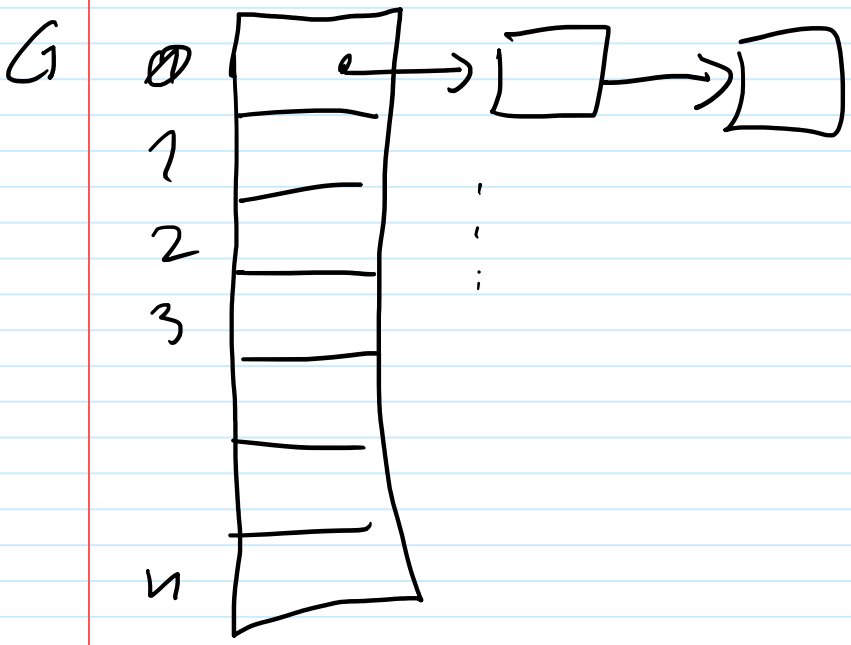
# ASD 2022.05.26

## Grafy, Sortowanie topologiczne




G[i] - lista krawędzi wychodzących z wierzchołka i.

