Program Set #1
Total Points: 30

Three problems must be implemented for full credit. Each section will state how many problems to implement within it. The required (starred) problem marked in the set must be implemented by everyone. See 2436 Grading Guide Sheet for additional grading/submission information. Partial credit will be given. (10 points each)

Section One- COSC 1436/COSC 1437 Review. Choose two problems.

1. Professor C. been living on a remote rectangular island for the past 5 years now, and it has always worried him that there is a volcano nearby. He needs to determine if he needs to move his home to a different island, or if he safe in his current location. A two-dimensional grid filled with grass, trees, water, and a volcano represents the island. A `T` within this grid represents a tree, a `W` represents water, and the `V` represents the volcano. The home is located at an `S`. Assume that in a single unit of time, one can move up, down, left, or right (but not diagonally) within the grid. However, one cannot jump in the water, or climb a tree. At the same time that he moves, the volcano's lava spreads in all directions (up, down, left, and right, but not diagonally) from where it currently is, and lava moves at the same rate as the professor does -- one unit of distance in one unit of time. Lava may pass unhindered through trees but cannot get past water. If the home can be reached by lava, then he must make it to the cave, `C` before the lava gets to him.

Write a program to determine if your professor will be able to reach safety if the volcano erupts one day, or if he will need to move somewhere else. The first line of input from a text file contains an integer T, which represents the number of test cases. For each test case, there is a line containing two integers `N` and `M`, which represent the number of rows and columns in your two -dimensional grid island. The next `N` lines each contains an `M` length string consisting of these characters:

```
T = Tree W = Water S = Home C = Cave V = Volcano . = Grass
```

Output, for each labeled case, `YES-Need to move!` if he needs to move his home to a better location, meaning he can't make it to the cave safely should there be an eruption. Otherwise, if the home keeps him safe in its current location, output `NO- Safe to stay!` Let the user input the file name from the keyboard. For C++ use the `string` class. Refer to the sample output below.

Sample File

```
3
5 5
WWS.T
..T.C
.....
.W...
.VWW.
3 10
WWWCTTT...
WWW.TTT...
```

Sample Run:

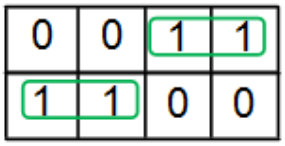
```
Enter file name: volcano.txt

Case 1: NO-Safe to stay!
Case 2: YES-Need to move!
Case 3: NO-Safe at stay!
```

```
S...TTV...
1 6
VWC.TS
```

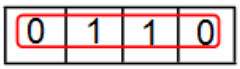
Name the program: `VolcanoXX.java` or `VolcanoXX.cpp`, where XX are your initials.

2. Write a program that prompts the user to enter the size of the rows and columns (between 2 and 5) for a matrix (2D array). The program will randomly fill the numbers 0 and 1 into the matrix, print the matrix, and output the minimum number of rectangles/squares that can cover all the 1 values in the matrix. The size of a rectangle/square is defined by the number of 1's in it. There may be two or more largest rectangles/squares of the same size, and values may "wrap around" or overlap to determine the size. Here are the rules in forming groups of ones:

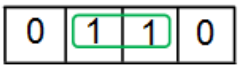
- Consider the consecutive 'ones' in the matrix and group them (green boxes).



- Each group should contain the largest number of 'ones' and no blank cell.



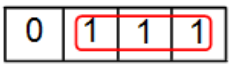
Incorrect



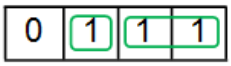
Correct

- The number of 'ones' in a group must be a power of 2 i.e. a group can contain:

$$16(=2^4)\ or\ 8(=2^3)\ or\ 4(=2^2)\ or\ 2(=2^1)\ or\ 1(=2^0)\ cells$$

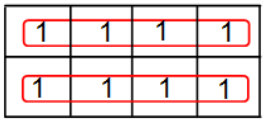


Incorrect

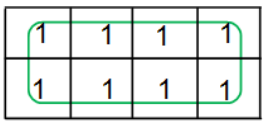


Correct

- Grouping is carried-on in decreasing order meaning, one has to try to group for 8 (octet) first, then for 4 (quad), followed by 2 and lastly for 1 (isolated 'ones').

