

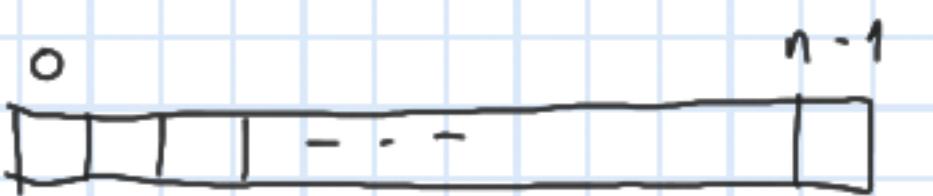
# Algorytmy i Struktury Danych

## Wykład 5

### Abstrakcyjne struktury danych

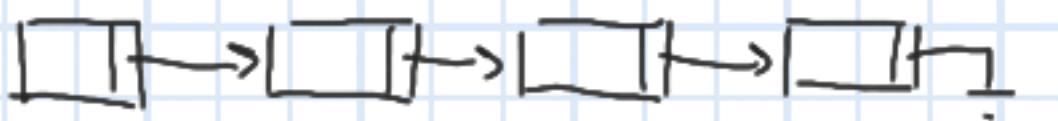
- "kontrakt" co do działania
- zestaw operacji
- fizyczna realizacja

### Tablica



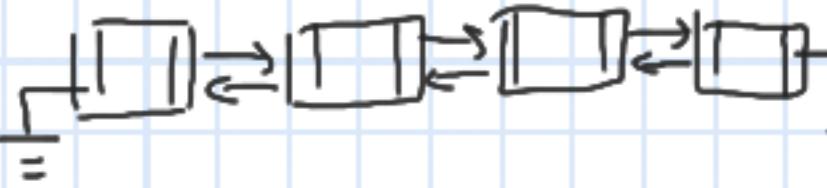
"Cos co ma ponumerowane komórki, do których można się bezpośrednio odwoływać"

### Lista jednoliciwunkowa



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

### Lista dwuściennkowa



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

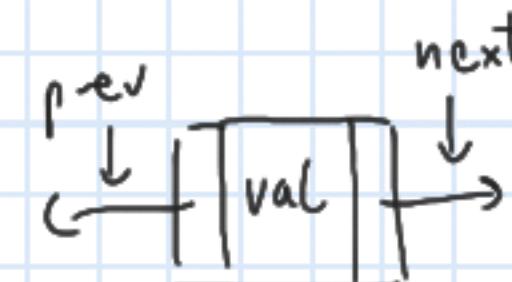
class DNode:

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

self.prev = None

self.next = None

self.val = None



Lista dwuściennkowa = jednogm wskaźnikiem

1 1 0 1 0 1 0 1

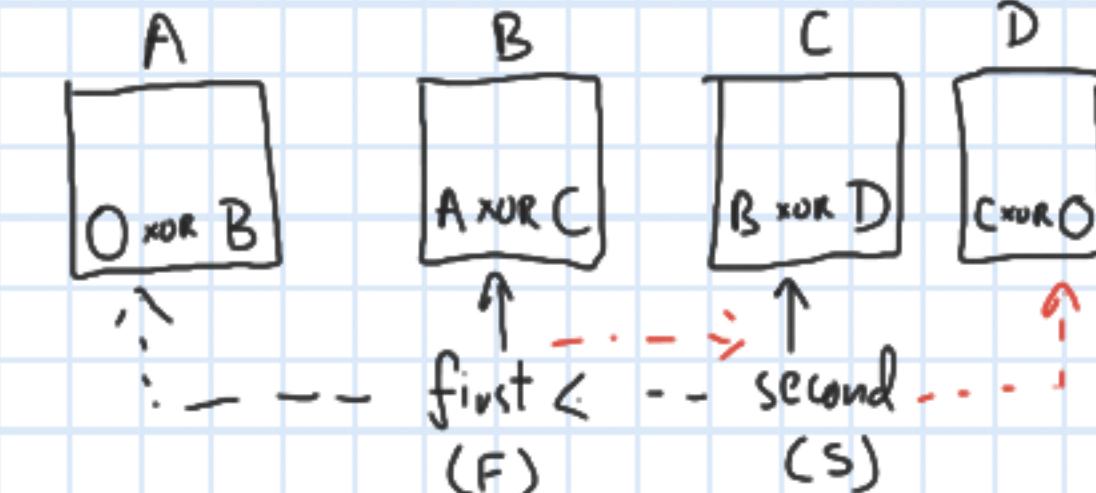
XOR 0 1 1 1 0 0 1 1

1 0 1 0 0 1 1 0

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

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

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



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

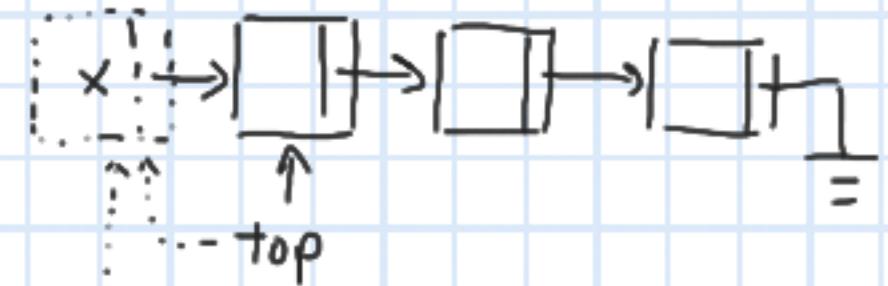
$$(A \text{ XOR } C) \text{ XOR } C = A$$

## 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

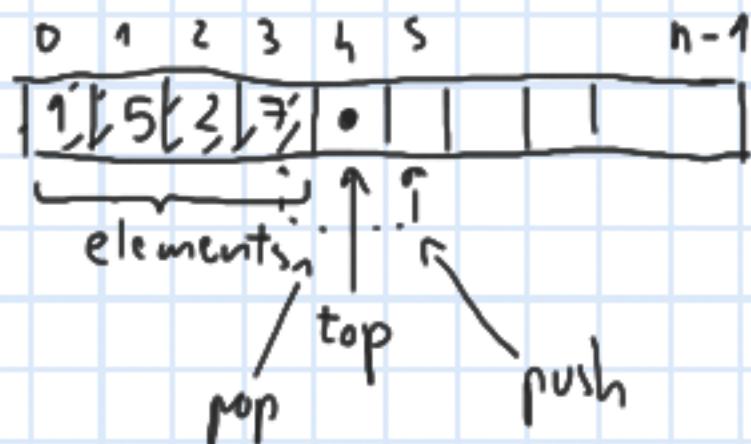
## Implementacja na liście jednokierunkowej



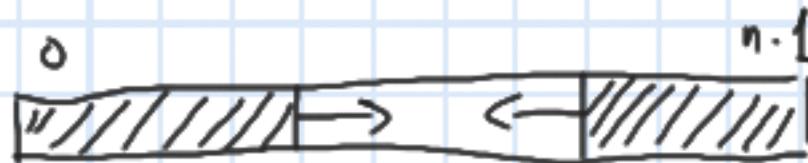
push(x)

pop()

## Implementacja tablicowa



## Dwa stosy



starsze komputery PC XT/AT

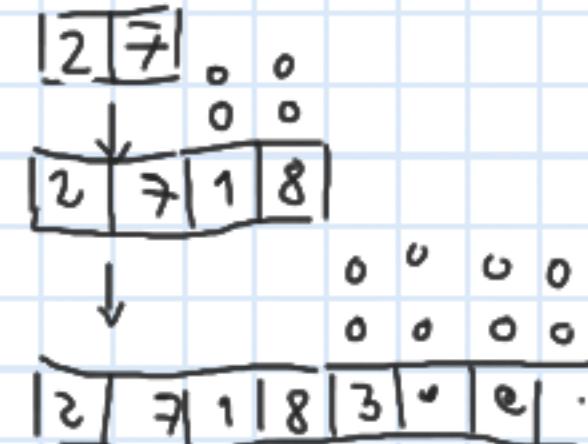
Co jeśli słowny nam się  
miejscie w zaallokowanej tablicy?

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

## Analiza kosztu zamontowanego

push - 3 zl

pop - 1 zl



## Kolejka (ang. queue)

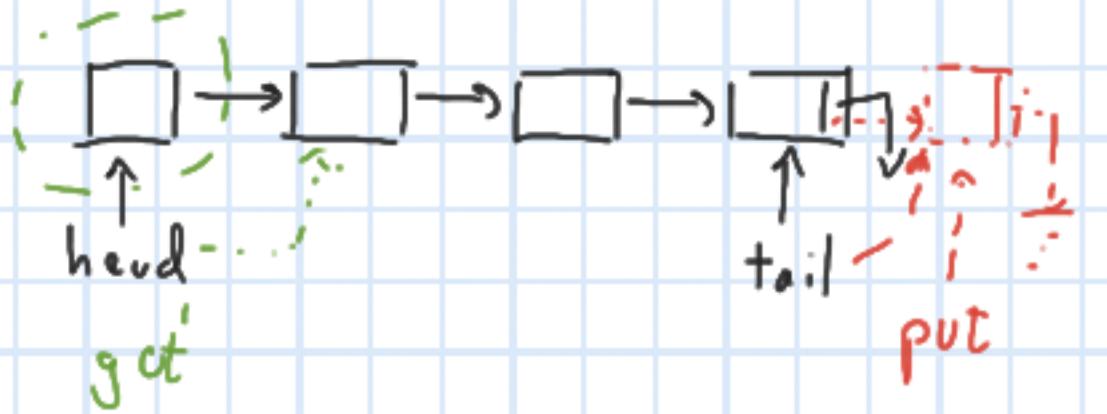
"coś, co przenosi ustwiać elementy na koniec"

używając "przeszły"

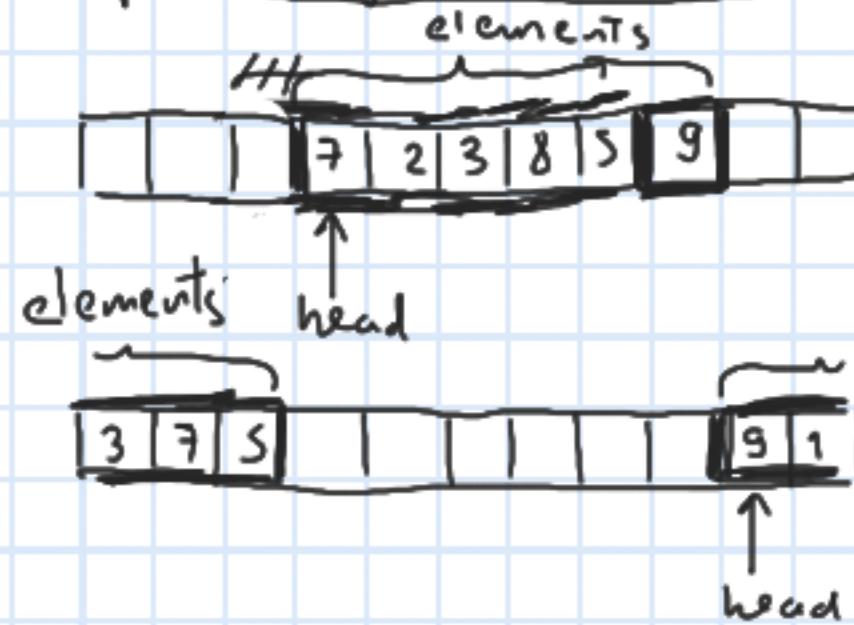
put / get

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 kolejowic  
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

