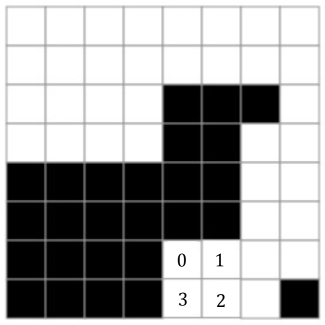
Vintage Instagram: QuadTrees

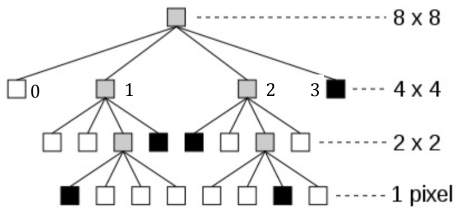
created by Karen Wang and Lisa Wang
thanks to Jerry Cain for borrowed materials

Introduction
Social media websites, geographical information systems, medical image archiving systems, astronomy databases, and many other services must be able to store massive amounts of image data, and thus require image compression techniques to store and retrieve pictures efficiently.

On a computer, images are commonly represented as two-dimensional arrays. Each pixel is an element of the array. However, in an image, we can have large groups of pixels that are the same color. Image compression algorithms find redundancy in an image and remove them to decrease the file size. One technique used in image compression utilizes the "quadtree" data structure.

What are quadtrees?
A quadtree is a type of tree data structure where each internal node has precisely four children. Every node in the tree represents a square, and if a node has children, each encodes one of that square's four quadrants.

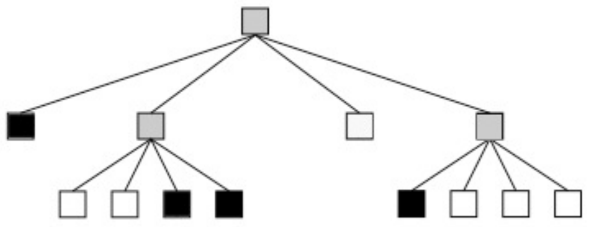




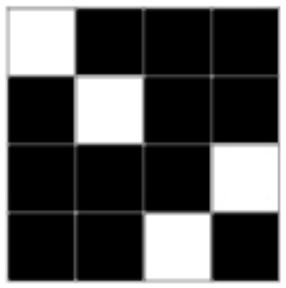
Source: wikipedia.org

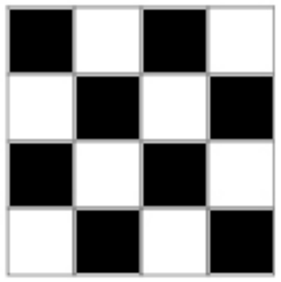
The 8 by 8 pixel image on the left is modeled by the quadtree on the right. Note that all leaf nodes are either black or white, and all internal nodes are shaded gray. The internal nodes are gray to reflect the fact that they contain both black and white pixels. When the pixels covered by a particular node are all the same color, the color is stored in the form of a Boolean and all four children are set to NULL. Otherwise, the node's sub-region is recursively subdivided into four sub-quadrants, each represented by one of four children.

Appetizer Activity:
Convert the following quadtree into its corresponding 2D array of pixels:



Convert the following 2D arrays of pixels into their corresponding quadtrees:
With which image is it more efficient (i.e. takes less time to convert) to use a quadtree, and with which is it less efficient than a 2D array?





Today's challenge: Create a vintage Instagram!

You've just been hired by the hot new startup called Vintage Instagram, a photo-sharing site for old-timer movie stars and other black and white images. The number of users is growing exponentially, so the startup hopes that you can help them compress the loads of images that the users upload every day.

Fortunately, your coworkers know a little bit about quadtrees and defined a few data structures to get you started. However, they did not implement anything yet, so it is up to you to design the algorithms.

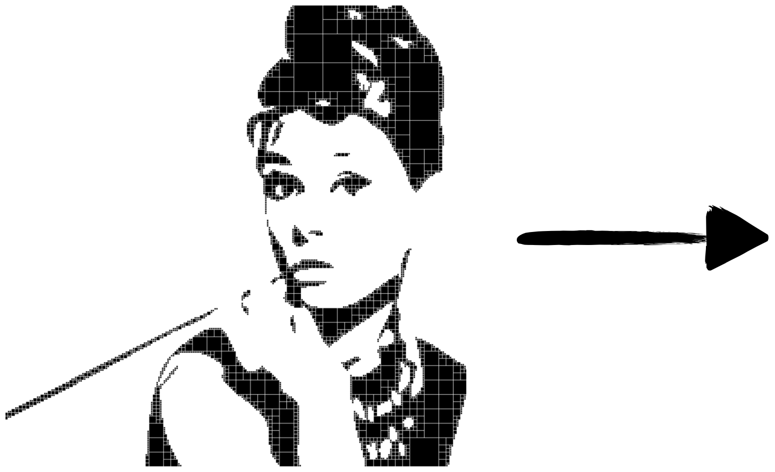
```
struct Quadtree {
    int lowx, highx; // smallest and largest x value covered by node
    int lowy, highy; // smallest and largest y value covered by node
    bool isBlack; // entirely black? true. Entirely white? False. Mixed? ignored
    Quadtree *children[4]; // 0 is NW, 1 is NE, 2 is SE, 3 is SW
};

enum Corners {NW, NE, SE, SW}; // to make indexing more intuitive
```

Problem 1:
Write a function that converts a square grid of booleans to a quadtree representing the same image. You can assume that the side length of the square is a power of two. The function should return a pointer to the root of the tree.

```
        Quadtree *convertGridToQuadtree(Grid<bool>& image)
```

Problem 2:
Design an algorithm that compresses an image represented as a quadtree. For example, the picture on the left has width 512, and after compression, it has target width 256.



You can assume that the side length of image represented by the quadtree as well as the targetWidth are powers of two. The function returns 0 on success, -1 on failure.

```
int compressImage(Quadtree * qt, int targetWidth)
```

Thought questions:
How would you change the color of a single pixel in the quadtree? What is the Big-O of that?
What is the Big-O of converting a 2D array of pixels into a quadtree?
Which part of convertToQuadtree seems to be doing redundant work that could be optimized?
What are some uses where quad trees might be useful? Where might it not be useful?

Craving for more?
Quadtrees are used in many other applications:
- Collision detection in video games and physical simulations
- Storing geographical data (geospatial indexing)
- Blog post "Damn Cool Algorithms: Spatial indexing with Quadtrees and Hilbert Curves" by Nick Johnson: **http://blog.notdot.net/2009/11/Damn-Cool-Algorithms-Spatial-indexing-with-Quadtrees-and-Hilbert-Curves**
- Take CS161 in the future to learn more about algorithms and how to speed them up.
- Take CS166 to explore the coolest data structures ever!