Árvores

Árvore Binária de Busca

(Fonte: Material adaptado dos Slides do prof. Monael.)

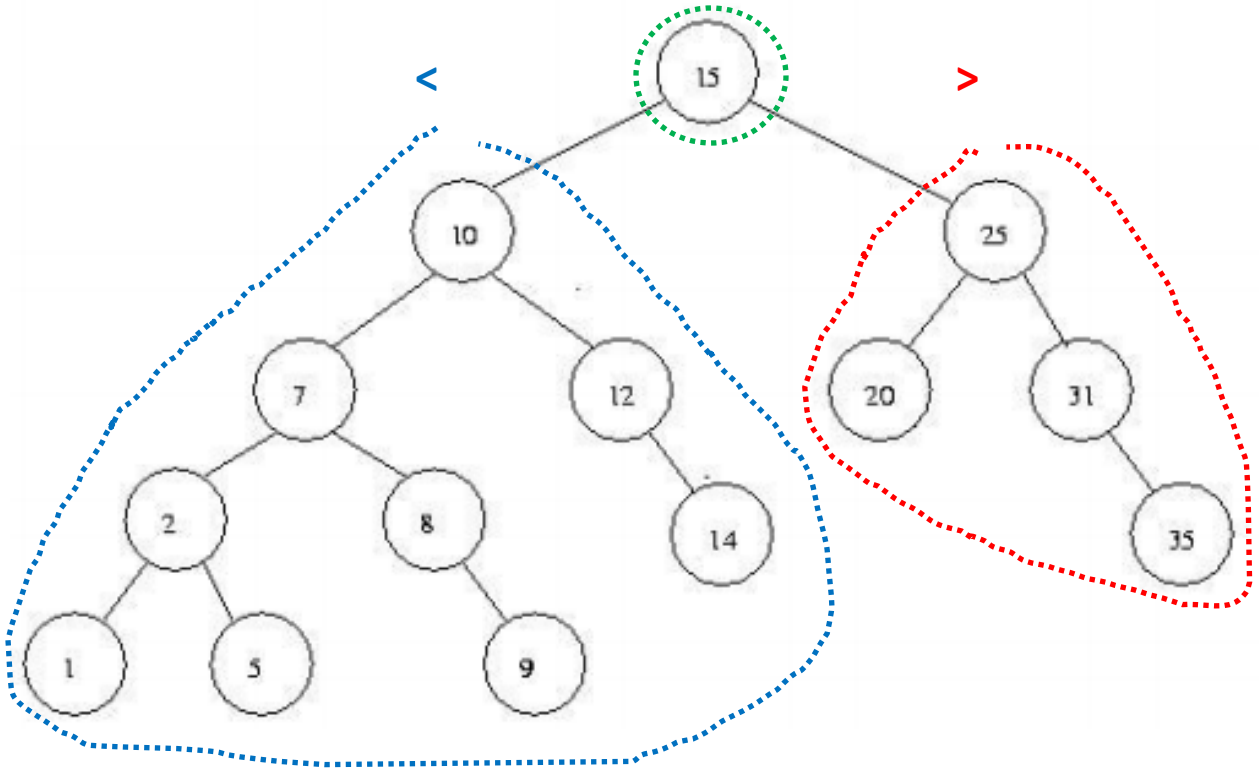
Árvores

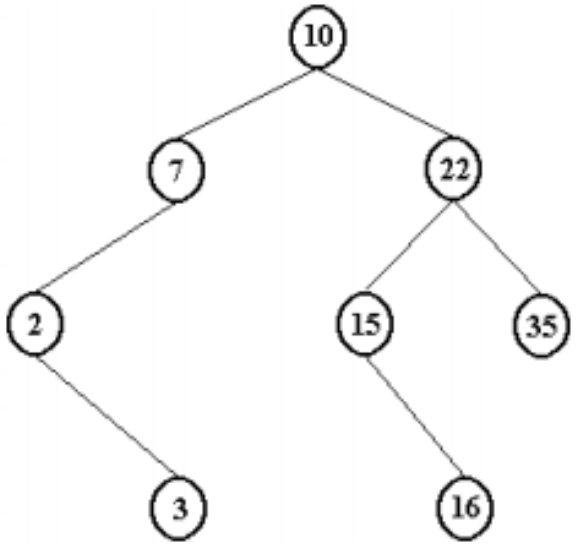
- Motivação:
  - Por que usar árvores?
    - Custo

Operação	Listas	Árvores
Inserção	$O(n)$	$O(\log_b n)$
Remoção	$O(n)$	$O(\log_b n)$
Busca	$O(n)$	$O(\log_b n)$

Árvore Binária de Busca

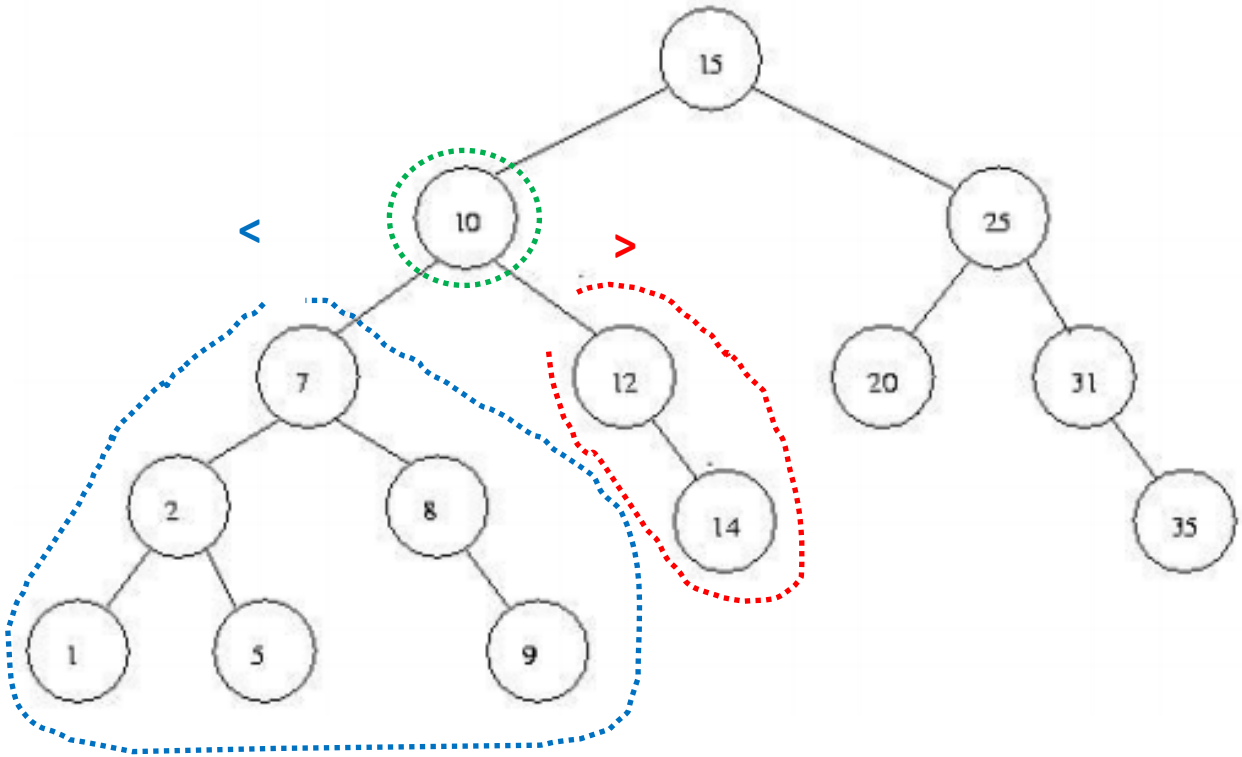
- Definição:
  - Árvore Binária de Busca
    - É um caso particular de Árvore Binária, onde todos nós da subárvore esquerda do nó k são menores que k, e todos os nós da subárvore direita de k são maiores que k. E tanto a subárvore da direita como a da esquerda também são árvores binárias de busca.

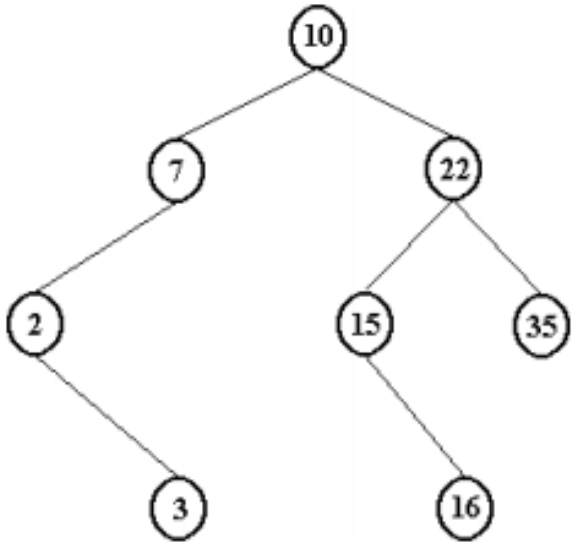




Árvore Binária de Busca

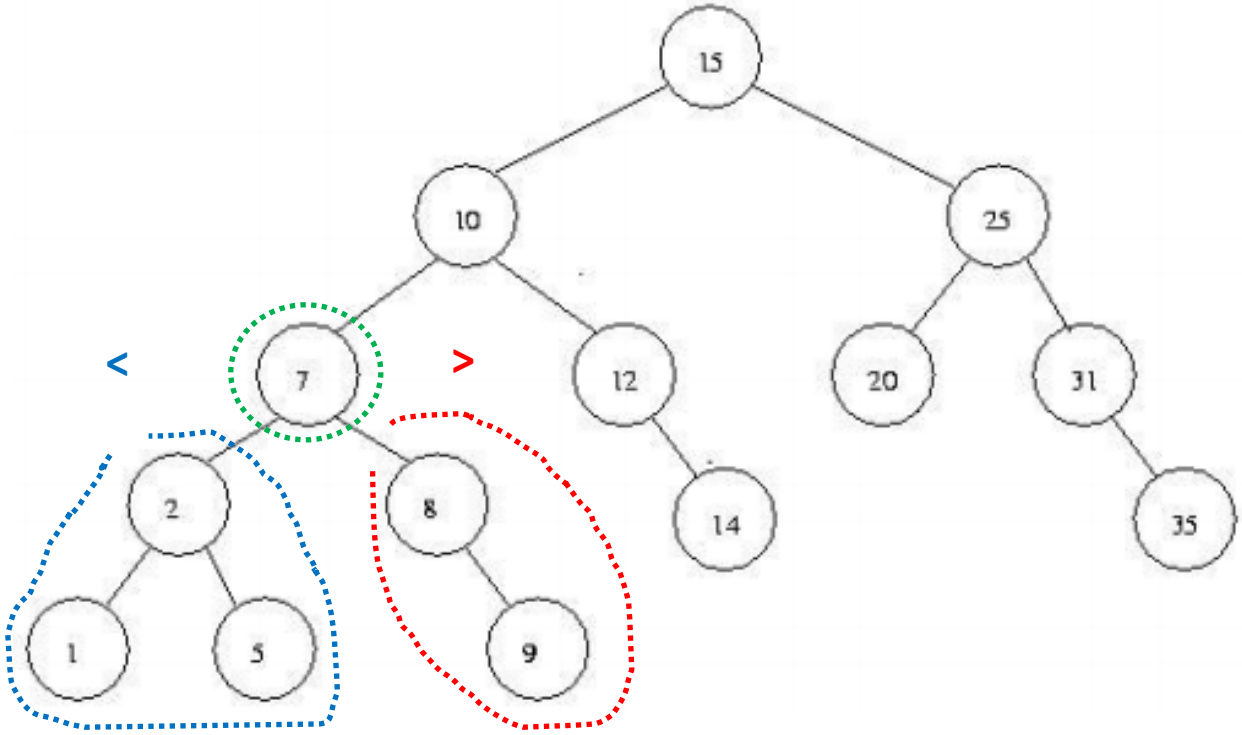
- Definição:
  - Árvore Binária de Busca
    - É um caso particular de Árvore Binária, onde todos nós da subárvore esquerda do nó k são menores que k, e todos os nós da subárvore direita de k são maiores que k. E tanto a subárvore da direita como a da esquerda também são árvores binárias de busca.

