DAA Experiment-6

(Batch-A/A1)

Name	Ansari Mohammed Shanouf Valijan
UID Number	2021300004
Class	SY B.Tech Computer Engineering(Div-A)
Experiment Number	6
Date of Performance	06-04-23
Date of Submission	09-04-23

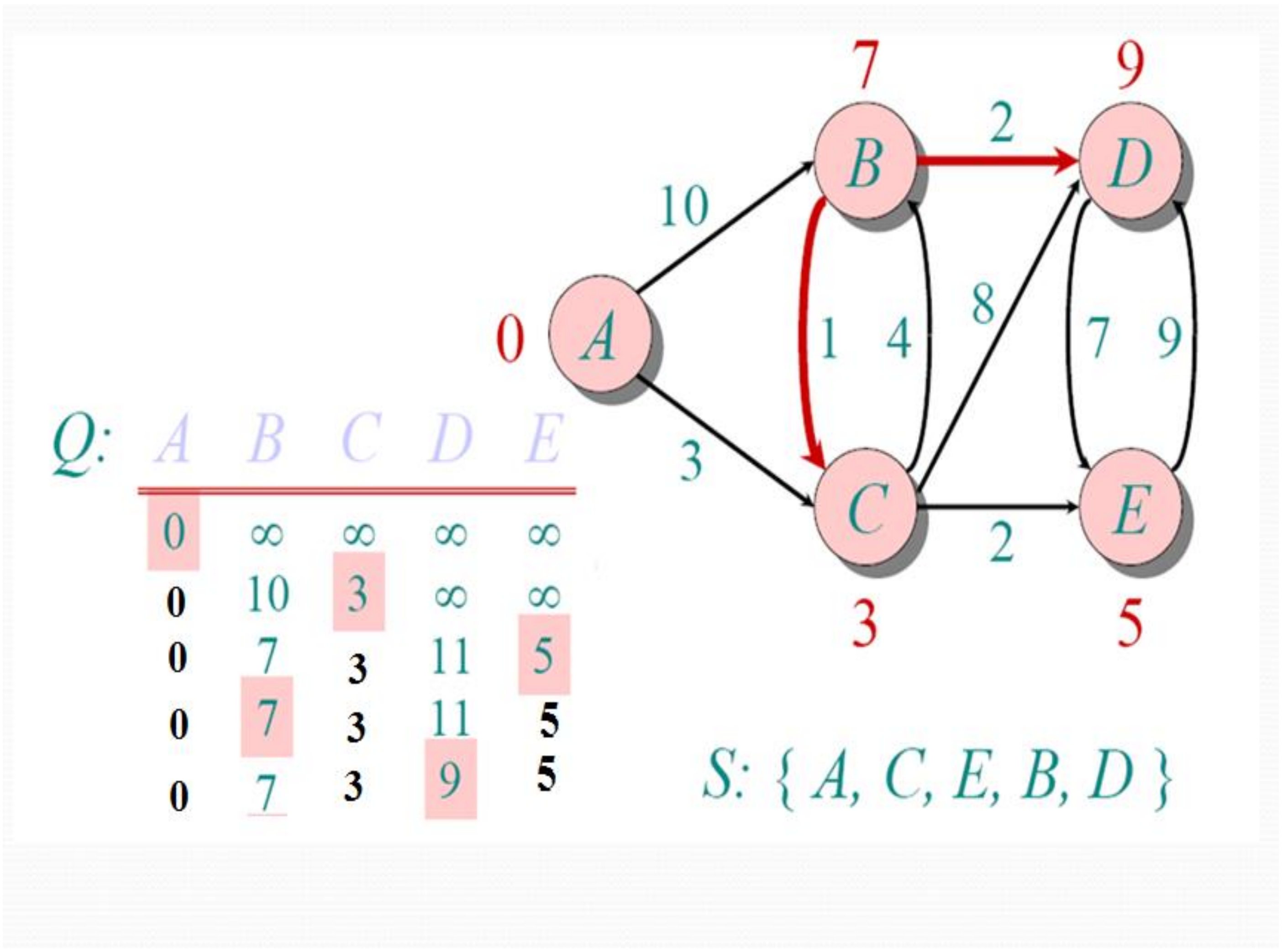
Aim:
To implement Dijkstra’s Algorithm.

Theory:
Dijkstra's algorithm is a popular algorithm used in computer science to find the shortest path between two nodes in a weighted graph. It was proposed by Dutch computer scientist Edsger W. Dijkstra in 1956. The algorithm works by starting at a specified source node and then visiting neighbouring nodes in order of their distance from the source node. It keeps track of the shortest distance from the source node to each visited node and updates this distance whenever a shorter path is found.

To implement Dijkstra's algorithm, we start by creating a priority queue and adding the source node to it with a distance of 0. We then repeat the following steps until the priority queue is empty:

I. Remove the node with the smallest distance from the priority queue.
II. For each neighbour of the current node, calculate the distance from the source node to that neighbour by adding the weight of the edge connecting them to the distance of the current node. If this distance is shorter than the previous distance to the neighbour, update the distance.
III. Add the neighbour to the priority queue if it is not already in it.

When the algorithm finishes, the shortest path from the source node to every other node in the graph will have been found. Dijkstra's algorithm is guaranteed to find the shortest path as long as the graph has no negative weight edges.



