Introduction.  High frequency signal noise is often a problem in real-world applications. For example, this noise manifests itself as "static" on phone lines or a fuzzy television reception, or “snow” in video images. While improving transmission line quality is always an important goal, there is a good bit of filtering that can be done on the digital side in order to improve the final quality. Suppose you are given a noisy black (intensity value of 0) and white (intensity value of 255) image and asked to filter it. To keep the algorithm somewhat simple, the image will contain large regions of black and white with small specs (several pixels of the opposite color) in it. These specs are to be removed from the image. To accomplish this, we will traverse through the image and eliminate any regions which are smaller than a given size (specified at run-time). This will, in effect, filter out the noise in the image and leave the original large signal elements. Small regions are filtered by converting them from white to black or vice-versa.

Filtering Algorithm.  The algorithm works by finding contiguous sets of pixels that are the same color by visiting all neighbors recursively. If the total number of contiguous pixels is less than a predetermined threshold (say for example 10 pixels), it is noise and this region should be removed. Otherwise, it is a valid region and is left alone.

1. Define a boolean array Visited with one entry for each pixel in the image, initially false for each entry. Since we know the exact number of pixels at the start of the algorithm, we can use a vector for this, using the constructor that initializes a specified number of elements to a specified value:

   std::vector<bool> visited(nPixels, false); // nPixels is number of pixels

2. Find the next pixel that has not been visited. Obviously, this will be the zero’th pixel at (0,0) on the first iteration.

3. Mark the pixel as visited, note the color of this pixel, and push this on a queue of pending pixels. Use the STL Double Ended Queue (deque) for this, since a deque can have elements pushed on the back and removed from the front.

   std::deque<int> pending;

4. Create a vector of pixel numbers in this region, initially empty.

   std::vector<int> region;

5. While the pending queue is not empty:

   (a) Obtain the pixel number from the front of the pending deque, and remove it.

   (b) Add this pixel to the current region vector.

   (c) For each of the four neighbors of this pixel (left, right, up, down, NOT diagonally), if the color of the neighbor pixel matches the color of this region and it has not been visited, mark the neighbor as visited and append it to the back of the pending deque. If it has been visited, ignore it. Be sure to be careful here when taking the left neighbor of a pixel on the left edge of the image, and pixels at the other boundaries. You may want to color it gray (pixel value 0x80) for debugging, but this is not necessary for the assignment.

6. At this point the region starting at the pixel found in step 2 is in the region vector. If this size of this vector is less than the threshold (10 pixels), it is noise and each pixel in the region vector should be toggled from black to white, or white to black. Otherwise, leave it alone.

7. When all pixels have been visited, the image is complete.