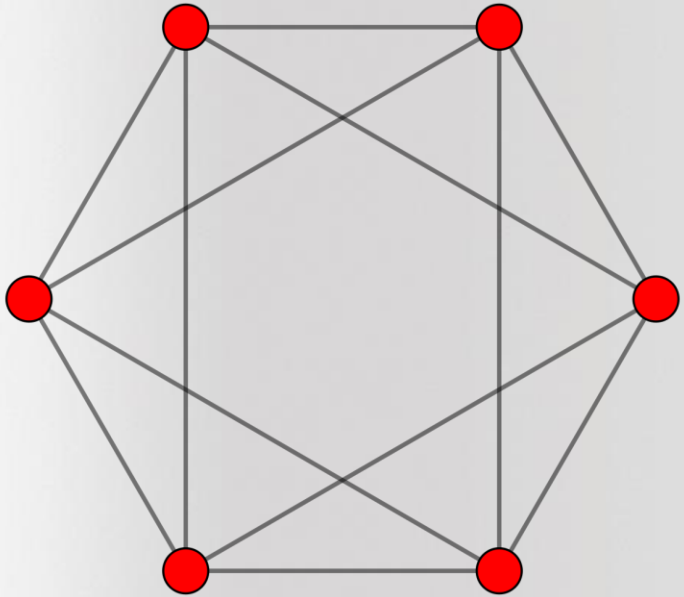
Vežbe br. 3

GRAFOVI

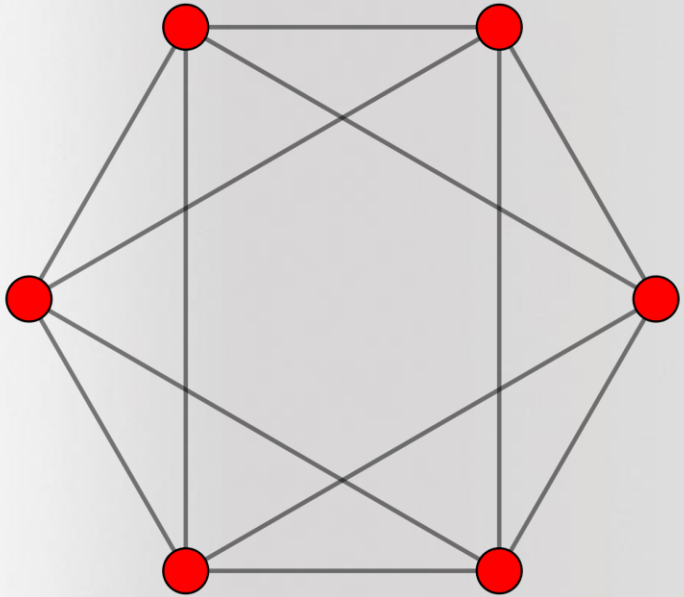
ŠTA JE GRAF?

- Struktura koja se sastoji od čvorova i grana
- Koristi se za modelovanje odnosa između objekata
- Matematički zapis: **G = (V, E)**
- Red grafa: broj čvorova |V|
- Veličina grafa: broj grana |E|



$|V| = ?$

$|E| = ?$



$|V| = 6$

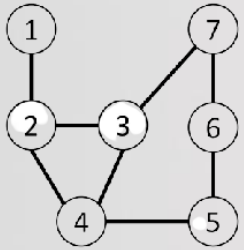
$|E| = 12$

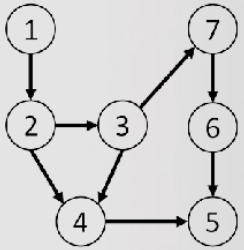
PRIMENA GRAFOVA

- Društvene mreže
- Web sajtovi
- Operativni sistemi
- Google mape
- .....

