

Algorytmy i Struktury Danych

Wykład 2

Problem sortouanie

Dane: ciąg a_0, \dots, a_{n-1} oraz dany ch
z operatorem \leq

Wynik: Permutacja a_0, \dots, a_{n-1} , taka że

$$a_0 \leq a_1 \leq \dots \leq a_{n-1}$$

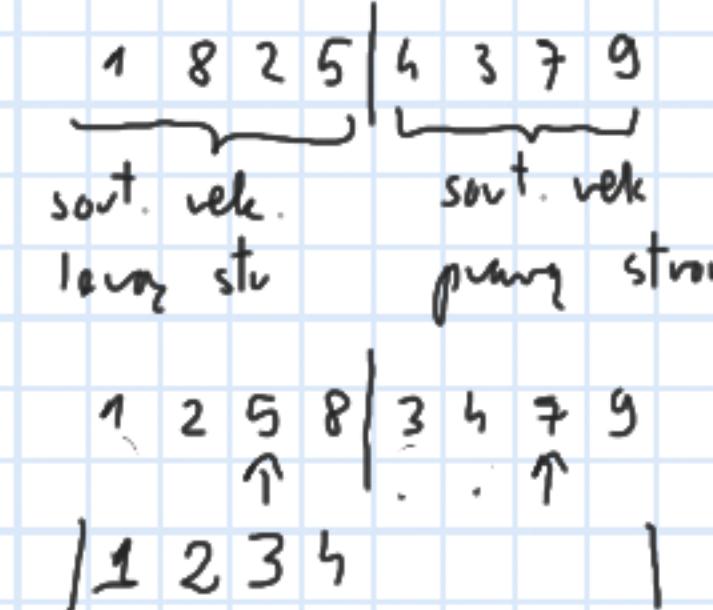
Uuagi

reprezentują danymi → tablica

- └ lista
- └ plik

algorytm sortowania

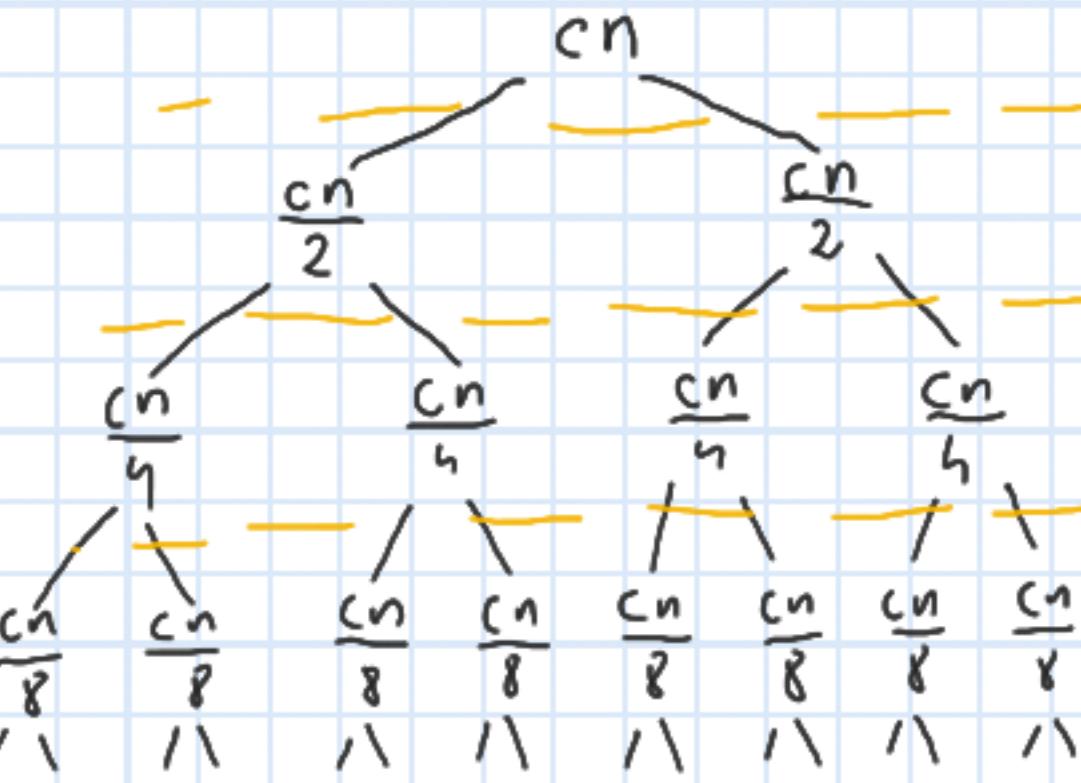
Sorți și merge și scalare (Merge Sort)



- ① Posortuj lewy pionowy danych
 - ② Posortuj prawy pionowy danych
 - ③ Scal posortowane fragmenty

Z Tziorosii vrazomy

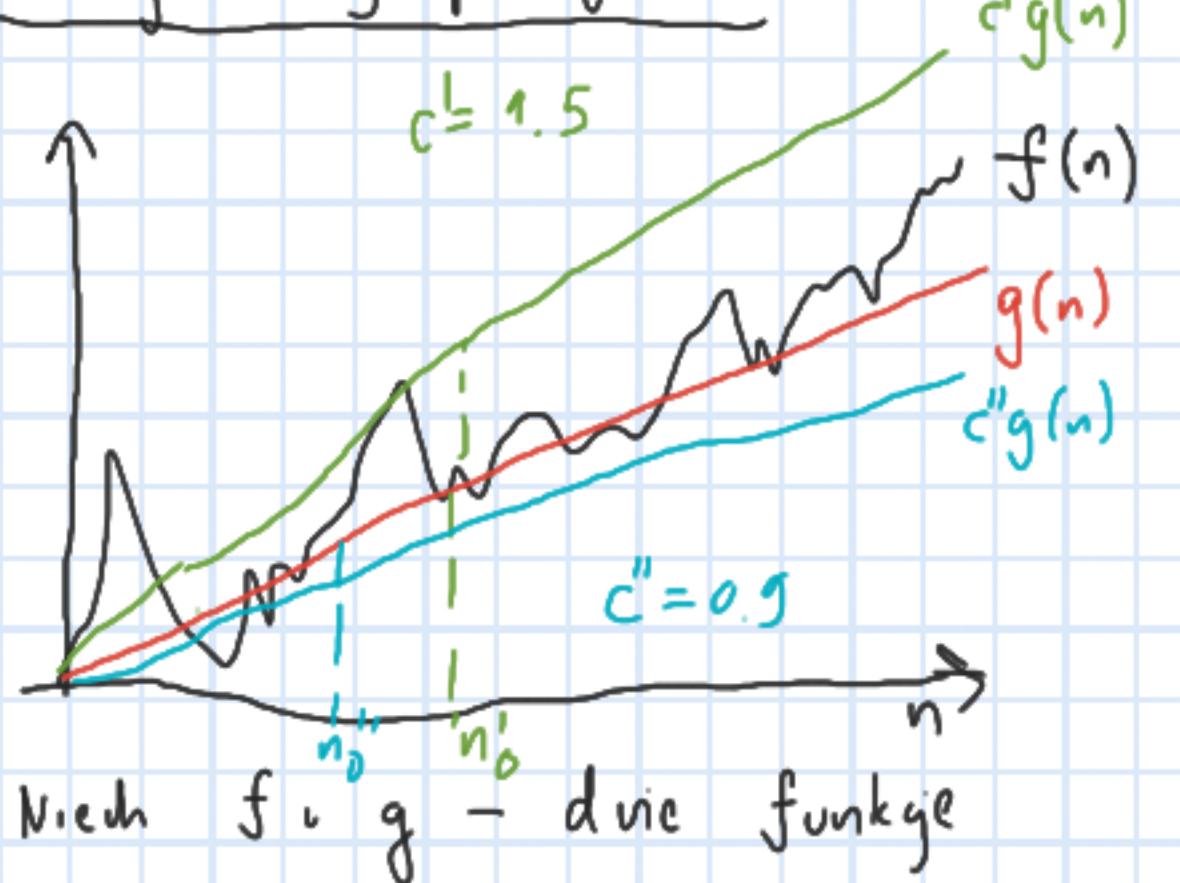
$$T(n) = \begin{cases} c & , n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + cn, & n^2 \end{cases}$$



Złożomie mowa to $T(n) \approx cn \log n$

$$\frac{n}{2^{\log n}} = \frac{n}{n} = 1$$

Notacja asymptotyczna



$$\begin{aligned}f &: \mathbb{N} \rightarrow \mathbb{N} \\g &: \mathbb{N} \rightarrow \mathbb{N}\end{aligned}$$

def

Mówimy, że f jest $O(g(n))$ jeśli:

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

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

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

Mówimy, że f jest $\Theta(g(n))$ jeśli jest $O(g(n))$
oraz $\Omega(g(n))$.

$$15n^3 + 7n^2 \text{ jest } \Theta(n^3)$$

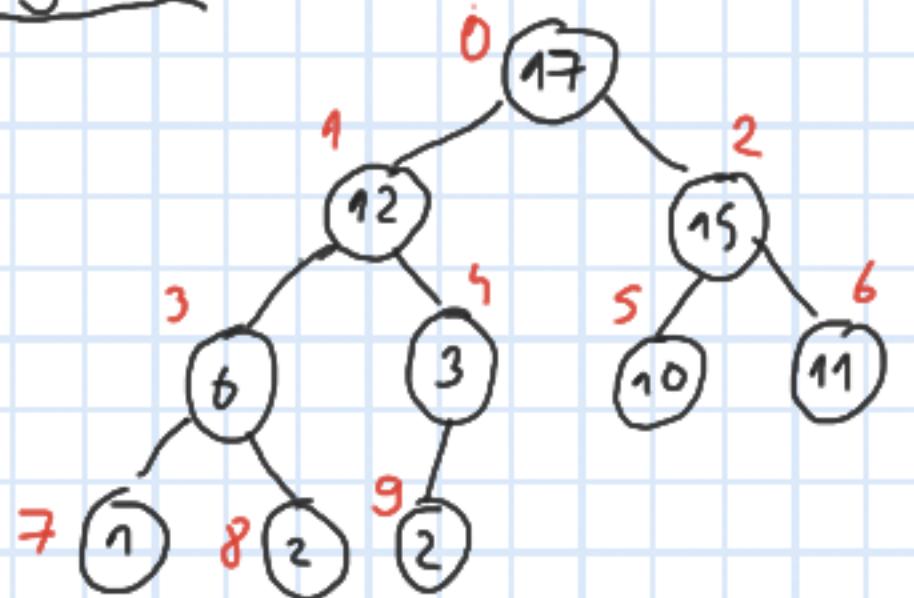
$$n^4 \text{ jest } O(n^5)$$

$$20n^2 + 7 \text{ jest } \Omega(n)$$

Sortovanie kopcove (Heap sort)

Kopiec - dnovo binarne, w ktorym kazdy wierzeciec ma jednego wieksza lub rowna niz jego dzieci

Pryklad

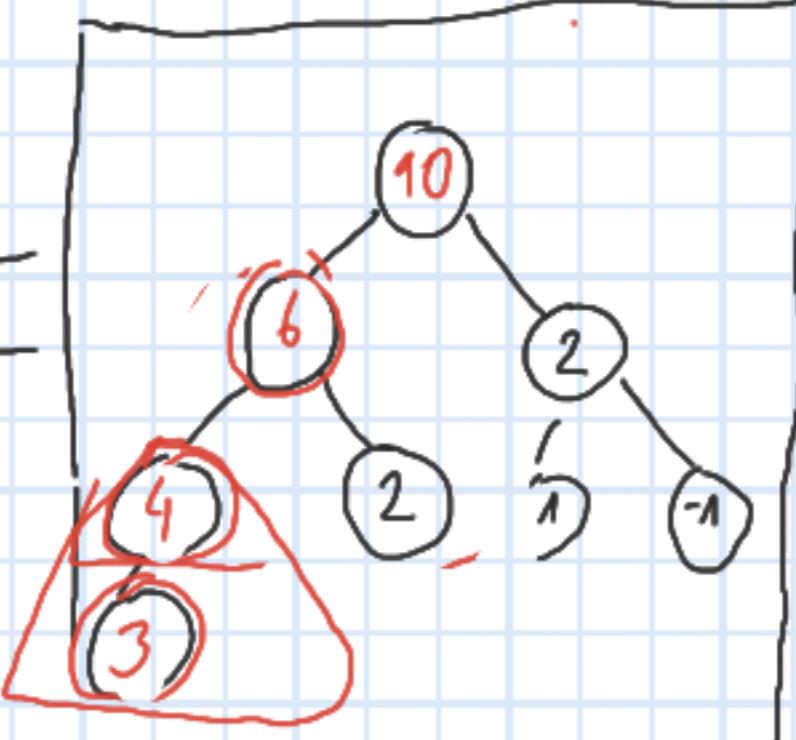


$$\text{left}(i) = 2i + 1$$

$$\text{right}(i) = 2i + 2$$

$$\text{parent}(i) = \lfloor i - 1/2 \rfloor$$

0	1	2	3	4	5	6	7	8	9
17	12	15	6	3	10	11	7	8	2



```

def left(i): return 2*i + 1
def right(i): return 2*i + 2
def parent(i): return (i-1)//2
  
```

Prywieracanie wtarnosci kopca

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

$$l = \text{left}(i)$$

$$r = \text{right}(i)$$

$$\text{max_ind} = i$$

if $l < n$ and $A[l] > A[\text{max_ind}]$: $\text{max_ind} = l$

if $r < n$ and $A[r] > A[\text{max_ind}]$: $\text{max_ind} = r$

if $\text{max_ind} \neq i$:

swap(A[i], A[max_ind])

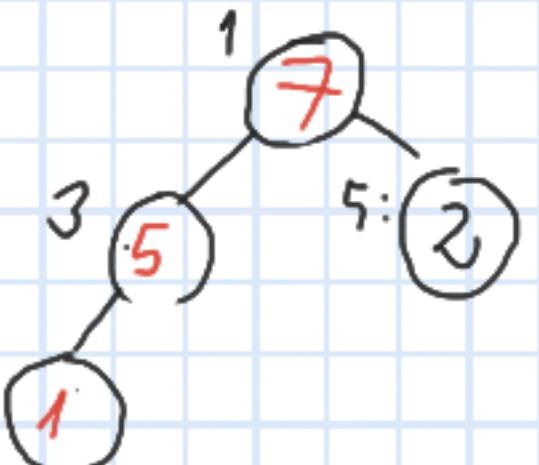
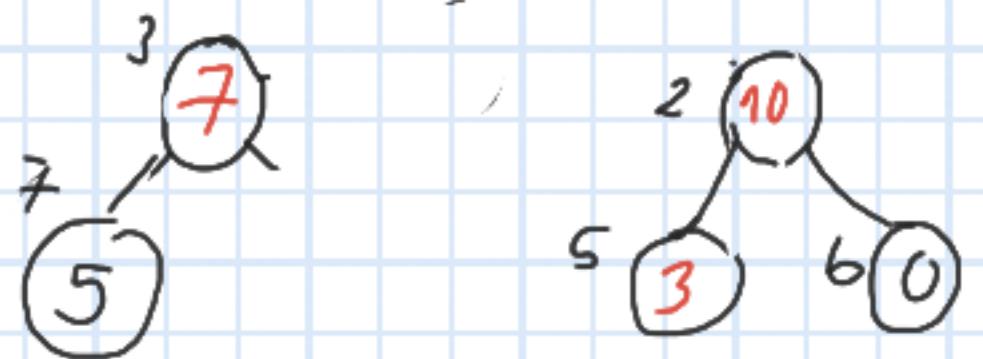
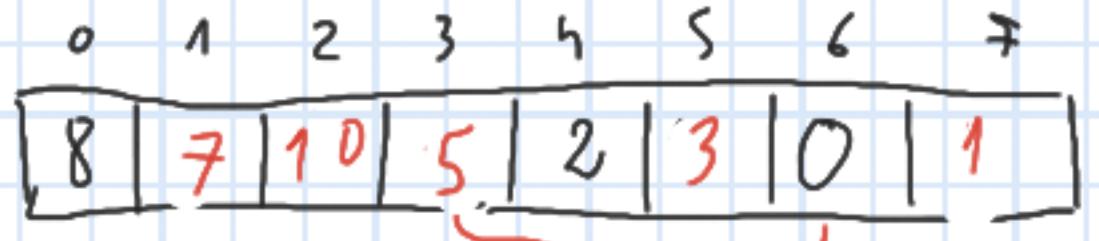
heapify(A, max_ind, n)

$\Theta(\log n)$



Budovanie kopca

```
def build_heap (A):
    n = len(A)
    for i in range( parent(n-1), -1, -1):
        heapify (A, i, n)
```



```
def heapSort (A):
    n = len(A)
    build_heap (A)
    for i in range (n-1, 0, -1):
        swap (A[0], A[i])
        heapify (A, 0, i)
```

$\Theta(n \log n)$

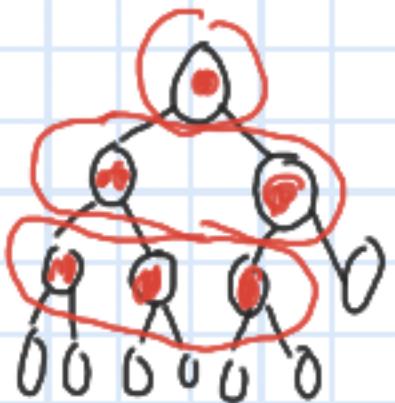
Algorytmy i Struktury Danych

Wykład 3

Analiza złożoności procedury Build Heap

wysokość kopca $\Theta(\log n)$

liczba elementów o: $\left\lceil \frac{n}{2^{h+1}} \right\rceil$
wysokość h



Szacowanie kosztu obliczeniowego:

$$O\left(\sum_{h=0}^{\lfloor \log_2 n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil h\right) = O\left(\sum_{h=0}^{\lfloor \log_2 n \rfloor} \frac{nh}{2^{h+1}}\right) \\ = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^{h+1}}\right) = O(n)$$

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

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

$$xf'(x) = x + 2x^2 + 3x^3 + \dots = \frac{x}{(1-x)^2} = \frac{1}{2} + 2\left(\frac{1}{2}\right)^2 + 3\cdot\left(\frac{1}{2}\right)^3 + \dots$$

$$x = \frac{1}{2} \Rightarrow \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

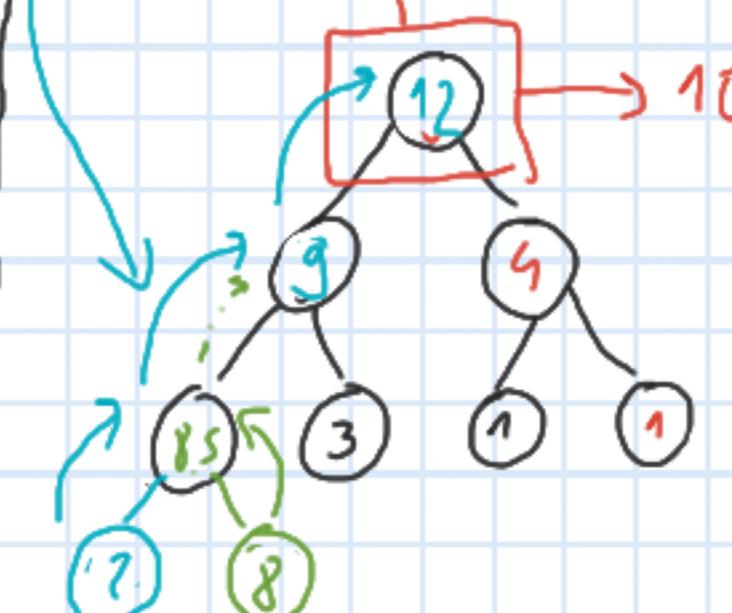
Kopiec jako kolejka priorytetowa

Struktura danych, do której mówimy układać elementy i ustawiać je zgodnie z priorytetem

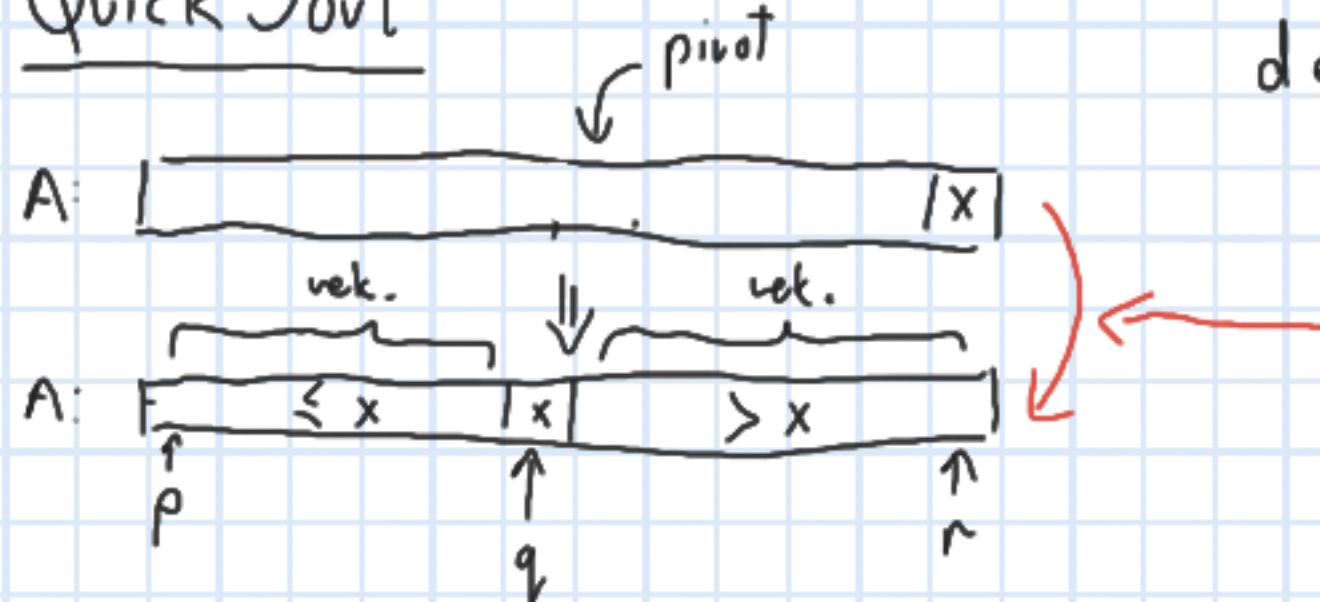
Operacje

- insert - ustaw element $\Theta(\log n)$

- extract max - ujmij element największy



Quick Sort



```
def quicksort(A, p, r):
```

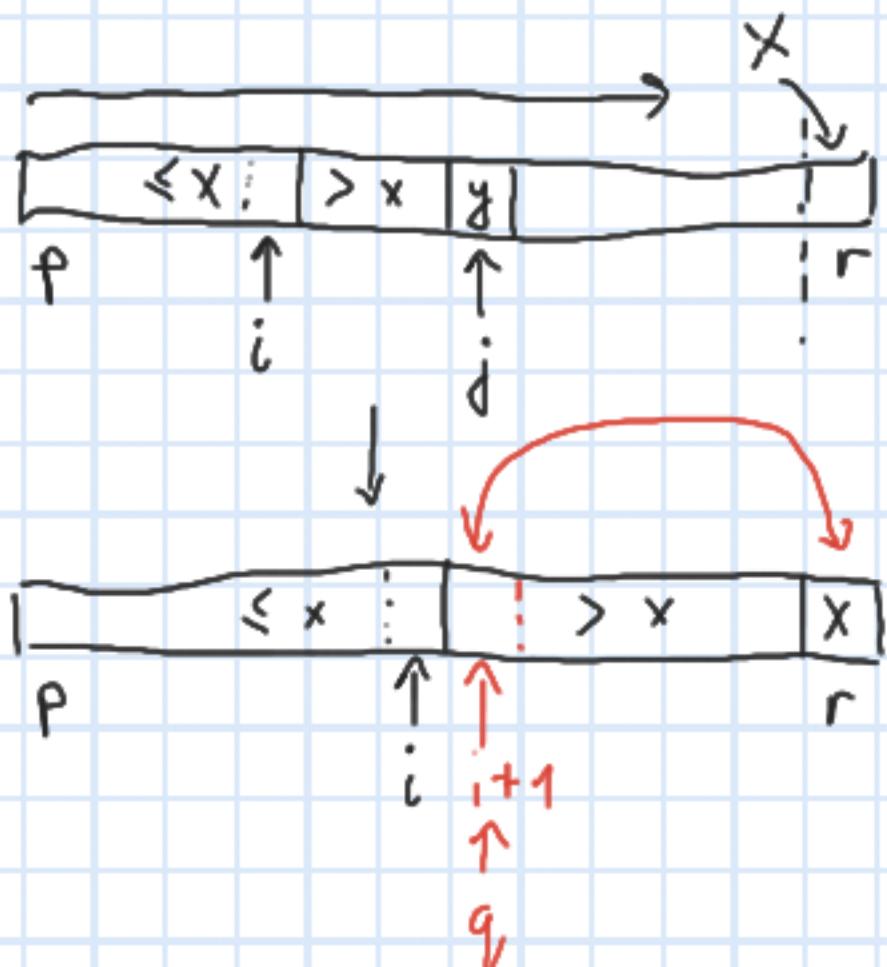
if $p < r$:

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

quicksort(A, p, q-1)

quicksort(A, q+1, r)

- u miejsce $A[v]$ umieszcza się "lepszy pivot"
- losowy element tablicy
- mediana z pierwszego, ostatniego i środkowego elementu
- ...



```
def partition(A, p, r):
```

$x = A[r]$

$i = p - 1$

for j in range(p, r):

if $A[j] \leq x$:

$i += 1$

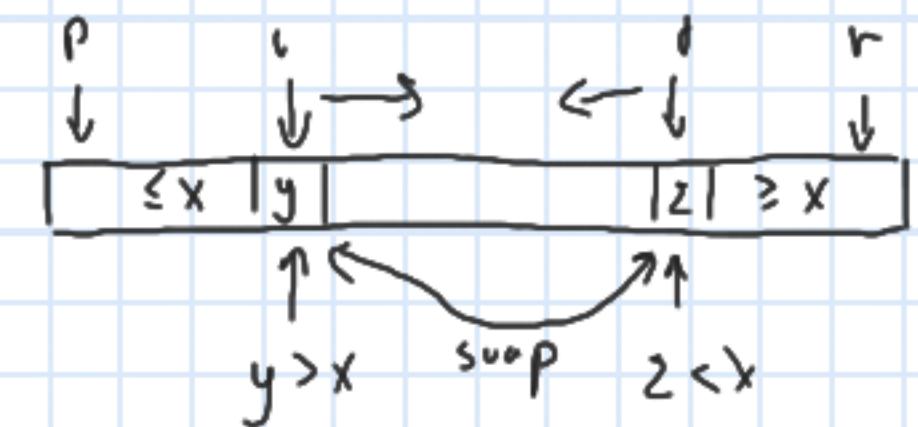
swap($A[i], A[j]$)

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

return $i + 1$

złożoność

$O(n)$



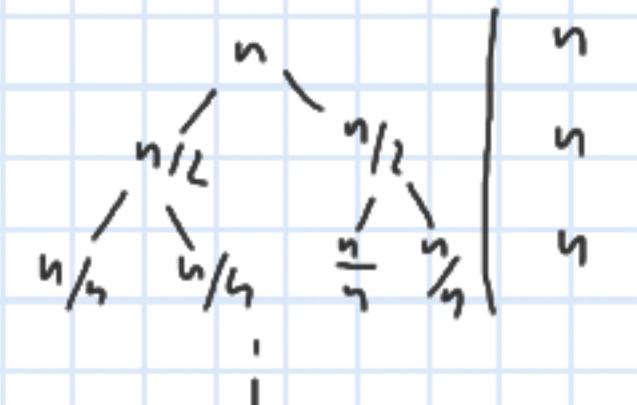
partition Hoare'a

partition Lomuto

Analiza złożoności obliczenowej QuickSorta

- idealne podziałы $\rightarrow T(n) = \begin{cases} c, & n \leq 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + cn, & n > 1 \end{cases}$

$$= \Theta(n \log n)$$



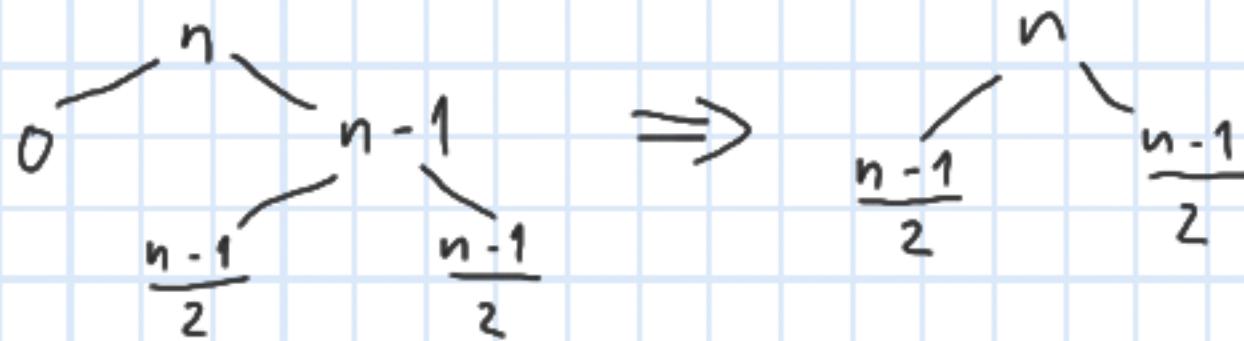
$$\Theta(n \log n)$$

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

$$\begin{aligned} T(n) &= cn + T(n-1) \\ &= cn + c(n-1) + T(n-2) \\ &\vdots \\ &= cn + c(n-1) + c(n-2) + \dots + c = \Theta(n^2) \end{aligned}$$

