



## Lab 10

# Minimum Spanning Tree

Quang D. C.  
dungcamquang@tdtu.edu.vn

November 20, 2023

### Note

In the previous lab, we learned graph and how to traverse it. In this lab, we will practice more exercises on graph and learn a new problem on graph, Minimum Spanning Tree. To find the Minimum Spanning Tree, we have two basic algorithms:

1. Prim's algorithm
2. Kruskal's algorithm

## Part I

# Classwork

*In this part, lecturer will:*

- Summarize the theory related to this lab.
- Instruct the lesson in this lab to the students.
- Explain the sample implementations.

*Responsibility of the students in this part:*

- Students practice sample exercises with solutions.
- During these part, students may ask any question that they don't understand or make mistakes. Lecturers can guide students, or do general guidance for the whole class if the errors are common.

## 1. Minimum Spanning Tree

Tree **T** is a connected graph that has  $V$  vertices and  $V-1$  edges, only one unique path between any two pair of vertices in **T**.

Given connected graph **G** with positive edge weights, find a min weight set of edges that connects all of the vertices.

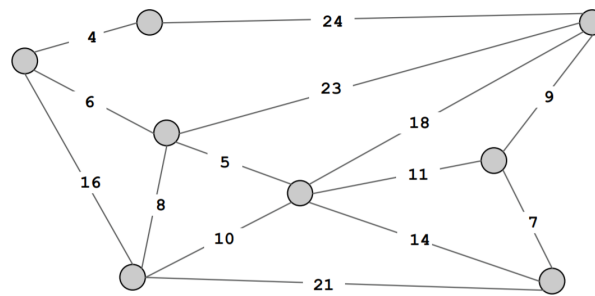
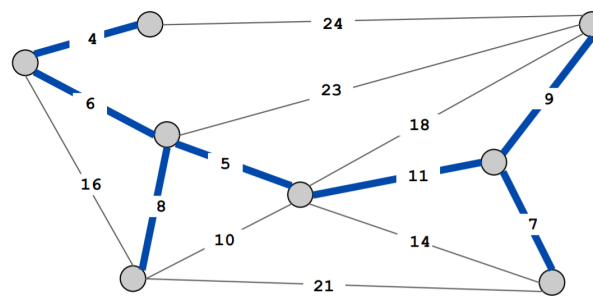


Figure 1: Graph

A spanning tree of a graph  $G$  is a subgraph  $T$  that is connected and acyclic.



$$\text{cost}(T) = 50$$

Figure 2: Minimum Spanning Tree

## 2. Prim's Algorithm

The way Prim's algorithm works is as follows :

- Initialize the minimum spanning tree with a initial vertex.
- Find all the edges that connect the tree to new vertices, find the minimum, and add it to the tree.
- Keep repeating step 2 until we get a minimum spanning tree (until all vertices are reached).

You can follow this instruction to step by step implement Prim algorithm:

1. Create a boolean array (`visited[]`) and two integer arrays (`parent[]`, `cost[]`)
2. Choose the initial vertex.
3. Find the vertex which has the minimum cost from the latest vertex and hasn't been visited (we can call this vertex is `min_id`).
4. Set `visited[min_id] = true`
5. Update `parent[]` from `min_id` to the vertices which haven't visited. Update `cost[]` if the cost from new vertex is smaller than from the previous.
6. Repeat step 3 until all vertices are visited.

### 3. Kruskal's Algorithm

To simplify Kruskal's implementation, we will use **Edge List** to store the graph. The steps for implementing Kruskal's algorithm are as follows:

1. Sort all the edges from low weight to high.
2. Take the edge with lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

To check the edge created a cycle or not, you can use the data structure called **Union-Find Disjoint Sets**.

This is Union-Find Disjoint Sets implementation, you can union the vertices and use `isSameSet(int i, int j)` method to check two vertices `i, j` create a cycle or not.

```
1  class UnionFind
2  {
3      private Vector<Integer> p, rank, setSize;
4
5      public UnionFind(int N) {
6          p = new Vector<Integer>(N);
7          rank = new Vector<Integer>(N);
8          setSize = new Vector<Integer>(N);
9          for (int i = 0; i < N; i++) {
10             p.add(i);
11             rank.add(0);
12             setSize.add(1);
13         }
14     }
15
16     public int findSet(int i) {
17         if (p.get(i) == i)
18             return i;
19         else {
20             int ret = findSet(p.get(i));
21             p.set(i, ret);
22             return ret;
23         }
24     }
25
26     public void unionSet(int i, int j) {
27         if (!isSameSet(i, j)) {
28             int x = findSet(i), y = findSet(j);
29             if (rank.get(x) > rank.get(y)) {
30                 p.set(y, x);
31                 setSize.set(x, setSize.get(x) + setSize.get(y));
32             }
33             else {
34                 p.set(x, y);
35                 setSize.set(y, setSize.get(y) + setSize.get(x));
36                 if (rank.get(x) == rank.get(y))
37                     rank.set(y, rank.get(y) + 1);
38             }
39         }
```

```
40     }  
41  
42     public boolean isSameSet(int i, int j){  
43         return findSet(i) == findSet(j);  
44     }  
45 }
```

## Part II

# Exercise

*Responsibility of the students in this part:*

- Complete all the exercises with the knowledge from **Part I**.
- Ask your lecturer if you have any question.
- Submit your solutions according to your lecturer requirement.

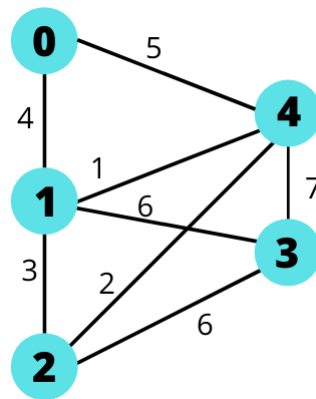
### Exercise 1

Download the **"Social circles: Facebook"** dataset: [Click here to download the dataset](#) and do the following requirements:

- (a) Construct the graph in Edge List format.
- (b) Find the number of vertices.
- (c) Find the number of edges.
- (d) Count the components of graph. (*Hint: You can convert to Adjacency List format and apply the graph traversal algorithm to solve this problem*).
- (e) Find the user that has the most friends (the highest degree in the graph) and return the label and number of friends of this user.
- (f) Find the list of mutual friends between user **u** and user **v**. (*Hint: You can convert to Adjacency List format to simplify this problem*).

### Exercise 2

Given a graph:



Read this graph from a text file with Adjacency Matrix.

- (a) Write the Kruskal function for the given graph. Print MST result on the screen.
- (b) Write the Prim function for the given graph starting from vertex 0. Print MST result on the screen.

#### 4. Reference

1. <https://www.cs.princeton.edu/courses/archive/spr07/cos226/lectures/mst.pdf>
2. <https://www.journaldev.com/43746/prims-algorithm-minimum-spanning-tree-java>

THE END