Aula 10 · Árvores de Segmentos, Árvore de Segmentos Preguiçosa, Árvore de Ordenação por Fusão

Desafios de Programação

Fernando Kiotheka Vinícius Tikara Date

UFPR

24/05/2023

Árvore de Segmentos

Árvore de Segmentos

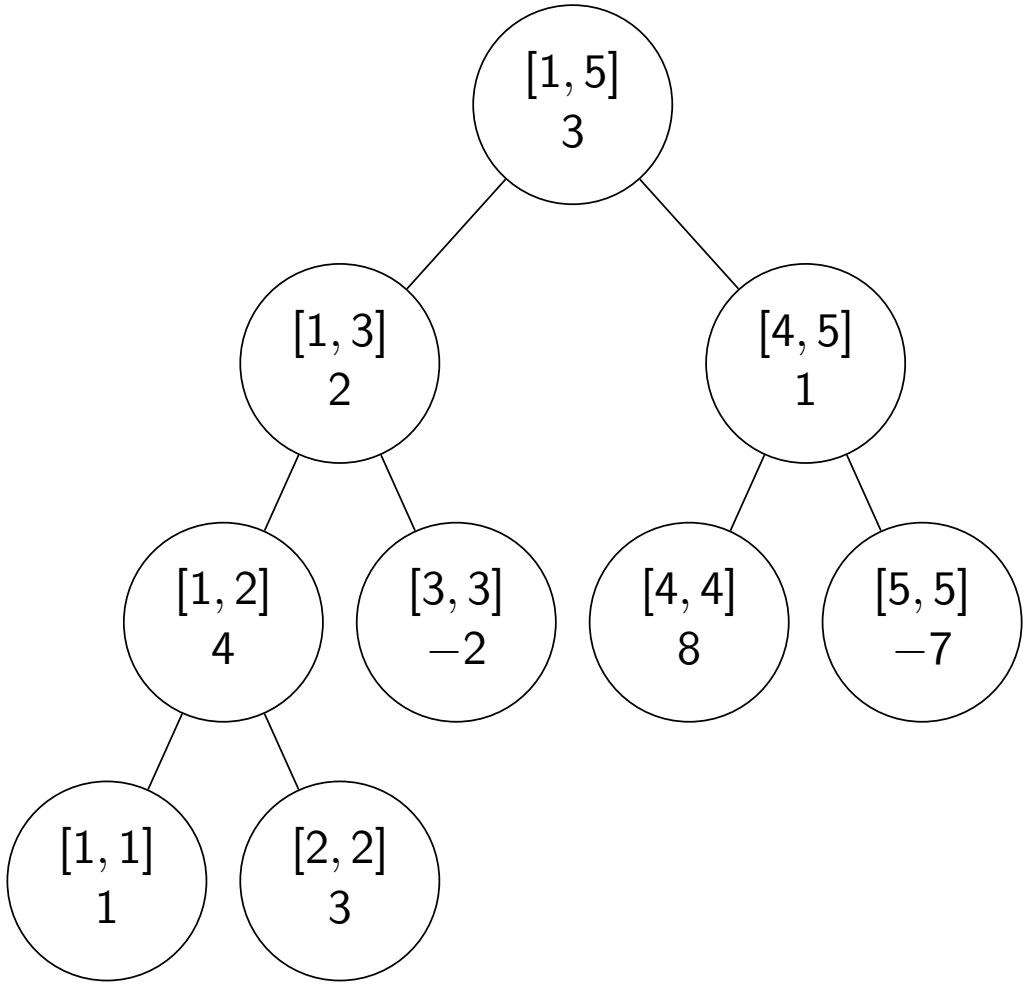
Podemos resolver este mesmo problema também com uma Árvore de Segmentos.

- Mais complicada que a BIT.
- Resolve, também, problemas mais complicados que os da BIT.
- É essencialmente o "encapsulamento" do conceito de divisão em conquista em uma estrutura.

A Árvore de Segmentos (Segment Tree, SegTree) é uma árvore binária onde cada nodo corresponde a um intervalo no seu vetor de consulta e atualização. Os nodos filhos correspondem às duas metades do intervalo.