- mieszanka szerygu i realna

- co drugi rozział \rightarrow podział idealny
- co drugi najgorsze



Usunigue rekurenji ogonovej

```
def quicksort(A, p, r):
```

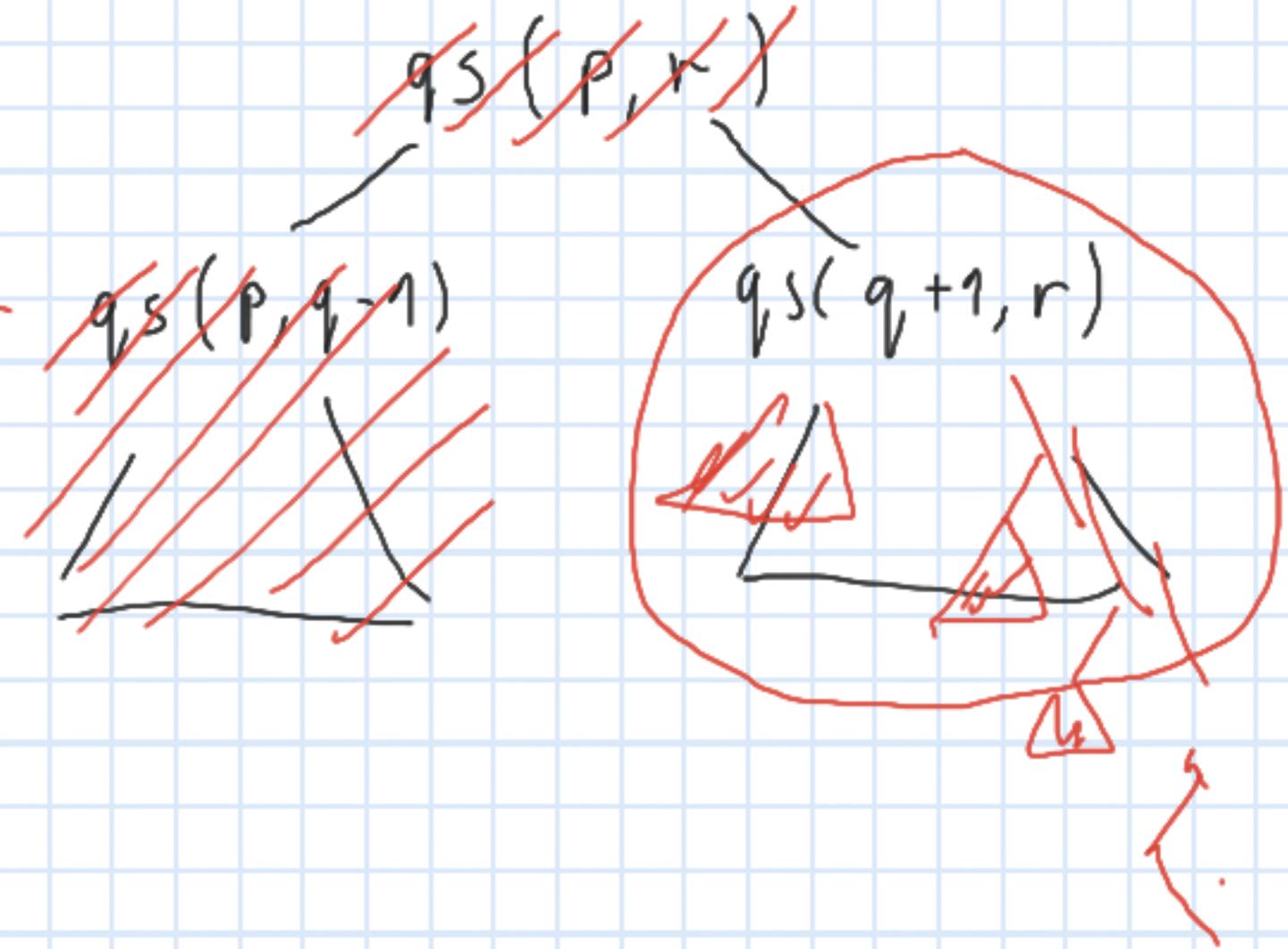
```
    while p < r:
```

```
        q = partition(A, p, r)
```

```
        quicksort(A, p, q - 1) ←
```

```
P = q + 1
```

quicksort(A, q + 1, r)



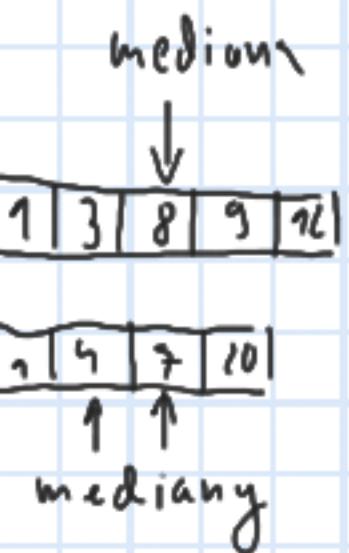
Statystyki pozycyjne

$\text{select}(A, k)$ - k -ta statystyka pozycyjna, czyli element A , który leży na k -ej pozycji w posortowaniu

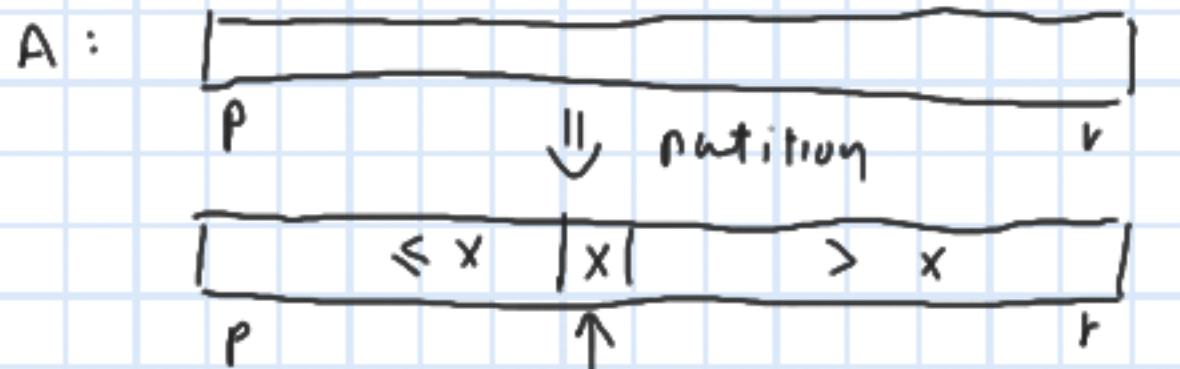
$$\text{select}(A, 0) = \min(A)$$

$$\text{select}(A, \text{len}(A)-1) = \max(A)$$

$$\text{select}(A, \text{len}(A)/2) = \text{median} A$$



select(A, k)



jeli $q = k$ to zwracamy x
jeli $q > k$ to rekurencyjnie na "połowie" tablicy
 $q < k$

zliczanie i zwracanie podziałów

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

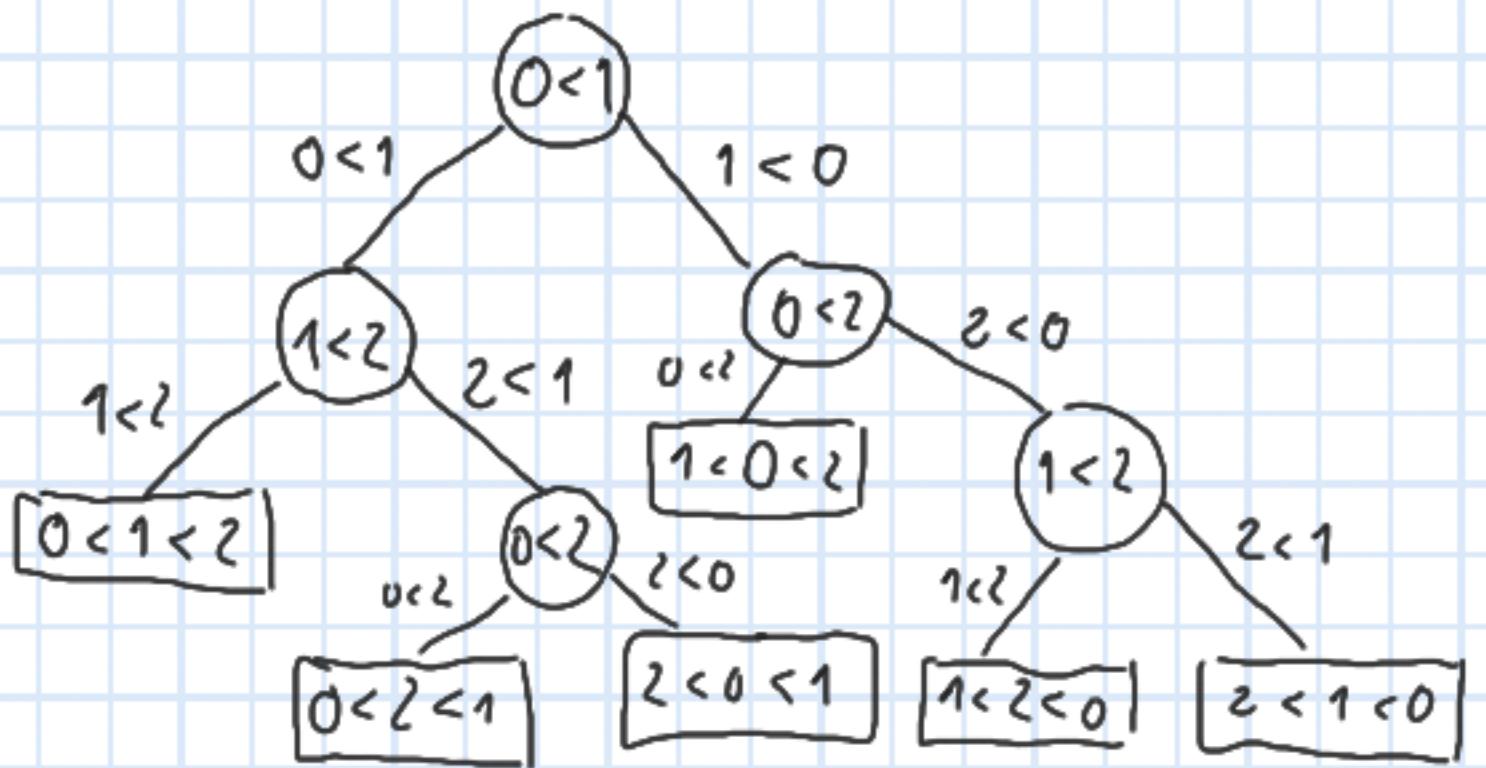
$$T(n) = cn + \frac{cn}{2} + \frac{cn}{4} + \frac{cn}{8} + \dots = O(n)$$

Algorytmy i Struktury Danych

Wykład 4

Dane ograniczenie na szybkość sortowania

x	y	z
0	1	2



Jeśli dane opisuje sortowanie tablicy n elementów
to musi mieć co najmniej $n!$ liści

Dane o wysokości h ma najwyżej 2^h liści
 $n! \leq 2^h$

Wysokość drzewa = liczba porównań!

$$\begin{aligned}
 n! &= n(n-1)(n-2) \cdots 1 = \\
 &n(n-1)(n-2) \cdots \underbrace{\left(\frac{n}{2}\right)\left(\frac{n}{2}-1\right)}_{\geq \left(\frac{n}{2}\right)^{\frac{n}{2}}} \cdots 1
 \end{aligned}$$

zakt. prawiste

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

Sortowanie przez zliczanie (Counting Sort)

Mamy tablicę A, która zawiera n liczb naturalnych z zakresu $\{0, 1, \dots, k-1\}$

$$k=4$$

def countingsort(A, k):

$$n = \text{len}(A)$$

$$C = [0] \times k$$

$$B = [0] \times n$$

for i in range(n):

$$C[A[i]] += 1$$

for i in range(1, k):

$$C[i] = C[i] + C[i-1]$$

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

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

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

założoność

$$\Theta(n+k)$$

A	0	1	2	3	4	5	6	7
	2	0	3	1	1	2	3	2

C	0	1	2	3
	1	2	3	2

C	0	1	2	3
	1	3	6	8

B	0	1	2	3	4	5	6	7
	0	1	1	2	2	2	3	3

0	1	3	6
	0	1	3

for i in range(n):

$$A[i] = B[i]$$

Rabin Sort - sortowanie pozycyjne

kra	kra	kił	avr
avr	ati	atı	
kot	kil	kił	
kit	arz	kot	avr
ati	kot	kit	kra
kil	arz	avr	

\Rightarrow

Sortowanie n słów, każde o długości t
 Zajęło $\Theta(n t)$ czasu

Kubetek może być mniej niż n , ale
 powinna to być liczba postanowiona $\propto n$

Bucket Sort - sortowanie kubatkowe

A - tablica listów umieszcanych

$0 \leq A[i] < 1$, elementy A będą wygenerowane zgodnie
 z rozkładem jednostajnym

n - rozmiar tablicy

Tworzymy n kubetków (listy na elementy A)

kubek 0	kubek 1	- - -	kubek $n-1$
$[0, \frac{1}{n})$	$[\frac{1}{n}, \frac{2}{n})$		$[\frac{n-1}{n}, \frac{n}{n})$

Preglądamy tablicę A i aktualnie listy $A[i]$ umieszczamy
 w kubetku $\lfloor n \cdot A[i] \rfloor$

Kiedy kubetek sąsiaduje (np. przy wąskim sortowaniu prostego)

Pisujemy dane z kubetków do A w kolejności posortowania

Złożoność $\Theta(n)$

Magninie 5ki - wybór k-go elementu w
czasie $O(n)$ bez ryzyka "okradzieniem"

A - tablica n liczb

Chcemy znaleźć liczbę, która po posortowaniu
byłaby na pozycji k

1. Podziel A na $\lceil \frac{n}{5} \rceil$ grup po 5
elementów

2. Rekurencyjnie znajdź x, medianę
wsród median naszych piętek

3. Wykonaj "partition" względem x jako pivot

4. Jeżeli x jest na pozycji k, to zwinić x

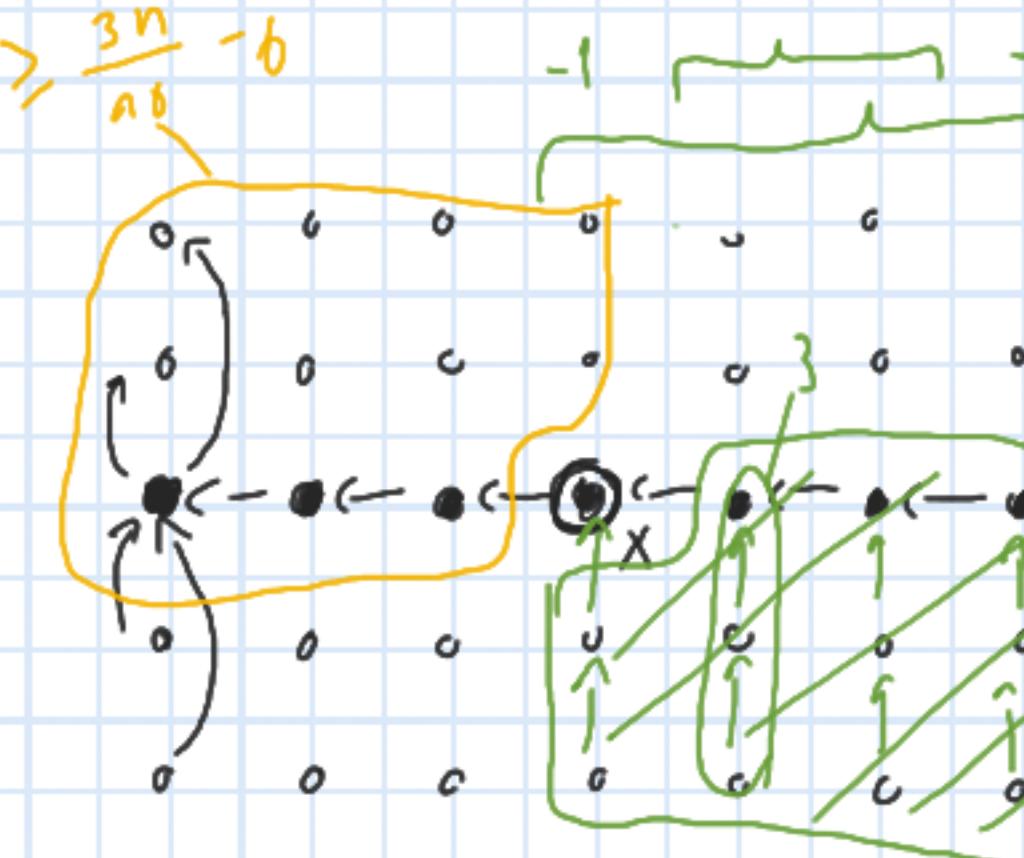
Jeżeli x jest na pozycji dalszej niż k to

szukaj rekurencyjnie w lewej części tablicy,

w mniejszym czasie w prawej

wybór
pivotu

klasyfikuj
select



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

$$\bar{T}(n) = \begin{cases} \Theta(1) & , n \leq \text{perna stała} \\ \bar{T}\left(\lceil \frac{n}{5} \rceil\right) + \bar{T}\left(\frac{7n}{10} + 6\right) + \Theta(n) & \end{cases}$$

mając
wybór
c taka, że
ta wartość
jest ujemna

Twierdzimy, że $\bar{T}(n) \leq cn$ dla pewnej $c \geq 1$

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

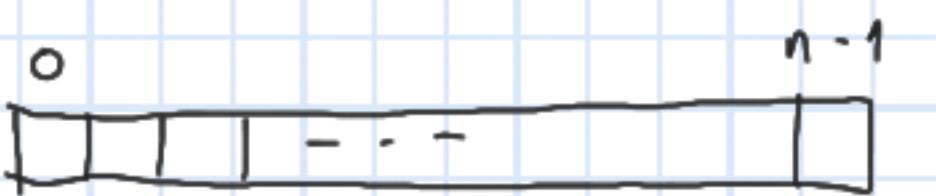
Algorytmy i Struktury Danych

Wykład 5

Abstrakcyjne struktury danych

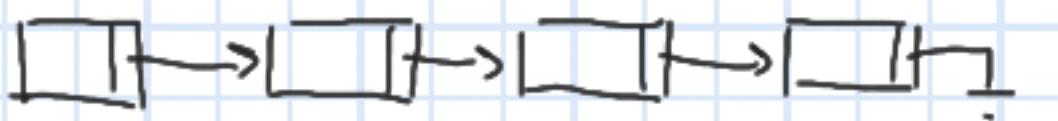
- "kontrakt" co do działania
- zestaw operacji
- fizyczna realizacja

Tablica



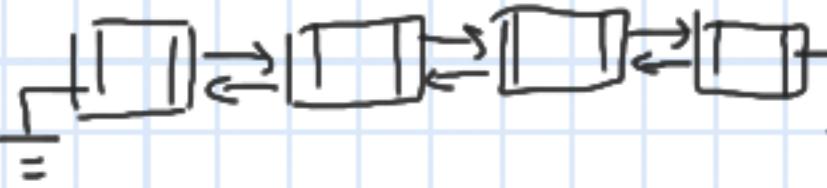
"Cos co ma ponumerowane komórki, do których można się bezpośrednio odwoływać"

Lista jednoliciwunkowa



"Cos co pozwala usuwać od przętla do końca i wpinać/usunąć elementy"

Lista dwuściennkowa



"Cos co pozwala wędrować w obie strony i wpinać/usuwać elementy"

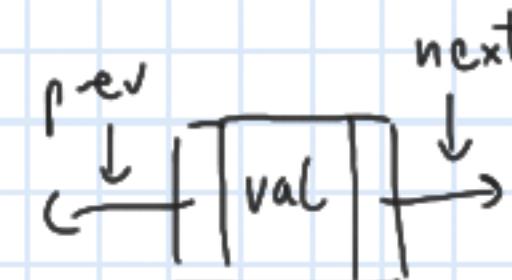
class DNode:

def __init__(self):

self.prev = None

self.next = None

self.val = None



Lista dwuściennkowa = jednogm wskaźnikiem

1 1 0 1 0 1 0 1

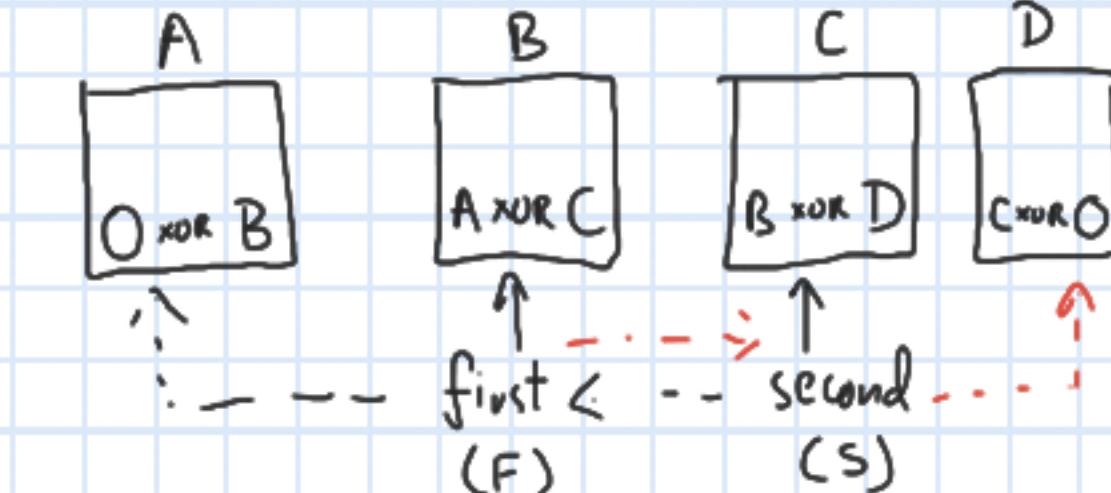
XOR 0 1 1 1 0 0 1 1

1 0 1 0 0 1 1 0

$$(A \text{ XOR } B) \text{ XOR } B = A$$

$$(A \text{ XOR } B) = (B \text{ XOR } A)$$

$$(A \text{ XOR } B) \text{ XOR } A = B$$



$$(A \text{ XOR } C) \text{ XOR } S.\text{ptr} =$$

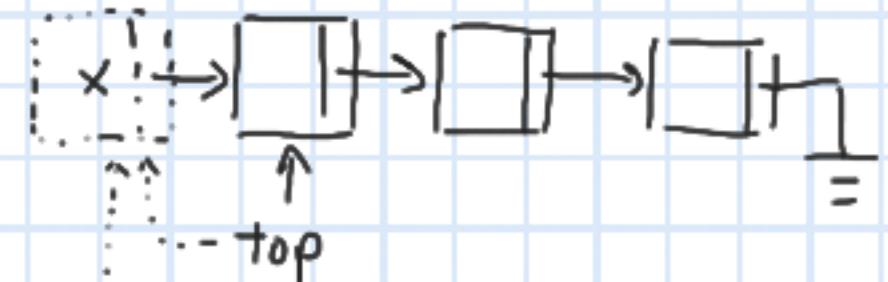
$$(A \text{ XOR } C) \text{ XOR } C = A$$

Stos (ang. stack)

"Cos, co powala odkładając elementy na szczyt, lub z niego zdejmując."

- push(x) ← ułóż x na stos
- pop() ← zdejmij element ze stosu
- isEmpty() ← czy stos jest pusty

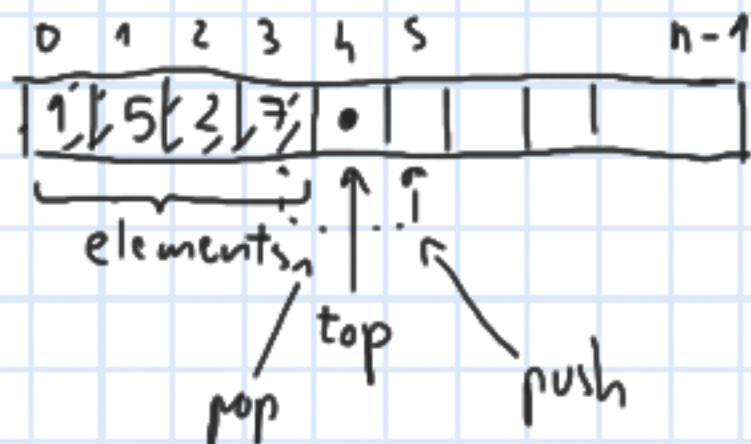
Implementacja na liście jednokierunkowej



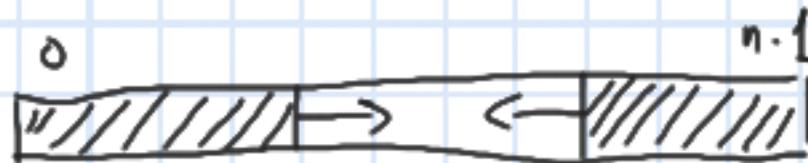
push(x)

pop()

Implementacja tablicowa



Dwa stosy



starsze komputery PC XT/AT

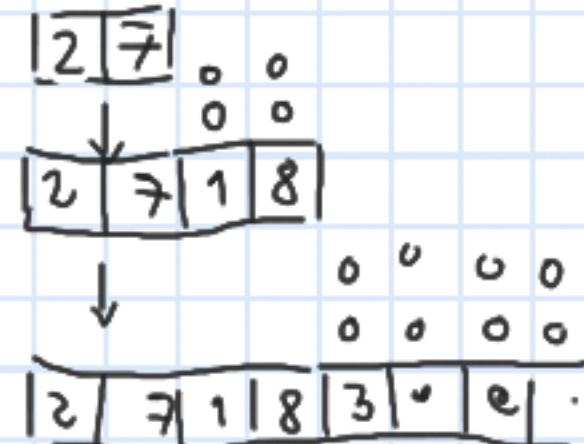
Co jeśli słowny nam się
miejscie w zaallokowanej tablicy?

- wówczas tworzymy nową,
2x większą tablicę i kopiujemy
do niej zawartość stosu

Analiza kosztu zamontowania

push - 3 zł

pop - 1 zł



Kolejka (ang. queue)

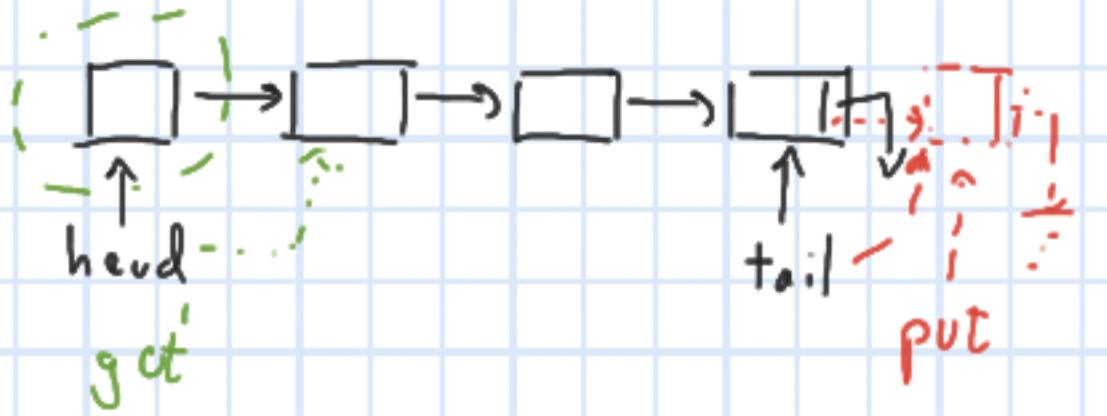
"coś, co przenosi ustwiać elementy na koniec"

używając "przeszły"

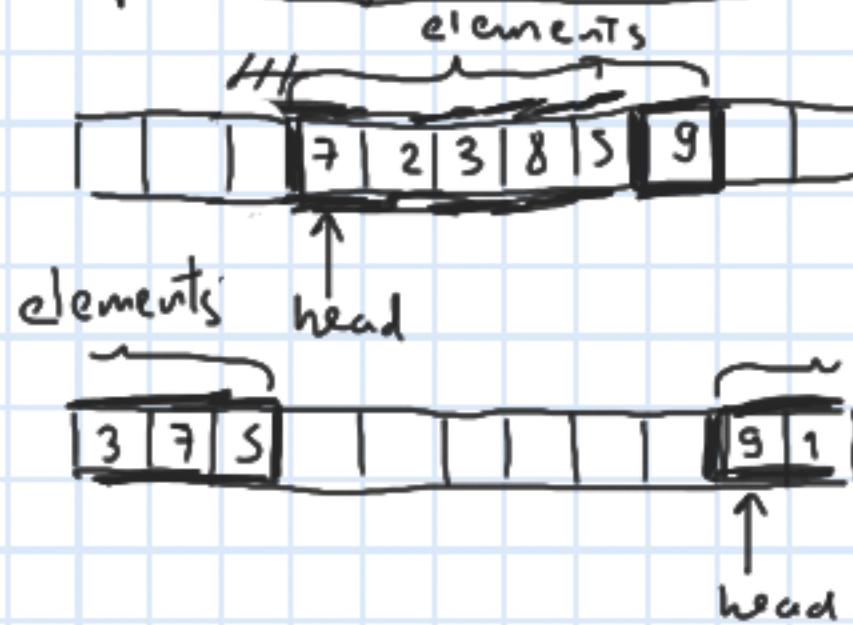
put / get

enqueue / dequeue

Implementacja listowa



Implementacja tablicowa



Przepłynienie: ta sama
metoda co przy stosie

Standardowa zagadka

Jak zaimplementować kolejkę
mając do dyspozycji tylko
dwa stosy?

