CS201- Data Structures
Programming Assignment No. 1

Problem 1- Way-Out from a Maze

This problem uses DynamicSafe2DArray as discussed during the lecture. A maze is an object of DynamicSafe2DArray of char type. The content of the array can be any character {**-** (path), ***** (block), **s** (start), **e** (end), **!** (visited)}.

The input maze may contain only one-path and you need to implement a recursive path finding approach that enumerate all cell of the array that are on the path in order of traverse from start s to end e. Here is an example of a maze.

*	s	*	*	*	*	*	*	*	*
*	-	-	*	*	*	*	*	*	*
*	*	-	-	*	*	*	*	*	*
*	*	*	-	*	*	*	*	*	*
*	*	*	-	-	-	-	*	*	*
*	*	*	*	*	*	-	*	-	e
*	*	*	*	*	*	-	*	-	*
*	*	*	*	*	*	-	*	-	*
*	*	*	*	*	*	-	-	-	*
*	*	*	*	*	*	*	*	*	*

All you need to develop a recursive routine for finding path, the search is only allowed to follow {Left, Right, Up and Down} from any location. The output for this problem is complete path from starting to end location. The validation of input cases required as start and end location must be at the boundary of the maze. There may be one or no path for all valid input cases.

Input

The input is from the file, the first line contains two integers n and m, representing the dimension of the maze. The maze is a 2-dimensional array of char. The next n lines contain the rows of the maze; each row contains m columns. Hence there are m integers in each line.

Output

The output file contains coordinates of each point starting from the start to end.

Input file	Output file
4 5 s - - - - * * * * - * * * * - * * * * e	0 0 0 1 0 2 0 3 0 4 1 4 2 4 3 4

Problem 2 – Brute Force Sequence matching

In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. In this problem there are two sequences of variable length given to you, you can keep them in simple DynamicSafeArray. The brute force sequence matching algorithm finds the maximum sequence of match from the two given sequence. Obviously this maximum match would not be larger than the smaller sequence. The sequence matching may disallow characters in either sequence that already appears in the earlier match, but missing from the next possible match. For example, consider the sequence S1 and S2

S1: ACTTGTTACTGTTACCT

S2: ACTGTACTGTACT

Matched: ACT*GT*ACTGT*AC*T

Here the matched sequence is greater than the length of the smallest sequence as we are disallowing match of a character that earlier matched. In the above example there are four such instances hence the match sequence is of length |s2| + 4,

Input

The input file contains a sequence S1 and S2 in the first two lines. Every sequence is between 1 to 80 characters long.

Output

The output file contains the matched sequence between the two given sequences.

Input file	Output file
ACGTCCTTCATT GTCTCATG	GTC*T*CAT*

Problem 3 – KQUERY

You are given an array of numbers, there can be n numbers where n is between ($1 <= n <= 30000$). Each number is between ($1 <= a_i <= 10^9$). You are also given "t" queries of the form q_t (i, j, k) where ($1 <= i <= j <= n$) for each query you need to return the number of elements greater than k in the sub-sequence $a_i, a_{i+1}, ….., a_j$ you need to process all t queries in the same fashion.

Input

The first line of the input file contains the size of the array "n". the next n lines contain n integers for the array. The next line contains the number of the queries that you need to process. Each next line contains three integers representing left and right indexes of arrays and an integer k for which you need to check how many numbers in between indexes are greater than k.

Output

The output contains the result of the queries per line.

Input file	Output file
10 12 7 13 22 34 19 102 77 23 10 5 0 3 7 0 9 30 6 7 100 2 5 30 3 8 20	2 3 1 1 5