SEC Inter University Junior Programming Contest, 2022

Problemset Analysis

steinum, Tahseen, Aritra741, tanvirtareq, border

September 3, 2022

A. SUBSTRING SEARCH

Setter: steinum

Concatanating all the strings, $s_1, s_2, \ldots, s_n$, resulting string s(we need to use delimiter character). Now we have to build a suffix array over s. We will have sa, isa – suffix array, inverse suffix array(i.e. $isa[sa[i]] = i$ for all i).

From the suffix array, we can calculate $lcp(l, r)$, which means: length of longest common prefix of suffix $sa[l]$ and suffix $sa[r]$.

Now, for query i, l, r, j we have to find the starting position(not occurrence) of $s_i[l \ldots r]$ in s. Let it be, $f(i, l)$. Suppose, $I = isa[f(i, l)]$. Now, we have to find minimum, L such that $L \leq I$ and $lcp(L, I) \geq r - l + 1$, again find the maximum R, such that $R \geq I$ and $lcp(I, R) \geq r - l + 1$. Now the answer is the number of suffixes that is responsible for s_j in $sa[L \ldots R]$. This part can be done using adjacency list type structure + binary search.

B. TREASURE

Setter: Tahseen

Let, a and b be the number of guards inside and outside the castle respectively.
At the beginning $a = n$ and $b = 0$
For each Activity, **in** or **out** we need to update a and b accordingly, i.e. for **in** command we need to do $a++, b--$, and for **out** command we need to do $a--, b++$. We need to find the minimum value of a overall activity. The answer is **YES** if the minimum value of a is smaller than 4, otherwise **NO**.

C. PASS THE PARCEL

Setter: tanvirtareq

The answer to the problem is $k \pmod{n}$, i.e. remainder when k is divided by n.

D. MATHEMAGIC

Setter: Aritra741

Let, $g(x) = \sum_{i=1}^{n} [x|a_i]$, i.e. number of multiples of x in the array a.
Then, $f(x) = 2^{g(x)} - 1$

Suppose we have a frequency array, $freq$. $freq[i]$ = number of occurrence of i in the array a

Therefore, $g(i) = \sum_{j=i}^{\lfloor \frac{10^5}{i} \rfloor} freq[i \times j]$

Hence, we can calculate, $g(a_i)$ for all a_i, where $1 \le i \le n$, as well as $f(a_i)$.

E. YET ANOTHER STRONG PROBLEM

Setter: Tahseen

Let's assume, our initial string is s_0, a k length binary string, after one iteration our string will become s_1 i.e. $s_1 = f(s_0, 1)$. And after n^{th} iteration our string will become, s_n, i.e. $s_n = f(s_0, n)$.

Now, think about the transition function from s_i to s_{i+1}. For any index, j in s_{i+1}, we can say that $s_{i+1}[j] = (s_i[j-1] + s_i[j] + s_i[j+1] + 1) \pmod 2$

We can reproduce the transition as below:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & \dots & \dots & 0 & 0 \\ 1 & 1 & 1 & 0 & \dots & \dots & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & \dots & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & 0 & 1 & 1 & 1 & 1 \\ 0 & \dots & \dots & \dots & \dots & 0 & 1 & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 & 1 \end{pmatrix} \begin{pmatrix} s_i[0] \\ s_i[1] \\ s_i[2] \\ \dots \\ \dots \\ s_i[k-2] \\ s_i[k-1] \\ 1 \end{pmatrix} = \begin{pmatrix} s_{i+1}[0] \\ s_{i+1}[1] \\ s_{i+1}[2] \\ \dots \\ \dots \\ s_{i+1}[k-2] \\ s_{i+1}[k-1] \\ 1 \end{pmatrix}$$

Hence, to find s_n we can just use the formula:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & \dots & \dots & 0 & 0 \\ 1 & 1 & 1 & 0 & \dots & \dots & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & \dots & 0 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & 0 & 1 & 1 & 1 & 1 \\ 0 & \dots & \dots & \dots & \dots & 0 & 1 & 0 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 & 1 \end{pmatrix}^n \begin{pmatrix} s_0[0] \\ s_0[1] \\ s_0[2] \\ \dots \\ \dots \\ s_0[k-2] \\ s_0[k-1] \\ 1 \end{pmatrix} = \begin{pmatrix} s_n[0] \\ s_n[1] \\ s_n[2] \\ \dots \\ \dots \\ s_n[k-2] \\ s_n[k-1] \\ 1 \end{pmatrix}$$

F. BROKEN WALL

Setter: border

For the given constraints, the problem can be solved in a lot of different ways. But using some observations, we can make the task of implementation much easier.

One important observation is, let $\{x_1, x_2, \ldots, x_n\}$ and $\{y_1, y_2, \ldots, y_m\}$ be two bitsets, the (i, j)th square of the wall can be described as $g[i][j] = x[i] \oplus y[j]$. Now there are two cases:

- **Case 1**: The wall is fully destroyed. Finding a valid wall for this case is trivial.
- **Case 2**: If the wall isn't fully destroyed, some row or column of the wall is left undamaged completely. So, this problem converts to finding one of the x or y sequences given one of them. For such small constraints, we can just brute-force all possible sequences and check if the generated wall is consistent will the non-destroyed squares of the original wall.

