DAA ASSIGNMENT-6

JAIDEV DAS IIT2019197
DEEPTARSHI BISWAS IIT2019195

PROBLEM STATEMENT

Single Shortest Distance Problem

Given a graph and a source vertex in the graph, find shortest paths from source to all vertices in the given graph.

DIJKSTRA (ADJACENCY MATRIX)

Algorithm

1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3) While *sptSet* doesn't include all vertices
    - Pick a vertex u which is not there in *sptSet* and has minimum distance value.
    - Include u to *sptSet*.
    - Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

DIJKSTRA (ADJACENCY MATRIX) contd.

Time Complexity Analysis

Time Complexity of the implementation is $O(V^2)$.

Space Complexity Analysis

Space Complexity of this implementation is $O(V+E)$

Remark:- This is a naïve method.

DIJKSTRA (ADJACENCY LIST)

Algorithm

1) Create a Min Heap of size V where V is the number of vertices in the given graph. Every node of min heap contains vertex number and distance value of the vertex.
2) Initialize Min Heap with source vertex as root (the distance value assigned to source vertex is 0). The distance value assigned to all other vertices is INF (infinite).
3) While Min Heap is not empty, do following
   - Extract the vertex with minimum distance value node from Min Heap. Let the extracted vertex be u.
   - For every adjacent vertex v of u, check if v is in Min Heap. If v is in Min Heap and distance value is more than weight of u-v plus distance value of u, then update the distance value of v

DIJKSTRA (ADJACENCY LIST) contd.

Time Complexity Analysis

The complexity of Dijkstra's shortest path algorithm is O(E log V) as the graph is represented using **adjacency list**. Here the E is the number of edges, and V is Number of vertices.

Space Complexity Analysis

Space Complexity of this implementation is O(V).

BELLMAN FORD ALGORITHM

- This step initializes distances from the source to all vertices as infinite and distance to the source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.
- This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
  - Do following for each edge u-v
  - If dist[v] > dist[u] + weight of edge uv, then update dist[v] dist[v] = dist[u] + weight of edge uv
- This step reports if there is a negative weight cycle in graph. Do following for each edge u v
  - If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”.

BELLMAN FORD ALGORITHM contd.

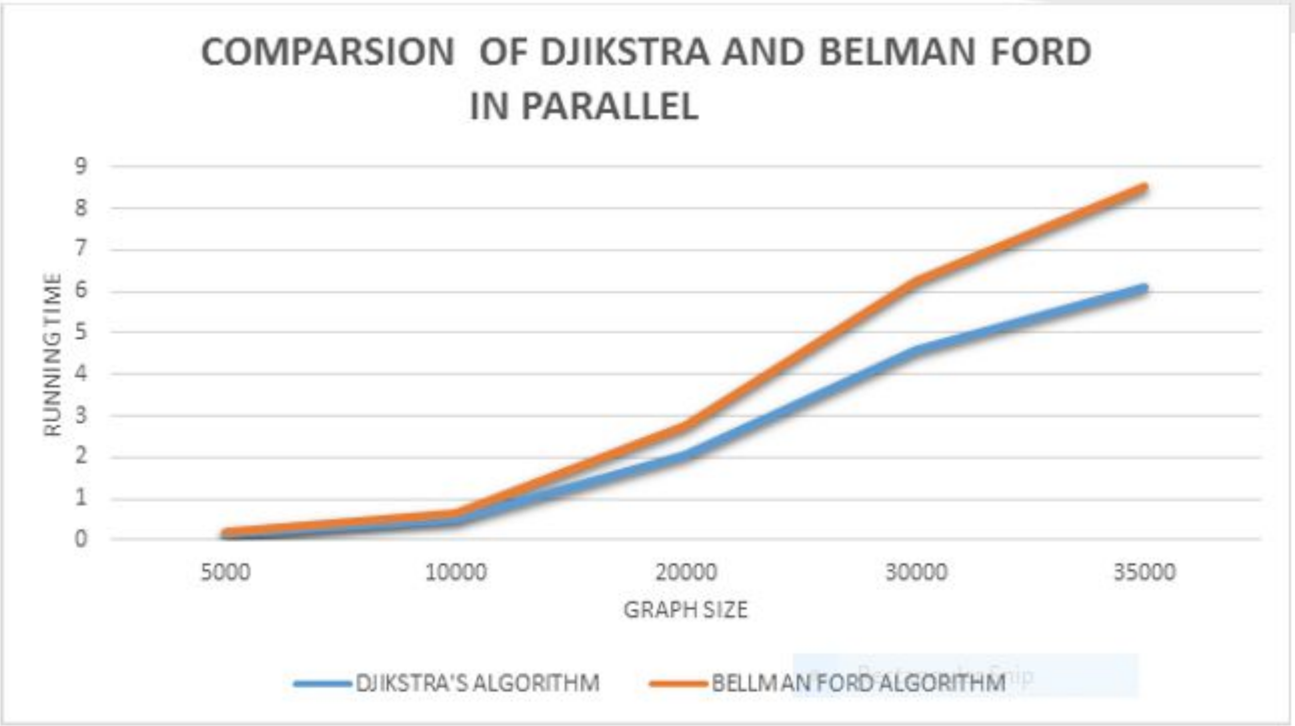
Time Complexity

Time Complexity is O(V*E).

Space Complexity Analysis

Space complexity of bellman ford algorithm is O(V).

COMPARISON BETWEEN DIJKSTRA AND BELLMAN FORD ALGORITHMS



FLOYD WARSHALL ALGORITHM

- We initialize the solution matrix same as the input graph matrix as a first step.

- Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
- When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices.
- For every pair (i, j) of the source and destination vertices respectively, there are two possible cases.
  - k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
  - k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]

FLOYD WARSHALL ALGORITHM contd.

Time Complexity Analysis

The time complexity is O(V^3)

Space Complexity Analysis

The **space complexity** of the **Floyd-Warshall algorithm** is O(V^2).

THANK YOU