

# Teoretyczne podstawy informatyki



Wykład 9:

Grafowy model danych

<http://hibiscus.if.uj.edu.pl/~erichter/Dydaktyka2009/TPI-2009>

# Graf

---

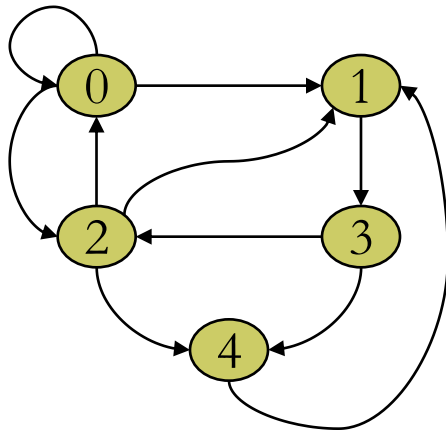
- Graf to jest relacja binarna.
- Dla grafów mamy ogromne możliwości wizualizacji jako **zbiór punktów** (zwanych **wierzchołkami**) **połączonych liniami lub strzałkami** (nazwanych **krawędziami**). Pod tym względem **graf stanowi uogólnienie drzewiastego modelu danych**.
- Podobnie jak drzewa, grafy występują w różnych postaciach: grafów **skierowanych** i **nieskierowanych** lub **etykietowanych** i **niezaetykietowanych**.
- Grafy są przydatne do analizy szerokiego zakresu problemów: obliczanie odległości, znajdowanie cykliczności w relacjach, reprezentacji struktury programów, reprezentacji relacji binarnych, reprezentacji automatów i układów elektronicznych.

# Podstawowe pojęcia

## □ Graf skierowany (ang. *directed graph*)

Składa się z następujących elementów:

- Zbioru  $V$  wierzchołków (ang. *nodes, vertices*)
- Relacji binarnej  $E$  na zbiorze  $V$ . Relacje  $E$  nazywa się zbiorem krawędzi (ang. *edges*) grafu skierowanego. Krawędzie stanowią zatem pary wierzchołków  $(u,v)$ .



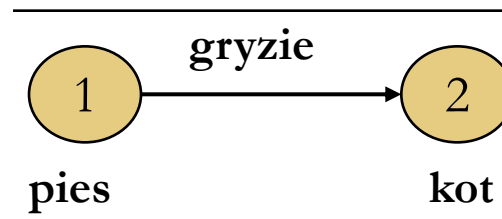
$$V = \{0,1,2,3,4\}$$

$$E = \{ (0,0), (0,1), (0,2), (1,3), (2,0), (2,1), (2,4), (3,2), (3,4), (4,1) \}$$

## Podstawowe pojęcia

### □ Etykiety:

- Podobnie jak dla drzew, dla grafów istnieje możliwość przypisania do każdego wierzchołka etykiety (ang. *label*).
- Nie należy mylić nazwy wierzchołka z jego etykietą. Nazwy wierzchołków muszą być niepowtarzalne, ale kilka wierzchołków może być oznaczonych tą samą etykietą.



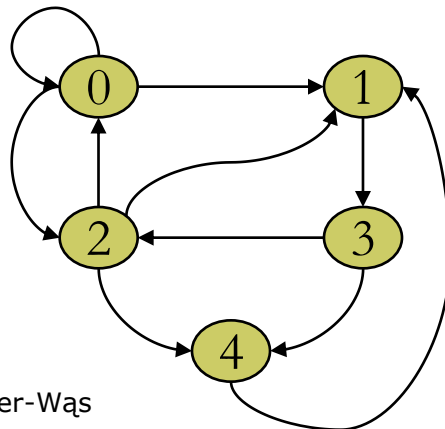
### □ Drogi:

- Droga (ang. *path*) w grafie skierowanym stanowi listę wierzchołków,  $(n_1, n_2, \dots, n_k)$  taka, że występuje krawędź łącząca każdy wierzchołek z następnym, to znaczy  $(n_i, n_{i+1}) \in E$  dla  $i=1, 2, \dots, k$ . Długość (ang. *length*) drogi wynosi  $k-1$ , co stanowi liczbę krawędzi należących do tej samej drogi.

# Podstawowe pojęcia

## □ Grafy cykliczne i acykliczne:

- **Cykl** (ang. *cycle*) w grafie skierowanym jest drogą o długości **1** lub więcej, która zaczyna się i kończy w **tym samym** wierzchołku.
- Długość cyklu jest długością drogi. Cykl jest **prosty** (ang. *simple*) jeżeli żaden wierzchołek (oprócz pierwszego) nie pojawia się na nim więcej niż raz.
- Jeżeli istnieje cykl nieprosty zawierający wierzchołek **n**, to można znaleźć prosty cykl który zawiera **n**. Jeżeli graf posiada jeden lub więcej cykli to mówimy że jest **grafem cyklicznym** (ang. *cyclic*). Jeśli cykle nie występują to, graf określa się mianem **acyklicznego** (ang. *acyclic*).



Przykłady cykli prostych:

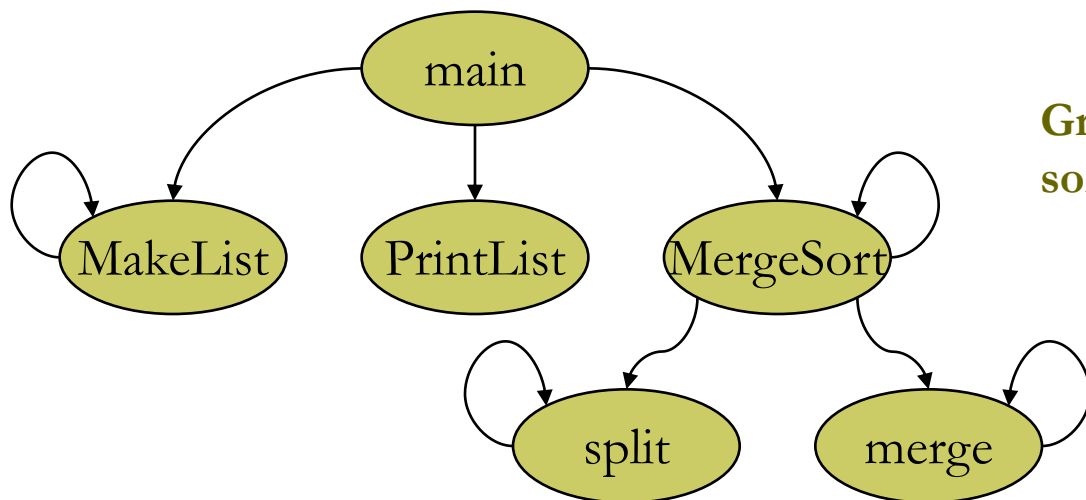
(0,0), (0,2,0), (1,3,2,1), (1,3,2,4,1)

Przykład cyklu nieprostego:

(0,2,1,3,2,0)

# Grafy wywołań

- Wywołania dokonywane przez zestaw funkcji można reprezentować za pomocą grafu skierowanego, zwanego grafem wywołań. Jego wierzchołki stanowią funkcje, a krawędź  $(P, Q)$  istnieje wówczas, gdy funkcja  $P$  wywołuje funkcję  $Q$ .
- Istnienie cyklu w grafie implikuje występowanie w algorytmie rekurencji.
- **Rekurencja** w której funkcja wywołuje samą siebie nazywamy **bezpośrednią** (ang. *direct*).
- Czasem mamy do czynienia z **rekurencją pośrednią** (ang. *indirect*) która reprezentuje cykl o długości większej niż 1, np.  $(P, Q, R, P)$ .



Graf wywołań dla algorytmu sortowania przez scalanie

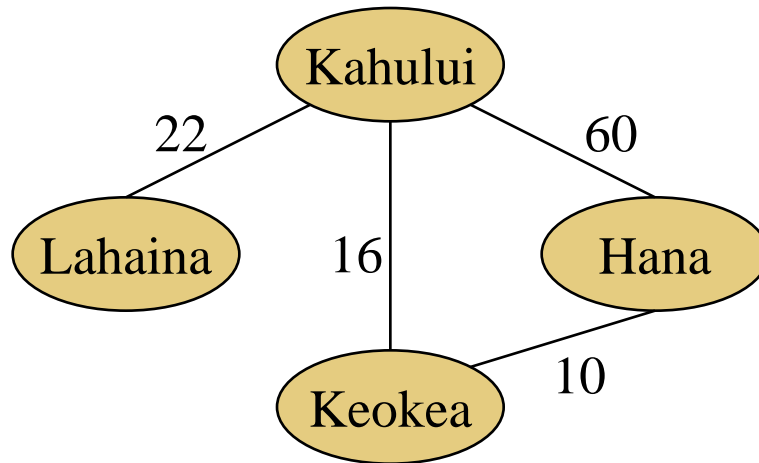
Rekurencja bezpośrednia

## Grafy nieskierowane

---

- Czasem zasadne jest połączenie wierzchołków krawędziami, które nie posiadają zaznaczonego kierunku. Z formalnego punktu widzenia taka krawędź jest zbiorem dwóch wierzchołków.
- Zapis  $\{u,v\}$  mówi że wierzchołki  $u$  oraz  $v$  są połączone w dwóch kierunkach. Jeśli  $\{u,v\}$  jest krawędzią nieskierowaną, wierzchołki  $u$  i  $v$  określa się jako **sąsiednie** (ang. *adjacent*) lub mianem **sąsiadów** (ang. *neighbors*).
- Graf zawierający krawędzie nieskierowane, czyli graf z relacją symetryczności krawędzi, nosi nazwę grafu **nieskierowanego** (ang. *undirected graph*).

## Grafy nieskierowane



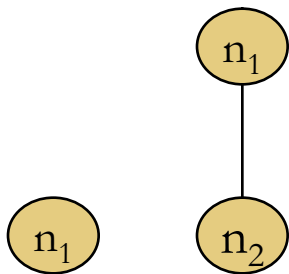
Graf nieskierowany reprezentujący drogi na wyspie Hawajów Maui.

