EEE2048: Computer Algorithms and Architecture
2019/20 Programming Assignment
Quadtree Encoding of a Binary Image Technical Report
Samuel Awonuga 6564365
University of Surrey

Explanation of functions

powerOfTwo

- This function takes a given argument 'x', and checks if the value is a power of 2.
- This works by getting the argument passed and continuously dividing it by 2 until the result of the division is close 1
- If continuous division by 2 results in 1, and there no remainder, then 1 is returned and the function is a power of two

validPic

- This function checks if the values required to draw the grid specified in the image file are valid.
- The function passes the width to the function 'powerOfTwo' to check if the width is a power of 2.
- The function also specifies the minimum pixel width as 2 and maximum pixel width as 16.
- The function then compares the width of the pixels to the minimum and maximum and checks if its within range. The function would also look to see if the function is a power of two and would terminate the program if it is not.

Variables	Data type	Description
widthOfPixels	Integer	The argument should contain the width of the image that is required to be checked. This argument is then stored in the widthOfPixels to be manipulated in the function
powerofTwo	Integer	This variable stores the return value of the function call powerOfTwo. This function call passes the parameter 'widthOfPixel' as an argument to the powerOfTwo function, and once a power of 2 has been calculated, the return value is stored in this variable.
minPixelWidth	Integer	This sets the minimum pixel width the program can manipulate as 2, anything below 2 would not be able to be split into quadrant, and thus program would prompt a message to user and terminate
maxPixelWidth	integer	This sets the maximum pixel width the program can manipulate as 16, anything above 16 would not be result in the program prompting a message to user and terminating

colourCheck

- This function does all the checks for the colours in grid.
- This is done by using a nested for loop to check all the coordinates of the grid from 0 to the stated maximum pixel width for both the x and y axis. Every time a black pixel is found, it increments a variable 'sumOfBlacks' to find the total number of black pixels.
- If the sum of the black pixels is equal to the total number of pixels, then the return value indicates the quadrant is all black.
- If the sum of black pixels is less than max pixels, then another pixel is present and if sum of black pixels is equal to 0, then the image is fully white

Variables	Data type	Description
maxPixelWidth	Integer	This stores the total number of pixels in the grid. This is done by squaring the argument passed as pixel width
sumOfBlacks	Integer	This is used to get the total number of black pixels present in the given grid.
x	Integer	This is used as a stepper variable in the for loop when scanning for a black pixel
y	integer	This is another stepper variable in the for loop used when scanning for a black pixel

displayBlackNode

This function would check where are the black nodes are present in the structure, and then displays the co-ordinates of the black nodes to the user

addNode (recursive function)

This function checks if another colour is present in the image grid by calling the colourCheck function. If another colour is present, it divides the image grid into quadrant and recursively call itself again. The recursive call is done until there is, we get a grid where all the pixels match. This is the essentially the base case

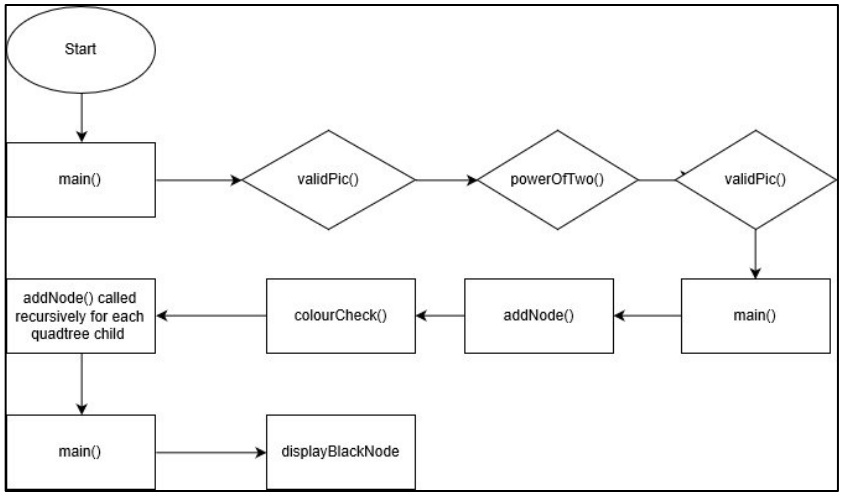
Variables	Data type	Description
returnedColour	Integer	This variable stores the return value of colourCheck. A value of 2 indicates that there are mixed pixels in the image, and therefore the grid needs to be divided again into quadrants to find position of black pixel

