Adrian Frydmański 209865 Data oddania: **22 XII 2015r.**Dawid Gracek 209929 Prowadzący: **dr inż. Jarosław Mierzwa**

Projektowanie efektywnych algorytmów  
– laboratorium

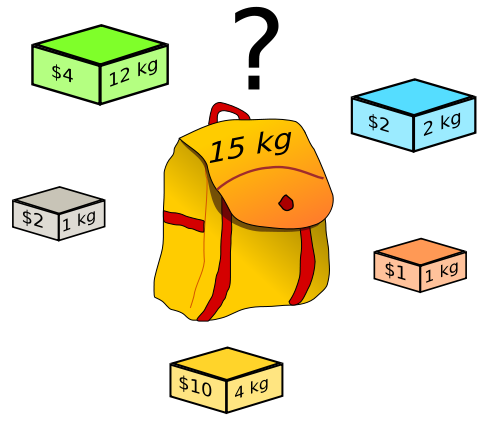
**Temat: „Rozwiązywanie problemu plecakowego algorytmem dynamicznym”**

# Zadanie

Należy zaimplementować algorytm dynamiczny dla optymalizacyjnego problemu plecakowego oraz dokonać testów polegających na pomiarze czasu działania algorytmu w zależności od wielkości instancji.

# Informacje wstępne

Dyskretny problem plecakowy (ang. discrete knapsack problem) jest jednym z najczęściej poruszanych problemów optymalizacyjnych, którego nazwa pochodzi od maksymalizacyjnego problemu wyboru przedmiotów, tak by ich sumaryczna wartość była jak największa i jednocześnie mieściły się w plecaku. Przy podanym zbiorze elementów o podanej wadze i wartości, należy wybrać taki podzbiór by suma wartości była możliwie jak największa, a suma wag była nie większa od danej pojemności plecaka.

  
Rysunek 1 Graficzne przedstawienie problemu

Problem plecakowy często przedstawia się jako problem złodzieja rabującego sklep – znajduje on N towarów; i–ty przedmiot jest wart *ci* oraz waży *wi*. Złodziej chce zabrać jak najwartościowszy łup, przy czym nie może zabrać więcej niż *B* kilogramów. Nie może też zabierać ułamkowej części przedmiotów, gdzie jest to możliwe w ciągłym problemie plecakowym.

# Opis działania algorytmu

Niech *w1*,... , *wn* oznaczają wagę elementów oraz *c1*, ..., *cn* ich wartości. Algorytm maksymalizuje wartość elementów, przy zachowaniu sumy ich wagi mniejszej bądź równej *W*. Niech *A*(*i*, *j*) będzie największą możliwą wartością, która może być uzyskana przy wadze nie większej od *j* i wykorzystaniu pierwszych *i* elementów. *A*(*n*,*W*) jest rozwiązaniem problemu.

Funkcję *A*(*i*,*j*) definiowana jest rekurencyjnie:

* *A*(0, *j*) = 0
* *A*(*i*, 0) = 0
* *A*(*i*, *j*) = *A*(*i* – 1, *j*) jeśli wi > *j*
* *A*(*i*, *j*) = max(*A*(*i* – 1, *j*), ci + *A*(*i* – 1, *j* – wi)) jeśli wi ≤ *j*

Rozwiązaniem problemu jest wynik dla A(*n*,*W*). Do efektywnego wykonania algorytmu używa się tablicy do zapamiętywania obliczonych podproblemów. Złożoność obliczenia wynosi O(*nW*), podobnie jak złożoność pamięciowa.

Pseudokod jest widoczny poniżej. Tablica A[1..n,0..W] przechowuje wyniki, wagi znajdują się w tablicy w[1..n], wartości w c[1..n], rozwiązanie znajduje się w komórce A[n,W].

for i = 0 to n do

A[i,0] = 0

for j = 0 to W do

A[0,j] = 0

for i = 1 to n do //rozważanie kolejno i pierwszych przedmiotów

for j = 0 to W do

if ( w[i] > j ) then //sprawdzenie, czy i-ty element mieści się w plecaku o rozmiarze j

A[i,j] = A[i-1,j]

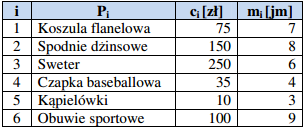
else

A[i,j] = max(A[i-1,j], A[i-1,j-w[i]] + c[i])

