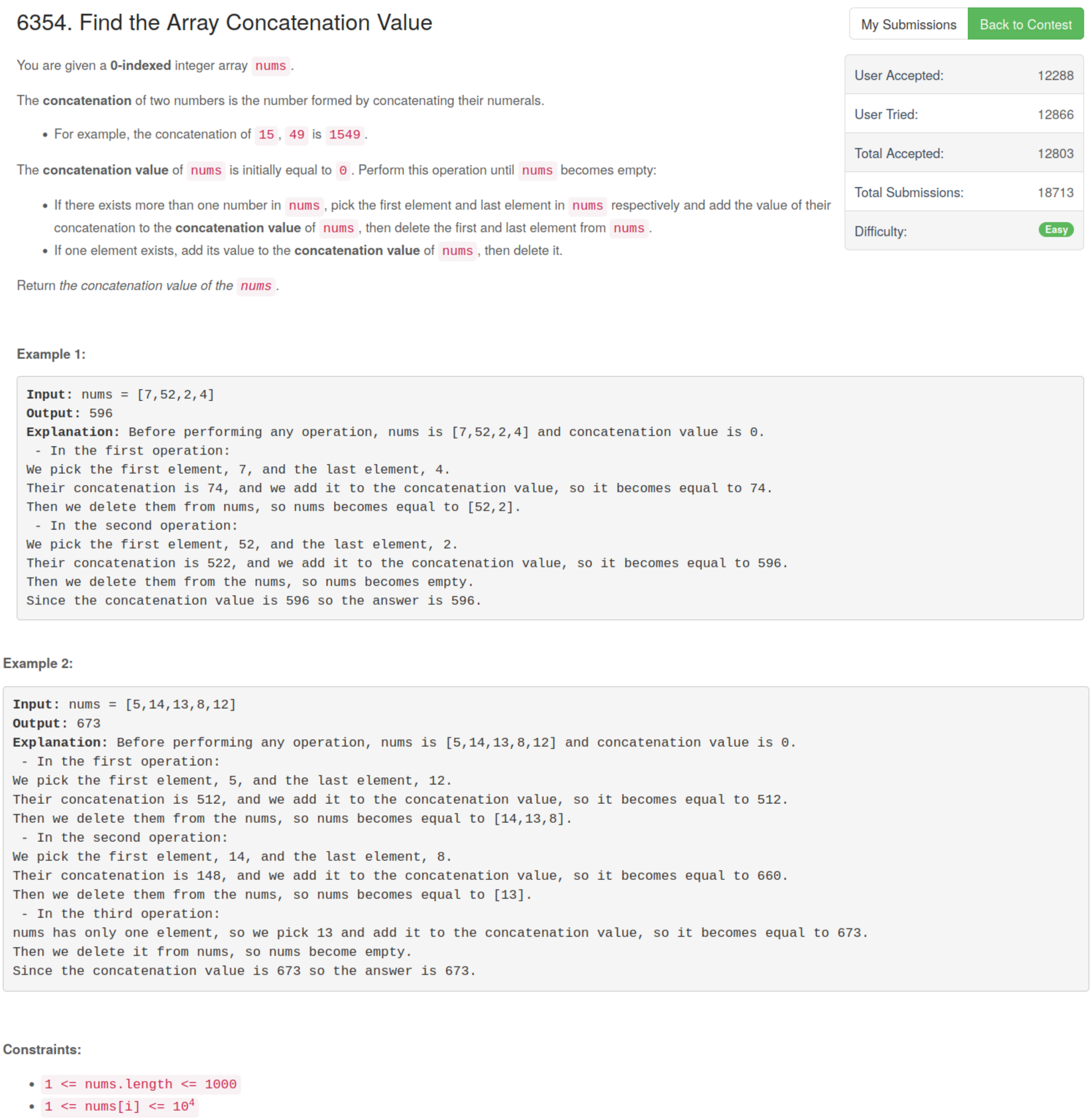
Weekly Contest 332

https://leetcode.com/contest/weekly-contest-332/

Q.1



6354. Find the Array Concatenation Value

User Accepted:	12288
User Tried:	12866
Total Accepted:	12803
Total Submissions:	18713
Difficulty:	Easy

You are given a **0-indexed** integer array `nums`.

