Programowanie dynamiczne

- rozwiązujemy problem poprzez złożenie rozwiązań odpowiednich podproblemów
- każdy podproblem rozwiązujemy tylko raz, rozwiązanie zapamiętując w tabeli
- zazwyczaj stosowane do rozwiązywania problemów optymalizacyjnych: przy danych ograniczeniach mamy zmaksymalizować lub zminimalizować pewien koszt

Programowanie dynamiczne

- podczas rozwiązywania możemy wyszczególnić cztery etapy:
- scharakteryzowanie struktury optymalnego rozwiązania
- rekurencyjne rozwiązanie kosztu optymalnego rozwiązania
- obliczenie optymalnego kosztu metodą wstępującą – rozpoczynając od najmniejszych podproblemów, rozwiązujemy coraz większe, wykorzystując zapamiętane rozwiązania
- konstruowanie optymalnego rozwiązania na podstawie wyników wcześniejszych obliczeń

Programowanie dynamiczne

- jeżeli interesuje nas sam koszt rozwiązania optymalnego, krok czwarty można pominąć
- w przeciwnym przypadku, często opłaca się zapamiętywać dodatkowe informacje podczas wykonywania kroku trzeciego

Problem plecakowy

- mamy plecak o określonej pojemności W oraz zestaw n przedmiotów, ktore możemy do tego plecaka włożyć
- każdy z przedmiotów ma określoną wagę w_i oraz cenę c_i
- chcemy do plecaka zapakować przedmioty o jak największej łącznej wartości
- problem rozwiążemy programowaniem dynamicznym

Problem plecakowy

- celem naszym jest maksymalizacja $\sum_{i=1}^{n} d_i c_i$ przy zachowaniu ograniczenia $\sum_{i=1}^{n} d_i w_i \leq W$, gdzie d_i to zmienna decydująca, czy przedmiot i wkładamy ($d_i = 1$) bądź nie ($d_i = 0$) do plecaka

Problem plecakowy

- założmy, że znamy optymalne rozwiązanie dla danej instancji problemu
- weźmy jeden z przedmiotów, który trafił do plecaka (niech będzie to przedmiot k)
- pozostała zawartość plecaka jest optymalnym rozwiązaniem problemu plecakowego dla pojemności plecaka $W - w_k$ i zestawu przedmiotów $\{1, 2, \ldots, k-1, k+1, k+2, \ldots n\}$
- gdyby było inaczej (nie byłoby to optymalne rozwiązanie), to zastępując przedmioty w pozostałej części plecaka tymi z optymalnego rozwiązania podproblemu, otrzymalibyśmy rozwiązanie lepsze niż optymalne

Problem plecakowy

- zdefiniujmy rekurencyjnie koszt optymalnego rozwiązania:
- koszt rozwiązania poblemu dla wagi W i zestawu przedmiotów $\{1, 2, \ldots, j\}$ to:

$$K(j, W) = \begin{cases} 0 \text{ dla } j = 0 \\ 0 \text{ dla } W = 0 \\ max(K(j-1, W-w_j) + c_j, K(j-1, W)) \end{cases}$$

Obliczanie kosztu metodą wstępującą

- znamy koszty rozwiązań dla przypadków brzegowych ($W = 0$ lub pusty zbiór przedmiotów)
- dzięki przedstawionemu wzorowi możemy kolejno wyznaczać rozwiązania większych podproblemów, aż dojdziemy do rozwiązania całego problemu (wyznaczymy $K(n, W)$)

Problem plecakowy

K - tablica o wymiarach $n + 1 \times W + 1$
for $i = 0, \ldots, n$ **do** $K[i, 0] = 0$
for $i = 0, \ldots, W$ **do** $K[0, W] = 0$
for $i = 1, \ldots n$ **do**
  for $j = 1, \ldots W$ **do**
    $K[i, j] = max(K[i - 1, j - w_i] + c_i, K[i - 1, j])$
  end for
end for
return $K[n, W]$

Problem plecakowy

- rozmiar plecaka $W = 10$
- przedmioty: $w_1 = 5, c_1 = 1$, $w_2 = 3, c_2 = 2$, $w_3 = 3, c_3 = 1$, $w_4 = 4, c_4 = 4$, $w_5 = 1, c_5 = 1$

Pakowanie plecaka

i \ W	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0										
2	0										
3	0										
4	0										
5	0										

Pakowanie plecaka

W

i		0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	1	1	1	1	1	1
	2	0										
	3	0										
	4	0										
	5	0										

Pakowanie plecaka

W

i	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0										
4	0										
5	0										

Pakowanie plecaka

i \ W	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0										
5	0										

Pakowanie plecaka

W

i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0	0	1	2	4	4	5	6	6	7	7
5	0										

Pakowanie plecaka

i \ W	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0	0	1	2	4	4	5	6	6	7	7
5	0	1	1	2	4	5	5	6	7	7	8

Problem plecakowy

- złożoność algorytmu wynosi $O(nW)$ (nie jest wielomianowa!)
- wyznaczyliśmy wartość plecaka, jak poznać jego zawartość?

