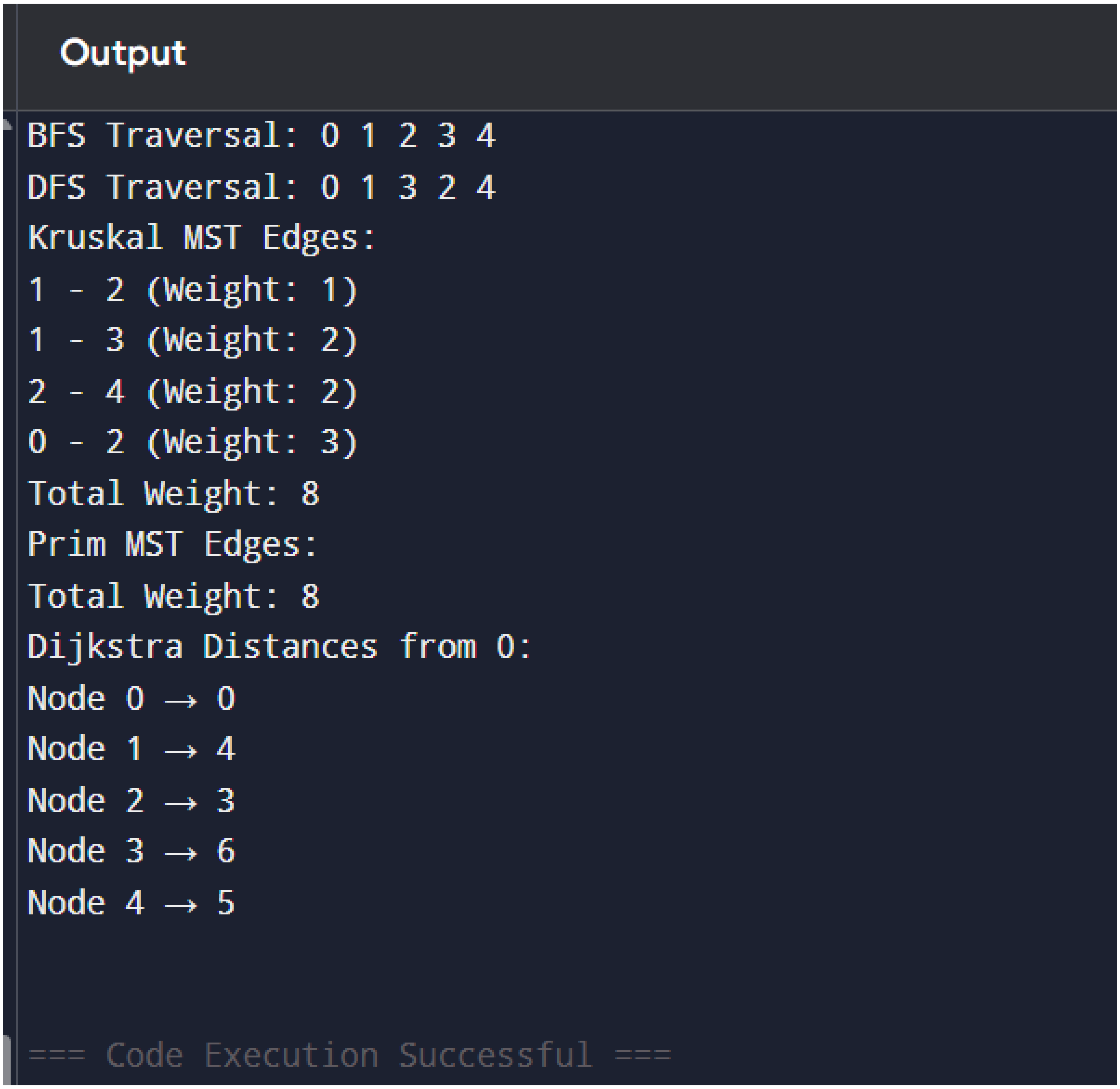
DSA ASSIGNMENT 9

QUESTION 1

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

void BFS(int start, vector<vector<int>>& adj, int n) {
  vector<bool> visited(n, false);
  queue<int> q;

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

  cout << "BFS Traversal: ";

