Task 1 Ans

18 Tháng Chín 2024 5:25 CH

ABString

```
public class ABString {
    public static boolean solution(String s){
        //C1: Duyệt từ phải -> trái (Khi gặp 'a', nếu đã gặp 'b' trước đó
thì trả về false)
        boolean findB = false;
        for(char c : s.toCharArray()){
            if(c == 'b'){
                findB = true;
            }else if(c == 'a' && findB){
                return false;
            }
        }
        return true;
        //C2: Dùng index của "ba" (ktra có tồn tại "ba" không? Nếu có trả
về false)
        //return !s.contains("ba");
        //C3: Dùng biểu thức chính quy (ktra có xuất hiện ký tự 'a' sau
'b' hay không)
        //return !s.matches(".*b.*a.*");
    }
    public static void main(String[] args) {
        System.out.println(solution("aabbb")); // Output: true
        System.out.println(solution("ba"));    // Output: false
        System.out.println(solution("aaa"));   // Output: true
        System.out.println(solution("b"));     // Output: true
        System.out.println(solution("abba"));  // Output: false
    }
}
```

AsphaltPatches

```
public class AsphaltPatches {
    public static int solution(String s){

// C1: Duyệt qua chuỗi và dùng indexOF() để tìm vị trí của 'x'
// Khởi tạo biến patches và i:
// Biến patches dùng để đếm số lần cần vá (số đoạn 'x').
// Biến i dùng để lưu chỉ số hiện tại khi duyệt chuỗi.
// Sử dụng vòng lặp while:
// Vòng lặp chạy cho đến khi duyệt hết chuỗi.
// Tìm vị trí của 'x' bằng indexOf:
// Phương thức indexOf('x', i) tìm vị trí tiếp theo của ký tự 'x' bắt đầu
từ vị trí i.
// Nếu không tìm thấy 'x', phương thức trả về -1, và bạn dừng vòng lặp
(break).
// Cập nhật số lần vá và nhảy qua 3 ký tự:
// Nếu tìm thấy 'x', bạn tăng biến patches (đếm số lần vá).
// Sau đó, bạn tăng biến i lên 3 để bỏ qua 3 ký tự tiếp theo (vì mỗi lần
vá sẽ bao phủ tối đa 3 ký tự).

//        int patches = 0;
//         int i = 0;
//         while(i < s.length()){
//             i = s.indexOf('x', i);
//             if(i == -1){
//                 break;
//             }
//             patches++;
//             i += 3;
//         }
//         return patches;

//C2: Duyệt chuỗi và tăng chỉ số i trực tiếp
//gặp 'x', tăng patches và nhảy qua 3 ký tự tiếp theo.
        int patches = 0;
        int i = 0;
        while (i < s.length()) {
            if (s.charAt(i) == 'x') {
                patches++;
                i += 3;  // Nhảy qua 3 ký tự khi gặp 'x'
            } else {
                i++;  // Nếu không gặp 'x', tăng i lên 1
            }
        }
        return patches;

//C3: Dùng for
    //     int patches = 0;
    //     for (int i = 0; i < s.length(); i++) {
    //         if (s.charAt(i) == 'x') {
    //             patches++;
    //             i += 2;  // Nhảy qua 2 ký tự nữa (vì vòng lặp for sẽ
tự tăng i thêm 1)
    //         }
    //     }
    //     return patches;

}
    public static void main(String[] args) {
        System.out.println(solution(".x..x")); // Output: 2
        System.out.println(solution("x.xxxxx.x.")); // Output: 3
        System.out.println(solution("xx.xxx..")); // Output: 2
        System.out.println(solution("xxxx")); // Output: 2
    }
}
```

CardPayments

```
public class CardPaymemt {
    public static int solution(int[] A, String[] D) {
        int balance = 0;
        int[] cardPaymentsCount = new int[12]; // Mảng để lưu số lượng
giao dịch bằng thẻ trong mỗi tháng
        int[] cardPaymentsSum = new int[12];   // Mảng để lưu tổng số
tiền thanh toán bằng thẻ trong mỗi tháng
```