Problem plecakowy

- zaczynamy od $K[n, W]$
- porównujemy $K[n, W]$ z $K[n-1, W]$ oraz $K[n-1, W-w_n] + c_n$
- jeżeli $K[n, W] = K[n-1, W]$ to nie włożyliśmy przedmiotu n do plecaka, przechodzimy do $K[n-1, W]$ i potwarzamy operację
- jeżeli $K[n, W] = K[n-1, W-w_n] + c_n$ to włożyliśmy przedmiot n do plecaka, przechodzimy do $K[n-1, W-w_n]$ i potwarzamy operację

Pakowanie plecaka

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0	0	1	2	4	4	5	6	6	7	7
5	0	1	1	2	4	5	5	6	7	7	8

Pakowanie plecaka

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0	0	1	2	4	4	5	6	6	7	7
5	0	1	1	2	4	5	5	6	7	7	8

Pakowanie plecaka

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0	0	1	2	4	4	5	6	6	7	7
5	0	1	1	2	4	5	5	6	7	7	8

Pakowanie plecaka

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0	0	1	2	4	4	5	6	6	7	7
5	0	1	1	2	4	5	5	6	7	7	8

Pakowanie plecaka

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0	0	1	2	4	4	5	6	6	7	7
5	0	1	1	2	4	5	5	6	7	7	8

Pakowanie plecaka

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1	1	1
2	0	0	0	2	2	2	2	2	3	3	3
3	0	0	1	2	2	3	3	3	3	3	4
4	0	0	1	2	4	4	5	6	6	7	7
5	0	1	1	2	4	5	5	6	7	7	8

Problem plecakowy

- jeżeli interesuje nas sama wartość rozwiązania, zamiast tablicy o wymiarach $n+1 \times W+1$ wystarczy nam tablica o wymiarach $2 \times W+1$
- w i-tym kroku wystarczy pamiętać tylko poprzedni wiersz tablicy

Problem plecakowy

K - tablica o wymiarach $2 \times W + 1$

for $i = 0, \ldots, W$ **do** $K[0, W] = 0$

for $i = 1, \ldots n$ **do**

  for $j = 1, \ldots W$ **do**

    $K[i\%2, j] =$

    $max(K[1 - (i\%2), j - w_i] + c_i, K[1 - (i\%2), j])$

  end for

end for

return $K[n\%2, W]$

Problem plecakowy

- dlaczego nie wystarczy tablica jednowierszowa?
- musimy znać wartość $K[i-1, j-w_i]$
- mając tylko jeden wiersz tablicy, obliczając $K[i,j]$ w $K[i, j-w_i]$ będzie optymalne rozwiązanie dla podproblemu o rozmiarze plecaka $j - w_i$ i przedmiotów $\{1, 2, \ldots, i\}$ (a nie $\{1, 2, \ldots, i-1\}$)
- jeżeli optymalne rozwiązanie podproblemu będzie wymagało włożenia przedmiotu i do plecaka, może się zdarzyć, że przedmiot ten włożymy do plecaka wielokrotnie
- chyba, że będziemy wypełniać tablicę w przeciwnym kierunku...

Minimalna odległość edycyjna

- dane są dwa słowa A i B
- dozwolone są operacje: wstawienie symbolu, usunięcie symbolu, zamiana symbolu na inny
- ile minimalnie operacji należy wykonać, aby przekształcić słowo A w słowo B
- problem ten możemy rozwiązać stosując programowanie dynamiczne

Minimalna odległość edycyjna

- rozważmy prefiks słowa A długości i i prefiks słowa B długości j
- minimalna odległość edycyjna pomiędzy $A[1,\ldots,i]$ a $B[1,\ldots,j]$ to mniejsza z:
- $OE(A[1,\ldots i-1], B[1,\ldots j]) + 1$ – usunięcie znaku
- $OE(A[1,\ldots i], B[1,\ldots j-1]) + 1$ – dołączenie znaku
- $OE(A[1,\ldots i-1], B[1,\ldots j-1])$ jeżeli $A[i] = B[j]$
- $OE(A[1,\ldots i-1], B[1,\ldots j-1]) + 1$ jeżeli $A[i] \neq B[j]$ – zamiana znaku

Minimalna odległość edycyjna

K - tablica o wymiarach $m + 1 \times n + 1$
for $i = 0, \ldots, m$ **do** $K[i, 0] = i$
for $j = 0, \ldots, n$ **do** $K[0, j] = j$
for $i = 1, \ldots n$ **do**
  for $j = 1, \ldots m$ **do**
    if $A[i] = B[j]$ **then**
      $K[i, j] = K[i - 1, j - 1]$
    else

$$K[i, j] = \min \begin{cases} K[i-1, j] + 1, \\ K[i, j-1] + 1, \\ K[i-1, j-1] + 1 \end{cases}$$

    end if
  end for
end for
return $K[m, n]$