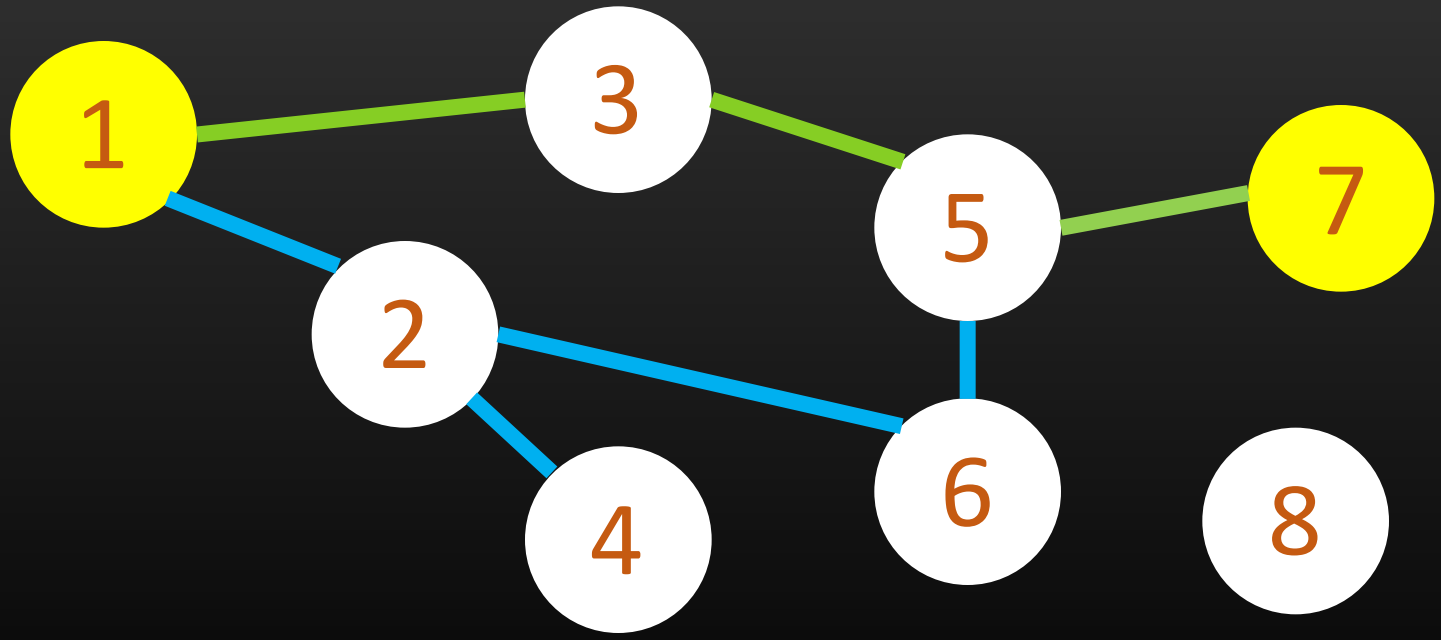
Graph III

Jason

Table of contents

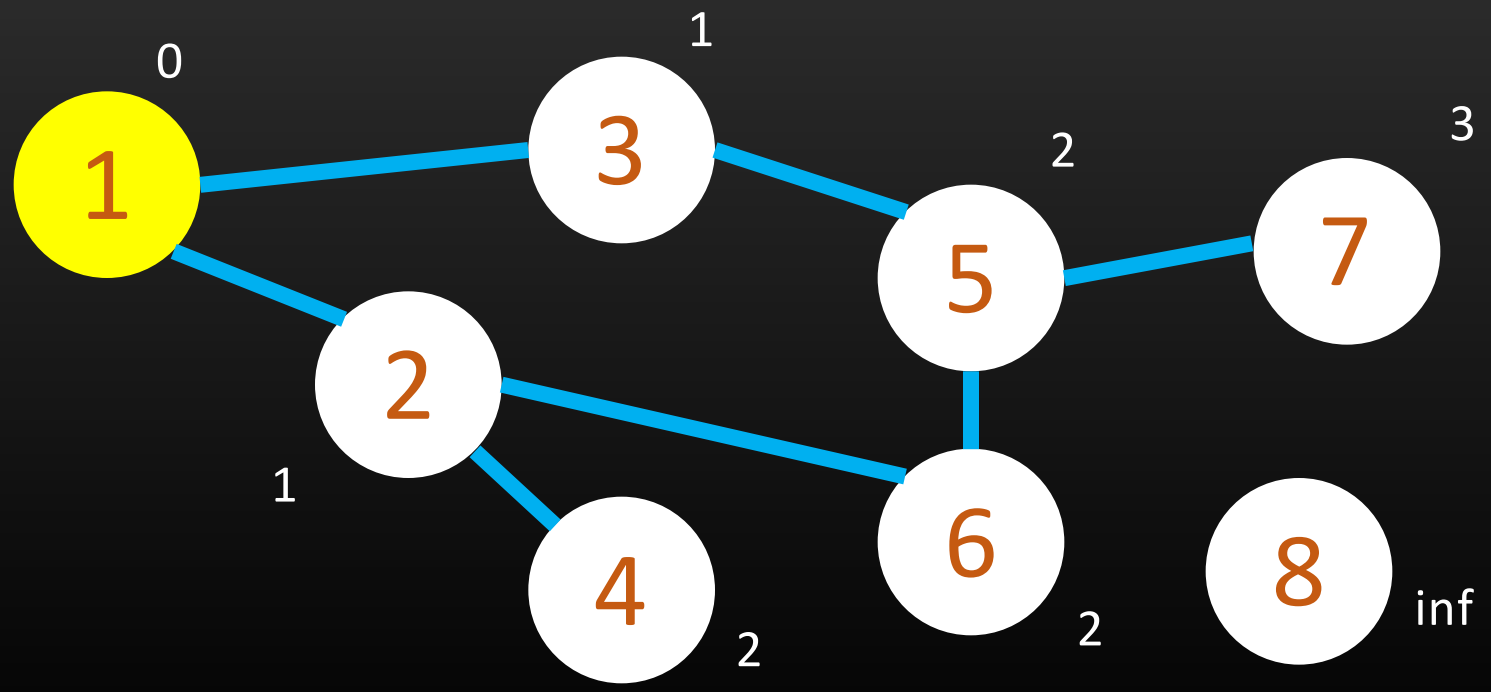
Shortest Path

- Find a path between two nodes in a graph such that the number of the edges in the path is minimized



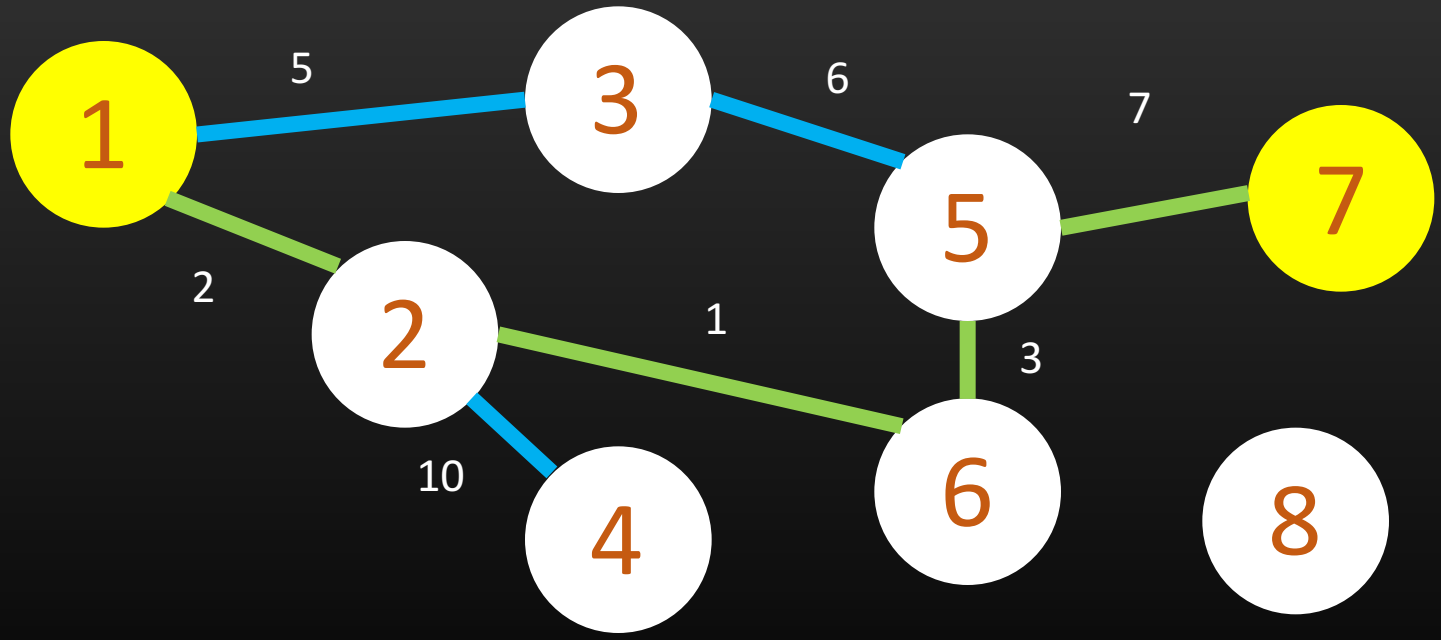
Shortest Path

- Breadth-First Search
  - The order of visiting allows you to find the shortest path of all nodes from a single source



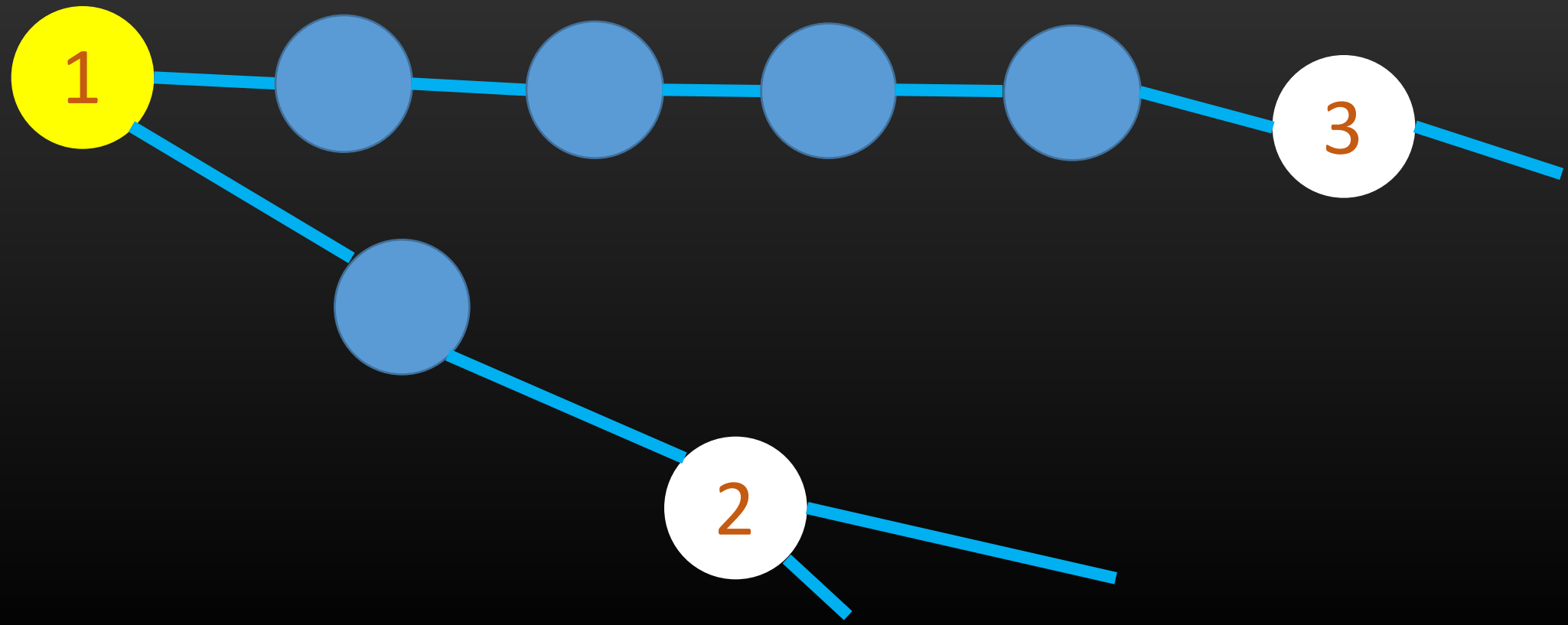
Shortest Path

- Find a path between two nodes in a graph such that the sum of the weights of the edge along the path is minimized



Shortest Path - Attempt

- For an edge uv with weight w, build w edges in series connecting uv



Shortest Path - Attempt

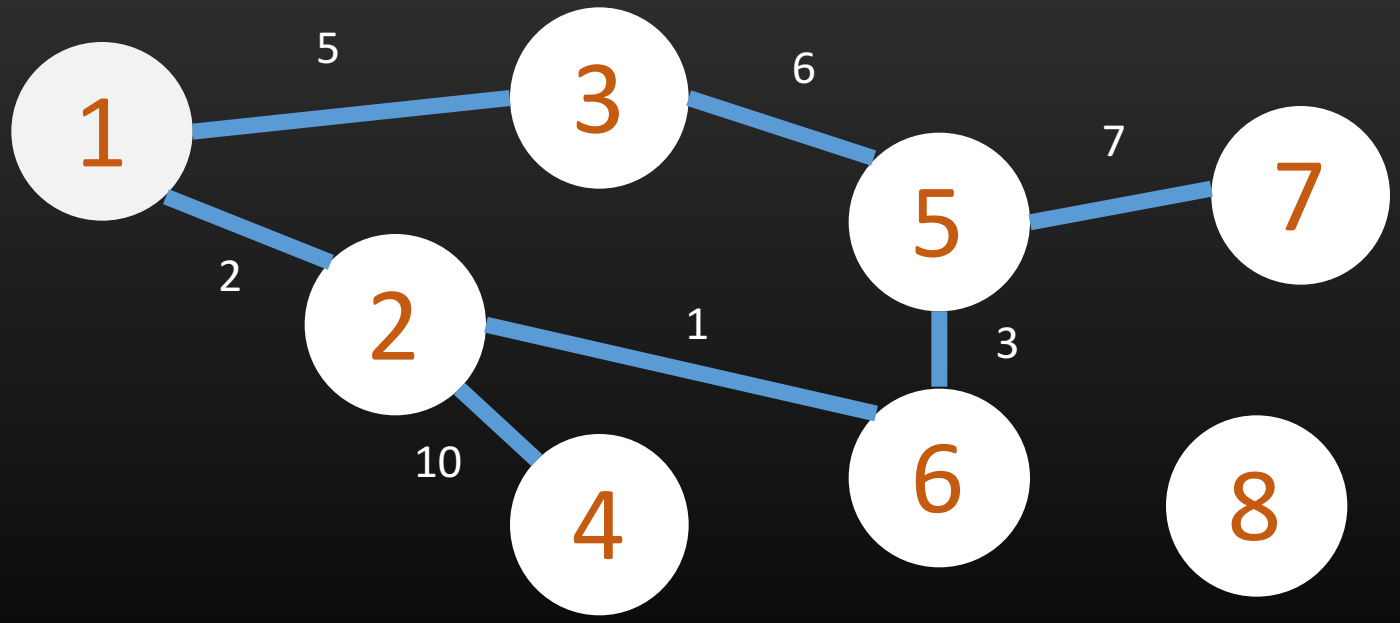
- What if w is large (e.g. 10^9)?
- What if w is not an integer (e.g. 3.1415)?

Dijkstra's algorithm

- Similar to Breadth-First Search
- For each step, pick the node that is not visited and has the shortest distance from the source
- Mark the node as visited and modify its neighbors' distance from the source
- Repeat the process until no new nodes can be picked

Dijkstra’s algorithm - example

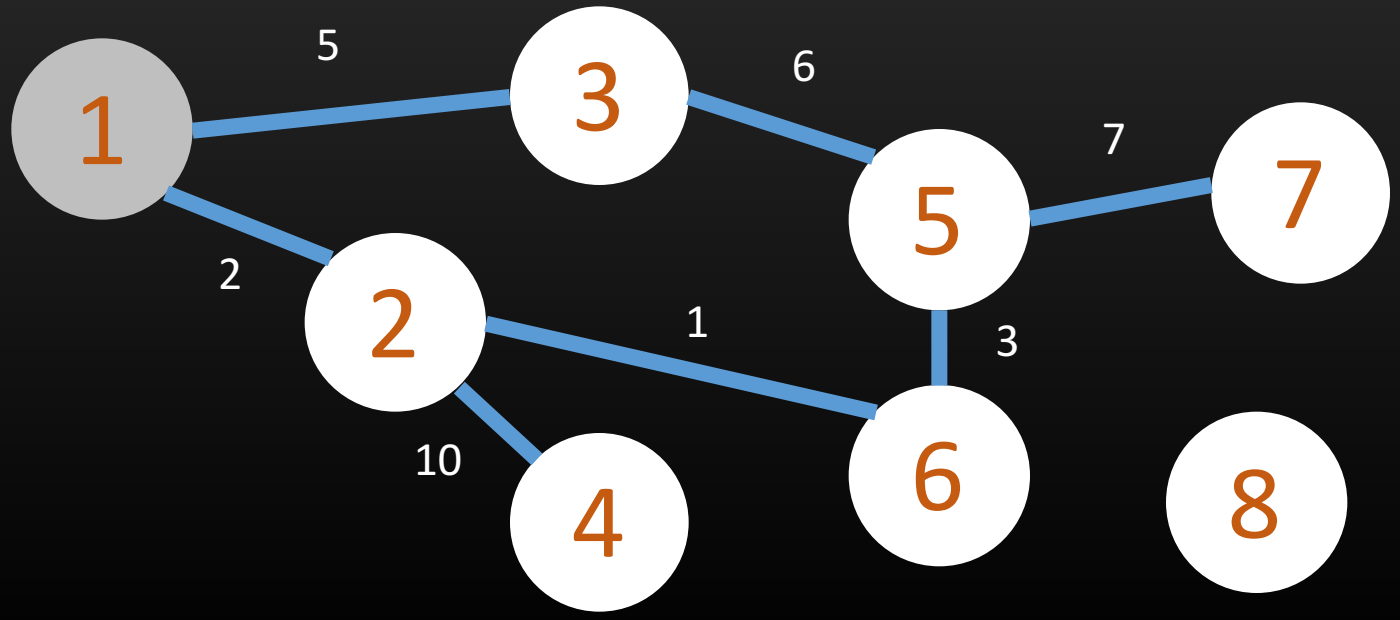
- Find the shortest paths from node 1



Dijkstra’s algorithm - example

- Set the distance of node 1 as 0, and set the distance of other nodes as infinity

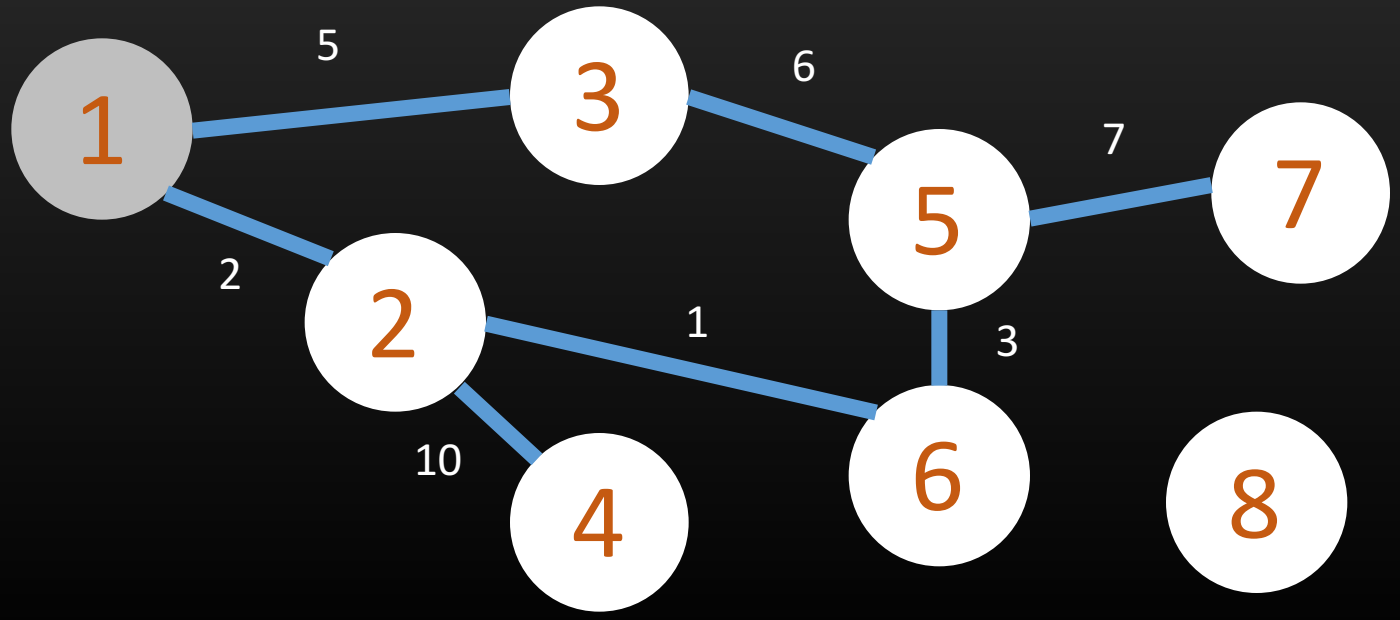
Node	1	2	3	4	5	6	7	8
Distance	0	∞	∞	∞	∞	∞	∞	∞



Dijkstra's algorithm - example

- Mark node 1 as visited and modify the distance of node 1's neighbors (node 2, 3) using the distance of node 1

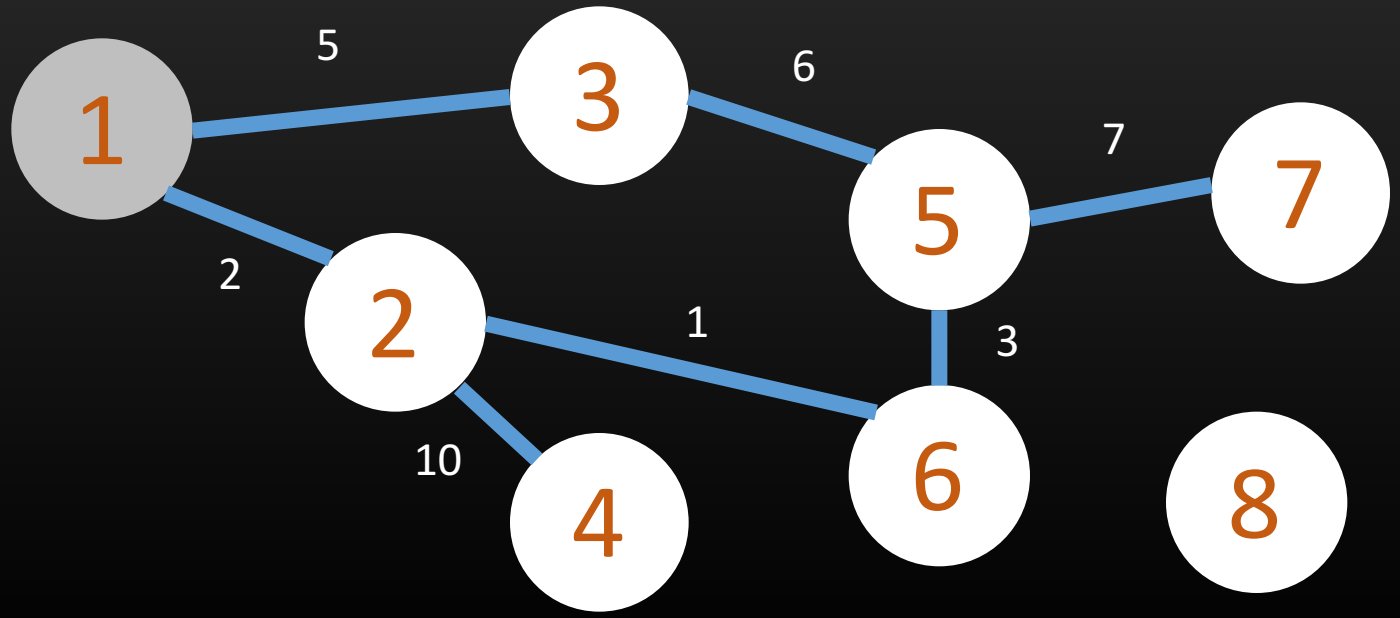
Node	1	2	3	4	5	6	7	8
Distance	0	∞	∞	∞	∞	∞	∞	∞



Dijkstra’s algorithm - example

- Calculate the new distance of node 2 (distance[1]+the weight of edge 1-2), which is 2

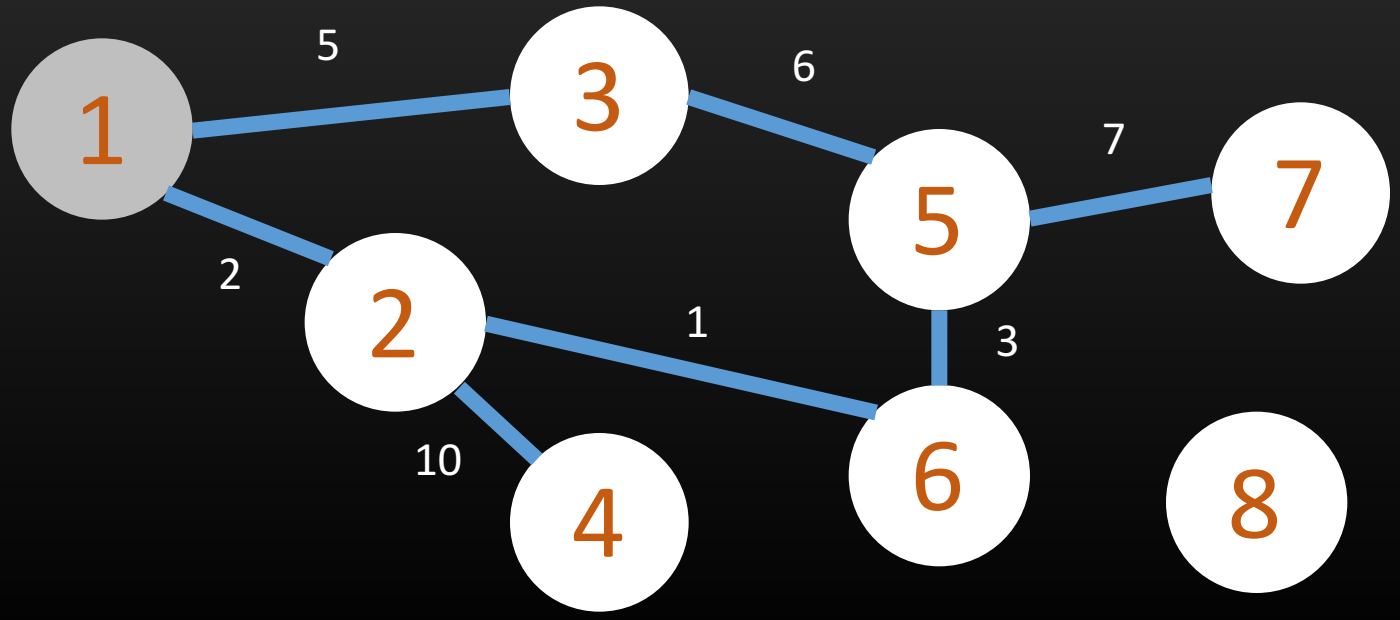
Node	1	2	3	4	5	6	7	8
Distance	0	∞	∞	∞	∞	∞	∞	∞



Dijkstra’s algorithm - example

- Compare the new distance to the current distance of node 2 and assign the smaller one to the current one (distance[2] = min(2, ∞)=2)

