Practice Assignment 3
Daniel Schroeder
11/21/17

CS325: Analysis of Algorithms, Fall 2017

Practice Assignment 3*

Due: Tue, 11/21/17

Homework Policy:

1. Students should work on practice assignments individually. Each student submits to TEACH one set of *typeset* solutions, and hands in a printed hard copy in class or slides it under my door before the midnight of the due day. The hard copy will be graded.
2. Practice assignments will be graded on effort alone and will not be returned. Solutions will be posted.
3. The goal of the assignments is for you to learn solving algorithmic problems. So, I recommend spending sufficient time thinking about problems individually before discussing them with your friends.
4. You are allowed to discuss the problems with others, and you are allowed to use other resources, but you *must* cite them. Also, you *must* write everything in your own words, copying verbatim is plagiarism.
5. More items might be added to this list. ☺

Problem 1.

(a) Find a graph that has multiple minimum spanning trees.

(b) Prove that any graph with distinct edge weights has a unique minimum spanning tree.

(c) Find a graph with non-distinct edge weights that has a unique minimum spanning tree (can you generalize (b)?).

Problem 2. Recall the job scheduling problem. The input is composed of the starting and finishing times of n jobs. We would like to find the maximum set of pairwise disjoint jobs. Consider the following alternative greedy algorithms for the job scheduling problem. For each algorithm, either prove or disprove (by presenting a counter example) that it always constructs an optimal schedule.

(a) Choose the job that ends last, discard all conflicting jobs, and recurse.

(b) Choose the job that starts first, discard all conflicting jobs, and recurse.

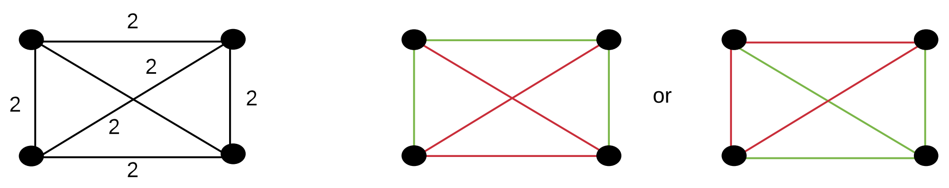
(c) Choose the job that starts last, discard all conflicting jobs, and recurse.

(d) Choose the job with shortest duration, discard all conflicting jobs, and recurse.

*Problems are from Jeff Erickson's lecture notes, and the book "Introduction to Algorithms", by Corman, Leiserson, Rivest and Stein.

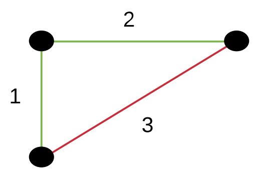
Problem 3. What is an optimal Huffman code for n characters whose frequencies are the first n Fibonacci numbers?

Problem 1.
(a) Find a graph that has multiple minimum spanning trees.



When a graph is complete, non-distinct, and has more than 3 vertices, there can be multiple MST's that connect all the vertices. As shown above, there are two distinct paths with a total weight of 6 that connect all 4 vertexes.

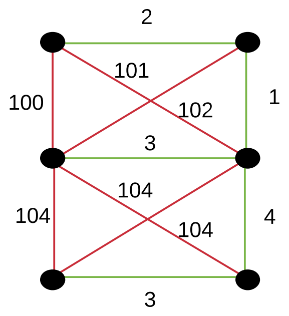
(b) Prove that any graph with distinct edge weights has a unique minimum spanning tree.



A graph that has all different edge weights can only have one unique minimum spanning tree because a graph of size N with vertices V has an MST of with a number of edges equal to V-1. In the example above of a graph with V size 3, this means that exactly one edge is excluded from the minimum spanning tree. Since all edges are distinct, there must be 2 edges that are less than the excluded third edge. This relates to the theory behind safe edges. If you choose an edge e to be a safe edge connecting two connected components of a graph, there could be ither paths that could connect the two components, say edge e' or edge e'', but these edges have a greater weight than safe edge e. This means that safe edge e must be in the MST becuase it connects the two connected components, doing so with the least weight.

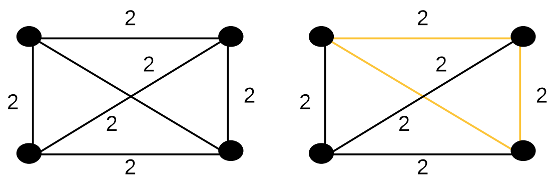
A more explicit proof by contradiction states that suppose a graph has to MST's T1 and T2, for these trees to be different, there must be an edge in T1 that does not exist in T2, call that e1. If you add e1 to T2, it creates a cycle with an edge e2 that is not in T1. From this we can gather that w(e1) < w(e2) or else e2 would have been in T1 since a minimum spanning tree connects all connected components by the least expensive edge. So, if w(e1) < w(e2) then w(T1) < w(T2) which means that T2 cannot be a minimum spannig tree.

