2. CVIČENÍ Z ADS 1

Viktor Němeček 25. 2. 2019

https://kam.mff.cuni.cz/~viki/vyuka/ads11819/

Na přednášce jste probrali mimo jiné výpočetní model RAM. Ten ve své mejčastěji používané podobě používá poměrnou logaritmickou cenu instrukce. To znamená, že vykonámí libovolné aritmetické instrukce nad čísly velkými n vás stojí $\lceil \log n / \log b \rceil$ času, kde b je velikost vstupu (co přesně je velikost vstupu záleží na konkrétním problému, můžete si představit, že je to například velikost největšího čísla, které se na vstupu objeví; všiměte si, že potom je cena instrukce omezená konstantou, dokud používáte čísla omezená polynomem v b, až u exponenciálně velkých čísel roste). Proč by měl model vypadat tak složitě?

Příklad 1. Představte si, že umíte v konstatním čase pracovat s libovolně velkými čísly. Zkuste vymyslet, jak v lineárním čase do jedné buňky zakódovat vektor tak, abyste ze zakódovaných vektorů byli schopni počítat skalární součin v čase $\Theta \log n$. Pomocí něj vyrobte algoritmus, pomocí nějž bude možné násobit matice $n \times n$ v čase $\Theta(n^2 \log n)$.

Příklad 2. Mějme přirozená čísla $a_1, \ldots, a_n < N$ a číslo $k > n \cdot N$. Dokažte, že

$$(a_1 k^0 + a_2 k^1 + \cdots + a_n k^{n-1}) \mod (k-1) = \sum_{i=1}^{n} a_i.$$

S tímto výsledkem zrychlete předchozí algoritmus na kvadratický.

Příklad 3. Vymyslete, jak libovolnou sekvenci celých (můžete nejprve zkusit přirozených) čísel zakódovat jednoznačně (včetně pořadí) do jednoho čísla C. Zkuste pomocí toho vytvořit postup, jak napsat libovolný program tak, aby zabíral jen konečně mnoho buněk paměti.

Dále si můžeme s RAMem chvíli hrát. Pro připomenutí, z aritmetických instrukcí máte sčítání, odčítání, celočíselné dělení, násobení, modulo a unární mínus, instrukci přiřazení, logické instrukce bitové AND, OR, XOR a posuny doprava a doleva, a nakonec instrukci `halt`, která zastaví program, instrukci `goto`, která skočí na specifikované návěstí (které můžete též umisťovat, každé z nich musí být unikátní, nejedná se alev pravém smyslu slova o instrukci) a podmíněnou instrukci `if` *podmínka* `then` *instrukce*, kde podmínka může být rovnost nebo nerovnost dvou čísel ($=, <>, <, >, <=, >=$), a instrukce libovolná z uvedených instrukcí kromě podmíněné instrukce. Operandy všech instrukcí mohou být konstanty, čísla z buněk na dané adrese, nebo čísla z buněk, jejichž adresa je napsána v buňce na dané adrese. Podmínka tedy může být například `5>[10]`, ne však třeba `5<[10]+3`. Dále si dovolíme i pojmenované proměnné, což bude ale pouze syntaktická zkratka za nějakou přímo adresovanou buňku, kterou v programu jinak nepoužíváme.

Čtení vstupu a výstupu na RAMu typicky neřešíme a předpokládáme, že vstup je na nějaké předem dané adrese v paměti

Příklad 4. Vymyslete, jak na RAMu prohodit obsah dvou buněk, aniž bychom využili libovolnou jinou buňku.

Příklad 5. Napište následující program na RAMu. Kolik instrukcí se celkem provede?

```
1       result := 0
2       for i := 1 to n
3         for j := i to n
4           result := result + 1
```

Příklad 6. Napište na RAMu program, který zjistí, zda je dané číslo mocnina dvojky.

∗ **Příklad 7.** Najděte program řešící předchozí příklad, který vždy doběhne v konstantním čase.

Příklad 8. Rozmyslete si, jak na RAMu napsat program, který vyžaduje více různých polí, jejichž velikost při psaní kódu neznáte.

Příklad 9. Rozmyslete si, jak na RAMu naimplementovat rekurzi (včetně lokálních proměnných). Pomocí vaší konstrukce by měl jít naprogramovat například rekurzivní algoritmus na počítání n-tého Fibbonaciho čísla.

Příklad 10. Rozmyslete si, jak rozšířit model RAM tak, aby mohl komunikovat s uživatelem během svého běhu. Napište program, který bude hádat číslo z vhodného rozsahu (například 1 až 1024), které si uživatel myslí, pomocí optimálního počtu otázek.