

ASD - Wykład 9

Algorytmy grafowe

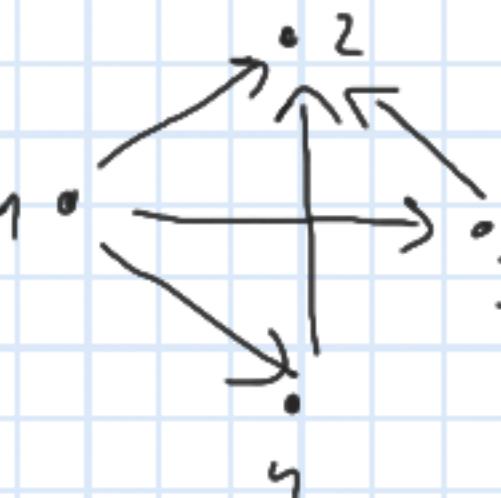
Graf skierowany:

$$G = (V, E)$$

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

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

na ogół nie dopuszcza się krawędzi
 (u, u)



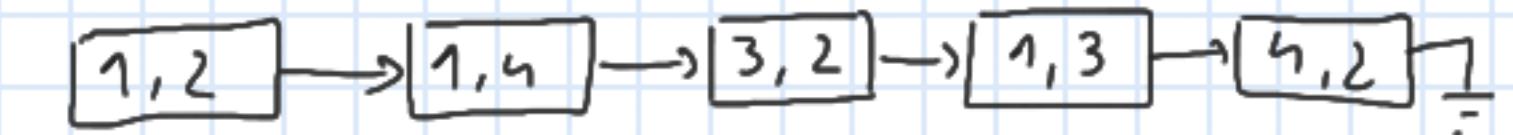
Z kązdy krawędzią / wierzchołkiem
można zaznaczyć dodatkowe informacje

Graf nieskierowany - krawędzie są zbiorem
dwuelementowym

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

Reprezentacje grafów

Lista / tablica krawędzi

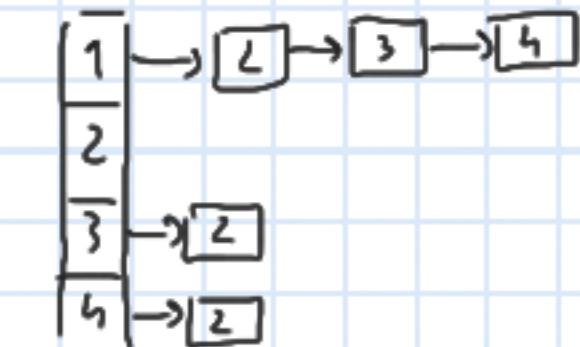


Reprezentacja macierzowa

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

ten fragment
wygenerowany dla
grafów nieskierowanych

Reprezentacja przez listy sąsiedztwa



Algorytm BFS (breadth-first search)

Peszulwanie w szerz

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

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

```
Q = Queue()
```

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

```
s.d = 0
```

```
s.visited = True
```

```
s.parent = None
```

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

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

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

```
    for v - sąsiedzi u:
```

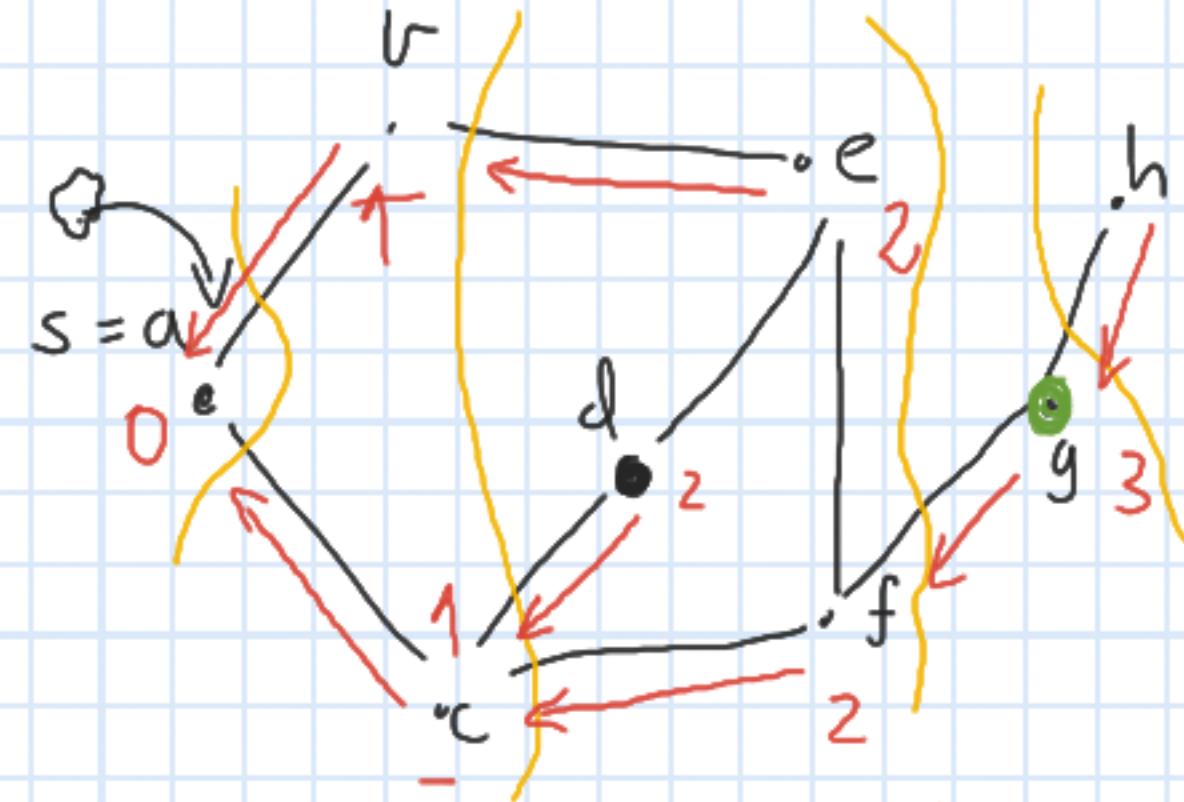
```
        if not v.visited:
```

```
            v.visited = True
```

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

```
            v.parent = u
```

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



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

Złożoność

rep. macierzowa

$O(V^2)$

rep. listowa

$O(V + E)$

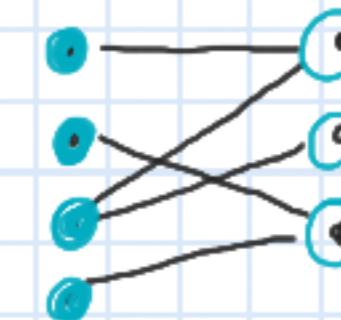
Zastosowania

- najkrótsze ścieżki (w grafie bez wag)

- spójność grafu

- wykrywanie cykli

- divedzimusi



Algorytm DFS (depth-first search)

Pniewzkuwanie w gętce

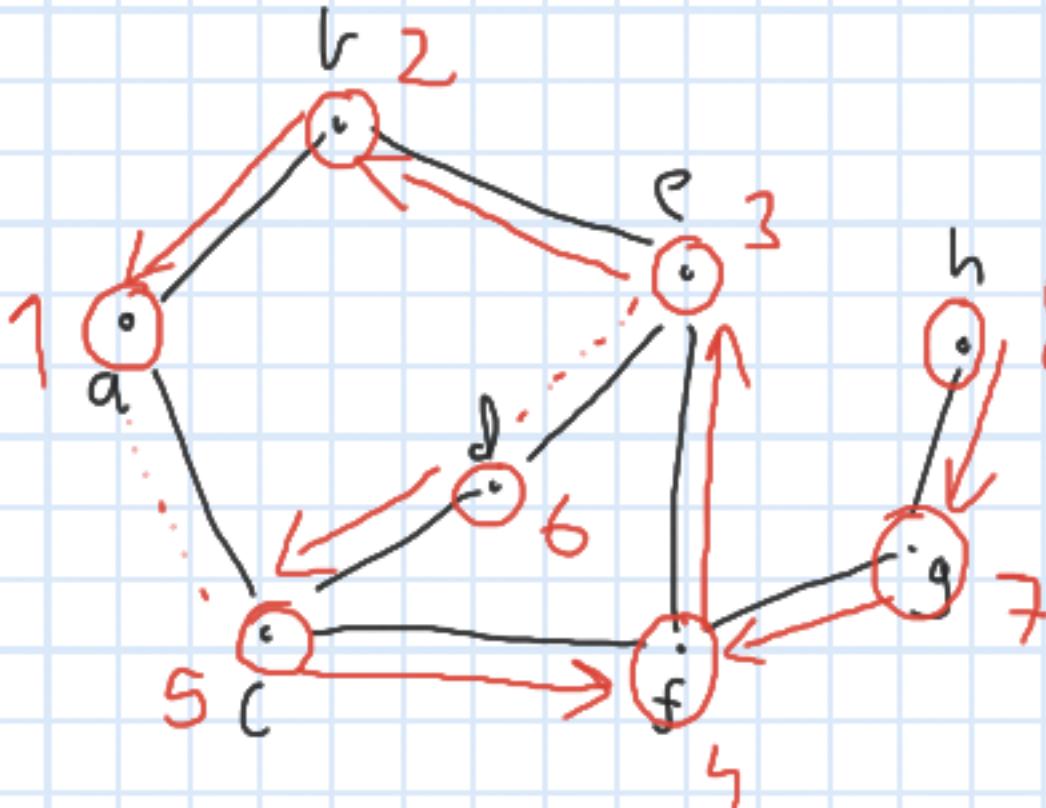
```
def DFS( G ):
    # G = (V, E)
    for v ∈ V:
        v.visited = False
        v.parent = None
    time = 0
```

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

Złożoność

nep. mieniona $O(V^2)$

nep. listowa $O(V+E)$



def DFSVisit(G, u):

nonlocal time

time += 1

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

for v - sąsiad u:

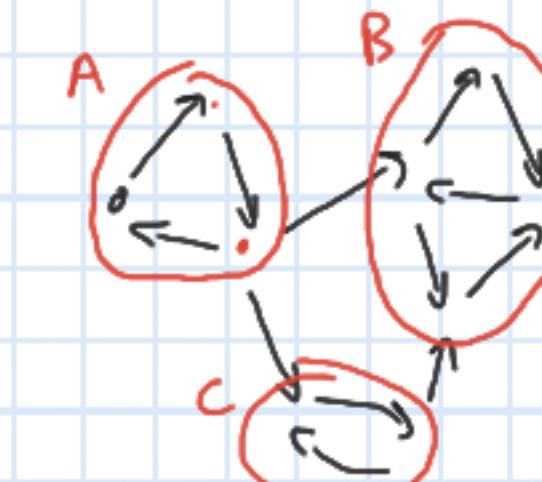
if not v.visited:

v.parent = u

DFSVisit(G, v)

