# Algorytmy i SD

# Struktury danych - Grafy

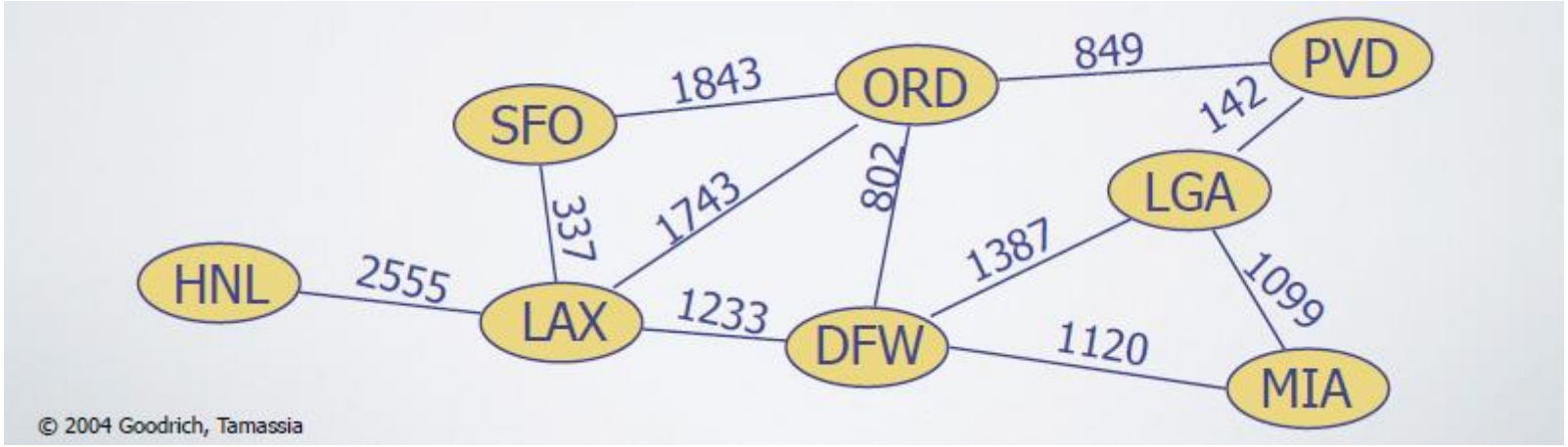
Piotr Ciskowski, Łukasz Jeleń

Wrocław, 2023

ADT **graf:**

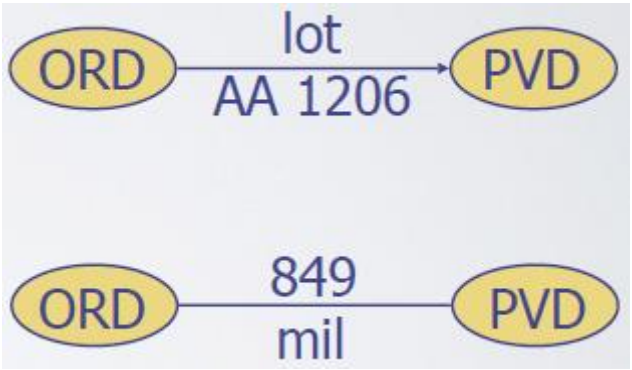
- para (V,E), gdzie:
  - V – jest zbiorem węzłów – wierzchołków
  - E – jest zbiorem par wierzchołków – krawędzi
  - wierzchołki i krawędzie są pozycjami przechowującym elementy

- przykład:
  - wierzchołki reprezentują lotniska - przechowują trzyliterowe kody
  - krawędzie reprezentują połączenia lotnicze - przechowują odległości



ADT **graf** – typy krawędzi**:**

- krawędź skierowana
  - skierowana para wierzchołków (u,v)
  - pierwszy jest początkiem
  - drugi jest końcem
  - np. lot



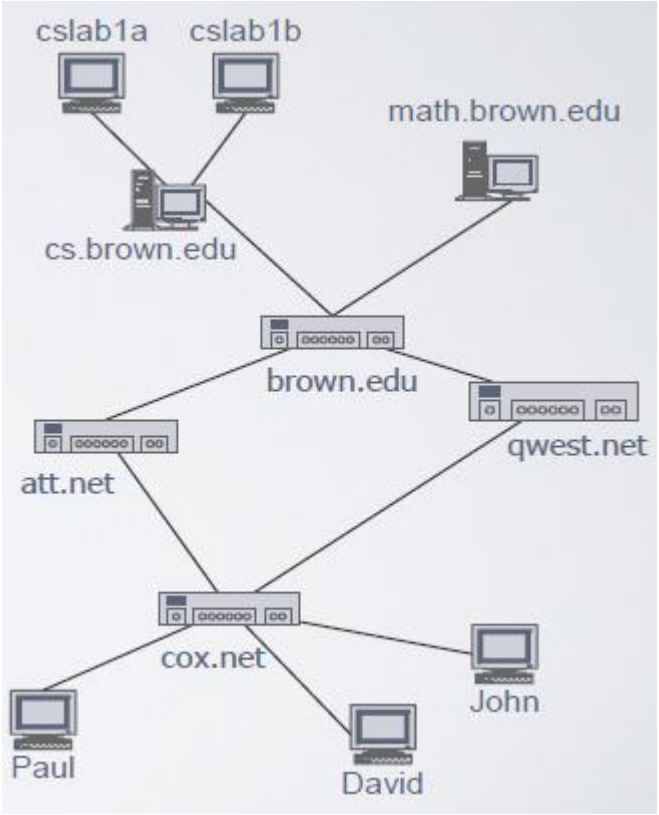

- krawędź nieskierowana
  - nieskierowana para wierzchołków (u,v)
  - kierunek nie ma znaczenia
  - np. mapa połączeń

- graf skierowany – wszystkie krawędzie są skierowane

- graf nieskierowany – wszystkie krawędzie są nieskierowane

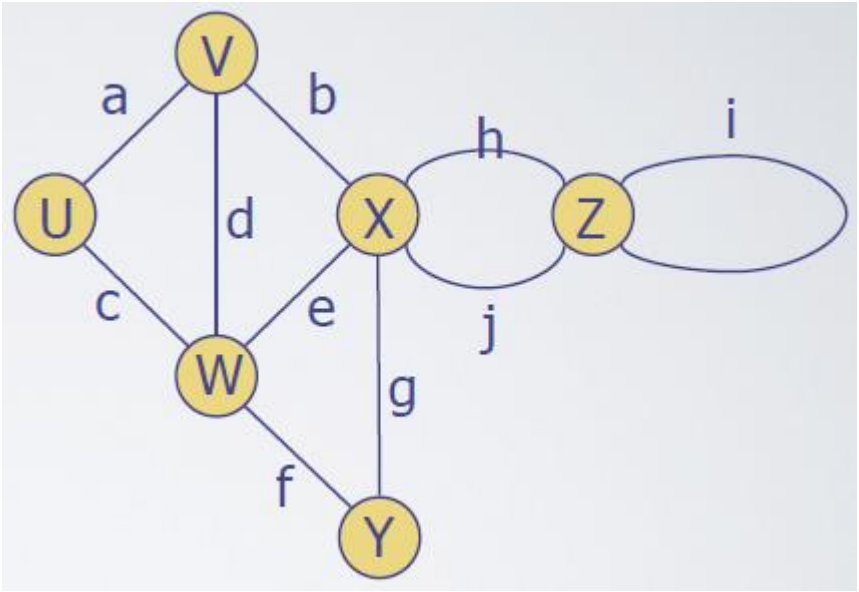
ADT **graf:**

- zastosowania:
  - obwody elektroniczne
    - płytki drukowane
  - sieci transportowe
    - sieć autostrad
    - sieć połączeń lotniczych
  - sieci komputerowe
    - LAN, internet, web
  - bazy danych
    - diagramy związków encji



ADT **graf:**

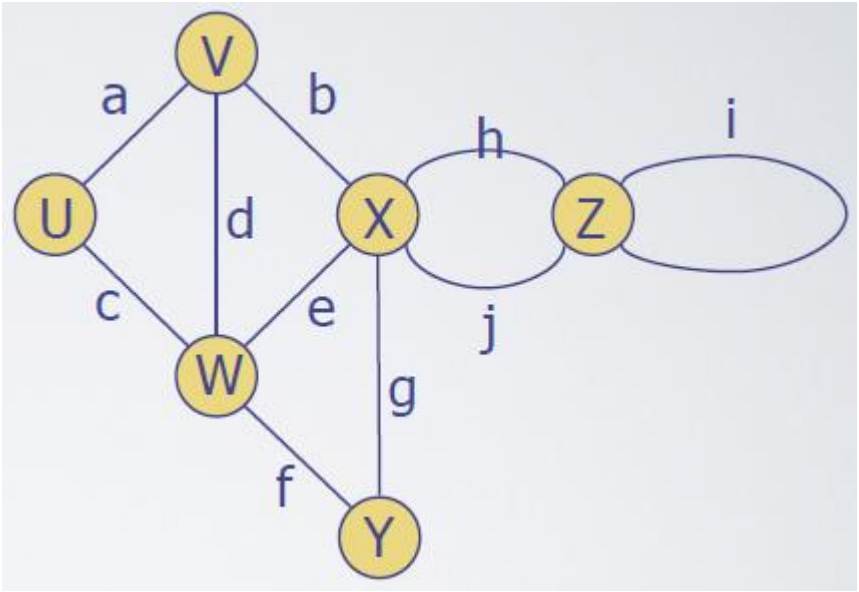
- wierzchołki końcowe krawędzi
  - U i V są końcami krawędzi a
- krawędzie incydentne do wierzchołków
  - a i b są incydentne do V
- wierzchołki sąsiednie
  - U i V są sąsiadami
- stopień wierzchołka
  - X ma stopień 5
- krawędzie równoległe
  - h i j są równoległe



