

```

#include <stdio.h>
#include <limits.h>

int main() {
    printf("Enter number of nodes: ");
    int n;
    scanf("%d", &n);
    int g[n][n];
    printf("Enter adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &g[i][j]);
        }
    }
    int s;
    printf("Enter source node: ");
    scanf("%d", &s);
    int d[n];
    int v[n];
    for (int i = 0; i < n; i++) {
        d[i] = INT_MAX;
        v[i] = 0;
    }
    d[s] = 0;
    for (int count = 0; count < n - 1; count++) {
        int u = -1;
        for (int i = 0; i < n; i++) {
            if (!v[i] && (u == -1 || d[i] < d[u])) {
                u = i;
            }
        }
        if (d[u] == INT_MAX) break;
        v[u] = 1;
        for (int i = 0; i < n; i++) {
            if (g[u][i] && !v[i] && d[u] != INT_MAX && d[u] + g[u][i] < d[i]) {
                d[i] = d[u] + g[u][i];
            }
        }
    }
    printf("Distance from node %d:\n", s);
}

```

```

        scanf("%d", &g[i][j]);
    }
}

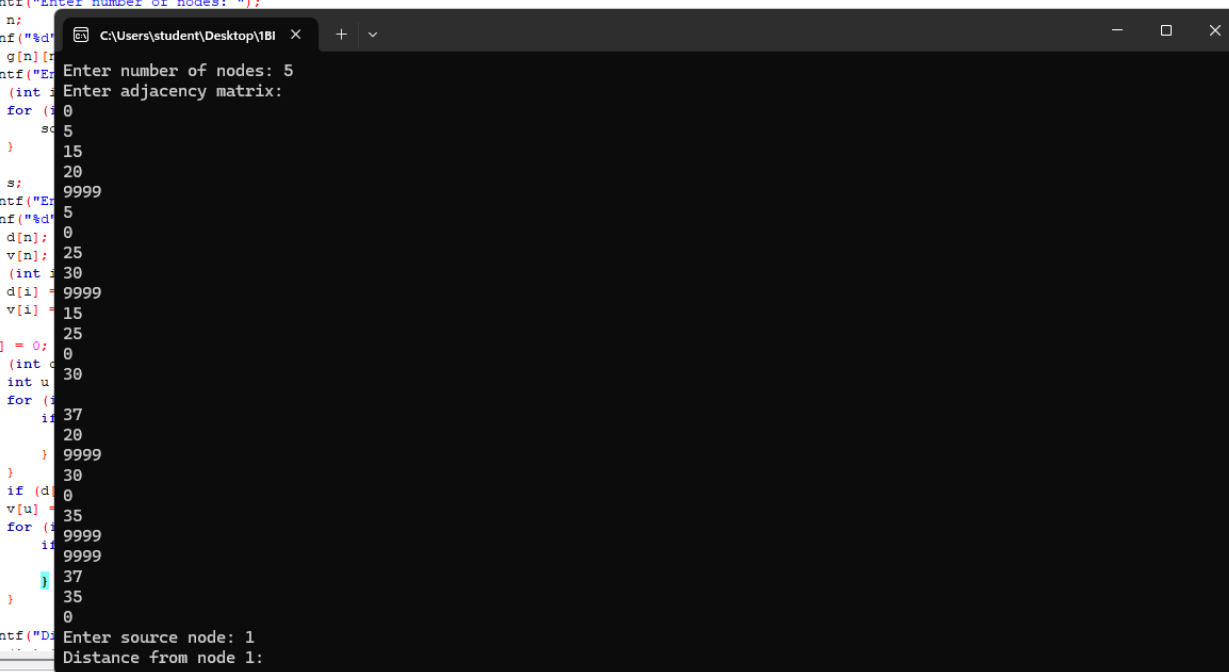
int s;
printf("Enter source node: ");
scanf("%d", &s);
int d[n];
int v[n];
for (int i = 0; i < n; i++) {
    d[i] = INT_MAX;
    v[i] = 0;
}
d[s] = 0;
for (int count = 0; count < n - 1; count++) {
    int u = -1;
    for (int i = 0; i < n; i++) {
        if (!v[i] && (u == -1 || d[i] < d[u])) {
            u = i;
        }
    }
    if (d[u] == INT_MAX) break;
    v[u] = 1;
    for (int i = 0; i < n; i++) {
        if (g[u][i] && !v[i] && d[u] != INT_MAX && d[u] + g[u][i] < d[i]) {
            d[i] = d[u] + g[u][i];
        }
    }
}

printf("Distance from node %d:\n", s);
for (int i = 0; i < n; i++) {
    if (d[i] == INT_MAX) {
        printf("INF ");
    } else {
        printf("%d ", d[i]);
    }
}

printf("\n");
return 0;
}

```

```
n() {  
    printf("Enter number of nodes: ");  
    n;  
    scanf("%d", &n);  
    g[n] = (int*) malloc(n * sizeof(int));  
    printf("Enter adjacency matrix:  
    (int) 0  
    for (i = 0; i < n; i++)  
        for (j = 0; j < n; j++)  
            scanf("%d", &g[i][j]);  
    s;  
    printf("Enter source node: ");  
    scanf("%d", &s);  
    d[n] = (int*) malloc(n * sizeof(int));  
    v[n] = (int*) malloc(n * sizeof(int));  
    (int) 0  
    d[i] = 9999;  
    v[i] = 15;  
    25  
    0  
    (int) 0  
    int u;  
    for (i = 0; i < n; i++)  
        if (d[i] < 9999)  
            u = i;  
    if (d[u] == 0)  
        v[u] = 35;  
    for (i = 0; i < n; i++)  
        if (i != u)  
            d[i] = 9999;  
    37  
    20  
    9999  
    30  
    if (d[u] == 0)  
        v[u] = 35;  
    for (i = 0; i < n; i++)  
        if (i != u)  
            d[i] = 9999;  
    37  
    35  
    0  
    printf("Distance from node 1: ");  
    scanf("%d", &d);  
}
```



