

Graf i digraf

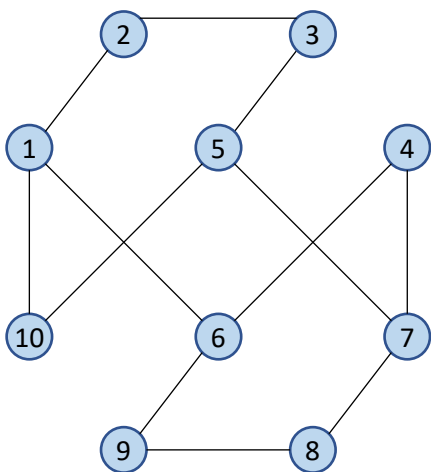
Graf $G=(V,E)$ to struktura, która składa się ze zbioru wierzchołków V oraz zbioru krawędzi E :

$$V = \{v_1, v_2, v_3, \dots, v_n\}$$

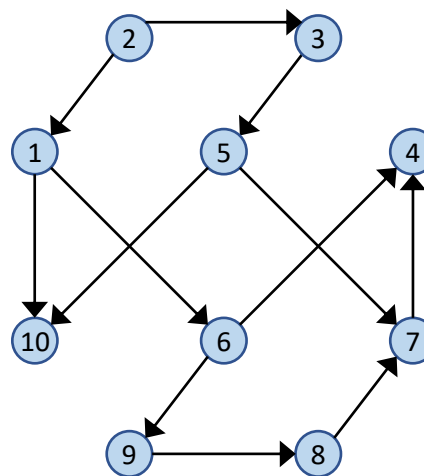
$$E = \{e_1, e_2, e_3, \dots, e_m\}, \text{ gdzie } \forall_{i=1..m} e_i=(u,v): u,v \in V$$

Liczbę n wierzchołków w grafie nazywamy **rzędem grafu** G .

Liczbę m krawędzi w grafie nazywamy **rozmiarem grafu** G .



Graf nieskierowany
(krawędzie nieskierowane)



Digraf = graf skierowany
(łuki = krawędzie skierowane)

Cechy grafów

Jeśli krawędź wchodzi lub wychodzi z wierzchołka to mówimy, że jest z nim **incydentna**.

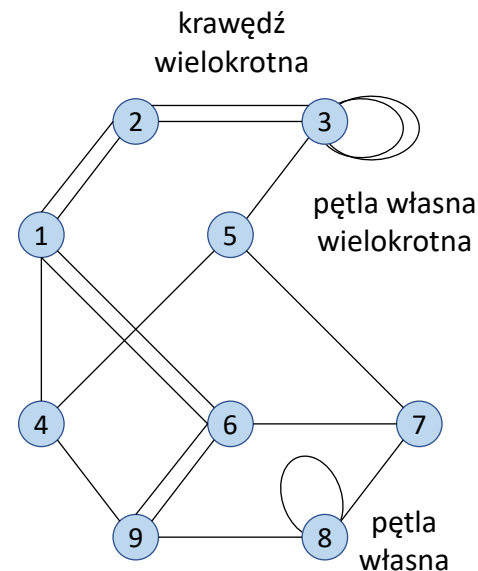
Liczba krawędzi incydentnych z wierzchołkiem v_i w grafie nieskierowanym to **stopień wierzchołka** v_i . W grafie skierowanym każdy wierzchołek ma:

- **stopień wejściowy** (liczba krawędzi wchodzących do tego wierzchołka, ang. in-degree) oraz
- **stopień wyjściowy** (liczba krawędzi wychodzących, ang. out-degree).

Jeśli krawędź wychodzi i wchodzi do tego samego wierzchołka to nazywamy ją **pętlą własną**.

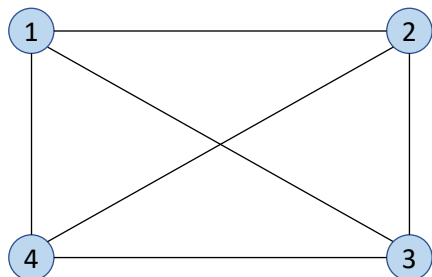
Multigraf to taki graf, w którym występują pętle i krawędzie wielokrotne. Jeśli dwa wierzchołki u, v są połączone kilkoma krawędziami to mówimy, że krawędzie te tworzą krawędź wielokrotną.

Graf prosty to graf bez pętli własnych i krawędzi wielokrotnych.



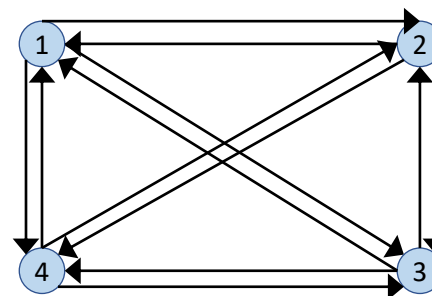
Multigraf

Graf pełny, nasycenie grafu



Graf pełny (prosty) to graf (nieskierowany) $G=(V,E)$, w którym każda para wierzchołków $u,v \in V$ jest połączona krawędzią $(u,v) \in E$. Taki graf ma $m = \frac{n(n-1)}{2}$ krawędzi, gdzie n to liczba wierzchołków grafu G .

Graf rzadki to graf, który ma niewiele krawędzi: $m \ll n^2$ (m – liczba krawędzi, n – liczba wierzchołków grafu).



Diraf pełny (prosty) to graf skierowany $G=(V,E)$, w którym każda para wierzchołków, $u,v \in V$, jest połączona dwoma łukami $(u,v), (v,u) \in E$. Taki graf ma $m = n(n-1)$ łuków, gdzie n to liczba wierzchołków grafu G .

Graf $G=(V,E)$ ma nasycenie X , jeśli zbiór jego krawędzi ma $X\%$ liczby krawędzi grafu pełnego o tej samej liczbie wierzchołków.

Ścieżka, droga i cykl

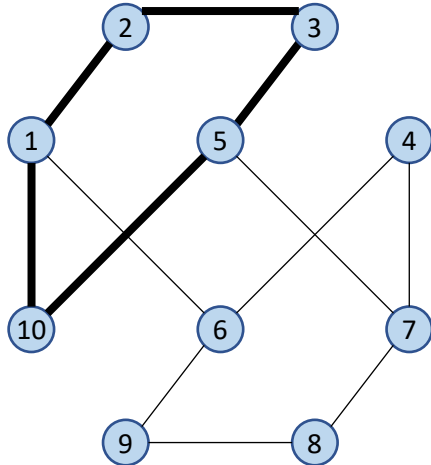
Ścieżka w grafie $G=(V,E)$ to skończony ciąg wierzchołków tego grafu. Ścieżkę w grafie G możemy zapisać tak: $v_i \rightarrow v_{i+1} \rightarrow v_{i+2} \rightarrow \dots \rightarrow v_{i+k}$, gdzie $\forall_{i \in \{1, \dots, |V|\}} v_i \in V$. **Długość ścieżki** to liczba jej krawędzi.

Ścieżka prosta to ścieżka, w której wierzchołki nie powtarzają się

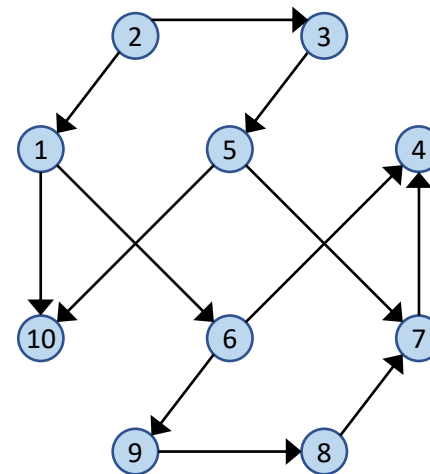
Droga w grafie $G=(V,E)$ to skończony ciąg krawędzi/łuków tego grafu.

Cykl w grafie $G=(V,E)$ to ścieżka zamknięta, tj. ścieżka, w której pierwszy wierzchołek jest jednocześnie ostatnim.

Graf zawierający cykl to **graf cykliczny**. Graf, w którym nie ma cyklu to **graf acykliczny**.



Graf nieskierowany cykliczny



Digraf acykliczny

Reprezentacje maszynowe grafu

Graf $G=(V,E)$, gdzie V to lista wierzchołków ($|V|=n$), a E to lista krawędzi/łuków ($|E|=m$) można zapisać maszynowo stosując jedną z reprezentacji:

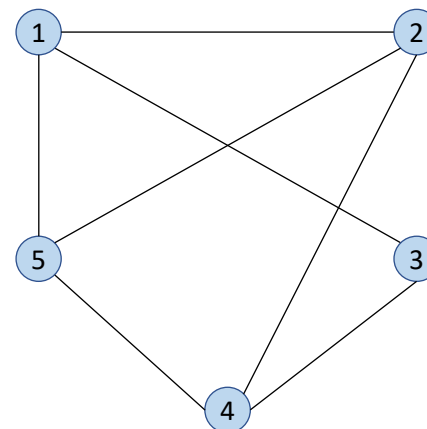
Struktura danych	Rozmiar*	Graf nieskierowany	Graf skierowany
Macierz sąsiedztwa	$n \times n$	√	√
Macierz incydencji	$n \times m$	√	√
Lista krawędzi	m	√	√
Lista incydencji/sąsiedztwa	$n + m$	√	
Lista następników	$n + m$		√
Lista poprzedników	$n + m$		√
Macierz grafu	$n \times (n + 4)$		√

* Inaczej można powiedzieć, że jest to złożoność pamięciowa i zapisać w notacji O , np. dla macierzy sąsiedztwa złożoność pamięciowa będzie wynosiła $O(n^2)$.

