SHORTEST PATH FOR ROAD NETWORK PROJECT REPORT

By –

Ajaydeep - 170001003

Rahul Anand - 170001038

INTRODUCTION

Shortest path problems are inevitable in road network application like driving guidance system, Traffic condition among a city changes from time to time and there may be huge request occur at one time ,in such cases optimal routing have to be found.to find a quick solution of such problems we need efficient algorithm. This project aims only at investigate the single source shortest path problem. We are using Dijkstra’s algorithm for road network implementation. The classic Dijkstra’s algorithm was designed to solve the singlesource shortest path problem for a static graph. It works starting from the source node and calculating the shortest path on the whole network. . For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities.

The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path1 from it to every possible destination in the network. Each collection of best paths will then form each node's in road network, which contains information about the topology of the network immediately around it. In addition to just selecting the optimal path, the protocol also involves updating the road network whenever one or more routers go to sleep and/o Nowadays, with the increase of traffic demand, some regulation measures have been adopted to alleviate traffic congestion. The restrictions add difficulties to the search of shortest path problems of road network, and some classical algorithms can not be adopted to find a right router new routers are added..

OBJECTIVES

Shortest path problem are inevitable in road network like driving guidance system, traffic condition among a cities. The project aim is to

1. Investigate a single source   shortest path problem for road network implementation.
2. FInd best possible optimal route
3. Analyze different algorithm for shortest path and check their complexity.

POSSIBLE METHODS

We are using four different algorithms for finding the shortest path for road network

1 Bellman ford

2 Floyd-warshall

3 Dijkstra Algorithm

4 Dials Algorithm

Another Algorithm(BFS OR DFS)

Just to find whether there a path or not between nodes

DESCRIPTION OF USED ALGORITHM WITH PSEUDO CODE AND ANALYSIS

A. BELLMAN FORD

Bellman Ford computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. Our graph is an undirected one, since the time taken for a packet to travel between two directly connected routers is independent of the direction of travel. Thus, necessary conversion of our graph from an undirected graph to a directed one to use the algorithm is required. It can calculate shortest path in a graph with negative edges as well, but this is of no use since all the edge are non-negative. We can use Bellman Ford for positive edges with a complexity of ($|E||V|$) per node.

Thus total complexity would be ($|E||V|2$), since we need to compute all source to all destination path.

Pseudo code

```
function BellmanFord(list vertices, list edges, vertex source)
  ::distance[],predecessor[]

  // This implementation takes in a graph, represented as
  // lists of vertices and edges, and fills two arrays
  // (distance and predecessor) with shortest-path
  // (less cost/distance/metric) information

  // Step 1: initialize graph
  for each vertex v in vertices:
     distance[v] := inf            // At the beginning , all vertices have a weight of infinity
     predecessor[v] := null        // And a null predecessor

  distance[source] := 0            // The weight is zero at the source

  // Step 2: relax edges repeatedly
  for i from 1 to size(vertices)-1:
     for each edge (u, v) with weight w in edges:
        if distance[u] + w < distance[v]:
           distance[v] := distance[u] + w
           predecessor[v] := u

  // Step 3: check for negative-weight cycles
  for each edge (u, v) with weight w in edges:
     if distance[u] + w < distance[v]:
        error "Graph contains a negative-weight cycle"

  return distance[], predecessor[]
```

COMPLEXITY OF BELLMAN FORD IN TERM OF OPERATION COUNT

The complexity of the Bellman-Ford algorithm depends on the number of edge examinations, or relaxation calls . As mentioned, the number of relaxation calls can be smaller than |V||E| with the BGL implementation. In fact, it is much smaller than |V||E| in the average case.

2 FLYOD –WARSHELL

Floyd-Warshall finds shortest path between all pairs of vertices which our objective. Thus it eliminates the need to apply the algorithm to each node.

It is generally a good choice for dense graphs. But ours will generally be a sparse one due to each router getting connected to a significantly low number of routers as compared to the square of total number of routers, i.e., the probability of each router being connected to every other router is extremely low. Overall complexity for the algorithm would be ($|V|3$).

```
1 let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
2 for each edge (u,v)
3    dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
4 for each vertex v
5    dist[v][v] ← 0
6 for k from 1 to |V|
7    for i from 1 to |V|
8       for j from 1 to |V|
9          if dist[i][j] > dist[i][k] + dist[k][j]
10            dist[i][j] ← dist[i][k] + dist[k][j]
11          end if
```

3 DIJKSTRA'S ALGORITHM

For a given source node in the graph, Dijkstra's Algorithm finds the shortest path between that node and every other. Our requirement is to calculate shortest path for each node to all nodes. Thus, a possible solution is to run Dijkstra on every node. By using Fibonacci Heap, the complexity per node comes out to be ($|E| + |V|\ log|V|$). Thus overall complexity is ($|E||V| + |V|2\ log|V|$).

