Assignment 10 (Advance Programming)

Name – Paras Aggarwal

UID – 22BCS16963

118. Pascal's Triangle

Given an integer numRows, return the first numRows of Pascal's triangle.

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

Example 1:

Input: numRows = 5

Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Example 2:

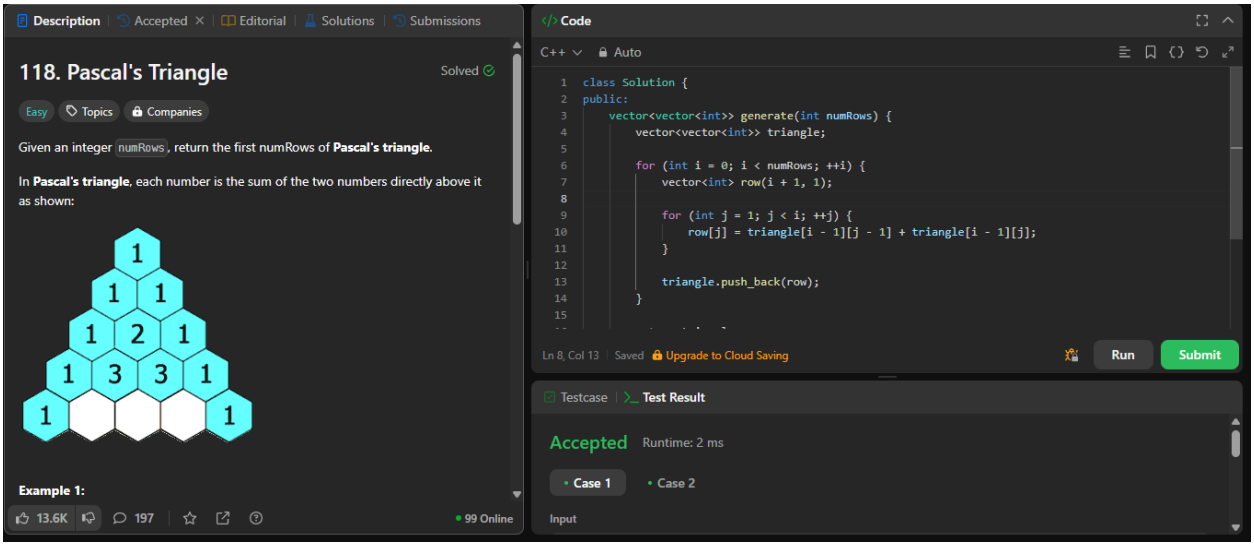
Input: numRows = 1

Output: [[1]]

Solution:

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> triangle;

        for (int i = 0; i < numRows; ++i) {
            vector<int> row(i + 1, 1);
```

```
            for (int j = 1; j < i; ++j) {
                row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
            }

            triangle.push_back(row);
        }
        return triangle;
    }
};
```



461. Hamming Distance

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y, return the Hamming distance between them.

Example 1:

Input: x = 1, y = 4

Output: 2

Explanation:

1 (0 0 0 1)

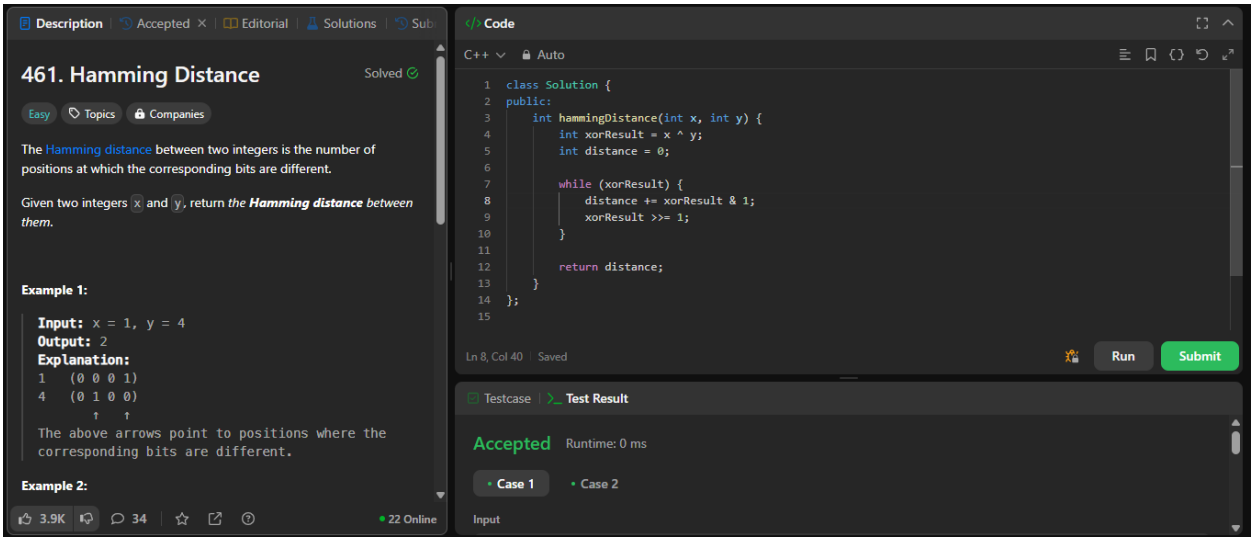
4 (0 1 0 0)

↑ ↑

The above arrows point to positions where the corresponding bits are different.

Solution:

```
class Solution {
public:
    int hammingDistance(int x, int y) {
        int xorResult = x ^ y;
        int distance = 0;
        while (xorResult) {
            distance += xorResult & 1;
            xorResult >>= 1;
        }
        return distance;
    }
};
```



621. Task Scheduler

You are given an array of CPU tasks, each labeled with a letter from A to Z, and a number n. Each CPU interval can be idle or allow the completion of one task. Tasks can be completed in any order, but there's a constraint: there has to be a gap of at least n intervals between two tasks with the same label.

Return the minimum number of CPU intervals required to complete all tasks.

Example 1:

Input: tasks = ["A","A","A","B","B","B"], n = 2

Output: 8

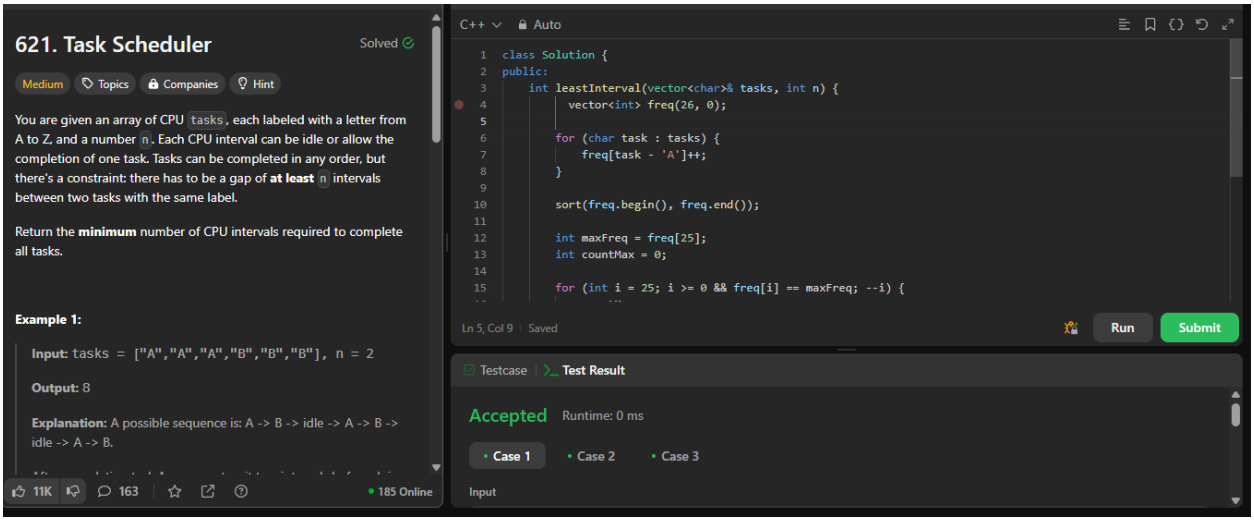
Explanation: A possible sequence is: A -> B -> idle -> A -> B -> idle -> A -> B.

After completing task A, you must wait two intervals before doing A again. The same applies to task B. In the 3rd interval, neither A nor B can be done, so you idle. By the 4th interval, you can do A again as 2 intervals have passed.

Solution:

```
class Solution {
public:
    int leastInterval(vector<char>& tasks, int n) {
        vector<int> freq(26, 0);

        for (char task : tasks) {
            freq[task - 'A']++;
        }

        sort(freq.begin(), freq.end());
```

```
        int maxFreq = freq[25];
        int countMax = 0;

        for (int i = 25; i >= 0 && freq[i] == maxFreq; --i) {
            countMax++;
        }

        int partCount = maxFreq - 1;
        int partLength = n + 1;
        int emptySlots = partCount * partLength + countMax;

        return max((int)tasks.size(), emptySlots);
    }
};
```



191. Number of 1 Bits

Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

Example 1:

Input: n = 11

Output: 3

Explanation:

The input binary string 1011 has a total of three set bits.

Example 2:

Input: n = 128

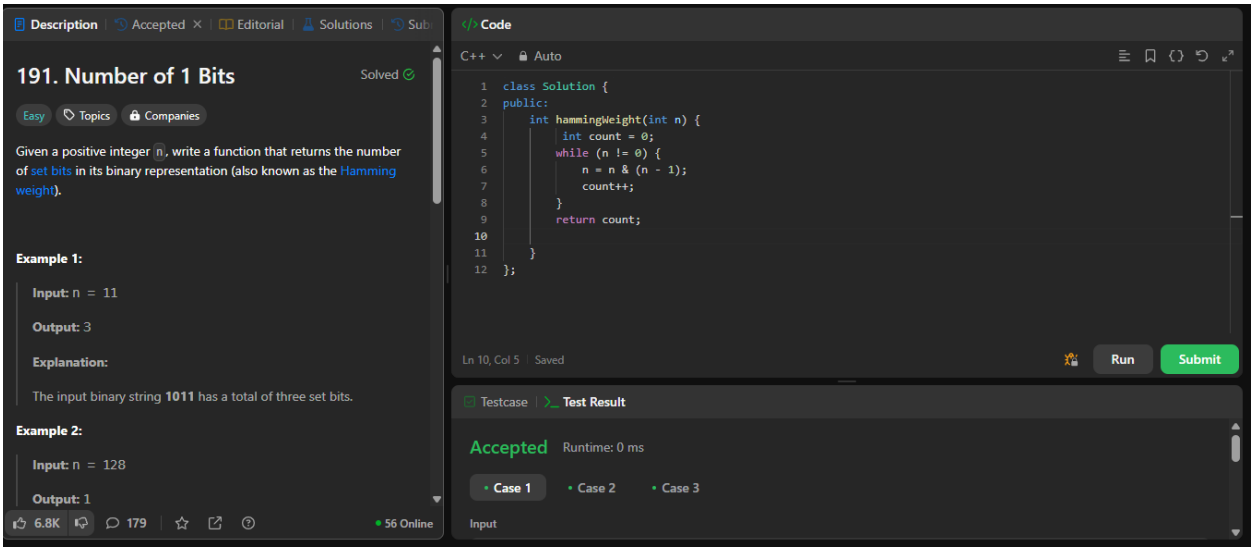
Output: 1

Explanation:

The input binary string 10000000 has a total of one set bit.

Solution:

```
class Solution {
public:
    int hammingWeight(int n) {
         int count = 0;
        while (n != 0) {
            n = n & (n - 1);
            count++;
        }
        return count;
};
```



42. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Solution:

```
class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        if (n == 0) return 0;
```

```
        vector<int> left_max(n, 0);
        vector<int> right_max(n, 0);

        left_max[0] = height[0];
        for (int i = 1; i < n; ++i) {
            left_max[i] = max(left_max[i-1], height[i]);
        }

        right_max[n-1] = height[n-1];
        for (int i = n-2; i >= 0; --i) {
            right_max[i] = max(right_max[i+1], height[i]);
        }

        int total_water = 0;
        for (int i = 0; i < n; ++i) {

            total_water += min(left_max[i], right_max[i]) - height[i];
        }

        return total_water;
    }
};
```

Description | Accepted | Editorial | Solutions

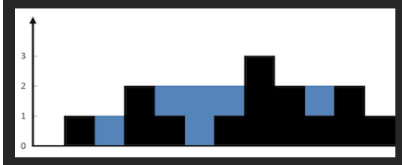
42. Trapping Rain Water

Solved

Hard | Topics | Companies

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:



Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black

33.8K | 377 | 473 Online

Code

C++ | Auto

```cpp
class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        if (n == 0) return 0;

        vector<int> left_max(n, 0);
        vector<int> right_max(n, 0);

        left_max[0] = height[0];
        for (int i = 1; i < n; ++i) {
            left_max[i] = max(left_max[i-1], height[i]);
        }

        right_max[n-1] = height[n-1];
```

Ln 29, Col 1 | Saved

Run | Submit

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input