1 Cvičení

Příklad 1.1

Sestavte tabulku udávající (přibližné) hodnoty funkcí n, $n \log n$, n^2, n^3, n^4, 2^n, $n!$ pro argumenty 20, 40, 60, 80, 100, 200, 500, 1000.
Argument chápejme jako délku vstupu jistého algoritmu (programu) a hodnotu funkce jako dobu běhu algoritmu (programu) nad takovým vstupem v mikrosekundách (tj. jako počet elementárních instrukcí provedených v tomto běhu, za předpokladu, že provedení jedné elementární instrukce trvá jednu mikrosekundu).

Příklad 1.2

Představme si, že máme program, jehož doba běhu (počet provedených elementárních instrukcí) na našem počítači je pro vstup délky n rovna $f(n)$. Řekněme, že během jisté vymezené doby (např. 10 minut) jsme schopni programem zpracovat vstup délky 100. Zjistěte, jak velký vstup budeme schopni programem v oné vymezené době zpracovat, nahradíme-li náš počítač novým, 1000× rychlejším počítačem. Přitom jako $f(n)$ postupně uvažujte funkce n, $n \log n$, n^2, n^3, n^4, 2^n, $n!$

Příklad 1.3

Vyjádřete následující funkci $T2$ (argumentu n) co nejjednodušeji (jako polynom)

$$T2(n) = \Big(\sum_{j=1}^{n-1} \Big(10 + \Big(\sum_{i=1}^{n-j} 34 \Big) + 8 \Big) \Big) + 6$$

(Je toho využito v analýze složitosti Bubblesortu v Přednášce 1.)

Příklad 1.4

Harmonické číslo

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i}$$

se občas objeví v analýze složitosti algoritmů. Dokažte pomocí integrálu, že

$$\ln n < H_n < \ln n + 1$$

(Bude toho využito při analýze složitosti algoritmu *quicksort* v průměrném případě.)

Příklad 1.5

Zjistěte, co dělají dva níže uvedené fragmenty programů pro stroje RAM. Význam většiny instrukcí možná při své programátorské zkušenosti uhádnete; kompletní definice modelu RAM je uvedena v pracovním textu. Jen připomeňme, že buňka s adresou 1 je indexregistr a hodnota operandu $*i$ je číslo uložené v buňce s adresou $i + j$, kde j je aktuální obsah indexregistru. (Oproti základní definici zde také užíváme symbolického adresování.)

```
      READ
      STORE   N
      LOAD    =2
cykl: STORE   temp
      LOAD    N
      JGTZ    body
      LOAD    temp
      WRITE
      HALT
body: SUB     =1
      STORE   N
      LOAD    temp
      MUL     temp
      JUMP    cykl
```

```
      LOAD    N
      STORE   1
zmena LOAD    *A
      STORE   X
cykl  LOAD    1
      SUB     =1
      JZERO   konec
      STORE   1
      LOAD    *A
      SUB     X
      JGTZ    zmena
      JUMP    cykl
konec HALT
```

2 Cvičení

Příklad 2.1

Odvoďte, kolik vrcholů má (úplný) binární strom hloubky n a jakou hloubku má binární strom o n vrcholech.

Příklad 2.2

Nechť $a, b > 1$. Ukažte:

- $\exists c : \forall x : \log_a x = c \cdot \log_b x$
- $a^{\log_b n} = n^{\log_b a}$

Příklad 2.3

Ukažte, že (pro $b \neq 0$)

$$\lim_{n\to\infty} \frac{an^2}{bn^3 - cn^2} = 0$$

(Můžete použít l'Hospitalovo pravidlo.)

Příklad 2.4

Ukažte, že

$$\lim_{n\to\infty} \frac{\log n}{n} = 0$$

Příklad 2.5

Nechť funkce f, g jsou definovány vztahy

$$f(n) = 158n^2 \log n + 2541n^2 + 145n$$

$$g(n) = max\,\{\,0\,,\,0.0005n^3 - 1578n^2\,\}$$

Uveďte všechny platné vztahy typu $f \in X(g)$, $g \in X(f)$, kde za X lze dosadit libovolný ze symbolů O, o, Ω, ω, Θ.