FormatArray

```
public class FormatArray {
    public static void solution(int[] A, int K){
        int maxLen = 0; // Tìm độ dài lớn nhất của các số trong mảng
        for (int num : A) {
            maxLen = Math.max(maxLen, String.valueOf(num).length());
        }

// Tính số hàng cần thiết để in mảng
        int numRows = (A.length + K - 1) / K;
        for (int row = 0; row < numRows; row++) {
            printTopBorder(K, maxLen, row, A.length);
            printRow(A, K, maxLen, row);
            printBottomBorder(K, maxLen, row, A.length);
        }
    }

 // Hàm in đường viền trên của hàng
    private static void printTopBorder(int K, int maxLen, int row, int totalLength) {
        for (int i = 0; i < K && (row * K + i) < totalLength; i++) {
            System.out.print("+");
            for (int j = 0; j < maxLen + 2; j++) {
                System.out.print("-");
            }
        }
        System.out.println("+");
    }

// Hàm in nội dung của hàng
    private static void printRow(int[] A, int K, int maxLen, int row) {
        for (int i = 0; i < K && (row * K + i) < A.length; i++) {
            System.out.print("|");
            System.out.printf("%" + (maxLen + 2) + "d", A[row * K + i]);
        }
        System.out.println("|");
    }

 // Hàm in đường viền dưới của hàng
    private static void printBottomBorder(int K, int maxLen, int row, int totalLength) {
        for (int i = 0; i < K && (row * K + i) < totalLength; i++) {
            System.out.print("+");
            for (int j = 0; j < maxLen + 2; j++) {
                System.out.print("-");
            }
        }
        System.out.println("+");
    }

    public static void main(String[] args) {
        int[] A = {4, 35, 80, 123, 12345, 44, 8, 5};
        int K = 8;
        solution(A, K);
        System.out.println();
        int[] B = {4, 35, 80, 123, 12345, 44, 8, 5, 24, 3};
        K = 4;
        solution(B, K);
        System.out.println();
        int[] C = {4, 35, 80, 123, 12345, 44, 8, 5, 24, 3, 22, 35};
        K = 4;
        solution(C, K);
    }
}
```

