UW Madison Data Structure

Nitit Jongsawatsataporn
n.jongsawatsataporn [at] mail.utoronto.ca

October 26, 2024

Problem A

Problem A

Given a sequence of number $A_1, A_2, \ldots, A_n$ arrange in circular order, want to divide into k consecutive segments such that

$$\wedge_{\text{interval}}(\vee_{A_i \in \text{interval}_j} A_i)$$

is maximized.
Constrain: $N, K \leq 5e5$, $1 \leq A_i \leq 1e9$.

Problem A (cont)

Observation

1. Suppose the interval starts at A_1, then there are at most 32 possible values of $\vee_{A_i \in \text{interval}} A_i$.
2. For each fix position i and a bit b, we find the next location $k > i$ that A_k has bit b. (k can wrap around the circle if needed)
3. Precomputation could be done efficiently using two pointers.
4. The precomputation take $O(32N)$.

Solution for a line case (assume the first interval starts at A_1)

1. Searching for the maximum bit by bit. If S is the answer, then we should be able to construct K consecutive sequences such that $(\vee_{A_i \in \text{interval}} A_i) \wedge (S) = S$.
2. Greedy algorithm works here! Let the pattern we check be L, then we just need to jump in the bit that presents in L, that would be the shortest segment that has bitwise or of its value to agree with L.
3. We test L bit by bit.
4. Time complexity $O(32^2 K)$

Problem A (cont)

Solution for circular case

1. Fix the search pattern L
2. Let's pretend the first interval start at index 1
3. Run the line case
4. If the line case work fine, we can continue, otherwise the last interval might needs some help.
5. Sometimes, A_1 is not needed, we can shrink the first interval until we can not remove the prefix anymore.
6. Check whether the last interval works with the additional segment.

Time complexity: $O(32N + 32^2K + 32^2)$

Problem B

Use Fenwick tree.
Useful link:

1. CP Agorithm
2. Codeforces blog

Note: Segment tree is too slow for this problem. However, non-recursive segment tree is quite fast and might be able to pass with optimal implementation. Link: Codeforces blog.

Problem C

The key is a well-known data structure called suffix array CP Algorithm. The use of suffix tree and suffix automata should pass as well.

Problem D

Problem D

Given N binary strings $s_1, \ldots, s_N$ (consisting of $0, 1$ only), print out the lexicographically first longest binary string that does not have any of the listed strings as a substring, or report that there is no longest such string (output -1).

Observation: Let's first consider the problem of checking whether a candidate string satisfies the requirements or not.

Inverse Problem

Given a string S and a number of pattern strings $s_1, \ldots, s_N$, check whether some s_i is a substring of S or not.

Inverse Problem

This problem is known as pattern matching. There are a number of standard algorithms.
Single pattern:

1. KMP: CP Algorithms.
2. Rabin-Karp: CP Algorithms.

Multiple patterns:

1. Aho-Corasick: CP Algorithms.

Problem D (cont)

Solution

1. Build the Aho-Corasick string matching automaton.
2. Remove all edges going out from an accepting node. This ensures that if a pattern is matched, we don't proceed.
3. The length of the answer is the maximum length we can walk without hitting an accepting node.
4. We output -1 if there is a loop.
5. Otherwise, we have a directed graph without a loop, so the nodes can be ordered by topological sort.
6. Using that order, we can compute the length of the answer by standard dynamic programming (dp[node] = maximum walk length that end at this node) and retrieving the lexicographically first path.

Time complexity: $O(|S|)$

Problem E

Problem E

Given a sequence of integers $M_1, \ldots, M_N$, check whether there is are indices $c < d < e < f$ such that $M_c = M_e \neq M_d = M_f$.

This is actually a data structure problem!

Problem E (cont)

Claim

If such position exists, we can pick c, d, e, f such that $M_c = M_e \neq M_d = M_f$, and for any $g \in (c, e)$, $h \in (d, f)$ we also get $M_g \neq M_c$ and $M_h \neq M_d$. (In other words, only need to consider consecutive occurrences.)

Proof:
Let $c < d < e < f$ have the property that $M_c = M_e \neq M_d = M_f$. Let $c' < d$ be the last time $M_{c'} = M_c$ but before position d and e' be the first time after d that $M_{e'} = M_c$. Then, $M_{c'} = M_{e'} \neq M_d = M_f$. Moreover, c', e' are consecutive occurrences of M_c.
Similarly, we can also guarantee that d, f are consecutive occurrences as well.

Problem E (cont)

Consider the consecutive occurrences for each of the character. Note that the total number of consecutive occurrences is at most N. The problem reduces to the following.

Reformulated problem

Given N intervals, find the lexicographically first pair of intervals (A, B) such that A intersects B but neither A contains B nor B contains A.

To check whether such pair exists or now, we can use stack (check if it's in FILO order or not). However, if we want minimal lexicographical order, a segment tree is needed.

Problme E (cont)

Solution

1. First, ignore all the character that appears only once.
2. The idea is to sweep from left to right. Then, for each character encountered, we just need to check, among active characters (that is a part of ongoing interval), which one is the lowest lexicographically.
3. We will find the minimum along the active characters by keeping a segment tree of active characters, and query the minimum.
4. Sweep a pointer along the array. We have two things to do. If the characters we found is NOT its first occurrence, we look between the previous (directly before this one) occurrence, and use RMQ to find a minimal lexicographical pairs that we want. We deactivate (remove from the segment tree) the previous occurrence.
5. If the characters we found is NOT its last occurrence, we activate (put in segment tree) the current character.

Time Complexity $O(|M| \log |M|)$

Problem F

Problem F

Given an empty array of size N $(1 - N)$. There are three types of queries.

1. $+1$ to all position from a_j to b_j
2. reverse one of the first operation (-1 to that range). Each operation can be reversed at most once
3. report how many slot from 1 to n has value 0.

Solution:

1. Persistence segment tree: Link good explanation by SecondThread
2. Normal segment tree

Problem F (cont)

Classical Problem

Given an array A originally consisted of 0. We have the following two operations

1. Select interval $[l, r]$ and an integer a, change $A[x] \leftarrow A[x] + a$ for $l \leq x < r$.
2. Give interval $[l, r]$, answer a pair (a, b) when $a = \min_{x \in [l,r]} A[x]$ and how many $x \in [l, r]$ such that $A[x] = a$.

Solution

1. Build a segment tree with three tags: minimum number, how many times that minimum number appeared on that interval, and the current lazy contribution that tagged on this node
2. To update is update like normal lazy segment tree

Observe that we never have to push the tag as we can keep the lazy tag not-summed when we only care at about value 0.
Time complexity: $O(N \log N)$