Coding Contest - ||| Year

Total Marks: 100 | Duration: 180 | Total Questions: 5

Module: Arrays & Strings

Marks: 30 | Duration: 0

Question 1:

Marks: +15 -0

Maximum Number of Guests on a Cruise

Maximum Number of Guests on a Cruise Party

A party is organized on a cruise ship that lasts for **T** hours. For each hour, the number of guests entering and leaving the party is given.

You are provided with two integer arrays:

- `E[]` where `E[i]` represents the number of guests **entering** the party at the `i`-th hour (0-indexed).
- `L[]` where `L[i]` represents the number of guests **leaving** the party at the `i`-th hour.

Your task is to determine the **maximum number of guests present** on the cruise at any given time within the `T` hours duration.

Input Format:

The first line contains a single integer `T`, the total number of hours the party lasts.

The next `T` lines each contain an integer representing the entry count `E[i]` for each hour.

The next `T` lines each contain an integer representing the leaving count `L[i]` for each hour.

Output Format:

Print a single integer which is the maximum number of guests present on the cruise at any instant within the `T` hours.

Input

```
5
7
0
5
1
```

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

Explanation of Input:

- `T = 5` (total hours)
- `E = [7,[5][1][3]` (guests entering each hour)
- `L = [1, 2, 1, 3,[4]` (guests leaving each hour)

Output

```
8
```

Explanation

Hour	Entry (E[i])	Exit (L[i])	Guests Present
1	7	1	6
2	0	2	4
3	5	1	8 (Max)
4	1	3	6
5	3	4	5

Number of guests present at any instance is **8**.

This question was asked in the following below companies

1. Infosys
2. IBM
3. TCS Ninja

Test Cases:

#	Input	Expected Output
1	5 7 0 5	8

	1 3 1 2 1 3 4	
2	3 10 5 8 2 3 4	14
3	4 4 3 2 1 1 2 3 4	4

Solution:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int T = sc.nextInt();
        int[] E = new int[T];
        int[] L = new int[T];

        for (int i = 0; i < T; i++) {
            E[i] = sc.nextInt();
        }

        for (int i = 0; i < T; i++) {
            L[i] = sc.nextInt();
        }

        int currentGuests = 0;
        int maxGuests = 0;

        for (int i = 0; i < T; i++) {
            currentGuests += E[i] - L[i];
            maxGuests = Math.max(maxGuests, currentGuests);
        }

        System.out.println(maxGuests);
    }
}
```

Question 2:

Marks: +15 -0

Filter Event Ticket Number

Remove Specific Characters from Ticket Number

An event management company prints unique ticket numbers to direct visitors to their designated audience class. Given a ticket number `str1`, your task is to **remove specific characters or substrings** based on the following rules:

- Remove **all occurrences** of the substring **"EF"** (only when the characters 'E' and 'F' appear together in that order).
- Remove **all occurrences** of the digits **"56"** (only when digits '5' and '6' appear together in that order).
- Remove **all occurrences** of the character **"G"**.
- If 'E' and 'F' appear but **not consecutively in the order "EF"**, they should be **kept** intact.

The input will only contain uppercase alphabets `(A-Z)` and digits `(0-9)`. No spaces or special characters will be present.

Input Format:

A single line containing the string `str1`, representing the ticket number.

Output Format:

A single line containing the processed string after removing all specified characters and substrings.

Constraints:

`1 <= length of str1 <= 1000`

`str1` contains only uppercase letters `(A-Z)` and digits `(0-9)`.

No spaces or special characters in the input.

Input:

```
4523EF58G
```

Output:

```
452358
```

Explanation:

- **"EF"** is together → removed
- **"56"** is not present → no action
- **"G"** is removed

Input:

```
E12F35G58
```

Output:

```
E12F3558
```

Explanation:

- **"E"** and **"F"** are **not together** → kept
- **"56"** is not present → no action
- **"G"** is removed

This question was asked in the following below companies

1. HCLTech
2. Meta
3. IBM

Test Cases:

#	Input	Expected Output
1	`4523EF58G`	`452358`
2	`E12F35G58`	`E12F3558`
3	`A1B2C3D4EF56G`	`A1B2C3D4`

Solution:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String str1 = scanner.nextLine();

        // Remove occurrences of "EF"
        str1 = str1.replaceAll("EF", "");
        // Remove occurrences of "56"
        str1 = str1.replaceAll("56", "");
        // Remove occurrences of "G"
        str1 = str1.replaceAll("G", "");

        System.out.println(str1);
    }
}
```

Module: Recursion

Marks: 20 | Duration: 0

Question 1:

Marks: +20 -0

Parsing A Boolean Expression

Parsing A Boolean Expression

You are given a string `expression` which represents a boolean expression. The expression can be one of the following forms:

- `'t'` representing **true**.
- `'f'` representing **false**.
- `'!(subExpr)'` which evaluates to the logical NOT of the inner boolean expression `subExpr`.
- `'&(subExpr1, subExpr2, ..., subExprn)'` which evaluates to the logical AND of the inner boolean expressions `subExpr1, subExpr2, ..., subExprn`, where n ≥ 1.
- `'|(subExpr1, subExpr2, ..., subExprn)'` which evaluates to the logical OR of the inner boolean expressions `subExpr1, subExpr2, ..., subExprn`, where n ≥ 1.

Your task is to **evaluate the expression and return its boolean result**.

Input

- `expression`: a valid boolean expression string consisting only of characters `'t'`, `'f'`, `'!'`, `'&'`, `'|'`, `'('`, `')'`, and `','`.

Output

- A boolean `true` or `false` indicating the evaluation result of the expression.

Constraints

- `1 <= expression.length <= 2 * 10^4`
- The expression is guaranteed to be valid and follow the described rules.

Explanation of Evaluating the Boolean Expression

The expression consists of:

- Literals `'t'` (true) and `'f'` (false).
- Operators:
- `!` (NOT) applied to a single sub-expression.
- `&` (AND), applied to one or more sub-expressions.
- `|` (OR), applied to one or more sub-expressions.

Each operator is followed by parentheses enclosing its argument sub-expressions, which can themselves be literals or nested operator-expressions.

Examples

Example 1:

```
text
Input: expression = "&(|(f))"
Output: false

Explanation:
| (f) evaluates to f (false).
Then & (f) evaluates to f (false).
So the final result is false.
```

Example 2:

```
text
Input: expression = "|(f,f,f,t)"
Output: true

Explanation:
false OR false OR false OR true is true.
```

Example 3:

```
text
Input: expression = "!(&(f,t))"
Output: true

Explanation:
&(f,t) = false AND true = false.
! (f) = NOT false = true.
```


This question was asked in the below following companies

1. IBM
2. Deloite
3. Adobe

Test Cases:

#	Input	Expected Output
1	`&(|(f))`	`false`
2	`|(f,f,f,t)`	`true`
3	`|(&(t,t),!(|(f,f,f)))`	`true`

Solution:

```
import java.util.*;
public class Main{
     public static void main(String[] args) {
        BooleanExpressionEvaluator evaluator = new BooleanExpressionEvaluator();
        Scanner sc = new Scanner(System.in);
        System.out.println(evaluator.parseBoolExpr(sc.nextLine()));
        // Optional: You can run the test cases here or pass inputs to parseBoolExpr method.
    }
}
class BooleanExpressionEvaluator {

    public boolean parseBoolExpr(String expression) {
        Deque stack = new ArrayDeque<>();

       for (char ch : expression.toCharArray()) {
    if (ch == ',') {
        continue;
    } else if (ch == ')') {
        // evaluation code (same as yours)
        int trueCount = 0, falseCount = 0;
        while (!stack.isEmpty() && (stack.peek() == 't' || stack.peek() == 'f')) {
            char val = stack.pop();
            if (val == 't') trueCount++;
            else falseCount++;
        }
        stack.pop(); // pop '('

        char operator = stack.pop(); // operator must be on top now

        char result;
        switch (operator) {
            case '!':
                result = (falseCount > 0) ? 't' : 'f';
                break;
            case '&':
                result = (falseCount > 0) ? 'f' : 't';
                break;
            case '|':
                result = (trueCount > 0) ? 't' : 'f';
                break;
            default:
                throw new IllegalArgumentException("Invalid operator: " + operator);
        }
        stack.push(result);

    } else if (ch == '!' || ch == '&' || ch == '|') {
        stack.push(ch);

    } else if (ch == '(') {
        stack.push(ch);

    } else {
        // operands (t or f)
        stack.push(ch);
    }
}

        // Final result on stack top
        return stack.pop() == 't';
```

```
    }

    // Main method for simple manual testing

}
```

Module: DSA

Marks: 50 | Duration: 0

Question 1:

Marks: +30 -0

Chicks in a Zoo

Chicks in a Zoo

The zoo initially has **1 chick** on day 1. Each day, the following happens:

- Every chick alive at the start of the day gives birth to **2 new chicks**.
- Each chick lives for exactly **6 days**. On the 7th day of its life, it dies and is no longer counted.

Given an integer **N** representing the day number, determine the **total number of chicks alive** on the **N**th day.

Input Format:

A single integer `N` representing the day number.

