QUADTREE ENCODING OF A BINARY IMAGE

COMPUTER ALGORITHMS AND ARCHITECTURE ASSIGNMENT 1

DECEMBER 9, 2019

6478252

TABLE OF CONTENTS

1.0 PROGRAM OVERVIEW

The goal of the project was to design a program in C language that uses a quadtree to encode a binary image of a silhouette. A quadtree encoding saves space when encoding a binary image as groups of pixels can be stored as one and can be subdivided until an individual colour is reached.

This implementation can be summarised into the following steps: -

1. Divide the current two-dimensional space i.e. array into four boxes.
2. If a box contains more than one colour, create a child node, storing in it the two-dimensional space of the box.
3. If a box contains a single colour, do not create a child for it.
4. Recurse for each of the child nodes.

The program takes in a txt file that contains the size, number of black pixels and their coordinates in the grid. The program can was compiled by **gcc** but any c compiler can be used. It is called using two arguments i.e. **“./MUTUMBAJ-quadtree input.txt**” where the first argument is the compiled program and the second argument is the input file.

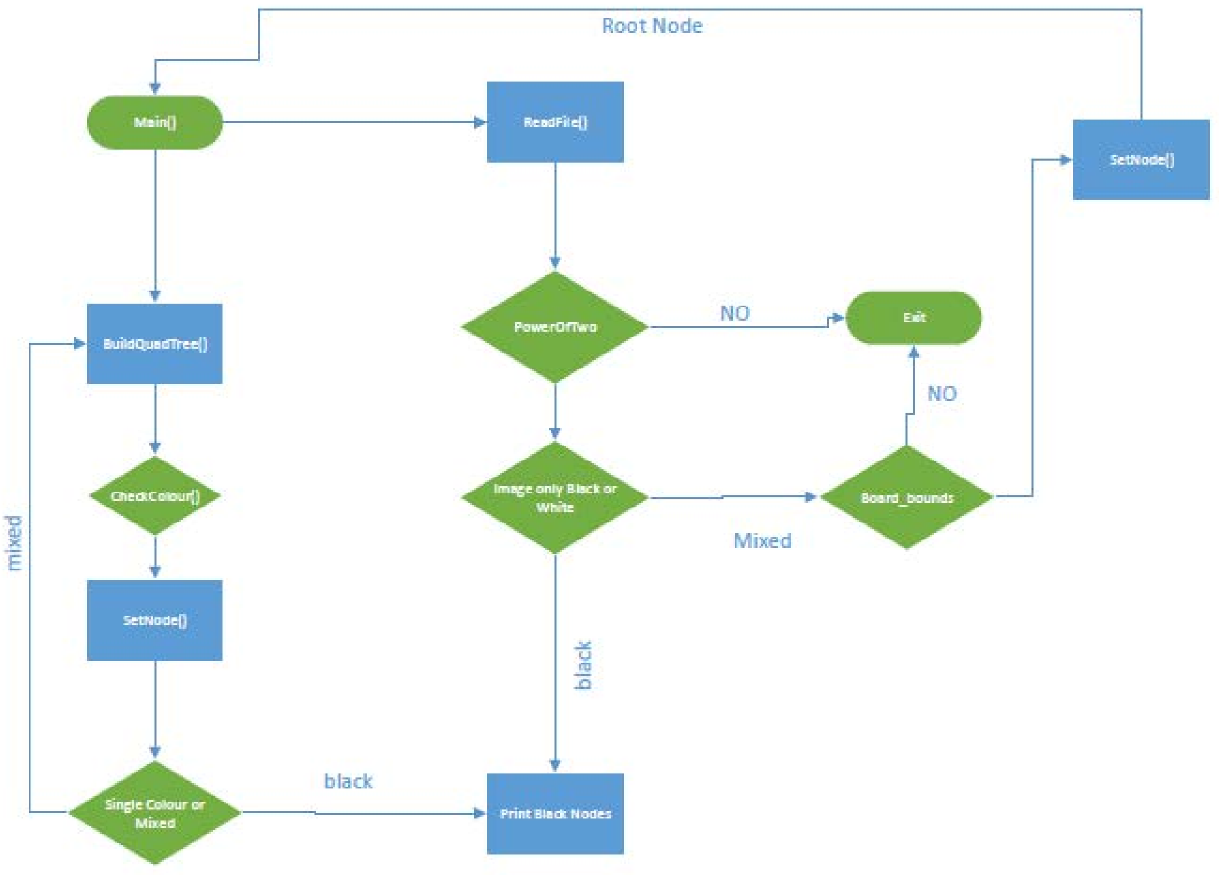
1.1 FUNCTIONS

The program execution can be broken down into the following functions in order of how they are called.

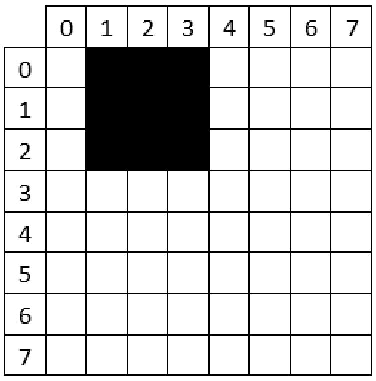
1. **main()**
   This is the main entry point of the program. It takes in the two arguments and the input file name. It performs a check to make sure the input file has been specified by checking if argc is 2.
   A 2D array is created in the form of a pointer to a pointer and a structure of the same type as the globally declared ‘node’ called root is created.
   This function calls **Readfile()** with the arguments of the input file and a pointer to the root node.
   It also calls **BuildQuadTree** function with arguments of the pointer to Root and the retrieved array Arr.
2. **ReadFile()**
   This is the function that reads the input file specified at command line. It takes in the filename specified by argv[1] and it takes a pointer to a node.
   It starts by opening the file, runs checks to see if the file can be opened.
   It then scans the input, and places it into a temp array, each time dynamically reallocating memory for a temporary array allowing it to grow. Several checks on the input are then made to ensure its correct. A 2d array is then dynamically allocated which is initialised with all white and then using data from the temp array, the black pixels are placed in their correct coordinates.
   The memory occupied by the temporary array is then freed back to the stack.
   The node is then set with the colour, Xpos, Ypos and size information by calling the SetNode function. The file is then closed and the 2D array returned to main.
3. **PowerofTwo()**
   This function checks if the passed width is a power of two. It is a boolian function that is called in the ReadFile() to check the passed width. It uses the fact that all powers of two have only one bit set. It returns either True or False. Standard bool header is included in the code to allow for this code to work.
4. **Board_bounds()**
   This is the function that checks If the passed coordinates are on the bounds of the binary image. It is valled in ReadFile to check black pixel coordinates from the file before they are placed into the 2D array.
5. **SetNode()**
   This function sets up the root node and its children. It is passed the node, coordinate information, the width and the colour. It assigns this information to the node and then allocates memory for its 4 children nodes. This function is first called in ReadFile to set up the root node and then recursively called in BuildQuadTree to create more children nodes on a child node. This is the function that sets up the information held in the Quad tree.
6. **CheckColour ()**

This is the function that checks the colour of a specified region the 2D array.
It is called in the BuildQuadTree function and passed the 2D array, top, bottom, left and right which specify the region of the array to be checked. It counts the number of black and white pixels and returns Black if there are no white pixels and vice versa. If there are both black and white pixels it returns mixed.

7. **BuildQuadTree()**
This is the function that builds the quadtree. It is called in main() and passed the 2D array and the address of the root node retrieved from the ReadFile() function.
It uses a switch case with a for loop to go through all the cases i.e. NW, NE, SW, SE respectively.
Each case is characterised by setting up top, bottom, left, right and size nodes that point to a specific node in binary image where the size is now half that of the root node.
Colour is then checked, and the value returned is used to SetNode on the child nodes. If the colour returned is black, then the node is printed. However, if the colour returned is mixed then the function recurses and further subdivides the array into the 4 children and then goes through the cases until uniform colour is reached.



2.0 TESTING

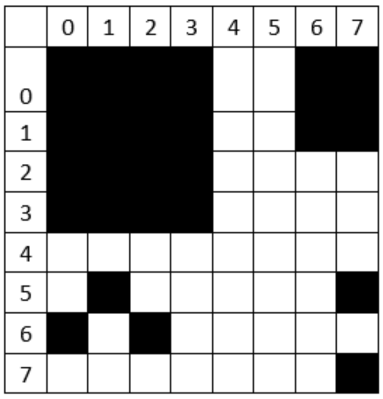


Input

8
9
1 0
2 0
3 0
1 1
2 1
3 1
1 2
2 2
3 3

Output

Black terminal node at Position (1,0) with width 1
Black terminal node at Position (1,1) with width 1
Black terminal node at Position (2,0) with width 2
Black terminal node at Position (1,2) with width 1
Black terminal node at Position (2,2) with width 1
Black terminal node at Position (3,3) with width 1

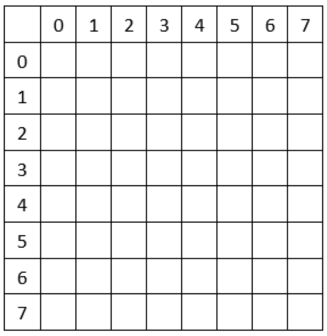


Input

8
25
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
3 0
3 1
3 2
3 3
6 0
6 1
7 0
7 1
1 5
0 6
2 6
7 5
7 7

Output

Black terminal node at Position (0,0) with width 4
Black terminal node at Position (6,0) with width 2
Black terminal node at Position (1,5) with width 1
Black terminal node at Position (0,6) with width 1
Black terminal node at Position (2,6) with width 1
Black terminal node at Position (7,5) with width 1
Black terminal node at Position (7,7) with width 1

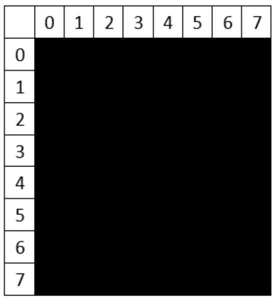


Input

8
0

Output

No black pixels in this input file

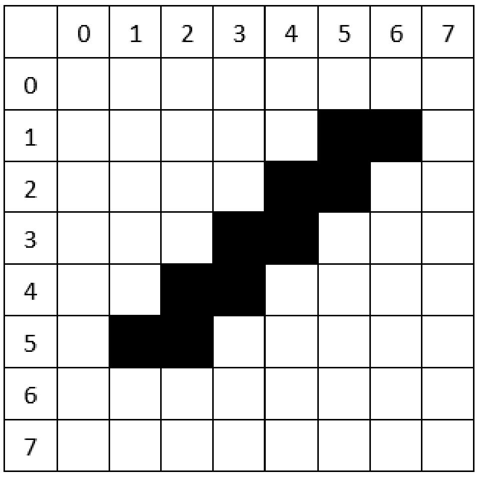


Input

8
64
All coordinates

Output

Black terminal node at position (0,0) with width 8



Input

8
10
1 5
2 4
2 5
3 3
3 4
4 2
4 3
5 1
5 2
6 1

Output

Black terminal node at Position (3,3) with width 1
Black terminal node at Position (5,1) with width 1
Black terminal node at Position (6,1) with width 1
Black terminal node at Position (4,2) with width 1
Black terminal node at Position (5,2) with width 1
Black terminal node at Position (4,3) with width 1
Black terminal node at Position (1,5) with width 1
Black terminal node at Position (2,4) with width 1
Black terminal node at Position (3,4) with width 1
Black terminal node at Position (2,5) with width 1

Input	Output

7
9
1 0
2 0
3 0
1 1
2 1
3 1
1 2
2 2
3 3
	Specified width is not a power of 2

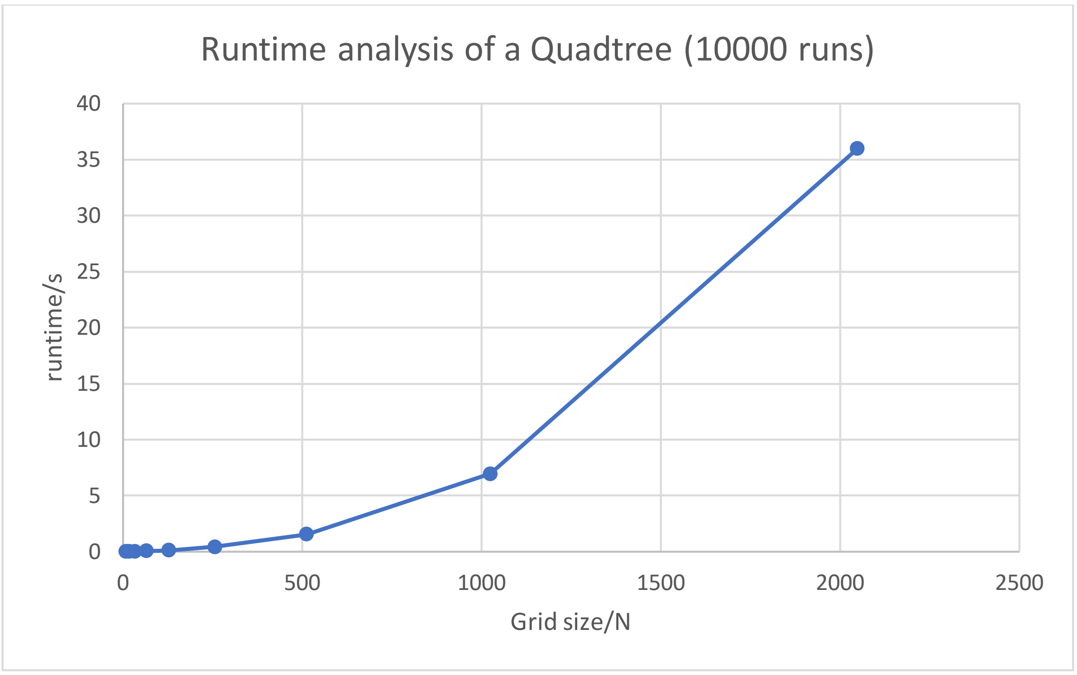
Input	Output

8
9
1 11
2 0
3 0
1 1
2 1
3 1
1 2
2 2
3 3
	pixel 1 off image bounds, please provide correct input

3.0 RUNTIME ANALYSIS

By adding a clock to the code that creates and traverses the quadtree over 10000 runs and changing the grid size, the following results were obtained.

size/N	8	16	32	64	128	256	512	1024	2048
runtime	0.024	0.02	0.027	0.068	0.115	0.438	1.558	6.954	35.96



The obtained graph is logarithmic as expected with the run time increasing exponentially with higher grid sizes.

The BuildQuadTree function divides the input size into 4 regions each time a recursive call is made making it have a time complexity of O(log(N)) in the average case.

CONCLUSION

The program performs according to the specification reading any input file with the correct format and printing out the node at the top left corner and the size where the black pixel(s) lie. Functions and their variables are made easy to understand and implement within the program. Further changes to the program can easily be made by changing the input arguments to the functions.