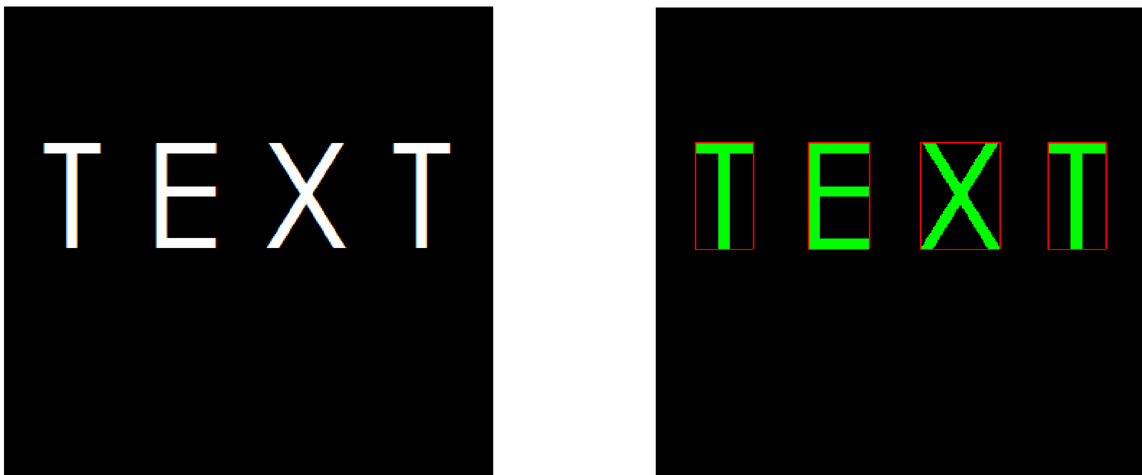
CS 315 Data Structures **Spring 2008**

Project # 3 – Final version

Due: April 1, 2008

Problem Statement: Optical character recognition (OCR) is a useful technology that has significantly advanced the process of digitizing documents. It is also supported by scanners and other devices. Using *image processing* techniques, a OCR reads a scanned document and recognize the individual characters in it. A first step in optical character recognition is to identify the boundaries of each character found in the image. A bounding box is the smallest enclosing rectangle that contains a particular character.



The goal of this project is to take as input an image like in the left-hand side and produce as output the image on the right-side in which the figure on the left has been processed to recognize where the characters are and a bounding box is drawn around each character. You can assume that the background image is (nearly) black and the letters are (nearly) white. A sample input/output is shown above in which the left-side image is the input and the right-side is the output produced by the program.

Goals of the project:

- learn to use the Queue data structure to perform breadth-first search (BFS) and identify the connected components in a graph.
- learn to combine a BFS search implementation with the *EasyBMP* library to identify the number of distinct characters in an image. (Each white pixel in the image will be treated as a node and an edge exists between two nodes if they are adjacent to each other. Thus each connected component will represent a letter. However, letters like **i** will have two components.)

Data Structure and Algorithm used:

The basic idea behind the solution is as follows: When the search process encounters a white pixel, it tries to identify all the pixels that are connected to this pixel. This search is similar to finding the way through a maze, a problem you are already familiar with (from CS 215). You can use a queue to solve this problem as follows: start scanning the image row by row and each row from left to right until you encounter a white pixel at (j,k) (j = row number, k = column number).

```
Let the image width be w and the height be h;
Set visited[j,k] = false for all j,k;
count = 0; Initialize Q;    // count is the number of symbols in
the image
for j from 1 to w - 1 do
 for k from 0 to h - 1 do
   if (color[j,k] is white) then
     count++;
     lowx = j; lowy = k; highx = j; highy = k;
     insert( [j,k]) into Q; visited[j,k] = true;
    while (Q is not empty) do
     p = delete(Q);
     let p = [x,y]; color[x,y] = green;
     if x < lowx then lowx = x;
     if y < lowy then lowy = y;
     if x > highx then highx = x;
     if y > highy then highy = y;
     for each neighbor n of p do
       if visited[n] is not true then
           visited[n] = true;
           if (color[n] is white)
              insert(n) into Q;
           end if;
       end if;
     end for;
   end while;
   draw a box with (lowx-1,lowy-1) as NW and
         (highx + 1, highy + 1) as SE boundary;
  // make sure that highx+1 or highy+1 don't exceed the dimensions
  else visited[j,k] = true;
 end if;
 end do;
end do;
```

Input: The input will be an image with nearly black background and non-black pixels forming letters or foreground images. You can assume that the background pixels are such that each of their R, G and B components < 20. The foreground images are therefore those pixels of which at least one color component is >= 20. Typical input is one in which all the pixels in the foreground are close to white.

Output: Your program should print the number of distinct letters (or foreground images) found. It should create an output image file whose name is user specified (see the format below). In the output file, the background pixels of the input file should be copied exactly and the color of the letters should be changed to green, and there should be a red bounding box around each letter.

Example: If the executable file produced by your C++ program is **run**, then with **in.bmp** as input image that is on the left column in page 1, if we execute the statement

% **run in.bmp out.bmp**

the output produced should be:

4

%

where **out.bmp** will be the image on the right column in page 1.

Grading:

- organization of code and annotation: 10 points
- counting the number of letters: 10 points
- drawing bounding rectangles: 70 points
- color change of foreground pixels: 10 points