ADT **graf:**

- pętla
  - i jest pętlą
- gęstość grafu
  - stosunek liczby krawędzi do max. liczby krawędzi

$$\frac{2|E|}{|V|(|V| - 1)}$$

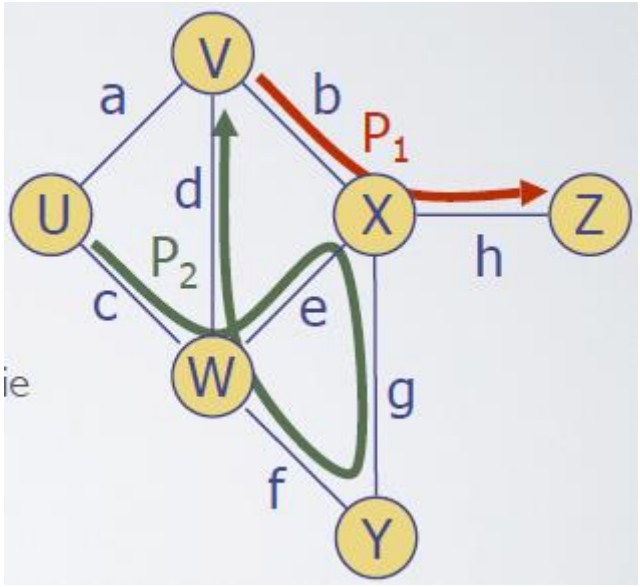


ADT **graf:**

- ścieżka
  - sekwencja kolejnych wierzchołków i krawędzi
  - rozpoczyna się od wierzchołka
  - kończy się na wierzchołku
  - każdą krawędź poprzedza i następuje jej wierzchołek końcowy

- ścieżka prosta
  - ścieżka, w której wszystkie wierzchołki i krawędzie różnią się od siebie

  - $P_1$ = (V,b,X,h,Z) – jest ścieżką prostą
  - $P_2$ = (U,c,W,e,X,g,Y,f,,W,d,V) – nie jest

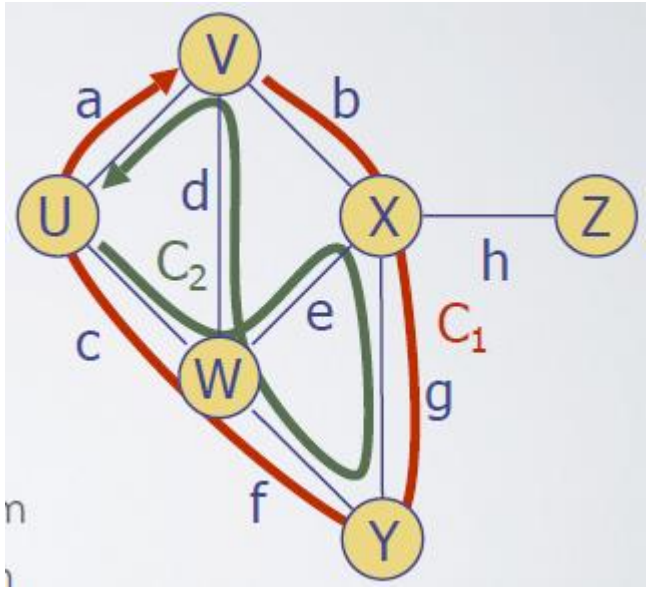


ADT **graf:**

- cykl
  - okrężna sekwencja kolejnych wierzchołków i krawędzi
  - każdą krawędź poprzedza i następuje jej wierzchołek końcowy

- cykl prosty
  - cykl, w którym wszystkie wierzchołki i krawędzie różnią się od siebie

  - $C_1$ = (V,b,X,g,Y,f,W,c,U,a,V) – jest prosty
  - C2 = (U,c,W,e,X,g,Y,f,W,d,V,a,V) – nie jest



ADT **graf** – główne metody**:**

- wierzchołki i krawędzie
  - są pozycjami
  - przechowują elementy

- metody dostępu:
  - endVertices(e) – tablica dwóch końcowych wierzchołków e
  - opposite(v,e) – przeciwległy wierzchołek do v względem e
  - areAdjacent(v,w) – prawda iff v i w są sąsiednie
  - replace (v,x) – zastąp element w wierzchołku v na x
  - replace (e,x) – zastąp element na krawędzi e na x

