# Find The Shortest Path Using Dijkstra Algorithm in C++

Nighthawks

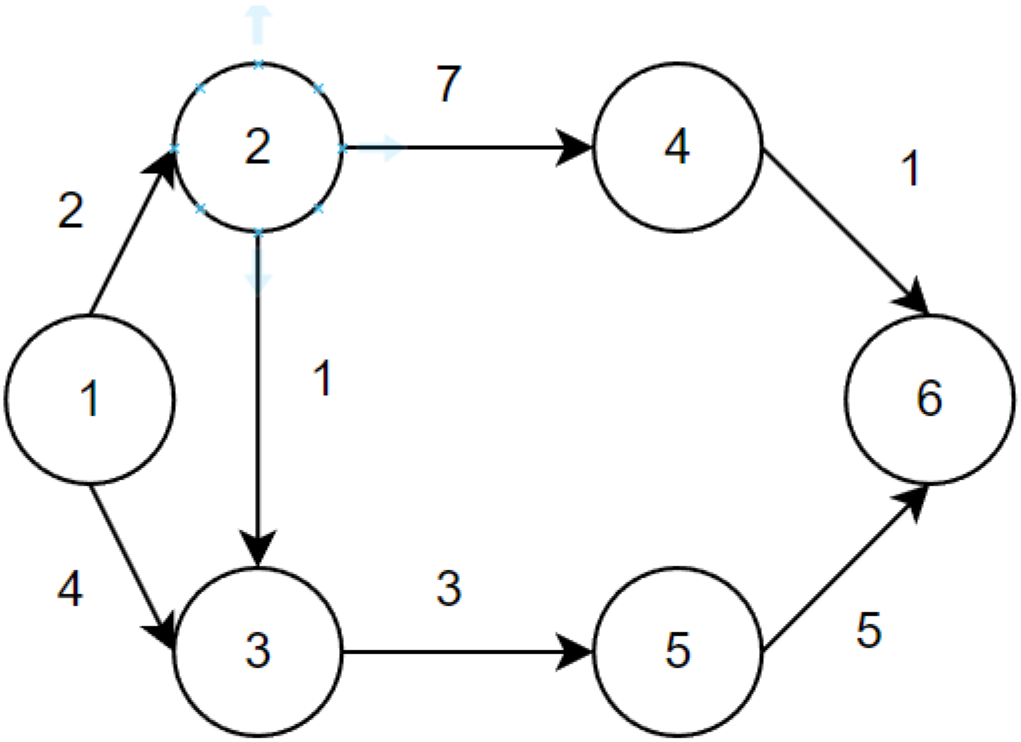
## Abstract

The Dijkstra method is a well-known algorithm for finding the optimum path in shortest-path search problems. The objective of this project is to create a program to find the shortest path between two points and find the time and length for robot from one point to another point. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

## Introduction

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

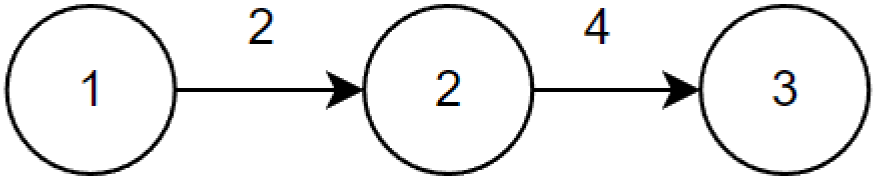
If the weighted graph is given, we need to find out a shortest path from one starting vertex to all other vertices. For example, in the following figure, if we select vertex 1 as starting vertex then we must find out shortest path to all other vertices.



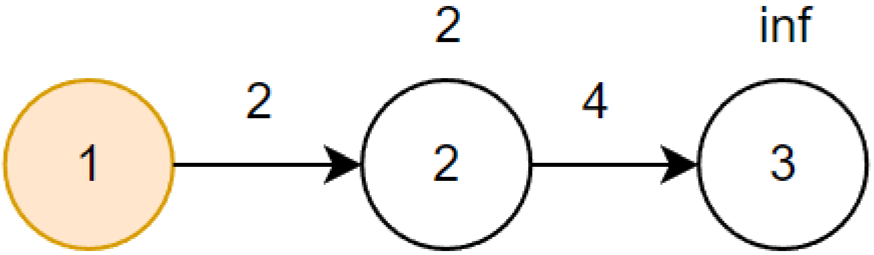
We can select any one vertex as source vertex. Since we have to find out a shortest path so it is a minimization problem. A minimization problem is an optimization problem so optimization problem can solve using greedy method.

Greedy method stated that a problem should be solved in stages by taking one step at time and considering one input at a time to get an optimal solution. Greedy method are also predefined procedures, and we follow that procedures to get a optimal solution.

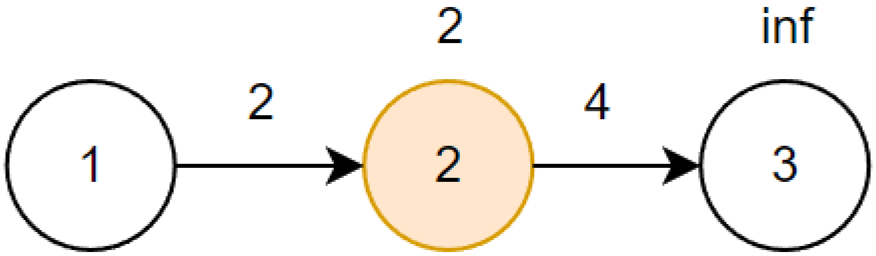
So, Dijkstra algorithm gives a procedure for getting a optimal solution that is minimum result that is shortest path.



For example, in the figure above, if we select 1 as staring vertex and we need to find the shortest path to 2 as well as 3.

