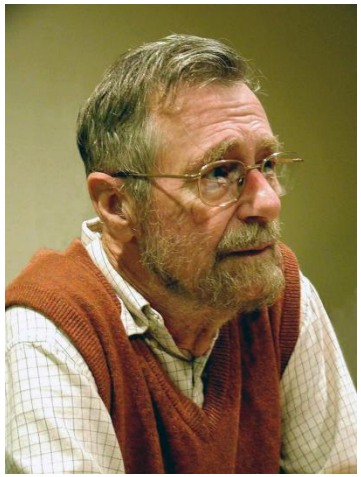
# Dijkstra's Algorithm

Single-source shortest paths



# Dijkstra's Algorithm

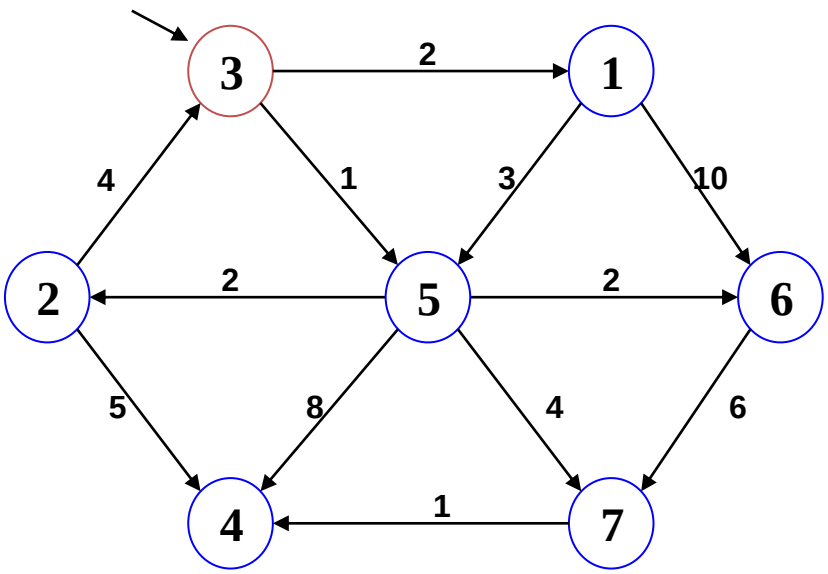
- Cities as vertices and connecting roadways as edges can be modeled as a Graph.
- An important application is to find the shortest path between two cities.
- Dijkstra(*Die-k-straw*) algorithm can be used to find the shortest path from a source vertex to all other vertices.

# Dijkstra's Algorithm

- It works for both directed and undirected non-negative weighted graphs.
- The algorithm maintains two sets of vertices.
- Selected vertices whose shortest distance has been computed.

# Dijkstra's Algorithm

- Unselected vertices are those vertices whose shortest distance is yet to be computed.
- Unselected vertices record the shortest distance from it to Selected vertices.
- Whenever a vertex is selected, unselected vertices update their shortest distances.