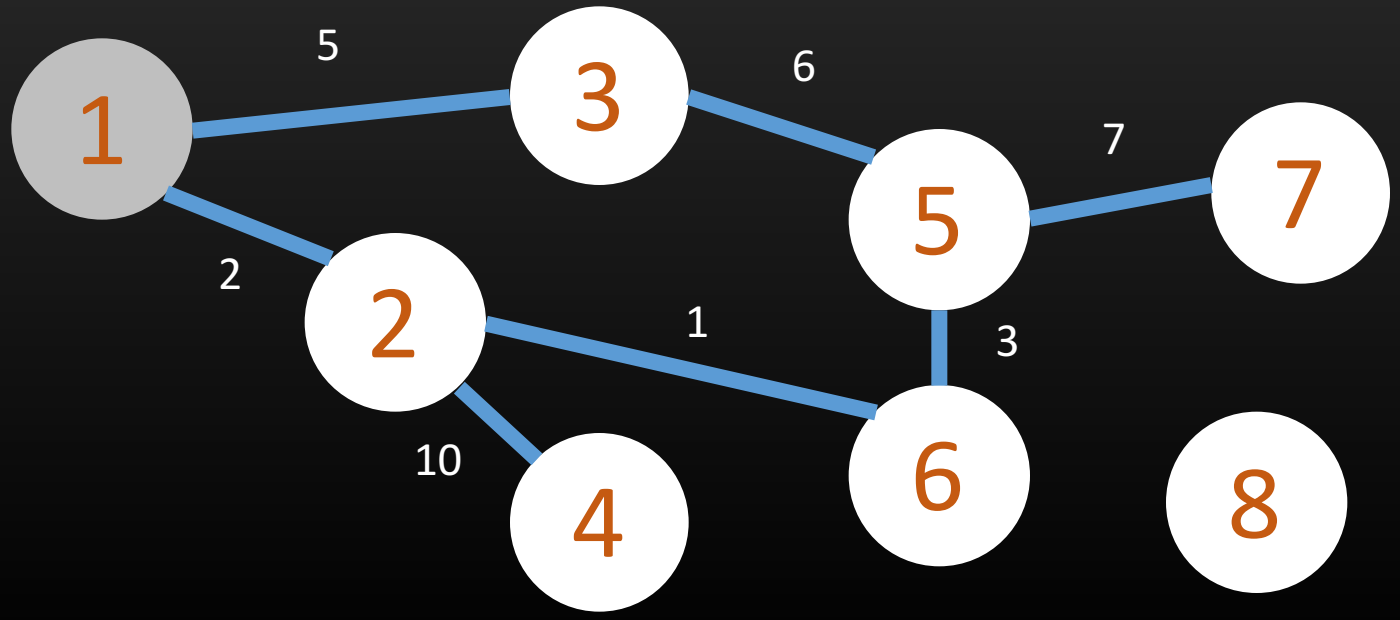
Node	1	2	3	4	5	6	7	8
Distance	0	2	∞	∞	∞	∞	∞	∞



Dijkstra’s algorithm - example

- Same for node 3

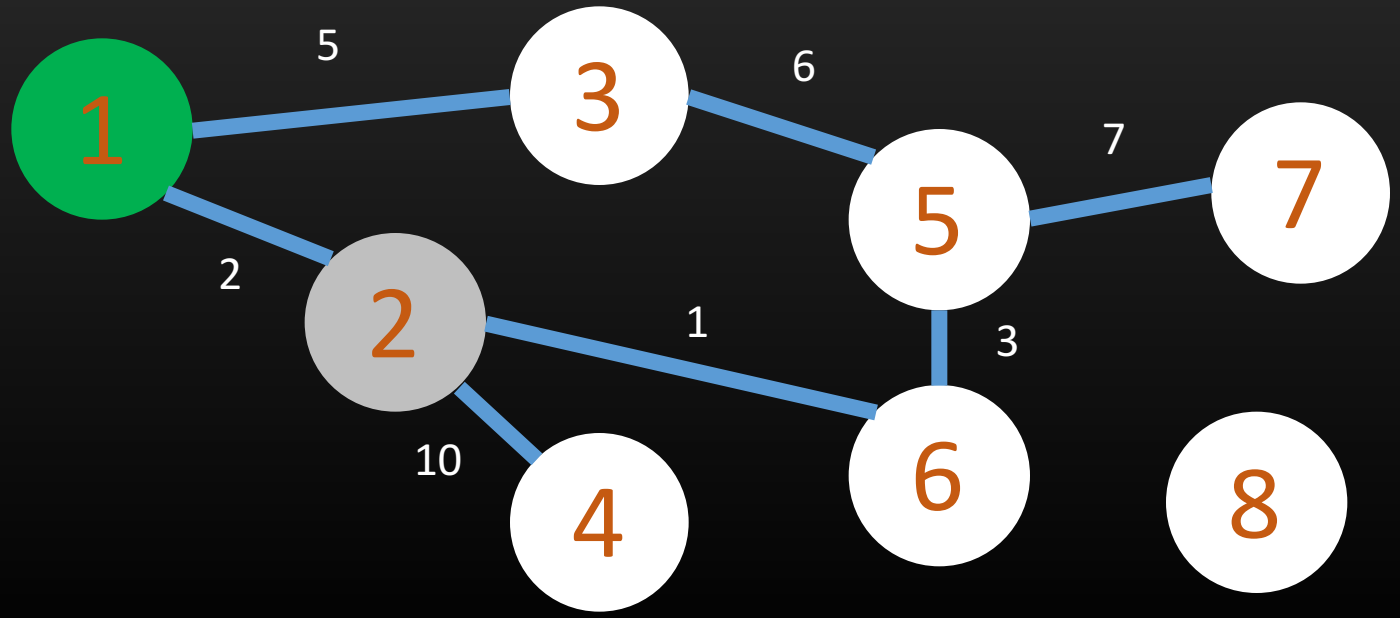
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	∞	∞	∞	∞	∞



Dijkstra's algorithm - example

- Select the unvisited node that is marked with the smallest distance, which is node 2

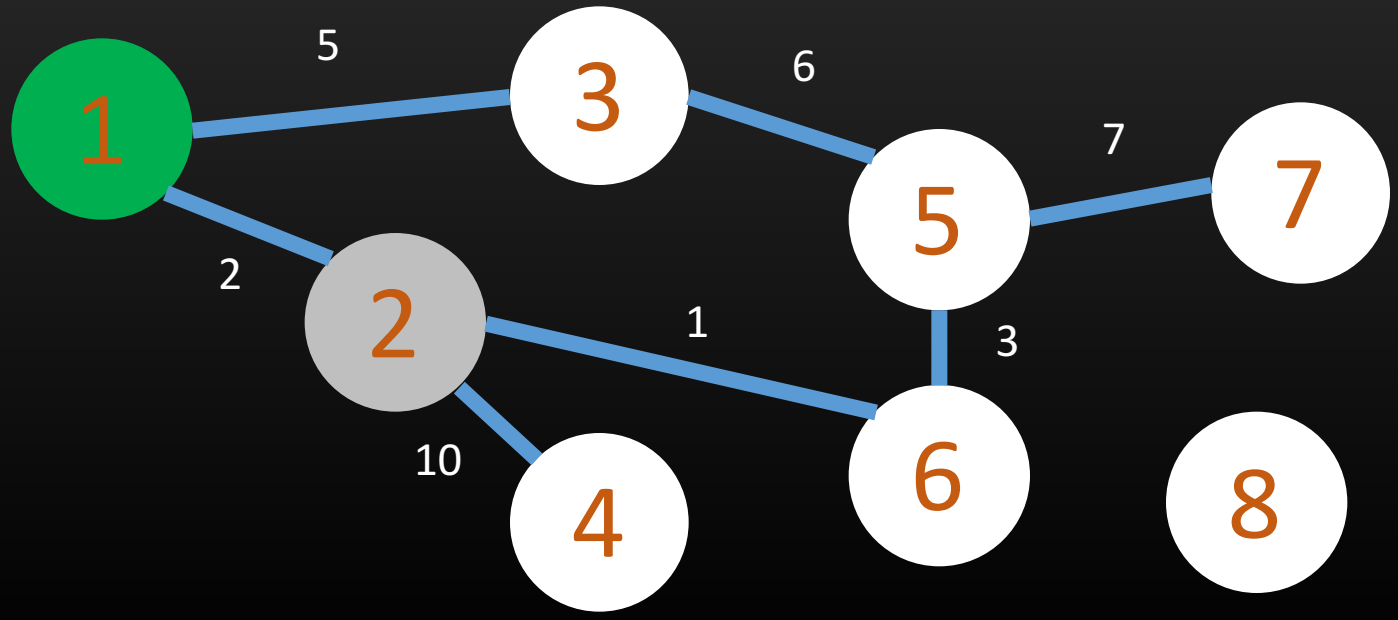
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	∞	∞	∞	∞	∞



Dijkstra's algorithm - example

- Mark node 2 as visited and modify the distance of node 2's neighbors (node 1, 4 and 6) using the distance of node 2

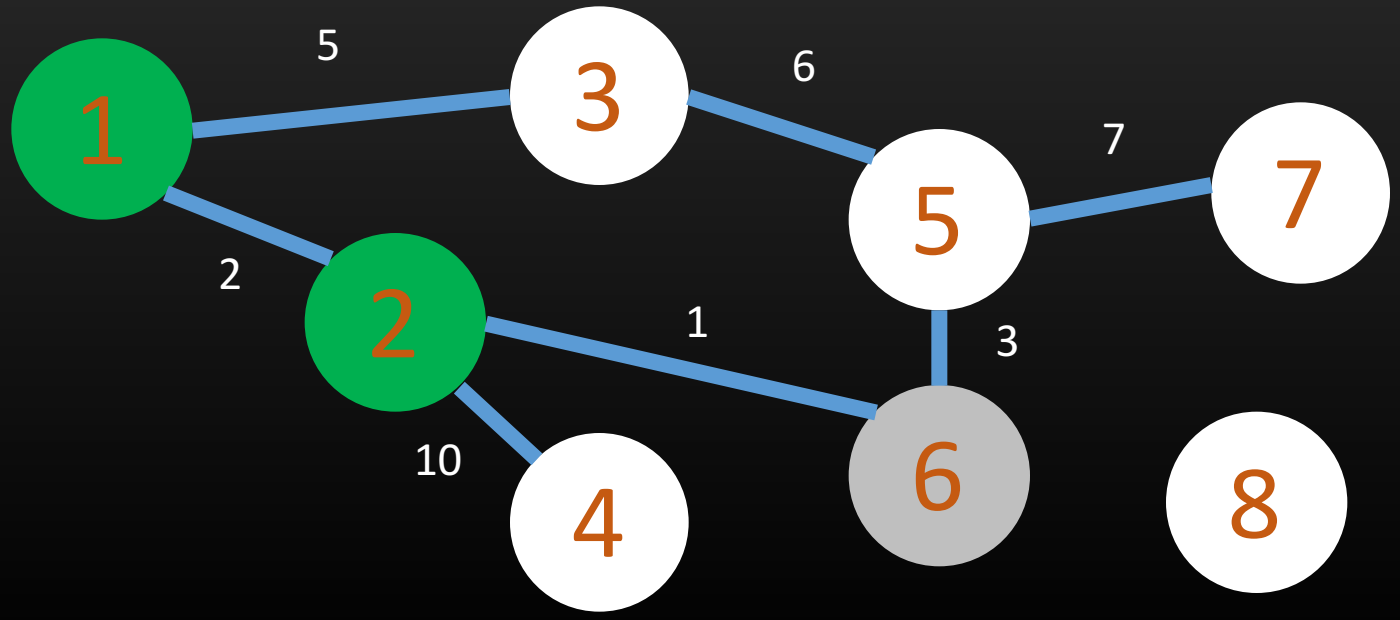
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	∞	3	∞	∞



Dijkstra's algorithm - example

- Select the unvisited node that is marked with the smallest distance, which is node 6

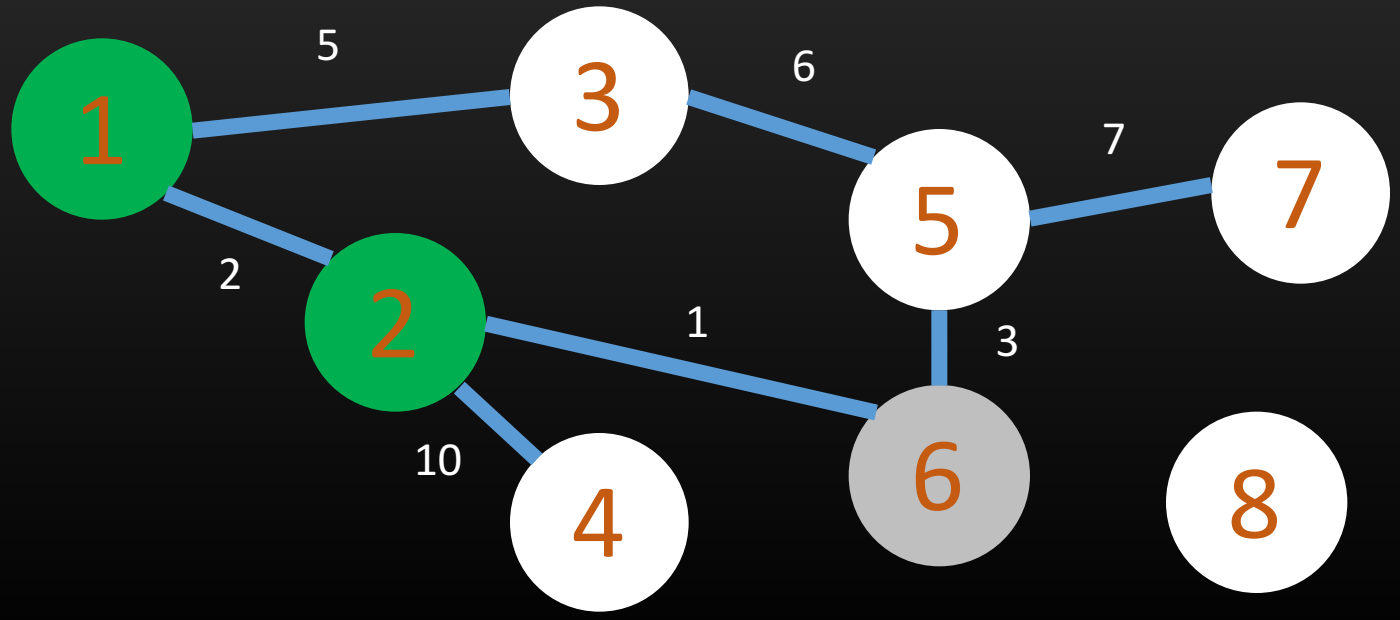
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	∞	3	∞	∞



Dijkstra’s algorithm - example

- Mark node 6 as visited and modify the distance of node 6’s neighbors (node 2 and 5) using the distance of node 6

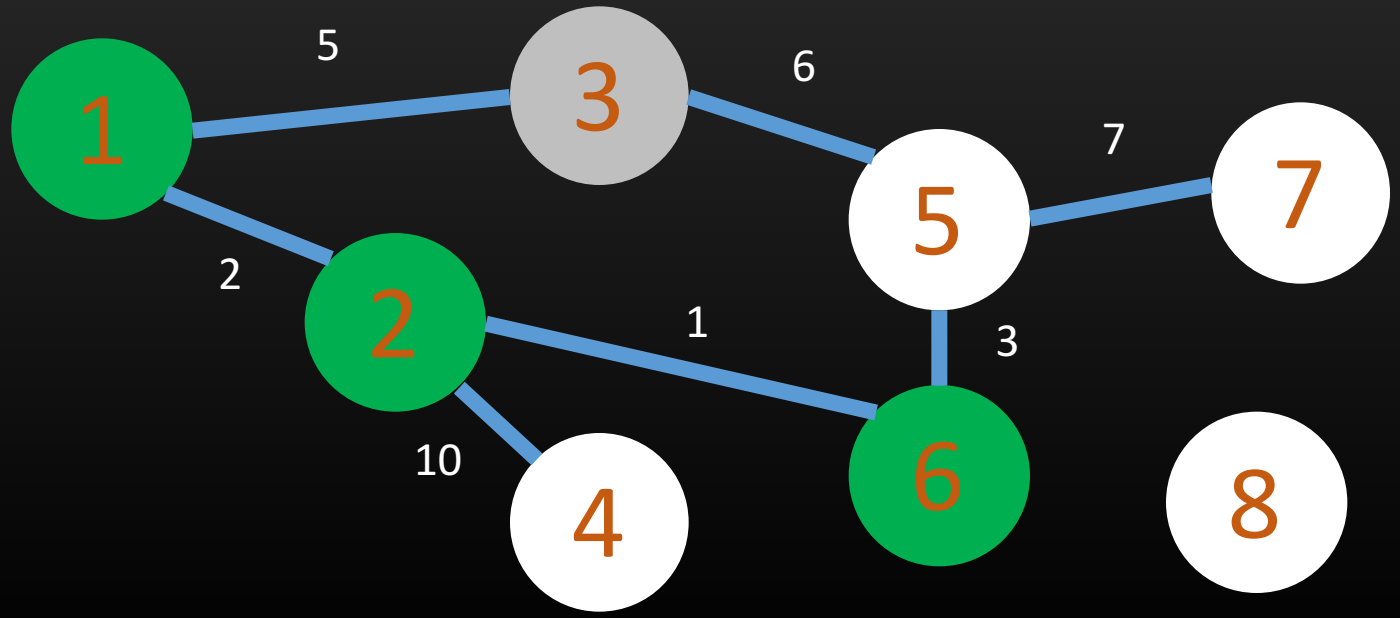
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	∞	∞



Dijkstra’s algorithm - example

- … and so on

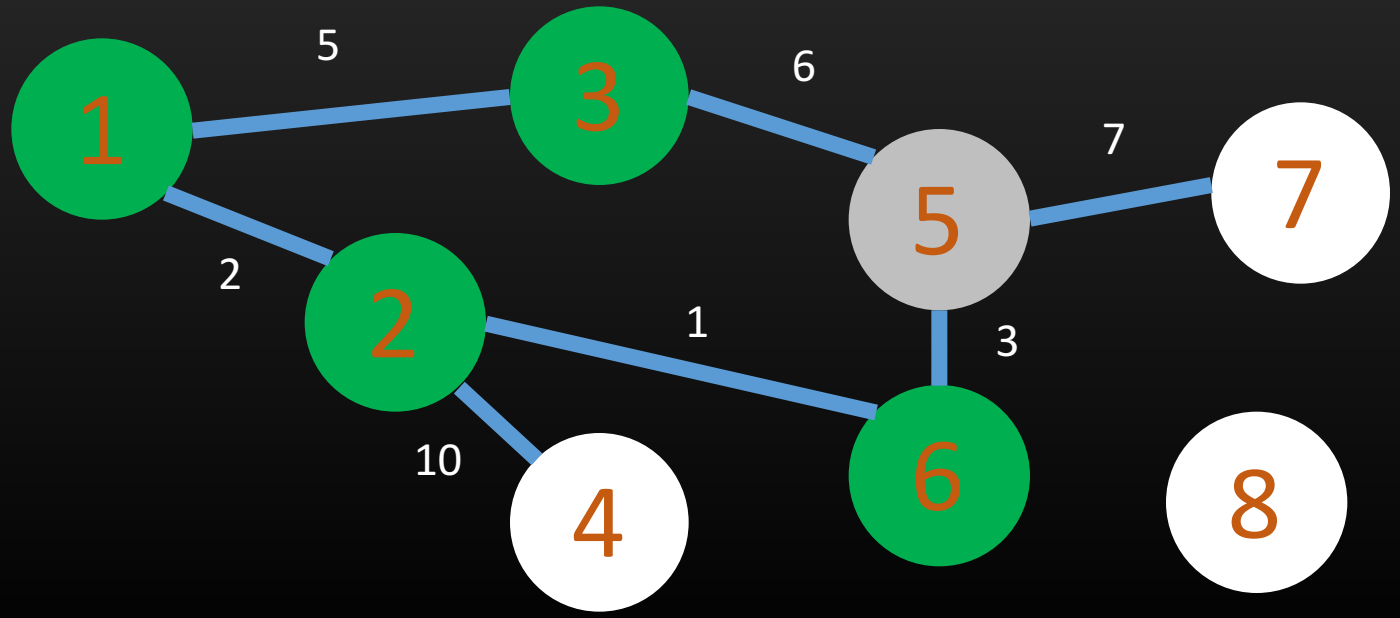
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	∞	∞



Dijkstra’s algorithm - example

- … and so on

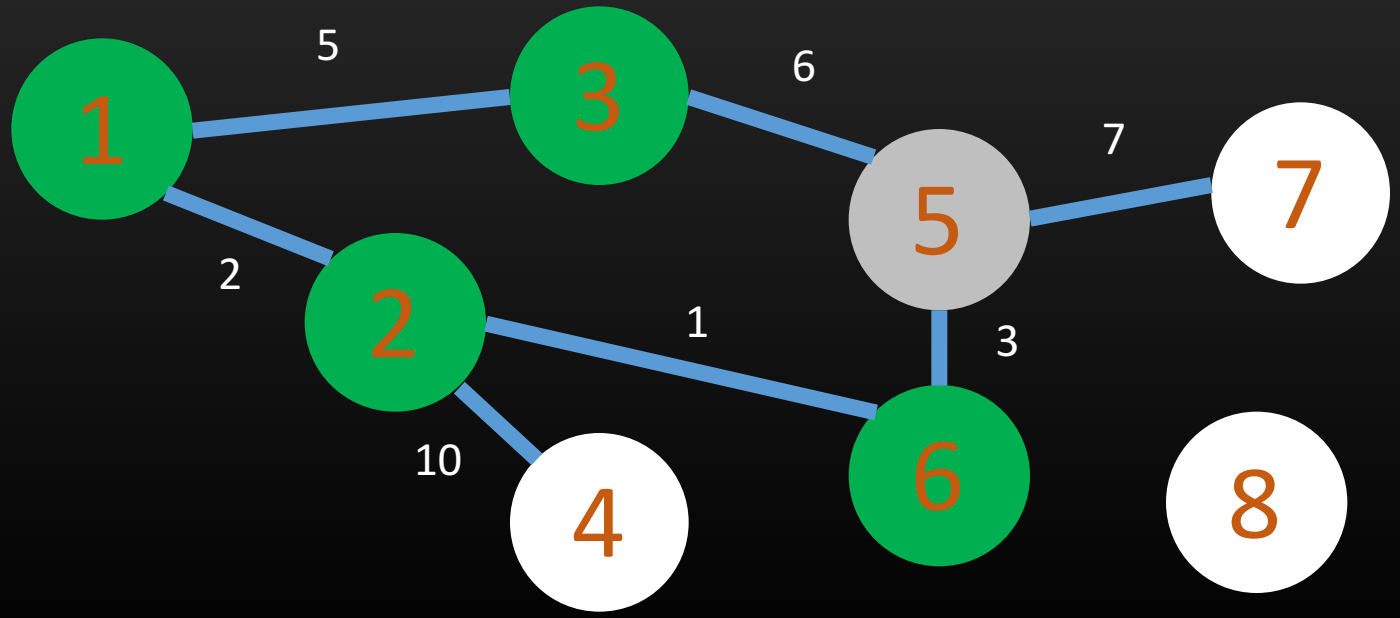
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	∞	∞



Dijkstra’s algorithm - example

- … and so on

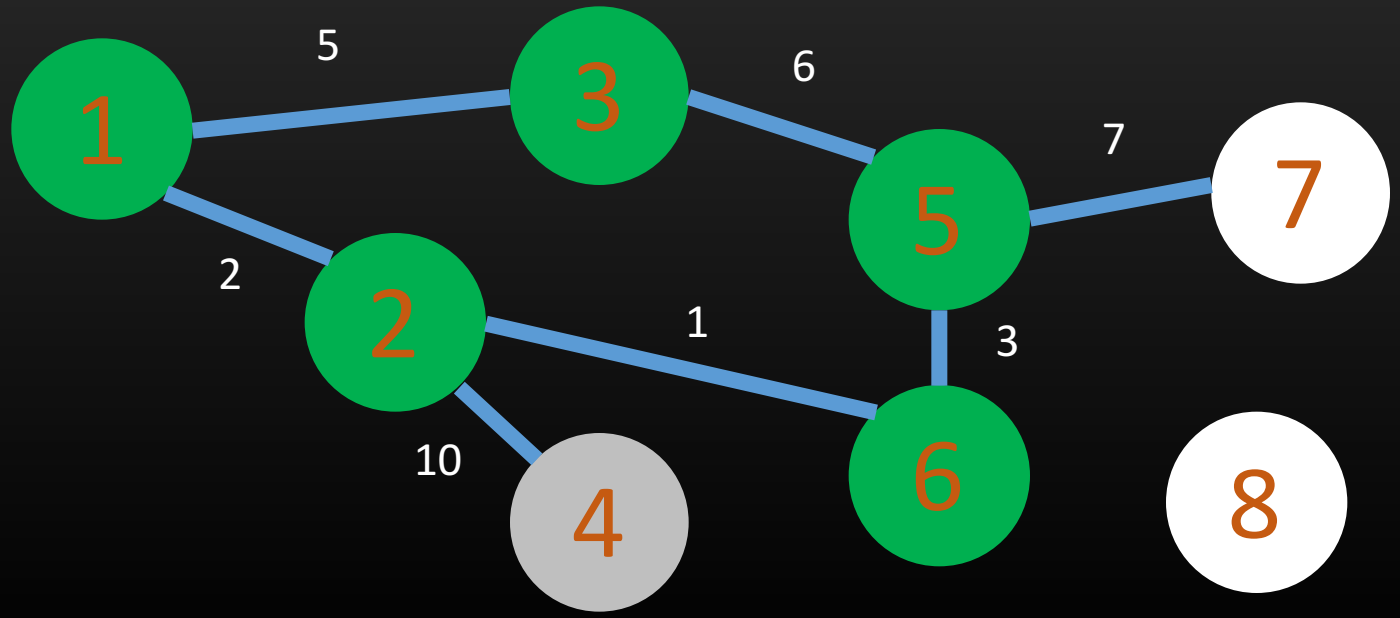
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞



Dijkstra’s algorithm - example

- … and so on

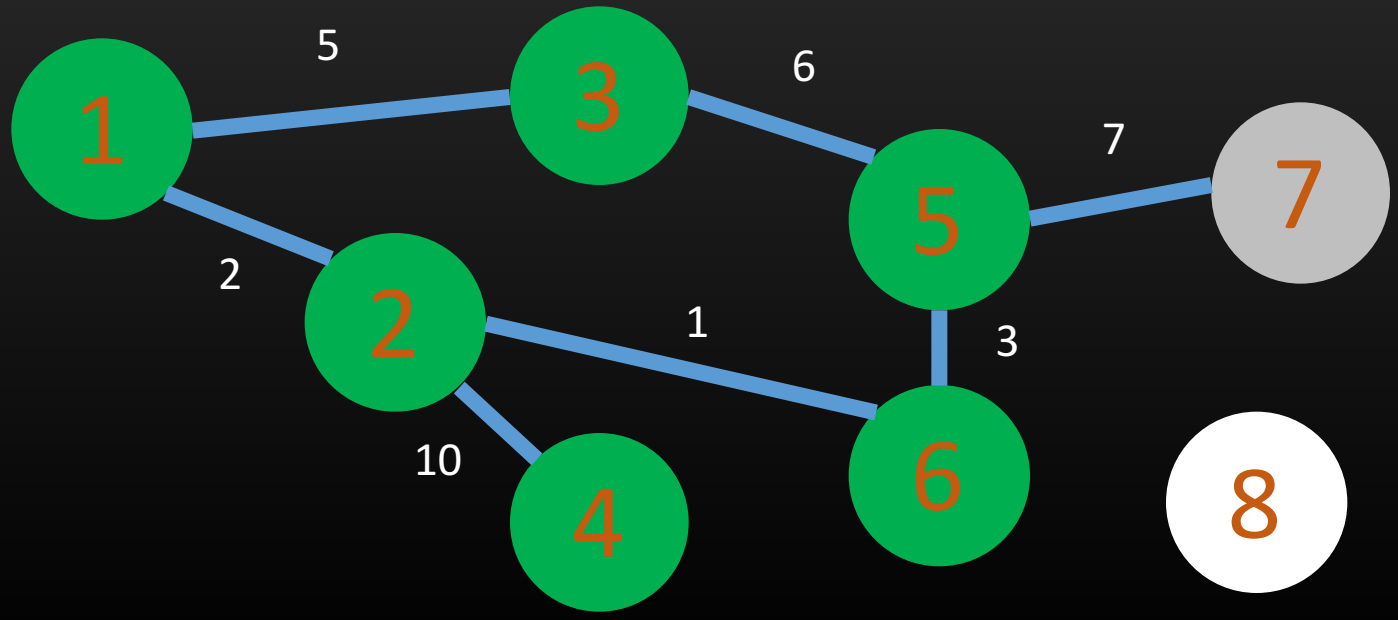
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞



Dijkstra’s algorithm - example

- … and so on

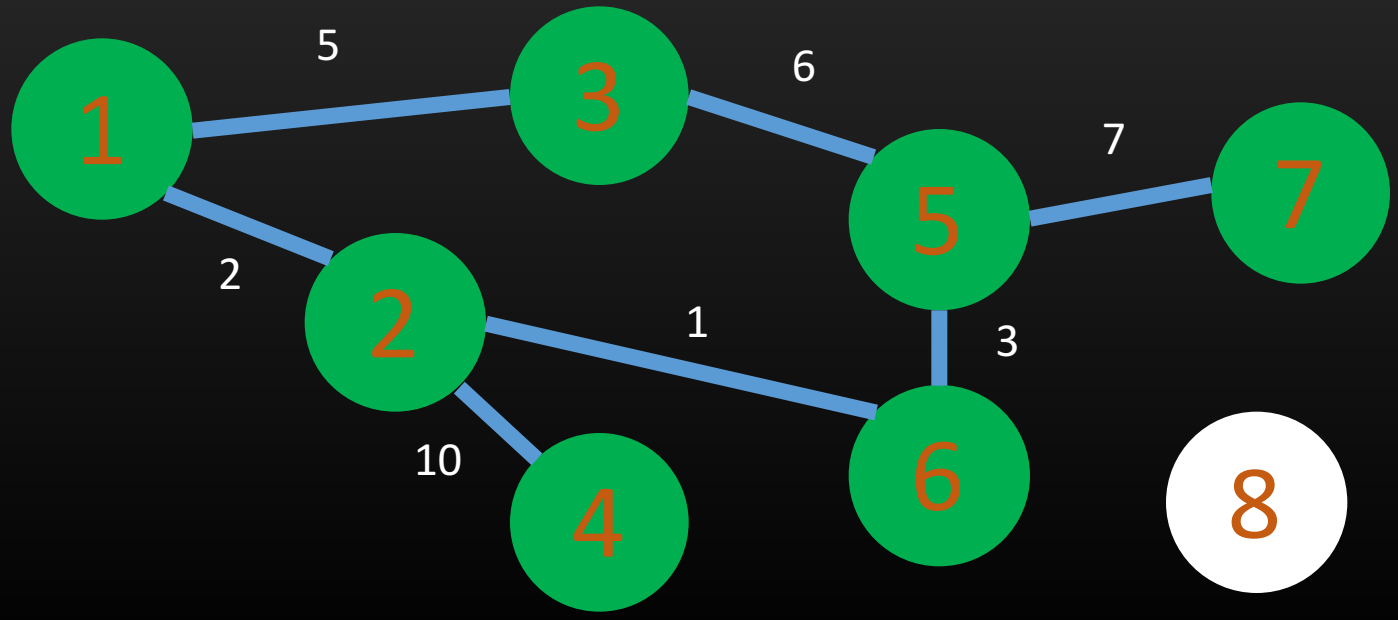
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞



Dijkstra’s algorithm - example

- Done

Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞



Dijkstra's algorithm-pseudocode

```
Procedure Dijkstra(){
    for each vertex v in the graph
        dist[v] = infinity
        visited[v] = false
    dist[source] = 0
    u = source
    do{
        visited[u] = true
        for each neighbor v of u do
            if dist[u]+weight of the edge u-v<dist[v] then
                dist[v]=dist[u]+weight of the edge u-v
        u = vertex that is not visited and has the minimum dist[u]
    }while dist[u] != infinity
}
```

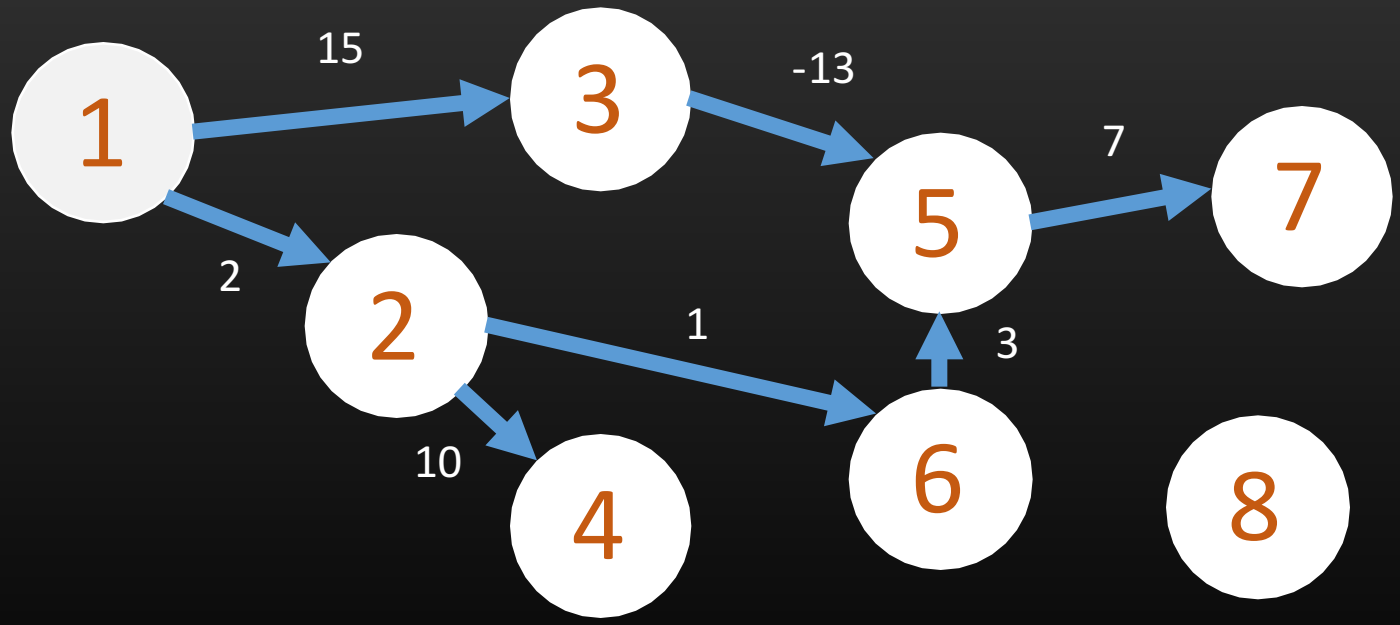
Dijkstra's algorithm

- Time complexity: $O(|V|^2+|E|)$
- We used $O(|V|)$ time when we find the unvisited node that is marked with the smallest distance, which makes it slow

- We can use a min-heap to accelerate the process
- Push the node into the heap when its distance is modified
- Time complexity: $O(|E| \log |E|)$

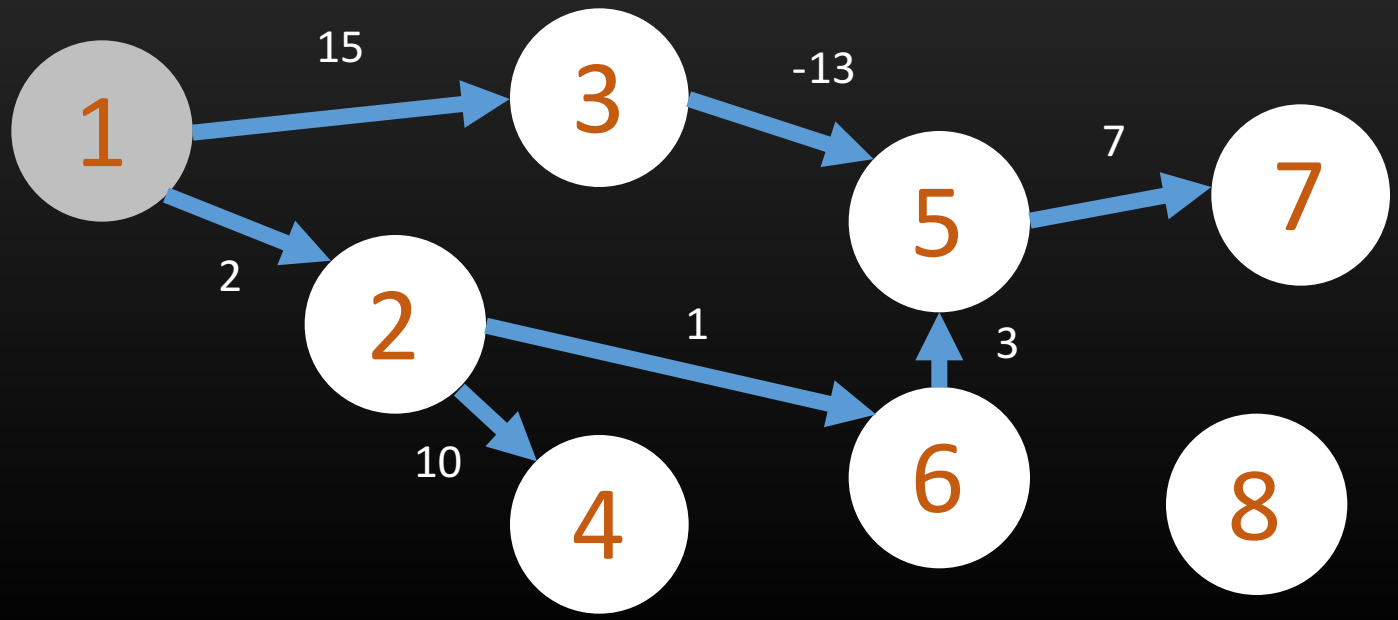
Dijkstra's algorithm – Limitation

- What if the weight of edges are negative?



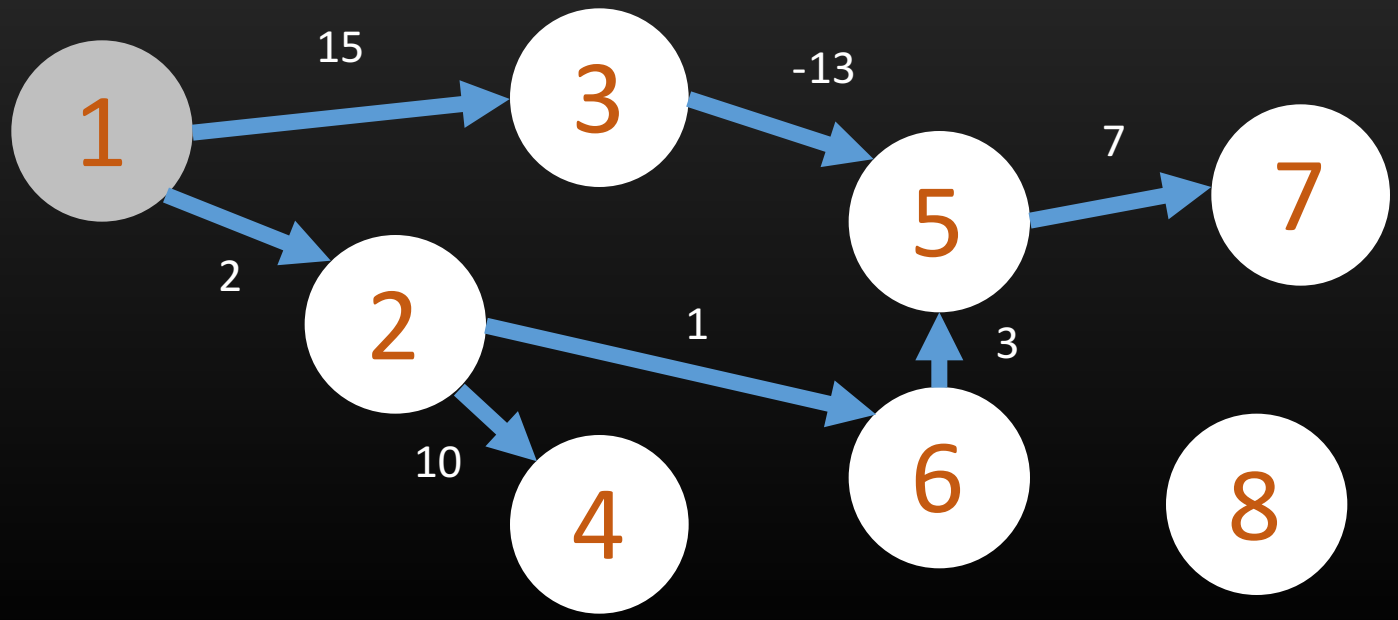
Dijkstra's algorithm – Limitation

Node	1	2	3	4	5	6	7	8
Distance	0	∞	∞	∞	∞	∞	∞	∞



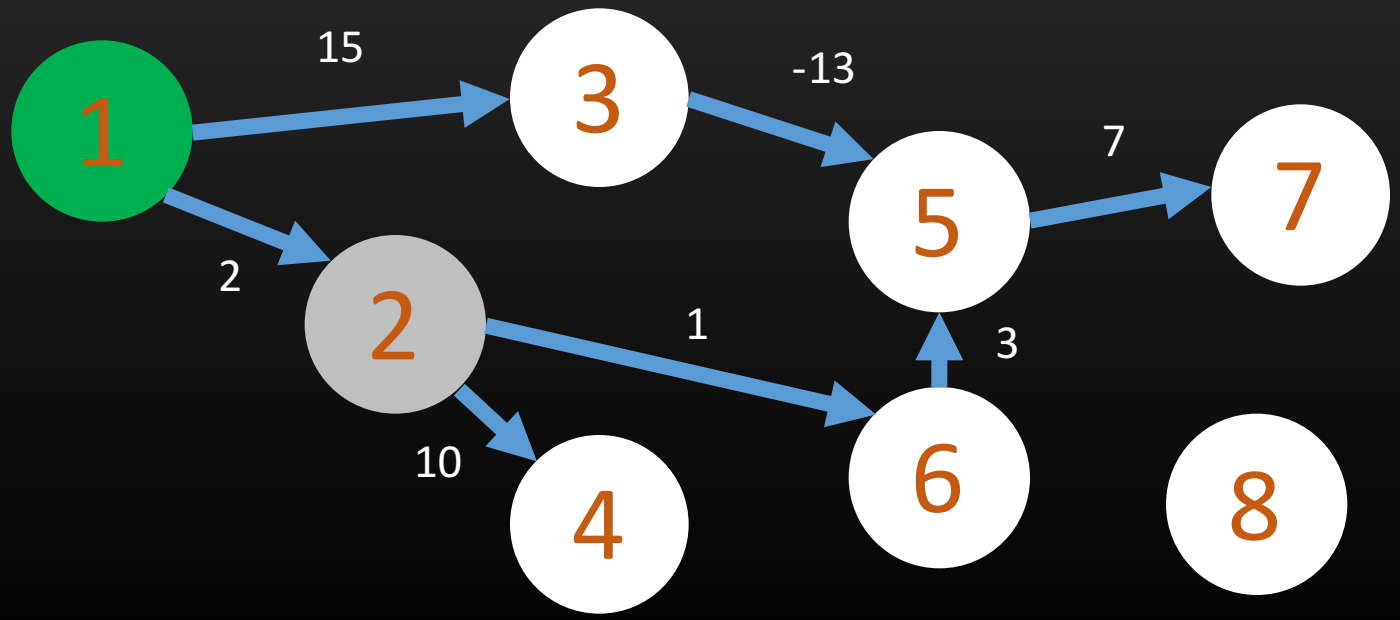
Dijkstra's algorithm – Limitation

Node	1	2	3	4	5	6	7	8
Distance	0	2	15	∞	∞	∞	∞	∞



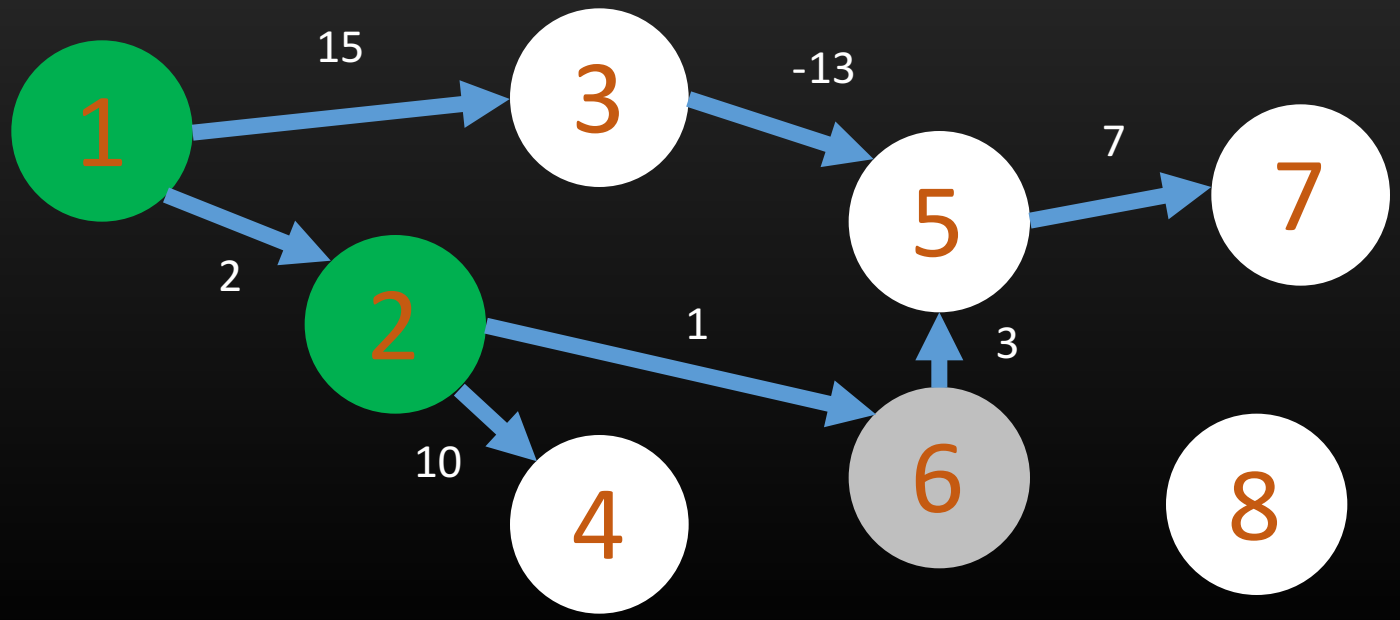
Dijkstra's algorithm – Limitation

Node	1	2	3	4	5	6	7	8
Distance	0	2	15	12	∞	3	∞	∞



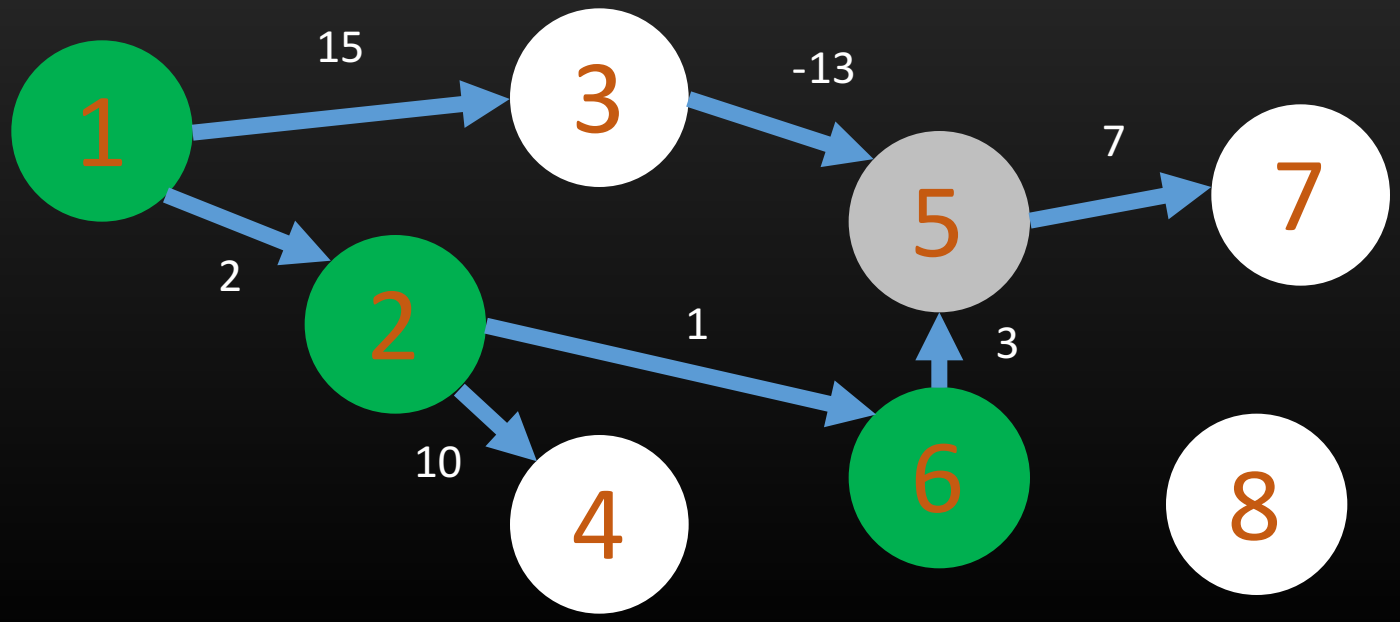
Dijkstra's algorithm – Limitation

Node	1	2	3	4	5	6	7	8
Distance	0	2	15	12	6	3	∞	∞



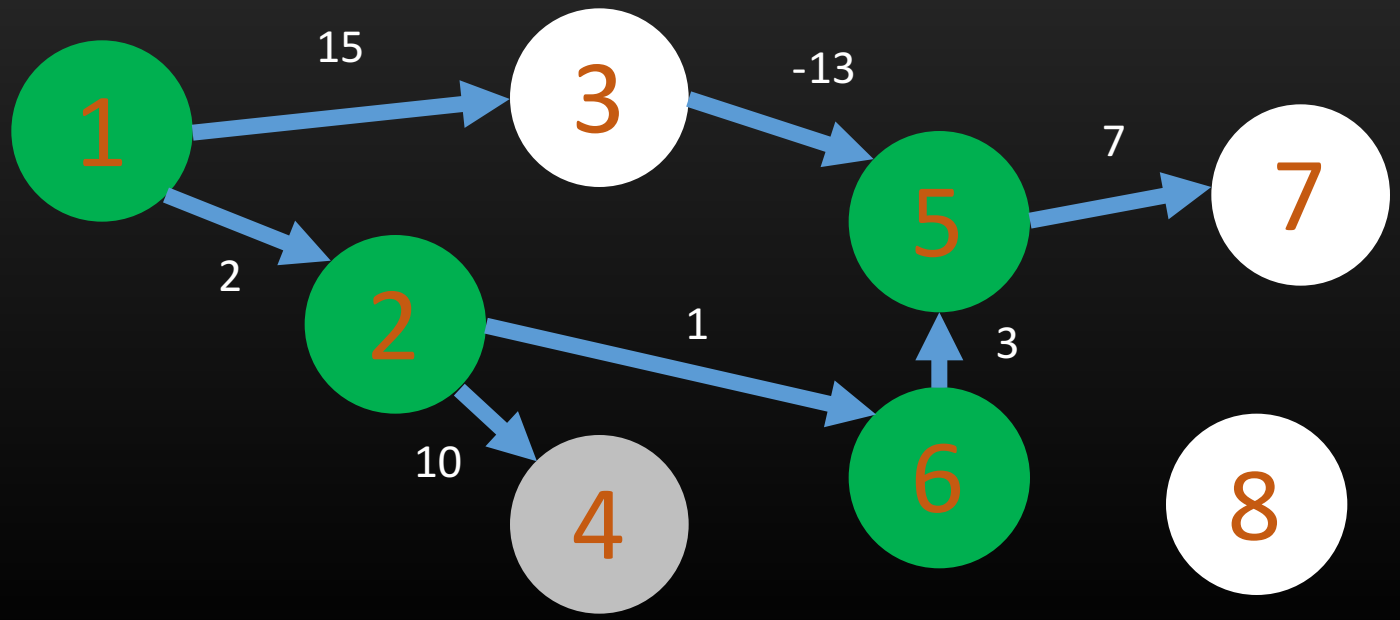
Dijkstra's algorithm – Limitation

Node	1	2	3	4	5	6	7	8
Distance	0	2	15	12	6	3	13	∞



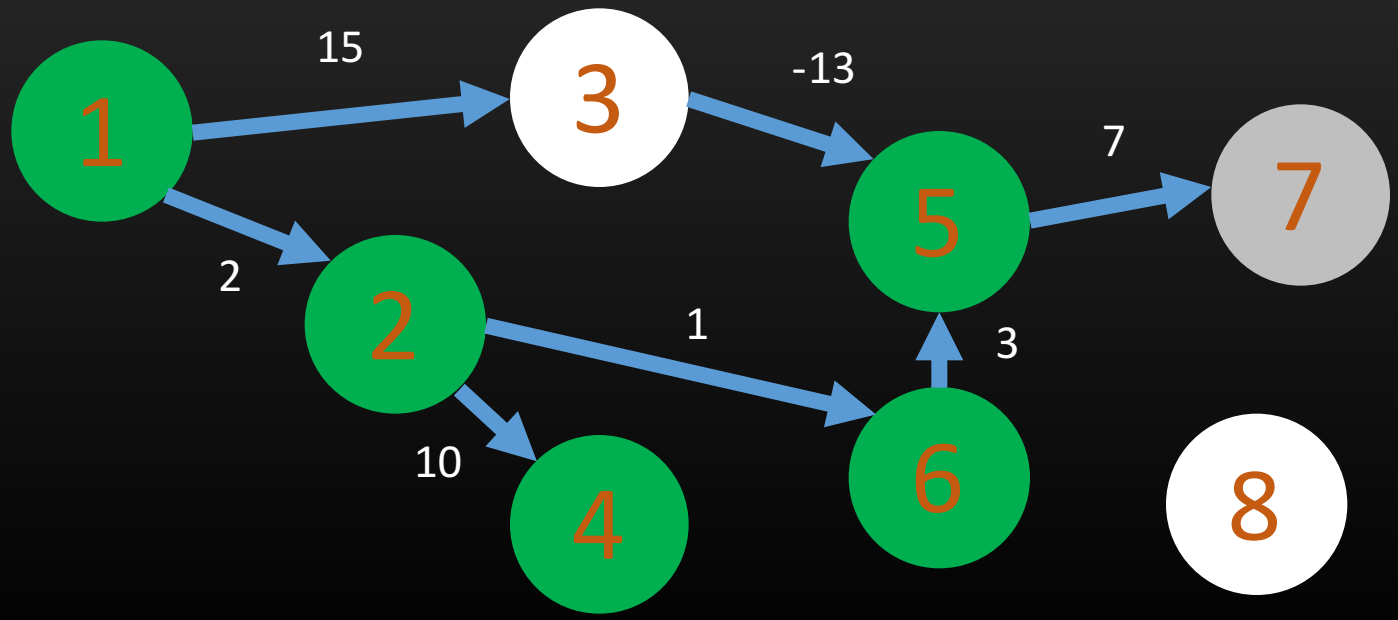
Dijkstra's algorithm – Limitation

Node	1	2	3	4	5	6	7	8
Distance	0	2	15	12	6	3	13	∞



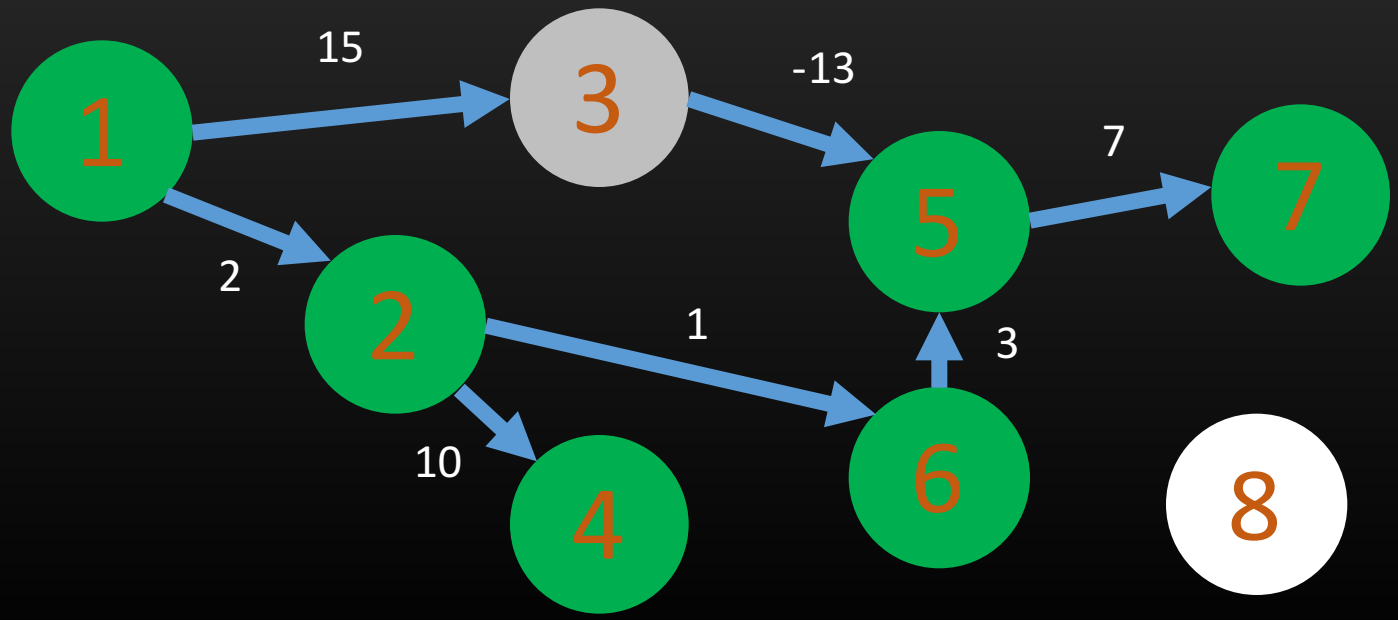
Dijkstra's algorithm – Limitation

Node	1	2	3	4	5	6	7	8
Distance	0	2	15	12	6	3	13	∞



Dijkstra's algorithm – Limitation

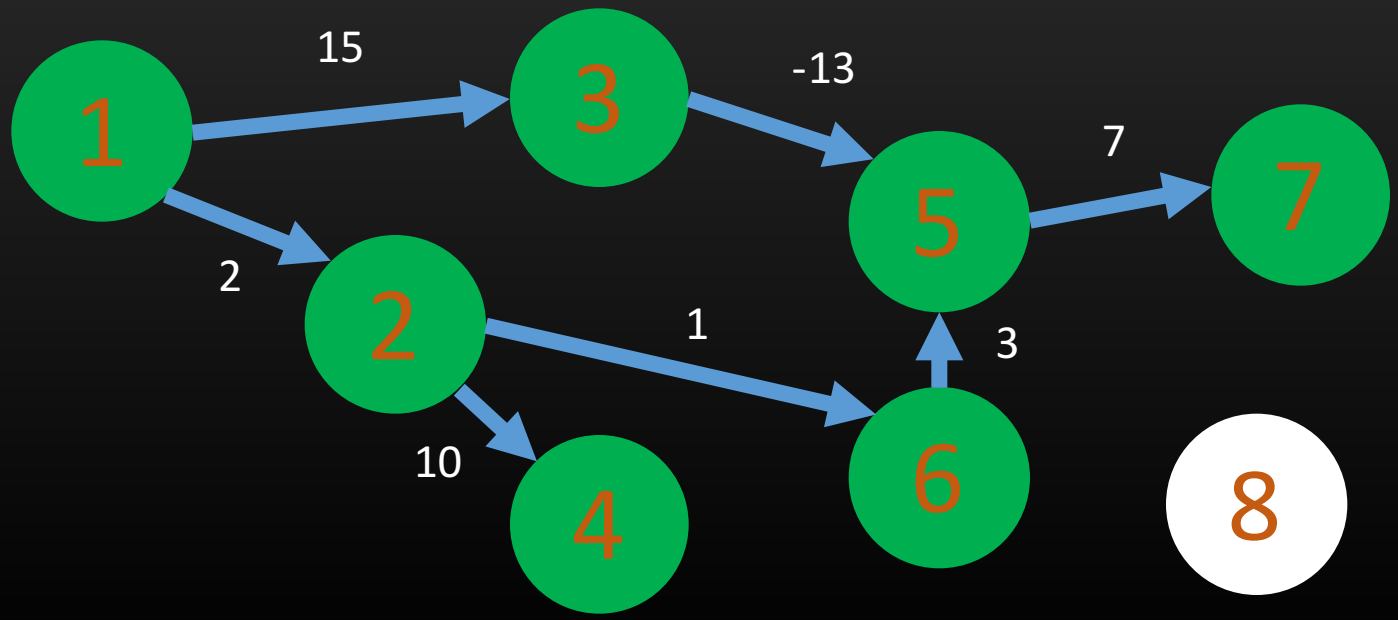
Node	1	2	3	4	5	6	7	8
Distance	0	2	15	12	2	3	13	∞



Dijkstra's algorithm – Limitation

????????

