Assalammualaikum

Mohammad Hanif Furqan Aufa Putra

5025221161

PRE-TEST PBO C

BT69-BitPlay69

Mencari nilai K terkecil dari operasi N $\oplus$ K > M, dengan N dan M sebagai bilangan bulat dan $\oplus$ adalah Operator XOR Bitwise. Selain operasi tersebut, pada baris pertama berisi satu bilangan bulat T dengan batasan $1 \leq T \leq 100$ sebagai Jumlah kasus uji.

Tujuan dari program ini adalah untuk menemukan nilai terkecil K sehingga (N XOR K) lebih besar dari M, di mana XOR adalah operasi bitwise XOR. Hal ini dilakukan dengan membandingkan masing-masing bit dari N dan M dan membangun K berdasarkan perbandingan tersebut.

Gambaran sederhana dari luar program yang akan dibuat:

Masukkan jumlah kasus uji (T) dari user.

Lalu untuk setiap kasus uji (diperlukan loop dengan input jumlah kasus uji):

Baca dua buah angka bulat N dan M.

Cetak hasil akhir (K) untuk kasus uji ini.

Melalui penjelasan dan penyelesaian saya sebelumnya

“Untuk mencari nilai K, maka harus dibuatkan sebuah fungsi terlebih dahulu yang bisa menghitung operasi bilangan tersebut dengan menggunakan “^”. Setelah mencoba beberapa cara, saya membuat fungsi seperti dibawah (n^k) <= m, maka k akan di increment dan mengulang dengan adanya while loop”

Saya menemukan kesalahan pada kode saya sebelumnya, setelah di inspeksi lebih lanjut, jika dilihat dari code yang saya buat saya menemukan bahwa saya terlalu banyak menggunakan variable dan tidak hanya itu, setelah saya cermati pada kode yang telah diberikan bapak tadi, pendekatan iteraritve yang saya lakukan pada code saya lebih lambat dibandingkan pendekatan operasi bitwise yang sebagaimana disebutkan sebagai salah satu keunggulan Bahasa c, dan nantinya akan sangat berguna jika seorang programmer sudah menguasai bitwise operation.

Lalu Kembali lagi pada penjelasan code yang diberikan bapak, Operasi XOR (^):XOR adalah operasi bitwise yang membandingkan setiap bit dari dua angka. Jika bit pada posisi yang berbeda, hasilnya adalah 1; jika sama, hasilnya adalah 0. Lalu user akan memiliki beberapa set N dan M untuk diselesaikan. Setiap set adalah kasus uji yang

terpisah. Proses untuk Setiap Kasus Uji Ambil dua angka, N dan M, sebagai masukan. Lalu tingkatkan M dengan increment (M++). Alasan untuk meingkatkan `M` dengan increment (yaitu, `M++`) adalah untuk memastikan bahwa kita menemukan `K` terkecil sehingga `(N ^ K) > M`.

Mengapa kenaikan ini diperlukan adalah:

1. Untuk menemukan sebuah bilangan `K` untuk di-XOR-kan dengan `N` yang sudah ditentukan sehingga hasilnya lebih besar dari `M`. Dengan kata lain, kita ingin mencari `K` yang memenuhi `(N ^ K) > M`.

2. Untuk mencapai hal ini, kode yang Anda berikan melakukan perbandingan bitwise antara representasi biner `N` dan `M`. Kode ini mulai membandingkan bit dari bit yang paling kecil (bit paling kanan) hingga bit yang paling besar (bit paling kiri).

3. Kasus Tepi, kasus ketika membandingkan bit, ada situasi tertentu yang perlu dipertimbangkan:

- Jika sebuah bit di `N` adalah 0 dan bit yang sesuai di `M` adalah 1, kita ingin mengatur bit yang sesuai di `K` menjadi 1. Hal ini memastikan bahwa ketika kita melakukan XOR `N` dan `K`, kita akan mendapatkan nilai 1 pada posisi bit tersebut, sehingga hasilnya lebih besar dari `M`.

- Jika sebuah bit di `N` adalah 1 dan bit yang sesuai di `M` adalah 0, kita harus mereset bit di `K` menjadi 0. Ini memastikan bahwa ketika kita XOR `N` dan `K`, kita tidak mendapatkan nilai 1 pada posisi bit tersebut, sehingga hasilnya tidak akan lebih besar dari `M`.

4. Kasus $M = 0$, Kenaikan `M` sebesar 1 memastikan bahwa kita menangani kasus tepi tertentu. Jika `M` pada awalnya adalah 0 dan kita tidak menambahnya, algoritme akan mengatur semua bit dalam `K` menjadi 1 (untuk membuat `(N ^ K)` sebesar mungkin), yang mungkin bukan merupakan `K` terkecil yang memenuhi kondisi `(N ^ K) > M`. Dengan menambah `M` menjadi 1, kita menjamin bahwa `K` harus paling tidak 1 untuk memenuhi kondisi tersebut, membuatnya menjadi nilai terkecil yang mungkin.

