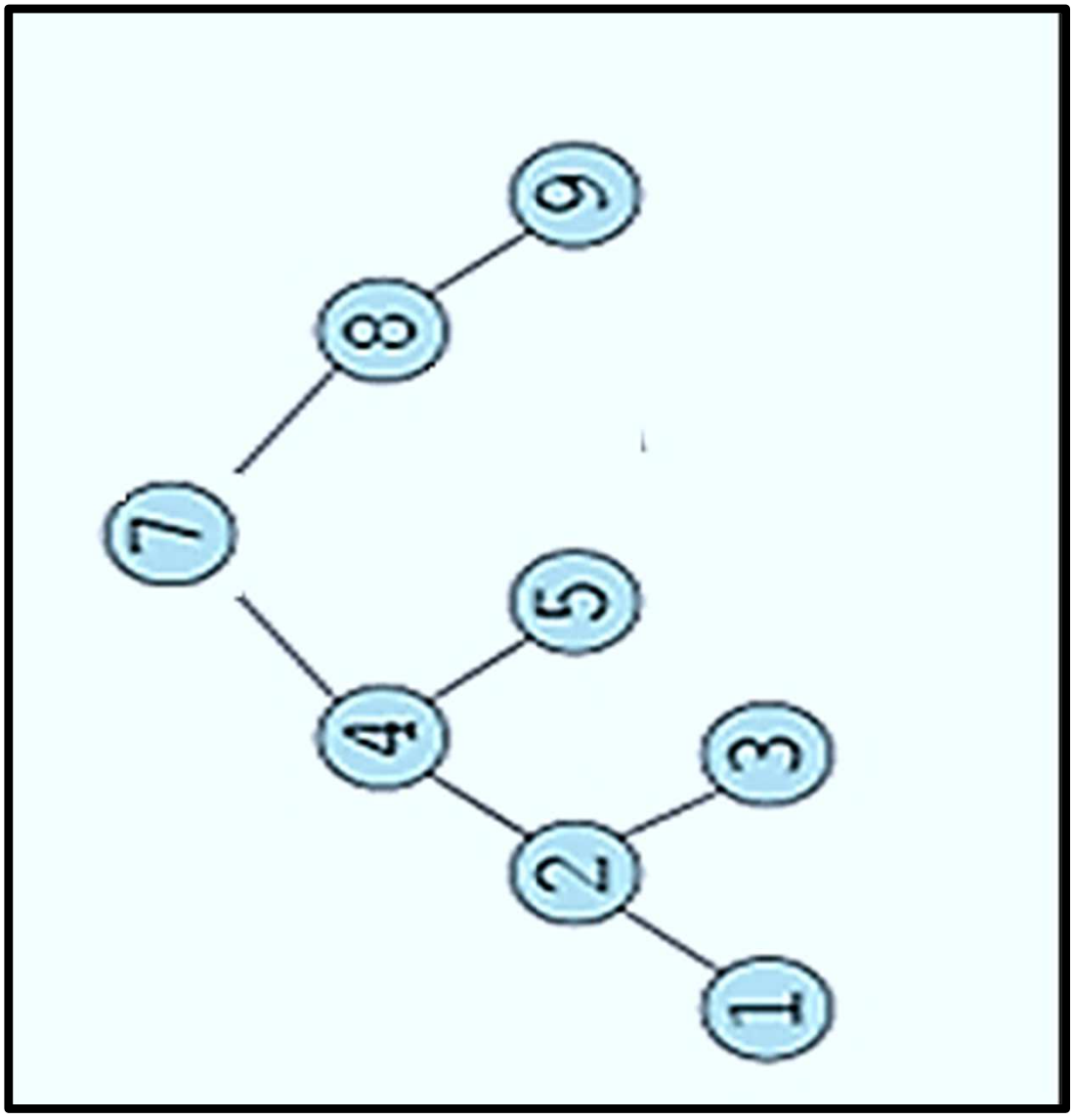

# Estrutura de Dados

## Árvores Binárias de Busca – Remoção de um Nó

**Profa. Dra. Lúcia Guimarães**

- **Esta aula foi baseada na Apostila de Estrutura de Dados – PUC-RIO**

  **Profs. Waldemar Celes e José Lucas Rangel**


6
8
7
5
4
3
2
1


# Arvores Binárias Remoção de um Nó

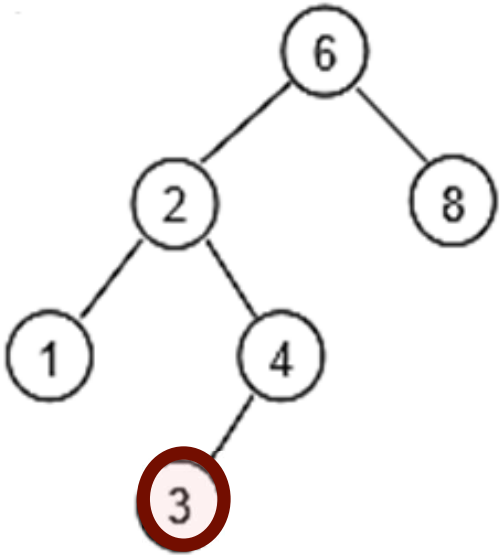
- **Basicamente, existem 3 casos para a remoção de um nó de uma árvore:**

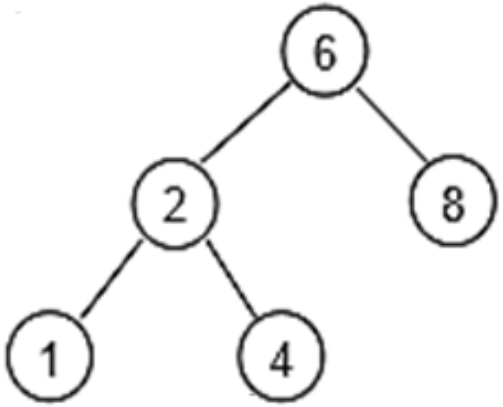
  - **Caso 1:** Remover uma folha
  - **Caso 2:** Remover um nó interno que possui uma única sub-árvore
  - **Caso 3:** Remover um nó interno que possui duas sub-árvores




