# Wstęp do informatyki

Wykład 13

Grafy – pojęcia, reprezentacja, przeglądanie grafu

Wstęp do informatyki

Instytut Informatyki UWr

# Temat wykładu

- Grafy – definicje
- Reprezentacja grafu w pamięci komputera
- Przeglądanie grafów (wszerz, w głąb)
- Zastosowania przeglądania

# Graf skierowany

**Graf skierowany** G(V,E):

- V – skończony zbiór wierzchołków
- $E \subset V \times V$ – zbiór krawędzi (pary wierzchołków)

**Oznaczenia**:

n – liczba wierzchołków

m – liczba krawędzi

**Obserwacja**:

$m \leq n^2$

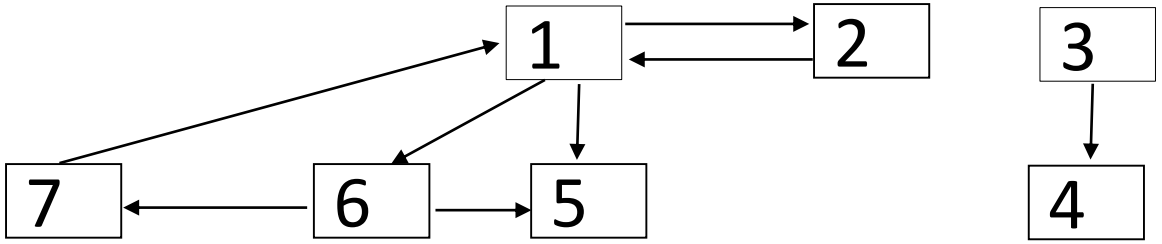
# Graf skierowany

**Przykład**

V = { 1, 2, 3, 4, 5, 6, 7 }

E ={ (1, 5), (1, 2), (6,7), (1, 6), (3,4), (7,1), (2,1), (6,5) }

n = 7, m = 8




Zastosowania

- połączenia drogowe, kolejowe, lotnicze
- relacje zależności w firmie
- zależności między przedmiotami w programie studiów!
- i inne…

# Graf nieskierowany (prosty)

Graf $G(V, E)$ jest grafem nieskierowanym gdy spełniony jest warunek:

$$(i, j) \in E \text{ wtw } (j, i) \in E$$

Inny sposób definicji grafu nieskierowanego:

W grafie nieskierowanym krawędzie to dwuelementowe **zbiory** wierzchołków (a **nie** uporządkowane **pary**)

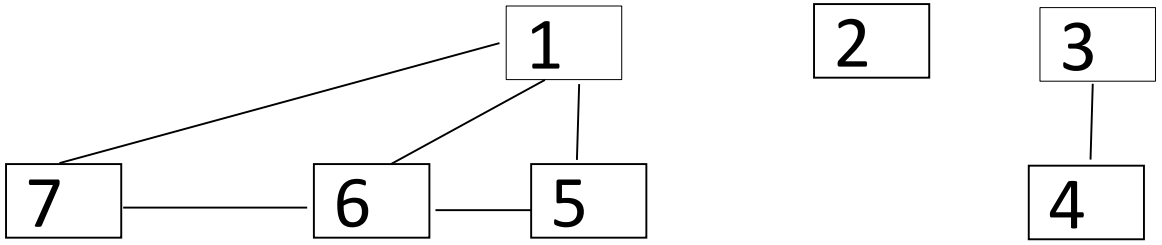
# Graf nieskierowany (prosty)

**Przykład**

$V = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$E = \{ \{1, 5\}, \{6,7\}, \{1, 6\}, \{3,4\}, \{7,1\}, \{6,5\} \}$

$n = 7, m = 6$




lub równoważnie:

$E = \{ (1, 5), (5,1), (6,7), (7,6), (1,6), (6,1), (3,4), (4,3), (7,1), (1,7), (6,5), (5,6) \}$

# Graf z wagami

Graf z wagami G(V,E,w) to graf G(V,E) wraz z funkcją $w: E \rightarrow \Re$ przypisującą krawędziom liczby (np. rzeczywiste) nazywane wagami

Uwagi:

- Graf z wagami może być grafem skierowanym bądź nieskierowanym
- wagi mogą być ograniczone do podzbioru liczb rzeczywistych bądź przyjmować wartości z innego zbioru;
- zazwyczaj (na tym wykładzie) wagi to liczby nieujemne.

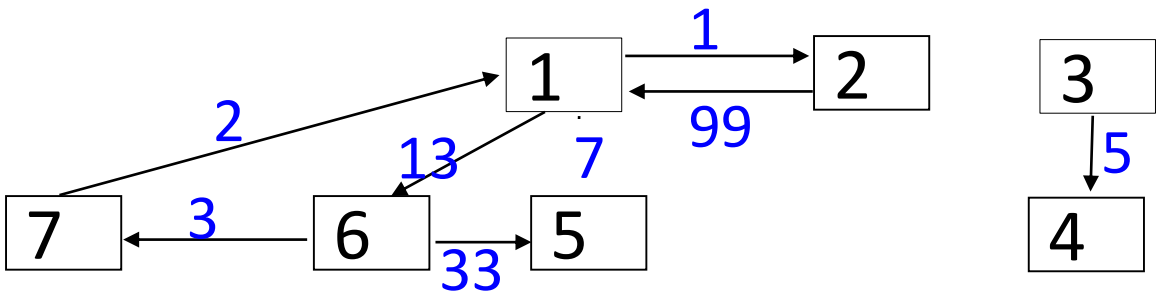
# Graf z wagami (skierowany)

