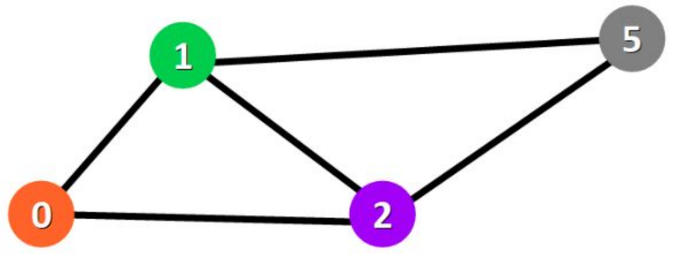
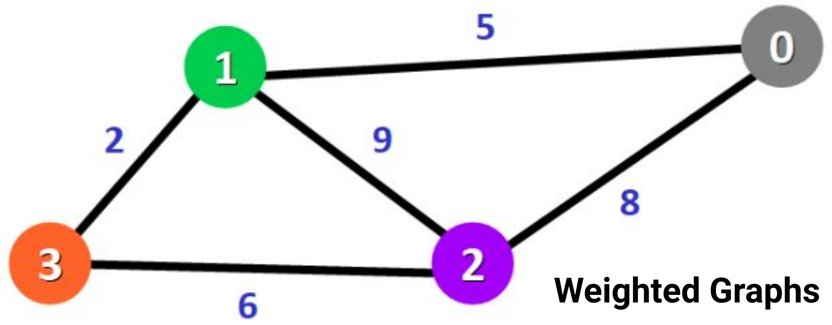
Dijkstra’s Shortest Path Algorithm

Ar. Gör. Elif Ece ERDEM

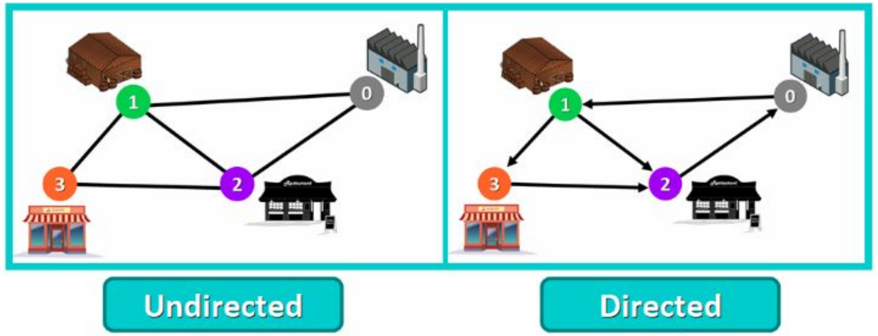
Basic Concepts: Graph

Graphs are data structures used to represent "connections" between pairs of elements.





Weighted Graphs



Purpose & Requirements

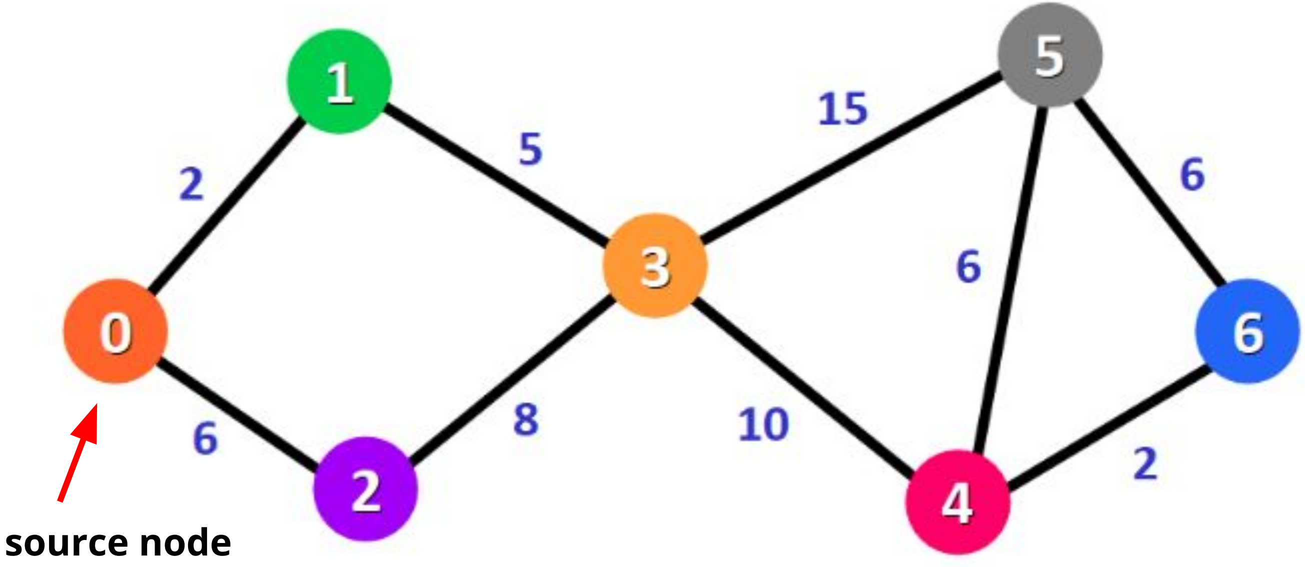
With Dijkstra's Algorithm, you can find the shortest path between nodes in a graph. Particularly, you can **find the shortest path from a node (called the "source node") to all other nodes in the graph**, producing a shortest-path tree.

