

## ASD - Wykład 2

### Literatura

T. Cormen, C. Leiserson, R. Rivest, C. Stein,  
Wprowadzenie do algorytmów, PWN 2012

S. Dasgupta, C. Papadimitriou, U. Vazirani,  
Algorytmy, PWN 2010

S. Skiena, The Algorithm Design Manual,  
Springer 2008

Czym się zajmujemy / co nas interesuje?

Efektywne mechanizmy procedury rozwiązywania  
dolne zdefiniowane problemy obliczeniowe

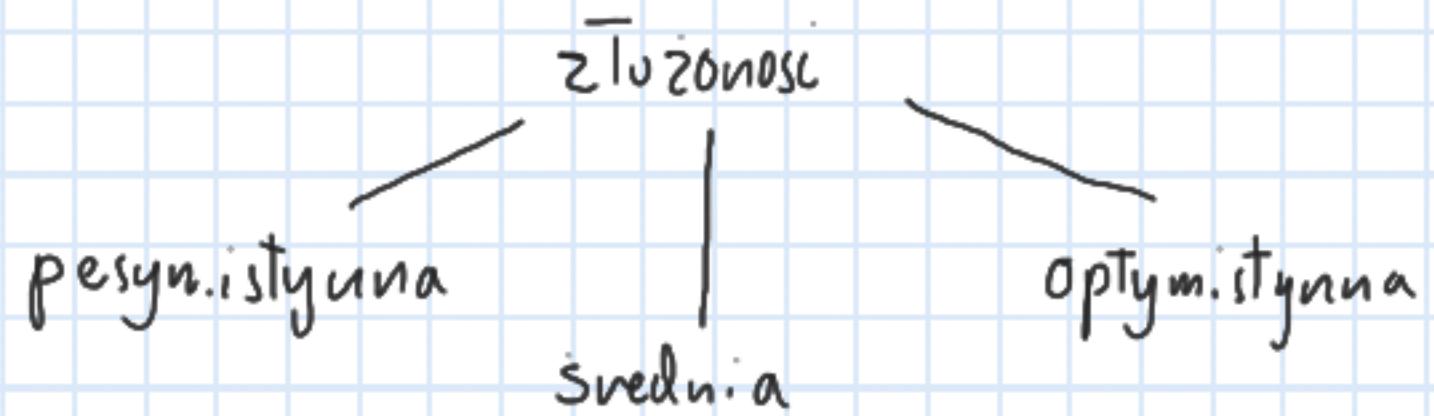
Co nas nie interesuje?

Szczegółowy techniczny

### Złożoność obliczeniowa

Złożoność czasowa algorytmu to funkcja, która mówi  
ile elementarnych operacji algorytm wykonując na danych  
określonego rozmiaru

Złożoność pamięciowa — j.w., ale mamy linki użytych komórek



## Notaya asymptotynna

$f, g$  - funkcje

$$f : \mathbb{N} \rightarrow \mathbb{N}$$

$$g : \mathbb{N} \rightarrow \mathbb{N}$$

def Mówimy, że  $f$  jest  $O(g(n))$  jeśli:

$$(\exists c > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) [ f(n) \leq c g(n) ]$$

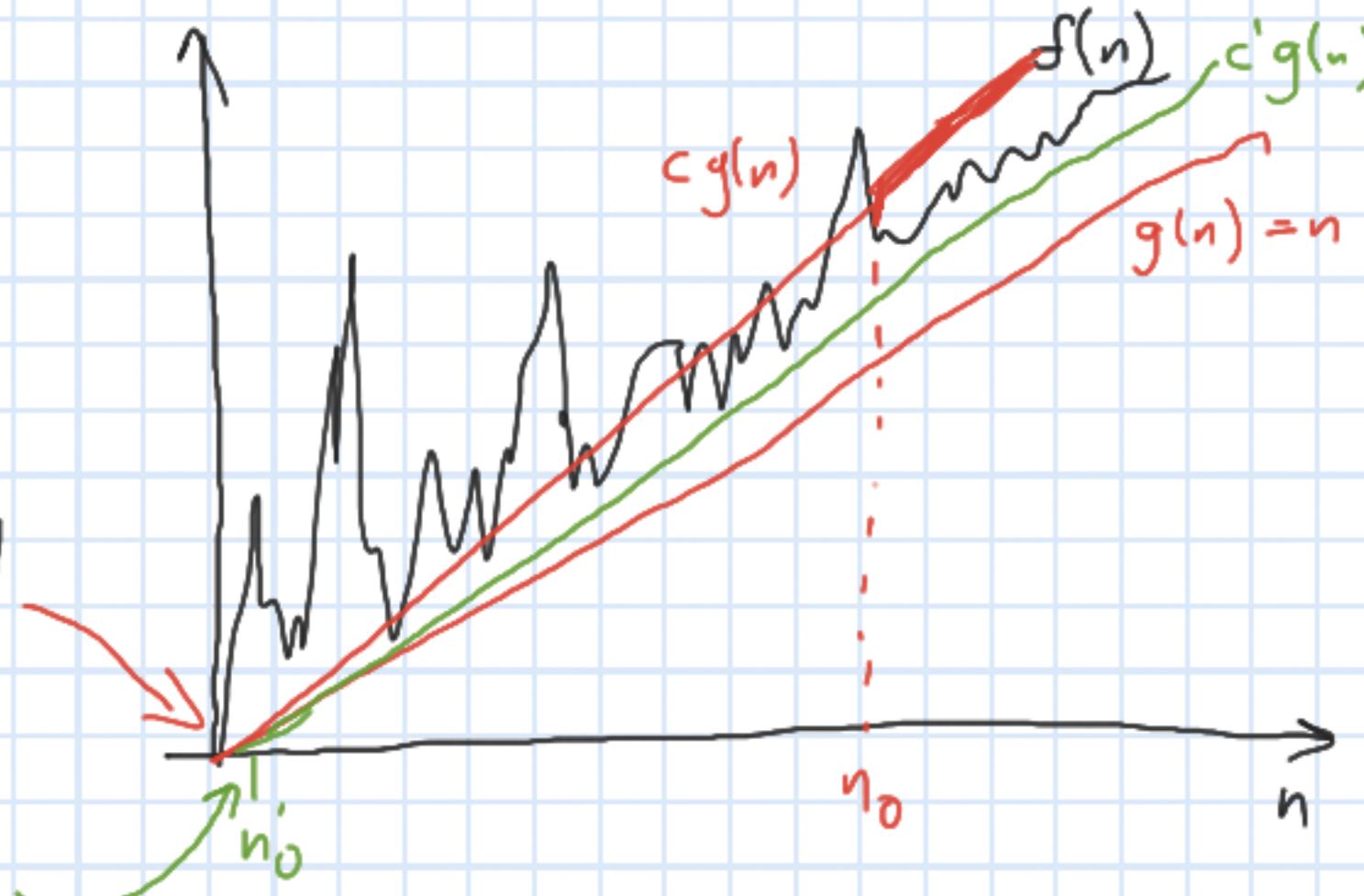
Mówimy, że  $f$  jest  $\Omega(g(n))$  jeśli:

$$(\exists c > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) [ c g(n) \leq f(n) ]$$

Mówimy, że  $f$  jest  $\Theta(g(n))$  jeśli

- jest  $O(g(n))$  i  $\Omega(g(n))$

$$(\exists c_1, c_2 > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) [ c_1 g(n) \leq f(n) \leq c_2 g(n) ]$$



## Problem srotowania

Dane: ciąg  $a_1, \dots, a_n$  danych z operatorem  $\leq$

Wynik: permutacja  $a'_1, \dots, a'_n$  ciągu

ujawnionego, takie że:

$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

## Uwagi

Reprezentacja danych  $\rightarrow$  tablica

$\hookrightarrow$  lista 1/2 -kierunkowa

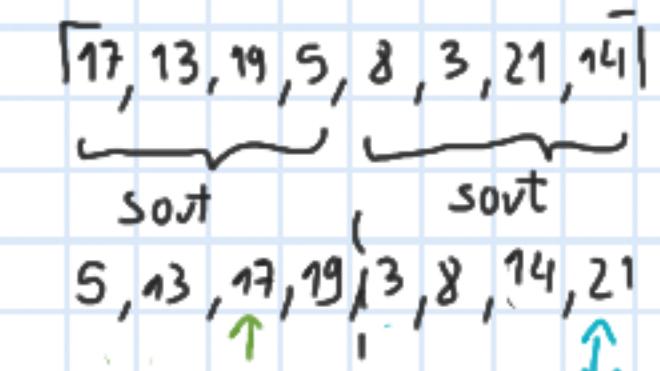
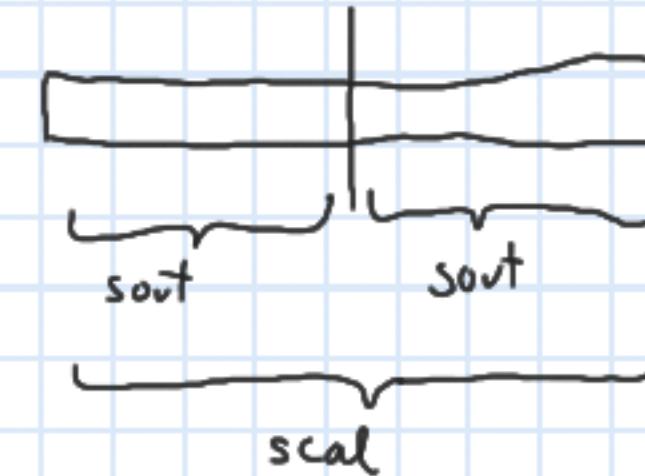
$\hookrightarrow$  plik

algorytm srotowania  $\rightarrow$  prosty  $\Theta(n^2)$

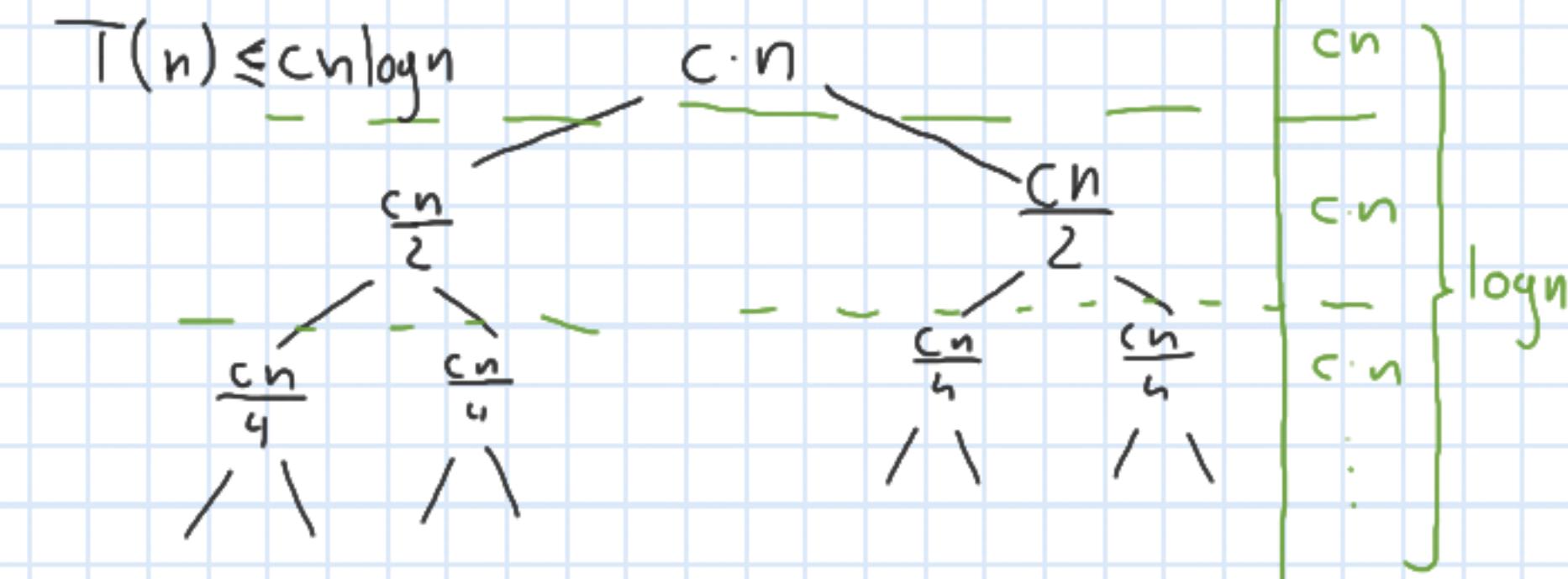
$\hookrightarrow$  szybkie  $\Theta(n \log n)$

## ① Srotowanie przez scalanie / merge sort

A:



$$T(n) = \begin{cases} c, & n=1 \\ 2 \cdot T\left(\frac{n}{2}\right) + cn, & n > 1 \end{cases} \Rightarrow T(n) = O(n \log n)$$



$$\frac{n}{2^h}, h = \log n \Rightarrow \frac{n}{2^{\log n}} = \frac{n}{n} = 1$$

## ② Soutvaranie kopcov / heap sort

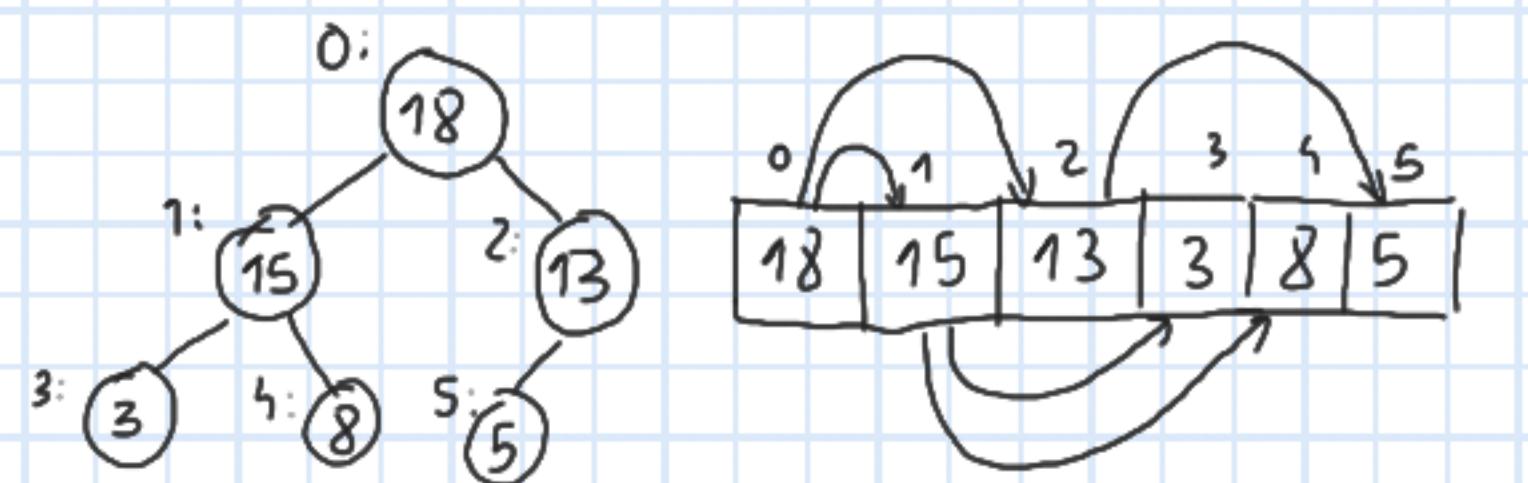
kopice - dnevo binarne, u ktorym v každym

uzile využitnym jej prechovyvanu

vlastoci vyska latr vyska niz u jeho

dzieciach

### Prykaz

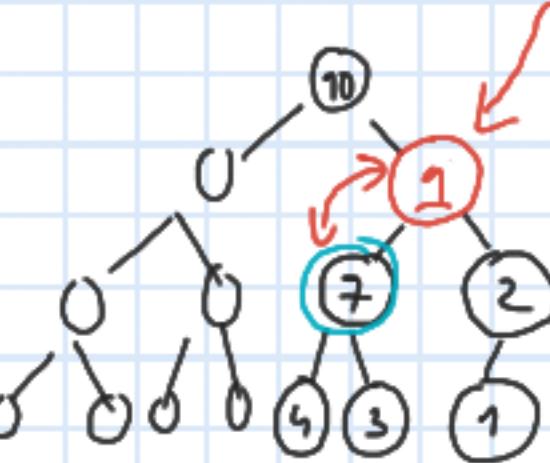


def left(i): return  $2i + 1$

def right(i): return  $2i + 2$

def parent(i): return  $(i - 1) // 2$

### Pnyrovanie účasníku kopca



def heapify(A, n, i)

l = left(i)

r = right(i)

max-ind = i

if l < n and A[l] > A[max-ind]:

max-ind = l

if r < n and A[r] > A[max-ind]:

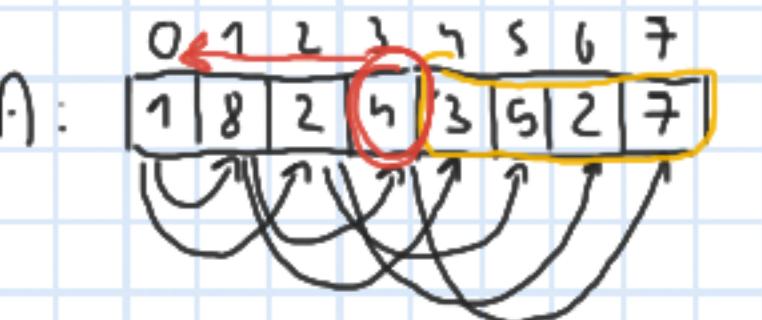
max-ind = r

if max-ind ≠ i:

A[i], A[max-ind] =  $\longleftrightarrow$  swap(A[i], A[max-ind])  
A[max-ind], A[i] =  $\longleftrightarrow$  heapify(A, n, max-ind)

$\Theta(\log n)$

Jak zbudować kopiec?



```
def build_heap( A ) :
```

n = len(A)

```
for i in range( parent(n-1), -1, -1 )
```

```
    heapify( A, n, i )
```

$O(n \log n)$

$\Theta(n)$

Soutwarzanie

```
def heap-sout( A ) :
```

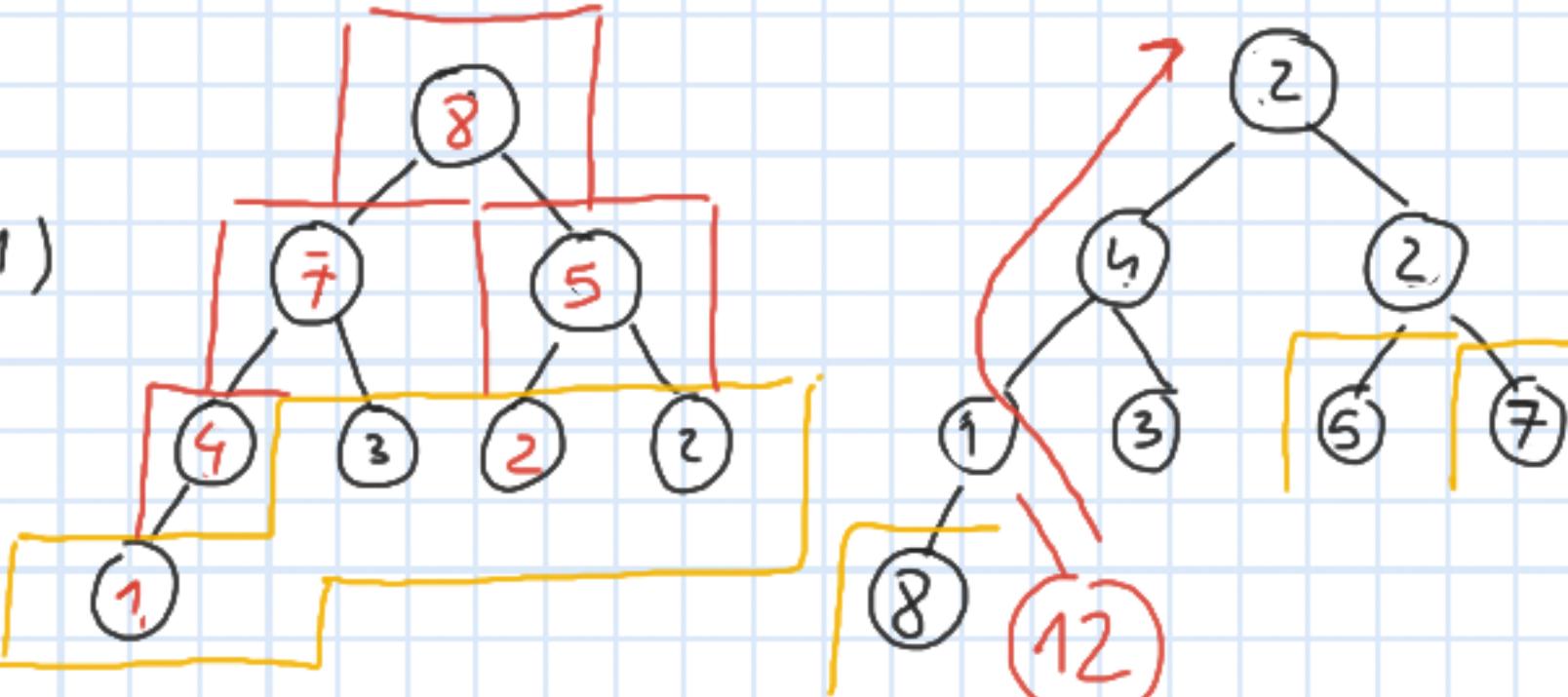
n = len(A)

```
build_heap(A)
```

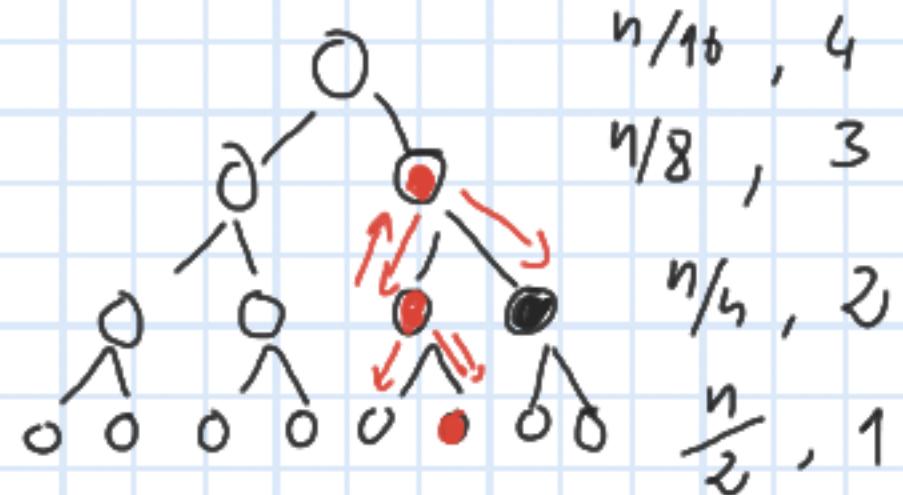
```
for i in range( n-1, 0, -1 ) :
```

```
    swap( A[0], A[i] )
```

```
    heapify( A, i, 0 )
```



# ASD - Wykład 3



$$\frac{n}{2} + \frac{n}{4} \cdot 2 + \frac{n}{8} \cdot 3 + \dots = \sum_{i=1}^{\lfloor \log n \rfloor} \left( \frac{n}{2^i} \cdot i \right)$$

$$= n \sum_{i=1}^{\lfloor \log n \rfloor} \frac{i}{2^i}$$

dla  $x = \frac{1}{2}$

$$\leq n \sum_{i=0}^{\infty} \frac{i}{2^i} \leq 2n$$

$$f(x) = 1 + x + x^2 + x^3 + \dots = \frac{1}{1-x}$$

$$f'(x) = 1 + 2x + 3x^2 + 4x^3 + \dots = \frac{1}{(1-x)^2}$$

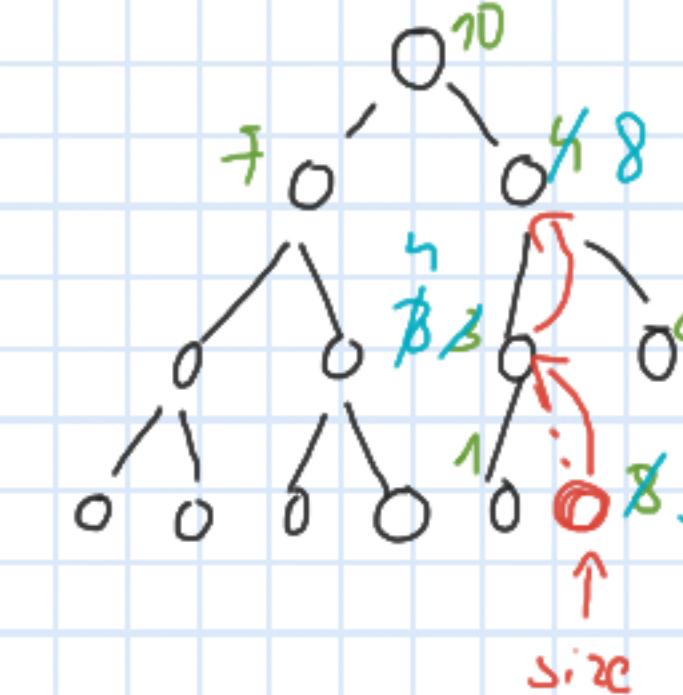
$$x f'(x) = x + 2x^2 + 3x^3 + 4x^4 + \dots = \frac{x}{(1-x)^2}$$

## Kolejka priorytetowa

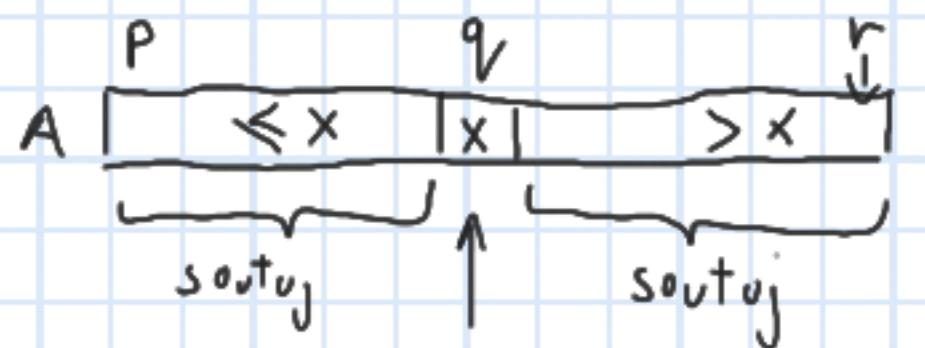
"Cos, do tego można wstawiać elementy w dowolnej kolejności i usuwać w kolejności zgodnej z priorytetem"

class PQ:

```
def __init__(self, n):
    self.T = [None] * n
    self.size = 0
```



## Quick Sort



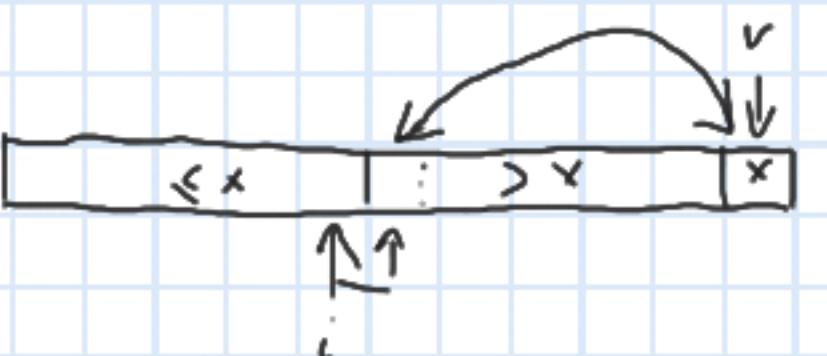
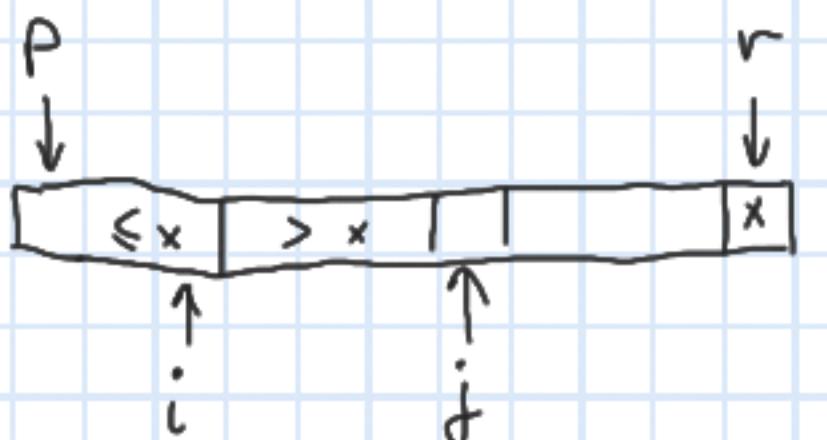
def quick-sort(A, p, r):

if  $p < r$ :

$q = \text{partition}(A, p, r)$

quick-sort(A, p, q - 1)

quick-sort(A, q + 1, r)



def partition(A, p, r):

$x = A[r]$  // zamiennic  $A[r]$

z losowym elementem

for  $j$  in range( $p, r$ ):  $\leq A[p], \dots, A[r]$

if  $A[j] \leq x$ :

$i += 1$

swap( $A[i], A[j]$ )

swap( $A[i+1], A[r]$ )

return  $i + 1$

Złożoność czasowa algorytmu

## Quick Sort

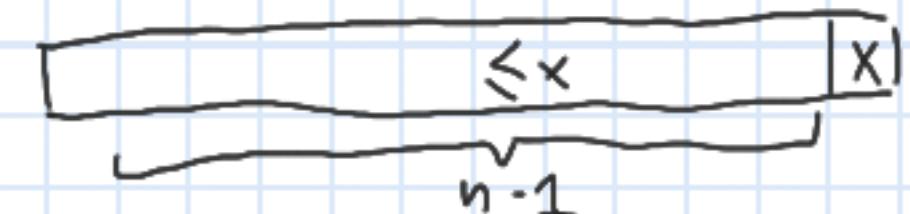
Idealne podzielenie

$$T(n) = \begin{cases} c, & n \leq 1 \\ 2T\left(\frac{n}{2}\right) + cn, & n > 1 \end{cases}$$

$$\bar{T}(n) = \Theta(n \log n)$$

Podejście podzielenia

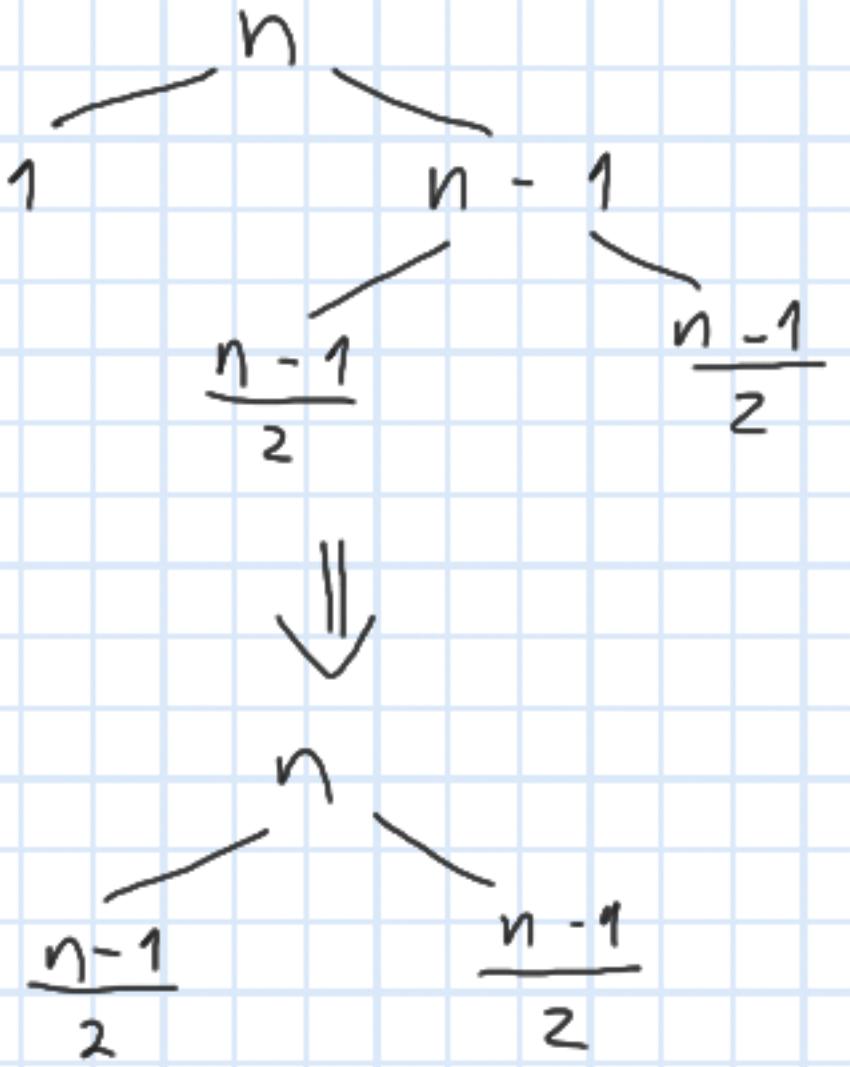
$$T(n) = \begin{cases} c, & n \leq 1 \\ T(n-1) + cn, & n > 1 \end{cases}$$



$$T(n) = \bar{T}(n-1) + cn = \bar{T}(n-2) + c(n-1) + cn$$

$$= c(1 + 2 + 3 + \dots + n) = \Theta(n^2)$$

Mieszanka podziałów - "co drugi peresorty, pozostałe idealne"