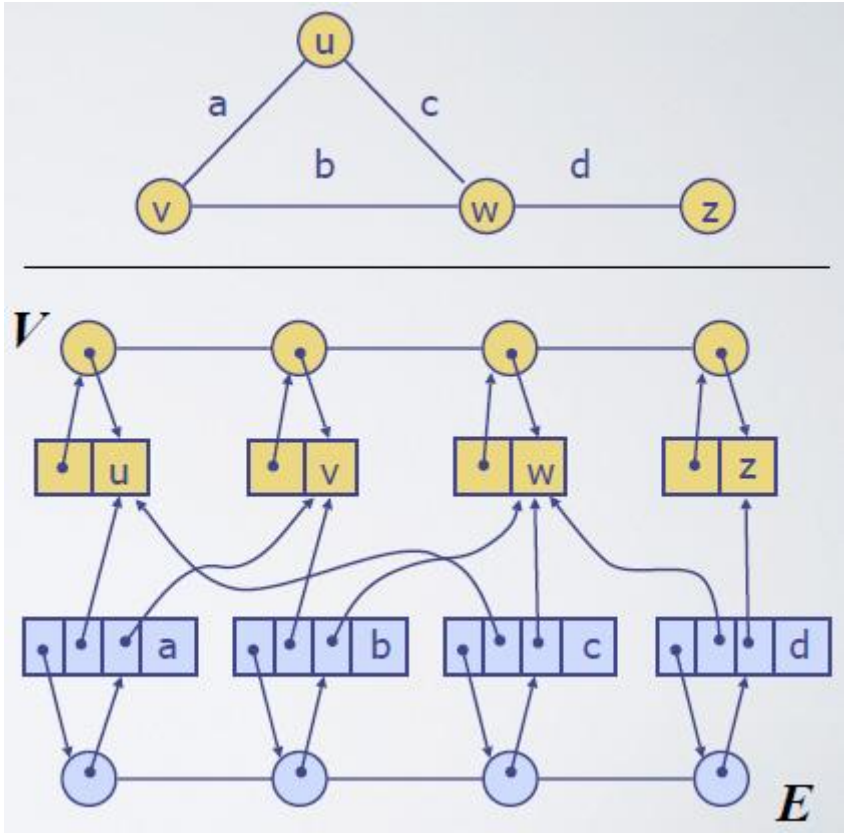
ADT **graf** – główne metody**:**

- metody uaktualniające:
  - insertVertex(o) – dodaj wierzchołek przechowujący element o
  - insertEdge(v,w,o) – dodaj krawędź (v,w) przechowującą element o
  - removeVertex(v) – usuń wierzchołek v (oraz przylegające krawędzie)
  - removeEdge€ - usuń krawędź e

- metody iterujące:
  - incidentEdges(v) – krawędzie przylegające do v
  - vertices() – wszystkie kwierzhołki w grafie
  - edges() – wszystkie krawędzie w grafie

ADT **graf** – lista krawędzi**:**

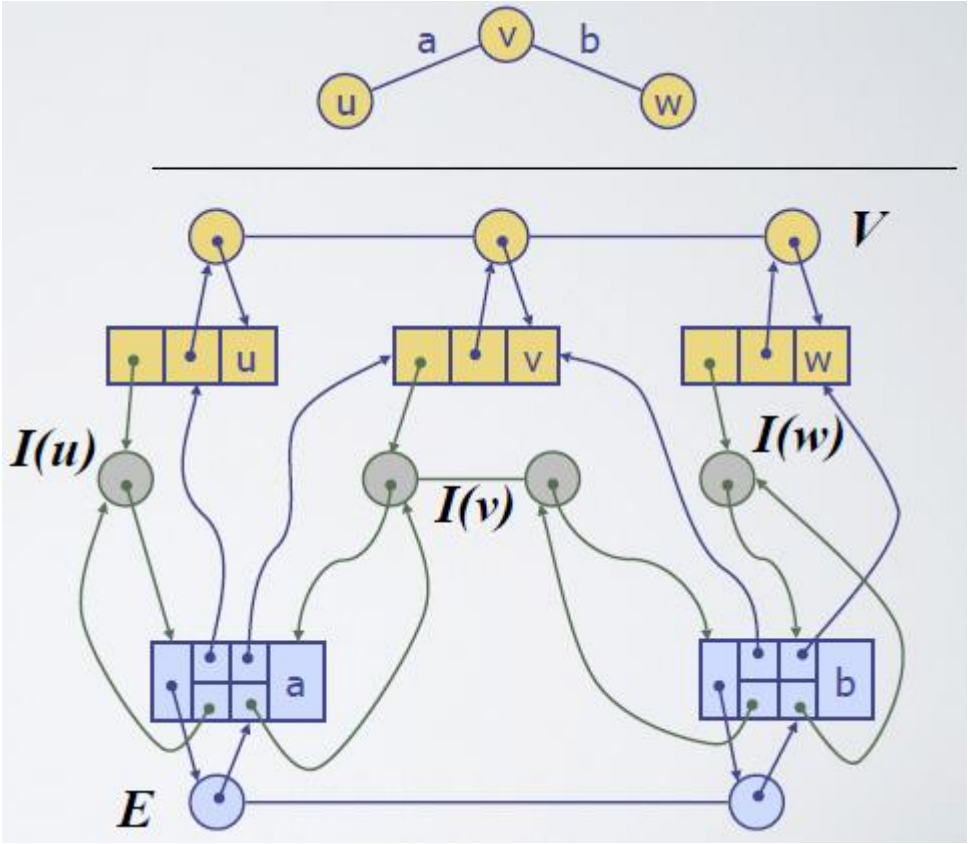
- obiekt – wierzchołek
  - element
  - referencja do pozycji w liście wierzchołków
- obiekt – krawędź
  - element
  - obiekt – wierzchołek początkowy
  - obiekt – wierzchołek końcowy
  - referencja do pozycji na liście krawędzi
- lista wierzchołków
  - sekwencja obiektów wierzchołka
- lista krawędzi
  - sekwencja obiektów krawędzi




