Ćwiczenie 3

Programowanie dynamiczne

[źródło: „Wprowadzenie do algorytmów”, T.H. Cormen, Ch.E. Leiserson, R.L.Rivest, Wyd. Naukowo-Techniczne Warszawa, 2001; „Złożoność obliczeniowa problemów kombinatorycznych”, Jacek Błażewicz, Wydawnictwa Naukowo-Techniczne Warszawa 1988].

I. Programowanie dynamiczne (optymalizacja dynamiczna)

Jest pewnym rozszerzeniem strategii „dziel i zwyciężaj”. „Programowanie” oznacza w tym kontekście tabelaryczną metodę rozwiązywania problemów, a nie pisanie programów komputerowych. Wiemy, że w algorytmach typu „dziel i zwyciężaj” dzieli się problem na niezależne podproblemy, rozwiązuje je rekurencyjnie, a następnie łączy się rozwiązania wszystkich podproblemów w celu utworzenia rozwiązania pierwotnego problemu. Programowanie dynamiczne można stosować wtedy, kiedy podproblemy **nie są niezależne**, tzn. kiedy podproblemy mogą zawierać te same podpodproblemy. Wtedy algorytm typu „dziel i zwyciężaj” wykonuje więcej pracy niż to w istocie jest konieczne, wielokrotnie bowiem rozwiązuje ten sam podproblem. W algorytmie opartym na programowaniu dynamicznym rozwiązuje się każdy podproblem tylko raz, po czym zapamiętuje się wynik w odpowiedniej tabeli, unikając w ten sposób wielokrotnych obliczeń dla tego samego podproblemu.

Programowanie dynamiczne jest zwykle stosowane do problemów optymalizacyjnych. W tego typu zagadnieniach możliwych jest wiele różnych rozwiązań. Z każdym rozwiązaniem jest związana pewna liczba (koszt). Rozwiązanie optymalne to takie, które ma optymalny (minimalny lub maksymalny) koszt. Może być wiele rozwiązań optymalnych, wszystkie o tym samym optymalnym koszcie.

Proces projektowania algorytmu opartego na programowaniu dynamicznym można podzielić na cztery etapy:

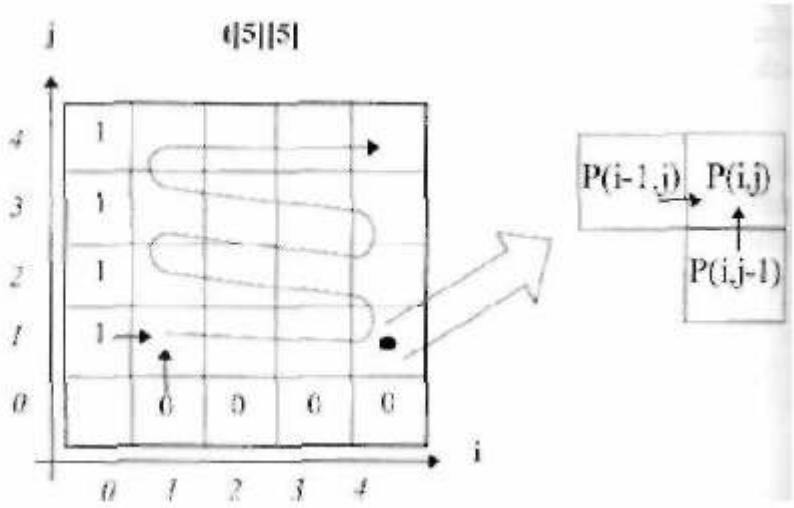
1) Scharakteryzowanie struktury optymalnego rozwiązania.
2) Rekurencyjne zdefiniowanie kosztu optymalnego rozwiązania.
3) Obliczenie optymalnego kosztu metodą wstępującą (ang. bottom-up) – czyli rozpoczynając od najmniejszych podproblemów, rozwiązywać coraz większe, wykorzystując zapamiętane rozwiązania mniejszych.
4) Konstruowanie optymalnego rozwiązania na podstawie wyników wcześniejszych obliczeń.