- **Caso 1: Remover uma folha**

  - **Caso mais simples**
    - Libera a memória alocada para o nó, que passa a ser NULL

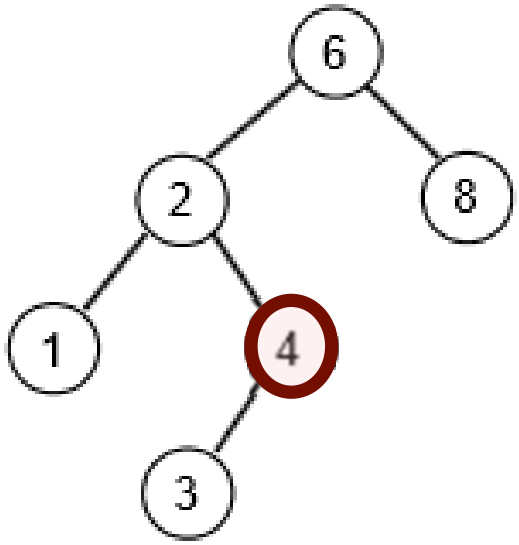


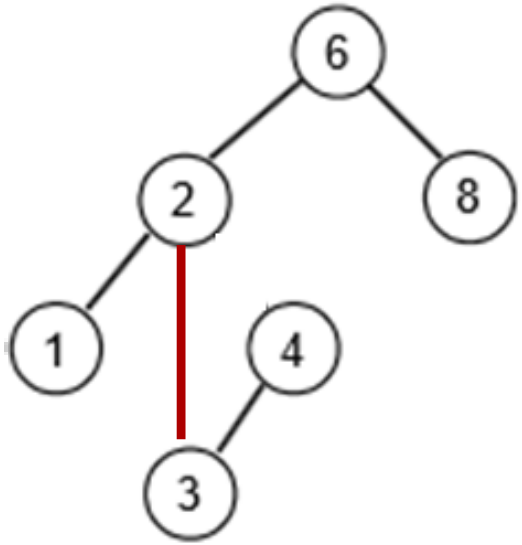



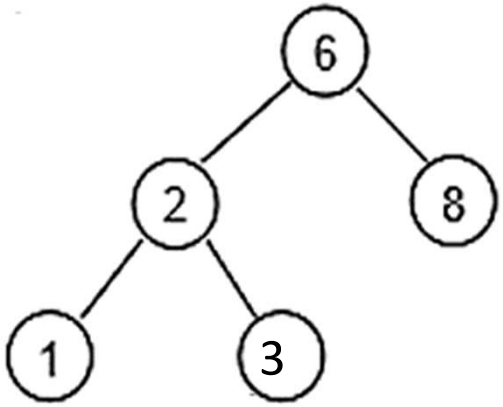




- **Caso 2: Remover um nó interno que possui uma única subárvore**
  - **Consiste em mover a raiz da sub-arvore para ocupar o lugar do nó excluído.**



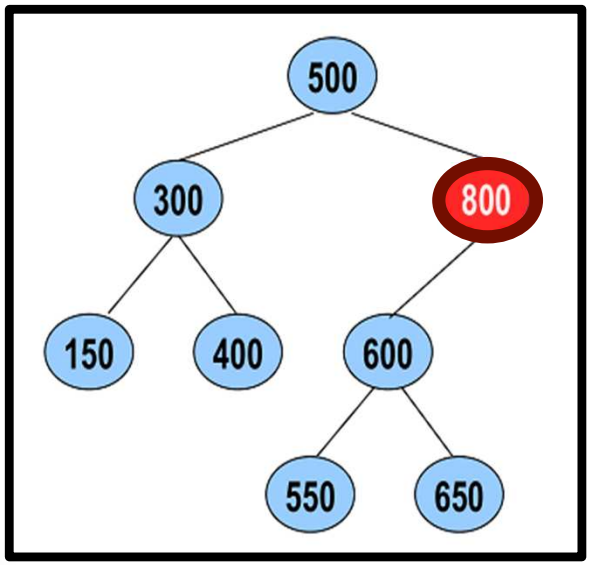



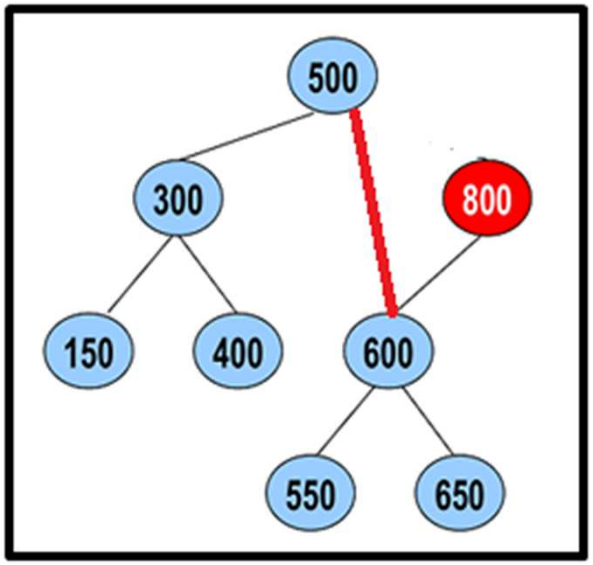



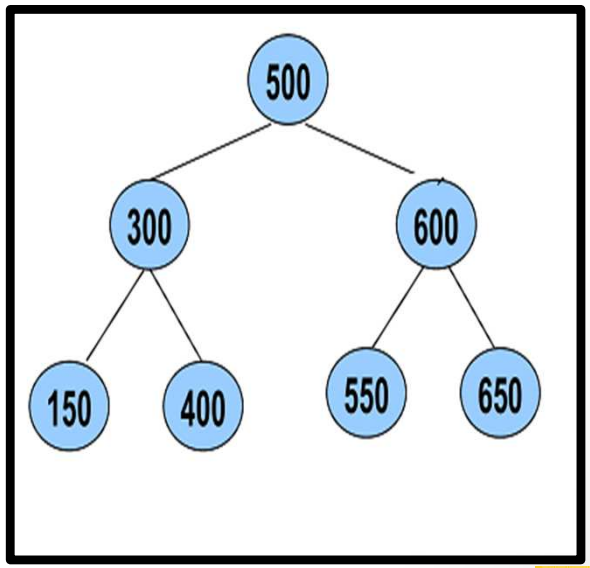




- **Caso 2: Remover um nó interno que possui uma única sub-árvore**
  - **Consiste em mover a raiz da sub-arvore para ocupar o lugar do nó excluído.**





