Árvores

Árvore Binária de Busca

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

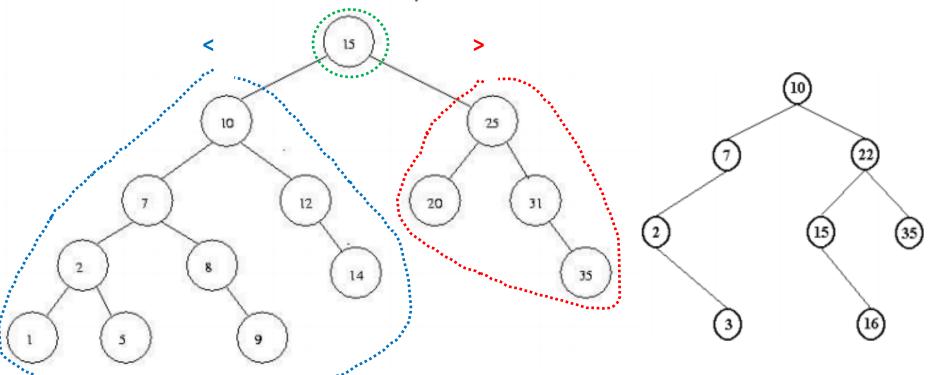
Árvores

- 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)	$O(\log_b n)$	

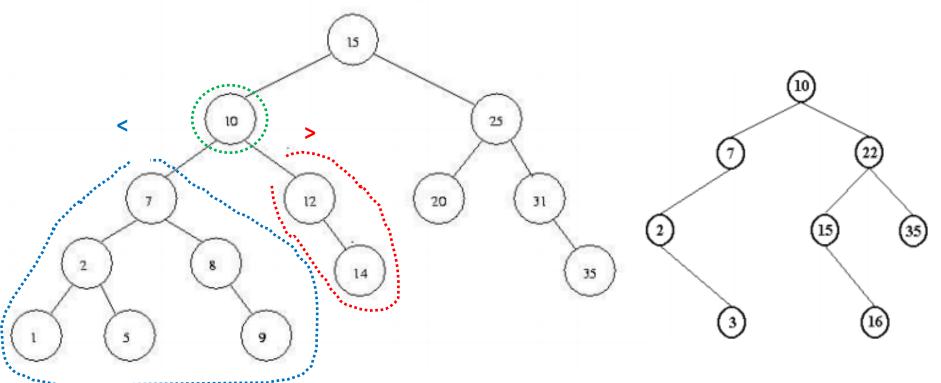
Definição:

- Árvore Binária de Busca



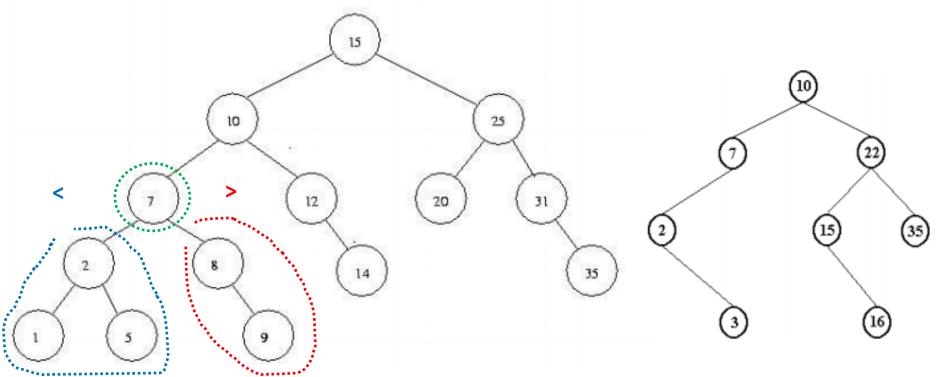
Definição:

– Árvore Binária de Busca



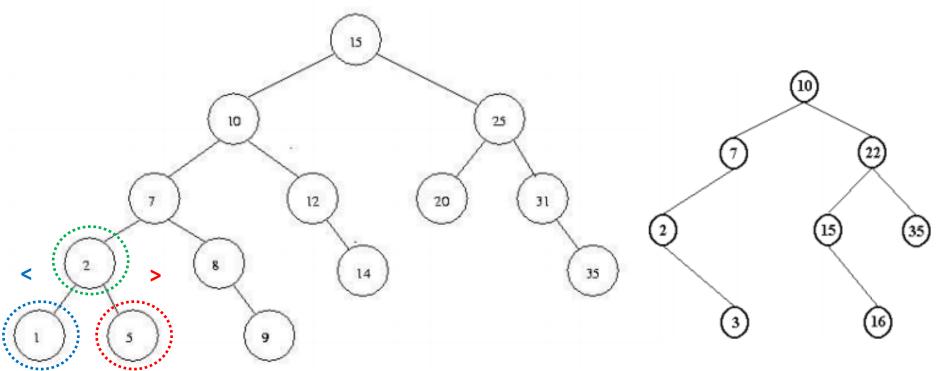
Definição:

– Árvore Binária de Busca



Definição:

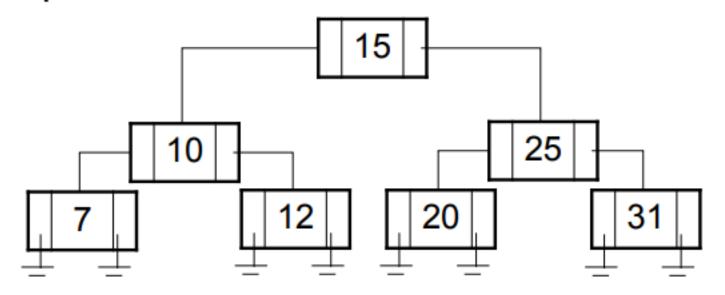
- Árvore Binária de Busca



Exemplo de Árvore Binária de Busca Estática:

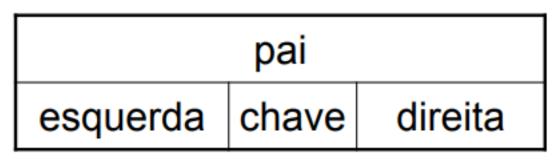
0	1	2	3	4	5	6
15	10	25	7	12	20	31

Exemplo de Árvore Binária de Busca Dinâmica:



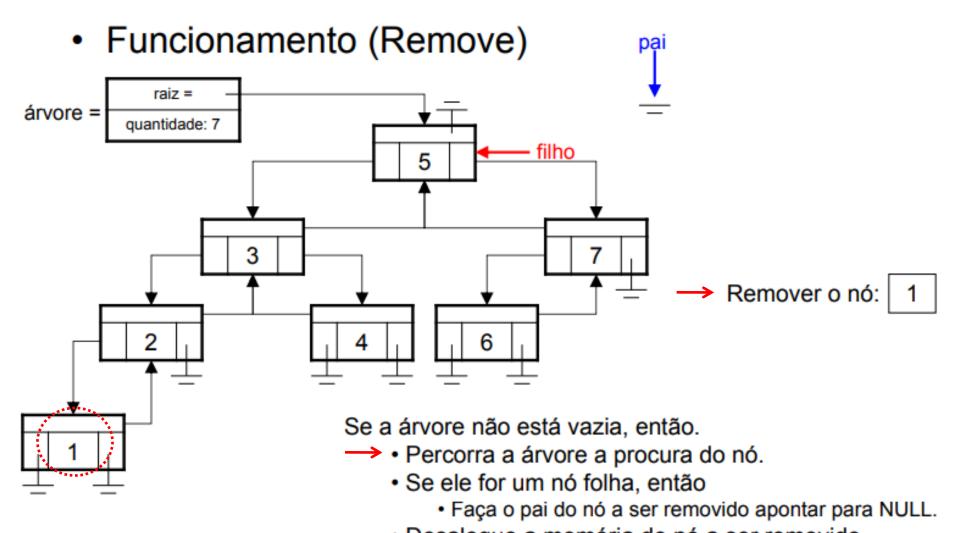
- Em Ciência da Computação uma árvore binária de busca dinâmica é uma estrutura de dados que:
 - Consiste de uma seqüencia de registros
 - Cada registro tem quatro campos:
 - uma campo para a chave.
 - um ponteiro para o filho da esquerda (subárvore da esquerda)
 - um ponteiro para o filho da direita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

