✅ Problem Name: Smart Traffic Light System

Problem Statement:

You are developing a basic simulation for a smart traffic light system at a four-way intersection. Based on the day of the week and the time of the day (morning/evening), the system decides **which road gets the green light first**.

The rules are:

- Use a `switch` statement for **day of the week** (`1` for Monday, `2` for Tuesday, ..., `7` for Sunday).
- Use `if-else` conditions inside the switch cases to determine:
  - If it's **Morning**, give priority to **Main Road**.
  - If it's **Evening**, give priority to **Side Road**.
- On **Sunday**, always give green light to **All Directions** regardless of time.

You are given:

- An integer `day` (`1-7`) representing the day of the week.
- A string `time` which is either `"Morning"` or `"Evening"`.

Input Format:

```
An integer day (1 ≤ day ≤ 7)
A string time ("Morning" or "Evening")
```

Output Format:

```
Print one of the following:
- "Main Road"
- "Side Road"
- "All Directions"
```

🧪 Sample Test Cases

Test Case 1

Input:

```
1
Morning
```

Output:

```
Main Road
```

Test Case 2

Input:

```
4
Evening
```

Output:

```
Side Road
```

Test Case 3

Input:

```
7
Morning
```

Output:

```
All Directions
```

Test Case 4

Input:

```
7
Evening
```

Output:

```
All Directions
```

✅ Java Solution:

```
import java.util.Scanner;

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

        int day = sc.nextInt();
        sc.nextLine(); // consume newline
        String time = sc.nextLine();

        switch(day) {
            case 7:
```

```
                System.out.println("All Directions");
                break;
            case 1: case 2: case 3: case 4: case 5: case 6:
                if (time.equals("Morning")) {
                    System.out.println("Main Road");
                } else if (time.equals("Evening")) {
                    System.out.println("Side Road");
                }
                break;
            default:
                System.out.println("Invalid Input");
        }
    }
}
```

✅ Problem Name: Mirror Number Validator

Problem Statement:

In a parallel universe, certain numbers are called "**Mirror Numbers**". A number is considered a **Mirror Number** if the **first digit** and the **last digit** are **the same**.

Your task is to:

1. Check if a given **positive integer** is a Mirror Number.
2. If it is, print `"Mirror Number"`.
3. If not, check:
    - If the number has **exactly 2 digits**, print `"Almost Mirror"`.
    - If it has more than 2 digits, print `"Not Mirror"`.
    - If the number is **only 1 digit**, print `"Single Digit"`.

Input Format:

`A single integer N (1 ≤ N ≤ 1000000)`

Output Format:

Print one of the following based on the number:

- `"Mirror Number"`
- `"Almost Mirror"`
- `"Not Mirror"`
- `"Single Digit"`

🧪 Sample Test Cases

Test Case 1

Input:

```
121
```

Output:

```
Mirror Number
```

Test Case 2

Input:

```
45
```

Output:

```
Almost Mirror
```

Test Case 3

Input:

```
98
```

Output:

```
Not Mirror
```

Test Case 4

Input:

```
7
```

Output:

```
Single Digit
```

✅ Java Solution:

```
import java.util.Scanner;

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

```
        int N = sc.nextInt();

        String str = String.valueOf(N);
        int length = str.length();

        if (length == 1) {
            System.out.println("Single Digit");
        } else {
            char first = str.charAt(0);
            char last = str.charAt(str.length() - 1);

            if (first == last) {
                System.out.println("Mirror Number");
            } else if (length == 2) {
                System.out.println("Almost Mirror");
            } else {
                System.out.println("Not Mirror");
            }
        }
    }
}
```

✅ Problem Name: Advanced Mirror Palindrome Checker

Problem Statement:

In a distant future, robots only accept special numbers to unlock their systems. A number is **valid** if it satisfies one of the following:

1. It is a **Mirror Number** (first digit == last digit).
2. It is a **Palindrome Number** (the number reads the same forwards and backwards).
3. If it is a **2-digit number**, and doesn't satisfy the above, print `"Weak Number"`.
4. If it is a **1-digit number**, print `"Trivial"`.
5. If none of the above conditions are satisfied, print `"Invalid Number"`.

Write a program to determine the **status** of a number based on the above rules.

Input Format:

`A single integer N (1 ≤ N ≤ 1000000)`

Output Format:

Print one of the following:

- `"Mirror Number"`
- `"Palindrome"`
- `"Weak Number"`
- `"Trivial"`

- `"Invalid Number"`

🧪 Sample Test Cases

Test Case 1

Input:

```
787
```

Output:

```
Palindrome
```

Test Case 2

Input:

```
34
```

Output:

```
Weak Number
```

Test Case 3

Input:

```
3453
```

Output:

```
Mirror Number
```

Test Case 4

Input:

```
7
```

Output:

```
Trivial
```

✅ Java Solution:

```
import java.util.Scanner;

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

        String str = String.valueOf(N);
        int length = str.length();

        if (length == 1) {
            System.out.println("Trivial");
        } else {
            String reversed = new StringBuilder(str).reverse().toString();
            boolean isPalindrome = str.equals(reversed);
            boolean isMirror = str.charAt(0) == str.charAt(str.length() - 1);

            if (isPalindrome) {
                System.out.println("Palindrome");
            } else if (isMirror) {
                System.out.println("Mirror Number");
            } else if (length == 2) {
                System.out.println("Weak Number");
            } else {
                System.out.println("Invalid Number");
            }
        }
    }
}
```

✅ Problem Name: Balanced Sum Finder

Problem Statement:

You're given an array of integers. A **Balanced Index** is an index `i` such that the **sum of elements to the left of `i`** is equal to the **sum of elements to the right of `i`**.

Write a program to:

- Find and print the **first balanced index** (0-based indexing).
- If no such index exists, print `"No Balance"`.

Note: The element at the index itself is **not** included in either sum.

Input Format:

An integer N (1 ≤ N ≤ 100000)
N space-separated integers (each between -10^4 to 10^4)

Output Format:

Print the index if a balanced index exists, else print "No Balance"

🧪 Sample Test Cases

Test Case 1

Input:

5
1 2 3 2 1

Output:

2

Test Case 2

Input:

4
2 0 0 2

Output:

1

Test Case 3

Input:

3
1 2 3

Output:

No Balance

Test Case 4

Input:

1

```
10
```

Output:

```
0
```

✅ Java Solution:

```
import java.util.*;

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

        int N = sc.nextInt();
        int[] arr = new int[N];

        int totalSum = 0;
        for (int i = 0; i < N; i++) {
            arr[i] = sc.nextInt();
            totalSum += arr[i];
        }

        int leftSum = 0;
        boolean found = false;

        for (int i = 0; i < N; i++) {
            int rightSum = totalSum - leftSum - arr[i];

            if (leftSum == rightSum) {
                System.out.println(i);
                found = true;
                break;
            }
            leftSum += arr[i];
        }

        if (!found) {
            System.out.println("No Balance");
        }
    }
}
```

Great! Here's a **bit harder HackerRank-style problem** focused on **strings**, combining character frequency analysis and a tricky condition to make it suitable for intermediate to advanced practice.

✅ Problem Name: Almost Unique String

Problem Statement:

A string is called **"Almost Unique"** if it satisfies **exactly one** of the following conditions:

1. **All characters are unique** (appear only once).
2. **Exactly one character repeats**, and the rest are unique.

Given a string `S`, your task is to:

- Check if it is an **"Almost Unique"** string.
- Print `"Almost Unique"` if it satisfies **exactly one** of the two conditions.
- Otherwise, print `"Not Unique"`.

Input Format:

```
A single string S (1 ≤ |S| ≤ 1000), containing only lowercase English
letters.
```

Output Format:

```
Print one of:
- "Almost Unique"
- "Not Unique"
```

🧪 Sample Test Cases

Test Case 1

Input:

```
abcde
```

Output:

```
Almost Unique
```

Test Case 2

Input:

```
aabcde
```

Output:

```
Almost Unique
```

Test Case 3

Input:

```
aabbcd
```

Output:

```
Not Unique
```

Test Case 4

Input:

```
aaaaa
```

Output:

```
Not Unique
```

✅ Java Solution:

```
import java.util.Scanner;

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

        int[] freq = new int[26];
        for (char c : s.toCharArray()) {
            freq[c - 'a']++;
        }

        int unique = 0, repeats = 0;
        for (int f : freq) {
            if (f == 1) unique++;
            else if (f > 1) repeats++;
        }

        if ((unique == s.length() && repeats == 0) || (unique == s.length() -
1 && repeats == 1)) {
            System.out.println("Almost Unique");
        } else {
            System.out.println("Not Unique");
        }
    }
}
```