Node	1	2	3	4	5	6	7	8
Distance	0	2	15	12	2	3	13	∞



Bellman-ford algorithm

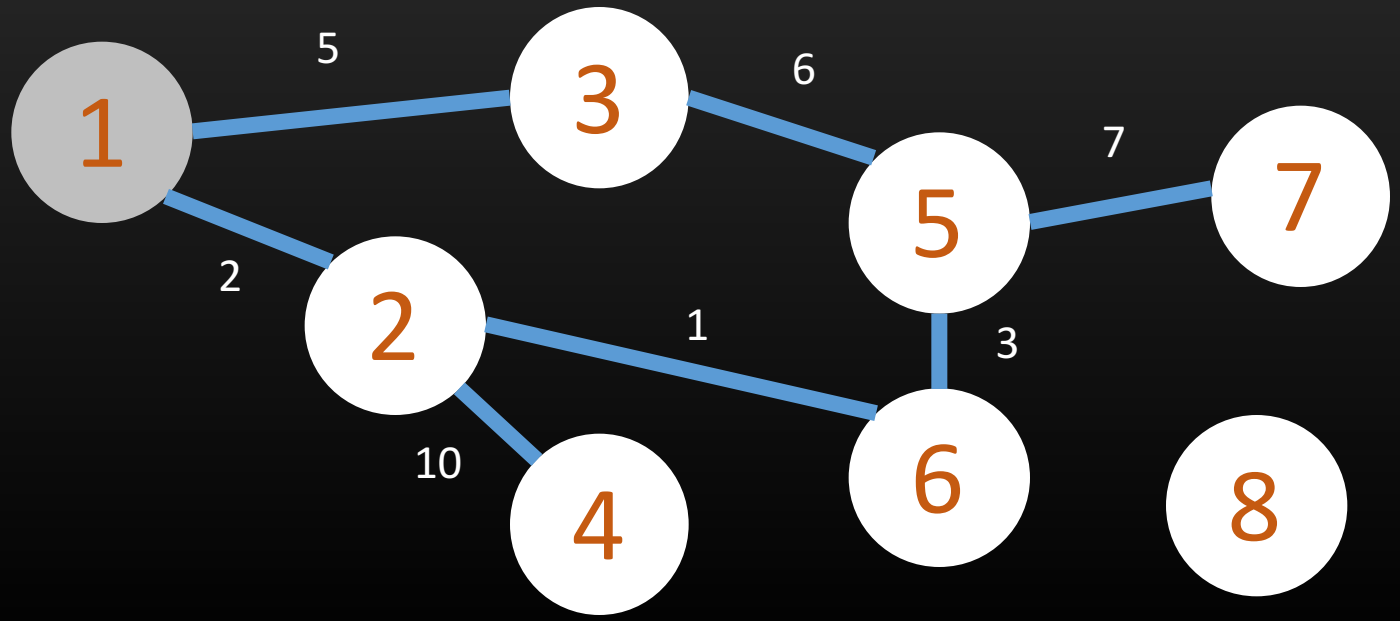
- Assume there are no negative cycles
- For a shortest path, it makes no sense to revisit nodes
- Each shortest path must contain not more than (V – 1) steps

- Therefore, we can find all shortest paths by repeating the following (|V|-1) times:
  - For every edge u-v
    - dist[v] = min(dist[v] , dist[u] + weight[u-v])

Bellman-ford algorithm - example

- Set the distance of node 1 as 0, and set the distance of other nodes as infinity

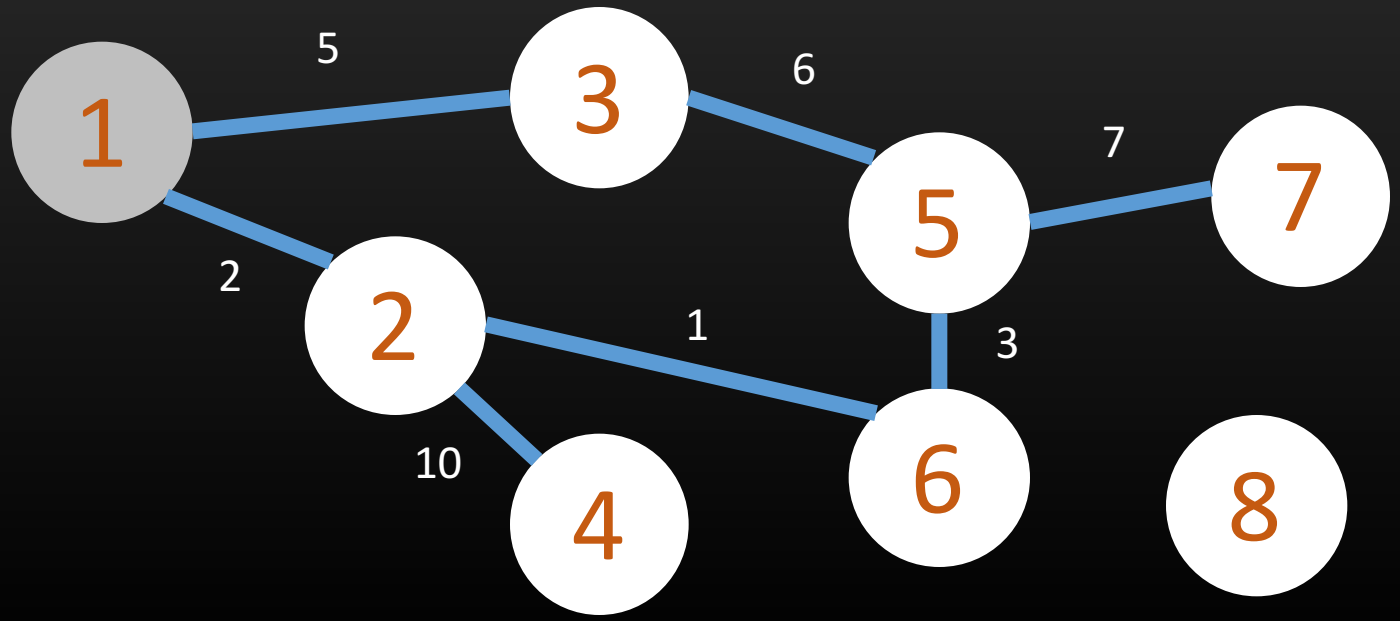
Node	1	2	3	4	5	6	7	8
Distance	0	∞	∞	∞	∞	∞	∞	∞



Bellman-ford algorithm - example

- 1st iteration

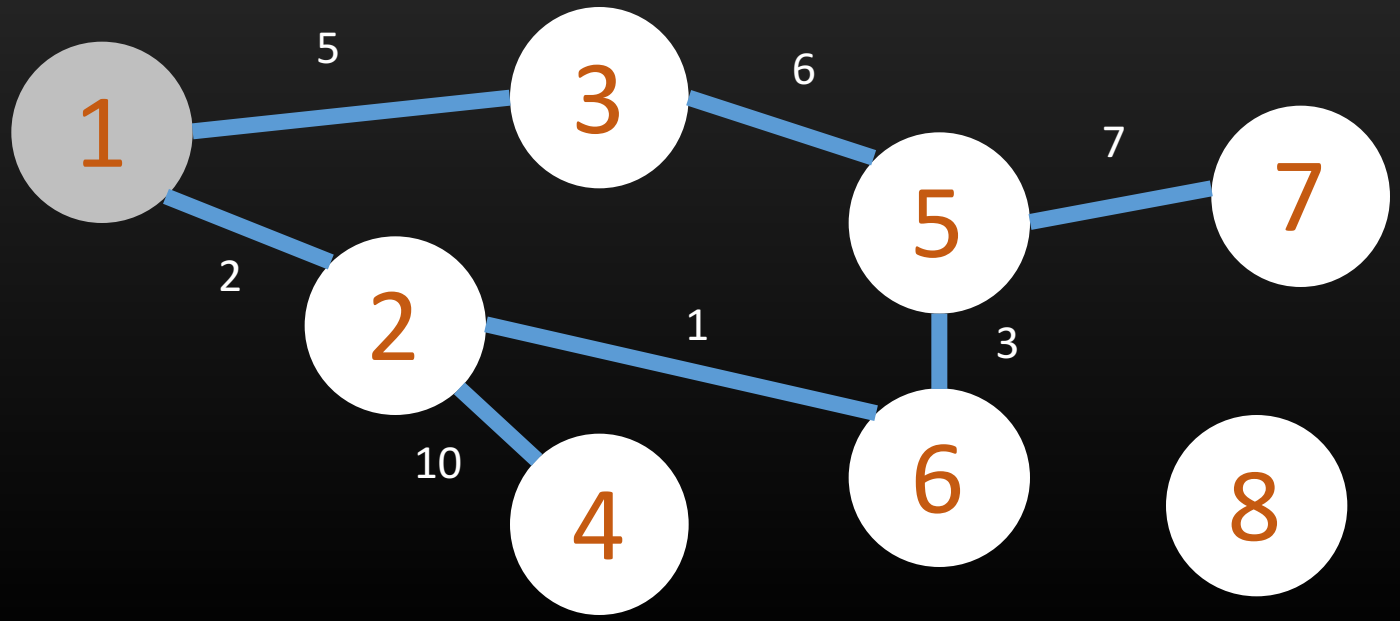
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	∞	∞	∞	∞	∞



Bellman-ford algorithm - example

- 2nd iteration

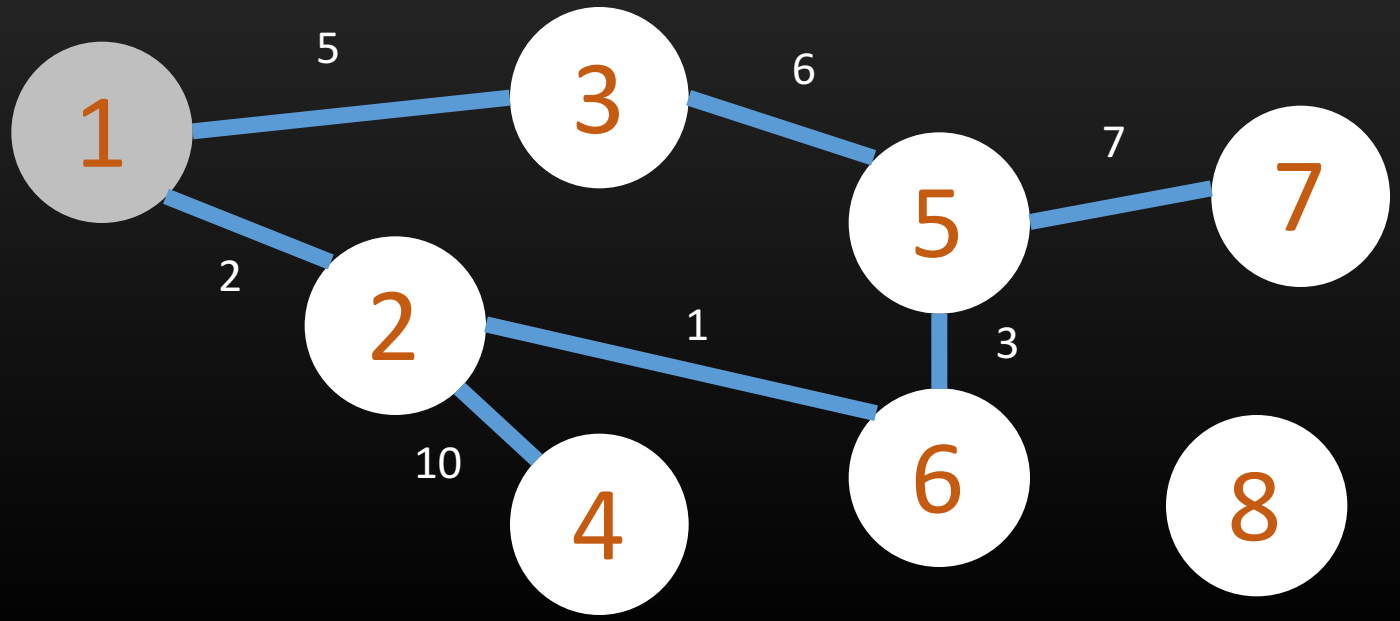
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	11	3	∞	∞



Bellman-ford algorithm - example

- 3rd iteration

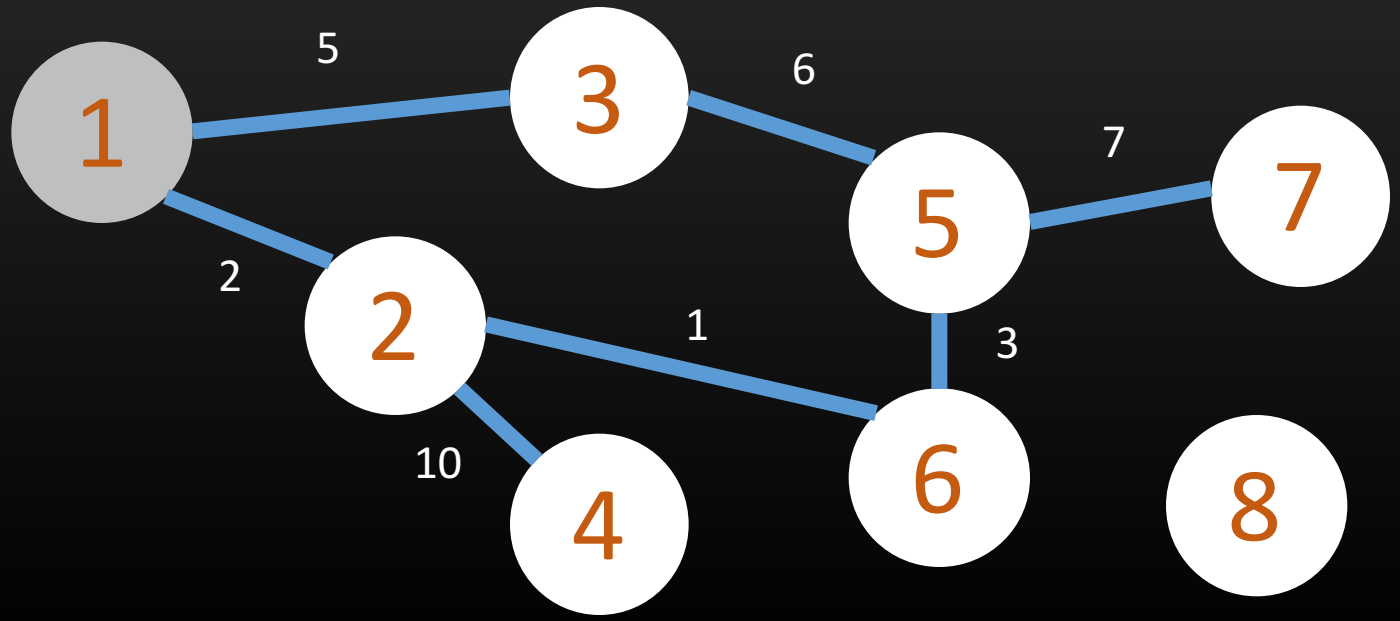
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	18	∞



Bellman-ford algorithm - example

- 4th iteration

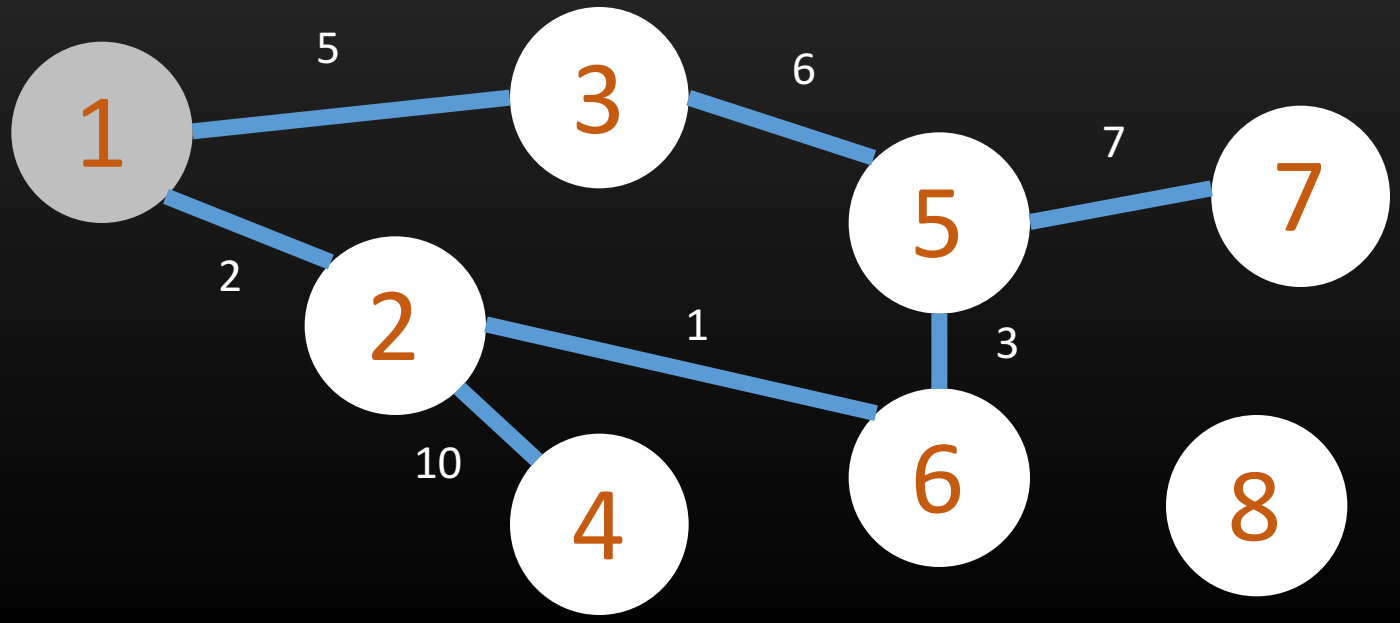
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞



Bellman-ford algorithm - example

- Same for 5th to 7th iteration

Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞

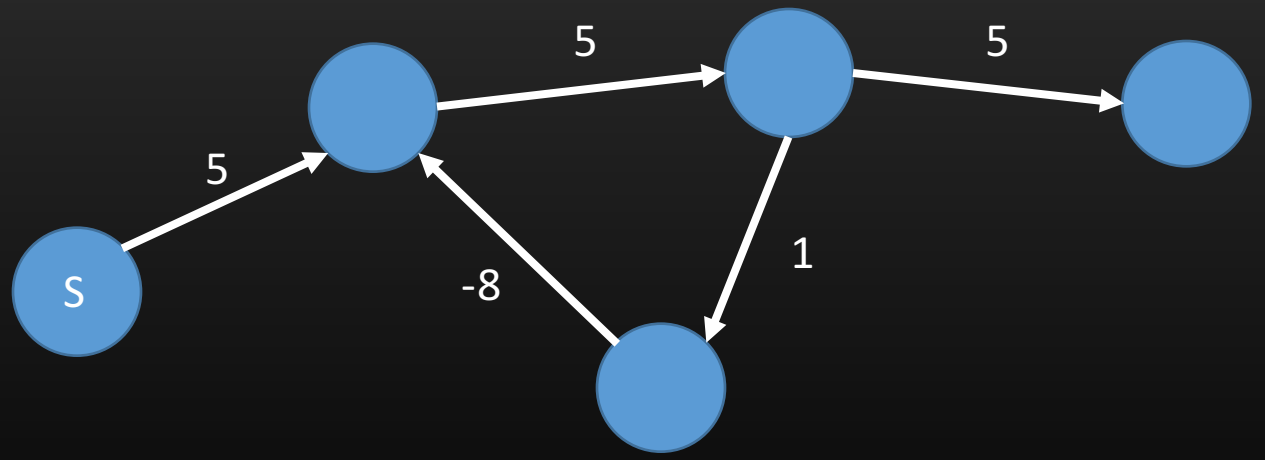


Bellman-ford algorithm - pseudocode

```
Procedure bellmanford{
    for each vertex v in the graph
        dist[v] = infinity
    dist[source] = 0
    for i from 1 to  |V|-1 do
        For every edge u-v
            dist[v] = min(dist[v] , dist[u] + weight[u-v])
}
```

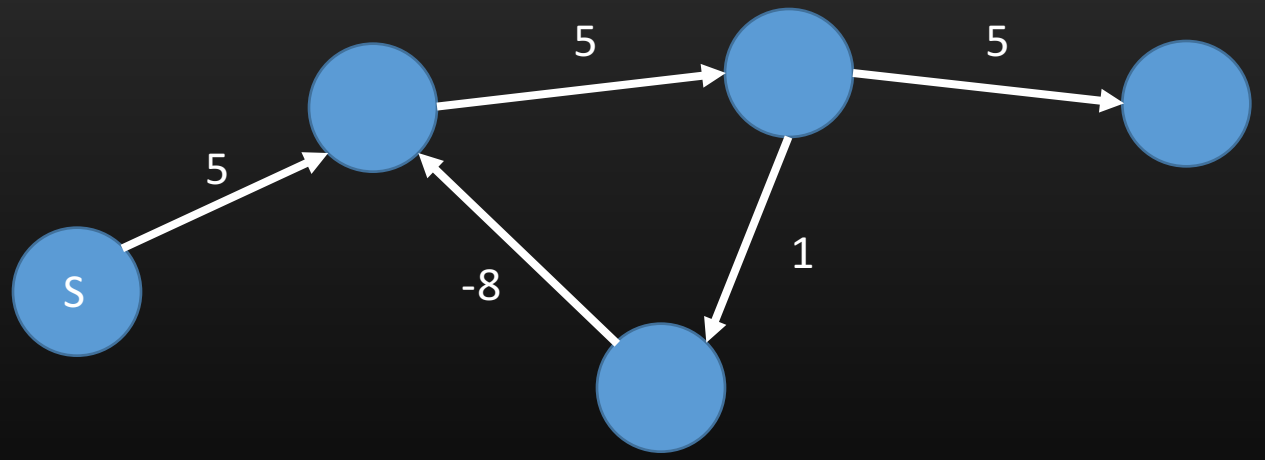
Bellman-ford algorithm

- Time complexity: O(|V||E|)
- How to detect negative cycles?



Bellman-ford algorithm

- After |V|-1 iterations, all shortest path should have been found
- If there still exist edges that can modify answers, then there exists a negative cycle



SPFA

- An improvement of the Bellman–Ford algorithm
- Optimize the process by a queue
- Work well on random graphs

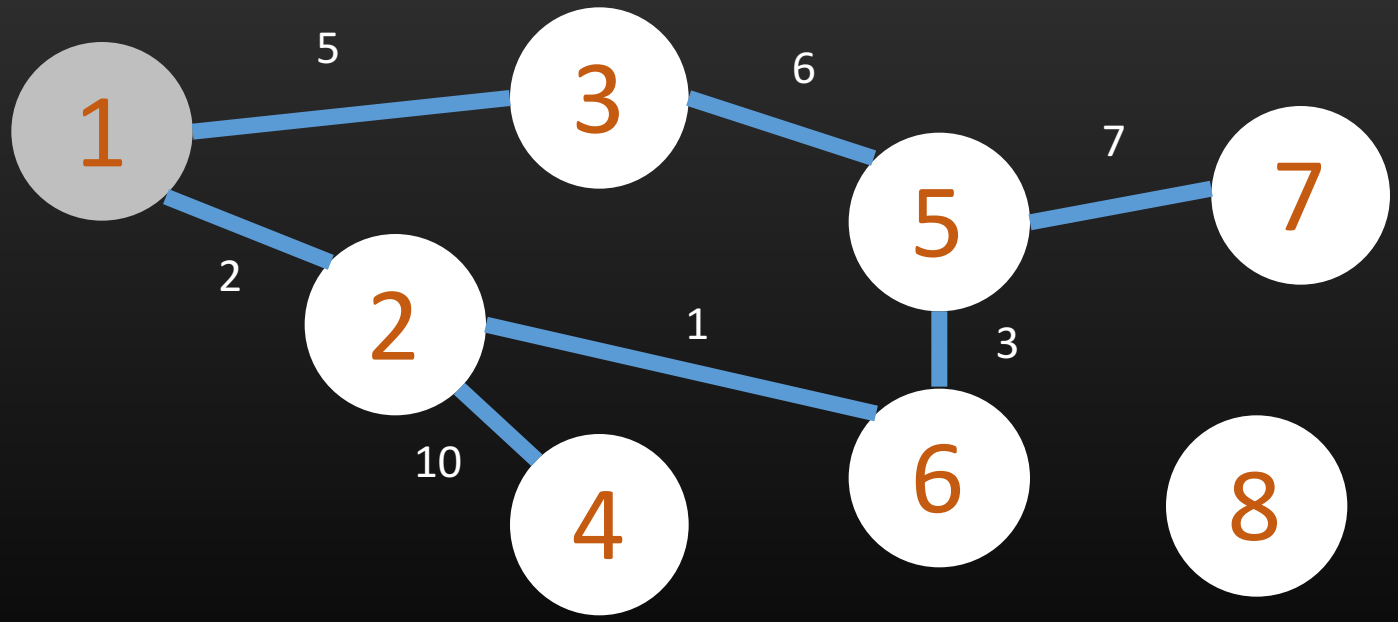
SPFA

- Push the source into the queue first
- While the queue is not empty, repeat the following:
  - Dequeue the first node from the queue
  - Modify the distance of its neighbors using its own distance
  - If the distance of the neighbor can be modified (dist[u]+weight[uv]>dist[v]) and the neighbor is not in the queue, push the neighbor into the queue

SPFA - example

1									

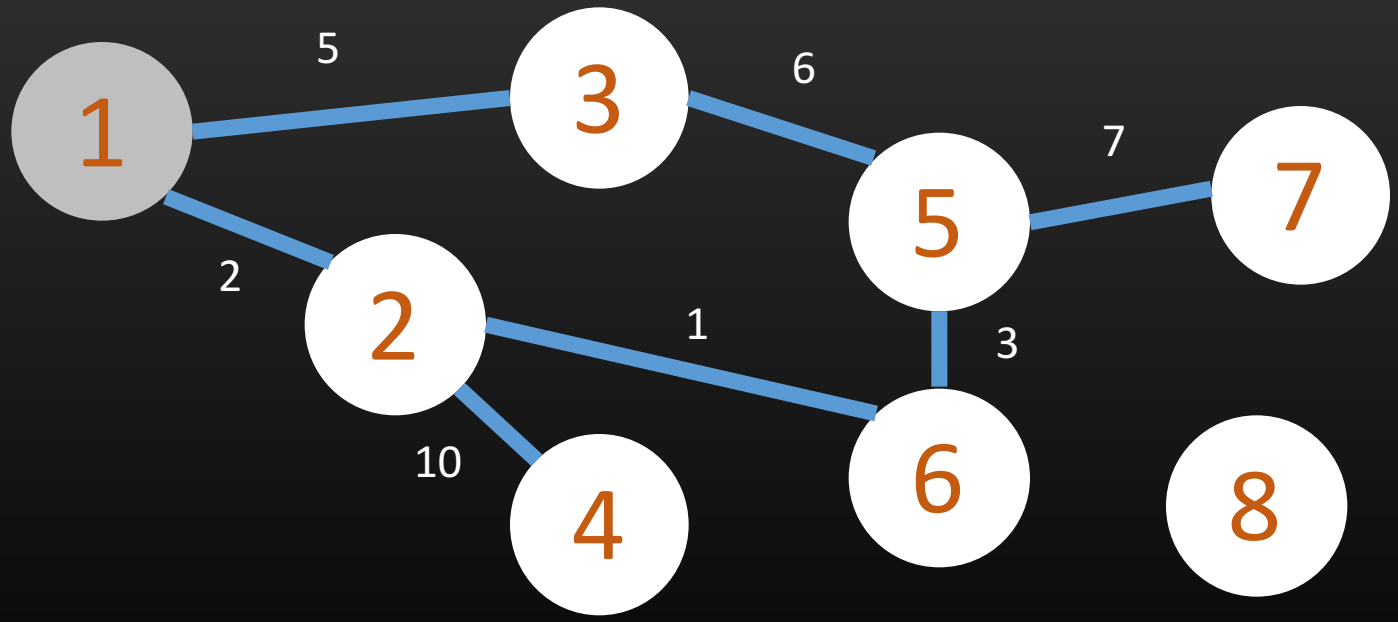
Node	1	2	3	4	5	6	7	8
Distance	0	∞	∞	∞	∞	∞	∞	∞



