

Třídy L a NL

Jedním specifickým druhem algoritmů jsou algoritmy, které používají extrémně malé množství paměti — asymptoticky menší než n , kde n je velikost vstupu.

Speciálně se zde zaměříme na algoritmy s **logaritmickou prostorovou složitostí**, tj. s prostorovou složitostí $\mathcal{O}(\log n)$.

- Je zřejmé, že algoritmus, jehož paměťová složitost je menší než n , nemá dostatek paměti na to, aby měl v paměti uloženou celou vstupní instanci.
- U algoritmů, které pracují s takto malým množstvím paměti, se proto velikost vstupu **nezapočítává** do množství použité paměti.

Logaritmické množství paměti

U takových algoritmů se předpokládá, že jsou vykonávány nějakým druhem stroje (např. Turingovým strojem), který má:

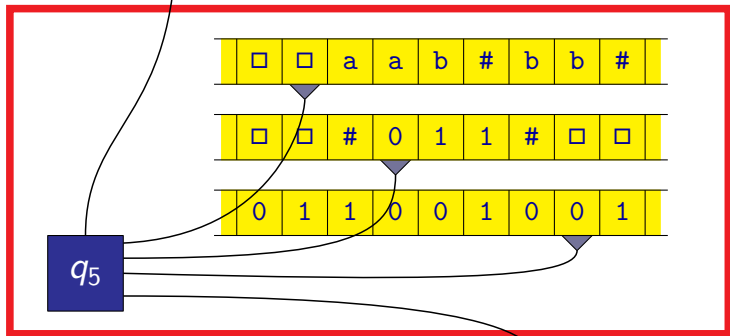
- **Vstupní pásku** — obsahuje vstupní slovo, které je ohraničeno zleva a zprava speciálními zarážkami '⊢' a '⊣', není možné na ni zapisovat (je read-only), má jednu hlavu, která se může posouvat oběma směry
- **Výstupní pásku** — je na ni možné pouze zapisovat (je write-only), není z ní možné číst, na začátku výpočtu je prázdná, pohyb hlavy je možný jen zleva doprava
- **Pracovní paměť** — je možné do ní zapisovat i z ní číst, např. u Turingových strojů má opět podobu jedné nebo více pásek

Množství použité paměti je dáno počtem bitů, které postačují pro uložení obsahu **pracovní paměti** během výpočtu.

Logaritmicke množství paměti

┌ b a b a a a a b a b b b a ─

Input



c a c c b d a

Output

Logaritmické množství paměti

Pokud je velikost vstupu n , $\mathcal{O}(\log n)$ bitů paměti v zásadě postačuje pouze na uložení nějakého fixního konečného počtu hodnot, kde každá z těchto hodnot zabírá maximálně $\mathcal{O}(\log n)$ bitů.

Pomocí k bitů je možné reprezentovat čísla v intervalu 0 až $2^k - 1$.

Logaritmický počet bitů tedy postačuje na reprezentaci čísla, jehož hodnota je omezena polynomem (tj. číslo, jehož maximální hodnota je $\mathcal{O}(n^c)$, kde c je nějaká konstanta).

Pomocí takových čísel je možné reprezentovat např:

- index políčka na vstupní pásce — de facto ukazatel do vstupních dat
- čítač, jehož hodnota je omezena polynomem
- u grafových algoritmů např. index vrcholu či hrany
- u algoritmů pracujících s maticemi např. číslo řádku či číslo sloupce

Naopak $O(\log n)$ bitů paměti **nepostačuje** pro uložení věcí jako třeba:

- Pokud je vstup tvořen posloupností n prvků, uložit si pro každý z těchto prvků alespoň 1 bit informace (např. nějaký příznak).
- U grafových algoritmů si např. pamatovat, které vrcholy byly navštíveny.

Příklad: Uvažujme problémy, kde vstup vypadá následovně:

Vstup: Dvojice čísel x a y , kde tato čísla jsou reprezentována binárně jako sekvence n bitů.

Algoritmy s logaritmickou prostorovou složitostí je možné počítat například následující věci:

- součet a rozdíl hodnot x a y (tj. hodnoty $x + y$ a $x - y$)
- součin hodnot x a y (tj. hodnotu $x \cdot y$)
- zjištění toho, zda platí $x = y$, $x < y$, $x \leq y$
- maximum či minimum (tj. hodnoty $\max(x, y)$ a $\min(x, y)$)

Příklad: Uvažujme problémy, kde vstup vypadá následovně:

Vstup: Posloupnost čísel $a_1, a_2, \dots, a_k$.

