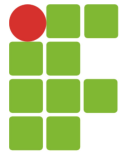
Árvores de Segmentos

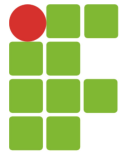
Tópicos Especiais em Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

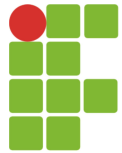
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

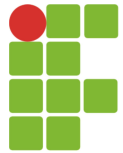


Sumário



Introdução

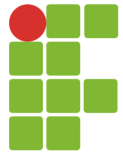
- Considere o seguinte problema, dado uma entrada $V[0, n-1]$ e indices i, j, determinar o índice k, $i \leq k \leq j$ tal que $V[k]$ seja o menor valor possível de V sobre o intervalo $[i, j]$.
- Em caso de empate, pegamos o k mais à esquerda possível.
- Consulta de mínimos sobre intervalos (**R**ange-**M**inimum-**Q**ueries ou RMQ).



RMQ

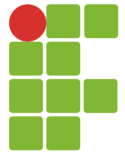
RMQ

- Entrada: $V[0, n-1]$ e índices i, j, $0 \leq i \leq j < n$;
- Saída: $\text{RMQ}_V(i, j) = \min\{\arg\min\{V[k] | i \leq k \leq j\}\}$



Algoritmo Força-Bruta

- Imediatamente conseguimos elaborar um algoritmo força-bruta.
- Basta varrer o vetor $V[i, j]$.



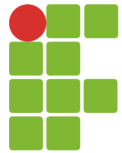
Algoritmo Força-Bruta

Algorithm 1: BRUTE-FORCE-RMQ

Input: $V[0, n-1], i, j, 0 \leq i \leq j < n$

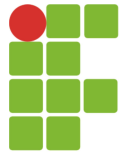
Output: $\text{RMQ}_V(i, j)$

```
1 (k, min_k) ← (i, V[i])
2 for( l ← i + 1; l ≤ j; l++ )
3     if( V[l] < min_k )
4         (k, min_k) ← (l, V[l])
5 return k
```



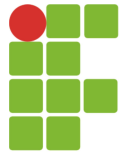
Algoritmo Força-Bruta

- Análise de pior-caso: $\Theta(n)$.
- No pior caso, para responder uma consulta precisamos olhar para toda a entrada.
- Precisamos de um método mais eficiente.

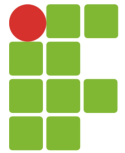


Em Busca de um Método mais Eficiente

- **Alternativa:** Árvores de Segmentos.
- Complexidade de pior caso $\langle \Theta(n), \Theta(\lg n) \rangle$.
- Suporta atualização!

