GRAFY

POJĘCIA PODSTAWOWE

WĘZEŁ (WIERZCHOŁEK)

- podstawowy element składowy grafu
- w teorii grafów - punkt pewnej przestrzeni (zbioru) V, nad którą zbudowany jest graf
- reprezentuje pewien obiekt
- może posiadać nazwę (klucz) oraz dodatkowe dane

KRAWĘDŹ

- podstawowy element składowy grafu
- w teorii grafów - para wyróżnionych wierzchołków grafu, czyli takich, które są ze sobą połączone (sąsiednie)
- reprezentuje relacje między obiektami
- może być jedno- lub dwukierunkowa

GRAF (GRAF PROSTY LUB NIESKIEROWANY)

Graf nieskierowany G składa się z dwóch zbiorów – V E, przy czym V jest niepustym zbiorem, którego elementy nazywane są wierzchołkami, a E jest rodziną dwuelementowych podzbiorów zbioru wierzchołków V, zwanych krawędziami.

$$E \subseteq \{\{u,v\} : u,v \in V\}$$

GRAF SKIEROWANY

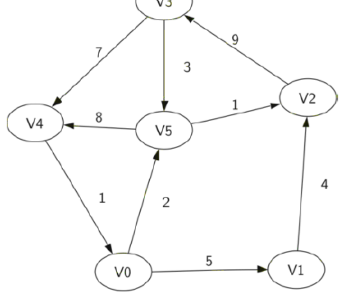
Graf skierowany (digraf) składa się z dwóch zbiorów – niepustego zbioru wierzchołków V oraz rodziny A par uporządkowanych elementów zbioru V, zwanych krawędziami lub łukami grafu skierowanego. Kolejność wierzchołków w parze wyznacza kierunek krawędzi – w przypadku pary v, u łuk biegnie z wierzchołka v do wierzchołka u.

WAGA

- wartości przypisane krawędziom
- mogą określać np.:
    - koszt przejścia między wierzchołkami
    - przepustowość połączenia
    - częstotliwość kontaktów

$$V = v0, v1, v2, v3, v4, v5$$

$$E = (v0, v1, 5), (v1, v2, 4), (v2, v3, 9), (v3, v4, 7), (v4, v0, 1), (v0, v5, 2), (v5, v4, 8), (v3, v5, 3), (v5, v2, 1)$$



ŚCIEŻKA

- sekwencja wierzchołków połączonych krawędziami

$$w_1, w_2, \ldots, w_n, (w_i, w_{i+1}) \in E, 1 \leq i \leq n-1$$

- nieważona długość ścieżki to po prostu liczba krawędzi
- ważona długość ścieżki to suma wag wszystkich krawędzi składających się na ścieżkę

DROGA

ścieżka, w której wierzchołki są różne (z wyjątkiem ewentualnej równości wierzchołków pierwszego i ostatniego – mamy wtedy do czynienia z tzw. cyklem)

CYKL

droga zamknięta, czyli taka, której koniec (ostatni wierzchołek) jest identyczny z początkiem (pierwszym wierzchołkiem)

GRAF PLANARNY

graf, którego wierzchołki można rozmieścić na płaszczyźnie w taki sposób, aby łączące je krawędzie nie przecinały się, przykład

```
import planarity
edgelist = [('a', 'b'), ('a', 'c'), ('a', 'd'),
            ('a', 'e'), ('b', 'c'),('b', 'd'),
            ('b', 'e'), ('c', 'd'), ('c', 'e'),
            ('d', 'e')]

print(planarity.is_planar(edgelist))


edgelist.remove(('a','b'))
print(planarity.is_planar(edgelist))


print(planarity.ascii(edgelist))
```

```
----1----
|  |    |
|  --5--|
|   || ||
--2--| ||
 ||  | ||
 |--3--||
 |    |||
 ---4----
```

GRAF JAKO ABSTRAKCYJNY TYP DANYCH

- Graph() - tworzy nowy pusty graf
- addVertex(vert) - dodaje węzeł do grafu
- addEdge(fromVert, toVert) - dodaje krawędź (skierowaną) do grafu
- addEdge(fromVert, toVert, weight) - dodaje krawędź ważoną
- getVertex(vertKey) - znajduje wierzchołek o podanym kluczu
- getVertices() - lista wszystkich wierzchołków w grafie
- getEdges() - lista wszystkich krawędzi w grafie
- in - sprawdza przynależność wierzchołka do grafu

REPREZENTACJE GRAFU

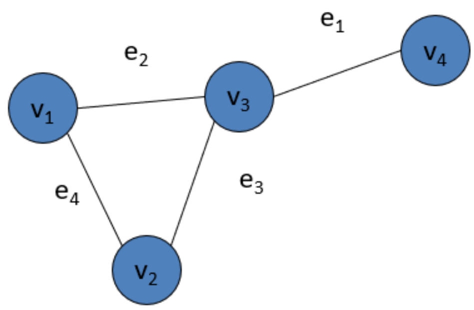
Istnieje wiele możliwych sposobów reprezentacji grafu najbardziej znane to:

- macierz sąsiedztwa
- lista sąsiedztwa
- lista krawędzi
- macierz incydencji

różnią się one między sobą zajętością pamięci oraz złożonością typowych operacji

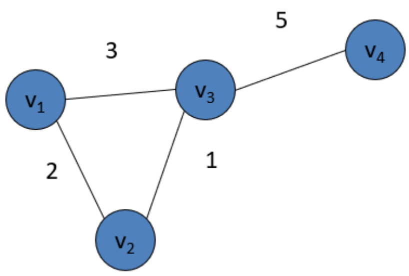
MACIERZ SĄSIEDZTWA

- implementacja w postaci macierzy n × n, gdzie n to liczba wierzchołków w grafie
- każdy wiersz i każda kolumna w macierzy odpowiada jakiemuś wierzchołkowi
- jeżeli na przecięciu wiersza v z kolumną w zapisana jest jakaś wartość (różna od zera), oznacza to, że istnieje krawędź (v, w)
- w naturalny sposób uwzględnia wagi krawędzi
- dla małych grafów duża przejrzystość - od razu widać, które wierzchołki są ze sobą połączone
- złożoność typowych operacji:
- wstawianie krawędzi: O(1)
- usuwanie krawędzi: O(1)
- sprawdzanie krawędzi: O(1)



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

1: 2→3→null
2: 1→3→null
3: 1→2→4→ null
4: 3→null



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

1: (2,2)→(3,3)→null
2: (1,2)→(3,1)→null
3: (1,3)→(2,1)→ (4,5)→ null
4: (3,5)→null

IMPLEMENTACJA GRAFU JAKO TYPU DANYCH

W Pythonie najlepszym kandydatem do zaimplementowania listy sąsiedztwa jest... słownik

Utworzono dwie klasy:

- Vertex - reprezentacja wierzchołka w grafie
- Graph - lista wierzchołków

```
class Vertex:
    def __init__(self,key):
        self.id = key
        self.connectedTo = {} #lista sąsiedztwa z wagami

    def addNeighbor(self,nbr,weight=0):
        self.connectedTo[nbr] = weight

    def __str__(self):
        return str(self.id) + ' connectedTo: ' + str([x.id for x in
self.connectedTo])

    def getConnections(self):
        return self.connectedTo.keys()

    def getId(self):
        return self.id

    def getWeight(self,nbr):
        return self.connectedTo[nbr]


class Graph:
    def __init__(self):
        self.vertList = {}
        self.numVertices = 0

    def addVertex(self,key):
        self.numVertices = self.numVertices + 1
        newVertex = Vertex(key)
        self.vertList[key] = newVertex
        return newVertex

    def getVertex(self,n):
        if n in self.vertList:
            return self.vertList[n]
        else:
            return None

    def __contains__(self,n):
        return n in self.vertList

    def addEdge(self,f,t,cost=0):
        if f not in self.vertList:
            nv = self.addVertex(f)
        if t not in self.vertList:
            nv = self.addVertex(t)
        self.vertList[f].addNeighbor(self.vertList[t], cost)

    def getVertices(self):
        return self.vertList.keys()
```

```
    def __iter__(self):
        return iter(self.vertList.values())


g = Graph()
for i in range(6):
    g.addVertex(i)
    g.vertList

g.addEdge(0,1,5)
g.addEdge(0,5,2)
g.addEdge(1,2,4)
g.addEdge(2,3,9)
g.addEdge(3,4,7)
g.addEdge(3,5,3)
g.addEdge(4,0,1)
g.addEdge(5,4,8)
g.addEdge(5,2,1)

for v in g:
    for w in v.getConnections():
        print("( %s , %s )" % (v.getId(), w.getId()))
```

WYSZUKIWANIE NAJKRÓTSZEJ ŚCIEŻKI ALGORYTM DIJKSTRY

Problem: znalezienie najkrótszej ścieżki z danego wierzchołka do wszystkich pozostałych wierzchołów w grafie spójnym z wagami.

