01/05/2017

QUADTREE ENCODING OF A BINARY IMAGE

Computers and Programming II Assignment

Eva Esteban

Abstract

This report describes the proposed C language programme, which implements a quadtree to represent a binary image array. This is an image processing technique often used in fields such as pattern recognition, medical imaging, robot vision and encoding. This document provides an outline of the programme's methodology, implementation and obtained results for a set of different inputs.

1. Introduction

During the years, different methods for storing image processing data have been developed. One of the data structures that can be used for this purpose is a quadtree, which is a tree structure in which every node has four children nodes. It can be implemented to represent an image whose size is an integer power of two. In this project a quadtree structure is used for storing a square two-dimensional binary image with a maximum width of 64 pixels.

2. Programme description

Methodology and implementation

The programme is implemented in the C language. It consists of a source code file, which includes a set of functions alongside the main function. A set of comments are incorporated to guide the user through the working of the code. The programme implements recursive algorithms, which are based on solving problems by solving smaller instances of the problems and combining their solutions.

The objective of the program is to implement a quadtree of nodes and iterate through it, printing the position and size of the black nodes, given a binary image. Data for the latter is provided in the input file to the programme. Each node has a pointer to it, as well as a colour i.e. white, black or grey, a size, x-y position coordinates, a pointer to its parent node and four pointers to its four children nodes. The grey nodes, which are the ones that contain both white and black pixels, will have four children nodes, as opposed to the black and white nodes, which will not have any children nodes.

The programme is compiled by executing the command **cc -c quadtree.c**, followed by **cc quadtree.o -o quadtree -lm** in the terminal. It is run by typing the command **./quadtree input.txt**, where **input.txt** is the input file. A total number of four functions are included in the programme:

- **printBlack:** Recursive function which outputs to screen the position and size of the black nodes in the final version of the quadtree. It does so by iterating through the quadtree and checking the colour of each node. It takes a pointer to a node (n) as a parameter, and does not return anything (void).

- **checkQuadrant:** Recursive function which checks a specific quadrant of the array and converts it into a node, setting its fields to the appropriate values. It will divide the quadrant into four other quadrants with assigned pointers only if the quadrant contains both white and black pixels. It takes a pointer to a node (n), an integer which is the width of the quadrant the node represents, the array and an integer that refers to the size of the quadrant as parameters. It does not return anything (void).

- **freeQuadtree:** Recursive function called at the end of the programme to free the memory allocated to the quadtree. It iterates through the quadtree, deallocating the memory starting from the nodes at bottom of the quadtree all the way to the top. It takes a pointer to a node (n) as a parameter, returning a pointer to a node, which will equal NULL is the memory has been successfully deallocated. This NULL value will then be assigned to the quadtree root node in the main function.

- **main:** Main function of the programme, where the previously described functions are called. It reads data from the input file after checking that it is not empty and creates an array with its elements set to either 0 i.e. black colour or 1 i.e. white colour, as specified in the input file. It uses dynamic memory allocation to create a pointer to the first node, which is then passed as a parameter to the function **checkQuadrant**. The array is divided into quadrants depending on the colour of the pixels and different pointers to child nodes are created using dynamic memory allocation. Once the quadtree has been implemented, **printBlack** function is called and the position and size of all the resulting black nodes in the quadtree is printed to screen. Following this, a call to **freeQuadtree** deallocates the memory assigned to the quadtree, the file is closed and the programme terminates.

Error checking

A set of tests are implemented in the main function to ensure that the input file contains correct data. The program will exit if an error occurs. An error message will be displayed on screen if:

- The input file is empty
- The size of the input array exceeds the maximum permitted (64 x 64)
- The input number of black pixels exceeds the number of elements in the array
- Pointer memory is not allocated correctly
- The colour of an element is different from white, black or grey

Calling hierarchy

The following diagram shows the calling hierarchy of the programme, which describes the order in which the different functions are called recursively and the decision conditions.

