Java LAB Midterm (Periodic assessment 2)

Name: Sankalp Mukim

Registration Number: 20BDS0128

Slots: L21+L22

Code: CSE1007

# Question:

Assume two players have filled a 6X6 matrix filled with 1’s and 2’s as follows, and you are given this matrix as input. Spawn four threads to perform the following tasks - Threads 1 and 2 should count the number of 3 consecutive 1’s, taken horizontally. Threads 3 and 4 should count the number of 3 consecutive 1’s, taken vertically. The 4 threads should access the ‘count’ variable in a mutually exclusive way. After all the 4 threads do their complete checking, they must repeat the same procedure for counting consecutive 3 zeros (\*twos) in the same way. Based on the two counts, declare the winner (Player 1 or 2).

Text

Description automatically generated

The first matrix in the following schematic gives the count for ones (totally 7 consecutive ones – 4 counted by threads 1 and 2 in the horizontal direction (shown in red colour) and 3 counted by threads 3 and 4 in the vertical direction (shown in green colour)). The second matrix shows the count for twos (totally 2 consecutive 2s – one counted by threads 1 and 2 and another counted by threads 3 and 4). So, declare player 1 as the winner.

counting 1s

4 1s vertically + 3 1s horizontallycounting 2s

1 2 triplet horizontally + 1 2 triplet vertically

# Code:

The code contains 3 classes (respectively in 3 files).

* Execution
  + Calls all object.start() for all the threads.
  + Calculates the total counts
  + Calculates and displays winning player.
* CountHorizontal
  + Calculates count of triplets in given 2d array in either odd or even Rows
  + Takes 3 parameters in constructor:
    - Main 2d array of int[][]
    - countOdd of Boolean
    - element of integer
* CountVertical
  + Calculates count of triplets in given 2d array in either odd or even Columns
  + Takes 3 parameters in constructor:
    - Main 2d array of int[][]
    - countOdd of Boolean
    - element of integer

## CountHorizontal.java

public class CountHorizontal extends Thread {

public boolean notFinished;

public boolean odd;

public int element;

public int count;

int[][] arr;

public CountHorizontal(int[][] getArr, boolean countOdd, int ele) {

arr = getArr;

odd = countOdd;

element = ele;

notFinished = true;

}

public boolean horizontalTriplet(int atX, int atY) {

return (arr[atX][atY] == element && arr[atX][atY + 1] == element && arr[atX][atY + 2] == element);

}

public void run() {

int start = odd ? 1 : 0;

for (int i = start; i < arr.length; i += 2) {

for (int j = 0; j < arr.length - 2; j++) {

if (horizontalTriplet(i, j)) {

count++;

}

}

}

notFinished = false;

}

}

## CountVertical.java

public class CountVertical extends Thread {

public boolean notFinished;

public boolean odd;

public int element;

public int count;

int[][] arr;

public CountVertical(int[][] getArr, boolean countOdd, int ele) {

arr = getArr;

odd = countOdd;

element = ele;

notFinished = true;

}

public boolean verticalTriplet(int atX, int atY) {

return (arr[atX][atY] == element && arr[atX + 1][atY] == element && arr[atX + 2][atY] == element);

}

public void run() {

int start = odd ? 1 : 0;

for (int i = start; i < arr.length - 2; i += 2) {

for (int j = 0; j < arr.length; j++) {

if (verticalTriplet(i, j)) {

count++;

}

}

}

notFinished = false;

}

}

## Execution.java

public class Execution {

public static void main(String[] args) {

int[][] mainArr = { { 1, 2, 2, 1, 1, 1 }, { 2, 1, 2, 2, 2, 1 }, { 2, 1, 1, 2, 1, 1 }, { 2, 2, 1, 1, 1, 2 },

{ 1, 2, 2, 1, 1, 1 }, { 2, 1, 1, 1, 2, 2 } };

// This code works for any nXn 2D array with no change

// We can calculate for digits other than 1 and 2 with minimal change

// counting 1s

int oneCount = 0;

CountHorizontal oddsHorizontalOnes = new CountHorizontal(mainArr, true, 1); // Thread 1

CountHorizontal evensHorizontalOnes = new CountHorizontal(mainArr, false, 1); // Thread 2

CountVertical oddsVerticalOnes = new CountVertical(mainArr, true, 1); // Thread 3

CountVertical evensVerticalOnes = new CountVertical(mainArr, false, 1); // Thread 4

oddsHorizontalOnes.start();

evensHorizontalOnes.start();

oddsVerticalOnes.start();

evensVerticalOnes.start();

while (oddsHorizontalOnes.notFinished || evensHorizontalOnes.notFinished || oddsVerticalOnes.notFinished

|| evensVerticalOnes.notFinished) {

try {

oddsHorizontalOnes.join();

oddsVerticalOnes.join();

evensHorizontalOnes.join();

evensVerticalOnes.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

oneCount += oddsHorizontalOnes.count;

oneCount += oddsVerticalOnes.count;

oneCount += evensHorizontalOnes.count;

oneCount += evensVerticalOnes.count;

System.out.println("Count of triplets for Team 1:\t" + oneCount);

// counting 2s

int twoCount = 0;

CountHorizontal oddsHorizontalTwos = new CountHorizontal(mainArr, true, 2);

CountHorizontal evensHorizontalTwos = new CountHorizontal(mainArr, false, 2);

CountVertical oddsVerticalTwos = new CountVertical(mainArr, true, 2);

CountVertical evensVerticalTwos = new CountVertical(mainArr, false, 2);

oddsHorizontalTwos.start();

evensHorizontalTwos.start();

oddsVerticalTwos.start();

evensVerticalTwos.start();

while (oddsHorizontalTwos.notFinished || evensHorizontalTwos.notFinished || oddsVerticalTwos.notFinished

|| evensVerticalTwos.notFinished) {

try {

oddsHorizontalTwos.join();

oddsVerticalTwos.join();

evensHorizontalTwos.join();

evensVerticalTwos.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

twoCount += oddsHorizontalTwos.count;

twoCount += oddsVerticalTwos.count;

twoCount += evensHorizontalTwos.count;

twoCount += evensVerticalTwos.count;

System.out.println("Count of triplets for Team 2:\t" + twoCount);

int winningTeam = oneCount > twoCount ? 1 : 2;

System.out.println("Therefore, the winning team is: Team " + winningTeam);

}

}

# Output:

