International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 08 |Aug -2016 www.irjet.net p-ISSN: 2395-0072

Review of Shortest Path Algorithm

Thorat Surekha[1], Rahane Santosh[2]

[1]*At/Post .Sangamner ,dist.Ahmednagar ,Country . India*

[2]*At/Post .Sangamner ,dist .Ahmednagar ,Country . India*

Abstract - *Shortest path algorithm find minimum distance path between source to destination .we use graph to solve shortest path distance problem . Graph is set of Edges and vertices. A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. Formally, a graph is a pair of sets (V, E), where V is the set of vertices and E is the set of edges, connecting the pairs of vertices. undirected graphs have edges that do not have a direction . The edges indicate a* two-way *relationship, in that each edge can be traversed in both directions. Directed graphs have edges with direction. The edges indicate a* one-way *relationship, in that each edge can only be traversed in a single direction. Edges contain Wight which is used to calculate shortest path from source to destination .consider example Airline route maps. Vertices represent airports, and there is an edge from vertex A to vertex B if there is a direct flight from the airport represented by A to the airport represented by B. There are different shortest path algorithm which solve shortest path problem. They are Dijkstra's ,Floyd – Warshall ,Bellman Ford Algorithm.*

Key Words: **Graph ,Dijkstra's Algorithm,Floyd-Warshall Algorithm ,Bellman –Ford Algorithm**

1.RESEARCH OBJECTIVES

To determine representation of graph in computer with basic terms also find problem of shortest path problem .To discuss general aspect of ,Dijkstra's Algorithm,Floyd-Warshall Algorithm ,Bellman –Ford Algorithm

2. LITERATURE REIVEW

A graph can be used to represent a map where the cities are represented by vertices and the routes or roads are represented by edges within the graph. In these paper brief descriptions and implementations of the three shortest path algorithms being studied are presented2.1 Representation Of Graph.Graph can be represented using adjacency matrix and adjacency list . Consider 2D array will be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex 'u' to vertex 'v' are efficient and can be done O(1). Consumes more space O(V^2). Even if the graph contains less number of edges, it consumes the same space. Adding a vertex is O(V^2) time. We also use linked lists for representation of graph .Let the array be a[]. An entry a[i] represents the linked list of vertices adjacent to the i^{th} vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists.

2.2 DIJKSTRA'S ALGORITHM :EXPLANATION AND IMPLEMENTATION

Dijkstra's algorithm is used for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. we can use a priority queue in which vertices are sorted by their increasing dist[] value. Then at each iteration, we will pick the vertex, u, with smallest dist[u] value and call relax(u,v) on all of t_s neighbours, The only difference is that now we add

the weight of the edge (u, v) to our distance instead of just adding 1. However, the algorithm only works as long as we do not have edges with negative weights. Otherwise, there is no guarantee that when we pick u as the closest vertex, dist[v] for some other vertex v will not become smaller than dist[u] at some time in the future. There are several ways to implement Dijkstra's algorithm. The main challenge is maintaining a priority queue of vertices that provides 3 operations –inserting new vertices to the queue, removing the vertex with smallest dist[], and decreasing the dist[] value of some vertex during relaxation. We can use a set to represent the queue. Dijkstra's algorithm is very fast, but it suffers from its inability to deal with negative edge weights.

2.3 Bellman –Ford Algorithm :Explanation And Implementation

Having negative edges in a graph may also introduce negative weight cycles that make us rethink the very definition of "shortest path". Fortunately, there is an algorithm that is more tolerant to having negative edges –the BellmanFord algorithm . That is why, a graph can contain cycles of negative weights, which will generate numerous number of paths from the starting point to the final destination, where each cycle will minimize the length of the shortest path .The Bellman Ford algorithm is a Dynamic Programming algorithm that solves the shortest path problem. It looks at the structure of the graph, and iteratively generates a better solution from a previous one, until it reaches the best solution. Bellman Ford can handle negative weights readily, because it uses the entire graph to improve a solution. The idea is to start with a base case solution S_0, a set containing the shortest distances from s to all vertices, using no edge at all. In the base case, d[s] = 0, and d[v] = ∞ for all other vertices v. We then proceed to relax every edge once, building the set S_1. This new set is an improvement over S_0, because it contains all the shortest distances using one edge –ie. d[v] is minimal in S_1 if the shortest path from s to v uses one edge. Now, we repeat this process iteratively, building S_2 from S_1, then S_3 from S_2, and so on..Each set S_k contains all the shortest distances from s using k edges –ie. d[v] is minimal in S_k if the shortest path from s to v uses at most k edges. we have the best solution after n1 iterations . The algorithm above basically implements this idea. We start with a base case S_0, and repeatedly relax every edge to generate S_{k+1} from S_k. Note that in the relaxation step, we don't relax an edge if dist[u] is infinity, or otherwise we may get overflow in the addition (conceptually we never want to relax such an edge anyway). Also the order of using the edges can affect the intermediate sets S_k, because we may first relax an edge (u,v), then relax another edge (v,w) in the same step, while choosing the reverse order of these two edges may not relax them both. However, we now show that S_{n1} is unique, and contains the shortest distance possible from s to any vertex v. the Bellman Ford algorithm is correct, but does it always terminate? It does, as we only have two loops, one running n-1 iterations, and the other going through all edges. Hence, the algorithm always terminates, and has a run time of O(n*m). While the Bellman Ford algorithm can handle negative weight edges readily, the correctness of the algorithm breaks down when negative weight cycles exist that is reachable from s. However, the nature of the algorithm allows us to detect these negative weight cycles. The idea is that, if a negative weight cycle exist, then S_{n1} will be the same as S_n, S_{n+1}, S_{n+2}, If we run the iteration step more than n1 times, we will not be changing the answer. On the other hand, if a negative weight cycle exist, then one of its edges must have negative weight, and any such edge can be relaxed further even after n1 iterations, decreasing some of the distances. Hence, to detect negative weight cycles, we just need to run the Bellman Ford algorithm, and when it terminates, check whether we can relax any edges. If we can, then that edge is reachable from a negative weight cycle, and the cycle is also reachable from the source.

