19 Složitost

Základní otázka teorie složitosti je „jak rychle dokážeme nějaký problém řešit“. Budeme přitom vycházet z modelu Turingova stroje jako základního výpočetního modelu. Abychom mohli na tuto otázku smysluplně odpovídat, musíme se omezit pouze na ty Turingovy stroje, které na všech vstupech ukončí svůj výpočet. Studium teorie složitosti se tedy omezuje na třídu rekurzivních jazyků, proto se jí také říká subrekurzivní vyčíslitelnost.

19.1 Model složitosti

Složitost je v nejobecnějším významu míra využití zdroje. Mezi základní zdroje, které může algoritmus využívat, je především čas a paměť (prostor). Můžeme se ovšem ocitnout v situaci, kdy máme pro výpočty jiné omezení, např. počet přístupů na disk, počet přístupů do sítě, energie nebo též množství nějakého vzácného prvku. Požadavek na nízkou časovou náročnost se ovšem zdá být ve většině případů univerzální, proto jeho studium matematické teorii složitosti a algoritmů dominuje.

Definice. Složitost Turingova stroje $\mathcal{M}$ definujeme postupně jako následující zobrazení:

- *časová složitost stroje $\mathcal{M}$ na vstupu* jako zobrazení $t_{\mathcal{M}} \colon \Sigma^* \to \mathbb{N}$, kde $t_{\mathcal{M}}(w)$ je počet kroků, které provede stroj $\mathcal{M}$ na vstupu w. V případě nedeterministického stroje bereme maximum ze všech možných výpočtů na daném vstupu.
- *časová složitost stroje* $\mathcal{M}$ jako zobrazení $T_{\mathcal{M}} : \mathbb{N}_0 \to \mathbb{N}$, kde $T_{\mathcal{M}}(n)$ je největší počet kroků, které stroj $\mathcal{M}$ provede na vstupu délky n, tj:

$$T_{\mathcal{M}}(n) = \max\{T_{\mathcal{M}}(w) \mid |w| = n\}$$

- *prostorová složitost stroje $\mathcal{M}$ na vstupu* jako zobrazení $s_{\mathcal{M}} \colon \Sigma^* \to \mathbb{N}$, kde $s_{\mathcal{M}}(w)$ je počet políček na pásce, které stroj $\mathcal{M}$ navštíví během výpočtu na slově w. V případě nedeterministického stroje bereme opět maximum ze všech možných výpočtů na daném vstupu.
- *prostorová složitost stroje* $\mathcal{M}$ jako zobrazení $S_{\mathcal{M}} : \mathbb{N}_0 \to \mathbb{N}$, kde $S_{\mathcal{M}}(n)$ je maximální počet políček, které navštíví stroj $\mathcal{M}$ během svého výpočtu na vstupu délky n, tj:

$$S_{\mathcal{M}}(n) = \max\{s_{\mathcal{M}}(w) \mid |w| = n\}$$

Říkáme také, že stroj $\mathcal{M}$ pracuje *v čase* $T_{\mathcal{M}}(n)$ a *v prostoru* $S_{\mathcal{M}}(n)$.

Poznámka. V definici složitosti bereme maximum přes všechny možné vstupy dané velikosti. Odpovídá to konvenci počítat vždy s nejhorším možným případem. Existují ale i jiné přístupy. Bylo by např. možné počítat průměrnou složitost, operace maxima by se nahradila aritmetickým průměrem. Tohoto přístupu se využívá především v teorii náhodnostních algoritmů, kde se složitost definuje jako střední hodnota (vážený aritmetický průměr).

Poznámka. Uvedená definice složitosti silně závisí na volbě Turingova stroje jako základního výpočetního modelu. Volbou jiného modelu se tato definice může značně lišit. Tento problém budeme rozebírat dále v textu.

