Problem Solving - III - Self Practice

1. Friendly pair

Problem statement :

Vino wants to check if the given two numbers are friendly pair or not. Friendly pair(Amicable numbers) are two different numbers related in a way such that the Ratio's number itself for each is same. Proper divisors of a number are all the positive integers that divide the number evenly, excluding the number itself. The sum of proper d all its proper divisors.

For example, the sum of proper divisors of 6 is 1 + 2 + 3 = 6.

Example 1:

```
Input : n = 6, m = 28
Output : Yes
```

> Explanation:
>
> Divisor of 6 are 1, 2, 3, 6.
>
> Divisor of 28 are 1, 2, 4, 7, 14, 28.
>
> Sum of divisor of 6 and 28 are 12 and 56 respectively.
>
> Abundancy index of 6 and 28 are 2. So they are friendly pair.

Example 2:

```
Input : n = 18, m = 26
Output : No
```

Constraints:

```
1<=N1,N2<=108
```

Analysis:

1. Read two integers n and m.
2. Find the sum of proper divisors of n and m.
3. Find the abundancy index of n and m.
4. Check if the abundancy index of n and m are equal.
5. If equal, print "Yes", else print "No".

Code:

```java
package com.self_practice;

public class FriendlyPair {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input :");
        int n = sc.nextInt();
        int m = sc.nextInt();
        int sum1 = 0, sum2 = 0;
        for(int i = 1; i <= n/2; i++) {
```

```
            if(n % i == 0) {
                sum1 += i;
            }
        }
        for(int i = 1; i <= m/2; i++) {
            if(m % i == 0) {
                sum2 += i;
            }
        }
        System.out.print("Output :");
        if(sum1/n == sum2/m) {
            System.out.println("Yes");
        } else {
            System.out.println("No");
        }
    }
}
```

Output:

```
Input : 6 28
Output : Yes
```

---

2. K pattern

Problem statement :

Print a pattern of stars in the shape of the letter 'K'. The pattern should consist of asterisks (*) forming the uppercase and lowercase letter 'K'. The program should take print the corresponding pattern.

Analysis:

1. Read the height of the letter 'K'.
2. Print the upper part of the letter 'K'.
3. Print the lower part of the letter 'K'.

Code:

```
package com.self_practice;

import java.util.Scanner;

public class KPattern {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input :");
        int height = sc.nextInt();
        System.out.println("Output :");
        for(int i = height; i >= 1; i--) {
            for(int j = 1; j <= i; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
        for(int i = 2; i <= height; i++) {
            for(int j = 1; j <= i; j++) {
                System.out.print("*");
            }
            System.out.println();
```

```
        }
    }
}
```

Output

```
Input :5
Output :
*****
****
***
**
*
**
***
****
*****
```

---

3. Count the number of subarrays

Problem Statement :

Given an array A[] of N integers and a range(L, R). The task is to find the number of subarrays having sum in the range L to R (inclusive).

Analysis:

1. Read the size of the array.
2. Read the elements of the array.
3. Read the range L and R.
4. Initialize a variable count to 0.
5. Traverse the array and find the sum of all subarrays.
6. If the sum is in the range L to R, increment the count.
7. Print the count.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class CountSubarrays {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("N: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the array elements: ");
        for(int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print("L: ");
        int L = sc.nextInt();
        System.out.print("R: ");
        int R = sc.nextInt();
        int count = 0;
        for(int i = 0; i < n; i++) {
            int sum = 0;
            for(int j = i; j < n; j++) {
```

```
                sum += arr[j];
                if(sum >= L && sum <= R) {
                    count++;
                }
            }
        }
        System.out.print("Output :");
        System.out.println(count);
    }
}
```

Output:

```
N: 3
Enter the array elements: 1 4 6
L: 3
R: 8
Output :3
```

---

4. Self Crossing

Problem Statement :

You are given an array of integers distance.

You start at the point (0, 0) on an X-Y plane, and you move distance[0] meters to the north, then distance[1] meters to the west, distance[2] meters to the south, distanc other words, after each move, your direction changes counter-clockwise.

Return true if your path crosses itself or false if it does not.

Analysis:

1. Read the array of integers.
2. Initialize a set to store the coordinates.
3. Initialize the coordinates (0, 0).
4. Traverse the array and update the coordinates.
5. Check if the coordinates are already present in the set.
6. If present, return true.
7. Else, add the coordinates to the set.
8. Return false.

Code:

```
package com.self_practice;

import java.util.Scanner;

public class SelfCrossing {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] distance = new int[4];
        System.out.print("Enter the array elements: ");
        for(int i = 0; i < 4; i++) {
            distance[i] = sc.nextInt();
        }
        System.out.print("Output :");
        System.out.println(isSelfCrossing(distance));
    }
```

```
    public static boolean isSelfCrossing(int[] distance) {
        int x = 0, y = 0;
        int n = distance.length;
        int[][] dirs = {{0, 1}, {-1, 0}, {0, -1}, {1, 0}};
        int dir = 0;
        for (int i = 0; i < n; i++) {
            int[] d = dirs[dir];
            x += d[0] * distance[i];
            y += d[1] * distance[i];
            if (i >= 3 && (distance[i] >= distance[i - 2] && distance[i - 1] <= distance[i - 3] ||
                    i >= 4 && distance[i - 1] == distance[i - 3] && distance[i] + distance[i - 4] >= distance[i - 2])) {
                return true;
            }
            dir = (dir + 1) % 4;
        }
        return false;
    }
}
```

Output:

```
Enter the array elements: 2 1 1 2
Output :true
```

---

5. Maximum Score

Problem statement :

Tom is playing a board game in which two lists of distinct numbers ‘A’ and ‘B’ arranged in a non-descending order are given. The game has certain rules and the player given list and the score is the sum of unique picked numbers. The rules are:

1. Choose any list ‘A’ or ‘B’.
2. Traverse from left to right.
3. After picking a number, if the picked number is present in both the arrays, you are allowed to traverse to the other array.

You are given arrays,’A’ and ‘B’ of size ‘N’ and ‘M’ respectively. Your task is to find the maximum score Tom can achieve.

Since the answer can be very large, print answer % (10^9 + 7).

For Example:1 If the given array are ‘A’ = [1,3,5,7,9] and ‘B’ = [3,5,100]”.The maximum score can be achieved is 109[1+3+5+100].

Analysis:

1. Read the size of the arrays.
2. Read the elements of the arrays.
3. Initialize two variables, sum and result to 0.
4. Traverse the arrays and find the maximum sum.
5. Print the result % (10^9 + 7).

Code:

```
package com.self_practice;

import java.util.Scanner;

public class MaximumScore {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        System.out.print("N: ");
        int n = sc.nextInt();
        int[] A = new int[n];
        System.out.print("Enter the array elements: ");
        for(int i = 0; i < n; i++) {
            A[i] = sc.nextInt();
        }
        System.out.print("M: ");
        int m = sc.nextInt();
        int[] B = new int[m];
        System.out.print("Enter the array elements: ");
        for(int i = 0; i < m; i++) {
            B[i] = sc.nextInt();
        }
        System.out.print("Output :");
        System.out.println(maximumScore(A, B));
    }

    public static int maximumScore(int[] A, int[] B) {
        int mod = 1000000007;
        int i = 0, j = 0;
        long sum = 0, result = 0;
        while(i < A.length && j < B.length) {
            if(A[i] < B[j]) {
                sum += A[i];
                i++;
            } else if(A[i] > B[j]) {
                sum += B[j];
                j++;
            } else {
                sum += A[i];
                result = (result + sum) % mod;
                sum = 0;
                i++;
                j++;
            }
        }
        while(i < A.length) {
            sum += A[i];
            i++;
        }
        while(j < B.length) {
            sum += B[j];
            j++;
        }
        result = (result + sum) % mod;
        return (int) result;
    }
}
```

Output:

```
N: 5
Enter the array elements: 1 3 5 7 9
M: 3
Enter the array elements: 3 5 100
Output :109
```

---

6. Minimize OR of Remaining Elements Using Operations

Problem Statement :

You are given a 0-indexed integer arraynums and an integer k.

In one operation, you can pick any index i of nums such that 0 <= i < nums.length - 1 and replace nums[i] and nums[i + 1] with a single occurrence of nums[i] & nums[i + operator.

Return the minimum possible value of the bitwise OR of the remaining elements of nums after applying at most k operations.

Analysis:

1. Read the size of the array and the value of k.
2. Read the elements of the array.
3. Initialize a variable result to 0.
4. Traverse the array and find the minimum possible value.
5. Print the result.

Code:

```
package com.self_practice;

import java.util.Scanner;

public class MinimizeOR {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("N: ");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.print("Enter the array elements: ");
        for(int i = 0; i < n; i++) {
            nums[i] = sc.nextInt();
        }
        System.out.print("K: ");
        int k = sc.nextInt();
        System.out.print("Output :");
        System.out.println(minimizeOR(nums, k));
    }

    public static int minimizeOR(int[] nums, int k) {
        int n = nums.length;
        for (int i = 0; i < n - 1; i++) {
            nums[i] = nums[i] & nums[i+1];
            nums[i+1] = 0;
            k--;
            if (k<=0)
                break;
        }
        int result=0;
        for (int i: nums){
            if (i != 0)
                result++;
        }
        return result;
    }
}
```

Output:

```
N: 5
Enter the array elements: 3 5 3 2 7
K: 2
Output : 3
```

7. Cake Distribution Problem

Problem Statement :

Geek is organizing a birthday party, so his friends brought a cake for him. The cake consists of N chunks, whose individual sweetness is represented by the sweetness a
Geek cuts the cake into K + 1 pieces to distribute among his K friends. One piece he took for himself. Each piece consists of some consecutive chunks. He is very kind, s
sweetness for him. You need to find the maximum sweetness that the Geek can get if he distributes the cake optimally.

Analysis:

1. Read the size of the sweetness array and the value of k.
2. Read the elements of the sweetness array.
3. Initialize a variable max to 0.
4. Traverse the array and find the maximum sweetness.
5. Print the result.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class CakeDistribution {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("N: ");
        int n = sc.nextInt();
        int[] sweetness = new int[n];
        System.out.print("Enter the sweetness array: ");
        for(int i = 0; i < n; i++) {
            sweetness[i] = sc.nextInt();
        }
        System.out.print("K: ");
        int k = sc.nextInt();
        System.out.print("Output :");
        System.out.println(cakeDistribution(sweetness, k));
    }

    public static int cakeDistribution(int[] sweetness, int k) {
        int n = sweetness.length;
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += sweetness[i];
        }
        int start = 0;
        int end = sum;
        int max = 0;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            int count = 0;
            int temp = 0;
            for (int i = 0; i < n; i++) {
                temp += sweetness[i];
                if (temp >= mid) {
                    count++;
                    temp = 0;
                }
            }
            if (count >= k + 1) {
                max = Math.max(max, mid);
                start = mid + 1;
            } else {
```

```
                end = mid - 1;
            }
        }
        return max;
    }
}
```

Output:

```
N: 5
Enter the sweetness array: 6 3 2 8 7 5
K: 2
Output : 9
```

8. Minimize Manhattan Distances

Problem Statement :

You are given a array points representing integer coordinates of some points on a 2D plane, where points[i] = [xi, yi]. The distance between two points is defined as thei
minimum possible value for maximum distance between any two points by removing exactly one point.

Analysis:

1. Read the size of the array.
2. Read the elements of the array.
3. Initialize a variable max to 0.
4. Traverse the array and find the minimum possible value.
5. Print the result.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class MinimizeManhattanDistances {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("N: ");
        int n = sc.nextInt();
        int[][] points = new int[n][2];
        System.out.print("Enter the array elements: ");
        for(int i = 0; i < n; i++) {
            points[i][0] = sc.nextInt();
            points[i][1] = sc.nextInt();
        }
        System.out.print("Output :");
        System.out.println(minimizeManhattanDistances(points));
    }

    public static int minimizeManhattanDistances(int[][] points) {
        int n = points.length;
        int max = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                int dist = Math.abs(points[i][0] - points[j][0]) + Math.abs(points[i][1] - points[j][1]);
                max = Math.max(max, dist);
            }
        }
        return max;
```

```
    }
}
```

9. Count Fertile Pyramids in a Land

Problem Statement :

A farmer has a rectangular grid of land with m rows and n columns that can be divided into unit cells. Each cell is either fertile (represented by a 1) or barren (represent considered barren.

A pyramidal plot of land can be defined as a set of cells with the following criteria:

1. The number of cells in the set has to be greater than 1 and all cells must be fertile.
2. The apex of a pyramid is the topmost cell of the pyramid. The height of a pyramid is the number of rows it covers. Let `(r, c)` be the apex of the pyramid, and its h cells `(i, j)` where `r <= i <= r + h - 1` and `c - (i - r) <= j <= c + (i - r)`.

An inverse pyramidal plot of land can be defined as a set of cells with similar criteria:

1. The number of cells in the set has to be greater than 1 and all cells must be fertile.
2. The apex of an inverse pyramid is the bottommost cell of the inverse pyramid. The height of an inverse pyramid is the number of rows it covers. Let `(r, c)` be the Then, the plot comprises of cells `(i, j)` where `r - h + 1 <= i <= r` and `c - (r - i) <= j <= c + (r - i)`. Some examples of valid and invalid pyramidal (anc below. Black cells indicate fertile cells.

Given a 0-indexed m x n binary matrix grid representing the farmland, return the total number of pyramidal and inverse pyramidal plots that can be found in grid.

Analysis:

1. Read the size of the matrix.
2. Read the elements of the matrix.
3. Initialize a variable count to 0.
4. Traverse the matrix and find the total number of pyramidal and inverse pyramidal plots.
5. Print the result.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class FertilePyramids {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("M: ");
        int m = sc.nextInt();
        System.out.print("N: ");
        int n = sc.nextInt();
        int[][] grid = new int[m][n];
        System.out.print("Enter the matrix elements: ");
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                grid[i][j] = sc.nextInt();
            }
        }
        System.out.print("Output :");
        System.out.println(countFertilePyramids(grid));
    }

    public static int countFertilePyramids(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;
        int count = 0;
        for (int i = 0; i < m; i++) {
```

```
            for (int j = 0; j < n; j++) {
                for (int h = 2; h <= m; h++) {
                    if (i + h - 1 < m && j - (h - 1) >= 0 && j + (h - 1) < n) {
                        boolean flag = true;
                        for (int k = 0; k < h; k++) {
                            if (grid[i + k][j - k] == 0 || grid[i + k][j + k] == 0) {
                                flag = false;
                                break;
                            }
                        }
                        if (flag) {
                            count++;
                        }
                    }
                    if (i - h + 1 >= 0 && j - (h - 1) >= 0 && j + (h - 1) < n) {
                        boolean flag = true;
                        for (int k = 0; k < h; k++) {
                            if (grid[i - k][j - k] == 0 || grid[i - k][j + k] == 0) {
                                flag = false;
                                break;
                            }
                        }
                        if (flag) {
                            count++;
                        }
                    }
                }
            }
        }
        return count;
    }
}
```

Output:

```
M: 2
N: 4
Enter the matrix elements: 0 1 1 0
1 1 1 1
Output :2
```

10. Minimum HP

Problem statement :

You are playing a board game where you die when your hp (health points) becomes 0 or negative. Now you start with some initial hp, and start moving in the board sta to reach the bottom right corner of the board by only moving right and down. Whenever you arrive on a cell in the board your hp may increase or decrease by the amo positive value it means your hp increases, if it has a negative value it means your hp decreases. You need to return the minimum initial health required so that you can

Analysis:

1. Read the size of the matrix.
2. Read the elements of the matrix.
3. Initialize a variable minHp to 1.
4. Traverse the matrix and find the minimum initial health required.
5. Print the result.

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class MinimumHP {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("M: ");
        int m = sc.nextInt();
        System.out.print("N: ");
        int n = sc.nextInt();
        int[][] dungeon = new int[m][n];
        System.out.print("Enter the matrix elements: ");
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                dungeon[i][j] = sc.nextInt();
            }
        }
        System.out.print("Output :");
        System.out.println(minimumHP(dungeon));
    }

    public static int minimumHP(int[][] dungeon) {
        int m = dungeon.length;
        int n = dungeon[0].length;
        int[][] dp = new int[m][n];
        dp[m - 1][n - 1] = Math.max(1, 1 - dungeon[m - 1][n - 1]);
        for (int i = m - 2; i >= 0; i--) {
            dp[i][n - 1] = Math.max(1, dp[i + 1][n - 1] - dungeon[i][n - 1]);
        }
        for (int i = n - 2; i >= 0; i--) {
            dp[m - 1][i] = Math.max(1, dp[m - 1][i + 1] - dungeon[m - 1][i]);
        }
        for (int i = m - 2; i >= 0; i--) {
            for (int j = n - 2; j >= 0; j--) {
                int right = Math.max(1, dp[i][j + 1] - dungeon[i][j]);
                int down = Math.max(1, dp[i + 1][j] - dungeon[i][j]);
                dp[i][j] = Math.min(right, down);
            }
        }
        return dp[0][0];
    }
}
```

Output:

```
M: 3
N: 3
Enter the matrix elements:
1 -3 4
-6 -7 2
15 20 -4
Output :3
```