Algorytmy i Struktury Danych

Wykład 12

Problem plecakowy (ang. Knapsack)

Dane: $I = \{0, \dots, n-1\}$ - przedmioty

$w : I \to \mathbb{N}$ - wagi

$p : I \to \mathbb{N}$ - cena

$B \in \mathbb{N}$ - maksymalna waga

Zadanie: Znaleźć podzbiór przedmiotów, których łączna waga nie przekracza B i których łączna cena jest maksymalna

① Funkcja do obliczania

$f(i,b)$ = maksymalna suma cen przedmiotów ze zbioru $\{0, \dots, i\}$, których łączna waga nie przekracza b

② Sformułowanie rekurencyjne

$$f(i,b) = \max\Big(\overbrace{f(i-1,b)}^{\text{nie wzięty } i\text{-go}},\ \overbrace{f(i-1, \underbrace{b-w(i)}_{\geq 0}) + p(i)}^{\text{wzięty } i\text{-ty przedmiot}} \Big)$$

(a gdyby wyszło < 0 to pomijamy ten człon)

$$f(0,b) = \begin{cases} p(0) & , w(0) \leq b \\ 0 & , w(0) > b \end{cases}$$

③ Implementacja

Odtwarzanie rozwiązania

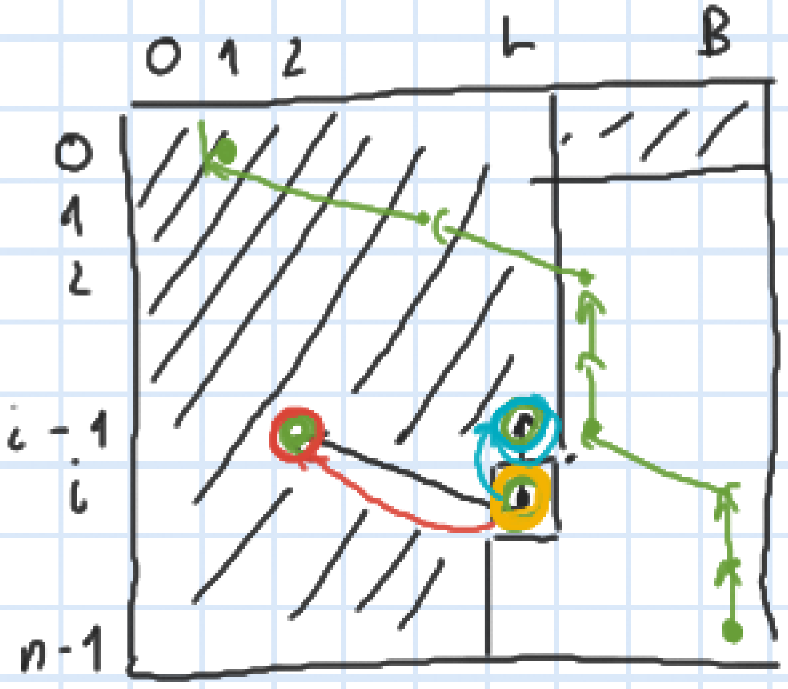
P - parent

```
def knapsack(W, P, B):
    n = len(W)
    F = [[0 for b in range(B+1)] for i in range(n)]

    for b in range(W[0], B+1):
        F[0][b] = P[0]

    for b in range(B + 1):
        for i in range(1, n):
            F[i,b] = F[i-1][b]
            if b - W[i] >= 0:
                F[i][b] = max(F[i][b], F[i-1][b-W[i]] + P[i])

    return F[n-1][B]
```

