Airo International Research Journal

Volume XVI, ISSN: 2320-3714
April, 2018
Impact Factor 0.75 to 3.19

UGC Approval Number 63012

STUDY AND ANALYSIS IN DIJKSTRA'S ALGORITHM AND BELLMAN FORD ALGORITHM ON RUN-TIME BASIS & IMPLEMENTATION IN ANGIOGRAPHY SYSTEM

Name: Albia Maqbool
Affilation Shri Venkateshwara University, Gajraula, Amroha, (UP)

Name: Dr. Manoj Kumar[2]
Affilation: Shri Venkateshwara University, Gajraula, Amroha, (UP)

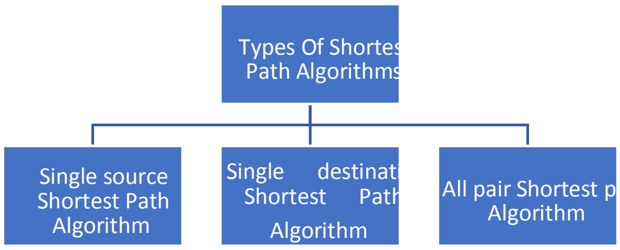
ABSTRACT

Many applications like transportation and communication network system or any type of network use shortest path algorithm for which we can find out the shortest path between two nodes. In the Single source shortest path algorithm, a shortest path is calculating from one node to another node. In this paper, I have analysis and compared the results of the shortest path algorithms (Dijkstra, Bellman Ford) on the basis of running time, whose running time is minimum that algorithm will be best algorithm for shortest route. I am using C# programming language as a source code to compare among the algorithms. I have also compared the algorithms on the running basis of time complexity and space. I have tried to give some advantages and disadvantages of both the algorithms also, after find out best algorithm we can implement it in the angiography system which helps in human life also.

KEYWORDS: Dijkstra's Algorithm, Bellman Ford Algorithm, Floyd's algorithm, Shortest Path, Angiography System.

INTRODUCTION

Here we consider that a shortest path algorithm problem by which we find out the shortest path between two nodes. There are so many shortest path algorithms depending on the source and destination


Types Of Shortes Path Algorithms
Single source Shortest Path Algorithm
Single destinati Shortest Path Algorithm
All pair Shortest p Algorithm

Types of shortest path algorithm:

a. Single source Shortest Path Algorithm

b. Single destination Shortest Path

c. Algorithm

d. All pair Shortest path Algorithm

SHORTEST PATH ALGORITHM

In Single source shortest path algorithm, we have to find out the shortest path from a source vertex to another vertex. In single destination shortest path algorithm, we have to find out the shortest path from all vertices to a single destination vertex. In All pair shortest path algorithm, we have to find out the shortest path from all vertices to another vertex. Due to the nature of routing applications, we need flexible and efficient shortest path procedures, both from a processing time point of view and also in terms of the memory requirements. In this paper, I am comparing single source shortest path algorithms (Dijkstra's and Bellman Ford)[1].

Here as we mentioned earlier, 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 this section, a graph representation of a map is explained further, and brief descriptions and implementations of the shortest path algorithms being studied are presented.

2. DIJKSTA'S AND BELLMAN FORD ALGORITHM

The working of djkstra's algorithm and bellman ford algorithm is as follows:

1) **Dijkstra's Algorithm-*:*** The algorithm stores all nodes in a priority queue ordered by distance of the node from the root – in the first iteration of the algorithm, only root has distance set to *0*, distance of all other nodes is equal to infinity. Than in each step Dijkstra's algorithm picks from the queue a node with the highest priority (least distance from the root) a processes it and reevaluates distances of all unprocessed descendants of the node. This means that the algorithm checks for all descendants that the following condition holds:

distance +edgeweight<distance

Run time complexity of Dijkstra's algorithm.

