HackerRank For Work

California HSPC 2018 180 minutes

Question - 1
Ransomizer

As seen in movies, ransom holders do not send out the note with instructions and details with their own handwriting, fearing that they will be identified. Instead, they cut up magazine headlines and titles and use them to assemble sentences. Your job is to write a program that detects whether you have enough magazine letters to create a ransom notes.

Given an arbitrary ransom note string and another string containing magazine letters, write a function that will return true if the ransom note can be constructed from the magazine letters; otherwise, return false. Each letter in the magazine string can only be used once in your ransom note. Ignore all punctuation.

Input Format:
The first line is the ransom note string.
The second line is the magazine string.

Output Format:
the boolean values true if the ransom string can be constructed with the available letters.

Sample Input 1:

```
pay up pal!
apples are pretty lucky.
```

Sample Output 1:

```
True
```

The desired string is "pay up pal!", and we need 2 a's, 1 l, 3 p's, 1 u, 1 y. From the magazines, we have 2 a's, 2 l's, 3 p's, 1 u, 2 y's. So we can make the ransom string.

Sample Input 2:

```
if you call the cops...
a song of ice and fire: a novel
```

Sample Output 2:

```
False
```

For the ransom string "if you call the cops...", we need 1 a, 2 c's, 1 e, 1 f, 1 h, 1 i, 2 l's, 2 o's, 1 p, 1 s, 1 u, 1 s, 1 y.
But we from the magazine string we dont have any y's. Therefore we cannot construct the ransom note.

Question - 2
Moving Minimum

Given an array of integers, there is a sliding filter of size *s*, gliding from the left to right. At each time, only the *s* numbers in the filter are considered for computation. At each iteration, move the filter 1 step to the right. Find the minimum at each iteration and return all the minimums as an array of integers.

Input Format:

The first line contains an integer, *i*, denoting the number of elements in array *a.*
Each *m* subsequent lines (where *0 ≤ m < j*) is an integer in *a.*
The last line contains the integer *s,* the size of the filter.

Constraints:

The array *a* is nonempty.
The integer *s* is positive.

Output Format:

The function should return an array of integers that is the minimums of the sliding filter

Sample Input 1:

```
5
-1
1
0
2
-3
3
```

Sample Output 1:

```
[-1,0,-3]
```

Explanation 1:

```
Filter position                   Min
------------------------         ---
[-1  1  0] 2  -3                 -1
 -1 [1  0  2] -3                 0
 -1  1 [0  2  -3]                -3
```

Sample Input 2:

```
7
3
4
-5
5
2
-1
-2
3
```

Sample Output 2:

```
[-5,-5,-5,-1,-2]
```

Explanation 2:

```
Filter position                   Min
------------------------         ---
[3  4  -5] 5  2  -1  -2          -5
 3 [4  -5  5] 2  -1  -2          -5
 3  4 [-5  5  2] -1  -2          -5
 3  4  -5 [5  2  -1] -2          -1
 3  4  -5  5 [2  -1  -2]         -2
```

Question - 3
Word Strikeout

Given a string *S* and a string *E*, we want to count how many different ways we can remove characters from S to produce *E***.**
The order of the letters should not be altered.

Input Format:
The first line contains a string, *S*.
The second line contains a string *E*.

Constraints:
S, T are not empty.

Output Format:
The function must return an integer representing the number of ways *S* can be made into *E*

Sample Input 1:

```
happpy
happy
```

Sample Output 1:

```
3
```

Explanation 1:
We can strike out letters in 3 ways to make "happy" from "happpy"

```
1         2         3
happpy    happpy    happpy
^^ ^^^    ^^^ ^^    ^^^^ ^
```

Sample Input 2:

```
boldldd
bold
```

Sample Output 2:

```
5
```

Explanation 2:
We can strike out letters in 5 ways to make "bold" from "boldldd"

```
1         2         3         4         5
boldldd   boldldd   boldldd   boldldd   boldldd
^^^^      ^^^  ^    ^^^   ^   ^^  ^^    ^^  ^ ^
```

Question - 4
Tridiagonal Matrices

A matrix is *Tridiagonal* if only the *diagonal*, *subdiagonal* and *superdiagonal* have non-zero elements. That is the diagonals can have any element, but the off-diagonals can only be 0.

The *diagonal* refers to the elements along the diagonal line that travels from top left to bottom right. The *subdiagonal* refers to the line of elements directly under the diagonal in the same direction, while the *superdiagonal* refers to the line of elements directly above the diagonal in the same direction.

