

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

“A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system”

Abhisek Singh¹

Branch:Electronics & Communication

Department: School of Engineering & Technology

Maharishi University of Information & Technology

(MUIT), Lucknow (U.P)

Dr.Vaishali Singh²

(Professor)

Department: School of Engineering & Technology

Maharishi University of Information & Technology (MUIT), Lucknow (U.P)

ABSTRACT: To Finding the shortest path in a network is a commonly encountered problem. For example you want to reach a target in the real world via the shortest path or in a computer network a network package should be efficiently routed through the network. This theory describes the problem modeled as a graph and the *Dijkstra* algorithm is used to solve the problem.

In computer networks, the routing is based on the shortest path problem. This will help in minimizing the overall costs of setting up computer networks. New technologies such as map-related systems are also applying the shortest path problem. This paper's main objective is to evaluate the Dijkstra's Algorithm, Floyd-Warshall Algorithm, Bellman-Ford Algorithm, and Genetic Algorithm (GA) in solving the shortest path problem. A short review is performed on the various types of shortest path algorithms. Further explanations and implementations of the algorithms are illustrated in graphical forms to show how each of the algorithms works. A framework of the GA for finding optimal solutions to the shortest path problem is presented. The results of evaluating the Dijkstra's, Floyd-Warshall and Bellman-Ford algorithms along with their time complexity conclude the paper.

Index Terms: Bellman-Ford Algorithm, Computer Networks, Dijkstra's Algorithm, Floyd-Warshall Algorithm, Genetic Algorithm (GA), Shortest Path Problem.

1 INTRODUCTION

A *graph* is made out of *nodes* and directed *edges* which define a connection from one node to another node. A node (or vertex) is a discrete position in a graph. Edges can be directed an undirected. Edges have an associated distance (also called costs or weight). The distance between two nodes a and b is labeled as [a,b]. The mathematical description for graphs is $G = \{V,E\}$, meaning that a graph is defined by a set of vertexes (V) and a collection of edges. The *order* of a graph is the number of nodes. The *size* of a graph is the number of edges.

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

Typical graph problems are described in the following list.

- finding the shortest path from a specific node to another node
- finding the maximum possible flow through a network where each edges has a pre-defined maximum capacity (maximum-flow minimum-cut problem)

The following will focus on finding the shortest path from one node to another node in a Network or graph.

The shortest path problem is a problem of finding the shortest path or route from a starting point to a final destination. Generally, in order to represent the shortest path problem we use graphs. A graph is a mathematical abstract object, which contains sets of vertices and edges. Edges connect pairs of vertices. Along the edges of a graph it is possible to walk by moving from one vertex to other vertices. Depending on whether or not one can walk along the edges by both sides or by only one side determines if the graph is a directed graph or an undirected graph. In addition, lengths of edges are often called weights, and the weights are normally used for calculating the shortest path from one point to another point. In the real world it is possible to apply the graph theory to different types of scenarios. For example, in order to represent a map we can use a graph, where vertices represent cities and edges represent routes that connect the cities. If routes are one-way then the graph will be directed; otherwise, it will be undirected. There exist different types of algorithms that solve the shortest path problem. However, only several of the most popular conventional shortest path algorithms along with one that uses genetic algorithm are going to be discussed in this paper, and they are as follows:

1. Dijkstra's Algorithm
2. Floyd-Warshall Algorithm
3. Bellman-Ford Algorithm
4. Genetic Algorithm (GA)

Other than GA, nowadays, there are also many intelligent shortest path algorithms that have been introduced in several past research papers. For example, the authors in [1] used a heuristic method for computing the shortest path from one point to another point within traffic networks.

obtained by extending the Dijkstra's algorithm [1]. In another paper [2], a heuristic GA was used for solving the single source shortest path (SSSP) problem. Its main goal was to investigate the SSSP problem within the Internet routing setting, particularly when considering the cost of transmitting messages/packets is significantly high, and the search space is normally very large. In a paper by Li, Qi, and Ruan [3], an efficient algorithm named Li-Qi (LQ) was proposed for the SSSP problem with the objective of finding a simple path of the smallest total weights from a specific initial or source vertex to every other vertex within the graph. The ideas of the queue and the relaxation form the basis of this newly introduced algorithm; the vertices may be queued several times, and furthermore, only the source vertex and relaxed vertices are being queued [3].

2 RESEARCH OBJECTIVES

The following list gives the objectives of this research paper:

- To determine and identify the concepts of the shortest path problem in a network problem.
- To determine the representation of graphs in computer in order to solve the shortest path problem, as well as to understand the different basic terms of a graph .
- To explain the general concepts and the implementations of Dijkstra's Algorithm, Floyd-Warshall Algorithm, Bellman-Ford Algorithm, and Genetic Algorithm.
- To evaluate each algorithm, and presents the evaluations' results.

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives, 12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

3 LITERATURE REVIEW

As 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 four shortest path algorithms being studied are presented.

Representation of the Graph (Dijkstra's algorithm)

The *Dijkstra Algorithm* finds the shortest path from a source to all destinations in a directed graph (single source shortest path problem). During this process it will also determine a spanning tree for the graph.

The idea of Dijkstra is simple. Dijkstra partitions all nodes into two distinct sets: unsettled and settled. Initially all nodes are in the unsettled sets, e.g. they must be still evaluated. A node is moved to the settled set if a shortest path from the source to this node has been found.

Initially the distance of each node to the source is set to a very high value.

First only the source is in the set of unsettled Nodes. The algorithm runs until the unsettled Nodes are empty. In each iteration it selects the node with the lowest distance from the source out of the unsettled nodes. It reads all edges which are outgoing from the source and evaluates for each destination node, in the edges which are not yet settled, if the known distance from the source to this node can be reduced while using the selected edge. If this can be done then the distance is updated and the node is added to the nodes which need evaluation.

In pseudocode the algorithm can be described as follows. Please note that Dijkstra also determines the pre-successor of each node on its way to the source.

In order to represent a graph in a computer we will use adjacency matrix a . The dimension of the matrix will be equal to $(n \times n)$, where n is number of vertices in graph. The element of matrix $a[i][j]$ is identified by an edge that connects the i -th and j -th vertices; the value here represents the weight of the corresponding edge. However, if there is no edge between vertices i and j , the value in $(a[i][j])$ will be equal to *infinity*. An array of edges is another common representation of the graph. If m is the number of edges in a graph, then in order to represent the graph we have to use $m \times 3$ two-dimensional arrays; in each row, the first vertex, the second vertex, and the edge that connects them are also stored. The benefit of using an array of edges in comparison to adjacency matrix is when there is more than one edge that connects two vertices we cannot use adjacency matrix in order to represent graph.

Dijkstra's Algorithm: Explanation and Implementation

Here we describe Dijkstra's algorithm for finding the shortest path from one source to all the other vertices in a graph. Afterwards, I provide the source code in C of a simple implementation.

To understand this you should know what a graph is, and how to store one in memory. If in doubt check [this](#) and [this](#).

Another solution for this problem is the [Bellman-Ford algorithm](#).

The Problem

Given the following graph calculate the length of the shortest path from **node 1** to every other node.

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

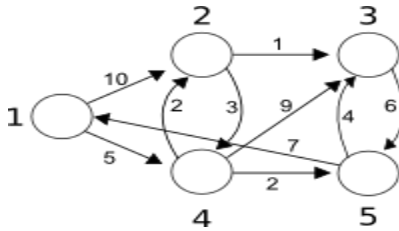
Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives, 12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>



Lets take the nodes **1** and **3**. There are several paths (**1 -> 4 -> 3**, **1 -> 2 -> 3**, etc.), but the shortest of them is **1 -> 4 -> 2 -> 3** of length **8**. Our job is to find it.

The Algorithm

Dijkstra's algorithm is one of the most common solutions to this problem. Even so, it only works on graphs which have **no edges of negative weight**, and the actual speed of the algorithm can vary from **$O(n \cdot \lg(\lg(n)))$** to **$O(n^2)$** .

Introduction

This is the third post in the Graph Traversals – Online Classes.

After learning how to move through a graph, we might be interested in learning more. One interesting problem is determining the shortest path between two vertices of a graph. The problem can be extended and defined in many other forms. I prefer to call it “minimizing the cost”. For e.g.

- When we measure the cost in terms of the distances between vertices, it can be called as the Shortest Path.
- When we measure the cost in terms of the money spent between vertices, it can be called as the Cheapest Path.
- When we measure the cost in terms of the time spent between vertices, it can be called as the Fastest Path.

There can be many more interpretations for this problem. Also, this means that the algorithm can be used to solve variety of problems and not just shortest path ones.

Avoiding Confusions about shortest path

There are few points I would like to clarify before we discuss the algorithm.

- There can be more than one shortest path between two vertices in a graph.
- The shortest path may not pass through all the vertices.
- It is easier to find the shortest path from the source vertex to each of the vertices and then evaluate the path between the vertices we are interested in.
- This algorithm can be used for directed as well as un-directed graphs
- For the sake of simplicity, we will only consider graphs with non-negative edges.

Note : This is not the only algorithm to find the shortest path, few more like Bellman-Ford, Floyd-Warshall, Johnson's algorithm are interesting as well.

Explanation – Shortest Path using Dijkstra's Algorithm

The idea of the algorithm is very simple.

1. It maintains a list of unvisited vertices.

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

2. It chooses a vertex (the source) and assigns a maximum possible cost (i.e. infinity) to every other vertex.

3. The cost of the source remains zero as it actually takes nothing to reach from the source vertex to itself.

4. In every subsequent step of the algorithm it tries to improve(minimize) the cost for each vertex. Here the cost can be distance, money or time taken to reach that vertex from the source vertex. The minimization of cost is a multi-step process.

1. For each unvisited neighbor (vertex 2, vertex 3, vertex 4) of the current vertex (vertex 1) calculate the new cost from the vertex (vertex 1).

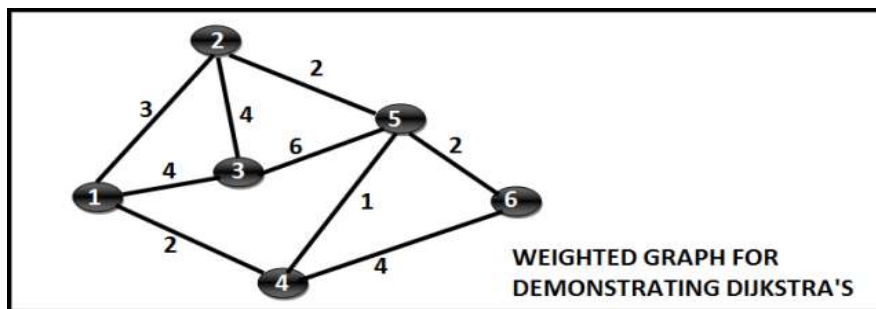
2. For e.g. the new cost of vertex 2 is calculated as the minimum of the two ((existing cost of vertex 2) or (sum of cost of vertex 1 + the cost of edge from vertex 1 to vertex 2))

5. When all the neighbors of the current node are considered, it marks the current node as visited and is removed from the unvisited list.