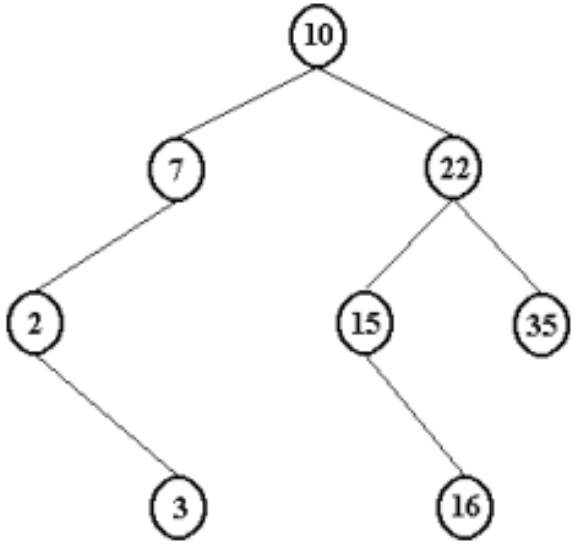




Árvore Binária de Busca

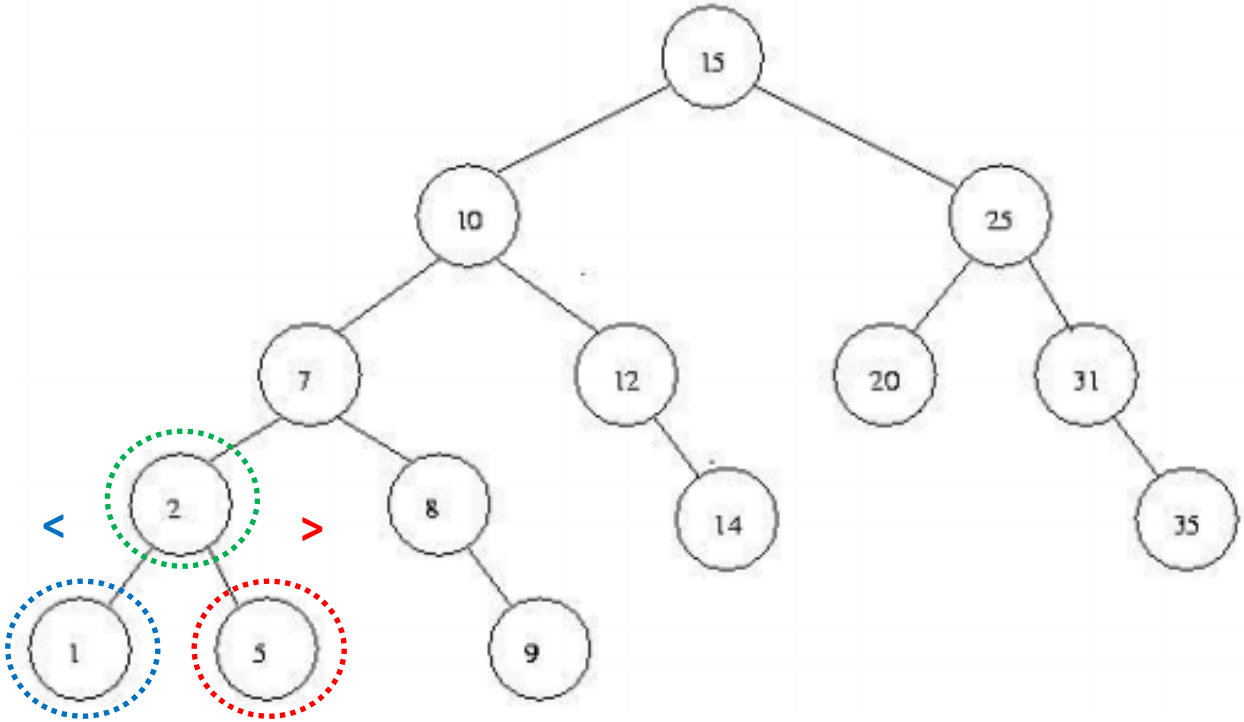
- Definição:
  - Árvore Binária de Busca
    - É um caso particular de Árvore Binária, onde todos nós da subárvore esquerda do nó k são menores que k, e todos os nós da subárvore direita de k são maiores que k. E tanto a subárvore da direita como a da esquerda também são árvores binárias de busca.

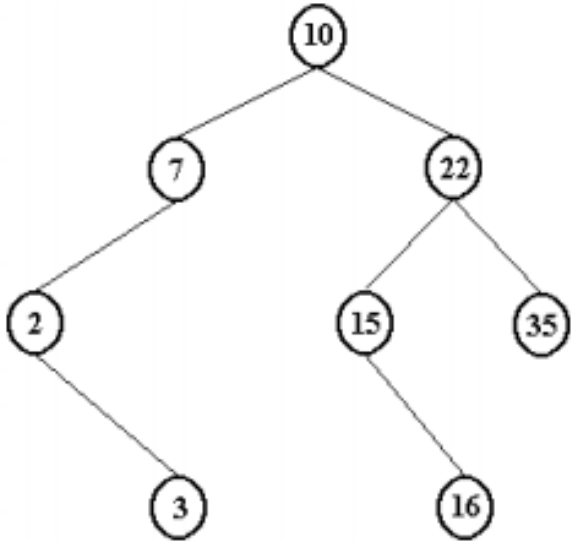




Árvore Binária de Busca

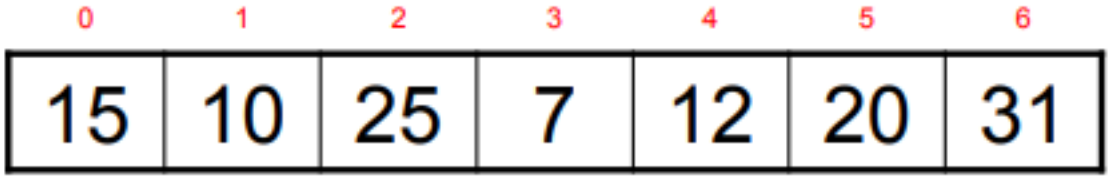
- Definição:
  - Árvore Binária de Busca
    - É um caso particular de Árvore Binária, onde todos nós da subárvore esquerda do nó k são menores que k, e todos os nós da subárvore direita de k são maiores que k. E tanto a subárvore da direita como a da esquerda também são árvores binárias de busca.



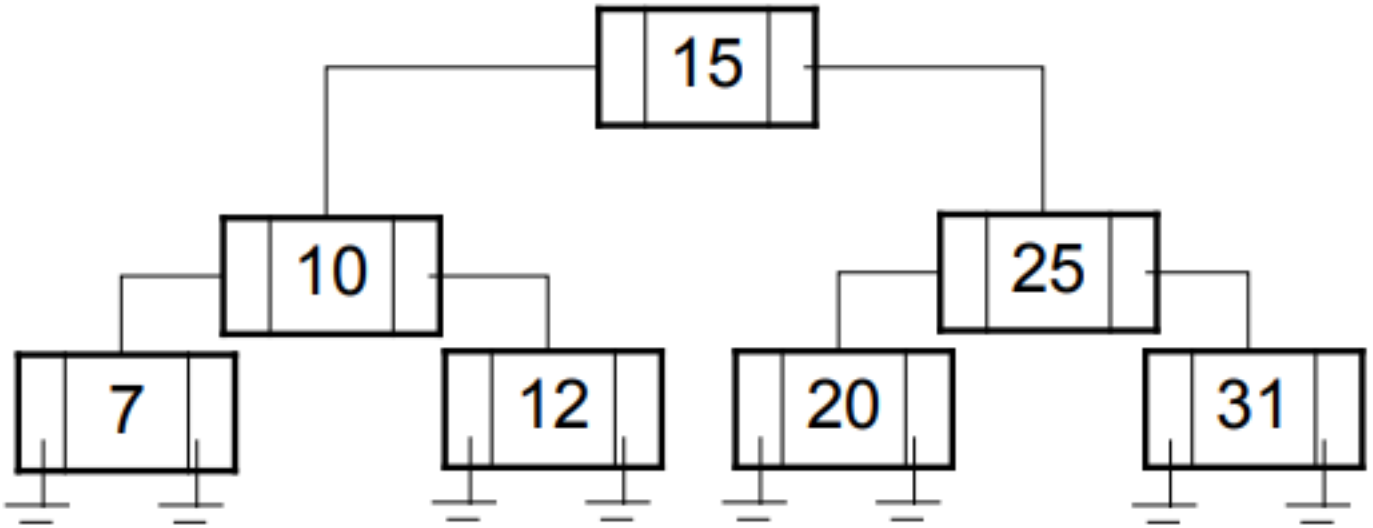


Árvore Binária de Busca

- Exemplo de Árvore Binária de Busca Estática:

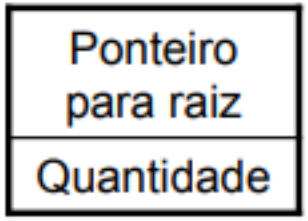


- Exemplo de Árvore Binária de Busca Dinâmica:

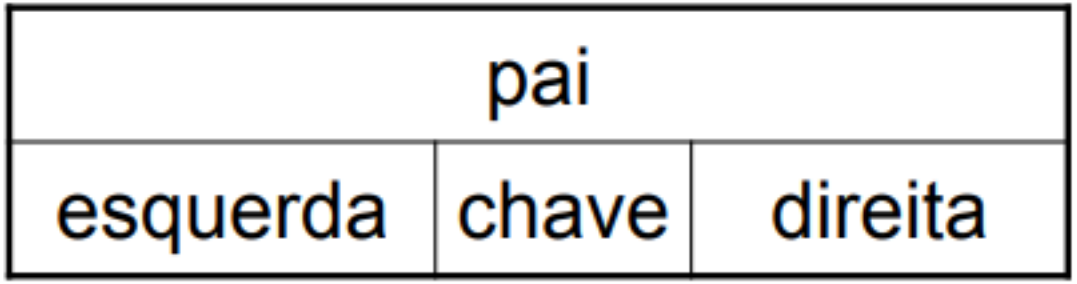


Árvore Binária de Busca

- Em Ciência da Computação uma árvore binária de busca dinâmica é uma estrutura de dados que:
  - Consiste de uma seqüencia de registros
  - Cada registro tem quatro campos:
    - uma campo para a chave.
    - um ponteiro para o filho da esquerda (subárvore da esquerda)
    - um ponteiro para o filho da direita (subárvore da direita)
    - um ponteiro para o pai. (nó pai)