time += 1

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



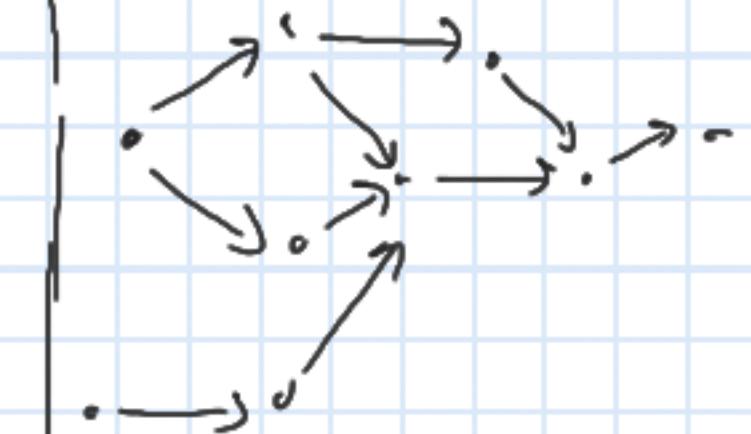
Zastosowania

- spójność

- dwudzielność

- wykrywanie cykli

- sortowanie topologiczne



- cykl Eulera

- silnie spójne

skladowe

- mosty / płat. cuty kolumn



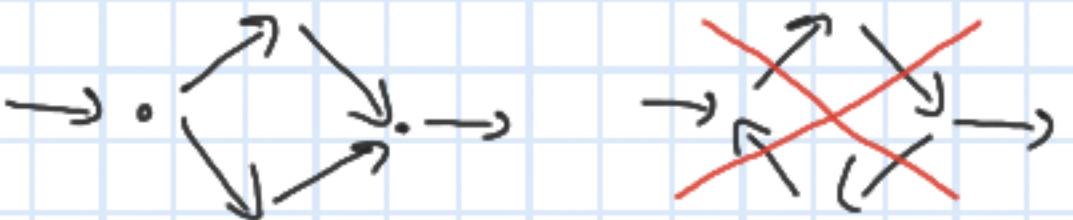
ASD - Uglystad 10

Kolokrium - mugi złożoności

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

Sortowane topologiczne

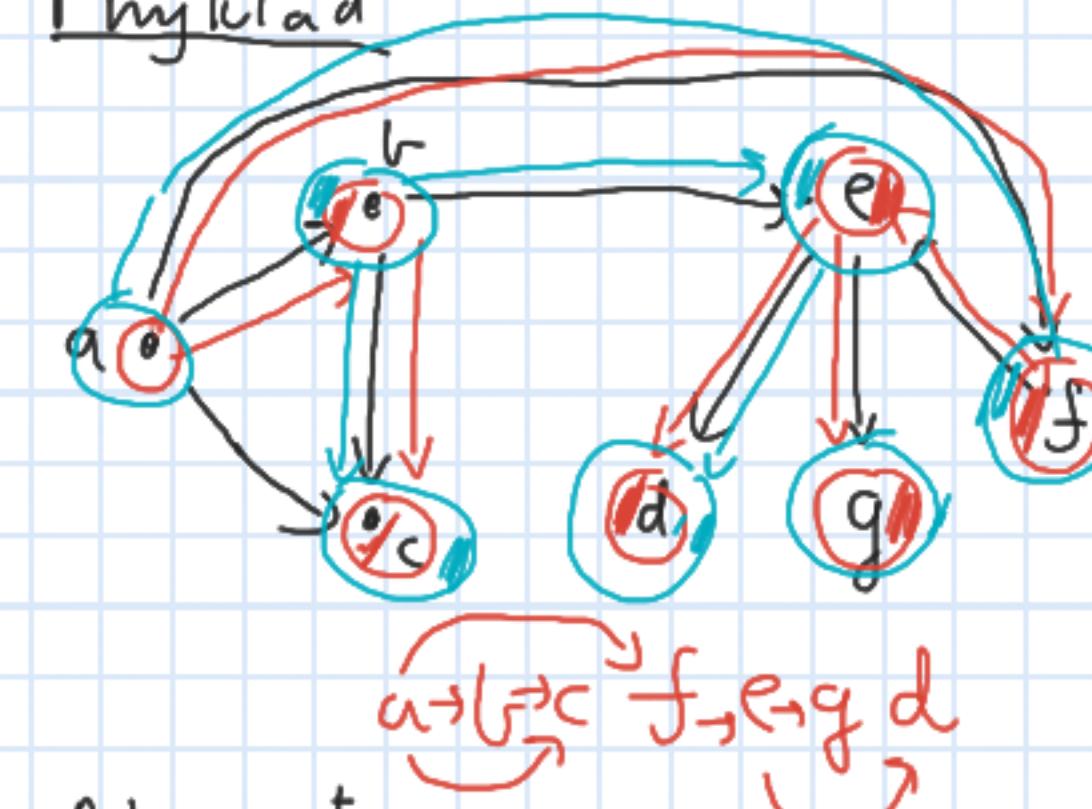
dag - directed acyclic graph



Sortowanie topologiczne dag'u - utwierdzenie

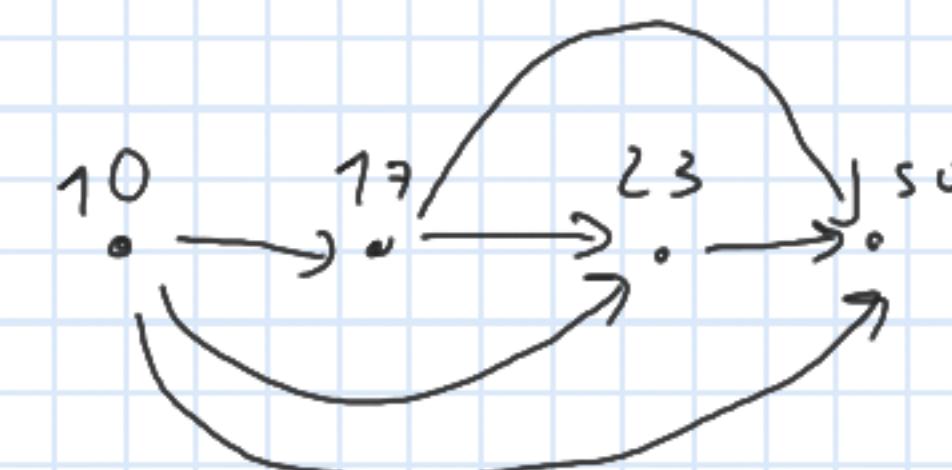
ciendoteków tak, aby wizualnie krajobraz uskazywał "z lewa na prawo"

Pmylutad



Algorytm

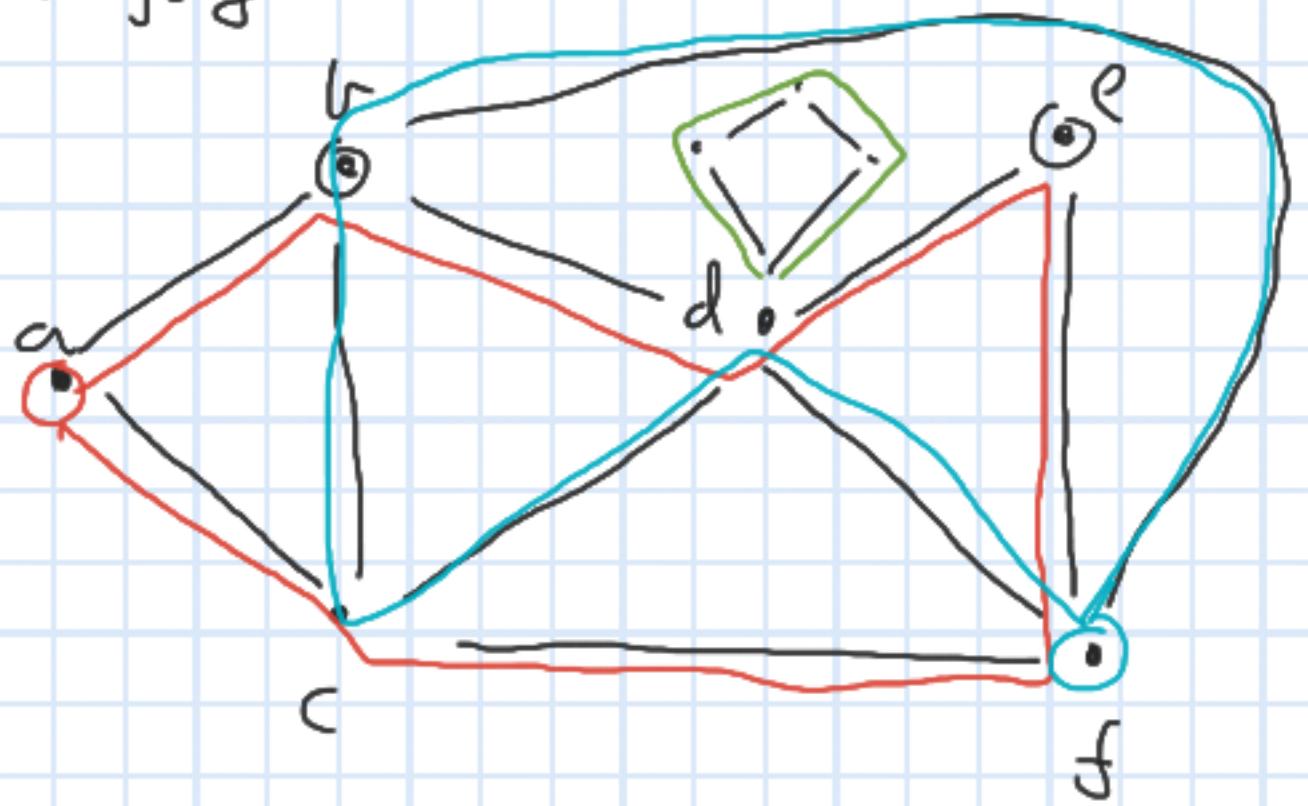
- wchodzimy DFS
- po zwroceniu ciendotki dopisujemy na koniec listy



Cykl Eulera

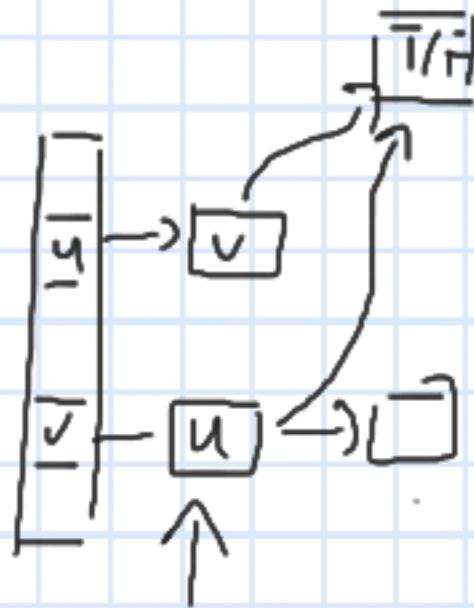
def Cykl Eulera to cykl, który przechodzi przez wszystkie krawędzie, odwiedzając każdą dokładnie raz

