Dijkstra's Algorithm

Most. Kaniz Fatema Isha

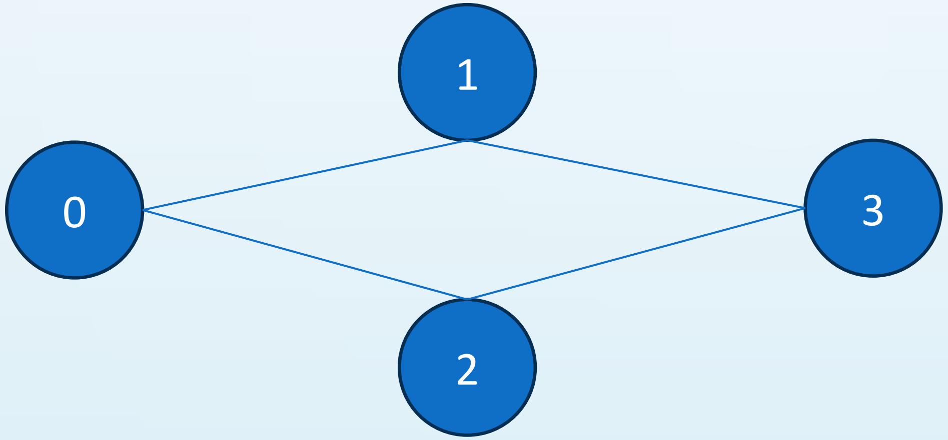
Md Mehrab Hossain Opi

Recap

- We learnt the basic of graph theory in previous semester.
  - Nodes, Edges, Paths, etc.
- We also learnt about graph traversal methods.
  - BFS and DFS.
- We saw how we can find the shortest path between two nodes in an unweighted graph.
  - Using BFS.
- But what if the graph is weighted?

Shortest Path Problem

- What is a path in a graph $G = (V, E)$?
- A sequence of vertices $(v_1, v_2, v_3, \dots, v_k)$ such that there is an edge between each pair of consecutive vertices in the sequence.
  - (v_i, v_{i+1}) is an edge in the set E.
- There can be multiple paths between two nodes.

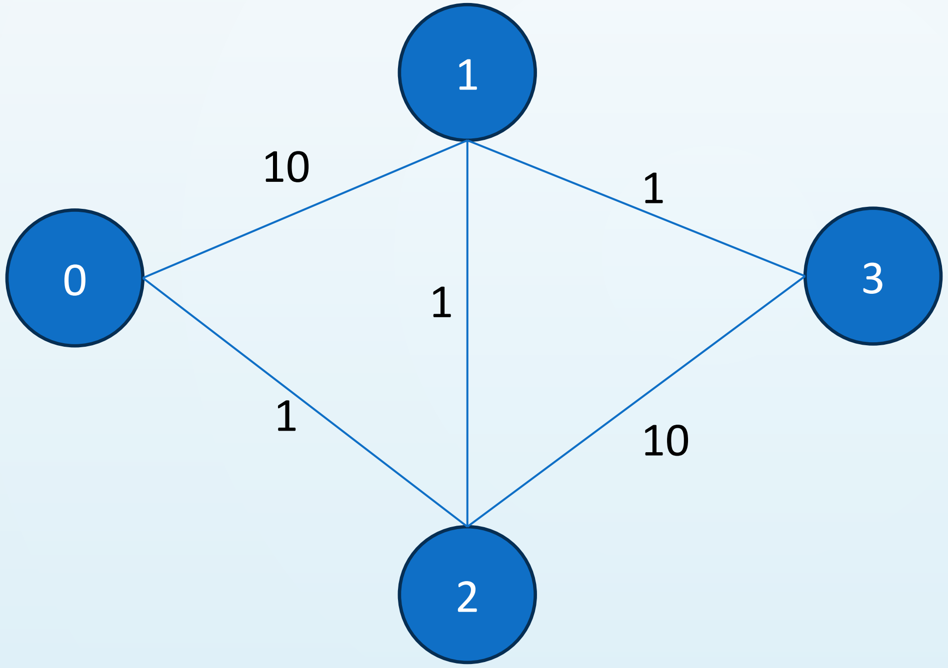


Shortest Path Problem

- How do we find the shortest path now?
- How can you say a path is longer/shorter than another one?
- We need to assign a value to each path to compare them.
  - Let’s call this value the cost of a path.
- We want to find the path with minimum cost.

Cost of a Path

- In an unweighted graph, we considered the cost as number of edges.
  - $dist[v] = dist[u] + 1$
- The same can't be applied when the graph is weighted.



Cost of a Path

- In general, the cost of a path is calculated by summing up the weights of the edges along the path.
- But you can change the cost function.
  - Multiply the weights for example.

Finding the Shortest Path

- Our target is to understand some well-known algorithms that will find the shortest path.
- There are some variations in shortest path.
  - Single-pair shortest path.
  - Single-Source Shortest Path.
  - Single-Destination Shortest Path.
  - All-Pairs Shortest Path.
- Let’s focus on **Single Source Shortest Path (SSSP)** Algorithms today.
  - Single Destination shortest path can be reduced to SSSP.

Single Source Shortest Path Algorithms

- From a source vertex **s**, find the shortest path to all other vertices in the graph.
- Some Common Algorithms
  - Dijkstra’s Algorithm
  - Bellman-Ford Algorithm
  - SPFA (Shortest Path Faster Algorithm)
  - Dial’s Algorithm
  - Thorup’s Algorithm
  - And So on.
- We will talk about Dijkstra’s Algorithm today.

Dijkstra’s Algorithm

- Created and published by Dr. Edsger W. Dijkstra a Dutch Computer Scientist.
    - Original Paper Titled “A note on two problems in connexion with graphs”
- An algorithm that was designed in 20 minutes.
    - Without pen and paper.
- Let’s see how the algorithm works.

