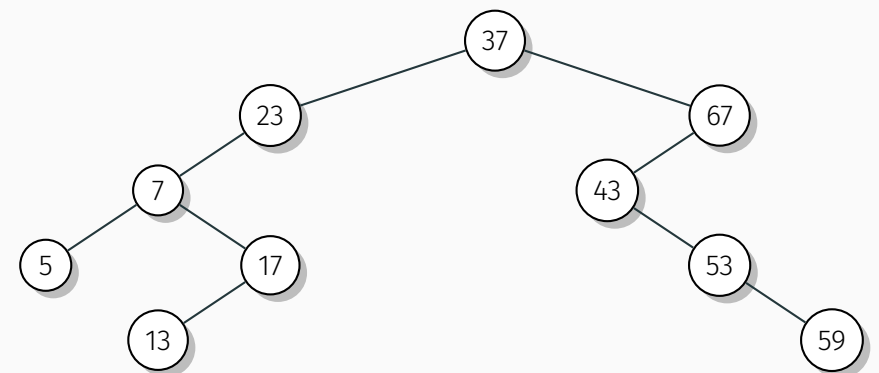
770/776

Binární vyhledávací strom – vložení klíče 17



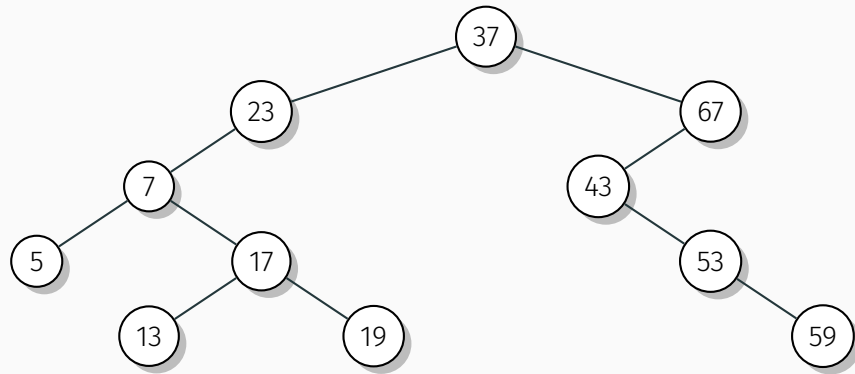
771/776

Binární vyhledávací strom – vložení klíče 13



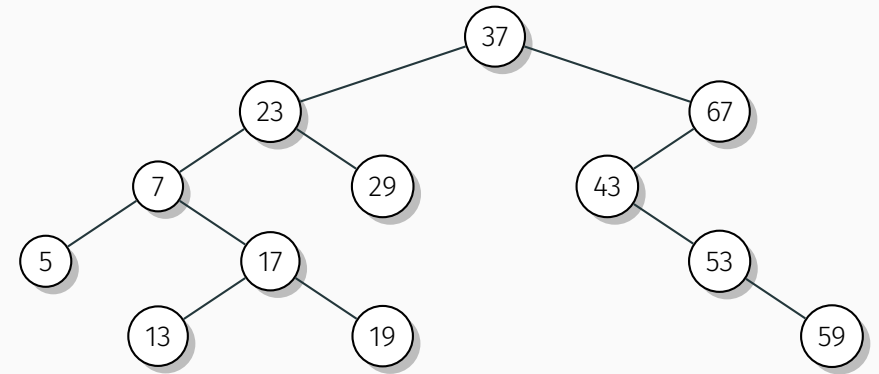
772/776

Binární vyhledávací strom – vložení klíče 19



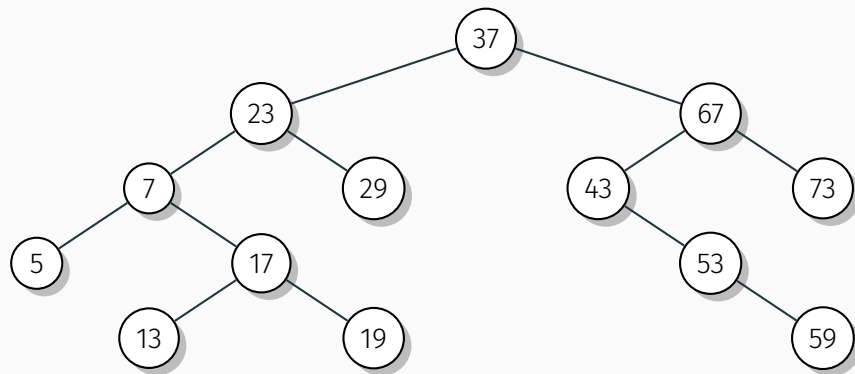
773/776

Binární vyhledávací strom – vložení klíče 29



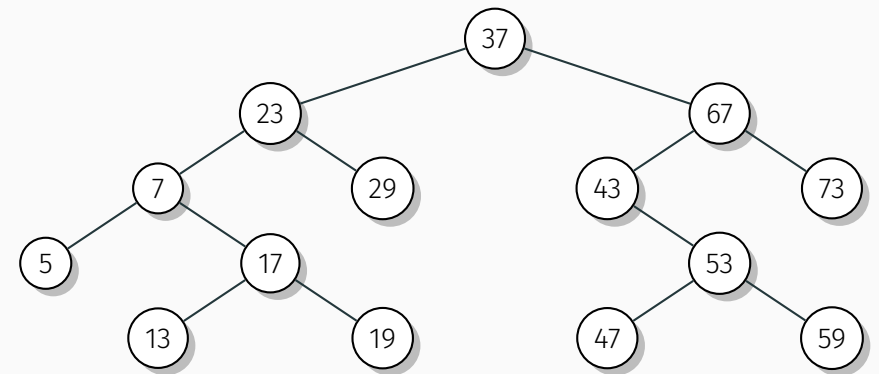
774/776

Binární vyhledávací strom – vložení klíče 73



775/776

Binární vyhledávací strom – vložení klíče 47



776/776