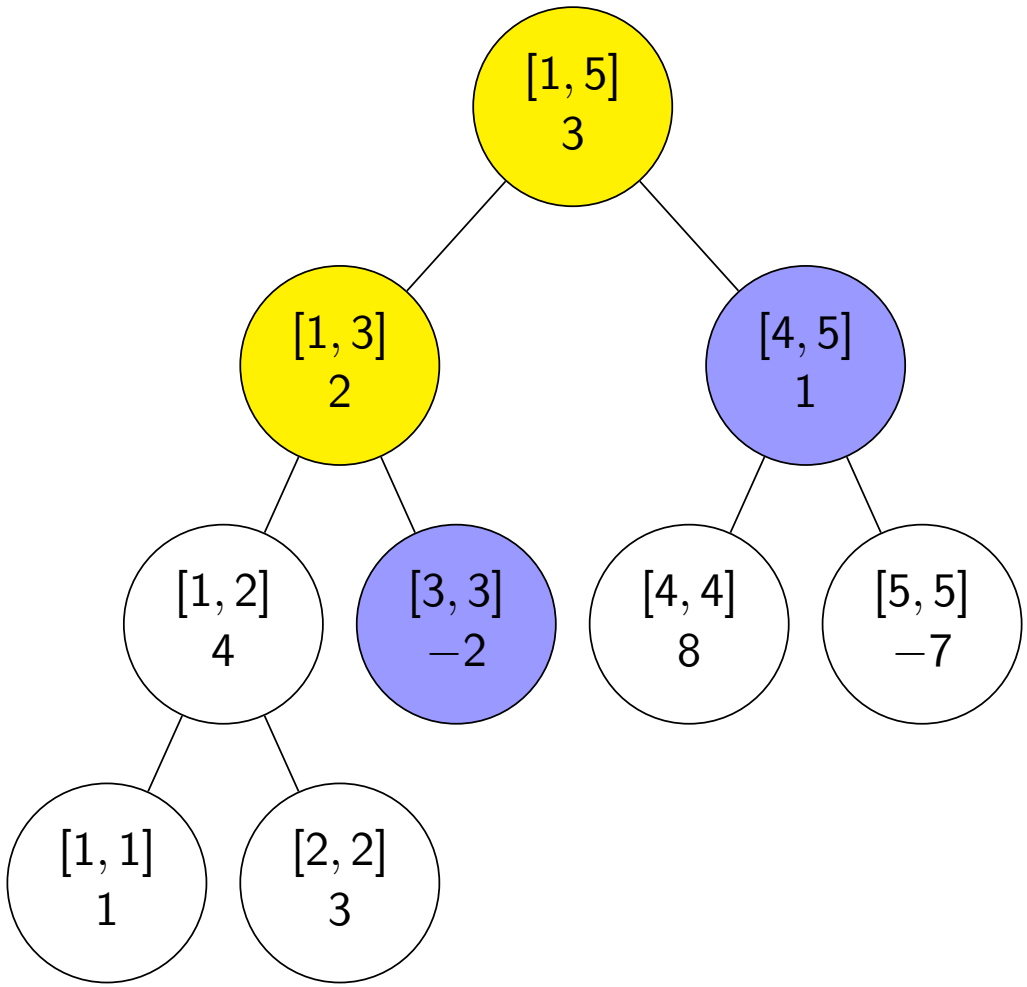
Árvore de Segmentos para um vetor de 5 elementos

Para o vetor $V = [1, 3, -2, 8, -7]$:



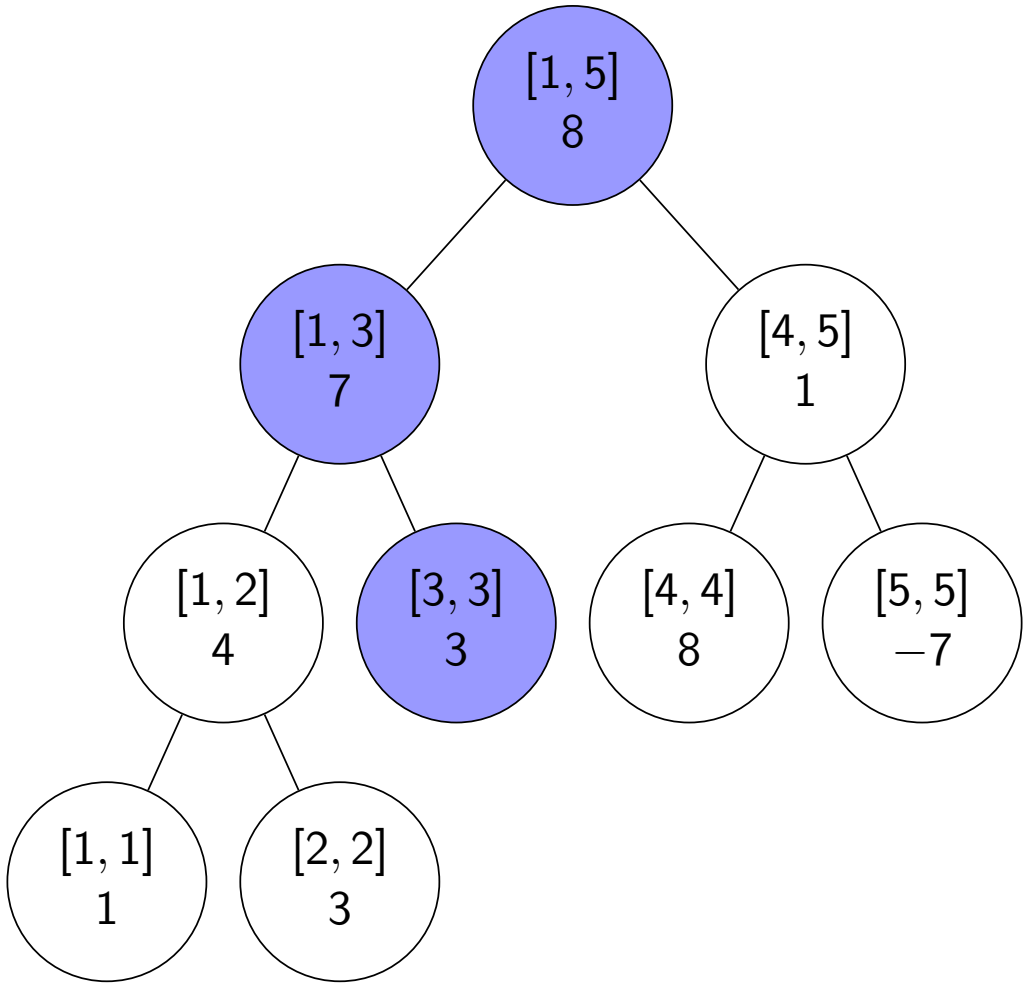
Consulta

Qual a soma do intervalo $[3, 5]$?



Atualização

Atribuindo o valor 3 ao elemento de índice 3:



Complexidade da Árvore de Segmentos

- **Construção:** $\mathcal{O}(N)$
- **Consulta:** $\mathcal{O}(\log N)$
- **Atualização:** $\mathcal{O}(\log N)$

Notas sobre a implementação recursiva

- É talvez a mais fácil de se entender e de memorizar.
- Usa muitos parâmetros nas funções pra poder guiar o código.
- Mais fácil de estender para permitir outras operações.
- Exige ao menos $4n$ memória.
- Confira mais detalhes no CP Algorithms (`https://cp-algorithms.com/data_structures/segment_tree.html`).

Consultas e alterações na árvore de segmentos recursiva

Código `strec.h`

```
vector<int> t (4*N);
int op_inclusive(int l, int r, int ti=1, int tl=1, int tr=N) {
    if (l > r) { return NEUTRAL; }
    if (l == tl && r == tr) { return t[ti]; }
    int tm = (tl + tr) / 2;
    return OP(op_inclusive(l, min(r, tm), ti*2, tl, tm),
        op_inclusive(max(l, tm+1), r, ti*2+1, tm+1, tr));
}
void set_value(int i, int v, int ti=1, int tl=1, int tr=N) {
    if (tl == tr) { t[ti] = v; return; }
    int tm = (tl + tr) / 2;
    if (i <= tm)
        set_value(i, v, ti*2, tl, tm);
    else
        set_value(i, v, ti*2+1, tm+1, tr);
    t[ti] = OP(t[ti*2], t[ti*2+1]);
}
```