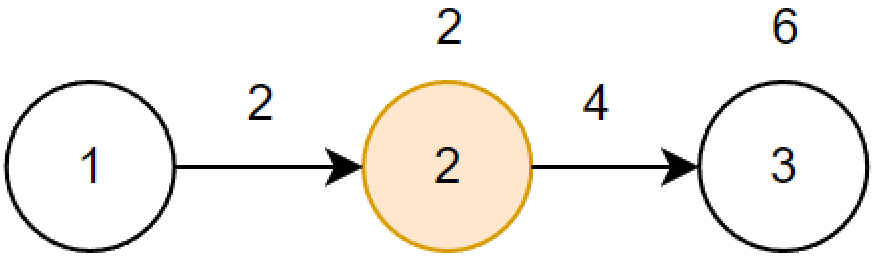


Since there is direct path from vertex 1 to vertex 2, we can the cost from 1 to 2 which total is equal to 2 but there is no direct path to vertex 3, so the total cost is infinity or unknown in this state.



But if follow the Dijkstra algorithm, we will select the first shortest vertex from the starting point. In this state it is vertex 2.

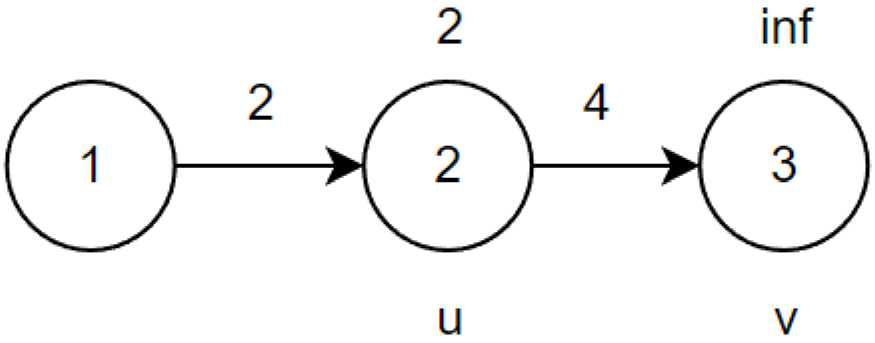
Dijkstra algorithm also stated that once you have selected one of the shortest path then check are that any shortest path from this vertex to another connect vertices.



In the example, since there is direct path from vertex 2 to vertex 3, we can calculate the shortest path from vertex 2 to vertex 3 which total is 6. Then we can change the infinity to 6. This means that there is shortest path from vertex 1 to vertex 3 even there is no direct path between them. Even it is pass through the vertex 2 but it is still considered path.

Dijkstra algorithm always select a vertex with the shortest path then it will updated the shortest path to other vertices if possible and this updating is call relaxation.

Relaxation



Relaxation means if the distance of vertex u plus cost between v and u is less than distance of vertex v so we will modify the distance of v to the distance of u plus cost of an edge u and v.

$$if(d[u] + c[u,v] < d[v])$$

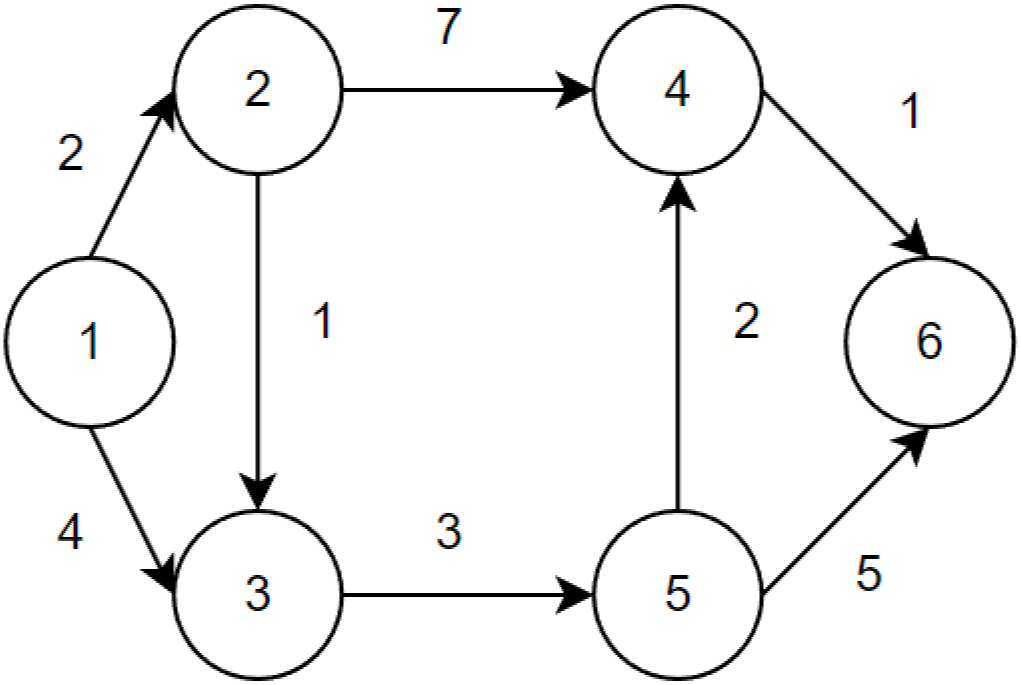
$$d[v] = d[u] + c(u,v)$$

$d[u] = distance\ of\ u\ from\ the\ starting\ vertex$

$d[v] = distance\ of\ v\ from\ the\ starting\ vertex$

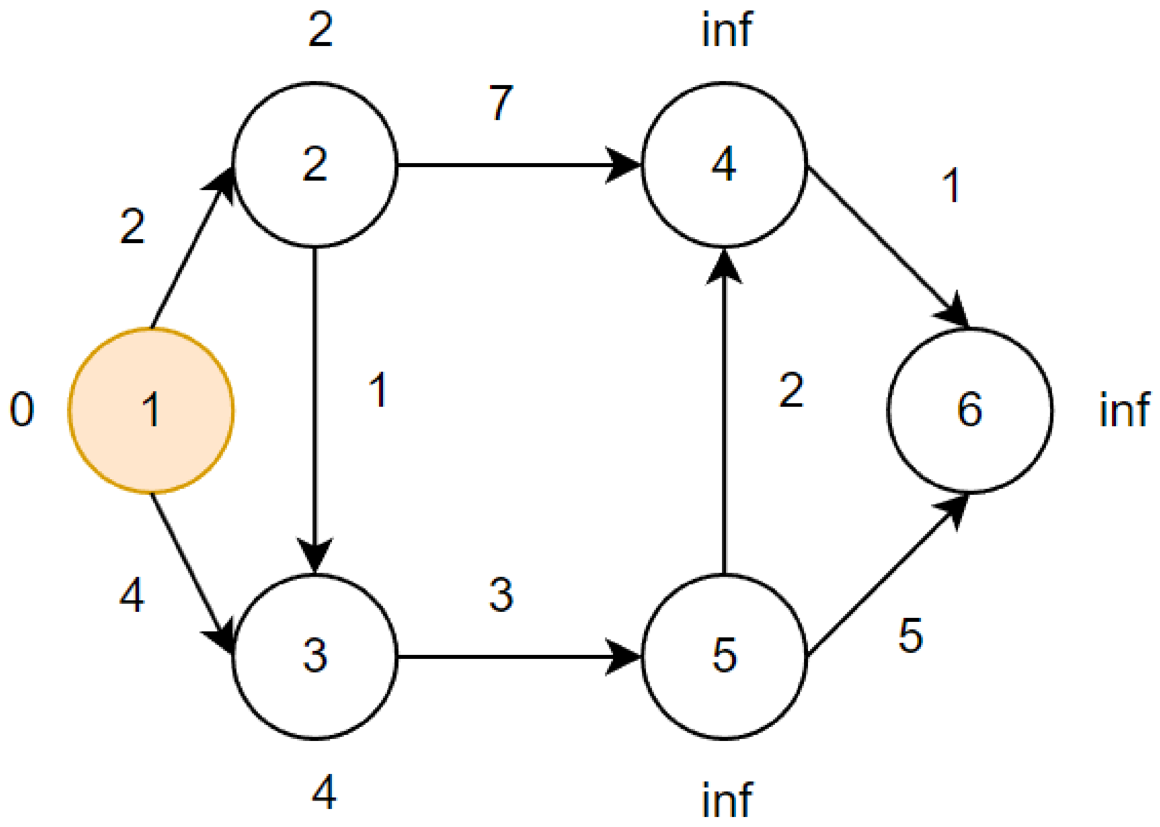
$c[u,v] = cost\ betwen\ vertex\ and\ vertex\ 3$

We do the relaxation with other vertices for the vertex whenever we select a shortest path.



If we select vertex 1 as the starting vertex, now we will give the distance for all the vertices by considering just single edge.

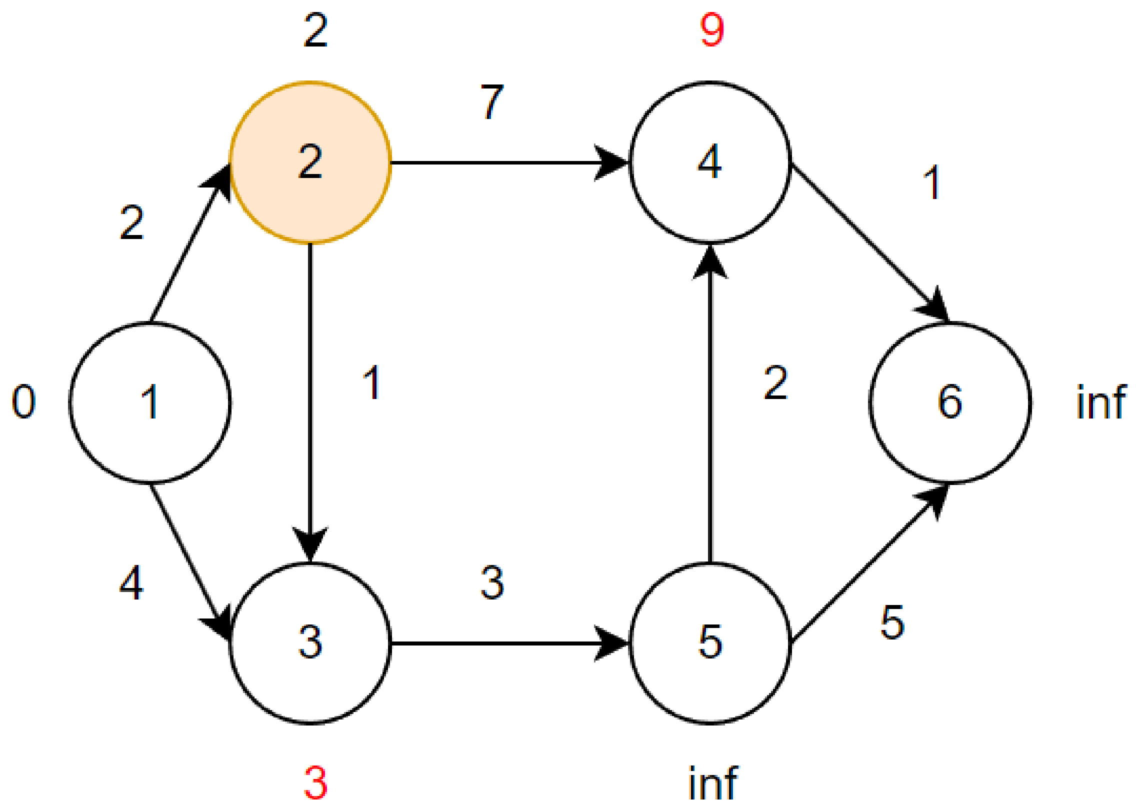
This is the initial thing we should do for any graph.



First, we need to select the shortest path. Since vertex 2 has the shortest path, we will select it. Once we select it, we will perform relaxation. We will check who are connected to vertex 2 which is vertex 4 and vertex 3.