Now given an `n x n` matrix, return `True` if and only if the matrix is *Tridiagonal*.

$$A = \begin{bmatrix} 1 & -2 & 0 & 0 & 0 \\ 1 & -1 & 3 & 0 & 0 \\ 0 & 5 & 0 & -1 & 0 \\ 0 & 0 & -2 & 1 & 4 \\ 0 & 0 & 0 & 1 & 3 \end{bmatrix}$$

For example, A is a tridiagonal matrix.
The diagonal of A is {1, -1, 0, 1, 3}.
The subdiagonal is {1, 5, -2, 1}.
The superdiagonal is {-2, 3, -1, 4}.
The rest of the entries are 0.

Input Format:
The first line contains an integer, *m*, denoting the number of rows in matrix A.
The second line contains an integer, *n*, denoting the number of columns in matrix A.
Each line *i* of the *m* subsequent lines (where $0 \le i < m$) contains *n* integers describing entries of A.

Constraints:
m = n, or the Matrix A is square.

Output Format:
return a boolean value of true for a tridiagonal matrix.

Sample Input 1:

```
4
4
0 2 0 0
2 2 3 0
0 3 3 4
0 0 4 4
```

Sample Output 1:

```
True
```

Explanation 1:
All the entries above the super diagonal and below the subdiagonal are 0.

Sample Input 2:

```
3
3
1 0 1
0 1 0
0 0 1
```

Sample Output 2:

```
False
```

Explanation 2:

The 3,1 entry of the matrix is nonzero, and it is not a member of the diagonal, subdiagonal or superdiagonal.

Question - 5

Island Hopping

Given an array of non-negative integers, you are initially positioned at the first index of the array.
Each element in the array represents your maximum jump length at that position.
Determine if you are able to reach the last index.

For example consider the array [1,2,3,1,0,1]. Index 2 contains the value 3. So if you land on index 2, you are allowed to jump forward 1 space, 2 space, or 3 spaces.
So the solution would be: forward 1, forward 1, forward 3.

Input Format:

The first line contains the number of elements *m* in the array
the m following lines contains integers corresponding to max jump distance from that index.

Constraints:

Every number in the array is positive, and the array is non-empty
You can only go FORWARD

Output Format:

The function must return a boolean value true for arrays that can reach the final index.

Sample Input 1:

```
5
3
1
0
1
3
```

Sample Output 1:

```
True
```

Explanation 1:

The first number 5 indicates how many islands are in the sea.
The 1st island has jump length is 3, so we can reach the 4th island.
From the 4th island, we can reach the 5th island since it has jump length of 1.
Then we have reached the last island.

Sample Input 2:

```
6
3
2
1
0
2
3
```

Sample Output 2:

```
False
```

Explanation 2:
We have 6 islands in this system.
all possible paths only goes to 4th island, but it has 0 jump length, so we are stuck on that island and cannot progress to the last index.

Question - 6
Survival of the Fittest

You are a measly plankton that photosynthesizes in the ocean. Some unknown catastrophic event has reduced the light level. To survive, you will have to alter your genome.

Given a starting genetic sequence (0's and 1's), a survival (ending) genetic sequence, and a list of possible mutations in the gene pool, determine if the gene pool contains the mutations needed to reach the ending gene sequence from the starting gene sequence. You are only allowed to reach another mutation by mutating 1 digit at a time.

For example, given a starting gene of 000, an ending gene of 111, and a gene pool of {011, 110, 101, 001}, it is possible to reach the ending gene from the starting gene by changing only one digit at a time using the following path: 000 -> 00<u>1</u> -> 0<u>1</u>1 -> <u>1</u>11. However if the gene pool was reduced to {011, 110, 101} it is no longer possible to reach the ending gene from the starting gene.

Input Format:
The first line contains a string *s,* the starting genetic sequence.
The second line contains a string *e,* the ending genetic sequence.
The third line contains an integer, *n*, denoting the number of genes in our gene pool represented as an array *a*.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains strings s_i that describe each mutation in our gene pool.

Constraints:
n >= 0
length of *s* is positive

Output Format:
The function must return a boolean value true if the plankton survives (if you can find a path), and false if the plankton dies (if you cannot find a path).

Sample Input 1:

```
0110
1100
3
1110
0100
1010
```

Sample Output 1:

```
True
```

Explanation 1:

We are able to reach the ending gene, 1100, from the start gene 0110, by using one gene in the gene pool: 0110 -> 1110 -> 1100.

