

// 25. Implementation of Minimum Spanning Tree using Kruskal Algorithm

```
#include <stdio.h>
#define MAX 30
typedef struct edge {
    int u, v, w;
} edge;
typedef struct edge_list {
    edge data[MAX];
    int n;
} edge_list;
edge_list elist;
int Graph[MAX][MAX], n;
edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();
```

```
void kruskalAlgo() {
    int belongs[MAX], i, j, cno1, cno2;
    elist.n = 0;

    for (i = 1; i < n; i++)
        for (j = 0; j < i; j++) {
            if (Graph[i][j] != 0) {
                elist.data[elist.n].u = i;
                elist.data[elist.n].v = j;
                elist.data[elist.n].w = Graph[i][j];
                elist.n++;
            }
        }

    sort();

    for (i = 0; i < n; i++)
        belongs[i] = i;

    spanlist.n = 0;

    for (i = 0; i < elist.n; i++) {
        cno1 = find(belongs, elist.data[i].u);
```

```

    cno2 = find(belongs, elist.data[i].v);

    if (cno1 != cno2) {
        spanlist.data[spanlist.n] = elist.data[i];
        spanlist.n = spanlist.n + 1;
        applyUnion(belongs, cno1, cno2);
    }
}
}

int find(int belongs[], int vertexno) {
    return (belongs[vertexno]);
}

void applyUnion(int belongs[], int c1, int c2) {
    int i;

    for (i = 0; i < n; i++)
        if (belongs[i] == c2)
            belongs[i] = c1;
}

void sort() {
    int i, j;
    edge temp;

    for (i = 1; i < elist.n; i++)
        for (j = 0; j < elist.n - 1; j++)
            if (elist.data[j].w > elist.data[j + 1].w) {
                temp = elist.data[j];
                elist.data[j] = elist.data[j + 1];
                elist.data[j + 1] = temp;
            }
}

void print() {
    int i, cost = 0;

    for (i = 0; i < spanlist.n; i++) {
        printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
        cost = cost + spanlist.data[i].w;
    }
}

```

```
    printf("\nSpanning tree cost: %d", cost);  
}
```

```
int main() {  
    int i, j, total_cost;
```

```
    n = 6;
```

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

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

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

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

```
    Graph[4][0] = 0;  
    Graph[4][1] = 0;  
    Graph[4][2] = 4;
```

```
Graph[4][3] = 3;  
Graph[4][4] = 0;  
Graph[4][5] = 0;  
Graph[4][6] = 0;
```

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

```
kruskalAlgo();  
print();  
}
```

```
D:\data structures lab\Minimum Spanning Tree using Kruskal Algorithm.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Minimum Spanning Tree using Prim's Algorithm.c [*] Minimum Spanning Tree using Kruskal Algorithm.c
1 // 25. Implementation of Minimum Spanning Tree using Kruskal Algorithm
2 #include <stdio.h>
3
4 #define MAX 30
5
6 typedef struct edge {
7     int u, v, w;
8 } edge;
9
10 typedef struct edge_list {
11     edge data[MAX];
12     int n;
13 } edge_list;
14
15 edge_list elist;
16
17 int Graph[MAX][MAX], n;
18 edge_list spanlist;
19
20 void kruskalAlgo();
21 int find(int belongs[], int vertexno);
22 void applyUnion(int belongs[], int c1, int c2);
23 void sort();
24 void print();
25
26
27 void kruskalAlgo() {
28     int belongs[MAX], i, j, cno1, cno2;
29     elist.n = 0;
30
31     for (i = 1; i < n; i++)
32         for (j = 0; j < i; j++) {
33             if (Graph[i][j] != 0) {
34                 elist.data[elist.n].u = i;
35                 elist.data[elist.n].v = j;
36                 elist.data[elist.n].w = Graph[i][j];
37                 elist.n++;
38             }
39         }
40
41     sort();
42
43     for (i = 0; i < n; i++)
44         belongs[i] = i;
45
46     spanlist.n = 0;
47
48     for (i = 0; i < elist.n; i++) {
49         cno1 = find(belongs, elist.data[i].u);
50         cno2 = find(belongs, elist.data[i].v);
51
52         if (cno1 != cno2) {
53             spanlist.data[spanlist.n] = elist.data[i];
54             spanlist.n = spanlist.n + 1;
55             applyUnion(belongs, cno1, cno2);
56         }
57     }
58 }
59
60 int find(int belongs[], int vertexno) {
61     return (belongs[vertexno]);
62 }
63
64 void applyUnion(int belongs[], int c1, int c2) {
65     int i;
66
67     for (i = 0; i < n; i++)
68         if (belongs[i] == c2)
69             belongs[i] = c1;
70 }
71
72
73 void sort() {
74     int i, j, temp;
75     for (i = 0; i < elist.n; i++)
76         for (j = i + 1; j < elist.n; j++)
77             if (elist.data[i].w > elist.data[j].w)
78                 temp = elist.data[i], elist.data[i] = elist.data[j], elist.data[j] = temp;
79 }
```

```
D:\data structures lab\Minimum Spanning Tree using Kruskal Algorithm.c - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Minimum Spanning Tree using Prim's Algorithm.c [*] Minimum Spanning Tree using Kruskal Algorithm.c
37 }
38 }
39 }
40
41 sort();
42
43 for (i = 0; i < n; i++)
44     belongs[i] = i;
45
46 spanlist.n = 0;
47
48 for (i = 0; i < elist.n; i++) {
49     cno1 = find(belongs, elist.data[i].u);
50     cno2 = find(belongs, elist.data[i].v);
51
52     if (cno1 != cno2) {
53         spanlist.data[spanlist.n] = elist.data[i];
54         spanlist.n = spanlist.n + 1;
55         applyUnion(belongs, cno1, cno2);
56     }
57 }
58
59
60 int find(int belongs[], int vertexno) {
61     return (belongs[vertexno]);
62 }
63
64 void applyUnion(int belongs[], int c1, int c2) {
65     int i;
66
67     for (i = 0; i < n; i++)
68         if (belongs[i] == c2)
69             belongs[i] = c1;
70 }
71
72
73 void sort() {
74     int i, j, temp;
75     for (i = 0; i < elist.n; i++)
76         for (j = i + 1; j < elist.n; j++)
77             if (elist.data[i].w > elist.data[j].w)
78                 temp = elist.data[i], elist.data[i] = elist.data[j], elist.data[j] = temp;
79 }
```

