COMP1921

Programming Project - Extending the Quadtree

Jacob Holland - sc15j3h

10th March, 2016

Contents

Project Structure and Contents

Modules

File Name	**Description**	**Function Controls**
quadtreeMain.c	Contains main sequence of functions to allow execution of the quadtree functions for output/manipulation.	N/A
quadtreeFunctions.c	Contains functions used to manipulate the quadtree.	makeNode, makeChildren. printOut, writeNode, writeTree, destroyNode
qtLinkedListFunctions.c	Contains functions used to acquire leaves from a given quadtree and manipulate them.	splitDecision, enterForList, scanForLeaves, writeLeaves, makeChildrenFromLeaves
valueTree.c	Contains functions to compute the minimum and maximum values of a quadtree within a given tolerance	dataFunction, indicator

Header Files

File Name	**Description**
quadtree.h	Contains the structure of the quadtree and its related functions from "quadtreefunctions.c"
qtLinkedList.h	Contains the linked lists of leaf nodes of the quadtree, as well as the corresponding functions of "qtLinkedListFunctions.c"
valueTree.h	contains corresponding functions for valueTree.c

Source Files

Function Name	**Description**
makeNode	creates and assigns the level and coordinates of the node for the quadtree.
makeChildren	splits a selected node down a level into 4 children.
printOut	print out the coordinates of each node to a given file so it can be represented on gnuplot.
writeNode	write the given node (and its children) in a quadtree to a specified file.
writeTree	create a file and call the writeNode function to write the nodes to that specified file.
destroyNode	destroy a given node and its children in the quadtree (typically used to delete the entire quadtree).
splitDecision	walk through the linked list of leaves and decide whether or not to split the specified leaves given its parameters.
enterForList	enter a newly scanned leaf into a linked list data structure
scanForLeaves	Scan through the nodes of a linked list to find the leaves. If a leaf is found, enter it into the list using enterForList.
writeLeaves	write all leaves in the linked list to a specified file, so it can be used in gnuplot.
makeChildrenFromLeaves	subdivide all leaves in the list by 1 level
dataFunction	return a value from a set of given coordinated on a quadtree and a set of 3 possible choices.
indicator	calculate the minimum and maximum value

	and check a node to see if it falls within these paramters.

Test Plans

Task 1

Trivial Test - we should be able to successfully create children so we have a greater variety of leaf nodes to choose from and potentially add to our list. this can be tested in the “makeChildren” function in the quadTreeFunctions.c file. on line 40 of the file, a print function can be enabled to report that a child has been created, as well as its level

Non-Trivial Test - The leaves of the program should successfully be found and added to the list. This can be observed using a simple print function in the “enterForList” function (in the qtLinkedListFunctions.c file). disabling the comment on line 51 will enable a print function confirming that a node has been added to the list and what level it exists on. A sample of the terminal output will be available to view in the next section, using the output of a quadtree from Task 3 as a test to see if it can cope with a larger data set.

Task 2

Trivial Test - we should check that the program successfully subdivides all of the current leaf nodes upon execution by at least one level. we will test this by creating children at the head, then splitting one of those children down. We will then call the makeChildrenFromLeaves function and write it to the quad.out file. and load it via gnuplot. This can be tested by commenting out line 34 and removing comments from line 28.

Non-Trivial Test - We should be able to check that the program can successfully subdivide the tree more than once. we should be able to call the makeChildrenFromLeaves function for a second time and see if the number of leaves shown in GNUPlot has grown another level. This can be tested by removing comments from 28-30 and commenting out line 33.

It has become apparent that with this test, the function failed to subdivide the leaves by another level. It can only divide them by one level only. Whilst this does not affect the functionality of the the functions in task 3, it presents a bug that should be resolved in later versions. The GNUPlot outputs (for before and after the attempt) will be made available in the next section.