Árvore



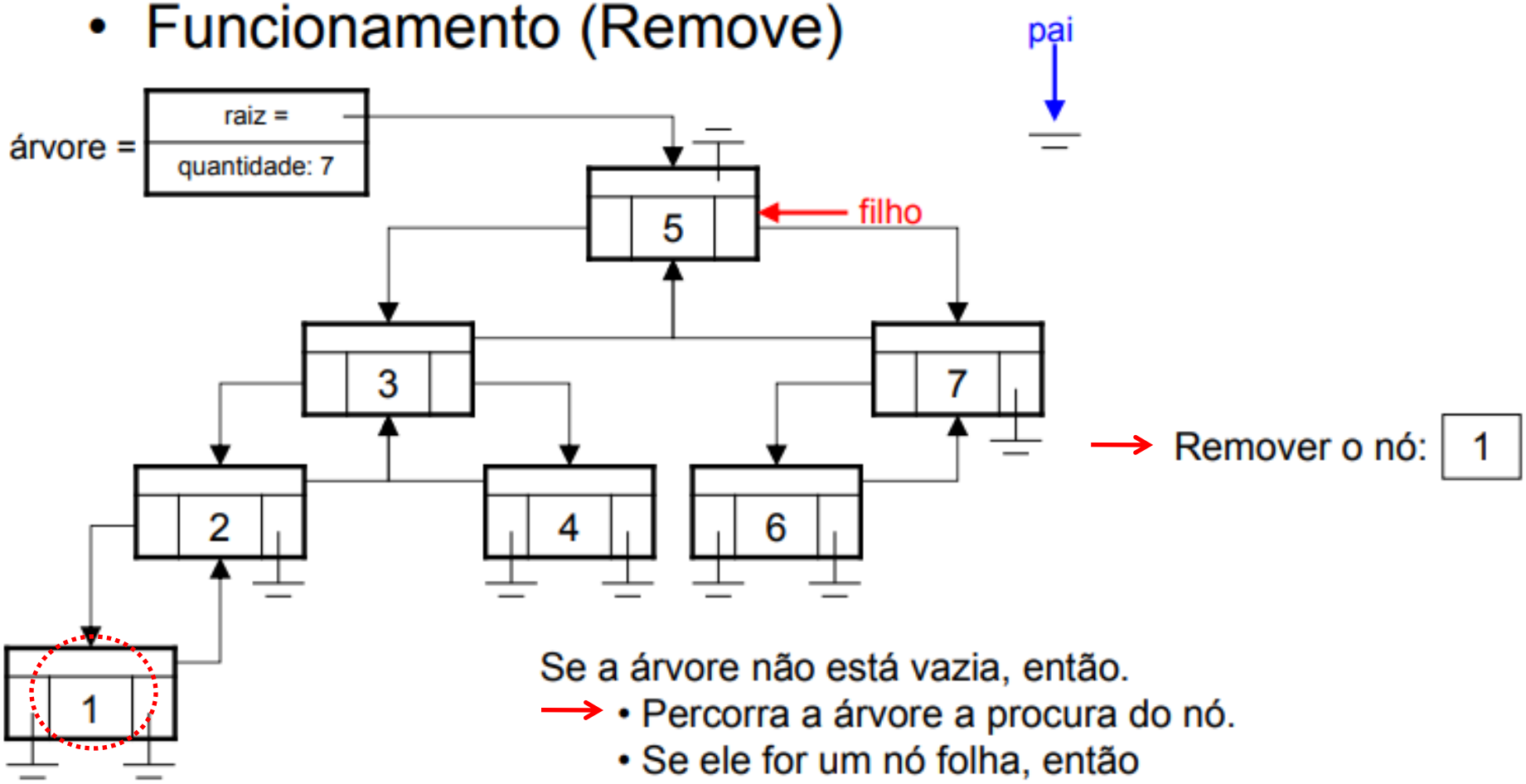
Nó da árvore

Árvore Binária de Busca

- Operações Básicas
  - Inserção de nó na Árvore.
  - Eliminação de nó da Árvore.
  - Percorrer nós da Árvore.

Árvore Binária de Busca

- Funcionamento (Remove)

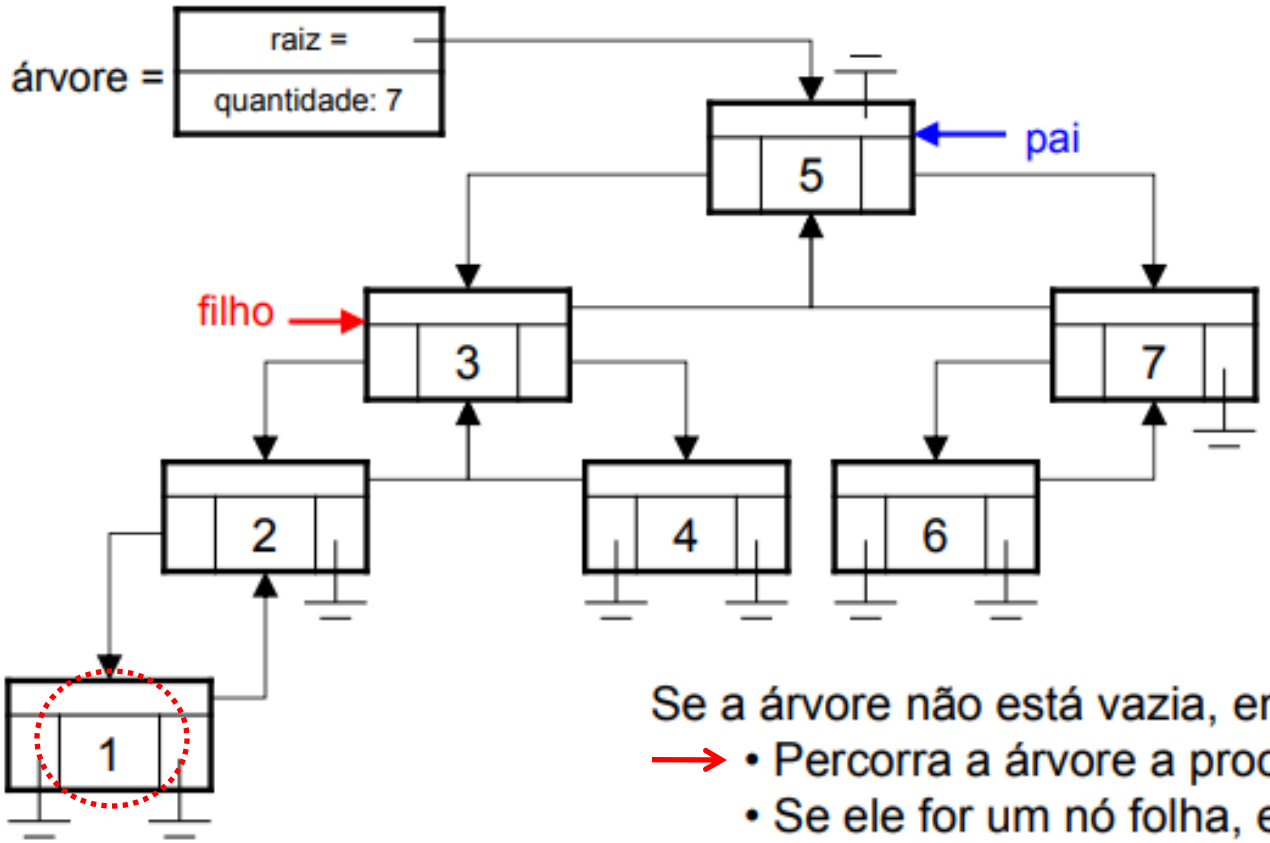


Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó folha, então
  - Faça o pai do nó a ser removido apontar para NULL.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



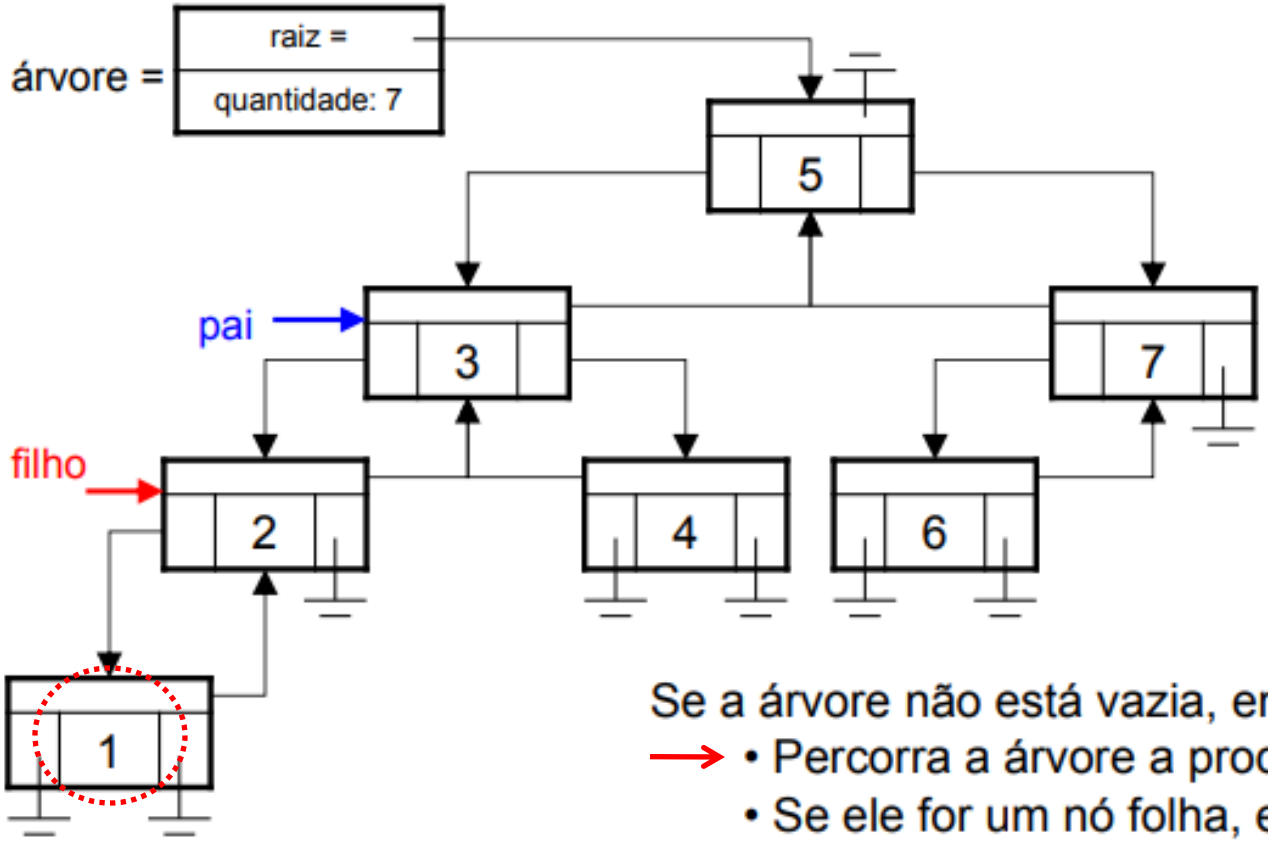
Remover o nó: 1

Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó folha, então
  - Faça o pai do nó a ser removido apontar para NULL.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