Reprezentacje maszynowe grafu: macierz sąsiedztwa

Macierz sąsiedztwa M dla grafu nieskierowanego $G=(V,E)$

- ma rozmiar $|V|^2$
- numer wiersza macierzy = numer wierzchołka grafu
- numer kolumny macierzy = numer wierzchołka grafu
- $M[i,j]=1$ iff w grafie G istnieje krawędź (v_i, v_j)
- $M[i,j]=0$ iff w grafie G nie istnieje krawędź (v_i, v_j)



Macierz sąsiedztwa dla grafu nieskierowanego jest symetryczna względem głównej przekątnej. Całą informacją o strukturze grafu jest zawarta w połowie macierzy pod (lub nad) przekątną.

Zaleta: bardzo szybka możliwość sprawdzenia czy istnieje krawędź między dwoma wierzchołkami.

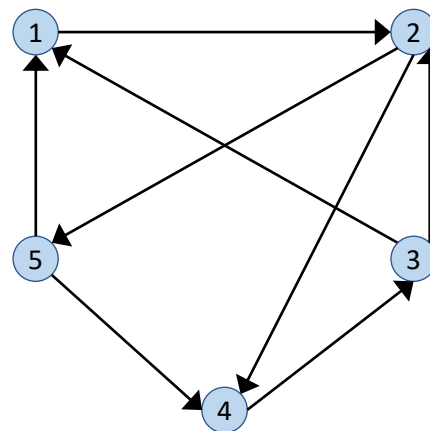
	1	2	3	4	5
1	0	1	1	0	1
2	1	0	1	1	1
3	1	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Reprezentacje maszynowe grafu: macierz sąsiedztwa

Macierz sąsiedztwa M dla grafu skierowanego $G=(V,E)$

- ma rozmiar $|V|^2$
- numer wiersza macierzy = numer wierzchołka grafu
- numer kolumny macierzy = numer wierzchołka grafu
- $M[i,j]=1$ iff w grafie G istnieje łuk (v_i, v_j)
- $M[i,j]=-1$ iff w grafie G istnieje łuk (v_j, v_i)
- $M[i,j]=0$ iff w grafie G nie istnieje łuk między v_i, v_j

Macierz sąsiedztwa dla grafu skierowanego nie jest symetryczna względem głównej przekątnej ale jej połowa jest wystarczająca do odwzorowania pełnej informacji o grafie.

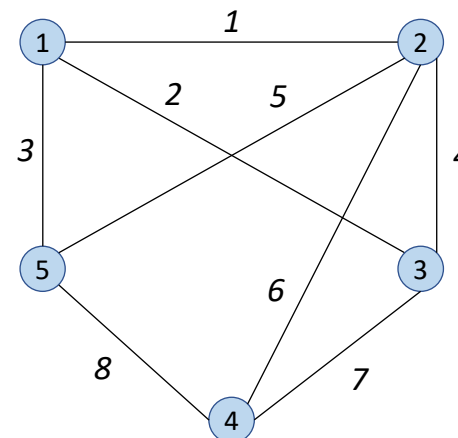


	1	2	3	4	5
1	0	1	-1	0	-1
2	-1	0	-1	1	1
3	1	1	0	-1	0
4	0	-1	1	0	-1
5	1	-1	0	1	0

Reprezentacje maszynowe grafu: macierz incydencji

Macierz incydencji M dla grafu nieskierowanego $G=(V,E)$

- ma rozmiar $|V| \times |E|$
- numer wiersza macierzy = numer wierzchołka grafu
- numer kolumny macierzy = numer\etykieta krawędzi grafu
- $M[i,j]=1$ iff w grafie G wierzchołek v_i jest incydentny z krawędzią e_j
- $M[i,j]=0$ iff w grafie G nie ma incydencji między wierzchołkiem v_i i krawędzią e_j



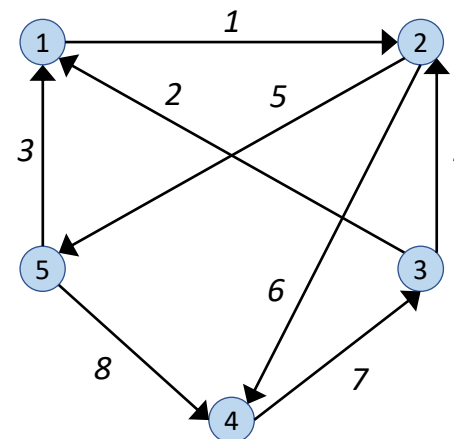
	1	2	3	4	5	6	7	8
1	1	1	1	0	0	0	0	0
2	1	0	0	1	1	1	0	0
3	0	1	0	1	0	0	1	0
4	0	0	0	0	0	1	1	1
5	0	0	1	0	1	0	0	1

Jest to bardzo nieefektywna reprezentacja. Nie poleca się jej stosowania w praktyce.

Reprezentacje maszynowe grafu: macierz incydencji

Macierz incydencji M dla grafu skierowanego $G=(V,E)$

- ma rozmiar $|V| \times |E|$
- numer wiersza macierzy = numer wierzchołka grafu
- numer kolumny macierzy = numer etykieta łuku
- $M[i,j]=-1$ iff w grafie G łuk e_j wychodzi z wierzchołka v_i
- $M[i,j]=1$ iff w grafie G łuk e_j wchodzi do wierzchołka v_i
- $M[i,j]=2$ iff w grafie G łuk e_j jest pętlą własną
- $M[i,j]=0$ iff w grafie G nie ma incydencji między wierzchołkiem v_i i łukiem e_j



	1	2	3	4	5	6	7	8
1	-1	1	1	0	0	0	0	0
2	1	0	0	1	-1	-1	0	0
3	0	-1	0	-1	0	0	1	0
4	0	0	0	0	0	1	1	-1
5	0	0	-1	0	1	0	0	-1

Reprezentacje maszynowe grafu: lista krawędzi

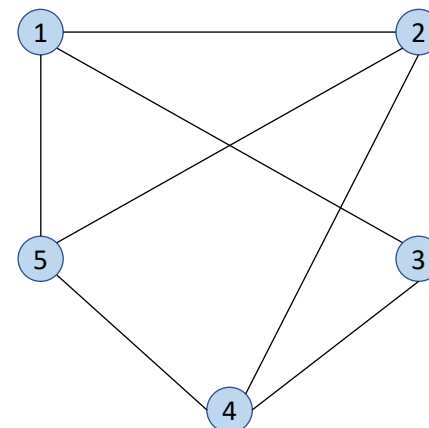
1	2
1	3
1	5
2	1
2	3
2	4
2	5
3	1
3	2
3	4
4	2
4	3
4	5
5	1
5	2
5	4

Lista krawędzi dla grafu nieskierowanego $G=(V,E)$

- ma rozmiar $|E|$
- ta reprezentacja jest często stosowana dla grafów rzadkich



1	2
1	3
1	5
2	3
2	4
2	5
3	4
4	5



Komentarz:

Listę krawędzi dla grafu nieskierowanego możemy stworzyć tak, że dla każdego wierzchołka zapiszemy na tej liście wszystkie incydentne z nim krawędzie. Wtedy lista będzie miała rozmiar $2 \times |E|$, ponieważ każda krawędź wystąpi na liście dwukrotnie.

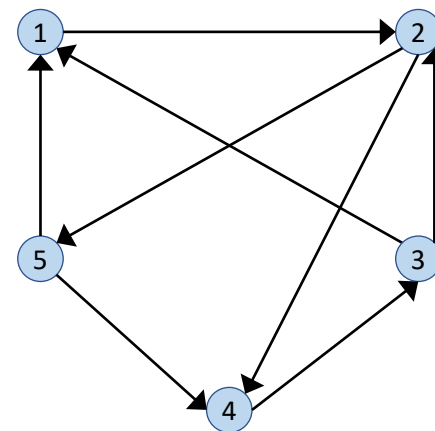
Nie jest to jednak konieczne (ani optymalne z punktu widzenia pamięci). Dlatego na liście każdą krawędź pojedynczo zachowując regułę, że wpisujemy krawędź (i,j) wtedy, gdy $i < j$, czyli np. jeśli mamy krawędź między wierzchołkiem 1 i 4, to wpisujemy ją jako $(1,4)$ ale już nie jako $(4,1)$.

Reprezentacje maszynowe grafu: lista krawędzi

Lista krawędzi dla digrafu $G=(V,E)$

- ma rozmiar $|E|$
- ta reprezentacja jest często stosowana dla grafów rzadkich

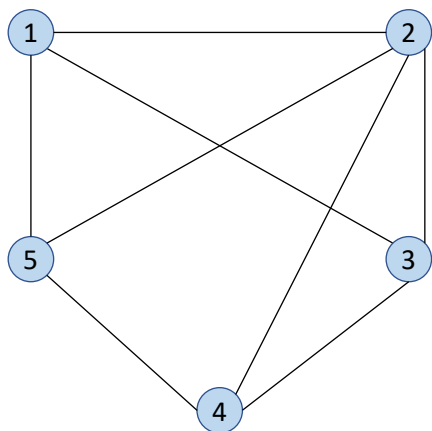
1	2
2	4
2	5
3	1
3	2
4	3
5	1
5	4



Reprezentacje maszynowe grafu: lista incydencji/sąsiedztwa

Lista incydencji (lista sąsiedztwa) dla grafu nieskierowanego $G=(V,E)$

- ma rozmiar $|V|+|E|$
- z każdym wierzchołkiem związana jest lista jego sąsiadów (wierzchołków, z którymi jest on połączony krawędzią)



1 → 2 → 3 → 5

2 → 1 → 3 → 4 → 5

3 → 1 → 2 → 4

4 → 2 → 3 → 5

5 → 1 → 2 → 4

Reprezentacje maszynowe grafu: lista następników i lista poprzedników

Lista następników dla digrafu $G=(V,E)$

- ma rozmiar $|V|+|E|$
- każdy wierzchołek jest głową listy zawierającej jego następniki

