

Assignment 4

Rupesh Bansal

22BCS50190

FL_IOT_603 A

1763. Longest Nice Substring

```
class Solution {  
public:  
    string longestNiceSubstring(string s) {  
        if (s.size() < 2) return "";  
        unordered_set<char> st(begin(s), end(s));  
        for (int i = 0; i < s.size(); i++) {  
            if (st.find((char) toupper(s[i])) == end(st) || st.find((char) tolower(s[i])) == end(st)) {  
                string s1 = longestNiceSubstring(s.substr(0, i));  
                string s2 = longestNiceSubstring(s.substr(i + 1));  
                return s1.size() >= s2.size() ? s1 : s2; }  
        }  
        return s; }  
};
```

The screenshot displays a coding platform interface. On the left, the problem description for '1763. Longest Nice Substring' is visible, including the definition of a 'nice' string and two examples. The main area shows the C++ code for the solution, which is a recursive function 'longestNiceSubstring' that checks for the presence of both uppercase and lowercase versions of each character in a substring. The code is saved and has been tested successfully, as indicated by the 'Accepted' status and 'Runtime: 0 ms' in the bottom right corner.

1763. Longest Nice Substring Solved

Easy Topics Companies Hint

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.

Example 1:
Input: $s = \text{"YazAAay"}$
Output: "aAa"
Explanation: "aAa" is a nice string because 'A/a' is the only letter of the alphabet in s , and both 'A' and 'a' appear. "aAa" is the longest nice substring.

Example 2:
Input: $s = \text{"Bb"}$
Output: "Bb"
Explanation: "Bb" is a nice string because both 'B' and 'b' appear. The whole string is a substring.

1.4K 62 18 Online

Code C++ Auto

```
1 class Solution {  
2 public:  
3     string longestNiceSubstring(string s) {  
4         if (s.size() < 2) return "";  
5         unordered_set<char> st(begin(s), end(s));  
6         for (int i = 0; i < s.size(); i++) {  
7             if (st.find((char) toupper(s[i])) == end(st) || st.find((char) tolower(s[i])) ==  
8                 end(st)) {  
9                 string s1 = longestNiceSubstring(s.substr(0, i));  
10                string s2 = longestNiceSubstring(s.substr(i + 1));  
11                return s1.size() >= s2.size() ? s1 : s2;  
12            }  
13        }  
14        return s;  
15    }  
16 }
```

Saved Ln 1, Col 1

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

$s =$

190. Reverse Bits

```
class Solution {  
  
public:  
  
    uint32_t reverseBits(uint32_t n) {  
  
        string bits = bitset<32>(n).to_string();  
  
        reverse(bits.begin(), bits.end());  
  
  
        int ans = stoll(bits, NULL, 2);  
  
        return ans;  
  
    }  
};
```

The screenshot shows the LeetCode interface for problem 190, "Reverse Bits". The problem description states: "Reverse bits of a given 32 bits unsigned integer." It includes a note about signed vs. unsigned integers and a Java example. The C++ solution is as follows:

```
1 class Solution {  
2 public:  
3     uint32_t reverseBits(uint32_t n) {  
4         string bits = bitset<32>(n).to_string();  
5         reverse(bits.begin(), bits.end());  
6  
7         int ans = stoll(bits, NULL, 2);  
8         return ans;  
9     }  
10 }  
11
```

The test results show "Accepted" with a runtime of 2 ms. The input field shows "n =".

191. Number of 1 Bits

```
class Solution {  
  
public:  
  
    int hammingWeight(int n) {  
  
        int count = 0;
```

```

        for(int i = 31; i >= 0; i--){
            if(((n >> i) & 1) == 1)
                count++;
        }
        return count;
    }
};

```

The screenshot shows the LeetCode interface for problem 191, "Number of 1 Bits". The problem description states: "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.

The solution code is written in C++:

```

class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        for(int i = 31; i >= 0; i--){
            if(((n >> i) & 1) == 1)
                count++;
        }
        return count;
    }
};

```

The test results show "Accepted" with a runtime of 0 ms. The input field shows "n =".

53. Maximum Subarray

```

class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = INT_MIN;
        int currentSum = 0;
        for (int i = 0; i < nums.size(); i++) {
            currentSum += nums[i];
            if (currentSum > maxSum) {

```

```

        maxSum = currentSum;
    }

    if (currentSum < 0) {
        currentSum = 0;
    }
}

return maxSum;
}
};

```

53. Maximum Subarray

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]`
 Output: 23
 Explanation: The subarray `[5,4,-1,7,8]` has the largest sum 23.

Solved ✓

```

1 class Solution {
2 public:
3     int maxSubArray(vector<int>& nums) {
4         int maxSum = INT_MIN;
5         int currentSum = 0;
6         for (int i = 0; i < nums.size(); i++) {
7             currentSum += nums[i];
8             if (currentSum > maxSum) {
9                 maxSum = currentSum;
10            }
11            if (currentSum < 0) {
12                currentSum = 0;
13            }
14        }
15        return maxSum;
16    }
17 };

```

Testcase Test Result
Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input
 nums =

240. [Search a 2D Matrix II](#)

```

class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int n = matrix.size(), m = matrix[0].size();
        int row = 0, col = m - 1;
    }
};

```

```

while (row < n && col >= 0) {
    if (matrix[row][col] == target) return true;
    else if (matrix[row][col] < target) row++;
    else col--;
}
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
10	13	14	17	24

Code:

```

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

```

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

matrix =

372. Super Pow

```

class Solution {
    const int base = 1337;
    int powmod(int a, int k)
    {
        a %= base;
        int result = 1;
        for (int i = 0; i < k; ++i)

```

```

        result = (result * a) % base;

    return result;

}

public:

int superPow(int a, vector<int>& b) {

    if (b.empty()) return 1;

    int last_digit = b.back();

    b.pop_back();

    return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;

}

};

```

372. Super Pow Solved

Medium Topics Companies

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.

Example 1:
Input: $a = 2, b = [3]$
Output: 8

Example 2:
Input: $a = 2, b = [1,0]$
Output: 1024

Example 3:
Input: $a = 1, b = [4,3,3,8,5,2]$
Output: 1

Constraints:

- $1 \leq a < 2^{31}$
- $1 \leq b.length \leq 2000$
- $0 \leq b[i] < 10$
- b does not contain any leading zeros except for the zero itself.

995 23 6 Online

Code

```

1 class Solution {
2     const int base = 1337;
3     int powmod(int a, int k)
4     {
5         a %= base;
6         int result = 1;
7         for (int i = 0; i < k; ++i)
8             result = (result * a) % base;
9         return result;
10    }
11 public:
12    int superPow(int a, vector<int>& b) {
13        if (b.empty()) return 1;
14        int last_digit = b.back();
15        b.pop_back();
16        return powmod(superPow(a, b), 10) * powmod(a, last_digit) % base;
17    }
18 }

```

Restored from local Upgrade to Cloud Saving Ln 1, Col 1

Testcase Test Result

Case 1 Case 2 Case 3 +

a =

2

</> Source