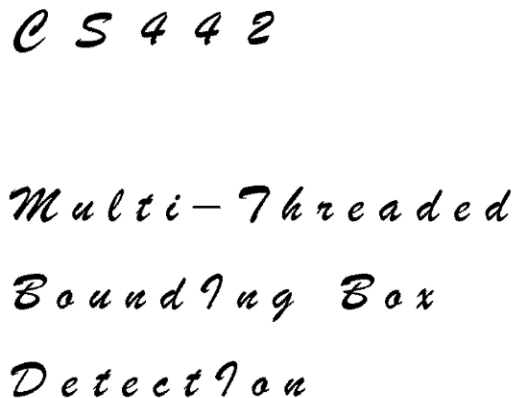


CS4420/5420  
Fall 2016/2017  
**Programming Assignment 2:**  
Multi-threaded-Image Processing  
Due Friday, December 2, 2016

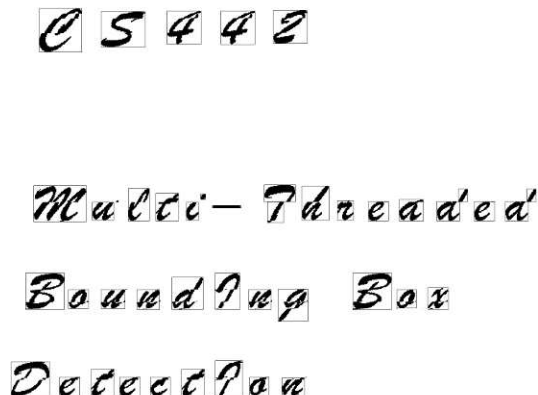
## 1 Introduction

A basic task in optical character recognition (OCR) or other image processing techniques is to identify regions within an image that might contain pixels of interest. In the case of optical character recognition, usually, a first pass is to create a *bounding box* around any possible character. For example, consider the following image that contains some text.



*CS442*  
*Multi-Threaded*  
*Bounding Box*  
*Detection*

As a first pass, you'd like to identify the 12 regions of this image that contain letters, and place bounding boxes around them, as seen the image on the following page.



*CS442*  
*Multi-Threaded*  
*Bounding Box*  
*Detection*

Notice that each bounding box is a square defined by two points  $(x_{min}, y_{min})$  and  $(x_{max}, y_{max})$ , where  $x_{min}$ ,  $y_{min}$  and  $x_{max}$ ,  $y_{max}$  are the minimum and maximum x, y coordinates of any black pixel in the region.

Your task for this assignment is to generate the second image from the first, i.e., find all of the regions of interest in the image, and draw a bounding box around the region. (Your program will also output the number of such regions to the standard error stream for testing purposes.) The images that we will process will be black and white, where each bit in the image corresponds to either on (black) or off (white).

## 2 Images

For the purposes of this assignment, you will be using PBM images. These images are stored as follows. (For more details, examine the web pages that describe the file format for PBM images: [A PBM Example](#) or the [PBM documentation site](#).) The first line contains the string constant “P4” at the beginning of the line. The second line contains two numbers separated by whitespace (  $x$   $y$  ) that give the size of the image. The numbers are written in ASC. For example, for a 500 (rows) by 100 (column) image, the second line would be “100 500”. The remaining characters in the file are organized in  $y$  rows of length  $\lceil x/8 \rceil$  bytes, where each byte in the row contains 8 bits of the image. The image bits in each byte are stored from most significant to least significant bits. So, if the first byte value is in a row is 129, the first pixel in the row is black (1), the next 6 pixels are white (0), and last pixel from the byte is black (1).

