Árvore de Segmentos

Definição e Implementação

Prof. Edson Alves - UnB/FGA

2020

Sumário

Definição

Árvore de Segmentos

Definição

Uma **árvore de segmentos** (*segment tree*) é uma estrutura de dados que tem suporte para duas operações sobre um vetor xs de N elementos: realizar uma consulta sobre um subintervalo de índices $[i, j]$ (`range_query(i, j)`) e atualizar o valor de $xs[i]$ (`update(i, value)`), ambas com complexidade $O(\log N)$.

Características da árvore de segmentos

- Uma árvore de segmentos é uma árvore binária completa cujos nós intermediários armazenam os resultados da operação subjacente sobre um subintervalos de índices $[i, j]$, e cujas folhas são os elementos do vetor xs
- Se um nó intermediário armazena o resultado da operação para $[i, j]$, seu filho à esquerda armazena os resultados para $[i, i + m)$, e seu filho à direita armazena os resultados para $[i + m, j]$, onde $m = \lfloor (j - i + 1)/2 \rfloor$
- As árvores de segmentos são estruturas mais flexíveis do que as árvores de Fenwick
- Por outro lado, elas são mais difíceis de implementar e precisam de mais memória

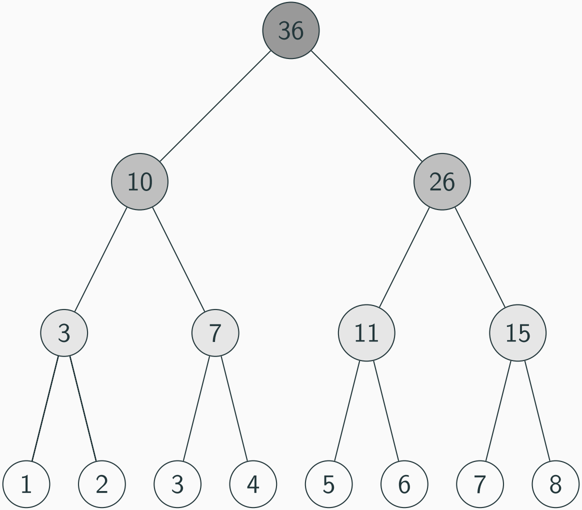
Operação subjacente

- Cada árvore de segmentos tem uma operação binária subjacente $\odot$
- Esta operação tem a propriedade de que, se $[a, c] = [a, b) \cup [b, c]$, $\odot([a, c])$ pode ser computado diretamente a partir dos valores de $\odot([a, b))$ e $\odot([b, c])$, isto é,

$$\odot([a, c]) = f(\odot([a, b)), \odot([b, c]))$$

- As operações subjacentes mais comuns são:
  (a) soma dos elementos do intervalo
  (b) elemento mínimo do intervalo
  (c) elemento máximo do intervalo
  (d) ou exclusivo dos elementos do intervalo
- Outras operações também podem ser implementadas, desde que possuam a propriedade supracitada

Visualização de uma árvore de segmentos para soma dos elementos



Implementação *bottom-up*

Número total de nós

- Suponha que $N = 2^k$, para algum k positivo
- Assim, o nível i da árvore terá 2^i nós que representam um intervalo de tamanho $N/2^i$
- A altura h da árvore é igual a $h = \log N = \log 2^k = k$
- Logo, o total de nós da árvore será igual a

$$\sum_{i=0}^{k} 2^i = 1 + 2 + \ldots + 2^k = 2^{k+1} - 1 < 2^{k+1} = 2N$$

- Assim, a árvore de segmentos deve reservar espaço para $2N$ nós

Construtor

- Em um implementação *bottom-up*, os nós da árvore de segmentos são armazenados em um vetor ns de $2N$ elementos, do mesmo tipo dos elementos de xs
- A posição 0 (zero) não é utilizada, sendo a raiz armazenada no índice 1 (um)
- Seja u o nó que ocupa o índice i de ns
- O filho a esquerda de u ocupará o índice $2i$, e o filho à direita o índice $2i + 1$
- O pai de u ocupará o índice $\lfloor i/2 \rfloor$
- Os elementos de xs ocuparam os índices de N até $2N - 1$
- Das folhas até a raiz, um nível por vez, serão preenchidos os nós internos, usando a operação subjacente

Visualização do construtor da árvore de segmentos

Operação subjacente: soma dos elementos

$xs =$

7	−3	−5	2

$ns =$

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

Visualização do construtor da árvore de segmentos

Copia dos elementos de xs para as folhas

$xs =$

7	-3	-5	2

$ns =$

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
				7	-3	-5	2

Visualização do construtor da árvore de segmentos

Preenchimento do nível intermediário

$xs =$

7	-3	-5	2

$ns =$

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
			-3	7	-3	-5	2

Visualização do construtor da árvore de segmentos

Preenchimento do nível intermediário

$xs =$

7	-3	-5	2

$ns =$

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
		4	-3	7	-3	-5	2

Visualização do construtor da árvore de segmentos

Preenchimento da raiz

$xs =$

7	-3	-5	2

$ns =$

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	1	4	-3	7	-3	-5	2

Implementação do construtor da árvore de segmentos

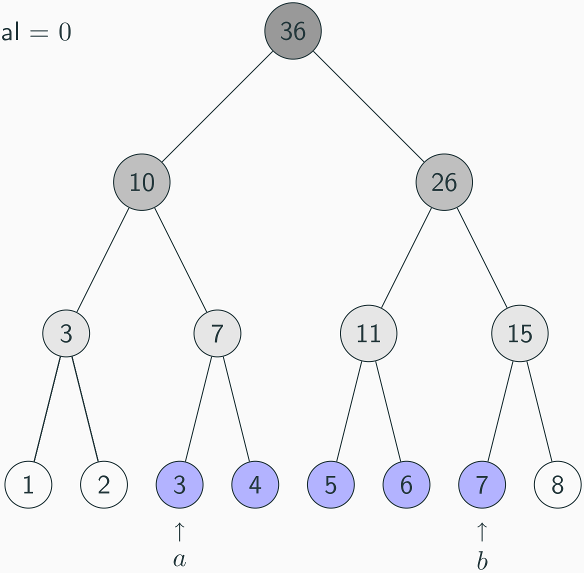
```cpp
#ifndef SEGMENT_TREE_H
#define SEGMENT_TREE_H

#include <vector>
#include <algorithm>

template<typename T>
class SegmentTree
{
    int N;
    std::vector<T> ns;

public:
    SegmentTree(const std::vector<T>& xs) : N(xs.size()), ns(2*N, -1)
    {
        std::copy(xs.begin(), xs.end(), ns.begin() + N);

        for (int i = N - 1; i > 0; --i)
            ns[i] = ns[2*i] + ns[2*i + 1];
    }

```

Range query

- Uma vez inicializada a árvore de segmentos, é possível determinar o resultado da operação subjacente para um intervalo $[a, b]$ arbitrário
- Para isto, três observações devem ser feitas:
    1. Se a é ímpar, ele é o filho à direita, logo ele deve ser processado separadamente
    2. O mesmo acontece se b é par
    3. Nos outros casos, os valores de a e b já foram processados por seus pais, e o processamento deve seguir para estes pais
- Assim, como a altura da árvore é igual a $\log N$, esta rotina tem complexidade $O(\log N)$
- Se a operação subjacente é a soma dos elementos, esta operação recebe o nome de *range sum query* (RSQ)

Visualização de RSQ(2, 6)

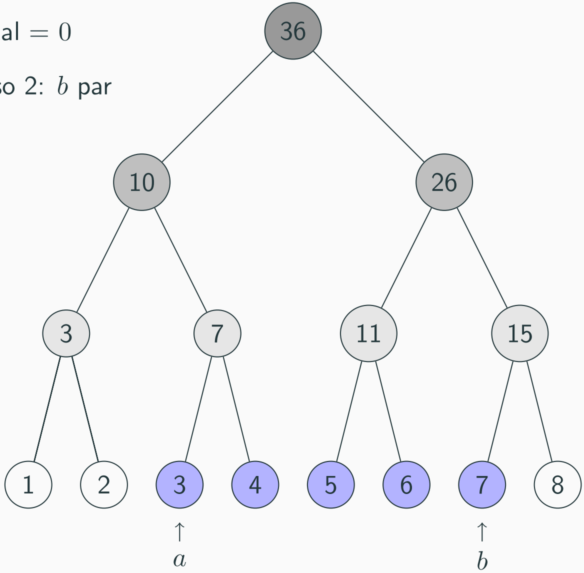
Total $= 0$



Visualização de RSQ(2, 6)

Total $= 0$

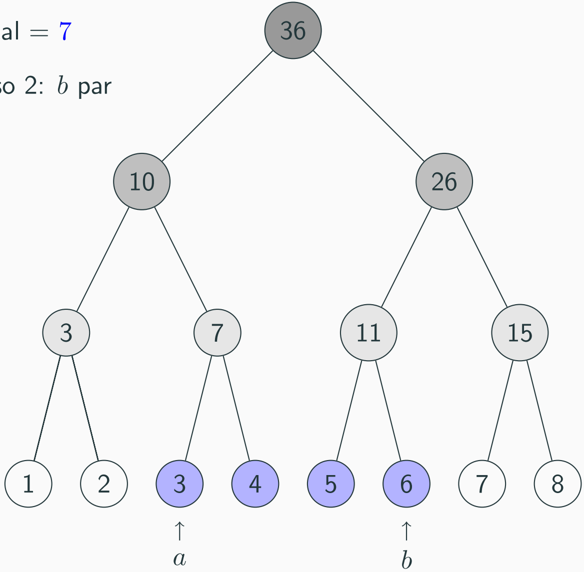
Caso 2: b par



Visualização de RSQ(2, 6)

Total $= \textcolor{blue}{7}$

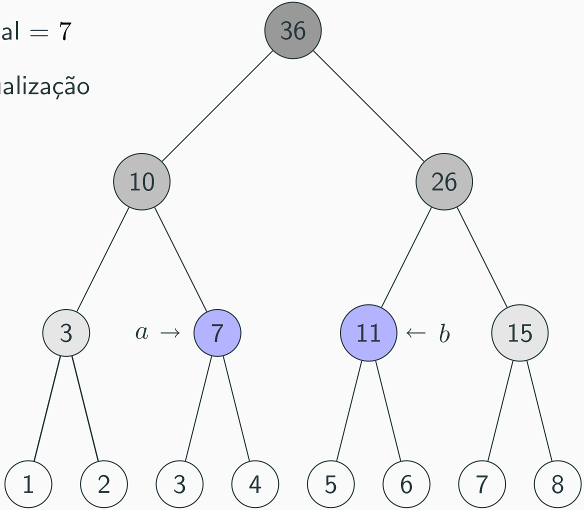
Caso 2: b par



Visualização de RSQ(2, 6)

Total $= \mathbf{7}$

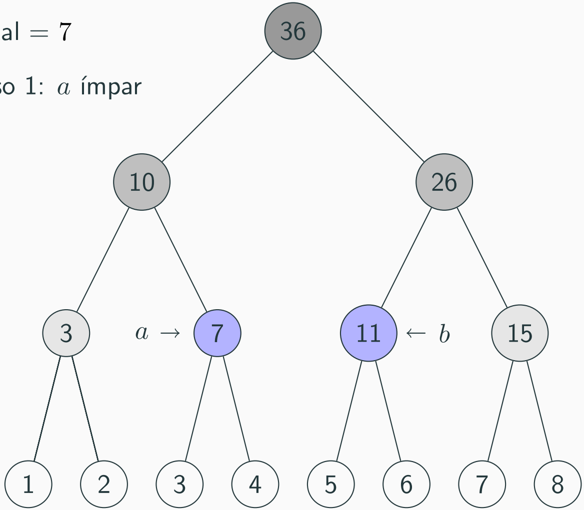
Atualização



Visualização de RSQ(2, 6)

Total $= \mathbf{7}$

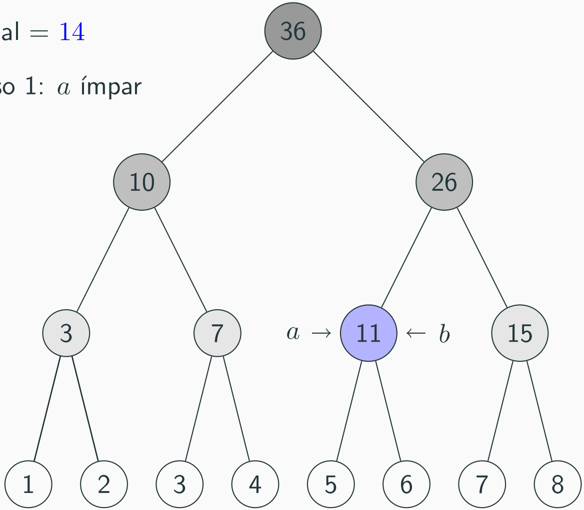
Caso 1: a ímpar



Visualização de RSQ(2, 6)

Total $= 14$

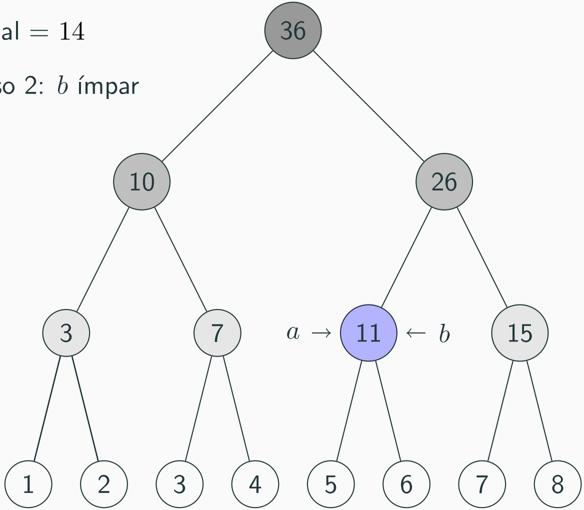
Caso 1: a ímpar



Visualização de RSQ(2, 6)

Total $= 14$

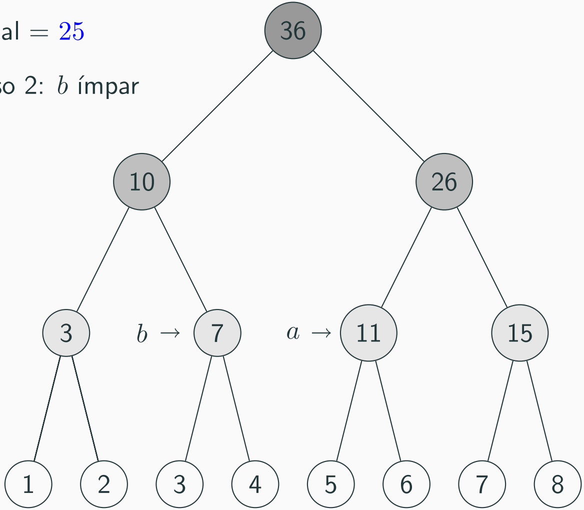
Caso 2: b ímpar



Visualização de RSQ(2, 6)

Total $= \color{blue}{25}$

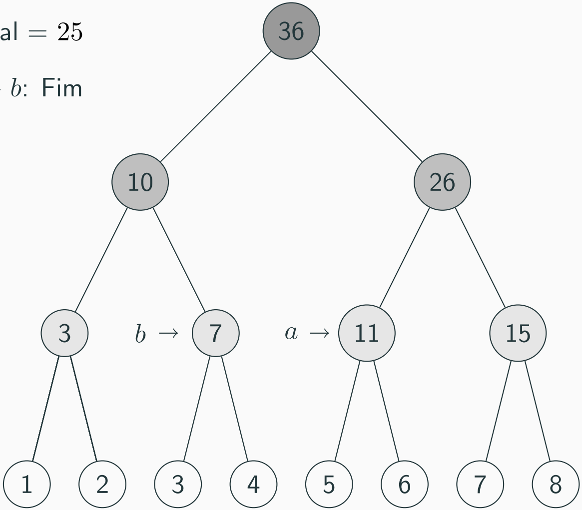
Caso 2: b ímpar



Visualização de RSQ(2, 6)

Total $= 25$

$a > b$: Fim



Implementação da *range query*

```
    T RSQ(int i, int j)
    {
        // As folhas estão na segunda metade de ns
        int a = i + N, b = j + N;
        T s = 0;

        while (a <= b)
        {
            if (a & 1)
                s += ns[a++];

            if (not (b & 1))
                s += ns[b--];

            a /= 2;
            b /= 2;
        }

        return s;
    }
```

Referências

1. **HALIM**, Felix; **HALIM**, Steve. *Competitive Programming 3*, 2010.
2. **LAAKSONEN**, Antti. *Competitive Programmer's Handbook*, 2018.