Task 3

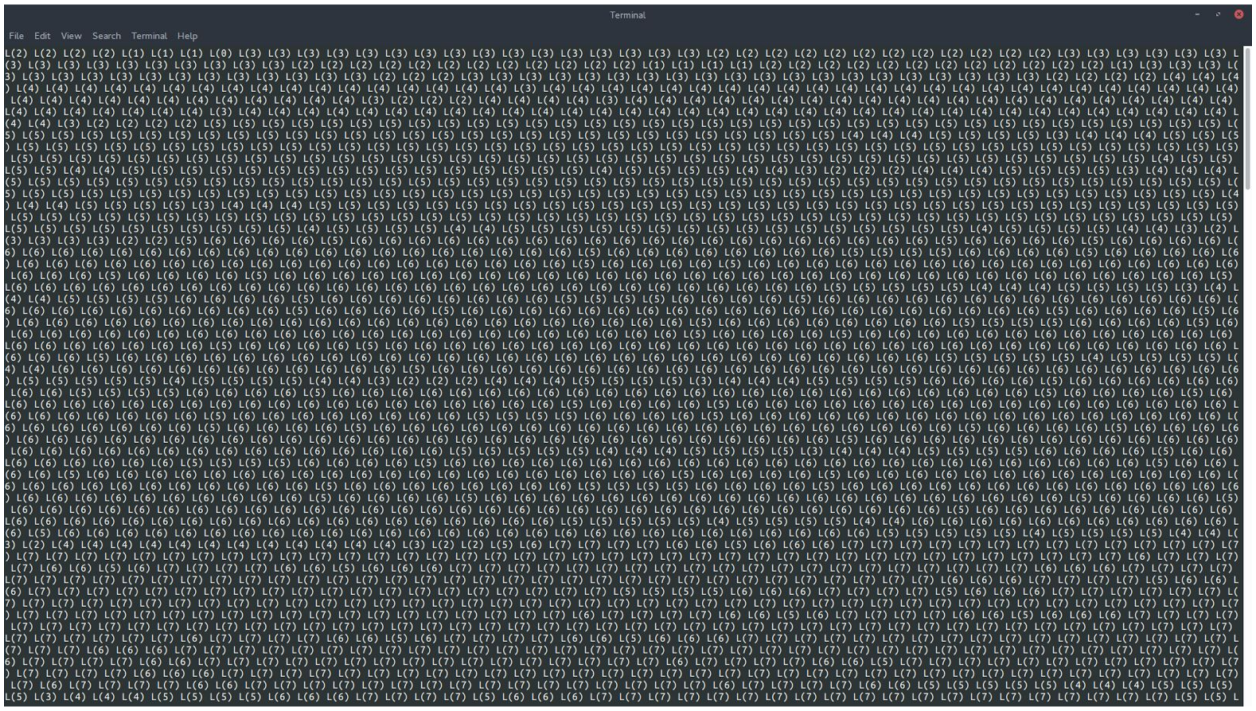
Trivial Test - In theory, we can adjust the tolerance parameter used in the indicator function to add more or less definition to our plot, due to an increase or decrease in the quantity of levels being created. Lowering the tolerance from a value such as 0.8 to 0.2 should result in a higher defined plot, as the number of leaf nodes (and corresponding children) should increase.

Non-Trivial Test - Upon execution of this program, we should test if the program has successfully written to the quad.out file as specified in earlier functions. Using the indicator function, we should be able to produce 3 distinct patterns by changing the “choice” parameter. To test this, we will call the indicator function on all nodes of the linked list via the “splitDecision” function (in the quadtreeMain file) on each different choice with a tolerance of 0.4. we will then open these files via GNUplot to test if their output is as expected.

The results for this test are in the “Test Results Section”