# Przykład wykonania algorytmu

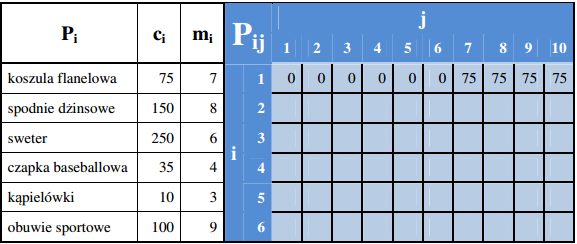
Dysponujemy następującą konfiguracją problemu:

Tabela 1. Dane przedmiotów



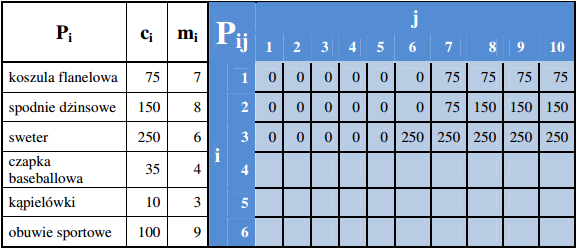
Maksymalna pojemność plecaka ***Mmax*** = 10 jm (jednostek miary).   
Zakładamy, że do plecaka możemy wsadzić dany przedmiot tylko raz. Co nie ma wpływu na generowanie tabeli. Dopiero przy odczytywaniu rozwiązania z niej będzie to ważne.  
***Pij*** jest wartością optymalną plecaka dla ***j****-tej* pojemności plecaka oraz dla przedmiotów mieszczących się w indeksach od 1 do ***i****.*W pierwszym wierszu możemy wypełnić plecak tylko jednym przedmiotem. Robimy to po kolei dla plecaka o pojemnościach ***j*** (1,2,…,10). Pierwsze 6 komórek jest zerami ponieważ koszula flanelowa zajmuje 7 jm, więc się nie mieści. Dopiero dla plecaka o wadze 7 wpisujemy jej wartość.

Tabela 2. Przykład – tablica dla pierwszego przedmiotu



W kolejnym kroku mamy już dwa przedmioty do dyspozycji (***i*** = 2). Przepisujemy z wiersza wyżej wartości optymalne dla tych wag plecaka ***j***, dla których nie możemy wsadzić przedmiotu drugiego, zgodnie z warunkiem podanym w pseudokodzie. Dla wag plecaka 8,9,10 możemy wsadzić już spodnie dżinsowe co nam zwiększy wartość optymalną ***Pij***. Biorąc pod uwagę te dwa warunki wypełniamy tabelę do wiersza trzeciego.

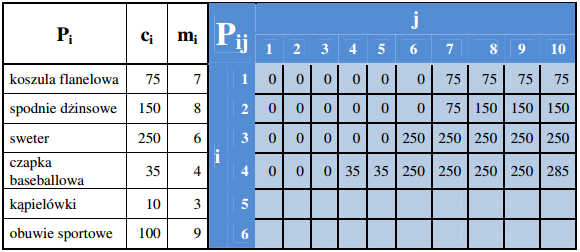
Tabela 3. Tablica dla trzech przedmiotów



Dla czwartego przedmiotu sytuacja się komplikuje. Czapka baseballowa waży mało więc mieści się już od wagi ***j*** = 4. Jednak sweter jest więcej warty, więc wsadzamy go na pierwszej możliwej pozycji. Dla maksymalnej wagi plecaka równej 10 możemy wsadzić i sweter i czapkę baseballową. Tę sytuację widać w pseudokodzie:   
A[i,j] = max(A[i-1,j], A[i-1,j-w[i]] + c[i]). Wybieramy większą z podanych wartości. W naszym przypadku jest to druga – poprzednia wartość optymalna dla danej wagi plecaka zsumowana z aktualnym przedmiotem.

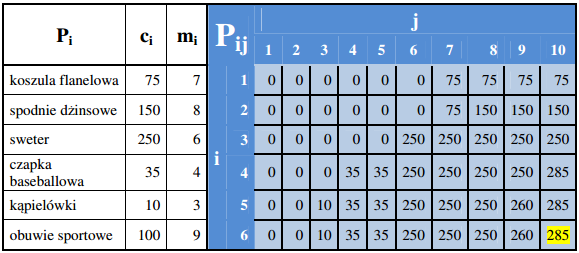
Uzupełniona tabela o czwarty przedmiot wygląda następująco:

Tabela 4. Tablica dla czterech przedmiotów



Pamiętając o warunkach podanych w pseudokodzie wypełniamy tablicę następująco:

Tabela 5. Wypełniona tablica



Końcowa wartość optymalna została zaznaczona na żółto. Wiemy już jaka jest maksymalna wartość plecaka. Teraz należy odczytać jakie przedmioty do plecaka włożyliśmy.