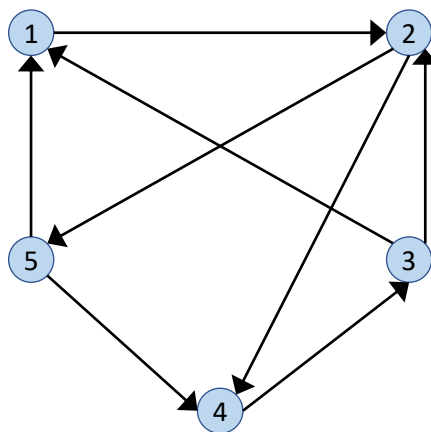
1 → 2

2 → 4 → 5

3 → 1 → 2

4 → 3

5 → 1 → 4



1 → 3 → 5

2 → 1 → 3

3 → 4

4 → 2 → 5

5 → 2

Lista poprzedników dla digrafu $G=(V,E)$

- ma rozmiar $|V|+|E|$
- każdy wierzchołek jest głową listy zawierającej jego poprzedniki

Reprezentacje maszynowe grafu: macierz grafu

Macierz grafu M dla grafu skierowanego $G=(V,E)$

- ma rozmiar $|V| \times (|V|+4)$

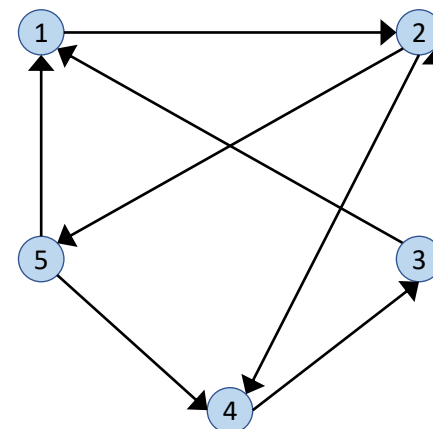
Macierz wypełniamy w taki sposób:

a) Komórki $M[i,j]$, dla $i, j = 1..|V|$ przyjmują wartości:

$$M[i,j] \in \begin{cases} \langle -|V|, -1 \rangle & \text{jeśli } (v_i, v_j) \notin E \wedge (v_j, v_i) \notin E \\ \langle 0, |V| \rangle & \text{jeśli } (v_i, v_j) \in E \wedge (v_j, v_i) \notin E \\ \langle |V|+1, 2 \cdot |V| \rangle & \text{jeśli } (v_i, v_j) \notin E \wedge (v_j, v_i) \in E \\ \langle 2 \cdot |V|+1, 3 \cdot |V| \rangle & \text{jeśli } (v_i, v_j) \in E \wedge (v_j, v_i) \in E \end{cases}$$

b) Dodatkowo:

- Komórka $M[i, |V|+1]$ zawiera pierwszy następnik v_i (0 jeśli brak)
- Komórka $M[i, |V|+2]$ zawiera pierwszy poprzednik v_i (0 jeśli brak)
- Komórka $M[i, |V|+3]$ zawiera pierwszy nieincydentny z v_i
- Komórka $M[i, |V|+4]$ zawiera pierwszy z cyklu (ta kolumna jest używana w przypadku multigrafów)



i \ j	1	2	3	4	5	6	7	8
1	-4	2	10	-4	10	2	3	1
2	8	-2	8	5	5	4	1	2
3	2	2	-5	9	-5	1	4	3
4	-4	10	3	-4	10	3	2	1
5	4	7	-5	4	-5	1	2	3

Reprezentacje maszynowe grafu: macierz grafu

Lista następników L_N

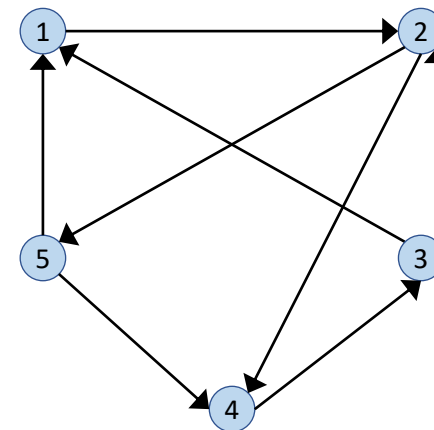
1 → 2
2 → 4 → 5
3 → 1 → 2
4 → 3
5 → 1 → 4

Lista poprzedników L_P

1 → 3 → 5
2 → 1 → 3
3 → 4
4 → 2 → 5
5 → 2

Lista brak incydencji L_B

1 → 1 → 4
2 → 2
3 → 3 → 5
4 → 1 → 4
5 → 3 → 5



W przypadku, gdy wierzchołek nie ma następnika/poprzednika/nieincydentnego na liście wpisujemy 0 (na potrzeby wypełnienia macierzy grafu). Na liście braku incydencji zaznaczamy też brak pętli własnych.

Krok 1.

Wypełnienie komórek $M[i, |V|+1]$ (tj. kolumna 6): tu w wierszu i wpisujemy pierwszy następnik wierzchołka v_i z listy L_N .

Wypełniamy komórkę $M[i, j]$ jeśli istnieje łuk $v_i \rightarrow v_j$. Do komórki $M[i, j]$ wstawiamy numer ostatniego następnika wierzchołka v_i z listy L_N .

i \ j	1	2	3	4	5	6	7	8
1		2				2		
2				5	5	4		
3	2	2				1		
4			3			3		
5	4			4		1		

Reprezentacje maszynowe grafu: macierz grafu

Lista następników L_N

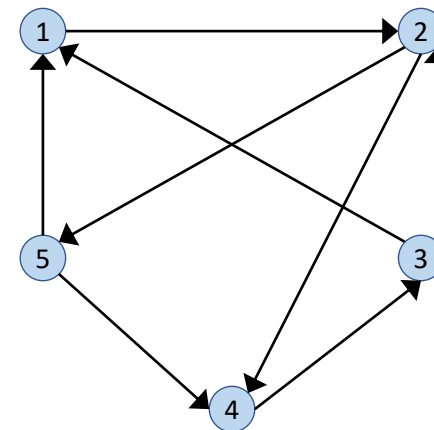
1 → 2
2 → 4 → 5
3 → 1 → 2
4 → 3
5 → 1 → 4

Lista poprzedników L_P

1 → 3 → 5
2 → 1 → 3
3 → 4
4 → 2 → 5
5 → 2

Lista brak incydencji L_B

1 → 1 → 4
2 → 2
3 → 3 → 5
4 → 1 → 4
5 → 3 → 5



W przypadku, gdy wierzchołek nie ma następnika/poprzednika/nieincydentnego na liście wpisujemy 0 (na potrzeby wypełnienia macierzy grafu). Na liście braku incydencji zaznaczamy też brak pętli własnych.

Krok 2.

Wypełnienie komórek $M[i, |V|+2]$ (tj. kolumna 7): tu w wierszu i wpisujemy pierwszy poprzednik wierzchołka v_i z listy L_P

Wypełniamy komórkę $M[i, j]$ jeśli istnieje łuk $v_j \rightarrow v_i$. Do komórki $M[i, j]$ wstawiamy sumę numeru ostatniego poprzednika wierzchołka v_i z listy L_P i liczby wierzchołków w grafie. Np. $M[2, 1]=3+5=8$

i \ j	1	2	3	4	5	6	7	8
1		2	10		10	2	3	
2	8		8	5	5	4	1	
3	2	2		9		1	4	
4		10	3		10	3	2	
5	4	7		4		1	2	

Reprezentacje maszynowe grafu: macierz grafu

Lista następników L_N

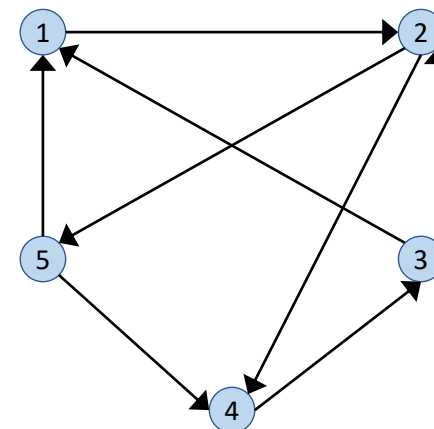
1 → 2
2 → 4 → 5
3 → 1 → 2
4 → 3
5 → 1 → 4

Lista poprzedników L_P

1 → 3 → 5
2 → 1 → 3
3 → 4
4 → 2 → 5
5 → 2

Lista brak incydencji L_B

1 → 1 → 4
2 → 2
3 → 3 → 5
4 → 1 → 4
5 → 3 → 5



W przypadku, gdy wierzchołek nie ma następnika/poprzednika/nieincydentnego na liście wpisujemy 0 (na potrzeby wypełnienia macierzy grafu). Na liście braku incydencji zaznaczamy też brak pętli własnych.

Krok 3.

Wypełnienie komórek $M[i, |V|+3]$ (tj. kolumna 8): tu w wierszu i wpisujemy pierwszy wierzchołek nieincydentny z wierzchołkiem v_i z listy L_B .