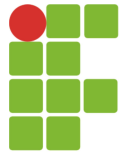


Sumário



Árvore de Segmentos

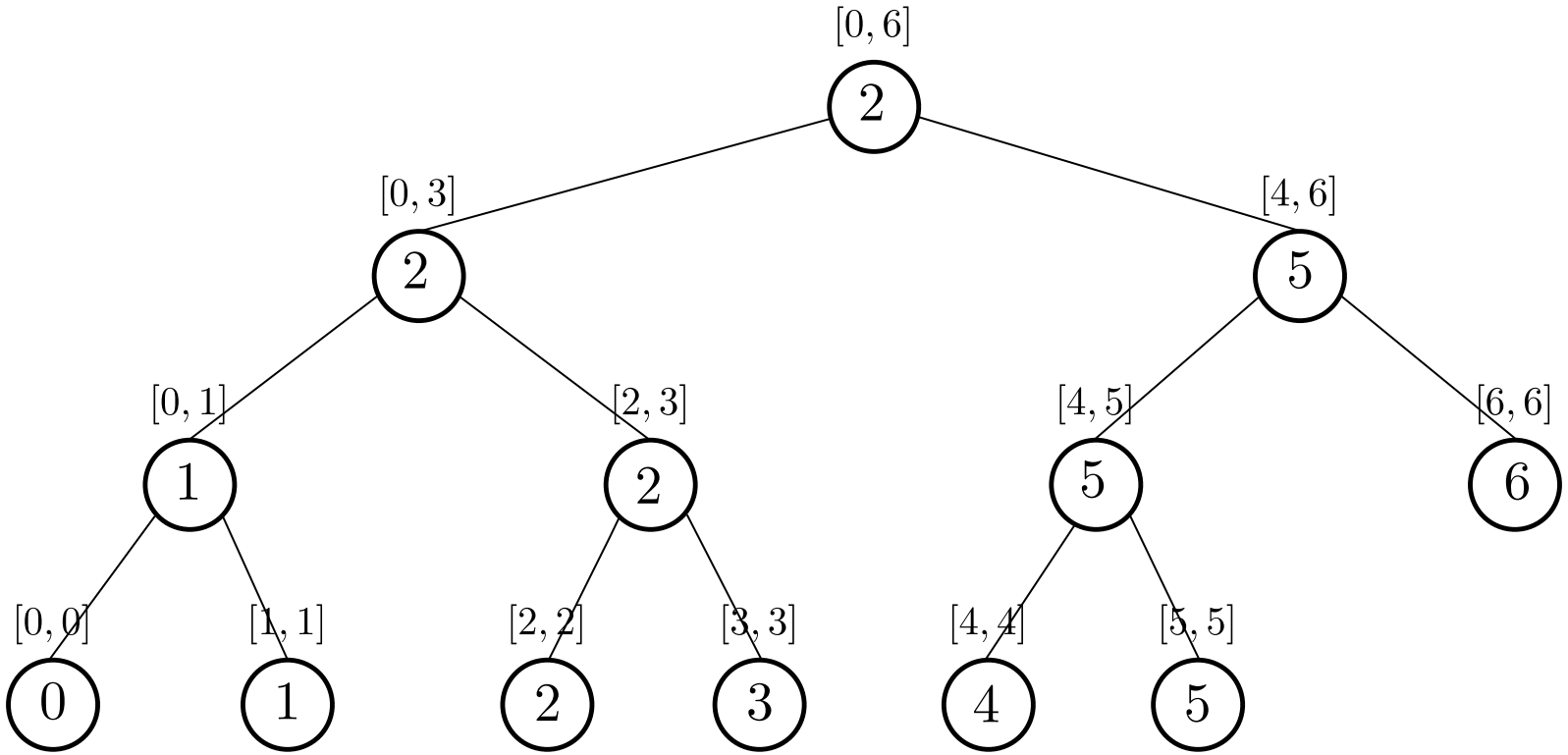
- Em uma árvore de segmentos, cada nó está associado a um intervalo da sequência de entrada.
- Ao navegar na árvore, podemos responder consultas de RMQ em tempo eficiente.
- Estrutura hierárquica e de natureza recursiva!



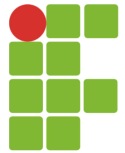
Árvore de Segmentos

- Cada nó de uma árvore de segmentos está associado a um intervalo da sequência de entrada.
- A raiz armazena o valor de $\mathrm{RMQ}_V(0, n-1)$.
- A i-ésima folha está associada ao valor de $\mathrm{RMQ}_V(i, i)$.
- Um nó interno que esteja associado ao intervalo $[i, j]$ armazena o valor $\mathrm{RMQ}_V(i, j)$. Cada nó interno tem dois filhos:
  - Filho da esquerda, definido recursivamente sobre o intervalo $[i, \lfloor (i+j)/2 \rfloor]$.
  - Filho da direita, definido recursivamente sobre o intervalo $[\lfloor (i+j)/2 \rfloor + 1, j]$.

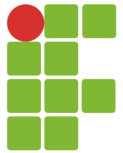
Árvore de Segmentos




```
  0  1  2  3  4  5  6
V = (18,17,12,20,16,12,21)
```

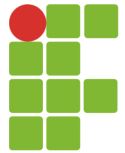


Sumário



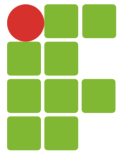
Representação de Árvores de Segmentos

- Podemos simplificar a representação das árvores de segmentos através de vetores.
- Se a entrada V tem tamanho n, a árvore de segmentos terá tamanho máximo $2n - 1$, pois cada nó interno e a raiz possuem 2 filhos.
- Utilizaremos um vetor $T[0, 4n - 1]$, com $4n$ elementos. Para representar a árvore. $T[0]$ não será utilizado. Começamos de $T[1]$ para facilitar a navegação na árvore.



