Kapitola 1

Úvod

1.1 Základní pojmy

1.1.1 Algoritmus. *Algoritmem* rozumíme dobře definovaný proces, tj. posloupnost výpočetních kroků, který přijímá hodnoty (zadání, vstup) a vytváří hodnoty (řešení, výstup).

1.1.2 Problém, úloha. *Úloha*, též *problém*, je obecná specifikace vztahu zadání/řešení. *Instancí* problému, úlohy $\mathcal{U}$ rozumíme konkrétní zadání všech parametrů, které daná úloha (problém) obsahuje. Jinými slovy, instance úlohy je správný příklad zadání.

1.1.3 Řekneme, že algoritmus $\mathcal{A}$ *řeší* úlohu $\mathcal{U}$, jestliže pro každý vstup (každou instanci problému $\mathcal{U}$) vydá správné řešení.

Poznamenejme, že předchozí věta znamená, že každý algoritmus, který řeší nějakou úlohu, se vždy zastaví. To znamená, že algoritmus, který se na nějakém vstupu nezastaví, nemůže řešit žádnou úlohu.

1.1.4 Analýza časové složitosti algoritmu. Existují dva základní způsoby měření časové náročnosti algoritmů.

1. Analýza nejhoršího případu. Jedná se o asymptotický odhad $T(n)$ času potřebného pro vyřešení každé instance velikosti n.
2. Průměrná složitost. Jedná se o asymptotický odhad $T_{aver}(n)$ průměrného času, který je potřeba pro vyřešení instance velikosti n, kde bereme v úvahu s jakou pravděpodobností se jednotlivé instance (typy instancí) vyskytují.

Pro posloupnost operací stejného druhu používáme ještě amortizovanou složitost. Amortizovaná složitost je průměrná složitost nejhoršího případu pro posloupnost n operací/instrukcí stejného druhu.

1.2 Asymptotický růst funkcí

Připomeňme základní pojmy týkající se růstu nezáporných funkcí.

1.2.1 Symbol $\mathcal{O}$. Je dána nezáporná funkce $g(n)$. Řekneme, že nezáporná funkce $f(n)$ *je* $\mathcal{O}(g(n))$, jestliže existuje kladná konstanta c a přirozené číslo n_0 tak, že

$$f(n) \leq c\, g(n) \quad \text{pro všechny } n \geq n_0.$$

□

$\mathcal{O}(g(n))$ můžeme též chápat jako třídu všech nezáporných funkcí $f(n)$:

$$\mathcal{O}(g(n)) = \{f(n) \mid \exists c > 0, n_0 \in \mathbb{N} \text{ tak, že } f(n) \leq c\, g(n) \ \ \forall n \geq n_0\}.$$

1.2.2 Symbol Ω. Je dána nezáporná funkce $g(n)$. Řekneme, že nezáporná funkce $f(n)$ *je* $\Omega(g(n))$, jestliže existuje kladná konstanta c a přirozené číslo n_0 tak, že

$$f(n) \geq c\, g(n) \quad \text{pro všechny } n \geq n_0.$$

□

$\Omega(g(n))$ můžeme též chápat jako třídu všech nezáporných funkcí $f(n)$:

$$\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0 \in \mathbb{N} \text{ tak, že } f(n) \geq c\, g(n) \ \ \forall\, n \geq n_0\}.$$

1.2.3 Poznámka. Fakt, že funkce $f(n)$ je $\Omega(g(n))$ je ekvivalentní faktu, že funkce $g(n)$ je $\mathcal{O}(f(n))$.

1.2.4 Symbol Θ. Je dána nezáporná funkce $g(n)$. Řekneme, že nezáporná funkce $f(n)$ *je* $\Theta(g(n))$, jestliže existují kladné konstanty c_1, c_2 a přirozené číslo n_0 tak, že

$$c_1\, g(n) \leq f(n) \leq c_2\, g(n) \quad \text{pro všechny } n \geq n_0.$$

□

$\Theta(g(n))$ můžeme též chápat jako třídu všech nezáporných funkcí $f(n)$:

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_0 \in \mathbb{N} \text{ tak, že } c_1\, g(n) \leq f(n) \leq c_2\, g(n) \ \ \forall\, n \geq n_0\}.$$

1.2.5 Poznámka. Platí $f(n)$ je $\Theta(g(n))$ právě tehdy, když $f(n)$ je zároveň $\mathcal{O}(g(n))$ a $\Omega(g(n))$.

V dalším zavedeme ještě dvě další třídy funkcí, totiž $o(g(n))$ a $\omega(g(n))$.

1.2.6 Symbol malé o. Je dána nezáporná funkce $g(n)$. Řekneme, že nezáporná funkce $f(n)$ *je* $o(g(n))$, jestliže pro každou kladnou konstantu c existuje přirozené číslo n_0 tak, že

$$0 \leq f(n) < c\, g(n) \quad \text{pro všechny } n \geq n_0.$$

□

$o(g(n))$ můžeme též chápat jako třídu všech nezáporných funkcí $f(n)$:

$$o(g(n)) = \{f(n) \mid \forall\, c > 0\ \exists\, n_0 \in \mathbb{N} \text{ tak, že } 0 \leq f(n) < c\, g(n) \ \ \forall\, n > n_0\}.$$

1.2.7 Poznámka. Fakt, že nezáporná funkce $f(n)$ je $\mathcal{O}(g(n))$, zhruba řečeno znamená, že funkce $f(n)$ neroste asymptoticky více než funkce $g(n)$. Naproti tomu fakt, že nezáporná funkce $f(n)$ je $o(g(n))$, znamená, že funkce $f(n)$ roste asymptoticky méně než funkce $g(n)$.

1.2.8 Symbol malé ω. Je dána nezáporná funkce $g(n)$. Řekneme, že nezáporná funkce $f(n)$ *je* $\omega(g(n))$, jestliže pro každou kladnou konstantu c existuje přirozené číslo n_0 tak, že

$$0 \leq c\,g(n) < f(n) \quad \text{pro všechny } n \geq n_0.$$

□

$\omega(g(n))$ můžeme též chápat jako třídu všech nezáporných funkcí $f(n)$:

$$\omega(g(n)) = \{f(n) \mid \forall\, c > 0\, \exists\, n_0 \in \mathbb{N} \text{ tak, že } 0 \leq c\,g(n) < f(n) \;\; \forall\, n > n_0\}.$$

1.2.9 Poznámka. Fakt, že nezáporná funkce $f(n)$ je $\Omega(g(n))$, zhruba řečeno znamená, že funkce $f(n)$ roste asymptoticky alespoň tak, jako funkce $g(n)$. Naproti tomu fakt, že nezáporná funkce $f(n)$ je $\omega(g(n))$, znamená, že funkce $f(n)$ roste asymptoticky více než funkce $g(n)$.

1.2.10 Značení. Protože symboly $\mathcal{O}, \Omega, \Theta$ představují množiny funkcí, budeme v dalším textu psát $f(n) \in \mathcal{O}(g(n))$. Je ovšem pravda, že v literatuře najdete i zápis $f(n) = \mathcal{O}(g(n))$. Při tomto zápisu je třeba mít na paměti, že znak rovnosti v zápise $f(n) = \mathcal{O}(g(n))$ nemá stejné vlastnosti jako klasická rovnost. Obdobně pro ostatní symboly.

1.2.11 Tvrzení. Jsou dány dvě nezáporné funkce $f(n)$ a $g(n)$. Pak platí

1. $f(n) \in o(g(n))$ právě tehdy, když $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$;
2. $f(n) \in \omega(g(n))$ právě tehdy, když $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.
3. Jestliže $\lim_{n\to\infty} \frac{f(n)}{g(n)} = a$ pro některé $a \in \mathbb{R}$, $a \neq 0$, pak $f(n) \in \Theta(g(n))$

Zdůvodnění: 1) Napíšeme, co znamená fakt $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$:

$$\forall\, \varepsilon > 0\, \exists\, n_0 \in \mathbb{N} \;\text{ tak, že }\; \forall\, n \geq n_0 \;\text{ platí }\; \left|\frac{f(n)}{g(n)}\right| < \varepsilon.$$

Vztah $|\frac{f(n)}{g(n)}| < \varepsilon$ lze přepsat na $f(n) < \varepsilon\, g(n)$. Označíme-li $c := \varepsilon$, dostáváme $f(n)$ je $o(g(n))$.

3) Obdobně fakt $\lim_{n\to\infty} \frac{f(n)}{g(n)} = a$, $a > 0$, znamená, že

$$\forall\, \varepsilon > 0\, \exists\, n_0 \in \mathbb{N} \;\text{ tak, že }\; \forall\, n \geq n_0 \;\text{ platí }\; \left|\frac{f(n)}{g(n)} - a\right| < \varepsilon.$$

Jinak zapsáno $(a-\varepsilon)g(n) < f(n) < (a+\varepsilon)g(n)$. Zvolíme-li $\varepsilon = \frac{a}{2}$, dostáváme

$$\frac{a}{2}\, g(n) < f(n) < \frac{3a}{2}\, g(n);$$

tedy $f(n)$ je $\Theta(g(n))$. □

1.2.12 Tranzitivita. Není těžké se přesvědčit, že platí následující tvrzení.

Tvrzení. Máme dány tři nezáporné funkce $f(n)$, $g(n)$ a $h(n)$.

1. Jestliže $f(n) \in \mathcal{O}(g(n))$ a $g(n) \in \mathcal{O}(h(n))$, pak $f(n) \in \mathcal{O}(h(n))$.
2. Jestliže $f(n) \in \Omega(g(n))$ a $g(n) \in \Omega(h(n))$, pak $f(n) \in \Omega(h(n))$.
3. Jestliže $f(n) \in \Theta(g(n))$ a $g(n) \in \Theta(h(n))$, pak $f(n) \in \Theta(h(n))$.

1.2.13 Reflexivita. Pro všechny nezáporné funkce $f(n)$ platí: $f(n) \in \mathcal{O}(f(n))$, $f(n) \in \Omega(f(n))$ a $f(n) \in \Theta(f(n))$.

1.2.14 Tvrzení. $f(n) \in \Theta(g(n))$ právě tehdy, když $g(n) \in \Theta(f(n))$. □

1.2.15 Příklady.

1. Pro každé $a > 1$ a $b > 1$ platí
$$\log_a(n) \in \Theta(\log_b(n)).$$
2. V celém textu značíme logaritmus o základu 2 symbolem lg, tj. $\lg(n) = \log_2(n)$. Platí
$$\lg n! \in \Theta(n \lg n).$$

Druhá část tvrzení vyplývá např. z následující věty.

1.2.16 Věta (Gauss). Pro každé $n \geq 1$ platí
$$n^{\frac{n}{2}} \leq n! \leq \left(\frac{n+1}{2}\right)^n.$$

□

Zdůvodnění: Využijeme fakt, že pro každá dvě kladná čísla a, b platí $\frac{a+b}{2} \geq \sqrt{ab}$.

Přepíšeme $(n!)^2$ takto

$$(n!)^2 = n\,(n-1)\,\ldots 2 \cdot 1 \cdot 1 \cdot 2\,\ldots\,(n-1)\,n = \prod_{i=1}^{n}(n-i+1)i.$$

Odtud
$$n! = \prod_{i=1}^{n}\sqrt{(n-i+1)i} \leq \prod_{i=1}^{n}\frac{n+1}{2} = \left(\frac{n+1}{2}\right)^n,$$

protože pro každé i platí $\sqrt{(n-i+1)i} \leq \frac{n-i+1+i}{2}$. Tím jsme dostali horní odhad.

Na druhé straně pro každé i platí $n \leq (n-i+1)i$ a proto je $n^n \leq (n!)^2$. Odmocněním dostaneme dolní odhad, totiž $n^{\frac{n}{2}} \leq n!$.

1.2.17 Věta. Máme dánu nezápornou funkci $f(n)$, která je neklesající. Jestliže platí $f(\frac{n}{2}) \in \Theta(f(n))$, pak

$$\sum_{i=1}^{n} f(i) \in \Theta(n\, f(n)).$$

□

Krátké zdůvodnění: Fakt, že $\sum_{i=1}^{n} f(i) \in \mathcal{O}(n\, f(n))$ je zřejmý: f je neklesající.

Dále existuje kladná konstanta c taková, že pro dostatečně velká n platí $c\, f(n) \leq f(\frac{n}{2})$. Proto platí

$$\sum_{i=1}^{n} f(i) \geq f(\frac{n}{2}) + \ldots + f(n) \geq \frac{n}{2}\, c\, f(n).$$

To znamená, že $\sum_{i=1}^{n} f(i) \geq \frac{c}{2}\, n\, f(n)$ a proto $\sum_{i-1}^{n} f(i) \in \Omega(n\, f(n))$.

1.2.18 Poznámka. Vlastnost z předchozí věty má např. funkce $f(n) = n^d$ pro přirozené číslo $d \geq 1$, nemá ji však funkce $f(n) = 2^n$. Pro asymptotický odhad $\sum_{i=1}^{n} 2^i$ se dá využít následující metoda:

Matematickou indukcí dokážeme, že existuje $c > 0$ takové, že

$$\sum_{i=1}^{n} 2^i \leq c \cdot 2^n.$$

Základní krok. Víme, že $\sum_{i=1}^{1} 2^i = 2$ a $2 \leq c \cdot 2$ pro každou konstantu $c \geq 1$.
Indukční krok. Předpokládejme, že platí $\sum_{i=1}^{n} 2^i \leq c \cdot 2^n$. Pak

$$\sum_{i=1}^{n+1} 2^i = \sum_{i=1}^{n} 2^i + 2^{n+1} \leq c\, 2^n + 2^{n+1} = \left(\frac{1}{2} + \frac{1}{c}\right) c \cdot 2^{n+1}.$$

Nyní k dokončení důkazu stačí zajistit, aby $\frac{1}{2} + \frac{1}{c} \leq 1$. A to je ekvivalentní s podmínkou $c \geq 2$.

1.2.19 Ještě jeden způsob získání odhadu. Tvrzení. Pro neklesající nezápornou funkci $f(n)$ platí

$$\int_0^n f(x)\, dx \leq \sum_{i=1}^{n} f(i) \leq \int_1^{n+1} f(x)\, dx.$$

Pro nerostoucí nezápornou funkci $f(n)$ platí

$$\int_1^{n+1} f(x)\, dx \leq \sum_{i=1}^{n} f(i) \leq \int_0^n f(x)\, dx.$$

K tomu, abychom se přesvědčili o výše uvedeném, stačí si nakreslit obrázek a součet $\sum_{i=1}^{n} f(i)$ si představit jako součet ploch obdélníků s jednou stranou 1 a jednou stranou $f(i)$. □

Poznamenejme, že v některých případech $\int_0^n f(x)\, dx$ může být nevlastní integrál. To řešíme tak, že použijeme vztah $\sum_{i=1}^{n} f(i) = f(1) + \sum_{i=2}^{n} f(i)$ a teprve $\sum_{i=2}^{n} f(i)$ omezujeme integrálem.

1.3 Řešení rekursivních vztahů

V této části se zabýváme asymptotickým odhadem funkcí, kde funkční hodnota $T(n)$ závísí na několika hodnotách menších argumentů a další funkci.

Příkladem takového vztahu/funkce je např. počet kroků třídícího algoritmu, kde setřídění seznamu n čísel převedeme na roztřídění každé poloviny seznamu nezávisle a pak „spojení“ takto roztříděných seznamů. Tedy počet kroků se dá vyjádřit následující rovností

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n.$$

1.3.1 Přímá metoda. Metoda spočívá v tom, že odhadneme asymptotický růst a odhad matematickou indukcí dokážeme. Ukažme si to na příkladě:

Příklad. Najděte asymptotické chování funkce $T(n)$, kde

$$T(n) = 2T(\frac{n}{2}) + n, \quad T(1) = 1.$$

Řešení. Náš odhad je: $T(n) \leq cn \lg n$.

Základní krok: Pro $n = 2$ platí $T(2) = 2T(1) + 2 = 4$. Tedy $T(2) \leq c \cdot 2$ pro každou konstantu $c \geq 2$.

Indukční krok. Předpokládejme, že vztah platí pro všechna $m < n$. Then

$$T(n) = 2T\left(\frac{n}{2}\right) + n \leq 2\,c \cdot \frac{n}{2} \lg \frac{n}{2} + n = cn(\lg n - 1) + n.$$

Navíc

$$cn \lg n - cn + n = cn \lg n + n(1 - c) \leq cn \lg n,$$

protože $c \geq 2$. Tedy $T(n) \in \mathcal{O}(n)$.

Obdobně se dá ukázat $T(n) \in \Omega(n \lg n)$.

1.3.2 Řešení rekursivních vztahů pomocí stromů rekurse. Získat dobrý odhad pro přímou metodu není snadná záležitost. Nyní si ukážeme použití rekursivních stromů, které nám pomůže když ne vždy rekursivní vztah vyřešit, tak „kvalifikovaný“ odhad najít. Tuto metodu si ukážeme na dvou příkladech; v prvním případě vztah přímo vyřešíme, ve druhém získáme odhad, který pak přímou metodou — t.j. indukcí dokážeme.

1.3.3 Příklad 1. Řešme rekursivní vztah

$$T(n) = 3T\left(\frac{n}{4}\right) + n^2.$$

Řešení: Vytvoříme si jednotlivé hladiny stromu, který popisuje rekursivní výpočet funkce $T(n)$. V nulté hladině máme pouze $T(n)$ a hodnotu n^2, kterou potřebujeme k výpočtu $T(n)$ (známe-li $T\left(\frac{n}{4}\right)$).

V první hladině se nám výpočet $T(n)$ rozpadl na tři výpočty $T(\frac{n}{4})$. K tomu potřebujeme hodnotu $3 \cdot (\frac{n}{4})^2 = \frac{3}{16}\, n^2$.

Při přechodu z hladiny i do hladiny $i+1$ se každý vrchol rozdělí na tři a každý přispěje do celkové hodnoty jednou šestnáctinou předchozího. Je proto součet v hladině i roven $(\frac{3}{16})^i\, n^2$.

Poslední hladina má vrcholy označené hodnotami $T(1)$ a tím rekurse končí. Počet hladin odpovídá $\lceil \log_4 n \rceil$. V poslední hladině je $3^{\log_4 n} = n^{\log_4 3}$ hodnot $T(1)$. Proto platí

$$T(n) = \sum_{i=0}^{\lceil \log_4 n \rceil} \left(\frac{3}{16}\right)^i n^2 + h(n),$$

kde $h(n) = T(1) \cdot n^{\log_4 3} \in \Theta(n^{\log_4 3})$. Odtud

$$T(n) < n^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + h(n) = n^2 \frac{1}{1-\frac{3}{16}} + h(n) = \frac{16}{13} n^2 + h(n).$$

Navíc, $h(n) \in o(n^2)$, protože $\log_4 3 < 1$. Ukázali jsme, že $T(n) \in \Theta(n^2)$. □

1.3.4 Příklad 2. Řešme rekurentní vztah

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n.$$

Řešení: Vytvoříme si jednotlivé hladiny stromu, který popisuje rekursivní výpočet funkce $T(n)$. V nulté hladině máme pouze $T(n)$ a hodnotu n, kterou potřebujeme k výpočtu $T(n)$ (známe-li $T(\frac{n}{3})$ a $T(\frac{2n}{3})$).

V první hladině se nám výpočet $T(n)$ rozpadl na výpočet $T(\frac{n}{3})$ a $T(\frac{2n}{3})$. K tomu potřebujeme hodnotu $\frac{n}{3} + \frac{2n}{3} = n$.

Ve druhé hladině se vrchol $T(\frac{n}{3})$ rozpadne na $T(\frac{n}{9})$ a $T(\frac{2n}{9})$; vrchol $T(\frac{2n}{3})$ se rozpadne na $T(\frac{2n}{9})$ a $T(\frac{4n}{9})$. Součet v druhé hladině je

$$\frac{n}{9} + \frac{2n}{9} + \frac{2n}{9} + \frac{4n}{9} = n.$$

Nejpozději ve stromě skončí větev odpovídající členům $\frac{2^i n}{3^i}$; skončí právě tehdy, když $\frac{2^i n}{3^i} = 1$. (První člen ve stromu skončí pro $\frac{n}{3^i} = 1$.) Proto ve vyšších hladinách už je součet menší. Poslední neprázdná hladina odpovídá takovému i, že

$$n \to \frac{2n}{3} \to \frac{2^2 n}{3^2} \to \ldots \to \frac{2^i n}{3^i} = 1,$$

t.j. $(\frac{2}{3})^i n = 1$, nebo-li $n = (\frac{3}{2})^i$ a $i = \log_{\frac{3}{2}} n$.

Je ovšem zřejmé, že nevyužijeme celou poslední hladinu — v poslední hladině bude pouze jediný list. Proto, přestože $\log_{\frac{3}{2}} n$ je větší než 1 (a také na základě analogie s $T(n) = 2T(\frac{n}{2}) + n$) zkusíme odhad

$$T(n) \leq d \cdot n \lg n.$$

Odhad je správný a proto platí $T(n) \in \mathcal{O}(n \lg n)$. □

1.3.5 Řešení pomocí Master Theorem

Věta — „Master Theorem“. Jsou dána přirozená čísla $a \geq 1$, $b > 1$ a nezáporná funkce $f(n)$. Předpokládejme, že funkce $T(n)$ je dána na přirozených číslech rekurentním vztahem

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n),$$

kde $\frac{n}{b}$ znamená buď $\lfloor \frac{n}{b} \rfloor$ nebo $\lceil \frac{n}{b} \rceil$.

1. Jestliže $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$ pro nějakou konstantu $\varepsilon > 0$, pak $T(n) \in \Theta(n^{\log_b a})$.
2. Jestliže $f(n) \in \Theta(n^{\log_b a})$, pak $T(n) \in \Theta(n^{\log_b a} \lg n)$.
3. Jestliže $f(n) \in \Omega(n^{\log_b a + \epsilon})$ pro nějakou konstantu $\varepsilon > 0$ a jestliže $a\,f(\frac{n}{b}) \leq c\,f(n)$ pro nějakou konstantu $c < 1$ pro všechna dostatečně velká n, pak $T(n) \in \Theta(f(n))$.

□

1.3.6 Poznámka. Věta 1.3.5 nepokrývá všechny možné případy. Případy, které nejsou pokryty:

1. Funkce $f(n) \in \mathcal{O}(n^{\log_b a})$, ale $f(n) \notin \mathcal{O}(n^{\log_b a - \varepsilon})$ pro žádné $\varepsilon > 0$. Jinými slovy, $f(n)$ není polynomiálně menší než $\mathcal{O}(n^{\log_b a})$.
2. Funkce $f(n) \in \Omega(n^{\log_b a})$, ale $f(n) \notin \Omega(n^{\log_b a + \varepsilon})$ pro žádné $\varepsilon > 0$ (jinými slovy, $f(n)$ není polynomiálně větší než $\Omega(n^{\log_b a})$) nebo neplatí $a\,f(\frac{n}{b}) \leq c\,f(n)$.

1.3.7 Myšlenka zdůvodnění Master Theorem. Master Theorem zdůvodníme pouze pro n, která jsou mocninami b; jinými slovy, pouze v případě, kdy nemusíme pracovat s horní a dolní celou částí. Zdůvodnění je založeno na následujících tvrzení.

Lemma 1. Jsou dána přirozená čísla $a \geq 1$, $b > 1$ a nezáporná funkce $f(n)$. Předpokládejme, že funkce $T(n)$ je dána na přirozených číslech rekurentním vztahem

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n).$$

Pak platí

$$T(n) = \sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j}) + h(n),$$

kde $h(n) \in \Theta(n^{\log_b a})$. □

Lemma 2. Pro $g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j})$ z lemmatu 1 platí:

1. Je-li $f(n) \in \mathcal{O}(n^{\log_b a - \varepsilon})$ pro $\varepsilon > 0$, pak $g(n) \in \mathcal{O}(n^{\log_b a})$.
2. Je-li $f(n) \in \Theta(n^{\log_b a})$, pak $g(n) \in \Theta(n^{\log_b a} \lg n)$.
3. Jestliže existuje $c < 1$ takové, že $a\,f(\frac{n}{b}) \leq c\,f(n)$, pak $g(n) \in \Theta(f(n))$.