## 3 Identifying Regions in the Images

In order to identify regions in the images, we will use the following specifications. Two black pixels  $(x,y)$  and  $(x_1,y_1)$  in the image are defined to be in the same region if  $|x_1 - x| \leq 3$  and  $|y_1 - y| \leq 3$ , i.e., if two pixels are separated by at most 2 white pixels, then they are in the same region. For example, consider the following image block.

```

1  0  0  1  0  1  1
1  0  0  1  0  1  1
1  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  0  0  1  1
0  0  0  0  0  1  1

```

This image block contain two regions that we show in the next image block.

```

1  0  0  1  0  1  1
1  0  0  1  0  1  1
1  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  0  0  0  0
0  0  0  0  0  2  2
0  0  0  0  0  2  2

```

Notice that, in the images that we will be dealing with, this separation will be barely visible. Furthermore, we will ignore any small regions. Here, we will define a small region to be region whose bounding box contains 100 or fewer pixels.

## 4 Images and Implicit Graphs

Finding regions in an image can be modeled as an implicit problem on graphs. Sometimes, a computational problem can be modeled as a problem on graphs where the vertices and edges in the graph are not given explicitly. In this case, we can reduce the problem of finding the regions in the image to a problem on graph-based problem.

As a starting point to finding a suitable strategy for solving this problem, consider the following questions:

1. If we model the problem of finding a region in an image as a graph based problem, what aspect of the image would be considered analogous to a vertex in a graph?
2. If we model the problem of finding a region in an image as a graph based problem, what are the edges in the graph? (Hint: It depends on how the regions are defined. See section 3.)
3. To find a region in the image, what graph-based problem to we need to solve?
4. What graph-based algorithm would you use to find all of the sufficiently large regions ( $> 100$  pixels) in the image and their bounding boxes? How long will your algorithm take to execute?

## 5 Multi-threaded Approach

After developing a single-threaded algorithm, you will devise a multi-threaded strategy by partitioning the image among a specified number of  $2^x$  threads, where  $x = 0, 1, 2, \dots$ . For example, if  $x = 4$ , the image should be partitioned as described in the following figure.

T1	T2	T3	T4
T5	T6	T7	T8
T9	T10	T11	T12
T13	T14	T15	T16

In other words, if the original width of the image is  $w$  and the original height is  $h$ , each thread processes a partition of width  $\lceil \frac{w}{x} \rceil$  and height  $\lceil \frac{h}{x} \rceil$ . Note that the rightmost and bottommost partitions may be smaller due to rounding. Also note that for  $x = 0$  we have the sequential approach.

One main problem you will have to solve is the fact that a single character may be split between two (or more) partitions! After processing each partitions individually, you will need to possibly merge multiple bounding boxes into one.

## 6 Project

You will design a C (or C++) project whose executable is called **bounding\_box**. The program **bounding\_box** will take two command line arguments: The first argument will be the name of a PBM file to be analyzed, the second one will be the number  $x$  as described in the previous section. You will use low-level multi-threading based on the POSIX Pthread library. A good introduction to the Pthread library can be found, e.g., at . Your program will read-in the PBM, partition the image among the specified number of threads, create each thread, and have each thread identify the sufficiently large regions. Subsequently, the boundary problem needs to be addressed by possibly merging boundary boxes as described in the previous section. Finally, you draw the bounding boxes around the regions in the image. Your program will then output the new PBM image to a file whose name is created by taking the old filename and adding the suffix **.new**. Your program will also output the number of regions that you discovered to the standard error (**cerr**). For example, if you execute the command:

```
./bounding_box abcdef.pbm 4
```

your program should create the PBM file **abcdef.pbm.new**.

You should test your program for a different number of threads and determine the overall execution time. Specifically, you should test your program on a machine with at least 8 cores (The computers in our lab at Stocker should have 8 (logical) processors), and you should run your program for values  $x = 0, 1, 2, 3, 4$ . From those timing results you can compute the *speed-up* and the *efficiency* of your approach. The speedup is defined as the parallel execution time divided by the sequential execution time. The efficiency is defined as the speed-up divided by the number of cores (or logical processors).

Create a report that summarizes your results. Include plots that show the efficiency and speed-up for  $x = 1, 2, 3, 4$  threads.

## Electronic Submission

Submit your program electronically using `"/home/drews/bin/442submit 2"`. (Make sure to include a Makefile.)