2.4 Floyd–Warshall Algorithm :Explanation And Implementation

The Floyd-Warshall algorithm improves upon this algorithm, running in $\Theta(n^3)$ time. The genius of the Floyd-Warshall algorithm is in finding a different formulation for the shortest path sub problem than the path length formulation introduced earlier. At first the formulation may seem most unnatural, but it leads to a faster algorithm. As before, we will compute a set of matrices whose entries are $d_{(k)}$ ij . We will change the meaning of each of these entries.

For a path p = hv1, v2,...,v`i we say that the vertices v2, v3,...,v`−1 are the intermediate vertices of this path. Note that a path consisting of a single edge has no intermediate vertices. We define $d_{(k)}$ ij to be the shortest path from i toj such that any intermediate vertices on the path are chosen from the set {1, 2,...,k}. In other words, we consider a path from i to j which either consists of the single edge (i, j), or it visits some intermediate vertices along the way, but these intermediate can only be chosen from {1, 2,...,k}. The path is free to visit any subset of these vertices, and to do so in any order. Thus, the difference between Floyd's formulation and the previous formulation is that here the superscript (k) restricts the set of vertices that the path is allowed to pass through, and there the superscript (m) restricts the number of edges the path is allowed to use. If the value of $d_{(32)^{(k)}}$ changes as k varies. The final answer is $d_{(n)}$ ij because this allows all possible vertices as intermediate vertices. Again, we could write a recursive program to compute d(k) ij , but this will be prohibitively slow.This algorithm can be easily modified to detect cycle .If we fill negative infinity value at the diagonal of matrix and run algorithm than matrix of predecessors will contain also all cycles in the graph (the diagonal will not contain only zeros ,if there is a cycle in the graph).

2.5 Time complexity

The time complexity for each algorithm is illustrated in Table I; *n* represents the total number of vertices, and *m* is the total number of edges.

Name	Time complexity
Dijkstra's Algorithm	n^2+m
Bellman Ford Algorithm	$O(n^3)$
Floyd–Warshall Algorithm	nm

3. CONCLUSIONS AND FUTURE WORK

Algorithms are acceptable in terms of their overall performance in solving the shortest path problem. All of these algorithms produce only one solution. improved in finding the shortest path or distance between two places in a map that The computed time complexity for each of the Dijkstra's, Floyd-Warshall and Bellman-Ford algorithms show that these represents any types of networks. In addition, other artificial intelligence techniques such as fuzzy logic and neural networks can also be implemented in improving existing shortest path algorithms in order to make them more intelligent and more efficient.

REFERENCES

[1] F. Benjamin Zhan "Three Fastest Shortest Path Algorithms on Real Road Networks Data Structures and Procedures "vol.1, no.1, pp. 70-82, 1997

[2] Kairanbay Magzhan, Hajar Mat Jani " A Review And Evaluations Of Shortest Path Algorithms"International Journal Of Scientific & Tec-Hnology Research Volume 2, Issue 6, June 2013

[3]Arjun RK1, Pooja Reddy2, Shama3, M. Yamuna4- "Research On The Optimization Of Dijkstra's Algorithm And Its Applications"International Journal of Science, Technology

& Management www.ijstm.comVolume No 04, Special Issue No. 01, April 2015 ISSN (online): 2394-1537

[4] S.G.Shirinivas,S.Vetrivel,Dr. N.M.Elango"Applications Of Graph Theory InComputer Science An Overview"-S.G. Shrinivas et. al. / International Journal of Engineering Science and Technology Vol. 2(9), 2010, 4610-4621

[5] Pooja Singal ,R.S.Chhillar "Dijkstra Shortest Path Algorithm using Global Positioning System" *International Journal of Computer Applications (0975 – 8887) Volume 101– No.6, September 2014*

[6]Dijkstra's Algorithm, Available - http://www.cs.cornell.edu/~wdtseng/icpc/notes/graph_part2.pdf

[7] Floyd-Warshall Algorithm, Available at http://lcm.csa.iisc.ernet.in/dsa/node164.html

[8]Bellman-Ford A lgorithm, Available at http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/