



Experiment 4

Student Name: Nitesh Bamber

Branch: BE-CSE

Semester: 6th

Subject Name: Advanced Programming - II

UID: 22BCS13791

Section/Group: 602/A

Date of Performance: 21-02-25

Subject Code: 22CSP-351

1763. Longest Nice Substring

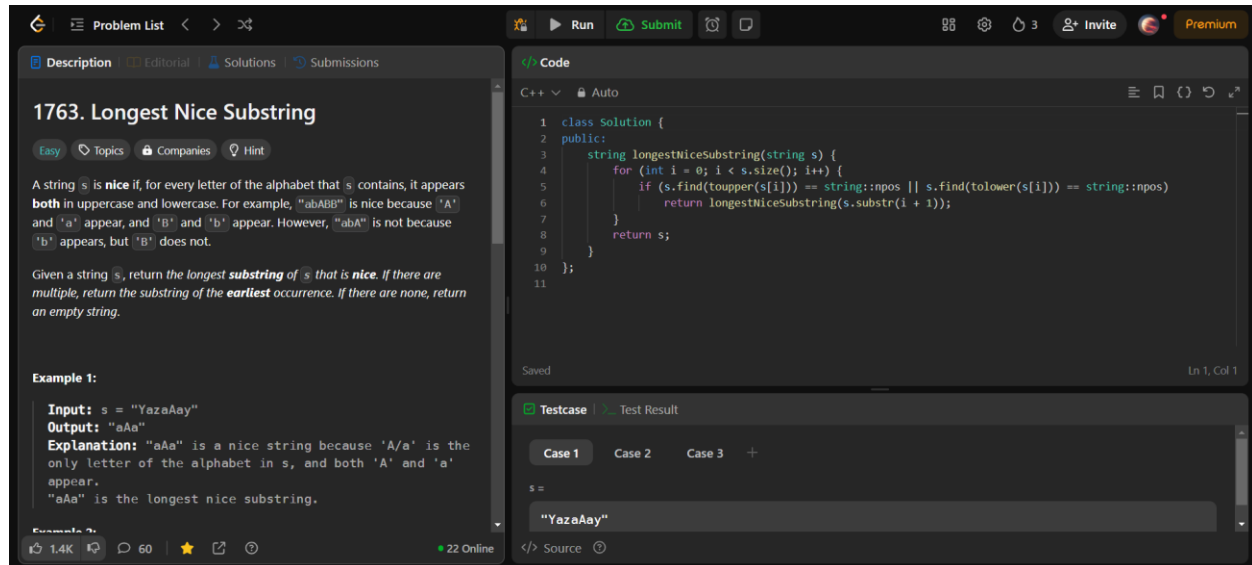
Aim- A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

Given a string s , return the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string.

CODE:-

```
class Solution {
public:
    string longestNiceSubstring(string s) {
        for (int i = 0; i < s.size(); i++) {
            if (s.find(toupper(s[i])) == string::npos || s.find(tolower(s[i])) == string::npos)
                return longestNiceSubstring(s.substr(i + 1));
        }
        return s;
    }
}
```

```
};
```