Ponteiro para raiz Quantidade Árvore



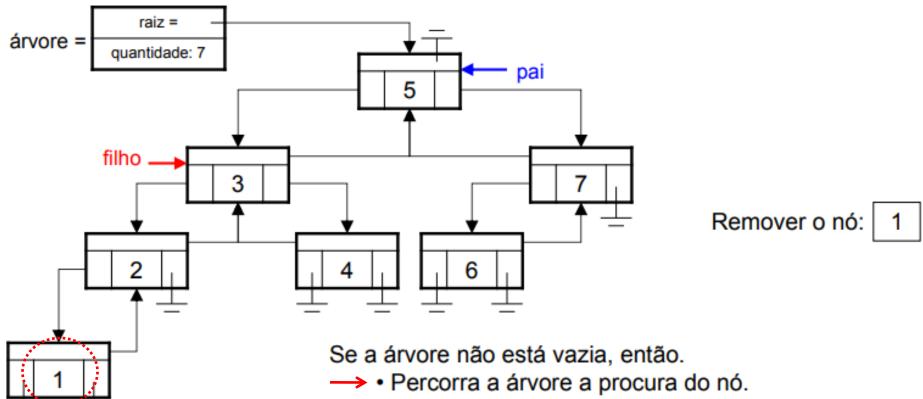
Nó da árvore

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



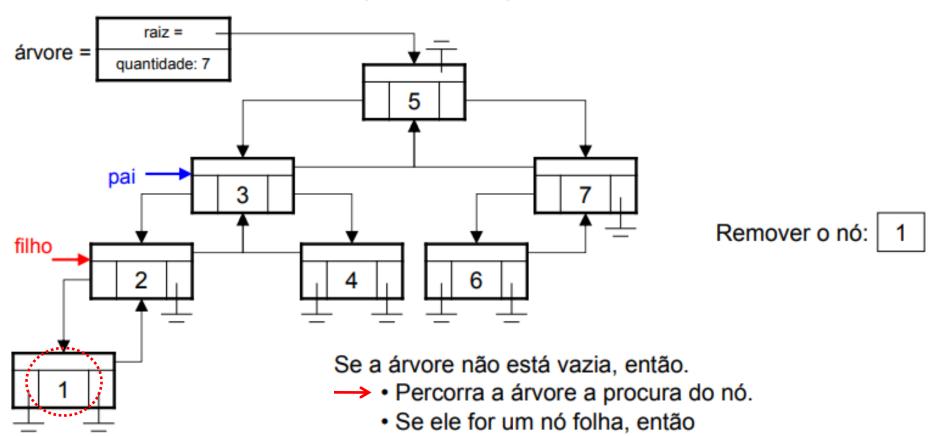
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Funcionamento (Remove)



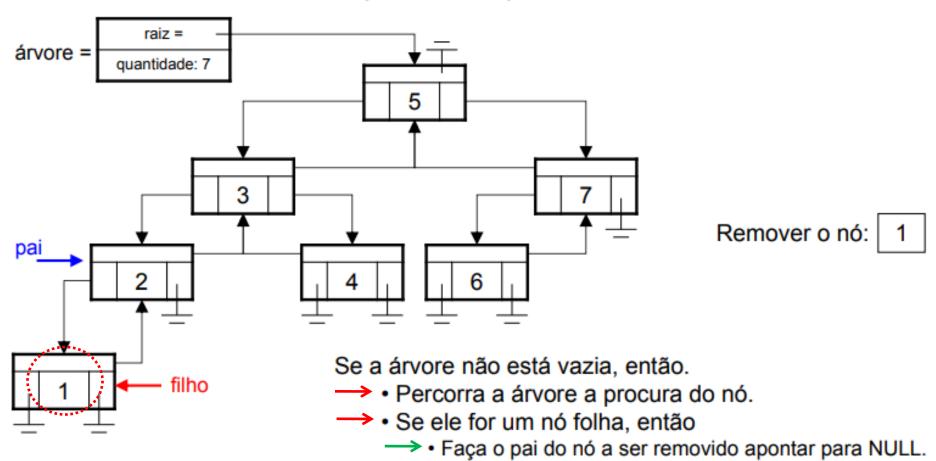
- Se ele for um nó folha, então
 - · Faça o pai do nó a ser removido apontar para NULL.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Funcionamento (Remove)



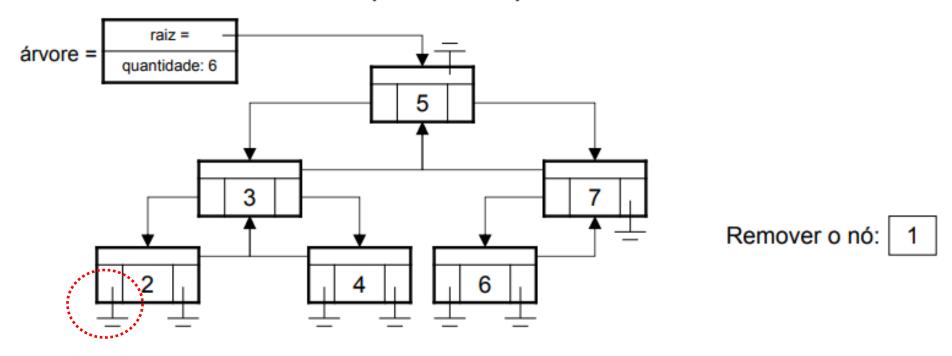
- Faça o pai do nó a ser removido apontar para NULL.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Funcionamento (Remove)