G. GOOD ARRAY

Setter: tanvirtareq

Here is some observation for this problem:

- **observation 0**: If the size of the array is odd, then the array can't be good.
- **observation 1**: If $k = 1$, then the array can't be good.
- **observation 2**: An array will be a **good array** if the maximum frequency of the numbers is not greater than $\frac{n}{2}$, where n is the size of that array.
- **observation 3**: We can find the number of **non-good array**. Then subtracting it from the total number of the possible array, we will find our desired answer.

So what will be the number of **non-good array**?
Suppose, we choose a value x from $[1, k]$, which has the maximum frequency, and the maximum frequency is i. So we have to choose i index from n available index in the array. We can do this in $\binom{n}{i}$ ways. Also, we can choose x from $[1, k]$ in k ways. And the remaining elements will be from $[1, x) \cup (x, k]$, and available indexes are $n - i$. Hence, we can fill the empty indexes of the array in $(k-1)^{(n-i)}$ ways(fun fact: there won't be any overcount, because $n - i < i$ and hence no values other than x can have maximum frequency).

Therefore, number of **non-good array** will be $\sum_{i=\frac{n}{2}+1}^{n} \binom{n}{i} \times k \times (k-1)^{(n-i)}$.

Hence, our answer(number of **good array**) will be, $\boxed{k^n - \sum_{i=\frac{n}{2}+1}^{n} \binom{n}{i} \times k \times (k-1)^{(n-i)}}$

H. MEXIMUM

Setter: Aritra741

For each node, i, if we can remove all the elements (which occurs in the path from 1 to i) from the set, $S = \{0, 1, 2, \ldots, n\}$, then for i our score will be equal to the minimum element in S.

We can solve this problem using DFS. We will maintain a set, $S = \{0, 1, 2, \ldots, n\}$. Before processing any of the child of node u, we will remove a_u from the set S(if this value exists in S), and calculate the score for node u. And after processing all of the child of node u, we will add a_u in the set S(if we removed this value before).

NB: you can just ignore all those values which are greater than n.

The DFS function will look like this.

```
vector<int> g[mx];
long long a[mx], maximum_mex = -1;
set<long long> S;
void dfs(int u = 1, int par = -1) {
        bool is_removed = 0;
        if (S.count(a[u])) {
                S.erase(a[u]);
                is_removed = 1;
        }
        maximum_mex = max(maximum_mex, *S.begin());
        for (int v : g[u]) {
                if (v != par) {
                        dfs(v, u);
                }
        }
        if (is_removed) {
                S.insert(a[u]);
        }
}
```

I. SUM EQUALS LCM

Setter: border

There are a lot of interesting ways to solve this problem. Let's discuss the easiest one. Observe that $\mathbf{lcm}(n-1, n) = (n-1) \times n$. If we use $(n-1)$ and n in our sequence, we still have to choose $(n-2)$ positive numbers whose sum equals $(n \times (n-1)) - (n + (n-1)) = (n \times (n-3)) + 1$. So, for the remaining $(n-2)$ numbers, we can use the number n, $(n-3)$ times and 1 once. The final sequence will look like: $\{1, n-1, n, n, \ldots, n\}$.

J. AVERAGE QUERY

Setter: steinum

Here is some observation for this problem:

- **observation 0**: If we need to minimize $U - 2^{-L}$, we have to minimize U, if there are multiple solutions where U is the same, we need to minimize L.
- **observation 1**: If x exists in s, then $c_i = \begin{cases} 1; \text{ if } s_i = x \\ 0; \text{ otherwise} \end{cases}$
- **observation 2**: If all elements in s is smaller/greater than x, then the answer is **NO**.
- **observation 3**: In all other cases(except the previous two observations), exactly two elements in the array c can be non-zero(if we want to minimize the cost).
  Hint: take two such elements, s_i and s_j that $s_i < x < s_j$, then $\frac{s_i \times c_i + s_j \times c_j}{c_i + c_j} = x \implies c_i(x - s_i) = c_j(s_j - x)$. So, we can use $c_i = \frac{s_j - x}{g}$ and $c_j = \frac{x - s_i}{g}$, where, $g = gcd(x - s_i, s_j - x)$.
- **observation 4**: We have to find a (i, j) for which $s_i < x < s_j$. and $c_i + c_j = \frac{(s_j - x) + (x - s_i)}{g}$ is minimum.

Now, make two array, $A = [x - s_i : s_i < x]$ and $B = [s_i - x : s_j > x]$.
Now, the problem turns into finding (i, j) for which $\frac{A_i + B_i}{gcd(A_i, B_i)}$.
We can, iterate over, $g \in [1, 10^9]$ and find $\frac{g \times p + g \times q}{g} = p + q$, where, $g \times p$ is the minimum multiple of g in A, and $g \times q$ is the minimum multiple of g in B. The minimum of this value over all g is our desired answer.

Note that, we don't need to iterate over all values in the range $[1, 10^9]$. We only need to iterate over the all the divisors of the elements in A and B.

To calculate all divisors of a number, we can prime factorize the number, and then just using backtrack we can generate all the divisors.