Etapy 1-3 stanowią trzon rozwiązania problemu za pomocą programowania dynamicznego. Jeżeli interesuje nas tylko koszt rozwiązania optymalnego, to etap 4 można pominąć. Jeżeli mimo wszystko wykonujemy etap 4, to często wygodnie jest zapamiętywać dodatkowe informacje w etapie 3, co niejednokrotnie znacznie ułatwia zrekonstruowanie optymalnego rozwiązania.

Przykład:

$$P(i,j) = \begin{cases} 1 & \text{jeśli } i = 0 \text{ oraz } j > 0, \\ 0 & \text{jeśli } i > 0 \text{ oraz } j = 0, \\ \dfrac{P(i-1,j) + P(i,j-1)}{2} & \text{jeśli } i > 0 \text{ oraz } j > 0. \end{cases}$$

Rys. 9 - j .
„Dwuwymiarowy"
wzór rekurencyjny.



Gdy mamy te wszystkie informacje, realizacja programowa jest natychmiastowa:

```
const  n=5;
void dynam(double P[n] [r] )
{
int    i,j;
for(i=1;i<n;i++)          //   inicjacja
  {
  P[i] [0]=0;
  P[0][i]-1;
  }
for(j=1;j<n;j++)          //   progresja
    for(i=1;i<n;i++)
        P[i][j]=(P[i-1][j]+P[i][j-1])/2,0;
}
```

II. Algorytmy zachłanne

Algorytmy służące do rozwiązywania problemów optymalizacyjnych polegają zwykle na podejmowaniu ciągu decyzji – w każdym kroku należy wybrać jedną z wielu możliwości. Dla wielu takich problemów istnieją prostsze i bardziej efektywne algorytmy. Są to algorytmy zachłanne.

Algorytm zachłanny (ang. *greedy algorithm*) wykonuje zawsze działanie, które wydaje się w danej chwili najkorzystniejsze. Wybiera zatem lokalnie optymalną możliwość w nadziei, że doprowadzi ona do globalnie optymalnego rozwiązania.

Algorytmy zachłanne są dość skuteczne i dają dobre rezultaty dla szerokiej gamy problemów. Algorytmy, które można traktować jako zastosowanie metody zachłannej: algorytm wyznaczania minimalnego drzewa rozpinającego, algorytm Dijkstry znajdowania najkrótszych ścieżek z ustalonego wierzchołka w grafie oraz heurystyczny algorytm zachłanny Chvátala dla problemu pokrycia zbioru.

1. Problem kasjera

Problem

Kasjer ma wydać resztę, będącą dowolną kwot między 0,01$ a 0,99$, przy użyciu minimalnej liczby monet.

Rozwiązanie oparte na algorytmie zachłannym

Najpierw używamy monety o największej dopuszczalnej wartości, redukując w ten sposób problem do wypłacenia mniejszej kwoty.

Przykład

Aby wydać 0,94 $, kasjer wypłaci:
- półdolarówkę (zostawiając do zapłacenia 0,44 $), następnie
- ćwierćdolarówkę (zostaje 0,19 $),
- dziesięciocentówkę (zostaje 0,09 $),
- pięciocentówkę (zostaje 0,04 $) i w końcu
- cztery jednocentówki

W sumie kasjer wypłaci osiem monet. Jest to minimalna liczba i faktycznie algorytm zachłanny jest optymalny dla monet amerykańskich.

Jak jest dla innych systemów monetarnych? (Dla zainteresowanych☺)

2. Problem wyboru zajęć

Problem

Dany jest zbiór S = {1,2,....n} składający się z n proponowanych zajęć, do których mają być przydzielone zasoby, takie jak na przykład sala wykładowa, w której może się odbywać w danej chwili tylko jedno z tych zajęć. Każde zajęcia maj swój **czas rozpoczęcia** s_i oraz **czas zakończenia** f_i , takie, że $s_i \leq f_i$. Jeżeli zajęcie o numerze i zostanie wytypowane, to zajmuje zasób przez prawostronnie otwarty przedział czasu $[s_i , f_i)$. Zajęcia o numerze i oraz j są **zgodne**, jeśli przedziały $[s_i , f_i)$, i $[s_j , f_j)$, nie zachodzą na siebie (czyli $s_i \geq f_j$ lub $s_j \geq f_i$).