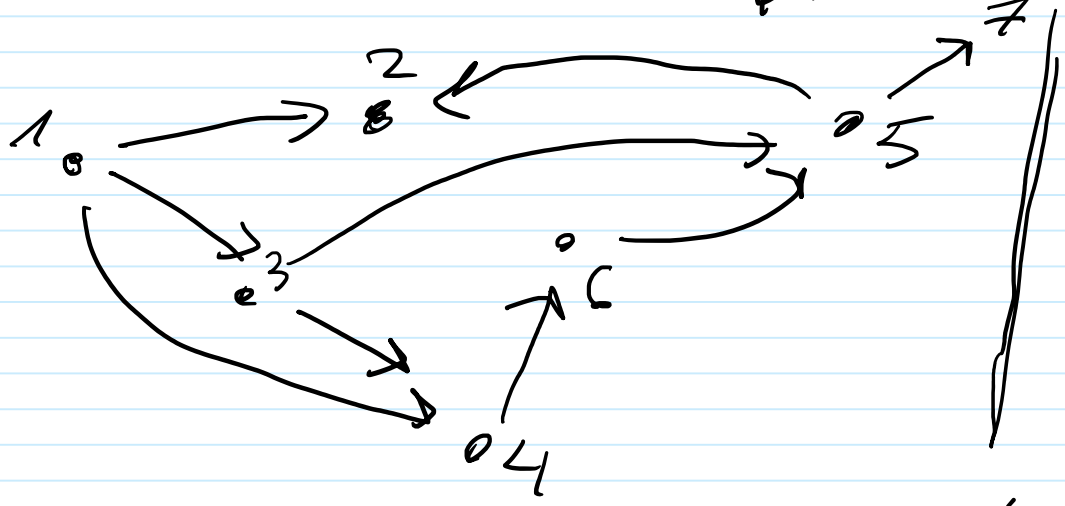
Def

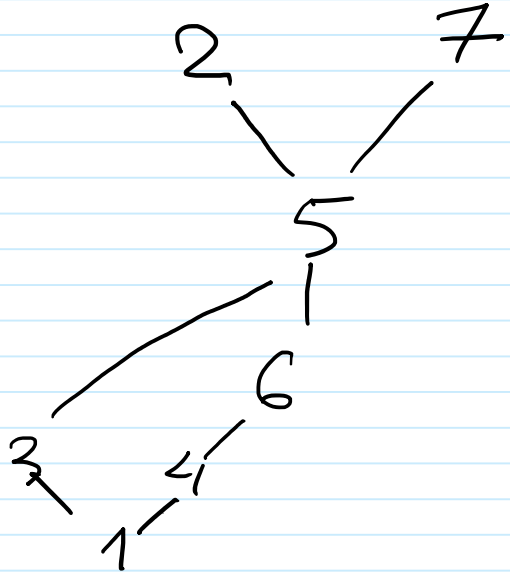
Graf jest DAG (directed acyclic graph) skierowany, bez cykli

Kiedy jest skierowany i nie zawiera cykli.

---

Niech G będzie DAG.




$a \leq b$ tzn $a = b$ lub istnieje ścieżka w G z a do b

## Problem

Dane: $G$ - skierowany DAG

Wy: liniowy porządek zgodny z porządkiem częściowym zdefiniowanym standardowo na $G$

---

## Fakt

Każdy skierowany częściowy porządek rozszerza się do porządku liniowego.

**Def**

Niech $P=(U, \leq_P)$ – porządek częściowy.

$L=(U, \leq_L)$ – porządek liniowy

$P$ rozszerza się do $L$, jeśli

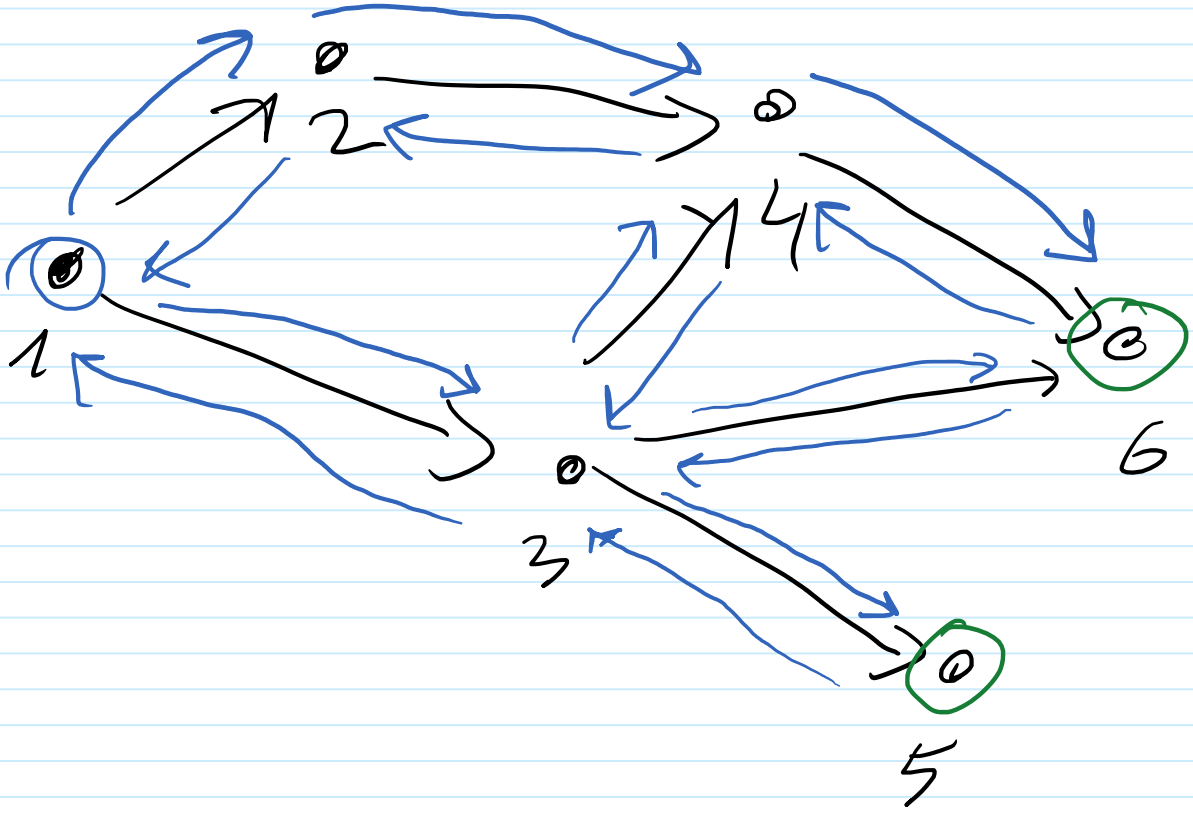
$$\forall a, b \in U \; [a \leq b \rightarrow a \leq_L b]$$