Dijkstra's Algorithm can only work with graphs that have **positive** weights.

Basics of Dijkstra’s Algorithm

- Dijkstra's Algorithm basically starts at the node that you choose (the source node) and it analyzes the graph to find the shortest path between that node and all the other nodes in the graph.
- The algorithm keeps track of the currently known shortest distance from each node to the source node and it updates these values if it finds a shorter path.
- Once the algorithm has found the shortest path between the source node and another node, that node is marked as "visited" and added to the path.
- The process continues until all the nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node.

Example

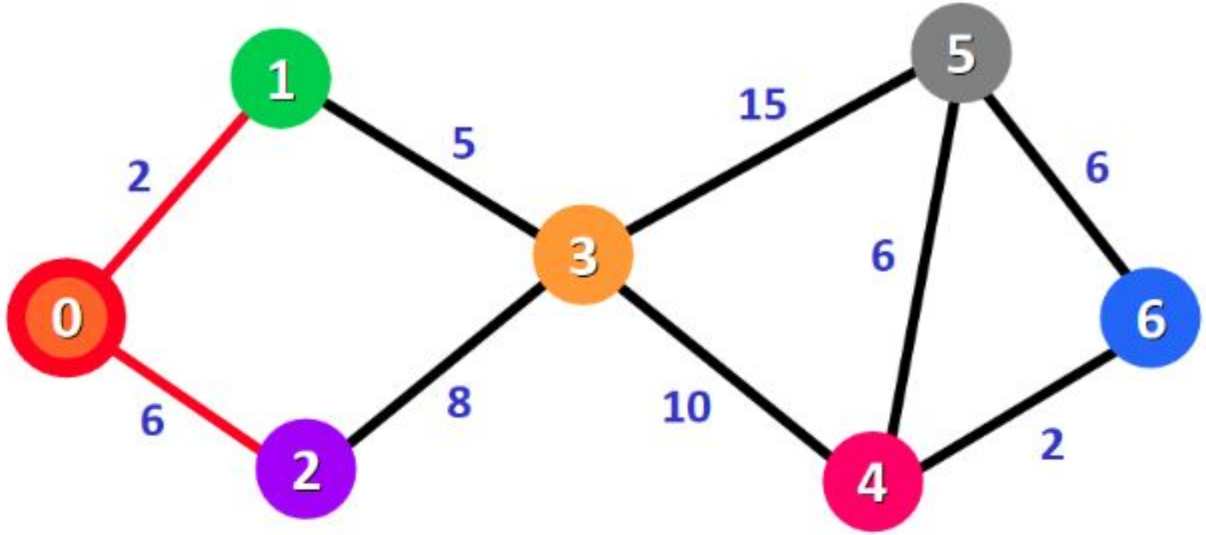


Distance:

0: 0
1: ∞
2: ∞
3: ∞
4: ∞
5: ∞
6: ∞

Unvisited Nodes: {0, 1, 2, 3, 4, 5, 6}

Example



Distance:

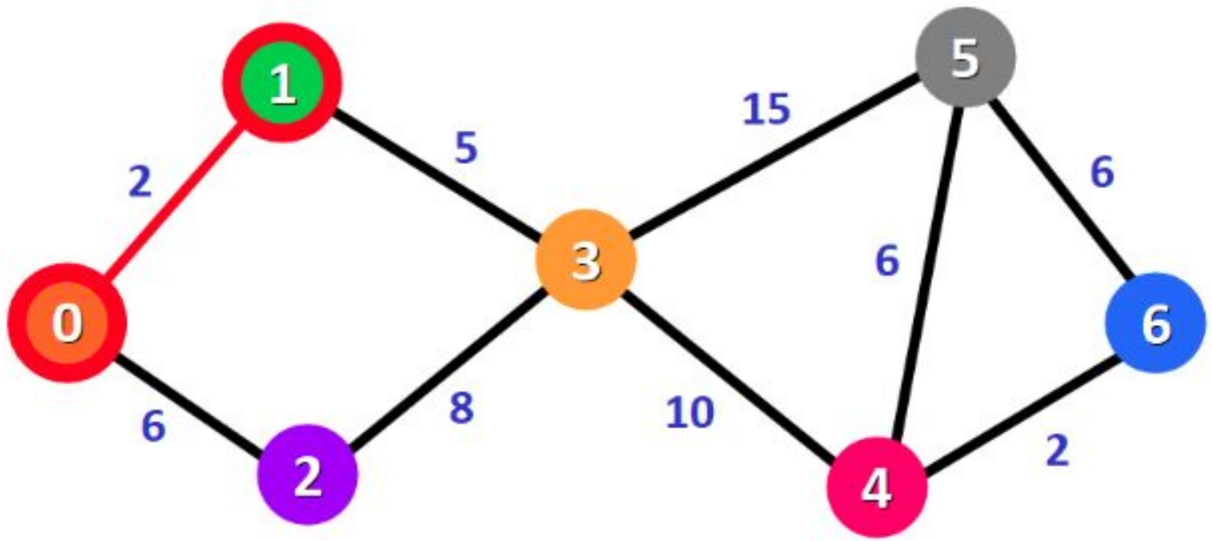
0:	0
1:	~~∞~~ 2
2:	~~∞~~ 6
3:	∞
4:	∞
5:	∞
6:	∞