**Przykład**

V = { 1, 2, 3, 4, 5, 6, 7 }

E ={ (1, 5), (1, 2), (6,7), (1, 6), (3,4), (7,1), (2,1), (6,5) }

n = 7, m = 8




w((1,5)) = 7

w((1,2)) = 1

w((6,7)) = 3

w((1,6)) = 13

w((3,4)) = 5

w((7,1)) = 2

w((2,1)) = 99

w((6,5)) = 33

# Graf prosty – sąsiedztwo, stopień

Def. Wierzchołek $w$ jest **sąsiadem** wierzchołka $v$ w grafie prostym $G(V, E)$ gdy $\{v, w\} \in E$

Def. **Stopień** $d(v)$ wierzchołka $v$ w grafie prostym $G(V, E)$ to liczba jego sąsiadów, czyli moc zbioru

$$\{ w \mid \{v, w\} \in E \}$$

# Graf skierowany – stopień

Def. **Stopień wyjściowy** wierzchołka $v$ w grafie skierowanym $G(V, E)$ to liczba krawędzi o początku w $v$ czyli moc zbioru

$\{ w \mid (v, w) \in E \}$

Def. **Stopień wejściowy** wierzchołka $v$ w grafie skierowanym $G(V, E)$ to liczba krawędzi o końcu w $v$ czyli moc zbioru

$\{ w \mid (w, v) \in E \}$

# Graf – ścieżki

Def. **Ścieżka** to ciąg wierzchołków $v_1, v_2, \ldots, v_k$ taki, że $(v_i, v_{i+1}) \in E$ dla każdego $i = 1,2, \ldots, k-1$.

Def. **Ścieżka prosta**: wszystkie wierzchołki na ścieżce są różne.

Def. **Długość ścieżki** $v_1, v_2, \ldots, v_k$ jest równa $k-1$, czyli liczbie krawędzi na ścieżce.

# Graf – ścieżki

Def. **Cykl** to ścieżka zaczynająca i kończąca się w tym samym wierzchołku

Def. **Osiągalność**

Wierzchołek $w$ jest osiągalny z $v$ jeśli istnieje ścieżka $v = v_1, v_2, \ldots, v_k = w$

# Graf – reprezentacja

**Macierz sąsiedztwa:**

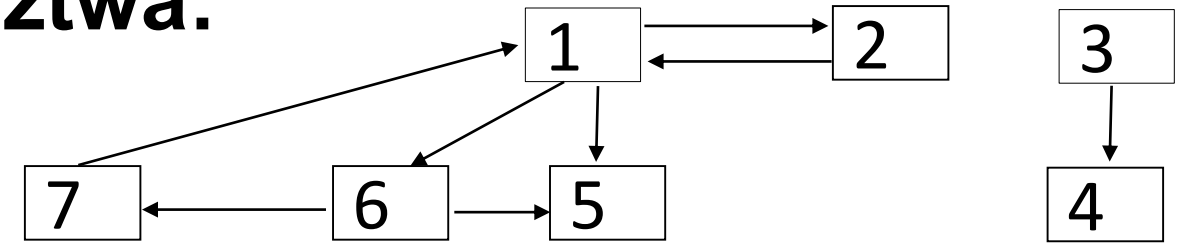
Graf $G(V, E)$ reprezentowany przez tablicę **dwuwymiarową** int a[n][n]

- a[i][j] = 1 gdy $(i, j) \in E$ **lub**

  a[i][j] = w((i,j)) dla grafu z wagami
- a[i][j] = 0 gdy $(i, j) \notin E$

# Graf – reprezentacja

**Macierz sąsiedztwa:**




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

Przykład ścieżki: 1,6,5

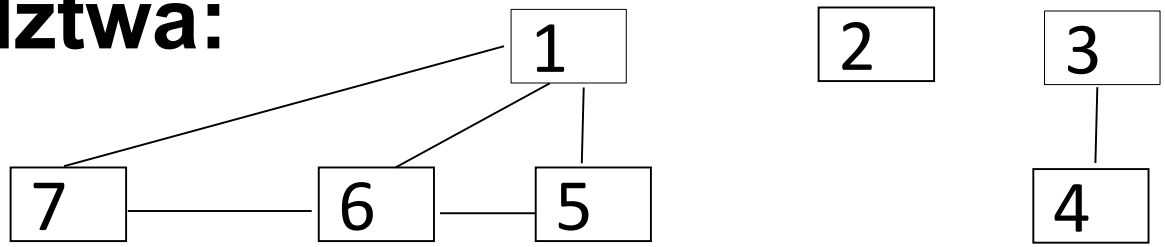
Przykład cyklu: 1,6,7,1

Stopień wyjściowy 1: 3

Stopień wejściowy 1: 2

# Graf – reprezentacja

**Macierz sąsiedztwa:**




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

Przykład ścieżki: 1,5,6,7

Przykład cyklu: 1,5,6,1

d(1)=3

d(2)=0

# Graf – reprezentacja

**Listy sąsiadów:**

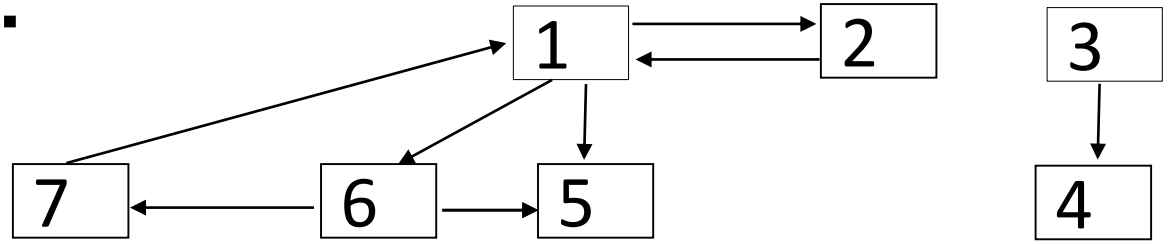
Graf $G(V, E)$ reprezentowany przez tablicę wiązanych list a[n]

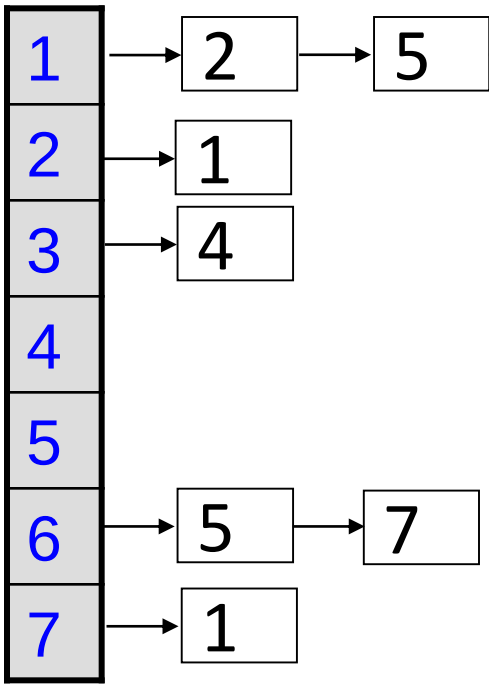
- a[i] to lista sąsiadów a[i], czyli zbiór

$$\{ j \ : (i, j) \in E \}$$

# Graf – reprezentacja

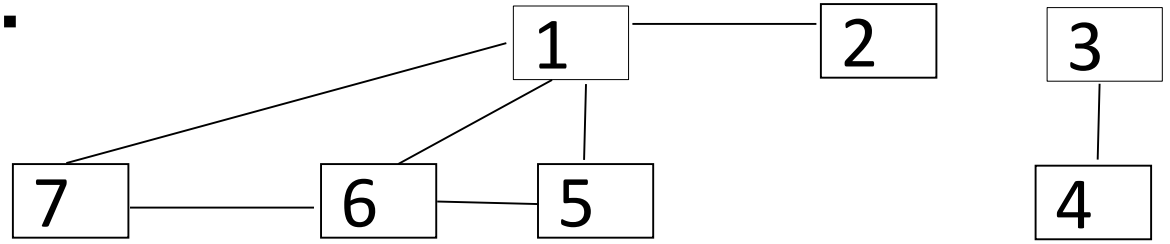
**Listy sąsiadów:**

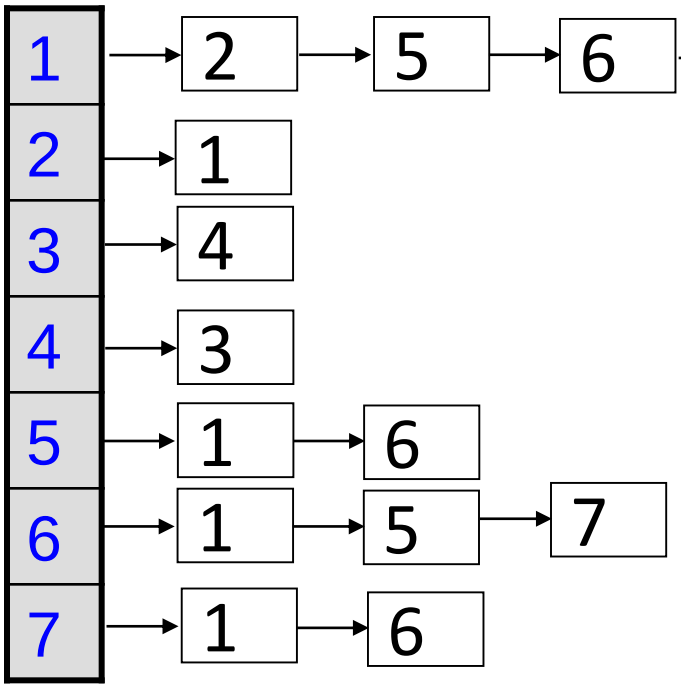


# Graf – reprezentacja

**Listy sąsiadów:**



# Reprezentacje i ich własności

**Złożoność pamięciowa:**

- macierz sąsiedztwa: $n^2$ (nawet gdy m=O(n))
- listy sąsiadów: O(n+m)

**Czas operacji (na niebiesko szybsze rozwiązanie):**

| Operacja | Tablicowa | Listowa |
|---|---|---|
| Sprawdź(u,v) | O(1) | O(d(u)) |
| Dodaj(u,v) | O(1) | O(d(u)) |
| Usuń(u,v) | O(1) | O(d(u)) |
| Przejrzyj sąsiadów u | O(n) | O(d(u)) |