rysunek: Łukasz Jeleń

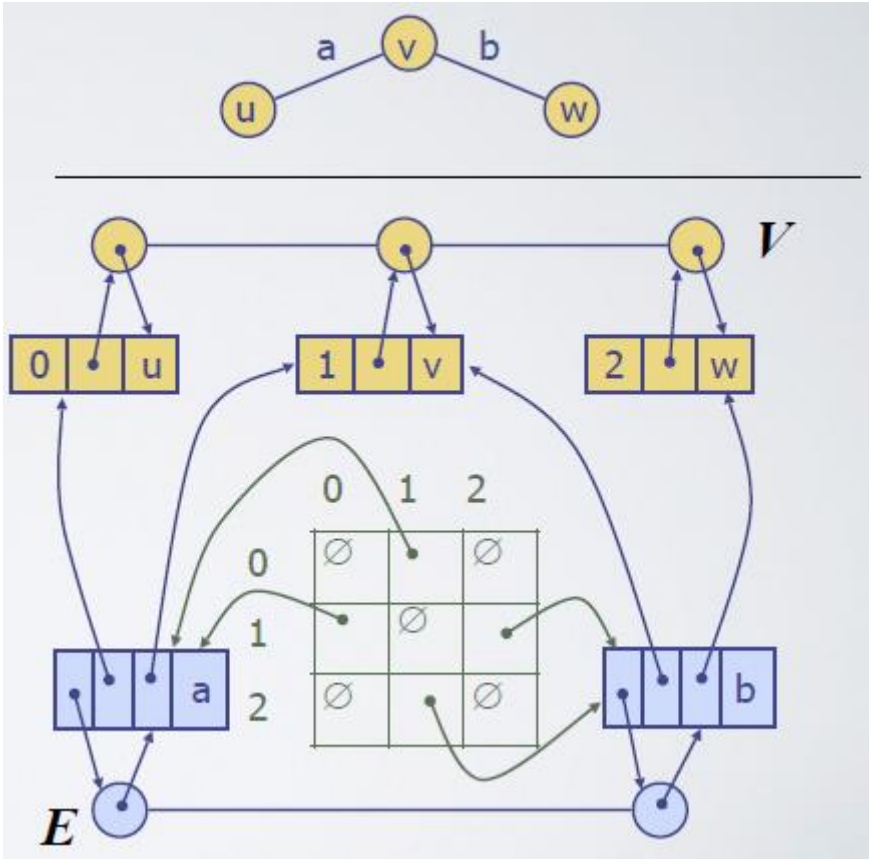
ADT **graf** – lista sąsiedztwa**:**

- struktura listy krawędzi
- lista incydencji
  dla każdego wierzchołka, I(v)
  - sekwencja referencji
    do obiektów krawędzi
    incydentnych
- rozszerzone obiekty krawędzi
  - referencje do listy sąsiedztwa
    wierzchołków końcowych



ADT **graf** – macierz sąsiedztwa**:**

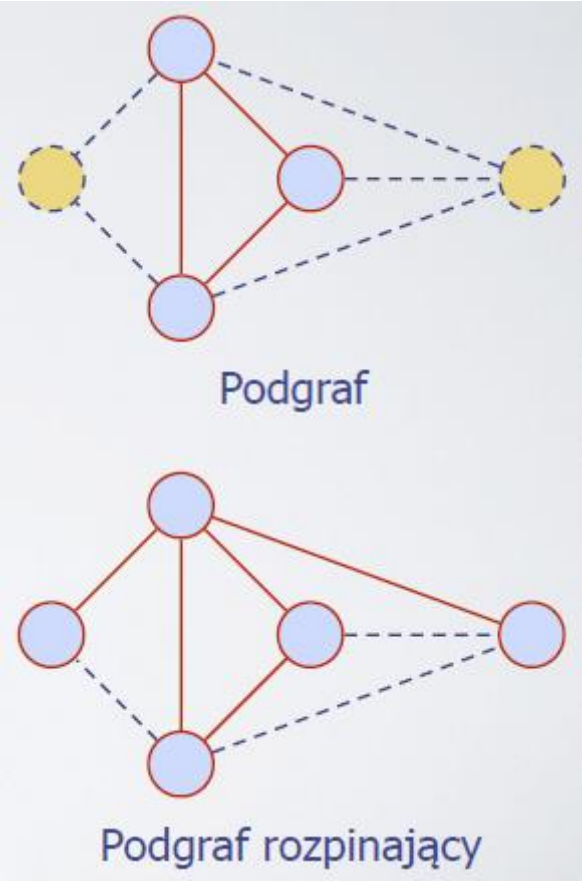
- struktura listy krawędzi
- rozszerzony
  obiekt wierzchołka
  - klucz – integer key (indeks)
    powiązany z wierzchołkiem
- tablica 2D sąsiedztwa
  - referencja
    do obiektu krawędzi
    dla sąsiednich
    wierzchołków
  - null dla
    wierzchołków
    niesąsiadujących



ADT **graf** – podgrafy**:**

- podgraf S grafu G
  jest takim grafem,
  że wierzchołki S
  są podzbiorem
  wierzchołków G

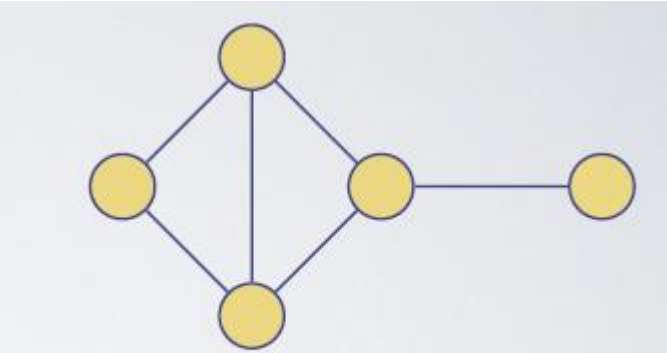
- podgraf rozpinający grafu G
  jest podgrafem,
  który zawiera
  wszystkie wierzchołki G