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

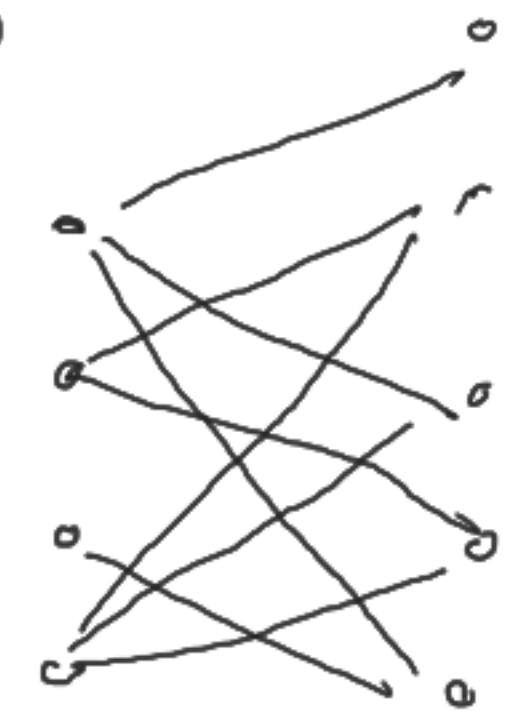
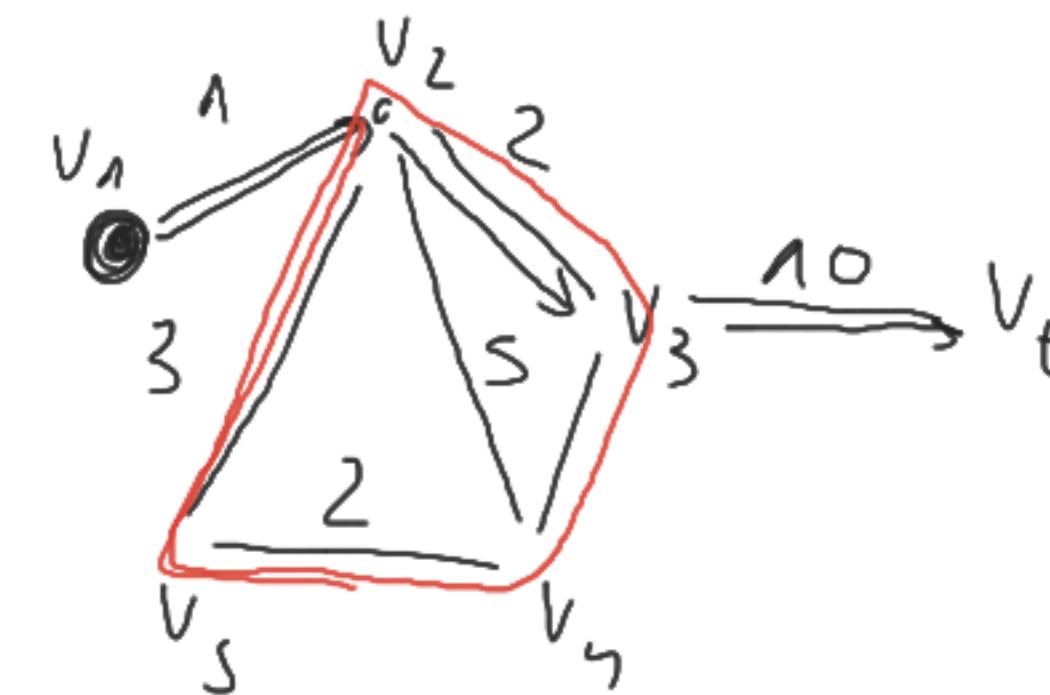
# Algorytmy Grafowe

$$G = (V, E)$$

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

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

$e_i$  - zbiór dwudziestu wierzchołków



# Algorytmy i Struktury Danych

## Wykład 6

### Algorytmy grafowe

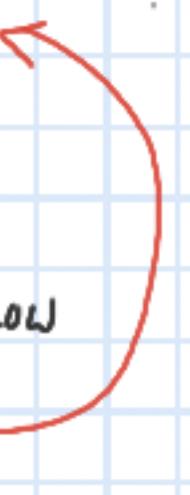
na ogół nie dopuszczamy  
petli, węzli krawędzi ( $u, u$ )

Graf skierowany:

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

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

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

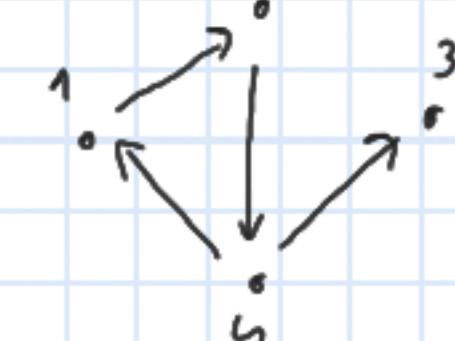
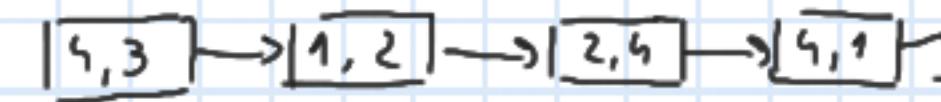


Graf nie skierowany jest zdefiniowany analogicznie,  
ale kiedy krawędź to dwielementowy zbiór wierzchołków

↳ 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



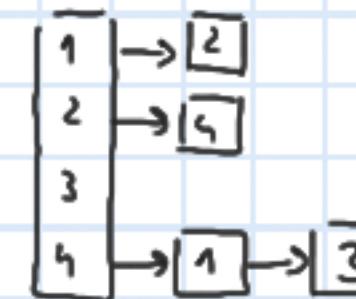
### Reprezentacja macierzowa

	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  $V_{max}$   
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:
```

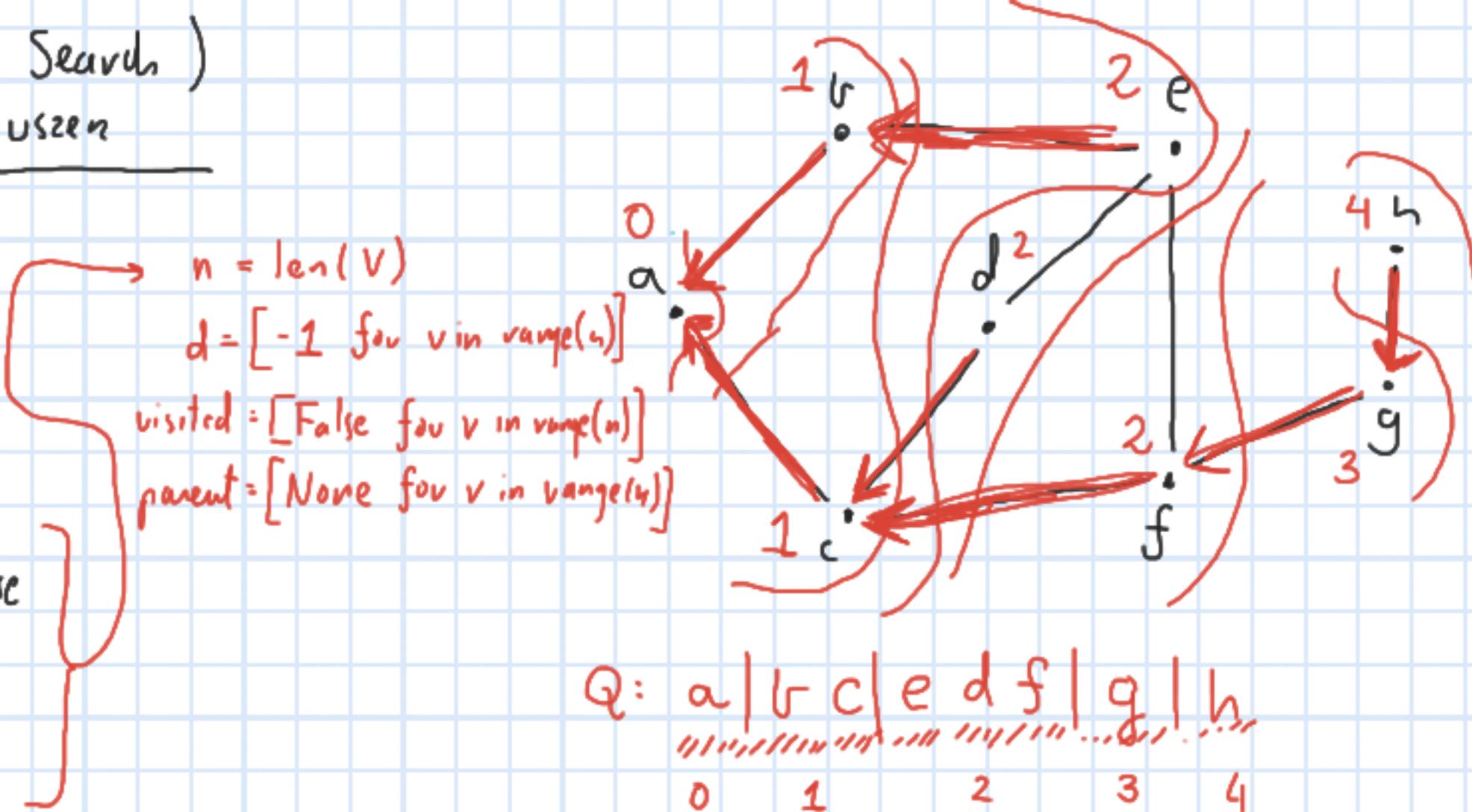
```
v.d = u.d + 1
```

