Editorial Bitwise Array

Dasar Pemrograman Final 2022

Algoritma/Pengetahuan yang Dibutuhkan

Bit operation dan array

Pendekatan/Observasi

Perhatikan bahwa ini hasil dari bilangan membentuk sebuah fungsi $f(i,j)$ dimana i adalah posisi bit dan j adalah kondisi awal bit (0 atau 1). Sehingga untuk membuat bilangan 32-bit utuh. Bit-bitnya dapat disusun seperti berikut:

$$bit_0 = f(0,j)$$
$$bit_1 = f(1,j)$$
$$bit_2 = f(2,j)$$
$$.$$
$$.$$
$$.$$
$$bit_k = f(k,j)$$

Berarti, jika kita memiliki angka misalkan 5 (101). Bitnya akan tersusun seperti berikut:

$$bit_0 = f(0,1)$$
$$bit_1 = f(1,0)$$
$$bit_2 = f(2,1)$$

Bagaimana cara mendapatkan nilai $f(i,j)$? Kita dapat menggunakan *array* bilangan bulat berukuran 2 yang diinisiasikan seperti berikut:

$bitAwal_0 =$ 0 (Semua bit-nya bernilai 0)

$bitAwal_1 =$ 2147483647 (Semua bit-nya bernilai 1)

Nilai $f(i,j)$ dapat diambil dari $bitAwal_j$ **pada posisi bit ke-*i* saja**. Jika j bernilai 0, maka ambil dari $bitAwal_0$ **posisi ke-*i***. Jika j bernilai 1, maka ambil dari $bitAwal_1$ **posisi ke-*i***.

Kemudian bagaimana cara menghandle *query* tipe pertama? Mudah saja, misalkan kita perlu melakukan operasi XOR dengan angka 15, maka lakukan XOR pada masing-masing $bitAwal_0$ dan $bitAwal_1$ dengan 15 maka *array* tersebut akan memberikan jawaban yang valid. Lakukan hal serupa untuk AND dan OR.

Kompleksitas waktu: $O(Q_2 \times N \times log(A_i) + Q_1)$

Kompleksitas memori: $O(N)$

Follow IG @zydhanlinnar11 kalo gemes sama soal-nya. (~~Promosi dulu lah ya~~)

Source Code

```c
// Bitwise Array - Zydhan Linnar Putra (05111940000118)
#include <stdio.h>
const int MAX_BITS = 30;

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

    int Q;
    scanf("%d", &Q);
    int bitAwal[2];
    bitAwal[0] = 0; // Semua bitnya 0
    bitAwal[1] = 2147483647; // Semua bitnya 1

    while(Q--) {
        int type;
        scanf("%d", &type);
        if (type == 1) {
            char t[5]; int x;
            scanf("%s %d", t, &x);
            // Lakukan operasi sesuai command pada bitAwal[0] dan bitAwal[1]
            for(int j=0; j<2; j++) {
                if (t[0] == 'X') bitAwal[j] ^= x;
                else if(t[0] == 'O') bitAwal[j] |= x;
                else bitAwal[j] &= x;
            }

            // Biar nestingnya ga terlalu dalem buat type kedua
            // Mending pakai continue daripada else
            continue;
        }

        for(i=0; i<N; i++) {
            // Taruh variabel elem, supaya A[i] tidak berubah ketika dilakukan operasi shift
            int elem = A[i];
            // Variabel printed menampung hasil akhir bilangan yang dibentuk dari bit-bit terkait
            int printed = 0;
            for(int j=0; j<MAX_BITS; j++) {
                // Ambil bit paling kanan
                int rightmostBit = elem & 1;
                // Kalau bit ke-j 0, ambil hasil dari bitAwal[0], taruh variabel bit.
                // Kalau bit ke-j 1, ambil hasil dari bitAwal[1], taruh variabel bit.
                int bit = (bitAwal[rightmostBit] >> j) & 1;
                // Masukkan hasil bit di atas ke posisi ke-j pada variabel printed
                printed |= (bit << j);
                // Geser bitnya ke kanan satu langkah
                elem >>= 1;
            }
            printf("%d%c", printed, (i == N - 1 ? '\n' : ' '));
        }
    }
}
```