Search results X Cccc X Build log X Build messages X CppCheck/Vera++ X CppCheck/Vera++ messages X Cscope X Debugger X DoxyBlocks X Fortran info X Closed files list X Thread search X

```
printf("Enter number of nodes: ");
```

```
scanf("%d", &n);
```

```
int i;
```

```
for (i = 0; i < n; i++)
```

```
{
```

```
scanf("%d", &adj[i][i]);
```

```
adj[i][i] = 0;
```

```
for (j = i + 1; j < n; j++)
```

```
{
```

```
scanf("%d", &adj[i][j]);
```

```
adj[j][i] = adj[i][j];
```

```
int s;
```

```
scanf("%d", &s);
```

```
int d[s];
```

```
d[s] = 0;
```

```
for (i = 0; i < n; i++)
```

```
{
```

```
scanf("%d", &adj[i][s]);
```

```
adj[s][i] = adj[i][s];
```

```
int u;
```

```
for (u = 0; u < n; u++)
```

```
{
```

```
scanf("%d", &adj[u][s]);
```

```
adj[s][u] = adj[u][s];
```

```
int i;
```

```
for (i = 0; i < n; i++)
```

```
{
```

```
scanf("%d", &adj[i][s]);
```

```
adj[s][i] = adj[i][s];
```

```
int u;
```

```
for (u = 0; u < n; u++)
```

```
{
```

```
scanf("%d", &adj[u][s]);
```

```
adj[s][u] = adj[u][s];
```

```
int i;
```

```
for (i = 0; i < n; i++)
```

```
{
```

```
scanf("%d", &adj[i][s]);
```

```
adj[s][i] = adj[i][s];
```

```
int u;
```

```
for (u = 0; u < n; u++)
```

```
{
```

```
scanf("%d", &adj[u][s]);
```

```
adj[s][u] = adj[u][s];
```

```
int i;
```

```
for (i = 0; i < n; i++)
```

```
{
```

```
scanf("%d", &adj[i][s]);
```

```
adj[s][i] = adj[i][s];
```

```
int u;
```

```
for (u = 0; u < n; u++)
```

```
{
```

```
scanf("%d", &adj[u][s]);
```

```
adj[s][u] = adj[u][s];
```

```
20  
9999
```

```
5
```

```
0
```

```
25
```

```
30
```

```
9999
```

```
15
```

```
25
```

```
0
```

```
30
```

```
37
```

```
20
```

```
9999
```

```
30
```

```
0
```

```
35
```

```
9999
```

```
37
```

```
35
```

```
0
```

```
Enter source node: 1
```

```
Distance from node 1:
```

```
5 0 20 25 57
```

```
Process returned 0 (0x0) execution time : 190.988 s  
Press any key to continue.
```

```

#include <stdio.h>

#define MAX 30

typedef struct edge {
    int u, v, cost;
} Edge;

Edge edges[MAX];
int parent[MAX];

int find(int i) {
    while (parent[i])
        i = parent[i];
    return i;
}

int uni(int i, int j) {
    if (i != j) {
        parent[j] = i;
        return 1;
    }
    return 0;
}

void kruskals(int c[MAX][MAX], int n) {
    int i, j, u, v, a, b, min, ne = 0, mincost = 0;

    for (i = 1; i <= n; i++)
        parent[i] = 0;

    while (ne < n - 1) {
        min = 9999;
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= n; j++) {
                if (c[i][j] < min) {
                    min = c[i][j];
                    u = a = i;
                    v = b = j;
                }
            }
        }
    }
}

```

```

        }
    }
}

u = find(u);
v = find(v);

if (uni(u, v)) {
    printf("(%d, %d) -> %d\n", a, b, min);
    mincost += min;
    ne++;
}

c[a][b] = c[b][a] = 9999;
}

printf("Minimum Cost = %d\n", mincost);

int main() {
    int c[MAX][MAX], n, i, j;

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    printf("Enter the cost matrix:\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
            if (c[i][j] == 0)
                c[i][j] = 9999;
        }
    }

    printf("The Minimum Spanning Tree is:\n");
    kruskals(c, n);

    return 0;
}

```

```
Find(v);
```

```
uni(u  
print  
minco  
ne++  
[b] =  
Minim  
K] [MA  
Enter  
i", e  
Enter  
1; i  
(j =  
scanf  
if (c  
c  
The M  
(c, n
```

```
C:\Users\student\Desktop\181 x + v  
Enter the number of vertices: 5  
Enter the cost matrix:  
0  
5  
15  
20  
9999  
5  
0  
25  
30  
9999  
15  
25  
0  
30  
9999  
30  
0  
35  
9999  
9999  
37  
35  
0  
The Minimum Spanning Tree is:  
(1, 2) -> 5  
(1, 3) -> 15
```

Build file: "no target" in "no project" (compiler: unknown) ==
Build finished: 0 error(s), 0 warning(s) (0 minute(s), 4 second(s)) ==

```
C:\Users\student\Desktop\181 x + v -
```

```
5
0
25
30
9999
15
25
0
30
37
20
9999
30
0
35
9999
9999
37
35
0
The Minimum Spanning Tree is:
(1, 2) -> 5
(1, 3) -> 15
(1, 4) -> 20
(4, 5) -> 35
Minimum Cost = 75

Process returned 0 (0x0)   execution time : 49.391 s
Press any key to continue.
```