19.2 Porovnávání růstu

Motivace. Protože časová i prostorová složitost stroje jsou definovány jako funkce $\mathbb{N} \to \mathbb{N}$ a ještě obecněji jako funkce $\mathbb{N} \to \mathbb{R}^+$ (abychom se nemuseli zabývat zaokrouhlováním), je studium teorie složitosti především studiem funkcí tohoto typu. Zastavme se na krátce u dvou problémů:

a) Máme-li dva stroje, které řeší tentýž problém, který z nich jej řeší lépe? Intuitivní odpovědí je pochopitelně ten, který jej řeší rychleji. Co ale znamená toto „rychleji"? Složitost jsme si nezaváděli jako čísla, která umíme snadno srovnat, ale funkce a ty nemusí být vždy porovnatelné. Příkladem jsou funkce $10n$ a n^2 pro $n < 10$ platí $10n > n^2$ a pro $n > 10$ zase $10n < n^2$, tak kterou z nich prohlásit za „tu, s menší hodnotou"? Vzhledem k praktickým omezením dnešních počítačů nás zajímá, jak se složitost vyvíjí pro velké vstupy, tj. pro $n \to \infty$. Zajímá nás tzv. *asymptotické chování* funkcí. Řečeno laicky, pro porovnání nám slouží kritérium, zda jedna funkce někdy tu druhou přeroste.

b) Je tu ještě další problém, který musíme vzít v potaz před následující definicí. Každý Turingův stroj můžeme „lineárně urychlovat" tím, že jej necháme namísto s jednotlivými symboly počítat s dvojicemi (nebo obecně k-ticemi) symbolů. Tento fakt se shrnuje do tvrzení, že pro každý stroj $\mathcal{M}$ a libovolné $m \in \mathbb{N}$ existuje ekvivalentní stroj $\mathcal{M}'$ splňující:

$$T'_{\mathcal{M}}(n) \leq \frac{T_{\mathcal{M}}(n)}{m} + n + 2$$

Toto zrychlení je tedy vždy nejvýše lineární a celková složitost nemůže klesnout pod n.

Tyto dvě úvahy musíme zahrnout do naší definice. Tato definice tedy musí zohledňovat chování funkcí pro vysoké hodnoty n a musí ignorovat násobení konstantou. V tomto momentě už by měla následující volba připadat naprosto přirozeně.

Definice. Řekneme, že funkce f *neroste asymptoticky rychleji než* funkce g, jestliže existují $n_0 \in \mathbb{N}$ a $c \in \mathbb{R}^+$ taková, že pro každé $n \geq n_0$ platí $f(n) \leq cg(n)$. O funkci g říkáme, že je *asymptotická horní závora pro* f. Množinu funkcí, které nerostou asymptoticky rychleji než g značíme $\mathcal{O}(g)$.

Příklad 1. Platí $10n \in \mathcal{O}(n^2)$ volbou $n_0 = 10$ a $c = 1$ nebo volbou $n_0 = 1$ a $c = 10$.

Příklad 2. Při poměřování asymptotického růstu nezáleží na základech logaritmů, totiž $\log_a n \in \mathcal{O}(\log_b n)$ pro libovolné $a, b > 1$. To vyplývá ze vzorce:

$$\log_a n = \log_a b \cdot \log_b n$$

Dvě logaritmické funkce o různých základech se liší pouze násobkem konstantou. U logaritmů tedy nebudeme uvádět základ, popřípadě si jej při výpočtech volit podle potřeby.

Příklad 3. Platí $\log n \in \mathcal{O}(n)$. To vyplyne, jakmile ukážeme, že $\log n \leq n$ pro každé $n \in \mathbb{N}$. To se dá ukázat geometricky nebo indukcí, totiž $\log 1 = 0 < 1$ a za předpokladu, že tvrzení platí pro n, je indukční krok tvaru:

$$\log(n+1) \leq \log(n+n) = \log n + \log 2 = \log n + 1 \leq n + 1$$

Věta 19.1. Vlastnosti symbolu $\mathcal{O}$:

a) Pro libovolnou funkci $f\colon \mathbb{N} \to \mathbb{R}^+$ platí $f \in \mathcal{O}(f)$.

b) Pro libovolné funkce $f, g, h\colon \mathbb{N} \to \mathbb{R}^+$ platí:

$$f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h) \Rightarrow f \in \mathcal{O}(h)$$

Relaci „neroste asymptoticky rychleji než“ je reflexivní a tranzitivní (není uspořádáním, neboť není antisymetrická), proto ji můžeme intuitivně chápat jako „menší nebo rovno“. K tomu dostáváme přidružené pojmy:

pojem	analogie	značka
neroste rychleji	$\leq$	$\mathcal{O}$
neroste pomaleji	$\geq$	Ω
roste stejně rychle	$\equiv$	Θ
roste pomaleji	$<$	o
roste rychleji	$>$	ω