Kolejka priorytetowa (ang. priority queue)

"Coś do tego uładamy elementy poszczególnie

z priorytetem, a uzywamy w kolejnoscie malejących priorytetów"

- insert
- extract max
- decrease key

Tablica asocjujaca / słownik

"Coś co można indeksować dowolnymi wartościom"

$$A["\text{stan}"] = 7$$

$$A["\text{mysz}"]$$

$$A[(1, 19)]$$

Implementacja

	insert	extract
- kopiec binarny	$O(\log n)$	$O(\log n)$
- posortowana tablica	$O(n)$	$O(1)$
- nieposortowana tablica	$O(1)$	$O(n)$
- j.u. dla listy		
- wiele innych		

Implementacja

- drzewo BST	$O(\log n)$
└ AVL	
└ RB / red-black	
└ B - drzewo	
└ drzewo Splay	
- skip lista	
- tablica hashująca	$O(1)$

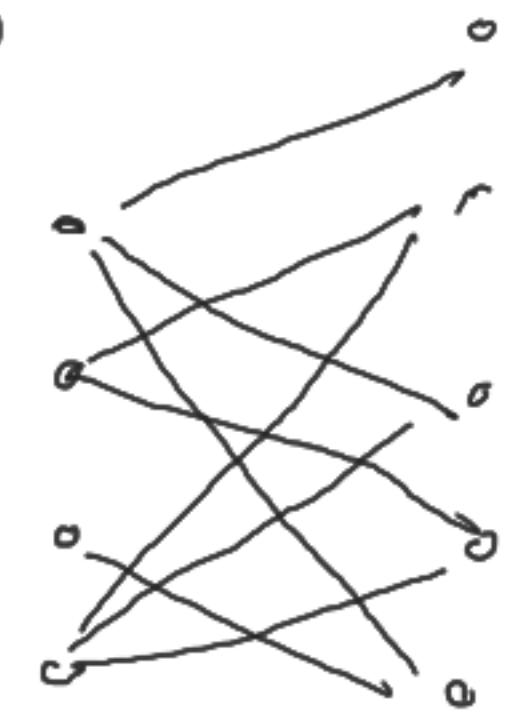
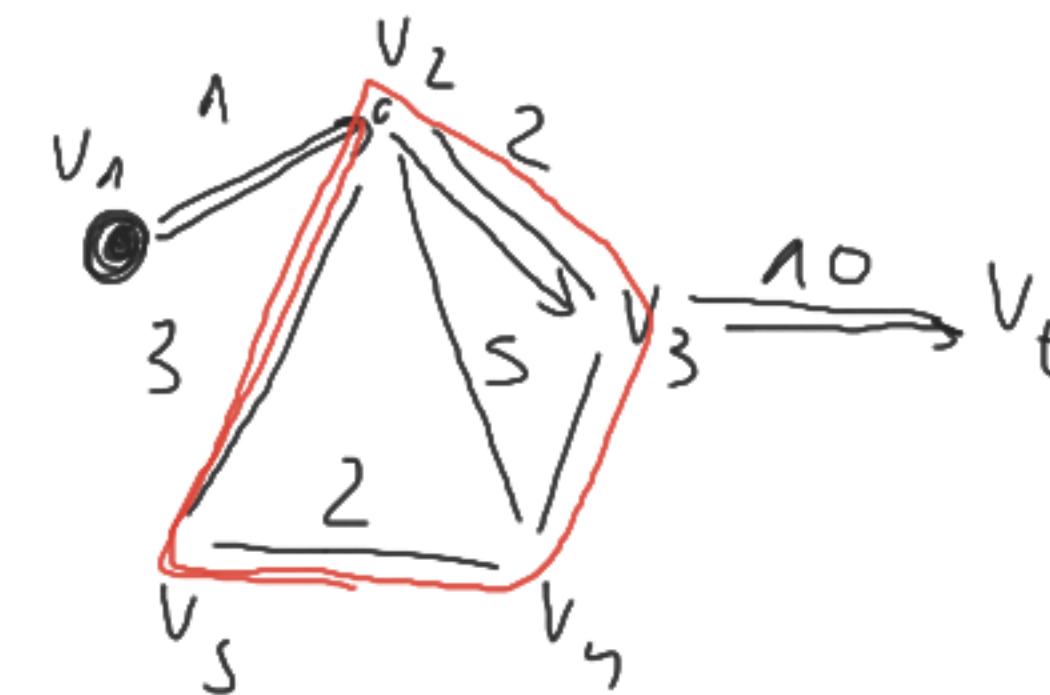
Algorytmy Grafowe

$$G = (V, E)$$

$$V = \{v_1, \dots, v_n\}$$

$$E = \{e_1, \dots, e_m\}$$

e_i - zbiór dwudziestu wierzchołków



Algorytmy i Struktury Danych

Wykład 6

Algorytmy grafowe

na ogół nie dopuszczamy
petli, węzli krawędzi (u, u)

Graf skierowany:

$$- G = (V, E)$$

$$- V = \{v_1, \dots, v_n\} - \text{zbiór wierzchołków}$$

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

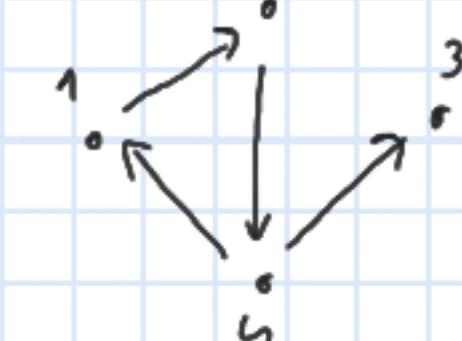
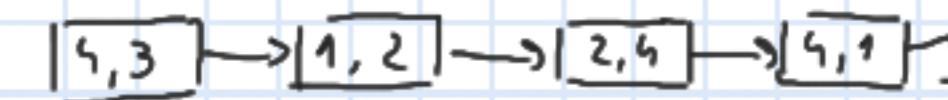


Graf nieskierowany jest zdefiniowany analogicznie,
ale kiedy krawędź to dwielementowy zbiór wierzchołków

↳ na ogół reprezentujemy jako graf skierowany
z krawędziami w die strony

Często z krawędziami i wierzchołkami związanej
dodatkowe informacje (np. wagę / długość)

Reprezentacja przez listy/tablice krawędzi



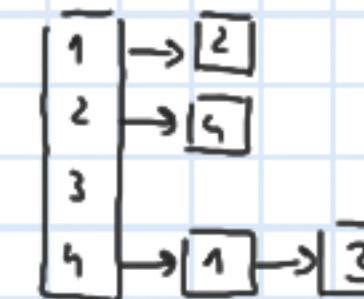
Reprezentacja macierzowa

	1	2	3	4
1	-	T	F	F
2	F	-	F	T
3	F	F	-	F
4	T	F	T	-

- tworzenie węzła istnieje krawędzi $O(1)$

- przegląd krawędzi wychodzących z
danego wierzchołka $O(n)$

Reprezentacja przez listy sąsiedztwa



- sprawdzenie węzła istnieje krawędzi
ma złożoność $O(n)$

- przegląd krawędzi wychodzących
z danego wierzchołka V_{out}
złożoność $O(d(v))$

stopień $v \rightarrow$ liczba wychodzących krawędzi

Algorytm BFS (Breadth-First Search)

mejście grafu uszere

```
def BFS(G, s)
# G = (V, E), s ∈ V
```

```
Q = Queue()
```

```
for v ∈ V: v.visited = False
```

```
s.d = 0
```

```
s.visited = True
```

```
s.parent = None
```

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

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

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

```
for v - sąsiad u:
```

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

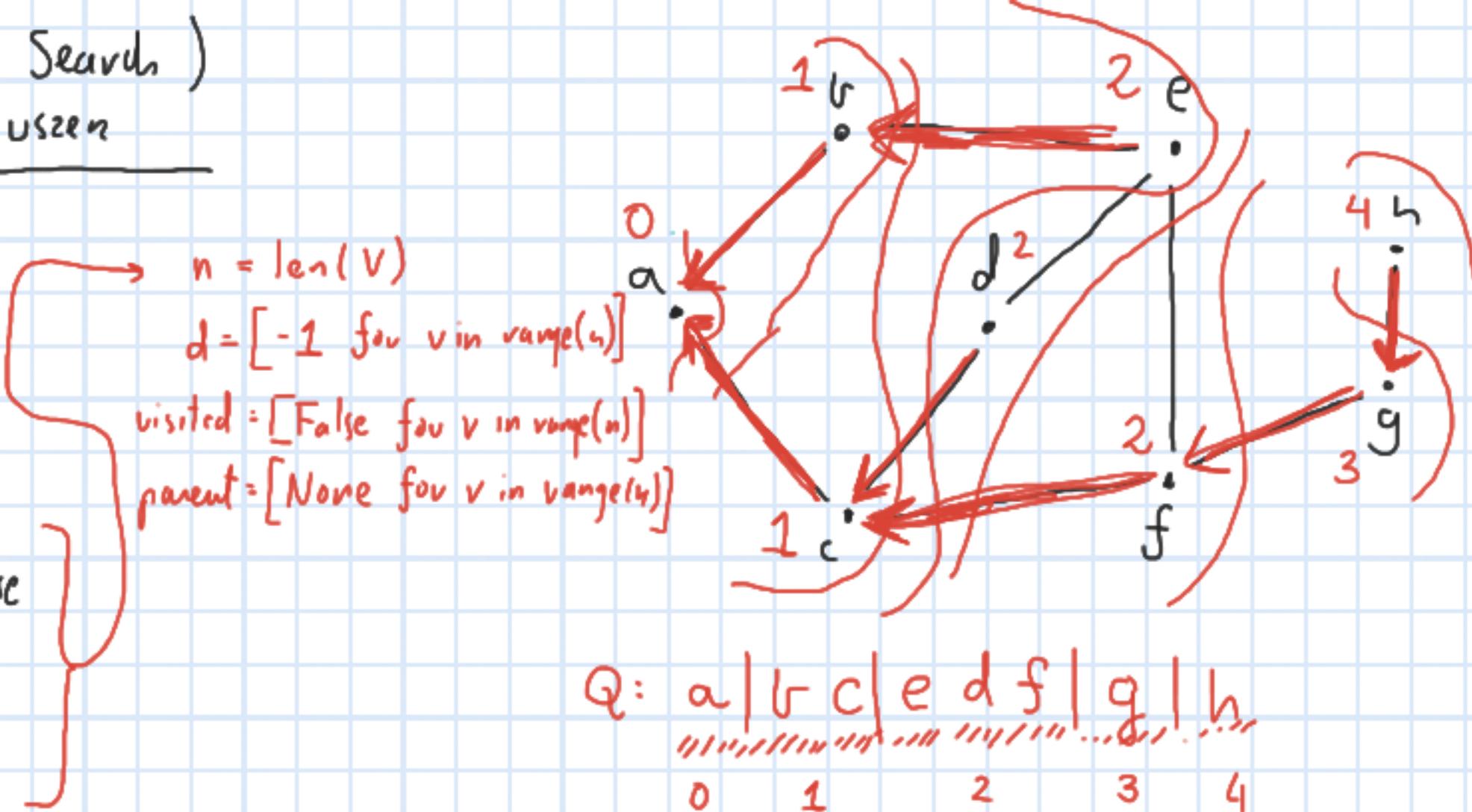
```
v.d = u.d + 1
```

```
v.parent = u
```

```
v.visited = True
```

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

return d, parent, visited



Q: a | b | c | d | e | f | g | h | i
0 1 2 3 4

BFS - znajduje najkrótsze ścieżki
w sensie liczby krawędzi

Inne zastosowania:

- tafowanie spojów w grafu
- diudzichowic
- wykrywanie cykli

Zastosowanie

$O(V+E)$ - rep. listami

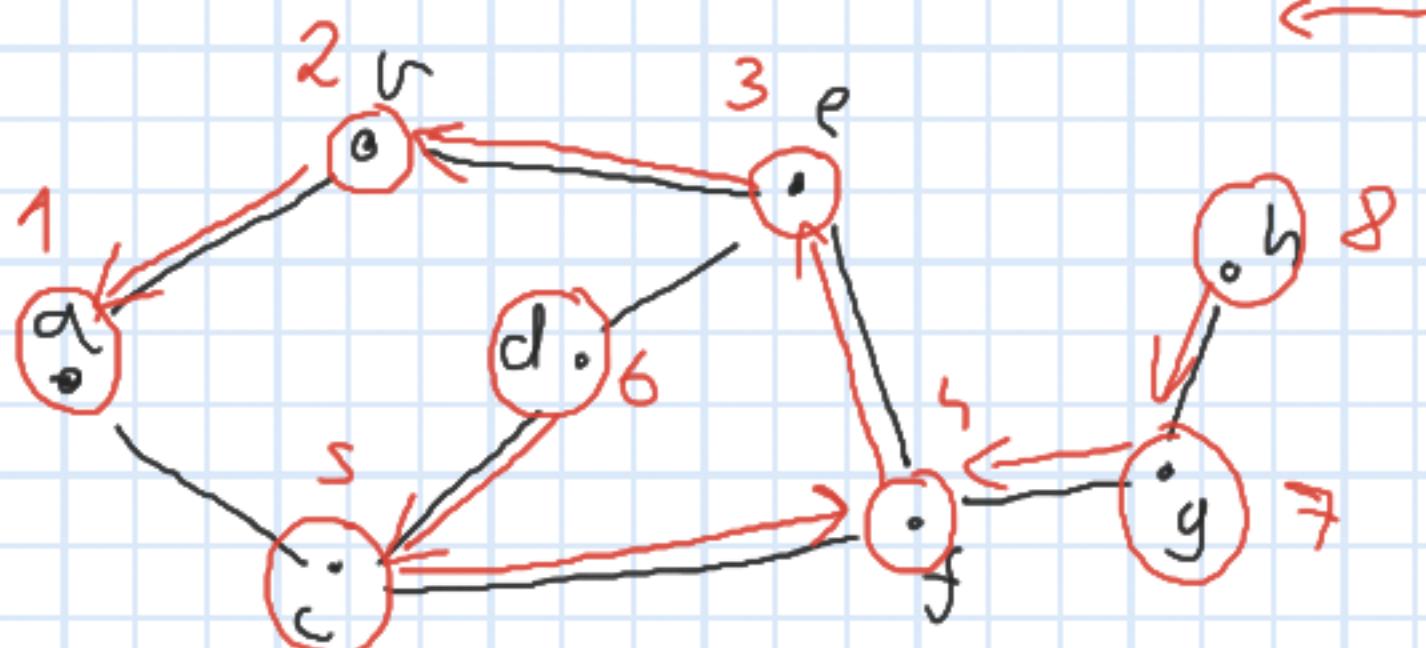
$O(V^2)$ - rep. macier

Algorytm DFS (depth -first search)
przeszukwanie w głębi

```

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)

```



dof DFSVisit (G, u)

nonlocal time

time += 1

a.visited = True

fou v - sg siad u:

if not v.visited:

$$v.\text{parent} = u$$

DFS Visit (G, v)

Time t = 1

więchnoTel u został odniedzony
- i to jest czas odniedzenia

— u zostat pmetronony /
crus pmetronemia

Zložení

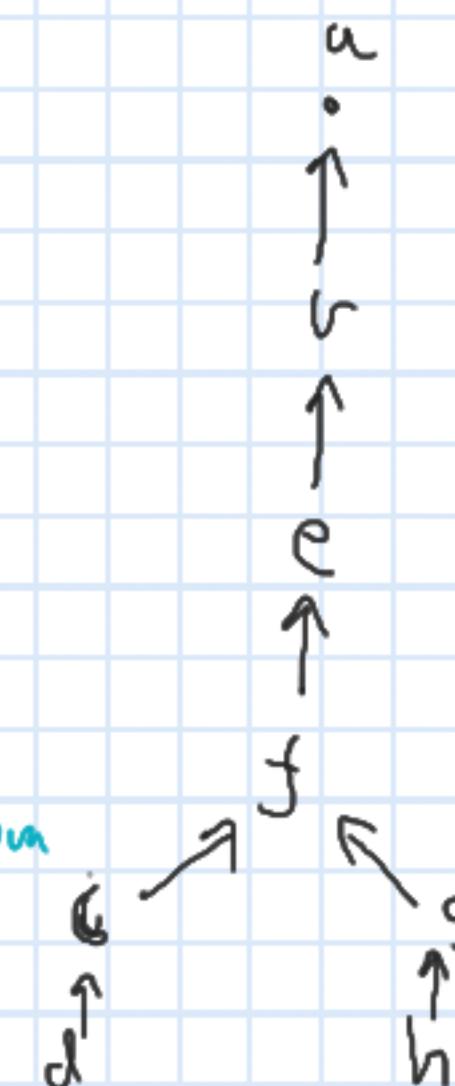
$O(V+E)$ - reprezentacja listowa

$O(V^2)$ - representacja macierzą

Zastosovania DFS

- spojnosť
 - dvochielnosť
 - výkryvacie cykly
 - sočtoracie topologické
 - silne spojné stredove
 - cykl Eulera
 - mosty / body artikulační

dnes DFS

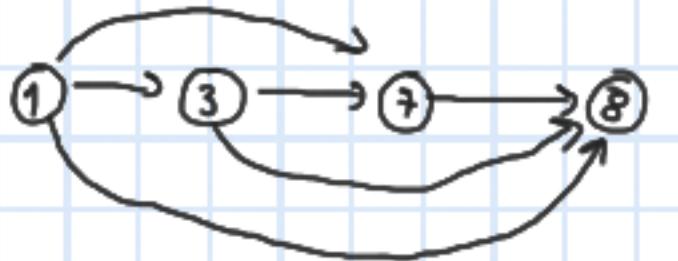


Sortowanie topologiczne

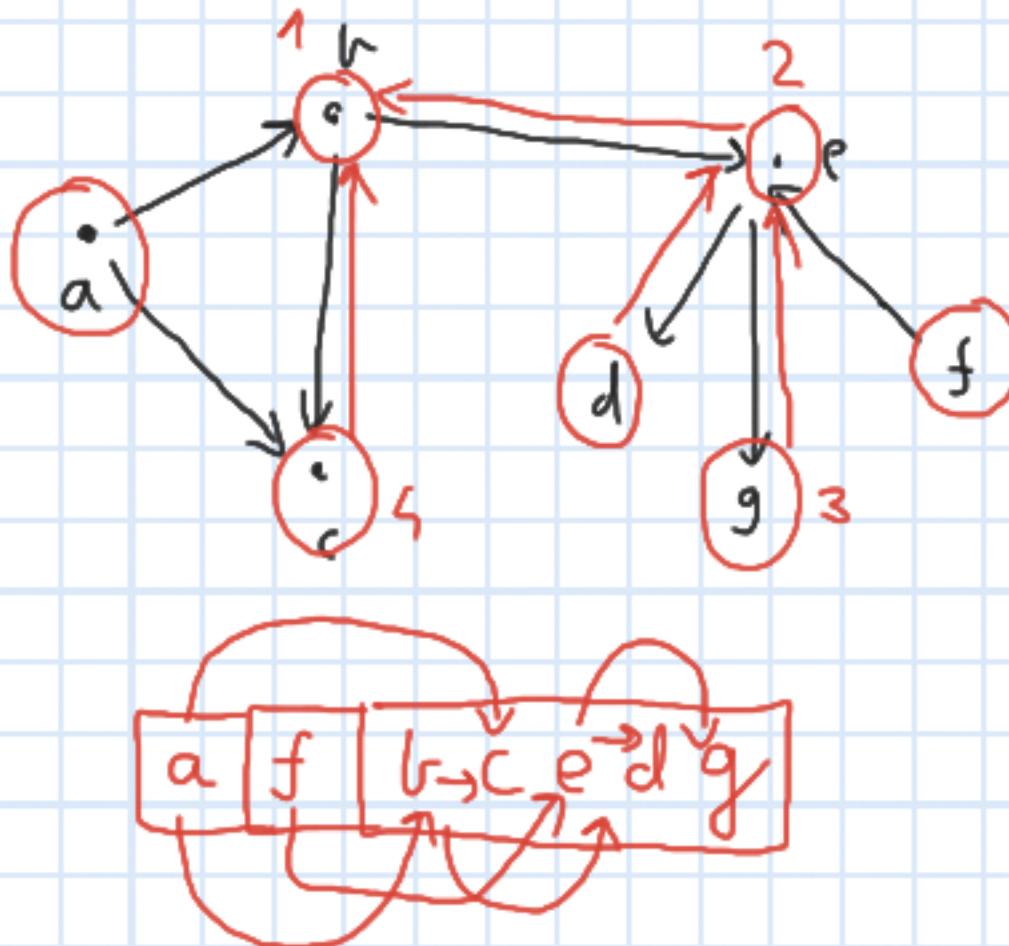
dag - directed acyclic graph

(skierowany grafacykliczny)

sortowanie topologiczne dagu: ułożenie jego wierzchołków w taki kolejności, że krawędzie wskazujące wychodzą z lewej na prawą



zastosowanie: wyznaczenie kolejności realizacji zadań jeśli niektóre muszą być wykonane przed innymi.



Algorytm

- uzupełnij DFS
- po "przebranemu" każdego wierzchołka dopisz go na koniec tworzonej listy

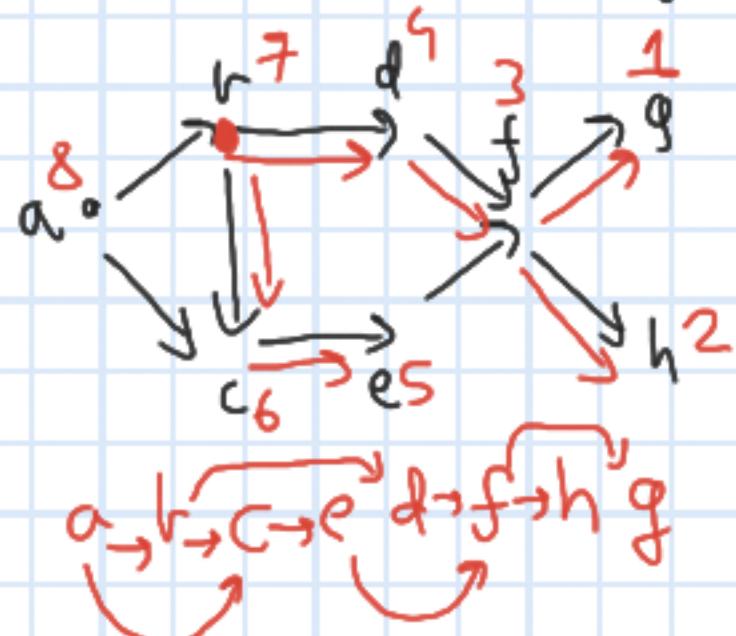
Algorytmy i Struktury Danych

Wykład 7

Zastosowania algorytmu DFS

① Sortowanie topologiczne DAGu

- uruchamiany DFS
- po metronemie wierzchołka dopisujemy go na koniec listy



② Cykl Eulera

def Cykl Eulera w grafie G to taki cykl, który przechodzi przez każdy krawędź G dokładnie raz

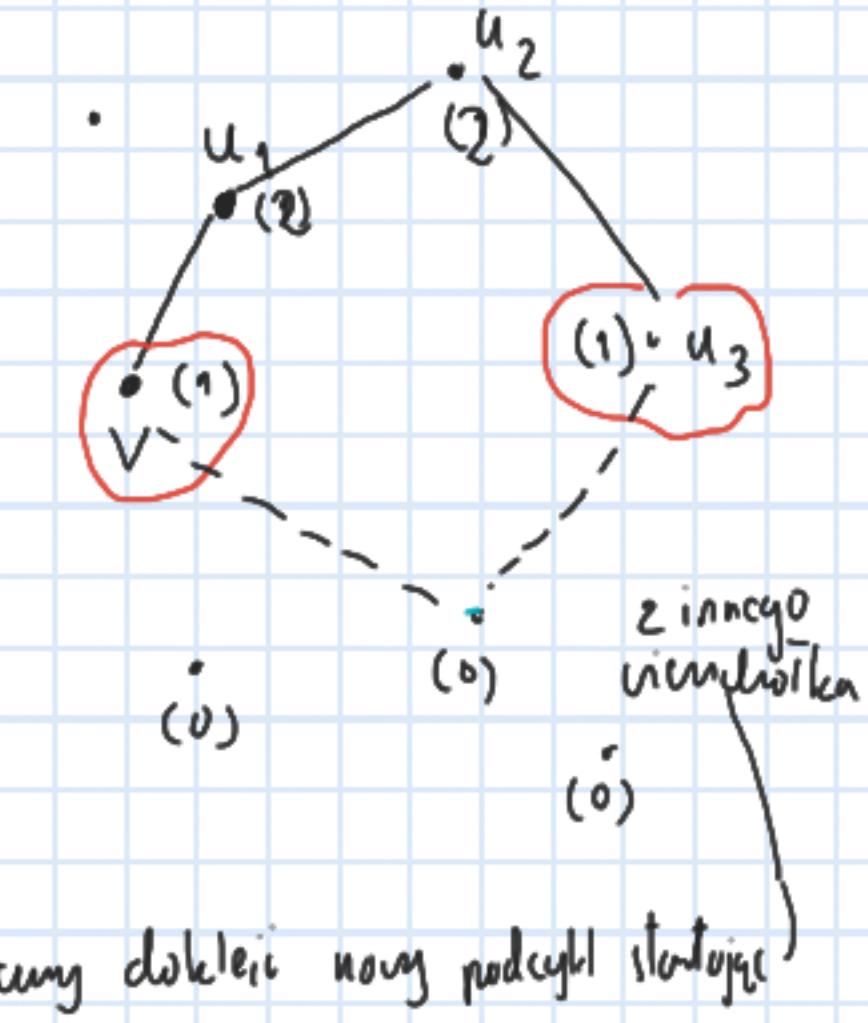
tw Graf nieskierowany i spójny posiada cykl Eulera
której kiedyś jego wierzchołek ma parzysty stopień

"dowód"

Ważne koniczyny: odciągle

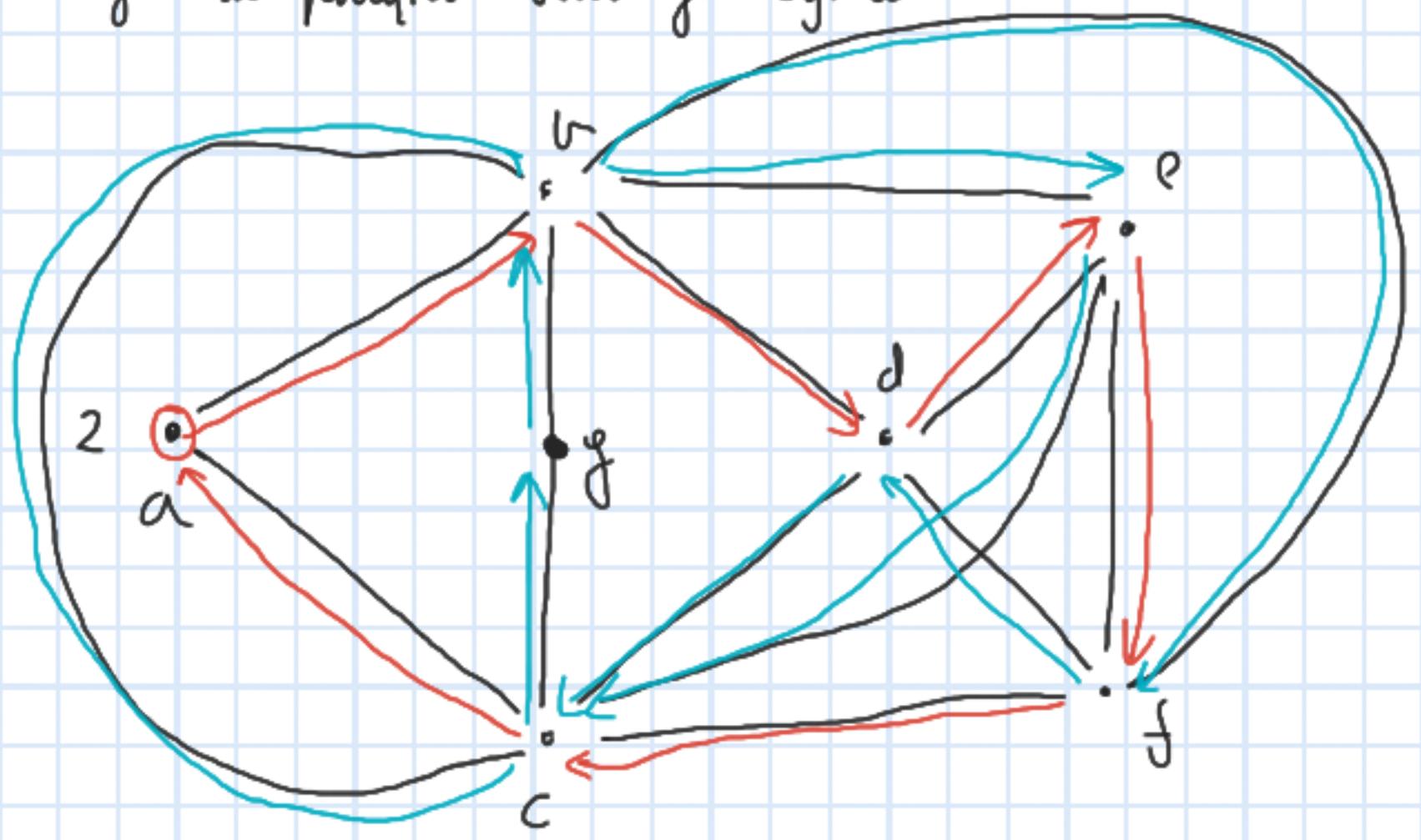
Ważne wystarczające:

- wyliczamy stopnie wierzchołków v ,
- startując z v wędrujemy, za każdym razem wybierając dovolną jeszcze nie wizytowaną krawędź
- w końcu z powrotem dotknemy do v
- = albo znaleźliśmy cykl Eulera, albo mamy dalej nowy podcykl startując



Algorytm

- wykonujemy DFS, ale:
- po drodze "uwaramy" krawędzie po których przeliszczyliśmy
- do danego wierzchołka mówimy wejść dwoma linkami raz
- po nawiązaniu danego wierzchołka dopisujemy go na koniec trwającego cyklu



a b c d e f g h i j

Złożoność

Taka sama jak DFS

$$\underbrace{O(V+E)}_{\text{rep. lista}}, \quad \underbrace{O(V^2)}_{\text{rep. macierza}}$$

także w czasie w porównaniu złożoności

$$O(V^2+VE), \quad O(V^3)$$

def Cykl Hamiltona to cykl prosty, który odwiedza każdy wierzchołek dokładnie raz

Algorytm: Brute-force — sprawdz $O(V!)$ możliwości

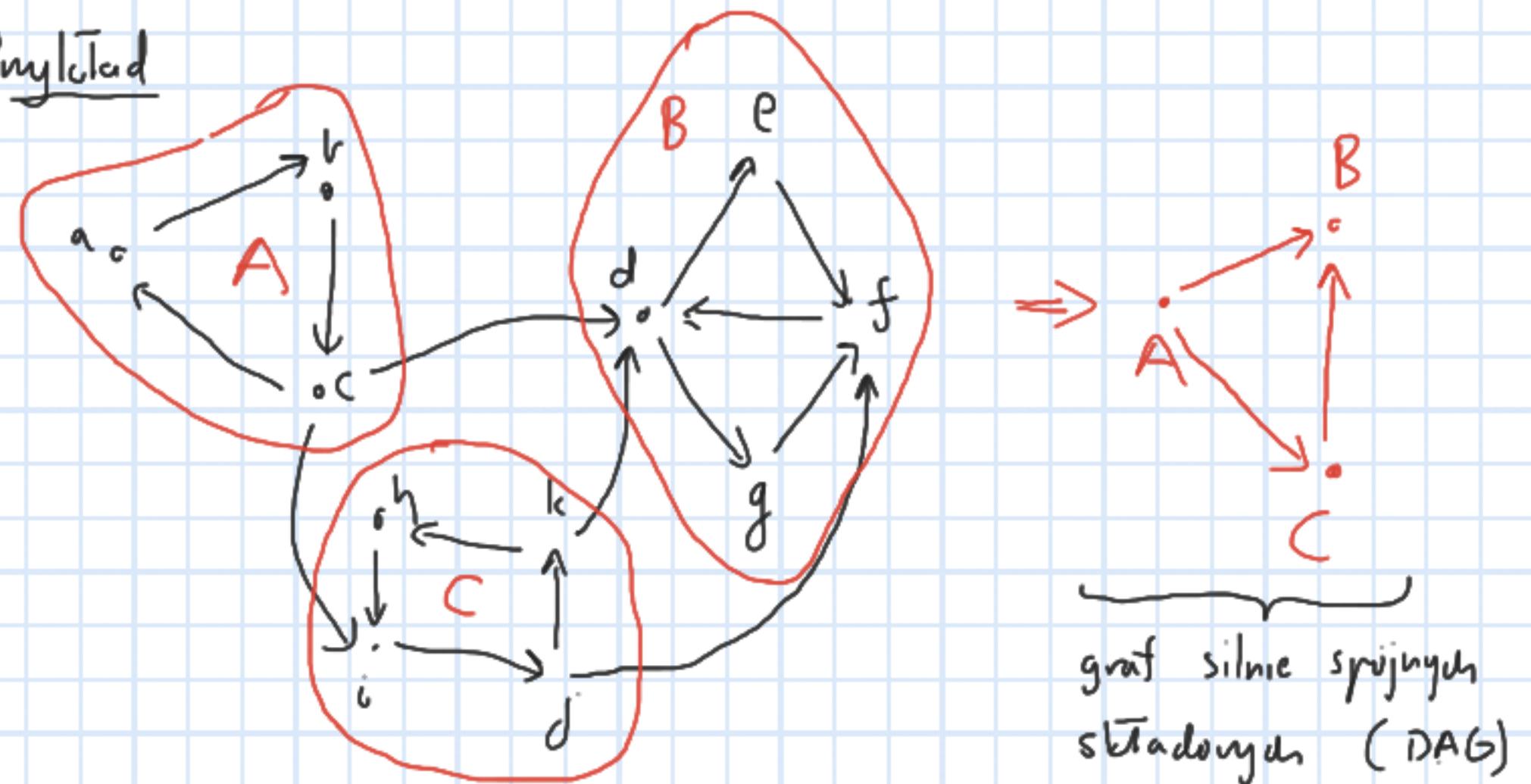
Nie da się wiele lepiej, bo rozpoznawanie w grafie

można cykl Hamiltona jest problemem NP-zupełnym

③ Silnie spójne składowe w grafie skierowanym

def Niech $G = (V, E)$ będzie pewnym grafem skierowanym. Mówimy, że $u, v \in V$ należą do tej samej silnie spójnej składowej jeśli istnieje ścieżka skierowana z u do v oraz z v do u .