Output Format:

A single integer representing the total number of chicks alive on day `N`.

Constraints:

- `1 <= N <= 100`

Input:

```
7
```

Output:

```
728
```

Example Walkthrough

Day 1:

Chicks Born: 1

Deaths: 0

Total Alive at End: 1

Day 2:

Chicks Born: 2

Deaths: 0

Total Alive at End: 3

Day 3:

Chicks Born: 6

Deaths: 0

Total Alive at End: 9

Day 4:

Chicks Born: 18

Deaths: 0

Total Alive at End: 27

Day 5:

Chicks Born: 54

Deaths: 0

Total Alive at End: 81

Day 6:

Chicks Born: 162

Deaths: 0

Total Alive at End: 243

Day 7:

Chicks Born: 486

Deaths: 1 (the original chick from Day 1 dies after living 6 days)

Total Alive at End: 728

Input:

```
2
```

Output:

```
3
```

Explanation:

- Day 1 → **1 chick**.
- Day 2 → The original chick gives birth to 2 chicks. Total = `1 + 2 = 3`.

Example 2:

Input:

```
3
```

Output:

```
9
```

Explanation:

- Day 1 → **1 chick**.
- Day 2 → `1` original + `2` new = `3` chicks.
- Day 3 → Each of the `3` chicks gives birth to `2` chicks:
- `3 + (3 * 2) = 9` chicks total.

Test Cases:

#	Input	Expected Output
1	`2`	`3`
2	`3`	`9`
3	`4`	`27`

Solution:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        sc.close();

        System.out.println(totalChicks(N));
    }

    public static long totalChicks(int N) {
        if (N == 1) return 1;

        long[] chicks = new long[N + 1];
```

```
        chicks[1] = 1;

        for (int day = 2; day <= N; day++) {
            long bornToday = chicks[day - 1] * 2;
            long dyingToday = (day > 6) ? chicks[day - 6] : 0;
            chicks[day] = chicks[day - 1] + bornToday - dyingToday;
        }

        return chicks[N];
    }
}
```

Question 2:

Marks: +20 -0

Cookie Distribution

Assign Cookies to Maximize Content Children

You are given two integer arrays:

- `greed`, where `greed[i]` represents the **greed factor** of the *i*-th child.
- `cookies`, where `cookies[j]` represents the **size** of the *j*-th cookie.

Each child will be **content** if they receive a cookie of size **greater than or equal to** their greed factor.

Your task is to determine the **maximum number of children** that can be made content by distributing the cookies optimally.

Input

- An integer array `greed` of length `n` (1 ≤ n ≤ 10^4), representing the greed factors of the children.
- An integer array `cookies` of length `m` (1 ≤ m ≤ 10^4), representing the sizes of the cookies.

Each element in `greed` and `cookies` satisfies:

```
1 ≤ greed[i], cookies[j] ≤ 10^5
```

Output

- Return an integer representing the maximum number of children that can be content.

Example 1:

Input:

```
greed = [1, 2, 3]
cookies = [1, 1]
```

Output:

```
1
```

Explanation:

Only one child (with greed factor `1`) can be content with a `1`-sized cookie.

Example 2:

Input:

```
greed = [1, 2]
cookies = [1, 2, 3]
```

Output:

```
2
```

Explanation:

Both children can be content because they receive cookies of size `1` and `2`, respectively.

Constraints:

- `1 ≤ n, m ≤ 10^4`
- `1 ≤ greed[i], cookies[j] ≤ 10^5`

Note:

The key idea is to **sort both arrays** and use a **greedy approach**, iterating through the children and assigning the smallest possible cookie that meets their greed factor.

Test Cases:

#	Input	Expected Output
1	3 1 2 3 2 1 1	1
2	2 1 2 3 1 2 3	2

3	4 1 2 4 5 5 1 2 2 4 6	4

Solution:

```
import java.util.Arrays;
import java.util.Scanner;

public class Main{
    public static int maxContentChildren(int[] greed, int[] cookies) {
        Arrays.sort(greed);
        Arrays.sort(cookies);
        int i = 0, j = 0, count = 0;

        while (i < greed.length && j < cookies.length) {
            if (cookies[j] >= greed[i]) {
                count++;
                i++;
            }
            j++;
        }
        return count;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] greed = new int[n];
        for (int i = 0; i < n; i++)
            greed[i] = sc.nextInt();

        int m = sc.nextInt();
        int[] cookies = new int[m];
        for (int i = 0; i < m; i++)
            cookies[i] = sc.nextInt();

        System.out.println(maxContentChildren(greed, cookies));
        sc.close();
    }
}
```