Representação de Árvores de Segmentos

- Com a representação do vetor, podemos navegar na árvore conceitual.
- Seja um nó no índice i do vetor:
  - Filho da esquerda: $2i$.
  - Filho da direita: $2i + 1$



Representação de Árvores de Segmentos

Algorithm 2: $\textsc{left}(x)$

Input: x

Output: Filho da esquerda de x no vetor

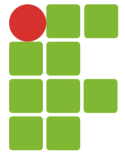
```
1 return 2x
```

Algorithm 3: $\textsc{right}(x)$

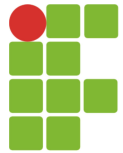
Input: x

Output: Filho da direita de x no vetor

```
1 return 2x + 1
```

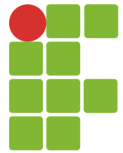


Sumário



Construção

- Para construir a árvore de segmentos, podemos adotar o seguinte procedimento recursivo.
- Caso base: o nó x é uma folha que representa o intervalo $[i, i]$. Armazena-se i em x
- Caso geral: o nó x é um nó interno (ou raiz) sobre o intervalo $[i, j]$. Recursivamente calcula-se os valores para os filhos da direita e da esquerda e, dentre os dois valores, pegamos aquele que referencia o menor valor de mínimo e o armazenamos em x.

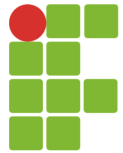


Construção

Algorithm 4: BUILD-ST(V, x, i, j)

Input: $V[0, n-1], x, i, j$
Output: $T[0, 4n-1]$

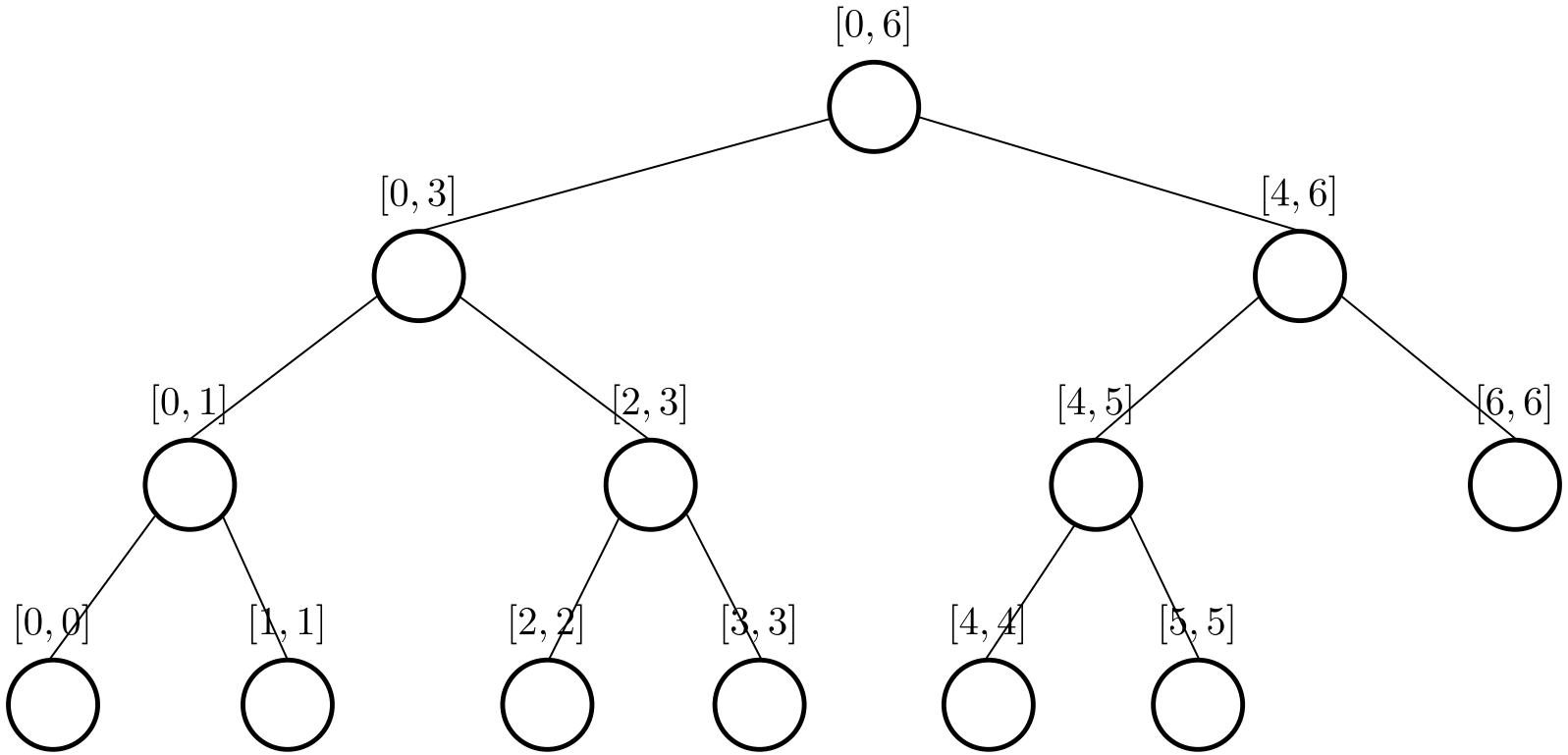
```
1  if( i = j )  T[x] ← i
2  else
3      BUILD-ST(V, LEFT(x), i, ⌊(i + j)/2⌋)
4      BUILD-ST(V, RIGHT(x), ⌊(i + j)/2⌋ + 1, j)
5      l = T[LEFT(x)]
6      r = T[RIGHT(x)]
7      if( V[l] ≤ V[r] )
8          T[x] = l
9      else
10         T[x] = r
```



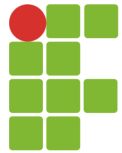
Árvores de Segmentos: Consultas de RMQ

- Chamada inicial: $\text{BUILD-ST}(T, V, 1, 0, n-1)$.

Árvores de Segmentos: Construção

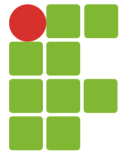



```
  0  1  2  3  4  5  6
V = (18,17,12,20,16,12,21)
```

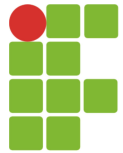


Análise: Construção

- A árvore de segmentos pode ser construída em tempo $\Theta(n)$.
- A árvore possui $2n - 1$ nós e durante a construção e processamos cada nó exatamente 1 vez com tempo de processamento constante.

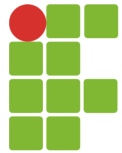


Sumário



Árvores de Segmentos: Consultas de RMQ

- Para responder consultas de RMQ, podemos navegar na árvore.
- Suponha um determinado nó x sobre um intervalo $[l, r]$. Queremos responder $\mathrm{RMQ}_V(i, j)$.
- Temos três casos:
  1. $[l, r] \cap [i, j] = \emptyset$: retorna-se indefinição, pois o nó não contribui para responder a consulta de mínimo.
  2. $[l, r] \subseteq [i, j]$: o intervalo do nó $[l, r]$ contribui parcialmente para a resposta de $\mathrm{RMQ}_V(i, j)$. Retornamos $T[x]$
  3. $[l, r] \cap [i, j] \neq \emptyset$: o nó em si não contribui para a resposta, mas os seus descendentes podem contribuir, procedemos recursivamente para os nós filhos.

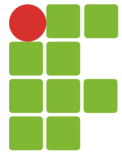


Árvores de Segmentos: Consultas de RMQ

Algorithm 5: ST-RMQ(T, V, x, l, r, i, j)