Unvisited Nodes: {~~0~~, 1, 2, 3, 4, 5, 6}

After updating the distances of the adjacent nodes, we need to:

-> Select the node that is closest to the source node based on the current known distances.
-> Mark it as **visited**.
-> Add it to the path.

Example

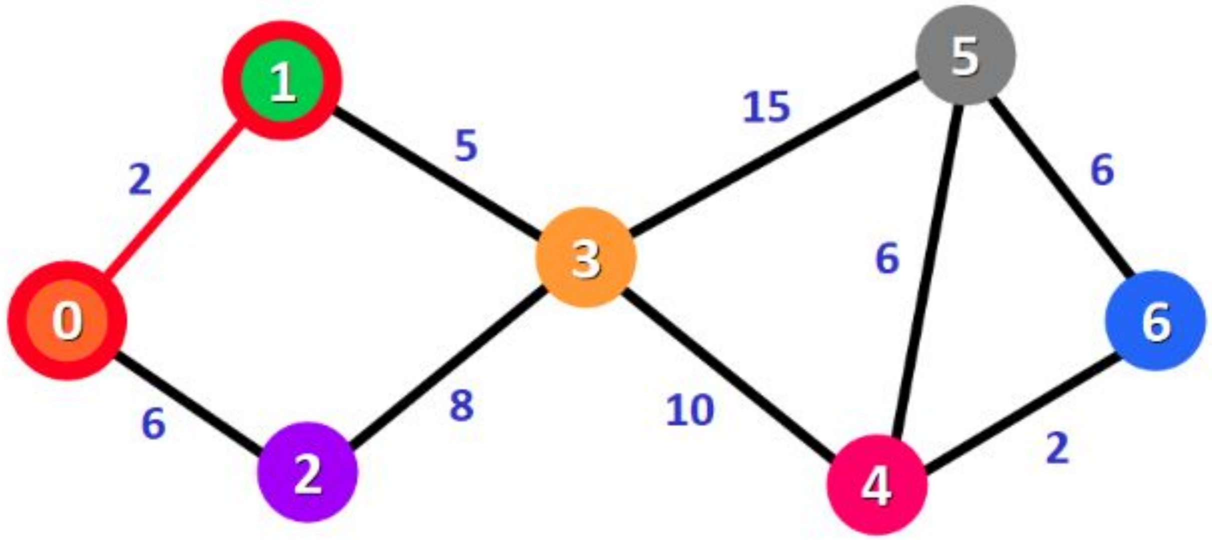


Distance:

0: 0
1: ~~∞~~ **2** ■
2: ~~∞~~ **6**
3: ∞
4: ∞
5: ∞
6: ∞

Unvisited Nodes: {~~0~~, ~~1~~, 2, 3, 4, 5, 6}

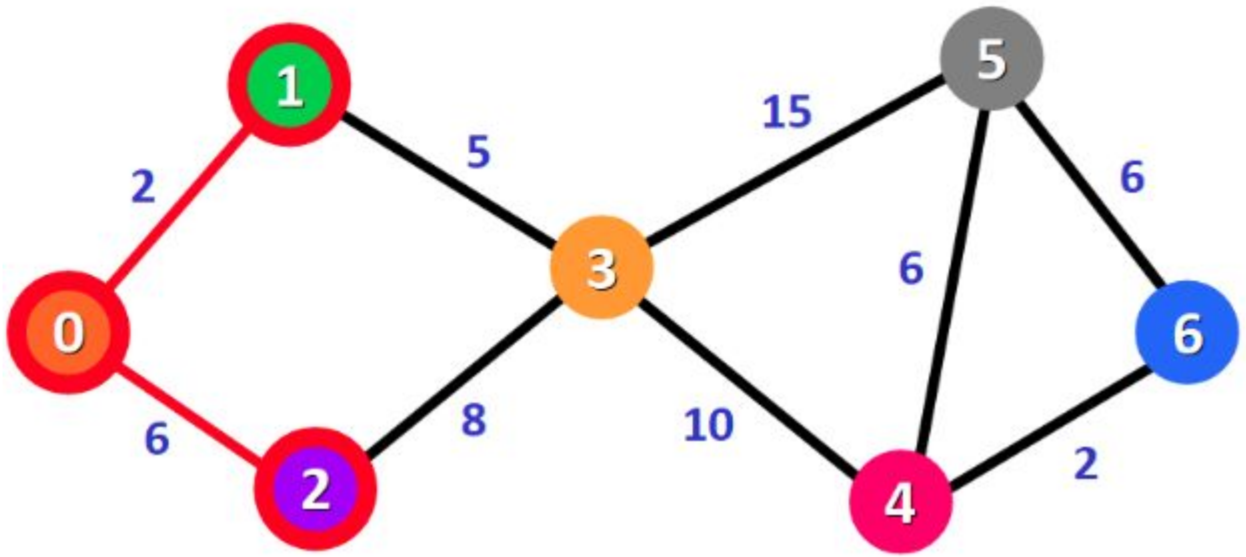
Example



Distance:

0: 0
1: ~~∞~~ 2 ■
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ∞
5: ∞
6: ∞

Example



Distance:

0: 0

1: ~~∞~~ **2** ■

2: ~~∞~~ **6** ■

3: ~~∞~~ **7**

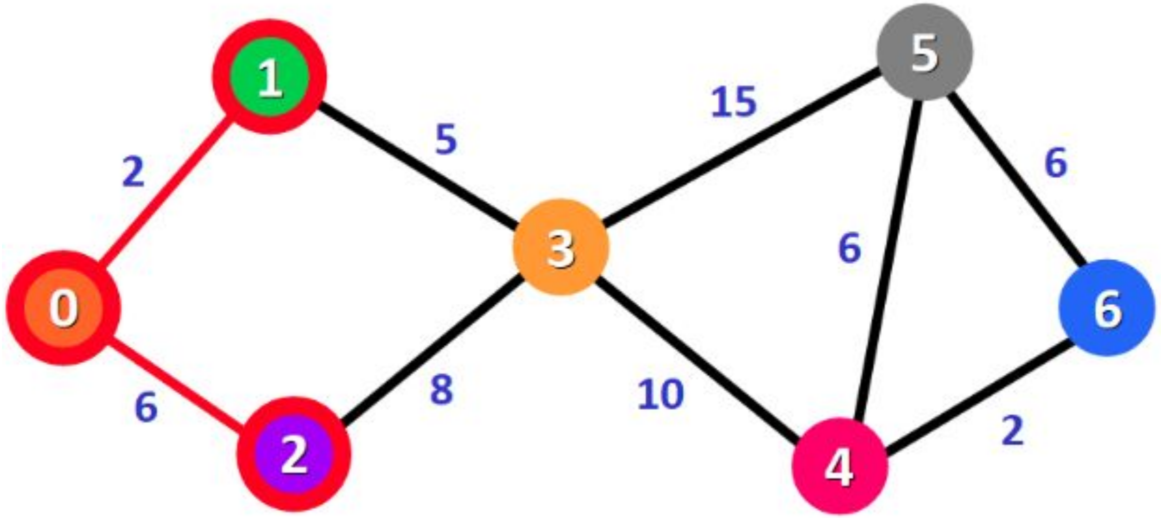
4: ∞

5: ∞

6: ∞

Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, 3, 4, 5, 6}

Example



Distance:

0: 0

1: $\cancel{\infty}$ 2 🟥

2: $\cancel{\infty}$ 6 🟥

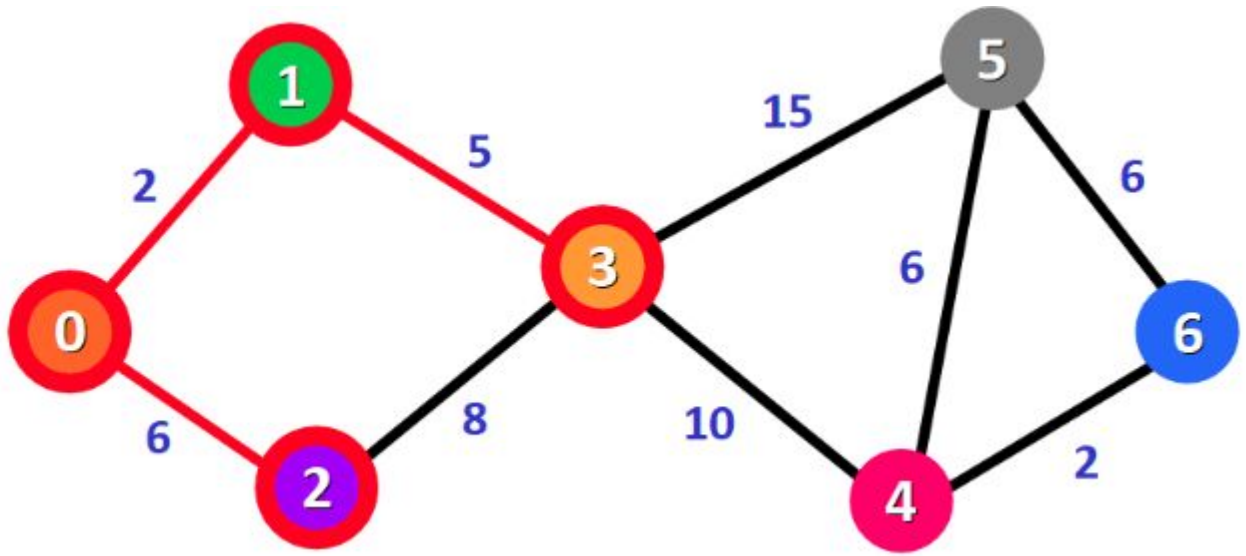
3: $\cancel{\infty}$ 7 from (5 + 2) vs. 14 from (6 + 8)