Wypełniamy komórkę $M[i, j]$ jeśli nie istnieje łuk między v_i oraz v_j . Do komórki $M[i, j]$ wstawiamy numer ostatniego wierzchołka nieincydentnego z wierzchołkiem v_i z listy L_B ze znakiem minus.

i \ j	1	2	3	4	5	6	7	8
1	-4	2	10	-4	10	2	3	1
2	8	-2	8	5	5	4	1	2
3	2	2	-5	9	-5	1	4	3
4	-4	10	3	-4	10	3	2	1
5	4	7	-5	4	-5	1	2	3

Reprezentacje maszynowe grafu

Podstawowe operacje dla różnych reprezentacji maszynowych grafu $G=(V,E)$, gdzie V to lista wierzchołków ($|V|=n$), a E to lista krawędzi/łuków ($|E|=m$) mają następujące złożoności (oszacowanie najgorszego przypadku):

Struktura danych	Sprawdzenie istnienia jednej krawędzi	Przejrzenie wszystkich sąsiadów wierzchołka*	Przejrzenie wszystkich krawędzi
Macierz sąsiedztwa	$O(1)$	$O(n)$	$O(n^2)$
Macierz incydencji	$O(m)$	$O(m)$	$O(m)$
Lista krawędzi	$O(m)$	$O(m)$	$O(m)$
Lista incydencji	$O(n)$	$O(n)$	$O(m)$
Lista następników	$O(n)$	$O(n)$	$O(m)$
Lista poprzedników	$O(n)$	$O(n)$	$O(m)$
Macierz grafu	$O(1)$	$O(n)$	$O(m)$

*tj. wierzchołków sąsiednich (w grafie nieskierowanym) lub następników/poprzedników (w digrafie)

Metody przechodzenia grafu

Metody przechodzenia grafu (w celu odwiedzenia wszystkich jego wierzchołków):

1. Przechodzenie grafu **w głąb (DFS, Depth First Search)**

metoda działa na tej samej zasadzie, co pre-order dla drzewa

2. Przechodzenie grafu **wszerz (BFS, Breadth First Search)**

metoda działa na tej samej zasadzie, co level-order dla drzewa

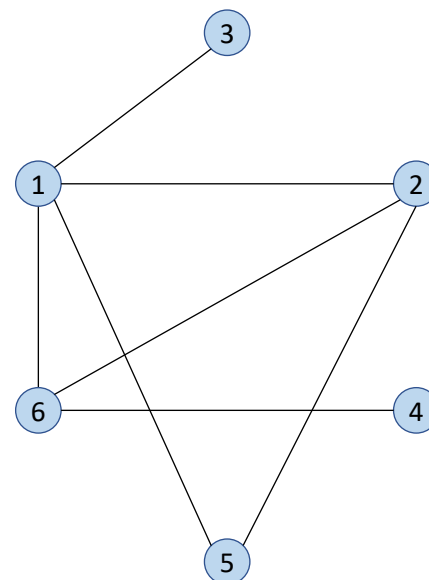
Obie metody mają zastosowanie zarówno dla grafów nieskierowanych jak i skierowanych, cyklicznych i acyklicznych. Od przechodzenia drzewa (in-order, level-order) różnią się tym, że muszą mieć zabezpieczenie przed zapętleniem (np. w przypadku cyklu w grafie).

Ich czas działania (złożoność) zależy od zastosowanej reprezentacji maszynowej grafu, która determinuje czas dostępu do potrzebnych informacji (np. znalezienia następnika bieżącego wierzchołka).

Metody przechodzenia grafu: DFS

Przechodzenie grafu **w głąb** (Depth First Search):

1. Wybierz wierzchołek startowy v (np. ten o najmniejszym indeksie lub ten o zerowym stopniu wejściowym w digrafie).
2. Odwiedź v , oznacz jako odwiedzony i odłóż na stosie.
3. Jeśli istnieją nieodwiedzone następniki v to wybierz pierwszy z nich ($v \leftarrow \text{następnik}(v)$) i idź do pkt. 2.
4. Jeśli wszystkie następniki v są odwiedzone oraz odwiedzono już wszystkie wierzchołki grafu to koniec.
5. Jeśli wszystkie następniki v są odwiedzone ale nie odwiedzono jeszcze wszystkich wierzchołków grafu zdejmij ze stosu ostatni wierzchołek.
6. Jeśli stos nie jest pusty to $v \leftarrow$ bieżący wierzchołek na szczycie stosu, w przeciwnym razie $v \leftarrow$ dowolny nieodwiedzony jeszcze wierzchołek grafu. Idź do pkt. 3.



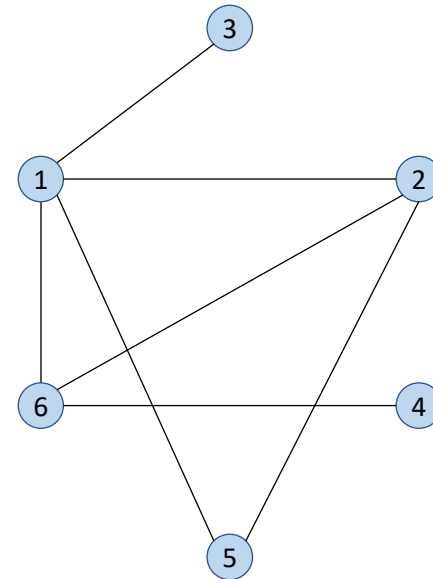
DFS dla powyższego grafu:
1, 2, 4, 6, 5, 3

Metody przechodzenia grafu: DFS

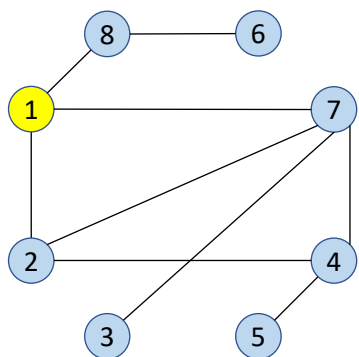
DFS: pseudokod

```
DFS(G, u)
  u.visited = true
  for each v ∈ G.Adj[u]
    if v.visited == false
      DFS(G, v)

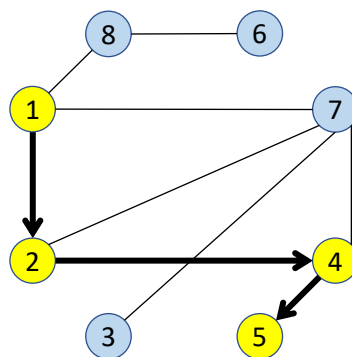
init() {
  for each u ∈ G
    u.visited = false
  for each u ∈ G
    DFS(G, u)
}
```



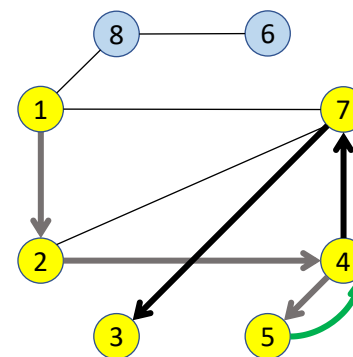
Metody przechodzenia grafu: DFS



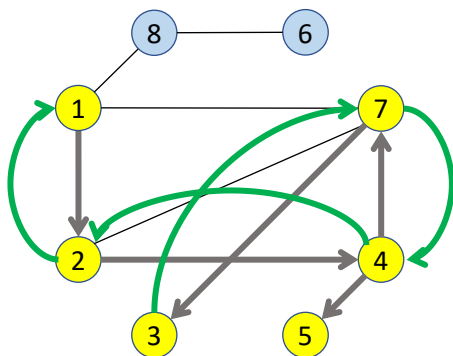
Wystartuj z wierzchołka v_1



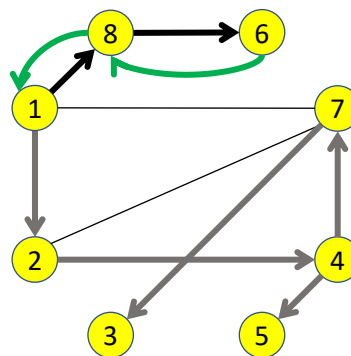
Idź w głąb grafu po nieodwiedzonych wierzchołkach dopóki możesz:
 $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$



v_5 nie ma następnika więc wycofaj się do v_4 . Idź w głąb po kolejnych nieodwiedzonych wierzchołkach
 $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 (\rightarrow 4) \rightarrow 7 \rightarrow 3$



Z v_3 wycofaj się po przebytej ścieżce do najbliższego wierzchołka z nieodwiedzonym następnikiem (v_1):
 $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 (\rightarrow 4) \rightarrow 7 \rightarrow 3 (\rightarrow 7 \rightarrow 4 \rightarrow 2 \rightarrow 1)$



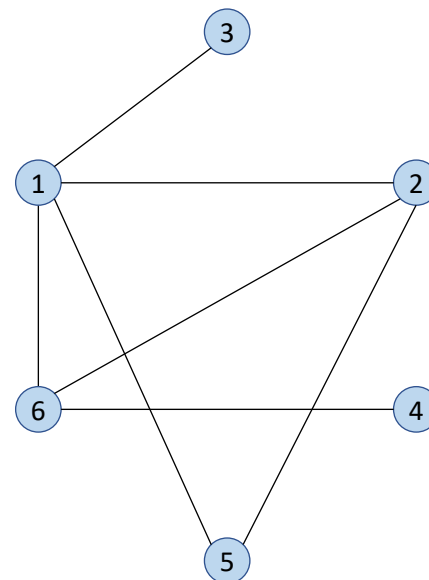
DFS:
1, 2, 4, 5, 7, 3, 8, 6

Idź w głąb po kolejnych nieodwiedzonych wierzchołkach.
 Z v_6 wycofaj się po przebytej ścieżce i zakończ: $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 (\rightarrow 4) \rightarrow 7 \rightarrow 3 (\rightarrow 7 \rightarrow 4 \rightarrow 2 \rightarrow 1) \rightarrow 8 \rightarrow 6 (\rightarrow 8 \rightarrow 1)$

Metody przechodzenia grafu: BFS

Przechodzenie grafu **wszerz** (Breadth First Search):

1. Wybierz wierzchołek startowy v (np. ten o najmniejszym indeksie lub ten o zerowym stopniu wejściowym w digrafie).
2. Odwiedź v i oznacz jako odwiedzonego.
3. Jeśli istnieją nieodwiedzone następniki wierzchołka v odwiedź je (w kolejności alfabetycznej) i oznacz jako odwiedzone.
4. Jeśli wszystkie wierzchołki grafu są odwiedzone to koniec.
5. Dla $i = 1 \dots \text{liczba-następników-}v$
 $v \leftarrow \text{następnik}_i(v)$
idź do pkt. 2.



