# Cvičení Programování I

Cvičící: **Pavel Surynek, KTIML,**
**pavel.surynek@seznam.cz**
Semestr: **Zima 2005/2006**
Kroužek: **Matematika/59**
Rozvrh: **Pátek 10:40-12:10 (učebna K2)**

## Stručné poznámky ke cvičení ze 4.11.2005

**1. Organizační záležitosti.** Na příštím cvičení, tj. 11.11. se bude psát krátká písemka. V písemce bude nějaká úloha na logickou úvahu a úloha na napsání jednoduchého programu v Pascalu.

Jakékoli **dotazy** ke cvičení lze posílat na uvedenou e-mailovou adresu. Osobní konzultace ke cvičení lze dohodnout e-mailem (alespoň den předem).

**2. Nenažraný velbloud.** Velbloud stojí na poušti u hromady banánů (tentokrát je to normální hromada v běžném slova smyslu, ne ta z předchozí úlohy). Banánů je 3000. Úkolem je dopravit co nejvíce banánů do 1000 km vzdálené oázy. Velbloud uveze nejvýše 1000 banánů. Problém je, že velbloud má jistou spotřebu, jmenovitě jeden banán na jeden kilometr. Cestou si velbloud může kdykoli jakýkoli počet banánů odložit a potom se k nim vrátit.

> **Řešení.** Následující řešení bylo předvedeno na cvičení. Velbloud je efektivně využit, když je plně naložen, tj. když si před tím, než vyrazí, naloží 1000 banánů. Předpokládejme, že si velbloud udělá zastávku, kde si bude vytvářet novou hromadu banánů (to je nutné jinak do oázy nic nedoveze). Velikost hromádky na zastávce musí být dělitelná 1000, jinak by v dalších krocích nebylo možno velblouda plně naložit a byl by využit neefektivně. Na hromádce tedy musí být 1000 nebo 2000 banánů. Hromádku o velikosti 1000 banánů je možno vytvořit ve vzdálenosti 400 km, velbloud vyrazí z počátku třikrát plně naložen, cestu k hromádce projde celkem pětkrát, což odpovídá spotřebě 2000 sežraných banánů. Potom řešíme úlohu, jak dopravit co nejvíc z 1000 banánů do 600 km vzdálené oázy, to už se vyřeší rovnou a vyjde, že velbloud do oázy dopraví **400 banánů**. Druhá možnost je, že si velbloud vyrobí hromádku s 2000 banánů. Takovou hromádku lze nahromadit ve vzdálenosti 200 km, velbloud vyrazí třikrát plně naložen, cestu k hromádce projde celkem pětkrát, což odpovídá 1000 sežraných banánů. Potom řešíme úlohu, jak dopravit 2000 banánů do 800 km vzdálené oázy. Opět bude nutné dopravovat banány se zastávkou, jinak nedostaneme lepší řešení než již nalezené. Na zastávce si velbloud nahromadí 1000 banánů. Druhou hromádku je možno vytvořit ve vzdálenosti 333 km od první zastávky, velbloud vyrazí dvakrát plně naložen od první hromádky, cestu přitom projde třikrát, což odpovídá 999 sežraných banánů. Zbývá vyřešit úlohu, jak dopravit co nejvíc z 1001 banánů do 467 km vzdálené oázy. To se vyřeší už rovnou a vyjde, že velbloud do oázy doveze **533 banánů** (jeden banán zůstane na druhé zastávce). To je lepší řešení než v případě, kdy je velikost první hromádky 1000 banánů.

**3. Třídění pomocí haldy.** Pomocí datové struktury halda je možné třídit data podle velikosti. Předpokládejme, že máme $N$ čísel, které chceme pomocí hlady seřadit podle velikosti. Jak se to dělá, už víme. Vaším úkolem je nyní spočítat počet kroků, resp. počet elementárních operací, kterých je k tomu zapotřebí (elementární operací se přitom myslí například porovnání dvou prvků, nakreslení spojnice mezi prvky atd.). Počet operací odhadněte také asymptoticky.

