DSA Assignment 10

1.

```cpp
#include <iostream>
#include <vector>
using namespace std;

class Graph {
public:
    int n;
    vector<vector<int>> adjMatrix;
    vector<vector<int>> adjList;

    Graph(int v) {
        n = v;
        adjMatrix.resize(n, vector<int>(n, 0));
        adjList.resize(n);
    }

    void addEdge(int u, int v) {
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;

        adjList[u].push_back(v);
        adjList[v].push_back(u);
    }

    int degree(int v) {
        return adjList[v].size();
    }

    void adjacentVertices(int v) {
        cout << "Adjacent to " << v << ": ";
        for (int x : adjList[v])
            cout << x << " ";
        cout << endl;
    }

    int numberOfEdges() {
        int edges = 0;
        for (int i = 0; i < n; i++)
            edges += adjList[i].size();
        return edges / 2;
    }
};

int main() {
    Graph g(5);
```

```
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 4);

    cout << "Degree of vertex 0: " << g.degree(0) << endl;
    g.adjacentVertices(0);
    cout << "Total edges: " << g.numberOfEdges() << endl;

    return 0;
}
```

OUTPUT

```
Degree of vertex 0: 2
Adjacent to 0: 1 2
Total edges: 4
```

2.

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

void BFS(int start, vector<vector<int>>& graph) {
    int n = graph.size();
    vector<int> visited(n, 0);
    queue<int> q;

    q.push(start);
    visited[start] = 1;

    cout << "BFS Traversal: ";

    while (!q.empty()) {
        int node = q.front();
        q.pop();

        cout << node << " ";

        for (int x : graph[node]) {
            if (!visited[x]) {
                visited[x] = 1;
                q.push(x);
            }
        }
    }
}
```

```
int main() {
    vector<vector<int>> graph = {
        {1,2},
        {0,3},
        {0,4},
        {1},
        {2}
    };

    BFS(0, graph);

    return 0;
}
```

OUTPUT

```
BFS Traversal: 0 1 2 3 4
```

3.

```
#include <iostream>
#include <vector>
using namespace std;

void dfs(int node, vector<vector<int>>& graph, vector<int>&
visited) {
    visited[node] = 1;
    cout << node << " ";

    for (int x : graph[node]) {
        if (!visited[x])
            dfs(x, graph, visited);
    }
}

int main() {
    vector<vector<int>> graph = {
        {1,2},
        {0,3},
        {0,4},
        {1},
        {2}
    };

    vector<int> visited(5, 0);

    cout << "DFS Traversal: ";
    dfs(0, graph, visited);

    return 0;
```

```
}
```

OUTPUT

```
DFS Traversal: 0 1 3 2 4 %
```

4.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Edge {
public:
    int u, v, w;
};

bool cmp(Edge a, Edge b) {
    return a.w < b.w;
}

int findParent(int v, vector<int>& parent) {
    if (parent[v] == v)
        return v;
    return parent[v] = findParent(parent[v], parent);
}

void unionSet(int a, int b, vector<int>& parent) {
    a = findParent(a, parent);
    b = findParent(b, parent);
    parent[b] = a;
}

int main() {
    int n = 4, e = 5;

    vector<Edge> edges = {
        {0,1,10},
        {0,2,6},
        {0,3,5},
        {1,3,15},
        {2,3,4}
    };

    sort(edges.begin(), edges.end(), cmp);

    vector<int> parent(n);
    for (int i = 0; i < n; i++)
        parent[i] = i;
```

```
    cout << "Kruskal MST:\n";

    int cost = 0;
    for (auto ed : edges) {
        if (findParent(ed.u, parent) != findParent(ed.v, parent))
{
            cost += ed.w;
            cout << ed.u << " - " << ed.v << " (Weight: " << ed.w
<< ")\n";
            unionSet(ed.u, ed.v, parent);
        }
    }

    cout << "Total cost: " << cost << endl;

    return 0;
}
```

OUTPUT

```
Kruskal MST:
2 - 3 (Weight: 4)
0 - 3 (Weight: 5)
0 - 1 (Weight: 10)
Total cost: 19
```

5.

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int main() {
    int n = 5;

    int graph[5][5] = {
        {0,2,0,6,0},
        {2,0,3,8,5},
        {0,3,0,0,7},
        {6,8,0,0,9},
        {0,5,7,9,0}
    };

    vector<int> key(n, INT_MAX);
    vector<int> mst(n, 0);
    vector<int> parent(n, -1);

    key[0] = 0;

    for (int count = 0; count < n - 1; count++) {
```

```
        int u = -1;

        for (int i = 0; i < n; i++)
            if (!mst[i] && (u == -1 || key[i] < key[u]))
                u = i;

        mst[u] = 1;

        for (int v = 0; v < n; v++) {
            if (graph[u][v] && !mst[v] && graph[u][v] < key[v]) {
                key[v] = graph[u][v];
                parent[v] = u;
            }
        }
    }

    cout << "Prim MST:\n";
    int cost = 0;

    for (int i = 1; i < n; i++) {
        cout << parent[i] << " - " << i << " (Weight: " <<
graph[i][parent[i]] << ")\n";
        cost += graph[i][parent[i]];
    }

    cout << "Total cost: " << cost << endl;

    return 0;
}
```

OUTPUT

```
Prim MST:
0 - 1 (Weight: 2)
1 - 2 (Weight: 3)
0 - 3 (Weight: 6)
1 - 4 (Weight: 5)
Total cost: 16
```

6.

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int main() {
    int n = 5;

    int graph[5][5] = {
        {0,10,0,5,0},
        {10,0,1,2,0},
```

```
        {0,1,0,9,4},
        {5,2,9,0,2},
        {0,0,4,2,0}
    };

    vector<int> dist(n, INT_MAX);
    vector<int> visited(n, 0);

    dist[0] = 0;

    for (int i = 0; i < n - 1; i++) {
        int u = -1;

        for (int j = 0; j < n; j++)
            if (!visited[j] && (u == -1 || dist[j] < dist[u]))
                u = j;

        visited[u] = 1;

        for (int v = 0; v < n; v++) {
            if (graph[u][v] && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
        }
    }

    cout << "Dijkstra Distances from Node 0:\n";
    for (int i = 0; i < n; i++)
        cout << "0 -> " << i << " = " << dist[i] << endl;

    return 0;
}
```

OUTPUT

```
Dijkstra Distances from Node 0:
0 -> 0 = 0
0 -> 1 = 7
0 -> 2 = 8
0 -> 3 = 5
0 -> 4 = 7
```