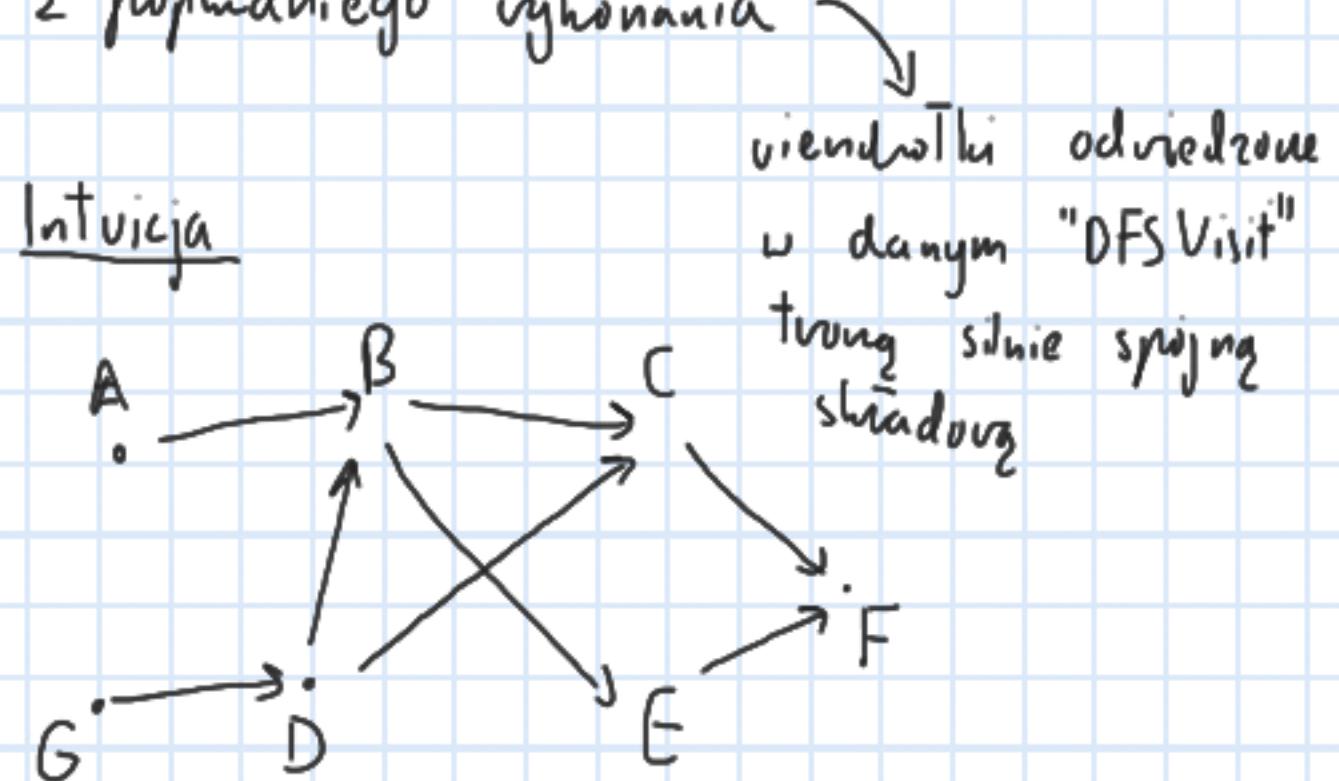
Ponkładać

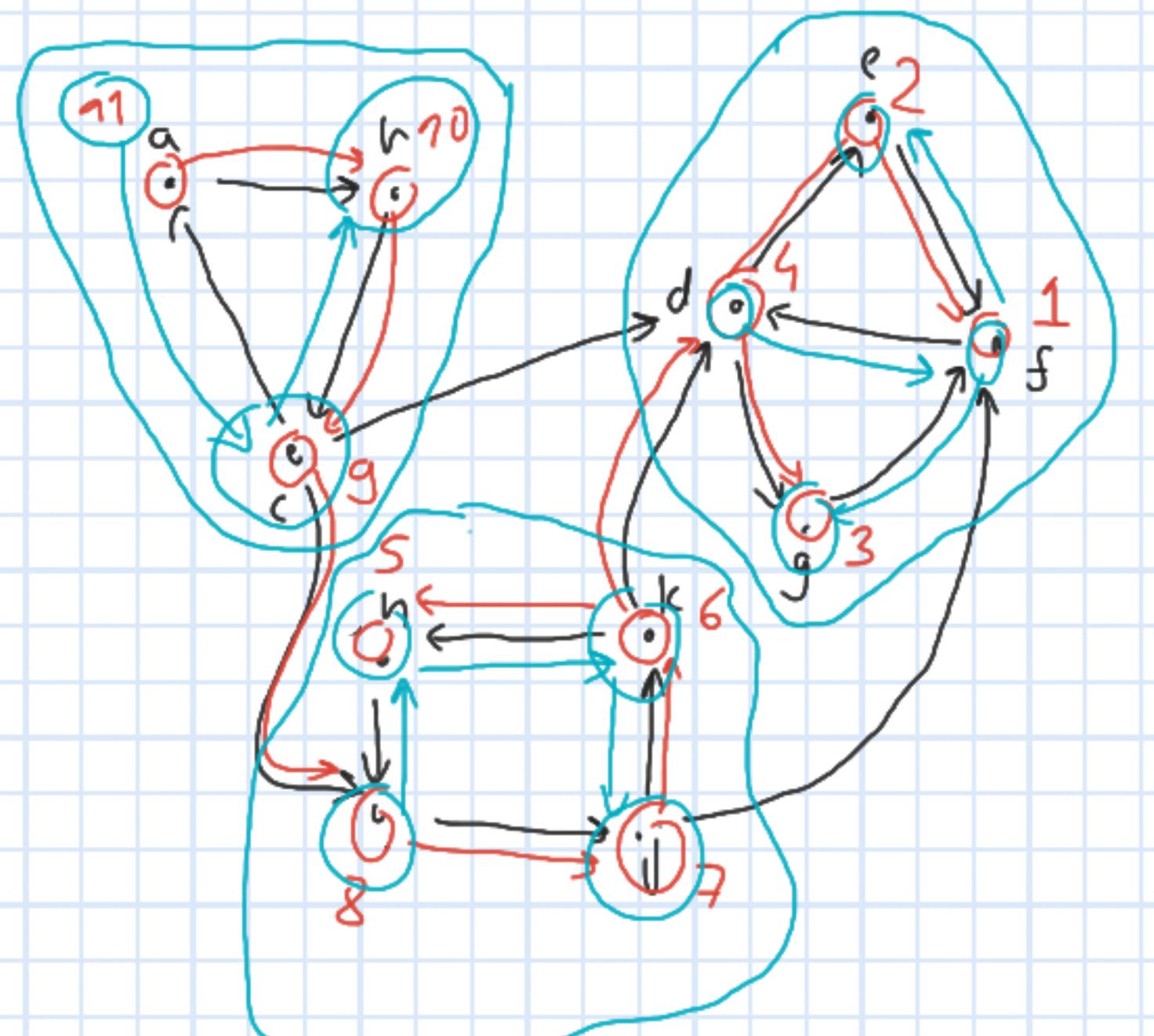


Algorytm

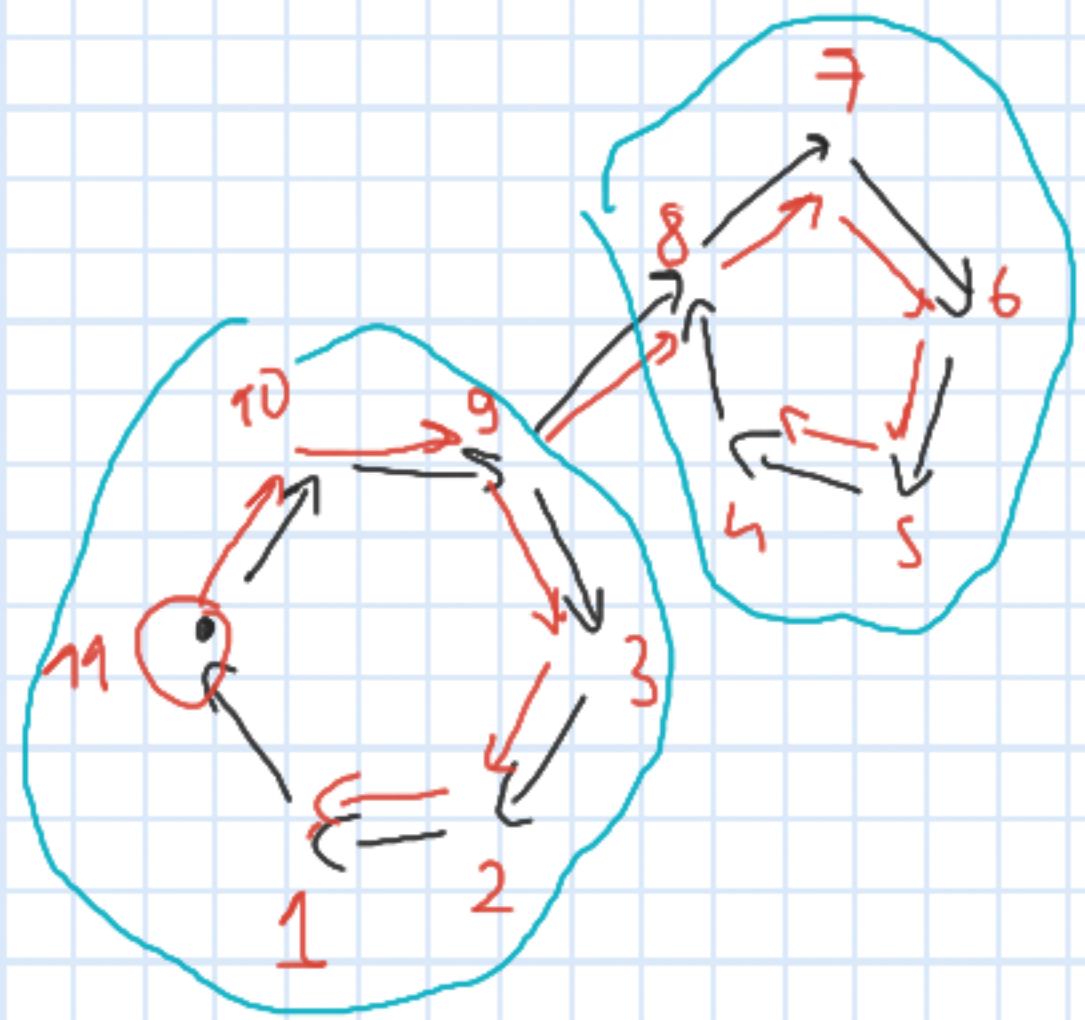
1. Wykonaj DFS na grafie G , zapisując czasę przedowania
2. Odwróci kierunek wszystkich krawędzi
3. Wykonaj DFS, zaczynając stacjonarne wendotki w kolejności malejących czasów przedowania z poprzedniego wykonania

Intuicja





Punkt und
Knoten



Zuordnung

$O(V+E)$, $O(V^2)$

↑
Liste

↑
Matrix

④ Mosty w grafach nieskierowanych

def

Krawędź $e \in$ grafie nieskierowanym G

jeśli mostem jest jej usunięcie powoduje, że graf

staje się rozszczepiony



tu Krawędź e jest mostem utw. gdy
nie leży na żadnym cyklu prostym w grafie

Dowód

e jest mostem \Rightarrow nie leży na cyklu
(bo usunięcie jej usunięcie nie rozszczepia grafu)

e nie leży na żadnym cyklu \Rightarrow e jest mostem

(bo jeśli $e = \{u, v\}$, to jej usunięcie
powoduje, że nie ma szlaku z u do v)

Algorytm

1. Użykuj DFS, dla każdego v zapisz
jego czas odcięcia $d(v)$

2. Dla każdego wierzchołka v oblicz

$$\text{low}(v) = \min \left(\begin{array}{c} d(v), \\ \min_{\substack{u \in \text{wierzchołki \\ sąsiednie do } v}} d(u), \\ \dots \end{array} \right)$$

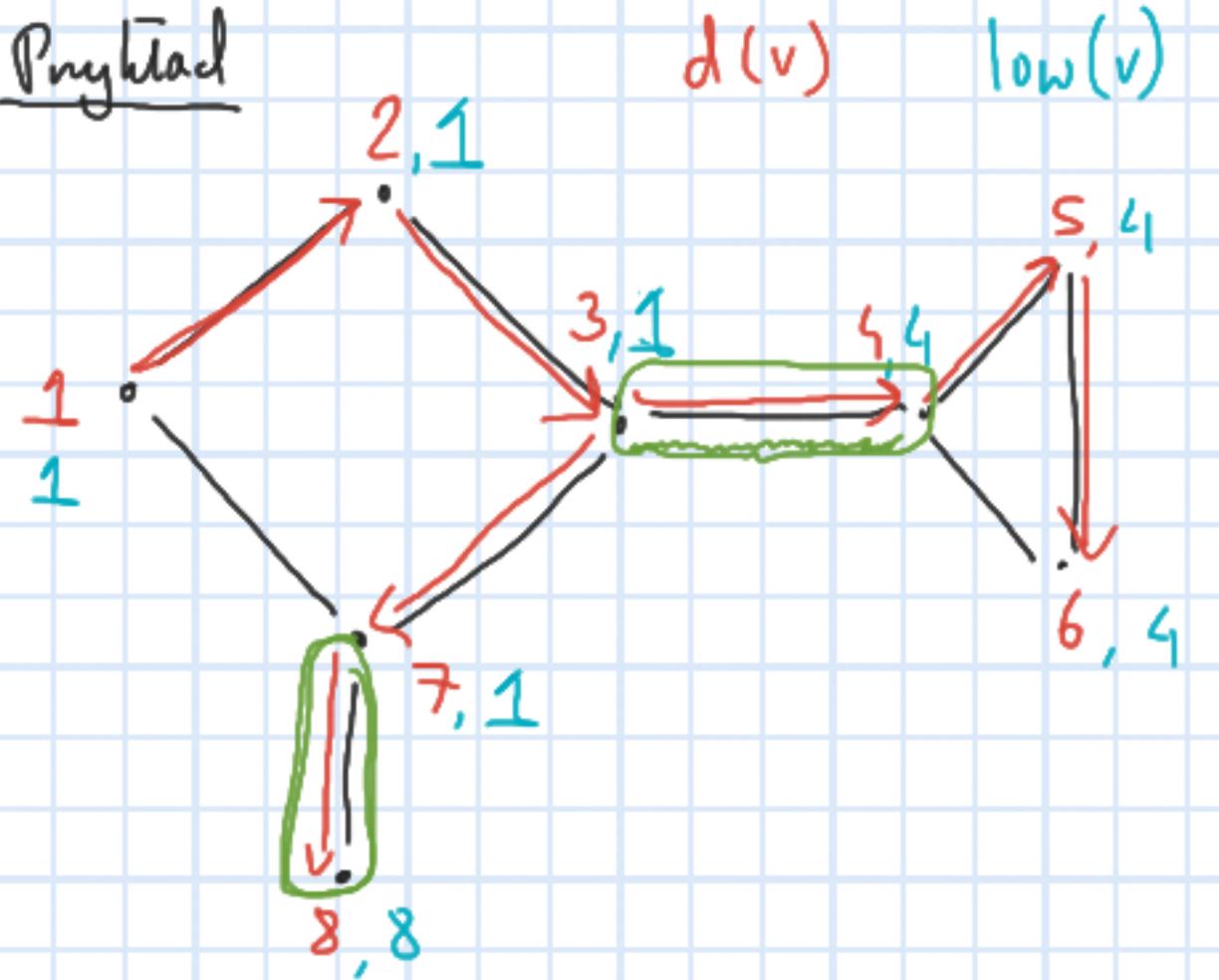
v to dzieci v
 v dwaDFS

3. Mosty to krawędzie $\{v, p(v)\}$

takie gdzie $d(v) = \text{low}(v)$

Idea: $\text{low}(v)$ - identifikator cyklu

Prybilad

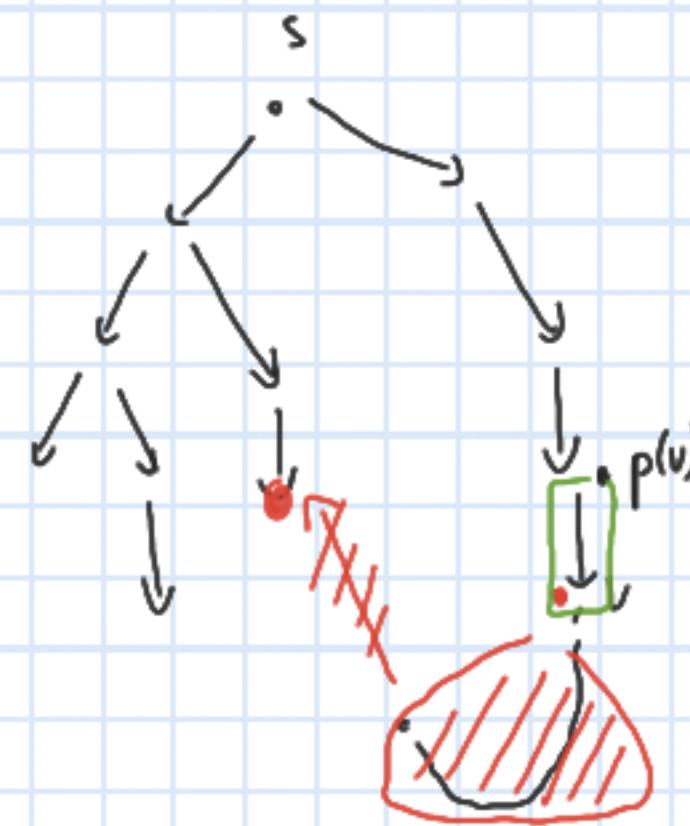


Jeli $d(v) = \text{low}(v)$ to kresydt

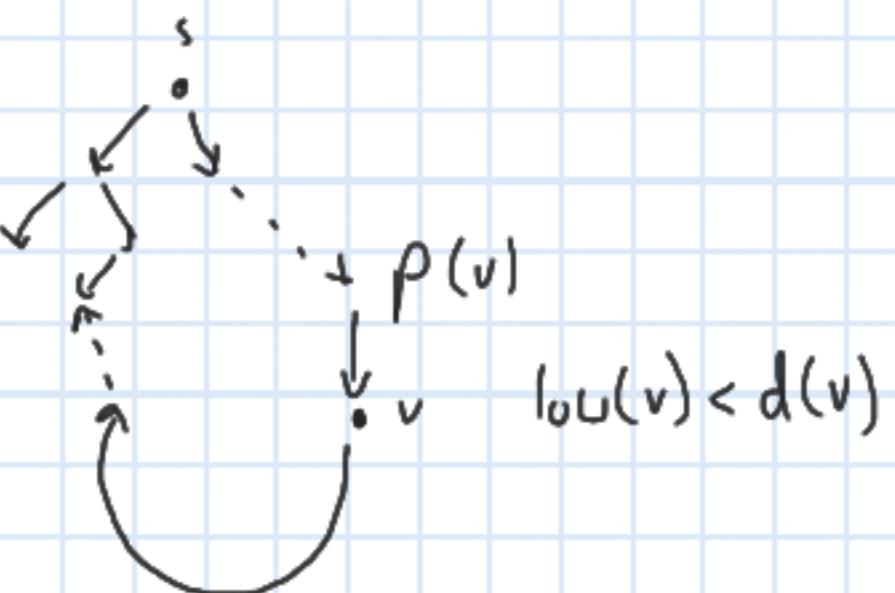
$\{p(v), v\}$ jst mostem

takej kresydt
do $p(v)$ lbo
viendostka
o mizsgu
cznie
odwiedzene
nic ma

$\therefore p(v)$
 $\therefore d(v) = \text{low}(v)$



Jeli $\text{low}(v) < d(v)$ to $\{p(v), v\}$ nic jst mostem



Algorytmy i Struktury Danych

Wykład 8

Reprezentacja grafów ważonych

$$G = (V, E)$$

①

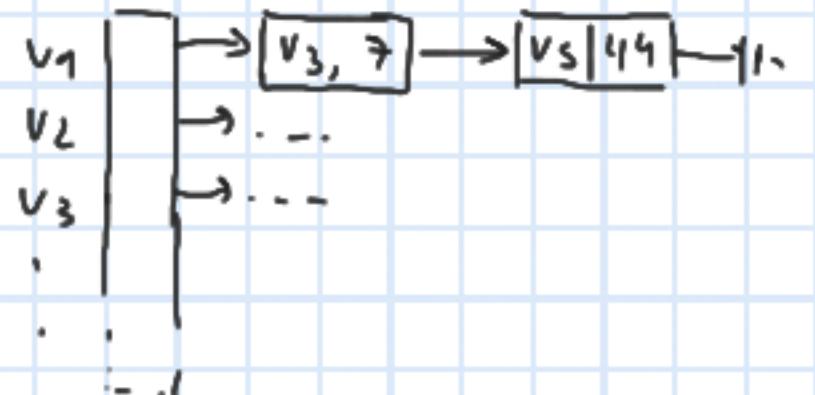
$$w: E \rightarrow \mathbb{N} \quad (\mathbb{Z}, \mathbb{R})$$

Reprezentacja macierzowa

w - matryca wag

$w[i][j]$ - waga krawędzi między
ciennostkami v_i oraz v_j
(∞ jak nie ma)

Reprezentacja listowa

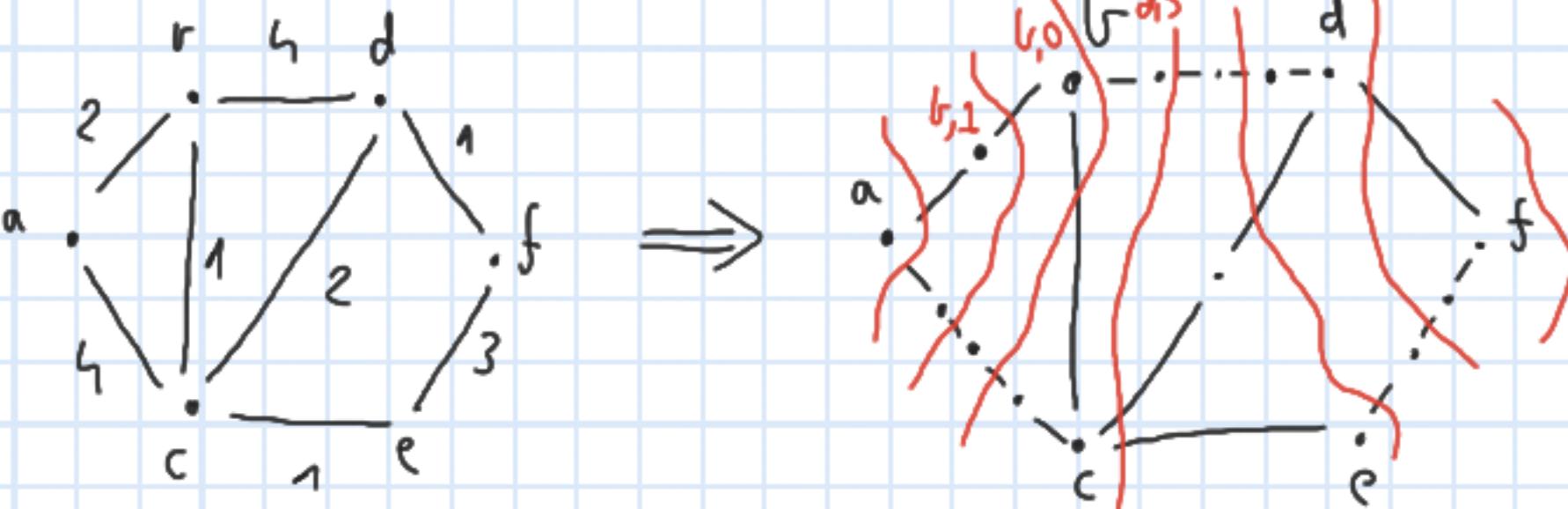


Problem znajdowania najkrótszych ścieżek

- 1 · 1 } na elementarnym poziomie trudne do uklasystania
- 1 - wszyscy } standardowe wersje problemu
- wszyscy - unikalny

Podejście elementarne / BFS

długość/wagi krawędzi to małe liczby naturalne



trzymany są same wendowią odpowiednio dzidele krawędzi

Algorytm Dijkstry - algorytm elementarny,

ale u każdym kroku skacze do najbliższego

predzięnego wierzchołka

{ nie wymagamy rag naturalnych, ale

{ muszą być nieujemne

Nataga

$$G = (V, E)$$

$w(u, v)$ - odległość z u do v

$u.d$ - oszacowanie odległości ze źródła do u

$u.parent$ - poprzednik na najkrótszej ścieżce
ze źródła

$O(E \log V)$

$\rightarrow O(V^2)$

np. macierowa
z "liniową kolejką"

Algorytm

(start z $s \in V$)

1. Umiesią wszystkie wierzchołki w kolejce priorytetowej z oszacowaniem odległości ∞

2. Zmien odległość s na 0

3. Polci wierzchołki są w kolejce

- wyjmij z kolejki wierzchołek u o minimalnej wartości $u.d$

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

dof relax(u, v):

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

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

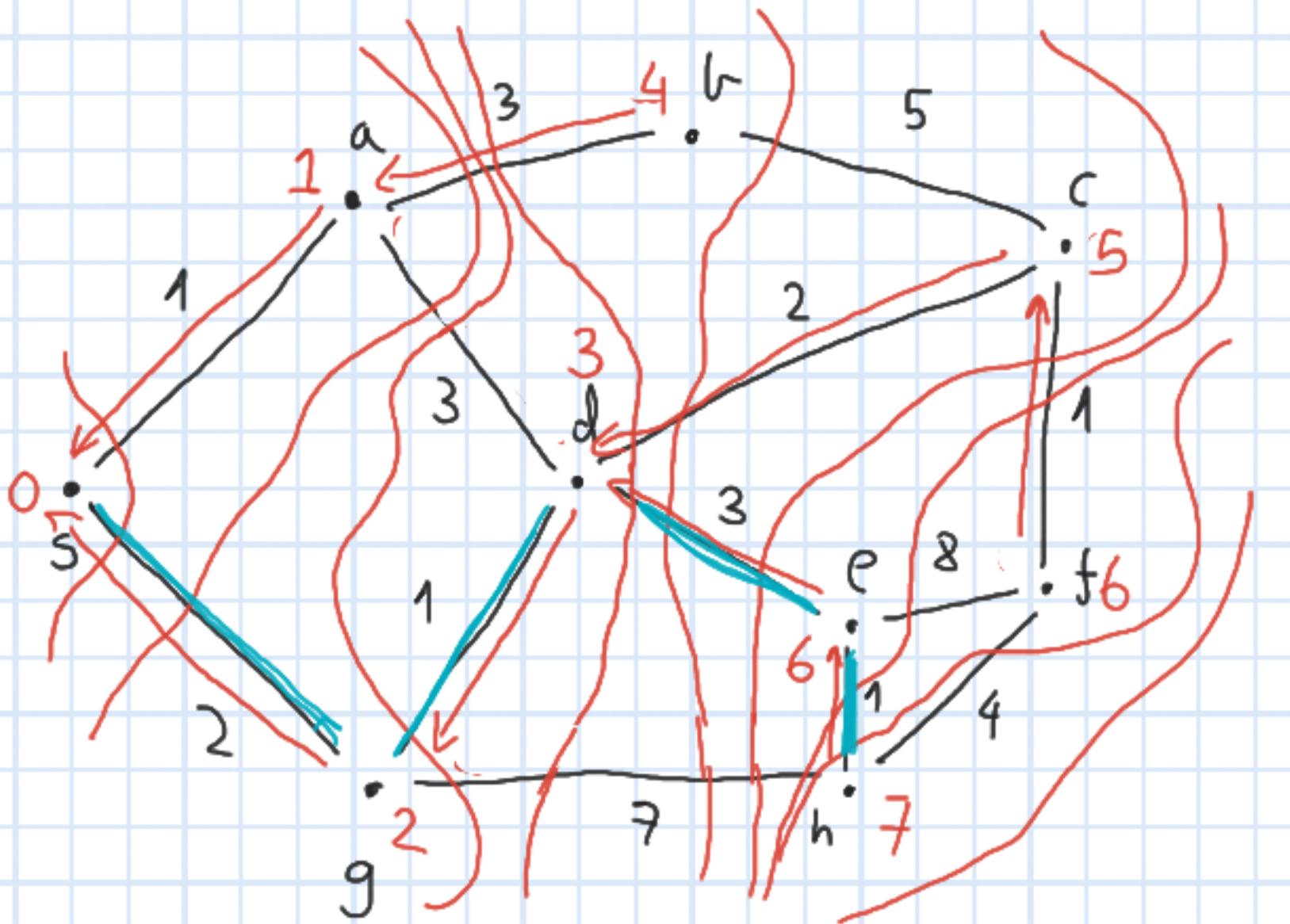
$v.parent = u$

algorytm Dijkstry dla rep. listowej
z kopcem binarnym

} w praktyce
realizowane
inaczej

$\curvearrowleft (u, v)$ - dla grafu skierowanego

Pnyktaad



Długiego algorytm Dijkstry jest poprawny?

- by realizować BFS z dodanymi niendotkami
(nieprawidłowy argument jeśli wagi nie są naturalne)
- dowód indukcyjny

tw Gdy algorytm Dijkstry wyznacza niendotki w

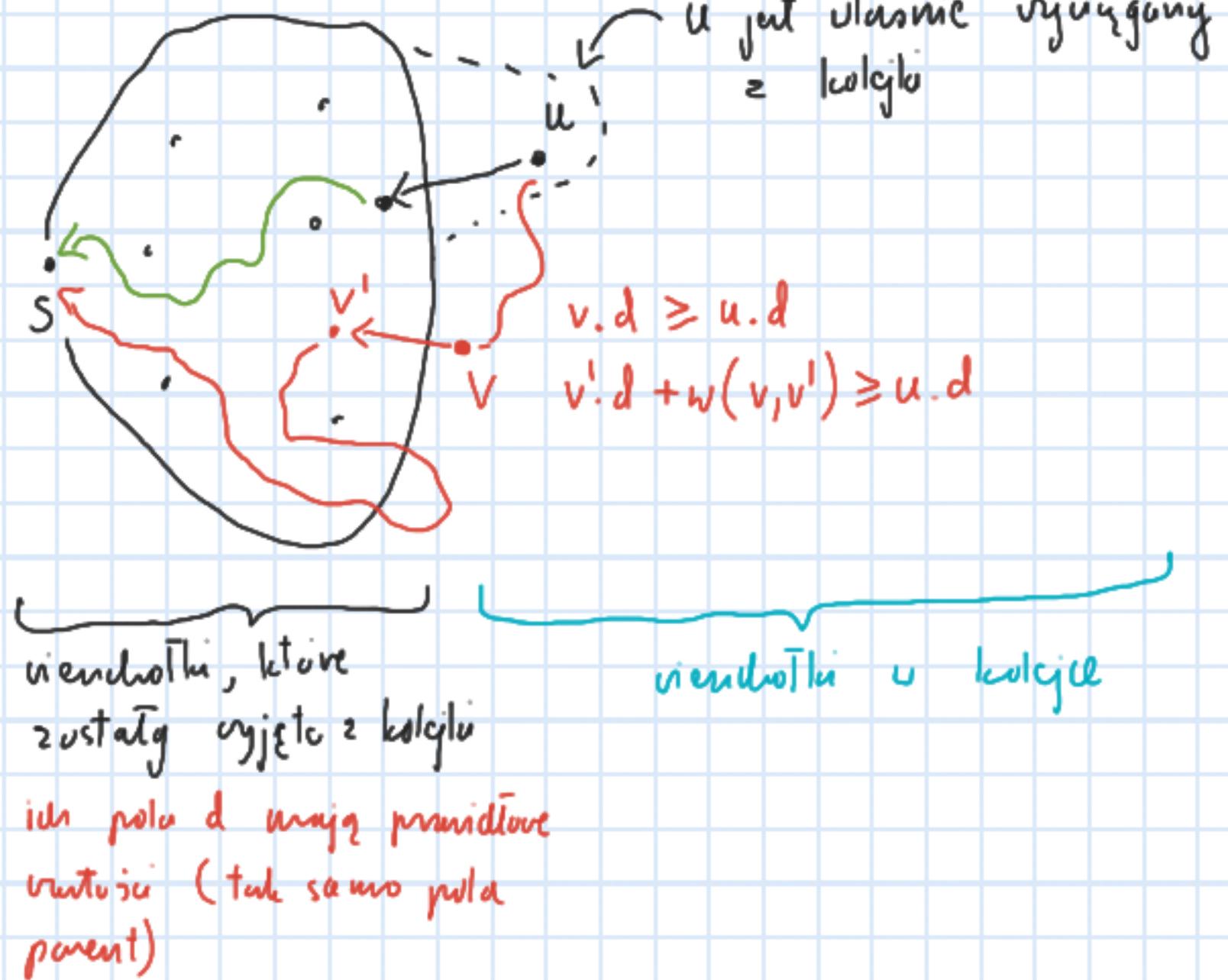
z kolejką, to jego pula $u.d$ zawsze dłuższa

najniższej ścieżki z s do u

Dowód (prz. indukcyj.)

