Soluciones del entrenamiento avanzado 1

Árboles de segmentos avanzados: *Lazy* y *Walking*

Alejandro Vivero Puga

9 de octubre de 2024

A. Range Update Queries (CSES)

Este es el problema con el que se explicó *lazy propagation* en la clase. Es estándar si se conoce esta técnica. Se puede ver la grabación de la clase sobre la técnica aquí y un manual aquí.

El código es el siguiente:

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

using namespace std;

typedef long long ll;

struct segTree{
  vector<ll> modificaciones;
  int tamano;
  void init(int n){
      tamano = 1;
      while(tamano < n) tamano *= 2;
      modificaciones.assign(2 * tamano, 0);
  }
  void push(int x){
      modificaciones[2 * x + 1] += modificaciones[x];
      modificaciones[2 * x + 2] += modificaciones[x];
      modificaciones[x] = 0;
  }
  void add(int l, int r, int u, int x, int lx, int rx){
      if(lx >= l && rx <= r){
          modificaciones[x] += u;
          return;
      }else if(lx >= r || rx <= l) return;
      push(x);
      int m = (lx + rx) / 2;
      add(l, r, u, 2 * x + 1, lx, m);
      add(l, r, u, 2 * x + 2, m, rx);
  }
  void add(int l, int r, int u){
      add(l, r, u, 0, 0, tamano);
  }
```

```
  ll get(int i, int x, int lx, int rx){
      if(rx - lx == 1){
          return modificaciones[x];
      }
      int m = (lx + rx) / 2;
      push(x);
      if(i < m) return get(i, 2 * x + 1, lx, m);
      else return get(i, 2 * x + 2, m, rx);
  }
  ll get(int i){
      return get(i, 0, 0, tamano);
  }
};

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, q; cin >> n >> q;
    segTree st; st.init(n);
    for(int i = 0; i < n; i++){
        int x; cin >> x;
        st.add(i, i+1, x);
    }
    while(q--){
        int type; cin >> type;
        if(type == 1){
            int l, r, u; cin >> l >> r >> u; l--; r--;
            st.add(l, r + 1, u);
        }else{
            int i; cin >> i; i--;
            cout << st.get(i) << "\n";
        }
    }
}
```

B. Hotel Queries (CSES)

Este es el problema con el que se explicó *walking* en la clase. Si se mantiene un árbol de segmentos de máximos, el problema se reduce a encontrar con *walking* el primer hotel con una cantidad de gente mayor o igual a la que piden, y después restar esa cantidad a ese hotel en el árbol de segmentos. Se puede ver la grabación de la clase sobre la técnica aquí y un manual aquí.

Nota: El código usa *lazy propagation* para restar a los hoteles. Esto es innecesario y se podría hacer con un árbol de segmentos

normal, pero para mostrar un código combinando *lazy propagation* y *walking*, se hizo así en clase.

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

using namespace std;

typedef long long ll;

struct segTree{
  vector<ll> modificaciones, valores;
  int tamano;
  void init(int n){
      tamano = 1;
      while(tamano < n) tamano *= 2;
      modificaciones.assign(2 * tamano, 0);
      valores.assign(2 * tamano, 0);
  }
  void push(int x){
      valores[2 * x + 1] += modificaciones[x];
      valores[2 * x + 2] += modificaciones[x];
      modificaciones[2 * x + 1] += modificaciones[x];
      modificaciones[2 * x + 2] += modificaciones[x];
      modificaciones[x] = 0;
  }
  void add(int l, int r, int u, int x, int lx, int rx){
      if(lx >= l && rx <= r){
          modificaciones[x] += u;
          valores[x] += u;
          return;
```

```
28          }else if(lx >= r || rx <= l) return;
29          push(x);
30          int m = (lx + rx) / 2;
31          add(l, r, u, 2 * x + 1, lx, m);
32          add(l, r, u, 2 * x + 2, m, rx);
33          valores[x] = max(valores[2 * x + 1], valores[2 * x+ 2]);
34      }
35      void add(int l, int r, int u){
36          add(l, r, u, 0, 0, tamano);
37      }
38      int walk(int x, int lx, int rx, int k){
39          if(rx - lx == 1){
40              if(valores[x] >= k) return lx;
41              else return -1;
42          }
43          int m = (lx + rx) / 2;
44          push(x);
45          if(valores[2 * x + 1] >= k){
46            return walk(2 * x + 1, lx, m, k);
47          }else return walk(2 * x + 2, m, rx, k);
48      }
49      int walk(int k){
50          return walk(0, 0, tamano, k);
51      }
52  };
53
54  int main(){
55      ios_base::sync_with_stdio(0);
56      cin.tie(0);
57
58      int n, q; cin >> n >> q;
59      segTree st; st.init(n);
60      for(int i = 0; i < n; i++){
61          int x; cin >> x;
62          st.add(i, i+1, x);
63      }
64      while(q--){
65          int k; cin >> k;
66          int i = st.walk(k);
```

```
67          if(i == -1) cout << 0 << " ";
68          else{
69              cout << i + 1 << " ";
70              st.add(i, i+1, -k);
71          }
72      }
73  }
```

C. Prefix sum queries (CSES)

Podemos mantener el valor de la suma de todos los elementos hasta la posición i usando un árbol de segmentos. Para cada i, sumamos x_i a todos los índices $j \geq i$. Por lo tanto, si tenemos un árbol de segmentos de máximos con *lazy propagation*, podemos obtener el valor máximo de la suma de todos los valores entre las posiciones 1 e i para cualquier i en un rango $[l, r]$. Además, eso es suficiente también para obtener el valor máximo de la suma de todos los valores entre las posiciones a e i para cualquier i en el rango $[a, b]$, que es lo que nos pide el problema, puesto que sirve con restar a todo el rango $[a, b]$ la suma de valores entre posiciones 1 y $a - 1$ y después encontrar el máximo.

