Universidad Mayor de San Andrés
Facultad de Ciencias Puras y Naturales
Carrera de Informática
DIV-2-2020
UMSA

Grover Osvaldo Rodriguez Apaza

26 de abril de 2022

1. Juego de Números

link

1.1. Descripción de la solución

Cada vez que se eliminaba dos monedas de la lista debia cumplir la siguiente ecuación:

$$f(A_i, A_j) = A_i * A_j - (A_i - 1) - (A_j - 1)$$

Para poder maximizar el ultimo elemento de una lista, se escribira la ecuacion de la siguiente manera.

$$\begin{aligned} f(A_i, A_j) &= A_i * A_j - (A_i - 1) - (A_j - 1) \\ f(A_i, A_j) &= A_i * A_j - A_i + 1 - A_j + 1 \\ f(A_i, A_j) &= A_i * A_j - A_i - A_j + 1 + 1 \\ f(A_i, A_j) &= A_i(A_j - 1) - (A_j - 1) + 1 \\ f(A_i, A_j) &= (A_j - 1)(A_i - 1) + 1 \end{aligned} \tag{1}$$

Finalmente si aplicamos la nueva expresion de la ecuación tendremos lo siguiente.

Sea la lista $A = a_1, a_2, a_3, a_4, \ldots, a_{n-2}, a_{n-1}, a_n$ con N elementos.

Cantidad de operaciones: 1
Sacando los elementos a_1, a_3 y aplicando la operación respectiva, los nuevos elementos de la lista son:

$$A = (a_1 - 1)(a_3 - 1) + 1, a_2, a_4, \dots, a_{n-2}, a_{n-1}, a_n$$

Cantidad de operaciones: 2
Sacando los elementos a_2, a_4 y aplicando la operación respectiva, los nuevos elementos de la lista son:

$$A = (a_1 - 1)(a_3 - 1) + 1, (a_2 - 1)(a_4 - 1) + 1, \dots, a_{n-2}, a_{n-1}, a_n$$

Cantidad de operaciones: 3
Sacando los dos primeros elementos de la lista anterior y apicando la operacion se tiene:

$$A = (a_1 - 1)(a_3 - 1)(a_2 - 1)(a_4 - 1) + 1, \dots, a_{n-2}, a_{n-1}, a_n$$

$\vdots$

Cantidad de operaciones: $n - 3$
Sacando los dos primeros elementos de la lista anterior y aplicando la operación se tiene:

$$A = (a_1 - 1)(a_3 - 1)(a_2 - 1)(a_4 - 1) * \cdots * (a_{n-2} - 1) + 1, a_{n-1}, a_n$$

Cantidad de operaciones: $n - 2$
Sacando los dos primeros elementos de la lista anterior y aplicando la operación se tiene:

$$A = (a_1 - 1)(a_3 - 1)(a_2 - 1)(a_4 - 1) * \cdots * (a_{n-2} - 1)(a_{n-1} - 1) + 1, a_n$$

Realizado $N - 1$ operaciones la lista tendrá solo una moneda y la resultante sera:

$$A = (a_1 - 1)(a_2 - 1)(a_3 - 1)(a_4 - 1) * \cdots * (a_{n-2} - 1)(a_{n-1} - 1)(a_n - 1) + 1$$

Finalmente se puede apreciar que la función aplicada a dos elementos de la lista es invariante a la transformación.

1.2. Codificación

Una moneda de la lista A, esta $1 \leq a_i \leq 10^{15}$ y como puede haber hasta 10^6 monedas, al momento de multiplicar dos elementos para evitar el desbordamiento use __int128 o multiplicacion rapida.