$\leq_P \subseteq \leq_L$.

# Algorytm

- wykonuj stamy przeszukiwanie grafu dfs
- z jednej strony dfs odwiedza wszystkie wierzchołki osiągalne z danego wierzchołka
- przechodzi przez wszystkie krawędzie
- wykrywa cykle

# Strategia działania algorytmu




1, 3, 5, 2, 4, 6

```
% Sortujemy topologicznie
% graf G o wierzchołach 0,..., size-1,

% Algorytm oparty jest o przeszukiwanie dfs

List topSort;

TopologicalSort(Graph G, int size){
   topSort=empty;
   String status[0,...,size]={"waiting", ...., "waiting"};
   for(i=0;i<size;i++)
       if(status[i]="waiting")
          visit(i)
}

visit(int i){
  if(status[i]="on_list") return;
  if(status[i]="in_processing") return Error(Cycle_Detected);
  status[i]="in_processing";
  Node *ptr=G[i].edges;
  while(ptr!=NULL){
    visit(ptr->end);
    ptr=ptr->next;
  }
  topSort=(i)*topSort; //add i to the head of topSort
  status[i]="on_list";
}
```

Na tej liście będą wierzchołki w konstruowanym porządku liniowym.

status:

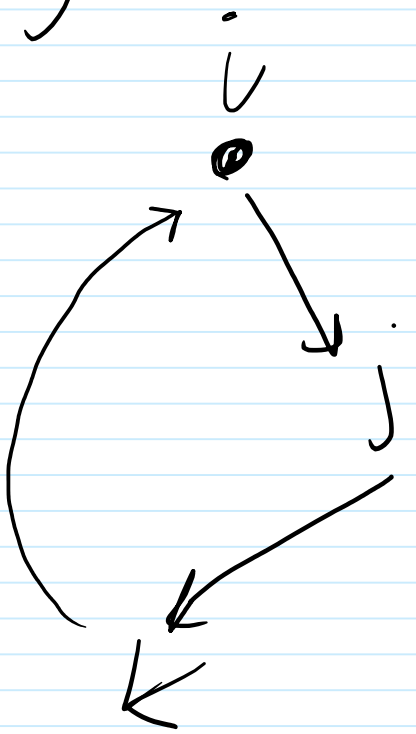
waiting - nic się nie działo z wierzchołkiem.

in-processing - przetwarzamy wierzchołek, funkcja visit() działa

on_list - wierzchołek jest już na liście topSort.

w tym momencie wszystkie wierzchołki osiągalne z i są na liście topSort.

in_processing



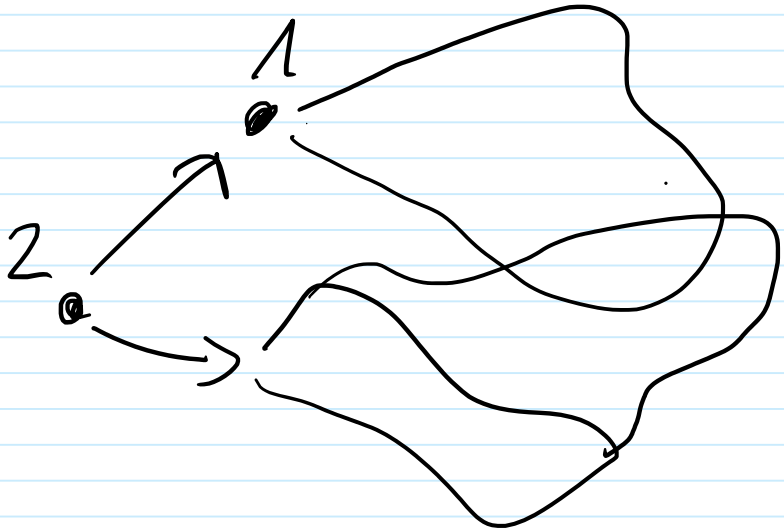

visit(i)

visit(j)