# Arvores Binárias Remoção de um Nó

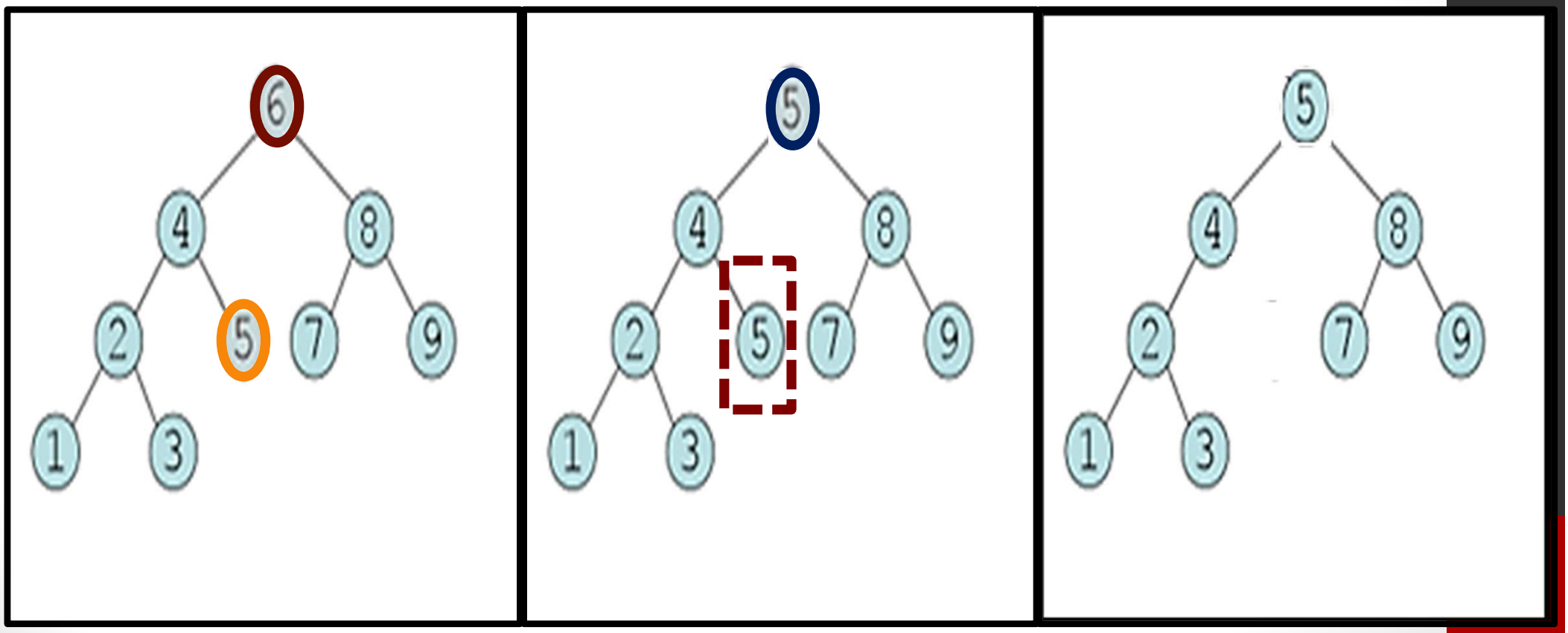
- **Caso 3:** um nó interno que **possui as duas sub-árvores (direita e esquerda)**
  - **Caso mais Complexo – ESTRATÉGIA RECURSIVA**
    - **Trocar** o valor do nó a ser removido **(Mantendo a lei de formação da árvore)**
      - **O maior valor do nó da sub-árvore da esquerda**

      **OU**
      - **O menor valor do nó da sub-árvore de direita**
    - Após a troca o nó, que será uma folha, é removido




- **Caso 3:** Remover um nó que possui duas sub-árvores

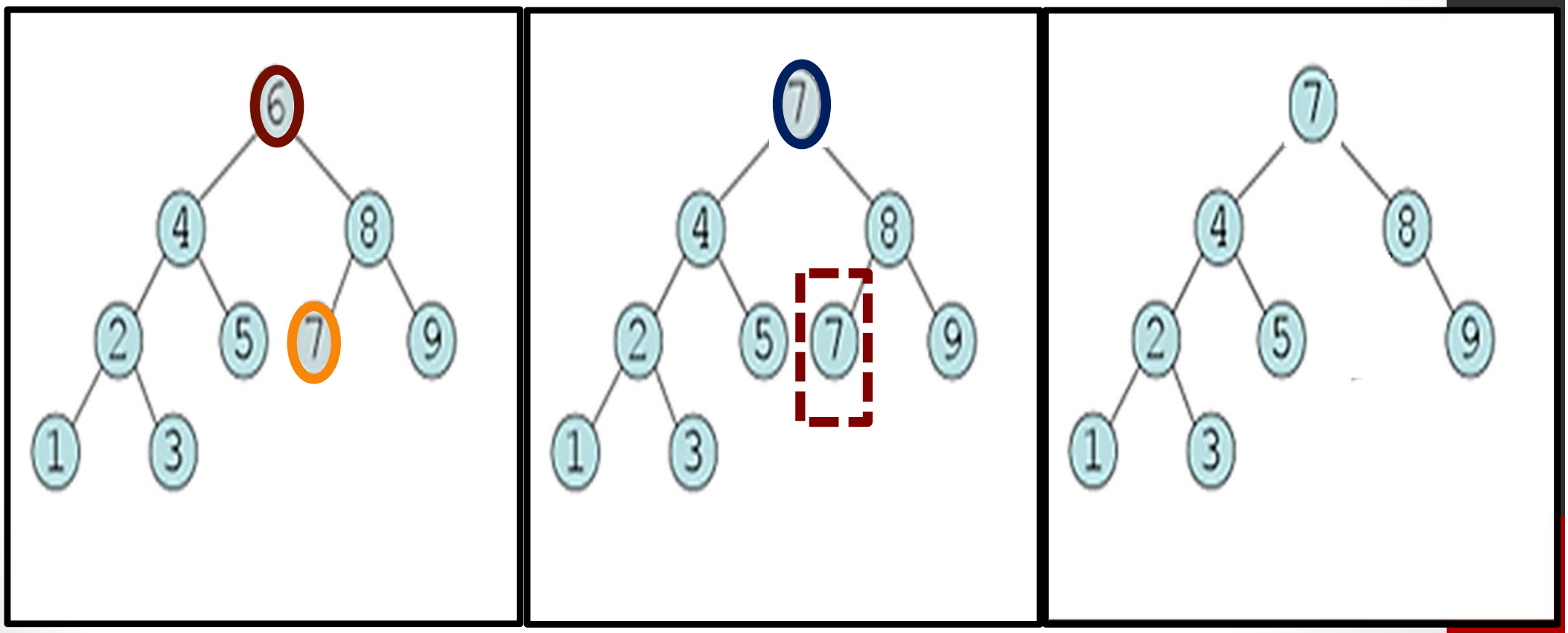
**Vamos efetuar pelo MAIOR VALOR DA SUB-ÁRVORE DA ESQUERDA**