Código utilizando Multiplicación rápida
Código

Código utilizando __int128
Código

1.3. Multiplicación Rápida

1.4. Explicación

Para multiplicar 2 números $ans = a * b$, se puede expresar en términos de sumas de la siguiente manera:

$$ans = \underbrace{a + a + a + \cdots + a}_{b-veces}$$

el algoritmo trivial que revuelve dicho problema es el siguiente: "'c++ ll ans = 0; for(int i = 1; i ¡= b; i++) ans += a; "' pero necesitamos encontrar una manera mas eficiente de multiplicar dos números. Podemos expresar la ecuación anterior de la siguiente manera:

$$ans = \underbrace{a + a + a + \cdots + a}_{\frac{b}{2}-veces} + \underbrace{a + a + a + \cdots + a}_{\frac{b}{2}-veces}$$

se puede calcular $\frac{b}{2}$ y multiplicarle por 2 para obtener ans.

$$b = \frac{b}{2} + \frac{b}{2} = 2 * \frac{b}{2}$$

similarmente podemos calcular $\frac{b}{2}$ de la forma:

$$\frac{b}{2} = \frac{b}{4} + \frac{b}{4} = 2 * \frac{b}{4}$$

de igual manera se puede calcular $\frac{b}{4}$ de la forma:

$$\frac{b}{4} = \frac{b}{8} + \frac{b}{8} = 2 * \frac{b}{8}$$

$\vdots$

El valor $\frac{b}{2^{k-1}}$ se puede calcular:

$$\frac{b}{2^{k-1}} = \frac{b}{2^k} + \frac{b}{2^k} = 2 * \frac{b}{2^k}$$

El valor $\frac{b}{2^k}$ se puede calcular:

$$\frac{b}{2^k} = \frac{b}{2^{k+1}} + \frac{b}{2^{k+1}} = 2 * \frac{b}{2^{k+1}}$$

cuando dividimos entre $\frac{b}{2}$, se tiene casos:

- Caso 1: b es par Cuando se divide un número par entre 2, no existe residuo.
- Caso 2: b es impar
  Cuando se divide un número impar entre 2, el residuo siempre es 1, entonces estariamos perdiendo un valor de a, para evitar ese caso, cada vez que b es impar, se suma el valor de a a la respuesta.
- Cuando $b = 0$, el resultado siempre sera 0, ya que $a * 0 = 0$.

Finalmente definamos la función recursiva $fast(a, b)$, donde el resultado de dicha función es el resultado de multiplicar $a * b$.

$$fast(a,b) = \begin{cases} b = 0 & 0 \quad (Caso\ base) \\ b\ es\ impar & 2 * fast(a, \frac{b}{2}) + a \\ b\ es\ par & 2 * fast(a, \frac{b}{2}) \end{cases}$$

1.5. Codificación Recursiva

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll fast(ll a, ll b){
    if(b == 0)
        return 0;
    return 2 * fast(a, b / 2) + (b & 1 ? a : 0);
}
```

1.6. Codificación Iterativa

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
```

```
ll fast(ll a, ll b){
    ll ans = 0;
    while(b){
        if(b & 1)
            ans = (ans + a);
        b >>= 1;
        a += a;
    }
    return ans;
}
```

Generalmentte estos resultados son muy grandes para almacenarlos, por tal motivo se hallar el valor $\%$ *mod*

1.7. Codificación Recursiva

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod = 1e9 + 7;
ll fast(ll a, ll b){
    if(b == 0)
        return 0;
    if(b % 2 == 0)
        return (2 % mod * fast(a, b / 2) % mod) % mod;
    return ((2 % mod * fast(a, b / 2) % mod ) % mod  + a % mod) % mod;
}
```

1.8. Codificación Iterativa

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll mod = 1e9 + 7;
ll fast1(ll a, ll b){
    ll ans = 0;
    while(b > 0){
        if(b & 1)
            ans = (ans + a) % mod;
        b = b / 2;
        a = (a + a) % mod;
    }
    return ans;
}
```

1.9. Analisis de complejidad.

La complejidad va a depender del parametro b, de la función.

- Cuando b es par: b se divide por 2.
- cuando b es impar: b se divide por 2.

Mientras b sea mayor o igual 1, en cada paso de la función recursiva se divide por 2.

De esa manera podemos expresar la siguiente ecuación.

$$\frac{b}{2^k} = 1$$

donde k es la cantidad de veces que b se dividira por 2.
Aplicando logaritmos a ambos lados se obtiene:

$$k = log_2 b$$

Complejidad: $O(log \quad b)$