Problem wyboru zajęć polega na wyborze największego podzbioru parami zgodnych zajęć .

Rozwiązanie

Bez utraty ogólności zakładamy, że zajęcia są uporządkowane ze względu na czas zakończenia:

$$f_1 \leq f_2 \leq \ldots \leq f_n$$

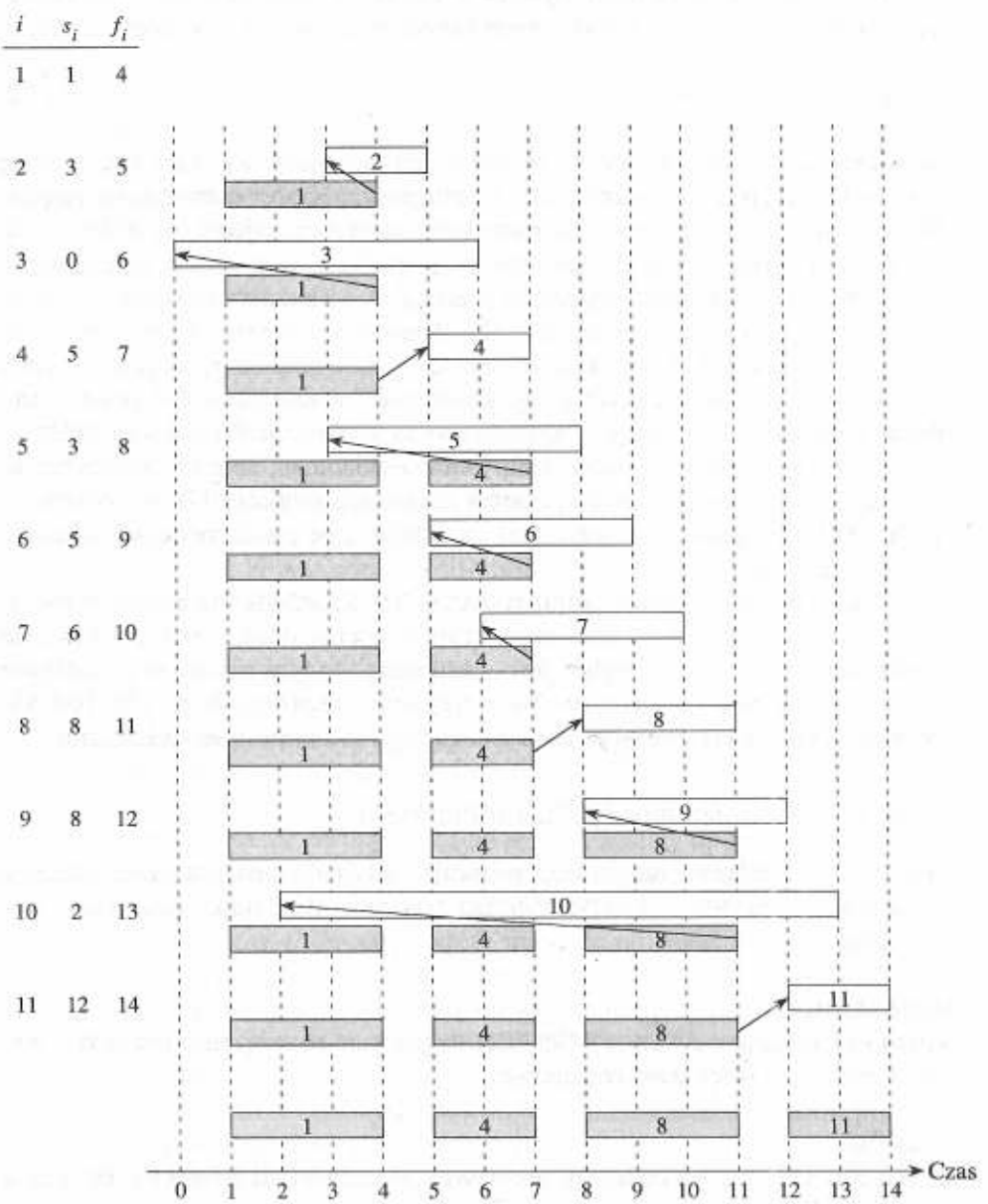
Jeżeli powyższe założenie nie jest spełnione, to przed realizacją algorytmu należy posortować ciąg czasów zakończenia (optymalny koszt sortowania O($n$log$n$).

