# 1. úkol MI-PAA

Jan Jůna (junajan)
13.10.2013

## *Specifikaci úlohy*

Problém batohu je jedním z nejjednodušších NP-těžkých problémů. V literatuře najdeme množství jeho variant, které mají obecně různé nároky na algoritmus řešení.

Je dáno:

- celé číslo *n* (počet věcí)
- celé číslo *M* (kapacita batohu)
- konečná množina $V = \{v_1, v_2, \dots, v_n\}$ (hmotnosti věcí)
- konečná množina $C = \{c_1, c_2, \dots, c_n\}$ (ceny věcí)

V našem řešení se omezíme na verzi „0/1 problém batohu“, tedy zkonstruujeme takovou množinu $X=\{x_1, x_2, \dots, x_n\}$, kde každé $x_i$ je 0 nebo 1, tak, aby platilo:

$v_1x_1 + v_2x_2 + ... + v_nx_n <= M$ (aby batoh nebyl přetížen).

a výraz

$c_1x_1 + c_2x_2 + ... + c_nx_n$

nabýval maximální hodnoty pro všechny takové množiny (cena věcí v batohu byla maximální).

Více viz. https://edux.fit.cvut.cz/courses/MI-PAA/tutorials/batoh

## *Rozbor variant řešení*

Problém je řešitelný více způsoby. V našem měření jsme použili 2 způsoby řešení, které jsou popsané níže:

- Řešení problému batohu **hrubou silou** (tj. exaktně). Problém je řešený generováním permutací možných řešení a dále vybráním nejlepšího řešení. Toto řešení je časově i paměťové náročné. Časová složitost závisí na velikosti vstupních dat.
- Řešení problému batohu **heuristikou** podle poměru cena/váha. Při řešení problému je nejprve seřazeno pole vstupních dat podle poměru jejich ceny / váhy. Při plnění batohu jsou pak postupně vybírány předměty s nejlepším poměrem až do zaplnění baťohu. Toto řešení není závisle na vstupních datech tak moc, jako řešení hrubou silou – časová složitost s přibývajícími daty roste pomaleji než u řešení hrubou silou. Nemusí ale poskytovat nejlepší řešení problému.

## *Popis kostry algoritmu*

Implementace obou algoritmů – výpočet hrubou silou (naive.py) i heuristikou (ratio.py) podle

poměru cena/váha – byla provedena v programovacím jazyce Python verze 3.2. Společné funkce algoritmů byly uloženy do souboru common.py, který je importován do obou algoritmů.

### Algoritmus pro výpočet hrubou silou

Algoritmus (**naive.py**) nejprve načte podle parametrů název souboru se vstupními daty a případně i počet opakování, kolikrát se má výpočet provést. Poté jsou načtena vstupní data, ty jsou rozpársována a serializována pro další použití. Další zpracování přejímá funkce *processProblem* která vytvoří počáteční instanci výsledku, do které se dále bude doplňovat nejlepší řešení a nulový vektor o velikosti vstupních dat.

Rekurzivní funkce *iterateProblem* přejímá vstupní data a vektor, tvořený z jedniček a nul. Zařazení prvku do batohu pak určuje jednička na stejné pozici, jakou má prvek ve vstupních datech. Funkce vždy ověří, zda vnoření již nepřekročuje velikost vstupních dat. Pokud ne, otestuje zda současná konfigurace vektoru nepřesahuje maximální velikost baťohu. V takovém případě vrací aktuální nejlepší současnou konfiguraci. Poté ověří, zda konfigurace vstupního vektoru nemá lepší cenu než současná nejlepší, pokud ano, uloží ji a pokračuje na další vnořování.

Pokud rekurze projde všechny permutace problému – ve vstupním vektoru se změnily všechny konfigurační bity – vrátí se nejlepší naměřený výsledek uložený v proměnné *BEST_PART*.

Pomocí druhého nepovinného parametru programu se dá určit počet opakování výpočtu problému na jednu instanci. Tento počet se dá použít především u krátkých výpočtů, u kterých by byl problém s měřením času. Instance je tak spočítána vícekrát a naměřený čas je podělen počtem opakování

Na **stdout** se vytiskne se výsledek a do **stderr** se vloží ID záznamu, čas celého opakovaného řešení instance, počet opakování a čas řešení instance (celkový čas na instanci / počet opakování). Poté se přejde k další instanci problému.