Input: $T[0, 2n-1], V[0, n-1], x, l, r, i, j$
Output: $\text{RMQ}_V(i, j)$

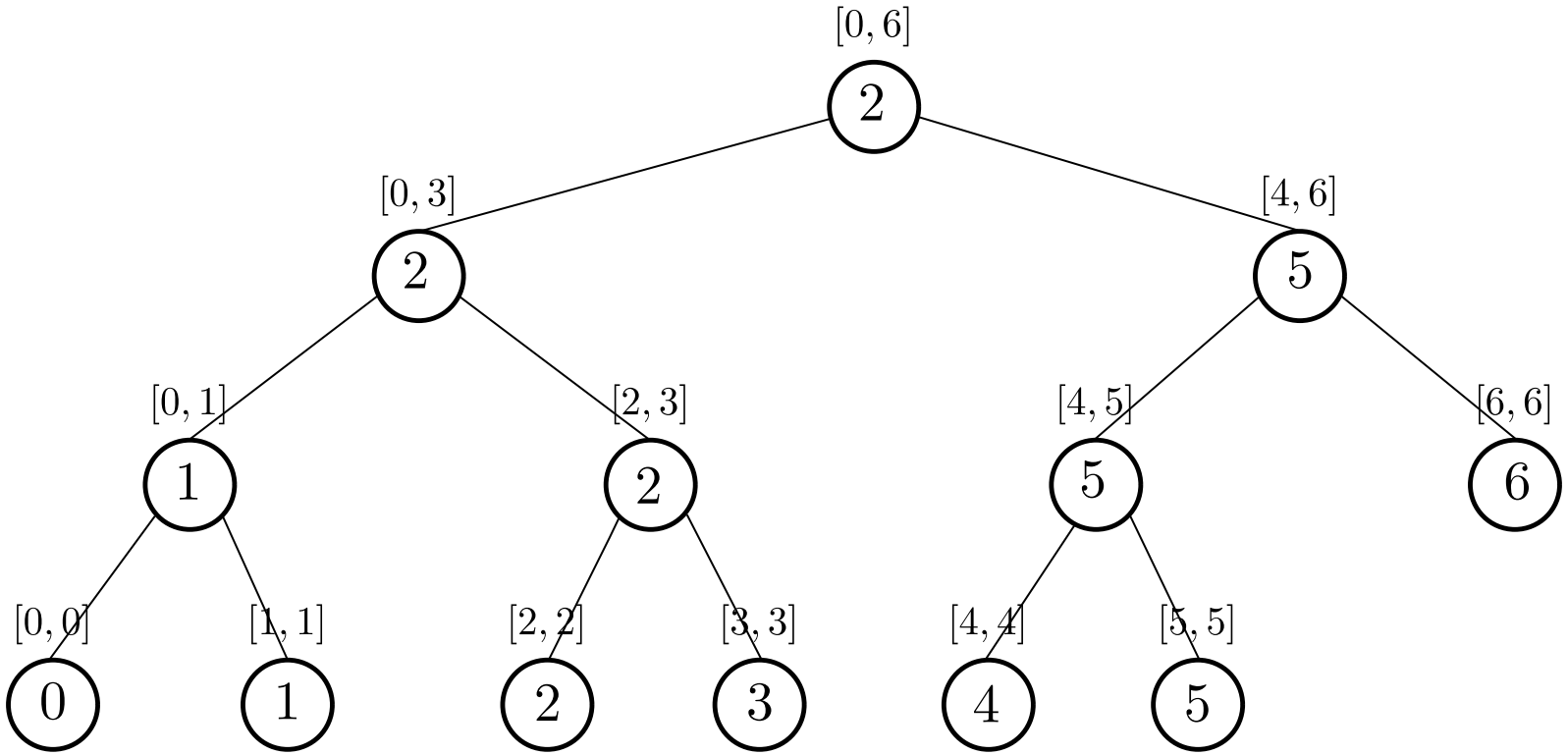
```
1 if( i > r ∨ j < l )  return ⊥
2 else if( l ≥ i ∧ r ≤ j )  return T[x]
3 rmq_l ← ST-RMQ(T, V, LEFT(x), l, ⌊(l + r)/2⌋, i, j)
4 rmq_r ← ST-RMQ(T, V, RIGHT(x), ⌊(l + r)/2⌋ + 1, r, i, j)
5 if( rmq_l = ⊥ ) return rmq_r
6 else if( rmq_r = ⊥ ) return rmq_l
7 if( V[rmq_l] ≤ V[rmq_r] )
8     return rmq_l
9 return rmq_r
```



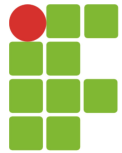
Árvores de Segmentos: Consultas de RMQ

- Chamada inicial: $\text{ST-RMQ}(T, V, 1, 0, n-1, i, j)$.

Árvores de Segmentos: Consultas de RMQ

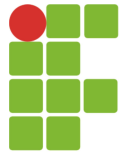



```
      0  1  2  3  4  5  6
V = (18,17,12,20,16,12,21)
```



Análise: Consultas de RMQ

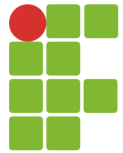
- Durante a consulta sobre a árvore, visita-se no máximo 4 nós por nível.
- Como são $\Theta(\lg n)$ níveis, o tempo total da consulta é $\Theta(\lg n)$.



Análise: Consultas de RMQ

Teorema

No máximo 4 nós por nível são visitados durante a consulta de $\mathrm{RMQ}_V(i, j)$.

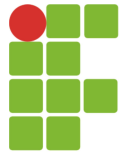


Análise: Consultas de RMQ

Demonstração.

A prova é por indução.