```
v.parent = u
```

```
v.visited = True
```

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

return d, parent, visited



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

BFS - znajduje najkrótsze ścieżki  
 w sensie liczby krawędzi

Inne zastosowania:

- tafowanie spojów grafu
- drzewa chwytowe
- wykrywanie cykli

Zastosowanie

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

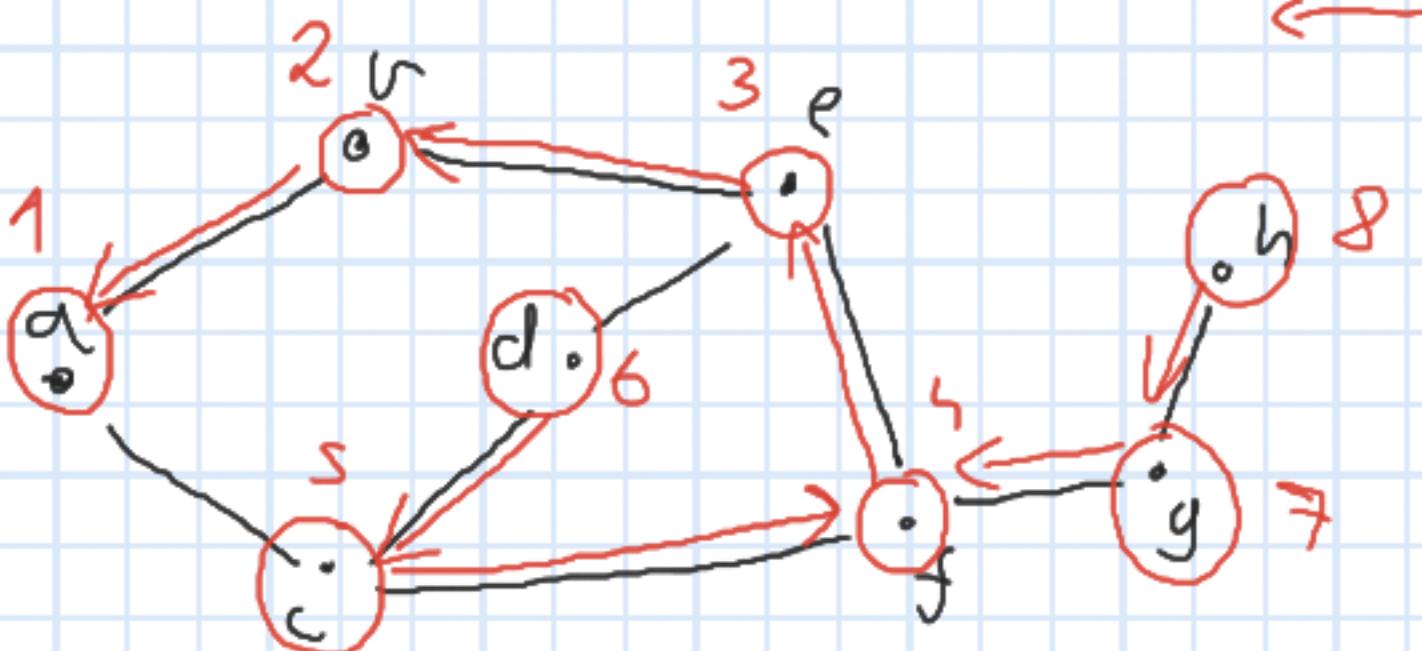
$O(V^2)$  - rep. macier

# Algorytm DFS (depth-first search)

przeszukiwanie w głębi

```

def DFS( G )
#  $G = (V, E)$ 
for  $v \in V$ :
    v.visited = False
    v.parent = None
time = 0
for  $u \in V$ :
    if not u.visited:
        DFSVisit(G, u)
    
