Metody rozwiązywania problemów

Beata Laszkiewicz

Część III

PROGRAMOWANIE DYNAMICZNE

Problem plecakowy

Mamy danych *n* przedmiotów:

- każdy o ustalonej wartości $p_i > 0$, i = 1, ..., n
- oraz o ustalonej masie $w_i > 0$, i = 1, ..., n,

oraz plecak, który może pomieścić W kilogramów, W>0. Naszym zadaniem jest wybrać pewną liczbę przedmiotów tak, by w plecaku pozostało jak najmniej wolnego miejsca oraz by jego wartość była jak największa.

Zadanie optymalizacyjne

Przykład:

Niech W=23, n=4, wartości i masy przedmiotów przedstawiono w tabeli:

i	1	2	3	4
p_i	2	5	7	10
w_i	1	3	2	3
$\frac{p_i}{w_i}$	2	$1\frac{2}{3}$	$3\frac{1}{2}$	$3\frac{1}{3}$

Czy strategie dają optymalne rozwiązanie?

Patrząc na przykład można zauważyć, że zastosowanie strategii III nie daje optymalnego rozwiązania - można zapakować plecak tak, aby jego wartość była równa 80.

Programowanie dynamiczne

- Metoda programowania dynamicznego pozwoli nam uzyskać optymalne rozwiązanie:
- Rozwiązujemy wszystkie mniejsze podproblemy.
 W przypadku problemu plecakowego oryginalny problem można zredukować na dwa sposoby:
 - zmniejszamy zestaw przedmiotów (patrzymy tylko na przedmioty 1, 2, ..., i dla $i \leq n$),
 - zmniejszamy pojemność plecaka (do $j \leq W$).
- Rozwiązania wszystkich podproblemów możemy w prosty sposób zapamiętać w tablicy dwuwymiarowej.

Programowanie dynamiczne

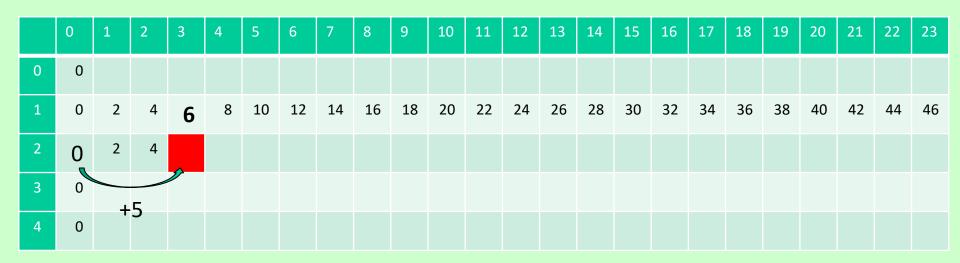
- P[i][j] oznacza najlepszą wartość plecaka, którego waga W nie przekracza j, zapakowanego pewną kombinacją przedmiotów o numerach od 1 do i.
- Rozwiązanie problemu przechowuje komórka P[n][W].

Tabela – pierwszy przedmiot

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0																							
1	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
2	0																							
3	0																							
4	0																							

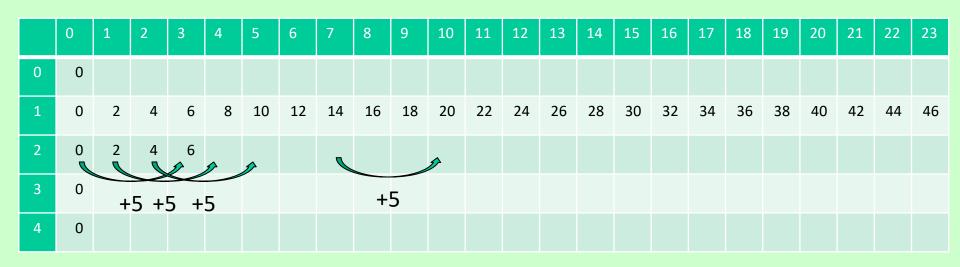
- Już dla W=1 można wypełnić plecak przedmiotem nr 1.
- Nie ma żadnych problemów, dla kolejnych j = 1, 2, ..., W dokładamy przedmiot do plecaka (waży tylko 1 kg).
- Warto zauważyć, że gdy plecak ma W = 0, to nie ma on żadnej wartości (wypełniona zerami zerowa kolumna).

