1 Zadání úlohy

Celé zadání se skládá ze dvou částí:

1. Naprogramujte řešení problému batohu hrubou silou (tj. exaktně). 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.
   - průměrnou a maximální relativní chybu v závislosti na n.

2 Rozbor možných variant řešení

Problém batohu lze řešit hrubou silou tak, že se nechají vygenerovat všechny možné kombinace výskytu jednotlivých položek v batohu. Následně se spočítá, která z vygenerovaných kombinací vyhovuje našim kritériím (největší cena, suma hmotnosti položek v batohu nepřekročí jeho nosnost).

Pro řešení problému batohu heuristikou se nejdříve seřadí položky, které chceme do batohu vkládat. Položky můžeme seřadit podle těchto kritérií:

- podle klesající ceny,
- podle rostoucí hmotnosti,
- podle klesajícího poměru cena/hmotnost.

Seřazené položky poté zkoušíme vkládat do batohu. Pokud se přidávaná položka do batohu nevejde, pokračujeme s další položkou v řadě.

3 Popis postupu řešení

V první řadě jsem se zaměřil na řešení problému batohu hrubou silou, díky kterému jsem získal přesné výsledky problému pro daný batoh a položky. Kvůli velké časové komplexitě algoritmu jsem nebyl schopný získat výsledky pro všechny instance problému. Poté jsem sestavil tři algoritmy pro řešení problému heuristikou. Na základě těchto algoritmů jsem získal datové soubory s vypočtenými výsledky. Pro zpracování těchto dat a následný výpočet průměrných a maximálních relativních chyb jsem vytvořil nový modul.

4 Kostra algotritmu

Modul, který řeší problém batohu přijímá 4 parametry:

- datový soubor s počátečními hodnotami,

- jméno algoritmu, kterým budeme problém řešit,
- zdali chceme, aby modul počítal čas, který algoritmus stráví na CPU,
- kolik iterací daného algoritmu budeme chtít pro výpočet průměrného času.

Po přečtení vstupních parametrů se načtou data ze souboru do pole. Podle naší volby proběhne jeden ze 4 algoritmů. Výstupy tohoto algoritmu jsou dále zapsány do souboru.

4.1 Problém batohu hrubou silou

Pro vygenerování všech možných kombinací výskytu položky v batohu jsem použil for cyklus, který má 2^n iterací. V každé iteraci se do pole primitivního typu int zapíše jednička na místa, která jsou reprezentována binární podobou čísla probíhající iterace. Takto vzniklou kombinaci položek otestuji, zdali nepřesáhla maximální nosnost batohu. Pokud ano, pokračuji další kombinací. Pokud ne, porovnám aktuální hodnotu položek s největší předchozí hodnotou. Pokud je větší, zapamatuji si tuto novou maximální hodnotu a kombinaci položek, které ji vytváří. Po skončení for cyklu získám nejlepší kombinaci a její hodnotu.

4.2 Problém batohu heuristikou

Pro výpočet řešení nejdříve sestavím pole, které obsahuje položky vkládané do batohu a jejich pozice v původním poli. Následně toto pole seřadím podle kritérií, které vyplývají z vybraného typu algoritmu:

- podle poměru cena/váha položky,
- podle klesající ceny položky,
- podle rostoucí hmotnosti položky.

Z takto seřazeného pole zkouším postupně přidávat položky do batohu. Pokud se položka do batohu již nevejde, pokračuji následující položkou v poli. Výpočet končí pokud jsme prošli celé pole nebo pokud jsme se dostali na maximální nosnost batohu.

4.3 Výpočet relativní chyby

Pro měření relativní chyby jsem vytvořil separátní modul, který má na vstupu dva parametry:

1. soubor s naměřenými výsledky,
2. soubor se správnými výsledky.

Algoritmus přečte oba soubory a vytvoří pole relativních chyb $((opt(n)apx(n))/opt(n))$ veškerých instancí obsažených v souboru. Následně vypočítá průměrnou relativní chybu, maximální relativní chybu a tyto hodnoty zapíše do souboru. Pro spouštění modulu se všemi soubory jsem vytvořil skript.