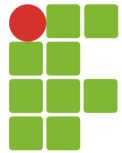
- Caso base: no primeiro nível apenas a raiz é acessada.
- Hipótese de Indução: no i-ésimo nível $n \leq 4$ nós são visitados.
- Passo de indução: precisamos mostrar que no $i+1$-ésimo nível $n \leq 4$ nós são acessados.



Análise: Consultas de RMQ

Demonstração.

Se no i-ésimo nível 1 ou 2 nós são visitados, teremos que no $i+1$-ésimo nível acessamos no máximo 4 nós. Só precisasmos nos preocupar no caso em que acessamos 3 ou 4 nós no i-ésimo nível.



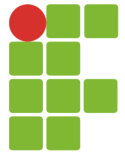
Análise: Consultas de RMQ

Demonstração.

Se no i-ésimo nível 1 ou 2 nós são visitados, teremos que no $i+1$-ésimo nível acessamos no máximo 4 nós. Só precisasmos nos preocupar no caso em que acessamos 3 ou 4 nós no i-ésimo nível.

Suponha que visitemos $n = 4$ nós no i-ésimo nível, v_1, v_2, v_3, v_4. A união dos intervalos cobertos por estes representam um intervalo $[l, r] \supseteq [i, j]$.

Estes nós também estão dispostos consecutivamente na árvore, quando lidos da esquerda para a direita.



Análise: Consultas de RMQ

Demonstração.

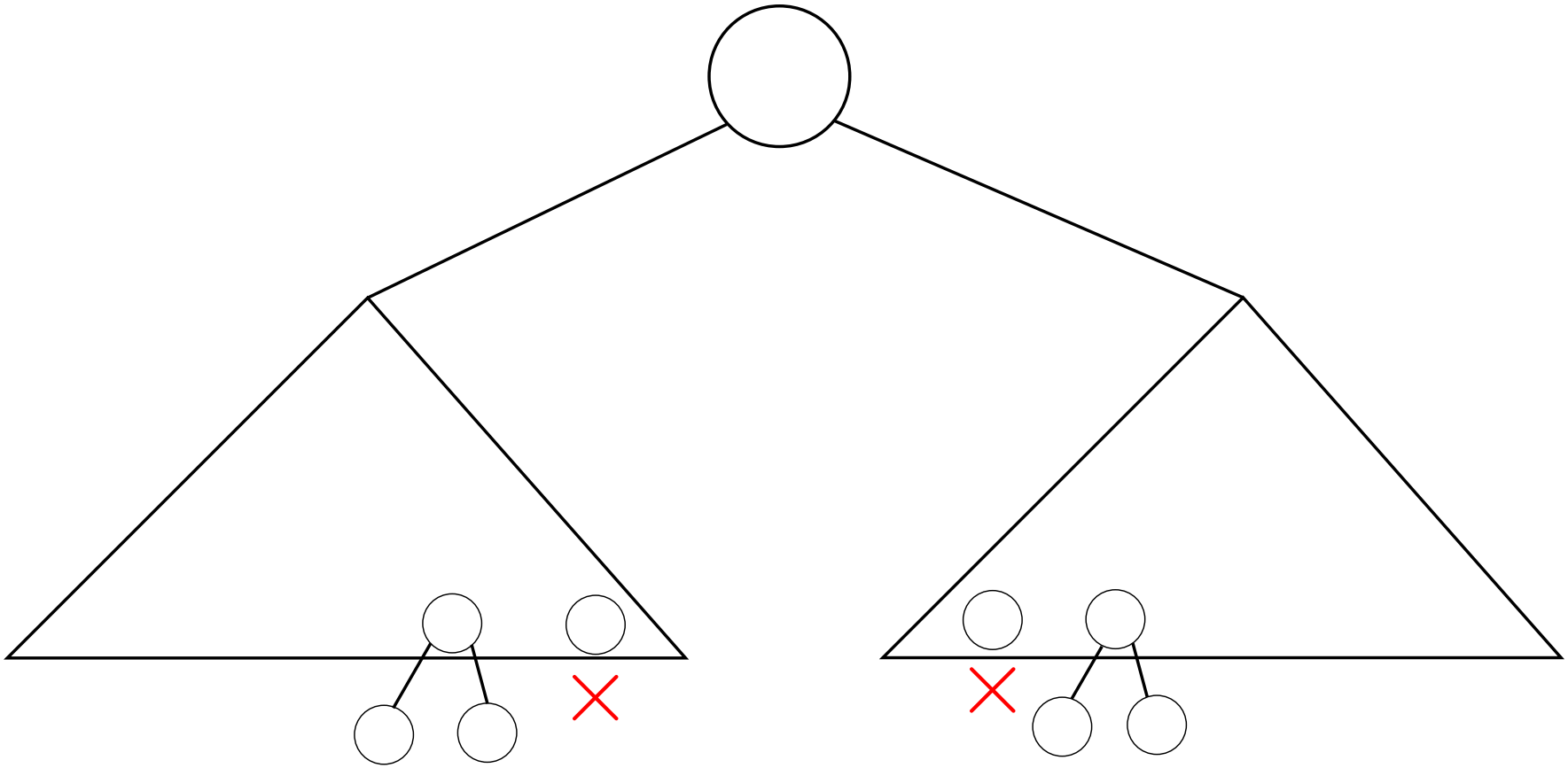
Podemos dizer que v_2 e v_3, os nós do meio, cobrem intervalos $[l', r'] \subset [i, j]$ e $[l'', r''] \subset [i, j]$ isto é, ambos os intervalos estão contidos em $[i, j]$. Mais do que isto, podemos dizer que $[l', r'] \cup [l'', r''] \subset [i, j]$. Caso contrário, a existência de v_1 e v_2 seria negada, uma vez que v_3 e v_4 já cobririam todo o intervalo $[l, r]$