### Algoritmus pro výpočet heuristikou podle poměru cena/váha

Algoritmus pro výpočet zadanou heuristikou (**ratio.py**) začíná stejně jako řešení hrubou silou a to načtením vstupních dat, jejich rozparsováním.

Oproti prvnímu algoritmu však před samotným výpočtem dojde k vypočtení poměru zadaných cen a váhy předmětů (cena/váha). Tento poměr je pak spolu s pořadovým číslem přidán do pole vstupních dat a toto pole je poté sestupně seřazeno podle vypočítáného poměru.

Algoritmus poté vybírá prvky s nejlepším poměrem a pokud součet jeho váhy a aktuální váhy baťohu nepřekročí maximální velikost, je prvek zařazen do baťohu. V opačném případě se rovnou přejde na další prvek v seřazených vstupních datech.

Pokud vstupní data již neobsahují žádný prvek k rozřazení, je zpracování ukončeno a zpět vrácen aktuální stav baťohu. V případě opakování se výpočet problému iteruje, v opačném případě je vytisknuto řešení.

## *Naměřené výsledky*

Algoritmus byl testován na počítači s procesorem Intel® Core™ i5-2540MSandy Bridge.

### Algoritmus pro výpočet hrubou silou

Pro měření brute force algoritmu byly použity vstupní data o velikosti 4,10,15,20,22,25 a 25 prvků. Časová složitost algoritmu na těchto datech je zobrazena v grafu níže.

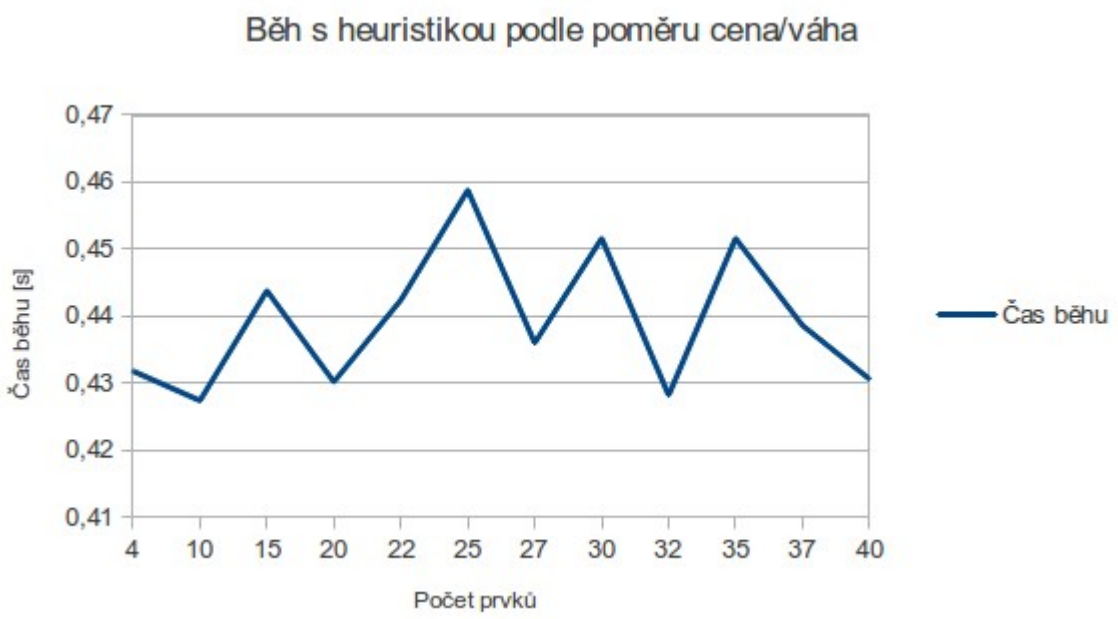

Data jsou také zobrazena v následující tabulce:

| | 4 | 10 | 15 | 20 | 22 | 25 | 27 |
|---|---|---|---|---|---|---|---|
| **Čas běhu [s]** | 0,0004 | 0,011 | 0,601 | 30,7544 | 158,4484 | 1363,56 | 5576,78 |

Výpočet hrubou silou byl nasazen na vstupní data o hodnotách 4 až 27 prvků. Podle grafu je vidět, že čas na výpočet těchto dat rostě exponenciálně vzhledem k velikosti vstupní množiny.