Definice. Uveďme formální definice všech předchozích pojmů:

$$\mathcal{O}(g) = \{f\colon \mathbb{N}_0 \to \mathbb{R} \mid \exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+ \forall n \geq n_0 : f(n) \leq cg(n)\}$$
$$\Omega(g) = \{f\colon \mathbb{N}_0 \to \mathbb{R} \mid \exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+ \forall n \geq n_0 : f(n) \geq cg(n)\}$$
$$\Theta(g) = \{f\colon \mathbb{N}_0 \to \mathbb{R} \mid \exists n_0 \in \mathbb{N}, c_1, c_2 \in \mathbb{R}^+ \forall n \geq n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Dále:

$$o(g) = \left\{ f\colon \mathbb{N}_0 \to \mathbb{R} \mid \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \right\}$$
$$\omega(g) = \left\{ f\colon \mathbb{N}_0 \to \mathbb{R} \mid \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \right\}$$

Cvičení 1. Uspořádejte následující funkce podle růstu:

2^n	$n \log n$	$n!$	$\log \log n$	e^n	$n^5 - 100n + 3$
n^n	$\log n!$	$\log^5 n$	1	2^{2^n}	$\arctan n$
n	$\sqrt{2^n}$	$\sqrt[5]{n}$	$(2n)!$	$(n!)^2$	$\log n^2$

Cvičení 2. Dokažte následující implikace:

a) $f \in o(g) \wedge g \in o(h) \Rightarrow f \in o(h)$

b) $f \in o(g) \Leftrightarrow g \in \omega(f)$

c) $f \in o(g) \Rightarrow f \in \mathcal{O}(g)$

d) $g \notin \mathcal{O}(f) \Rightarrow f \in o(g)$

Asymptotickou symboliku můžeme používat spolu s některými základními aritmetickými operacemi:

a) Výrazem $\mathcal{O}(f) + \mathcal{O}(g)$ většinou dáváme najevo, že stroj provádí za sebou dva úseky v časech $\mathcal{O}(f)$ a $\mathcal{O}(g)$. Je potom logické klást:

$$\mathcal{O}(f) + \mathcal{O}(g) = \mathcal{O}(f + g)$$

b) Podobně můžeme použít $\mathcal{O}(f) \cdot \mathcal{O}(g) = \mathcal{O}(fg)$ pro případy, kdy stroj opakuje úsek, jehož časová složitost je $\mathcal{O}(g)$, a počet iterací je asymptoticky ohraničen funkcí f. Díky tomu stačí znát odhad počtu iterací a ne jeho přesný počet.

c) V některých případech bude užitečné psát $2^{\mathcal{O}(n)}$, je ale třeba si uvědomit $2^{\mathcal{O}(n)} \neq \mathcal{O}(2^n)$, protože např:

$$\mathcal{O}(2^n) \not\ni 4^n = 2^{2n} \in 2^{\mathcal{O}(n)}$$

19.3 Vliv modelu

Jak už bylo řečeno, volba výpočetního modelu silně ovlivňuje rychlost výpočtů. Např. RAM (random access machine) pracující v čase $\mathcal{O}(f(n))$ lze převést na vícepáskový deterministický Turingův stroj pracující v čase $\mathcal{O}(f^3(n)(f(n)+n)^2)$. Některé vztahy probereme formálně.

Věta 19.2 (Cena za jednu pásku.)**.** Pro každý vícepáskový Turingův stroj pracující v čase $\mathcal{O}(f(n))$ existuje ekvivalentní jednopáskový Turingův stroj pracující v čase $\mathcal{O}(f^2(n))$.

Důkaz. Obsahy všech pásek zapíšeme za sebe a každý krok výpočtu vícepáskového stroje simulujeme průchodem celé pásky. Těchto průchodů je potřeba $\mathcal{O}(f(n))$. Bylo-li k pásek, pak máme za sebou k úseků délky nejvýše $\mathcal{O}(f(n))$. Celkem se provede nejvýše $\mathcal{O}(kf^2(n)) = \mathcal{O}(f^2(n))$ kroků. □

Věta 19.3 (Cena za determinizmus.)**.** Pro každý nedeterministický Turingův stroj pracující v čase $\mathcal{O}(f(n))$ existuje ekvivalentní deterministický Turingův stroj pracující v čase $2^{\mathcal{O}(f(n))}$.