- Droga to lista wierzchołków. Nieco trudniej jest sprecyzować co to jest cykl, tak aby nie była to każda lista  $(v_1, v_2, \dots, v_{k-1}, v_k, v_{k-1}, \dots, v_2, v_1)$

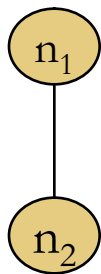


# Pewne pojęcia z teorii grafów

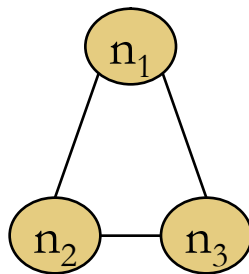
- **Teoria grafów** jest dziedziną matematyki zajmującą się właściwościami grafów.
- **Grafy pełne:**
  - Nieskierowany graf posiadający krawędzie pomiędzy każdą parą różnych wierzchołków nosi nazwę grafu pełnego (ang. *complete graph*). Graf pełny o  $n$  wierzchołkach oznacza się przez  $K_n$ .
  - Liczba krawędzi w nieskierowanym grafie  $K_n$  wynosi  $n(n-1)/2$ , w skierowanym grafie  $K_n$  wynosi  $n^2$ .



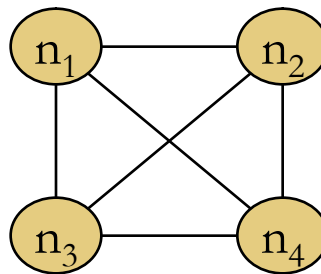
$K_1$



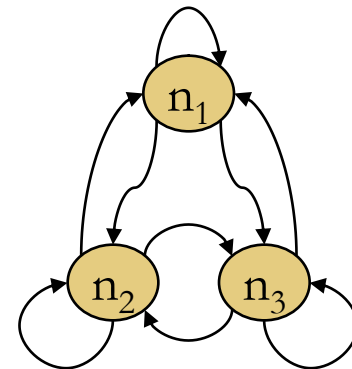
$K_2$



$K_3$



$K_4$

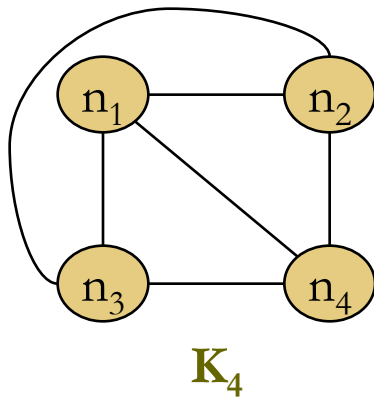


$K_3$

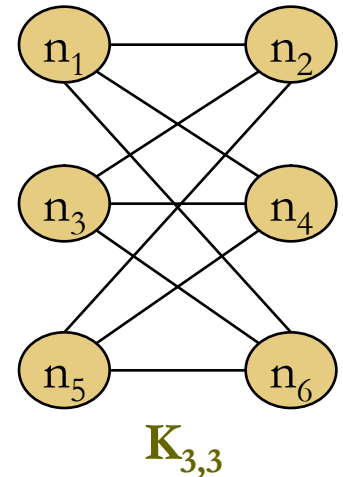
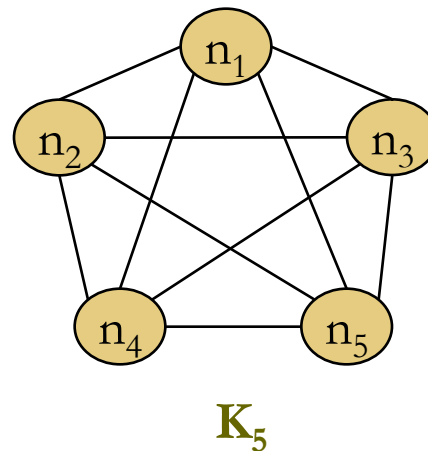
# Grafy planarne i nieplanarne

- O grafie nieskierowanym mówi się że jest **planarny** (ang. *planar*) wówczas, gdy istnieje możliwość rozmieszczenia jego wierzchołków na płaszczyźnie, a następnie narysowania jego krawędzi jako linii ciągłych które się nie przecinają.
- Grafy **nieplanarne** (ang. *nonplanar*) to takie które nie posiadają reprezentacji płaskiej.

Reprezentacja planarna:

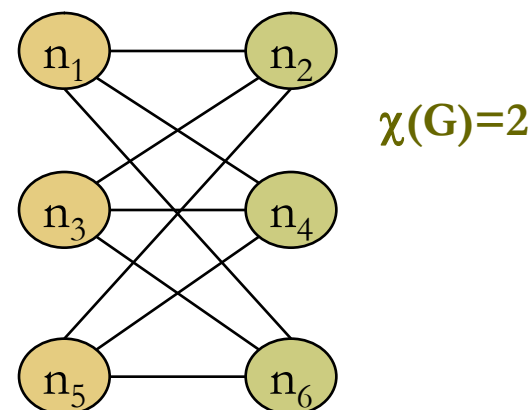
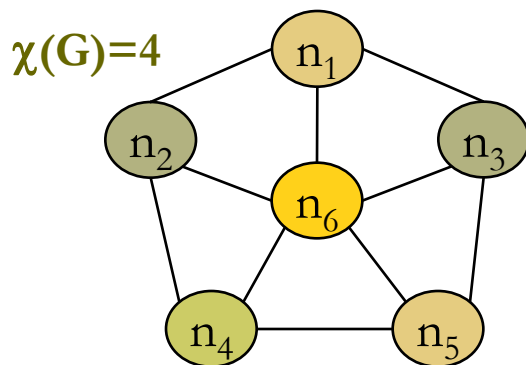


Najprostsze grafy nieplanarne:



# Zastosowania planarności i kolorowanie grafów

- Planarność ma duże zastosowanie do graficznych reprezentacji w informatyce dla projektowania różnego rodzaju układów (np. scalonych, bramek, etc.)
- **Kolorowanie grafu** (ang. *graph coloring*) polega na przypisaniu do każdego wierzchołka pewnego koloru, tak aby żadne dwa wierzchołki połączone krawędzią nie miały tego samego koloru.
- Minimalna liczba kolorów potrzebna do takiej operacji nazwana jest **liczbą chromatyczną grafu** (ang. *chromatic number*), oznaczaną  $\chi(G)$ .
  - Jeżeli graf jest pełny to jego liczba chromatyczna jest równą liczbie wierzchołków
  - Jeżeli graf możemy pokolorować przy pomocy dwóch kolorów to nazywamy go **dwudzielnym** (ang. *bipartite graph*). Np.  $K_{3,3}$ .



## Sposoby implementacji grafów

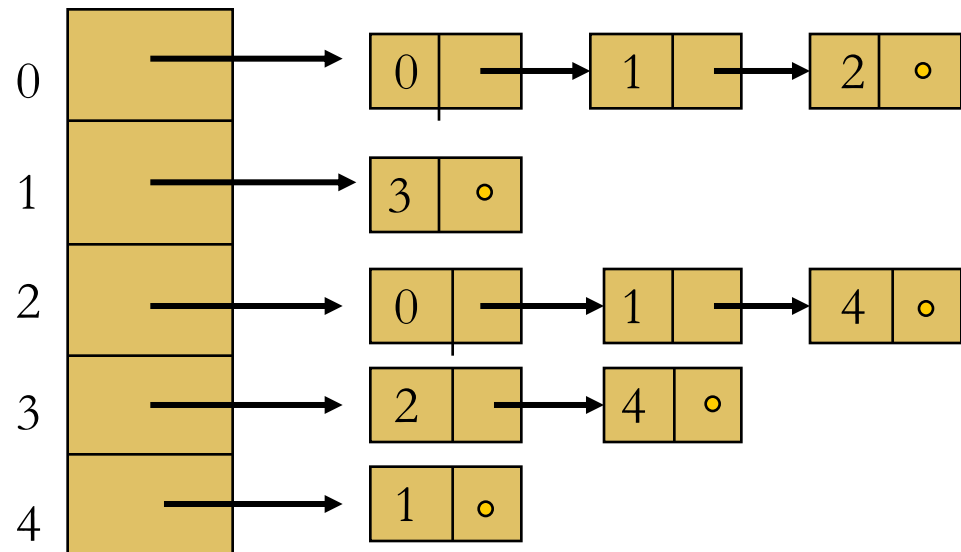
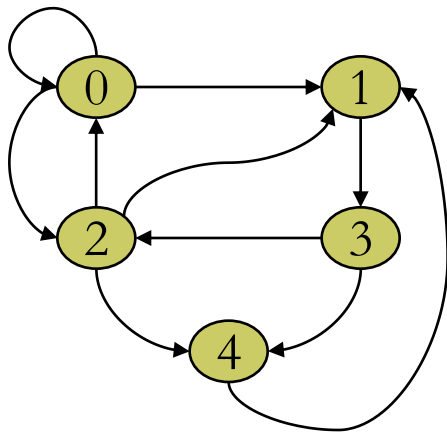
- Istnieją **dwie standardowe metody reprezentacji grafów**.
  - Pierwsza z nich, **listy sąsiedztwa** (ang. *adjacency lists*), jest, ogólnie rzecz biorąc, podobna do implementacji relacji binarnych.
  - Druga, **macierze sąsiedztwa** (ang. *adjacency matrices*), to nowy sposób reprezentowania relacji binarnych, który jest bardziej odpowiedni dla relacji, w przypadku którym liczba istniejących par stanowi znaczącą część całkowitej liczby par, jakie mogłyby teoretycznie istnieć w danej dziedzinie.
- Wierzchołki są ponumerowane kolejnymi liczbami całkowitymi **0,1,....., MAX-1** lub oznaczone za pomocą innego adekwatnego typu wyliczeniowego (używamy poniżej typu **NODE** jako synonimy typu wyliczeniowego).
- Wówczas można skorzystać z podejścia opartego na wektorze własnym.
- Element **successors[u]** zawiera wskaźnik do listy jednokierunkowej wszystkich bezpośrednich następników wierzchołka **u**. Następni mogą występować w dowolnej kolejności na liście jednokierunkowej.

### Listy sąsiedztwa:

```
typedef struct CELL *LIST;  
struct CELL {  
    NODE nodeName;  
    LIST next;  
}  
LIST successors[MAX]
```

# Reprezentacja grafu za pomocą list sąsiedztwa

- Listy sąsiedztwa zostały posortowane wg. kolejności, ale następniki mogą występować w **dowolnej kolejności** na odpowiedniej liście sąsiedztwa.