Řekněme, že n je celkový počet bitů nutný k zápisu čísel $a_1, a_2, \dots, a_k$. Algoritmy s prostorovou složitostí $\mathcal{O}(\log n)$ je možné spočítat například následující:

- setřídít prvky od nejmenšího po největší
- součet $a_1 + a_2 + \dots + a_k$

Poznámka: Všimněte si, že spočítání součtu čísel $a_1, a_2, \dots, a_k$ v prostoru $\mathcal{O}(\log n)$ není úplně triviální úloha, protože některá z čísel mohou mít více než $\mathcal{O}(\log n)$ bitů — vezměme si třeba případ, kdy počet čísel je $\sqrt{n}$ a každé z nich má $\sqrt{n}$ bitů.

Příklad: Také následující problém je možné řešit s prostorovou složitostí $\mathcal{O}(\log n)$:

Násobení matic

Vstup: Matice A a B , jejichž prvky jsou přirozená čísla.

Výstup: Matice $A \cdot B$.

Poznámka: Podobně jako v předchozím případě vezměme jako velikost vstupu n celkový počet bitů nutných k zápisu matic A a B (tj. k zápisu všech jejich prvků).

Může se stát, že některé prvky těchto matic mají více než $\mathcal{O}(\log n)$ bitů. Úloha tedy opět není tak jednoduchá, jak může na první pohled vypadat.

Také následující problém je možné snadno řešit deterministickým algoritmem s prostorovou složitostí $\mathcal{O}(\log n)$:

Vstup: Slovo w tvořené různými druhy závorek
($[_1,]_1, [_2,]_2, \dots, [_r,]_r$).

Otázka: Jedná se o správně uzávorkovanou posloupnost?

Správně uzávorkovanou posloupností se zde myslí posloupnost patřící do jazyka generovaného následující bezkontextovou gramatikou:

$$A \rightarrow \varepsilon \mid AA \mid [_1 A]_1 \mid [_2 A]_2 \mid \dots \mid [_r A]_r$$

Je zajímavé, že pro naprostou většinu polynomiálních redukcí použitých např. v důkazech **NP**-obtížnosti, **PSPACE**-obtížnosti, apod. různých problémů (jaké jsme si ukazovali v předchozích přednáškách či jaké jsou popsány v literatuře) platí, že příslušnou redukci je možné implementovat jako (deterministický) algoritmus pracující s logaritmickým množstvím paměti.

Takové redukce se označují jako **logspace redukce**.

Definice

Logspace redukce rozhodovacího problému A na rozhodovací problém B je deterministický algoritmus Alg s prostorovou složitostí $\mathcal{O}(\log n)$, který:

- Dostane jako vstup instanci x problému A .
- Vyprodukuje jako výstup instanci y problému B .
- V problému B bude pro vstup y odpověď **ANO** právě tehdy, když v problému A bude pro vstup x odpověď **ANO**.

Věta

Jestliže existuje:

- logspace redukce problému A na problém B a
- logspace redukce problému B na problém C

tak existuje i:

- logspace redukce problému A na problém C .

Důkaz: Předpokládejme, že:

- Alg_1 je logspace redukce problému A na problém B
- Alg_2 je logspace redukce problému B na problém C

Následující jednoduchá konstrukce, která funguje pro polynomiální redukce, zde nefunguje:

- na instanci x problému A aplikovat redukci Alg_1 , kterou dostaneme instanci y problému B , a na tu pak aplikovat redukci Alg_2

Problém je v tom, že takový algoritmus by sice byl redukce, ale ne logspace redukce:

- Instance y problému B vytvořená redukcí Alg_1 může být až polynomiálně velká vzhledem k velikosti původní instance x problému A
 - pracovní paměť logaritmické velikosti nebude stačit na uložení této instance y

Je třeba postupovat odlišně — algoritmus převádějící instanci problému A na instanci problému C bude pracovat následovně:

- Bude simulovat činnost algoritmu Alg_2 .
- Bude si pamatovat pozici jeho hlavy na vstupní pásce — tato pozice bude reprezentována binárně (stačí na ni $\mathcal{O}(\log n)$ bitů).
- Kdykoli algoritmus Alg_2 potřebuje přečíst symbol ze vstupu:
 - Spustí od začátku simulaci algoritmu Alg_1 .
 - V krocích, kde by Alg_1 zapsal symbol na výstup, není tento výstup nikam zapisován, je pouze inkrementováno počítadlo zapsaných symbolů.
 - V okamžiku, kdy by Alg_1 zapsal symbol na pozici, kterou potřebuje Alg_2 číst, je simulace Alg_1 ukončena, algoritmu Alg_2 je předán příslušný symbol a pokračuje se v simulaci činnosti algoritmu Alg_2 .