SPFA - example

1	3	2							

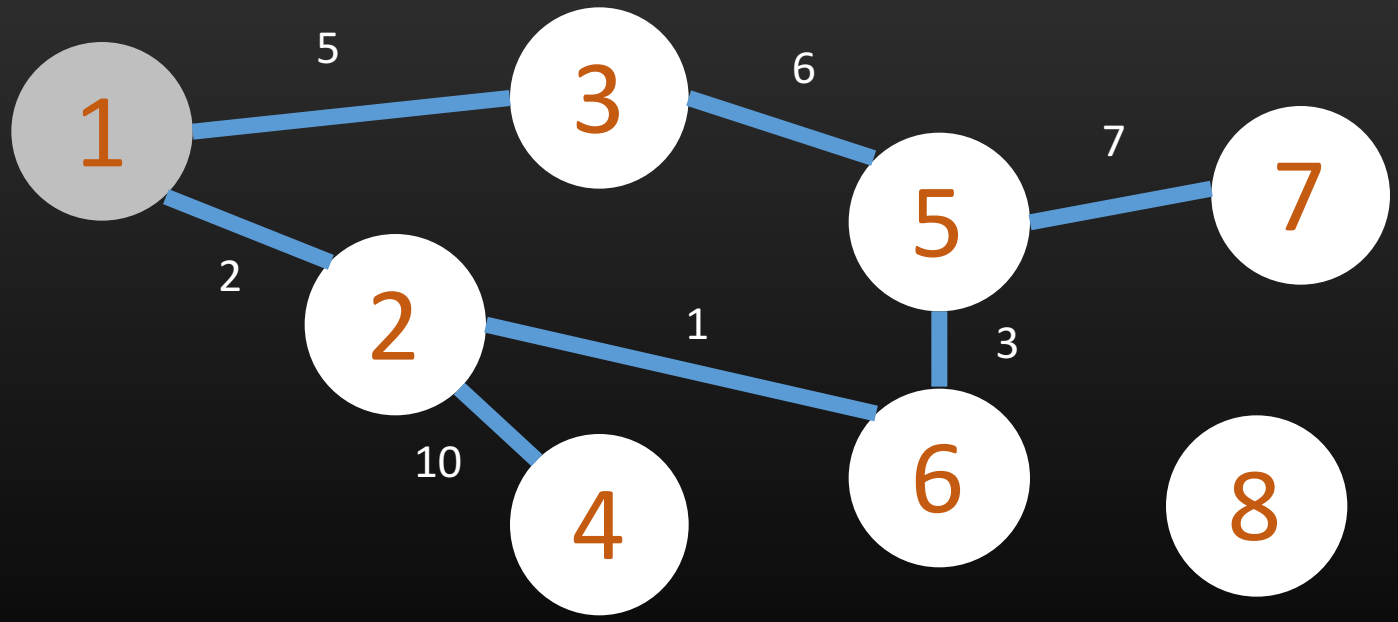
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	∞	∞	∞	∞	∞



SPFA - example

1	3	2							

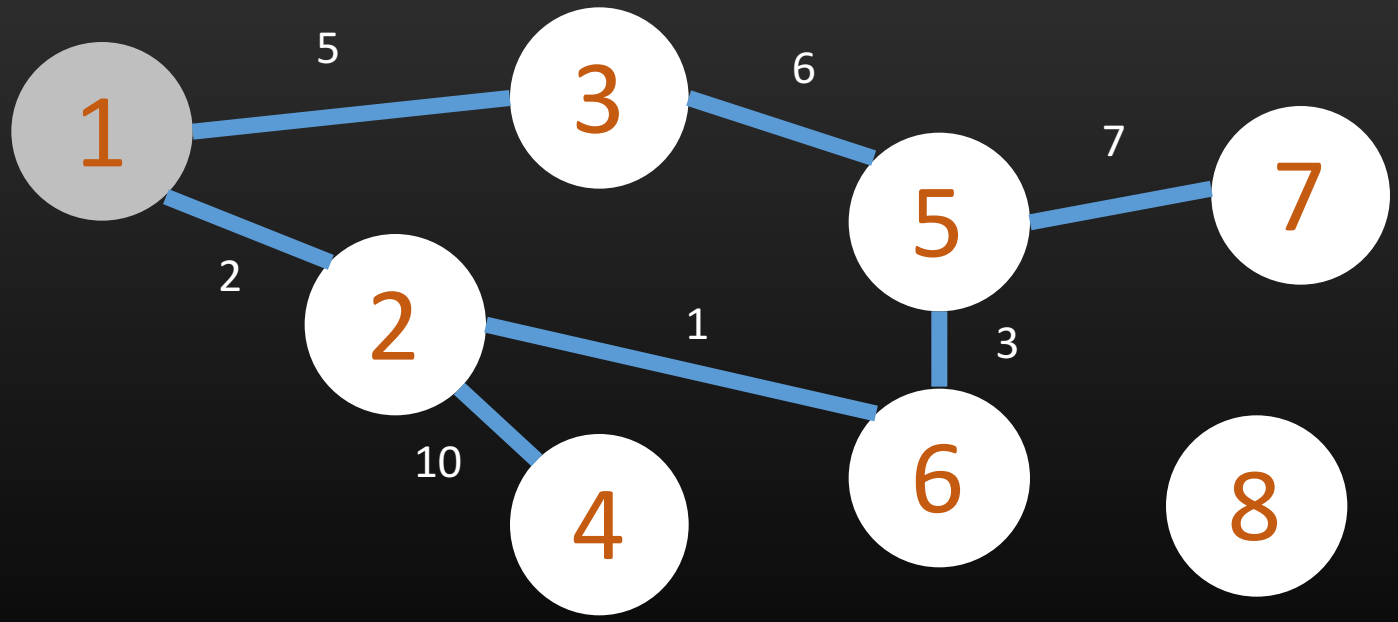
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	∞	∞	∞	∞	∞



SPFA - example

1	3	2	5						

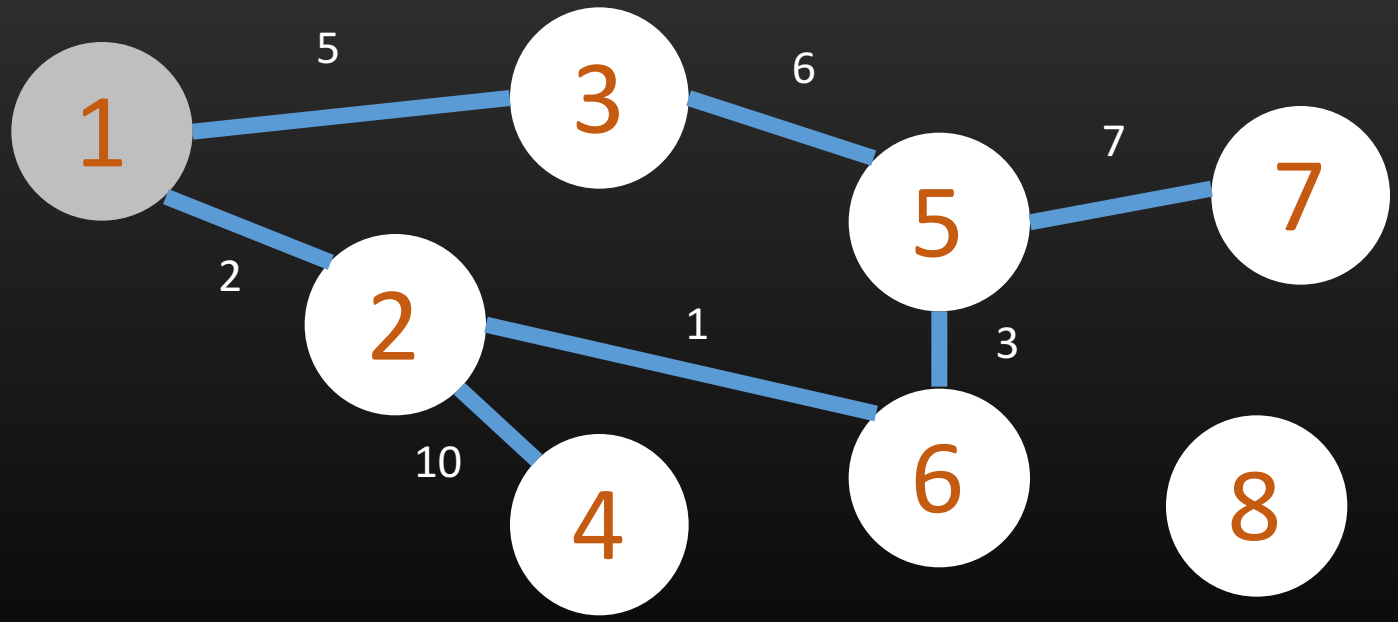
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	∞	11	∞	∞	∞



SPFA - example

1	3	2	5						

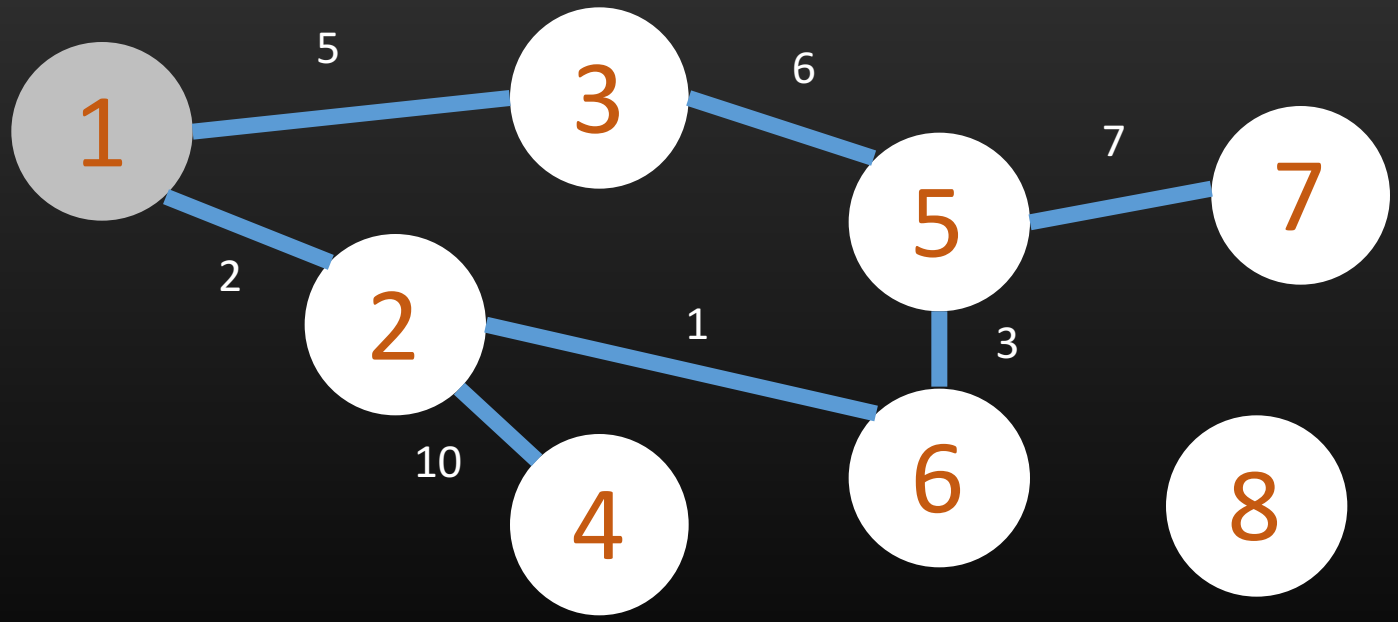
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	∞	11	∞	∞	∞



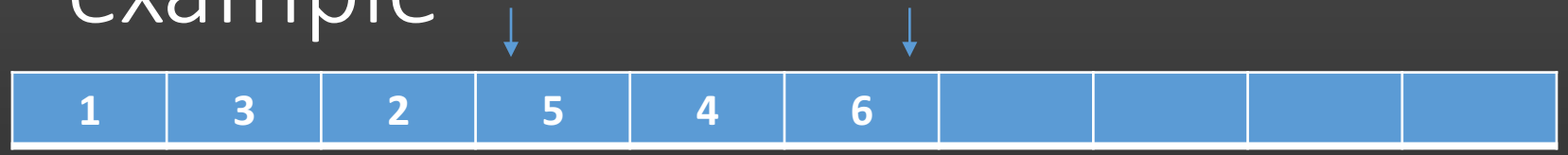
SPFA - example

1	3	2	5	4	6				

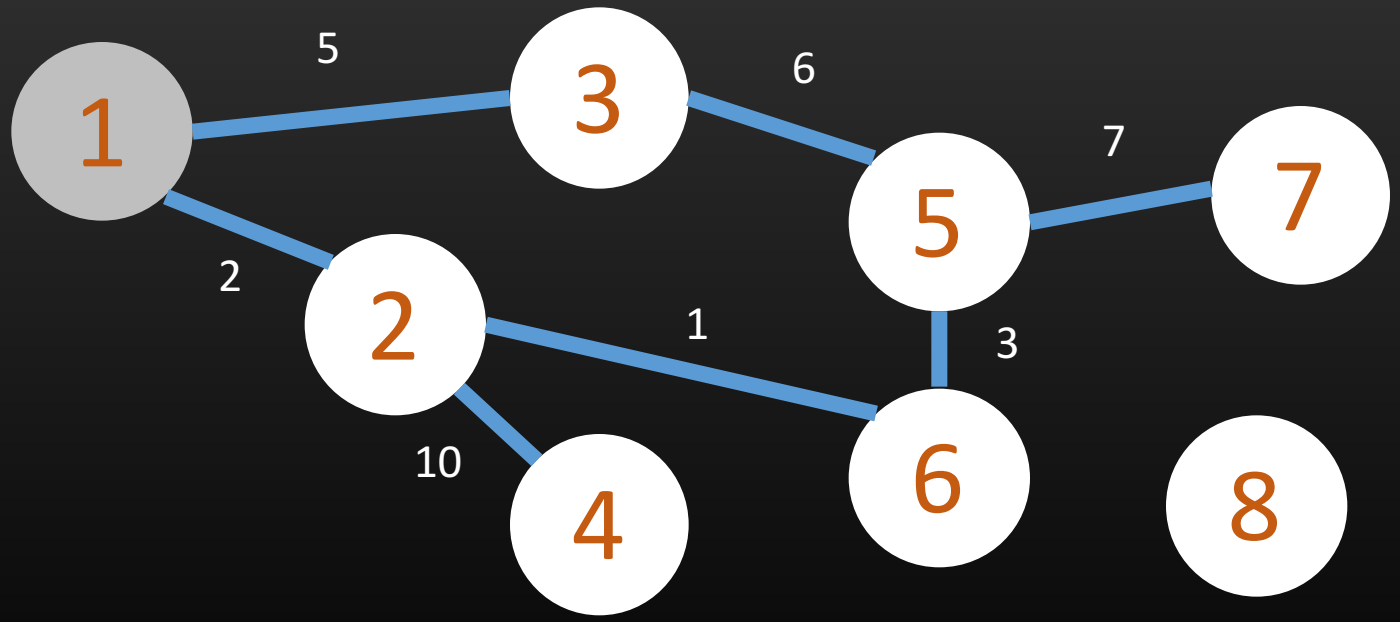
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	11	3	∞	∞



SPFA - example



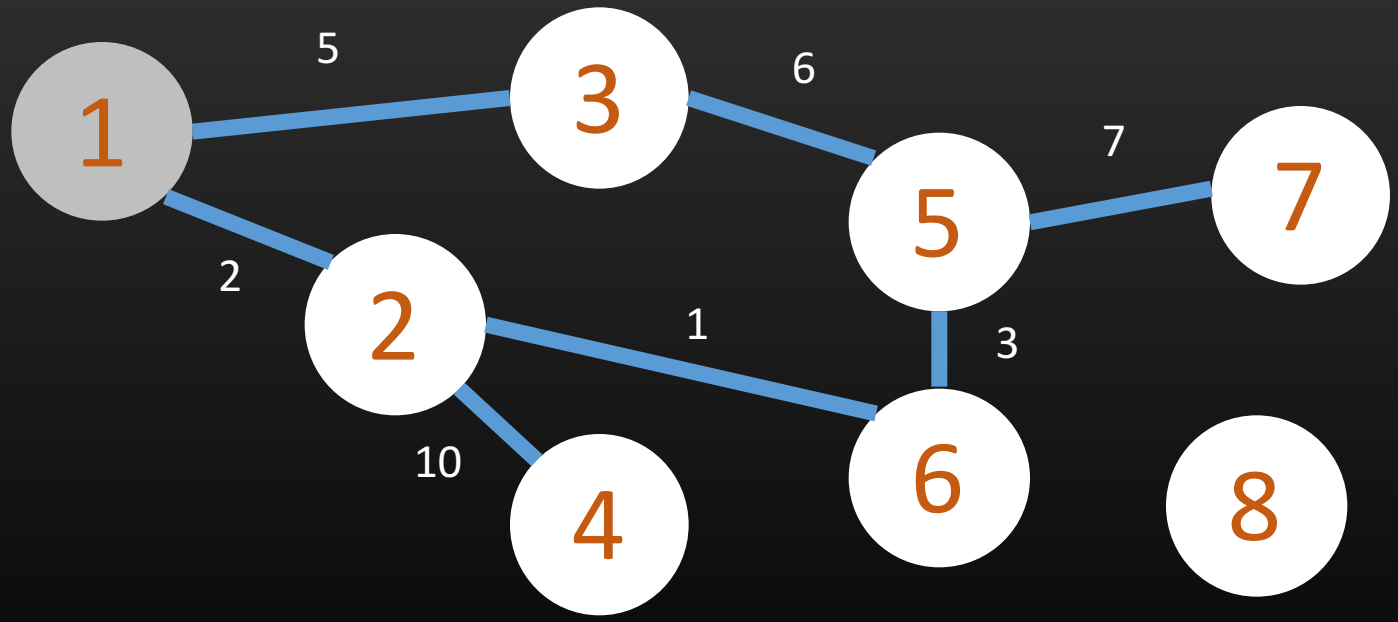
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	11	3	∞	∞



SPFA - example

1	3	2	5	4	6	7			

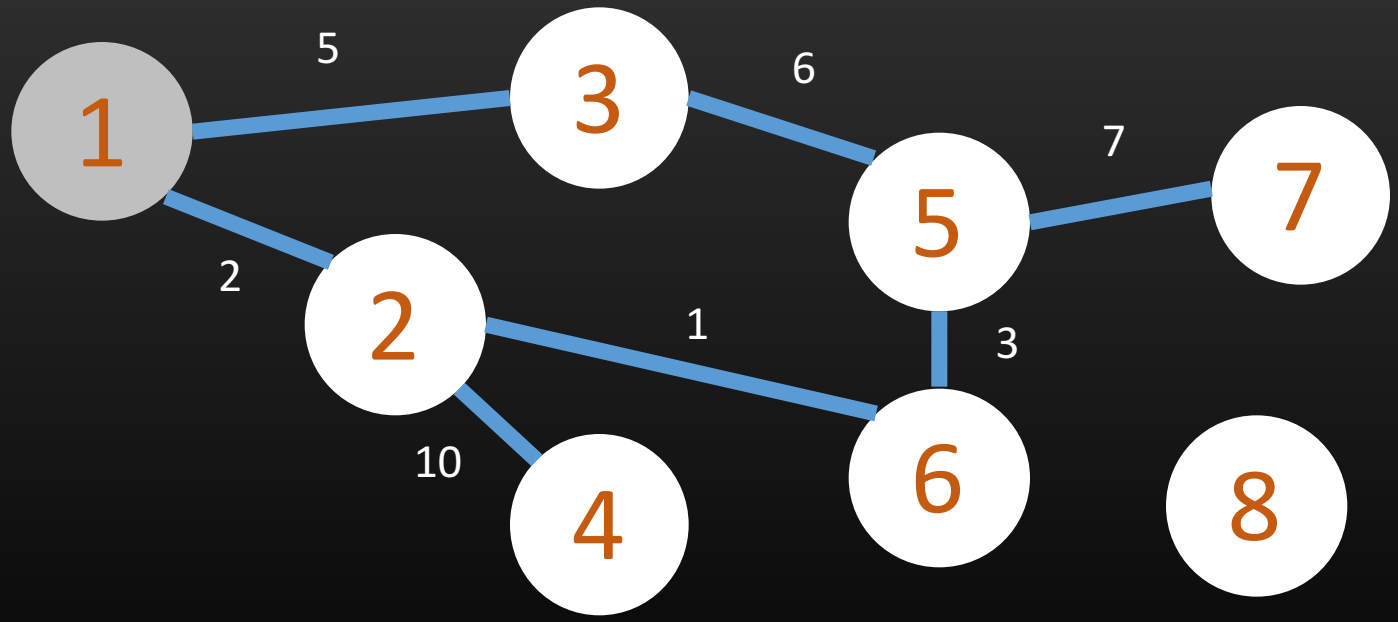
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	11	3	18	∞



SPFA - example

1	3	2	5	4	6	7			

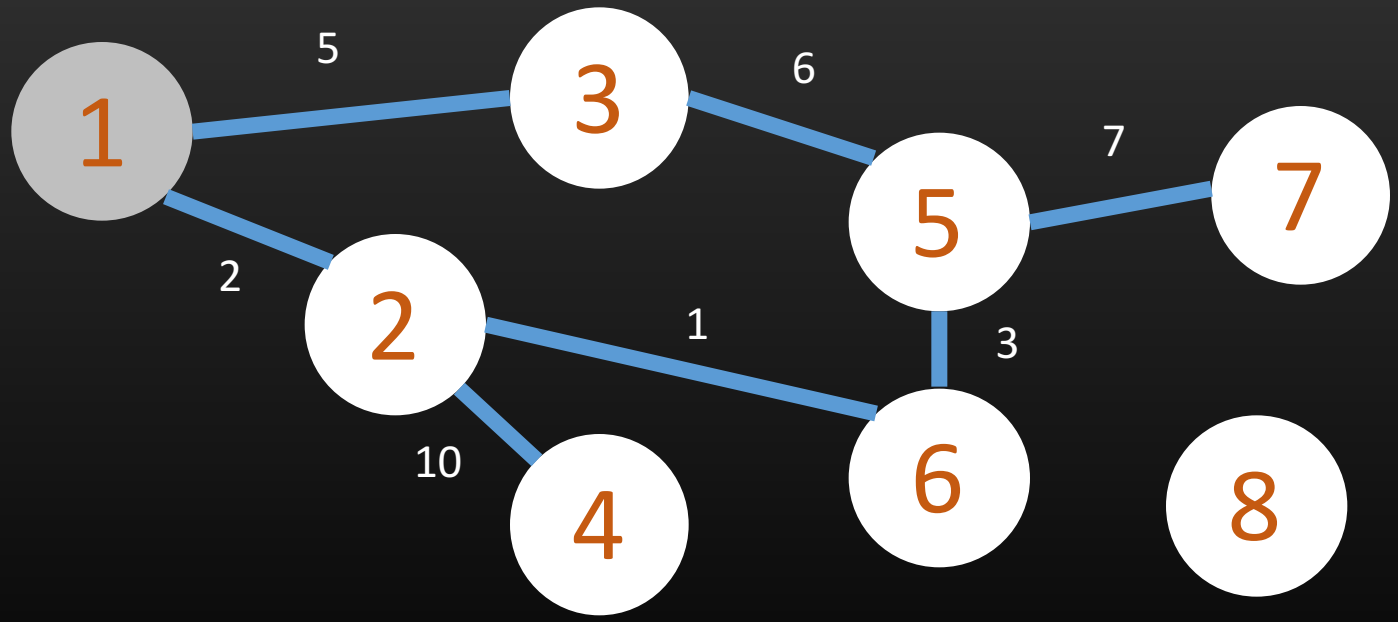
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	11	3	18	∞



SPFA - example

1	3	2	5	4	6	7			

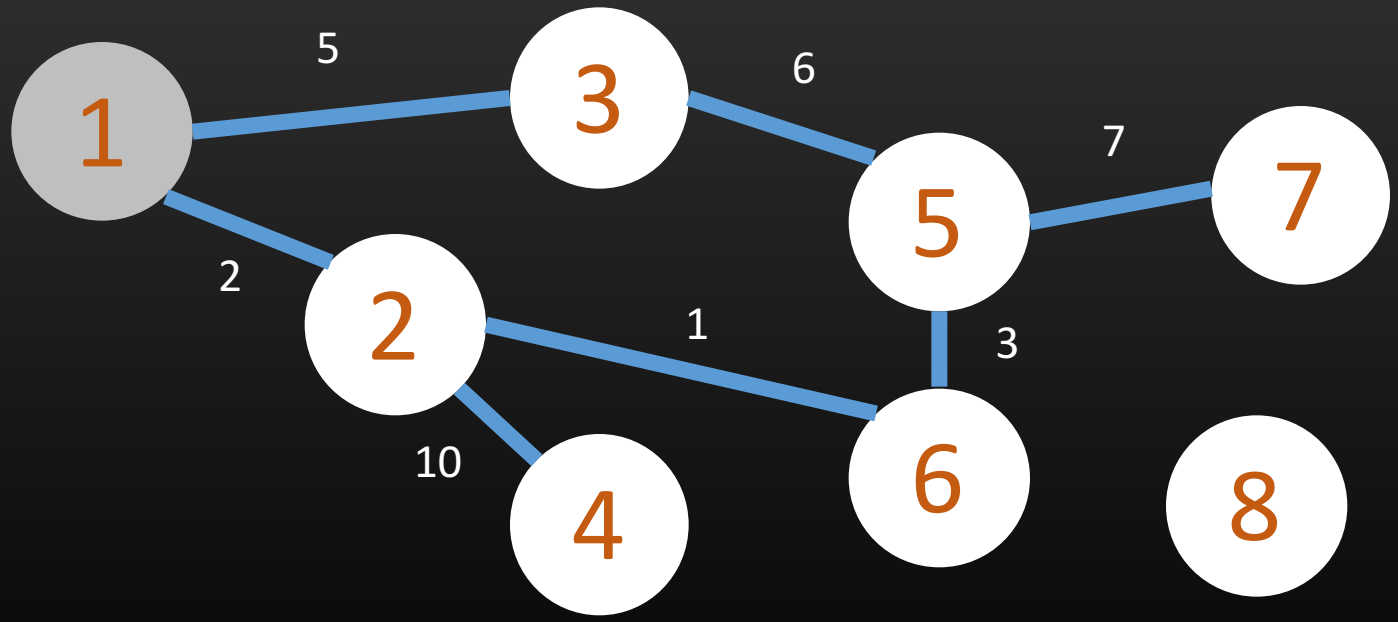
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	11	3	18	∞



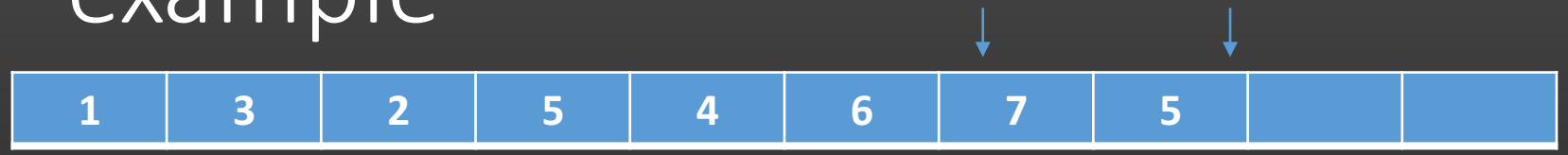
SPFA - example

1	3	2	5	4	6	7	5		

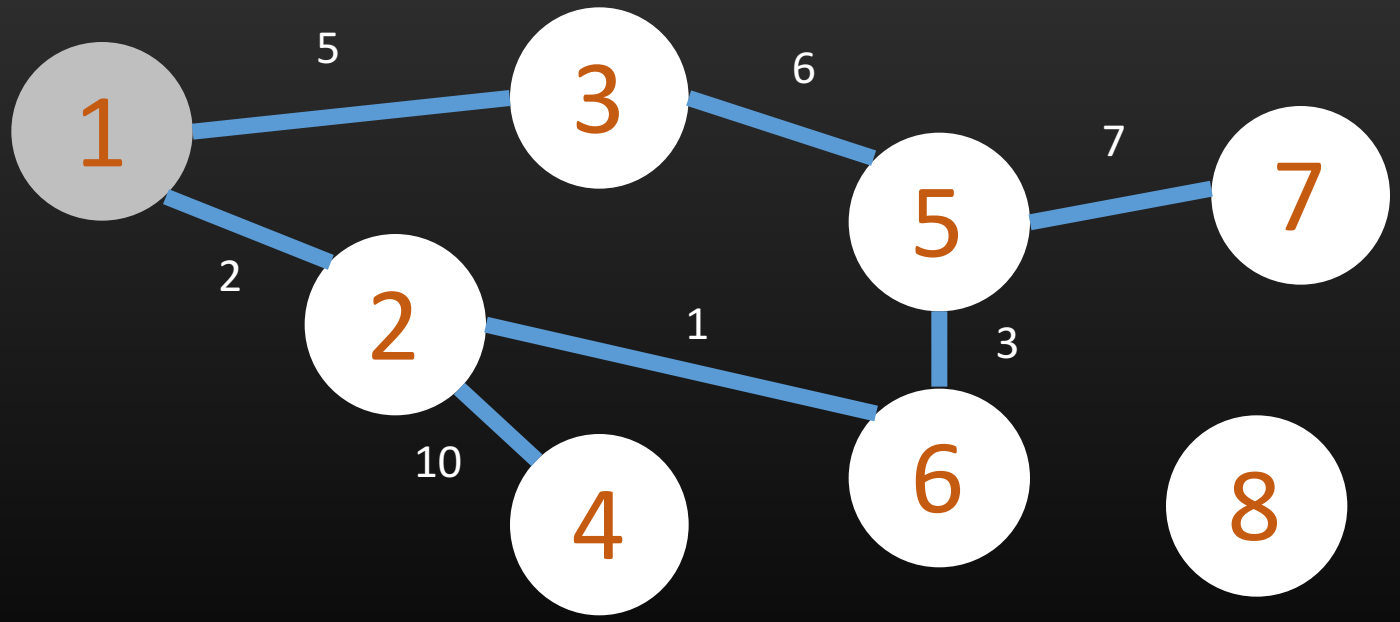
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	18	∞