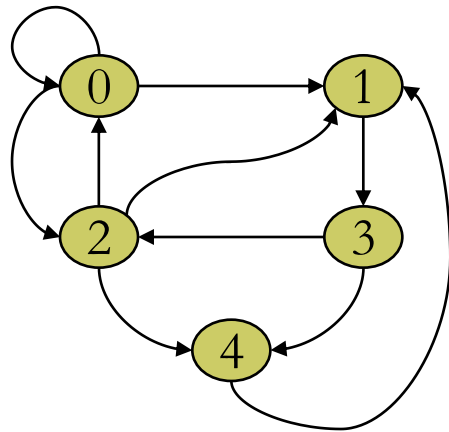


## Reprezentacja grafu za pomocą macierzy sąsiedztwa

- Tworzymy dwuwymiarową tablicę;

**BOOLEAN** `vertices`[MAX][MAX];

w której element `vertices`[*u*][*v*] ma wartość **TRUE** wówczas, gdy istnieje krawędź (*u*, *v*), zaś **FALSE**, w przeciwnym przypadku.



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

## Porównanie macierzy sąsiedztwa z listami sąsiedztwa.

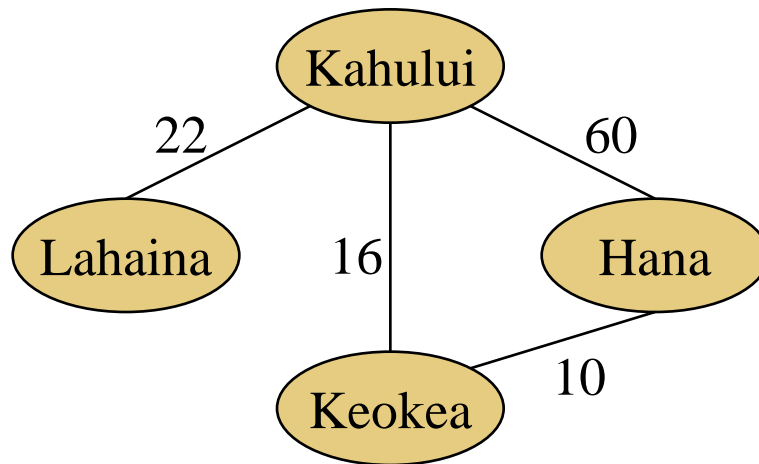
- ❑ **Macierze sąsiedztwa** są preferowanym sposobem reprezentacji grafów wówczas, gdy **grafy są gęste** (ang. *dense*), to znaczy, kiedy liczba krawędzi jest bliska maksymalnej możliwej ich liczby.
- ❑ Dla grafu skierowanego o **n** wierzchołkach maksymalna liczba krawędzi wynosi  **$n^2$** .
- ❑ Jeśli **graf jest rzadki** (ang. *sparse*) to reprezentacja oparta na listach sąsiedztwa może pozwolić zaoszczędzić pamięć.
- ❑ Istotne różnice między przedstawionymi reprezentacjami grafów są widoczne już przy wykonywaniu prostych operacji.

Preferowany sposób reprezentacji:

OPERACJA	GRAF GESTY	GRAF RZADKI
Wyszukiwanie krawędzi	Macierz sąsiedztwa	Obie
Znajdowanie następników	Obie	Lista sąsiedztwa
Znajdowanie poprzedników	Macierz sąsiedztwa	Obie

## Spójna składowa grafu nieskierowanego

- Każdy graf nieskierowany można podzielić na jedna lub większą liczbę **spójnych składowych** (ang. *connected components*).
- Każda spójna składowa to taki zbiór wierzchołków, że dla każdych dwóch z tych wierzchołków istnieje łącząca je ścieżka. Jeżeli graf składa się z jednej spójnej składowej to mówimy że jest **spójny** (ang. *connected*).



To jest graf spójny

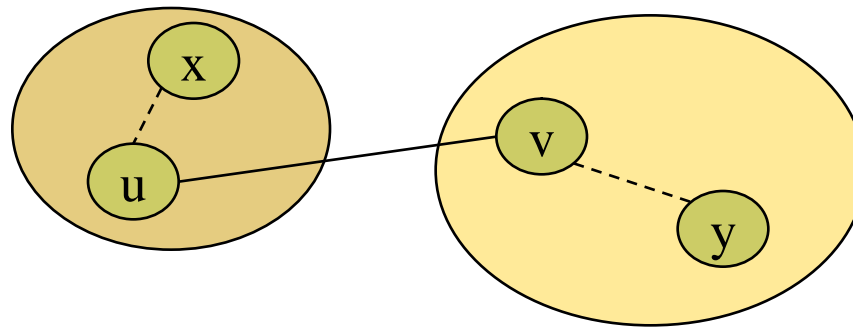


# Spójne składowe jako klasy równoważności

- Pojęcie **spójnych składowych** można potraktować formalnie jako klasy równoważności relacji równoważności **P** zdefiniowanej na wierzchołkach grafu nieskierowanego jako **uPv** wtedy i tylko wtedy, gdy istnieje droga z wierzchołka **u** do **v**.
- **Sprawdzenie czy P jest relacją równoważności**
  - Relacja **P** jest zwrotna, to znaczy zachodzi **uPu** dla dowolnego wierzchołka **u**, gdyż istnieje droga o długości **0** z dowolnego wierzchołka do niego samego.
  - Relacja **P** jest symetryczna. Jeśli zachodzi **uPv**, to istnieje droga z wierzchołka **u** do **v**. Ponieważ graf jest **nieskierowany**, odwrotny porządek wierzchołków również stanowi drogę. Stąd zachodzi **vPu**.
  - Relacja **P** jest przechodnia. Załóżmy, że relacje **uPw** oraz **wPv** są prawdziwe. Wówczas istnieje pewna droga, na przykład **(x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>j</sub>)** z **u do w**. Zatem **u=x<sub>1</sub>** oraz **w=x<sub>j</sub>**. Ponadto istnieje droga **(y<sub>1</sub>, y<sub>2</sub>, ..., y<sub>k</sub>)** z wierzchołka **w do v**, gdzie **w=y<sub>1</sub>** oraz **v=y<sub>k</sub>**. Składając obie drogi razem otrzymujemy drogę z **u do v**, czyli **(u=x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>j</sub>=w=y<sub>1</sub>, y<sub>2</sub>, ..., y<sub>k</sub>=v)**.
  - Relacja **P** dzieli graf na **klasy równoważności**. Każda klasa równoważności zdefiniowana relacją drogi odpowiada spójnej składowej tego grafu.

# Algorytm wyznaczania spójnych składowych

- Chcemy określić spójne składowe grafu  $G$ . Przeprowadzamy rozumowanie indukcyjne.
- **Podstawa:**
  - Graf  $G_0$  zawiera jedynie wierzchołki grafu  $G$  i żadnej jego krawędzi. Każdy wierzchołek stanowi odrębną spójną składową.
- **Indukcja:**
  - Zakładamy, że znamy już spójne składowe grafu  $G_i$  po rozpatrzeniu pierwszych  $i$  krawędzi, a obecnie rozpatrujemy  $(i+1)$  krawędź  $\{u, v\}$ .
    - jeżeli wierzchołki  $u, v$  należą do jednej spójnej składowej to nic się nie zmienia
    - jeżeli do dwóch różnych, to łączymy te dwie spójne składowe w jedną.



## Struktura danych dla wyznaczania spójnych składowych

---

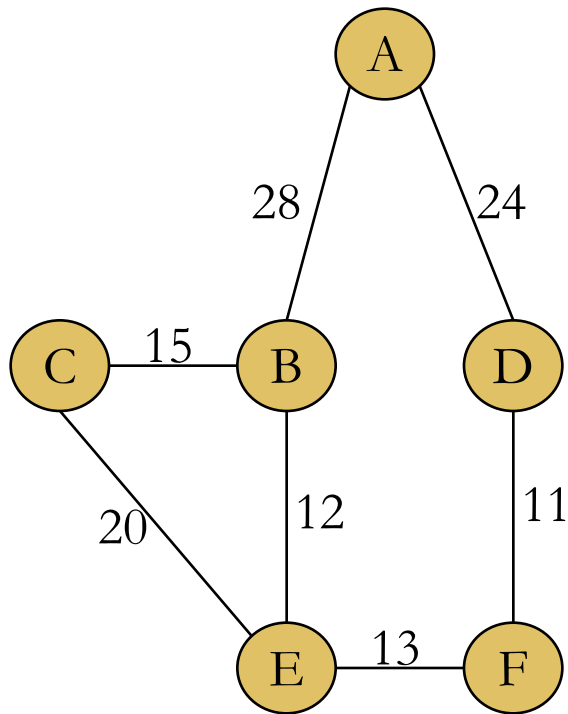
- Biorąc pod uwagę przedstawiony algorytm, musimy zapewnić szybkość wykonywalność następujących operacji:
  1. gdy jest określony wierzchołek to znajdź jego bieżącą spójną składową
  2. połącz dwie spójne składowe w jedną
- Zaskakująco dobre wyniki daje ustawienie wierzchołków każdej składowej w strukturze drzewiastej, gdzie spójna składowa jest reprezentowana przez korzeń.
  - aby wykonać operacje (1) należy przejść do korzenia
  - aby wykonać operacje (2) wystarczy korzeń jednej składowej określić jako potomka korzenia składowej drugiej.
  - Przyjmijmy zasadę że korzeń drzewa o mniejszej wysokości czynimy potomkiem.
- Przy takiej konstrukcji czas wykonania instrukcji (1) jest  $O(\log n)$ , czas wykonania instrukcji (2) jest  $O(1)$ . Wyznaczenie wszystkich spójnych składowych to  $O(m \log n)$  gdzie  $m$  jest liczbą krawędzi a  $n$  liczbą wierzchołków.

## Minimalne drzewa rozpinające

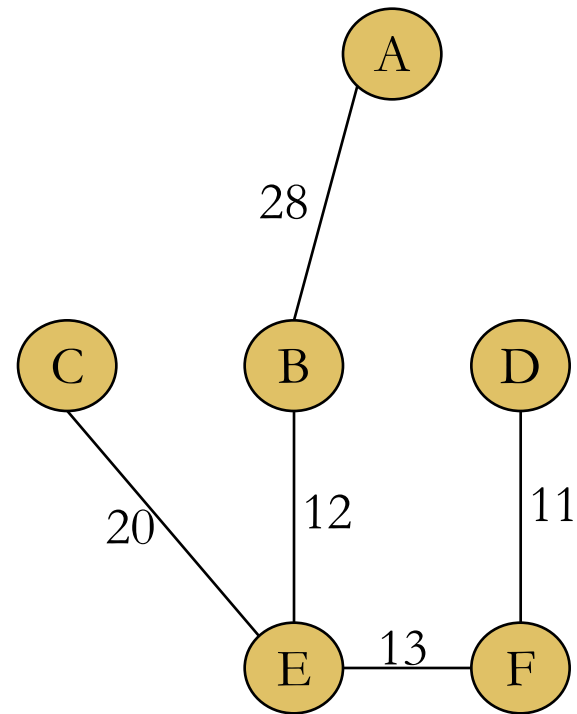
---

- **Drzewo rozpinające** (ang. *spanning tree*) grafu nieskierowanego **G** stanowi zbiór wierzchołków tego grafu wraz z podzbiorem jego krawędzi, takich że:
  - łączą one wszystkie wierzchołki, czyli istnieje droga między dwoma dowolnymi wierzchołkami która składa się tylko z krawędzi drzewa rozpinającego.
  - tworzą one drzewo nie posiadające korzenia, nieuporządkowane. Oznacza to że nie istnieją żadne (proste) cykle.
- Jeśli graf **G** stanowi pojedynczą spójną składową to drzewo rozpinające zawsze istnieje. **Minimalne drzewo rozpinające** (ang. *minimal spanning tree*) to drzewo rozpinające, w którym suma etykiet jego krawędzi jest najmniejsza ze wszystkich możliwych do utworzenia drzew rozpinających tego grafu.