# Arvores Binárias Remoção de um Nó

- **Caso 3:** Remover um nó que possui duas sub-árvores

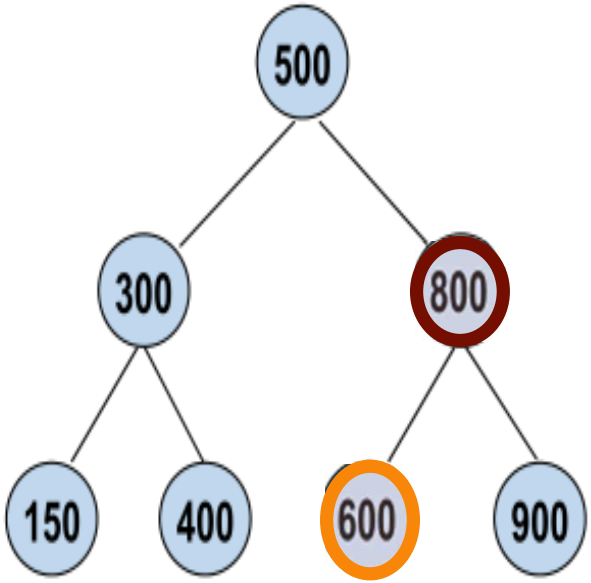
**Vamos efetuar pelo menor VALOR DA SUB-ÁRVORE DA DIREITA**

