Vežbe br. 4

GRAFOVI

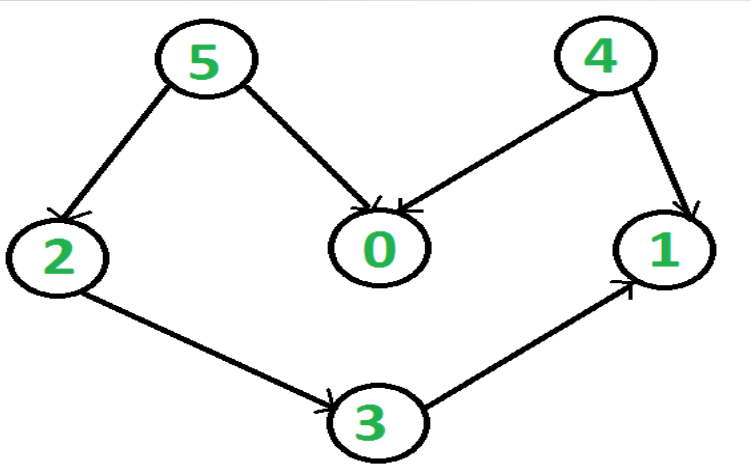
(deo 2)

Topološko sortiranje grafa

- Predstavlja vid analize, tj. “uređenja“ grafa
- Primenjuje se na **usmerenim** grafovima
- Vraća niz čvorova, tako da se u tom nizu svaki čvor pojavljuje pre onih na koje “pokazuje“, tj. do kojih vodi
- Može se posmatrati kao modifikacija DFS-a

Topološko sortiranje grafa

- ***Zadatak***: Implementirati topološko sortiranje i testirati ga na sledećem grafu:



```
class Graph:
    def __init__(self):
        self.graph = {}

    def addEdge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append(v)

    def topologicalSortUtil(self, v, visited, stack):
        visited[v] = True

        for i in self.graph[v]:
            if not visited[i]:
                self.topologicalSortUtil(i, visited, stack)

        stack.insert(0, v)

    def topologicalSort(self):
        visited = [False] * len(self.graph)
        stack = []

        for i in range(len(self.graph)):
            if not visited[i]:
                self.topologicalSortUtil(i, visited, stack)
        print(stack)
```

```
g = Graph()

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

print("Topoloski sortiran graf:")
g.topologicalSort()
```

Minimalno razapinjuće stablo (Minimum spanning tree – MST)

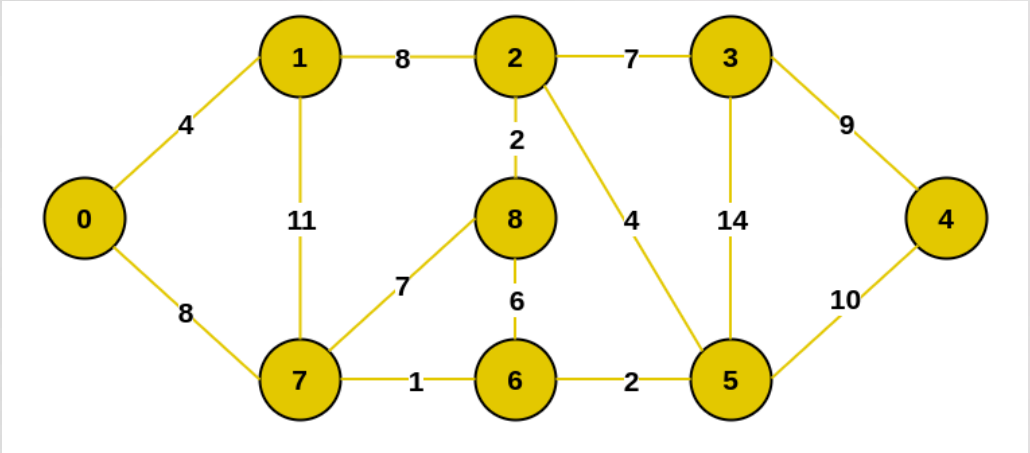
- Može se formirati za **neusmerene težinske** grafove
- Predstavlja strukturu (“stablo“) koja sadrži sve čvorove nekog grafa, pri čemu je zbir težina grana te strukture najmanji mogući
- Jedan graf može imati više različitih MST
- Dva najpoznatija algoritma za određivanje MST:
  - Primov algoritam
  - Kruskalov algoritam

Primov algoritam

- **Ideja:**
  - Voditi računa o 2 skupa čvorova: onim koji se do sada nalaze u MST i onim koji se još uvek ne nalaze
  - Početi od izvornog čvora i dodavati u MST granu koja ima najmanju težinu, a povezuje neki od čvorova iz prvog skupa sa nekim od čvorova iz drugog skupa
  - Ponavljati postupak sve dok MST ne sadrži V – 1 granu
- **Kompleksnost:**
  - **$O(V^2)$,** ako se koristi matrica susedstva
  - **$O(V * logV)$**, ako se koristi lista susedstva