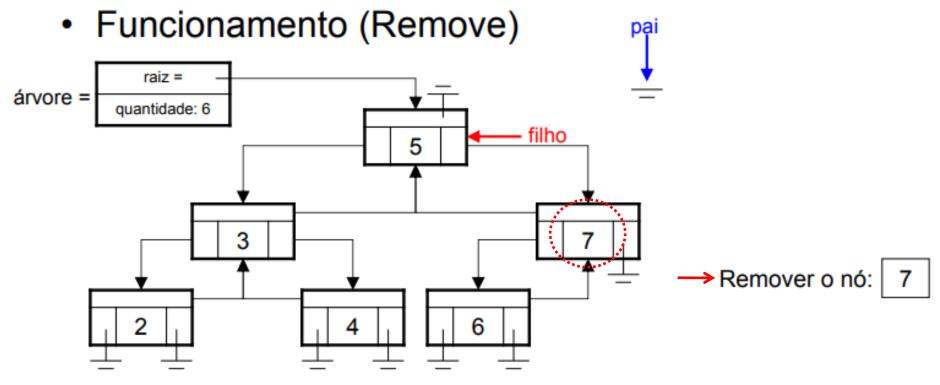


- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.

Funcionamento (Remove)

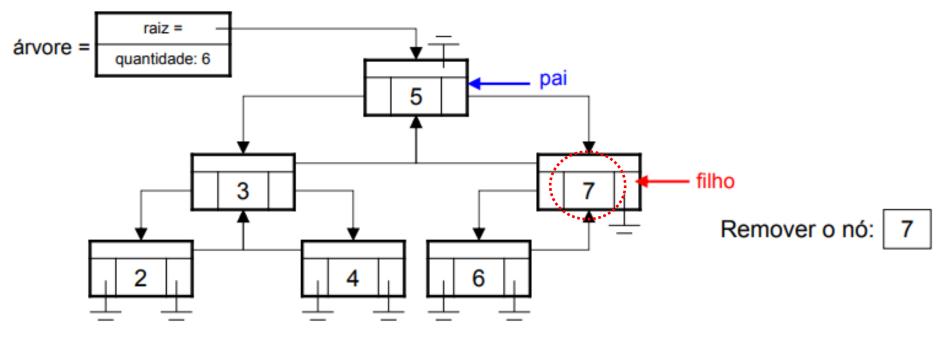


- Percorra a árvore a procura do nó.
- Se ele for um nó folha, então
 - Faça o pai do nó a ser removido apontar para NULL.
- Desaloque a memória do nó a ser removido
- Decremente o campo quantidade da árvore.



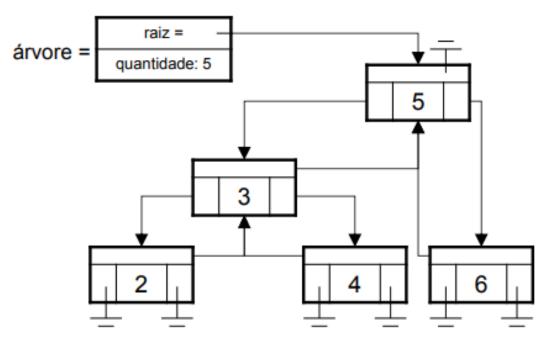
- Percorra a árvore a procura do nó.
 - Se ele for um nó galho com apenas um filho, então
 - Faça o pai do nó a ser removido apontar para o neto.
 - · Faça o neto apontar para o avô como pai.
 - Desaloque a memória do nó a ser removido
 - Decremente o campo quantidade da árvore.

Funcionamento (Remove)



- 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.

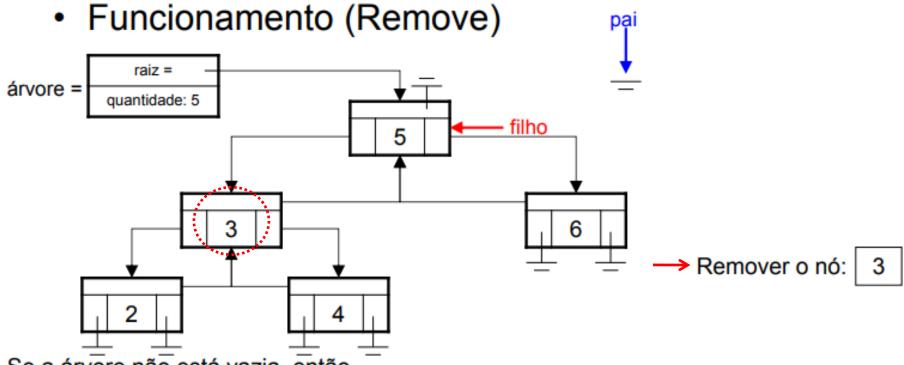
Funcionamento (Remove)



Remover o nó:

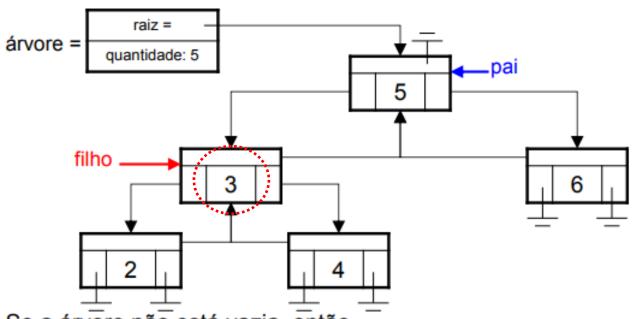
| 7

- 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.



- 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.

Funcionamento (Remove)

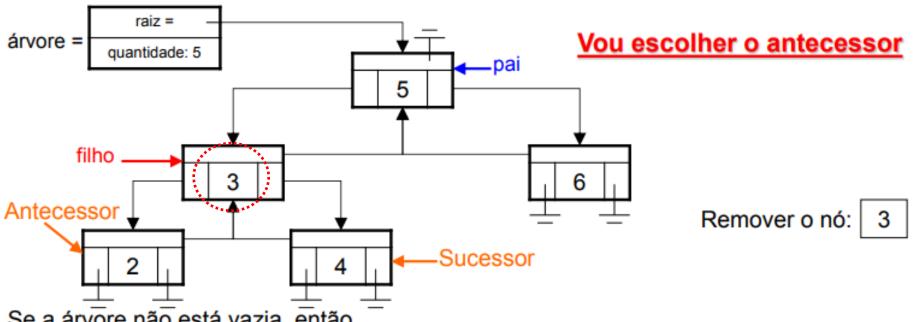


Remover o nó:

3

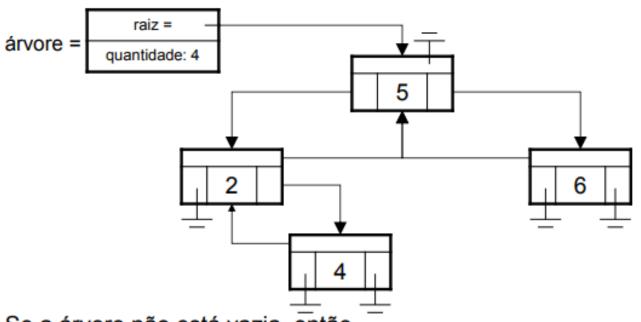
- Percorra a árvore a procura do nó.
- → Se ele for um nó galho com dois filho, então
 - Busque o antecessor ou o sucessor do nó a ser removido.
 - Se o sucessor ou antecessor do nó a ser removido tiver filhos, chame o procedimento recursivamente
 - Caso contrário, substitua o nó a ser removido pelo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

Funcionamento (Remove)



- 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.

Funcionamento (Remove)

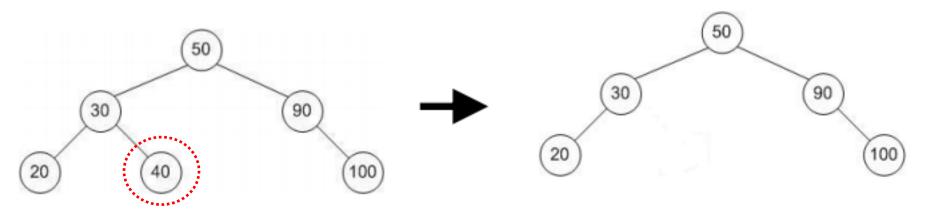


Remover o nó:

3

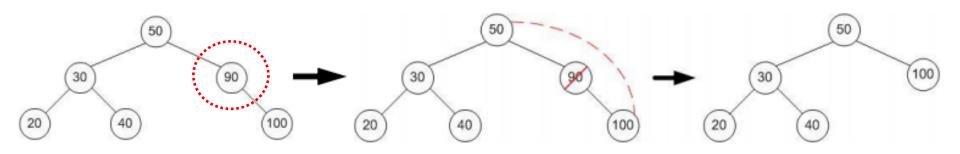
- Percorra a árvore a procura do nó.
- → Se ele for um nó galho com dois filho, então
 - Busque o antecessor ou o sucessor do nó a ser removido.
 - > Se o sucessor ou antecessor do nó a ser removido tiver filhos, chame o procedimento recursivamente
 - -> Caso contrário, substitua o nó a ser removido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