The **concatenation** of two numbers is the number formed by concatenating their numerals.

- For example, the concatenation of `15`, `49` is `1549`.

The **concatenation value** of `nums` is initially equal to `0`. Perform this operation until `nums` becomes empty:

- If there exists more than one number in `nums`, pick the first element and last element in `nums` respectively and add the value of their concatenation to the **concatenation value** of `nums`, then delete the first and last element from `nums`.
- If one element exists, add its value to the **concatenation value** of `nums`, then delete it.

Return *the concatenation value of the* `nums`.

Example 1:

```
Input: nums = [7,52,2,4]
Output: 596
Explanation: Before performing any operation, nums is [7,52,2,4] and concatenation value is 0.
 - In the first operation:
We pick the first element, 7, and the last element, 4.
Their concatenation is 74, and we add it to the concatenation value, so it becomes equal to 74.
Then we delete them from nums, so nums becomes equal to [52,2].
 - In the second operation:
We pick the first element, 52, and the last element, 2.
Their concatenation is 522, and we add it to the concatenation value, so it becomes equal to 596.
Then we delete them from the nums, so nums becomes empty.
Since the concatenation value is 596 so the answer is 596.
```

Example 2:

```
Input: nums = [5,14,13,8,12]
Output: 673
Explanation: Before performing any operation, nums is [5,14,13,8,12] and concatenation value is 0.
 - In the first operation:
We pick the first element, 5, and the last element, 12.
Their concatenation is 512, and we add it to the concatenation value, so it becomes equal to 512.
Then we delete them from the nums, so nums becomes equal to [14,13,8].
 - In the second operation:
We pick the first element, 14, and the last element, 8.
Their concatenation is 148, and we add it to the concatenation value, so it becomes equal to 660.
Then we delete them from the nums, so nums becomes equal to [13].
 - In the third operation:
nums has only one element, so we pick 13 and add it to the concatenation value, so it becomes equal to 673.
Then we delete it from nums, so nums become empty.
Since the concatenation value is 673 so the answer is 673.
```

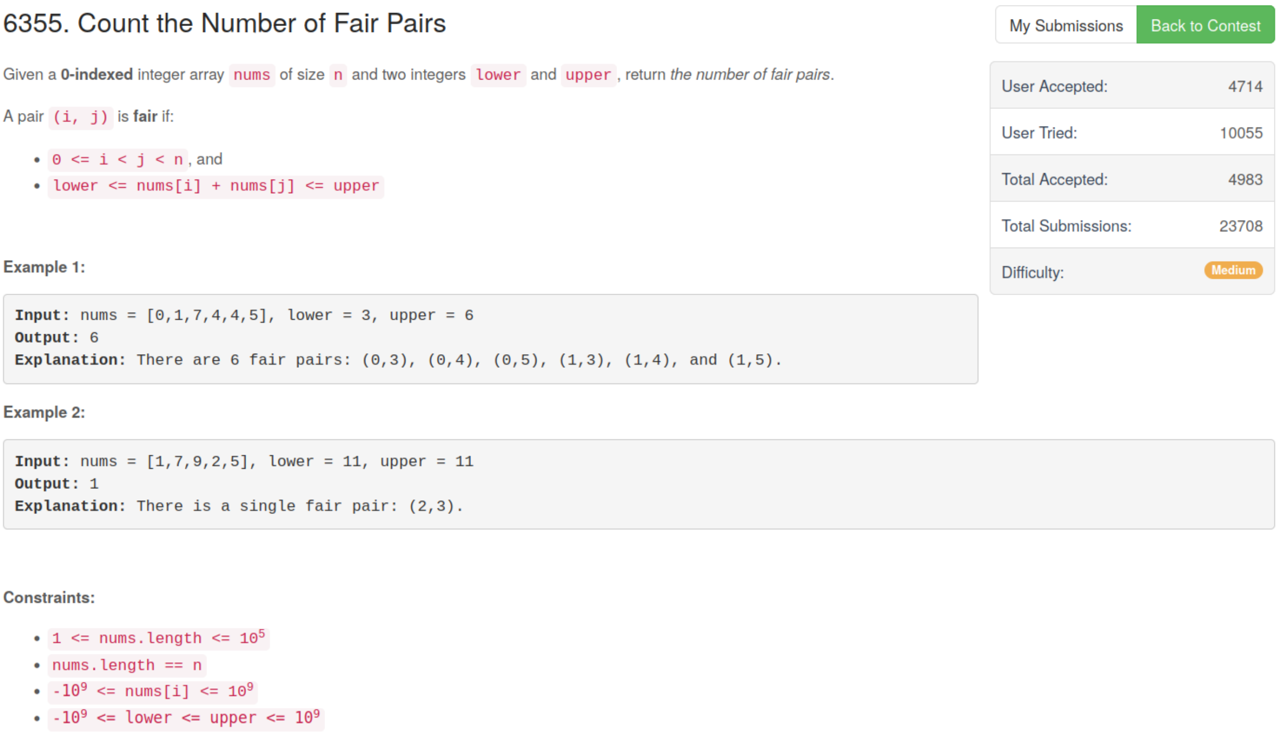
Constraints:

- `1 <= nums.length <= 1000`
- `1 <= nums[i] <=` 10^4

```python
class Solution:
    def findTheArrayConcVal(self, nums: List[int]) -> int:
        n = len(nums)
        ans = 0
        j = n-1
        i = 0
        for i in range(n//2):
            if i>j:
                break
            j = j-1
            num1 = "{}".format(nums[i])
            num2 = "{}".format(nums[n-i-1])
            temp = num1+num2
            temp = int(temp)
            ans += temp

        # print(i, j)

        if n%2 != 0:
            ans += nums[i+1]

        return ans
```

```
class Solution {
public:
    long long findTheArrayConcVal(vector<int>& nums) {
        long long ans = 0;
        for (int i = 0, j = nums.size()-1; i <= j; ++i, --j) {
            if (i == j) ans += nums[i];
            else {
                int val = nums[i];
                vector<int> digits;
                for (int x = nums[j]; x; x /= 10) digits.push_back(x % 10);
                reverse(digits.begin(), digits.end());
                for (auto& d : digits) val = 10*val + d;
                ans += val;
            }
        }
        return ans;
    }
};
```

Q.2.



6355. Count the Number of Fair Pairs

User Accepted:	4714
User Tried:	10055
Total Accepted:	4983
Total Submissions:	23708
Difficulty:	Medium

Given a **0-indexed** integer array `nums` of size `n` and two integers `lower` and `upper`, return *the number of fair pairs*.

A pair `(i, j)` is **fair** if:

- `0 <= i < j < n`, and
- `lower <= nums[i] + nums[j] <= upper`

Example 1:

```
Input: nums = [0,1,7,4,4,5], lower = 3, upper = 6
Output: 6
Explanation: There are 6 fair pairs: (0,3), (0,4), (0,5), (1,3), (1,4), and (1,5).
```

Example 2:

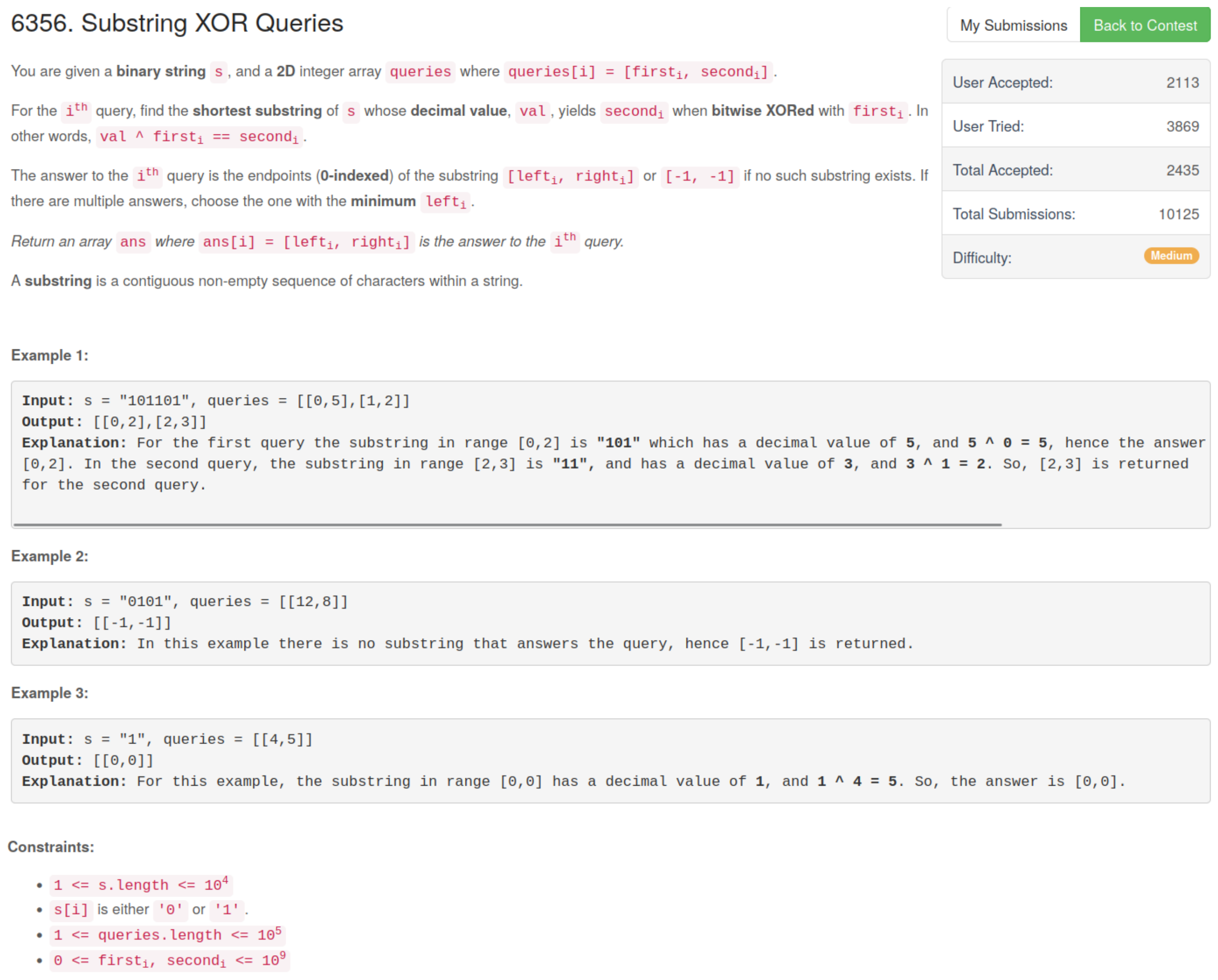
```
Input: nums = [1,7,9,2,5], lower = 11, upper = 11
Output: 1
Explanation: There is a single fair pair: (2,3).
```

Constraints:

- 1 <= nums.length <= 10^5
- `nums.length == n`
- -10^9 <= nums[i] <= 10^9
- -10^9 <= lower <= upper <= 10^9

```cpp
class Solution {
public:
    long long countFairPairs(vector<int>& nums, int lower, int upper) {
        sort(nums.begin(), nums.end());
        int n = nums.size(), lo = n-1, hi = n-1;
        long long ans = 0;
        for (int i = 0; i < n; ++i) {
            while (0 <= hi && nums[i] + nums[hi] > upper) --hi;
            while (0 <= lo && nums[i] + nums[lo] >= lower) --lo;
            ans += hi - lo;
            if (i > lo && i <= hi) --ans;
        }
        return ans/2;
    }
};
```

Q.3.



6356. Substring XOR Queries

User Accepted:	2113
User Tried:	3869
Total Accepted:	2435
Total Submissions:	10125
Difficulty:	Medium

You are given a **binary string** `s`, and a **2D** integer array `queries` where `queries[i] = [`$first_i$`, `$second_i$`]`.

For the i^{th} query, find the **shortest substring** of `s` whose **decimal value**, `val`, yields $second_i$ when **bitwise XORed** with $first_i$. In other words, `val ^ `$first_i$` == `$second_i$.