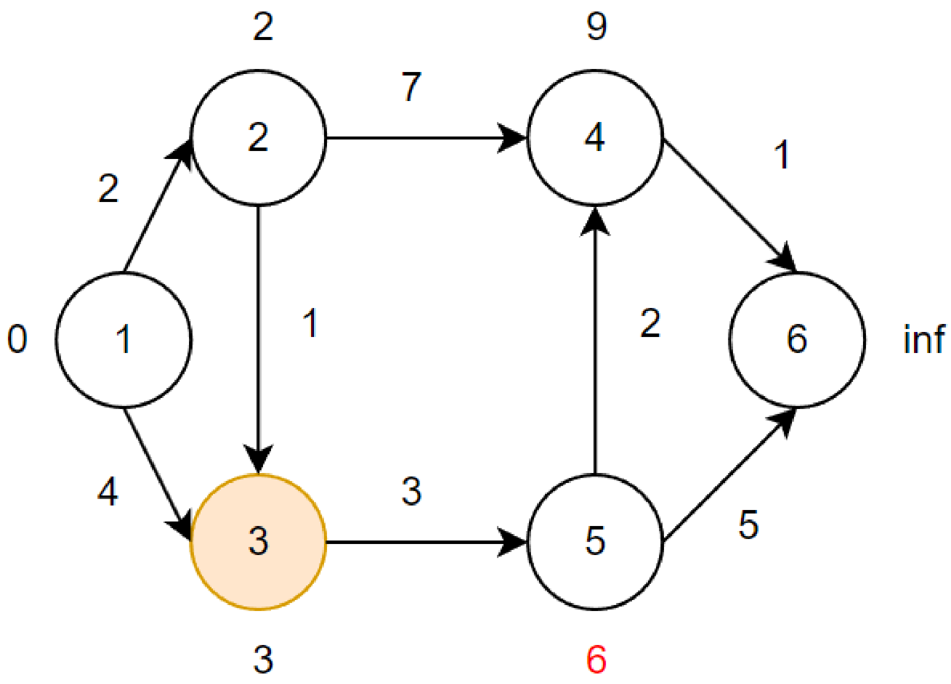
For vertex 2 and vertex 3 since the vertex 2 distance plus the cost edge is 3 which is smaller than the distance vertex 3 which is 4, then will modify the distance vertex 3 to 3.

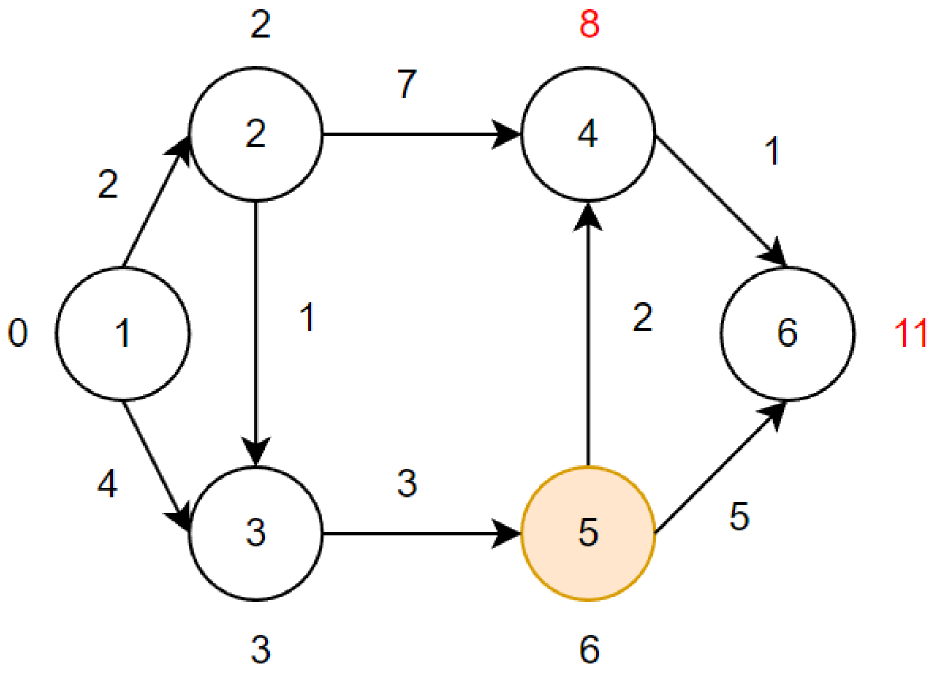
For vertex 2 and vertex 4 since the vertex 2 distance plus the cost edge is 9 which is smaller than the distance vertex 3 which is infinity, then will modify the distance vertex 3 to 9.