MonitorsDelivery

```
import java.util.Arrays;

public class MonitorsDelivery{
    public static int solution(int[] d, int[] c, int p){
        int fullFilled = 0;
        // Tạo mảng chứa cặp (distance, monitors) từ D và C
        int[][] order = new int[d.length][2];
        for(int i = 0; i < d.length; i++){
            order[i][0] = d[i]; //Khoảng cách
            order[i][1] = c[i]; //Số lượng monitors yêu cầu
        }
        //// Sắp xếp đơn hàng theo khoảng cách tăng dần
        Arrays.sort(order, (a, b) -> Integer.compare(a[0], b[0]));
        for(int i = 0; i < d.length; i++){
            int monitorsNeed = order[i][1]; //Số monitors cần cho đơn hàng
            //Nếu monitors còn lại đủ cho 1 đơn hàng
            if(p >= monitorsNeed){
                p -= monitorsNeed; // Trừ số màn hình đã giao đi
                fullFilled++; //Tăng số đơn hàng có thể chuẩn bị
            }else{
                break; // Không đủ monitors để tại 1 đơn hàng
            }
        }
        return fullFilled;
    }

    public static void main(String[] args) {
        // Test cases
        int[] D1 = {5, 11, 1, 3};
        int[] C1 = {6, 1, 3, 2};
        int P1 = 7;
        System.out.println(solution(D1, C1, P1));  // Expected output: 2
        int[] D2 = {10, 15, 1};
        int[] C2 = {10, 1, 2};
        int P2 = 3;
        System.out.println(solution(D2, C2, P2));  // Expected output: 1
        int[] D3 = {11, 18, 1};
        int[] C3 = {9, 18, 8};
        int P3 = 7;
        System.out.println(solution(D3, C3, P3));  // Expected output: 0
        int[] D4 = {1, 4, 2, 5};
```

```
    public static int solution(int[] A, String[] D) {
        int balance = 0;
        int[] cardPaymentsCount = new int[12]; // Mảng để lưu số lượng
giao dịch bằng thẻ trong mỗi tháng
        int[] cardPaymentsSum = new int[12];   // Mảng để lưu tổng số
tiền thanh toán bằng thẻ trong mỗi tháng
        // Duyệt qua tất cả các giao dịch
        for (int i = 0; i < A.length; i++) {
            int amount = A[i];
            String date = D[i];
            int month = Integer.parseInt(date.substring(5, 7)) - 1; //
Lấy tháng (chỉ số từ 0 đến 11)
            // Cập nhật số dư
            balance += amount;
            // Nếu là thanh toán bằng thẻ
            if (amount < 0) {
                cardPaymentsCount[month]++;
                cardPaymentsSum[month] += amount;
            }
        }
        // Áp dụng phí hàng tháng nếu cần thiết
        for (int month = 0; month < 12; month++) {
            if (cardPaymentsCount[month] < 3 ||
cardPaymentsSum[month] > -100) {
                balance -= 5;
            }
        }
        return balance;
    }

    public static void main(String[] args){
        int[] A1 = {100, 100, 100, -10};
        String[] D1 = {"2020-12-31", "2020-12-22", "2020-12-03",
"2020-12-29"};
        System.out.println(solution(A1, D1));  // Output: 230
        int[] A2 = {180, -50, -25, -25};
        String[] D2 = {"2020-01-01", "2020-01-01", "2020-01-01",
"2020-01-31"};
        System.out.println(solution(A2, D2));  // Output: 25
        int[] A3 = {1, -1, 0, -105, 1};
        String[] D3 = {"2020-12-31", "2020-04-04", "2020-04-04",
"2020-04-14", "2020-07-12"};
        System.out.println(solution(A3, D3));  // Output: -164
        int[] A4 = {100, 100, -10, -20, -30};
        String[] D4 = {"2020-01-01", "2020-02-01", "2020-02-11",
"2020-02-05", "2020-02-08"};
        System.out.println(solution(A4, D4));  // Output: 80
        int[] A5 = {-60, 60, -40, -20};
        String[] D5 = {"2020-10-01", "2020-02-02", "2020-10-10",
"2020-10-30"};
        System.out.println(solution(A5, D5));  // Output: -115
    }
}
```

CastleBuilding

```
public class CastleBuilding{
    public static int solution(int[] a){
        int p = 0, q = 0, h = 0, v = 0;
        while (q < a.length) {
            // Find range [P...Q] where all values are the same
            while (q < a.length - 1 && a[q + 1] == a[q]) {
                q++;
            }
            // After determining the range [P...Q], check if it's a hill
or a valley
            if (p == 0 && q == a.length - 1) {
                // Case: Array only has equal numbers (e.g., [1,1,1])
                return 1;
            } else if (p == 0) {
                // Case: Range [P...Q] is at the start of the array
                if (a[q] > a[q + 1]) {
                    h++;
                } else if (a[q] < a[q + 1]) {
                    v++;
                }
            } else if (q == a.length - 1) {
                // Case: Range [P...Q] is at the end of the array
                if (a[p] > a[p - 1]) {
                    h++;
                } else if (a[p] < a[p - 1]) {
                    v++;
                }
            } else {
                // Case: Range [P...Q] is in the middle of the array
                if (a[p] > a[p - 1] && a[q] > a[q + 1]) {
                    h++;
                } else if (a[p] < a[p - 1] && a[q] < a[q + 1]) {
                    v++;
                }
            }
            p = ++q;
        }
        return h + v;
    }

    public static void main(String[] args) {
        int[][] testCases = {
            {2, 2, 3, 4, 3, 3, 2, 2, 1, 1, 2, 5},   // 2 hills, 2 valleys
            {3, 3, 3, 3, 3, 3, 3, 3, 3, 1},         // 1 hill, 1 valley
            {2, 1, 2, 1, 2, 1, 2, 1},               // 4 hills, 4 valleys
            {5, 4, 4, 3, 2, 1, 1, 1},               // 1 hill, 1 valley
            {1, 2, 3, 2, 1, 1, 2, 3, 2, 1},         // 2 hills, 3 valleys
            {5},                                    // Neither hill nor
valley
            {2, 2, 2, 1, 1, 2, 2, 2},               // 2 hills, 1 valley
            {4, 4, 4, 5, 5, 4, 4, 4},               // 1 hill, 2 valleys
            {1, 2, 3, 4, 5, 2, 1},                  // 1 hill, 2 valleys
        };
        for (int i = 0; i < testCases.length; i++) {
            int res = solution(testCases[i]);
            System.out.println(solution(D2, C2, P2));  // Expected output: 1
            int[] D3 = {11, 18, 1};
            int[] C3 = {9, 18, 8};
            int P3 = 7;
            System.out.println(solution(D3, C3, P3));  // Expected output: 0
            int[] D4 = {1, 4, 2, 5};
            int[] C4 = {4, 9, 2, 3};
            int P4 = 19;
            System.out.println(solution(D4, C4, P4));  // Expected output: 4
        }
    }
}
```