# Minimalne drzewa rozpinające



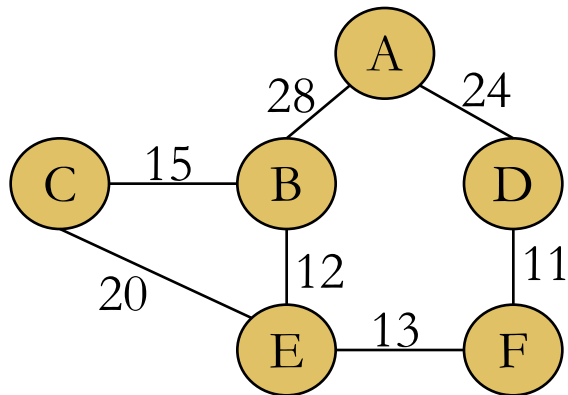
**Graf nieskierowany**



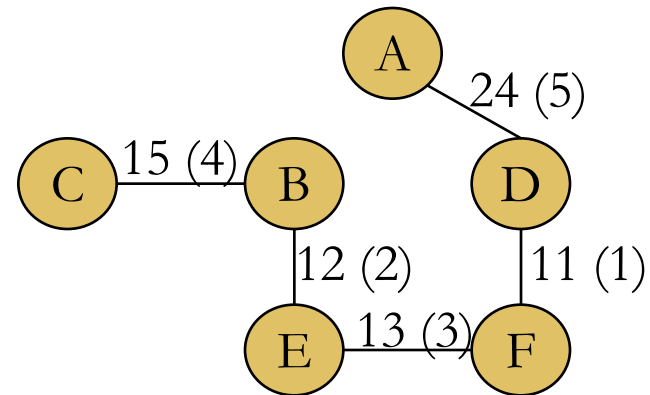
**Drzewo rozpinające**

# Znajdowanie minimalnego drzewa rozpinającego

- Istnieje wiele algorytmów. Jeden z nich to algorytm Kruskala, który stanowi proste rozszerzenie algorytmu znajdowania spójnych składowych. Wymagane zmiany to:
  - należy rozpatrywać krawędzie w kolejności zgodnej z rosnącą wartością ich etykiet,
  - należy dołączyć krawędź do drzewa rozpinającego tylko w takim wypadku gdy jej końce należą do dwóch różnych spójnych składowych.



**Graf nieskierowany**



**Minimalne drzewo rozpinające**

(w nawiasach podano kolejność dodawanych krawędzi)

## Znajdowanie minimalnego drzewa rozpinającego

---

- **Algorytm Kruskala** jest dobrym przykładem **algorytmu zachłannego** (ang. *greedy algorithm*), w przypadku którego podejmowany jest szereg decyzji, z których każdą stanowi wybranie opcji najlepszej w danym momencie. Lokalnie podejmowane decyzje polegają w tym przypadku na wyborze krawędzi dodawanej do formowanego drzewa rozpinającego.
- Za każdym razem wybierana jest krawędź o najmniejszej wartości etykiety, która nie narusza definicji drzewa rozpinającego, zabraniającej utworzenia cyklu.
- Dla algorytmu Kruskala można wykazać, że jego rezultat jest optymalny globalnie, to znaczy że daje on w wyniku drzewo rozpinające o minimalnej wadze.
- Czas wykonania algorytmu jest  **$O(m \log m)$**  gdzie  **$m$**  to jest większa z wartości liczby wierzchołków i liczby krawędzi.

## Uzasadnienie poprawności algorytmu Kruskala

- Niech  $G$  będzie nieskierowanym grafem spójnym.  
(Dla niektórych etykiet dopuszczamy dodanie nieskończenie malej wartości tak aby wszystkie etykiety były różne, graf  $G$  będzie miał wobec tego unikatowe minimalne drzewo rozpinające, które będzie jednym spośród minimalnych drzew rozpinających grafu  $G$  o oryginalnych wagach).
- Niech ciąg  $e_1, e_2, \dots, e_m$  oznacza wszystkie krawędzie grafu  $G$  w kolejności zgodnej z rosnącą wartością ich etykiet, rozpoczynając od najmniejszej.
- Niech  $K$  będzie drzewem rozpinającym grafu  $G$  o odpowiednio zmodyfikowanych etykietach, utworzonym przez zastosowanie algorytmu Kruskala, a  $T$  niech będzie unikatowym minimalnym drzewem rozpinającym grafu  $G$ .



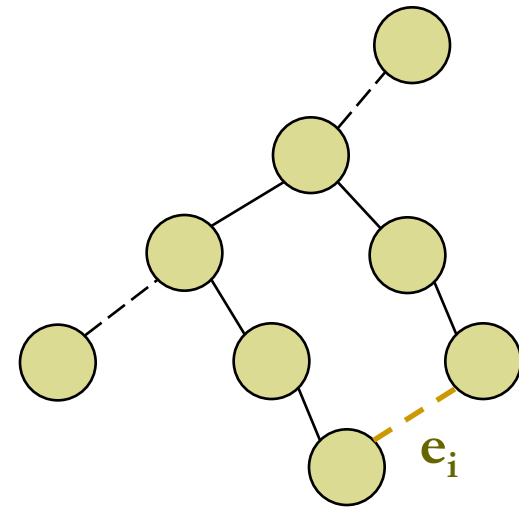
## Uzasadnienie poprawności algorytmu Kruskala

- Należy udowodnić że  $\mathbf{K}$  i  $\mathbf{T}$  stanowią to samo drzewo. Jeśli są różne musi istnieć co najmniej jedna krawędź, która należy do jednego z nich a nie należy do drugiego.
- Niech  $e_i$  oznacza pierwsza taka krawędź spośród uporządkowanych krawędzi, to znaczy każda z krawędzi  $e_1, e_2, \dots, e_{i-1}$  albo należy do obu drzew  $\mathbf{K}$  i  $\mathbf{T}$  albo nie należy do żadnego z nich.
- Istnieją dwa przypadki w zależności czy krawędź  $e_i$  należy do drzewa  $\mathbf{K}$  czy do drzewa  $\mathbf{T}$ . W każdym z tych przypadków wykażemy sprzeczność, co będzie stanowić dowód, że  $e_i$  nie może istnieć, a stąd że  $\mathbf{K}=\mathbf{T}$ , oraz że  $\mathbf{K}$  stanowi minimalne drzewo rozpinające grafu  $\mathbf{G}$ .

# Uzasadnienie poprawności algorytmu Kruskala

## □ Przypadek 1:

- krawędź  $e_i$  należy do  $\mathbf{T}$ , ale nie należy do  $\mathbf{K}$ .
- Jeżeli algorytm Kruskala odrzuca  $e_i$ , oznacza to że  $e_i$  formuje cykl z pewną drogą  $\mathbf{P}$ , utworzoną z uprzednio wybranych krawędzi drzewa  $\mathbf{K}$ .
- Jeżeli krawędzie drogi  $\mathbf{P}$  należą to  $\mathbf{K}$  to należą także do  $\mathbf{T}$ .
- A więc  $\mathbf{P} + e_i$  utworzyłoby cykl w  $\mathbf{T}$  co jest sprzeczne z definicją drzewa rozpinającego. Stąd niemożliwe jest aby  $e_i$  należała do  $\mathbf{T}$  a nie należała do  $\mathbf{K}$ .

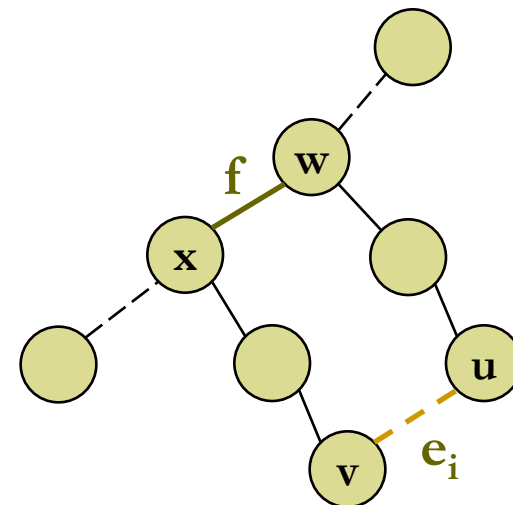


**Droga  $\mathbf{P}$  (linia ciągła) należy zarówno do drzewa  $\mathbf{T}$  jak i  $\mathbf{K}$ , krawędź  $e_i$  należy tylko do  $\mathbf{T}$ .**

# Uzasadnienie poprawności algorytmu Kruskala

## □ Przypadek 2:

- krawędź  $e_i$  należy do  $K$ , ale nie należy do  $T$ .  
Niech krawędź  $e_i$  łączy wierzchołki  $u$  i  $v$ .  
Ponieważ drzewo  $T$  jest spójne, musi istnieć w  $T$  pewna acykliczna droga z wierzchołka  $u$  do  $v$ . Niech nosi ona nazwę  $Q$ . Ponieważ w skład  $Q$  nie wchodzi  $e_i$ ,  $Q + e_i$  tworzy cykl prosty w grafie  $G$ .
  1. krawędź  $e_i$  posiada najwyższą wartość etykiety. Musiałoby to oznaczać że  $K$  zawiera cykl co jest niemożliwe.
  2. na drodze  $Q$  istnieje krawędź  $f$  która ma wartość etykiety wyższą niż  $e_i$ . Można by więc usunąć  $f$  a wprowadzić  $e_i$  nie niszcząc spójności. A więc rozpięte drzewo miałoby wartość mniejsza niż wartość dla  $T$  co jest w sprzeczności z początkowym twierdzeniem że  $T$  jest minimalne.