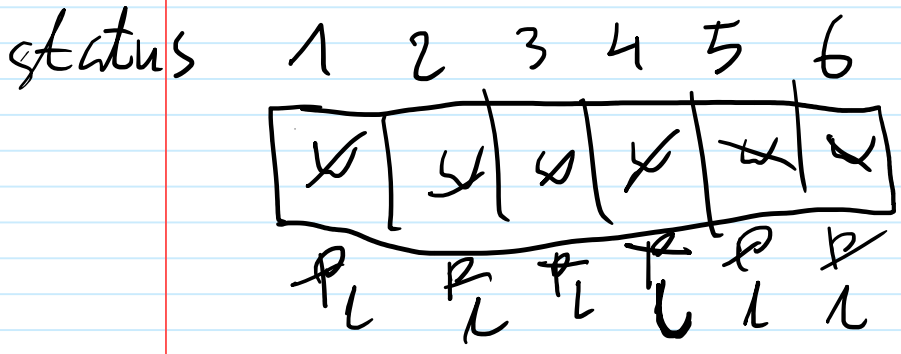
visit(k)

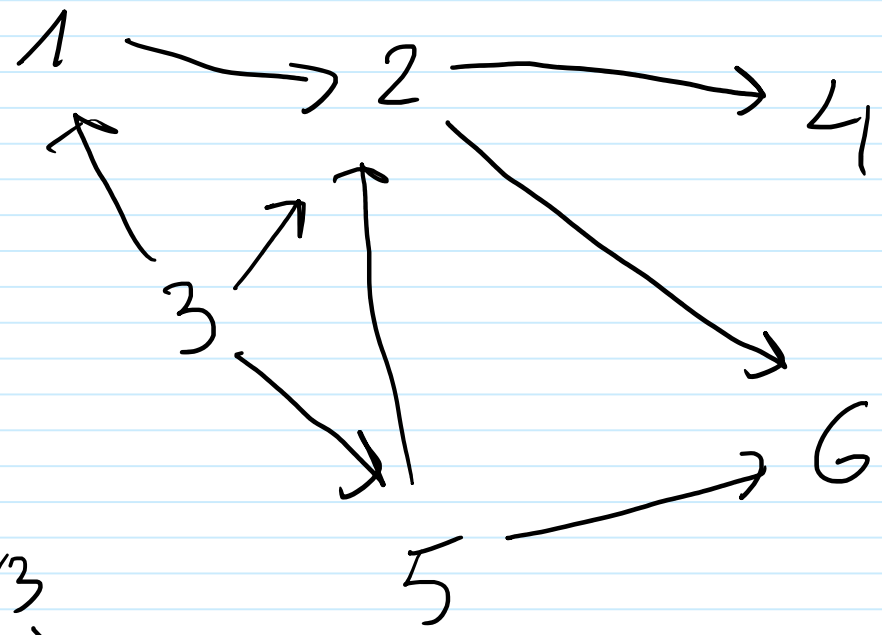
visit(i)

Wykryliśmy cykl, graf nie jest DAG-iem.


1
2

W - waiting
P - in processing
L - on list




visit(1)

visit(2)

visit(4) visit(6)

visit(3

visit(2) visit(5)

visit(2) visit(6)

top Sort: 3, 5, 1, 2, 6, 4

W tamtej algorytmie wykrywającym z wtamtej dfs.

- Jeśli w grafie jest cykl, to zwracamy błąd (bo dfs wykrył cykl).
- Kończymy funkcję visit(i) tylko gdy wszystkie wierzchołki osiągalne z i są już na liście wynikowej topSort.

W takim razie możemy dołożyć i na początek tej listy.

# Minimalne drzewo rozpinające w grafie
(algorytm Prima)

**Def**

Graf z wagami krawędzi to graf $G = (V, E, w)$, gdzie $(V, E)$ to graf.

$$w : E \to \mathbb{N}^{>0}$$

$w(e)$ - waga krawędzi $e$.