### Staty stylu peresortyjnego

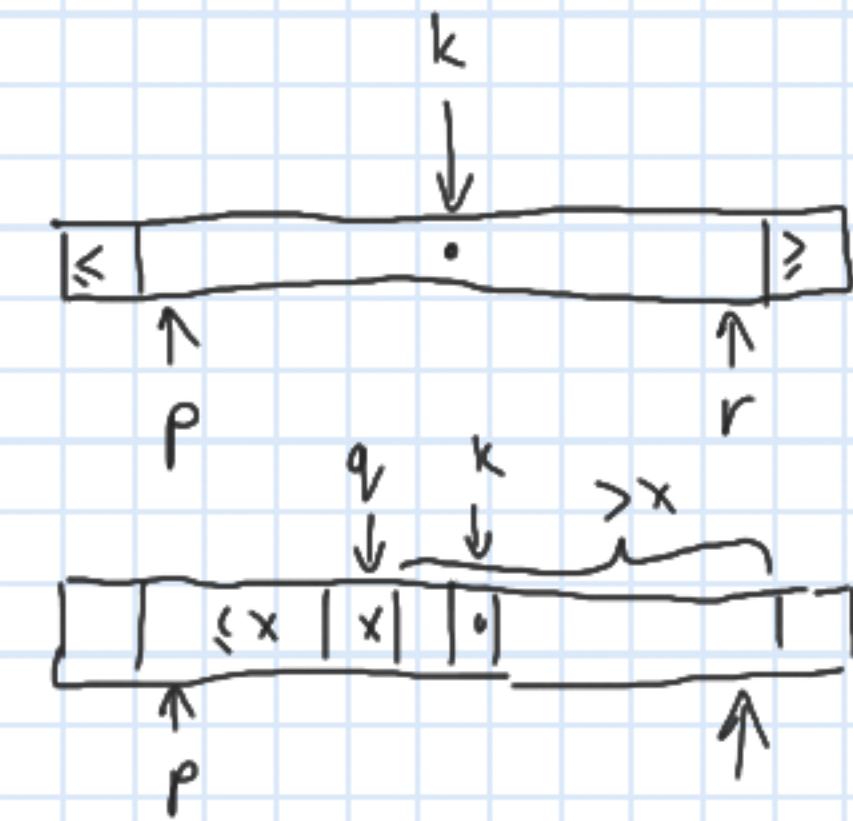
min / max - omywisty algorytm  $\Theta(n)$

$$T(n) = \begin{cases} c & , n \leq 1 \\ T\left(\frac{n}{2}\right) + cn, & n > 1 \end{cases}$$

U ogólnosu chciemy obliczyć element, który po posortowaniu byłby pod indeksem  $k$

$$T(n) = cn + \frac{cn}{2} + \frac{cn}{4} + \dots = \Theta(n)$$

```
def select(A, p, k, r)
    if p == r: return A[p]
    if p < r:
        q = partition(A, p, r)
        if q == k: return A[q]
        elif q < k: return select(A, q+1, k, r)
        else: return select(A, p, k, q-1)
```



Statystyki pozycyjne w persymetryjnym  
uzasadnie liniowym — Magazne Piątki

### Algorytm

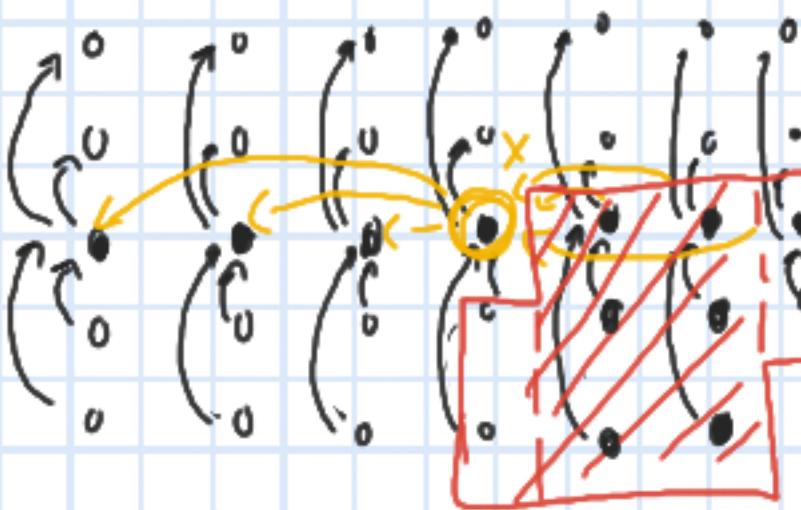
- ① Podziel ocenioną tablicę na  $\lceil \frac{n}{5} \rceil$  grup po 5 elementów

U każdej grupie uznajemy medianę

- ② Rekurencyjnie uznajemy  $x$  jako medianę median

- ③ Kontynuujemy tak jak w select, traktując  $x$  jak pivot w partition

możemy wybrać tak duże  $c$ ,  
że ta wartość jest zawsze  
ujemna



ile jest elementów większe od  $x$ ?

$$3 \left( \lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2 \right) \geq \frac{3n}{10} - 6$$

### Złożoność czasowa

$$T(n) = \begin{cases} \Theta(1), & n \leq \text{stała} \\ T\left(\lceil \frac{n}{5} \rceil\right) + T\left(\frac{3n}{10} + 6\right) + \Theta(n) \end{cases}$$

Twierdzimy, że  $T(n) \leq cn$ , dla pewnego  $c$

Dowód indukcyjny.

$$\begin{aligned} T(n) &\leq c \lceil \frac{n}{5} \rceil + \frac{7nc}{10} + 6c + an \\ &\leq \frac{2cn}{10} + \frac{7cn}{10} + 7c + an \\ &= cn + \left( -\frac{1}{10}cn + 7c + an \right) \end{aligned}$$

# ASD - Wykład 5

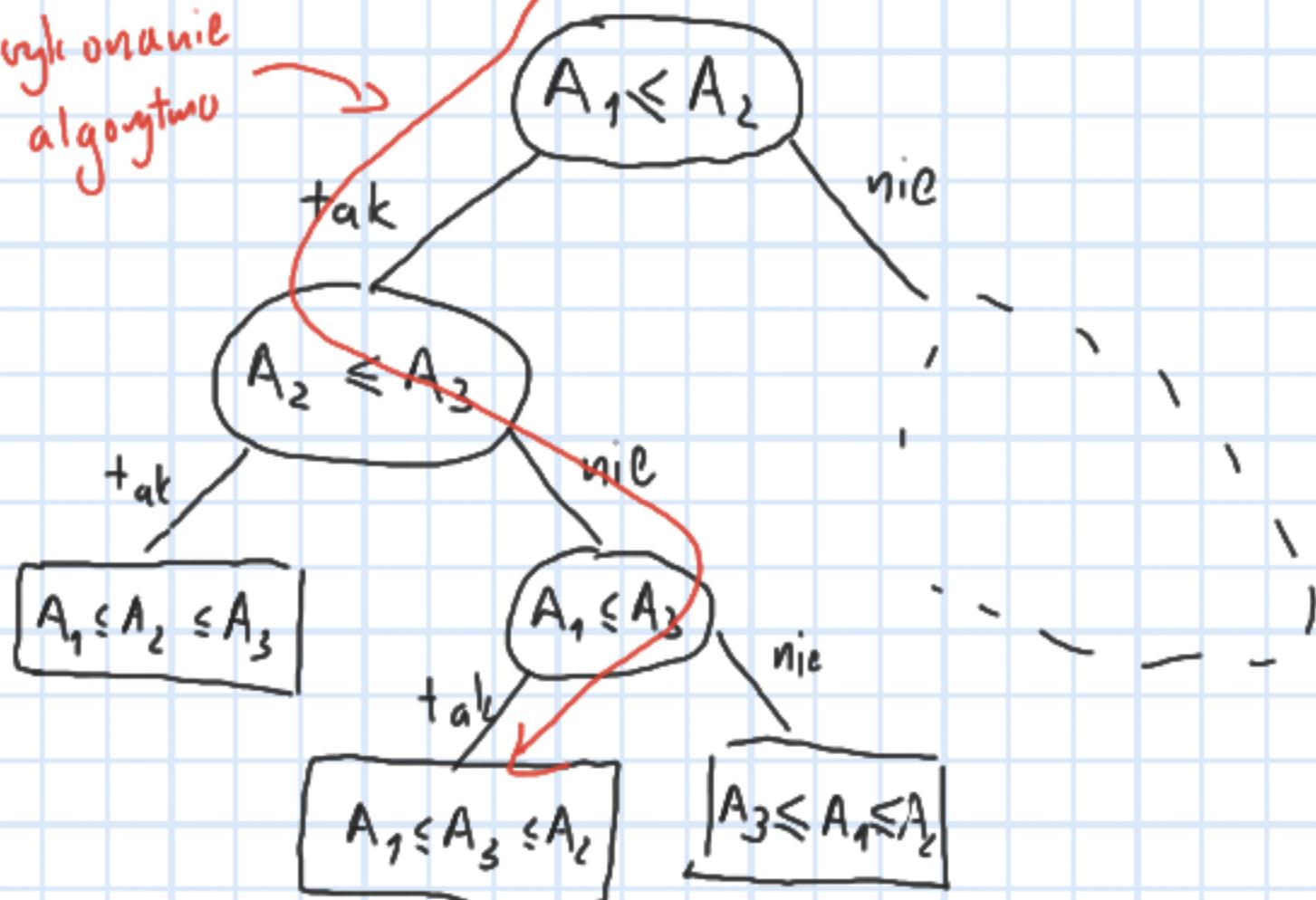
Dolne ograniczenie na złożoność czasową  
sortowania

A: 

$A_1$	$A_2$	$A_3$
-------	-------	-------

 ← tabela n elementowa

wykonanie  
algorytmu



wysokość drzewa

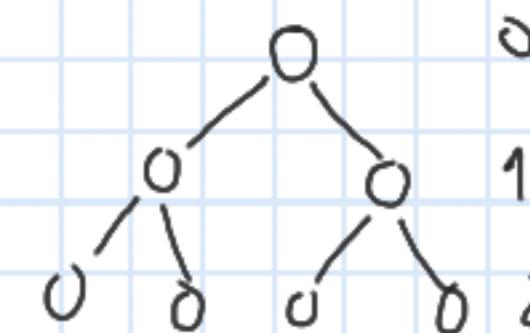
$$h \geq \log(n!)$$

$$\frac{n}{2}(\log n - 1) = \log\left(\frac{n}{2}\right)^{\frac{n}{2}} \leq \log n! \leq \log n^n = n \log n$$

$\Theta(n \log n)$

Mamy drzewa binarne, które ma co najmniej  $n!$  liści

Drzewo binarne o wysokości  $h$  ma  $\leq 2^h$



## Soutovanie u časie liniuym

Counting Sout - soutovanie prez zlizanie

- sortujemy n elementov tablice z

klucami, ktore sú luktami naturalnymi

miedzy 0 a k - 1

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

C	1   1   1   1   1 0 1 2 3 4	$\rightarrow$	2   2   1   2   2 0 1 2 3 4
---	--------------------------------	---------------	--------------------------------

vykonalanie ( $x$ )

B	0   0   1   1   2   3   3   4   4 0 1 2 3 4 5 6 7 8	$\rightarrow$	0   2   4   5   7   9 0 1 2 3 4
---	--	---------------	------------------------------------

def counting\_sout(A, k):

n = len(A)

C = [0] \* k

B = [0] \* n

for x in A: C[x] += 1

(\*) for i in range(1, k): C[i] = C[i] + C[i-1]

(\*\*) for i in range(n-1, -1, -1):

B[C[A[i]]-1] = A[i]

C[A[-]] -= 1

for i in range(n):

A[i] = B[i]

$\Theta(n + k)$

## Radix Sort - sortowanie pozycyjne

k	v	a
a	r	t
k	o	t
k	i	t
a	t	i
k	i	l

$\Rightarrow$

k	v	a
a	t	i
k	i	l
a	v	t
k	o	t
k	i	t

$\Rightarrow$

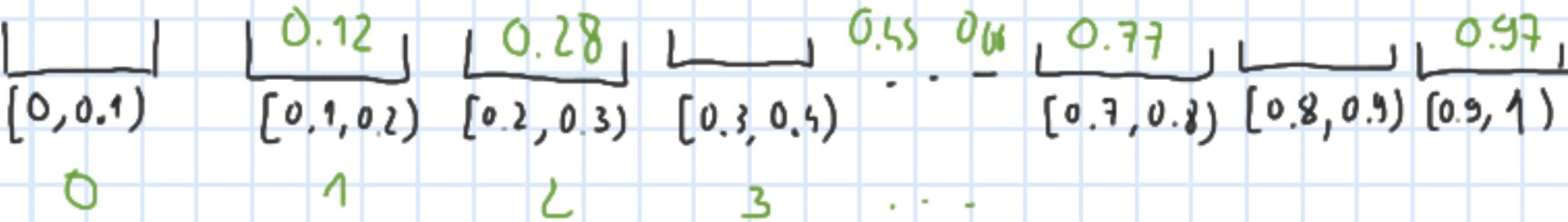
k	i	l
k	i	t
k	o	t
a	v	t
a	t	i
k	v	a

Sortowanie kubetkowe - algorytm stosowany wtedy, gdy wiemy, że dane pochodzą z rozkładu jednastajnego na pewnym przedziale

Rozważmy  $n$  elementową tablicę A, które klone pochodzą z rozkładu jednolistajnego na  $[0,1)$

$$n = 10$$

0.12, 0.77, 0.45, 0.66, 0.28, 0.97, ...



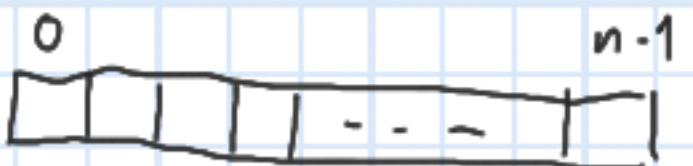
Tużymy n kubetków

## Abstrakcyjne struktury danych

• coś co oferuje pewien "kontrakt" opisujący  
zbiór operacji w jaki sposób  
wykonie działać

- fizyczna realizacja

## Tablica



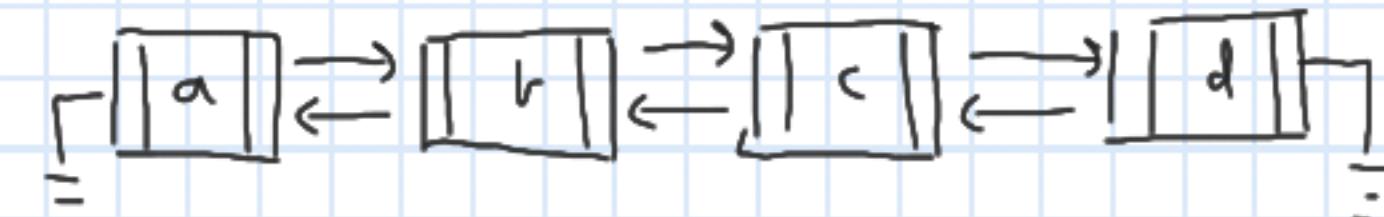
"Coś, do tego można się odwoływać  
po numerach komórek"

## Lista jedno/dwukierunkowa



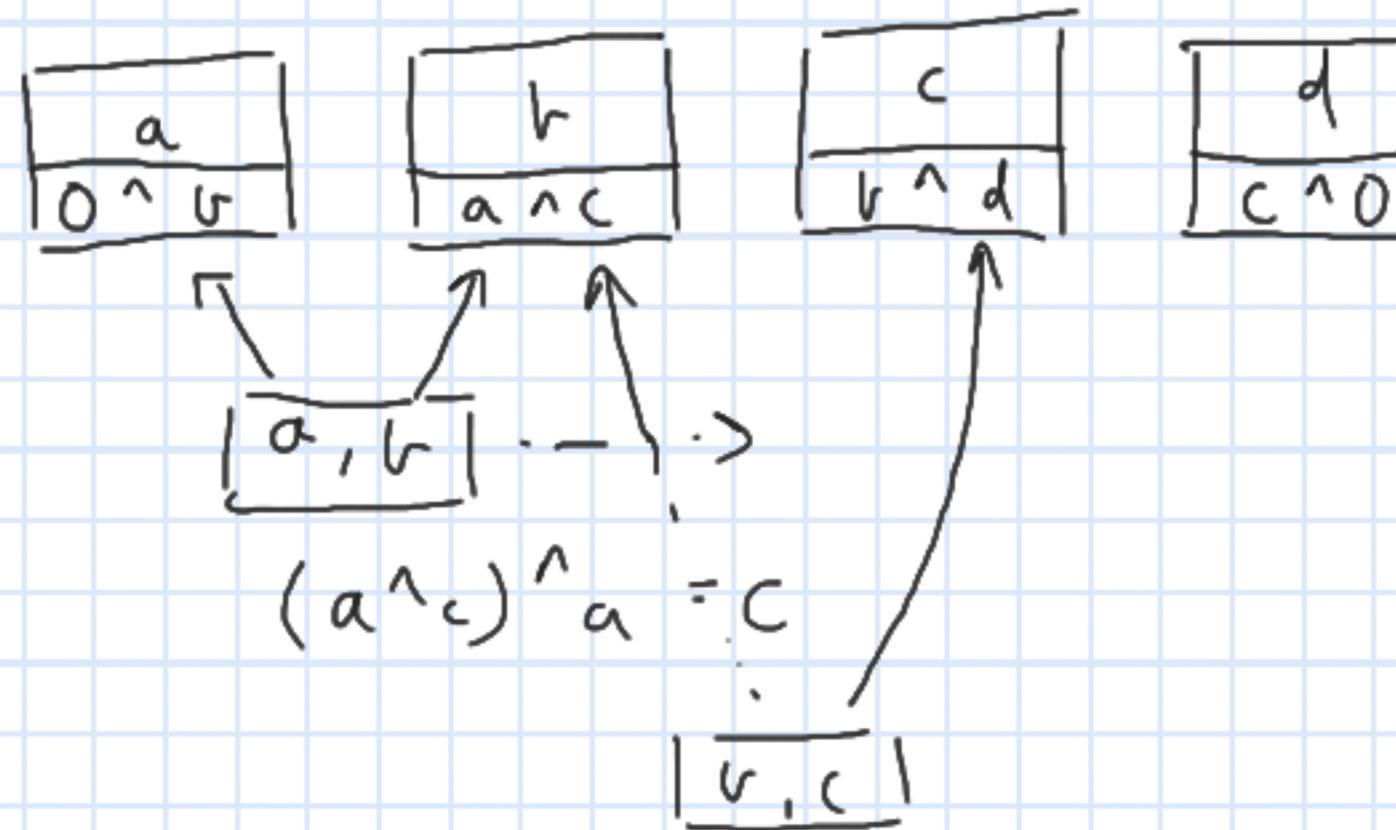
"Coś, co pozwala przerwać się od punktu do  
konca i wpinać/usuwać elementy"

## Lista dwukierunkowa z jednym uskażenikiem



$$\begin{array}{r} \begin{array}{cccccc} 1 & 0 & 1 & 1 & 0 & A \\ \times OR & 0 & 1 & 1 & 0 & 0 \\ \hline 1 & 1 & 0 & 1 & 0 & \end{array} \\ B : \begin{array}{cccccc} 1 & 1 & 0 & 1 & 0 \\ \times OR & 0 & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 1 & 0 & \end{array} \\ \Rightarrow A \oplus B : 1 & 0 & 1 & 1 & 0 \end{array} \quad (A \oplus B)$$

XOR = ^



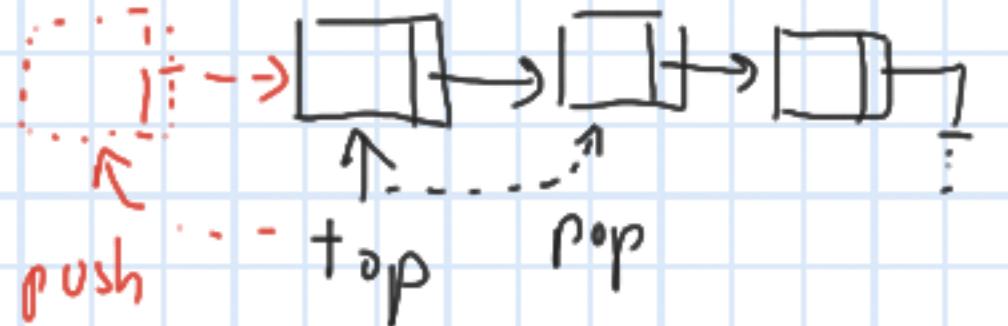
## Stos

"Cos, co powraca ostatni element  
na szczyt i z niego zdejmowac"

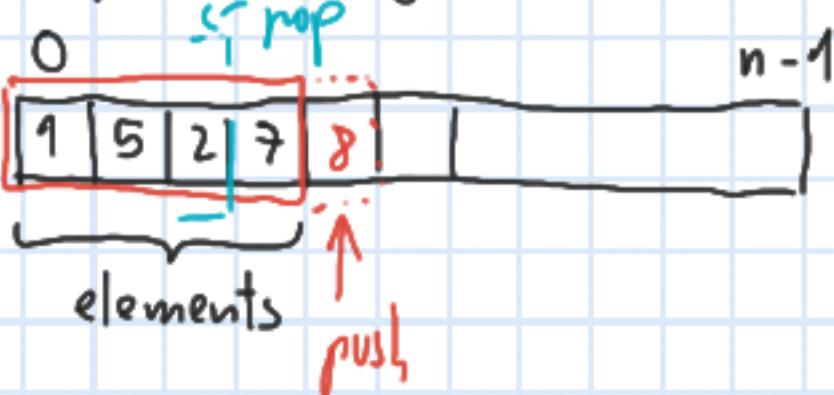
## Operacje

- push(x)
- pop()
- is\_empty()

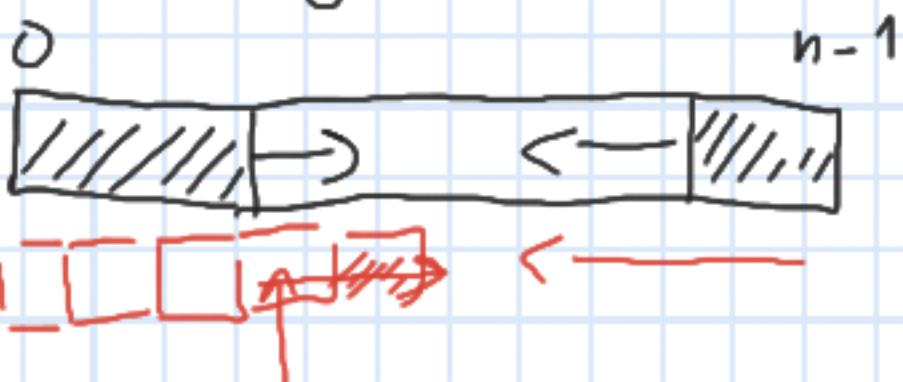
## Implementacja listowa



## Implementacja tablicowa

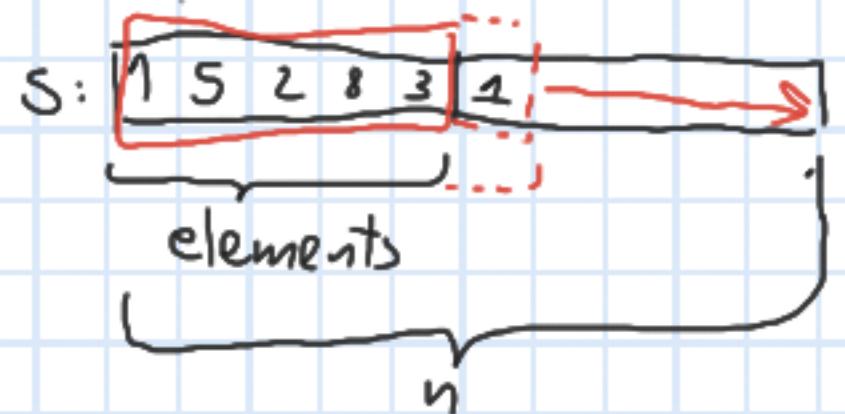


Dwa sposoby?



# ASD - Wykład 5

## Implementacja tablicowa stosu



Co zrobić gdy stos się przepięci?

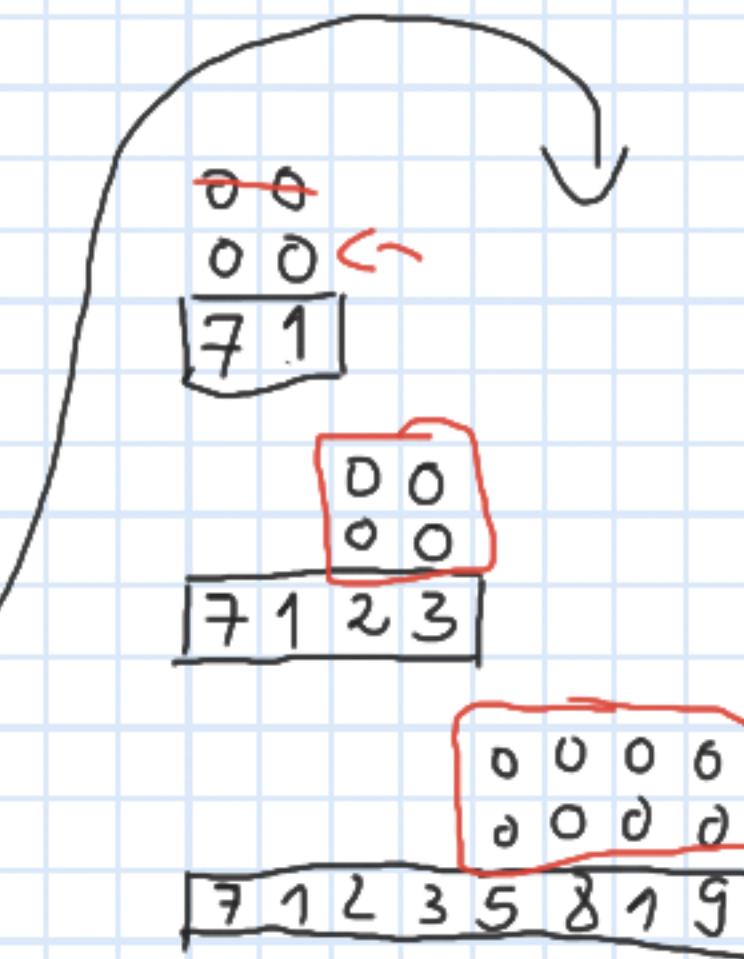
- zaalokować nowy, 2 razy większy i skopiować tam zawartość starego stosu

Analiza pesymistyczna:

- pop  $O(1)$
- push  $O(n)$

Analiza zamożtyzowana:

- push 3 z 1
- pop 1 z 1.

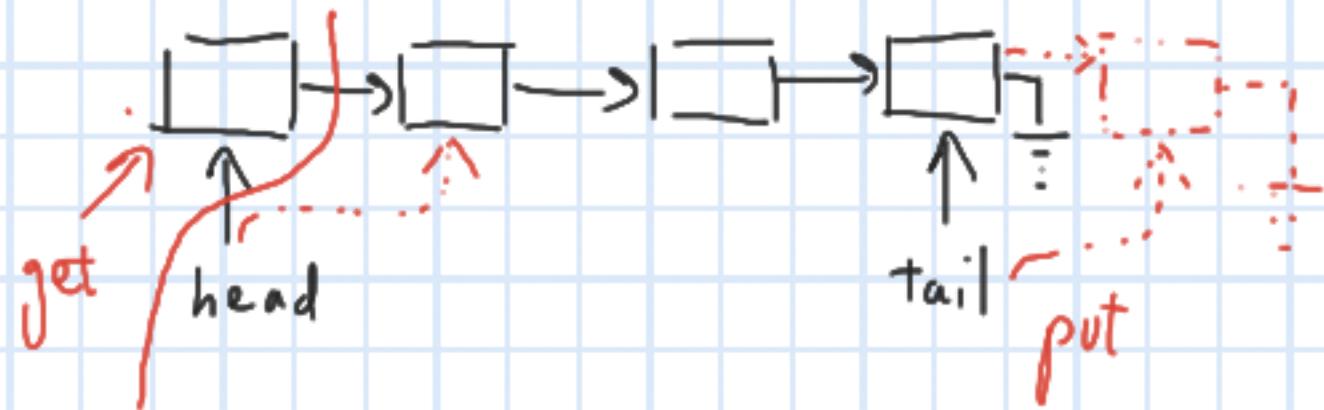


## Kolejka (queue)

Operacje  
 - put  
 - get

"Coś co pozwala ustawić na koniec i pobierać z początku"

## Implementacja listowa



## Implementacja tablicowa



## Kluczowe zadanie

W jaki sposób zaimplementować kolejkę, jeśli do dyspozycji mamy dwa słowy chamy zamontowanej złożoności  $O(1)$  dla  $\text{put}$  i  $\text{get}$

## Kolejka priorytetowa

"co pozwala na umieszczenie danych posortowanych z priorytetem i uciążliwie w kolejności malejących / rosnących priorytetów"

→ kopie (binarny)  
 → posortowana tablica  
 → niesposortowana tablica

lub  
 lista

# Metody konstrukcji algorytmów

- dziel i zuyugzaj ← QuickSort, MergeSort

- algorytmy zachłanne

- programowanie dynamiczne

## metoda zamiary wykładniego

algorytmu rekurencyjnego na wiedomiany

algorytm iteracyjny naucz spamisty uanic

## Uyników

## Prykład elementarny

$$F_1 = 1$$

$$F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

```
def fib(n):
```

if  $n \leq 2$ : return 1

return fib(n-1) + fib(n-2)

$f_l(n)$

$\text{fib}(n-1)$

$$f.b(n-2)$$

$$\sin(n-3)$$

$$\cancel{f_{1,b}(n-2)}$$

$$f.b(n-4)$$

## Implementasi dynamic programming

```
def fib_dyn( n ):  
    F = [1] * (n + 2)  
    for i in range(2, n + 1):  
        F[i] = F[i - 1] + F[i - 2]  
    return F[n]
```

```
def fib_dyn2( n ):  
    if n <= 1: return 1  
    F1 = 1  
    F2 = 1  
    for i in range(2, n + 1):  
        Fi = F1 + F2  
        F2 = F1  
        F1 = Fi  
    return Fi
```

a copy  
+  
loop.  
/eng?