6. Select a vertex from the list of unvisited nodes (which has the smallest cost) and repeat step 4.

7. At the end there will be no possibilities to improve it further and then the algorithm ends

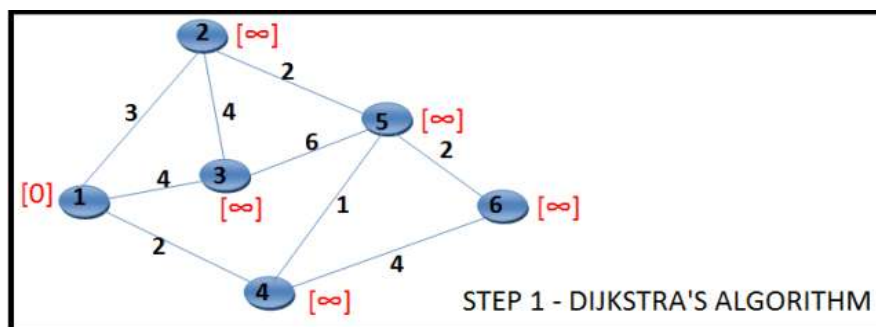
For demonstration we will consider the below graph:



Step Wise Execution

Step 1:

Mark Vertex 1 as the source vertex. Assign a cost zero to Vertex 1 and (infinite to all other vertices). The state is as follows:



Step 2:

For each of the unvisited neighbors (Vertex 2, Vertex 3 and Vertex 4) calculate the minimum cost as min(current cost of vertex under consideration, sum of cost of vertex 1 and

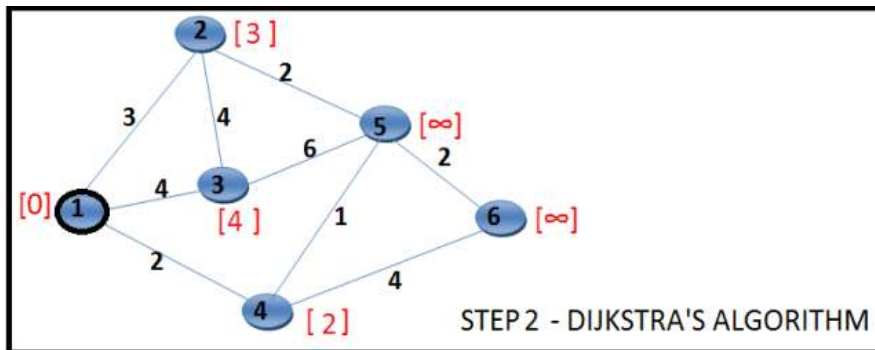
How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

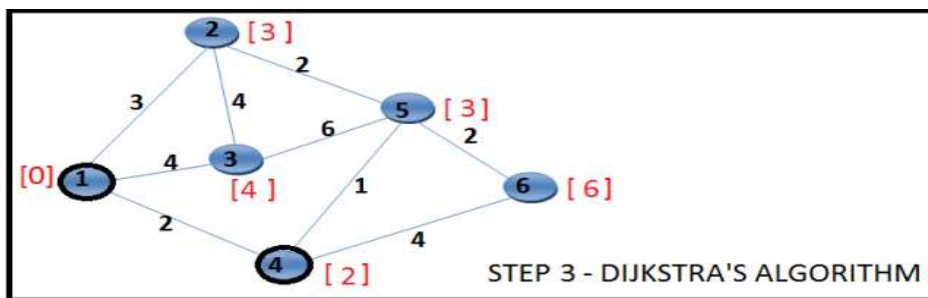
Retrieved from <https://ijeponline.org/index.php/journal/article>

connecting edge). Mark Vertex 1 as visited, in the diagram we border it black. The new state would be as follows:



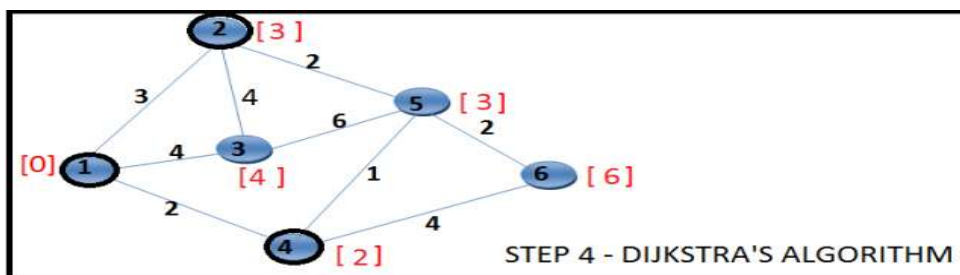
Step 3:

Choose the unvisited vertex with minimum cost (vertex 4) and consider all its unvisited neighbors (Vertex 5 and Vertex 6) and calculate the minimum cost for both of them. The state is as follows:



Step 4:

Choose the unvisited vertex with minimum cost (vertex 2 or vertex 5, here we choose vertex 2) and consider all its unvisited neighbors (Vertex 3 and Vertex 5) and calculate the minimum cost for both of them. Now, the current cost of Vertex 3 is [4] and the sum of (cost of Vertex 2 + cost of edge (2,3)) is $3 + 4 = [7]$. Minimum of 4, 7 is 4. Hence the cost of vertex 3 won't change. By the same argument the cost of vertex 5 will not change. We just mark the vertex 2 as visited, all the costs remain same. The state is as follows:



Step 5:

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

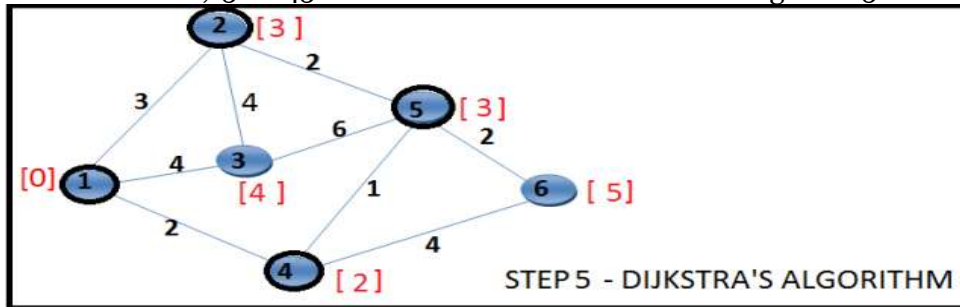
How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