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

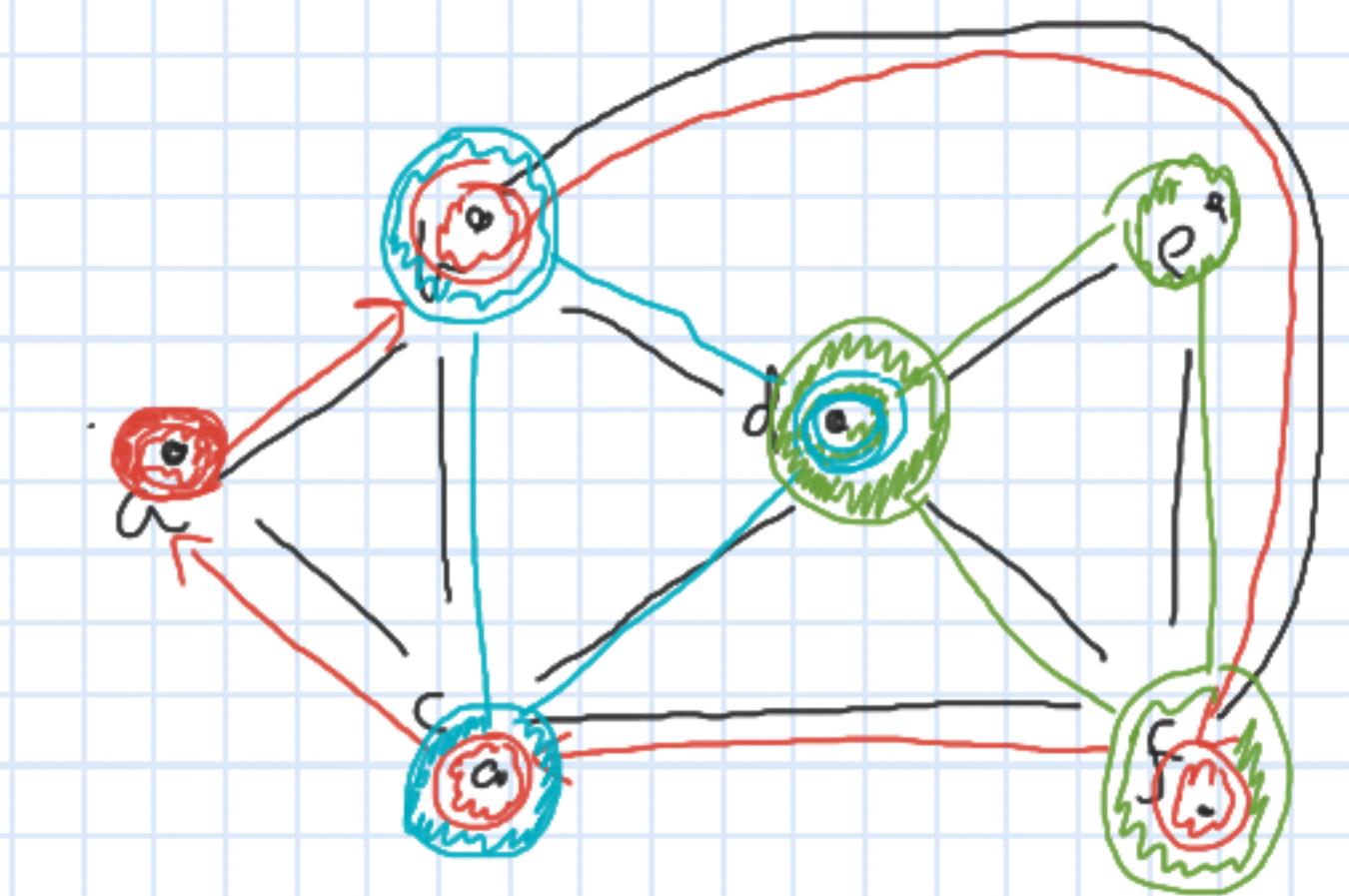


Algorytm

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



$O(V+E)$



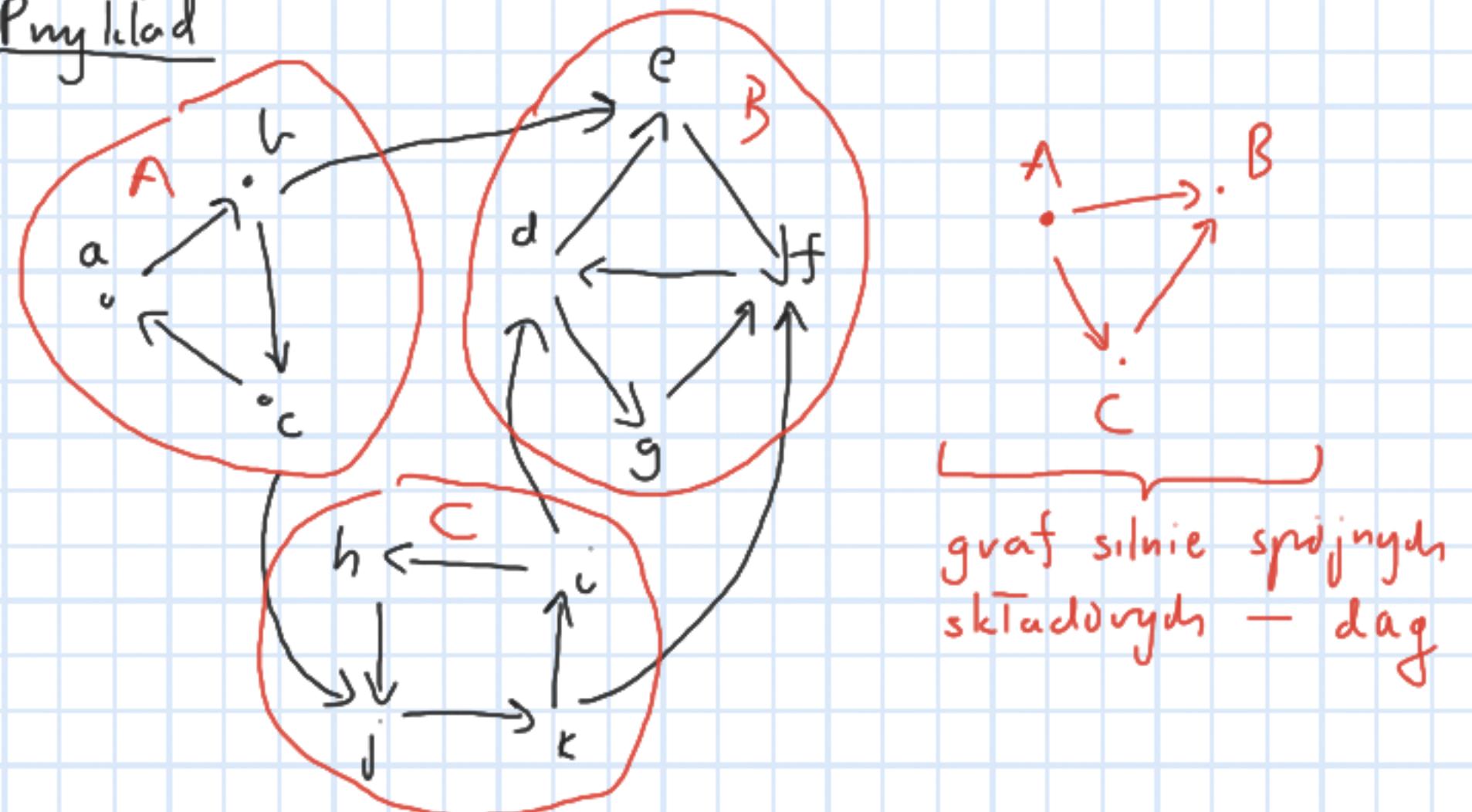
a b f c b d e f d c a

Silnie spójne składowe

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

u do v oraz z v do u .

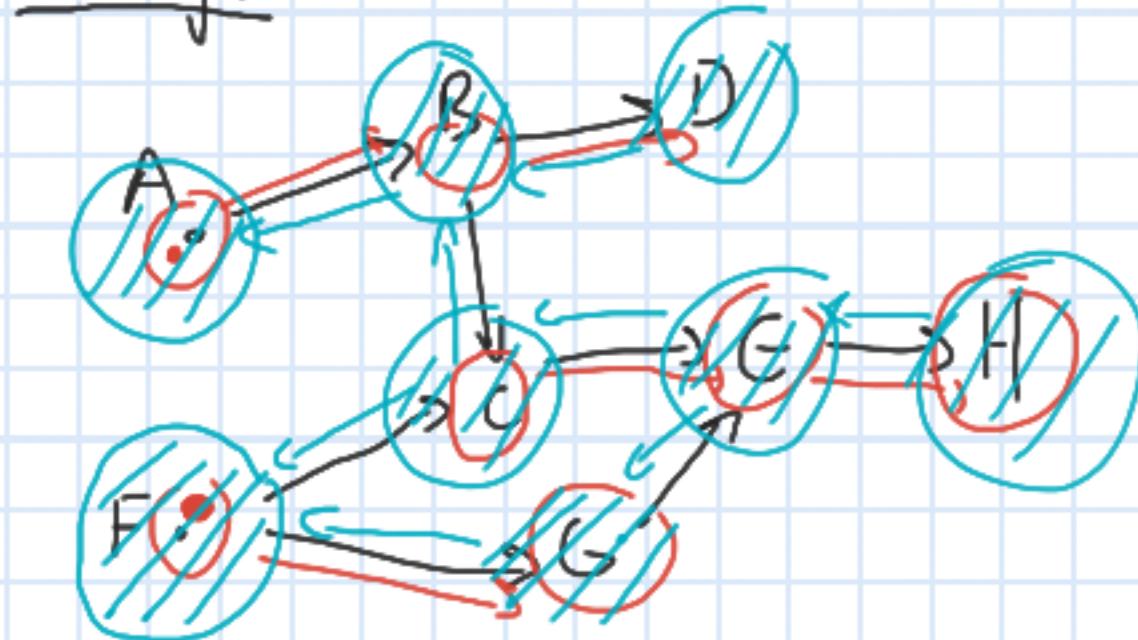
Ponkład

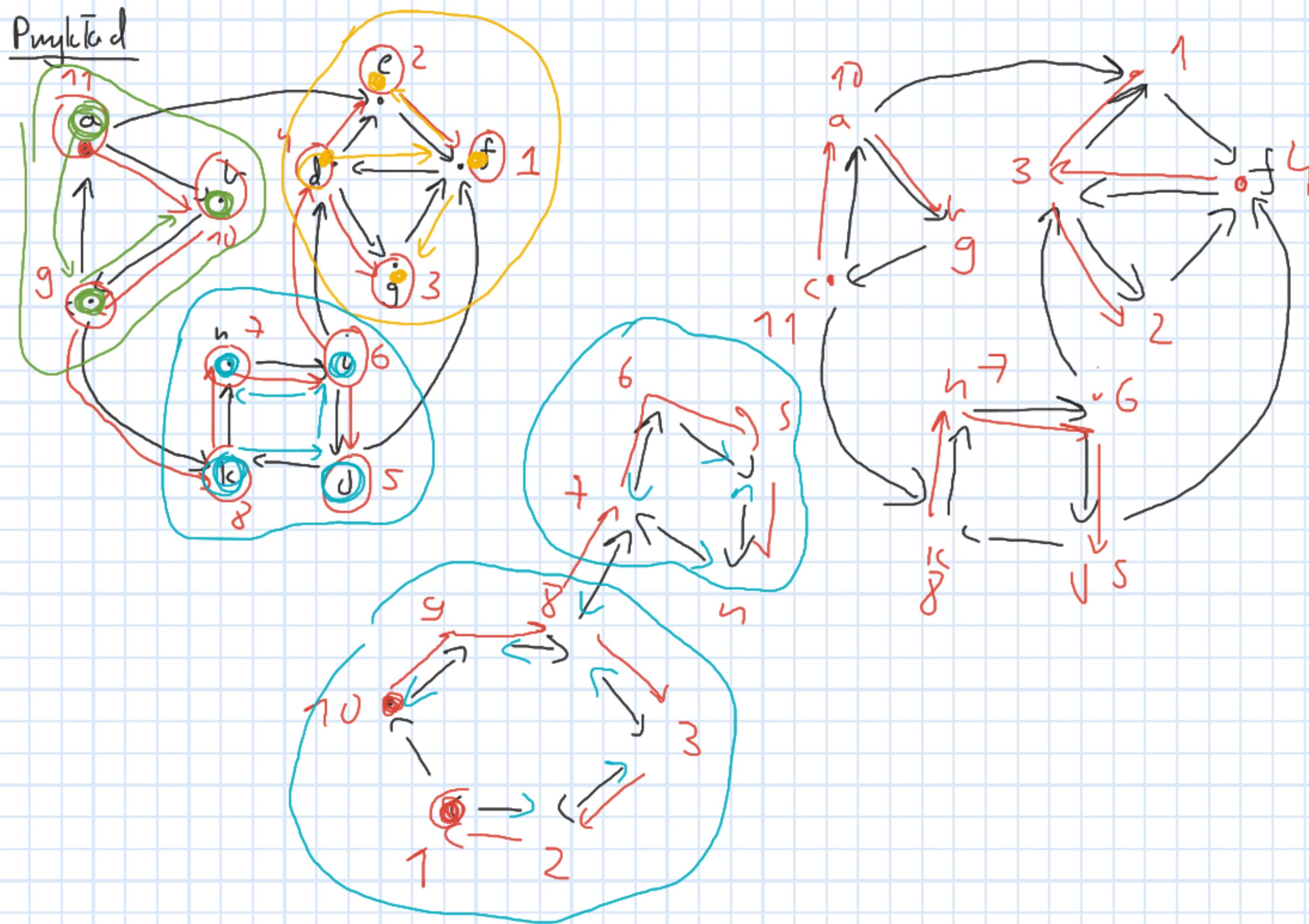


Algorytm

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

Intuicja





Mosty w grafach nieskierowanych

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



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

Dowód

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

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

Algorytm

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

② dla każdego wierzchołka oblicz:

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

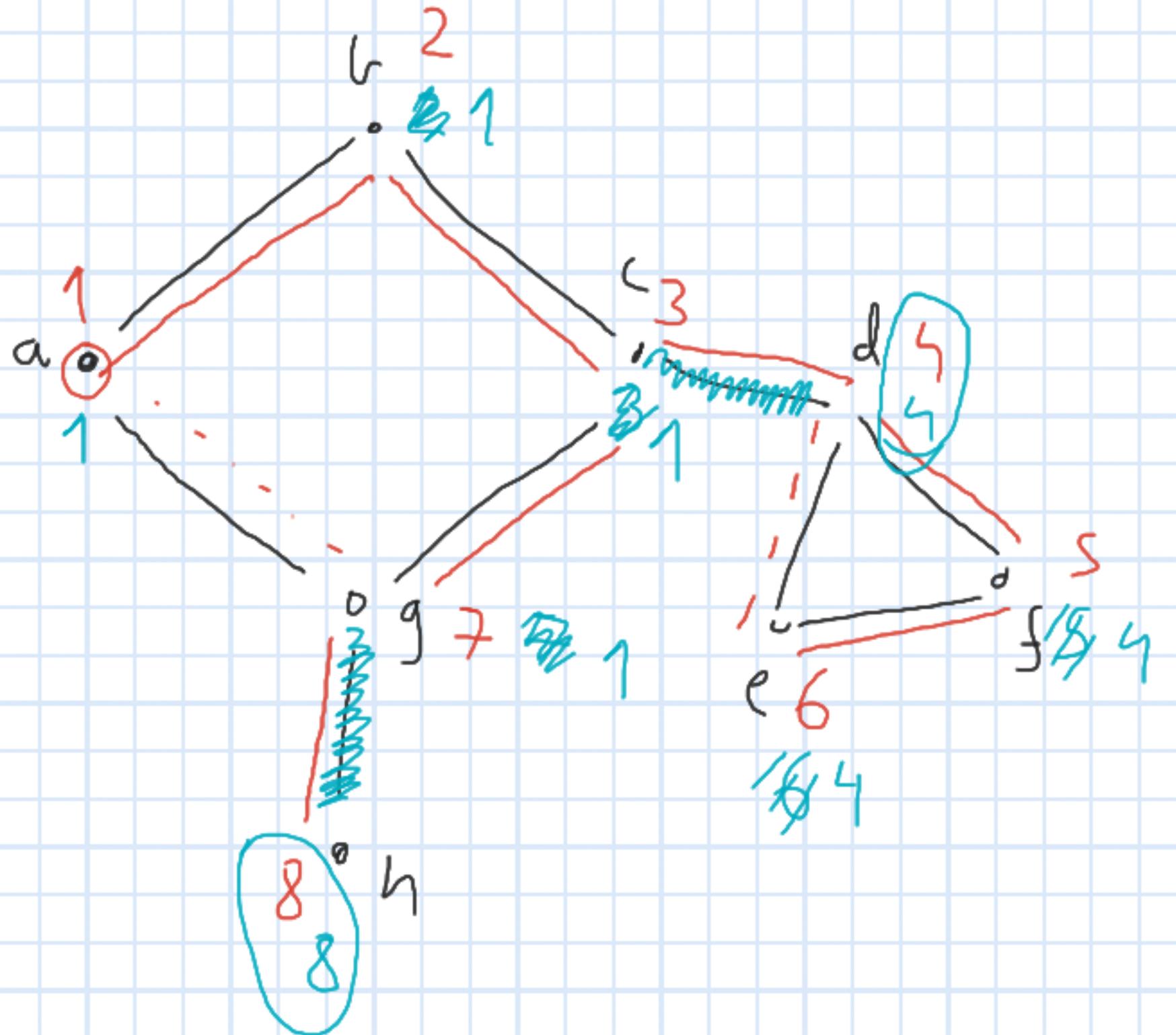
$$\min_{w - \text{drzeka } v \text{ w drzewie DFS}} \text{low}(w)$$

