Quadtree Encoding of a Binary Image

EEE2048: Computer Algorithms and Architecture, 2018/19 Programming Assignment

Shivaum Mehta

02/12/2018

University of Surrey

# Introduction

## Abstract:

My program is designed to compress a black and white image of maximum size 64 x 64 through the use of a quadtree data structure, it will then print out all if the nodes of the black pixels as well as their size.

## Summary:

A quadtree is a data structure that contains a root node with four child nodes which split the image into four quadrants (North-West, North-East, South-West, South-East) and repeats this process with the child nodes as many times as needed until all of the nodes contain only one colour. This means that the image is compressed to the coordinate of a black node as well as its size.

My approach to this problem was to split it into 5 sub-stages:

1. Take input from a file provided by the used at runtime.
2. Read data from the file and validate it
3. Store the data in a dynamically allocated 2D array
4. Use a quadtree structure to locate black nodes in the 2D array
5. Print out the position and sizes of each black node in order

## Restrictions:

There are several assumptions and restrictions made on the use of the program, these include:

* Colour restrictions – the program is restricted to only using black and white images and will not be able to process images with other colours as well
* Size limit – the width of the image is limited to 64 pixels and can only be a power of 2 as any other will not divide so nicely
* Scale – at the moment the program can only handle images with a 1:1 height: width ratio

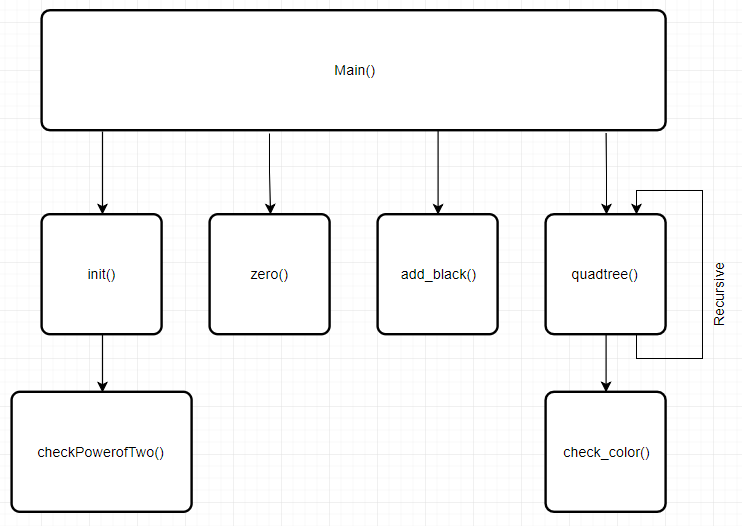
## To run the program:

To run the program, begin by placing the root folder (Assignment) in your c drive, then open the terminal on your device and navigate to the folder by typing **cd Assignment**. The program will then need compiling, this can be done by typing **gcc quadtree.c -o quadtree** in the terminal. You are now ready to run the program; this program requires it to be run with a text input file along with it, so you can run program by entering **quadtree example.txt** where example.txt is the path of your input file. If your input file is valid the program will output the black node positions as well as their sizes to the terminal, if you want to save the output to another text file use **quadtree example.txt > output.txt** and the output will be saved to output.txt.

