6261263

4 April 2015

# Programming Assignment Report

## Quadtree Encoding of a Binary Image

This project was to create a program that used a quadtree structure to encode a binary image. The program was programmed in the C language. The program is passed a file containing the image width, number of black pixels and their coordinates, this file is then analysed by the program and the encoded black pixels are printed, the rest of the pixels are white.

By encoding a binary image it saves space when storing the compressed image in memory. This is because not every single pixels value (black or white) needs to be stored; instead groups of pixels that share the same pixel value are stored as one.

To use the program the user must first open the command line and locate the program file (normally done by cd Desktop/quadtree etc and then ls to show files). The user needs to make sure the file to be passed to the program is saved in the same location. The c code needs to be compiled by typing "**cc -c main.c**" and pressing enter and then "**cc -lm main.o -o main**" and enter again. Then to run the quadtree program the user must type "**./main [input.txt]**" where [input.txt] is the file name of the file to be passed to the program and then press enter. The program will now analyse the inputted file and print out the encoded black pixels.

### Method

The program can be split into 3 main parts:

- Reading passed file- The passed file is read by the program and stored with the node structure as at the root node in memory.
- Analysing image- The root node is recursively analysed in quadrants to determine if the quadrant is black, white or mixed, this is stored as a node on the quadtree. If the quadrant is mixed the function is called again, this time with the mixed quadrant as the node to be split into quadrants and analysed. This continues until all the broken down quadrants are either black or white.
- Printing results- The black nodes bottom left coordinate and width is printed.

The program has some restrictions to run normally. The restrictions are that the image must be binary (the pixels must be either 1 or 0, i.e black or white) and that the image width must be a power of two. If these restrictions are not met the quadtree structure will not work and a new method of image encoding needs to be used. According to the specification the image width has a maximum of 64 pixels, this is defined at the start of the code and can easily be changed if the specification changes.

All these restrictions are checked during the program running and if the passed file does not comply, an error message is printed explaining the problem and then the program closes. The program also implements other checks at the beginning to save run time.

The checks that implemented are:

- A file is passed to main when the program is called. ***"Error with input file. Program exit"***
- The file that is passed is opened and there is content in the file. ***"Error reading file. Program exit"***
- That the file can be stored in a string. ***"Error Storing file. Program exit"***
- Image width provided in the file is not a power of two. ***"The image width is not a power of 2. Program exit"***
- Image width provided in the file is greater than the maximum image width that is defined in the code or the image width provided in the file is less that the minimum image width that is defined in the code. ***"The image is too small/large: it is less/more than the minimum/maximum image width. Program exit"***
- Number of black pixels provided by the file is more than the number of pixels in the image ***"There are more black pixels than the size of the image. Program exit***"
- Number of black pixels provided by the file is the same as the number of pixels in the image therefore the image is all black ***"The image is only black pixels"***
- Number of black pixels provided by the file is zero therefore the image is all white ***"The image is only white pixels"***
- Number of black pixels provided by the file is less than zero, this would be a typing error as this would never be a possible number of black pixels. ***"There are less than 0 black pixels. Program exit"***
- The X or Y value of the coordinate read from the file is not within the boundary from the pixel width, i.e the value is less than 0 or greater than the pixel width. ***"Coordinate (x,y) is outside of the bounds of the image width. Program exit"***
- Number of black pixels provided by the file is not the same as the number of black pixels counted by the program ***"The given number of black pixels is not the same as the counted number of black pixels. Program exit"***

When the program was being programmed, tested lines were added to the program at each stage. The program was then run we test images and the outputs were compared with expected outputs. If they did not match, the program was reviewed, edited and reran until it functioned normally.

## Functions

**Main**

The main function is the designated start of the program which calls all the following functions in the program. The main is passed the file which is identified when the program is called.

**ReadFile**

This is the first function called by main(). This function is passed the input file from main. The function starts by reading the file and dynamically allocating memory for a buffer based on the size of the file and then stores the content of the file in the buffer. Next the program reads through and stores the relevant information starting with the PixelWidth and then the NumberofBlack and then the coordinates. Memory is dynamically allocated based on the PixelWidth to store the values of the coordinates; at first all the pixels are made white but

when the program reads the coordinates it changes the relevant pixels to black. The node is then set using the function SetNode. At the end of the function the array and buffer are freed and the file is closed.

The function makes multiples checks on the information provided to make sure the information is within boundaries, to save time and make sure the program can function normally. (See the checks above).

**SetNode**

This function is used to set information passed to the function on the node that is passed. The first parameter is the node that the information will be set on. The PosX, PosY, Width, Value is copied to the passed node. Next the four child nodes memory is allocated. Finally memory for the pixel array is dynamically allocated based on width and then the 2d Pixel array passed to the function is copied to the new PixelArray.

**PowerOfTwo**

The power of two function is used to check if x which is passed to the function is a power of two. The function uses a while loop which loops checking x is still even and that x is greater than 1, whilst each loop it divides x by 2. When the loop finishes it returns the value of x. This is then used in the calling function to check x is equal to one.

**BuildQuadTree** (**Recursive function**)

This functions only parameter is node being passed to it. At the start of the function there are a few calculations for variables which are used later. The main part of the function is the for loop which switches between 4 cases; each case is for a different quadrant of the pixel array. The case calls the CheckColour function to find out if that quadrant is Black, White or Mixed and copies that to Value. The temporary array for that quadrant is then created. After a few more calculations to find the coordinates for that quadrant the node is set with the relevant values and the array and then the temporary array is freed. If value is black then the coordinates and width is printed and this case breaks. If value is mixed the BuildQuadTree function is called recursively, this time with the child of the node as the root node. Finally if value is white nothing is called or printed and this case breaks.

**CheckColour**

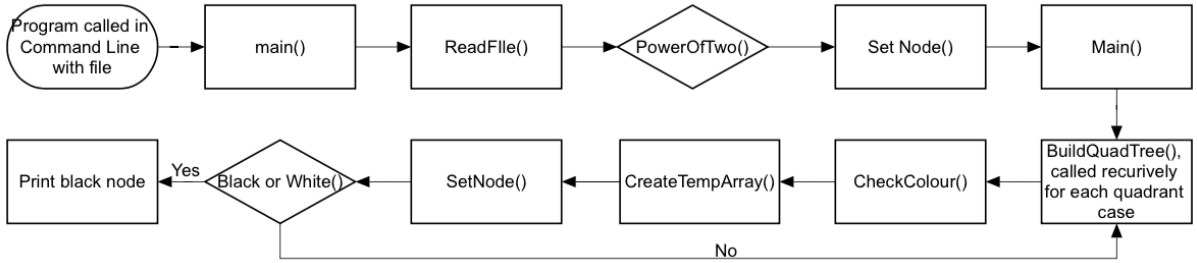
This function takes the node that is being passed to it and between top, bottom, left and right, it counts the number of black pixels and white pixels, then if there are no white pixels it returns black, if there are no black pixels it returns white and if there are both lack and white pixels it returns mixed.

**CreateTempArray**

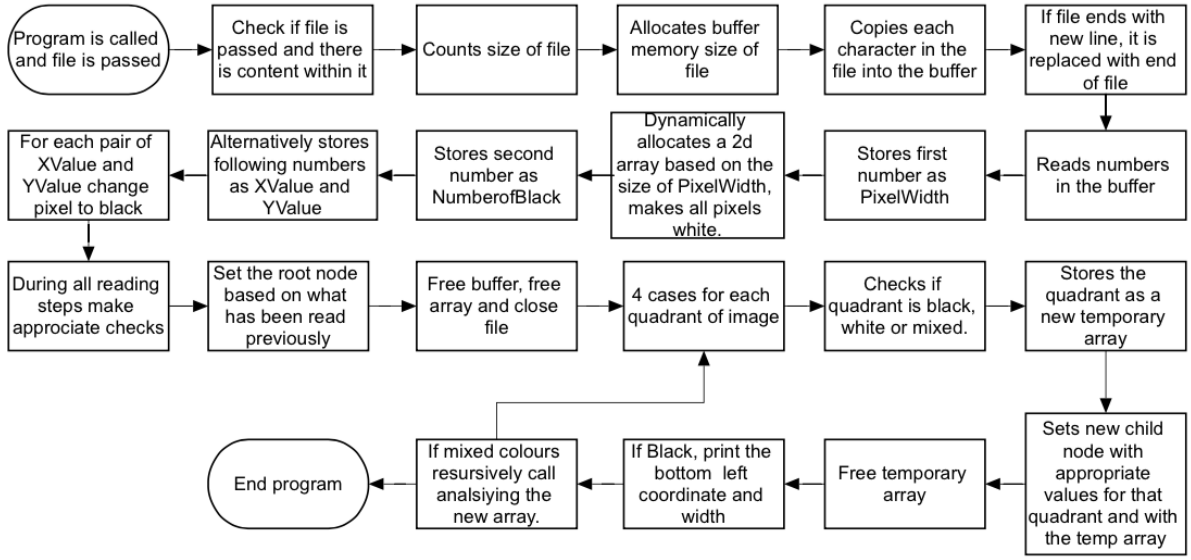
This function creates a temporary array for the quadrant that has called the function and copies the array values from that quadrant into the temporary array. First the function dynamically allocates a 2d array in memory for the width of the quadrant. The second half of the function copies the quadrant pixel values into the new temporary array, because they have have different coordinates this has to be done with two for loops and a counter inside the for loops. The function returns the new temporary array.