Dijkstra’s Algorithm

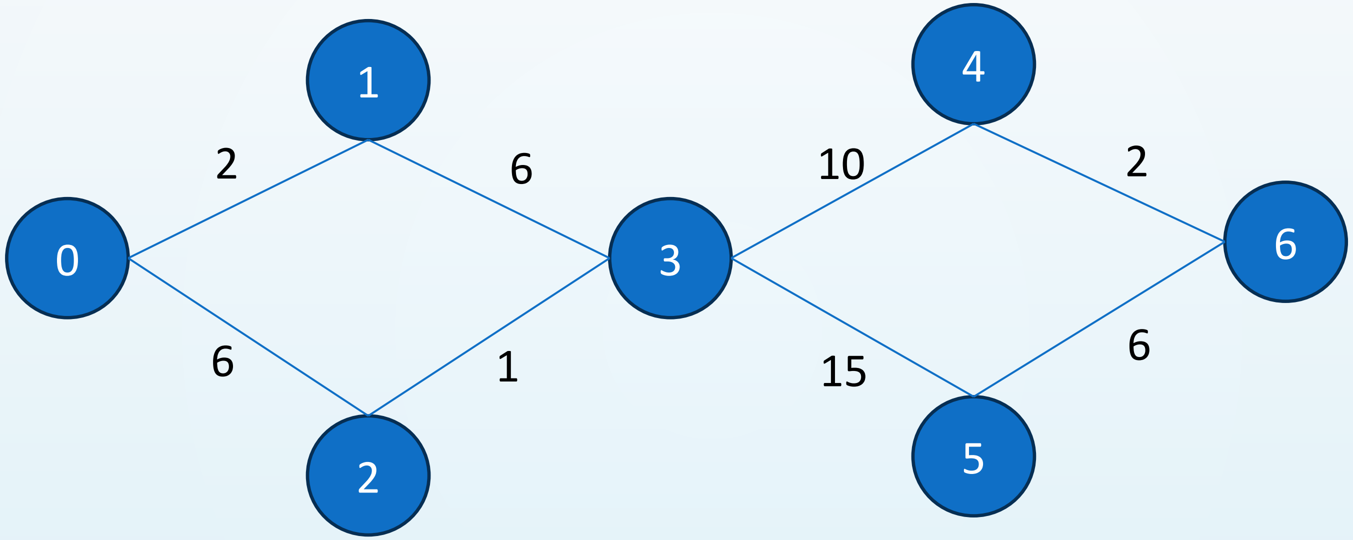
- Choose a starting node.
- Let dist[N] be the distance of node N from the source node.
- Initially, set the distance of all the nodes to infinity.
- We will try to improve the distances step by step.

Dijkstra’s Algorithm

1. Mark all nodes an unvisited. Create a set of all the unvisited nodes – unvisited set.
2. Assign to every node a distance from start value.
    1. For starting node it is zero, For other nodes it is infinity.
3. From the unvisited set, select the node with smallest finite distance as current node.
4. For the current node, consider all of its unvisited neighbors and update their distance through the current node.
5. Mark the current node as visited and remove it from the unvisited.
6. Once the loop exits every visited node will contain its shortest distance from the starting node.

Algorithm Simulation

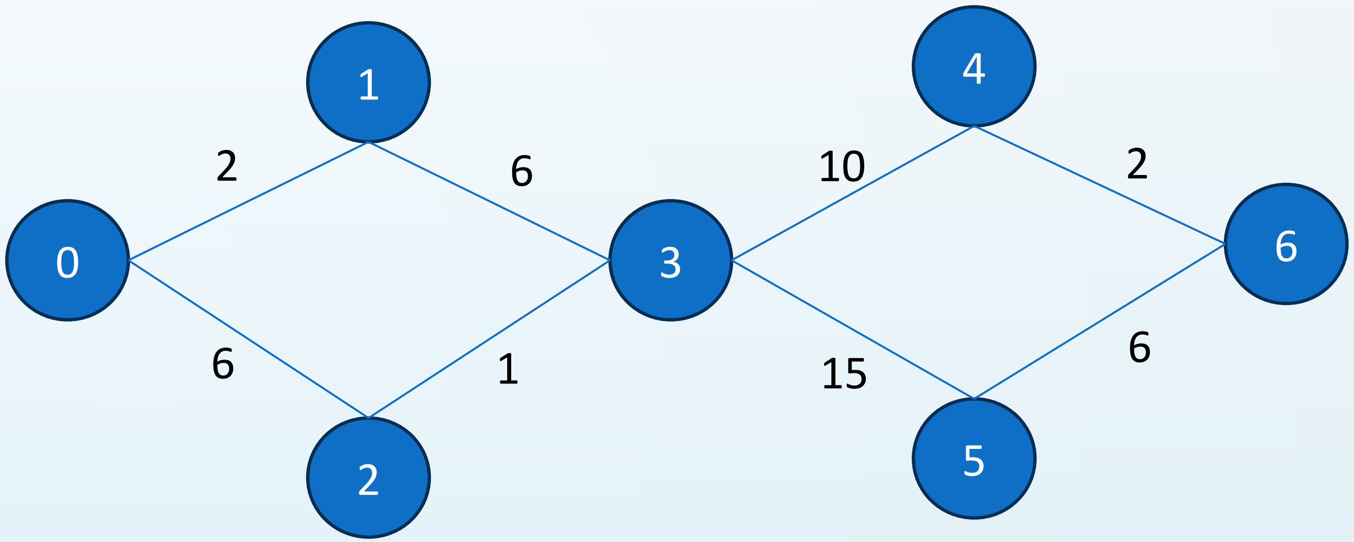
- Let’s consider the following graph.
- We will select node 0 as starting node.



