

### // 23. Implementation of Shortest Path Algorithms using Dijkstra's Algorithm

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {
    int cost[MAX][MAX], distance[MAX], pred[MAX];
    int visited[MAX], count, mindistance, nextnode, i, j;

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if (Graph[i][j] == 0)
                cost[i][j] = INFINITY;
            else
                cost[i][j] = Graph[i][j];

    for (i = 0; i < n; i++) {
        distance[i] = cost[start][i];
        pred[i] = start;
        visited[i] = 0;
    }

    distance[start] = 0;
    visited[start] = 1;
    count = 1;

    while (count < n - 1) {
        mindistance = INFINITY;

        for (i = 0; i < n; i++)
            if (distance[i] < mindistance && !visited[i]) {
                mindistance = distance[i];
                nextnode = i;
            }

        visited[nextnode] = 1;
        for (i = 0; i < n; i++)
            if (!visited[i])
                if (mindistance + cost[nextnode][i] < distance[i]) {
                    distance[i] = mindistance + cost[nextnode][i];
                    pred[i] = nextnode;
                }
    }
}
```

```

    }
    count++;
}

for (i = 0; i < n; i++)
    if (i != start) {
        printf("\nDistance from source to %d: %d", i, distance[i]);
    }
}

int main() {
    int Graph[MAX][MAX], i, j, n, u;
    n = 7;

    Graph[0][0] = 0;
    Graph[0][1] = 0;
    Graph[0][2] = 1;
    Graph[0][3] = 2;
    Graph[0][4] = 0;
    Graph[0][5] = 0;
    Graph[0][6] = 0;

    Graph[1][0] = 0;
    Graph[1][1] = 0;
    Graph[1][2] = 2;
    Graph[1][3] = 0;
    Graph[1][4] = 0;
    Graph[1][5] = 3;
    Graph[1][6] = 0;

    Graph[2][0] = 1;
    Graph[2][1] = 2;
    Graph[2][2] = 0;
    Graph[2][3] = 1;
    Graph[2][4] = 3;
    Graph[2][5] = 0;
    Graph[2][6] = 0;

    Graph[3][0] = 2;
    Graph[3][1] = 0;
    Graph[3][2] = 1;
    Graph[3][3] = 0;
    Graph[3][4] = 0;
    Graph[3][5] = 0;

```

```
Graph[3][6] = 1;
```

```
Graph[4][0] = 0;
```

```
Graph[4][1] = 0;
```

```
Graph[4][2] = 3;
```

```
Graph[4][3] = 0;
```

```
Graph[4][4] = 0;
```

```
Graph[4][5] = 2;
```

```
Graph[4][6] = 0;
```

```
Graph[5][0] = 0;
```

```
Graph[5][1] = 3;
```

```
Graph[5][2] = 0;
```

```
Graph[5][3] = 0;
```

```
Graph[5][4] = 2;
```

```
Graph[5][5] = 0;
```

```
Graph[5][6] = 1;
```

```
Graph[6][0] = 0;
```

```
Graph[6][1] = 0;
```

```
Graph[6][2] = 0;
```

```
Graph[6][3] = 1;
```

```
Graph[6][4] = 0;
```

```
Graph[6][5] = 1;
```

```
Graph[6][6] = 0;
```

```
u = 0;
```

```
Dijkstra(Graph, n, u);
```

```
return 0;
```

```
}
```

```
D:\data structures lab\dijkstra's algorithm.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug binary search 1.c merge sort.c merge sort 1.c quick sort.c heap sort.c alv tree.c breadth first search.c depth first search.c dijkstra's algorithm.c

1 // 23. Implementation of Shortest Path Algorithms using Dijkstra's Algorithm
2 #include <stdio.h>
3 #define INFINITY 9999
4 #define MAX 10
5
6 void Dijkstra(int Graph[MAX][MAX], int n, int start);
7
8 void Dijkstra(int Graph[MAX][MAX], int n, int start) {
9     int cost[MAX][MAX], distance[MAX], pred[MAX];
10    int visited[MAX], count, mindistance, nextnode, i, j;
11
12    for (i = 0; i < n; i++)
13        for (j = 0; j < n; j++)
14            if (Graph[i][j] != 0)
15                cost[i][j] = INFINITY;
16            else
17                cost[i][j] = Graph[i][j];
18
19    for (i = 0; i < n; i++) {
20        distance[i] = cost[start][i];
21        pred[i] = start;
22        visited[i] = 0;
23    }
24
25    distance[start] = 0;
26    visited[start] = 1;
27    count = 1;
28
29    while (count < n - 1) {
30        mindistance = INFINITY;
31
32        for (i = 0; i < n; i++)
33            if (distance[i] < mindistance && !visited[i]) {
34                mindistance = distance[i];
35                nextnode = i;
36            }
37    }
38}
```