Q: A	B	C	D	E
0	∞	∞	∞	∞
0	10	3	∞	∞
0	7	3	11	5
0	7	3	11	5
0	7	3	9	5

S: { A, C, E, B, D }

Shown above is an example of a directed graph where Dijkstra's algorithm is being used to find the shortest distances from source node A to all other nodes. Here, the distance of the source node is considered as 0 while the same for all other nodes is considered to be infinity. These distances are updated as the algorithm progresses, as visible in the table alongside the graph.

Algorithm:

[A] For Dijkstra's algorithm-

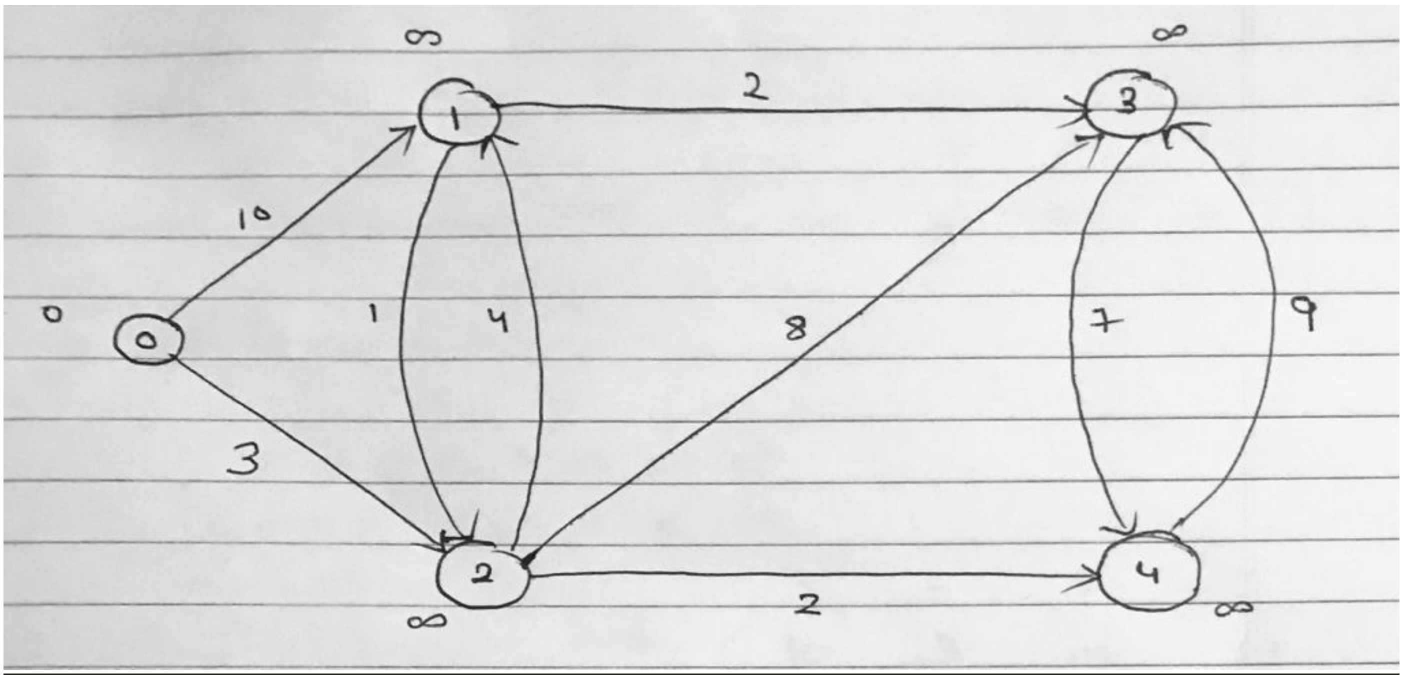
I. Start.
II. Enqueue the source node in priority queue.
III. Perform the following steps while the queue is not empty.
IV. Dequeue a node from the priority queue.
V. For each of its neighbours, update the distances if smaller than current one.
VI. If neighbour is not visited, enqueue it.
VII. End.