W poniższej procedurze zakładamy, że s i f są reprezentowane jako tablice.

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n ← length[s]
2  A ← {1}
3  j ← 1
4  for i← 2 to n
5      do if s_i ≥ f_j
6          then A ← A ∪ {i}
7          j ← i
8  return A
```

Wynikiem powyższej procedury jest zbiór A zawierający numery zajęć. Zmienna j zawiera ostatnio dodane do A zajęcie.

Przykład



Rys. 17.1. Działanie procedury GREEDY-ACTIVITY-SELECTOR na 11-elementowym zbiorze zajęć przedstawionym po lewej stronie rysunku. Każdy poziomy rząd na rysunku odpowiada jednej iteracji pętli **for** w wierszach 4-7. Zajęcia włączone do zbioru A zostały zacieniowane, a zajęcie i (białe na rysunku) jest aktualnie rozpatrywane. Jeśli czas rozpoczęcia s_i zajęcia i jest wcześniejszy niż czas zakończenia f_j zajęcia ostatnio dodanego do A (strzałka między nimi wskazuje na lewo), to zostaje ono odrzucone. W przeciwnym razie (strzałka wskazuje do góry lub w prawo) zostaje ono wybrane i dodane do zbioru A

Powyższy schemat ilustruje działanie procedury ACTIVITY_SELECTOR na 11-elementowym zbiorze zajęć. Każdy poziomy rząd na rysunku odpowiada jednej iteracji pętli "for" w wierszach 4-7. Strzałka w lewo wskazuje zajęcia odrzucone, strzałka w prawo - zajęcia wybrane i dodane do zbioru A. Zajęcia włączone do zbioru A zostały zacieniowane, a zajęcie i (białe na rysunku) jest aktualnie rozpatrywane. Jeżeli czas rozpoczęcia s_i zajęcia i jest wcześniejszy niż czas zakończenia f_j zajęcia ostatnio dodanego do A (strzałka między nimi wskazuje na lewo), to zostaje ono odrzucone. W przeciwnym razie (strzałka go góry lub w prawo) zostaje ono wybrane i dodane do zbioru A.

Ponieważ zajęcia są rozpatrywane w porządku rosnącego czasu zakończenia f_j jest zawsze największym czasem zakończenia zajęcia należącego do A, tj.

$$f_j = \max \{f_k : k \in A\}$$

W wierszach 2-3 jest wybierane zajęcie 1, jednoelementowy zbiór {1} staje się wartością zmiennej A, a zmiennej j zostaje przypisany numer tego zajęcia. W wierszach 4-7 są kolejno rozpatrywane wszystkie zajęcia i; każde z nich zostaje dołączone, jeżeli jest zgodne ze wszystkimi dotychczas dołączonymi zajęciami. Aby stwierdzić, czy zajęcie i jest zgodne z każdym zajęciem ze zbioru A, wystarczy zgodnie z powyższym równaniem sprawdzić (wiersz 5), czy jego czas rozpoczęcia s_i nie jest wcześniejszy niż czas zakończenia f_j zajęcia ostatnio dodanego do A. Jeśli zajęcie i jest zgodne, to w wierszach 6-7 zostaje ono dodane do zbioru A oraz jest aktualizowana wartość j. Procedura GREEDY-ACTIVITY-SELECTOR jest dość efektywna. Zakładając, że dane wejściowe są uporządkowane rosnąco według zakończenia zajęcia, GREEDY-ACTIVITY-SELECTION wyznacza maksymalny podzbiór zajęć z n-elementowego zbioru S w czasie $\Theta(n)$