```



def DFSVisit( $G, u$ )

nonlocal time

time += 1

u.visited = True

for  $v - sasiad u$ :

if not v.visited:

v.parent = u

DFSVisit( $G, v$ )

time += 1

wierzchołek  $u$  został odwiedzony  
i to jest czas odwiedzenia

$v$  został pnietrowany /  
czas pnietrowania

## założenia

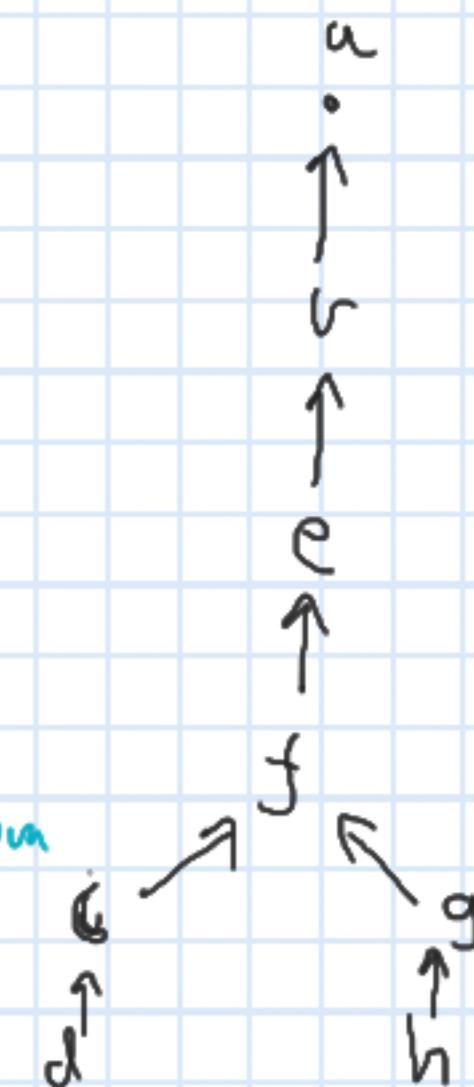
$O(V+E)$  - reprezentacja listowa

$O(V^2)$  - reprezentacja macienną

## Zastosowania DFS

- spójność
- dwudzielność
- wykrywanie cykli
- sortowanie topologiczne
- silnie spójne składowe
- cykl Eulera
- mosty / punkty artykulacyjni

## durus DFS

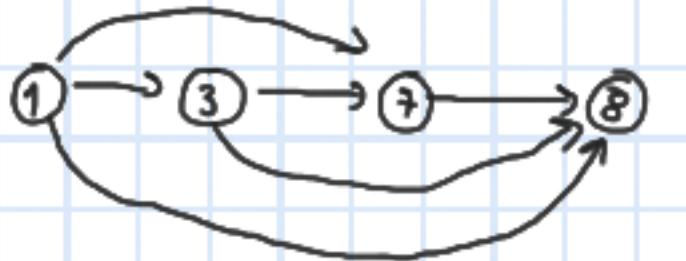


## Sortowanie topologiczne

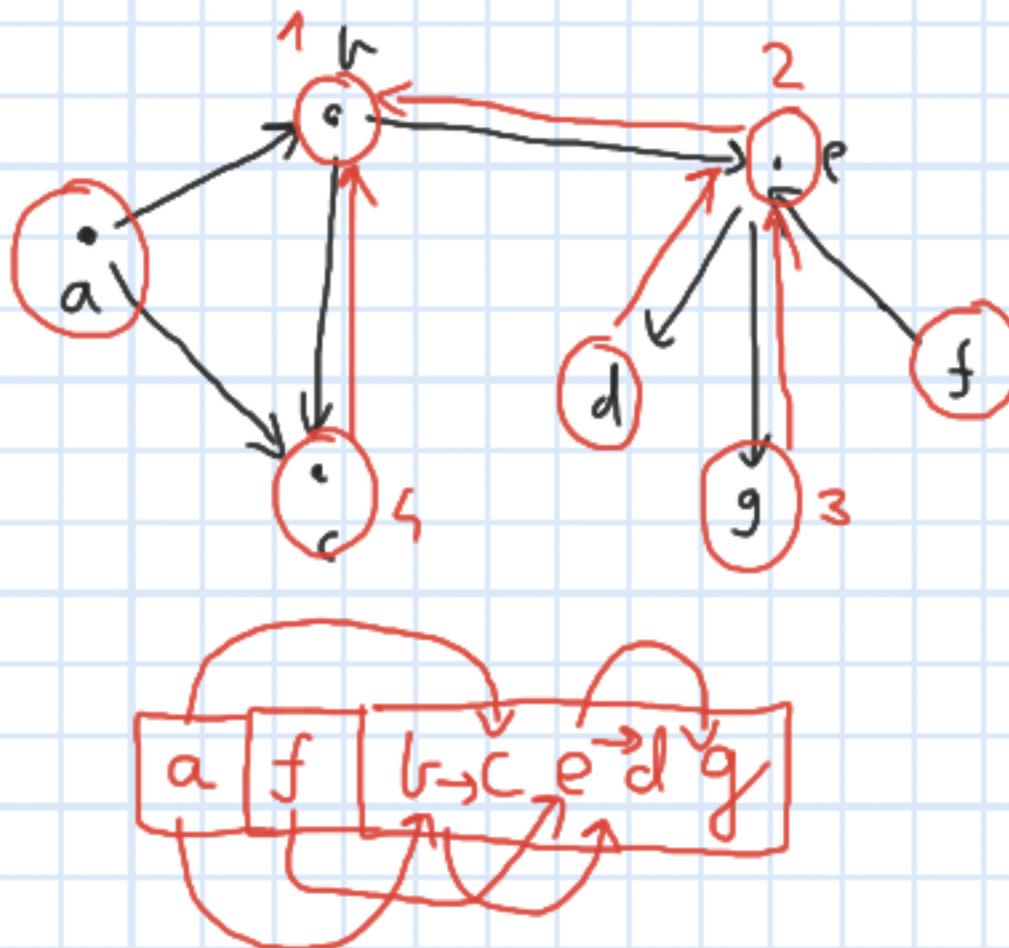
dag - directed acyclic graph

(skierowany grafacykliczny)

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



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



## Algorytm

- uzupełnianie DFS

- po "przebranieniu" każdego wierzchołka dodać go na koniec tworzonej listy

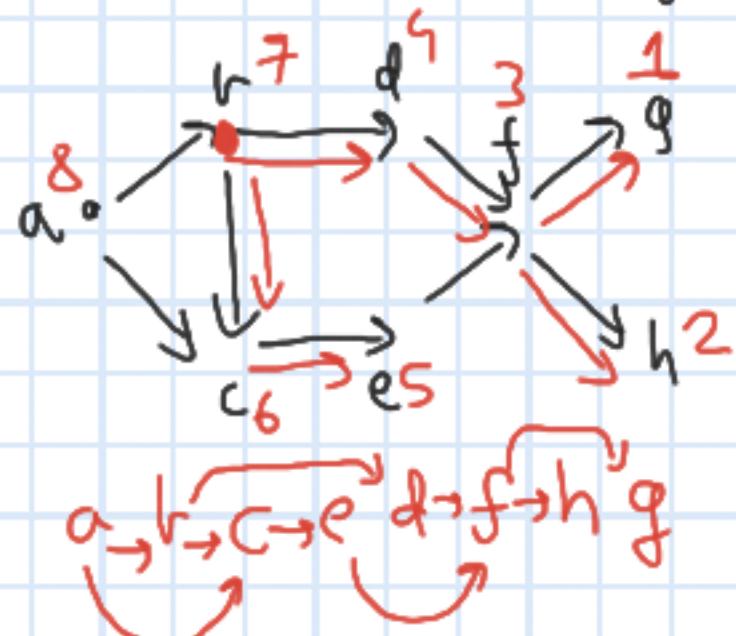
# Algorytmy i Struktury Danych

## 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

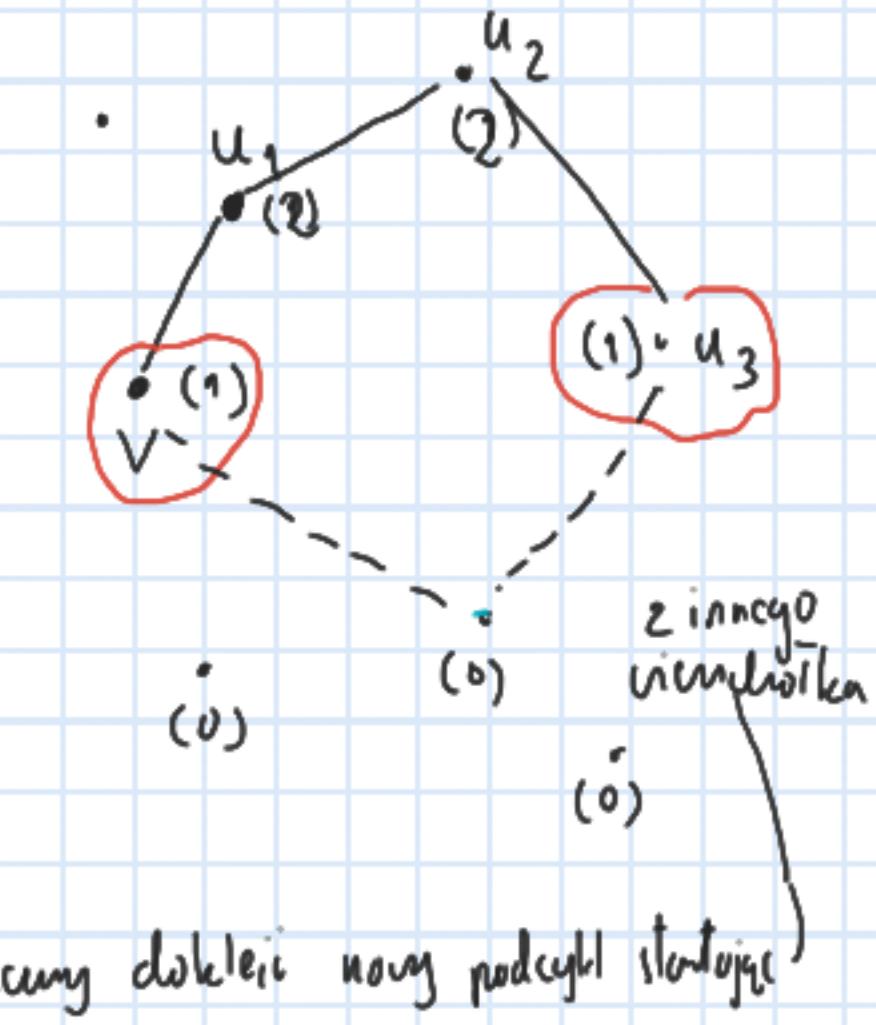
tw Graf nieskierowany i spójny posiada cykl Eulera  