D:\data structures lab\Minimum Spanning Tree using Kruskal Algorithm.c - [Executing] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDH-GCC 4.9.2 64-bit Release

(globals)

Project Classes Debug Minimum Spanning Tree using Prim's Algorithm.c [*] Minimum Spanning Tree using Kruskal Algorithm.c

```
73 void sort() {
74     int i, j;
75     edge temp;
76
77     for (i = 1; i < elist.n; i++)
78         for (j = 0; j < elist.n - 1; j++)
79             if (elist.data[j].w > elist.data[j + 1].w) {
80                 temp = elist.data[j];
81                 elist.data[j] = elist.data[j + 1];
82                 elist.data[j + 1] = temp;
83             }
84 }
85
86
87 void print() {
88     int i, cost = 0;
89
90     for (i = 0; i < spanlist.n; i++) {
91         printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
92         cost = cost + spanlist.data[i].w;
93     }
94
95     printf("\nSpanning tree cost: %d", cost);
96 }
97
98 int main() {
99     int i, j, total_cost;
100
101     n = 6;
102
103     Graph[0][0] = 0;
104     Graph[0][1] = 4;
105     Graph[0][2] = 4;
106     Graph[0][3] = 0;
107     Graph[0][4] = 0;
108     Graph[0][5] = 0;
109     Graph[0][6] = 0;
```

Line: 153 Col: 2 Sel: 0 Lines: 153 Length: 2866 Insert Done parsing in 0.016 seconds

73°F Cloudy

D:\data structures lab\Minimum Spanning Tree using Kruskal Algorithm.c - [Executing] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDH-GCC 4.9.2 64-bit Release

(globals)

Project Classes Debug Minimum Spanning Tree using Prim's Algorithm.c [*] Minimum Spanning Tree using Kruskal Algorithm.c

```
110
111     Graph[1][0] = 4;
112     Graph[1][1] = 0;
113     Graph[1][2] = 2;
114     Graph[1][3] = 0;
115     Graph[1][4] = 0;
116     Graph[1][5] = 0;
117     Graph[1][6] = 0;
118
119     Graph[2][0] = 4;
120     Graph[2][1] = 2;
121     Graph[2][2] = 0;
122     Graph[2][3] = 3;
123     Graph[2][4] = 4;
124     Graph[2][5] = 0;
125     Graph[2][6] = 0;
126
127     Graph[3][0] = 0;
128     Graph[3][1] = 0;
129     Graph[3][2] = 3;
130     Graph[3][3] = 0;
131     Graph[3][4] = 3;
132     Graph[3][5] = 0;
133     Graph[3][6] = 0;
134
135     Graph[4][0] = 0;
136     Graph[4][1] = 0;
137     Graph[4][2] = 4;
138     Graph[4][3] = 3;
139     Graph[4][4] = 0;
140     Graph[4][5] = 0;
141     Graph[4][6] = 0;
142
143     Graph[5][0] = 0;
144     Graph[5][1] = 0;
145     Graph[5][2] = 2;
146     Graph[5][3] = 0;
147     Graph[5][4] = 3;
148     Graph[5][5] = 0;
149     Graph[5][6] = 0;
150
151     kruskalAlgo();
152     print();
153 }
```

Line: 153 Col: 2 Sel: 0 Lines: 153 Length: 2866 Insert Done parsing in 0.016 seconds

73°F Cloudy

D:\data structures lab\Minimum Spanning Tree using Kruskal Algorithm.exe

```
2 - 1 : 2
5 - 2 : 2
3 - 2 : 3
4 - 3 : 3
1 - 0 : 4
Spanning tree cost: 14
-----
Process exited after 3.196 seconds with return value 23
Press any key to continue . . .
```