# Graf – przeszukiwanie

**Problem przeszukiwania**

**Wejście:** graf $G(V, E)$, oraz $s \in V$, gdzie:

- $V = \{1, 2, \ldots, n\}$
- $E$ reprezentowane przez listy sąsiadów a[1..n]

**Wynik:**

- wyznaczenie wszystkich wierzchołków osiągalnych z s

# Graf – przeszukiwanie

**Struktury danych:**

B – zbiór wierzchołków odwiedzonych, dla których nie zbadaliśmy jeszcze wszystkich wychodzących krawędzi

odw – tablica reprezentująca zbiór już odwiedzonych wierzchołków

a – tablica list; reprezentacja zbioru krawędzi E

L – tablica list; L[v] to lista niesprawdzonych krawędzi dla v

**Wynik działania algorytmu dla $v = 1, .., n$:**

- odw[v]=1, gdy v osiągalny z s
- odw[v]=0 w przeciwnym przypadku

# Graf – przeszukiwanie

**Struktury danych:**

B – zbiór wierzchołków odwiedzonych, dla których nie zbadaliśmy jeszcze wszystkich wychodzących krawędzi

odw – tablica reprezentująca zbiór już odwiedzonych wierzchołków

a – tablica list; reprezentacja G

L – tablica list; L[v] to lista niesprawdzonych krawędzi dla v

**Idea rozwiązania:**

- B ← { s } i zaznacz s jako odwiedzony
- Dopóki B niepusty:
  - Weź v z B
  - Jeśli wszystkie krawędzie wychodzące z v były sprawdzone: usuń v z B, w przeciwnym przypadku:
    - weź niesprawdzoną krawędź (v,w)
    - jeśli w nieodwiedzony: dodaj w do B i zaznacz jako odwiedzony.

# Graf – przeszukiwanie

**Struktury danych:**

B – zbiór wierzchołków odwiedzonych, dla których nie zbadaliśmy jeszcze wszystkich wychodzących krawędzi

odw – tablica reprezentująca zbiór odwiedzonych wierzchołków

a – tablica list; reprezentacja G (zbioru krawędzi)

L – tablica list; L[v] to lista niesprawdzonych krawędzi wychodzących z v

**Operacje na liście:**

- pierwszy(X) – wartość pierwszego elementu na liście
- ogon(X) – lista X bez jej pierwszego elementu

**Operacje na zbiorze**

- Weź(Z) – element zbioru Z lub NULL gdy Z pusty
- Dodaj x do Z: $Z \leftarrow Z + \{ x \}$
- Usuń x z Z: $Z \leftarrow Z - \{ x \}$

# Graf – przeszukiwanie

**Algorytm Inicjalizuj(G)**

1. Dla v=1,2,..,n:
   - odw[v] ← 0; L[v] ← a[v]

**Algorytm Przeszukaj(G,s):**

1. B ← { s }, odw[s] ← 1
2. Dopóki B ≠∅
   1. v ← weź(B)
   2. Jeśli L[v] pusta: B ← B – { v }, w przeciwnym przypadku
      a) w ← pierwszy(L[v])
      b) L[v] ← ogon(L[v])
      c) jeśli odw[w]=0:
         – B ← B + { w }
         – odw[w] ← 1

# Graf – przeszukiwanie - poprawność

**Fakt**

Jeśli v jest osiągalny z s, to v zostanie odwiedzony w trakcie Przeszukaj(G,s).

**Dowód**

**Obserwacje**:

- każdy odwiedzony węzeł jest dodawany do B (jeden raz)
- v osiągalny z s wtw istnieje ścieżka z s do v

**Indukcja** ze

względu na długość ścieżki z s do v:

- **Krok bazowy**: Jeśli ścieżka długości 1 (v sąsiadem s) – v zostanie odwiedzony dzięki 2.1 (gdyż s umieszczamy w B, w 2.1 sprawdzamy jego sąsiadów)

# Graf – przeszukiwanie - poprawność

- **Krok indukcyjny**:

  **Zał**.: Wierzchołki, do których prowadzą ścieżki o długości <k są odwiedzone i dodane do B.

  **Teza**: Wierzchołki, do których prowadzą ścieżki o długości k są odwiedzone i dodane do B.

  **Dowód**:

  Niech v – osiągalny z s, ścieżka $s=v_1,\ldots,v_k,v_{k+1}=v$. Wówczas:

  – $v_1,\ldots,v_k$ to też ścieżka

  – z założenia indukcyjnego wszystkie wierzchołki na tej ścieżce będą odwiedzone (i dodane do B)

  – v zostanie wybrany z $L[v_k]$ i odwiedzony (o ile nie będzie odwiedzony wcześniej)

# Graf – przeszukiwanie - czas

**Własności**

- Każdy wierzchołek co najwyżej raz dodany do B
- Wierzchołek v zostanie wybrany d(v)+1 razy w kroku 2.1, po czym zostanie usunięty

**Wniosek**

Algorytm zakończy działanie po

$$\sum_{v \in V} (d(v)+1) = O(n+m)$$

obrotach pętli 2.