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:             // Initialization
        dist[v] ← INFINITY                  // Unknown distance from source to v
        prev[v] ← UNDEFINED                 // Previous node in optimal path from source
        add v to Q                          // All nodes initially in Q (unvisited nodes)

    dist[source] ← 0                        // Distance from source to source

    while Q is not empty:
        u ← vertex in Q with min dist[u]    // Node with the least distance
                                            // will be selected first
        remove u from Q

        for each neighbor v of u:           // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:               // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

TIME COMPLEXITY FOR DIJKSTRA ALGORITHM

The run time of first for loop is O(V). In each iteration of the while loop, Extract_Min of the heap is logV. The inner for loop iterates each adjacent node of the current node, the total run time is O(E). Therefore, the time complexity of this algorithm is O((V + E)*log(V) = O(E* log(V)). As the number of nodes in a graph increases, the running time of the applied algorithm will become longer and longer. Usually, a road network of a city has more than 10^4 nodes. A fast shortest path algorithm becomes more desirable.

4 Dials algorithm

If maximum weight is small or if range of edge weight is small then we use dial’s algorithm. Having the complexity O(E +W V)

We use bucket data structure .

```
//

  while (loop infinitly)

  {

      // Go sequentially through buckets till one non-empty

      // bucket is found

      while (search for an un-empty bucket )

          idx <- idx+1

              end while


       if (no non-empty bucket is found)

          exit

                  end if



       u = B(idx).front() pop first element of non-empty bucket



      // Process all adjacents of extracted vertex 'u' and

      // update their distanced if required.

      for (iterate all the vertices connected to u)

           v = (*i).first
```

```
     weight = (*i).second


     du = dist(u).first
     dv = dist(v).first


     // If there is shorted path to v through u.
     if (check if the distance between dv > du + weight) //


        // bucket, so erase its entry using iterator
        // in O(1)
        if (dv != INF)
           B[dv].erase(dist[v].second)
    end if
        //  updating the distance
        set dist(v).first <- du + weight // update the new distance
        dv <- dist[v].first;


        // pushing vertex v into updated distance's bucket
        B[dv].push_front(v);


        // storing updated iterator in dist[v].second
        dist[v].second <- B(dv).begin();
    end if
```

6. BFS(IT IS USED TO DETERMINE WHETHER THERE EXIT PATH BETWEEN VERTEX)
// as it complexity is just O(E+V) when implemented using adjacency list

```
BFS (G, s)                  //Where G is the graph and s is the source node
      let Q be queue.
      Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.

      mark s as visited.
      while ( Q is not empty)
           //Removing that vertex from queue,whose neighbour will be visited now
           v  =  Q.dequeue( )

          //processing all the neighbours of v
          for all neighbours w of v in Graph G
               if w is not visited
                        Q.enqueue( w )             //Stores w in Q to further visit its neighbour
                  mark w as visited.
```

FUTURE WORK-

The future research can go into two directions. First, well known algorithms can be adapted into the public transport needs. For example, the algorithm for finding second shortest path, third etc, paths for any vehicle can be developed. More can be proposed: finding the shortest path going through specific nodes, through specific number of nodes or by the most reliable path.

The other direction is development of new algorithms for traffic issues and not just adaptation of existing algorithms.

REAL LIFE EXAMPLE-

1. In video games, these algorithms are frequently used to find the shortest path between two points on a map. "Pathfinding," as it is called in this context, can be used by AI to plot routes, or by the game engine to assist users in plotting routes.
2. In Biology it is used to find the network model in spreading of a infectious disease.
3. It can be used to implement with vehicle routing problem, due to which traffic and road network will be improved,
4. It can also be used in maps.
5. It has also be use to solve the word ladder puzzle.

REFERENCE

1 Geeks for geeks (Design and analysis of algorithm)

2 Ji-xian Xiao; Fang-Ling Lu, "An improvement of the shortest path algorithm based on Dijkstra algorithm," Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on , vol.2, no., pp.383,385, 26-28 Feb. 2010

3 Tieyuan Yin; Jianyong Yang, "Notice of RetractionDynamic application of the path selection in the road," Software Engineering andService Sciences (ICSESS), 2010 IEEE International Conference on , vol., no., pp.249,252, 16-18 July 2010