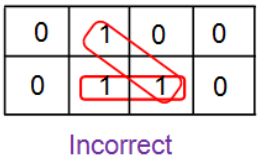


Incorrect

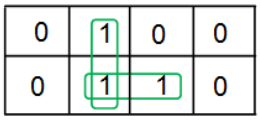


Correct

- Grouping is done either horizontally or vertically or in terms of rectangles/squares. Diagonal grouping of 'ones' is not permitted.

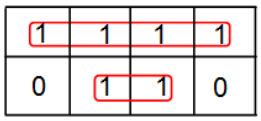


Incorrect

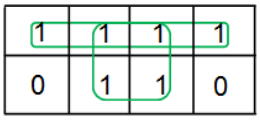


Correct

- The same element(s) may repeat in multiple groups only if this increases the size of the group.

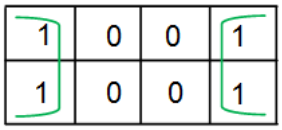


Incorrect



Correct

- The elements around the edges of the matrix are considered to be adjacent and can be grouped together.



Finally, the program should ask if the user wants to run the program again (Check case). Error check input size. Refer to the sample output below.

Sample Run:

```
Enter the number of rows (2-5): 2
Enter the number of cols (2-5): 4

Generated grid:

1 0 0 1
1 0 0 0

The minimum number of rectangles/squares formed is 2

Run Again (Y/N): y

Enter the number of rows (2-5): 2
Enter the number of cols (2-5): 4

Generated grid:

1 0 0 1
1 0 0 1

The minimum number of rectangles/squares formed is 1

Run Again (Y/N): Y

Enter the number of rows (2-5): 4
Enter the number of cols (2-5): 4

Generated grid:
```

```
1 0 1 0
0 1 0 0
0 0 0 0
0 0 0 1

The minimum number of rectangles/squares formed is 4

Run Again (Y/N): y

Enter the number of rows (2-5): 4
Enter the number of cols (2-5): 4

Generated grid:

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

The minimum number of rectangles/squares formed is 2

Run Again (Y/N): Y

Enter the number of rows (2-5): 4
Enter the number of cols (2-5): 4

Generated grid:

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

The minimum number of rectangles/squares formed is 3

Run Again (Y/N): y


Enter the number of rows (2-5): 3
Enter the number of cols (2-5): 5

Generated grid:

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

The minimum number of rectangles/squares formed is 2

Run Again (Y/N): N
```

Name the program: `MinRectanglesXX.java` or `MinRectanglesXX.cpp`, where `XX` are your initials.

3. The United States is not a landlocked country: the country touches at least one ocean (in fact, it touches three). There are 44 countries (including Bolivia and Mongolia, for example) which are landlocked. That is,

they do not touch an ocean, but by going through one other country, an ocean can be reached. For example, a person in Mongolia can get to an ocean by passing through Russia. Liechtenstein and Uzbekistan are the only two countries in the world which are land-landlocked (double landlocked). That is, not only are they landlocked, but all countries which surround these two countries are landlocked countries. Thus, one would have to pass through at least two different countries when leaving Uzbekistan before arriving at an ocean.

Write a program to determine how landlocked each country is on a given map. A country is not landlocked (recorded as `0`) if it touches water in any adjacent cell in either a horizontal, vertical, or diagonal direction. If a country is landlocked, you must calculate the minimum number of borders that one must cross in order to travel from the country to water. Each step of such a journey must be to a cell that is adjacent in either a horizontal, vertical, or diagonal direction. Crossing a border is defined as taking a step from a cell in one country to an adjacent cell in a different country. Note that countries may not be connected to themselves (as in a country formed of islands). In this case, the landlocked value for the country is the minimal of each connected region of the country. Input from a data file, the first line contains `N` and `M` (`1 ≤ N, M ≤ 1000`).

On each of the next `N` lines, there are `M` capital letters. Each country will be represented by a unique letter, with the exception that the letter `W` is reserved to indicate the water in the oceans or seas that will be used to determine the how landlocked each country is. The output consists of a label followed by the country letter followed by a space, followed by the landlockedness for that particular country. Output should be in alphabetical order. Let the user input the file name from the keyboard. Use user-defined functions/methods in your program. For C++ use the `string` class. Refer to the sample output below.