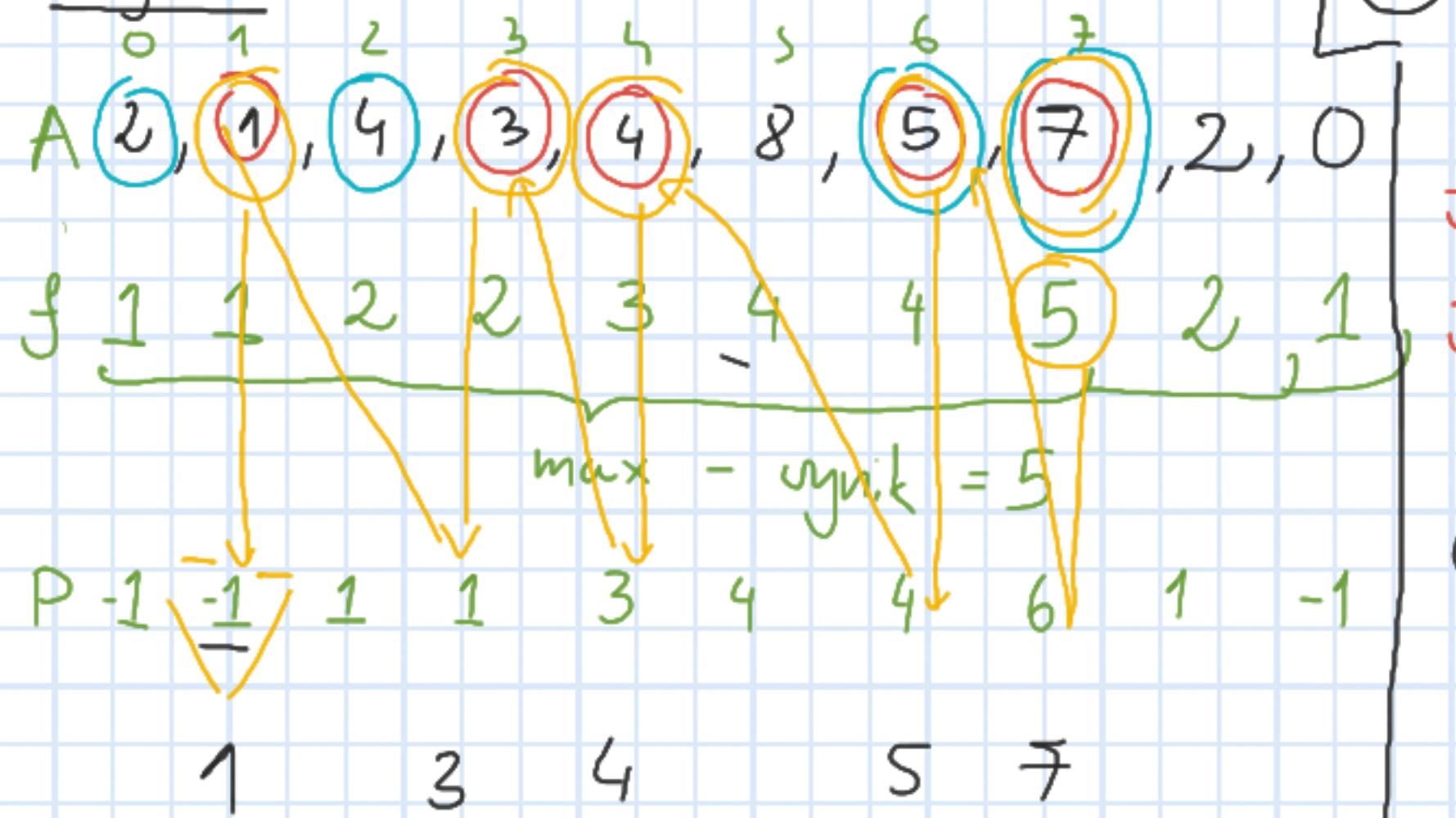
# ASD - Użytkad 6

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)]
```

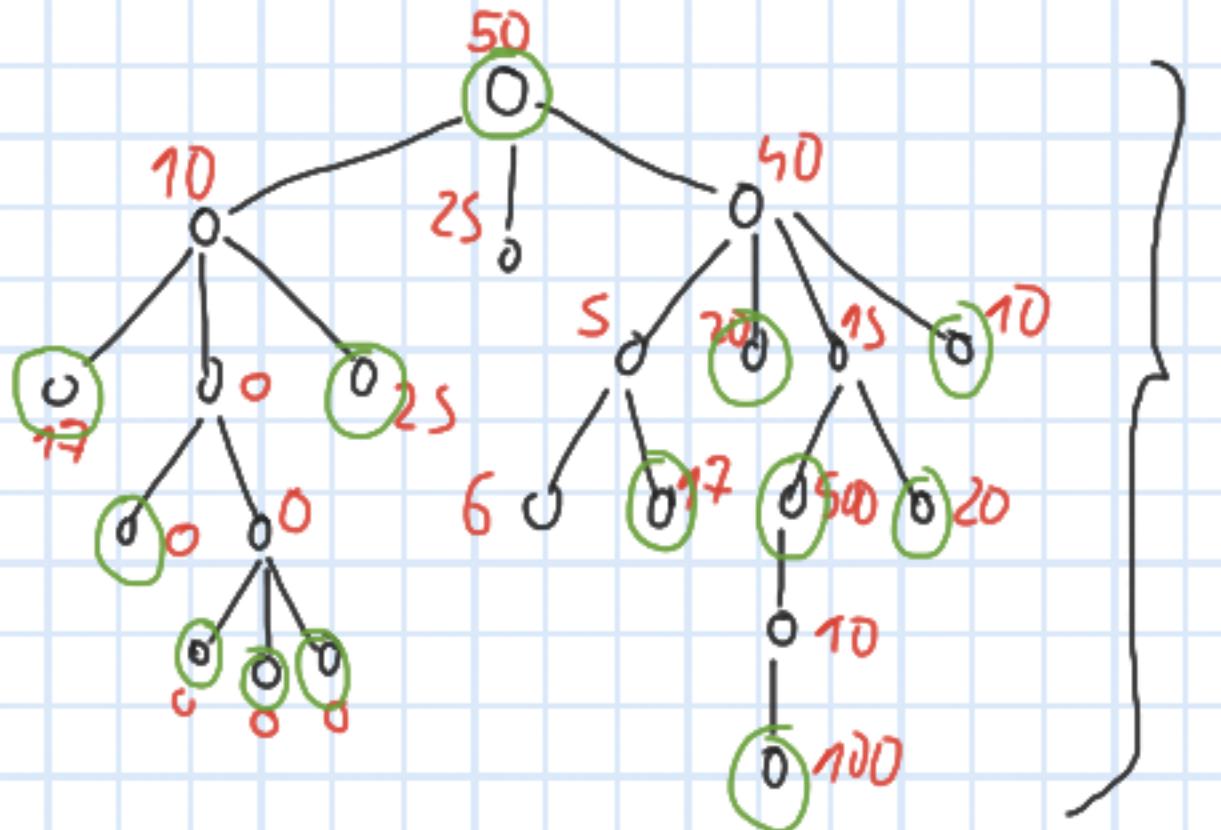
```
    for i in range(1, n):  
        for j in range(i):  
            if A[i] > A[j] and F[j]+1 > F[i]:  
                F[i] = F[j]+1  
                P[i] = j  
  
        if F[i] > F[maxi]:  
            maxi = i  
  
    return maxi, F, P
```

```
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 Zalczność 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):
```

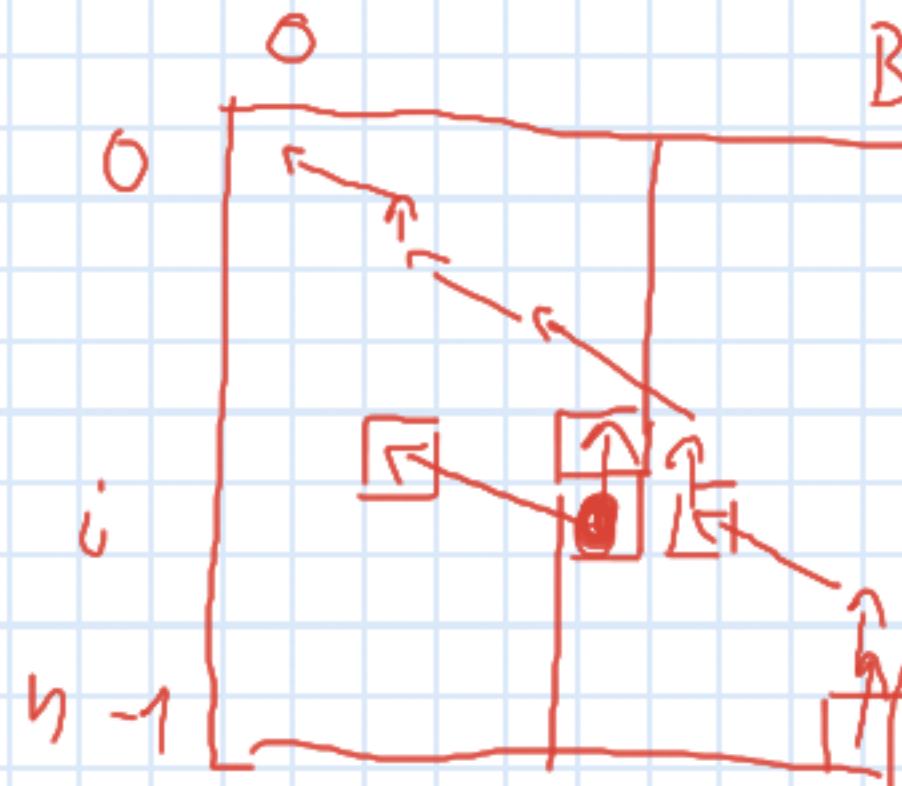
```
            F[i][b] = F[i-1][b]
```

```
            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

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

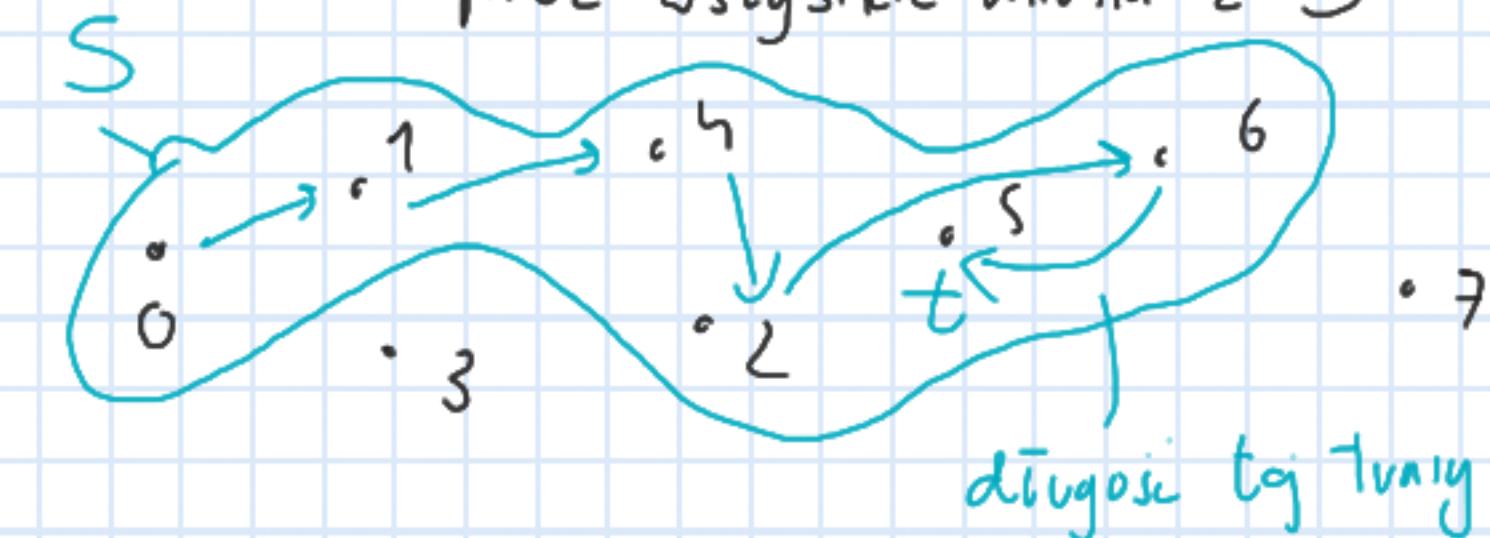
efektywność pamięciowa

## 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  $S$



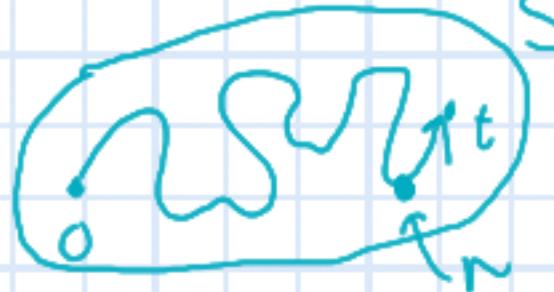
## Rozwiązywanie

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

## 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

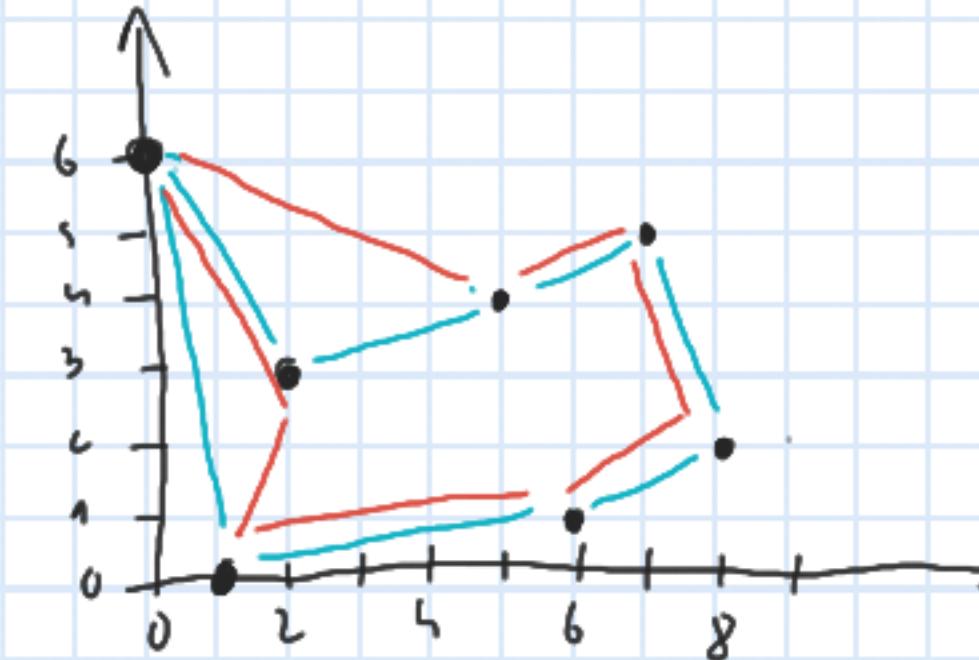
$$\Theta(n \cdot 2^n)$$

paragonowa

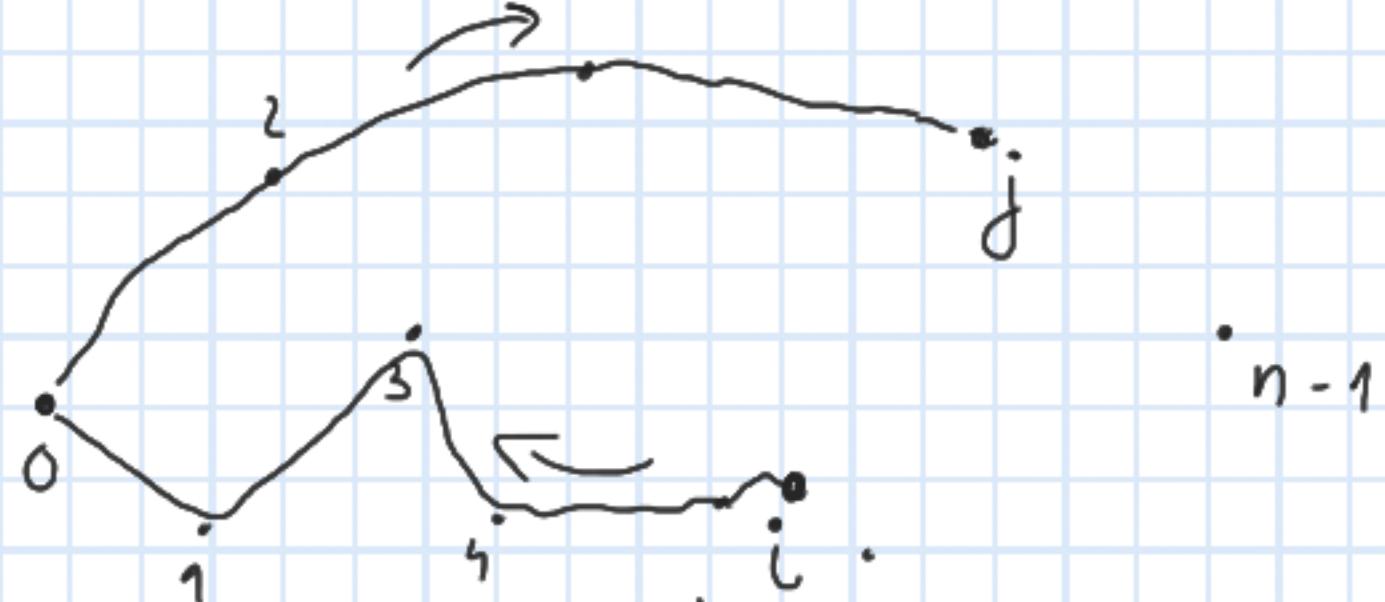
## Bitoniczny problem komiwojażera

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

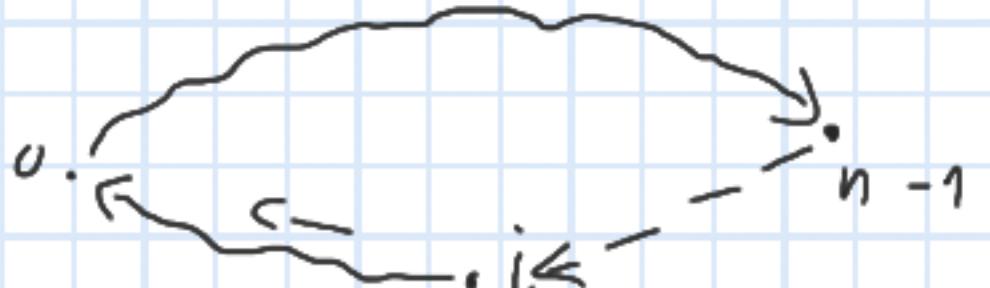


$f(i, j) = \begin{cases} \text{koszt suieku z } 0 \text{ do } i \text{ oraz} \\ \quad 2 \text{ } 0 \text{ do } j, \\ i < j \end{cases}$

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

## Rozwiązywanie

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

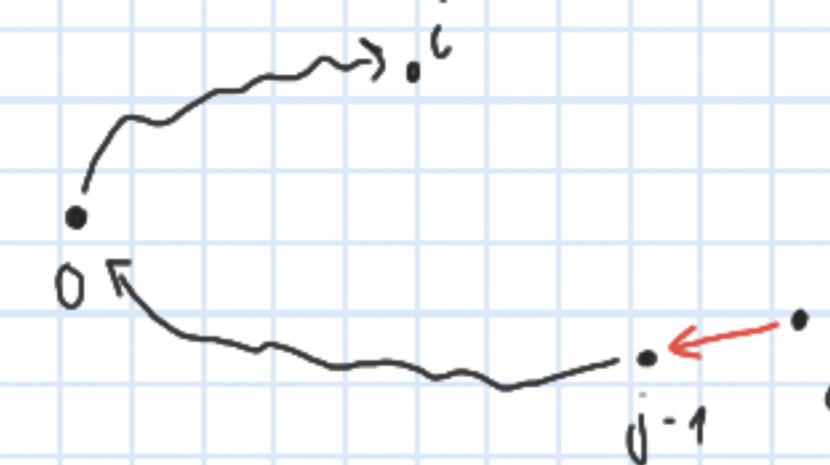


## Zapis rekurencyjny funkcji f

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

Następnie rozważamy dwa przypadki

a)

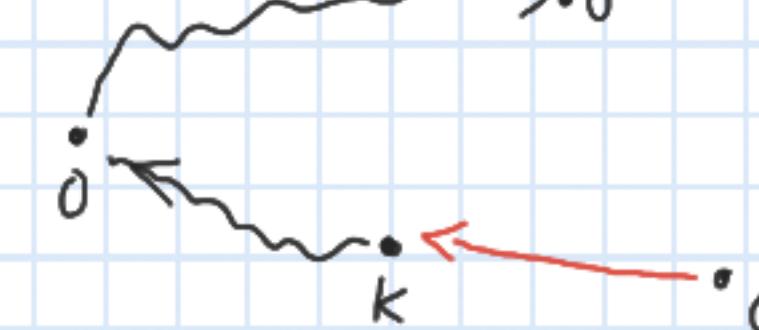


$$f(i, j) = f(i, j-1) + d(j-1, j)$$

$$i < j-1$$

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

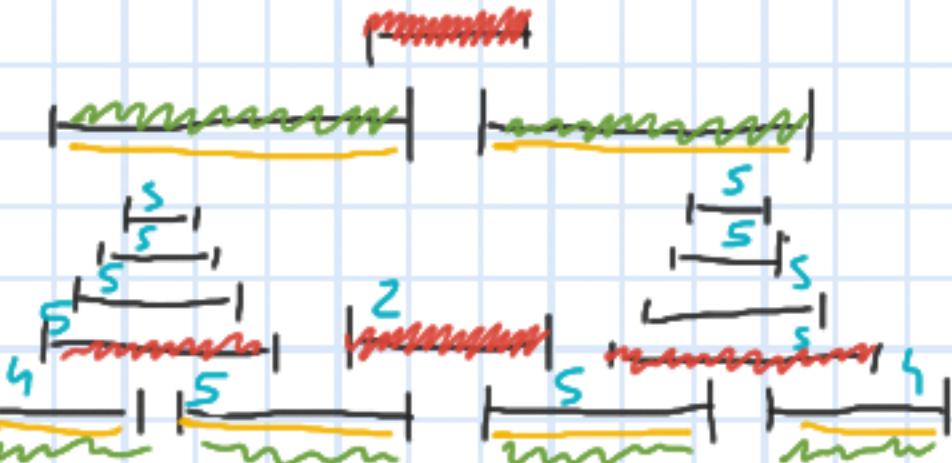
## Problem wyboru zadań

Dane: zbiór przedziałów (zadań)

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

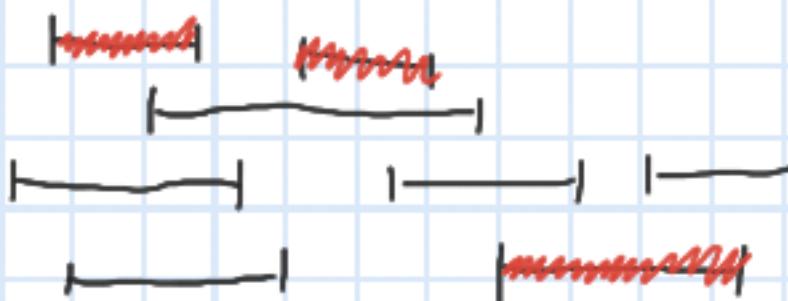
## Pomyśły

① "najkrótszy najpierw"



② "najmniej przeciąga najpierw"

③ "najwcześniej konczy się najpierw"

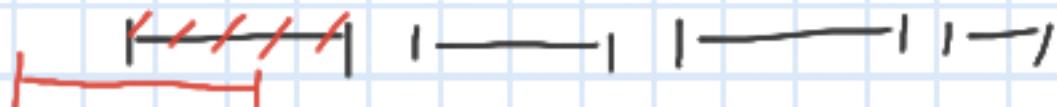


## Uzasadnienie poprawności

Rozważmy dowolne rozwiążanie optymalne

- jeśli zawiera najwcześniej konczy się przedział to OK

- jeśli nie zawiera to: dokładamy najwcześniej konczy 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

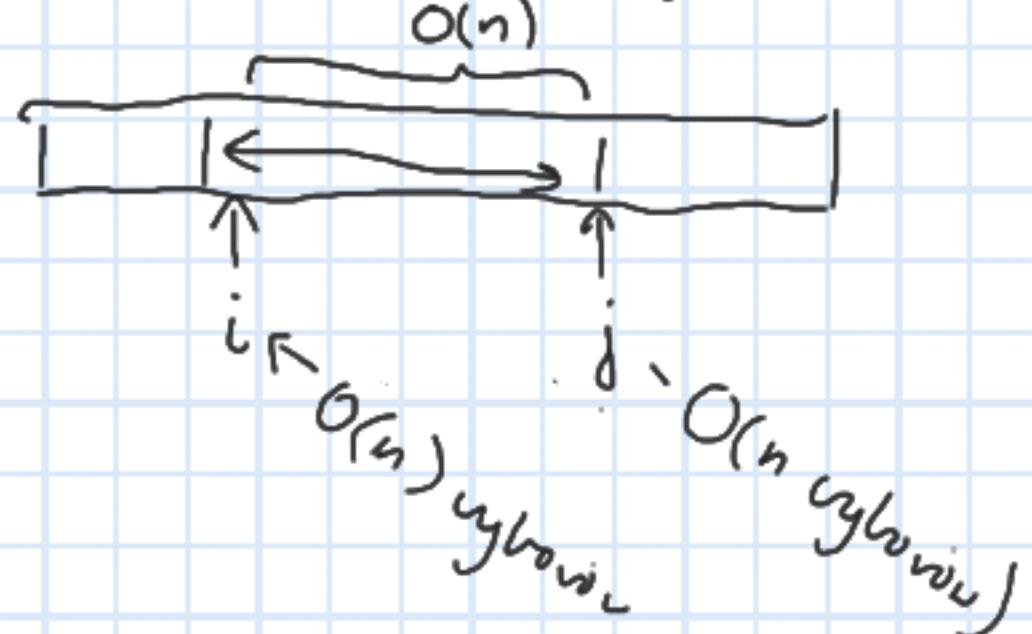
Dane:  $A[0, \dots, n-1]$  - tablica liczb całkowitych

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

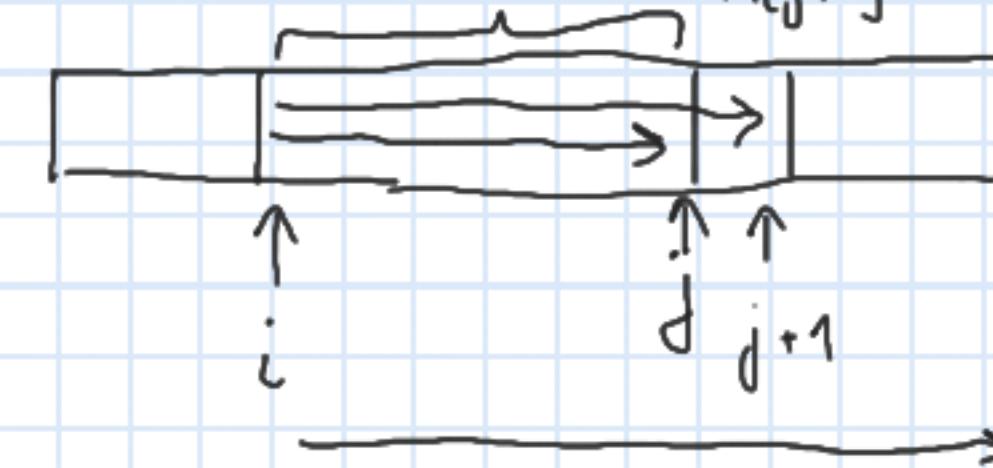
## Poniedziałek

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

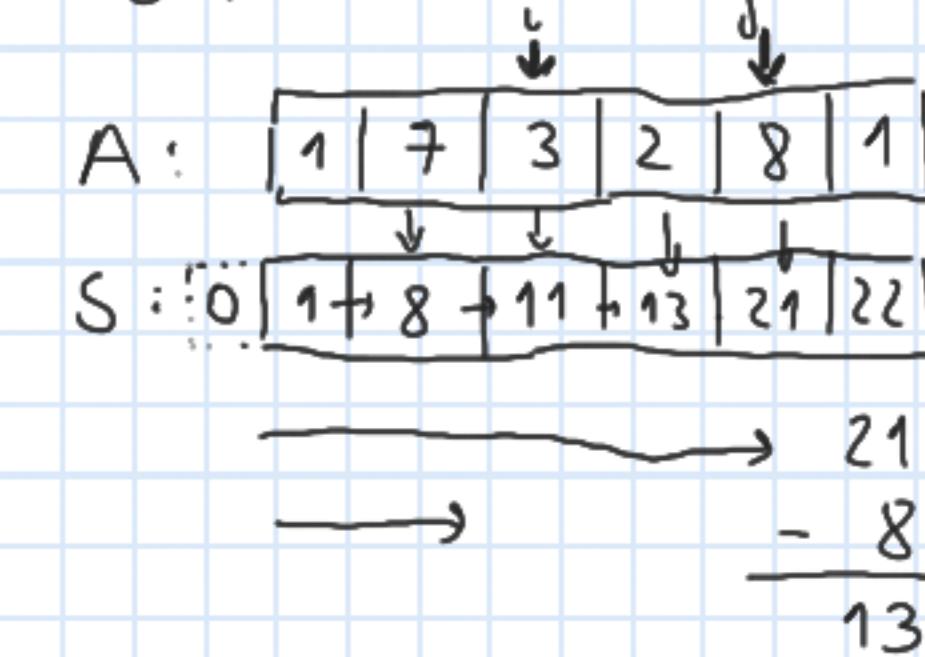
① Elementarne rozwiązywanie  $O(n^3)$



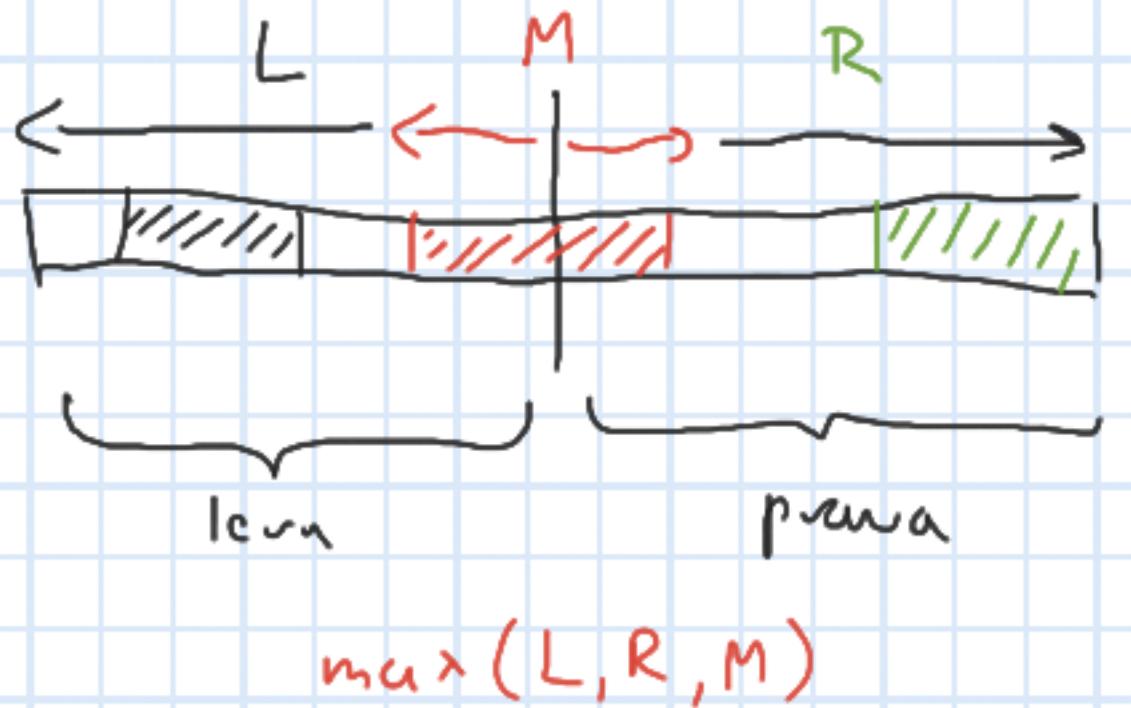
② Algorytm  $O(n^2)$  przez sumy prefiksowe



Sumy prefiksowe:



### ③ Algorytm dziel-i-zwyciążaj



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

### ④ Programowanie dynamiczne

spójnego

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

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

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

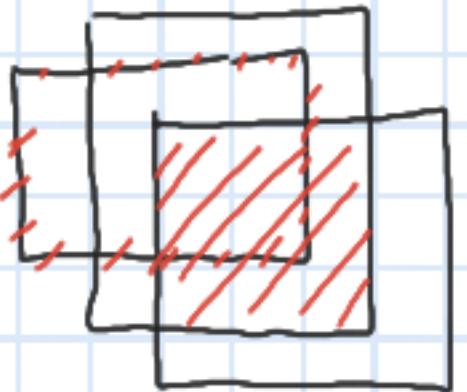
$$A: 1, 3, -7, -2, 1, 9, -3, 2, -100, 4$$

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

ale trzeba zachować!