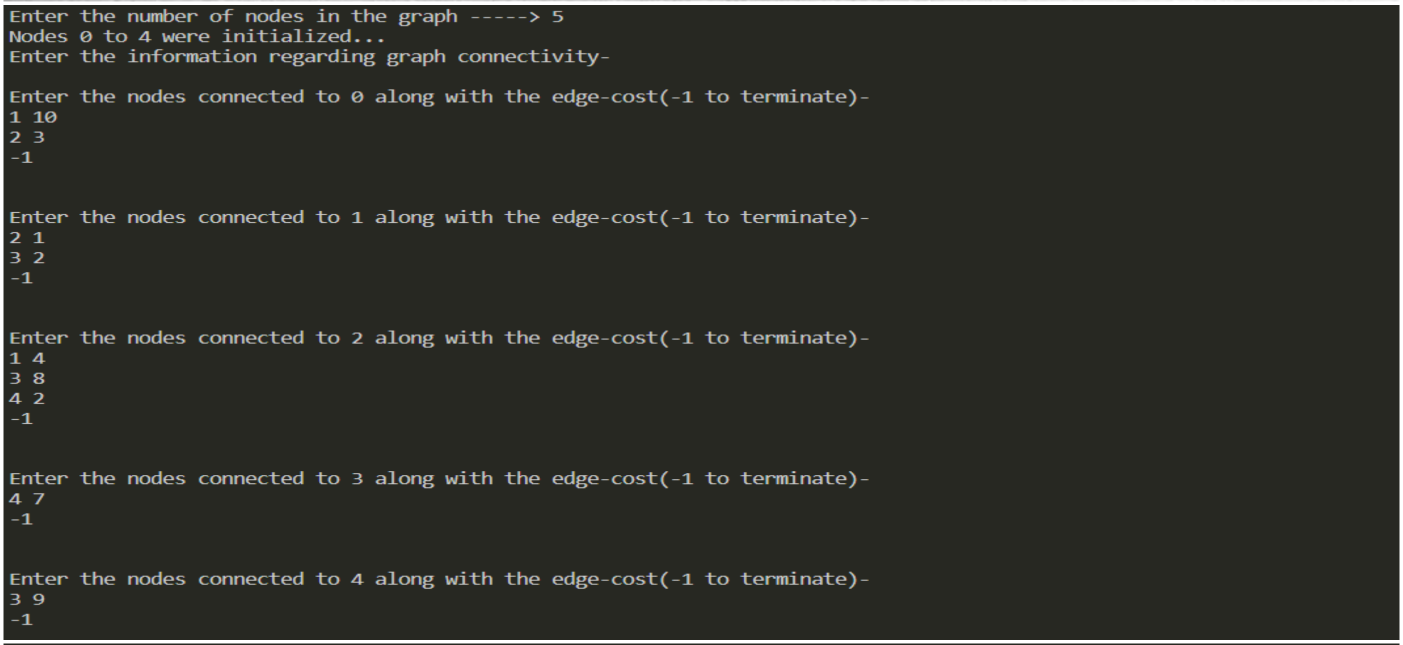
Program:

```c
//header files
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

//graph handling
int** graphData;
int* visited;
int* distances;
int* prevNode;
int numOfNodes;

//queue handling
int* nodesQueue;
int front=-1, rear=-1;

void enqueue(int elem){
    rear++;
    nodesQueue[rear]=elem;
}

int dequeue(){
    front++;
    return nodesQueue[front];
}

void sortQueue(){
    int temp;
    for(int i=front+1; i<rear; i++){
        if(distances[nodesQueue[i]]>distances[nodesQueue[i+1]]){
            temp=nodesQueue[i];
            nodesQueue[i]=nodesQueue[i+1];
            nodesQueue[i+1]=temp;
            i=front;
        }
    }
}

//dijkstra's algorithm
void dijkstras(int sourceNode){
    int currNode, tempDistance;
    enqueue(sourceNode);
    printf("Table showing the updated distances as the new nodes are getting visited-
\n\n");
    printf("Node\t");
    for(int i=0; i<numOfNodes; i++){
        if(i!=sourceNode) printf("%d\t",i);
    }
    printf("\n-\t");
    for(int i=0; i<numOfNodes-1; i++){
        printf("%c\t",236);
    }

    while(front!=rear){
        currNode=dequeue();
        for(int i=0; i<numOfNodes; i++){
            if(graphData[currNode][i]!=-1){
                tempDistance=distances[currNode]+graphData[currNode][i];
                if(distances[i]==-1||tempDistance<distances[i]){
                    distances[i]=tempDistance;
                    prevNode[i]=currNode;
                }
                if(visited[i]==0){
                enqueue(i);
                visited[i]=1;
                }
            }
        }

        sortQueue();

        printf("\n%d\t",currNode);
        for(int i=0; i<numOfNodes; i++){
            if(i!=sourceNode){
                if(distances[i]==-1){
                    printf("%c\t",236);
                }
                else{
                    printf("%d\t",distances[i]);
                }
            }
        }
    }
}

//function for printing path with minimum cost
void printPath(int targetNode, int sourceNode){
    if(targetNode!=sourceNode){
        printPath(prevNode[targetNode], sourceNode);
        printf(" -> %d",targetNode);
    }
    else{
        printf("%d",sourceNode);
    }
}

//main function
void main(){
    int node,distance,sourceNode;

    printf("\nEnter the number of nodes in the graph -----> ");
    scanf("%d",&numOfNodes);

    graphData=(int**)malloc(numOfNodes*sizeof(int*));
    visited=(int*)malloc(numOfNodes*sizeof(int));
    distances=(int*)malloc(numOfNodes*sizeof(int));
    prevNode=(int*)malloc(numOfNodes*sizeof(int));
    nodesQueue=(int*)malloc(numOfNodes*sizeof(int));
    for(int i=0; i<numOfNodes; i++){
        visited[i]=0;
        distances[i]=-1;
        graphData[i]=(int*)malloc(numOfNodes*sizeof(int));
        for(int j=0; j<numOfNodes; j++){
            graphData[i][j]=-1;
        }
    }

    printf("Nodes 0 to %d were initialized...\n",numOfNodes-1);

    printf("Enter the information regarding graph connectivity-");
    for(int i=0; i<numOfNodes; i++){
        printf("\n\nEnter the nodes connected to %d along with the edge-cost(-1 to
terminate)-\n",i);
        while(true){
            scanf("%d",&node);
            if(node==-1){
                break;
            }
            scanf("%d",&distance);
            graphData[i][node]=distance;
        }
    }

    printf("\nEnter the source node -----> ");
    scanf("%d",&sourceNode);
    distances[sourceNode]=0;
    visited[sourceNode]=1;

    printf("\nFollowing information was obtained from Dijkstra's algorithm-\n\n");
    dijkstras(sourceNode);

    printf("\n\nFollowing are the shortest paths obtained-");
    for(int i=0; i<numOfNodes; i++){
        if(i!=sourceNode){
            printf("\n\nFrom Node-%d to Node-%d(Cost: %d)\n",sourceNode,i,distances[i]);
            printPath(i,sourceNode);
        }
    }
    printf("\n\n");

    free(visited);
    free(distances);
    free(prevNode);
    free(nodesQueue);
    for(int i=0; i<numOfNodes; i++){
        free(graphData[i]);
    }
    free(graphData);

}
```

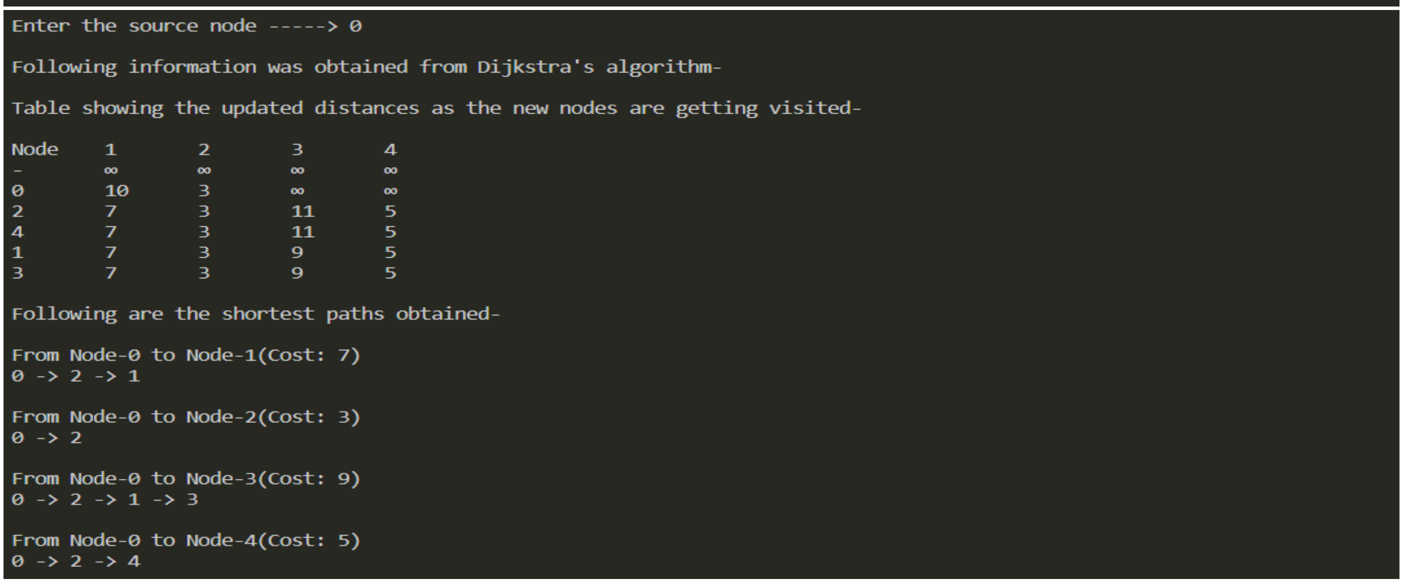
Implementation:

The written program was tested for the following two graphs-





```
Enter the number of nodes in the graph -----> 5
Nodes 0 to 4 were initialized...
Enter the information regarding graph connectivity-

Enter the nodes connected to 0 along with the edge-cost(-1 to terminate)-
1 10
2 3
-1


Enter the nodes connected to 1 along with the edge-cost(-1 to terminate)-
2 1
3 2
-1


Enter the nodes connected to 2 along with the edge-cost(-1 to terminate)-
1 4
3 8
4 2
-1


Enter the nodes connected to 3 along with the edge-cost(-1 to terminate)-
4 7
-1


Enter the nodes connected to 4 along with the edge-cost(-1 to terminate)-
3 9
-1
```



```
Enter the source node -----> 0

Following information was obtained from Dijkstra's algorithm-

Table showing the updated distances as the new nodes are getting visited-

Node    1       2       3       4
-       ∞       ∞       ∞       ∞
0       10      3       ∞       ∞
2       7       3       11      5
4       7       3       11      5
1       7       3       9       5
3       7       3       9       5

Following are the shortest paths obtained-

From Node-0 to Node-1(Cost: 7)
0 -> 2 -> 1

From Node-0 to Node-2(Cost: 3)
0 -> 2

From Node-0 to Node-3(Cost: 9)
0 -> 2 -> 1 -> 3

From Node-0 to Node-4(Cost: 5)
0 -> 2 -> 4
```

Example - I

Currant distances $\Rightarrow \begin{array}{cccc} 1 & 2 & 3 & 4 \\ (\infty & \infty & \infty & \infty) \end{array}$

Current Node : 0

$d(0,1) = 0 + 10 = 10 < \infty$
$d(0,2) = 0 + 3 = 3 < \infty$

$\therefore$ Updated distances $\Rightarrow \begin{array}{cccc} 1 & 2 & 3 & 4 \\ (10 & 3 & \infty & \infty) \end{array}$

Current Node : 2

$d(2,1) = 3 + 4 = 7 < 10$
$d(2,3) = 3 + 8 = 11 < \infty$
$d(2,4) = 3 + 2 = 5 < \infty$