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



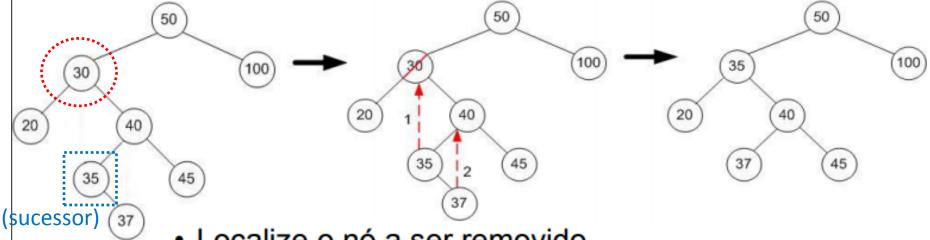
- Localize o nó a ser removido.
- Se for um nó folha então
 - Se não for a raiz
 - » Faça o pai do nó a ser removido apontar para NULL.
 - Se for a raiz
 - » Faça o campo raiz da árvore apontar para NULL

- 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.

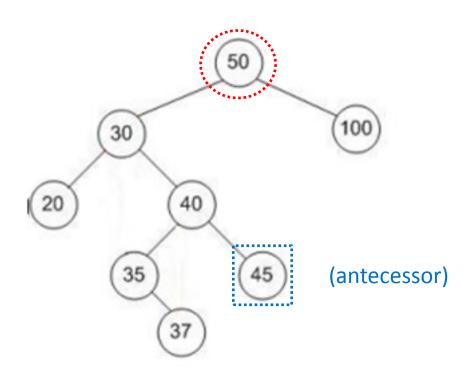
- Remoção (Lembretes)
 - Caso 3: Removendo um nó com 2 filhos.



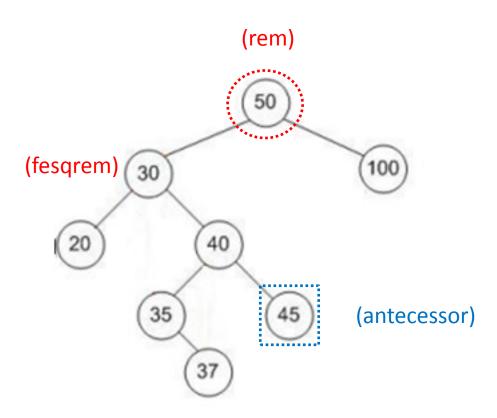
- Localize o nó a ser removido.
- Se o nó tiver 2 filhos
 - Busque o antecessor ou o sucessor.

Substitua o nó a ser removido pelo antecessor ou sucessor.

Antecessor



Antecessor



(e REMOVE)

Busca o Antecessor do nó.

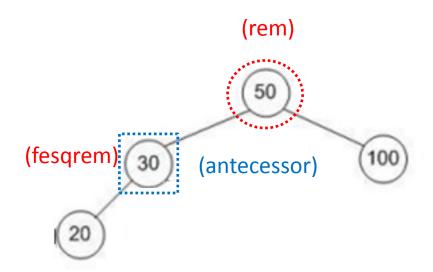
```
struct tNo * antecessor(struct tNo *fesqrem) {
         struct tNo *ant, *pt = fesqrem;
2.
3.
         while(pt != NULL)
                                  Descendo à direita na
4.
             ant = pt;
                                                            16.
                             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

(e REMOVE)

Busca o Antecessor do nó.

```
struct tNo * antecessor(struct tNo *fesgrem) {
2.
         struct tNo *ant, *pt = fesgrem;
                                  Se o antecessor for filho
3.
         while(pt != NULL) {
                                  esquerdo do seu pai,
                                                           16.
             ant = pt;
                                                                    else
                                  então o campo filho da
5.
             pt = pt->direita;
                                                            17.
                                                                         if(ehFilhoEsquerda(ant)==1) {
                                  esquerda do vô passa a
6.
                                  apontar para o neto.
                                                                           pai(ant)->esquerda = NULL;
                                                            18.
7.
         if(filhoEsquerda(ant) != NULL)
                                                            19.
            if(ehFilhoEsquerda(ant)==1)
8.
                                                           20.
                                                                         else
9.
                pai(ant)->esquerda = ant->esquerda;
                                                                             pai(ant)->direita = NULL;
                                                           21.
10.
                                                           22.
11.
                                                           23.
            else
12.
                pai(ant)->direita = ant->esquerda;
                                                           24.
                                                                     return ant;
13.
                                                           25. }
14.
            (ant->esquerda)->pai = pai(ant);
                                                         Ao encontrar o antecessor,
15.
                                                         verificar se ele tem filho
                                                         esquerdo. Se ele tiver.
```

Antecessor



(e REMOVE)

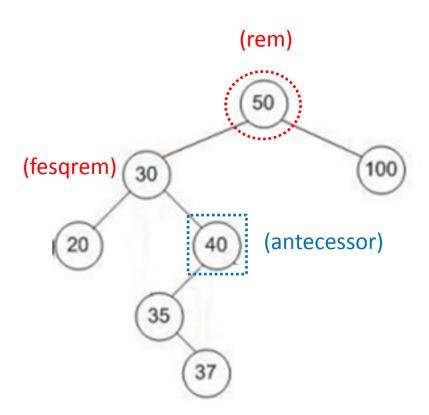
Busca o Antecessor do nó.