③ Mosty to krawędzie

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

$$d(v)$$

$|v|_U(v)$



ASD - Wykład 11

def Krawędź e w spójnym grafie

nieslurowanym jest mostem jeśli jej usunięcie rozspojnią graf

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

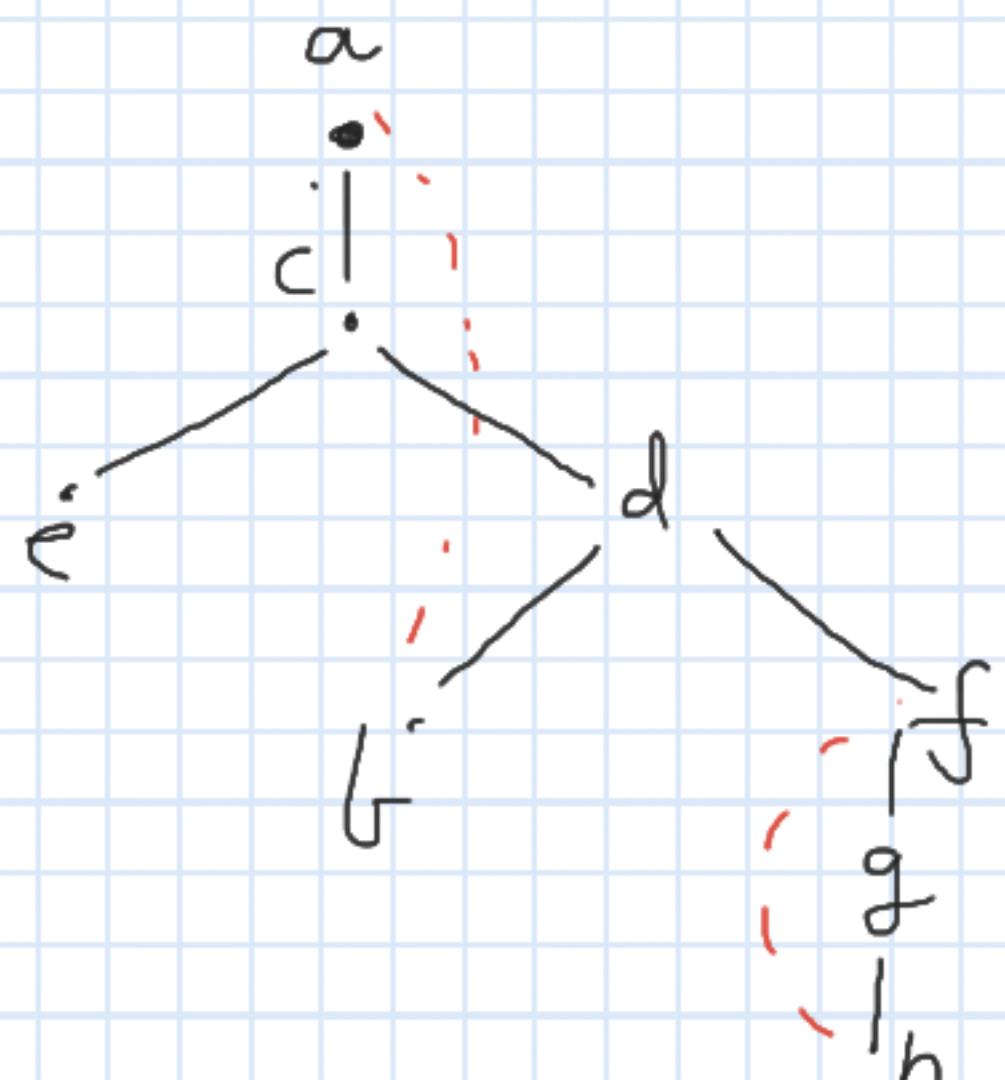
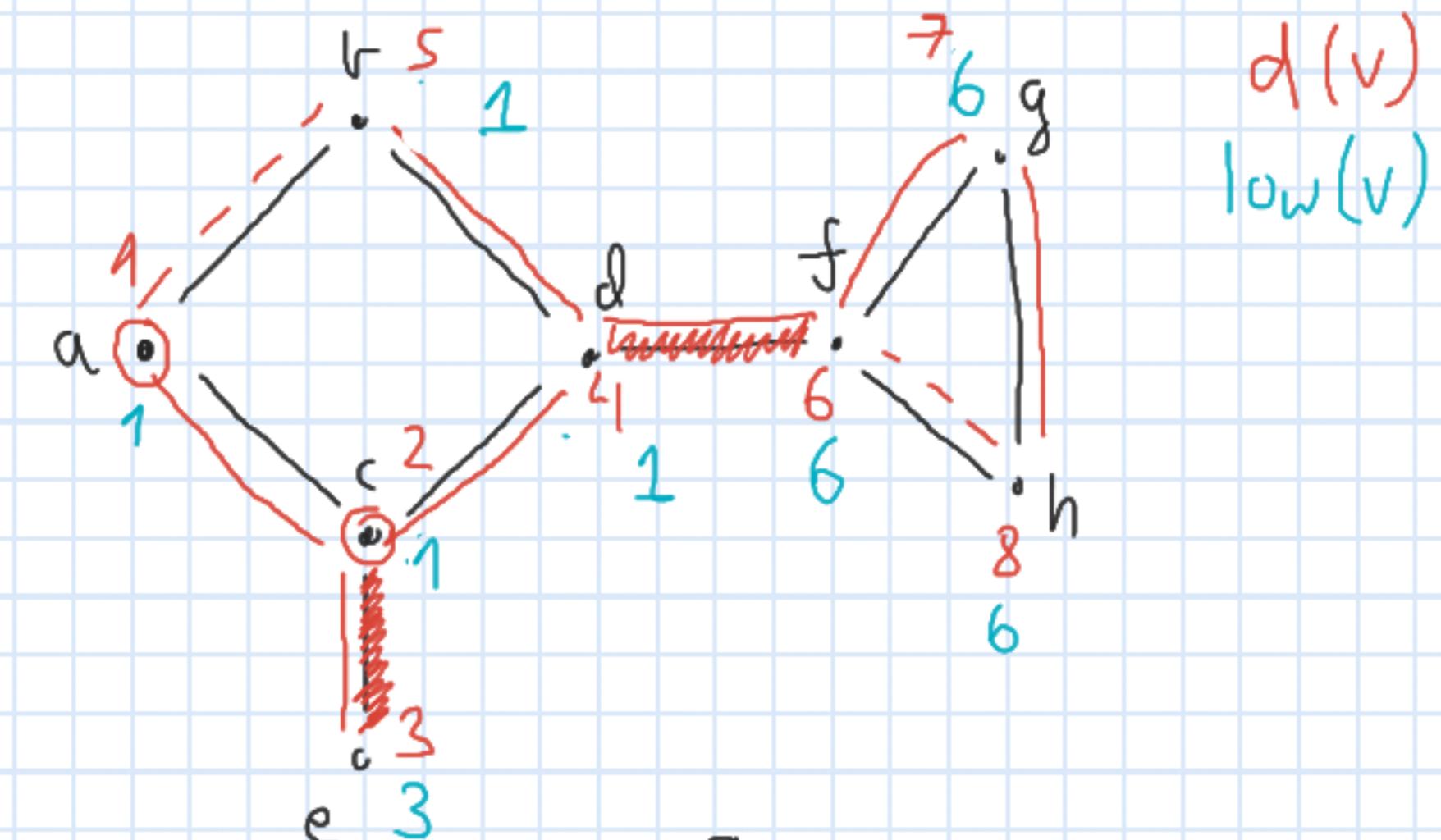
Algorytm

① Wykonaj DFS, dla każdego wierzchołka v zapamiętuj czas odniedzenia $d(v)$

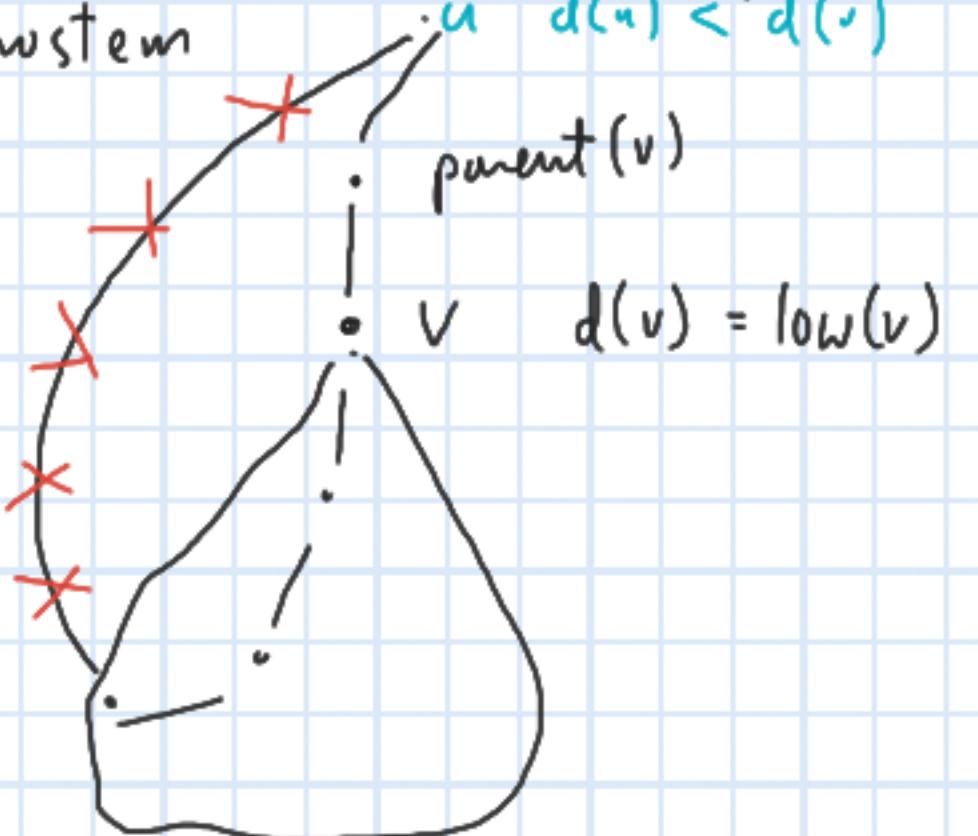
② Dla każdego wierzchołka v obliczamy

$$low(v) = \min \left(d(v), \min_{\substack{u - istnieje \\ krzyżdzi ustawna \\ z u do v}} d(u), \min_{\substack{w - dzieci \\ v - dziecie \\ DFS}} low(w) \right)$$

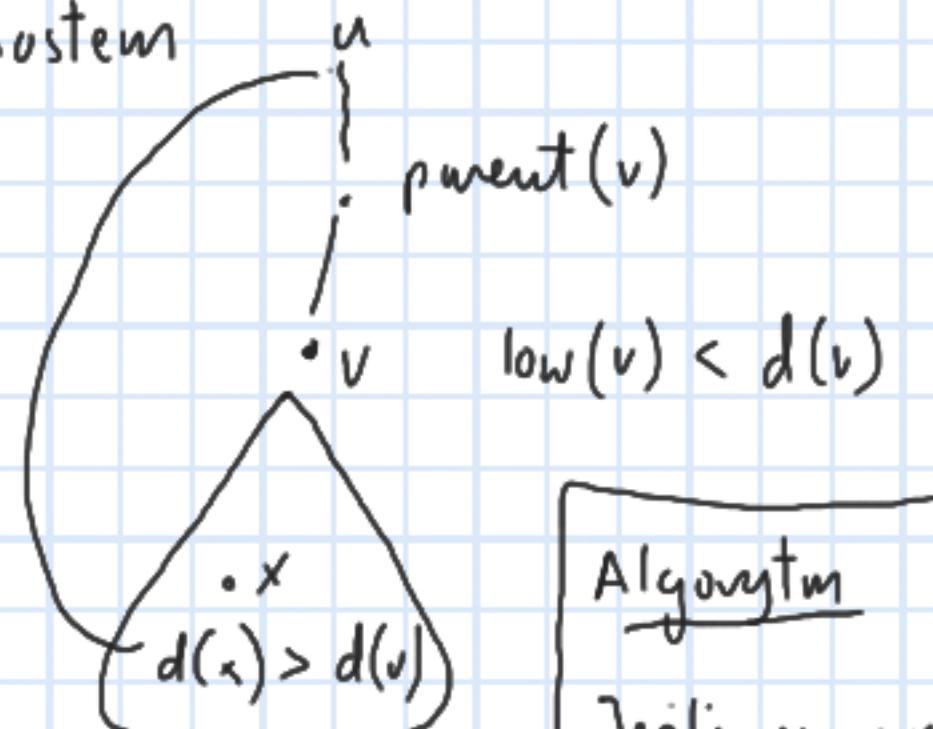
③ Mosty to krawędzie $\{v, \text{parent}(v)\}$ gdzie $d(v) = low(v)$



