# California State University, Fresno

# DEPARTMENT OF COMPUTER SCIENCE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **Algorithms & Data Structures** | | | Semester: | **Spring 2022** |
|  | | | | | |
| Points |  | Document author: | **MingkuanPang** | | |
|  | Author’s email: | **Yafking20 @mail.fresnostate.edu. email** | | |
| Laboratory number: | **11 – Kruskal\_Prim** | | |
|  | | | | | |

**1. Statement of Objectives**

This lab asks for implementation of Kruskal’s Algorithm and Prim Algorithm. Those algorithms are used to generate minimum spanning tree.

**2. Experimental Procedure**

**Data Structure introduction:**

**Edge:**

A screenshot of a computer

Description automatically generated with medium confidence

Vtx is a pair that contains two vitexes that connected by the edge, and weight is the value of the edge.

**Graph:**

Text

Description automatically generated

The vertexes represents the vertex of the graph, and edges represents the edges between two vertexes, and adjacency represents the adjacencies for each vertex.

**exist\_e:**

Text

Description automatically generated

Use to determine whether there is a edge between those two input nodes. If there is then return true, otherwise return false.

**Exist\_v():**

Text

Description automatically generated

Use to determine whether there is such vertex in the graph, if yes then return true, return false otherwise.

**Insert\_adjacency()**

Text

Description automatically generated

Add adjacency to the input vertex, if there is not such vertex in the graph then exit the program otherwise add the adjacency to the vertex and also add weight to the edge.

**Get\_weight():**

Graphical user interface, text

Description automatically generated

Get the weight of the edge between two vertexes. Return -1 if there is not such vertex in the graph, otherwise return the weight of the edge.

**Remove()：**

Text

Description automatically generated

Remove an element from vector, first switch the element with the last element of the vector, then use pop () function to remove the element.

**Exist():**

Text

Description automatically generated

Check if the element is in the vector, return true if it is , otherwise return false.

**Get\_min()**

Text

Description automatically generated

Return the index of minimum element in the vector, after searching the minimum index, set its value to be INT\_MAX.

**My\_sort()**

Text

Description automatically generated

Sort the edges by its weight in ascending order.

**M\_stoi()**

Text

Description automatically generated

Convert a string into a vector.

**Make\_v()**

Text

Description automatically generated

Create a vector contains numbered in range from 1 to size. For example, if the size is 5 then it will create an array like [1,2,3,4,5].

**printG()**

Text

Description automatically generated

Print the adjacency list for each vertex.

**Find()**

Text

Description automatically generated

Returns the parents value of the input vertex.

**Unite ()**

A screenshot of a computer

Description automatically generated with medium confidence

For those two vertexes u and v. Make the parents of v to be u. Then search the parents list, for those vertexes whose parents is the same as v, change it to u.

**KruskalMST()**

Text

Description automatically generated

Return a minimum spanning tree generated by Kruskal’s algorithm. First sort all the edges in the graph by their weights by ascending order. Then start from the minimum edge to the maximum edge one by one. If adding the edge will not create a circle in the graph, then add it to the minimum spanning tree, otherwise skip the edge. Until every vertex is connected.

**Prim\_mst()**

Text

Description automatically generated

A screenshot of a computer

Description automatically generated

Return a minimum spanning tree generated by Prim’s algorithm. Travel through the graph from the start vertex. Then from its adjacencies, choose the one that makes the cost of the minimum spanning tree increase least. Then continue with that one to be added until all the vertexes be process.

**3. Analysis**

**Main function**

Text

Description automatically generated

It will ask user to create a graph by inputting the vertexes in the graph and the adjacencies for the vertex and also the weight. For example:

Like the graph in the picture:A picture containing text, clock, clipart

Description automatically generated

It will be:

Text

Description automatically generated

After entering the graph, it will ask user to input the starting node



After that it will print the original graph then will print the minimum spanning tree generated by Prim algorithm and Kruskal algorithm and also the cost of the minimum spanning tree for each algorithm:

Text

Description automatically generated

A picture containing text, clock, clipart

Description automatically generated

A picture containing clipart

Description automatically generated

After that, it will ask user if want to continue the program, enter ‘n’ for quit, anything else will continue the program.

**Output:**

Text

Description automatically generated

**4. Encountered Problems**

To be honest, when I tried to implement those two algorithms by taking the graph as a matrix of adjacencies, my program just will not work for some reason, there is not debug error, just the result of minimum spanning tree is not correct. I can not figure it out why, so I made the data structure for the graph and then let user to define their own graph.

**5. Conclusions**

This experiment gave me a deeper understanding of Kruskal's Algorithm and Prim's Algorithm. Also made me handier in the simplification and optimization of data results like graph. These two algorithms have also been widely applied in practice, which are very important and must be known.

**6. References**

I did not use any reference in this lab.