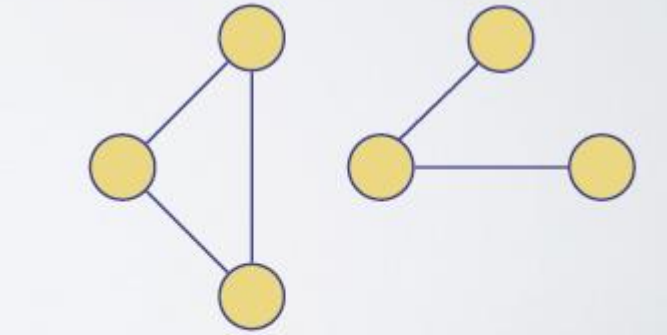
ADT **graf** – łączność**:**

- graf jest połączony (spójny)
  jeśli istnieje ścieżka
  między każdą parą wierzchołków

- elementem połączonym grafu G
  jest maksymalny
  podgraf połączony grafu G



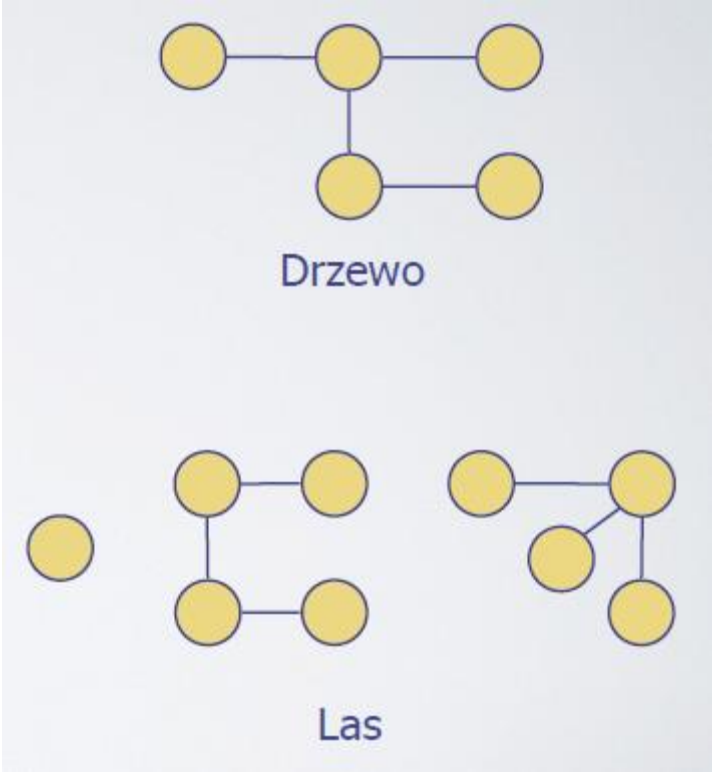
Graf połączony



Graf niepołączony (niespójny) z dwoma połączonymi elementami

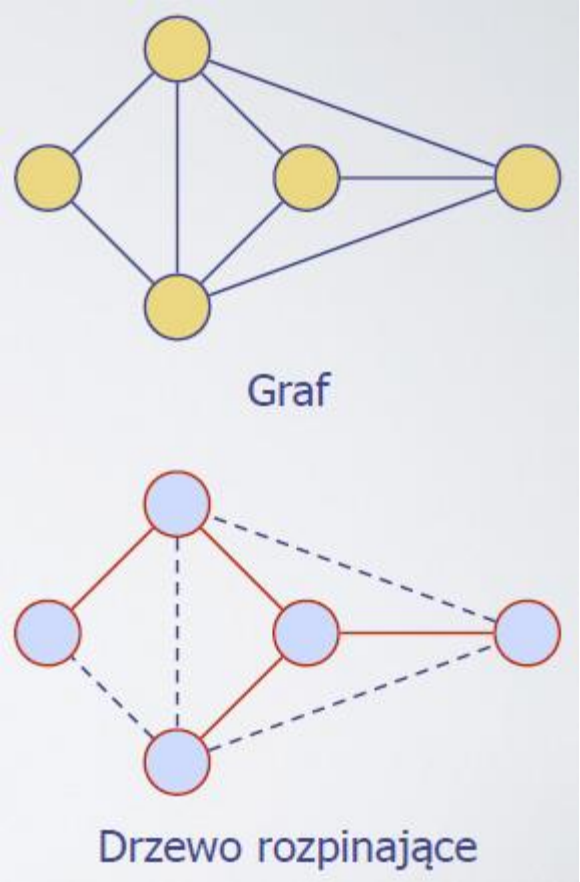
ADT **graf** – drzewa i lasy**:**

- drzewo jest grafem nieskierowanym takim, że:
  - jest połączone
  - nie zawiera cykli
- lasem jest graf nieskierowany bez cykli
- komponenty połączone lasu są drzewami



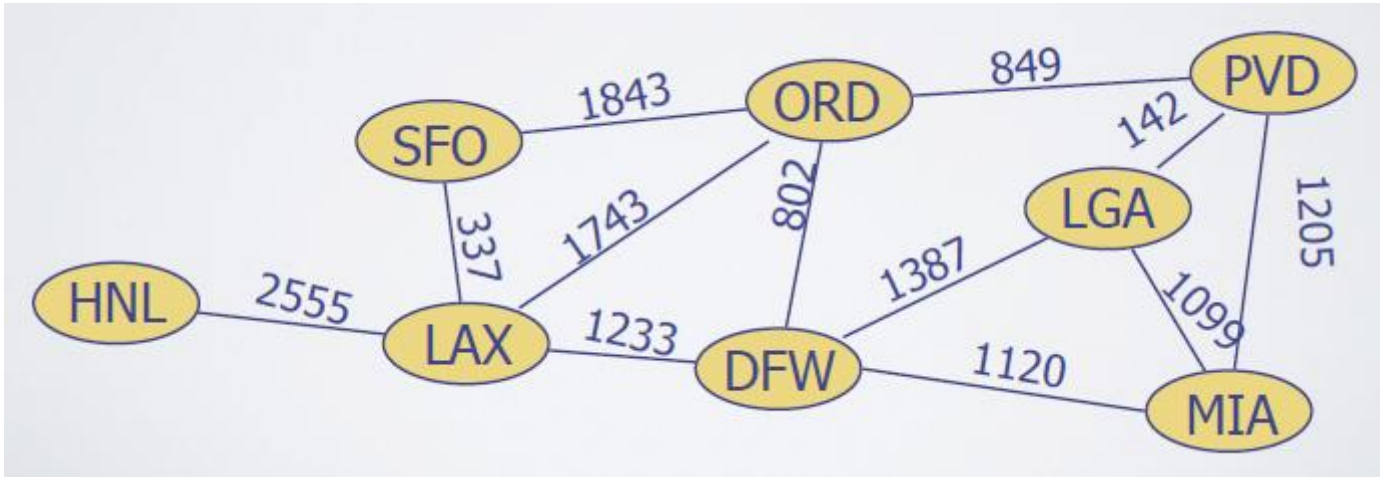
ADT **graf** – drzewa i lasy rozpinające**:**

- drzewo rozpinające grafu połączonego jest podgrafem połączonym, które jest drzewem
- drzewo rozpinające nie jest unikalne dopóki graf nie jest drzewem
- drzewa rozpinające mają zastosowanie w sieciach komunikacyjnych
- las rozpinający grafu jest podgrafem rozpinającym, który jest lasem



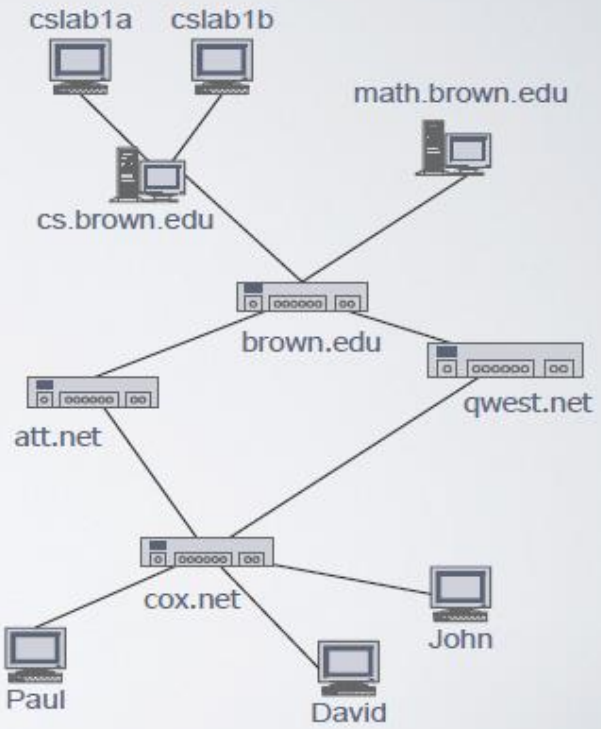
ADT **graf ważony:**

- w grafie ważonym każda krawędź ma przypisaną wartość liczbową - wagę krawędzi
- wagi krawędzi mogą reprezentować odległości, czas, koszt, itp.



# PROBLEM

- Załóżmy, że chcemy połączyć wszystkie komputery w nowo tworzonym laboratorium/biurze
  - minimalna ilość kabla - koszty
- Rozwiązanie:
  - Model grafu ważonego (G)
    - wierzchołki - komputery
    - krawędzie - wszystkie możliwe pary (u, v) komputerów
    - deg(u, v) = w(u, v) - odpowiada długości kabla potrzebnego do połączenia komputera v z komputerem u
  - Moglibyśmy wyznaczyć najkrótszą drogę od wierzchołka v
    - nieoptymalne
  - Znajdziemy drzewo T, które zawiera wszystkie wierzchołki G i posiada najmniejsze łączne wagi ze wszystkich drzew rozpinających.



# MINIMALNE DRZEWA ROZPINAJĄCE

- Mając dany nieskierowany graf G, chcemy znaleźć drzewo T, które zawiera wszystkie wierzchołki i minimalizuje sumę:

$$w(T) = \sum_{((v,u) \subseteq T)} w((v,u))$$

- Problem wyznaczania drzewa rozpinającego o najmniejszej wadze nazywa się problemem minimalnego drzewa rozpinającego (MST).

