Estruturas de dados

BIT e Segtree

BIT
Binary Indexed Tree
Fenwick Tree

- Introdução a soma de prefixo
- Introdução a BIT
- Update em range e query em ponto.
- Busca binária na BIT
- Soma de Prefixo 2D
- BIT 2D
- BIT 2D Esparsa
- BIT de min, max, gcd, etc

Introdução a soma de prefixo

Motivação: dado um array V de N elementos, responda Q perguntas: qual a soma dos elementos no intervalo [L..R]?

- Representação da pergunta: Qi(L, R) = V[L] + v[L + 1] + .. + V[R-1] + v[R]
- Solução direta: **O(N*Q)**
- Exemplo (1-indexado):

  V = [1, 4, 2, 1, 3, -1, 10, 2, -5]

  Q(1, 3) = 7

  Q(4, 7) = 13

Introdução a soma de prefixo

- **Relação entre intervalo e prefixo**
  - Q(L, R) = S(R) - S(L - 1), onde S(k) = V[1] + V[2] + ... + V[k]
- Explicação
  - S(R) = V[1] + V[2] + ... V[L - 1] + V[L] + V[L+1] + ... + V[R - 1] + V[R]
  - S(L - 1) = V[1] + V[2] + ... V[L - 1]
  - Q(L, R) = S(R) - S(L-1) = V[L] + V[L+1] + ... + V[R - 1] + V[R]
- Construção
  - S[1] = V[1]
  - S[i] = S[i-1] + V[i], para i > 1
- Encontrando qualquer range de soma igual a K. Questão:
  - Questão: https://olimpiada.ic.unicamp.br/pratique/p2/2019/f1/soma/
  - Solução direta: **O(N^2)**
  - Solução usando soma de prefixo: O(N) ou O(N*log(N))
  - Solução usando two-pointer (Só funciona para vetores com v[i] >= 0, cuidado com os 0's)

Solução para a questão soma - OBI

- Queremos: Q(L, R) = k
- Como, Q(L, R) = S(R) - S(L), temos que S(R) - S(L) = k
- Se fixarmos o R, só vamos precisar encontrar um S(L), tal que L < R e que S(L) = S(R) - k.
- Então só precisamos manter em uma estrutura o valor da soma de todos os prefixos S(L), tal que L < R.
- Exemplo: S(R) = 13 e k = 3. Vamos precisar contar quantos S(L) = 10 existem.

Solução para a questão soma - OBI

- Queremos: Q(L, R) = k
- Como, Q(L, R) = S(R) - S(L), temos que S(R) - S(L) = k
- Se fixarmos o R, só vamos precisar encontrar um S(L), tal que L < R e que S(L) = S(R) - k.
- Então só precisamos manter em uma estrutura o valor da soma de todos os prefixos S(L), tal que L < R.
- Exemplo: S(R) = 13 e k = 3. Vamos precisar contar quantos S(L) = 10 existem.

```
1   int v[MAXN], n, k;
2   int main() {
3     cin >> n >> k;
4
5     for(int i=0; i<n; i++)
6       cin >> v[i];
7
8     ll ans = 0, sum = 0;
9     map<ll, int> mp;
10
11    for(int i=0; i<n; i++){
12      sum += v[i];
13      if(sum == k)
14        ans++;
15
16      ans += mp[sum-k];
17      mp[sum]++;
18    }
19
20    cout << ans << endl;
21    return 0;
22  }
```

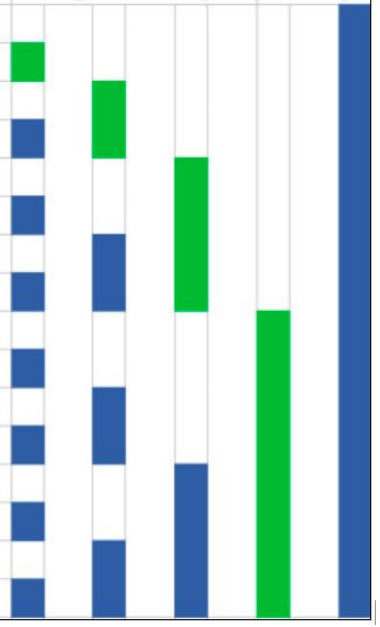
Complexidade da soma de prefixo

- Build: O(N)
- Query: O(1)
- Update: O(N)

Binary Indexed Tree

- Soma de prefixo:
  - S[i] = V[1] + V[2] + V[3] + ... + V[i]
- BIT:
  - B[i] = V[i - (2^P(i)) + 1] + V[i - (2^k) + 2] + ... + V[i], onde P(i) é a posição do bit menos significativo de i em binário e 0-indexado.
  - B[i] guarda a soma de 2^P(i) elementos.
  - Apesar da definição complicada, a implementação é muito pequena (apesar de confusa). Normalmente, no início, a maioria das pessoas só decoram o código.
  - Exemplo: B[14] = V[14 - 2^1 + 1] + V[14 - 2^1 + 2] = V[13] + V[14]
    - $14_{10} = 1110_2 = 2^3 + 2^2 + 2^1$
    - P(14) = 1
  - .Exemplo: B[12] = V[9] + V[10] + V[11] + V[12]
    - $12_{10} = 1100_2 = 2^3 + 2^2$
    - P(12) = 2

Binary Indexed Tree

number	binary representation	range of responsibility
16	10000	
15	01111	
14	01110	
13	01101	
12	01100	
11	01011	
10	01010	
9	01001	
8	01000	
7	00111	
6	00110	
5	00101	
4	00100	
3	00011	
2	00010	
1	00001	

Fonte:
https://leetcode.com/discuss/general-discussion/1093346/introduction-to-fenwick-treebinary-indexed-treebit

Binary Indexed Tree

- Assuma que a bit foi construída de forma mágica.
- Query(i) = soma do prefixo até i
  - Vamos calcular o S(14) usando a BIT:
  - S(14) = B(14) + S(14 - 2^P(14)) = B(14) + S(14 - 2^1) = B(14) + S(12)
  - S(12) = B(12) + S(12 - 2^P(12)) = B(12) + S(12 - 2^2) = B(12) + S(8)
  - S(8) = B(8) + S(8 - 2^P(8)) = B(8) + S(8 - 2^3) = B(8) + S(0)
  - S(0) = 0
  - Resultado = S(14) = B(14) + B(12) + B(8)
- Na query, vamos subtraindo o bit menos significativo.
  - Em C++, dá para encontrar o bit menos significativo em O(1).

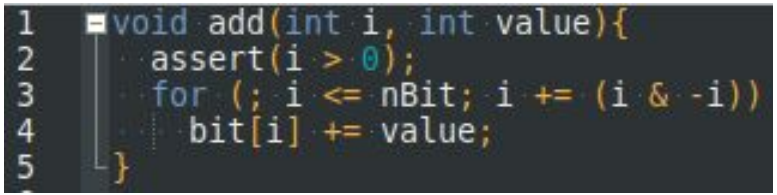
Binary Indexed Tree

- Assuma que a bit foi construída de forma mágica.
- Query(i) = soma do prefixo até i
  - Vamos calcular o S(14) usando a BIT:
  - S(14) = B(14) + S(14 - 2^P(14)) = B(14) + S(14 - 2^1) = B(14) + S(12)
  - S(12) = B(12) + S(12 - 2^P(12)) = B(12) + S(12 - 2^2) = B(12) + S(8)
  - S(8) = B(8) + S(8 - 2^P(8)) = B(8) + S(8 - 2^3) = B(8) + S(0)
  - S(0) = 0
  - Resultado = S(14) = B(14) + B(12) + B(8)
- Na query, vamos subtraindo o bit menos significativo.
  - Em C++, dá para encontrar o bit menos significativo em O(1).

```
1   ll get(int i){
2       ll s = 0;
3       for (; i > 0; i -= (i & -i))
4           s += bit[i];
5       return s;
6   }
7   //1-indexed [l, r]
8   ll bit get(int l, int r){
9       return get(r) - get(l - 1);
10  }
```

Binary Indexed Tree

- Update(i, x) = soma x na posição i.
  - A pergunta é: quais posições foram afetadas por essa mudança?
  - É meio difícil de ver e gastaríamos muito tempo com isso.
  - Magicamente sabemos que para passar por todas essas posições afetadas basta ir somando o bit menos significativo da posição atual, ficando um código muito parecido com o da query.
  - A mágica pode ser amenizada se você printar para cada número i, as posições que ele afeta em binário.



Binary Indexed Tree

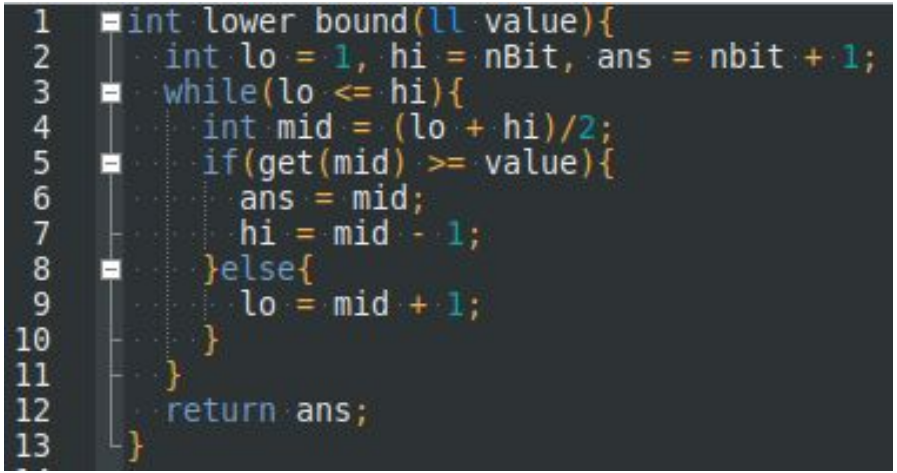
- Construir a BIT
  - O(N): Usando vetor de prefixo e a definição da BIT.
  - O(N * log(N)): assume inicialmente que todos os elementos são 0 e faz N operações de update.
- Complexidade:
  - Build: O(N)
  - Update: O(log(N))
  - Query: O(log(N))

Binary Indexed Tree

- Formas de usar a BIT:
  - Update em ponto e query em intervalo.
    - Funcionamento normal;
  - Update em range e query em ponto.
    - Update(a, b, x) = soma x em todas as posições no intervalo de [a..b]
      - Quebra em dois update: update(a, +x) e update(b+1, -x).
    - Query(i) = valor atual na posição i.
  - Update em range e query em range
    - Update(a, b, x) = soma x em todas as posições no intervalo [a..b]
    - Query(a, b) = soma dos elementos no intervalo [a..b]
    - Para resolver esse problema será necessário manter duas BITs
    - Link para quem quiser ver como implementa: https://cp-algorithms.com/data_structures/fenwick.html#3-range-updates-and-range-queries

Binary Indexed Tree

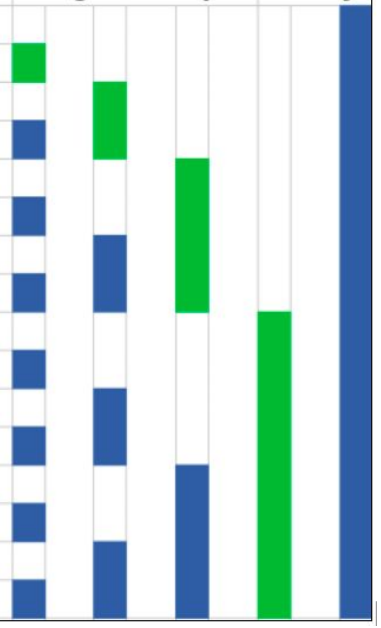
- Busca binária na BIT(lower_bound):
  - lower_bound(x): encontrar a primeira posição cujo a soma de prefixo seja maior ou igual a x. Se não existir, retorna N+1.
- Solução $O(\log^2(N))$:
  -



```
int lower bound(ll value){
    int lo = 1, hi = nBit, ans = nbit + 1;
    while(lo <= hi){
        int mid = (lo + hi)/2;
        if(get(mid) >= value){
            ans = mid;
            hi = mid - 1;
        }else{
            lo = mid + 1;
        }
    }
    return ans;
}
```

Binary Indexed Tree

- Solução O(log(N)):

number	binary representation	range of responsibility
16	10000	
15	01111	
14	01110	
13	01101	
12	01100	
11	01011	
10	01010	
9	01001	
8	01000	
7	00111	
6	00110	
5	00101	
4	00100	
3	00011	
2	00010	
1	00001	

Binary Indexed Tree

- Solução O(log(N)):
  - Erro: no lugar do p deveria ter sido pos

```
1   int lower_bound(ll value){
2       ll sum = 0;
3       int pos = 0;
4       for (int i = nLog; i >= 0; i--){
5           int newPos = p + (1<<i);
6           if ((newPos <= nBit) and (sum + bit[newPos] < value)){
7               sum += bit[newPos];
8               pos = newPos;
9           }
10      }
11      return pos + 1;
12  }
```

- Upper_bound
  - Para fazer o upper_bound basta trocar o **< value** por **<= value**.

Soma de Prefixo - 2D

- **Motivação**: dado uma matriz A de dimensão N x M, responda Q perguntas: qual a soma dos elementos A[x][y] onde Lx <= x <= Rx e Ly <= y <= Ry?
- Seja, S[a][b] = somatório de A[x][y], onde 1 <= x <= a and 1 <= y <= b.
- Podemos calcular S[a][b] usando o princípio da inclusão e exclusão:
  - S[a][b] = A[a][b] + S[a-1][b] + S[a][b-1] - S[a-1][b-1]
  - S[i][0] = S[0][j] = 0, para todo i e j.

BIT - 2D

- Vai seguir o mesmo princípio:
  - B[i][j] = Somatório de A[x][y], para todo x e y tal que i - (2^P(i)) + 1 <= x <= i e j - 2^P(j) + 1 <= y <= j. P(k) é a posição do bit menos significativo de k em binário e 0-indexado.

```
1   //1-indexed
2   t bit get(int i, int j){
3     t bit sum = 0;
4     for (int a = i; a > 0; a -= (a & -a))
5       for (int b = j; b > 0; b -= (b & -b))
6         sum += bit[a][b];
7     return sum;
8   }
9   //1-indexed
10  t bit get(int a1, int b1, int a2, int b2){
11    return get(a2, b2) - get(a2, b1 - 1) - get(a1 - 1, b2) + get(a1 - 1, b1 - 1);
12  }
13  //1-indexed [i, j]
14  void add(int i, int j, t bit value){
15    for (int a = i; a <= nBit; a += (a & -a))
16      for (int b = j; b <= mBit; b += (b & -b))
17        bit[a][b] += value;
18  }
19
```

BIT Esparsa - 2D

- Reduzindo a memória usada.
  - Ordered Set = SET + find_by_order e order_of_key
  - order_of_key(k) : o número de items estritamente menores que k.
  - find_by_order(k) : o K-th element no set (0-indexado).

