Data Analysis and Algorithm

Practical 8

Write a program to implement Dijkstra's algorithm.

Date.: 24-10-21

Name – Yash Vasudeo Prajapati

Rollno - 022

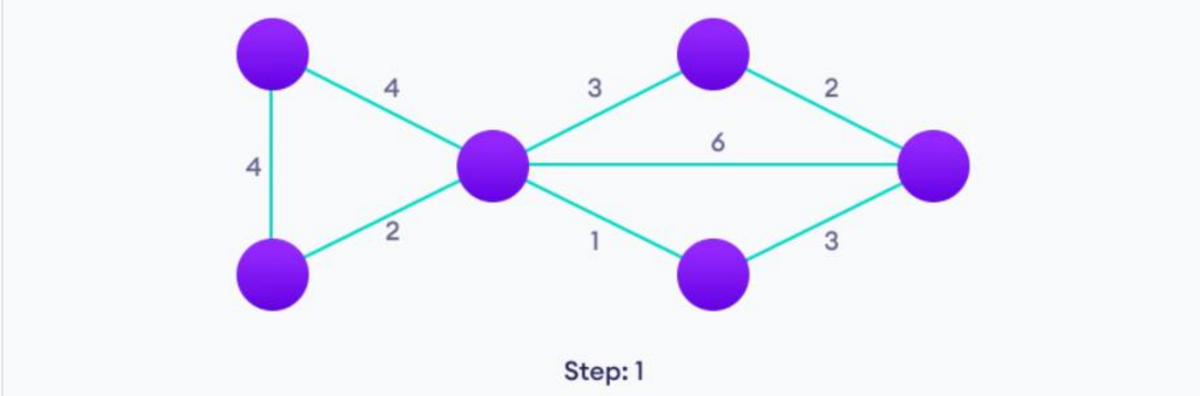
MSc. Computer Science

Theory:-

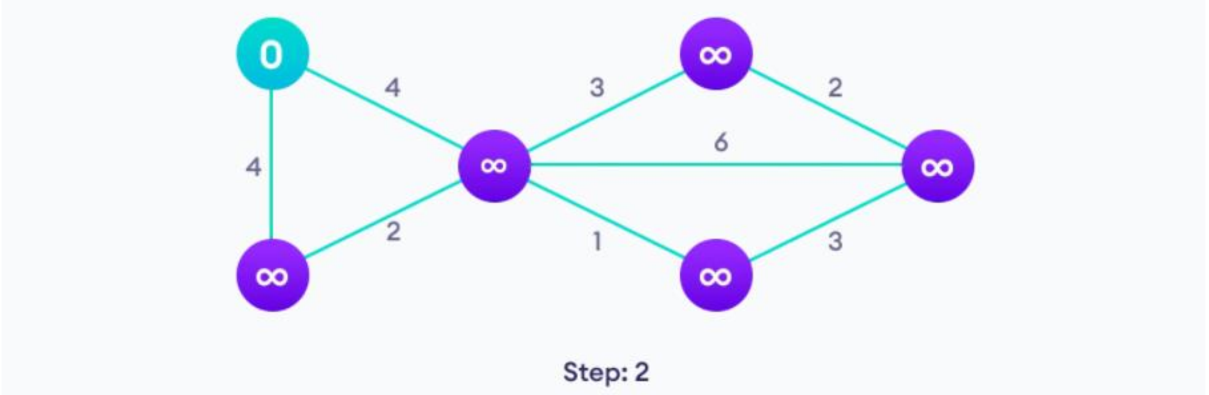
Dijkstra's algorithm is the iterative algorithmic process to provide us with the shortest path from one specific starting node to all other nodes of a graph. It is different from the minimum spanning tree as the shortest distance among two vertices might not involve all the vertices of the graph.

Dijkstra's Algorithm works on the basis that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.