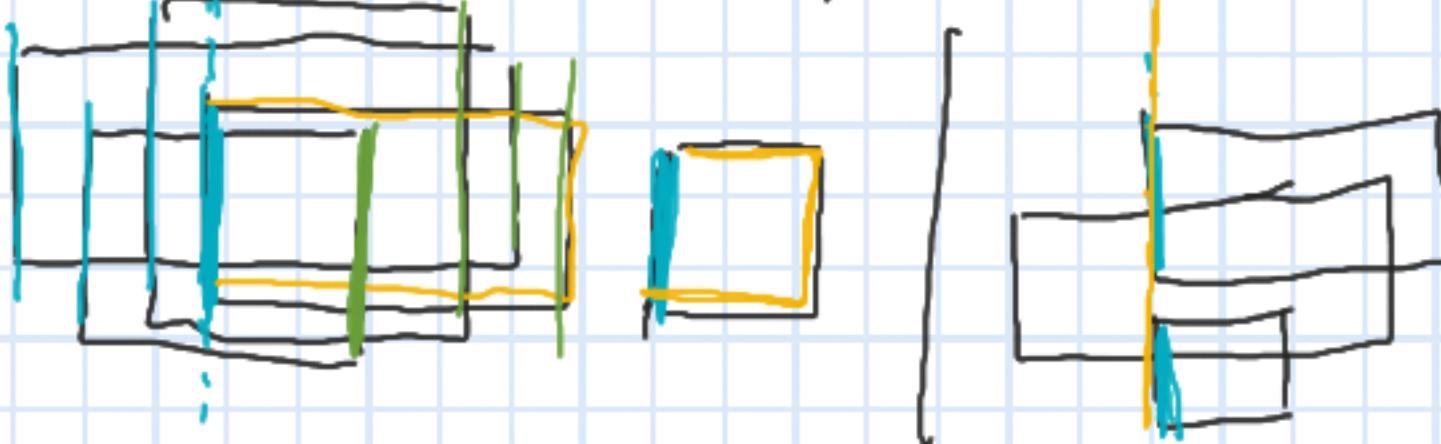
# Problem Pniewienia prostokątów

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  $O(n)$

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

$\mathbb{R}^2$

$$\left. \begin{array}{l} P_1 \\ P_1 \cap P_2 \\ P_1 \cap P_2 \cap P_3 \\ P_1 \cap P_2 \cap P_3 \cap P_4 \\ P_1 \cap P_2 \cap P_3 \cap P_4 \cap P_5 \\ P_1 \cap P_2 \cap P_3 \cap P_4 \cap P_5 \cap P_6 \\ P_1 \cap P_2 \cap P_3 \cap P_4 \cap P_5 \cap P_6 \cap P_7 \end{array} \right\} A$$

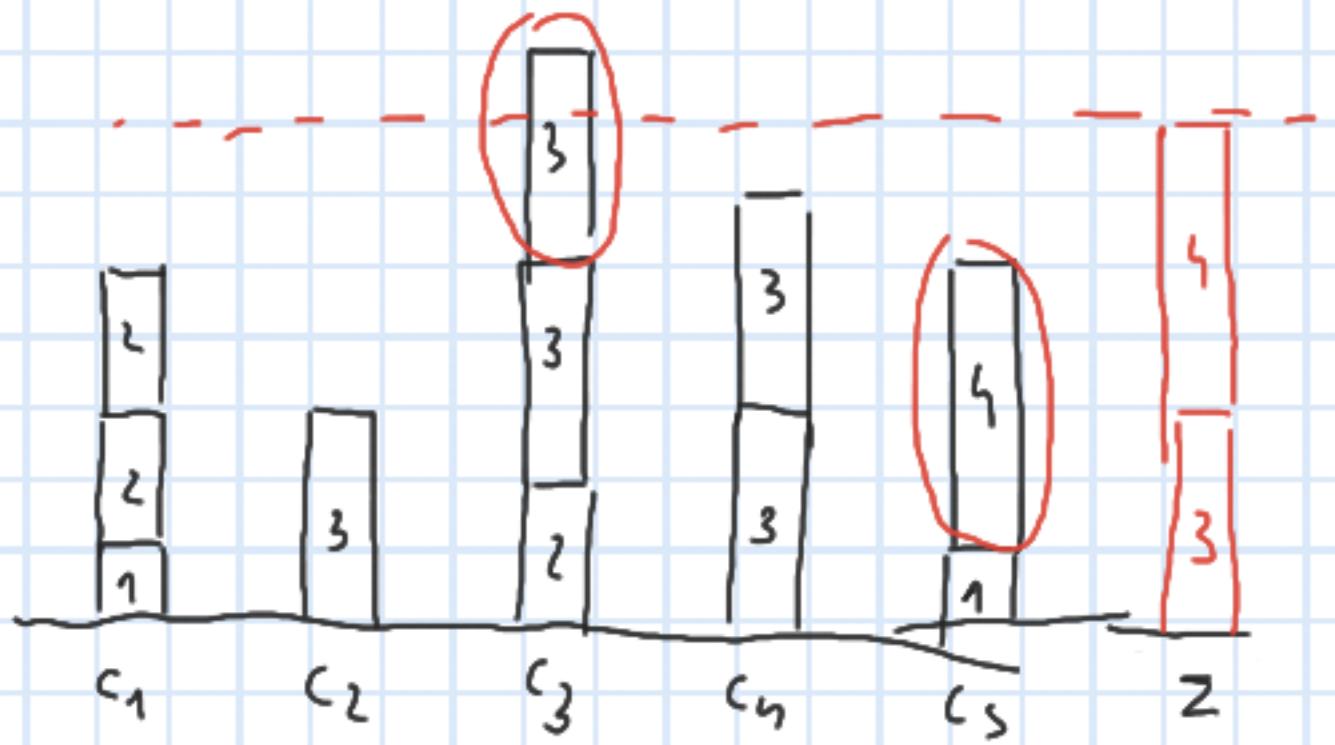
$$\left. \begin{array}{l} P_3 \cap P_4 \cap P_5 \cap P_6 \cap P_7 \\ P_4 \cap P_5 \cap P_6 \cap P_7 \\ P_5 \cap P_6 \cap P_7 \\ P_6 \cap P_7 \\ P_7 \end{array} \right\} B$$

$\mathbb{R}$

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

## Problem najwyższej wieży

Dane:

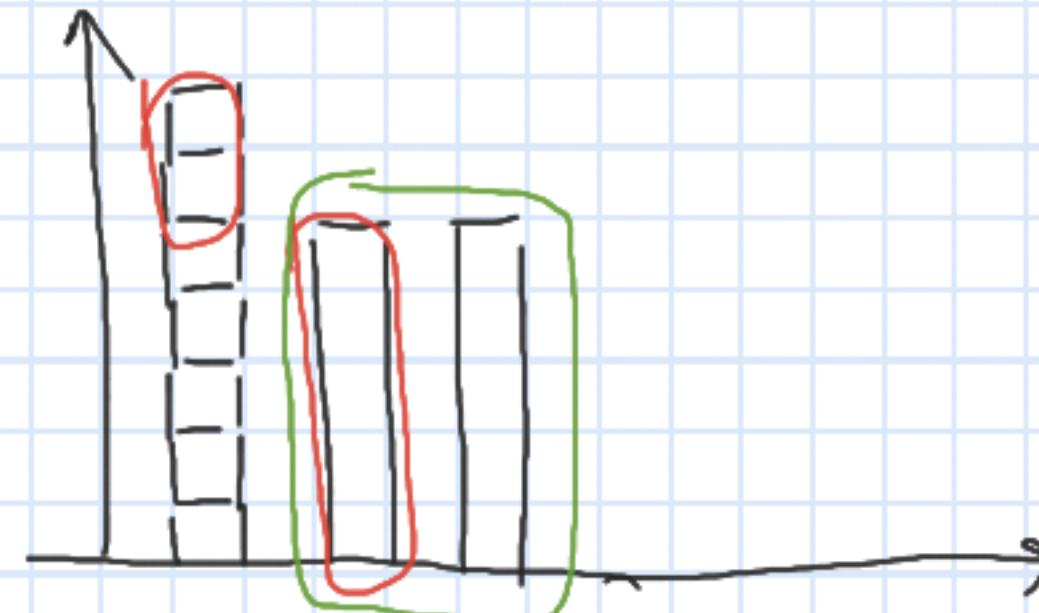


① Kradnij największe klocki



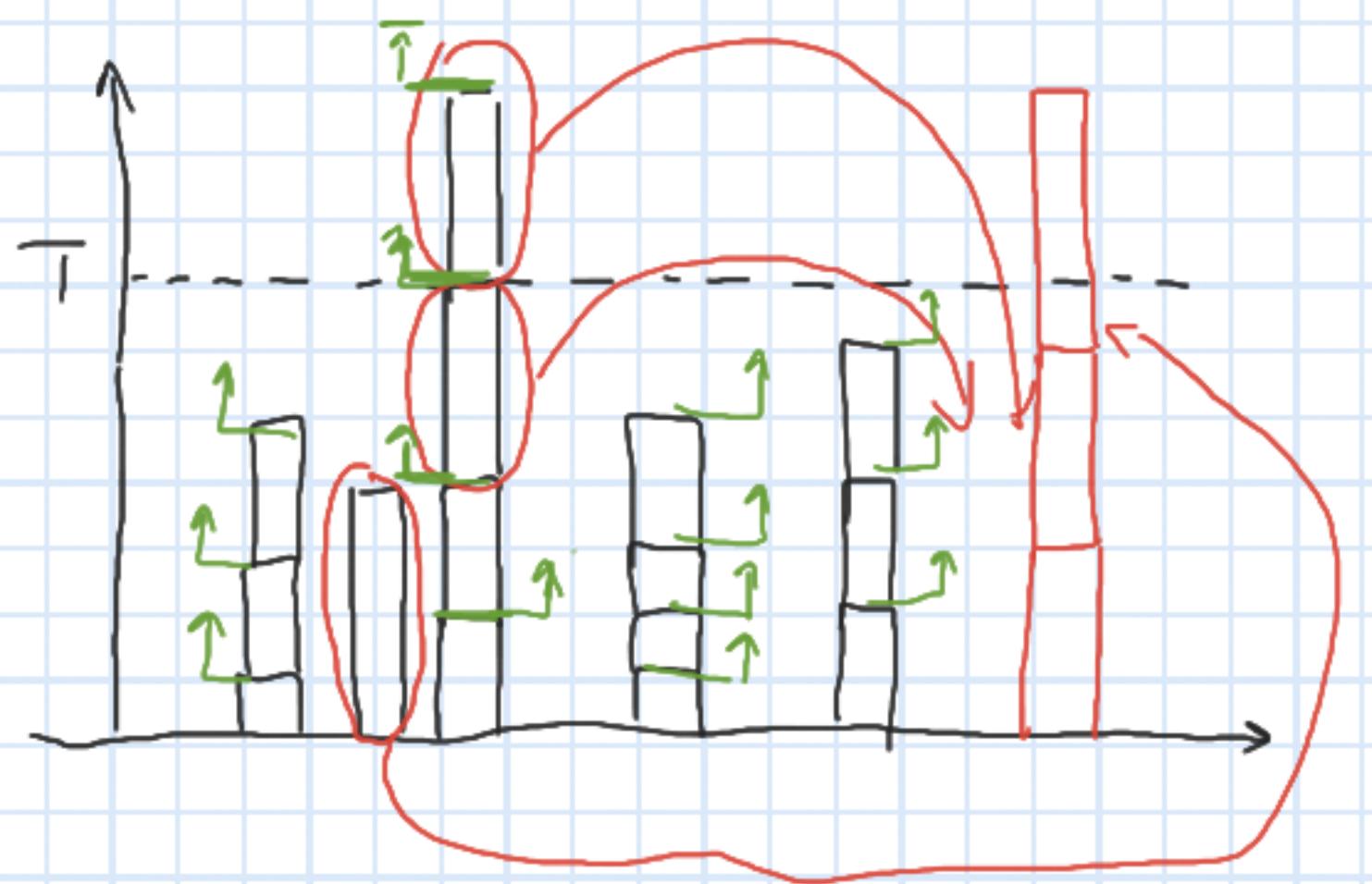
Zadanie: uchwyc jaką najmniejszą liczbę klocków taka, żeby uzyskać najwyższą wieżę

② Kradnij największe klocki z najwyższą wieżą



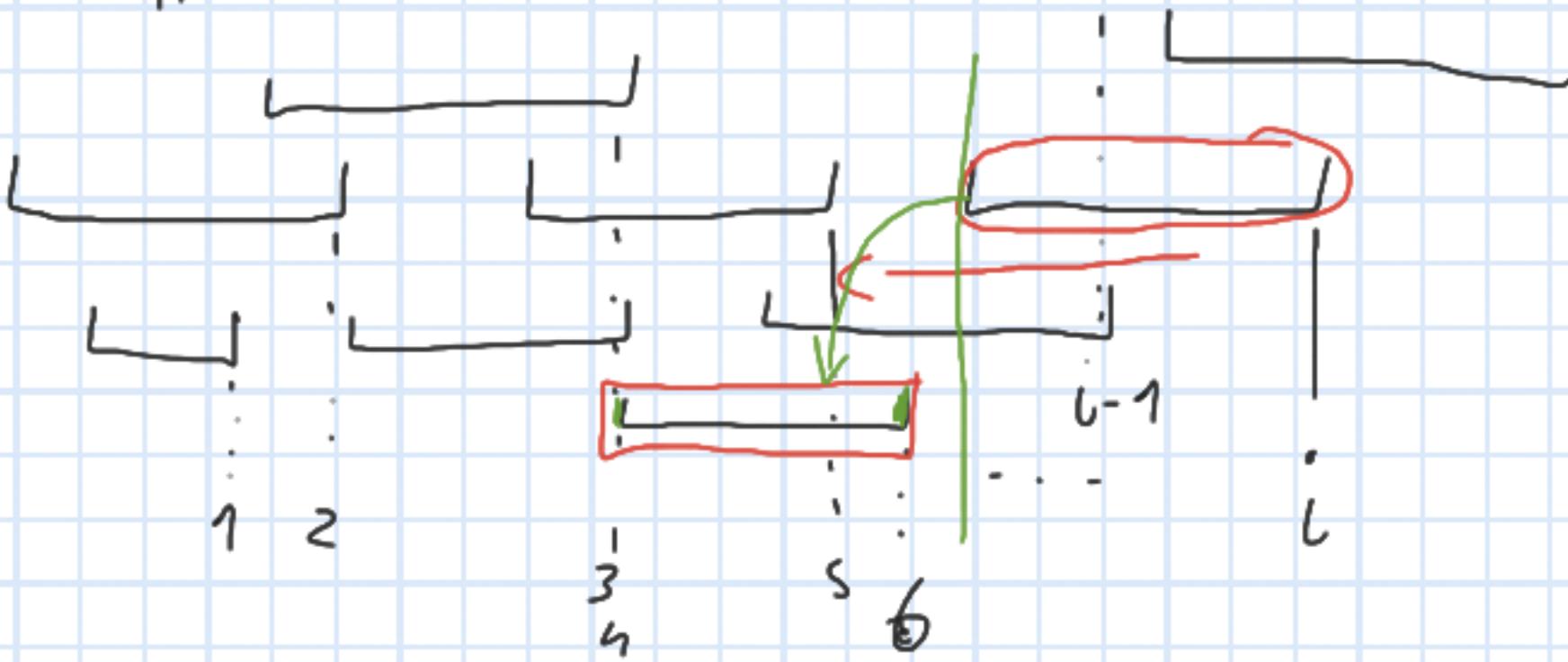
③

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

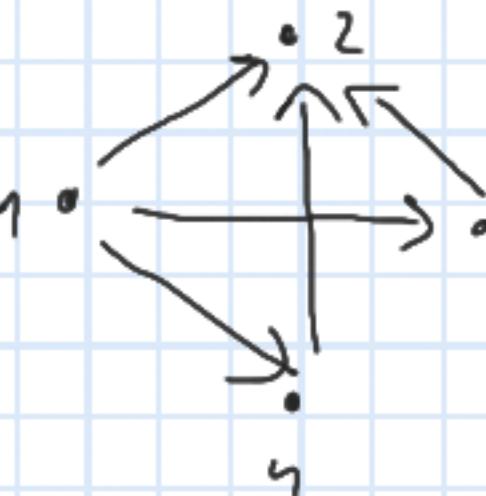
Graf skierowany:

$$G = (V, E)$$

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

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

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



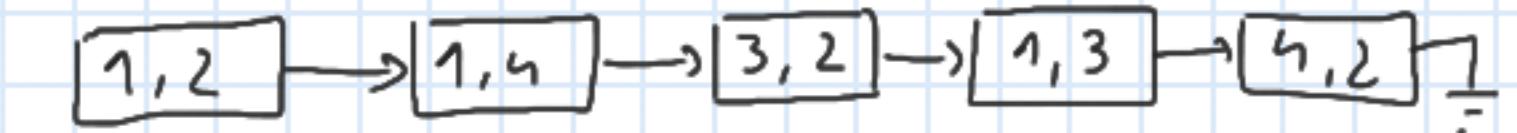
Z kązdy krawędzią / wierzchołkiem  
można zaznaczyć dodatkowe informacje

Graf nieskierowany - krawędzie są zbiorem  
dwuelementowym

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

## Reprezentacje grafów

Lista / tablica krawędzi

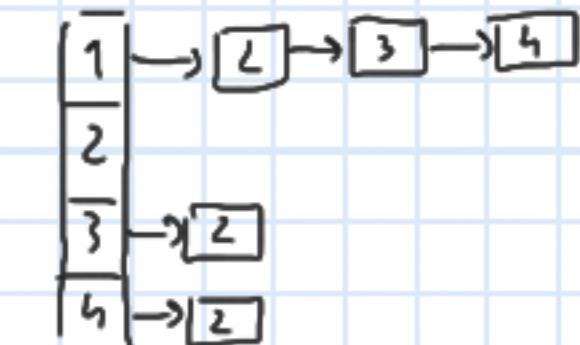


Reprezentacja macierzowa

	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:
```

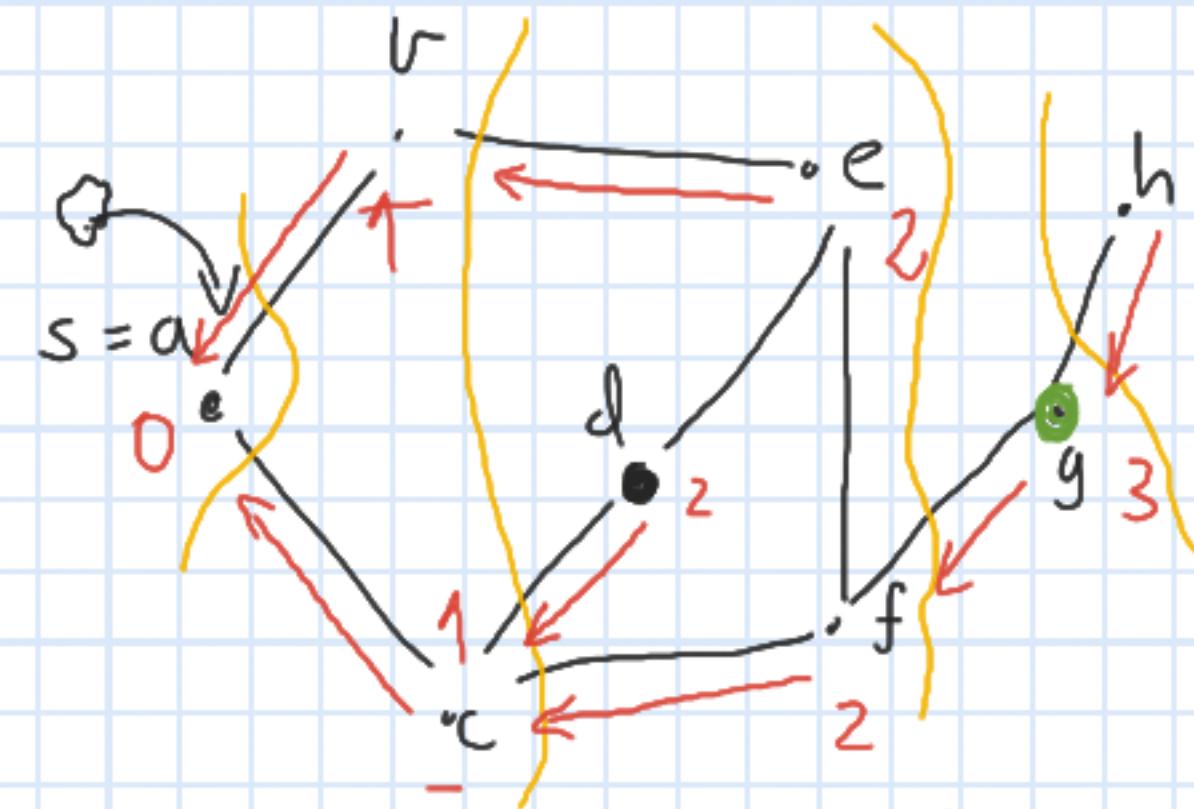
```
        if not v.visited:
```

```
            v.visited = True
```

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

```
v.parent = u
```

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



Złożoność

rep. maturalna

$O(V^2)$

rep. listowa

$O(V + E)$

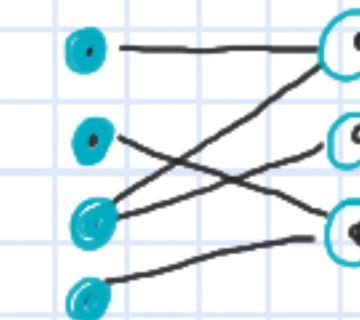
Zastosowania

- najkrótsze ścieżki (w grafie bez wag)

- spójność grafu

- wykrywanie cykli

- divedzimusi



# Algorytm DFS (depth-first search)

Pniewzkuwanie w gętce

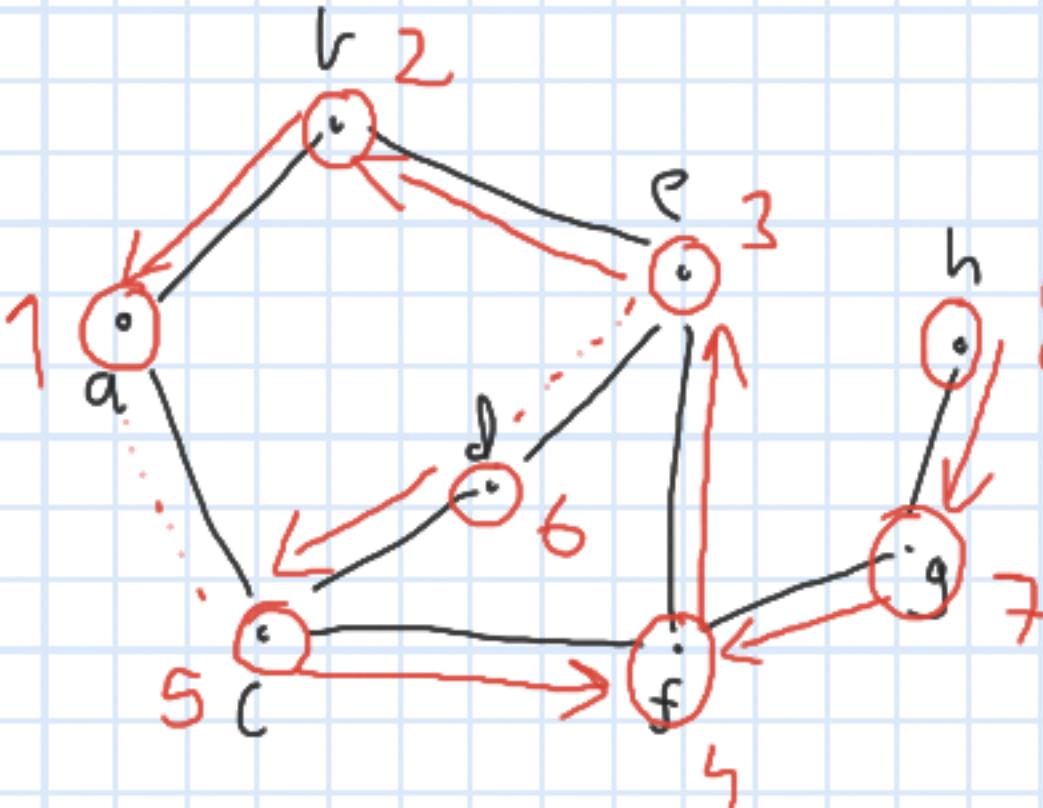
```
def DFS( G ):
    # G = (V, E)
    for v ∈ V:
        v.visited = False
        v.parent = None
    time = 0
```

```
    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 wdrożek zostało odwiedzony / was odwiedzony*

for v - sąsiad u:

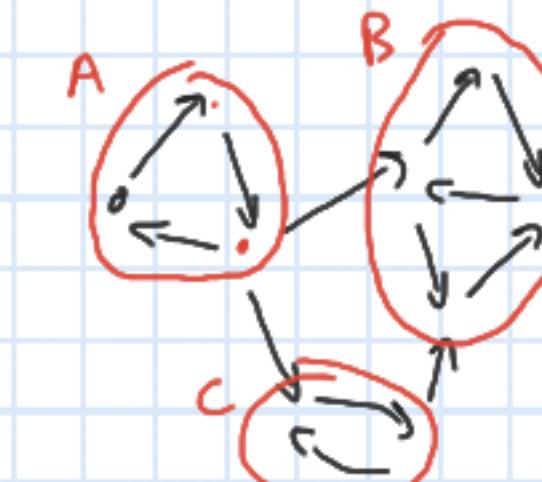
if not v.visited:

v.parent = u

DFSVisit(G, v)

time += 1

← *v zostało pociągnięty / was pociągnięty*



## Zastosowania

- spójność

- dwudzielność

- wykrywanie cykli

- sortowanie topologiczne



- cykl Eulera

- silnie spójne

skladowe

- mosty / płat. cuty kolumni



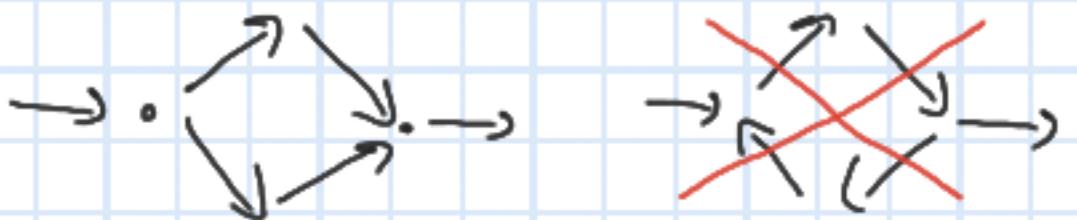
## ASD - Uglystad 10

Kolokrium - mugi złożoności

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

## Sortowane topologiczne

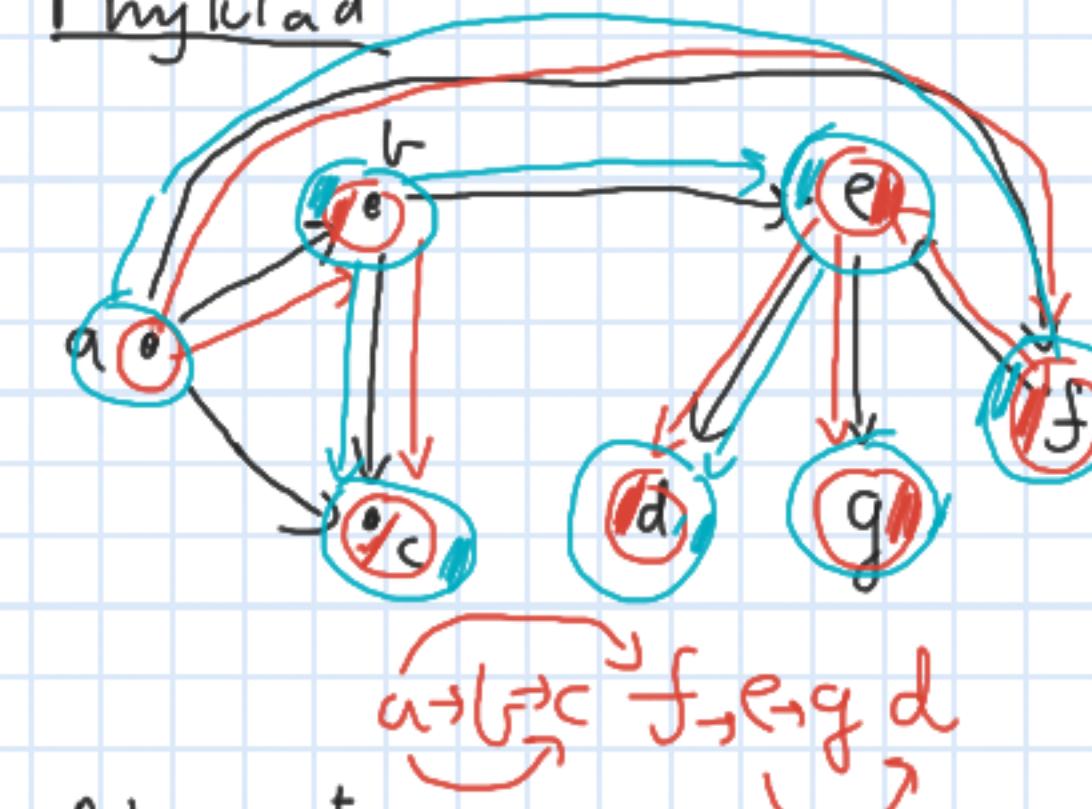
dag - directed acyclic graph



Sortowanie topologiczne dag'u - utwierdzenie

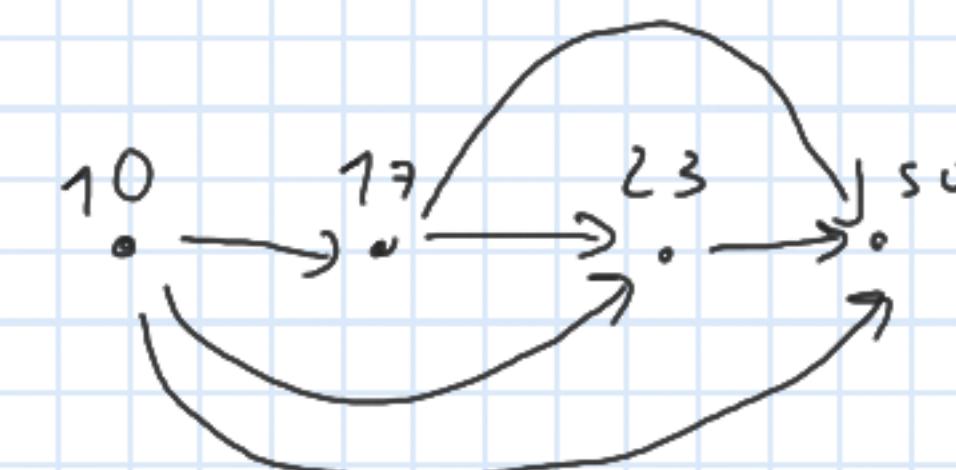
ciendoteków tak, aby wizualnie krajobraz uskazywał "z lewa na prawo"