BIT Esparsa - 2D

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#define mp make_pair
using namespace std;
using namespace __gnu_pbds;
using pii = pair<int, int>;
typedef tree<pii, null_type, less<pii>, rb_tree_tag, tree_order_statistics_node_update> OST;
OST bit[MAXN];
void add(int x, int y){
  for(int i = x; i < MAXN; i += i & -i)
    bit[i].insert(mp(y, x));
}
void remove(int x, int y){
  for(int i = x; i < MAXN; i += i & -i)
    bit[i].erase(mp(y, x));
}
int get(int x, int y){
  int ans = 0;
  for(int i = x; i > 0; i -= i & -i)
    ans += bit[i].order_of_key(mp(y+1, 0));
  return ans;
}
```

BIT - Mudando operações

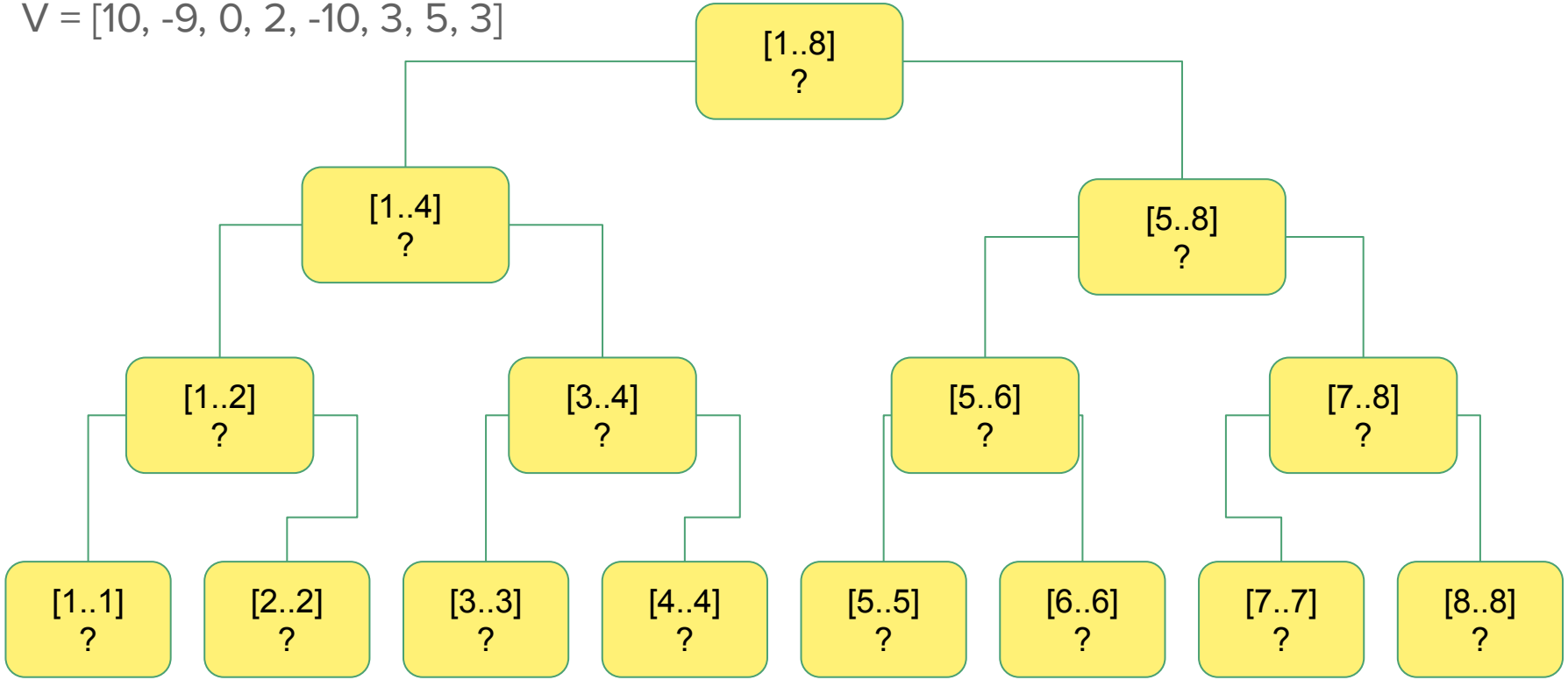
- A BIT funciona bem para todas as operações que possuem inverso.
- A BIT pode funcionar de forma limitada com outras operações
  - Só dá para fazer query de prefixo.
  - Se tiver update ele deve sempre melhorar a resposta.
    - Min -> update troca v[i] por um valor menor
    - Max -> update troca v[i] por um valor maior
