Questions:

1. Problem 2, page 189. This is a theoretical problem, thinking about how the algorithms for minimal spanning trees and shortest paths work
2. Problem 4, page 190. Come up with an algorithm.
3. Problem 24, page 201. Come up with an algorithm.

Answers:

1.a. True. The minimum spanning tree of T has been found / defined already. If we were to increase the cost (weight) of each edge by the edge’s square, we’re raising each edge by a proportional factor, so squaring each edge would not change the results.

1.b. True. See above.

2.

Store both sequences, S(sequence) and S’(subsequence) in vectors

Keep S’.size() in a variable, i for reference

Keep a S’ iterator in a variable, j, initialize to zero

While j is not equal to i, iterate through elements of S

{

Check if current S element exists in S’[j]

If element exists and j is not equal to i

Increment j

}

If j is equal to i

Print a message to user that a valid subsequence exists

Else

Print a message to user that there is no valid subsequence

3.

I’m going to need some help working through this one. I’ve got some ideas, but this one has been pretty hard for me to flesh out.

I know we need to traverse the tree layer, and inspect the nodes in those layers to find the largest node value. Subsequently, need to set each node’s value in that layer to the maximum we just found. Maybe a more efficient way would be to look for LCM? That may ensure we have the lowest possible edge lengths at the end.

What kind of throws me is this is in the greedy chapter of the book, so iterating through a set of nodes to find a value, then reiterating to reset values does not seem greedy to me.