Task 3 Ans

18 Tháng Chín 2024 6:21 CH

DiceRolls

```
import java.util.Arrays;

public class DiceRolls {
    public static int[] solution(int[] a, int f, int m){
        //Tổng lần đổ xúc xắc
        int diceRoll = a.length + f;
        int total = m * diceRoll;
        //Tính tổng của A
        int rememberRes = 0;
        for (int numA : a){
            rememberRes += numA;
        }

// Tính tổng điểm của các lần đổ xúc xắc bị quên
        int forgotSum = total - rememberRes;

// Kiểm tra nếu tổng điểm của các lần đổ xúc xắc bị quên không hợp lệ
        if(forgotSum < f || forgotSum > 6 * f){
            return new int[]{0};
        }

// Khởi tạo mảng kết quả để lưu các giá trị của các lần đổ xúc xắc bị quên
        int[] res = new int[f];
        for (int i = 0; i < f; i++) {
// Tính giá trị của lần đổ xúc xắc hiện tại, không vượt quá 6
            res[i] = Math.min(6, forgotSum - (f - 1 - i));

 // Giảm tổng điểm của các lần đổ xúc xắc bị quên
            forgotSum -= res[i];
        }
        return res;
    }

     public static void main(String[] args) {
        // Test case 1
        int[] A1 = {3, 2, 4, 3};
        int F1 = 2;
        int M1 = 4;
        System.out.println(Arrays.toString(solution(A1, F1, M1)));  //
Expected output: [6, 6]
        // Test case 2
        int[] A2 = {1, 5, 6};
        int F2 = 4;
        int M2 = 3;
        System.out.println(Arrays.toString(solution(A2, F2, M2)));  //
Expected output: possible result like [2, 1, 2, 4]
        // Test case 3
        int[] A3 = {1, 2, 3, 4};
        int F3 = 4;
        int M3 = 6;
        System.out.println(Arrays.toString(solution(A3, F3, M3)));  //
Expected output: [0]
        // Test case 4
        int[] A4 = {6, 1};
        int F4 = 1;
        int M4 = 1;
        System.out.println(Arrays.toString(solution(A4, F4, M4)));  //
Expected output: [0]
    }
}

//sumA: lần đổ nhớ (N là phần tử trong A)
//sumB: lần đổ quên (F là số lần đổ xúc xắc quên)
//M = (sumA + sumB) / (N + F)
// => sumB =  M * (N + F) - sumA
//Tổng sum(B) của các kết quả quên phải nằm trong khoảng từ F đến 6 * F
//(vì mỗi lần đổ xúc xắc có giá trị từ 1 đến 6).
//Nếu sum(B) < F: Không thể có kết quả hợp lệ vì tổng cần thấp hơn số lượng
kết quả.
//Nếu sum(B) > 6 * F: Không thể có kết quả hợp lệ vì tổng cần cao hơn giá
trị tối đa có thể đạt được.
```

DistinctNumbersCount

```
import java.util.*;

public class DistinctNumbersCount {
    public static int solution(int[] a){
        //Priority Queue
        // int deleteCount = 0;
        // //Đếm tần suất xuất hiện của các phần tử
        // Map<Integer, Integer> freq = new HashMap<>();
        // for(int n : a){
        //     freq.put(n, freq.getOrDefault(n, 0) + 1);
        // }
        // //Đưa tần suất vào Priority Queue
        // PriorityQueue<Integer> max_Heap = new PriorityQueue<>
(Collections.reverseOrder());
        // for(int freqs : freq.values()){
        //     max_Heap.add(freqs);
        // }
        // //Xử lý các tần suất trùng lặp
        // while(!max_Heap.isEmpty()){
        //     int curFreq = max_Heap.poll();
        //     if(!max_Heap.isEmpty() && curFreq == max_Heap.peek()){
        //         // Nếu tần suất hiện tại trùng với tần suất tiếp theo
        //         if(curFreq > 1){
        //             // Giảm tần suất xuống và đưa lại vào heap
        //             max_Heap.add(curFreq - 1);
        //         }
        //         deleteCount++;
        //     }
        // }
        // return deleteCount;

        //TreeMap or TreeSet
        int deleteCount = 0;
        Map<Integer, Integer> freq = new HashMap<>();
        for (int num : a) {
            freq.put(num, freq.getOrDefault(num, 0) + 1);
        }
        TreeSet<Integer> uniqueFrequencies = new TreeSet<>();
        for (int frequency : freq.values()) {
            // Bước 3: Nếu tần suất đã tồn tại trong TreeSet, ta phải điều
chỉnh nó
            while (frequency > 0 && uniqueFrequencies.contains(frequency)) {
                frequency--;  // Giảm tần suất
                deleteCount++;  // Tăng số lần xóa
            }
            // Thêm tần suất không trùng vào TreeSet
            if (frequency > 0) {
                uniqueFrequencies.add(frequency);
            }
        }
        return deleteCount;
    }

    public static void main(String[] args) {
        // Các test case
        int[] A1 = {1, 1, 2, 2, 2};
        int[] A2 = {5, 3, 3, 2, 5, 2, 3, 2};
        int[] A3 = {127, 15, 3, 8, 10};
        int[] A4 = {10000000, 10000000, 5, 5, 5, 2, 2, 2, 0, 0};
        System.out.println(solution(A1));  // Expected output: 1
        System.out.println(solution(A2));  // Expected output: 2
        System.out.println(solution(A3));  // Expected output: 4
        System.out.println(solution(A4));  // Expected output: 4
    }
}
```

BankTransfer

```
public class BankTransfer {
    public int[] solution(String r, int[] v){
        //Duyệt tuần tự và tính số dư tối thiểu
        int n = r.length();
        int minA = 0, curA = 0;
        int minB = 0, curB = 0;

        for(int i = 0; i < n; i++){
            if(r.charAt(i) == 'A'){
                curB -= v[i];
                curA += v[i];
                if(curB < minB){ // Cập nhật số dư tối thiểu cho B (nếu B bị
âm)
                    minB = curB;
                }
            }else{
                curA -= v[i];
                curB += v[i];
                if(curA < minA){
                    minA = curA;
                }
            }

            minA = Math.min(minA, curA);
            minB = Math.min(minB, curB);
        }
        return new int[]{-minA, -minB};
    }

    public static void main(String[] args) {
        BankTransfer b = new BankTransfer();

        String R1 = "BAABA";
        int[] V1 = {2, 4, 1, 1, 2};
        int[] result1 = b.solution(R1, V1);
        System.out.println("Minimum balance: " + result1[0] + ", " +
result1[1]);  // Expected: [2, 4]
```

EqualSegments (M)

```
import java.util.*;

public class EqualSegments {
    public static int solution(int[] a){
        if (a.length < 2) return 0; // Not enough elements to form a segment

        // Step 1: Calculate sums of all segments of length 2
        // Map<Integer, Integer> total = new HashMap<>();
        // for (int i = 0; i < a.length - 1; i++) {
        //     int sum = a[i] + a[i + 1];
        //     total.put(sum, total.getOrDefault(sum, 0) + 1);
        // }
        // // Step 2: Find the maximum number of non-intersecting segments for each sum
        // int maxSeg = 0;
        // for (Map.Entry<Integer, Integer> entry : total.entrySet()) {
        //     int currentSum = entry.getKey();

        //     // Greedily select non-intersecting segments with the same sum
        //     int seg = 0;
        //     int i = 0;
        //     while (i < a.length - 1) {
        //         if (a[i] + a[i + 1] == currentSum) {
        //             seg++;
        //             i += 2; // Move to the next segment to avoid overlap
        //         } else {
        //             i++;
        //         }
        //     }

        //     maxSeg = Math.max(maxSeg, seg);
        // }

        // return maxSeg;

        HashMap<Integer, Integer> sumFrequency = new HashMap<>();
        int n = a.length;
        // Bước 1: Duyệt qua mảng để tính tổng của các đoạn con liên tiếp có chiều dài 2
        for (int i = 0; i < n - 1; i++) {
            int sum = a[i] + a[i + 1];
            sumFrequency.put(sum, sumFrequency.getOrDefault(sum, 0) + 1);
        }
        // Bước 2: Tìm tổng có tần suất cao nhất
        int maxSum = 0;
        int maxFreq = 0;
        for (int sum : sumFrequency.keySet()) {
            if (sumFrequency.get(sum) > maxFreq) {
                maxFreq = sumFrequency.get(sum);
                maxSum = sum;
            }
        }
        // Bước 3: Đếm số lượng đoạn không giao nhau với tổng bằng maxSum
        int count = 0;
        for (int i = 0; i < n - 1; i++) {
            if (a[i] + a[i + 1] == maxSum) {
                count++;
                i++;  // Bỏ qua đoạn giao nhau
            }
        }
        return count;
    }

    public static void main(String[] args) {
        // Test cases
        int[] A1 = {10, 1, 3, 1, 2, 2, 1, 0, 4};
        System.out.println(solution(A1));  // Expected output: 3

        int[] A2 = {5, 3, 1, 3, 2, 3};
        System.out.println(solution(A2));  // Expected output: 1

        int[] A3 = {9, 9, 9, 9, 9};
        System.out.println(solution(A3));  // Expected output: 2

        int[] A4 = {1, 5, 2, 4, 3, 3};
        System.out.println(solution(A4));  // Expected output: 3
    }
}
```

FreeingStorageSpace (M)

```
//sliding window
//Chỉ chạy đúng 3/4 test case
import java.util.HashMap;

public class FreeingStorageSpace {

    public static int solution(int[] A, int R){
        int n = A.length;
// Nếu cần loại bỏ tất cả các kệ, không còn loại mặt hàng nào
        if (R == n) {
            return 0;
        }

        // Khởi tạo HashMap để đếm số lượng từng loại mặt hàng
        HashMap<Integer, Integer> itemCount = new HashMap<>();
        for (int item : A) {
            itemCount.put(item, itemCount.getOrDefault(item, 0) + 1);
        }

        int maxTypes = itemCount.size(); // Số loại mặt hàng ban đầu
        int currentTypes = maxTypes;

        // Sử dụng Sliding Window để loại bỏ R kệ liên tiếp
        for (int i = 0; i <= n - R; i++) {
            HashMap<Integer, Integer> removedItems = new HashMap<>();
            for (int j = i; j < i + R; j++) {
                int item = A[j];
                removedItems.put(item, removedItems.getOrDefault(item, 0) + 1);
                if (itemCount.get(item) - removedItems.get(item) == 0) {
                    currentTypes--;
                }
            }

            maxTypes = Math.max(maxTypes, currentTypes);

            // Khôi phục lại số lượng mặt hàng cho cửa sổ tiếp theo
            for (int j = i; j < i + R; j++) {
                int item = A[j];
                if (itemCount.get(item) - removedItems.get(item) == 0) {
                    currentTypes++;
                }
                removedItems.put(item, removedItems.get(item) - 1);
            }
        }
        return maxTypes;
    }

    public static void main(String[] args) {
        // Test cases
        int[] A1 = {2, 1, 2, 3, 2, 2};
        System.out.println(solution(A1, 3));  // Expected output: 2
        int[] A2 = {2, 3, 1, 1, 2};
        System.out.println(solution(A2, 2));  // Expected output: 3
        int[] A3 = {20, 10, 10, 20, 30, 20};
        System.out.println(solution(A3, 3));  // Expected output: 3
        int[] A4 = {1, 100000, 1};
        System.out.println(solution(A4, 3));  // Expected output: 0
    }
}
```

GardenArrangement (M)

```
public class GardenArrangement {
    public static long solution(int[] a){
        long sumTrees = 0;

        // Step 1: Calculate total number of trees
        for (int trees : a) {
            sumTrees += trees;
        }

        // Step 2: Determine target number of trees per section
        long temp = (sumTrees + a.length - 1) / a.length; // Equivalent to Math.ceil(totalTrees / N)
        long planNeeded = 0, //to plan & to move

        // Step 3: Calculate the total actions needed
        for (int trees : a) {
            if (trees < temp) {
                planNeeded += (temp - trees); // Trees needed
            }
            // We do not need to count excess trees because they can be moved
        }
        return planNeeded;
    }

    public static void main(String[] args) {
        // Test cases
        System.out.println(solution(new int[]{1, 2, 2, 4})); // Expected output: 4
        System.out.println(solution(new int[]{4, 2, 4, 6})); // Expected output: 2
        System.out.println(solution(new int[]{1, 1, 2, 1})); // Expected output: 3
    }
}
```

LongestEvenCount (M)

```
import java.util.HashMap;
public class LongestEvenCount {
    public static int solution(String s){
        int maxLength = 0;
```

ChoosingNumbers (M)

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;
public class ChoosingNumbers {
    public static int solution(int[] a) {
        int n = a.length;
        if (n < 2) return n; // Không đủ số để tạo thành một chuỗi
        Arrays.sort(a); // Sắp xếp mảng
        int maxCount = 1; // Tối đa 1 số nếu không có dãy số nào dài hơn

        // Tạo tập hợp các số để kiểm tra sự tồn tại
        Set<Integer> set = new HashSet<>();

        for (int num : a) {
            set.add(num);
        }

        // Duyệt các cặp số khác nhau để tính sự khác biệt
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                int d = a[j] - a[i];
                if (d == 0) continue; // Bỏ qua sự khác biệt bằng 0
                int count = 0;
                int current = a[i];

                // Đếm số lượng phần tử theo mẫu khác biệt
                while (set.contains(current)) {
                    count++;
                    current += d;
                }
                maxCount = Math.max(maxCount, count);
            }
        }

        // Xử lý trường hợp tất cả số giống nhau
        int identicalCount = 1;
        for (int i = 1; i < n; i++) {
            if (a[i] == a[i - 1]) identicalCount++;
            else {
                maxCount = Math.max(maxCount, identicalCount);
                identicalCount = 1;
            }
        }
        maxCount = Math.max(maxCount, identicalCount);
        return maxCount;
    }

    public static void main(String[] args) {
        // Test cases
        int[] A1 = {4, 7, 1, 5, 3};
        System.out.println(solution(A1)); // 4

        int[] A2 = {12, 12, 12, 15, 10};
        System.out.println(solution(A2)); // 3

        int[] A3 = {18, 26, 18, 24, 24, 20, 22};
        System.out.println(solution(A3)); // 5
    }
}
```

CleaningRobot

CollectingRainwater

```
public class CollectingRainWater {
    public static int solution(String s) {
        int countTanks = 0;

        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == 'H') { // Kiểm tra nếu vị trí hiện tại là nhà
                // Kiểm tra nếu nhà có thể được che phủ bởi bể chứa ở bên phải
                if (i < s.length() - 1 && s.charAt(i + 1) == '-') {
                    countTanks++;
                    i += 2; // Bỏ qua nhà tiếp theo vì nó đã được che phủ
                }
                // Kiểm tra nếu nhà có thể được che phủ bởi bể chứa ở bên trái
                else if (i > 0 && s.charAt(i - 1) == '-') {
                    countTanks++;
                }
                // Nếu không có ô trống ở cả hai bên, trả về -1
                else {
                    return -1;
                }
            }
        }
        return countTanks;
    }

    public static void main(String[] args) {
        // Test cases
        System.out.println(solution("-H-HH--"));  // Expected output: 2
        System.out.println(solution("H"));        // Expected output: -1
        System.out.println(solution("HH-HH"));    // Expected output: -1
        System.out.println(solution("-H-H-H-H-H")); // Expected output: 3
    }
}
```

CreateDiverseWord

```
public class CreateDeverseWord {
    public String solution(int AA, int AB, int BB){
        String res = "";
        int sumStr = AA + AB + BB; // Tính tổng số chuỗi con cần sử dụng

        if(AA == 0 && BB == 0 && AB == 0) return ""; //nếu tất cả đều bằng 0, trả về chuỗi rỗng
        while (sumStr > 0) {

            // Nếu còn chuỗi "AA" và việc thêm "AA" không tạo ra "AAA"
            if (AA > 0 && (res.length() < 2 || !res.substring(res.length() - 2).equals("AA"))) {
                res += "AA";
                AA--;  // Giảm số lượng chuỗi "AA"
            }
            // Nếu còn chuỗi "BB" và việc thêm "BB" không tạo ra "BBB"
            else if (BB > 0 && (res.length() < 2 || !res.substring(res.length() - 2).equals("BB"))) {
                res += "BB";
                BB--; // Giảm số lượng chuỗi "BB"
            } else if (AB > 0) { // Nếu còn chuỗi "AB"
                res += "AB";
                AB--; // Giảm số lượng chuỗi "AB"
            } else {
                break;  // Nếu không thể thêm chuỗi nào nữa mà không vi phạm điều kiện, thoát khỏi vòng l
            }
            sumStr--; // Giảm tổng số chuỗi con cần sử dụng
        }
        return res;
    }

    public static void main(String[] args) {
        CreateDeverseWord sol = new CreateDeverseWord();

        // Test cases
        System.out.println(sol.solution(5, 0, 2)); // Expected: AABBAABBAA
        System.out.println(sol.solution(1, 2, 1)); // Possible: ABABABAA
        System.out.println(sol.solution(0, 2, 0)); // Expected: ABAB
        System.out.println(sol.solution(0, 0, 10)); // Expected: BB
        System.out.println(sol.solution(0, 0, 0));
    }
}

//Điều kiện cơ bản:
//Nếu AA, AB và BB đều bằng 0 thì trả về một chuỗi trống.
//Nếu AA hoặc BB lớn hơn gấp đôi tổng của hai số còn lại thì không thể tránh khỏi
//các "AAA" hoặc "BBB" liên tiếp. Vì vậy, hãy xử lý vấn đề này bằng cách hạn chế sử dụng dãy AA hoặc BB.
//Greedy
//Bắt đầu xây dựng chuỗi bằng cách xen kẽ giữa "AA", "AB" và "BB" trong khi tôn trọng các
//điều kiện "AAA" và "BBB".
//Ưu tiên sử dụng chuỗi thường xuyên nhất trước tiên để đảm bảo bạn cân bằng việc sử dụng AA và BB.
```

DivedeIntoGroups (M)

```
import java.util.Arrays;

public class DivideIntoGroups {
    public int solution(int[] A) {
        // Sắp xếp mảng A theo thứ tự tăng dần
        Arrays.sort(A);
        int n = A.length; // Lấy kích thước của mảng
        int minDifference = Integer.MAX_VALUE; // Khởi tạo biến để lưu sự khác biệt tối thiểu, bắt đầu v
lớn nhất

        // Duyệt tất cả các cặp chỉ số i và j để tìm sự khác biệt tối đa trong các đoạn
        for (int i = 1; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                // Tính toán sự khác biệt tối đa giữa ba đoạn:
                // 1. Từ phần tử đầu tiên đến phần tử trước i
                // 2. Từ phần tử i đến phần tử trước j
                // 3. Từ phần tử j đến phần tử cuối cùng
                int maxDifference = Math.max(A[i - 1] - A[0], Math.max(A[j - 1] - A[i], A[n - 1] - A[j]));

                // Cập nhật sự khác biệt tối thiểu
                minDifference = Math.min(minDifference, maxDifference);
            }
        }

        return minDifference; // Trả về sự khác biệt tối thiểu
    }

    public static void main(String[] args) {
```

lập

ới giá trị

);

```java
    }

    public static void main(String[] args) {
        BankTransfer b = new BankTransfer();

        String R1 = "BAABA";
        int[] V1 = {2, 4, 1, 1, 2};
        int[] result1 = b.solution(R1, V1);
        System.out.println("Minimum balance: " + result1[0] + ", " +
result1[1]);  // Expected: [2, 4]

        String R2 = "ABAB";
        int[] V2 = {10, 5, 10, 15};
        int[] result2 = b.solution(R2, V2);
        System.out.println("Minimum balance: " + result2[0] + ", " +
result2[1]);  // Expected: [0, 15]

        String R3 = "B";
        int[] V3 = {100};
        int[] result3 = b.solution(R3, V3);
        System.out.println("Minimum balance: " + result3[0] + ", " +
result3[1]);  // Expected: [100, 0]
    }
}

//C1:
// Khởi tạo số dư ban đầu của A và B là 0.
// Duyệt qua từng giao dịch trong mảng R và V.
// Nếu giao dịch là chuyển tiền vào A (ngân hàng B chuyển cho A), tăng số dư
của A và giảm số dư của B.
// Nếu giao dịch là chuyển tiền vào B (ngân hàng A chuyển cho B), tăng số dư
của B và giảm số dư của A.
// Mỗi lần số dư của một ngân hàng bị âm, cập nhật giá trị số dư tối thiểu
cần thiết.
```

MaxSwitchingSlice

```java
public class MaxSwitchingSlice {
    public int solution(int[] a){
        //Two Pointers

        // int n = a.length;
        // if(n == 1){
        //     return 1;
        // }
        // int maxLength = 1;
        // int curLength = 1;

        // // Bắt đầu từ phần tử thứ hai và duyệt đến hết mảng
        // for(int i = 1; i < n; i++){
        //     if(i >= 2 && a[i] == a[i - 2]) {
        //         curLength++; // Tăng độ dài nếu thỏa mãn điều kiện
switching
        //     } else {
        //         curLength = 2; // Đặt lại độ dài nếu không thỏa mãn
        //     }
        //     maxLength = Math.max(maxLength, curLength); // Cập nhật độ
dài lớn nhất
        // }
        // return maxLength;


        //Greedy
        int n = a.length;
        if (n == 1) return 1;

        int maxLen = 1; // Khởi tạo biến lưu độ dài lớn nhất của dãy con
        int currentLen = 1; // Khởi tạo biến lưu độ dài hiện tại của dãy con

        int evenVal = a[0];  // Giá trị ban đầu cho các vị trí chẵn
        int oddVal = a.length > 1 ? a[1] : a[0];  // Giá trị ban đầu cho các
vị trí lẻ

        for (int i = 1; i < n; i++) {
            if (i % 2 == 0) {
                if (a[i] == evenVal) {
                    currentLen++; // Tăng độ dài hiện tại nếu giá trị bằng
với giá trị chẵn ban đầu
                } else {
                    evenVal = a[i]; // Cập nhật giá trị chẵn mới
                    currentLen = 2;  // Đặt lại độ dài hiện tại
                }
            } else {
                if (a[i] == oddVal) {
                    currentLen++;
                } else {
                    oddVal = a[i];
                    currentLen = 2;
                }
            }
            maxLen = Math.max(maxLen, currentLen); // Cập nhật độ dài lớn
nhất
        }
        return maxLen;
    }

    public static void main(String[] args) {
        MaxSwitchingSlice maxSwitch = new MaxSwitchingSlice();
        // Test case 1
        int[] A1 = {3, 2, 3, 2, 3};
        System.out.println("Test case 1: " + maxSwitch.solution(A1)); //
Expected output: 5
        // Test case 2
        int[] A2 = {7, 4, -2, 4, -2, -9};
        System.out.println("Test case 2: " + maxSwitch.solution(A2)); //
Expected output: 4
        // Test case 3
        int[] A3 = {7, -5, -5, -5, 7, -1, 7};
        System.out.println("Test case 3: " + maxSwitch.solution(A3)); //
Expected output: 3
        // Test case 4
        int[] A4 = {4};
        System.out.println("Test case 4: " + maxSwitch.solution(A4)); //
Expected output: 1
    }
}

//C1
// Sử dụng hai con trỏ (hoặc biến) để duyệt qua mảng từ đầu đến cuối và theo
dõi độ dài
//của mảng con thỏa mãn điều kiện "switching slice".
// Sử dụng một biến để lưu độ dài của mảng con hiện tại.
// Tại mỗi bước duyệt, kiểm tra xem các phần tử ở vị trí chẵn có bằng nhau
không
//và các phần tử ở vị trí lẻ có bằng nhau không.
// Nếu không thỏa mãn, cập nhật độ dài mảng con mới.
//C2
// Ta duyệt qua mảng và lưu hai biến để theo dõi phần tử hiện tại ở vị trí
chẵn và vị trí lẻ.
// Tại mỗi bước, nếu phần tử hiện tại không thỏa mãn điều kiện "switching
slice", reset lại độ dài slice.
// Sau khi duyệt hết mảng, tìm được độ dài mảng con dài nhất.
```

BattleshipRecognition

```java
import java.util.*;

public class BattleshipRecognition {
    public static int[] solution(String[] b){
        //BFS (queue)
        int n = b.length; // Số hàng của bảng
        int m = b[0].length(); // Số cột của bảng

        int patrol = 0, submarine = 0, destroyer = 0;

        char[][] board = new char[n][m]; // Bảng ký tự
        boolean[][] temp = new boolean[n][m]; // Bảng đánh dấu các ô đã
duyệt

        for (int i = 0; i < n; i++){
            board[i] = b[i].toCharArray();
        }
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
 // Nếu ô hiện tại là '#' và chưa được duyệt
                if(board[i][j] == '#' && !temp[i][j]){
                    int s = bfs(board, temp, i, j);

// Nếu kích thước là 1, tăng biến đếm tàu tuần tra
                    if(s == 1) patrol++;
                    else if(s == 2) submarine++;
                    else if(s == 3) destroyer++;
                }
            }
        }
        return new int[]{patrol, submarine, destroyer};
    }

    private static int bfs(char[][] b, boolean[][] t, int x, int y){
        int n = b.length;
        int m = b[0].length;

        int[] dX = {-1, 1, 0, 0}; // Hướng di chuyển theo hàng
        int[] dY = {0, 0, -1, 1}; // Hướng di chuyển theo cột

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{x, y});
        t[x][y] = true; // Đánh dấu ô bắt đầu đã được duyệt
        int s = 0;

        while(!q.isEmpty()){
            int[] c = q.poll();
            int cX = c[0]; // Tọa độ hàng của ô hiện tại
            int cY = c[1]; // Tọa độ cột của ô hiện tại
```

```java
        System.out.println(solution(new int[]{4, 2, 4, 6})); // Expected output: 2
        System.out.println(solution(new int[]{1, 1, 2, 1})); // Expected output: 3
    }
}
```

LongestEvenCount (M)

```java
import java.util.HashMap;
public class LongestEvenCount {
    public static int solution(String s){
        int maxLength = 0;

        // Duyệt từng điểm bắt đầu của cửa sổ con
        for (int start = 0; start < s.length(); start++) {
            HashMap<Character, Integer> freq = new HashMap<>();

            // Duyệt từng điểm kết thúc của cửa sổ con
            for (int end = start; end < s.length(); end++) {
                char currentChar = s.charAt(end);

                // Cập nhật số lần xuất hiện của ký tự hiện tại
                freq.put(currentChar, freq.getOrDefault(currentChar, 0) + 1);

                // Kiểm tra nếu tất cả ký tự trong cửa sổ con đều có số lần xuất hiện chẵn
                if (allEven(freq)) {
                    maxLength = Math.max(maxLength, end - start + 1);
                }
            }
        }
        return maxLength;
    }

    // Kiểm tra tất cả các giá trị trong HashMap đều chẵn
    private static boolean allEven(HashMap<Character, Integer> freq) {
        for (int count : freq.values()) {
            if (count % 2 != 0) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        System.out.println(solution("bdaaadadb")); // Output: 6
        System.out.println(solution("abacb"));    // Output: 0
        System.out.println(solution("zthtzh"));   // Output: 6
    }
}
```

ArraySlicing (M)

```java
public class ArraySlicing {
    public static int solution(int[] a){
        if (a.length == 1) return 1; //Mảng có 1 phần tử
        int s = 0;
        int[] maxR = new int[a.length]; //Giá trị lớn nhất từ đầu đến i
        int[] minL = new int[a.length]; //Giá trị nhỏ nhất từ đầu đến I

        maxR[0] = a[0];
        for (int i = 1; i < a.length; i++) {
// Cập nhật giá trị lớn nhất từ đầu đến i
            maxR[i] = Math.max(maxR[i - 1], a[i]);
        }

        minL[a.length - 1] = a[a.length - 1];
        for (int i = a.length - 2; i >= 0; i--) {
            minL[i] = Math.min(minL[i + 1], a[i]);
        }

        //Đếm số đoạn cắt
        for (int i = 0; i < a.length - 1; i++) {
            if (maxR[i] <= minL[i + 1]) {
// Tăng biến đếm nếu giá trị lớn nhất từ đầu đến i nhỏ hơn hoặc bằng giá trị nhỏ nhất từ i+1 đến cuối
                s++;
            }
        }
        return s + 1; // Thêm 1 cho đoạn cắt cuối
    }

    public static void main(String[] args) {
        System.out.println(solution(new int[]{2, 4, 1, 6, 5, 9, 7})); // Output: 3
        System.out.println(solution(new int[]{4, 3, 2, 6, 1}));       // Output: 1
        System.out.println(solution(new int[]{2, 1, 6, 4, 3, 7}));    // Output: 3
    }
}
```

LongestTwoDigitFragment (M)

```java
public class LongestTwoDigitFragment {
    public int solution(int[] a) {
        int lenMax = 0;
        int n = a.length;

        // Duyệt qua tất cả các điểm bắt đầu có thể của mảng con
        for (int i = 0; i < n; i++) {
            boolean[] numUsed = new boolean[10]; // Mảng đánh dấu xem chữ số nào đã được sử dụng
            int count = 0; // Đếm số lượng các chữ số khác nhau trong mảng con
            boolean flag = true; // Biến đánh dấu để kiểm tra tính hợp lệ của mảng con

            // Kiểm tra mọi điểm kết thúc có thể của mảng con bắt đầu từ `start`
            for (int j = i; j < n; j++) {
                String nums = Integer.toString(a[j]);
// Kiểm tra từng ký tự trong chuỗi số
                for (char c : nums.toCharArray()) {
                    if (!numUsed[c - '0']) { // Nếu chữ số chưa được sử dụng
                        numUsed[c - '0'] = true; // Đánh dấu chữ số này đã được sử dụng
                        count++; // Tăng số lượng chữ số khác nhau
                    }
                }

                // Nếu sử dụng nhiều hơn hai chữ số khác nhau, dừng lại
                if (count > 2) {
                    flag = false; // Đặt cờ `flag` thành false để dừng kiểm tra mảng con
                    break;
                }

                // Nếu mảng con hiện tại hợp lệ (vẫn chỉ có tối đa 2 chữ số khác nhau)
                if (flag) {
                    lenMax = Math.max(lenMax, j - i + 1); // Cập nhật độ dài tối đa của mảng con
                }
            }
        }
        return lenMax;
    }

    public static void main(String[] args) {
        LongestTwoDigitFragment sol = new LongestTwoDigitFragment();
        int[] A1 = {23, 333, 33, 30, 0, 505};
        int[] A2 = {615, 88, 498, 99, 9};
        int[] A3 = {123, 456};
        System.out.println(sol.solution(A1)); // Output: 4
        System.out.println(sol.solution(A2)); // Output: 2
        System.out.println(sol.solution(A3)); // Output: 0
    }
}
```

MinDistinct (M)

```java
import java.util.PriorityQueue;
public class MinDistinct {
    public static int solution(int[] A) {
        int N = A.length;
        int[] count = new int[N + 1];  // Mảng tần số cho các số trong phạm vi [1, N]

        // Đếm tần suất của mỗi số trong mảng
        for (int num : A) {
            count[num]++;
        }

        // Hàng đợi ưu tiên để lưu trữ các số bị thiếu theo thứ tự tăng dần
        PriorityQueue<Integer> missingNumbers = new PriorityQueue<>();
        for (int i = 1; i <= N; i++) {
            if (count[i] == 0) {
                missingNumbers.add(i);  // Thu thập số còn thiếu
            }
        }

        long moves = 0;

        // Xử lý từng số trong mảng
        for (int i = 1; i <= N; i++) {
            while (count[i] > 1) {  // Nếu chúng ta có bản sao của 'i'
                if (missingNumbers.isEmpty()) {
                    return -1;  // Trường hợp không thể xảy ra, nhưng để đảm bảo an toàn
                }
                int missing = missingNumbers.poll();  // Lấy số bị thiếu nhỏ nhất
                moves += Math.abs(i - missing);  // Tính chi phí để thay đổi số dư thừa thành số bị thiếu
                if (moves > 1_000_000_000) {
                    return -1;  // Thoát sớm nếu chi phí vượt quá giới hạn
                }
                count[i]--;  // Đã xử lý một số dư thừa
            }
        }
        return (int) moves;
    }

    public static void main(String[] args) {
        // Testing the function with provided examples
        System.out.println(solution(new int[]{1, 2, 1}));          // Expected output: 2
        System.out.println(solution(new int[]{2, 1, 4, 4}));       // Expected output: 1
        System.out.println(solution(new int[]{6, 2, 3, 5, 6, 3})); // Expected output: 4
    }
}
```

PathDetection (M)

```java
import java.util.HashMap;
```

```java
                // Cập nhật sự khác biệt tối thiểu
                minDifference = Math.min(minDifference, maxDifference);
            }
        }

        return minDifference; // Trả về sự khác biệt tối thiểu
    }

    public static void main(String[] args) {
        DivideIntoGroups sol = new DivideIntoGroups();
        int[] A1 = {11, 5, 3, 12, 6, 8, 1, 7, 4};
        int[] A2 = {10, 14, 12, 1000, 11, 15, 13, 1};
        int[] A3 = {4, 5, 7, 10, 10, 12, 12, 12};
        int[] A4 = {5, 10, 10, 5, 5};
        System.out.println(sol.solution(A1)); // Output: 3
        System.out.println(sol.solution(A2)); // Output: 5
        System.out.println(sol.solution(A3)); // Output: 2
        System.out.println(sol.solution(A4)); // Output: 0
    }
}
```

SquareTiles (M)

```java
public class SquareTiles {
    public static int solution(int M, int N) {
        // Calculate the total area available
        int totalArea = M + 4 * N;

        // Xác định độ dài cạnh lớn nhất có thể
        int maxSide = (int) Math.sqrt(totalArea);

        // Kiểm tra xem hình vuông có thể được điền đầy đủ không
        while (maxSide > 0) {
            int requiredArea = maxSide * maxSide;
            int remainingArea = requiredArea;

            // Sử dụng càng nhiều ô 2x2 càng tốt
            int used2x2Tiles = Math.min(N, remainingArea / 4);
            remainingArea -= used2x2Tiles * 4;

            // Sử dụng gạch 1x1 cho khu vực còn lại
            if (remainingArea <= M) {
                return maxSide;
            }

            // Hãy thử một chiều dài cạnh nhỏ hơn
            maxSide--;
        }

        // Nếu không thể tạo được hình vuông, trả về 0
        return 0;
    }

    public static void main(String[] args) {
        // Test cases
        System.out.println(solution(8, 0)); // Output: 2
        System.out.println(solution(4, 3)); // Output: 4
        System.out.println(solution(0, 18)); // Output: 8
        System.out.println(solution(13, 3)); // Output: 5
    }
}
```

StringMisspellingVar(Opal Variant)

```java
public class OpalVariant {
    public static String solution(String S, String T) {
        // Kiểm tra nếu hai chuỗi bằng nhau
        if (S.equals(T)) {
            return "EQUAL";  // Trả về "EQUAL" nếu hai chuỗi hoàn toàn giống nhau
        }

        // Kiểm tra nếu chuỗi T dài hơn chuỗi S một ký tự
        if (T.length() == S.length() + 1) {
            for (int i = 0; i < T.length(); i++) {
                // Tạo một chuỗi mới bằng cách loại bỏ ký tự tại vị trí i trong T
                String modified = T.substring(0, i) + T.substring(i + 1);
                // So sánh chuỗi mới với chuỗi S
                if (modified.equals(S)) {
                    return "INSERT " + T.charAt(i);  // Trả về "INSERT" và ký tự bị loại bỏ
                }
            }
        }

        // Kiểm tra nếu chuỗi S dài hơn chuỗi T một ký tự
        if (S.length() == T.length() + 1) {
            for (int i = 0; i < S.length(); i++) {
                // Tạo một chuỗi mới bằng cách loại bỏ ký tự tại vị trí i trong S
                String modified = S.substring(0, i) + S.substring(i + 1);
                // So sánh chuỗi mới với chuỗi T
                if (modified.equals(T)) {
                    return "REMOVE " + S.charAt(i);  // Trả về "REMOVE" và ký tự bị loại bỏ
                }
            }
        }

        // Kiểm tra nếu chuỗi S và T có độ dài bằng nhau
        if (S.length() == T.length()) {
            for (int i = 0; i < S.length() - 1; i++) {
                // Kiểm tra điều kiện để xác định nếu có thể thực hiện hoán đổi hai ký tự
                if (S.charAt(i) != T.charAt(i) && S.charAt(i + 1) != T.charAt(i + 1) &&
                    S.charAt(i) == T.charAt(i + 1) && S.charAt(i + 1) == T.charAt(i)) {
                    return "SWAP " + S.charAt(i);  // Trả về "SWAP" và ký tự đã hoán đổi
                }
            }
        }

        // Nếu không thỏa mãn bất kỳ điều kiện nào, trả về "IMPOSSIBLE"
        return "IMPOSSIBLE";
    }

        public static void main(String[] args) {
            System.out.println(solution("gain", "again")); // INSERT a
            System.out.println(solution("parks", "park")); // REMOVE s
            System.out.println(solution("form", "from")); // SWAP o
            System.out.println(solution("o", "odd")); // IMPOSSIBLE
            System.out.println(solution("fift", "fifth")); // IMPOSSIBLE
        }
}
```

WordSplit (M)

```java
import java.util.HashSet;

public class WordSplit {
    public int solution(String S) {
        // Khởi tạo một HashSet để theo dõi các ký tự đã thấy trong một phần của chuỗi
        HashSet<Character> seen = new HashSet<>();

        // Khởi tạo biến để đếm số lượng phần chuỗi con không chứa ký tự trùng lặp
        int substrCount = 1; // Bắt đầu với một phần chuỗi con

        // Duyệt qua từng ký tự trong chuỗi
        for (char c : S.toCharArray()) {
            // Nếu ký tự hiện tại đã xuất hiện trước đó trong phần chuỗi con hiện tại
            if (seen.contains(c)) {
                // Tăng số lượng phần chuỗi con (bắt đầu một phần chuỗi con mới)
                substrCount++;
                // Xóa các ký tự đã thấy trong phần chuỗi con hiện tại
                seen.clear();
            }
            // Thêm ký tự hiện tại vào HashSet
            seen.add(c);
        }

        // Trả về số lượng phần chuỗi con không chứa ký tự trùng lặp
        return substrCount;
    }

    public static void main(String[] args) {
        WordSplit sol = new WordSplit();
        System.out.println(sol.solution("world")); // Output: 1
        System.out.println(sol.solution("dddd"));   // Output: 4
        System.out.println(sol.solution("cycle")); // Output: 2
        System.out.println(sol.solution("abba")); // Output: 3
    }
}
```

RecyclingTrucks (M)

```java
public class RecyclingTrucks {

public int solution(int[] D, String[] T) {
        int n = D.length;
        // Khởi tạo mảng times để lưu thời gian mỗi loại rác
        // times[0] cho rác nhựa (Plastic), times[1] cho thủy tinh (Glass), times[2] cho kim loại (Metal
        int[] times = new int[3];
        // Khởi tạo mảng lastHouse để lưu chỉ số của ngôi nhà cuối cùng chứa từng loại rác
        int[] lastHouse = new int[3];
        // Khởi tạo mảng travelTimes để lưu thời gian di chuyển cho từng loại rác
        int[] travelTimes = new int[3];

        // Duyệt qua từng ngôi nhà để tính tổng thời gian thu gom
        for (int i = 0; i < n; i++) {
            // Duyệt qua từng loại rác trong ngôi nhà hiện tại
            for (char c : T[i].toCharArray()) {
                // Nếu loại rác là nhựa (P), cập nhật thời gian và vị trí ngôi nhà cuối cùng có rác nhựa
                if (c == 'P') {
                    times[0] += 1; // Thêm thời gian tải rác nhựa
                    lastHouse[0] = i;
                }
                // Nếu loại rác là thủy tinh (G), cập nhật thời gian và vị trí ngôi nhà cuối cùng có rác
                else if (c == 'G') {
                    times[1] += 1; // Thêm thời gian tải rác thủy tinh
                    lastHouse[1] = i;
                }
                // Nếu loại rác là kim loại (M), cập nhật thời gian và vị trí ngôi nhà cuối cùng có rác k
```

thủy tinh

kim loại

```
        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{x, y});
        t[x][y] = true; // Đánh dấu ô bắt đầu đã được duyệt
        int s = 0;

        while(!q.isEmpty()){
            int[] c = q.poll();
            int cX = c[0]; // Tọa độ hàng của ô hiện tại
            int cY = c[1]; // Tọa độ cột của ô hiện tại
            s++;

 // Duyệt qua các hướng di chuyển
            for(int i = 0; i < 4; i++){
                int nX = cX + dX[i]; // Tọa độ hàng của ô kế tiếp
                int nY = cY + dY[i]; // Tọa độ cột của ô kế tiếp

// Kiểm tra nếu ô kế tiếp hợp lệ và chưa được duyệt
                if(nX >= 0 && nY >= 0 && nX < n && nY < m && !t[nX][nY] &&
b[nX][nY] == '#'){
                    t[nX][nY] = true;
                    q.add(new int[]{nX, nY});
                }
            }
        }
        return s; // Trả về kích thước của tàu
    }

    public static void main(String[] args) {
        // Test case 1
        String[] B1 = {".##.#", "#.#..", "#...#", "#.##."};
        System.out.println("Number of ship: " +
Arrays.toString(solution(B1))); // Expected output: [2, 1, 2]
        // Test case 2
        String[] B2 = {".#..#", "##..#", "...#."};
        System.out.println("Number of ship: " +
Arrays.toString(solution(B2))); // Expected output: [1, 1, 1]
        // Test case 3
        String[] B3 = {"##.", "#.#", ".##"};
        System.out.println("Number of ship: " +
Arrays.toString(solution(B3))); // Expected output: [0, 0, 2]
        // Test case 4
        String[] B4 = {"...", "...", "..."};
        System.out.println("Number of ship: " +
Arrays.toString(solution(B4))); // Expected output: [0, 0, 0]
    }
}
```

SanatoriumAccommodation

```
import java.util.Arrays;

public class SanatoriumAccommodation {
    public static int solution(int[] a){
        // Greedy
        Arrays.sort(a); // Sắp xếp khách theo số lượng giới hạn
        int rooms = 0;
        int i = 0;

        while (i < a.length) {
            int groupSize = a[i]; // Lấy giới hạn của khách hiện tại
            rooms++; // Mở phòng mới
            i += groupSize; // Chuyển đến khách tiếp theo sau nhóm
        }
        return rooms;
    }

    public static void main(String[] args) {
        int[] A1 = {1, 1, 1, 1, 1};
        System.out.println(solution(A1)); // Output: 5
        int[] A2 = {2, 1, 4};
        System.out.println(solution(A2)); // Output: 2
        int[] A3 = {2, 7, 2, 9, 8};
        System.out.println(solution(A3)); // Output: 2
        int[] A4 = {7, 3, 1, 1, 4, 5, 4, 9};
        System.out.println(solution(A4)); // Output: 4
    }
}
```

FixTheTable

```
import java.util.Arrays;

public class FixTheTable {
    public static int solution(int[] a){
        Arrays.sort(a);

        int l = 0; // Chiều dài nhỏ nhất có thể của bảng
        int r = a[a.length - 1] - a[0]; // Chiều dài bảng lớn nhất có thể

        // Nếu chỉ có một lỗ thì chúng ta chỉ cần 1 tấm ván
        if (a.length == 1) {
            return 1;
        }
        if(a.length == 0) return 0;

        // Binary search over the board length(duyệt theo chiều dài)
        while(l < r){
            int m = (l + r) / 2;
            if(checkCoverTwoBoard(a, m)){
                r = m; // Thử độ dài ngắn hơnh
            }else{
                l = m + 1; //Tăng độ dài
            }
        }
        return l; // Độ dài tối thiểu có thể hoạt động
    }

     // Hàm kiểm tra xem hai tấm ván có chiều dài 'L' có thể che hết tất cả
các lỗ không
    private static boolean checkCoverTwoBoard(int[] a, int size){
        // Đặt tấm ván đầu tiên bắt đầu từ lỗ đầu tiên
        int endBoard1 = a[0] + size;
        int i = 0;

        // Bỏ qua tất cả các lỗ mà bảng đầu tiên có thể che phủ
        while(i < a.length && a[i] <= endBoard1){
            i++;
        }

        // Đặt bảng thứ hai
        if(i < a.length){
            int endBoard2 = a[i] + size;

            // Kiểm tra xem tấm ván thứ hai có thể che được các lỗ còn lại
không
            while(i < a.length && a[i] <= endBoard2){
                i++;
            }
        }
        // Nếu tất cả các lỗ được che phủ, trả về true
        return i == a.length;
    }

    public static void main(String[] args) {
        // Test cases
        int[] A1 = {11, 20, 15};
        System.out.println(solution(A1));  // Expected output: 4

        int[] A2 = {15, 20, 9, 11};
        System.out.println(solution(A2));  // Expected output: 5

        int[] A3 = {0, 44, 32, 30, 42, 18, 34, 16, 35};
        System.out.println(solution(A3));  // Expected output: 18

        int[] A4 = {9};
        System.out.println(solution(A4));  // Expected output: 1
    }
}
```

XYSplit

```
public class XYSplit {
    public static int solution(String s){
        int sumX = 0, sumY = 0;
        int lX = 0, lY = 0; //Đếm bên trái

        //đếm tổng x, y có trong chuỗi
        for(char c : s.toCharArray()){
            if(c == 'x') sumX++;
            if(c == 'y') sumY++;
        }
        int split = 0;

        // Duyệt qua tất cả các vị trí có thể tách chuỗi
        for(int i = 0; i < s.length() - 1; i++){
            char c = s.charAt(i);

            // Cập nhật số lượng 'x' và 'y' cho phần bên trái
            if(c == 'x'){
                lX++;
            }else if(c == 'y'){
                lY++;
            }
             // Tính số lượng 'x' và 'y' cho phần bên phải
            int rX = sumX - lX;
            int rY = sumY - lY;
            if(lX == lY || rX == rY){
                split++;
            }
        }
        return split;
    }

    public static void main(String[] args) {
        // Testing the function with provided examples
        System.out.println(solution(new int[]{1, 2, 1}));        // Expected output: 2
        System.out.println(solution(new int[]{2, 1, 4, 4}));     // Expected output: 1
        System.out.println(solution(new int[]{6, 2, 3, 5, 6, 3})); // Expected output: 4
    }
}
```

PathDetection (M)

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Map;
import java.util.Queue;
import java.util.Set;
public class PathDetection {
    public static boolean solution(int n, int[] a, int[] b) {
    // Khởi tạo một mảng để theo dõi các kết nối trực tiếp
    // directConnections[i] sẽ là true nếu có kết nối trực tiếp từ đỉnh i+1 đến đỉnh i+2
    boolean[] directConnections = new boolean[N - 1];

    // Duyệt qua các cạnh và đánh dấu các kết nối
    for (int i = 0; i < A.length; i++) {
        // Nếu có kết nối từ đỉnh A[i] đến đỉnh B[i] - 1
        if (A[i] == B[i] - 1) {
            directConnections[A[i] - 1] = true;
        // Nếu có kết nối từ đỉnh B[i] đến đỉnh A[i] - 1
        } else if (B[i] == A[i] - 1) {
            directConnections[B[i] - 1] = true;
        }
    }

    // Kiểm tra xem tất cả các kết nối cần thiết có tồn tại không
    for (int i = 0; i < N - 1; i++) {
        // Nếu không có kết nối trực tiếp từ đỉnh i+1 đến đỉnh i+2
        if (!directConnections[i]) {
            return false;
        }
    }

    // Nếu tất cả các kết nối cần thiết đều tồn tại
    return true;

        //BFS (cách 2)
        Queue<Integer> q = new LinkedList<>();
        Map<Integer, Set<Integer>> g = new HashMap<>();
        for(int i = 1; i <= n; i++){
            g.put(i, new HashSet<>());
        }
        for(int i = 0; i < a.length; i++){
            int edge = a[i];
            int vertice = b[i];
            g.get(edge).add(vertice);
            g.get(vertice).add(edge);
        }
        if(!g.containsKey(1) || !g.containsKey(n)) return false;
        boolean[] flag = new boolean[n + 1];
        q.offer(1);
        flag[1] = true;
        while(!q.isEmpty()){
            int node = q.poll();
            if(node == n){
                for(int i = 1; i < n; i++){
                    if(!g.get(i).contains(i + 1)) return false;
                }
                return true;
            }
            for(int j : g.get(node)){
                if(!flag[j]){
                    flag[j] = true;
                    q.offer(j);
                }
            }
        }
        return false;


    }

    public static void main(String[] args) {
        // Test cases
        System.out.println(solution(4, new int[]{1, 2, 4, 4, 3}, new int[]{2, 3, 1, 3, 1})); // Should return true
        System.out.println(solution(4, new int[]{1, 2, 1, 3}, new int[]{2, 4, 3, 4})); // Should return false
        System.out.println(solution(6, new int[]{2, 4, 5, 3}, new int[]{3, 5, 6, 4})); // Should return false
        System.out.println(solution(3, new int[]{1, 2}, new int[]{2, 3})); // Should return true
    }
}
```

PriceFluctuation (M)

```
public class MaxIncomeCalculator {
    // Phương thức tính toán thu nhập tối đa
    public static int solution(int[] A) {
        long maxIncome = 0; // Biến lưu trữ thu nhập tối đa
        int n = A.length; // Độ dài của mảng A

        // Duyệt qua mảng từ phần tử thứ 2 đến phần tử cuối cùng
        for (int i = 1; i < n; i++) {
            // Nếu phần tử hiện tại lớn hơn phần tử trước đó
            if (A[i] > A[i - 1]) {
                // Cộng phần chênh lệch vào thu nhập tối đa
                maxIncome += A[i] - A[i - 1];
            }
        }

        // Trả về thu nhập tối đa sau khi lấy phần dư với 1 tỷ
        return (int) (maxIncome % 1_000_000_000);
    }

    public static void main(String[] args) {
        // Khởi tạo mảng ví dụ
        int[] A1 = {5, 1, 5, 1, 5};

        // In ra kết quả của phương thức solution với mảng A1
        System.out.println(solution(A1));
    }
}
```

Sticks (M)

```
//Binary Search
public class Sticks {
    public static int solution(int a, int b) {
        int trai = 0;
        int phai = (a + b) / 4;
        int canhToiDa = 0;

        // Sử dụng tìm kiếm nhị phân để tìm độ dài lớn nhất
        while (trai <= phai) {
            int g = (trai + phai) / 2;
            if (canMakeSquare(a, b, g)) {
                canhToiDa = g;
                trai = g + 1; // Tìm kiếm ở phía lớn hơn
            } else {
                phai = g - 1; // Tìm kiếm ở phía nhỏ hơn
            }
        }

        return canhToiDa;
    }

    // Kiểm tra xem có thể tạo được 4 cạnh có độ dài 'side' không
    private static boolean canMakeSquare(int a, int b, int n) {
        if (n == 0) return false; // Không thể tạo cạnh bằng 0
        int demA = a / n; // Số thanh có thể cắt từ A
        int demB = a / n; // Số thanh có thể cắt từ B
        return demA + demB >= 4; // Cần ít nhất 4 thanh để tạo hình vuông
    }

    public static void main(String[] args) {
        // Các ví dụ kiểm tra
        System.out.println(solution(10, 21)); // Nên trả về 7
        System.out.println(solution(13, 11)); // Nên trả về 5
        System.out.println(solution(2, 1));   // Nên trả về 0
        System.out.println(solution(1, 8));   // Nên trả về 2
    }
}
```

AngryFrogs (M)

```
public class AngryFrog {
    public static int solution(int[] blocks) {
        // int n = blocks.length;
        // int[] left = new int[n];
        // int[] right = new int[n];

        // // Khởi tạo giá trị ban đầu
        // left[0] = 1;
        // right[n - 1] = 1;

        // // Tính khoảng cách nhảy sang trái (giảm chiều cao)
        // for (int i = 1; i < n; i++) {
        //     if (blocks[i] <= blocks[i - 1]) {
        //         left[i] = left[i - 1] + 1;
        //     } else {
        //         left[i] = 1;
        //     }
        // }

        // // Tính khoảng cách nhảy sang phải (tăng chiều cao)
        // for (int i = n - 2; i >= 0; i--) {
        //     if (blocks[i] <= blocks[i + 1]) {
        //         right[i] = right[i + 1] + 1;
        //     } else {
        //         right[i] = 1;
```



```
                times[0] += 1; // Thêm thời gian tải rác nhựa
                lastHouse[0] = i;
            }
            // Nếu loại rác là thủy tinh (G), cập nhật thời gian và vị trí ngôi nhà cuối cùng có rác
            else if (c == 'G') {
                times[1] += 1; // Thêm thời gian tải rác thủy tinh
                lastHouse[1] = i;
            }
            // Nếu loại rác là kim loại (M), cập nhật thời gian và vị trí ngôi nhà cuối cùng có rác
            else if (c == 'M') {
                times[2] += 1; // Thêm thời gian tải rác kim loại
                lastHouse[2] = i;
            }
        }
    }

    // Tính tổng thời gian di chuyển cho tất cả các ngôi nhà
    int totalTravelTime = 0;
    for (int i = 0; i < n; i++) {
        totalTravelTime += D[i];
        // Cập nhật thời gian di chuyển cho từng loại rác
        for (int j = 0; j < 3; j++) {
            if (i <= lastHouse[j]) {
                travelTimes[j] = totalTravelTime;
            }
        }
    }

    // Tính thời gian di chuyển cho mỗi loại xe tải
    for (int i = 0; i < 3; i++) {
        // Nếu có ít nhất một ngôi nhà có loại rác tương ứng
        if (lastHouse[i] >= 0) {
            // Thêm thời gian di chuyển hai chiều
            times[i] += travelTimes[i] * 2;
        }
    }

    // Trả về thời gian tối đa của tất cả các xe tải (thời gian lâu nhất)
    return Math.max(times[0], Math.max(times[1], times[2]));
}

    public static void main(String[] args) {
        RecyclingTrucks sol = new RecyclingTrucks();
        int[] D1 = {2, 5};
        String[] T1 = {"PGP", "M"};
        System.out.println(sol.solution(D1, T1)); // Output: 15

        int[] D2 = {3, 2, 4};
        String[] T2 = {"MPM", "", "G"};
        System.out.println(sol.solution(D2, T2)); // Output: 19

        int[] D3 = {2, 1, 1, 1, 2};
        String[] T3 = {"", "PP", "PP", "GM", ""};
        System.out.println(sol.solution(D3, T3)); // Output: 12
    }
}
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class PotholeFixer {
    public static int solution(String S, int B) {
        List<Integer> segments = new ArrayList<>();
        int n = S.length();
        int i = 0;

        // Identify segments of consecutive potholes
        while (i < n) {
            if (S.charAt(i) == 'x') {
                int j = i;
                while (j < n && S.charAt(j) == 'x') {
                  j++;
                }
                segments.add(j - i); // length of consecutive 'x'
                i = j;
            } else {
                i++;
            }
        }

        // Sort segments by their fixing cost (length + 1)
        segments.sort(Comparator.comparingInt(a -> a + 1));

        int totalPotholesFixed = 0;
        for (int segment : segments) {
            int cost = segment + 1;
            if (B >= cost) {
                B -= cost;
                totalPotholesFixed += segment;
            } else {
                break;
            }
        }

        return totalPotholesFixed;
    }

    public static void main(String[] args) {
        System.out.println(solution("...xxx..x....xxx..", 7)); // Output: 5
        System.out.println(solution("..xxxxx", 4)); // Output: 3
        System.out.println(solution("x.x.xxx...x", 14)); // Output: 6
        System.out.println(solution("..", 5)); // Output: 0
    }
}
```

thủy tinh

kim loại

```
        int rY = sumY - lY;
        if(lX == lY || rX == rY){
            split++;
        }
    }
    return split;
}

public static void main(String[] args) {
    // Test cases
    System.out.println(solution("ayxbx"));  // Expected output: 3
    System.out.println(solution("xzzzy"));  // Expected output: 0
    System.out.println(solution("toyxmy")); // Expected output: 5
    System.out.println(solution("apple"));  // Expected output: 4
}
}
```

```
    //     }
    // }

    // // Tính khoảng cách nhảy sang phải (tăng chiều cao)
    // for (int i = n - 2; i >= 0; i--) {
    //     if (blocks[i] <= blocks[i + 1]) {
    //         right[i] = right[i + 1] + 1;
    //     } else {
    //         right[i] = 1;
    //     }
    // }

    // // Tìm khoảng cách lớn nhất
    // int maxDistance = 1;
    // for (int i = 0; i < n; i++) {
    //     maxDistance = Math.max(maxDistance, left[i] + right[i]  - 1);
    // }
    // return maxDistance;

    int n = blocks.length;
    if (n == 2) {
        return 2;  // Trường hợp đặc biệt: nếu chỉ có 2 khối, kết quả luôn là 2
    }

    // Mảng lưu đoạn giảm dần và tăng dần
    int[] left = new int[n];
    int[] right = new int[n];

    left[0] = 1;
    right[n - 1] = 1;

    // Tính đoạn giảm dần
    for (int i = 1; i < n; i++) {
        if (blocks[i] <= blocks[i - 1]) {
            left[i] = left[i - 1] + 1;
        } else {
            left[i] = 1;
        }
    }

    // Tính đoạn tăng dần
    for (int i = n - 2; i >= 0; i--) {
        if (blocks[i] <= blocks[i + 1]) {
            right[i] = right[i + 1] + 1;
        } else {
            right[i] = 1;
        }
    }

    // Tìm đoạn dài nhất
    int maxDistance = 1;
    for (int i = 0; i < n; i++) {
        maxDistance = Math.max(maxDistance, left[i] + right[i] - 1);
    }
    return maxDistance;
}

public static void main(String[] args) {
    // Test các trường hợp
    System.out.println(solution(new int[]{2, 6, 8, 5})); // Kết quả: 3
    System.out.println(solution(new int[]{1, 5, 5, 2, 6})); // Kết quả: 4
    System.out.println(solution(new int[]{1, 1})); // Kết quả: 2
}
}
```

AssassinsStealth (M)

```
import java.util.LinkedList;
import java.util.Queue;

public class Assassin {


public static boolean solution(String[] B) {
    int n = B.length;  // Số hàng của bảng
    int m = B[0].length();  // Số cột của bảng
    boolean[][] visible = new boolean[n][m];  // Bảng để theo dõi các ô mà lính gác có thể nhìn thấy
    int startX = 0, startY = 0;  // Vị trí của sát thủ

    // Duyệt qua toàn bộ bảng để tìm vị trí của sát thủ và đánh dấu tầm nhìn của các lính gác
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            char cell = B[i].charAt(j);  // Lấy ký tự tại ô (i, j)
            if (cell == 'A') {
                startX = i;  // Lưu vị trí hàng của sát thủ
                startY = j;  // Lưu vị trí cột của sát thủ
            } else if (cell == '<' || cell == '>' || cell == '^' || cell == 'v') {
                // Nếu ô chứa lính gác, đánh dấu tầm nhìn của lính gác
                markVisibility(B, visible, i, j, cell);
            }
        }
    }

    // Kiểm tra nếu ô bắt đầu (vị trí sát thủ) hoặc ô kết thúc bị lính gác nhìn thấy
    if (visible[startX][startY] || visible[n - 1][m - 1]) {
        return false;  // Nếu có bất kỳ ô nào bị lính gác nhìn thấy, không thể đến đích
    }

    // Sử dụng BFS để tìm đường từ ô bắt đầu đến ô kết thúc
    return bfs(B, visible, startX, startY, n, m);
}

private static void markVisibility(String[] B, boolean[][] visible, int x, int y, char direction) {
    int n = B.length;  // Số hàng của bảng
    int m = B[0].length();  // Số cột của bảng
    int dx = 0, dy = 0;  // Sự thay đổi trong hàng và cột

    // Xác định sự thay đổi trong hàng và cột dựa trên hướng của lính gác
    if (direction == '<') dy = -1;  // Hướng trái
    if (direction == '>') dy = 1;  // Hướng phải
    if (direction == '^') dx = -1;  // Hướng lên
    if (direction == 'v') dx = 1;  // Hướng xuống

    int currX = x + dx;  // Vị trí hàng hiện tại
    int currY = y + dy;  // Vị trí cột hiện tại

    // Đánh dấu các ô trong tầm nhìn của lính gác cho đến khi gặp tường hoặc lính gác khác
    while (currX >= 0 && currX < n && currY >= 0 && currY < m) {
        char cell = B[currX].charAt(currY);  // Lấy ký tự tại ô hiện tại
        if (cell == 'X' || cell == '<' || cell == '>' || cell == '^' || cell == 'v') {
            break;  // Dừng lại khi gặp tường hoặc lính gác khác
        }
        visible[currX][currY] = true;  // Đánh dấu ô hiện tại là có thể bị nhìn thấy
        currX += dx;  // Di chuyển đến ô tiếp theo theo hướng
        currY += dy;  // Di chuyển đến ô tiếp theo theo hướng
    }
}

private static boolean bfs(String[] B, boolean[][] visible, int startX, int startY, int n, int m) {
    boolean[][] visited = new boolean[n][m];  // Bảng để theo dõi các ô đã được thăm
    Queue<int[]> queue = new LinkedList<>();  // Hàng đợi để thực hiện BFS
    queue.add(new int[]{startX, startY});  // Thêm ô bắt đầu vào hàng đợi
    visited[startX][startY] = true;  // Đánh dấu ô bắt đầu là đã được thăm

    // Các hướng di chuyển: phải, xuống, trái, lên
    int[][] directions = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

    // Thực hiện BFS
    while (!queue.isEmpty()) {
        int[] current = queue.poll();  // Lấy ô hiện tại từ hàng đợi
        int x = current[0];  // Vị trí hàng của ô hiện tại
        int y = current[1];  // Vị trí cột của ô hiện tại

        // Kiểm tra nếu đã đến góc dưới bên phải
        if (x == n - 1 && y == m - 1) {
            return true;  // Đã đến đích, trả về true
        }

        // Xét các hướng di chuyển: phải, xuống, trái, lên
        for (int[] dir : directions) {
            int newX = x + dir[0];  // Tính vị trí hàng của ô mới
            int newY = y + dir[1];  // Tính vị trí cột của ô mới

            // Kiểm tra điều kiện hợp lệ cho ô mới: nằm trong bảng, chưa được thăm, không bị nhìn thấy, và không
phải là tường
            if (newX >= 0 && newX < n && newY >= 0 && newY < m
                && !visited[newX][newY] && !visible[newX][newY]
                && B[newX].charAt(newY) == '.') {
                visited[newX][newY] = true;  // Đánh dấu ô mới là đã được thăm
                queue.add(new int[]{newX, newY});  // Thêm ô mới vào hàng đợi
            }
        }
    }

    // Không tìm thấy đường đi đến đích, trả về false
    return false;
}
public static void main(String[] args) {
    String[] board1 = {"X....>", "..v..X.", ".>..X..", "A......"};
    String[] board2 = {"...Xv", "AX..<", ".XX.."};
    String[] board3 = {"...", ">.A"};
    String[] board4 = {"A.v", "..."};
    System.out.println(solution(board1)); // Output: false
    System.out.println(solution(board2)); // Output: true
    System.out.println(solution(board3)); // Output: false
    System.out.println(solution(board4)); // Output: false
}
}
```