

ABC 365 - E 解説

問題文

https://atcoder.jp/contests/abc365/tasks/abc365_e

前提知識

高校数学, 排他的論理和, 累積和, Python もしくは Java の文法

解説

[方針] 本問題は XOR の性質を利用する問題であり, XOR においては, 各ビットを独立して計算できることを利用する. すなわち, 本問題の数式を 2 進数表記で書き表すことから考えると良い.

A_i を 2 進数表記すると,

$$A_i = \sum_{k=1}^{30} 2^{k-1} B_{i,k}$$

となる. ここで, $B_{i,k}$ は 0 か 1 である. A_i の最大値は問題文の制約より 10^8 なので $10^8 < 2^{30}$ であることを利用すると, A_i の 2 進数表記は 30 桁以内であることが分かる. そのため k の上限値は 30 としている.

2 進数表記の具体例として, $A_1 = 13$ の場合, $A_1 = 1101_{(2)}$ となるので, $B_{1,1} = 1, B_{1,2} = 0, B_{1,3} = 1, B_{1,4} = 1$ とすると,

$$A_1 = 2^0 B_{1,1} + 2^1 B_{1,2} + 2^2 B_{1,3} + 2^3 B_{1,4}$$

となる ($k \geq 5$ は明らかに $B_{1,k} = 0$ なので省略している).

本問題は、次の式を計算する問題であった。

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^N (A_i \oplus A_{i+1} \oplus \cdots \oplus A_j)$$

先ほどの 2 進数表記を用いると、XOR がビットごとに独立して計算できることから、

$$\sum_{k=1}^{30} 2^{k-1} \left(\sum_{i=1}^{N-1} \sum_{j=i+1}^N (B_{i,k} \oplus B_{i+1,k} \oplus \cdots \oplus B_{j,k}) \right)$$

となる。これ以降では、大きな () で囲まれた部分の計算について考える。

累積和のように、累積 XOR を導入すると、 $B_{i,k} \oplus B_{i+1,k} \oplus \cdots \oplus B_{j,k}$ を簡潔に書くことができ、計算の高速化も可能である。これが実際にそうであることを確認していく。

累積和で計算のための配列を用意し、初期値を 0 とするように、累積 XOR の配列 C を用意し、初期値を 0 とする。形式的には次のようになる。

$$C_0 = 0, C_i = B_1 \oplus B_2 \oplus \cdots \oplus B_i$$

もし、 C が累積和の場合は形式的には次のようになる。

$$C_0 = 0, C_i = B_1 + B_2 + \cdots + B_i$$

このように、 $+$ の部分を \oplus に置き換えることで、累積 XOR を考えることができる。

累積和では、 $C_j - C_{i-1} = B_i + B_{i+1} + \cdots + B_j$ であった。累積 XOR においても同様のことが成り立つ。すなわち、 $C_j \oplus C_{i-1} = B_i \oplus B_{i+1} \oplus \cdots \oplus B_j$ である。このことが成り立つことを示す。

まず、 $C_j = B_1 \oplus B_2 \oplus \cdots \oplus B_{i-1} \oplus B_i \oplus \cdots \oplus B_j$ が言える。

また、 $C_{i-1} = B_1 \oplus B_2 \oplus \cdots \oplus B_{i-1}$ が言える。

$$C_j \oplus C_{i-1} = (\textcolor{red}{B_1} \oplus \textcolor{red}{B_2} \oplus \cdots \oplus \textcolor{red}{B_{i-1}} \oplus B_i \oplus \cdots \oplus B_j) \oplus (\textcolor{red}{B_1} \oplus \textcolor{red}{B_2} \oplus \cdots \oplus \textcolor{red}{B_{i-1}})$$

XOR においては、 $A \oplus A = 0$ であることから、赤色の部分が消える。

すなわち、 $C_j \oplus C_{i-1} = B_i \oplus B_{i+1} \oplus \cdots \oplus B_j$ である。よって、成り立つことが示された。

引き算は交換法則が成り立たないが、XOR は交換法則が成り立つので、 $C_j \oplus C_{i-1} = C_{i-1} \oplus C_j$ である。

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^N (B_{i,k} \oplus B_{i+1,k} \oplus \cdots \oplus B_{j,k})$$

先述した上記の式に累積 XOR を導入すると次のように式変形できる．以下では，数式を簡潔にするためにちょっとした計算を行う．ここはあまり本質的ではないので，興味がない場合は読み飛ばしても良い．

$$\sum_{i=1}^{N-1} \sum_{j=i+1}^N (C_{i-1,k} \oplus C_{j,k})$$

一番左のシグマを $i = 1$ から $i = 0$ にすると，

$$\sum_{i=0}^{N-2} \sum_{j=i+2}^N (C_{i,k} \oplus C_{j,k})$$

となる．左から二番目のシグマを $j = i + 2$ から $j = i + 1$ にすると，

$$\sum_{i=0}^{N-2} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k}) - \sum_{i=0}^{N-2} (C_{i,k} \oplus C_{i+1,k})$$

となる．一番左のシグマを $N - 2$ から $N - 1$ にすると，

$$\sum_{i=0}^{N-1} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k}) - \sum_{i=0}^{N-2} (C_{i,k} \oplus C_{i+1,k}) - (C_{N-1,k} \oplus C_{N,k})$$

初項以外は，次のように一つの項にまとめることができる．

$$\sum_{i=0}^{N-1} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k}) - \sum_{i=0}^{N-1} (C_{i,k} \oplus C_{i+1,k})$$