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

using namespace std;

typedef long long ll;

struct segTree {
    int size;
    vector<ll> mx, upd;
    void init(int n){
        size = 1;
        while (size < n) size *= 2;
        mx.assign(2 * size, 0);
        upd.assign(2 * size, 0);
    }
    void push(int x){
        mx[2 * x + 1] += upd[x];
        mx[2 * x + 2] += upd[x];
        upd[2 * x + 1] += upd[x];
        upd[2 * x + 2] += upd[x];
        upd[x] = 0;
    }
    void add(int l, int r, int v, int x, int lx, int rx){
        if (lx >= l && rx <= r){
            mx[x] += v;
            upd[x] += v;
```

```
            return;
        }else if(lx >= r || rx <= l) return;
        int m = (lx + rx) / 2;
        push(x);
        add(l, r, v, 2 * x + 1, lx, m);
        add(l, r, v, 2 * x + 2, m, rx);
        mx[x] = max(mx[x * 2 + 1], mx[x * 2 + 2]);
    }
    void add(int l, int r, int v){
        add(l, r, v, 0, 0, size);
    }

    ll calc(int l, int r, int x, int lx, int rx){
        if (lx >= r || l >= rx)
            return -1e18;
        if (lx >= l && rx <= r){
            return mx[x];
        }
        int m = (lx + rx) / 2;
        push(x);
        ll s1 = calc(l, r, 2 * x + 1, lx, m);
        ll s2 = calc(l, r, 2 * x + 2, m, rx);
        return max(s1, s2);
    }

    ll calc(int l, int r){
        return calc(l, r, 0, 0, size);
    }
};


int main(){
    int n, q; cin >> n >> q;
    segTree st; st.init(n);
    vector<int> v(n);
    for(int i = 0; i < n; i++){
        cin >> v[i];
        st.add(i, n, v[i]);
    }
```

```
    while(q--){
        int t; cin >> t;
        if(t == 1){
            int i, x; cin >> i >> x; i--;
            st.add(i, n, x - v[i]);
            v[i] = x;
        }else{
            int l, r; cin >> l >> r; l--;
            if(l > 0){
                ll prev = st.calc(l - 1, l);
                cout << max(st.calc(l, r) - prev, 0ll) << "\n";
            }else cout << max(st.calc(l, r), 0ll) << "\n";
        }
    }
}
```

D. Range Updates and Sums (CSES)

Este problema consiste en hacer una *lazy propagation* más complicada. En particular, es útil hacerla de arriba a abajo usando una función *push*, aunque también es posible haciendo la propagación de abajo a arriba. En concreto, nos importa el orden en el que se hacen las operaciones de *set* y *sum*, y debemos mantenerlas en la función *push*. Cada vez que haya una operación *set*, lo marcaremos para igualar todos los valore a ese y, si hay operaciones de *sum* posteriores, aplicarlas sobre el valor al que se ha asignado. Por lo tanto, tendremos una pareja en la cual el segundo valor nos indicará si debemos asignar o simplemente sumar el primer valor a todo el rango.

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

using namespace std;

typedef long long ll;

struct segTree{
    vector<ll> v;
    vector<pair<ll, int>> upd;
    int size = 1;
    void init(int n){
        while(size < n) size *= 2;
        v.assign(2 * size, 0);
        upd.assign(2 * size, {0, 0});
    }
    void push(int x, int lx, int rx){
        int szChild = (rx - lx)/2; //Tamaño del rango del hijo.
        if(upd[x].second == 1){
            //Ha habido una operación set,
            //así que setteo todo el rango a upd[x].first;
            v[2 * x + 1] = szChild * upd[x].first;
            v[2 * x + 2] = szChild * upd[x].first;
            //También tienen que ser iguales.
            upd[2 * x + 1] = upd[x];
            upd[2 * x + 2] = upd[x];
            upd[x] = {0, 0};
```

```
        } else {
            //No setteo, solo sumo.
            v[2 * x + 1] += szChild * upd[x].first;
            v[2 * x + 2] += szChild * upd[x].first;
            //Sumo lo que he sumado ahora.
            upd[2 * x + 1].first += upd[x].first;
            upd[2 * x + 2].first += upd[x].first;
            upd[x] = {0, 0};
        }
    }
    void add(int l, int r, int x, int lx, int rx, ll u){
        if(lx >= l && rx <= r){
            v[x] += u * (rx - lx);
            upd[x].first += u;
            return;
        }
        if(lx >= r || l >= rx) return;
        int m = (lx + rx) / 2;
        push(x, lx, rx);
        add(l, r, 2 * x + 1, lx, m, u);
        add(l, r, 2 * x + 2, m, rx, u);
        v[x] = v[2 * x + 1] + v[2 * x + 2];
        return;
    }
    void add(int l, int r, int u){
        return add(l, r, 0, 0, size, u);
    }
    void set(int l, int r, int x, int lx, int rx, ll u){
        if(lx >= l && rx <= r){
            v[x] = u * (rx - lx);
            upd[x].first = u;
            upd[x].second = 1;
            return;
        }
        if(lx >= r || l >= rx) return;
        int m = (lx + rx) / 2;
        push(x, lx, rx);
        set(l, r, 2 * x + 1, lx, m, u);
        set(l, r, 2 * x + 2, m, rx, u);
```

```
66          v[x] = v[2 * x + 1] + v[2 * x + 2];
67          return;
68      }
69      void set(int l, int r, int u){
70          return set(l, r, 0, 0, size, u);
71      }
72
73      void build(vector<ll> &a, int x, int lx, int rx){
74          if (rx - lx == 1){
75              if (lx < (int)a.size()){
76                  v[x] = a[lx];
77              }
78              return;
79          }
80          int m = (lx + rx) / 2;
81          build(a, 2 * x + 1, lx, m);
82          build(a, 2 * x + 2, m, rx);
83          v[x] = v[2 * x + 1] + v[2 * x + 2];
84      }
85      void build(vector<ll>& a){
86          build(a, 0, 0, size);
87      }
88
89      ll sum(int l, int r, int x, int lx, int rx){
90          if(lx >= l && rx <= r) return v[x];
91          else if(lx >= r || rx <= l) return 0;
92          int m = (lx + rx) / 2;
93          push(x, lx, rx);
94          ll s1 = sum(l, r, 2 * x + 1, lx, m);
95          ll s2 = sum(l, r, 2 * x + 2, m, rx);
96          return s1 + s2;
97      }
98
99      ll sum(int l, int r){
100         return sum(l, r, 0, 0, size);
101     }
102 };
103
104 int main(){
```

```
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, q; cin >> n >> q;
    vector<ll> t(n);
    for(auto &i : t) cin >> i;
    segTree st; st.init(n);
    st.build(t);
    while(q--){
        int type, a, b; cin >> type >> a >> b; a--; b--;
        if(type == 1){
            int x; cin >> x;
            st.add(a, b+1, x);
        }else if(type == 2){
            int x; cin >> x;
            st.set(a, b+1, x);
        }else{
            cout << st.sum(a, b+1) << "\n";
        }
    }
}
```

E. Polynomial Queries (CSES)

Si cada operación de tipo 1 nos crea una progresión aritmética diferente que empieza en sitios distintos, la solución inicial sería para cada rango tendríamos que mantener muchas progresiones aritméticas con términos iniciales diferentes, lo cual sería muy lento. Lo primero que debemos hacer es, en cierto modo, normalizar las progresiones aritméticas, es decir, hacer que todas sean empezando en 1 y sumando una constante a todo el rango, igual a la suma de los términos iniciales de las progresiones que hay. Siguiendo con esta idea, lo que nos mantendremos para nodo y lo que actualizaremos con la función *push* será, por un lado, la suma de los términos iniciales de las progresiones aritméticas que incluyan al rango, y por otro la cantidad de estas. Con esa información, podemos calcular lo que hay que sumar a cada rango, y siempre que dividamos el rango en 2, podemos pasarles esta información de manera adecuada.