### Algoritmus pro výpočet heuristikou podle poměru cena/váha

Výpočet heuristikou byl taktéž otestován na zadaných vstupních datech. Kromě časové složitosti byla měřena i relativní chyba, kterou tento algoritmus má oproti řešení metodou brute force.




Data jsou také zobrazena v následující tabulce:

| | **4** | **10** | **15** | **20** | **22** | **25** | **27** | **30** | **23** | **25** | **27** | **40** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Čas běhu** | 0,4318 | 0,4274 | 0,4438 | 0,4302 | 0,4424 | 0,4588 | 0,436 | 0,4516 | 0,4282 | 0,4516 | 0,4386 | 0,4306 |

Pro řešení algoritmem využívajícím heuristiku vstupní množina neovlivňuje dobu výpočtu tak moc, jako při řešení hrubou silou. Po prvotním seřazení pole se vstupní data projdou pouze jednou.

Relativní chyba se pro maximalizační problémy počítá jako rozdíl ceny optima a ceny přibližného řešení podělený opět cenou přibližného řešení. Tedy: $\varepsilon = ( C(OPT)-C(APX) ) / C(OPT)$

- kde C(OPT) je cena optima
- a C(APX) je cena přibližného řešení

Naměřené odchylky jsou uvedeny v následujcí tabulce:

| **Počet prvků** | **Odchylka** |
|---|---|
| 4 | 0,02895 |
| 10 | 0,117586 |
| 15 | 0,031795 |
| 20 | 0,123561 |
| 22 | 0,143805 |

| | |
|---|---|
| 25 | 0,156953 |
| 27 | 0,151419 |
| 30 | 0,151552 |
| 32 | 0,139902 |
| 35 | 0,125734 |
| 37 | 0,096786 |
| 40 | 0,131717 |

Pro spočítání odchylky byl vytvořen script **countDiv.py**. Průměr časové složitosti je počítán pomocí scriptu **countTime.py**

## *Závěr*

Z naměřených dat vyplývá, že první řešení problému, tj. řešení hrubou silou, je silně závislé na velikosti vstupních dat. Pro najití problému je potřeba prohledat celý stavový prostor, který s přibývajícími daty velmi rychle roste.

Druhý algoritmus používá seřazené pole, které prochází za účelem přiřazení prvků do baťohu. Odpadá zde procházení celého stavového prostoru a řešení je tak mnohem rychlejší než v prvním případě. Nevýhodou je pak možnost dosažení pouze suboptimálního výsledku.

## *Popis zdrojových kódů*

**Slozky:**

inst - obsahuje instance problemu

sol - vzorove reseni problemu

mytime_ratio - soubor s casy behu algoritmu s heuristikou

mytime_naive - soubor s casy behu algoritmu hrube sily

mysol_ratio - soubor s vysledky algoritmu s heuristikou

mysol_naive - soubor s vysledky algoritmu hrube sily

**Soubory:**

common.py - soubor obsahujici spolecne funkce

naive.py - implementace algoritmu hrubou silou

ratio.py - implementace algoritmu s heuristikou

magicrun_ratio - spoustec pro algoritmus s heuristikou - vytvori slozky, spusti script

magicrun_naive - spoustec pro algoritmus hrube sily - vytvori slozky, spusti script

countDiv.py - spocita prumernou odchylku souboru v zadane slozce oproti slozce s vysledky
- spusteni napr.: ./countDiv.py sol mysol_ratio/

countTime.py - spocita prumerny cas behu algoritmu pro vysledne casy jednotlivych instanci
- spusteni napr.: ./countTime.py mytime_ratio/*

## *Zdroje*

Popis problému baťohu: https://edux.fit.cvut.cz/courses/MI-PAA/tutorials/batoh

Zdrojové kódy: http://honza.dreamware.cz/Batoh.zip