Algorithm Simulation

- We will need a distance array.
    - Initially all will have infinity except 0(source node)

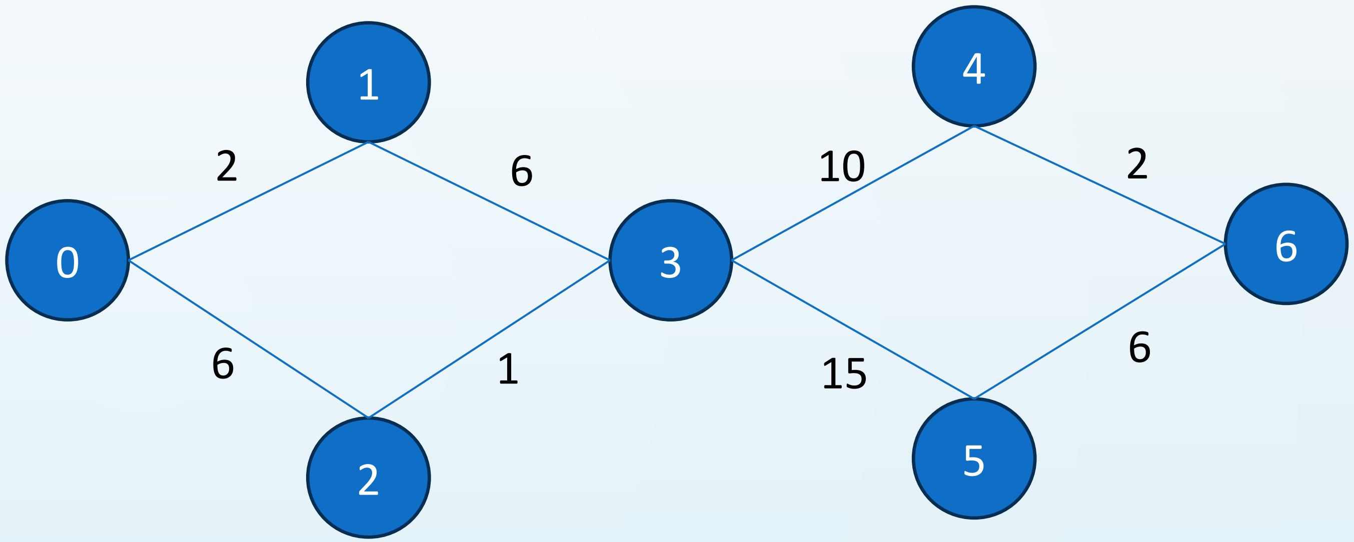
distance						
0	1	2	3	4	5	6
0						



Algorithm Simulation

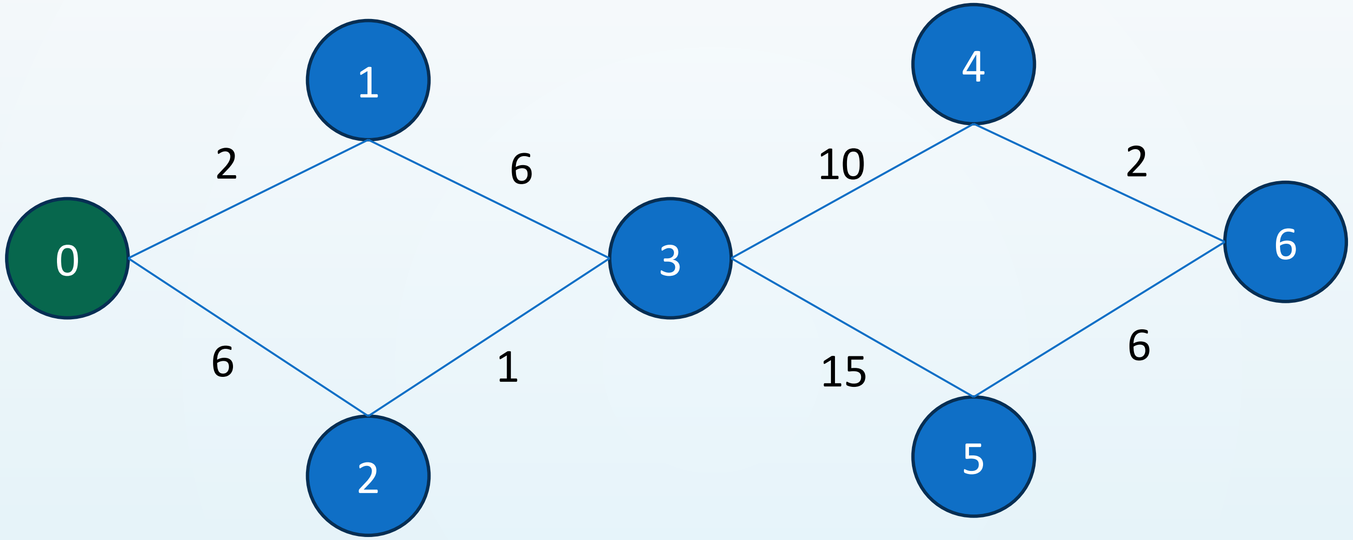
- We also need an unvisited set.
  - We will maintain a visited array for this purpose.

visited						
0	1	2	3	4	5	6
0	0			0	0	



Algorithm Simulation

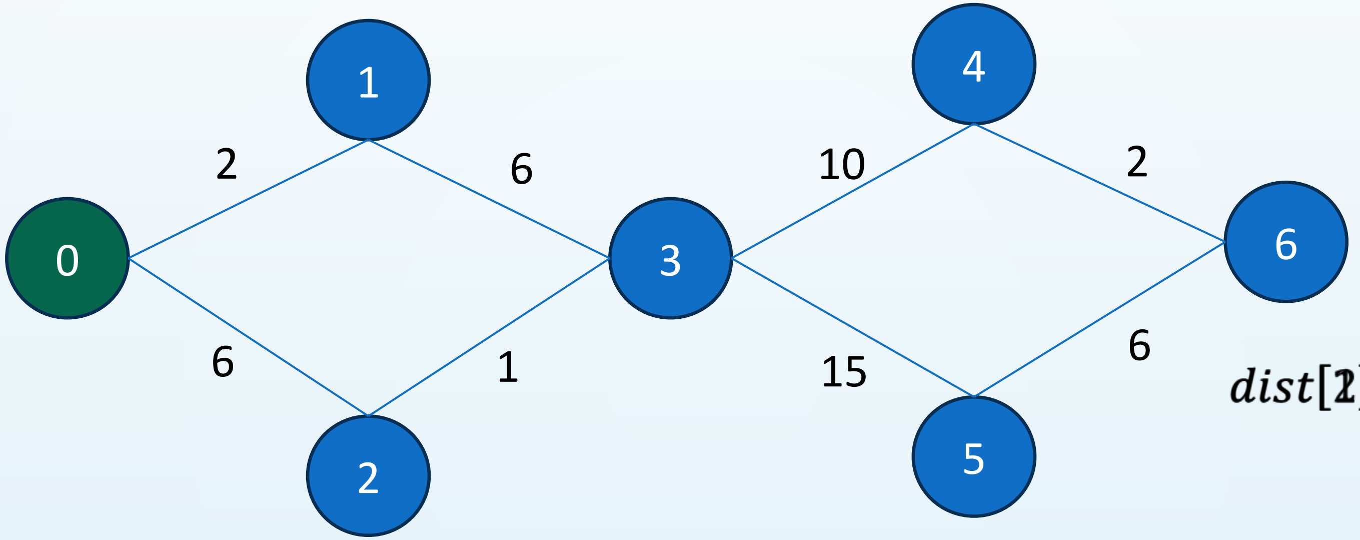
- We need to find the node with **minimum distance** that is **not marked visited**.
  - It will be the source node initially.