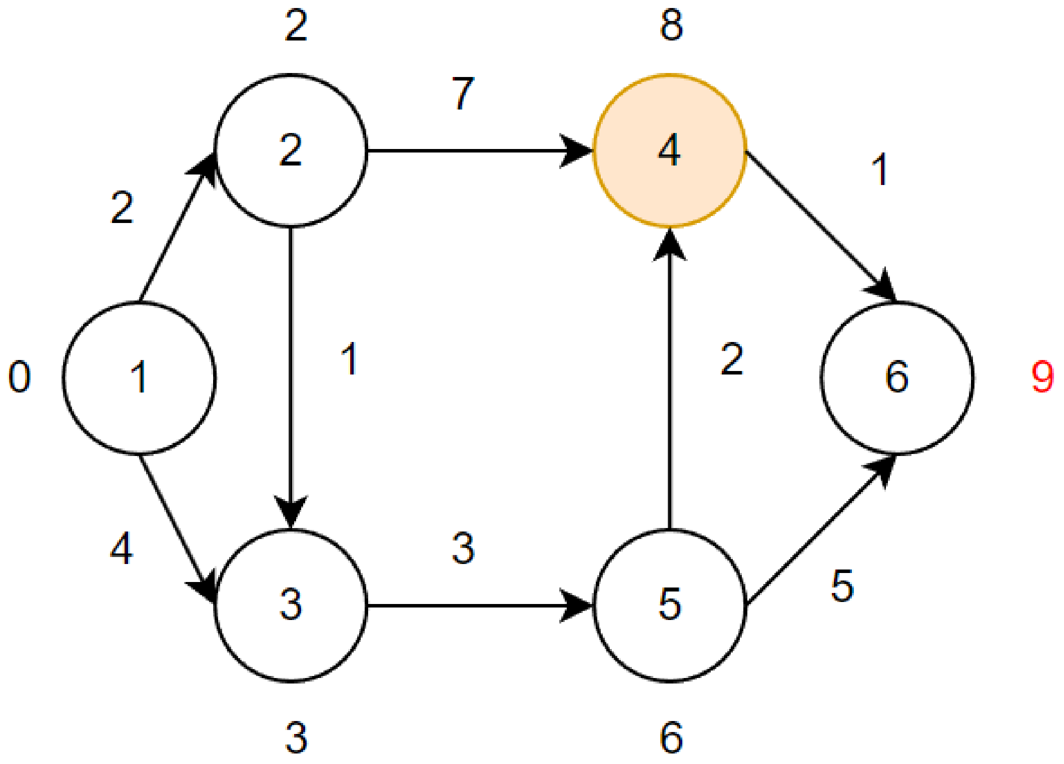


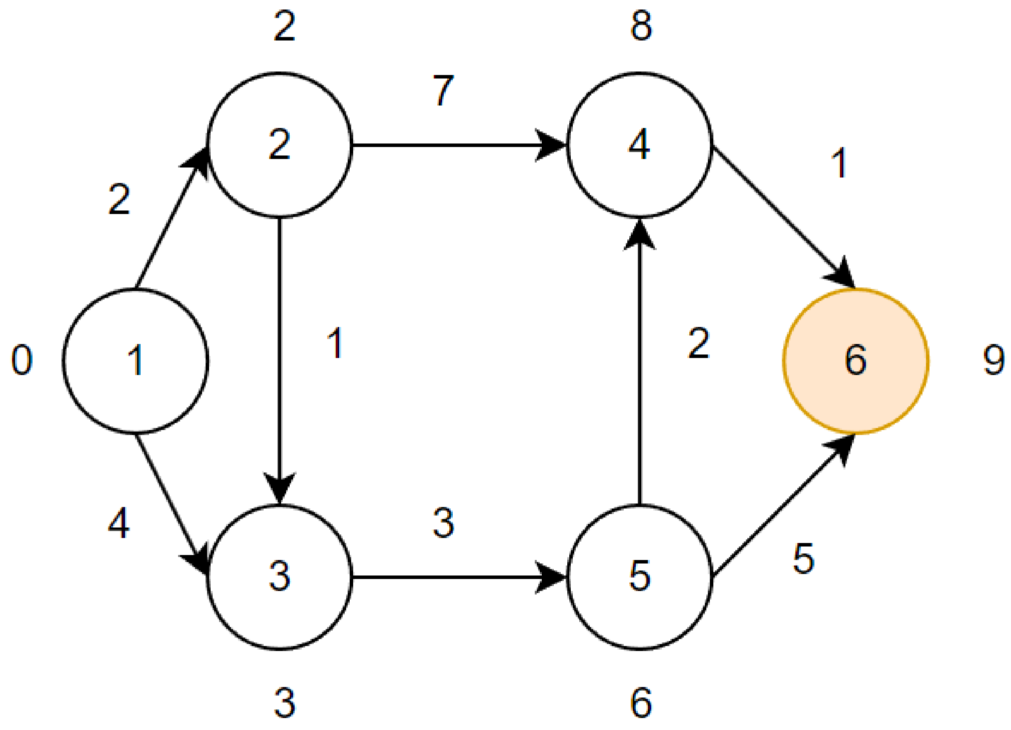
Same steps repeat for other vertices.

1. Select a shortest path
2. Perform relaxation


2
9
7
2
4
2
1
1
0
1
2
6
inf
4
3
3
5
5
3
6


2
8
7
2
4
2
1
1
0
1
2
6
11
4
3
3
5
5
3
6


2
8
7
2
4
1
2
0
1
1
2
6
9
4
3
3
5
5
3
6

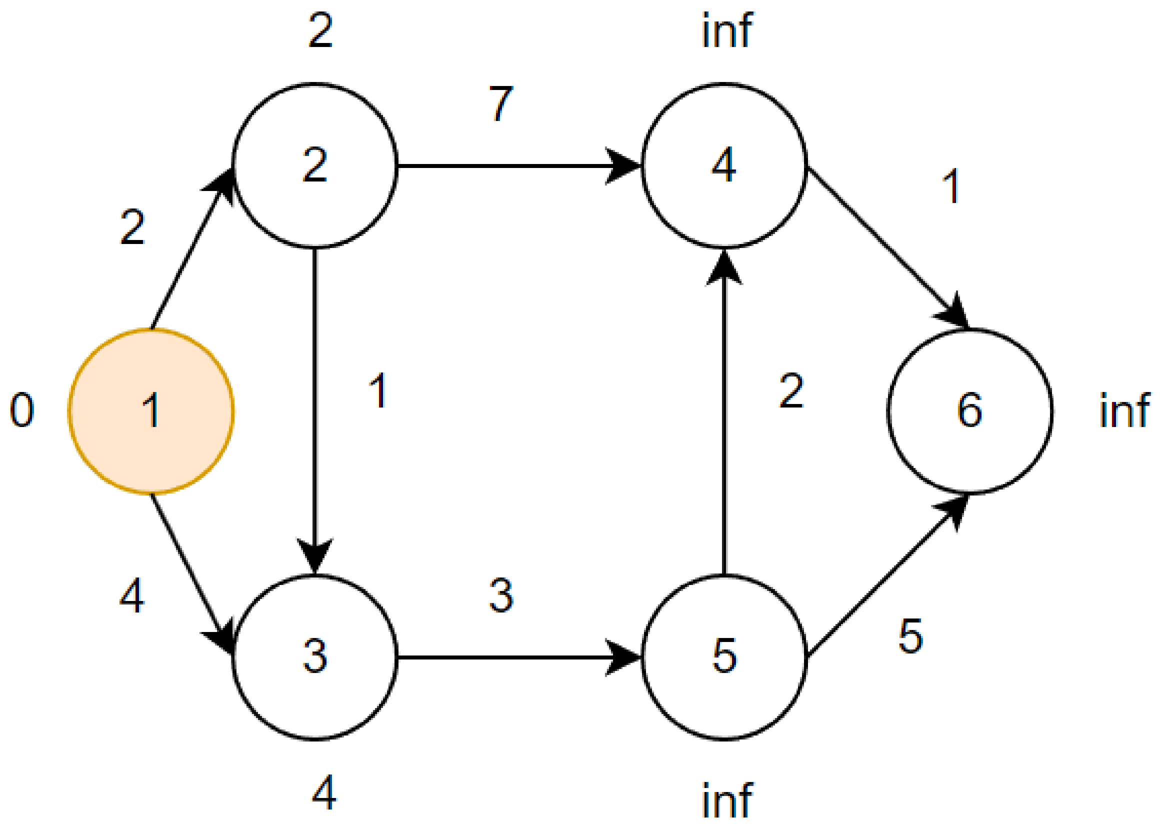

2
8
7
2
4
1
2
0
1
1
2
6
9
4
3
3
5
5
3
6

Now we have found the shortest path starting from vertex 1 to all vertices.

| v | d[v] |
|---|---|
| 2 | 2 |
| 3 | 3 |
| 4 | 8 |
| 5 | 6 |
| 6 | 9 |

The time complexity of the Dijkstra algorithm is $O(n^2)$

The Dijkstra algorithm can also write in the table.



| Selected vertex | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 2 | 2 | 4 | inf | inf | inf |
| 3 | 2 | 3 | 9 | inf | inf |
| 5 | 2 | 3 | 9 | 6 | inf |
| 4 | 2 | 3 | 8 | 6 | 11 |
| 6 | 2 | 3 | 8 | 6 | 9 |
|  | 2 | 3 | 8 | 6 | 9 |

First, we will select the smallest vertex which is vertex 2 and then perform the relaxation. Then we will keep repeating the same step on other vertices.

Pseudocode

Main Function

```
1.  Start
2.  User input number of point
3.   Loop from 0 to number of point
4.  User input point id, coordinate X, coordinate Y
5.  Create new node and add note to the notesList
6.  End Loop
7.  User input number of connection edge
8.  Loop from 0 to number of connection edge
9.  User input node1 id and node2 id
10. End Loop
11. User input the starting point id
12. Set the start point distacneFromStart to 0
13. Call the Dijkstra function
14. User input the destination Id
15. Call the find shortest path function
16. Function return shortest path
17. Output shortest path.
```

```
18. End
```

Dijkstras Function

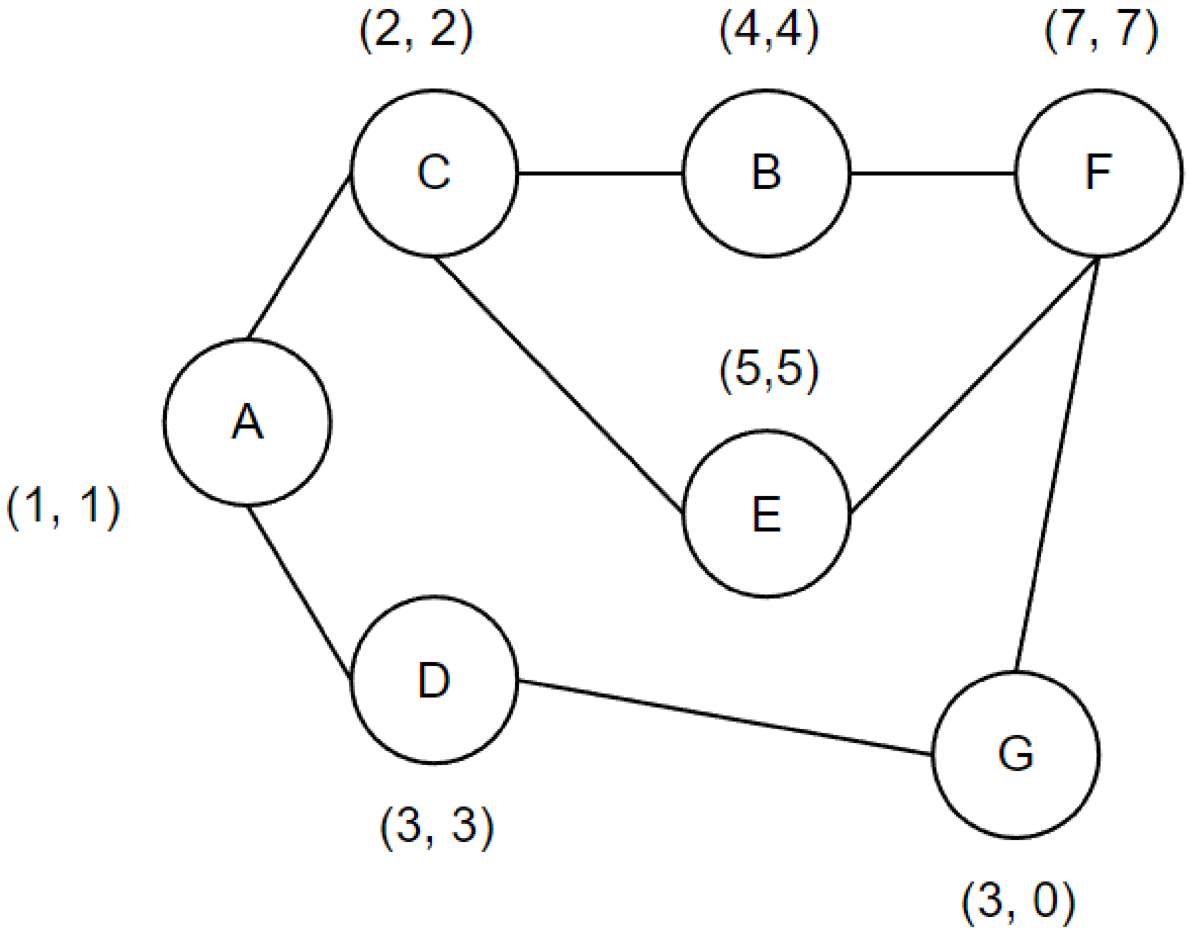
```
1. Start
2. Initial minNode
3. Set the minNode distanceFromStart to infinity
4. Loop from 0 to nodeList size
    a. Loop from 0 to nodeList size
        i. If the node is not visited before, the distanceFromStart is smaller than minNode distanceFromStart
        ii. Set the minNode to current node
        iii. End loop
    b. Loop from 0 to edgesList size
        i. Calculate the distanceBetweenNode between minNode to adjacentNode
        ii. If the distanceBetweenNode is smaller than the adjacentNode distanceFromStart
        iii. Change the adjacentNode distanceFronStart to distanceBetweenNode
5. End
```