ここで， $C_{i,k} \oplus C_{i+1,k} = B_{i+1,k}$ であるから，

$$\sum_{i=0}^{N-1} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k}) - \sum_{i=0}^{N-1} B_{i+1,k}$$

となる．一番右のシグマを $i = 0$ から $i = 1$ にすると，

$$\sum_{i=0}^{N-1} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k}) - \sum_{i=1}^N B_{i,k}$$

となる。

この式は、大きな $()$ で囲まれた部分の計算であったので、その外側を付け加えると、

$$\sum_{k=1}^{30} 2^{k-1} \left(\sum_{i=0}^{N-1} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k}) - \sum_{i=1}^N B_{i,k} \right)$$

となる。分配法則を用いて、この式を展開すると、

$$\sum_{k=1}^{30} 2^{k-1} \left(\sum_{i=0}^{N-1} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k}) \right) - \sum_{k=1}^{30} 2^{k-1} \left(\sum_{i=1}^N B_{i,k} \right)$$

となる。まず、右側の式について考える。

$$\sum_{k=1}^{30} 2^{k-1} \left(\sum_{i=1}^N B_{i,k} \right)$$

これは、 A_i の総和を 2 進数表記で考えたものであるから、

$$\sum_{i=1}^N A_i = \sum_{k=1}^{30} 2^{k-1} \left(\sum_{i=1}^N B_{i,k} \right)$$

である。したがって、式は次のようになる。

$$\sum_{k=1}^{30} 2^{k-1} \left(\sum_{i=0}^{N-1} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k}) \right) - \sum_{i=1}^N A_i$$

次に、左側の式について考える。実はこの式は真面目に計算するよりも、素早く求める方法がある。 C は累積 XOR であるから、 $C_i \in \{0, 1\}$ である。それから、以下の部分を見ると、 $i < j$ であるような (i, j) のペアを考えていることが分かる。

$$\sum_{i=0}^{N-1} \sum_{j=i+1}^N$$

$0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1, 1 \oplus 1 = 0$ であるので、 $C_{i,k} \oplus C_{j,k}$ の値が 1 となるには $C_{i,k} \neq C_{j,k}$ である必要がある。つまり、以下の式を計算することは、 $C_{i,j} \neq C_{j,k}$ であるペアの個数を求めることに等しい。そのようなペアの個数を求めることは、 C 中にある 0 の個数と 1 の個数の積に等しいことも分かる。

$$\sum_{i=0}^{N-1} \sum_{j=i+1}^N (C_{i,k} \oplus C_{j,k})$$

よって、 C 中にある 0 の個数と 1 の個数を求めることで、先述した左側の式を求めることができ、次に A_i の総和を引くことで、答えを求めることができる。あとはこれをプログラムすれば良い。

実装:Python

ソースコード 1 に Python による解法を示す。このコードは、下記 URL にある私の提出コードから確認可能である。

<https://atcoder.jp/contests/abc365/submissions/56340290>

ソースコード 1 Python のコード

```
1 n = int(input())
2 a = list(map(int, input().split()))
3
4 bit_SIZE = 30
5
6 # 1 つ目の式
7 def f(bit_idx):
8     # cnt[0] := 0 の個数
9     # cnt[1] := 1 の個数
10    # 初期値0 についてもカウントするのでcnt[0] = 1 としている
11    cnt = [1, 0]
12
13    c_last_val = 0
14    for i in range(n):
15        if a[i] >> bit_idx & 1:
16            c_last_val ^= 1
17        else:
18            c_last_val ^= 0
19            cnt[c_last_val] += 1
20    return cnt[0] * cnt[1] << bit_idx
21
22 ans = 0
23 for bit_idx in range(bit_SIZE + 1):
24     ans += f(bit_idx) # 1 つ目の式
25 ans -= sum(a) # 2 つ目の式
26
```

```
27 print(ans)
```

実装:Java

ソースコード 2 に Java による解法を示す. このコードは, 下記 URL にある私の提出コードから確認可能である.

<https://atcoder.jp/contests/abc365/submissions/56340269>

ソースコード 2 Java のコード

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         int n = scanner.nextInt();
7         int[] a = new int[n];
8         for (int i = 0; i < n; i++) {
9             a[i] = scanner.nextInt();
10        }
11        int bit_SIZE = 30;
12
13        long ans = 0;
14        for (int bit_idx = 0; bit_idx <= bit_SIZE; bit_idx++) {
15            ans += f(bit_idx, a, n); // 1 つ目の式
16        }
17        long sum_a = 0;
18        for (int i = 0; i < n; i++) {
19            sum_a += a[i];
20        }
21        ans -= sum_a; // 2 つ目の式
22
23        System.out.println(ans);
24    }
25
26    // 1 つ目の式
27    public static long f(int bit_idx, int a[], int n) {
28        // cnt[0] := 0 の個数
```

```
29         // cnt[1] := 1 の個数
30         // 初期値0 についてもカウントするのでcnt[0] = 1 としている
31         int[] cnt = {1, 0};
32
33         int c_last_val = 0;
34         for (int i = 0; i < n; i++) {
35             if ((a[i] >> bit_idx & 1) != 0) {
36                 c_last_val ^= 1;
37             } else {
38                 c_last_val ^= 0;
39             }
40             cnt[c_last_val]++;
41         }
42         return (long) cnt[0] * cnt[1] << bit_idx;
43     }
44 }
```