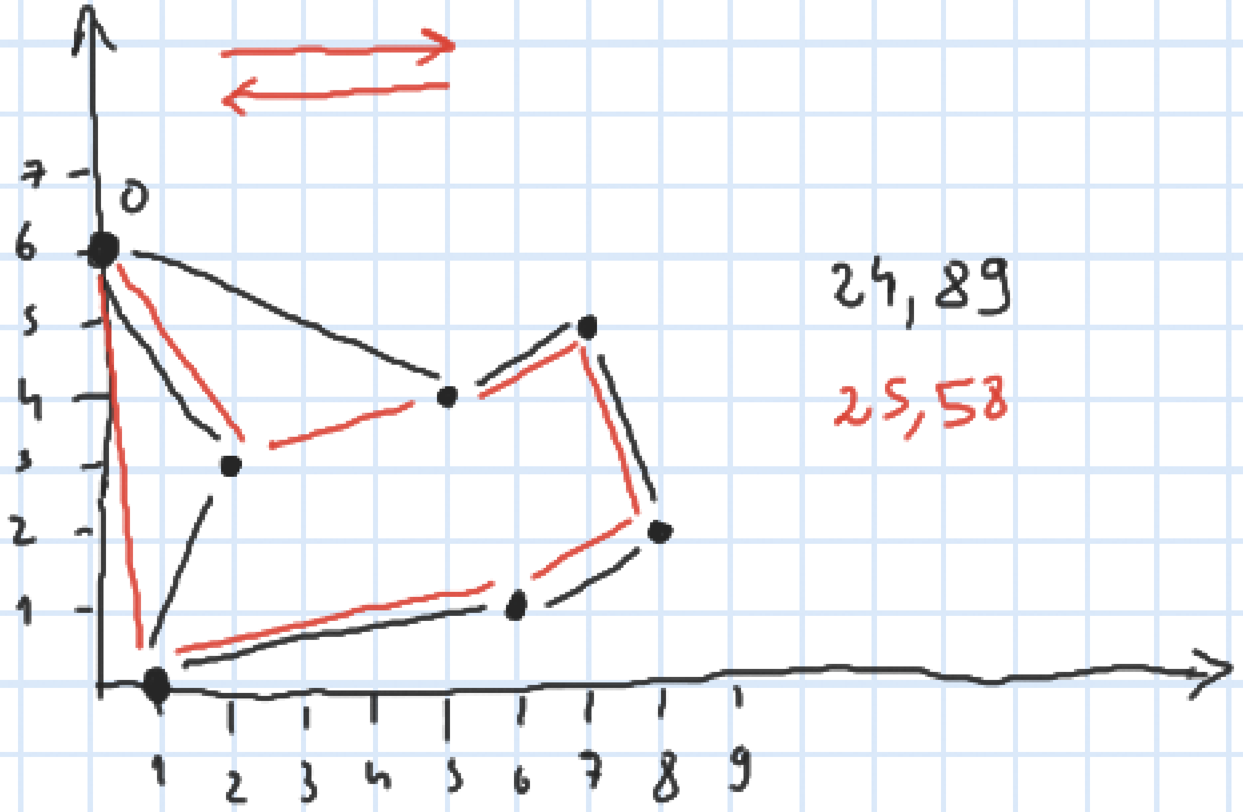


Problem komiwojażera (travelling salesperson problem, TSP)

Dane: $C = \{0, \dots, n-1\}$

$d : C \times C \to \mathbb{R}$ – odległość między miastami

Zadanie: Znaleźć kolejność odwiedzania miast, tak by zacząć od 0, skończyć w 0, odwiedzić wszystkie miasta minimalizując sumę odległości między kolejnymi



Algorytm Brute-Force

$O(n!)$ – pełny przegląd

Algorytm dynamiczny w przypadku ogólnym

① $f(S,t)$ = długość najkrótszej trasy, która startuje w 0, odwiedza wszystkie miasta ze zbioru S i kończy w mieście t $(0 \in S,\ t \in S)$