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



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



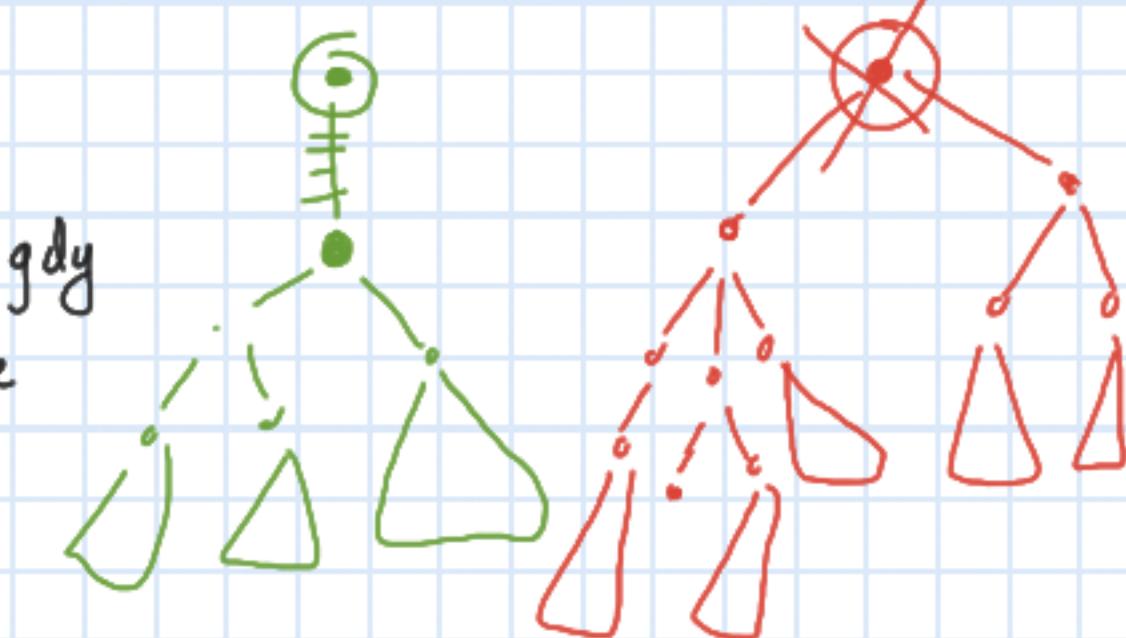
Algorytm

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

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

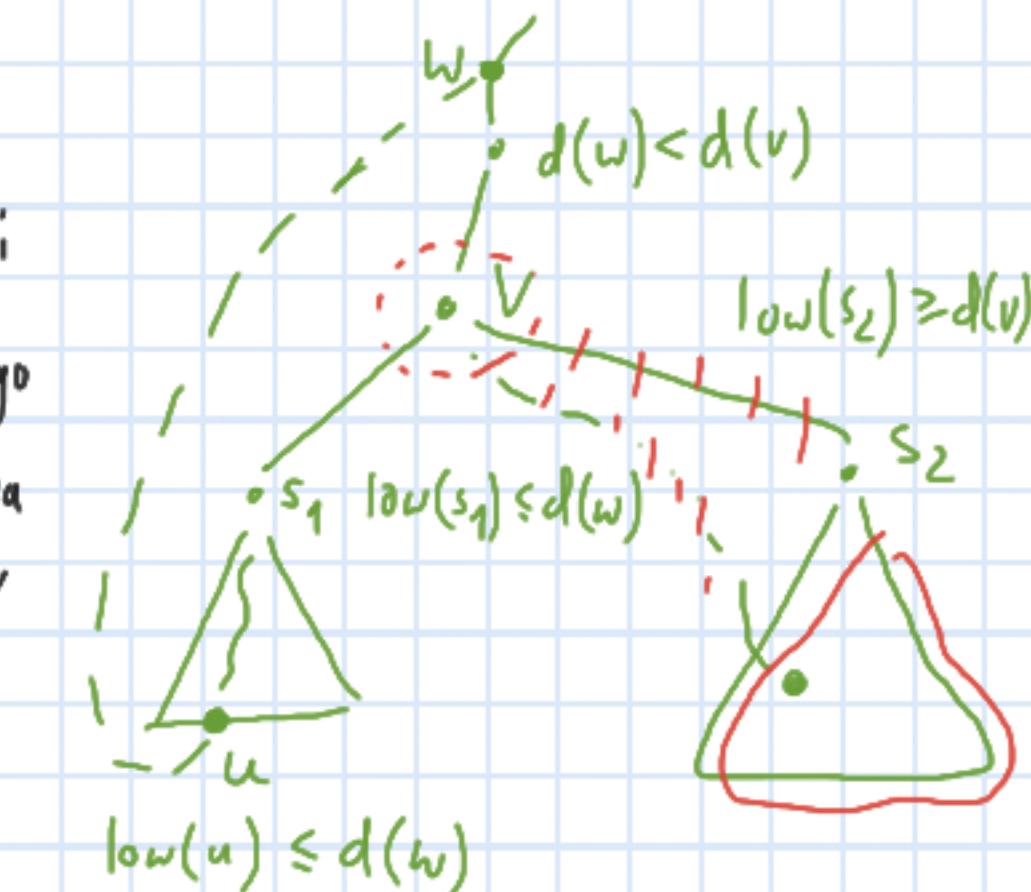
tw

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



tw Wierzchołek nie będący

koniem jest punktem artykulacji wtedy dla pary najmniejszej jednego syna nie istnieje krawędź łącząca $\{v, w\}$, taka że w to potomek v a w jest predkiem v



Algorytmy na grafach wazonych

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

Reprezentacja grafów wazonych

$$G = (V, E)$$

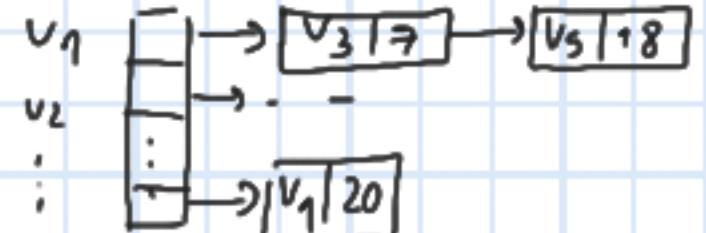
$$w: E \rightarrow \mathbb{N} \quad (\mathbb{Z}, \mathbb{Q}, \mathbb{Q}_+)$$

Reprezentacja macienna

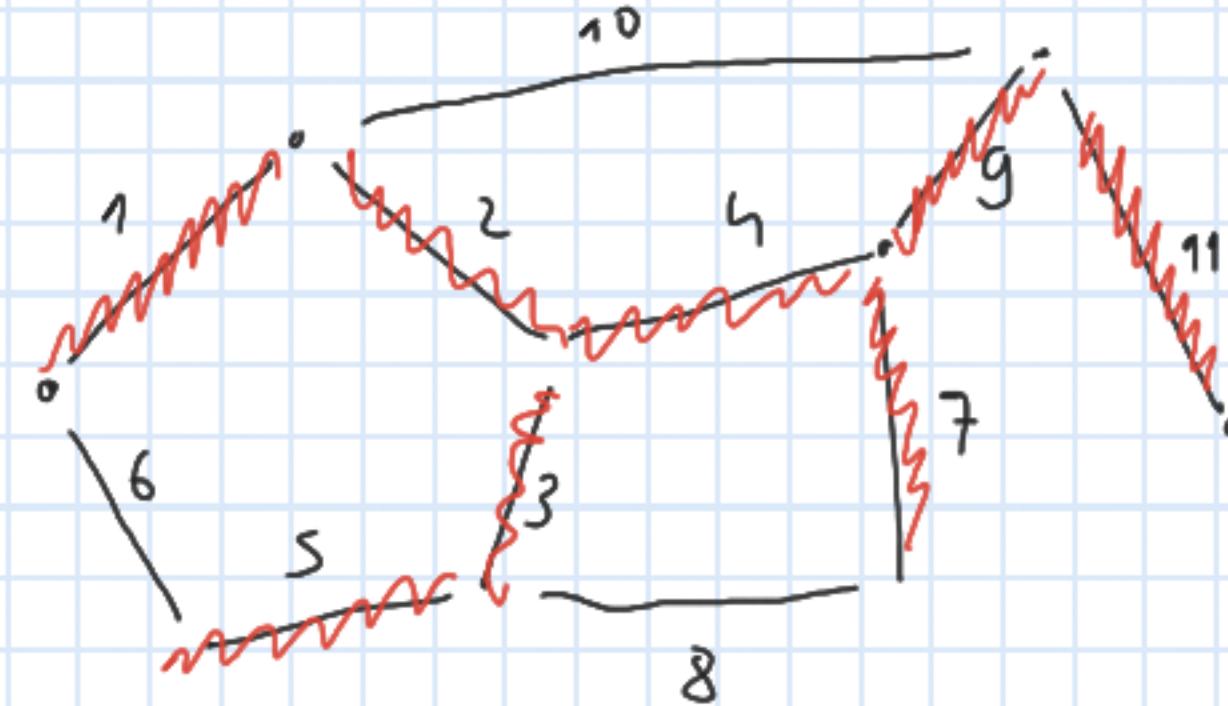
w - matematyczna wag

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

Reprezentacja listowa



Minimalne dno rozpinające



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

Obserwacja

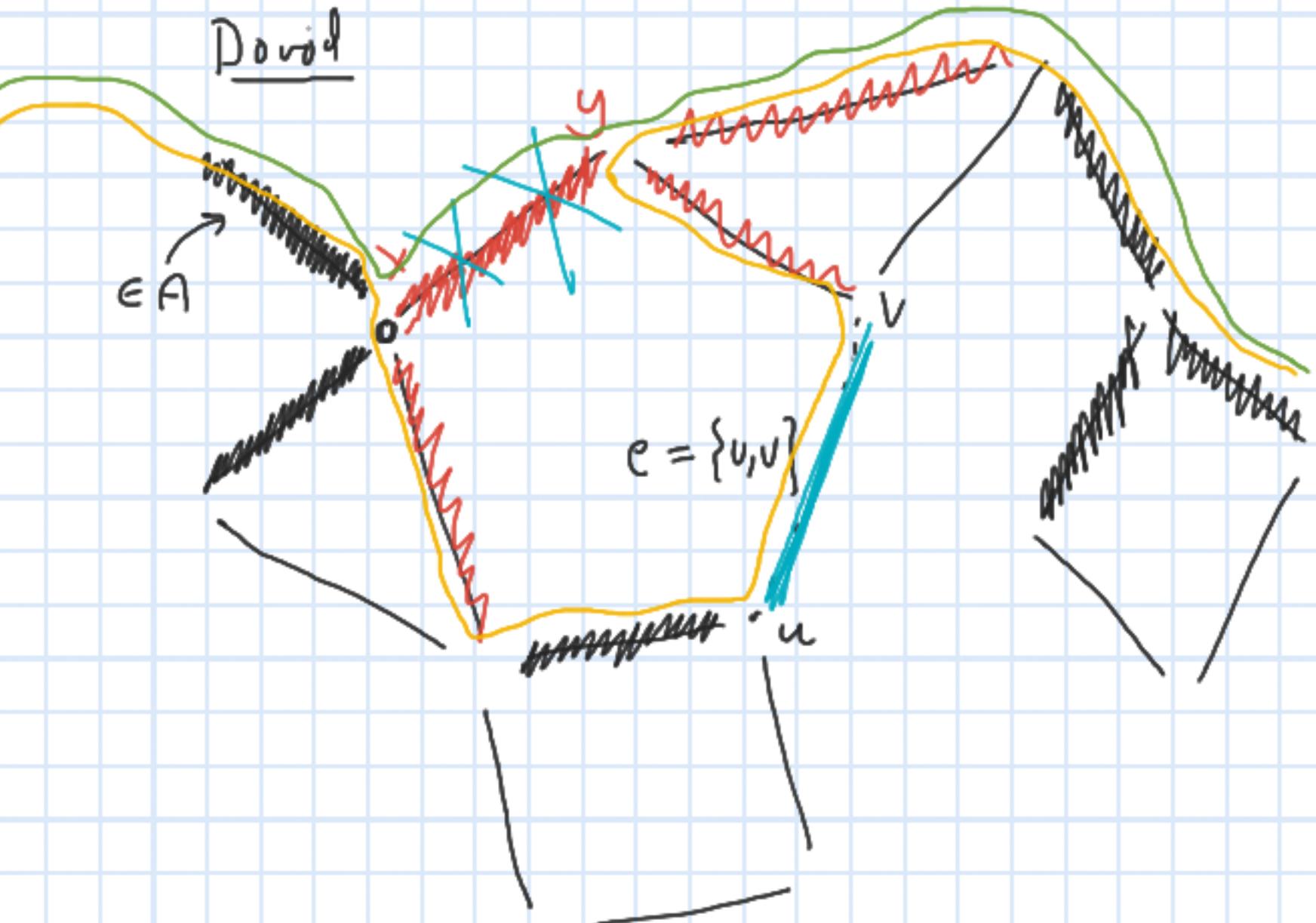
$$G = (V, E), \quad w: E \rightarrow \mathbb{N}$$

(II+)