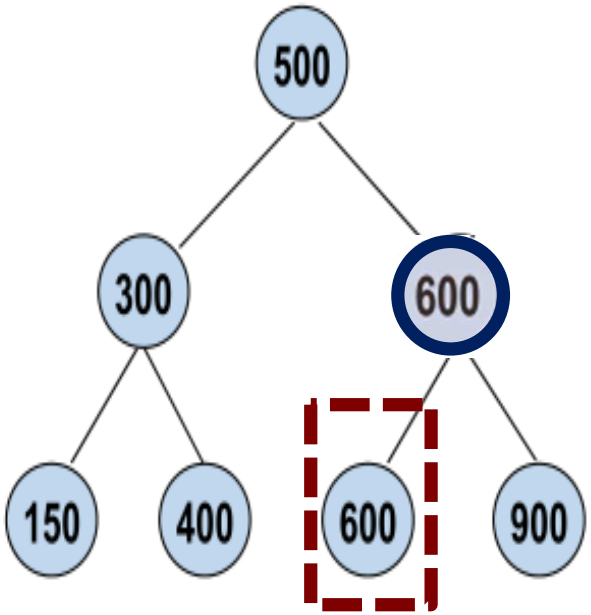


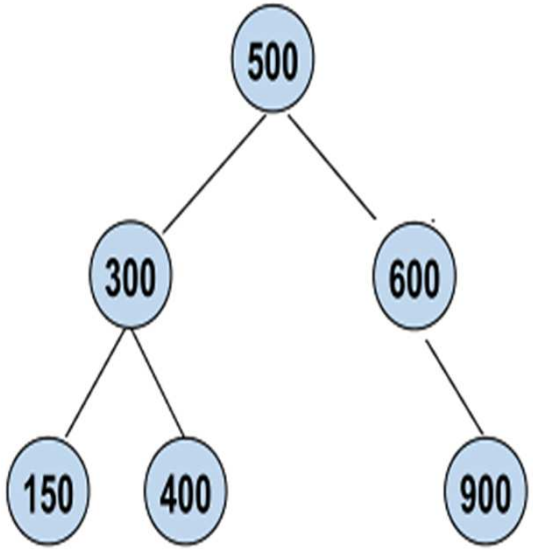

# Arvores Binárias Remoção de um Nó

- **Caso 3:** Remover um nó que possui duas sub-árvores

**Vamos efetuar pelo MAIOR VALOR DA SUB-ÁRVORE DA ESQUERDA**



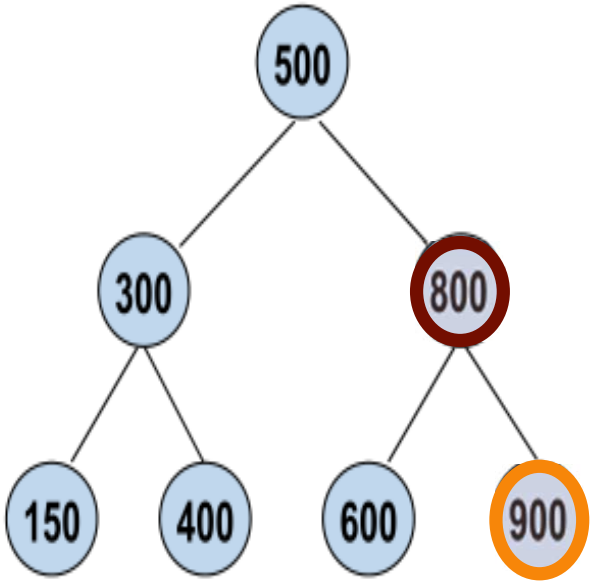



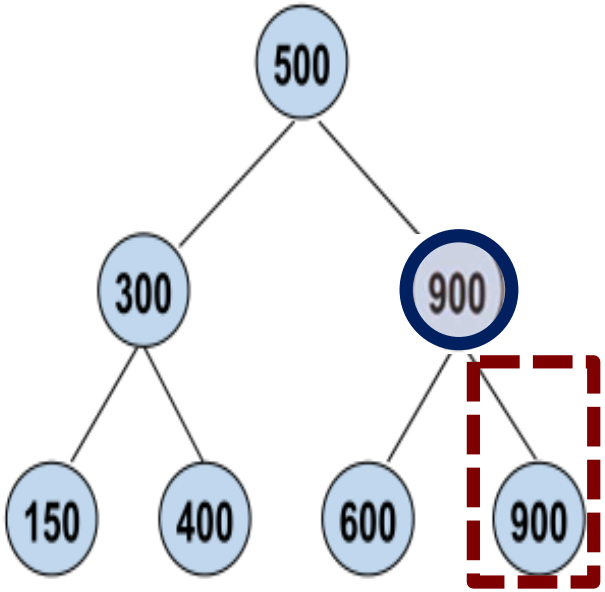


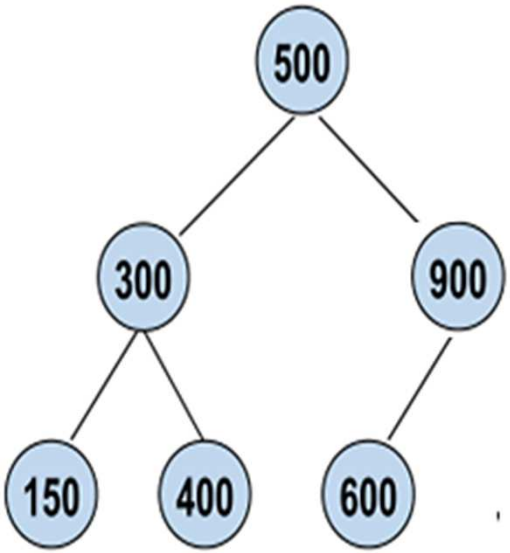
# Arvores Binárias Remoção de um Nó

- **Caso 3:** Remover um nó que possui duas sub-árvores

**Vamos efetuar pelo menor VALOR DA SUB-ÁRVORE DA DIREITA**



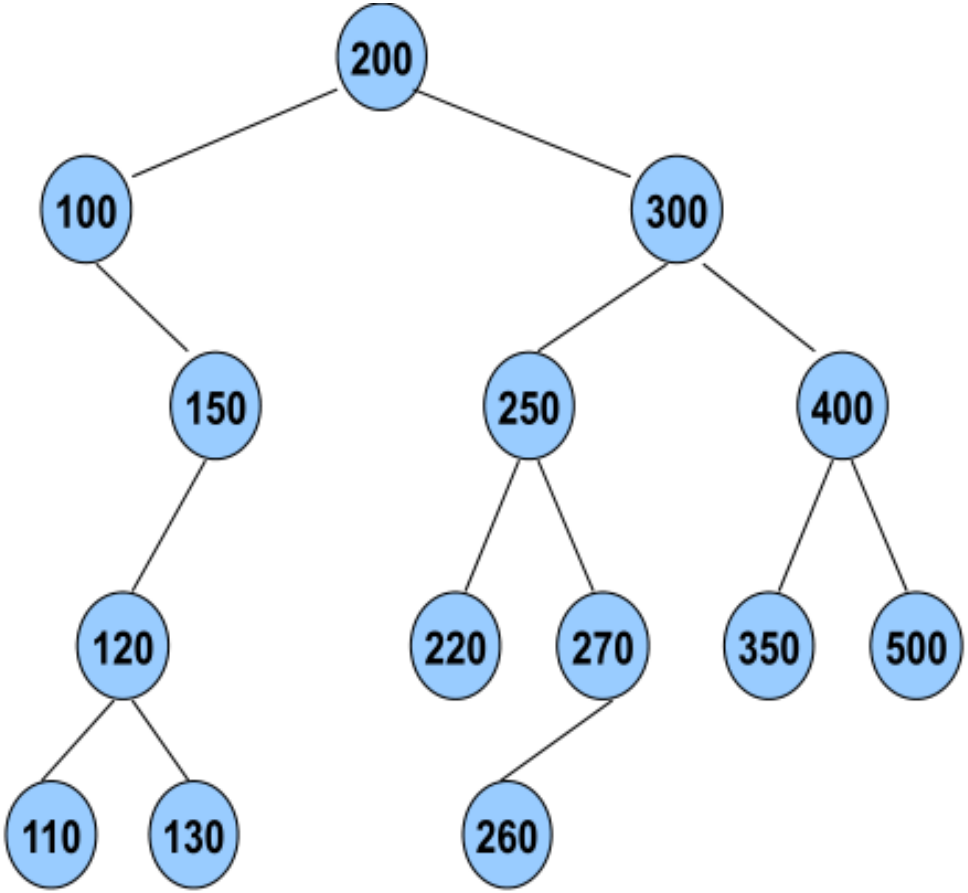


# Arvores Binárias Remoção de um Nó

- **Caso 3:** um nó interno que possui as duas sub-árvores (direita e esquerda)
  - **PARA AS NOSSAS AULAS** adotaremos a estratégia:
    - **Trocar** o **MAIOR valor** do nó da **sub-árvore da esquerda**