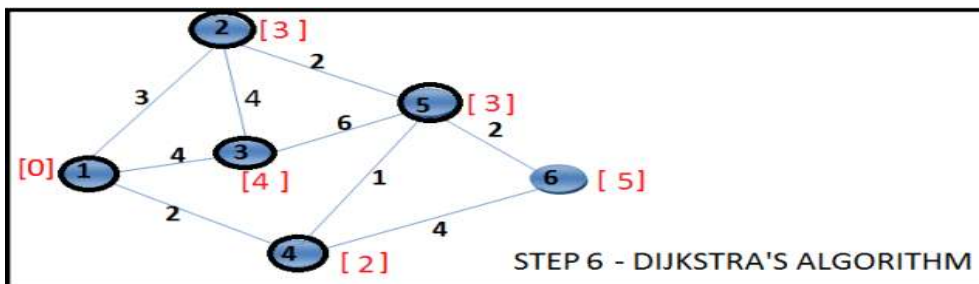
Retrieved from <https://ijeponline.org/index.php/journal/article>

Choose the unvisited vertex with minimum cost (vertex 5) and consider all its unvisited neighbors (Vertex 3 and Vertex 6) and calculate the minimum cost for both of them. Now, the current cost of Vertex 3 is [4] and the sum of (cost of Vertex 5 + cost of edge (5,3)) is $3 + 6 = [9]$. Minimum of 4, 9 is 4. Hence the cost of vertex 3 won't change. Now, the current cost of Vertex 6 is [6] and the sum of (cost of Vertex 5 + cost of edge (3,6)) is $3 + 2 = [5]$. Minimum of 6, 5 is 5. Hence the cost of vertex 6 changes to 5. The state is as follows:



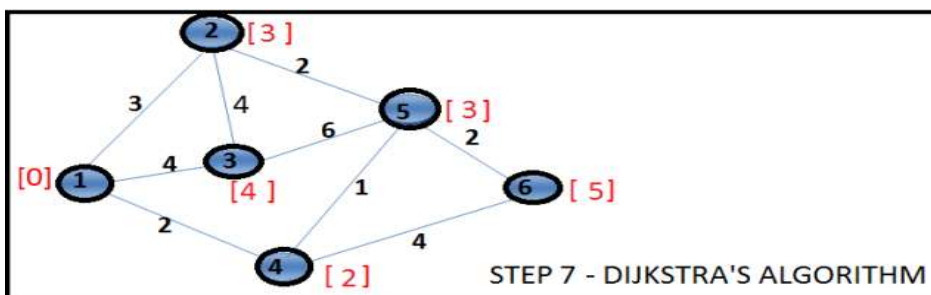
Step 6:

Choose the unvisited vertex with minimum cost (vertex 3) and consider all its unvisited neighbors (none). So mark it visited. The state is as follows:



Step 7:

Choose the unvisited vertex with minimum cost (vertex 6) and consider all its unvisited neighbors (none). So mark it visited. The state is as follows:



Now there is no unvisited vertex left and the execution ends. At the end we know the shortest paths for all the vertices from the source vertex 1. Even if we know the shortest path length, we do not know the exact list of vertices which contributes to the shortest path until we maintain them separately or the data structure supports it.

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

Pseudo Code

```
function dijkstra(G, S)
```

```
    dist[source] <- 0
```

```
    previous[source] <- NULL
```

```
    for each vertex V in G
```

```
        if V NOT S
```

```
            dist[V] <- infinite
```

```
            previous[V] <- NULL
```

ADD V to Q \\ if we choose the Q as a Priority Queue, extracting minimum will be easy.

```
    while Q IS NOT EMPTY
```

```
        U <- Extract MIN from Q
```

```
        for each unvisited neighbour V of U
```

```
            tempDist <- dist[U] + edge_weight(U, V)
```

```
            if tempDist < dist[V]
```

```
                dist[V] <- tempDist
```

```
                previous[V] <- U
```

```
    return dist[], previous[]
```


How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

Code: Shortest Path Using Dijkstra Algorithm

```
public class Dijkstra {  
  
    public void dijkstra(Vertex source, List vertices) {  
        source.min = 0;  
        PriorityQueue Q = new PriorityQueue();  
        for(Vertex v : vertices){  
            Q.add(v);  
        }  
  
        while (!Q.isEmpty()) {  
            Vertex u = Q.poll();  
  
            // Visit each edge exiting u  
            for (Edge e : u.incidentEdges) {  
                Vertex v = e.end;  
                int weight = e.weight;  
                int tempDistance = u.min + weight;  
                if (tempDistance < v.min) {  
                    Q.remove(v); // remove to re-insert it in the queue with the new cost.  
                    v.min = tempDistance;  
                    v.previous = u;  
                    Q.add(v);  
                }  
            }  
        }  
    }  
}
```

To download the complete code please visit [techieme github repository](#).


Analysis of the algorithm

The outer loop runs for $|V|$ times. The inner loop runs for $|V-1|$ times for a complete graph as each vertex has $|V-1|$ edges. Also, for each iteration of the inner loop we do an extract Min and a reduce Key operation for the vertex.

Hence the total running time has an upper bound of $O(|V| * |V-1|)$. This is the upper bound, $O(|V|^2)$

Point worth noting is that the complexity will actually depend on the implementation of the data structure which is used as the Queue. We will be discussing and comparing all the running times for the three shortest path algorithms together in coming posts.