Důkaz. Připomeňme, že převod nedeterministického stroje na deterministický provádíme tak, že procházíme výpočetní strom nedeterministického stroje do šířky. Označme míru větvení tohoto stroje a. Potom $a^{\mathcal{O}(f(n))}$ je horní závora pro počet uzlů v tomto stromě. Na druhé pásce si přitom zaznamenáváme polohu ve stromě pomocí řetězce délky $\mathcal{O}(f(n))$. Složitost této simulace je $\mathcal{O}(f(n)a^{\mathcal{O}(f(n))}) = 2^{\mathcal{O}(f(n))}$. Převod na jednopáskový stroj má kvadratickou cenu, což je stále $(2^{\mathcal{O}(f(n))})^2 = 2^{\mathcal{O}(f(n))}$. □

Cvičení 3. Mějme jazyk $L = \{a^n b^n \mid n \geq 1\}$. Nalezněte:

a) Deterministický jednopáskový stroj rozpoznávající L v čase $\mathcal{O}(n^2)$.

b) Deterministický dvoupáskový stroj rozpoznávající L v čase $\mathcal{O}(n)$.

c) Deterministický jednopáskový stroj rozpoznávající L v čase $\mathcal{O}(n \log n)$.

Stačí uvádět myšlenky, není potřeba stroje formálně konstruovat.

19.4 Třídy složitosti

Protože můžeme mít více Turingových strojů, které řeší daný problém, definujeme *složitost problému* přirozeně jako složitost toho nejrychlejšího z nich.

Definice. Pro libovolnou funkci $f : \mathbb{N}_0 \to \mathbb{R}$ definujeme následující třídy problémů:

$$\begin{aligned}
\mathrm{TIME}(f) &= \{L \mid \text{existuje deterministický Turingův stroj } \mathcal{M} \\
&\qquad \text{akceptující } L \text{ a } T_{\mathcal{M}} \in \mathcal{O}(f)\} \\
\mathrm{NTIME}(f) &= \{L \mid \text{existuje nedeterministický Turingův stroj } \mathcal{N} \\
&\qquad \text{akceptující } L \text{ a } T_{\mathcal{N}} \in \mathcal{O}(f)\} \\
\mathrm{SPACE}(f) &= \{L \mid \text{existuje deterministický Turingův stroj } \mathcal{M} \\
&\qquad \text{akceptující } L \text{ a } S_{\mathcal{M}} \in \mathcal{O}(f)\} \\
\mathrm{NSPACE}(f) &= \{L \mid \text{existuje nedeterministický Turingův stroj } \mathcal{N} \\
&\qquad \text{akceptující } L \text{ a } S_{\mathcal{N}} \in \mathcal{O}(f)\}
\end{aligned}$$

Poznámka. Deterministický stroj je speciálním případem nedeterministického, proto platí:

$$\mathrm{TIME}(f) \subseteq \mathrm{NTIME}(f) \qquad \mathrm{SPACE}(f) \subseteq \mathrm{NSPACE}(f)$$

Poznámka. Žádný stroj nemůže navštívit více políček pásky, než kolik má k dispozici kroků. Proto je prostorová složitost vždy menší nebo stejná jako časová. Z toho dostáváme další dva vztahy:

$$\mathrm{TIME}(f) \subseteq \mathrm{SPACE}(f) \qquad \mathrm{NTIME}(f) \subseteq \mathrm{NSPACE}(f)$$

Definice. Definujeme časové třídy problémů:

$$\mathrm{P} = \bigcup_{k \in \mathbb{N}} \mathrm{TIME}\left(n^k\right) \qquad \mathrm{EXPTIME} = \bigcup_{k \in \mathbb{N}} \mathrm{TIME}\left(2^{n^k}\right)$$

$$\mathrm{NP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}\left(n^k\right) \qquad \mathrm{NEXPTIME} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}\left(2^{n^k}\right)$$

Poznámka. Převod vícepáskového na jednopáskový stroj měl kvadratickou cenu. Polynomiální cenu mají i převody mezi jinými modely (RAM, stroje s více počítadly atd.) Výjimkou je vztah mezi deterministickými a nedeterministickými modely, kde se zatím nepodařil ukázat lepší než exponenciální nárůst. Proto se pro nedeterministické výpočty vyhradily samostatné třídy složitosti a v následující větě již neberou v úvahu. Třídy P, NP, EXPTIME a NEXPTIME nejsou citlivé na volbu výpočetního modelu, říkáme, že jsou *robustní*.

