Softverski algoritmi u sistemima automatskog upravljanja

Grafovi 2

Danilo Kaćanski, Ivan Radman

Topološko sortiranje

Osnovni algoritam za uređivanje usmerenog grafa bez ciklusa (DAG)

🛠️ **Ideja:**

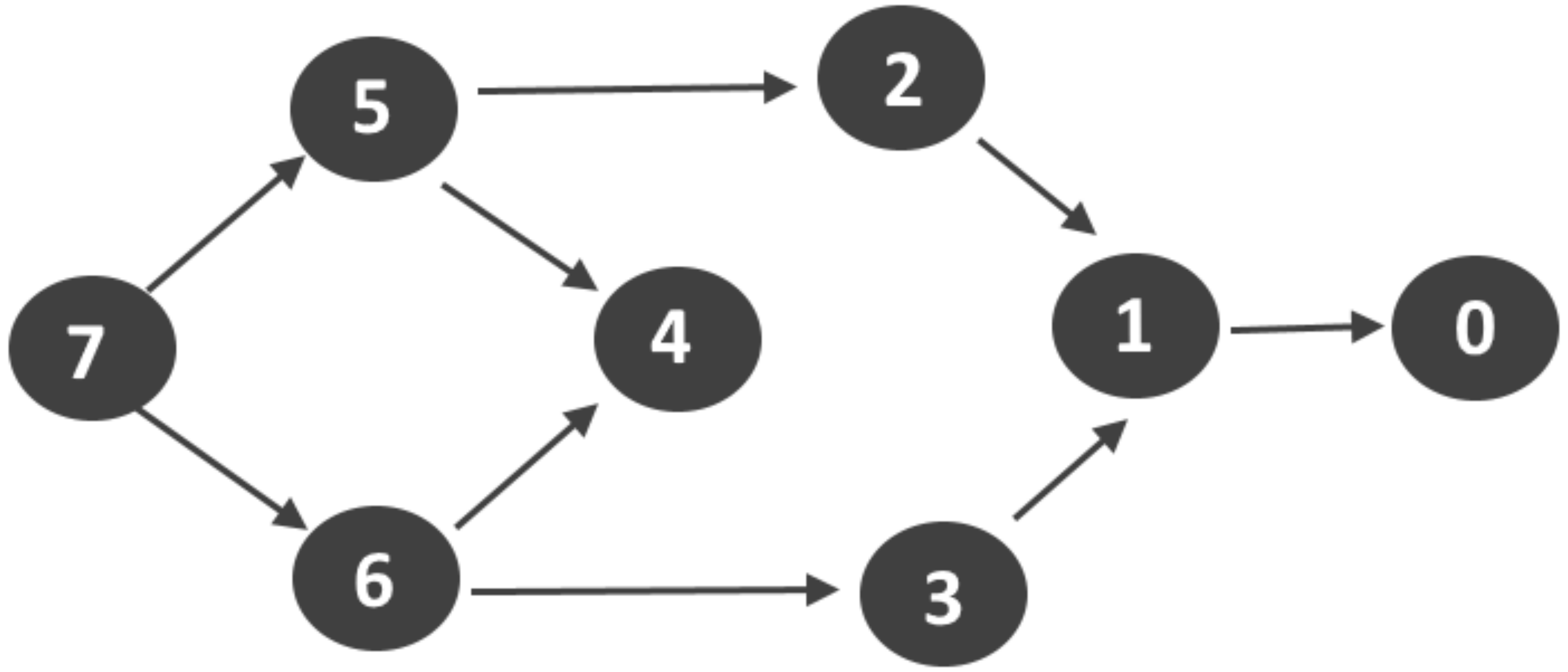
- Radi se nad **usmerenim acikličnim grafom** (*DAG*)
- Vraća niz čvorova tako da svaki čvor dolazi **pre svih čvorova do kojih vodi**
- U suštini: Prvo se izvršavaju zavisnosti, pa tek onda zavisni elementi
- Implementacija najčešće koristi **modifikovani DFS**

⏱️ **Kompleksnost:**

- **Vremenska složenost:** $O(V + E)$

📌 **Primenjuje se u:**

- Planiranju zadataka (*task scheduling*)
- Kompajlerima (*redosled izvršavanja*)
- Analizi zavisnosti

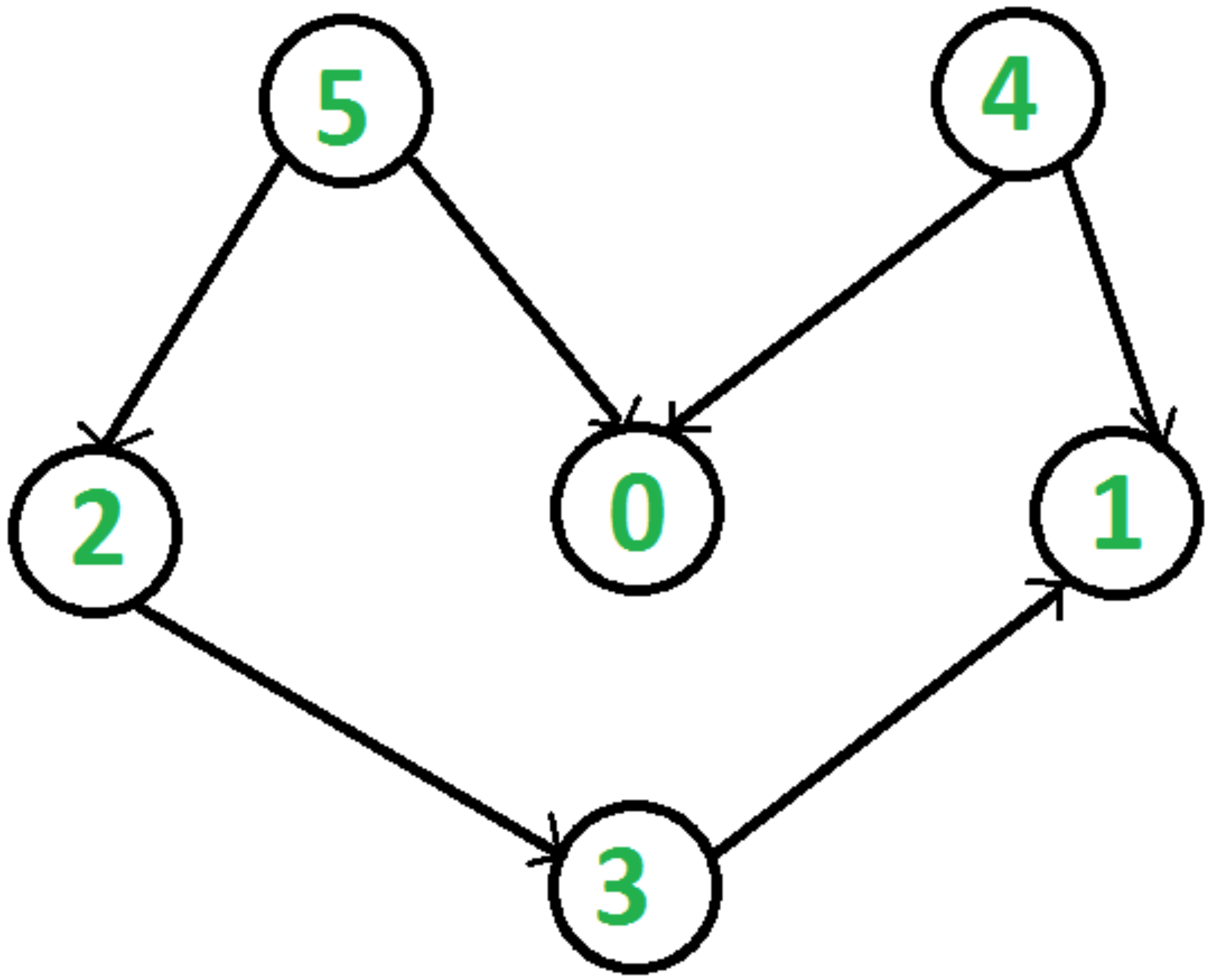


Topological Sort : 7 6 5 4 3 2 1 0

Topološko sortiranje

🔍**Zadatak:**

- Implementirati topološko sortiranje i testirati ga na sledećem grafu:



Topološko sortiranje

✅ Topološko sortiranje:

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

🗺️ Reprezentacija grafa sa slike i poziv funkcije:

```
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()
```