której kiedyś jego wierzchołek ma parzysty stopień

"dowód"

Wavneki liczące : oznacze

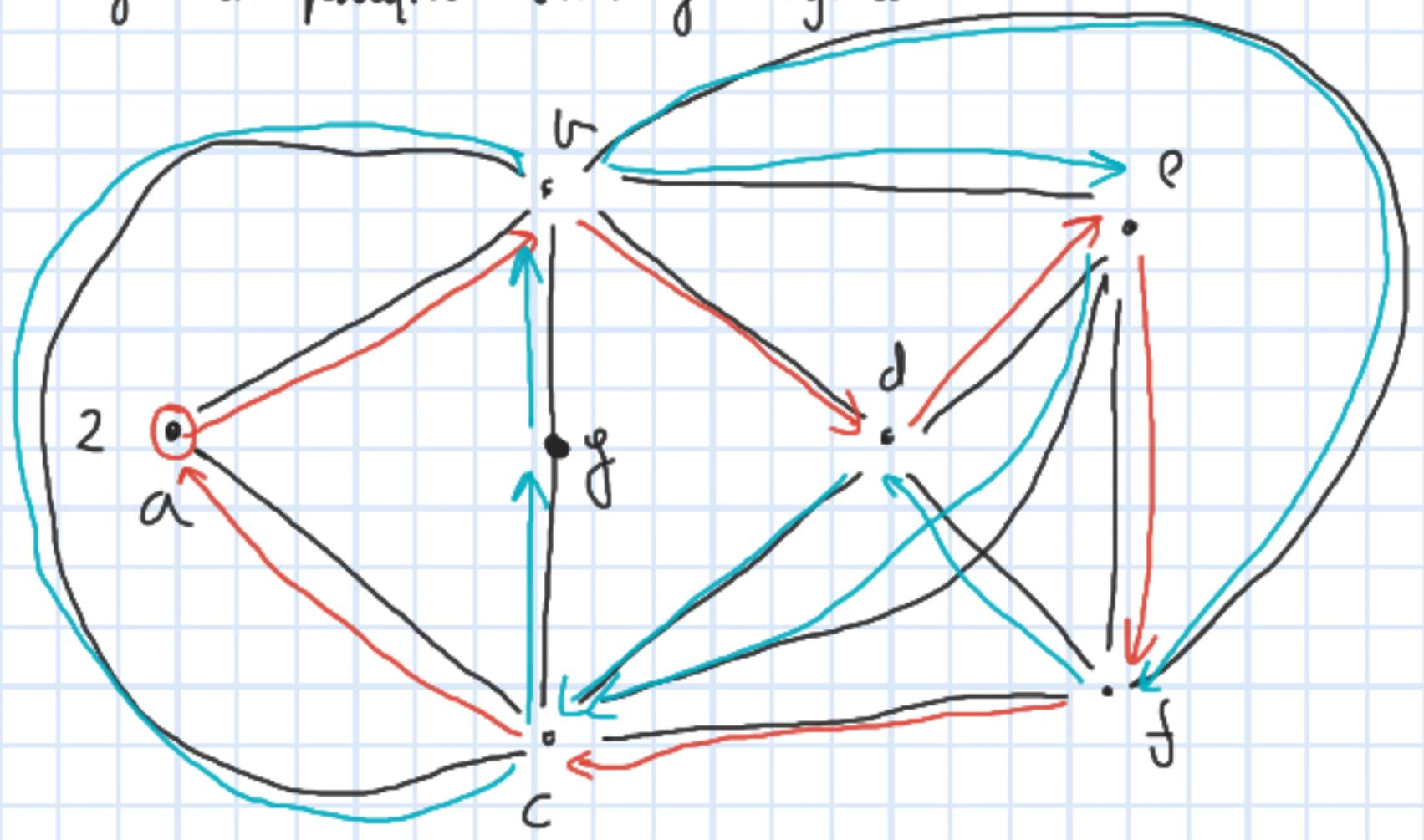
Wavneki wystawiające :

- 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 graf

ma cykl Hamiltona jest problemem NP-zupełnym

### ③ Silnie spójne składowe w grafie skierowanym

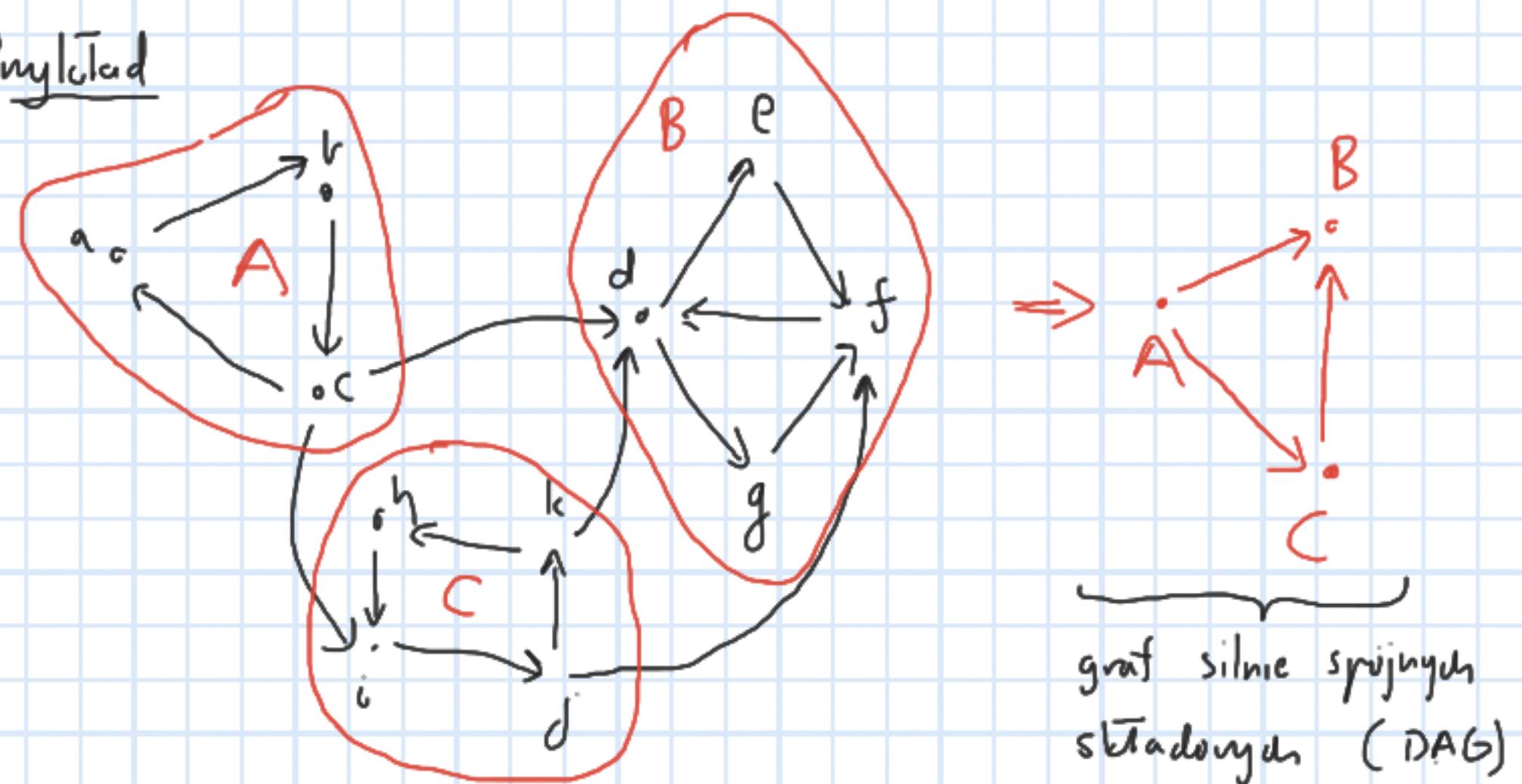
def Niech  $G = (V, E)$  będzie pewnym grafem

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 przetworzenia

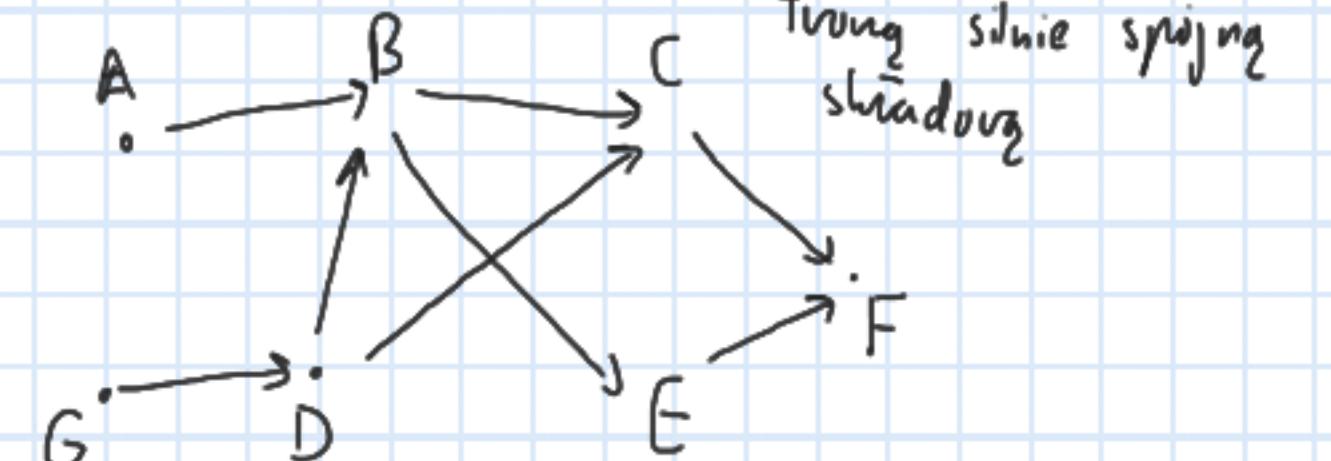
2. Odwróci kierunek wszystkich krawędzi

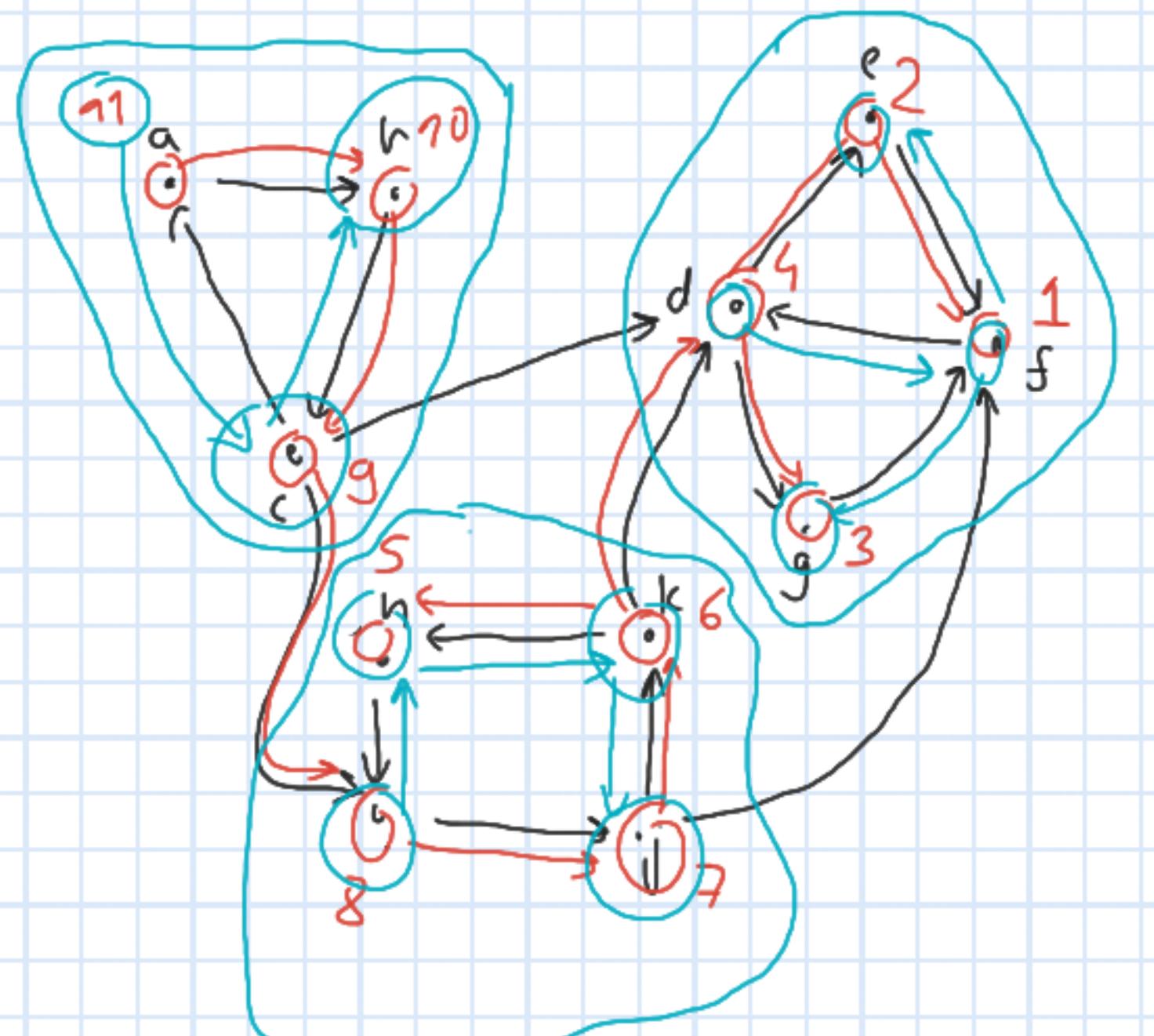
3. Wykonaj DFS, zaczynając od stacjonarnego wierzchołka

w kolejności malejących czasów przetworzenia

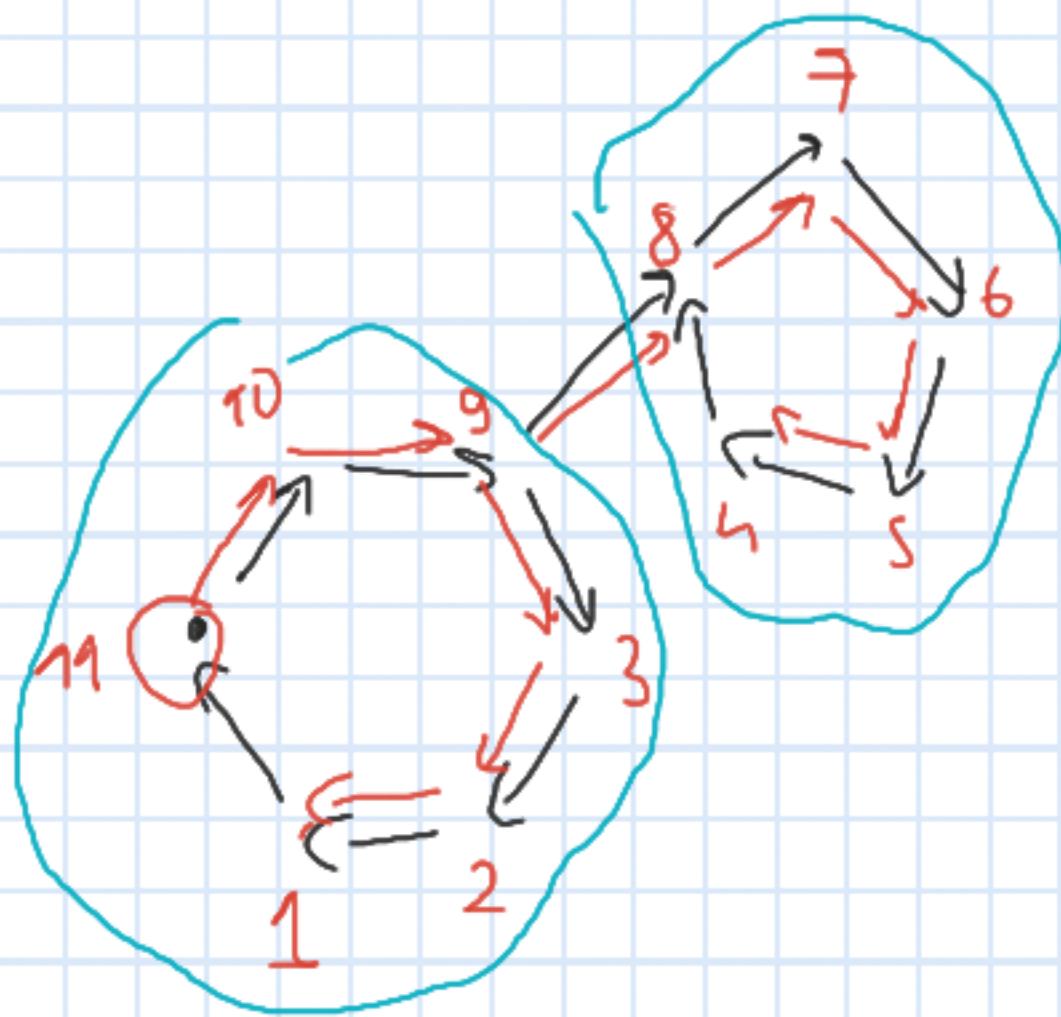
z poprzedniego wykonania

#### Intuicja





Punktland



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 rozspaja grafu)

$e$  nie leży na żadnym cyklu  $\Rightarrow$   $e$  jest mostem

(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

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

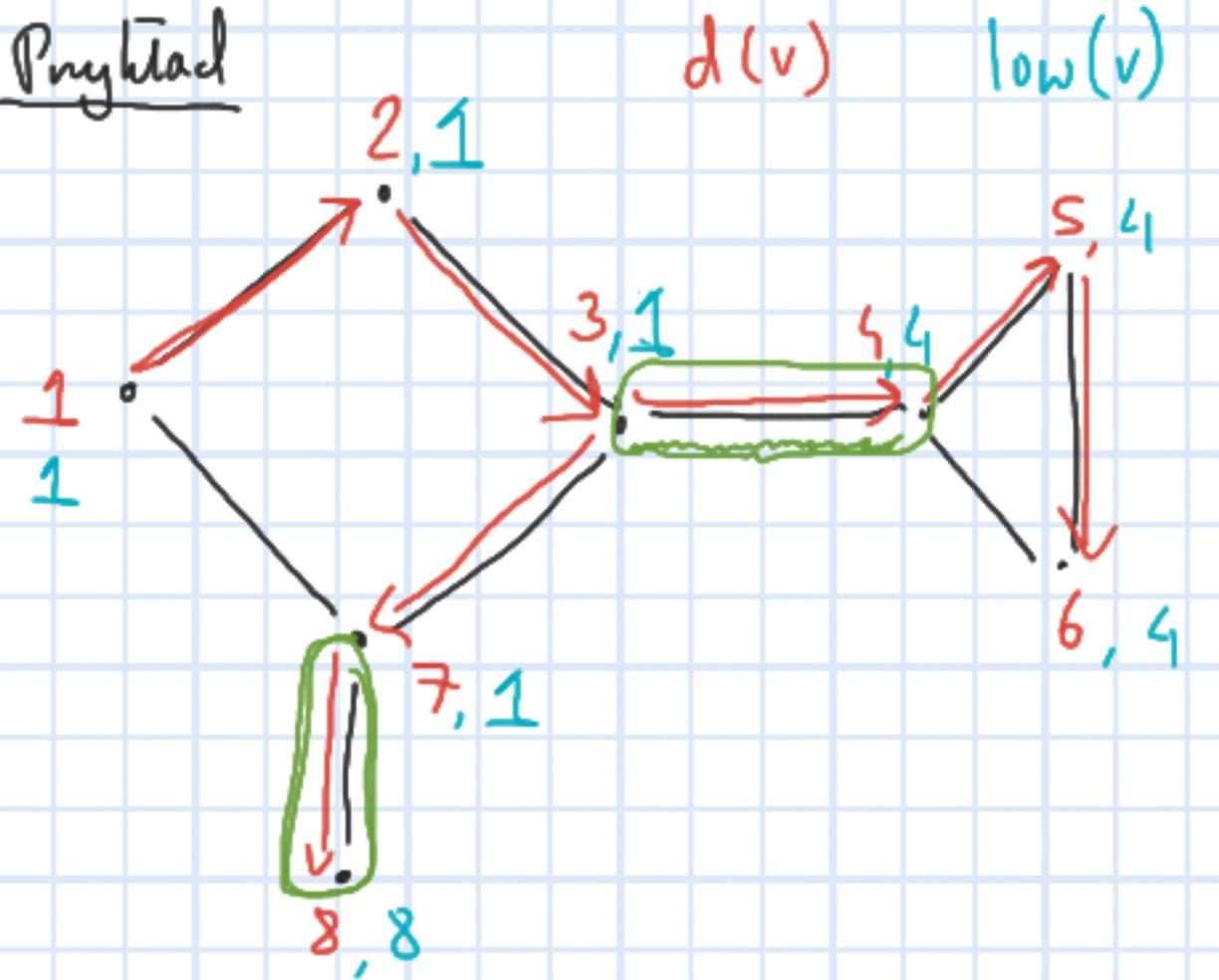
$$\min_{\substack{u \text{ to dzieci } v \\ u \text{ dające DFS}}} (\text{low}(u))$$

wzgl.  $v$  u mniejsze  
DFS