Construção da árvore de segmentos recursiva

Código `strecbuild.h`

```
void build(vector<int>& src, int ti=1, int tl=1, int tr=N) {
    if (tl == tr) {
        if (tl < src.size()) { t[ti] = src[tl]; }
        return;
    }
    int tm = (tl + tr) / 2;
    build(src, ti*2, tl, tm);
    build(src, ti*2+1, tm+1, tr);
    t[ti] = OP(t[ti*2], t[ti*2+1]);
}
```

Notas sobre a versão iterativa

- O funcionamento não é exatamente o mesmo da recursiva, porém as mesmas complexidades são mantidas, e os conceitos são parecidos.
- A árvore gerada pela versão iterativa deve ser entendida mais como uma coleção de árvores binárias perfeitas.
- Exige só $2n$ de memória.
- Confira mais detalhes no blog do Codeforces (`https://codeforces.com/blog/entry/18051`).

Consultas e alterações na árvore de segmentos iterativa

Código `stit.h`

```
vector<int> t (2*N);

int op_inclusive(int l, int r) {
    r++;
    int left = NEUTRAL, right = NEUTRAL;
    for (l += N, r += N; l < r; l /= 2, r /= 2) {
        if (l & 1) left = OP(left, t[l++]);
        if (r & 1) right = OP(right, t[--r]);
    }
    return OP(left, right);
}

void set_value(int i, int v) {
    t[i += N] = v;
    for (i /= 2; i > 0; i /= 2)
        t[i] = OP(t[i*2], t[i*2+1]);
}
```

Construção da árvore de segmentos iterativa

Código `stitbuild.h`

```
void build(vector<int>& src) {
    for (int i = 1; i < src.size(); i++)
        t[N+i] = src[i];
    for (int i = N - 1; i > 0; i--)
        t[i] = OP(t[2*i], t[2*i+1]);
}
```

Usando a árvore de segmentos para somas

Código sumseg.cpp

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5+15;
#define NEUTRAL 0
#define OP(X, Y) (X + Y)
#include "stit.h"
#include "stitbuild.h"
int main() {
    int n; cin >> n; vector<int> src (n+1);
    for (int i = 1; i <= n; i++) { cin >> src[i]; }
    build(src);
    char op; int a, b, v, i; string s;
    while (cin >> op)
        if (op == '=') {
            cin >> i >> v; set_value(i, v);
        } else if (op == 'q') {
            cin >> a >> b;
            cout << op_inclusive(a, b) << "\n";
        } else getline(cin, s);
}
```

Usando a árvore de segmentos para somas (continuado)

Entrada	Saída
5 1 2 0 0 0 = 5 3 # 1 2 0 0 3 q 1 3 q 1 5 = 2 1 # 1 1 0 0 3 = 3 -3 # 1 1 -3 0 3 q 1 2 q 1 3 q 1 5 q 3 3	3 6 2 -1 2 -3

Usando a árvore de segmentos para máximo

Código `maxseg.cpp`

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5+15, oo = 987654321;
#define NEUTRAL -oo
#define OP(X, Y) max(X, Y)
#include "strec.h"
#include "strecbuild.h"
int main() {
    int n; cin >> n; vector<int> src (n+1);
    for (int i = 1; i <= n; i++) { cin >> src[i]; }
    build(src);
    char op; int a, b, v, i; string s;
    while (cin >> op)
        if (op == '=') {
            cin >> i >> v; set_value(i, v);
        } else if (op == 'q') {
            cin >> a >> b;
            cout << op_inclusive(a, b) << "\n";
        } else getline(cin, s);
}
```

Usando a árvore de segmentos para máximo (continuado)

Entrada	Saída
5 1 2 0 0 0 = 5 3 # 1 2 0 0 3 q 1 3 q 1 5 = 2 1 # 1 1 0 0 3 = 3 -3 # 1 1 -3 0 3 q 1 2 q 1 3 q 1 5 q 3 3	2 3 1 1 3 -3

Árvore de Segmentos Preguiçosa

Adição em segmento

Dado um vetor V, realizaremos operações de dois tipos:

- **Modificação:** adicionar um valor X a todos os números no intervalo $[L..R]$
- **Consulta:** consultar o valor de um número no índice I

Ideia de solução

Vamos guardar nos vértices da árvore quanto deve ser adicionado aos valores no segmento que cada um deles representa, mantendo a complexidade de $\mathcal{O}(\lg n)$.
Em consultas, basta buscar o elemento na árvore somando todos os valores ao longo do caminho.