    - gcd -> update troca v[i] por algum divisor de v[i]
  - A inversão para o sufixo aqui pode ser útil.
- Exemplo: https://neps.academy/br/exercise/348

SegTree
Segment Tree
Árvore de Segmento

- Introdução a Segtree
- Mudando para min, max, gcd, Kadane
- Busca binária na segtree
- Segtree com lazy
- Merge Sort Tree
- Segtree persistente
- Segtree esparsa

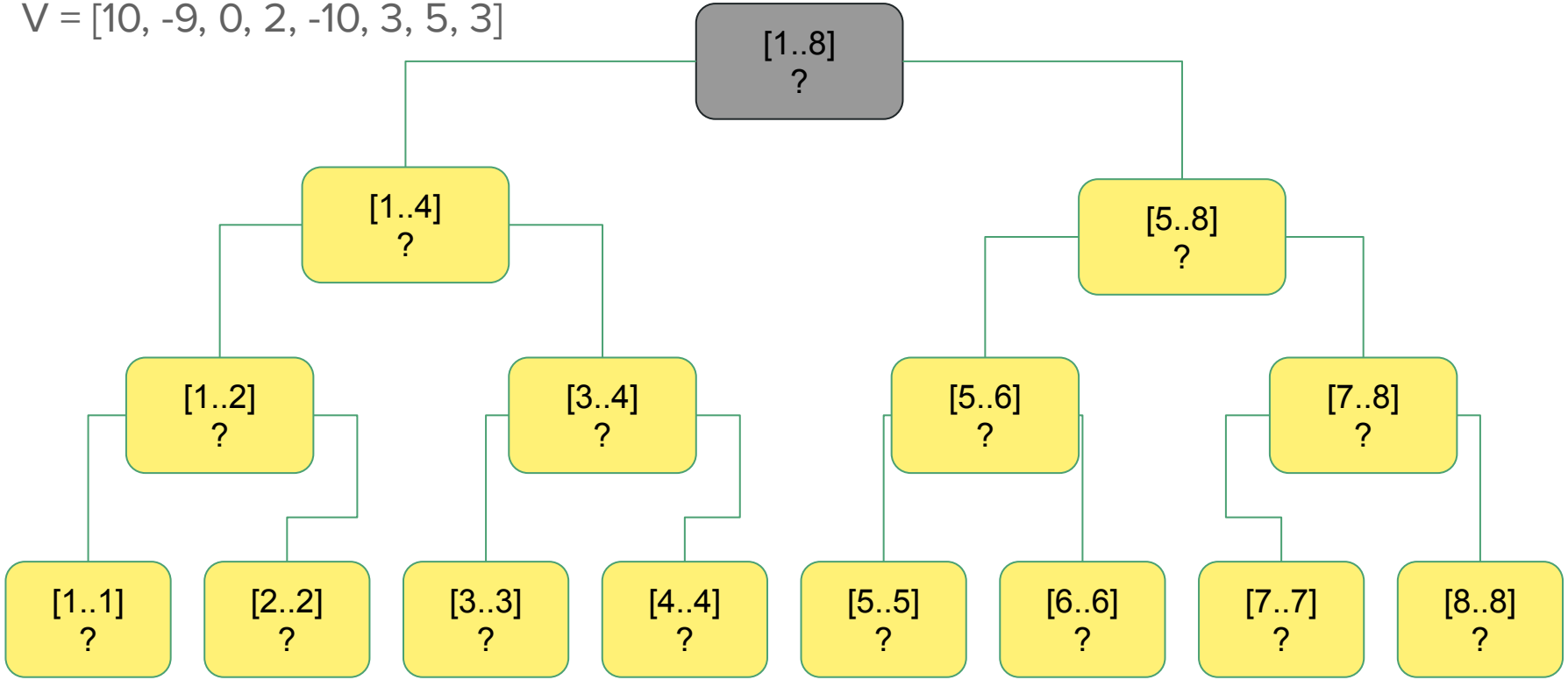
Árvore de Segmento - SegTree (exemplo: soma)

V = [10, -9, 0, 2, -10, 3, 5, 3]



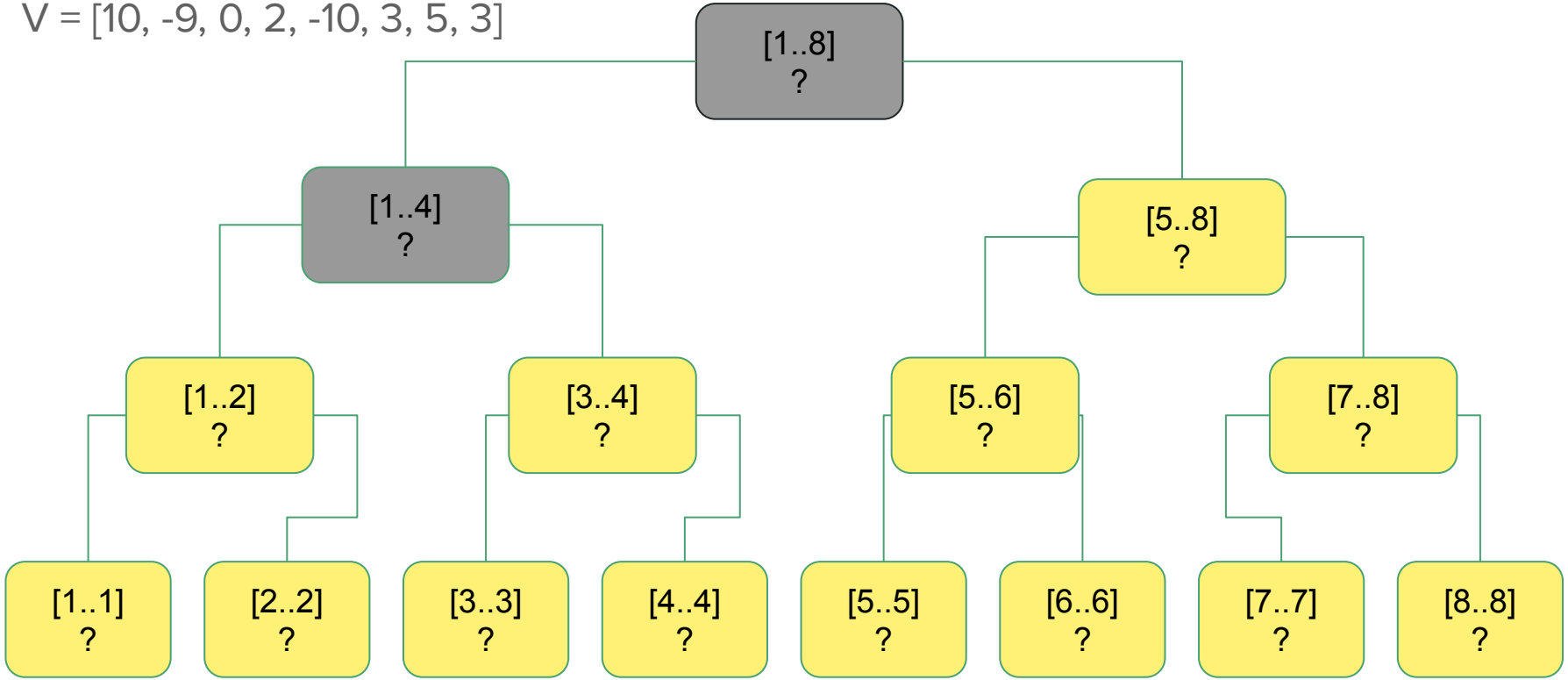
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



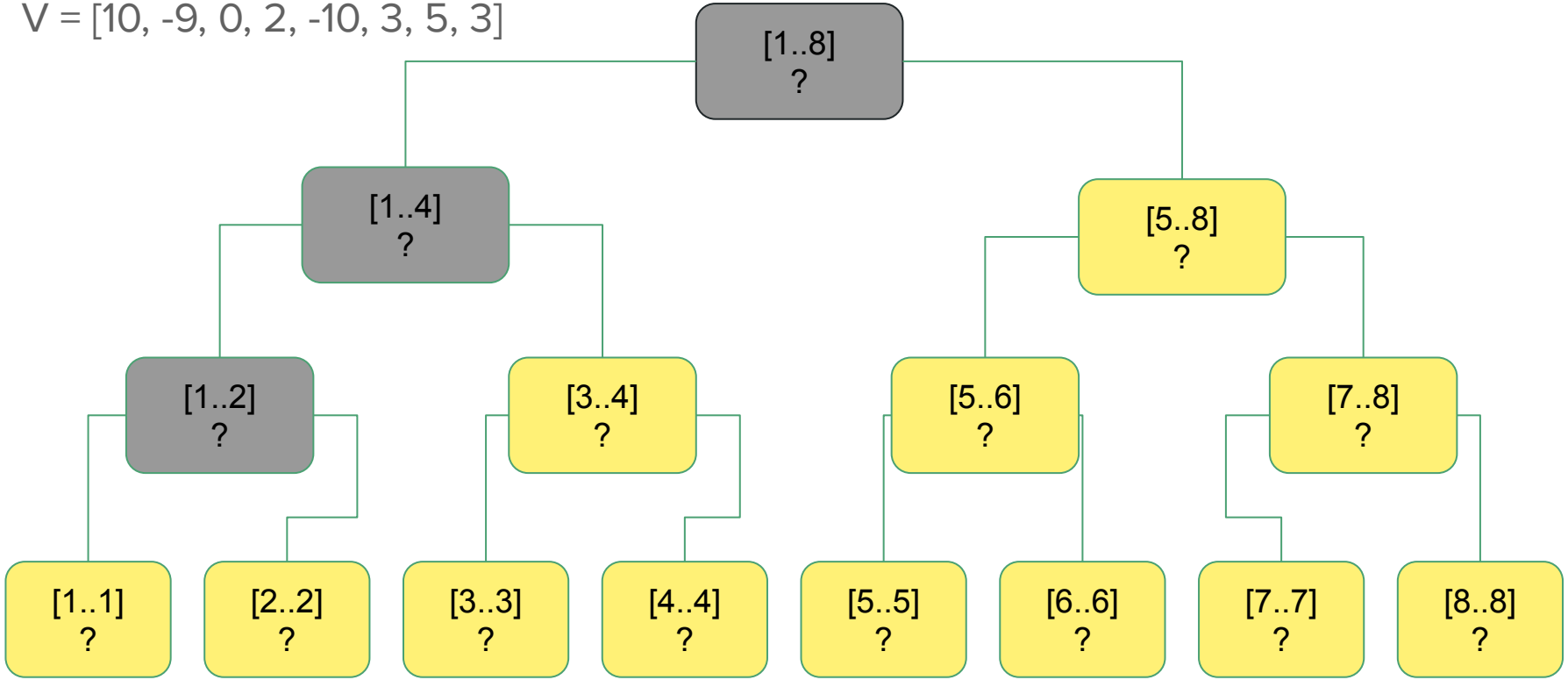
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



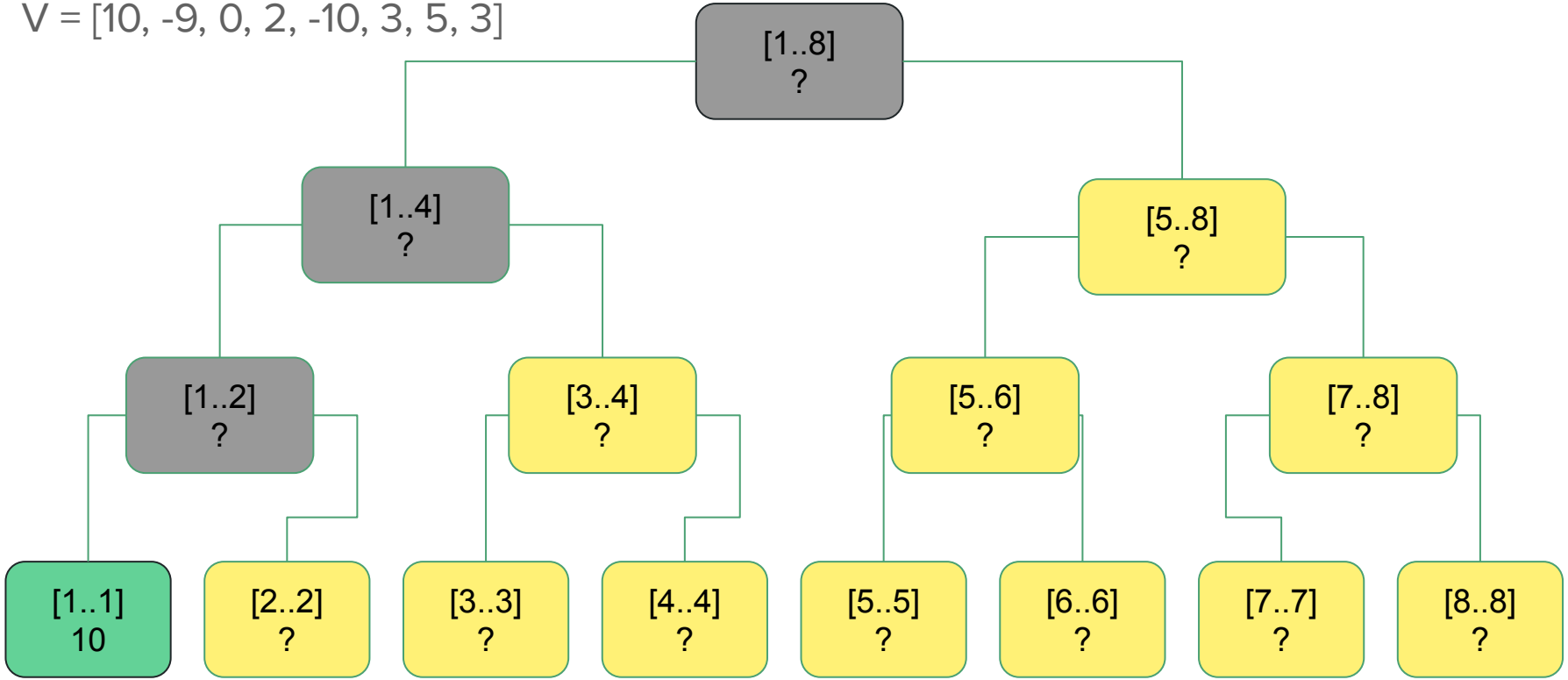
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



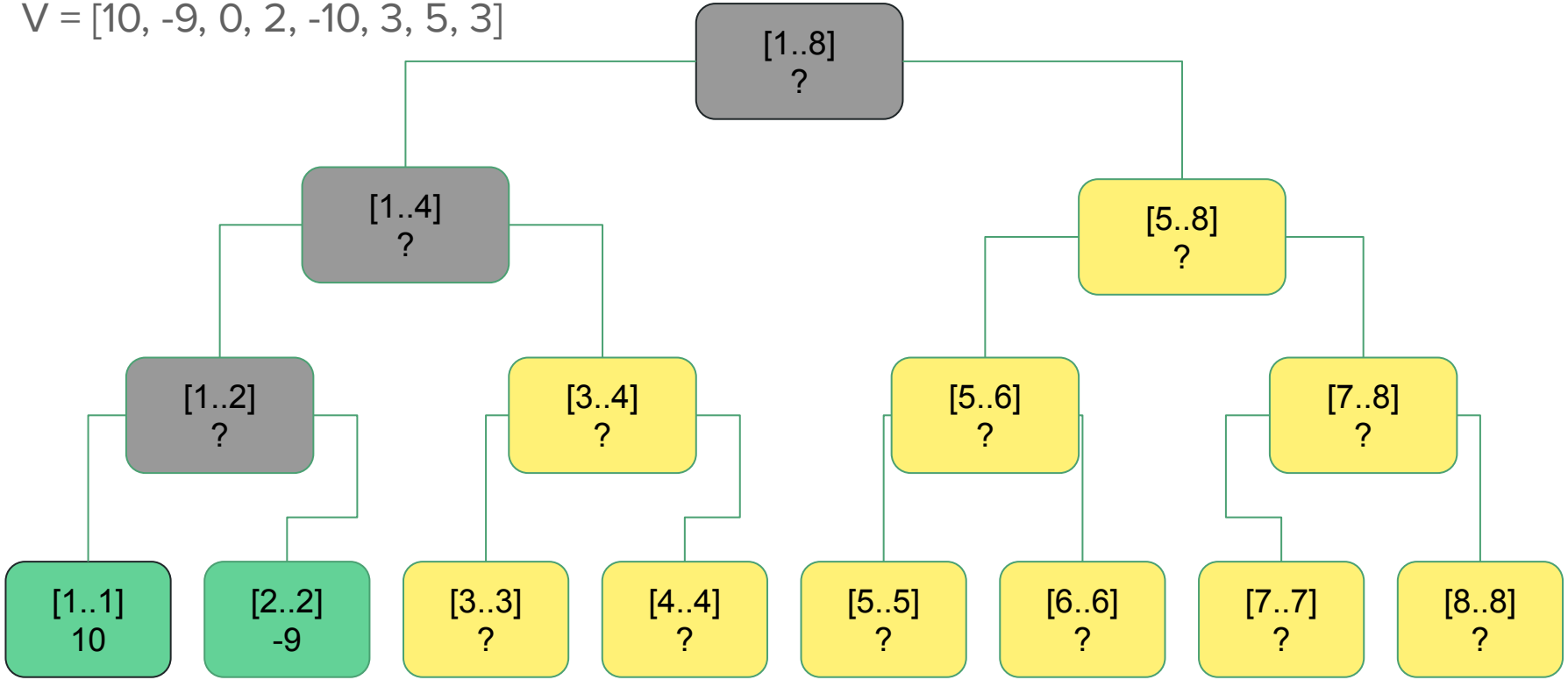
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



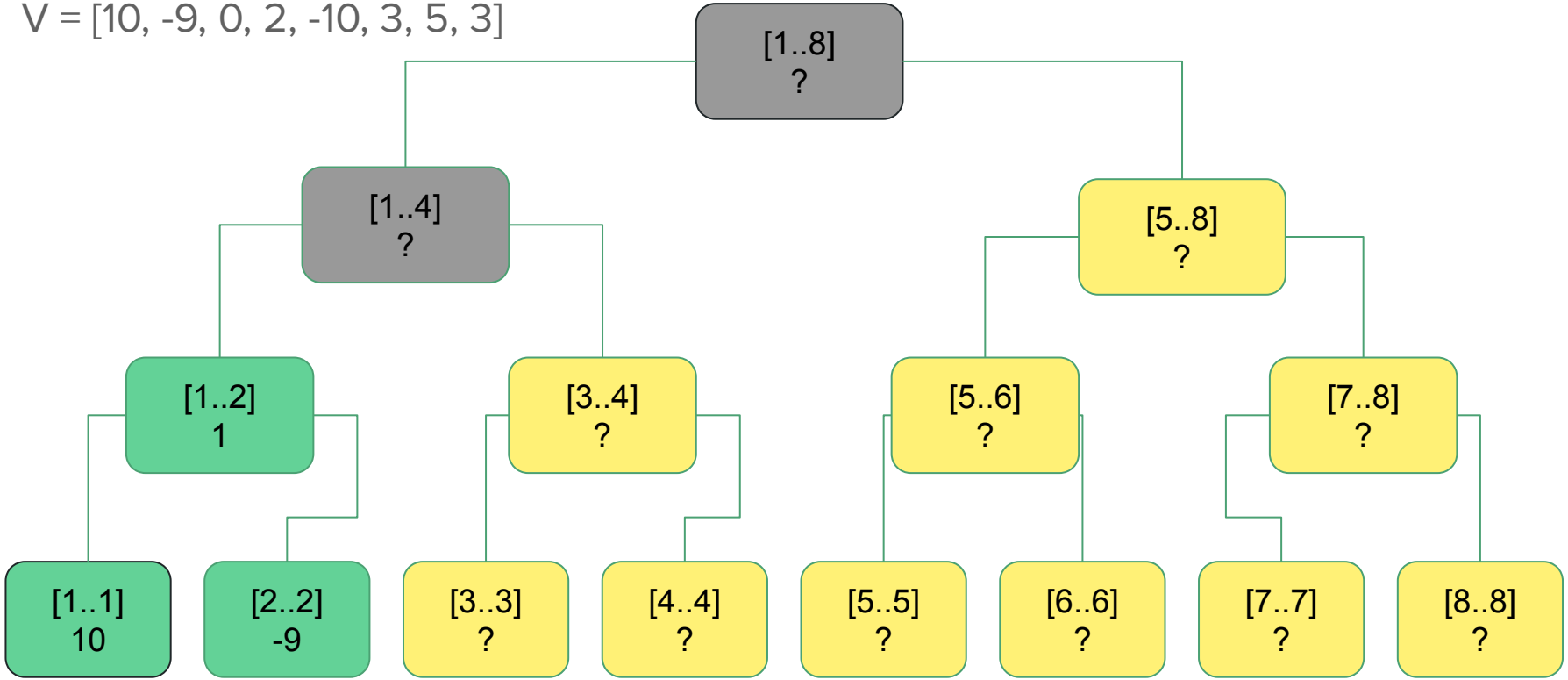
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



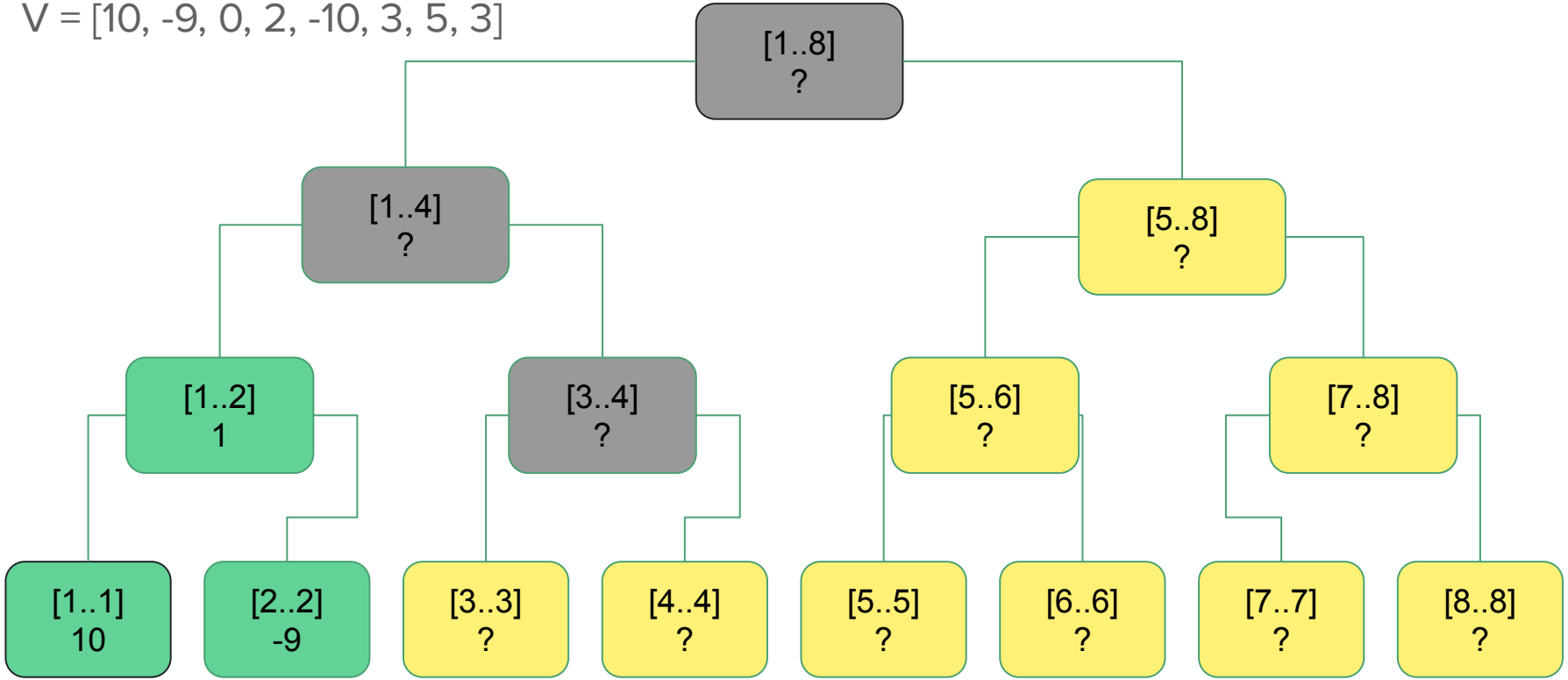
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



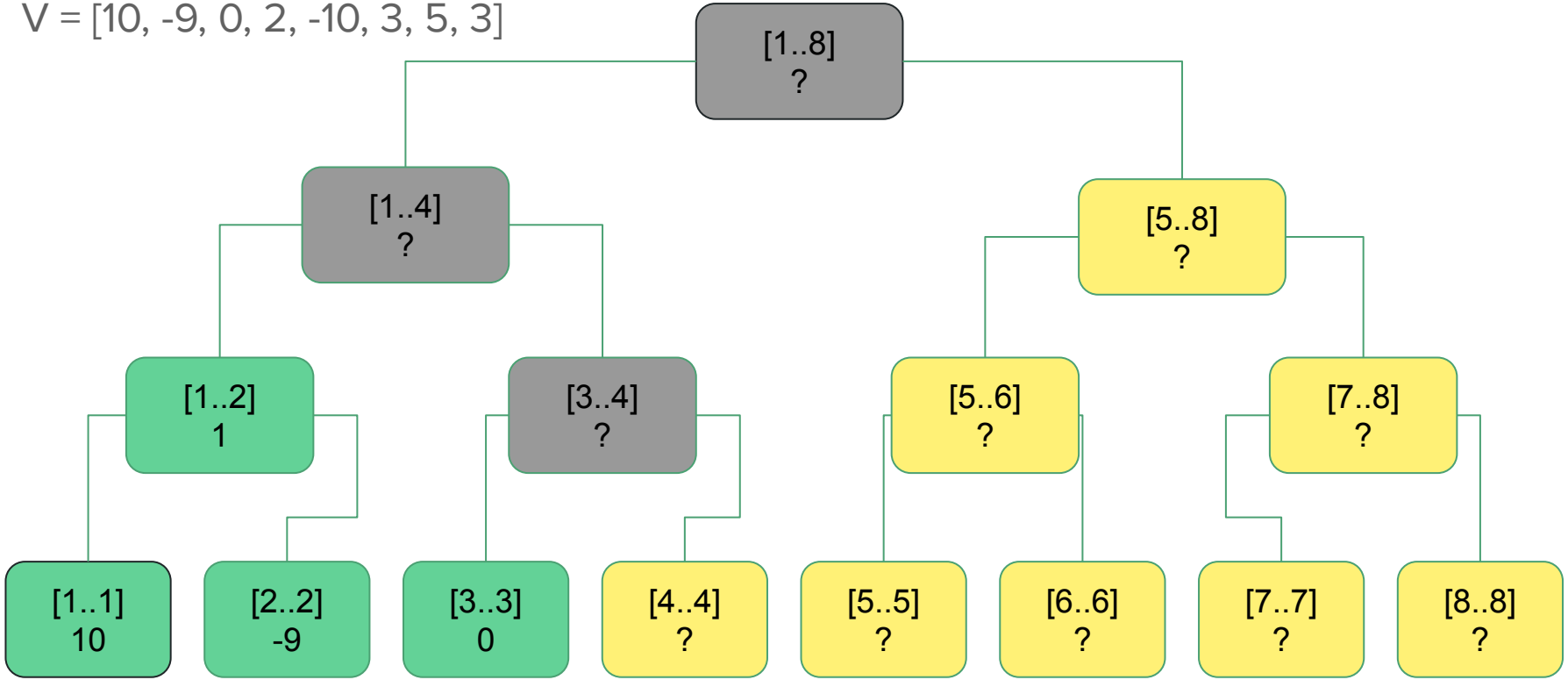
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



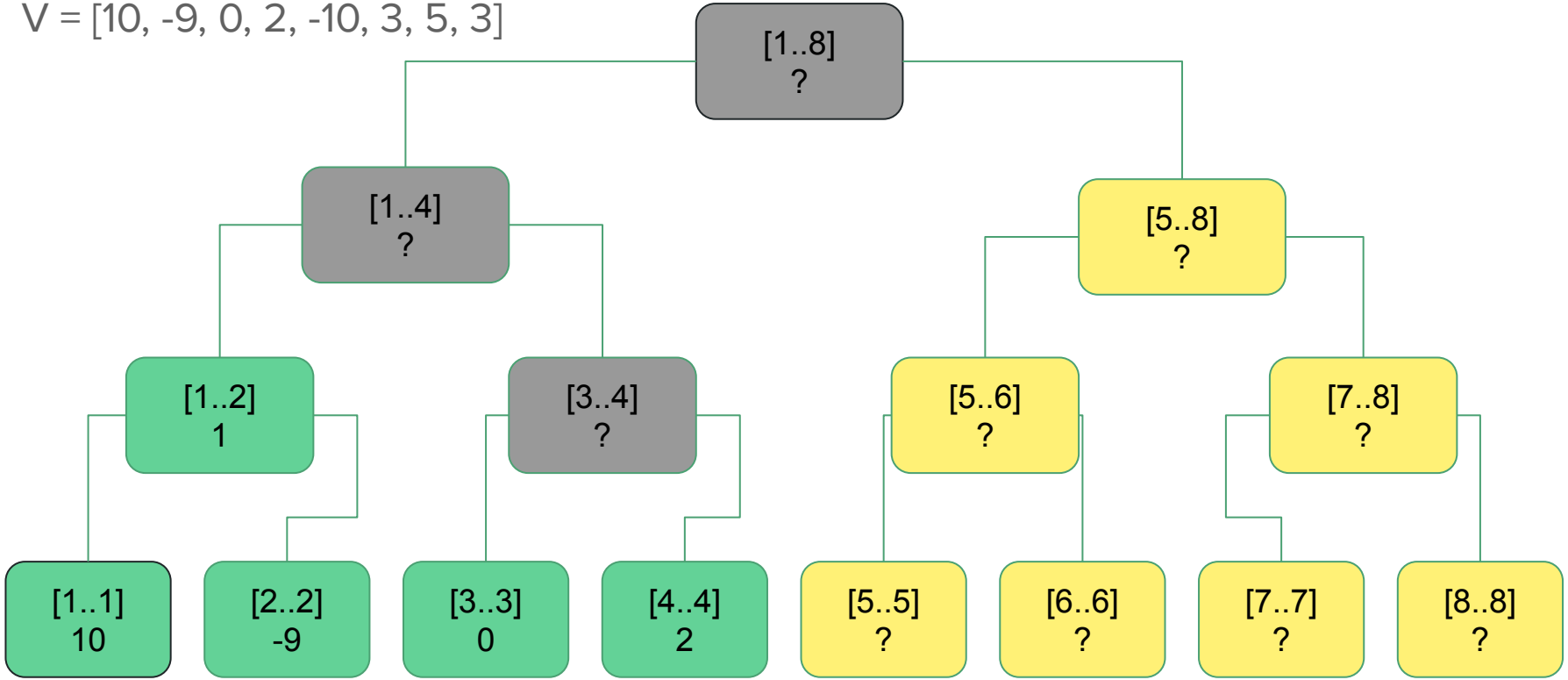
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



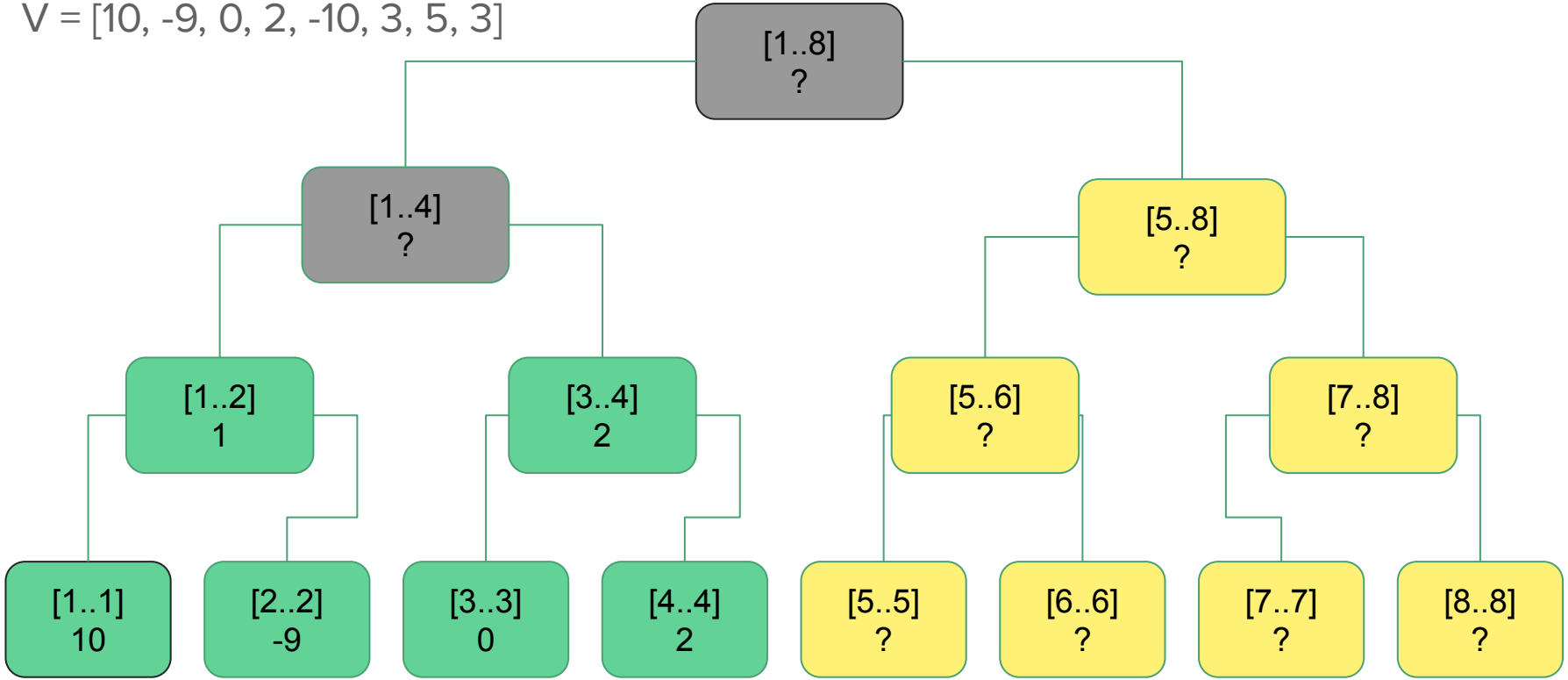
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



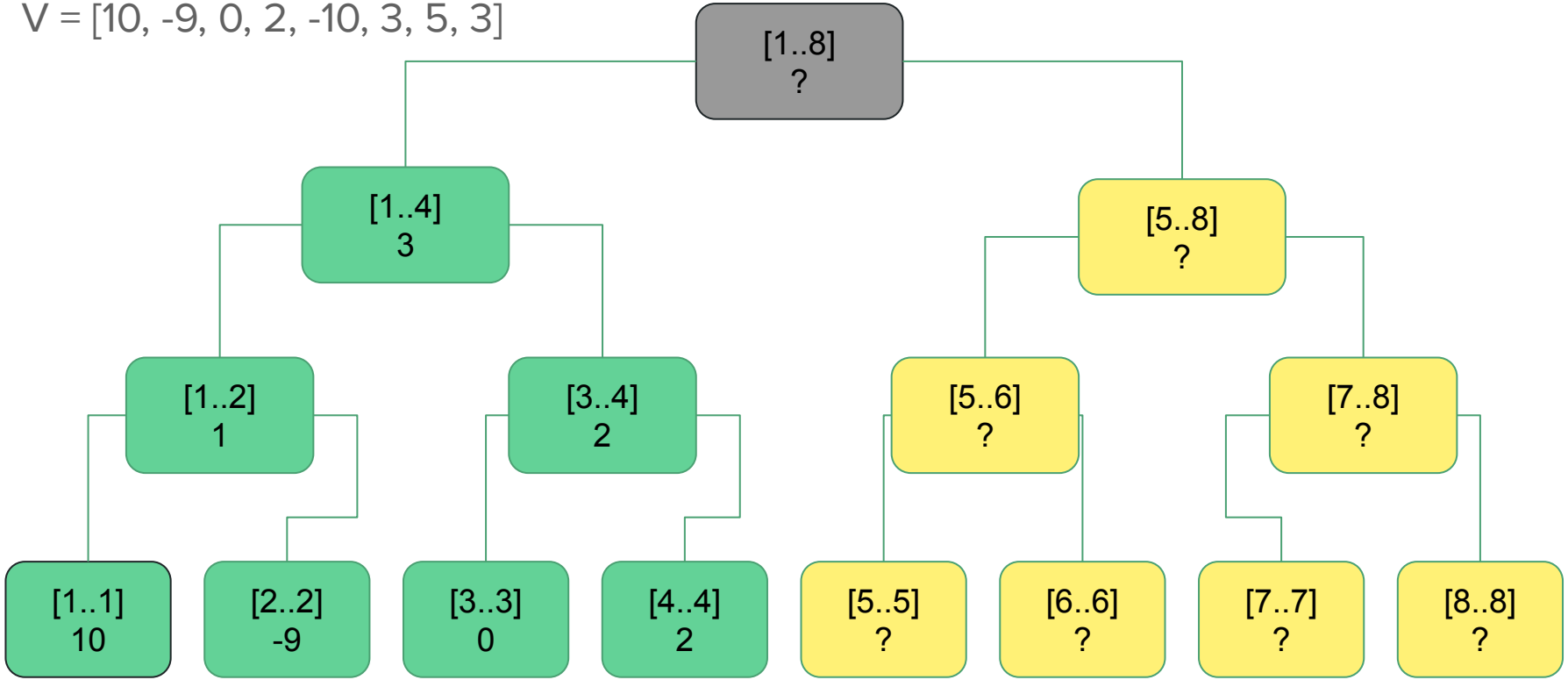
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



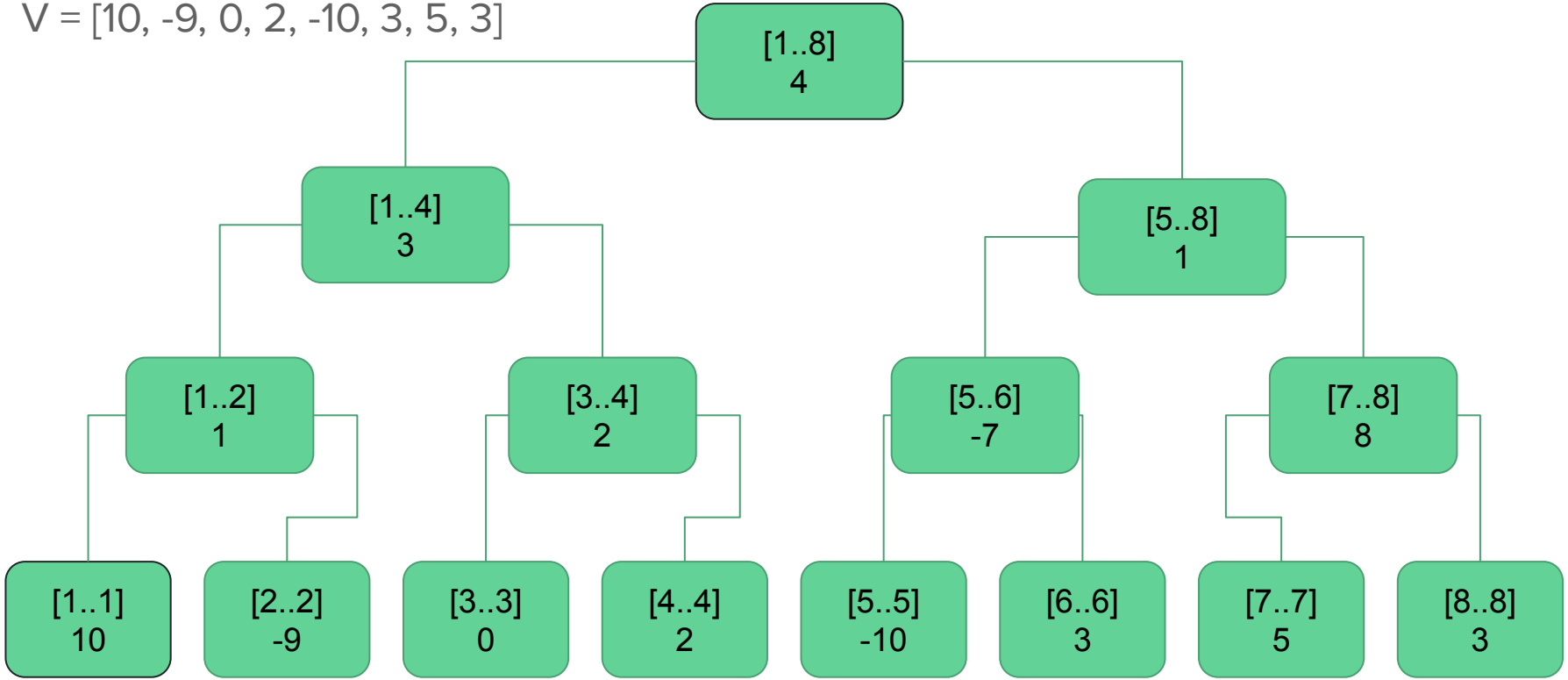
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



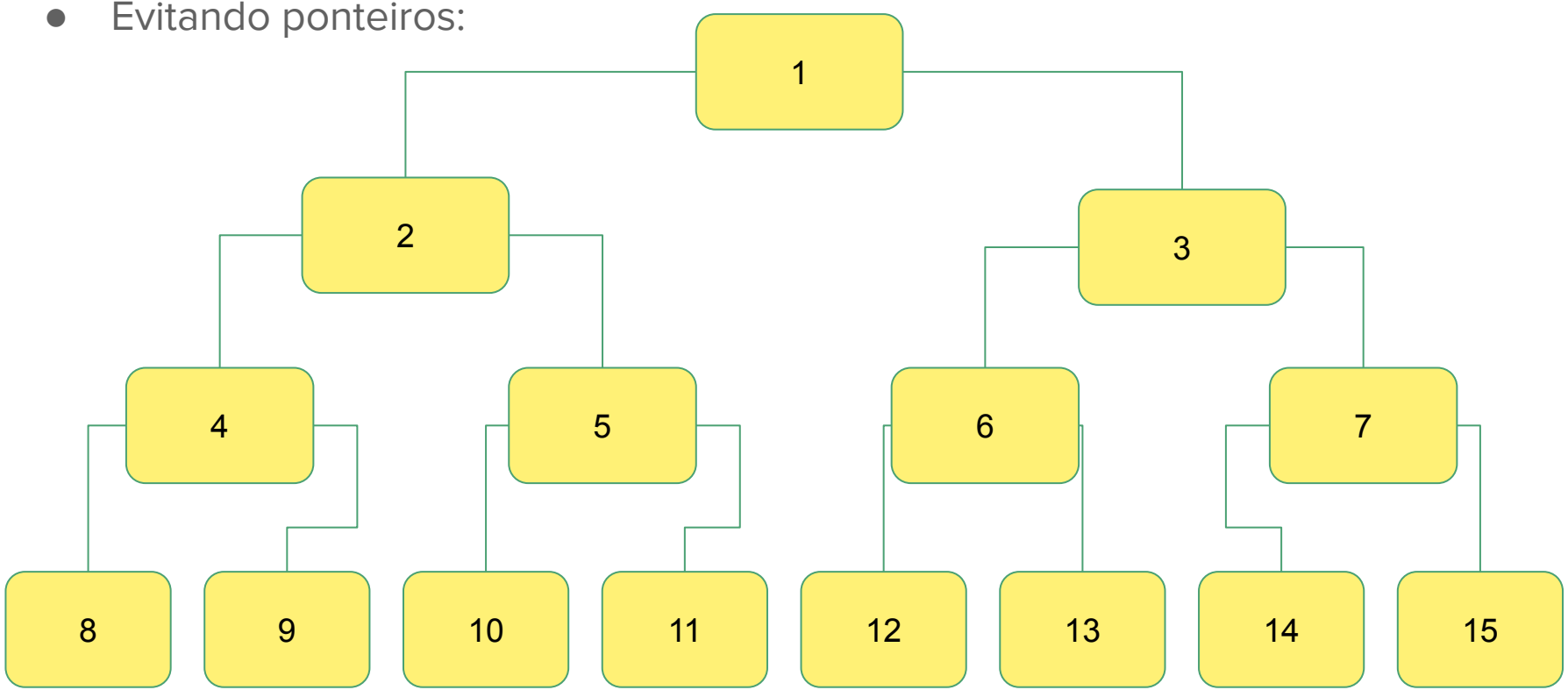
Árvore de Segmento - SegTree

V = [10, -9, 0, 2, -10, 3, 5, 3]



Árvore de Segmento - SegTree (build)

- Evitando ponteiros:



Árvore de Segmento - SegTree (build)

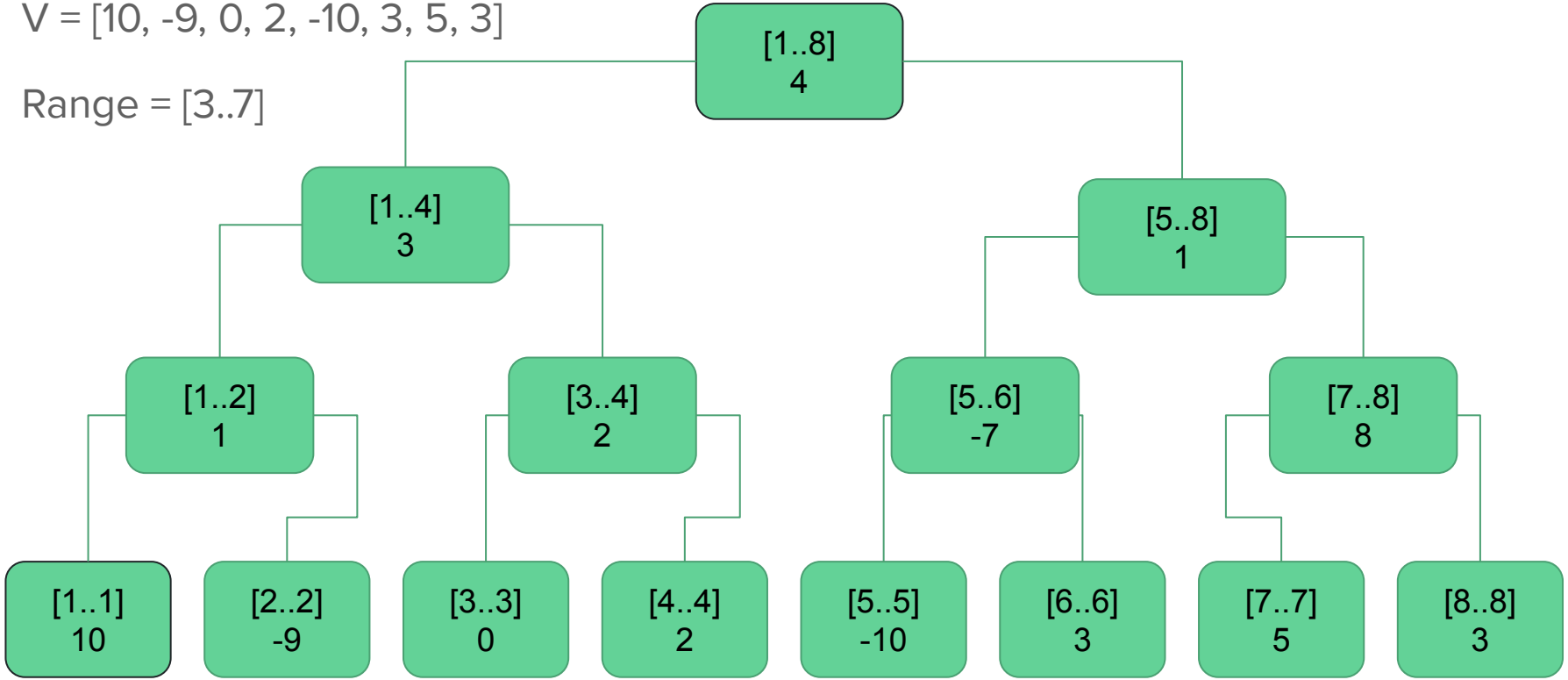
- Build da Segtree:
  - A raiz é o nó 1.
  - O filho esquerdo é 2 * (valor do no)
  - O filho direito é 2 * (valor do no) + 1
  - O valor do nó pode ir até 4*N

```cpp
using ll = long long;
const int MAXN = 100010;
ll st[MAXN*4 + 10];
int v[MAXN];

ll join(ll a, ll b){
    return a + b;
}

void build(int node, int i, int j){
    if (i == j){
        st[node] = v[i];
        return;
    }
    int m = (i + j) / 2;
    int l = node * 2;
    int r = l + 1;
    build(l, i, m);
    build(r, m + 1, j);
    st[node] = join(st[l], st[r]);
}

// Chamar build(1, 1, n)
```

Árvore de Segmento - SegTree (query)

V = [10, -9, 0, 2, -10, 3, 5, 3]

Range = [3..7]



Árvore de Segmento - SegTree (query)

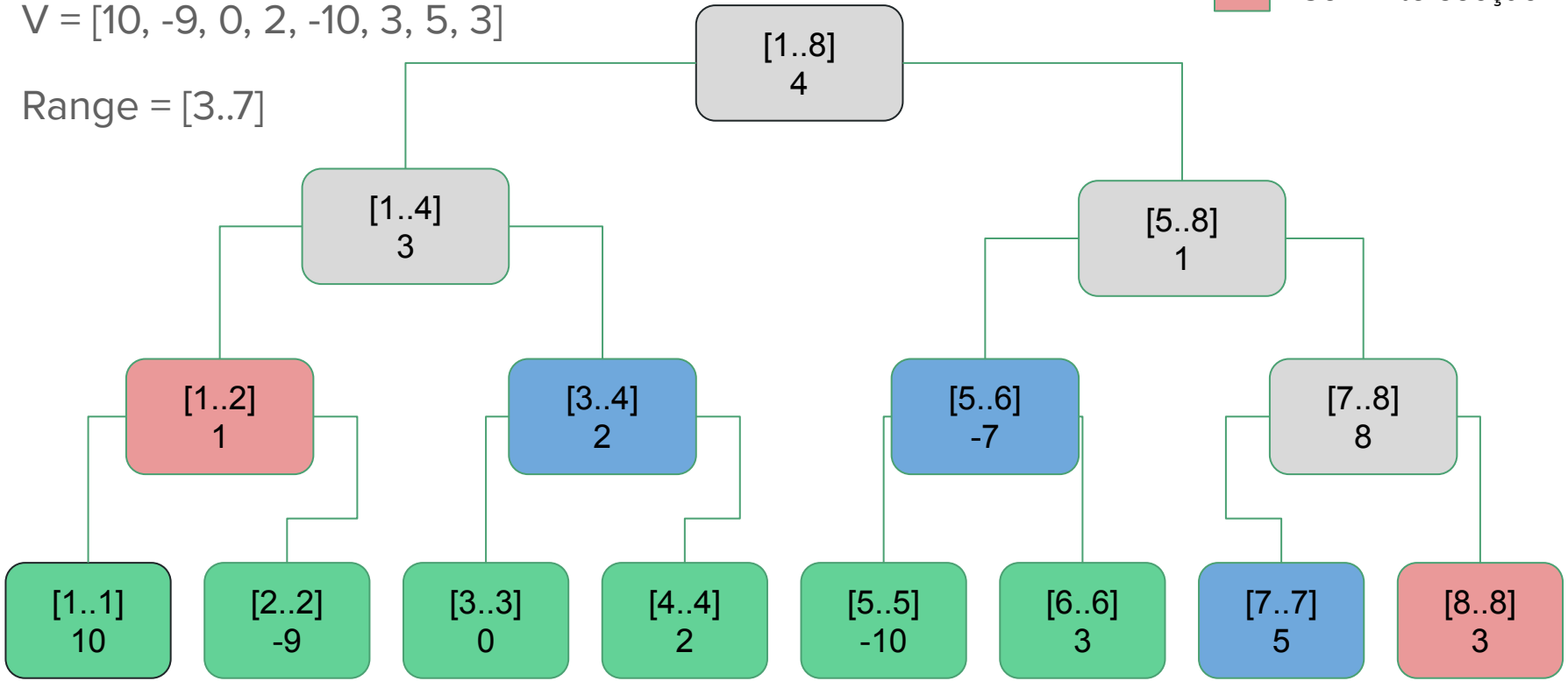
Está contido

Tem intersecção

Sem intersecção

V = [10, -9, 0, 2, -10, 3, 5, 3]

Range = [3..7]



Árvore de Segmento - SegTree (query)

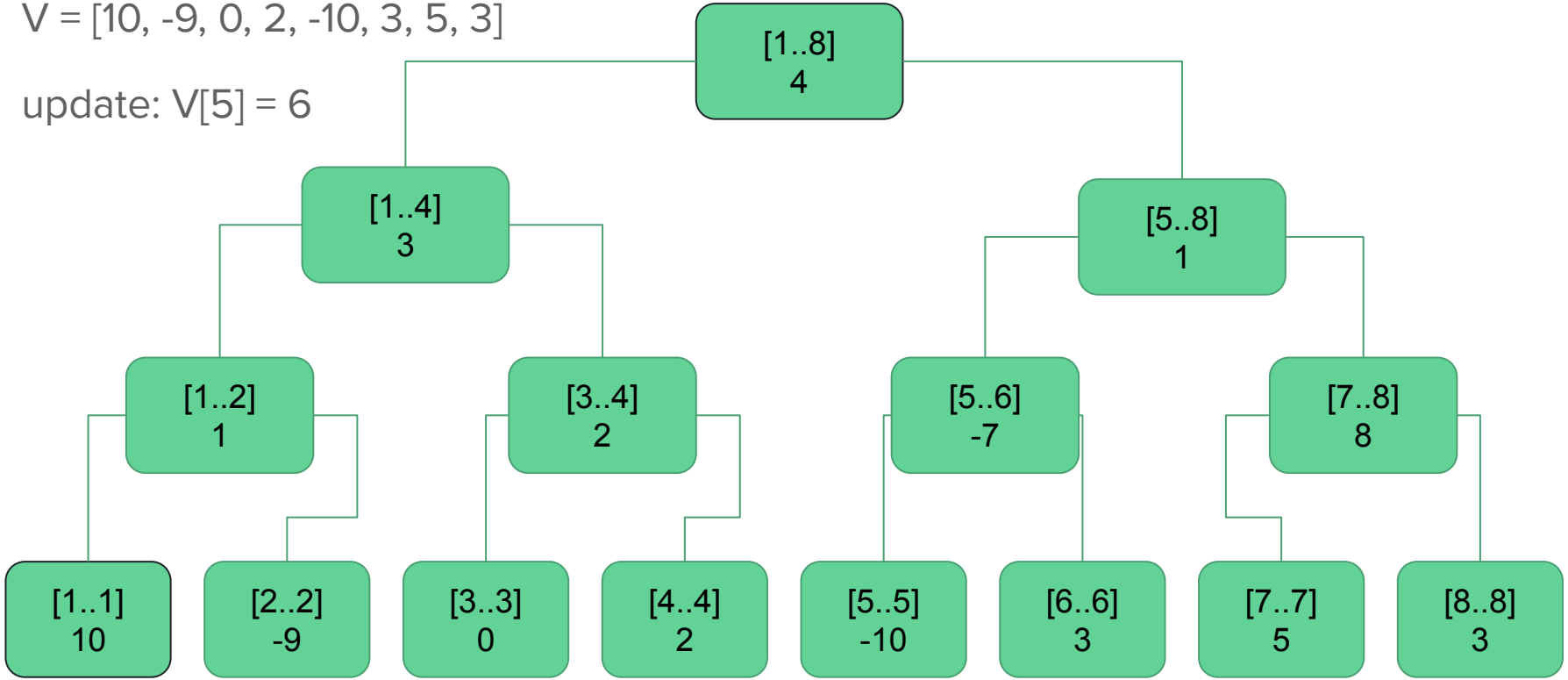
- Query em range: soma do intervalo de [a..b]

```
23  ll neutral = 0;
24
25  ll join(ll a, ll b){
26      return a + b;
27  }
28
29  ll query(int node, int i, int j, int a, int b){
30      if ((i > b) or (j < a))
31          return neutral;
32      if ((a <= i) and (j <= b))
33          return st[node];
34      int m = (i + j) / 2;
35      int l = node * 2;
36      int r = l + 1;
37      return join(query(l, i, m, a, b), query(r, m + 1, j, a, b));
38  }
39  // Chamar query(1, 1, n, a, b)
```

Árvore de Segmento - SegTree (update)

V = [10, -9, 0, 2, -10, 3, 5, 3]

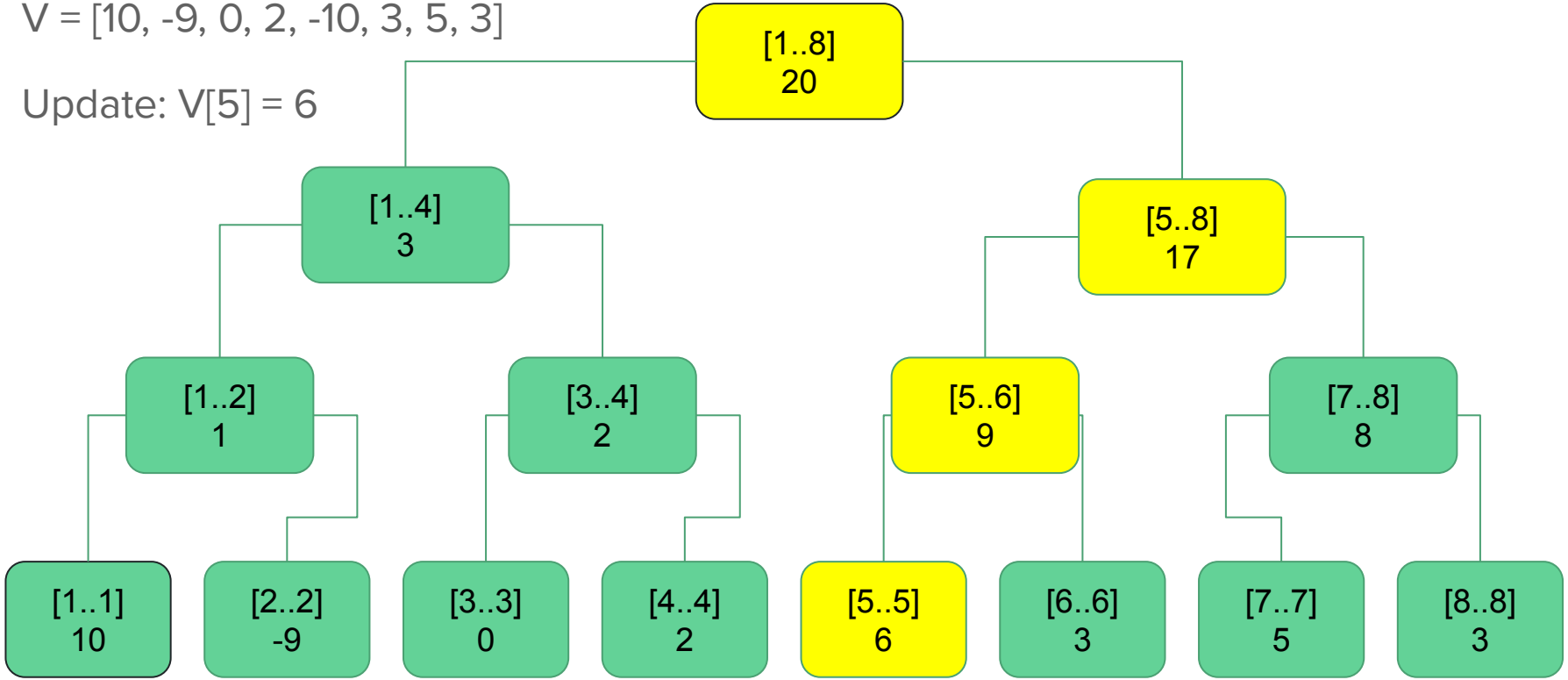
update: V[5] = 6



Árvore de Segmento - SegTree (update)

V = [10, -9, 0, 2, -10, 3, 5, 3]

Update: V[5] = 6



Árvore de Segmento - SegTree (query)

- Update: trocar o valor da posição idx por value. Ou seja, v[idx] = value

```
42  ll join(ll a, ll b){
43      return a + b;
44  }
45
46  void update(int node, int i, int j, int idx, int value){
47      if (i == j){
48          st[node] = value;
49          return;
50      }
51      int m = (i + j) / 2;
52      int l = node * 2;
53      int r = l + 1;
54      if (idx <= m)
55          update(l, i, m, idx, value);
56      else
57          update(r, m + 1, j, idx, value);
58      st[node] = join(st[l], st[r]);
59  }
60  // Chamar update(1, 1, n, idx, value)
```

Árvore de Segmento - SegTree

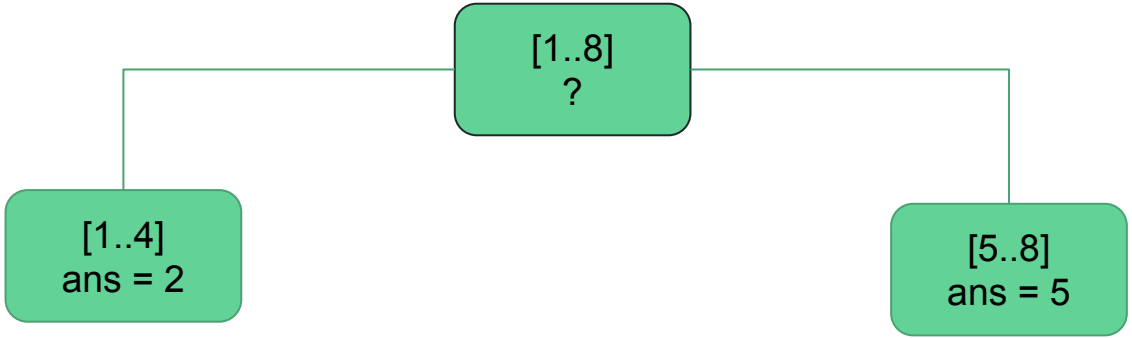
- Trocando para mínimo:
  - Neutral = INF
  - join = min(a, b)
- Trocando para máximo
  - Neutral = -INF
  - join = max(a, b)
- Trocando para GCD
  - Neutral = 0
  - join = gcd(a, b)
- Trocando para qualquer coisa
  - “Se eu tenho a resposta para dois segmentos, eu consigo juntar a resposta deles para formar a resposta da união de forma eficiente?”

Árvore de Segmento - SegTree (Kadane)

- Dado um vetor V, precisamos encontrar o subvetor contínuo que possui a maior soma.
  - O problema para o vetor todo possui um algoritmo O(N)
  - Mas se a gente quiser responder Q queries:
    - Dado o intervalo de [a..b] qual o subarray que esteja contido nesse intervalo e que possui a maior soma?

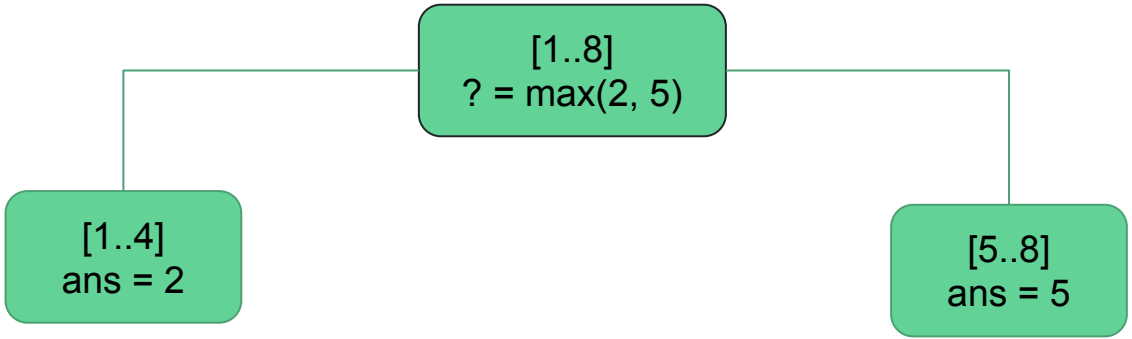
Árvore de Segmento - SegTree (Kadane)

- Para fazer com segtree precisamos saber juntar duas respostas de forma fácil.



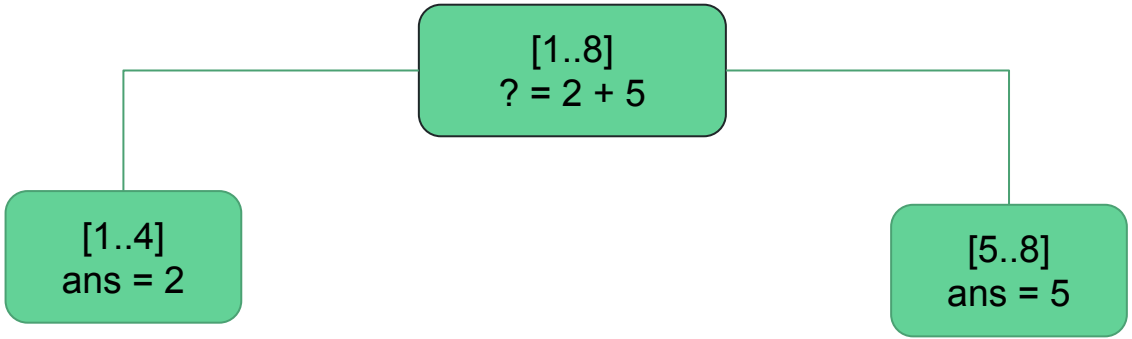
Árvore de Segmento - SegTree (Kadane)

- Para fazer com segtree precisamos saber juntar duas respostas de forma fácil.



Árvore de Segmento - SegTree (Kadane)

- Para fazer com segtree precisamos saber juntar duas respostas de forma fácil.



Árvore de Segmento - SegTree (Kadane)

- Para fazer com segtree precisamos saber juntar duas respostas de forma fácil.
  - Nem sempre é possível juntar diretamente, às vezes precisamos salvar mais informações.
  - Para esse problema, vamos precisar salvar:
    - Soma (sum)
    - Melhor prefixo (pref)
    - Melhor sufixo (suff)
    - Resposta para o intervalo (ans)

Árvore de Segmento - SegTree (Kadane)

```cpp
struct Node {
    ll sum, pref, suff, ans;
};

Node make_node(int val) {
    Node res;
    res.sum = val;
    res.pref = res.suff = res.ans = max(0, val);
    return res;
}

Node join(Node l, Node r) {
    Node res;
    res.sum = l.sum + r.sum;
    res.pref = max(l.pref, l.sum + r.pref);
    res.suff = max(r.suff, r.sum + l.suff);
    res.ans = max(max(l.ans, r.ans), l.suff + r.pref);
    return res;
}
```

Árvore de Segmento - SegTree

- Como fazer uma busca binária na segtree em O(log(N))
  - Busca binária no Prefixo
  - Busca binária em um intervalo qualquer

Árvore de Segmento - SegTree

- Busca binária no Prefixo
  - Exemplos, dado um vetor V com N elementos não-negativos:
    - Encontrar a primeira ocorrência de um zero.
    - Encontrar o menor k cujo a soma dos elementos do intervalo [1..k] seja maior que x.
    - Encontrar a posição do k-ésimo 0.

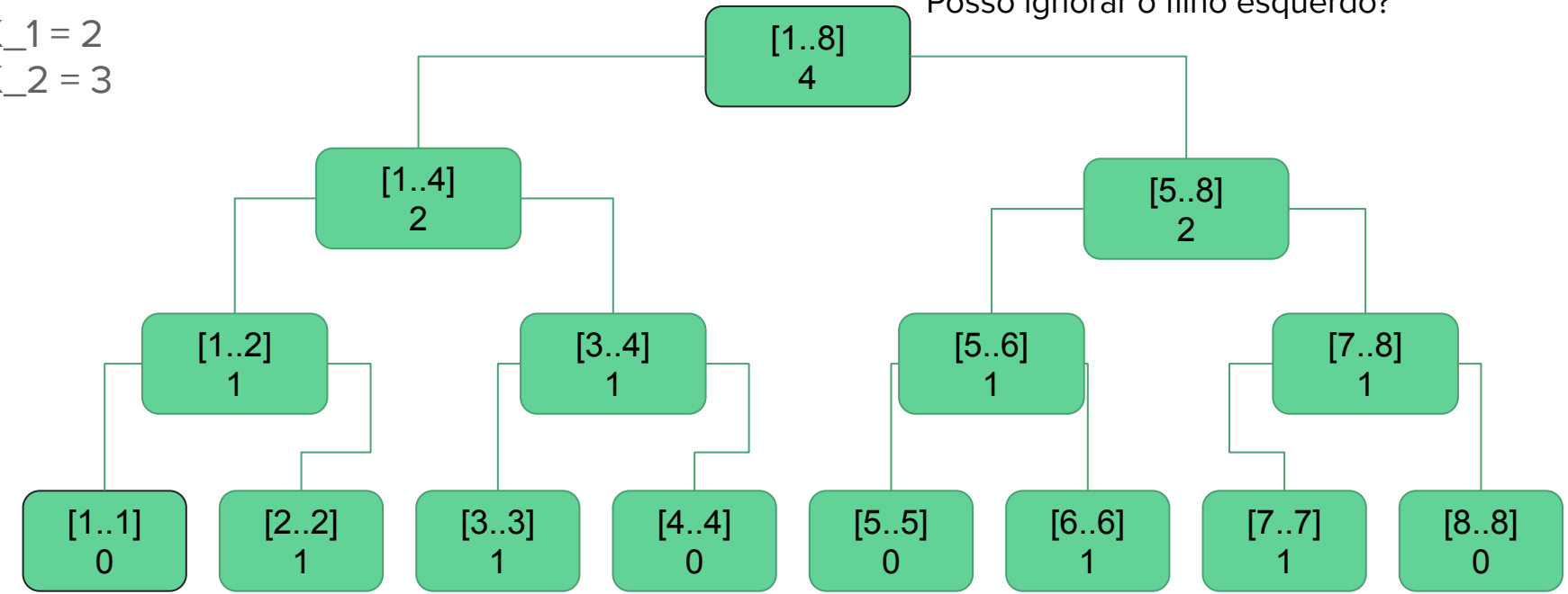
Árvore de Segmento - SegTree

Encontrar a posição do k-ésimo 0

V = [1, 0, 0, 1, 1, 0, 0, 1]

K_1 = 2
K_2 = 3

Posso ignorar o filho esquerdo?



Árvore de Segmento - SegTree

Encontrar a posição do k-ésimo 0

```
1  int find_kth(int node, int i, int j, int k) {
2    if (k > st[node])
3      return -1;
4    if (i == j)
5      return i;
6
7    int m = (i + j) / 2;
8    int l = node * 2;
9    int r = l + 1;
10
11   if (st[l] >= k)
12     return find_kth(l, i, m, k);
13   else
14     return find_kth(r, m+1, j, k - st[l]);
15 }
```

Árvore de Segmento - SegTree

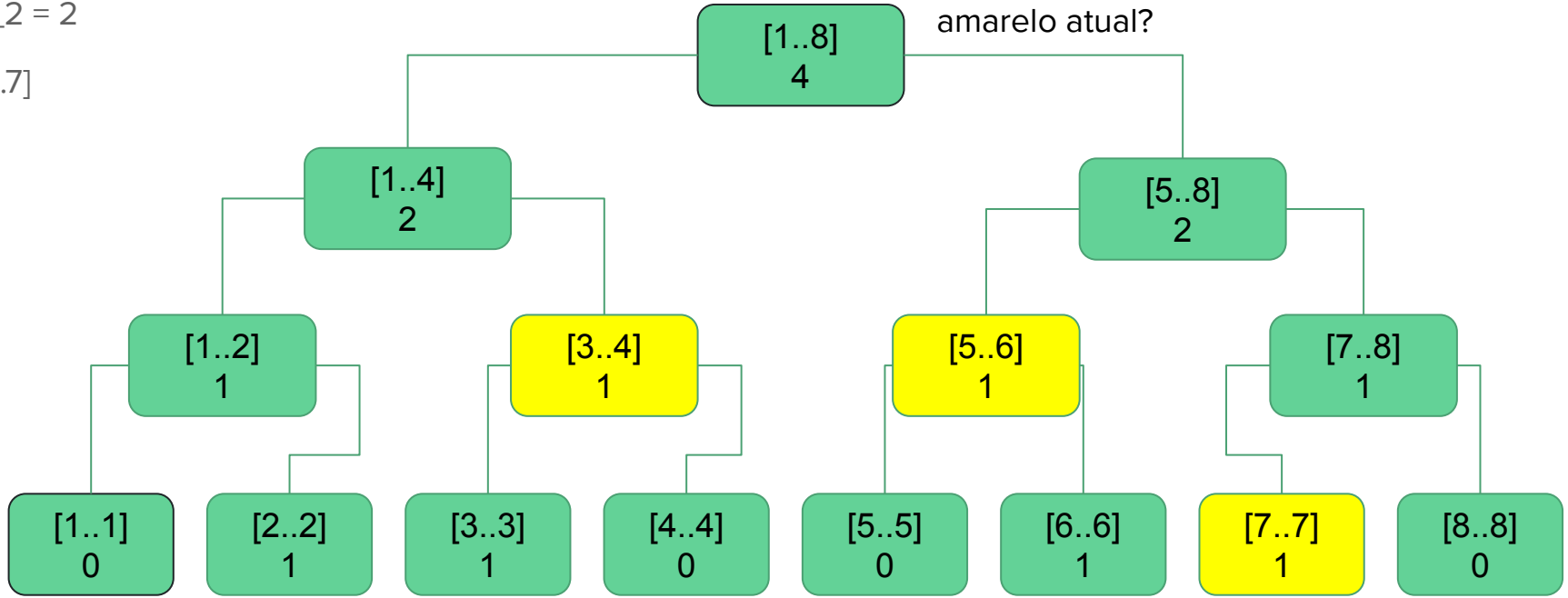
Encontrar a posição do k-ésimo 0 em um intervalo [a..b]

V = [1, 0, 0, 1, 1, 0, 0, 1]

K_2 = 2

[3..7]

Posso ignorar o segmento amarelo atual?



Árvore de Segmento - SegTree

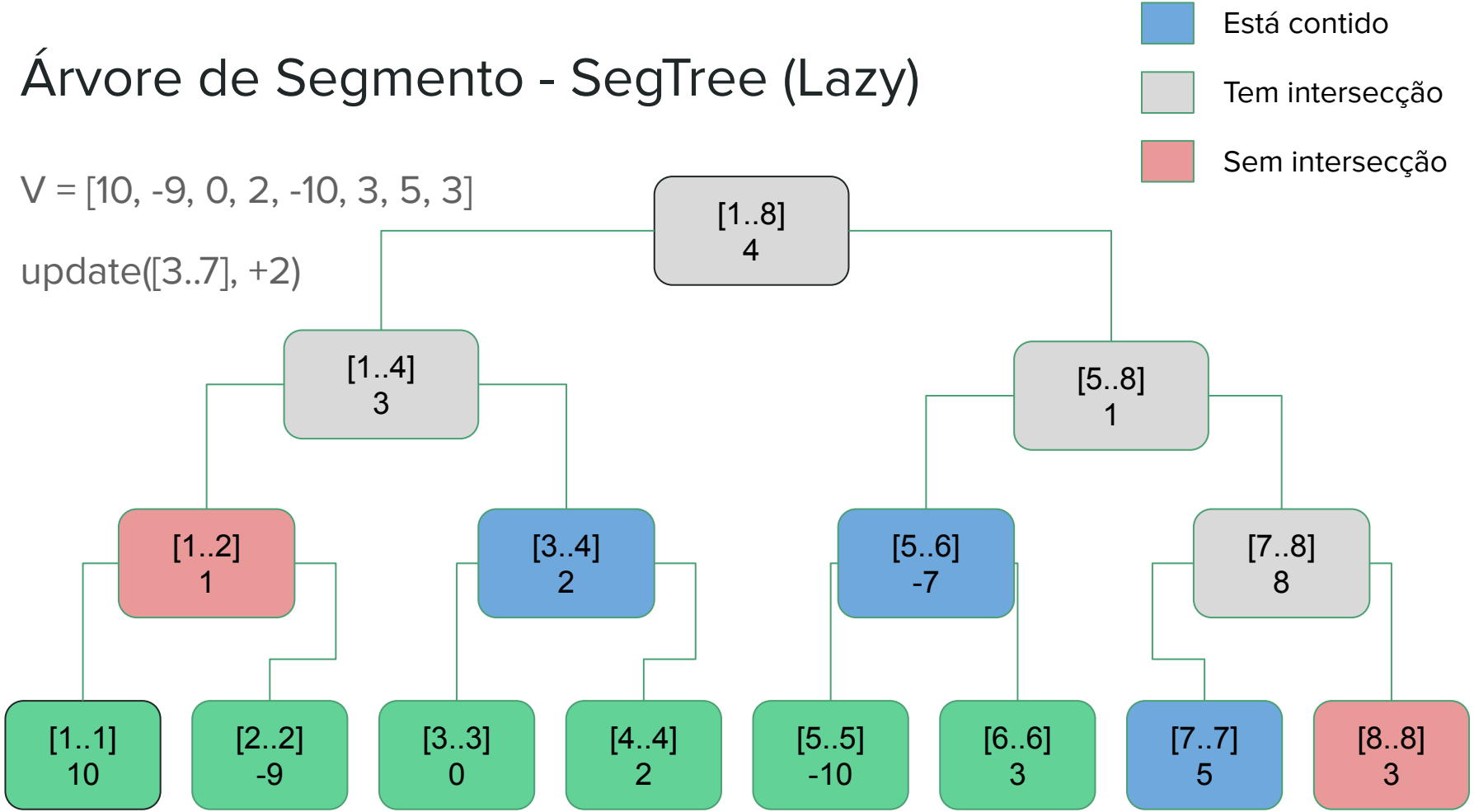
Encontrar a posição do k-ésimo 0 em um intervalo [a..b]