Zajęcie wybierane przez GREEDY-ACTIVITY-SELECTION ma zawsze najwcześniejszy czas zakończenia wśród zajęć, które mogą być dołączone bez zakłócenia zgodności zbioru A. Ten wybór jest „zachłanny” w tym sensie, że pozostawia intuicyjnie możliwie najwięcej swobody przy wyborze pozostałych zajęć. Jest tak, bo wybór taki maksymalizuje ilość nie zajętego czasu po jego dokonaniu.

Koszt czasowy

Zakładając, że dane wejściowe są uporządkowane rosnąco według czasów zakończenia zajęć , procedura ACTIVITY_SELECTOR dla n-elementowego zbioru zajęć działa w czasie $Q(n)$.

Zajęcia wybierane przez GREEDY-ACTIVITY-SELECTOR maja zawsze najwcześniejszy czas zakończenia wśród wszystkich zajęć. Po wybraniu wszystkich zajęć ze zbioru A pozostaje maksymalna ilość nie zajętego czasu.

3. Ogólne własności metody zachłannej

Jak przekonać się, czy zastosowanie strategii zachłannej daje optymalne rozwiązanie problemu?

Problemy, dla których może być zastosowana strategia zachłanna mają dwie cechy charakterystyczne:

- własność wyboru zachłannego,
- własność optymalnej podstruktury.

a) Własność wyboru zachłannego

Jeżeli wybory "lokalne" są optymalne, to wybór "globalny" (ostateczny) jest optymalny.
Różnica między strategią zachłanną a programowaniem dynamicznym polega na tym, że w programowaniu dynamicznym w każdym kroku podejmowane są decyzje, których wybór zależy od rozwiązań podproblemów. W algorytmie zachłannym wybory są podejmowane jako najlepsze (z punktu widzenia zadania) w danej chwili. Wybory podejmowane w algorytmie zachłannym nie są zależne od wyborów przeszłych, w przeciwieństwie do wyborów podejmowanych przy strategii programowania dynamicznego. Można formalnie udowodnić (stosując metodę indukcji), że dany problem ma własność wyboru zachłannego.

b) Własność optymalnej podstruktury

Problem ma własność optymalnej podstruktury, jeżeli optymalne rozwiązanie jest funkcją optymalnych rozwiązań podproblemów. Ta własność jest również spełniona dla problemów rozwiązywalnych metodą programowania dynamicznego. Na przykład dla problemu wyboru zajęć własność optymalnej podstruktury polega na tym, że: Jeżeli optymalne rozwiązanie A tego problemu rozpoczyna się od zajęć o numerze 1, to $A' = A \setminus \{1\}$ jest optymalnym rozwiązaniem problemu optymalnego wyboru zajęć dla zbioru $S' = \{ i \in S: s_i \geq f_1\}$.

Przykład:

[źródło: „Złożoność obliczeniowa problemów kombinatorycznych”, Jacek Błażewicz, Wydawnictwa Naukowo-Techniczne Warszawa 1988].

PROBLEM PLECAKOWY

Problem optymalizacyjny. Parametrami są: skończony zbiór elementów $A = \{a_1, a_2, ...,a_n\}$, z których każdy ma określony rozmiar $s(a_i)$ i wartość $w(a_i) \in N$ oraz pojemność b plecaka (N oznacza zbiór liczb całkowitych dodatnich). Rozwiązaniem jest taki podzbiór elementów $A' \subset A$, który maksymalizuje łączną wartość wybranych elementów $\sum_{ai \in A'} w(a_i)$ przy warunku nie przekroczenia dopuszczalnej pojemności, czyli $\sum_{ai \in A'} s(a_i) \leq b$.

