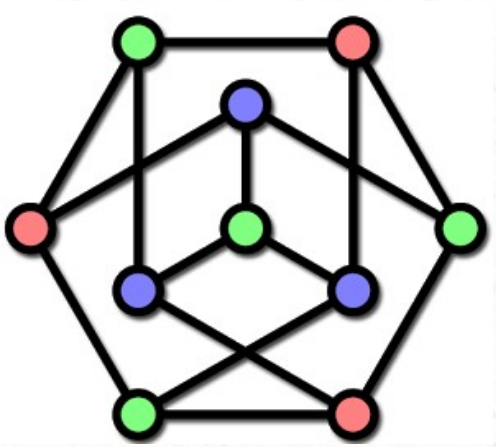
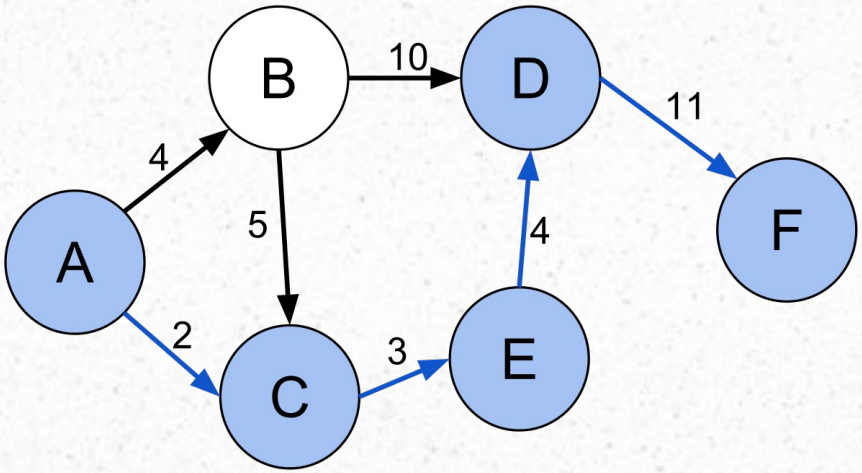


APPLIED GRAPH THEORY AND ALGORITHMS (CSC4066)

Dr. Hasin A Ahmed
Assistant Professor
Department of Computer Science
Gauhati University

Shortest Path Problem

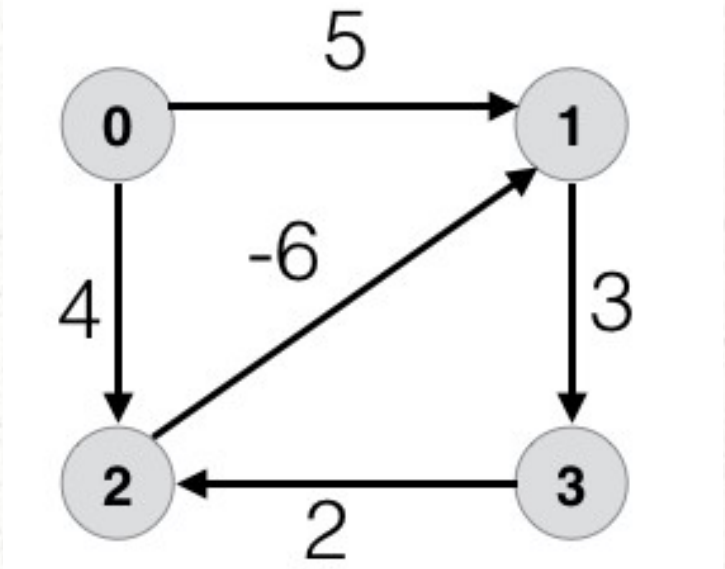


shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized

Versions

- Single source shortest path problem
- Single destination shortest path problem
- All pair shortest path problem