```
1  int find_kth_in_range(int node, int i, int j, int a, int b, int &k) {
2    if ((i > b) or (j < a))
3      return -1;
4    if ((a <= i) and (j <= b)) {
5      if(st[node] < k){
6        k -= st[node];
7        return -1;
8      }
9      return find_kth(node, i, j, k);
10   }
11
12   int m = (i + j) / 2;
13   int l = node * 2;
14   int r = l + 1;
15
16   int rs = find_kth_in_range(l, i, m, a, b, k);
17   if(rs != -1) return rs;
18   return find_kth_in_range(r, m+1, j, a, b, k);
19 }
```

Árvore de Segmento - SegTree

- E se for necessário fazer query e update em range?
  - Podemos usar a técnica de propagação preguiçosa.
  - Vamos ver o problema da soma com atualização em range.

Árvore de Segmento - SegTree (Lazy)



Árvore de Segmento - SegTree (Lazy)

- Propagação e armazenamento da operação lazy

```
1   using ll = long long;
2   const int MAXN = 100010;
3   ll st[MAXN*4 + 10];
4   ll lazy[MAXN*4 + 10];
5
6   ll join(ll a, ll b){
7     return a + b;
8   }
9
10  inline void upLazy(int &node, int &i, int &j){
11    if (lazy[node] != 0){
12      st[node] += lazy[node] * (j - i + 1);
13      //st[node] += lazy[node];
14      if (i != j){
15        lazy[(node << 1)] += lazy[node];
16        lazy[(node << 1) + 1] += lazy[node];
17      }
18      lazy[node] = 0;
19    }
20  }
```

Árvore de Segmento - SegTree (Lazy)

- Query

```
1  ll neutral = 0;
2  ll query(int node, int i, int j, int a, int b){
3      upLazy(node, i, j);
4      if ((i > b) or (j < a))
5          return neutral;
6      if ((a <= i) and (j <= b)){
7          return st[node];
8      }
9      int m = (i + j) / 2;
10     int l = (node << 1);
11     int r = l + 1;
12     return join(query(l, i, m, a, b), query(r, m + 1, j, a, b));
13 }
```

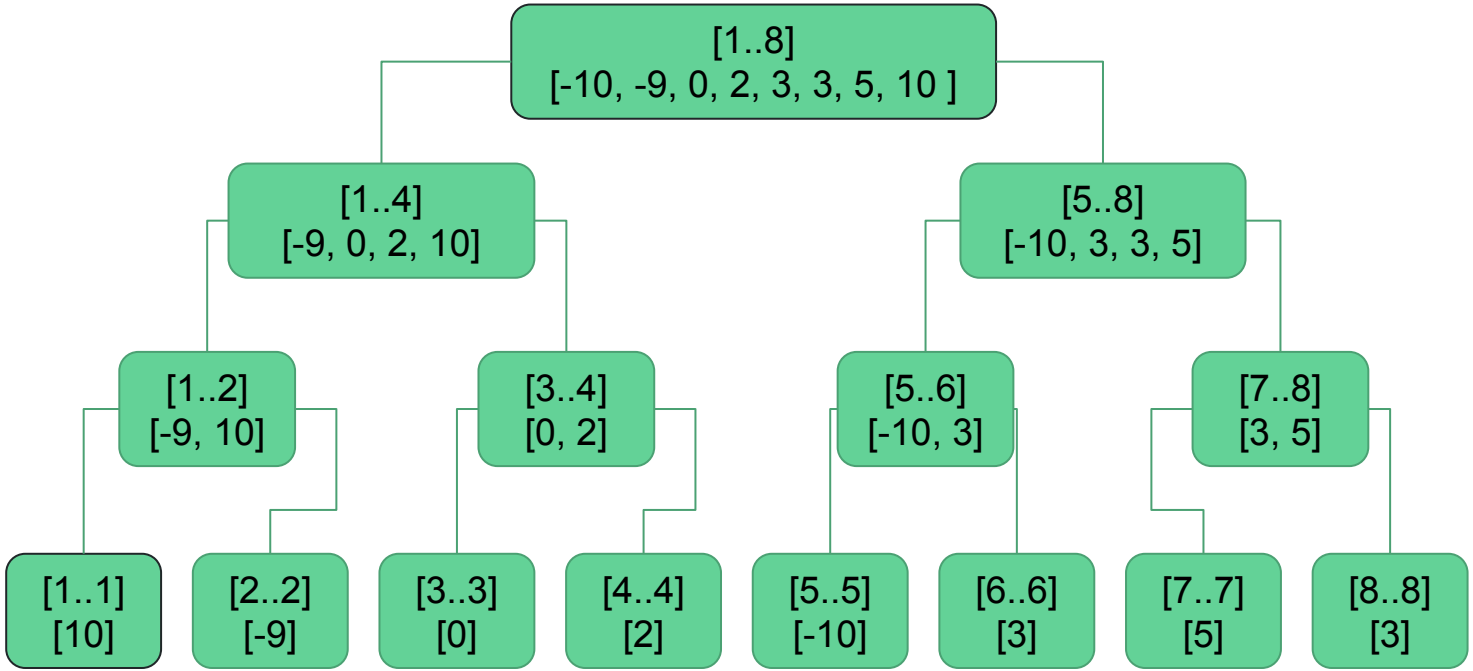
Árvore de Segmento - SegTree (Lazy)

- Update

