ICS 46
Homework Assignment 8

SUMMARY
In this assignment, you will be implementing **Kruskal's algorithm** and a test driver for your program. **It is in your best interest to start early! Read this document carefully before starting.**

DEVELOPMENT
It is essential that you develop your program incrementally. Test each data structure and function as you complete it.

REFERENCES
These are recommended resources to help you. **Do not blindly copy and paste any code.** Use these references to guide your development process.

- Minimum Spanning Tree lecture slides
- Union Find Set Implementation files
- Kruskal's Algorithm Wikipedia entry

DOWNLOADS
Add these to your assignment directory for testing.

- Small test case: small.graph
- Large (standard) test case: large.graph
- Random Graph Generator: genGraph.cpp

PART 1: DATA STRUCTURES

To implement Kruskal's algorithm, you will need to implement a few data structures:

1) Vertex
2) Edge
3) Graph

You may add getter methods where necessary. You may add any private helpers that are necessary. Remember to test your code incrementally to ensure correctness and save yourself time!

VERTEX (CLASS)

Represents a vertex on a graph

Data Member	Type	Description
id	int	Unique ID number for the Vertex
edges	std::vector<Edge>	Container holding all of this Vertex's outgoing Edges

Member Method	Description
Vertex(int i)	Constructor that initializes the id to i
void add_edge(const Edge & e)	Inserts the Edge into the edges vector
void print(ostream & out)	Prints Vertex information (for debugging)

EDGE (CLASS)

Represents an edge between two vertices on a graph; edges are one-directional (go from src→ dst)

Data Member	Type	Description
src	int	ID number of source Vertex
dst	int	ID number of destination Vertex
weight	int	This Edge's weight (cost) value

Member Method	Description
Edge(int s, int d, int w)	Constructor that initializes all data members
void print(ostream & out)	Prints Edge information (for debugging)

GRAPH (CLASS)

Represents a graph

Data Member	Type	Description
vertices	Vertex *	Dynamically allocated array of Vertex; each Vertex's ID number corresponds to its index in this array
num_vertex	int	Number of vertices in this Graph
num_edge	int	Number of edges in this Graph

Member Method	Description
Graph(string file_name)	Constructor that takes in a *.graph* file; it should open the file and initialize this Graph using the data in the file**, assume that the file is valid
void add_edge(int src, int dst, int w)	Adds a new edge with the given values to the appropriate Vertex(s) in this Graph
void print(ostream & out)	Prints Graph information (for debugging)
~Graph( )	Destructor

****About .graph Files**

.graph files start with an integer indicating how many vertices are in the graph. Every line after is a triple of integers representing an edge in the form of:

```
SRC_VERTEX_ID   DEST_VERTEX_ID   WEIGHT
```

PART 2: KRUSKAL'S ALGORITHM

Implement a main program that finds the minimum spanning tree of a graph. **Include a MAKEFILE for your program that builds your program into an executable named *kruskals*.**
For the algorithm, refer to the Minimum Spanning Tree lecture slides and Wikipedia entry.

The primary function (that does most of the work) should be called:

void kruskals (const Graph & g)

You do not have to match this signature exactly; it is only an example.

The program should take **one argument**:

1) A *.graph* file

You may assume the argument is valid.

Your program should output the **total cost** of the minimum spanning tree, followed by the tree itself displayed with the following form:

[EDGE] (TOTAL_COST)

For example, if you ran: *kruskals rdm.graph*
The output could would look something like:

```
200
[ 1-2 ] ( 2 )
[ 2-5 ] ( 2 )
[ 2-3 ] ( 3 )
...
[ 0-1 ] ( 4 )
```

Here, *200* is the total cost of this tree. Note that this is only an example, not an exact output.

REPORT

Submit a PDF with the following contents to Gradescope.

1. Code Implementation and Time Complexity

Copy and paste (or screenshot) **your entire function called kruskals** (described in Part 2) along with any essential, non-trivial helper functions. Add time complexities as comments (in Big-O notation) next to the function declarations. T(N) is not required.

2. Standard Graph Test w/ Valgrind

Screenshot a compilation of your program, execution, and output of the following command in the terminal:

```
valgrind  kruskals  large.graph
```

4. Random Graph Test w/ Valgrind

Screenshot a compilation of your program, execution, and output of the following commands in the terminal:

```
genGraph  >  rdm.graph
valgrind  kruskals  rdm.graph
```

Homeworks that are not formatted nicely will not be accepted. Pay careful attention to memory management. Do not share memory across objects. Obvious code copying, from a textbook, from another student, or within your own program, will be down-graded (write everything once (perhaps by writing auxiliary functions), then call the function or method each time that operation is required). Feel free to add additional functions or methods, but please use the data members outlined in the homework.

SOURCE CODE SUBMISSION

Submit an archive of your source code to Canvas. It should include:

- **Makefile**
- **.h and .cpp files**
- **rdm.graph (that you generated and tested for your report)**

Do not include any other extraneous files.