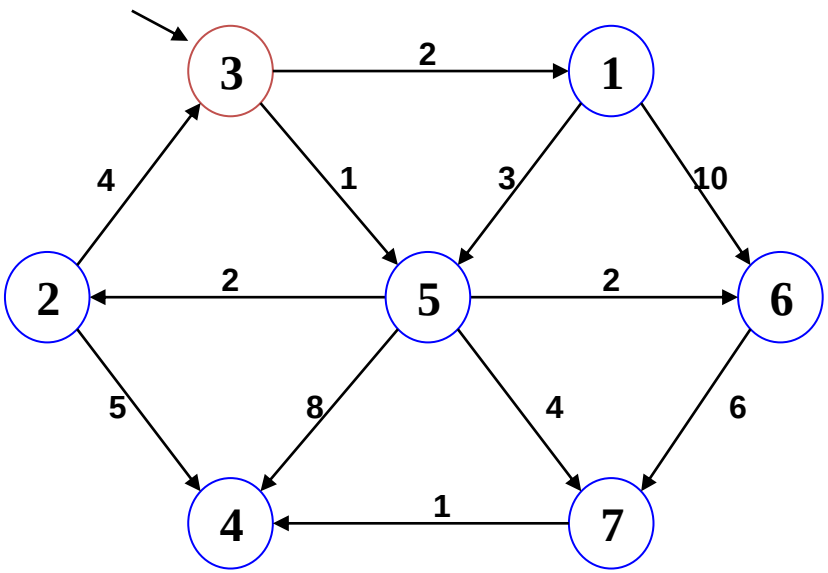



| cost | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | | | | 3 | 10 | |
| 2 | | 0 | 4 | 5 | | | |
| 3 | 2 | | 0 | | 1 | | |
| 4 | | | | 0 | | | |
| 5 | | 2 | | 8 | 0 | 2 | 4 |
| 6 | | | | | | 0 | 6 |
| 7 | | | | 1 | | | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Selected | F | F | ~~F~~ | F | F | F | F |
| dist | ~~∞~~ | ∞ | 0 | ∞ | ~~∞~~ | ∞ | ∞ |
| prev | - | - | - | - | - | - | - |

**Nearest unselected vertex u=3**

```
for each unselected adjacent vertex of u(3) do
    if(dist[w]>dist[u]+cost[u,w)) then
        dist[w]:= dist[u]+cost[u,w]
```

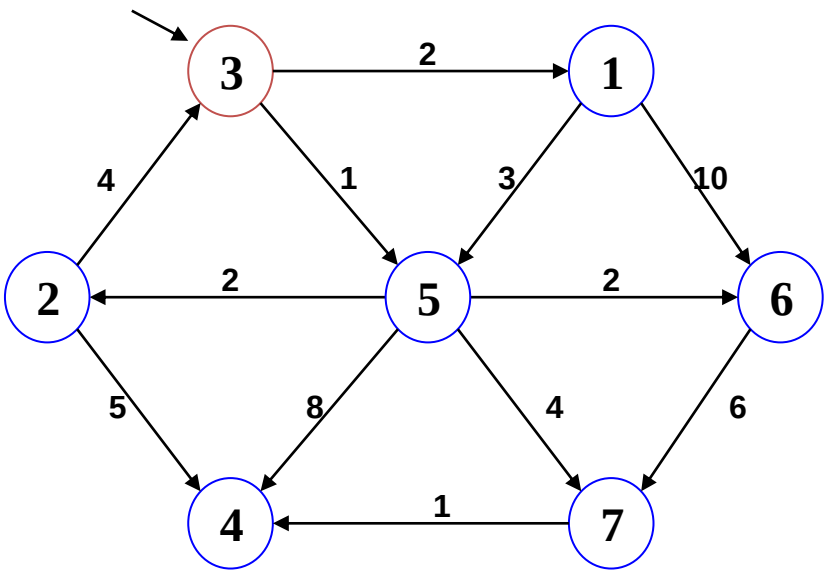



| cost | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | | | | 3 | 10 | |
| 2 | | 0 | 4 | 5 | | | |
| 3 | 2 | | 0 | | 1 | | |
| 4 | | | | 0 | | | |
| 5 | | 2 | | 8 | 0 | 2 | 4 |
| 6 | | | | | | 0 | 6 |
| 7 | | | | 1 | | | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Selected | F | F | T | F | ~~F~~ | F | F |
| dist | 2 | ~~∞~~ | 0 | ~~∞~~ | 1 | ~~∞~~ | ~~∞~~ |
| prev | 3 | | | | 3 | | |

**Nearest unselected vertex u=5**

```
for each unselected adjacent vertex of u(5) do
    if(dist[w]>dist[u]+cost[u,w)) then
        dist[w]:= dist[u]+cost[u,w]
```