📤 Očekivani izlaz:

```
[4, 5, 2, 3, 1, 0]
```

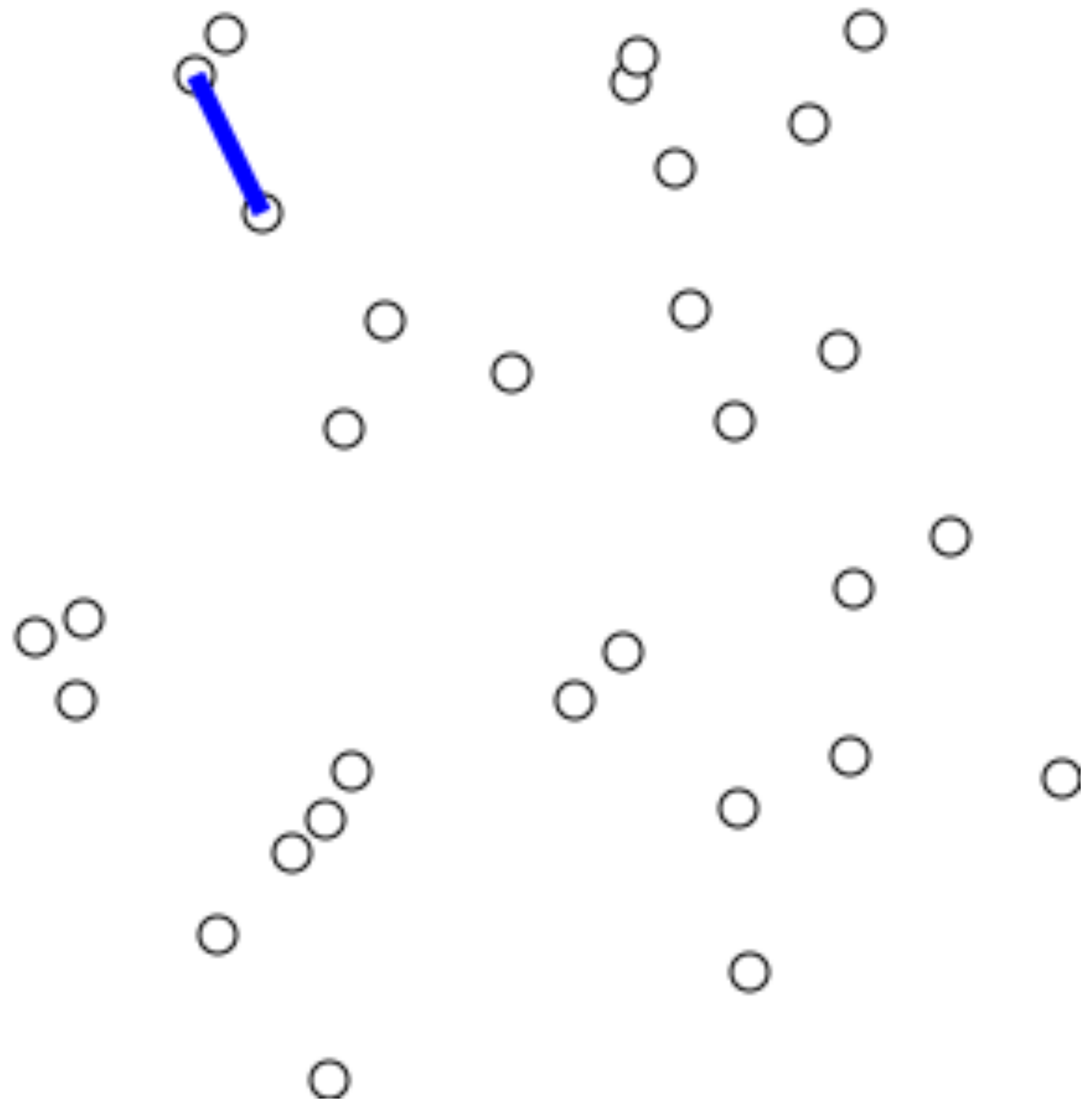
Minimum Spannig Tree (*MST*)

🛠️ **Ideja:**

- **Ulaz:** neusmeren težinski graf (graf sa nenegativnim težinama)
- **Cilj:** pronaći podskup grana koji:
  - Povezuje **sve čvorove grafa**
  - **Ne sadrži cikluse** (*tj. formira stablo*)
  - Ima **najmanju moguću ukupnu težinu**

📌 **Osobine:**

- *MST* **nije jedinstven** — može postojati više različitih
- U grafu sa *V* **čvorova**, *MST* uvek ima tačno *V - 1* **granu**
- Ne menja strukturu grafa — koristi se kao **podgraf**