□

1.3.8 Jak jsme uvedli výše, nepokrývá Master Theorem všechny případy. Ukážeme ještě jedno tvrzení, které je zobecněním Master Theorem a dovoluje najít asymptotické odhady pro širší třídu funkcí.

Tvrzení. Jsou dána čísla $a \geq 1$, $b > 0$ a nezáporná funkce $f(n)$. Jestliže $f(n) \in \Theta(n^{\log_b a} \lg^k n)$ pro $k \geq 0$, pak pro funkci $T(n)$ danou rovnicí

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n),$$

platí: $T(n) \in \Theta(n^{\log_b a} \lg^{k+1} n)$. □

Důkaz je analogický důkazu Master Theorem, bod 2. Stačí použít zobecnění bodu 2 z Lemmatu 2, viz 1.3.7.

1.3.9 Amortizovaná složitost. Jedná se o výpočet průměrné složitosti nejhoršího případu pro posloupnost n opakování dané instrukce. Jestliže n opakování v nejhorším případě vyžaduje čas $\mathcal{O}(T(n))$, pak jedno provedení vyžaduje čas $\mathcal{O}(T(n))/n$, a to je amortizovaná složitost jedné instrukce.

Jsou tři základní způsoby, jak amortizovanou složitost zjišťovat.

- První je tzv. *agregační* — postupuje se přímo podle předchozího odstavce.
- Druhá metoda je tzv. *účetní*. Každému provedení instrukce přiřadíme jistý kredit. Jestliže provedení instrukce nespotřebuje celý kredit, zbývající část kreditu je možno využít v dalších provedení instrukce, které jsou náročnější a na které by jejich kredit nestačil. Podmínkou ale je, aby žádná instrukce v posloupnosti nespotřebovala víc než je součet jejího kreditu a zatím nevyužitých částí kreditů.
- Třetí metoda je tzv. *potenciálová*. Označme D_i stav po provedení i-té instrukce. Máme tedy posloupnost n stavů (většinou datových struktur) $D_0, \ldots, D_{n-1}$. Každé D_i je přiřazeno nezáporné číslo, tzv. potenciál $\Phi(D_i)$. Označme ještě c_i skutečnou cenu přechodu od D_{i-1} k D_i. Pak amortizovaná cena $\hat{c}_i$ příslušná D_i je definována jako

  $$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}).$$

  Pak platí

  $$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).$$

  Odtud dostáváme podmínky na potenciály, totiž pro každé i musí platit $\Phi(D_i) \geq \Phi(D_0)$.

1.3.10 Na přednášce si ukážeme výpočet amortizované složitosti všemi třemi způsoby na příkladu následujícího pseudokódu

INCREMENT(A)
1. $i = 0$

2. `while` $i < A.length$ a $A[i] = 1$
3. $A[i] := 0$
4. $i := i + 1$
5. `if` $i < A.length$
6. $A[i] := 1$

Kapitola 2

Časová složitost a správnost algoritmů

2.1 Časová složitost algoritmů

Výpočet časového odhadu ukážeme na příkladě Euklidova algoritmu, který pro dvě kladná nenulová přirozená čísla najde jejich největší společný dělitel.

2.1.1 Euklidův algoritmus.

Rekurzivní verze Euklidova algoritmu:

Vstup: Kladná přirozená čísla a, b.

Výstup: $\gcd(a, b)$.

EUKLID(a, b)
1. `if` $b = 0$
2. `return` a
3. `else return` EUKLID$(b, a \pmod b)$

Pro zjištění časového odhadu vycházíme z jeho rekurzivního tvaru. Nejprve dokážeme tři pomocná tvrzení.

2.1.2 Horní odhad časové složitosti Euklidova algoritmu dokazuje následující tvrzení.

Tvrzení. Označme x_k a y_k dvojici čísel $x_k > y_k$ po k-tém rekurzivním volání. Pak platí $y_{k+2} < \frac{y_k}{2}$.

Důkaz. Víme, že $y_{k+2} < y_{k+1} < y_k$. Jestliže $y_{k+1} \leq \frac{y_k}{2}$, pak $y_{k+2} < y_{k+1}$ dokazuje, že $y_{k+2} < \frac{y_k}{2}$.

Předpokládejme, že $y_{k+1} > \frac{y_k}{2}$. Pak

$$y_{k+2} < x_{k+1} - y_{k+1} = y_k - y_{k+1} < \frac{y_k}{2}.$$

2.1.3 Dolní odhad dokážeme pomocí několika lemmat.

Lemma 1: Je-li $a > b \geq 1$ a algoritmus EUKLID(a, b) potřebuje k rekurzivních volání, pak $a \geq F(k+2)$ a $b \geq F(k+1)$, kde $F(i)$ je i-tý člen Fibonacciho posloupnosti.

Připomeňme, že Fibonacciho posloupnost je posloupnost:

$$F(0) = 0,\ F(1) = 1,\ F(n) = F(n-1) + F(n-2) \text{ pro } n \geq 2.$$

Důkaz je možné vést indukcí podle počtu rekurzivních volání:

Základní krok: Pro $k = 1$ je $b \geq 1 = F(2)$ a $a > b \geq 1$, tj. $a \geq 2 = F(3)$.

Indukční krok: Předpokládejme, že tvrzení platí pro počet $k \geq 2$ rekurzivních volání. Předpokládejme, že pro dvojici a, b, $a > b$ je potřeba $k+1$ volání. Procedura EUKLID(a, b) volá proceduru EUKLID$(b, a(\bmod b))$, která potřebuje k volání. Z indukčního předpokladu víme, že $b \geq F(k+2)$ a $z = a(\bmod b) \geq F(k+1)$. Máme $z = a - qb$ pro vhodné q celé a $z < b$. Protože $z < b$, je $q \geq 1$; odtud

$$a = qb + z \geq qF(k+2) + F(k+1) \geq F(k+2) + F(k+1) = F(k+3).$$

Ukázali jsme, že $a \geq F(k+3)$ a $b \geq F(k+2)$.

Lemma 2: EUKLID$(F(k+2), F(k+1))$ potřebuje $k+1$ rekurzivních volání.

Lemma 3: Pro každé $n \geq 0$ platí $F(n+2) \geq \left(\frac{3}{2}\right)^n$.

Důkaz. Použijeme matematickou indukci.

Základní krok. Pro $n = 0$ a $n = 1$ tvrzení platí, protože $F(2) = 1 \geq \left(\frac{3}{2}\right)^0$ a $F(3) = 2 \geq \left(\frac{3}{2}\right)^1$.

Indukční krok. Předpokládejme, že platí $F(n) \geq \left(\frac{3}{2}\right)^{n-2}$ a $F(n+1) \geq \left(\frac{3}{2}\right)^{n-1}$. Pak

$$F(n+2) = F(n+1) + F(n) \geq \left(\frac{3}{2}\right)^{n-1} + \left(\frac{3}{2}\right)^{n-2} = \left(\frac{3}{2}\right)^n \left(\frac{2}{3} + \frac{4}{9}\right) \geq \left(\frac{3}{2}\right)^n.$$

Stačí si uvědomit, že $\left(\frac{2}{3} + \frac{4}{9}\right) = \frac{10}{9}$.

2.1.4 Tvrzení: Algoritmus EUKLID(a, b) vyžaduje $\mathcal{O}(\lg b)$ rekurzivních volání. Tedy jeho složitost vztažená k počtu celočíselných dělení je lineární (neboť velikost vstupu je úměrná $\lg(a + b)$).

2.2 Správnost algoritmů

2.2.1 K ověření správnosti algoritmu je třeba ověřit dvě věci

1. algoritmus se na každém vstupu zastaví,
2. algoritmus po zastavení vydá správný výstup – řešení.

Použití obou kroků si nejprve ukážeme na velmi dobře známých algoritmech.

Na přednášce ukážeme správnost některých následujících algoritmů.

2.2.2 Bublinkové třídění.

Vstup: posloupnost přirozených čísel $a[1], a[2], \ldots, a[n]$.

Výstup: posloupnost setříděná do neklesající posloupnosti.

```
begin
     for k = n step -1 to 2 do
          for j = 1 step 1 to k − 1 do
              if a[j] > a[j + 1] then
                  zaměň a[j] a a[j + 1]
end
```

2.2.3 Fakt, že se algoritmus 2.2.2 zastaví, je zaručen tím, že vnější cyklus se opakuje $(n-1)$-krát.

2.2.4 Tvrzení. Po i-tém proběhnutí vnějšího cyklu, tj. pro $k = n - i$, platí

a) $a[n-i+1], a[n-i+2], \ldots, a[n]$ jsou největší z čísel $a[1], a[2], \ldots, a[n]$

b) $a[n-i+1] \leq a[n-i+2] \leq \ldots \leq a[n]$.

Důkaz tohoto tvrzení se vede indukcí podle n počtu průchodů vnitřním cyklem.

Základní krok: Pro $i = 0$, tj. před proběhnutím vnitřního cyklu, je $n-i+1 = n+1$ a takový člen posloupnosti není. Pro $i = 1$, tj. po jednom proběhnutí vnitřního cyklu, je $a[n-1+1] = a[n]$ a je to největší prvek posloupnosti.

Indukční krok: Jestliže tvrzení platí před k-tým průchodem vnitřního cyklu, pak po jeho průchodu je $a[n-k+1] \geq a[j]$ pro $j \leq n-k$, tedy platí a) a navíc je nejmenší z $a[n-k+1]$, $a[n-k+2]$, …, $a[n]$.

2.2.5 Správnost Euklidova algoritmu 2.1.1 Protože se zbytky při dělení čísla r číslem t stále zmenšují a jsou to přirozená čísla, musí jednou nastat případ, kdy zbytek je nula. Proto se algoritmus vždy zastaví.

Uvědomte si, že nejpozději po prvním průchodu krokem 2 platí $r \geq t$.

2.2.6 Tvrzení. Dvojice čísel r, t a dvojice čísel t, z z Euklidova algoritmu 2.1.1 mají stejné společné dělitele.

2.2.7 Variant. Pro důkaz faktu, že se algoritmus na každém vstupu zastaví, je založen na nalezení tzv. *variantu*. Variant je hodnota udaná přirozeným číslem, která se během práce algoritmu snižuje až nabude nejmenší možnou hodnotu (a tím zaručuje ukončení algoritmu po konečně mnoha krocích). Poznamenejme, že někdy se též hodnota zvětšije k předem známé maximální hodnotě.

V příkladu 2.2.2 se jednalo o číslo k, v příkladu 2.1.1 se jednalo o zbytek z při dělení čísla r číslem t.

2.2.8 Invariant. *Invariant*, též *podmíněná správnost algoritmu*, je tvrzení, které

- platí před vykonáním prvního cyklu algoritmu, nebo po prvním vykonání cyklu,
- platí-li před vykonáním cyklu, platí i po jeho vykonání,
- při ukončení práce algoritmu zaručuje správnost řešení.

Pro algoritmus pro bublinkové třídění je invariantem tvrzení 2.2.4, pro Euikleidův algoritmus tvrzení 2.2.6.

2.2.9 Minimální kostra. Je dán prostý neorientovaný graf $G = (V, E)$ s množinou vrcholů V a množinou hran E. Dále je dáno ohodnoceni a hran, tj zobrazení $a\colon E \to \mathbb{N}$. Úkolem je najít kostru K grafu G takovou, že

$$\sum_{e \in K} a(e) \quad \text{je nejmenší.}$$

Ukážeme správnost jakéhokoli algoritmu založeného na následujícím schematu.

2.2.10 Obecné schema.

Vstup: souvislý neorientovaný graf $G = (V, E)$ a ohodnocení hran a.

Výstup: hrany minimální kostry K.

1. (Inicializace)
   $K := \emptyset$, $\mathcal{S} = \{\{v\} \mid v \in V\}$;
2. (Výběr hrany.)
   Dokud $\mathcal{S}$ není jednoprvková
   vybereme hranu $e \in E \setminus K$ takovou, že
   vede mezi dvěma různými množinami z $\mathcal{S}$, označme je C_1, C_2, a aspoň pro jednu z nich je nejlevnější hrana vedoucí z ní.
3. (Úpravy.)
   $K := K \cup \{e\}$;
   $\mathcal{S} := (\mathcal{S} \setminus \{C_1, C_2\}) \cup \{C_1 \cup C_2\}$.

2.2.11 Ukončení schematu pro minimální kostru (variant). Uvedené schema není algoritmus – není v něm uvedeno, jakým způsobem vybíráme hranu e v kroku 2. Jestliže však tento krok implementujeme kteroukoli metodou, která zajistí, že hranu v konečném čase najdeme, pak schema musí skončit. Ano, zpracováním každého výběru hrany v kroku 2 se zmenší počet množin v systému $\mathcal{S}$ o jednu. Protože $\mathcal{S}$ má na začátku práce schematu n množin, po $n-1$ krocích 3 bude $\mathcal{S}$ jednoprvková a schema skončí.

2.2.12 Tvrzení (invariant). Jestliže množina hran K před vykonáním kroku 2 je částí některé minimální kostry a vybereme-li hranu e podle schematu 2.2.10, pak množina hran $K \cup \{e\}$ je také částí některé minimální kostry.

Důkaz: Předpokládejme, že množina K vytvořená schematem 2.2.10 je částí minimální kostry T_{min}. Vezměme hranu e z kroku 2. Platí buď $e \in T_{min}$ nebo $e \notin T_{min}$.

První případ je jednodušší: jestliže $e \in T_{min}$, pak $K \cup \{e\} \subseteq T_{min}$ a opravdu, nová množina K je částí některé minimální kostry – totiž T_{min}.

Uvažujme tu horší variantu, totiž $e \notin T_{min}$ a předpokládejme, že hrana $e = \{u, v\}$ spojuje dvě komponenty souvislosti K, které označíme C_1 a C_2, tj. $u \in C_1$ a $v \in C_2$. Předpokládejme, že e je nejlevnější hrana vycházející ven z komponenty C_1. Protože minimální kostra T_{min} je souvislý graf, existuje cesta P v T_{min} z vrcholu u do vrcholu v. Označme e_1 hranu P, která vychází z množiny C_1.

Protože e je nejlevnější hrana vycházející z C_1 a e_1 také vychází z C_1, platí $a(e) \leq a(e_1)$.

Přidáme-li ke stromu jednu hranu, uzavřeme právě jednu kružnici; tj. $T_{min} \cup \{e\}$ obsahuje kružnici a to $P \cup \{e\}$. Proto $T = (T_{min} \cup \{e\}) \setminus \{e_1\}$ je také kostrou. Cena kostry T je $a(T_{min}) + a(e) - a(e_1)$. Protože T_{min} je minimální kostra, musí platit

$$a(T_{min}) + a(e) - a(e_1) \geq a(T_{min}), \quad \text{tj.} \quad a(e) \geq a(e_1).$$

Odtud $a(e) = a(e_1)$ a proto $a(T) = a(T_{min})$, proto T je také nějaká minimální kostra a navíc $K \cup \{e\} \subseteq T$.

2.2.13 Pozorování. Jak Kruskalův algoritmus, tak Primův algoritmus jsou zvláštní případy obecného schematu 2.2.10.

2.3 Nejkratší cesty.

Je dán prostý orientovaný graf $G = (V, E)$, kde $V = \{1, 2, \ldots, n\}$, a ohodnocení hran a, tj. zobrazení $a\colon E \to \mathbb{Z}$.

2.3.1 Matice délek A. *Matice délek* je čtvercová matice $\mathbf{A} = (a(i,j))$ řádu n, kde n je počet vrcholů grafu G, a

$$a(i,j) = \begin{cases} 0, & \text{pro } i = j \\ a(e), & \text{pro } e = (i,j) \in E \\ \infty, & \text{pro } (i,j) \notin E \end{cases}$$

2.3.2 Matice vzdáleností U. *Matice vzdáleností* je čtvercová matice $\mathbf{U} = (u(i,j))$ řádu n, kde n je počet vrcholů grafu G, a

$$u(i,j) = \begin{cases} 0, & \text{pro } i = j, \\ \text{délka nejkratší cesty z } i \text{ do } j, & \text{jestliže existuje cesta z } i \text{ do } j \\ \infty, & \text{jestliže neexistuje cesta z } i \text{ do } j \end{cases}$$

2.3.3 Pozorování. Předpokládejme, že vrchol y je orientovaně dostupný z vrcholu x v grafu G. Pak platí:

1. Jestliže graf G obsahuje pouze cykly kladné délky (tj. neobsahuje ani cykly záporné délky ani nulové délky), pak nejkratší sled z vrcholu x do vrcholu y existuje a je současně nejkratší cestou z x do y.
2. Jestliže v grafu G neexistuje cyklus záporné délky, pak nejkratší sled z x do y má stejnou délku jako nejkratší cesta z x do y.
3. Jestliže v grafu G neexistuje cyklus záporné délky, pak pro každý sled C z x do y existuje cesta z x do y, která je kratší nebo stejně dlouhá jako sled C.

2.3.4 Trojúhelníková nerovnost. Jestliže v grafu G neexistuje cyklus záporné délky, pak pro každé tři vrcholy x, y, z platí

$$u(x, y) \leq u(x, z) + u(z, y).$$

Důkaz: Jestliže vrchol z není orientovaně dostupný z vrcholu x nebo vrchol y není orientovaně dostupný z vrcholu z, pak trojúhelníková nerovnost triviálně platí.

V opačném případě označme P_1 nejkratší cestu z vrcholu x do vrcholu z a P_2 nejkratší cestu z vrcholu z do vrcholu y. Spojení obou cest je sled P_1, P_2 s délkou rovnou součtu délek cest P_1 a P_2. Protože graf neobsahuje cykly záporné délky, tento sled obsahuje cestu, která je kratší nebo stejně dlouhá jako délka P, tj. jako $u(x, y) + u(z, y)$. Proto i pro délku nejkratší cesty z x do y platí $u(x, y) \leq u(x, z) + u(z, y)$.

2.3.5 Bellmanův princip optimality. Jestliže v grafu G neexistuje cyklus záporné délky, pak pro každé tři vrcholy x, y, z platí

$$u(x, y) = \min_{z \neq y}(u(x, z) + a(z, y)).$$

Důkaz: Vztah jistě platí pro vrcholy x, y, pro které neexistuje cesta z x do y.

Předpokládejme, že existuje cesta z x do y, tj. $u(x, y) < \infty$. Protože $u(z, y) \leq a(z, y)$ pro každé dva vrcholy z, y, víme z trojúhelníkové nerovnosti, že $u(x, y) \leq u(x, z) + a(z, y)$. Proto

$$u(x, y) \leq \min_{z \neq y}(u(x, z) + a(z, y)).$$

Rovnost nastává pro vrchol z, který je předposlední na nejkratší cestě z vrcholu x do vrcholu y.

2.3.6 Nejkratší cesty z výchozího vrcholu r. Úloha: Najděte délky nejkratších cest z výchozího vrcholu r.

2.3.7 Obecné schema.

Vstup: orientovaný graf $G = (V, E)$ a ohodnocení hran a.

Výstup: hodnoty $U(v)$ rovné $u(r, v)$.

1. (Inicializace.)
   $U(r) := 0$, $U(v) := \infty$ pro $v \neq r$;
2. (Zpracování hran.)
   Existuje-li hrana $e = (v, w)$ taková, že
   $U(w) > U(v) + a(e)$
   položíme $U(w) := U(v) + a(e)$.
3. (Ukončení.)
   Jestliže $U(w) \leq U(v) + a(e)$ pro každou hranu $e = (v, w)$, stop;
   jinak pokračujeme krokem 2.

2.3.8 Tvrzení. Jestliže v grafu G neexistuje cyklus záporné délky a hodnota $U(v) \neq \infty$, pak $U(v)$ je délka některé cesty z vrcholu r do vrcholu v.

Nástin důkazu: Označme $U_t(y)$ hodnotu $U(y)$ v okamžiku t. Platí: jestliže v nějakém okamžiku t_k je $U_{t_k}(x) < \infty$, tak musí existovat sled z r do x

$$r = v_1, e_1, v_2, e_2, \ldots, v_{k-1}, e_{k-1}, v_k = x$$

a časové okamžiky $t_1 < t_2 < \ldots < t_k$ tak, že

$$U_{t_i}(v_i) = \sum_{j=1}^{i} a(e_j).$$

Nyní je třeba dokázat, že se nejedná o sled, ale o cestu. Kdyby se ve sledu opakoval vrchol, tj. kdyby např. $v_i = v_j$ pro $i < j$, pak $U_{t_i}(v_i) > U_{t_j}(v_j)$ a proto se dá dokázat, že $v_i, e_i, v_{i+1}, e_{i+1}, \ldots, v_j$ obsahuje cyklus záporné délky.

2.3.9 Věta. Jestliže graf G neobsahuje cyklus záporné délky a hodnoty $U(v)$ byly získány podle schematu 2.3.7, pak $U(v) = u(r, v)$.

Důkaz: Sporem. Kdyby tvrzení věty neplatilo, po skončení práce schematu by existoval vrchol v takový, že $U(v) > u(r, v)$. To také znamená, že $u(r, v) < \infty$. Vezměme nejkratší cestu P z vrcholu r do vrcholu v. Protože $U(r) = u(r, r)$ a poslední vrchol je v, pro který $U(v) > u(r, v)$, na cestě P existuje hrana $e = (x, y)$ taková, že $U(x) = u(r, x)$ a $U(y) > u(r, y)$. Vezměme první takovou hranu $e = (x, y)$. Pro tyto vrcholy x, y platí:

$$U(y) > u(r, y) = u(r, x) + a(x, y) = U(x) + a(x, y).$$

Tedy, obecné schema nemělo skončit, protože trojúhelníková nerovnost neplatí pro hranu $e = (x, y)$. Tedy není pravda, že se postup zastavil.

2.3.10 Nejprve uvedeme jednoduchý algoritmus, nazveme ho Algoritmus I, který vychází z obecného schematu 2.3.7. Vždy probereme všechny hrany grafu v libovolném, ale pevně daném pořadí. Jestliže při průchodu nedojde ke změně žádné hodnoty $U(x)$, pak už platí trojúhelníková nerovnost pro všechny hrany a můžeme algoritmus ukončit.

Protože cesta v grafu s n vrcholy má nejvýše $n-1$ hran, nemá-li graf záporné cykly, musí následující algoritmus skončit po nejvýše n průchodech krokem 2.

Fakt, že průchodů krokem 2 je maximálně n dává invariant tohoto algoritmu; je to $n - k$ kde k je počet již proběhlých kroků 2.

Dále nám toto pozorování umožňuje poznat graf se zápornými cykly. Jestliže i při n-tém průchodu krokem 2 došlo ke změně některé hodnoty $U(x)$, pak graf obsahuje cyklus záporné délky a výsledky, které jsme algoritmem dostaly, jsou nesprávné.

Poznamenejme, že časové nároky algoritmu I jsou $\mathcal{O}(m.n)$, kde $n = |V|$ a $m = |E|$.

2.3.11 Algoritmus I.

Vstup: orientovaný graf $G = (V, E)$ a ohodnocení hran a.

Výstup: hodnoty $U(v)$ rovné $u(r, v)$.

1. (Inicializace.)
   $U(r) := 0$, $U(v) := \infty$ pro $v \neq r$;
2. (Zpracování hran.)
   Pro každou hranu $e \in E$ provedeme
   jestliže $U(KV(e)) > U(PV(e)) + a(e)$
   položíme $U(KV(e)) := U(PV(e)) + a(e)$.
3. (Ukončení.)
   Jestliže během kroku 2 nedošlo ke změně hodnoty $U(v)$, stop, a vrátíme $U(v)$.
   Jinak pokračuj krokem 2.