For each vertex within a graph we assign a label that determines the minimal length from the starting point s to other vertices v of the graph. In a computer we can do it by declaring an array $d[]$. The algorithm works sequentially, and in each step it tries to decrease the value of the label of the vertices. The algorithm stops when all vertices have been visited. The label at the starting point s is equal to zero ($d[s]=0$); however, labels in other vertices v are equal to *infinity* ($d[v]=\infty$), which means that the length from the starting point s to other vertices is unknown. In a computer we can just use a very big number in order to represent *infinity*. In addition, for each vertex v we have to identify whether it has been visited or not. In order to do that, we declare an array of *Boolean* type called $u[v]$, where initially, all vertices are

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

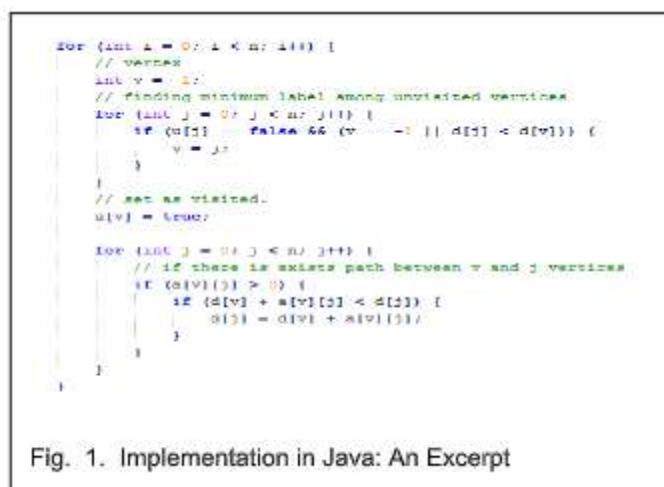
assigned as unvisited ($u[v] = false$). The Dijkstra's algorithm consists of n iterations. If all vertices have been visited, then the algorithm finishes; otherwise, from the list of unvisited vertices we have to choose the vertex which has the minimum (smallest) value at its label (At the beginning, we will choose a starting point s). After that, we will consider all neighbors of this vertex (Neighbors of a vertex are those vertices that have common edges with the initial vertex). For each unvisited neighbor we will consider a new length, which is equal to the sum of the label's value at the initial vertex v ($d[v]$) and the length of edge l that connects them. If the resulting value is less than the value at the label, then we have to change the value in that label with the newly obtained value [4].

$$d[\text{neighbors}] = \min(d[\text{neighbors}], d[v] + l) \quad (1)$$

After considering all of the neighbors, we will assign the initial vertex as visited ($u[v] = true$). After repeating this step n times, all vertices of the graph will be visited and the algorithm finishes or terminates. The vertices that are not connected with the starting point will remain by being assigned to *infinity*. In order to restore the shortest path from the starting point to other vertices, we need to identify array $p[]$, where for each vertex, where $v \neq s$, we will store the number of vertex $p[v]$, which penultimate vertices in the shortest path. In other words, a complete path from s to v is equal to the following statement [5]

$$P = (s, \dots, p[p[p[v]]], p[p[v]], p[v], v) \quad (2)$$

Fig. 1 shows an excerpt of the Dijkstra's algorithm, which is written in Java.



```
for (int i = 0; i < nr; i++) {
    // vertex
    int v = -1;
    // finding minimum label among unvisited vertices.
    for (int j = 0; j < nr; j++) {
        if (u[j] == false && (v == -1 || d[i] < d[j])) {
            v = j;
        }
    }
    // set as visited.
    u[v] = true;

    for (int j = 0; j < nr; j++) {
        // if there is exists path between v and j vertices
        if (a[v][j] > 0) {
            if (d[v] + a[v][j] < d[j]) {
                d[j] = d[v] + a[v][j];
            }
        }
    }
}
```

Fig. 1. Implementation in Java: An Excerpt

Floyd-Warshall Algorithm: Explanation and Implementation

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 update 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 $\{0, 1, 2, \dots, k-1\}$ as intermediate vertices. For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

- 1) k is not an intermediate vertex in shortest path from i to j . We keep the value of $dist[i][j]$ as it is.
- 2) k is an intermediate vertex in shortest path from i to j . We update the value of $dist[i][j]$ as $dist[i][k]$

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

+ dist[k][j].

The following figure is taken from the Cormen book. It shows the above optimal substructure property in the all-pairs shortest path problem.

Consider the graph G , where vertices were numbered from 1 to n . The notation $dijk$ means the shortest path from i to j , which also passes through vertex k . Obviously if there is exists edge between vertices i and j it will be equal to $dijo$, otherwise it can assigned as *infinity*. However, for other values of $dijk$ there can be two choices: (1) If the shortest path from i to j does not pass through the vertex k then value of $dijk$ will be equal to $dijk-1$. (2) If the shortest path from i to j passes through the vertex k then first it goes from i to k , after that goes from k to j . In this case the value of $dijk$ will be equal to $dikk-1 + dkjk-1$. And in order to determine the shortest path we just need to find the minimum of these two statements [6]:

$$dijo = \text{the length of edge between vertices } i \text{ and } j \quad (3)$$

$$dijk = \min(dijk-1, dikk-1 + dkjk-1) \quad (4)$$

Fig. 2 shows an excerpt of the Floyd-Warshall algorithm, which is written in Java.

```
for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (d[i][j] > d[i][k] + d[k][j]) {
                /*
                 * d[i][j] = is equal to
                 * the shortest path from i-th to j-th vertices.
                 */
                d[i][j] = d[i][k] + d[k][j];
            }
        }
    }
}
```

Fig. 2. Implementation in Java: An Excerpt

Bellman-Ford Algorithm: Explanation and Implementation

Given a graph and a source vertex src in graph, find shortest paths from src to all vertices in the given graph. The graph may contain negative weight edges. We have discussed Dijkstra's algorithm for this problem. Dijkstra's algorithm is a Greedy algorithm and time complexity is $O(V \log V)$ (with the use of Fibonacci heap). *Dijkstra doesn't work for Graphs with negative weight edges, Bellman-Ford works for such graphs. Bellman-Ford is also simpler than Dijkstra and suites well for distributed systems. But time complexity of Bellman-Ford is $O(VE)$, which is more than Dijkstra.*

Algorithm

Following are the detailed steps.