# Spójność, składowe

Def. **Podgraf** grafu G(V,E) indukowany przez zbiór $V' \subseteq V$ to graf G'(V',E') gdzie

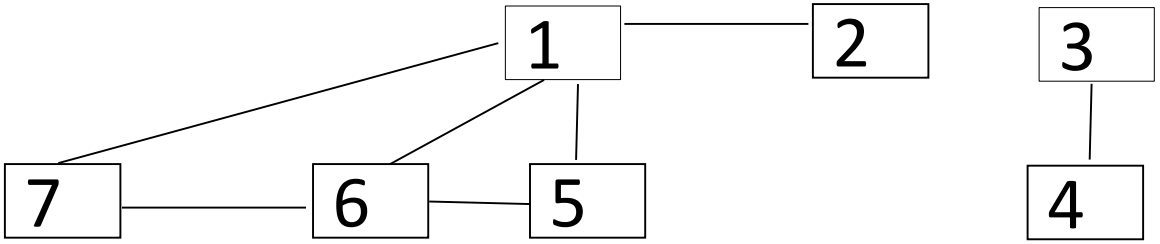
$$(i,j) \in E' \Leftrightarrow i,j \in V' \text{ oraz } (i,j) \in E$$

Def. Graf prosty G(V,E) nazywamy **spójnym** wtw gdy dla każdej pary wierzchołków v, $v' \in E$ istnieje ścieżka łącząca v i v'.

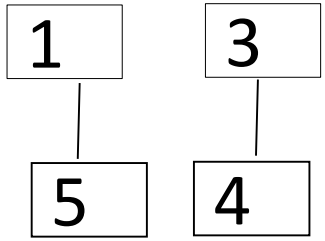
Def. **Składową spójności** grafu prostego G(V,E) nazywamy podgraf G'(V',E') grafu G indukowany przez $V' \subseteq V$, taki że G' jest spójny i $G' \cup \{v\}$ nie jest spójny dla każdego $v \in V - V'$

# Spójność, składowe

Graf G(V,E):




Podgraf indukowany przez 1,5,3,4:




Składowe spójności grafu G:

- Podgraf indukowany przez {1,2,5,6,7}
- Podgraf indukowany przez {3,4}

# Przeszukiwanie – zastosowania

**Spójność**

**We:** graf prosty G(V,E), V = {1, 2, …, n}

**Wy:** TAK gdy G jest spójny, NIE w przeciwnym przypadku

**Rozwiązanie:**

1. Inicjalizuj(G)
2. Uruchom Przeszukaj(G, s), gdzie s to dowolny element V.
3. Dla v=1,2,…n:
   - Jeśli odw[v]=0: zwróć NIE i zakończ
4. Zwróć TAK

# Przeszukiwanie – zastosowania

**Składowe spójności w grafie nieskierowanym**

**We:** graf prosty G(V,E), V = {1, 2, …, n}

**Wy:**

- odw[v]=odw[w] dla każdych v,w z tej samej składowej spójności
- odw[v]≠odw[w] dla każdych v,w z różnych składowych spójności

**Algorytm – idea:**

1. Inicjalizuj(G)
2. i ← 1
3. Dopóki są jakieś nieodwiedzone wierzchołki
   - Wybierz nieodwiedzony wierzchołek v
   - Przeszukaj podgraf wierzchołków osiągalnych z v i oznacz ich numer składowej przez i
   - i ← i + 1

# Przeszukiwanie – składowe spójności

**Algorytm Przeszukaj(G,s,i):**

1. B ← { s }
2. Dopóki B ≠∅
    1. v ← weź(B),
    2. odw[v] ← i
    3. Jeśli L[v] pusta: B ← B – { v }, wpp
        1. w ← pierwszy(L[v])
        2. L[v] ← ogon(L[v])
        3. jeśli odw[w] ≠ i:
            1. B ← B + { w }
            2. odw[w] ← i

**Algorytm Skladowe(G):**

Inicjalizuj(G)

i ← 1

Dla v = 1, 2, …, n:

- Jeśli odw[v]=0:
    - Przeszukaj(G,v,i)
    - i ← i + 1

# Przeszukiwanie wszerz (BFS)

# Przeszukiwanie wszerz (BFS)

**Def**. Dla ustalonego grafu (nieskierowanego) G(V,E) i wierzchołka s:

> $V_i$ to zbiór wierzchołków, dla których najkrótsza ścieżka do s składa się z i krawędzi

Przykłady:

- $V_0$={ s }
- $V_1$ – sąsiedzi s
- $V_2$ – sąsiedzi $V_1$, nie należący do $V_0 \cup V_1$
- itd.

# Przeszukiwanie wszerz (BFS)

**Przeszukiwanie wszerz (Breadth First Search)**

**Kolejność odwiedzania wierzchołków**

Najpierw odwiedź elementy zbioru $V_1$, potem $V_2$, następnie $V_3$, itd.

**Sposób realizacji**

Umieszczamy nowo odwiedzane wierzchołki w kolejce:

- B - zbiór wierzchołków odwiedzonych, dla których nie zbadaliśmy jeszcze wszystkich wychodzących krawędzi traktujemy jako kolejkę.