2.3.12 Při práci algoritmu I se může stát, že při nevhodné volbě pořadí hran první průchod krokem 2 změní jen málo třeba i jen jednu hodnotu $U(x)$ — to nastane v případě, že z vrcholu r vychází jen jedna hrana a ta bude probírána jako poslední. Uvedeme proto ještě sofistikovanější variantu schematu 2.3.7 — jedná se o algoritmus II.

V tomto algoritmu udržujeme množinu M vrcholů „podezřelých“ z toho, že pro hrany, které z nich vycházejí by nemusela platit trojúhelníková nerovnost. Jinými slovy, jestliže $x \notin M$, pak pro každou hranu e s $PV(e) = x$ již trojúhelníková nerovnost platí.

Na začátku práce je $M = \{r\}$. Množinu M udržujeme tak, že kdykoli snižujeme hodnotu $U(x)$ pro nějaký vrchol x, vrchol x do množiny M zařadíme.

2.3.13 Algoritmus II.

Vstup: orientovaný graf $G = (V, E)$ a ohodnocení hran a.

Výstup: hodnoty $U(v)$ rovné $u(r, v)$.

1. (Inicializace.)
   $U(r) := 0$, $U(v) := \infty$ pro $v \neq r$; $M := \{r\}$
2. (Zpracování hran.)
   Dokud $M \neq \emptyset$, vybereme $x \in M$;
   $M := M \setminus \{x\}$
   pro každou hranu e s $PV(e) = x$ provedeme
   jestliže $U(KV(e)) > U(x) + a(e)$

položíme $U(KV(e)) := U(x) + a(e)$; $M := M \cup \{KV(e)\}$.

3. (Ukončení.)
Vrátíme $U(v)$; stop.

2.3.14 Nejkratší cesty mezi všemi dvojicemi vrcholů. Úkolem je najít celou matici vzdáleností (a ne jen jeden její řádek).

Množinu vrcholů grafu G označíme $V = \{1, 2, \ldots, n\}$. Floydův algoritmus (v literatuře též nazývaný Floyd-Warshallův algoritmus) je založen na konstrukci matic $\mathbf{U}_k = (u_k(i,j))$ řádu n pro $k = 0, 1, \ldots, n$ s následující vlastnosti:

$u_k(i,j)$ je délka nejkratší cesty z i do j, která prochází pouze vrcholy $1, 2, \ldots, k$.

2.3.15 Tvrzení. Platí

1. $\mathbf{U}_0$ je matice délek $\mathbf{A}$.
2. $\mathbf{U}_n$ je matice vzdáleností $\mathbf{U}$.
3. Matici $\mathbf{U}_{k+1}$ získáme z matice $\mathbf{U}_k$ takto:

$$u_{k+1}(i,j) = \min\{u_k(i,j), u_k(i,k+1) + u_k(k+1,j)\}.$$

Důkaz: První dvě vlastnosti jednoduše vyplývají z definice matic $\mathbf{U}_0$ a $\mathbf{U}_n$.

Třetí vlastnost dostaneme, když si uvědomíme, že nejkratší cesta z i do j, která vede pouze přes vrcholy $1, 2, \ldots, k+1$ se buď vrcholu $k+1$ vyhne (a pak je délky $u_k(i,j)$), nebo vrcholem $k+1$ prochází a pak je délky $u_k(i,k+1) + u_k(k+1,j)$.

2.3.16 Floydův algoritmus.

Vstup: matice délek $\mathbf{A}$.

Výstup: matice vzdáleností $\mathbf{M} = \mathbf{U}$.

1. *[Inicializace]*
   $\mathbf{M} := \mathbf{A}$
2. `begin`
   `for` $k = 1, 2, \ldots, n$ `do`
   `for` $i = 1, 2, \ldots, n$ `do`
   `for` $j = 1, 2, \ldots, n$ `do`
   `begin`
   `if` $M(i,j) > M(i,k) + M(k,j)$ `then`
   $M(i,j) := M(i,k) + M(k,j)$
   `end`
   `end`

2.3.17 Ukončení Floydova algoritmu je zaručeno tím, že vnější cyklus se provádí n-krát, tj. variant je k, které se roste od 1 do n.

Invariantem je 2.3.14 a vlastnost 3 z 2.3.15.

2.4 Huffmanův kód pro kompresi dat.

Jsou dána data obsahující znaky z abecedy C a pro každý znak $c \in C$ je dána četnost $c.freq$ výskytu c v datech. Kódovat znaky můžeme buď slovy stejné délky; délka jednotlivého kódového slova je dána počtem znaků — je to nejmenší k takové, že $|C| \leq 2^k$. V takovém případě je délka komprimovaných dat rovna součinu počtu znaků a délky jednotlivého kódového slova.

Jinou možností je kódovat znaky slovy o nestejné délce. V případě kódových slov o nestejné délce je však třeba, aby žádné kódové slovo pro znak abecedy C nebylo prefixem jiného kódového slova (jinak be se ztížilo dokódování). V tomto případě je délka dat po kompresi rovna

$$\sum_{c \in C} c.freq \cdot |w(c)|,$$

kde $w(c)$ je kódové slovo znaku c a $|w(c)|$ je jeho délka.

Každý kód si můžeme představit jako binární strom T, kde listy jsou ohodnoceny znaky abecedy C, hrany symbolem 0 nebo 1 a to tak, že ohodnocení cesty od kořene stromu k listu c je kódové slovo znaku c. Délka dat po kompresi je pak dána výrazem

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c),$$

kde $d_T(c)$ je hloubka listu c ve stromě T.

Huffmanův kód je binární kód nestejné délky jehož binární strom T má nejmenší možnou hodnotu $B(T)$.

2.4.1 Konstrukce Huffmanova kódu.

Vstup: Máme dánu abecedu C, $n = |C|$, a četnosti $c.freq$ jednotlivých znaků $c \in C$ v textu.
Výstup: Strom T optimálního binárního kódu.

1. Vytvoříme n jednoprvkových stromů T_c, každý kořen je označený $c; c.freq$; $Q := C$; $\mathcal{T} := \{T_c \,|\, c \in C\}$.

2. Dokud $|Q| \neq 1$, vybereme $x \in Q$ s nejmenší hodnotou $x.freq$ a $y \in Q$ s druhou nejmenší hodnotou $y.freq$;
   do Q přidáme nový prvek z, položíme $z.freq := x.freq + y.freq$, a x, y odstraníme z Q;
   vytvoříme strom T_z s kořenem z (označeným $z; z.freq$) takto: levý podstrom z je strom T_x, pravý podstrom je strom T_y;
   z množiny $\mathcal{T}$ odebereme stromy T_x a T_y a přidáme strom T_z.

3. Pro $Q = \{q\}$ a $\mathcal{T} = \{T_q\}$ je T_q binární strom, který určuje binární kód takto: každou hranu do levého následníka označíme 0, do pravého následníka označíme 1. Položíme $T := T_q$.

2.4.2 Variant. Po každém průchodu bodem 2 má množina Q o jeden prvek méně (totéž platí pro množinu $\mathcal{T}$). Tedy po $n-1$ průchodech bodem 2 algoritmus skončí.

2.4.3 Invariant.

Tvrzení. Nechť C je abeceda a $c.freq$, $c \in C$, jsou frekvence výskytů znaků v datech. Nechť x a y jsou dva znaky s nejmenšími frekvencemi. Vytvoříme $C' = (C \setminus \{x, y\}) \cup \{z\}$, kde $z.freq = x.freq + y.freq$ a označíme T' optimální strom (tj. strom s nejmenším $B(T')$) pro C'.

Pak strom T, který jsme dostali z T' nahrazením vrcholu z stromem s kořenem z, levým následníkem x a pravým následníkem y, je optimální strom pro C.

Myšlenka důkazu. Dá se dokázat, že kdykoli ze stromu T' pro abecedu C' vytvoříme strom T tak, že list z s $z.freq = x.freq + y.freq$ nahradíme výše popsaným stromem (kořen z, levý podstrom x, pravý podstrom y), tak

$$B(T) = B(T') + x.freq + y.freq.$$

Nyní k dokončení důkazu potřebujeme vědět, že je vždy možné najít optimální kód pro abecedu C, takový, že v něm znaky x a y mají stejnou délku a liší se pouze v posledním bitu. A to říká následující lemma.

2.4.4 Lemma. Máme dánu abecedu C s frekvencemi $c.freq$. Nechť x a y jsou dva znaky s nejmenšími frekvencemi. Pak existuje optimální kód nestejné délky, kde kódová slova pro x a y mají stejnou délku a liší se pouze v posledním bitu.

Myšlenka důkazu. Označíme T strom optimálního kódu a označíme a, b ty prvky abecedy C, které jsou v poslední hladině stromu T, mají společného bezprostředního předchůdce a $a.freq \leq b.freq$. Platí $x.freq \leq a.freq$ a $y.freq \leq b.freq$.

Jestliže $x.freq = b.freq$, pak všechny prvky x, y, a, b mají stejnou frekvenci a můžeme vyměnit x s a a y s b a dostaneme strom se stejnou hodnotou B.

Předpokládejme, že $x.freq \neq b.freq$. Vytvoříme nový strom T' tak, že vyměníme x s a a y s b. Dá se spočítat, že

$$B(T)-B(T') = (a.freq-x.freq)(d_T(a)-d_T(x))+(b.freq-y.freq)(d_T(b)-d_T(y)).$$

Výraz na pravé straně je nezáporný. Kladný být nemůže (pak by strom T nebyl optimální — strom T' by měl menší hodnotu $B(T')$); proto je T' také optimální a lemma se dokázáno.

Kapitola 3

Turingovy stroje

Nejprve uvedeme klasický model, který předcházel moderní výpočetní techniku a velmi pomohl k jejímu rychlému vývoji. Jedná se o tzv. Turingův stroj, model zavedený ve 30. letech minulého století Alanem Turingem.

3.1 Deterministický Turingův stroj

3.1.1 Turingův stroj si můžeme představit takto: skládá se

- z řídící jednotky, která se může nacházet v jednom z konečně mnoha stavů,
- potenciálně nekonečné pásky (nekonečné na obě strany) rozdělené na jednotlivá pole a
- hlavy, která umožňuje číst obsah polí a přepisovat obsah polí pásky.

Na základě symbolu X, který čte hlava na pásce, a na základě stavu q, ve kterém se nachází řídící jednotka, se řídící jednotka Turingova stroje přesune do stavu p, hlava přepíše obsah čteného pole na Y a přesune se buď doprava nebo doleva (tato akce je popsána tzv. přechodovou funkcí).

3.1.2 Formální definice. Turingův stroj je $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, kde

- Q je konečná množina stavů,
- Σ je konečná množina vstupních symbolů,
- Γ je konečná množina páskových symbolů, přitom $\Sigma \subset \Gamma$,
- B je prázdný symbol (též nazývaný *blank*), jedná se o páskový symbol, který není vstupním symbolem, (tj. $B \in \Gamma \setminus \Sigma$),
- δ je přechodová funkce, tj. parciální zobrazení z množiny $(Q \setminus F) \times \Gamma$ do množiny $Q \times \Gamma \times \{L, R\}$, (zde L znamená pohyb hlavy o jedno pole doleva, R znamená pohyb hlavy o jedno pole doprava),
- $q_0 \in Q$ je počáteční stav a
- $F \subseteq Q$ je množina koncových stavů.

3.1.3 Situace TM. *Situace Turingova stroje* (též konfigurace TM, angličtině nazývaná instantaneous description (ID)), plně popisuje obsah pásky, pozice hlavy na pásce a stav, ve kterém se nachází řídící jednotka. Jedná se o

$$X_1\, X_2\, \ldots\, X_{i-1}\, q\, X_i\, X_{i+1}\, \ldots\, X_k,$$

kde symboly $X_1\, X_2\, \ldots\, X_k$ jsou páskové symboly a kromě nich jsou na pásce pouze blanky B; řídící jsednotka je ve stavu q a hlava čte symbol X_i.

3.1.4 Počáteční situace. Na začátku práce se Turingův stroj nachází v počátečním stavu q_0, na pásce má na n polích vstupní slovo $a_1\, a_2\, \ldots\, a_n$ ($a_i \in \Sigma$), ostatní pole obsahují blank B a hlava čte pole pásky se symbolem a_1. Tedy formálně počáteční situaci zapisujeme

$$q_0\, a_1\, \ldots\, a_n.$$

3.1.5 Krok Turingova stroje. Předpokládejme, že se Turingův stroj nachází v situaci

$$X_1\, X_2\, \ldots\, X_{i-1}\, q\, X_i\, \ldots\, X_k.$$

Pak na základě přechodové funkce TM v jednom kroku přejde do následující situace a to takto:

Jestliže $\delta(q, X_i)$ není definováno, TM se zastaví.

Jestliže $\delta(q, X_i) = (p, Y, R)$, TM se přesune do stavu p, na pásku místo symbolu X_i napíše symbol Y a hlavu posune o jedno pole doprava. Formálně to zapisujeme takto

$$X_1\, X_2\, \ldots\, X_{i-1}\, q\, X_i\, \ldots\, X_k \vdash X_1\, X_2\, \ldots\, X_{i-1}\, Y\, p\, X_{i+1}\, \ldots\, X_k. \tag{3.1}$$

Jestliže $\delta(q, X_i) = (p, Y, L)$, TM se přesune do stavu p, na pásku místo symbolu X_i napíše symbol Y a hlavu posune o jedno pole doleva. Formálně to zapisujeme takto

$$X_1\, X_2\, \ldots\, X_{i-1}\, q\, X_i\, \ldots\, X_k \vdash X_1\, \ldots\, X_{i-2}\, p\, X_{i-1}\, Y\, X_{i+1}\, \ldots\, X_k. \tag{3.2}$$

(Jestliže v případě 3.2 je $i = 1$, pak $q\, X_1\, \ldots\, X_k \vdash p\, B\, Y\, \ldots\, X_k$.)

3.1.6 Výpočet Turingova stroje nad slovem $w = a_1 a_2 \ldots a_k$, je konečná posloupnost jeho kroků, která začíná v počáteční situaci $q_0\, a_1\, \ldots\, a_k$.

Formálně se jedná o reflexivní a tranzitivní uzávěr $\vdash^\star$ relace $\vdash$ z 3.1.5 (na množině všech situací daného Turingova stroje).

Jestliže během výpočtu Turingova stroje nad slovem w se Turingův stroj dostane do jednoho z koncových stavů $q' \in F$, říkáme, že se TM *úspěšně zastavil*. Obsah pásky při úspěšném zastavení je *výstupem* TM, nad vstupem $w = a_1\, a_2 \ldots a_n$.

Jestliže se během výpočtu Turingův stroj zastaví ve stavu, který není koncový, říkáme, že se TM *neúspěšně zastavil*.

3.1.7 Definice — jazyk přijímaný TM. Vstupní slovo $w \in \Sigma^\star$ je *přijato* Turingovým strojem M, jestliže se Turingův stroj na slově w úspěšně zastaví.

Množina slov $w \in \Sigma^\star$, která Turingův stroj přijímá, se nazývá *jazyk přijímaný M* a značíme ji $L(M)$.

3.1.8 Definice — funkce realizovaná TM. Je dáno zobrazení $f\colon \Sigma^\star \to \Sigma^\star$. Řekneme, že TM M *realizuje* zobrazení f, jestliže pro každé $w \in \Sigma^\star$, pro které je $f(w)$ definováno, se M úspěšně zastaví s výstupem $f(w)$ (tj. $q_0 w \vdash^\star \alpha q_F \beta$, kde $\alpha\beta = f(w)$). Pro w, pro něž $f(w)$ není definováno, se M zastaví neúspěšně.

V případě, že f je funkce $f\colon \mathbb{N}^k \to \mathbb{N}$, tj. přiřazuje k-tici přirozených čísel $(n_1, n_2, \ldots n_k)$ přirozené číslo $f(n_1, \ldots, n_k)$, je vstupem TM slovo $w = 0^{n_1}10^{n_2}1\ldots10^{n_k}$. TM realizuje funkci f, jestliže se úspěšně zastaví nad slovem w v situaci, kdy na pásce je slovo $0^{f(n_1,\ldots,n_k)}$. Jestliže vstupní slovo není ve tvaru $0^{n_1}10^{n_2}1\ldots10^{n_k}$ nebo funkce není definovaná, TM se zastaví neúspěšně.

Poznámka. Někdy se požaduje, aby při úspěšném zastavení hlava TM četla první symbol slova $f(w)$, resp. $0^{f(n_1,\ldots,n_k)}$; my to nevyžadujeme; chceme pouze, aby na pásce zbylo slovo $f(w)$, resp. $0^{f(n_1,\ldots,n_k)}$ na sousedních polích pásky.

3.1.9 Časová složitost Turingova stroje je parciální zobrazení $T(n)$ z množiny všech přirozených čísel do sebe definované:

Jestliže pro nějaký vstup délky n se Turingův stroj nezastaví, $T(n)$ není definováno. V opačném případě je $T(n)$ rovno maximálnímu počtu kroků, po nichž dojde k zastavení Turingova stroje, kde maximum se bere přes všechny vstupy délky n.

3.1.10 Paměťová složitost Turingova stroje $S(n)$. Jestliže pro nějaký vstup délky n Turingův stroj použije nekonečnou část pásky (pak se nemůže v konečném čase zastavit), $S(n)$ není definováno. V opačném případě je $S(n)$ rovno největšímu rozdílu pořadových čísel polí, které byly během výpočtu použity, kde maximum se bere přes všechny vstupy délky n.

3.1.11 Výše jsme definovali, co je to jazyk přijímaný TM. Jedná se o množinu $L(M)$ všech slov w na nichž se TM úspěšně zastaví (tj. při výpočtu se dostane do koncového stavu).

Jestliže w je slovo, které v jazyce $L(M)$ neleží, TM se při práci nad ním může neúspěšně zastavit nebo nezastavit vůbec.

3.1.12 Jazyk přijímaný/rozhodovaný Turingovým strojem. Definice. Řekneme, že jazyk L je *přijímán* nějakým Turingovým strojem, jestliže existuje TM M takový, že $L = L(M)$.

Řekneme, že Turingův stroj *rozhoduje* jazyk L, jestliže tento jazyk přijímá a navíc se na každém vstupu zastaví.

3.1.13 Poznámky.

- Každý jazyk, který je rozhodován Turingovým strojem, je také tímto Turingovým strojem přijímán. Naopak to ale neplatí. Uvidíme, že existují jazyky, které jsou přijímány nějakým Turingovým strojem, ale neexistuje Turingův stroj, který by je rozhodl.
- Základní model Turingova stroje, tak jak jsme ho uvedli v minulých odstavcích, není jediným modelem. Jiná varianta Turingova stroje pracuje s nekonečnou páskou s pevným levým okrajem. U tohoto modelu se TM neúspěšně zastaví i v případě, že hlava čte nejvíc levé pole pásky

a přechodová funkce nařizuje pohyb hlavy doleva. Počáteční situace TM s pevným levým krajem má vždy vstupní slovo napsané na začátku pásky (tj. od levého okraje).

Další varianty umožňují hlavě Turingova stroje aby se nepohnula. To znamená, že přechodová funkce δ je parciální zobrazení z $(Q \setminus F) \times \Gamma$ do $Q \times \Gamma \times \{R, L, S\}$, kde symbol S znamená, že hlava čte stejné pole.

Všechny tyto modely jsou ekvivalentní v tom smyslu, že pro každý Turingův stroj M_1 jednoho typu existuje Turingův stroj M_2 jiného typu tak, že oba stroje realizují stejné zobrazení / přijímají nebo rozhodují stejný jazyk.

3.1.14 Techniky pro návrh Turingova stroje — informace pamatovaná stavem. Jestliže chceme pomocí TM zkontrolovat, zda se nějaký další symbol vstupního slova rovná/nerovná prvnímu symbolu, můžeme postupovat takto: stav, do kterého se dostaneme po přečtení 0, označíme $(q, 0)$; stav, do kterého se dostaneme po přečtení 1, označíme $(q, 1)$. Tím poznáme, jaký byl čtený symbol, jen z pojmenování stavu.

Je samozřejmé, že není nutné takové pojmenování zavádět. Jestliže se jedná o TM s pouze několika stavy, můžeme stav $(q, 0)$ označit q_1, $(q, 1)$ označit q_2 a informaci o symbolu, který byl přečten, zohledňuje přechodová funkce. Ovšem v případě, že pracujeme s TM o několika desítkách či stovkách stavů, je takového pojmenováni „mnemotechnickou pomůckou“.

3.1.15 Techniky pro návrh Turingova stroje — více stop. Pro zjednodušení práce na návrhu Turingových strojů si můžeme představit, že páska má vic stop. Formálně to znamená, že jednotlivý páskový symbol je vlastně dvojice (v případě dvou stop) nebo obecně m-tice (v případě m stop). Tvar takového páskového symbolu může nést další informace. Např. chceme-li jednoduše popsat páskový symbol, který znamená „zkontrolovaný“ vstupní symbol a, můžeme takový páskový symbol „pojmenovat“ $(\star, a)$, kde ta $\star$ nám kóduje fakt, že symbol a byl zkontrolován. Protože každý vstupní symbol a má být také páskovým symbolem, ztotožňujeme, v případě dvou stop, symbol a s dvojicí (B, a) a vlastní blank B s dvojicí (B, B).

Opět platí, že jsme to dělat nemuseli, mohli jsem pro „zkontrolovaný“ vstupní symbol a použít nějaký jiný znak, např. A, ale ve složitějších konstrukcích je využití více stop výhodné — použijeme více stop např. při konstrukci Turingova stroje s jednou páskou, který simuluje vícepáskový Turingův stroj zavedený v následujícím odstavci.

3.1.16 Turingův stroj s k páskami. Turingův stroj s k páskami se skládá z řídící jednotky, která se nachází v jednom z konečně mnoha stavů $q \in Q$, množiny vstupních symbolů Σ, množiny páskových symbolů Γ, přechodové funkce δ, počátečního stavu q_0, páskového symbolu B a množiny koncových stavů F. Dále je dáno k pásek a k hlav; i-tá hlava vždy čte jedno pole i-té pásky. Přechodová funkce δ je parciální zobrazení, které reaguje na stav, ve kterém se Turingův stroj nachází a na k-tici páskových symbolů, kterou jednotlivé hlavy snímají. (Formálně je δ parciální zobrazení, $\delta\colon (Q \setminus F) \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$).

Na začátku:

- Vstupní slovo je na první pásce; kromě vstupního slova obsahují všechna pole pásky blank B.
- Všechny ostatní pásky mají ve všech polích blank B.
- Řídící jednotka je v počátečním stavu q_0.
- První hlava čte první symbol vstupního slova.

3.1.17 Krok Turingova stroje s k páskami je určen přechodovou funkcí. Jestliže přechodová funkce je definována, pak (na základě přechodové funkce):

- Řídící jednotka se přesune do nového stavu.
- Každá hlava přepíše obsah pole, které čte (může i stejným symbolem).
- Každá hlava se posune doprava nebo doleva (přechodová funkce udává pohyb každé hlavy nezávisle na pohybech ostatních hlav).
- Jestliže se Turingův stroj nachází v koncovém stavu, přechodová funkce není definována a Turingův stroj se zastaví.

3.1.18 Jazyk přijímaný Turingovým strojem s k páskami. Obdobně jako pro Turingův stroj s jednou páskou definujeme:

Turingův stroj se *úspěšně zastaví*, jestliže řídící jednotka vstoupila do koncového stavu. Jestliže Turingův stroj nemá definován následující krok a není v koncovém stavu, říkáme, že se Turingův stroj zastavil *neúspěšně*.

Slovo $w \in \Sigma^\star$ je *přijímáno* Turingovým strojem, jestliže se na něm Turingův stroj úspěšně zastaví. Všechna slova přijímaná Turingovým strojem tvoří *jazyk přijímaný* tímto strojem.