Input: Graph and a source vertex src

Output: Shortest distance to all vertices from src . If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

1) This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array $dist[]$ of size $|V|$ with all values as infinite except $dist[src]$ where src is source vertex.

2) This step calculates shortest distances. Do following $|V|-1$ times where $|V|$ is the number of vertices in given graph.

.....**a)** Do following for each edge $u-v$

.....If $dist[v] > dist[u] + \text{weight of edge } uv$, then update $dist[v]$

..... $dist[v] = dist[u] + \text{weight of edge } uv$

3) This step reports if there is a negative weight cycle in graph. Do following for each edge $u-v$

.....If $dist[v] > dist[u] + \text{weight of edge } uv$, then "Graph contains negative weight cycle"

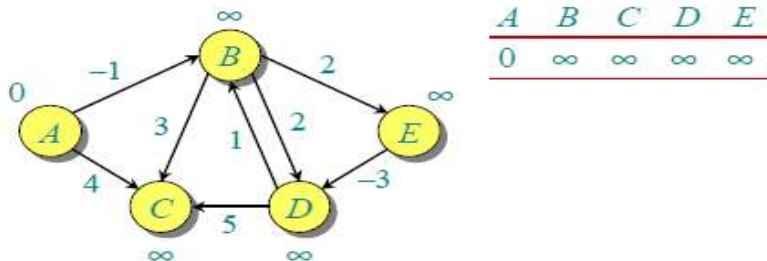
The idea of step 3 is, step 2 guarantees shortest distances if graph doesn't contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

How does this work? Like other Dynamic Programming Problems, the algorithm calculate shortest paths in bottom-up manner. It first calculates the shortest distances for the shortest paths which have at-most one edge in the path. Then, it calculates shortest paths with at-most 2 edges, and so on. After the i th iteration of outer loop, the shortest paths with at most i edges are calculated. There can be maximum $|V| - 1$ edges in any simple path, that is why the outer loop runs $|v| - 1$ times. The idea is, assuming that there is no negative weight cycle, if we have calculated shortest paths with at most i edges, then an iteration over all edges guarantees to give shortest path with at-most $(i+1)$ edges (Proof is simple, you can refer this or MIT Video Lecture)

Example

Let us understand the algorithm with following example graph. The images are taken from this source.

Let the given source vertex be 0. Initialize all distances as infinite, except the distance to source itself. Total number of vertices in the graph is 5, so *all edges must be processed 4 times*.



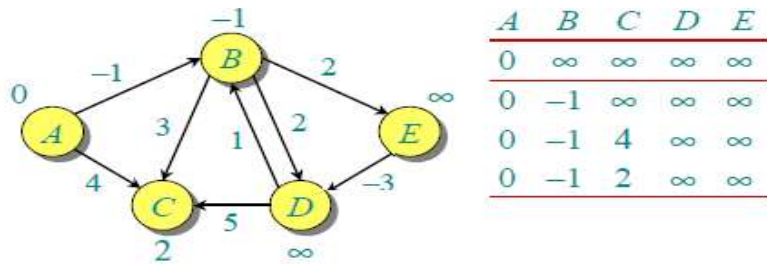
Let all edges are processed in following order: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D). We get following distances when all edges are processed first time. The first row in shows initial distances. The second row shows distances when edges (B,E), (D,B), (B,D) and (A,B) are processed. The third row shows distances when (A,C) is processed. The fourth row shows when (D,C), (B,C) and (E,D) are processed.

How to Cite:

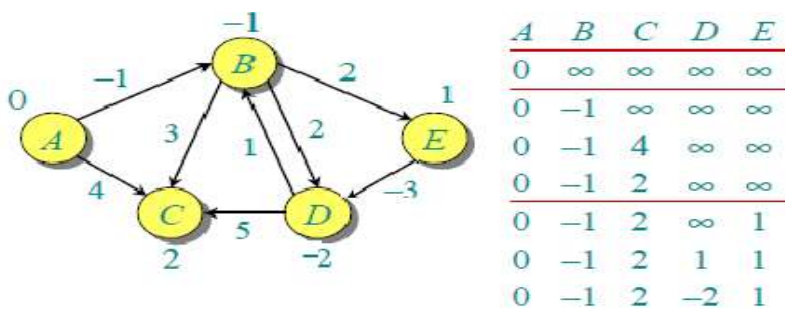
Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>



The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time (The last row shows final values).



The second iteration guarantees to give all shortest paths which are at most 2 edges long. The algorithm processes all edges 2 more times. The distances are minimized after the second iteration, so third and fourth iterations don't update the distances.

In comparison to Dijkstra's algorithm, the Bellman-Ford algorithm admits or acknowledges the edges with negative weights. That is why, a graph can contain cycles of negative weights, which will generate numerous number of paths from

Genetic Algorithm

Find global minima for highly nonlinear problems

A genetic algorithm (GA) is a method for solving both constrained and unconstrained optimization problems based on a natural selection process that mimics biological evolution. The algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm randomly selects individuals from the current population and uses them as parents to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution.

You can apply the genetic algorithm to solve problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear.