Tabela – drugi przedmiot



- Drugi przedmiot można dołożyć dopiero wtedy, gdy waga plecaka W=3;
 dlatego wcześniej pakujemy tylko przedmiot nr 1.
- Dla j=3 zastanawiamy się:
 - Czy lepiej dołożyć przedmiot nr 2 do plecaka (trzeba coś wyciągnąć i zrobić miejsce),
 - czy brać poprzednie upakowanie...

Tabela – drugi przedmiot

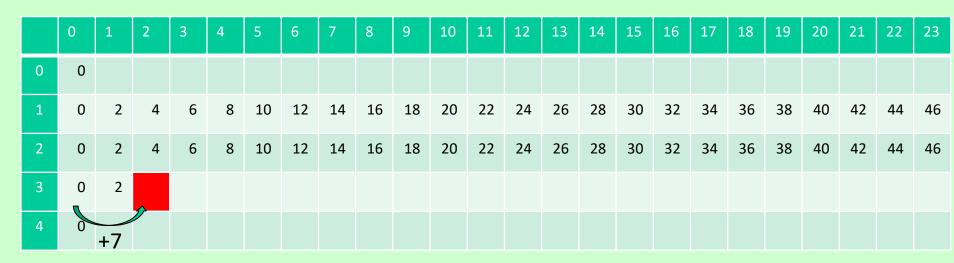


- Dla kolejnych możliwych wag plecaka (j=4, 5,) postępujemy analogicznie...
 - Czy lepiej dołożyć przedmiot nr 2 do plecaka (trzeba coś wyciągnąć i zrobić miejsce),
 - czy brać poprzednie upakowanie...

Po analizie dwóch przedmiotów

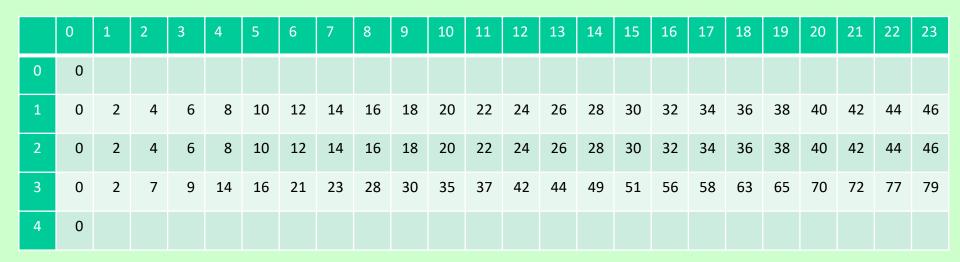
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0																							
1	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
2	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
3	0																							
4	0																							

Trzeci przedmiot



- Trzeci przedmiot można dołożyć dopiero wtedy, gdy waga plecaka W=2;
 dlatego wcześniej wybieramy wcześniejsze upakowanie plecaka.
- Analogicznie jak w poprzednim kroku zastanawiamy się (do j=2, 3,)
 - Czy lepiej dołożyć przedmiot nr 3 do plecaka (trzeba coś wyciągnąć i zrobić miejsce),
 - czy brać poprzednie upakowanie...

Tabela po analizie trzeciego przedmiotu



Analogicznie postępujemy z przedmiotem nr 4....

Efekt końcowy

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
2	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
3	0	2	7	9	14	16	21	23	28	30	35	37	42	44	49	51	56	58	63	65	70	72	77	79
4	0	2	7	10	14	17	21	24	28	31	35	38	42	45	49	52	56	59	63	66	70	73	77	80

Do czego jest potrzebny zerowy wiersz?

- Żeby napisać porządny algorytm, który pracowałby również dla przedmiotu o numerze 1 (w kolejnych krokach odwołujemy się do wartości P[i][j] z wcześniejszych wierszy...)
- Wiersz musi być wyzerowany....

