Design Document

Design Process

I began by understanding what was being asked the task was to identify each object in the image with a unique label. Although I believe that an iterative approach might have been sufficient, checking all 4 neighbors, top bottom, left, right in identifying an object. However, the algorithm in the project offers a much more elegant solution. The algorithm only checks 2 neighbors reducing the problem by half and then creates relationship between connected labels. For this project I referred to the project instruction where the algorithm was explained and illustrated diagrammatically. I also referred back to the previous project, because the file processing was very similar. The file had a similar structure, a square matrix separated by delimiter ‘,’. I used an iterative approach in the first pass and a recursive approach for the second pass. The main issues were debugging the program; I ran into many segmentation faults and index out of bounds errors which I had to debug. There was also an issue with my usage of break statement for which I had to employ the help of Professor Helsing to debug.

Data Structure

The data structures we used in this code are two-dimensional vectors. Since the file being processed is a square matrix of ‘1’s and ‘0’s and to accurately depict the arrangement of the data a 2D data structure is necessary. Vectors are dynamically sized, it gives me an advantage over using arrays. I used a string vector to store the data because a string adds additional security and control in data manipulation. I used another two-dimensional vector of strings to store the relationships between the labels; second column (column 1) to store the label I’m switching, and first column (column 0) to store what I’m switching to.

The use of 2D vectors was the most optimal in this case, because they allow accurate representation of data. And the data is easily accessible using vectors much like the arrays I can reach any specific element using coordinates or indices. And the dynamic sizing of the vectors allows for more flexibility of the program. It would allow the program to run with a matrix of any size.

Only array I used was in the last program to determine the biggest object function. I used an array instead of a vector since the information is known. The arrays can be processed faster since they don't dynamically allocate memory; they arrays have although minor but a definite advantage in time complexity. Making it the most efficient choice.

System Functionality

First there is the main function that controls the flow of the program. The main variables passed to functions as arguments. The 2D string vector that stores data from the file is declared in the main function.

I created a readfile function that is used to retrieve the input file. The function take two argument: the name of the file, and an empty 2D vector passed by reference from main. Inside the function I declare a file stream to process file, a string to read the line and a temp string to read each element. When the file is opened, I have a conditional that checks if the input file is readable if not then displays an error message. If the file is readable the function reads the rectangular matrix using a while loop. The code block under while loop breaks the row at delimiter ',' and reads the row into a temp vector called "line_vector". The temp vector is appended to the 2D vector and then cleared to read the next row/line. After reading in the matrix the file is closed.

The first pass is done using an iterative approach, The first pass takes the argument 2d vector picture and is called in main. The “firstpass “function is used to make the first pass and assign unique labels to objects. However, the objects are not all unique; the labels may be connected in multiple coordinates. The algorithm forms relationships between the labels. I used a 2D vector for the relationships. First column (column 0) hold the label I convert to and the second column (column 1) holds the label I’m switching to. First pass considers 3 cases for a ‘1’: case 1 being that top and left are 0; to which we assign a unique label, case 2 being that either top or left are nonzero in this case we assign it the nonzero label, case 3 being that both top and left have nonzero in which case the smallest label value is adopted and a relationship is formed between the labels. I then display the results of the first pass

The second pass is done using a recursion. The “secondPass” takes the 2D vector and the relation vector as arguments and is called in function “firstPass”. The base case for the recursive function is the size of the relationship vector and it recurs till the relationship vector is 0. I always take the values from the top row of the relationship vector. The label being switched from second column and the label being switched to from the first. I erase column the first row from the vector. I then compare these values to other relationship in the relation vector, for any duplicate or transitive relationships and handle each case. Then I iterate through the “picture” vector and switch the labels. If the relationship has relations the recursion continues otherwise I display the results of the second pass and move and call another function to determine the biggest object.

The “biggestObject” function takes in the picture matrix as an argument; passed by reference from the function “seecondPass”. I initialize an array called counter that uses the number of objects as an argument. The function iterates through the first picture and identifies

unique labels which are then appended to a vector called labels. I then use nested loops to loop each iteration of labels and increment count for each label. I then find the biggest count value and output the corresponding label as the biggest label.

Team Work

Uzair Akram translated the algorithm as described in instructions into code. Designed the program implementing an iterative approach for the first pass and recursion on second pass. I also created an algorithm to find the biggest object and wrote it into code. I tested and debugged the code. I wrote the design document.