CSE310 Programming Project 3
OUT: WED, 11/20/2013
DUE: WED, 12/04/2013

In this programming project, you will be implementing the data structures `min-heap` and `disjoint set`, as well as `Kruskal's algorithm` for computing a minimum spanning tree. You need to submit your project electronically, using a single file with the format `CSE310-P03-LastName-FirstName.zip`. We will use your electronic submission to compile and run on test cases.

You program should read in a graph from an ASCII file named `data.txt` and write the output to an ASCII file named `output.txt`. The first line of `data.txt` contains two integers `nV` and `nE`, which represent the number of vertices and number of weighted edges in an undirected graph G. The next `nE` lines each contain three integers `u, v, cost`, where `u` and `v` are the end vertices of the corresponding edge and `cost` is the cost of that edge. The vertices are numbered from 1 to `nV`. The edges are numbered from 1 to `nE`. For example, a graph of 6 vertices and 8 edges is given as

```
6 8
1 2 10
1 3 9
2 3 7
2 4 2
3 5 5
4 5 3
4 6 8
5 6 4
```

Your program should initialize `nV` disjoint sets (one for each vertex) and initialize a min-heap of `nE` elements by performing `nE` insertions to the initially empty heap, one insertion per edge. When this is done, your program will print out the set structure and the heap structure. Your algorithm will then compute an MST using Kruskal's algorithm. Whenever an edge is deleted from the heap, you will print out the new heap structure. Whenever two sets are union-ed together, you will print out the new set structure. Eventually, you will print out the edges in the MST and the total cost. For the sample input, your program should generate the following output.

```
Set Structure:
0, 0, 0, 0, 0, 0

Heap Structure:
```

```
2, 4, 3, 5, 7, 9, 8, 10

Heap Structure:
3, 4, 8, 5, 7, 9, 10

Set Structure:
0, 4, 0, -1, 0, 0

Heap Structure:
4, 5, 8, 10, 7, 9

Set Structure:
0, 4, 0, -1, 4, 0

Heap Structure:
5, 7, 8, 10, 9

Set Structure:
0, 4, 0, -1, 4, 4

Heap Structure:
7, 9, 8, 10

Set Structure:
0, 4, 4, -1, 4, 4

Heap Structure:
8, 9, 10

Heap Structure:
9, 10

Heap Structure:
10
```

```
Set Structure:
4, 4, 4, -1, 4, 4

MST:
2 4 2
4 5 3
5 6 4
3 5 5
1 3 9

Total Cost: 23
```

Grading Policies: Your program should be implemented using `C++`. Your program should be working on `general.asu.edu`. **No credit will be given for programs that do not work on `general.asu.edu`.** Therefore, you should **start with something that works**, and keep adding functions to a working program.

(10 pts) Disjoint set operations: You should implement union by rank and find with path compression.

(10 pts) Heap operations: Remember this is a min-heap. You only need to implement the operations necessary for the MST algorithm.

(10 pts) MST algorithm: Using heap operations to select the next edge. Using set operations to decide whether two nodes are already connected.

(10 pts) Input and output of graphs: Your output should match the sample output.

(10 pts) Modularity: one module for heap, one module for set, one module for mst. Other modules may be added if necessary. You will also need to provide a Makefile. The executable file should be named `run`.

The following are some helpful hints for those with difficulties in programming. You don't have to follow/use these.

- Data structures for `edges` of the graph:

```
typedef struct TAG_edge{
    int v1;     /* from vertex */
    int v2;     /* to vertex */
    int cost;   /* cost of edge */
}EDGE;

typedef EDGE *ElementT;
```

- Data structures for `heap` on the edges:

```
typedef struct TAG_HEAP{
    int capacity;  /* max size of the heap */
    int size;      /* current size of the heap */
    ElementT *elements;  /* array of pointers to elements */
}HEAP;

typedef HEAP *heap;
```