Není těžké si rozmyslet následující:

- Předpokládejme, že problém A je logspace převoditelný na problém B .
- Pokud bude existovat algoritmus s logaritmickou prostorovou složitostí řešící problém B , bude existovat i algoritmus s logaritmickou prostorovou složitostí řešící problém A .
- Pokud by tedy neexistoval algoritmus s logaritmickou prostorovou složitostí řešící problém A , nebude existovat ani algoritmus s logaritmickou prostorovou složitostí řešící problém B .

Logaritmické množství paměti

Pro následující problém není znám žádný **deterministický** algoritmus s logaritmickou prostorovou složitostí:

Dosažitelnost v grafu (Graph Reachability)

Vstup: Orientovaný graf $G = (V, E)$ se dvěma vyznačenými vrcholy s a t .

Otázka: Existuje v grafu G cesta z vrcholu s do vrcholu t ?

Očividně však existuje velmi jednoduchý **nedeterministický** algoritmus s prostorovou složitostí $\mathcal{O}(\log n)$ řešící tento problém:

- Pamatuje si vždy jen aktuální vrchol v a hodnotu čítače c .
- Inicializuje $v := s$ a $c := m - 1$, kde m je počet vrcholů grafu G .
- Nedeterministicky hádá cestu a s každým krokem snižuje čítač o 1.

Připomeňme definice následujících tříd složitosti:

Třída LOGSPACE (zkráceně L)

Třída **LOGSPACE** (či zkráceně **L**) je tvořena právě těmi rozhodovacími problémy, pro které existuje **deterministický** algoritmus s prostorovou složitostí $\mathcal{O}(\log n)$.

Třída NLOGSPACE (zkráceně NL)

Třída **NLOGSPACE** (či zkráceně **NL**) je tvořena právě těmi rozhodovacími problémy, pro které existuje **nedeterministický** algoritmus s prostorovou složitostí $\mathcal{O}(\log n)$.

- Je zjevné, že $L \subseteq NL$.
- Podobně jako v případě tříd P a NP se neví, jestli $P = NP$, také pro třídy L a NL se neví, zda $L = NL$.
(Zdá se spíše pravděpodobné, že tato rovnost neplatí, není to ale dokázáno.)

Příklad: Viděli jsme, že platí následující:

Problém „*Dosažitelnost v grafu*“ je ve třídě NL.

Vypadá to, že tento problém zřejmě není ve třídě L, není to ale jisté.

Definice

- Problém A je **NL-těžký**, jestliže každý problém z NL je logspace převeditelný na problém A .
- Problém A je **NL-úplný**, jestliže je NL -těžký a zároveň sám patří do třídy NL .
- Pokud by tedy libovolný NL -úplný problém byl řešitelný deterministickým algoritmem s logaritmickou prostorovou složitostí, znamenalo by to, že $L = NL$.
- Pokud bude existovat alespoň jeden problém, který je v NL , ale není v L , tak určitě nemůže existovat deterministický algoritmus s logaritmickou prostorovou složitostí pro žádný NL -těžký problém.

Věta

„Dosažitelnost v grafu“ je NL-úplný problém.

Myšlenka důkazu:

To, že tento problém patří do třídy NL jsme už viděli.

Je třeba ukázat, že každý problém A z NL je logspace převeditelný na problém dosažitelnosti v grafu.

Protože problém A patří do třídy NL, existuje nedeterministický stroj M (např. Turingův stroj nebo jiný typ stroje) s logaritmickou prostorovou složitostí, který ho řeší.

Počet všech možných konfigurací stroje M nad daným vstupem x velikosti n bude polynomiální.

V logaritmickém prostoru je možné vygenerovat graf, kde:

- **vrcholy** — konfigurace stroje M
- **hrany** — přechody mezi těmito konfiguracemi

Pomocí logspace redukcí z tohoto problému se dá ukázat NL-obtížnost řady dalších problémů.

Příklady některých NL-úplných problémů:

2-UNSAT

Vstup: Booleovská formule φ v konjunktivní normální formě, kde každá klauzule obsahuje právě 2 literály.

Otázka: Je formule φ nesplnitelná (tj. je to kontradikce)?

Přijímání slova NKA

Vstup: Nedeterministický konečný automat $\mathcal{A}$ a slovo w .

Otázka: Přijímá automat $\mathcal{A}$ slovo w (tj. platí $w \in \mathcal{L}(\mathcal{A})$)?