Implementação da árvore de segmentos

Código `stsegadd.h`

```
vector<ll> t (4*N);
void add_inclusive(int l, int r, int d,
        int ti=1, int tl=1, int tr=N) {
    if (l > r) { return; }
    if (l == tl && r == tr) { t[ti] += d; return; }
    int tm = (tl + tr) / 2;
    add_inclusive(l, min(r, tm), d, ti*2, tl, tm);
    add_inclusive(max(l, tm+1), r, d, ti*2+1, tm+1, tr);
}
ll get(int i, int ti=1, int tl=1, int tr=N) {
    if (tl == tr) { return t[ti]; }
    int tm = (tl + tr) / 2;
    if (i <= tm) { return t[ti] + get(i, ti*2, tl, tm); }
    else { return t[ti] + get(i, ti*2+1, tm+1, tr); }
}
```

Implementação da construção da árvore de segmentos

Código `stsegaddbuild.h`

```
void build(vector<int>& src,
        int ti=1, int tl=1, int tr=N) {
    if (tl == tr) {
        if (tl < src.size()) { t[ti] = src[tl]; }
        return;
    }
    int tm = (tl+tr)/2;
    build(src, ti*2, tl, tm);
    build(src, ti*2+1, tm+1, tr);
    t[ti] = 0;
}
```

Utilização da adição em segmento

Código stsegadd.cpp

```
#include <bits/stdc++.h>
using namespace std; using ll = long long;
const int N = 2e5+15;
#include "stsegadd.h"
#include "stsegaddbuild.h"
int main() {
    int n; cin >> n; vector<int> src (n+1);
    for (int i = 1; i <= n; i++) { cin >> src[i]; }
    build(src);
    char op; int a, b, d, i; string s;
    while (cin >> op)
        if (op == '+') {
            cin >> a >> b >> d; add_inclusive(a, b, d);
        } else if (op == 'q') {
            cin >> i; cout << get(i) << "\n";
        } else getline(cin, s);
}
```

Utilização da adição em segmento (continuado)

Entrada	Saída
5 1 2 0 0 0 + 3 5 3 # 1 2 3 3 3 q 1 q 2 q 3 + 1 2 1 # 2 3 3 3 3 q 1 + 1 3 -2 # 0 1 1 0 3 q 1 q 2 q 3	1 2 3 2 0 1 1

Atribuição em segmento

Dado um vetor V, realizaremos operações de dois tipos:

- **Modificação:** atribuir um valor X a todos os elementos no intervalo $[L..R]$
- **Consulta:** consultar o valor de um número no índice I

Ideia de solução

- Guardar em todos os vértices uma informação adicional: se ele está totalmente preenchido com o mesmo valor
- Em vez de atualizar todos os vértices cobertos pelo intervalo $[L..R]$ da operação, atualizamos apenas alguns
- Só passamos a informação à frente quando um vértice for dividido

Ideia de solução

- Se realizarmos uma atribuição sobre o intervalo $[0..N-1]$, só a raiz da árvore será alterada
- Em seguida, realizando outra atribuição sobre o intervalo $[0..N/2]$, a raiz da árvore deve "empurrar" a informação para seus dois nós filhos

Implementação da propagação

Código `stsegatrpush.h`

```
 void push(int ti) {
    if (mark[ti]) {
        t[ti*2] = t[ti*2+1] = t[ti];
        mark[ti*2] = mark[ti*2+1] = true;
        mark[ti] = false;
    }
}
```

Implementação da árvore de segmentos

Código `stsegatr.h`

```
vector<int> t (4*N); vector<bool> mark (4*N);
#include "stsegatrpush.h"
void set_inclusive(int l, int r, int v,
       int ti=1, int tl=1, int tr=N) {
    if (l > r) { return; }
    if (l == tl && tr == r) {
       t[ti] = v; mark[ti] = true; return; }
    push(ti); int tm = (tl + tr) / 2;
    set_inclusive(l, min(r, tm), v, ti*2, tl, tm);
    set_inclusive(max(l, tm+1), r, v, ti*2+1, tm+1, tr);
}
int get(int i, int ti=1, int tl=1, int tr=N) {
    if (tl == tr) { return t[ti]; }
    push(ti); int tm = (tl + tr) / 2;
    if (i <= tm) { return get(i, ti*2, tl, tm); }
    else { return get(i, ti*2+1, tm+1, tr); }
}
```