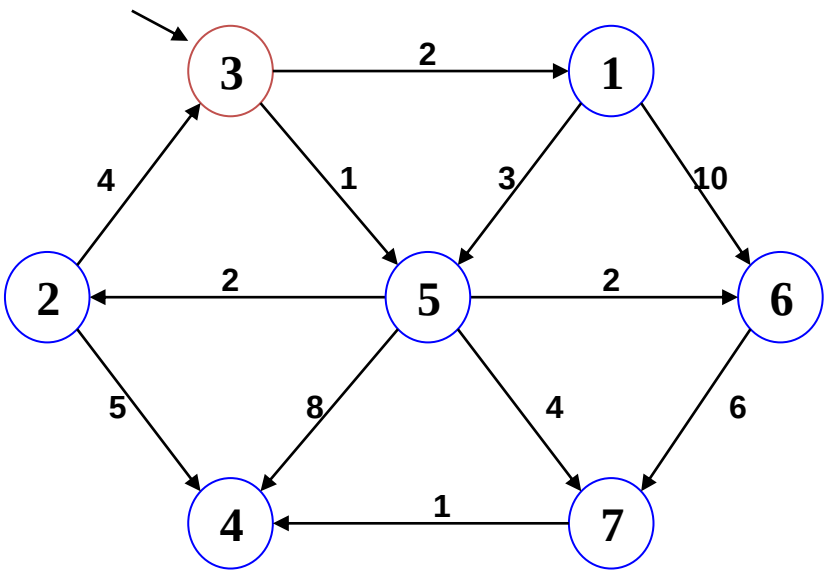



| cost | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | | | | 3 | 10 | |
| 2 | | 0 | 4 | 5 | | | |
| 3 | 2 | | 0 | | 1 | | |
| 4 | | | | 0 | | | |
| 5 | | 2 | | 8 | 0 | 2 | 4 |
| 6 | | | | | | 0 | 6 |
| 7 | | | | 1 | | | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Selected | ~~F~~ | F | T | F | T | F | F |
| dist | 2 | 3 | 0 | 9 | 1 | 3 | 5 |
| prev | 3 | 5 | | 5 | 3 | 5 | 5 |

**Nearest unselected vertex u=1**

```
for each unselected adjacent vertex w of u(1) do
    if(dist[w]>dist[u]+cost[u,w)) then
        dist[w]:= dist[u]+cost[u,w]
```

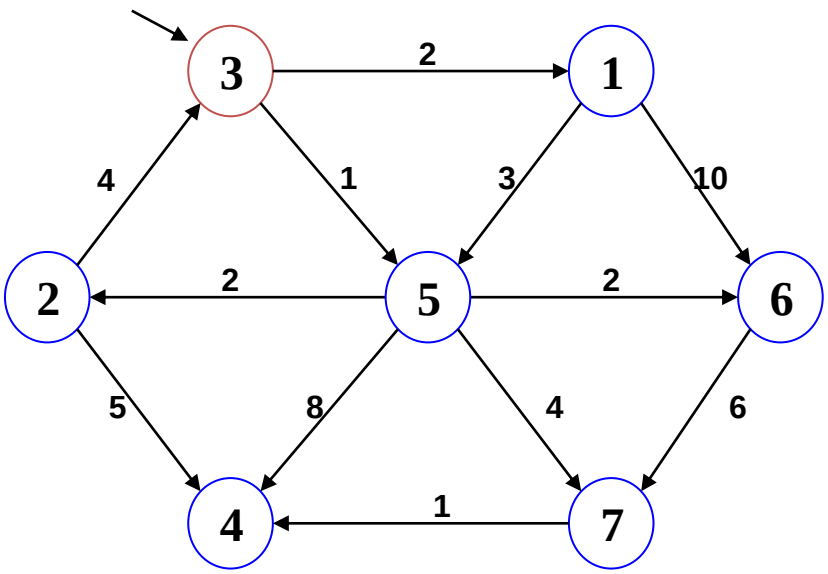



| cost | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | | | | 3 | 10 | |
| 2 | | 0 | 4 | 5 | | | |
| 3 | 2 | | 0 | | 1 | | |
| 4 | | | | 0 | | | |
| 5 | | 2 | | 8 | 0 | 2 | 4 |
| 6 | | | | | | 0 | 6 |
| 7 | | | | 1 | | | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Selected | T | ~~F~~ | T | F | T | F | F |
| dist | 2 | 3 | 0 | ~~9~~ | 1 | 3 | 5 |
| prev | 3 | 5 | | 5 | 3 | 5 | 5 |