# Arvores Binárias Remoção de um Nó

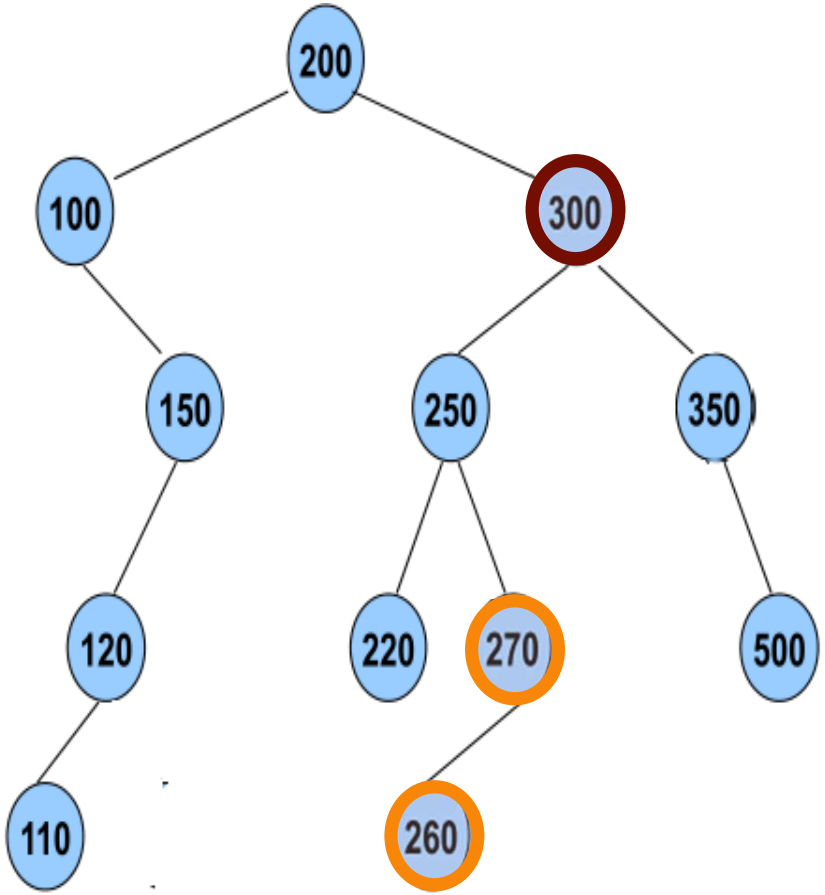
- **Exercício –** Dada a árvore abaixo, como ela ficaria se removêssemos os seguintes nós: 130, 400, 300

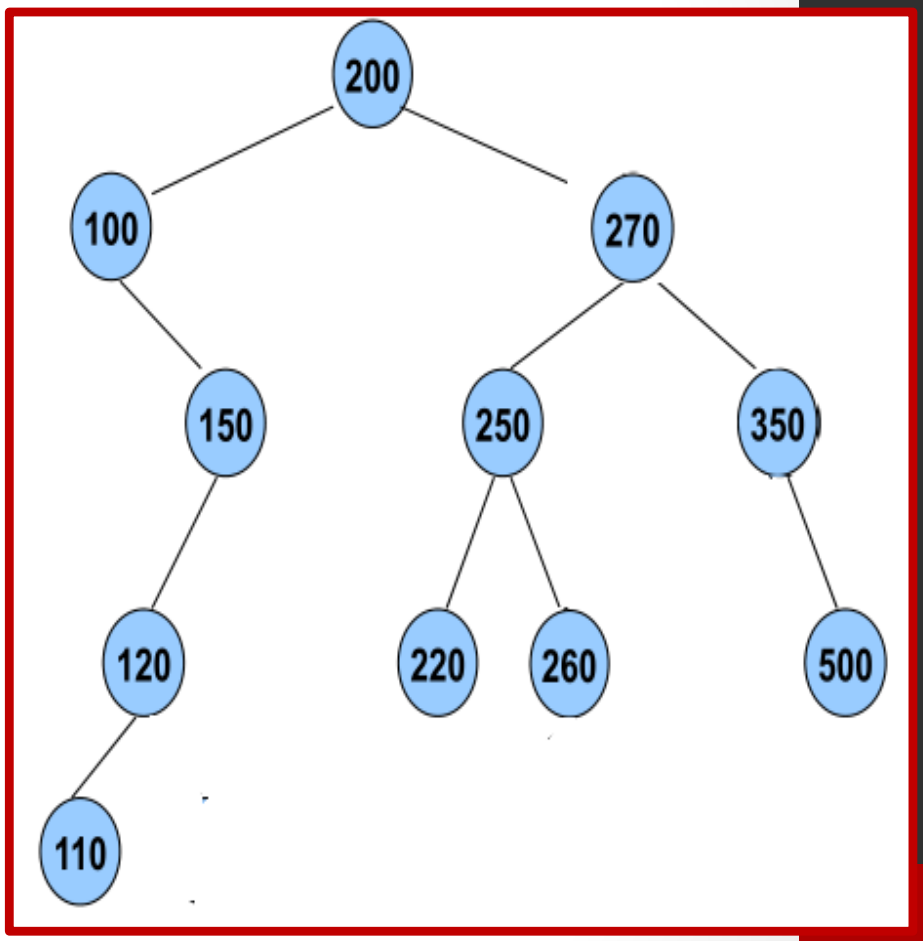




# Arvores Binárias Remoção de um Nó

- **Exercício –** Dada a árvore abaixo, como ela ficaria se removêssemos os seguintes nós: 130, 400, 300