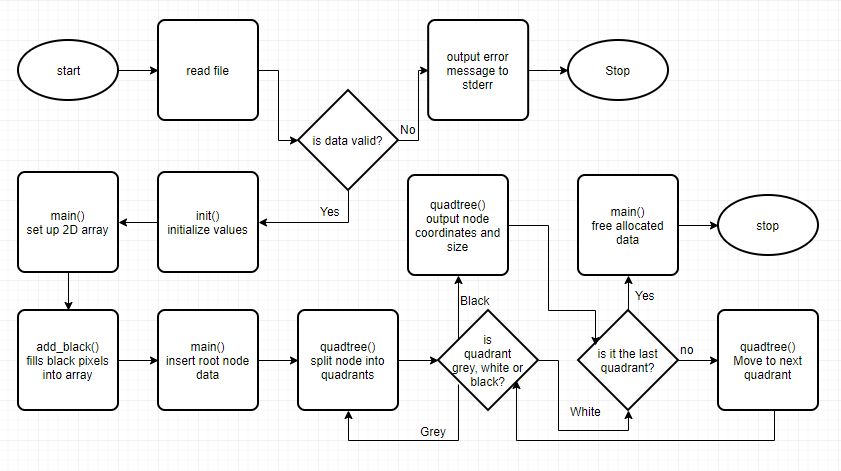
# Functions

|  |  |
| --- | --- |
| int **main**(int argc, char const \*argv[]) | The main() function is responsible for controlling the overall flow of the data among most of the functions, it is also responsible for ending the program on multiple cases where data validations return false from other functions or within main() itself.  main() is where the 2D array image is set up using the malloc() to dynamically allocate it memory according to the size taken from the input file.  At the end of a run the dynamically allocated memory is freed so to save memory. |
| int **init**(int \*width, int \*pix, FILE \*input\_file, const char \*filename) | First checks if the file the user entered as the input file actually exists or else outputs error message to stderr and ends the program. Next checks if the image size and number of black pixels are valid inputs from the file and if they are both in the specified range of operation for the program. If all input validations are successful, the values are passed back to the main function. |
| int **zero**(int width, int pix, FILE \*input\_file) | When the size read from the file is 1, this function is called instead of quadtree() as it has to be approached differently.  The function works out the colour of the pixel as well as validates the extra coordinates given. |
| int **checkPowerofTwo**(int x) | Takes in input (x) and checks if it is a power of 2 by continuously halving x and using modulo division to check if the new value is divisible by 2 until x reaches 1. Used to validate the width input from the file. |
| int **add\_black**(int \*\* image\_array, int black, FILE \*input\_file) | Uses a for loop to read and temporarily store the x and y coordinates of each black pixel from the input file. These coordinates are then added to the 2d image array and passed back to the main function. |
| int **check\_color**(int \*\* image\_array, int top,  int left, int size) | quadtree() will pass on a quadrant of the image to be checked to check\_color() which will iterate through the quadrant and tally up the colours that encounters in each pixel. At the end it determines whether the quadrant is white, black or grey depending on the amount of each colour and passes this back to quadtree(). |
| void **quadtree**(int top, int left, int size, int \*\* s) | This is a recursive function that has 2 main purposes. Firstly, to create new child nodes by encountering grey quadrants, after creating each node it will recursively call itself on them until it encounters a white of black quadrant where it can perform its second purpose which is to output the location each black node as well as its size. |

# Data hierarchy



**Figure 1: Functional Hierarchy Diagram**



**Figure 2: Data Flow** **Diagram**

# Error checking/Tests

|  |  |  |
| --- | --- | --- |
| **image** | **output** | **Expected output** |
|  | Black terminal node at position (2,2) with size 2  Black terminal node at position (4,1) with size 1  Black terminal node at position (5,1) with size 1  Black terminal node at position (4,2) with size 2  Black terminal node at position (4,4) with size 2  Black terminal node at position (6,4) with size 1 | Black terminal node at position (2,2) with size 2  Black terminal node at position (4,1) with size 1  Black terminal node at position (5,1) with size 1  Black terminal node at position (4,2) with size 2  Black terminal node at position (4,4) with size 2  Black terminal node at position (6,4) with size 1 |
|  | Black terminal node at position (3,7) with size 1  Black terminal node at position (4,4) with size 1  Black terminal node at position (5,5) with size 1  Black terminal node at position (4,6) with size 1  Black terminal node at position (4,7) with size 1  Black terminal node at position (5,6) with size 1  Black terminal node at position (6,6) with size 2  Black terminal node at position (2,8) with size 1  Black terminal node at position (2,9) with size 1  Black terminal node at position (3,8) with size 1  Black terminal node at position (2,10) with size 1  Black terminal node at position (4,8) with size 2  Black terminal node at position (4,10) with size 1  Black terminal node at position (5,11) with size 1  Black terminal node at position (6,8) with size 2  Black terminal node at position (6,11) with size 1  Black terminal node at position (9,5) with size 1  Black terminal node at position (8,6) with size 1  Black terminal node at position (8,7) with size 1  Black terminal node at position (9,6) with size 1  Black terminal node at position (10,4) with size 1  Black terminal node at position (10,6) with size 1  Black terminal node at position (10,7) with size 1  Black terminal node at position (11,7) with size 1  Black terminal node at position (8,8) with size 2  Black terminal node at position (8,11) with size 1  Black terminal node at position (9,11) with size 1  Black terminal node at position (10,8) with size 1  Black terminal node at position (10,9) with size 1  Black terminal node at position (11,8) with size 1  Black terminal node at position (10,10) with size 1  Black terminal node at position (12,8) with size 1  Black terminal node at position (12,9) with size 1  Black terminal node at position (12,10) with size 1 | Black terminal node at position (3,7) with size 1  Black terminal node at position (4,4) with size 1  Black terminal node at position (5,5) with size 1  Black terminal node at position (4,6) with size 1  Black terminal node at position (4,7) with size 1  Black terminal node at position (5,6) with size 1  Black terminal node at position (6,6) with size 2  Black terminal node at position (2,8) with size 1  Black terminal node at position (2,9) with size 1  Black terminal node at position (3,8) with size 1  Black terminal node at position (2,10) with size 1  Black terminal node at position (4,8) with size 2  Black terminal node at position (4,10) with size 1  Black terminal node at position (5,11) with size 1  Black terminal node at position (6,8) with size 2  Black terminal node at position (6,11) with size 1  Black terminal node at position (9,5) with size 1  Black terminal node at position (8,6) with size 1  Black terminal node at position (8,7) with size 1  Black terminal node at position (9,6) with size 1  Black terminal node at position (10,4) with size 1  Black terminal node at position (10,6) with size 1  Black terminal node at position (10,7) with size 1  Black terminal node at position (11,7) with size 1  Black terminal node at position (8,8) with size 2  Black terminal node at position (8,11) with size 1  Black terminal node at position (9,11) with size 1  Black terminal node at position (10,8) with size 1  Black terminal node at position (10,9) with size 1  Black terminal node at position (11,8) with size 1  Black terminal node at position (10,10) with size 1  Black terminal node at position (12,8) with size 1  Black terminal node at position (12,9) with size 1  Black terminal node at position (12,10) with size 1 |
|  | Black terminal node at position (0,0) with size 1 | Black terminal node at position (0,0) with size 1 |
|  |  |  |
|  | width not power of 2 /\* stderr \*/ | width not power of 2 /\* stderr \*/ |
|  | Black terminal node at position (0,0) with size 2 | Black terminal node at position (0,0) with size 2 |