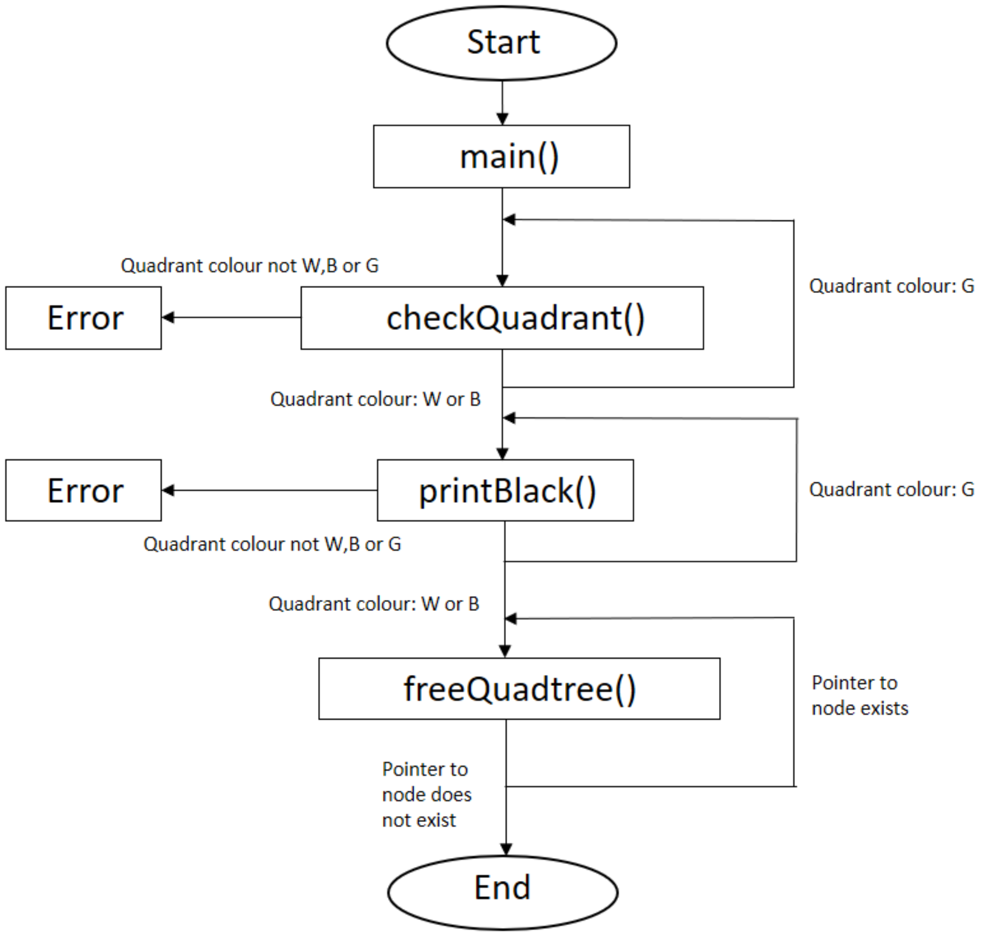
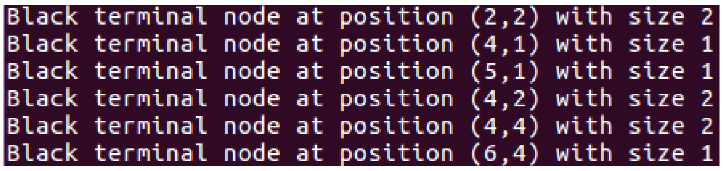


Figure 1: Calling hierarchy of programme

3. Testing and results

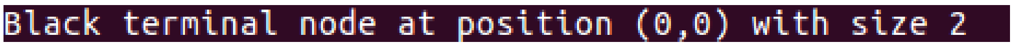
The programme has been tested for the provided input file, as well as a for a set of different files. These evaluate both singular cases such as having an array full of white or black pixels, as well as more generic cases with input arrays of different sizes. The obtained results were compared with the expected results, which were calculated and checked by hand. The comparisons showed no difference between the expected output and the actual output of the programme. Below is some of the input files' data with their corresponding results.

Input file provided on surreylearn:



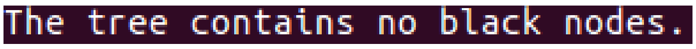
```
Black terminal node at position (2,2) with size 2
Black terminal node at position (4,1) with size 1
Black terminal node at position (5,1) with size 1
Black terminal node at position (4,2) with size 2
Black terminal node at position (4,4) with size 2
Black terminal node at position (6,4) with size 1
```

Completely black array of size 4x4:



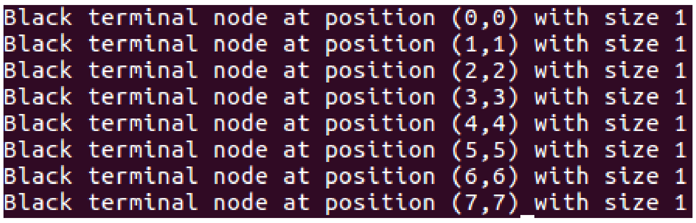
```
Black terminal node at position (0,0) with size 2
```

Completely white array of size 64x64:



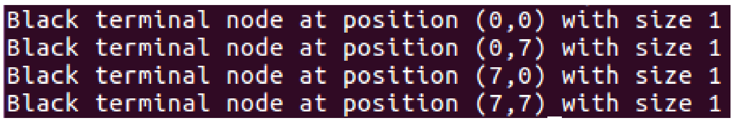
```
The tree contains no black nodes.
```

Array of size 8x8 with a straight line of black pixels from position (0,0) to (7,7)



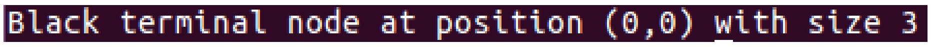
```
Black terminal node at position (0,0) with size 1
Black terminal node at position (1,1) with size 1
Black terminal node at position (2,2) with size 1
Black terminal node at position (3,3) with size 1
Black terminal node at position (4,4) with size 1
Black terminal node at position (5,5) with size 1
Black terminal node at position (6,6) with size 1
Black terminal node at position (7,7) with size 1
```

Array of size 4x4 with one individual black pixel on each corner:



```
Black terminal node at position (0,0) with size 1
Black terminal node at position (0,7) with size 1
Black terminal node at position (7,0) with size 1
Black terminal node at position (7,7) with size 1
```

Completely black array of size 8x8:



```
Black terminal node at position (0,0) with size 3
```

4. Conclusion

Implementing this project has helped me develop my programming and debugging skills, whilst learning about recursive algorithms, dynamic memory allocation, memory trees, pointers and structures. I have gained knowledge about the "typedef" instruction, which I have realised makes the code easier to understand. This assignment has also helped me to understand how memory trees are created, used for different purposes and then destroyed in a recursive way, which saves time and avoids complication.

5. References

- Jones, Bradley L. Aitken, Peter G (2003) 'Sams teach yourself C in 21 days'.
- Guillemaut, Dr Jean-Yves (2017) 'Computers & Programming II'. University of Surrey.
- http://gamedevelopment.tutsplus.com
- http://codingunit.com
- http://en.wikipedia.org/wiki/Quadtree