# ALGORYTM KRUSKALA

- Buduje minimalne drzewo rozpinające z zastosowaniem klastrów
  - grupowania węzłów
- Początkowo wszystkie węzły stanowią osobne klastry
- Krawędzie przechowywane w kolejce priorytetowej
  - wagi są kluczami
- Dla wszystkich krawędzi:
  - Q.removeMin();
- Jeśli u i v nie należą do tego samego klastra to dodajemy (v, u) do T
- Łączymy klastry zawierające u i v w jeden

**Algorytm** *Kruskal(G)*

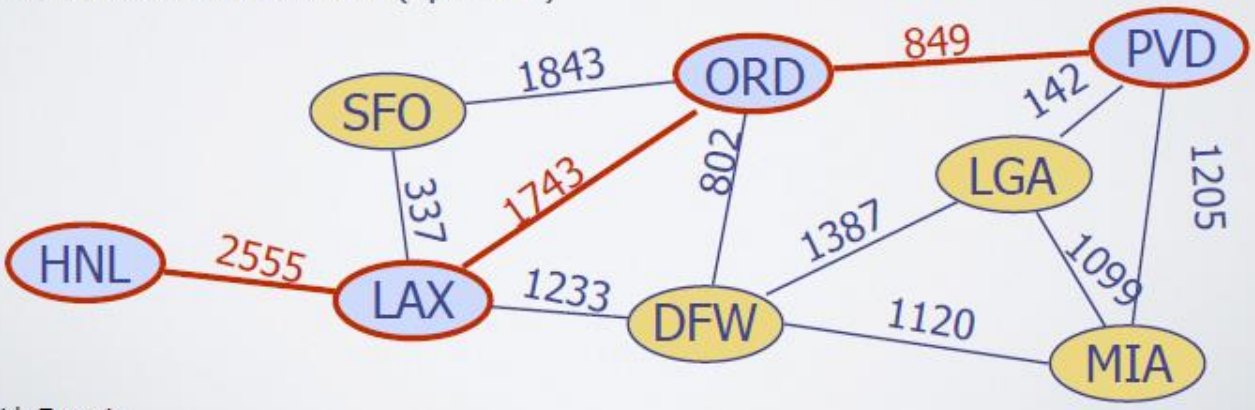
**Wejście:** Graf ważony G z n wierzchołkami i m krawędziami

**Wyjście:** Minimalne drzewo rozpinające T dla grafu G

**for** każdy wierzchołek v w G
  $C(v) \leftarrow \{k\}$
  $Q \leftarrow \{E\}$ // E - lista krawędzi
  $T \leftarrow \emptyset$
**while** T.size() < n-1 **do**
  $(u, v) \leftarrow Q.removeMin()$
  **if** C(v) ≠ C(u)
    T.Add(v, u)
    Merge(C(v), C(u))
**return** *T*

# NAJKRÓTSZA ŚCIEŻKA

- Mając dany graf ważony i dwa wierzchołki $u$ i $v$ chcemy wyznaczy ścieżkę między nimi o najmniejszej całkowitej wadze.
  - Długość ścieżki jest sumą wag jej krawędzi.
- Przykład:
  - Najkrótsza ścieżka między Providence i Honolulu
- Zastosowania
  - Przekierowywanie pakietów Internetowych
  - Rezerwacje lotów
  - Wskazówki dla kierowców (np.: GPS)



# ALGORYTM DIJKSTRY

- Odległość wierzchołka $v$ od wierzchołka $s$ jest długością najkrótszej ścieżki między $s$ and $v$
- Algorytm Dijkstra wyznacza odległości wszystkich wierzchołków począwszy od wierzchołka startowego $s$
- Założenia:
  - graf jest spójny/połączony
  - krawędzie są nieskierowane
  - wagi krawędzi są nieujemne

- Podobnie jak w przypadku algorytmu Prima będziemy tworzyć "**chmurę**" ze wszystkich wierzchołków (MST) począwszy od $s$
- Przechowujemy każdy wierzchołek $v$, etykietę $d(v)$ reprezentującą odległość $v$ od $s$ w podgrafie zawierającym MST wraz z sąsiednimi wierzchołkami
- Przy każdej iteracji
  - Dodajemy wierzchołek $u$ nieznajdujący się w MST o najmniejszej odległości, $d(u)$
  - Uaktualniamy etykiety wierzchołków sąsiednich do $u$

# ALGORYTM DIJKSTRA

- Kolejka priorytetowa przechowuje wierzchołki nieznajdujące sie w bieżącym MST
  - Klucz: odległość
  - Element: wierchołek
  - *insert*(*k*,*e*) zwraca lokalizację
  - *replaceKey*(*l*,*k*) zamienia klucz dla danego elementu
- Przechowujemy dwie etykiety dla każdego wierzchołka:
  - Odległość (d(v))
  - lokalizacje w kolejce priorytetowej

```
Algorithm DijkstraDistances(G, s)
  Q ← nowa kolejka priorytetowa bazująca
      na kopcu
  for all v ∈ G.vertices()
    if v = s
      setDistance(v, 0)
    else
      setDistance(v, ∞)
    l ← Q.insert(getDistance(v), v)
  setLocator(v,l)
  while ~Q.isEmpty()
  u ← Q.removeMin()
  for all e ∈ G.incidentEdges(u)
    {relaksacja krawędzi e }
    z ← G.opposite(u,e)
    r ← getDistance(u) + weight(e)
    if r < getDistance(z)
      setDistance(z,r)
      Q.replaceKey(getLocator(z),r)
```