```
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.
            (ant->esquerda)->pai = pai(ant);
14.
15.
```

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

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

Antecessor

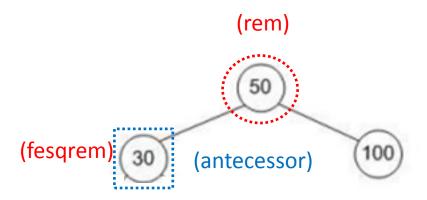


(e REMOVE)

Busca o Antecessor do nó.

```
Ao encontrar o antecessor.
   struct tNo * antecessor(struct tNo *fesgrem) {
                                                                          verificar se ele tem filho
2.
         struct tNo *ant, *pt = fesqrem;
                                                                          esquerdo. Se ele não tiver,
3.
         while(pt != NULL) {
                                                                          então ele é uma folha.
4.
             ant = pt;
                                                            16.
                                                                    else
5.
             pt = pt->direita;
                                                            17.
                                                                         if(ehFilhoEsquerda(ant)==1)
6.
                                                                            pai(ant)->esquerda = NULL;
                                                            18.
7.
         if(filhoEsquerda(ant) != NULL) {
                                                            19.
            if(ehFilhoEsquerda(ant)==1)
8.
                                                           20
                                                                         else
9.
                pai(ant)->esquerda = ant->esquerda;
                                                           21.
                                                                             pai(ant)->direita = NULL;
10.
                                                           22.
11.
                                                           23.
            else
12.
                pai(ant)->direita = ant->esquerda;
                                                           24.
                                                                     return ant;
13.
                                                           25. }
14.
            (ant->esquerda)->pai = pai(ant);
15.
```

Antecessor

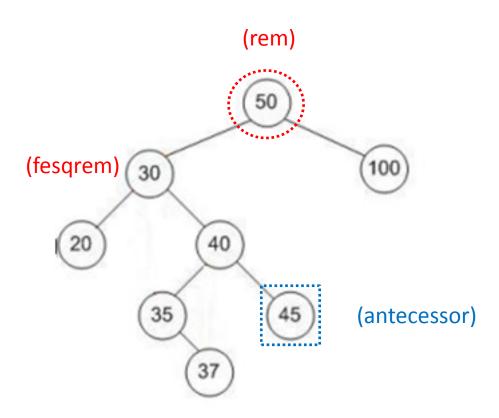


(e REMOVE)

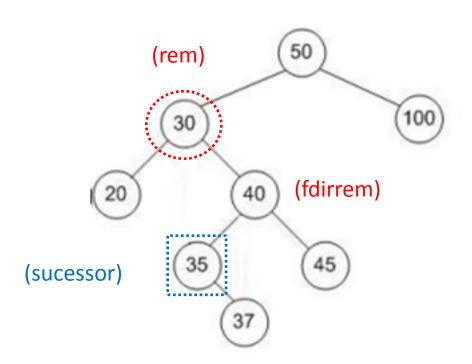
Busca o Antecessor do nó.

```
Ao encontrar o antecessor.
   struct tNo * antecessor(struct tNo *fesqrem) {
                                                                          verificar se ele tem filho
2.
         struct tNo *ant, *pt = fesgrem;
                                                                          esquerdo. Se ele não tiver,
3.
                                                                         então ele é uma folha.
         while(pt != NULL) {
4.
                                                            16.
             ant = pt;
                                                                    else
5.
             pt = pt->direita;
                                                            17.
                                                                         if(ehFilhoEsquerda(ant)==1) {
6.
                                                           18.
                                                                           pai(ant)->esquerda = NULL;
7.
         if(filhoEsquerda(ant) != NULL) {
                                                           19.
            if(ehFilhoEsquerda(ant)==1)
8.
                                                           20.
                                                                         else
9.
                pai(ant)->esquerda = ant->esquerda;
                                                           21.
                                                                             pai(ant)->direita = NULL;
10.
                                                           22.
11.
                                                           23.
            else
12.
                pai(ant)->direita = ant->esquerda;
                                                           24.
                                                                     return ant;
13.
                                                           25. }
            (ant->esquerda)->pai = pai(ant);
14.
15.
```

Antecessor



Sucessor



(e REMOVE)

Busca o Sucessor do nó.