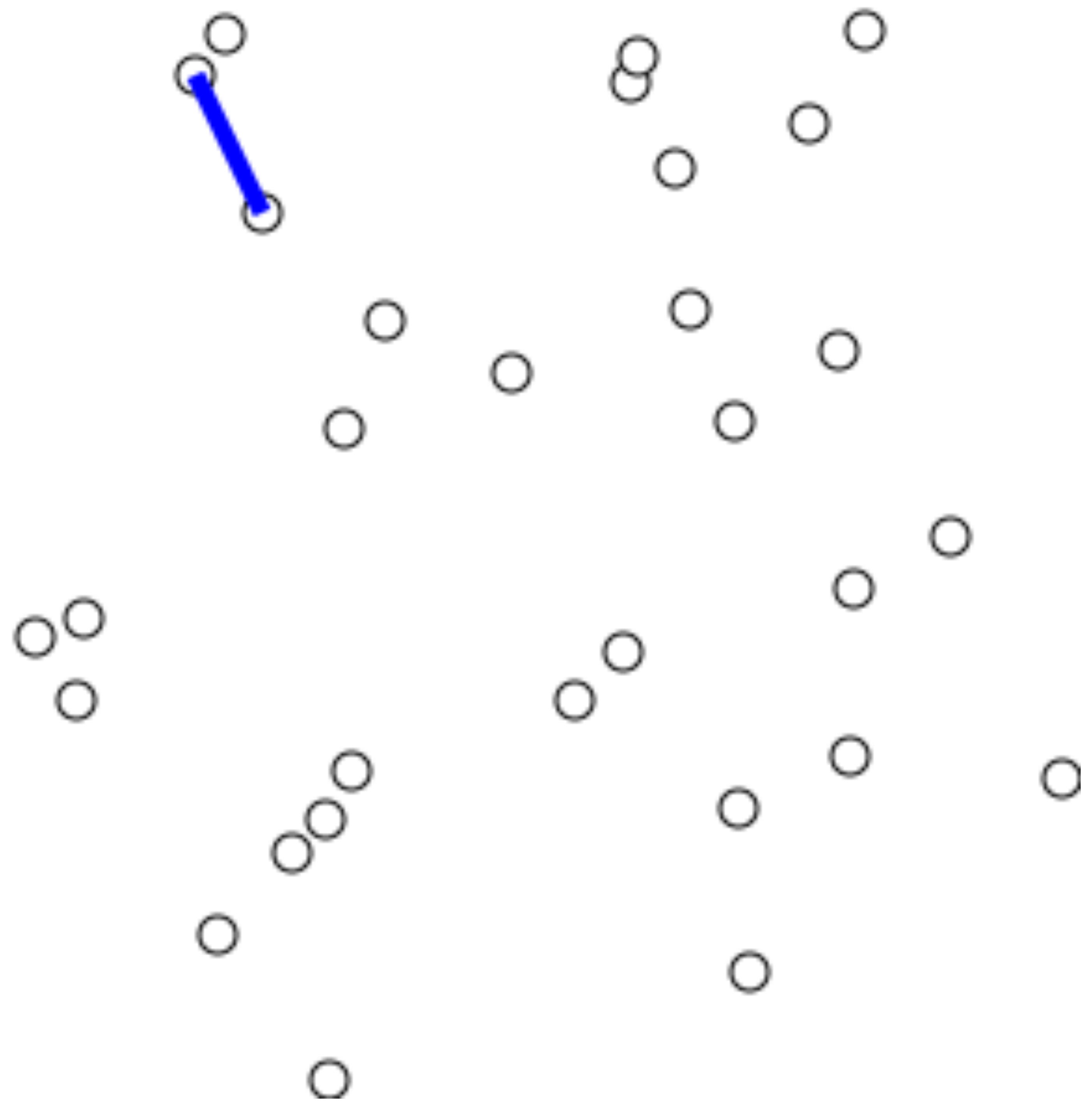
Minimum Spannig Tree (*MST*)

🛠️ **Najpoznatiji algoritmi:**

- **Primov algoritam** — gradi *MST* dodavanjem *najbližeg čvora*
- **Kruskalov algoritam** — gradi *MST* dodavanjem *najlakših grana*