3. Mosty to krawędzie  $\{v, p(v)\}$   
takie gdzie  $d(v) = \text{low}(v)$

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

Prybilad

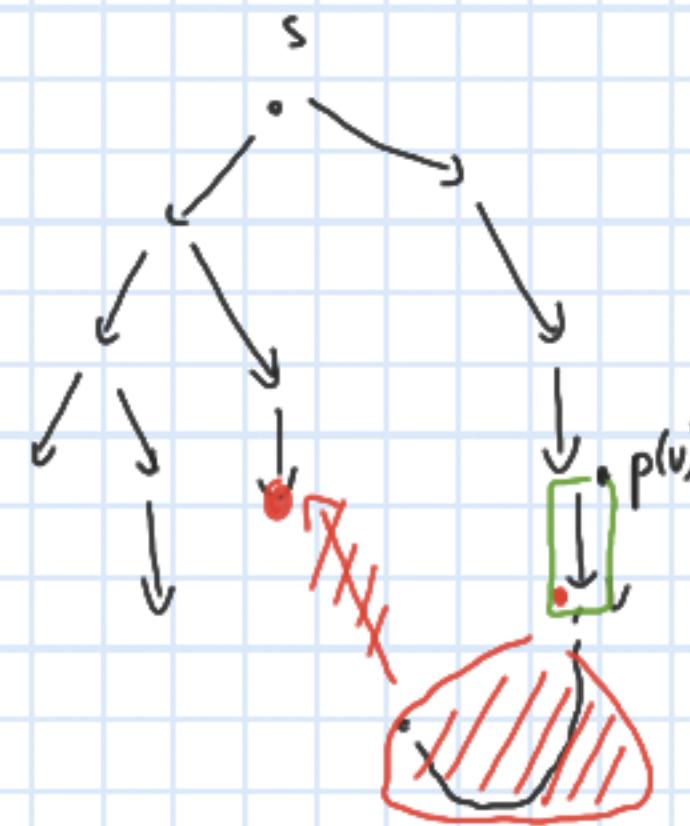


Jeli  $d(v) = \text{low}(v)$  to kresydt

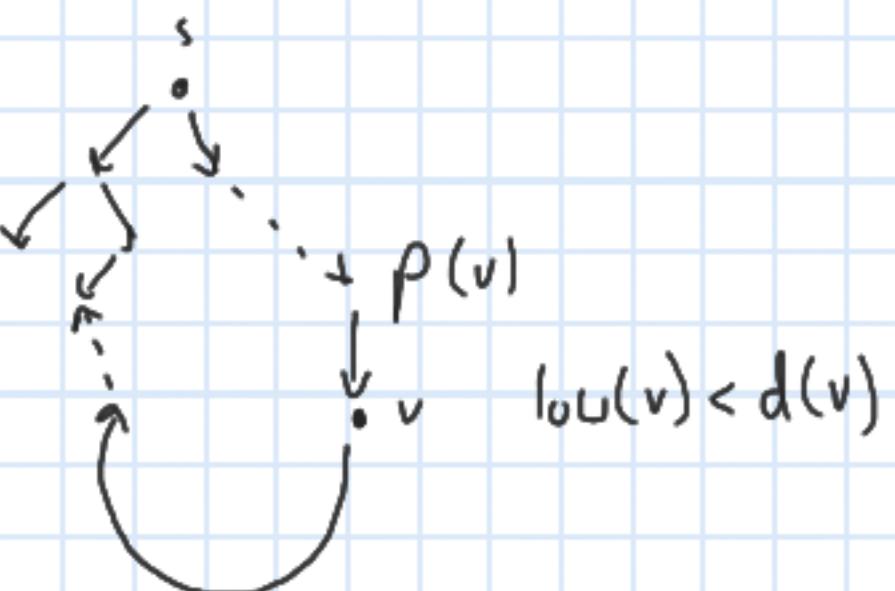
$\{p(v), v\}$  jst mostem

takej kresydt  
do  $p(v)$  lbo  
viendostka  
o mizgu  
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

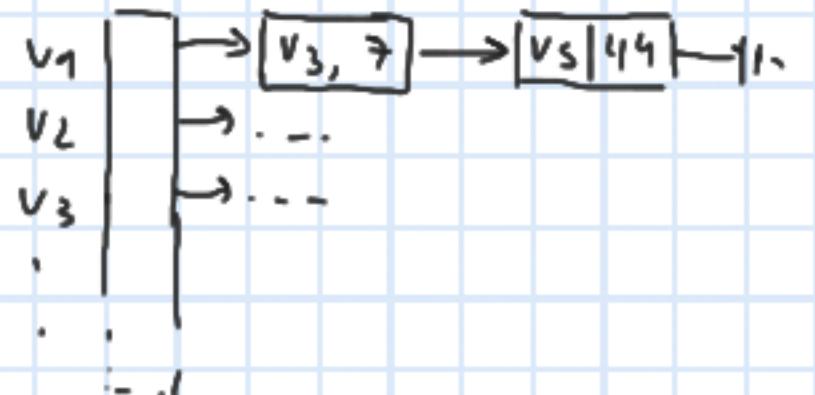
$$w: E \rightarrow \mathbb{N} \quad (\mathbb{Z}, \mathbb{R})$$

### Reprezentacja macierzowa

w - matryca wag

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

### Reprezentacja listowa

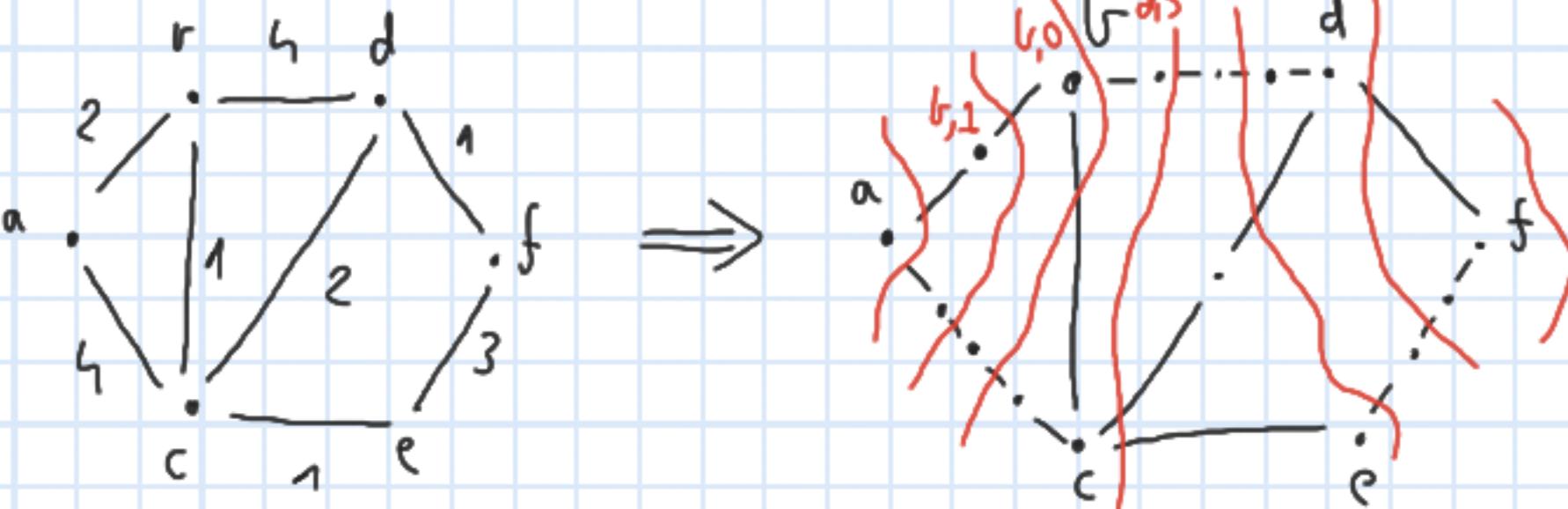


### Problem znajdowania najkrótszych ścieżek

- 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

przeciwlego 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. Umiesci wszystkie wierzchołki w kolejce priorytetowej z oszacowaniem odległości  $\infty$

} u praktyce realizowane inaczej

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( $u, v$ ):

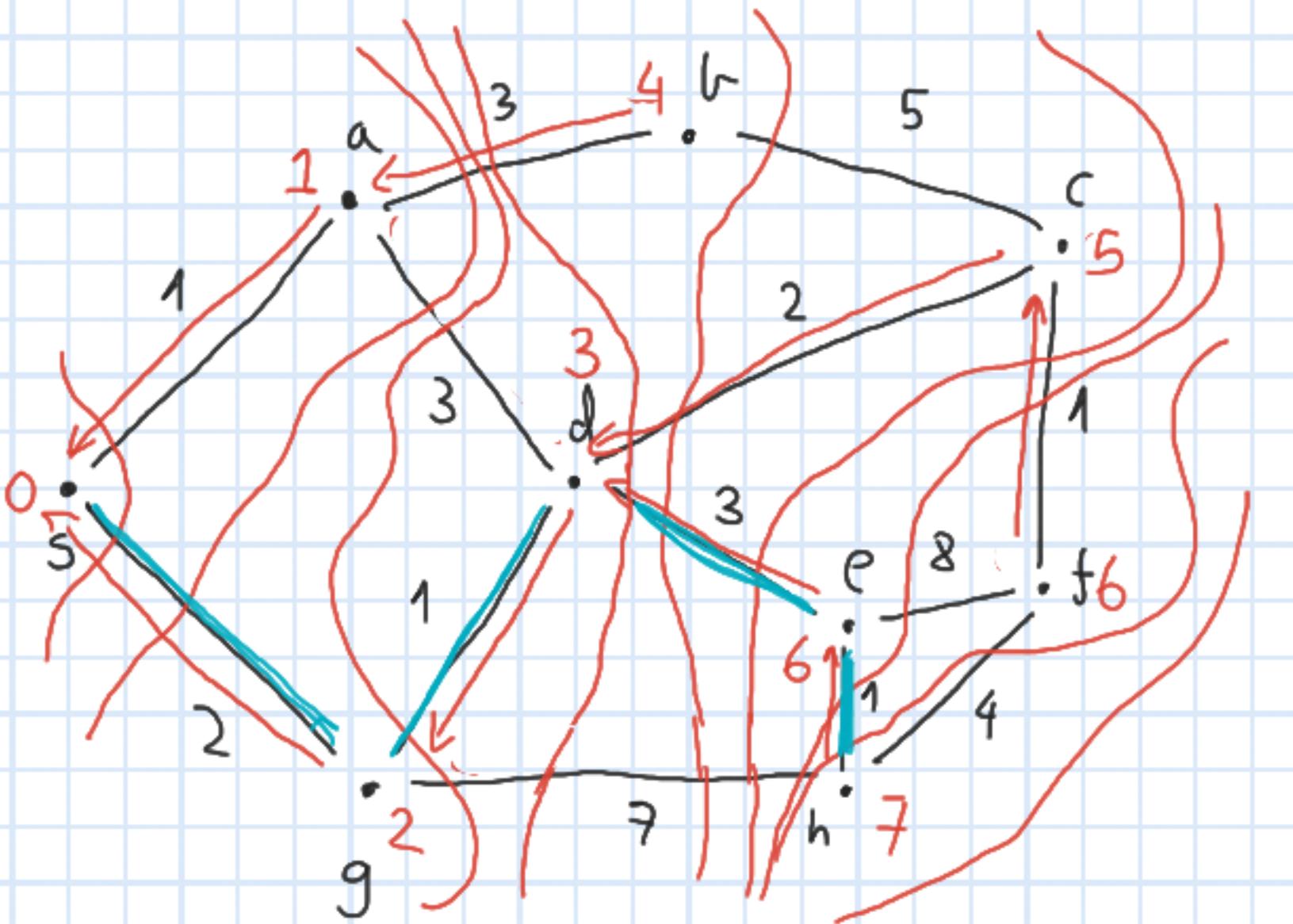
if  $v.d > u.d + w(u, v)$ :

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

$v.parent = u$

algorytm Dijkstry dla rep. listowej  
