Vasos

Autora del problema: Blanca Huergo Muñoz (Miembro del comité científico)

Resumen del enunciado

Dada una hilera de vasos con tubos a diferentes alturas, predecir cuánta agua llegará al *j*-ésimo vaso si echamos *x* mililitros en el primero.

Explicación de la solución

- Antes de responder ninguna consulta, simulamos cómo se llenan los vasos, guardando para cada tubo cuánta agua ha de echarse para llegar a él por la izquierda (v_1) y cuánta ha de echarse para llegar a él por la derecha (v_2). Esta simulación la podemos realizar mediante un stack.
- Hacemos una *sparse table/segment tree* de la función *max* con los índices de v_2
- Para cada query, usamos el segment tree para ver el tubo más cercano por la izquierda al vaso actual al que no le ha llegado el agua por la derecha aún y lo mismo hacia la derecha. Sabiendo esto, podremos responder a la query.

Solución en C++

```
#include <iostream>
#include <vector>
using namespace std;

vector<int> alturas, v1, v2;
vector<int> seg;
int n;

void preprocess() {
	v1 = vector<int>(alturas.size()); // litros para añadirlo al stack
	v2 = vector<int>(alturas.size()); // litros para quitarlo del stack
	vector<int> stackTubos;
	stackTubos.push_back(0);
	int litrosActuales = alturas[0]; // litros totales echados
	v1[0] = alturas[0];
	int siguienteTubo = 1;
	int nivelAgua = 0; // nivel del agua a la derecha del último tubo del stack
	while(stackTubos.size()) {
	int tuboActual = stackTubos[stackTubos.size()-1];

	// llega el nivel de agua al lado derecho del tubo actual
	if (siguienteTubo == n-1 || alturas[tuboActual] < alturas[siguienteTubo]) {
		stackTubos.pop_back();

		int numVasos = siguienteTubo-tuboActual;
		litrosActuales += (alturas[tuboActual] - nivelAgua)*numVasos;
		nivelAgua = alturas[tuboActual];

		v2[tuboActual] = litrosActuales;
```

```
                // queda el stack vacío pero aún hay tubos por procesar
                if (stackTubos.empty() && siguienteTubo < n-1) {
                litrosActuales = (siguienteTubo+1) * alturas[siguienteTubo];
                nivelAgua = 0;
                v1[siguienteTubo] = litrosActuales;
                stackTubos.push_back(siguienteTubo);
                siguienteTubo++;
                }

        // llega el nivel de agua al lado derecho del tubo actual y al izquierdo del
siguiente
        } else if (alturas[tuboActual] == alturas[siguienteTubo]) {
                stackTubos.pop_back();
                stackTubos.push_back(siguienteTubo);

                int numVasos = siguienteTubo-tuboActual;
                litrosActuales += (alturas[tuboActual] - nivelAgua)*numVasos;
                nivelAgua = 0;

                v2[tuboActual] = v1[siguienteTubo] = litrosActuales;
                siguienteTubo++;

        // llega el nivel de agua al lado izquierdo del siguiente tubo
        } else {
                stackTubos.push_back(siguienteTubo);

                int numVasos = siguienteTubo-tuboActual;
                litrosActuales += (alturas[siguienteTubo] - nivelAgua)*numVasos;
                nivelAgua = 0;

                v1[siguienteTubo] = litrosActuales;
                siguienteTubo++;
        }
        }
}

// construye árbol de segmentos de índice de máximo con v2
void makeTree(int x, int L, int R) {
        if (L < R) {
        int mid = L + (R-L)/2;
        makeTree(2*x, L, mid);
        makeTree(2*x+1, mid+1, R);
        if (v2[seg[2*x]] >= v2[seg[2*x+1]])
                seg[x] = seg[2*x];
        else
                seg[x] = seg[2*x+1];
        } else {
        seg[x] = L;
        }
}