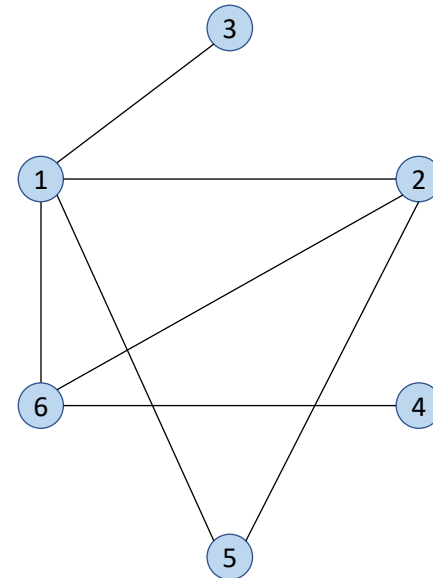
BFS dla powyższego grafu:
1, 2, 3, 5, 6, 4

Metody przechodzenia grafu: DFS

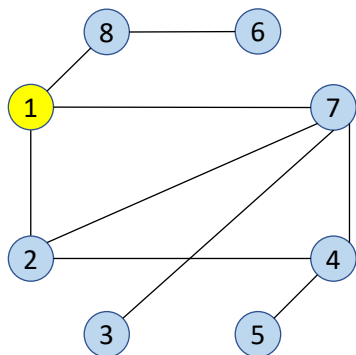
BFS: pseudokod

```
BFS(G, u)
  create stack Q
  u.visited = true
  Q.push(u)

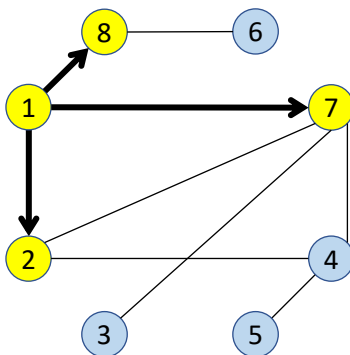
  while Q != empty
    u = Q.pop()
    for each v ∈ G.Adj[u]
      if v.visited == false
        v.visited = true
        Q.push(v)
```



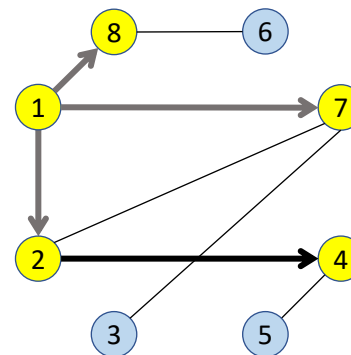
Metody przechodzenia grafu: BFS



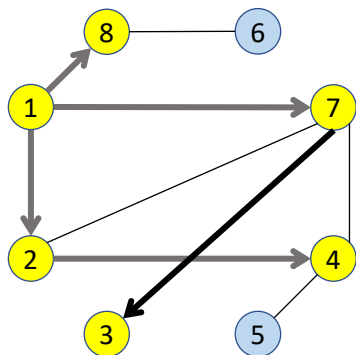
Wystartuj z wierzchołka v_1 .
 v_1 = wierzchołek bieżący



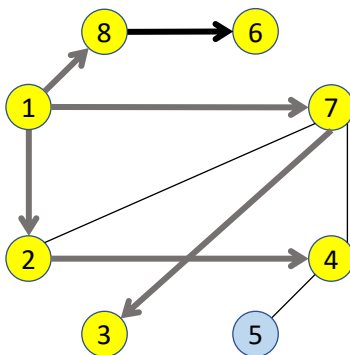
Odwiedź nieodwiedzone następniki wierzchołka bieżącego:
 $1 \rightarrow 2 \rightarrow 7 \rightarrow 8$



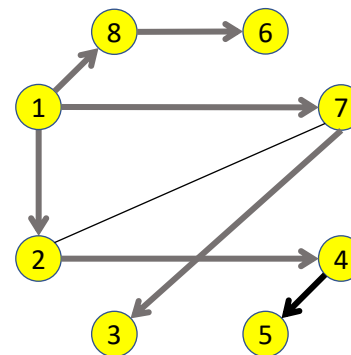
Odwiedź nieodwiedzone następniki v_2 :
 $1 \rightarrow 2 \rightarrow 7 \rightarrow 8 \rightarrow 4$



Odwiedź nieodwiedzone następniki v_7 :
 $1 \rightarrow 2 \rightarrow 7 \rightarrow 8 \rightarrow 4 \rightarrow 3$



Odwiedź nieodwiedzone następniki v_8 :
 $1 \rightarrow 2 \rightarrow 7 \rightarrow 8 \rightarrow 4 \rightarrow 3 \rightarrow 6$



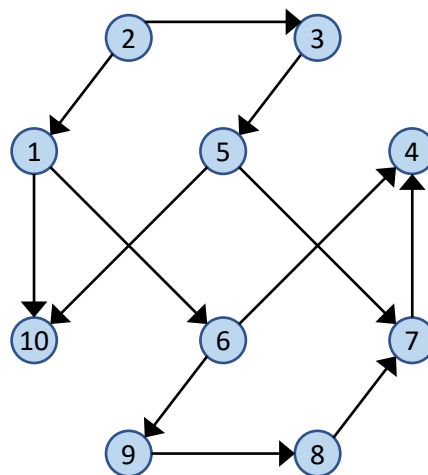
Odwiedź nieodwiedzone następniki v_4 :
 $1 \rightarrow 2 \rightarrow 7 \rightarrow 8 \rightarrow 4 \rightarrow 3 \rightarrow 6 \rightarrow 5$
 Odwiedź nieodwiedzone nast. v_3, v_6, v_5 .

BFS:
 1, 2,
 7, 8,
 4, 3,
 6, 5

Ćwiczenie

W głąb (DFS)

1, 6, 4, 9, 8, 7, 10, 2, 3, 5



Wszerz (BFS)

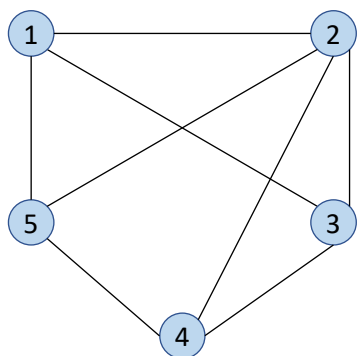
1, 6, 10, 4, 9, 8, 7, 2, 3, 5

Sortowanie topologiczne

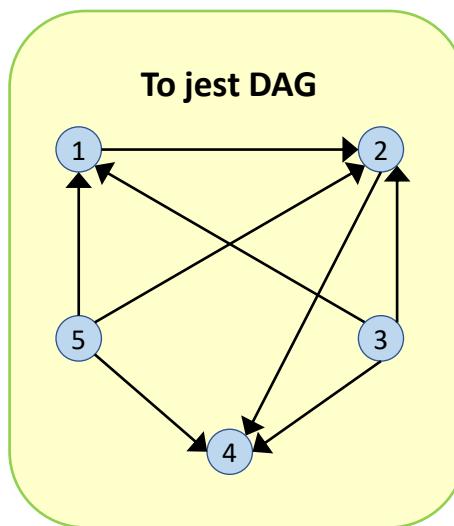
Sortowanie topologiczne grafu skierowanego $G=(V, E)$ polega na liniowym uporządkowaniu wierzchołków tego grafu w taki sposób, że dla każdej pary wierzchołków $u, v \in V$ połączonych łukiem z u do v , w porządku topologicznym u występuje przed v .

Sortowanie topologiczne grafu G jest możliwe tylko wówczas, gdy G jest **grafem skierowanym acyklicznym** (w skrócie jest to **DAG** – Directed Acyclic Graph).

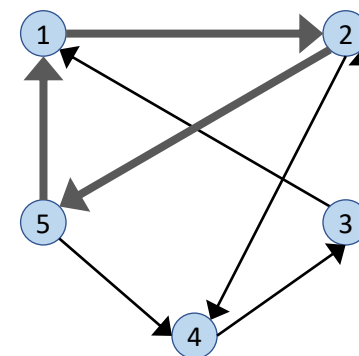
To nie jest DAG



To jest DAG



To nie jest DAG



Metody sortowania topologicznego wierzchołków grafu:

1. Algorytm Kahna

sortowanie z usuwaniem wierzchołków niezależnych (mających zerowy stopień wejściowy)

[Kahn AB (1962) Topological sorting of large networks. Communications of the ACM 5 (11):558–562]

2. Algorytm oparty na procedurze DFS

[Tarjan RE (1976) Edge-disjoint spanning trees and depth-first search. Acta Informatica,6(2):171–185]

3. Algorytmy równoległe

np. zrównoleglona wersja algorytmu Kahna

[Dekel E, Nassimi D, Sahni S (1981) Parallel matrix and graph algorithms. SIAM Journal on Computing 10(4):657–675]