4.4 Měření času

Abych co možná nejlépe změřil čas, který algoritmus stráví na CPU, použil jsem java třídu ThreadMXBean. Pokud chci změřit čas, dám modulu na vstupu parametr kolikrát chci daný algoritmus zopakovat. Po každé iteraci změřený čas zapíšu do pole a po skončení všech iterací hodnoty v poli zprůměruji. Výsledek zapíši do souboru.

5 Naměřené výsledky

Pro implementaci jsem využil programovací jazyk Java. Testy běžely na operačním systému macOS.

5.1 Měření chyb

Na základě naměřených hodnot zobrazených v tabulkách 1 2 3 lze vidět, že heuristika podle poměru cena/váha se nejvíce blíží svými hodnotami k hodnotám vygenerovaných exaktní metodou. O něco horší je potom heuristika podle klesající ceny a úplně nejhorší je heuristika podle rostoucí váhy, kde průměrná chyba dosahuje 9 %. Což se blíží průměru maximálních chyb u heuristiky podle poměru.

5.2 Měření času

Exaktní metoda má časovou složitost exponenciální. Je tomu tak, neboť v algoritmu generujeme 2^n možností. Zato algoritmy založené na heuristice pracují v čase $n * \log(n)$. To je dáno potřebou seřadit pole položek podle stanovených kritérií. Veškeré hodnoty jsou zobrazeny v tabulce 4.

6 Závěr

Řešení problému batohu hrubou silou přináší exaktní výsledky. Nicméně běh algoritmu spotřebuje velké množství času. Heuristické algoritmy vrací nepřesné výsledky, ale poskytnou je v rozumném čase. Pokud ovšem zvolíme kvalitní heuristiku, například poměr cena/váha, získáme řešení v čase $n * \log(n)$ a navíc bude relativní chyba výsledků velmi nízká.

	Poměr cena/váha	
Počet položek	Průměrná chyba	Maximální chyba
4	2,175 %	36,364 %
10	1,286 %	11,480 %
15	0,476 %	8,543 %
20	0,600 %	8,434 %
22	0,687 %	7,229 %
25	0,498 %	3,679 %
27	0,502 %	10,602 %
30	0,507 %	5,514 %
32	0,341 %	3,341 %
35	0,280 %	4,609 %
37	0,344 %	8,197 %
40	0,200 %	2,337 %
Průměr	0,658 %	9,194 %

Table 1: Relativní chyba u řešení heuristikou založené na klesajícím poměru cena/váha

	Váha	
Počet položek	Průměrná chyba	Maximální chyba
4	11,072 %	65,613 %
10	9,717 %	36,345 %
15	4,193 %	20,481 %
20	8,253 %	22,859 %
22	10,713 %	20,436 %
25	9,061 %	21,846 %
27	8,864 %	15,595 %
30	8,975 %	16,922 %
32	9,438 %	19,545 %
35	9,486 %	17,599 %
37	10,196 %	21,445 %
40	10,072 %	21,473 %
Průměr	9,170 %	25,013 %

Table 2: Relativní chyba u řešení heuristikou založené na rostoucí váze

	Cena	
Počet položek	Průměrná chyba	Maximální chyba
4	1,419 %	24,765 %
10	1,717 %	13,977 %
15	0,177 %	5,532 %
20	1,316 %	8,378 %
22	2,827 %	12,475 %
25	2,114 %	13,170 %
27	2,073 %	11,579 %
30	1,511 %	7,741 %
32	1,783 %	8,795 %
35	1,500 %	8,136 %
37	1,202 %	4,390 %
40	2,058 %	9,034 %
Průměr	1,641 %	10,664 %

Table 3: Relativní chyba u řešení heuristikou založenou na klesající ceně

Počet položek	Hrubá síla	Klesající poměr cena/váha	Rostoucí váha	Klesající cena
4	1,84 μs	1.06 μs	0.93 μs	1.00 μs
10	2,231 ms	2.10 μs	1.22 μs	1.28 μs
15	103,65 ms	2.41 μs	3.16 μs	3.41 μs
20	6,22 s	5.82 μs	4.30 μs	5.25 μs
22	20,87 s	5.97 μs	5.48 μs	6.36 μs
25	4,91 min	6.19 μs	5.74 μs	7.07 μs
27	22,74 min	6.47 μs	6.97 μs	9.61 μs
30	1,02 h	7.87 μs	7.94 μs	10.04 μs

Table 4: Srovnání doby běhu algoritmů