Programming Assignment Report

Quad tree Encoding of a Binary Image

Table of Contents

Programming Assignment Guide

1 Overview of the Program Design

This project was to create a program that used a quad tree 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 analyzed 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/quad tree 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. I have used two files first is solution.c which contains the definition of all the functions plus the main function and the second file which is solution.h which acts as the header file and contains declaration of functions, structure of Treenode and the constants that is defined and used in the program. The c code needs to be compiled by typing "g**cc solution.c –o quadtree**" and press enter. In this case, we use the -o option to specify a different output file for the executable. Then to run the quad tree program the user must type "**./quadtree [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 analyze the input 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.
• **Analyzing image -** The root node is recursively analyzed in quadrants to determine if the quadrant is black, white or mixed, this is stored as a node on the quad tree. 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 analyzed. 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 quad tree structure will not work and a new method of image encoding needs to be used. According to the specification the image width has a minimum of 1 pixel, 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.

1.1 Functions and Their Descriptions

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.

Read File

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 Image width and then the number of black pixels and then the coordinates. Memory is dynamically allocated based on the Image width 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 create Node. 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.

Create Node

This function is used to set information passed to the function on the Treenode that is passed. The first parameter is the node that the information will be set on. The XPosition, YPosition, 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.

Power Of Two

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

Build Quad Tree (Recursive Function)

This function only parameter is Treenode 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 while loop which switches between 4 cases; each case is for a different quadrant of the pixel array. The case calls the Check Colour 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 breaks the case.

Check Colour

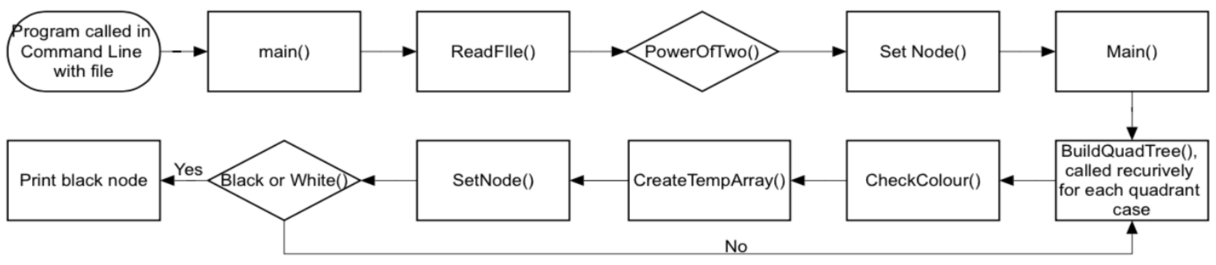
This function takes the node that is being passed to it and between top, bottom, left and right, it counts the number of black pixels and white pixels, then if there are no white pixels it returns black, if there are no black pixels it returns white and if there are both black and white pixels it returns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

Create Temporary Array

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 according to width. The second half of the function copies the quadrant pixel values into the new temporary array, because they have different coordinates this has to be done with two while loops. The function returns the new temporary array.

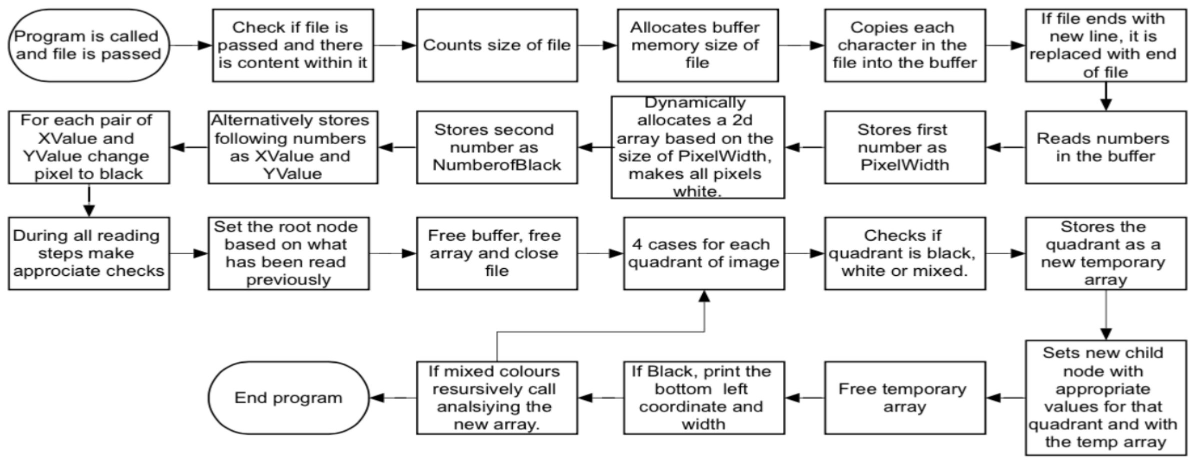
1.2 Structural Flow Chart

Functions



*In the Above chart Set Node Function is written as Create Node in the code.

