# Finding shortest paths from source to all vertices in the given graph.

DAA ASSIGNMENT-1 , GROUP 2

Saloni Singla
IIB2019004

Sandeep kumar
IIB2019005

Amanjeet kumar
IIB2019006

***Abstract*—This Paper contains the algorithm for finding shortest paths from source to all vertices in the given graph.**

## I. Problem Statement

Given a graph G and a starting source vertex s, what is the shortest paths from s to every other vertices of G.

## II. Keywords

Graph, Dijkstra, Bellman-ford, Breadth-first-search

## III. Introduction

The Single-Source Shortest Path (SSSP) problem consists of finding the shortest paths between a given vertex v and all other vertices in the graph.

## IV. Algorithmic Design

### A. *Approach 1(By BFS on unweighted graph)*

Breadth-first search (BFS) visits the nodes of a graph in increasing order of their distance from the starting node. Thus, we can calculate the distance from the starting node to all other nodes using breadth-first search.
Breadth-first search goes through the nodes one level after another. First the search explores the nodes whose distance from the starting node is 1, then the nodes whose distance is 2, and so on. This process continues until all nodes have been visited.

1) The idea is to use a modified version of Breadth-first search in which we keep storing the predecessor of a given vertex while doing the breadth-first search.
2) This algorithm will work even when negative weight cycles are present in the graph.
3) We first initialize an array dist[0, 1, …., v-1] such that dist[i] stores the distance of vertex i from the source vertex.
4) Now we get the length of the path from source to any other vertex in O(1) time from array d, and for printing the path from source to any vertex we can use array p and that will take O(V) time in worst case as V is the size of array P.
5) So most of the time of the algorithm is spent in doing the Breadth-first search from a given source which we know takes O(V+E) time. Thus the time complexity of our algorithm is O(V+E).

### B. *Approach 2(By Dijkstra on weighted Graph(with Positive weights)*

Dijkstra's algorithm can be used for non-negative weight edges, Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph.

1) Dijkstra's algorithm set all vertices distances to infinity except for the source vertex and it sets the source distance to 0.
2) Dijkstra's algorithm is efficient of all algorithms,because it only processes each edge in the graph only once by assuming that there are no negative edges.
3) Dijkstra's algorithm calculates the distances from the node to other nodes of the graph and stores in a mi-priority queue to compare in the form of (distance , vertex). The nodes are ordered by their distances.
4) Using a priority queue, the next node to be processed can be retrieved in logarithmic time. The queue gets updated to the distance from the current vertex + edge weight. And if the popped vertex is visited before it continues without using it. It is Applied again until the priority queue is empty.

### C. *Approach 3(By Bellman-Ford on weighted Graph(with negative weights))*

The Bellman–Ford algorithm finds shortest paths from a starting node to all nodes of the graph. The algorithm can process all kinds of graphs, provided that the graph does not contain a cycle with negative length. If the graph contains a negative cycle, the algorithm can detect this.
The algorithm keeps track of distances from the starting node to all nodes of the graph. Initially, the distance to the starting node is 0 and the distance to any other node is infinite. The algorithm then reduces the distances by finding edges that shorten the paths until it is not possible to reduce any distance.

1) The main idea of this algorithm is simple: Relax all E edges (in arbitrary order) V -1 times!
2) Initially dist[s] = 0, the base case. If we relax an edge s $\rightarrow$ u, then dist[u] will have the correct value.
3) If we then relax an edge u $\rightarrow$ v, then dist[v] will also have the correct value.
4) If we have relaxed all E edges V -1 times, then the shortest path from the source vertex to the furthest vertex

from the source (which will be a simple path with V -1 edges) should have been correctly computed.

**Algorithm 1:** To find SSSP in an unweighted graph using BFS.

```
Input: source src,adjacency list l
Output: distance array
Function BFS(src, l):
    dist[src] ← INT_MAX
    distance[src] ← 0
    q.push(src)
    while q.empty() = false do
        parent ← q.front()
        q.pop()
        for auto nbr : l[parent] do
            if dist[nbr] = INT_MAX then
                q.push(nbr)
                distance[nbr] ← distance[parent] + 1
```

**Algorithm 2:** Dijkstra's Algorithm

```
Input: source s,no. of vertex n,adjacency list vec
Output: dist array
Function SSSP(src, n, vec):
    for i ← 0 to n − 1 do
        dist[i] ← Inf
    dist[s] ← 0
    for i ← 0 to n − 1 do
        visited[i] ← false
    visited[s] ← true
    p.push(0, s)
    while q.empty() = false do
        t ← p.top()
        p.pop()
        visited[t.second] ← true
        for i ← 0 to vec[t.second].size() − 1 do
            visited[i] ← false
            if visited[vec[t.second][i].first] = false
              AND t.first + vec[t.second][i].second <
              distance[vec[t.second][i].first] then
                distance[vec[t.second][i].first] ←
                  t.first + vec[t.second][i].second
                p.push(distance[vec[t.second][i].first], vec[t.second][i].first)
```

**Algorithm 3:** Bellman's Algorithm

```
Input: source s, no. of nodes V,adjacency list
       AdjList
Output: dist array
Function SSSP(s, V, AdjList):
    for i ← 0 to V − 1 do
        dist[i] ← Inf
    dist[s] ← 0
    for i ← 0 to V − 2 do
        for u ← 0 to V − 1 do
            for j ← 0 to (int)AdjList[u].size() do
                v ← AdjList[u][j]
                dist[v.first] ←
                  min(dist[v.first], dist[u] + v.second)
```

## V. Algorithm Analysis

### A. *TimeComplexity*

Approach 1: Time complexity of breadth first search(BFS) is O(V+E), where V is the number of nodes and E is the number of edges.