Podstawa indukcji — dla s jest to weryfikowany
sposób mierzenia

Krok indukcyjny



Algorytm Bellmana - Forda

Najkrótsze ścieżki gdy dopuszczamy ujemne wagę

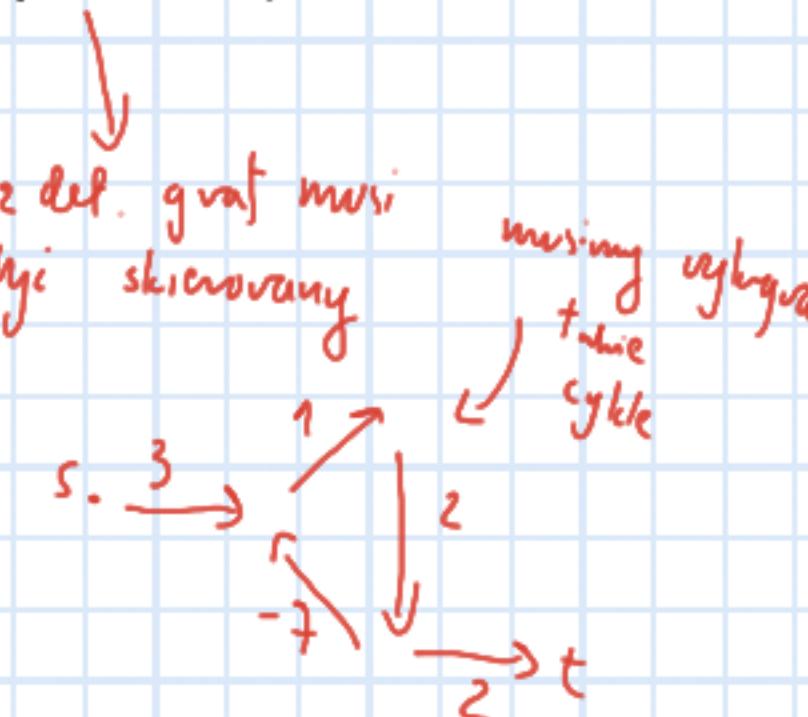
① Inicjalizacja

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)

$\boxed{O(|V||E|)}$

③ Wyfiltracja

wy dla każdego $(u, v) \in E$:

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

Jśli weryfikacja tego nie jest ujemna to dla każdego i :

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

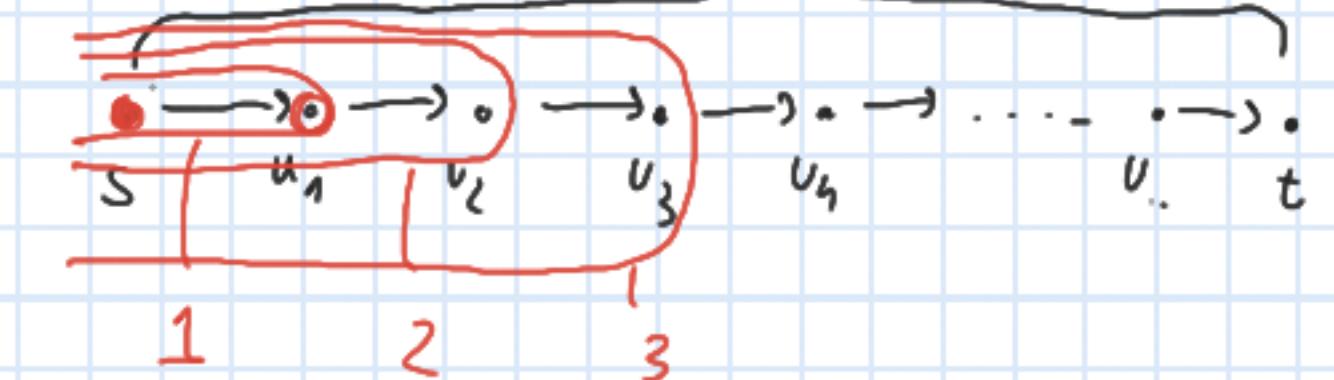
Po zsumowaniu tych nierówności:

$$\sum_{i=1}^k v_i.d \leq$$

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

Czyli krok ① + ② daje dobry wynik, jeśli nie ma cykli o ujemnej wagie?

najkrótsza ścieżka z s do t



Kiedy iteracja relaksuje w najmniej jednego kroku kraję najkrótszej ścieżki

Czyli ③ wykrywa ujemne cykle

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

$$\left\{ \begin{array}{l} v_k = v_0 \\ \dots \end{array} \right.$$

Spłaszczenie

Najkrótsze ścieżki między każdą parą wierzchołków

- $|V|$ wyciągnięcie Dijkstry $O(VE \log V)$

- $|V|$ wyciągnięcie Bellmana-Forda $O(V^2 E)$

Istnieje specjalizowany algorytm Floyda - Warshalla

który znajduje rozwiązanie w czasie $O(V^3)$

Algorytmy i Struktury Danych

Wykład 9

Największe skutki międzykandydu-
pung wieńczołków

Konvenya

W specjalizowanych algorytmach
stosuje się reprezentację macierzy

$D[u][v]$ - dt. najmłodszy succ.
z u do v

$P[u][v]$ - "parent" – výchozík
tři před v na nejkratší
cestě z u do v

Algorytm Floyd - Warshalla

Idea: Jeli znamy najwotrze siedlisko
migru kolidz poniż cienchotkami
które jako ujemne nerchotki

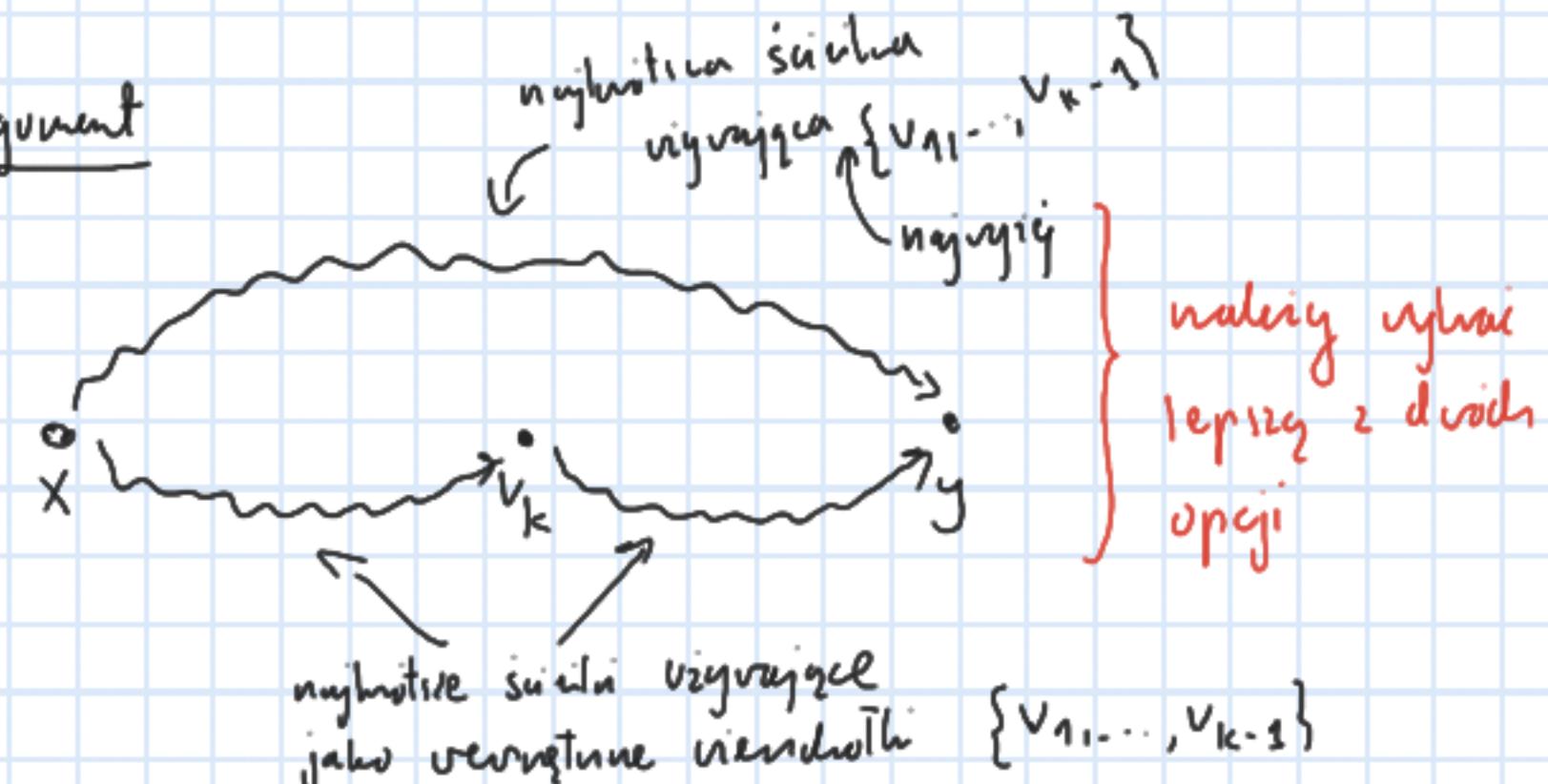
viv viv

$$\{v_1, \dots, v_{k-1}\}$$

to wiec my osiągnąć to samo dla

$$\{v_1, \dots, v_k\}$$

Argument



Konwencja

$$V = \{v_1, \dots, v_n\}$$

$S^{(t)}$ - macierz długoci najkrótszych ścieżek opartych o węzły tne
wierzchołki ze zbioru $\{v_1, \dots, v_t\}$

$S^{(0)}$ - macierz wag krawędzi między
wierzchołkami
 $(\infty$ oznacza brak krawędzi)

$$P^{(0)}[x][x] = \text{None}$$

$$P^{(0)}[x][y] = \begin{cases} x - jaki waga krawędzi z x do y \\ \text{None} - w p.p. \end{cases}$$

założonośc
 $\Theta(n^3)$

Algorytm

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

; for $x \in V$:

; ; for $y \in V$:

$$S^{(t)}[x][y] = \min(S^{(t-1)}[x][y], S^{(t-1)}[x][v_t] + S^{(t-1)}[v_t][y])$$

$$D = S^{(n)}$$

uzupełnianie macy P

$$\left\{ \begin{array}{l} P^{(t)}[x][y] = P^{(t-1)}[x][y] \\ P^{(t)}[x][y] = P^{(t-1)}[v_t][y] \end{array} \right.$$

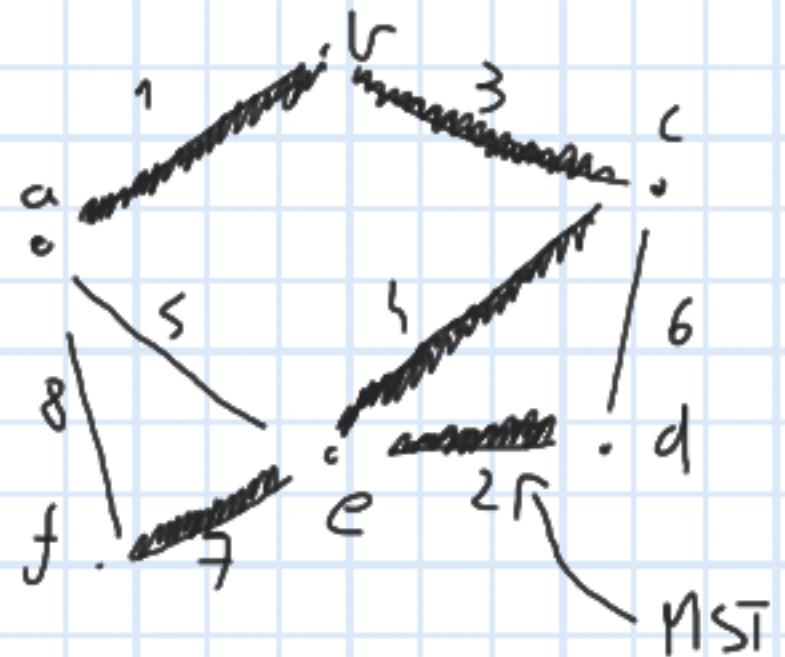
U implementacjach
wykonamy tylko
jednej macy

Minimalne drzewa rozpinające (minimal spanning trees, MST)

$G = (V, E)$ - graf niesklawowany ← spójny

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

Cel: znaleźć podzbiór krawędzi tworzący spójny podgraf (obejmujący wszystkie wierzchołki), o minimalnej sumie wag



Obszary

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

Jśli $A \subseteq E$ jest podzbiorem krawędzi pewnego MST dla G oraz $e = \{u, v\}$ jest krawędzią, taką że:

a) $e \notin A$

b) $A \cup \{e\}$ nie zawiera cyklu

c) e ma minimalną wagę wśród krawędzi spajających parzyste wierzchołki

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

$$\text{Twierdzenie } T = (T' - \{e'\}) \cup \{e\}$$

$$w(e') \geq w(e)$$

$$w(T) \leq w(T'), \text{ więc } T \text{ to MST}$$

Dowód



Jśli $e = \{u, v\}$ n.c należy do żadnego MST zaczynającego się w A , to istnieje inna krawędź, stojąca do (spur A)

zapierająca trasę v MST rozpięającą A
z u do v

Niedł. to będąc $e' = \{x, y\}$

T' - rozpięjące A :



Algorytm Kruskala dla MST

1. Posortuj krawędzie po wadze

2. $A = \emptyset$

3. Pręgadaj krawędzie w kolejności
niespołigowych wag

- jeśli $A \cup \{e\}$ nie tworzy
cyklu to

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

4. Zwróć A

$O(E \log V)$ - złożoność czasowa
(o ile szybko wykrywamy
krawędzie tworzące cykl)

Algorytm Prima dla MST

v - wierzchołek startowy



1. Umieścimylistycie wierzchołki

w kolejce priorytetowej z wagą ∞

2. Zmieni wagę v na 0

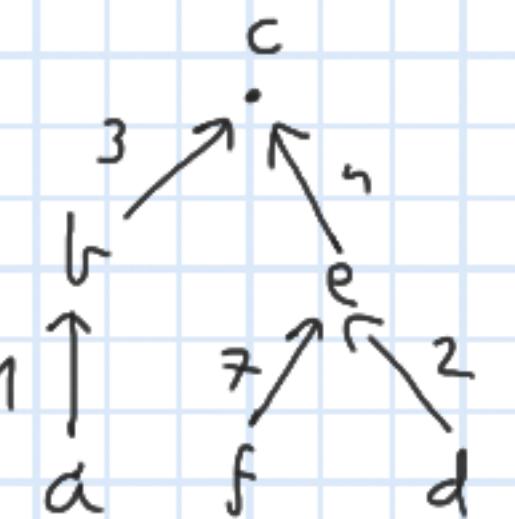
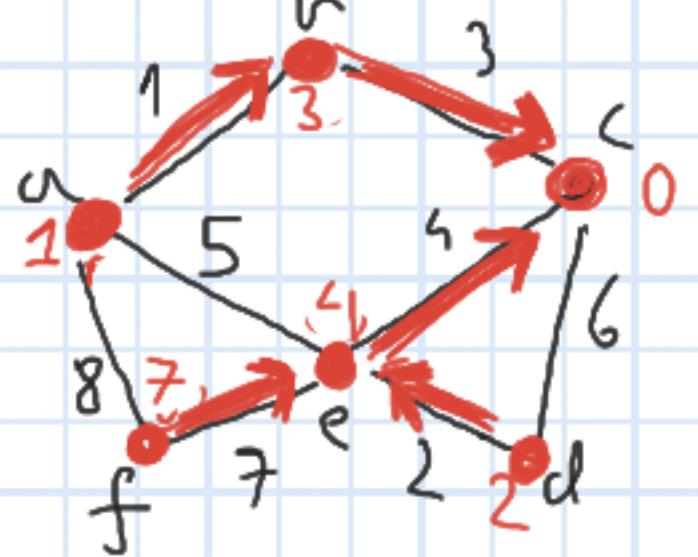
3. Później wierzchołki są w kolejce:

- wybierz wierzchołek u o minimalnej wadze z
kolejki

- dla każdej krawędzi $\{u, x\}$, jeśli
waga $w(\{u, x\}) < \text{waga } x \text{ w kolejce, to}$
zmień wagę x na $w(\{u, x\})$
(aktualizuj parent)

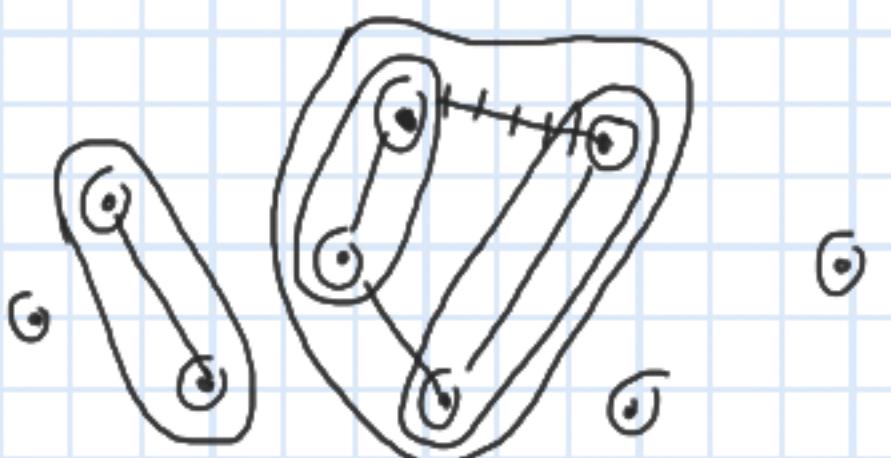
$O(E \log V)$

Pozycja

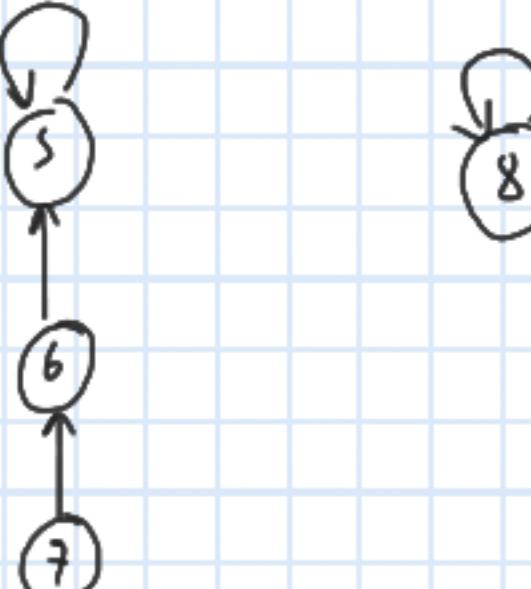
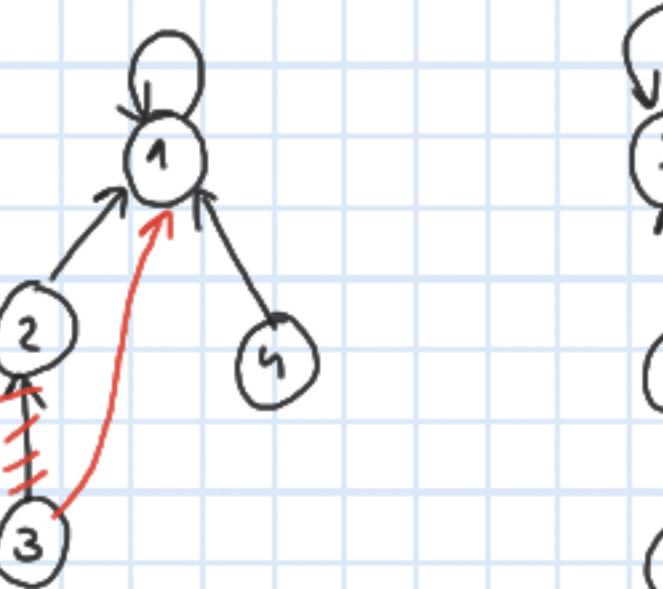


Rodzina zlioniu roztagonych - struktura
Find/Union

Idea



Struktura Find/Union oparta o las zlioniu roztagonych



Operacja Find: Wędrujemy w gory dnia do konenia
i jego zwracamy jako identyfikator zlioniu
+ kompresja scialki

Operacja Union: dotagiamy koneni jednego dnia do
drugiego

z kaidym zlioniem tzw. "range"
i dotagiamy dnia o mniejszej rangie do
tego o wiekszej (jeli rangi wieksze, to obaj te
knie dotagamy, ale rangi mniejsze o 1)

```
class Node  
    def __init__(self, value)  
        self.parent = None  
        self.rank = 0  
        self.value = value
```

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

Jeli stosujemy tzw. emisie wg. rangu
over kompresji ścieżki to ciąg
w operacji ma złożoność $O(m \log^* m)$

gdrīc

$\log^+(m)$ to líně závislý logický výraz, který je rovno 1, když má argument m větší než 1, a je rovno 0, když má argument m menší než 1.

$$\log(\log(\log(m))) \leq 1$$

```

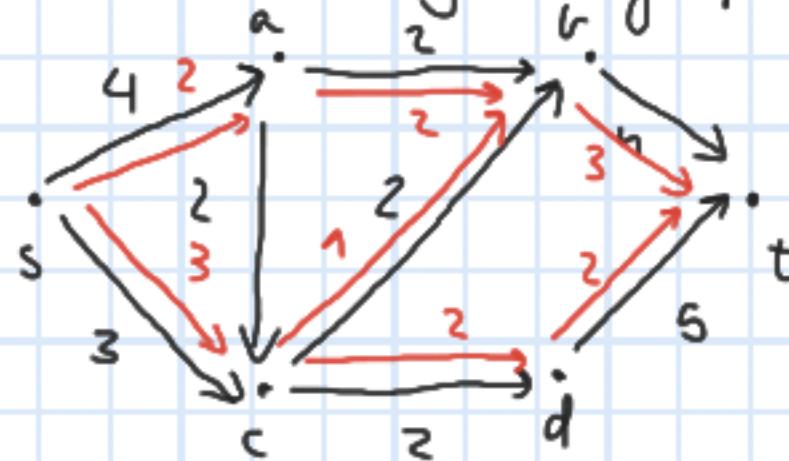
ef union( x,y ) :
    x = findset( x )
    y = findset( y )
    if x.rank > y.rank :
        y.parent = x
    else :
        x.parent = y
    if x.rank == y.rank :
        y.rank += 1

```

Algorytmy i Struktury Danych

Wykład 10

Problem maksymalnego przepływu



z s tylko kierowane ujemne
do t tylko kierowane ujemne

Zadanie

Znaleźć "przepływ" o maksymalnym wartości

Wejście: $G = (V, E)$ - graf skierowany

(dla każdych $u, v \in V$, co najwyżej jedna z $(u, v), (v, u)$ istnieje)

$s, t \in V$ - źródło i węzeł

$c: V \times V \rightarrow \mathbb{N}$ - funkcja pojemności

$c(u, v) = 0$ jeśli $(u, v) \notin E$

$c(u, u) = 0$ - w grafie nie ma self-loop

Przepływ

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

f spłnia następujące warunki

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

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

Wartość przepływu

z def. 0

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

Sieci residualne

$G = (V, E)$ } sieci przepływu
 $s, t \in V$
 $c: V \times V \rightarrow \mathbb{N}$
 $f: V \times V \rightarrow \mathbb{N}$ } przepływ

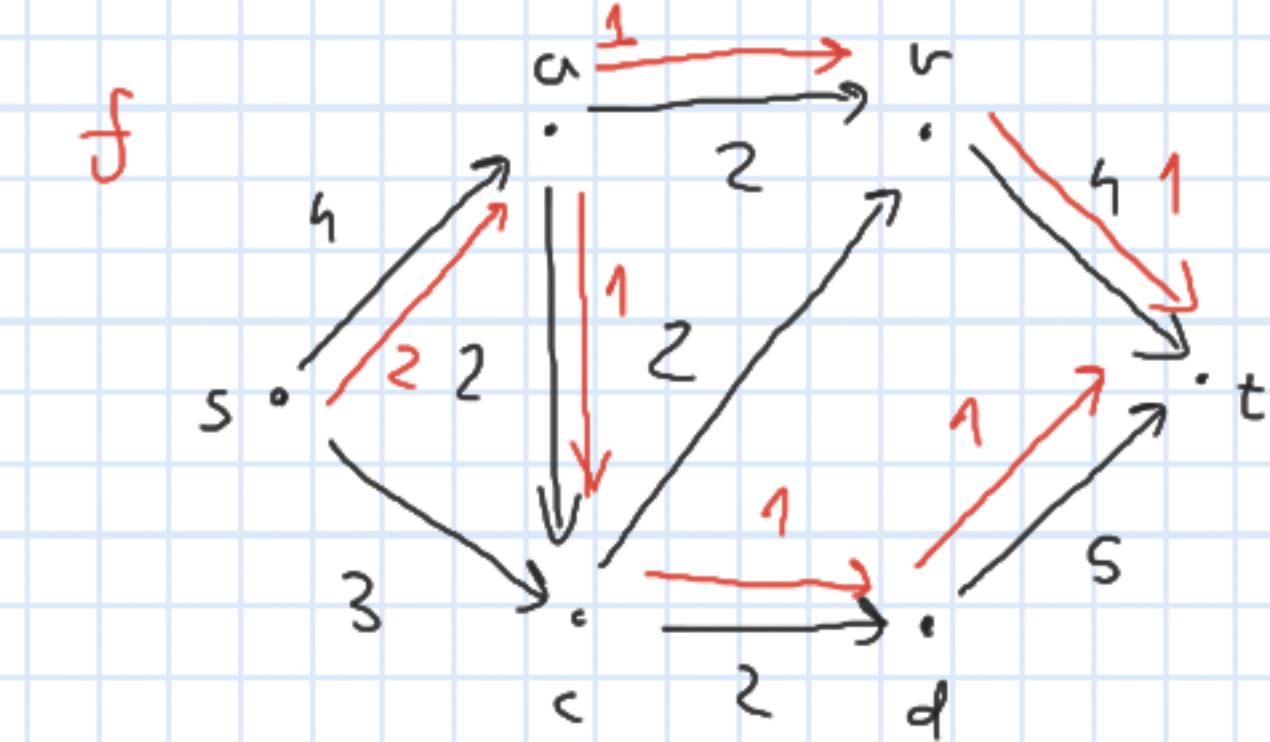
Definiujemy sieci residualne G_f, c_f

nowa funkcja $c_f: V \times V \rightarrow \mathbb{N}$

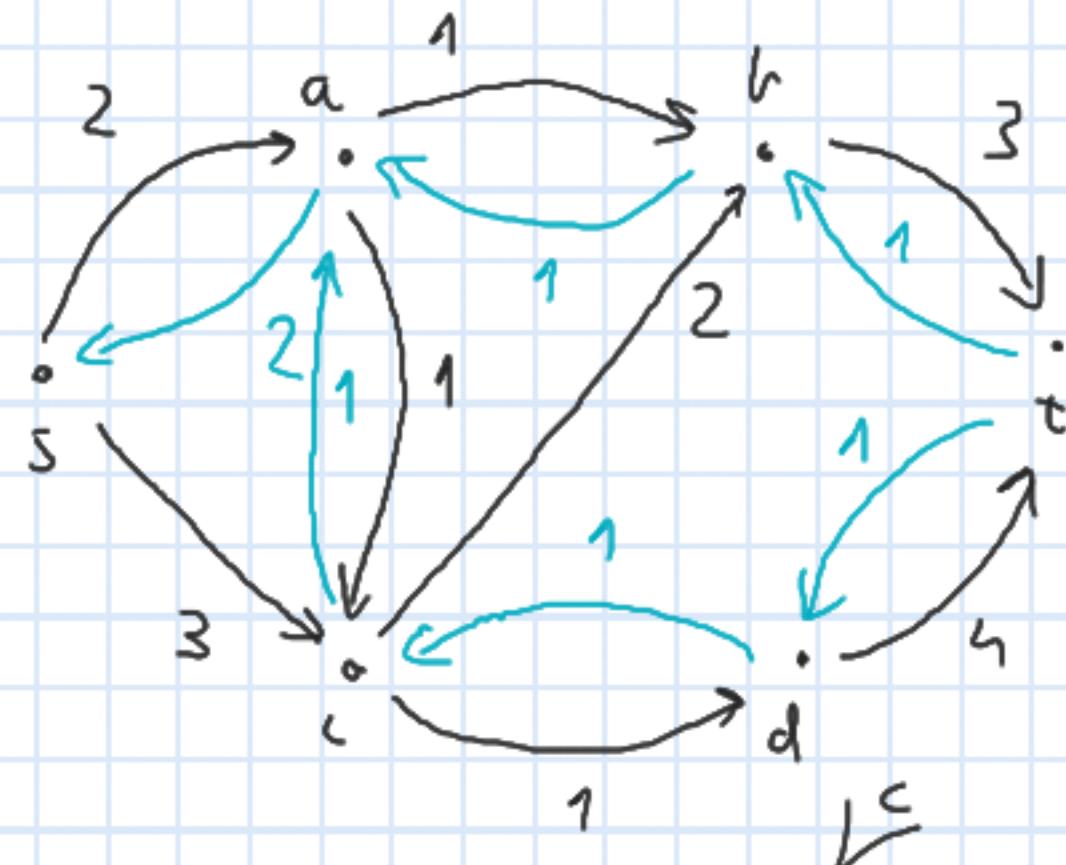
$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v), & (u, v) \in E \\ f(v, u), & (v, u) \in E \\ 0, & \text{w.p.p.} \end{cases}$$

$\| G_f = G$

uzdługi sieci przenoszącej
mocnejszy przepływ
o wartości tej sieci



↓ sieci residualne



Sieciowa przenosząca dla G, f to sieci z s dot w G_f, c_f
 Wartość sieci przenoszącej jest minimalna wartość $c_f(u, v)$ na

Metoda Forda - Fulkersona

1. Jeśli istnieje siatka pośrednicząca dla G, f to przepływ f uzdrowi siatki

2. Wysz do kroku 1

Czy ten algorytm jest poprawny?
(czy mamy upadek w lokalne maksimum?)

