MI-PAA: 1. Řešení problému batohu metodou hrubé síly a jednoduchou heuristikou

Vojtěch Mráz

21. října 2018

1 Úvod

Problém batohu je jednoduchý matematický úkol, kdy se snažíme do našeho batohu, o hmotnostní kapacitě n, naložit co nejvíce předmětů z dané množiny M. Každý tento předmět má svojí hmostnost a svojí cenu.

Možnosti řešení a interpretace úkolu se mohou lišit ve způsobu vkládání předmětů do batohu, avšak nikdy bychom neměli přesáhnout požadovanou hmotnost n.

V tomto úkolu si ukážeme možné řešení a porovnáme je spolu z pohledu času a relativní chyby, jež může vzniknout při rozdílné volbě implementace.

2 Zadání úlohy

1. Naprogramujte řešení 0/1 problému batohu **hrubou silou** (tj. nalezněte skutečné optimum). Na zkušebních datech pozorujte závislost výpočetního času na n.
2. Naprogramujte řešení problému batohu **heuristikou** podle poměru cena/váha. Pozorujte
   - závislost výpočetního času na n. Grafy jsou vítány (i pro exaktní metodu)
   - průměrnou a maximální **relativní chybu** (tj. zhoršení proti exaktní metodě) v závislosti na n.

3 Možná řešení

Jak jsem již napsal výše, zadaný problém je možné řešit více způsoby.

Nejuchopitelnějším řešením, které vždy vrátí nejlepší možnou cenu kombinace předmětů je vyzkoušením všech možných možností, tedy hrubou silou. Bohužel čas vypracování úlohy tímto zpusobem se zvedá exponenciálně a už pro jen trochu větší počet předmětů v množině je čas dosti vysoký. Více v sekci 5.

Pro zlepšení časové a paměťové náročnosti je možno problém batohu řešit pomocí heurestik (pravidlo, podle kterého vkládáme předměty do batohu). Některé z heurestik:

- podle klesající ceny
- podle rostoucí hmotnosti
- podle klesajícího poměru cena/hmotnost (více v sekci 5)

Využití heurestiky ale nemusí vždy dávat správný výsledek. Tento rozdíl ve výsledku nazýváme relativní chybou. Jak se ale dočteme dále, použití heurestik se stále vyplatí.

4 Rámcový popis řešení

Implementace problému byla napsána v jazyce C# ve frameworku .NET Core (multiplatformní).

Programu je předám argument souboru se specifickým formátem, který obsahuje instance problému (v našem případě přesně 50). Jednotlivé instance uloží do pole struktur, kde každá struktura reprezentuje jednu instanci a tím tedy její *Id, počet předmětů, kapacita batohu a jednotlivé předměty.*

V progamu je definovaná konstanta $NUMBER\_OF\_RUNS$, která v našem případě byla ve většině případech nastavena na 2000 (jeden běh znamená projití všech instancí ze souboru právě jednou) a vytvoří pole časů algoritmu hrubé síly a algoritmu heurestiky. Dodatečně si ještě vytvoří pole řešení pro oba algoritmy (slouží pro výpočet průměrné a maximální relativní chyby).

Následně spustí algoritmus hrubé síly a uloží výsledek každé instance, stejně tak čas, ve kterém byly spočteny všechny instance souboru (po prvním běhu se již výsledky znova nepřepisují, jelikož jsou stejné).

Stejný postup se uskuteční i pro heurestické řešení.

V poslední řadě program prochází jednotlivé výsledky obou algoritmů, z kterých následně vypočítává relativní chybu. Pokud je zrovna vypočítaná chyba tou největší, uloží ji do proměnně $maxMistake$, tedy maximální chyby. Všechny relativní chyby se sečtou a vydělí počtem runů ($NUMBER\_OF\_RUNS$).

Output programu vypadá následovně (relativní chyby se přepisují na procentuální reprezentaci s třemi desetinnými místy):

```
Console.WriteLine("\nNumber of runs: " + NUMBER_OF_RUNS);
Console.WriteLine("BF Avg Time: " + allBfTimes / NUMBER_OF_RUNS);
Console.WriteLine("PW Avg Time: " + allPwTimes / NUMBER_OF_RUNS);
Console.WriteLine("Max Mistake: " + Math.Round(maxMistake * 100, 3));

Console.WriteLine("Avg Mistake: " +
        Math.Round((double) (avgMistake / instances.Count) * 100, 3));
```

5 Popis algoritmu

V této části popisuju konkrétní implementaci řešení. Algortimus podle poměru cena/váha již jasně definuje postup řešení, ale algoritmus hrubou silou je stále možné naprogramovat více způsoby. Například pomocí dvou *for* cyklů nebo stromem a rekurzí, kdy každý root/node má dvě větve/potomky a to takové, kdy předmět byl do batohu vložen a kdy ne. Pro tuto úlohu jsem zvolil druhou kombinaci.