$\therefore$ Updated distances $\Rightarrow \begin{array}{cccc} 1 & 2 & 3 & 4 \\ (7 & 3 & 11 & 5) \end{array}$

Current Node : 4

$d(4,3) = 5 + 9 = 14 > 11$ ... hence won't be updated

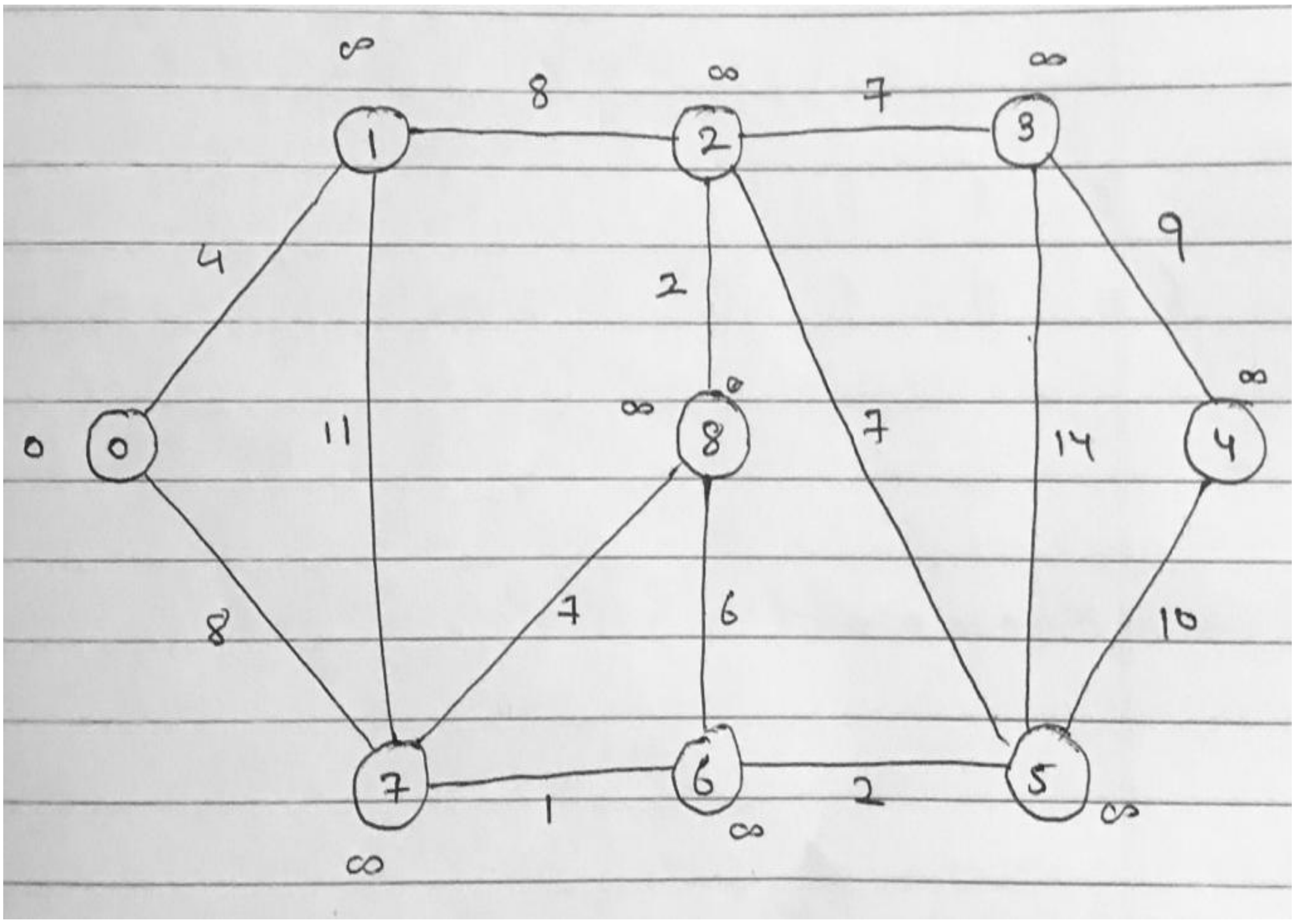
Current Node : 1

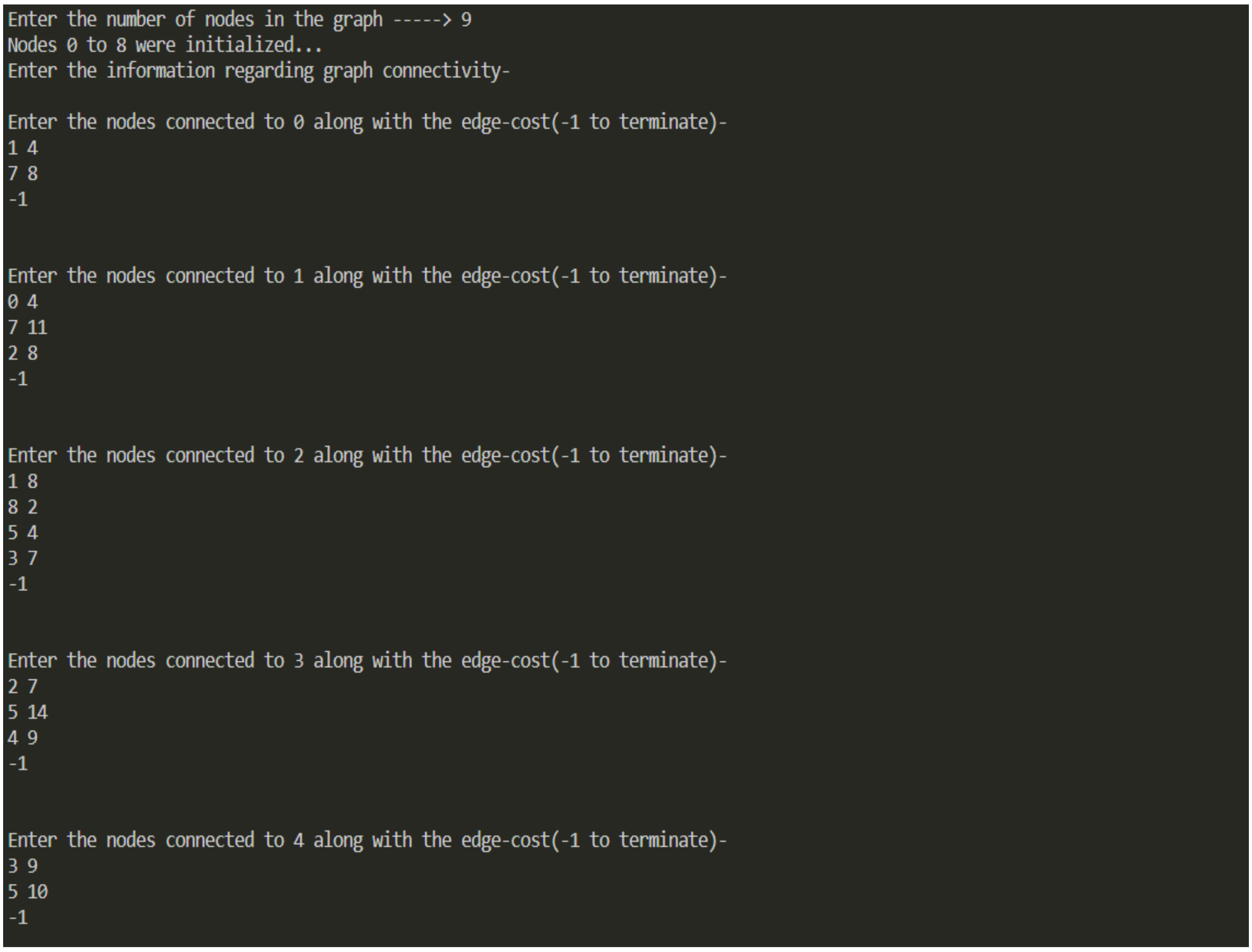