Node	0	1	2	3	4	5	6
visited	0	0			0	0	
distance	0						

Algorithm Simulation

- Check all unvisited neighbors of 0 and update their distance.
  - For neighbor v of node u, $dist[v] = \min(dist[v], dist[u] + w)$

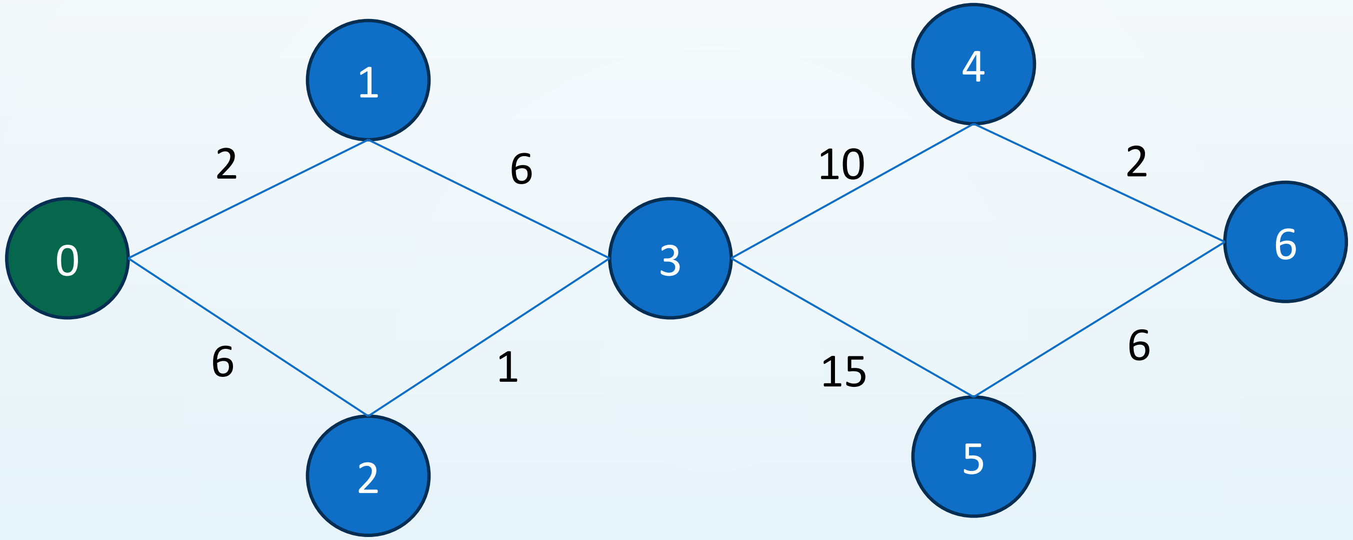


$$
\begin{aligned}
dist[1] &= \min(dist[1], dist[0] + 2) \\
&= \min(\infty, 0 + 2) \\
&= 2
\end{aligned}
$$

Node	0	1	2	3	4	5	6
visited	0	0			0	0	
distance	0						

Algorithm Simulation

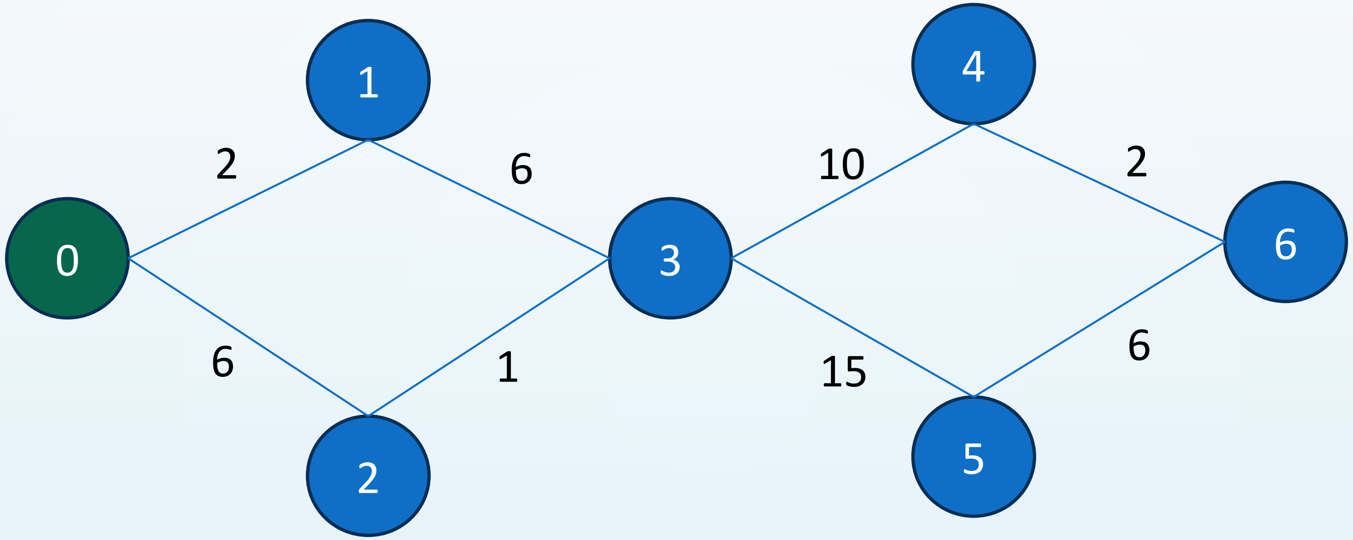
- Finally, mark 0 as visited.