SameDigitMerge

```
public class SameDigitMerge {
// Hàm lấy chữ số đầu tiên của một số
    private static int getFirstDigit(int number) {
// Chia số cho 10 cho đến khi số nhỏ hơn 10
        while (number >= 10) {
            number /= 10;
        }
        return number;
    }

// Hàm lấy chữ số cuối của một số
    private static int getLastDigit(int number) {
// Trả về phần dư của số khi chia cho 10 (chữ số cuối cùng)
        return number % 10;
    }

    public static int solution(int[] numbers){
// Mảng đếm số lần xuất hiện của các chữ số cuối
        int[] lastDigitCount = new int[10];

// Mảng đếm số lần xuất hiện của các chữ số đầu
        int[] firstDigitCount = new int[10];
        for (int number : numbers) {
            int firstDigit = getFirstDigit(number);
            int lastDigit = getLastDigit(number);

            firstDigitCount[firstDigit]++;
            lastDigitCount[lastDigit]++;
        }
        int total = 0;
        for (int digit = 0; digit < 10; digit++) {
// Tính tổng số cặp chữ số đầu và cuối giống nhau
            total += lastDigitCount[digit] * firstDigitCount[digit];
        }
        return total;
    }

    public static void main(String[] args) {
        // Test cases
        int[] sampleNumbers1 = {30, 12, 29, 91};
        int[] sampleNumbers2 = {122, 21, 21, 23};
        System.out.println(solution(sampleNumbers1)); // Output: 3
        System.out.println(solution(sampleNumbers2)); // Output: 5
    }
}
```

ShortestUniqueSubstring

```
import java.util.HashMap;

public class ShortestUniqueSubstring {
    public static int solution(String s){
        int N = s.length();

        for (int length = 1; length <= N; length++) {
// Tạo một HashMap để lưu trữ các chuỗi con và số lần xuất hiện của chúng
            HashMap<String, Integer> substringCount = new HashMap<>();

 // Tạo tất cả các chuỗi con có độ dài hiện tại
            for (int i = 0; i <= N - length; i++) {
                String substring = s.substring(i, i + length);

// Tăng số lần xuất hiện của chuỗi con trong HashMap
                substringCount.put(substring, substringCount.getOrDefault(substring, 0) + 1);
            }

// Kiểm tra xem có chuỗi con nào xuất hiện đúng một lần không
            for (String substring : substringCount.keySet()) {
                if (substringCount.get(substring) == 1) {
                    return length;
                }
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        System.out.println(solution("abaaba")); // Output: 2
        System.out.println(solution("zyzyzyz")); // Output: 5
        System.out.println(solution("aabbbabaaa")); // Output: 3
    }
}
```