Primov algoritam

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



```python
class Graph:
    def __init__(self):
        self.graph = {}

    def addEdge(self, u, v, w):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append((v, w))
        self.graph[v].append((u, w))
```

```python
def prim(self):
    visited = [False] * len(self.graph)
    E = 0
    visited[0] = True

    print("Grana   :  Tezina\n")
    while E < len(self.graph) - 1:
        minimum = math.inf
        a = 0
        b = 0
        tezina = 0
        for m in range(len(self.graph)):
            if visited[m]:
                for n in range(len(self.graph)):
                    if not visited[n]:
                        mn_exists = False
                        for (cvor, grana) in self.graph[m]:
                            if cvor == n:
                                mn_exists = True
                                tezinaMN = grana
                            if mn_exists:
                                if minimum > tezinaMN:
                                    minimum = tezinaMN
                                    a = m
                                    b = n
        print(str(a) + " - " + str(b) + ":   " + str(minimum))
        visited[b] = True
        E += 1
```

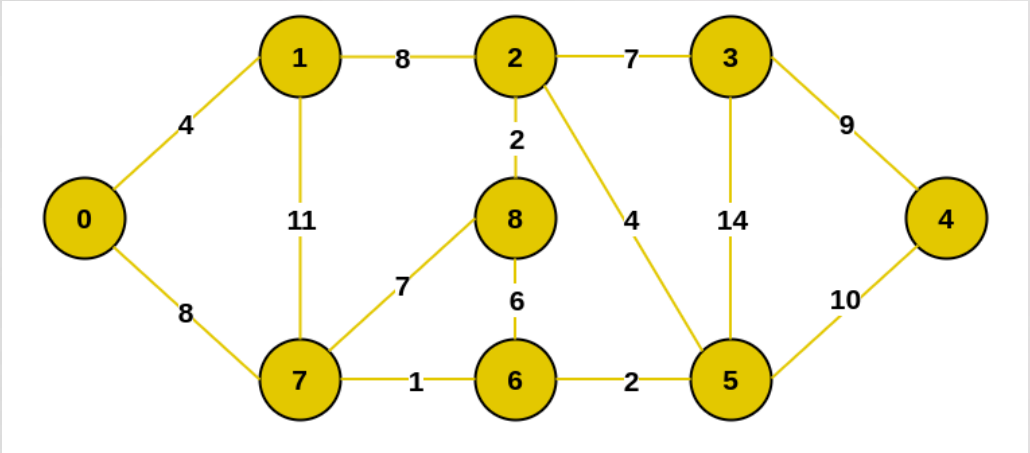
```python
g = Graph()

g.addEdge(0, 1, 4)
g.addEdge(0, 7, 8)
g.addEdge(1, 2, 8)
g.addEdge(1, 7, 11)
g.addEdge(2, 3, 7)
g.addEdge(2, 5, 4)
g.addEdge(2, 8, 2)
g.addEdge(3, 4, 9)
g.addEdge(3, 5, 14)
g.addEdge(4, 5, 10)
g.addEdge(5, 6, 2)
g.addEdge(6, 7, 1)
g.addEdge(6, 8, 6)
g.addEdge(7, 8, 7)


print("Primov algoritam:")
g.prim()
```

Kruskalov algoritam

- **Ideja:**
  - Sortirati grane po težinama, počevši od najmanje
  - Dodati najmanju granu u MST, pod uslovom da dodavanje nove grane ne formira konturu
  - Ponavljati postupak sve dok MST ne sadrži V – 1 granu
- **Kompleksnost:** **O(E * logV)**

Kruskalov algoritam

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



```
class Graph:
    def __init__(self):
        self.graph = {}
        self.list = []

    def addEdge(self, u, v, w):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append((v, w))
        self.graph[v].append((u, w))
        self.list.append([u, v, w])

    def find(self, parent, i):
        if parent[i] != i:
            parent[i] = self.find(parent, parent[i])
        return parent[i]

    def union(self, parent, rank, x, y):
        if rank[x] < rank[y]:
            parent[x] = y
        elif rank[x] > rank[y]:
            parent[y] = x
        else:
            parent[y] = x
            rank[x] += 1
```

```
def kruskal(self):
    result = []
    i, e = 0
    self.list = sorted(self.list, key=lambda item: item[2])
    parent = []
    rank = []

    for node in range(len(self.graph)):
        parent.append(node)
        rank.append(0)

    while e < len(self.graph) - 1:
        u, v, w = self.list[i]
        i = i + 1
        x = self.find(parent, u)
        y = self.find(parent, v)

        if x != y:
            e = e + 1
            result.append([u, v, w])
            self.union(parent, rank, x, y)

    print("Grane koje se nalaze u MST:")
    for u, v, weight in result:
        print("%d -- %d == %d" % (u, v, weight))
```

```
g = Graph()

g.addEdge(0, 1, 4)
g.addEdge(0, 7, 8)
g.addEdge(1, 2, 8)
g.addEdge(1, 7, 11)
g.addEdge(2, 3, 7)
g.addEdge(2, 5, 4)
g.addEdge(2, 8, 2)
g.addEdge(3, 4, 9)
g.addEdge(3, 5, 14)
g.addEdge(4, 5, 10)
g.addEdge(5, 6, 2)
g.addEdge(6, 7, 1)
g.addEdge(6, 8, 6)
g.addEdge(7, 8, 7)


print("Kruskalov algoritam:")
g.kruskal()
```

TO BE CONTINUED... ☺