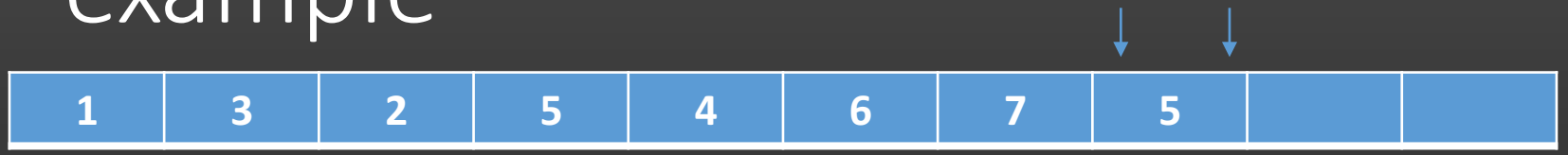
SPFA - example



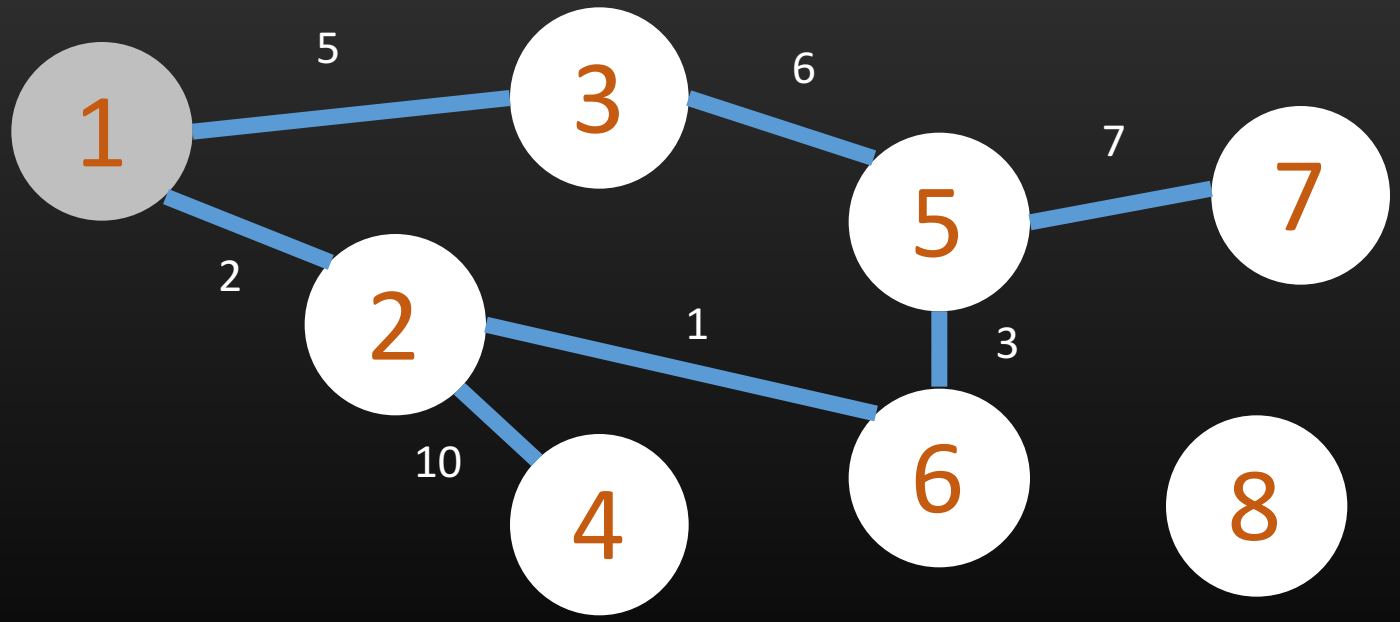
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	18	∞



SPFA - example



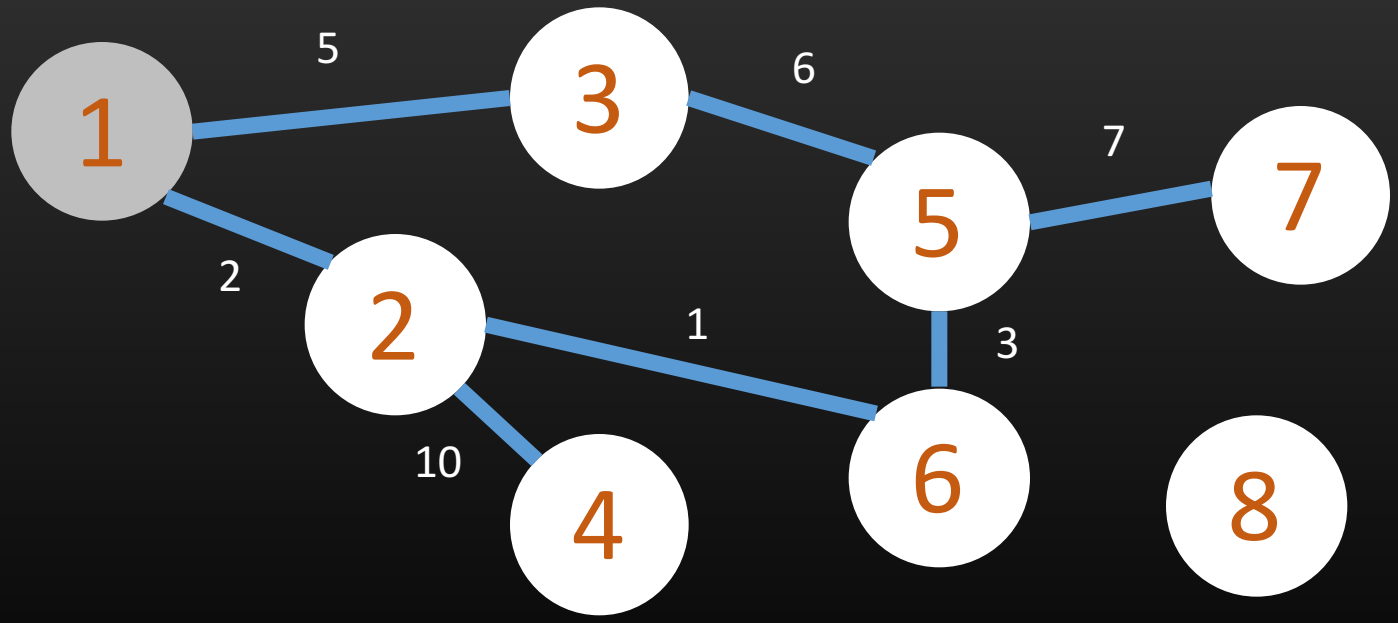
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	18	∞



SPFA - example

1	3	2	5	4	6	7	5	7	

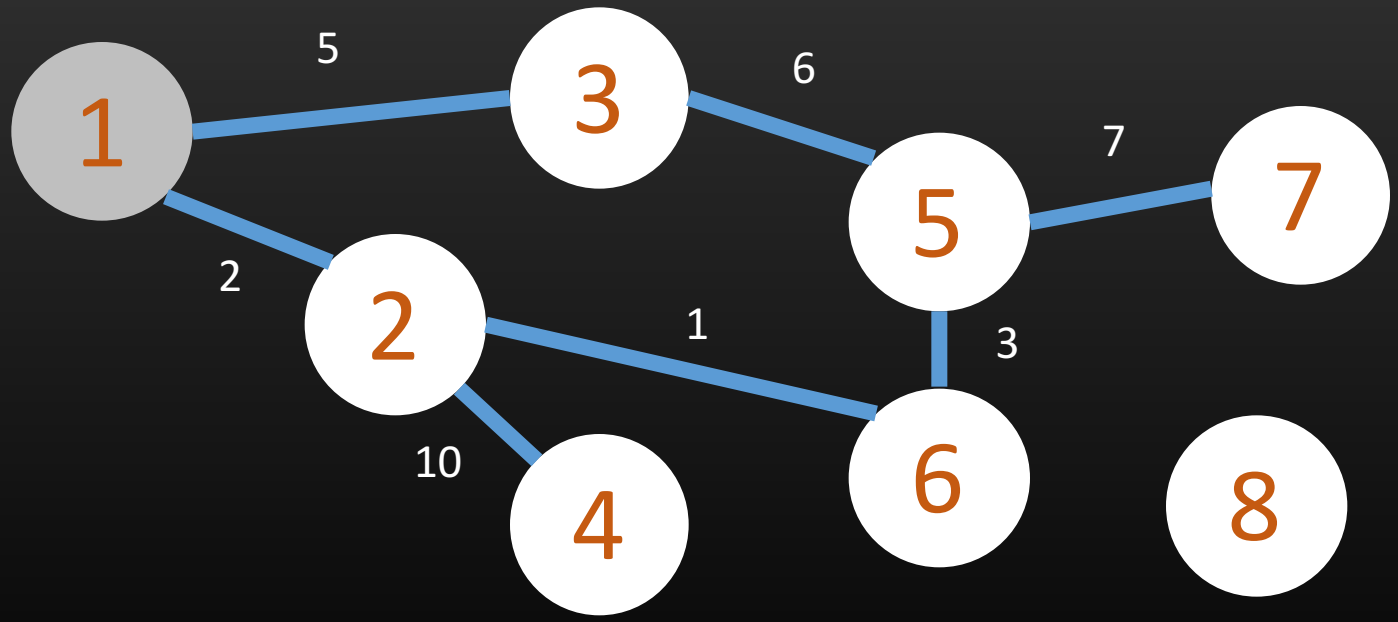
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞



SPFA - example

1	3	2	5	4	6	7	5	7	

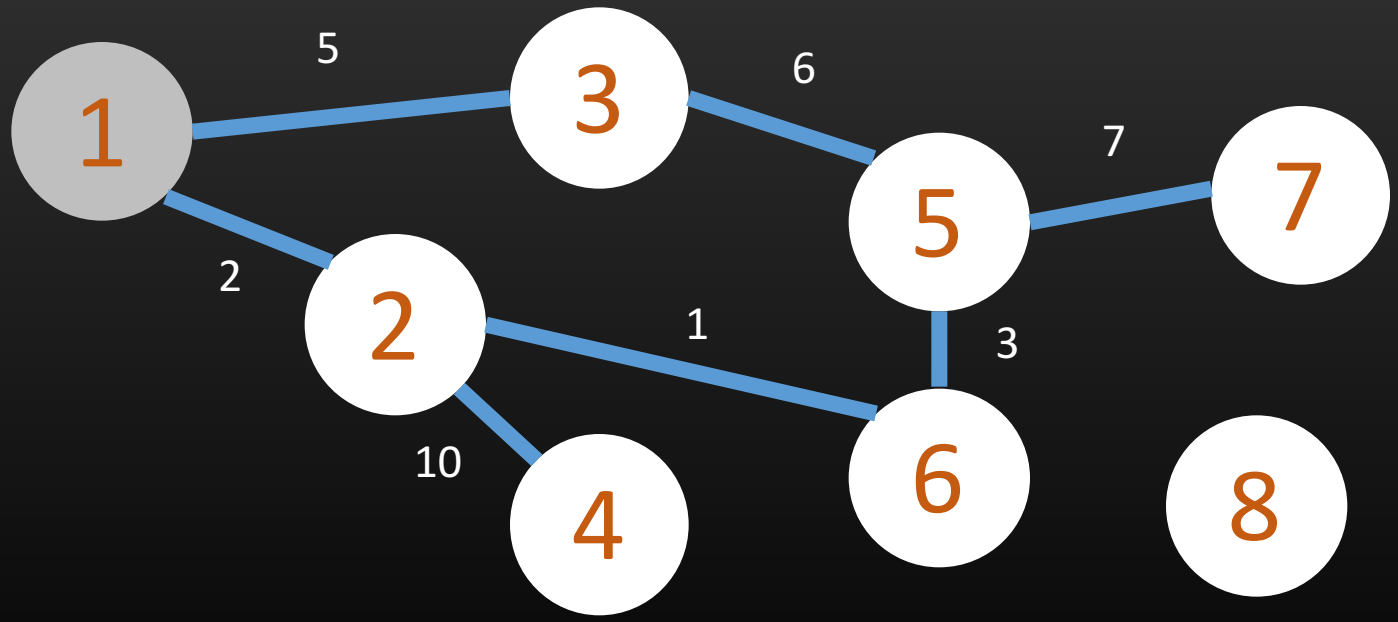
Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞



SPFA - example

1	3	2	5	4	6	7	5	7	

Node	1	2	3	4	5	6	7	8
Distance	0	2	5	12	6	3	13	∞



SPFA - pseudocode

```
Procedure spfa(){
    for each vertex v in the graph
        dist[v] = infinity
    dist[source] = 0
    Q.enqueue(source)
    while the queue Q is not empty
        u = Q.dequeue()
        for each neighbor v of u do
            if dist[u] + weight of the edge u-v < dist[v] then
                dist[v] = dist[u] + weight of the edge u-v
                if v is not in Q
                    Q.enqueue(v)
}
```

SPFA

- Empirical average time complexity for random graph: O(|E|)
- Worst case performance: O(|V||E|)
- Specific data can be constructed to defeat this algorithm

Floyd-Warshall Algorithm

- All-pairs shortest path algorithm
- Based on dynamic programming

- Compute shortest paths that only use vertex 1 as intermediate point
- Compute shortest paths that only use vertex 1 to 2 as intermediate points
- Compute shortest paths that only use vertex 1 to 3 as intermediate points
- …
- Compute shortest paths that only use vertex 1 to V as intermediate points

Floyd-Warshall Algorithm

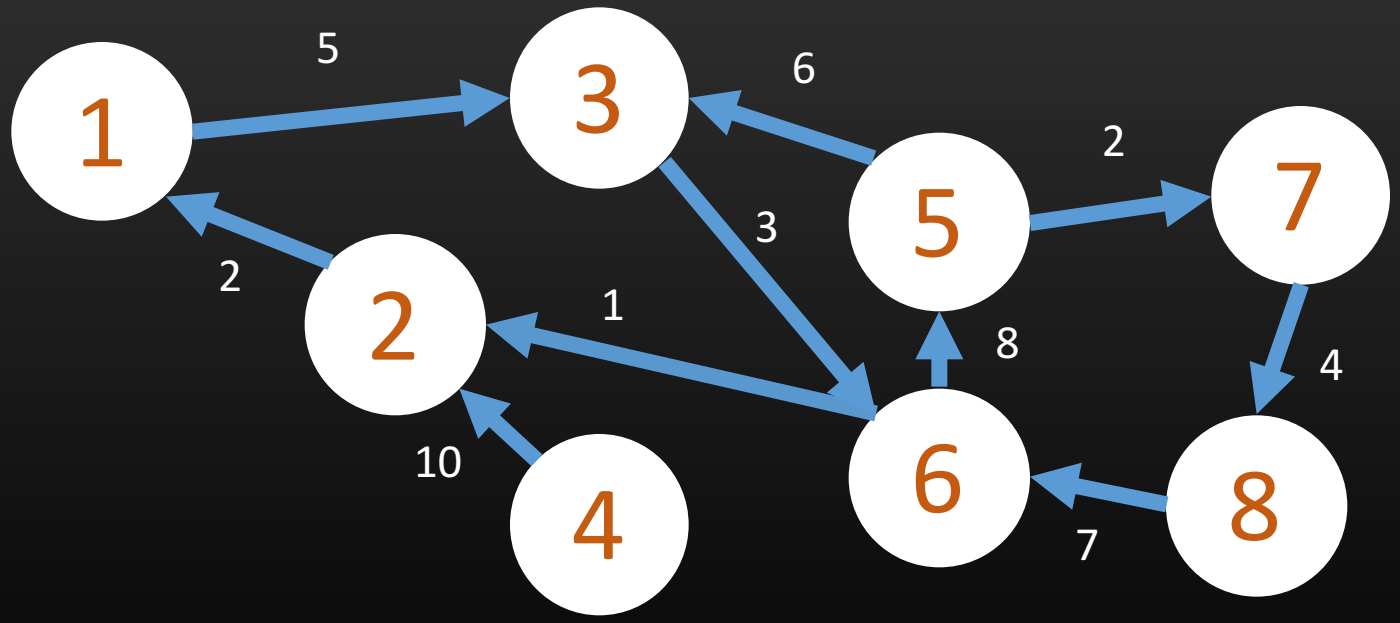
- For all pairs (i, j), set w[i][j] = inf
- For all i, set w[i][i] = 0
- Store edges in w[][] where w[i][j] = distance from i to j
- For k = 1 to V
  - For i = 1 to V
    - For j = 1 to V
      - w[i][j] = min(w[i][j], w[i][k] + w[k][j])

Floyd-Warshall Algorithm

- Time Complexity: $O(|V|^3)$
- Space Complexity: $O(|V|^2)$

Example

- Given a directed graph, find the minimum weight cycle



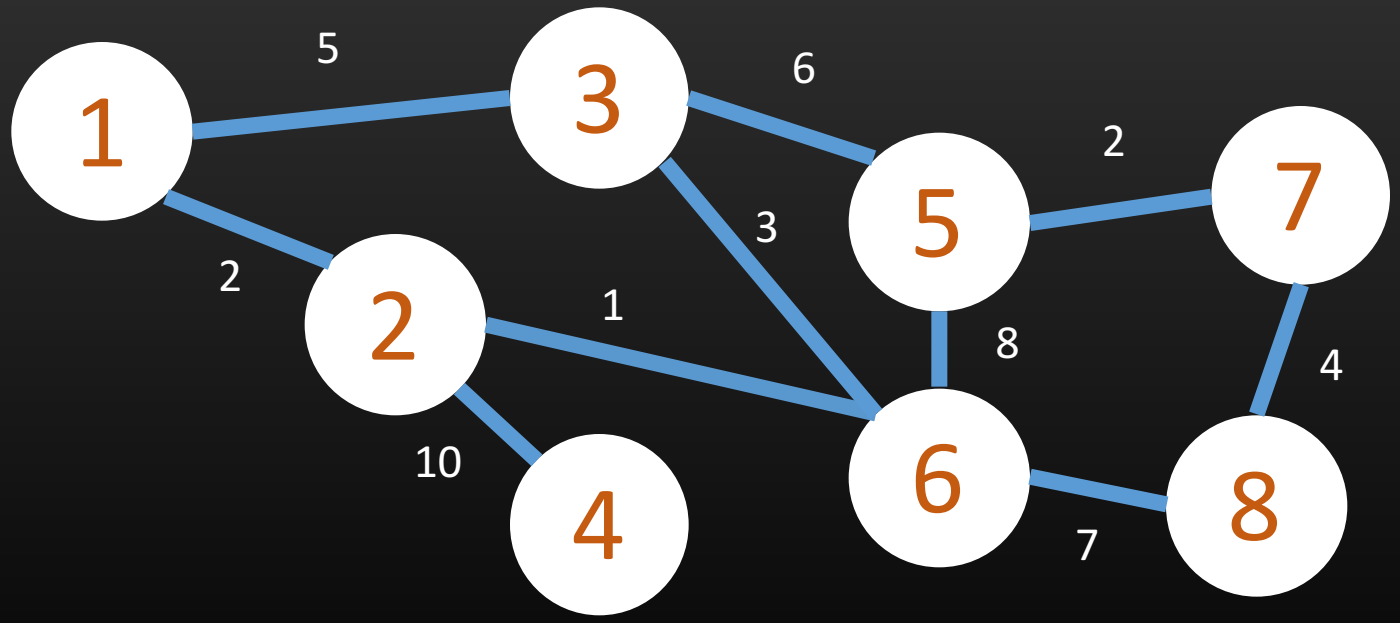
Example

- Compute shortest path of all pairs using Floyd-Warshall algorithm
- Find the pair (i, j) that w[i][j]+w[j][i] is minimum

- Time complexity: $O(V^3)$

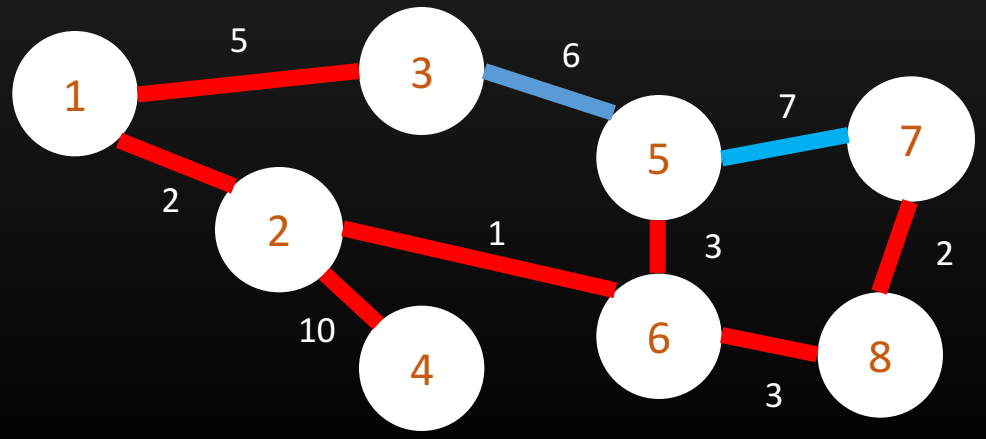
Example

- Given an undirected graph, find the minimum weight cycle



Minimum spanning tree

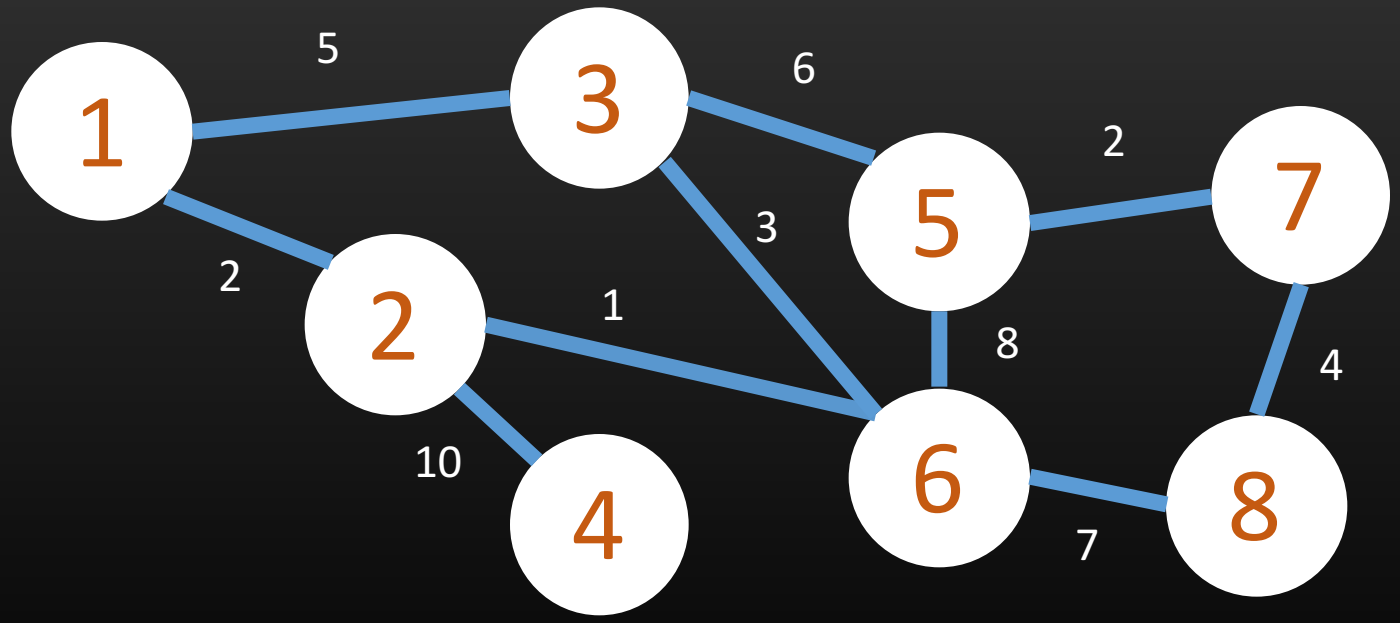
- Suppose we have a graph G
- We select (V –1) edges in G, such that all V nodes are connected by them
- These (V –1) edges form a spanning tree
- A minimum spanning tree is the spanning tree whose sum of weight is minimum



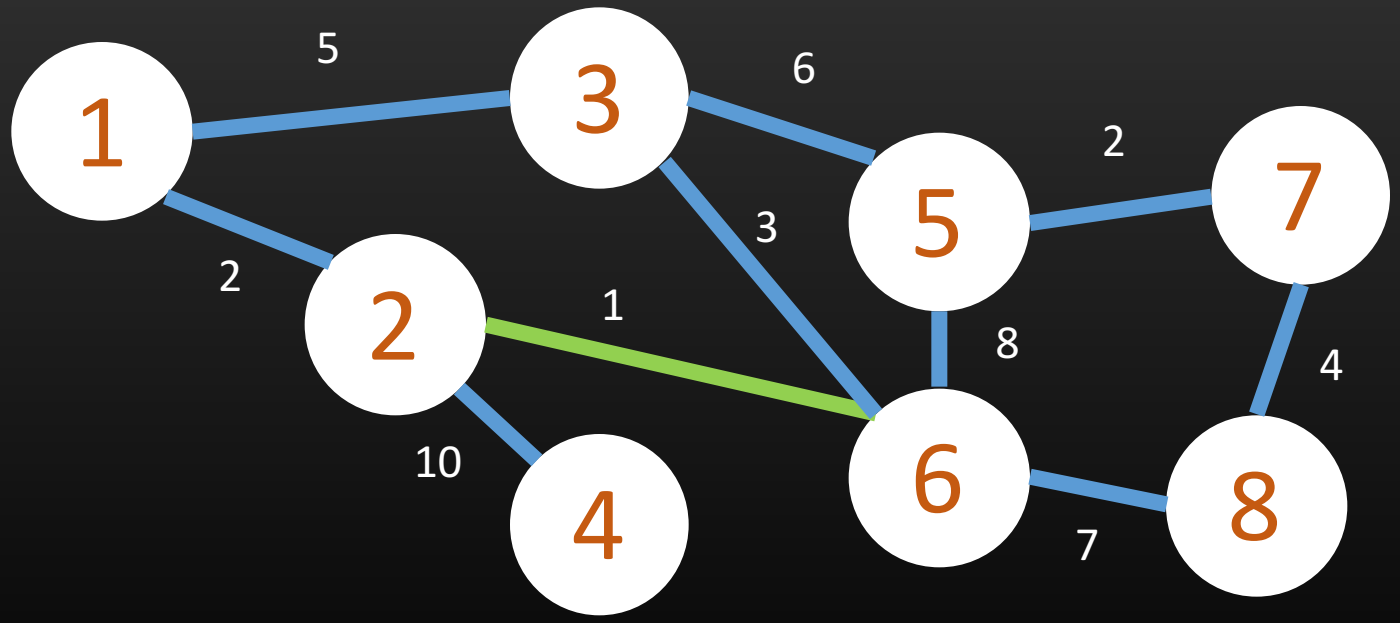
Kruskal's Algorithm

- Greedy algorithm
- It must be optimal to combine two subtrees into one tree using the edge with the smallest weight
- Repeat the process until all vertices are connected

