AND xor OR

locked

by bokamvinaykumar

Editorial by Omar Khaled

Statistics
Difficulty: Hard
Publish Date: Apr 30 2017

Pre-requisites: Bit Manipulation, Data Structure, Stack.

Difficulty Level: Hard.

Hints: Make a stack that maintains a strictly increasing sub-sequence and simplify the giving expression for S_i.

Editorial:

Firstly: it is needed to simplify the given expression $S_i = (((M_1 \wedge M_2) \oplus (M_1 \vee M_2)) \wedge (M_1 \oplus M_2))$. Let M_1 consists of i bits $(a_1, a_2, \ldots\ldots, a_i)$ and M_2 consists of j bits $(b_1, b_2, \ldots\ldots, b_j)$.

Let $res_1 = (M_1 \wedge M_2)$ = a number which have bits of value 1 at the positions where $a_i = b_j = 1$ and $i = j$.
Let $res_2 = (M_1 \vee M_2)$ = a number which have bits of value 1 at the positions where $a_i = 1$ or $b_j = 1$ and $i = j$.
Let $res_3 = (M_1 \oplus M_2)$ = a number which have bits of value 1 at the positions where $a_i = 1$ or $b_j = 1$, but not both, and $i = j$.
Let $res_4 = ((M_1 \wedge M_2) \oplus (M_1 \vee M_2))$ = a number which have bits of value 1 at the positions where $a_i = 1$ or $b_j = 1$, but not both, and $i = j$. Let $S_i = res_5 = (((M_1 \wedge M_2) \oplus (M_1 \vee M_2)) \wedge (M_1 \oplus M_2))$ = a number which have bits of value 1 at the positions where $a_i = 1$ or $b_j = 1$, but not both, and $i = j$.

So after simplifying the expression, $S_i = M_1 \oplus M_2$.

Secondly: to find S_i in every possible interval, it is needed to have a stack (st). This stack will ensure that S_i in every interval will be calculated using the correct M_1 and M_2. (st) should have always a strictly increasing sub-sequence. Now push A_1 and A_2 in (st) and set $S_i = A_1$ xor A_2. After that loop over the input values (A_i) starting from $i >= 3$. Then 2 cases will be considered.

Case 1#:

If A_i > the top (T) of the stack, then push A_i in the stack. $M_1 = T, M_2 = A_i$, so $S_i = T \oplus A_i$.

Case 2#:

If $A_i <= T$, then keep popping from the stack (st) until either the stack is empty or $A_i > T$. Whenever you pop a number with value (V) from (st), maximize between the old S_i and the current S_i which equals $V \oplus A_i$. This is done because when $A_i <= T$, that means A_i could be M_1 or M_2 in longer intervals so we keep popping from (st) and calculating S_i in the new interval using A_i.

For more clarification, consider this example:

5
3 6 9 5 7

So push A_1 and A_2 in (st) and set $S_i = A1 \oplus A_2$. Now loop over the input values A_i from $i >= 3$. As $(A_3 = 9 > T = 6)$, so apply case 1 and set $M_1 = T$, $M_2 = A_i$, $S_i = max(s_i, T \oplus A_3)$. At $(i = 4)$ these are the values $((A_4 = 5) <= (T = 9))$, so apply case 2 and keep popping from the stack and maximize S_i. Now $S_i = max(S_i, (T = 9) \oplus A_4)$ then pop from (st) so T will be updated and $S_i = max(S_i, (T = 6) \oplus A_4)$ then pop from (st) so T will be updated and $S_i = max(S_i, (T = 3) \oplus A_4)$. Now $A_4 > T$, so stop popping from (st). At ($i = 5$) these are the values $(A_5 = 7 > T = 5)$, so apply case 1 and set $M_1 = T = 5$, $M_2 = A_5 = 7$, $S_i = max(S_i, T \oplus A_5)$.

Note: There is no need to use long integers as the result of $A_i \oplus A_j$ will never exceeds 10^9 where $A_i, A_j <= 10^9$.

Time Complexity: $O(N)$.
Memory Space Complexity: $O(N)$.

Solution#:

```
#include <bits/stdc++.h>

using namespace std;

#define MAXN 1000000

int a[MAXN + 1];

void solve(int n) {

    stack<int> s;

    int result = INT_MIN, cur;

    for (int i = 0; i < n; ++ i) {
       while (!s.empty() && s.top() >= a[i]) {

          int tmp = s.top(); s.pop();
          result = max(result, tmp ^ a[i]);
       }

       if (!s.empty()) result = max(result, a[i] ^ s.top());
       s.push(a[i]);
    }

    printf("%d\n", result);
}

int main() {

    int N;

    scanf("%d", &N);
    for (int i = 0; i < N; ++ i) scanf("%d", &a[i]);

    solve(N);

    return 0;
}
```