3.2 Počítač s libovolným přístupem – RAM

3.2.1 V tomto oddílu zavedeme další z formálních modelů algoritmu — počítač s libovolným přístupem (tzv. RAM), který je blíže „klasickému“ počítači než Turingův stroj. Ukážeme, 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 $\mathcal{O}(g(n))$.

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

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

4.3 Třída $\mathcal{NPC}$

4.3.1 Redukce a polynomiální redukce úloh. 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) M, který pro každou instanci I úlohy $\mathcal{U}$ zkonstruuje instanci I' úlohy $\mathcal{V}$ a to tak, že

$$I \text{ je ANO-instance } \mathcal{U} \text{ právě tehdy, když } I' \text{ je ANO-instance } \mathcal{V}.$$

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

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

Jestliže navíc algoritmus M pracuje v polynomiálním čase, říkáme, že $\mathcal{U}$ se *polynomiálně* redukuje na $\mathcal{V}$ a značíme

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

Fakt, že se úloha $\mathcal{U}$ redukuje na úlohu $\mathcal{V}$ zhruba řečeno znamená, že $\mathcal{U}$ není obtížnější než $\mathcal{V}$.

4.3.2 Tvrzení. Jsou dány tři rozhodovací úlohy $\mathcal{U}$, $\mathcal{V}$ a $\mathcal{W}$. Jestliže platí

$$\mathcal{U} \lhd_p \mathcal{V} \text{ a } \mathcal{V} \lhd_p \mathcal{W}, \quad \text{pak} \quad \mathcal{U} \lhd_p \mathcal{W}.$$

4.3.3 $\mathcal{NP}$ úplné úlohy. Řekneme, že rozhodovací úloha $\mathcal{U}$ je $\mathcal{NP}$ *úplná*, jestliže

1. $\mathcal{U}$ je ve třídě $\mathcal{NP}$;
2. každá $\mathcal{NP}$ úloha se polynomiálně redukuje na $\mathcal{U}$.

Třída všech $\mathcal{NP}$ úplných úloh se značí $\mathcal{NPC}$.

Zhruba řečeno, $\mathcal{NP}$ úplné úlohy jsou ty „nejtěžší“ mezi všemi $\mathcal{NP}$ úlohami.

4.3.4 Tvrzení. Jsou dány dvě $\mathcal{NP}$ úlohy $\mathcal{U}$ a $\mathcal{V}$, pro které platí $\mathcal{U} \lhd_p \mathcal{V}$. Pak

1. jestliže $\mathcal{V}$ je ve třídě $\mathcal{P}$, pak také $\mathcal{U}$ je ve třídě $\mathcal{P}$;
2. jestliže $\mathcal{U}$ je $\mathcal{NP}$ úplná úloha, pak také $\mathcal{V}$ je $\mathcal{NP}$ úplná úloha.

4.3.5 Tvrzení. Kdyby některá $\mathcal{NP}$ úplná úloha patřila do třídy $\mathcal{P}$ (tj. byla by polynomiálně řešitelná), pak $\mathcal{P} = \mathcal{NP}$. Jinými slovy, každá $\mathcal{NP}$ úloha by byla polynomiálně řešitelná.

4.3.6 $\mathcal{NP}$ obtížné úlohy. Jestliže o některé úloze $\mathcal{U}$ pouze víme, že se na ní polynomiálně redukuje některá $\mathcal{NP}$ úplná úloha, pak říkáme, že $\mathcal{U}$ je $\mathcal{NP}$ těžká, nebo též $\mathcal{NP}$ obtížná. Poznamenejme, že to vlastně znamená, že $\mathcal{U}$ je alespoň tak těžká jako všechny $\mathcal{NP}$ úlohy.