The answer to the i^{th} query is the endpoints (**0-indexed**) of the substring `[`$left_i$`, `$right_i$`]` or `[-1, -1]` if no such substring exists. If there are multiple answers, choose the one with the **minimum** $left_i$.

Return an array `ans` *where* `ans[i] = [`$left_i$`, `$right_i$`]` *is the answer to the* i^{th} *query.*

A **substring** is a contiguous non-empty sequence of characters within a string.

Example 1:

```
Input: s = "101101", queries = [[0,5],[1,2]]
Output: [[0,2],[2,3]]
Explanation: For the first query the substring in range [0,2] is "101" which has a decimal value of 5, and 5 ^ 0 = 5, hence the answer
[0,2]. In the second query, the substring in range [2,3] is "11", and has a decimal value of 3, and 3 ^ 1 = 2. So, [2,3] is returned
for the second query.
```

Example 2:

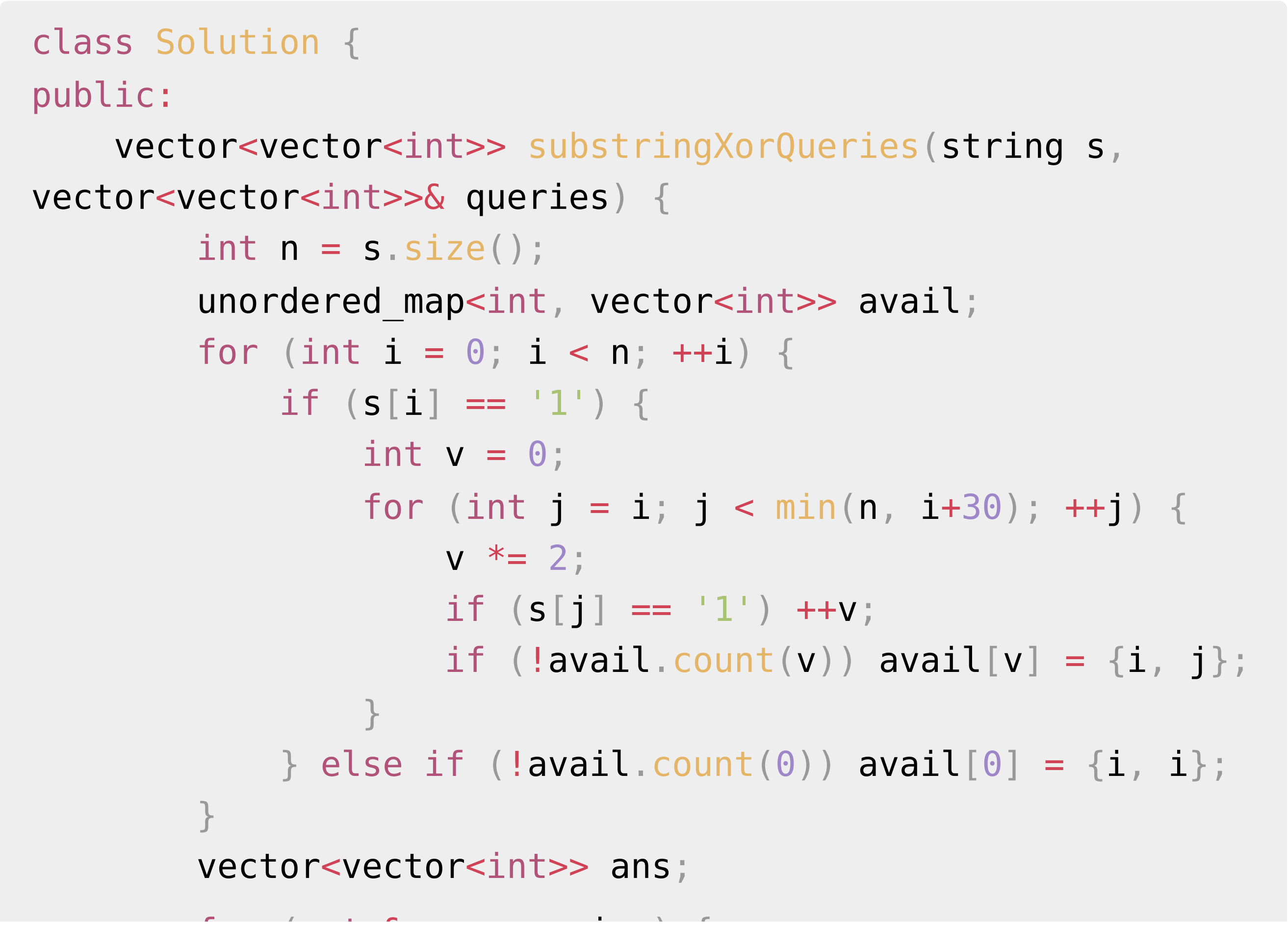
```
Input: s = "0101", queries = [[12,8]]
Output: [[-1,-1]]
Explanation: In this example there is no substring that answers the query, hence [-1,-1] is returned.
```