Sample File:

```
7 10
WWWWWCCDEW
WWWWCCEEEW
WTWWWCCCCW
WWFFFFFFWW
WWFAAAAFWW
WWFABCAFFW
WWFAAAAFWW
```

Sample Run:

```
Enter file name: countries.txt

Countries and landlocked values:

A 1
B 2
C 0
D 1
E 0
F 0
T 0
```

Name the program: `LandlockedXX.java` or `LandlockedXX.cpp`, where `XX` are your initials.

4. A rectangular two-dimensional grid shows the style of houses at each location. A neighborhood is defined to be a contiguous set of houses of the same type that exceed some minimum number of houses. The program will use 4 different ways of defining neighbors:

- 4 neighbors normal- Cells have at most 4 neighbors, the cells above, below, left, and right of a cell. Cells on the edge only have 3 neighbors. The 4 corner cells only have 2 neighbors.
- 4 neighbors wrap- All cells have 4 neighbors. The edge and corner cells neighbors wrap around to the other edges.
- 8 neighbors normal- Cells have at most 8 neighbors, the cells above, below, left, right, and the 4 diagonals of a cell. Cells on the edge only have 5 neighbors. The 4 corner cells only have 3 neighbors.
- 8 neighbors wrap- All cells have 8 neighbors. The edge and corner cells neighbors wrap around to the other edges.

Consider this example of neighbors for the 8-neighbor wrap model with cells numbered for clarity.

```
1     2     3     4
5     6     7     8
9     10    11    12
13    14    15    16
```

Consider cell number 1 in the upper left corner. With wrapping its 8 neighboring cells are:

```
16 13 14
4     2
8  5  6
```

Write a program that prints the number of neighborhoods in a given data set given the minimum size of a neighborhood and the 4 ways of defining neighboring cells. A given house can belong to at most one neighborhood per data set.

Input:

- The first line will contain a single integer `N` that indicates the number of data sets.
- The first line in a data set will be a single integer `M` that indicates the minimum number of connected houses (size) required that need to be connected to form a neighborhood. `M >= 1`.
- The first line of a data set will be two integers `R` and `C` separated by a single space. `R` indicates the number of rows in the data set's matrix and `C` indicates the number of columns per row in the data set. `R` and `C` will both be `>= 3`.
- The next `R` lines will be the rows of the matrix in order. Each line will contain `C` upper case letters separated by single spaces. All letters will be between A and Z inclusive. Each letter represents the style of the house at that location in the matrix.

Output:

For each data set print out a labeled line with 4 integers separated by single spaces. The integers are the number of neighborhoods in the data set given the minimum required size based on the 4 rules for neighbors: 4 normal, 4 wrap, 8 normal, and 8 wrap in that order. For C++ use the `string` class. Let the user input the file name from the keyboard. Refer to the sample output below.

Sample File:

```
3
3
4 3
A B C
D E F
G H I
J K L
3
5 7
A B A B A B A
A B A B A B A
A B C B A C A
A B C B A C A
A B A B A B A
5
3 7
A B B C B B A
B C C B C C B
A B B C B A A
```

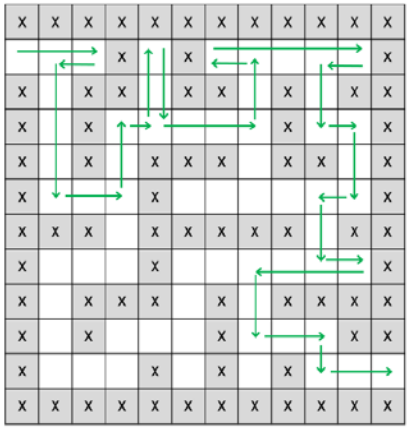
Sample Run:

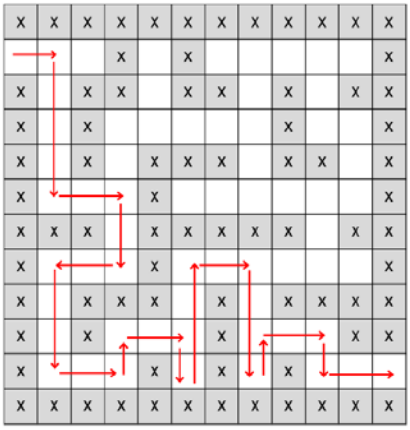
```
Enter file name: neighbor.txt

Set 1: 0 0 0 0
Set 2: 5 6 5 6
Set 3: 0 1 2 3
```

Name the program: `FindNeighborsXX.java` or `FindNeighborsXX.cpp`, where XX are your initials.