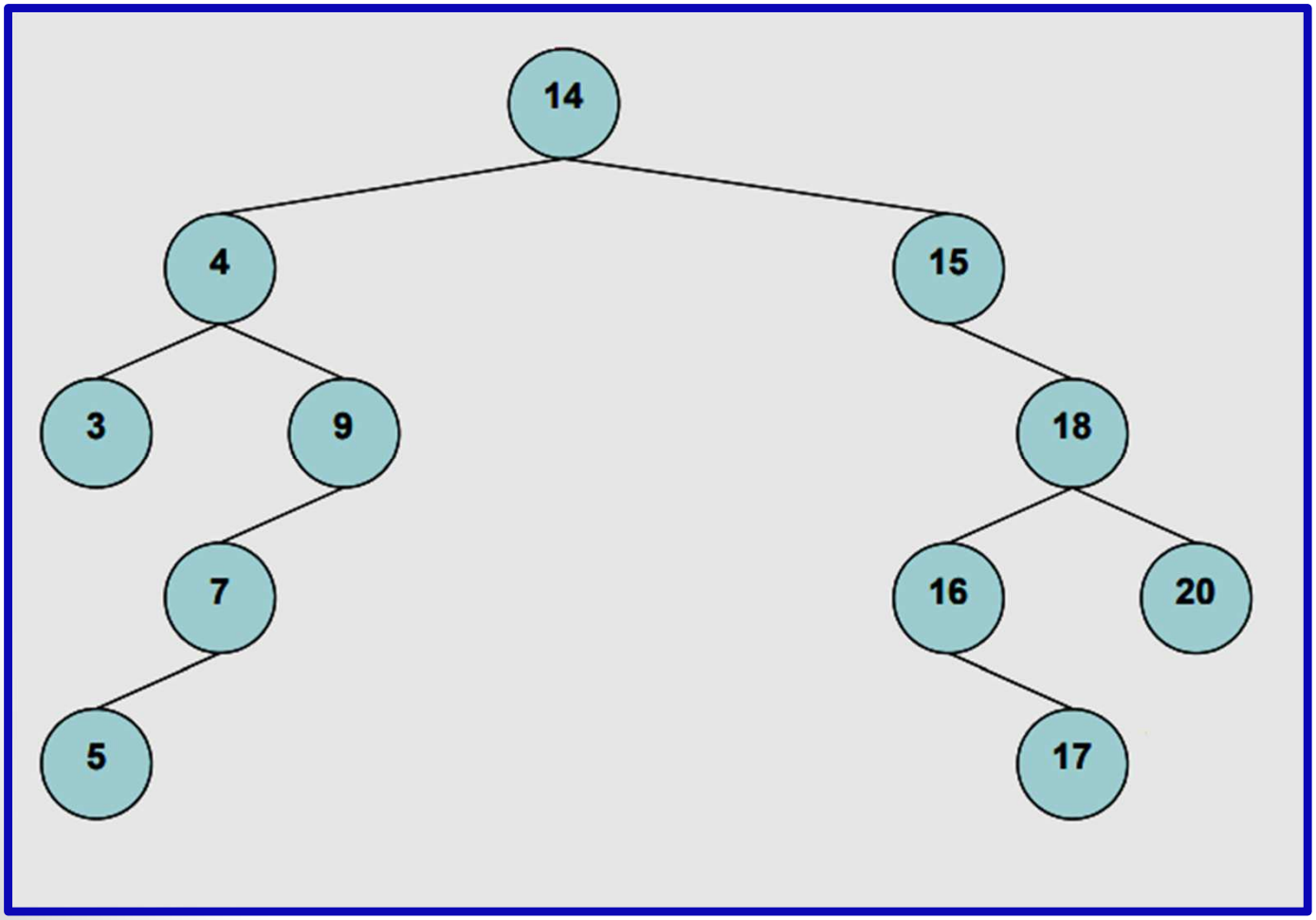

**Falta excluir o 300**

**NÃO SE ESQUEÇA o MAIOR valor da sub-árvore da esquerda**

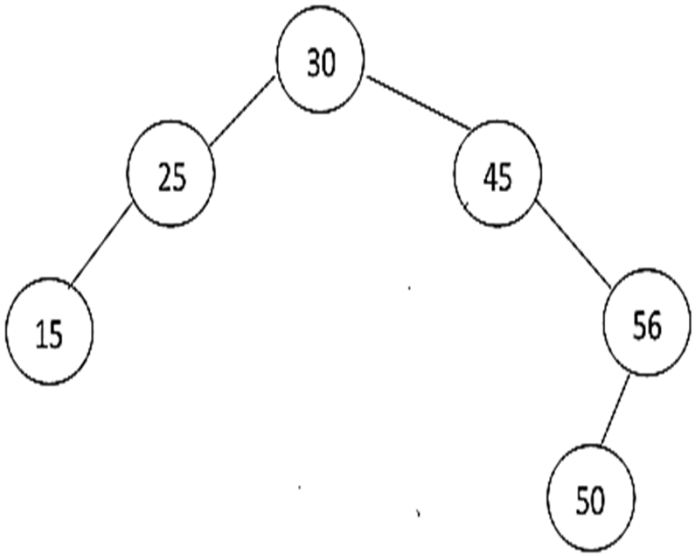



- **Exercícios**

  Remova o nó 14 da árvore usando a regra **o maior valor da sub-árvore da esquerda**




- **Exercícios – Resolva os exercícios procurando a forma mais eficiente .**
  1. Mostre passo a passo a árvore binária de busca resultante das seguintes operações:
     - Inserção de 5, 9, 7, 4, 12, 15, 11
     - Remoção do 9
  2. Dada a árvore abaixo, Pede-se: **(cada item desenhe a nova árvore)**




a. Insira nesta árvore os números 27, 58, 26, 49, 28
b. Remova o elemento 30
c. Remova o elemento 49
d. Remova o elemento 25
e. Determine a profundidade da árvore resultante
f. Quantos nós possui o nível 2 da árvore resultante?