# Structural Flow Chart

## Functions




## Processes

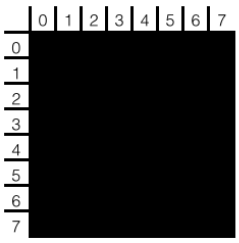


## Results and Data Findings

The black nodes are printed in the form: "***Position (x,y), Width z, is all black"*** where x,y are the bottom left coordinates and z is the width of the node.

**Example outputs:**

**Input:**

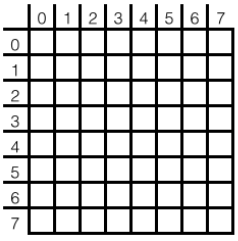


```
8
64
**All
nodes
```

**Output:**

```
The image is only black pixels.
```

**Input:**

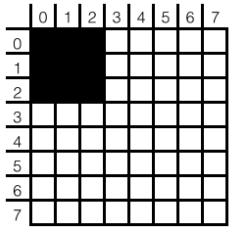


```
8
0
```

**Output:**

```
The image is only white pixels.
```

**Input:**

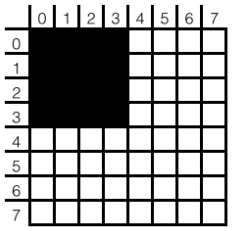


```
8
9
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

**Output:**

```
Position (0,1), Width 2, is all black
Position (2,0), Width 1, is all black
Position (2,1), Width 1, is all black
Position (0,2), Width 1, is all black
Position (1,2), Width 1, is all black
Position (2,2), Width 1, is all black
```

**Input:**



```
8    2 1
16   2 2
0 0  2 3
0 1  3 0
0 2  3 1
0 3  3 2
1 0  3 3
1 1
1 2
1 3
2 0
```

**Output:**

```
Position (0,3), Width 4, is all black
```

**Input:**



```
7    4 4
13   4 5
1 1  5 4
1 2  5 5
2 1
2 2
2 3
3 2
3 3
3 4
4 3
```

**Output:**

```
The image width is not a power of 2.
Program exit

   (if the width was 8, see below)
Position (1,1), Width 1, is all black
Position (2,1), Width 1, is all black
Position (1,2), Width 1, is all black
Position (2,3), Width 2, is all black
Position (4,3), Width 1, is all black
Position (3,4), Width 1, is all black
Position (4,5), Width 2, is all black
```

**Input:**



```
8    5 3
15   4 4
4 1  5 4
5 1  6 4
2 2  4 5
3 2  5 5
4 2
5 2
2 3
3 3
4 3
```

**Output:**

```
Position (2,3), Width 2, is all black
Position (4,1), Width 1, is all black
Position (5,1), Width 1, is all black
Position (4,3), Width 2, is all black
Position (4,5), Width 2, is all black
Position (6,4), Width 1, is all black
```

**Input:**



```
8    2 2
25   2 3
0 0  2 4
0 1  3 0
0 2  3 1
0 3  3 2
0 4  3 3
1 0  3 4
1 1  4 0
1 2  4 1
1 3  4 2
1 4  4 3
2 0  4 4
2 1
```

**Output:**

```
Position (1,1), Width 1, is all black
Position (2,1), Width 1, is all black
Position (3,1), Width 1, is all black
Position (1,2), Width 1, is all black
Position (1,3), Width 1, is all black
Position (2,3), Width 2, is all black
Position (4,1), Width 1, is all black
Position (5,1), Width 1, is all black
Position (4,3), Width 2, is all black
Position (1,4), Width 1, is all black
Position (1,5), Width 1, is all black
Position (2,5), Width 2, is all black
Position (4,5), Width 2, is all black
```

**Input:**



```
8    4 3
16   4 4
2 2  4 5
2 3  5 2
2 4  5 3
2 5  5 4
3 2  5 5
3 3
3 4
3 5
4 2
```

**Output:**

```
Position (2,3), Width 2, is all black
Position (4,3), Width 2, is all black
Position (2,5), Width 2, is all black
Position (4,5), Width 2, is all black
```

## Conclusion

The program functions correctly and adheres to the specification. The program analyses the image quickly and efficiently. Variables have been worded sensibility, functions has been used correctly, key numbers have been defined so the code is easier to understand and when needed, memory has been allocated efficiently and freed subsequently. Many checks have been added to make the program quicker and more stable. The program prints the required output in a clear form.

In the future if the program needed to be changed it would be easy for someone to read through the code and make the necessary changes.