

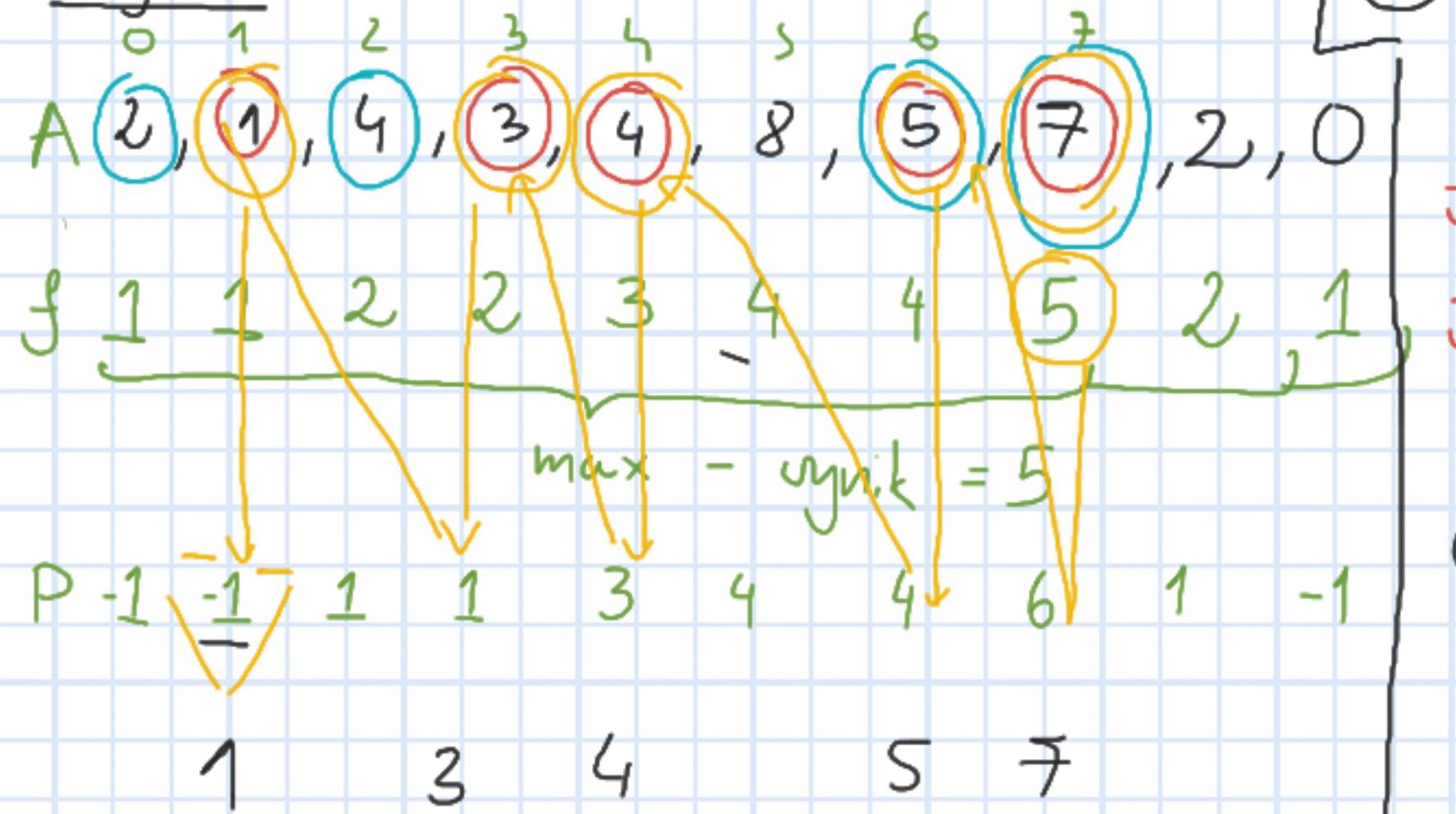
# ASD - Użytkad 6

Najdłuzszy rosnący podciąg

Dane:  $A[0, \dots, n-1]$  - tablica liczb

Zadanie: Znaleźć długość najdłuzszego (niekoniecznie spójnego) rosnącego podciągu.

## Prykład



① Ustalamy funkcje, które będziemy obliczać

$f(i)$  = długość najdłuzszego rosnącego podciągu w tablicy  $A[0, \dots, i]$  kończącego się linką  $A[i]$

wynik:  $\max_{i \in \{0, \dots, n-1\}} f(i)$

② Wyrażenie funkcji  $f$  w postaci rekurencyjnej

$$f(i) = \max \{ f(j) + 1 \mid j < i \wedge A[j] < A[i] \}$$

$$f(0) = 1$$

konwencja:  $\max \emptyset = 1$

③ implementacja

## Implementacija

```
def lis(A):  
    n = len(A), maxi = 0  
    F = [1 for i in range(n)]  
    P = [-1 for i in range(n)]
```

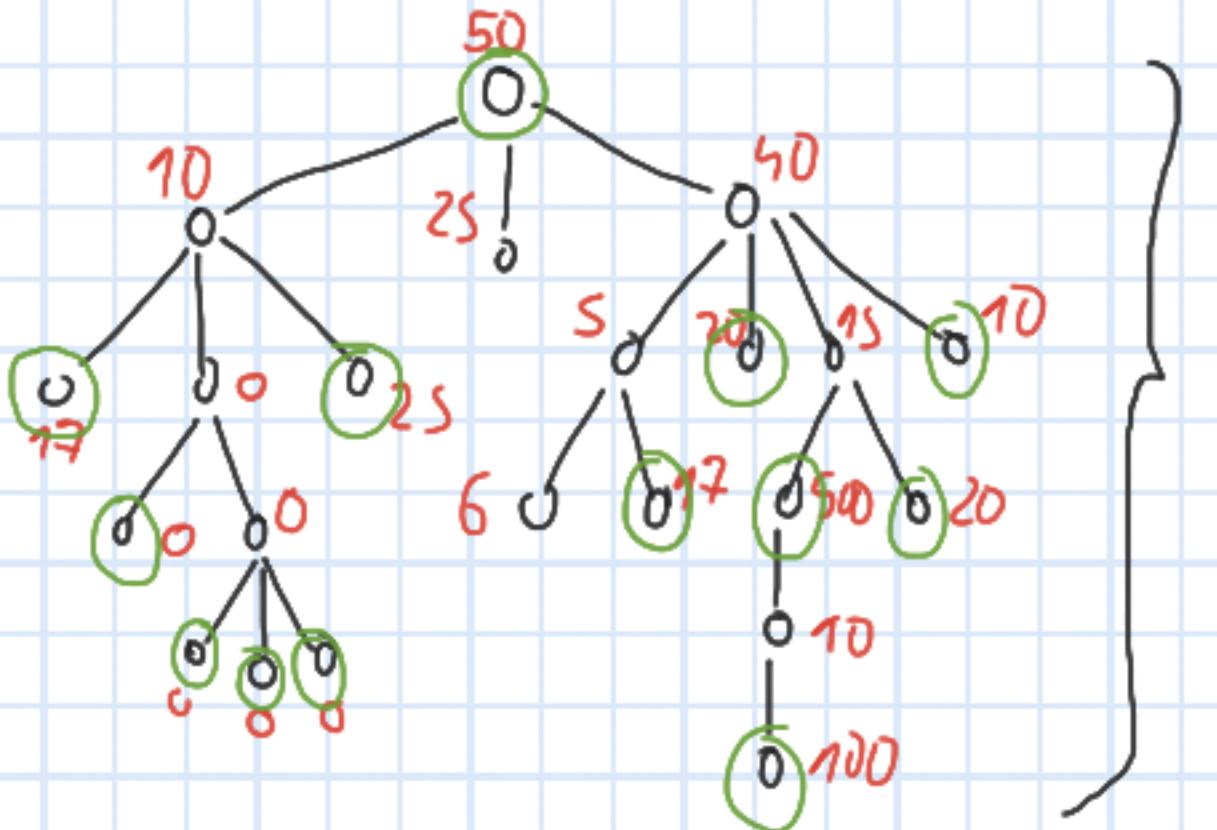
```
    for i in range(1, n):  
        for j in range(i):  
            if A[i] > A[j] and F[j]+1 > F[i]:  
                F[i] = F[j]+1  
                P[i] = j  
  
        if F[i] > F[maxi]:  
            maxi = i  
  
    return maxi, F, P
```

```
def printSol(A, P, i):  
    if P[i] != -1:  
        printSol(A, P, P[i])  
    print(A[:])
```

$O(n^2)$

$O(n \log n)$   
da si je raznigrađac

## Problem imprezy firmowej



impreza jest dopuszczalna  
jeśli dla każdego zaproszonego  
pracownika nie zaprosiliśmy  
jego bezpośredniego przełożonego

wartość imprezy jest sumą  
uspółmenników fun zaproszonych

class Employee:

def \_\_init\_\_(self, fun):

self.fun = fun

self.emp = []

self.f = -1

self.g = -1

zadanie: znaleźć wartość  
najlepszej dopuszczalnej  
imprezy

← tabela dzieci węzła

① Określenie obliczanych funkcji

v - węzeł dnia

f(v) - wartość najlepszej imprezy poddrzewa  
zakonserwowanego w v

g(v) - j.w. pod warunkiem, że v nie  
idzie na imprezę

wynik: f(root)

② Zalczności rekurencyjne

$$g(v) = \sum_{u-\text{pracownik } v} f(u)$$

u - pracownik v

$$f(v) = \max(g(v), \text{fun}(v) + \sum_{u-\text{pracownik } v} g(u))$$

### ③ Implementacija

```
def g( v ):  
    if v.g != -1: return v.g  
    v.g = 0  
    for u in v.emp:  
        v.g += f( u )  
    return v.g
```

```
def f(v):  
    if v.f != -1: return v.f  
    f1 = g(v)  
    f2 = v.fun  
    for u in v.emp:  
        f2 += g(u)  
    v.f = max( f1, f2 )  
    return v.f
```

# Problem plecakowy

Dane :  $I = \{0, \dots, n-1\}$  - przedmioty  
 $w: I \rightarrow \mathbb{N}$  - wagи  
 $p: I \rightarrow \mathbb{N}$  - ceny/profitы  
 $B \in \mathbb{N}$  - maks. waga

Zadanie : Znaleźć podzbiór  $I$  o maksymalnej

sumarycznej wadze i tążnej wadze nie  
przekraczającej  $B$

## ① Funkcja do obliczania

$f(i, b)$  = maksymalna suma cen  
przedmiotów ze zbioru  $\{0, \dots, i\}$   
nie przekraczających tążnej wagi  $b$

wynik:  $f(n, B)$

## ② Sformułowanie rekurencyjne

$$f(i, b) = \max \left( \begin{array}{l} f(i-1, b), \\ f(i-1, b-w(i)) + p(i) \end{array} \right)$$

nic liczący  $i$ -go przedmiotu  
 o ile  $\geq 0$   
 liczący  $i$ -ty przedmiot