Součástí popisů jsou tabulky, znázorňující časovou závislost na počtu předmětů.

Metoda hrubé síly

Algoritmus této metody prochází všechny možné kombinace vložení předmětů do našeho batohu. Pokud si tedy přetransformujeme naší množinu předmětů M na množinu, která obsahuje pouze 0 či 1, například {0,1,0} (což znamená, že máme 3 předměty a do batohu jsme vložili pouze druhý), tak hledáme všechny podmnožiny, tedy: {0,0,0}, {0,0,1}, {0,1,0}, {0,1,1}, {1,0,0}, {1,0,1}, {1,1,0} a {1,1,1}.

Celkový počet podmnožin množiny n předmětů je 2^n. Program musí projít každou z těchto podmnožin (o nejvýše n prvcích) a vypočítat její hodnotu jako součet cen předmětů. Z tohoto důvodu je časová složitost $O(n \cdot 2^n)$.

n	Průměrný čas (ms)	Počet běhů
4	0,0376	2000
10	2,1347	2000
15	88,3177	2000
20	2654,4656	200
22	10023,942	200
25	86089,754	10

Tabulka 1: Průměrný čas pro 50 instancí a n předmětů (hrubá síla)

V tabulce 1 je znázorněna časová náročnost této metody. Časová náročnost byla měřena jako procesorový čas, který je potřebný k výpočtu všech 50 instancí (při měření jedné instance se čas přibližoval nule).

Z grafu je patrný exponenciální růst času pro zvyšující se počet předmětů.

Z důvodu velkého nárustu časové složitosti nebyl program testován na veškerých vstupech. Pro vstupy s více než 25 předměty nebylo možné změřit odpovídající počet běhů pro odhadnutí průmerného času, proto se v tabulce nevyskytují.

Heuristika podle poměru cena/váha

Implementace algoritmu probíhá ve dvou krocích:

1. nejdříve se pole předmětů seřadí sestupně podle poměru cena/váha,
2. jako druhý krok se předměty s největším poměrem přidávají do batohu, dokud není překročena kapacita batohu, předměty s vyšším poměrem však mohou být přeskočeny, když už se nevejdou.

Jelikož jsem uložil předměty do pole dvojic pomocí Listu, mohl jsem jednoduše zavolat C# funkci spolu s LinQ knihovnou *OrderByDescending(o => o.Item2 / o.Item1)*. Tato funkce vrací nový List již seřazených předmětů. V druhém kroku jsem pouze procházel for cyklem list a přidával do batohu.

Časová složitost tohoto algoritmu je v průměru $O(n \cdot \log n)$.

n	Průměrný čas (ms)	Počet běhů
4	0,0363	2000
10	0,0781	2000
15	0,1200	2000
20	0,1472	2000
22	0,1759	2000
25	0,1769	2000
27	0,2010	2000
30	0,2332	2000
32	0,2347	2000
35	0,2473	2000
37	0,2721	2000
40	0,3182	2000

Tabulka 2: Průměrný čas pro 50 instancí a n předmětů (heurestika)

Tabulka 2 znázorňuje časovou složitost výpočtu za pomoci heuristiky. Časová náročnost má pro rostoucí počet předmětů linární charakter. Z tabulky lze vyčíst, že časová náročnost je mnohem nižší než u exaktního výpočtu v předchozím případě.

Relativní chyba výpočtu

Jak již bylo popsané výše, heurestické řešení problému vnáší do řešení možnost nepřesného výsledku. Tuto nepřenost jsme nazvali relativní chybou řešení, tedy procentuální zvýraznění odchylky od optimálního řešení.

Tabulka 3 znázorňuje průměrnou a maximální odchylku pro 50 instancí.

n	Průměrná relativní chyba (%)	Maximální relativní chyba (%)
4	1,938	24,74
10	1,429	11,48
15	0,492	8,543
20	0,541	8,434
22	0,686	7,229
25	0,715	3,679
27	0,782	10,602

Tabulka 3: Průměrné a maximální relativní chyby uvedené v procentech)

6 Závěr

Na naměřených hodnotách časové složitosti lze usoudit správnost předpokladů z úvodu zprávy. Výpočet pomocí heuristiky je časově mnohem méně náročný a je tedy možné s ním zkoumat instance s větším počtem vstupních proměnných.

Oproti metodě hrubou silou ale nemáme zaručeno, že výsledek heuristického algoritmu je správný. Tabulka 3 znázorňuje, že maximální odchylka může být až v řádu desítek procent, tedy řešení je naprosto mylné.

Z tabulky lze ale také vyčíst, že průměrné odchylky jsou vzhledem k těm maximálním nepatrné (jelikož v hodně případech heurestické algoritmus vypočítal také správné řešení).

Pokud tedy nehledáme přesné řešení, můžeme potrvdit tvrzení, že řešit problém batohu pomocí heurestiky se vyplatí.