Here we find out complexity of Dijkstra's algorithm, for this we have to executes loops every time the main loop, one vertex is extracted from the queue. We assuming that there are V vertices in the graph, the queue may contain $O(V)$ vertices. Each pop operation takes $O(\log V)$ time assuming the heap implementation of priority queues. So the total time required to execute the main loop itself is $O(V \log V)$. In addition, we must consider the time spent in the function expand, which applies the function handle_edge to each outgoing edge. Because expand is only called once per vertex, handle_edge is only called once per edge. It might call push(v'), but there can be at most V such calls during the entire execution, so the total cost of that case arm is at most $O(V \log V)$. The other case arm may be called $O(E)$ times, however, and each call to increase priority takes $O(\lg V)$ time with the heap implementation.

Therefore the total run time is $O(V \log V + E \log V)$, which is $O(E \log V)$ because V is $O(E)$ assuming a connected graph.

2) Bellman Ford Algorithm-: The Bellman-Ford algorithm is based on the relaxation operation. The relaxation procedure takes two nodes as arguments and an edge connecting these nodes. If the distance from the source to the first node plus the edge length is less than distance to the second node, than the first node is denoted as the predecessor of the second node and the distance to the second node is recalculated [(distance(A)+ edge.length]. Otherwise no changes are applied.

```
BELLMAN-FORD(G,w,s)
    INITIALIZE-SINGLE-SOURCE(G,s)
    for i = 1 to |G.V|-1
        for each edge (u,v) ∈ G.E
            RELAX(u,v,w)
    for each edge (u,v) ∈ G.E
        if v.d>u.d + w(u,v)
            return FALSE
    return TRUE

INITIALIZE-SINGLE-SOURCE(G,s)
    for each vertex v ∈ G.V
        v.d = ∞
        v.pi = NIL
    s.d = 0

RELAX(u,v,w)
    if v.d>u.d + w(u,v)
        v.d = u.d + w(u,v)
        v.pi = u
```

Basically the algorithm works as follows:

1. Initialize d's, π's, and set s.d = 0 $\Rightarrow$ O(V)
2. Loop |V|-1 times through all edges checking the relaxation condition to compute minimum distances $\Rightarrow$ (|V|-1) O(E) = O(VE)
3. Loop through all edges checking for negative weight cycles which occurs if any of the relaxation conditions fail $\Rightarrow$ O(E)

The run time of the Bellman-Ford algorithm is O(V + VE + E) = O(VE).

Note that if the graph is a DAG (and thus is known to not have any cycles), we can make Bellman-Ford more efficient by first topologically sorting G (O(V+E)), performing the same initialization (O(V)), and then simply looping through each vertex u in topological order relaxing only the edges in Adj[u] (O(E)). This method only takes O(V + E) time. This procedure (with a few slight modifications) is useful for finding critical paths for PERT charts.

COMPARISON ON THE BASIS OF COMPLEXITY AND SPACE

We consider a graph[G] with the vertices or nodes [V] and the edges[E].Now If we find the complexity of Dijkstra Algorithm with the Bellman Ford i.e.

Algorithm	**Time Complexity**	**Space Complexity**
Dijkstra's Algorithm	O(E+V(log V))	O(V)
Bellman Ford	O(EV)	O(V)

(i). Advantages and Disadvantages:

a) **Dijkstra's Algorithm-** The advantages and disadvantages are as follows:

1. It is a Greedy Algorithm.
2. It doesn't work on negative weight.
3. It can work for directed and undirected graph only.
4. It requires global information.

b) Bellman Ford Algorithm- The advantages and disadvantages are as follows:

1. It is a dynamic Algorithm.
2. It can work on negative weight.
3. It can only work for directed graph.
4. It only requires local information.

6. COMPARISON USING C# CODE

Now, I will determine the efficiency of shortest path algorithm. I have created a window based application to find out the running time of both the algorithms. I have created a WindowFormsApplication1, in which I have created a Form and add a list box to display the running time of Dijkstra's and bellman ford algorithm. I have implemented Dijkstra's algorithm and Bellman Ford algorithm using C# code. I have created two functions for Dijkstra's and Bellman Ford algorithms. From the Form_Load ( ) method, both functions are called and display the shortest path for every node from a single source. And I have used stopwatch to calculate the running time of Dijkstra's algorithm and Bellman Ford algorithm in microseconds. I used Random numbers to generate a graph. A. To strore a Graph : public struct Edge

```
{

publicint   u, v, w;

};
```

int NODES ; /* the number of nodes */ int EDGES; /* the number of edges */ int[]d=new int [10000]; /* d[i] is the minimum distance from source node s to node i */