The genetic algorithm differs from a classical, derivative-based, optimization algorithm in two main ways, as summarized in the following table.

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

Classical Algorithm	Genetic Algorithm
Generates a single point at each iteration. The sequence of points approaches an optimal solution.	Generates a population of points at each iteration. The best point in the population approaches an optimal solution.
Selects the next point in the sequence by a deterministic computation.	Selects the next population by computation which uses random number generators.

In practice, therefore, we can implement this genetic model of computation by having arrays of bits or characters to represent the

CHROMOSOMES. Simple bit manipulation operations allow the implementation of CROSSOVER, MUTATION and other operations. Although a substantial amount of research has been performed on variable-length strings and other structures, the majority of work with GENETIC ALGORITHMS is focussed on fixed-length character strings. We should focus on both this aspect of fixed-lengthness and the need to encode the representation of the solution being sought as a character string, since these are crucial aspects that distinguish GENETIC PROGRAMMING, which does not have a fixed length representation and there is typically no encoding of the problem. When the GENETIC ALGORITHM is implemented it is usually done in manner that involves the following cycle: Evaluate the FITNESS of all of the INDIVIDUALS in the POPULATION. Create a new population by performing operations such as CROSSOVER, fitness-proportionate REPRODUCTION and MUTATION on the individuals whose fitness has just been measured. Discard the old population and iterate using the new population. One iteration of this loop is referred to as a GENERATION. There is no theoretical reason for this as an implementation model. Indeed, we do not see this punctuated behavior in POPULATIONS in nature as a whole, but it is a convenient implementation model. The first GENERATION (generation 0) of this process operates on a

POPULATION of randomly generated INDIVIDUALS. From there on, the genetic operations, in concert with the FITNESS measure, operate to improve the population.

PSEUDO CODE

Algorithm GA is

```
// start with an initial time
t := 0;

// initialize a usually random population of individuals
initpopulation P (t);

// evaluate fitness of all initial individuals of population
evaluate P (t);

// test for termination criterion (time, fitness, etc.)
while not done do

// increase the time counter
t := t + 1;
```

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

```
// select a sub-population for offspring production
P' := selectparents P (t);

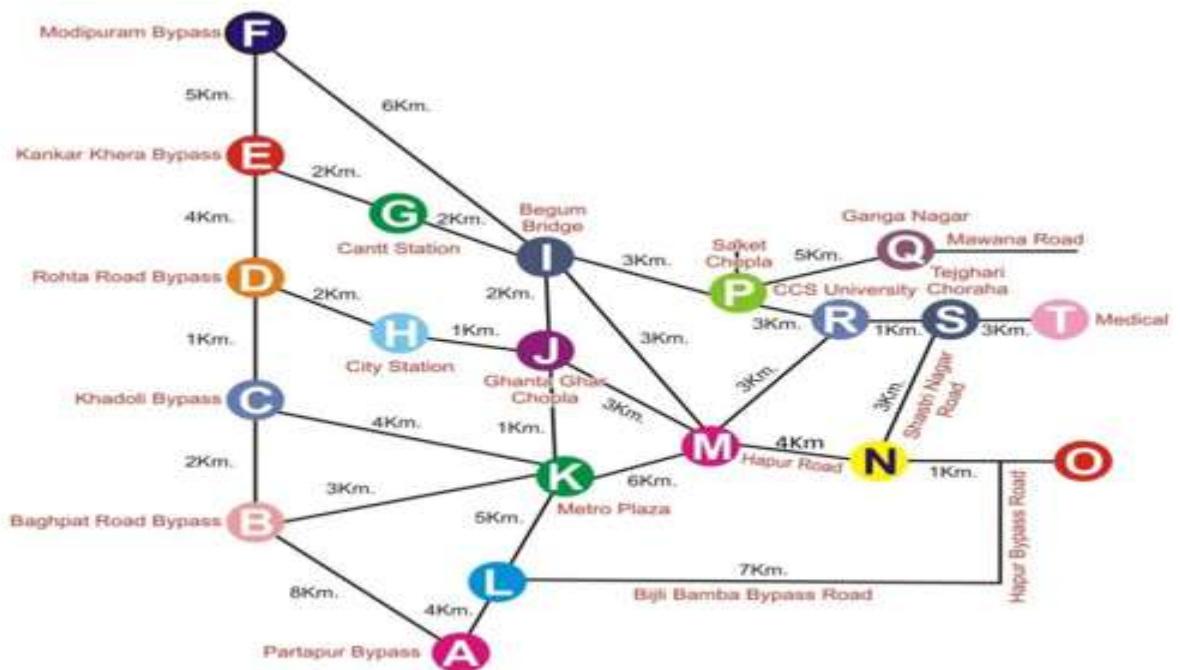
// recombine the "genes" of selected parents
recombine P' (t);

// perturb the mated population stochastically
mutate P' (t);

// evaluate it's new fitness
evaluate P' (t);

// select the survivors from actual fitness
P := survive P,P' (t);
od
end GA.
```

CALCULATION SHORTEST PATH OF MODEL MEERUT CITY BY USING DIJEKSTRA's ALGO.