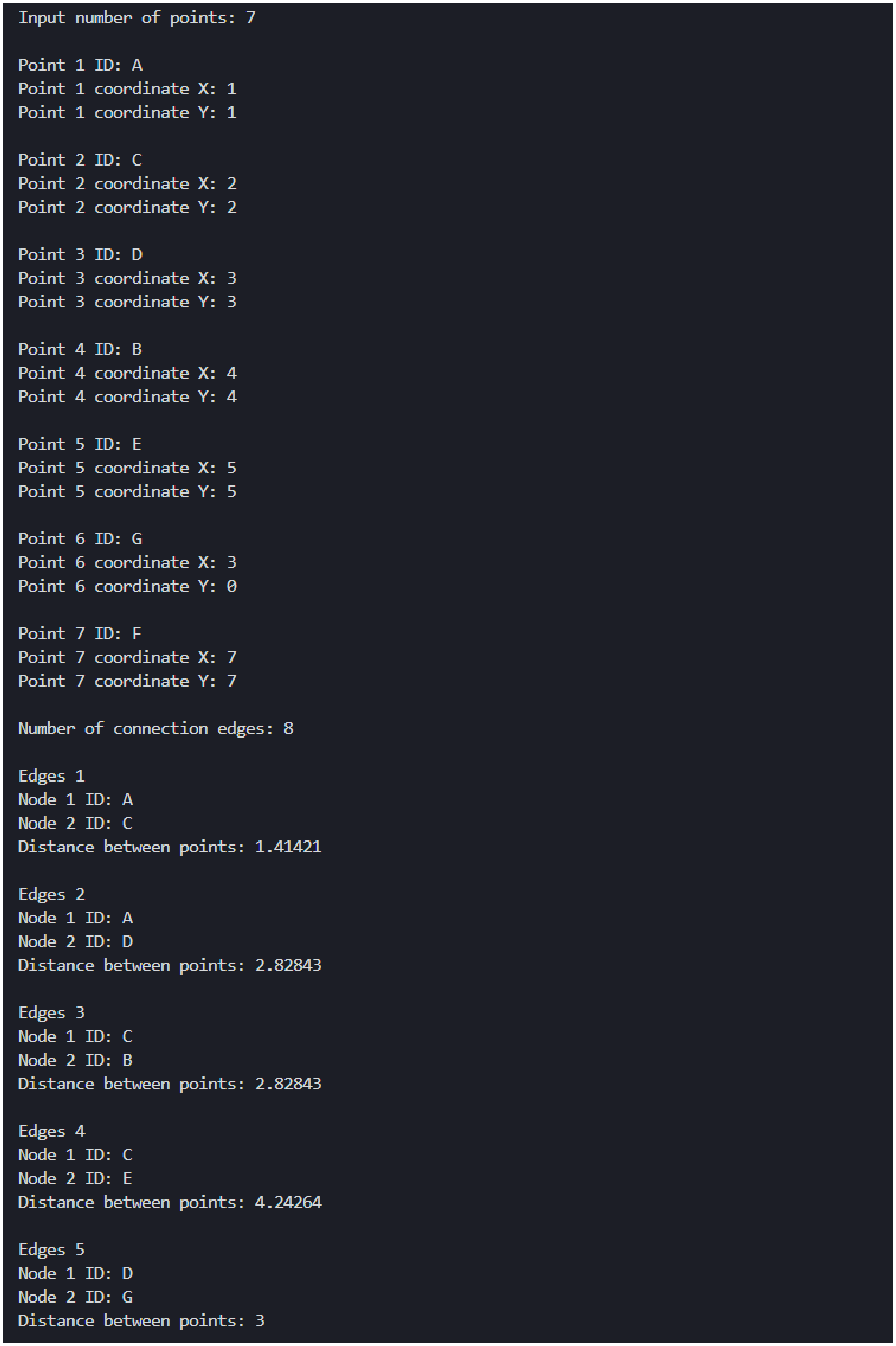
findShortestPath (string destinationId)

```
1. Start
2. Initial previous node
3. Loop from 0 to nodeList size
    a. Find the destination note by using the destination id
    b. End loop
```

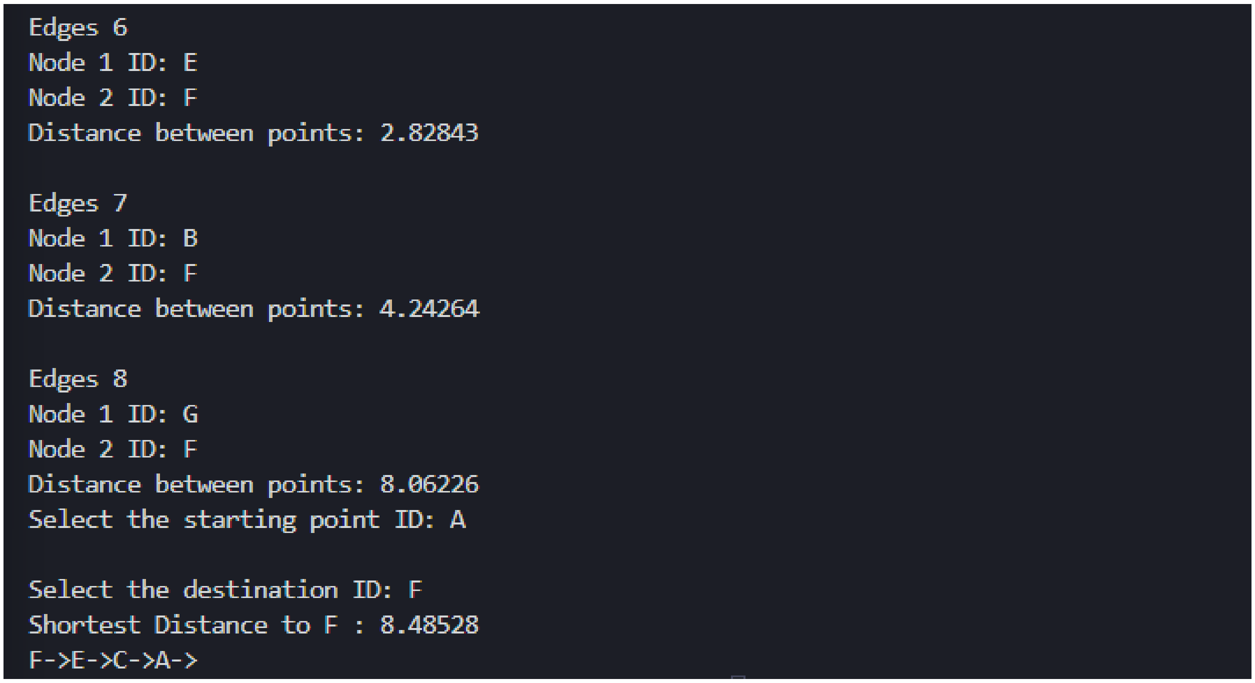
```
4. While loop previous node is not null
    a. Add the shortest path string with the previous node id
    b. Set the previous node to the previous node previous node
5. Return shortestPath
6. End
```