Node	0	1	2	3	4	5	6
visited	1	0			0	0	
distance	0						

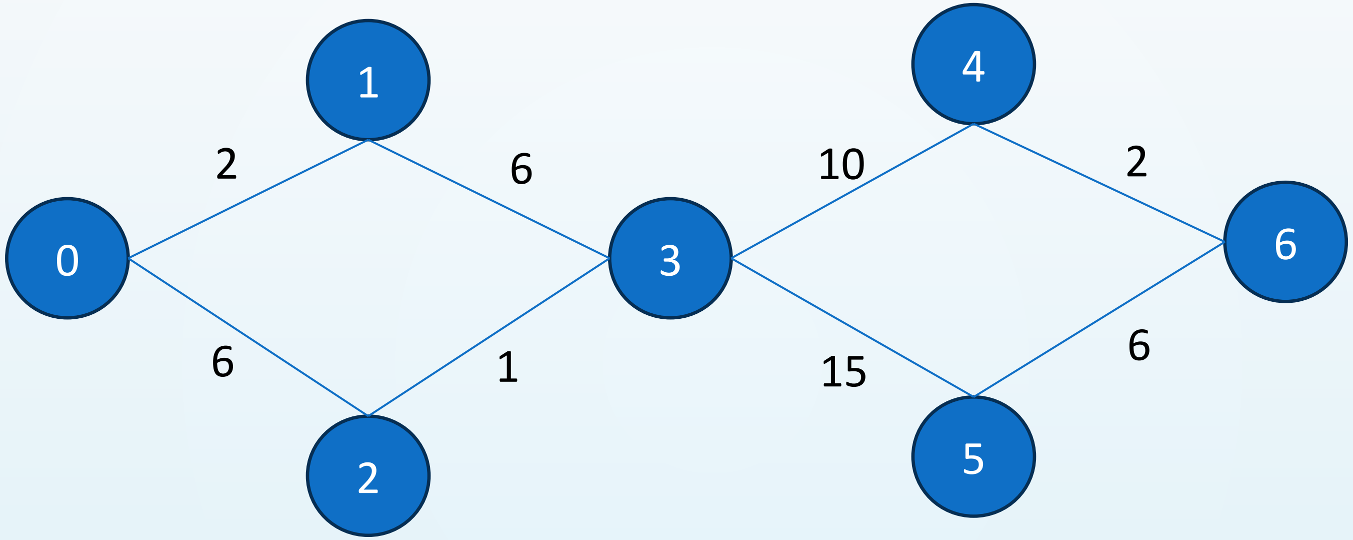
Algorithm Simulation

- We repeat this loop (find unvisited node & update neighbors distance) until there’s no unvisited node.



Node	0	1	2	3	4	5	6
visited	1	0			0	0	
distance	0						

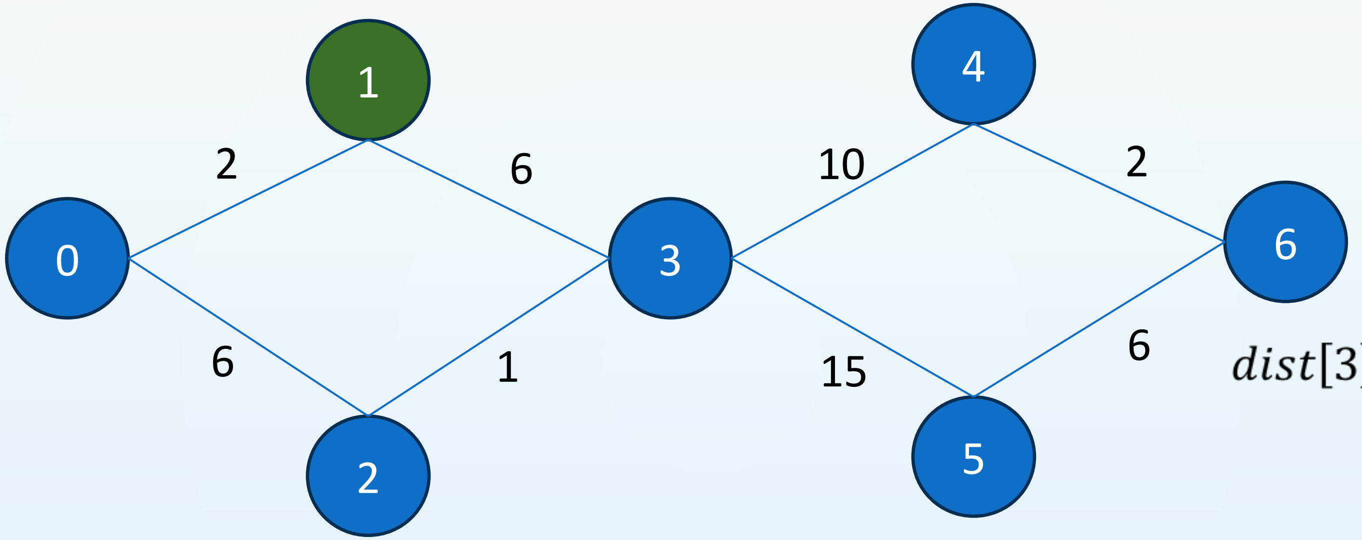
- Unvisited node with minimum distance.
  - 1



Node	0	1	2	3	4	5	6
visited	1	0			0	0	
distance	0						

Algorithm Simulation

- Update distance of neighbor (node 3).

