

Secuencia

Autor del problema: Félix Moreno Peñarrubia (Miembro del comité científico)

Resumen del enunciado

Dado m , se genera una secuencia con $a_1 = m$ y a_i es el menor múltiplo de i que es mayor igual que a_{i-1} .

Encontrar el n -ésimo valor de m que hace que la secuencia no tenga repetidos.

Explicación de la solución

- Si analizamos el comportamiento de la secuencia $b_i = a_i / i$, vemos que se estabiliza a partir de cierto punto.
- Intentando que no haya repetidos a partir del punto donde se estabiliza, vemos que para los números especiales se satisface que $b_i = i$ para algún i , es decir, $a_i = i^2$.
- De hecho, se puede ver que el n -ésimo número especial satisface que $a_n = n^2$, y la idea clave es ir calculando los términos al revés, es decir, a partir de a_n queda únicamente determinado a_{n-1} , a partir de a_{n-1} conseguimos a_{n-2} , etcétera, hasta llegar a a_1 .

Solución en C++

```
#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

int main() {
    ll n;
    cin >> n;
    ll x = n*n;
    for(ll i=n-1; i >= 1; --i) {
        ll d = x%i;
        if(d == 0) d = i;
        x -= d;
    }
    cout << x << endl;
}
```

Autor de la solución: Félix Moreno Peñarrubia (Miembro del comité científico)

Desordenado

Autor del problema: Félix Moreno Peñarrubia (Miembro del comité científico)

Resumen del enunciado

Reordenar una secuencia de números de manera que:

- La secuencia final no esté ordenada.
- Todos los elementos hayan cambiado de posición.

Explicación de la solución

- Desplazar la secuencia: así se garantiza la segunda condición, pero podría no respetarse la primera.
- Si después de un desplazamiento la secuencia está ordenada, desplazar de nuevo.

Solución en C++

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
typedef vector<int> vi;
```

```
vi shift(const vi& a) {  
    int n = a.size();  
    vi b = vi(n);  
    for(int i=0; i < n; ++i) b[i] = a[(i+1)%n];  
    return b;  
}
```

```
void solve() {  
    int n;  
    cin >> n;  
    vi a(n);  
    int ma = 1e9+1;  
    for(int i=0; i < n; ++i) {  
        cin >> a[i];  
        ma = min(ma, a[i]);  
    }  
    vi b = shift(a);  
    if(b[0] == ma) b = shift(b);  
    for(int i=0; i < n; ++i) {  
        cout << b[i];  
        if(i < n-1) cout << " ";  
    }  
    cout << endl;  
}
```

```
int main() {  
    int T;
```

```
cin >> T;  
while(T-->0) {  
    solve();  
}
```

Autor de la solución: Félix Moreno Peñarrubia (Miembro del comité científico)

Comiendo

Autor del problema: Félix Moreno Peñarrubia (Miembro del comité científico)

Resumen del enunciado

Tenemos montones de galletas y hay que comerlas todas en unos cuantos pasos. En cada paso, podemos comer galletas de cada montón tal que de un montón comamos tantas como la suma de las comidas del resto.

Explicación de la solución

- La suma tiene que ser par y no puede haber ningún montón con más galletas que la suma del resto.
- Iterativamente nos comemos todas las galletas del montón con más galletas y del resto de manera que nos aseguremos que la condición anterior se siga cumpliendo.
- Cuando la condición de la suma ya no vaya a cumplirse, nos podemos comer todas las otras galletas en dos días.

Solución en C++

```
#include<bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef vector<ll> vi;
typedef vector<vi> vvi;
typedef pair<ll, ll> ii;
typedef vector<ii> vii;

void solve() {
    int n;
    cin >> n;
    vi a(n);
    ll mx = 0;
    ll sum = 0;
    for(int i=0; i < n; ++i) {
        cin >> a[i];
        mx = max(mx, a[i]);
        sum += a[i];
    }
    if(sum%2 == 1 || 2*mx > sum) {
        cout << "NO" << endl;
        return;
    }
    vii b;
    for(int i=0; i < n; ++i) {
        b.emplace_back(a[i], i);
    }
    sort(b.begin(), b.end());
```