Odpowiadający temu problemowi optymalizacyjnemu decyzyjny problem plecakowy można przedstawić w następujący sposób:

Parametry: Skończony zbiór elementów $A = \{a_1, a_2, ...,a_n\}$, rozmiar $s(a_i)$ oraz wartość $w(a_i)$ dla każdego elementu $a_i \in N$, stałe b oraz y.

Pytanie: Czy istnieje podzbiór $A' \subset A$, taki że:

$$\sum_{ai \in A'} s(a_i) \leq b$$

$$\sum_{ai \in A'} w(a_i) \geq y$$

Jeden z konkretnych problemów w tym ostatnim przypadku jest określony przez zbiór A składający się z pięciu elementów o rozmiarach wynoszących odpowiednio 5, 3, 2, 4, 3 i wartościach 3, 4, 2, 6, 1 oraz przez pojemność $b = 10$ i stałą $y = 12$. Odpowiedź brzmi „Tak”.

Dyskretny problem plecakowy

Złodziej rabujący sklep znalazł n przedmiotów; i-ty przedmiot ma wartość c_i złotych i waży w_i kilogramów, gdzie c_i i w_i są nieujemnymi liczbami całkowitymi. Dąży on do zabrania jak najwartościowszego łupu, ale do swojego plecaka nie może wziąć więcej niż W kilogramów. Złodziej nie może dzielić przedmiotów (zabrać do plecaka tylko część wybranego przedmiotu) ani wielokrotność przedmiotu. Interesuje nas odpowiedź na pytanie: Jakie przedmioty z puli n wybranych przedmiotów może zabrać złodziej, przy wymienionych wyżej ograniczeniach.

Dyskretny problem plecakowy nie może być rozwiązany metodą zachłanną.

- Czy dyskretny problem plecakowy ma własność wyboru zachłannego?
  Odpowiedź : NIE
  Kontrprzykład
  Dane: n=3. Zbiór wybranych przedmiotów składa si z następujących elementów:
  - *Przedmiot 1* waga 10 kg i wartość 60 zł
  - *Przedmiot 2* o waga 20 kg i wartość 100 zł
  - *Przedmiot 3* o waga 30 kg i wartość 120 zł
  Plecak ma maksymalną pojemność 50 kg.

Kryterium wyboru przy zastosowaniu metody zachłannej jest cena jednostkowa, czyli cena 1 kg przedmiotu. Według tego kryterium najwyższą cenę jednostkową ma :

- *Przedmiot 1* (6 zł/kg),
- *Przedmiot 2* (5 zł/kg),
- *Przedmiot 3* (4 zł/kg).

i to właśnie *Przedmiot 1* zostałby wybrany jako pierwszy. Następny wybrany byłby *Przedmiot 2*. Do plecaka nie trafiłby *Przedmiot 3*, ponieważ wszystkie trzy przedmioty mają za dużą wagę łączną (60 kg). Rozwiązanie polegające na wyborze *Przedmiotu 1* i *Przedmiotu 2* nie jest optymalne. Rozwiązaniem optymalnym jest natomiast wybór *Przedmiotu2* i *Przedmiotu 3* (łączna waga wynosi 50 kg, łączna warto 220 zł).

- Czy dyskretny problem plecakowy ma własność optymalnej podstruktury?
  Odpowiedź : TAK.

Dyskretny problem plecakowy wykazuje cechę optymalnej podstruktury. Rozważmy najwartościowszy ładunek plecaka o masie nie większej niż W. Jeżeli usuniemy z tego ładunku przedmiot j o wadze w_j, to pozostający ładunek jest najwartościowszym zbiorem przedmiotów o wadze nie przekraczającej W - w_j, jakie złodziej może wybrać z n-1 oryginalnych przedmiotów z wyjątkiem j.
Okazuje się, że dyskretny problem plecakowy można rozwiązać stosując technikę programowania dynamicznego.

