Soluciones entrenos OIE

Horses

A Mansur le encanta criar caballos como sus antepasados hacían. Pero este no ha sido siempre el caso, hace N años solo tenía un caballo y soñaba con conseguir mucho dinero.

Numeramos los años de 0 a $N-1$ en orden cronológico (el año $n-1$ es el más reciente). El clima de cada año influcenciaba el crecimiento de sus caballos. Para cada año i, Mansur recuerda un coeficiente de crecimiento entero positivo X_i. Si empezó el año con i con h caballos; terminó el año con $h * X_i$ caballos.

Los caballos solo poañon venderse al final de cada año. Para cada año i, Mansur recuerda un entero positivo Y_i: el precio al que poaño vender cada caballo al final del año i. Al final de cada año, era posible vender una cantidad arbitraria de caballos, todos al mismo precio Y_i.

Mansur se pregunta cuál es la mayor cantidad de dinero que podría tener ahora si hubiese elegido los mejores momentos para vender sus caballos durante los N años. Has tenido el honor de ser invitado a las vacaciones de Mansur, y te ha pedido a ti una respuesta a esta pregunta.

La memoria de Mansur mejora durante la tarde, así que te da una M actualizaciones. Cada actualización cambiará o bien uno de los valores X_i o uno de los valores Y_i. Tras cada actualización te pregunta la mayor cantidad de dinero que podría haber ganado vendiendo sus caballos. Las actualizaciones son acumulativas: cada una de tus respuestas debe tener en cuenta todas las actualizaciones anteriores. Date cuenta de que un mismo X_i o Y_i puede ser actualizado múltiples veces.

Las respuestas pueden ser enormes, para evitar trabajar con números demasiado grandes solo se requiere que des las respuestas módulo $10^9 + 7$.

Ejemplo

Supongamos que hay $N = 3$ años, con la siguiente información:

	0	1	2
X	2	1	3
Y	3	4	1

En este caso, una de las soluciones óptimas es vender un caballo al final del año 0, y luego tres caballos al final del año 2. El proceso entero sería así:

1. Inicialmente, Mansur tiene un caballo.
2. Tras el año 0 tendrá $1 * X_0 = 2$ caballos.
3. Tras el año 1 tendrá $2 * X_1 = 2$ caballos.
4. Ahora puede vender esos caballos. El beneficio total será de $2 * Y_1 = 8$.

	0	1	2
X	2	1	3
Y	3	2	1

Supongamos entonces que hay $M = 1$ actualizaciones: cambiando Y_1 a 2. Tras esta actualización tendremos:

En este caso, una de las soluciones óptimas es vender un caballos tras el año 0 y luego tres caballos tras el año 2. El proceso entero será:

1. Inicialmente, Mansur tiene un caballo.
2. Tras el año 0 tendrá $1 * X_0 = 2$ caballos.
3. Ahora vende uno de esos caballos por $Y_0 = 3$, y le queda un caballo.
4. Tras el año 1 tendrá $1 * X_1 = 1$ caballo.
5. Tras el año 2 tendrá $1 * X_2 = 3$ caballos.
6. Ahora puede vender esos caballos por $3 * Y_2 = 3$. El beneficio total será de $3 + 3 = 6$.

Tarea

Se te dan N, X, Y y la lista de actualizaciones, antes de la primera actualización y después de cada actualización, calcula la máxima cantidad de dinero que Mansur podría haber conseguido de sus caballos, módulo $10^9 + 7$. Implementa las funciones *init*, *updateX*, y *updateY*.

int init(int N**, int** $X[]$**, int** $Y[]$**)**

1. The grader will call this function first and exactly once.
2. N: el número de años.
3. X: un array de longitud N. Para $0 \leq i \leq N - 1$, X_1 da el coeficiente de crecimiento del año i.
4. Y: un array de longitud N. Para $0 \leq i \leq N - 1$, XY_1 da el precio de un caballo tras el año i.
5. Tanto X como Y especifican los valores iniciales dados por Mansur (antes de las actualizacioens).
6. Cuando *init* termine, los arrays X e Y permanecerán válidos y puedes modificar su contenido como desees.
7. La función debe devolver la máxima cantidad de dinero que Mansur podría conseguir con esos valores iniciales de X e Y módulo $10^9 + 7$.

int updateX(int *pos*, **int** *val*)

1. *pos*: un entero en el rango $0, ..., N - 1$.
2. *val*: el nuevo valor de X_{pos}.
3. La función debe devolver la máxima cantidad de dinero que Mansur podría conseguir tras esta actualización módulo $10^9 + 7$.

int updateY(int *pos*, **int** *val*)

1. *pos*: un entero en el rango $0, ..., N-1$.
2. *val*: el nuevo valor de Y_{pos}.
3. La función debe devolver la máxima cantidad de dinero que Mansur podría conseguir tras esta actualización módulo $10^9 + 7$.