Dosažitelné neterminály v bezkontextové gramatice

Vstup: Bezkontextová gramatika $\mathcal{G} = (\Pi, \Sigma, S, P)$
a neterminál $B \in \Pi$.

Otázka: Existují nějaká $\alpha, \beta \in (\Pi \cup \Sigma)^*$ taková, že $S \Rightarrow^* \alpha B \beta$?

Prázdnot jazyka přijímaného DKA

Vstup: Deterministický konečný automat $\mathcal{A}$.

Otázka: Je $\mathcal{L}(\mathcal{A}) = \emptyset$?

Univerzalita DKA

Vstup: Deterministický konečný automat $\mathcal{A}$.

Otázka: Je $\mathcal{L}(\mathcal{A}) = \Sigma^*$?

Ekvivalence DKA

Vstup: Deterministické konečné automaty $\mathcal{A}_1$ a $\mathcal{A}_2$.

Otázka: Je $\mathcal{L}(\mathcal{A}_1) = \mathcal{L}(\mathcal{A}_2)$?

Generování prvku asociativní operací

Vstup: Konečná množina X , asociativní binární operace $\circ$ na množině X (zadaná tabulkou specifikující hodnotu $x \circ y$ pro každou dvojici $x, y \in X$), podmnožina $S \subseteq X$ a prvek $t \in X$.

Otázka: Je možné prvek t vygenerovat z prvků množiny S ?

Prvek t je možné vygenerovat z prvků množiny S , jestliže existuje posloupnost $x_1, x_2, \dots, x_k$ prvků z množiny S taková, že

$$t = x_1 \circ x_2 \circ \dots \circ x_k$$

Podobně jako se ke třídě NP zavádí třída co-NP, je ke třídě NL možné zavést třídu co-NL.

Definice

Třída co-NL je tvořena doplňkovými problémy k problémům ze třídy NL.

Zatímco v případě tříd NP a co-NP se neví, jestli platí $NP = co-NP$ (a spíše se zdá, že $NP \neq co-NP$), v případě tříd NL a co-NL se ví, že platí následující poměrně překvapující výsledek:

$$NL = co-NL$$

Věta — Immerman-Szelepcsényi (1987)

Doplňkový problém k problému „*Dosažitelnost v grafu*“ je ve třídě NL.

Myšlenka důkazu: Pro daný orientovaný graf $G = (V, E)$ a dvojici vrcholů s a t je třeba nedeterministickým algoritmem s logaritmickou prostorovou složitostí ověřit, že **neexistuje** cesta z s do t .

Označme S množinu všech vrcholů dosažitelných z s :

$$S = \{v \in V \mid \text{v } G \text{ existuje cesta z } s \text{ do } v\}$$

Je třeba ověřit, že $t \notin S$.

Množinu S určitě není možné držet v paměti — $\mathcal{O}(\log n)$ bitů by na to nestačilo.

Nedosažitelnost v grafu v NL

Řekněme, že bychom však znali přesnou velikost množiny S .

To by bylo jedno číslo $m = |S|$, na jehož uložení by $\mathcal{O}(\log n)$ bitů stačilo.

Nedeterministický algoritmus, který by toto číslo znal, by ověřil, že $t \notin S$, následujícím způsobem:

- Postupně by pro všechny vrcholy $v \in V - \{t\}$ hádal, zda pro daný vrchol v platí $s \in S$ nebo ne.

Pro ty vrcholy v , pro které by hádal, že $v \in S$ platí, by ověřil, že tomu tak skutečně je — hádal by příslušnou cestu z s do v .

Pokud by při tomto ověřování byl neúspěšný, skončí neúspěchem (dá odpověď **NE**).

- Zároveň by počítal počet těch vrcholů, u kterých hádal, že patří do množiny S .
- Nakonec by zkontroloval, zda je tento počet roven m , a dal odpověď **ANO** právě tehdy, když tomu tak je.

Zbývá tedy vyřešit, jak spočítat velikost množiny S .

Pokud by na množství paměti nezáleželo, mohli bychom počítat množinu S postupem, kdy bychom postupně počítali následující posloupnost množin

$$S_0, S_1, S_2, \dots, S_{n-2}, S_{n-1}$$

kde pro $i = 0, 1, \dots, n-1$:

$$S_i = \{v \in V \mid \text{v } G \text{ existuje cesta z } s \text{ do } v \text{ délky nejvýše } i\}$$

Zjevně platí:

- $S_0 = \{s\}$
- $S_{i+1} = S_i \cup \{v \in V \mid \exists u \in S_i : (u, v) \in E\}$
- $S = S_{n-1}$

Nedosažitelnost v grafu v NL

Algoritmus bude postupně pro $i = 0, 1, \dots, n - 1$ počítat velikosti množin S_i , tj. hodnoty

$$m_0, m_1, m_2, \dots, m_{n-2}, m_{n-1}$$

kde $m_i = |S_i|$.