SmallestDigitSum

```
public class SmallestDigitSum {
    public static int solution(int n){
        if (n == 0) return 0;

// Mảng để lưu số lần xuất hiện của các chữ số từ 0 đến 9
        int[] digits = new int[10];
        int res = 0;

        for (int digit = 9; digit > 0; digit--) {
// Trong khi n lớn hơn hoặc bằng chữ số hiện tại
            while (n >= digit) {
// Tăng số lần xuất hiện của chữ số hiện tại
                digits[digit]++;
```

```
            {2, 2, 2, 1, 1, 2, 2, 2},               // 2 hills, 1 valley
            {4, 4, 4, 5, 5, 4, 4, 4},               // 1 hill, 2 valleys
            {1, 2, 3, 4, 5, 2, 1},                  // 1 hill, 2 valleys
        };
        for (int i = 0; i < testCases.length; i++) {
            int res = solution(testCases[i]);
            System.out.println("Result: " + res);
        }
    }
}
```

CommonLetter

```
import java.util.*;

public class CommonLetter {
    public int[] solution(String[] s){
        int n = s.length; // Lấy số lượng chuỗi trong mảng
        int m = s[0].length(); // Lấy độ dài của chuỗi đầu tiên (giả sử
tất cả các chuỗi đều có cùng độ dài)

        HashMap<String, Integer> map = new HashMap<>();
        for(int i = 0; i < m; i++){
            for(int j = 0; j < n; j++){

// Tạo một chuỗi đại diện cho ký tự tại vị trí (j, i) và cột i
                String a = s[j].charAt(i) + ":" + i;

                if(map.containsKey(a)){

// Nếu tồn tại, trả về mảng chứa vị trí của ký tự trùng lặp
                    return new int[]{map.get(a), j, i};
                }
                map.put(a, j); // Nếu không tồn tại, thêm chuỗi này vào
HashMap với giá trị là chỉ số hàng
            }
        }
        return new int[]{};
    }

    public static void main(String[] args){
        CommonLetter c = new CommonLetter();
        String[] s1 = {"abc", "bca", "dbe"};
        String[] s2 = {"zzzz", "ferz", "zdsr", "fgtd"};
        String[] s3 = {"gr", "sd", "rg"};
        String[] s4 = {"bdafg", "ceagi"};
        System.out.println(Arrays.toString(c.solution(s1)));
        System.out.println(Arrays.toString(c.solution(s2)));
        System.out.println(Arrays.toString(c.solution(s3)));
        System.out.println(Arrays.toString(c.solution(s4)));
    }
}
```

CountBananas

```
public class countBananas {
    public static int solution(String s){
        int countA = 0;
        int countB = 0;
        int countN = 0;

        for (char c : s.toCharArray()) {
            switch (c) {
                case 'B':
                    countB++;
                    break;
                case 'A':
                    countA++;
                    break;
                case 'N':
                    countN++;
                    break;
            }
        }

        // Calculate the maximum number of times "BANANA" can be formed
        int numBANANAs = Math.min(countB, Math.min(countA / 3, countN /
2));

        return numBANANAs;
    }

    public static void main(String[] args){
        System.out.println(solution("NAABXXAN"));
        System.out.println(solution("NAANAAXNABABYNNBZ"));
        System.out.println(solution("QABAAAWOBL"));
    }
}
```

