CS2040S: Data Structures and Algorithms

Discussion Group Problems for Week 12

For: April 7–April 11

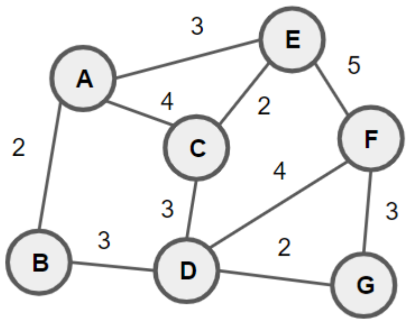
Goals:

- Priority Queue
- Union-Find review
- MST

Problem 1. MST Review

Note to tutors: 10 - 15 minutes.

Problem 1.a.



Can you use the cycle and cut property of MST that we learnt in class to determine which edges must be in the MST?

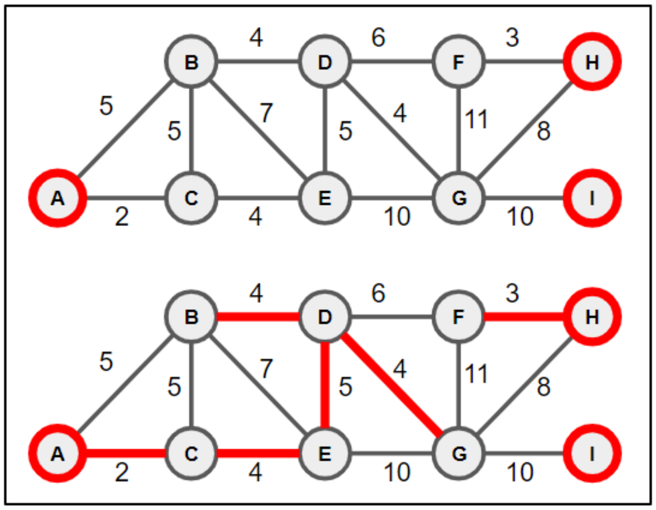
Perform Prim's, Kruskal's, and Boruvka's (optional) MST algorithm on the graph above.

Problem 1.b. Henry The Hacker has some ideas for a faster MST algorithm! Recently, he read about *Fibonacci Heaps*. A Fibonacci heap is a priority queue that implements insert, delete, and decreaseKey in $O(1)$ amortized time, and implements extractMin in $O(\log n)$ amortized time, assuming n elements in the heap. If you run Prim's Algorithm on a graph of V nodes and E edges using a Fibonacci Heap, what is the running time?

Problem 2. **Power Plant**

Note to tutors: 10 - 15 minutes.

Given a network of power plants, houses and potential cables to be laid, along with its associated cost, find the cheapest way to connect every house to at least one power plant. The connections can be routed through other houses if required.



In the diagram above, the top graph shows the initial layout of the power plants (modelled as red nodes), houses (modelled as gray nodes) and cables (modelled as bidirectional edges with its associated cost). The highlighted edges shown in the bottom graph shows an optimal way to connect every house to at least one power plant. Come up with an algorithm to find the minimum required cost. You may assume that there exists at least one way to do so.

Problem 3. (Horseplay)
(Relevant Kattis Problem: https://open.kattis.com/problems/bank)

There are m bales of hay that need to be bucked and n horses to buck them.

The i^{th} bale of hay has a weight of k_i kilograms.

The i^{th} horse has a strength s_i—the maximum weight, in kilograms, it can buck—and can be hired for d_i dollars.

Bucking hay is a very physically demanding task, so to avoid the risk of injury every horse can buck at most one bale of hay.

Determine, in $O(n \log n + m \log m)$, the minimum amount, in dollars, needed to buck all bales of hay. If it is not possible to buck them all, determine that as well.

Problem 4. (Union-Find Review)

Problem 4.a. What is the worst-case running time of the find operation in Union-Find with path compression (but no weighted union)?

Problem 4.b. Here's another algorithm for Union-Find based on a linked list. Each set is represented by a linked list of objects, and each object is labelled (e.g., in a hash table) with a set identifier that identifies which set it is in. Also, keep track of the size of each set (e.g., using a hash table). Whenever two sets are merged, relabel the objects in the smaller set and merge the linked lists. What is the running time for performing m Union and Find operations, if there are initially n objects each in their own set?

More precisely, there is: (i) an array *id* where *id*[j] is the set identifier for object j; (ii) an array *size* where *size*[k] is the size of the set with identifier k; (iii) an array *list* where *list*[k] is a linked list containing all the objects in set k.

```
Find(i, j):
    return (id[i] == id[j])

Union(i, j):
    if size[i] < size[j] then Union(j,i)
    else // size[i] >= size[j]
        k1 = id[i]
        k2 = id[j]
        for every item m in list[k2]:
            set id[m] = k1
        append list[k2] on the end of list[k1] and set list[k2] to null
        size[k1] = size[k1] + size[k2]
        size[k2] = 0
```

Assume for the purpose of this problem that you can append one linked list on to another in $O(1)$ time. (How would you do that?)

Problem 4.c. Imagine we have a set of n corporations, each of which has a (string) name. In order to make a good profit, each corporation has a set of jobs it needs to do, e.g., corporation j has tasks $T^j[1 \ldots m]$. (Each corporation has at most m tasks.) Each task has a priority, i.e., an integer, and tasks must be done in priority order: corporation j must complete higher priority tasks before lower priority tasks.

Since we live in a capitalist society, every so often corporations decide to merge. Whenever that happens, two corporations merge into a new (larger) corporation. Whenever that happens, their tasks merge as well.

Design a data structure that supports three operations:

- `getNextTask(name)` that returns the next task for the corporation with the specified name.
- `executeNextTask(name)` that returns the next task for the corporation with the specified name and removes it from the set of tasks that corporation does.
- `merge(name1, name2, name3)` that merges corporation with names `name1` and `name2` into a new corporation with `name3`.

Give an efficient algorithm for solving this problem.