## Pmylutad



## Algorytm

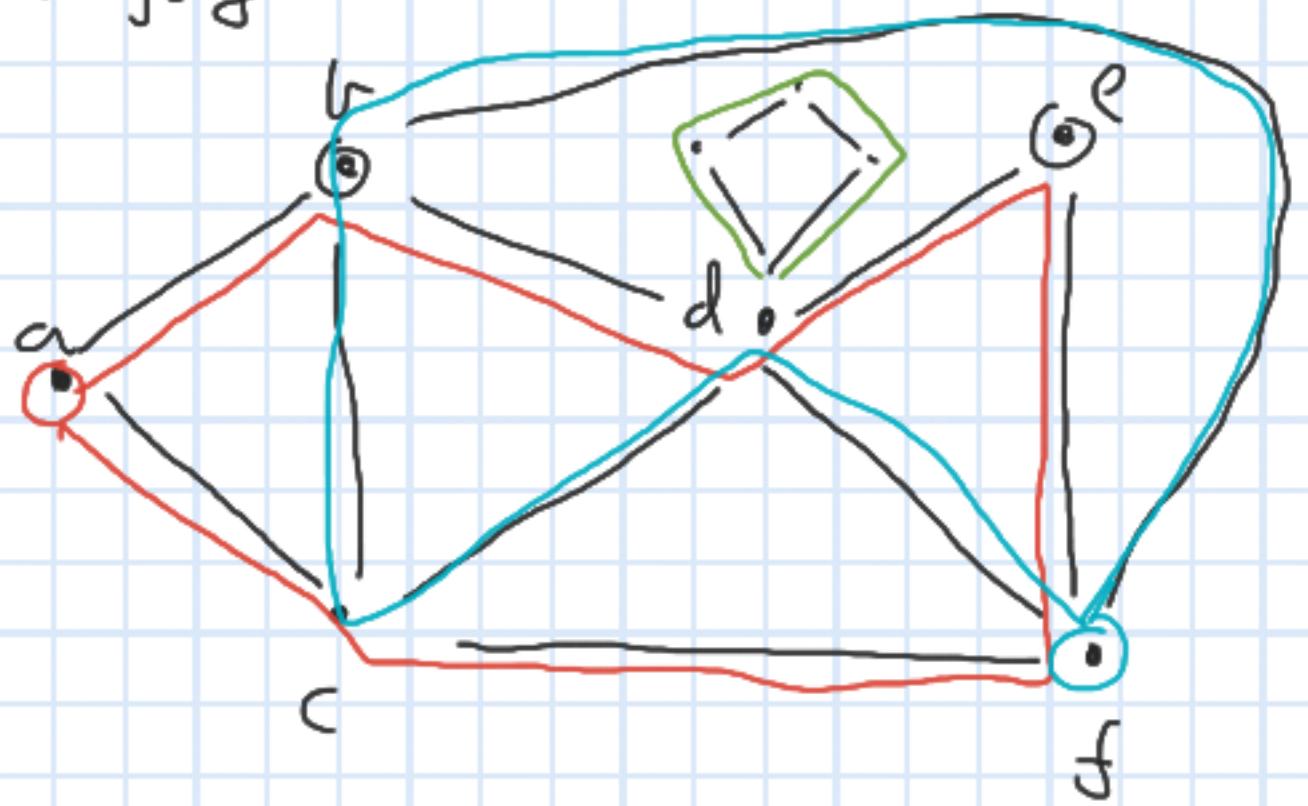
- wchodzimy DFS
- po zwroceniu ciendotki dopisujemy na koniec listy



## Cykl Eulera

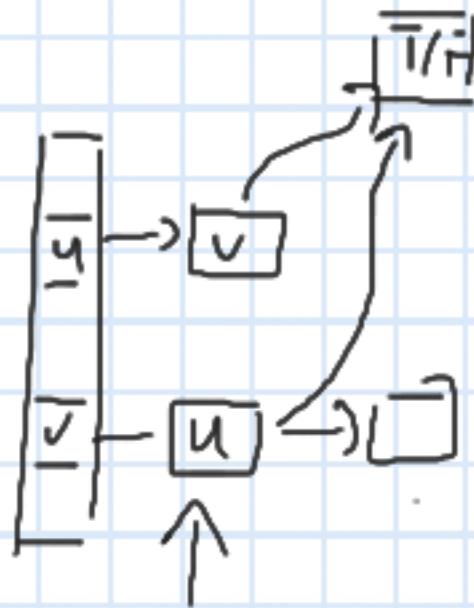
def Cykl Eulera to cykl, który przechodzi przez wszystkie krawędzie, odwiedzając każdą dokładnie raz

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

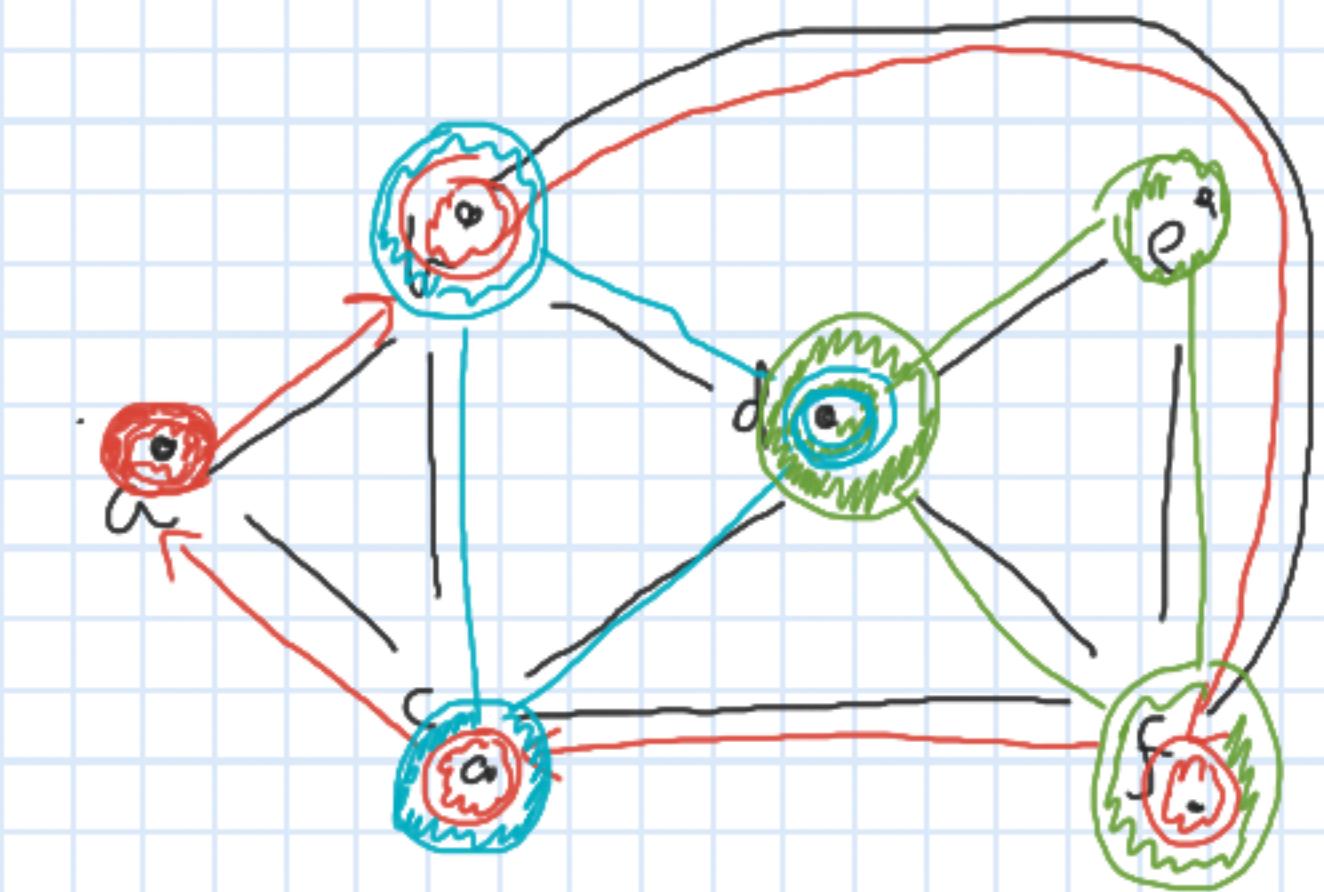


## Algorytm

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



$O(V+E)$



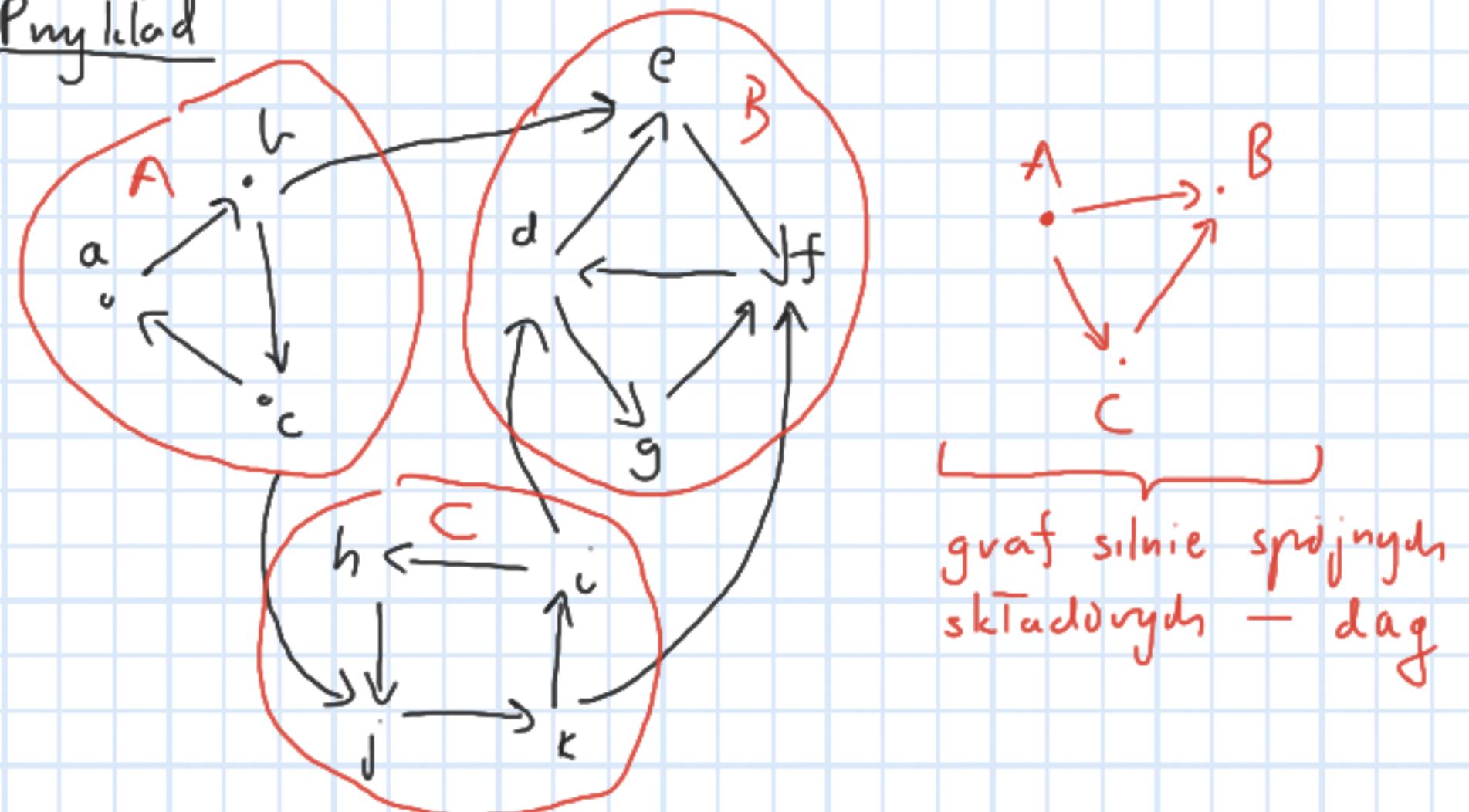
a b f c b d e f d c a

## 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  $u$

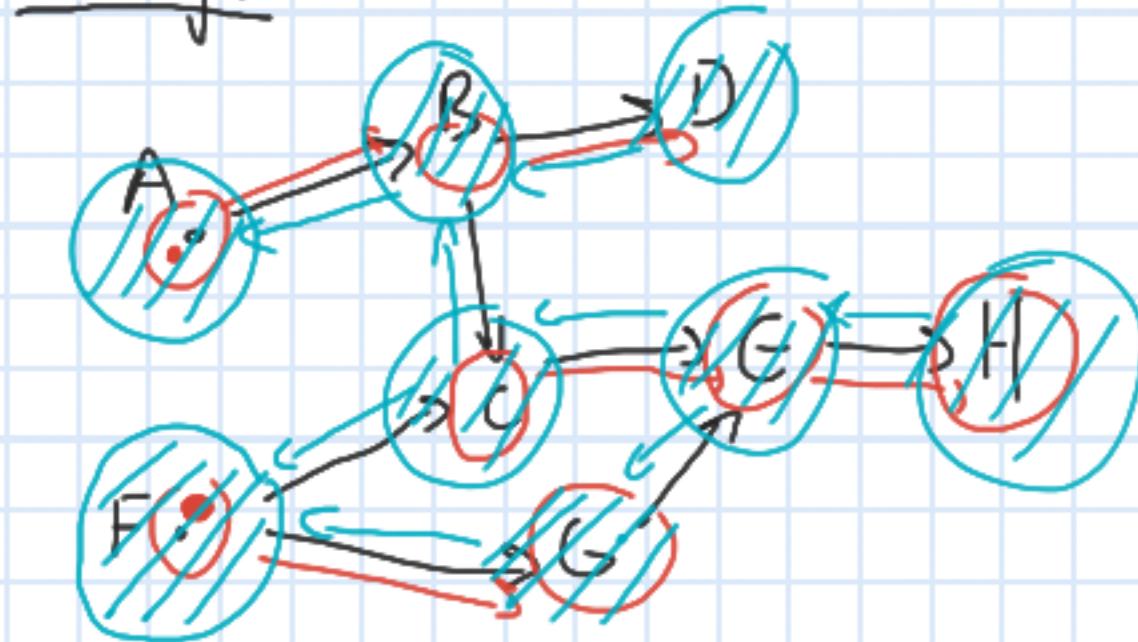
## Ponkład

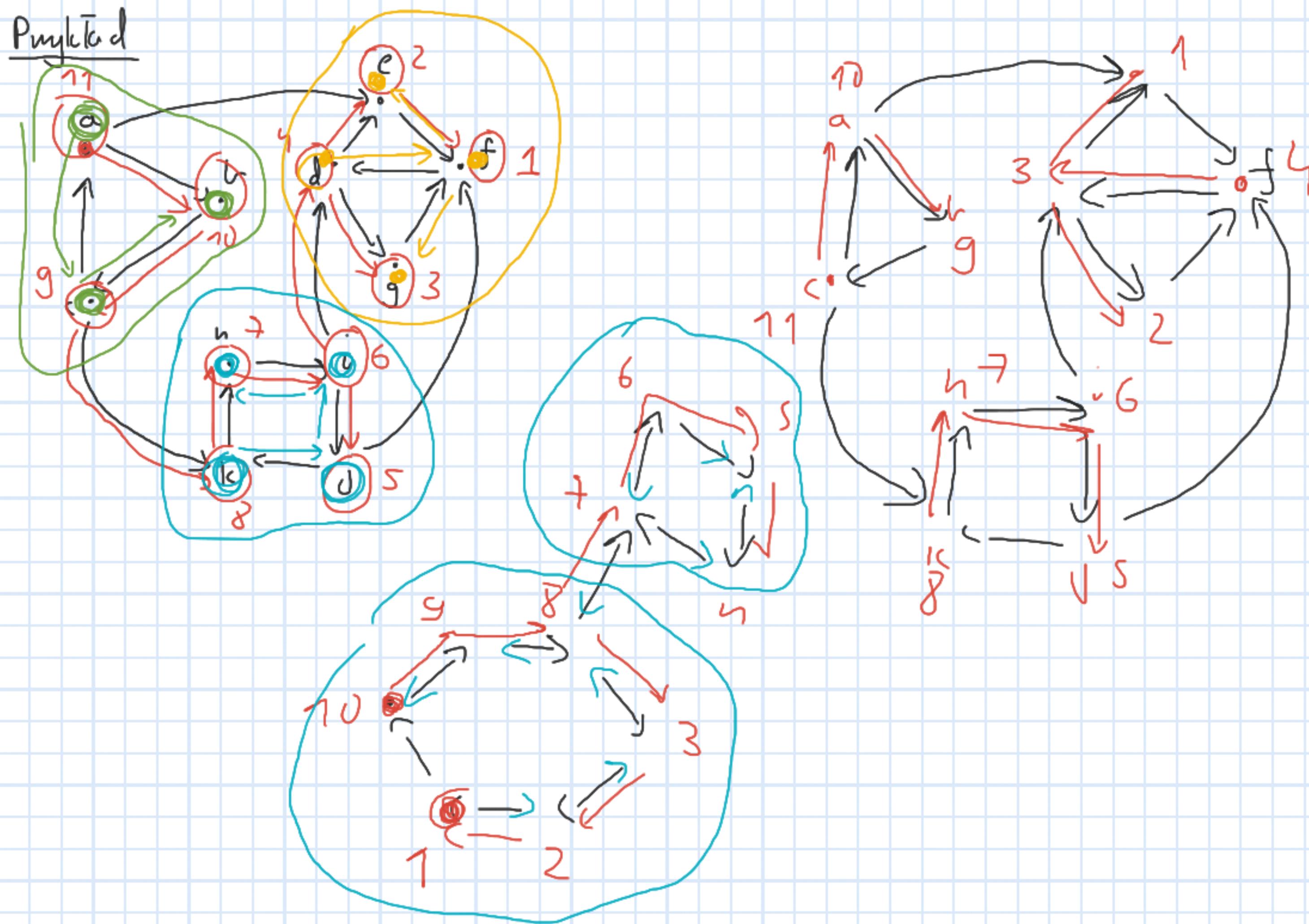


## Algorytm

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:

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

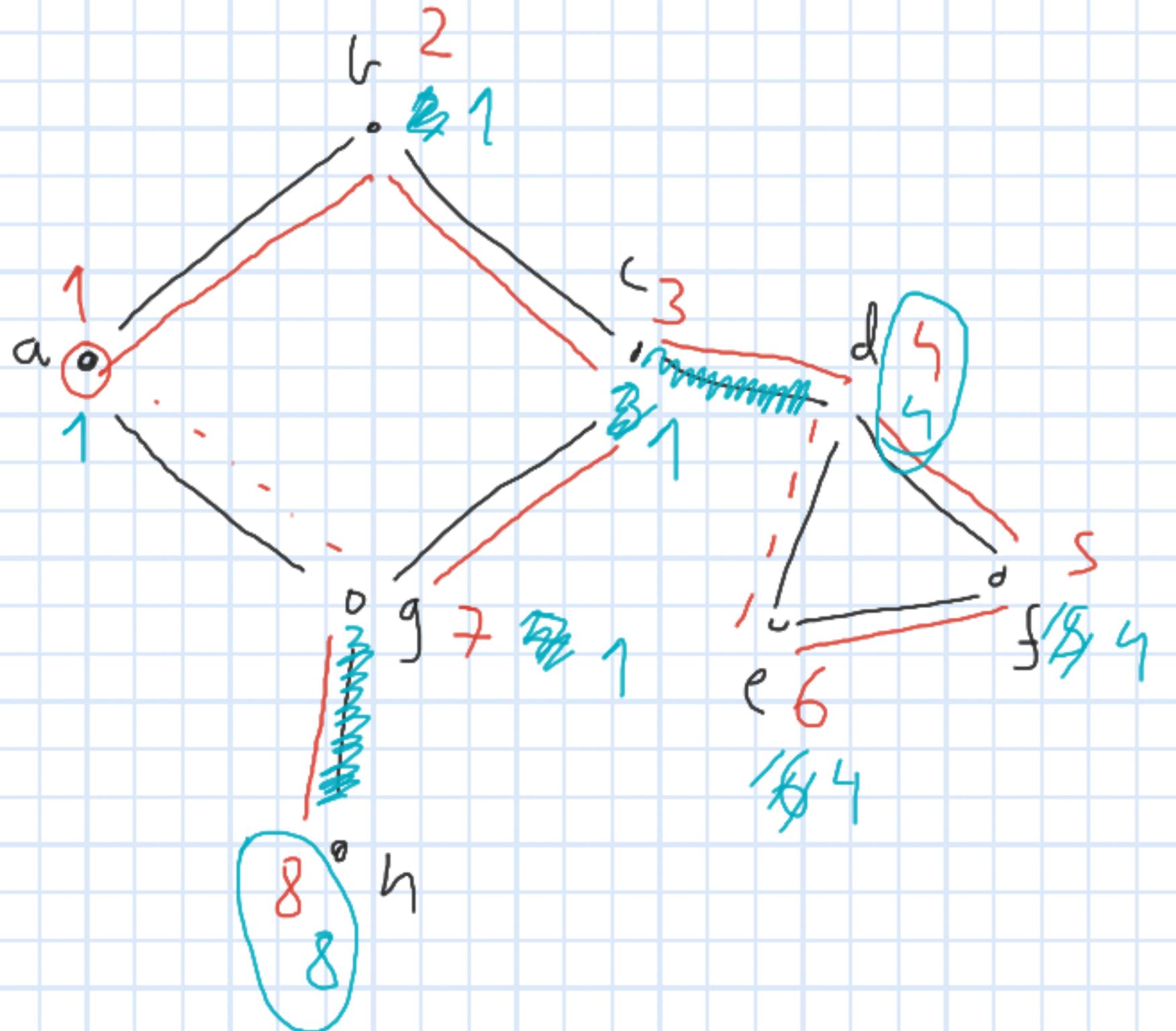
$$\min_{w - \text{drzeka } v \text{ w drzewie DFS}} \text{low}(w)$$

③ Mosty to krawędzie

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

$$d(v)$$

$|v|_U(v)$



ASD - Wykład 11

def Krawędź e w spójnym grafie nieslurowanym jest mostem jeśli jej usunięcie rozspojnią graf

tu Krągdzi e jut mostem wtu. gdy  
nie lezy na zadnym cyklu prostym

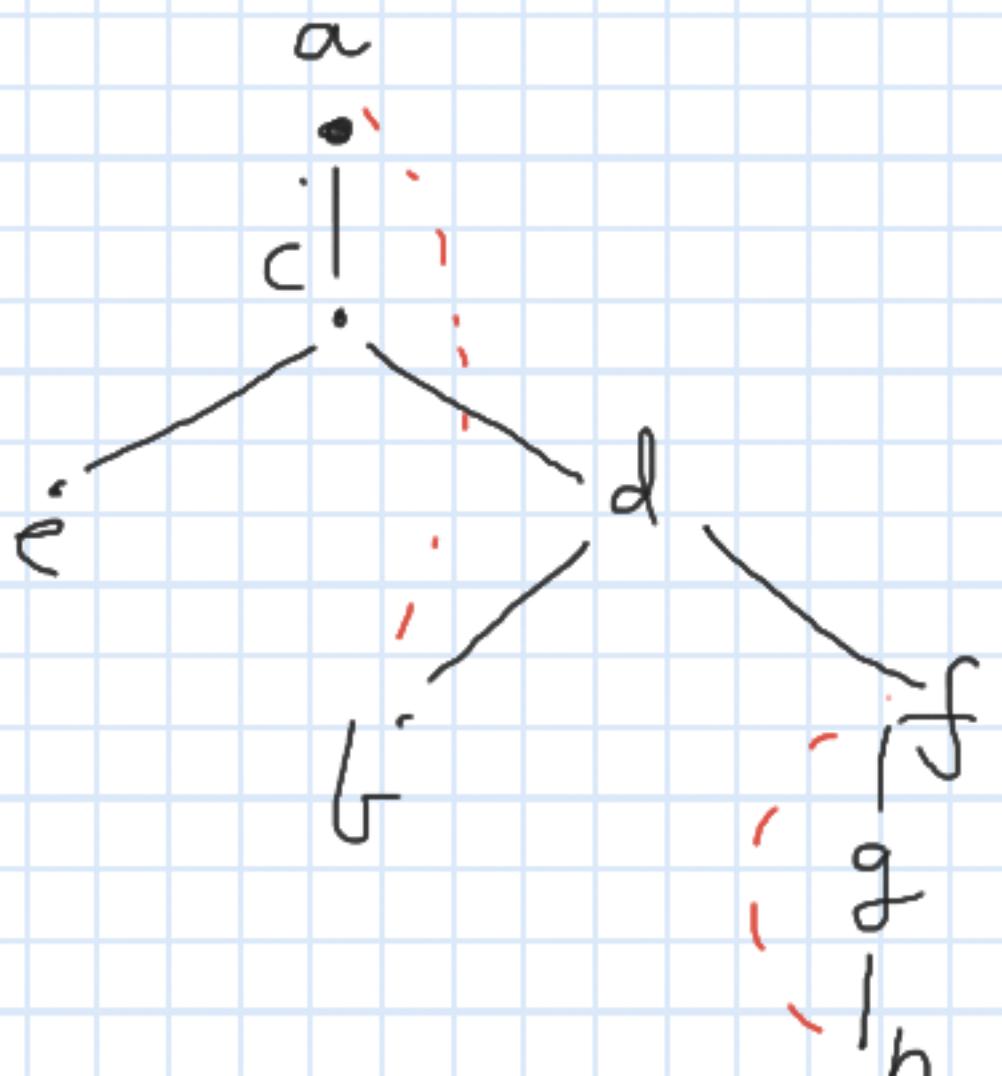
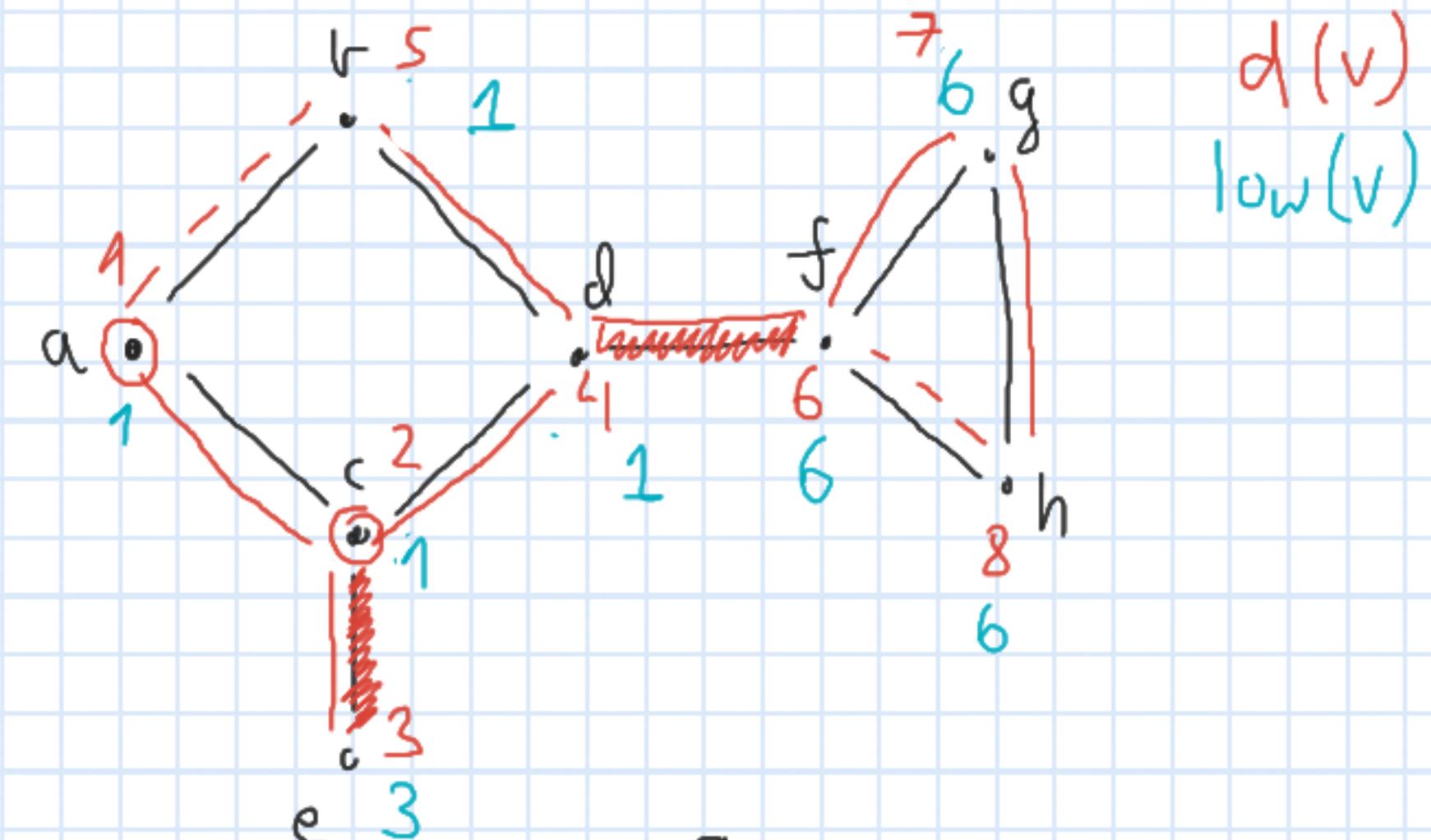
## Algorytm