Required Problem- Comprehensive.

5 (**). Write a program that finds the exit thorough a maze. There are two rules to guarantee one can find the exit – the "left-hand" and "right-hand" rules. Visualize placing your left or right hand on the nearest wall upon entering the maze. Then move through the maze without that hand ever losing contact with the wall until you reach the exit. You will enter dead ends and then turn towards the right for the left-hand rule and toward the left for the right-hand rule. As one steps forward with an opening to the left for the left-hand rule, you would keep your hand on the corner of the wall and you would turn to the left and continue. A similar move around a right corner for the right-hand rule allows you to stay in contact with the wall. The process continues until you arrive at the exit. One will never get lost and keep retracing previous paths. However, you may frequently return to a spot you previously visited. The mazes are the same. The left one uses left-hand rule and right one uses right-hand rule.





There will be exactly one entrance and one exit for each maze and there is always a solution. One can only move horizontally and vertically, no diagonal moves allowed, and the start and finish positions cannot be corners. The top-left corner is always position `(1,1)`.

Input:

The first line of data file contains the number of test cases, which will not exceed 20. For each test case, the first line contains three items separated by single spaces: the number of rows `R` and columns `C` in the maze, which will not exceed 30, followed by either `'L'` for the left-hand rule or `'R'` for the right-hand rule. The next line contains four integers separated by single spaces: `r1 c1 r2 c2` where `(r1,c1)` is the start position and `(r2,c2)` is the finish position. The start and finish positions will not be adjacent to one another and will not be a corner of the maze. Both `r1` and `r2` will be in the range `1..R` and both `c1` and `c2` will be in the range `1..C`. The next `R` lines will each contain `C` characters separated by single spaces with no extra space at the end of the line. An `'X'` indicates that the position is blocked and cannot be entered while an `'O'` indicates that the position is empty and may be entered.

Output:

For each test case, label the case and rule (`Left` or `Right`) then output a list of positions visited in the form `"(r,c)"` where `r` and `c` are the row and column numbers with no spacing. Format the positions with 10 rows per column right justified downward from start to end. Display a blank line after each test case. Let the user input the file name from the keyboard. For C++ use the `string` class; otherwise, use any data structure of your choice. Do not use recursion. Refer to the sample output below.

Sample File:

```
2
12 12 L
2 1 11 12
X X X X X X X X X X X X
O O O X O X O O O O O X
X O X X O X X O X O X X
X O X O O O O O X O O X
X O X O X X X O X X O X
X O O O X O O O O O O X
X X X O X X X X X O X X
X O O O X O O O O O O X
X O X X X O X O X X X X
```

```
X O X O O O X O O O X X
X O O O X O X O X O O O
X X X X X X X X X X X X
12 12 R
2 1 11 12
X X X X X X X X X X X X
O O O X O X O O O O O X
X O X X O X X O X O X X
X O X O O O O O X O O X
X O X O X X X O X X O X
X O O O X O O O O O O X
X X X O X X X X X O X X
X O O O X O O O O O O X
X O X X X O X O X X X X
X O X O O O X O O O X X
X O O O X O X O X O O O
X X X X X X X X X X X X
```

Sample Run:

```
Enter file name: maze.txt

Case 1: Left

  (2,1)  (5,4)    (3,8)  (4,11)    (9,8)
  (2,2)  (4,4)    (2,8)  (5,11)   (10,8)
  (2,3)  (4,5)    (2,7)  (6,11)   (10,9)
  (2,2)  (3,5)    (2,8)  (6,10)  (10,10)
  (3,2)  (2,5)    (2,9)  (7,10)  (11,10)
  (4,2)  (3,5)   (2,10)  (8,10)  (11,11)
  (5,2)  (4,5)   (2,11)  (8,11)  (11,12)
  (6,2)  (4,6)   (2,10)  (8,10)
  (6,3)  (4,7)   (3,10)   (8,9)
  (6,4)  (4,8)   (4,10)   (8,8)


Case 2: Right

  (2,1)     (8,3)   (11,6)    (10,9)
  (2,2)     (8,2)   (10,6)   (10,10)
  (3,2)     (9,2)    (9,6)   (11,10)
  (4,2)    (10,2)    (8,6)   (11,11)
  (5,2)    (11,2)    (8,7)   (11,12)
  (6,2)    (11,3)    (8,8)
  (6,3)    (11,4)    (9,8)
  (6,4)    (10,4)   (10,8)
  (7,4)    (10,5)   (11,8)
  (8,4)    (10,6)   (10,8)
```