```
D:\data structures lab\dijkstra's algorithm.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug binary search 1.c merge sort.c merge sort 1.c quick sort.c heap sort.c alv tree.c breadth first search.c depth first search.c dijkstra's algorithm.c

37 }
38
39 visited[nextnode] = 1;
40 for (i = 0; i < n; i++)
41     if (!visited[i])
42         if (mindistance + cost[nextnode][i] < distance[i]) {
43             distance[i] = mindistance + cost[nextnode][i];
44             pred[i] = nextnode;
45         }
46     count++;
47 }
48
49 for (i = 0; i < n; i++)
50     if (i != start) {
51         printf("\nDistance from source to %d: %d", i, distance[i]);
52     }
53 }
54 }
55 int main() {
56     int Graph[MAX][MAX], i, j, n, u;
57     n = 7;
58
59     Graph[0][0] = 0;
60     Graph[0][1] = 0;
61     Graph[0][2] = 1;
62     Graph[0][3] = 2;
63     Graph[0][4] = 0;
64     Graph[0][5] = 0;
65     Graph[0][6] = 0;
66
67     Graph[1][0] = 0;
68     Graph[1][1] = 0;
69     Graph[1][2] = 2;
70     Graph[1][3] = 0;
71     Graph[1][4] = 0;
72     Graph[1][5] = 3;
73     Graph[1][6] = 0;
74 }
```

```
D:\data structures lab\dijkstra algorithm.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug binary search 1.c merge sort.c merge sort1.c quick sort.c heap sort.c alv tree.c breadth first search.c depth first search.c dijkstra algorithm.c
73 Graph[1][6] = 0;
74
75 Graph[2][0] = 1;
76 Graph[2][1] = 2;
77 Graph[2][2] = 0;
78 Graph[2][3] = 1;
79 Graph[2][4] = 3;
80 Graph[2][5] = 0;
81 Graph[2][6] = 0;
82
83 Graph[3][0] = 2;
84 Graph[3][1] = 0;
85 Graph[3][2] = 1;
86 Graph[3][3] = 0;
87 Graph[3][4] = 0;
88 Graph[3][5] = 0;
89 Graph[3][6] = 1;
90
91 Graph[4][0] = 0;
92 Graph[4][1] = 0;
93 Graph[4][2] = 3;
94 Graph[4][3] = 0;
95 Graph[4][4] = 0;
96 Graph[4][5] = 2;
97 Graph[4][6] = 0;
98
99 Graph[5][0] = 0;
100 Graph[5][1] = 3;
101 Graph[5][2] = 0;
102 Graph[5][3] = 0;
103 Graph[5][4] = 2;
104 Graph[5][5] = 0;
105 Graph[5][6] = 1;
106
107 Graph[6][0] = 0;
108 Graph[6][1] = 0;
109 Graph[6][2] = 0;
110 Graph[6][3] = 1;
```

```
D:\data structures lab\dijkstra algorithm.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug binary search 1.c merge sort.c merge sort1.c quick sort.c heap sort.c alv tree.c breadth first search.c depth first search.c dijkstra algorithm.c
82
83 Graph[3][0] = 2;
84 Graph[3][1] = 0;
85 Graph[3][2] = 1;
86 Graph[3][3] = 0;
87 Graph[3][4] = 0;
88 Graph[3][5] = 0;
89 Graph[3][6] = 1;
90
91 Graph[4][0] = 0;
92 Graph[4][1] = 0;
93 Graph[4][2] = 3;
94 Graph[4][3] = 0;
95 Graph[4][4] = 0;
96 Graph[4][5] = 2;
97 Graph[4][6] = 0;
98
99 Graph[5][0] = 0;
100 Graph[5][1] = 3;
101 Graph[5][2] = 0;
102 Graph[5][3] = 0;
103 Graph[5][4] = 2;
104 Graph[5][5] = 0;
105 Graph[5][6] = 1;
106
107 Graph[6][0] = 0;
108 Graph[6][1] = 0;
109 Graph[6][2] = 0;
110 Graph[6][3] = 1;
111 Graph[6][4] = 0;
112 Graph[6][5] = 1;
113 Graph[6][6] = 0;
114
115 u = 0;
116 Dijkstra(Graph, n, u);
117
118 return 0;
119 }
```

```
D:\data structures lab\dijkstra algorithm.exe
Distance from source to 1: 3
Distance from source to 2: 1
Distance from source to 3: 2
Distance from source to 4: 4
Distance from source to 5: 4
Distance from source to 6: 3
-----
Process exited after 2.938 seconds with return value 0
Press any key to continue . . .
```