© 2018 by The Author(s). ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system
International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

SOLUTION OF SHORTEST PATH FOR METRO TRAIN IN MEERUT CITY

6.1 CALCULATION SHORTEST PATH OF MODEL MEERUT CITY.

i	L(A)	L(B)	L(C)	L(D)	L(E)	L(F)	L(G)	L(H)	L(I)	L(J)	L(K)	L(L)	L(M)	L(N)	L(O)	L(P)	L(Q)	L(R)	L(S)	L(T)
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	8	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	0	8	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	0	8	10	11	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	0	8	10	11	15	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
5	0	8	10	11	15	18	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
6	0	8	10	11	15	18	14	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
7	0	8	10	11	15	18	14	11	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
8	0	8	10	11	15	18	14	11	12	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
9	0	8	10	11	15	18	14	11	12	10	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
10	0	8	10	11	15	18	14	11	12	10	9	∞	∞	∞	∞	∞	∞	∞	∞	∞
11	0	8	10	11	15	18	14	11	12	10	9	4	∞	∞	∞	∞	∞	∞	∞	∞
12	0	8	10	11	15	18	14	11	12	10	9	4	15	∞	∞	∞	∞	∞	∞	∞
13	0	8	10	11	15	18	14	11	12	10	9	4	15	12	∞	∞	∞	∞	∞	∞
14	0	8	10	11	15	18	14	11	12	10	9	4	15	12	11	∞	∞	∞	∞	∞
15	0	8	10	11	15	18	14	11	12	10	9	4	15	12	11	15	∞	∞	∞	∞
16	0	8	10	11	15	18	14	11	12	10	9	4	15	12	11	15	20	∞	∞	∞
17	0	8	10	11	15	18	14	11	12	10	9	4	15	12	11	15	20	16	∞	∞
18	0	8	10	11	15	18	14	11	12	10	9	4	15	12	11	15	20	16	15	∞
19	0	8	10	11	15	18	14	11	12	10	9	4	15	12	11	15	20	16	15	18

(TABLE -RESULT CALCULATION OF SHORTEST PATH IN MEERUT CITY USING DIJKSTRA’S ALGORITHM METHOD, WE CAN SEE ANY DISTANCE IN KM FROM A “PARTAPUR BYPASS” CITY TO T “MEDICAL” USING SHORTEST PATH A->L->O->N->S-T IS 18 KM.).

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives, 12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

CONCLUSION AND FUTURE SCOPE

This articles focuses on the Image Analysis of Remote Sensing Data Integrating Spectral, Temporal and Spatial Features of Objects in the place of satellite image processing and then we analysis shortest path route using Dijkstra's algorithm . We have used the multi-spectral remote sensing facts to locate the spectral signature of one of a kind objects of the Meerut city for the land cover classification, how the use of land modifications in accordance to time and also performed the temporal analysis to analyze the impact of climate over the surface. Some band combinations of remote sensed data are advantageous in the land cover classification. Spatial distributions of land cowl sorts such as roads; city area, agriculture land, and water resources can without challenge be interpreted via taking their Normalized difference vegetation index (NDVI). We have carried out the ground survey to achieve the threshold values of NDVI and on the foundation of it we have obtained the False Color Composite (FCC) of categorized objects. The labeled facts should be used for municipal planning and management. The long-term goal of the thesis is to optimize the land use pattern for economically and environmentally sustainable city development. We can implanted it in metro instruct venture in Meerut city the use of Image analysis technique.

We can recommend to all The scientist we need to use image analysis approach before beginning any underground undertaking barring touching any building . By the definition of shortest route Dijkstra s algorithm we advise to all The Mathematician and college students use of Dijkstra's Algorithm is exceptional for fixing shortest route planning in any Networks the usage of photo analysis technique .

In Meerut city Using Dijkstra's Algorithm (according to table method) distance from A "Partapur Bypass" city to T "Medical" using shortest path A->L->O->N->S->T Distance is 18 Km.

REFERENCES

1. Begni Gérard, Escadafal Richard, Fontannaz Delphine and Hong-Nga Nguyen Anne-Thérèse, 2005. Remote sensing: a tool to monitor and assess desertification. Les dossiers thématiques du CSFD. Issue 2. 44 pp.
2. NASA (1986), *Report of the EOS data panel*, Earth Observing System, Data and Information System, Data Panel Report, Vol. IIa., NASA Technical Memorandum 87777, June 1986, 62 pp.
3. C. L. Parkinson, A. Ward, M. D. King (Eds.) *Earth Science Reference Handbook - A Guide to NASA's Earth Science Program and Earth Observing Satellite Missions*, National Aeronautics and Space Administration Washington, D.C.
4. GRAS-SAF (2009), *Product User Manual*, GRAS Satellite Application Facility, Version 1.2.1, 31 March 2009.
5. Anderson, J.T., Gregory, R.S. and Collins, W.T. (2002). "Acoustic classification of marine habitats in coastal Newfoundland." *ICES Journal of Marine Science* 59(1): 156-167.
6. Andrieux, N., Delachartre, P., Vray, D., and Gimenez, G. (1995). "Lake-bottom recognition using a wideband sonar system and time-frequency analysis." *Journal of the Acoustical Society of America* 98(1): 552-559.

© 2018 by The Author(s).  ISSN: 1307-1637 International journal of economic perspectives is licensed under a Creative Commons Attribution 4.0 International License.

Corresponding author: **Abhisek Singh and Dr.Vaishali Singh**

Submitted: 27Oct 2018, Revised: 09 Nov 2018, Published : Dec 2018

How to Cite:

Abhisek Singh and Dr.Vaishali Singh (Dec 2018). A Comparison In Different Types Of Shortest Path Algorithms and using best algorithm their Implementation in Shortest Route network system

International Journal of Economic Perspectives,12(1), 120-136.

Retrieved from <https://ijeponline.org/index.php/journal/article>

7. Anonymous. (2001a). "ECHOplus aids in seabed archaeology; one of the first seatronics products." *Sea Technology* 42(10): 69.

8. Anonymous. (2001b). "ECHOplus sale to national Coral Reef Institute." *Sea Technology* 43(7): 59.

9. Barnhardt, W.A., Kelley, J. T., Dickson, S.M. and Belknap, D.F. (1998). "Mapping the Gulf of Maine with Side-Scan Sonar: A New Bottom-Type Classification for Complex Seafloors." *Journal of Coastal Research* 14(2): 646-659.