Poznámka. Protože nárůst složitosti při převodu nedeterministického stroje na deterministický je nejvýše asymptoticky exponenciální, dostáváme následující inkluze:

$$\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{NEXPTIME}$$

Na tomto místě uveďme, že doposud nebylo dokázáno, zda inkluze $\mathrm{P} \subseteq \mathrm{NP}$ je či není ostrá, tj. zda ve skutečnosti nenastává rovnost $\mathrm{P} = \mathrm{NP}$. Tento „P rovná se NP“ problém se ukázal obzvláště náročný a v roce 2000 na jeho vyřešení Clay Mathematics Institute vypsal odměnu jeden milion dolarů. Přitom řešení tohoto problému těsně souvisí s efektivitou většiny moderních kryptografických protokolů a jejich bezpečností. Pokud by se totiž ukázalo, že $\mathrm{P} = \mathrm{NP}$, jedním z důsledků by byla existence efektivního algoritmu na prolomení šifer s veřejným klíčem.

Cvičení 4. Ukažte, že třída P je uzavřena na operace doplňku, sjednocení, průniku i komplementu.

19.5 Polynomiální vyčíslitelnost

Z praktického hlediska jsou použitelné pouze algoritmy pracující v polynomiálním čase, tzn. problémy v třídě P, i když existují i výjimky. Např. používaným algoritmem pro řešení systému lineárních nerovnic je simplexový algoritmus, ačkoli má asymptoticky exponenciální složitost a ačkoli existují algoritmy pro tento problém pracující v polynomiálním čase.

Příklad 4. Uvažme *problém cesty* definovaný jako problém příslušnosti do jazyka:

$$\text{PATH} = \{G\#s\#t \mid \text{graf } G \text{ obsahuje cestu z } s \text{ do } t\}$$

Platí PATH $\in$ P.

Příklad 5. Uvažme *problém hamiltonovské cesty* jako problém příslušnosti do jazyka:

$$\text{HAMPATH} = \{G\#s\#t \mid \text{v grafu } G \text{ existuje hamiltonovská cesta z } s \text{ do } t\}$$

Platí HAMPATH $\in$ NP. Nedeterministický stroj pro tento jazyk bude pracovat následovně: Nedeterministický stroj konstruuje cestu z s tak, že vždy nedeterministicky zvolí následující hranu do dalšího vrcholu. Je-li poslední vrchol t a navštívili jsme všechny vrcholy grafu právě jednou, pak akceptujeme, jinak zamítáme. Tento algoritmus má zřejmě polynomiální složitost a akceptuje právě tehdy, když v grafu existuje hamiltonovská cesta z s do t.

Příklad 6. Problém rozpoznání prvočíselnosti:

$$\text{COMPOSITES} = \{n \in \mathbb{N} \mid n = pq \text{ pro nějaká } p, q > 1\}$$

Potom COMPOSITES $\in$ NP. Nedeterministický stroj uhodne dvě čísla p a q, vynásobí je a výsledek porovná se vstupem, pokud se hodnoty shodují, pak akceptuje, jinak zamítá.

Všechny předchozí příklady problémů z NP měli společnou strukturu: Vždy jsme nedeterministicky uhodli řešení a poté deterministicky ověřili, že uhádnuté řešení splňovalo zadání. Ukazuje se, že toto schéma přesně vystihuje třídu NP. Jsou to právě ty problémy, které lze polynomiálně verifikovat. Provedeme formalizaci:

Definice. *Polynomiální verifikátor* pro jazyk L je deterministický Turingův stroj $\mathcal{V}$ takový, že $w \in L$ právě tehdy, když existuje řetězec c takový, že $\mathcal{V}$ akceptuje $w\#c$ v polynomiálním čase vzhledem ke $|w|$.

Věta 19.4. Platí $L \in$ NP právě tehdy, když existuje polynomiální verifikátor pro L.

Důkaz. $\Rightarrow$: Je-li $L \in$ NP, pak podle definice existuje nedeterministický Turingův stroj $\mathcal{N}$ akceptující L v polynomiálním čase. Mějme $w \in L$ libovolné. Za řetězec c zvolme posloupnost cifer, která kóduje seznam nedeterministických výběrů, které provedl stroj $\mathcal{N}$ na vstupu w. Polynomiální verifikátor $\mathcal{V}$ bude pracovat přesně jako stroj $\mathcal{N}$, přičemž nedeterministické volby bude provádět deterministicky podle řetězce c. Zřejmě je $\mathcal{V}$ polynomiální, protože byl $\mathcal{N}$ polynomiální a akceptuje právě $w\#c$ pro $w \in L$.