# Przeszukiwanie wszerz

**Algorytm generyczny:**

1. B ← { s }
2. Dopóki B ≠∅
    1. v ← weź(B), odw[v] =true
    2. Jeśli L[v] pusta:

        B ← B – { v }, wpp:
        1. w ← pierwszy(L[v])
        2. L[v] ← ogon(L[v])
        3. jeśli odw[w]=false:
            1. B ← B + { w }
            2. odw[w] =true

**Kolejka**:

- KolPusta() – tworzy pustą kolejkę i zwraca ją jako wynik
- WstawK(A,x) – wstawia element x na koniec kolejki A
- WeźK(A) – wynikiem element z początku kolejki
- ZdejmijK(A) – zdejmuje i zwraca jako wynik element z początku kolejki A (o ile A nie jest pusta)
- CzyPusta(A) – prawda gdy kolejka A jest pusta

# Przeszukiwanie wszerz

**Algorytm BFS(V,s):**

1. K ← KolPusta()
   WstawK(K,s)
2. Dopóki ¬CzyPusta(K)
   1. v ← WeźK(K)
   2. Jeśli L[v] pusta:
      ZdejmijK(K), wpp:
      1. w ← pierwszy(L[v])
      2. L[v] ← ogon(L[v])
      3. jeśli odw[w]=0:
         1. WstawK(K,w)
         2. odw[w] =1

**Kolejka**:

- KolPusta() – tworzy pustą kolejkę i zwraca ją jako wynik
- WstawK(A,x) – wstawia element x na koniec kolejki A
- WeźK(A) – wynikiem element z początku kolejki (zawartość kolejki bez zmian)
- ZdejmijK(A) – zdejmuje i zwraca jako wynik element z początku kolejki A (o ile A nie jest pusta)
- CzyPusta(A) – prawda gdy kolejka A jest pusta

# Przeszukiwanie wszerz - zastosowania

# Przeszukiwanie wszerz - zastosowania

**Problem najkrótszych ścieżek**

**We:** G(V,E) – graf prosty, $s \in V$

**Wy**: tablica d taka, że dla każdego $v \in V$:

- d[v] – długość najkrótszej ścieżki z s do v jeśli v osiągalny z s
- d[v] = –1 gdy v nie jest osiągalny z s.

**Algorytm – idea:**

- Ustaw d[s]=0, d[v] = – 1 dla pozostałych wierzchołków
- Uruchom BFS(G,s), z modyfikacją:

  gdy w jest odwiedzany po raz pierwszy (2.2.3) jako sąsiad v, ustaw $d[w] \leftarrow d[v]+1$

# Problem najkrótszych ścieżek

**Algorytm Najkrotsze(G,s):**

1. Dla v=1,2,…,n: d[ v ] ← - 1
2. K ← KolPusta()
   d[ s ] ← 0
   WstawK(K,s)
2. Dopóki ¬CzyPusta(K)
   1. v ← WeźK(K)
   2. Jeśli L[v] pusta: ZdejmijK(K), wpp:
      1. w ← pierwszy(L[v])
      2. L[v] ← ogon(L[v])
      3. jeśli odw[w]=0:
         1. WstawK(K,w)
         2. odw[w] ← 1
         3. d[ w ] ← d[v] + 1

# Problem najkrótszych ścieżek - poprawność

**Oznaczenia**:

- $\delta(s,v)$ to długość najkrótszej ścieżki z s do v
- $V_k = \{ v : \delta(s,v) = k \}$ – zbiór wierzchołków w odległości k od s

**Obserwacje**:

- $V_k$ to zbiór sąsiadów wierzchołków z $V_{k-1}$, które nie należą do $V_0 \cup \ldots \cup V_{k-1}$
- Każdy wierzchołek v osiągalny z s jest dokładnie raz dodawany do K, wtedy też ustalana jest wartość d[ v ]

# Problem najkrótszych ścieżek - poprawność

**Lemat** Dla każdego $i \geq 0$ zachodzi:

a) Elementy $V_i$ są umieszczone w K za elementami $(V_0 \cup V_1 \cup \ldots \cup V_{i-1})$.

b) Elementy $V_i$ są umieszczone w K przed elementami $V_{i+1}$

**Dowód**

**Indukcja** ze względu na i:

1. i=0: $V_0=\{s\}$, umieszczony jako pierwszy w K, potem jego sąsiedzi, czyli wszystkie elementu $V_1$
2. 

**Zał**.: (a) i (b) zachodzi dla $i=1,2,\ldots,k-1$.

**Teza**: (a) i (b) zachodzi dla i=k

**Dowód tezy**:

(a) Wynika z (b) dla $i=k-1$.

(b) Elementy $V_k$ to sąsiedzi elementów $V_{k-1}$, którzy nie należą do $V_0 \cup V_1 \cup \ldots \cup V_{k-1}$. Zatem elementy $V_k$ (i tylko one) zostaną dodane do kolejki wówczas, gdy na jej szczycie znajdują się elementy $V_{k-1}$.

# Problem najkrótszych ścieżek - poprawność

**Tw.** Algorytm Najkrotsze(G,s) rozwiązuje problem najkrótszych ścieżek dla grafu G i wierzchołka s.

**Dowód**

- Z lematu wynika, że $v \in V_k$ jest odwiedzany (pierwszy raz) jako sąsiad $v' \in V_{k-1}$
- Korzystając z powyższego można pokazać przez indukcję ze względu na k (długość najkrótszej ścieżki z s), że $d[v] = \delta(s,v)$ dla każdego v.

# Przeszukiwanie w głąb

# Przeszukiwanie w głąb (DFS)

**Przeszukiwanie w głąb (Depth First Search)**

**Intuicyjnie**

Idź „jak najdalej” krawędziami dopóki napotykasz nieodwiedzone wierzchołki.