Untuk mudahnya, dengan menambah `M` dengan 1 memastikan bahwa kita menemukan `K` terkecil yang memenuhi kondisi `(N ^ K) > M` sambil menangani kasus tepi dengan benar dan mengikuti logika perbandingan bitwise yang digunakan dalam kode.

```c
#include <stdio.h>
typedef long long ll;

int main()
{
    int cases;
    scanf("%d",&cases);

    while (cases--)
    {
        ll n, m, res = 0, b = 1;
        scanf("%lld %lld", &n, &m);
        m++;
        while (n||m)
        {
            ll N = n & 1, M = m & 1;
            if (!N && M) res |= b;
            else if (N && !M) res = 0;
            b <<= 1; n >>= 1; m >>= 1;

        }
        printf("%lld\n", res);

    }
    return 0;
}
```

Dari code bapak diatas, dideklarasikan secara typedef long long menjadi ll agar lebih aman. Lalu setelah input baris ke 1 yang merupakan jumlah kasus untuk loop dan untuk setiap baris mulai baris ke 2 akan dilakukan increment pada m agar memastikan n^k > m akan mulai membandingkan bit-bit N dan M dari kanan ke kiri (bit yang paling kecil ke bit yang paling besar).

- Jika bit di N adalah 0 dan bit di M adalah 1, setel bit yang sesuai di K ke 1.

- Jika bit di N adalah 1 dan bit di M adalah 0, setel K ke 0 (setel ulang K seperti penjelasan diatas).

- Pindah ke bit berikutnya di sebelah kiri.

- Ulangi proses ini sampai semua bit N dan M telah dibandingkan.

- Nilai akhir K akan memenuhi kondisi (N ^ K) > M, dan nilai K untuk setiap kasus uji.

Example

Input:

```
4                  <- Loop untuk jumlah kasus uji
3 5                <- Baris-baris selnjutnya untuk angka yang dimasukkan rumus
3 2
69 696
696 96
```

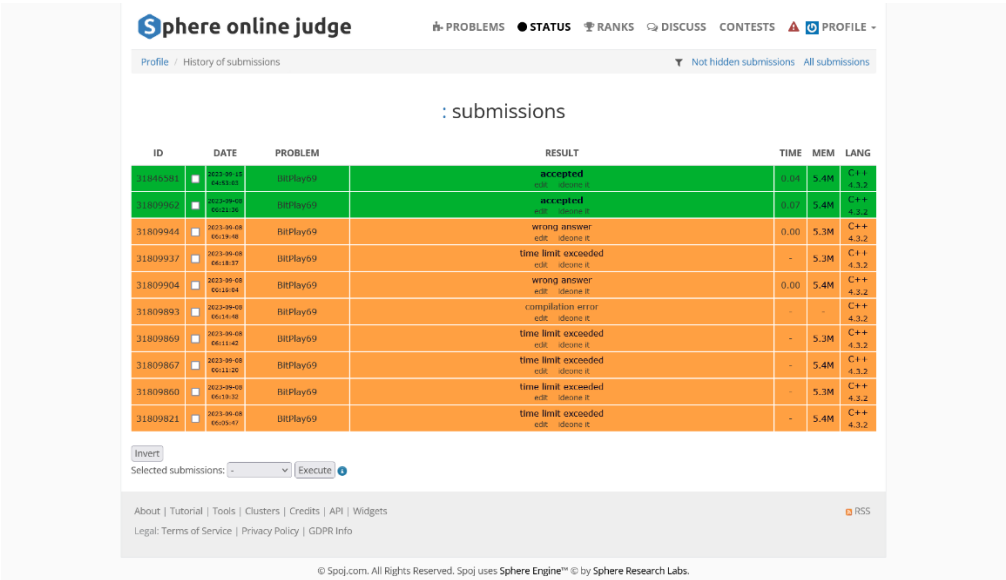
Output:

```
4                  <- K yang diperoleh setelah menjalan kan rumus pada fungsi
0
640
0
```

Dengan menjalan program tersebut diperoleh hasil seperti dibawah ini

```
PS F:\Coding\PBO-2023> cd "f:\Coding\PBO-2023\" ; if ($?) { g++ bt69lowest.cpp -o bt69lowest } ; if ($?) { .\bt69lowest }
4
3 5
4
3 2
0
69 696
640
696 96
0
PS F:\Coding\PBO-2023>
```

Dan dibawah bukti sudah Verdict Accept perbaikan



Terima Kasih telah membaca, Waalaikumsalam wr wb.