COSC-211 Final Exam Review

Part 1

Your goal, throughout this exercise, is to review the data structures we've discussed in the course. If I ask you what data structure you would use in a particular scenario, don't simply say "I'd use a stack" and move on to the next question—take this time to review what a stack is, what operations it supports, different implementations we've considered and how they work, and what the runtimes are for each operation. You will only get as much out of this review as you put into it!

The setting: Imagine you work for UPS and you are responsible for planning the routes that different trucks take to deliver packages. Throughout the day, you are issued many different requests, where each request consists of a list of stops. Your job is to determine an order in which a single truck should visit all the stops listed in the request (there is one truck per request, but each request has its own separate truck). Because UPS owns many different trucks and packages become ready for delivery at all different times, you might have many requests pending at the same time.

1. What data structure will you use to hold onto the list of pending requests, assuming your goal is to get each truck on the road as soon as possible after its request is issued? What if requests that arrive later in the day are more likely to include same-day deliveries and your goal is to guarantee as many same-day deliveries as possible? What if some customers have paid more to get their packages delivered faster?

Representing a route request: Each new request consists of a list of stops that must be visited by a single truck. You can choose to schedule the stops in any order, but you must start and end the route at the warehouse. It takes a certain amount of time to drive between each pair of stops, and you are given the travel time information as part of the request. For example, a request to visit stops `A`, `B`, and `C` might look like:

```
Stops: A, B, C
Warehouse: W
W <--> A: 2
W <--> B: 5
W <--> C: 4
A <--> C: 3
B <--> A: 3
B <--> C: 4
```

2. What data structure will you use to hold onto the information contained in a request? How will you first initialize this data structure when you get a new request?

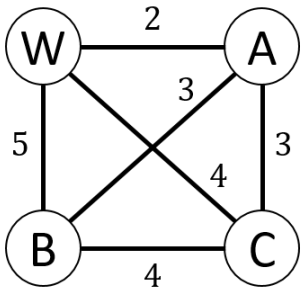
COSC-211 Final Exam Review

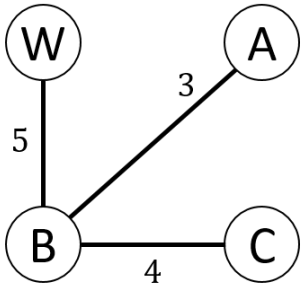
Part 2

Let's suppose that in Part 1 you decided to represent the information in a request using a graph, where each vertex represents a stop and edges are weighted with the travel time between the vertices. You're now ready to come up with a way of choosing routes—that is, choosing the order in which to visit the vertices in the graph. The goal is for the route to have the shortest travel time possible.

It turns out this is a Hard problem to solve (here "hard" is a technical term that means we don't know any better way to solve it than to try out all the possibilities and see which is best, and nobody really believes that a better way might exists). Instead, you decide to use an approximation: you will build a tree on the graph rooted at the warehouse, traverse the tree, and let your route be the order in which you visit the vertices during your traversal.

For example, the graph representing the request from Part 1 and one possible tree we could build on this graph look like this:





3. In what order do we visit the vertices when we do an in-order traversal of this tree, starting at the warehouse?

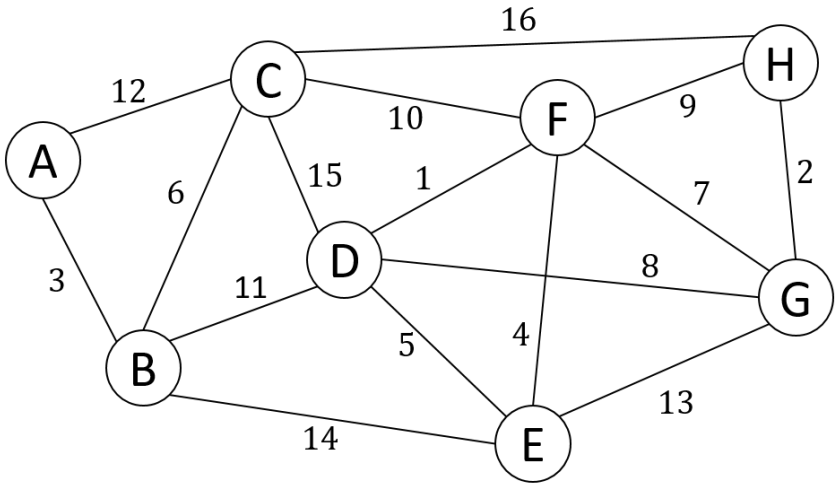
4. We want to come up with the lowest cost possible using this method of finding approximately good routes. Which edges should you include in your tree (in this particular graph and in general)?

This is called the *Minimum Spanning Tree* (MST) problem.

There are many algorithms for find an MST in a graph. One option is Kruskal's algorithm, proposed by Joseph Kruskal in 1956. The idea behind Kruskal's algorithm is that we want our tree to contain edges with as low a cost as possible, so we'll consider our edges one by one in increasing order of weight, and add the edge to our MST if doing so does not cause there to be a cycle in the tree.

```
Kruskal(G = (V,E))
   for each edge (u,v) in E, in increasing order of weight:
      if adding (u,v) to the MST doesn't cause a cycle
         add (u,v) to the MST
```

5. Run Kruskal's algorithm by hand to build an MST for the following graph (your goal here should be to get a feel for what the algorithm does):



COSC-211 Final Exam Review

Part 3

Data structures for Kruskal's algorithm: In order to implement Kruskal's algorithm, you will need to use several data structures to keep track of what's going on. In fact, each line of the pseudocode in Part 2 corresponds to a decision you'll have to make about what data structures are best suited to the different components of the algorithm. As you're making these decisions, consider what operations you will need to be able to do quickly, and which of the data structures we've discussed in class perform those operations efficiently.

6. What data structure will you use to keep track of what edge you should consider next?

7. What data structure will you use to determine whether you can add the edge to the MST without causing a cycle?

8. What data structure will you use to keep track of what edges you have added to the MST?