$\Leftarrow$: Máme-li polynomiální verifikátor $\mathcal{V}$ pro jazyk L, pak můžeme sestrojit nedeterministický stroj $\mathcal{N}$ akceptující L jako stroj, který dostane na vstupu w, nedeterministicky uhádne řetězec c a poté simuluje činnost stroje $\mathcal{V}$ na $w\#c$. Z definice polynomiálního verifikátoru vyplývá, že stroj $\mathcal{N}$ pracuje s polynomiální složitostí. Proto $L \in \mathrm{NP}$.

□

Cvičení 5. Dokažte, že třída NP je uzavřena na sjednocení, průnik i zřetězení.

Poznámka. Otázka uzavřenosti třídy NP na komplement je dodnes otevřeným problémem.

Cvičení 6. Definujeme třídu:

$$\mathrm{coNP} = \{L^C \mid L \in \mathrm{NP}\}$$

Rozhodněte, zda platí následující tvrzení:

a) $\mathrm{NP}^C = \mathrm{coNP}$

b) Třída coNP je uzavřena na průnik.

c) $L_1 \in \mathrm{NP}, L_2 \subsetneq L_1, L_2 \in \mathrm{coNP} \Rightarrow L_1 \setminus L_2 \in \mathrm{NP}$

19.6 Polynomiální redukce

Definice. Řekneme, že problém A se polynomiálně redukuje na B, píšeme $A \leq_p B$, jestliže se na něj redukuje a příslušná funkce je vyčíslitelná Turingovým strojem pracujícím v polynomiálním čase.

Věta 19.5. Nechť $A \leq_p B$. Potom platí:

a) $B \in \mathrm{P} \Rightarrow A \in \mathrm{P}$

b) $A \notin \mathrm{P} \Rightarrow B \notin \mathrm{P}$

c) $B \in \mathrm{NP} \Rightarrow A \in \mathrm{NP}$

d) $A \notin \mathrm{NP} \Rightarrow B \notin \mathrm{NP}$

Analogicky teorii vyčíslitelnosti definujeme koncept těžkého a úplného problému, tentokrát vzhledem k polynomiální redukci.

Definice. Nechť $\mathcal{C}$ je třída jazyků a A libovolný jazyk. Řekneme, že A je

a) *$\mathcal{C}$-těžký*, jestliže $C \leq_p A$ pro libovolné $C \in \mathcal{C}$

b) *$\mathcal{C}$-úplný*, jestliže je $\mathcal{C}$-těžký a $A \in \mathcal{C}$

Poznámka. Každý netriviální problém z třídy P je P-úplný. Pojem úplnosti má tedy smysl uvažovat až pro větší třídy, především pro třídy NP. Stejně jako tomu bylo u třídy rekurzivně spočetných jazyků, stačí nám dokázat o jednom problému, že je NP-úplný. O dalších nám stačí sestrojit redukci nějakého problému, o kterém už víme, že je NP-úplný. Historicky prvním byl takovým problémem SAT. Tento problém je definován jako problém pro danou výrokovou formuli určit, zda je splnitelná. Snadno se ukáže, že SAT $\in$ NP: Nedeterministicky uhodneme valuaci proměnných, potom snadno deterministicky polynomiálně ověříme, že tato valuace splňuje zadanou formuli.

Cvičení 7. Rozhodněte, zda jsou následující formule splnitelné:

a) $(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y)$

b) $(x \vee \neg y) \wedge (x \vee y \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee \neg z)$

c) $(x \vee \neg y) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee \neg z)$

d) $(u \vee \neg v \vee \neg w) \wedge (w \vee \neg y \vee z) \wedge (w \vee \neg z \vee x) \wedge (x \vee y \vee z)$

Věta 19.6 (Cook, Levin)**.** Problém SAT je NP-těžký.