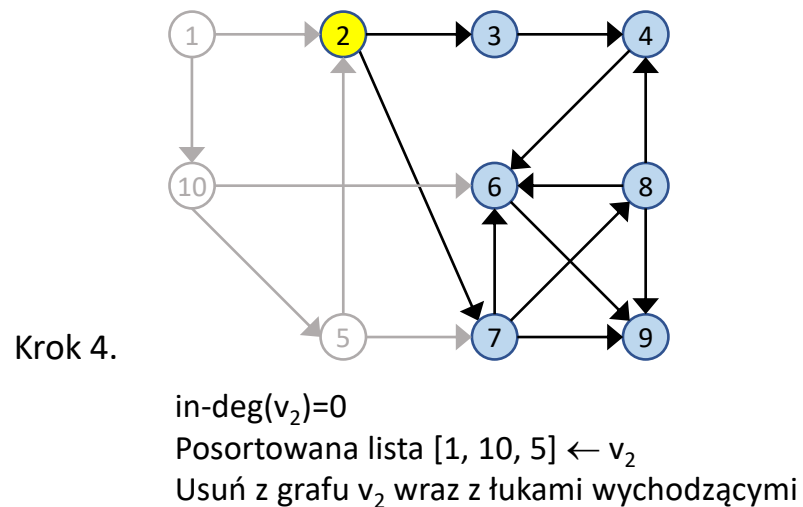
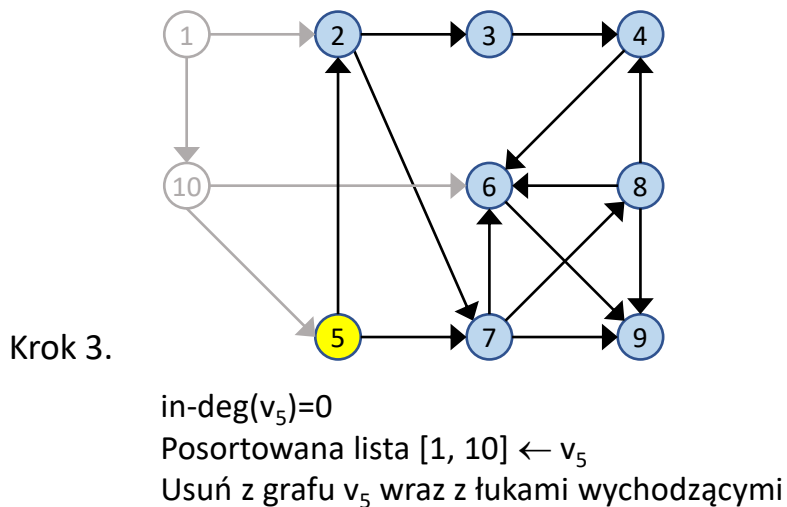
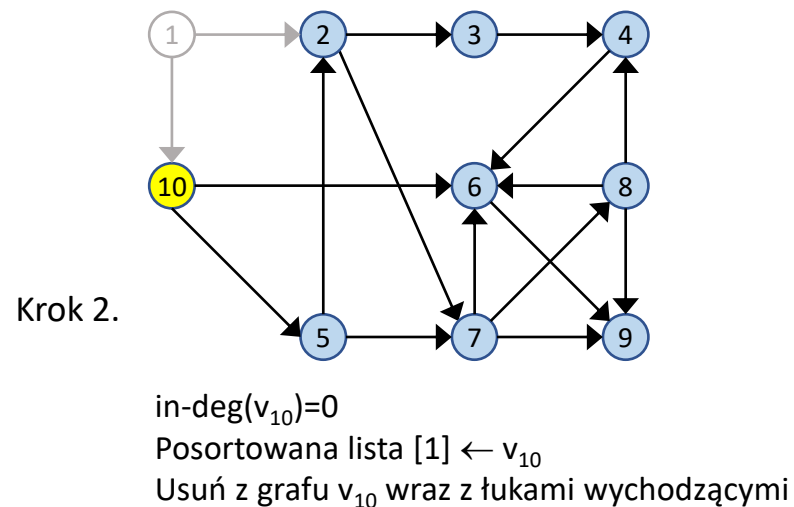
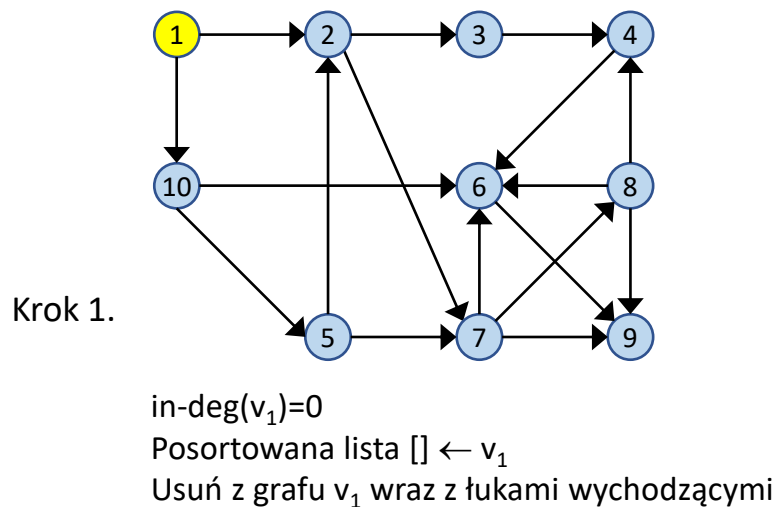
Algorytm Kahna

Dane wejściowe: skierowany acykliczny graf $G=(V,E)$, gdzie V jest niepustym zbiorem wierzchołków, E jest niepustym zbiorem łuków.

1. Utwórz listę L , na którą będą wstawiane wierzchołki w porządku topologicznym.
2. Znajdź w grafie $G=(V,E)$ wierzchołek niezależny $u \in V$, tj. taki wierzchołek, który ma zerowy stopień wejściowy (ang. in-degree): $\text{in-deg}(u) = 0$.
3. Dopisz wierzchołek u na koniec listy L : $L \leftarrow u$.
4. Usuń wierzchołek u z grafu $G=(V,E)$: $V = V \setminus \{u\}$.
5. Jeśli graf $G=(V,E)$ zawiera jeszcze jakieś wierzchołki ($V \neq \emptyset$) idź do pkt. 2.
W przeciwnym razie koniec.

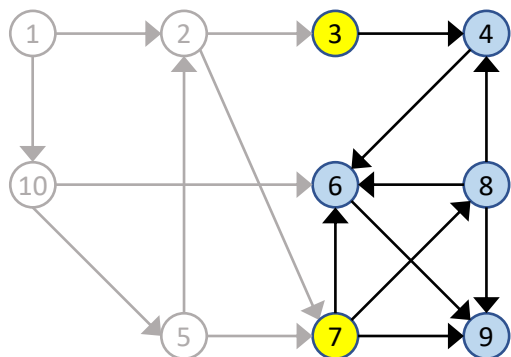
Dane wyjściowe: Lista L zawierająca wierzchołki grafu G posortowane topologicznie.

Sortowanie topologiczne: algorytm Kahna



Sortowanie topologiczne: algorytm Kahna

Krok 5.

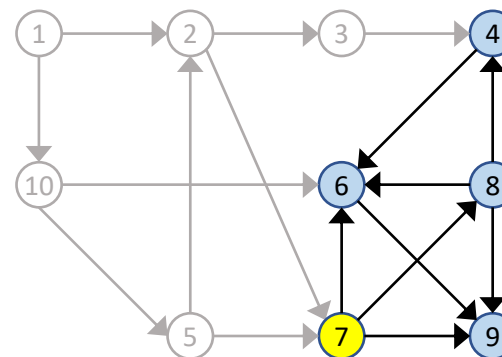


$\text{in-deg}(v_3)=\text{in-deg}(v_7)=0$

Posortowana lista $[1, 10, 5, 2] \leftarrow v_3$ (bierzemy pierwszy)

Usuń z grafu v_3 wraz z łukami wychodzącymi

Krok 6.

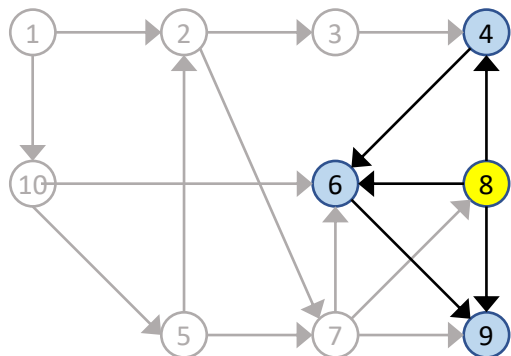


$\text{in-deg}(v_7)=0$

Posortowana lista $[1, 10, 5, 2, 3] \leftarrow v_7$

Usuń z grafu v_7 wraz z łukami wychodzącymi

Krok 7.

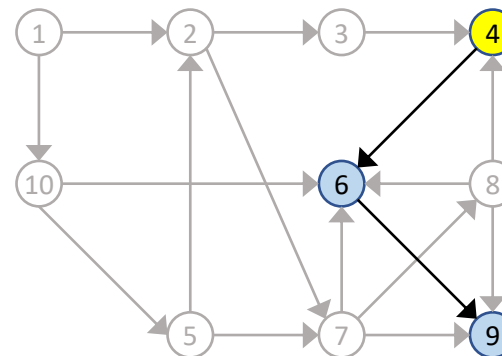


$\text{in-deg}(v_8)=0$

Posortowana lista $[1, 10, 5, 2, 3, 7] \leftarrow v_8$

Usuń z grafu v_8 wraz z łukami wychodzącymi

Krok 8.



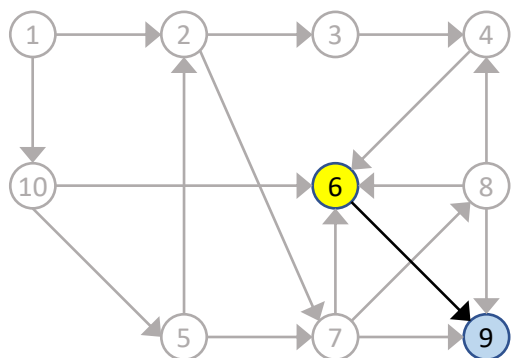
$\text{in-deg}(v_4)=0$

Posortowana lista $[1, 10, 5, 2, 3, 7, 8] \leftarrow v_4$

Usuń z grafu v_4 wraz z łukami wychodzącymi

Sortowanie topologiczne: algorytm Kahna

Krok 9.

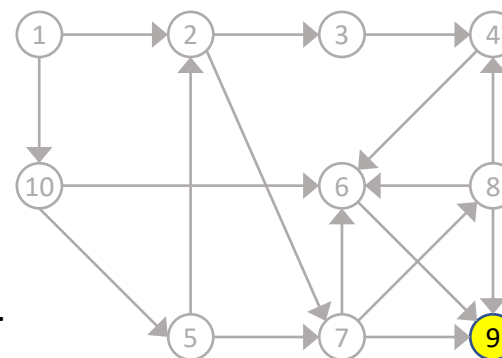


$\text{in-deg}(v_6)=0$

Posortowana lista $[1, 10, 5, 2, 3, 7, 8, 4] \leftarrow v_6$

Usuń z grafu v_6 wraz z łukami wychodzącymi

Krok 10.