Utilização da atribuição em segmento

Código `stsegatr.cpp`

```
#include <bits/stdc++.h>
using namespace std; using ll = long long;
const int N = 2e5+15;
#include "stsegatr.h"
#include "stsegaddbuild.h"
int main() {
    int n; cin >> n; vector<int> src (n+1);
    for (int i = 1; i <= n; i++) { cin >> src[i]; }
    build(src);
    char op; int a, b, v, i; string s;
    while (cin >> op)
        if (op == '=') {
            cin >> a >> b >> v; set_inclusive(a, b, v);
        } else if (op == 'q') {
            cin >> i; cout << get(i) << "\n";
        } else getline(cin, s);
}
```

Utilização da atribuição em segmento (continuado)

Entrada	Saída
5 1 2 0 0 0 = 4 5 3 # 1 2 0 3 3 q 1 q 2 q 3 q 4 = 1 2 1 # 1 1 0 3 3 q 1 = 2 3 2 # 1 2 2 3 3 q 1 q 2 q 4	1 2 0 3 1 1 2 3

Adição em segmento com consulta de máximo

Dado um vetor V, realizaremos operações de dois tipos:

- **Modificação:** adicionar um valor X a todos os elementos no intervalo $[L..R]$
- **Consulta:** consultar o valor máximo em um intervalo $[L..R]$

Ideia de solução

- Cada vértice da árvore guarda o valor máximo no segmento que representa e os adendos que não foram propagados ("empurrados" para os filhos)
- Realizamos a propagação tanto ao consultar quanto ao modificar

Implementação da propagação

Código `lstpush.h`

```
void push(int ti, int tl, int tm, int tr) {
    if (sety[ti] != -1) {
        t[ti*2] = sety[ti] * FACTOR(tm - tl + 1);
        lazy[ti*2] = 0; sety[ti*2] = sety[ti];
        t[ti*2+1] = sety[ti] * FACTOR(tr - (tm+1) + 1);
        lazy[ti*2+1] = 0; sety[ti*2+1] = sety[ti];
        sety[ti] = -1;
    }
    t[ti*2] += lazy[ti] * FACTOR(tm - tl + 1);
    lazy[ti*2] += lazy[ti];
    t[ti*2+1] += lazy[ti] * FACTOR(tr - (tm+1) + 1);
    lazy[ti*2+1] += lazy[ti];
    lazy[ti] = 0;
}
```

Implementação da árvore de segmentos preguiçosa

Código `lstaddset.h`

```
void set_inclusive(int l, int r, int d,
       int ti=1, int tl=1, int tr=N) {
   if (l > r) { return; }
   if (l == tl && tr == r) {
      t[ti] = ll(d) * FACTOR(tr - tl + 1);
      sety[ti] = d; lazy[ti] = 0; return; }
   int tm = (tl + tr) / 2; push(ti, tl, tm, tr);
   set_inclusive(l, min(r, tm), d, ti*2, tl, tm);
   set_inclusive(max(l, tm+1), r, d, ti*2+1, tm+1, tr);
   t[ti] = OP(t[ti*2], t[ti*2+1]);
}
void add_inclusive(int l, int r, int d,
       int ti=1, int tl=1, int tr=N) {
   if (l > r) { return; }
   if (l == tl && tr == r) {
      t[ti] += ll(d) * FACTOR(tr - tl + 1); lazy[ti] += d; return; }
   int tm = (tl + tr) / 2; push(ti, tl, tm, tr);
   add_inclusive(l, min(r, tm), d, ti*2, tl, tm);
   add_inclusive(max(l, tm+1), r, d, ti*2+1, tm+1, tr);
   t[ti] = OP(t[ti*2], t[ti*2+1]);
}
```