Příklad 2.6

Ukažte, že pro $a > 1$, $b \in \mathbb{R}$

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0$$

(Návod: b stačí uvažovat celé kladné (proč ?); pak lze (opakovaně) použít l'Hospitalovo pravidlo.)

Příklad 2.7

Ukažte, že pro $a > 0$

$$\lim_{n\to\infty} \frac{(\log n)^b}{n^a} = 0$$

(Návod: vyjděte z předchozího příkladu a použijte substituci $n \to \log n$, $a \to 2^a$.)

Příklad 2.8

Zformulujte důsledky předchozích příkladů pro srovnávání asymptotické složitosti algoritmů (značení O, o, ...).

3 Cvičení

Referát:
Naprogramujte hlavní jádro algoritmu Heapsort pro RAM a analyzujte jeho časovou složitost.

Příklad 3.1

Rozmyslete detailně algoritmus (se složitostí $O(n)$) pro hledání max. vzdálenosti v polygonu (zadán v pracovním textu a zmíněn na přednášce), včetně důkazu jeho správnosti.

4 Cvičení

Referát:
Analýza složitosti algoritmu Quicksort v průměrném případě. (Optimální) dolní odhad pro problém třídění.

Příklad 4.1

Na přednášce jsme si ukazovali tvrzení (obsažené v pracovním textu):
Tvrzení. Nechť $a \geq 1$, $b > 1$ jsou konstanty, f je funkce (typu $\mathbb{N} \to \mathbb{N}$) a pro funkci $T : \mathbb{N} \to \mathbb{N}$ platí rekurentní vztah

$$T(n) = aT(n/b) + f(n).$$

Pak platí:

1. Je-li $f(n) = O(n^c)$ a $c < \log_b a$, pak $T(n) = \Theta(n^{\log_b a})$.
2. Je-li $f(n) = \Theta(n^{\log_b a})$, pak $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
3. Je-li $f(n) = \Theta(n^c)$ a $c > \log_b a$, pak $T(n) = \Theta(n^c)$.

Při nastínění důkazu (zkoumáním 'výpočetního stromu') jsme, v mírně zjednodušeném případě $f(n) = n^c$ a $T(1) = 1$, odvodili vztah

$$T(n) \;=\; n^c \left(1 + \left(\frac{a}{b^c}\right)^1 + \left(\frac{a}{b^c}\right)^2 + \ldots + \left(\frac{a}{b^c}\right)^{\log_b n}\right)$$

Připomeňte si součet (začátku) geometrické řady a odvoďte tvrzení. (Návod je v pracovním textu.)

5 Cvičení

Referát:
Strassenuv algoritmus pro nasobeni matic.

Příklad 5.1

Pro problém (z pracovního textu)

> *Název problému*: *Výběr aktivit*
>
> *Vstup*: množina konečně mnoha aktivit $\{1, 2, \ldots, n\}$ s pevně určenými časovými intervaly (s_1, f_1), (s_2, f_2), ..., (s_n, f_n), kde $(\forall i, 1 \leq i \leq n) : s_i < f_i$
> *Výstup*: množina obsahující největší možný počet vzájemně kompatibilních aktivit (tj. aktivit s vzájemně se nepřekrývajícími intervaly)

lze použít hltavý (greedy) algoritmus

> Dokud to jde, opakuj následující krok:
> vyber aktivitu, která je kompatibilní s dosud vybranými (na začátku nejsou vybrány žádné) a skončí co nejdříve (když je takových víc, tak kteroukoli z nich).

Dokažte indukcí podle počtu aktivit n, že uvedený přístup skutečně vede k optimálnímu řešení.

Příklad 5.2

Projděte podrobně důkaz použitelnosti hltavého přístupu při konstrukci minimální kostry grafu (který je stručně popsán v pracovním textu).

