COMP1521: Quadtree Report

i.) Structure of the code

Modules:
main.c – Contains the main functions that run the processes of the application.
node.c – Contains any code that relates to creating the nodes of the quadtree.
output.c – Contains any code related to printing out the quadtree.
valueTree.c – Contains code relating to task four.

Functions:
main.c

node.c
output.c
valueTree.c

Header Files:
functions.h – Contains all the functions needed by the other c modules.
structs.h – Contains the structs to create the nodes.

ii.) Tasks

Task 1: A limit on tree level

(i) Looking at the node struct in the code provided, a calculation for the size of the node can be determined:

int: 4 bytes
double: 8 bytes * 2 (due to the array size) = 16 bytes
pointer: 8 bytes * 4 (due to the array size) = 32 bytes
total: 52 bytes

One single node is approximately 52 bytes and to find out the maximum size of each level would be 52 bytes * 4^a, where a is the level of the quadtree.

Therefore a table of memory that would be used for a full tree between five and ten levels is as follows:

Quadtree Level	Maximum Number of Nodes	Size of Each Level (Megabytes)
Level 5	1024	0.1
Level 6	4096	0.2
Level 7	16384	0.9
Level 8	65536	3.4
Level 9	262144	13.6
Level 10	1048576	54.5

If the limit for memory is 20 Megabytes, then the maximum level that should be chosen would be the ninth, as shown in the table above.

(ii) The maximum level parameter is stored in the node.c file, under the makeChildren(Node *parent) function.

An if statement was used to limit the program to nine levels since once the program reached the first node of the tenth level, an error message was printed and the program would close down. However if the number of levels was lower than ten, the makeChildren(Node *parent) function would run normally and add children to any leaf nodes specified.

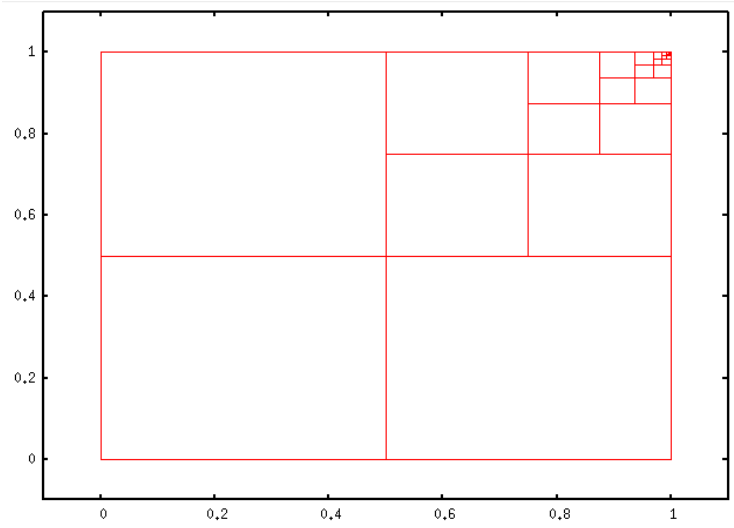
Testing

The test used was to increase the level of a specific branch of the tree with the makeChildren function and print out the results in Gnuplot.

If nine levels or less were created the program be would expected to execute successfully and create the quad.out file that was needed by Gnuplot to print out the quadtree.

```
int main( int argc, char **argv )
    {

    // create the head node:  level 0
    Node *head = makeNode( 0.0,0.0,0 );

    // split to level 1
    makeChildren( head );

    // split one node to level 2
    makeChildren( head->child[2] );

    // split one node to level 3
    makeChildren( head->child[2]->child[2] );

    // split one node to level 4
    makeChildren( head->child[2]->child[2]->child[2] );

    // split one node to level 5
    makeChildren( head->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 6
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 7
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 8
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 9
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2] );

    // write entire tree to quad.out
    writeTree( head );
```

```
[sc16wyrw@comp-pc6035 quadtree]$ make -B
cc -O -c -o main.o main.c
cc -O -c -o node.o node.c
cc -O -c -o output.o output.c
cc -O -c -o valueTree.o valueTree.c
cc main.o node.o output.o valueTree.o -lm -o ./quadtree
[sc16wyrw@comp-pc6035 quadtree]$ ./quadtree
[sc16wyrw@comp-pc6035 quadtree]$ gnuplot

        G N U P L O T
        Version 4.6 patchlevel 2    last modified 2013-03-14
        Build System: Linux x86_64

        Copyright (C) 1986-1993, 1998, 2004, 2007-2013
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type set to 'x11'
gnuplot> load 'quad.gnu'
```