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

using namespace std;

typedef long long ll;

struct segTree{
    vector<ll> v;
    vector<pair<ll, ll>> upd;
    //first: the sum of initial terms. second: the number of APs.
    int size;
    ll dif;
    void init(int n, ll _dif = 1){
        //En este caso se suma de 1 en 1,
        //pero se podría sumar de dif en dif.
        size = 1;
        dif = _dif;
        while(size < n) size *= 2;
        v.assign(2 * size, 0);
        upd.assign(2 * size, {0, 0});
    }
    ll ap(ll x){
        return dif*((x*(x+1))/2);
```

```
    }
    void push(int x, int lx, int rx){
        int szChild = (rx - lx)/2;
        v[2 * x + 1] += upd[x].first * szChild
            + ap(szChild-1) * upd[x].second;
        upd[2 * x + 1].first += upd[x].first;
        upd[2 * x + 1].second += upd[x].second;
        ll initialTermsRight = upd[x].first
            + szChild * upd[x].second;
        v[2 * x + 2] += initialTermsRight * szChild
            + ap(szChild-1) * upd[x].second;
        upd[2 * x + 2].first += initialTermsRight;
        upd[2 * x + 2].second += upd[x].second;
        upd[x] = {0, 0};
    }
    void set(int l, int r, int x, int lx, int rx, ll u){
        if(lx >= l && rx <= r){
            v[x] += (u + lx - l) * (rx - lx) + ap(rx - lx - 1);
            upd[x].first += u + lx - l;
            upd[x].second++;
            return;
        }
        if(lx >= r || l >= rx) return;
        int m = (lx + rx) / 2;
        push(x, lx, rx);
        set(l, r, 2 * x + 1, lx, m, u);
        set(l, r, 2 * x + 2, m, rx, u);
        v[x] = v[2 * x + 1] + v[2 * x + 2];
        return;
    }
    void set(int l, int r, ll u = 1){
        //En este caso siempre se empieza sumando 1,
        //pero podría empezarse sumando u.
        return set(l, r, 0, 0, size, u);
    }
    void build(vector<ll> &a, int x, int lx, int rx){
        if (rx - lx == 1){
            if (lx < (int)a.size()){
                v[x] = a[lx];
```

```
            }
            return;
        }
        int m = (lx + rx) / 2;
        build(a, 2 * x + 1, lx, m);
        build(a, 2 * x + 2, m, rx);
        v[x] = v[2 * x + 1] + v[2 * x + 2];
    }
    void build(vector<ll>& a){
        build(a, 0, 0, size);
    }
    ll sum(int l, int r, int x, int lx, int rx){
        if(lx >= l && rx <= r) return v[x];
        else if(lx >= r || rx <= l) return 0;
        int m = (lx + rx) / 2;
        push(x, lx, rx);
        ll s1 = sum(l, r, 2 * x + 1, lx, m);
        ll s2 = sum(l, r, 2 * x + 2, m, rx);
        return s1 + s2;
    }
    ll sum(int l, int r){
        return sum(l, r, 0, 0, size);
    }
};

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, q; cin >> n >> q;
    vector<ll> a(n);
    for(auto &i : a) cin >> i;
    segTree st; st.init(n);
    st.build(a);
    while(q--){
        int type, a, b; cin >> type >> a >> b; a--; b--;
        if (type == 1) {
            st.set(a, b+1);
        } else {
```

```
            cout << st.sum(a, b+1) << "\n";
        }
    }
}
```

F. Resto (OIE 2024)

La observación principal en este problema es que el valor de x solo cambia $O(\log x)$ veces si vamos aplicando operaciones de módulo. La demostración es que, obviamente, el valor solo cambiará si se le aplica el módulo con un número $y < x$, y entonces tenemos dos casos. Primer caso, $y \geq \frac{x}{2}$, entonces, al aplicar el módulo, x se reduce por al menos la mitad. Segundo caso, $y < \frac{x}{2}$, al aplicar el módulo, el valor final debe ser menor a y, por lo que, de nuevo, x se reduce por al menos la mitad. Vemos que, siempre que aplicamos una operación de módulo sobre x con otro número $y < x$, se reduce este a la mitad, por lo tanto solo habrá $O(\log n)$ valores diferentes de x. Esta también es la demostración de la complejidad del algoritmo de Euclides.

Usando este hecho, el problema se reduce a mantener un árbol de segmentos de mínimos y, para cada consulta, encontrar con *walking* el siguiente valor menor al actual, hasta que no haya ninguno en todo el arreglo.

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

using namespace std;

typedef long long ll;

struct segTree{
  vector<ll> valores;
  int tamano;
  void init(int n){
      tamano = 1;
      while(tamano < n) tamano *= 2;
      valores.assign(2 * tamano, 1e18);
  }
  void set(int i, int u, int x, int lx, int rx){
      if(rx - lx == 1){
        valores[x] = u;
        return;
      }
      int m = (lx + rx) / 2;
      if(i < m) set(i, u, 2 * x + 1, lx, m);
      else set(i, u, 2 * x + 2, m, rx);
```

```
23          valores[x] = min(valores[2 * x + 1], valores[2 * x+ 2]);
24      }
25      void set(int i, int u){
26          set(i, u, 0, 0, tamano);
27      }
28      int walk(int x, int lx, int rx, int k){
29          if(rx - lx == 1){
30              if(valores[x] <= k) return lx;
31              else return -1;
32          }
33          int m = (lx + rx) / 2;
34          if(valores[2 * x + 1] <= k){
35            return walk(2 * x + 1, lx, m, k);
36          }else return walk(2 * x + 2, m, rx, k);
37      }
38      int walk(int k){
39          return walk(0, 0, tamano, k);
40      }
41  };
42
43  int main(){
44      ios_base::sync_with_stdio(0);
45      cin.tie(0);
46
47      int T; cin >> T;
48      while(T--){
49          int n, q; cin >> n >> q;
50          segTree st; st.init(n);
51          vector<ll> a(n);
52          for(int i = 0; i < n; i++){
53              int x; cin >> x;
54              st.set(i, x);
55              a[i] = x;
56          }
57          while(q--){
58              int type; cin >> type;
59              if(type == 1) {
60                  ll x; cin >> x;
61                  int ind = st.walk(x);
```

```
                ll ans = 0;
                int lstInd = 0;
                while(ind != -1){
                    ans += (ind - lstInd) * x;
                    x %= a[ind];
                    lstInd = ind;
                    ind = st.walk(x);
                }
                ans += (n - lstInd) * x;
                cout << ans << "\n";
            } else {
                int p; ll x; cin >> p >> x; p--;
                a[p] = x;
                st.set(p, x);
            }
        }
    }
}
```