```
1  void update(int node, int i, int j, int a, int b, int value){
2    upLazy(node, i, j);
3    if ((i > j) or (i > b) or (j < a))
4      return;
5    if ((a <= i) and (j <= b)){
6      lazy[node] = value;
7      upLazy(node, i, j);
8    }else{
9      int m = (i + j) / 2;
10     int l = (node << 1);
11     int r = l + 1;
12     update(l, i, m, a, b, value);
13     update(r, m + 1, j, a, b, value);
14     st[node] = join(st[l], st[r]);
15   }
16 }
```

Árvore de Segmento - SegTree

- Podemos salvar um vetor em cada nó, e fazer uma Merge Sort Tree.

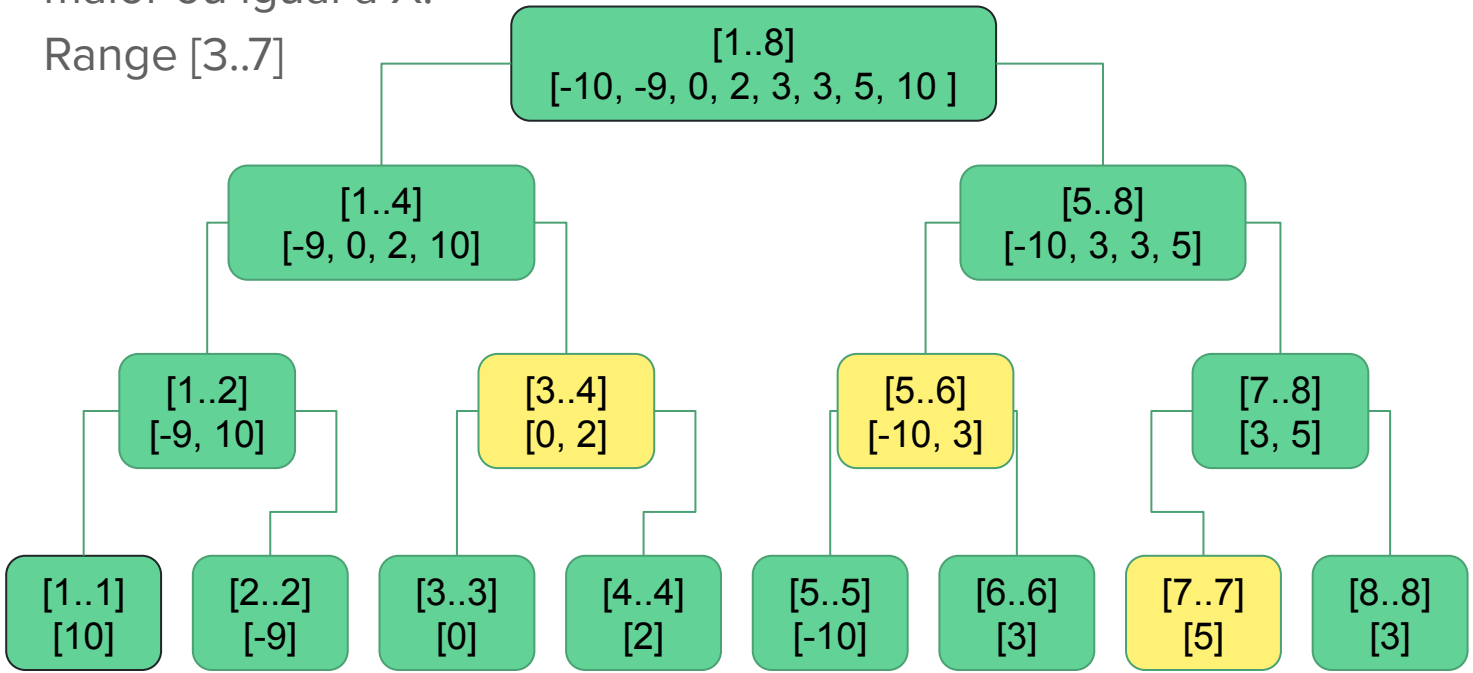


Árvore de Segmento - SegTree

- Podemos salvar um vetor em cada nó, e fazer uma Merge Sort Tree.
- Que tipo de query dá para responder?
  - Dado (L, R, X) encontrar o menor elemento no range [L..R] que seja maior ou igual a X. (Sem update)

Árvore de Segmento - SegTree

- Dado (L, R, X) encontrar o menor elemento no range [L..R] que seja maior ou igual a X.
- Range [3..7]



Árvore de Segmento - SegTree

- Podemos salvar um vetor em cada nó, e fazer uma Merge Sort Tree.
- Que tipo de query dá para responder?
  - Dado (L, R, X) encontrar o menor elemento no range [L..R] que seja maior ou igual a X. (Sem update)
  - E se tivesse update?

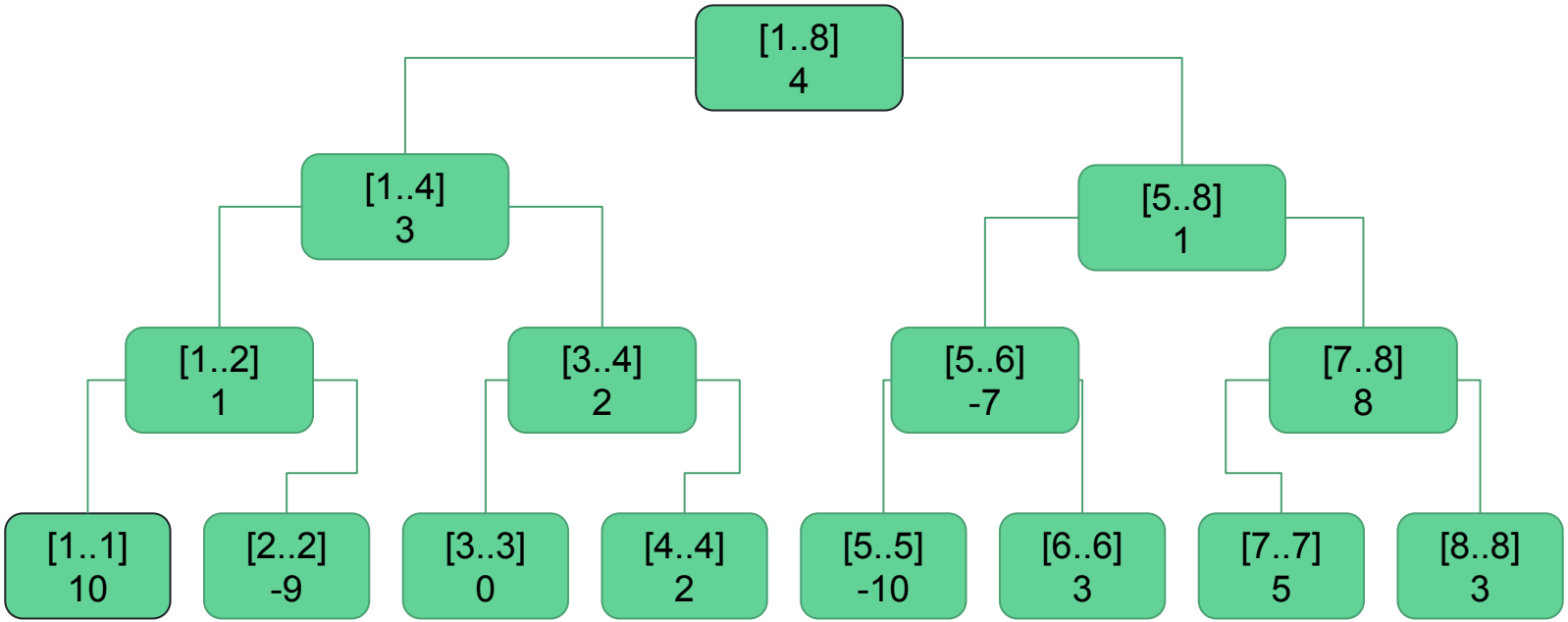
Árvore de Segmento - SegTree

- Podemos salvar um vetor em cada nó, e fazer uma Merge Sort Tree.
- Que tipo de query dá para responder?
  - Dado (L, R, X) encontrar o menor elemento no range [L..R] que seja maior ou igual a X. (Sem update)
  - E se tivesse update?
    - Podemos usar uma árvore balanceada que permita lower_bound.
      - Set do C++

Árvore de Segmento - SegTree (Persistente)

V = [10, -9, 0, 2, -10, 3, 5, 3]

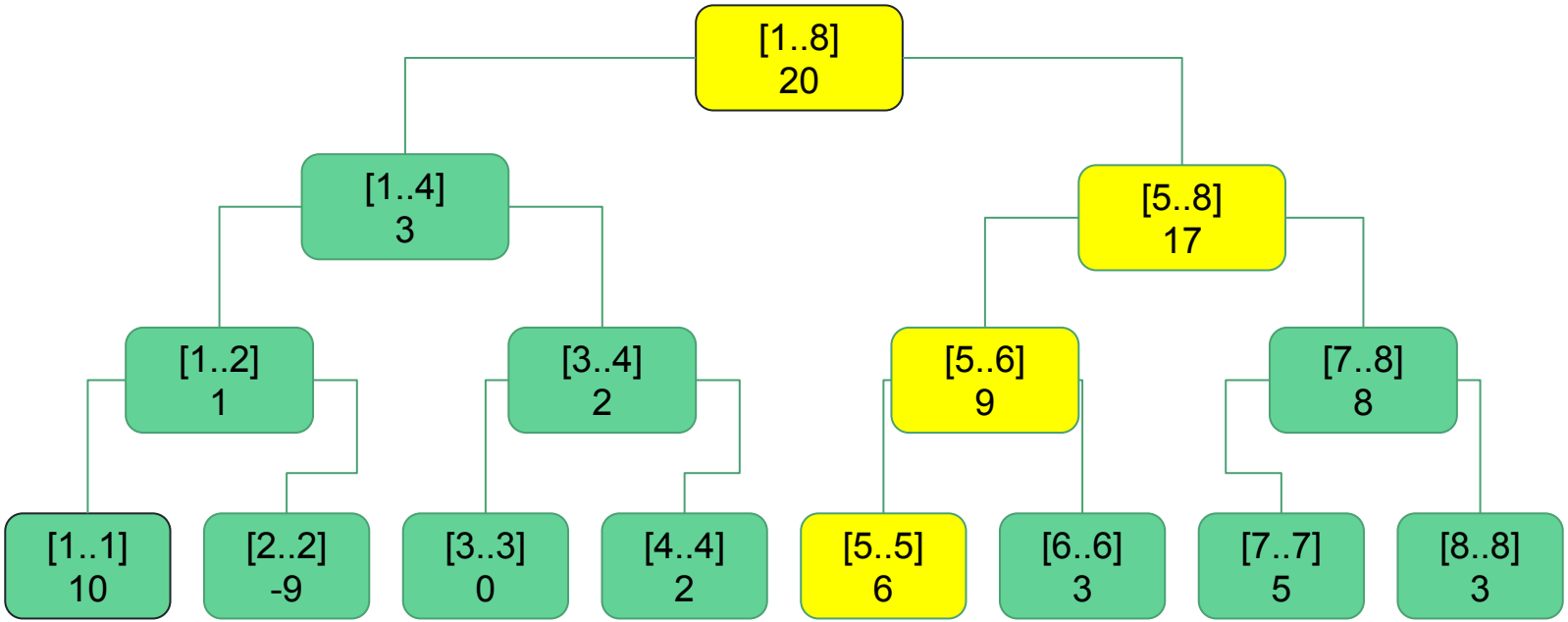
Update: V[5] = 6



Árvore de Segmento - SegTree (Persistente)

V = [10, -9, 0, 2, -10, 3, 5, 3]

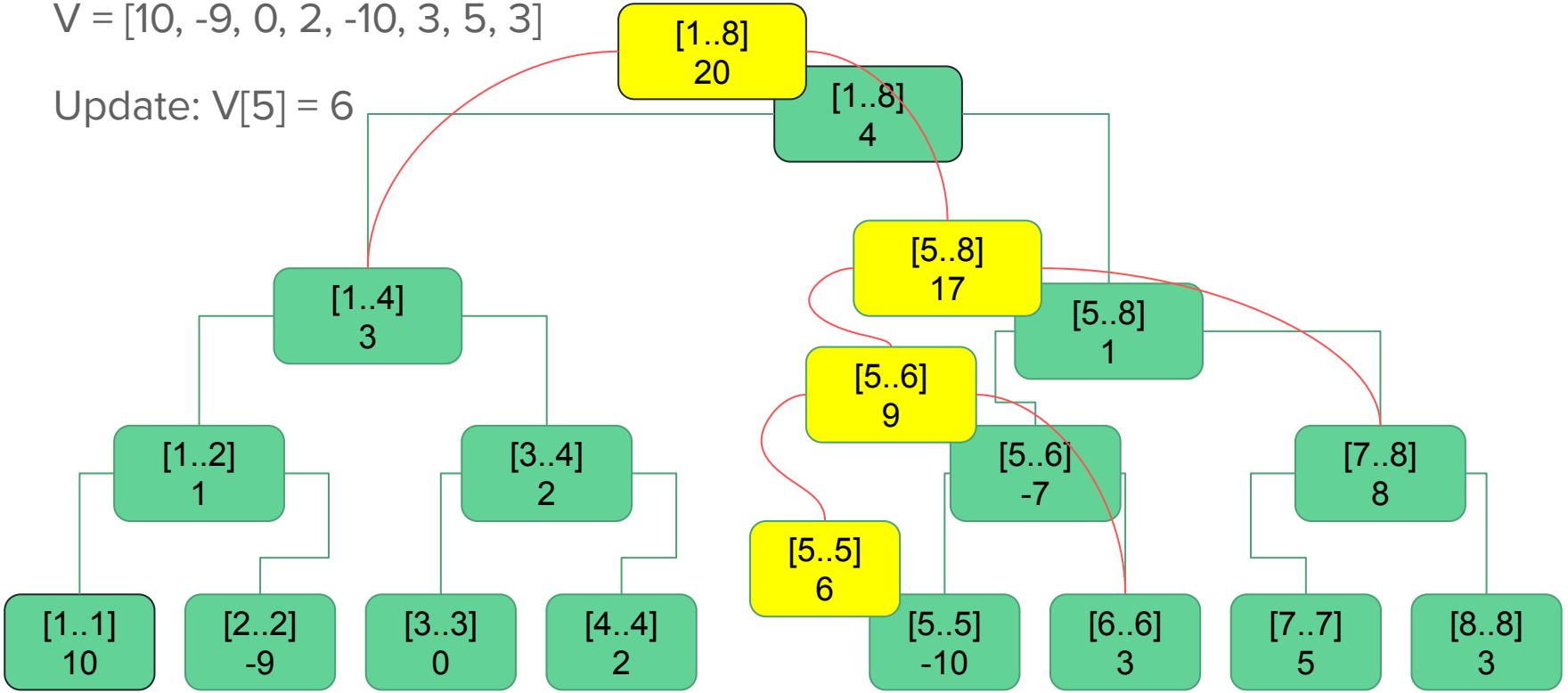
Update: V[5] = 6



Árvore de Segmento - SegTree (Persistente)

V = [10, -9, 0, 2, -10, 3, 5, 3]

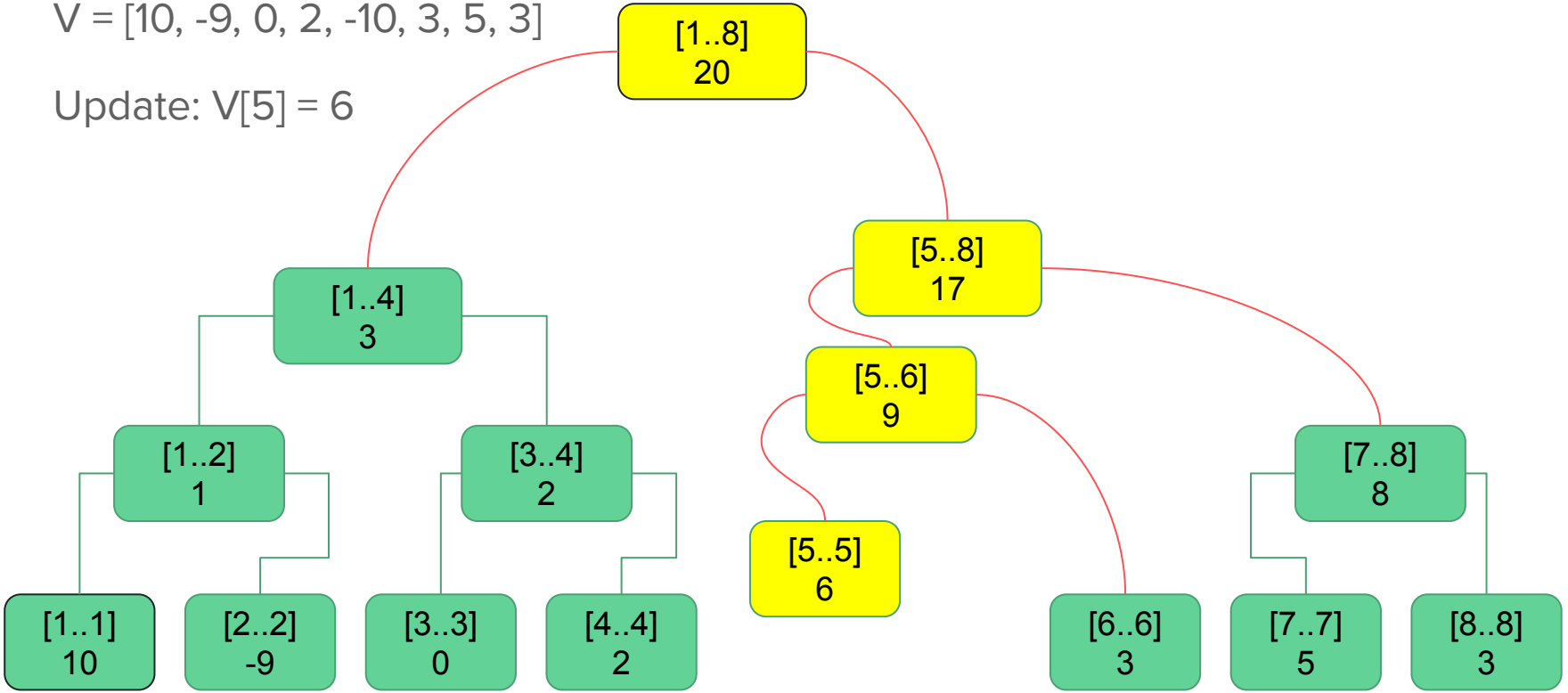
Update: V[5] = 6



Árvore de Segmento - SegTree (Persistente)

V = [10, -9, 0, 2, -10, 3, 5, 3]

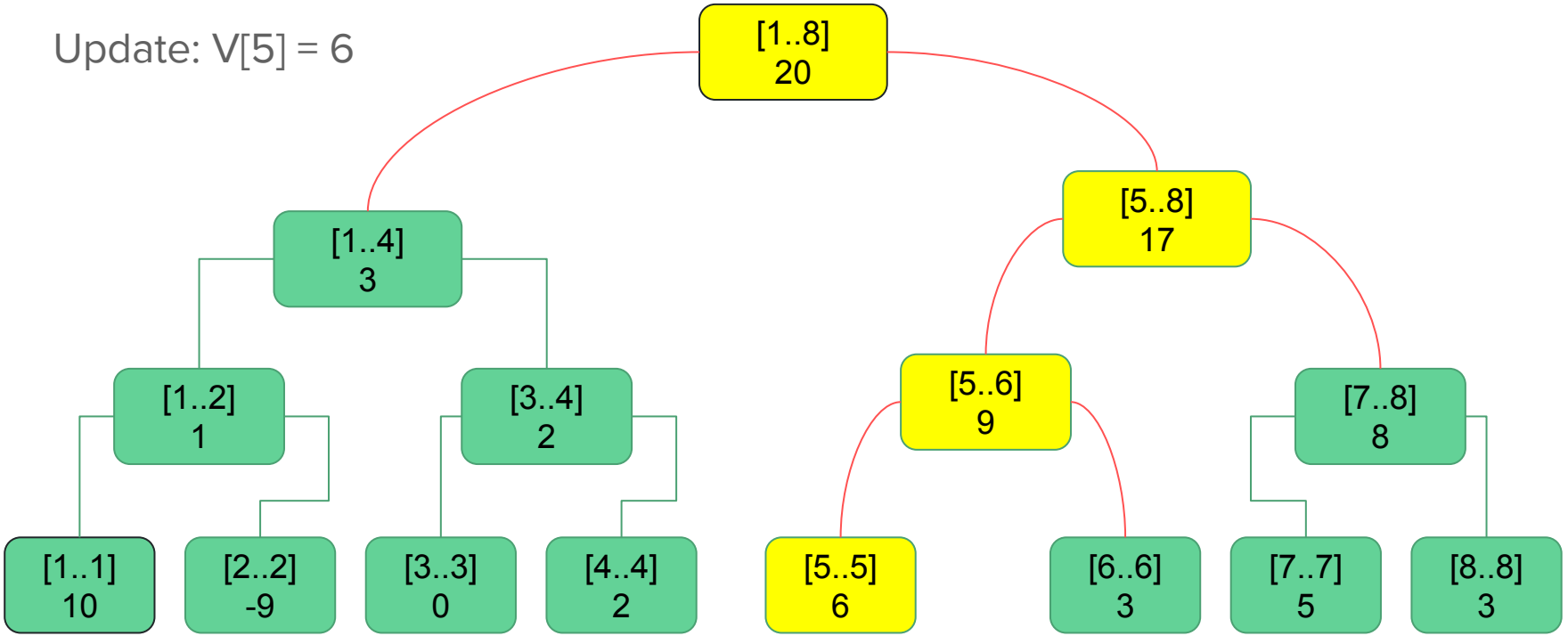
Update: V[5] = 6



Árvore de Segmento - SegTree (Persistente)

V = [10, -9, 0, 2, -10, 3, 5, 3]

Update: V[5] = 6



Árvore de Segmento - SegTree (Persistente)

- Alguns detalhes:
  - Então podemos manter todas as versões de uma árvore, só precisamos guardar um ponteiro para a raiz.
  - A complexidade ainda continua O(log(N)) para cada operação
  - Cada update só gera O(log(N)) nós novos e aproveita os outro que já existem.
  - Nunca mudamos o valor de um nó, apenas criamos outro quando necessário.
  - Agora temos o poder de voltar no tempo
- Como usar isso?
  - Aplicações mais diretas
  - Qual o k-ésimo elemento no range [a, b]?
  - Quantos valores distintos existem no range [a..b]?

Árvore de Segmento - SegTree

- E se o vetor tiver tamanho na ordem de 10^9:
  - Exemplo:
    - Inicialmente todas as 10^9 posições são 0.
    - query = calcular a soma em um intervalo [a, b]
    - update = somar x no elemento da posição k
  - Se for possível fazer offline, você pode usar a compressão de coordenadas.
  - Senão, você pode implementar uma segtree usando ponteiros.
    - Durante o build, em intervalos onde só tem zero, você não precisa descer na recursão. Faz algo parecido com o lazy.
    - No updade e na query, vai gerando os filhos de acordo com a necessidade.
    - Com isso, a quantidade de nós gerados é na ordem de $O(Q * \log(10^9))$, onde Q é a quantidade de queries.

Árvore de Segmento - SegTree (esparsa)

```
struct Vertex {
    int left, right;
    int sum = 0;
    Vertex *left_child = nullptr, *right_child = nullptr;

    Vertex(int lb, int rb) {
        left = lb;
        right = rb;
    }

    void extend() {
        if (!left_child && left + 1 < right) {
            int t = (left + right) / 2;
            left_child = new Vertex(left, t);
            right_child = new Vertex(t, right);
        }
    }
```

Árvore de Segmento - SegTree (esparsa)

```
    void add(int k, int x) {
        extend();
        sum += x;
        if (left_child) {
            if (k < left_child->right)
                left_child->add(k, x);
            else
                right_child->add(k, x);
        }
    }

    int get_sum(int lq, int rq) {
        if (lq <= left && right <= rq)
            return sum;
        if (max(left, lq) >= min(right, rq))
            return 0;
        extend();
        return left_child->get_sum(lq, rq) + right_child->get_sum(lq, rq);
    }
};
```

Obrigado!

Qualquer dúvida fico a disposição!