Jestliže se navíc Turingův stroj na všech slovech zastaví, říkáme že Turingův stroj jazyk *rozhoduje*.

3.1.19 Poznámky. Na každý Turingův stroj s jednou páskou se můžeme dívat jako na Turingův stroj s k páskami, kde $k = 1$. Proto Turingův stroj s jednou páskou je zvláštní případ Turingova stroje s k páskami.

Stejně jako u Turingova stroje s jednou páskou i pro více pásek existuje několik variant — pásky mohou mít pevné levé konce, Turingův stroj v jednom kroku nemusí pohnout některou z hlav. Opět platí, že všechny tyto varianty „mají stejnou sílu“, tj. jestliže nějaký jazyk L je přijímán/rozhodován TM jednoho typu, je přijímán/rozhodován i TM druhého typu.

3.1.20 Věta. Ke každému Turingovu stroji M_1 s k páskami existuje Turingův stroj M_2 s jednou páskou, který má stejné chování jako M_1.

Navíc, jestliže M_1 potřeboval k úspěšnému zastavení n kroků, pak M_2 potřebuje $\mathcal{O}(n^2)$ kroků.

Myšlenka důkazu. Turingův stroj M_2 má jedinou pásku rozdělenu do $2k$ stop. Každá páska M_1 je simulována dvěma stopami M_2 – a to tak, že první stopa vždy obsahuje informaci o poloze odpovídající hlavy TM M_1, ve druhé stopě je obsah simulované pásky.

Simulace jednoho kroku TM M_1:

Hlava TM M_2 se nachází na pásce tak, že všechna pole s informací o poloze hlavy jsou nalevo. Hlava nejprve přejede pásku tak, aby stroj navštívil pozice všech hlav (a zapamatoval si obsahy odpovídajících polí). Tím získá všechny informace, které určují krok TM M_1.

Na základě přechodové funkce TM M_1 při postupu doleva změní nejen obsahy sudých stop, ale i posune označení polohy hlav buď o jedno pole doleva nebo doprava podle hodnoty přechodové funkce TM M_1.

Jestliže TM M_1 udělal od začátku práce n kroků, potřebuje TM M_2 na jeden krok maximálně $\mathcal{O}(n)$ kroků.

3.1.21 Nedeterministický Turingův stroj. Jestliže pro Turingův stroj (ať již s jednou páskou nebo s více páskami) připustíme, aby v jedné situaci mohl provést několik různých kroků, dostáváme nedeterministický Turingův stroj. Formálně zadefinujeme nedeterministický Turingův stroj (NTM) pouze pro variantu s jednou páskou, která je nekonečná na obě strany.

Nedeterministický Turingův stroj je sedmice $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$, kde

- Q je konečná množina stavů,
- Σ je konečná množina vstupních symbolů,
- Γ je konečná množina páskových symbolů, přitom $\Sigma \subset \Gamma$,
- B je prázdný symbol (též nazývaný *blank*), jedná se o páskový symbol, který není vstupním symbolem, (tj. $B \in \Gamma \setminus \Sigma$),
- δ je přechodová funkce, tj. parciální zobrazení z množiny $(Q \setminus F) \times \Gamma$ do množiny $\mathcal{P}_f(Q \times \Gamma \times \{L, R\})$ ($\mathcal{P}_f(X)$ je množina konečných podmnožin X),
- $q_0 \in Q$ je počáteční stav a
- $F \subseteq Q$ je množina koncových stavů.

Krok nedeterministického Turingova kroku je definován analogicky jako pro (deterministický) Turingův stroj:

Pro $(p, Y, R) \in \delta(q, X_i)$

$$X_1\, X_2\, \ldots\, X_{i-1}\, q\, X_i\, \ldots\, X_k \vdash X_1\, X_2\, \ldots\, X_{i-1}\, Y\, p\, X_{i+1}\, \ldots\, X_k. \tag{3.3}$$

Pro $(p, Y, L) \in \delta(q, X_i)$

$$X_1\, X_2\, \ldots\, X_{i-1}\, q\, X_i\, \ldots\, X_k \vdash X_1\, \ldots\, X_{i-2}\, p\, X_{i-1}\, Y\, X_{i+1}\, \ldots\, X_k. \tag{3.4}$$

3.1.22 Jazyk přijímaný nedeterministickým Turingovým strojem se skládá ze všech slov $w \in \Sigma^\star$, pro něž

$$q_0\, w \vdash^\star Y_1\, Y_2\, \ldots Y_i\, q_f\, Y_{i+1}\, \ldots\, Z_m,$$

pro některý koncový stav q_f.

Neformálně: slovo w je přijato nedeterministickým Turingovým strojem právě tehdy, když existuje „přijímací výpočet“, tj posloupnost kroků, po nichž se stroj dostane do koncového stavu.

Jestliže nedeterministický Turingův stroj M přijímá jazyk L a navíc každý jeho výpočet vždy končí po konečně mnoha krocích, říkáme, že M *rozhoduje* jazyk L.

3.1.23 Věta. Je-li jazyk L přijímán, resp. rozhodován nedeterministickým Turingovým strojem M, pak existuje deterministický Turingův stroj M_1 s jednou páskou, který L přijímá, resp, rozhoduje.

3.2 Počítač s libovolným přístupem – RAM

3.2.1 V tomto oddílu připomeneme další z formálních modelů algoritmu — počítač s libovolným přístupem (tzv. RAM), který je blíže „klasickému“ počítači než Turingův stroj. Platí, že vše, co lze přijmout/realizovat Turingovým strojem, lze „spočítat“ počítačem s libovolným přístupem. To nám dále dovolí volně přecházet mezi počítačovými programy a Turingovými stroji podle toho, který model bude pro danou situaci příhodnější.

3.2.2 Počítač s libovolným přístupem, též nazývaný *RAM* se skládá z programové jednotky, aritmetické jednotky, paměti a vstupní a výstupní jednotky.

3.2.3 Programová jednotka obsahuje programový registr a vlastní program (programový registr ukazuje na instrukci, která má být provedena).

3.2.4 Aritmetická jednotka provádí aritmetické operace sčítání, odčítání, násobení a celočíselné dělení.

3.2.5 Paměť je rozdělena na paměťové buňky, každá buňka může obsahovat celé číslo. Předpokládáme neomezený počet paměťových buněk a neomezenou velikost čísel uložených v paměťových buňkách. Pořadové číslo paměťové buňky je *adresa* této buňky.

Buňka s adresou 0 je *pracovní registr*, s adresou 1 je *indexový registr*.

3.2.6 Vstupní jednotka je tvořena vstupní páskou a hlavou. Vstupní páska je rozdělena na pole (v každém poli může být celé číslo). Hlava snímá v každém okamžiku jedno pole. Po přečtení pole se hlava posune o jedno pole doprava.

3.2.7 Výstupní jednotka je tvořena výstupní páskou a hlavou. Obdobně jako v případě vstupní jednotky je páska rozdělena na pole. Výstupní hlava zapíše číslo do pole výstupní pásky a posune se o jedno pole doprava.

3.2.8 Konfigurace počítače s libovolným přístupem je přiřazení, které každému poli vstupní i výstupní pásky, každé paměťové buňce a programovému registru přiřazuje celé číslo. *Počáteční konfigurace* je konfigurace, pro kterou existuje přirozené číslo n s následujícími vlastnostmi:

- kromě prvních n vstupních polí obsahují všechna pole, paměťové buňky číslo 0,
- programový registr obsahuje číslo 1,
- prvních n polí obsahuje vstup počítače.

3.2.9 Výpočet počítače s libovolným přístupem je posloupnost konfigurací, taková, že začíná počáteční konfigurací a každá následující konfigurace je určena programem počítače.

3.2.10 Program počítače s libovolným přístupem používá následující příkazy:

- příkazy přesunu: LOAD operand, STORE operand,
- aritmetické příkazy: ADD operand, SUBTRACT operand, MULTIPLY operand, DIVIDE operand,
- vstupní a výstupní příkazy: READ, WRITE,
- příkazy skoku: JUMP návěští, JZERO návěští, JGE návěští,
- příkazy zastavení: STOP, ACCEPT, REJECT.

3.2.11 Operand je buď číslo j, zapisujeme $= j$, nebo obsah j-té paměťové buňky, zapisujeme j, nebo obsah paměťové buňky s adresou $i+j$, kde i je obsah indexového registru, zapisujeme $*j$.

3.2.12 Návěští je přirozené číslo, které udává pořadové číslo instrukce, která bude prováděna, dojde-li ke skoku.

3.2.13 Časová složitost. Řekneme, že program P pro RAM pracuje s časovou složitostí $\mathcal{O}(f(n))$, jestliže pro každý vstup délky n je počet kroků počítače $T(n)$ ve třídě $\mathcal{O}(f(n))$.

3.2.14 Paměťová složitost. Řekneme, že program P pro RAM pracuje s pamětí velikosti m, jestliže během výpočtu nebyl proveden žádný příkaz, který by měl adresu operandu větší než m a byl proveden příkaz s adresou m. Dále řekneme, že program P pracuje s paměťovou složitostí $\mathcal{O}(g(n))$, jestliže pro každý vstup délky n program P pracuje s velikostí paměti ve třídě $\mathcal{O}(g(n))$.

3.2.15 Poznámka. Jestliže se na nějakém vstupu program pro RAM nezastaví, není definována ani časová ani paměťová složitost.

3.2.16 Věta. Ke každému Turingovu stroji M existuje program P pro RAM takový, že oba mají stejné chování. Navíc, jestliže M potřeboval n kroků, P má časovou složitost $\mathcal{O}(n^2)$.

3.2.17 Věta. Pro každý program P pro RAM existuje Turingův stroj M s pěti páskami takový, že P i M mají stejné chování.

3.2.18 Věta. Jestliže program P pro RAM splňuje následující podmínky:

- program obsahuje pouze instrukce, které zvětšují délku binárně zapsaného čísla maximálně o jednu;
- program obsahuje pouze instrukce, které Turingův stroj s více páskami provede na slovech délky k v $\mathcal{O}(k^2)$ krocích,

pak Turingův stroj z věty 3.2.17 simuluje n kroků programu P pomocí $\mathcal{O}(n^3)$ svých kroků.

3.2.19 Důsledek. Je dán program P pro RAM, který splňuje podmínky z věty 3.2.17. Pak existuje Turingův stroj s jednou páskou, který má stejné chování jako P a n kroků programu P simuluje pomocí $\mathcal{O}(n^6)$ svých kroků.

Kapitola 4

Třídy složitosti

4.1 Rozhodovací úlohy

4.1.1 Teorie složitosti pracuje zejména s tzv *rozhodovacími* úlohami. Rozhodovací úlohy jsou takové úlohy, jejichž „řešením“ je buď odpověď „ANO“ nebo odpověď „NE“.

4.1.2 Příklad. *SAT – splňování Booleovských formulí*: Je dána výroková formule φ v CNF. Rozhodněte, zda je φ splnitelná.

Na danou formuli φ je tedy odpověď (tj. řešení) buď „ANO“ nebo „NE“. Všimněte si, že v tomto případě se neptáme po ohodnocení, ve kterém je formule pravdivá – zajímá nás pouze fakt, zda je splnitelná.

4.1.3 Řada praktických úloh není podobného druhu jako uvedený příklad. Často se jedná o tzv. optimalizační úlohy, tj. úlohy, kde mezi přípustnými řešeními hledáme přípustné řešení v jistém smyslu optimální. Obvykle to bývá tak, že je dána účelová funkce, která každému přípustnému řešení přiřadí číselnou hodnotu, a úkolem je najít přípustné řešení, pro které je hodnota účelové funkce optimální, tj. buď největší nebo naopak nejmenší. V dalším textu se s řadou těchto úloh setkáme. Takovými úlohami jsou například úlohy nalezení minimální kostry v ohodnoceném neorientovaném grafu i nalezení nejkratších cest v daném ohodnoceném orientovaném grafu.

Nyní uvedeme další příklad.

4.1.4 Problém obchodního cestujícího – TSP. Jsou dána města 1,2, $\ldots, n$. Pro každou dvojici měst i, j je navíc dáno kladné číslo $d(i, j)$ (tak zvaná vzdálenost měst i, j). *Trasa* je dána permutací π množiny $\{1, 2, \ldots, n\}$ do sebe. Délka trasy T odpovídající permutaci π je

$$d(T) = \sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) + d(\pi(n), \pi(1)).$$

Neformálně, trasa je pořadí měst, ve kterém má obchodní cestující města projít, a to tak, aby každé město navštívil přesně jednou a vrátil se do toho města, ze kterého vyšel. Cena trasy je pak součtem všech vzdáleností, které při své cestě urazil.

4.1.5 Rozhodovací verze.

- *Minimální kostra:* Je dán neorientovaný graf $G = (V, E)$, ohodnocení $c\colon E \to \mathbb{N}$ a dále číslo K. Existuje minimální kostra, jejíž cena je nejvýše K?
- *Nejkratší cesty:* Je dána matice délek $\mathbf{A} = (a(i, j))$, výchozí vrchol r, cílový vrchol c a číslo K. Existuje cesta z vrcholu r do vrcholu c délky nejvýše K?
- *Problém obchodního cestujícího:* Kromě čísel $d(i, j)$ z 4.1.4 je dáno číslo K. Existuje trasa π délky nejvýše K?

4.1.6 Vyhodnocovací verze.

- *Minimální kostra:* Je dán neorientovaný graf $G = (V, E)$ a $c\colon E \to \mathbb{N}$. Najděte cenu minimální kostry ohodnoceného grafu.
- *Nejkratší cesty:* Je dána matice délek $\mathbf{A} = (a(i, j))$, výchozí vrchol r a cílový vrchol c. Najděte délku nejkratší cesty z vrcholu r do vrcholu c.
- *Problém obchodního cestujícího:* Jsou dána čísla $d(i, j)$ a 4.1.4. Najděte cenu optimální trasy, tj. trasy s nejmenší možnou délkou.

4.1.7 Optimalizační verze.

- Minimální kostra: Je dán neorientovaný graf $G = (V, E)$ a $c\colon E \to \mathbb{N}$. Najděte minimální kostru ohodnoceného grafu.
- Je dána matice délek $\mathbf{A} = (a(i, j))$, výchozí vrchol r, cílový vrchol c. Najděte nejkratší cestu z vrcholu r do vrcholu c.
- Jsou dána čísla $d(i, j)$ a 4.1.4. Najděte optimální trasu, tj. trasu s nejmenší možnou délkou.

4.1.8 Dá se dokázat, že když je kterákoli verze dané úlohy polynomiálně řešitelná, jsou polynomiálně řešitelné všechny tři verze. Ukážeme si to na příkladu obchodního cestujícího.

Předpokládejme, že existuje algoritmus $\mathcal{A}$, který rozhodne, zda pro instanci TSP a číslo K existuje trasa délky nejvýše K.

Uvažujme libovolnou instanci TSP. Označme d největší $d(i, j)$; dále označme $A := n \cdot d$, kde n je počet měst. Zavoláme algoritmus $\mathcal{A}$ pro $K := \lceil \frac{A}{2} \rceil$. Jestliže algoritmus $\mathcal{A}$ dá pro K odpověď „ano“, tak jako K volíme střed mezi 0 a K, jestliže algoritmus $\mathcal{A}$ dá pro K odpověď „ne“, tak jako K volíme střed mezi K a $2K$. Takto postupujeme tak dlouho, dokud nemá interval délku nula. Nyní je K hodnota optimální trasy, tj. řešení vyhodnocovací verze úlohy TSP. Uvědomte si, že vzhledem k tomu, že nás zajímají pouze **celočíselná** K, stane se to po maximálně $\lg(A) = \lg(n \cdot d)$ což je $\mathcal{O}(\lg(n))$ opakování.

Ukázali jsme, že po $\mathcal{O}(\lg(n))$ voláních algoritmu $\mathcal{A}$ známe hodnotu optimální trasy, označme ji D_{opt}.

Uvažujme úplný graf G na množině $V = \{1, \ldots, n\}$ ohodnocený délkami $d(i, j)$. Nyní „zorientujeme hrany“ a to tak, že hraně $\{i, j\}$, kde $i < j$, přiřadíme uspořádanou dvojici (i, j),a tyto dvojice uspořádáme lexikograficky. Probíráme dvojice (i, j) v tomto pořadí a pro každou dvojici vytvoříme novou instanci $I_{i,j}$

TSP tak, že z v předchozí instanci změníme pouze délku $d(i,j)$ na hodnotu $d(i,j) := n \cdot d$. Zavoláme algoritmus $\mathcal{A}$ na instanci $I_{i,j}$ a $K = D_{opt}$. Jestliže algoritmus $\mathcal{A}$ odpoví „ano“, hraně (i,j) ponecháme tuto novou délku. Jestliže algoritmus $\mathcal{A}$ odpoví „ne“, hraně (i,j) vrátíme původní délku a přejdeme na další dvojici v uspořádání. V okamžiku, kdy máme pouze n hran s původní délkou, těchto n hran tvoří (některou) optimální trasu TSP.

Uvědomte si, že v druhé části jsme použili pouze $\mathcal{O}(n^2)$ volání algoritmu $\mathcal{A}$. Odtud dostáváme: Kdyby existoval polynomiální algoritmus na řešení rozhodovací verze TSP, pak existuje i polynomiální algoritmus na řešení optimalizační verze TSP.

4.2 Třídy $\mathcal{P}$ a $\mathcal{NP}$

4.2.1 Instance úlohy jako slovo nad vhodnou abecedou. Instance libovolné rozhodovací úlohy můžeme zakódovat jako slova nad vhodnou abecedou. Ukažme si to na příkladě problému SAT a úlohy nalezení nejkratší cesty v daném orientovaném ohodnoceném grafu.

- Pro problém SAT (splňování booleovských formulí) je instancí libovolná formule φ v konjunktivním normálním tvaru (CNF). Označme jednotlivé logické proměnné formule φ jako x_1, x_2, $\ldots, x_n$. Pak φ můžeme zakódovat jako slovo nad abecedou $\{x, 0, 1, (,), \vee, \wedge, \neg\}$ takto: proměnná x_i se zakóduje slovem xw, kde w je binární zápis čísla i, ostatní symboly jsou zachovány.

  Například formuli $\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_4)$ odpovídá slovo

  $$(x1 \vee \neg x10 \vee x11) \wedge (\neg x1 \vee x100).$$

- U úlohy nalezení nejkratší cesty z vrcholu r do vrcholu c můžeme postupovat takto: Instanci tvoří matice délek daného orientovaného ohodnoceného grafu, dvojice vrcholů r a c a číslo k. Matici není těžké zakódovat jako slovo, za ní pak následuje pořadové číslo vrcholu r, pořadové číslo vrcholu c a číslo k, vše oddělené např. symbolem #.

4.2.2 Úloha jako jazyk nad abecedou. Protože řešením rozhodovací úlohy je buď „ANO“ nebo „NE“, rozdělíme instance úlohy na tzv. „ANO-instance“ a „NE-instance“. *Jazyk* úlohy $\mathcal{U}$, značíme jej $L_{\mathcal{U}}$, se skládá ze všech slov odpovídajících ANO-instancím úlohy $\mathcal{U}$.

Uvědomte si, že některá slova nad abecedou Σ nemusí odpovídat žádné instanci dané úlohy. Tato slova chápeme jako „NE-instance“. Můžeme proto říci, že množina všech NE instancí tvoří doplněk jazyka $L_{\mathcal{U}}$, tj. je to $\Sigma^\star \setminus L_{\mathcal{U}}$.

4.2.3 Třída $\mathcal{P}$. Řekneme, že rozhodovací úloha $\mathcal{U}$ leží ve třídě $\mathcal{P}$, jestliže existuje deterministický Turingův stroj, který rozhodne jazyk $L_{\mathcal{U}}$ a pracuje v polynomiálním čase; tj. funkce $T(n)$ je $\mathcal{O}(p(n))$ pro nějaký polynom $p(n)$.

4.2.4 Příklady.

- *Minimální kostra v grafu.* Je dán neorientovaný graf G s ohodnocením hran c. Je dáno číslo k. Existuje kostra grafu ceny menší nebo rovno k?

- *Nejkratší cesty v acyklickém grafu.* Je dán acyklický graf s ohodnocením hran a. Jsou dány vrcholy r a c. Je dáno číslo k. Existuje orientovaná cesta z vrcholu r do vrcholu c délky menší nebo rovno k?
- *Toky v sítích.* Je dána síť s horním omezením c, dolním omezením l, se zdrojem z a spotřebičem s. Dále je dáno číslo k. Existuje přípustný tok od z do s velikosti alespoň k?
- *Minimální řez.* Je dána síť s horním omezením c, dolním omezením l. Dále je dáno číslo k. Existuje řez, který má kapacitu menší nebo rovnu k?

Uvedli jsme všechny úlohy v rozhodovací verzi. Velmi často se mluví i o jejich optimalizačních verzích jako o polynomiálně řešitelných úlohách.

4.2.5 Třída $\mathcal{NP}$. Řekneme, že rozhodovací úloha $\mathcal{U}$ leží ve třídě $\mathcal{NP}$, jestliže existuje nedeterministický Turingův stroj, který rozhodne jazyk $L_{\mathcal{U}}$ a pracuje v polynomiálním čase.

4.2.6 Poznámka. V definici 4.2.3 jsme místo existence Turingova stroje mohli požadovat existenci programu P pro RAM, který řeší $\mathcal{U}$ v polynomiálním čase. Abychom přiblížili, které jazyky (rozhodovací úlohy) leží ve třídě $\mathcal{NP}$, zavedeme pojem nedeterministického algoritmu jako analogii RAM.

4.2.7 Nedeterministický algoritmus pracuje ve dvou fázích,

1. Algoritmus náhodně vygeneruje řetězec s (odpovídá řešení dané úlohy).
2. Deterministický algoritmus (Turingův stroj, program pro RAM) na základě vstupu a řetězce s dá odpověď ANO nebo NEVIM. (Deterministicky a polynomiálně ověří řešení.)

Řekneme, že nedeterministický algoritmus řeší úlohu $\mathcal{U}$, jestliže

1. Pro každou ANO instanci úlohy $\mathcal{U}$ existuje řetězec s, na jehož základě algoritmus dá odpověď ANO.
2. Pro žádnou NE instanci úlohy $\mathcal{U}$ neexistuje řetězec s, na jehož základě algoritmus dá odpověď ANO.

Řekneme, že nedeterministický algoritmus *pracuje v čase* $\mathcal{O}(T(n))$, jestliže každý průchod oběma fázemi 1 a 2 pro instanci velikosti n potřebuje $\mathcal{O}(T(n))$ kroků. □

4.2.8 Poznámka. Fakt, že nedeterministický algoritmus pracuje v polynomiálním čase, znamená, že každá z fází vyžaduje polynomiální čas a tudíž i řetězec s musí mít polynomiální délku (vzhledem k velikosti instance).

V definici 4.2.5 jsme místo existence nedeterministického Turingova stroje mohli požadovat existenci nedeterministického algoritmu, který řeší úlohu $\mathcal{U}$ v polynomiálním čase.

4.2.9 Příklady $\mathcal{NP}$ úloh.

- *Kliky v grafu.* Je dán neorientovaný graf G a číslo k. Existuje klika v grafu G o alespoň k vrcholech?

