

XOR 고양이

출제자: 김준겸

정해: Trie

풀이

먼저, XOR 연산의 성질에 대해서 생각해봅시다. XOR 연산은 교환법칙을 만족하기 때문에, 연산 과정에서 같은 수가 2번 등장한다면 $a \oplus a = 0$ 이 되고 상쇄되어 의미 없는 연산이 됩니다. 이 연산의 성질을 이용하면, 어떤 장소 A 에서 B 로 특정 경로를 거쳐 이동할 수 있는지 여부를 빠르게 판단할 수 있습니다.

특정 경로를 지나는 도중 경유하는 장소의 주소를 a_1, a_2, \dots, a_t 라고 합시다. 그러면 각 이동마다 발생하는 소음은

$$a_1 \oplus a_2, a_2 \oplus a_3, \dots, a_{t-1} \oplus a_t$$

가 됩니다. 이 소음을 모두 XOR하면 가운데 항들은 모두 사라지고 $a_1 \oplus a_t$ 가 됩니다. 결과적으로 A 에서 B 로 갈 수 있는지 여부를 판단하는 것은 이 두 장소의 주소를 XOR한 값이 K 이하인지를 판단하는 것과 동치입니다.

이 문제는 N 이 20만으로 꽤 크기 때문에, Naive하게 모든 쌍에 대해서 이를 판단하는 것은 불가능합니다. 대신 Binary Trie를 사용해서 개수를 빠르게 구할 수 있습니다. 트라이에 모든 주소를 이진수 형태로 넣어두면 특정 수 x 와 XOR한 결과가 K 이하가 되는 수의 개수를 $M = 2^{30}$ 에 대해 $O(\lg M)$ 에 찾을 수 있습니다. 따라서 총 시간복잡도는 $O(N \lg M)$ 이 됩니다.

코드

```
#include <bits/stdc++.h>
#define all(x) (x).begin(), (x).end()
#define endl "\n"
#define ends "
#define fio() ios_base::sync_with_stdio(0); cin.tie(0)

using namespace std;

typedef long long ll;
int a[202020];
int n, k;

struct trie {
    int val;
    trie* a[2];
    trie() : val(0) { memset(a, 0, sizeof(a)); }
    void insert(int v, int idx) {
        if (idx == -1) return;
        if (v & 1) a[1] = new trie();
        else a[0] = new trie();
        a[v >> 1] -> insert(v >> 1, idx - 1);
    }
    int query(int v, int idx) {
        if (idx == -1) return val;
        if (v & 1) return a[1] ? a[1]->query(v >> 1, idx - 1) : 0;
        else return a[0] ? a[0]->query(v >> 1, idx - 1) : 0;
    }
};
```

```

val++;
if (idx == -1) return;
if (!a[0]) a[0] = new trie();
if (!a[1]) a[1] = new trie();
int t = v & (1 << idx) ? 1 : 0;
a[t]->insert(v, idx - 1);
}
int query(int v, int idx) {
if (idx == -1) return val;
int t = v & (1 << idx) ? 1 : 0;
if (!a[0]) a[0] = new trie();
if (!a[1]) a[1] = new trie();
if (k & (1 << idx))
return a[t]->val + a[1 - t]->query(v, idx - 1);
else
return a[t]->query(v, idx - 1);
}
};

int main() {
fio();
cin >> n >> k;
for (int i = 0; i < n; i++) cin >> a[i];
trie root;
for (int i = 0; i < n; i++)
root.insert(a[i], 30);

ll s = 0;
for (int i = 0; i < n; i++) {
s += root.query(a[i], 30) - 1;
}
cout << s;
return 0;
}

```