# TCSS 343: Design and Analysis of Algorithms
Spring 2018, Homework #5 — programming assignment
Due: Tuesday, May 29 (Part A); Friday, Jun 1 (Part B)

This final homework consists out of 2 parts:

- Part A is an exercise about Kruskal's algorithm. It counts for 5 homework points. Please submit a hard copy of your solution in class on Tuesday May 29. It is highly recommended that you finish part A before starting on part B.
- Part B is a programming assignment. It counts for 20 homework points. Details are given below. The deliverables for part B should be submitted on Canvas by Fri Jun 1, 11.59pm. They are:
  - Modified source code for the assignment in part B.
  - A report for part B.

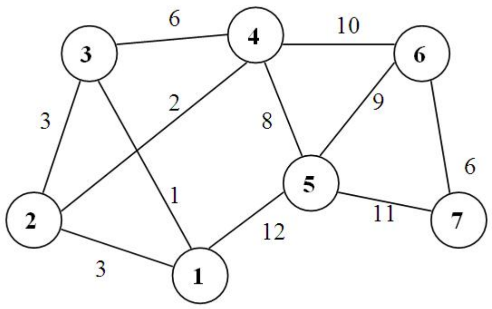
Make sure you have acknowledged all persons with whom you worked. Even though you are encouraged to work together on problems, the work you turn in is expected to be your own. When in doubt, invoke the Gilligan's Island rule (see the syllabus) or ask the instructor.

# Homework 5A

Kruskal's algorithm to find a minimum spanning tree can be implemented in different ways. An alternative to pre-sorting the edges is to form a heap initially and then use the heap DeleteMin to extract the next edge. This may be better than sorting if the graph is dense. In this exercise we assume that Kruskal's algorithm is implemented in this way using a heap. Furthermore, we assume that the algorithm uses union-by-rank (use height – ties go to the left tree) but no path compression on find operations.

For the graph below, show what happens during Kruskal's algorithm using the implementation described above. I.e., show or describe the decisions the algorithm makes – preferably each time an edge is added to the solution subgraph. Also show what both the heap of edge weights and the forest representation (uptree) look like after each union operation.

An appropriate way to show your work could be based on Figure 9.5 on p. 326 of the textbook. However, you should replace the column that has the sorted list of edges by a column that shows the heap of edge weights (since we are not pre-sorting edges but keeping them in a heap instead). Furthermore, you should add a column to show the forest representation (which should be a picture like Figure 9.8(a) or (b)).



## Homework 5B

**Learning Objectives** Reading, understanding, and using a graph ADT; Kruskal's algorithm and its implementation (heap, uptree); more practice with debugging and constructing good test cases.

**Problem description** The power grid (the infrastructure and wires needed to provide electricity) in Tacoma has been destroyed by a huge fire. An emergency plan has been put into place, but it is expensive and so therefore cannot be a long-term solution. You are part of the reconstruction team. You are one of the software engineers who need to figure out a good way to connect the power grid.

You have decided that the best way to do this is to first list all the places that need power. Each one of these places will be represented by a vertex in a graph. Making a direct power connection between some pairs of places will be impossible (because connecting them would require digging up parts of the city that can't be disrupted or because it would require construction that is far too expensive). Of the remaining possible connections, you would have to estimate how much it would cost to connect the two places.

Fortunately, someone has already provided all of this information to you. Your task is to come up with a set of connections such that there is an electrical path from every vertex to every other vertex. The head of your software design team has decided that Kruskal's minimum spanning tree algorithm is the best way to solve this problem. You should implement the algorithm in a way that is asymptotically optimal for Kruskal's algorithm (otherwise, what's the point?). You should use a heap implementation of the algorithm.

You are provided with code that implements a simple graph ADT. You will need to write code that implements Kruskal's algorithm. You should allow the user to specify a graph file that has place and cost data (terminal input is fine – you do not need to write a GUI). As output, you should list the set of edges in the minimum spanning tree your algorithm generates. Also, you should output the total cost of the minimum spanning tree.

**Tips** Familiarize yourself with the graph ADT code before doing anything. Notice that when the graph data is read in, it returns a hash table that maps labels to vertices. Then read and understand how Kruskal's algorithm works (i.e. do part A of this homework). Only then will you be ready to write code.

**Starter Code** The following files are available on the course moodle: Vertex.java, Edge.java, SimpleGraph.java, GraphInput.java, InputLib.java, KeyboardReader.java. You should generate your own test cases and test them on your code.

**Submission format** To facilitate the grading process, please make a class `PowerGrid` that contains method `public Set<Edge> kruskal(SimpleGraph graph)` where `SimpleGraph` and `Edge` are the classes from the starter code. This method takes a graph as an input and produces a set of edges that comprise a minimum spanning tree of the given graph.

**Software Engineering** Understand the graph ADT code and Kruskal's algorithm before writing code. You will waste valuable time if you don't. (Thought questions: Is the graph ADT code written in a way you would have implemented it? Why or why not? Are there object-oriented principles you think are not being adhered to in the code?) As usual, comment your code and use good programming practices (good variable names and good indentation), so that it is easy to figure out what it is doing when/if you detect a bug.

**What to Turn In**

- On the course Canvas page, turn in a zip-file with:
  - The code you created. You should not turn in the starter code, since you should not have changed it.
  - Two test files, containing graph data with a non-trivial number of vertices (at least 10) and a non-trivial number of edges (at least 25). Think carefully about your test cases, because they should illustrate that your algorithm works and will therefore be useful in debugging.
  - A report (1-2 pages) that described how you approached the problem, what troubles you had, and what you learned.

**Grading** This programming assignment counts for 20 homework points. It will be graded on functionality, code organization, and readability of code.