Nearest unselected vertex u=2

```
for each unselected adjacent vertex w of u(2)
do
    if(dist[w]>dist[u]+cost[u,w)) then
        dist[w]:= dist[u]+cost[u,w]
```

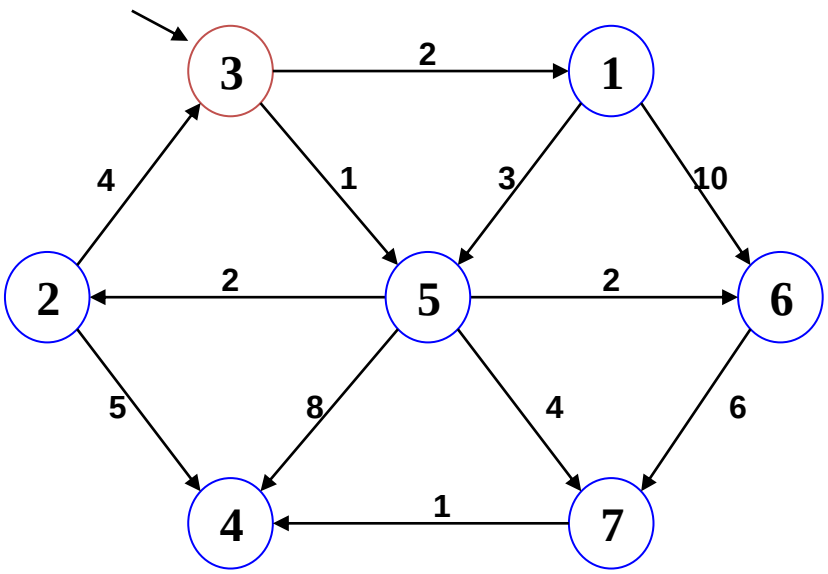



| cost | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | | | | 3 | 10 | |
| 2 | | 0 | 4 | 5 | | | |
| 3 | 2 | | 0 | | 1 | | |
| 4 | | | | 0 | | | |
| 5 | | 2 | | 8 | 0 | 2 | 4 |
| 6 | | | | | | 0 | 6 |
| 7 | | | | 1 | | | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Selected | T | T | T | F | T | ~~F~~ | F |
| dist | 2 | 3 | 0 | 8 | 1 | 3 | 5 |
| prev | 3 | 5 | | 2 | 3 | 5 | 5 |

**Nearest unselected vertex u=6**

```
for each unselected adjacent vertex w of u(6) do
    if(dist[w]>dist[u]+cost[u,w)) then
        dist[w]:= dist[u]+cost[u,w]
```