```
    struct tNo * sucessor(struct tNo *fdirrem)

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

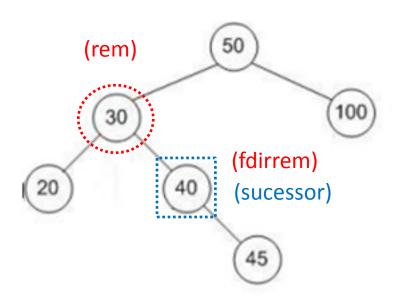
(e REMOVE)

Busca o Sucessor do nó.

```
    struct tNo * sucessor(struct tNo *fdirrem)

2.
      struct tNo *suc, *pt = fdirrem;
                                Se o sucessor for filho
3.
      while(pt != NULL) {
                                Direito do seu pai, então
4.
          suc = pt;
                                o campo filho da direita 16.
                                                               else
5.
          pt = pt->esquerda;
                                do vô passa a apontar
                                                         17.
                                                                    if(ehFilhoDireita(suc)==1) {
6.
                                para o neto.
                                                                       pai(suc)->direita = NULL;
                                                         18.
      if(filhoDireita(suc) != NULL)
                                                         19.
          if(ehFilhoDireita(suc)==1)
8.
                                                        20.
                                                                    else
9.
              pai(suc)->direita = suc->direita;
                                                        21.
                                                                         pai(suc)->esquerda = NULL;
10.
                                                        22.
11.
          else
                                                        23.
12.
             pai(suc)->esquerda = suc->direita;
                                                        24.
                                                                return suc;
13.
                                                        25. }
14.
           (suc->direita)->pai = pai(suc);
                                                     Ao encontrar o sucessor.
15.
                                                      verificar se ele tem filho direito.
                                                      Se ele tiver.
```

Sucessor



(e REMOVE)

Busca o Sucessor do nó.

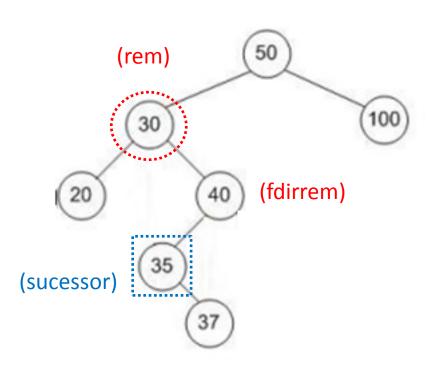
```
    struct tNo * sucessor(struct tNo *fdirrem)

      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.
           (suc->direita)->pai = pai(suc);
14.
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



(e REMOVE)

Busca o Sucessor do nó.

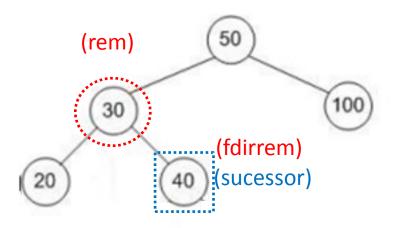
```
    struct tNo * sucessor(struct tNo *fdirrem)

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

Sucessor



(e REMOVE)

Busca o Sucessor do nó.

```
    struct tNo * sucessor(struct tNo *fdirrem)

                                                                    Ao encontrar o sucessor.
2.
                                                                    verificar se ele tem filho direito.
      struct tNo *suc, *pt = fdirrem;
                                                                    Se ele não tiver, então ele é
3.
      while(pt != NULL) {
                                                                    uma folha.
4.
          suc = pt;
                                                        16.
                                                               else
5.
          pt = pt->esquerda;
                                                        17.
                                                                   if(ehFilhoDireita(suc)==1)
6.
                                                        18.
                                                                       pai(suc)->direita = NULL;
7.
      if(filhoDireita(suc) != NULL)
                                                        19.
8.
          if(ehFilhoDireita(suc)==1)
                                                        20.
                                                                   else
9.
              pai(suc)->direita = suc->direita;
                                                        21.
                                                                         pai(suc)->esquerda = NULL;
10.
                                                        22.
11.
          else
                                                        23.
12.
             pai(suc)->esquerda = suc->direita;
                                                        24.
                                                                return suc;
13.
                                                        25. }
          (suc->direita)->pai = pai(suc);
14.
15.
```

Sucessor