6 Cvičení

Referát:
Algoritmus (Cocke-Younger-Kasami) pro rozpoznávání bezkontextových jazyků (aplikace metody dynamického programování).

Na přednášce jsme ilustrovali metodu dynamického programování na příkladu (optimálního) uzávorkování řetězce matic. V referátu na cvičení pak byla tato metoda 'vyplňování tabulek' ilustrována na problému rozpoznávání bezkontextových jazyků. Procvičte si ji ještě na následujícím příkladu.

Příklad 6.1

Problém nejdelší společné podposloupnosti
Slovem *posloupnost* zde míníme konečnou posloupnost prvků nějaké množiny (abecedy), označené třeba Σ; je to tedy slovo (neboli řetězec) v abecedě Σ, tedy prvek Σ^*. Abeceda může být např. množina standardních velkých písmen a příkladem posloupnosti (slova) je pak třeba (A, B, C, B, D, A, B). (Nemůže-li dojít k nejednoznačnostem, lze čárky vynechávat a psát $ABCBDAB$.)
Řekneme, že posloupnost u je *podposloupností* posloupnosti v, lze-li u dostat vynecháním (vymazáním) některých členů posloupnosti v. Např. ACA je podposloupností $ABCBDAB$, ale např. DCA podposloupností $ABCBDAB$ není.
Poznámka. Můžete zkusit podat formální definici pojmu *podposloupnost posloupnosti*.
Všimněte si, že podposloupnost neznamená totéž co podslovo (v čem je rozdíl ?).
Jak očekáváme, řekneme, že u je *společnou podposloupností* posloupností v, w, je-li u podposloupností jak v, tak w. Problém, který nás zde zajímá, je specifikován následovně (LCS je z 'Longest Common Subsequence')

> *Název*: Problém LCS (*nejdelší společné podposloupnosti*)
>
> *Vstup*: dvě posloupnosti v, w v nějaké abecedě Σ
> *Výstup*: nejdelší společná podposloupnost posloupností v, w

Uvědomte si, že výstup obecně nemusí být vstupem určen jednoznačně (nicméně jeho délka ano); uveďte nějaký jednoduchý příklad, který to ilustruje.
Zjistěte požadovaný výstup, jsou-li vstupem $ABCBDAB$ a $BDCABA$. U takto malého vstupu 'není co řešit'. Když se ale alespoň na chvíli zamyslíte nad tím, jak byste problém (v rozumném čase) řešili (a pak třeba postup řešení naprogramovali) pro posloupnosti, jejichž délky jsou řádově třeba ve stovkách, zřejmě zjistíte, že to vůbec není triviální.
Jak v daném kontextu očekáváte, nasadíme metodu dynamického programování, tedy budeme vyplňovat tabulku – nejdříve řešeními těch nejmenších podinstancí problému, pak těch větších atd. atd. Samozřejmě to chce nejprve důkladnější porozumění problému, než bychom (doufejme) přišli na to, jakou tabulku a jak vyplňovat.

Bude se nám hodit značení $\text{PREFIX}_i(u)$ ($i \geq 0$); bude označovat posloupnost, která je prefixem (počátkem) posloupnosti u a má délku i (klademe $\text{PREFIX}_i(u) = u$, jestliže $i > |u|$; $|u|$ označuje délku posloupnosti u).
Klíčové pozorování je jednoduché:
Nechť $v = x_1x_2 \dots x_m$, $w = y_1y_2 \dots y_n$ a $u = z_1z_2 \dots z_k$ je LCS posloupností v, w. Pak

- jestliže $x_m = y_n$, pak $z_k = x_m = y_n$ a $\text{PREFIX}_{k-1}(u)$ je LCS posloupností $\text{PREFIX}_{m-1}(v)$ a $\text{PREFIX}_{n-1}(w)$
- jestliže $x_m \neq y_n$, pak $z_k \neq x_m$ implikuje, že u je LCS posloupností $\text{PREFIX}_{m-1}(v)$ a w
- jestliže $x_m \neq y_n$, pak $z_k \neq y_n$ implikuje, že u je LCS posloupností v a $\text{PREFIX}_{n-1}(w)$