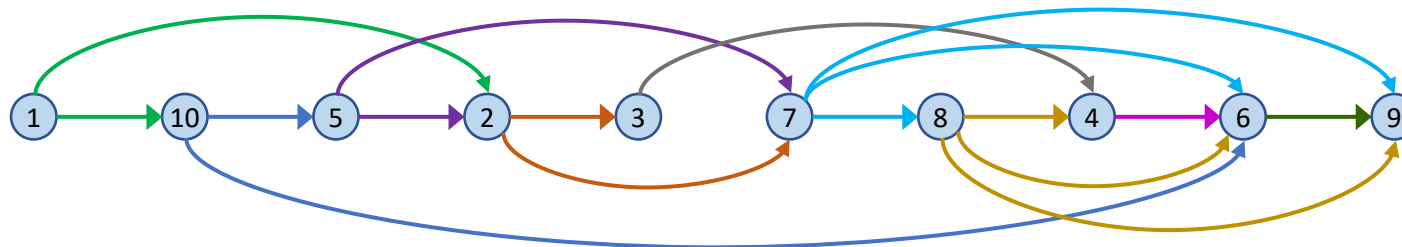
$\text{in-deg}(v_9)=0$

Posortowana lista $[1, 10, 5, 2, 3, 7, 8, 4, 6] \leftarrow v_9$

Usuń z grafu v_9 wraz z łukami wychodzącymi

Wierzchołki grafu posortowane topologicznie: **1, 10, 5, 2, 3, 7, 8, 4, 6, 9**

Graf w postaci liniowej z wierzchołkami uporządkowanymi topologicznie



Sortowanie topologiczne: algorytm oparty na DFS

Algorytm oparty na DFS

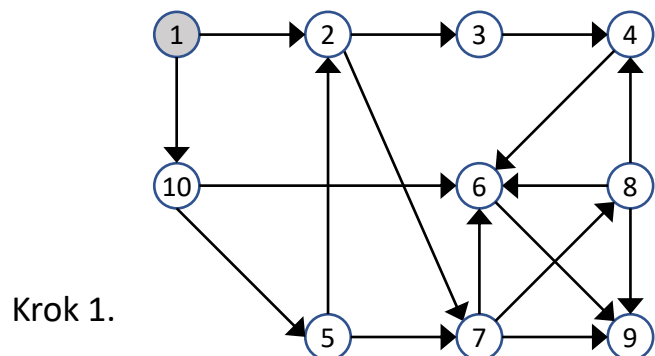
Dane wejściowe: skierowany acykliczny graf $G=(V,E)$, gdzie V jest niepustym zbiorem wierzchołków, E jest niepustym zbiorem łuków.

1. Przypisz wszystkim wierzchołkom grafu G kolor biały.
2. Utwórz listę L , na którą będą wstawiane wierzchołki w porządku topologicznym.
3. Utwórz stos S , na który będą wstawiane przetworzone wierzchołki.*
4. Wybierz w grafie $G=(V,E)$ biały wierzchołek startowy u (najlepiej jeśli będzie to wierzchołek niezależny).
5. Odwiedź wierzchołek u , pokoloruj go na szaro.
6. Jeśli istnieją białe następniki u to wybierz jeden z nich: $u \leftarrow \text{następnik}(u)$. Idź do pkt. 5.
7. Jeśli u nie ma białego następnika, pokoloruj u na czarno i wstaw na stos: $S \leftarrow u$.
8. Cofnij się do szarego poprzednika u na przebytej ścieżce jeśli istnieje taki poprzednik: $u \leftarrow \text{poprzednik}(u)$ oraz przejdź do pkt. 6.
9. Jeśli u nie ma szarego poprzednika i wszystkie wierzchołki grafu G są czarne (są na stosie), idź do pkt. 11.
10. Jeśli u nie ma szarego poprzednika i w grafie G są jeszcze białe wierzchołki, idź do pkt. 4.
11. Dla $i=1..|V|$: zdejmij i -ty wierzchołek ze stosu S i wpisz na koniec listy posortowanej: $L \leftarrow S(i)$.

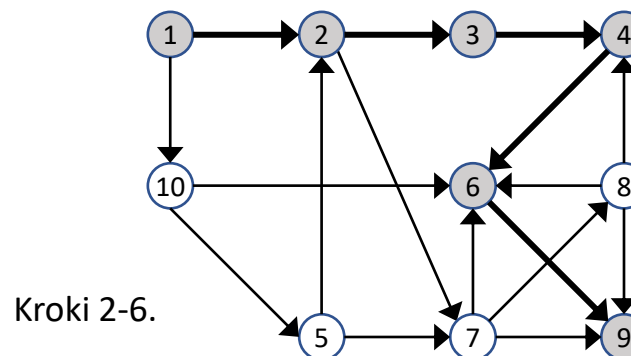
Dane wyjściowe: Lista L zawierająca wierzchołki grafu G posortowane topologicznie.

*Zamiast implementować stos, wierzchołki można od razu wstawiać na listę L , ale zawsze na początek listy!

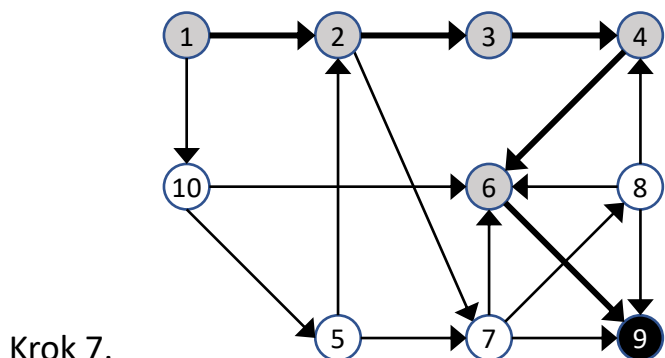
Sortowanie topologiczne: algorytm oparty na DFS



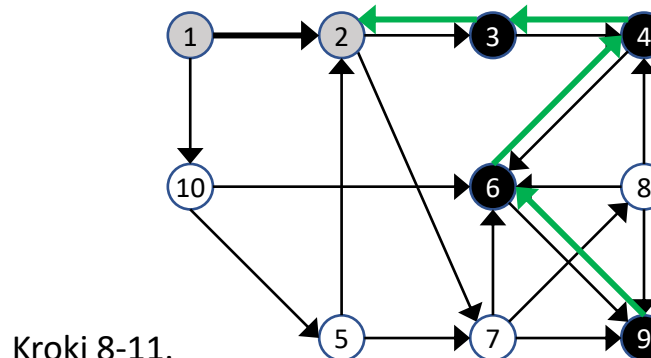
Odwiedź wierzchołek startowy: v_1
Pokoloruj go na szaro.



Idź w głąb grafu (DFS) kolorując odwiedzane wierzchołki na szaro.

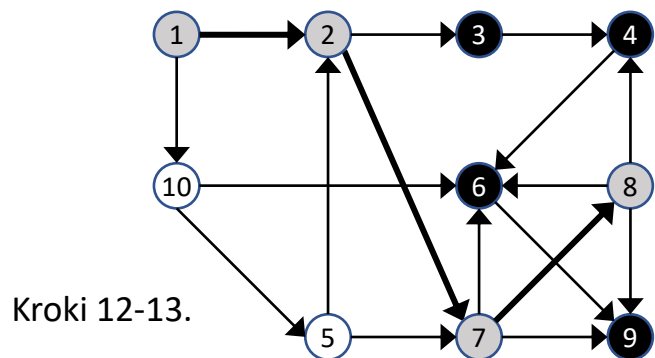


v_9 nie ma białych następników. Pokoloruj go na czarno i wrzuć na stos: $S[] \leftarrow v_9$

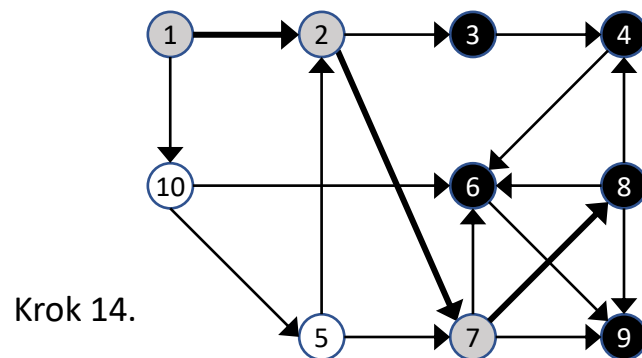


Wycofaj się po ścieżce do pierwszego wierzchołka, mającego białe następniki. Wierzchołki, z których się cofasz koloruj na czarno i wrzucaj na stos: $S[3,4,6,9]$

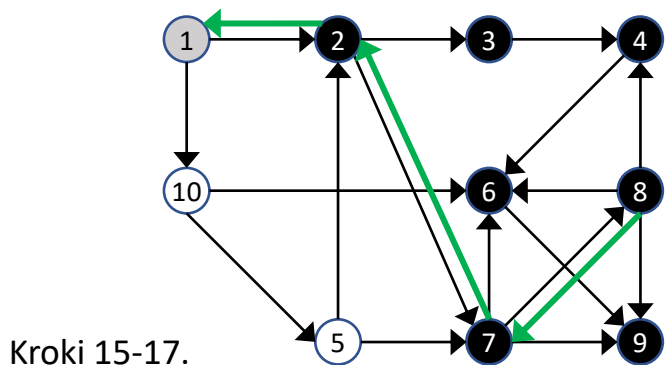
Sortowanie topologiczne: algorytm oparty na DFS