G. Wall (IOI 2014)

Las operaciones se pueden traducir a decir que todos los elementos en un rango tienen que ser mínimo h o máximo h, según la operación. Una vez traducidas, podemos mantener la pared con un árbol de segmentos y, con *lazy propagation*, mantener el intervalo en el que pueden estar los valores de ese rango.

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

using namespace std;

struct segTree{
    vector<pair<int, int>> v;
    int siz;
    void init(int n){
        siz = 1;
        while(siz < n) siz *= 2;
        v.assign(2 * siz, {0, 0});
    }
    void push(int x){
        v[2 * x + 1].first =
            min(max(v[2 * x + 1].first, v[x].first), v[x].second);
        v[2 * x + 1].second =
            max(min(v[2 * x + 1].second, v[x].second), v[x].first);
        v[2 * x + 2].first =
            min(max(v[2 * x + 2].first, v[x].first), v[x].second);
        v[2 * x + 2].second =
            max(min(v[2 * x + 2].second, v[x].second), v[x].first);
    }
    void set(int l, int r, int x, int lx, int rx, int h, int t){
        if(lx >= l && rx <= r){
            if(t == 1){
                v[x].first = max(v[x].first, h);
                v[x].second = max(v[x].second, v[x].first);
            }else{
                v[x].second = min(v[x].second, h);
                v[x].first = min(v[x].first, v[x].second);
            }
```

```
32              return;
33          }else if(lx >= r || rx <= l) return;
34          push(x);
35          v[x] = {0, 100000};
36          int m = (lx + rx) / 2;
37          set(l, r, 2 * x + 1, lx, m, h, t);
38          set(l, r, 2 * x + 2, m, rx, h, t);
39      }
40      void set(int l, int r, int h, int t){
41          set(l, r, 0, 0, siz, h, t);
42      }
43      int calc(int i, int x, int lx, int rx){
44          if(rx - lx == 1){
45              return v[x].first;
46          }
47          push(x);
48          v[x] = {0, 100000};
49          int m = (lx + rx) / 2;
50          if(i < m) return calc(i, 2 * x + 1, lx, m);
51          else return calc(i, 2 * x + 2, m, rx);
52      }
53      int calc(int i){
54          return calc(i, 0, 0, siz);
55      }
56  };
57
58  void buildWall(int n, int k, int* op, int* left, int* right,
59      int* height, int* finalHeight){
60      segTree st; st.init(n);
61      for(int i = 0; i < k; i++){
62          st.set(left[i], right[i] + 1, height[i], op[i]);
63      }
64      for(int i = 0; i < n; i++){
65          finalHeight[i] = st.calc(i);
66      }
67  }
```

H. Picky Byteasar (POI 2022)

Este problema requiere tanto de *lazy propagation* como de *walking*. El *walking* se usará solo para saber hasta qué lámpara tenemos que cambiar los colores a por b. La parte más complicada en este caso es la *lazy*. La idea es, para cada nodo, mantener la suma de lámparas de cada color. Adicionalmente, para poder hacer el *push* y modificar el arreglo, queremos mantener también a qué color corresponde cada color. Por ejemplo, inicialmente, cada color corresponde a sí mismo, pero tras cambiar todas las lámparas de color a a color b en un nodo, podemos decir que a corresponde a b. Mantener estas correspondencias puede ser un poco lioso y requerir detalles. Por ejemplo, no solo debemos decir que a corresponde a b, sino que todos los colores que ya correspondían a a también pasan a corresponder a b ahora.

Nota: Este problema puede requerir algo de optimización constante para que no de *Time Limit Exceeded*. Por ejemplo, yo tuve que cambiar de *structs* a arreglos.

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

using namespace std;

int ord(char c){
    return (int)(c - 'a');
}

const int siz = (1 << 20);
int v[2 * siz][5];
int upd[2 * siz][5];
bool change[2 * siz];

void buildd(string& s, int x, int lx, int rx){
    for(int i = 0; i < 5; i++){
        upd[x][i] = i;
        v[x][i] = 0;
    }
    change[x] = 0;
    if(rx - lx == 1){
        if(lx < (int)s.size()){
```

```
            v[x][ord(s[lx])]++;
        }
        return;
    }
    int m = (lx + rx) / 2;
    buildd(s, 2 * x + 1, lx, m);
    buildd(s, 2 * x + 2, m, rx);
    for(int i = 0; i < 5; i++){
        v[x][i] = v[2 * x + 1][i] + v[2 * x + 2][i];
    }
}
void build(string& s){
    buildd(s, 0, 0, siz);
}

void prop(int x){
    if(change[x] == 0) return;
    int nw1[5];
    int nw2[5];
    for(int i = 0; i < 5; i++){
        nw1[i] = 0;
        nw2[i] = 0;
    }
    for(int i = 0; i < 5; i++){
        nw1[upd[x][i]] += v[2 * x + 1][i];
        nw2[upd[x][i]] += v[2 * x + 2][i];
        upd[2 * x + 1][i] = upd[x][upd[2 * x + 1][i]];
        upd[2 * x + 2][i] = upd[x][upd[2 * x + 2][i]];
    }
    for(int i = 0; i < 5; i++){
        v[2 * x + 1][i] = nw1[i];
        v[2 * x + 2][i] = nw2[i];
        upd[x][i] = i;
    }
    change[2 * x + 1] = 1; change[2 * x + 2] = 1;
    change[x] = 0;
}

void settt(int l, int r, int a, int b, int x, int lx, int rx){
```

```
61      if(lx >= l && rx <= r){
62          v[x][b] += v[x][a]; v[x][a] = 0;
63          for(int i = 0; i < 5; i++){
64              if(upd[x][i] == a) upd[x][i] = b;
65          }
66          change[x] = 1;
67          return;
68      }else if(lx >= r || rx <= l) return;
69      int m = (lx + rx) / 2;
70      prop(x);
71      settt(l, r, a, b, 2 * x + 1, lx, m);
72      settt(l, r, a, b, 2 * x + 2, m, rx);
73      for(int i = 0; i < 5; i++){
74          v[x][i] = v[2 * x + 1][i] + v[2 * x + 2][i];
75      }
76  }
77  void sett(int l, int r, int a, int b){
78      settt(l, r, a, b, 0, 0, siz);
79  }
80
81  int walkk(int a, int p, int x, int lx, int rx, int add){
82      if(rx - lx == 1){
83          //The statement guarantees that there are P a's.
84          return lx;
85      }
86      int m = (lx + rx) / 2;
87      prop(x);
88      if(v[2 * x + 1][a] + add >= p){
89          return walkk(a, p, 2 * x + 1, lx, m, add);
90      }else{
91          return walkk(a, p, 2 * x + 2, m, rx, add + v[2 * x + 1][a]);
92      }
93  }
94  int walk(int a, int p){
95      return walkk(a, p, 0, 0, siz, 0);
96  }
97
98  int gettt(int i, int x, int lx, int rx){
99      if(rx - lx == 1){
```

```
        for(int j = 0; j < 5; j++){
            if(v[x][j] > 0) return j;
        }
        assert(false);
        return 0;
    }
    prop(x);
    int m = (lx + rx) / 2;
    if(i < m) return gettt(i, 2 * x + 1, lx, m);
    else return gettt(i, 2 * x + 2, m, rx);
}
int gett(int i){
    return gettt(i, 0, 0, siz);
}

void solve(){
    int n, q; cin >> n >> q;
    string s; cin >> s;
    build(s);
    while(q--){
        int p; cin >> p; char c1, c2; cin >> c1 >> c2;
        int a = ord(c1);
        int b = ord(c2);
        int ind = walk(a, p);
        sett(0, ind + 1, a, b);
    }
    for(int i = 0; i < n; i++){
        cout << char(gett(i) + 'a');
    }
    cout << endl;
}

int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    solve();
}
```