① Wykonaj DFS, dla kaidego wierzchołka  $v$  zapamiętując czas odniedzienia  $\delta(v)$

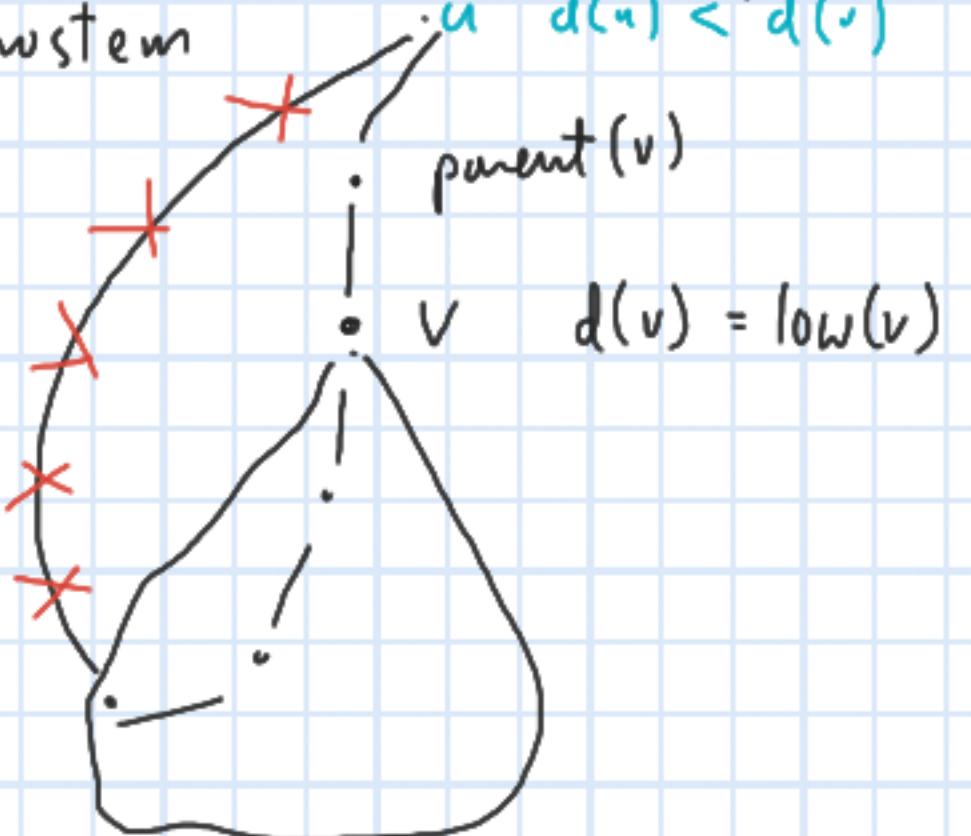
② Dla każdego wierzchołka  $v$  obliczamy

$$\text{low}(v) = \min \left( d(v), \min_{\substack{u - \text{istnie} \\ \text{krzyżdż ustenna} \\ z u do v}} d(u), \min_{\substack{w - \text{drzecko} \\ v \in \text{drzecie} \\ \text{DFS}}} \text{low}(w) \right)$$

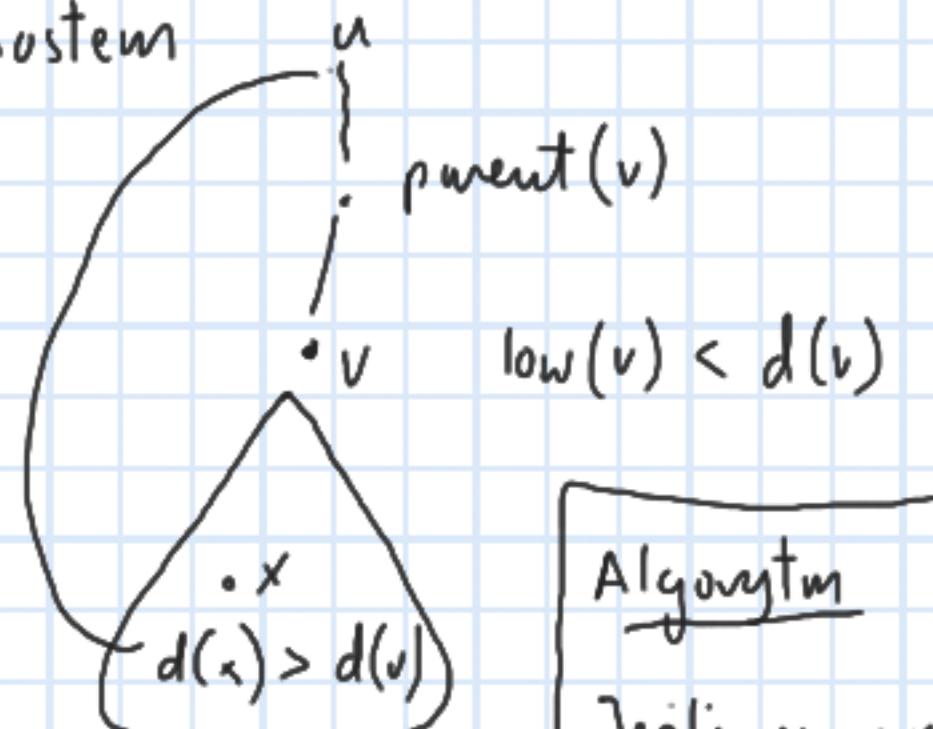
③ Mostly to range over  $\{v, \text{parent}(v)\}$  giving  $d(v) = \text{low}(v)$



Jesli  $d(v) = \text{low}(v)$  to  $\{v, \text{parent}(v)\}$   
jest mostem



Jesli  $d(v) \neq \text{low}(v)$  to  $\{v, \text{parent}(v)\}$   
nie jest mostem



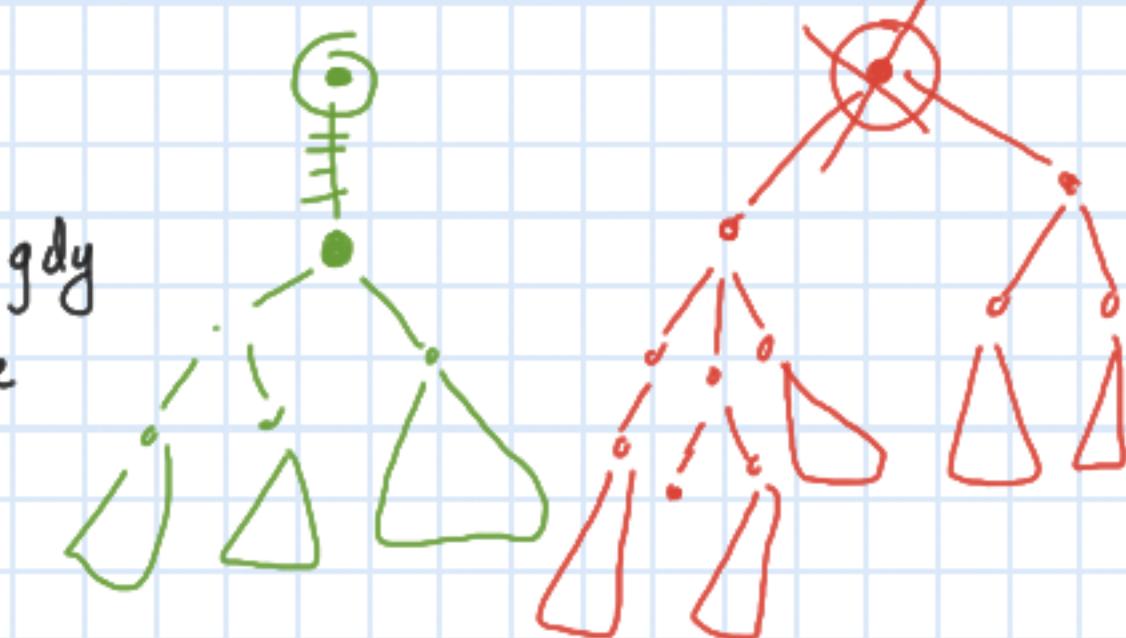
### Algorytm

Jesli  $v$  posiada syna  $s$  takiego, ze  $\text{low}(s) \geq d(v)$  to  
 $v$  jest punktem artykulacji (koniec osobnu)

def Punkt artykulacji nieskierowanego grafu  $G$  to taki wierzchołek  $v$ , którego usunięcie zwiększa liczbę spójnych składowych

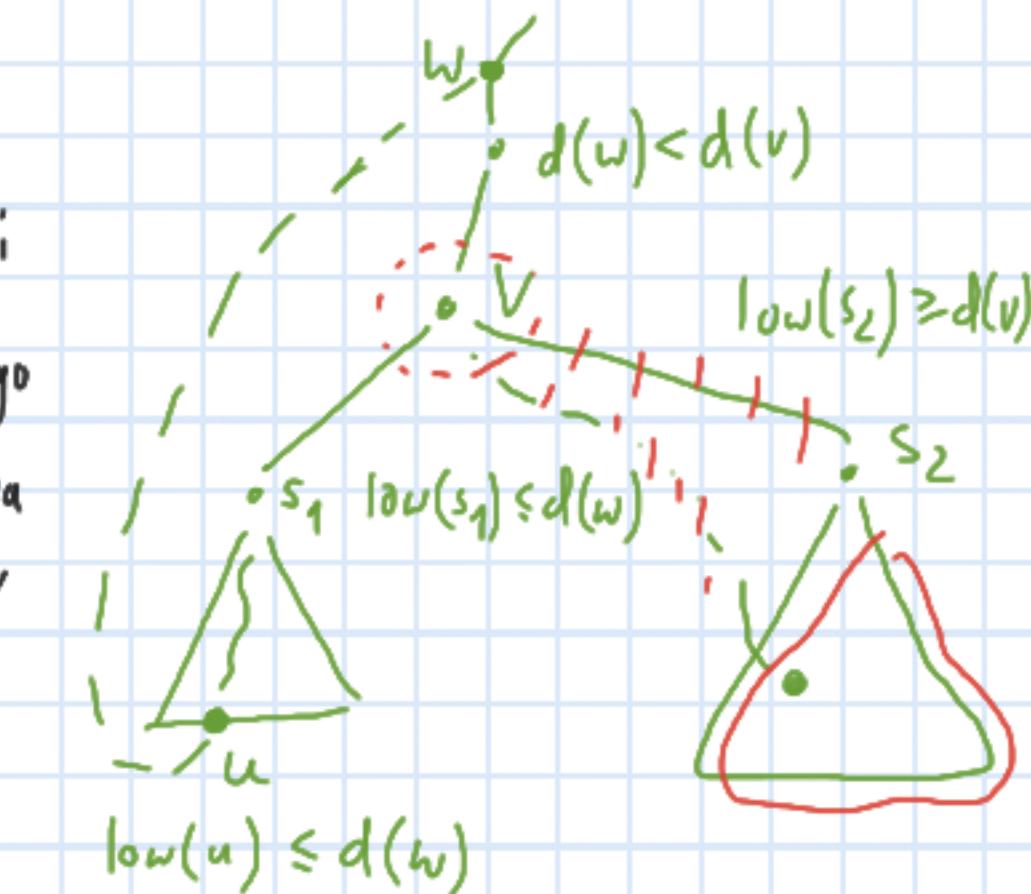
tw

Koniec drzewa DFS jest punktem artykulacji ktw. gdy posiada co najmniej dwoje dzieci w drzewie DFS



tw Wierzchołek nie będący

końcem jest punktem artykulacji ktw. gdy dla pary najmniej jednego syna nie istnieje krawędź łącząca  $\{v, w\}$ , taka że  $w$  to potomek  $v$  a  $w$  jest predkiem  $v$



## Algorytmy na grafach wazonych

- minimalne dno rozpinajce  
(MST - minimal spanning tree)
- najciętsze ścieżki

## Reprezentacja grafów wazonych

$$G = (V, E)$$

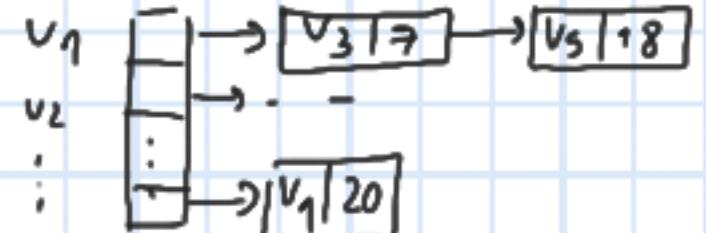
$$w: E \rightarrow \mathbb{N} \quad (\mathbb{Z}, \mathbb{Q}, \mathbb{Q}_+)$$

## Reprezentacja macienna

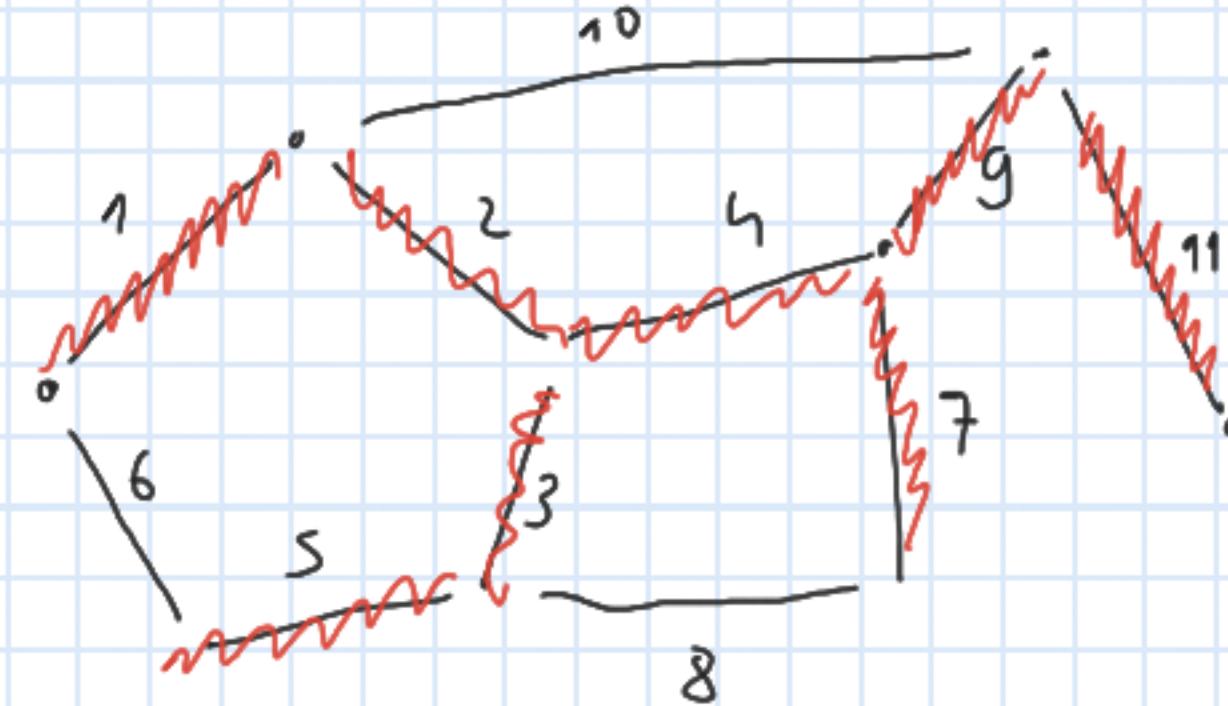
$w$  - matematyczna wag

$w[i][j]$  - waga krawędzi między  
wierzchołkami  $v_i$  oraz  $v_j$

## Reprezentacja listowa



## Minimalne dno rozpinające



MST to zbiór krawędzi, które łączą wszystkie  
wierzchołki (w spójny podgraf) i których  
suma wag jest minimalna

## Obserwacja

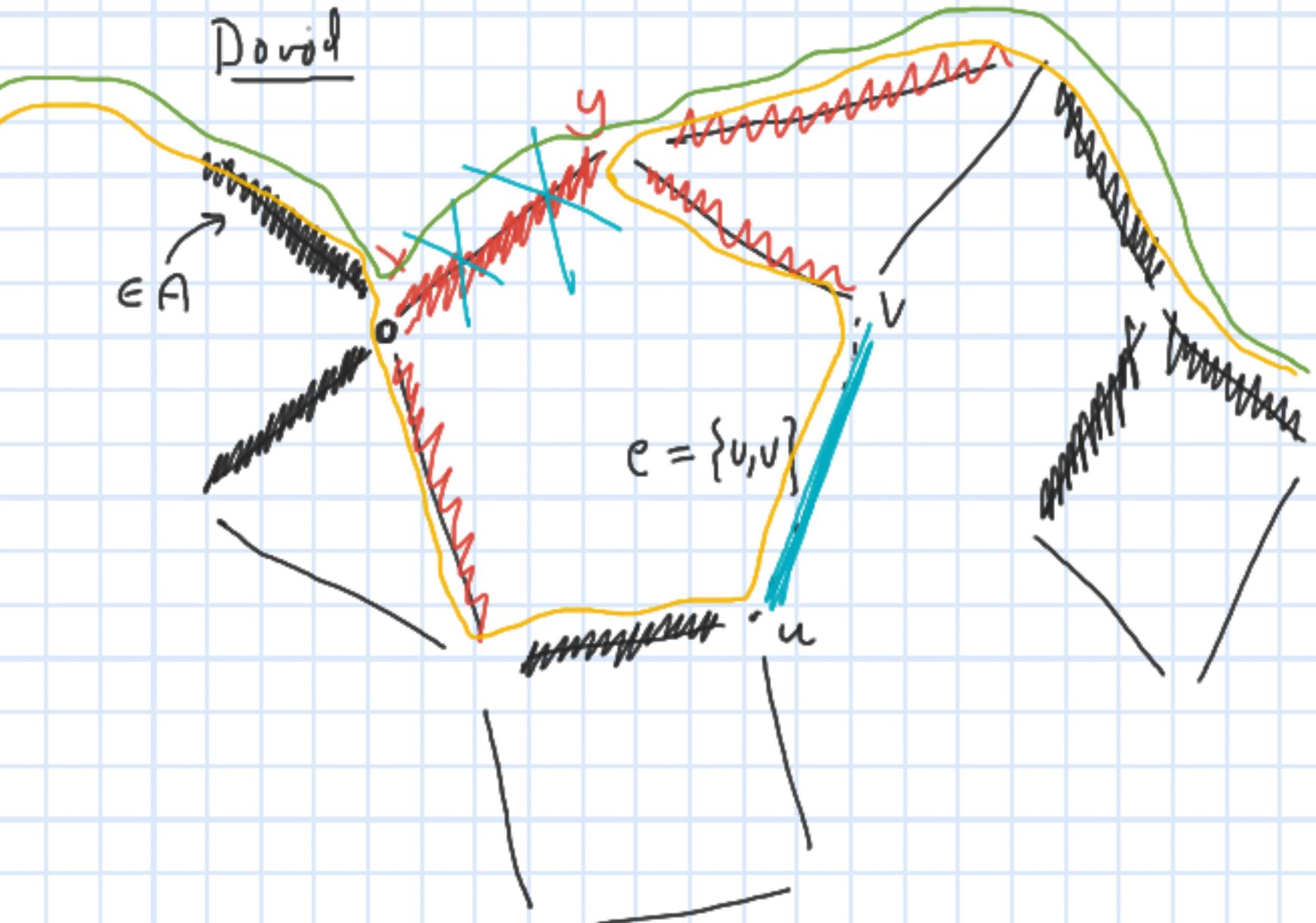
$$G = (V, E), \quad w: E \rightarrow \mathbb{N}$$

(II+)

Jesli  $A \subseteq E$  jest podzbiorem krawędzi  
pełnego MST dla  $G$  oraz  $e = \{u, v\}$  jest  
krawędzią, taką iż:

- a)  $e \notin A$
  - b)  $A \cup \{e\}$  nie tworzy cyklu
  - c) krawędź  $e$  ma minimalną wagę
- sposród krawędzi spełniających a) i b)

to  $A \cup \{e\}$  jest podzbiorem krawędzi  
pełnego MST dla  $G$



Jesli  $e = \{u, v\}$  nie należy do zadanego  
MST zaciszającego  $A$  to jaką inną krawędź  
 $\{x, u\} \notin A$  stwierzyć do zapewnienia  
trasy z  $u$  do  $v$

## Algorytm Kruskala znajdowania MST

① Posortuj krawędzie po wagach

②  $A = \emptyset$

③ Przeglądaj krawędzie w kolejności  
niesmalejącej wag

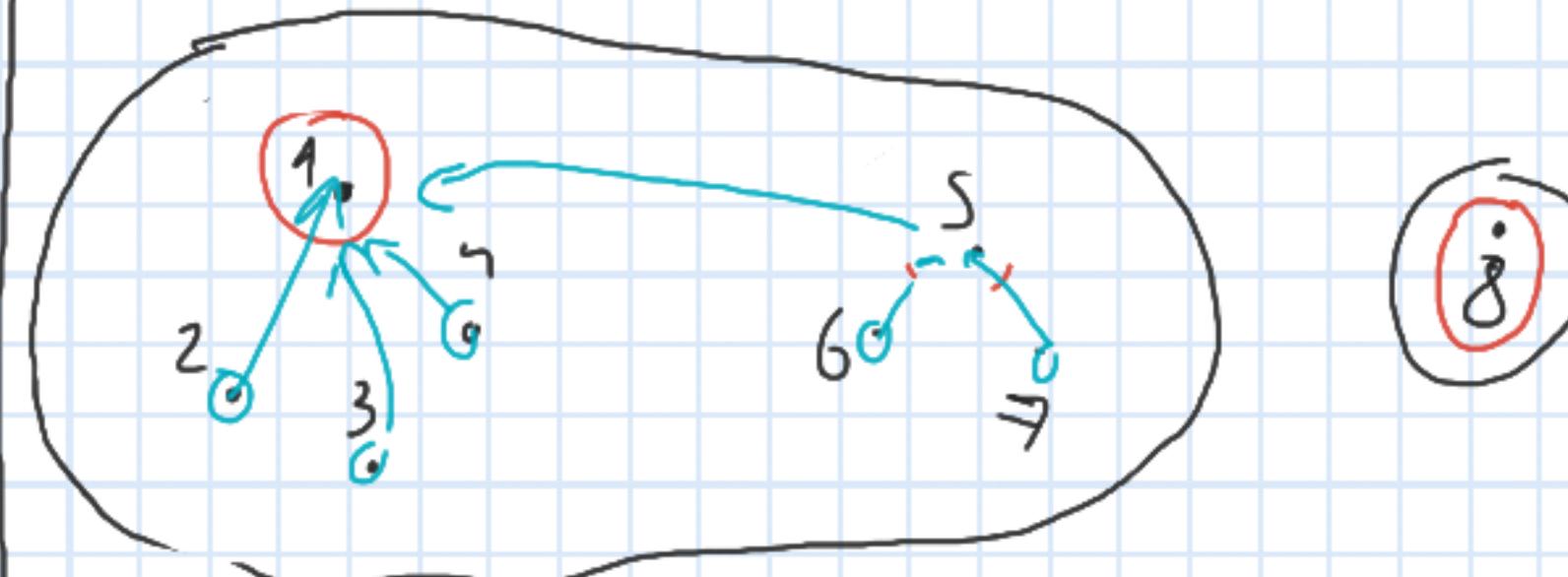
jeśli  $A \cup \{e\}$  nie zauważa

cyklu to  $A := A \cup \{e\}$

⑤ Zwróć A



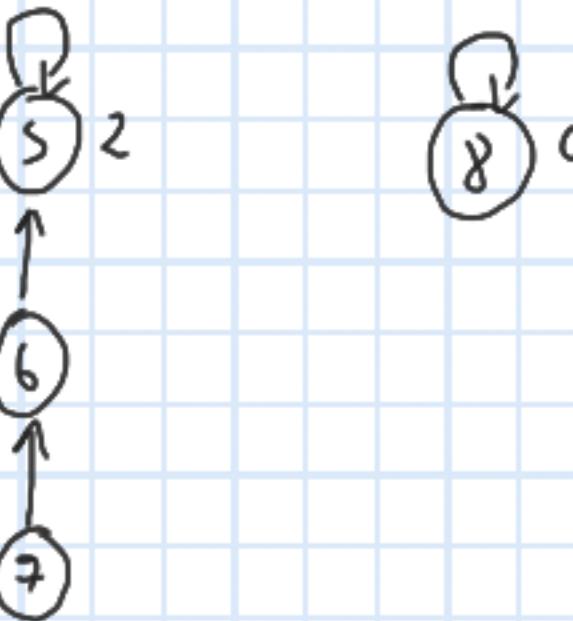
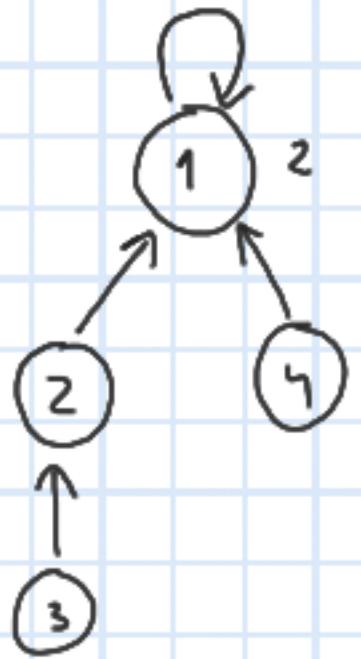
## Lasy zbiorów rozłącznych / Find-Union



### Trzy operacje

- makeSet - stwórz nowy zbiór jednoklentyony
- find - znajdź reprezentanta zbioru do którego należy dany element
- union - połącz dwa zbiorы

Приклад реалізації зв'язків взаємної



class Node:

```
def __init__(self, value):  
    self.parent = self  
    self.value = value  
    self.rank = 0
```

```
def find(x):
```

```
    if x.parent != x:  
        x.parent = find(x.parent)  
    return x.parent
```

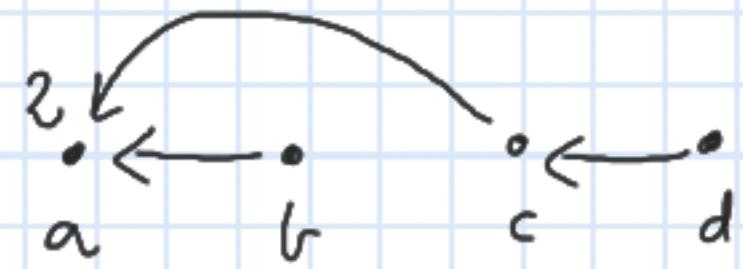
```
def union(x, y):
```

```
    x = find(x)  
    y = find(y)  
    if x == y: return
```

```
    if x.rank > y.rank:  
        y.parent = x
```

```
    else:  
        x.parent = y
```

```
        if x.rank == y.rank:  
            y.rank += 1
```



Obserwacja 1 Ranga to givne ograniczenie  
na wysokość drzewa

Obserwacja 2 Jeśli drzewo ma ranga  $n$  to  
zawiera co najmniej  $2^n$  węzłów

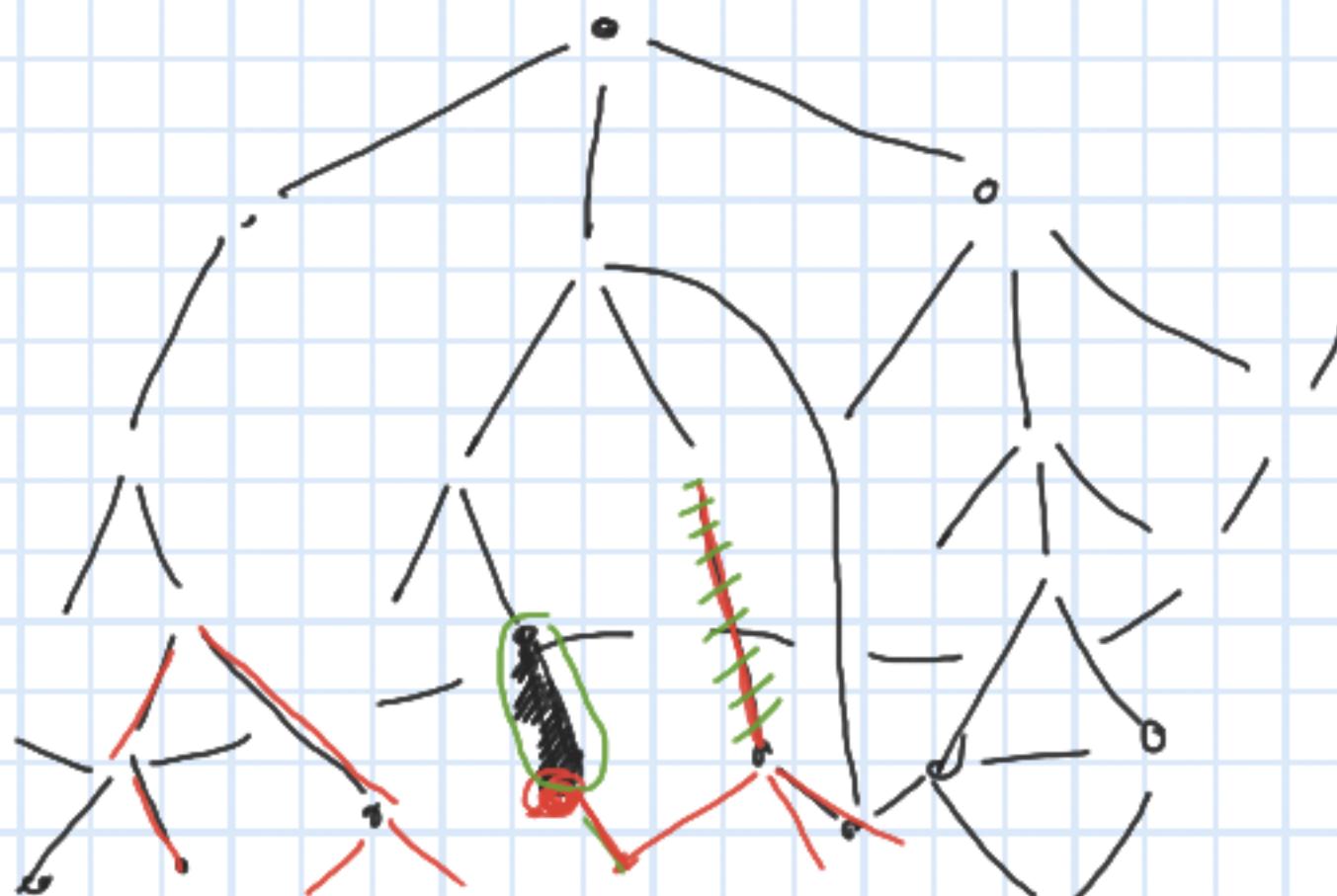
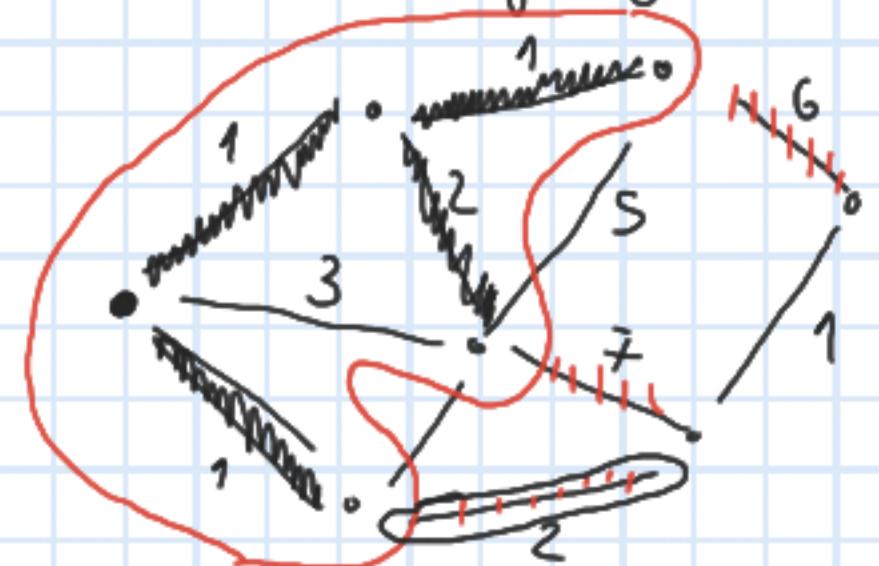
Obserwacja 3 Jeśli mamy  $T$  grupie  $n$  elementów  
to czas  $m$  operacji  $\text{find}$  i  $\text{union}$   
ma złożoność  $O(m \log n)$