Zasada jest prosta. Zaczynamy od wzięcia el. Optymalnego, czyli ***i*** = ***i*max**oraz ***j***=***jmax****.*   
W naszym przypadku ***i*** = 6 i ***j*** = 10;   
**P**(6,10): sprawdzamy, czy w komórce wyżej (***i***-1) nie ma tej samej wartości. Jeżeli jest, to przechodzimy do niej. **P**(5,10): sprawdzamy dalej. Jest. Przechodzimy do ***i***-1.  
P(4,10): sprawdzamy dalej. Nie ma. Zapisujemy przedmiot do wyniku. Od ***j*** odejmujemy wagę przedmiotu ***i***. ***j*** = 10 – 4.   
**P**(3,6): sprawdzamy dalej. Nie ma. Zapisujemy przedmiot do wyniku. Od ***j***  odejmujemy wagę przedmiotu ***i***. ***j*** = 6 – 6.  
Jeżeli ***j*** <= 0 kończymy proces.  
Jak widać nasze znalezione przedmioty to: **4** i **3**, czyli czapka baseballowa i sweter, ich wartości sumują się do 285. Wszystko się zgadza.

# Opis implementacji algorytmu

Funkcją znajdującą rozwiązanie algorytmem dynamicznym jest PlecakDynamic z vectorem rozmiarów przedmiotów sizes i ich wartości values, całkowitym rozmiarem plecaka totalsize i liczbą przedmiotów itemsnumw argumentach. Vector jako struktura danych został wykorzystany ze względu na jego wygodę użycia i wystarczającą wydajność. Funkcja wygląda następująco:

vector<short> PlecakDynamic(vector<int> sizes, vector<int> values, int totalsize, int itemsnum){

vector<vector<int>>p(itemsnum + 1, vector<int>(totalsize + 1, 0));

vector<short> wynik; // vector z wynikiem

for (auto i = 1; i <= itemsnum; i++) // rozważanie kolejno i pierwszych przedmiotów

for (auto j = 1; j <= totalsize; j++)

//sprawdzenie, czy i-ty element mieści się w plecaku o rozmiarze j

if (j >= sizes[i-1] && p[i-1][j] < p[i-1][j - sizes[i-1]] + values[i-1])

p[i][j] = p[i-1][j - sizes[i-1]] + values[i-1];

else

p[i][j] = p[i-1][j];

for (auto i = itemsnum, j = totalsize; i >= 1 && j > 0; i--) // wygenerowanie rozwiązania

if (p[i][j] != p[i-1][j]){

j = j - sizes[i-1];

wynik.push\_back(i-1);

}

return wynik;

}

# Testowanie

Pliki wejściowe dla programu mają następującą postać: liczba elementów plecaka i pary opisujące rzeczy (rozmiar i wartość).

Funkcja generująca rzeczy do umieszczenia w plecaku dopuszcza tylko takie rozwiązania, w których suma wartości wszystkich wygenerowanych przedmiotów przekracza całkowitą wielkość plecaka zwiększoną o 25%. W testowaniu losowych instancji wielkość plecaka jest ustawiana automatycznie według wzoru:

*Rozmiar = 1,5 \* liczba\_elementów*

Początkowy rozmiar plecaka to 1500, a liczba elementów to 1000. Rozmiar plecaka jest zwiększany o 1500 w każdej iteracji aż do 21000 i liczba elementów wzrasta od 1000 do 14000.

W rozszerzonych testach program wykonuje algorytm dla powyższych rozmiarów, dla każdego ustawiając wszystkie liczby elementów od 1000 do 14000 (co 1000).

Do zbadania czasu działania algorytm był wykonywany 100 razy (dla losowo generowanych instancji o zadanym rozmiarze), a następnie jego wyniki zostały uśrednione.

Czas wykonywania algorytmu był mierzony dzięki własnej klasie MyTimer. Zlicza ona takty zegara procesora i dzieli tę liczbę przez jego częstotliwość, a następnie mnoży przez 1000 aby wynik był podany w ms.

# Wyniki

Poniżej przedstawiono wyniki pomiarów. Najpierw przeprowadzony został szybki test dla pojedynczych wielkości plecaka dla danej liczby elementów, potem został on rozszerzony o więcej wartości. W takiej samej kolejności prezentowane są wyniki.