To by nás mělo přivést k následujícímu:
Budeme vyplňovat tabulku s řádky $i = 0, 1, 2, \dots, m$, kde m je délka první zadané posloupnosti v, a sloupci $j = 0, 1, 2, \dots, n$, kde n je délka druhé zadané posloupnosti w. Přitom do políčka (i, j) přijde délka LCS posloupností $\text{PREFIX}_i(v)$ a $\text{PREFIX}_j(w)$; pro $i, j > 0$ bude políčko (i, j) zároveň obsahovat odkaz na $(i-1, j-1)$, $(i-1, j)$, nebo $(i, j-1)$ (podle toho, který z případů v našem výše uvedeném pozorování nastává).
Proveďte vyplnění tabulky pro výše uvedené posloupnosti $ABCBDAB$ a $BDCABA$. Postup zapište aspoň stručným pseudokódem, promyslete si správnost algoritmu a analyzujte jeho časovou složitost.

7 Cvičení

Referát:
Detailní výklad vzájemné simulace Turingova stroje a stroje RAM.

Příklad 7.1

Sestrojte Turingův stroj realizující zdvojení slov v abecedě $\{a, b\}$. (Je-li na začátku zapsáno na pásce slovo u, po skončení výpočtu bude zapsáno uu.)
(Příklad byl začat na přednášce.)

Příklad 7.2

Mějme standardní Turingův stroj (předpokládající oboustranně nekonečnou pásku) $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Sestrojme k němu Turingův stroj $M' = (Q', \Sigma, \Gamma', \delta', q'_0, F')$, který předpokládá jen jednostranně (tj. pravostranně) nekonečnou pásku—tedy z nejlevější buňky (na níž stojí hlava na počátku) nemůže přejít doleva—a přitom simuluje stroj M.
Naznačíme možný způsob konstrukce:
$Q' = \{q'_0, q_1\} \cup \{q_x \mid x \in \Sigma\} \cup \{q_U \mid q \in Q\} \cup \{q_D \mid q \in Q\}$
$\Gamma' = \Sigma \cup (\Gamma \times \Gamma) \cup \{\not\in, \Box\}$
$F' = \{q_U \mid q \in F\} \cup \{q_D \mid q \in F\}$
$\delta'(q'_0, x) = (q_x, \not\in, +1)$ … pro $x \in \Sigma$
$\delta'(q_x, y) = (q_y, (x, \Box), +1)$ … pro $x, y \in \Sigma$
$\delta'(q_x, \Box) = (q_1, (x, \Box), -1)$ … pro $x \in \Sigma$
$\delta'(q_1, z) = (q_1, z, -1)$ … pro $z \neq \not\in$
$\delta'(q_1, \not\in) = ((q_0)_U, \not\in, +1)$
Obrázkem si znázorněte pásku a (na malém příkladu) počáteční fázi práce stroje M' (pozn.: asi vás napadne pojem 'dvoustopá páska'); doplňte pak instrukce stroje M' (tedy dodefinujte zobrazení δ') tak, aby skutečně simuloval M. (Ještě kousek nápovědy: U v indexu u stavu znamená 'up', D znamená 'down').

Příklad 7.3

Definujte nějak rozumně Turingův stroj s dvěma páskami (každé pásce přísluší jedna hlava); vstupní slovo je vždy uloženo na první pásce. Ukažte pak, že problém rozpoznání palindromů (tj. slov, jež jsou shodné se svým zrcadlovým obrazem) je na takovém stroji řešitelný v čase $O(n)$. V jakém čase lze tento problém řešit na standardním stroji (jedna páska, jedna hlava) ?

Příklad 7.4

Navrhněte simulaci dvoupáskového stroje standardním jednopáskovým. Zobecněte pak na simulaci k-páskového stroje standardním jednopáskovým. Jaká je časová ztráta při vaší simulaci ? Tj., když k-páskový stroj M má složitost $T_M(n)$, jak dokážete odhadnout složitost $T_{M'}$ vašeho jednopáskového stroje M' simulujícího M ?