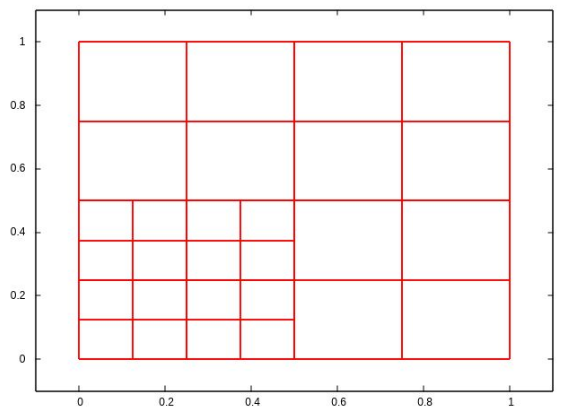
Test Results

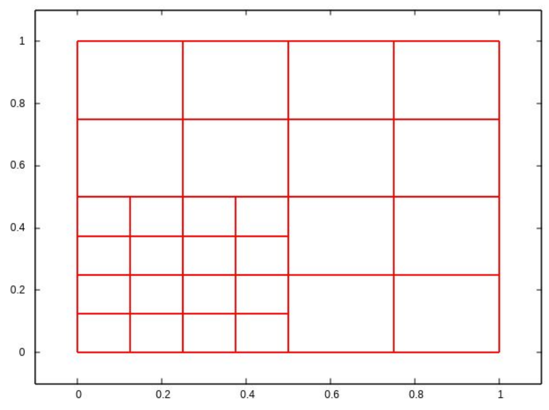
Task 1 - Non Trivial Test



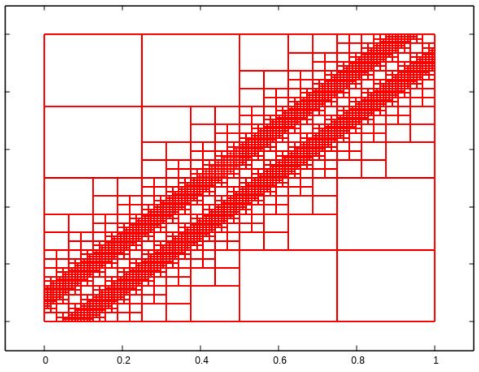
```
L(2) L(2) L(2) L(2) L(1) L(1) L(1) L(0) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(3) L(3) L(3) L(3) L(3) L(3) L
(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(1) L(1) L(1) L(1) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(2) L(1) L(3) L(3) L(3) L(
3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(2) L(2) L(2) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(3) L(2) L(2) L(2) L(2) L(4) L(4) L(4
) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(3) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4)
 L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(3) L(2) L(2) L(2) L(4) L(4) L(4) L(4) L(3) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4)
L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(3) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L
(4) L(4) L(3) L(2) L(2) L(2) L(2) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(
5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(3) L(4) L(4) L(4) L(5) L(5) L(5
) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5)
 L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(4) L(5) L(5)
L(5) L(5) L(4) L(4) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(4) L(5) L(5) L(5) L(5) L(4) L(4) L(3) L(2) L(2) L(2) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(3) L(4) L(4) L(4) L
(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(
5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(4
) L(4) L(4) L(5) L(5) L(5) L(5) L(3) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5)
 L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5)
L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(4) L(5) L(5) L(5) L(5) L(4) L(4) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(5) L(4) L(5) L(5) L(5) L(5) L(4) L(4) L(3) L(2) L
(3) L(3) L(3) L(3) L(2) L(2) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(
6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6
) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6)
 L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5)
L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(5) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(3) L(4) L
(4) L(4) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(
6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(5) L(6) L(6
) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6)
 L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6)
L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L
(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(5) L(4) L(5) L(5) L(5) L(5) L(
4) L(4) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6
) L(5) L(5) L(5) L(5) L(5) L(4) L(5) L(5) L(5) L(5) L(4) L(4) L(3) L(2) L(2) L(2) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(3) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6)
 L(6) L(6) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6)
L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L
(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(
6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6
) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6)
 L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(5) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(3) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6)
L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L
(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(
6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6
) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(5)
 L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6)
L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(5) L(4) L(5) L(5) L(5) L(5) L(4) L(4) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L
(6) L(5) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(6) L(5) L(5) L(5) L(5) L(5) L(4) L(5) L(5) L(5) L(5) L(4) L(4) L(
3) L(2) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(4) L(3) L(2) L(2) L(5) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(5) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7
) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(6) L(7) L(7) L(7)
 L(7) L(6) L(6) L(5) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(5) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(6) L(6) L(6) L(7) L(7) L(7) L(7)
L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(5) L(6) L(6) L
(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(5) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(
7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7
) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(5) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(5) L(6) L(6) L(6) L(7) L(7) L(7) L(7)
 L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7)
L(7) L(7) L(7) L(7) L(7) L(7) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(5) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(5) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L
(7) L(7) L(7) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(
6) L(7) L(7) L(7) L(7) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(5) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7
) L(7) L(7) L(7) L(7) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7)
 L(7) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(6) L(7) L(7) L(7) L(7) L(6) L(6) L(5) L(5) L(5) L(5) L(5) L(4) L(4) L(4) L(5) L(5) L(5)
L(5) L(3) L(4) L(4) L(4) L(5) L(5) L(5) L(5) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(5) L(6) L(6) L(6) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(7) L(5) L(5) L
```

Task 2 - Non Trivial Test (Before on the left, after on the right)

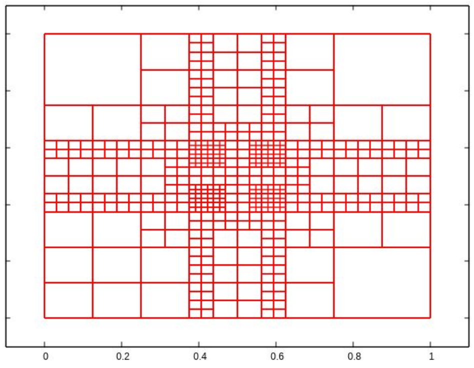




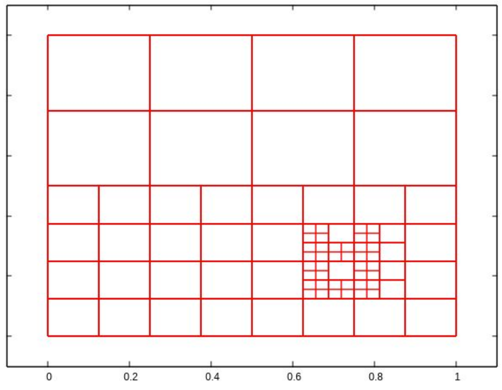
Task 3 - Non Trivial Test



Choice 0 at Tolerance = 0.4



Choice 1 at Tolerance = 0.4



Choice 2 at Tolerance = 0.4