CSC 316 – Data Structures

Programming Project #3 – Kruskal's Algorithm for Minimum Spanning Trees
Due Date: November 9, 2017, 11:59pm

Project Objectives
This task deals with Minimum Spanning Trees (MST) on weighted graphs. More specifically, you will implement Kruskal's MST algorithm. MST problems arise very often, especially in applications related to computer networks. Furthermore, an efficient implementation of Kruskal's algorithm requires a number of the data structures we have discussed in class (i.e., heaps, up-trees and the union-find algorithms, and graphs), and you will have the opportunity to integrate all of them in a single program.

Input
Each line of the input file will represent one edge in an undirected graph. It will contain two integers – the endpoints of the edge – followed by a real number – the weight of the edge. You may assume that the vertices are numbered 0 through n, that $n < 1000$, and that the number of edges, m, is such that $m < 5000$. The last line of the file will contain -1, to denote the end of input. You are *not* expected to check the input for errors of any sort.

Files `graph1` and `graph2` in the `Project3` directory are sample input files. The output of your program will be tested on an input file not available in advance.

Description
As your program reads in an edge e, it will:

- construct an `edge` record for it (consisting of four fields, `vertex1, vertex2, weight,` and `next`),
- insert the record for e in the adjacency list of the two vertices,
- insert the edge into a heap.

Since the number of edges will be less than 5000, you can represent the heap as an array of 5000 edge records; the key (priority) of each edge will be its weight. Obviously, you will need to implement the heap operations `insert` and `deleteMin` [**Note:** Kruskal's algorithm may be implemented without using a heap. One can sort the edges and consider them in increasing weight. This is **strongly discouraged** in this task, and the output has been designed so that you will need to implement a heap.]

In order to implement Kruskal's algorithm, you also need to implement the up-tree operations to keep track of how connected components change during the execution of the algorithm. Since vertices will be numbered 0 through n (your program should determine n), you can use the array implementation described in class[1]. Initially, each vertex is in a connected component by itself, and adding an edge between two components has the effect of a `union` operation. Make sure that you

[1] It should not be hard to modify the `find` and `union` algorithms to work for the array representation of up-trees.

implement a balanced version of `union`, but regarding the `find` operation, you may implement it with or without path compression.

Finally, you are free to choose any of the data structures we have discussed in class (e.g., linked lists, search trees, etc.) to implement the set of edges in the MST that Kruskal's algorithm returns.

Deliverables: Source Code and Output
The programs you submit will prompt the user for the names of the input and output files. Specifically, your code will implement:

1. The `heap` structure, and the `insert` and `deleteMin` operations, using an array-based implementation.
2. The `up-tree` structure, along with the `union` and `find` operations; use an array representation of up-trees.
3. An adjacency list representation of graphs.
4. Kruskal's algorithm for computing the MST of a graph.

Since we need to test whether you have implemented all data structures and operations described above, your program will print the content of several data structures, as follows.

Printing the Heap. Once all the edges have been read in and the heap (partially ordered by edge weight) constructed, your program will print the contents of the heap array in m lines (recall that m is also the number of edges in the graph). Each line i will represent the edge in position i of the heap. Each line should have exactly 9 characters: the first endpoint of the edge, right justified in a field of 4; a space; then the second endpoint, also right justified in a field of 4. The first number must always be less than the second. If an edge joins vertices `634` and `43`, the output should contain the line

␣␣43␣␣634

Printing the MST. Once your program has computed the MST, it will print n lines, each representing one edge of the MST. Each line should have 9 characters, exactly as described above. Furthermore, the edges must appear in the output in lexicographic order: all the edges on vertex 0 first, then those on 1, etc. Also, all the edges of vertex 0 must be sorted according to the second vertex, etc[2]. The output from the sample input files `graph1` and `graph2` are in files `graph1_output` and `graph2_output`.

Printing the Adjacency List. Finally, your program will print $n + 1$ lines, each corresponding to one of the $n + 1$ vertices of the graph. The i-th line, $i = 0, \cdots, n$, will contain, in increasing order, all the vertices that are adjacent to vertex i (but not vertex i itself). Each vertex should be right justified in a field of 4 characters, with vertices printed on the same line separated by a space.

Obey the format carefully. The output of your program will be verified automatically with the `diff` *utility.*

[2]There are different ways to accomplish this, depending on how you have decided to implement the set of edges of the MST. If you have chosen to represent the set of edges as a linked list, for instance, you may want to keep the list sorted by the first vertex, and also sort edges with the same first vertex according to the second vertex.

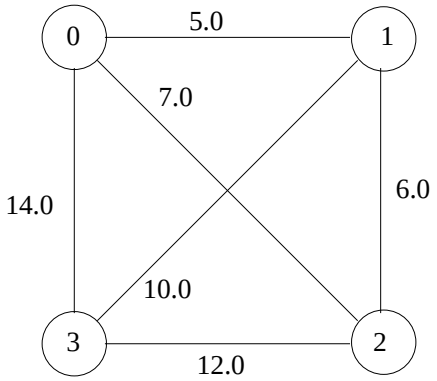

Figure 1: Graph for example input

Example
Suppose that the input to your program corresponds to the graph shown in Figure 1:

```
2 0 7.0
3 2 12.0
0 3 14.0
1 0 5.0
3 1 10.0
1 2 6.0
-1
```

Edges are inserted into the heap in the order given. Let e_0 be the first edge (of weight 7.0), e_1 the second edge (of weight 12.0), and so on. The array representing the final heap, after all insertions have taken place, will contain the edges in this order: $e_3, e_0, e_5, e_1, e_4, e_2$. The MST contains edges e_3, e_5, and e_4. The output of your program should be 13 lines: the first 6 are the contents of the heap of edges after all insertions[3], the next 3 lines are the edges of the MST, sorted by vertex, and the last four lines are the adjacency list.

```
␣␣␣0␣␣␣␣1
␣␣␣0␣␣␣␣2
␣␣␣1␣␣␣␣2
␣␣␣2␣␣␣␣3
␣␣␣1␣␣␣␣3
␣␣␣0␣␣␣␣3
␣␣␣0␣␣␣␣1
␣␣␣1␣␣␣␣2
␣␣␣1␣␣␣␣3
␣␣␣1␣␣␣␣2␣␣␣␣3
␣␣␣0␣␣␣␣2␣␣␣␣3
␣␣␣0␣␣␣␣1␣␣␣␣3
␣␣␣0␣␣␣␣1␣␣␣␣2
```

[3]Note that edge e_3 is given in the input file as $\{1, 0\}$, but in the output it appears as $\{0, 1\}$.

Submission: Submit all your programs using the **submit** tools, by 11:59pm on November 9, 2017.

Grading

Adjacency list structure	10 Points
Heap implementation	30 Points
Up-tree implementation	30 Points
Kruskal's algorithm	30 Points
	100 Points

Important reminder: Homework is an individual, not a group, project. Students may discuss approaches to problems together, but each student should write up and fully understand his/her own solution. Students may be asked to explain solutions orally if necessary.