

Computer Networks

Name : M. Sai Saranya

Regno: 22BAI1471

Course Title : Computer Networks

Course code : BCSE308P

Slot : L45-46

Faculty : Dr Neelananarayanan V

S.No	Experiment Name	Date	Page No.	Marks
1.	Basic Network Configuration Commands	10-01-2024		
2.	Client-Server Application Echo	17-01-2024		
3.	IP Address Validation and Simple application of ATM using TCP	24-01-2024		
4.	CRC code generator using socket programming	07-02-2024		
5. a)	Echo programming using UDP	21-02-2024		
5. b)	IP address validation using UDP	21-02-2024		

S.No	Experiment Name	Date	Page No.	Marks
5. c)	ATM simulation using UDP	21-02-2024		
6.	Stop and wait ARQ	28-02-2024		
7.	Sliding window protocol	13-03-2024		
8.	Bellman-Ford Algorithm	20-03-2024		

Experiment No. 8

Experiment Name: Bellman-Ford Algorithm using c programming

Date: 20-3-2024

Problem Statement

To understand the given graph and apply the bellman ford algorithm using c programming

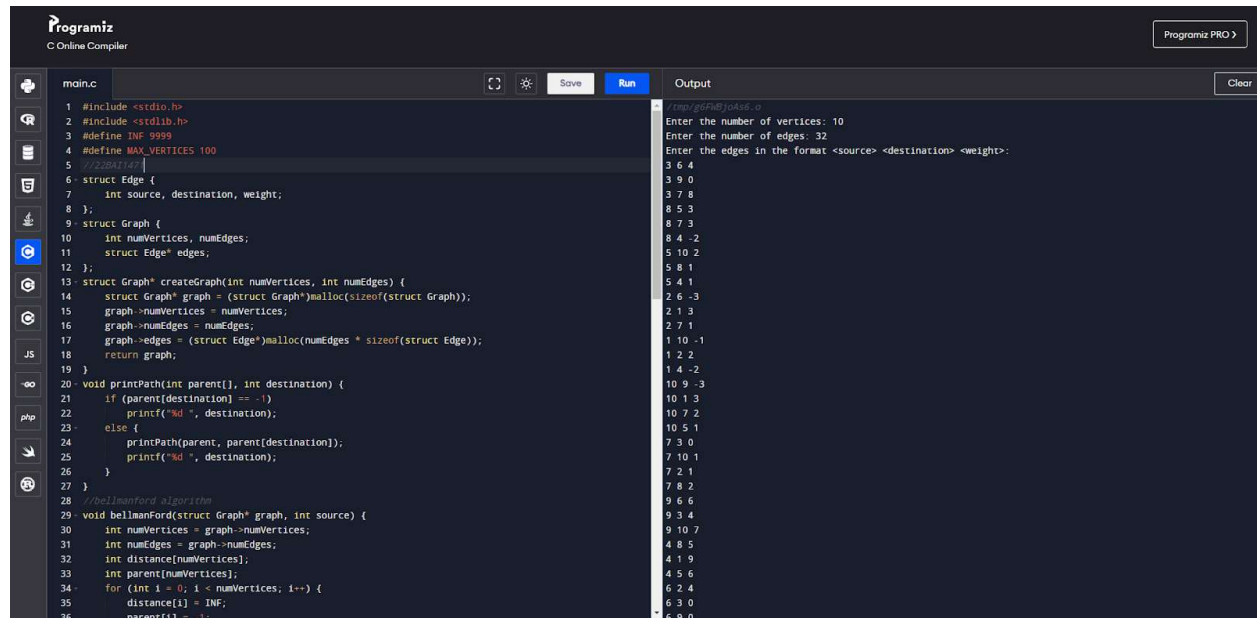
Algorithm

Dijkstra algorithm and Prim's algorithm won't allow negative values in the cycle.

