

Homework 5

Due April 24

The first problem is a warmup — answers to most parts should be just a sentence or two.

The next seven of these problems depend on your having a deep understanding of several algorithms, like the minimum spanning tree algorithm, depth-first traversal (or search), breadth-first traversal/search, Dijkstra’s shortest-path algorithm, and topological sort. In most cases, the solution to a problem is a modest modification of one of these algorithms, although in some cases a little extra cleverness is required. But in no case do you have to spend a lot of time trying to come up with something as complex as any of these algorithms from the start; you just want to modify them to make them do something a little new.

The last two are the “think hard and come up with something new” problems. As on the previous assignment, I recommend that you try to solve some simple examples first, and work with friends to generate ideas. If, on the last problem, you get sick of computing by hand and don’t want to write a program, solve some easier problem (like “36 speeds and 4 carts”).

Problem 5.1

Several quick-and-easy warmup questions. Your answers should be just a sentence or two for each of these.

(a) You will be given a list of all the k -letter words in English; there are n of them. Some of them are suitable for use in the Jumble puzzle in the newspaper, i.e., no permutation of the letters produces another word; some are not. For example, UVULA is a good jumble word, but SKATE is not (because “TAKES” and “STEAK” and “TEAKS” are also words). Describe an algorithm that finds all the “good for jumble” words quickly. (A good running time would be $O(nk \log k)$).

(b) From part a, you have a list of m different k -letter “jumble words”, i.e., words suitable for use in a jumble puzzle. You’re allowed to do some pre-computation so that you can solve the following problem quickly: Someone gives you a set of letters (“VUUAL”) that is a shuffled version of one of the words, and you have to say what the word is (“UVULA”). You should be able to solve the problem in $O(k^2)$ or better. For the pre-computation, you’re allowed to use $O(n)$ space (but *not* $O(nk)$ space), and as much time as you please.

(c) If $G = (V, E)$ is a connected graph with positive edge weights and $u, v \in V$ are vertices in G , explain why the shortest path from u to v has at most $|V| - 1$ edges in it.

(d) Dijkstra’s algorithm requires that all edge-lengths (or weights) be non-negative. If we have a graph G where some edges have negative weights, there must be a most-negative weight, $-W$. If we add W to every edge-weight, we get a new graph G' where all weights are nonnegative. We can apply Dijkstra’s

algorithm to G' to find shortest distances in G . Or can we? Either explain why this putative algorithm works, or give an example on which it fails.

(e) What's the big- O bound for

- Insert(Object o , Position p) in in a double-linked list, where this operation inserts the object o in a new node at the position just following p ?
- Finding an element in a BST that has n nodes?
- Finding an element in a balanced BST with n nodes?
- Emptying a stack with n elements? If you thought that emptying a stack would happen a lot, could you improve your stack implementation to make it empty-able in much better big- O time?

(f) We've used a hashtable to implement "arrays with arbitrary indices". To make $foo[a] = b$, where a and b are arbitrary object references, we put the pair (a, b) into a hashtable, with a used as the hash-key. Suppose you want to implement a "pairing": we have many object pairs of the form (c, d) ; we'd like to be able, given c , to recover d , and given d , to recover c . (A humble analogy: you know lots of couples; when someone says "Gina" you want to be able to say "yeah, she goes out with Fred," but if someone else says "Fred", you want to say "he goes out with Gina." We'll assume that each person goes out with one other person, but we want to be able to add or delete couples from our record quickly.) How would you implement such a pairing?

Problem 5.2

(a) Let G be a simple connected graph with n vertices and m edges. Explain why $O(\log m)$ is $O(\log n)$ (we did this in class, but I'd like you to explain it again.)

(b) Show that the hypothesis that G is simple is necessary by exhibiting a sequence of non-simple graphs where graph i has n_i nodes and m_i edges, but $\log m_i \approx n_i \log n_i$.