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

### ③ Implementacja

```
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):
```

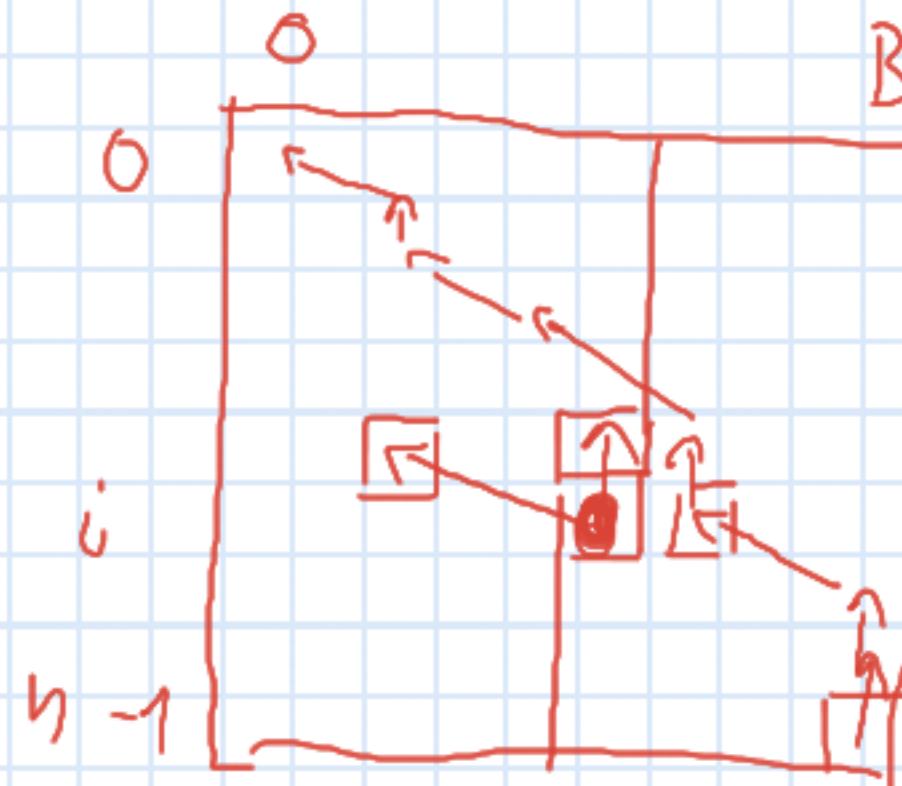
```
            F[i][b] = F[i-1][b]
```

```
            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]
```



## ASD - Wykład 7

Problem komiwojażera

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

$d : C \times C \rightarrow \mathbb{R}$  - metryka nad  $C$

Zadanie: Znaleźć trasę zatrzymającą się

w mieście  $O$ , przebiegającą przez

wszystkie inne miasta (przez każde

dokładnie raz) i wracającą do miasta  $O$

o minimalnej sumarycznej długości

## Algorytm brute-force

Spróbuj każdej kolejności odwiedzenia miast

$$\mathcal{O}(n \cdot n!)$$

efektywność czasowa

$$\mathcal{O}(n)$$

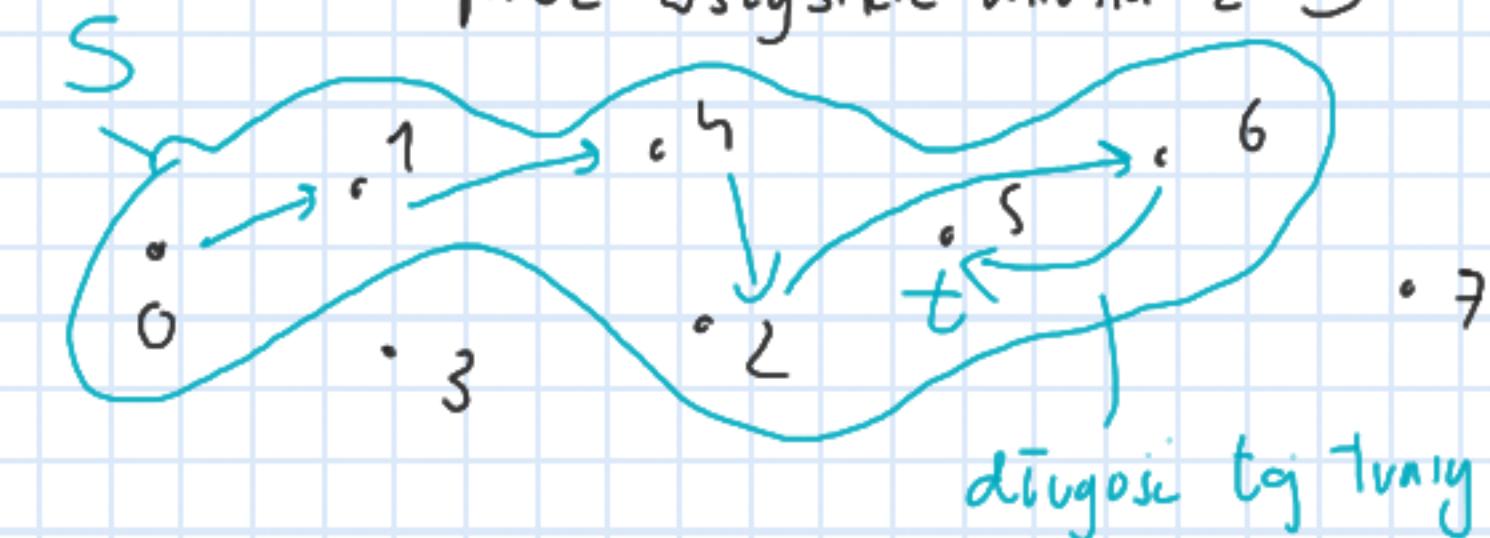
efektywność pamięciowa

## Algorytm dynamyczny

$S$  - podzbiór miast, taki że  $0 \in S$

$t \in S$  - miasto

$f(S, t) =$  długość (w sensie  $d$ ) najkrótszej trasy  
z miasta  $O$  do miasta  $t$  przebiegającej  
przez wszystkie miasta z  $S$



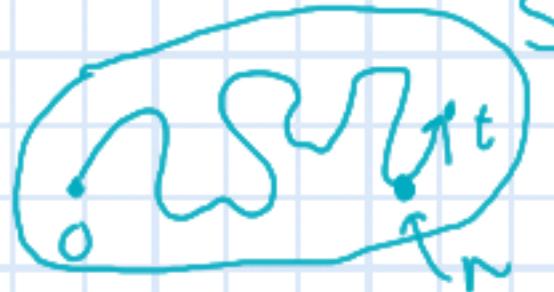
## Rozwiązywanie

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

## Sformułowanie rekurencyjne dla f

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

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



## Złożoność

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

crasova

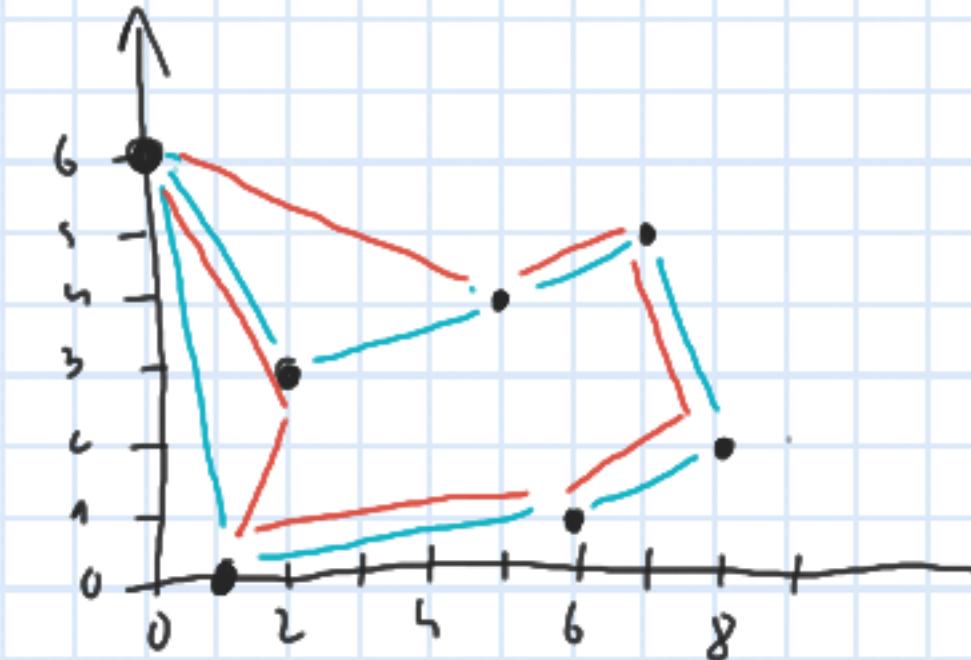
$$\Theta(n \cdot 2^n)$$

paragonowa

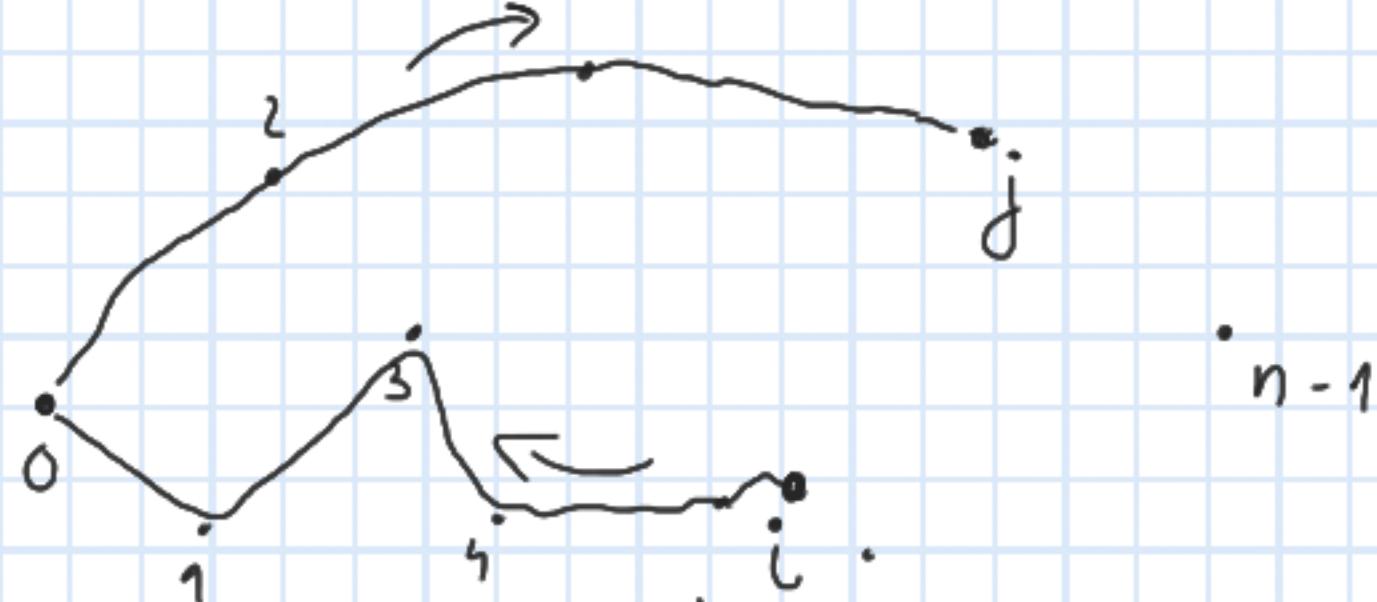
## Bitoniczny problem komiwojażera

Wersja problemu w 1D

- miasta to punkty w  $\mathbb{R}^2$  (żadne dwa miasta nie mają tej samej wsp. x)
- miasto 0 ma najmniejszą wsp. x
- szukamy trasy, która przebiega z lewa na prawo i z powrotem (kierunek na osi x zmieniamy raz)



## Algorytm dynamiczny

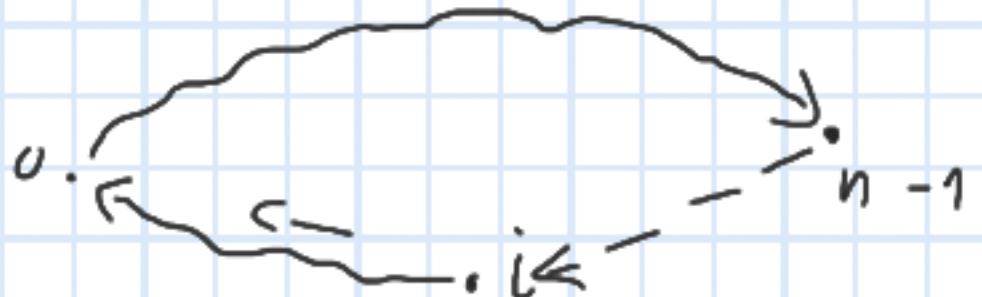


$f(i, j) = \begin{cases} \text{koszt suieku z } 0 \text{ do } i \text{ oraz} \\ \quad 2 \text{ } 0 \text{ do } j, \\ i < j \end{cases}$

które używają tanniej wizytacji miast  $\{0, \dots, j\}$ . ale żadnego nie moutaję