```

sort(a.begin(), a.end());
vi ra(n);
for(int i=0; i < n; ++i) {
    ra[b[i].second] = i;
}

vvi days;
int fp = 0;
int sp = n-1;
bool sumstate = false;
while(!sumstate && fp < sp-1) {
    ll mn = min(a[fp], a[sp]);
    vi nd(n);
    if(sum-2*mn <= 2*a[sp-1]) {
        mn = sum/2-a[sp-1];
        nd[fp] = mn;
        nd[sp] = mn;
        sum -= 2*mn;
        sumstate = true;
    }
    else if (mn == a[sp]) {
        nd[fp] = mn;
        nd[sp] = mn;
        sum -= 2*mn;
    }
    else {
        while(!sumstate && a[fp] + nd[sp] < a[sp]) {
            if(sum - 2*a[fp] <= 2*a[sp-1]) {
                mn = sum/2-a[sp-1];
                nd[fp] = mn;
                nd[sp] += mn;
                sum -= 2*mn;
                sumstate = true;
            }
            else {
                nd[fp] = a[fp];
                nd[sp] += a[fp];
                sum -= 2*a[fp];
                fp++;
            }
        }
        if(!sumstate) {
            if(sum - 2*(a[sp]-nd[sp]) <= 2*a[sp-1]) {
                mn = sum/2-a[sp-1];
                nd[fp] = mn;
                nd[sp] += mn;
                sum -= 2*mn;
                sumstate = true;
            }
            else {
                nd[fp] = a[sp]-nd[sp];
                sum -= 2*(a[sp]-nd[sp]);
                nd[sp] = a[sp];
            }
        }
    }
}

```

```

        }
    }
    ll sum2 = 0;
    ll sumnd = 0;
    ll mxnd = 0;
    for(int i=0; i < n; ++i) {
        a[i] -= nd[i];
        sumnd += nd[i];
        mxnd = max(mxnd, nd[i]);
        sum2 += a[i];
    }
    days.push_back(nd);
    if(a[fp] == 0) fp++;
    if(a[sp] == 0) sp--;
}
bool nz = false;
ll mx2 = 0;

for(int i=0; i < n; ++i) {
    if(a[i] > 0) nz = true;
    mx2 = max(mx2, a[i]);
}
if(nz) days.push_back(a);

cout << "SI" << endl;
int d = days.size();
cout << d << endl;
for(int i=0; i < d; ++i) {
    for(int j=0; j < n; ++j) {
        cout << days[i][ra[j]];
        if(j < n-1) cout << " ";
    }
    cout << endl;
}

}

int main() {
    int T;
    cin >> T;
    while(T--) {
        solve();
    }
}

```

Autor de la solución: Félix Moreno Peñarrubia (Miembro del comité científico)

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)

Reescritura

Autor del problema: Edgar Moreno Martínez (Miembro del comité científico)

Resumen del enunciado

Determinar si se puede convertir una string en otra con transformaciones de la forma $a \rightarrow bc$ sin ciclos.

Explicación de la solución

- Nos fijamos en la última letra de cada string. Dos casos:
 - Son la misma letra: las borramos.
 - Son diferentes: solo podemos aplicar una transformación a la última letra de la primera string.
- Si al final tenemos dos string vacías lo hemos conseguido. Si en algún momento la primera es más larga que la segunda o no podemos transformar una letra no lo podemos conseguir.

Solución en C++

```
#include<bits/stdc++.h>

using namespace std;

bool solve() {
    int n;
    cin >> n;
    map<char, pair<char, char>> mp;
    for(int i = 0; i < n; ++i) {
        char a,b,c,d;
        cin >> a >> d >> d >> b >> c;
        mp[a] = pair<char, char>(b,c);
    }
    string s, t;
    cin >> s >> t;
    reverse(s.begin(), s.end());
    reverse(t.begin(), t.end());
    while(s.size() != 0 and t.size() != 0) {
        if(s.size() > t.size()) return false;
        char last = s.back();
        if(last == t.back()) {
            s.pop_back();
            t.pop_back();
            continue;
        }
        if(mp.count(last) == 0) return false;
        s.pop_back();
        s.push_back(mp[last].second);
        s.push_back(mp[last].first);
    }
}
```

```
        return s.size() == 0 and t.size() == 0;
    }

    int main() {
        int T;
        cin >> T;
        while(T-->0) {
            if(solve())
                cout << "SI" << endl;
            else
                cout << "NO" << endl;
        }
    }
}
```

Autor de la solución: Edgar Moreno Martínez (Miembro del comité científico)