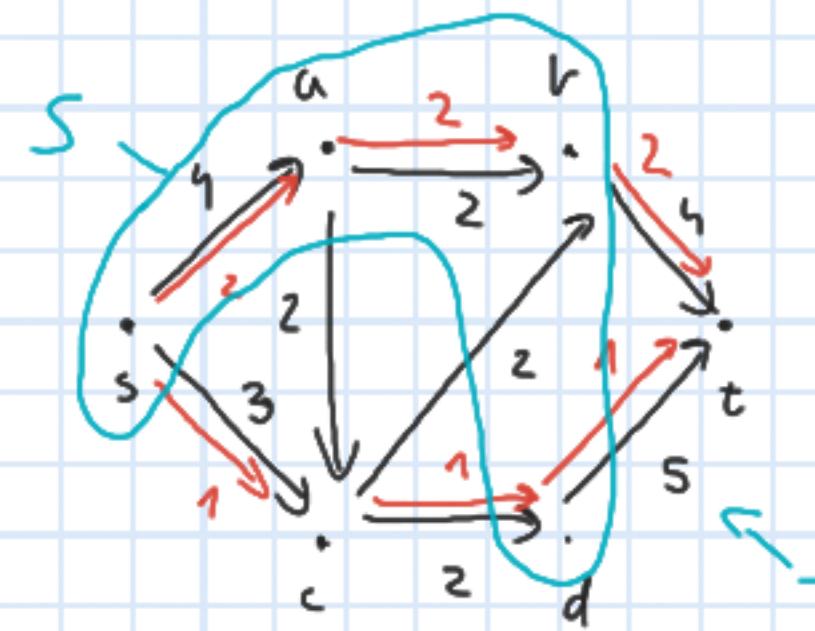
Pniewoj w sieci

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

Pniewoj sieci to podzielić V na dwa zbiorów

$$S, \bar{T} = V - S$$

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



Pnepustowoci pniewoju

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

$$c(S, \bar{T}) = 3 + 2 + 4 + 5 = 14$$

Przepływu netto pn. pniewoju

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

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

$$f(S, \bar{T}) = 1 + 2 + 1 - 1 = 3$$

Lemat

Niech f będzie przepływem w sieci G ze źródłem s , ujemnym t . gdyż (S, \bar{T}) jest przedwojcem. Wówczas $f(S, \bar{T}) = |f|$

Dowód

z warunku zachowania przepływu:

$$(\forall u \in V - \{s, t\}) \left[\sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) = 0 \right]$$

Wantyć przepływ to:

$$\begin{aligned}
 |f| &= \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) \\
 &+ \sum_{u \in S - \{s\}} \sum_{v \in V} f(u, v) - \sum_{u \in S - \{s\}} \sum_{v \in V} f(v, u) \\
 &= \sum_{u \in S} \sum_{v \in V} f(u, v) - \sum_{u \in S} \sum_{v \in V} f(v, u) \\
 &= \left[\sum_{u \in S} \sum_{v \in \bar{T}} f(u, v) \right] - \left[\sum_{u \in S} \sum_{v \in T} f(v, u) \right] = f(s, \bar{T}) \\
 &= 0 \leftarrow \left[+ \sum_{u \in S} \sum_{v \in S} f(u, v) - \sum_{u \in S} \sum_{v \in S} f(v, u) \right] \\
 &= f(S, \bar{T})
 \end{aligned}$$

Lemat

Umiestoje prieplývu jut nereikia nis
prieplūstovuji daryvneyo priepluoju

Dowiad

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

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

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

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

Tuendnic (o maksymalnym przepływie i minimalnym przekroju
max-flow/min-cut theorem)

(2) \Rightarrow (3)

Niech $G = (V, E)$, $s, t \in V$, $c: V \times V$ będzie sieć

przepływu : f będzie przepływem v tyle samo

Następujące warunki są równoważne :

- (1) f jest maksymalnym przepływem
- (2) G_f, c_f nie zawiera dodatkowych przekresek
- (3) dla każdego przekroju S, \bar{S} zachodzi
 $|f| = c(S, \bar{S})$

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

$\{ s \in S, \quad t \notin S$

$$\bar{T} = V - S$$

$| t \in \bar{T}$

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

Dowód

(3) \Rightarrow (1) – natychmiastowe z powyższego lematu

(1) \Rightarrow (2) – oznacza, że innych możliwych przekresek niż maksymalny przepływ i nikt były maksymalny

Jśli $(u, v) \in E$ to $c_f(u, v) = 0 \Rightarrow f(u, v) = c(u, v)$

Jśli $(v, u) \in E$ to $c_f(u, v) = 0 \Rightarrow f(v, u) = 0$

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

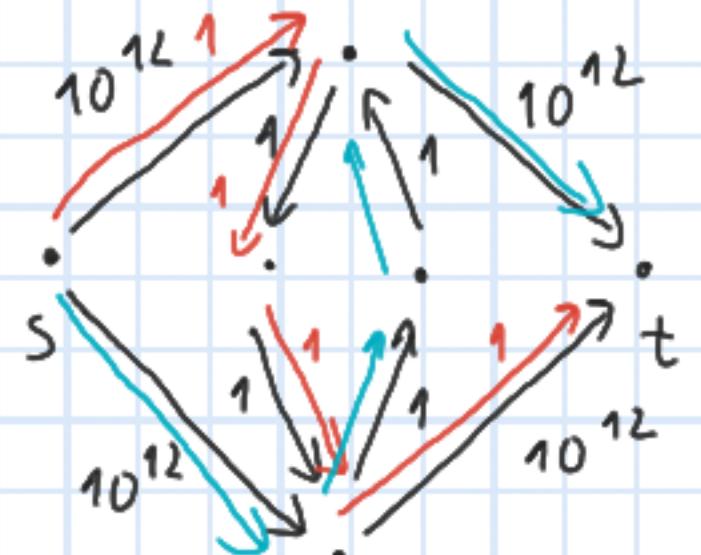
Złożoność metody Forda - Fulkersona

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

zī. szukania
światła powiększających

wartosci maks.
przepływu

To mnoże się bardzo dużo!

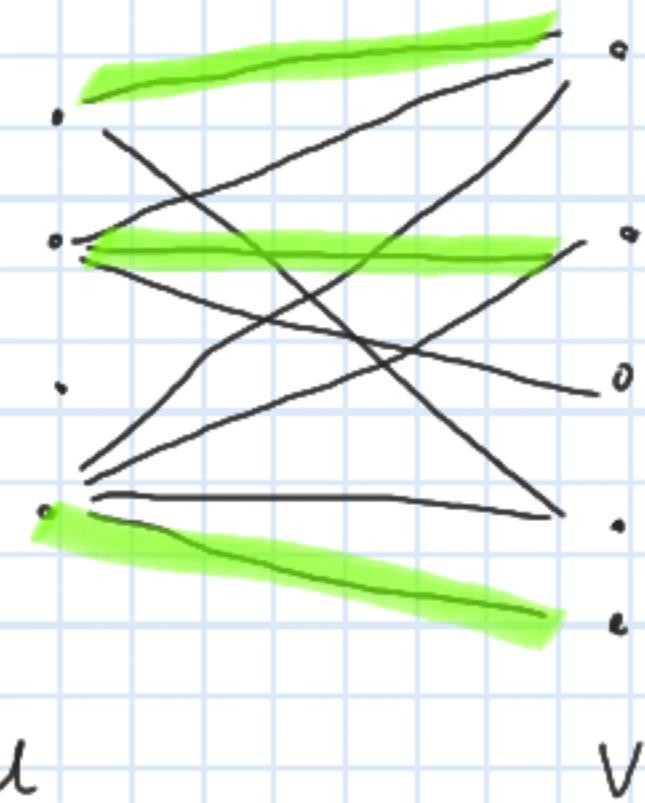


Jesli zawsze światła powiększające znajdują się w algorytmie BFS, to złożoność wynosi

$$O(VE^2)$$

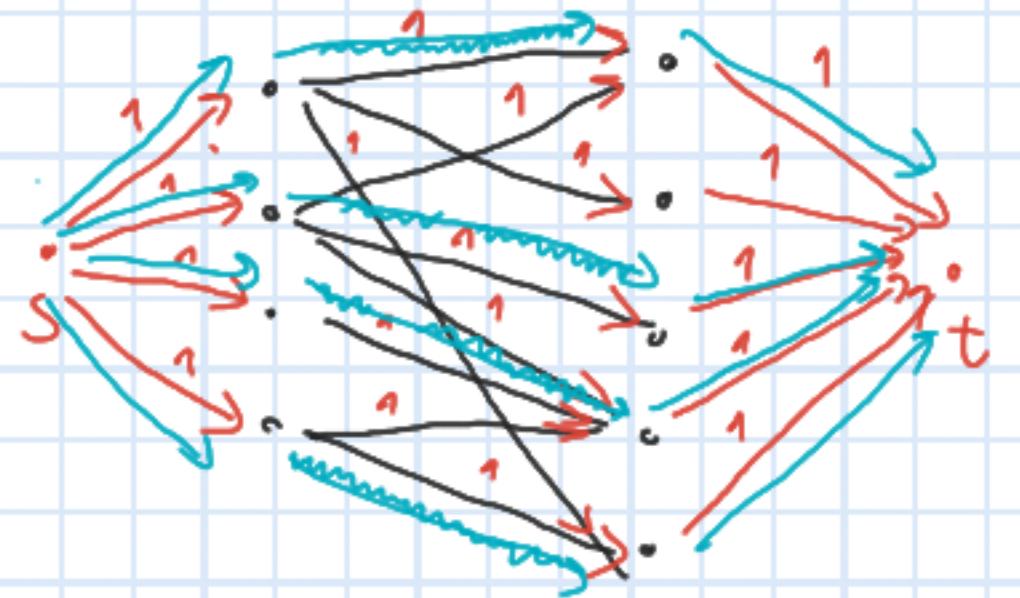
Skojarzenia w grafach dwudziestych

Graf dwudziesty - graf którego zbiór wierzchołków podzielony na dwie części U, V, takie że każda krawędź łączy jeden wierzchołek z U i jeden z V



Skojarzenie to zbiór krawędzi, które nie mają wspólnego wierzchołka

Najlepsze skojarzenie znajdujące się
w obrębie maksymalnego przedziału:



$$c(u_i, v_j) = 1$$

Złożoność
 $O(\epsilon v) = O(V^3)$

Algorytmy i Struktury Danych

Wykład 11

Metody konstruowania algorytmów

- dziel i zzygaj (QuickSort / MergeSort)

- zadbane (alg. Kruskala)

- programowanie dynamiczne

metoda zaminiany cykliczniego

algorytmu rekurencyjnego na

udomianowy iteracyjny

Punkty elementarne

$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, n \geq 2$$

$$1, 1, 2, 3, 5, 8, 13, 21, \dots$$

$$F_n = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$

Oblinianie n-tej liczby Fibonacciego algorytmem dynamicznym

- tworzymy tablicę, w której zapamiętujemy wyniki wcześniejsze
i z nich korzystamy zamiast rekurencji.

def fib(n):

$$F = [1] * (n+1)$$

for i in range(2, n+1):

$$F[i] = F[i-1] + F[i-2]$$

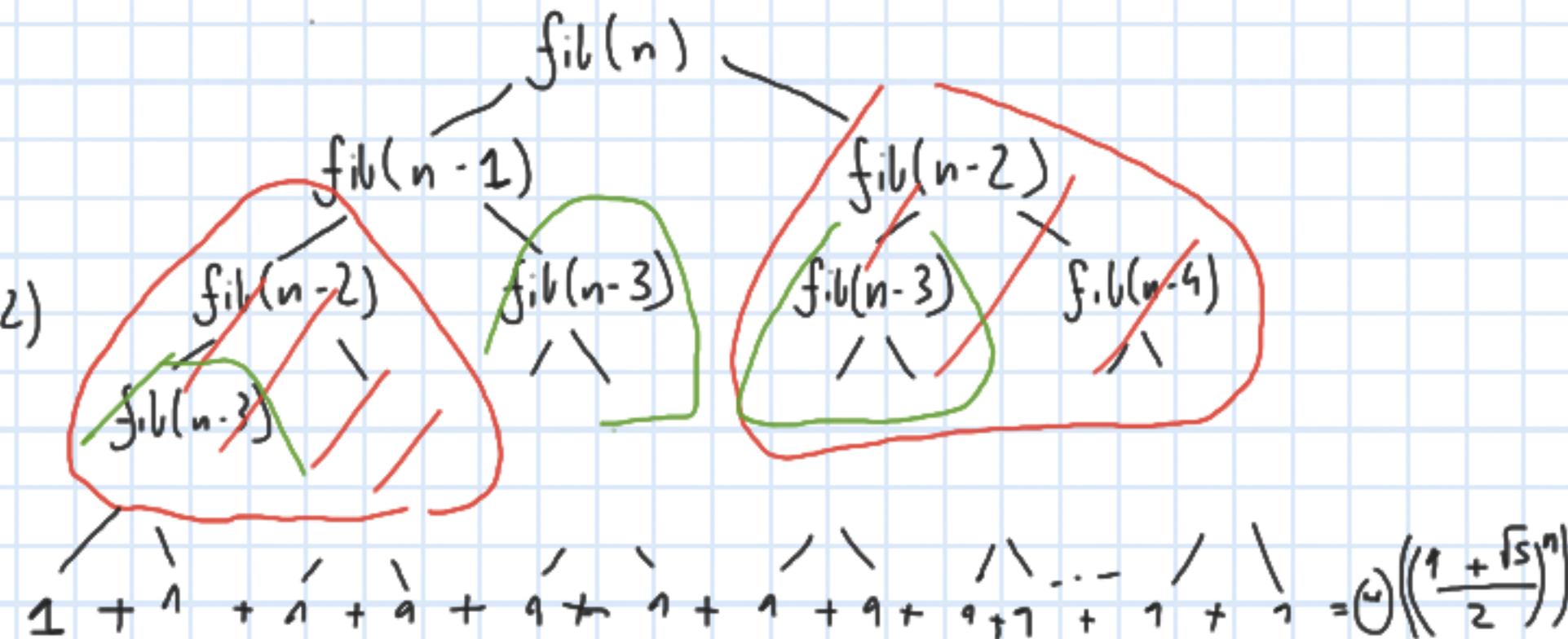
return F[n]

$O(n)$

def fib(n):

if n < 2: return 1

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



Najdłuzszy rosnący podciąg

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

Zadanie: Chcemy znaleźć najdłuższy (niekoniecznie spójny) podciąg A

0	1	2	3	4	5	6	7	8	9
2	1	4	3	1	5	2	7	8	3
f	1	1	2	2	1	3	2	4	5
P	-1	-1	1	1	-1	2	4	5	7

① Stworzenie funkcji, liczącej obliczamy

~~$f(i)$ = długość najdłuższego podciągu
linii $A[0], \dots, A[i]$~~

~~$f(i)$ = długość najdłuższego podciągu
lącącego się na $A[:i]$~~

~~def ps(A, P, i):~~

~~if P[i] ≠ -1:~~

~~ps(A, P, P[i])~~

~~print(A[i])~~

② Zapisanie funkcji w postaci rekurencyjnej

$$f(0) = 1 \quad f(-x) = 0 \quad A[-1] = -\infty$$

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

$\max_i f(i)$ — długość najdłuższego ciągu rosnącego

③ Implementacja

def lis(A): $P = [-1] * n$

$n = \text{len}(A)$

$F = [1] * n$

for i in range(1, n):

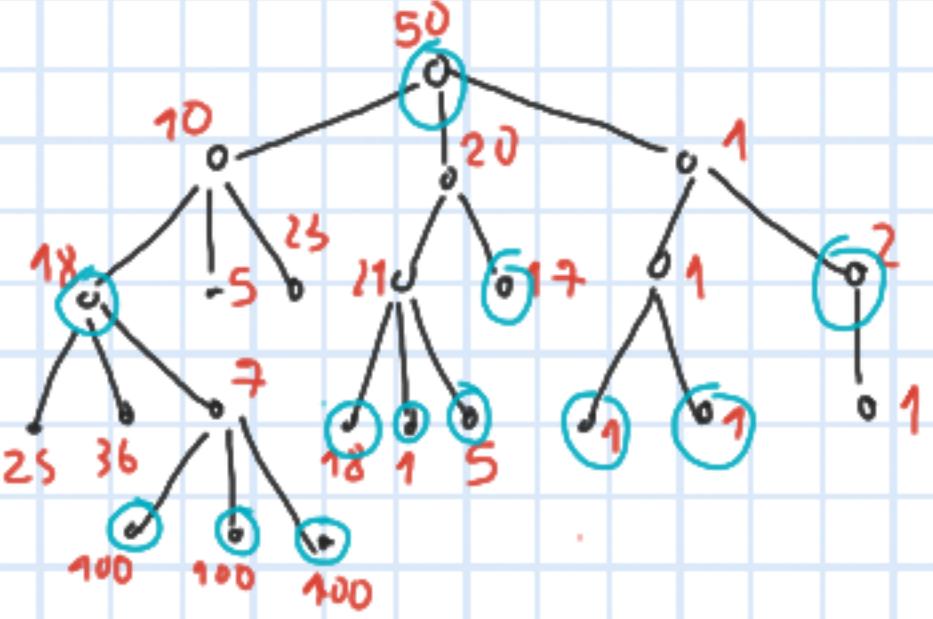
for j in range(i):

if $A[j] < A[i]$ and $F[j] + 1 > F[i]:$

$F[i] = F[j] + 1$ $P[i] = j$

return $\max(F), P$

Problem imprezy firmowej



Chcemy wyhalać wiele kroków o maksymalnej sumie, takie że żadne dwa nie są powiązane krawędzią

```
class Employee:
    def __init__(self, fun):
        self.emp = []
        self.fun = fun
        self.f = -1
        self.g = -1
```

① Określenie funkcji, które wydzielamy obliczanie

$f(v)$ = wartość najlepszej imprezy w poddaniu zakończonym w v

uprzed
dawnego

$g(v)$ = wartość najlepszej imprezy w poddaniu zakończonym w v , o ile v nie idzie na tę imprezę

② Zapis rekurencyjny

$$f(v) = \max \left(v.fun + \sum_{u_i - \text{dzieci} v} g(u_i), g(v) \right)$$

$$g(v) = \sum_{u_i - \text{dzieci} v} f(u_i)$$

③ Implementacja

```
def f(v):
    if v.f >= 0:
        return v.f
    x = v.fun
    for ui in v.emp:
        x += g(ui)
    y = g(v)
    v.f = max(x, y)
    return v.f
```

def g(v):

if v.g >= 0:
 return v.g

x = 0

for ui in v.emp:

x += f(ui)

v.g = x

return v.g

Algorytmy i Struktury Danych

Wykład 12

Problem plecakowy (ang. Knapsack)

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

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

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

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

Zadanie: Znaleźć przedmioty przedmiotów, których

Tągma waga nie przekracza B i których

Tągma cena jest maksymalna

① Funkcja do obliczania

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

② Sformułowanie rekurencyjne

nie liczymy

$$f(i, b) = \max\left(f(i-1, b), f(i-1, b - w(i)) + p(i)\right)$$

liczymy i ty przedmiot

≥ 0

(a gdyby wynio < 0
to pomijamy ten wybór)

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

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 ):
```

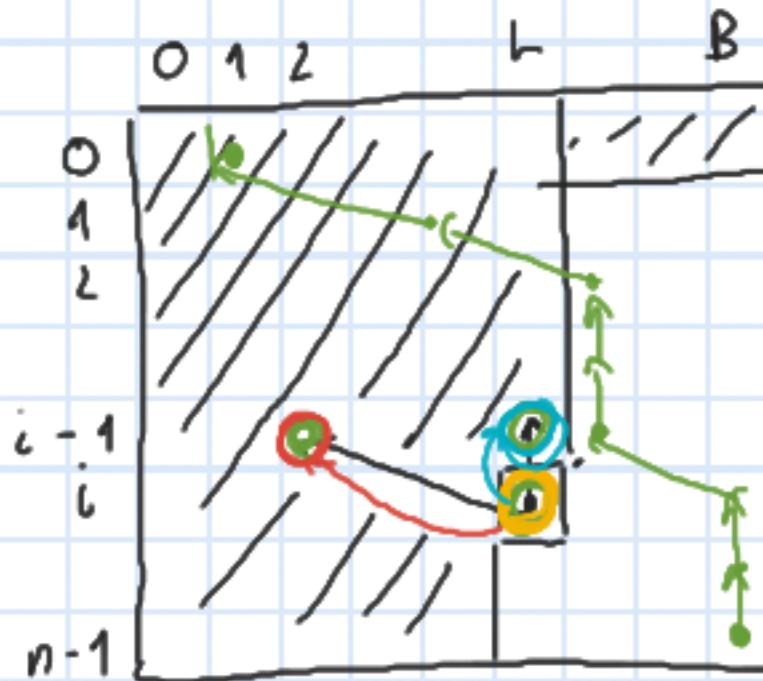
```
            F[i][b] = F[i-1][b]
```