Ciągły problem plecakowy

Ciągły problem plecakowy różni się od dyskretnego problemu plecakowego tym, że złodziej może zabierać ułamkowe części przedmiotów (dogodniej jest mówić o „substancjach", a nie „przedmiotach").

Ciągły problem plecakowy może być rozwiązany metodą zachłanną.

- Czy ciągły problem plecakowy ma własność wyboru zachłannego?

Odpowiedź : TAK

Algorytm

1) Policzyć cenę jednostkową każdego przedmiotu.
2) Zabrać największą możliwą ilość najbardziej wartościowej substancji.
3) Jeżeli zapas tej substancji się wyczerpał, a w plecaku wciąż jest jeszcze wolne miejsce, złodziej wybiera następną pod względem ceny jednostkowej substancję i wypełnia nią plecak.
4) Kroki 2 i 3 są powtarzane do momentu, gdy plecak będzie już pełen. Powyższy algorytm wymaga, aby substancje były posortowane według malejącej ceny jednostkowej. Przy tym założeniu wybór określony w algorytmie ma własność wyboru zachłannego.

Przykład

Dane jak w przykładzie dla dyskretnego problemu plecakowego.

Przypomnijmy:

- *Przedmiot 1* (*Substancja 1*) o wadze 10 kg i wartości 60 zł (6 zł/kg)
- *Przedmiot 2* (*Substancja 2*) o wadze 20 kg i wartości 100 zł (5 zł/kg)
- *Przedmiot 3* (*Substancja 3*) o wadze 30 kg i wartości 120 zł (4 zł/kg)

Plecak ma maksymalną pojemność 50 kg.

Rozwiązanie:

Złodziej wkłada do plecaka :

10 kg *Substancji 1* (wartość 60 zł),

20 kg *Substancji 2* (wartość 100 zł),

20 kg *Substancji 3* (wartość 80 zł)

Łączna wartość wynosi: 60 zł + 100 zł + 80 zł = 240 zł.

- Czy ciągły problem plecakowy ma własność optymalnej podstruktury?

Odpowiedź: TAK.

Jeżeli usuniemy z optymalnego ładunku w kilogramów pewnej substancji j, to pozostający ładunek powinien być najwartościowszym ładunkiem o wadze co najwyżej W-w, który złodziej może skompletować z n- 1 oryginalnych substancji, plus w_j - w kilogramów substancji j.

Koszt czasowy

Sortowanie ciągu substancji według kosztu jednostkowego jest realizowane kosztem optymalnym Q($n$log$n$). Zasadniczy algorytm rozwiązujący ciągły problem plecakowy ma koszt Q(n). Zatem koszt łączny wynosi Q($n$log$n$).

4. Przykłady zastosowania strategii zachłannej

- algorytm kompresji plików metod Huffmana

- algorytm Kruskala wyznaczania minimalnego drzewa rozpinającego grafu

5. Przykładowe rozwiązanie problemu plecakowego (definicja powyżej) za pomocą programowania dynamicznego:

1) **Scharakteryzowanie struktury optymalnego rozwiązania.**

Tworzymy macierz $V[0..n, 0..b]$, gdzie $1 \le i \le n$ i $0 \le s \le b$, element $V[i,s]$ przechowuje maksymalną wartość dla dowolnego podzbioru elementów $\{1, 2, \dots, i\}$ o sumarycznym rozmiarze nie większym niż s.
Jeżeli obliczymy wszystkie elementy tej macierzy to element $V[n,b]$ będzie zawierał rozwiązanie problemu.

2) **Rekurencyjne zdefiniowanie kosztu optymalnego rozwiązania.**