Double [,] G = new double [1000, 1000];

/* graph to store the graph adjacency matrix */

B. To store the adjacency matrix of graph using Random numbers:

```
Random rn1 = new Random ();

for (m = 0; m    length; m++)

{

For (n = 0; n < length; n++)

{

W [m, n] = rn1.Next (0, 10000);

G [m, n] = w [m, n];

}

}
```

C. To Store the Edges with their weight:

```
k = 0;

For (i = 0; i< NODES; ++i)

{

For (j = 0; j < NODES; ++j)

{

If (w [i, j]!= 0)

{Edges[k].u = i; edges[k].v = j; edges[k].w = w [i, j]; k++;

}

1++;

}

}

EDGES = k;
```

D. To Find out the running time using stopwatch:

```
Stopwatch s = new Stopwatch ();

s.Start();

BellmanFord(source_vertex); /* Call for Bellman Ford Algorithm */ s.Stop();

Long time = s.ElapsedTicks /

Stopwatch.Frequency / (1000L *1000L));

listBox1.Items.Add ("time taken by Bellman ford is"+ time+ microseconds"); s.Start();

Dijkstra(source_vertex); /* Call for Dijkstra’s Algorithm */ s.Stop();

Long time = s.ElapsedTicks /Stopwatch.Frequency / (1000L *1000L)); listBox1.Items.Add ("time taken by Dijkstra’s algorithm is"+ time+"microseconds");
```

First Run

N	Dijkstra's Algorithm	Bellman Ford Algorithm
5	1570	751
10	1617	764
50	1853	4655
100	2777	32026
500	23923	4205010
1000	92550	33416105
	Second Run	
N	Dijkstra's Algorithm	Bellman Ford Algorithm
5	1469	667
10	3570	687
50	1918	9631
100	2921	32822
500	23794	4224362
1000	96896	33603891
	Third Run	
N	Dijkstra's Algorithm	Bellman Ford Algorithm
5	1667	667
10	1455	697
50	1758	4557
100	2644	37841
500	25087	4158252
1000	92649	33594017
	Fourth Run	
N	Dijkstra's	Bellman Ford
5	1560	688
10	1411	678
50	1748	4476
100	2566	31904
500	24285	4196981
1000	92477	34341142

	Fifth Run	
5	1606	770
10	1495	728
50	1659	4486
100	3479	31950
500	24423	4147861
1000	126832	33644137
	Average	
5	1523.8	694.6
10	1919.6	719.8
50	1797.2	5571
100	2867.4	33538.6
500	24292.4	4186523.2
1000	100178.8	33719028.6

Airo International Research Journal

Volume XVI, ISSN: 2320-3714
April, 2018
Impact Factor 0.75 to 3.19 UGC Approval Number 63012

We can observe from this table that for the small number of vertices (N=5, 10) Bellman Ford is taking less time in comparison with Dijkstra's algorithm. And for the large number of vertices (N=50, 100, 500, 1000) Dijkstra's is taking less time in comparison with Bellman Ford.

RESULT ANALYSIS (DIJKSTRA'S AND BELLMAN FORD ON AVG. RUNNING TIME BASIS)

In this study we have studied about two single source shortest path algorithms and their comparison. There is advantage and disadvantage in algorithms. To find the running time of each algorithm I used one Program for comparing the running time (in Microseconds). After running the same program on five different runs (for each different value of N=5, 10, 50, 100, 500, 1000), I calculated the average running time for each algorithm and then showed the result with the help of a chart. From the chart I can conclude that for a small number of nodes (N=5, 10) Bellman Ford is the most efficient algorithm to find out the shortest path.