z kopcem binarnym

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

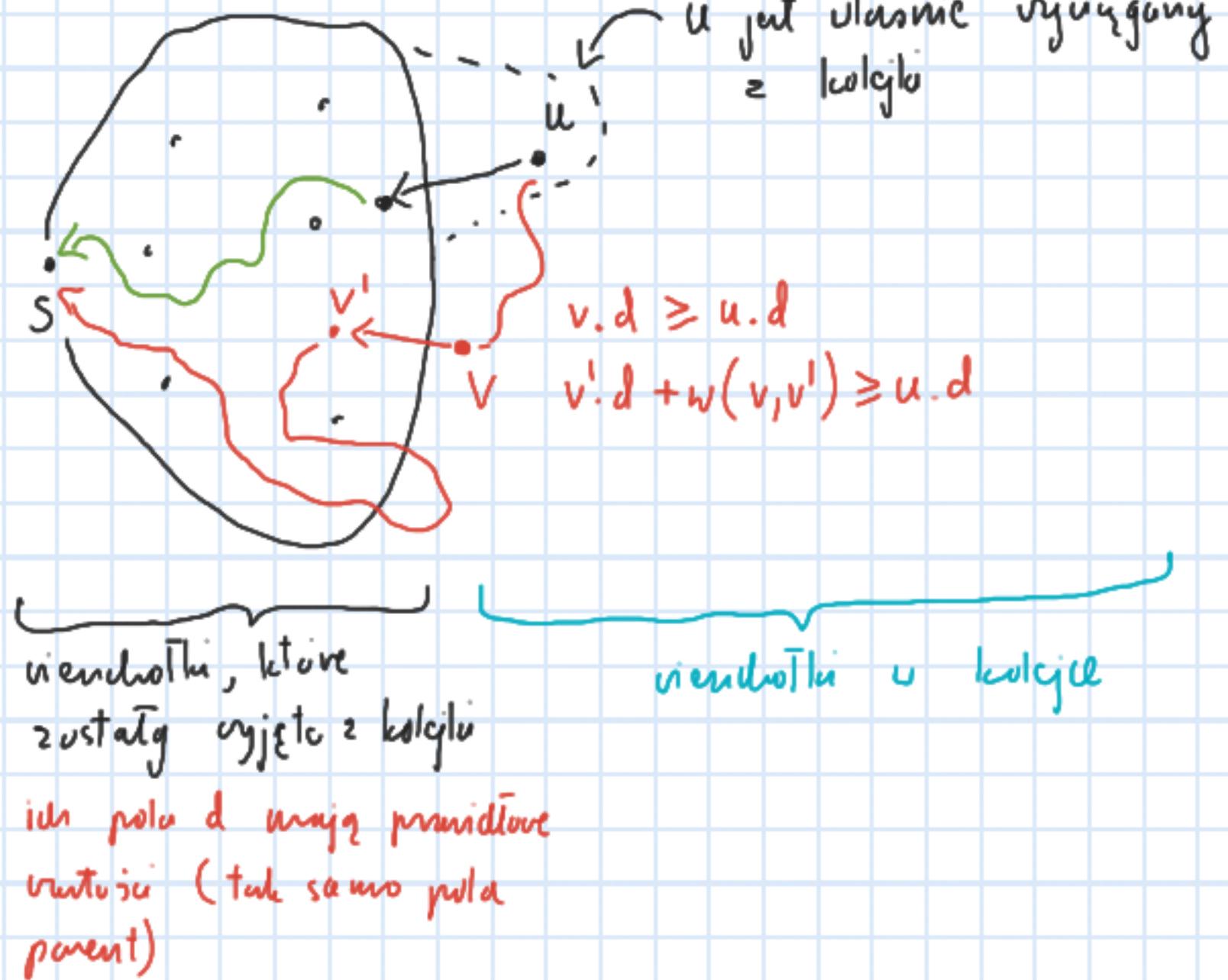
z kolejką, to jego pula  $u.d$  zawsze dłuższa

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

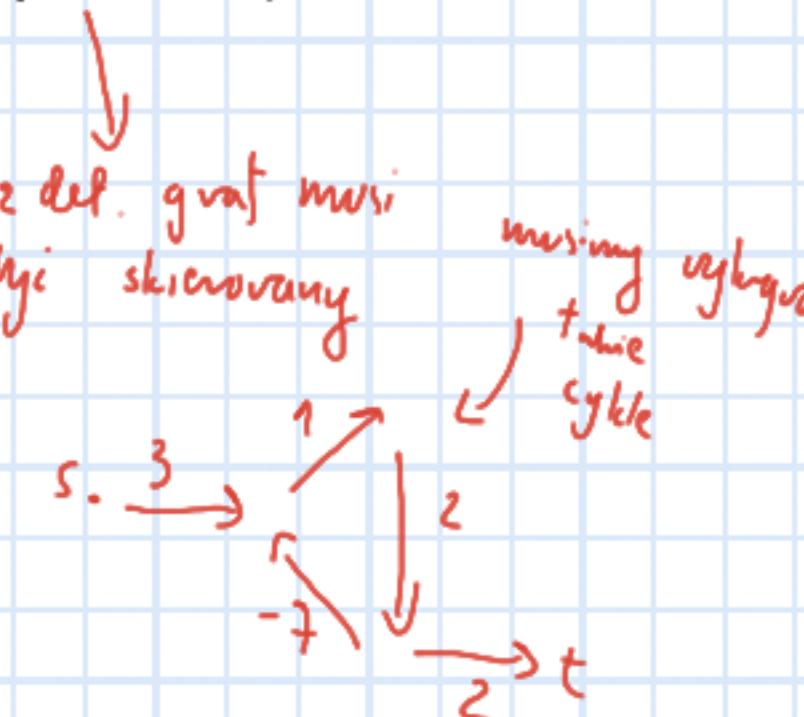
## ① Inicjalizacja

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  $i$ :

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

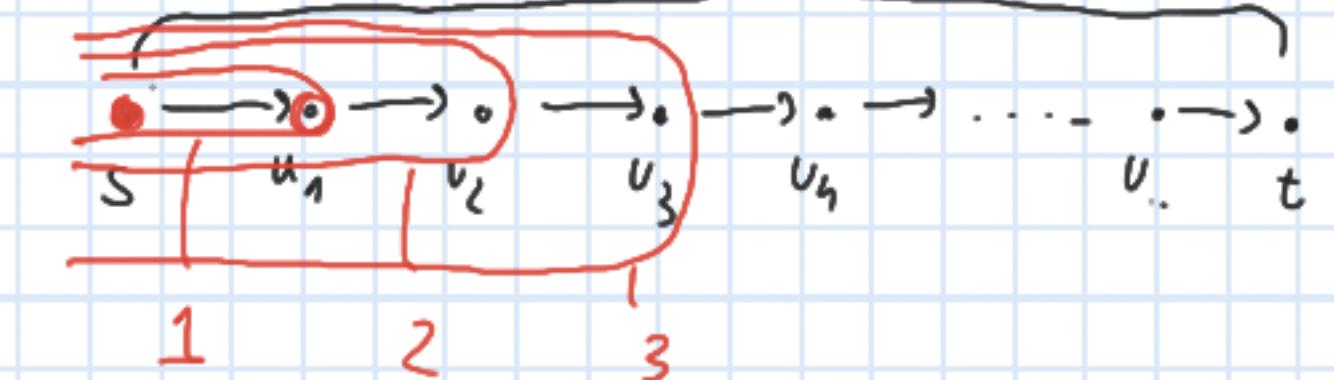
Po zsumowaniu tych nierówności:

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

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

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 najkrótszą ścieżkę

Czyli ③ wykrywa ujemne cykle

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

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

Sprowadzić

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

Najlepiej umieścić między kredy  
pewne wierszotkiów

## Konvenya

W specjalizowanych algorytmach  
stosuje się reprezentację macierzy

$D[u][v]$  - dt. najmniejszy suci  
z  $u$  do  $v$

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

## Algorytm Floyd - Warshalla

Idea: jeśli znamy najkrótsze ścieżki między każdą parą wierzchołków, które jako unikalne wierzchołki