- *Nejkratší cesty v obecném grafu.* Je dán orientovaný graf s ohodnocením hran a. Jsou dány vrcholy r a v. Je dáno číslo k. Existuje orientovaná cesta z vrcholu r do vrcholu v délky menší nebo rovno k?
- *k-barevnost.* Je dán neorientovaný graf G. Je graf G k-barevný?
- *Problém batohu.* Je dáno n předmětů $1, 2, \ldots, n$. Každý předmět i má cenu c_i a váhu w_i. Dále jsou dána čísla A a B. Je možné vybrat předměty tak, aby celková váha nepřevýšila A a celková cena byla alespoň B? Přesněji, existuje podmnožina předmětů $I \subseteq \{1, 2, \ldots, n\}$ taková, že

$$\sum_{i \in I} w_i \leq A \quad \text{a} \quad \sum_{i \in I} c_i \geq B?$$

4.2.10 Cookova věta. Úloha SAT, splňování formulí v konjunktivním normálním tvaru, je $\mathcal{NP}$ úplná úloha.

4.2.11 Důkaz. Není těžké se přesvědčit, že úloha SAT je ve třídě $\mathcal{NP}$. První fáze nedeterministického algoritmu vygeneruje ohodnocení logických proměnných a na základě tohoto ohodnocení jsme schopni v polynomiálním čase ověřit, zda je v tomto ohodnocení formule pravdivá nebo ne.

V druhé části důkazu ukážeme, že každá $\mathcal{NP}$ úloha se polynomiálně redukuje na problém SAT. Pro každý $\mathcal{NP}$ problém $\mathcal{U}$ existuje nedeterministický Turingův stroj M, který přijímá jazyk této úlohy $L_{\mathcal{U}}$ v polynomiálním čase. To znamená, že pro každé slovo w délky n nad danou abecedou se M zastaví po maximálně $p(n)$ krocích, a to buď v koncovém stavu, jestliže $w \in L_{\mathcal{U}}$, nebo v nekoncovém stavu, jestliže $w \notin L_{\mathcal{U}}$.

Proto stačí pro každý nedeterministický Turingův stroj M a dané slovo w zkonstruovat formuli $\varphi_{M,w}$ v CNF takovou, že

$$w \in L(M) \quad \text{právě tehdy, když } \varphi_{M,w} \text{ je splnitelná}$$

Je dán nedeterministický Turingův stroj M s množinou stavů Q, vstupní abecedou Σ, páskovou abecedou Γ, přechodovou funkcí δ, počátečním stavem q_0 a koncovým stavem q_f. Předpokládejme, že M přijímá slovo w a potřebuje přitom $p(n)$ kroků.

Zavedeme logické proměnné:

- $h_{i,j}$, $i = 0, 1, \ldots, p(n)$, $j = 1, 2, \ldots, p(n)$; fakt, že hodnota proměnné $h_{i,j}$ je rovna 1 znamená, že hlava Turingova stroje v čase i čte j-té pole pásky.
- s_i^q, $i = 0, 1, \ldots, p(n)$, $q \in Q$; fakt, že hodnota proměnné s_i^q je rovna 1 znamená, že Turingův stroj v čase i je ve stavu q.
- $t_{i,j}^A$, $i = 0, 1, \ldots, p(n)$, $j = 1, 2, \ldots, p(n)$, $A \in \Gamma$; fakt, že hodnota proměnné $t_{i,j}^A$ rovna 1 znamená, že v čase i v j-tém poli pásky je páskový symbol A.

Nyní je třeba formulemi popsat následující fakta:

1. V každém okamžiku je Turingův stroj v právě jednom stavu.

2. V každém okamžiku čte hlava Turingova stroje právě jedno pole vstupní pásky.
3. V každém okamžiku je na každém poli pásky Turingova stroje právě jeden páskový symbol.
4. Na začátku práce (tj. v čase 0) je Turingův stroj ve stavu q_0, hlava čte první pole pásky a na pásce je na prvních n polích vstupní slovo, ostatní pole pásky obsahují B.
5. Krok Turingova stroje je určen přechodovou funkcí, tj. stav stroje, obsah čteného pole a poloha hlavy v čase $i+1$ je dána přechodovou funkcí.
6. V polích pásky, které v čase i hlava nečte, je obsah v čase $i+1$ stejný jako v i.
7. Jestliže se M dostane v čase i do koncového stavu, pak i v čase $i+1$ v koncovém stavu zůstane.
8. Na konci práce Turingova stroje, tj. v čase $p(n)$, je stroj ve stavu q_f.

Ukážeme jak utvořit formule pro jednotlivé body

Bod 1. V okamžiku i je Turingův stroj v aspoň jednom stavu:

$$\bigvee_{q\in Q} s_i^q.$$

V okamžiku i Turingův stroj není ve dvou různých stavech:

$$\bigwedge_{q\neq q'} (\neg s_i^q \vee \neg s_i^{q'}).$$

Nyní fakt, že Turingův stroj je v okamžiku i v právě jednom stavu je konjunkce obou výše uvedených formulí:

$$\varphi_1^i = (\bigvee_{q\in Q} s_i^q) \wedge \bigwedge_{q\neq q'} (\neg s_i^q \vee \neg s_i^{q'}).$$

Formule $\varphi_1 = \bigwedge_i \Big((\bigvee_{q\in Q} s_i^q) \wedge \bigwedge_{q\neq q'} (\neg s_i^q \vee \neg s_i^{q'})\Big)$.

Bod 2. V okamžiku i čte hlava Turingova stroje aspoň jedno pole pásky:

$$\bigvee_{1\leq j\leq p(n)} h_{i,j}.$$

V okamžiku i nečte hlava Turingova stroje dvě různá pole:

$$\bigwedge_{j\neq k} (\neg h_{i,j} \vee \neg h_{i,k}).$$

Nyní fakt, že hlava Turingova stroje v okamžiku i čte přesně jedno pole pásky je konjunkce obou výše uvedených formulí:

$$\varphi_2^i = (\bigvee_{1\leq j\leq p(n)} h_{i,j}) \wedge \bigwedge_{j\neq k} (\neg h_{i,j} \vee \neg h_{i,k}).$$

Formule $\varphi_2 = \bigwedge_i \Big((\bigvee_{1 \le j \le p(n)} h_{i,j}) \wedge \bigwedge_{j \neq k} (\neg h_{i,j} \vee \neg h_{i,k}) \Big)$.

Bod 3. V okamžiku i je v j-tém poli pásky Turingova stroje aspoň jeden páskový symbol:

$$\bigvee_{A \in \Gamma} t_{i,j}^A.$$

V okamžiku i v j-tém poli pásky Turingova stroje nejsou dva různé páskové symboly:

$$\bigwedge_{A \neq A'} (\neg t_{i,j}^A \vee \neg t_{i,j}^{A'}).$$

Nyní fakt, že Turingův stroj má v okamžiku i v j-tém poli právě jeden páskový symbol je konjunkce obou výše uvedených formulí:

$$\varphi_3^{i,j} = (\bigvee_{A \in \Gamma} t_{i,j}^A) \wedge \bigwedge_{A \neq A'} (\neg t_{i,j}^A \vee \neg t_{i,j}^{A'}).$$

Formule $\varphi_3 = \bigwedge_i \bigwedge_j \Big((\bigvee_{A \in \Gamma} t_{i,j}^A) \wedge \bigwedge_{A \neq A'} (\neg t_{i,j}^A \vee \neg t_{i,j}^{A'}) \Big)$.

Bod 4. Na začátku práce (tj. v čase 0) je Turingův stroj ve stavu q_0, hlava čte první pole pásky a na pásce je na prvních n polích vstupní slovo $a_1 a_2 \dots a_n$, ostatní pole obsahují B.

$$\varphi_4 = s_0^{q_0} \wedge h_{0,1} \wedge t_{0,1}^{a_1} \wedge \dots \wedge t_{0,n}^{a_n} \wedge t_{0,n+1}^B \wedge \dots \wedge t_{0,p(n)}^B.$$

Bod 5. Jestliže Turingův stroj je v čase i ve stavu q, hlava je na j-tém poli pásky, hlava čte páskový symbol A a $\delta(q, A)$ se skládá z trojic (p, C, D) (zde $D = 1$ znamená posun hlavy doprava, $D = -1$ znamená posun hlavy doleva), pak formule má tvar:

$$\bigwedge_j \bigwedge_{A \in \Gamma} ((s_i^q \wedge h_{i,j} \wedge t_{i,j}^A) \Rightarrow \bigvee (s_{i+1}^p \wedge t_{i+1,j}^C \wedge h_{i+1,j+D})).$$

Formule φ_5 je konjungce všech výše uvedených formulí pro všechny i, $i < p(n)$, j, $q \in Q$ a $A \in \Gamma$.

Bod 6. Obsah polí, které hlava nečte, zůstává v čase $i + 1$ stejný:

$$\bigwedge_j \bigwedge_{A \in \Gamma} ((\neg h_{i,j} \wedge t_{i,j}^A) \Rightarrow t_{i+1,j}^A).$$

Formule φ_6 je konjungce všech výše uvedených formulí pro všechny i, $i < p(n)$, j a $A \in \Gamma$.

Bod 7. Jestliže se stroj dostane do koncového stavu, už v něm zůstává.

$$s_i^{g_f} \Rightarrow s_{i+1}^{q_f}, \;\; g_f \in F.$$

Formule φ_7 je konjunkce všech výše uvedených formulí pro všechna i, $i < p(n)$.

Bod 8. Na konci práce Turingova stroje, tj. v čase $p(n)$ je stroj ve stavu q_f.

$$\varphi_8 = s_{p(n)}^{q_f}.$$

Výslednou formuli dostaneme jako konjunkci formulí

$$\varphi_{M,w} = \bigwedge_{k=1}^{8} \varphi_k.$$

4.3 Převody úloh

4.3.1 Metoda

Ukázat, že rozhodovací úloha $\mathcal{V}$ je $\mathcal{NP}$ úplná, je potřeba a stačí

1. ověřit, že $\mathcal{V} \in \mathcal{NP}$;
2. najít $\mathcal{NP}$ úplnou úlohu $\mathcal{U}$, pro kterou

$$\mathcal{U} \lhd_p \mathcal{V}.$$

Zatím jediná $\mathcal{NP}$ úplná úloha, kterou známe, je SAT, splňování booleovských formulí v konjunktivním normálním tvaru. Ukážeme řadu polynomiálních redukcí a tím ukážeme, že i další rozhodovací úlohy jsou $\mathcal{NP}$ úplné.

4.3.2 $3 - CNF\ SAT$.

Úloha: Je dána formule φ v konjunktivním normálním tvaru, kde každá klauzule má 3 literály.
Otázka: Je formule φ splnitelná?

4.3.3 Tvrzení. Platí

$$SAT \lhd_p 3 - CNF\ SAT.$$

4.3.4 Nástin převodu SAT na $3 - CNF\ SAT$.

Je dána formule φ v konjunktivním normálním tvaru. Zkonstruujeme formuli ψ, která

1. je v konjunktivním normálním tvaru, kde každá klauzule obsahuje maximálně 3 literály;
2. je splnitelná právě tehdy, když je splnitelná formule φ.

Označme $C_1, C_2, \ldots, C_k$ všechny klauzule formule φ. Jestliže každá z klauzulí obsahuje nejvýše 3 literály, nemusíme nic konstruovat, v tomto případě je $\psi = \varphi$.

Pro každou klauzuli C_i, která obsahuje víc než 3 literály, sestrojíme formuli ψ_{C_i} takto: Nechť $C_i = l_1 \vee l_2 \vee \ldots \vee l_s$, kde l_j jsou literály. Zavedeme nové logické proměnné $x_1, x_2, \ldots, x_{s-3}$ a položíme

$$\psi_{C_i} = (l_1 \vee l_2 \vee x_1) \wedge (\neg x_1 \vee l_3 \vee x_2) \wedge (\neg x_2 \vee l_4 \vee x_3) \wedge \ldots \wedge (\neg x_{s-3} \vee l_{s-1} \vee l_s).$$

Platí: Formule ψ_{C_i} je splnitelná právě tehdy, když C_i je splnitelná.

Formuli ψ dostaneme jako konjunkci všech klauzulí formule φ, které mají nejvýše 3 literály a formulí ψ_{C_i} pro klauzule C_i o více než 3 literálech.

Předpokládejme, že formule φ má k klauzulí a nejdelší klauzule má s literálů. Pak v konstrukci ψ jsme přidali maximálně $(s-3)k$ nových logických proměnných (rovnost nastává v případě, že každá z klauzulí formule φ obsahuje přesně $s > 3$ literálů). Navíc jsme formuli prodloužili o maximálně o $2(s-3)k$ literálů (každá nová logická proměnná se ve formuli ψ objevuje přesně dvakrát). Tedy délka formule ψ se pouze polynomiálně zvětšila vzhledem k délce formule φ.

4.3.5 Důsledek.

Protože úloha $3 - CNF\ SAT$ je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.6 Obarvení vrcholů grafu. Je dán prostý neorientovaný graf bez smyček $G = (V, E)$. *Obarvení vrcholů* grafu G je přiřazení, které každému vrcholu v grafu G přiřazuje jeho barvu $b(v)$, $b(v)$ je prvek množiny (barev) B, pro které platí, že žádné dva vrcholy spojené hranou nemají stejnou barvu. (Jinými slovy, jestliže $\{u, v\}$ je hrana grafu G, pak $b(u) \neq b(v)$.)

Graf G se nazývá *k-barevný*, jestliže jeho vrcholy je možné obarvit k barvami (tj. množina B má k prvků).

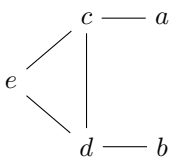
4.3.7 k-barevnost.
Úloha: Je dán prostý neorientovaný graf G bez smyček a číslo k.
Otázka: Je graf G k-barevný?

4.3.8 Tvrzení. Platí

$$3 - CNF\ SAT \lhd_p \text{3-barevnost.}$$

4.3.9 Základní myšlenka převodu. Je dána formule φ, která je v CNF a každá klauzule má 2 nebo 3 literály. K důkazu je třeba zkonstruovat prostý neorientovaný graf G bez smyček takový, že φ je splnitelná právě tehdy, když G je 3-barevný.

Konstrukce využívá pomocný graf G_1 o pěti vrcholech $\{a, b, c, d, e\}$ a pěti hranách



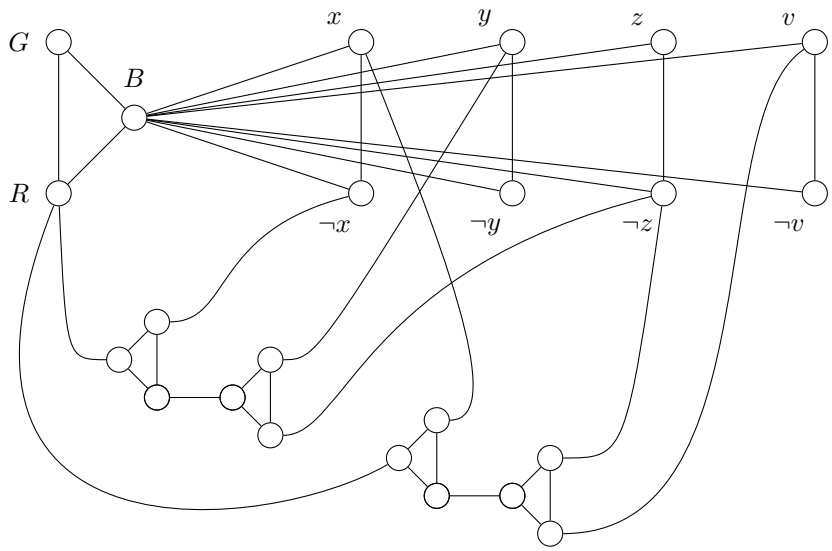
s touto vlastností:

- Jestliže vrcholy a a b mají stejnou barvu, pak tuto barvu musí mít i vrchol e.
- Jestliže jeden z vrcholů a a b má barvu z, pak lze tento graf obarvit tak, aby i vrchol e měl barvu z.

Mějme formuli φ, označme $x_1, x_2, \ldots, x_n$ všechny logické proměnné, které se ve formuli φ vyskytují. Vytvoříme neorientovaný graf $G = (V, E)$, kde

- V obsahuje všechny literály, tj. $x_1, \neg x_1, \ldots, x_n, \neg x_n$, vrcholy R, G, B.
- E obsahuje hrany tak, že R, G, B, $B, x_i, \neg x_i$ pro každé $i = 1, \ldots, n$ tvoří trojúhelník.
- Pro každou klauzuli obsahující literály l_1, l_2, l_3 přidáme do grafu dvě kopie pomocného grafu G_1 a to takto: Literály l_2 a l_3 odpovídají vrcholům a a b první kopie pomocného grafu G_1, vrcholy l_1 a e odpovídají vrcholům a, b a vrchol R je spojen hranou s vrcholem e druhé kopii grafu G_1,

Příklad grafu G pro dvě klauzule $C_1 = \neg z \vee y \vee \neg x$ a $C_2 = t \vee \neg z \vee x$ (x, y, z, t jsou logické proměnné) je na následujícím obrázku.



Předpokládejme, že formule φ je splnitelná; máme tedy pravdivostní ohodnocení, ve kterém je φ pravdivá. Obarvíme graf G třemi barvami z (zelená),c (červená) a m (modrá) takto:

- Vrcholy R, G, B: $b(R) = c$, $b(G) = z$, $b(B) = m$.
- Vrchol odpovídající literálu l má barvu z právě tehdy, když je l pravdivý, v opačném případě jej obarvíme c.

Protože každá klauzule obsahuje alespoň jeden literál, který je pravdivý, tj. jeho vrchol je obarven barvou z, je možné obarvit i zbývající vrcholy tak, aby G byl tříbarevný.

Předpokládejme, že graf G je tříbarevný. Přejmenujme barvy tak, aby platilo: $b(R) = c$, $b(G) = z$, $b(B) = m$. Nyní definujeme pravdivostní ohodnocení logických proměnných $x_1, x_2, \ldots, x_n$ takto:

proměnná x_i je pravdivá iff $b(x_i) = z$ a proměnná x_i je nepravdivá iff $b(x_i) = c$.

Z vlastností pomocného grafu G_1 vyplývá, že v každé klauzuli je alespoň jeden literál, který je obarven barvou z, tudíž je pravdivý.

Není těžké nahlédnout, že počet vrcholů i hran grafu G je polynomiální vůči délce formule φ.

4.3.10 Důsledek. Protože 3-barevnost je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.11 Tvrzení. Platí

$$\text{3-barevnost} \lhd_p \mathit{ILP}.$$

4.3.12 Převod 3-barevnosti na ILP. Je dán prostý neorientovaný graf bez smyček $G = (V, E)$. Zkonstruujeme instanci I úlohy celočíselného lineárního programování takovou, že I má přípustné řešení právě tehdy, když graf G je 3-barevný.

Všechny proměnné budou nabývat hodnot 0 nebo 1 (tj. bude se jednat o tzv. 0-1 celočíselné lineární programování).

Proměnné: Pro každý vrchol $v \in V$ zavedeme tři proměnné:

$$x_v^c, x_v^m, x_v^z.$$

Význam: Fakt, že proměnná x_v^b je rovna 1, $b \in \{c, m, z\}$, znamená, že vrchol v má barvu b.

Podmínky:

- Pro každý vrchol $v \in V$ máme rovnici, která zaručuje, že vrchol v má právě jednu barvu – buď c nebo m nebo z:

$$x_v^c + x_v^m + x_v^z = 1.$$

- Pro každou hranu $e = \{u, v\}$ máme tři nerovnosti (pro každou barvu jednu) zaručující, že oba vrcholy u a v nemohou mít stejnou barvu:

$$x_u^c + x_v^c \leq 1, \;\; x_u^m + x_v^m \leq 1, \;\; x_u^z + x_v^z \leq 1.$$

Platí: Graf G je 3-barevný právě tehdy, když I má přípustné řešení.

Instance I má $3\,|V|$ proměnných a $|V| + 3\,|E|$ podmínek. Jedná se tedy o instanci velikosti $\mathcal{O}(n + m)$, kde $n = |V|$ a $m = |E|$.

4.3.13 Důsledek. Protože ILP je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.14 Problém rozkladu.
Úloha: Je dána konečná množina X a systém jejích podmnožin $\mathcal{S}$.

Otázka: Je možné z $\mathcal{S}$ vybrat prvky tak, že tvoří rozklad množiny X? Jinými slovy, existuje $\mathcal{A} \subseteq \mathcal{S}$ tak, že $\mathcal{A}$ je rozklad množiny X?

4.3.15 Tvrzení. Platí

$$\text{3-barevnost} \;\lhd_p\; \text{problém rozkladu.}$$

4.3.16 Převod 3-barevnosti na problém rozkladu. Je dán neorientovaný prostý graf bez smyček $G = (V, E)$. Zkonstruujeme množinu X a systém jejích podmnožin $\mathcal{S}$ tak, že graf G je tříbarevný právě tehdy, když ze systému $\mathcal{S}$ lze vybrat rozklad množiny X.

Množina X:

- Pro každý vrchol $v \in V$ dáme do množiny X prvky

$$v, p_v^c, p_v^m, p_v^z.$$

- Pro každou hranu $e = \{u, v\}$ dáme do množiny X prvky

$$q^c_{uv}, q^m_{uv}, q^z_{uv}, q^c_{vu}, q^m_{vu}, q^z_{vu}.$$

Množina X má $4\,|V| + 6\,|E|$ prvků.

Systém podmnožin $\mathcal{S}$ tvoří tyto množiny:

1) Pro každý vrchol $v \in V$:

$$\{v, p^c_v\}, \{v, p^m_v\}, \{v, p^z_v\}.$$

2) Pro každý vrchol $v \in V$ označme $N(v)$ množinu všech sousedů vrcholu v (tj. $N(v) = \{u \,|\, \{u, v\} \in E\}$). Do $\mathcal{S}$ dáme množiny:

$$S^c_v = \{p^c_v, q^c_{vu} \,|\, u \in N(v)\}, S^m_v = \{p^m_v, q^m_{vu} \,|\, u \in N(v)\}, S^z_v = \{p^z_v, q^z_{vu} \,|\, u \in N(v)\}.$$

3) Pro každou hranu $e = \{u, v\}$ dáme do $\mathcal{S}$ množiny:

$$\{q^c_{uv}, q^m_{vu}\}, \{q^c_{uv}, q^z_{vu}\}, \{q^m_{uv}, q^c_{vu}\}, \{q^m_{uv}, q^z_{vu}\}, \{q^z_{uv}, q^c_{vu}\}, \{q^z_{uv}, q^m_{vu}\}.$$

Systém $\mathcal{S}$ má $3\,|V|$ množin z 1), $3\,|V|$ množin z 2) a $6\,|E|$ množin z 3).

Je-li graf G 3-barevný, je možné jeho vrcholy obarvit barvami $\{c, m, z\}$. Označme $b(v)$ barvu vrcholu $v \in V$. Z systému $\mathcal{S}$ vybereme $\mathcal{A}$ takto:

$\mathcal{A}$ se skládá z:

1) $\{v, p^{b(v)}_v\}$ pro všechny $v \in V$,

2) $S^{b_1}_v$ a $S^{b_2}_v$, kde b_1 a b_2 jsou zbylé dvě barvy, kterými není obarven v,

3) $\{q^{b(u)}_{uv}, q^{b(v)}_{vu}\}$ pro každou hranu $e = \{u, v\}$.

Uvědomte si, že protože b je obarvení, množiny ze 3) jsou v $\mathcal{S}$.

Jestliže existuje rozklad $\mathcal{A} \subseteq \mathcal{S}$ množiny X, pak sestrojíme obarvení grafu G takto:

$$b(v) := b, b \in \{c, m, z\} \quad \text{právě tehdy, když} \quad \{v, p^b_v\} \in \mathcal{A}.$$

Není těžké dokázat, že z volby systému $\mathcal{S}$ a $\mathcal{A}$ vyplývá: b je obarvení vrcholů třemi barvami.

4.3.17 Důsledek. Protože problém rozkladu je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.18 SubsetSum.
Úloha: Jsou dána kladná čísla a_1, a_2, …, a_n a číslo K.
Otázka: Lze vybrat podmnožinu čísel a_1, a_2, …, a_n tak, aby jejich součet byl roven číslu K?