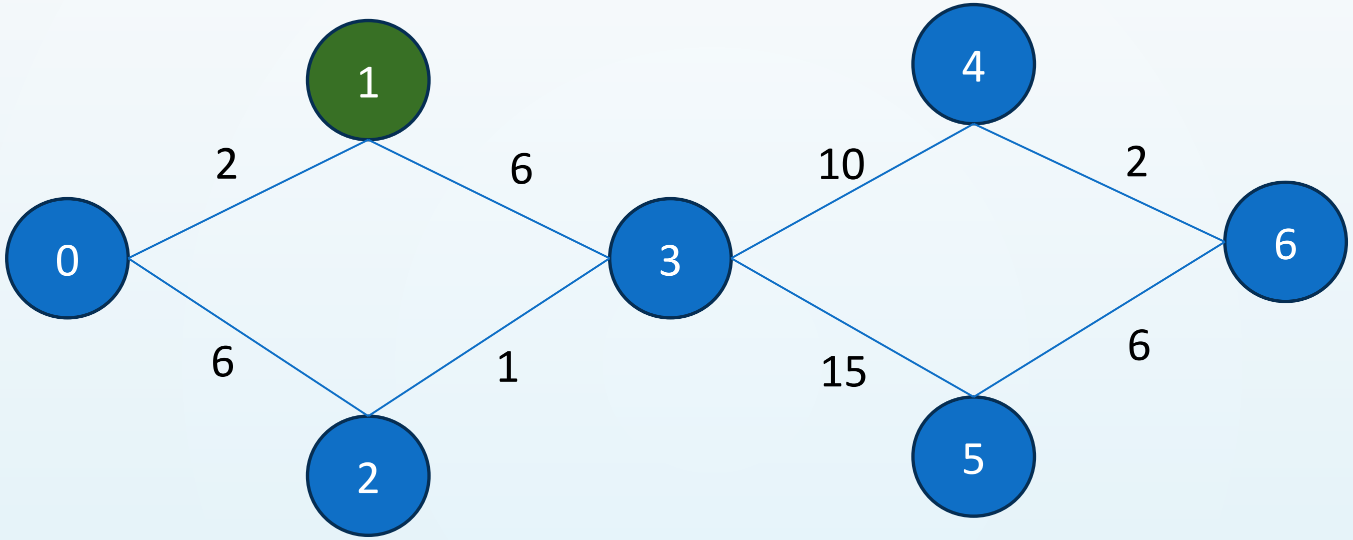


$$\begin{aligned} dist[3] &= \min(dist[3], dist[1] + 6) \\ &= \min(\infty, 2 + 6) \\ &= 8 \end{aligned}$$

Node	0	1	2	3	4	5	6
visited	1	0			0	0	
distance	0			**8**			

Algorithm Simulation

- Finally, mark 1 as visited.

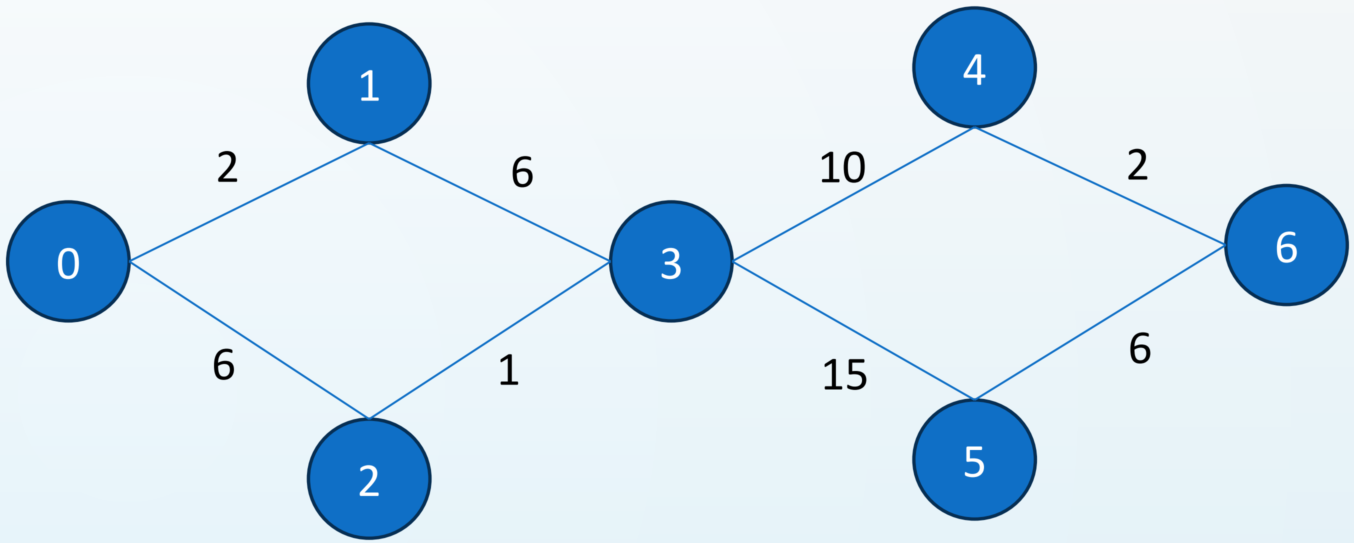


Node	0	1	2	3	4	5	6
visited	1	1			0	0	
distance	0						

Algorithm Simulation

Current Node	2

Neighbor Node	3



$$dist[3] = \min(dist[3], dist[2] + 1)$$
$$= \min(8, 6 + 1)$$
$$= 7$$

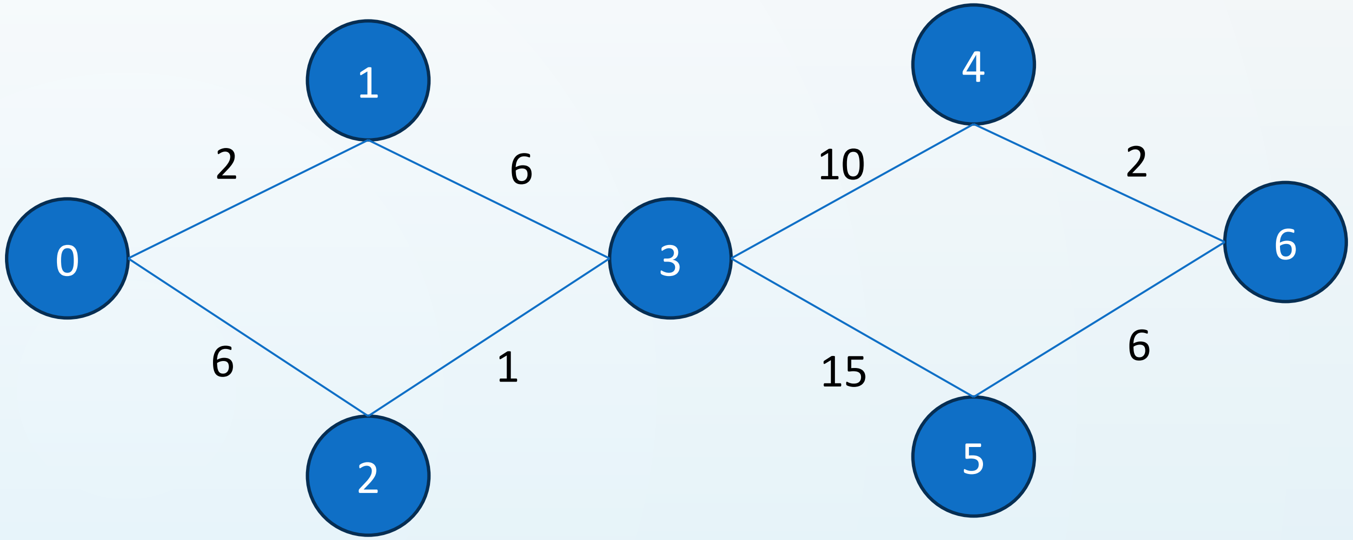
No more neighbors.
Mark 2 as visited.