(c) Find a graph with non-distinct edge weights that has a unique minimum spanning tree (can you generalize (b)?).



This ladder example has multiple non-distinct edges (104 and 3) but there is still a clear MST.

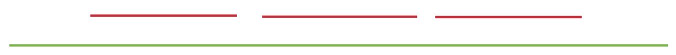
An occasion when there are multiple MST's when there are non-distinct edges is when two non-distinct edges for a cycle in the MST. This means that you can get an two MST's of the same weight by simply swapping one edge for the other with the same weight. My example from Problem 1 shows this:



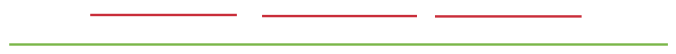
The yellow cycle contains all edges of the same weight, so any 2 of the 3 edges can be chosen for the MST.

Problem 2. Recall the job scheduling problem. The input is composed of the starting and finishing times of n jobs. We would like to find the maximum set of pairwise disjoint jobs. Consider the following alternative greedy algorithms for the job scheduling problem. For each algorithm, either prove or disprove (by presenting a counter example) that it always constructs an optimal schedule.

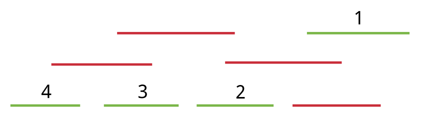
(a) Choose the job that ends last, discard all conflicting jobs, and recurse.
If we choose the job that ends last and it spans the entirety of the time, all other jobs will be skipped

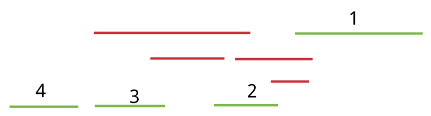


(b) Choose the job that starts first, discard all conflicting jobs, and recurse.
Similarly if the first job spanned the entire length of time and ended last, we would miss all the intermediate jobs like the last example.



(c) Choose the job that starts last, discard all conflicting jobs, and recurse.

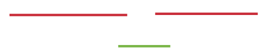




It seems like this greedy algorithm would work. It is sort of the reverse of the accepted algorithm from class where we start at the beginning and take the job that finishes first. If we start at the end and take the job that starts first, we leave all earlier jobs untouched if they are not conflicting. This also is safe of the two cases above where the job we picked spanned across the entire time.

(d) Choose the job with shortest duration, discard all conflicting jobs, and recurse.

This algorithm does not work for the simple fact that one small job could eliminate two disjoint jobs that conflict with it but not eachother. Example:



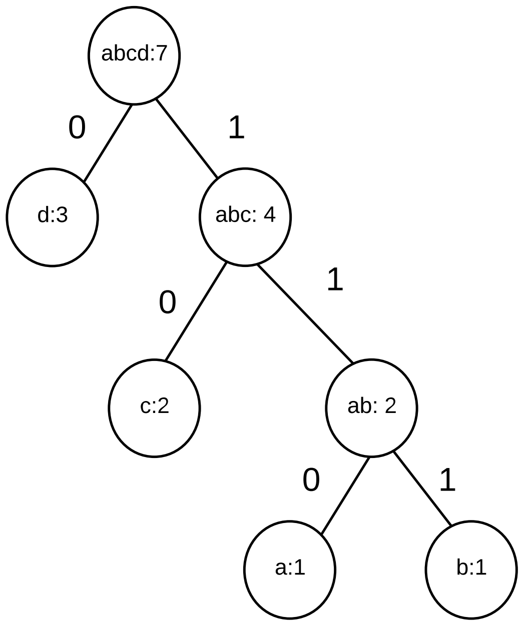
If these were the only 3 jobs given as input, choosing the smallest and eliminating all the conflicts would result in only one job getting executed when there could be two.

A Huffman Code for n characters whose frequencies are equal to the first nth number of the fibonacci sequence is as follows:

We want the characters with the two least frequencies (i.e. fib(1) and fib(2)) to be our bottom leaves, and every character after that to be the leaf of a parenting tree. Ideally, for every character greater than fib(2) the length of the huffman code string will be 2+i (with i being the fib(n) - 2). This will create a linear style tree where each level differes by one character in the huffman string.

For Fib(4):

a: 1 b:1 c:2 d:3



Character	**Code**
d	0
c	10
b	110
a	111

As the frequency grows with each new fibonacci number, the code length increases by one until the two leaves at the end of the tree (fib(1) and fib(2)) are of equal length. This will also make the most frequent number (fib(n)) a code length of 1.