Def

Niech $G = (V, E)$ graf nieskierowany, spójny, skończony.

$T$ jest drzewem rozpinającym $G$ jeśli:

$T = (V, F)$,

$T$ jest spójny

$T$ jest drzewem (to znaczy, że $T$ ma minimalną ilość krawędzi równą $|V| - 1$).

~~Twierdzenie~~

**Def**

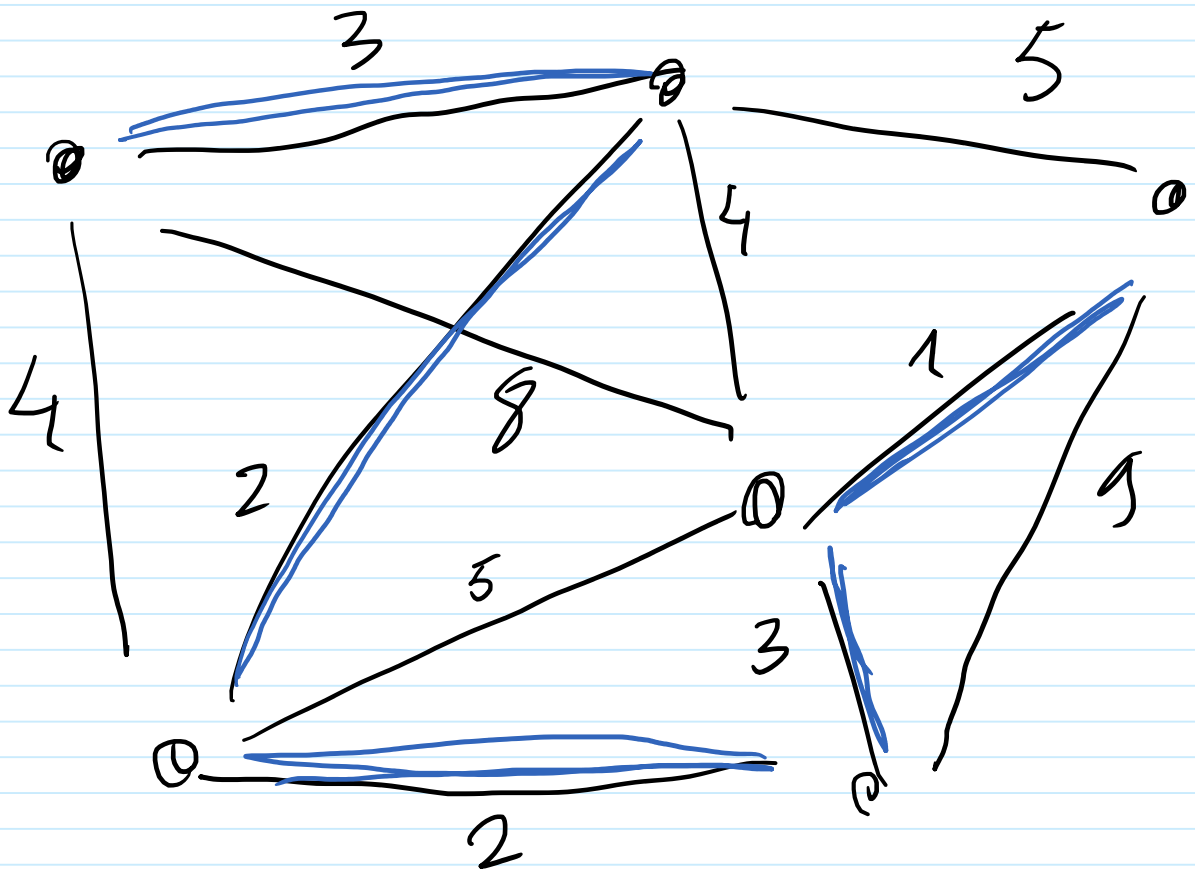
Niech $G = (V, E, w)$ – graf nieskierowany, spójny z wagami.

$T = (V, F)$ jest minimalnym drzewem rozpinającym $G$ jeśli:

- $T$ jest drzewem rozpinającym $G$
- suma wag krawędzi w $T$ jest minimalna wśród wszystkich drzew rozpinających $G$.