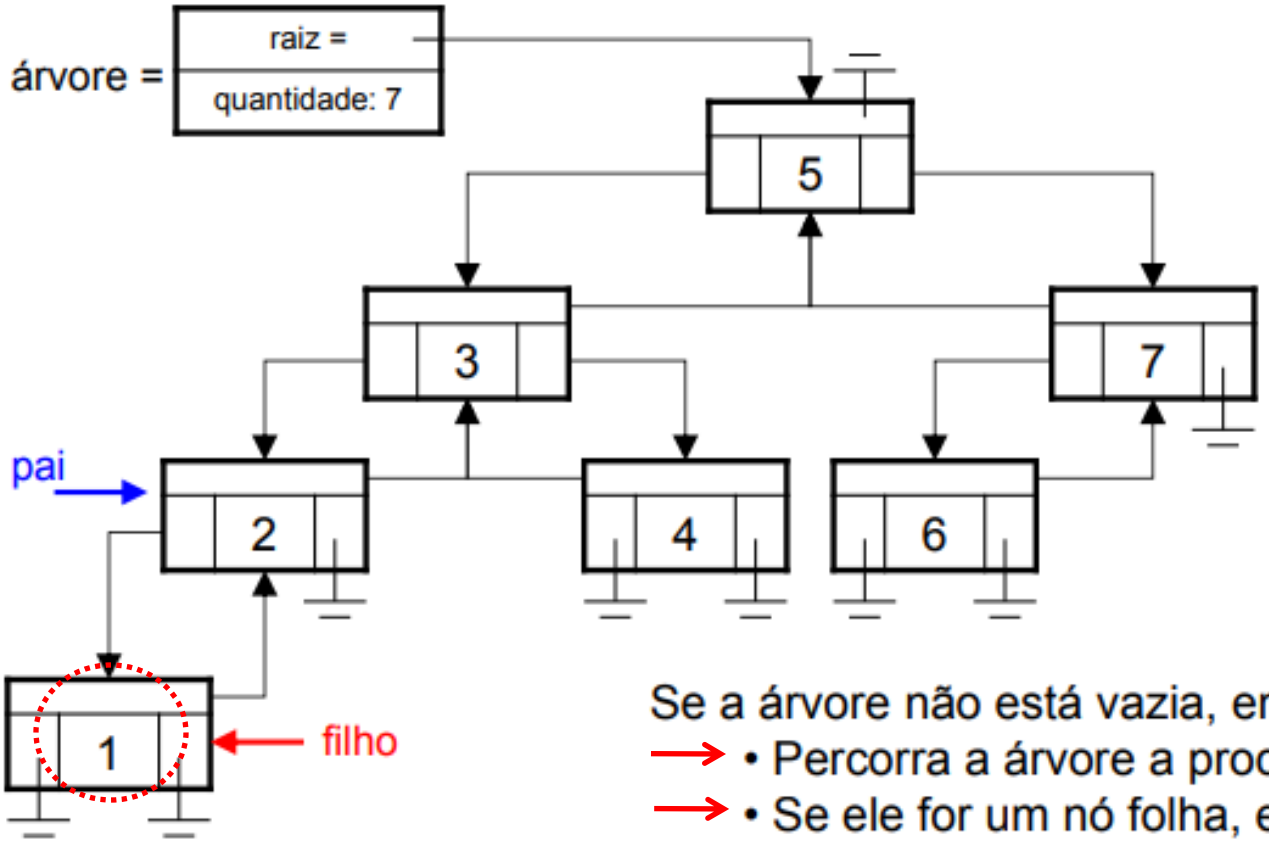
Remover o nó: 1

Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó folha, então
  - Faça o pai do nó a ser removido apontar para NULL.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



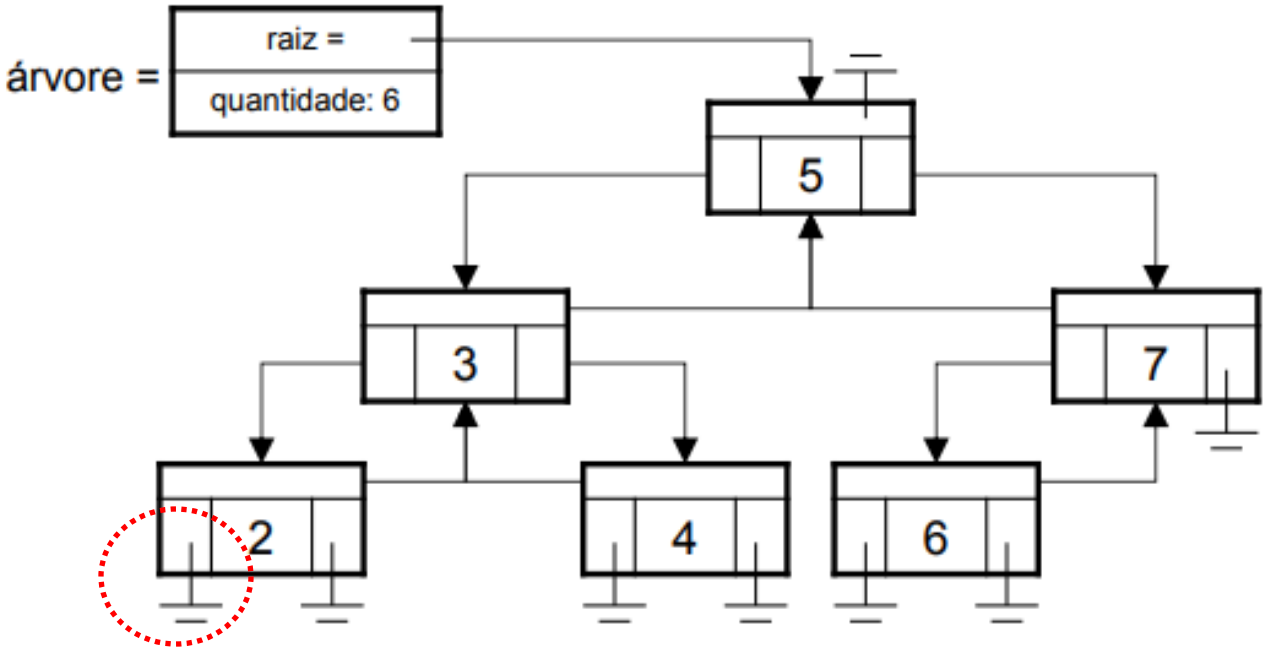
Remover o nó: 1

Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó folha, então
  - Faça o pai do nó a ser removido apontar para NULL.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



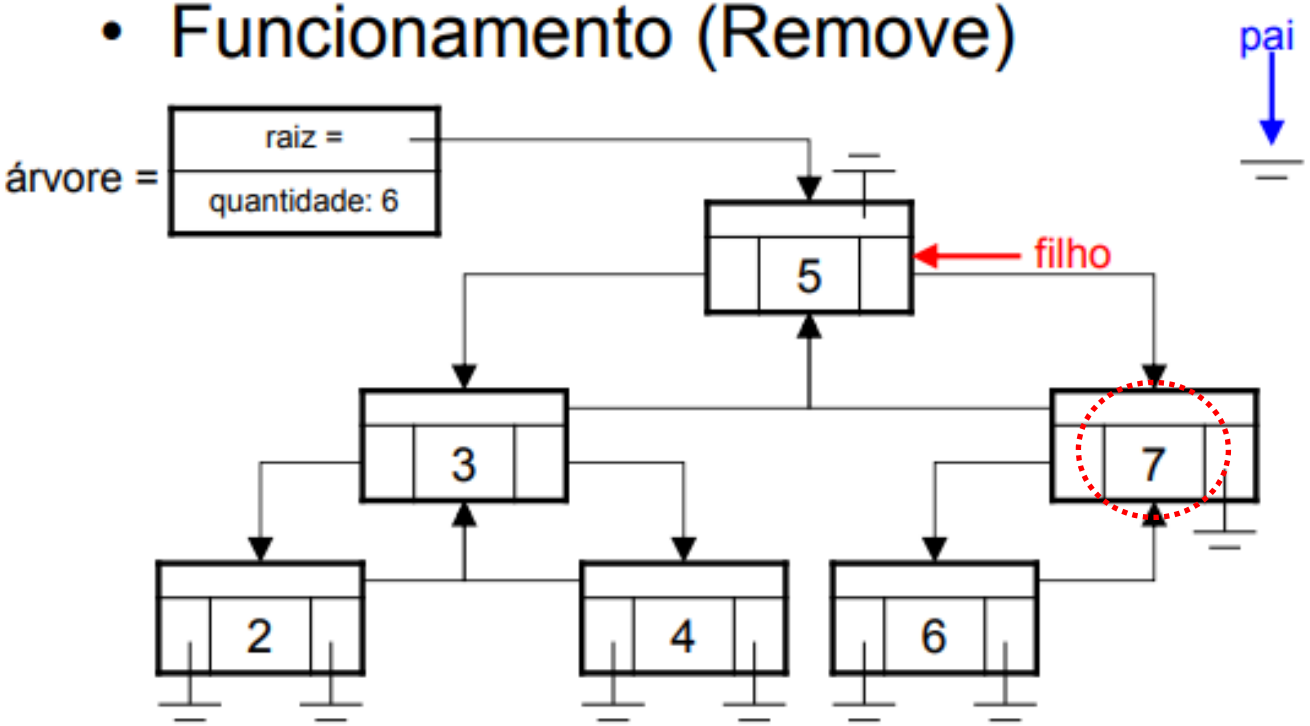
Remover o nó: 1

Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- → Se ele for um nó folha, então
  - → Faça o pai do nó a ser removido apontar para NULL.
- → Desaloque a memória do nó a ser removido
- → Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



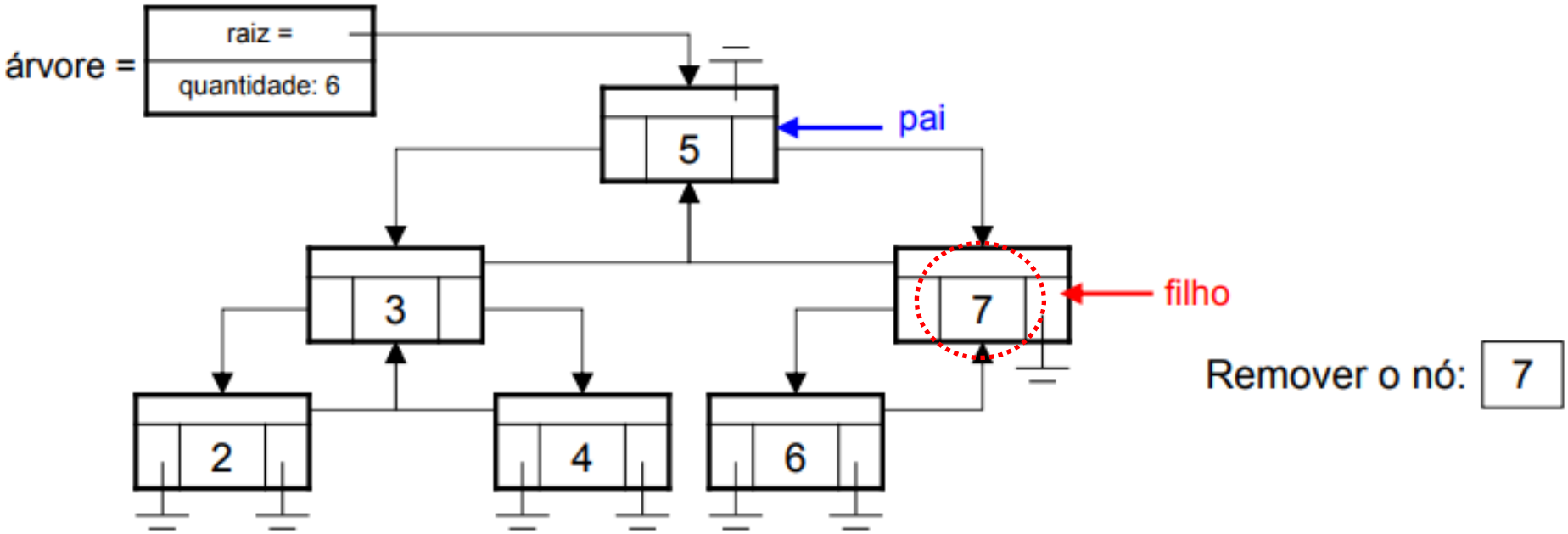
→ Remover o nó: 7

Se a árvore não está vazia, então.