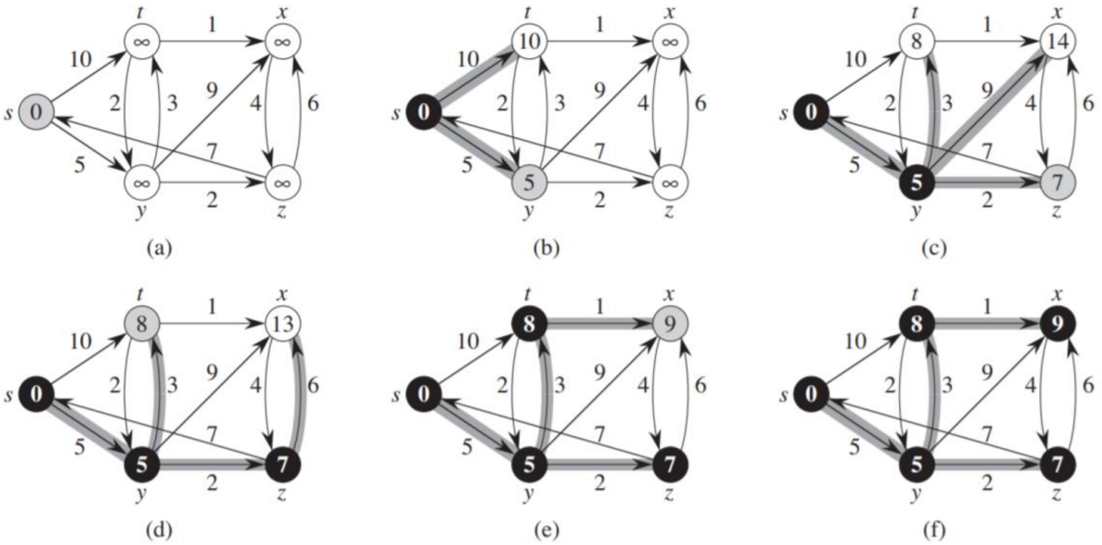
Reguła zachłanna: Spośród wszystkich wierzchołków, które mogą przedłużyć najkrótszą ścieżkę do tej pory znalezioną, wybierz ten, którego dodanie prowadzi dalej do najkrótszej ścieżki.

Wejście: G=(V,E,w) – graf spójny z wagami, s - wierzchołek startowy (źródło).

Wyjście: dla każdego wierzchołka u osiągalnego z s obliczona odległość z s do u.

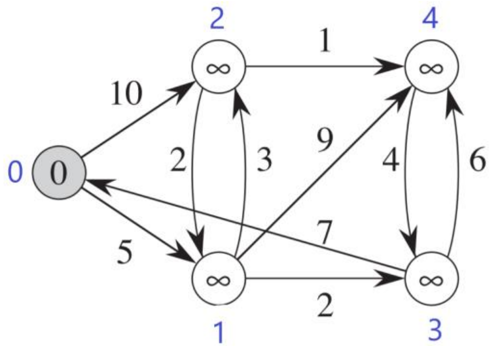
Algorytm:

```
for( każdy wierzchołek u∈V)
    ustaw odległość d[u]=∞
    p[u]=NIL
d[s]=0
utwórz kolejkę priorytetową Q (typu min) ze wszystkich wierzchołków ∈V,
w której wierzchołki są zorganizowane według wartości d
while( kolejka Q nie jest pusta)
    zwróć element u kolejki o najmniejszej wartości d oraz usuń go z
    kolejki
    for( każda krawędź (u,v)∈E)
        if( d[v]>d[u]+w(u,v) )
            d[v]=d[u]+w(u,v)
            p[v]=u
```

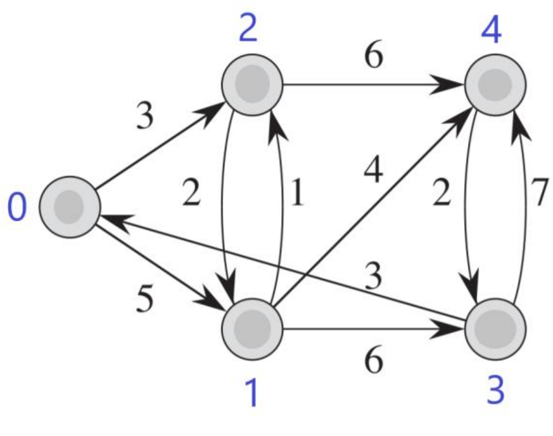


Zadania do samodzielnego rozwiązania

Zadanie 1. Wykonaj analizę działania algorytmu Dijkstry dla powyższego grafu i wierzchołka startowego s. Plik do wykorzystania Grafy_zadania.xlsx, arkusz zadanie 1



Zadanie 2. Wykonaj analizę działania algorytmu Dijkstry dla poniższej przedstawionego grafu. Najpierw dla wierzchołka startowego 0, następnie dla wierzchołka 3. (Pliki do wykorzystania: Grafy_zadania.xlsx, arkusz zadanie_2).



Zadanie 3. Zaproponuj program, który pozwoli użytkownikowi na zdefiniowanie grafu, zgodnie z poniższymi założeniami:

- Program po uruchomieniu pyta użytkownika jaki graf chce zbudować (skierowany, nieskierowany, ważony, inny możliwy).
- Użytkownik może podać ilość wierzchołków oraz połączeń pomiędzy nimi.
- Z otrzymanych informacji program wyświetla macierz sąsiedztwa oraz listę sąsiedztwa oraz wyświetla interpretację graficzną grafu.

Program powinien podejmować zrozumiałą komunikację z użytkownikiem, dane wprowadzane i wyprowadzane powinny być opatrzone zrozumiałym opisem. Program powinien być zapisany czytelnie, z zachowaniem zasad czystego formatowania kodu, należy stosować znaczące nazwy zmiennych i funkcji.

Zadanie 4. Zaproponuj implementację algorytmu wyznaczania najkrótszej drogi z wykorzystaniem algorytmu Dijkstry.