$d(1,2) = 7 + 1 = 8 > 3$ ... wont be updated.
$d(1,3) = 7 + 2 = 9 < 11$

$\therefore$ Updated distances $\Rightarrow \begin{array}{cccc} 1 & 2 & 3 & 4 \\ (7 & 3 & 9 & 5) \end{array}$

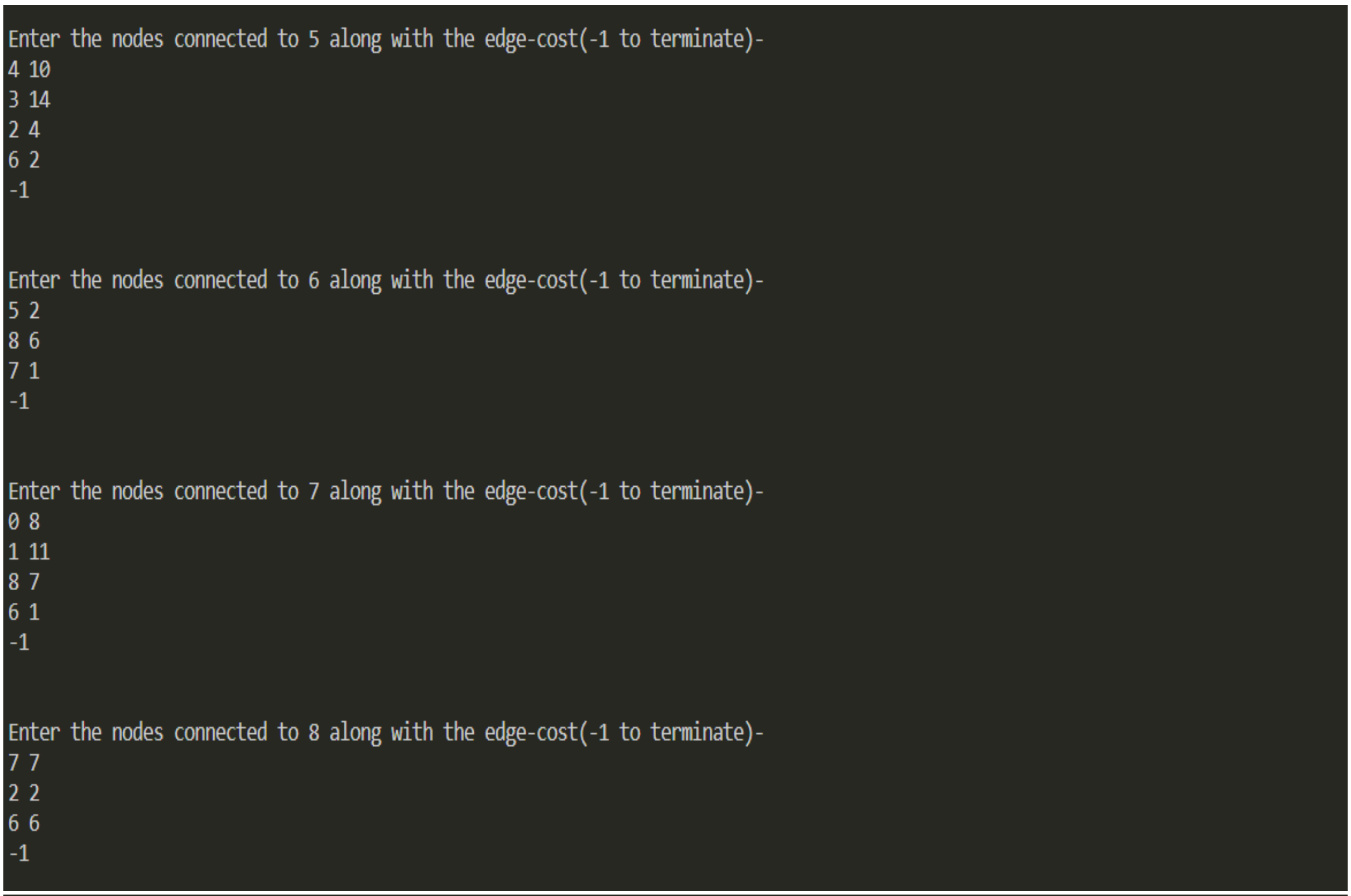
Hence we get the minimum distance from 0 to other nodes

Teacher's Signature:....................

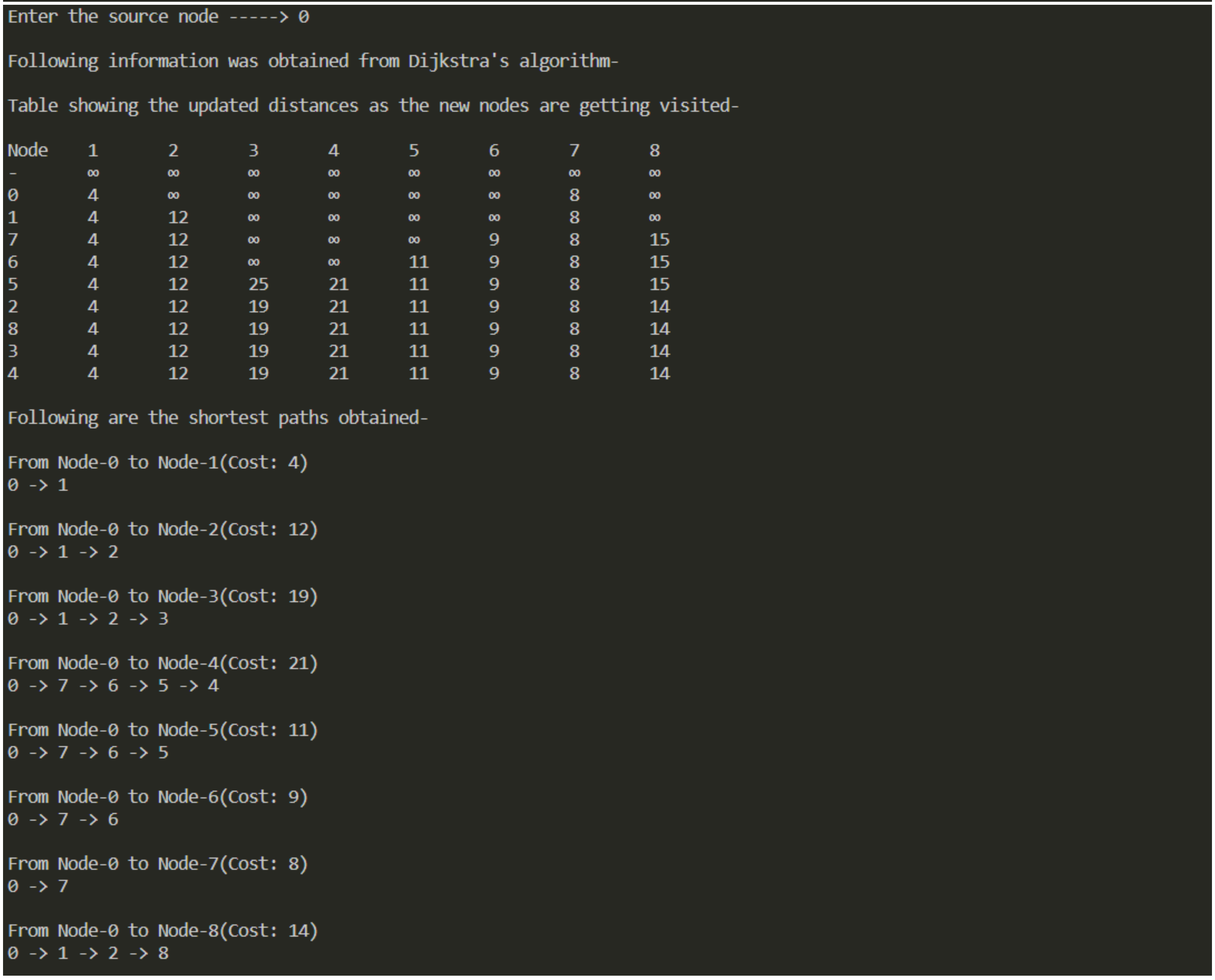




```
Enter the number of nodes in the graph -----> 9
Nodes 0 to 8 were initialized...
Enter the information regarding graph connectivity-

Enter the nodes connected to 0 along with the edge-cost(-1 to terminate)-
1 4
7 8
-1


Enter the nodes connected to 1 along with the edge-cost(-1 to terminate)-
0 4
7 11
2 8
-1


Enter the nodes connected to 2 along with the edge-cost(-1 to terminate)-
1 8
8 2
5 4
3 7
-1


Enter the nodes connected to 3 along with the edge-cost(-1 to terminate)-
2 7
5 14
4 9
-1


Enter the nodes connected to 4 along with the edge-cost(-1 to terminate)-
3 9
5 10
-1
```



```
Enter the nodes connected to 5 along with the edge-cost(-1 to terminate)-
4 10
3 14
2 4
6 2
-1


Enter the nodes connected to 6 along with the edge-cost(-1 to terminate)-
5 2
8 6
7 1
-1


Enter the nodes connected to 7 along with the edge-cost(-1 to terminate)-
0 8
1 11
8 7
6 1
-1


Enter the nodes connected to 8 along with the edge-cost(-1 to terminate)-
7 7
2 2
6 6
-1
```



```
Enter the source node -----> 0

Following information was obtained from Dijkstra's algorithm-

Table showing the updated distances as the new nodes are getting visited-

Node    1       2       3       4       5       6       7       8
-       ∞       ∞       ∞       ∞       ∞       ∞       ∞       ∞
0       4       ∞       ∞       ∞       ∞       ∞       8       ∞
1       4       12      ∞       ∞       ∞       ∞       8       ∞
7       4       12      ∞       ∞       ∞       9       8       15
6       4       12      ∞       ∞       11      9       8       15
5       4       12      25      21      11      9       8       15
2       4       12      19      21      11      9       8       14
8       4       12      19      21      11      9       8       14
3       4       12      19      21      11      9       8       14
4       4       12      19      21      11      9       8       14

Following are the shortest paths obtained-

From Node-0 to Node-1(Cost: 4)
0 -> 1

From Node-0 to Node-2(Cost: 12)
0 -> 1 -> 2

From Node-0 to Node-3(Cost: 19)
0 -> 1 -> 2 -> 3

From Node-0 to Node-4(Cost: 21)
0 -> 7 -> 6 -> 5 -> 4

From Node-0 to Node-5(Cost: 11)
0 -> 7 -> 6 -> 5

From Node-0 to Node-6(Cost: 9)
0 -> 7 -> 6

From Node-0 to Node-7(Cost: 8)
0 -> 7

From Node-0 to Node-8(Cost: 14)
0 -> 1 -> 2 -> 8
```