Example 3:

```
Input: s = "1", queries = [[4,5]]
Output: [[0,0]]
Explanation: For this example, the substring in range [0,0] has a decimal value of 1, and 1 ^ 4 = 5. So, the answer is [0,0].
```

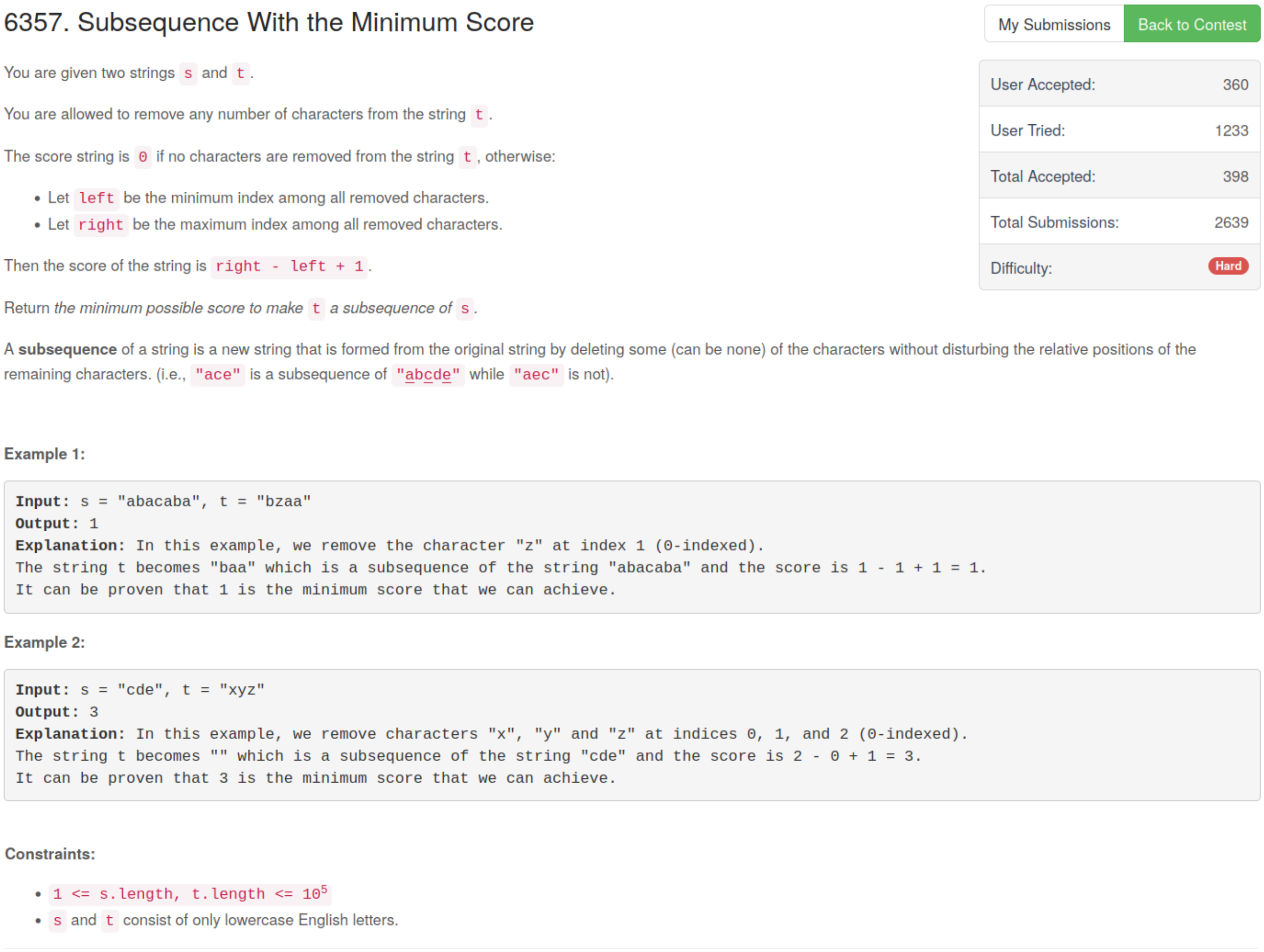
Constraints:

- $1 <= s.length <= 10^4$
- `s[i]` is either `'0'` or `'1'`.
- $1 <= queries.length <= 10^5$
- $0 <= first_i, second_i <= 10^9$



```
class Solution {
public:
    vector<vector<int>> substringXorQueries(string s,
vector<vector<int>>& queries) {
        int n = s.size();
        unordered_map<int, vector<int>> avail;
        for (int i = 0; i < n; ++i) {
            if (s[i] == '1') {
                int v = 0;
                for (int j = i; j < min(n, i+30); ++j) {
                    v *= 2;
                    if (s[j] == '1') ++v;
                    if (!avail.count(v)) avail[v] = {i, j};
                }
            } else if (!avail.count(0)) avail[0] = {i, i};
        }
        vector<vector<int>> ans;
        for (auto& q : queries) {
            int v = q[0] ^ q[1];
            if (avail.count(v)) ans.push_back(avail[v]);
            else ans.push_back({-1, -1});
        }
        return ans;
    }
};
```

Q.4.



6357. Subsequence With the Minimum Score

User Accepted:	360
User Tried:	1233
Total Accepted:	398
Total Submissions:	2639
Difficulty:	Hard

You are given two strings `s` and `t`.

You are allowed to remove any number of characters from the string `t`.

The score string is `0` if no characters are removed from the string `t`, otherwise:

- Let `left` be the minimum index among all removed characters.
- Let `right` be the maximum index among all removed characters.

Then the score of the string is `right - left + 1`.

Return *the minimum possible score to make* `t` *a subsequence of* `s`.

A **subsequence** of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., `"ace"` is a subsequence of `"abcde"` while `"aec"` is not).

Example 1:

```
Input: s = "abacaba", t = "bzaa"
Output: 1
Explanation: In this example, we remove the character "z" at index 1 (0-indexed).
The string t becomes "baa" which is a subsequence of the string "abacaba" and the score is 1 - 1 + 1 = 1.
It can be proven that 1 is the minimum score that we can achieve.
```

Example 2:

```
Input: s = "cde", t = "xyz"
Output: 3
Explanation: In this example, we remove characters "x", "y" and "z" at indices 0, 1, and 2 (0-indexed).
The string t becomes "" which is a subsequence of the string "cde" and the score is 2 - 0 + 1 = 3.
It can be proven that 3 is the minimum score that we can achieve.
```

Constraints:

- `1 <= s.length, t.length <= `10^5
- `s` and `t` consist of only lowercase English letters.

```
class Solution {
public:
    int minimumScore(string s, string t) {
        vector<int> p;
        int j = 0;
        for (int i = 0; i < s.size(); ++i) {
            if (j < t.size() && s[i] == t[j]) ++j;
            p.push_back(j);
        }
        int ans = t.size() - j;
        j = t.size()-1;
        for (int i = s.size()-1; i >= 0; --i) {
            ans = min(ans, max(0, j - p[i] + 1));
            if (0 <= j && s[i] == t[j]) --j;
        }
        return min(ans, j+1);
    }
};
```