Sample Input 2:

```
01100
11000
4
10101
01001
11001
11011
```

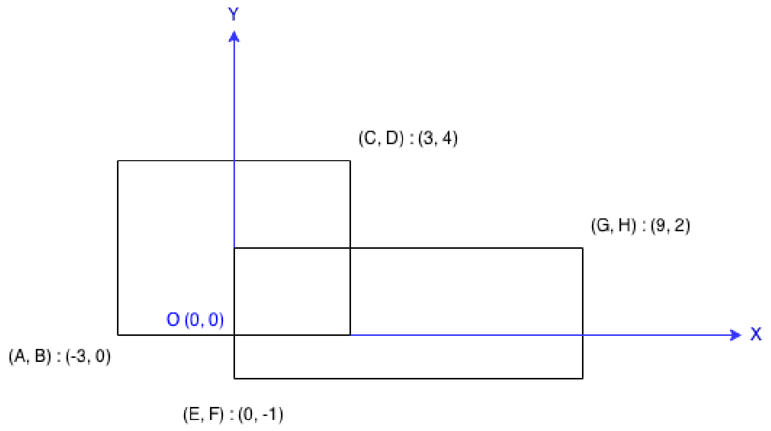
Sample Output 2:

```
False
```

Explanation 2:
There is no set of genes in the gene pool that will allow us to get from 01100 to 11000.

Question - 7
Intersection Over Union

Find the intersection and the union of the two rectangles in a **2D** plane. Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.



The intersection is the overlapping area of the 2 rectangles.
The Union is the total covered area of the 2 rectangles.

Input Format:
The first line contains A
The second contains B
The third contains C
The fourth contains D
The fifth contains E
The sixth contains F
The seventh contains G
And the eighth line contains H

Constraints:
Each area of the rectangle are smaller than the maximum value of **int**

Output Format:
An array containing 2 integers, [*x*,*y*] where *x* is the intersection, *y* is the union.

Sample Input 1:

```
-3
0
3
4
0
-1
9
2
```

Sample Output 1:

```
6
45
```

Explanation 1:

The first rectangle J has lower left coordinate (-3,0) and upper coordinate (3,4).
The second rectangle K has lower left coordinate (0,-1) and upper coordinate (9,2).
Then J has area 24, and K has area 27.
The overlapping area of the rectangles is of area 6, and the total area of the rectangles together is 45.

Sample Input 2:

```
0
0
3
3
2
2
4
4
```

Sample Output 2:

```
1
12
```

Explanation 2:

The first rectangle J has lower left coordinate (0,0) and upper coordinate (3,3).
The second rectangle K has lower left coordinate (2,2) and upper coordinate (4,4).
Then J has area 9, and K has area 4.
The overlapping area of the rectangles is of area 1, and the total area of the rectangles together is 12.

Question - 8

Spaceship Landing

You are an engineer at NASA. Given a 2 Dimensional map on the elevation of the planet XC-0827A, and the size and the shape of a spaceship, you have to find the best landing spot for the space ship!

In order to land the ship, you must find the lowest contiguous group of tiles, that is, the group of tiles (at the same height and touching) that collectively have the lowest elevation and can contain the spaceship.

You are provided with schematics of the ship, in form of an array of 0's and 1's. (0's are empty spaces, and 1's are occupied).

Input Format:

The first line contains an integer, *j* , denoting the number of rows in ship map *s*
The second line contains an integer, *k*, denoting the number of columns in ship map *s*
Each line *i* of the *j* subsequent lines (where $0 \le i < j$) contains *k* integers describing s_{in}. (0 or 1)

The next line contains an integer, *m*, denoting the number of rows in array *a*.
The next line contains an integer, *n*, denoting the number of columns in array *a*.
Each line *i* of the *m* subsequent lines (where $0 \le i < m$) contains *n* non-negative integers describing a_{in}.

Constraints:

m,n > j,k
There are only 1 optimal location to land the spacecraft on all maps.

Output Format:

The function must return a pair of integers in an array [*x,y*] where *x, y* are the coordinate points counting from the top left corner, indexed from 0.
x, y should refer to the top left corner of the spacecraft as well.
x refers to the horizontal direction of the map, *y* refers to the vertical direction of the map.

That is:

```
   012345
  +------>
0 |
1 |
2 V
```

Sample Input 1:

```
2
2
1 1
0 1

4
4
1 1 2 2
1 1 2 2
3 3 0 0
3 3 0 0
```

Sample Output 1:

```
[2,2]
```

Explanation 1:

The 2x2 spacecraft can land on the lowest elevation 0 at tile [2,2]
Even though elevation 1 [0,0], elevation 2 [2,0], elevation 3[0,2] would work, it is not the lowest elevation.

Sample Input 2:

```
3
3
1 1 1
0 0 1
1 1 1
5
5
```

```
1 1 1 1 1
2 0 0 0 1
2 1 1 0 3
2 0 0 0 1
1 1 1 1 1
```

Sample Output 2:

```
[1, 1]
```

Explanation 2:

The spacecraft should land on the elevation 0.
The peculiar spacecraft has a perfect fit the in the continuous elevation of 0 beginning at [1,1]. Then everywhere else, we cannot fit the spacecraft since it cannot find contiguous spaces of the same elevation.