V každém okamžiku si bude pamatovat jen aktuální hodnotu m_i .

Začne s hodnotou $m_0 = 1$.

Pro $i = 0, 1, \dots, n - 2$ spočítá hodnotu m_{i+1} z m_i následovně:

- Bude probírat všechny vrcholy $v \in V - \{s\}$.
- Pro každý z těchto vrcholů v zjistí, zda $v \in S_{i+1}$, následovně:
Vyzkouší všechny $u \in V$ takové, že $(u, v) \in E$.
Pro každý z nich zjistí, zda $u \in S_i$ (viz dále).
Pokud najde alespoň jeden vrchol u , pro který platí $(u, v) \in E$ a $u \in S_i$, zvýší počítadlo vrcholů v S_{i+1} o 1.

Se znalostí hodnoty m_i je možné testovat, zda $u \in S_i$, následujícím způsobem:

- Pro všechny vrcholy $w \in V$ postupně hádat, zda $w \in S_i$ (tj. zda existuje cesta z s do w délky nejvýše i).
U těch w , kde algoritmus uhodne, že platí $w \in S_i$, to musí ověřit nedeterministickým uhodnutím příslušné cesty délky nejvýše i .
(Pokud není při ověření úspěšný, končí celkovým neúspěchem.)
- Pro daný vrchol u si pamatuje, co pro něj uhodl, zda $u \in S_i$ nebo $u \notin S_i$.
- Počítat vrcholy, u kterých bylo uhodnuto, že patří S_i .
- Nakonec se ověří, zda je tento počet roven m_i .
Pokud ne, skončí se neúspěchem.

Nedosažitelnost v grafu v NL

Stejnou myšlenku je možné použít pro libovolný nedeterministický algoritmus, který řeší nějaký rozhodovací problém A a má prostorovou složitost $S(n)$ (kde $S(n) \in \Omega(\log n)$).

Pro takový algoritmus můžeme uvažovat graf jeho konfigurací a výše uvedeným způsobem k němu vytvořit nedeterministický algoritmus s prostorovou složitostí $\mathcal{O}(S(n))$, který bude ověřovat, že v původním grafu není žádná přijímající konfigurace dosažitelná z počáteční konfigurace.

Tento nedeterministický algoritmus tak bude řešit doplňkový problém k problému A s prostorovou složitostí $\mathcal{O}(S(n))$.

Dostáváme tak následující:

Důsledek

Pro každou funkci $S(n) \geq \log n$ platí

$$\text{NSPACE}(S(n)) = \text{co-NSPACE}(S(n))$$

NL-úplné problémy

Z předchozího je zřejmé, že každý NL-úplný problém je zároveň co-NL-úplný a naopak.

Je také zřejmé, že pro každý problém A platí, že A je NL-úplný problém právě tehdy, když doplňkový problém problému A je NL-úplný.

Pro všechny dříve uvedené NL-úplné problémy tedy platí, že i jejich doplňkové problémy jsou NL-úplné.

Například z NL-úplnosti problému 2-UNSAT plyne NL-úplnost problému 2-SAT:

2-SAT

Vstup: Booleovská formule φ v konjunktivní normální formě, kde každá klauzule obsahuje právě 2 literály.

Otázka: Je formule φ splnitelná?

P-úplné problémy

P-úplné problémy

Připomeňme, že P (resp. $PTIME$) je třída rozhodovacích problémů řešitelných algoritmem s **polynomiální** časovou složitostí.

Definice

- Problém A je **P-těžký**, jestliže každý problém z P je logspace převeditelný na problém A .
 - Problém A je **P-úplný**, jestliže je P -těžký a zároveň sám patří do třídy P .
-
- Je jasné, že $NL \subseteq P$.
 - Zda platí $NL = P$, není známo. (Zdá se, že spíše platí $NL \neq P$)
 - Pokud by libovolný P -úplný problém byl v NL , tak by každý problém z P byl v NL , a platilo by $NL = P$.
 - Pokud by naopak alespoň jeden problém z P nebyl v NL , pak žádný z P -úplných problémů nebude v NL .