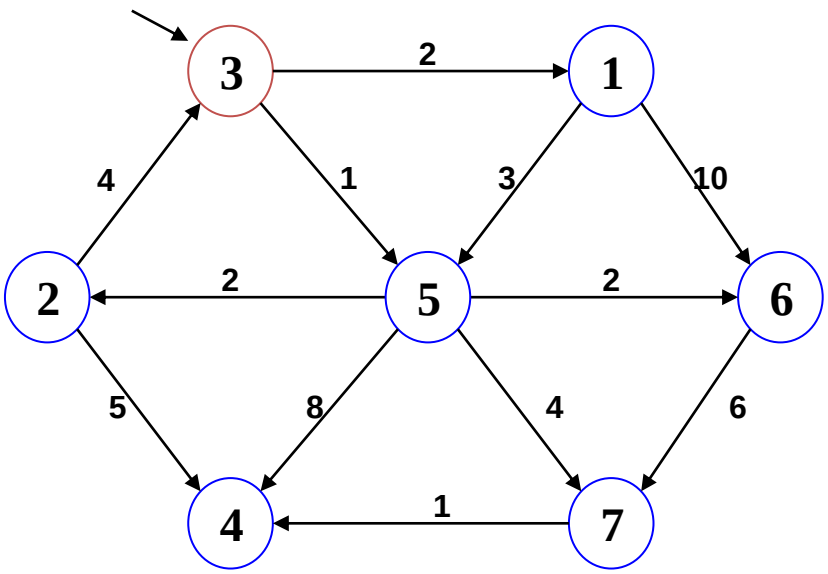



| cost | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | | | | 3 | 10 | |
| 2 | | 0 | 4 | 5 | | | |
| 3 | 2 | | 0 | | 1 | | |
| 4 | | | | 0 | | | |
| 5 | | 2 | | 8 | 0 | 2 | 4 |
| 6 | | | | | | 0 | 6 |
| 7 | | | | 1 | | | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Selected | T | T | T | F | T | T | ~~F~~ |
| dist | 2 | 3 | 0 | ~~8~~ | 1 | 3 | 5 |
| prev | 3 | 5 | | 2 | 3 | 5 | 5 |

**Nearest unselected vertex u=7**

```
for each unselected adjacent vertex w of u(7) do
    if(dist[w]>dist[u]+cost[u,w)) then
        dist[w]:= dist[u]+cost[u,w]
```

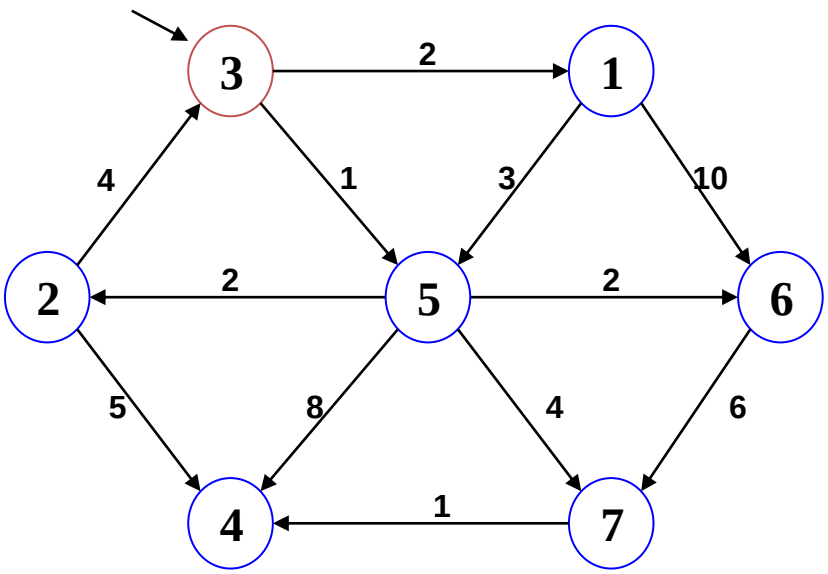



| cost | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | | | | 3 | 10 | |
| 2 | | 0 | 4 | 5 | | | |
| 3 | 2 | | 0 | | 1 | | |
| 4 | | | | 0 | | | |
| 5 | | 2 | | 8 | 0 | 2 | 4 |
| 6 | | | | | | 0 | 6 |
| 7 | | | | 1 | | | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Selected | T | T | T | ~~F~~ | T | T | T |
| dist | 2 | 3 | 0 | 6 | 1 | 3 | 5 |
| prev | 3 | 5 | | 7 | 3 | 5 | 5 |