$\sum_{e \in F} w(e)$ – minimalne.

$T = (V, F)$




$$\sum_{e \in F} w(e) = 11.$$

**Fakt**

Niech $G=(V,E,w)$ - spójny, nieskierowany graf z wagami,

Niech $T=(W,F)$ - drzewo, t.j. $W \subseteq V$

$F \subseteq E$

czyli $T$ jest podgrafem $G$

$T$ jest poddrzewem pewnego minimalnego drzewa rozpinającego $G$.

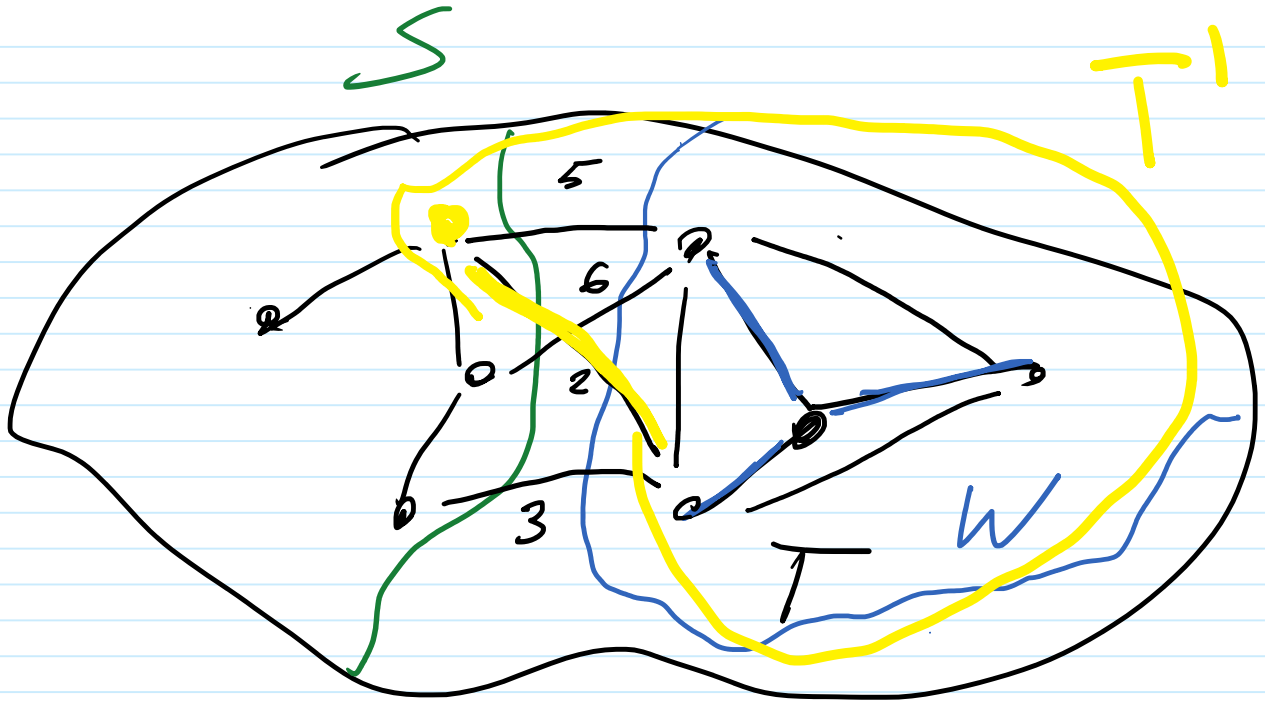
Niech

$$S = V \setminus W \neq \emptyset$$

$$H = (W \times S) \cap E = \{(a,b) : a \in W, b \in S, (a,b) \in E\}$$

$e$ - krawędź w $H$ o minimalnej wadze $e=(a,b), b \in S$

Tezа: $T' = (W \cup \{b\}, F \cup \{e\})$ - poddrzewo pewnego minimalnego drzewa rozpinającego $G$




Algorytm Prima działa tak, że zaczyna od drzewa pustego, i dokłada kolejne krawędzie i wierzchołki do rozwiązania zgodnie z opisaną w Fakcie strategią.