Tabela 6 Pomiar czasów wykonywania się algorytmu dla pojemności równej 1,5 liczby elementów

|  |  |
| --- | --- |
| Liczba elementów | Algorytm dynamiczny |
| [ms] |
| 1000 | 0,05 |
| 2000 | 0,20 |
| 3000 | 0,46 |
| 4000 | 0,89 |
| 5000 | 1,21 |
| 6000 | 1,83 |
| 7000 | 2,43 |
| 8000 | 3,39 |
| 9000 | 4,04 |
| 10000 | 5,06 |
| 11000 | 6,19 |
| 12000 | 8,12 |
| 13000 | 9,41 |
| 14000 | 10,77 |

Tabela 7 Pomiar czasów wykonywania się algorytmu dla różnych pojemności i liczb elementów

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pojemność | Czas [ms] | | | | | | | | | | | | | |
|
| 1500 | 0,04 | 0,10 | 0,11 | 0,13 | 0,34 | 0,30 | 0,42 | 0,46 | 0,65 | 0,58 | 0,55 | 0,68 | 0,62 | 0,95 |
| 3000 | 0,09 | 0,13 | 0,25 | 0,29 | 0,52 | 0,46 | 0,51 | 0,99 | 0,88 | 0,89 | 0,76 | 1,23 | 0,97 | 1,43 |
| 4500 | 0,11 | 0,20 | 0,30 | 0,60 | 0,72 | 0,69 | 0,95 | 1,22 | 1,20 | 1,31 | 1,46 | 1,60 | 1,67 | 1,86 |
| 6000 | 0,19 | 0,27 | 0,41 | 0,74 | 0,90 | 1,22 | 1,39 | 1,62 | 1,86 | 2,02 | 2,33 | 2,62 | 2,76 | 2,91 |
| 7500 | 0,18 | 0,33 | 0,67 | 0,80 | 1,11 | 1,43 | 1,81 | 1,96 | 2,26 | 2,55 | 2,76 | 3,22 | 3,45 | 3,82 |
| 9000 | 0,21 | 0,40 | 1,02 | 1,09 | 1,47 | 1,92 | 2,34 | 2,44 | 2,78 | 3,16 | 3,50 | 3,87 | 4,04 | 4,38 |
| 10500 | 0,24 | 0,51 | 0,74 | 1,33 | 2,19 | 2,10 | 2,65 | 2,93 | 3,38 | 3,58 | 4,04 | 4,28 | 4,84 | 5,04 |
| 12000 | 0,28 | 0,58 | 0,99 | 1,52 | 2,08 | 2,47 | 3,06 | 3,22 | 3,64 | 4,15 | 4,49 | 4,92 | 5,45 | 5,93 |
| 13500 | 0,31 | 0,63 | 1,14 | 1,90 | 2,20 | 2,84 | 3,39 | 3,69 | 4,17 | 4,69 | 5,11 | 5,79 | 6,16 | 6,62 |
| 15000 | 0,37 | 0,78 | 1,36 | 2,04 | 2,44 | 2,91 | 3,71 | 4,07 | 4,69 | 5,05 | 5,71 | 6,07 | 6,79 | 7,54 |
| 16500 | 0,43 | 0,80 | 1,54 | 2,38 | 3,06 | 3,34 | 4,11 | 4,44 | 5,06 | 5,75 | 6,26 | 6,94 | 7,46 | 8,17 |
| 18000 | 0,48 | 0,81 | 1,58 | 2,85 | 4,73 | 3,67 | 4,40 | 5,01 | 5,54 | 6,38 | 6,88 | 7,58 | 8,29 | 9,13 |
| 19500 | 0,60 | 1,00 | 1,81 | 2,53 | 3,64 | 4,17 | 4,71 | 5,25 | 6,00 | 6,70 | 7,67 | 8,23 | 9,12 | 9,44 |
| 21000 | 0,51 | 1,05 | 1,90 | 2,89 | 3,66 | 4,31 | 4,93 | 5,69 | 6,63 | 7,21 | 8,11 | 9,20 | 9,60 | 10,44 |
| Liczba elementów | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 | 8000 | 9000 | 10000 | 11000 | 12000 | 13000 | 14000 |

Wykres 1 Czas wykonywania się algorytmu w zależności od pojemności i liczby elementów (słupkowy)

Wykres 2 Czas wykonywania się algorytmu w zależności od pojemności i liczby elementów (powierzchniowy)

# Podsumowanie i wnioski

Zwiększanie pojemności plecaka, albo liczby przedmiotów powoduje nieznaczny wzrost czasu wykonywania algorytmu. Zwiększanie obu parametrów powoduje znacznie szybszy wzrost, co dobrze obrazuje wykres 2.

# Bibliografia

* „The knapsack problem”, http://www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf
* Jarosław Henke, „Problem plecakowy”, http://www-users.mat.uni.torun.pl/~henkej/knapsack.pdf