→ • Percorra a árvore a procura do nó.
- Se ele for um nó galho com apenas um filho, então
  - Faça o pai do nó a ser removido apontar para o neto.
  - Faça o neto apontar para o avô como pai.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)

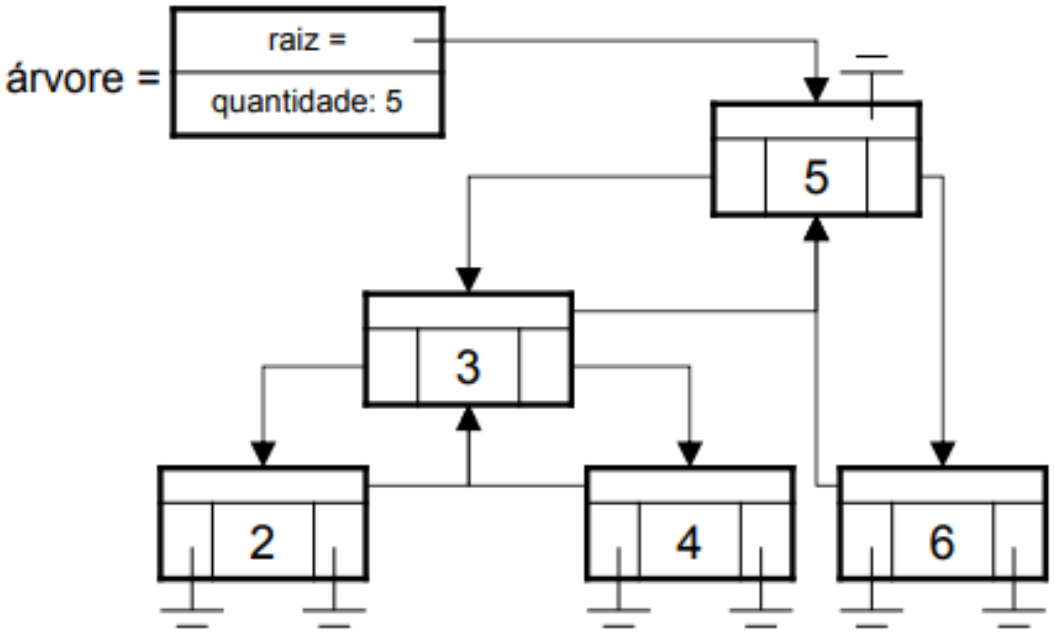


Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó galho com apenas um filho, então
  - Faça o pai do nó a ser removido apontar para o neto.
  - Faça o neto apontar para o avô como pai.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



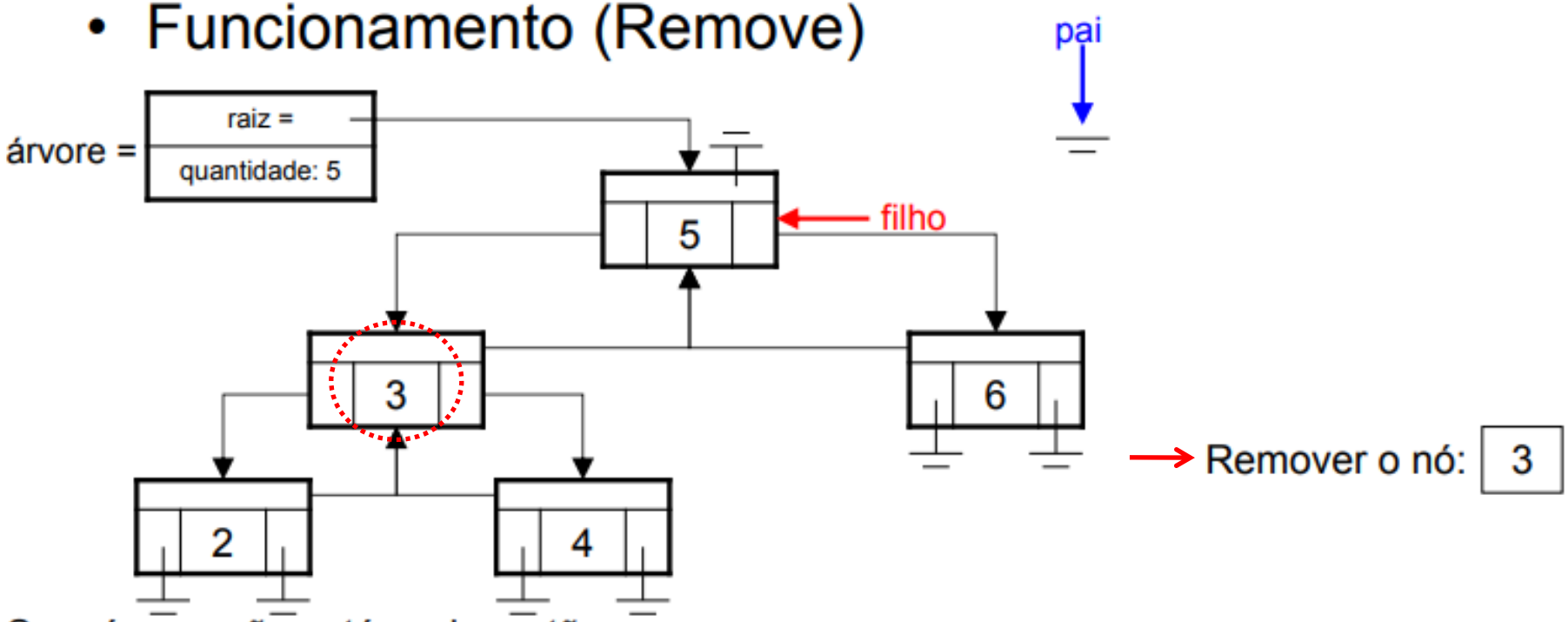
Remover o nó: 7

Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó galho com apenas um filho, então
  - Faça o pai do nó a ser removido apontar para o neto.
  - Faça o neto apontar para o avô como pai.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)

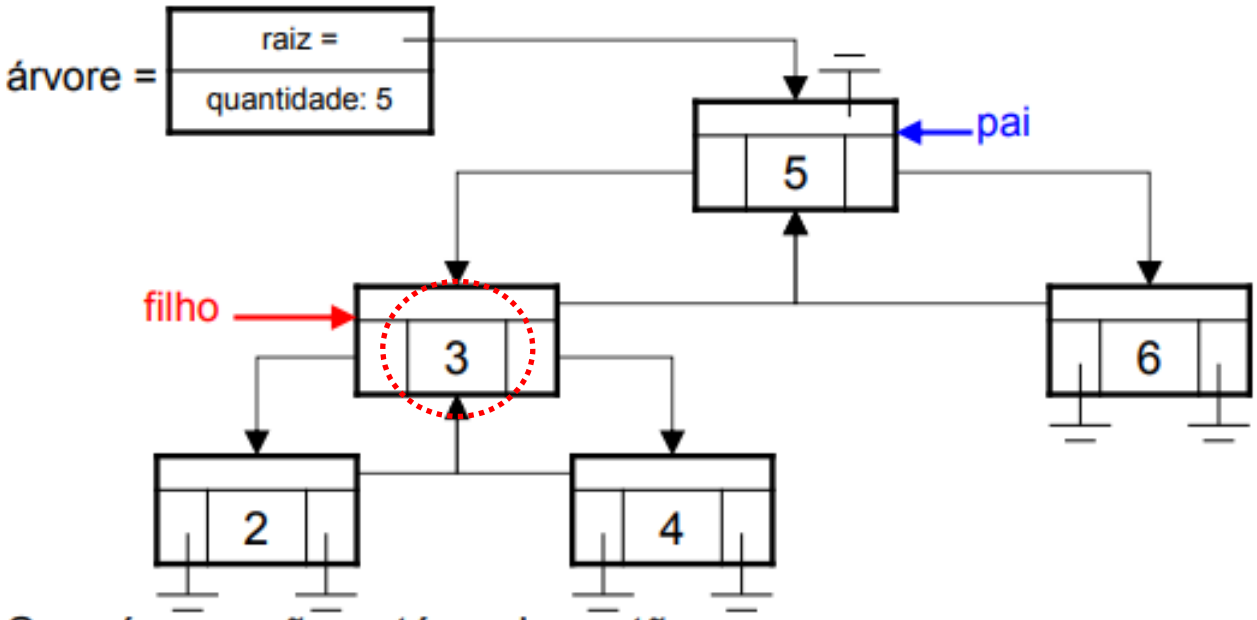


Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó galho com dois filho, então
  - Busque o antecessor ou o sucessor do nó a ser removido.
  - Se o sucessor ou antecessor do nó a ser removido tiver filhos, chame o procedimento recursivamente.
  - Caso contrário, substitua o nó a ser removido pelo sucessor ou antecessor.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



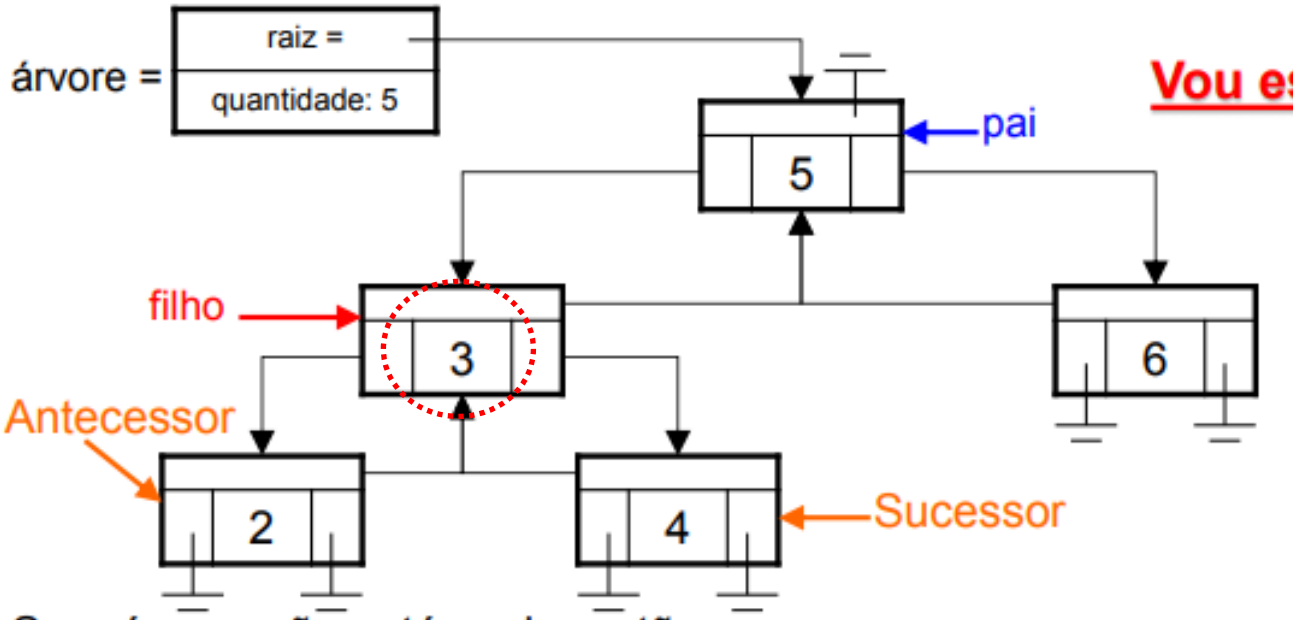
Remover o nó: 3

Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó galho com dois filho, então
  - Busque o antecessor ou o sucessor do nó a ser removido.
  - Se o sucessor ou antecessor do nó a ser removido tiver filhos, chame o procedimento recursivamente
  - Caso contrário, substitua o nó a ser removido pelo sucessor ou antecessor.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