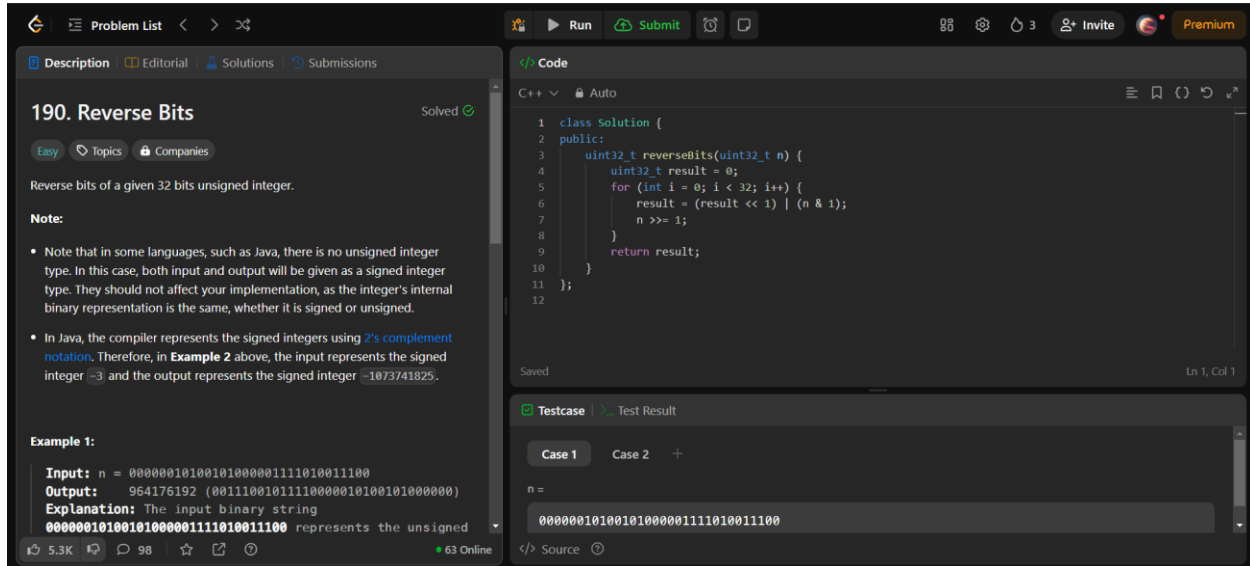
The screenshot shows a coding platform interface. On the left, the problem description for "1763. Longest Nice Substring" is visible. It defines a "nice" string as one where every letter appears in both uppercase and lowercase. The example shows "aAa" as a nice substring of "YazeAay". On the right, a C++ solution is provided, which uses a recursive function to find the longest nice substring by checking each character and its counterpart in the alphabet.

190. Reverse Bits

Aim- Reverse bits of a given 32 bits unsigned integer.

CODE:-

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t result = 0;
        for (int i = 0; i < 32; i++) {
            result = (result << 1) | (n & 1);
            n >>= 1;
        }
        return result;
    }
};
```



190. Reverse Bits Solved

Reverse bits of a given 32 bits unsigned integer.

Note:

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 2** above, the input represents the signed integer -3 and the output represents the signed integer -1073741825 .

Example 1:

Input: $n = 000000100101000001111010011100$
Output: 964176192 (0011100101110000010100101000000)
Explanation: The input binary string 000000100101000001111010011100 represents the unsigned integer 4381. To reverse the bits, we have to convert it to its binary representation, which is 000000100101000001111010011100. The output is the binary representation of the reversed input, which is 0011100101110000010100101000000.

```

1 class Solution {
2 public:
3     uint32_t reverseBits(uint32_t n) {
4         uint32_t result = 0;
5         for (int i = 0; i < 32; i++) {
6             result = (result << 1) | (n & 1);
7             n >>= 1;
8         }
9         return result;
10    }
11 };
12

```

Testcase | Test Result

Case 1 Case 2 +

n = 000000100101000001111010011100

191. Number of 1 Bits

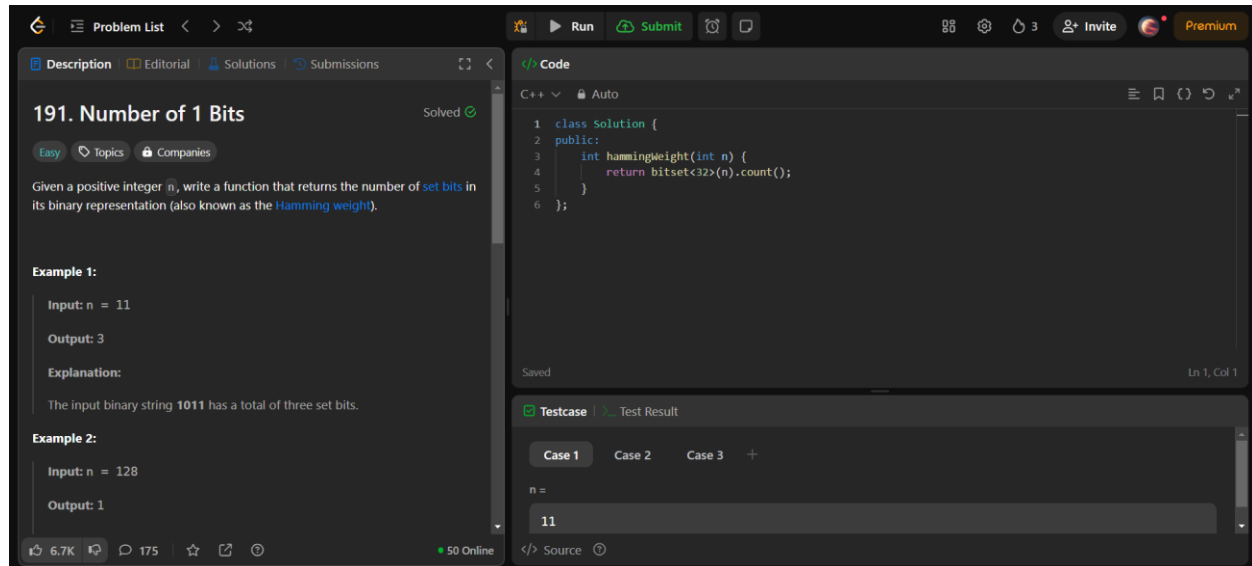
Aim- Given a positive integer n , write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

CODE:-