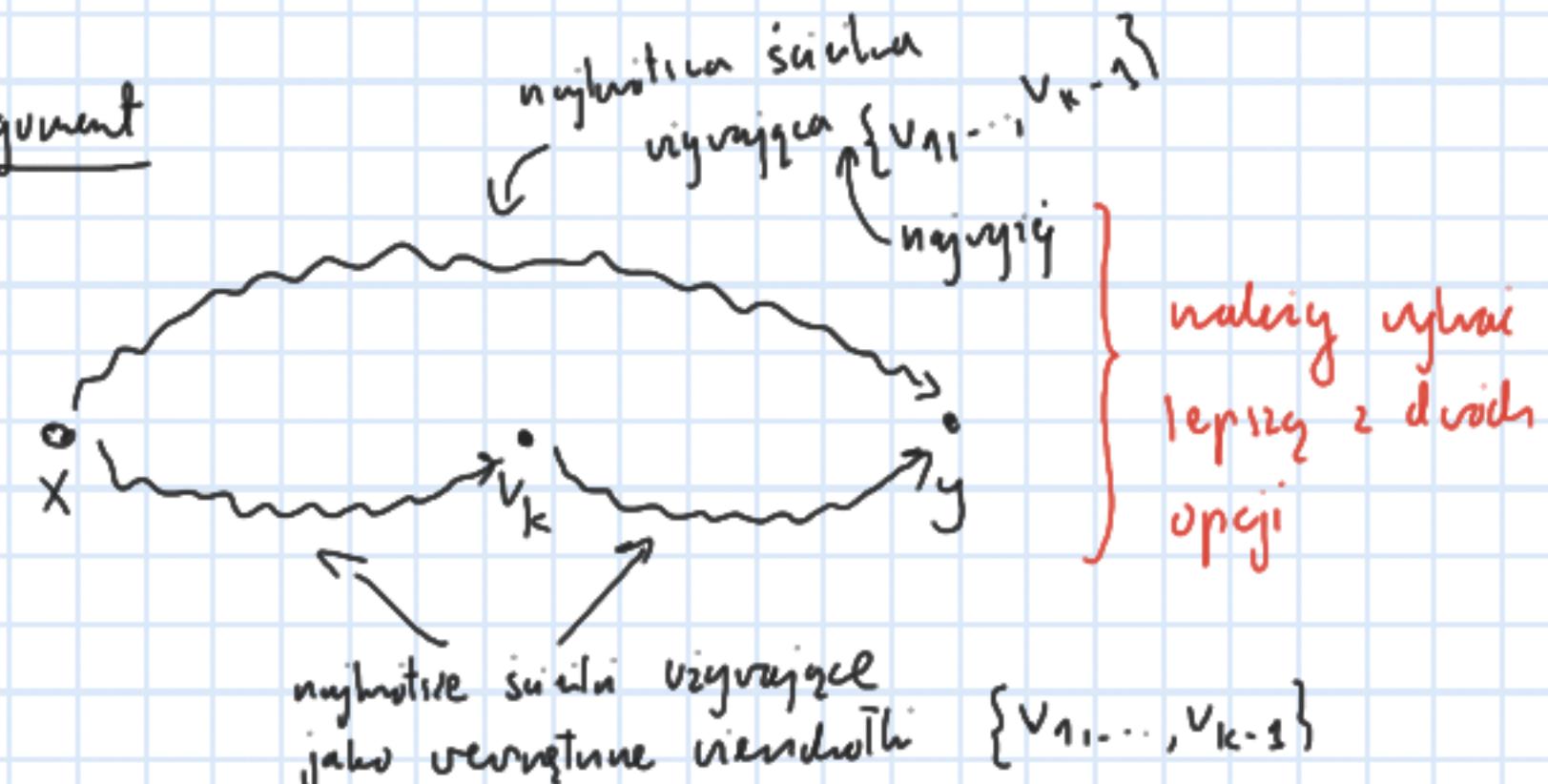
Uiyuy

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

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

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

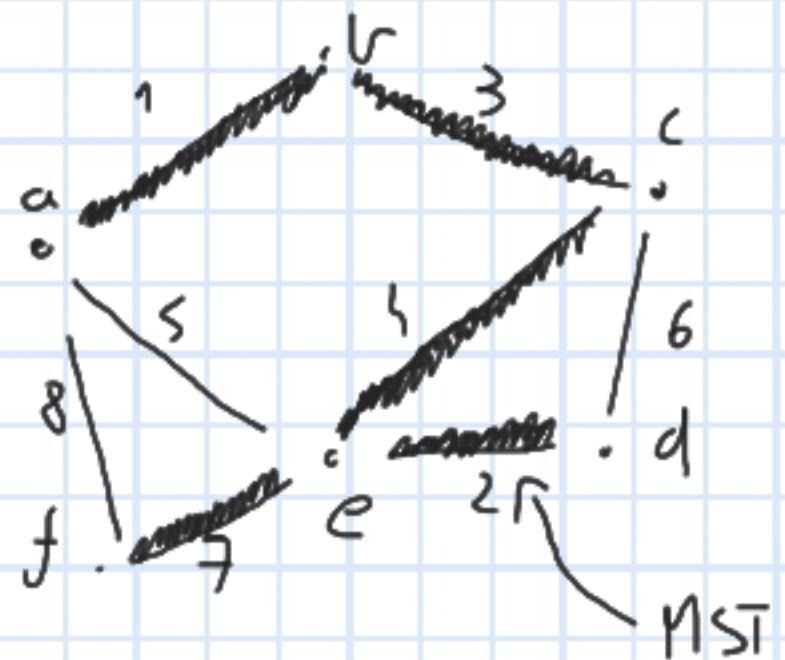
U implementacjach  
wykonamy tylko  
jednej macy

## 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

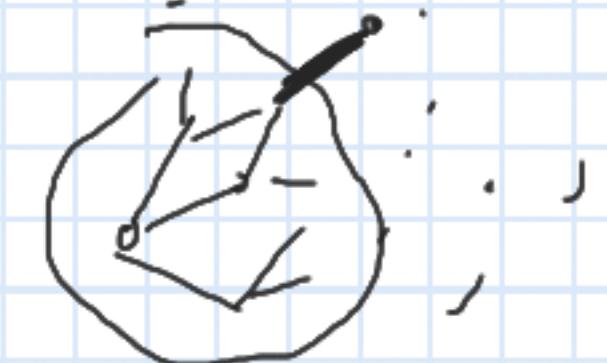
$$A := A \cup \{e\}$$

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  $\infty$

2. Zmieni wagę v na 0

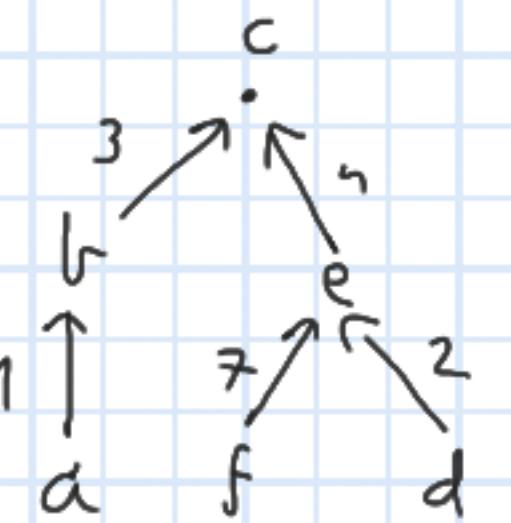
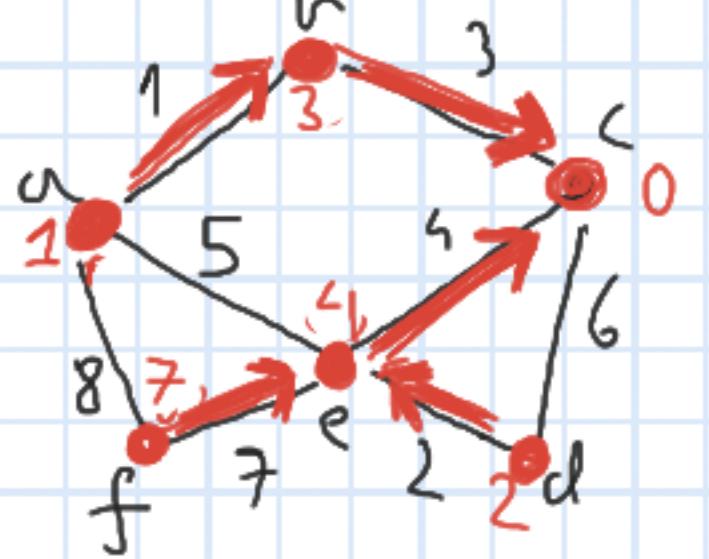
3. Później wierzchołki są w kolejce:

- 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)

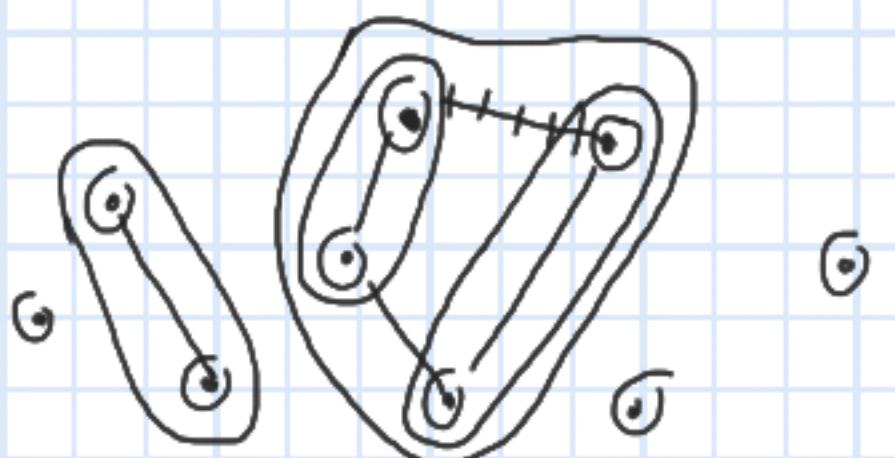
$O(E \log V)$

## Pozycja

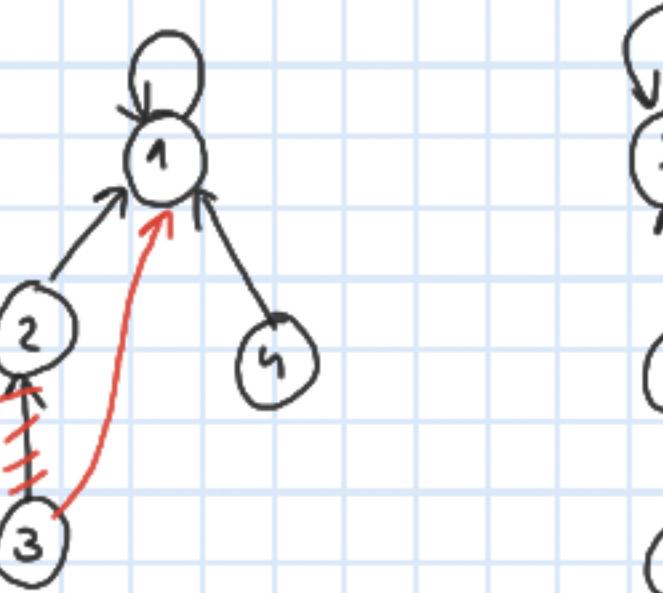


Rodzina zlioniu roztagonych - struktura  
Find/Union

## 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
```

```
def findset(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 lustra zastosowan log  
do m postulowa by wynik  
spadku do  $\leq 1$

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

```

def 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

```