```
//Name: 7. Graph: Minimum Spanning Tree
#include <iostream>
 #include <algorithm>
 #include <climits>
 using namespace std;
 const int MAX_VERTICES = 10;
struct Edge {
   int src, dest, weight;
 };
 class DisjointSet {
 public:
   int parent[MAX_VERTICES];
   int rank[MAX_VERTICES];
   DisjointSet(int n) {
      for (int i = 0; i < n; i++) {
         parent[i] = i;
         rank[i] = 0;
      }
   }
   int find(int u) {
      if (u != parent[u])
         parent[u] = find(parent[u]);
      return parent[u];
   }
```

```
    void unionSets(int u, int v) {
        int rootU = find(u);
        int rootV = find(v);
        if (rootU != rootV) {
            if (rank[rootU] > rank[rootV])
                parent[rootV] = rootU;
            else if (rank[rootU] < rank[rootV])
                parent[rootU] = rootV;
            else {
                parent[rootV] = rootU;
                rank[rootU]++;
            }
        }
    }
};
void kruskal(Edge edges[], int E, int V) {
    DisjointSet ds(V);
    Edge mst[MAX_VERTICES];
    int mstIndex = 0;
    sort(edges, edges + E, [](Edge a, Edge b) {
        return a.weight < b.weight;
    });
    for (int i = 0; i < E; i++) {
        Edge edge = edges[i];
```

```
        int u = edge.src;
        int v = edge.dest;
        if (ds.find(u) != ds.find(v)) {
            ds.unionSets(u, v);
            mst[mstIndex++] = edge;
        }
    }
    cout << "Kruskal's MST:\n";
    for (int i = 0; i < mstIndex; i++) {
        cout << char(mst[i].src + 'A') << " -- " << char(mst[i].dest + 'A') << " ==
" << mst[i].weight << endl;
    }
 }
void prim(int graph[MAX_VERTICES][MAX_VERTICES], int V) {
    int parent[MAX_VERTICES];
    int key[MAX_VERTICES];
    bool inMST[MAX_VERTICES];
    for (int i = 0; i < V; i++) {
        key[i] = INT_MAX;
        inMST[i] = false;
        parent[i] = -1;
    }
    key[0] = 0; // Star ng from the first vertex
    for (int count = 0; count < V - 1; count++) {
```

```
        int minKey = INT_MAX, minIndex;
        for (int v = 0; v < V; v++) {
            if (!inMST[v] && key[v] < minKey) {
                minKey = key[v];
                minIndex = v;
            }
        }
        inMST[minIndex] = true;
        for (int v = 0; v < V; v++) {
            if (graph[minIndex][v] && !inMST[v] && graph[minIndex][v] < key[v]) {
                parent[v] = minIndex;
                key[v] = graph[minIndex][v];
            }
        }
    }
    cout << "Prim's MST:\n";
    for (int i = 1; i < V; i++) {
        cout << char(parent[i] + 'A') << " -- " << char(i + 'A') << " == " <<
graph[i][parent[i]] << endl;
    }
 }
 int main() {
    const int V = 4;
    const int E = 5;
```

```
    Edge edges[E] = {
        {0, 1, 10},
        {0, 2, 6},
        {0, 3, 5},
        {1, 3, 15},
        {2, 3, 4}
    };
    int graph[MAX_VERTICES][MAX_VERTICES] = {0};
    for (int i = 0; i < E; i++) {
        graph[edges[i].src][edges[i].dest] = edges[i].weight;
        graph[edges[i].dest][edges[i].src] = edges[i].weight;
    }
kruskal(edges, E, V);
prim(graph, V);
return 0;
}
```