$$\begin{cases} V[0,s] = 0 & dla\ 0 \le s \le b \\ V[i,s] = \max(V[i\text{-}1,\ s],\ w(a_i) + V[i\text{-}1,\ s\text{-}s(a_i)]) & dla\ 1 \le i \le n,\ 0 \le s \le b \end{cases}$$

gdzie

$V[i-1,s]$ oznacza najlepsze (poprzednio obliczone i zapamiętane w tabeli) rozwiązanie dla rozważanej aktualnie pojemności plecaka s oraz podzbioru elementów $\{1, \dots i\text{-}1\}$

$V[i-1,s-s(a_i)]$ - oznacza najlepsze (poprzednio obliczone i zapamiętane w tabeli) rozwiązanie dla aktualnie rozważanej pojemności plecaka s pomniejszonej o rozmiar elementu $s(a_i)$, który próbujemy dołożyć do plecaka, dla podzbioru elementów $\{1, \dots i\text{-}1\}$

3) **Obliczenie optymalnego kosztu metodą wstępującą (ang. bottom-up).**

Wypełnij pierwszy wiersz zgodnie ze wzorem $V[0,s] = 0\ dla\ 0 \le s \le b$ (*bottom*)
Oblicz pozostałe elementy macierzy używając wzoru $V[i,s] = \max(V[i\text{-}1,\ s],\ w(a_i) + V[i\text{-}1,\ s\text{-}s(a_i)])\ \ dla\ 1 \le i \le n,\ 0 \le s \le b$. Wypełniaj tabelę rząd po rzędzie.

V[i,s]	s=0	1	2	3	…	…	b
i = 0	0	0	0	0	0	0	0
1	→	→	→	→	→	→	→
2	→	→	→	→	→	→	→
…	→	→	→	→	→	→	→
n	→	→	→	→	→	→	→

bottom

↓

up

Przykład:

Dane:

b = 10

i	1	2	3	4
$w(a_i)$	10	40	30	50
$s(a_i)$	5	4	6	3

Rozwiązanie:

V[i,s]	0	1	2	3	4	5	6	7	8	9	10
i=0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	10	10	10	10	10	10
2	0	0	0	0	40	40	40	40	40	50	50
3	0	0	0	0	40	40	40	40	40	50	70
4	0	0	0	50	50	50	50	90	90	90	90

Przykładowe obliczenia dla poszczególnych elementów tabeli:

V[1,5] = max{V[0,5], 10 + V[0, 5-5]} = 10

V[2,4] = max{V[1,4], 40 + V[1, 4-4]} = 40

V[2,9] = max{V[1,9], 40 + V[1, 5]} = max{10, 40 + 10} = 50

V[3,4] = max{V[2,4], 0} = 40

V[3,6] = max{V[2,6], 30 + V[2, 6-6]} = max{40, 30} = 40

V[4,8] = max{V[3,8], 50 + V[3, 8-3]} = max{40, 50 + 40} = 90

Przedstawiona metoda nie daje informacji, który podzbiór elementów dał rozwiązanie optymalne (w tym przypadku były to elementy {2, 4}).

Zadanie 3

1. W porcie stoi statek "MS KONTENER" o ładowności b ton gotowy do załadunku. Statek można załadować różnymi kontenerami, których liczba jest równa n. Kontener i-ty posiada swój ciężar s(i) oraz wartość w(i).
   Podaj optymalny załadunek na statek, tzn. które kontenery zostaną załadowane, w taki sposób, aby wartość ładunku była maksymalna.

2. Problem ten rozwiąż za pomocą algorytmu programowania dynamicznego.
   Efektywność uzyskanego rozwiązania porównaj z algorytmem zachłannym, mierząc czas uzyskania rozwiązania dla obu metod, dla tych samych instancji problemu.

   Podaj wykresy porównawcze dla obu metod przy:

   a) stałej ładowności statku, zmiennej liczbie kontenerów,
   b) zmiennej ładowności, liczba kontenerów pozostaje stała.

3. Sformułuj wnioski dotyczące efektywności zastosowanych metod oraz ich złożoności obliczeniowej. Podaj przynależność badanego problemu do określonej klasy problemów ze względu na ich złożoność obliczeniową.

Termin oddania: 30.05.2009r.