int tuboIzquierdaRec(int x, int L, int R, int vasoUB, int aguaEchada) {
        if (L >= vasoUB || v2[seg[x]] <= aguaEchada)
```

```
        return -1;
        if (L == R) {
        if (v2[seg[x]] > aguaEchada)
                return seg[x];
        return -1;
        }
        int mid = L + (R-L)/2;
        int rInd = tuboIzquierdaRec(2*x+1, mid+1, R, vasoUB, aguaEchada);
        if (rInd != -1)
        return rInd;
        return tuboIzquierdaRec(2*x, L, mid, vasoUB, aguaEchada);
}

// retorna el mayor ind < vasoUB tal que v2[ind] > aguaEchada
int tuboIzquierda(int vasoUB, int aguaEchada) {
        return tuboIzquierdaRec(1, 0, n-2, vasoUB, aguaEchada);
}

int tuboDerechaRec(int x, int L, int R, int vasoLB, int aguaEchada) {
        if (R < vasoLB || v2[seg[x]] <= aguaEchada)
        return -1;
        if (L == R) {
        if (v2[seg[x]] > aguaEchada)
                return seg[x];
        return -1;
        }
        int mid = L + (R-L)/2;
        int lInd = tuboDerechaRec(2*x, L, mid, vasoLB, aguaEchada);
        if (lInd != -1)
        return lInd;
        return tuboDerechaRec(2*x+1, mid+1, R, vasoLB, aguaEchada);
}

// retorna el menor ind >= vasoLB tal que v2[ind] > aguaEchada
int tuboDerecha(int vasoLB, int aguaEchada) {
        return tuboDerechaRec(1, 0, n-2, vasoLB, aguaEchada);
}

int cantidadEnElVaso(int aguaEchada, int vaso) {
        if (aguaEchada == 0)
        return 0;
        // el agua que se ha echado no es suficiente para que llegue al vaso
        if ((vaso == 0 && aguaEchada == 0) || (vaso > 0 && v1[vaso-1] > aguaEchada))
        return 0;
        int tuboDer = tuboDerecha(vaso, aguaEchada); // encontrar primer tubo a la derecha
del vaso al q no le ha llegado agua por la derecha
        if (tuboDer != -1 && v1[tuboDer] <= aguaEchada) {
        // caso 1: ya ha llegado el agua al nivel del tubo
        return alturas[tuboDer];
        } else {
        // caso 2: no ha llegado el agua al nivel del tubo
        int tuboIzq = tuboIzquierda(vaso, aguaEchada); // encontrar primer tubo a la
izquierda del vaso al q no le ha llegado agua por la derecha
        if (tuboIzq == -1) {
```

```
                if (tuboDer == -1) {
                // el agua ha llegado a la altura derecha de todos los tubos-> está
repartida de forma igual
                return aguaEchada/n;
                } else {
                // el agua ha llegado a la altura derecha de todos los tubos hasta tuboDer
(no incluido) + el agua no ha llegado a la altura de tuboDer-> está repartida de forma
igual entre los vasos anteriores
                return aguaEchada/(tuboDer+1);
                }
        } else if (tuboDer == -1) {
                // hay tubos a la izquierda a los que no ha llegado el agua a la derecha
pero a la derecha no -> está repartida de forma igual a partir de tuboIzq
                return (aguaEchada - v1[tuboIzq])/(n - tuboIzq - 1);
        } else {
                // el agua está repartida entre los vasos entre tuboizq y tuboder
                return (aguaEchada - v1[tuboIzq])/(tuboDer - tuboIzq);
        }
        }
}

int main() {
        int q, x, j, F;
        cin >> n;
        alturas = vector<int>(n-1);
        for (int i = 0; i < n-1; i++)
        cin >> alturas[i];
        preprocess();
        seg = vector<int>(4*n);
        makeTree(1, 0, n-2);
        cin >> q >> F;
        int C = 0;
        while(q--) {
        cin >> x >> j;
        if (F == 2) x -= C;
        C = cantidadEnElVaso(x, j-1);
        cout << C << '\n';
        }
        return 0;
}
```

Autora de la solución: Blanca Huergo Muñoz (Miembro del comité científico)