**Maximum XOR With an Element From Array**

#include <bits/stdc++.h>

class TrieNode {

public:

int data;

TrieNode\* child[2];

TrieNode(int data) {

this->data = data;

for (int i = 0; i < 2; i++) {

child[i] = NULL;

}

}

};

class Trie {

public:

TrieNode\* root;

Trie() {

root = new TrieNode('\0');

}

void insert(int num) {

TrieNode\* prev = root;

for (int i = 31; i >= 0; i--) {

int bit = (num >> i) & 1;

TrieNode\* child;

if (prev->child[bit] == NULL) {

child = new TrieNode(bit);

prev->child[bit] = child;

}

else {

child = prev->child[bit];

}

prev = child;

}

}

int findMaxXor(int num) {

int val = 0;

TrieNode\* prev = root;

for (int i = 31; i >= 0; i--) {

int bit = (num >> i) & 1;

TrieNode\* child;

if (bit & 1) {

if (prev->child[0]) {

child = prev->child[0];

val = val | (1 << i);

}

else {

child = prev->child[1];

}

}

else {

if (prev->child[1]) {

child = prev->child[1];

val = val | (1 << i);

}

else {

child = prev->child[0];

}

}

prev = child;

}

return val;

}

};

vector<int> maxXorQueries(vector<int>& arr, vector<vector<int>>& queries) {

Trie trie;

sort(arr.begin(), arr.end());

vector<pair<int, pair<int, int>>> offlineQuery;

int q = queries.size();

for (int i = 0; i < q; i++) {

offlineQuery.push\_back({ queries[i][1], { queries[i][0], i } });

}

sort(offlineQuery.begin(), offlineQuery.end());

vector<int> ans(q, 0);

int idx = 0;

for (int i = 0; i < offlineQuery.size(); i++) {

int ai = offlineQuery[i].first;

int xi = offlineQuery[i].second.first;

int qindex = offlineQuery[i].second.second;

while (idx < arr.size() && arr[idx] <= ai) {

trie.insert(arr[idx]);

idx++;

}

if (idx == 0) {

ans[qindex] = -1;

}

else {

ans[qindex] = trie.findMaxXor(xi);

}

}

return ans;

}