8 Cvičení

Referát:
Polynomiální převeditelnost problému nezávislé množiny na problém hamiltonovského cyklu.

Poznámka. Definice problémů uvedených v příkladech jsou obsaženy v pracovním textu.

Příklad 8.1

Připomeňte si myšlenku polynomiálního převodu problému hamiltonovského cyklu na problém hamiltonovské kružnice (HC$\triangleleft$HK; bylo naznačeno na přednášce) a ukažte, že uvedený převod splňuje žádané podmínky (ve vstupním orientovaném grafu existuje hamiltonovský cyklus právě tehdy, když ve výstupním neorientovaném grafu existuje hamiltonovská kružnice).

Příklad 8.2

Ukažte, že problém hamiltonovské kružnice je polynomiálně převeditelný na problém obchodního cestujícího (HK $\triangleleft$ TSP).

Příklad 8.3

Ukažte, že problém splnitelnosti booleovských formulí je polynomiálně převeditelný na variantu téhož problému, kde se předpokládají právě 3 literály v každé klauzuli (SAT $\triangleleft$ 3-SAT).

Příklad 8.4

Ukažte, že problém 3-SAT je polynomiálně převeditelný na problém ILP (celočíselného lineárního programování) (3-SAT $\triangleleft$ ILP).

Příklad 8.5

Zkuste prokázat, že problém 3-SAT je polynomiálně převeditelný na problém obarvení grafu 3 barvami (3-SAT $\triangleleft$ 3-CG).

9 Cvičení

Referát:
Detailní popis konstrukce v důkazu Cookovy věty.

V pracovním textu se tvrdí, že problém

> *Název*: QBF (*problém pravdivosti kvantifikovaných booleovských formulí*)
>
> *Vstup*: formule $(\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4)\ldots(\exists x_{2n-1})(\forall x_{2n})\mathcal{F}(x_1, x_2, \ldots, x_{2n})$, kde $\mathcal{F}(x_1, x_2, \ldots, x_{2n})$ je booleovská formule v konjunktivní normální formě.
> *Otázka*: je daná formule pravdivá ?

je PSPACE-úplný.
Poznámka. Problém je někdy označován i zkratkou Q-SAT pro zdůraznění vztahu k problému SAT. Není ovšem přirozené mluvit o splnitelnosti u plně kvantifikovaných formulí (proč ?).
Víme, že SAT je používán jako 'výchozí' NP-úplný problém (Cookova věta); podobně QBF slouží obvykle jako 'výchozí' PSPACE-úplný problém.

Příklad 9.1

Načrtněte algoritmus, který řeší problém QBF a má prostorovou složitost omezenou polynomem.
Návod. Řekneme, že formule $\mathcal{F}(x_1, x_2, \ldots, x_{2n})$ je OK pro posloupnost booleovských hodnot $b_1, b_2, \ldots, b_i$, kde $0 \leq i \leq 2n$, jestliže

> buď $i = 2n$ a $\mathcal{F}(b_1, b_2, \ldots, b_{2n}) = true$,
>
> nebo $i < 2n$, i je liché a $\mathcal{F}$ je OK jak pro $b_1, b_2, \ldots, b_i, true$, tak pro $b_1, b_2, \ldots, b_i, false$,
>
> nebo $i < 2n$, i je sudé a $\mathcal{F}$ je OK pro aspoň jednu z posloupností $b_1, b_2, \ldots, b_i, true$ a $b_1, b_2, \ldots, b_i, false$.

Ověřte nejprve, že formule $(\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4)\ldots(\exists x_{2n-1})(\forall x_{2n})\mathcal{F}(x_1, x_2, \ldots, x_{2n})$ je pravdivá právě tehdy, když $\mathcal{F}$ je OK pro prázdnou posloupnost.
Uvedený návod tedy ukazuje rekurzivní postup, který přímočaře vede ke kýženému algoritmu.
Polynomem jakého stupně jste schopni omezit prostorovou složitost vašeho algoritmu ?