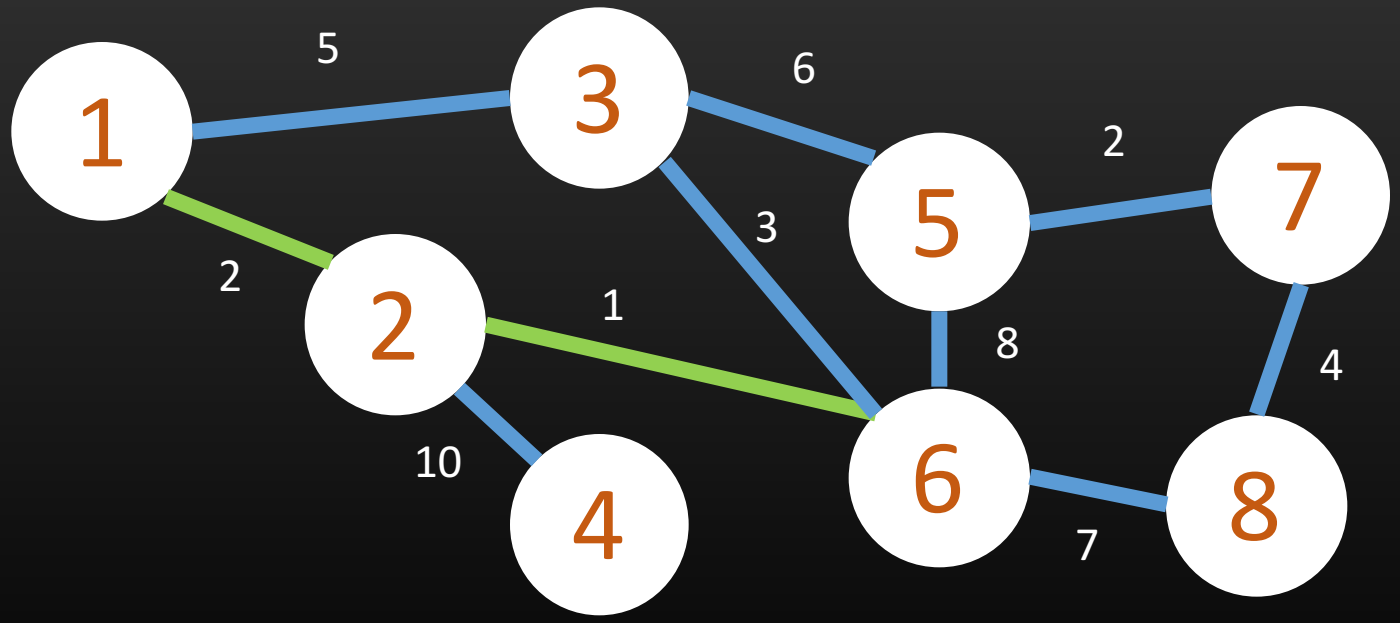
Kruskal’s Algorithm - example



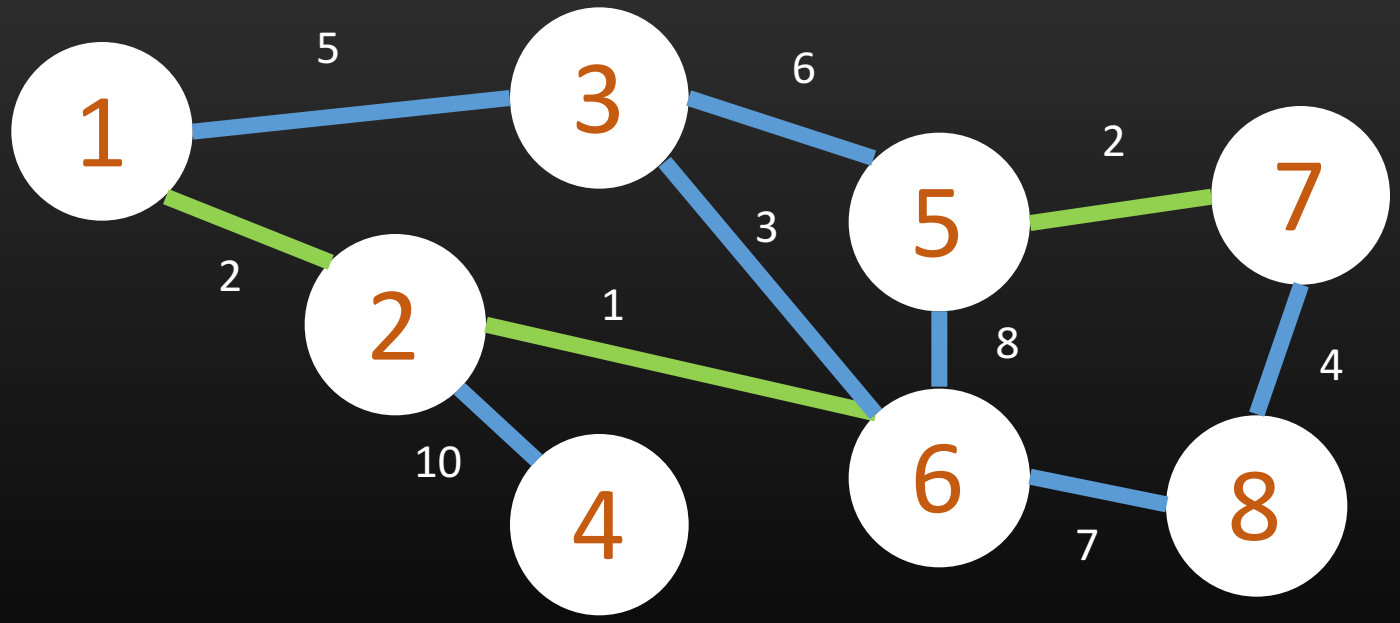
Kruskal’s Algorithm - example



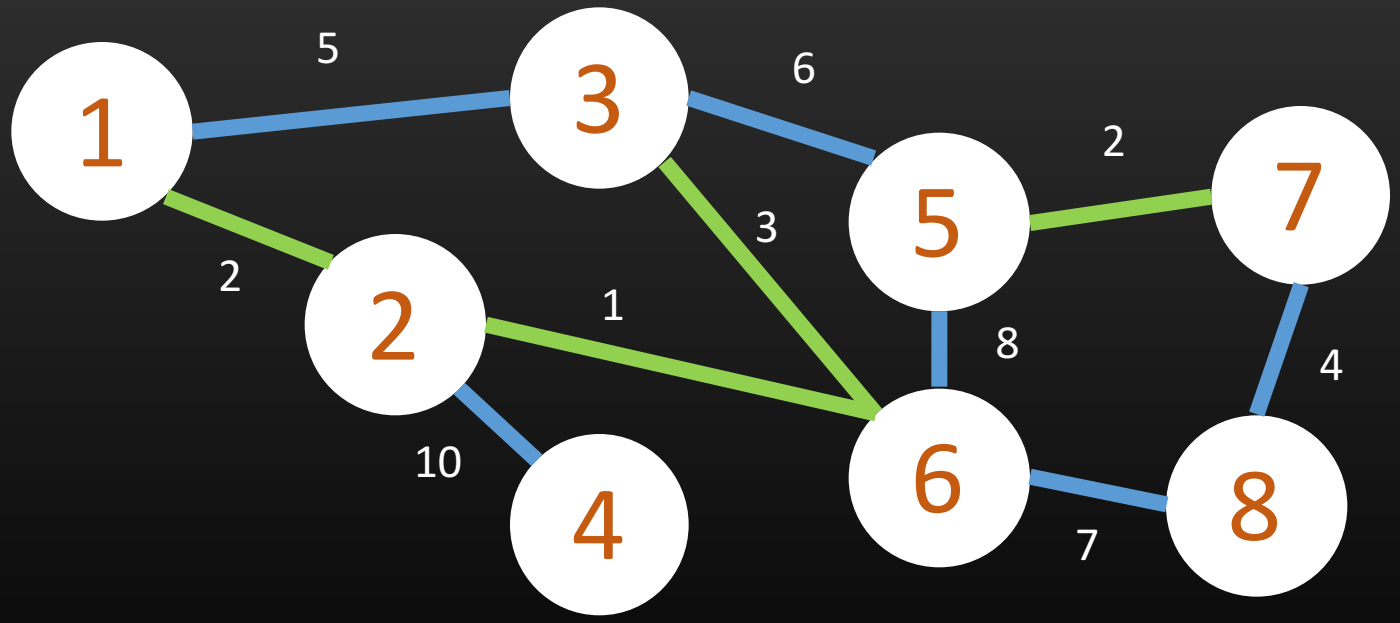
Kruskal’s Algorithm - example



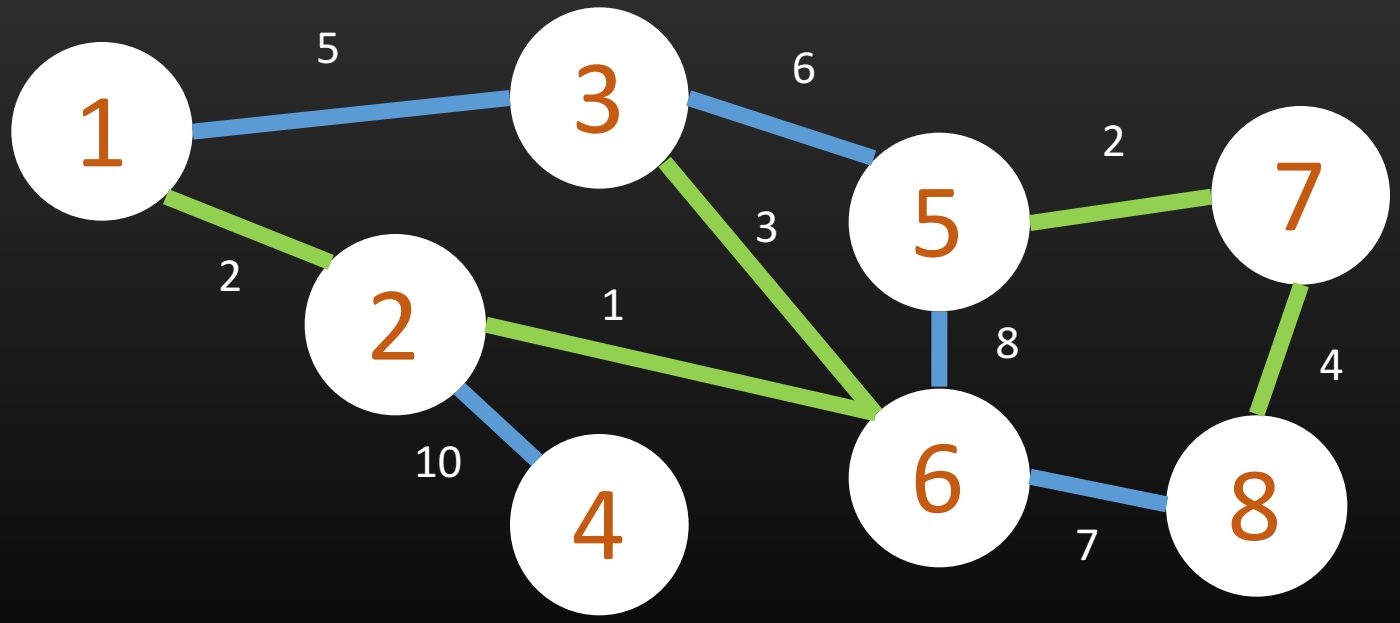
Kruskal’s Algorithm - example



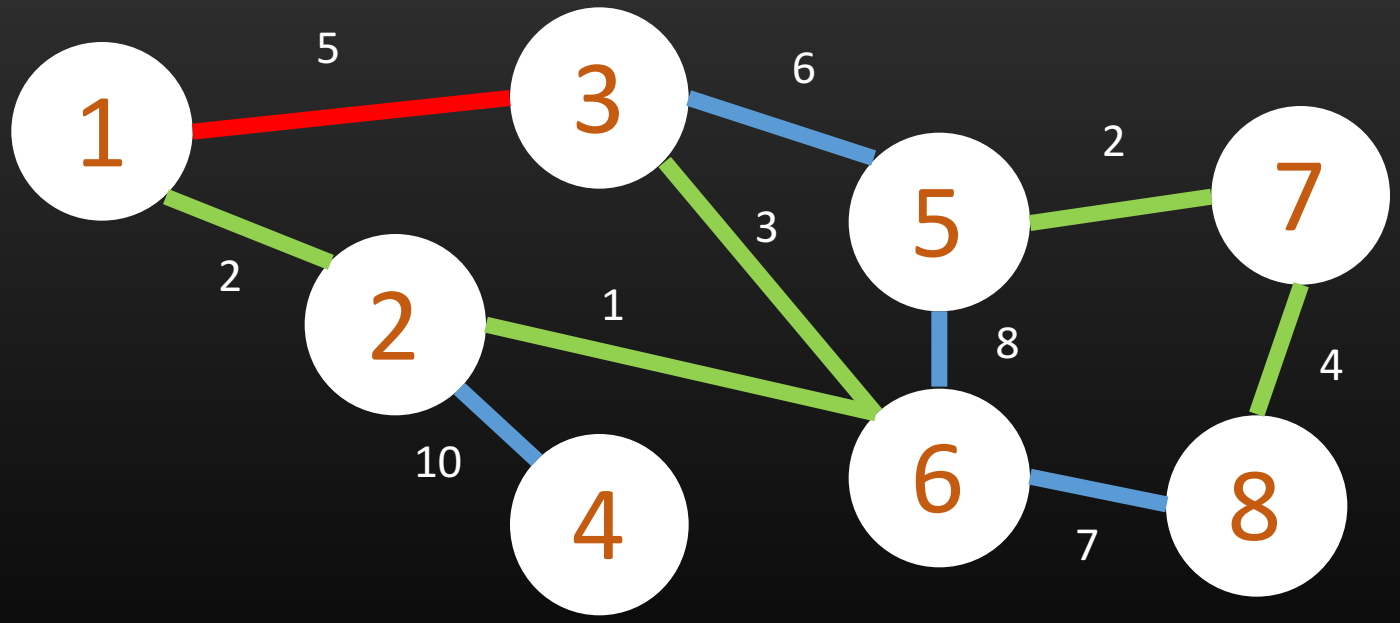
Kruskal's Algorithm - example



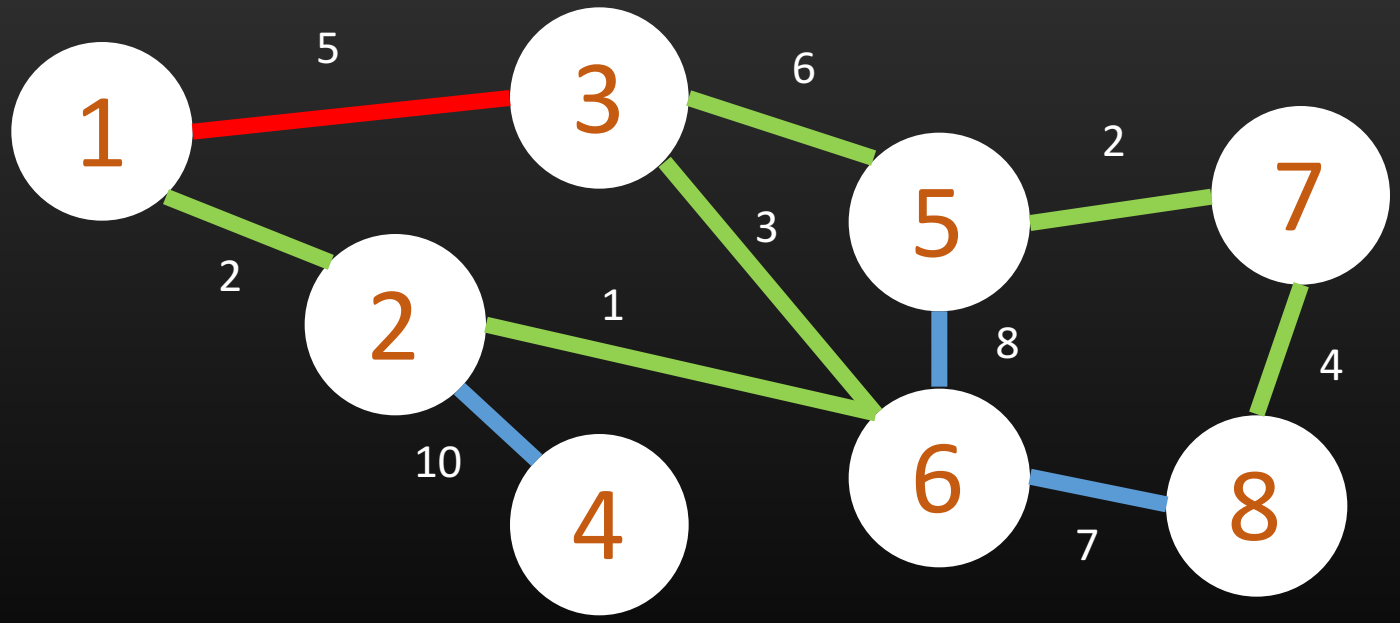
Kruskal’s Algorithm - example



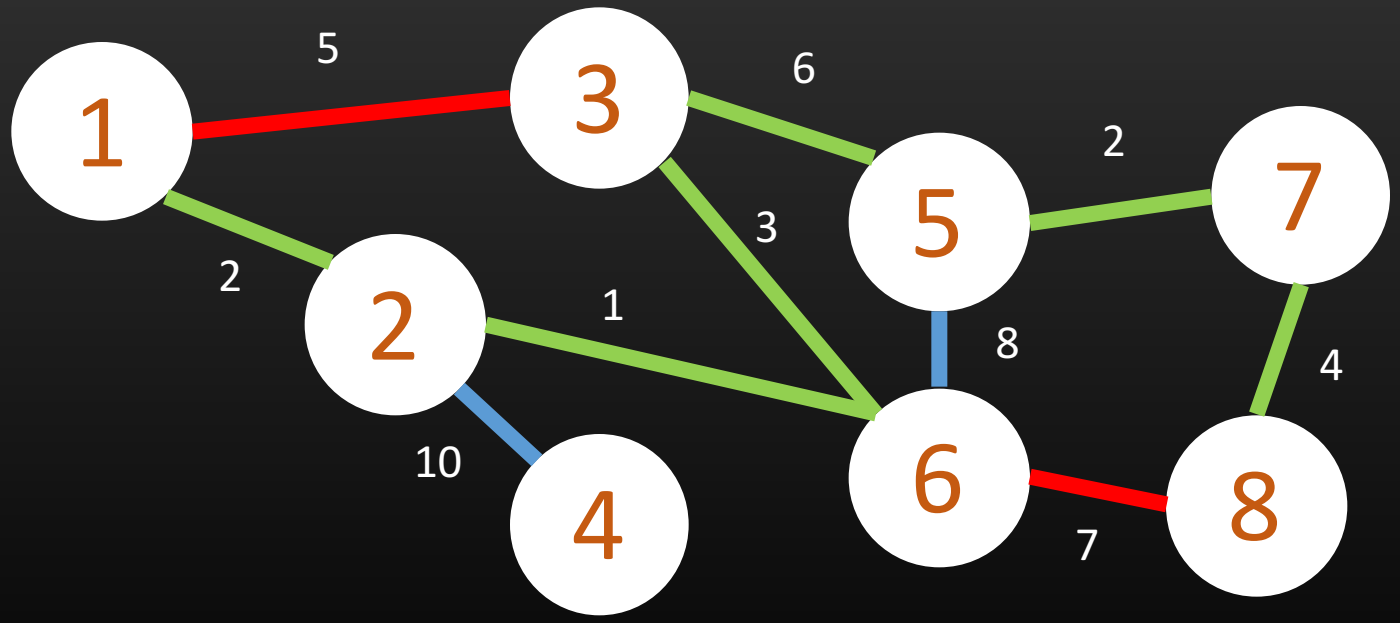
Kruskal’s Algorithm - example



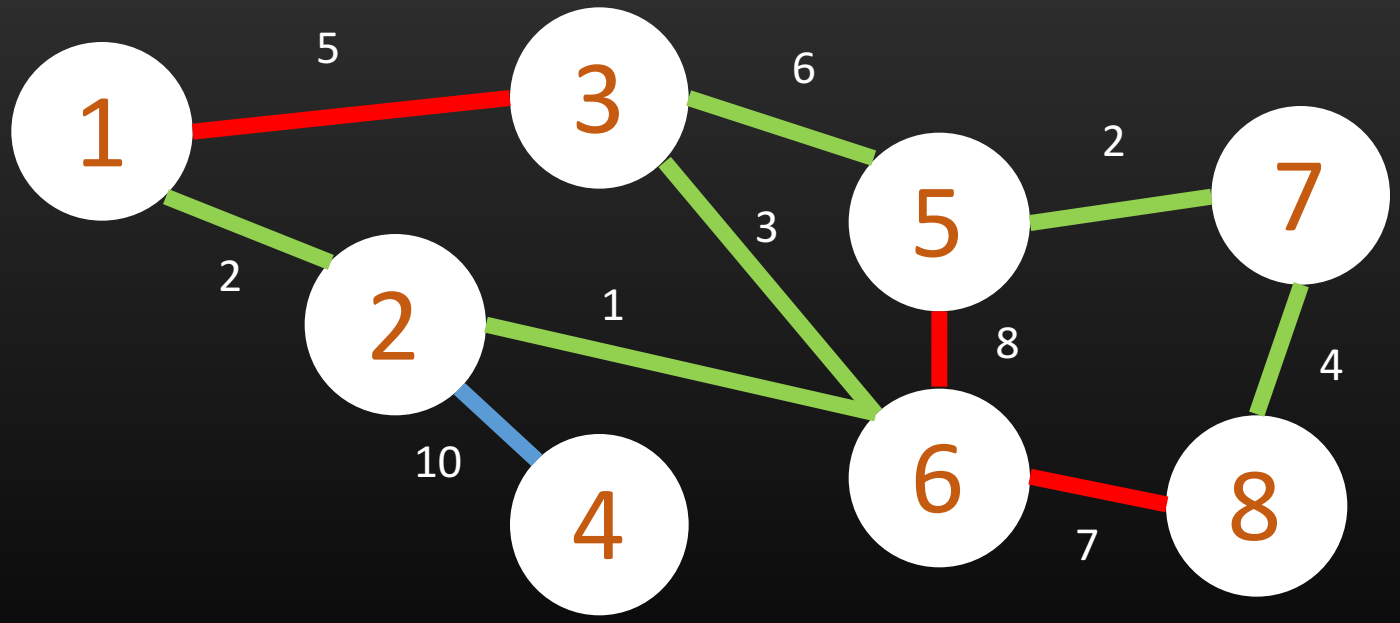
Kruskal’s Algorithm - example



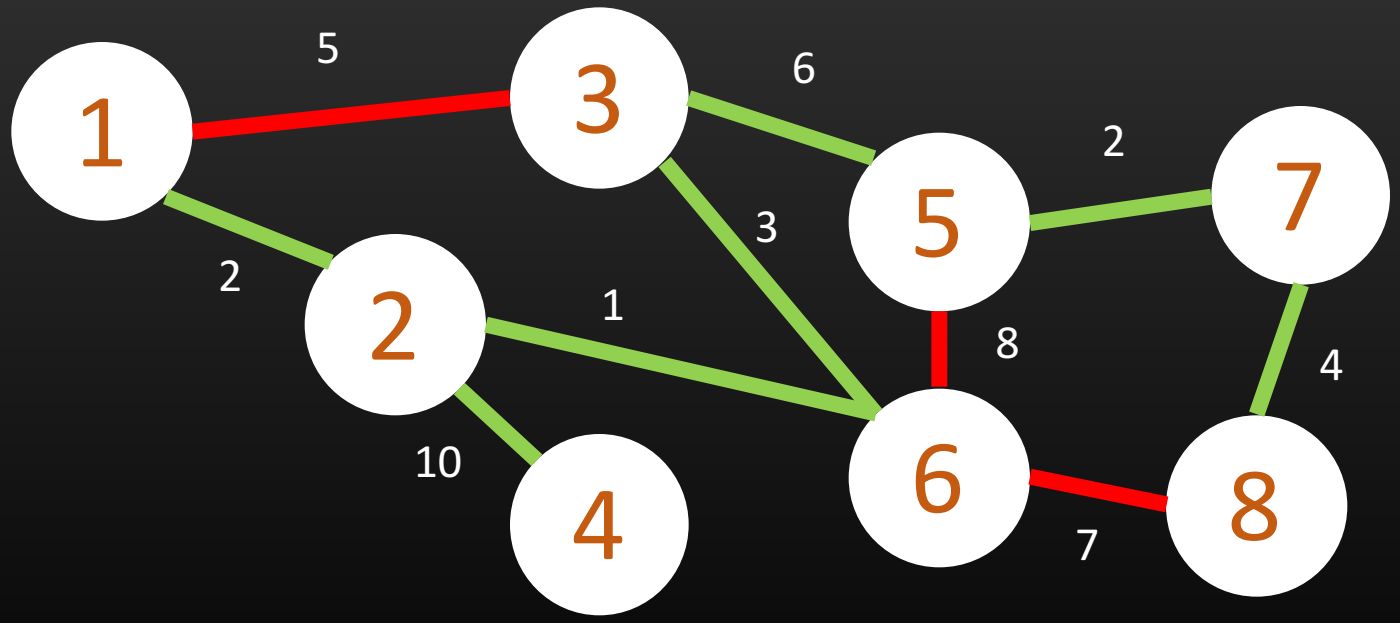
Kruskal’s Algorithm - example



Kruskal’s Algorithm - example



Kruskal’s Algorithm - example



Kruskal's Algorithm

- Use disjoint set to store the components

- Use an array to store the parent of each vertex
- Initially, p[i] = i

- Find(x) : to determine the root of x
- Union(x,y) : join two components into one

```
void initialize(){
    for (int i = 1; i <= n; i++)
        p[i] = i;
}

int find(int x){
    if (p[x] == x) return x;
    else p[x] = find(p[x]);
}

void Union(int x,int y){
    p[find(x)] = find(y);
}
```

Kruskal's Algorithm - pseudocode

- Sort all edges in increasing order of weight
- Initialize p[ ] // set p[i] = i for all i
- For each edge u-v
  - If find(u) != find(v) then // if u and v are not connected yet
    - Mark it as one of the edges of the minimum spanning tree
    - Union(u, v) // connect the component of u with that of v

Kruskal's Algorithm

- Time complexity:

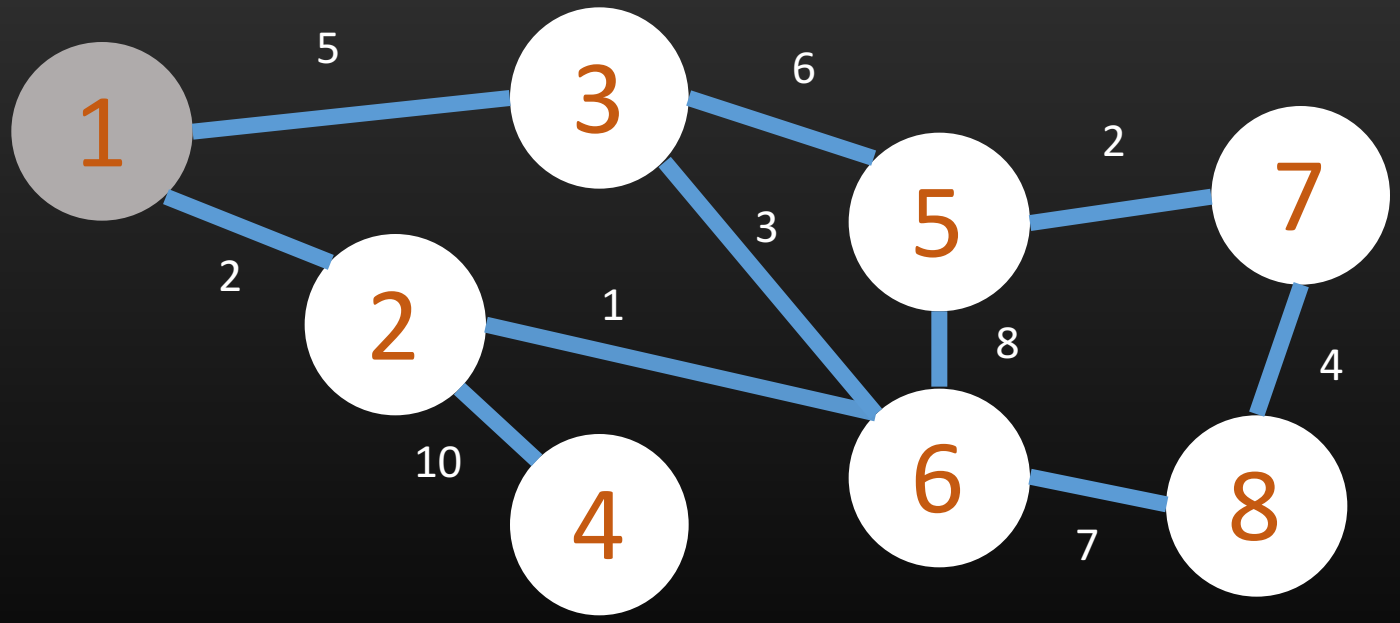
- Sorting edges: $O(E \log E)$
- Select E edges: $O(E)$
- Check subtrees: $O(\alpha(V))$ ($\alpha(n) < 5$ for any practical input size n)
- Total time complexity: $O(E \lg E + E\, \alpha(V))$