P-úplné problémy hrají také důležitou roli jako problémy, které:

- Jsou řešitelné v polynomiálním čase.
- Jsou však **špatně paralelizovatelné** v tom smyslu, že se zdá, že pro ně neexistují **efektivní paralelní** algoritmy, tj. algoritmy, které:
 - používají polynomiální počet procesorů
 - pracují v polylogaritmickém čase
(tj. v čase $\mathcal{O}(\log^k n)$ pro nějakou konstantu k)

Poznámka: Třída problémů, které se dají řešit takovými efektivními paralelními algoritmy se označuje **NC**.

Třídou **NC** (a paralelními algoritmy obecně) se budeme zabývat později.

Booleovský obvod je orientovaný acyklický graf tvořený vrcholy dvou typů — **vstupy** (**inputs**) a **hradly** (**gates**), a hranami, které představují **vodiče** (**wires**):

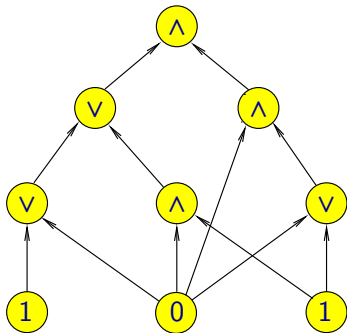
- **Vstupy** — nevedou do nich žádné hrany, jsou označeny jmény booleovských proměnných (každý vstup jiným jménem)
— hodnota daného vstupu je dána přiřazením booleovské hodnoty příslušné proměnné
- **Hradla** — jsou tří různých typů:
 - **NOT** — negace; vede do něj vždy právě jedna hrana
 - **AND** — konjunkce; vedou do něj vždy alespoň dvě hrany
 - **OR** — disjunkce; vedou do něj vždy alespoň dvě hrany
- Jeden z vrcholů je označen jako výstupní.

Circuit Value Problem (CVP)

Circuit Value Problem (CVP)

Vstup: Popis booleovského obvodu G a pravdivostní ohodnocení ν reprezentující hodnoty přiřazené jeho vstupům.

Otázka: Je při daném přiřazení hodnot vstupů ν na výstupu obvodu G hodnota 1?

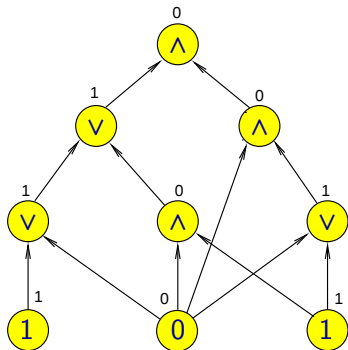


Circuit Value Problem (CVP)

Circuit Value Problem (CVP)

Vstup: Popis booleovského obvodu G a pravdivostní ohodnocení ν reprezentující hodnoty přiřazené jeho vstupům.

Otázka: Je při daném přiřazení hodnot vstupů ν na výstupu obvodu G hodnota 1?



Circuit Value Problem (CVP)

Věta

CVP je P -úplný problém.

Myšlenka důkazu:

To, že problém CVP patří do třídy P je očividné — přímočarý algoritmus vyhodnocující hodnoty jednotlivých hradel má očividně polynomiální časovou složitost.

Je třeba ukázat, že každý problém z P je logspace převeditelný na CVP.

Předpokládejme, že máme dán nějaký konkrétní problém A ze třídy P .

Pro problém A existuje polynomiální algoritmus, který ho řeší.

Tento algoritmus může být realizován nějakým strojem (např. Turingovým strojem) M s polynomiální časovou složitostí omezenou nějakým konkrétním polynomem $p(n)$.

Circuit Value Problem (CVP)

Délka výpočtu nad vstupem velikosti n je maximálně $p(n)$.

Jednotlivé konfigurace stroje $\mathcal{M}$ je možné kódovat jako sekvence $\mathcal{O}(p(n))$ bitů.

Ke vstupu x se vyrobí obvod:

- Bude se skládat z $p(n) + 1$ „úrovní“, které budou odpovídat konfiguracím $\alpha_0, \alpha_1, \dots, \alpha_{p(n)}$, kterými stroj $\mathcal{M}$ projde při výpočtu nad vstupem x .
- Vstupy budou reprezentovat počáteční konfiguraci α_0 .
- Mezi úrovně i a $i + 1$ (kde $0 \leq i < p(n)$) se přidá obvod, který z binární reprezentace konfigurace α_i spočítá binární reprezentaci konfigurace α_{i+1} .
- Za poslední úroveň (úroveň $p(n)$) se přidá obvod, který bude generovat výstup 1 právě tehdy, když hodnoty na této úrovni reprezentují přijímající konfiguraci.