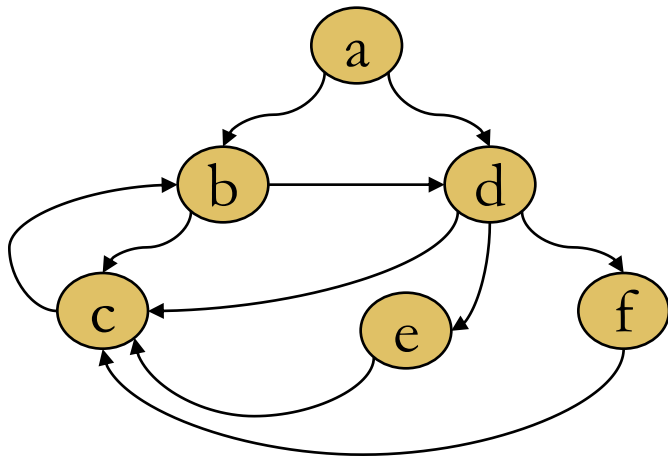
**Droga  $Q$  (linia ciągła) należy do drzewa  $T$ , można dodać krawędź  $e_i$  i usunąć krawędź  $f$**

# Algorytm przeszukiwania w głąb

---

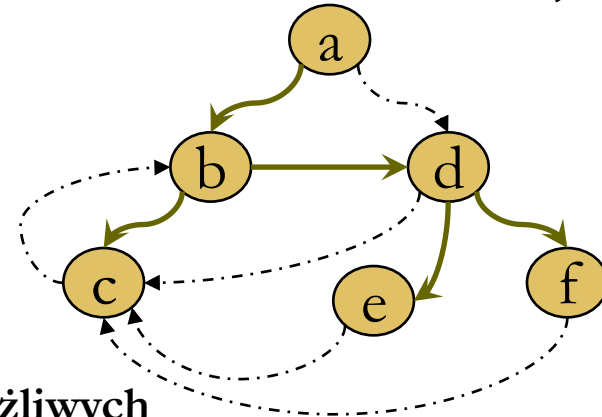
- ❑ Jest to podstawowa metoda badania grafów skierowanych.
- ❑ Bardzo podobna do stosowanych dla drzew, w których startuje się od korzenia i rekurencyjnie bada wierzchołki potomne każdego odwiedzonego wierzchołka.
- ❑ Trudność polega na tym że w grafie mogą pojawiać się cykle... Należy wobec tego znaczyć wierzchołki już odwiedzone i nie wracać więcej do takich wierzchołków.
- ❑ Z uwagi na fakt, że w celu uniknięcia dwukrotnego odwiedzenia tego samego wierzchołka jest on odpowiednio oznaczany, graf w trakcie jego badania zachowuje się podobnie do drzewa.
- ❑ W rzeczywistości można narysować drzewo, którego krawędzie rodzic-potomek będą niektórymi krawędziami przeszukiwanego grafu  $G$ .
- ❑ Takie drzewo nosi nazwę **drzewa przeszukiwania w głąb** (ang. *depth-first-search*) dla danego grafu.

# Algorytm przeszukiwania w głąb

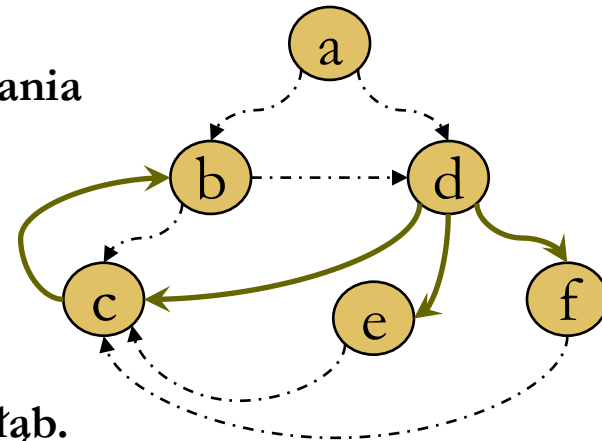


Graf skierowany

Las przeszukiwania:  
dwa drzewa o korzeniach a, d

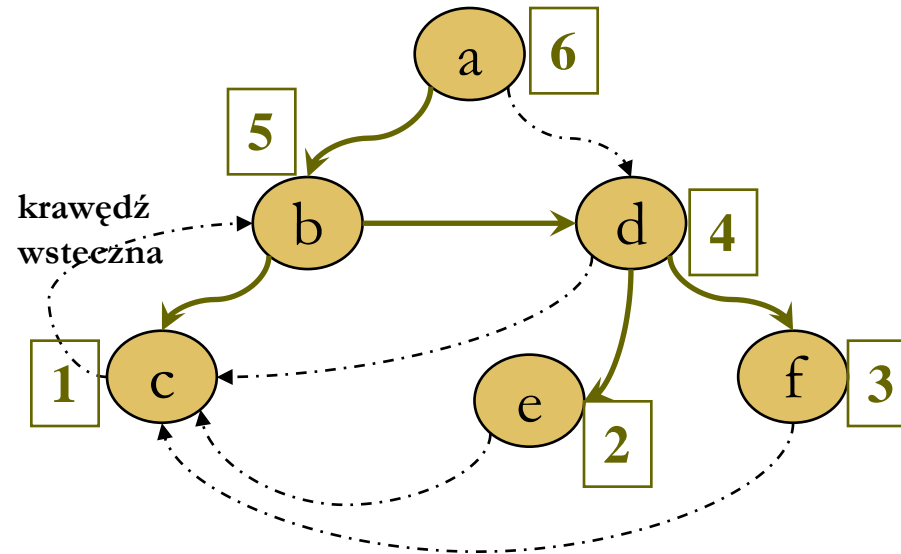


Jedno z możliwych  
drzew  
przeszukiwania



Las przeszukiwania w głąb.

## Drzewo przeszukiwania w głąb



- Po (podczas) konstruowaniu drzewa przeszukiwania w głąb można ponumerować jego wierzchołki w **kolejności wstecznej** (ang. *post-order*).

## Rekurencyjna funkcja przeszukiwania w głąb: void dfs

```
enum MARKTYPE {VISITED, UNVISITED};  
typedef struct{  
    enum MARKTYPE mark;  
    LIST successors;  
} GRAPH[MAX];
```

```
typedef struct CELL *LIST;  
struct CELL {  
    NODE nodeName;  
    LIST next;  
};
```

```
void dfs(NODE u, GRAPH G)  
{  
    LIST p; /* lista sąsiedztwa dla wierzchołka u */  
    NODE v; /* wierzchołek w komórce wskazywanej przez p */  
  
    G[u].mark = VISITED;  
    p = G[u].successors;  
    while (p != NULL) {  
        v = p->nodeName;  
        if (G[v].mark == UNVISITED) dfs(v, G);  
        p = p->next;  
    }  
}
```

## Znajdowanie cykli w grafie skierowanym

---

- Podczas przeszukiwania w głąb grafu skierowanego  $G$  można wszystkim wierzchołkom przypisać numery zgodne z kolejnością wsteczną w czasie rzędu  $O(m)$ .
- Krawędzie wsteczne to takie dla których początki są równe lub mniejsze końcom ze względu na numerację wsteczną.
- Zawsze gdy istnieje krawędź wsteczna w grafie musi istnieć cykl.
- Prawdziwe jest również twierdzenie odwrotne. Aby stwierdzić czy w grafie występuje cykl należy przeprowadzić numerację wsteczną a następnie sprawdzić wszystkie krawędzie.
- Całkowity czas wykonania testu cykliczności to  $O(m)$ , gdzie  $m$  to większa z wartości liczby wierzchołków i liczby krawędzi.



## Sortowanie topologiczne

---

- Załóżmy, że graf skierowany  $G$  jest acykliczny.
- Dla każdego grafu możemy określić las poszukiwania w głąb, określając numerację wsteczną jego wierzchołków.
- Załóżmy, że  $(n_1, n_2, \dots, n_k)$  określa listę wierzchołków grafu  $G$  w kolejności odwrotnej do numeracji wstecznej. To znaczy:  $n_1$  jest wierzchołkiem opatrzonym numerem  $n$ ,  $n_2$  wierzchołkiem opatrzonym numerem  $n-1$  i ogólnie wierzchołek  $n_i$  jest opatrzony numerem  $n-i+1$ .
- Kolejność wierzchołków na tej liście ma ta własność, że wszystkie krawędzie grafu  $G$  biegną od początku do końca, tzn. początek poprzedza koniec.

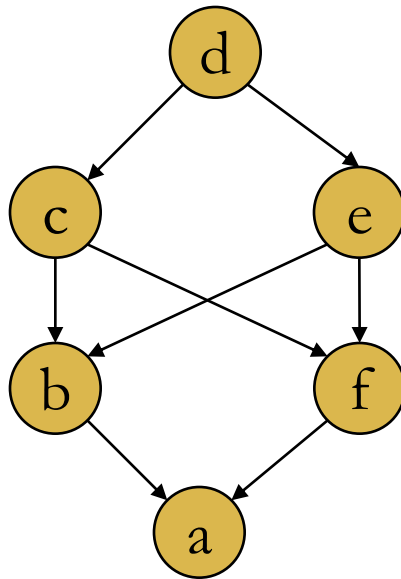
## Sortowanie topologiczne

---

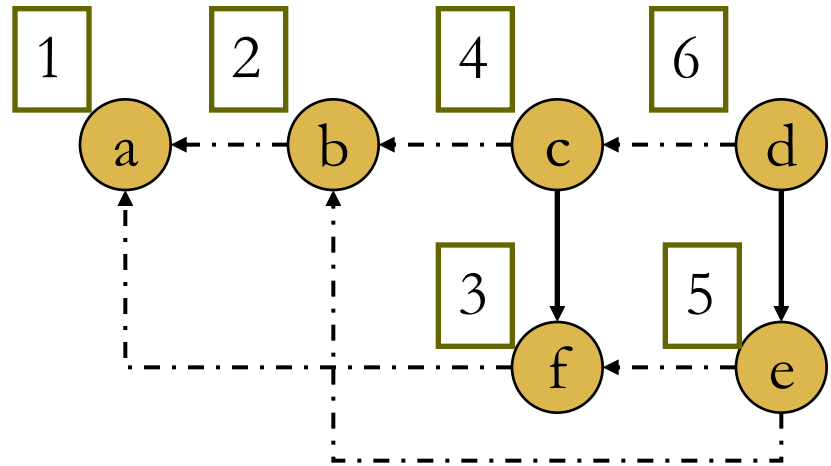
- Takie uporządkowanie nazywamy **topologicznym** (ang. *topological order*), a proces znajdowania takiego uporządkowania to **sortowanie topologiczne** (ang. *topological sorting*).
- Jedynie grafy acykliczne posiadają uporządkowanie topologiczne.
- Wykonując poszukiwanie w głąb możemy je określić w czasie  **$O(m)$** .
- Jedną z możliwości: odkładać kolejno znalezione wierzchołki „na stos”. Po zakończeniu lista znajdująca się na stosie będzie reprezentować uporządkowanie topologiczne grafu.

