CSCE 221 Cover Page
Programming Assignment #6
Due December 6th at midnight to CSNet

First Name Raymond Last Name Zhu UIN 923008555

User Name rawrbyte E-mail address rawrbyte@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero. According to the University Regulations, Section 42, scholastic dishonesty are including: acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion read more: Aggie Honor System Office

Type of sources			
People	Peer Teachers		
Web pages (provide URL)	Stackoverflow		
Printed material	textbook		
Other Sources			

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

"On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work."

Your Name Raymond Zhu Date December 6th, 2015

1. Program Description

   (a) The purpose of this assignment is mainly to take in input values and create a graph, and utilize our knowledge of the graph and disjoint set data structures from the previous lab and the MST algorithm to create an efficient program to graph a miminum spanning tree with implementation of kruskal's algorithm.

2. Data Structures Description

   (a) The assignment uses kruskal's algorithm which is a greedy algorithm that finds the lowest weighted edges of a graph and find a shortest path from those edges. The thing that stands out from kruskal's algorithm is how the algorithm is structured to where the nodes do not expand similar to dijkstra's or primm's algorithm rather it builds a shortest path from the smallest edges.

3. Alogorithm Description

   - void buildGraph() O(n): creates an empty adjacency list and edge list
   - double insertEdge(int i, int j, double w) O(log(n)): creates an edge between two nodes if the edge does not exist
   - double getWeight(int i, int j) O(log(n)): traverses through the edge list to find the edge with an i and j
   - void sortEdge() O(nlog(n)): uses STL's vector sort and a less than operator to sort the edge list
   - double MSTAlgo() O(nlog(n)): implementations of kruskal's algorithm. Creates a set for every node, sorts the edges in the edge list, and unions the smallest sets to create a minimum spanning tree. It counts and returns the total weight of the minimum spanning tree as its cost.

4. Program Organization and Description of Classes

```
disjointset.h
#pragma once

#include "TemplateDoublyLinkedList.h"
#include <cstddef>
#include <iostream>
#include <vector>

using namespace std;

// Disjoint Set
template <typename T>
class DisjointSet {
private:
```

```
        vector<DListNode<T>*> nodeLocator;
public:
        ~DisjointSet(){
                for(int i = 0; i < nodeLocator.size(); ++i){
                        if(nodeLocator[i] != NULL){
                                while(nodeLocator[i]->getNext
                                    () != NULL){
                                        nodeLocator[i]->
                                            delete_after();
                                }
                                delete nodeLocator[i];
                        }
                }
        }
        DisjointSet(int n){}
        vector<DListNode<T>*> getNodeLocator() const{return
            nodeLocator;}
        DListNode<T>* MakeSet(int key, T node);
        DListNode<T>* Union(DListNode<T> nodeI, DListNode<T>
            nodeJ);
        DListNode<T>* FindSet(DListNode<T> node);
        DListNode<T>* FindSet(int nodeKey);
};

templatedoublylinkedlist.h
#pragma once

#include <string>
#include <cstdlib>
#include <iostream>
#include <stdexcept>
using namespace std;

// list node
template <typename T>
class DListNode {
private:
  int key;
  int listSize;
  T obj;
  DListNode *prev, *next, *representative;
  DListNode *trailer; //just the representative node has this
      pointer assigned
public:
  DListNode(int k, T e= T(), DListNode *p = NULL, DListNode *n
       = NULL)
    : key(k), obj(e), prev(p), next(n) { listSize = 1; trailer
        = this; representative = this;}
  T getElem() const { return obj; }
  T& getElemt() { return obj; }
  DListNode<T> * getNext() const { return next; }
  DListNode<T> * getPrev() const { return prev; }
  void setNext(DListNode* n) { this->next = n; }
  void setPrevious(DListNode* p) { this->prev = p; }
  DListNode<T>* insert_before(T d); // insert the int before
      this node
  // return a pointer to the inserted node
```

```
  DListNode<T>* insert_after(T d); // insert the int after
      this node
  // return a pointer to the inserted node
  void delete_before(); // delete the node before this node
  void delete_after(); // delete the node after this node
  int getKey() { return key; }
  DListNode<T>* getRepresentative() const { return
      representative; }
  DListNode<T>* getTrailer() const { return trailer; }
  void setRepresentative(DListNode* rep);
  void setTrailer(DListNode* trail);
  int getListSize() { return listSize; }
  void setListSize(int lSize) { this->listSize = lSize; }
};
```

5. Instructions to Compile and Run your Program
   - type "make" to makeall
   - and then
   - ./main test1.mat to execute the program with a file
6. Logical Exceptions
   (a) No logical error has been found in testing from the program itself
7. C++ object orientated or generic programming features
   (a) Generic programming with templates
8. Tests

```
Part 1
[rawrbyte]@sun ~/CSCE221/lab6/Part1> (16:21:06 12/06/15)
:: ./main test1.mat
The Adjacency Matrix of the Graph is:
    0    9    3    5
    9    0    0    2
    3    0    0    0
    5    2    0    0

Part 2
[rawrbyte]@sun ~/CSCE221/lab6/Part2> (16:21:40 12/06/15)
:: ./main test1.mat
The Adjacency Matrix of the Graph is:
    0    9    3    5
    9    0    0    2
    3    0    0    0
    5    2    0    0
The total value of the Minimum Spanning Tree is: 10
The Minimum Spanning Tree is:
Node  Node  Weight
```

1	3	2
0	2	3
0	3	5