Lab 2

K M Tawsik Jawad (U01034189)

[Jawad.2@wright.edu](mailto:Jawad.2@wright.edu)

**Wireless Sensor Networks:**

The job of wireless sensor networks is to detect or track objects in an environment. In this particular problem, it is required to process information collected by the wireless network to detect objects of interest in an environment.

**Problem Statement:**

**Part I:**

A sensor network is used in a two-dimensional field from where it is required to find potential objects of interest (POI). The sensor data is characterized by letters in the alphabet or colours. The key here is that only two colours or letters can be used in making a contiguous region. A contiguous region is described by whole traversal of a region by repeated movement from one cell to another. One constraint in the problem is that one colour can only form one contiguous region and another colour can form more than one contiguous region. If this condition is satisfied, then a potential object of interest is detected at that particular place. It is required to find out the maximum number of objects of interest in a particular environment using JAVA.

**Input:**

The input contains a single line that is used to indicate the size of the sensor field. The field would be N\*N square. The next N lines indicate the information gathered by the wireless network. Each line has N colours.

**Output:**

The output would be a single line that denotes the number of potential objects of interest in the input field.

**Solution:**

In order to find the objects of interest, first of all it is needed to find the contiguous regions. To do that, directional array is initialized for horizontal and vertical directions. Then the traversal is done for individual colours to check for the contiguous regions. Checks are provided at each iteration so that the elements are not compared with each other and diagonal traversal is not taken place.

**Algorithm Design:**

1. Create a buffered reader and read the input file to find N.
2. Create 2D arrays of size N\*N to store the colours in the grid.
3. Run the nested loops for each colour in each line to find out the total number of contiguous regions formed by them.
4. Check if one colour creates just one contiguous region and the other one creates more than one contiguous region
5. Keeping the check that number of colours does not exceed 2.
6. Keeping a Boolean array for checking if the colours are visited or not and counter array to count the number of potential objects in a particular region.
7. Find the maximum number of potential objects of interest and write the result in the output text file

**PseudoCode:**

Main()

Create a bufferedReader object to read N from input file  
 input color\_array[N\*N]  
 for each c in color\_array:  
 visited[c] = false  
 for x=N to 0  
 for y=N to 0  
 for i=0 to N-1  
 for j=0 to N-1  
 getObj(N, color\_array, i, j, i+x-1,j+y-1)

if(x!=y)  
 for i=0 to N-y  
 for j=0 to N-x  
 getObj(N, color\_array, i, j, i+y-1, j+x-1)

getObj(N, color\_array, row, col, row1, col1)

horizontalDirArray = {0,-1,1,0}  
 verticalDirArray = {1,0,-1,1}  
 trackRow[N\*N]  
 trackColumn[N\*N]  
 count = 0  
 checkColor[count] = color\_array  
 for i = row to row1:  
 for j = col to col1:  
 checker = 1  
 while(checker>0)  
 contiguous = 0  
 for a = 1 to a<=checker  
 for b=0 to horizontalDirArray.length  
 i1 = trackRow[a] + verticalDirArray[b]  
 j1 = trackColumn[a] + horizontalDirArray[b]  
 visited[color\_array] = true  
  
 contiguous+=1  
 checkColor[count++] = color\_array

if(checkColor[0]==checkColor[1] OR checkColor[1]==checkColor[2])  
 return True

**Time Complexity:**

For calling the getObj function from main, four nested loops are used here the time complexity would be O(n4). For initializing arrays and variables it would be O(1) and for running single loops it is O(n). So total time complexity:

O(n4) + O(n) + O(1) => O(n4)

**Part II:**

Here it is required to find the largest contiguous region with the same status and the largest contiguous region with different status. Here in lieu of objects of interest, focus is more provided on the size of the object of interest. For this part of different status, the largest contiguous region must have at least 2 different statuses.

**Input:**

The first line specifies the size of sensor field. The field is N\*N matrix. The next N lines each have different status in the field. The sensor status value range is specified between 0 to 106and it is specified that in the field there must be at least two different statuses.

**Output:**

Output is two different information on two different lines. The first line is the size of the largest region with the same status. The second line is size of the largest region with different status.

**Solution:**

In this problem the largest region of interest is calculated in a similar way as part I. A visited Boolean array is firstly initialized as false for every element of the status array and later on for each traversal from cell to cell the array index is turned true respectively. Directional array on horizontal and vertical axes are initialized. First of all, largest contiguous region with the same status is calculated outside the nested loops for simplicity and reducing time complexity. Later on, to calculate the same for different status, similar approach was taken as part I where nested loops were required to find out the maximum size of the contiguous region showing different status.

**Algorithm Design:**

1. Create a buffered reader and read the input file to find N.
2. Create 2D arrays of size N\*N to store the status array in the grid.
3. Calculate the largest contiguous region from the initial status array.
4. The status array elements are stored in another array called StoreNums for next calculations.
5. The storedArray elements are compared with the initial array and based upon comparison, different status cells are updated in the array.
6. The updated array is used to call the largestROI function which calculates the largest contiguous region having different statuses.
7. Finally the two sizes of two regions are written in the output file.

**PseudoCode:**

Main()

Create a bufferedReader object to read N from input file  
 input status\_array[N\*N]  
 for each c in status\_array:  
 visited[c] = false  
 counter = 0  
 storeNums[counter++] = status\_array[index][index]  
 sameStatusRegion = largestROI(n, status\_array, n-1, n-1)  
 differentStatusRegion = sameStatusRegion  
 for i = 0 to counter:  
 for j = i+1 to counter:  
 for x = 0 to n-1:  
 for y = 0 to n-1:  
 if(status\_array[x][y] == storeNums[i]):  
 status\_array[x][y] = storeNums[j]  
 size = largestROI(n, status\_array, n-1, n-1)  
 if(size > differentStatusRegion):  
 differentStatusRegion = size

largestROI(N, status\_array, row, col)

horizontalDirArray = {0,-1,1,0}  
 verticalDirArray = {1,0,-1,1}  
 trackRow[N\*N]  
 trackColumn[N\*N]  
 count = 0  
 checkColor[count] = color\_array  
 for i = row to row1:  
 for j = col to col1:  
 checker = 1  
 while(checker>0)  
 contiguous = 0  
 for a = 1 to a<=checker  
 for b=0 to horizontalDirArray.length  
 i1 = trackRow[a] + verticalDirArray[b]  
 j1 = trackColumn[a] + horizontalDirArray[b]  
 visited[color\_array] = true  
  
 contiguous+=1  
 checkColor[count++] = color\_array

return Max(contiguous)

**Time Complexity:**

For calling the largestROI function from main, four nested loops are used here the time complexity would be O(n4). For initializing arrays and variables it would be O(1) and for running single loops it is O(n). So total time complexity:

O(n4) + O(n) + O(1) => O(n4)