```
            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]
```

Odtwarzanie rozwiązań
P - parent

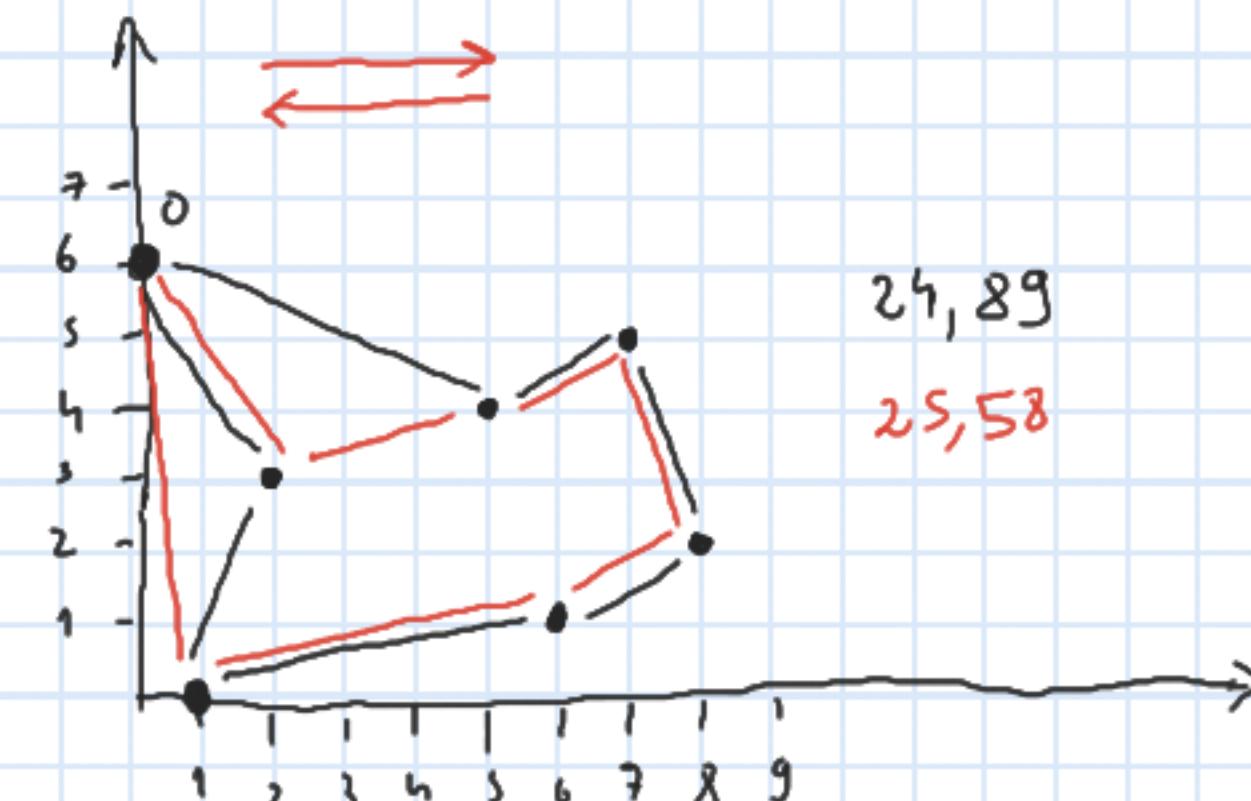


Problem komiwojażera (travelling salesperson problem, TSP)

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

$d: C \times C \rightarrow \mathbb{R}$ - 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 dynamiczny w przypadku ogólnym

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

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

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

Algorytm Brute-Force

$O(n!)$ - pełny przegląd

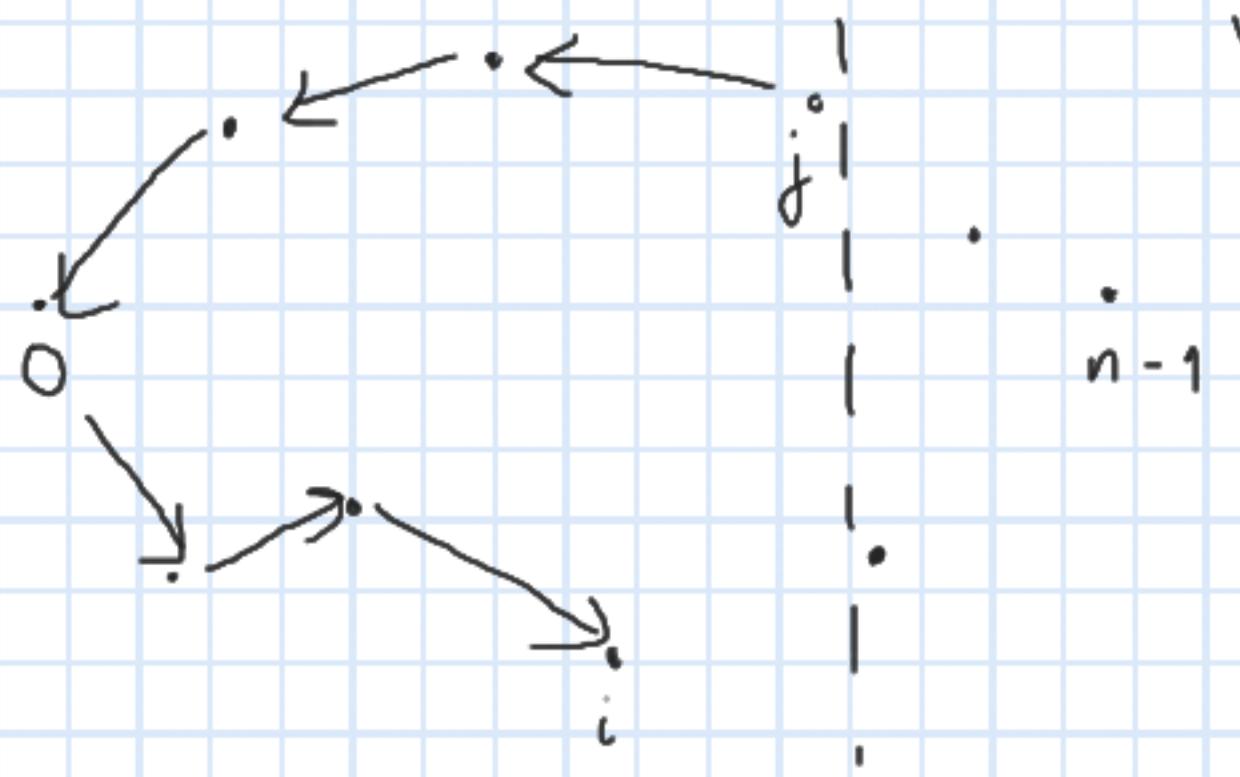
Odmytywanie wyniku

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

Problem komiwojciwa (wersja litoninna)

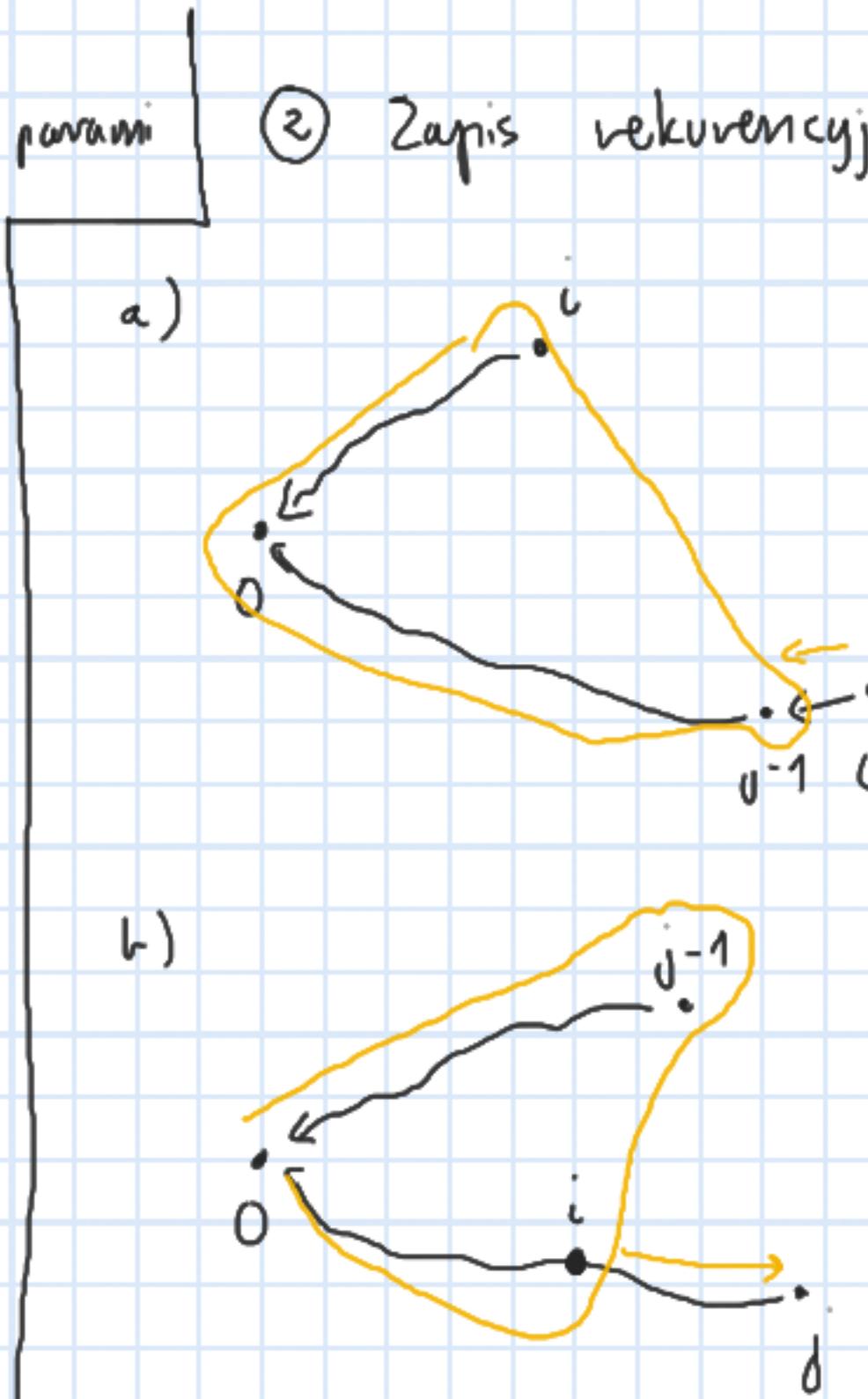
(usprzęgłe × parami
wizne)

② Zapis rekurencyjny



① $f(i, j) = \text{koszt szeregu } z 0 \text{ do } i \text{ oraz } z 0 \text{ do } j$
 $i < j$
 które odniedziałają wszystkie miasta
 $0, 1, \dots, j$ i żadnego nie powtarzają

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



$$i < j-1$$

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

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

③ Implementacija (rekurenja se spomistvaniem)

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

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

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

Złożoność

$O(n^2)$

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

if $F[i][j] \neq \inf$: 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])

$F[j-1][j] = \text{best}$

else:

$F[i][j] = \text{tspf}(i, j-1, F, D) + D[j-1][j]$

return $F[i][j]$

Algorytmy i Struktury Danych

Wykład 13

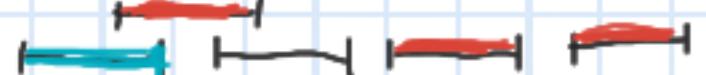
Algorytmy zadziałane — podejmij "najlepszą lokalną decyzję" i mieć nadzieję, że to zadziała

Uwagi

- często nie daje optymalnego rozwiązania
↳ w czasie daju!
- często bardzo szybkie
- czasem daje duże rozwiązanie przyblzone

Dział poprawne

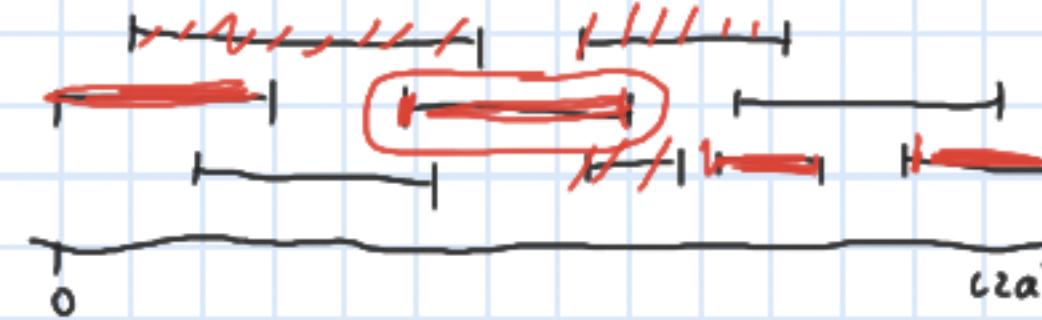
- uznamy perne rozwiązanie optymalne



- dotknąć do tego rozwiązania przedział koniuszy się najwcześniej
- usunąć powstałe konflikty → jest dokładnie jeden

Problem wyboru zadań

Dane:



4

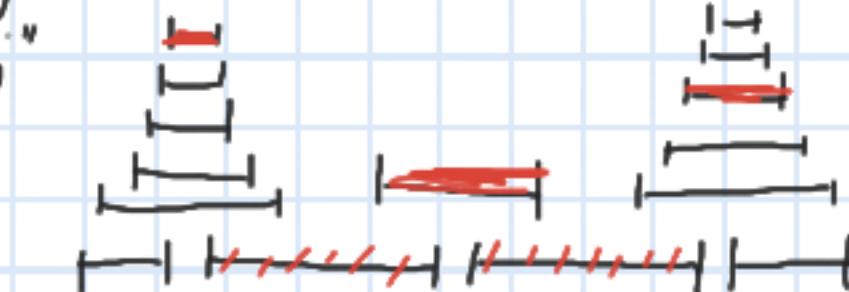
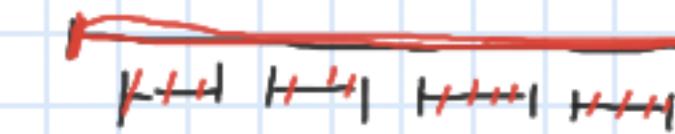
Zadanie: Wybrać jak najwięcej niekonkurencyjnych się przedziałów

Pomysły

- wyliczamy najkrótsze przedziały
- wyliczamy przedział koniuszy się najwcześniej
- wyliczamy przedział zamykający się najwcześniej
- wyliczamy krótki sposób "koniuszy się najwcześniej", "startując najpierw"
- wyliczamy przedział generujący najmniej konfliktów



← rozwiązanie poprawne



Kody Huffmmana

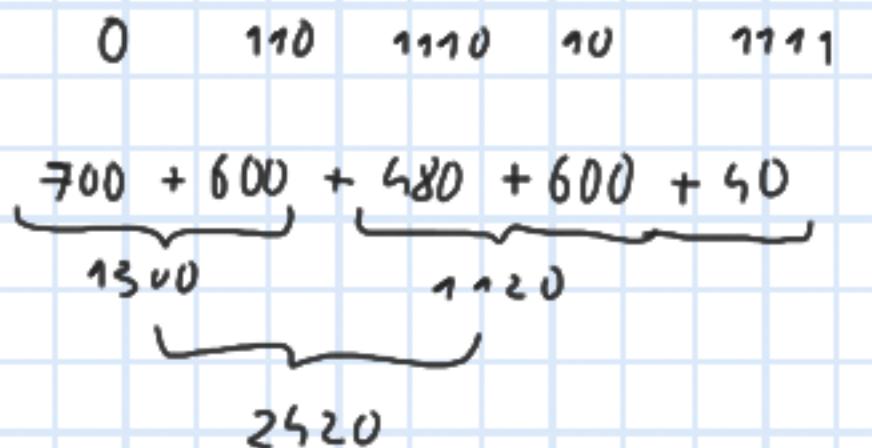
Kod binarny z symbolami różnych

długości nie musi być optymalny

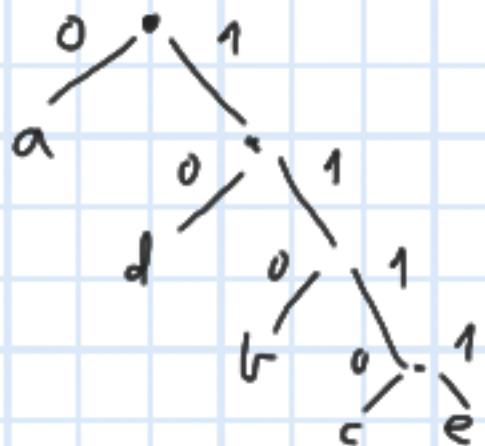
symbole:	a	b	c	d	e
wystąpienia	700	200	120	300	10
kod	000	001	010	011	100

$$1330 \times 3 = 3990$$

inny kod



Kod prefiksowy - kod zadnego symbolu nie jest prefiksem innego kodu

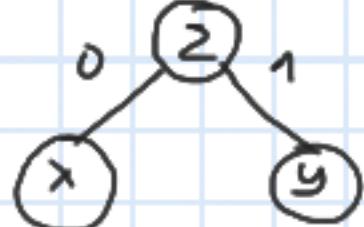


Naturalny algorytm zachowany tworzenia kodu

- vez dva symbole o najmniejszej częstotliwości \rightarrow $x \vee y$

$$f(z) = f(x) + f(y)$$

- połącz je w jeden nowy symbol o sumie częstotliwości $x \vee y$



Ponkład

level 0:	50	5	60	65	20
	a	b	c	d	e

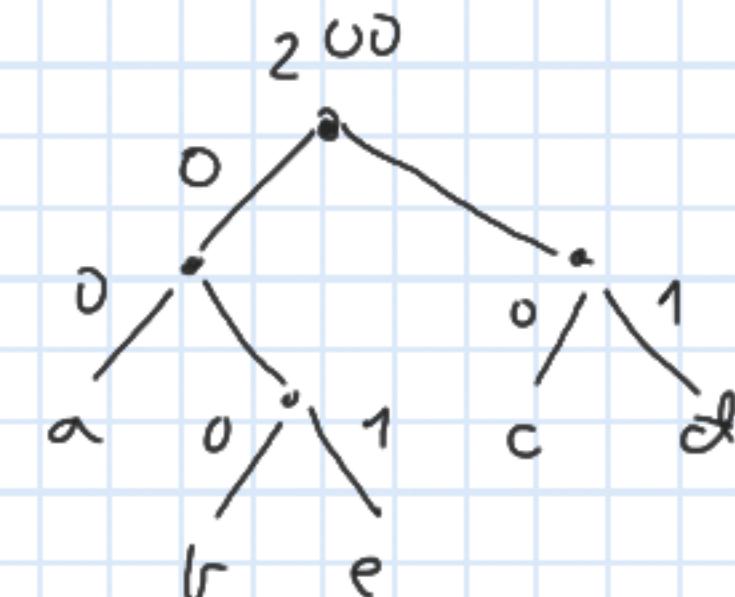
level 1:	50	60	65	20
	a	c	d	e

level 2:	50	60	65	20
	a	c	d	e

level 3:	75	60	65	125
	a	c	d	b

level 4:	115	125	125	125
	a	b	c	d

Krok 4:



Dla naszego algorytmu jest poprawny?

\bar{T} - drzewo kodujące

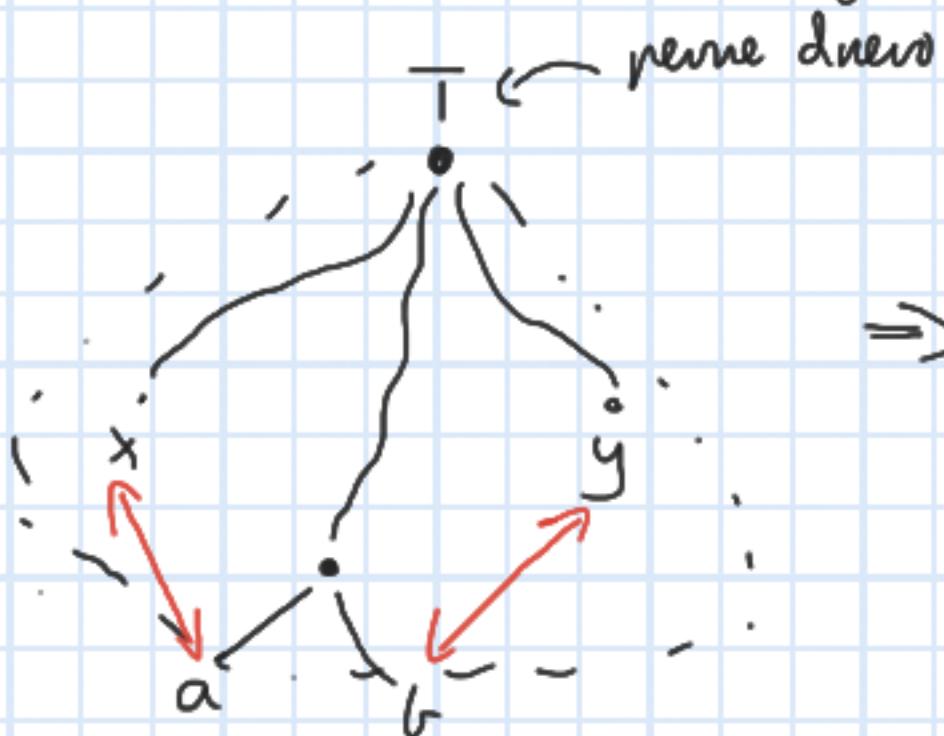
$B(\bar{T})$ - koszt drzewa

$$B(\bar{T}) = \sum_{\text{symbol} s} f(s) \cdot d_{\bar{T}}(s)$$

↑
symbol s
wystąpienie s

długość kodu s
w drzewie \bar{T}

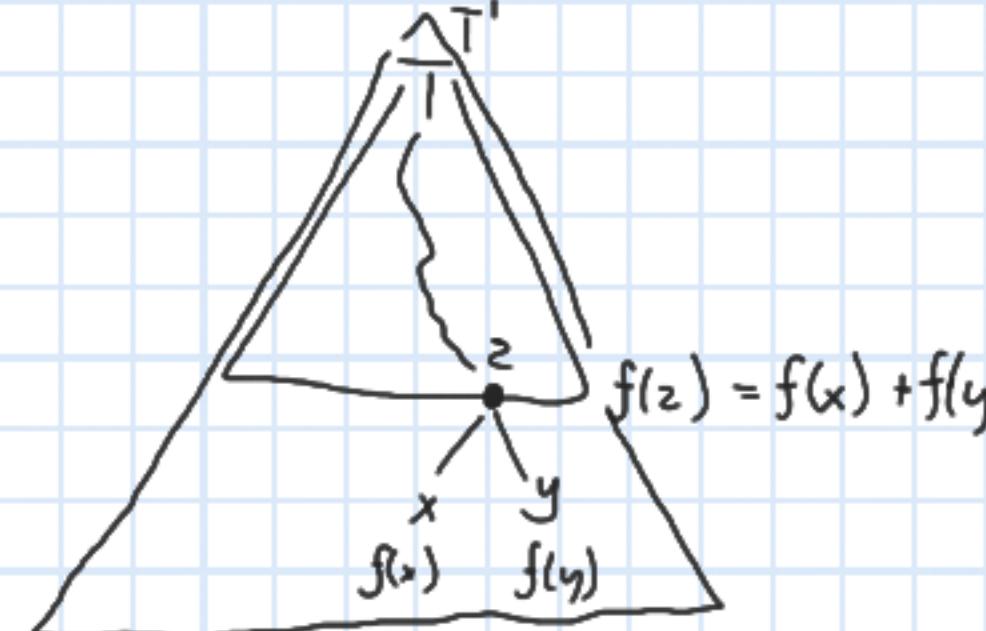
Krok 1: Dwa najbardziej skomplikowane symbole možna umieszczać w wspólnym węźle



a, b - symbole na najdłuższej ścieżce

x, y - najbardziej skomplikowane symbole

Krok 2: Optymalna podstruktura



$$B(T') = B(\bar{T}) + f(x) + f(y)$$

$$\begin{aligned} B(\bar{T}') &= B(\bar{T}) - f(a)d_{\bar{T}}(a) + f(a)d_{\bar{T}'}(x) \\ &\quad - f(b)d_{\bar{T}}(b) + f(b)d_{\bar{T}'}(y) \\ &\quad - f(x)d_{\bar{T}}(x) + f(x)d_{\bar{T}'}(a) \\ &\quad - f(y)d_{\bar{T}}(y) + f(y)d_{\bar{T}'}(b) \\ &= \sqrt{d_{\bar{T}}(a)(-f(a) + f(x)) + d_{\bar{T}}(b)(-f(b) + f(y))} \\ &\quad \quad \quad \boxed{d_{\bar{T}}(x)(-f(x) + f(a)) + d_{\bar{T}}(y)(-f(y) + f(b))} \\ &= \boxed{(f(x) - f(a)) \underbrace{[d_{\bar{T}}(a) - d_{\bar{T}'}(x)]}_{\leq 0} + (d_{\bar{T}}(b) - d_{\bar{T}'}(y)) \underbrace{[f(y) - f(b)]}_{\geq 0}} \\ &\leq B(\bar{T}) \end{aligned}$$

Problem plecakowy (dyskretny / ciągły)

Dane: p_1, \dots, p_n — przedmioty

$v(p_1), \dots, v(p_n)$ — wartość przedmiotu

$m(p_1), \dots, m(p_n)$ — waga przedmiotu

M — ile uniesie zbiadz

Zadanie: Ułożyć przedmioty, których masa

nie przekracza M i których wartość
jest maksymalna

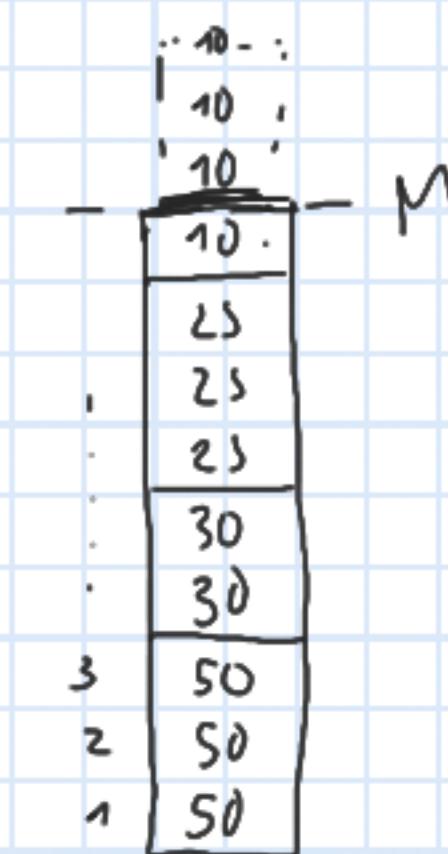
Uwaga dyskretna \rightarrow algorytm dynamiczny

Uwaga ciągła \rightarrow algorytm zadający

↓
wyliczamy przedmioty o
najlepszym stosunku ceny
do wagi

dyskretny — przedmiot bieremy lub nie

ciągły — można brać fragmenty
przedmiotów



Algorytm zadający działa dla prob. ciągłej,

ale nie dla dyskretnego

$$v(p) : \boxed{70} \quad | \quad \boxed{55} \quad \boxed{35}$$

$$m(p) : \boxed{70} \quad | \quad \boxed{60} \quad \boxed{40}$$

$$M = 100$$

$$v(p) \quad \boxed{2} \quad | \quad \boxed{M-1}$$

$$m(p) \quad \boxed{2} \quad | \quad \boxed{M}$$

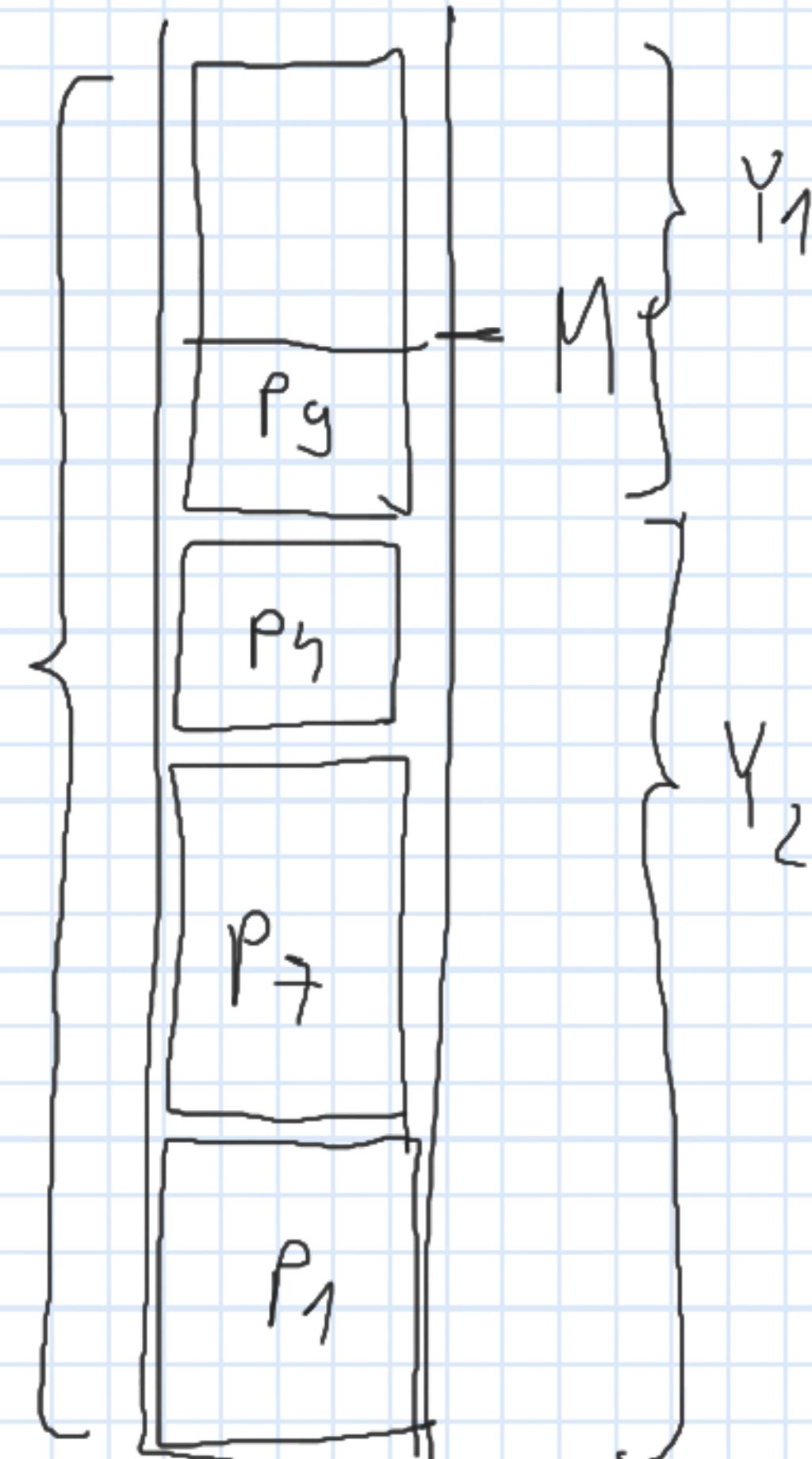
$$M$$

Algorytm zadania 2: Ułóżmy najcenniejsze przedmioty

$v(p)$	2	1	1
$w(p)$	M	1	1
		M	

X

> suma wartości
tych przedmiotów
 \geqslant wartości
nauk. dyskret.



$$Y_1 + Y_2 = X$$



$$Y_1 \geq \frac{X}{2}$$

lub

$$Y_2 \geq \frac{X}{2}$$

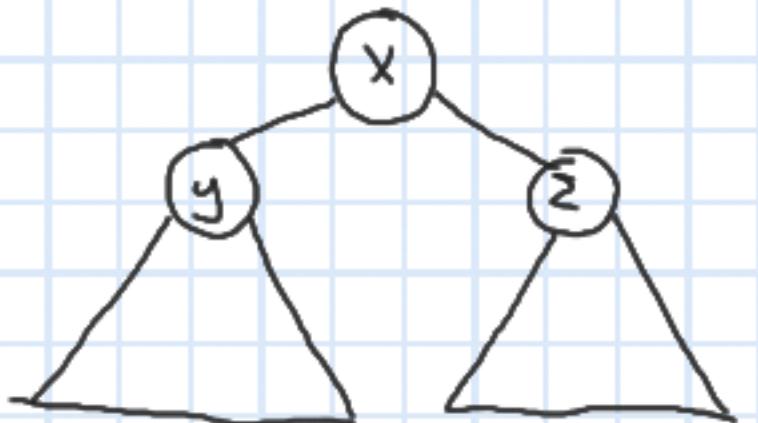
Algorytmy i Struktury Danych

Wykład 14

Pogląd

Tablice asocjacyjne - "cos co przerala na indeksowanie
dowolnym typem danych"

Dzewo BST (ang. binary-search tree)



podstawa
zasada:
 $y \leq x \leq z$

class BSTNode:

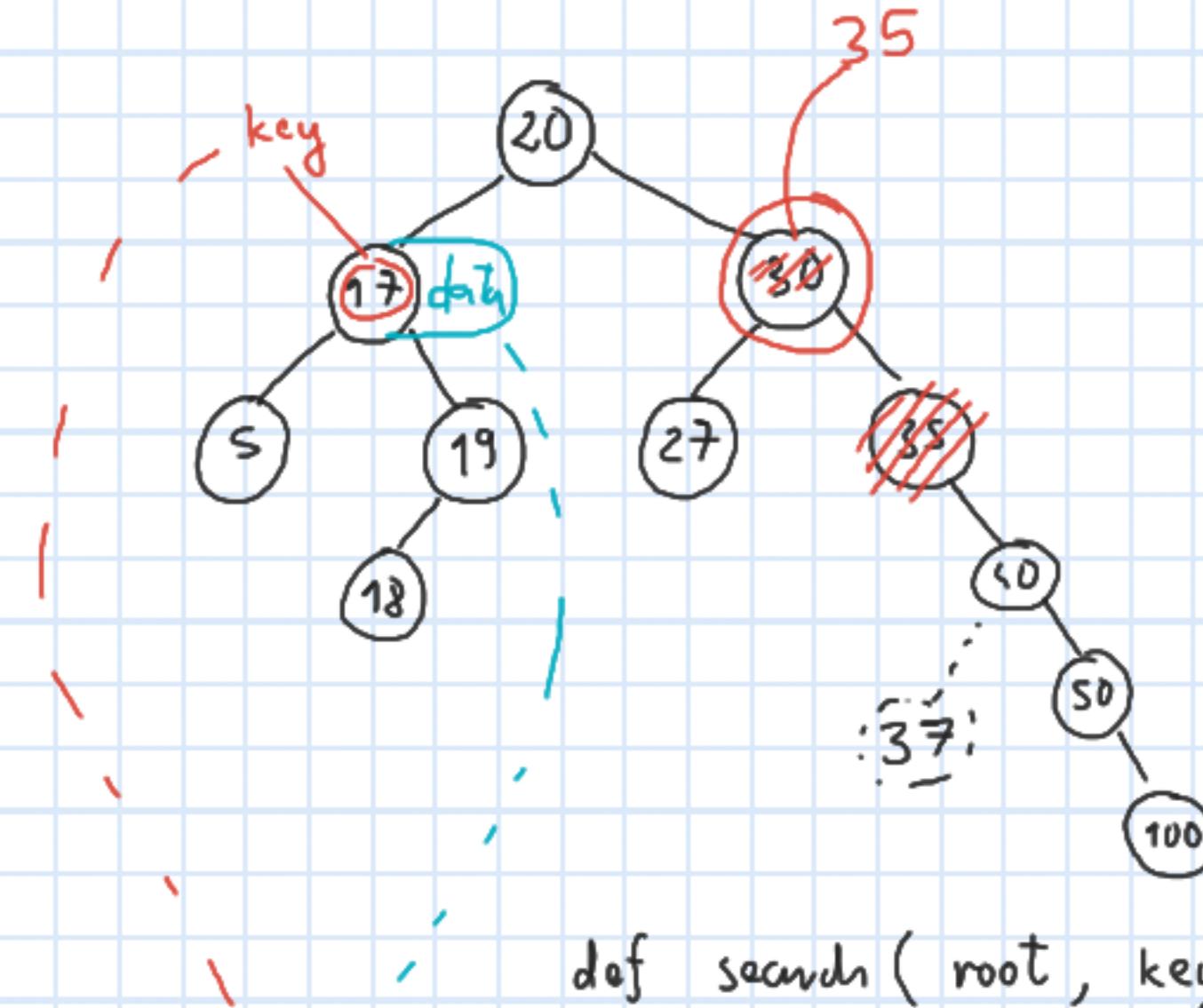
def __init__(self):

self.left = self.right = None