4: ∞

5: ∞

6: ∞

Example

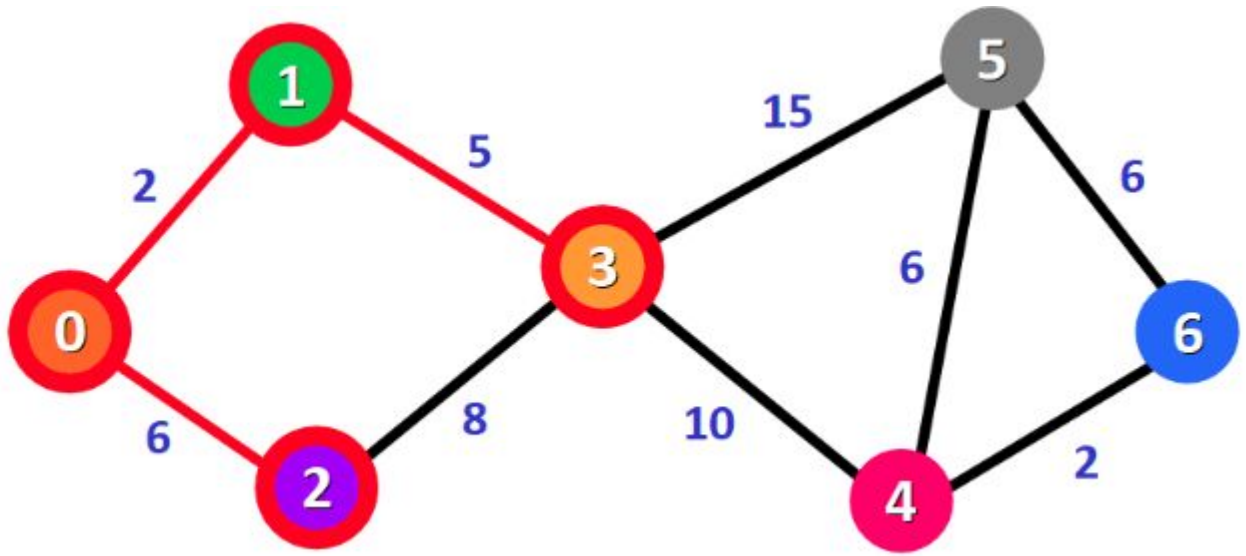


Distance:

0:	0
1:	~~∞~~ **2** 🟥
2:	~~∞~~ **6** 🟥
3:	~~∞~~ **7** 🟥
4:	∞
5:	∞
6:	∞

Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, 4, 5, 6}

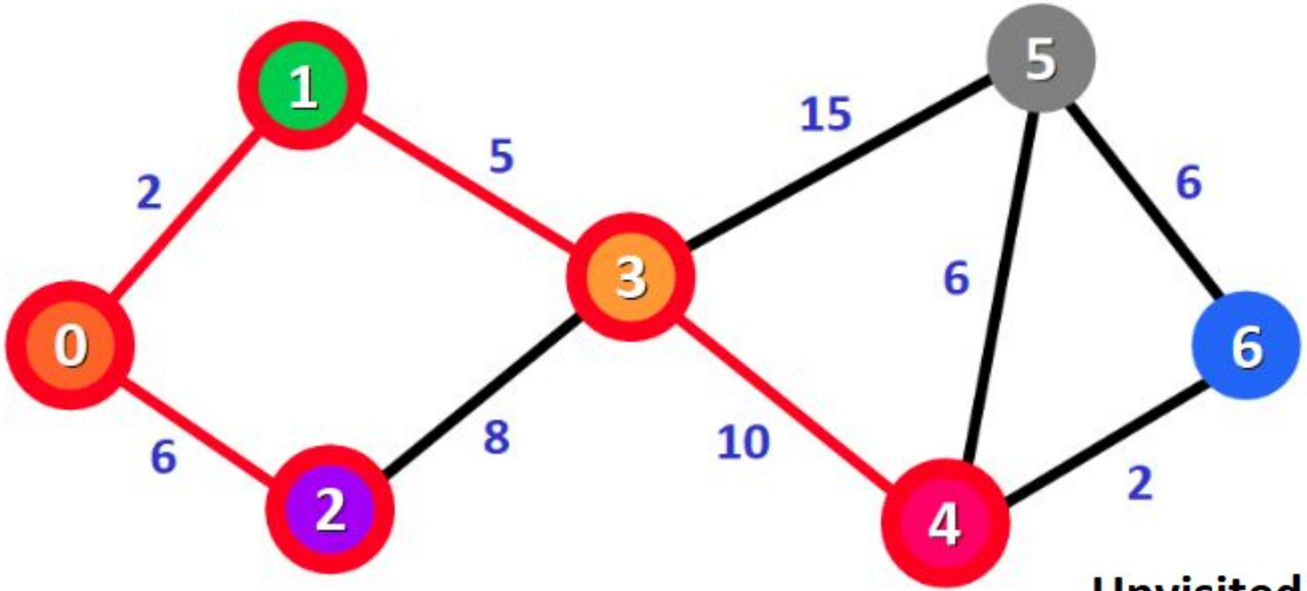
Example



Distance:

0: 0
1: ~~∞~~ 2 ■
2: ~~∞~~ 6 ■
3: ~~∞~~ 7 ■
4: ~~∞~~ 17 from (2 + 5 + 10)
5: ~~∞~~ 22 from (2 + 5 + 15)
6: ∞

Example

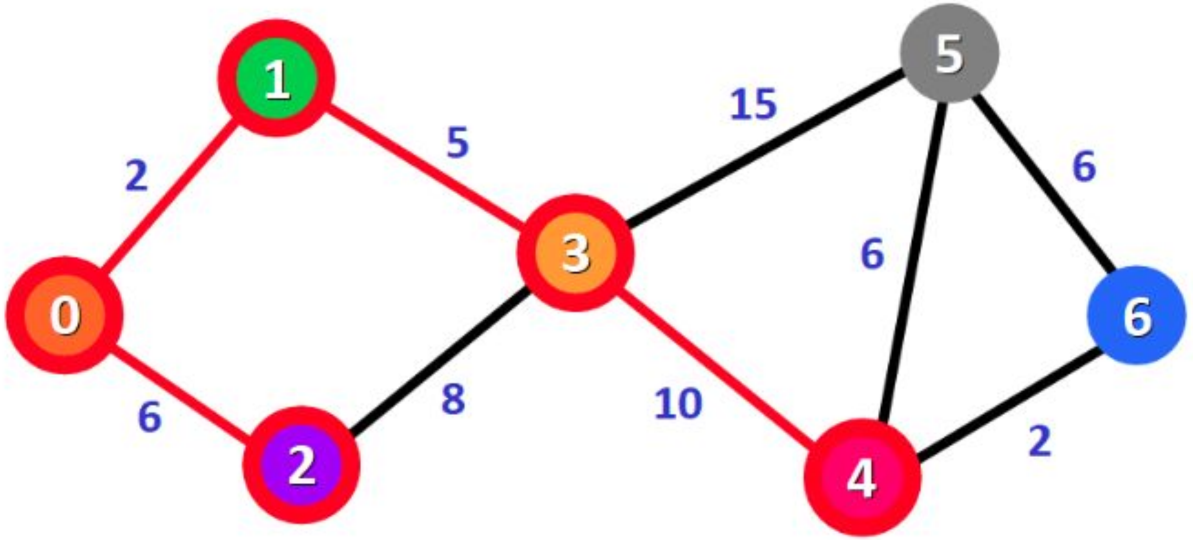


Distance:

0:	0
1:	~~∞~~ 2 🟥
2:	~~∞~~ 6 🟥
3:	~~∞~~ 7 🟥
4:	~~∞~~ 17 🟥
5:	~~∞~~ 22
6:	∞

Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, 6}

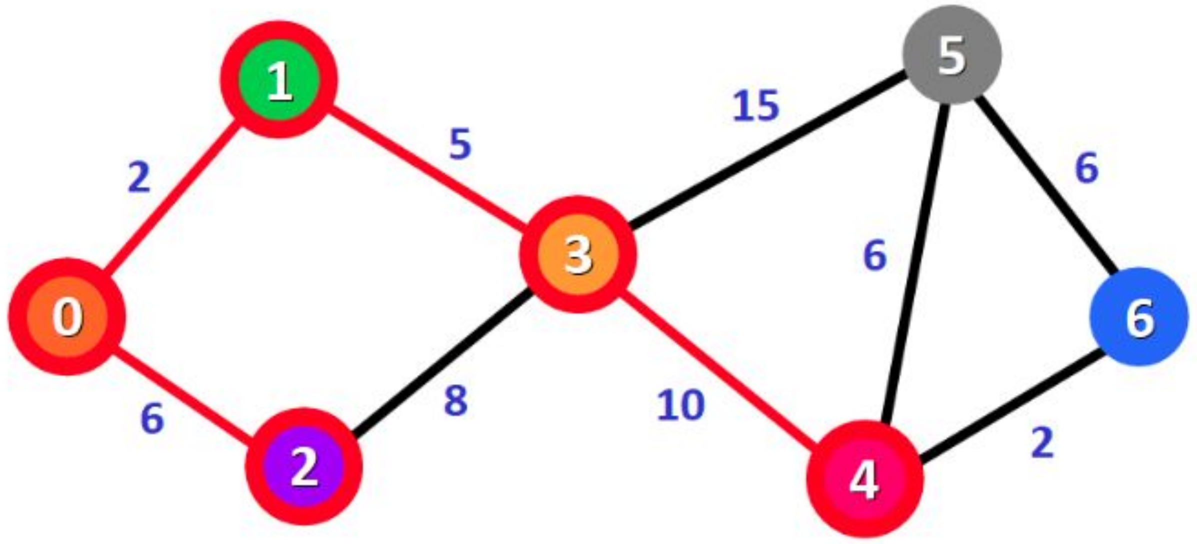
Example



Distance:

0: 0
1: $\cancel{\infty}$ 2 ■
2: $\cancel{\infty}$ 6 ■
3: $\cancel{\infty}$ 7 ■
4: $\cancel{\infty}$ 17 ■
5: $\cancel{\infty}$ 22 vs. 23 (2 + 5 + 10 + 6)
6: $\cancel{\infty}$ 19 from (2 + 5 + 10 + 2)

Example

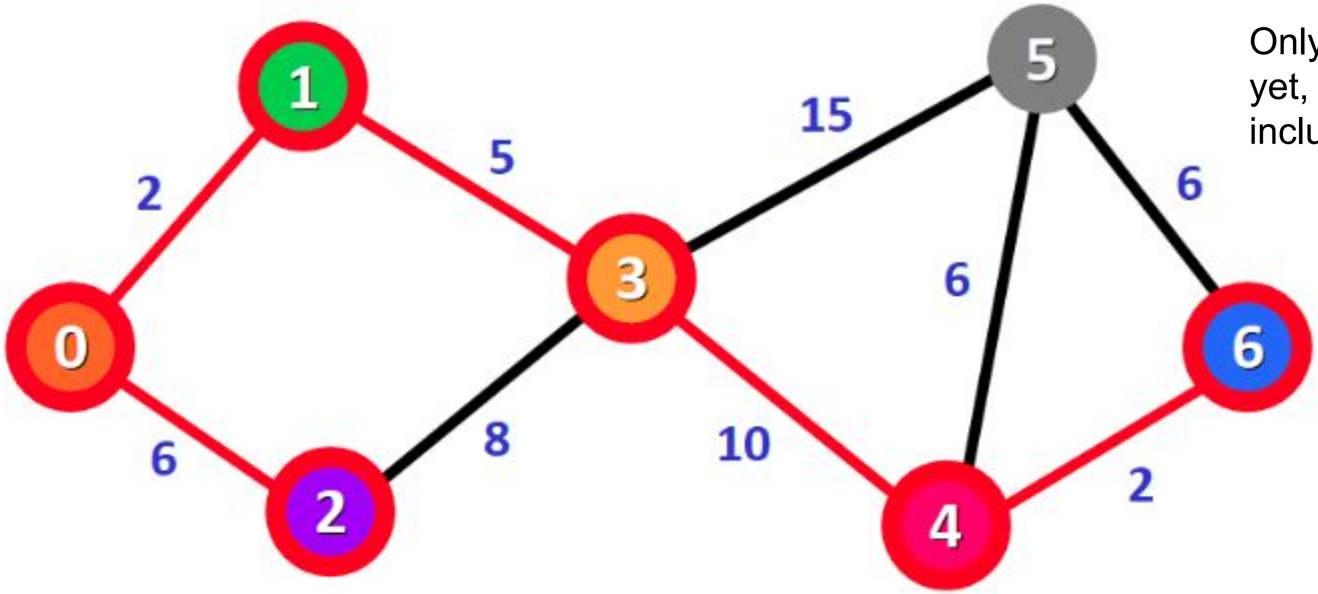


Distance:

0: 0
1: ~~∞~~ 2 ■
2: ~~∞~~ 6 ■
3: ~~∞~~ 7 ■
4: ~~∞~~ 17 ■
5: ~~∞~~ 22
6: ~~∞~~ 19 ■