Implementação da árvore de segmentos preguiçosa

Código `lst.h`

```
vector<ll> t (4*N), lazy (4*N), sety (4*N, -1);
#include "lstpush.h"
#include "lstaddset.h"
ll op_inclusive(int l, int r,
       int ti=1, int tl=1, int tr=N) {
   if (l > r) { return NEUTRAL; }
   if (l <= tl && tr <= r) { return t[ti]; }
   int tm = (tl + tr) / 2; push(ti, tl, tm, tr);
   return OP(op_inclusive(l, min(r, tm), ti*2, tl, tm),
             op_inclusive(max(l, tm+1), r, ti*2+1, tm+1, tr));
}
```

Implementação da construção

Código `lstbuild.h`

```
void build(vector<int>& src,
        int ti=1, int tl=1, int tr=N) {
    if (tl == tr) {
        if (tl < src.size()) { t[ti] = src[tl]; }
        return;
    }
    int tm = (tl + tr) / 2;
    build(src, ti*2, tl, tm);
    build(src, ti*2+1, tm+1, tr);
    t[ti] = OP(t[ti*2], t[ti*2+1]);
}
```

Máximo na árvore de segmentos preguiçosa

Código `lst.cpp`

```
#include <bits/stdc++.h>
using namespace std; using ll = long long;
const int N = 1e5+15;
#define NEUTRAL 0
#define FACTOR(sz) 1
#define OP(X, Y) max(X, Y)
#include "lst.h"
#include "lstbuild.h"
int main() {
    int n; cin >> n; vector<int> src (n+1);
    for (int i = 1; i <= n; i++) { cin >> src[i]; }
    build(src);
    char op; int a, b, d; string s;
    while (cin >> op)
        if (op == '+') {
            cin >> a >> b >> d; add_inclusive(a, b, d);
        } else if (op == 'q') {
            cin >> a >> b; cout << op_inclusive(a, b) << "\n";
        } else getline(cin, s);
}
```

Máximo na árvore de segmentos preguiçosa (continuado)

Entrada	Saída
5 1 2 0 0 0 + 4 5 3 # 1 2 0 3 3 q 1 1 q 1 2 q 1 3 q 2 3 q 2 4 + 1 2 1 # 2 3 0 3 3 q 1 3 q 1 4 q 3 3 + 2 3 2 # 2 5 2 3 3 q 1 3 q 1 5	1 2 2 2 3 3 3 0 5 5

Aplicação da árvore de segmentos preguiçosa iterativa

Código `lstitapply.h`

```
ll apply(int ti, dlta d, int sz) {
    if (d.set != -1) {
        t[ti] = ll(d.set) * FACTOR(sz);
        delta[ti] = { 0, d.set };
    }
    if (d.add != 0) {
        t[ti] += ll(d.add) * FACTOR(sz);
        delta[ti].add += d.add;
    }
    return t[ti];
}
```

Empurra/Puxa da árvore de segmentos preguiçosa iterativa

Código `lstitpushpull.h`

```
void pull(int i) {
    for (int s = __builtin_ctz(i)+1; s < L; s++) {
        int ti = i >> s;
        t[ti] = OP(t[2*ti], t[2*ti+1]);
    }
}
void push(int i) {
    int sz = 1 << (L-1);
    for (int s = L; s > 0; s--, sz /= 2) {
        int ti = i >> s;
        apply(2*ti, delta[ti], sz);
        apply(2*ti+1, delta[ti], sz);
        delta[ti] = {};
    }
}
```

Aplicação da árvore de segmentos preguiçosa iterativa

Código `lstitapplyinc.h`

```
void apply_inclusive(int l, int r,
        char op = '\0', ll x = 0) {
    r++;
    dlta d;
    if (op == '+') { d.add = x; }
    if (op == '=') { d.set = x; }
    int tl = l += N, tr = r += N, sz = 1;
    push(tl); push(tr);
    for (; l < r; l /= 2, r /= 2, sz *= 2) {
        if (l & 1) { apply(l++, d, sz); }
        if (r & 1) { apply(--r, d, sz); }
    }
    pull(tl); pull(tr);
}
void add_inclusive(int l, int r, ll d) {
    apply_inclusive(l, r, '+', d);
}
```

