# Compute the Minimum Spanning Tree using Prim's Algorithm

## Objectives

* Extend your weighted graph into a undirected graph.
* Compute the minimum spanning tree of an undirected graph.

## Overview

This assignment focuses on Undirected Graphs. In this assignment, a telecommunications company "BT&T" wants to build a telecommunications grid across the United States. Since the company is new, they don't have a lot of capital, but they do want several major cities in the country connected. You are a new hire for the company and it's your job to figure out the minimum amount of infrastructure that needs to be built in order to solve this problem.

Note: Parts of this assignment are going to be difficult.

## Part 1: Update WeightedGraph

Update your WeightedGraph class.

We have to make a few changes to WeightedGraph. Set both set methods to be virtual. That's it. Make that small change and you are good to go for Part 2.

virtual void set(int, int, int); // Sets index i,j to weight  
virtual void set(int, int); // Calls "set(i,j,weight)" with weight=1

## Part 2: Write UndirectedGraph

Build an UndirectedGraph class which extends your WeightedGraph class for an directed graph. Here's my header file undirectedgraph.h. There are no private data members and the constructor is empty. This looks shorter, but will be significantly harder than anything you've been asked to write this semester.

#pragma once  
#include "weightedgraph.h"  
class UndirectedGraph : public WeightedGraph  
{  
public:  
 UndirectedGraph(int mVertices); // Call the base constructor with mVertices  
 // but otherwise is empty.  
  
 void set(int, int, int); // Sets index i,j to weight AND index j,i to weight  
 // First param: i  
 // Second param: j  
 // Third param: weight  
 void set(int, int); // Calls set(i, j, 1)  
  
 int getSumEdgeWeights(); // Comutes the sum of the edge weights  
 // for only the upper triangle and the diagonal.  
 // This will require a fancy for-loop.  
 UndirectedGraph prim(); // The big one. See below.  
};

## Part 3: Write Prim's Algorithm

This is a difficult algorithm to write. Hopefully this pseudocode will make things easier.

* Create a blank graph named mst with a matching number of vertices to this graph.
* Set a starting vertex to be 0.
* Create an integer array of length Number of Vertices named distance.
* Create an integer array of length Number of Vertices named adjacent.
* Create a Boolean array of length Number of Vertices named available.
* Set all of distance to be INT MAX.
* Set all of adjacent to be -1.
* Set all of available to be true.
* Set distance[start] to be 0.
* Create an integer availableCount set to Number of Vertices.
* while availableCount greater than 0
  + Find the index position of the smallest element in distance where the matching index of available is also true. This index position is named u.
  + Set available[u] to be false.
  + Decrement availableCount by 1.
  + If u is not equal to start, update mst at (u, adjacent[u]) to be equal to this graph's weight at (u, adjacent[u]).
  + For each vertex v.
    - If the weight at (u,v) is greater than 0 and less than distance[v]...
      * Update distance[v] to be the weight at (u,v).
      * Update adjacent[v] to be u.
* Delete all of your dynamically created arrays.
* Return mst

## Part 4. Write the Program.cpp file.

Ask the user for a number of vertices and edges, then create a graph based on those dimensions. Ask the user to read the starting vertex, ending vertex, and edge weight for each edge defined. Afterward, print the graph, the sum of the edge weights, the minimum spanning tree, and the sum of the edge weights for the minimum spanning tree.

Review these sample input and output descriptions.

This is the simplest example. Edge ab is 4. Edge bc is 5. Edge ac is 6. Prim's algorithm should return edges ab and bc for a total edge weight of 9.

Number of vertices in our matrix: 3  
Number of edges in our matrix: 3  
Enter a starting vertex: 0  
Enter an ending vertex: 1  
Enter a weight: 4  
Enter a starting vertex: 0  
Enter an ending vertex: 2  
Enter a weight: 6  
Enter a starting vertex: 1  
Enter an ending vertex: 2  
Enter a weight: 5  
  
Original Graph.  
 0 4 6  
 4 0 5  
 6 5 0  
Sum of Edge Weights: 15  
Minimum Spanning Graph.  
 0 4 0  
 4 0 5  
 0 5 0  
Sum of Edge Weights in MST: 9

This is a run of the graph that we used many times in class. It's a difficult example, but there's only one solution.

Number of vertices in our matrix: 6  
Number of edges in our matrix: 8  
Enter a starting vertex: 0  
Enter an ending vertex: 1  
Enter a weight: 3  
Enter a starting vertex: 0  
Enter an ending vertex: 2  
Enter a weight: 2  
Enter a starting vertex: 1  
Enter an ending vertex: 3  
Enter a weight: 5  
Enter a starting vertex: 1  
Enter an ending vertex: 4  
Enter a weight: 1  
Enter a starting vertex: 2  
Enter an ending vertex: 3  
Enter a weight: 2  
Enter a starting vertex: 2  
Enter an ending vertex: 4  
Enter a weight: 4  
Enter a starting vertex: 3  
Enter an ending vertex: 5  
Enter a weight: 1  
Enter a starting vertex: 4  
Enter an ending vertex: 5  
Enter a weight: 5  
  
Original Graph.  
 0 3 2 0 0 0  
 3 0 0 5 1 0  
 2 0 0 2 4 0  
 0 5 2 0 0 1  
 0 1 4 0 0 5  
 0 0 0 1 5 0  
Sum of Edge Weights: 23  
Minimum Spanning Graph.  
 0 3 2 0 0 0  
 3 0 0 0 1 0  
 2 0 0 2 0 0  
 0 0 2 0 0 1  
 0 1 0 0 0 0  
 0 0 0 1 0 0  
Sum of Edge Weights in MST: 9

## Part 5.

Final Test. Our company "BT&T" wants the following cities in the country connected. There are 15 cities with 73 edges. Each triplet of values represents, in order, city A, city B, and the cost of connecting the two cities. You should be able to copy and paste this into your program.

15  
73  
13 12 5  
12 1 19  
12 4 16  
7 10 18  
8 4 13  
3 5 2  
11 9 17  
2 12 15  
5 3 20  
13 11 16  
13 7 12  
14 4 1  
3 8 11  
0 3 6  
9 6 1  
11 1 14  
10 14 9  
12 0 11  
12 14 15  
12 10 10  
6 7 3  
8 9 19  
0 7 4  
9 10 20  
13 0 13  
5 13 2  
3 7 4  
8 14 14  
1 3 10  
3 6 16  
6 4 11  
7 1 11  
7 6 17  
2 7 17  
0 1 4  
7 13 18  
14 8 13  
5 11 7  
6 2 6  
2 11 19  
9 2 7  
5 2 8  
4 12 14  
2 3 19  
1 4 14  
12 11 5  
3 13 7  
3 14 17  
10 1 2  
11 5 11  
6 13 15  
5 0 6  
2 0 6  
1 10 20  
14 12 3  
3 10 14  
12 9 20  
3 9 10  
11 12 6  
9 3 16  
2 4 4  
8 0 16  
4 5 17  
8 13 7  
12 3 2  
8 1 8  
3 4 5  
10 7 10  
2 5 8  
3 0 17  
4 2 4  
14 10 6  
4 1 9

If you plug this graph into your minimum spanning tree algorithm and come up with the minimum sum of the edges, email that magic number to me. I'll replace any previous homework score with a 100.

## Files

Here are the files that should go into your final submission.

Main.cpp  
WeightedGraph.cpp  
WeightedGraph.h  
UndirectedGraph.cpp  
UndirectedGraph.h