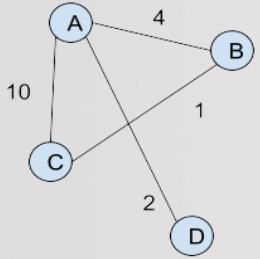
PODELA GRAFOVA

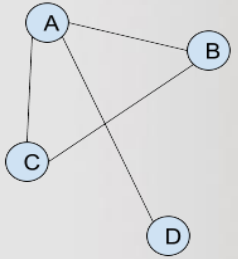
- Neusmereni i usmereni





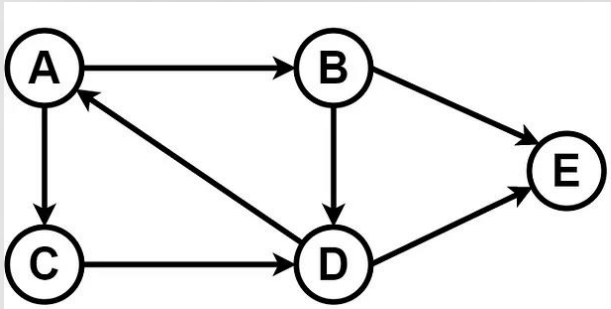
- Težinski i bestežinski





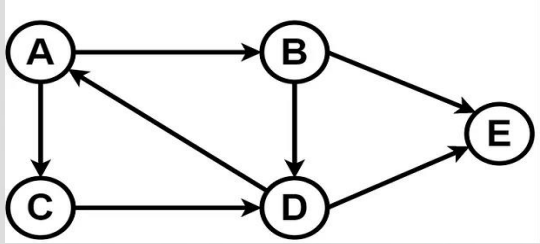
REPREZENTACIJA GRAFOVA U PYTHON-U

- **Matrica susedstva**



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	0	1
E	0	0	0	0	0

- Realizuje se u Python-u preko niza nizova

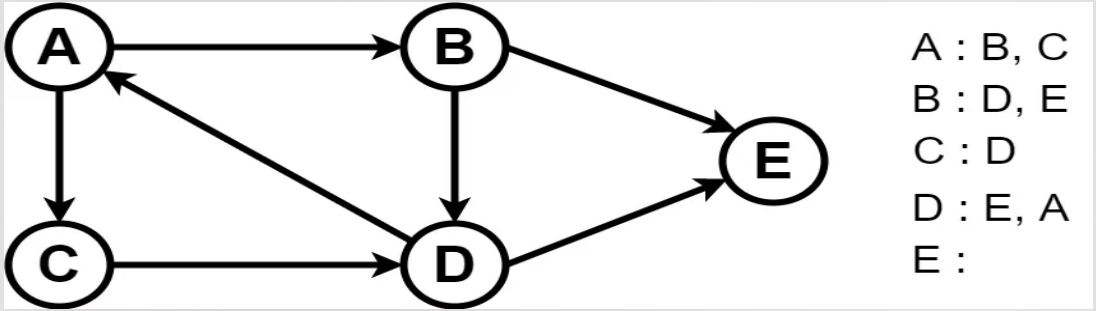


	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	0	1
E	0	0	0	0	0

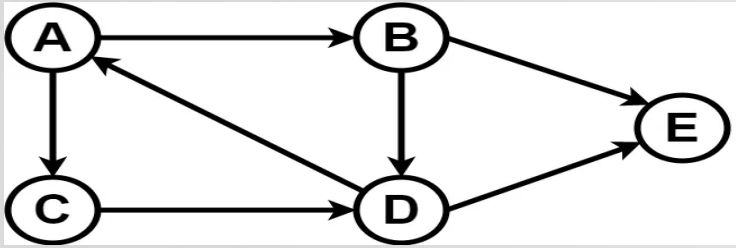
```
graph = [
        [0, 1, 1, 0, 0],
        [0, 0, 0, 1, 1],
        [0, 0, 0, 1, 0],
        [1, 0, 0, 0, 1],
        [0, 0, 0, 0, 0]
]
```

REPREZENTACIJA GRAFOVA U PYTHON-U

- **Lista susedstva**

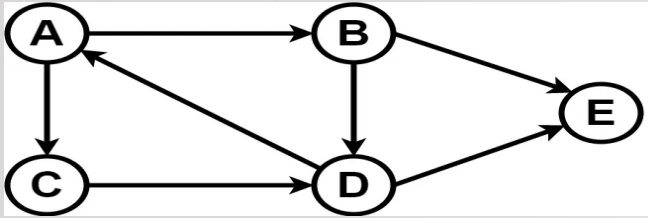


- Realizuje se u Python-u preko dictionary-ja



A : B, C
B : D, E
C : D
D : E, A
E :

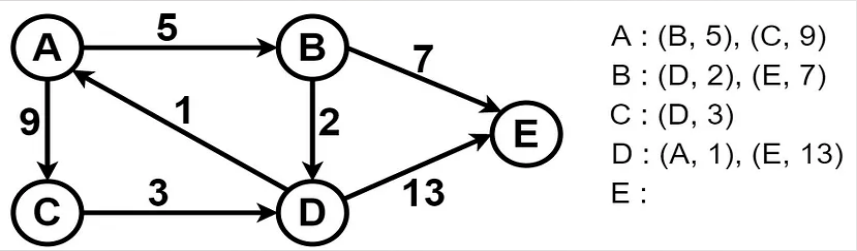
```
graph = {
        'A': ['B', 'C'],
        'B': ['D', 'E'],
        'C': ['D'],
        'D': ['E', 'A'],
        'E': [],
}
```



A : B, C
B : D, E
C : D
D : E, A
E :

Može i drugačije ☺

```
graph = dict()

def dodajGranu(cvor1, cvor2):
    if cvor1 not in graph:
        graph[cvor1] = []
    if cvor2 not in graph:
        graph[cvor2] = []

    graph[cvor1].append(cvor2)


dodajGranu('A', 'B')
dodajGranu('A', 'C')
```



A : (B, 5), (C, 9)
B : (D, 2), (E, 7)
C : (D, 3)
D : (A, 1), (E, 13)
E :

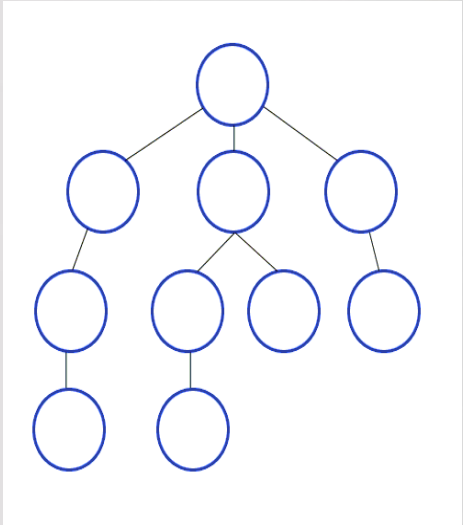
```
graph = dict()

def dodajGranu(cvor1, cvor2, tezina):
    if cvor1 not in graph:
        graph[cvor1] = []
    if cvor2 not in graph:
        graph[cvor2] = []

    graph[cvor1].append((cvor2, int(tezina)))


dodajGranu('A', 'B', 5)
dodajGranu('A', 'C', 9)
```

OBILAZAK GRAFA

- Cilj: „posetiti“ sve čvorove grafa (ako je moguće)
- Dva najpoznatija algoritma za obilazak (pretragu) grafa:
  - BFS
  - DFS

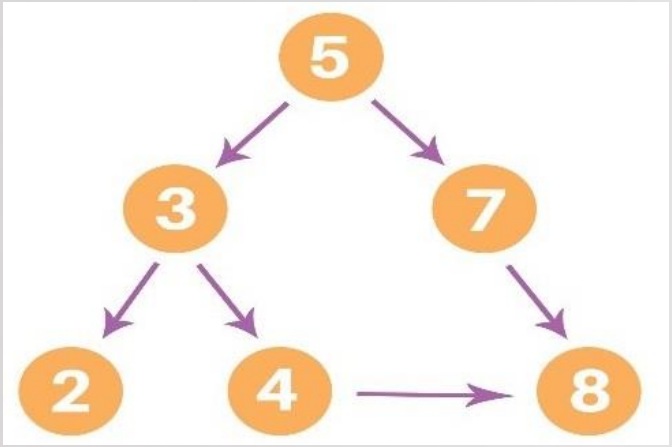
BFS (*Breadth-First Search*)

- Najjednostavniji algoritam pretrage grafa
- Počinje od startnog čvora i pretražuje „u širinu“: obilazi sve čvorove koji su na istoj udaljenosti od startnog čvora, pa prelazi na sledeći „nivo“...
- Ideja: čuvati niz čvorova koji su „posećeni“ i red čvorova koje treba „posetiti“
- Kompleksnost: **O(V + E)**



BFS (*Breadth-First Search*)

- ***Zadatak***: Implementirati BFS algoritam i testirati ga na sledećem grafu:



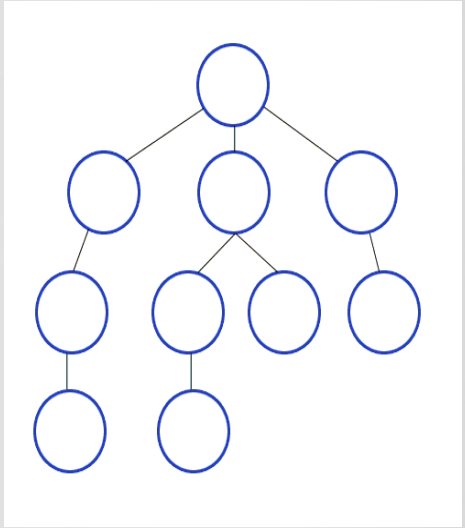
BFS (*Breadth-First Search*)

```
def BFS(visited, graph, cvor):
    visited.append(cvor)
    queue.append(cvor)

    while queue:
        m = queue.pop(0)
        print(m, end=" ")

        for sused in graph[m]:
            if sused not in visited:
                visited.append(sused)
                queue.append(sused)
```

```
graph = {
    '5': ['3', '7'],
    '3': ['2', '4'],
    '7': ['8'],
    '2': [],
    '4': ['8'],
    '8': [],
}

visited = []
queue = []

print("BFS za dati graf: ")
DFS(visited, graph,  '5')
```

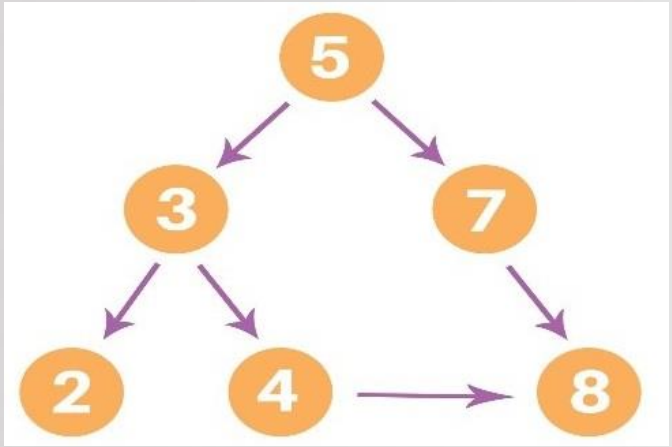
DFS (*Depth-First Search*)

- Rekurzivno pretraživanje grafa
- Počinje od startnog čvora i pretražuje „u dubinu“: uzme prvog suseda startnog čvora, analizira njegove susede sve do samog kraja grafa, pa se vraća na drugog suseda startnog čvora...
- Kompleksnost: **O(V+E)**



DFS (*Depth-First Search*)

- ***Zadatak***: Implementirati DFS algoritam i testirati ga na sledećem grafu:



DFS (*Depth-First Search*)

```
def DFS(visited, graph, cvor):
    if cvor not in visited:
        print(cvor)
        visited.append(cvor)

        for sused in graph[cvor]:
            DFS(visited, graph, sused)
```

```
graph = {
    '5': ['3', '7'],
    '3': ['2', '4'],
    '7': ['8'],
    '2': [],
    '4': ['8'],
    '8': [],
}

visited = []

print("DFS za dati graf: ")
DFS(visited, graph,  '5')
```

BFS vs DFS

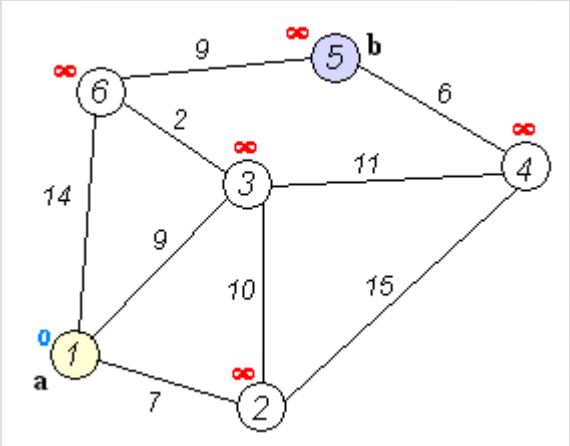
BFS	DFS
„FIFO“ princip	„LIFO“ princip
Za čvorove blizu „izvornog“	Za čvorove daleko od „izvornog“
Sporiji	Brži
Zahteva više memorije	Zahteva manje memorije
Pronalazi najkraći put	Ne pronalazi najkraći put
Ne može da uđe u beskonačnu petlju	Može da uđe u beskonačnu petlju

PRONALAZAK NAJKRAĆEG PUTA

- Cilj: pronaći u težinskom grafu najkraći put od polaznog čvora do svih ostalih čvorova
- Dva najpoznatija algoritma za pronalazak najkraćeg puta:
  - Dijkstra
  - Bellman - Ford

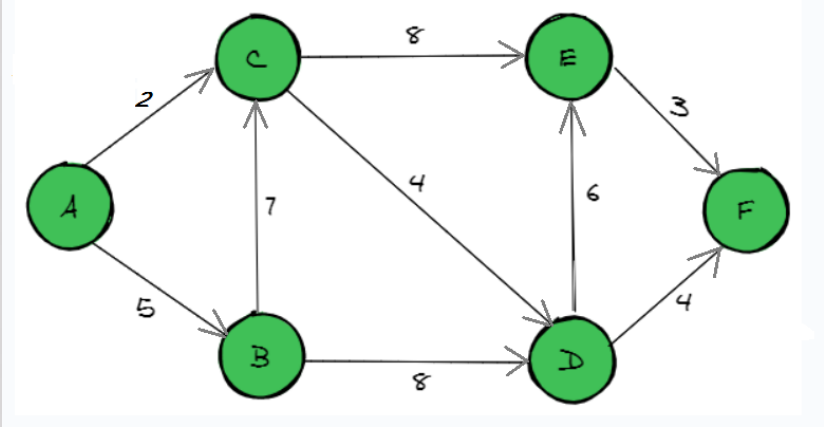
Dijkstra

- Osnovni algoritam za pronalaženje najkraćeg puta u grafu
- Ideja: Čuvati skup „neposećenih“ čvorova. Krenuti od početnog čvora i osvežiti najkraći put do svih njemu susednih čvorova. Označiti početni čvor kao „posećen“. Kao sledeći čvor za analizu uzeti onaj koji je na najmanjoj udaljenosti od polaznog čvora (računajući trenutno definisane udaljenosti). Ponavljati postupak dok se svi čvorovi ne označe kao „posećeni“.
- Kompleksnost: O(E + V*logV)



Dijkstra

- ***Zadatak***: Implementirati Dijkstra algoritam i testirati ga na sledećem grafu:

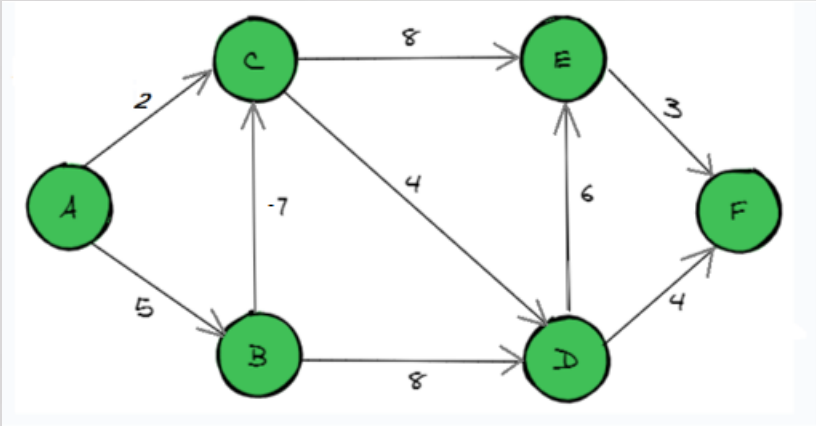


```
def dijkstra(graph, source):
    unvisited = graph
    dist = {}

    for cvor in unvisited:
        dist[cvor] = math.inf
    dist[source] = 0

    while unvisited:
        min_cvor = None

        for cvor in unvisited:
            if min_cvor is None or dist[min_cvor] > dist[cvor]:
                min_cvor = cvor

        for (cvor, value) in unvisited[min_cvor]:
            if value + dist[min_cvor] < dist[cvor]:
                dist[cvor] = value + dist[min_cvor]
        unvisited.pop(min_cvor)

    print("Udaljenost cvorova od pocetnog cvora: ")
    for cvor in dist:
        print("{0}\t\t{1}".format(cvor, dist[cvor]))
```

```
graph = {
        'a': [('b', 5), ('c', 2)],
        'b': [('c', 7), ('d', 8)],
        'c': [('d', 4), ('e', 8)],
        'd': [('e', 6), ('f', 4)],
        'e': [('f', 3)],
        'f': []
}

dijkstra(graph, 'a')
```