## Rozwiązywanie

$$\min_i (f(i, n-1) + d(i, n-1))$$

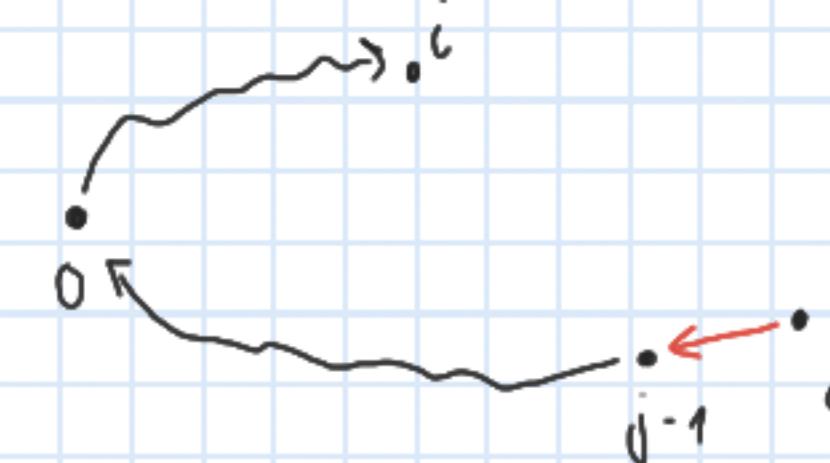


## Zapis rekurencyjny funkcji f

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

Następnie rozważamy dwa przypadki

a)



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

$$i < j-1$$

b)

$$f(j-1, j) = \min_{k < j-1} f(k, j-1) + d(k, j)$$



## Implementasi

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

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

```
def tspf(i, j, F, D):
```

```
    if F[i][j] != \infty: return F[i][j]
```

```
    if i == j - 1:
```

```
        best = \infty
```

```
        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]
```

# Algorytmy zuchanne (ang. greedy)

- podejmuj decyzje, które "w tej chwili" ulegają się najlepsze

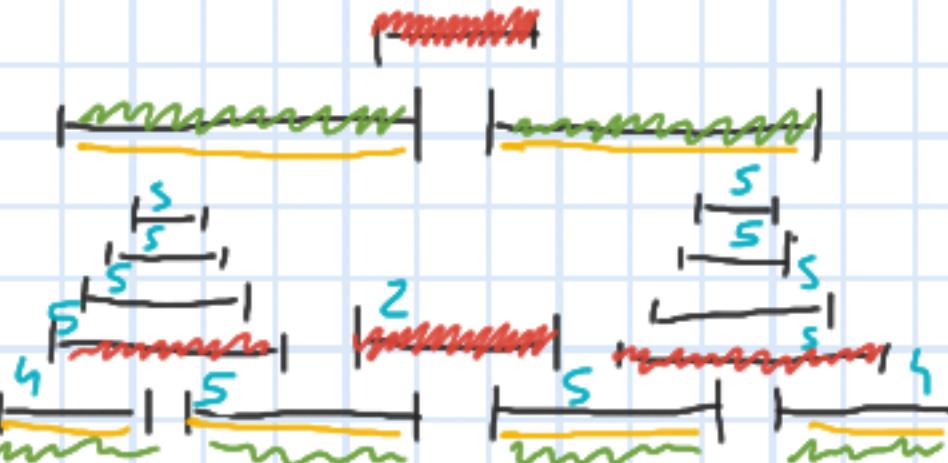
## Problem wyboru zadań

Dane: zbiór przedziałów (zadań)

Zadanie: Wybrać jak najwięcej przedziałów, które nie mają czasu wspólnego

## Pomyśły

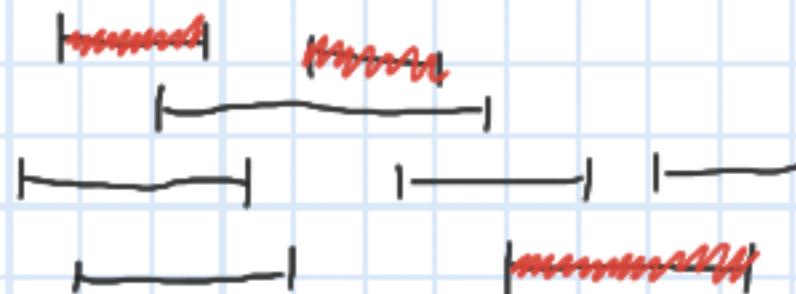
① "najkrótszy najpierw"



② "najmniej maści najpierw"



③ "najwcześniej koninący się najpierw"

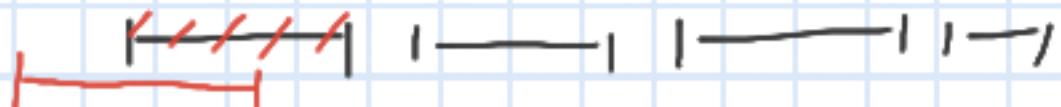


## Uzasadnienie poprawności

Rozważmy dowolne rozwiążanie optymalne

- jeśli zawiera najwcześniej koninący się przedział to OK

- jeśli nie zawiera to: dokładamy najwcześniej koninący się przedział i usuwamy to co przecina



## Ciągły problem plecakowy

Dane: substancje  $1, \dots, n$

dla każdej substancji i mamy takie

$v(i)$  - objętość i (dostępna)

$p(i)$  - wartość i

$B$  - "taką objętość, której możemy zabrać"

Zadanie: zdecydować jaką objętość każdej

substancji należy zabrać, aby ich

taką wartość była maksymalna i

nic przekroju  $B$

① "najcenniejsza substancja najpierw"

$p(i)$	2	1 1 1	..	1
$v(i)$	B	1 1 1		1

② "najmniejcie najpierw"

$p(i)$	0.01 0.01	B B	B
$v(i)$	1 .. 1	2 2	2

③ Dla każdej substancji oblicz

$$\alpha(i) = \frac{p(i)}{v(i)}$$

byliście najbardziej opłacalne najpierw

# ASD - Wykład 8

Problem: Suma Spójnego Podciągu

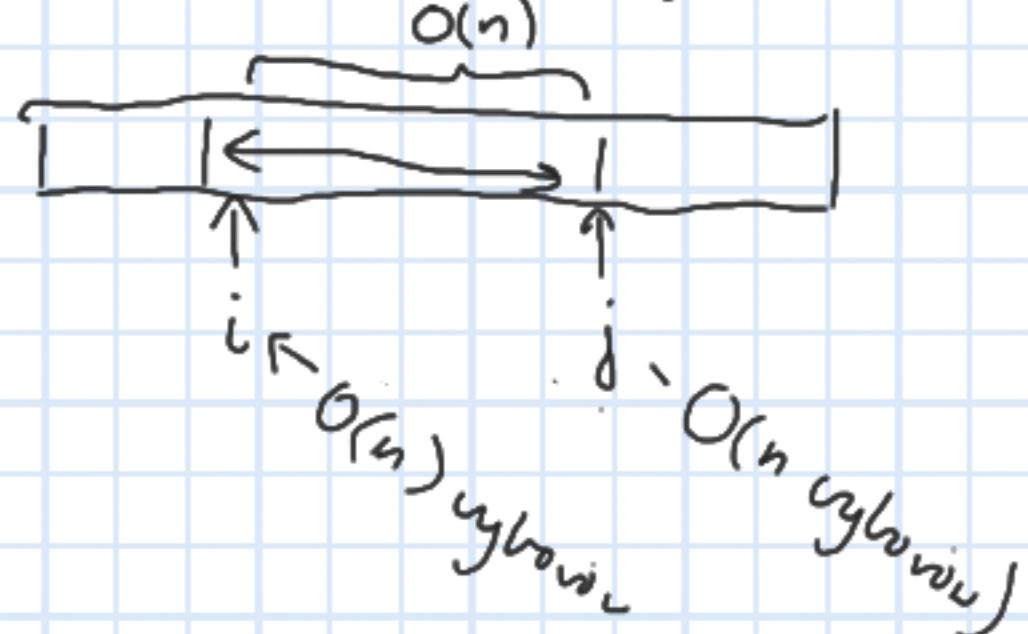
Dane:  $A[0, \dots, n-1]$  - tablica liczb całkowitych