I. Prefix queries (TeamsCode 2023)

Comenzaremos creando un arreglo b tal que $b_i = a_i - \sum_{j=1}^{i-1} a_j$. Podemos ver que los i que satisfacen las condiciones son aquellos tales que $2 \leq i \leq n$ y $b_i \geq 0$. Ahora, la idea clave para resolver el problema se basa en cómo afectan las consultas al arreglo b. Después de cada consulta, solo b_l puede pasar de negativo a positivo. Esto se debe a que el arreglo b cambia de la siguiente manera:

- Los elementos con índice $i < l$ permanecen sin cambios.
- Para los elementos con índice $l \leq i \leq r$, $b_i := b_i - x \cdot (i - l - 1)$. b_l aumenta, b_{l+1} permanece igual y el resto disminuyen.
- Para los elementos con índice $i > r$, $b_i := b_i - x \cdot (r - l + 1)$. Todos disminuyen.

Teniendo esto en cuenta, sabemos que podemos mantener un *set* con todas las respuestas posibles. Cada vez que algún b_i se vuelve positivo, agregamos i al conjunto, sabiendo que haremos como máximo $n + q$ inserciones. Luego, para responder a las consultas, simplemente recorremos el conjunto de menor a mayor, eliminando los elementos hasta que encontremos uno que sea positivo, lo imprimimos o el conjunto queda vacío, en cuyo caso devolvemos -1. Para saber si algún b_i es positivo o no, podemos usar un árbol de segmentos con *lazy* que soporte actualizaciones en rango y consultas en rango.

Nota: Este árbol de segmentos lo hice hace mucho tiempo. Si os fijáis, implemento la suma de abajo a arriba en vez de arriba a abajo con función *push*, que es algo más liosa.

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  struct segTree{
6      vector<long long> v;
7      vector<long long> upd;
8      int size;
9      void init(int n){
```

```
        size = 1;
        while(size < n) size *= 2;
        v.assign(2 * size, 0);
        upd.assign(2 * size, 0);
    }
    void set(int l, int r, int x, int lx, int rx, long long u){
        if(lx >= l && rx <= r){
            v[x] += u * (rx - lx);
            upd[x] += u;
            return;
        }
        if(lx >= r || l >= rx) return;
        int m = (lx + rx) / 2;
        set(l, r, 2 * x + 1, lx, m, u);
        set(l, r, 2 * x + 2, m, rx, u);
        v[x] = v[2 * x + 1] + v[2 * x + 2] + upd[x] * (rx - lx);
    }
    void set(int l, int r, long long u){
        return set(l, r, 0, 0, size, u);
    }

    void build(vector<int>& a, int x, int lx, int rx){
        if (rx - lx == 1){
            if (lx < (int)a.size()){
                v[x] = a[lx];
            }
            return;
        }
        int m = (lx + rx) / 2;
        build(a, 2 * x + 1, lx, m);
        build(a, 2 * x + 2, m, rx);
        v[x] = v[2 * x + 1] + v[2 * x + 2];
    }
    void build(vector<int>& a){
        build(a, 0, 0, size);
    }

    pair<long long, int> sum(int l, int r, int x, int lx, int rx){
        if(lx >= l && rx <= r) return {v[x], rx - lx};
```