Name the program: `MazeTraversalXX.java` or `MazeTraversalXX.cpp`, where XX are your initials.

Extra Credit: Choose one of the two problems below for extra credit. See 2436 Grading Guide Sheet for additional grading/submission information. Partial credit will be given. No additional extra credit for implementing both programs.

1. A carpet company has an extra unique flair because they pride themselves on conscientious covering of any room using only carpet cut in the shape of squares. Write a C++ program to help them cover a room with the fewest pieces of carpet needed. The carpet squares all measure an integer number of feet on each edge, and only accepts jobs for rooms with both width (`W`) and length (`L`) in integer number of feet. Obviously if a room is perfectly square, they can finish the job with a single piece of carpet. And the maximum number of carpet squares needed is `LxW` squares, each 1-foot on a side. But we can always find a smaller number. Input from the keyboard two integers the width and length of the room up to a maximum size of 25. Output on the first line, `"N squares can cover W x L "` (replacing `N`, `W`, and `L` with their values). On the next line print the sizes of the squares in increasing order, separated by spaces. Finally, output a blank line and then print a map of the room, using a different capital letter (`A, B, …`) for each square of carpet used. The map may not match the sample output, but the minimum number of squares must. For C++ use the `string` class. Refer to the sample output below.

Sample Runs:

```
Enter the carpet width and length: 11 10

6 squares can cover 11x10.
2 2 4 5 5 6

AAAAAABBBBB
AAAAAABBBBB
AAAAAABBBBB
AAAAAABBBBB
AAAAAABBBBB
AAAAAACCCCC
DDDDEECCCCC
DDDDEECCCCC
DDDDFFCCCCC
DDDDFFCCCCC
```

Name the program: `CarpetCoverXX.java` or `CarpetCoverXX.cpp`, where `XX` are your initials.

2. Given a square grid of uppercase letters, write a program to find the largest square formed around a given location called the center. The location of each character in the grid is given by an ordered pair `(r,c)` that indicates the row `r` and the column `c` of a character relative to the origin. The origin is the top left position of the grid and is position `(0,0)`. A largest square is the largest contiguous `s x s` area that contains only the same letter as its center.

Input:

The first line will contain a single integer `n` that indicates the number of data sets to follow.

The first line of each data set will contain a single integer `p` that indicates the number of rows and columns in the square matrix of uppercase letters to follow. The next `p` lines will contain the `p x p` square matrix. The following line will contain a single integer `m` that indicates the number of center locations `(r, c)` to follow. Each of the next `m` lines will contain two integers `r` and `c` that represent the center for which you are to find the largest square.

Output:

For each labeled data set, print `(r c) s` where `r` and `c` are the row and column of the center of the largest square and `s` is the number of letters in an edge of the largest square that contains only the same letter as its center. One blank line should be printed between sets of output. For C++ use the `string` class. Let the user input the file name from the keyboard. Refer to the sample output below.

Sample File:

```
2
9
AAABBBAAA
AAABBBAAA
AAABBBAAA
AAABBBAAA
AABBBBBBB
AABBBBBBB
ABBBBBBBB
BBBBBBBBB
BBBBBBBBB
3
3 2
1 1
6 5
5
BBBBB
BBBBB
BBBBB
BBBBB
BBBBB
2
1 2
2 2
```

Sample Run:

```
Enter file name: squares.txt

Data Set 1:

(3 2) 1
(1 1) 3
(6 5) 5

Data Set 2:

(1 2) 3
(2 2) 5
```

Name the program: `LargestSquareXX.java` or `LargestSquareXX.cpp`, where `XX` are your initials.