1.7 Počítač s libovolným přístupem – RAM

1.7.1 V tomto oddílu zavedme 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.

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

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

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

1.7.5 Paměť je rozdělena na paměťové buňky, každá buňka může obsahovat celé číslo. Předpokádáme 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.*

1.7.6 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.

1.7.7 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.

1.7.8 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.

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

1.7.10 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.

1.7.11 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 $*j$.

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

1.7.13 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 $T(n)$ roven $\mathcal{O}(f(n))$.

1.7.14 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 $\mathcal{O}(g(n))$.

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

1.7.16 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 $\mathcal{O}(n^2)$.

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

1.7.18 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 1.7.17 simuluje n kroků programu P pomocí $\mathcal{O}(n^3)$ svých kroků.

1.7.19 Důsledek. Je dán program P pro RAM, který splňuje podmínky z věty 1.7.17. 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.

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

1.8.1 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 orinetovaném 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.

  Na příklad 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 #.

1.8.2 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 $\Sigma^\star \setminus L_{\mathcal{U}}$.

1.8.3 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 $p(n)$.

1.8.4 Příklady.

- Minimální kostra v grafu. Je dán neorinetovaný 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.

1.8.5 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.

1.8.6 Poznámka. V definici 1.8.3 jsme místo existence Turingova stroje mohli požadovat existenci programu P pro RAM, který řeší $\mathcal{U}$ v polynomíálním čase. Abychom přiblížili, které jazyky (rozhodovací úlohy) leží ve třídě $\mathcal{NP}$, zavedeme pojem nedeterministického algoritmu jako analogii RAM.

1.8.7 Nedeterministický algoritmus pracuje ve dvou fázích,

1. Algoritmus náhodně vygeneruje řetězec s.

2. Deterministický algoritmus (Turingův stroj, program pro RAM) na základě vstupu a řetězce s dá odpověď ANO nebo NEVIM.

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.

1.8.8 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 1.8.5 jsme místo existence nedeterministického Turingova stroje mohli požadovat existenci nedeterministického algoritmu, který řeší úlohu $\mathcal{U}$ v polynomiálním čase.

1.8.9 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

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