Negative Cycle

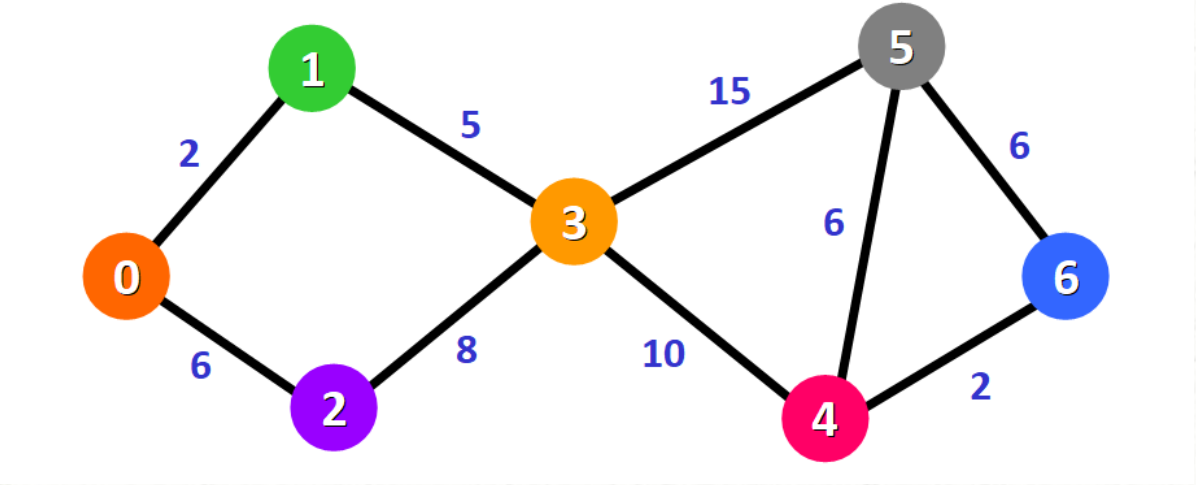


A negative cycle is one in which the overall sum of the cycle becomes negative

Algorithms

- Dijkstra's algorithm solves the single-source shortest path problem with non-negative edge weight.
- Bellman–Ford algorithm solves the single-source problem if edge weights may be negative.
- A* search algorithm solves for single-pair shortest path using heuristics to try to speed up the search.
- Floyd–Warshall algorithm solves all pairs shortest paths.
- Johnson's algorithm solves all pairs shortest paths, and may be faster than Floyd–Warshall on sparse graphs.

