Quad-trees

This assignment is to implement a tree structure to represent images, which is particularly useful for some kinds of images. An image is considered to be a $2^n \times 2^n$ matrix, each of whose entries is 0 (white) or 1 (black). Each entry is called a pixel. Instead of storing the whole matrix, different data structures are used to reduce the size. The quad-tree is one such.

A quad-tree is a rooted tree in which every node has either 0 or 4 children. Every node in the tree corresponds to some submatrix of the whole matrix, with size $2^i \times 2^i$ for some $0 \leq i \leq n$. The root node is the entire matrix. If all entries in a submatrix corresponding to a node are equal, the node is a leaf node, and it stores the value of the pixels in the submatrix. If there are distinct entries in the submatrix, divide it into 4 submatrices of size $2^{i-1} \times 2^{i-1}$, called top-left, top-right, bottom-right, bottom-left, and recursively construct the subtrees corresponding to the 4 submatrices.

The size of the tree can be much smaller than the size of the matrix when there are large regions in the image with the same colour, which happens in many cases. This is true when rectangular windows of large size are used to set the colours of the pixels.

You have to implement a class called quad_tree to represent such images. Only a header file that contains the class definition should be submitted. The operations to be performed are specified below.

1. `quad_tree(int` n`)`: a constructor that initializes to a matrix of size $2^n \times 2^n$ with all pixels 0. The value of n will be at most 20, and is called the height of the quad_tree.

2. `~quad_tree()` : the destructor that destroys the tree.

3. `quad_tree(quad_tree const &Q)` : copy constructor.

4. `void set(int x1, int y1, int x2, int y2, int b)` : set all pixels in the submatrix with rows x1 to x2 and columns y1 to y2 (inclusive) to the value b. It can be assumed that $0 \leq$ x1 $\leq$ x2 $< 2^n$ and $0 \leq$ y1 $\leq$ y2 $< 2^n$, if the tree has height n.

5. `int get(int x1, int y1) const` : return the value of the pixel (x1,y1).

6. `int size() const` : return the height n.

7. `void overlap(quad_tree const &Q)`: the new value of the matrix is the pixel-wise boolean OR with the matrix corresponding to `Q`. It can be assumed that `Q` has the same height as the image to which the operation is applied.

8. `void intersect(quad_tree &Q)` : same as the previous except that the boolean AND of the two images is computed.

9. `void complement()` : complement all the entries in the matrix.

10. `void resize(int m)`: Change the size of the matrix to $2^m \times 2^m$. This is done as follows. If $m \geq n$, replace each pixel of the original image by a $2^{m-n} \times 2^{m-n}$ matrix with all values equal to the original pixel. If $m < n$, divide the matrix into $2^m \times 2^m$ submatrices, each of size $2^{n-m} \times 2^{n-m}$, and replace each submatrix by a pixel whose value occurs more often in the submatrix. If equal, choose 1.

11. `void extract(int x1, int y1, int m)` : the new value is the $2^m \times 2^m$ submatrix with rows from x1 to x1+$2^m - 1$ and columns $y1$ to $y1 + 2^m - 1$. It can be assumed that the submatrix is well-defined.

Submit a single file named RollNo_3.h . Make sure that you include any header files that are needed, and do not write any main function. Write a separate main program and check that it works correctly using, g++ -include headerfile.h mainprogram.cpp .

This is a standard data structure and you can find many references for it. A good starting point is the Wikipedia article on quad-trees. You may even find some code for some of these operations, but if it does not give all, it will be easier to write your own. If you do use any code from anywhere, please write a comment in the beginning of your file, indicating the source.

The basic operations - constructors, set and get will have 5 marks. 5 marks for the boolean operations and 5 each for the other two. Note that all operations modify the image to which they are applied, and do not return a new image.