Main

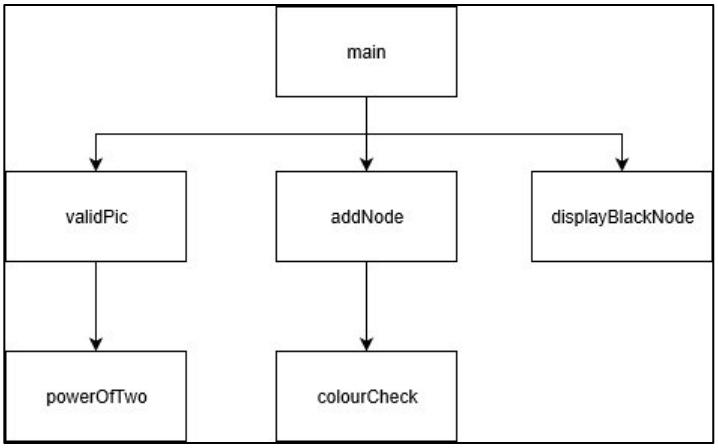
The main function deals with reading from a given input file, this part gets the width of the file and the number of black pixels from the file. Once the width, co-ordinates of black pixels and number of black pixels has been given, an image grid that maps to the coordinate specified is created. This functions also calls another function called 'validPic' to check whether the input file is correct in its setting

variables	Data type	Description
widthFromFile	Integer	This is used store the width of image that is read from the file
numOfBlackPixels	Integer	This is used to get the total number of black pixels from the file
fptr	Integer	This is a file pointer variable used to store the content
picFrame	Integer	This stores the return value of the validPic function to check if the function
grid	Integer array	This is used to store the canvas of the image
maxPixelWidth	Integer	This is used to store the maximum pixel width
xFromFile	Integer	Variable should be x coordinate specified in the file
yFromFile	Integer	Variable should be y coordinate specified in the file
x	Integer	Stepper variable used to loop through grid in x axis
y		Stepper variable used to loop through grid in y axis

Flow of program

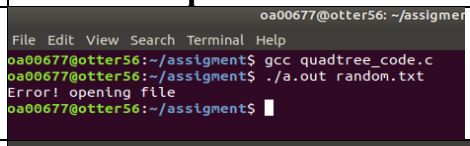
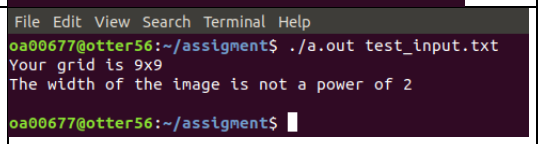
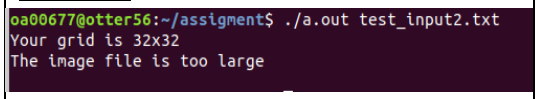
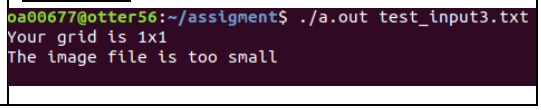


Hierarchy chart



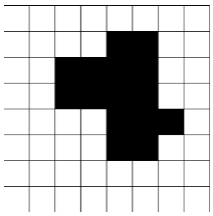
Validations and testing

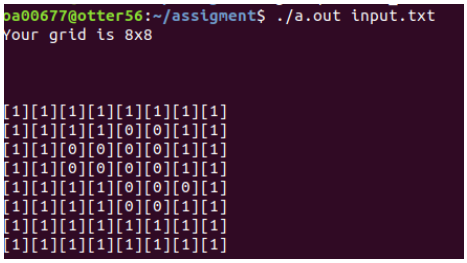
A testing section presenting example results of the run of the program. This should demonstrate the correct operation of the program on a range of pertinently chosen input test images.

