# 

# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BSCS 13**

**Name: Muhammad Haadhee sheeraz**

**REG NO: 478359**

# Lab 9: Minimum Spanning Tree

**Date: 24th July, 2024**

**Time: 1:00pm – 4:50pm**

# Course Instructor: Dr. Farzana Jabeen

# Lab 9: Implementation of Kruskal's and Prim's Algorithms to compute MST

**Introduction:**

This lab is based on graphs data structure. You should learn how to implement the Prim’s algorithm and Kruskal’s Algorithm to compute the minimum spanning tree (MST) of an undirected weighted graph.

**Objectives**

The objectives of this lab are the following:

* Understand the concept of Minimum Spanning Trees (MSTs).
* Implement algorithms like Kruskal's and Prim's to find MSTs.
* Analyze the time and space complexity of these algorithms.

**Tools/Software Requirement**

Visual Studio 2012 or gcc or g++ / Code Spaces

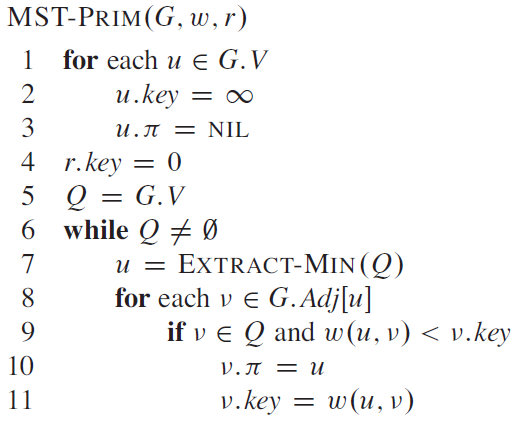
**Description**

A **Minimum Spanning Tree (MST)** of an undirected, weighted graph is a subgraph that:

1. **Connects all the vertices** in the graph.
2. Contains **exactly V−1V-1V−1 edges**, where VVV is the number of vertices.
3. Has the **minimum possible total edge weight** compared to other spanning trees.

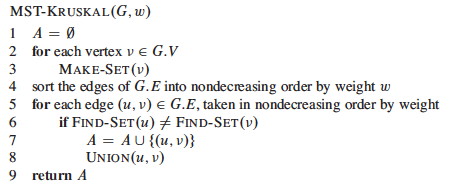
**Prim’s Algorithm**

Prim's Algorithm is a **greedy algorithm** that builds the MST by adding edges one at a time, always selecting the smallest edge that connects a new vertex to the growing tree.



**Kruskal's Algorithm**

Kruskal's Algorithm is a **greedy algorithm** that builds the MST by sorting all edges in increasing order of weight and adding them to the MST if they do not form a cycle.



**Tasks**

**Task 1: Implementation of Algorithms**

1. Kruskal's Algorithm Implementation:
   * Implement a function to sort edges by weight.
   * Implement a disjoint-set data structure to track connected components.
   * Implement the Kruskal's algorithm to find the MST.
2. Prim's Algorithm Implementation:
   * Implement a priority queue to store edges.
   * Implement Prim's algorithm to find the MST.

**Task 2: Analysis and Experimentation**

1. Time and Space Complexity:
   * Analyze the time and space complexity of Kruskal's and Prim's algorithms.
   * Compare the performance of Kruskal's and Prim's algorithms on different graph sizes and densities.

The next step is to compute the time complexity of both algorithms. Does the running time of algorithm shows variations based on the structure of the input and why? Plot the running time of the best and worst case complexities for different input sizes in a excel sheet and add it in the solution section.