self.parent = None

self.key = None

self.data = None



def search(root, key) :

while root != None :

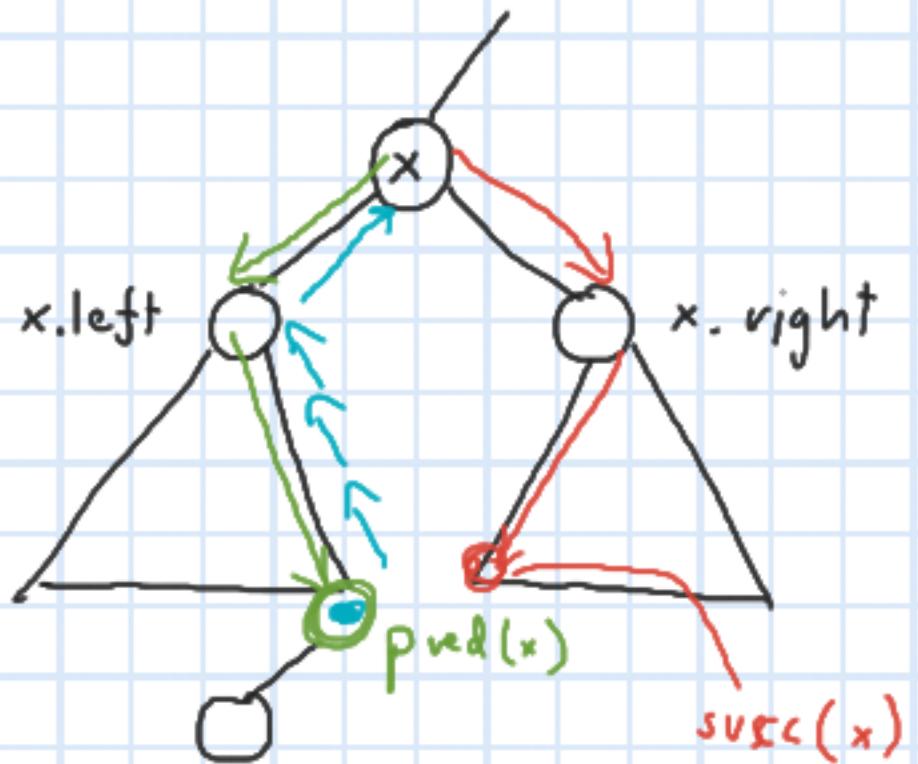
if root.key == key: return root

elif key < root.key: root = root.left

else: root = root.right

return None

Oblinanie następnika i poprzednika w drzewie BST



$succ(x)$ - jeśli x ma prawe dziecko, to znajdź minimum w prawym poddrzewie x

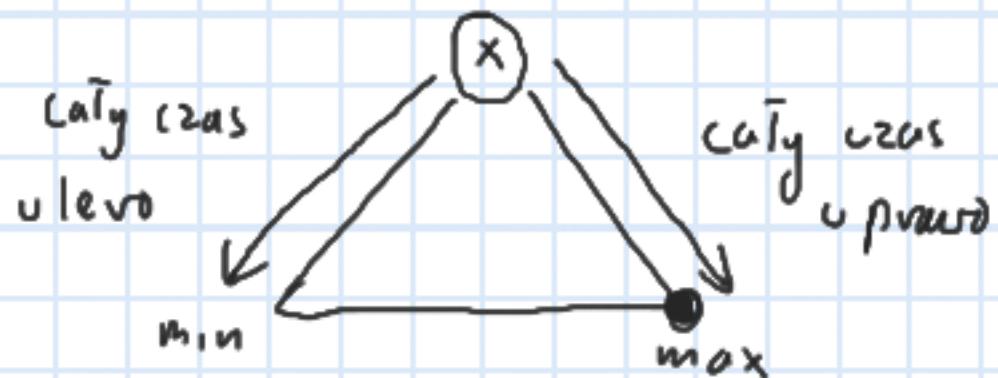
- jeśli x nie ma prawnego dziecka, to

wędruj w głąb drzewa, poki jutci

prawym synem

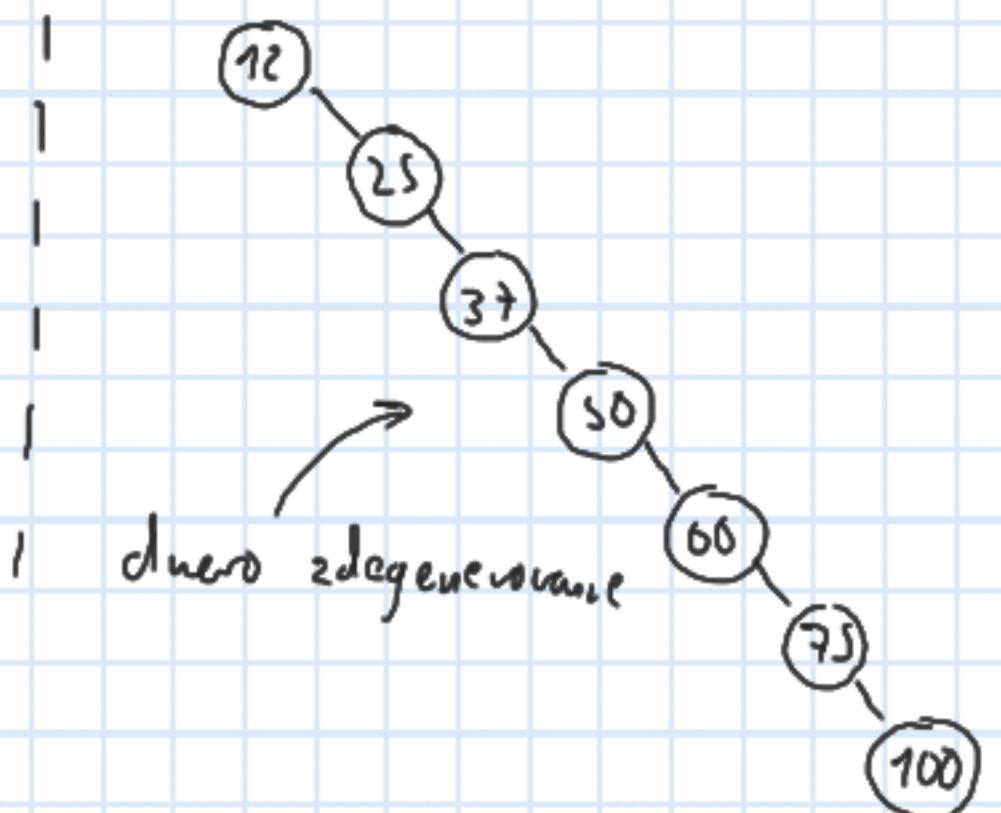
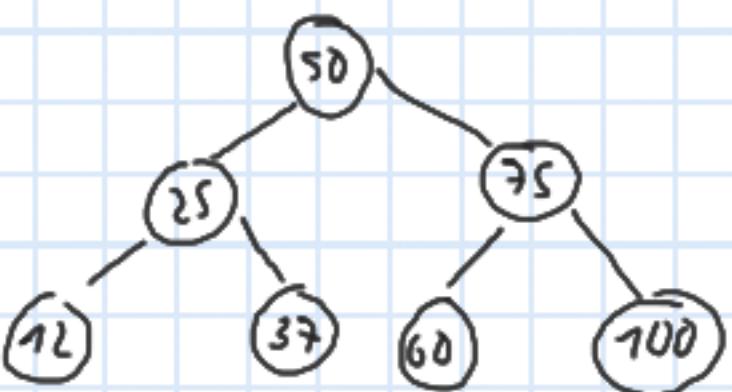
$pred(x)$ - analogicznie

Uzywając min/max w drzewie BST



Złożoność

$O(h)$ - gdzie h to wysokość drzewa



drzewo zdegenerowane

Dzewia czarno - biała

Są to dżewia BST, w których dodatkowo obowiązują następujące zasady:

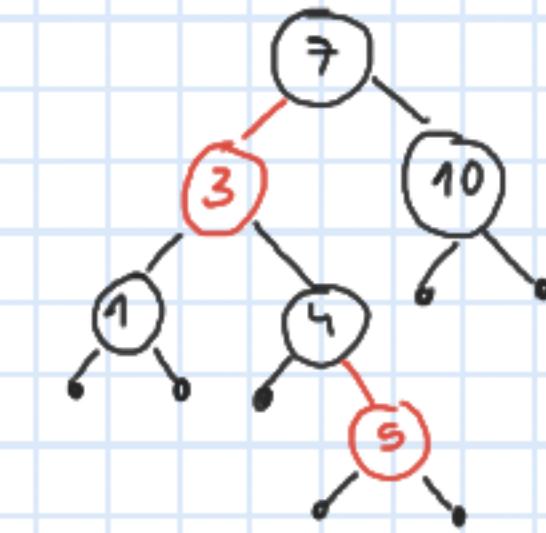
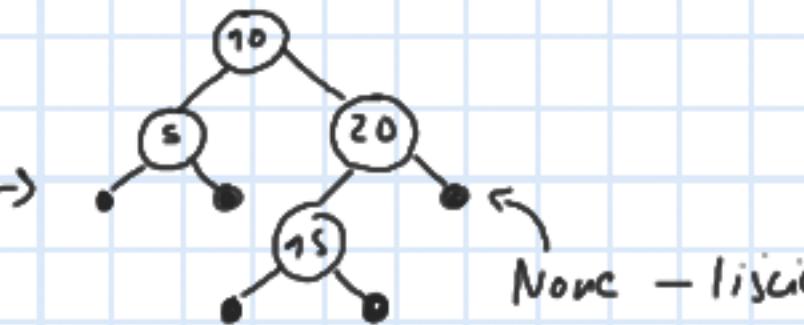
① każdy węzeł jest albo czerwony, albo czarny

② koncej jest czarny

③ każdy liśń jest czarny

④ jeśli węzeł jest czerwony, to obaj jego synowie są czarne

⑤ każda prosta ścieżka z ustalonego węzła do liśnia zawiera tyle samo czarnych węzłów



Pogląd

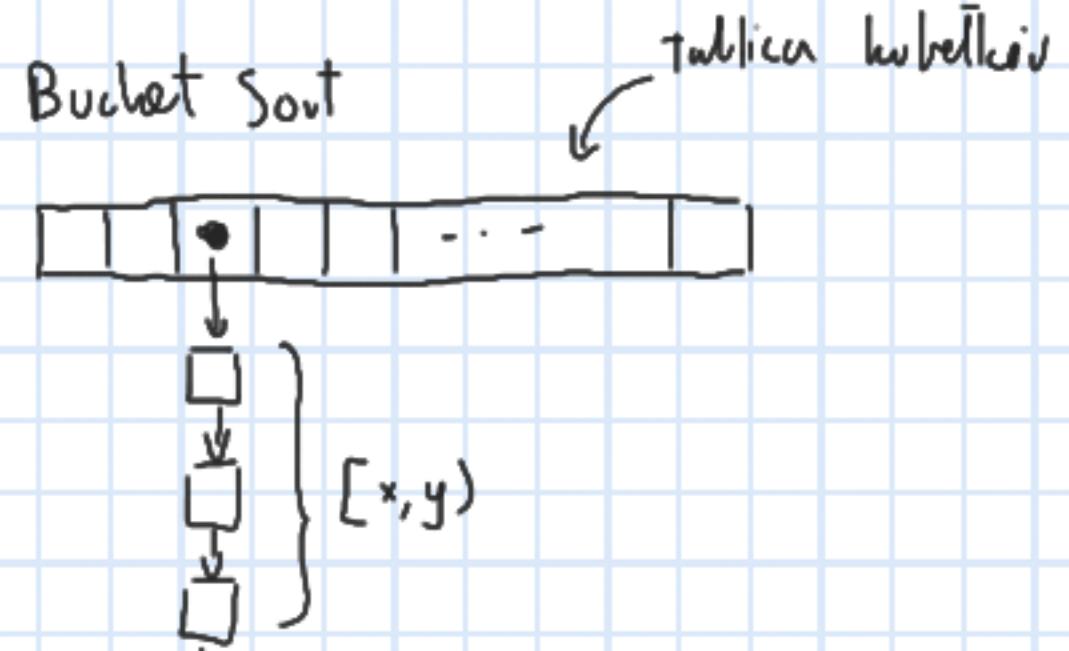
$H(x)$ - wysokość dżewia zakonczonego w x

$BH(x)$ - j.w., ale linie tylko czarne węzły (czarna wysokość)

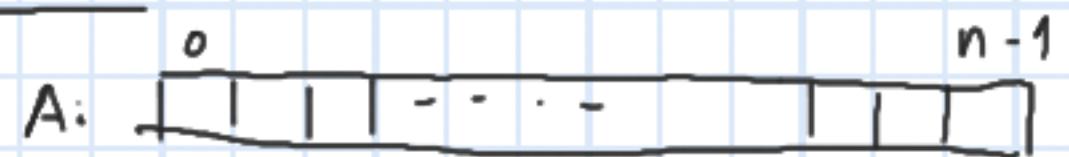
$$H(x) \leq 2BH(x)$$

Dżewo zakonczone w x ma w najmniej $2^{BH(x)} - 1$ węzłów

Tablice haszujce



Idea



Tworzymy funkcję haszującą, wyznaczającą indeks danego elementu.

$$h: \text{Dane} \rightarrow \text{IN}$$

Element o kluczu d umieszczamy pod indeksem

$$h(d) \bmod n$$

Założenie: podobne klucze mają zawsze różne wartości funkcji haszujących

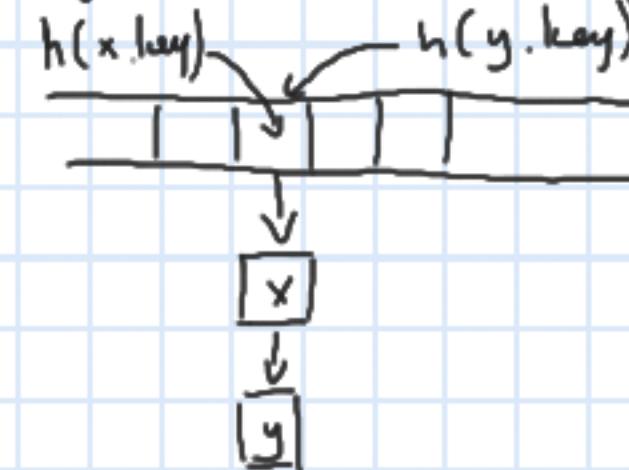
Co jeśli dla pewnych x i y zachodzi:

$$h(x.\text{key}) \equiv h(y.\text{key}) \pmod{n}$$

Rozwiązywanie konfliktów

↳ metoda listowa

tablica haszująca jest takaż listą; dwa elementy o tym samym hashu przedstawiane są na jednej linii



Bardziej skomplikowane
ustawianie elementów

2 tablice

→ adresowanie otwarte

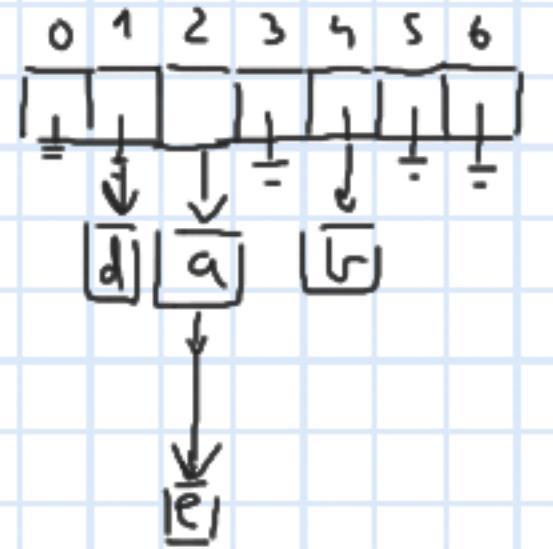
- umysłicie klucze bezpowtarzalne w tablicy

- jeśli dane pole jest zbytec, to dla nowego klucza
zajmujący inne

- np. następuje $h(x.\text{key}) + at \pmod{n}$
- $h(x.\text{key}) + th_2(x.\text{key})$

lub inny sposób

Pogląd rozwiązywanie konfliktów



a, b, c, d, e

$$h(a) \bmod 7 = 2$$

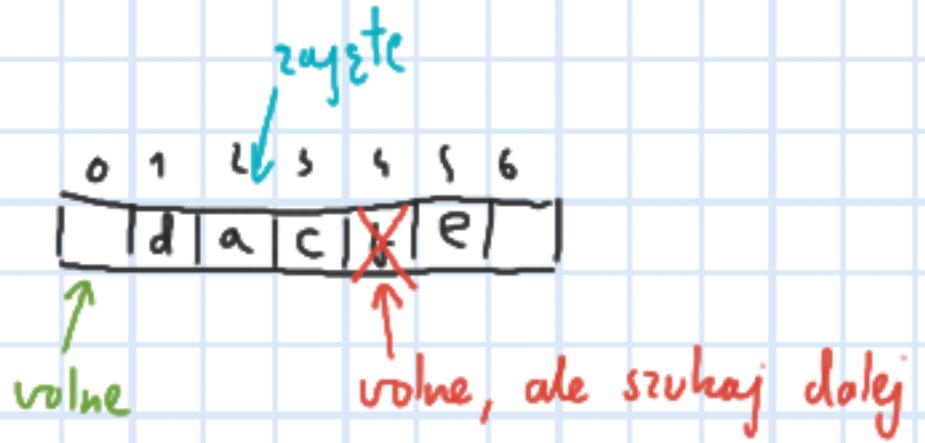
$$h(b) \bmod 7 = 4$$

$$h(c) \bmod 7 = 2$$

$$h(d) \bmod 7 = 1$$

$$h(e) \bmod 7 = 2$$

listowe



otwarte,
liniowe

Jak stworzyć funkcję hanującą?

- zamiana danych na indeks



- harmoniczne universalne

p - liczba pierwonna

$$k \in \{0, \dots, p-1\}$$

$$h(k) = [(ak + b) \bmod p] \bmod n$$

↑ ↑
wyliczony losowo

Algorytmy i Struktury Danych

Wykład 15

Dewa przedziałowe — intencuje nas struktura
przechowująca przedziały i pozwalająca odnosić
się do przedziałów zawierających dany punkt

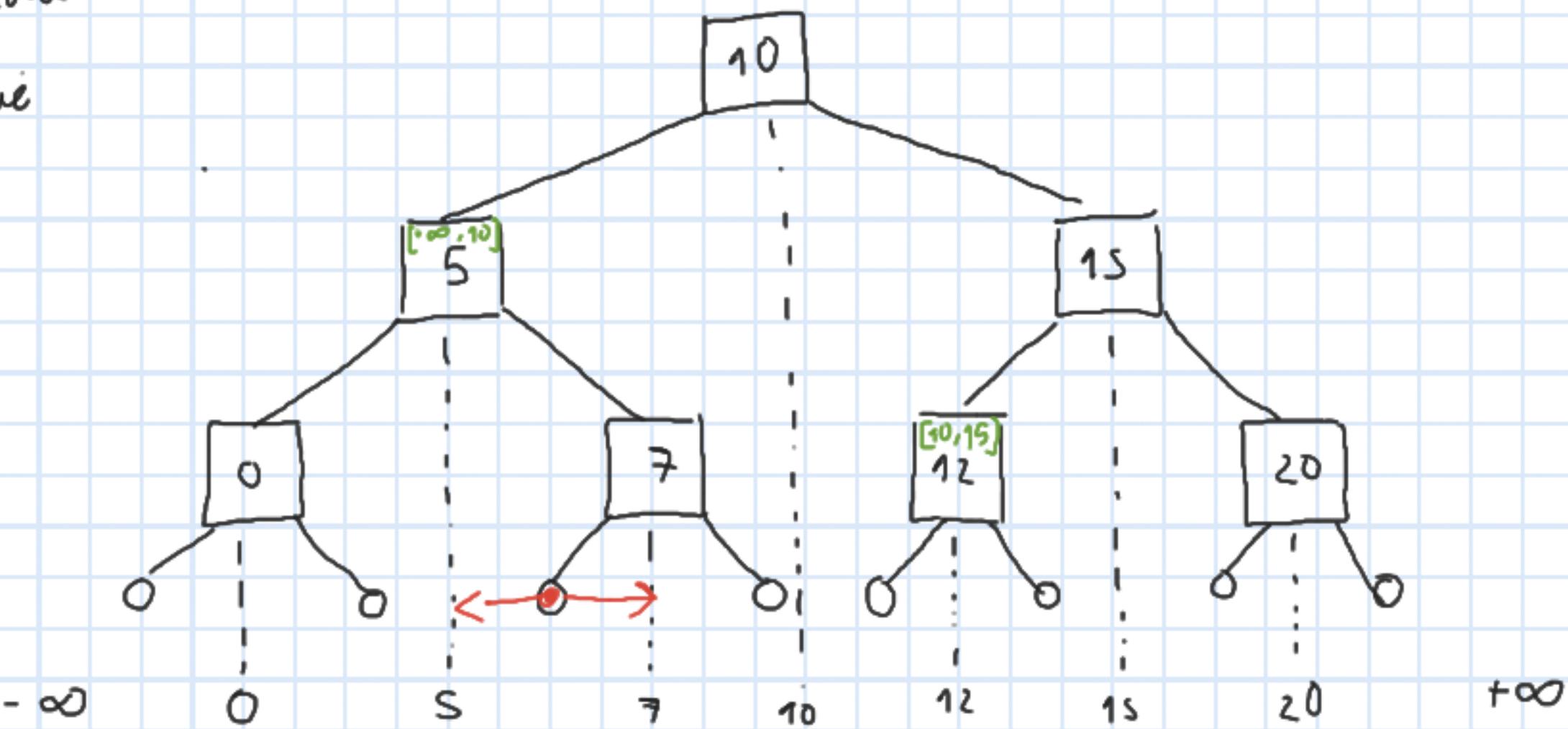
Punkty

Rozważmy przedziały

$[0, 10], [5, 20], [7, 12], [10, 15]$



Konstruujemy dwoje BST zaznaczające punkty
graniczne przedziałów bazowych

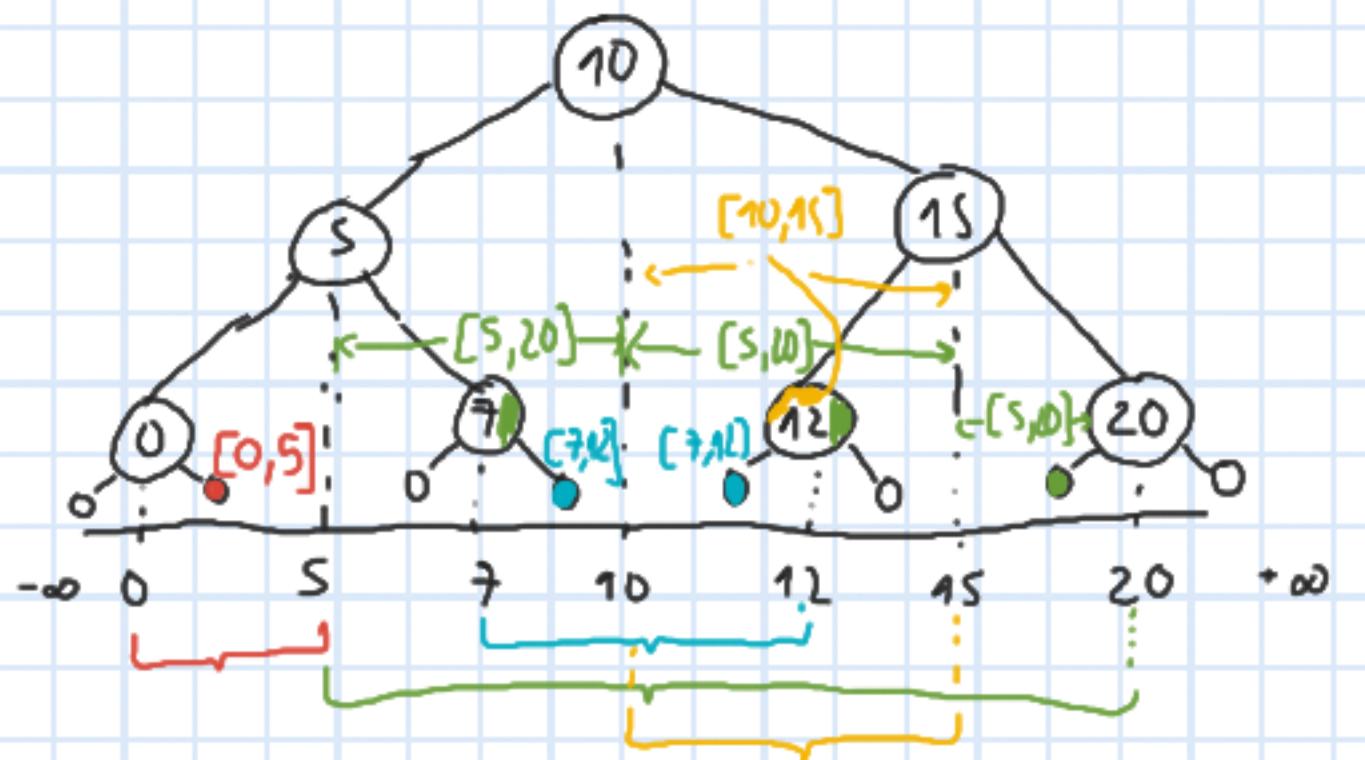


Każdy liść odpowiada za dany przedział bazowy

Każdy węzeł odpowiada za sumę przedziałów bazowych
w jego poddrzewach

$$\text{span}(10) = [-\infty, \infty], \quad \text{span}(7) = [5, 10], \quad \text{span}(15) = [10, \infty]$$

Predział przedstawiany w tych węzłach, którymi spraw całkowicie się w tym przedziale zatrzymał (a nie jest to prawda dla węzła 15)



Złożoność pamięciowa - kiedy przedział na danym porządku dnia występuje najwyżej 2 razy \Rightarrow n przedziałów daje złożoność pam. $O(n \log n)$

Ustalanie przedziału

Zatrzymajac od konca rozważany węźły

- jeśli przedział "pasuje idealnie" (jest równy spanowi węzła) to ustalany go.

- jeśli nie to:

- przenieś przedział ze spanem lewego dziecka ustalany po lewej stronie (rek.)
- przenieś przedział ze spanem prawnego dziecka ustalany po prawej

Złożoność czasowa: $O(\log n)$

ustawiam u dany poddane o ile przenieść niepotrzebne

Oblinianie przedziałów, do których nalezy dany punkt

- znajdujemy przedział brązowy, do którego należy dany punkt
- wędrujemy u góry gromadząc napotkane przedziały

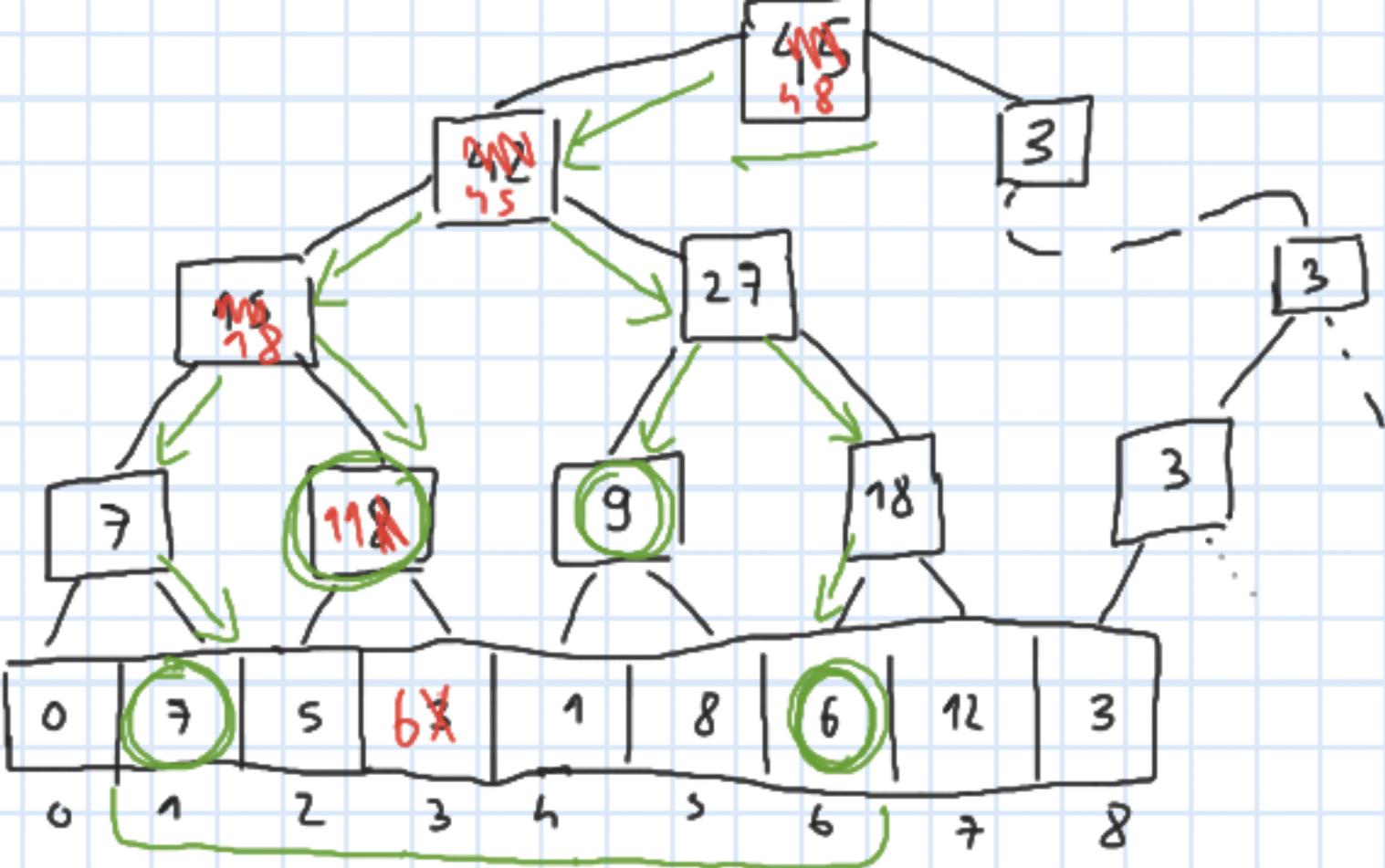
Pryktad konstrukcji drzewa przedstóvego, specjalizowanego do innego zadania

Interesuje nas struktura danych przedstawiona n lińi indeksowanych od 0 do $n-1$ i oferująca następujące operacje:

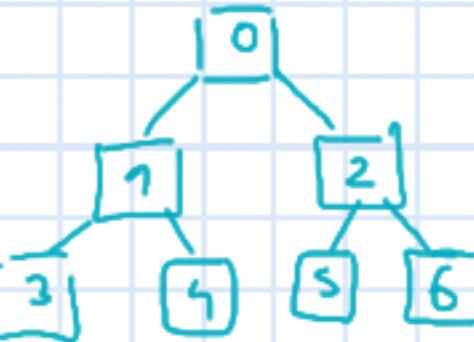
- $\text{update}(i, x)$ - zmień i -ty lińi na x
- $\text{sum}(i, j)$ - oblicz sumę lińi od i -ej do j -ej

Update - zmieniamy element tablicy i oznaczony $O(\log n)$
wahlując sumy w górz drzewa

$\text{sum}(i, j)$ - analogiczne do ustalania przedziałów poprzednio

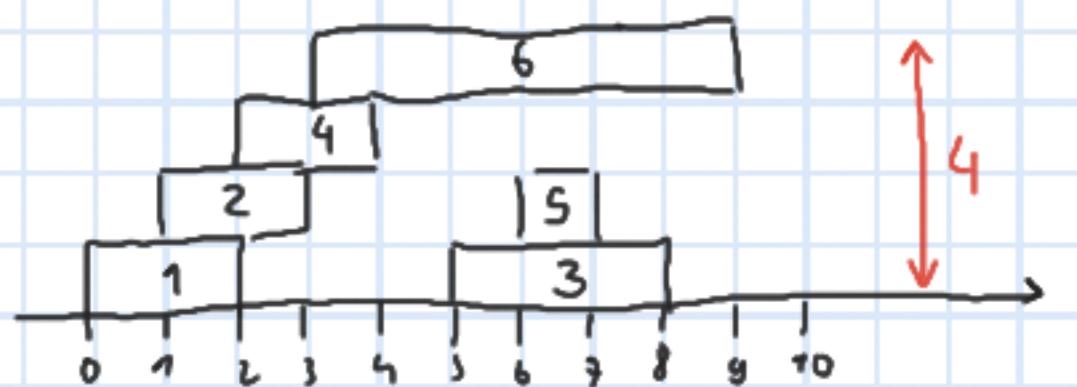


$$\text{sum}(1,6) \\ 7 + 11 + 9 + 6$$



$$\text{parent}(x) = \left\lfloor \frac{x-1}{2} \right\rfloor \\ \text{left}(x) = 2x+1 \\ \text{right}(x) = 2x+2$$

Punktor: Spadające klocki



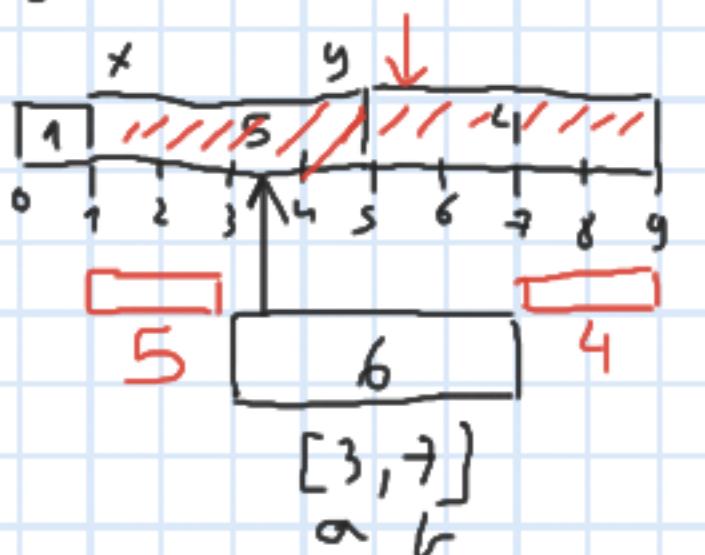
Zadane klocki spadają w kolejności z góry: dotkują się do najbliższego klocka, który dotyka.

Należy wyznaczyć wysokość pozostały konstrukcji.

Stan konstrukcji po pierwszych i -takach można zapisać jako ciąg mezadodrążnych na sieci mediatów



Gdyby spadł klock [1,5] to nie ciąg zmienia się na.



złożoność $O(nlogn)$

Ciąg rozłącznych mediatów reprezentujących stan konstrukcji przedstawiony u dnieje prawidłowo - czarnym (sortujemy mediaty względem punktu początkowego)

Jak znaczymy klock $[a, b]$?

① znajdujemy medial $[x, y]$ zaczynający publiczny $a + \frac{1}{2}$
o wys. h

② - usuwanie $[x, y]$ z dnerwa
- ustanawianie $[a, x]$ o wys. h

③ - jeśli $[b, y]$ jest pusty, to ponownie powiększa dla punktu $y + \frac{1}{2}$
- jezeli nie jest pusty, ustawić $[b, y]$ o wys. h

④ Ustawić $[x, y]$ z wysokością równą max. wysokości odniesionej obrazu + 1

⑤ Ustawić $[x, y]$ z wysokością równą max. wysokości odniesionej obrazu + 1

Realizacija u općini o dresu pmedžatice

Dla n klocków mamy najwyżej 2^n różnych punktów pionów/końca \rightarrow numerowanych klocków tak, że każdy ma pionek i koniec u zasięgu od 0 do $2^n - 1$