Test	Expected Output		Test description	Actual Output
There is no file with the name the user passed as an argument	“Error! opening file"		If an attempt to open the file passed as an argument during compilation returns NULL, the program displays the following output and then exits	
Image width provided in the file is not a power of 2. To test this an input file other that is not 2 is presented in the test_input.txt file	"The width of the image is not a power of 2"		The function reads the first line of the file and stores the width in the integer variable “widthFromFile”, this variable is passed as a parameter to the validPic function, this function would check if pic is a power of 2 and returns 0 if it is not. A return value of 0 in this section of the code would terminate the program	
Image should be within the specified maximimum and minimum range. Maximum is 16 and minimum is 2	If the pixel is less than minimum pixel width	"The image file is too small	After given the width of pixel from file, this is compared to given maximum and mimum specified in the validPic function, and if the width doesn’t meet requirement the return value is 0 and the program is expected to terminate	**Too small**  **Too large** 
	If the pixel is more than maximum pixel width	The image file is too large		

Checking for correct values

To check if a correct output that matches the input file is given, I added code to print out the grid in the program, and compare results with the file. This test check was successful





Checking for correct behaviour of quad tree

to check if the program continuously divides by 4 if a mixed colour is found, I added a puts statement in the colourCheck that outputs when another colour is found. This number should divide the maximum pixel width by 4

Complexity

To do this part, I added the "time.h" header library and measured the execution of the quadtree per loop. I also changed the maximum pixel size to 64 in the validPic() function

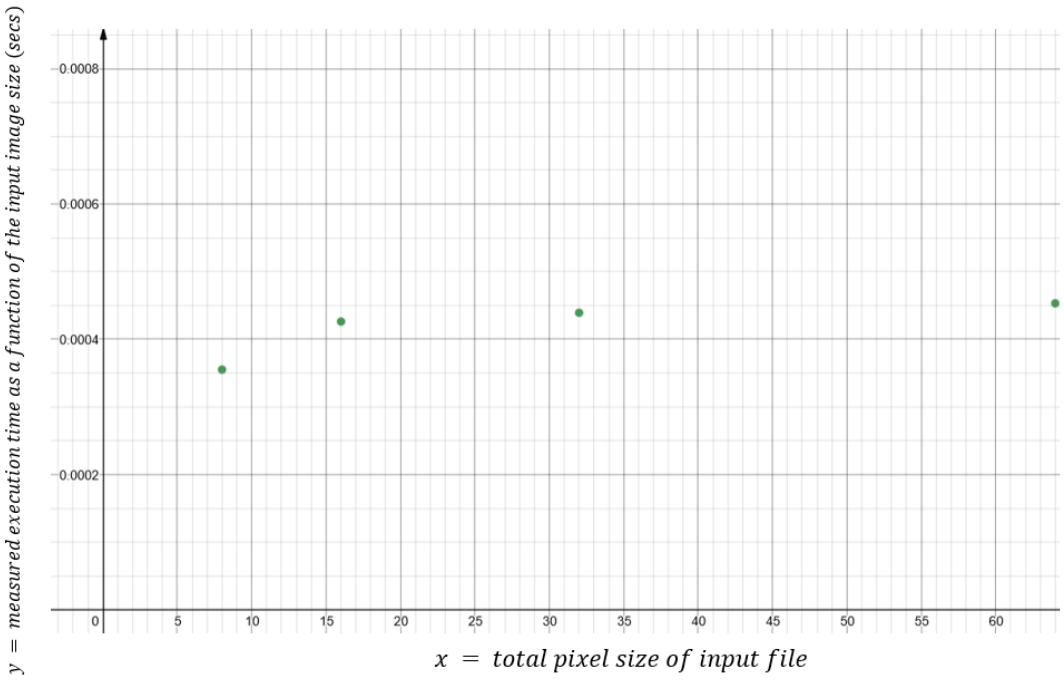
The code is as follows below

```
clock_t start = clock();
int stepper;
for (stepper=0; stepper<LARGE_NUMBER; stepper++){

      //creating values for the quadtree
      quadTree showQuadtree = addNode(0,0,maxPixelWidth, grid);
      displayBlackNodes(showQuadtree);


}clock_t end = clock();
printf("Running-time (secs) for %d repeats: %f\n", LARGE_NUMBER, (double) (end-start) / (double) CLOCKS_PER_SEC);


printf("\n");
```

In the table below, x, the independent variable, represents the pixel size and the y, the dependent variable, represents run-time. From the graph below, we can see that on average, there seems to be an increase in the measure execution with size of input file

$Input\ file\ size(x)$					$Average\ time(y)$
8	0.000065	0.000067	0.000065	0.000101	0.00035525
16	0.000391	0.000372	0.000428	0.000515	0.00042625
32	0.000583	0.000567	0.000429	0.000178	0.00043925
64	0.000469	0.000328	0.000464	0.00552	0.00045325

Graph plot



We can see that the graph roughly resembles an O(logn) graph, this is seen in the steep increase in complexity from around 8-16-pixel size, and then slowed increase in around 16-64. This resembles the following complexity of a O(logn) function