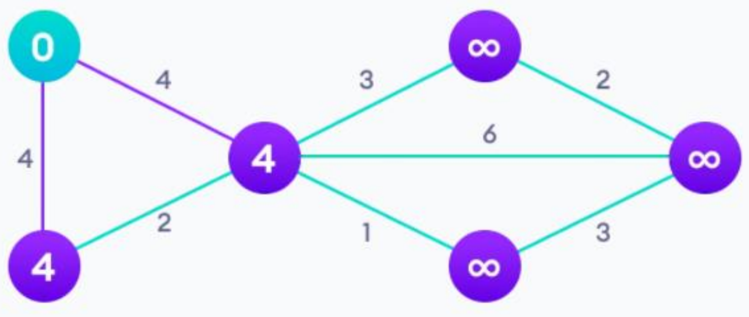
Example:-



Start with a weighted graph

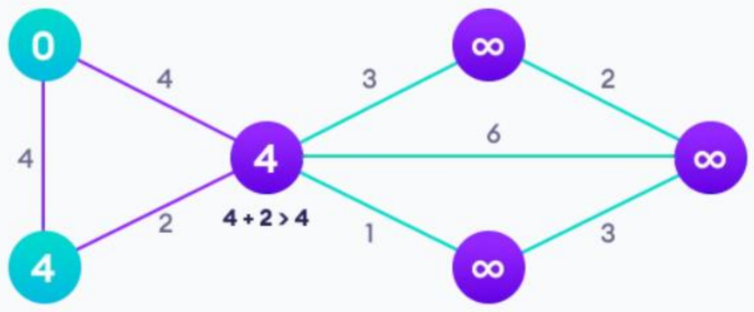


Choose a starting vertex and assign infinity path values to all other devices



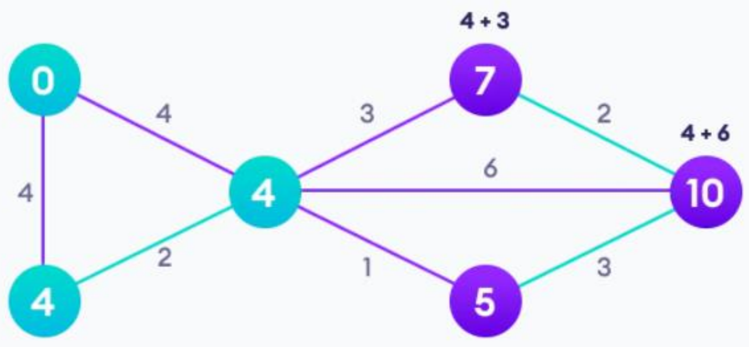
Step: 3

Go to each vertex and update its path length



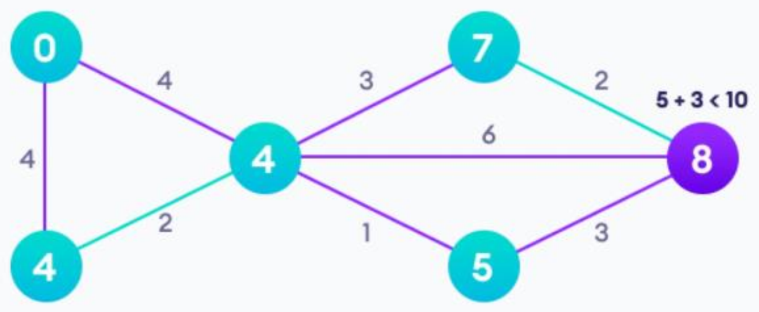
Step: 4

If the path length of the adjacent vertex is lesser than new path length, don't update it



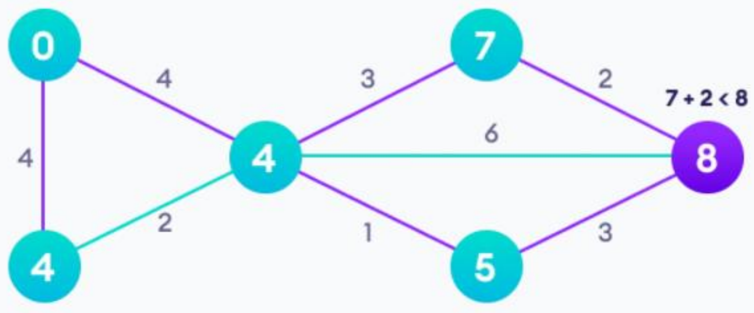
Step: 5

Avoid updating path lengths of already visited vertices



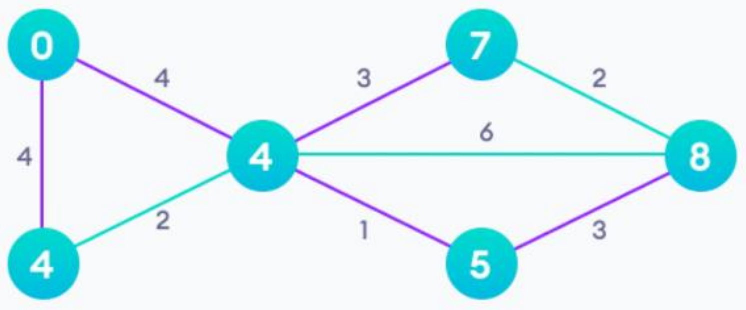
Step: 6

After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7



Step: 7

Notice how the rightmost vertex has its path length updated twice



Step: 8

Repeat until all the vertices have been visited

Algorithm:-

1. Mark all nodes as unvisited.

2. Mark the selected initial node with a current distance of 0 and the rest with infinity.

3. Set the initial node as current node.

4. For the current node, consider all of its unvisited neighbors and calculate their distances by adding the current distance of current node to the weight of the edge connecting neighbor node and current node.