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

    for (int next : adj[node]) {
      if (!visited[next]) {
        visited[next] = true;
        q.push(next);
      }
    }
    }
    cout << endl;
}

void DFSUtil(int node, vector<vector<int>>& adj, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (int next : adj[node]) {
        if (!visited[next])
            DFSUtil(next, adj, visited);
    }
}

void DFS(int start, vector<vector<int>>& adj, int n) {
    vector<bool> visited(n, false);
    cout << "DFS Traversal: ";
    DFSUtil(start, adj, visited);
    cout << endl;
}

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

void unionSet(int u, int v, vector<int>& parent, vector<int>& rank) {
    u = findParent(u, parent);
    v = findParent(v, parent);

    if (u != v) {
        if (rank[u] < rank[v]) parent[u] = v;
        else if (rank[v] < rank[u]) parent[v] = u;
        else {
            parent[v] = u;
            rank[u]++;
        }
    }
}

void Kruskal(int n, vector<vector<int>>& edges) {
    sort(edges.begin(), edges.end(), [](auto& a, auto& b){
        return a[2] < b[2];
    });

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

    cout << "Kruskal MST Edges:\n";
    int mstCost = 0;

    for (auto edge : edges) {
        int u = edge[0], v = edge[1], w = edge[2];

        if (findParent(u, parent) != findParent(v, parent)) {
            cout << u << " - " << v << " (Weight: " << w << ")\n";
            mstCost += w;
            unionSet(u, v, parent, rank);
        }
    }
    cout << "Total Weight: " << mstCost << "\n";
}

void Prim(int n, vector<vector<pair<int,int>>>& adj) {
    vector<int> key(n, INT_MAX);
    vector<bool> inMST(n, false);
    key[0] = 0;

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

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

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

        inMST[u] = true;
        mstCost += key[u];

        for (auto x : adj[u]) {
            int v = x.first;
            int w = x.second;

            if (!inMST[v] && w < key[v])
                key[v] = w;
        }
    }

    cout << "Total Weight: " << mstCost << "\n";
}

void Dijkstra(int start, int n, vector<vector<pair<int,int>>>& adj) {
    vector<int> dist(n, INT_MAX);
    dist[start] = 0;
```

```
        },

    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0, start});

    while (!pq.empty()) {
        int d = pq.top().first;
        int node = pq.top().second;
        pq.pop();

        if (d > dist[node]) continue;

        for (auto edge : adj[node]) {
            int next = edge.first;
            int w = edge.second;

            if (dist[node] + w < dist[next]) {
                dist[next] = dist[node] + w;
                pq.push({dist[next], next});
            }
        }
    }

    cout << "Dijkstra Distances from " << start << ":\n";
    for (int i = 0; i < n; i++)
        cout << "Node " << i << " → " << dist[i] << endl;
}
int main() {
    int n = 5;

    vector<vector<int>> adjList = {
        {1, 2},
        {0, 3},
        {0, 3, 4},
        {1, 2},
        {2}
    };

    BFS(0, adjList, n);
    DFS(0, adjList, n);

    vector<vector<int>> edges = {
        {0,1,4}, {0,2,3}, {1,2,1},
        {1,3,2}, {2,3,4}, {2,4,2}, {3,4,3}
    };
    Kruskal(5, edges);

    vector<vector<pair<int,int>>> adjWeighted = {
        {{1,4},{2,3}},
        {{0,4},{2,1},{3,2}},
        {{0,3},{1,1},{3,4},{4,2}},
        {{1,2},{2,4},{4,3}},
        {{2,2},{3,3}}
    };
    Prim(5, adjWeighted);

    Dijkstra(0, 5, adjWeighted);

    return 0;
}
```



```
BFS Traversal: 0 1 2 3 4
DFS Traversal: 0 1 3 2 4
Kruskal MST Edges:
1 - 2 (Weight: 1)
1 - 3 (Weight: 2)
2 - 4 (Weight: 2)
0 - 2 (Weight: 3)
Total Weight: 8
Prim MST Edges:
Total Weight: 8
Dijkstra Distances from 0:
Node 0 → 0
Node 1 → 4
Node 2 → 3
Node 3 → 6
Node 4 → 5



=== Code Execution Successful ===
```