Node	0	1	2	3	4	5	6
visited	1	1			0	0	
distance	0						

Algorithm Simulation

Current Node	3
Neighbor Node	5

$$dist[5] = \min(dist[5], dist[3] + 15)$$
$$= \min(\infty, 7 + 15)$$
$$= 22$$

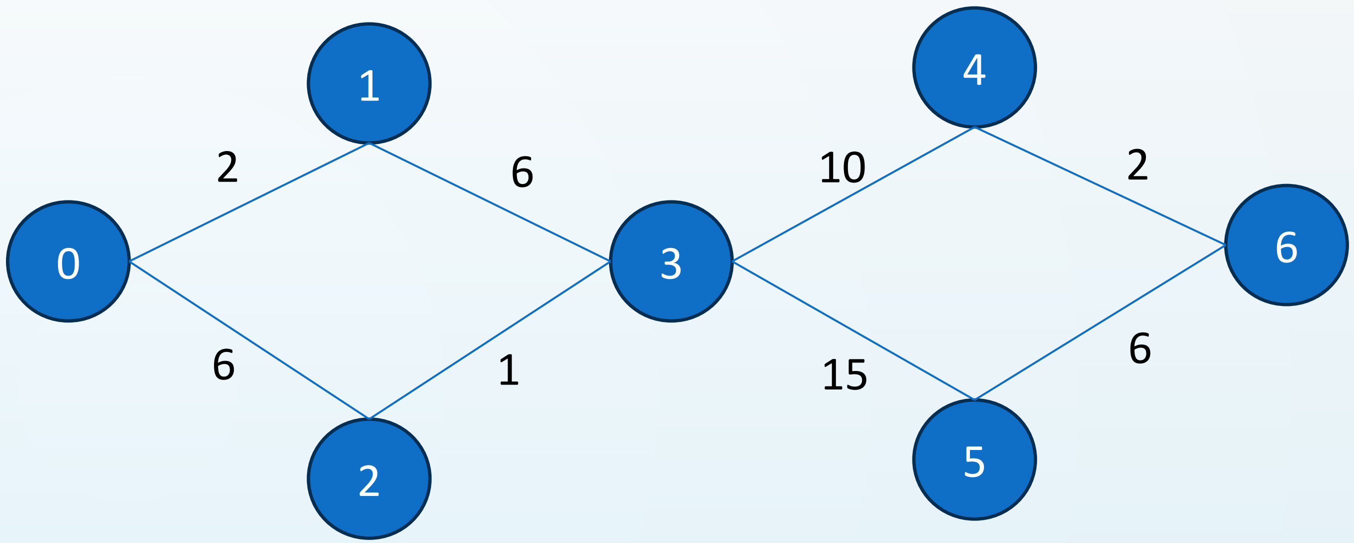


Node	0	1	2	3	4	5	6
visited	1	1	1		0	0	
distance	0						

Algorithm Simulation

Current Node	4

Neighbor Node	6



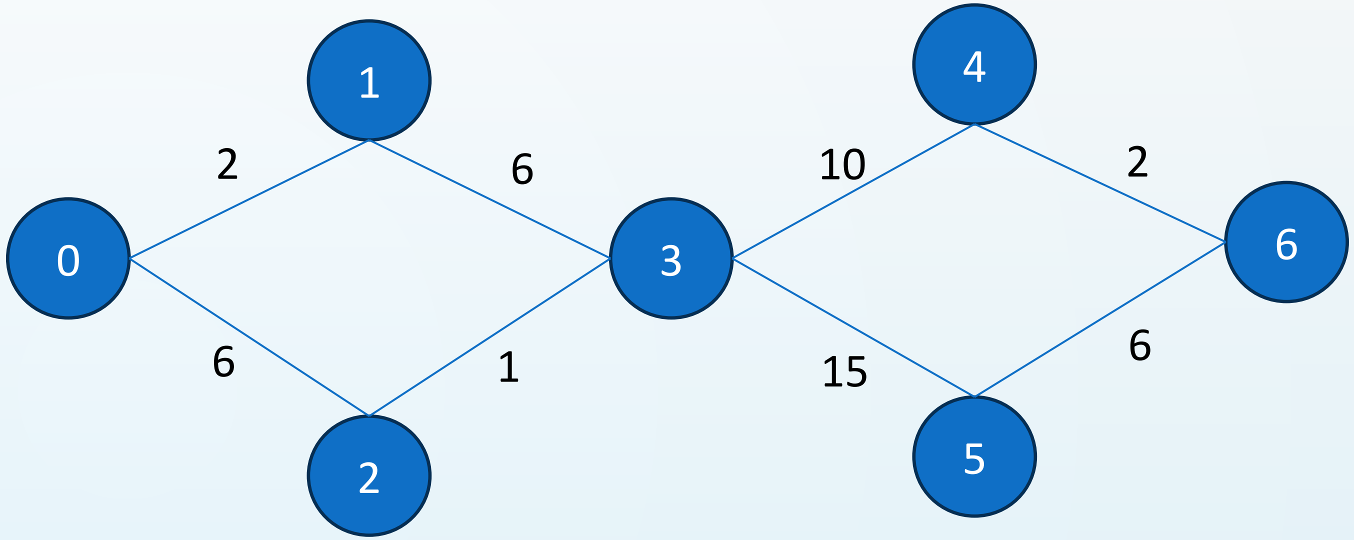
$$dist[6] = \min(dist[6], dist[4] + 2)$$
$$= \min(\infty, 17 + 2)$$
$$= 19$$