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

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

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



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

Algorytm Kruskala znajdowania MST

① Posortuj krawędzie po wagach

② $A = \emptyset$

③ Przeglądaj krawędzie w kolejności
niesmalejącej wag

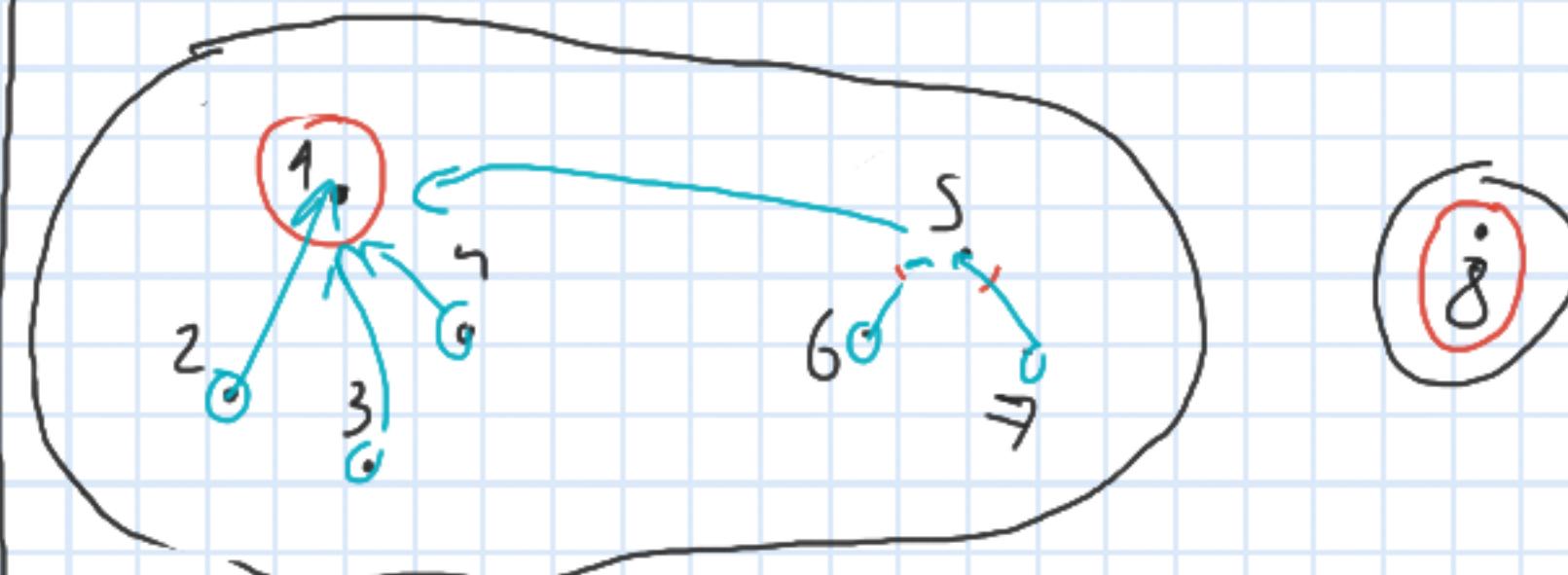
jeśli $A \cup \{e\}$ nie zauważa

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

⑤ Zwróć A



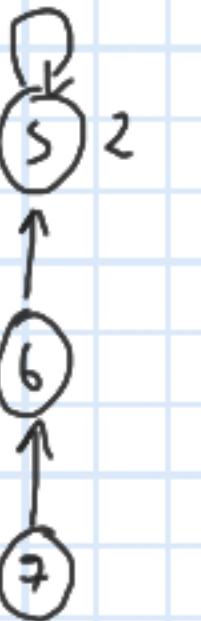
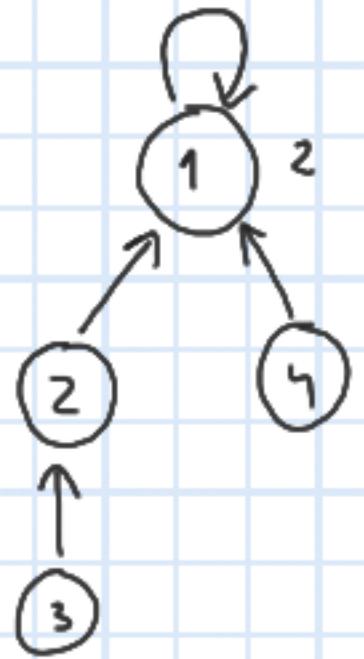
Lasy zbiorów rozłącznych / Find-Union



Trzy operacje

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

Понятие рефлексивной связности



class Node:

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

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

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

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

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

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

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

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



Obserwacja 1 Ranga to givne ograniczenie
na wysokość drzewa

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

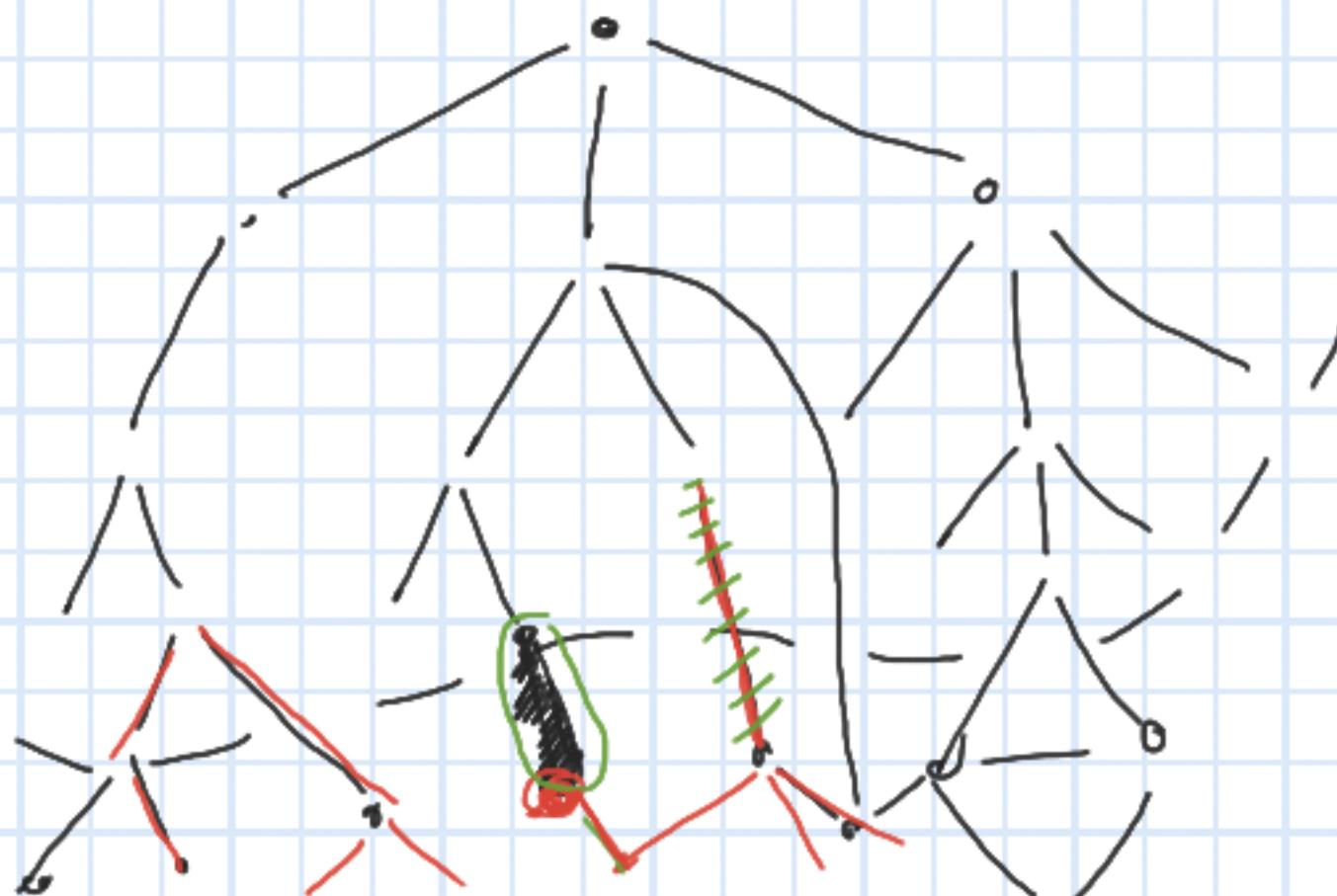
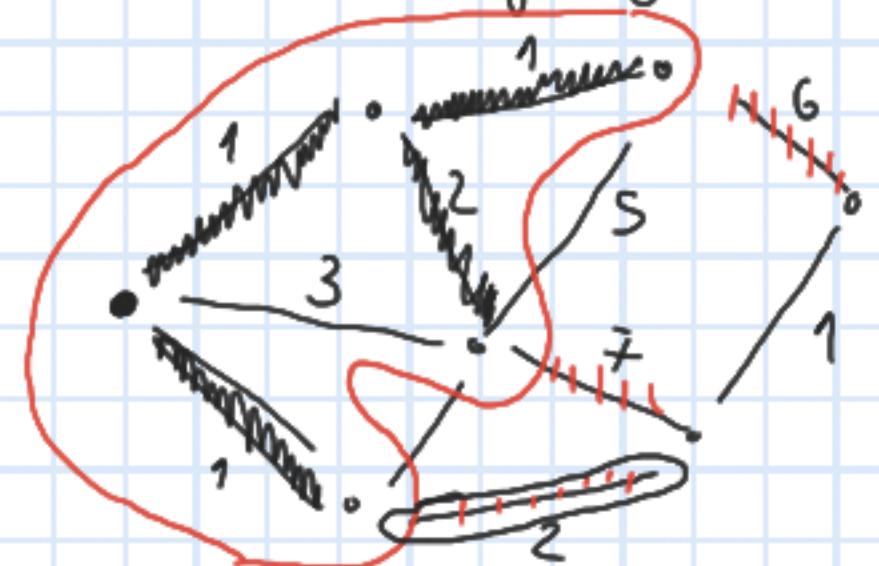
Obserwacja 3 Jeśli mamy T grupie n elementów
to czas m operacji find i union
ma złożoność $O(m \log n)$

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

ASD - Wykład 12

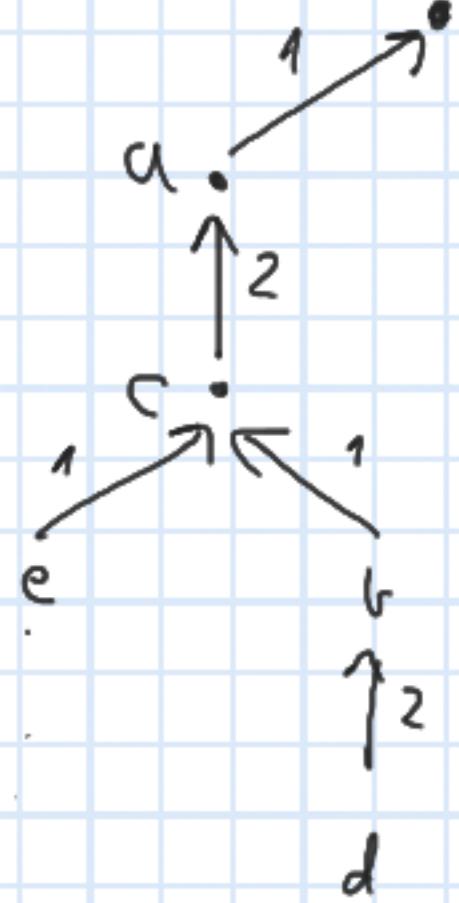
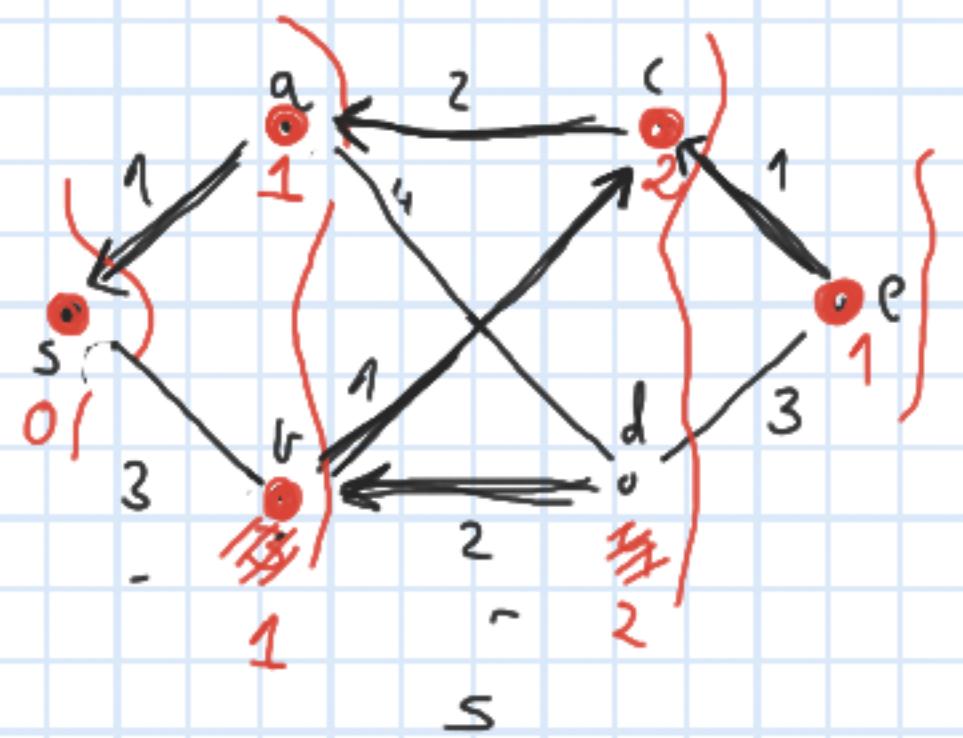
Algorytm Prima znajdowania minimalnego \longrightarrow Opis algorytmu

druga wersja



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

Prywatny wykłaniam algorytmu Pima



Problem znajdowania najkrótszych ścieżek

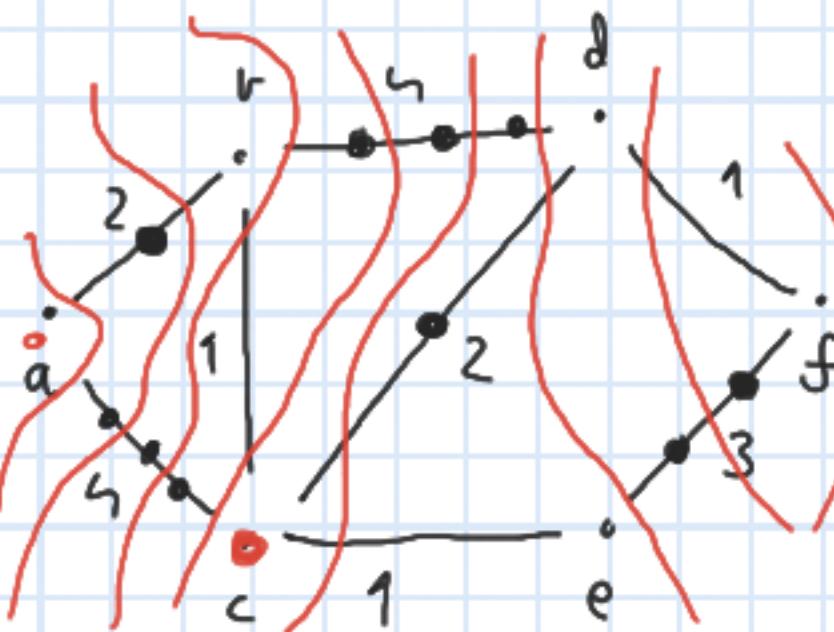
Warianty

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

} na elementarnym poziomie trudne do wykonywania

} standardowe wersje problemu

Podając elementarne



BFS na sztucznie
rozmnожonych koordziach

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

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

Notacja

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

$w(u,v)$ - waga krawędzi $\{u,v\}$

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

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

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

① Umieść wszystkie wierzchołki w kolejce priorytetowej (min)
z oszacowaniem odległości ∞