Bellman Ford algorithm works by overestimating the length of the path from the starting vertex to all other vertices. After that relaxes those estimates by finding new paths that are

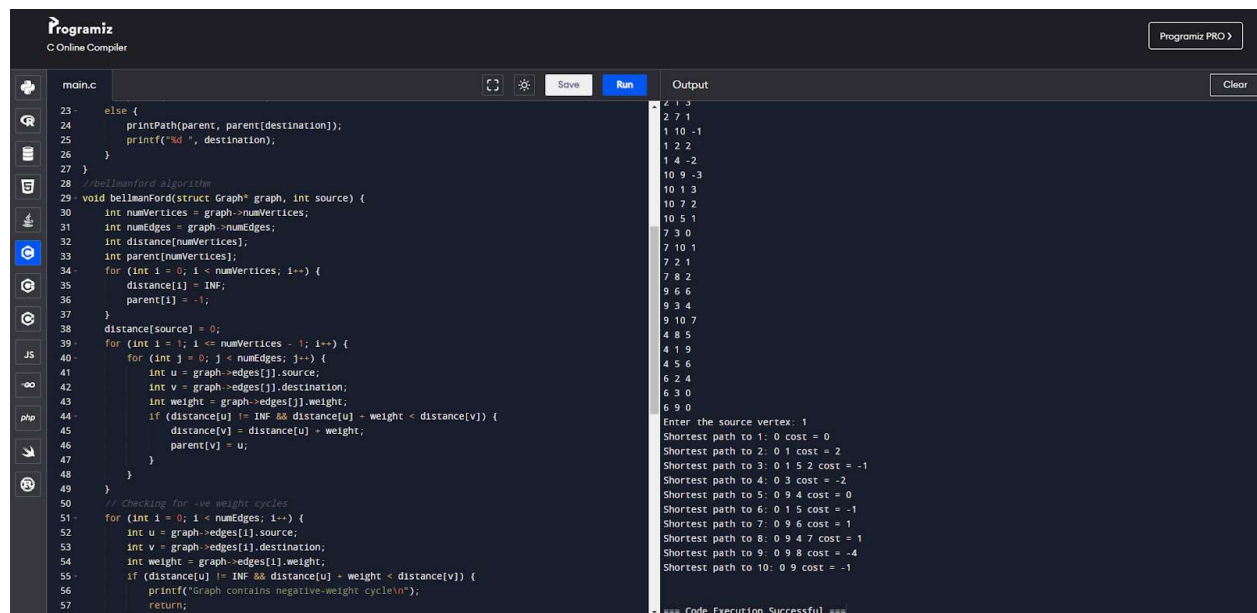
shorter than the previously overestimated paths. This way it finds all the shortest distances from source vertex to destination. Here vertices are 10 and number of edges given are 32 .

