LAB -10

Question :

1. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.
2. Program on Kruskal's algorithm.
3. Implement "N-Queens Problem" using Backtracking.

1.SOURCE CODE:

```
#include <stdio.h>
#define MAX 100
#define INF 9999

void dijkstras(int c[MAX][MAX], int n, int src) {
   int dist[MAX];  // To store the shortest distance from src to each vertex
   int vis[MAX];   // To track if a vertex is visited
   int count, min, u, i, j;

   // Initialize distance and visited arrays
   for (i = 1; i <= n; i++) {
      dist[i] = c[src][i];
      vis[i] = 0;
   }

   dist[src] = 0;
   vis[src] = 1;
   count = 1;

   while (count != n) {
      min = INF;

      // Find the vertex with the minimum distance which is not yet visited
      for (j = 1; j <= n; j++) {
         if (dist[j] < min && vis[j] != 1) {
            min = dist[j];
```

```
                u = j;
            }
        }

        vis[u] = 1;
        count++;

        // Update the distances of the adjacent vertices of the selected vertex
        for (j = 1; j <= n; j++) {
            if ((min + c[u][j] < dist[j]) && (vis[j] != 1)) {
                dist[j] = min + c[u][j];
            }
        }
    }

    // Print the shortest distances from src to all vertices
    printf("Shortest distances from node %d:\n", src);
    for (i = 1; i <= n; i++) {
        printf("%d -> %d : %d\n", src, i, dist[i]);
    }
}

int main() {
    int n, src;
    int cost[MAX][MAX];

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

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

```
            }
        }
    }

    printf("Enter the source node: ");
    scanf("%d", &src);

    dijkstras(cost, n, src);

    return 0;
}
```

RESULT:

```
C:\Users\student\Desktop\IBM

Enter the number of vertices: 5
Enter the cost matrix:
0 60 100 9999 10
9999 0 9999 50 9999
9999 9999 0 9999 20
9999 9999 20 0 9999
9999 9999 9999 5 0
Enter the source node: 1
Shortest distances from node 1:
1 -> 1 : 0
1 -> 2 : 60
1 -> 3 : 35
1 -> 4 : 15
1 -> 5 : 10

Process returned 0 (0x0)   execution time : 40.158 s
Press any key to continue.
```

2.SOURCE CODE:

```
#include <stdio.h>
#define MAX 100
#define INF 9999

// Structure to represent an edge in the graph
struct Edge {
   int u, v, weight;
};

// Structure to represent a subset for union-find
struct Subset {
   int parent;
   int rank;
};

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[yroot].parent = xroot;
      subsets[xroot].rank++;
   }
```

```
}

void kruskals(int c[MAX][MAX], int n) {
   struct Edge edges[MAX * MAX];  // Array to store all edges
   struct Subset subsets[MAX];
   int ne = 0;  // Number of edges in minimum spanning tree
   int mincost = 0;  // Cost of minimum spanning tree
   int i, j, k;

   // Initialize subsets for union-find
   for (i = 1; i <= n; i++) {
      subsets[i].parent = i;
      subsets[i].rank = 0;
   }

   // Store all edges in the graph in the edges array
   k = 0;
   for (i = 1; i <= n; i++) {
      for (j = 1; j <= n; j++) {
         if (c[i][j] != INF) {
            edges[k].u = i;
            edges[k].v = j;
            edges[k].weight = c[i][j];
            k++;
         }
      }
   }

   // Sort edges based on their weights
   for (i = 0; i < k - 1; i++) {
      for (j = 0; j < k - i - 1; j++) {
         if (edges[j].weight > edges[j + 1].weight) {
            struct Edge temp = edges[j];
            edges[j] = edges[j + 1];
            edges[j + 1] = temp;
```

```
            }
        }
    }

    // Iterate through all sorted edges
    for (i = 0; i < k; i++) {
        int u = edges[i].u;
        int v = edges[i].v;

        int set_u = find(subsets, u);
        int set_v = find(subsets, v);

        if (set_u != set_v) {
            // Include this edge in the minimum spanning tree
            printf("%d -> %d : %d\n", u, v, edges[i].weight);
            Union(subsets, set_u, set_v);
            mincost += edges[i].weight;
            ne++;
        }
    }

    printf("Minimum cost of spanning tree: %d\n", mincost);
}

int main() {
    int n;
    int cost[MAX][MAX];

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

    printf("Enter the cost matrix:\n");
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &cost[i][j]);
```

```
            if (cost[i][j] == 0 && i != j) {
                cost[i][j] = INF;
            }
        }
    }

    kruskals(cost, n);

    return 0;
}
```

RESULT:

```
C:\Users\student\Desktop\IBI
Enter the number of vertices: 7
Enter the cost matrix:
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
1 -> 6 : 10
3 -> 4 : 12
2 -> 7 : 14
2 -> 3 : 16
4 -> 5 : 22
5 -> 6 : 25
Minimum cost of spanning tree: 99

Process returned 0 (0x0)   execution time : 340.851 s
Press any key to continue.
```

3.SOURCE CODE:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int place(int x[], int k) {
for (int i = 1; i < k; i++) {
if (x[i] == x[k] || abs(x[i] - x[k]) == abs(i - k)) {
return 0;
}
}
return 1;
}

void nqueens(int n) {
int x[MAX];
int k = 1;
x[k] = 0;

while (k > 0) {
x[k] = x[k] + 1;

while (x[k] <= n && !place(x, k)) {


x[k] = x[k] + 1;
}

if (x[k] <= n) {
if (k == n) {
for (int i = 1; i <= n; i++) {
printf("%d ", x[i]);
```

```
}
printf("\n");
} else {
k++;
x[k] = 0;
}
} else {
k--;
}
}
}

int main() {
int n;
printf("Enter the number of queens: ");


scanf("%d", &n);

if (n < 1 || n > MAX) {
printf("Invalid number of queens. Please enter a value between 1 and %d.\n",
MAX);
return 1;
}

printf("The solutions are:\n");
nqueens(n);

return 0;
}
```

RESULT:

```
Enter the number of queens: 5
The solutions are:
1 3 5 2 4
1 4 2 5 3
2 4 1 3 5
2 5 3 1 4
3 1 4 2 5
3 5 2 4 1
4 1 3 5 2
4 2 5 3 1
5 2 4 1 3
5 3 1 4 2
```