**Nearest unselected vertex u=4**

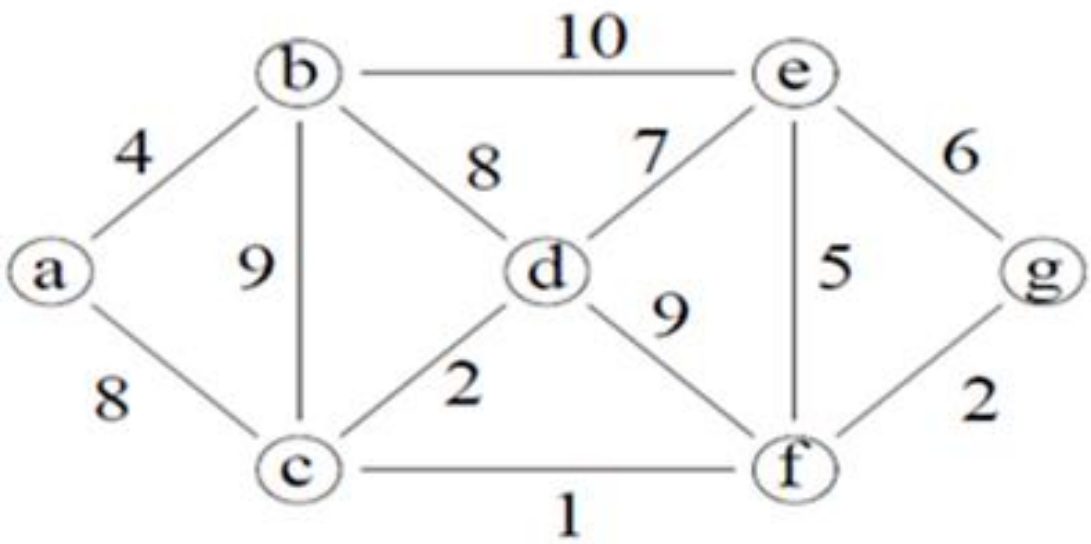
```
for each unselected adjacent vertex w of u(4) do
    if(dist[w]>dist[u]+cost[u,w)) then
        dist[w]:= dist[u]+cost[u,w]
```




| cost | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | | | | 3 | 10 | |
| 2 | | 0 | 4 | 5 | | | |
| 3 | 2 | | 0 | | 1 | | |
| 4 | | | | 0 | | | |
| 5 | | 2 | | 8 | 0 | 2 | 4 |
| 6 | | | | | | 0 | 6 |
| 7 | | | | 1 | | | 0 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Selected | T | T | T | T | T | T | T |
| dist | 2 | 3 | 0 | 6 | 1 | 3 | 5 |
| prev | 3 | 5 | | 7 | 3 | 5 | 5 |

# Dijkstra's Algorithm



```
Algorithm ShortestPaths(v,cost,dist,n)
//n is the number of vertices in the Graph G
//cost is the adjacency matrix of G
//v is the source vertex
//dist[j] 1<=j<=n records the shortest distance of
//vertec j from source vertex v
```

```
{
    for j:= 1 to n do
        S[i]:=false;
    dist[v]:=0;
    for k:= 1 to n do
    {
        choose u such that S[u]=false and dist[u] is   the
        minimum;
        S[u]:=true;
        for each adjacent vertex w of u with s[w]=false do
            if(dist[w]>dist[u]+cost[u,w]) then
                dist[w] := dist[u]+cost[u,w]

    }
}
```

# Dijkstra's Algorithm

- All the vertices are stored in a min-heap based on its **`dist`**.
- Choosing minimum distance unselected vertex takes $\log|V|$ time.
- When we change the value of **`dist`**, each vertex will percolate and it takes $\log|V|$ complexity.

# Dijkstra's Algorithm

- The outer **`for`** loop executes $|V|$ times
- Selecting minimum distance vertex takes $\log|V|$ times with a min-heap.
- The inner for loop over the entire run of algorithm executes $|E|$ times and each time update takes $\log|V|$ time. Hence, the time complexity is
- $O((|V| + |E|)\log|V|)$