Tw Czas  $m$  operacji  $\text{find}$  i  $\text{union}$ , gdy mamy  
 $T$  grupie  $n$  elementów, wynosi  $O(m \log^T n)$

# ASD - Wykład 12

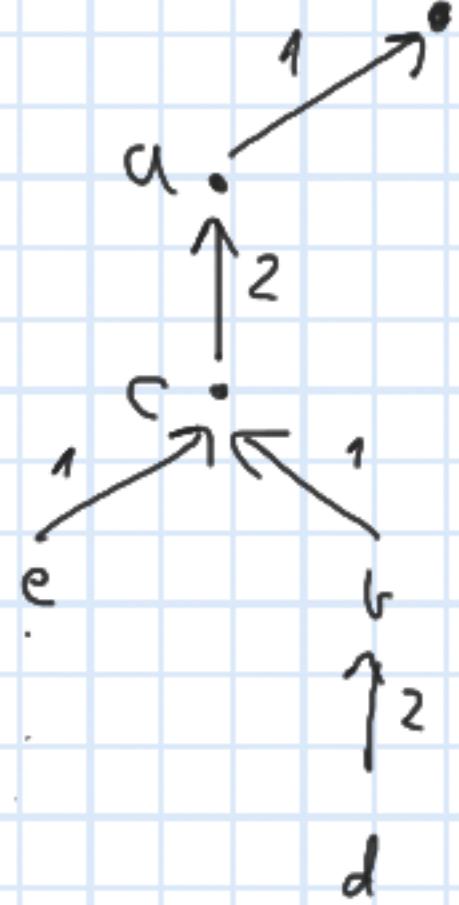
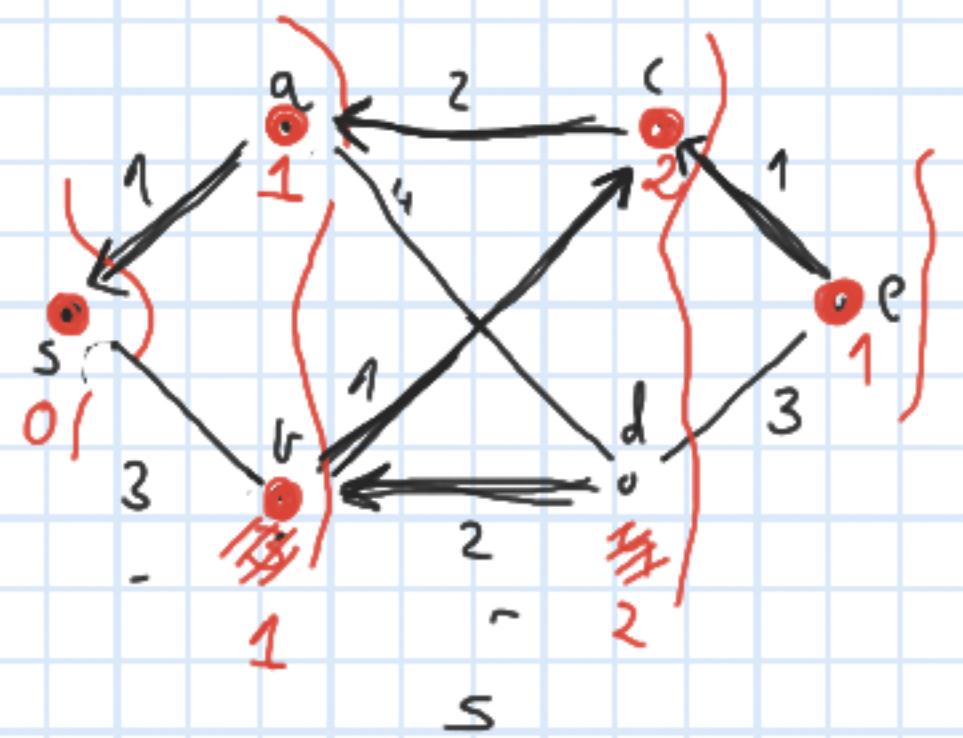
Algorytm Prima znajdowania minimalnego —> Opis algorytmu

druga wersja



1. Startujemy z wierzchołka  $s$
2. Wszystkie wierzchołki umieszczaemy w kolejce priorytetowej z wagą  $\infty$
3. Zamieniamy wagę  $s$  na  $0$
4. Póki są wierzchołki w kolejce:
  - ujmij wierzchołek  $t$  o minimalnej  
wadze
  - dla każdej krawędzi  $\{t,u\}$ ,  
jeśli waga  $u \geq w(t,u)$  to zamień  
wagę  $u$  na  $w(t,u)$  (i uaktualnij  
u.parent)

## Prywatny wykłaniam algorytmu Pima



## Problem znajdowania najkrótszych ścieżek

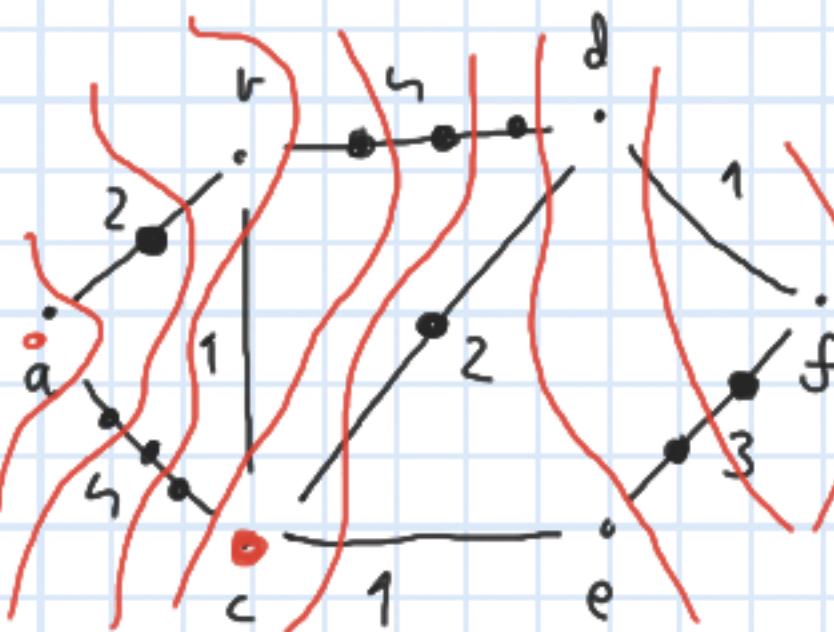
### Warianty

- 1 - 1
- 1 - wszyscy
- wszyscy - wszyscy

} na elementarnym poziomie trudne do wykonywania

} standardowe wersje problemu

### Podając elementarne



BFS na szczeblach  
rozmnожonych krogiach

Algorytm Dijkstry - algorytm elementarny, ale w każdym kroku skane od wierzchołka do prawdziwego wierzchołka

| Wagi nie muszą być naturalne, ale  
| muszą być nieujemne

Notacja

$G = (V, E)$ ,  $w: E \rightarrow \mathbb{R}_+$  - wersjony graf

$w(u,v)$  - waga krawędzi  $\{u,v\}$

$u.d$  - oszacowanie odległości z wierzchołka startowego najkrótsza

$u.parent$  - poprzednik na ścieżce (ze startowego do  $u$ )

Algorytm (startujemy z wierzchołka  $s$ )  
 $O(E \log V)$

① Umieść wszystkie wierzchołki w kolejce priorytetowej (min)  
z oszacowaniem odległości  $\infty$

② zmień odległość  $s$  na 0

③ podczas wierzchołków w kolejce:

- ujmij z kolejki wierzchołek  $u$  (o minimalnej wartości  $u.d$ )

- dla każdej krawędzi  $\{u,v\}$  wykonaj relaksację:

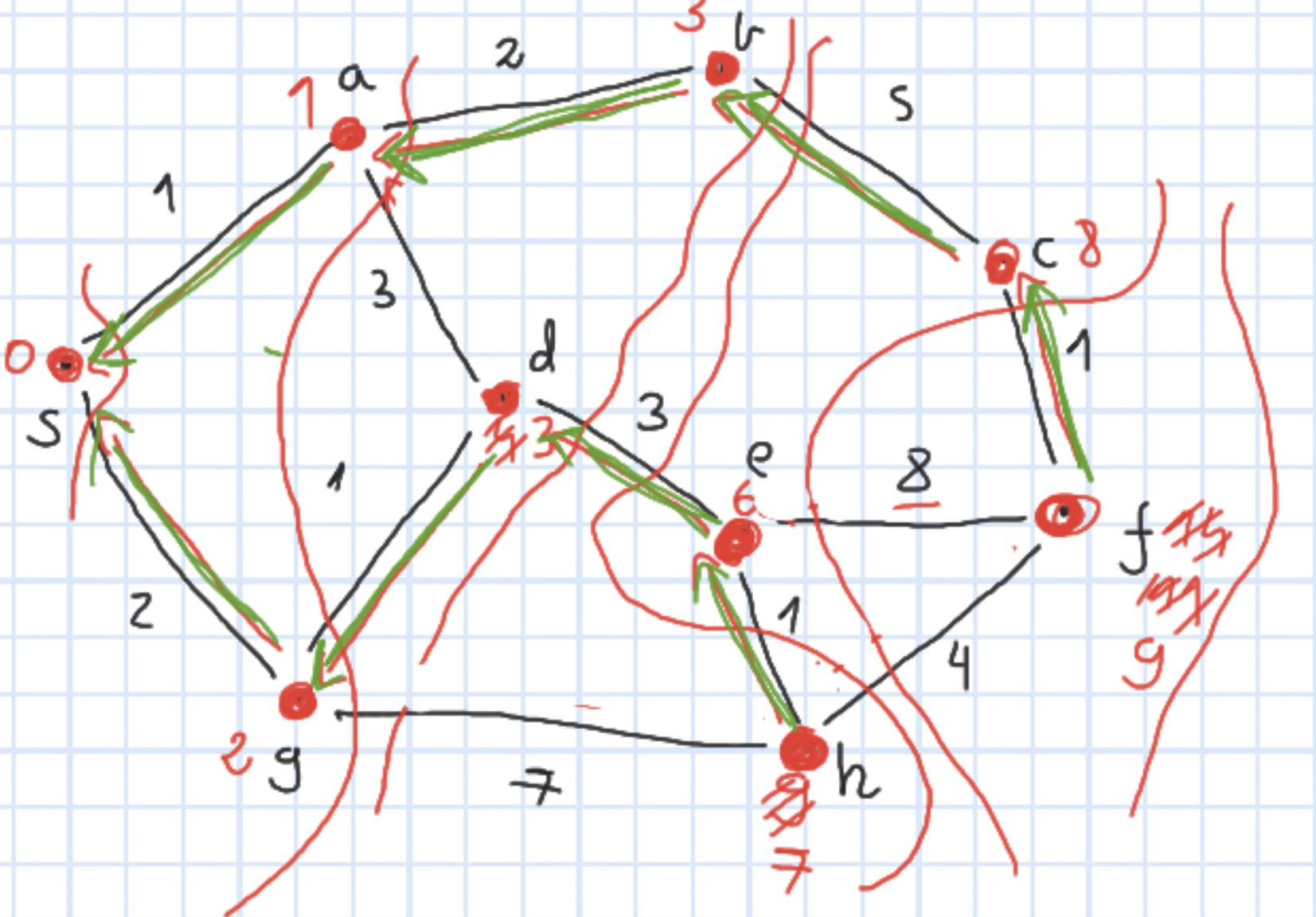
def relax( $u, v$ ):

if  $v.d > u.d + w(u,v)$ :

$v.d = u.d + w(u,v)$

$v.parent = u$

Punktad nykonomia algorytm Dijkstra

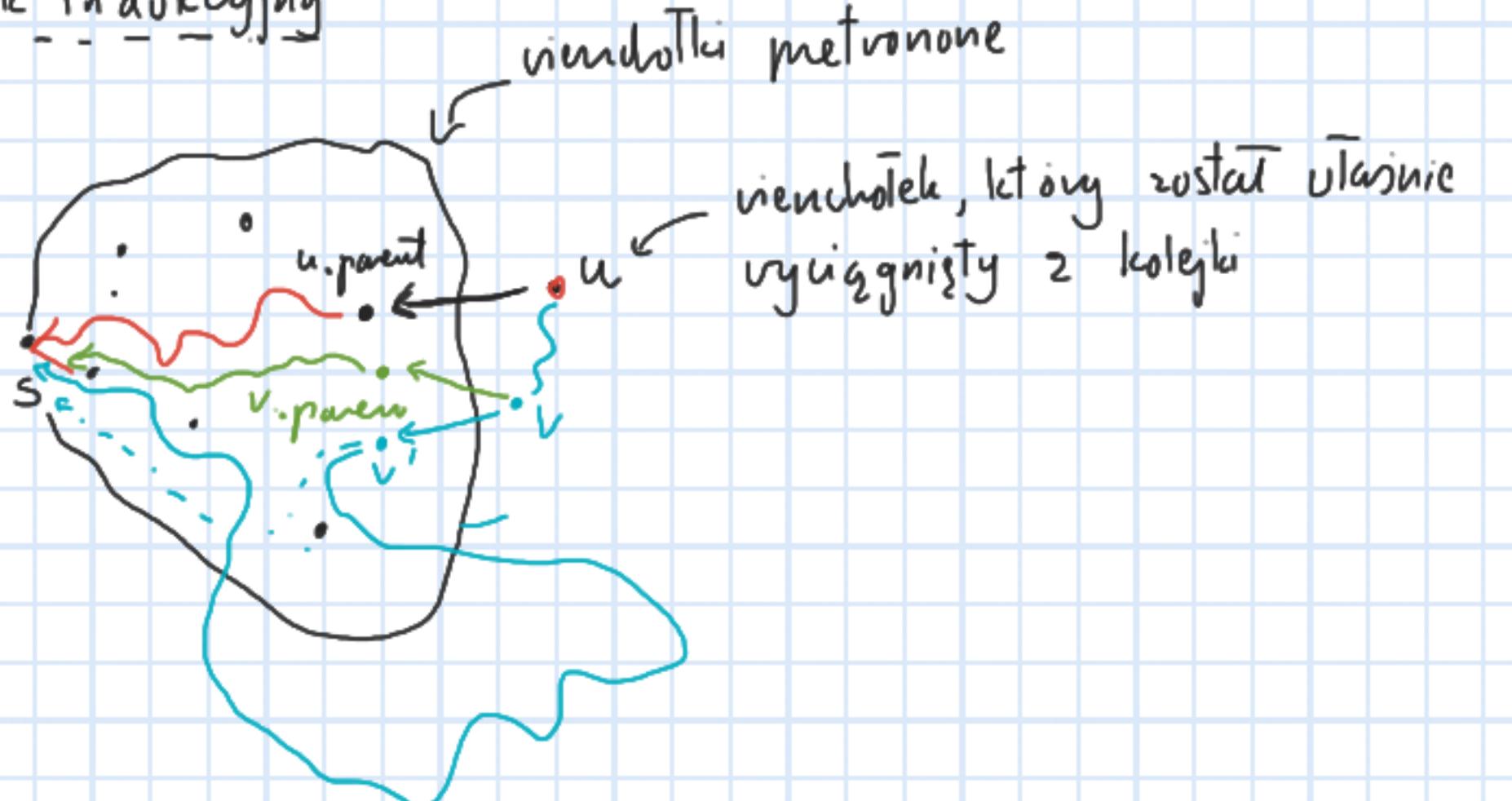


tw Gdy algorytm Dijkstry wyciąga  
wierzchołek u z kolejki, to jego pole u.d  
zawiera długość najkrótszej ścieżki z s do u

Dowód (przez indukcję)

dla wierzchołka startowego oznaczając zadanie

krok indukcyjny



# Algorytm Bellmana - Forda

- najkrótsze ścieżki jeśli wagi krawędzi ujemne

## ① Inicjalizacja

for  $v \in V$ :

$v.d = \infty$

$v.parent = \text{None}$

$s.d = 0$

## ② Relaksacja

for  $u \in \text{range}(|V| - 1)$ :

for  $(u, v) \in E$ :

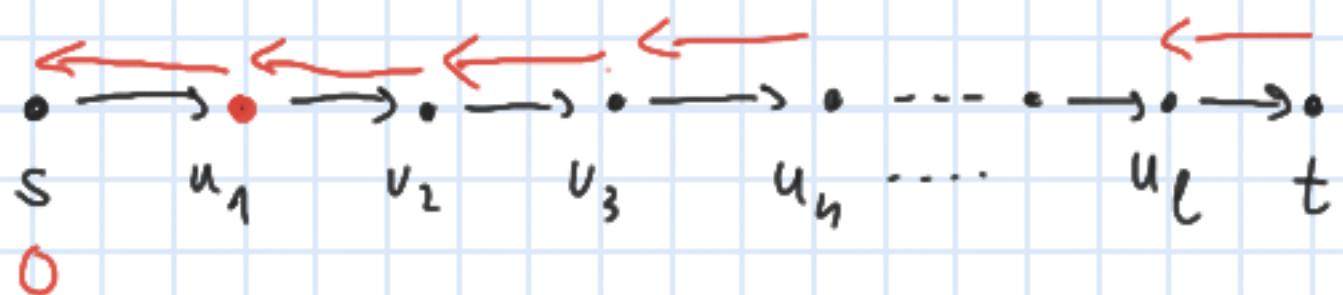
Relax( $u, v$ )

## ③ Uczyfikacja

wy dla kaidej  $(u, v) \in E$ :

$v.d \leq u.d + w(u, v)$ ?

Czy to działa?



Złożoność

$O(VE)$

# ASD - Wykład 13

Algorytm Bellmana-Forda (dla  $G = (V, E)$ )

$$w : E \rightarrow \mathbb{R}$$

## ① Inicjalizacja

for  $v \in V$

$$v.d = \infty$$

$v.parent = \text{None}$

$$s.d = 0$$

## ② Relaksacja

for  $i$  in range ( $|V| - 1$ )

for  $(u, v) \in E$ :

Relax ( $u, v$ )

## ③ Weryfikacja

Czy dla każdej  $(u, v) \in E$ :

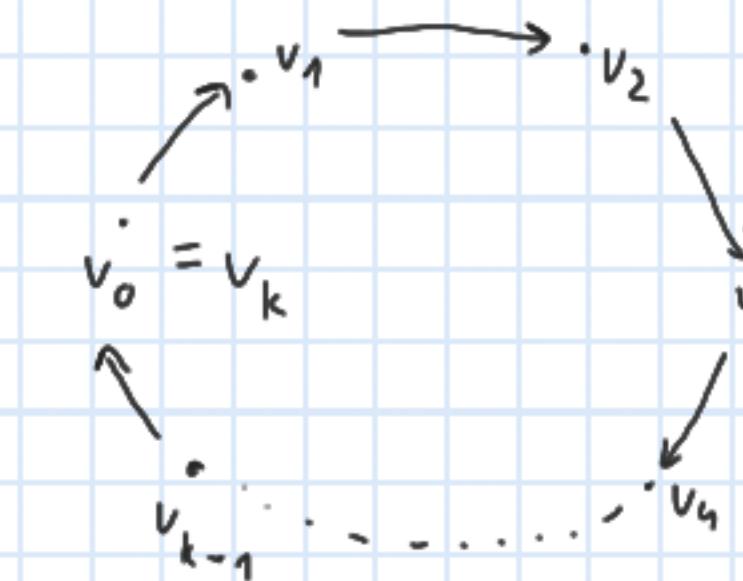
$$v.d \leq u.d + w(u, v)?$$

Jesli tak - OK

Jesli nie - ujemny cykl

Zauważmy, że  $G$  posiada pierścieniowy cykl o ujemnej wartości, osiągającą ze

żródła  $s$



$$\sum_{i=0}^{k-1} w(v_i, v_{i+1}) < 0$$

symetryczny

Jesli w taki sytuacji weryfikacja by powiedziała, to:

$$\forall v_i : v_{i+1}.d \leq v_i.d + w(v_i, v_{i+1})$$

$\downarrow$

$(v_i, v_{i+1})$

Sumując te nierówności:

$$\sum_{i=0}^{k-1} v_{i+1}.d \leq \sum_{i=0}^{k-1} (v_i.d + w(v_i, v_{i+1}))$$

$$\Rightarrow 0 \leq \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

Najkrótsze ścieżki między każdą parą wierzchołków

-  $|V|$  wersja algorytmu Dijkstry

$$O(V \cdot E \log V)$$

-  $|V|$  wersja algorytmu Bellmana - Forda

$$O(V^2 E)$$

### Konwencja

Stosujemy reprezentację macienną

Stosujemy macierze najkrótszych odległości

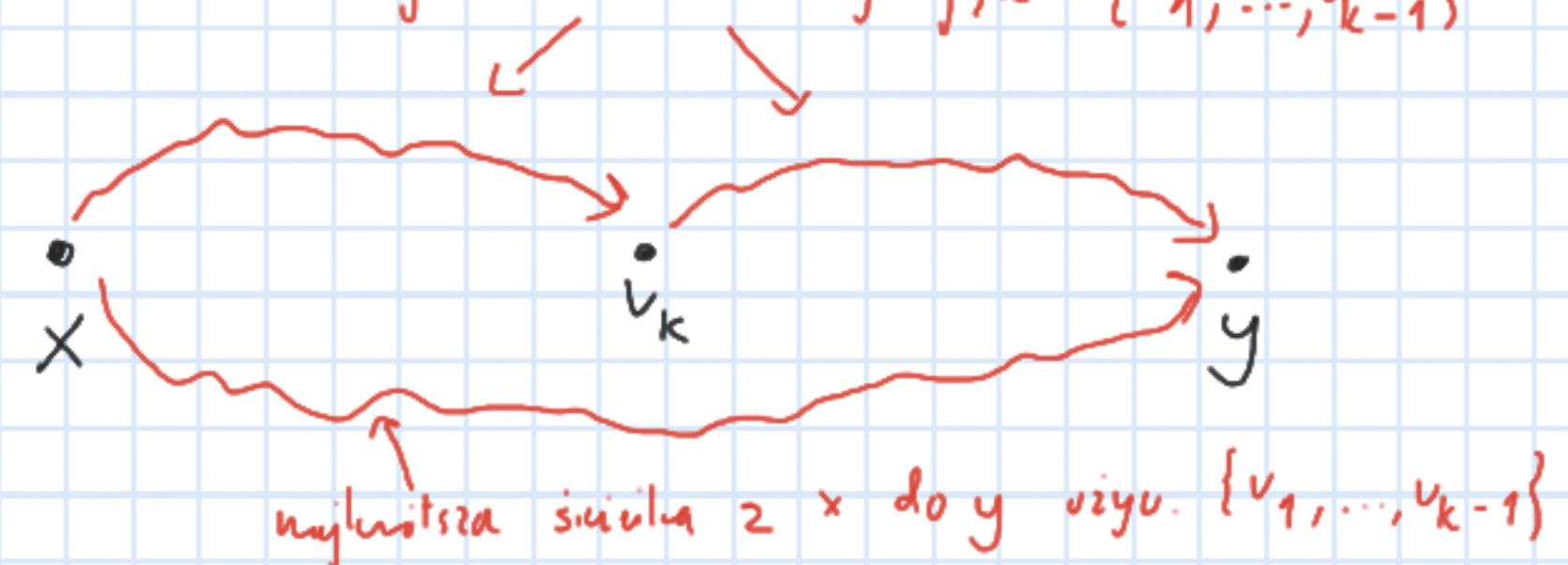
$D[u][v]$  - dł. najkrótszej ścieżki z  $u$  do  $v$

### Algorytm Floyda - Warshalla

$$G = (V, E), w: V \times V \rightarrow \mathbb{R}, V = \{v_1, \dots, v_n\}$$

Idea: Jeśli dla pewnego  $k$  znamy najkrótsze ścieżki między każdą parą wierzchołków, ale ograniczone do wierzchołków wewnętrznych  $\{v_1, \dots, v_{k-1}\}$  to mamy Tato rok obliczać najkrótsze ścieżki z wierzchołkami wewnętrznymi  $\{v_1, \dots, v_k\}$

najkrótsze ścieżki wierzchołki  $\{v_1, \dots, v_{k-1}\}$



# Implementacja algorytmu Floyda - Warshalla

$D^{(t)}[u][v]$  - długość najkrótszej ścieżki z  
u do v, jeśli można po drodze  
koniecznie z wierzchołkami  
 $\{v_1, \dots, v_t\}$

$$D^{(0)} = D$$

2 fazowe

$$\mathcal{O}(V^3)$$

for  $k$  in range ( $1, n+1$ ):

    for  $u \in V$ :

        for  $v \in V$ :

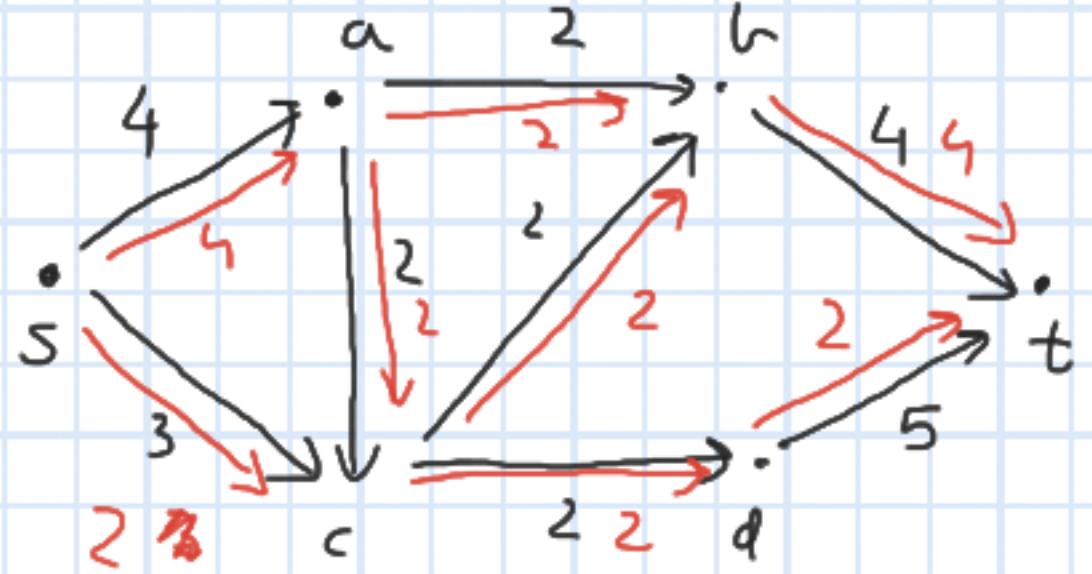
$$D^{(k)}[u][v] = \min \left( D^{(k-1)}[u][v], D^{(k-1)}[u][v_k] + D^{(k-1)}[v_k][v] \right)$$

poniższy u implementacj.

return  $D^{(n)}$

ASD - Wykład 13b

Problem maksymalnego przepływu



Ujęcie:  $G = (V, E)$  - graf skierowany  
(krzędzie tylko jednostronny)

$c: V \times V \rightarrow \mathbb{N}$  ← pojemność krzędzi

Jesli  $(u, v) \notin E$  to  $c(u, v) = 0$

$s$  - źródło, nie ma krzędzi wychodzących

$t$  - ujście, nie ma krzędzi wychodzących

Zadanie

Znaleźć "przepływ" f o maksymalnej wartości

to funkcja

$$f: V \times V \rightarrow \mathbb{N}$$

$$(\forall u, v) [f(u, v) \leq c(u, v)]$$

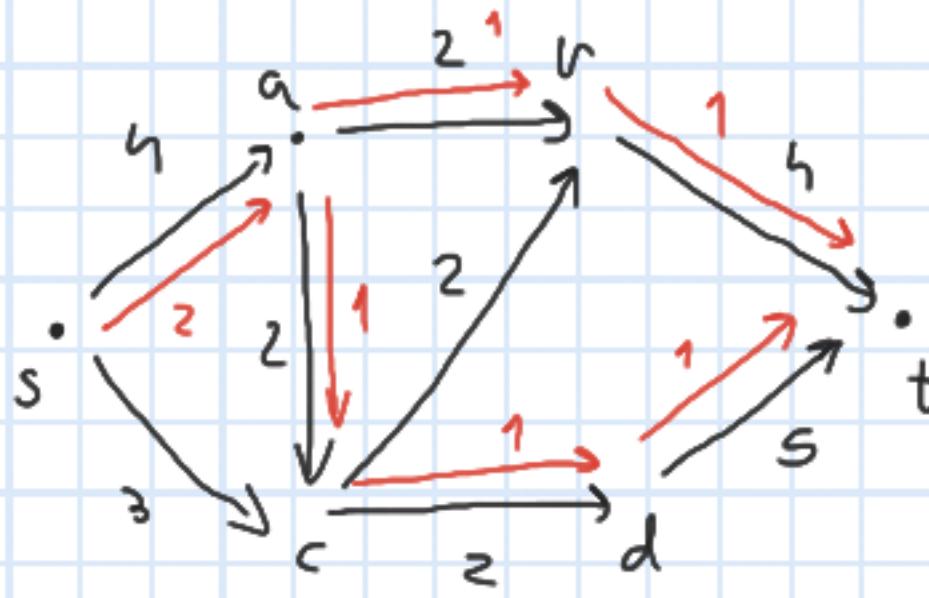
$$(\forall v \in V - \{s, t\}) \left[ \sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u) \right]$$

$$|f| = \sum_{v \in V} f(s, v) - \underbrace{\sum_{v \in V} f(v, s)}_{=0}$$

## Sieci reszidualne

$$\left. \begin{array}{l} G = (V, E) \\ s, t \in V \\ c: V \times V \rightarrow \mathbb{N} \\ f: V \times V \rightarrow \mathbb{N} \end{array} \right\}$$