> **Řešení. a) Vytváření haldy.** V první fázi vytváříme haldu tak, že do původně prázdné hlady postupně vkládáme zadané prvky. Vložení prvku do haldy o velikosti $K$ spotřebuje jednu operaci na vložení prvku do poslední hladiny haldy (vytvoření spojnice) a nejvýše $\log_2 K$ operací na opravu podmínky na uspořádání pro nově vložený prvek (při opravě podmínky na

uspořádání se provádí nejvýše jedna oprava pro jednu hladinu, hladin je $\log_2 K$ ). Celkem je tedy na vytvoření haldy potřeba $\sum_{K=1}^{N} 1 + \log_2 K \leq \sum_{K=1}^{N} 1 + \log_2 N \leq N + N \log_2 N$ .

**b) Zatřiďování nejmenších prvků.** Na utržení minima z haldy spotřebuje jednu operaci. Přesun náhradního prvku do kořene haldy je také jedna operace (3, když počítáme zrušení a vytváření spojnic). Další operaci spotřebuje zařazení prvku do výsledné posloupnosti. Poté je potřeba opravit podmínku na uspořádání kvůli novému kořenu, na to je potřeba nejvýše $\log_2 K$ operací, kde $K$ je velikost haldy. Provádí se nejvýše jedna oprava pro jednu hladinu v haldě, přičemž hladin je $\log_2 K$. Celkem je tedy na zatřiďování potřeba nejvýše $\sum_{K=N}^{1} 3 + \log_2 K \leq \sum_{K=N}^{1} 3 + \log_2 N \leq 3N + N \log_2 N$ .

**Závěr.** Na setřídění $N$ prvků s použitím datové struktury halda je zapotřebí nejvýše $3N + 2N \log_2 N$ operací. $3N + 2N \log_2 N$ je $O(N \log_2 N)$. Označme $P(N)$ skutečný počet operací na setřídění $N$ prvků pomocí haldy, platí, že $P(N) \leq 3N + 2N \log_2 N$ , samozřejmě také platí $P(N) \in O(N \log_2 N)$ . To jsme jen symbolicky zapsali dokázané tvrzení o počtu kroků algoritmu. Platí, že $P(N) \in \Theta(N \log_2 N)$ ? To neplatí, protože $P(N)$ může být někdy hodně malé (v případě, že se neprovádí žádná oprava uspořádání) a nelze tedy najít takovou konstantu $c$ , aby $P(N) \geq cN \log_2 N$ pro skoro všechna $N$ .

**4. Parciální správnost algoritmu.** Ověřování správnosti (korektnost) algoritmů je obtížné, někdy neproveditelné. Z tohoto důvodu se často ukazuje jen tzv. parciální správnost algoritmů. Algoritmus je parciálně správný, jestliže platí, že když algoritmus skončí, tak vydá správný výstup (předpokládá se, že správnost výstupu dokážeme ověřit). Dokažte, že třídění pomocí haldy je parciálně správný algoritmus.

**Řešení.** Parciální správnost algoritmu budeme dokazovat sporem. Nechť algoritmus vydal nesprávný výstup. Jak vypadá nesprávný výstup ? Nesprávný výstup buď neobsahuje všechny tříděné prvky, v tomto případě dostáváme ihned spor s tím, že algoritmus skončí, až když je halda prázdná. Nebo výsledná posloupnost obsahuje všechny prvky, ale špatně setříděné. Uvažujme, že jsme původně třídili prvky 55, 31, 4, 27, 99, 42 a 2 a nechť algoritmus vydal posloupnost 2, 4, 27, 42, 55, 31, 99 (správný výsledek by byl 2, 4, 27, 31, 42, 55, 99). Uvažujme okamžik, kdy do výsledné posloupnosti algoritmus vkládá číslo 42. V tomto okamžiku halda obsahuje prvky 42, 55, 31 a 99. Jestliže bylo do výsledné posloupnosti vloženo číslo 42, muselo se tomto okamžiku nacházet na vrcholu haldy, v haldě se ale také nacházelo číslo 31, které není v kořeni. To je ale spor s podmínkou na uspořádání haldy.

