

Assignment 10 (Advance Programming)

Name – Ankit Kharb

UID – 22BCS16964

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); // initialize row with 1s
```

```

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

        triangle.push_back(row);
    }

    return triangle;
}
};

```

The screenshot displays a coding platform interface for the problem "118. Pascal's Triangle". The problem description states: "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:". Below the text is a diagram of Pascal's triangle with 5 rows. The numbers are 1, 1, 1, 2, 1, 3, 3, 1, 1, 4, 6, 4, 1, 1, 5, 10, 10, 5, 1. The numbers 2, 3, 3, 4, 6, 4 are highlighted in orange, and the number 10 is highlighted in green. The code editor shows a C++ solution using a vector of vectors to store the triangle. The test case section shows a single case with numRows = 5, and the output is 5. The interface also includes a "Problem List" tab, a "Solved" status indicator, and a "Premium" badge.

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 xorVal = x ^ y;
        int count = 0;
        while (xorVal) {
            count += xorVal & 1; // check last bit
            xorVal >>= 1;        // right shift
        }
        return count;
    }
};
```

The screenshot shows the LeetCode interface for problem 461, 'Hamming Distance'. The left sidebar contains the problem description, which states that the Hamming distance is the number of positions where corresponding bits are different. It provides an example with x=1 and y=4, showing their binary representations (0001 and 0100) and highlighting the two differing bits with arrows. The right pane shows the C++ code for the solution, which uses XOR to find differing bits and a loop to count them. The bottom section shows a test case with x=1.

461. Hamming Distance Solved ✓

Easy Topics Companies

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.

```
1 class Solution {
2 public:
3     int hammingDistance(int x, int y) {
4         int xorVal = x ^ y;
5         int count = 0;
6         while (xorVal) {
7             count += xorVal & 1; // check last bit
8             xorVal >>= 1;        // right shift
9         }
10        return count;
11    }
```

Ln 1, Col 1 Saved Run Submit

Testcase Test Result

Case 1 Case 2 +

x =

1

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);
```

```
        // Count frequency of each task
```

```
        for (char task : tasks) {
```

```
            freq[task - 'A']++;
```

```
        }
```

```
        // Sort frequencies to get the task with maximum frequency at the end
```

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

```
        int maxFreq = freq[25]; // Highest frequency
```

```
        int maxFreqCount = 1;
```

```
        // Count how many tasks have the same maximum frequency
```

```

for (int i = 24; i >= 0; i--) {
    if (freq[i] == maxFreq) {
        maxFreqCount++;
    } else {
        break;
    }
}

// Calculate the minimum time using the greedy formula
int partCount = maxFreq - 1;
int partLength = n - (maxFreqCount - 1);
int emptySlots = partCount * partLength;
int availableTasks = tasks.size() - (maxFreq * maxFreqCount);
int idles = max(0, emptySlots - availableTasks);

return tasks.size() + idles;
}
};

```

621. Task Scheduler Solved

Medium Topics Companies Hint

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 -> B -> A.

Code Editor:

```

1 class Solution {
2 public:
3     int leastInterval(vector<char>& tasks, int n) {
4         vector<int> freq(26, 0);
5
6         // Count frequency of each task
7         for (char task : tasks) {
8             freq[task - 'A']++;
9         }
10
11         // Sort frequencies to get the task with maximum frequency at the end

```

Ln 1, Col 1 | Saved Run Submit

Testcase Test Result

Case 1 Case 2 Case 3 +

tasks =

["A","A","A","B","B","B"]

11K 163 159 Online

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;
    };
};
```

The screenshot shows a coding platform interface. On the left, the problem description for '191. Number of 1 Bits' is visible, including an 'Easy' difficulty tag and an example where input n=11 results in output 3. On the right, the C++ code for the solution is displayed in a code editor. The code defines a function 'hammingWeight' that counts the number of set bits in an integer n by repeatedly removing the rightmost set bit. Below the code editor, the 'Test Result' section shows 'Accepted' status with a runtime of 0 ms. At the bottom, there are buttons for 'Run' and 'Submit'.

```

1 public:
2     int hammingWeight(int n) {
3         int count = 0;
4         while (n != 0) {
5             n = n & (n - 1); // Remove the rightmost set bit
6             count++;
7         }
8         return count;
9     }
10 }
11 };
12

```

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;
```

```
    }
```

```
};
```


Problem List

Accepted

Editorial

Solutions

Solved

42. Trapping Rain Water

HardTopicsCompanies

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]

33.8K377512 Online

<https://leetcode.com/problems/trapping-rain-water/>

Code

```
18
19
20
21
22
23
24
25
26
27
28
    right_max = height[right];
    } else {
        total_water += right_max - height[right];
    }
    right--;
    }
    return total_water;
};
```

Ln 28, Col 1SavedRunSubmit

Testcase

Test Result

AcceptedRuntime: 0 ms

Case 1Case 2

Input

height =