Processes

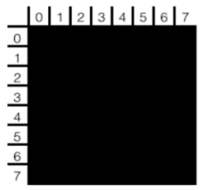


Program is called and file is passed
Check if file is passed and there is content within it
Counts size of file
Allocates buffer memory size of file
Copies each character in the file into the buffer
If file ends with new line, it is replaced with end of file
Reads numbers in the buffer
Stores first number as PixelWidth
Dynamically allocates a 2d array based on the size of PixelWidth, makes all pixels white.
Stores second number as NumberofBlack
Alternatively stores following numbers as XValue and YValue
For each pair of XValue and YValue change pixel to black
During all reading steps make approciate checks
Set the root node based on what has been read previously
Free buffer, free array and close file
4 cases for each quadrant of image
Checks if quadrant is black, white or mixed.
Stores the quadrant as a new temporary array
Sets new child node with appropriate values for that quadrant and with the temp array
Free temporary array
If Black, print the bottom left coordinate and width
If mixed colours resursively call analsiying the new array.
End program

2 Testing Section with Results

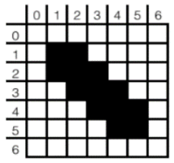
The black nodes are printed in the form: “***Position (x,y), Width z, is all black”*** where x and y are the bottom left coordinates and z is the width of the node.

Input: **Output:** **Input:** **Output:**



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

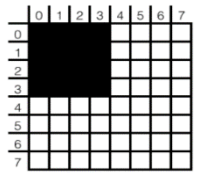
```
The image is only black pixels.
```



```
7     44
13    45
11    54
12    55
21
22
23
32
33
34
43
```

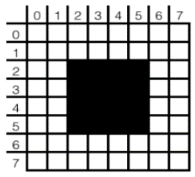
```
The image width is not a power of 2.
Program exit

   (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
```



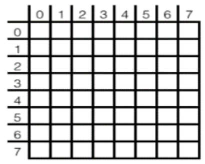
```
8     21
16    22
00    23
01    30
02    31
03    32
10    33
11
12
13
20
```

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



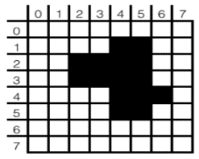
```
8     43
16    44
22    45
23    52
24    53
25    54
32    55
33
34
35
42
```

```
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
```



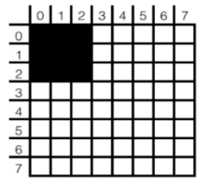
```
8
0
```

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



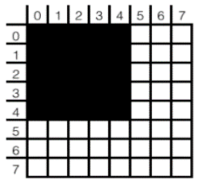
```
8     53
15    44
41    54
51    64
22    45
32    55
42
52
23
33
43
```

```
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
```



```
8
9
00
01
02
10
11
12
20
21
22
```

```
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
```



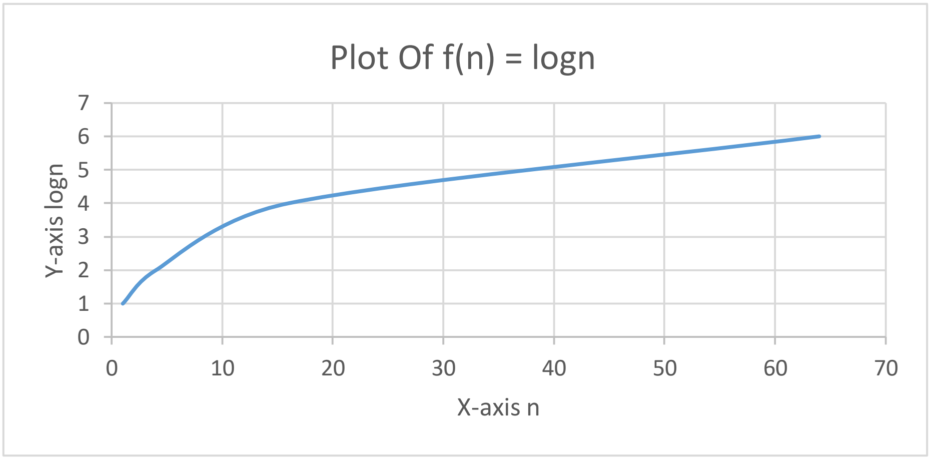
```
8     22
25    23
00    24
01    30
02    31
03    32
04    33
10    34
11    40
12    41
13    42
14    43
20    44
21
```

```
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
```

3 Computational complexity Of Algorithm

The create Node function is used to create a new node into an existing Quad Tree. This function checks whether the given node has the same color or mixed. If it is the, then we immediately break out from that case. If it is the mixed, we select the appropriate child to contain this node based on its location. This function is O(Log N).

The Logarithm function f(n) = logn. Logarithm function gets slighty slower as n grows.


Plot Of f(n) = logn
Y-axis logn
X-axis n
0
1
2
3
4
5
6
7
0
10
20
30
40
50
60
70