Represent a graph of your college campus using adjacency list/adjacency matrix. Nodes should represent the various departments/ institutes and links should represent the distance between them. Find minimum spanning tree
1) Using Kruskal's Algorithm
2) Using Prim's Algorithm.

Code

```
// Prim's algorithm
#include <iostream>
#include <vector>
#include <limits>

using namespace std;

const int INF = numeric_limits<int>::max();

void primMST(const vector<vector<int>>& graph) {
    int V = graph.size();
    vector<int> parent(V, -1); // Array to store the constructed MST
    vector<int> key(V, INF);    // Key values used to pick minimum weight edge
    vector<bool> inMST(V, false); // To represent if the vertex is included in MST

    // Start from the first vertex
    key[0] = 0; // Make key 0 so that this vertex is picked as the first vertex
    parent[0] = -1; // First node is always the root of MST

    for (int count = 0; count < V - 1; count++) {
        // Find the minimum key vertex from the set of vertices not yet included in MST
        int minKey = INF, minIndex;
        for (int v = 0; v < V; v++) {
            if (!inMST[v] && key[v] < minKey) {
                minKey = key[v];
                minIndex = v;
            }
        }
```

```
        // Add the picked vertex to the MST set
        inMST[minIndex] = true;

        // Update key values and parent index of the adjacent vertices of the picked
vertex
        for (int v = 0; v < V; v++) {
            // Update key only if graph[u][v] is smaller than key[v] and v is not in MST
            if (graph[minIndex][v] && !inMST[v] && graph[minIndex][v] < key[v]) {
                parent[v] = minIndex;
                key[v] = graph[minIndex][v];
            }
        }
    }

    // Print the constructed MST
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++) {
        cout << parent[i] << " - " << i << "\t" << graph[i][parent[i]] << " \n";
    }
}

int main() {
    // Example graph represented as an adjacency matrix
    vector<vector<int>> graph = {
        {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}
    };

    primMST(graph);

    return 0;
}
```

```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

Kruskal's Algorithm

Code

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Edge {
   int source, destination, weight;
};

// Comparator function to sort edges based on their weight
bool compareEdges(const Edge &a, const Edge &b) {
   return a.weight < b.weight;
}

// Disjoint Set Union (Union-Find) data structure
class DisjointSet {
public:
   DisjointSet(int n) {
      parent.resize(n);
      rank.resize(n, 0);
      for (int i = 0; i < n; i++)
         parent[i] = i;
   }
```

```
    int find(int u) {
        if (u != parent[u])
            parent[u] = find(parent[u]); // Path compression
        return parent[u];
    }

    void unionSet(int u, int v) {
        int rootU = find(u);
        int rootV = find(v);

        if (rootU != rootV) {
            // Union by rank
            if (rank[rootU] < rank[rootV]) {
                parent[rootU] = rootV;
            } else if (rank[rootU] > rank[rootV]) {
                parent[rootV] = rootU;
            } else {
                parent[rootV] = rootU;
                rank[rootU]++;
            }
        }
    }

private:
    vector<int> parent, rank;
};

void kruskalMST(const vector<Edge> &edges, int V) {
    DisjointSet ds(V);
    vector<Edge> result; // To store the resultant MST

    // Sort all edges based on their weight
    vector<Edge> sortedEdges = edges;
    sort(sortedEdges.begin(), sortedEdges.end(), compareEdges);

    for (const Edge &edge : sortedEdges) {
```

```
        int u = edge.source;
        int v = edge.destination;

        // Check if including this edge would cause a cycle
        if (ds.find(u) != ds.find(v)) {
            ds.unionSet(u, v);
            result.push_back(edge);
        }
    }

    // Print the constructed MST
    cout << "Edge \tWeight\n";
    for (const Edge &edge : result) {
        cout << edge.source << " - " << edge.destination << "\t" << edge.weight << " \n";
    }
}

int main() {
    // Example graph represented as a list of edges
    vector<Edge> edges = {
        {0, 1, 10},
        {0, 2, 6},
        {0, 3, 5},
        {1, 3, 15},
        {2, 3, 4}
    };

    int V = 4; // Number of vertices (0 to 3)
    kruskalMST(edges, V);

    return 0;
}
```

```
Edge    Weight
2 - 3   4
0 - 3   5
0 - 1   10
```