CreatePalindrome

```
import java.util.Random;

public class CreatePalindrome{
    public static String solution(String s){
        int n = s.length();
        char[] c = s.toCharArray();  // Chuyển chuỗi đầu vào thành mảng
ký tự
        Random r = new Random();

        for(int i = 0; i < n/2; i++){
            int j = n - 1 - i; // Tính chỉ số đối xứng với I

            if(c[i] == '?' && c[j] == '?'){
// Sinh một ký tự ngẫu nhiên từ 'a' đến 'z'
                char randomChar = (char) ('a' + r.nextInt(26));
                c[i] = c[j] = randomChar;
            }else if(c[i] == '?'){
                c[i] = c[j]; // Gán ký tự tại vị trí j cho vị trí i
            }else if(c[j] == '?'){
                c[j] = c[i]; // Gán ký tự tại vị trí i cho vị trí j
        for (int digit = 9; digit > 0; digit--) {
// Trong khi n lớn hơn hoặc bằng chữ số hiện tại
            while (n >= digit) {
// Tăng số lần xuất hiện của chữ số hiện tại
                digits[digit]++;
// Giảm n đi giá trị của chữ số hiện tại
                n -= digit;
            }
        }

        for (int digit = 1; digit <= 9; digit++) {
 // Trong khi chữ số hiện tại vẫn còn xuất hiện
            while (digits[digit] > 0) {
// Thêm chữ số hiện tại vào kết quả
                res = res * 10 + digit;
// Giảm số lần xuất hiện của chữ số hiện tại
                digits[digit]--;
            }
        }
        return res;
    }

    public static void main(String[] args) {
        System.out.println(solution(16)); // Output: 79
        System.out.println(solution(19)); // Output: 199
        System.out.println(solution(7));  // Output: 7
    }
}
```

TheWidestPath

```
import java.util.Arrays;

public class TheWidestPath {
    public static int solution(int[] X, int[] Y) {
// Sắp xếp mảng X theo thứ tự tăng dần
        Arrays.sort(X);
// Khởi tạo biến maxGap để lưu khoảng cách lớn nhất
        int maxGap = 0;

        for (int i = 1; i < X.length; i++) {
// Tính khoảng cách giữa phần tử hiện tại và phần tử trước đó
            int gap = X[i] - X[i - 1];

// Nếu khoảng cách hiện tại lớn hơn maxGap, cập nhật maxGap
            if (gap > maxGap) {
                maxGap = gap;
            }
        }

        return maxGap;
    }

    public static void main(String[] args) {
        // Test cases
        int[] X1 = {1, 8, 7, 3, 4, 1, 8};
        int[] Y1 = {6, 4, 1, 8, 5, 1, 7};
        System.out.println(solution(X1, Y1));  // Output: 3
        int[] X2 = {5, 5, 5, 7, 7, 7};
        int[] Y2 = {3, 4, 5, 1, 3, 7};
        System.out.println(solution(X2, Y2));  // Output: 2
        int[] X3 = {6, 10, 1, 4, 3};
        int[] Y3 = {2, 5, 3, 1, 6};
        System.out.println(solution(X3, Y3));  // Output: 4
        int[] X4 = {4, 1, 5, 4};
        int[] Y4 = {4, 5, 1, 3};
        System.out.println(solution(X4, Y4));  // Output: 3
    }
}
```

ValueOccurrences

```
public class ValueOccurrences{
    public static int solution(int[] a){
        int moves = 0; // lưu số lần di chuyển cần thiết
        int curNum = a[0];  // lưu giá trị hiện tại của phần tử đầu tiên
        int count = 1; // đếm số lần xuất hiện của curNum

        for(int i = 1; i <= a.length; i++){
// Nếu phần tử hiện tại bằng với curNum, tăng biến count
            if(i < a.length && a[i] == curNum){
                count++;
            }else{
// Nếu count nhỏ hơn curNum, tăng moves bằng giá trị nhỏ hơn giữa curNum - count và count
                if(count < curNum){
                    moves += Math.min(curNum - count, count);
                }else if(count > curNum){ // Nếu count lớn hơn curNum, tăng moves bằng count - curNum
                    moves += count - curNum;
                }

 // Nếu còn phần tử trong mảng, cập nhật curNum và reset count
                if(i < a.length){
                    curNum = a[i];
                    count = 1;
                }
            }
        }
        return moves;
    }

    public static void main(String[] args) {
        // Ví dụ 1
        int[] A1 = {1, 1, 3, 4, 4, 4};
        System.out.println("Example 1: " + solution(A1));  // Output: 3
        // Ví dụ 2
        int[] A2 = {1, 2, 2, 2, 5, 5, 5, 8};
        System.out.println("Example 2: " + solution(A2));  // Output: 4
        // Ví dụ 3
```

```
                char randomChar = (char) ('a' + r.nextInt(26));
                c[i] = c[j] = randomChar;
            }else if(c[i] == '?'){
                c[i] = c[j]; // Gán ký tự tại vị trí j cho vị trí i
            }else if(c[j] == '?'){
                c[j] = c[i]; // Gán ký tự tại vị trí i cho vị trí j
            }else if(c[i] != c[j]){
                return "NO"; // Trả về "NO" vì không thể tạo thành chuỗi
đối xứng
            }
        }
// Nếu độ dài chuỗi là lẻ và ký tự ở giữa là '?'
        if(n % 2 == 1 && c[n/2] == '?'){
            c[n/2] = 'a'; // Gán ký tự 'a' cho vị trí giữa
        }
        return new String(c);
    }

    public static void main(String[] args){
        System.out.println(solution("?ab??a"));
        System.out.println(solution("bab??a"));
        System.out.println(solution("?a?"));
    }
}
```