```

class Solution {
public:
    int hammingWeight(int n) {
        return bitset<32>(n).count();
    }
};

```



The screenshot shows a coding platform interface. On the left, the problem description for '191. Number of 1 Bits' is visible, marked as 'Solved'. It asks for a function to return the number of set bits in a positive integer's binary representation. Example 1 shows input n=11 and output 3. Example 2 shows input n=128 and output 1. On the right, the C++ code is displayed in a dark-themed editor. The code defines a class Solution with a public method hammingWeight that uses bitset to count the number of set bits. Below the code, the 'Testcase' section shows Case 1 with input n=11.

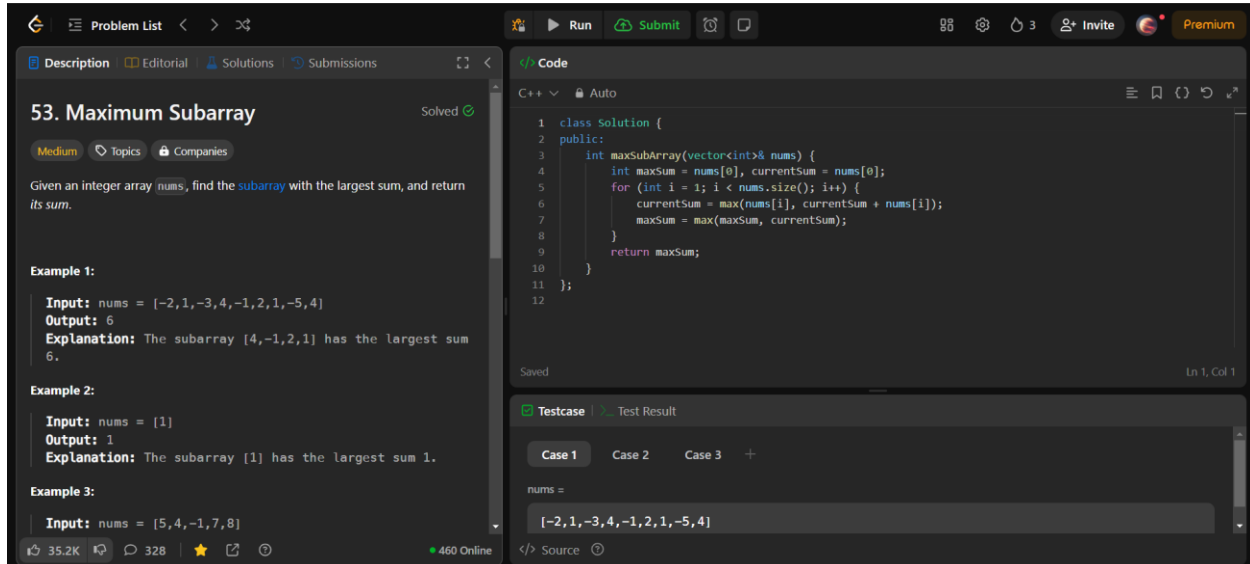
```
1 class Solution {
2 public:
3     int hammingWeight(int n) {
4         return bitset<32>(n).count();
5     }
6 };
```

53. Maximum Subarray

Aim- Given an integer array *nums*, find the subarray with the largest sum, and return its sum.

CODE:-

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = nums[0], currentSum = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }
        return maxSum;
    }
};
```



53. Maximum Subarray Solved

Medium Topics Companies

Given an integer array `nums`, find the **subarray** with the largest sum, and return its sum.

Example 1:
Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`
Output: 6
Explanation: The subarray `[4,-1,2,1]` has the largest sum 6.

Example 2:
Input: `nums = [1]`
Output: 1
Explanation: The subarray `[1]` has the largest sum 1.