**5. Konečnost algoritmu.** Konečnost algoritmů se ověřuje tak, že jednotlivé stavy algoritmu ohodnotíme - například přirozenými nebo nezápornými reálnými čísly a snažíme se dokázat, že toto ohodnocení v průběhu algoritmu klesá.

**a)** Dokažte, že Euklidův algoritmus na nalezení největšího společného dělitele dvou čísel je konečný.

**b)** Uvažujme dva algoritmy, jejichž stavy jsou ohodnoceny reálnými čísly. Oba algoritmy končí při ohodnocení 0. Pro ohodnocení stavů $h : S \to R$, kde $S$ je množina stavů, prvního algoritmu splatí, že

$\exists \delta \in R, \delta > 0$, že $\forall i \quad h(stav(i)) \geq h(stav(i+1)) + \delta$, kde $stav(i)$ je stav algoritmu v kroku $i$ . Pro druhý algoritmus platí, že

$\forall i \; \exists \delta \in R, \delta > 0$, že $h(stav(i)) \geq h(stav(i+1)) + \delta$, kde $stav(i)$ je stav algoritmu v kroku $i$ . Vyšetřete konečnost obou algoritmů.

**Řešení. a) Vsuvka:** Jak pracuje Euklidův algoritmus ? Mějme čísla $n$ a $m$, hledáme $NSD(m,n)$. Platí, že $NSD(m,n) = NSD(m, n-m)$, přičemž $n > m$. Dále platí, že $NSD(m,m) = m$. Algoritmus pracuje tak, že v každém kroku algoritmu hledáme

$NSD(m,n)$. Když jsou $n$ a $m$ různá, od většího z $n$ a $m$ odečteme menší $n$ a , jinak už máme $NSD(m,n)$ (tj. když jsou $n$ a $m$ stejná).

Stavem algoritmu jsou čísla $n$ a $m$. Nabízí se tedy ohodnocení $h((m,n)) = m + n$. Přirozená čísla mají nejmenší prvek, ohodnocení se v každém kroku zmenší aspoň o 1, algoritmus tedy skončí. **Rozmyslete !**

**b)** První algoritmus jistě skončí. Nechť je na začátku běhu algoritmu ohodnocení počátečního stavu $h(stav(0)) = K$. Jistě platí, že nejvýše po $\frac{K}{\delta}$ krocích algoritmus skončí. Ohodnocení se snižuje „stejnoměrně“ (viz. matematická analýza - stejnoměrná konvergence, spojitost atd.). Druhý algoritmus obecně nemusí skončit. Nechť například ohodnocení počátečního stavu je $h(stav(0)) = 2$ a nechť $\delta_i = \frac{1}{2^i}$ ($\delta$ je v každém kroku jiné, proto ten index). V takovém případě nemůžeme dokázat, že ohodnocení někdy klesne na 0, protože i po nekonečně mnoha krocích klesne ohodnocení aspoň o 1 ($\sum_{i=1}^{\infty} \delta_i = \sum_{i=1}^{\infty} \frac{1}{2^i} = 1$), což nemusí stačit.

**6. Jak dlouho může běžet program.** Je možné pro libovolné $t > 0$ napsat program, který běží více než $t$ vteřin a pak zastaví. Předpokládáme přitom, že počítač, na kterém program běží, splňuje běžná fyzikální omezení, tj. například provedení elementární operace trvá určitou dobu, velikost paměti je konečná atd.

**Řešení.** Bylo za DCV. Odpověď zní: **možné to není**. Jestliže má počítač konečnou paměť, řekněme $k$ bitů, pak existuje $2^k$ různých stavů, ve kterých se počítač (program) může nacházet. Elementární operace převádí počítač (program) z jednoho stavu do stavu následujícího. Jestliže se stane, že počítač (program) přejde do některého ze stavů, ve kterých již byl, pak můžeme říct, že program je zacyklený a poběží věčně. Nechť provedení elementární operace trvá $t_o$ vteřin. Když program běží déle než $t_o 2^k$, pak se jistě do některého ze stavů dostal podruhé, je tedy zacyklen a poběží už navěky. Požadujeme-li tedy, aby program běžel déle než $t_o 2^k$ vteřin a pak zastavil, není tomuto požadavku možno vyhovět.