Puedes asumir que todos los valores iniciales, así como los valores de X_i y Y_i están entre 1 y 10^9 inclusive.

Tras llamar a *init*, el grader llamará a *updateX* y *updateY* un total de M veces entre las dos.

Subtareas

subtask	points	N	M	additional constraints
1	17	$1 \leq N \leq 10$	$M = 0$	$X_i, Y_i \leq 10$, $X_0 * X_1 * ... * X_{N-1} \leq 1000$
2	17	$1 \leq N \leq 1000$	$0 \leq M \leq 1000$	none
3	20	$1 \leq N \leq 500000$	$1 \leq M \leq 100000$	$X_i \geq 2$ and $val \geq 2$ for *init* and *updateX* respectively
4	23	$1 \leq N \leq 500000$	$1 \leq M \leq 10000$	none
5	23	$1 \leq N \leq 500000$	$1 \leq M \leq 100000$	none

Solución

Antes de empezar, este problema requiere conocer la estructura de datos segment tree, si no se conoce es recomendable mirarla antes de continuar.

En primer lugar observamos que como no hay límite en la cantidad de caballos que podemos vender cad año siempre existe una solución óptima en la que vendemos todos los caballos el mismo año, este es uno de los años d que maximizan el producto $X_0 * ... * X_d * Y_d$. No podemos calcular estos productos directamente porque los valores del vector X pueden ser tan altos como 10^9 así que no hay ningún tipo de variable que pueda almacenar números tan altos como los que podrían resultar de estos productos.

Para comparar dos años i, j es suficiente comparar Y_i con $X_{i+1} * ... * X_j * Yj$. Como los valores del vector Y están acotados entre 1 y 10^9 y los valores de X son mayores o iguales que 1 si tenemos $X_{i+1} * ... * X_j > 10^9$ podemos garantizar no solo que el año j es mejor que el año i, sino que es mejor que cualquier año anterior a i.

En uno de los casos de prueba no hay valores en X iguales a 1, en este caso el producto de 30 valores de X consecutivos es al menos $2^{30} > 10^9$, por tanto solo será necesario comprobar los últimos 30 años, cosa que podemos hacer de forma lineal para cada query, para encontrar el año óptimo para vender.

En el caso general, tenemos que resolver el problema de qué hacer cuando tengamos algunos valores de X iguales a 1. En este caso podemos formar intervalos desde l hasta r tales que $X_{l+1} = ... = X_r = 1$, ya que el año óptimo dentro de este intervalo es simplemente el año d que maximiza Y_d. Utilizaremos un segment tree para calcular rápido el máximo valor de Y en un intervalo y un set para guardar los índices l donde $X_l > 1$ que serán los extremos de la izquierda de los intervalos. Esto nos permite calcular el año óptimo dado que el set con los valores de l y el segment tree con los valores de Y esté actualizado en complejidad $O(\log(10^9) * \log(N)$ porque la cantidad de intervalos que tendremos que tratar es como máximo $\log(10^9)$ y para cada uno buscamos el valor de Y máximo, lo cual tarda $\log(N)$ por el segment tree.

Una vez tenemos el año óptimo d tenemos que calcular las ganancias, esto es el producto $X_0 * ... * X_d * Y_d$ en módulo $10^9 + 7$. Para hacer esto rápidamente y poder actualizar un valor de X cuando sea necesario usaremos de nuevo un segment tree.

Una vez diseñana la estrategia, debemos traducirla a código.

Los datos que tendremos que guardar en forma de variables globales serán n, los dos segment trees y el set con los valores de l. Utilizaremos por comodidad un vector con los valores de X para poder acceder a ellos sin tener que hacer una query en el segment tree.

Necesitaremos funciones para actualizar y hacer queries en cada segment tree, esto es parte de la implementación del segment tree y se da por conocida.

Utilizaremos también una función *solve* para volver a resolver el problema cada vez que actualizamos los datos que tenemos guardados, la llamaremos cada vez que tengamos que dar una respuesta.

Las funciones *init*, *updateX* y *updateY* simplemente actualizarán los datos como corresponda y llamarán a *solve*.