Důkaz. Důkaz nebudeme uvádět, neboť je převážně technicky. Myšlenka je ta, že každý Turingův stroj přeložíme do výrokové formule, přičemž tato formule je splnitelná právě tehdy, když stroj akceptuje vstup. Navíc tato formule má délku úměrnou délce výpočtů původního stroje. Pokud tedy stroj pracoval v polynomiálním čase, je tato redukce polynomiální. □

Příklad 7. Problém 3SAT je podproblém SAT určit, zda zadaná formule v 3-CNF je splnitelná. Formule je v 3-CNF, jestliže je v konjunktivní normální formě a každá klauzule má maximálně tři literály. Ukážeme, že 3SAT je NP-úplný pomocí redukce SAT $\leq$ 3SAT. Pro každou formuli φ v CNF sestrojíme formuli ψ v 3-CNF takovou, že φ je splnitelná právě tehdy, když je ψ splnitelná (formule φ a ψ nemusí být ekvivalentní). Zřejmě můžeme zachovat všechny klauzule, které neobsahují více než tři literály. Každou zbývající klauzuli $x_1 \vee \ldots \vee x_n$ $(n > 3)$ ve φ nahradíme formulí v 3-CNF:

$$(x_1 \vee x_2 \vee y_1) \wedge \ldots \wedge (\neg y_i \vee x_{i+3} \vee y_{i+1}) \wedge \ldots \wedge (\neg y_{n-4} \vee x_{n-1} \vee x_n)$$

kde $y_1, \ldots, y_{n-4}$ jsou nové proměnné, které se nevyskytují v původní formuli φ. Tato redukce je přitom polynomiální, pro každou klauzuli přidáme asymptoticky lineární počet nových proměnných a vše můžeme provést v jedním průchodem formule φ.

Příklad 8. Mnoho grafových problémů je NP-úplných, proveďme důkaz pro následující problém kliky:

$$\text{CLIQUE} = \{G\#k \mid \text{graf } G \text{ obsahuje kliku velikosti } k\}$$

Připomeňme, že klika je úplný podgraf. Provedeme redukci problému 3SAT. Mějme formuli φ v 3-CNF s k klauzulemi. Vytvoříme graf, jehož vrcholy jsou všechny literály ze všech klauzulí. Dva literály jsou spojeny hranou právě tehdy, když se nacházejí v jiné klauzuli a jsou navzájem kompatibilní (jeden není negací druhého). Potom původní formule byla splnitelná právě tehdy, když takto sestrojený graf obsahuje kliku velikosti k. Tato redukce je přitom polynomiální.

Cvičení 8. Dokažte, že *problém podgrafového izomorfizmu* je NP-úplný. Tento problém je definován jako problém pro dané dva grafy H a G určit, zda G obsahuje H jako svůj podgraf (je izomorfní nějakému jeho podgrafu).

19.7 Prostorové třídy složitosti

Prostor je kvalitativně odlišný zdroj než čas, především jej můžeme používat opakovaně (tzv. „recyklovat“), což vede k tomu, že celková prostorová složitost problémů bude nižší než jejích časová složitost. Změna modelu také nevede k tak velkým cenám, jako tomu bylo u času. Např. převod vícepáskového stroje na jednopáskový nemá žádný vliv na použitý počet políček, celková složitost bude prostě součet všech použitých políček na všech páskách.

Věta 19.7. Pro každou funkci $f\colon \mathbb{N}_0 \to \mathbb{R}^+$ platí:

$$\mathrm{SPACE}\left(f(n)\right) \subseteq \mathrm{TIME}\left(2^{\mathcal{O}(f(n))}\right) \qquad \mathrm{NSPACE}\left(f(n)\right) \subseteq \mathrm{NTIME}\left(2^{\mathcal{O}(f(n))}\right)$$

Důkaz. Zvolme za k velikost páskové abecedy stroje, který pracuje v prostoru $\mathcal{O}(f(n))$. Potom tento stroj může navštívit pouze $k^{\mathcal{O}(f(n))}$ možných konfigurací, přičemž v každém kroku výpočtu navštíví právě jednu. Stroj tedy musí skončit v čase $2^{\mathcal{O}(f(n))}$. □

Definice. Zadefinujeme analogické prostorové třídy:

$$\mathrm{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{SPACE}\left(n^k\right) \qquad \mathrm{EXPSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{SPACE}\left(2^{n^k}\right)$$
$$\mathrm{NPSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{NSPACE}\left(n^k\right) \qquad \mathrm{NEXPSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{NSPACE}\left(2^{n^k}\right)$$

Příklad 9. Pomocí polynomiální redukce můžeme dokázat inkluzi:

$$\mathrm{NP} \subseteq \mathrm{PSPACE}$$

Vzpomeňme si, že problém SAT je NP-úplný. Tento problém přitom snadno vyřešit v lineární prostoru. Stačí, když si na druhou pásku vypíšeme nějakou valuaci (což je vlastně posloupnost nul a jedniček) proměnných a zkusíme formuli vyhodnotit (třeba na další pásce). Přitom nespotřebujeme více prostoru, než je trojnásobek vstupu (obsah druhé ani třetí pásky nikdy nepřekročí délku formule). Protože polynomiální redukce musí probíhat také v polynomiálním prostoru (nemůžeme spotřebovat více políček, než máme k dispozici kroků), je každý problém z NP v PSPACE.