Prim’s Algorithm

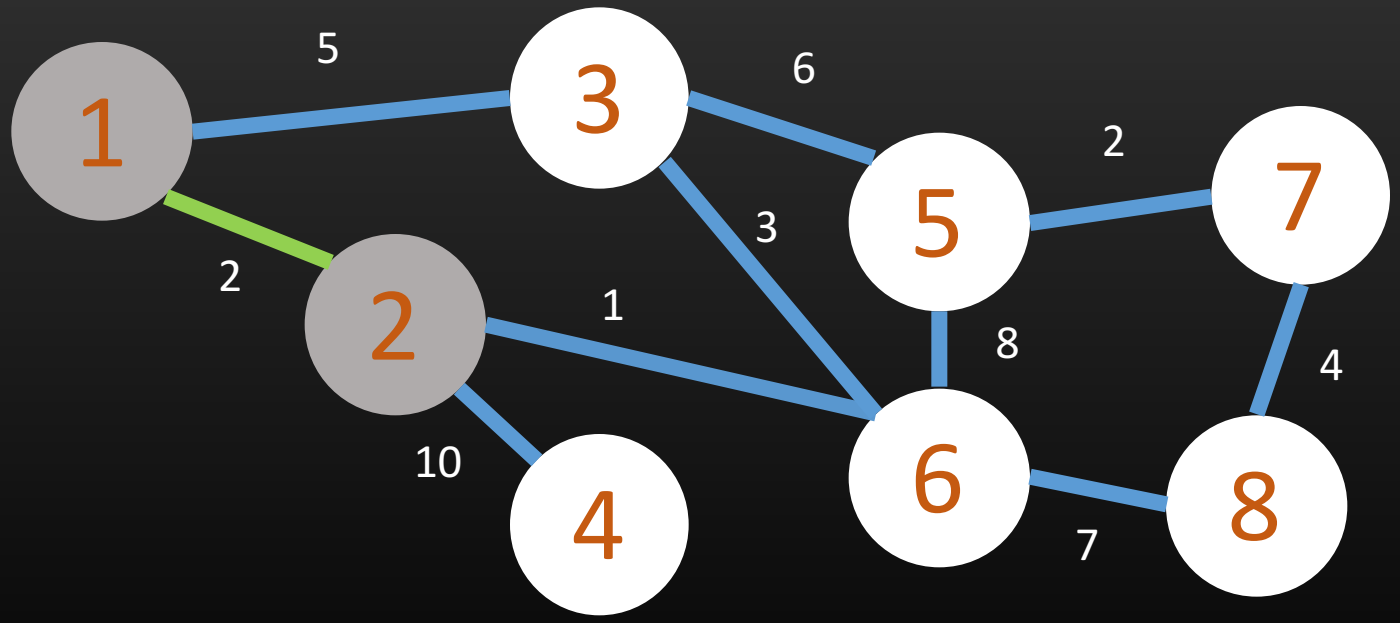
- Greedy algorithm
- Starts from any vertex
- It must be optimal to connect the current tree with the new vertex that is nearest to the tree
- Repeat the process until all vertices are connected

Prim’s Algorithm - example

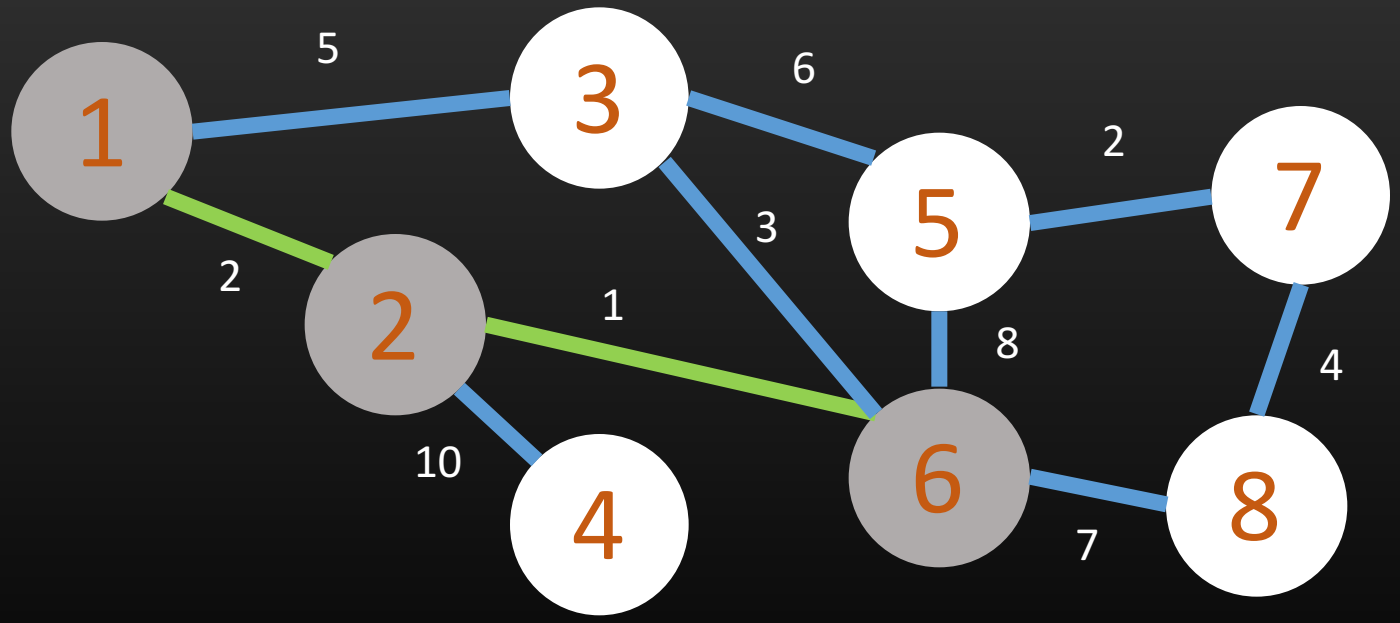
- Start from any node



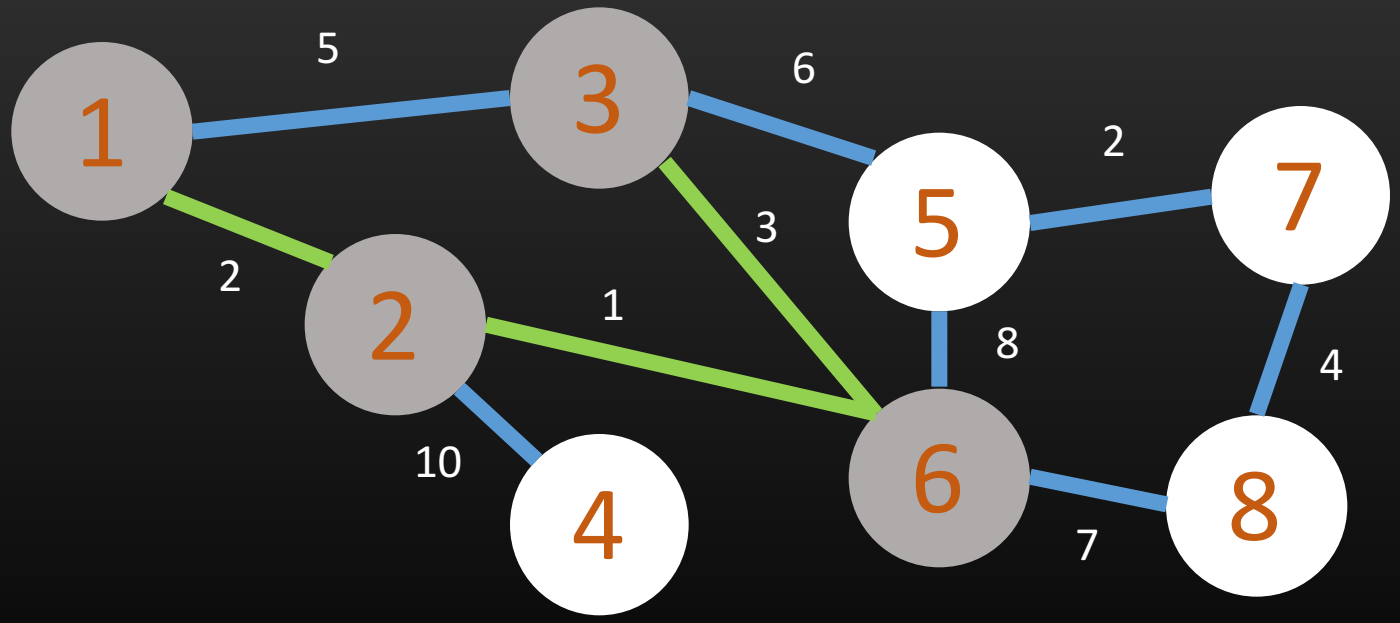
Prim’s Algorithm - example



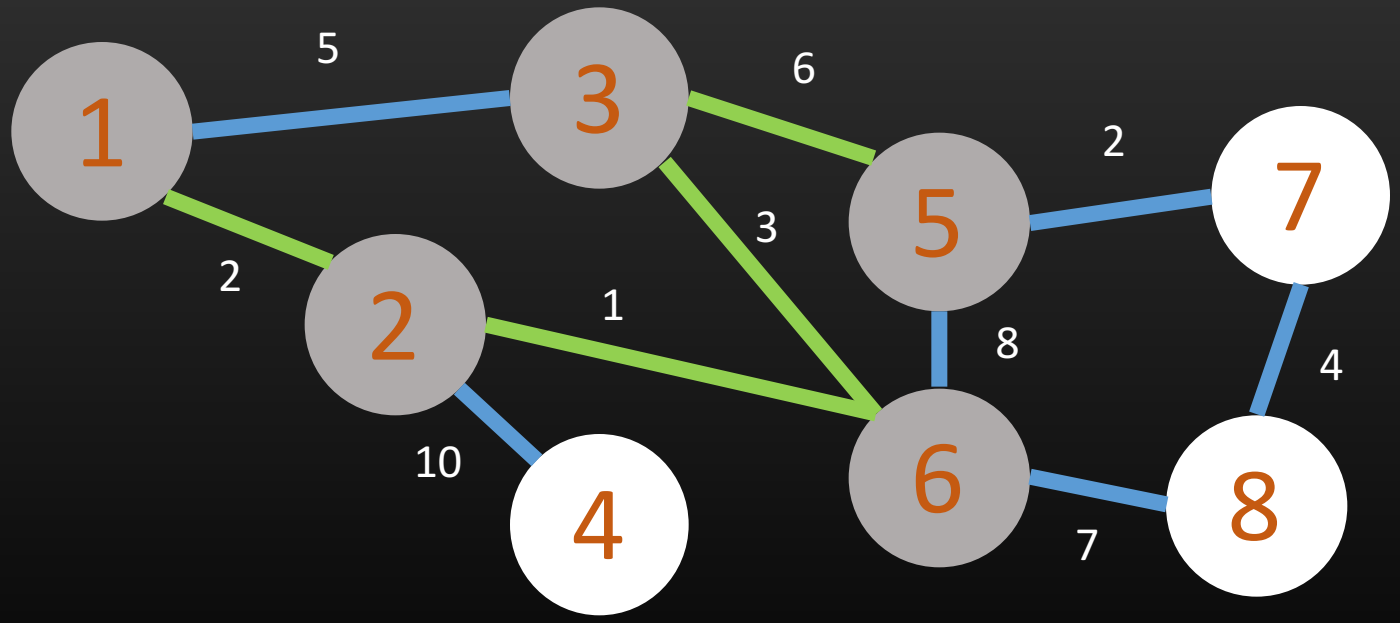
Prim’s Algorithm - example



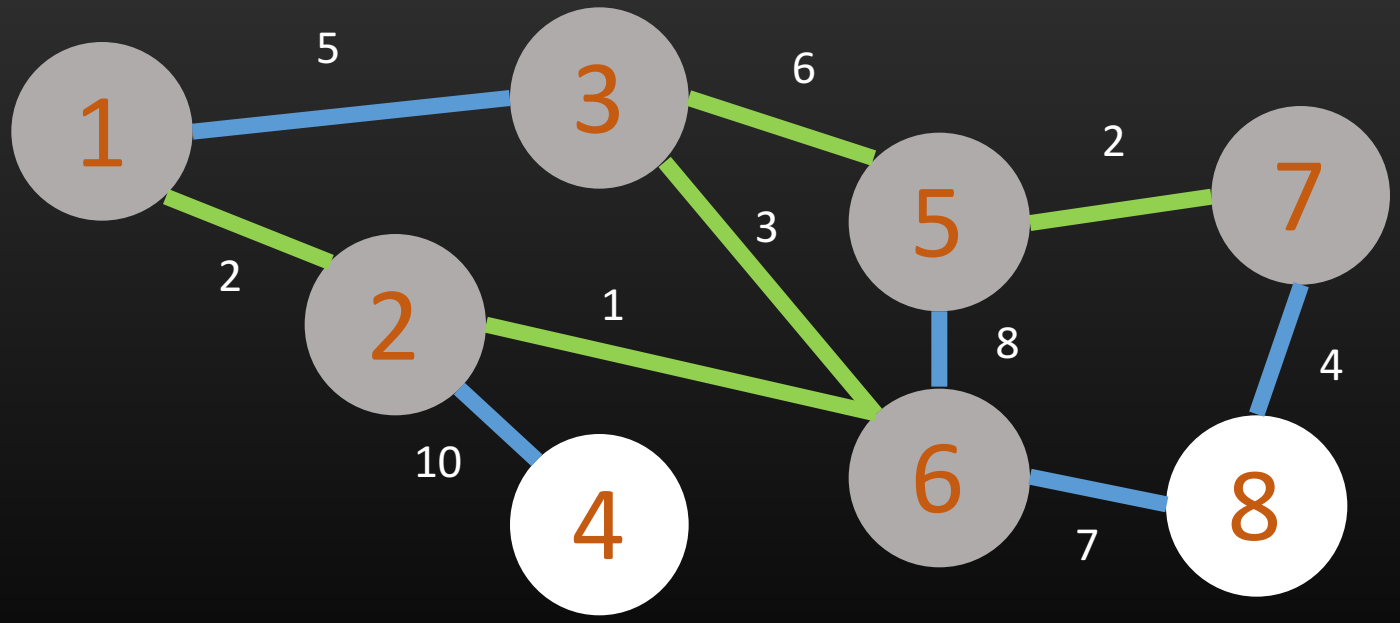
Prim’s Algorithm - example



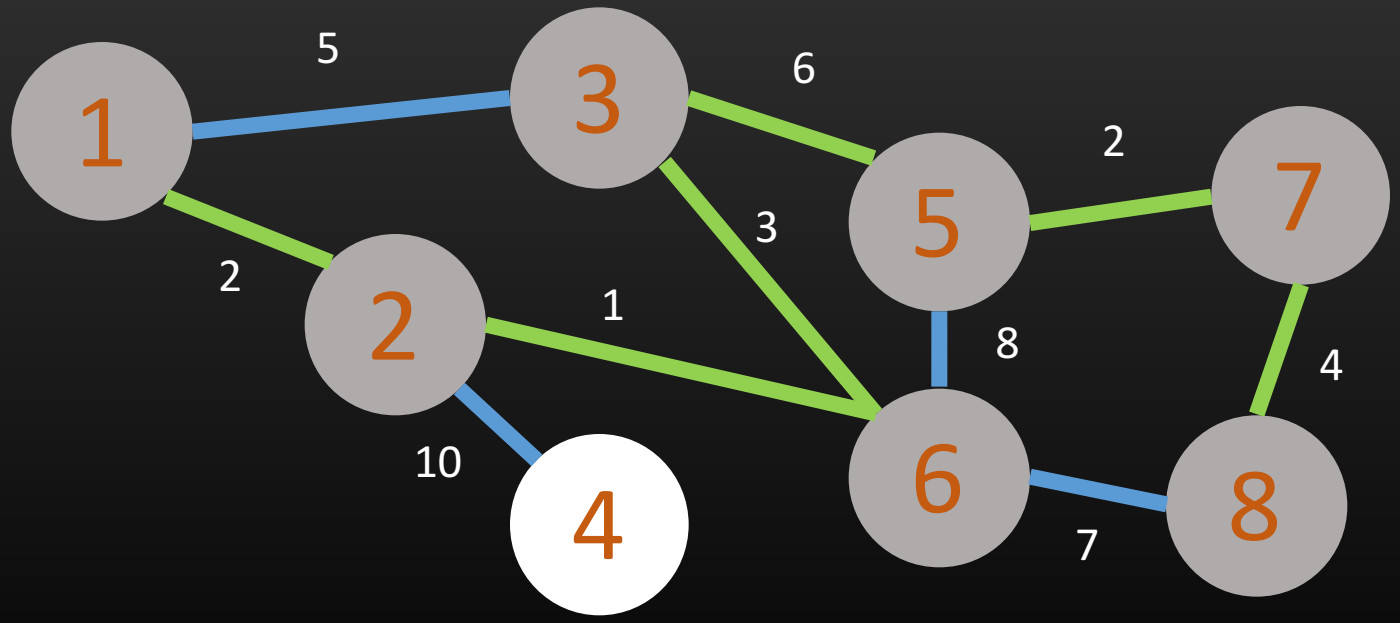
Prim’s Algorithm - example



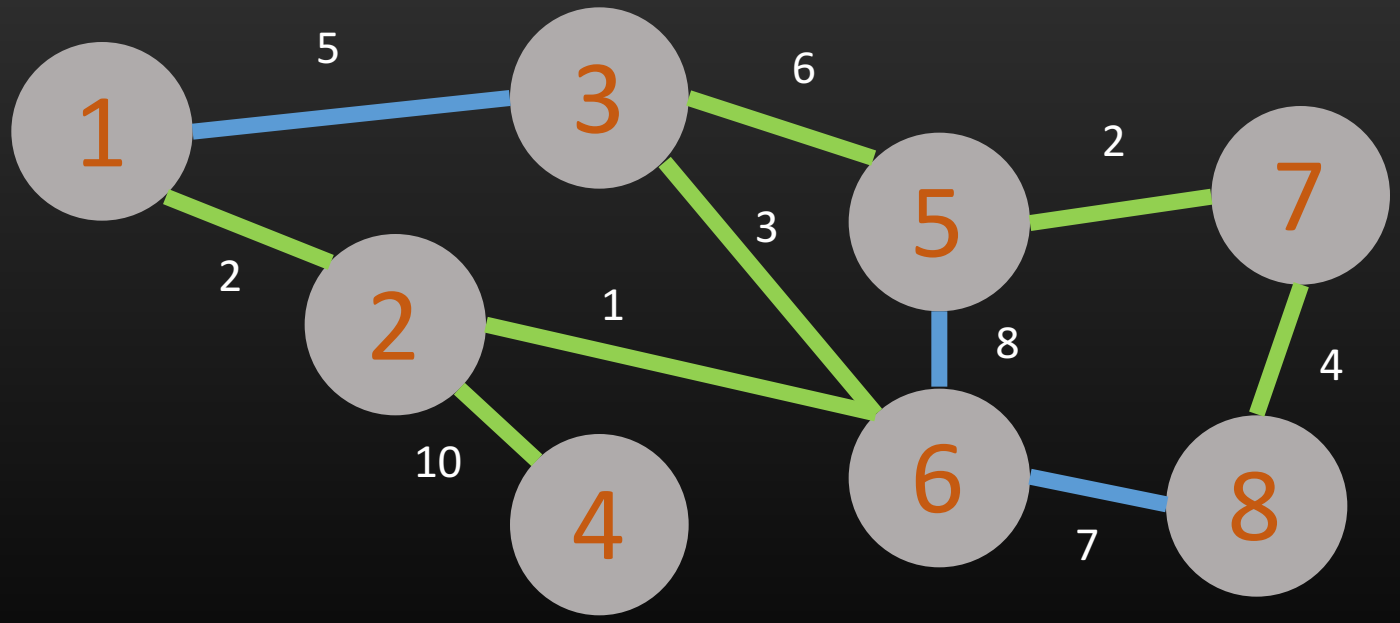
Prim’s Algorithm - example



Prim’s Algorithm - example



Prim’s Algorithm - example



Prim’s Algorithm - pseudocode

```
for each vertex v in the graph
    dist[v] = infinity
    visited[v] = false
dist[root] = 0
u = source
Repeat |V| times :
    visited[u] = true
    mark e[u] as one of the edges of the minimum spanning tree
    for each neighbor v of u do
        if weight of the edge u-v < dist[v] then
            dist[v]=weight of the edge u-v
            e[v]=the edge u-v
    u = vertex that is not visited and has the minimum dist[u]
```

Prim’s Algorithm

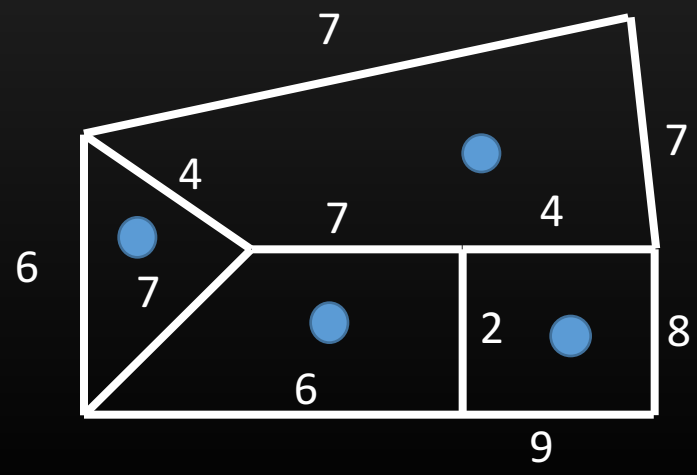
- Similar to Dijkstra’s Algorithm

- Time complexity:
  - $O(|V|^2+|E|)$
  - $O(|E| \log |E|)$ if min-heap is used to find the nearest vertex

Problem

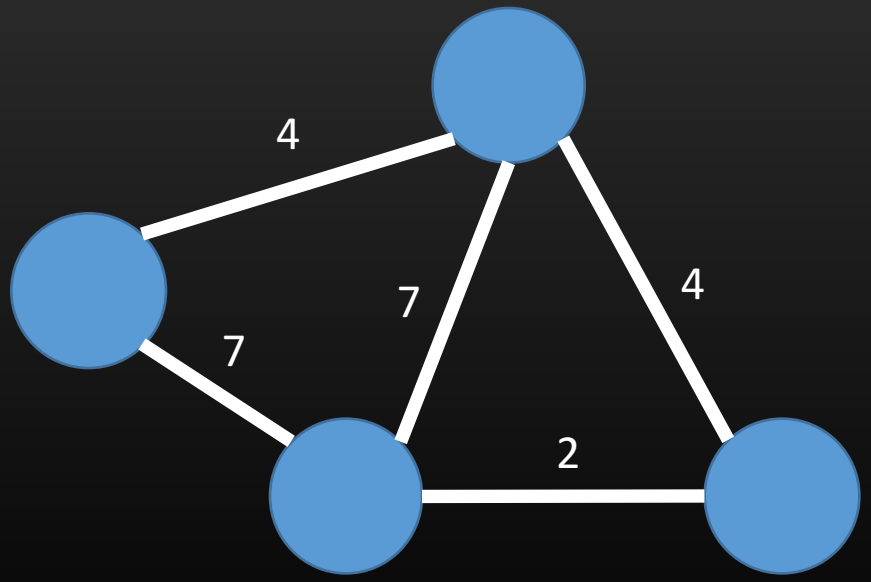
- Animal Farm (CCC 2010 Senior)

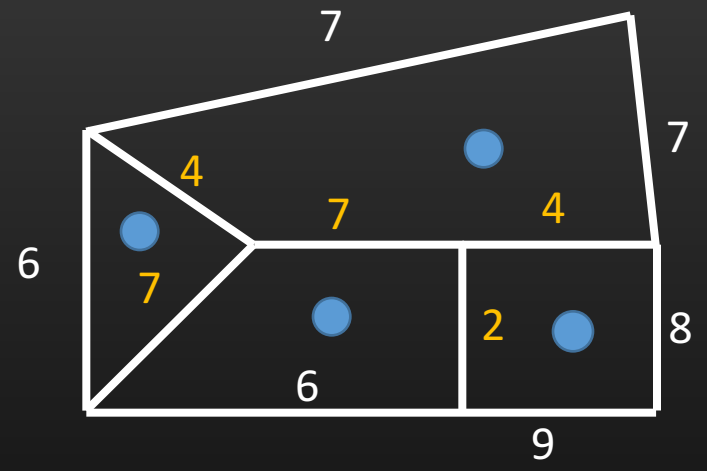
- Some animals are separated by some fences
- Each fence has a cost
- Once a fence has been trampled, any animal may pass over it without incurring any cost
- Determine the minimum cost for all of the animals to meet in the same area by trampling over some fences



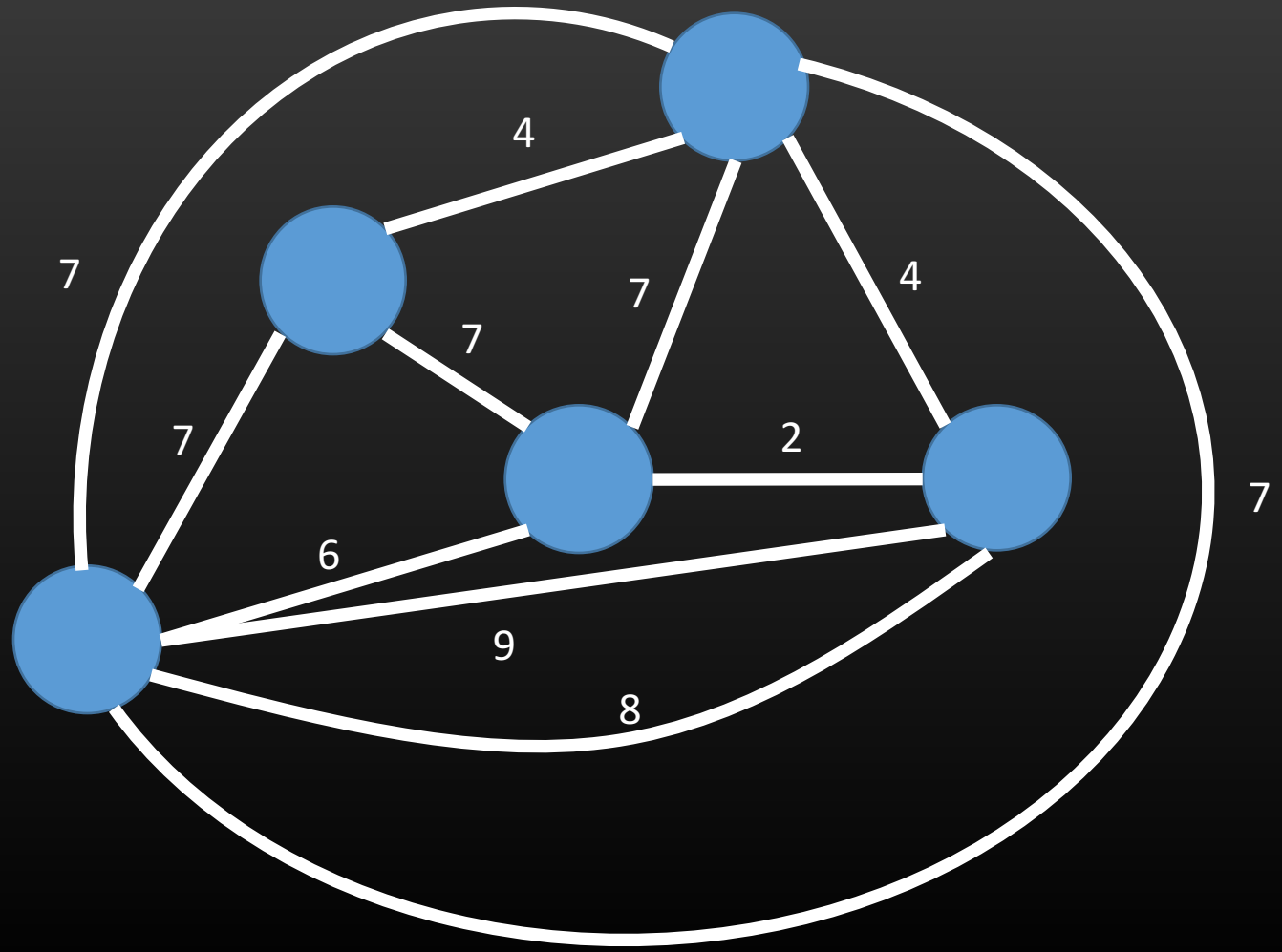
Problem

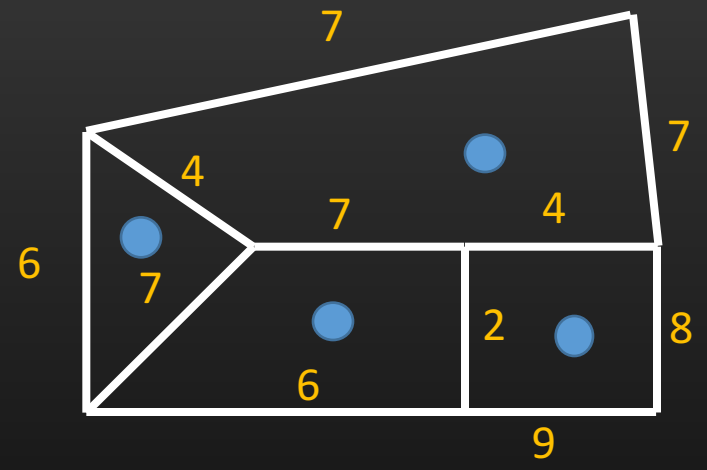
- Minimum Spanning Tree





Problem





Problems

- HKOJ 01041 Shortest Path
- HKOJ M1127 Minimum Spanning Tree
- HKOJ M1223 Lucky Path
- HKOJ T033 Second Trip Discount Scheme