Example 3:
Input: `nums = [5,4,-1,7,8]`

35.2K 328 460 Online

```

1 class Solution {
2 public:
3     int maxSubArray(vector<int>& nums) {
4         int maxSum = nums[0], currentSum = nums[0];
5         for (int i = 1; i < nums.size(); i++) {
6             currentSum = max(nums[i], currentSum + nums[i]);
7             maxSum = max(maxSum, currentSum);
8         }
9         return maxSum;
10    }
11 };
12

```

Testcase | Test Result

Case 1 Case 2 Case 3 +

nums =

[-2,1,-3,4,-1,2,1,-5,4]

Source

240. Search a 2D Matrix

Aim- Write an efficient algorithm that searches for a value target in an $m \times n$ integer matrix matrix. This matrix has the following properties:

- *Integers in each row are sorted in ascending from left to right.*
- *Integers in each column are sorted in ascending from top to bottom.*

CODE:-

```

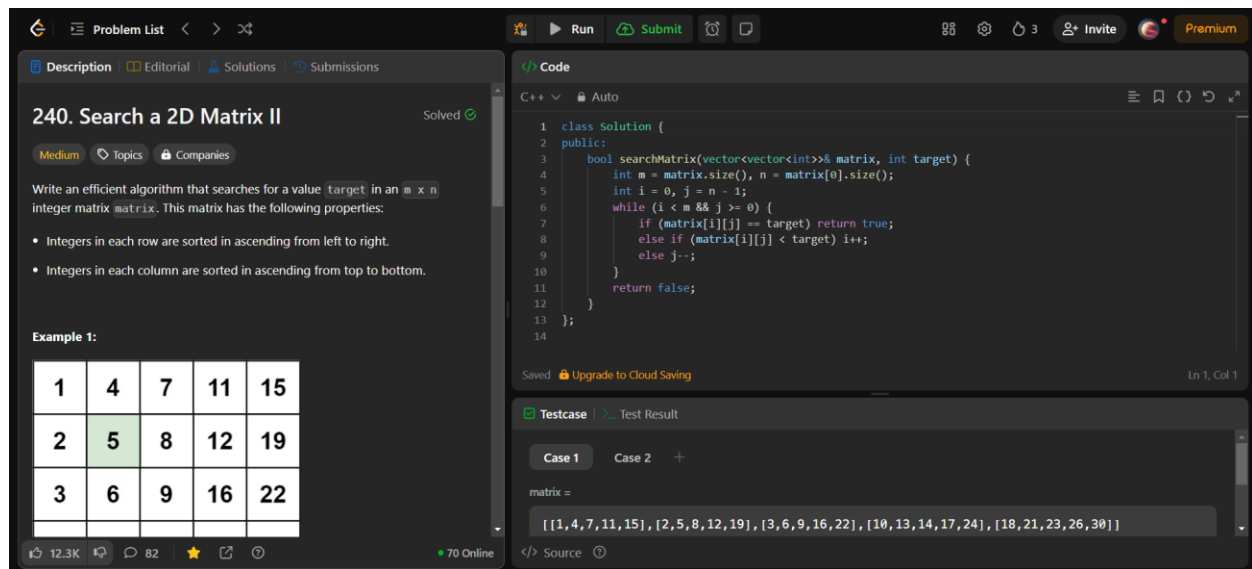
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size(), n = matrix[0].size();
        int i = 0, j = n - 1;
        while (i < m && j >= 0) {
            if (matrix[i][j] == target) return true;
            else if (matrix[i][j] < target) i++;
        }
    }
};

```

```

        else j--;
    }
    return false;
}
};

```



240. Search a 2D Matrix II Solved

Medium Topics Companies

Write an efficient algorithm that searches for a value `target` in an `m x n` integer matrix `matrix`. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

Example 1:

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22

Code

```

1 class Solution {
2 public:
3     bool searchMatrix(vector<vector<int>>& matrix, int target) {
4         int m = matrix.size(), n = matrix[0].size();
5         int i = 0, j = n - 1;
6         while (i < m && j >= 0) {
7             if (matrix[i][j] == target) return true;
8             else if (matrix[i][j] < target) i++;
9             else j--;
10        }
11        return false;
12    }
13 };
14

```

Testcase

Case 1 Case 2 +

matrix =

```

[[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]

```

372. Super Pow

Aim- Your task is to calculate $a^b \bmod 1337$ where a is a positive integer and b is an extremely large positive integer given in the form of an array.