Jinými slovy, existuje $J \subseteq \{1, 2, \ldots, n\}$ tak, že

$$\sum_{i \in J} a_i = K.$$

4.3.19 Tvrzení. Platí

$$\text{problém rozkladu} \lhd_p \text{SubsetSum}.$$

4.3.20 Převod problému rozkladu na SubsetSum. Je dána konečná množina X a systém jejích podmnožin $\mathcal{S}$. Přejmenujeme prvky X tak, že $X = \{0, 1, \ldots, n-1\}$ a $\mathcal{S} = \{S_1, S_2, \ldots, S_r\}$.

Zvolíme přirozené číslo p větší než r (počet prvků $\mathcal{S}$). Každé podmnožině S_i přiřadíme kladné číslo a_i takto: Ke každé množině S_i označíme χ_{S_i} její charakteristickou funkci; tj. $\chi_{S_i}(j) = 1$ právě tehdy, když $j \in S_i$. Pak

$$S_i \longrightarrow \sum_{j=0}^{n-1} \chi_{S_i}(j)\, p^j = a_i.$$

Nakonec zvolíme číslo $K = \sum_{i=0}^{n-1} p^i$.

Protože $p > r$, není těžké ukázat, že

$$\sum_{i \in J} a_i = K \text{ právě tehdy, když } \mathcal{A} = \{S_i \mid i \in J\} \text{ je rozklad } X.$$

4.3.21 Důsledek. Protože SubsetSum je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.22 Poznámka. Nyní není těžké sestrojit polynomiální redukci problému SubsetSum na problém dělení kořisti nebo na problém batohu. Proto jsou i tyto dvě úlohy $\mathcal{NP}$ úplné.

4.3.23 Problém klik.
Úloha: Je dán prostý neorientovaný graf $G = (V, E)$ bez smyček a číslo k.
Otázka: Existuje v grafu G klika o alespoň k vrcholech?

4.3.24 Tvrzení. Platí

$$3-CNF\ SAT \lhd_p \text{problém klik}.$$

4.3.25 Nástin převodu $3-CNF\,SAT$ na problém klik. Je dána formule φ v CNF, s k klauzulemi $C_1, C_2, \ldots, C_k$, kde každá klauzule má 3 literály. Sestrojíme k-partitní neorientovaný graf $G = (V, E)$ takto:

G má pro každou klauzuli jednu stranu; strana odpovídající klauzuli C se skládá ze 3 vrcholů označených literály klauzule C. Hrany grafu G vedou vždy mezi dvěma stranami a to tak, že spojují dva literály, které nejsou komplementární (tj. jeden není negací druhého).

Platí: Formule φ je splnitelná právě tehdy, když v grafu G existuje klika o k vrcholech. (Poznamenejme, že k je počet klauzulí formule φ.)

Jestliže φ je pravdivá v ohodnocení u, vybereme v každé klauzuli formule φ jeden literál, který je v daném ohodnocení pravdivý. Pak množina vrcholů odpovídajících těmto literálům tvoří kliku v G o k vrcholech.

Jestliže v grafu G existuje klika A o k vrcholech, pak A má jeden vrchol v každé straně grafu G. Položme jako pravdivé všechny literály, které se nacházejí v A a hodnoty ostatních logických proměnných zadefinujme libovolně. Pak v tomto ohodnocení je formule φ pravdivá.

Zkonstruovaný graf G má tolik vrcholů jako má formule φ literálů, tj. n vrcholů, kde n je délka formule φ. Vzhledem k tomu, že prostý graf s n vrcholy má $\mathcal{O}(n^2)$ hran, jedná se o polynomiální redukci.

4.3.26 Důsledek. Protože problém klik je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.27 Nezávislé množiny. Je dán prostý neorientovaný graf $G = (V, E)$ bez smyček. Množina vrcholů $N \subseteq V$ se nazývá *nezávislá množina* v G, jestliže žádná hrana grafu G nemá oba krajní vrcholy v N. Jinými slovy, indukovaný podgraf množinou N je diskrétní graf.

Úloha: Je dán prostý neorientovaný graf G bez smyček a číslo k.
Otázka: Existuje v G nezávislá množina o k vrcholech?

4.3.28 Tvrzení. Platí

$$\text{problém klik} \lhd_p \text{nezávislé množiny.}$$

4.3.29 Převod problému klik na nezávislé množiny. Je dán prostý neorientovaný graf bez smyček $G = (V, E)$. Definujeme opačný graf $G^{op} = (V, E^{op})$ takto:

$$\{u, v\} \in E^{op} \text{ právě tehdy, když } u \neq v \text{ a } \{u, v\} \notin E.$$

Platí: Množina $A \subseteq V$ je klika v grafu G právě tehdy, když je maximální nezávislou množinou v grafu G^{op}. (Jinými slovy, A je nezávislá množina a přidáním libovolného vrcholu už nebude nezávislá.)

To, že se jedná o polynomiální redukci vyplývá z faktu, že všech hran v grafu G i doplňkovém grafu G^{op} je $\frac{n(n-1)}{2}$, kde n je počet vrcholů.

4.3.30 Důsledek. Protože úloha o nezávislých množinách je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.31 Vrcholové pokrytí. Je dán prostý neorientovaný graf bez smyček $G = (V, E)$. Podmnožina vrcholů $B \subseteq V$ se nazývá *vrcholové pokrytí* G, jestliže každá hrana grafu G má alespoň jeden krajní vrchol v množině B.

Poznamenejme, že celá množina vrcholů V je vrcholovým pokrytím, problém je najít vrcholové pokrytí o co nejmenším počtu vrcholů.

Úloha: Je dán prostý neorientovaný graf G bez smyček a číslo k.
Otázka: Existuje v grafu G vrcholové pokrytí o k vrcholech?

4.3.32 Tvrzení. Platí

$$\text{nezávislé množiny} \lhd_p \text{vrcholové pokrytí.}$$

4.3.33 Nástin převodu nezávislých množin na vrcholové pokrytí. Platí: Je-li množina N nezávislá množina grafu G, pak množina $V \setminus N$ je vrcholovým pokrytím grafu G. A naopak, je-li B vrcholové pokrytí grafu G, pak množina $V \setminus B$ je nezávislá množina v G.

Proto: Je dán prostý neorientovaný graf G bez smyček a číslo k. Pak v G existuje nezávislá množina o k vrcholech právě tehdy, když v G existuje vrcholové pokrytí o $n - k$ vrcholech, kde $n = |V|$ je počet vrcholů grafu G.

4.3.34 Důsledek. Protože problém vrcholového pokrytí je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.35 Existence hamiltonovského cyklu.
Úloha: Je dán orientovaný graf G.

Otázka: Existuje v grafu G hamiltonovský cyklus? (Jinými slovy, existuje v grafu G cyklus procházející všemi vrcholy?)

4.3.36 Tvrzení. Platí

$$\text{vrcholové pokrytí} \lhd_p \text{existence hamiltonovského cyklu.}$$

4.3.37 Základní myšlenka převodu. Převod je založen na využití speciálního grafu H o 4 vrcholech a 6 orientovaných hranách. Graf H má tuto vlastnost: Má-li být graf součástí hamiltonovského cyklu, pak jsou jen dva základní způsoby průchodu grafem H, buď se projdou všechny vrcholy za sebou, nebo při dvojím průchodu vždy dva a dva.

Předpokládejme, že je dán neorientovaný prostý graf $G = (V, E)$ bez smyček a číslo k. Je možno vytvořit orientovaný graf G' takový, že v G existuje vrcholové pokrytí o k vrcholech právě tehdy, když v G' existuje hamiltonovský cyklus.

Graf G' se, zhruba řečeno, vytvoří takto: Za každou hranu grafu G do G' dáme kopii grafu H. Kromě takto získaných vrcholů přidáme ještě vrcholy $1, 2, \ldots, k$. Celkově tedy počet vrcholů grafu G' je $4\,|E| + k$. Hrany grafu G' jsou jednak hrany všech kopií grafu H, jednak hrany vedoucí mezi nimi a dále hrany do a z vrcholů $1, 2, \ldots, k$. Celkově je hran grafu G' také úměrně počtu hran grafu G plus k-násobek počtu vrcholů grafu G. To znamená, že redukce je polynomiální.

4.3.38 Důsledek. Protože problém existence hamiltonovského cyklu je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.39 Hamiltonovská kružnice
Podobně jako jsme v 4.3.36 ukázali, že se problém vrcholového pokrytí polynomiálně redukuje na problém existence hamiltonovského cyklu, dá se sestrojit i redukce

$$\text{vrcholové pokrytí} \lhd_p \text{existence hamiltonovského kružnice.}$$

Rozdíl je pouze v tom, že pomocný graf je složitější.

4.3.40 Důsledek. Protože problém existence hamiltonovské kružnice je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.41 Tvrzení. Platí

existence hamiltonovské kružnice $\lhd_p$ problém obchodního cestujícího.

Převod zmíněný v tvrzení je velmi jednoduchý a je ponechán studentům jako domácí úkol.

4.3.42 Důsledek. Protože problém obchodního cestujícího je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.43 Tvrzení. Platí

existence hamiltonovského cyklu $\lhd_p$ existence orientované hamiltonovské cesty.

Převod zmíněný v tvrzení je jednoduchý a ukážeme si jej na cvičení.

4.3.44 Důsledek. Protože problém existence hamiltonovské cesty je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.45 Tvrzení. Platí

existence orientované hamiltonovské cesty $\lhd_p$ nejdelší cesty v orientovaném grafu.

Převod zmíněný v tvrzení je velmi jednoduchý.

4.3.46 Důsledek. Protože problém nejdelších cest v orientovaném grafu je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.47 Tvrzení. Platí

nejdelší cesty v orientovaném grafu $\lhd_p$ nejkratší cesty v orientovaném grafu.

Převod zmíněný v tvrzení je velmi jednoduchý.

4.3.48 Důsledek. Protože problém nejkratších cest v orientovaném grafu je ve třídě $\mathcal{NP}$, jedná se o $\mathcal{NP}$ úplnou úlohu.

4.3.49 Heuristiky. Jestliže je třeba řešit problém, který je $\mathcal{NP}$ úplný, musíme pro větší instance opustit myšlenku přesného nebo optimálního řešení a smířit se s tím, že získáme „dostatečně přesné“ nebo „dostatečně kvalitní“ řešení. K tomu se používají heuristické algoritmy pracující v polynomiálním čase. Algoritmům, kde umíme zaručit „jak daleko“ je nalezené řešení od optimálního, se také říká aproximační algoritmy.

4.3.50 Heuristika pro vrcholové pokrytí — 1. Uvažujme následující heuristický algoritmus, který pro daný neorientovaný graf najde jeho vrcholové pokrytí. Algoritmus je založen na „hladovém postupu“.

Vstup: neorientovaný graf $G = (V, E)$.

Výstup: vrcholové pokryti C grafu G.

```
begin
    C := ∅
        while E ≠ ∅ do
            vyber vrchol v s největším stupněm
            C := C ∪ {v}
            odstraň v spolu s hranami s ním incidentními
end
return C
```

Přestože algoritmus „vypadá rozumně“, v některých případech najde vrcholové pokrytí, které má podstatně víc vrcholů než nejméně početné pokrytí. Zde tím „podstatně“ rozumíme toto: existuje graf G, který má vrcholové pokrytí o k vrcholech, ale výše uvedený algoritmus najde vrcholové pokrytí o $\Theta(k \lg k)$ vrcholech.

4.3.51 Heuristika pro vrcholové pokrytí — 2. Uvažujme ještě jeden heuristický algoritmus, který pro daný neorientovaný graf najde jeho vrcholové pokrytí.

Vstup: neorientovaný graf $G = (V, E)$.

Výstup: vrcholové pokryti C grafu G.

```
begin
    C := ∅
        while E ≠ ∅ do
            vyber hranu {u, v}
            C := C ∪ {u, v}
            odstraň vrcholy u, v spolu se všemi hranami s nimi incidentními
end
return C
```

Dá se dokázat, že druhý algoritmus vždy najde vrcholové pokrytí, které obsahuje maximálně dvakrát tolik vrcholů než je počet nejméně početného vrcholového pokrytí.

4.3.52 Tvrzení. Označme C_{min} nejméně početné vrcholové pokrytí grafu G. Pak druhá heuristika najde vrcholové pokrytí C takové, že

$$|C| \leq 2\,|C_{min}|.$$

Důkaz. Označme F množinu všech hran, která byla vybrána algoritmem. Pak $|C| = 2\,|F|$; ano, za každou vybranou hranu jsme do množiny C vložili dva vrcholy — krajní vrcholy této hrany. Navíc, žádné dvě hrany v množině F nemají společný vrchol; tudíž je jejich pokrytí je třeba $|F|$ vrcholů. Proto $|C_{min}| \geq |F|$ a $|C| = 2\,|F| \leq 2\,|C_{min}|$.

4.3.53 Aproximační algoritmus. Definice. Uvažujme optimalizační problém $\mathcal{U}$. Polynomiální algoritmus $\mathcal{A}$ se nazývá R *aproximační algoritmus*, jestliže existuje reálné číslo R takové, že pro každou instanci algoritmus $\mathcal{A}$ najde přípustné řešení ne horší než R krát hodnota optimálního řešení.

To znamená, že pro minimalizační úlohu najde řešení, které nemá hodnotu účelové funkce větší než R krát hodnotu optimálního řešení; pro maximalizační úlohu najde řešení, které nemá hodnotu účelové funkce menší než R krát hodnotu optimálního řešení.

Druhá heuristika pro nalezení vrcholového pokrytí je tedy 2 aproximační algoritmus pro problém vrcholového pokrytí.

Ne pro všechny úlohy, jejichž rozhodovací verze jsou NP úplné, aproximační algoritmy existují. Příkladem je problém obchodního cestujícího, jak ukazuje následující věta.

4.3.54 Tvrzení. Kdyby existovala konstanta r a polynomiální algoritmus $\mathcal{A}$ takový, že pro každou instanci obchodního cestujícího I najde trasu délky $D \leq r\,OPT(I)$, kde $OPT(I)$ je délka optimální trasy instance I, pak

$$\mathcal{P} = \mathcal{NP}.$$

4.3.55 Zdůvodnění tvrzení 4.3.54. Za předpokladu tvrzení 4.3.54 bychom uměli polynomiálně vyřešit problém existence hamiltonovské kružnice. Naznačíme odpovídající převod.

Je dán neorientovaný graf $G = (V, E)$, $V = \{1, 2, \ldots, n\}$, a ptáme se, zda v něm existuje hamiltonovská kružnice. Zkonstruujeme instanci obchodního cestujícího takto: Pro města $\{1, 2, \ldots, n\}$ položíme

$$d(i,j) = \begin{cases} 1, & \{i,j\} \in E \\ r\,n + 1, & \{i,j\} \notin E \end{cases}$$

Trasa v instanci popsané výše může mít délku n, jestliže je tvořena všemi hranami délky 1. V tomto případě jsou všechny hrany hranami grafu G a trasa představuje hamiltonovskou kružnici. Nebo musí trasa mít délku alespoň $n - 1 + n\,r + 1 = n\,r + n$. To je v případě, že aspoň jedna spojnice v trase není tvořena hranou grafu G.

Tedy jestliže algoritmus $\mathcal{A}$ najde trasu délky jiné než n, pak v grafu G neexistuje hamiltonovská kružnice. Takto bychom polynomiálním algoritmem byli schopni rozhodnout existenci hamiltonovské kružnice. Protože existence hamiltonovské kružnice je $\mathcal{NP}$ úplný problém, platilo by $\mathcal{P} = \mathcal{NP}$.

4.3.56 Trojúhelníková nerovnost. Řekneme, že instance obchodního cestujícího splňuje trojúhelníkovou nerovnost, jestliže pro každá tři města i, j, k platí:

$$d(i,j) \leq d(i,k) + d(k,j).$$

4.3.57 Tvrzení. Jestliže instance I obchodního cestujícího splňuje trojúhelníkovou nerovnost, pak existuje polynomiální algoritmus $\mathcal{A}$, který pro I najde trasu délky D, kde $D \leq 2\,OPT(I)$ ($OPT(I)$ je délka optimální trasy v I).

4.3.58 Slovní popis algoritmu z tvrzení 4.3.57. Instanci I považujeme za úplný graf G s množinou vrcholů $V = \{1, 2, \ldots, n\}$ a ohodnocením d.

1. V grafu G najdeme minimální kostru (V, K).

2. Kostru (V, K) prohledáme do hloubky z libovolného vrcholu.

3. Trasu T vytvoříme tak, že vrcholy procházíme ve stejném pořadí jako při prvním navštívení během prohledávání grafu. T je výstup em algoritmu.

Zřejmě platí, že délka kostry K je menší než $OPT(I)$. Ano, vynecháme-li z optimální trasy některou hranu, dostaneme kostru grafu G. Protože K je minimální kostra, musí být délka K menší než $OPT(I)$ (předpokládáme, že vzdálenosti měst jsou kladné). Vzhledem k platnosti trojúhelníkové nerovnosti, je délka T menší nebo rovna dvojnásobku délky kostry K.

4.3.59 Christofidesův algoritmus. Jestliže instance I obchodního cestujícího splňuje trojúhelníkovou nerovnost, pak následující algoritmus najde trasu T délky D takovou, že $D \leq \frac{3}{2} OPT(I)$.

Instanci I považujeme ze úplný graf G s množinou vrcholů $V = \{1, 2, \ldots, n\}$ a ohodnocením d.

1. V grafu G najdeme minimální kostru (V, K).

2. Vytvoříme úplný graf H na množině všech vrcholů, které v kostře (V, K) mají lichý stupeň.

3. V grafu H najdeme nejlevnější perfektní párování P.

4. Hrany P přidáme k hranám K minimální kostry. Graf $(V, P \cup K)$ je eulerovský graf. V grafu $(V, P \cup K)$ sestrojíme uzavřený eulerovský tah.

5. Trasu T získáme z eulerovského tahu tak, že vrcholy navštívíme v pořadí, ve kterém jsme do nich poprvé vstoupili při tvorbě eulerovského tahu.

Platí, že délka takto vzniklé trasy je maximálně $\frac{3}{2}$ krát větší než délka optimální trasy.

4.3.60 Poznámka. Odhad délky trasy, kterou jsme získali v 4.3.58, i odhad pro trasu získanou Christofidesovým algoritmem není možné zlepšit.

4.4 Třída co-$\mathcal{NP}$

4.4.1 Pozorování. Je-li jazyk L ve třídě $\mathcal{P}$, pak i jeho doplněk $\overline{L}$ patří do třídy $\mathcal{P}$. Obdobné tvrzení se pro jazyky třídy $\mathcal{NP}$ neumí dokázat.

4.4.2 Definice. Jazyk L patří do třídy co-$\mathcal{NP}$, jestliže jeho doplněk patří do třídy $\mathcal{NP}$.

4.4.3 Příklady.

- Jazyk $USAT$, který je doplňkem jazyka SAT splnitelných booleovských formulí, leží ve třídě co-$\mathcal{NP}$. (Jazyk $USAT$ se skládá ze všech nesplnitelných booleovských formulí a ze všech slov, které neodpovídají booleovské formuli.)

- Jazyk $TAUT$, který se skládá ze všech slov odpovídajících tautologii výrokové logiky, patří do třídy co-$\mathcal{NP}$.

4.4.4 Otázka, zda co-$\mathcal{NP} = \mathcal{NP}$, je otevřená.

4.4.5 Lemma. Mějme dva jazyky L_1 a L_2, pro které platí $L_1 \lhd_p L_2$. Pak platí také $\overline{L_1} \lhd_p \overline{L_2}$, (kde $\overline{L}$ je doplněk jazyka L).

Zdůvodnění. Jestliže $L_1 \lhd_p L_2$, $L_1 \subseteq \Sigma^\star$, $L_2 \subseteq \Phi^\star$, pak existuje polynomiální algoritmus M, který pro každé slovo $w \in \Sigma^\star$ zkonstruuje slovo $M(w) \in \Phi^\star$ a to tak, že

$$w \in L_1 \quad \text{právě tehdy, když} \quad M(w) \in L_2.$$

To ale znamená, že

$$w \notin L_1 \quad \text{právě tehdy, když} \quad M(w) \notin L_2,$$

a tedy $\overline{L_1} \lhd_p \overline{L_2}$.

4.4.6 Tvrzení. Platí co-$\mathcal{NP} = \mathcal{NP}$ právě tehdy, když existuje $\mathcal{NP}$ úplný jazyk, jehož doplněk je ve třídě $\mathcal{NP}$.

Důkaz. Jestliže co-$\mathcal{NP} = \mathcal{NP}$, pak každý doplněk nějakého $\mathcal{NP}$ úplného jazyka leží ve třídě $\mathcal{NP}$.

Předpokládejme, že existuje $\mathcal{NP}$ úplný jazuk L, jehož doplněk $\overline{L}$ leží ve třídě $\mathcal{NP}$. Ukážeme, že
co-$\mathcal{NP} \subseteq \mathcal{NP}$ a $\mathcal{NP} \subseteq$ co-$\mathcal{NP}$.

Vezměme libovolný jazyk L_1 ze třídy co-$\mathcal{NP}$. Pak $\overline{L_1} \in \mathcal{NP}$. Protože L je $\mathcal{NP}$ úplný jazyk, $\overline{L_1} \lhd_p L$ a podle předchozího lemmatu $L_1 \lhd_p \overline{L} \in \mathcal{NP}$. Odtud $L_1 \in \mathcal{NP}$.

Vezměme libovolný jazyk L_2 takový, že $L_2 \in \mathcal{NP}$. Pak $L_2 \lhd_p L$ a tedy (opět podle předchozího lemmatu) $\overline{L_2} \lhd_p \overline{L} \in \mathcal{NP}$. Proto $L_2 \in$ co-$\mathcal{NP}$.

4.5 Třídy $\mathcal{P}$SPACE a $\mathcal{NP}$SPACE

4.5.1 Je dán Turingův stroj M (deterministický nebo nedeterministický). Připomeňme, že M pracuje s paměťovou složitostí $p(n)$ právě tehdy, když pro každé slovo délky n nepoužije paměťovou buňku větší než $p(n)$. Uvědomte si: jestliže Turingův stroj přijímá jazyk s polynomiální časovou složitostí, pak se musí zastavit na **každém** vstupu (ať už leží v přijímané jazyce, nebo ne); tj. Turingův stroj jazyk rozhoduje. Podobné tvrzení neplatí pro Turingův stroj, který nějaký jazyk přijímá s polynomiální paměťovou složitostí; ano, Turingův stroj se může zacyklit na slově, které nepřijímá.

4.5.2 Třída $\mathcal{P}$SPACE. Jazyk L patří do třídy $\mathcal{P}$SPACE jestliže existuje deterministický Turingův stroj M, který přijímá jazyk L a pracuje s polynomiální paměťovou složitostí.

4.5.3 Tvrzení. Platí

$$\mathcal{P} \subseteq \mathcal{P}\text{SPACE}.$$

4.5.4 Třída $\mathcal{NP}$SPACE. Jazyk L patří do třídy $\mathcal{NP}$SPACE jestliže existuje nedeterministický Turingův stroj M, který přijímá jazyk L a pracuje s polynomiální paměťovou složitostí.

4.5.5 Tvrzení. Platí

$$\mathcal{NP} \subseteq \mathcal{NP}\text{SPACE}.$$

4.5.6 Věta. Je dán Turingův stroj M (deterministický nebo nedeterministický), který přijímá jazyk L s paměťovou složitostí $p(n)$ (kde p je nějaký polynom). Pak existuje konstanta c taková, že M přijme slovo w délky n po nejvýše $c^{p(n)+1}$ krocích.

4.5.7 Myšlenka důkazu věty 4.5.6. Konstantu c volíme tak, abychom měli zajištěno, že Turingův stroj M má při práci se vstupem délky n méně než $c^{p(n)+1}$ různých situací. Zajímají nás totiž pouze takové výpočty, ve kterých se situace neopakují. Označme t počet páskových symbolů Turingova stroje M a označme s počet stavů M. Pak M má $p(n)\, s\, t^{p(n)}$ různých situací; ano, stroj se může nacházet v s různých stavech, hlava může skenovat jedno z $p(n)$ polí pásky, a páska může mít jeden z $t^{p(n)}$ různých obsahů.

