#include <iostream>

#include<string.h>

#define ASCII\_SIXE 256

using namespace std;

char smallest\_alphabet(char a[], int n)

{

// initializing smallest alphabet to 'z'

char min = 'z';

// find smallest alphabet

for (int i=0; i<n-1; i++)

if (a[i] < min)

min = a[i];

// returning smallest alphabet

return min;

}

char getMaxOccuringChar(char\* str)

{

// Create array to keep the count of individual

// characters and initialize the array as 0

int count[ASCII\_SIXE] = {0};

// Construct character count array from the input

// string.

int len = strlen(str);

int max = 0; // Initialize max count

char result; // Initialize result

// Traversing through the string and maintaining

// the count of each character

for (int i = 0; i < len; i++) {

count[str[i]]++;

if (max < count[str[i]]) {

max = count[str[i]];

int leng=strlen(count)

result=smallest\_alphabet(count[str[],leng);

}

}

return result;

}

// Driver program to test the above function

int main()

{

char str[] = "testsample";

cout << getMaxOccuringChar(str);

}

Waiter hackerank

import java.util.Scanner;

import java.util.Stack;

public class Solution {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int N = sc.nextInt();

int Q = sc.nextInt();

Stack<Integer> A = new Stack<Integer>();

for (int i = 0; i < N; i++) {

A.push(sc.nextInt());

}

int prime = 2;

for (int i = 0; i < Q; i++) {

Stack<Integer> B = new Stack<Integer>();

Stack<Integer> nextA = new Stack<Integer>();

while (!A.empty()) {

int number = A.pop();

if (number % prime == 0) {

B.push(number);

} else {

nextA.push(number);

}

}

printStack(B);

A = nextA;

prime = nextPrime(prime);

}

printStack(A);

sc.close();

}

static void printStack(Stack<Integer> s) {

while (!s.empty()) {

System.out.println(s.pop());

}

}

static int nextPrime(int begin) {

for (int i = begin + 1;; i++) {

if (isPrime(i)) {

return i;

}

}

}

static boolean isPrime(int n) {

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) {

return false;

}

}

return true;