```
49          else if(lx >= r || rx <= l) return {0, 0};
50          int m = (lx + rx) / 2;
51          pair<long long, int> s1 = sum(l, r, 2 * x + 1, lx, m);
52          pair<long long, int> s2 = sum(l, r, 2 * x + 2, m, rx);
53          return {s1.first + s2.first + upd[x]
54              * (s1.second + s2.second), (s1.second + s2.second)};
55      }
56
57      long long sum(int l, int r){
58          pair<long long, int> p = sum(l, r, 0, 0, size);
59          return p.first;
60      }
61  };
62
63
64  int main(){
65      ios_base::sync_with_stdio(0);
66      cin.tie(0);
67
68      int n, q; cin >> n >> q;
69      vector<int> a(n);
70      set<int> positive; //Contiene las respuestas posibles.
71      long long sum = 0;
72      for(int i = 0; i < n; i++){
73          int x; cin >> x;
74          a[i] = x;
75          if(i > 0 && x - sum >= 0) positive.insert(i);
76          sum += x;
77      }
78      segTree st; st.init(n); st.build(a);
79      while(q--){
80          int l, r, x; cin >> l >> r >> x; l--; r--;
81          st.set(l, r + 1, (long long)x);
82          if(l > 0 && st.sum(l, l + 1) - st.sum(0, l) >= 0){
83              positive.insert(l);
84          }
85          bool done = 0;
86          while((int)positive.size() > 0 && done == 0){
87              int ind = *positive.begin();
```

```
88              if(st.sum(ind, ind + 1) - st.sum(0, ind) >= 0){
89                  done = 1;
90                  cout << ind + 1 << "\n";
91              }else{
92                  positive.erase(positive.begin());
93              }
94          }
95          if(!done) cout << -1 << "\n";
96      }
97  }
```

J. If You Are Homeless... Just Build a House (Eolymp)

Sean $v_1, v_2, \ldots, v_{hw}$ los valores ordenados de las celdas sobre las que estás considerando construir la casa. Entonces, el valor óptimo al que igualar las celdas es la mediana de v. Es un hecho conocido y bastante útil. La demostración es simple también. Si el valor actual al que quieres igualar las celdas es x y hay k celdas con valores mayores que x. Entonces, incrementar x por 1 sumaría al coste $(hw-k)-k$, porque se reduce en 1 el coste de igualar todas las celdas mayores y se aumenta en 1 el coste de igualar las menores o iguales, que son $hw-k$. Por lo tanto, si inicialmente asignamos $x = v_1$, la iremos incrementando mientras $hw - 2k < 0 \implies k > \frac{hw}{2}$. A partir de la cual dejará de ser óptimo aumentarlo más, porque pasaríamos a sumar al coste. Por lo tanto, el valor óptimo es $v_{\lfloor \frac{hw}{2} \rfloor}$ indexado en 0, que es la mediana de v.

Por lo tanto, lo que haremos será iterar sobre la esquina superior izquierda de la casa y mantener en un árbol de segmentos los valores de las celdas que están actualmente en la casa. El árbol de segmentos se construye sobre los posibles valores que puede tomar la celda (de 0 a 100000), y para cada valor se suma 1 por cada celda que tenga ese valor. Entonces, podemos hacer un *walking* para encontrar la mediana de los elementos y saber el valor óptimo x, y el coste para calcular la casa empezando en esa esquina superior izquierda es $x \cdot \frac{hw}{2} - sum(0, x) + sum(x + 1, 100000) - x \cdot \frac{hw}{2}$, donde $sum(l, r)$ es la suma de valores de todas las celdas con valores entre l y r que están incluidas en la casa. Hay que tener cuidado con errores de 1 en función de si hw es par o no, y si se indexa en 1 o 0.

Nota: El código va bastante lento, tarda 3664ms. Es posible que se pueda optimizar, si alguien quiere intentarlo, adelante. Además, es posible que el código compartido no sea exactamente lo que se ha explicado, no recuerdo lo que pensé en concurso y lo he vuelto a pensar de nuevo para explicarlo, y no he tenido tiempo para volver a hacer el código. Sin embargo, estoy bastante seguro de que es lo mismo o muy parecido.

```
1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  #define endl "\n"
```

```
6
7   typedef long long ll;
8   typedef long double ld;
9
10  const int MOD = 998244353;
11  const int maxA = 100000;
12
13  struct segTree{
14      vector<int> cnt;
15      vector<ll> sums;
16      vector<int> upd;
17      int siz;
18      void init(int n){
19          siz = 1;
20          while(siz < n) siz *= 2;
21          sums.assign(2 * siz, 0);
22          upd.assign(2 * siz, 0);
23          cnt.assign(2 * siz, 0);
24      }
25
26      void set_point(int i, int val, int add, int x, int lx, int rx){
27          if(rx - lx == 1){
28              sums[x] += val;
29                  if(add == -1) sums[x] -= 2 * val;
30              return;
31          }
32          int m = (lx + rx) / 2;
33          if(i < m) set_point(i, val, add, 2 * x + 1, lx, m);
34          else set_point(i, val, add, 2 * x + 2, m, rx);
35          sums[x] = sums[2 * x + 1] + sums[2 * x + 2];
36      }
37
38      void set(int l, int r, int add, int x, int lx, int rx){
39          if(lx >= l && rx <= r){
40              upd[x] += add;
41              cnt[x] += add;
42              return;
43          }else if(lx >= r || rx <= l) return;
44          int m = (lx + rx) / 2;
```

```
45          set(l, r, add, 2 * x + 1, lx, m);
46          set(l, r, add, 2 * x + 2, m, rx);
47          cnt[x] = max(cnt[2 * x + 1], cnt[2 * x + 2]) + upd[x];
48      }
49
50      void set(int l, int r, int add){
51          set_point(l, l, add, 0, 0, siz);
52          set(l, r, add, 0, 0, siz);
53      }
54
55      pair<int, int> walk(int want, int x, int lx, int rx, int crr){
56          if(rx - lx == 1){
57              return {lx, cnt[x]};
58          }
59          int m = (lx + rx) / 2;
60          crr += upd[x];
61          pair<int, int> mine;
62          if(cnt[2 * x + 1] + crr >= want){
63              mine = walk(want, 2 * x + 1, lx, m, crr);
64          }else mine = walk(want, 2 * x + 2, m, rx, crr);
65          mine.second += upd[x];
66          return mine;
67      }
68
69      pair<int, int> walk(int want){
70          return walk(want, 0, 0, siz, 0);
71      }
72
73      ll sum(int l, int r, int x, int lx, int rx){
74          if(lx >= l && rx <= r) return sums[x];
75          else if(lx >= r || rx <= l) return 0;
76          int m = (lx + rx) / 2;
77          ll c1 = sum(l, r, 2 * x + 1, lx, m);
78          ll c2 = sum(l, r, 2 * x + 2, m, rx);
79          return c1 + c2;
80      }
81
82      ll sum(int l, int r){
83          return sum(l, r, 0, 0, siz);
```

