**Task 1:**

Write an efficient program to return the duplicate characters from given String, for example, if given String is "C++" then your program should print "+" Similarly, if the input is "Java and JavaScript" then your program should print "J", "a" and "v". User Input is not case sensitive z = Z.

**Extra not-mandatory subtask:**

Also print the count of occurrences next to the duplicate characters:

Input -> Java and JavaScript

Output -> “j” occurs 2 times,   
 “a” occurs 5 times,  
 “v” occurs 2 times

**Task 2:**

Given two non-empty arrays of integers, write a function that determines whatever the second array is a subsequence of the first one.

A subsequence of an array is a set of numbers that aren`t necessarily adjacent in the array, but that are in the same order as they appear in the array.

For example:

the numbers [1,3,4] form a subsequence of the array [1, 2, 3, 4], and so do the numbers [2, 4].

Note that a single number is an array and the array itself are both valid subsequences of the array.

Sample Input:

Array to check -> 5,1,22,25,6,-1,8,10  
Sequence to check if is subsequence of the array -> 1,6,-1,10

Sample output:

True  
  
**Extra not-mandatory subtask:**

Modify the method so that it accepts one modifier that changes the logic (adjacent). If this modifier is given by the user a subsequence is valid only when its numbers are contained in the array, in the same order and are adjacent.

Sample Input:

Array to check -> 5,1,22,25,6,-1,8,10  
Sequence to check if is subsequence of the array -> 1,6,-1,10,adjacent

Sample output:

False

**Task 3:**

Purple vs Brown is a game played on a 2D grid that in theory can be infinite (in our case we will assume that x <= y < 1 000).

Each cell on this grid can be either Purple (represented by 1) or Brown (represented by 0) The game always receives an initial state of the grid which we will call “Generation Zero”. After that, a set of 4 rules are applied across the grid and those rules form the next generation.

Rules that create the next generation:

1. Each Brown cell surrounded by exactly 3 or exactly 6 Purple cells will also become Purple in the next generation.

2. A Brown cell will stay Brown in the next generation if it has either 0, 1, 2, 4, 5, 7 or 8 Purple neighbors.

3. Each Purple cell surrounded by 0, 1, 4, 5, 7 or 8 Purple neighbors will become Brown in the next generation.

4. A Purple cell will stay Purple in the next generation if it has either 2, 3 or 6 Purple neighbors.

Important facts:   
- Each cell can be surrounded by up to 8 cells 4 on the sides and 4 on the corners. Exceptions are the corners and the side of the grid.   
- All the 4 rules apply at the same time for the whole grid for the next generation to be formed.

Your Task:   
Create a program that accepts:   
 - The size of our grid - X, Y (X being the width and Y being the height)   
 - Then the next Y lines should contain strings (each string is long X characters) created by O`s and 1`s which will represent the “Generation Zero” state and help us build the grid   
 - The last arguments to the program should be coordinates (X1 and Y1) and the number N.

(X1 and Y1) will be coordinates of a cell in the grid We would like to calculate in how many generations from Generation Zero until generation N this cell was Purple. (The calculation should include generation Zero and generation N)

Print your result in the console.

Example 1: 3x3 grid, in the initial state, the second row is all 1s. how many times will the cell (1. 0) (top center) become Purple in 10 turns?

3,3

000

111

000

1,0,10

expected result: 5

Example 2: 4x4 grid. Input:

4,4

1001

1111

0100

1010

2,2,15

expected result: 14

Notes:

**Try using the four fundamental OOP principles while solving the tasks, where appropriate.**

**No frameworks needed, use plain JAVA / C# or any other OOP capable language, without external libraries.**

**Use only the console for Input and output.**

**Implement validations of the user input.**

**Format the code to your preferences to improve readability.**

**Put comments in the code to improve readability.**

**Use appropriate and descriptive names when naming variables and functions.**

**Try avoiding code lines longer than 125 characters.**

**Use GitHub or BitBucket for VCS.**

**Start small and iterate to improve your code.**