Wynik:  $\max_{i,j} \left( \sum_{k=i}^j A[k] \right)$

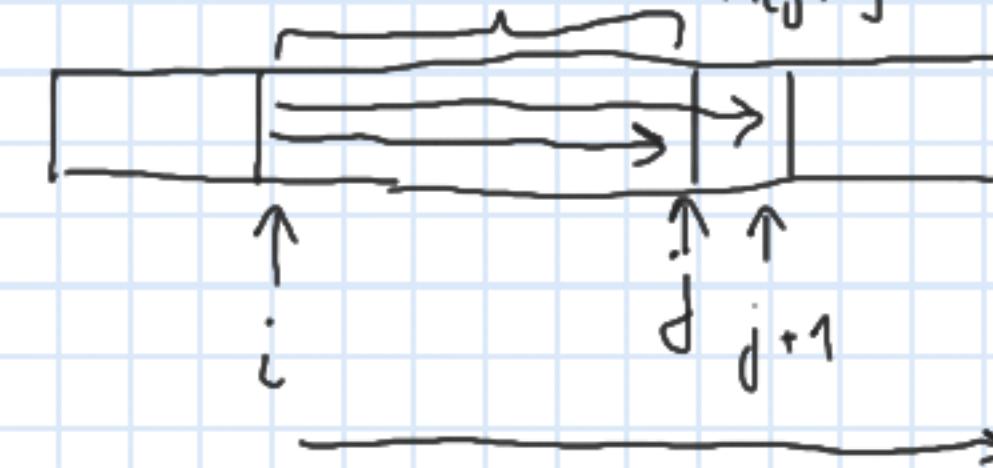
## Poniedziałek

1, 2, -5, 3, -1, 2, 1, -10, 2

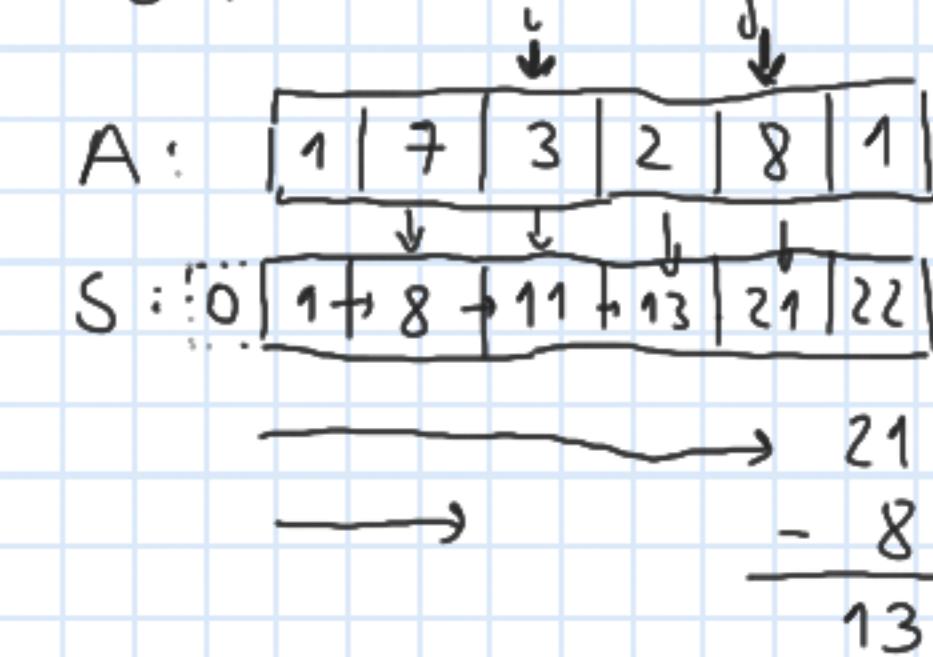
① Elementarne rozwiązywanie  $O(n^3)$



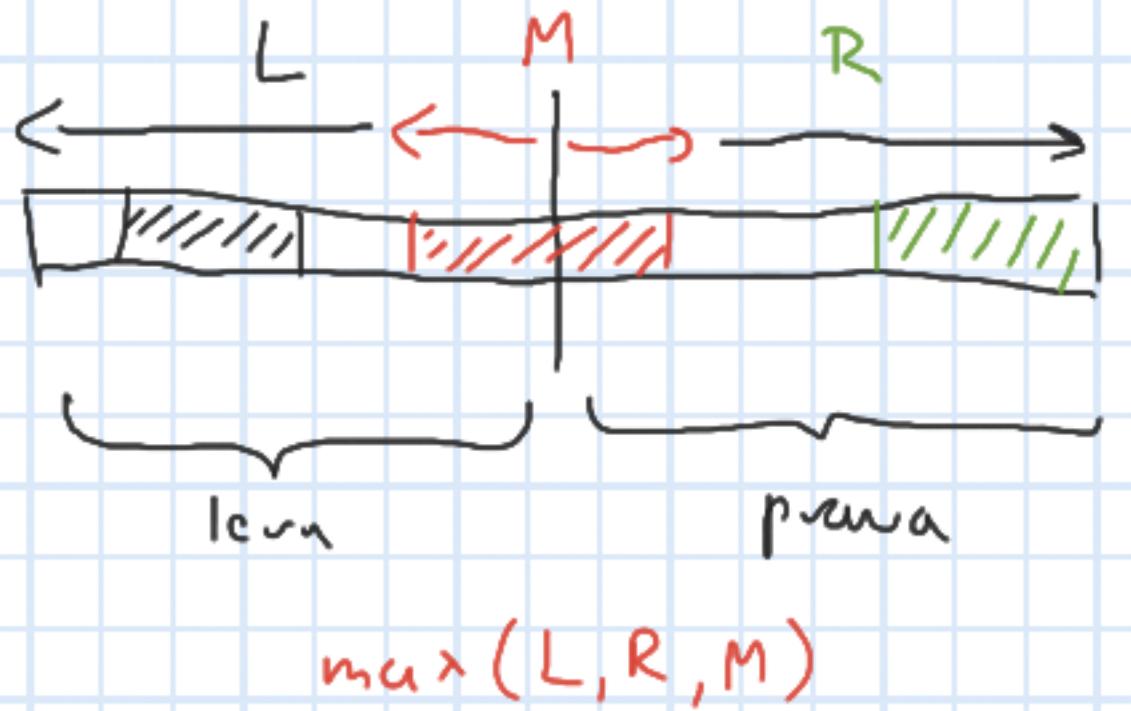
② Algorytm  $O(n^2)$  przez sumy prefiksowe



Sumy prefiksowe:



### ③ Algorytm dziel-i-zwyciążaj



$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + O(n) \\ &= O(n \log n) \end{aligned}$$

### ④ Programowanie dynamiczne sprzęego

$f(i)$  = największa możliwe suma ciągu konieclego się na  $A[i]$  (na  $i$ -tym elemencie ciągu)

$$f(0) = A[0]$$

$$f(i) = \max(f(i-1) + A[i], A[i])$$

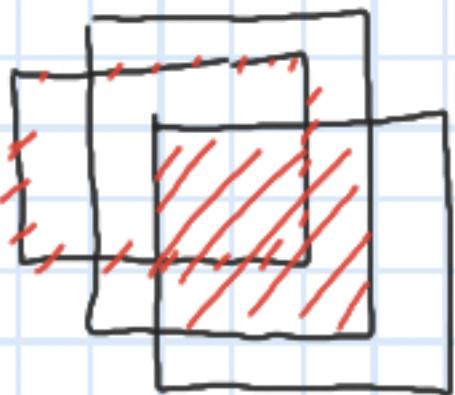
$$A: 1, 3, -7, -2, 1, 9, -3, 2, -100, 4$$

$$f: 1 \ 4 \ -3 \ -2 \ 1 \ 10 \ 7 \ 9, -91, 4$$

ale trzeba zachować!

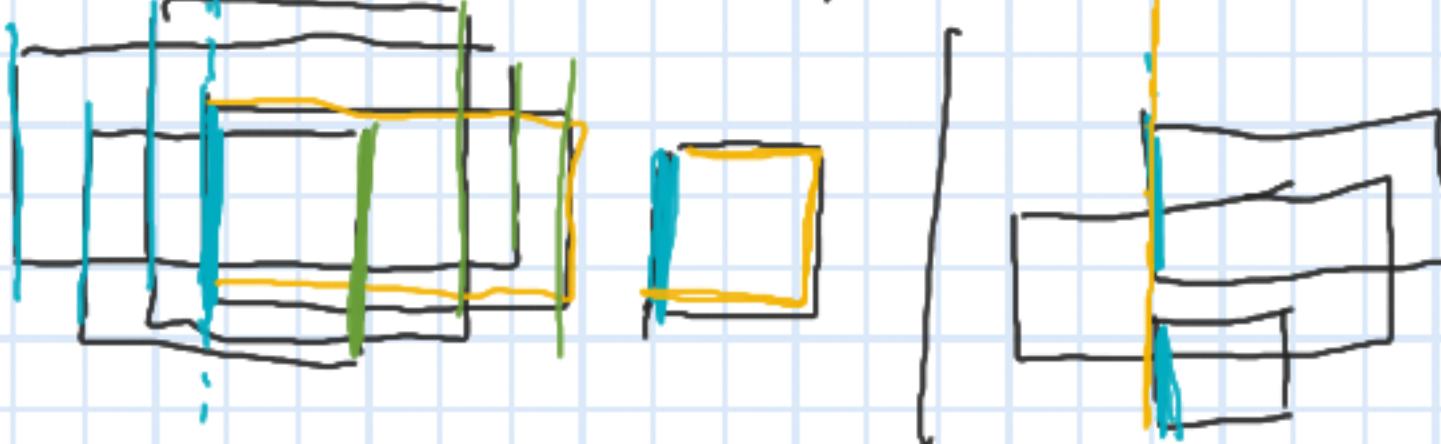
# Problem Pniewienia prostokątów

Dane: Ciąg prostokątów  $P_1, \dots, P_n$ , których  
boki są równoległe do osi układu  
osiów prostokątów



Zadanie: Wyhniac taki prostokąt, żeby  
pniesieć porozstały do miasta  
maksymalne pole

③  $O(n)$  z rozumieniem problemu



① Rozwiązywanie  $O(n^2)$

- przyjmujemy każdy prostokąt
- obliczamy przecięcie pozostałych

② Rozwiązywanie  $O(n)$

$$P_1, P_2, P_3, P_4, P_5, P_6, P_7$$

 $\mathbb{R}^2$ 

$$\left. \begin{array}{l} P_1 \\ P_1 \cap P_2 \\ P_1 \cap P_2 \cap P_3 \\ P_1 \cap P_2 \cap P_3 \cap P_4 \\ P_1 \cap P_2 \cap P_3 \cap P_4 \cap P_5 \end{array} \right\} A$$

$$P_1 \cap \dots \cap P_6$$

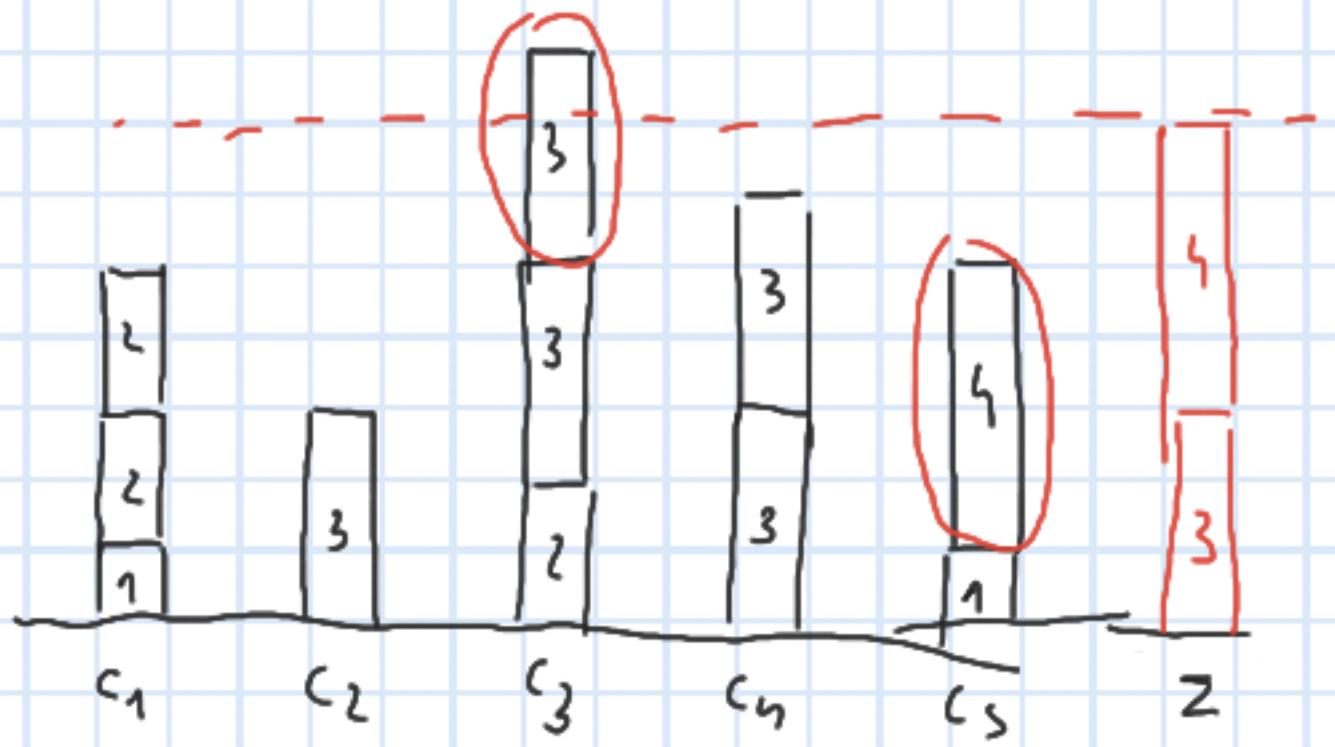
$$\left. \begin{array}{l} P_3 \cap P_4 \cap P_5 \cap P_6 \cap P_7 \\ P_4 \cap P_5 \cap P_6 \cap P_7 \\ P_5 \cap P_6 \cap P_7 \\ P_6 \cap P_7 \\ P_7 \end{array} \right\} B$$

 $\mathbb{R}$ 

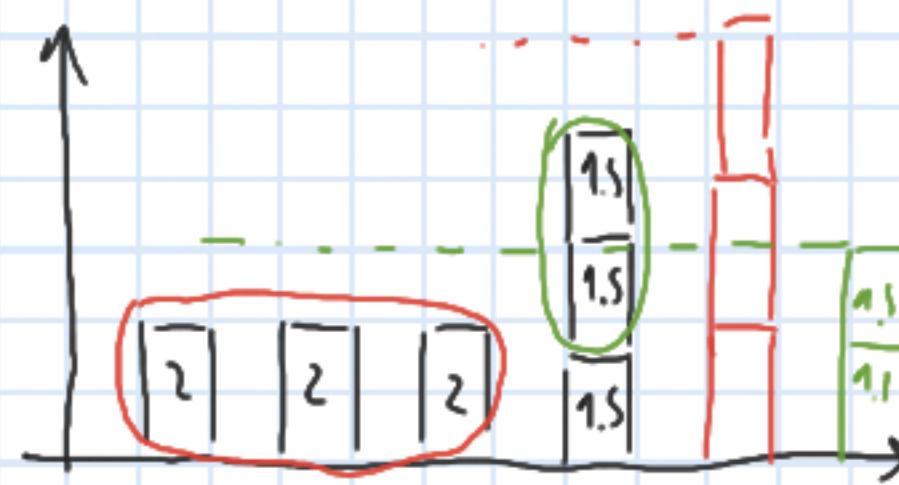
$$P_1 \cap \dots \cap P_6$$

## Problem najwyższej wieży

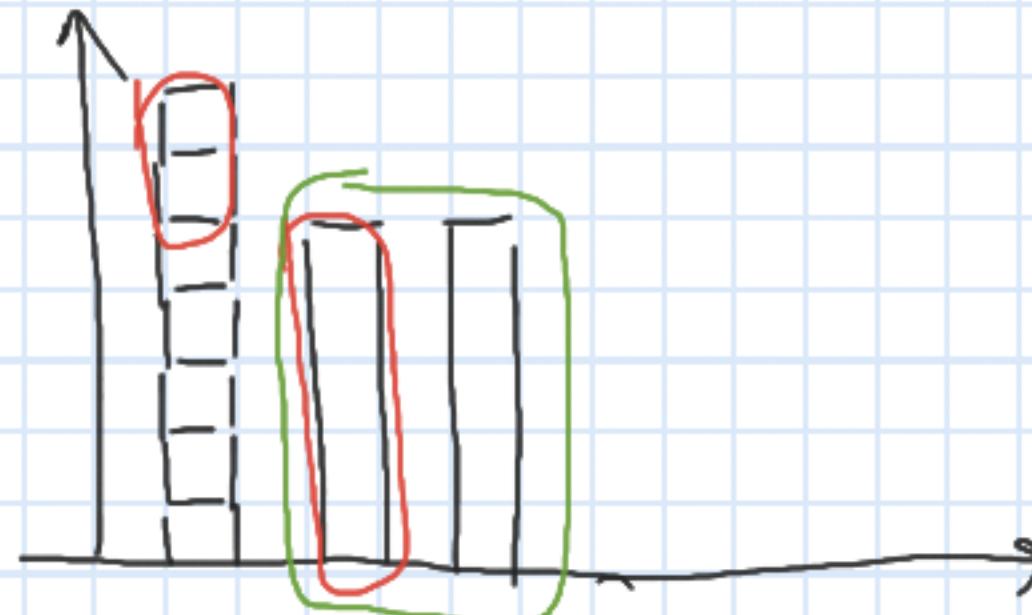
Dane:



① Kradnij największe klocki



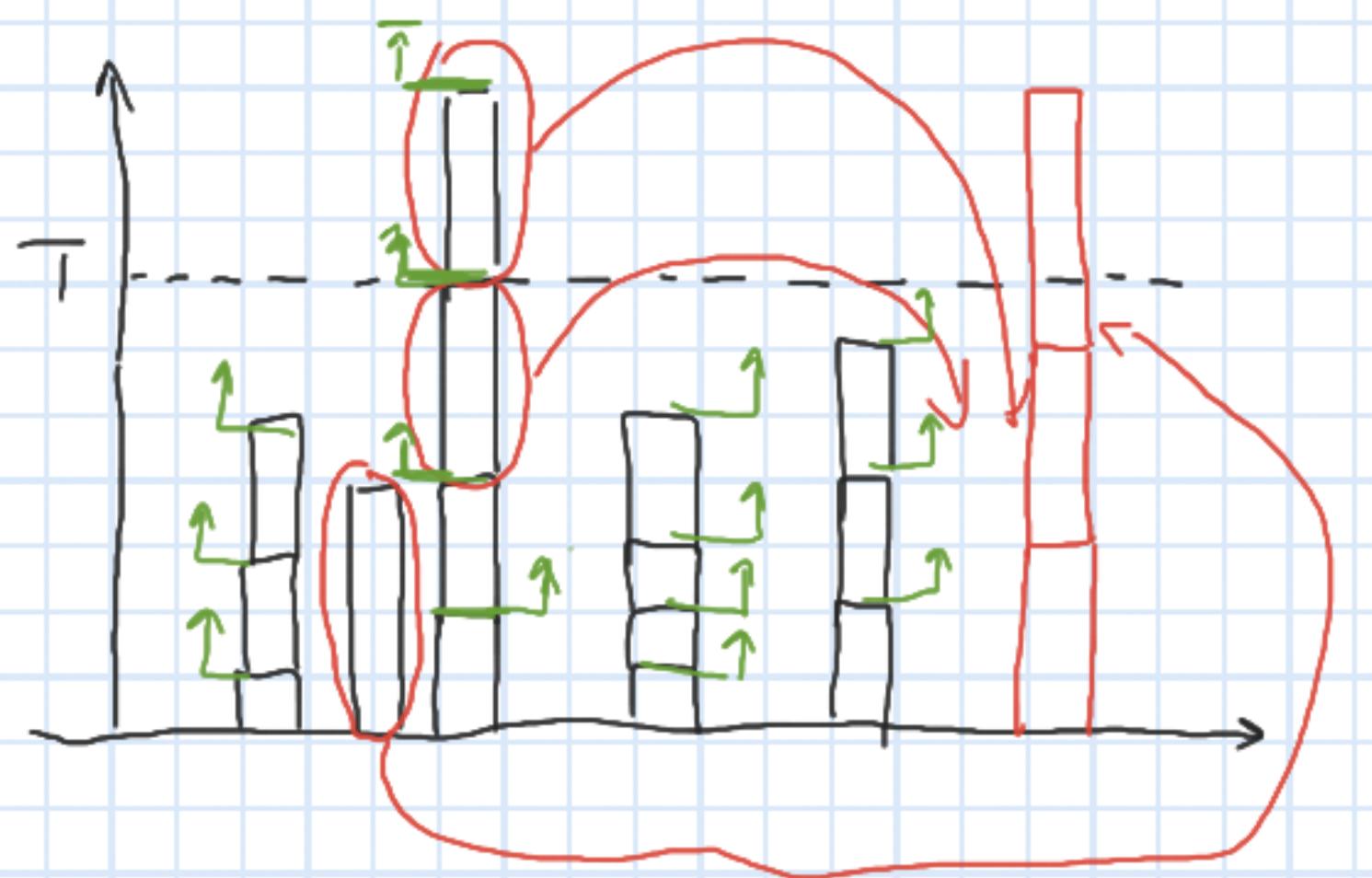
② Kradnij największe klocki z najwyższej wieży



Zadanie: uchwyc jutko najmniejszą liczbę klocków tak, żeby uzyskać najwyższą wieżę

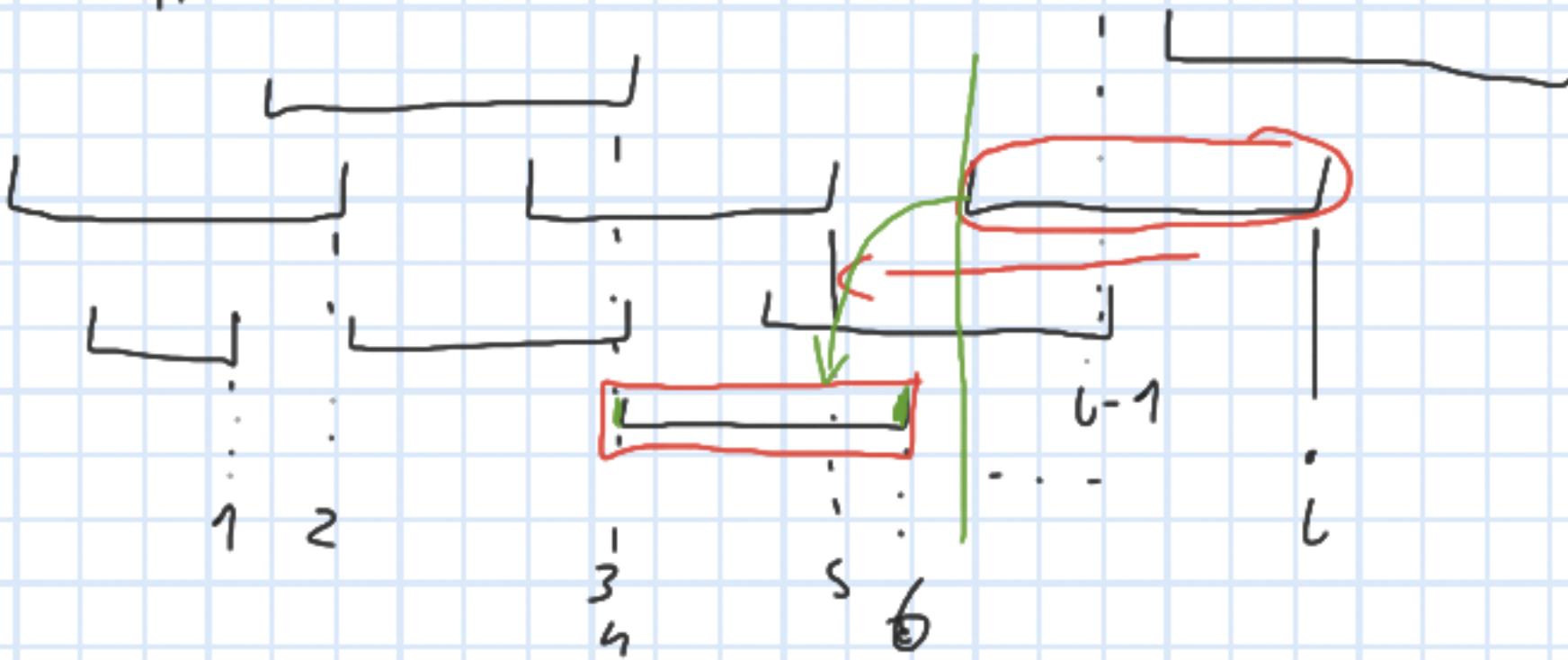
③

Miles heurystyk



## ASD - Wykiad 9

Zadanie offline 5



$f(i, t)$  = maks. pojemność budynków na terenie do końca  $i$ -go, kosztujących  $\leq t$

$$f(i, t) = \max \left( f(i-1, t), \text{pojemność}(i) + f(\text{prev}(i), t - \text{cena}(i)) \right)$$

$O(n^2 p)$

$O(n^2 + np)$

bud. j nie nadodzi na i

$O(n \log n + np)$

# ASD - Wykład 9

## Algorytmy grafowe

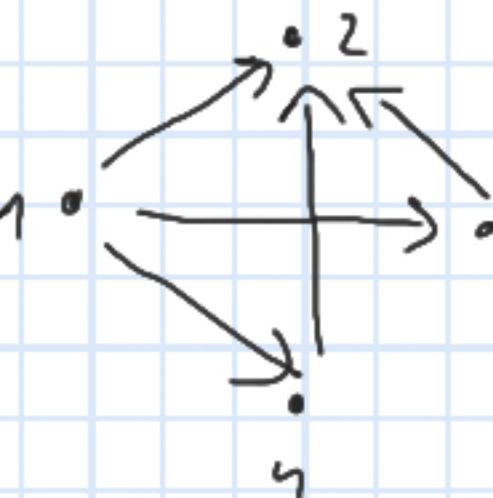
Graf skierowany:

$$G = (V, E)$$

$V = (v_1, \dots, v_n)$  - zbiór wierzchołków

$$E = \{e_1, \dots, e_m\}, E \subseteq V \times V$$

na ogół nie dopuszcza się krawędzi  
 $(u, u)$



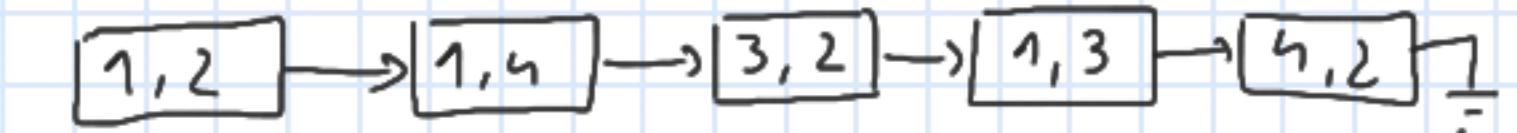
Z kązdy krawędzią / wierzchołkiem  
można zaznaczyć dodatkowe informacje

Graf nieskierowany - krawędzie są zbiorem  
dwuelementowym

negsto realizowane jako grafy skierowane,  
gdzie krawędzie są "w obie strony"

## Reprezentacje grafów

Lista / tablica krawędzi

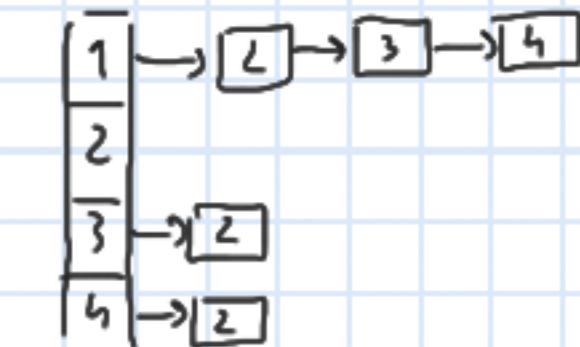


Reprezentacja macierzowa

	1	2	3	4
1	-	1	1	1
2	0	-	0	0
3	0	1	-	0
4	0	1	0	-

ten fragment  
wygenerowany dla  
grafów nieskierowanych

Reprezentacja przez listy sąsiedztwa



# Algorytm BFS (breadth-first search)

Peszulwanie w szerz

```
def BFS( G, s)
```

```
#  $G = (V, E)$ ,  $s \in V$ 
```

```
Q = Queue()
```

```
for  $v \in V$ :  $v.visited = False$ 
```

```
s.d = 0
```

```
s.visited = True
```

```
s.parent = None
```

```
Q.put(s)
```

```
while not Q.empty():
```

```
    u = Q.get()
```

```
    for v - sąsiedzi u:
```

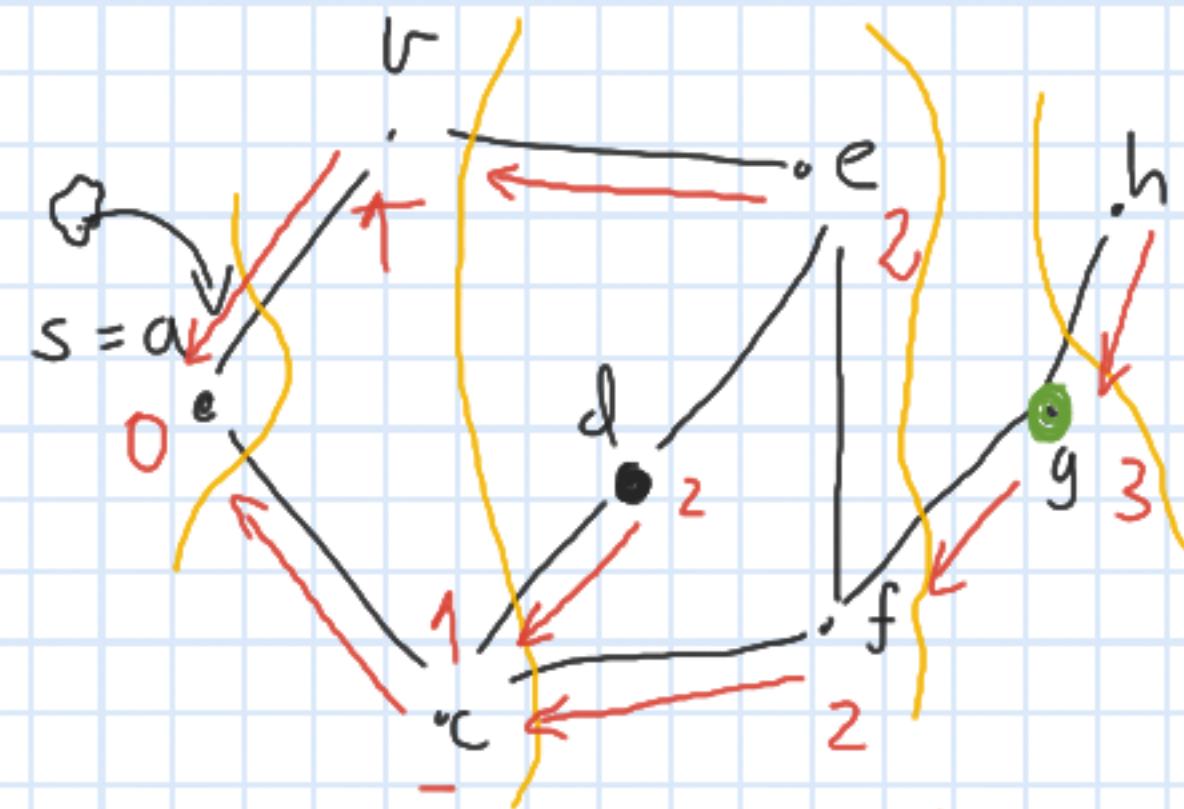
```
        if not v.visited:
```

```
            v.visited = True
```

```
v.d = u.d + 1
```

```
v.parent = u
```

```
Q.put(v)
```



Złożoność

rep. maturalna

$O(V^2)$

rep. listowa

$O(V + E)$

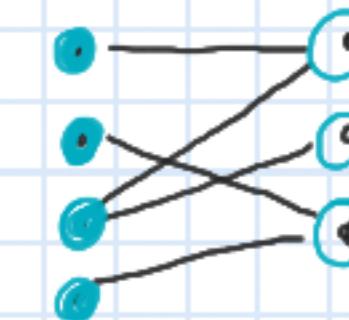
Zastosowania

- najkrótsze ścieżki (w grafie bez wag)

- spójność grafu

- wykrywanie cykli

- divedzimusi



# Algorytm DFS (depth-first search)

Pniewzkuwanie w gętce

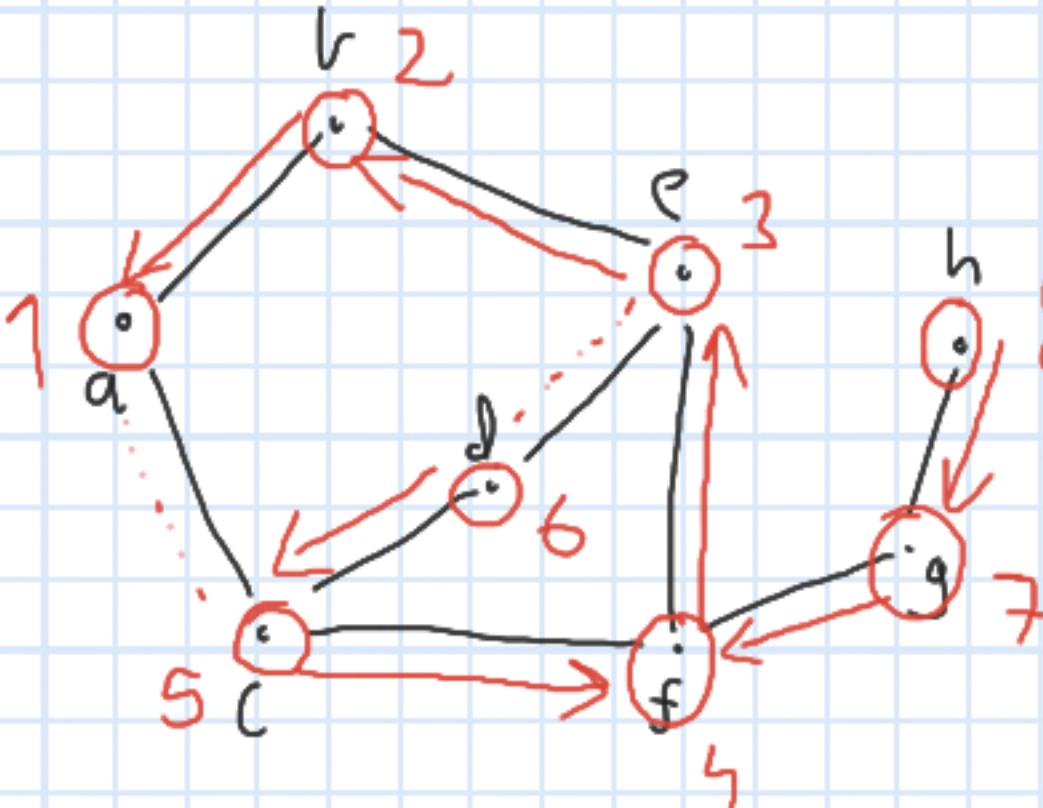
```
def DFS( G ):
    # G = (V, E)
    for v ∈ V:
        v.visited = False
        v.parent = None
    time = 0
```

```
    for u ∈ V:
        if not u.visited:
            DFSVisit(G, u)
```

## Złożoność

nep. mieniona  $O(V^2)$

nep. listowa  $O(V+E)$



def DFSVisit(G, u):

nonlocal time

time += 1

u.visited = True ← *u zostało zastąpione / u was overwritten*

for v - sąsiad u:

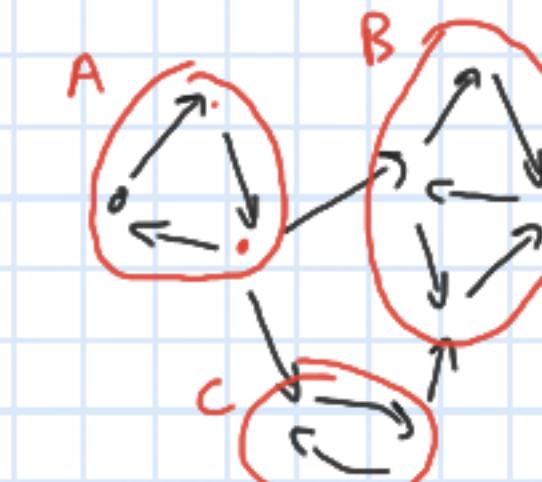
if not v.visited:

v.parent = u

DFSVisit(G, v)

time += 1

← *v zostało ponownie zastąpione / v was overwritten again*



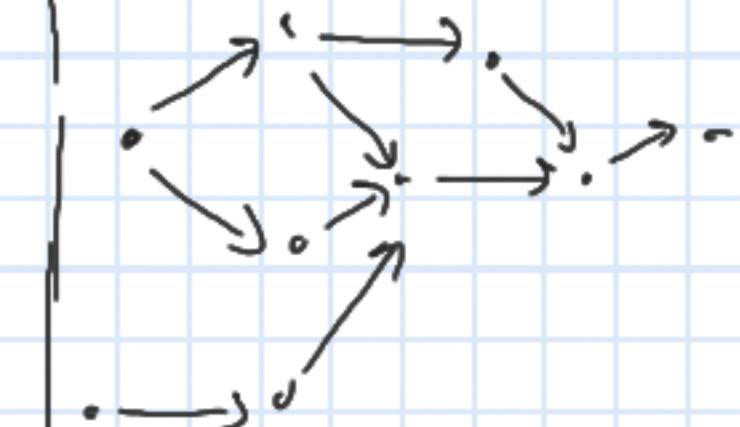
## Zastosowania

- spójność

- dwudzielność

- wykrywanie cykli

- sortowanie topologiczne



- cykl Eulera

- silnie spójne

skladowe

- mosty / płat. cuty kolumni



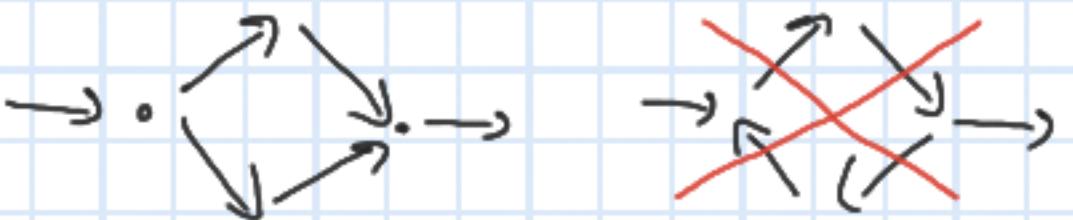
## ASD - Uglystad 10

Kolokrium - mugi złożoności

- złożoność uzupełnia + 2.5 pkt | + 1 pkt
- złożoność akceptowalna + 1.5 pkt | + 3 pkt