Věta 19.8 (Savitch)**.** Pro libovolnou funkci $f : \mathbb{N}_0 \to \mathbb{N}$ splňující $f(n) \geq n$ platí:

$$\mathrm{NSPACE}\left(f(n)\right) \subseteq \mathrm{SPACE}\left(f^2(n)\right)$$

Důkaz. Mějme nedeterministický stroj $\mathcal{N}$ s prostorovou složitostí $f(n)$. Stroj $\mathcal{N}$ upravíme tak, aby měl pouze jednu akceptující konfiguraci - třeba tak, že stroj donutíme před akceptováním smazat obsah pásky a posunout hlavu úplně vlevo. Označme tuto akceptující konfiguraci c_{acc}. Ekvivalentní deterministický stroj $\mathcal{M}$ implementuje proceduru $comp(c_1, c_2, t)$, která akceptuje, pokud lze ve stroji $\mathcal{N}$ během nejvýše t kroků přejít z konfigurace c_1 do konfigurace c_2, jinak zamítá. Označíme-li c_w iniciální konfiguraci pro vstup w, potom stačí, abychom na začátku výpočtu spustili $comp(c_w, c_{\mathrm{acc}}, k^{f(n)})$, kde k je velikost páskové abecedy. Stroj s prostorovou složitostí $f(n)$ totiž nemůže navštívit více než $k^{f(n)}$ konfigurací. Proceduru $comp$ pro trojici (c_1, c_2, t) implementujeme rekurzivně:

a) Jestliže $t = 1$, ověříme pouze, jestli $c_1 = c_2$ nebo $c_1 \vdash_{\mathcal{N}} c_2$.

b) Jestliže $t > 1$, zavoláme $comp(c_1, c, \lfloor t/2 \rfloor)$ a $comp(c, c_2, \lceil t/2 \rceil)$ pro každou konfiguraci c délky $f(n)$. Akceptujeme, jestliže obě volání akceptují

Prostorová složitost každého volání procedury *comp* je $|c_1| + |c_2| + |c| + |t| + C \in \mathcal{O}(f(n))$ (C je nějaká konstanta). Celková prostorová složitost stroje $\mathcal{M}$ je tedy prostorová složitost procedury *comp* krát hloubka rekurze, tedy:

$$T_{\mathcal{M}}(n) \in \mathcal{O}(f(n) \log k^{f(n)}) = \mathcal{O}(f^2(n))$$

□

Poznámka. Důsledkem Savitchovy věty jsou rovnosti:

$$\text{NPSPACE} = \text{PSPACE}$$

$$\text{NEXPSPACE} = \text{EXPSPACE}$$

Dohromady jsme již dokázali následující řetězec:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPTIME}$$

Cvičení 9. Určete vztahy inkluze/rovnost mezi následujícími dvojicemi složitostních tříd. Své tvrzení zdůvodněte.

a) $\text{TIME}\left(n^2\right)$ a $\text{TIME}\left(n^3\right)$

b) $\text{SPACE}\left(2n^2\right)$ a $\text{SPACE}\left(100n^2\right)$

c) $\text{SPACE}\left(n^2\right)$ a $\text{TIME}\left(n^2\right)$

d) $\text{NSPACE}\left(n^2\right)$ a $\text{SPACE}\left(n^5\right)$

e) P a $\text{TIME}\left(2^n\right)$

Příklad 10. Jen pro zajímavost uveďme příklad problému, který je PSPACE-úplný. Jde opět o případ problému splnitelnosti, tj. určit pro danou formuli, zda existuje valuace, která ji splňuje. Formule tentokrát bereme z množiny kvantifikovaných výrokových formulí, jsou to výrokové formule se všeobecnými a existenčními kvantifikátory (kvantifikuje se přes množinu $\{0, 1\}$).