However, if nine levels were exceeded, then the should program print out an error message in the terminal and would not create a quad.out file, thus Gnuplot was unable to find the quad.out to plot the tree.

```
int main( int argc, char **argv )
    {

    // create the head node:  level 0
    Node *head = makeNode( 0.0,0.0,0 );

    // split to level 1
    makeChildren( head );

    // split one node to level 2
    makeChildren( head->child[2] );

    // split one node to level 3
    makeChildren( head->child[2]->child[2] );

    // split one node to level 4
    makeChildren( head->child[2]->child[2]->child[2] );

    // split one node to level 5
    makeChildren( head->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 6
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 7
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 8
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 9
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2] );

    // split one node to level 10
    makeChildren( head->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2]->child[2] );

    // write entire tree to quad.out
    writeTree( head );
```

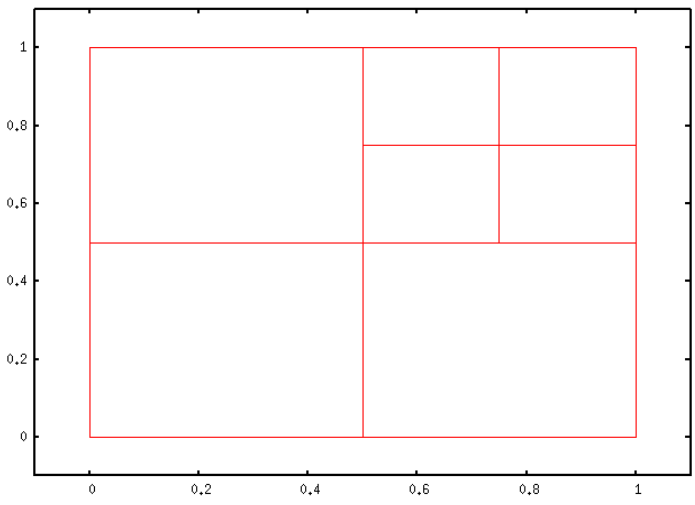
```
[sc16wyrw@comp-pc6035 quadtree]$ make -B
cc -O -c -o main.o main.c
cc -O -c -o node.o node.c
cc -O -c -o output.o output.c
cc -O -c -o valueTree.o valueTree.c
cc main.o node.o output.o valueTree.o -lm -o ./quadtree
[sc16wyrw@comp-pc6035 quadtree]$ ./quadtree
Error: Program terminated because maximum level of tree exceeded
```

The results of the tests show that the maximum level that the quadtree can form is nine levels and when it exceeds this the program closes. This is the same as our expected result.

Task 2: Growing the quadtree

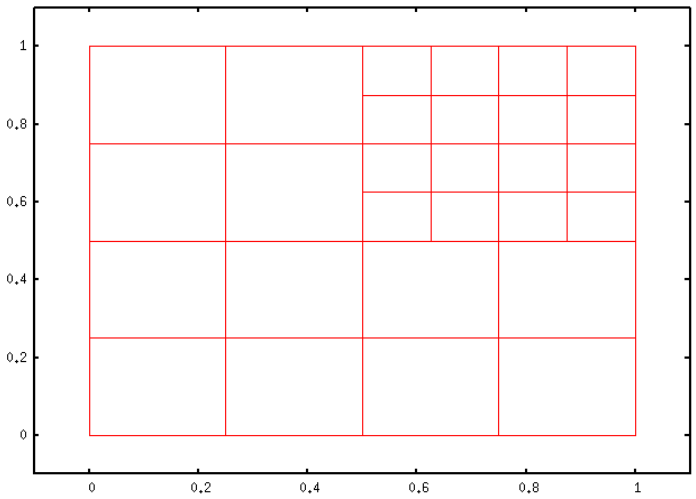
The test used was to compare two different graph plots using Gnuplot. The first plot was to print out the quadtree with no alterations to the main file. The results can be seen below.

```
int main( int argc, char **argv )
    {

    // create the head node:  level 0
    Node *head = makeNode( 0.0,0.0,0 );

    // split to level 1
    makeChildren( head );

    // split one node to level 2
    makeChildren( head->child[2] );

    // write entire tree to quad.out
    writeTree( head );

    // delete entire tree
    destroyNode( head );

    return 0;
    }
```