Vou escolher o antecessor

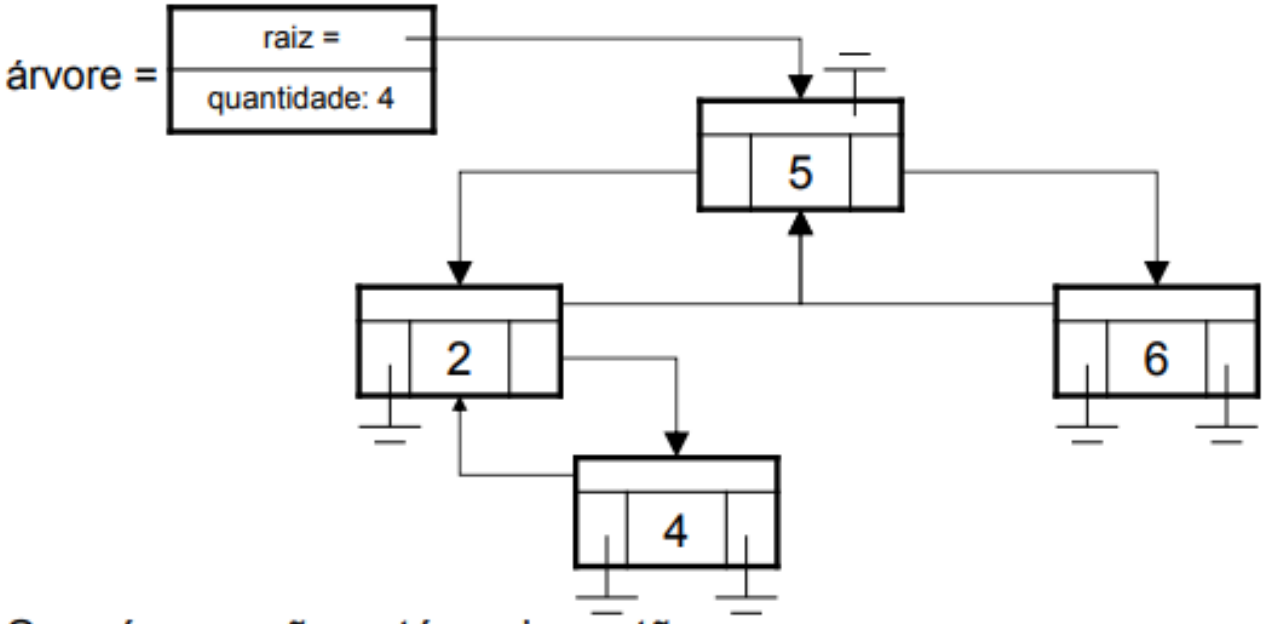
Remover o nó: 3

Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó galho com dois filho, então
  - Busque o antecessor ou o sucessor do nó a ser removido.
  - Se o sucessor ou antecessor do nó a ser removido tiver filhos, chame o procedimento recursivamente.
  - Caso contrário, substitua o nó a ser removido pelo sucessor ou antecessor.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

- Funcionamento (Remove)



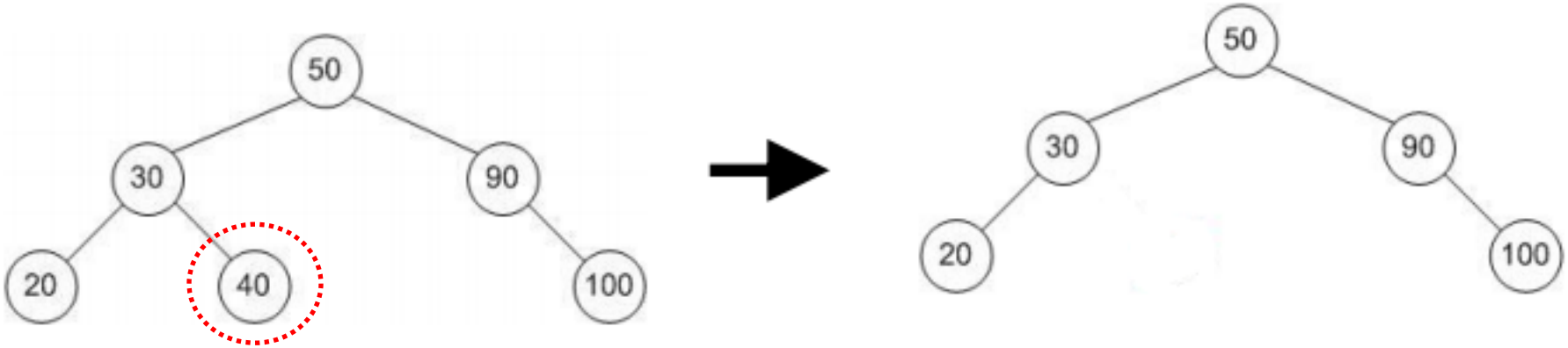
Remover o nó: 3

Se a árvore não está vazia, então.

- Percorra a árvore a procura do nó.
- Se ele for um nó galho com dois filho, então
  - Busque o antecessor ou o sucessor do nó a ser removido.
  - Se o sucessor ou antecessor do nó a ser removido tiver filhos, chame o procedimento recursivamente
  - Caso contrário, substitua o nó a ser removido pelo sucessor ou antecessor.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Árvore Binária de Busca

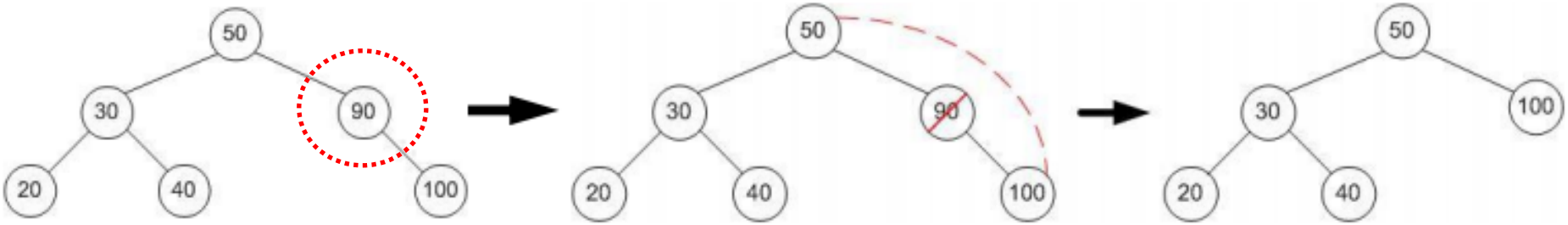
- Remoção (Lembretes)
  - Caso 1: Removendo um nó folha.



- Localize o nó a ser removido.
- Se for um nó folha então
  - Se não for a raiz
    - » Faça o pai do nó a ser removido apontar para NULL.
  - Se for a raiz
    - » Faça o campo raiz da árvore apontar para NULL

Árvore Binária de Busca

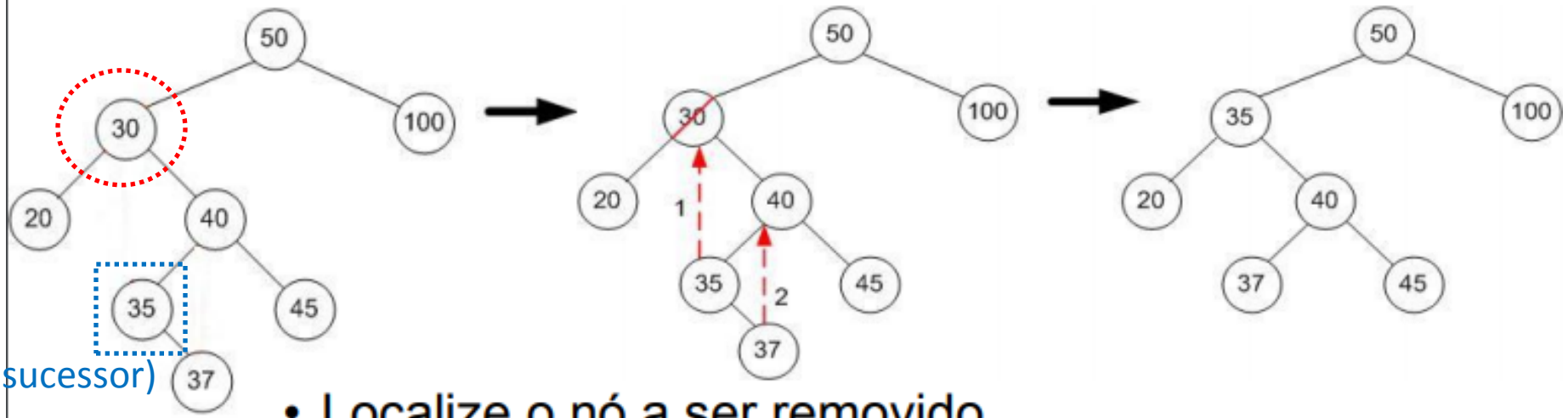
- Remoção (Lembretes)
  - Caso 2: Removendo um nó com 1 filho.



- Localize o nó a ser removido.
- Se o nó tiver apenas 1 filho
  - Se não for a raiz
    - » Faça o pai do nó a ser removido apontar para o neto.
  - Se for a raiz
    - » Faça o campo raiz da árvore apontar para o neto.

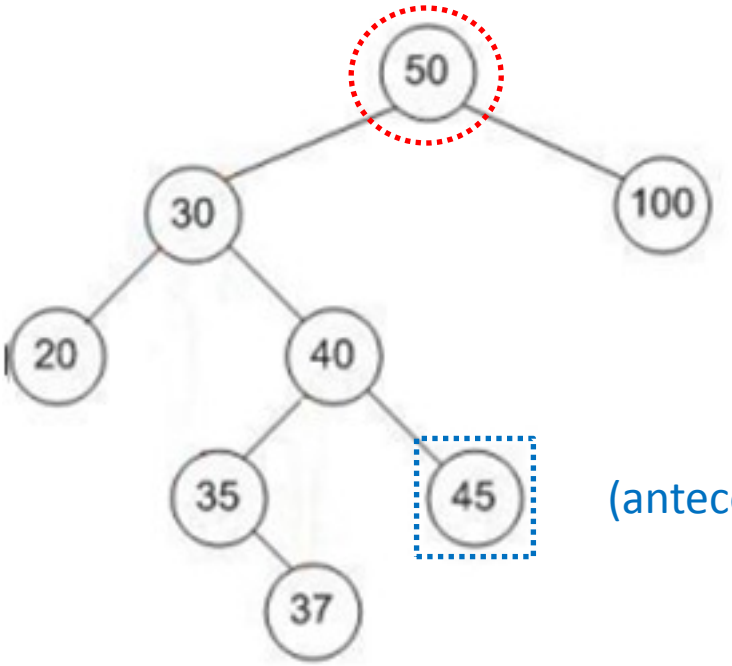
Árvore Binária de Busca

- Remoção (Lembretes)
  - Caso 3: Removendo um nó com 2 filhos.



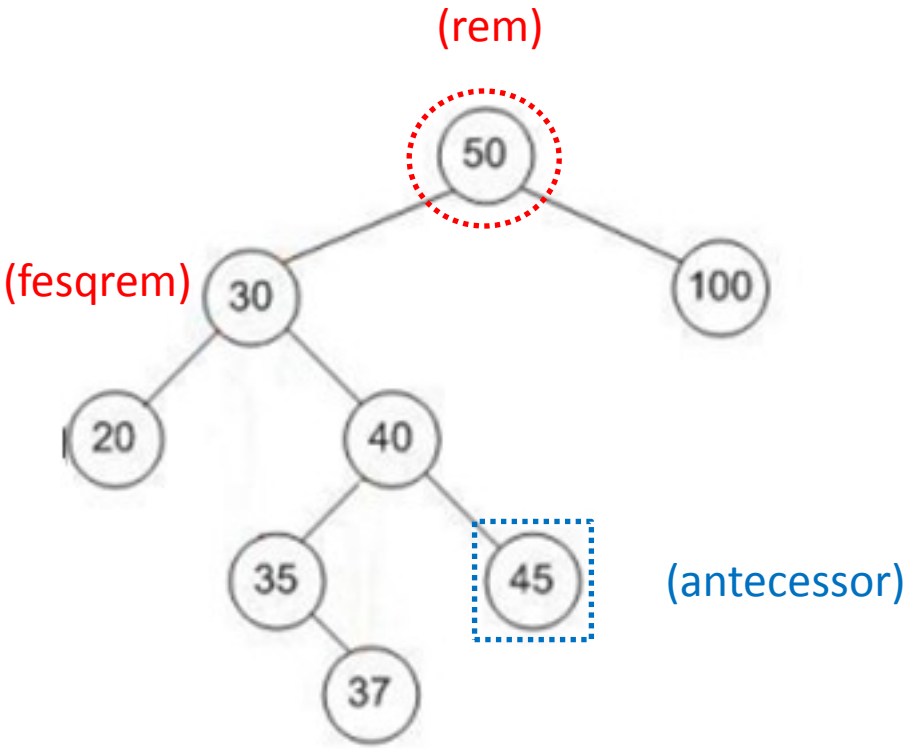
- Localize o nó a ser removido.
- Se o nó tiver 2 filhos
  - Busque o antecessor ou o sucessor.
  - Substitua o nó a ser removido pelo antecessor ou sucessor.