Položme $c = t + s$. Z binomické věty vyplývá, že

$$c^{p(n)+1} = (t+s)^{p(n)+1} = t^{p(n)+1} + (p(n)+1)\, t^{p(n)}\, s + \dots.$$

Odtud $c^{p(n)+1} \geq p(n)\, t^{p(n)}\, s$.

4.5.8 Věta. Je-li jazyk L ve třídě $\mathcal{P}$SPACE ($\mathcal{NP}$SPACE), pak L je rozhodován deterministickým (nedeterministickým) Turingovým strojem M s polynomiální paměťovou složitostí, který se vždy zastaví po nejvýše $c^{q(n)}$ krocích, kde $q(n)$ je vhodný polynom a c konstanta.

4.5.9 Myšlenka důkazy věty 4.5.8. Předpokládejme, že $L \in \mathcal{P}$SPACE. Pak existuje Turingův stroj M_1, který přijímá jazyk L s paměťovou složitostí $p(n)$ ($p(n)$ je vhodný polynom). Víme (z věty 4.5.6), že existuje konstanta c taková, že Turingův stroj M_1 potřebuje nejvýše $c^{p(n)+1}$ kroků.

Vytvoříme Turingův stroj M_2, který bude mít dvě pásky: první páska bude simulovat M_1, druhá bude počítat kroky na první pásce. Jestliže počet kroků překročí $c^{p(n)+1}$, Turingův stroj M_2 se neúspěsně zastaví. Počítání kroků M_2 provádí v c-adické soustavě (tak, aby zabralo jen $\mathcal{O}(p(n))$ polí pásky).

Hledaný Turingův stroj M je Turingův stroj s jednou páskou, který simuluje Turingův stroj M_2. Turingův stroj M pracuje v s časovou složitostí $\mathcal{O}(c^{2p(n)})$, tedy v maximálně $d\, c^{2p(n)}$ krocích. Nyní stačí položit $q(n) = 2p(n) + \log_c d$ nebo jakýkoli polynom větší.

4.5.10 Savitchova věta. Platí

$$\mathcal{P}\texttt{SPACE} = \mathcal{NP}\texttt{SPACE}.$$

4.5.11 Nástin myšlenky důkazu Savitchovy věty. Zřejmě $\mathcal{P}\texttt{SPACE} \subseteq \mathcal{NP}\texttt{SPACE}$. Důkaz opačné inkluze $\mathcal{NP}\texttt{SPACE} \subseteq \mathcal{P}\texttt{SPACE}$ spočívá v tom, že jsme schopni nedeterministický Turingův stroj M pracující s paměťovou složitostí $p(n)$ simulovat deterministickým Turingovým strojem N, který pracuje s paměťovou složitostí $\mathcal{O}([p(n)]^2)$ (o časové složitosti nic dokazovat nebudeme). Konstrukce N je založena na následující rekursivní proceduře $DOSTP(I, J; m)$, kde I a J jsou situace NTM M a m je kladné přirozené číslo. $DOSTUP(I, J; m)$ vrátí 1, jestliže $I \vdash^\star J$ v nejvýše m krocích.

$DOSTUP(I, J; m)$

Vstup: Situace I a J NTM M a m je kladné přirozené číslo.

Výstup: TRUE, jestliže J je dostupná z I v nejvýše m krocích, FALSE v opačném případě.

```
begin
   if m = 1 then
         if I = J nebo I ⊢ J then return TRUE
         else return FALSE
   end
   else (induktivní část]
         for každou možnou situaci K do
         if DOSTUP(I, K; ⌊m/2⌋) a DOSTUP(K, J; ⌈m/2⌉ then
               return TRUE
         return FALSE
   end
end
```

Uvědomte si, že rekurzivní procedura $DOSTUP(I, J; m)$ má vždy na zásobníku jen jednu trojici $(I_1, J_1; m)$, nejvýše jednu trojici $(I_2, J_2; \frac{m}{2})$, nejvýše jednu $(I_3, J_3; \frac{m}{4})$, atd. Tedy současně nemá na zásobníku víc než $\lg m$ různých trojic.

Je dán nedeterministický Turingův stroj M, který přijímá jazyk L s polynomiální paměťovou složitostí $p(n)$. Pro vstup w voláme proceduru $DOSTUP(I_0, J; m)$, kde I_0 je počáteční situace M, J je některá přijímající situace M a $m = c^{p(n)+1}$ (c je konstanta z 4.5.6). Dá se dokázat, že pro vykonání procedury $DOSTUP(I, J; m)$ deterministickým Turingovým strojem stačí paměťová složitost $\mathcal{O}([p(n)]^2)$. To vyplývá z toho, že $DOSTUP(I_0, J; m)$ má na zásobníku maximálně $\lg c^{p(n)+1} = d\,p(n)$ trojic $(I, J; r)$ a každá z trojic má nejvýše délku $\mathcal{O}(p(n))$. (Uvědomte si, že nám nezáleží na tom, jak dlouho deterministický Turingův stroj pracuje, zajímáme se pouze o paměťové nároky.)

4.5.12 Důsledek. Platí

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{P}\texttt{SPACE}.$$

4.6 Testování prvočíselnosti

4.6.1 Jazyky L_p a L_s. Jazyk L_p obsahuje všechna prvočísla, jazyk L_s obsahuje všechna složená čísla; přesněji:

$$L_p = \{w \mid w \text{ je binární zápis prvočísla}\}$$

$$L_s = \{w \mid w \text{ je binární zápis složeného čísla}\}.$$

Jazyk L_s je (až na číslo 1) doplňkem jazyka L_p; přidáme-li 1 do jazyka L_s, pak dostáváme

$$L_s = \overline{L_p}, \quad L_p = \overline{L_s}.$$

4.6.2 Tvrzení. Jazyk L_s leží ve třídě $\mathcal{NP}$.

Zdůvodněni: Jestliže číslo n je složené, znamená to, že má dělitele r, pro nějž platí $1 < r < n$. Známe-li některého (tzv. vlastního) dělitele r, jsme schopni dělením čísla n číslem r zjistit, že n je opravdu složené číslo. Pro prvočíslo žádný takový vlastní dělitel neexistuje.

Nyní si stačí uvědomit, že vlastní dělitel je hledaný certifikát s polynomiální velkostí. Ano, délka binárního slova odpovídajícího n, je $k = \lg n$, délka dělitele r je $\mathcal{O}(k)$ a celočíselné dělení dvou binárních čísel délky k lze provést v polynomiálním čase vzhledem k délce binárního zápisu čísel.

4.6.3 Důsledek. Jazyk L_p je ve třídě co-$\mathcal{NP}$.

4.6.4 Tvrzení. Jazyk L_p je ve třídě $\mathcal{NP}$.

Najít polynomiální certifikát pro jazyk obsahující prvočísla je podstatně těžší než pro jazyk obsahující složená čísla. V tomto případě se jedná např. o generátor grupy $(\mathbb{Z}_p \setminus \{0\}, \odot, 1)$ (p prvočíslo); tj primitivní prvek konečného tělesa $(\mathbb{Z}_p, \oplus, \odot, 0, 1)$.

4.6.5 Důsledek. Jazyky L_p a L_s patří do průniku tříd $\mathcal{NP}$ a co-$\mathcal{NP}$.

4.6.6 V dalším ukážeme, že existuje pravděpodobnostní algoritmus — Millerův test prvočíselnosti, který pro dané velké liché číslo n s pravděpodobností aspoň $\frac{1}{2}$ rozhodne, zda n je prvočíslo. Dříve než algoritmus uvedeme, připomeneme několik faktů z algebry, které budeme potřebovat.

- Množina $\mathbb{Z}_n$ tzv. zbytkových tříd modulo n je

  $$\mathbb{Z}_n = \{0, 1, \ldots, n-1\}.$$

- Na množině $\mathbb{Z}_n$ jsou definovány operace $\oplus$ a $\odot$ takto

  $$a \oplus b = c, \quad \text{kde } c \text{ je zbytek při dělení čísla } a + b \text{ číslem } n,$$

  $$a \odot b = c, \quad \text{kde } c \text{ je zbytek při dělení čísla } a.b \text{ číslem } n.$$

- $(\mathbb{Z}_n, \oplus, 0)$ je komutativní grupa, $(\mathbb{Z}_n, \odot, 1)$ je komutativní monoid a platí distributivní zákony

  Navíc, prvek $a \in \mathbb{Z}_n$ má inverzní prvek (vzhledem k operaci $\odot$) právě tehdy, když a a n jsou nesoudělná čísla.

  Proto $(\mathbb{Z}_n, \oplus, \odot, 0, 1)$ pro n prvočíslo je těleso; pro složená n, tělesem není.

- Podle malé Fermatovy věty pro a nesoudělné s prvočíslem p platí

$$a^{p-1} \equiv 1 \,(\mathrm{mod}\, n).$$

- Je-li H podgrupa konečné grupy G, pak počet prvků podgrupy H dělí počet prvků grupy G.
- Operace sčítání, násobení, umocňování a dělení v $\mathbb{Z}_n$ je možné provést v polynomiálním čase vzhledem k velikosti čísel, se kterými se operace provádějí.

4.6.7 Millerův test prvočíselnosti.

Vstup: velké liché přirozené číslo n.

Výstup: „prvočíslo“ nebo „složené“.

1. Spočítáme $n - 1 = 2^l\, m$, kde m je liché číslo.
2. Náhodně vybereme $a \in \{1, 2, \ldots, n-1\}$.
3. Spočítáme $a^m \,(\mathrm{mod}\, n)$,
   jestliže $a^m \equiv 1 \,(\mathrm{mod}\, n)$, stop, výstup „prvočíslo“.
4. Opakovaným umocňováním počítáme
   $a^{2\,m} \,(\mathrm{mod}\, n), a^{2^2\,m} \,(\mathrm{mod}\, n), \ldots, a^{2^l\,m} \,(\mathrm{mod}\, n)$.
5. Jestliže $a^{2^l\,m} \not\equiv 1 \,(\mathrm{mod}\, n)$, stop, výstup „složené“.
6. Vezmeme k takové, že $a^{2^k\,m} \not\equiv 1 \,(\mathrm{mod}\, n)$ a $a^{2^{k+1}\,m} \equiv 1 \,(\mathrm{mod}\, n)$.
   Jestliže $a^{2^k\,m} \equiv -1 \,(\mathrm{mod}\, n)$, stop, výstup „prvočíslo“.
   Jestliže $a^{2^k\,m} \not\equiv -1 \,(\mathrm{mod}\, n)$, stop, výstup „složené“.

4.6.8 Věta.

1. Jestliže pro vstup n dá Millerův test prvočíselnosti odpověď „složené“, pak je číslo n složené.
2. Jestliže pro vstup n dá Millerův test prvočíselnosti odpověď „prvočíslo“, pak n je prvočíslo s pravděpodobností větší než $\frac{1}{2}$.

Idea důkazu. Add 1. Jestliže je číslo n prvočíslo, tak nemůžeme dostat výstup „složené“. Malá Fermatova věta totiž zaručuje, že nemůžeme skončit v kroku 5 s výstupem „složené“. Dále pro n prvočíslo je $(\mathbb{Z}_n, \oplus, \odot)$ konečné těleso. V tělese existují pouze dva prvky, které umocněné na druhou dávají 1 (tzv. odmocniny z 1) — totiž číslo 1 a -1. Proto nemůžeme skončit ani v kroku 6 výstupem „složené“.

Add 2. Ukázat druhou vlastnost je obtížnější. Důkaz není těžký pro taková složená n, pro která existuje $a \in \mathbb{Z}_n$, a nesoudělné s n, a $a^{n-1} \not\equiv 1 \,(\mathrm{mod}\, n)$.

Pro ostatní složená čísla, tzv. „pseudoprvočísla“, (též „Carmichaelova čísla“), je důkaz dost obtížný.

Ukážeme základní myšlenku důkazu pro složená n: Spočítáme počet takových a vybraných v kroku 2, pro která dostaneme jistě správnou odpověď (tj. nedostaneme odpověď prvočíslo). Protože každé a má stejnou pravděpodobnost být vybráno, stačí, abychom ukázali, že jich je aspoň tolik, kolik jich může dát odpověď špatnou (prvočíslo).

Vybereme-li v kroku 2 neinvertibilní číslo a, určitě dostaneme odpověď složené, protože žádná mocnina neinvertibilního čísla nemůže být rovna 1.

Předpokládejme, že složené číslo n není pseudoprvočíslo, tj. existuje $a \in \mathbb{Z}_n$, a nesoudělné s n, a $a^{n-1} \not\equiv 1 \,(\mathrm{mod}\, n)$. Označme

$$\mathbb{Z}_n^\star = \{a \in \mathbb{Z}_n \mid a \text{ je invertibilní}\}$$

$$K = \{a \in \mathbb{Z}_n \mid a^{n-1} = 1\}.$$

Víme, že $K \neq \mathbb{Z}_n^\star$, přitom $(K, \odot)$ je podgrupa grupy $(\mathbb{Z}_n^\star, \odot)$. Proto počet prvků K dělí počet prvků $\mathbb{Z}_n^\star$. Odtud počet prvků v množině K je nejvýše dvakrát méně než prvků v množině $\mathbb{Z}_n^\star$; jinými slovy

$$|\mathbb{Z}_n^\star \setminus K| \geq |K|.$$

Vybereme-li $a \in \mathbb{Z}_n^\star \setminus K$, dostaneme správnou odpověď „složené“, protože $a^{n-1} \neq 1$.

Špatnou odpověď můžeme dostat pouze pro $a \in K$ a těch je méně než nebo stejně jako $a \in \mathbb{Z}_n^\star \setminus K$.

Pro pseudoprvočísla platí $|K| = |\mathbb{Z}_n^\star|$ a musíme argumentovat krokem 6, kde se dá ukázat, že počet a, která vedou v kroku 6 na odmocninu z 1 různou od -1 je aspoň tak velký jako počet těch a, která vedou na -1.

4.7 Třídy založené na pravděpodobnostních algoritmech

4.7.1 Randomizovaný Turingův stroj. RTM je, zhruba řečeno, Turingův stroj M se dvěma nebo více páskami, kde první páska má stejnou roli jako u deterministického Turingova stroje, ale druhá páska obsahuje náhodnou posloupnost 0 a 1, tj. na každém políčku se 0 objeví s pravděpodobností $\frac{1}{2}$ a 1 také s pravděpodobností $\frac{1}{2}$.

Na začátku práce:

- stroj M se nachází v počátečním stavu q_0;
- první páska obsahuje vstupní slovo w, zbytek pásky pak blanky B;
- druhá páska obsahuje náhodnou posloupnost 0 a 1;
- případné další pásky obsahují B;
- všechny hlavy jsou nastaveny na prvním políčku dané pásky.

Na základě stavu q, ve kterém se stroj M nachází, a na základě obsahu políček, které jednotlivé hlavy čtou, přechodová funkce δ určuje, zda se M zastaví nebo přejde do nového stavu p, přepíše obsah první pásky (**nikoli ale obsah druhé pásky**) a hlavy posune doprava, doleva nebo zůstanou stát (posuny hlav jsou nezávislé).

Formálně, je-li M ve stavu q, hlava na první pásce čte symbol X, na druhé pásce je číslo a a

$$\delta(q, X, a) = (p, Y, D_1, D_2), \quad q, p \in Q, a \in \{0, 1\}, X, Y \in \Gamma, D_1, D_2 \in \{L, R, S\},$$

pak M se přesune do stavu p, na první pásku napíše Y a i-tá hlava se posune doprava pro $D_i = R$, doleva pro $D_i = L$ nebo zůstane na místě pro $D_i = S$.

Jestliže $\delta(q, X, a)$ není definováno, M se zastaví.

M se úspěšně zastaví právě tehdy, když se přesune do koncového (přijímacího) stavu q_f.

4.7.2 Poznámka. Rozdíl mezi RTM a obyčejným TM je v roli druhé pásky. Turingův stroj s dvěma páskami může přepisovat i obsah druhé pásky a to je v případě RTM zakázáno. Navíc při dvou běžích RTM může být průběh práce RTM různý (záleží na náhodně vygenerovaném obsahu druhé pásky). To se u vícepáskového deterministického TM stát nemůže.

Může se zdát, že tento model je nerealistický — nemůžeme před začátkem práce naplnit nekonečnou pásku. Toto je ale „realizováno“ tak, že v okamžiku, kdy druhá hlava čte dosud nenavštívené políčko druhé pásky, náhodně se vygeneruje 0 nebo 1 každé s pravděpodobností $\frac{1}{2}$ a tento symbol už se nikdy během jednoho průběhu práce TM nezmění.

4.7.3 Příklad. Je dán RTM M, kde $Q = \{q_0, q_1, q_2, q_3, q_f\}$, $\Gamma = \{0, 1, B\}$ a přechodová funkce δ je definována tabulkou:

		$0,0$	$1,0$	$0,1$	$1,1$	$B,0$	$B,1$
$\rightarrow$	q_0	$(q_1,0,R,S)$	$(q_2,1,R,S)$	$(q_3,0,S,R)$	$(q_3,1,S,R)$	–	–
	q_1	$(q_1,0,R,S)$	–	–	–	(q_4,B,S,S)	–
	q_2	–	$(q_2,1,R,S)$	–	–	(q_4,B,S,S)	–
	q_3	$(q_3,0,R,R)$	–	–	$(q_3,1,R,R)$	(q_4,B,S,S)	(q_4,B,S,S)
$\leftarrow$	q_4	–	–	–	–	–	–

Předpokládejme, že na vstupu má RTM M slovo w, pak:

- Jestliže první symbol druhé pásky je 0 (tj. náhodně jsme vygenerovali 0), M zkontroluje, zda $w = 0^n$ nebo $w = 1^n$ pro nějaké $n > 0$.
- Jestliže první symbol druhé pásky je 1 (tj. náhodně jsme vygenerovali 1), hlava na druhé pásce se posune doprava a M zkontroluje, zda se obsah druhé pásky od druhého políčka shoduje se vstupem w.

Nenastane-li ani jeden z předchozích případů, M se neúspěšně zastaví.

V případě RTM je třeba spočítat pravděpodobnost s jakou se M pro dané vstupní slovo w úspěšně zastaví, tj. zastaví v „přijímacím“ stavu q_f. V našem příkladě je odpověď tato:

- Jestliže w je prázdné slovo, M se v q_f nikdy nezastaví (tj. pro žádný náhodný obsah druhé pásky).
- Jestliže $w = 0^n$ nebo $w = 1^n$ pro $n > 0$, M se zastaví v q_f s pravděpodobností

$$\frac{1}{2} + \frac{1}{2}\left(\frac{1}{2}\right)^n = \frac{1}{2} + 2^{-(n+1)}.$$

- Jestliže w je jiného tvaru, tj. obsahuje jak 0, tak 1, pak pravděpodobnost, že se M zastaví v q_f je

$$\frac{1}{2}\left(\frac{1}{2}\right)^{|w|} = 2^{-(|w|+1)}.$$

4.7.4 Třída $\mathcal{RP}$. Jazyk L patří do třídy $\mathcal{RP}$ právě tehdy, když existuje RTM M takový, že:

1. Jestliže $w \notin L$, stroj M se ve stavu q_f zastaví s pravděpodobností 0.
2. Jestliže $w \in L$, stroj M se ve stavu q_f zastaví s pravděpodobností, která je alespoň rovna $\frac{1}{2}$.
3. Existuje polynom $p(n)$ takový, že každý běh M (tj. pro jakýkoli obsah druhé pásky) trvá maximálně $p(n)$ kroků, kde n je délka vstupního slova.

Miller-Rabinův test prvočíselnosti je příklad algoritmu, který splňuje všechny tří podmínky (utvoříme-li k němu odpovídající RTM) a proto jazyk L, který se skládá ze všech složených čísel, patří do třídy $\mathcal{RP}$.

4.7.5 Turingův stroj typu Monte-Carlo. RTM splňující podmínky 1 a 2 z předchozí definice 4.7.4, se nazývá RTM typu *Monte-Carlo*.

Uvědomte si, že RTM typu Monte-Carlo obecně nemusí pracovat v polynomiálním čase.

4.7.6 Tvrzení. Je dán jazyk $L \in \mathcal{RP}$, pak pro každou kladnou konstantu $0 < c < \frac{1}{2}$ je možné sestrojit RTM M (algoritmus) s polynomiální složitostí a takový, že:

1. Jestliže $w \notin L$, stroj M se úspěšně zastaví (tj. zastaví se ve stavu q_f) s pravděpodobností 0.
2. Jestliže $w \in L$, stroj M se úspěšně zastaví (tj. zastaví se ve stavu q_f) s pravděpodobností aspoň $1 - c$.

4.7.7 Třída $\mathcal{ZPP}$. Jazyk L patří do třídy $\mathcal{ZPP}$ právě tehdy, když existuje RTM M takový, že:

1. Jestliže $w \notin L$, stroj M se úspěšně zastaví (tj. zastaví se ve stavu q_f) s pravděpodobností 0.
2. Jestliže $w \in L$, stroj M se úspěšně zastaví (tj. zastaví se ve stavu q_f) s pravděpodobností 1.

3. Střední hodnota počtu kroků M v jednom běhu je $p(n)$, kde $p(n)$ je polynom a n je délka vstupního slova.

To znamená: M neudělá chybu, ale nezaručujeme vždy polynomiální počet kroků při jednom běhu, pouze střední hodnota počtu kroků je polynomiální.

4.7.8 Turingův stroj typu Las-Vegas. RTM splňující podmínky z předchozí definice 4.7.7, se nazývá typu *Las-Vegas*.

4.7.9 Tvrzení. Jestliže jazyk L patří do třídy $\mathcal{ZPP}$, pak i jeho doplněk $\overline{L}$ patří do třídy $\mathcal{ZPP}$.

Stejný RTM M typu Las-Vegas slouží „k přijetí“ jak jazyka L, tak i jeho doplňku $\overline{L}$; stačí koncové (přijímající) stavy RTM M prohlásit za nekoncové a ze všech nekoncových stavů M udělat koncové.

4.7.10 Poznámka. Pro jazyky ze třídy $\mathcal{RP}$ se tvrzení obdobné 4.7.9 neumí dokázat. To motivuje následující třídu jazyků.

4.7.11 Třída co-$\mathcal{RP}$. Jazyk L patří do třídy co-$\mathcal{RP}$ právě tehdy, když jeho doplněk $\overline{L}$ patří do třídy $\mathcal{RP}$.

4.7.12 Věta.

$$\mathcal{ZPP} = \mathcal{RP} \cap \text{ co-}\mathcal{RP}.$$

Nástin důkazu. Ukážeme nejprve $\mathcal{RP} \cap$ co-$\mathcal{RP} \subseteq \mathcal{ZPP}$.

Předpokládejme, že jazyk L leží v obou třídách $\mathcal{RP}$ i co-$\mathcal{RP}$. Existují proto dva RTM M_1 a M_2 typu Monte Carlo pracující v polynomiálním čase a takové, že

M_1 — pro jazyk L;

M_2 — pro jazyk $\overline{L}$.

Označme $p(n)$ ten větší z polynomů, které určují počet kroků M_1 a M_2. Sestrojíme RTM M typu Las-Vegas pro jazyk L takto: Pro dané vstupní slovo w

1. M nechá pracovat M_1 po dobu $p(n)$ kroků. Jestliže M_1 úspěšně skončí, M také skončí úspěšně.

2. M nechá pracovat M_2 po dobu $p(n)$ kroků. Jestliže M_2 úspěšně skončí, M skončí ale neúspěšně.

3. Jestliže M neskončí ani v kroku 1 ani v kroku 2, M pokračuje opět krokem 1.

Dá se dokázat, že RTM M je typu Las-Vegas.