Approach 2: Time complexity of Dijkstra's algorithm is O($V + ElogE$), because it goes through all nodes of the graph and adds the distance for maximum of one time when at the current node.

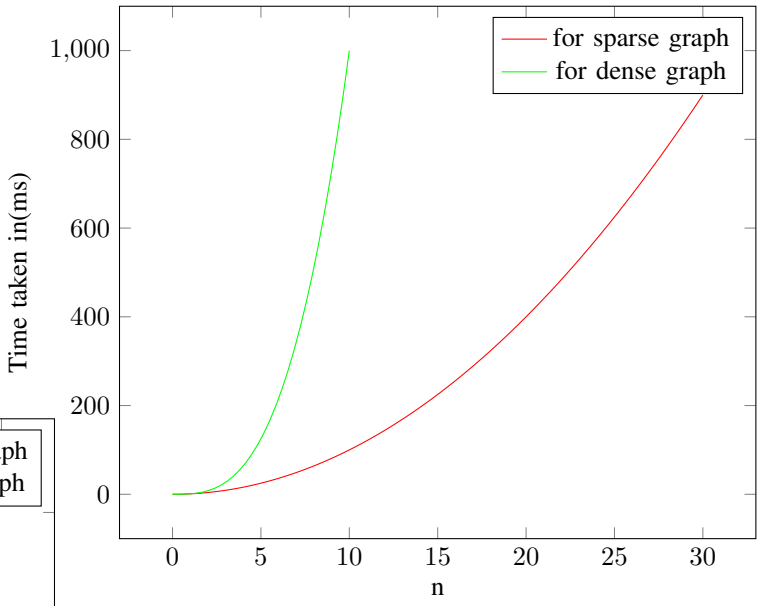
Approach 3: Time complexity of Bellman Ford's algorithm is O($V * E$), because it consists of ($V1$) rounds and iterates through all E edges during a round.

### B. *Space Complexity*

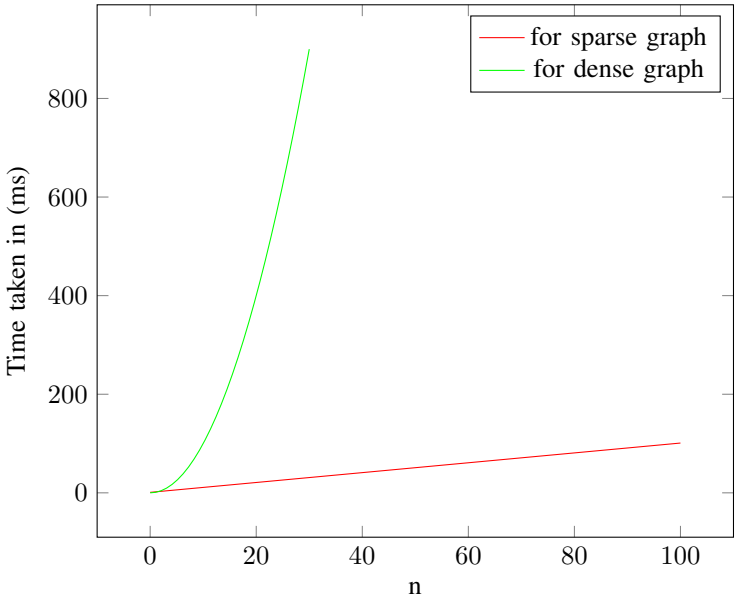
The space complexity for BFS and Dijkstra's algorithms is [$O(V+E)$ (for adjacency list) + $O(V)$ (for other arrays)], and for Bellman Ford's algorithm, the complexity is O(V) +O(E).

## VI. Experimental study

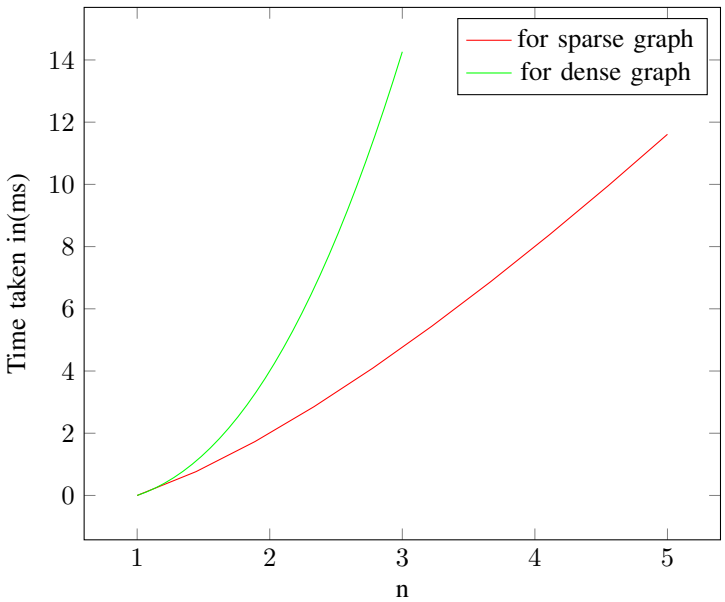
Complexity analysis for Approach 3(Bellman-ford)




Complexity analysis For Approach 1(BFS)




Time Complexity analysis For Approach 2(Dijkstra)




## VII. Conclusion

Here we discussed algorithms to solve SSSP for undirected graph and also for Unweighted graph, the distances can be computed in almost linear time complexity but for weighted graph, there exist several algorithms for different types of graphs.

## VIII. References

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
https://www.geeksforgeeks.org/shortest-path-unweighted-graph/
https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/