Antecessor



(antecessor)

Antecessor



Árvore Binária de Busca

(e REMOVE)

- Busca o Antecessor do nó.

```
1.  struct tNo * antecessor(struct tNo *fesqrem)   {
2.        struct tNo *ant, *pt = fesqrem;
3.        while(pt != NULL)   {
4.              ant = pt;
5.              pt = pt->direita;
6.        }
7.        if(filhoEsquerda(ant) != NULL)   {
8.            if(ehFilhoEsquerda(ant)==1)      {
9.                  pai(ant)->esquerda = ant->esquerda;
10.           }
11.           else    {
12.                 pai(ant)->direita = ant->esquerda;
13.           }
14.           (ant->esquerda)->pai = pai(ant);
15.       }
16.       else   {
17.           if(ehFilhoDireita(ant))   {
18.               pai(ant)->direita = NULL;
19.           }
20.           else    {
21.               pai(ant)->esquerda = NULL;
22.           }
23.       }
24.       return ant;
25. }
```

Descendo à direita na subárvore esquerda, até localizar o antecessor.

Árvore Binária de Busca

(e REMOVE)

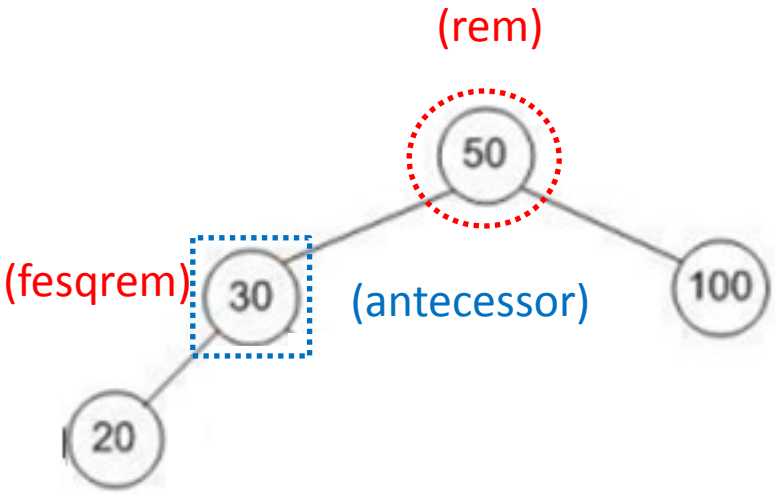
- Busca o Antecessor do nó.

```
1.  struct tNo * antecessor(struct tNo *fesqrem)   {
2.        struct tNo *ant, *pt = fesqrem;
3.        while(pt != NULL)   {
4.              ant = pt;
5.              pt = pt->direita;
6.        }
7.        if(filhoEsquerda(ant) != NULL)   {
8.           if(ehFilhoEsquerda(ant)==1)      {
9.                 pai(ant)->esquerda = ant->esquerda;
10.          }
11.          else    {
12.                pai(ant)->direita = ant->esquerda;
13.          }
14.          (ant->esquerda)->pai = pai(ant);
15.       }
16.       else   {
17.             if(ehFilhoEsquerda(ant)==1)   {
18.                pai(ant)->esquerda = NULL;
19.             }
20.             else    {
21.                pai(ant)->direita = NULL;
22.             }
23.       }
24.       return ant;
25. }
```

Se o antecessor for filho esquerdo do seu pai, então o campo filho da esquerda do vô passa a apontar para o neto.

Ao encontrar o antecessor, verificar se ele tem filho esquerdo. Se ele tiver.

Antecessor



Árvore Binária de Busca

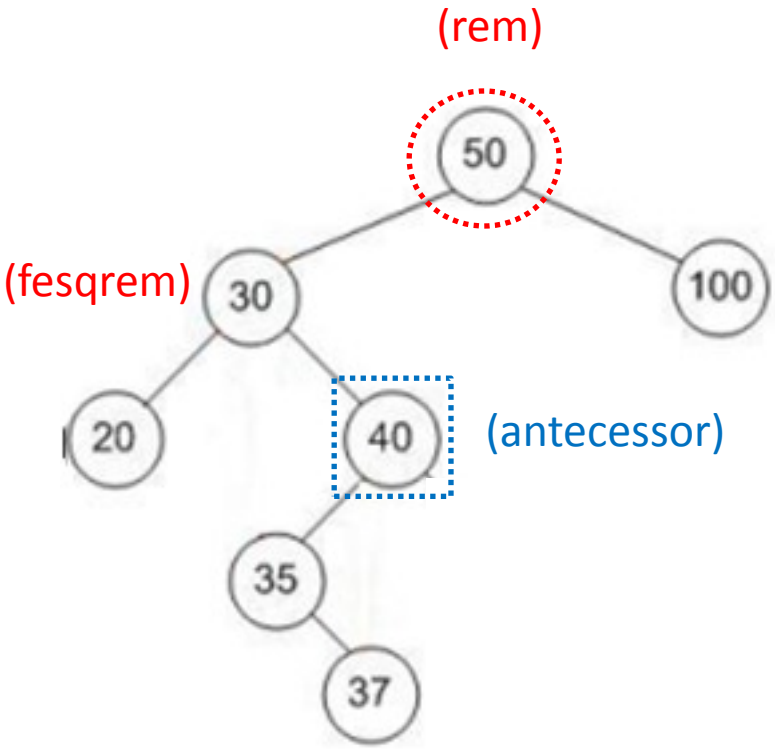
(e REMOVE)

- Busca o Antecessor do nó.

```
1.  struct tNo * antecessor(struct tNo *fesqrem)   {
2.        struct tNo *ant, *pt = fesqrem;
3.        while(pt != NULL)   {
4.              ant = pt;
5.              pt = pt->direita;
6.        }
7.        if(filhoEsquerda(ant) != NULL)   {
8.           if(ehFilhoEsquerda(ant)==1)      {
9.                 pai(ant)->esquerda = ant->esquerda;
10.          }
11.          else    {
12.                pai(ant)->direita = ant->esquerda;
13.          }
14.          (ant->esquerda)->pai = pai(ant);
15.       }
16.       else   {
17.             if(ehFilhoEsquerda(ant)==1)   {
18.                pai(ant)->esquerda = NULL;
19.             }
20.             else     {
21.                pai(ant)->direita = NULL;
22.             }
23.       }
24.       return ant;
25. }
```

Ao encontrar o antecessor, verificar se ele tem filho esquerdo. Se ele tiver.

Antecessor



Árvore Binária de Busca

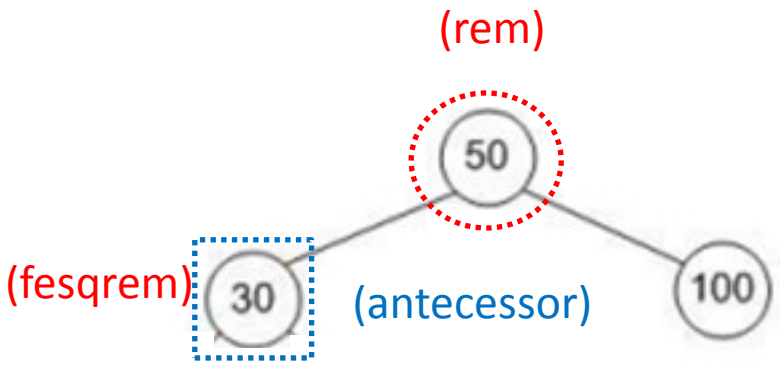
(e REMOVE)

- Busca o Antecessor do nó.

```
1.  struct tNo * antecessor(struct tNo *fesqrem)   {
2.        struct tNo *ant, *pt = fesqrem;
3.        while(pt != NULL)   {
4.              ant = pt;
5.              pt = pt->direita;
6.        }
7.        if(filhoEsquerda(ant) != NULL)   {
8.            if(ehFilhoEsquerda(ant)==1)     {
9.                  pai(ant)->esquerda = ant->esquerda;
10.           }
11.           else    {
12.                 pai(ant)->direita = ant->esquerda;
13.           }
14.           (ant->esquerda)->pai = pai(ant);
15.       }
16.       else   {
17.           if(ehFilhoEsquerda(ant)==1)   {
18.               pai(ant)->esquerda = NULL;
19.           }
20.           else    {
21.               pai(ant)->direita = NULL;
22.           }
23.       }
24.       return ant;
25. }
```

Ao encontrar o antecessor, verificar se ele tem filho esquerdo. Se ele não tiver, então ele é uma folha.

Antecessor



Árvore Binária de Busca

(e REMOVE)

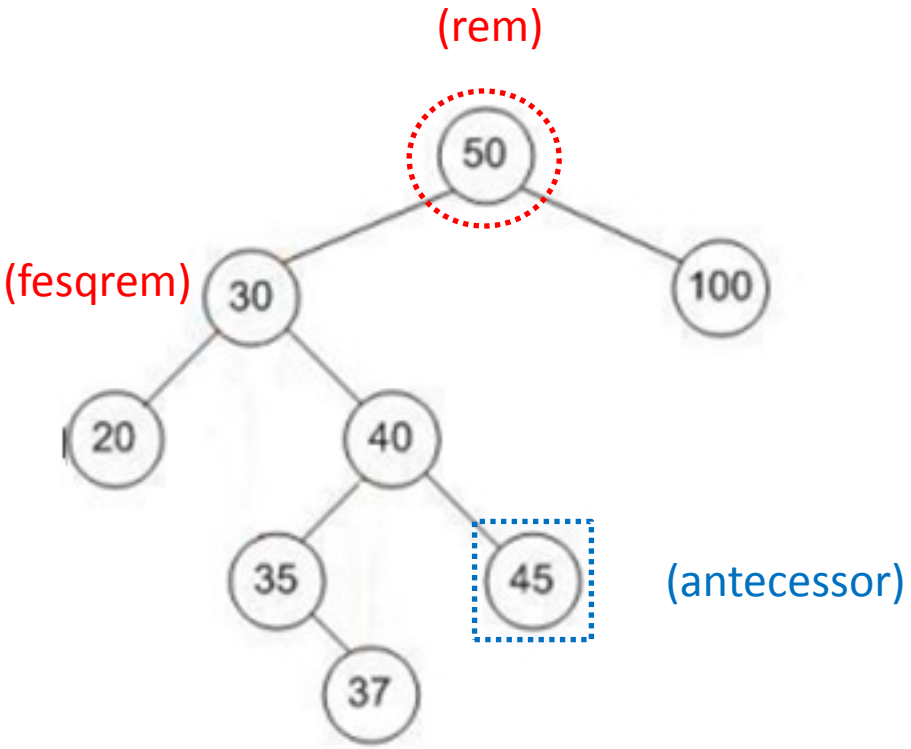
- Busca o Antecessor do nó.

```
1.  struct tNo * antecessor(struct tNo *fesqrem)   {
2.        struct tNo *ant, *pt = fesqrem;
3.        while(pt != NULL)   {
4.              ant = pt;
5.              pt = pt->direita;
6.        }
7.        if(filhoEsquerda(ant) != NULL)   {
8.            if(ehFilhoEsquerda(ant)==1)     {
9.                  pai(ant)->esquerda = ant->esquerda;
10.           }
11.           else    {
12.                 pai(ant)->direita = ant->esquerda;
13.           }
14.           (ant->esquerda)->pai = pai(ant);
15.       }
```