## Sortowane topologiczne

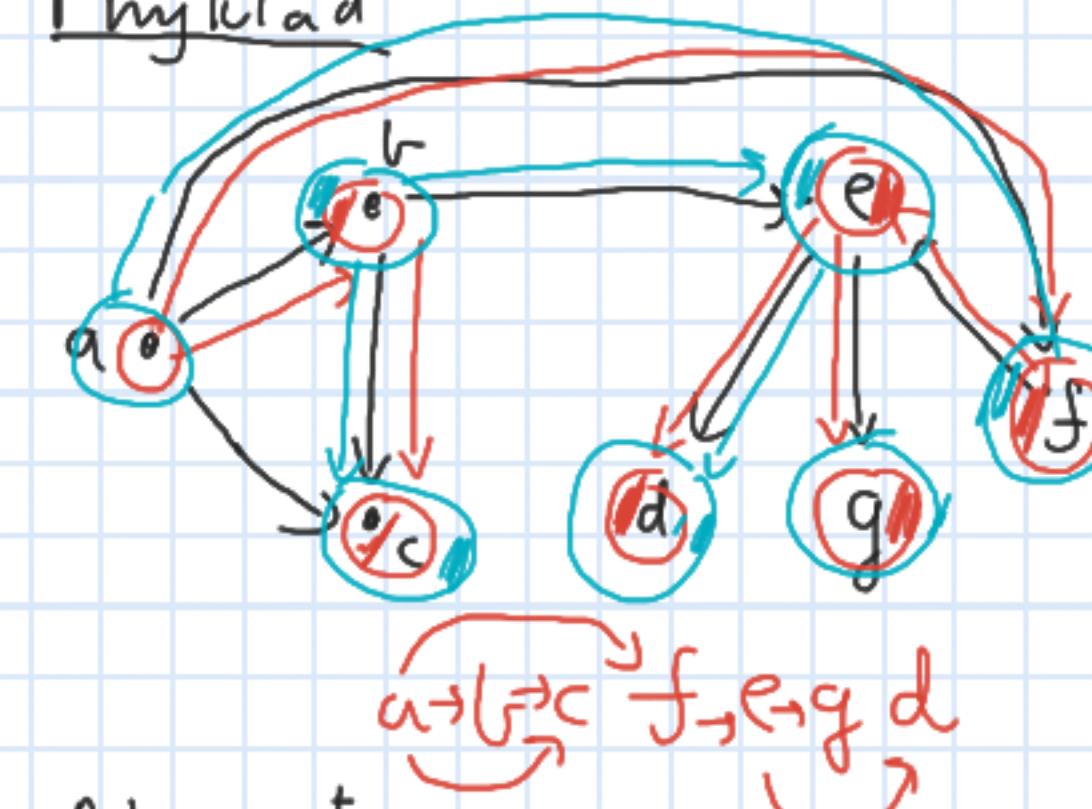
dag - directed acyclic graph



Sortowanie topologiczne dag'u - utworzenie

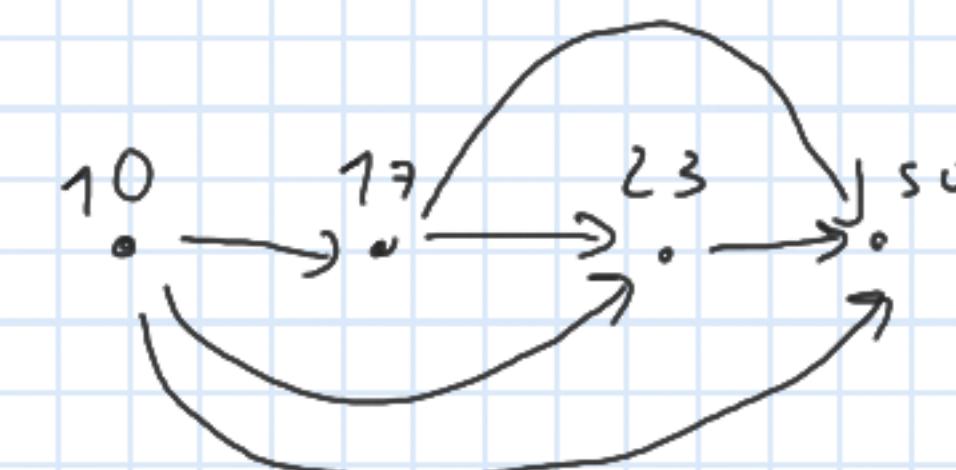
ciendotek tak, aby wszystkie krawędzie  
uskażrły "z lewa na prawo"

## Pmykład



## Algorytm

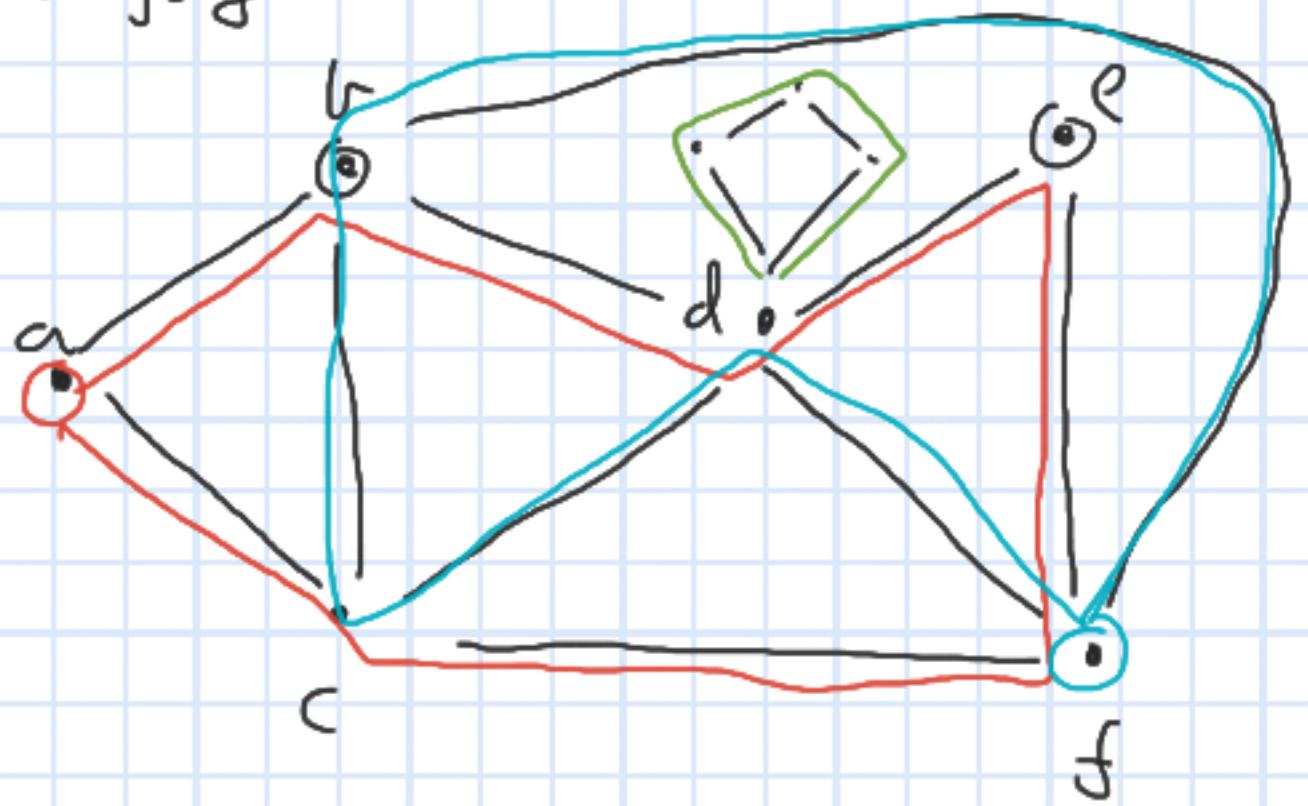
- wchodzimy DFS
- po zwroceniu się do końca dopyśaj go na początek listy



## Cykl Eulera

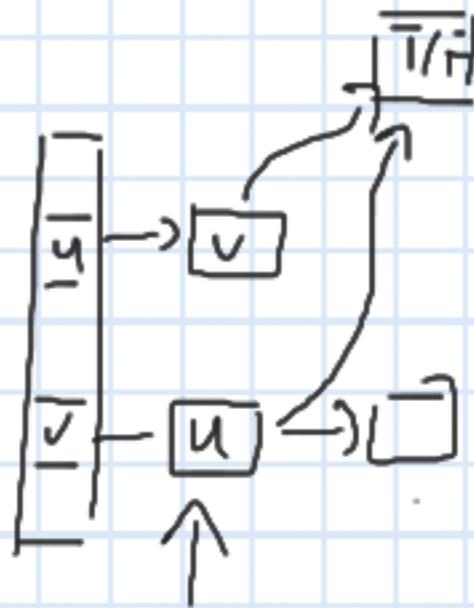
def Cykl Eulera to cykl, który przechodzi przez wszystkie krawędzie, odwiedzając każdą dokładnie raz

tu Sprawny graf nieskierowany posiada cykl Eulera wtw. gdy każdy jego wierzchołek ma stopień parzysty

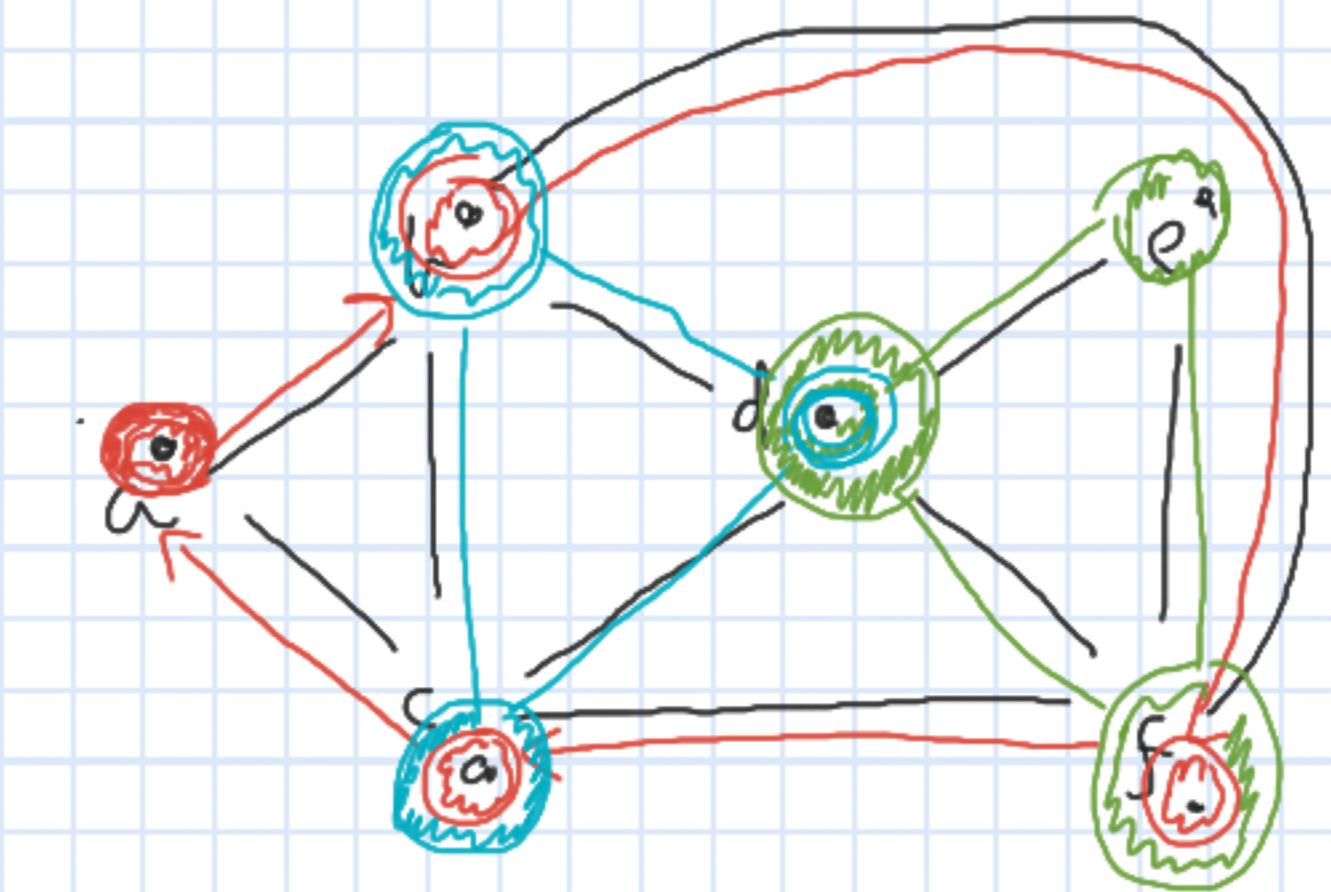


## Algorytm

- wykonujemy DFS, nie stosując pola visited, ale usuwając krawędzie po których przeszliśmy
- po skończeniu wierzchołka dopisujemy go na koniec trawnego cyklu