Árvore de segmentos preguiçosa iterativa

Código `lstit.h`

```
const int L = ceil(log2(N));
struct dlta { int add = 0, set = -1; };
vector<ll> t (2*N); vector<dlta> delta (2*N);
#include "lstitapply.h"
#include "lstitpushpull.h"
#include "lstitapplyinc.h"
ll op_inclusive(int l, int r) {
    r++;
    int tl = l += N, tr = r += N, sz = 1;
    push(tl); push(tr);
    ll ans = NEUTRAL;
    for (; l < r; l /= 2, r /= 2, sz *= 2) {
        if (l & 1) { ans = OP(ans, apply(l++, dlta(), sz)); }
        if (r & 1) { ans = OP(ans, apply(--r, dlta(), sz)); }
    }
    pull(tl); pull(tr);
    return ans;
}
```

Construção da árvore de segmentos preguiçosa iterativa

Código `lstitbuild.h`

```
void build(vector<int>& src) {
    for (int i = 1; i < src.size(); i++)
        t[N+i] = src[i];
    for (int ti = N-1; ti > 0; ti--)
        t[ti] = OP(t[2*ti], t[2*ti+1]);
}
```

Máximo na árvore de segmentos preguiçosa iterativa

Código `lstit.cpp`

```
#include <bits/stdc++.h>
using namespace std; using ll = long long;
const int N = 2e5+15;
#define NEUTRAL 0
#define FACTOR(sz) 1
#define OP(X, Y) max(X, Y)
#include "lstit.h"
#include "lstitbuild.h"
int main() {
    int n; cin >> n; vector<int> src (n+1);
    for (int i = 1; i <= n; i++) { cin >> src[i]; }
    build(src);
    char op; int a, b, d; string s;
    while (cin >> op)
        if (op == '+') {
            cin >> a >> b >> d; add_inclusive(a, b, d);
        } else if (op == 'q') {
            cin >> a >> b; cout << op_inclusive(a, b) << "\n";
        } else getline(cin, s);
}
```

Máximo na árvore de segmentos preguiçosa iterativa (continuado)

Entrada	Saída
5 1 2 0 0 0 + 4 5 3 # 1 2 0 3 3 q 1 1 q 1 2 q 1 3 q 2 3 q 2 4 + 1 2 1 # 2 3 0 3 3 q 1 3 q 1 4 q 3 3 + 2 3 2 # 2 5 2 3 3 q 1 3 q 1 5	1 2 2 2 3 3 3 0 5 5

Somas na árvore de segmentos preguiçosa

- Para fazer somas, é necessário mudar o `FACTOR(sz)` para `sz` para somar corretamente no nó de forma preguiçosa.
- Isso significa que em nós colocamos o valor certo em nós mais acima onde atualizar a sua soma significa somar multiplicando pelo número de nós que aquele nó representa.
- No caso do mínimo e do máximo isso não era necessário pois o máximo e o mínimo não acumulavam, apenas pegavam um dos valores.

Soma na árvore de segmentos preguiçosa

Código `lstsum.cpp`

```
#include <bits/stdc++.h>
using namespace std; using ll = long long;
const int N = 1e5+15;
#define NEUTRAL 0
#define FACTOR(sz) (sz)
#define OP(X, Y) (X + Y)
#include "lst.h"
#include "lstbuild.h"
int main() {
    int n; cin >> n; vector<int> src (n+1);
    for (int i = 1; i <= n; i++) { cin >> src[i]; }
    build(src);
    char op; int a, b, d; string s;
    while (cin >> op)
        if (op == '+') {
            cin >> a >> b >> d; add_inclusive(a, b, d);
        } else if (op == 'q') {
            cin >> a >> b; cout << op_inclusive(a, b) << "\n";
        } else getline(cin, s);
}
```

Soma na árvore de segmentos preguiçosa (continuado)

Entrada	Saída
5 1 2 0 0 0 + 4 5 3 # 1 2 0 3 3 q 1 1 q 1 2 q 1 3 q 2 3 q 2 4 + 1 2 1 # 2 3 0 3 3 q 1 3 q 1 4 q 3 3 + 2 3 2 # 2 5 2 3 3 q 1 3 q 1 5	1 3 3 2 5 5 8 0 9 15

E isso é tudo!

- O conteúdo que vocês precisam para fazer a competição que logo começará está todo aqui.
- Existem versões mais avançadas da Árvore de Segmentos! Variações populares incluem:
  - Árvore de Segmentos Beats (do anime "Angel Beats").
  - Árvore de Segmentos Persistente.
- Bons estudos!