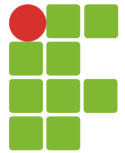
Logo, os nós v_2 e v_3 não geram chamadas recursivas.

Conclui-se que apenas os nós v_1 e v_2 podem gerar, no máximo, 2 chamadas recursivas cada e portanto, no $i + 1$-ésimo nível, no máximo 4 nós são visitados.

Análise: Consultas de RMQ

Demonstração.

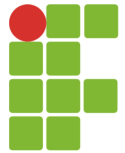




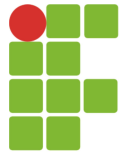
Análise: Consultas de RMQ

Demonstração.

A prova para $n = 3$ é análoga, mas no caso, apenas o nó v_2 não gera chamadas recursivas, enquanto v_1 e v_3 potencialmente geram duas chamadas recursivas cada. □

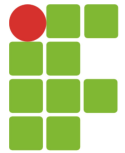


Sumário



Árvores de Segmentos: Atualização

- Supondo que um valor $V[k]$ seja atualizado, como propagar essa atualização na árvore de segmentos?
- Atualização *bottom-up*. Primeiro atualizamos a folha que corresponde ao intervalo $[k, k]$ e depois atualizamos os ancestrais desta folha na volta da recursão.
- Todos os nós do caminho da raiz até a folha afetada são (potencialmente) atualizados.



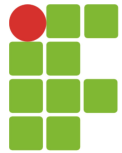
Árvores de Segmentos: Atualização

Algorithm 6: ST-UPDATE$(T, V, x, l, r, k, value)$

Input: $T[0, 2n-1], V, x, l, r, k, value$

Output: T e V atualizados.