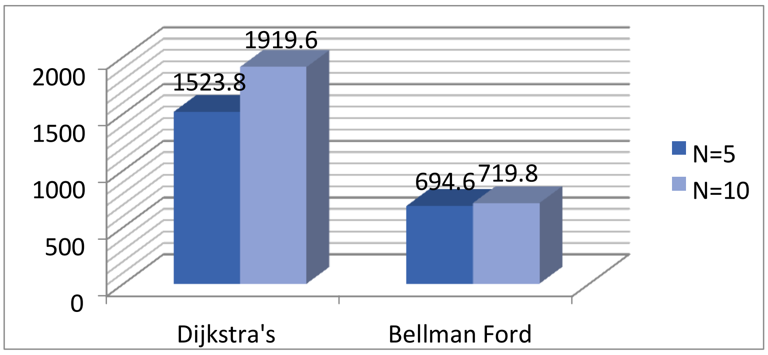


Figure: 1 , Average Running Time For N=5, N=1

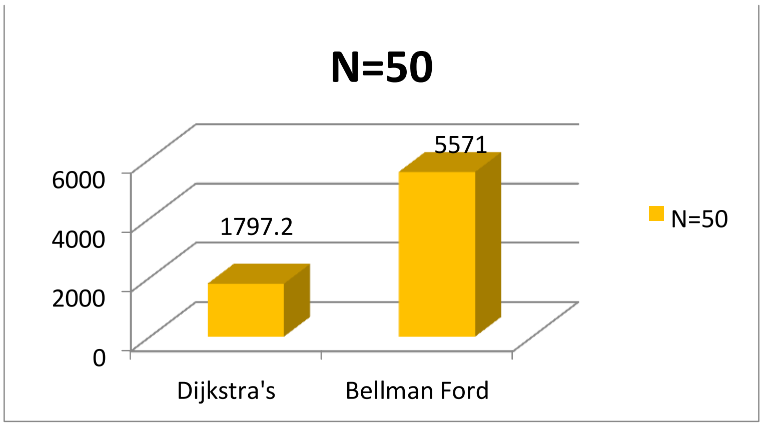
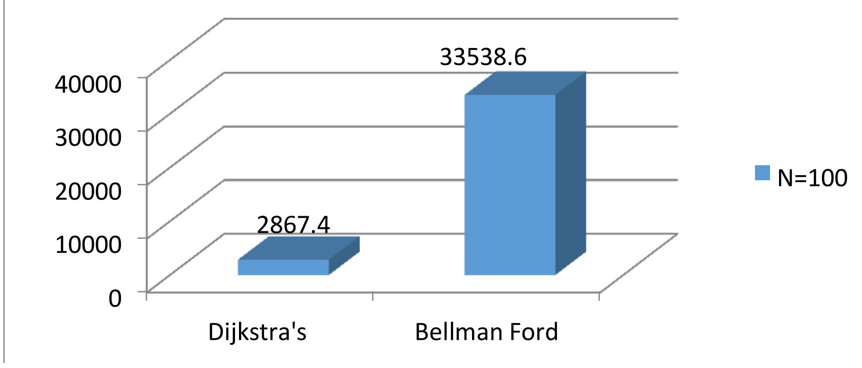


Figure-2, Average Running Time For N=50. For N=50, Dijkstra's Algorithm is efficient algorithm.

For N=100,again Dijkstra's algorithm is efficient algorithm, there is a very big difference in running time of Bellman Ford running time and Dijkstra's algorithm.



N=100

Figure: 3, Average Running Time For N=100

For N=500, 1000, Dijkstra’s algorithm is efficient algorithm in comparison to Bellman Ford.

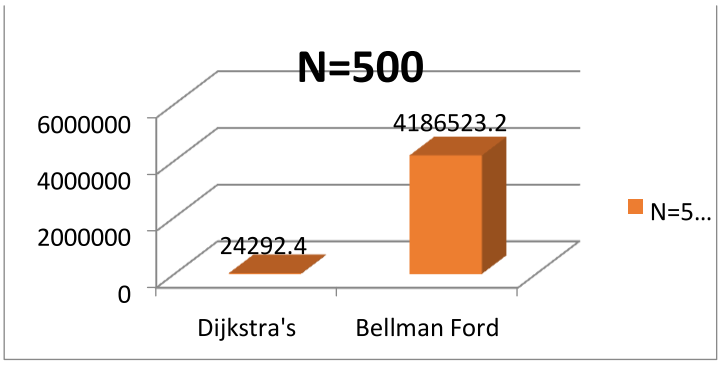


Figure: 4 , Average Running Time For N=500.