# Sortowanie topologiczne

Uporządkowanie topologiczne to **(d,e,c,f,b,a)**



Skierowany graf cykliczny



Las przeszukiwania w głąb

## Sortowanie topologiczne - zastosowania

---

- ❑ **Uporządkowanie topologiczne** przydaje się wówczas, gdy istnieją pewne ograniczenia odnośnie kolejności w jakiej mają być wykonywane zadania.
- ❑ Jeśli krawędź wiodącą od wierzchołka **u** do wierzchołka **v** jest rysowana wówczas, gdy zadanie **u** musi zostać wykonane przed zadaniem **v**, to uporządkowaniem zapewniającym wykonanie wszystkich zadań jest właśnie uporządkowanie topologiczne.
- ❑ Podobny przykład to graf wywołań nierekurencyjnego zbioru funkcji, kiedy należy przeanalizować każdą funkcję dopiero po dokonaniu analizy funkcji ją wywołującej.
- ❑ Jeśli krawędzie wiodą od funkcji wywołujących do wywoływanych, kolejność, w której należy przeprowadzić takie analizy, to **odwrócenie porządku topologicznego**, czyli uporządkowanie wsteczne.
- ❑ Zapewnia to że każda funkcja zostanie przeanalizowana dopiero po dokonaniu analizy wszystkich innych wywoływanych przez nią funkcji.

## Sortowanie topologiczne - zastosowania

---

- **Istnienie cyklu** w grafie reprezentującym priorytety zadań mówi o tym, że nie istnieje takie uporządkowanie, dzięki któremu możliwe byłoby wykonanie wszystkich zadań.
- **Istnienie cyklu** w grafie wywołań pozwala stwierdzić występowanie rekurencji.

# Problem osiągalności

---

- Naturalne pytanie związane z grafem skierowanym jest:
  - które wierzchołki są osiągalne z danego wierzchołka **u** przy założeniu, że po grafie można się poruszać tylko zgodnie z kierunkiem krawędzi?  
Taki zbiór wierzchołków określa się mianem **zbioru osiągalności** (ang. *reachable set*) danego wierzchołka **u**.
  
- Możemy wykorzystać rekurencyjną funkcję poszukiwania w głąb.
- Całkowity czas wykonania takiego zapytania to  **$O(mn)$** .

## Znajdowanie spójnych składowych

---

- Do znajdowania spójnych składowych możemy użyć algorytmu poszukiwania w głąb.
- Traktujemy graf nieskierowany jako graf skierowany, w którym każda krawędź nieskierowana została zastąpiona dwiema krawędziami skierowanymi wiodącymi w obu kierunkach.
- Do reprezentacji grafu używamy list sąsiedztwa.
- Tworzymy las przeszukiwania w głąb grafu skierowanego. Każde drzewo w tym lesie odpowiada jednej składowej spójności grafu nieskierowanego.
- Czas wykonania algorytmu  $O(m)$  (przy użyciu struktury drzewiastej, patrz poprzedni wykład, czas wykonania wynosi  $O(m \log n)$ ).

## Algorytm Dijkstry znajdowania najkrótszych dróg.

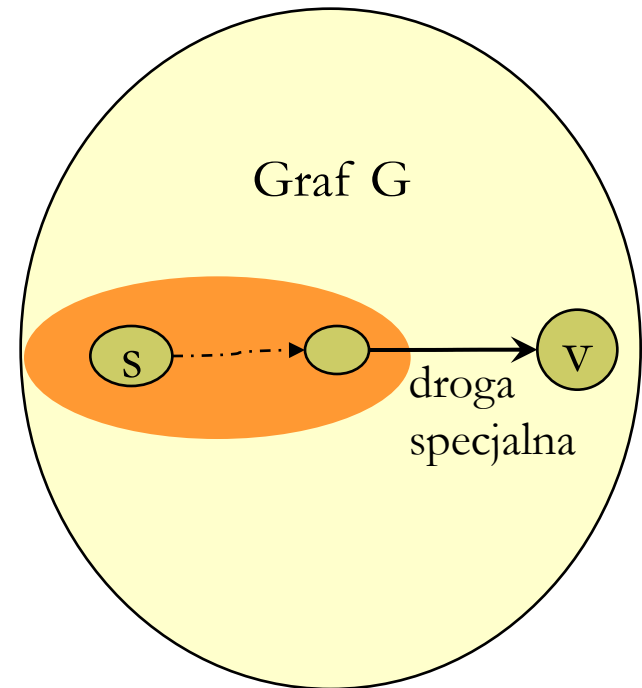
---

- Rozpatrujemy graf **G** (skierowany lub nieskierowany), w którym wszystkie krawędzie zaetykietowano wartościami reprezentującymi ich długości.
- **Długość** (ang. *distance*) danej drogi stanowi wartość sumy etykiet związanych z nią krawędzi. Minimalna odległość z wierzchołka **u** do wierzchołka **v** to minimalna długość którejś z dróg od **u** do **v**.



# Algorytm Dijkstry znajdowania najkrótszych dróg.

- ❑ Traktujemy wierzchołek **s** jako wierzchołek źródłowy. W etapie pośrednim wykonywania algorytmu w grafie **G** istnieją tzw. **wierzchołki ustalone** (ang. *settled*), tzn. takie dla których znane są odległości minimalne. W szczególności zbiór takich wierzchołków zawiera również wierzchołek **s**.
- ❑ Dla nieustalonego wierzchołka **v** należy zapamiętać długość **najkrótszej drogi specjalnej** (ang. *special path*) czyli takiej która rozpoczyna się w wierzchołku źródłowym, wiedzie przez ustalone wierzchołki, i na ostatnim etapie przechodzi z obszaru ustalonego do wierzchołka **v**.



## Algorytm Dijkstry znajdowania najkrótszych dróg.

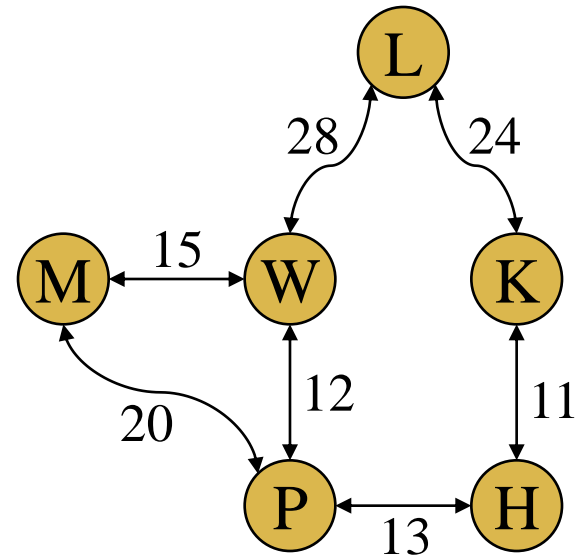
---

- Dla każdego wierzchołka  $u$  zapamiętujemy wartość **dist(u)**.
- Jeśli  $u$  jest wierzchołkiem ustalonym, to **dist(u)** jest długością najkrótszej drogi ze źródła do wierzchołka  $u$ .
- Jeśli  $u$  nie jest wierzchołkiem ustalonym, to **dist(u)** jest długością drogi specjalnej ze źródła do  $u$ .
- Na czym polega **ustalanie wierzchołków**:
  - znajdujemy wierzchołek  $v$  który jest nieustalony ale posiada najmniejszą **dist(v)** ze wszystkich wierzchołków nieustalonych
  - przyjmujemy wartość **dist(v)** za minimalną odległość z  $s$  do  $v$
  - dostosowujemy wartości wszystkich **dist(u)** dla innych wierzchołków, które nie są ustalone, wykorzystując fakt, że wierzchołek  $v$  jest już ustalony.
  - Czyli porównujemy stare **dist(u)** z wartością **dist(v)+etykieta(v, u)** jeżeli taka  $(v, u)$  krawędź istnieje.
- Czas wykonania algorytmu jest  **$O(m \log n)$** .

# Algorytm Dijkstry znajdowania najkrótszych dróg.

## Etapy wykonania algorytmu Dijkstry

	ETAPY ustalania wierzchołków				
MIASTO	(1)	(2)	(3)	(4)	(5)
H	0*	0*	0*	0*	0*
P	13	13	13*	13*	13*
M	INF	INF	33	33	33*
W	INF	INF	25	25*	25*
L	INF	35	35	35	35
K	11	11*	11*	11*	11*



## Indukcyjny dowód poprawności algorytmu

---

- W celu wykazania poprawności algorytmu Dijkstry należy przyjąć, że etykiety krawędzi są nieujemne.
- Indukcyjny dowód poprawności względem **k** prowadzi do stwierdzenia że:
  1. dla każdego wierzchołka ustalonego **u**, wartość **dist(u)** jest minimalną odległością z **s** do **u**, a najkrótsza droga do **u** składa się tylko z wierzchołków ustalonych.
  2. dla każdego nieustalonego wierzchołka **u**, wartość **dist(u)** jest minimalną długością drogi specjalnej z **s** do **u** (jeśli droga nie istnieje wartość wynosi **INF**).

# Indukcyjny dowód poprawności algorytmu

---

## □ Podstawa:

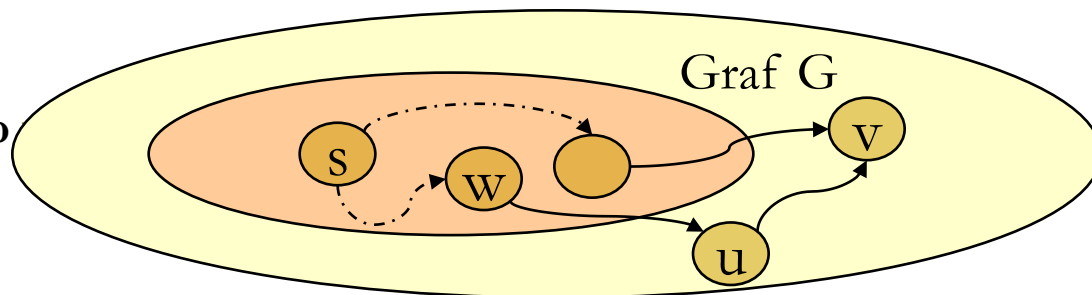
- Dla  $k=1$  wierzchołek  $s$  jest jedynym wierzchołkiem ustalonym. Inicjalizujemy  $\text{dist}(s)$  wartością  $0$ , co spełnia warunek (1).
- Dla każdego innego wierzchołka  $u$ ,  $\text{dist}(u)$  jest inicjalizowane wartością etykiety krawędzi  $(s, u)$ , o ile taka istnieje. Jeżeli nie istnieje, wartością inicjalizacji jest  $INF$ .  
Zatem spełniony jest również warunek (2).