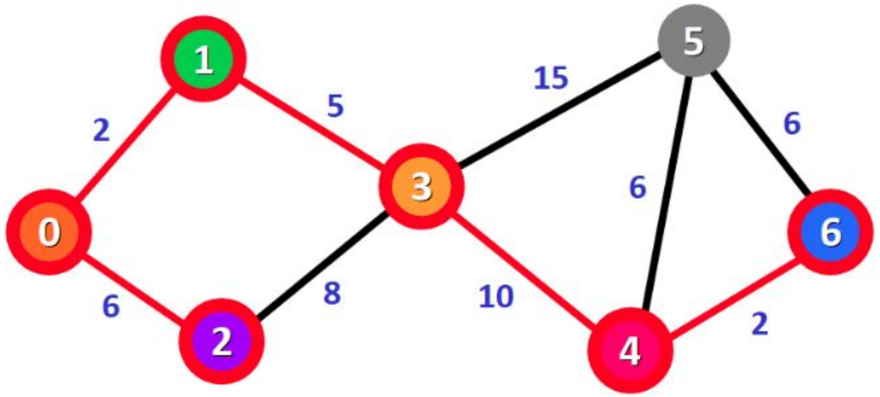
Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, ~~6~~}

Example



Only one node has not been visited yet, node 5. Let's see how we can include it in the path.

Example



There are three different paths that we can take to reach node 5 from the nodes that have been added to the path:

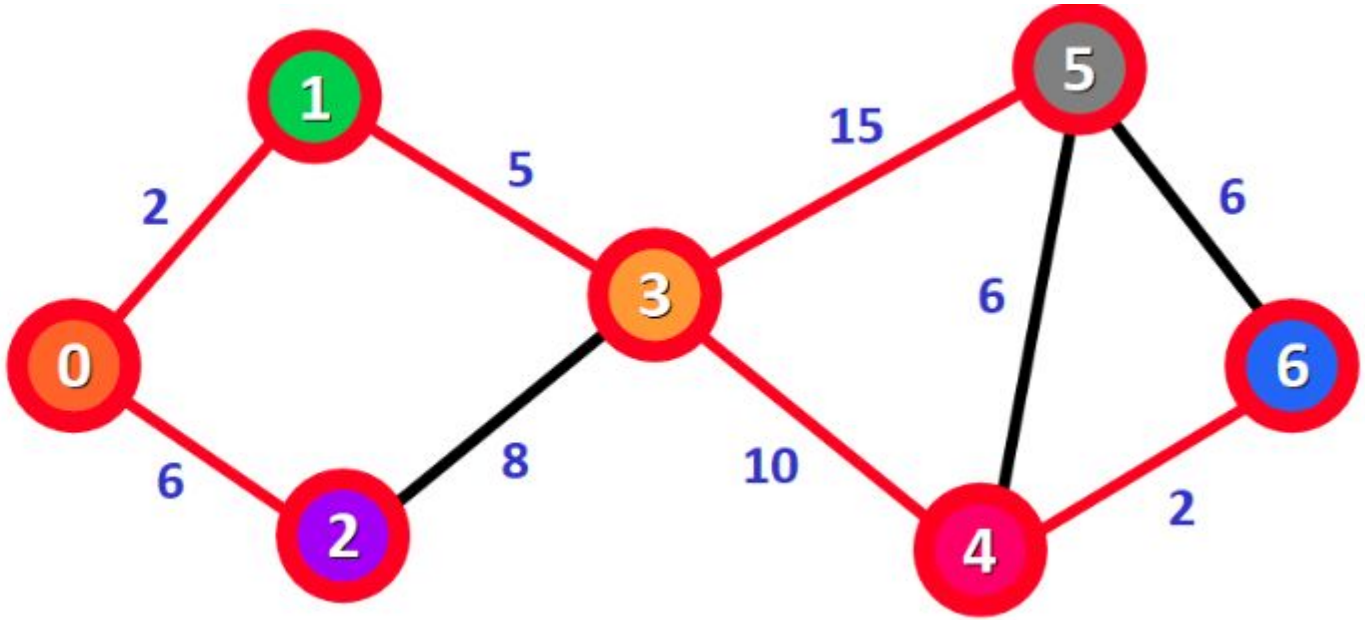
Option 1: 0 -> 1 -> 3 -> 5 with a distance of 22 (2 + 5 + 15).

Option 2: 0 -> 1 -> 3 -> 4 -> 5 with a distance of 23 (2 + 5 + 10 + 6).

Option 3: 0 -> 1 -> 3 -> 4 -> 6 -> 5 with a distance of 25 (2 + 5 + 10 + 2 + 6)

We select the shortest path: 0 -> 1 -> 3 -> 5 with a distance of 22.

Example



Distance:

0:	0
1:	~~∞~~ 2
2:	~~∞~~ 6
3:	~~∞~~ 7
4:	~~∞~~ 17
5:	~~∞~~ 22
6:	~~∞~~ 19

For example, if you want to reach node 6 starting from node 0, you just need to follow the red edges and you will be following the shortest path 0 -> 1 -> 3 -> 4 - > 6 automatically.

Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~}