Node	0	1	2	3	4	5	6
visited	1	1	1		1	0	
distance	0			7			

Algorithm Simulation

Current Node	6

Neighbor Node	5



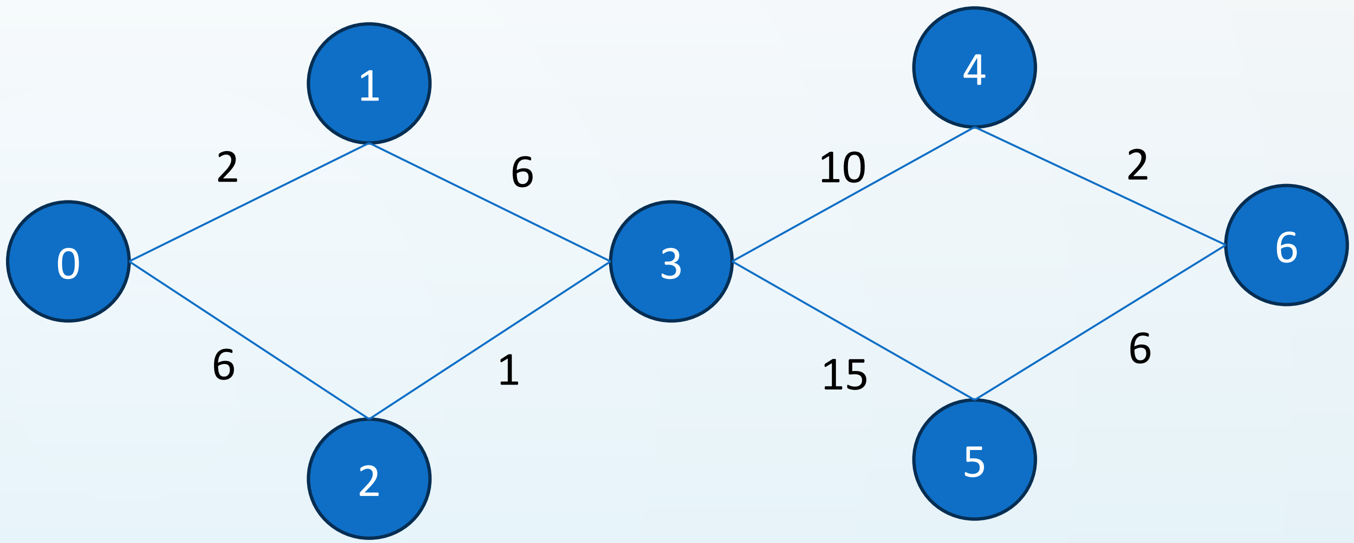
$$
\begin{aligned}
dist[5] &= \min(dist[5], dist[6] + 6) \\
&= \min(22,25) \\
&= 22
\end{aligned}
$$

Node	0	1	2	3	4	5	6
visited	1	1	1		1	0	
distance	0			7			

Algorithm Simulation

Current Node	5

Neighbor Node	



Node	0	1	2	3	4	5	6
visited	1	1	1		1	1	1
distance	0			7			

Implementation

Dijkstra (Graph, source)

1. Create vertex set Q
2. for each vertex v in Graph:
   1. distance[v]:= infinity
   2. add v to Q
3. distance[source] := 0
4. While Q is not empty:
   1. u:= vertex in Q with smallest distance[] value
   2. Remove u from Q
   3. For each neighbor v of u where v is in Q:
      1. **Relax** distance[v]

The process of relaxing an edge (u,v) consists of testing whether going through vertex u improves the shortest path to vertex v found so far and, if so, updating distance[v]

Complexity Analysis

- What will be the complexity now?
- Initializing the distance array will take $O(V)$.
- In the main loop, we find the vertex with minimum distance.
  - This will take $O(V)$
- And the loop will run for each vertex, hence $O(V)$.
- So the total complexity is $O(V^2)$.

Improving the Complexity

- In Dijkstra’s Algorithm, we repeatedly tried to find the minimum value in an array.
- Can you remember any data structure that can find the minimum of an array in constant time?
- We can use a min-heap in this case.
- The min-heap will support two operations:
  - Extract-Min
    - Find the node with minimum distance
  - Decrease-Key
    - Reduce the distance of a given node.

Pseudocode with Min-Heap


```
Dijkstra(G,s)
1. distance := array of size |G|, initialized to ∞.
2. minHeap := a new min-heap
3. for each vertex v in G:
    1. minHeap.insert(v,0) if v is the source else
    2. minHeap.insert(v,∞)
4. while minHeap is not empty:
    1. u:= minHeap.extractMin()
    2. for each neighbor v of u:
        1. alt:= distance[u] + weight(u,v)
        2. if alt<distance[v]:
            1. distance[v] = alt
            2. minHeap.decreaseKey(v,alt)
```

Using STL

- We can either use set or priority_queue that supports the operation of heap.
- In both data structure we will store data as pairs – the distance and the index of the vertex.
- You need to use some workaround while using priority_queue.
  - It is a max-heap.
  - It does not support removing of a random element.
    - Can’t use decreaseKey operation.
- Interestingly, a 2007 technical report concluded the variant of the algorithm not using decrease-key operations ran faster than the decrease-key variant, with a greater performance gap for sparse graphs.

References

- **Introduction to Algorithm**, 4th ed, Leiserson, Charles Eric, Ronald L. Rivest, Thomas H. Cormen, and Clifford Stein.
- Dijkstra’s Algorithm for Adjacency List Representation | Greedy Algo-8 – GeeksforGeeks
- Dijkstra's Shortest Path Algorithm using priority_queue of STL - GeeksforGeeks
- Dijkstra on sparse graphs - Algorithms for Competitive Programming (cp-algorithms.com)