

Dijkstra's

```
include <stdio.h>
```

```
include <limits.h>
```

```
#define MAX-VERTICES 100
```

```
void dijkstra(int c[MAX-VERTICES][MAX-VERTICES], int start,
```

```
int dist[MAX-VERTICES];
```

```
int vis[MAX-VERTICES];
```

```
int count, u, i, j;
```

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

```
    dist[i] = INT_MAX;
```

```
    vis[i] = 0;
```

```
}
```

```
dist[start] = 0;
```

```
count = 0;
```

```
while (count < n-1) {
```

```
    int min_dist = INT_MAX;
```

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

```
        if (!vis[i] && dist[i] < min_dist) {
```

```
            min_dist = dist[i];
```

```
            u = i;
```

```
        }
```

```
}
```

```
vis[u] = 1;
```

```
for (j=0; j<n; j++) {
```

```
    if (!vis[j] && c[u][j] && dist[u] !=
```

```
    INT_MAX && dist[u] + c[u][j]
```

```
    < dist[j]) {
```

```
        dist[j] = dist[u] + c[u][j];
```

```
}
```

```
}
```

```
count++;
```

```
}
```

```
printf("Shortest distances from source %d:\n", start);
```

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

```
    if (dist[i] == INT_MAX) {
```

```

        printf("%.d → %.d : unreachable\n", src, i);
    } else {
        printf("%.d → %.d : %.d\n", src, i, dist[i]);
    }
}

int main()
{
    int n, src, i, j;
    int c[MAX-VERTICES][MAX-VERTICES];
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter the cost adjacency matrix (enter 999 for infinity);\n");

    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            scanf("%d", &c[i][j]);

    printf("Enter the source vertex: ");
    scanf("%d", &src);
    dijkstra(c, n, src);

    return 0;
}

```

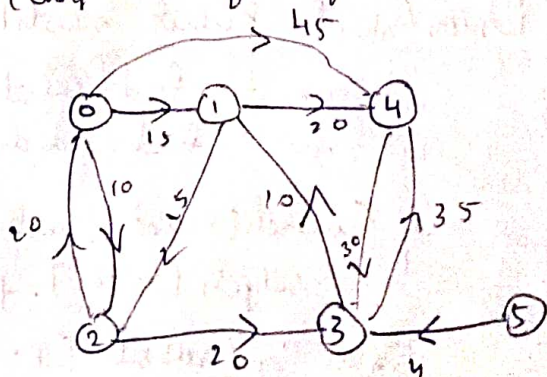
O/p: Enter the number of vertices: 6
Enter the cost adjacency matrix (enter 999 for infinity);

0	15	10	999	45	999
999	0	15	999	20	999
20	999	0	20	999	999
999	10	999	0	35	999
999	999	999	30	0	999
999	999	999	4	999	0

Enter the source vertex: 5

Shortest distances from source 5:

5 → 0: 49	5 → 4: 34
5 → 1: 14	5 → 5: 0
5 → 2: 29	
5 → 3: 4	



Kruskal

```
#include <stdio.h>
#include <stdlib.h>
#define MAX-VERTICES 100

struct Edge {
    int src, dest, weight;
};

struct Subset {
    int parent;
    int rank;
};

int find(struct Subset subsets[], int i);
void Union(struct Subset subsets[], int x, int y);
void kruskalMST(struct Edge edges[], int n, int e);

int compare(const void * a, const void * b) {
    struct Edge * edge1 = (struct Edge *)a;
    struct Edge * edge2 = (struct Edge *)b;
    return edge1->weight - edge2->weight;
}

int find(struct Subset subsets[], int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
}

void Union(struct Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);
    if (subsets[xroot].rank < subsets[yroot].rank) {
        subsets[xroot].parent = yroot;
    } else if (subsets[xroot].rank > subsets[yroot].rank) {
        subsets[yroot].parent = xroot;
    } else {
        subsets[xroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
```

```
void KruskalMST(struct Edge edges[], int n, int e) {
```

```
    struct Edge result[n];
```

```
    int i = 0;
```

```
    int mincost = 0;
```

```
    struct Subset subsets[MAX_VERTICES];
```

```
    for (int v = 0; v < n; v++) {
```

```
        subsets[v].parent = v;
```

```
        subsets[v].rank = 0;
```

```
    }
```

```
    qsort(edges, e, sizeof(struct Edge), compare);
```

```
    int edge-count = 0;
```

```
    while (edge-count < n - 1 && i < e) {
```

```
        struct Edge edge = edges[i++];
```

```
        int x = find(subsets, next-edge.src);
```

```
        int y = find(subsets, next-edge.dest);
```

```
        if (x != y) {
```

```
            result[edge-count++] = next-edge;
```

```
            Union(subsets, x, y);
```

```
            mincost += next-edge.weight;
```

```
        }
```

```
    }
```

```
    printf("Edges in the Minimum Spanning Tree: \n");
```

```
    for (i = 0; i < edge-count; i++) {
```

```
        printf("%d - %d : %d \n", result[i].src, result[i].dest,  
               result[i].weight);
```

```
    }
```

```
    printf("Minimum cost of MST = %d \n", mincost);
```

```
}
```

```
int main() {
```

```
    int n, i, j;
```

```
    struct Edge graph[MAX_VERTICES * MAX_VERTICES];
```

```
    int e = 0;
```

```
    printf("Enter the number of vertices: ");
```

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



```
print f("Enter the cost adjacency matrix (enter 9999 for infinity)");
```

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

```
    for (j=0; j<n; j++) {
```

```
        int weight;
```

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

```
        if (weight != 9999 && i != j) {
```

```
            graph[e].src = i;
```

```
            graph[e].dest = j;
```

```
            graph[e].weight = weight;
```

```
            e++;
```

```
        }
```

```
    }
```

```
    KruskalMST(graph, n, e);
```

```
    return 0;
```

```
}
```

O/p:

Enter the number of vertices : 7

Enter the cost adjacency matrix (enter 9999 for infinity);

0	28	9999	9999	9999	10	9999
28	0	16	9999	9999	9999	14
9999	16	0	12	9999	9999	9999
9999	9999	12	0	22	9999	18
9999	9999	9999	22	0	25	24
10	9999	9999	9999	25	0	9999
9999	14	9999	18	24	9999	0

Edges in the Minimum Spanning Tree:

0-5 : 10

3-2 : 12

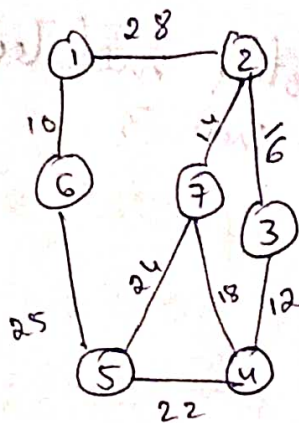
6-1 : 14

1-2 : 16

7-4 : 22

4-5 : 25

Minimum cost of MST : 99



10/12