📌 **Praktična primena:**

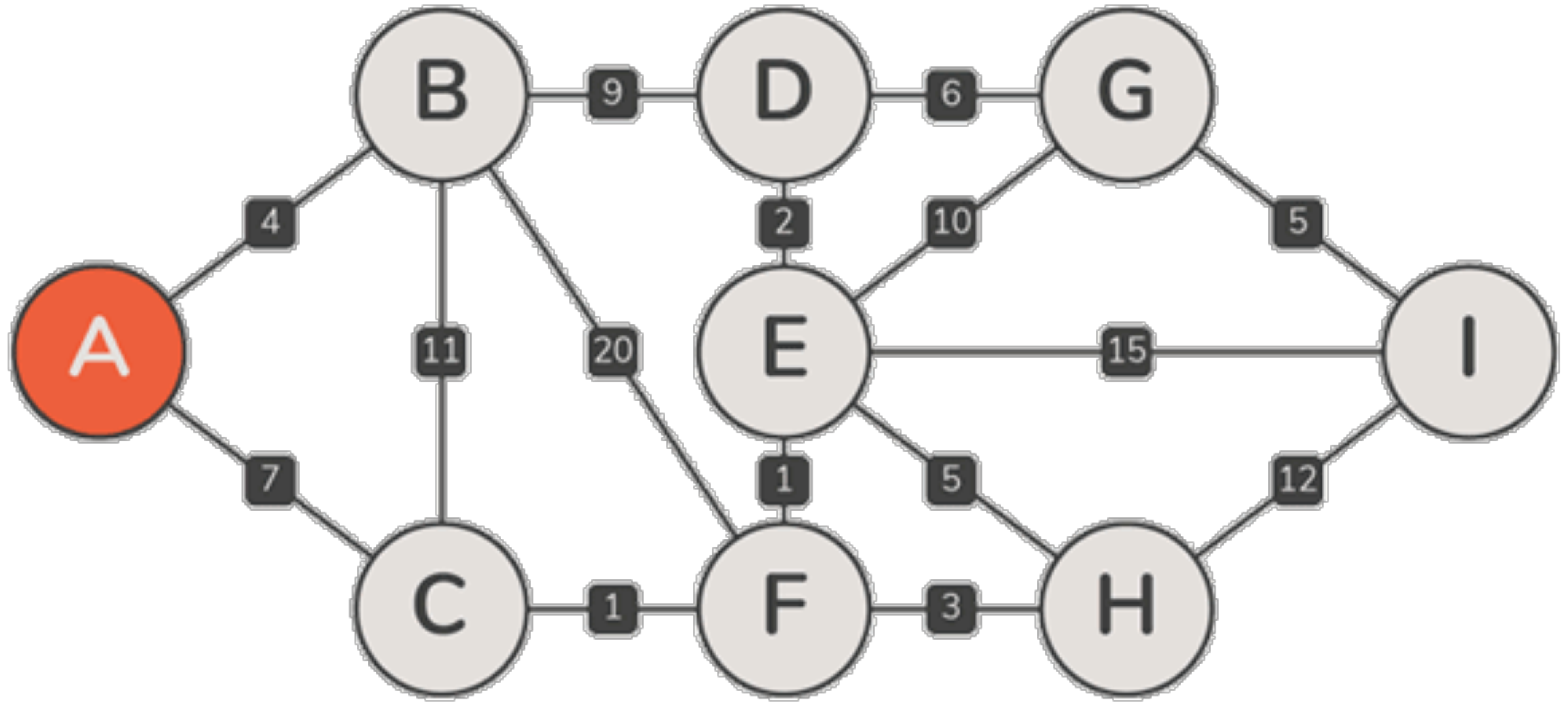
- Optimizacija mreža (*npr. kablovska mreža, vodovod*)
- Projektovanje puteva, minimizacija troškova povezivanja



Primov algoritam

🛠️ **Ideja:**

- Voditi računa o 2 skupa čvorova: onim koji su *već u MST* i onim koji *još nisu*
- Početi od izvornog čvora i dodavati **granu sa najmanjom težinom koja povezuje oba skupa**
- Ponavaljati dok MST ne sadrži **tačno** *V-1* **granu**