Definice

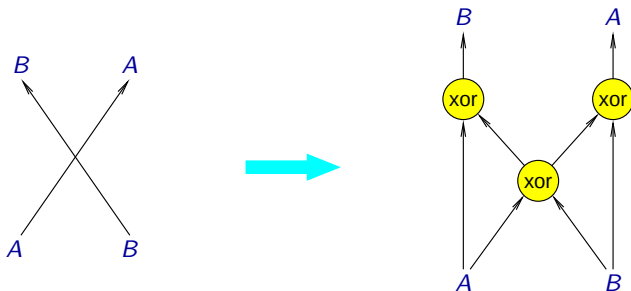
Orientovaný acyklický graf $G = (V, E)$ je **topologicky uspořádaný**, jestliže jsou jeho vrcholy očíslovány čísly $\{1, 2, \dots, n\}$ takovým způsobem, že pro každou hranu $(i, j) \in E$ platí $i < j$ (tj. hrany vedou pouze z vrcholů s nižším indexem do vrcholů s vyšším indexem).

- Není těžké si rozmyslet, že výše popsanou konstrukci v důkazu **P**-úplnosti problému CVP je možné provést tak, aby výsledný graf vytvořeného obvodu byl topologicky uspořádaný.
- Problém CVP tedy zůstává **P**-úplný i ve speciálním případě, kdy požadujeme, aby byl graf obvodu topologicky uspořádaný.

Circuit Value Problem (CVP) — planární graf

Pomocí logspace redukce z CVP se dá ukázat **P**-úplnost CVP také ve speciálním případě, kdy požadujeme, aby graf obvodu byl **planární**.

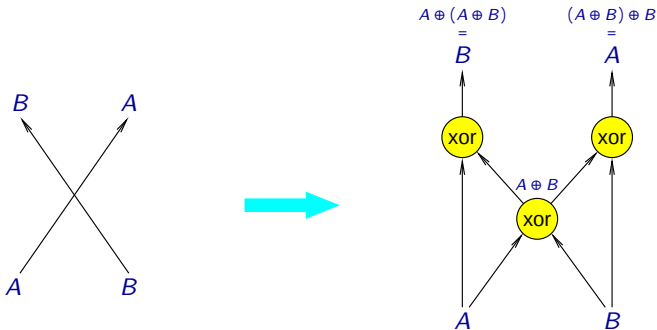
Myšlenka důkazu: Křížení vodičů je možné nahradit pomocí tří hradel typu **XOR**:



Circuit Value Problem (CVP) — planární graf

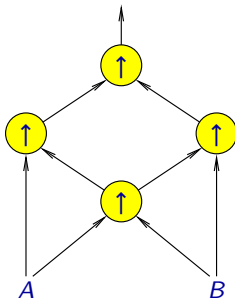
Pomocí logspace redukce z CVP se dá ukázat **P**-úplnost CVP také ve speciálním případě, kdy požadujeme, aby graf obvodu byl **planární**.

Myšlenka důkazu: Křížení vodičů je možné nahradit pomocí tří hradel typu **XOR**:



Circuit Value Problem (CVP) — planární graf

Hradlo typu **XOR** je možné zrealizovat pomocí čtyř hradel typu **NAND** tak, aby odpovídající graf byl planární:



$$\begin{aligned}(A \uparrow (A \uparrow B)) \uparrow ((A \uparrow B) \uparrow B) &\Leftrightarrow \neg((A \uparrow (A \uparrow B)) \wedge ((A \uparrow B) \uparrow B)) \\&\Leftrightarrow \neg(A \uparrow (A \uparrow B)) \vee \neg((A \uparrow B) \uparrow B) \Leftrightarrow (A \wedge (A \uparrow B)) \vee ((A \uparrow B) \wedge B) \\&\Leftrightarrow (A \wedge \neg(A \wedge B)) \vee (\neg(A \wedge B) \wedge B) \Leftrightarrow (A \wedge (\neg A \vee \neg B)) \vee ((\neg A \vee \neg B) \wedge B) \\&\Leftrightarrow (A \wedge \neg B) \vee (\neg A \wedge B) \Leftrightarrow A \text{ xor } B\end{aligned}$$

Monotone Circuit Value Problem (MCVP)

Booleovský obvod je **monotónní**, jestliže neobsahuje hradla typu NOT.

Monotone Circuit Value Problem (MCVP)

Vstup: Popis monotónního booleovského obvodu G , kde navíc do každého hradla typu AND a OR vstupují právě dva vodiče, a pravdivostní ohodnocení ν .