Example - II

$$\begin{array}{lcccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \text{Current distances} \Rightarrow (& \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty) \end{array}$$

Current Node : 0

$\partial(0,1) = 0+4 = 4 < \infty$; $\partial(0,7) = 0+8 = 8 < \infty$

Current distances $\Rightarrow$ (4 ∞ ∞ ∞ ∞ ∞ 8 ∞)

Current Node : 1

$\partial(1,2) = 4+8 = 12 < \infty$; $\partial(1,7) = 4+11 = 15 > 8$

Current distances $\Rightarrow$ (4 12 ∞ ∞ ∞ ∞ 8 ∞)

Current Node : 7

$\partial(7,8) = 8+7 = 15 < \infty$; $\partial(7,6) = 8+1 = 9 < \infty$

Current distances $\Rightarrow$ (4 12 ∞ ∞ ∞ 9 8 15)

Current Node : 6

$\partial(6,5) = 9+2 = 11 < \infty$

Current distances $\Rightarrow$ (4 12 ∞ ∞ 11 9 8 15)

Current Node : 5

$\partial(5,3) = 11+14 = 25 < \infty$; $\partial(5,4) = 11+10 = 21 < \infty$

Current distances $\Rightarrow$ (4 12 25 21 11 9 8 15)

Current Node : 2

$\partial(2,3) = 12+7 = 19 < 25$ ... will be updated

$\partial(2,8) = 12+2 = 14 < 15$ ... ↗

$\therefore$ Current distances $\Rightarrow$ (4 12 19 21 11 9 8 14)

Visiting further nodes 8, 3, 4 makes no change

$$\begin{array}{lcccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \text{Hence we get } \underset{u)}{\text{min}} \text{ distances} : (& 4 & 12 & 19 & 21 & 11 & 9 & 8 & 14 \end{array}$$

Teacher's Signature:..................

Inference:

On observing the outputs and solving the used examples, I realized that Dijkstra's algorithm works in steps. Initially, it considers that infinite cost will be required to go from source node to destination node. Further, at each step, it goes to the node with minimum distance on it, then it updates the distances of the neighbouring nodes based on that distance if required. It repeats that until all the nodes have been visited. While trying to code Dijkstra's algorithm, I was also able to come to the realization that in order to print the shortest path, one can create an array that stores the nodes which aided in reducing the distances of the corresponding nodes. Using this information coupled with recursion, all the shortest paths can be printed.

Conclusion:

By performing this experiment, I was able to understand Dijkstra's algorithm. I was also able to implement the same and print all the shortest paths obtained from source node to destination nodes.