By these all charts, we can conclude that for small number of nodes (N < 50) Bellman Ford perform better than Dijkstra’s algorithm. Dijkstra’s algorithm takes twice the running time of Bellman Ford algorithm. But a large number of nodes (N>50) Dijkstra’s algorithm becomes more efficient. For N=50, Bellman Ford algorithm is three times to Dijkstra’s running time. For N=100, Bellman Ford is 11 times to Dijkstra’s algorithm.

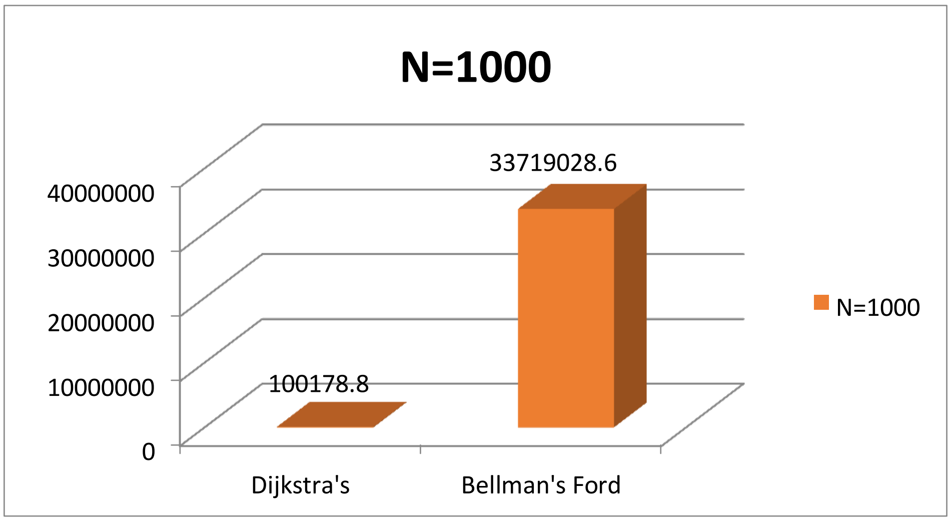


Figure: 5 , Average Running Time For N=1000.

For N=500, 1000, Dijkstra's algorithm outperforms in comparison to Bellman Ford algorithm

COMPARISONS AND EXPERIMENT

When using a naive implementation of Dijkstra's algorithm the time complexity is quadratic, which is much better that the cubic time complexity of the Bellman Ford algorithm. However, Dijkstra's algorithm returns only a subset of Bellman Ford algorithm. Specifically, it returns the shortest path between a given vertex and all other vertices while the Bellman Ford algorithm returns the shortest path between all vertices. It is interesting to note that if you run Dijkstra's algorithm n times, on n different vertices, you will have a theoretical time complexity of O(n* n2)=O(n3). In other words, if you use Dijkstra's algorithm to find a path from every vertex to every other vertex you will have the same efficiency and result as using Bellman Ford algorithm.

In order to test the efficiency of these algorithms I ran several test cases. I implemented Dijkstra's algorithm using a priority queue and I ran each test case 1,000 times. All of the results are aggregations of the 1,000 runs, which gives me a larger, more manageable number. I ran six test cases, for each algorithm, varying the number of vertices in the graph. I used an automated method for creating edge so the sparseness of each graph is always the same.

In my implementation, the Bellman Ford algorithm is actually faster when the number of vertices is small. Only after the number of vertices grows to more than ten does the Dijkstra algorithm become faster. When running Dijkstra's algorithm n times (to get all-pairs shortest-path) the time complexity quickly grows greater that Bellman Ford algorithm. Additional tests show that when running

Dijkstra's algorithm for more than a quarter of the vertices the time complexity exceeds that of Bellman Ford algorithm.