5. Compare the newly calculated distance to the current distance assigned to the neighboring node and set is as the new current distance of neighboring node.

6. When done considering all of the unvisited neighbors of the current node, mark the current node as visited.

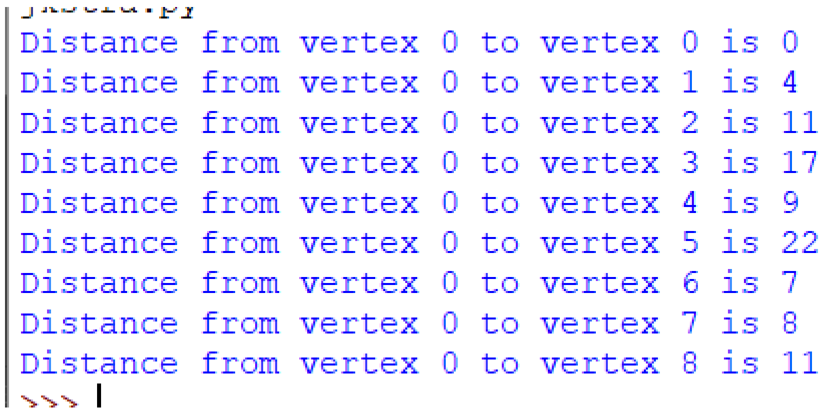
7. If the destination node has been marked visited then stop. The algorithm has finished.

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

Program:-

```
from queue import PriorityQueue

class Graph():
    def __init__(self,vertex):
        self.v = vertex
        self.edges = [[-1 for i in range(vertex)] for j in range(vertex)]
        self.visited = []

    def add_edge(self, u, v, weight):
        self.edges[u][v] = weight
        self.edges[v][u] = weight

    def dijkstra(self, start_vertex):
        #{0: inf, 1: inf, 2: inf,.... n: inf}
        D = {v:float('inf') for v in range(self.v)}
        #{0: 0, ...}
        D[start_vertex] = 0
        pq = PriorityQueue()
        pq.put((0,start_vertex))
        while not pq.empty():
            (dist, current_vertex) = pq.get()
            self.visited.append(current_vertex)

            for neighbor in range(self.v):
                if self.edges[current_vertex][neighbor] != -1:
                    distance = self.edges[current_vertex][neighbor]
                    if neighbor not in self.visited:
                        old_cost = D[neighbor]
                        new_cost = D[current_vertex] + distance
                        if new_cost < old_cost:
                            pq.put((new_cost, neighbor))
                            D[neighbor] = new_cost
        return D


if __name__ == "__main__":
    g = Graph(9)
    g.add_edge(0, 1, 4)
    g.add_edge(0, 6, 7)
    g.add_edge(1, 6, 11)
    g.add_edge(1, 7, 20)
    g.add_edge(1, 2, 9)
    g.add_edge(2, 3, 6)
    g.add_edge(2, 4, 2)
    g.add_edge(3, 4, 10)
    g.add_edge(3, 5, 5)
    g.add_edge(4, 5, 15)
    g.add_edge(4, 7, 1)
    g.add_edge(4, 8, 5)
    g.add_edge(5, 8, 12)
    g.add_edge(6, 7, 1)
    g.add_edge(7, 8, 3)

    D = g.dijkstra(0)

    for vertex in range(len(D)):
        print("Distance from vertex 0 to vertex", vertex, "is", D[vertex])
```

Output:-



```
Distance from vertex 0 to vertex 0 is 0
Distance from vertex 0 to vertex 1 is 4
Distance from vertex 0 to vertex 2 is 11
Distance from vertex 0 to vertex 3 is 17
Distance from vertex 0 to vertex 4 is 9
Distance from vertex 0 to vertex 5 is 22
Distance from vertex 0 to vertex 6 is 7
Distance from vertex 0 to vertex 7 is 8
Distance from vertex 0 to vertex 8 is 11
>>> 
```

Complexity Analysis:-

We go through each edge once which gives us O(E), where E is the number of edges.
Also we visit each node/ vertex once which gives us O(V), where V is the number of vertices.
Since we use priority queue we take O(V log|V|) time to sort the nodes and O(1) time to find the closest neighbour.

Discarding O(1) as the lower order complexity we get
O(E + V log|V|) as the time complexity for Dijkstra algorithm.

Conclusion:-

We implement Dijkstra algorithm and find the time complexity of the same