**Formalnie (kolejność odwiedzania wierzchołków)**

Dla każdego v:

po odwiedzeniu v odwiedź wszystkie (nieodwiedzone wcześniej) wierzchołki osiągalne z v, potem kontynuuj przeglądanie pozostałej części grafu

**Sposób realizacji**

Umieszczamy nowo odwiedzane wierzchołki na stosie

# Przeszukiwanie w głąb

**Algorytm generyczny:**

1. B ← { s }
2. Dopóki B ≠∅
    1. v ← weź(B), odw[v] ← 1
    2. Jeśli L[v] pusta:

        B ← B – { v }, wpp:

        1. w ← pierwszy(L[v])
        2. L[v] ← ogon(L[v])
        3. jeśli odw[w]=0:
            1. B ← B + { w }
            2. odw[w] =1

**Stos**:

- StosPusty() – tworzy pusty stos i zwraca go jako wynik
- WstawS(A,x) – wstawia element x na szczyt stosu A
- WeźS(A) – wynikiem element ze szczytu stosu
- ZdejmijS(A) – zdejmuje i zwraca jako wynik element ze szczytu stosu A (o ile stos nie jest pusty)
- CzyPusty(A) – prawda gdy stos A jest pusty

# Przeszukiwanie w głąb

**Algorytm DFS(G,s):**

1. S ← StosPusty()
   WstawS(S,s)
2. Dopóki ¬CzyPusty(S)
   1. v ← WeźS(S), odw[w]← 1
   2. Jeśli L[v] pusta:
      ZdejmijS(S,v), wpp:
      1. w ← pierwszy(L[v])
      2. L[v] ← ogon(L[v])
      3. jeśli odw[w]=0:
         1. WstawS(S,w)
         2. odw[w] = 1

**Stos**:

- StosPusty() – tworzy pusty stos i zwraca go jako wynik
- WstawS(A,x) – wstawia element x na szczyt stosu A
- WeźS(A) – wynikiem element ze szczytu stosu
- ZdejmijS(A) – zdejmuje i zwraca jako wynik element ze szczytu stosu A (o ile stos nie jest pusty)
- CzyPusty(A) – prawda gdy stos A jest pusty

# Przeszukiwanie w głąb - zastosowania

# Drzewa i lasy

**Def. (**Ukorzenione) **drzewo skierowane** T(V,E) to graf skierowany spełniający warunki:

- istnieje dokładnie jeden $s \in V$ (korzeń), który nie jest końcem żadnej krawędzi;
- Każdy wierzchołek poza korzeniem jest końcem jednej krawędzi;
- Każdy wierzchołek $v \in V$ jest osiągalny z korzenia s.

**Def. Las skierowany** to graf, który składa się z pewnej liczby (wierzchołkowo) rozłącznych drzew skierowanych.

**Formalnie:** Las skierowany to graf skierowany G(V,E), w którym $V=V_1 \cup \ldots \cup V_k$ dla parami rozłącznych $V_1,\ldots,V_k$ takich, że

- podgraf G indukowany przez $V_i$ jest drzewem skierowanym dla i=1,…,k
- jeśli $(u,v) \in E$ to $u,v \in V_i$ dla jakiegoś $i \in \{1,2,\ldots,k\}$

# Las DFS

**Oznaczenia: G(V,E), V={1,2,…,n}**

Przeglądanie DFS całego grafu:

**DFS(G)**

- Inicjalizuj(G)
- Dla v=1,2,…,n:
  - jeśli odw[v]=0: DFS(G,v)

**Def.** Las DFS dla G(V,E) to graf H(V,F) taki, że

$(u,v) \in F$ wtw v odwiedzany po raz pierwszy (i dodany do stosu) jako sąsiad u w algorytmie DFS(G).

# Las DFS

**Fakt (oczywisty)** Las DFS jest lasem skierowanym.

**Obserwacja**

Jeśli H(V,F) jest lasem DFS grafu G(V,E), to $F \subseteq E$.

**Oznaczenie**

dfs(G) – las DFS grafu G

# DFS – własności

**Fakt 4**

Jeśli $(u,v) \in dfs(G)$, to v wstawiony na stos później niż u, a zdjęty wcześniej niż u).

**Fakt 5**

v osiągalny z u w dfs(G) wtw

v wstawiony na stos później niż u, a zdjęty wcześniej niż u.

**Dowód**: $\Rightarrow$ wynika z Faktu 4

$\Leftarrow$ Indukcja ze względu na liczbę wierzchołków włożonych na S pomiędzy wstawieniem u na stos i zdjęciem u ze stosu.

**Fakt 6**

v osiągalny z u w dfs(G) wtw

w G istnieje ścieżka z u do v, której żaden element nie został wstawiony na stos przed wstawieniem u.

**Dowód:** $\Rightarrow$ wynika z Faktu 4

$\Leftarrow$ Indukcja po długości najkrótszej ścieżki spełniającej podane warunki + fakt, że pierwszy element na ścieżce będzie dodany do S pomiędzy wstawieniem i usunięciem u z S + Fakt 5.

# DFS – własności

**Def.** (v,u) jest **krawędzią powracającą** grafu skierowanego G(V,E) względem lasu L(V,F) gdy $(v,u) \in E$ oraz v jest osiągalny z u w L.

**Def.** G(V,E) (skierowany) jest **acykliczny**, gdy nie ma w nim cyklu

**Twierdzenie** G jest acykliczny wtw w G nie ma krawędzi powracającej względem dfs(G).

**Dowód**.

$\Rightarrow$ Jeśli (v,u) jest krawędzią powracającą, to w G mamy cykl złożony z: ścieżki od u do v oraz krawędzi (v,u).

