Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 3 PART1 REPORT

GÖKTUĞ ALİ AKIN
161044018

Course Assistant:ÖZGÜ GÖKSU

1 INTRODUCTION

1.1 Problem Definition

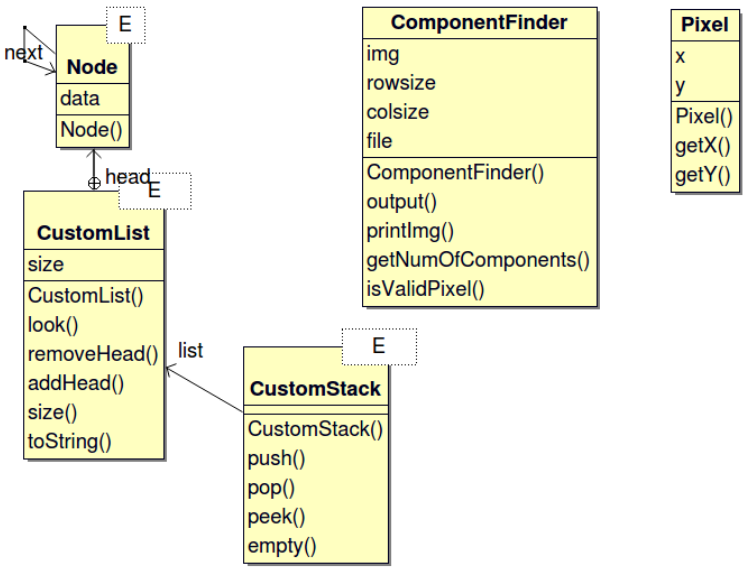
In this homework, problem is finding white components in a binary image. Binar image contains white and black components. White component is a set of matrix locations. These components are connected by 4 directions. Theese directions are up,left,right and down.

1.2 System Requirements

There is no specific properties required in this system. Every operating system which has JVM, can run this program. For big images, process speed will change depending on the hardware that runs the this program.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

User of this program can run with command line. There is no graphical user interface designed for this program. For linux based systems, user can run this program with command line argument which is path of the txt file which contains binary digital image.

2.3 Problem Solution Approach

Input from txt file

Binary digital image contained in txt file in this program. First thing is, reading the binary image from the txt file, and store this binary image in two dimensional integer array. By this approach, we can easly manipulate the binary image with accesing the certain coordinates of components which are black or white.

Representing matrix location

Since binary digital image has matrix locations, I designed a class named '*Pixel*' that represents the 2D point which is a point in the binary digital image. So, we can easly understand and manipluate the point. This class simply has two data members, one of them is the x location, other is the y location.

Custom data structures

CustomStack class implemented using CustomList class. The classes are so simply that, enables the user can use this class with LIFO principles. There is no indexed acces, or any additional method in theese classes except original stack methods. CustomStack class has data member type CustomList to store the datas with Composition principle. I used CustomList class to implement CustomStack easly and abstractly.

Main algorithm of finding components

The main idea of the homework is, finding the white components in the binary image. To do this, firstly, we have to traverse the two dimensional array which we get it from the input txt file. In order to traverse the two dimensoinal integer array, I used nested two for loops, outer for loop provides the traves rows and the inner for loop traverse the columns.While traversing the array, main idea is : 'traverse until catch a white component(1)' . After catching a white component, we have to push this catche location to the stack which contains matrix locations by *Pixel* class. And then, we have to search other white components which are connected to catched component. The algorithm after catched, explained below :

- If a white a component was taken, we have to search connected components.
    1) Check directions (up,down,left,right). If any direction is 1 which means it is white component too, then jump to this location and **push** this new location to the stack. In order to avoid search the location which program alread looked, we

have boolan two dimensional array that provides us to mark locations that we already checked. So, we will not look a location which we looked before.

2) If there is no valid matrix location, so program must come back to previous location. For come back, we have just **pop** of the stack. This logic also named “backtracking”.
   - Valid matrix location : In this algorithm, valid matrix location must have 3 requirements.
     I. Location must be ‘1’ which means it is white component.
     II. Location must be unchecked. Boolean array named checked help to solve this issue.
     III. Location can not have x,y values that are out of bounds of array which contains binary image.

     isValidPixel() method used while validty checking.

Repeating these two statements, will search every component connected to component that we catched first. While searching components, single while loop used inside nested two for loops. This single while loop, iterates while our stack which contains matrix locations is not empty. So, when the stack is empty, searching is finished, checked locations are marked.

After searching is finished, then continue to the next component to be searched with nested for loops.

Time complexity

At the worst case, the algorithm works O(row*column) but while popping back(returning) it becomes O(2N), but O(2N) means **O(N)** time complexity.(Depth-First-Search).

Benefits of using stack data structure

This algorithm is based on “Depth-First-Search” algrotihm. When using this principle, we put our locations that are visited and continue to the next location. If there is no moveable location, it is time to come back. Stacks are so useful for doing this algortihm because of LIFO principle.. Same logic is used recursion style of this algorithm. Return in recursion, equals **pop()** method, calling function recursively equals **push()** in stack based algorithm. We can easily traverse and search the array with this data structure, try all posibilities brute-forcely.

3 RESULT

3.1 Test Cases

I implement simple C driver program that test my java code. This C program, generates random binary digital image with command line argument which represents size of row. I accept square matrix for testing. After generating a random binary digital image, this program runs my java program ComponentFinder and outputs printed.

Example :

Code that generates random image and runs the java program with system command.

```
main(int argc, char const *argv[])

srand(time(NULL));
FILE *fp = fopen("randinput.txt","w");
int size = atoi(argv[1]);
for (int i = 0; i < size; ++i)
{
    for (int j = 0; j < size; ++j)
        fprintf(fp, "%d ",  rand()&1);
    fprintf(fp, "\n");
}
fclose(fp);
system("java hw randinput.txt");
printf("------------\n");
```

Outputs with 2 different runs :

```
gok2@root:~/Desktop/hw3$ ./a.out 10
Input image(randinput.txt) [10]x[10] :
1 1 0 1 1 0 0 1 0 1
1 0 1 1 1 1 0 0 0 1
1 1 0 0 0 1 1 0 0 1
1 1 0 1 0 1 1 0 1 0
1 0 0 0 1 1 0 0 0 0
1 1 1 1 1 1 0 0 0 0
1 1 1 0 0 1 1 1 1 0
1 1 1 0 1 0 1 1 0 1
0 0 0 1 1 1 1 0 0 1
0 1 0 0 1 0 1 1 1 1
Number of components in binary image : 6
```

```
gok2@root:~/Desktop/hw3$ ./a.out 5
Input image(randinput.txt) [5]x[5] :
1 1 0 0 1
0 1 0 0 0
0 1 1 0 0
0 0 1 0 0
0 0 1 0 1
Number of components in binary image : 3
------------
```

3.2 Running Results

```
gok2@root: ~/Desktop/hw3
gok2@root:~/Desktop/hw3$ java hw input.txt
Input image(input.txt) [3]x[3] :
0 1 1
0 0 0
1 1 1
Number of components in binary image : 2
gok2@root:~/Desktop/hw3$
```

```
gok2@root:~/Desktop/hw3$ java hw input2.txt
Input image(input2.txt) [22]x[47] :
0 0 1 1 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 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0
0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 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 0 0 0 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 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 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 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 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 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 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 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 1 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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 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 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 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 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 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 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 0 0 0 0 0 0 0 0 0 0 0 0 0 1 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
Number of components in binary image : 9
gok2@root:~/Desktop/hw3$
```