DiversityString

```
import  java.util.*;

public class DiversityString {
    public static String solution(int n){
    //C1: Không shuffle các chữ cái
        // String alphabet = "abcdefghijklmnopqrstuvwxyz";
        // StringBuilder res = new StringBuilder();

        // for(int i = 0; i < n; i++){
        //     res.append(alphabet.charAt(i % 26));
        // }
        // return res.toString();

        //C2:
        Random r = new Random();

// Tạo StringBuilder với dung lượng ban đầu là n
        StringBuilder res = new StringBuilder(n);

        for (int i = 0; i < n; i++) {
            char randChar = (char) ('a' + r.nextInt(26)); // Tạo ký tự
ngẫu nhiên từ 'a' đến 'z'
            res.append(randChar);  // Thêm ký tự ngẫu nhiên vào
StringBuilder
        }

        return res.toString();
    }

    public static void main(String[] args) {
        System.out.println(solution(3));   // Output: "abc"
        System.out.println(solution(5));   // Output: "abcde"
        System.out.println(solution(30));  // Output:
"abcdefghijklmnopqrstuvwxyzabcd"
        System.out.println(solution(52));  // Output:
"abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz"
    }
}
```

EndsTheSame (M)

```
public class EndsTheSame {
    public static int solution(String s){
        // int N = s.length();
        // int count = 0;
        // if (s.charAt(0) == s.charAt(N - 1)) {
        //     count++;
        // }
        // for (int i = 1; i < N; i++) {
        //     if (s.charAt(i) == s.charAt(i - 1)) {
        //         count++;
        //     }
        // }

        // return count;


        //Dùng đệ qui
        // Kiểm tra nếu chuỗi chỉ có 1 ký tự
        if (s.length() == 1) {
            return 0;  // Chỉ có một ký tự thì không thể có ký tự liền kề
giống nhau
        }
        int dem = (s.charAt(0) == s.charAt(s.length() - 1)) ? 1 : 0;  //
Kiểm tra ký tự đầu và cuối
        // Hàm đệ quy xử lý phần còn lại của chuỗi từ vị trí 1 đến N-1
        return dem + countAdjacent(s, 1, s.length());
    }
    private static int countAdjacent(String s, int a, int n) {
        // Điều kiện dừng: Khi đã duyệt hết chuỗi
        if (a == n) {
            return 0;
        }

        // So sánh ký tự hiện tại với ký tự trước đó
        int count = (s.charAt(a) == s.charAt(a - 1)) ? 1 : 0;

        // Đệ quy kiểm tra phần còn lại của chuỗi
        return count + countAdjacent(s, a + 1, n);
    }

    public static void main(String[] args){
        System.out.println(solution("abbaa")); // Output: 3
        System.out.println(solution("aaaa")); // Output: 4
```

```
        int[] A1 = {1, 1, 3, 4, 4, 4};
        System.out.println("Example 1: " + solution(A1));  // Output: 3
        // Ví dụ 2
        int[] A2 = {1, 2, 2, 2, 5, 5, 5, 8};
        System.out.println("Example 2: " + solution(A2));  // Output: 4
        // Ví dụ 3
        int[] A3 = {1, 1, 1, 1, 3, 3, 4, 4, 4, 4, 4};
        System.out.println("Example 3: " + solution(A3));  // Output: 5
        // Ví dụ 4
        int[] A4 = {10, 10, 10};
        System.out.println("Example 4: " + solution(A4));  // Output: 3
    }
}
```

```
        System.out.println(solution("abab")); // Output: 0
    }
}
```

EraseOneLetter

```
public class EraseOneLetter {
    public static String solution(String s) {
        int n = s.length();

        for (int i = 0; i < n - 1; i++) {
            if (s.charAt(i) > s.charAt(i + 1)) {
                // Loại bỏ ký tự ở vị trí i
                return s.substring(0, i) + s.substring(i + 1);
            }
        }
        return s.substring(0, n - 1);
    }

    public static void main(String[] args) {
        System.out.println(solution("acb"));       // Output: "ab"
        System.out.println(solution("hot"));       // Output: "ho"
        System.out.println(solution("codility"));  // Output: "cdility"
        System.out.println(solution("aaaa"));      // Output: "aaa"
    }
}
```

EvenPairsOnCycle

```
public class EvenPairsOnCycle {
    public static int solution(int[] a){
        //Theo dõi các phần tử đã sử dụng
         boolean[] usedElement = new boolean[a.length];
         int count = 0;

         for(int i = 0; i < a.length; i++){
 //Tính index của phần tử kế tiếp (vòng tròn)
             int indexElement = (i + 1) % a.length;

 //Kiểm tra nếu cả hai phần tử chưa được sử dụng và tổng của chúng là số
chẵn
             if(!usedElement[i] && !usedElement[indexElement] && (a[i] +
a[indexElement]) % 2 == 0){
                 usedElement[i] = true;
                 usedElement[indexElement] = true;
                 count++;
             }
         }
         return count;
     }

    public static void main(String[] args) {
        // Test cases
        int[] A1 = {4, 2, 5, 8, 7, 3, 7};
        System.out.println(solution(A1));  // Kết quả: 2
        int[] A2 = {14, 21, 16, 35, 22};
        System.out.println(solution(A2));  // Kết quả: 1
        int[] A3 = {5, 5, 5, 5, 5, 5};
        System.out.println(solution(A3));  // Kết quả: 3
    }
}
```

ForbiddenTriosSwaps

```
public class ForbiddenTriosSwaps {
    public static int solution(String s){
        int swap = 0; // Biến đếm số lần hoán đổi cần thiết
        int count = 1; // Biến đếm số lượng ký tự liên tiếp giống nhau

        for(int i = 1; i < s.length() - 1; i++){
// Nếu ký tự hiện tại giống ký tự trước đó
            if(s.charAt(i) == s.charAt(i - 1)){
                count += 1;
            }else{
// Nếu có ít nhất 3 ký tự liên tiếp giống nhau
                if(count >= 3){
 // Tính số lần hoán đổi cần thiết và cộng vào biến swap
                    swap += (count - 1) / 2;
                }
                count = 1;  // Đặt lại biến đếm số lượng ký tự liên tiếp
            }
        }

// Kiểm tra lần cuối nếu chuỗi kết thúc bằng một dãy ký tự liên tiếp
giống nhau
        if(count >= 3){
            swap += (count - 1) / 2;  // Tính số lần hoán đổi cần thiết
và cộng vào biến swap
        }
        return swap;
    }

    public static void main(String[] args) {
        // Test cases
        System.out.println(solution("baaaaa"));        // Output: 1
        System.out.println(solution("baaabbaabbba"));  // Output: 2
        System.out.println(solution("baabab"));        // Output: 0
    }
}
```