Afterwards the function growTree was added, so the program should still execute and produce a quad.out, providing nine levels was not exceeded. This file would then be loaded into Gnuplot and should show that every node has been extended by one.

```
int main( int argc, char **argv )
    {

    // create the head node:  level 0
    Node *head = makeNode( 0.0,0.0,0 );

    // split to level 1
    makeChildren( head );

    // split one node to level 2
    makeChildren( head->child[2] );

    //grow all leaf nodes by one
    growTree( head );

    // write entire tree to quad.out
    writeTree( head );

    // delete entire tree
    destroyNode( head );

    return 0;
    }
```



It is clear from the results that the quadtree has been extended uniformly with all nodes being split further, as shown on Gnuplot.

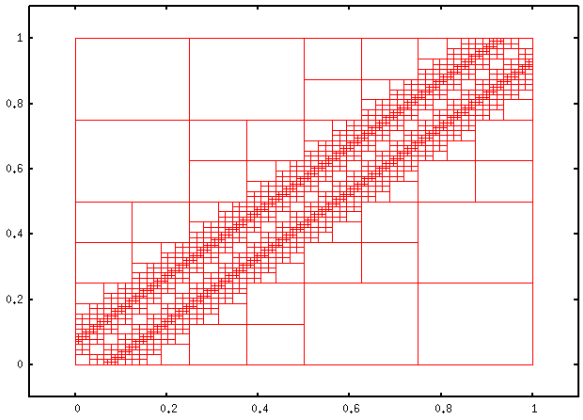
Task 3: The leaf node list

Task 4: Generating a data-dependent Quadtree

Testing

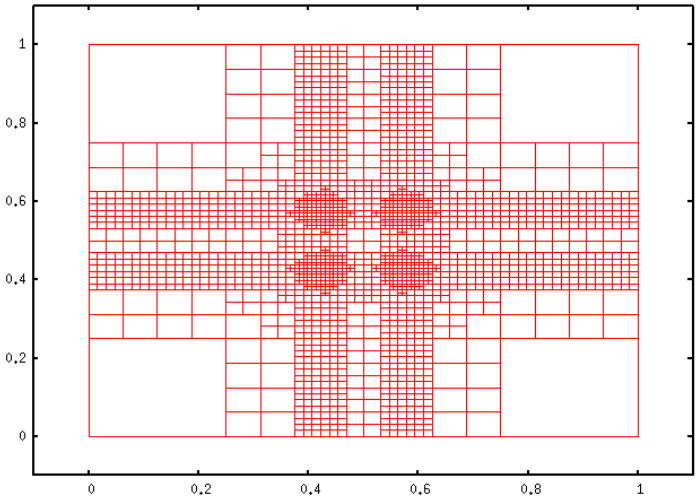
Choice=0 and tolerance=0.5 were the values that were given to us.

```
void searchTree(Node *node)
{
    int i;
    if( node->child[0] == NULL )
        {
        if( indicator (node, 0.5, 0) == false )
            {
            makeChildren( node );
            }
        }
    else
        {
        for ( i=0; i<4; ++i )
            {
            searchTree(  node->child[i] );
            }
        }
    return;
}
```



Choice = 1 and tolerance = 0.2 were the next values tested

```
void searchTree(Node *node)
{
    int i;
    if( node->child[0] == NULL )
        {
        if( indicator (node, 0.2, 1) == false )
            {
            makeChildren( node );
            }
        }
    else
        {
        for ( i=0; i<4; ++i )
            {
            searchTree(  node->child[i] );
            }
        }
    return;
}
```



Reflection