}

}

Puxxle crossword

|  |
| --- |
| import java.util.ArrayList; |
|  | import java.util.Arrays; |
|  | import java.util.List; |
|  | import java.util.Scanner; |
|  | import java.util.Set; |
|  | import java.util.stream.Collectors; |
|  | import java.util.stream.IntStream; |
|  |  |
|  | public class Solution { |
|  | static final int SIZE = 10; |
|  | static final int[] R\_OFFSETS = { 0, 1 }; |
|  | static final int[] C\_OFFSETS = { 1, 0 }; |
|  |  |
|  | public static void main(String[] args) { |
|  | Scanner sc = new Scanner(System.in); |
|  |  |
|  | char[][] grid = new char[SIZE][SIZE]; |
|  | for (int r = 0; r < SIZE; r++) { |
|  | String line = sc.next(); |
|  | for (int c = 0; c < SIZE; c++) { |
|  | grid[r][c] = line.charAt(c); |
|  | } |
|  | } |
|  | String[] words = sc.next().split(";"); |
|  |  |
|  | char[][] solution = solve(grid, words); |
|  | IntStream.range(0, SIZE).forEach(r -> System.out.println(String.valueOf(solution[r]))); |
|  |  |
|  | sc.close(); |
|  | } |
|  |  |
|  | static char[][] solve(char[][] grid, String[] words) { |
|  | return search(grid, Arrays.stream(words).collect(Collectors.toSet()), 0, 0, 0); |
|  | } |
|  |  |
|  | static char[][] search(char[][] grid, Set<String> remainWords, int r, int c, int direction) { |
|  | if (r == SIZE) { |
|  | return grid; |
|  | } |
|  | if (c == SIZE) { |
|  | return search(grid, remainWords, r + 1, 0, 0); |
|  | } |
|  | if (direction == R\_OFFSETS.length) { |
|  | return search(grid, remainWords, r, c + 1, 0); |
|  | } |
|  |  |
|  | int insertLength = countInsertLength(grid, r, c, direction); |
|  | if (insertLength > 1) { |
|  | for (String remainWord : new ArrayList<>(remainWords)) { |
|  | if (canInsert(grid, r, c, direction, insertLength, remainWord)) { |
|  | List<Integer> insertOffsets = new ArrayList<Integer>(); |
|  |  |
|  | for (int insertOffset = 0; insertOffset < insertLength; insertOffset++) { |
|  | int insertR = r + R\_OFFSETS[direction] \* insertOffset; |
|  | int insertC = c + C\_OFFSETS[direction] \* insertOffset; |
|  |  |
|  | if (grid[insertR][insertC] == '-') { |
|  | grid[insertR][insertC] = remainWord.charAt(insertOffset); |
|  |  |
|  | insertOffsets.add(insertOffset); |
|  | } |
|  | } |
|  | remainWords.remove(remainWord); |
|  |  |
|  | char[][] subResult = search(grid, remainWords, r, c, direction + 1); |
|  | if (subResult != null) { |
|  | return subResult; |
|  | } |
|  |  |
|  | remainWords.add(remainWord); |
|  | for (int insertOffset : insertOffsets) { |
|  | int insertR = r + R\_OFFSETS[direction] \* insertOffset; |
|  | int insertC = c + C\_OFFSETS[direction] \* insertOffset; |
|  |  |
|  | grid[insertR][insertC] = '-'; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | return null; |
|  | } else { |
|  | return search(grid, remainWords, r, c, direction + 1); |
|  | } |
|  | } |
|  |  |
|  | static int countInsertLength(char[][] grid, int r, int c, int direction) { |
|  | int prevR = r - R\_OFFSETS[direction]; |
|  | int prevC = c - C\_OFFSETS[direction]; |
|  | if (prevR >= 0 && prevR < SIZE && prevC >= 0 && prevC < SIZE && grid[prevR][prevC] != '+') { |
|  | return 0; |
|  | } |
|  |  |
|  | int insertLength = 0; |
|  | while (r >= 0 && r < SIZE && c >= 0 && c < SIZE && grid[r][c] != '+') { |
|  | insertLength++; |
|  |  |
|  | r += R\_OFFSETS[direction]; |
|  | c += C\_OFFSETS[direction]; |
|  | } |
|  | return insertLength; |
|  | } |
|  |  |
|  | static boolean canInsert(char[][] grid, int r, int c, int direction, int insertLength, String word) { |
|  | return word.length() == insertLength && IntStream.range(0, word.length()).allMatch(insertOffset -> { |
|  | int insertR = r + R\_OFFSETS[direction] \* insertOffset; |
|  | int insertC = c + C\_OFFSETS[direction] \* insertOffset; |
|  |  |
|  | return grid[insertR][insertC] == '-' || grid[insertR][insertC] == word.charAt(insertOffset); |
|  | }); |
|  | } |
|  | } |

Bigger then greater

import java.io.\*;  
import java.util.\*;  
import java.text.\*;  
import java.math.\*;  
import java.util.regex.\*;  
  
public class Solution {  
     
    public static void solve (char str[])  
    {  
        int i, flag=0;  
        loop:  
        for(i=str.length-1; i>0; i--)  
        {  
             
            if(str[i]>str[i-1])  
            {  
                int j = str.length-1;  
                while(flag==0 && j!=i-1){  
                    if(str[i-1]<str[j])  
                    {  
                        char t = str[j];  
                        str[j] = str[i-1];  
                        str[i-1] = t;  
                        flag = 1;  
                        break loop;  
                    }  
                    j--;  
                }  
            }  
                 
        }  
         
      Arrays.sort(str,i,str.length);  
         
        if(flag==0)  
            System.out.println("no answer");  
        else  
            System.out.println(str);  
         
    }  
     
    public static void main(String[] args) {  
        char name[];  
        int t;  
        Scanner in = new Scanner(System.in);  
        t = in.nextInt();  
        for(int i=0; i<t; i++)  
        {  
            name = in.next().toCharArray();  
            solve(name);  
        }  
    }  
  
}

Diwali