(c) Show that the hypothesis that G is connected is necessary by exhibiting a sequence of disconnected graphs in which $\log n_i = i \log m_i$.

Problem 5.3

(a) Explain intuitively why if T is a tree and u and v are nodes of T , there's a unique path from u to v in T (assuming you traverse each edge at most once, so paths like u, w, u, w, u, w, v are not allowed).

(b) Show how to modify Dijkstra's algorithm to not only output the distance from v to each vertex in G , but also to output a tree T rooted at v such that the path in T from v to a vertex u is a shortest path in G from v to u .

Problem 5.4

Can we use a queue instead of a stack as an auxiliary data structure in the topological sorting algorithm given below? Why or why not?

Algorithm 5.4.1 TopologicalSort(G)

Input: A digraph G with n vertices

Output: A topological ordering v_1, \dots, v_n of G

```
 $S \leftarrow$  an initially empty stack
incounter  $\leftarrow$  an array of size  $n$ 
for all  $u$  in  $G.vertices()$  do
    incounter[ $u$ ]  $\leftarrow$  the in-degree of  $u$ 
    if incounter[ $u$ ] = 0 then
         $S.push(u)$ 
    end if
end for
 $i \leftarrow 1$ 
while ! $S.isEmpty()$  do
     $u \leftarrow S.pop()$ 
    Set  $u$  to be vertex number  $i$  in the topological ordering
    for all outgoing edge  $(u, w)$  of  $u$  do
        incounter[ $w$ ]  $\leftarrow$  incounter[ $w$ ] - 1
        if incounter[ $w$ ] = 0 then
             $S.push(w)$ 
        end if
    end for
     $i \leftarrow i + 1$ 
end while
```

Problem 5.5

Give a linear-time (linear in the number of *edges*) algorithm which, given an undirected graph G and a particular edge e in it, determines whether G has a cycle containing e .

Problem 5.6

Give an algorithm that takes as input a directed simple (no loops) graph with positive edge lengths, and returns the length of the shortest cycle in the graph (or indicates that the graph is acyclic). Your algorithm should take time at most $O(|V|^3)$.

Problem 5.7

Give a linear-time algorithm that takes as input a connected, undirected graph G and indicates whether there is an edge that can be removed from G while still leaving G connected. Can you reduce the running time of your algorithm to $O(|V|)$?

Problem 5.8

You work at an amusement park. You've just installed the WARRIOR — an amazing new rollercoaster. It's so radical that no one knows how fast it can run and still be safe. You can set it for any speed from 1 to 100 (integers only). You've got five carriages to use; if you set it too fast, the carriage will fall off the tracks and be destroyed. If you run it at a slow enough speed, everything is fine and the carriage can be reused. You can perform one test per day. The Boss tells you "I want to know the top speed for this thing, and I want to get it running soon so that we can start paying for it. That means you've GOT to leave me one carriage to actually use once we know the top speed. In the meantime, get out there and start testing! And don't just try speed 1, then speed 2, then speed 3...these tests each take a whole day, and we don't have 100 days to spend on this, damnit! That'll be the end of summer!"

So: you've got five carts. You're allowed to destroy four of them. Devise a testing strategy to determine the maximum safe speed in as few days as possible, regardless of what that maximum speed is. The manufacturer has told you one thing: "We tested it at speed 1. It works OK. So speed 1 is safe."

Then briefly describe your testing strategy, and tell us how many days from now you'll know the answer. (If you run a test today and it tells you the answer, we'll say that you solved the problem in 1 day; if you need another test, and tomorrow's test answers things, we'll say you solved it in two days. Tell us the LARGEST number of days you could possibly use, if you employ your very clever strategy rather than the "just increase by one speed each day" approach that your boss has maligned.)

(The best way to do this, once you've got an algorithm, is to write some code to implement it. If that seems too painful to you, solve a simpler problem: suppose that the Warrior has only 30 speeds, and you've got only four carts (and hence can use only three for testing). Be sure to tell us whether you're solving the small or the large problem.)