Código

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

const int MOD = 1000000007;

int n;
vector<ll> stY; //Segment tree del vector Y para tener el máximo en un rango.
vector<ll> stX; //Segment tree del vector X para tener el producto en un rango.
vector<ll> vX; //Guarda el vector X, podríamos usar solo el segment tree.
set<int> non1Set; //Guarda los opuestos de los índices i tales que vX[i] > 1 y el 0.
//Son los opuestos para poder usarlos de mayor a menor con el bucle for, incluímos el 0 por ser el último índice que queremos revisar.

void stXupdate(int idx, int x, int i, int l, int r){
    if (l == r)
        stX[i] = x;
    else{
        int avg = (l+r)/2;
        if (idx <= avg)
            stXupdate(idx, x, 2*i, l, avg);
        else
            stXupdate(idx, x, 2*i+1, avg+1, r);
        stX[i] = stX[2*i] * stX[2*i+1] % MOD;
    }
}

ll stXquery(int ql, int qr, int i, int l, int r){
    if (qr < l || r < ql)
        return 1;
    if (ql <= l && r <= qr)
        return stX[i];
    int avg = (l+r)/2;
    ll a = stXquery(ql, qr, 2*i, l, avg);
    ll b = stXquery(ql, qr, 2*i+1, avg+1, r);
    return a*b%MOD;
}
```

```cpp
void stYupdate(int idx, int y, int i, int l, int r){
    if (l == r)
        stY[i] = y;
    else{
        int avg = (l+r)/2;
        if (idx <= avg)
            stYupdate(idx, y, 2*i, l, avg);
        else
            stYupdate(idx, y, 2*i+1, avg+1, r);
        stY[i] = max(stY[2*i], stY[2*i+1]);
    }
}

ll stYquery(int ql, int qr, int i, int l, int r){
    if (qr < l || r < ql)
        return 0;
    if (ql <= l && r <= qr)
        return stY[i];
    int avg = (l+r)/2;
    ll a = stYquery(ql, qr, 2*i, l, avg);
    ll b = stYquery(ql, qr, 2*i+1, avg+1, r);
    return max(a, b);
}

//Con esta función hallamos la solución óptima una vez acutalizados los datos.
int solve(){
    ll best = 0; //Guarda la mejor solución hasta ahora.
    ll right = n-1;
    ll left;
    //Trabajamos por intervalos, los bordes de los intervalos son los elementos de
    ↪ non1Set.
    for (ll i : non1Set){ //Para obtener los elementos de mayor a menor, guardamos
    ↪ los opuestos.
        left = -i;
        //La mejor solución en este intervalo es la que tenga la Y máxima, porque
        ↪ todas las X interiores valen 1.
        ll candidate = stYquery(left, right, 1, 0, n-1);
        best = max(candidate, best);
        //Multiplicamos por el valor de X del primer elemento del intervalo
        ↪ preparandonos para comparar con el siguiente intervalo.
        best *= vX[left];
        //Actualizamos right para el siguiente intervalo.
        right = left-1;
        //Si la mejor solución hasta ahora es mayor que 10^9 no puede haber otra
        ↪ mejor.
        if (best > 1000000000 || left == 0)
            break;
    }
    best %= MOD;
    //Si todavía no se han procesado todos los índices, hay que multiplicar best
    ↪ por los valores de X restantes.
    if (right >= 0)
        best = best * stXquery(0, right, 1, 0, n-1) % MOD;
    return best;
}
```

```
int init(int N, int X[], int Y[]){
    //Inicializamos las variables y llamamos a solve para devolver la solución.
    n = N;
    vX = vector<ll>(n);
    stX = vector<ll>(4*n);
    stY = vector<ll>(4*n);
    for (int i = 0; i < n; i++){
        vX[i] = X[i];
        stXupdate(i, X[i], 1, 0, n-1);
        stYupdate(i, Y[i], 1, 0, n-1);
        if (X[i] > 1)
            non1Set.emplace(-i);
    }
    non1Set.emplace(0);
    return solve();
}

int updateY(int pos, int val){
    //Actualizamos el segment tree de la Y y llamamos a solve.
    stYupdate(pos, val, 1, 0, n-1);
    return solve();
}

int updateX(int pos, int val){
    //Actualizamos el set non1Set, no queremos sacar el 0 nunca.
    if (pos != 0){
        if (vX[pos] == 1 && val > 1){
            non1Set.emplace(-pos);
        }
        else if (vX[pos] > 1 && val == 1){
            non1Set.erase(-pos);
        }
    }
    //Actualizamos el vector vX y el segment tree de la X y llamamos a solve.
    vX[pos] = val;
    stXupdate(pos, val, 1, 0, n-1);
    return solve();
}
```