Xiong-Yao Zha
xzha

ECE368 Project #4 – Report

The following report will analyze the performance of the program written for the fourth programming assignment. The objective of this assignment is to reroot a packed rectangle represented by a binary tree. The code is a continuation on project 3, therefore a lot of the aspects will not be re-described in this document.

The reroot routine is coded as follow. Due to the constriction on rerooting not being able to be called from the root directly, two separate calls are needed for root->left and root->right. Then considering each subtree that is passed into the reroot function, the appropriate height and width of the other rectangle will also be assigned. The function will call a helper that will reroot, add the corresponding width and height based on the cut line of both the parent node, and new rerooted node. The function pre-orderly recursive move through the whole tree until all internal nodes are reached. Along the way, the width and height that are calculated will be compared with the previously calculated ones to find the minimum width and height that gives the smallest area. The output and runtime of the function are listed as follow:

*Testcases	Run Time	Optimal Width	Optimal Height
r0.po	0.000000E+00	1.100000e+01	8.000000e+00
r1.po	0.000000E+00	1.560376e+06	1.881875e+06
r2.po	0.000000E+00	2.622907e+06	4.923995e+06
r3.po	0.000000E+00	8.618902e+06	2.728840e+06
r4.po	0.000000E+00	9.515508e+06	1.018544e+07
r5.po	0.000000E+00	2.286623e+07	1.365289e+07
r6.po	0.000000E+00	1.300000e+01	1.100000e+01

The reroot perform rerooting in pre order fashion, and tail recursion can be removed. Thus making the space complexity O(log(n)), or the height of the tree. The time complexity of the rerooting routine is O(n), because all the internal nodes in the tree must be visited exactly once to calculate the rerooted area.

The generation of a strictly binary tree is created through the usage of stack. The process is as follows. The function will take the post-orderly printed input and create a stack that is the same size (a bit of an overkill, but doing this makes it safe to operate without invalid reads). Also, I am using array implementation of the binary tree; therefore an array of int to store the root.lcnodes/rcnodes is the logical way to do it. The program will read in the input and push nodes with cutline '-' (the rectangles). Then once it read in a cutline of either H/V, the program would pop the top element from the stack and assign it to the root's right, then pop again and assign it to root's left. Then, it will push the node into the stack. It will continue until it has exhausted the input post order sequence. The space complexity is O(n) as n stack nodes have to be created, and the time complexity is O(n) also as all input nodes have to be visited.

*Tested on Ubuntu 13.10 - MacBook Pro Retina Mid 2012