Output example

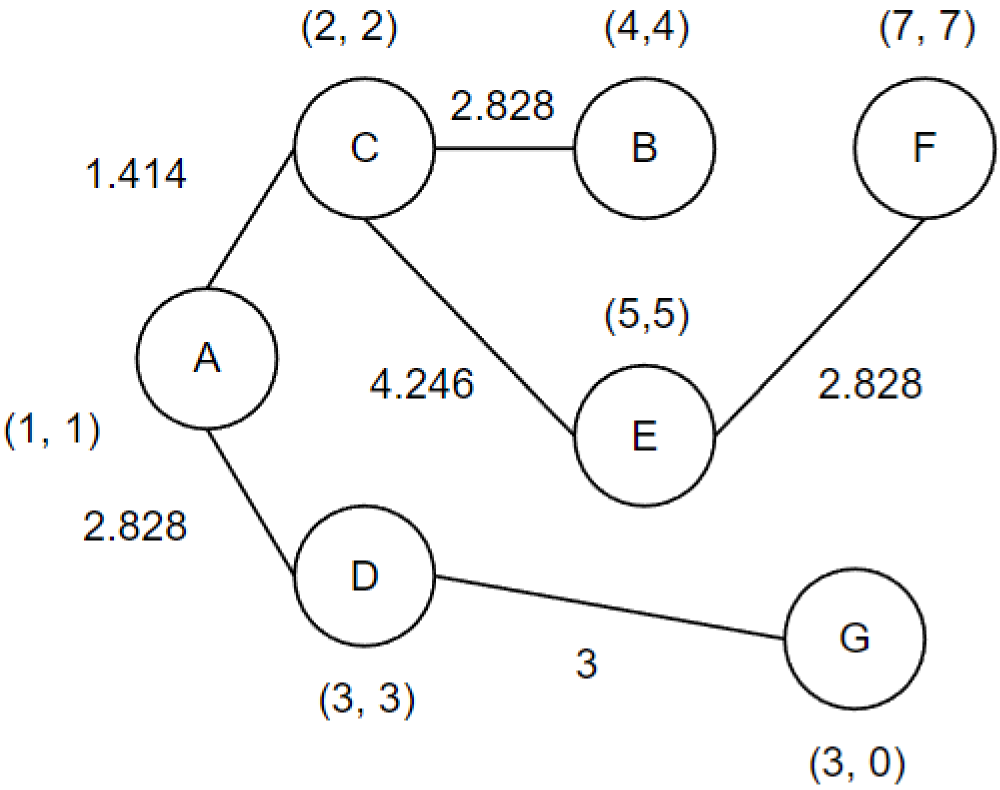




```
Input number of points: 7

Point 1 ID: A
Point 1 coordinate X: 1
Point 1 coordinate Y: 1

Point 2 ID: C
Point 2 coordinate X: 2
Point 2 coordinate Y: 2

Point 3 ID: D
Point 3 coordinate X: 3
Point 3 coordinate Y: 3

Point 4 ID: B
Point 4 coordinate X: 4
Point 4 coordinate Y: 4

Point 5 ID: E
Point 5 coordinate X: 5
Point 5 coordinate Y: 5

Point 6 ID: G
Point 6 coordinate X: 3
Point 6 coordinate Y: 0

Point 7 ID: F
Point 7 coordinate X: 7
Point 7 coordinate Y: 7

Number of connection edges: 8

Edges 1
Node 1 ID: A
Node 2 ID: C
Distance between points: 1.41421

Edges 2
Node 1 ID: A
Node 2 ID: D
Distance between points: 2.82843

Edges 3
Node 1 ID: C
Node 2 ID: B
Distance between points: 2.82843

Edges 4
Node 1 ID: C
Node 2 ID: E
Distance between points: 4.24264

Edges 5
Node 1 ID: D
Node 2 ID: G
Distance between points: 3
```



```
Edges 6
Node 1 ID: E
Node 2 ID: F
Distance between points: 2.82843

Edges 7
Node 1 ID: B
Node 2 ID: F
Distance between points: 4.24264

Edges 8
Node 1 ID: G
Node 2 ID: F
Distance between points: 8.06226
Select the starting point ID: A

Select the destination ID: F
Shortest Distance to F : 8.48528
F->E->C->A->
```



## Dijkstra's Algorithm Applications

1. To find the shortest path
2. In social networking applications
3. In a telephone network
4. To find the locations in the map

## Disadvantage of Dijkstra's algorithm

The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources. Since it is using the greedy method to find the optimal solution it might take a lot of time to perform relaxation on each node. Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path.

## Conclusion

In conclusion, Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph. It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph. In this paper, we assume the path for each node is not direction, the connected nodes can visit each other on the same edge, but in the real application such as the map navigation return road might not be same with the road we go to the destination. Since this program was use on the drone or robot to find the shortest path between two points, the path for the go and return will be the same.

References

1. Programiz (2022, December 10). Dijkstra's Algorithm. https://www.programiz.com/dsa/dijkstra-algorithm
2. GeeksForeeks. (2022, December 10). Dijkstra’s Shortest Path Algorithm | Greedy Algo-7. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
3. Abdul Bari. (2018, February 10). 3.6 Dijkstra Algorithm - Single Source Shortest Path - Greedy Method. https://www.youtube.com/watch?v=XB4MIexjvY0&t=677s&ab_channel=AbdulBari
4. Ravikiran Jaliparthi Venkat, (2014, December 13). PATH FINDING - Dijkstra’s Algorithm.
5.