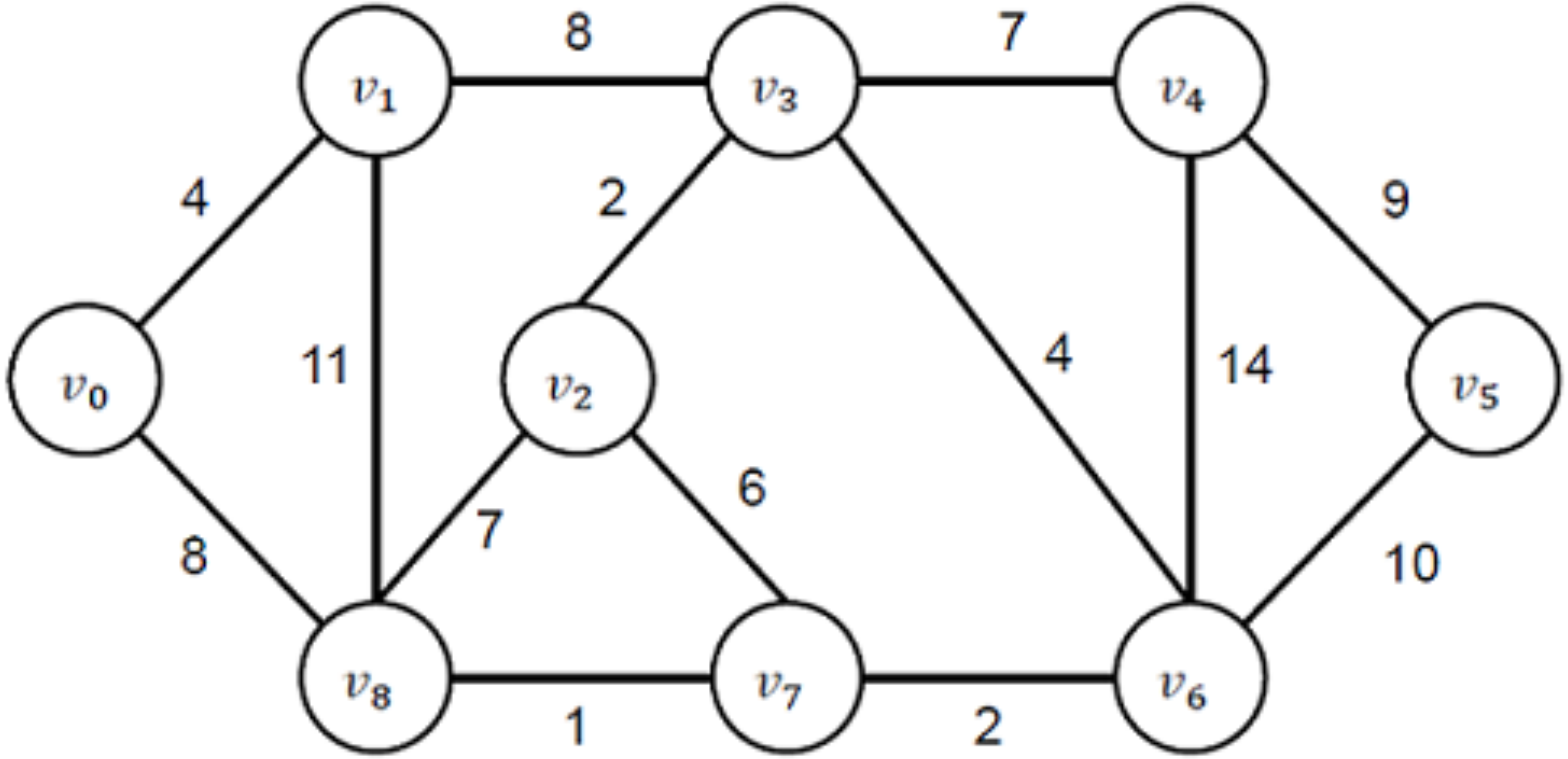
⏱️ **Kompleksnost:**

- Ako se koriti **matrica susedsva**: $O(V^2)$
- Ako se koristi **lista susedstva** i **prioritetni red** (npr. m*in-heap*): $O(V \cdot log\, V)$

Primov algoritam

🔍**Zadatak:**

- Implementirati Primov algoritam i testirati ga na sledećem grafu:



Primov algoritam

✅ Primov algoritam:

```
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))

    def prim(self):
        visited = [False] * len(self.graph)
        visited[0] = True
        E = 0
        print("Grana  -  Tezina")
        while E < len(self.graph) - 1:
            minimum = float('inf')
            a = b = 0
            for m in range(len(self.graph)):
                if visited[m]:
                    for to, weight in self.graph[m]:
                        if not visited[to] and weight < minimum:
                            minimum = weight
                            a = m
                            b = to
            print(f"{a} - {b}    {minimum}")
            visited[b] = True
            E += 1
```

🗺️ Reprezentacija grafa sa slike i poziv funkcije:

```
g = Graph()
g.addEdge("v0", "v1", 4)
g.addEdge("v0", "v7", 8)
g.addEdge("v1", "v2", 8)
g.addEdge("v1", "v7", 11)
g.addEdge("v2", "v3", 7)
g.addEdge("v2", "v8", 2)
g.addEdge("v2", "v5", 4)
g.addEdge("v3", "v4", 9)
g.addEdge("v3", "v5", 14)
g.addEdge("v4", "v5", 10)
g.addEdge("v5", "v6", 2)
g.addEdge("v6", "v7", 1)
g.addEdge("v6", "v8", 6)
g.addEdge("v7", "v8", 7)

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

Primov algoritam

✅Primov algoritam:

```
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))

    def prim(self):
        visited = [False] * len(self.graph)
        visited[0] = True
        E = 0
        print("Grana  -  Tezina")
        while E < len(self.graph) - 1:
            minimum = float('inf')
            a = b = 0
            for m in range(len(self.graph)):
                if visited[m]:
                    for to, weight in self.graph[m]:
                        if not visited[to] and weight < minimum:
                            minimum = weight
                            a = m
                            b = to
            print(f"{a} - {b}    {minimum}")
            visited[b] = True
            E += 1
```

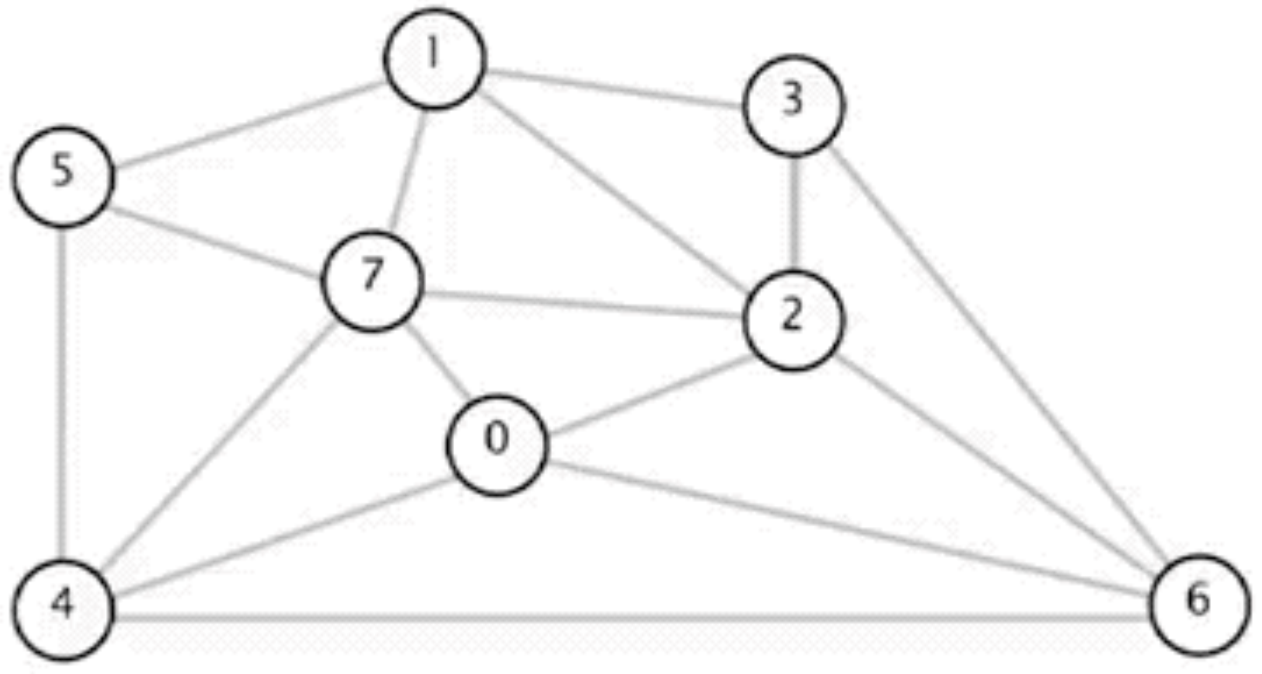
📤Očekivani izlaz:

```
Grana    :    Težina
v0 - v1 :    4
v0 - v7 :    8
v7 - v6 :    1
v6 - v5 :    2
v5 - v2 :    4
v2 - v8 :    2
v2 - v3 :    7
v3 - v4 :    9
```

Kruskalov algoritam

🛠️ **Ideja:**

1. **Sotirati** sve grane grafa po težinama (*od najamnje ka najvećoj*)
2. Inicijalno, svaki čvor je u **svom posebnom skupu**
3. Prolaziti kroz sotirane grane:
   - Ako krajevi grane priapadaju **različitim skupovima** → **dodati** granu u MST
   - **Spojiti** ta dva skupa
4. Zaustaviti kada MST sadrži **tačno** *V-1* **granu**



an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

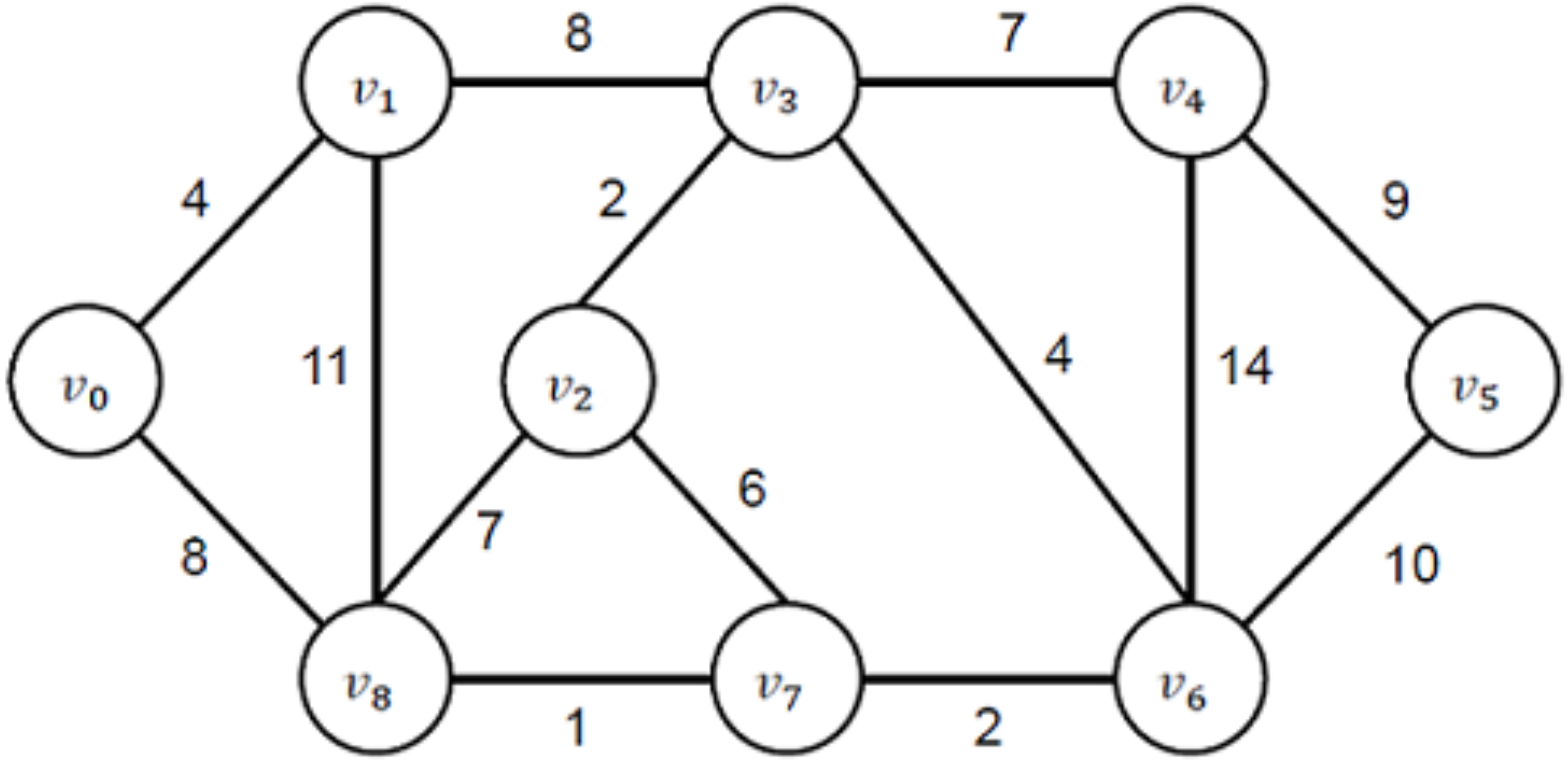
⏱️ **Kompleksnost:**

- Ako se koriti **matrica susedsva**: $O(V^2)$
- Ako se koristi **lista susedstva** i **prioritetni red** (npr. m*in-heap*): $O(V \cdot log\ V)$

Kruskalov algoritam

🔍**Zadatak:**

- Implementirati Kruskalov algoritam i testirati ga na sledećem grafu:



Kruskalov algoritam

🧩union() + find() sa optimizacijom:

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

    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.edges.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):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1
```

✅Kruskalov algoritam:

```
def kruskal(self):
        result = []
        i, e = 0, 0
        self.edges = sorted(self.edges, key=lambda item: item[2])
        parent = {}
        rank = {}
        for node in self.graph:
            parent[node] = node
            rank[node] = 0

        while e < len(self.graph) - 1:
            u, v, w = self.edges[i]
            i += 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                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(f"{u} - {v}: {weight}")
```