Means 32 entries in the terminal. But by using the traditional method vertices are counted from 0 to 9 (0 to n-1) accordingly output is being printed.



```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define INF 9999
4 #define MAX_VERTICES 100
5 //BELLMANFORD
6 struct Edge {
7     int source, destination, weight;
8 };
9 struct Graph {
10     int numVertices, numEdges;
11     struct Edge* edges;
12 };
13 struct Graph* createGraph(int numVertices, int numEdges) {
14     struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
15     graph->numVertices = numVertices;
16     graph->numEdges = numEdges;
17     graph->edges = (struct Edge*)malloc(numEdges * sizeof(struct Edge));
18     return graph;
19 }
20 void printPath(int parent[], int destination) {
21     if (parent[destination] == -1)
22         printf("%d ", destination);
23     else {
24         printPath(parent, parent[destination]);
25         printf("%d ", destination);
26     }
27 }
28 //bellmanford algorithm
29 void bellmanFord(struct Graph* graph, int source) {
30     int numVertices = graph->numVertices;
31     int numEdges = graph->numEdges;
32     int distance[numVertices];
33     int parent[numVertices];
34     for (int i = 0; i < numVertices; i++) {
35         distance[i] = INF;
36         parent[i] = -1;
37     }
38     distance[source] = 0;
39     for (int i = 1; i <= numVertices - 1; i++) {
40         for (int j = 0; j < numEdges; j++) {
41             int u = graph->edges[j].source;
42             int v = graph->edges[j].destination;
43             int weight = graph->edges[j].weight;
44             if (distance[u] != INF && distance[u] + weight < distance[v]) {
45                 distance[v] = distance[u] + weight;
46                 parent[v] = u;
47             }
48         }
49     }
50     // Checking for -ve weight cycles
51     for (int i = 0; i < numEdges; i++) {
52         int u = graph->edges[i].source;
53         int v = graph->edges[i].destination;
54         int weight = graph->edges[i].weight;
55         if (distance[u] != INF && distance[u] + weight < distance[v]) {
56             printf("Graph contains negative-weight cycle\n");
57             return;
58         }
59     }
60 }
```

```
Output
Enter the number of vertices: 10
Enter the number of edges: 32
Enter the edges in the format <source> <destination> <weight>:
3 6 4
3 9 0
3 7 8
8 5 3
8 7 3
8 4 -2
5 10 2
5 8 1
5 4 1
2 6 -3
2 1 3
2 7 1
1 10 -1
1 2 2
1 4 -2
10 9 -3
10 1 3
10 7 2
10 5 1
7 3 0
7 10 1
7 2 1
7 8 2
9 6 6
9 3 4
9 10 7
4 8 5
4 1 9
4 5 6
6 2 4
6 3 0
6 0 0
```



```
main.c
23 else {
24     printPath(parent, parent[destination]);
25     printf("%d ", destination);
26 }
27 }
28 //bellmanford algorithm
29 void bellmanFord(struct Graph* graph, int source) {
30     int numVertices = graph->numVertices;
31     int numEdges = graph->numEdges;
32     int distance[numVertices];
33     int parent[numVertices];
34     for (int i = 0; i < numVertices; i++) {
35         distance[i] = INF;
36         parent[i] = -1;
37     }
38     distance[source] = 0;
39     for (int i = 1; i <= numVertices - 1; i++) {
40         for (int j = 0; j < numEdges; j++) {
41             int u = graph->edges[j].source;
42             int v = graph->edges[j].destination;
43             int weight = graph->edges[j].weight;
44             if (distance[u] != INF && distance[u] + weight < distance[v]) {
45                 distance[v] = distance[u] + weight;
46                 parent[v] = u;
47             }
48         }
49     }
50     // Checking for -ve weight cycles
51     for (int i = 0; i < numEdges; i++) {
52         int u = graph->edges[i].source;
53         int v = graph->edges[i].destination;
54         int weight = graph->edges[i].weight;
55         if (distance[u] != INF && distance[u] + weight < distance[v]) {
56             printf("Graph contains negative-weight cycle\n");
57             return;
58         }
59     }
60 }
```

```
Output
2 1 3
2 7 1
1 10 -1
1 2 2
1 4 -2
10 9 -3
10 1 3
10 7 2
10 5 1
7 3 0
7 10 1
7 2 1
7 8 2
9 6 6
9 3 4
9 10 7
4 8 5
4 1 9
4 5 6
6 2 4
6 3 0
6 9 0
Enter the source vertex: 1
Shortest path to 1: 0 cost = 0
Shortest path to 2: 0 1 cost = 2
Shortest path to 3: 0 1 5 2 cost = -1
Shortest path to 4: 0 3 cost = -2
Shortest path to 5: 0 9 4 cost = 0
Shortest path to 6: 0 1 5 cost = -1
Shortest path to 7: 0 9 6 cost = 1
Shortest path to 8: 0 9 4 7 cost = 1
Shortest path to 9: 0 9 8 cost = -4
Shortest path to 10: 0 9 cost = -1
=== Code Execution Successful ===
```