10 Cvičení

Referát:
Podrobný popis konstrukce v důkazu Savitchovy věty.

Příklad 10.1

Pro prokázání PSPACE-úplnosti problému QBF je rovněž nutno ukázat, že každý problém z PSPACE je polynomiálně převeditelný na QBF. My to podrobně dělat nebudeme, všimneme si jen jednoho důležitého obratu v důkazu.
Připomeňme si z důkazu Cookovy věty, že je vcelku snadné k Turingovu stroji M s polynomiální prostorovou složitostí (v Cookově větě šlo primárně o časovou složitost, ale to v této chvíli nehraje roli) zavést pro daný vstup, resp. danou délku vstupu n, systém C (booleovských) proměnných, jejichž hodnoty popisují konkrétní konfiguraci stroje M; proměnných je v systému C polynomiálně mnoho vzhledem k n a jsou schopny popsat jakoukoli konfiguraci dosažitelnou při výpočtu nad vstupem délky n.
Rovněž je snadné sestrojit formuli $g_0(C, C')$, která je splněna právě tehdy, když systémy C, C' popisují dvě konfigurace, u nichž lze z první nejvýš jedním krokem výpočtu (stroje M) přejít do druhé.
Všimněme si nyní, že obecnější formuli $g_k(C, C')$, která je splněna právě tehdy, když systémy C, C' popisují dvě konfigurace, u nichž lze z první nejvýše po 2^k krocích výpočtu (stroje M) přejít do druhé, můžeme sestrojit rekurzivně takto:
$g_{k+1}(C, C') \iff \exists D : g_k(C, D) \wedge g_k(D, C')$
(D a E, E' dále jsou pomocné sytémy proměnných popisujících konfigurace.)
Zdůvodněte, proč pro naše účely bude (výrazně) užitečnější následující rekurzivní definice:
$g_{k+1}(C, C') \iff \exists D \forall E \forall E' : ((E = C \wedge E' = D) \vee (E = D \wedge E' = C')) \Longrightarrow g_k(E, E')$
(Nápověda. Jde o velikost formulí g_k.)
Po promyšlení důkazu Savitchovy věty zkuste domyslet zbylé kroky důkazu PSPACE-úplnosti problému QBF.

Poznámka.
PSPACE-úplné problémy se často dají přeformulovat tak, že se de facto ptají na existenci vítězné strategie při hře s oponentem.
Např. u instance $(\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4) \ldots (\exists x_{2n-1})(\forall x_{2n}) \mathcal{F}(x_1, x_2, \ldots, x_{2n})$ problému Q-SAT si lze představit, že první hráč (H1) zvolí hodnotu proměnné x_1, H2 (oponent) pak volí hodnotu x_2, H1 volí x_3, H2 x_4, atd.; když po volbě hodnot pro všechny proměnné platí $\mathcal{F}(x_1, x_2, \ldots, x_{2n}) = true$, vyhrál H1, jinak vyhrál H2.
Je zřejmé, že formule $(\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4) \ldots (\exists x_{2n-1})(\forall x_{2n}) \mathcal{F}(x_1, x_2, \ldots, x_{2n})$ je pravdivá právě tehdy, má-li v uvedené hře H1 vítěznou strategii.