$\Leftarrow$ Jeśli w G jest cykl $v_1,\ldots,v_k$:

- BSO przyjmijmy: $v_1$ to pierwszy odwiedzony wierzchołek tego cyklu.
- Wówczas $v_2,\ldots,v_k$ są osiągalne z $v_1$ w DFS(G) [ Fakt 6 ]
- Więc $(v_k,v_1)$ to krawędź powracająca.

# Sortowanie topologiczne

**Problem sortowania topologicznego**

**We:** G(V,E) – graf skierowany acykliczny

**Wy:** Uporządkowany ciąg $X=x_1,\ldots,x_n$ wszystkich elementów V spełniający warunek:

Jeśli $(u,v)\in E$, to u występuje przed v w ciągu X.

**Algorytm SortTop(G)**

1. X $\leftarrow$ stos pusty
2. Inicjalizuj(G)
3. Uruchom DFS(G) z modyfikacją:
   - gdy v jest usuwany ze stosu S: wstaw v na stos X
4. Zwróć stos X jako wynik [ wypisując po kolei kolejne zdejmowane elementy ]

# Sortowanie topologiczne

**Zastosowania**

- Ustalanie kolejności prac projektowych/budowlanych, przedmiotów wybieranych w trakcie studiów, itp.

# Sortowanie topologiczne

**Twierdzenie**

Algorytm SortTop rozwiązuje problem sortowania topologicznego w czasie O(n+m).

**Dowód** Weźmy $(u,v) \in E$ dla acyklicznego G(V,E).

Wystarczy pokazać, że (∗) v jest zdjęty z S przed u. Rozważmy przypadki:

1. v osiągalny z u w dfs(G): (∗) zachodzi z Faktu 5
2. v nie jest osiągalny z u w dfs(G):
   - **Przyp. 1**: v nie został dodany do S przed u: sprz. z Fakt 6
   - **Przyp. 2**: v został dodany do S przed u:

     Jeśli v zdjęty później niż u, to u osiągalny z v (Fakt 5):

     Sprzeczność z zał., że G acykliczny.

     Jeśli v zdjęty z S wcześniej niż u: OK.

# Silnie spójne składowe

**Def.**

Graf skierowany G(V,E) jest **silnie spójny** gdy dla każdych $u,v \in V$ istnieje ścieżka prowadząca od u do v.

**Def.**

**Silnie spójną składową** grafu skierowanego G(V,E) nazywamy podgraf G'(V',E') grafu G indukowany przez $V' \subseteq V$, taki że G'(V',E') jest silnie spójny i podgraf G indukowany przez $V' \cup \{v\}$ nie jest silnie spójny dla każdego $v \in V - V'$

**Def**.

Grafem transponowanym grafu skierowanego G(V,E) nazywamy graf $G^T(V,E^T)$ taki, że $(u,v) \in E \Leftrightarrow (v,u) \in E^T$

# Silnie spójne składowe

**Problem: silnie spójne składowe**

**We:** graf skierowany G(V,E), V = {1, 2, …, n}

**Wy:**

- odw[v]=odw[w] dla każdych v,w z tej samej silnie spójnej składowej
- odw[v]≠odw[w] dla każdych v,w z różnych silnie spójnych składowych

# Silnie spójne składowe

**Intuicje – fakty:**

Oznaczenie:

dfs(G) – las przeglądania w głąb grafu G.

Fakt A. Każda silnie spójna składowa grafu G jest zawarta w pewnym drzewie T lasu dfs(G).

Fakt B. Niech v to korzeń najpóźniej utworzonego drzewa w dfs(G). Wówczas zbiór wierzchołków osiągalnych z v w grafie $G^T$ jest spójną składową grafu G.

Fakt C. Niech G(V,E) to graf skierowany. Po usunięciu z dfs(G) zbioru wierzchołków X, uzyskamy dfs(G'), gdzie G' to podgraf G uzyskany przez usunięcie z G wierzchołków ze zbioru X.

# Silnie spójne składowe

**Algorytm – szkic:**

1. Uporządkuj wierzchołki w ciąg X, odwrotny do kolejności ich zdejmowania ze stosu w przeglądaniu dfs (tj. zgodny z porządkiem otrzymanym w wyniku alg. SortTop(G)).
2. Wywołaj przeglądanie w głąb na grafie $G^T$, wybierając wierzchołki w kolejności ciągu X, tworząc las $H^T$. Jednocześnie:
   - Numeruj wierzchołki tak samo jak w algorytmie wyznaczania składowych spójności CZYLI…
   - Przyjmij, że silnie spójne składowe to drzewa lasu $H^T$.

# Silnie spójne składowe

**SilnieSpojne(G):**

1. DFS(G)
2. V' $\leftarrow$ zbiór V uporządkowany odwrotnie do kolejności zdejmowania ze stosu w trakcie DFS(G) [czyli tak jak w alg. sort. topologicznego]; niech V'=($x_1$,…,$x_n$)
3. G'(V',E') – graf transponowany grafu G (ze zmienioną kolejnością wierzchołków, patrz punkt 2)
4. Inicjalizuj(G')
5. j $\leftarrow$ 1
6. Dla i=1,2,…,n:
   - if odw[$x_i$]=0: Przegladaj(G',$x_i$,j) // j określa numer składowej
   - j $\leftarrow$ j+1