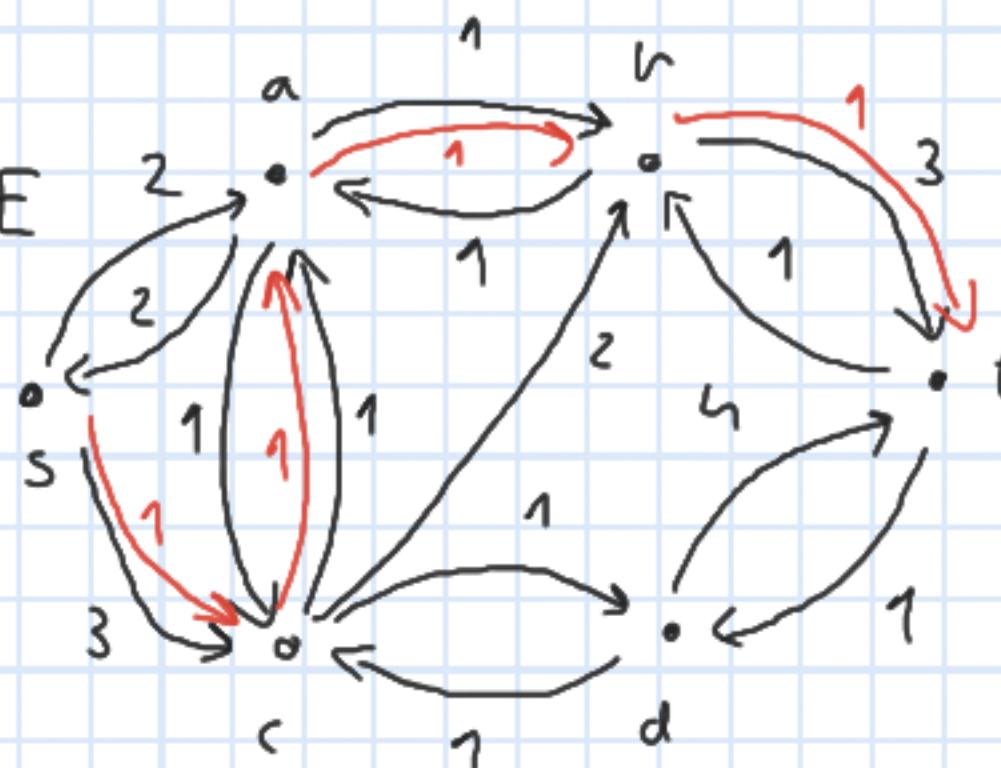
sieci przepływu



Definiujemy sieci residualne  $G_f, c_f$

przez funkcje  $c_f$ :

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v), & (u,v) \in E \\ f(v,u), & (v,u) \in E \\ 0 & \text{inaczej} \end{cases}$$



Metoda Forda - Fulkersona

- ① Jeśli istnieje siecika powiększająca dla  $G_f, c_f$ , to ponownie zgodnie z nią przepływy
- ② wróć do kroku ①

Sieciaka powiększającego dla  $G_f$  to sieciaka z s do t w  $G_f, c_f$

Wantosując tej sieciki jest największa wartość  $c_f(u,v)$  na sieci

Pnekovj u sieci

$$G = (V, E), s, t \left. \begin{array}{l} \\ \end{array} \right\} \text{sieci pretpostavka}$$

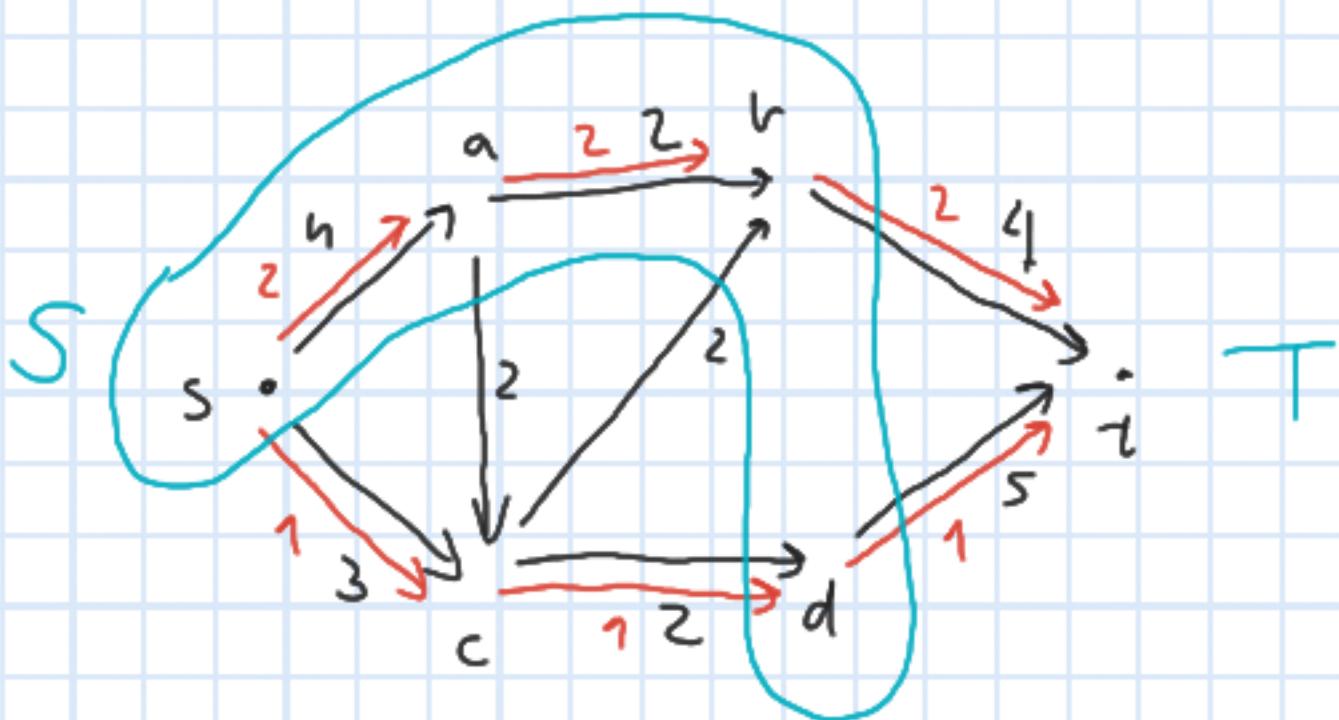
$$c: V \times V \rightarrow \mathbb{N}$$

$$f: V \times V \rightarrow \mathbb{N}$$

Pnekovj sieci to podzbiot  $V$  na

$$S, V - S = T$$

$$\text{gdzie } s \in S, t \in T$$



$$c(S, T) = 3 + 2 + 4 + 5 = 14$$

$$f(S, T) = 4 - 1 = 3$$

Pnepustovici pnekova S, T to:

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$$

Pneplyw netto:

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$

Lemat

$$f(S, T) = |f|$$

Dowód

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

$= 0$  (utrzymać zachowania mocy)

$$+ \left[ \sum_{u \in S - \{s\}} \sum_{v \in V} f(u, v) - \sum_{u \in S - \{s\}} \sum_{v \in V} f(v, u) \right]$$

$$= \sum_{u \in S} \sum_{v \in V} f(u, v) - \sum_{u \in S} \sum_{v \in V} f(v, u)$$

$$= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$

$$+ \left( \sum_{u \in S} \sum_{v \in S} f(u, v) - \sum_{u \in S} \sum_{v \in S} f(v, u) \right) = 0$$

$$= f(S, T)$$

Lemat

$$|f| \leq c(S, T)$$

Dowód

$$|f| = f(S, T)$$

$$= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$

$$\leq \sum_{u \in S} \sum_{v \in T} f(u, v)$$

$$\leq \sum_{u \in S} \sum_{v \in T} c(u, v) = c(S, T)$$

tu (max-flow / min-cut theorem)

Niech  $G = (V, E)$ , s,t,  $c: V \times V \rightarrow \mathbb{N}$

łedzie siecią przepływu ovvz f niech będzie

przepływem w tej sieci. Następujące

warunki są równoważne:

① f jest maksymalnym przepływem w G

②  $G_f, c_f$  nie ma sieci powiększającej

③ Dla pewnego podkroju  $S, \bar{T}$  zachodzi

$$|f| = c(S, \bar{T})$$

Dowód

$$\textcircled{3} \Rightarrow \textcircled{1}$$

$$\textcircled{1} \Rightarrow \textcircled{2}$$

$$\textcircled{2} \Rightarrow \textcircled{3}$$

$$S = \left\{ v \in V \mid \begin{array}{l} \text{istnieje ścieżka z } s \text{ do } v \\ \text{w } G_f, c_f \end{array} \right\}$$

$$T = V - S$$

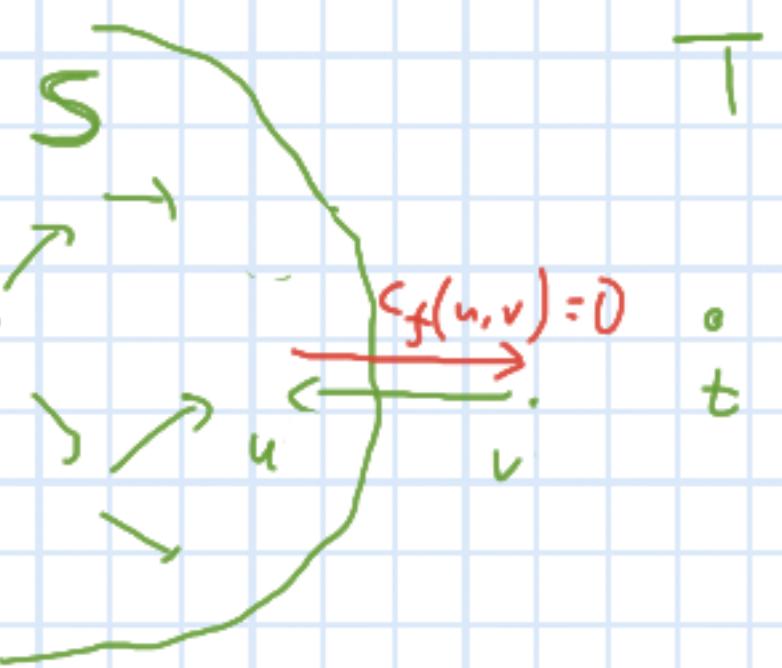
$$\text{zauważamy: } s \in S, \quad t \in T$$

innej istniejący ścieżka  
połączająca  
 $s \in S, t \in T$

$$= c(u, v)$$

$$= 0$$

$$|f| = f(S, \bar{T}) = \sum_{u \in S} \sum_{v \in \bar{T}} (f(u, v) - f(v, u)) = c(S, \bar{T})$$



Jeli  $(u, v) \in E$  to  $c_f(u, v) = 0$ , czyli  $f(u, v) = c(u, v)$

Jeli  $(v, u) \in E$  to  $c_f(u, v) = 0$ , myli  $f(v, u) = 0$

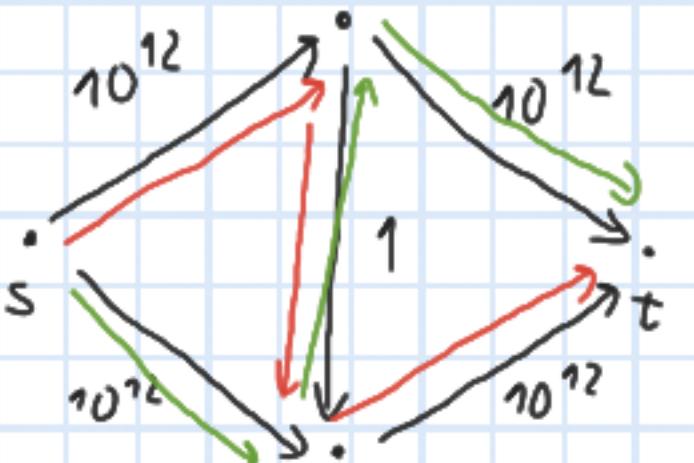
Jeli  $(u, v) \notin E$  i  $(v, u) \notin E$  to  $f(u, v) = 0 = c(u, v)$   
 $f(v, u) = 0$

## Złożoność metody Forda-Fulkersona

$$O((V+E) |f^*|) = O(E|f^*|)$$

zakładamy, że  
 $|E| > |V|$   
 wartości maksymalnego przepływu

To może być bardzo dwo!



Czas działania moja istotnie przyspieszać uzywając suwego powiskrącego algorytmu BFS

algorytm Edmonda-Karpia

$$O(VE^2)$$

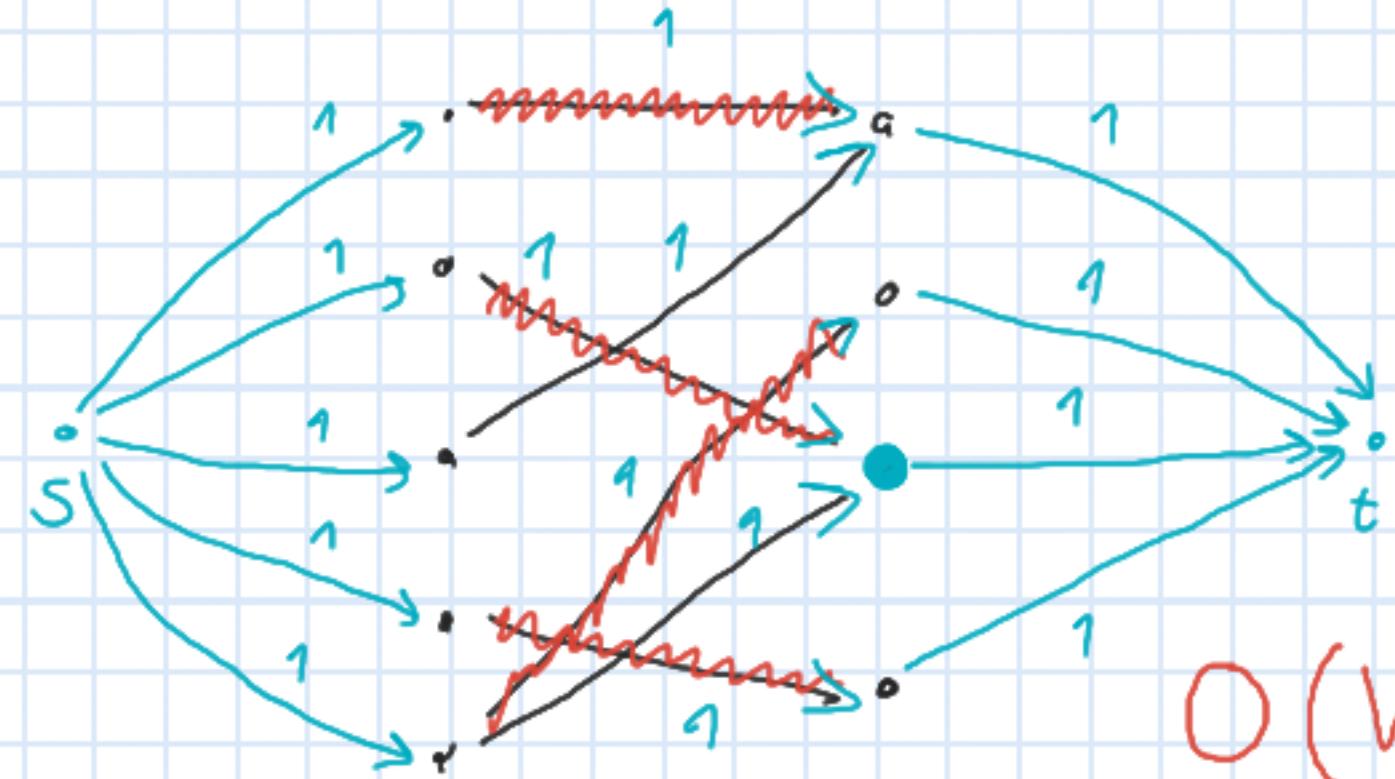
## Kilka wariantów problemu

① krawędzie w obie strony

② wiele źródeł i wiele wjścia



③ maksymalne skojarzenie w grafie dwudzielnym



# ASD - Wykład 14

## Tablice asocjacyjne

T - tab. asoc. (słownik, mapa)

$$T["\text{Stan}"] = 17$$

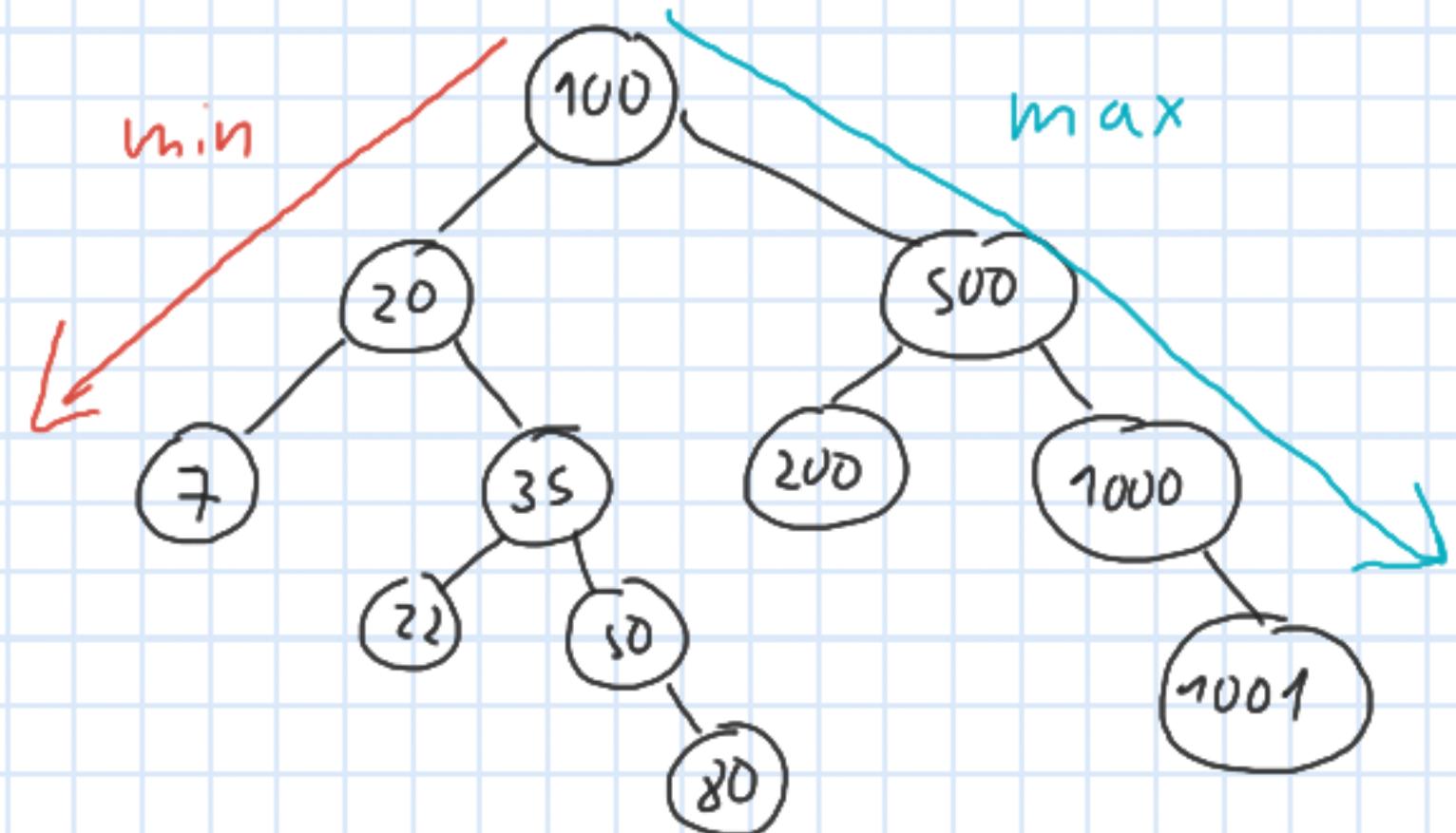
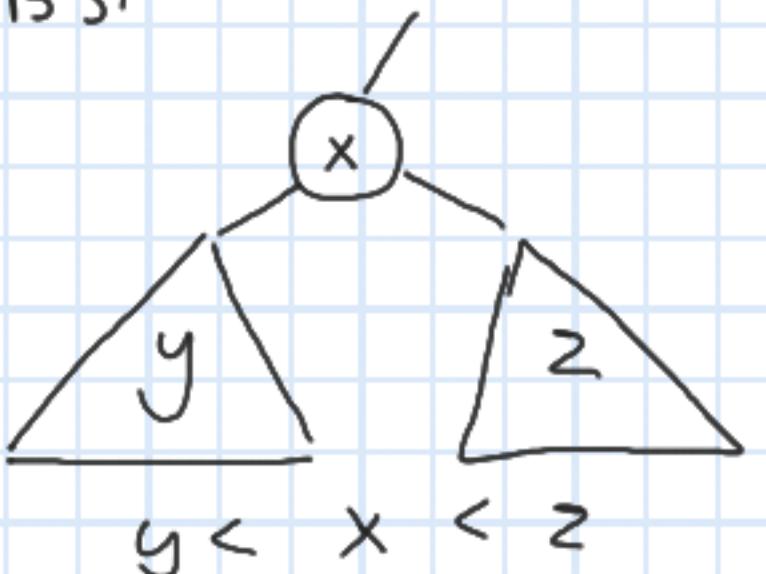
$$T[107] = 35$$

Tablice asocjacyjne, w których indeksy

mają określone relacje porządkowe

↳ implementacja przez dwa wykorzystanie  
binarnego (binary-search tree, BST)

Dwie BST



class BSTNode:

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

self.key = key

self.parent = None

self.left = None

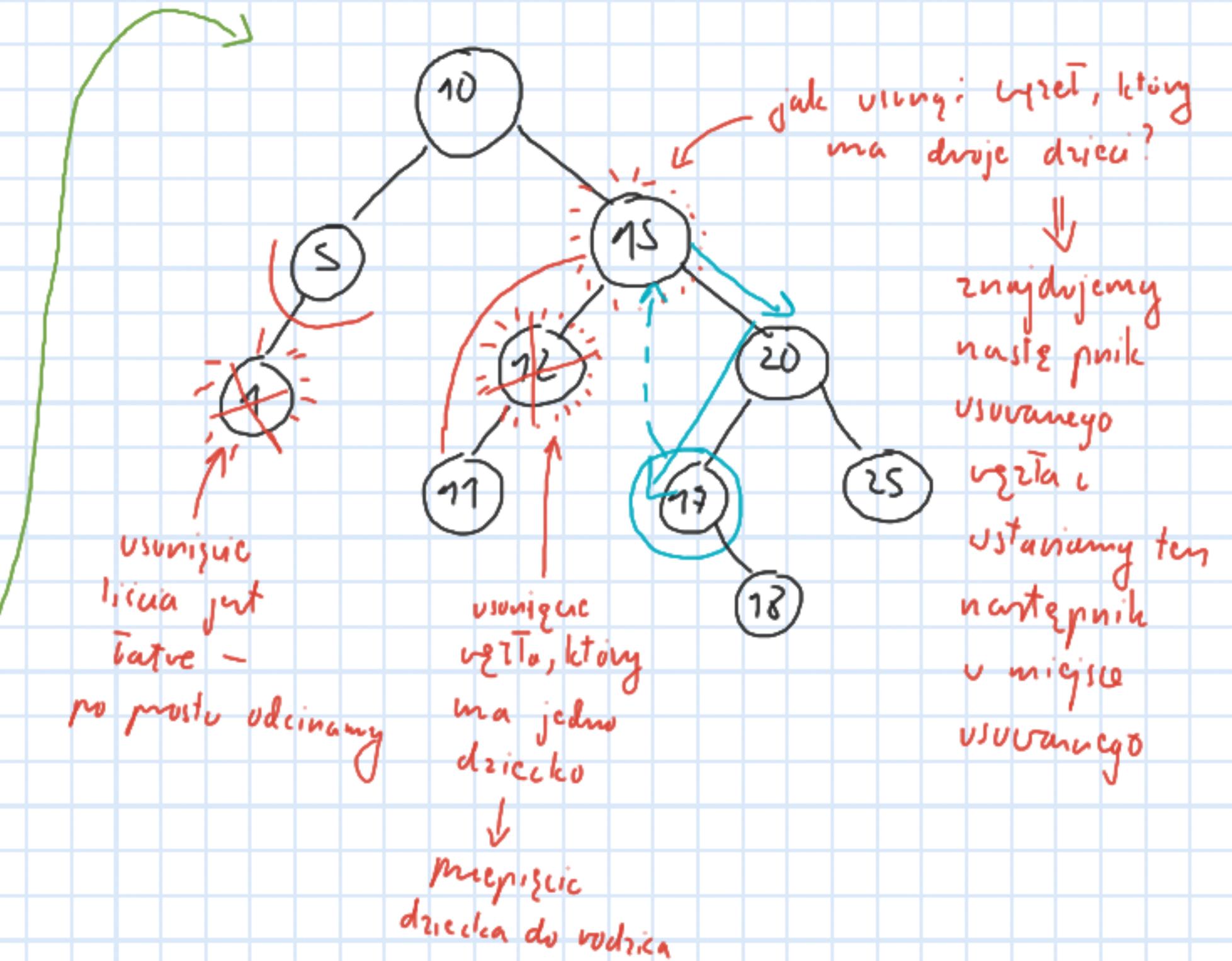
self.right = None

## Uyszukiwaniie w dneic BST

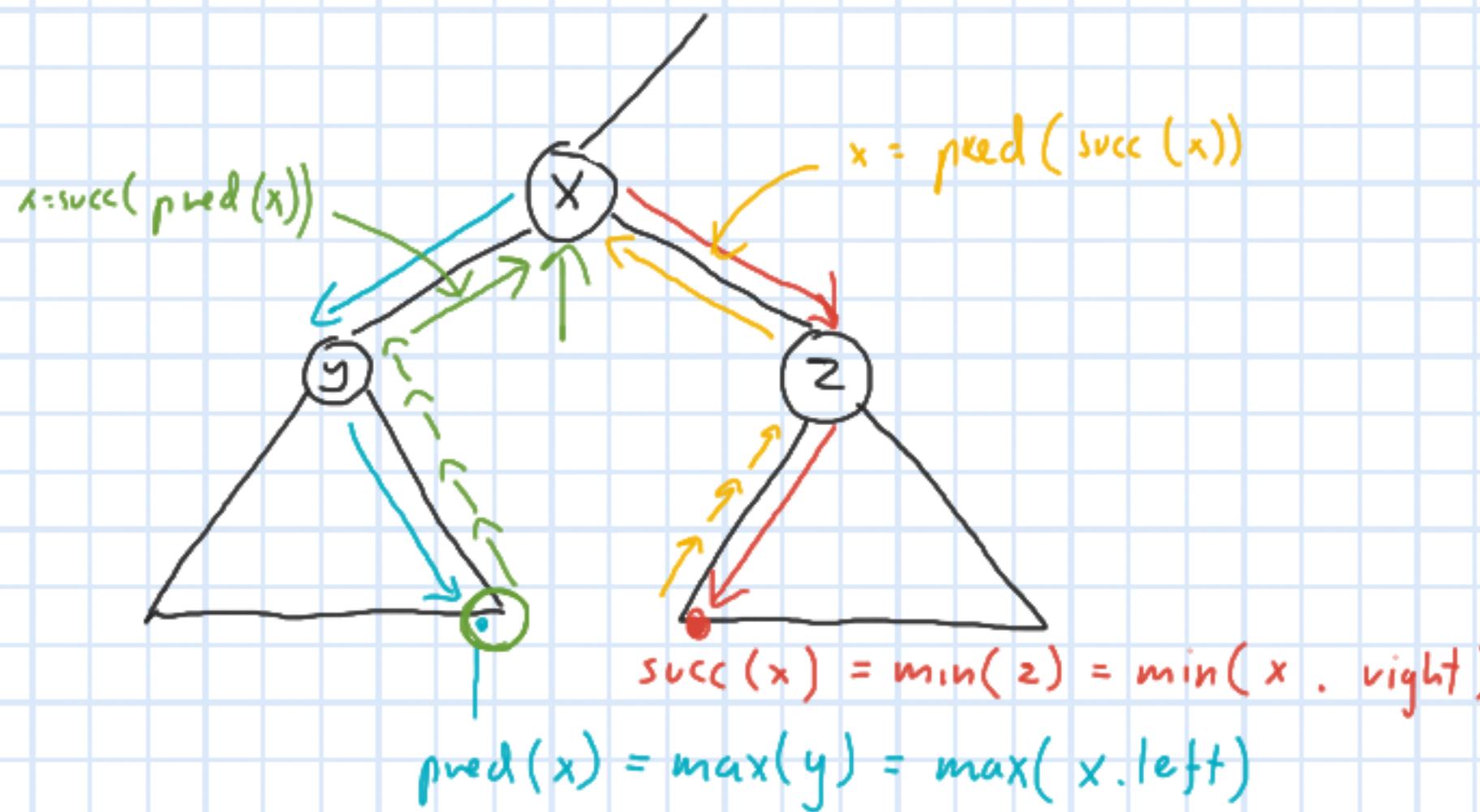
```
def find( root , key ) :  
    while root != None :  
        if root.key == key : return root  
        elif key < root.key : root = root.left  
        else : root = root.right  
    return None
```

## Inne operasye

- ustanianie
  - min / max
  - usuwanie
  - następnik (succ), poprzednik (pred)



Następnik i poprzednik w drzewie BST

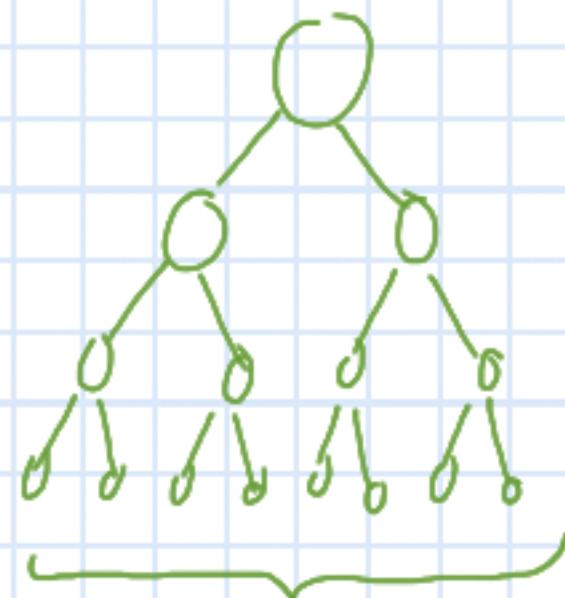


Złożoność operacji na drzewie BST to

$O(h)$



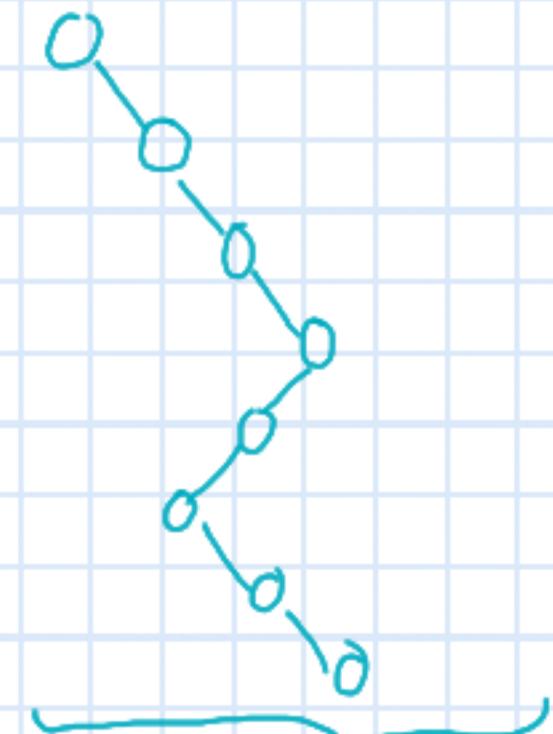
wysokość drzewa



idealne rozmiarzenie  
drzewo

$n - \sqrt{2} \cdot 2^h$

$$h = O(\log n)$$



idealne nierozmiarzenie  
drzewo

$n - \sqrt{2} \cdot 2^h$

$$h = O(n)$$

## Drewa neronow - warne

Drewa BST, które dodatkowo spełniają następujące warunki:

- każdy węzeł jest czerwony lub czarny
- każdy liść (None) jest czarny
- jeśli węzeł jest czerwony to obaj synowie są brązowe
- każda prosta ścieżka z ustalonego węzła do liścia ma tyle samo brązowych węzłów

$x$  - węzeł

$H(x)$  - wysokość drzewa ukończonego w  $x$

$BH(x)$  - j.w. ale linie tylko węzły brązowe

$BH(x)$

$\geq -1$  - co najmniej tyle węzłów jest u drzewa ukończonego w  $x$

$$H(x) \leq 2BH(x)$$