② $f(S,t) = \min\limits_{r \in S - \{t\}} f(S - \{t\}, r) + d(r,t)$

$f(\{0\}, 0) = 0$

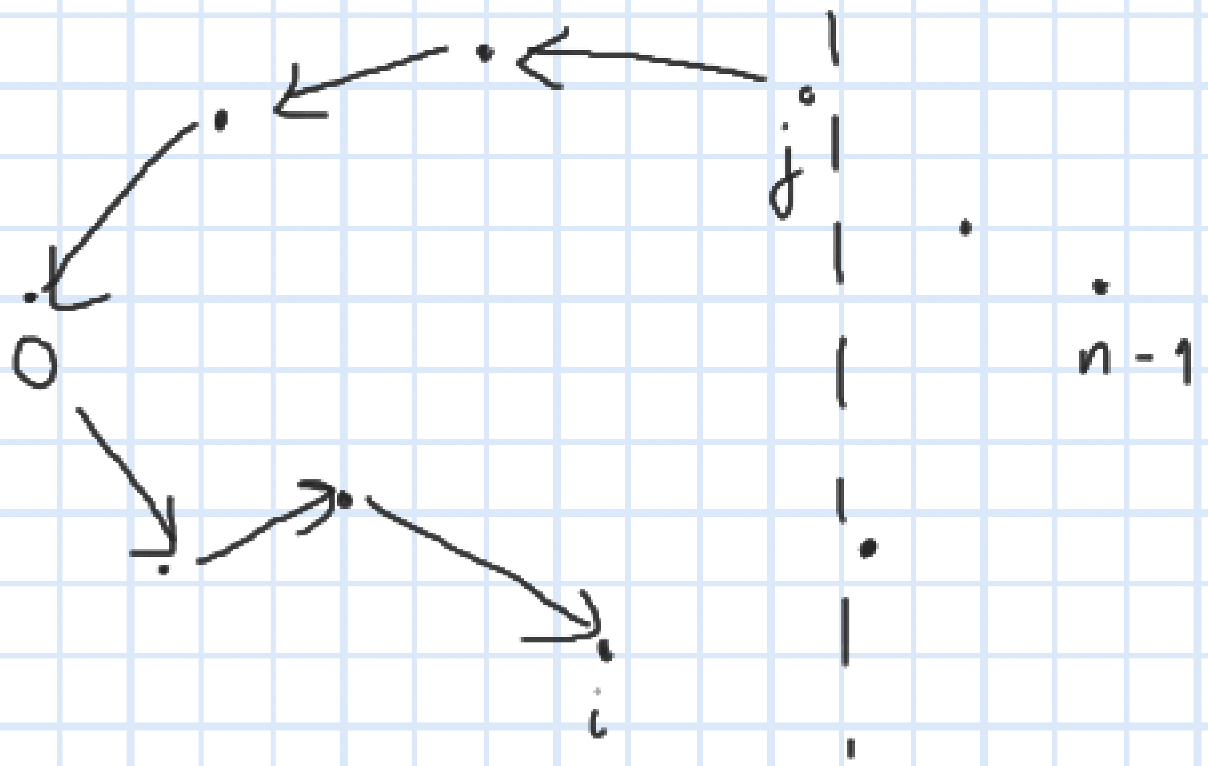
$O(2^n \cdot n^2)$

Odczytywanie wyniku

$$\min_{t \in \{1, \dots, n-1\}} f(C,t) + d(t,0)$$

Problem komiwojażera (wersja bitoniczna)

(współrzędne x parami różne)



① $f(i, j)$ = koszt ścieżek z 0 do i oraz
z 0 do j

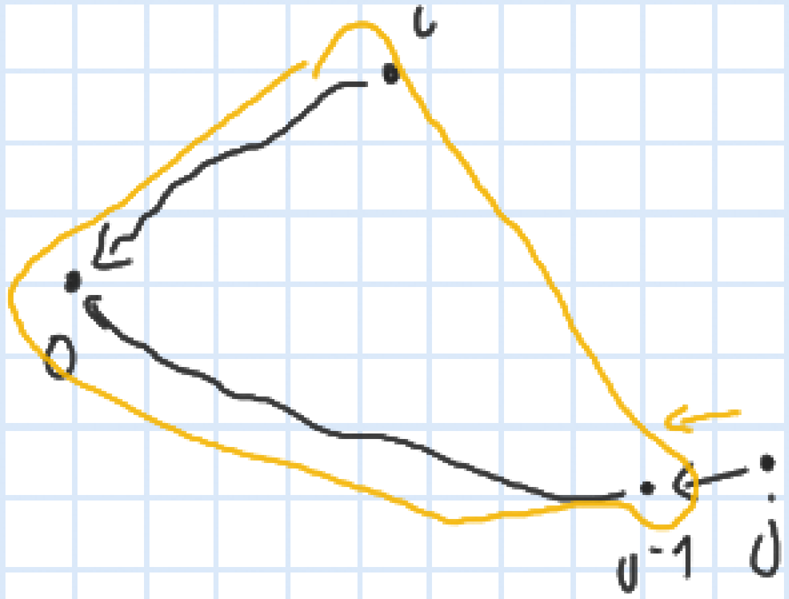
$i < j$

które odwiedzają wszystkie miasta
$0, 1, \dots, j$ i żadnego nie powtarzają

$f(0, 1) = d(0, 1)$

② Zapis rekurencyjny

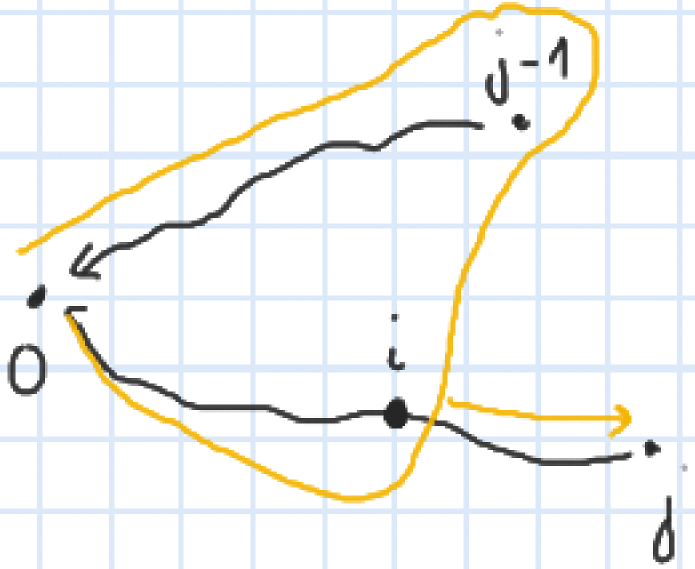
a)



$i < j-1$

$f(i,j) = f(i,j-1) + d(j-1,j)$

b)



$f(j-1,j) = \min\limits_{i<j-1} \left(f(i,j-1) + d(i,j) \right)$

③ Implementacja (rekurencja ze spamiętywaniem)

$D[i][j] = d(i,j)$

$F[i][j] = [[\text{inf}] \times n \text{ for } i \text{ in range}(n)]$

$F[0][1] = D[0][1]$

```
def tspf(i, j, F, D):
    if F[i][j] ≠ inf: return F[i][j]
    if i = j-1:
        best = inf
        for k in range(j-1):
            best = min(best, tspf(k, j-1, F, D) + D[k][j])
        F[j-1][j] = best
    else:
        F[i][j] = tspf(i, j-1, F, D) + D[j-1][j]
    return F[i][j]
```

Złożoność

$O(n^2)$