Algorytm:

Dane:

```
n - liczba przedmiotów
p[1..n] - tablica wartości kolejnych przedmiotów
w[1..n] - tablica mas kolejnych przedmiotów
W - maksymalna pojemność plecaka (W jednostek)
```

Wynik:

P[0..n][0..W] - tablica najlepszych "upakowań" plecaka

```
for j = 0, 1, 2, ..., W
   P[0][j] = 0;
for i = 0, 1, 2, ..., n
   P[i][0] = 0;

for i = 1, 2, ..., n
   for j = 1, 2, 3, ..., W
      if (j≥w[i] and P[i-1][j]<P[i][j-w[i]]+p[i]) //(*)
        P[i][j] = P[i][j-w[i]]+p[i];
   else
        P[i][j] = P[i-1][j];</pre>
```

(*) – sprawdzamy, co się bardziej opłaca: jeśli masa plecaka jest nie mniejsza niż masa i-tego przedmiotu oraz opłaca się dołożyć i-ty przedmiot – dokładamy go, zwiększając odpowiednio wartość plecaka.

Programowanie dynamiczne

- Metoda stosowana zwykle do problemów optymalizacyjnych, jej istotą jest obliczenie rozwiązania wszystkich podproblemów optymalizacyjnych, zaczynając od problemów małych, a kończąc na większych.
- Nie potrafimy przewidzieć, z których rozwiązań mniejszych podproblemów będziemy korzystać w kolejnych krokach i dlatego przygotowujemy się na każdą ewentualność – rozwiązujemy wszystkie mniejsze podproblemy.
- Obliczone rozwiązania są najczęściej zapisywane w tablicy, dzięki czemu dostęp do nich jest bardzo szybki.

Programowanie dynamiczne

- Programowanie dynamiczne poprawia inne metody (np. dziel i zwyciężaj) w sytuacji, kiedy wymagają one wielokrotnego liczenia rozwiązań tych samych podproblemów.
- Żaden z podproblemów nie jest rozwiązywany wielokrotnie.
- Metoda dzieli problem na podproblemy w taki sposób, by rozwiązanie optymalne było łatwo osiągalne z optymalnych rozwiązań podproblemów.

Programowanie dynamiczne a problem plecakowy

 Mniejsze podproblemy odpowiadają mniejszej wadze plecaka i mniejszej liczbie przedmiotów.

Zasady optymalności Bellmana

Na każdym kroku podejmować najlepszą decyzję z uwzględnieniem stanu wynikającego z poprzednich decyzji.

Złożoność

- Algorytm zachłanny: $O(n \log_2 n)$
- Programowanie dynamiczne: nW
 - zależy od liczby danych i wartości jednej z nich (co jest charakterystyczne dla programowania dynamicznego),
 - algorytm efektywny, jeśli W nie jest za duże.

Jak poznać numery przedmiotów?

- Można utworzyć dodatkową tablicę Q, skojarzoną z tablicą P.
- Można próbować odzyskać numery wybranych przedmiotów analizując tablicę P.

Symulacja

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 3 0 2 7 9 14 16 21 23 28 30 35 37 42 44 49 51 56 58 63 65 70 72 77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3 0 2 7, 9 14 16 21 23 28 30 35 37 42 44 49 51 56 58 63 65 70 72 77	1	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
	2	0	2	4	6	8/	10	12	14	16\	18	20	22	24	26	↑ 28	30 ,	∧ 32	34	∧ 36	38	40 /	42	44	46
	3	0	2	7	. 9	14	, 16	21	23	28					44	49	51	56	58	63	65	<u>↑</u> 70	, 72 ,	77	↑ 79
4 0 2 10 17 1 24 31 5 38 45 45 52 50 59 66 70 73 77	4	٥	2		10		17		24		31	\$	38	X	45	A	52	58	59		66	√70 △	73	77	80

Algorytm odzyskiwania numerów przedmiotów

```
i = n
j = W
while (j > 0)
    if (P[i][j] == P[i-1][j])
        i = i-1
    else
        wypisz nr i
        j = j - w[i]
```