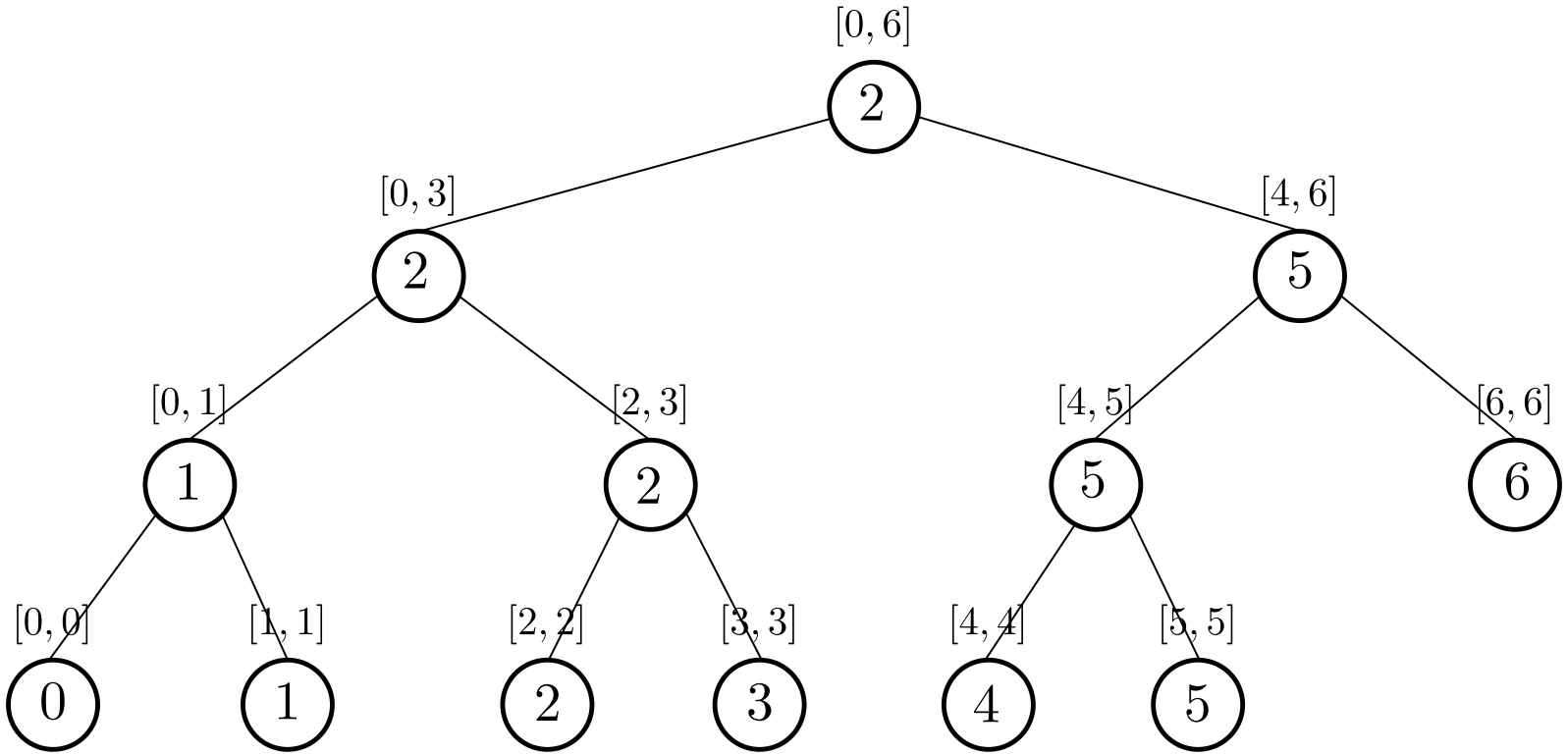
```
1  if( l = r )
2  |   V[k] ← value
3  |   return
4  mid ← ⌊(l + r)/2⌋
5  if( k ≤ mid )  ST-UPDATE(T, LEFT(x), l, mid, k, value)
6  else ST-UPDATE(T, RIGHT(x), mid + 1, r, k, value)
7  value_l ← T[LEFT(x)]
8  value_r ← T[RIGHT(x)]
9  if( V[value_l] ≤ V[value_r] )  T[x] ← value_l
10 else  T[x] ← value_r
```



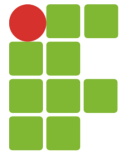
Árvore de Segmentos: Atualização

- Chamada inicial: $\text{ST-UPDATE}(T, V, 1, 0, n-1, k, value)$.

Árvores de Segmentos: Atualização

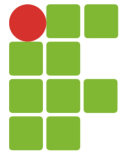



```
    0  1  2  3  4  5  6
V = (18,17,12,20,16,12,21)
```

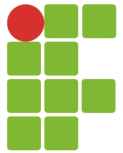


Análise: Atualização

- No máximo 1 nó em cada nível é atualizado.
- Temos $\Theta(\lg n)$ níveis.
- Tempo de atualização $\Theta(\lg n)$.



Sumário



Análise: Árvores de Segmentos

- Estrutura extremamente poderosa para responder $\mathrm{RMQ}_V(i, j)$ em tempo eficiente.
- Pode ser modificada para outros problemas que exigem outros tipos de consultas sobre intervalos. É uma estrutura bastante flexível.
- Tempo de construção: $\Theta(n)$.
- Tempo de consulta: $\Theta(\lg n)$.
- Tempo de atualização: $\Theta(\lg n)$.
- $\langle \Theta(n), \Theta(\lg n), \Theta(\lg n) \rangle$.