**Solution**

|  |
| --- |
| Solution |
| Task 1 Code: #include <iostream>  #include <vector>  #include <algorithm>  #include <queue>  using namespace std;  struct Edge {      int u, v, weight;      bool operator<(const Edge& e) const {          return weight < e.weight;      }  };  class DisjointSet {  private:      vector<int> parent, rank;  public:      DisjointSet(int n) {          parent.resize(n);          rank.resize(n, 0);          for (int i = 0; i < n; i++) parent[i] = i;      }      int find(int x) {          if (parent[x] != x)              parent[x] = find(parent[x]);          return parent[x];      }      void unite(int x, int y) {          int rx = find(x);          int ry = find(y);          if (rx != ry) {              if (rank[rx] < rank[ry]) parent[rx] = ry;              else if (rank[rx] > rank[ry]) parent[ry] = rx;              else {                  parent[ry] = rx;                  rank[rx]++;              }          }      }  };  // Kruskal's Algorithm  vector<Edge> kruskal(int n, vector<Edge>& edges) {      sort(edges.begin(), edges.end());      DisjointSet ds(n);      vector<Edge> mst;      for (Edge& e : edges) {          if (ds.find(e.u) != ds.find(e.v)) {              mst.push\_back(e);              ds.unite(e.u, e.v);          }      }      return mst;  }  // Prim's Algorithm  vector<Edge> prim(int n, vector<vector<pair<int, int>>>& adj) {      vector<bool> visited(n, false);      priority\_queue<pair<int, pair<int, int>>, vector<pair<int, pair<int, int>>>, greater<>> pq;      vector<Edge> mst;      pq.push({0, {0, -1}}); // weight, {to, from}      while (!pq.empty()) {          auto [w, p] = pq.top(); pq.pop();          int u = p.first, parent = p.second;          if (visited[u]) continue;          visited[u] = true;          if (parent != -1) mst.push\_back({parent, u, w});          for (auto& [v, weight] : adj[u]) {              if (!visited[v]) {                  pq.push({weight, {v, u}});              }          }      }      return mst;  }  // Complexity Analysis (for reference)  /\*  Kruskal:  - Time: O(E log E) due to sorting + union-find ops (almost constant)  - Space: O(V + E)  Prim:  - Time: O((V + E) log V) using priority queue  - Space: O(V + E)  \*/  int main() {      int n = 5; // Number of vertices      vector<Edge> edges = {          {0, 1, 10}, {0, 2, 1}, {0, 3, 4},          {1, 2, 3}, {2, 3, 2}, {1, 4, 0},          {3, 4, 7}      };      vector<Edge> mstK = kruskal(n, edges);      cout << "Kruskal's MST:\n";      for (Edge& e : mstK) cout << e.u << " - " << e.v << " : " << e.weight << endl;      vector<vector<pair<int, int>>> adj(n);      for (Edge& e : edges) {          adj[e.u].push\_back({e.v, e.weight});          adj[e.v].push\_back({e.u, e.weight});      }      vector<Edge> mstP = prim(n, adj);      cout << "\nPrim's MST:\n";      for (Edge& e : mstP) cout << e.u << " - " << e.v << " : " << e.weight << endl;      return 0;  }  Task 1 Output Screenshot:    Task 2 Code, graph, time comparison table:  // Minimum Spanning Tree Algorithms: Kruskal's and Prim's  #include <iostream>  #include <vector>  #include <algorithm>  #include <queue>  #include <ctime>  #include <cstdlib>  using namespace std;  struct Edge {      int u, v, weight;      bool operator<(const Edge& other) const {          return weight < other.weight;      }  };  class DisjointSet {  public:      vector<int> parent, rank;      DisjointSet(int n) {          parent.resize(n);          rank.resize(n, 0);          for (int i = 0; i < n; i++) parent[i] = i;      }      int find(int x) {          if (parent[x] != x)              parent[x] = find(parent[x]);          return parent[x];      }      void unite(int x, int y) {          int xroot = find(x);          int yroot = find(y);          if (xroot == yroot) return;          if (rank[xroot] < rank[yroot]) parent[xroot] = yroot;          else if (rank[xroot] > rank[yroot]) parent[yroot] = xroot;          else { parent[yroot] = xroot; rank[xroot]++; }      }  };  int kruskalMST(int n, vector<Edge>& edges) {      sort(edges.begin(), edges.end());      DisjointSet ds(n);      int mstWeight = 0;      for (Edge& e : edges) {          if (ds.find(e.u) != ds.find(e.v)) {              ds.unite(e.u, e.v);              mstWeight += e.weight;          }      }      return mstWeight;  }  int primMST(int n, vector<vector<pair<int, int>>>& adj) {      priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;      vector<bool> visited(n, false);      pq.push({0, 0});      int mstWeight = 0;      while (!pq.empty()) {          auto [weight, u] = pq.top(); pq.pop();          if (visited[u]) continue;          visited[u] = true;          mstWeight += weight;          for (auto& [v, w] : adj[u]) {              if (!visited[v]) pq.push({w, v});          }      }      return mstWeight;  }  vector<Edge> generateEdges(int n, int density) {      vector<Edge> edges;      for (int i = 0; i < n; i++) {          for (int j = i + 1; j < n; j++) {              if (rand() % 100 < density) {                  int w = rand() % 100 + 1;                  edges.push\_back({i, j, w});              }          }      }      return edges;  }  vector<vector<pair<int, int>>> buildAdjList(int n, const vector<Edge>& edges) {      vector<vector<pair<int, int>>> adj(n);      for (const Edge& e : edges) {          adj[e.u].push\_back({e.v, e.weight});          adj[e.v].push\_back({e.u, e.weight});      }      return adj;  }  int main() {      srand(time(0));      vector<int> sizes = {10, 50, 100, 200, 500};      vector<int> densities = {20, 40, 60, 80, 100};      cout << "Nodes\tEdges\tDensity\tKruskal(ms)\tPrim(ms)" << endl;      for (int i = 0; i < sizes.size(); i++) {          int n = sizes[i];          int density = densities[i];          vector<Edge> edges = generateEdges(n, density);          vector<vector<pair<int, int>>> adj = buildAdjList(n, edges);          clock\_t start = clock();          int w1 = kruskalMST(n, edges);          clock\_t end = clock();          double kruskalTime = 1000.0 \* (end - start) / CLOCKS\_PER\_SEC;          start = clock();          int w2 = primMST(n, adj);          end = clock();          double primTime = 1000.0 \* (end - start) / CLOCKS\_PER\_SEC;          cout << n << "\t" << edges.size() << "\t" << density << "%\t"               << kruskalTime << "\t\t" << primTime << endl;      }      return 0;  }    Task 2 Output Screenshot: |

### **Deliverables**

Compile a single word document by filling in the solution part and submit this Word file on LMS. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS.