Self Practice - Problem Solving - II

1. Count ways to reach the n'th stair

Problem Statement : There are n stairs, a person standing at the bottom wants to reach the top. The person can climb either 1 stair or 2 stairs at a time. Count the number of ways, the person can reach the top (order does matter).

Analysis

- Get the number of stairs
- Find the number of ways to reach the top
- Print the number of ways
- Use the recursive approach

Code:

```java
package com.self_practice;

import java.util.Scanner;

class CountWaysToReachNthStair {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of stairs: ");
        int n = sc.nextInt();
        System.out.println("Number of ways = " + countWays(n));
    }

    public static int countWays(int n) {
        if (n == 0 || n == 1) {
            return 1;
        }
        return countWays(n - 1) + countWays(n - 2);
    }
}
```

Output:

```
Enter the number of stairs: 4
Number of ways = 5
```

---

2. Largest prime factor

Problem statement : Given a number N, the task is to find the largest prime factor of that number.

Analysis

- Get the number
- Find the largest prime factor
- Print the largest prime factor
- Use the recursive approach

Code:

```java
package com.self_practice;

import java.util.Scanner;

class LargestPrimeFactor {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number: ");
        long n = sc.nextLong();
        System.out.println("Largest prime factor = " + largestPrimeFactor(n));
    }

    public static long largestPrimeFactor(long n) {
        long i;
        long max = -1;

        while (n % 2 == 0) {
            max = 2;
            n >>= 1;
        }

        for (i = 3; i <= Math.sqrt(n); i += 2) {
            while (n % i == 0) {
                max = i;
                n = n / i;
            }
        }

        if (n > 2) {
            max = n;
        }

        return max;
    }
}
```

Output:

```
Enter the number: 15
Largest prime factor = 5
```

3. Mirror Upper triangle star pattern

Problem Statement: You are tasked with creating a Java program to print a mirror upper star triangle pattern. Given an integer N, the program should print a pattern with N rows where each row contains a mirrored upper triangle of stars.

Analysis

- Get the number of rows
- Print the mirror upper triangle star pattern
- Use the recursive approach

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class MirrorUpperTriangleStarPattern {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int n = sc.nextInt();
        printPattern(n);
    }
    private static void printPattern(int n){
        for(int i=0; i<n; i++){
            for(int j=0;j<n-i;j++){
                System.out.print(" ");
            }
            for(int j=0;j<=i;j++){
                System.out.print("* ");
            }
            System.out.println();
        }
        for(int i=n-2; i>=0; i--){
            for(int j=0;j<n-i;j++){
                System.out.print(" ");
            }
            for(int j=0;j<=i;j++){
                System.out.print("* ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
Enter any number:
5
     *
    * *
    * * *
   * * * *
  * * * * *
   * * * *
    * * *
     * *
      *
```

---

4. Occurrences of given digit

Problem statement: Rohan needs to find the number of occurrence of a digit in a given number .The input may lie within the range of integer. If the digit does not occur in the input it should print 0 else the count of digits.

Analysis

- Get the number and digit
- Find the number of occurrences of the digit
- Print the number of occurrences
- Use the recursive approach

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class OccurrencesOfGivenDigit {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number: ");
        int n = sc.nextInt();
        System.out.println("Enter the digit: ");
        int d = sc.nextInt();
        System.out.println("Number of occurrences = " + countOccurrences(n, d));
    }

    public static int countOccurrences(int n, int d) {
        int count = 0;
        while (n > 0) {
            if (n % 10 == d) {
                count++;
            }
            n = n / 10;
        }
        return count;
    }
}
```

Output:

```
Enter the number:
123245
Enter the digit:
2
Number of occurrences = 2
```

---

5. Single Number III

Problem Statement: Given an integer array nums, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once. You can return the answer in any order.You must write an algorithm that runs in linear runtime complexity and uses only constant extra space.

Analysis

- Get the array
- Find the two elements that appear only once
- Print the two elements

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class SingleNumberIII {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        int[] result = singleNumber(arr);
        System.out.println("The two elements that appear only once: ");
        for (int i = 0; i < result.length; i++) {
            System.out.print(result[i] + " ");
        }
    }

    public static int[] singleNumber(int[] nums) {
        int diff = 0;
        for (int num : nums) {
            diff ^= num;
        }
        diff &= -diff;
        int[] result = new int[2];
        for (int num : nums) {
            if ((num & diff) == 0) {
                result[0] ^= num;
            } else {
                result[1] ^= num;
            }
        }
        return result;
    }
}
```

Output:

```
Enter the number of elements:
6
Enter the elements:
1 2 1 2 3 5
The two elements that appear only once:
5 3
```

---

6. Find Xor-Beauty of Array

Problem Statement: You are given a 0-indexed integer array nums.The effective value of three indices i, j, and k is defined as ((nums[i] | nums[j]) & nums[k]).The xor-beauty of the array is the XORing of the effective values of all the possible triplets of indices (i, j, k) where 0 <= i, j, k < n.Return the xor-beauty of nums.

Note that:

- val1 | val2 is bitwise OR of val1 and val2.
- val1 & val2 is bitwise AND of val1 and val2.

Analysis

- Get the array
- Find the xor-beauty of the array
- Print the xor-beauty of the array

Code:

```
package com.self_practice;

import java.util.Scanner;

public class FindXorBeautyOfArray {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
```

```java
        }
        System.out.println("Xor-Beauty of the array = " + xorBeauty(arr));
    }

    public static int xorBeauty(int[] nums) {
        int n = nums.length;
        int ans = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    ans ^= ((nums[i] | nums[j]) & nums[k]);
                }
            }
        }
        return ans;
    }
}
```

Output:

```
Enter the number of elements:
10
Enter the elements:
15 45 20 2 34 35 5 44 32 30
Xor-Beauty of the array = 34
```

---

7. Number of Steps to Reduce a Number in Binary Representation to One

Problem Statements: Given the binary representation of an integer as a string s, return the number of steps to reduce it to 1 under the following rules: If the current number is even, you have to divide it by 2.If the current number is odd, you have to add 1 to it.It is guaranteed that you can always reach one for all test cases.

Analysis

- Get the binary representation of an integer
- Find the number of steps to reduce it to 1
- Print the number of steps

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class NumberOfStepsToReduceNumberInBinaryRepresentationToOne {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the binary representation of an integer: ");
        String s = sc.next();
        System.out.println("Number of steps = " + numSteps(s));
    }

    public static int numSteps(String s) {
        int steps = 0;
        while (s.length() > 1) {
            if (s.charAt(s.length() - 1) == '0') {
                s = s.substring(0, s.length() - 1);
            } else {
                s = addOne(s);
            }
            steps++;
        }
        return steps;
    }

    public static String addOne(String s) {
        StringBuilder sb = new StringBuilder();
        int carry = 1;
        for (int i = s.length() - 1; i >= 0; i--) {
            if (carry == 0) {
                sb.append(s.charAt(i));
            } else if (s.charAt(i) == '0') {
                sb.append('1');
                carry = 0;
            } else {
                sb.append('0');
            }
        }
        if (carry == 1) {
            sb.append('1');
        }
        return sb.reverse().toString();
    }
}
```

Output

```
Enter the binary representation of an integer:
1101
Number of steps = 6
```

8. Minimum Moves to Capture the Queen

Problem Statement: There is a 1-indexed 8 x 8 chessboard containing 3 pieces.

You are given 6 integers a, b, c, d, e, and f where:

- (a, b) denotes the position of the white rook.
- (c, d) denotes the position of the white bishop.
- (e, f) denotes the position of the black queen.

Given that you can only move the white pieces, return the minimum number of moves required to capture the black queen. Note that:

- Rooks can move any number of squares either vertically or horizontally, but cannot jump over other pieces.
- Bishops can move any number of squares diagonally, but cannot jump over other pieces.
- A rook or a bishop can capture the queen if it is located in a square that they can move to.
- The queen does not move.

Analysis

- Get the positions of the white rook, white bishop, and black queen
- Find the minimum number of moves required to capture the black queen
- Print the minimum number of moves

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class MinimumMovesToCaptureTheQueen {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the positions of the white rook, white bishop, and black queen: ");
        int a = sc.nextInt();
        int b = sc.nextInt();
        int c = sc.nextInt();
        int d = sc.nextInt();
        int e = sc.nextInt();
        int f = sc.nextInt();
        System.out.println("Minimum number of moves = " + minMoves(a, b, c, d, e, f));
    }

    public static int minMoves(int a, int b, int c, int d, int e, int f) {
        if ((a == e && b == f) || (Math.abs(a - e) == Math.abs(b - f)) || (a == e || b == f)) {
            return 1;
        }
        return 2;
    }
}
```

Output:

```
Enter the positions of the white rook, white bishop, and black queen:
4 4 5 5 6 6
Minimum number of moves = 1
```

---

9. Rectangular numbers

Problem statement : Print the pattern in such a way that the outer rectangle is of the number ‘N’ and the number goes on decreasing as we move inside the rectangles.

For example, if ‘N’ = 4, then pattern will be:

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

Analysis

- Get the number
- Print the pattern
- Use the looping approach

Code:

```java
package com.self_practice;

import java.util.Scanner;

public class RectangularNumbers {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number: ");
        int n = sc.nextInt();
        printPattern(n);
    }

    private static void printPattern(int n) {
        int len = 2 * n - 1;
        int[][] arr = new int[len][len];
        int startRow = 0;
        int endRow = len - 1;
        int startCol = 0;
        int endCol = len - 1;
        int num = n;
        while (startRow <= endRow && startCol <= endCol) {
            for (int i = startCol; i <= endCol; i++) {
                arr[startRow][i] = num;
                arr[endRow][i] = num;
            }
            for (int i = startRow; i <= endRow; i++) {
                arr[i][startCol] = num;
                arr[i][endCol] = num;
            }
            startRow++;
            endRow--;
            startCol++;
            endCol--;
            num--;
        }
        for (int i = 0; i < len; i++) {
            for (int j = 0; j < len; j++) {
                System.out.print(arr[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
Enter the number:
4
4 4 4 4 4 4 4
4 3 3 3 3 3 4
4 3 2 2 2 3 4
4 3 2 1 2 3 4
4 3 2 2 2 3 4
4 3 3 3 3 3 4
4 4 4 4 4 4 4
```

---

10. Stickler Thief

Problem Statement: Stickler the thief wants to loot money from a society having n houses in a single line. He is a weird person and follows a certain rule when looting the houses. According to the rule, he will never loot two consecutive houses. At the same time, he wants to maximize the amount he loots. The thief knows which house has what amount of money but is unable to come up with an optimal looting strategy. He asks for your help to find the maximum money he can get if he strictly follows the rule. ith house has a[i] amount of money present in it.

Analysis

- Get the number of houses and the amount of money in each house
- Find the maximum money the thief can get
- Print the maximum money the thief can get
- Use the dynamic programming approach

Code:

```
package com.self_practice;

import java.util.Scanner;

public class SticklerThief {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of houses: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the amount of money in each house: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Maximum money the thief can get = " + maxMoney(arr));
    }

    public static int maxMoney(int[] arr) {
        int n = arr.length;
        if (n == 0) {
            return 0;
        }
        if (n == 1) {
            return arr[0];
        }
        if (n == 2) {
            return Math.max(arr[0], arr[1]);
        }
        int[] dp = new int[n];
        dp[0] = arr[0];
        dp[1] = Math.max(arr[0], arr[1]);
        for (int i = 2; i < n; i++) {
            dp[i] = Math.max(dp[i - 1], arr[i] + dp[i - 2]);
        }
        return dp[n - 1];
    }
}
```

Output:

```
Enter the number of houses:
6
Enter the amount of money in each house:
2 7 9 3 1 5
Maximum money the thief can get = 17
```