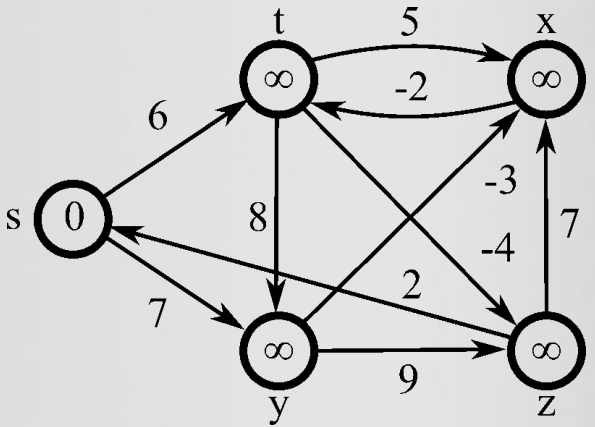
Dijkstra

- ***Zadatak 2***: Testirati Dijkstra algoritam na sledećem grafu:



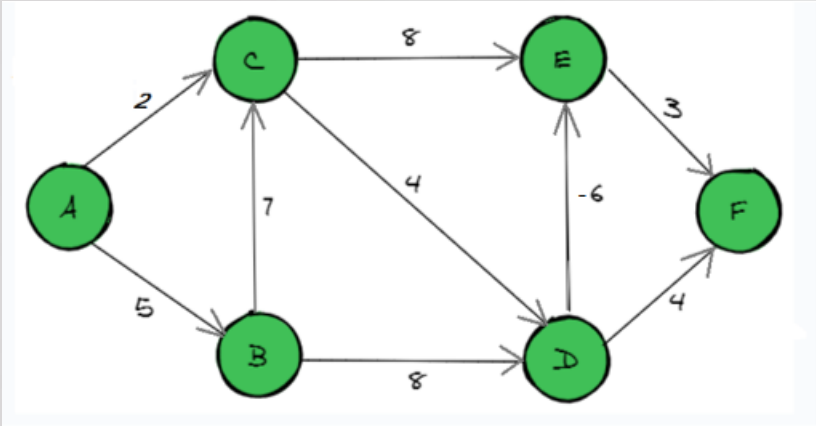
Bellman-Ford

- Rešava problem pronalaska najkraćeg puta u grafu koji sadrži negativne težine
- Ideja: slična kao kod Dijkstra algoritma, ali se ne pamte „posećeni“ i „neposećeni“ čvorovi, već se u svakoj iteraciji posmatra svaki čvor
- Kompleksnost: O(V * E)



Bellman-Ford

- ***Zadatak***: Implementirati Bellman-Ford algoritam i testirati ga na sledećem grafu:



```
def bellmanFord(graph, source):
    dist = {}

    for cvor in graph:
        dist[cvor] = math.inf
    dist[source] = 0

    for _ in range(len(graph) - 1):
        for cvor in graph:
            for (destCvor, value) in graph[cvor]:
                if dist[cvor] != math.inf and dist[cvor] + value < dist[destCvor]:
                    dist[destCvor] = dist[cvor] + value

    for cvor in graph:
        for (destCvor, value) in graph[cvor]:
            if dist[cvor] != math.inf and dist[cvor] + value < dist[destCvor]:
                print("Graf sadrzi negativnu kruznu putanju!")
                return

    print("Udaljenost cvorova od pocetnog cvora: ")
    for cvor in dist:
        print("{0}\t\t{1}".format(cvor, dist[cvor]))
```

```
graph = {
        'a': [('b', 5), ('c', 2)],
        'b': [('c', 7), ('d', 8)],
        'c': [('d', 4), ('e', 8)],
        'd': [('e', -6), ('f', 4)],
        'e': [('f', 3)],
        'f': []
}

bellmanFord(graph, 'a')
```

TO BE CONTINUED... ☺