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define INF 9999
```

```

#define MAX_VERTICES 100
//22BAI1471
struct Edge {
    int source, destination, weight;
};
struct Graph {
    int numVertices, numEdges;
    struct Edge* edges;
};
struct Graph* createGraph(int numVertices, int numEdges) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = numVertices;
    graph->numEdges = numEdges;
    graph->edges = (struct Edge*)malloc(numEdges * sizeof(struct Edge));
    return graph;
}
void printPath(int parent[], int destination) {
    if (parent[destination] == -1)
        printf("%d ", destination);
    else {
        printPath(parent, parent[destination]);
        printf("%d ", destination);
    }
}
//bellmanford algorithm
void bellmanFord(struct Graph* graph, int source) {
    int numVertices = graph->numVertices;
    int numEdges = graph->numEdges;
    int distance[numVertices];
    int parent[numVertices];
    for (int i = 0; i < numVertices; i++) {
        distance[i] = INF;
        parent[i] = -1;
    }
    distance[source] = 0;
    for (int i = 1; i <= numVertices - 1; i++) {
        for (int j = 0; j < numEdges; j++) {
            int u = graph->edges[j].source;
            int v = graph->edges[j].destination;
            int weight = graph->edges[j].weight;
            if (distance[u] != INF && distance[u] + weight < distance[v]) {
                distance[v] = distance[u] + weight;
                parent[v] = u;
            }
        }
    }
}

```

```

    }
}
// Checking for -ve weight cycles
for (int i = 0; i < numEdges; i++) {
    int u = graph->edges[i].source;
    int v = graph->edges[i].destination;
    int weight = graph->edges[i].weight;
    if (distance[u] != INF && distance[u] + weight < distance[v]) {
        printf("Graph contains negative-weight cycle\n");
        return;
    }
}
for (int i = 0; i < numVertices; i++) {
    printf("Shortest path to %d: ", i + 1);
    printPath(parent, i);
    printf("cost = %d\n", distance[i]);
}
}

int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    struct Graph* graph = createGraph(numVertices, numEdges);

    printf("Enter the edges in the format <source> <destination> <weight>:\n");
    for (int i = 0; i < numEdges; i++) {
        int source, destination, weight;
        scanf("%d %d %d", &source, &destination, &weight);
        graph->edges[i].source = source - 1;
        graph->edges[i].destination = destination - 1;
        graph->edges[i].weight = weight;
    }

    int source;
    printf("Enter the source vertex: ");
    scanf("%d", &source);

    bellmanFord(graph, source - 1);
}

```

```
    return 0;  
}
```

Output

/tmp/g6FWBjoAs6.o

Enter the number of vertices: 10

Enter the number of edges: 32

Enter the edges in the format <source> <destination> <weight>:

3 6 4

3 9 0

3 7 8

8 5 3

8 7 3

8 4 -2

5 10 2

5 8 1

5 4 1

2 6 -3

2 1 3

2 7 1

1 10 -1

1 2 2

1 4 -2

10 9 -3

10 1 3

10 7 2

10 5 1

7 3 0

7 10 1

7 2 1

7 8 2

9 6 6

9 3 4

9 10 7

4 8 5

4 1 9

4 5 6

6 2 4

6 3 0

6 9 0

Enter the source vertex: 1

Shortest path to 1: 0 cost = 0

Shortest path to 2: 0 1 cost = 2

Shortest path to 3: 0 1 5 2 cost = -1

Shortest path to 4: 0 3 cost = -2
Shortest path to 5: 0 9 4 cost = 0
Shortest path to 6: 0 1 5 cost = -1
Shortest path to 7: 0 9 6 cost = 1
Shortest path to 8: 0 9 4 7 cost = 1
Shortest path to 9: 0 9 8 cost = -4
Shortest path to 10: 0 9 cost = -1

=== Code Execution Successful ===

Conclusion

Thus Bellman-ford algorithm is highly efficient when it comes to the graphs with negative weights or edges and thus gives optimal output which cannot be obtained from other graph methods like Dijkstra algo and Prims algorithm.

This algorithm is widely used in the Computer networks field, used in networking protocols to find the shortest path to send data packets.