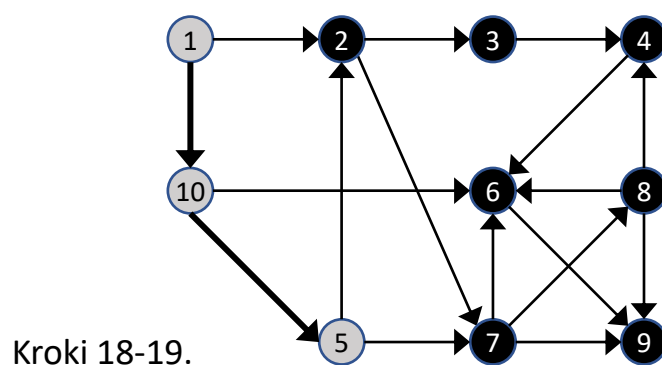
Idź w głąb grafu (DFS) po białych wierzchołkach kolorując je na szaro.



v_8 nie ma białych następników. Pokoloruj go na czarno i wrzuć na stos: $S[3,4,6,9] \leftarrow v_8$

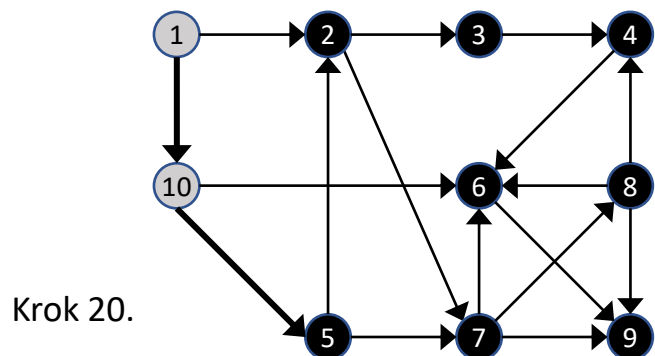


Wycofaj się po ścieżce do pierwszego wierzchołka, z białymi następnikami. Wierzchołki, z których się cofasz koloruj na czarno i wrzucaj na stos: $S[2,7,8,3,4,6,9]$

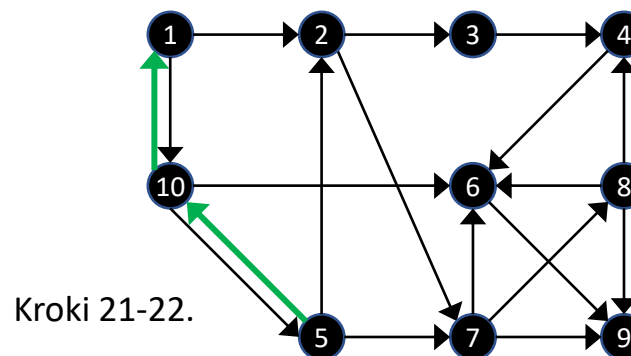


Idź w głąb grafu (DFS) po białych wierzchołkach kolorując je na szaro.

Sortowanie topologiczne: algorytm oparty na DFS



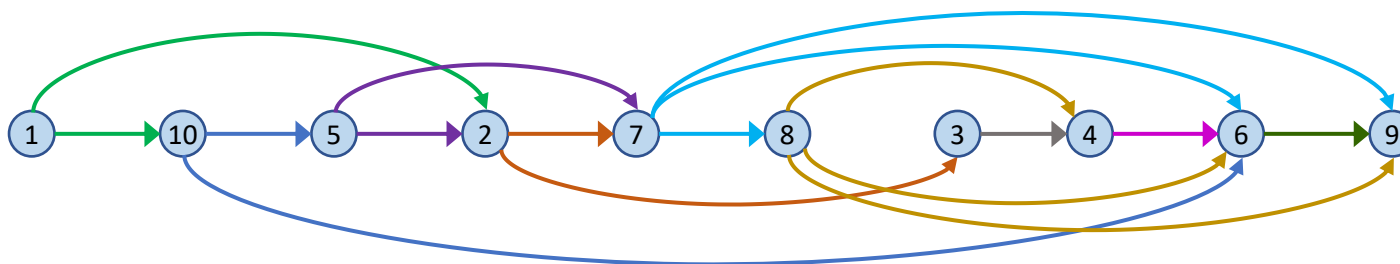
v_5 nie ma białych następników.
 Pokoloruj go na czarno i wrzuć na stos:
 $S[2,7,8,3,4,6,9] \leftarrow v_5$



Wycofaj się po ścieżce do pierwszego wierzchołka.
 Wierzchołki, z których się cofasz koloruj na czarno i wrzucaj na stos:
 $S[1,10,5,2,7,8,3,4,6,9]$
 Wszystkie wierzchołki w grafie są czarne. Koniec.

Wierzchołki grafu posortowane topologicznie: 1, 10, 5, 2, 7, 8, 3, 4, 6, 9

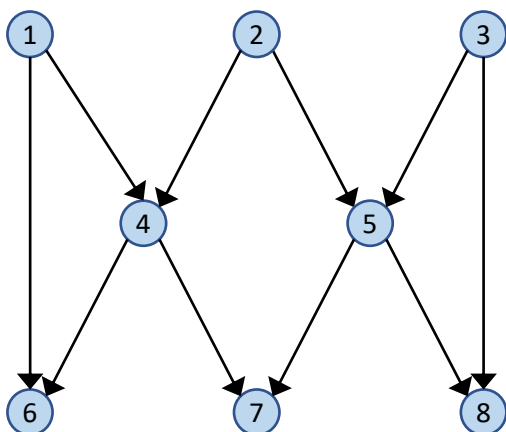
Graf w postaci liniowej z wierzchołkami uporządkowanymi topologicznie



Sortowanie topologiczne

W jednym grafie skierowanym G może istnieć **wiele różnych porządków topologicznych** jego wierzchołków.

Przykładowy graf $G=(V,E)$
 $V=\{1,2,3,4,5,6,7,8\}$
 $E=\{(1,4),(1,6),(2,4),(2,5),(3,5),$
 $(3,8),(4,6),(4,7),(5,7),(5,8)\}$



Różne uporządkowania topologiczne
wierzchołków grafu G :

- 1, 2, 3, 4, 5, 6, 7, 8
- 1, 2, 4, 6, 3, 5, 8, 7
- 3, 2, 5, 8, 1, 4, 7, 6
- 3, 2, 1, 5, 4, 8, 7, 6
- 1, 3, 2, 4, 6, 5, 8, 7
- ...

Sortowanie topologiczne a cykl w grafie: Algorytm Kahna

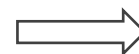
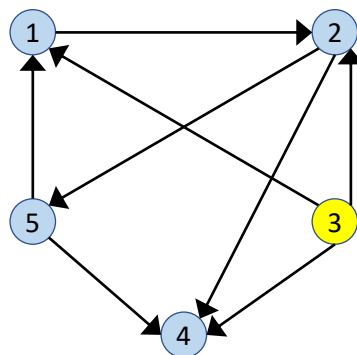
Jeśli w grafie, którego wierzchołki sortujemy topologicznie istnieje cykl, to może on zostać wykryty (tj. algorytm może stwierdzić istnienie cyklu). Kiedy to się dzieje:

Algorytm Kahna

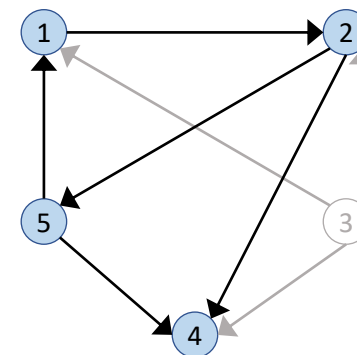
- w każdym kroku szuka w grafie niezależnego wierzchołka (wierzchołka, do którego nie wchodzi żadne łuki), aby go wstawić na posortowaną listę, a następnie usuwa go z grafu wraz z łukami wychodzącymi.
- Jeśli w którymś momencie algorytm **nie znajdzie ani jednego wierzchołka niezależnego, ale w grafie nadal są jakieś wierzchołki** (tzn. nie wszystkie wierzchołki grafu wejściowego znalazły się na liście posortowanej), to znaczy, że w pozostałej części grafu znajduje się cykl.
- W takim przypadku sortowanie topologiczne zostaje przerwane (nie można uporządkować topologicznie wierzchołków grafu).

Przykład

$\text{in-deg}(v_3)=0$
Posortowana lista $[\] \leftarrow v_3$
Usuń z grafu v_3 wraz z
łukami wychodzącymi



Nie ma wierzchołka
bez wchodzących
krawędzi. Cykl!



Sortowanie topologiczne a cykl w grafie: algorytm oparty na DFS

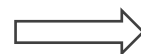
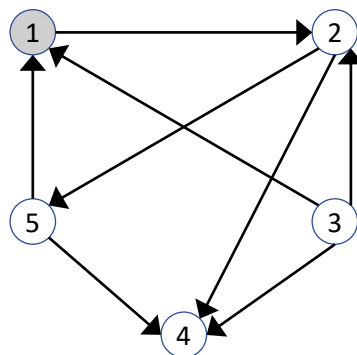
Jeśli w grafie, którego wierzchołki sortujemy topologicznie istnieje cykl, to może on zostać wykryty (tj. algorytm może stwierdzić istnienie cyklu). Kiedy to się dzieje:

Algorytm oparty na DFS

- wchodząc do białego wierzchołka przeszukuje jego następniki. Następniki mogą być białe (nie były jeszcze odwiedzone i można do nich wejść w kolejnym kroku) lub czarne (zostały już odwiedzone i wpisane na posortowaną listę, a procedura wycofała się z takich wierzchołków).
- Jeśli w którymś momencie okaże się, że **biały wierzchołek, który właśnie odwiedzamy ma szarego następnika**, to znaczy, że w grafie istnieje cykl.
- Do tego cyklu należy na pewno wierzchołek, w którym jesteśmy oraz jego szary następnik. W takim przypadku sortowanie topologiczne zostaje przerwane (nie można uporządkować topologicznie wierzchołków grafu).

Przykład

Odwiedź wierzchołek startowy v_1 . Pokoloruj go na szaro.



Idź w głąb grafu (DFS) po białych wierzchołkach i koloruj je na szaro. v_5 ma szarego następnika. Cykl!