```
    }
};

void solve(){
    int n, m; cin >> n >> m;
    int h, w; cin >> h >> w;
    vector<vector<int>> v(n, vector<int>(m));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> v[i][j];
        }
    }
    int ind = (h * w) / 2;
    if((h * w) % 2 == 1) ind++; //1-indexed.
    segTree st; st.init(maxA + 1);
    ll ans = 1e18;
    for(int i = 0; i < n; i++){
        if(i + h - 1 >= n) break;
        for(int j = 0; j < m; j++){
            if(j == 0){
                for(int i1 = i; i1 < i + h; i1++){
                    for(int j1 = j; j1 < j + w; j1++){
                        st.set(v[i1][j1], maxA + 1, 1);
                    }
                }
            }else{
                for(int i1 = i; i1 < i + h; i1++){
                    st.set(v[i1][j-1], maxA + 1, -1);
                    st.set(v[i1][j + w - 1], maxA + 1, 1);
                }
            }
            pair<ll, int> p = st.walk(ind);
            ll val = p.first;
            int cnt = p.second;
            ll cost = cnt * val - st.sum(0, val+1)
                + st.sum(val + 1, maxA + 1) - (h * w - cnt) * val;
            ans = min(ans, cost);
            if(j + w == m){
                for(int i1 = i; i1 < i + h; i1++){
```

```
                    for(int j1 = j; j1 < m; j1++){
                        st.set(v[i1][j1], maxA + 1, -1);
                    }
                }
                break;
            }
        }
    }
    cout << ans << endl;
}

signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    //~ int tt; cin >> tt;
    int tt = 1;
    for(int t = 1; t <= tt; t++){
        solve();
    }
}
```

K. Jousting Tournament (IOI 2012)

AVISO: Hubo un error y se puso involuntariamente este problema que usa una técnica conocida como *sweep line*, que daremos en la siguiente clase de teoría. Sin embargo, no es un gran problema, ya que este era el problema más difícil con diferencia, y quién haya llegado hasta este ya va bastante avanzado.

El primer paso para resolver el problema es cambiar de forma los rangos $[S[i], E[i]]$ a algo más cómodo, porque si depende del estado del arreglo en ese entonces se complica mucho el problema. Por lo tanto, lo primero que intentaremos hacer será pasar las parejas $(S[i], E[i])$ a parejas (l, r) tal que el intervalo de caballeros entre l y r contenga todos los caballeros que luchan en esa ronda o que perdieron contra un caballero que lucha en esa ronda, definido recursivamente. Es decir, que si el caballero a perdió contra b, y b perdió contra c, y c está entre $S[i]$ y $E[i]$, entonces a también debe estar en $[l, r]$. Esta transformación siempre puede hacerse, puesto que todos los caballeros que nos interesan siempre forma un rango contiguo del arreglo original.

Para hacerlo, lo que haremos será, cada vez que se hace una ronda entre todos los caballeros $[l, r]$ del arreglo original, solo 1 de ellos seguirá en pie, por lo que $r - l$ de ellos dejarán de estar. Por comodidad, podemos asumir que l seguirá en pie y todos los caballeros en $[l + 1, r]$ están fuera. Si mantenemos un árbol de segmentos con los caballeros que siguen en pie (una posición tiene un valor de 1 si ese caballero sigue en pie, y 0 en caso contrario), podemos usar *walking* para saber la posición en el arreglo original del i-ésimo caballero en pie tras un cierto número de rondas (pues es la primera posición j tal que la suma de 0 a j es i). Con esto, podemos asignar l a la posición del $S[i]$-ésimo caballero en pie antes de la ronda i. Para r, sin embargo, debemos recordar que hemos asumido que el ganador de cada ronda es el l-ésimo, por lo que si asignamos a r el valor del $E[i]$-ésimo caballero en pie, estaremos descartando todos los caballeros a los que ganó el $E[i]$-ésimo caballero. Por suerte, esto tiene fácil arreglo, pues encontraremos el $(E[i] + 1)$-ésimo caballero en pie, y le restaremos 1 a su posición. De esta manera, lo que hacemos es incluir todos los caballeros abatidos entre el caballero en posición $E[i]$ y en posición $E[i] + 1$, que sabemos que fueron derrotados por el caballero $E[i]$. Una vez tenemos el rango $[l, r]$, solo resta aplicar *lazy propagation* para hacer todos los elementos en el rango $[l+1, r]$ iguales a 0, y mantener el árbol de segmentos deseado.

Una vez tenemos estos segmentos, el problema se ha reducido al siguiente. Dados C intervalos $[l_i, r_i]$ encuentra la menor posición p tal que el número de intervalos que satisfacen $p \in [l_i, r_i]$ y $R > \text{máx}_{l_i \leq j \leq r_i} K[j]$ antes de llegar a un intervalo $p \in [l_i, r_i]$ y $R < \text{máx}_{l_i \leq j \leq r_i} K[j]$ es máximo.

Para calcular esta p, podemos usar *sweep line*, de manera que iteramos sobre la p y mantenemos en todo momento los intervalos que incluyen p, que diremos que están *activos*. Para todos los intervalos activos, usaremos un árbol de segmentos para guardarnos en su índice de consulta el máximo valor en el intervalo que incluyen (notad que como asumimos que p está dentro del intervalo, de hecho no se incluye r_i, pues tendríamos un elemento de más). De esta manera, la ronda en la que se eliminaría al caballero popular sería el menor índice tal que el máximo de su intervalo es mayor a R, y podemos obtener este con *walking*. Una vez tenemos la ronda en la que se elimina el caballero popular, sabemos cuántas rondas ganará sumando todos los intervalos activos antes de tal ronda, que se puede hacer de nuevo con un árbol de segmentos de suma. Para acabar, comparamos las rondas que gana en la posición actual y el máximo número de rondas que ha ganado, y si es mayor, actualizamos el valor y la posición óptima, y hemos acabado.

Nota: Se ha explicado esta solución porque me parece la más intuitiva y la más fácil de pensar, además de que, aunque el código es largo, no es muy enrevesado. Sin embargo, hay muchas soluciones a este problema. La que hice yo inicialmente tan solo requiere dos árboles de segmentos. Otra solución que podéis consultar es esta, si os apetece.

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

using namespace std;

struct segTreeFindRange{
    vector<int> sum, upd;
    int sz;
    void build(int x, int lx, int rx, int n){
        if(rx - lx == 1){
            if(lx < n){
                sum[x] = 1;
            }
            return;
        }
```

```
        int m = (lx + rx) / 2;
        build(2 * x + 1, lx, m, n);
        build(2 * x + 2, m, rx, n);
        sum[x] = sum[2 * x + 1] + sum[2 * x + 2];
    }
    void init(int n){
        sz = 1;
        while(sz < n) sz *= 2;
        sum.assign(2 * sz, 0);
        upd.assign(2 * sz, 0);
        build(0, 0, sz, n);
    }
    void push(int x){
        if(!upd[x]) return;
        sum[2 * x + 1] = 0;
        sum[2 * x + 2] = 0;
        upd[2 * x + 1] = 1;
        upd[2 * x + 2] = 1;
        upd[x] = 0;
    }
    void setTo0(int l, int r, int x, int lx, int rx){
        if(lx >= l && rx <= r){
            sum[x] = 0;
            upd[x] = 1;
            return;
        }else if(lx >= r || rx <= l) return;
        int m = (lx + rx) / 2;
        push(x);
        setTo0(l, r, 2 * x + 1, lx, m);
        setTo0(l, r, 2 * x + 2, m, rx);
        sum[x] = sum[2 * x + 1] + sum[2 * x + 2];
    }
    void setTo0(int l, int r){
        setTo0(l, r, 0, 0, sz);
    }
    int walk(int val, int x, int lx, int rx, int curr){
        if(rx - lx == 1){
            if(curr + sum[x] == val) return lx;
            else return -1;
```

```
        }
        int m = (lx + rx) / 2;
        push(x);
        if(sum[2 * x + 1] + curr >= val){
            return walk(val, 2 * x + 1, lx, m, curr);
        }else{
            return walk(val, 2 * x + 2, m, rx, curr+sum[2 * x + 1]);
        }
    }
    int walk(int val){
        return walk(val, 0, 0, sz, 0);
    }
};

struct segTreeMax{
    vector<int> mx;
    int sz;
    void init(int n){
        sz = 1;
        while(sz < n) sz *= 2;
        mx.assign(2 * sz, 0);
    }
    void set(int i, int val, int x, int lx, int rx){
        if(rx - lx == 1){
            mx[x] = val;
            return;
        }
        int m = (lx + rx) / 2;
        if(i < m) set(i, val, 2 * x + 1, lx, m);
        else set(i, val, 2 * x + 2, m, rx);
        mx[x] = max(mx[2 * x + 1], mx[2 * x + 2]);
    }
    void set(int i, int val){
        set(i, val, 0, 0, sz);
    }
    int calc(int l, int r, int x, int lx, int rx){
        if(lx >= l && rx <= r) return mx[x];
        else if(lx >= r || rx <= l) return 0;
        int m = (lx + rx) / 2;
```

```
93          int c1 = calc(l, r, 2 * x + 1, lx, m);
94          int c2 = calc(l, r, 2 * x + 2, m, rx);
95          return max(c1, c2);
96      }
97      int calc(int l, int r){
98          if(l >= r) return 0;
99          return calc(l, r, 0, 0, sz);
100     }
101     int walk(int val, int x, int lx, int rx){
102         if(rx - lx == 1){
103             if(mx[x] > val) return lx;
104             else return -1;
105         }
106         int m = (lx + rx) / 2;
107         if(mx[2 * x + 1] > val){
108             return walk(val, 2 * x + 1, lx, m);
109         }else return walk(val, 2 * x + 2, m, rx);
110     }
111     int walk(int val){
112         return walk(val, 0, 0, sz);
113     }
114 };
115 
116 struct segTreeSum{
117     vector<int> sum;
118     int sz;
119     void init(int n){
120         sz = 1;
121         while(sz < n) sz *= 2;
122         sum.assign(2 * sz, 0);
123     }
124     void set(int i, int val, int x, int lx, int rx){
125         if(rx - lx == 1){
126             sum[x] = val;
127             return;
128         }
129         int m = (lx + rx) / 2;
130         if(i < m) set(i, val, 2 * x + 1, lx, m);
131         else set(i, val, 2 * x + 2, m, rx);
```

```
132         sum[x] = sum[2 * x + 1] + sum[2 * x + 2];
133     }
134     void set(int i, int val){
135         set(i, val, 0, 0, sz);
136     }
137     int calc(int l, int r, int x, int lx, int rx){
138         if(lx >= l && rx <= r) return sum[x];
139         else if(lx >= r || rx <= l) return 0;
140         int m = (lx + rx) / 2;
141         int c1 = calc(l, r, 2 * x + 1, lx, m);
142         int c2 = calc(l, r, 2 * x + 2, m, rx);
143         return c1 + c2;
144     }
145     int calc(int l, int r){
146         if(l >= r) return 0;
147         return calc(l, r, 0, 0, sz);
148     }
149 };
150 
151 int GetBestPosition(int N, int C, int R, int *K, int *S, int *E){
152     vector<pair<int, int>> queries(C);
153     segTreeFindRange rng; rng.init(N);
154     vector<vector<int>> starts(N), ends(N);
155     for(int i = 0; i < C; i++){
156         S[i]++; E[i]++;
157         int l = rng.walk(S[i]);
158         int r = rng.walk(E[i]+1) - 1;
159         if(r == -2) r = N - 1;
160         rng.setTo0(l+1, r+1); //I do r+1 because it's half-open.
161         queries[i] = {l, r};
162         starts[l].push_back(i);
163         ends[r].push_back(i);
164     }
165     segTreeSum stSum; stSum.init(C);
166     segTreeMax stMax; stMax.init(C);
167     segTreeMax mxRng; mxRng.init(N-1);
168     for(int i = 0; i < N-1; i++){
169         mxRng.set(i, K[i]);
170     }
```

```
171         int mxWins = 0;
172         int ans = 0;
173         for(int i = 0; i < N; i++){
174             //I assume that the knight is at position i.
175             for(int queryInd : starts[i]){
176                 stSum.set(queryInd, 1);
177                 int l = queries[queryInd].first;
178                 int r = queries[queryInd].second;
179                 stMax.set(queryInd, mxRng.calc(l, r));
180                 //I don't do r+1 because I don't want to include r
181                 //as there's the knight in the middle.
182             }
183             //Now I get the first round I would lose.
184             int ind = stMax.walk(R);
185             if(ind == -1) ind = C;
186             int wins = stSum.calc(0, ind);
187             if(mxWins < wins){
188                 mxWins = wins;
189                 ans = i;
190             }
191             for(int queryInd : ends[i]){
192                 stSum.set(queryInd, 0);
193                 stMax.set(queryInd, 0);
194             }
195         }
196         return ans;
197     }
```