# Indukcyjny dowód poprawności algorytmu

## □ Krok indukcyjny:

- Załóżmy, że warunki (1) i (2) są spełnione po ustaleniu  $k$  wierzchołków oraz niech  $v$  będzie  $(k+1)$  ustalonym wierzchołkiem.
- Warunek (1) jest wciąż spełniony ponieważ  $\text{dist}(v)$  jest najmniejszą długością drogi z  $s$  do  $v$ .
  - Załóżmy, że tak nie jest. Musiała by więc istnieć hipotetyczna krótsza droga do  $v$  wiodąca przez  $w$  i  $u$ . Jednakże wierzchołek  $v$  został obrany jako  $k+1$  ustalony, co oznacza, że w tym momencie  $\text{dist}(u)$  nie może być mniejsze od  $\text{dist}(v)$ , gdyż wówczas jako  $(k+1)$  wierzchołek wybrany zostałby wierzchołek  $u$ .
  - Na podstawie warunku (2) hipotezy indukcyjnej wiadomo, że  $\text{dist}(u)$  jest minimalną długością drogi specjalnej wiodącej do  $u$ . Jednak droga z  $s$  przez  $w$  do  $u$  jest drogą specjalną, tak więc jej długość równa jest co najmniej  $\text{dist}(u)$ . Stąd domniemana krótsza droga z  $s$  do  $v$  wiodąca przez  $w$  i  $u$  ma długość równą co najmniej  $\text{dist}(v)$ , ponieważ pierwsza jej część, - z  $s$  do  $u$  - ma długość  $\text{dist}(u)$ , a  $\text{dist}(u) \geq \text{dist}(v)$ . Stąd warunek (1) jest spełniony dla  $k+1$  wierzchołków.

Hipotetyczna krótsza droga do wiodąca przez  $w$  i  $u$ .

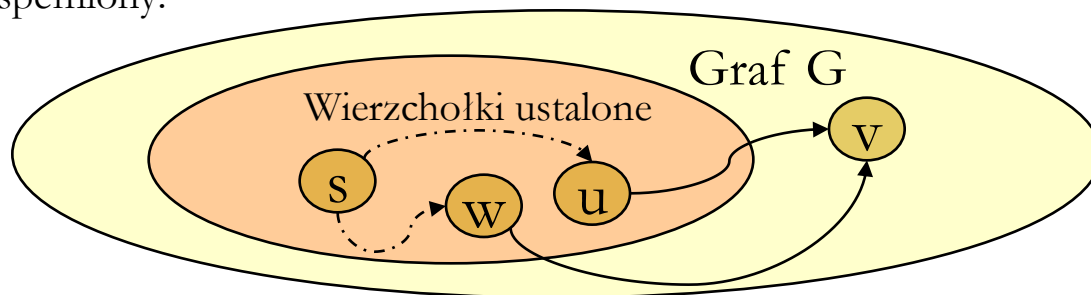


# Indukcyjny dowód poprawności algorytmu

## □ Krok indukcyjny – ciąg dalszy:

- Teraz należy pokazać, że warunek (2) jest spełniony po dodaniu do wierzchołków ustalonych wierzchołka  $v$ .
  - Weźmy pod uwagę pewien wierzchołek  $u$ , który wciąż pozostaje nieustalony po dodaniu  $v$  do wierzchołków ustalonych. W najkrótszej drodze specjalnej do  $u$  musi istnieć pewien wierzchołek przedostatni. Wierzchołkiem tym może być zarówno  $v$ , jak i pewien inny wierzchołek  $w$ .
  - Przyjmijmy, że wierzchołkiem przedostatnim jest  $v$ . Długość drogi z  $s$  przez  $v$  do  $u$  wynosi  **$\text{dist}(v) + \text{wartość etykiety } v \rightarrow u$** .
  - Przyjmijmy, że wierzchołkiem przedostatnim jest  $w$ . Na podstawie warunku (1) hipotezy indukcyjnej można stwierdzić, że najkrótsza droga z  $s$  do  $w$  składa się jedynie z wierzchołków, które zostały ustalone przed  $v$ , stąd wierzchołek  $v$  nie występuje w tej drodze.
  - A więc długość drogi specjalnej do  $u$  się nie zmienia po dodaniu  $v$  do wierzchołków ustalonych.
  - Ponieważ w momencie ustalania wierzchołka  $v$  przeprowadzona jest operacja dostosowywania  **$\text{dist}(u)$** , warunek (2) jest spełniony.

Dwie możliwości określenia przedostatniego wierzchołka w drodze specjalnej do  $u$ .



## Algorytm Floyda-Warshalla znajdowania najkrótszych dróg.

---

- Jeśli potrzebne jest poznanie minimalnych odległości między wszystkimi parami wierzchołków w grafie o  $n$  wierzchołkach, które posiadają etykiety o wartościach nieujemnych, można uruchomić **algorytm Dijkstry** dla **każdego z  $n$  wierzchołków** jako wierzchołka **źródłowego**.
- Czas wykonania algorytmu Dijkstry wynosi  $O(m \log n)$ , gdzie  $m$  oznacza większą wartość z liczby wierzchołków i liczby krawędzi. Znalezienie w ten sposób minimalnych odległości między wszystkimi parami wierzchołków zajmuje czas **rzędu  $O(m n \log n)$** .
- Jeśli  $m$  jest bliskie swojej maksymalnej wartości  $m \approx n^2$  to można skorzystać z implementacji algorytmu Dijkstry który działa w czasie  $O(n^2)$ . Wykonanie go  $n$  razy daje czas **rzędu  $O(n^3)$** .
- Istnieje inny algorytm znajdowania minimalnych odległości między wszystkimi parami wierzchołków, noszący nazwę **algorytmu Floyda-Warshalla**.
- Jego wykonanie zajmuje czas **rzędu  $O(n^3)$** . Operuje na macierzach sąsiedztwa a nie listach sąsiedztwa i jest koncepcyjnie prostszy.



## Algorytm Floyda-Warshalla znajdowania najkrótszych dróg.

- Podstawa algorytmu jest działanie polegające na rozpatrywaniu po kolei każdego wierzchołka grafu jako **elementu centralnego** (ang. *pivot*).
- Kiedy wierzchołek **u** jest elementem centralnym, staramy się wykorzystać fakt, że **u** jest wierzchołkiem pośrednim między wszystkimi parami wierzchołków.
- Dla każdej pary wierzchołków, na przykład **v** i **w**, jeśli suma etykiet krawędzi **(v, u)** oraz **(u, w)** (na rysunku **d+e**), jest mniejsza od bieżąco rozpatrywanej etykiety **f** krawędzi wiodącej od **v** do **w**, to wartość **f** jest zastępowana wartością **d+e**.

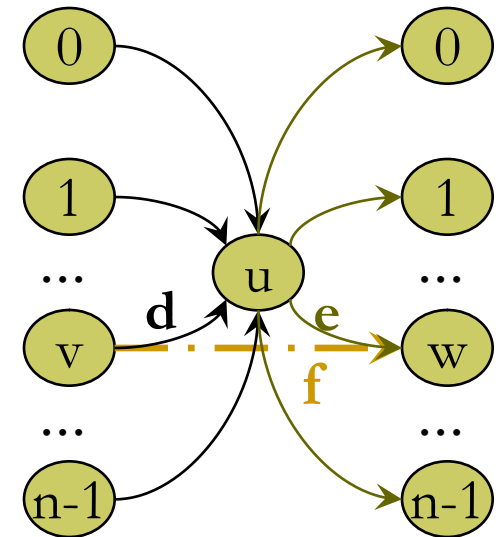
Node  $u, v, w$ ;

```

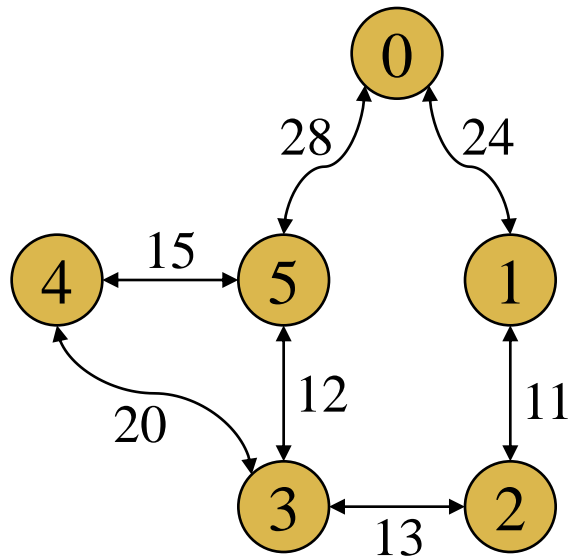
for (v = 0; w < MAX; v++)
  for (w=0; w < MAX; w++)
    dist[v][w] = edge[v][w];
for (u=0; u < MAX; u++)
  for (v=0; v < MAX; v++)
    for (w=0; w < MAX; w++)
      if( dist[v][u]+dist[u][w] < dist[v][w])
        dist[v][w] = dist [v][u] + dist [u][w];

