Homework 5, CSCE 240, Summer 2016

You just wrote a program to do a recursive depth first search on a complete binary tree, minimizing an objective function along the way.

You are to revise this code to be able to deal with all possible rooted trees, not just complete binary trees. This should need only minor modifications to your code from Homework 4. Namely, you'll need a variable-length list of child node subscripts, and you're need a loop to iterate through the children instead of just pulling up the left child and then the right child. Other than that, most things in your code should be re-usable.

However, this also presents the problem of testing your code, because allowing arbitrary rooted trees is more complicated than dealing just with complete binary trees.

So you will need to add two functions as well as the functionality to exercise those two functions.

One function will generate "arbitrary" trees. I put the "arbitrary" in quotation marks because they won't be totally arbitrary. You are to write a function that will generate trees with a bush factor of at most five (that is, less than or equal to five children for any given node) and that will sample random numbers to determine whether or not to add such a child. That is, your code to generate trees should run a loop from 0 through 4 inclusive. This loop will attempt to create a child node with probability 0.8. If the random number is less than or equal to 0.2, you skip the node creation for that subscript. If the random number is greater than 0.2, you generate the node.

To make life simpler, I have provided both a self contained random number function and a file of random numbers. The sequences of random numbers generated by the functions on linux, Mac, and Windows are different, so having a standard sequence allows you to test your code on multiple platforms and get the same answers on all of them.

A second new function will do a breadth first search of the tree and will do the same minimization. It's not the case that you can guarantee your code works if you get the same answers both ways, but you have a higher probability of being correct if two different functions doing two different algorithms come up with the same answer.

You might consider doing the breadth first search first, on your existing complete binary tree, and then converting your code to run on arbitrary trees.

When you test your programs, you should have three kinds of input files, one that invokes the generation of trees, one that runs the depth first search, and one that runs the breadth first search.