② zmień odległość s na 0

③ podczas wierzchołków w kolejce:

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

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

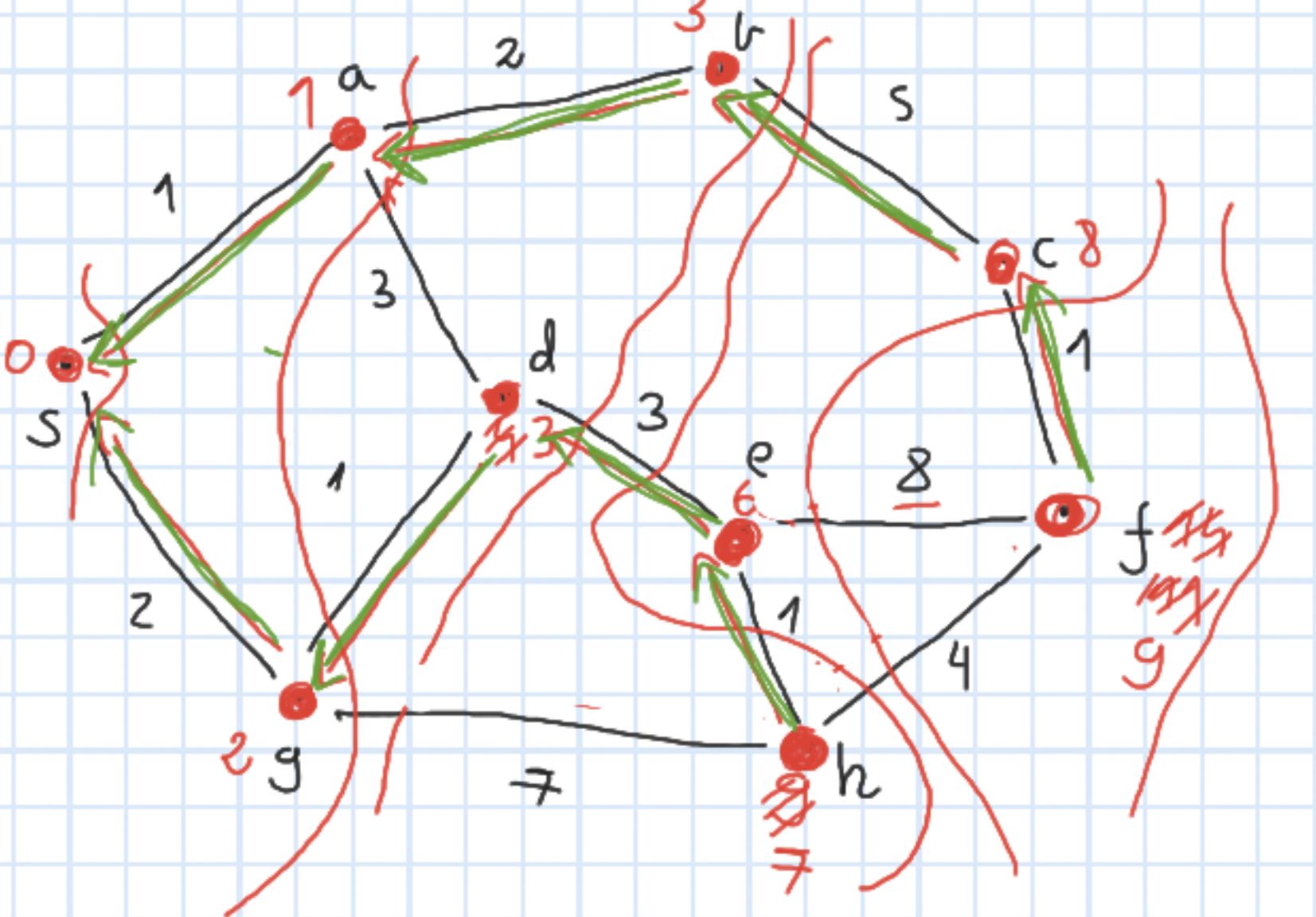
def relax(u, v):

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

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

$v.parent = u$

Punktad nykonomia algorytm Dijkstra

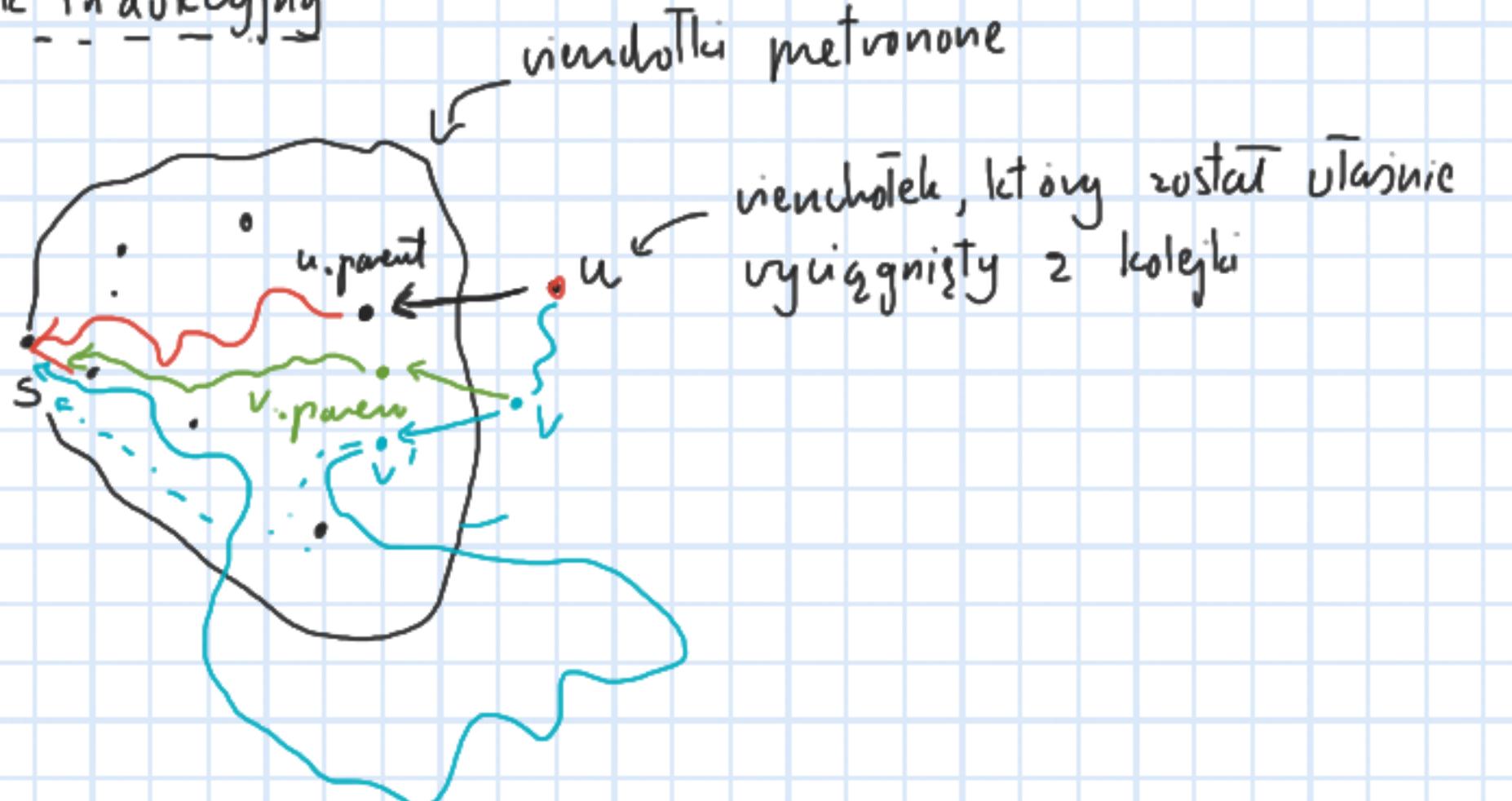


tw Gdy algorytm Dijkstry wyciąga
wierzchołek u z kolejki, to jego pole u.d
zawiera długość najkrótszej ścieżki z s do u

Dowód (przez indukcję)

dla wierzchołka startowego oznaczając zadanie

krok indukcyjny



Algorytm Bellmana - Forda

- najkrótsze ścieżki jeśli wagi krawędzi ujemne

① Inicjalizacja

for $v \in V$:

$v.d = \infty$

$v.parent = \text{None}$

$s.d = 0$

② Relaksacja

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

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

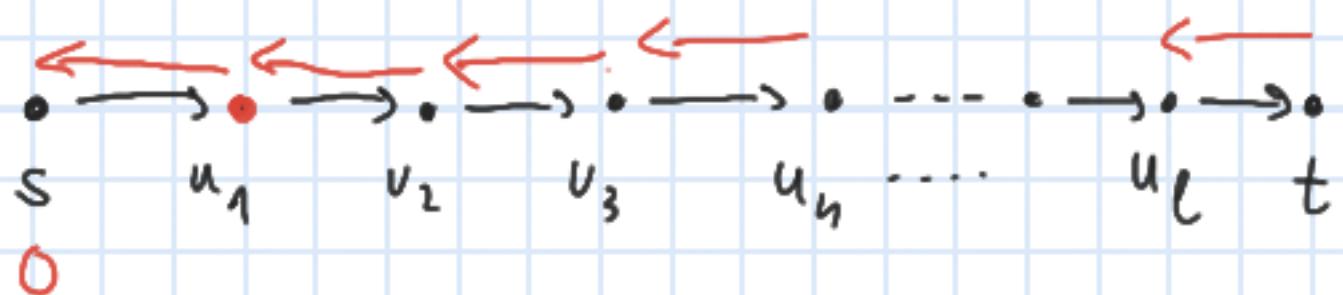
Relax(u, v)

③ Uczyfikacja

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

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

Czy to działa?