```

$\text{edge}[v][w]$  –etykieta krawędzi, wierzchołki numerowane



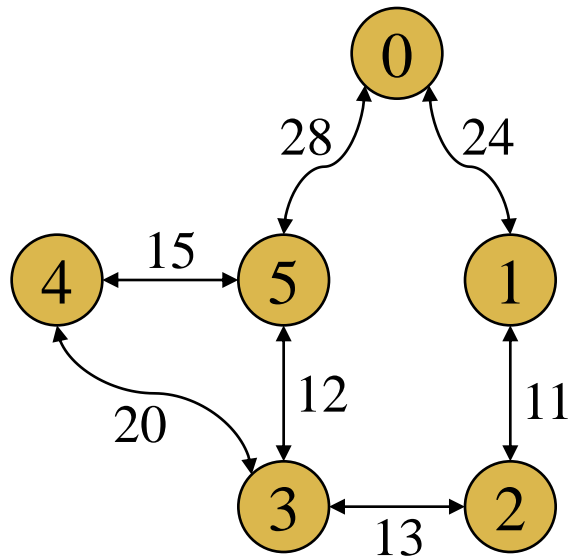
## Algorytm Floyda-Warshalla - Przykład



Macierz edge, która odzwierciedla początkową postać macierzy dist

	0	1	2	3	4	5
0	0	24	INF	INF	INF	28
1	24	0	11	INF	INF	INF
2	INF	11	0	13	INF	INF
3	INF	INF	13	0	20	12
4	INF	INF	INF	20	0	15
5	28	INF	INF	12	15	0

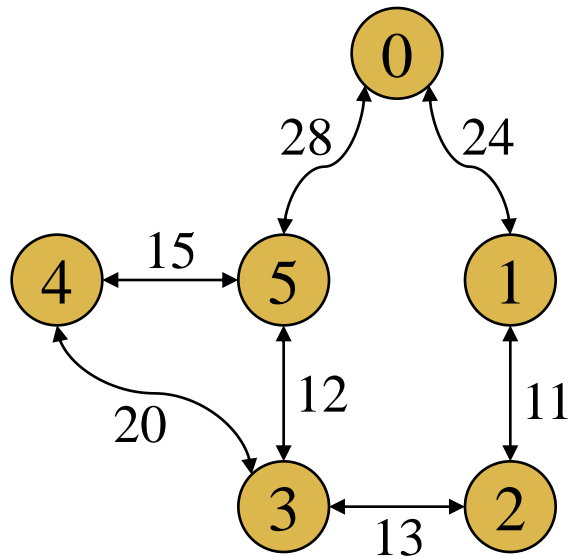
## Algorytm Floyda-Warshalla - Przykład



Macierz dist, po użyciu wierzchołka 0 jako elementu centralnego

	0	1	2	3	4	5
0	0	24	INF	INF	INF	28
1	24	0	11	INF	INF	52
2	INF	11	0	13	INF	INF
3	INF	INF	13	0	20	12
4	INF	INF	INF	20	0	15
5	28	52	INF	12	15	0

## Algorytm Floyda-Warshalla - Przykład

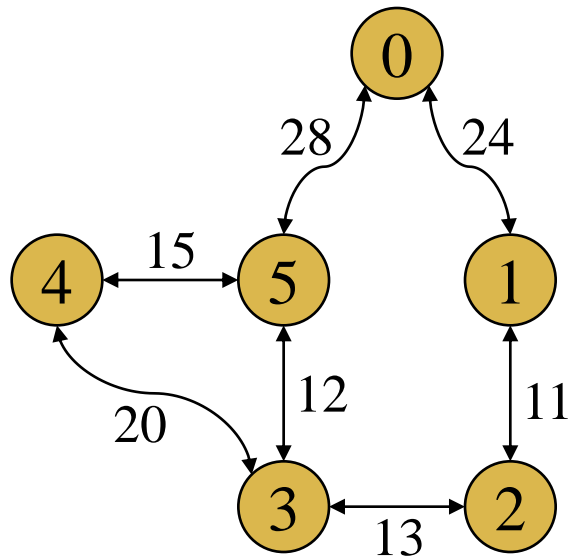


Macierz dist, po użyciu wierzchołka 1 jako elementu centralnego

	0	1	2	3	4	5
0	0	24	35	INF	INF	28
1	24	0	11	INF	INF	52
2	35	11	0	13	INF	63
3	INF	INF	13	0	20	12
4	INF	INF	INF	20	0	15
5	28	52	63	12	15	0

itd... itd...

## Algorytm Floyda-Warshalla - Przykład



Ostateczna postać macierzy dist.

	0	1	2	3	4	5
0	0	24	35	40	43	28
1	24	0	11	24	44	52
2	35	11	0	13	33	25
3	40	24	13	0	20	12
4	43	44	33	20	0	15
5	28	36	25	12	15	0

## Uzasadnienie poprawności algorytmu Floyda-Warshalla

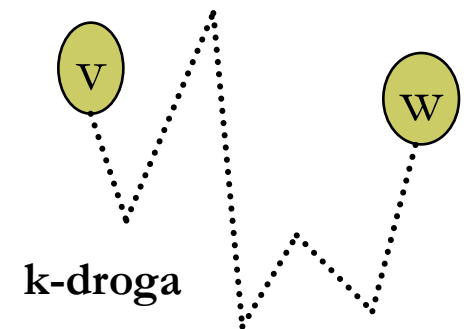
- Na dowolnym etapie działania algorytmu **Floyda-Warshalla** odległość z wierzchołka **v** do wierzchołka **w** stanowi długość najkrótszej z tych dróg, które wiodą jedynie przez wierzchołki użyte dotąd jako elementy centralne.
- Ponieważ wszystkie wierzchołki zostają w końcu użyte jako elementy centralne, elementy **dist[v][w]** zawierają po zakończeniu działań minimalne długości wszystkich możliwych dróg.
- Definiujemy **k**-drogę z wierzchołka **v** do wierzchołka **w** jako drogę z **v** do **w** taką, że żaden jej wierzchołek pośredni nie ma numeru wyższego od **k**.
- Należy zauważyć, że nie ma ograniczenia odnośnie tego, że **v** lub **w** mają mieć wartość **k** lub mniejszą.
- **k=-1** oznacza że droga nie posiada wierzchołków pośrednich.

---

numery wyższe od k

---

numery niższe od k



# Dowód indukcyjny

## □ Teza indukcyjna $S(k)$ :

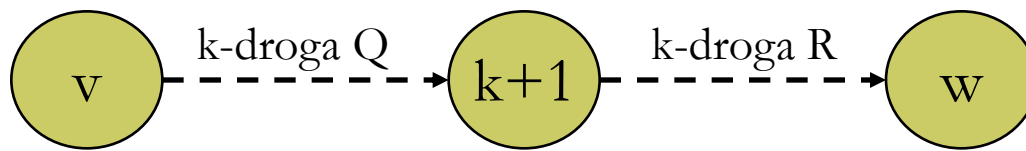
- jeżeli etykiety krawędzi mają wartości nieujemne, to po przebiegu  $k$  – pętli, element  $\text{dist}[v][w]$  ma wartość najkrótszej  $k$  – drogi z  $v$  do  $w$  lub ma wartość  $INF$ , jeżeli taka droga nie istnieje.

## □ Podstawa:

- Podstawą jest warunek  $k = -1$ . Krawędzie i drogi składające się z pojedynczego wierzchołka są jedynymi  $(-1)$  drogami.

## □ Krok indukcyjny

- Załóżmy że  $S(k)$  jest spełnione i rozważmy co się dzieje z elementami  $\text{dist}[v][w]$  w czasie  $k+1$  przebiegu pętli.
- Załóżmy, że  $P$  jest najkrótszą  $(k+1)$  – drogą wiodącą z  $v$  do  $w$ . Mamy do czynienia z dwoma przypadkami, w zależności czy droga  $P$  prowadzi przez wierzchołek  $k+1$ .



**$k$ -drogę  $P$  można rozbić na dwie  $k$ -drogi,  $Q$  oraz  $R$ .**

## Dowód indukcyjny

### □ Przypadek 1:

- Jeżeli  $P$  jest  $k$ -drogą, to znaczy, kiedy  $P$  nie wiedzie przez wierzchołek  $k+1$ , to na podstawie hipotezy indukcyjnej wartość elementu  $\text{dist}[v][w]$  jest równa długości  $P$  po zakończeniu  $k$ -tej iteracji. Nie można zmienić wartości  $\text{dist}[v][w]$  podczas przebiegu wykonywanego dla wierzchołka  $k+1$  traktowanego jako element centralny, gdyż nie istnieją żadne krótsze  $(k+1)$ -drogi.

### □ Przypadek 2:

- Jeżeli  $P$  jest  $(k+1)$ -drogą, można założyć, że  $P$  przechodzi przez wierzchołek  $k+1$  tylko raz, gdyż cykl nigdy nie może spowodować zmniejszenia odległości (przy założeniu że wszystkie etykiety mają wartości nieujemne).
- Stąd droga  $P$  składa się z  $k$ -drogi  $Q$ , wiodącej od wierzchołka  $v$  do  $k+1$ , oraz  $k$ -drogi  $R$ , wiodącej od wierzchołka  $k+1$  do  $w$ . Na podstawie hipotezy indukcyjnej wartości elementów  $\text{dist}[v][k+1]$  oraz  $\text{dist}[k+1][w]$  będą długościami dróg odpowiednio,  $Q$  i  $R$ , po zakończeniu  $k$ -tej iteracji.



## Dowód indukcyjny

---

- Ostatecznie wnioskujemy, że w  $(k+1)$  przebiegu, wartością elementu  $\text{dist}[v][w]$  staje się długość najkrótszej  $(k+1)$ -drogi dla wszystkich wierzchołków  $v$  oraz  $w$ .
- Jest to twierdzenie  $S(k+1)$ , co oznacza koniec kroku indukcyjnego.
- Weźmy teraz, że  $k=n-1$ . Oznacza to, że wiemy iż po zakończeniu wszystkich  $n$  przebiegów, wartość  $\text{dist}[v][w]$  będzie minimalną odległością dowolnej  $(n-1)$ -drogi wiodącej z wierzchołka  $v$  do  $w$ . Ponieważ każda droga jest  $(n-1)$  drogą, więc  $\text{dist}[v][w]$  jest minimalną długością drogi wiodącej z wierzchołka  $v$  do  $w$ .

# Podsumowanie informacji o algorytmach grafowych

PROBLEM	ALGORYTM(Y)	CZAS WYKONANIA
Minimalne drzewo rozpinające	Algorytm Kruskala	$O(m \log n)$
Znajdowanie cykli	Przeszukiwanie w głąb	$O(m)$
Uporządkowanie topologiczne	Przeszukiwanie w głąb	$O(m)$
Osiągalność w przypadku pojedynczego źródła	Przeszukiwanie w głąb	$O(m)$
Spójne składowe	Przeszukiwanie w głąb	$O(m)$
Najkrótsza droga dla pojedyncz. źródła	Algorytm Dijkstry	$O(m \log n)$
Najkrótsza droga dla wszystkich par	Algorytm Dijkstry	$O(m n \log n)$
	Algorytm Floyd	$O(n^3)$