Otázka: Je při pravdivostním ohodnocení ν na výstupu obvodu G hodnota 1?

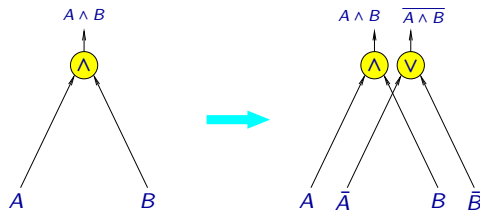
Logspace redukci z problému CVP je možné ukázat následující:

Věta

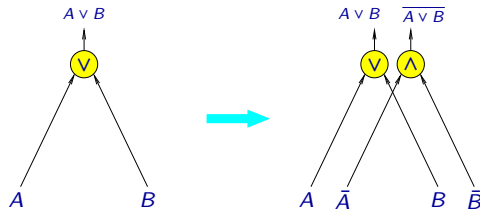
MCVP je P-úplný problém.

Monotone Circuit Value Problem (MCVP)

- Náhrada hradla typu **AND**:



- Náhrada hradla typu **OR**:

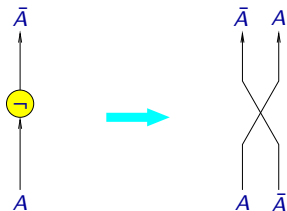


Monotone Circuit Value Problem (MCVP)

- Náhrada vstupů:



- Náhrada hradla typu NOT:



Příklady P-úplných problémů

Pomocí logspace redukcí z MCVP se dá ukázat P-obtížnost celé řady dalších problémů.

Uvedeme si příklady některých P-úplných problémů.

Kombinatorická hra

Vstup: Kombinatorická hra dvou hráčů, jejíž graf je explicitně dán, tj. jsou explicitně vyjmenovány jednotlivé pozice a možné tahy.

Otázka: Má *Hráč I* v dané hře vyhrávající strategii?

Příklady P-úplných problémů

Generování slova bezkontextovou gramatikou

Vstup: Bezkontextová gramatika $\mathcal{G}$ a slovo $w \in \Sigma^*$.

Otázka: Patří slovo w do jazyka generovaného gramatikou $\mathcal{G}$ (tj. platí $w \in \mathcal{L}(\mathcal{G})$)?

Prázdnost jazyka generovaného bezkontextovou gramatikou

Vstup: Bezkontextová gramatika $\mathcal{G}$.

Otázka: Platí $\mathcal{L}(\mathcal{G}) = \emptyset$?

Nekonečnost jazyka generovaného bezkontextovou gramatikou

Vstup: Bezkontextová gramatika $\mathcal{G}$.

Otázka: Je $\mathcal{L}(\mathcal{G})$ nekonečný?

Příklady P-úplných problémů

Generování prvku binární operací

Vstup: Konečná množina X , binární operace $\circ$ na množině X (zadaná tabulkou specifikující hodnotu $x \circ y$ pro každou dvojici $x, y \in X$), podmnožina $S \subseteq X$ a prvek $t \in X$.

Otázka: Je možné prvek t vygenerovat z prvků množiny S ?

Prvek t je možné vygenerovat z prvků množiny S , jestliže existuje nějaký výraz skládající se z konstant reprezentujících prvky z množiny S , na které je libovolným způsobem aplikována operace $\circ$, a hodnota tohoto výrazu je t .

Jiným způsobem se to dá říct také tak, že prvek t patří do nejmenší množiny Y (kde $Y \subseteq X$), která splňuje dvě následující podmínky:

- $S \subseteq Y$
- pro každé dva prvky $x, y \in Y$ platí, že $x \circ y \in Y$.

Příklady P-úplných problémů

Maximální tok v síti

Vstup: Síť G se stanovenými kapacitami hran, se zdrojem s a stokem t , a číslo k .

Otázka: Má k -tý bit čísla reprezentujícího maximální tok v síti G ze zdroje s do stoku t hodnotu 1?

Prohledávání do hloubky (depth-first search)

Vstup: Orientovaný graf $G = (V, E)$, kde je u každého vrcholu stanoveno explicitní pořadí hran, které z něj vedou, počáteční vrchol $s \in V$ a dvojice vrcholů $u, v \in V$.

Otázka: Bude při procházení grafu G do hloubky, které začne ve vrcholu s , a které bude procházet hrany vedoucí z každého vrcholu v uvedeném pořadí, vrchol u navštíven dříve než vrchol v ?