Nyní ukážeme, že $\mathcal{ZPP} \subseteq \mathcal{RP} \cap$ co-$\mathcal{RP}$.

Předpokládejme, že jazyk L leží ve třídě $\mathcal{ZPP}$, existuje tedy pro něj RTM M_1 typu Las-Vegas. Označme $p(n)$ polynom, který udává střední hodnotu počtu kroků RTM M_1 pro vstupní slovo délky n. Vytvoříme RTM M typu Monte Carlo pracující polynomiálním čase pro jazyk L.

M nechá na vstupu w pracovat RTM M_1 po dobu $2p(n)$. Jestliže M_1 úspěšně skončí, M úspěšně skončí; ve všech ostatních případech RTM M skončí neúspěšně.

Dá se dokázat, že M splňuje všechny podmínky pro RTM typu Monte Carlo. Protože pracuje v čase $2p(n)$, jedná se o polynomiální RTM typu Monte Carlo. Proto je jazyk L ve třídě $\mathcal{RP}$.

Protože třída $\mathcal{ZPP}$ je uzavřena na doplňky, je každý jazyk ze třídy $\mathcal{ZPP}$ také ve třídě co-$\mathcal{RP}$.

4.7.13 Věta. Platí

$$\mathcal{P} \subseteq \mathcal{ZPP}, \ \mathcal{RP} \subseteq \mathcal{NP}, \quad \text{co-}\mathcal{RP} \subseteq \ \text{co-}\mathcal{NP}.$$

První inkluze je zřejmá, každý polynomiální Turingův stroj můžeme považovat za randomizovaný Turingův stroj typu Las-Vegas.

Druhá inkluze je složitější. Její důkaz spočívá v tom, že pro daný polynomiální RTM M typu Monte Carlo pracující v polynomiálním čase zkonstruujeme nedeterministický Turingův stroj, který přijímá jazyk $L(M)$.

Třetí inkluze jednoduše vyplývá z definic tříd co-$\mathcal{RP}$, co-$\mathcal{NP}$ a z druhé inkluze.

Kapitola 5

Nerozhodnutelnost

5.1 Rekursivní a rekursivně spočetné jazyky

5.1.1 Rekursivní jazyky. Řekneme, že jazyk L je *rekursivní*, jestliže existuje Turingův stroj M, který rozhoduje jazyk L.

Připomeňme, že Turingův stroj M rozhoduje jazyk L znamená, že jej přijímá a na každém vstupu se zastaví (buď úspěšně nebo neúspěšně).

Třída rekursivních jazyků se často značí R.

5.1.2 Rekursivně spočetné jazyky. Řekneme, že jazyk L je *rekursivně spočetný*, jestliže existuje Turingův stroj M, který tento jazyk přijímá.

Jinými slovy, M se pro každé slovo w, které patří do L, úspěšně zastaví a pro slovo w, které nepatří do L se buď zastaví neúspěšně nebo se nezastaví vůbec.

Třída rekursivně spočetných jazyků se často značí RS.

5.1.3 Poznámka. Jazykům, které nejsou rekursivní, také říkáme, že jsou *algoritmicky neřešitelné* nebo *nerozhodnutelné*. Obdobně mluvíme o úlohách, které jsou nerozhodnutelné nebo algoritmicky neřešitelné. První pojem se užívá častěji pro rozhodovací úlohy, druhý i pro úlohy konstrukční či optimalizační.

Každý rekursivní jazyk je též rekursivně spočetný. V dalším textu ukážeme, že naopak to neplatí, tj. existují rekursivně spočetné jazyky, které nejsou rekursivní.

5.1.4 Tvrzení. Jestliže jazyk L je rekursivní, pak je rekursivní i jeho doplněk $\overline{L}$.

5.1.5 Tvrzení. Jestliže jazyk L i jeho doplněk $\overline{L}$ jsou oba rekursivně spočetné, pak L je rekursivní.

5.1.6 Tvrzení. Pro jazyk L může nastat jedna z následujících možností:

1. L i $\overline{L}$ jsou oba rekursivní.
2. Jeden z L a $\overline{L}$ je rekursivně spočetný a druhý není rekursivně spočetný.
3. L i $\overline{L}$ nejsou rekursivně spočetné.

5.1.7 Kód Turingova stroje. Každý Turingův stroj M lze zakódovat jako binární slovo. Mějme Turingův stroj M s množinou stavů $Q = \{q_1, q_2, \ldots, q_n\}$, množinou vstupních symbolů $\Sigma = \{0, 1\}$, množinou páskových symbolů $\Gamma = \{X_1, X_2, \ldots, X_m\}$, kde $X_1 = 0$, $X_2 = 1$ a $X_3 = B$. Dále počáteční stav je stav q_1, koncový stav je q_2. Označme D_1 pohyb hlavy doprava a D_2 pohyb hlavy doleva. (Tj. $D_1 = R$ a $D_2 = L$.)

Jeden přechod stroje M

$$\delta(q_i, X_j) = (q_k, X_l, D_r)$$

zakódujeme slovem

$$w = 0^i 10^j 10^k 10^l 10^r.$$

Kód Turingova stroje M, značíme jej $\langle M \rangle$, je

$$\langle M \rangle = 111\, w_1\, 11\, w_2\, 11 \ldots 11\, w_p\, 111,$$

Kde $w_1, \ldots, w_p$ jsou slova odpovídající všem přechodům stroje M.

5.1.8 Binární slova můžeme uspořádat do posloupnosti a tudíž je očíslovat. Jedno z možných očíslování je toto: K binárnímu slovu w utvoříme $1w$ a toto chápeme jako binární zápis přirozeného čísla.

Tedy např. ϵ je první slovo, 0 je druhé slovo, 1 je třetí slovo, atd, 100110 je $1100110 = 64 + 32 + 4 + 2 = 102$, tj. 100110 je 102-hé slovo. V dalším textu o binárním slovu na místě i mluvíme jako o slovu w_i. Tedy $w_1 = \epsilon$, $w_{102} = 100110$.

Jedná se vlastně o uspořádání slov nejprve podle délky a mezi slovy stejné délky o lexikografické uspořádání.

5.1.9 Diagonální jazyk L_d. Nejprve uděláme následující úmluvu. Jestliže binární slovo w nemá tvar z 5.1.7, považujeme ho za kód Turingova stroje M, který nepřijímá žádné slovo (neudělá nikdy žádný krok). Tj. $L(M) = \emptyset$.

Jazyk L_d se skládá ze všech binárních slov w takových, že Turingův stroj s kódem w nepřijímá slovo w. (Tedy L_d obsahuje i všechna slova w, která neodpovídají kódům nějakého Turingova stroje, ovšem obsahuje i další binární slova.)

5.1.10 Věta. Neexistuje Turingův stroj, který by přijímal jazyk L_d. Jinými slovy, $L_d \neq L(M)$ pro každý Turingův stroj M.

Nástin důkazu. Postupujeme sporem. Kdyby existoval Turingův stroj M takový, že $L_d = L(M)$, pak by tento Turingův stroj měl kód roven nějakému binárnímu slovu, tj. $\langle M \rangle = w_i$ pro nějaké i.

Na otázku, zda toto slovo w_i patří nebo nepatří do jazyka L_d, nemůžeme dát odpověď, která by nevedla ke sporu.

Kdyby $w_i \in L_d$, pak w_i splňuje podmínku: Turingův stroj s kódem w_i nepřijímá slovo w_i. Ale $L_d = L(M)$ kde $w_i = \langle M \rangle$ — spor.

Kdyby $w_i \notin L_d$, pak Turingův stroj s kódem w_i přijímá slovo w_i. Ale to je podmínka pro to, aby slovo w_i patřilo do $L(M)$ — spor.

Proto neexistuje Turingův stroj, který by přijímal jazyk L_d.

5.1.11 Univerzální jazyk. *Univerzální jazyk* L_{UN} je množina slov tvaru $\langle M\rangle\, w$, kde $\langle M\rangle$ je kód Turingova stroje a $w \in \{0,1\}^\star$ je binární slovo takové, že $w \in L(M)$.

5.1.12 Univerzální Turingův stroj. Popíšeme, velmi zhruba, Turingův stroj, který přijímá univerzální jazyk L_{UN}. Tomuto Turingovu stroji se také říká *univerzální Turingův stroj* a značíme ho U.

Univerzální Turingův stroj U má 4 pásky. První páska obsahuje vstupní slovo $\langle M\rangle\, w$, druhá páska simuluje pásku Turingova stroje M a třetí páska obsahuje kód stavu, ve kterém se Turingův stroj M nachází. Dále má U ještě čtvrtou, pomocnou pásku.

Na začátku práce Turingova stroje U je na první pásce vstupní slovo $\langle M\rangle\, w$, ostatní pásky obsahují pouze B, blanky. Připomeňme, že kód Turingova stroje získáme takto. Předpokládejme, že Turingův stroj M se skládá z $(Q, \{0,1\}, \{0,1,B\}, \delta, q_1, \{q_2\})$, kde $Q = \{q_1, q_2, \ldots, q_n\}$. Označme 0 jako X_1, 1 jako X_2, B jako X_3, pohyb doprava R jako D_1, pohyb doleva L jako D_2. Pak jednotlivé přechody $\delta(q_i, X_j) = (q_k, X_l, D_m)$ kódujeme

$$t = 0^i 1 0^j 1 0^k 1 0^l 1 0^m, \quad \text{kde } 1 \le i,k \le n, 1 \le j,l \le 3, 1 \le m \le 2.$$

Turingův stroj M má kód

$$111\, t_1\, 11\, t_2\, 11 \ldots 11\, t_r\, 111.$$

Turingův stroj U nejprve zkontroluje, že vstup je opravdu kódem Turingova stroje M následovaný binárním slovem. Jestliže není, U se neúspěšně zastaví.

V případě, že vstupní slovo je tvaru kód Turingova stroje M následovaný binárním slovem w, U přepíše slovo w na druhou pásku a na třetí pásku napíše 0. To je proto, že Turingův stroj je na začátku práce ve stavu q_1 kódovaném jako 0.

Nyní Turingův stroj U simuluje kroky Turingova stroje M s tím, že kdykoli se stroj M dostane do stavu q_2 (koncový „přijímací“ stav M), U se úspěšně zastaví. Toto poznáme tak, že na třetí pásce se objeví 00 předcházené a následované B, blanky.

Poznamenejme, že je třeba ještě řada dalších technických detailů. Např. při přepisování slova w na druhou pásku to děláme tak, že za 0 ve vstupním slově w na pásku napíšeme 10, za 1 ve w na druhou pásku zapíšeme 100. Je-li na druhou pásku potřeba (vzhledem k přechodové funkci Turingova stroje M) na druhou pásku napsat B, napíšeme 1000. Čtvrtá páska slouží k tomu, abychom na druhou pásku byli schopni vždy napsat stav pásky TM M.

5.1.13 Důsledek. Univerzální jazyk L_{UN} je rekursivně spočetný.

5.1.14 Tvrzení. Univerzální jazyk L_{UN} není rekursivní.

Kdyby totiž L_{UN} byl rekursivní, existoval by Turingův stroj M, který rozhodne L_{UN}. Tj. M se vždy zastaví; na slovech z jazyka L_{UN} se úspěšně zastaví, na slovech neležících v L_{UN} se neúspěšně zastaví. Na základě tohoto Turingova stroje M bychom byli schopni rozhodnout diagonální jazyk L_d, o kterém víme, že není ani rekursivně spočetný, viz 5.1.10.

5.1.15 Redukce. Připomeňme definici redukce z **??**.

Jsou dány dvě rozhodovací úlohy $\mathcal{U}$ a $\mathcal{V}$. Řekneme, že úloha $\mathcal{U}$ se *redukuje* na úlohu $\mathcal{V}$, jestliže existuje algoritmus (program pro RAM, Turingův stroj) $\mathcal{A}$, který pro každou instanci I úlohy $\mathcal{U}$ zkonstruuje instanci I' úlohy $\mathcal{V}$ a to tak, že

$$I \text{ je ANO instance } \mathcal{U} \text{ iff } I' \text{ je ANO instance } \mathcal{V}.$$

Fakt, že úloha $\mathcal{U}$ se redukuje na úlohu $\mathcal{V}$ značíme

$$\mathcal{U} \lhd \mathcal{V}.$$

Jsou dány dva jazyky $L_1 \subseteq \Sigma^\star$, $L_2 \subseteq \Gamma^\star$. Řekneme, že jazyk L_1 se *redukuje* na jazyk L_2, jestliže existuje algoritmus (program pro RAM, Turingův stroj) $\mathcal{A}$, který pro každé slovo $w \in \Sigma^\star$ zkonstruuje slovo $A(w) \in \Gamma^\star$ a to tak, že

$$w \in L_1 \text{ iff } A(w) \in L_2.$$

Fakt, že jazyk L_1 se redukuje na jazyk L_2 značíme

$$L_1 \lhd L_2.$$

5.1.16 Tvrzení. Jsou dány dvě úlohy $\mathcal{U}$ a $\mathcal{V}$ takové, že $\mathcal{U} \lhd \mathcal{V}$. Pak platí:

1. Jestliže $\mathcal{V}$ je rozhodnutelná, pak i $\mathcal{U}$ je rozhodnutelná.
2. Jestliže $\mathcal{U}$ je nerozhodnutelná, pak i $\mathcal{V}$ je nerozhodnutelná.
3. Jestliže jazyk úlohy $\mathcal{U}$ není rekursivně spočetný, pak i jazyk úlohy $\mathcal{V}$ není rekursivně spočetný.

5.1.17 Tvrzení. Jsou dány jazyky

$$L_e = \{\langle M\rangle \mid L(M) = \emptyset\}, \quad L_{ne} = \{\langle M\rangle \mid L(M) \neq \emptyset\}.$$

Pak jazyk L_{ne} je rekursivně spočetný, ale ne rekursivní. Jakyk L_e není ani rekursivně spočetný.

5.1.18 Poznámka. Uvědomme si, že jazyk L_e je doplňkem jazyka L_{ne}. Ano, jestliže slovo w není kódem nějakého Turingova stroje, pak ho považujeme za kód stroje, který nepřijímá žádné slovo, tj. patří do jazyka L_e.

Univerzální Turingův stroj U se dá využít i k tomu abychom ukázali, že jazyk L_{ne} je rekursivně spočetný. Z redukce $L_{UN} \lhd L_{ne}$ a 5.1.16 dostáváme, že L_{ne} není rekursivní. Fakt, že L_e není ani rekursivně spočetný pak vyplývá z 5.1.5.

5.1.19 Věta (Rice). Jakákoli netriviální vlastnost rekursivně spočetných jazyků (jazyků přijímaných Turingovým strojem) je nerozhodnutelná.

Poznamenejme, že netriviální vlastností se rozumí každá vlastnost, kterou má aspoň jeden rekursivně spočetný jazyk a nemají ho všechny rekursivně spočetné jazyky.

5.2 Další nerozhodnutelné úlohy

5.2.1 V minulém oddíle jsme uvedli několik nerozhodnutelných jazyků — úloh. Věta (Rice) dokonce říká, že každá netriviální vlastnost rekursivních jazyků je nerozhodnutelná. Na druhou stranu úlohy týkající se rekursivních jazyků se mohou zdát jako značně umělé. V této části ukážeme další úlohy, které jsou nerozhodnutelné. Poznamenejme ještě, že univerzální jazyk L_{UN} hraje pro nerozhodnutelné jazyky/úlohy obdobnou roli jako hrál problém splnitelnosti booleovských formulí pro $\mathcal{NP}$ úplné úlohy.

Označme $\mathcal{UN}$ úlohu odpovídající univerzálnímu jazyku L_{UN}; tj. tuto úlohu: Instance se skládá z TM M a slova w. Jedná se o ano instanci právě tehdy, když $w \in L(M)$.

5.2.2 Postův korespondenční problém (PCP). Jsou dány dva seznamy slov A, B nad danou abecedou Σ.

$$A = (w_1, w_2, \ldots, w_k), \quad B = (x_1, x_2, \ldots, x_k),$$

kde $w_i, x_i \in \Sigma^\star$, $i = 1, 2, \ldots, k$. Řekneme, že dvojice A, B *má řešení*, jestliže existuje posloupnost $i_1, i_2, \ldots, i_r$ indexů, tj $i_j \in \{1, 2, \ldots, k\}$, taková, že

$$w_{i_1}\, w_{i_2}\, \ldots\, w_{i_r} \;=\; x_{i_1}\, x_{i_2}\, \ldots\, x_{i_r}.$$

Otázka: Existuje řešení dané instance?

5.2.3 Příklady.

1. Jsou dány seznamy

	1	2	3	4	5
A	011	0	101	1010	010
B	1101	00	01	00	0

   Tato instance má řešení, např. $2, 1, 1, 4, 1, 5$ je

$$w_2\, w_1\, w_1\, w_4\, w_1\, w_5 = 00110111010011010 = x_2\, x_1\, x_1\, x_4\, x_1\, x_5.$$

2. Jsou dány seznamy

	1	2	3	4	5
A	11	0	101	1010	010
B	101	00	01	00	0

   Tato instance nemá řešení.

5.2.4 Modifikovaný Postův korespondenční problém (MPCP). Jsou dány dva seznamy slov A, B nad danou abecedou Σ.

$$A = (w_1, w_2, \ldots, w_k), \quad B = (x_1, x_2, \ldots, x_k),$$

kde $w_i, x_i \in \Sigma^\star$, $i = 1, 2, \ldots, k$. Řekneme, že dvojice A, B *má řešení*, jestliže existuje posloupnost $1, i_1, i_2, \ldots, i_r$ indexů, tj $i_j \in \{1, 2, \ldots, k\}$, taková, že

$$w_1\, w_{i_1}\, w_{i_2}\, \ldots\, w_{i_r} \;=\; x_1\, x_{i_1}\, x_{i_2}\, \ldots\, x_{i_r}.$$

Otázka: Existuje řešení dané instance?

5.2.5 Poznámka. Modifikovaný Postův korespondenční problém se od Postova korespondenčního problému liší tím, že v MPCP vyžadujeme, aby hledaná posloupnost indexů vždy začínala jedničkou. Význam MPCP spočívá v tom, že se dá dokázat následující věta.

5.2.6 Věta. Platí

$$\mathcal{UN} \lhd \text{MPCP} \lhd \text{PCP}.$$

Nástin druhé redukce. Máme dánu instanci MPCP

$$A = (w_1, w_2, \ldots, w_k), \quad B = (x_1, x_2, \ldots, x_k),$$

Předpokládejme, že # a $*$ nejsou prvky Σ, Vytvoříme novou instanci PCP

$$C = (y_0, y_1, \ldots, y_k, y_{k+1}), \quad D = (z_0, z_1, \ldots, z_k, z_{k+1}),$$

kde

1. Pro každé $i = 1, \ldots, k$ slovo y_i vzniklo ze slova w_i tím, že jsme **za** každý symbol slova w_i umístili symbol $*$; obdobně z_i vzniklo ze slova x_i přidáním symbolu $*$ **před** každý symbol slova x_i.
2. $y_0 = *y_1$; $z_0 = z_1$.
3. $y_{k+1} = \#$, $z_{k+1} = *\#$.

Není těžké nahlédnout, že A, B má řešení $1, i_1, \ldots, i_r$ právě tehdy, když má řešení C, D a to musí být $0, i_1, \ldots, i_r, k+1$.

5.2.7 Poznámka. První redukce je obtížnější. Jedná se o popis práce Turingova stroje pomocí slov nad vhodnou abecedou. Trik spočívá v tom, že posloupnost pro MPCP musí začínat prvním slovem (to zajistí, že Turingův stroj začne pracovat v počátečním stavu s daným obsahem pásky). Pro seznam A bude slovo vždy „dohánět výpočet podle přechodové funkce Turingova stroje“, který bude odpovídat seznamu B. Bude tedy slovo vytvořené podle seznamu A prefixem slova vytvořeného podle seznamu B. Slova se stanou stejnými teprve v okamžiku, kdy se TM dostaneme do koncového stavu; tj. kdy se ve slově podle seznamu B objeví koncový stav.

5.2.8 Důsledek. Postův korespondenční problém je nerozhodnutelný.

5.2.9 Poznámka. Kdybychom omezili možnou délku hledané posloupnosti $i_1, i_2, \ldots, i_r$, (tj. omezili r), problém by se stal algoritmicky řešitelným — existoval by algoritmus hrubé síly. Také, kdybychom místo seznamů A, B uvažovali množiny slov, problém by byl dokonce polynomiálně řešitelný.

5.2.10 Víceznačnost bezkontextových gramatik. Je dána bezkontextová gramatika $\mathcal{G} = (N, \Sigma, S, P)$, kde N je množina neterminálních symbolů, Σ je množina terminálních symbolů, S je startovací symbol a P je množina pravidel typu $X \to \alpha$ pro $X \in N$, $\alpha \in (N \cup \Sigma)^\star$.

Otázka: Rozhodněte, zda existuje slovo w, které má dva různé derivační stromy.

5.2.11 Věta. Platí

$$\text{PCP} \lhd \text{víceznačnost bezkontextových gramatik.}$$

5.2.12 Nástin redukce pro důkaz věty 5.2.11. Je dána instance PCP, tj. seznamy slov $A = (w_1, w_2, \ldots, w_k)$ a $B = (x_1, x_2, \ldots, x_k)$. Sestrojíme bezkontextovou gramatiku $\mathcal{G} = (\{S, A, B\}, \Sigma \cup \{a_1, a_2, \ldots, a_k\}, S, P)$, kde P obsahuje tato pravidla

$$S \to A \mid B,$$

$$A \to w_1\, A\, a_1 \mid w_2\, A\, a_2 \mid \ldots \mid w_k\, A\, a_k,$$

$$A \to w_1\, a_1 \mid w_2\, a_2 \mid \ldots \mid w_k\, a_k,$$

$$B \to x_1\, B\, a_1 \mid x_2\, B\, a_2 \mid \ldots \mid x_k\, B\, a_k,$$

$$B \to x_1\, a_1 \mid x_2\, a_2 \mid \ldots \mid x_k\, a_k,$$

Pak gramatika $\mathcal{G}$ je víceznačná právě tehdy, když nějaké slovo $wa_{i_1}a_{i_2}\ldots a_{i_r}$, $w \in \Sigma^\star$, má dvě různá odvození. Tato situace nastává právě tehdy, když instance PCP má řešení. (Uvědomte si, že dvě různá odvození jsou možná jen, můžeme-li stejné slovo $wa_{i_1}a_{i_2}\ldots a_{i_r}$ odvodit při použití pravidla $S \to A$ i $S \to B$, tedy w vytvořit ze seznamu A i ze seznamu B při použití slov se stejným indexem.)

5.2.13 Věta. Jsou dány bezkontextové gramatiky $\mathcal{G}_1$ a $\mathcal{G}_2$. Označme $L(\mathcal{G}_1)$ a $L(\mathcal{G}_2)$ jazyky generované gramatikami $\mathcal{G}_1$ a $\mathcal{G}_2$. Následující úlohy jsou nerozhodnutelné.

1. $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$.
2. $L(\mathcal{G}_1) = L(\mathcal{G}_2)$.
3. $L(\mathcal{G}_1) \subseteq L(\mathcal{G}_2)$.
4. $L(\mathcal{G}_1) = \Sigma^\star$.

5.2.14 Tiling problém. Jsou dány čtvercové dlaždičky velikosti 1 cm^2 několika typů. Každá dlaždička má barevné okraje. Máme neomezený počet dlaždiček každého typu.

Otázka: Je možné dlaždičkami vydláždit každou plochu daného typu tak, aby se dlaždičky dotýkaly hranami stejné barvy, za předpokladu, že dlaždičky nesmíme rotovat?

5.2.15 Věta. Tiling problém je nerozhodnutelný.

Tedy speciálně je nerozhodnutelné, zda každou neomezenou plochu je možné vydláždit předem danou sadou dlaždiček.