Dijkstra's algorithm

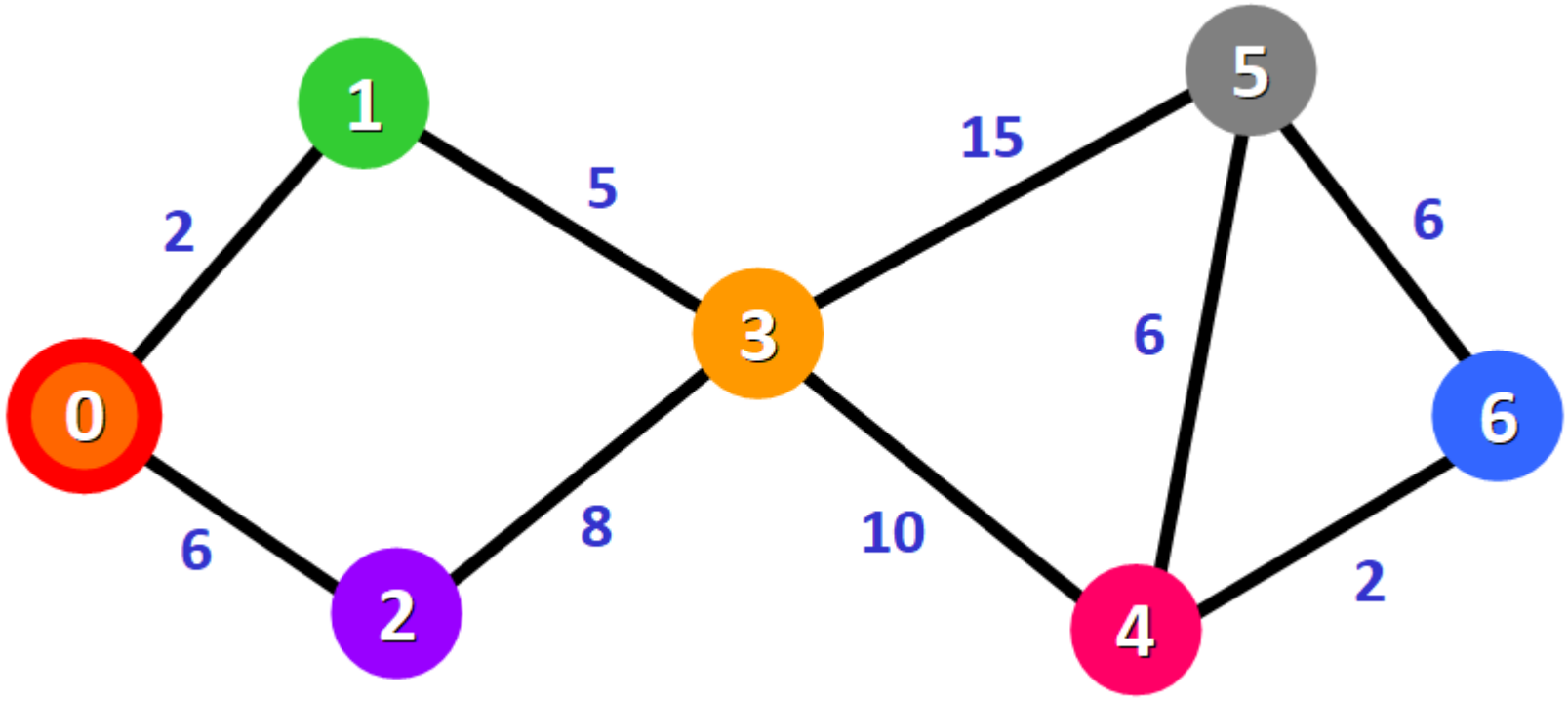


Distance:

0: 0
1: ∞
2: ∞
3: ∞
4: ∞
5: ∞
6: ∞

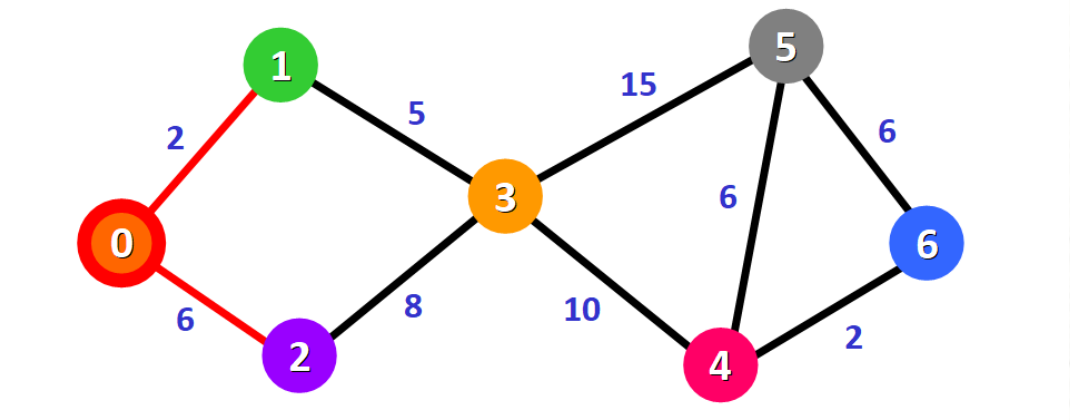
Unvisited Nodes: {0, 1, 2, 3, 4, 5, 6}

Dijkstra's Algorithm



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

Dijkstra's Algorithm



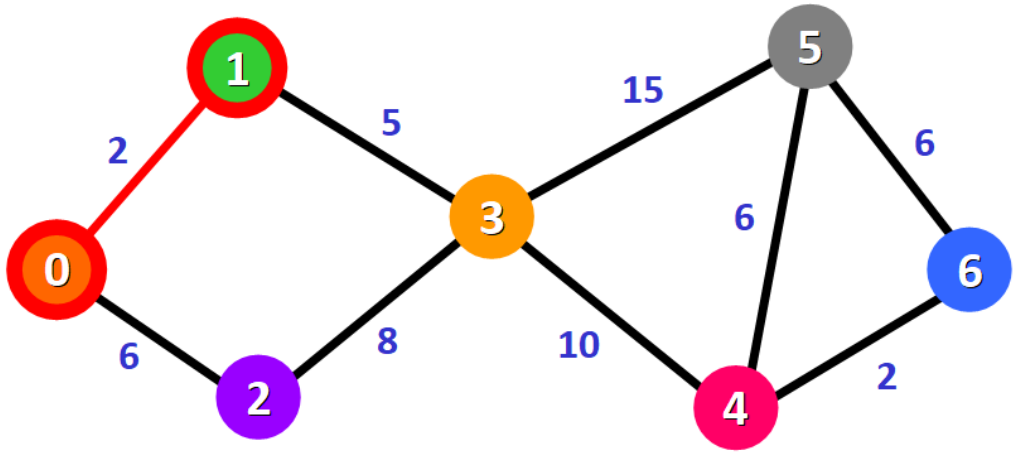
Distance:

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

Dijkstra's Algorithm

If(d(x) + c(x, y) < d(y))
Then we update d(y) = d(x) + c(x, y)

Dijkstra's Algorithm



Distance:

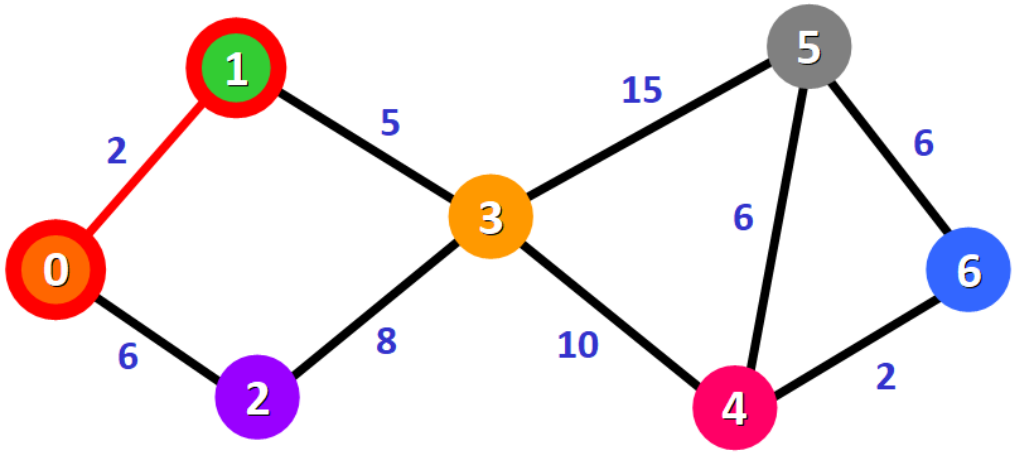
0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ∞
4: ∞
5: ∞
6: ∞

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

Dijkstra's Algorithm

If(d(x) + c(x, y) < d(y))
Then we update d(y) = d(x) + c(x, y)