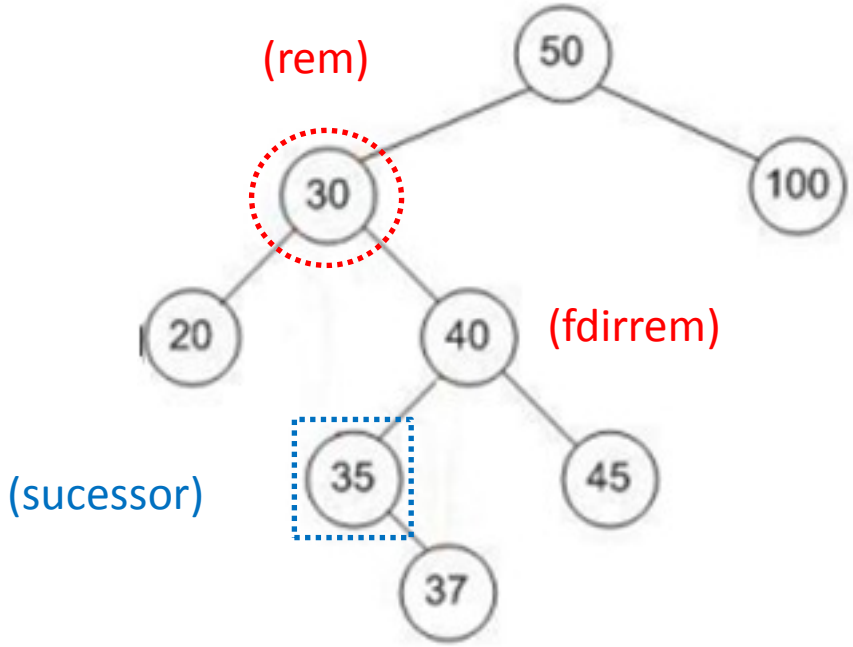
Ao encontrar o antecessor, verificar se ele tem filho esquerdo. Se ele não tiver, então ele é uma folha.

```
16.       else   {
17.           if(ehFilhoEsquerda(ant)==1)   {
18.               pai(ant)->esquerda = NULL;
19.           }
20.           else    {
21.               pai(ant)->direita = NULL;
22.           }
23.       }
24.       return ant;
25.   }
```

Antecessor



Sucessor



Árvore Binária de Busca

(e REMOVE)

- Busca o Sucessor do nó.

```
1.  struct tNo * sucessor(struct tNo *fdirrem)    {
2.      struct tNo *suc, *pt = fdirrem;
3.      while(pt != NULL)   {
4.          suc = pt;
5.          pt = pt->esquerda;
6.      }
7.      if(filhoDireita(suc) != NULL)    {
8.          if(ehFilhoDireita(suc)==1)      {
9.              pai(suc)->direita = suc->direita;
10.         }
11.         else   {
12.             pai(suc)->esquerda = suc->direita;
13.         }
14.         (suc->direita)->pai = pai(suc);
15.     }
16.     else   {
17.         if(ehFilhoDireita(suc)==1)   {
18.             pai(suc)->direita = NULL;
19.         }
20.         else   {
21.             pai(suc)->esquerda = NULL;
22.         }
23.     }
24.     return suc;
25. }
```

Descendo à esquerda na subárvore direita, até localizar o sucessor.

Árvore Binária de Busca

(e REMOVE)

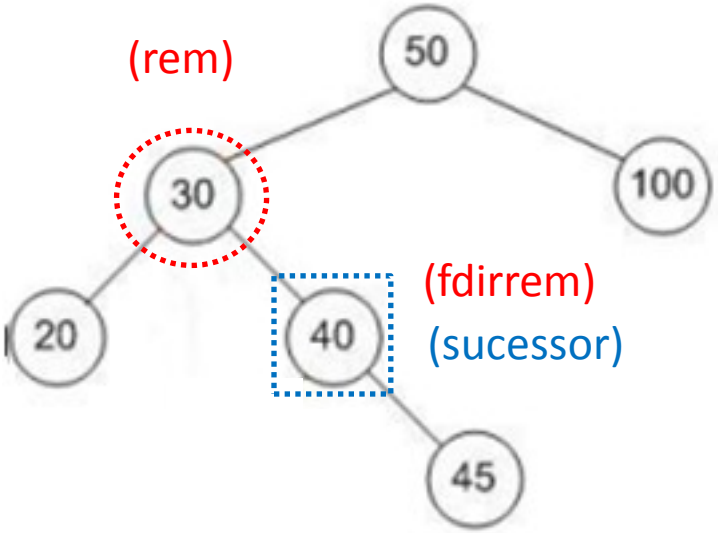
- Busca o Sucessor do nó.

```
1.  struct tNo * sucessor(struct tNo *fdirrem)    {
2.      struct tNo *suc, *pt = fdirrem;
3.      while(pt != NULL)   {
4.          suc = pt;
5.          pt = pt->esquerda;
6.      }
7.      if(filhoDireita(suc) != NULL)    {
8.          if(ehFilhoDireita(suc)==1)       {
9.              pai(suc)->direita = suc->direita;
10.         }
11.         else   {
12.             pai(suc)->esquerda = suc->direita;
13.         }
14.         (suc->direita)->pai = pai(suc);
15.     }
16.     else   {
17.         if(ehFilhoDireita(suc)==1)   {
18.             pai(suc)->direita = NULL;
19.         }
20.         else   {
21.             pai(suc)->esquerda = NULL;
22.         }
23.     }
24.     return suc;
25. }
```

Se o sucessor for filho Direito do seu pai, então o campo filho da direita do vô passa a apontar para o neto.

Ao encontrar o sucessor, verificar se ele tem filho direito. Se ele tiver.

Sucessor



Árvore Binária de Busca

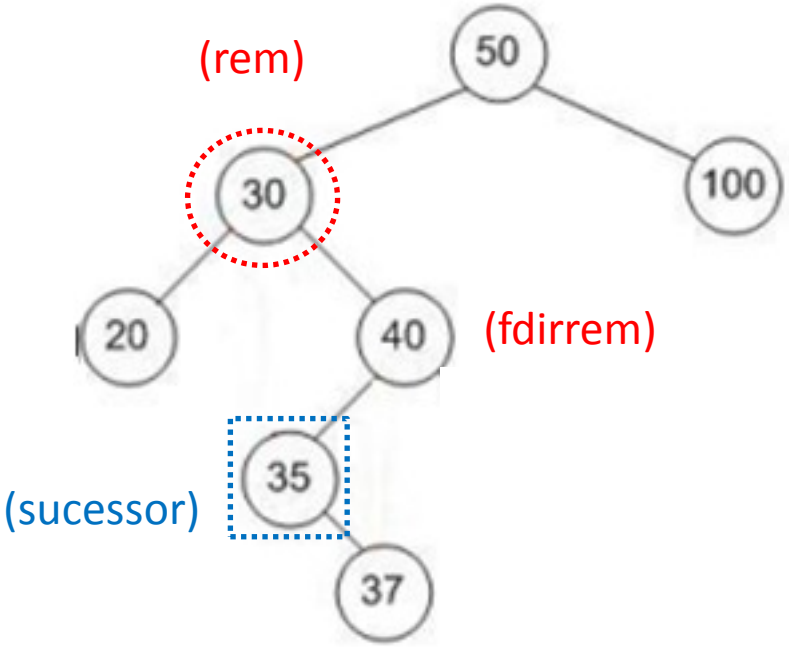
(e REMOVE)

- Busca o Sucessor do nó.

```
1.  struct tNo * sucessor(struct tNo *fdirrem)    {
2.      struct tNo *suc, *pt = fdirrem;
3.      while(pt != NULL)   {
4.          suc = pt;
5.          pt = pt->esquerda;
6.      }
7.      if(filhoDireita(suc) != NULL)    {
8.          if(ehFilhoDireita(suc)==1)      {
9.              pai(suc)->direita = suc->direita;
10.         }
11.         else   {
12.             pai(suc)->esquerda = suc->direita;
13.         }
14.         (suc->direita)->pai = pai(suc);
15.     }
16.     else   {
17.         if(ehFilhoDireita(suc)==1)   {
18.             pai(suc)->direita = NULL;
19.         }
20.         else   {
21.             pai(suc)->esquerda = NULL;
22.         }
23.     }
24.     return suc;
25. }
```

Ao encontrar o sucessor,
verificar se ele tem filho direito.
Se ele tiver.

Sucessor



Árvore Binária de Busca

(e REMOVE)

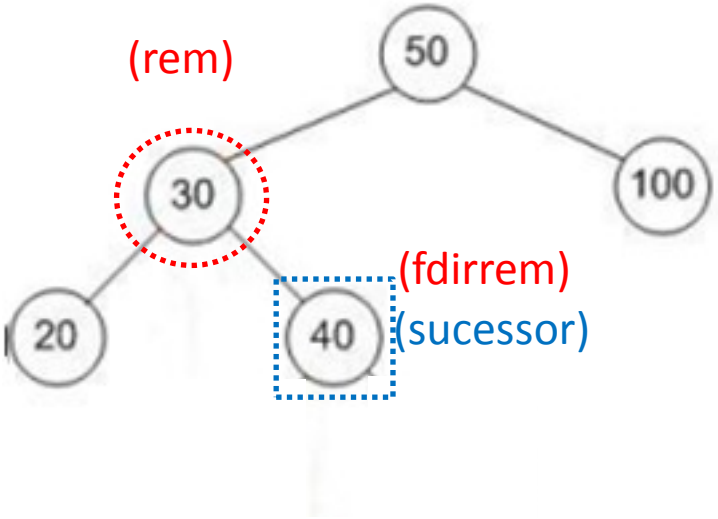
- Busca o Sucessor do nó.

```
1.  struct tNo * sucessor(struct tNo *fdirrem)    {
2.      struct tNo *suc, *pt = fdirrem;
3.      while(pt != NULL)   {
4.          suc = pt;
5.          pt = pt->esquerda;
6.      }
7.      if(filhoDireita(suc) != NULL)    {
8.          if(ehFilhoDireita(suc)==1)      {
9.              pai(suc)->direita = suc->direita;
10.         }
11.         else   {
12.             pai(suc)->esquerda = suc->direita;
13.         }
14.         (suc->direita)->pai = pai(suc);
15.     }
```

Ao encontrar o sucessor, verificar se ele tem filho direito. Se ele não tiver, então ele é uma folha.

```
16.     else   {
17.         if(ehFilhoDireita(suc)==1)   {
18.             pai(suc)->direita = NULL;
19.         }
20.         else   {
21.             pai(suc)->esquerda = NULL;
22.         }
23.     }
24.     return suc;
25. }
```

Sucessor



Árvore Binária de Busca

(e REMOVE)

- Busca o Sucessor do nó.

```
1.  struct tNo * sucessor(struct tNo *fdirrem)    {
2.      struct tNo *suc, *pt = fdirrem;
3.      while(pt != NULL)   {
4.          suc = pt;
5.          pt = pt->esquerda;
6.      }
7.      if(filhoDireita(suc) != NULL)    {
8.          if(ehFilhoDireita(suc)==1)       {
9.              pai(suc)->direita = suc->direita;
10.         }
11.         else   {
12.             pai(suc)->esquerda = suc->direita;
13.         }
14.         (suc->direita)->pai = pai(suc);
15.     }
```

Ao encontrar o sucessor, verificar se ele tem filho direito. Se ele não tiver, então ele é uma folha.

```
16.     else   {
17.         if(ehFilhoDireita(suc)==1)   {
18.             pai(suc)->direita = NULL;
19.         }
20.         else   {
21.             pai(suc)->esquerda = NULL;
22.         }
23.     }
24.     return suc;
25. }
```

Sucessor