- **Arvores Binárias – Estruturas em C**

```
typedef struct NoArvore
{
    int info;
    struct NoArvore *esq;
    struct NoArvore *dir;
}Arv;
```

- Criar uma estrutura para armazenar a raiz dessa árvore

```
struct Arvore
{
    NoArv *raiz;
}
```

```
typedef struct Arvore
{
    NoArv *raiz;
}Arv;
```




- **O Procedimento em C irá considerar os três casos**
  - Caso 1: Remover uma folha
  - Caso 2: Remover um nó que possui um único filho
  - Caso 3: Remover um nó que possui os dois filhos




- **O Procedimento em C**

```
Arv* remover( Arv *RAIZ, int num)
{
   NoArv *aux=RAIZ->raiz;
   if(aux->info==num && aux->dir==NULL && aux->esq==NULL)
   {
      free(aux);
      free(RAIZ);
      return NULL;
   }
   RAIZ->raiz = remover_aux(RAIZ->raiz,num);
   return RAIZ;

}
```




- **O Procedimento em C**

```
Arv* remover( Arv *RAIZ, int num)
{
   NoArv *aux=RAIZ->raiz;
   if(aux->info==num && aux->dir==NULL && aux->esq==NULL)
   {
      free(aux);
      free(RAIZ);
      return NULL;
   }
   RAIZ->raiz = remover_aux(RAIZ->raiz,num);
   return RAIZ;

}
```




- **O Procedimento em C**

**A árvore só tem a raiz**

```
Arv* remover( Arv *RAIZ, int num)
{
   NoArv *aux=RAIZ->raiz;
   if(aux->info==num && aux->dir==NULL && aux->esq==NULL)
   {
      free(aux);
      free(RAIZ);
      return NULL;
   }
   RAIZ->raiz = remover_aux(RAIZ->raiz,num);
   return RAIZ;

}
```




- **O Procedimento em C**

```
Arv* remover( Arv *RAIZ, int num)
{
   NoArv *aux=RAIZ->raiz;
   if(aux->info==num && aux->dir==NULL && aux->esq==NULL)
   {
      free(aux);
      free(RAIZ);
      return NULL;
   }
   RAIZ->raiz = remover_aux(RAIZ->raiz,num);
   return RAIZ;

}
```

```
NoArv* remover_aux(NoArv *pai, int num)
{
   if(pai==NULL)
   {
      printf("\n \n não encontrado na árvore");
   }
   else
   {
       if(num > pai->info)
      {
         pai->dir= remover_aux(pai->dir,num);
      }
      else
      {
         if(num < pai->info)
         {
            pai->esq = remover_aux(pai->esq,num);
         }
         /*else achou o nó a ser removido*/
```

```
NoArv * remover_aux(NoArv *pai, int num)
{
    if(pai==NULL)
    {
        printf("\n \n não encontrado na árvore");
    }
    else
    {
         if(num > pai->info)
        {
            pai->dir= remover_aux(pai->dir,num);
        }
        else
        {
            if(num < pai->info)
            {
               pai->esq = remover_aux(pai->esq,num);
            }
            /*else achou o nó a ser removido*/
```

```
NoArv Arv* remover_aux(NoArv Arv *pai, int num)
{
   if(pai==NULL)
   {
      printf("\n \n não encontrado na árvore");
   }
   else
   {
       if(num > pai->info)
      {
         pai->dir= remover_aux(pai->dir,num);
      }
      else
      {
         if(num < pai->info)
         {
            pai->esq = remover_aux(pai->esq,num);
         }
         /*else achou o nó a ser removido*/
```

```
NoArv Arv* remover_aux(NoArv Arv *pai, int num)
{
   if(pai==NULL)
   {
      printf("\n \n não encontrado na árvore");
   }
   else
   {
       if(num > pai->info)
      {
         pai->dir= remover_aux(pai->dir,num);
      }
      else
      {
         if(num < pai->info)
         {
            pai->esq = remover_aux(pai->esq,num);
         }
         /*else achou o nó a ser removido*/
```

```
NoArv Arv* remover_aux(NoArv Arv *pai, int num)
{
   if(pai==NULL)
   {
      printf("\n \n não encontrado na árvore");
   }
   else
   {
       if(num > pai->info)
      {
         pai->dir= remover_aux(pai->dir,num);
      }
      else
      {
         if(num < pai->info)
         {
            pai->esq = remover_aux(pai->esq,num);
         }
         /*else achou o nó a ser removido*/
```

```
else /*achou o nó a ser removido*/
{
        if(pai->dir==NULL && pai->esq==NULL) /* No sem filhos */
        {
                free(pai);    pai=NULL;
         }
          else
         {
                if (pai->esq==NULL) /*so tem filho da direita */
                {
                   NoArv *aux=pai;  pai = pai->dir;   free(aux);
                }
                else
                {
                   if((pai->dir==NULL))/* so tem filho da esquerda */
                   {
                      NoArv *aux=pai;  pai = pai->esq;  free(aux);
                   }
                   /*else tem os dois filhos */
```

CASO 1 - FOLHA

```
else /*achou o nó a ser removido*/
{
        if(pai->dir==NULL && pai->esq==NULL) /* No sem filhos */
        {
                free(pai);    pai=NULL;
         }
          else
         {
                if (pai->esq==NULL) /*so tem filho da direita */
                {
                   NoArv *aux=pai;  pai = pai->dir;   free(aux);
                }
                else
                {
                   if((pai->dir==NULL))/* so tem filho da esquerda */
                   {
                      NoArv *aux=pai;  pai = pai->esq;  free(aux);
                   }
                   /*else tem os dois filhos */
```

CASO 2 – SÓ UMA FILHO

```
else /*achou o nó a ser removido*/
{
      if(pai->dir==NULL && pai->esq==NULL) /* No sem filhos */
      {
            free(pai);    pai=NULL;
       }
        else
       {
            if (pai->esq==NULL) /*so tem filho da direita */
            {
               NoArv *aux=pai;  pai = pai->dir;   free(aux);
            }
            else
            {
               if((pai->dir==NULL))/* so tem filho da esquerda */
               {
                  NoArv *aux=pai;  pai = pai->esq;  free(aux);
               }
               /*else tem os dois filhos */
```

```
            else /* tem os dois filhos */
             {
                NoArv *aux;
                aux = pai->esq;
                while (aux->dir !=NULL)
                {
                   aux=aux->dir;
                }
                pai->info =aux->info; /* troca as informações */
                aux->info = num;
                pai->esq = remover_aux(pai->esq,num);
             }
          }
       }

     }// achou o nó a ser removido
   }
  } //else do não estar vazio
  return pai;
}
```

CASO 3 – DOIS FILHOS