Dijkstra'a Algorithm

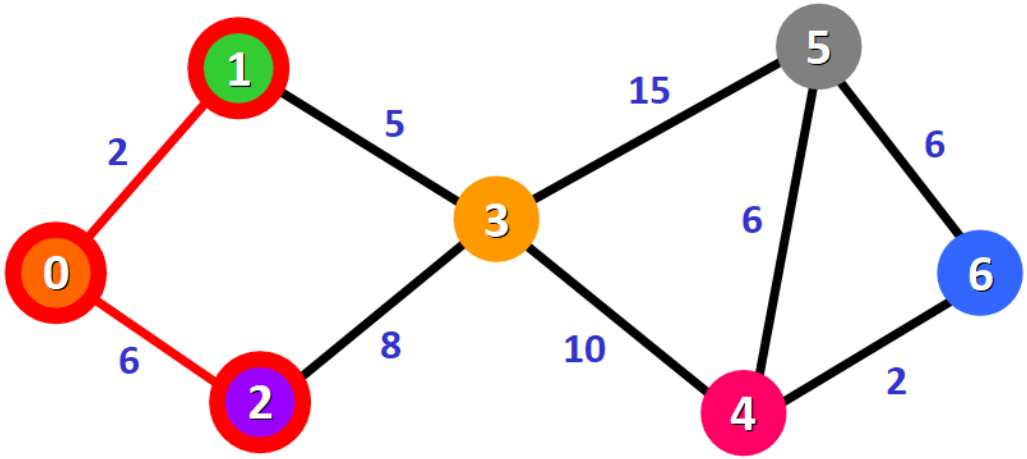


Distance:

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

For node 3, the total distance is 7 because we add the weights of the edges that form the path 0 -> 1 -> 3 (2 for the edge 0 -> 1 and 5 for the edge 1 -> 3).

Dijkstra's Algorithm



Distance:

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

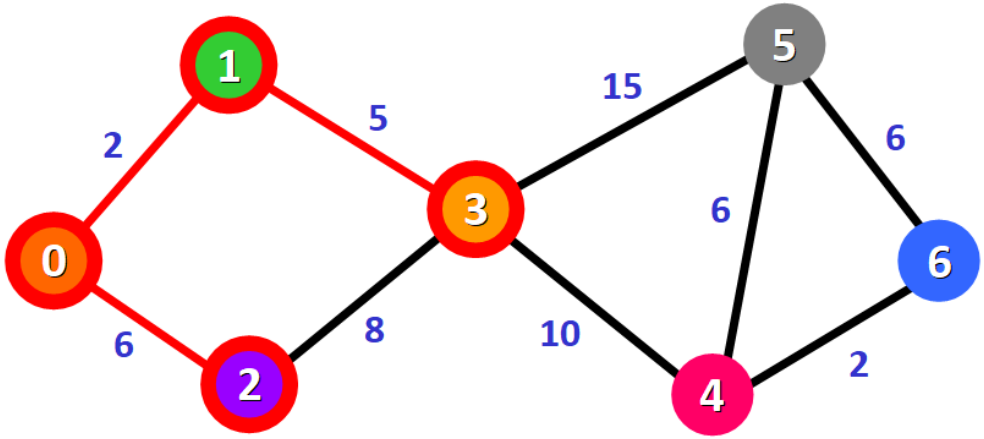
Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, 3, 4, 5, 6}

Dijkstra's Algorithm

Distance:

0: 0
1: ~~∞~~ **2**
2: ~~∞~~ **6**
3: ~~∞~~ **7 from (5 + 2) vs. 14 from (6 + 8)**
4: ∞
5: ∞
6: ∞

Dijkstra's Algorithm

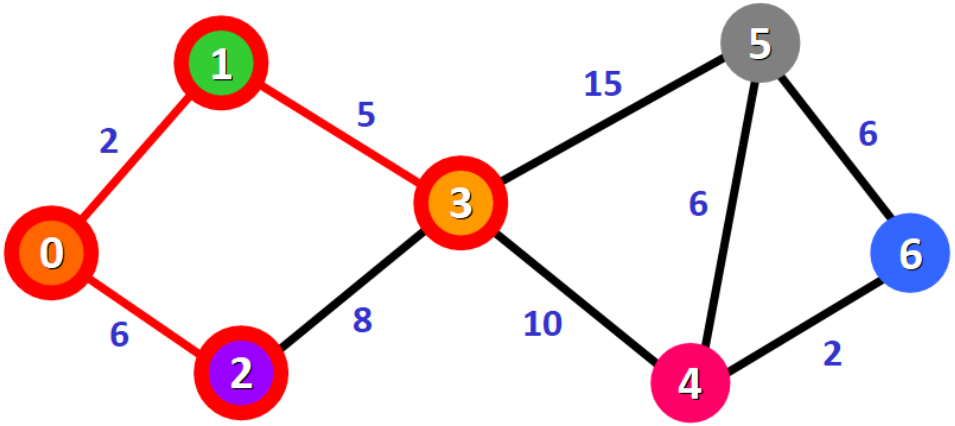


Distance:

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

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

Dijkstra's Algorithm

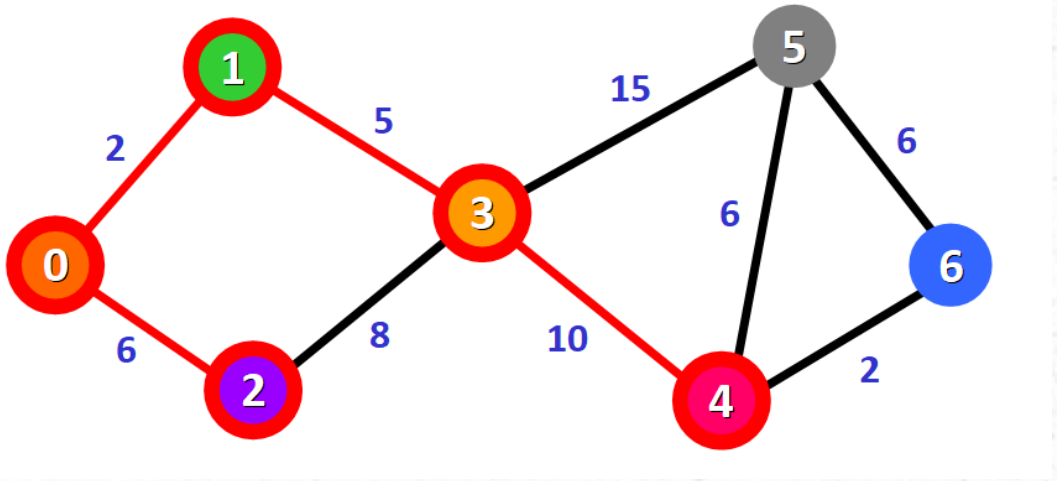


Distance:

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

For node 4: the distance is 17 from the path 0 -> 1 -> 3 -> 4.
For node 5: the distance is 22 from the path 0 -> 1 -> 3 -> 5.

Dijkstra's Algorithm

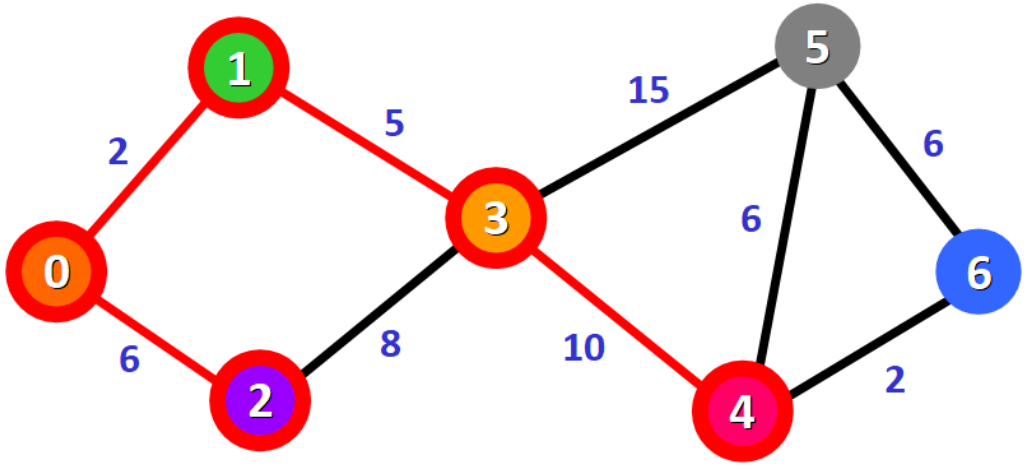


Distance:

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

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

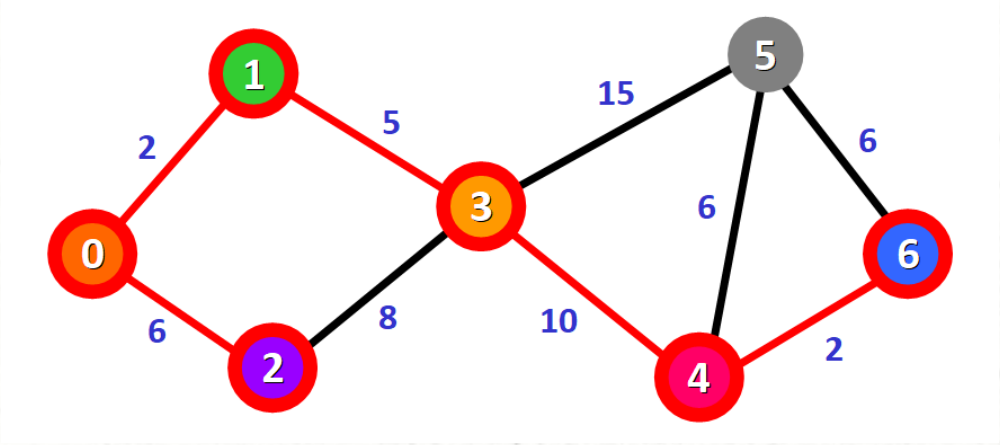
Dijkstra's Algorithm



Distance:

0: 0
1: ~~∞~~ 2
2: ~~∞~~ 6
3: ~~∞~~ 7
4: ~~∞~~ 17
5: ~~∞~~ 22 vs. 23 (2 + 5 + 10 + 6)
6: ~~∞~~ 19 from (2 + 5 + 10 + 2)

Dijkstra's Algorithm



Distance:

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

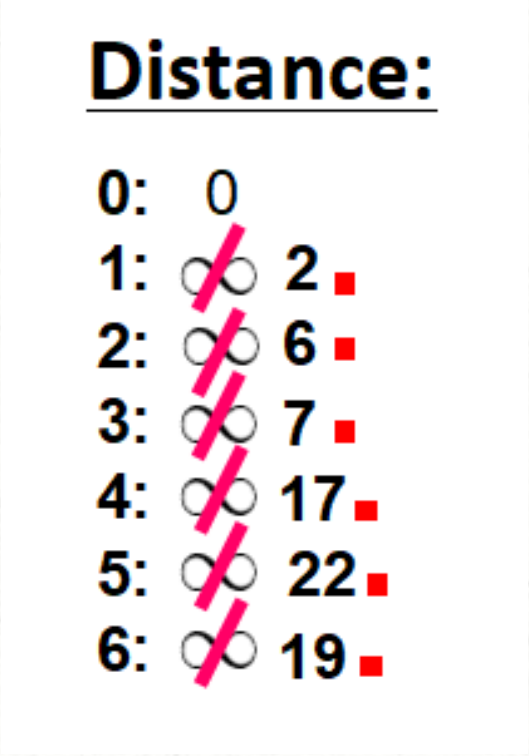
Unvisited Nodes: {~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, ~~6~~}

Dijkstra's Algorithm

Distance:

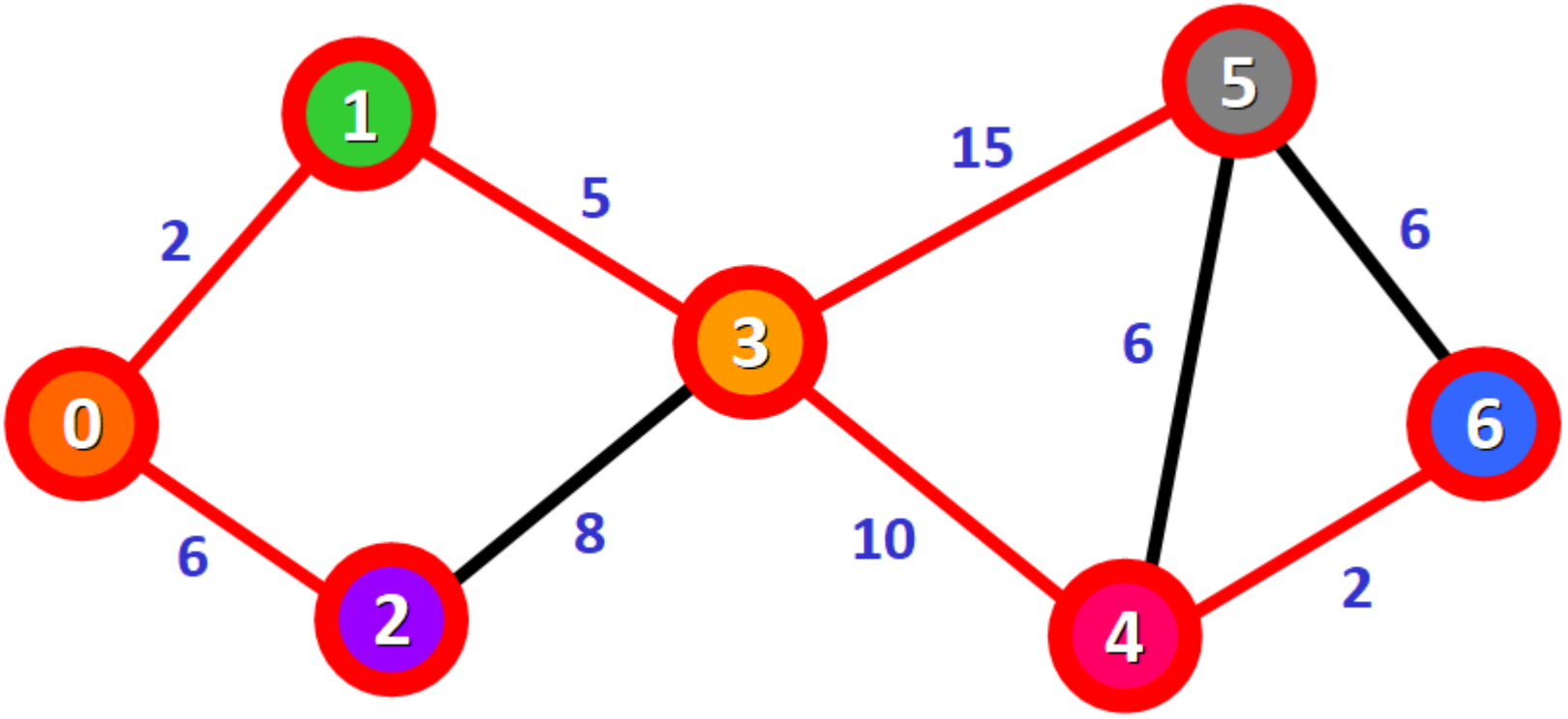
0: 0
1: ~~∞~~ **2**
2: ~~∞~~ **6**
3: ~~∞~~ **7**
4: ~~∞~~ **17**
5: ~~∞~~ **22 from (2 + 5 + 15) vs. 23 from (2 + 5 + 10 + 6) vs. 25 from (2 + 5 + 10 + 2 + 6)**
6: ~~∞~~ **19**

Dijkstra's Algorithm



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

Dijkstra's Algorithm



Steps

1)Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.

2)Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

3)Set the node with lowest distance as the current node

4)For the current node, consider all of its unvisited neighbors and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.

Steps

5)When we are done considering all of the unvisited neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.

6)If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

7)Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current node, and go back to step 4.

Pseudocode of Dijkstra's Algorithm

```
 1  function Dijkstra(Graph, source):
 2
 3      for each vertex v in Graph.Vertices:
 4          dist[v] ← INFINITY
 5          prev[v] ← UNDEFINED
 6          add v to Q
 7      dist[source] ← 0
 8
 9      while Q is not empty:
10          u ← vertex in Q with min dist[u]
11          remove u from Q
12
13          for each neighbor v of u still in Q:
14              alt ← dist[u] + Graph.Edges(u, v)
15              if alt < dist[v]:
16                  dist[v] ← alt
17                  prev[v] ← u
18
19      return dist[], prev[]
```

Thank you

Any Question

??????????