CODE:-

```

class Solution {
public:
    static const int MOD = 1337;

    int modPow(int a, int k) {
        a %= MOD;
    }
}

```

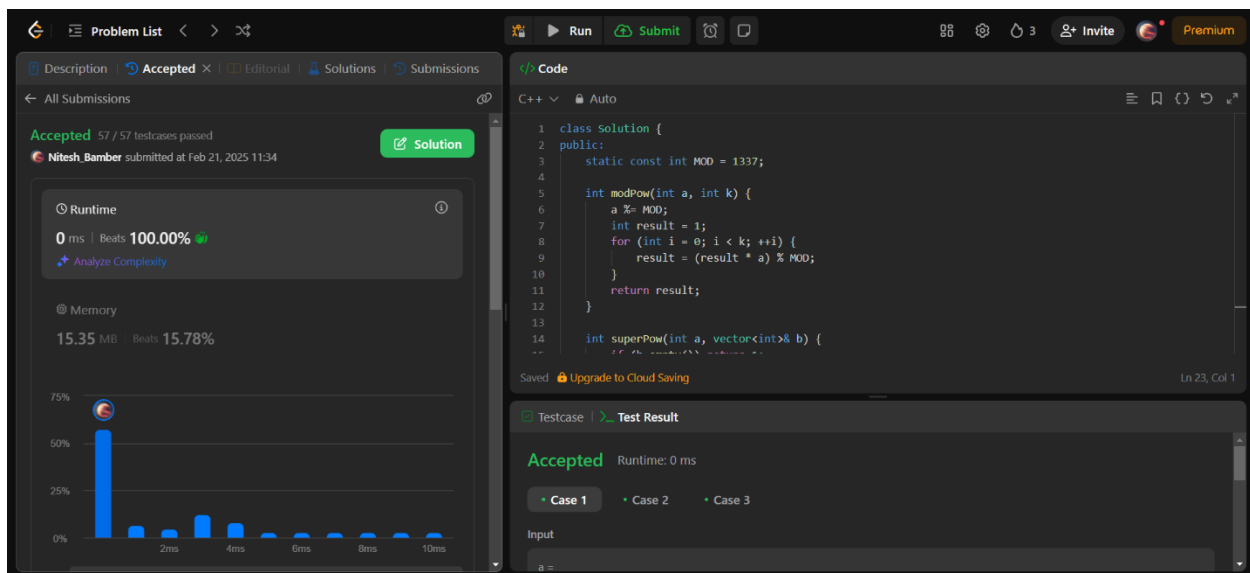
```
int result = 1;

for (int i = 0; i < k; ++i) {
    result = (result * a) % MOD;
}

return result;
}

int superPow(int a, vector<int>& b) {
    if (b.empty()) return 1;
    int lastDigit = b.back();
    b.pop_back();
    int part1 = modPow(a, lastDigit);
    int part2 = modPow(superPow(a, b), 10);
    return (part1 * part2) % MOD;
}

};
```



The screenshot displays a coding competition interface. On the left, the 'Problem List' tab is active, showing a submission by 'Nitesh_Bamber' that is 'Accepted' with 57/57 testcases passed. The submission was made on Feb 21, 2025, at 11:34. The 'Runtime' section shows 0 ms and 100.00% beats. The 'Memory' section shows 15.35 MB and 15.78% beats. A bar chart at the bottom shows the distribution of runtime results. On the right, the 'Code' tab is active, showing the C++ solution. The code defines a class 'Solution' with a static constant 'MOD = 1337'. It implements a 'modPow' function for modular exponentiation and a 'superPow' function that recursively calculates the result. The 'Testcase' tab at the bottom shows the submission is 'Accepted' with a runtime of 0 ms.

```
1 class Solution {
2 public:
3     static const int MOD = 1337;
4
5     int modPow(int a, int k) {
6         a %= MOD;
7         int result = 1;
8         for (int i = 0; i < k; ++i) {
9             result = (result * a) % MOD;
10        }
11        return result;
12    }
13
14    int superPow(int a, vector<int>& b) {
15        if (b.empty()) return 1;
16        int lastDigit = b.back();
17        b.pop_back();
18        int part1 = modPow(a, lastDigit);
19        int part2 = modPow(superPow(a, b), 10);
20        return (part1 * part2) % MOD;
21    }
22 }
```