Złożoność

$O(VE)$

ASD - Wykład 13

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

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

① Inicjalizacja

for $v \in V$

$$v.d = \infty$$

$v.parent = \text{None}$

$$s.d = 0$$

② Relaksacja

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

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

Relax (u, v)

③ Weryfikacja

Czy dla każdej $(u, v) \in E$:

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

Jesli tak - OK

Jesli nie - ujemny cykl

Zauważmy, że G posiada pierścień

cykl o ujemnej wartości, osiągającą ze średnia s



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

szerzenie

Jesli w taki sytuacji weryfikacja by powiedziała, to:

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

\downarrow

(v_i, v_{i+1})

Sumując te nierówności:

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

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

Najkrótsze ścieżki między każdą parą wierzchołków

- $|V|$ wersja algorytmu Dijkstry

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

- $|V|$ wersja algorytmu Bellmana - Forda

$$O(V^2 E)$$

Konwencja

Stosujemy reprezentację macienną

Stosujemy macierze najkrótszych odległości

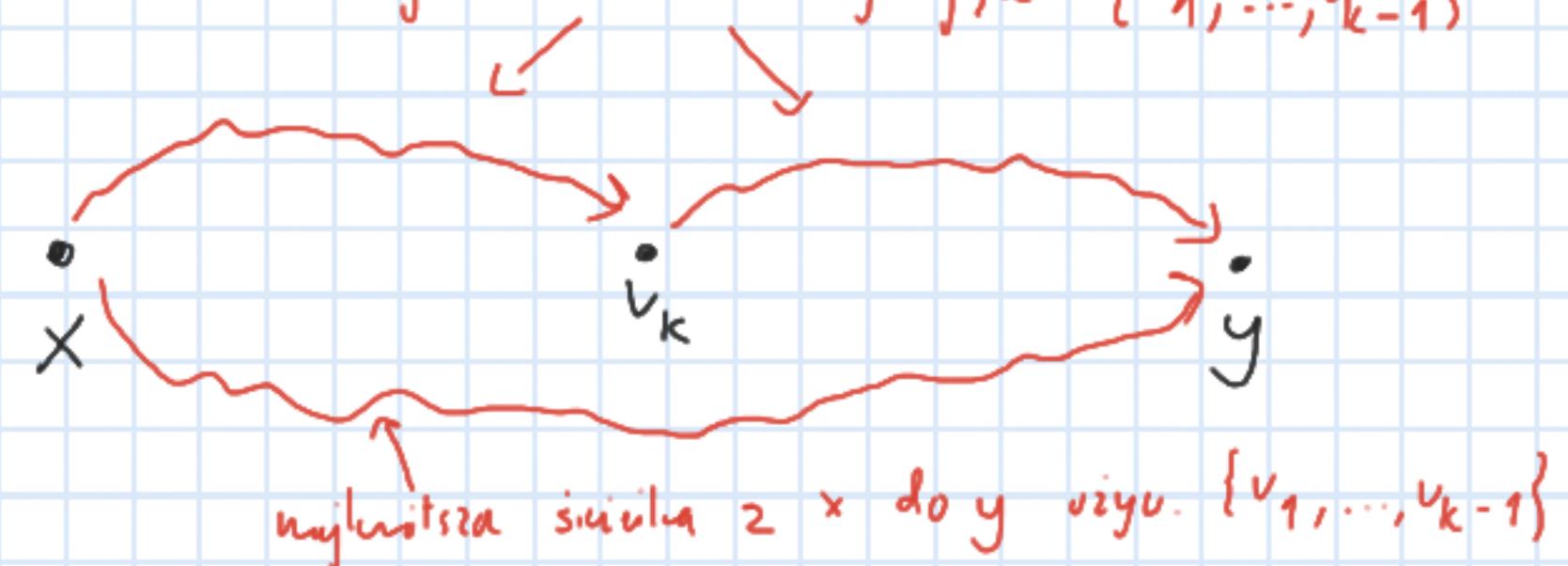
$D[u][v]$ - dł. najkrótszej ścieżki z u do v

Algorytm Floyda - Warshalla

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

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

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



Implementacja algorytmu Floyda - Warshalla

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

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

2 fazowe

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

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

 for $u \in V$:

 for $v \in V$:

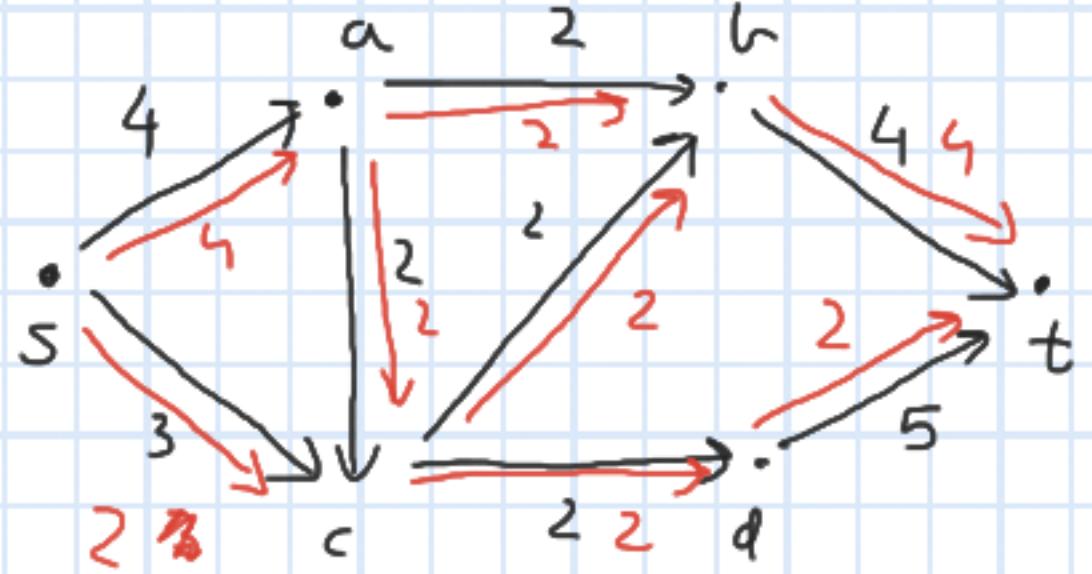
$$D^{(k)}[u][v] = \min \left(D^{(k-1)}[u][v], D^{(k-1)}[u][v_k] + D^{(k-1)}[v_k][v] \right)$$

poniższy u implementacj.

return $D^{(n)}$

ASD - Wykład 13b

Problem maksymalnego przepływu



Ujęcie: $G = (V, E)$ - graf skierowany
(krzgdzi tylko w jedno
strone)

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

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

s - źródło, nie ma krzgów wychodzących

t - ujście, nie ma krzgów wychodzących

Zadanie

Znaleźć "przepływ" f o maksymalnej wartości

to funkcja

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

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

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

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

Sieci reszidualne

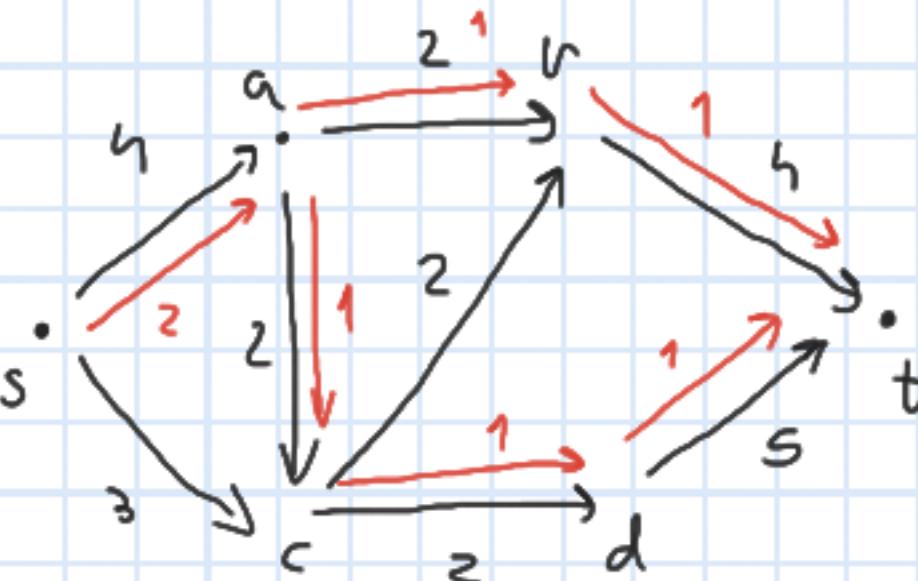
$$G = (V, E)$$

$$s, t \in V$$

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

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

sieci przepływu



Definiujemy sieci residualne G_f, c_f

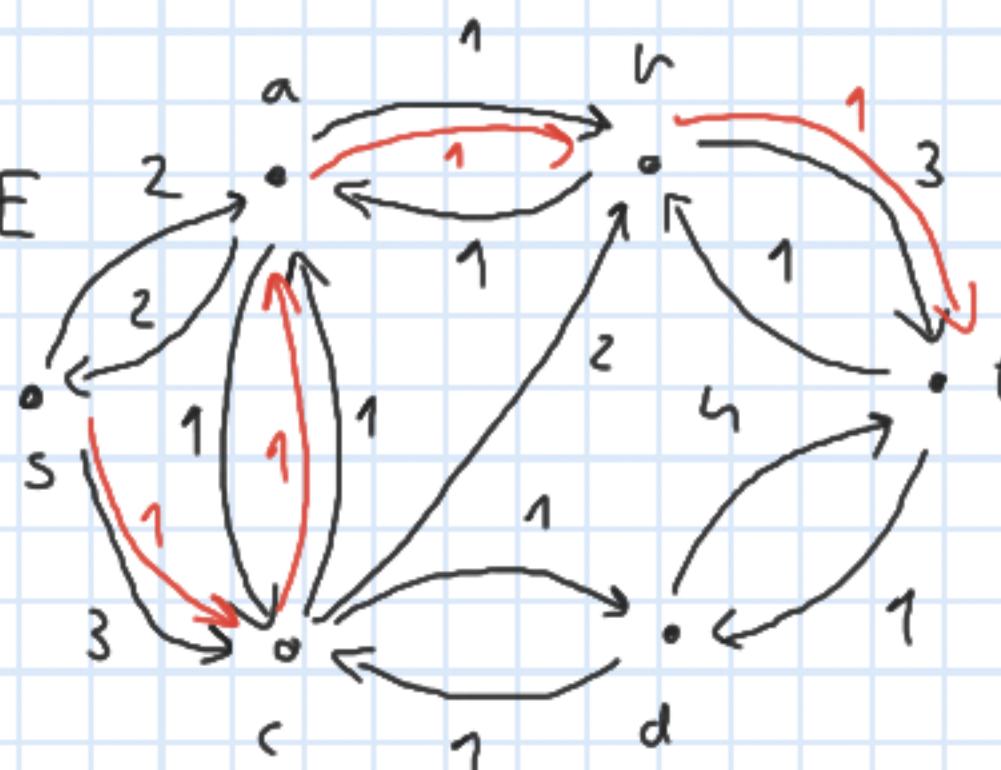
przez funkcję c_f :

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

Metoda Forda - Fulkersona

- ① Jeśli istnieje siecika przenosząca dla G_f, c_f , to powiększyć zgodnie z nią przepływy.

② wróć do kroku ①



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

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

Pnekovj u sieci

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

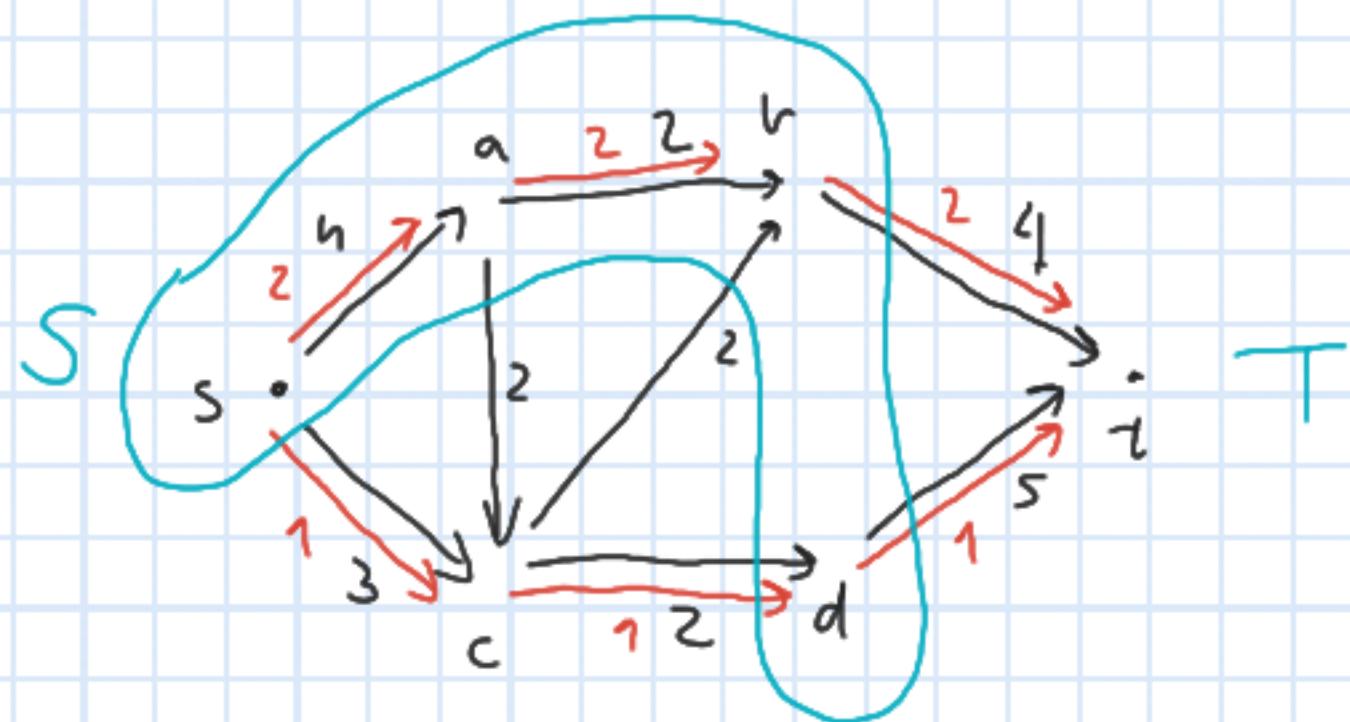
$$c: V \times V \rightarrow \mathbb{N}$$

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

Pnekovj sieci to podzbiot V na

$$S, V - S = T$$

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



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

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

Pnepustovici pnekova S, T to:

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

Pneplyw netto:

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

Lemat

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

Dowód

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

= 0
 (utrzymać
 zachowania
 pusty u)

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

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

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

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

$$= f(S, T)$$

Lemat

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

Dowód

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

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

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

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

tu (max-flow / min-cut theorem)

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

być sieć przepływu oznaczającą f niech będzie

przepływem w tej sieci. Następujące

warunki są równoważne:

- ① f jest maksymalnym przepływem w G
- ② G_f, c_f nie maścieli powiększających
- ③ Dla pewnego podkroju S, \bar{T} zachodzi
 $|f| = c(S, T)$

Dowód

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

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