$O(V+E)$



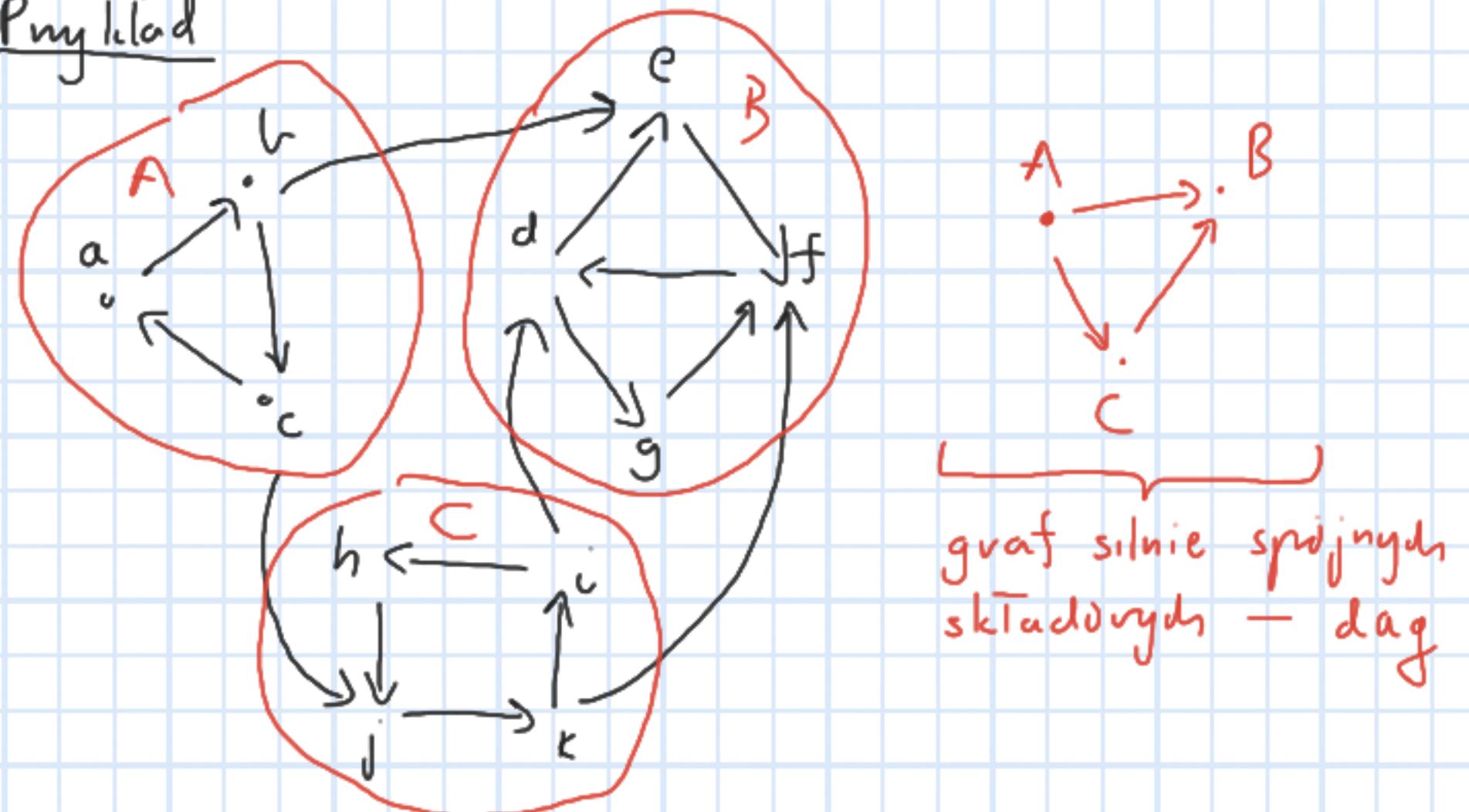
a b f c b d e f d c a

## Silnie spójne składowe

def Niech  $G = (V, E)$  będzie grafem skierowanym. Mówimy, że  $u, v \in V$  należą do tej samej silnie spójnej składowej jeśli istnieją succiki skierowane z

$u$  do  $v$  oraz z  $v$  do  $u$

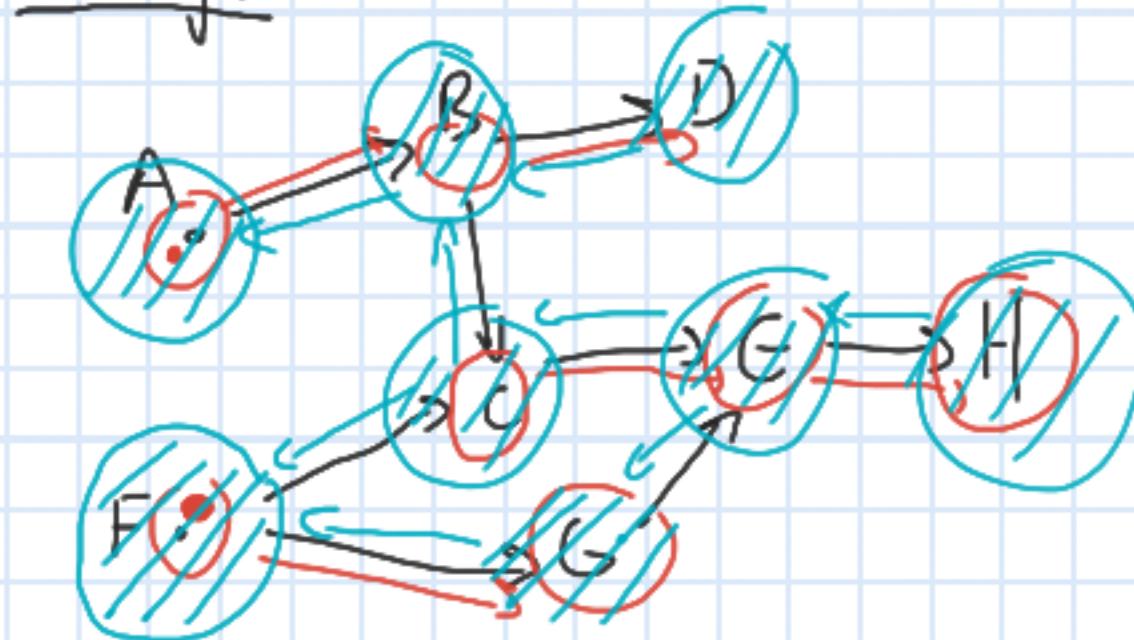
## Ponkład

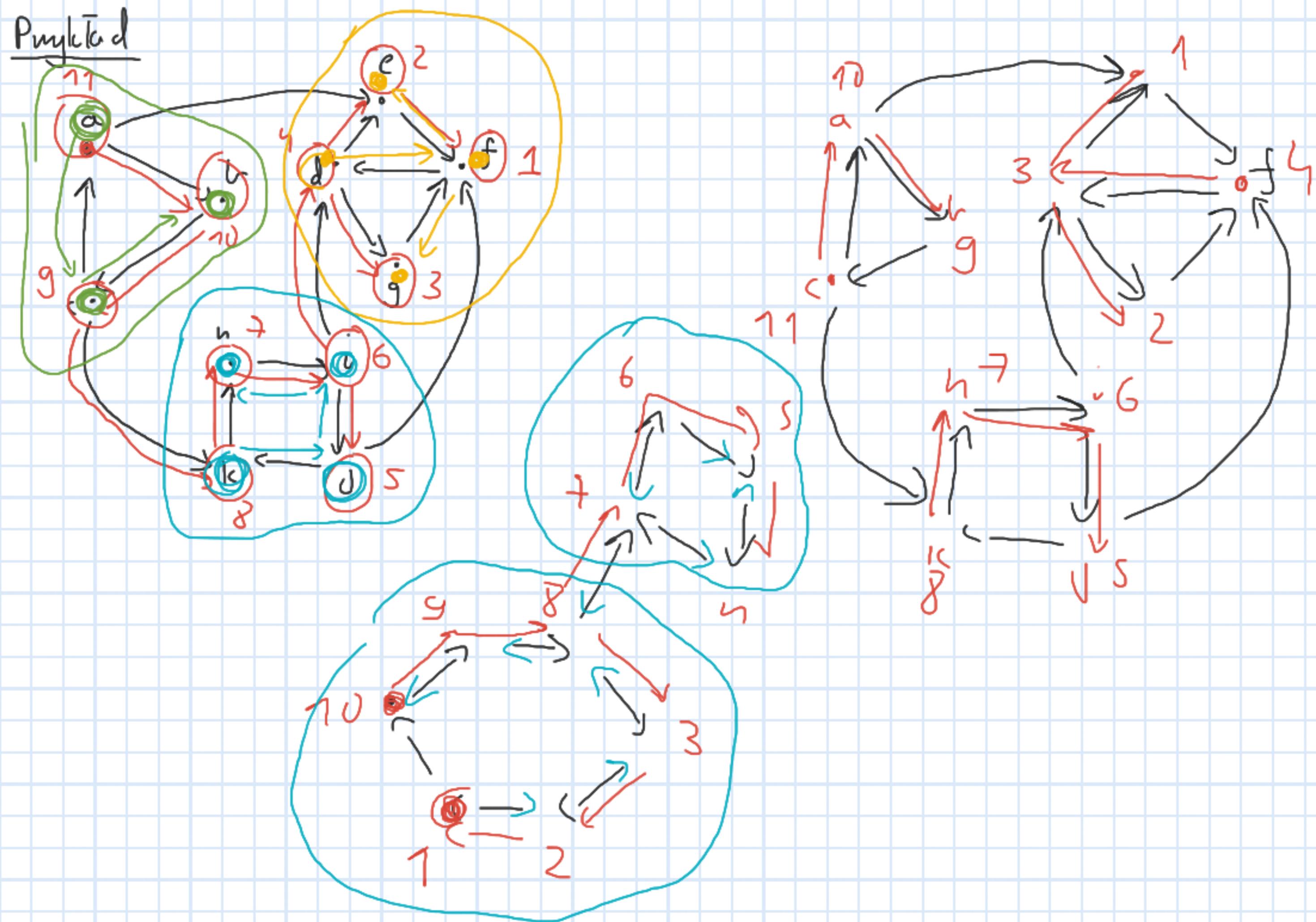


## Algorytm

1. Wykonaj DFS na grafie  $G$ , zapamiętując wizyty przedtem.
2. Odwrócić kierunek wszystkich krawędzi.
3. Wykonaj kolejny DFS, w kolejności malejących (zapamiętanych) wizyt przedtem.

## Intuicja





## Mosty w grafach nieskierowanych

def Krawędź  $e$  w spójnym grafie nieskierowanym jest mostem jeśli jej usunięcie rozspojniła graf



tw Krawędź  $e$  jest mostem w t.j. gdy nie leży na żadnym cyklu prostym w grafie

## Dowód

$e$  jest mostem  $\Rightarrow e$  nie leży na cyklu  
(gdyby leżała to usunięcie nie rozspojniłoby grafu)

$e$  nie leży na  $\Rightarrow e \in \{u, v\}$  jest mostem, bo usunięcie żadnym cyklu  
a pokazuje, że nie ma ścieżki z  $u$  do  $v$

## Algorytm

① wykonaj DFS, dla każdego wierzchołka  $v$  zapamiętuj jego czas odwiedzenia  $d(v)$

② dla każdego wierzchołka oblicz:

$$\text{low}(v) = \min \left( d(v), \min_{u - \text{mamy kraw} \ddot{\text{e}} \text{ usterkę z } v \text{ do } u} d(u) \right)$$

$$\min_{w - \text{drzeka } v \text{ w drzewie DFS}} \text{low}(w)$$

③ Mosty to krawędzie

$\{v, p(v)\}$  gdzie  $d(v) = \text{low}(v)$   
 $v$  rodzic  $v$  w DFSie

$d(v)$

|uu(v)