Příklad 10.2

Prokažte, že existence vítězné strategie pro H1 v následující hře je PSPACE-úplný problém. 'Hrací deskou' je orientovaný graf G; na jistém výchozím vrcholu leží 'hrací kámen'. H1 a H2 se střídají v tazích; tahem je zde posun kamene po některé (orientované) hraně do vrcholu, na němž ještě kámen neležel. Hráč, který nemůže provést takový tah, prohrává.
Návod. Příslušnost k PSPACE se ukáže podobně jako u Q-SAT. PSPACE-obtížnost lze prokázat převodem Q-SAT na zkoumaný problém. Např. lze postupovat takto: K formuli $(\exists x_1)(\forall x_2)(\exists x_3)(\forall x_4) \ldots (\exists x_{2n-1})(\forall x_{2n}) : C_1 \wedge C_2 \wedge \ldots \wedge C_m$, kde každá C_i $(i = 1, 2, \ldots, m)$ je disjunkce několika literálů, tedy prvků množiny $\{ x_1, \neg x_1, x_2, \neg x_2, \ldots, x_{2n}, \neg x_{2n} \}$ sestrojíme graf s vrcholy $x_1, x_2, \ldots, x_{2n}$, $\neg x_1, \neg x_2, \ldots, \neg x_{2n}$, $u_1, u_2, \ldots, u_{2n}$, $v_1, v_2, \ldots, v_{2n}$, $C_1, C_2, \ldots, C_m$ a hranami (u_i, x_i), $(u_i, \neg x_i)$, (x_i, v_i), $(\neg x_i, v_i)$ pro vš. $i = 1, 2, \ldots, 2n$, dále (v_i, u_{i+1}) pro vš. $i = 1, 2, \ldots, 2n-1$, a dále (v_{2n}, C_j) pro vš. $j = 1, 2, \ldots, m$. Vysvětlete, jaké další hrany je nutno doplnit, aby konstrukce plnila žádaný účel.

Poznámka.
Výše uvedený problém slouží mj. k určitému prokázání obtížnosti deskových her jako GO, šachy apod. (na n-rozměrných 'šachovnicích'). Ale to zde nebudeme rozebírat.
Ne všechny PSPACE-úplné problémy mají onu přirozenou interpretaci ve hře dvou hráčů. Důležitým PSPACE-úplným problémem je např. *ekvivalence nedeterministických automatů* (či regulárních výrazů). Jiný PSPACE-úplný problém je popsán v dalším příkladu (jeho PSPACE-obtížnost zde ale nedokazujeme).

Příklad 10.3

Uvažujme problém, jehož instancí je orientovaný graf s vybraným vrcholem v a dále k 'oblázků'. Můžeme v jakémkoli pořadí provádět následující elementární kroky:

- na vrchol x můžeme položit oblázek, pokud v daný okamžik leží oblázky na všech vrcholech, z nichž vede hrana do x,
- oblázek položený na vrchol můžeme odebrat (a znovu použít později).

Otázkou je, zda existuje posloupnost kroků, při níž položíme oblázek na zadaný vrchol v.
Prokažte, že problém je v PSPACE.
Dále vysvětlete, jak je možné uvedenou hrou modelovat situaci přidělování paměti při výpočtu (stačí daný počet registrů k provedení daného výpočtu ? ...).

11 Cvičení

Referát:
Konstrukce univerzálního Turingova stroje.

Příklad 11.1

Projděte podrobně myšlenky důkazu rozhodnutelnosti Presburgerovy aritmetiky (teorie sčítání) - důkaz je načrtnut v pracovním textu.

12 Cvičení

Referát:
Konstrukce v důkazu převeditelnosti problému zastavení na Postův korespondenční problém (čímž se prokáže nerozhodnutelnost PKP).

Příklad 12.1

Demonstrujte podrobně algoritmickou převeditelnost problému zastavení (HP) na problém univerzálního zastavení (UHP).

Příklad 12.2

Vyslovte přesně Riceovu větu. Zjistěte (a zdůvodněte), pro které z následujících problémů plyne jejich nerozhodnutelnost z Riceovy věty.
Instancí (tj. vstupem problému) je vždy Turingův stroj M, proto uvádíme jen otázky:
a/ Zastaví se M na řetězec 001 ?
b/ Má M více než sto stavů ?
c/ Má v nějakém případě výpočet stroje M více kroků než tisícinásobek délky vstupu ?
d/ Platí, že pro libovolné n se M na vstupech délky nejvýše n vícekrát zastaví než nezastaví ?
e/ Je pravda, že pro lib. vstupní slovo M realizuje jeho zdvojení ?