The chart in figure 1 shows the total number of seconds for various values of n (for 1,000 iterations of each algorithm). As the number of vertices doubles from 80 to 160, the time increases by a factor 8, which is cubic time complexity. There is only a small increase in time complexity for Dijkstra’s algorithm over the same values for n. In fact, the time for Dijkstra’s algorithm increases by 2.3 as the value of n doubles, which is a logarithmic time complexity[2].

However, we must keep in mind that Bellman Ford algorithm finds the shortest path between all vertices, while Dijkstra’s algorithm finds the shortest path from a single vertex to all other vertices. Therefore, the comparison between the two is not necessarily valid. If we want to use

Dijkstra’s algorithm to find the shortest path for all vertices we must run it n times – once for each vertex.

The chart in figure 2 shows the total number of seconds for the same values of n, but with each iteration of Dijkstra’s algorithm being repeated n times. The result is that Dijkstra’s algorithm has also found the shortest path between all vertices but we the time requires increases by 6 when the value of n doubles.

RESULT ANALYSIS

Both Bellman Ford and Dijkstra’s algorithm may be used for finding the shortest path between vertices. The biggest difference is that Bellman Ford algorithm finds the shortest path between all vertices and Dijkstra’s algorithm finds the shortest path between a single vertex and all other vertices. **The space overhead for Dijkstra’s algorithm is considerably more than that for** Bellman Ford **algorithm.** In addition, Dijkstra’salgorithm is much easier to implement.

In most cases, for a small values number of vertices, the savings of using Dijkstra’s algorithm are negligible and probably not worth the effort and overhead required. However, when the number of vertices increases the performance of Bellman Ford algorithm drops quickly. Therefore, **the use of**

Dijkstra’s algorithm can provide a solution when performance is a factor. On the other hand, if you will need the shortest path between several vertices on the same graph when we want to consider Dijkstra’s algorithm. In the test case, running the algorithm for more than ¼ of the vertices decreased performance below that of running Bellman Ford algorithm.

CONCLUSION AND FUTURE WORK

On the basis of above the results performance we analysis the results and also we compare execution time of source code of Dijkstra’s Algorithm and Bellman Ford Algorithm on run-time basis then we found conclusion that Dijkstra’s Algorithm is best Algorithm which takes minimum time for execution code for taking large numbers of nodes (See Figure-5 & Table-1) for calculate the shortest path in any network. We also suggest to all The scientist, The Mathematician and students use of Dijkstra’s Algorithm is best for solving shortest route planning in any Networks. If this algorithm is use as a implement it in a angiography system (Medical department) during check out of blockage of vein in human heart then by which this implementation we can save more human’s life.

REFERENCES

[1]. Swati vishnoi, HinaHasmi,3rd International Conference on System Modeling& Advancement in Research Trends (SMART) ,TeerthankerMahaveer University , Moradabad ,2014.

[2]. http://rebustechnologies.com/shortest-path-algorithm-comparison/“Shortest

Path Algorithm Comparison Posted onDecember 5, 2011Bydchamber".

[3]. http://en.algoritmy.net/article/45514/Dijkstras-algorithm

[4]. http://en.algoritmy.net/article/47389/Bellman-Ford-algorithm

[5]. 9th DIMACS Implementation Challenge. Shortest Paths. http://www.dis.uniroma1.it/~challenge9/, 2006.

[6]. R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan. Faster algo- rithms for the shortest path problem. Journal of the ACM, 37(2):213– 223, 1990.

[7]. H. Bast, S. Funke, and D. Matijevic. TRANSIT—ultrafast shortest- path queries with lineartimepreprocessing. In 9th DIMACS Imple- mentation Challenge [1], 2006.

[8]. H. Bast, S. Funke, D. Matijevic, P. Sanders, and D. Schultes. In transit to constant time shortestpath queries in road networks. In Workshop on Algorithm Engineering and Experiments (ALENEX), pages 46–59, 2007.

[9]. H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast routing in road networks with transit nodes. Science, 316(5824):566, 2007.

[10]. Praveen Kumar, Surender Singh, International Journal of Software Computing and Testing eISSN: 2456-2351 Vol. 3: Issue 1.