Árvore Binária

Rômulo César Silva

Unioeste

Junho de 2016

Sumário

Árvore

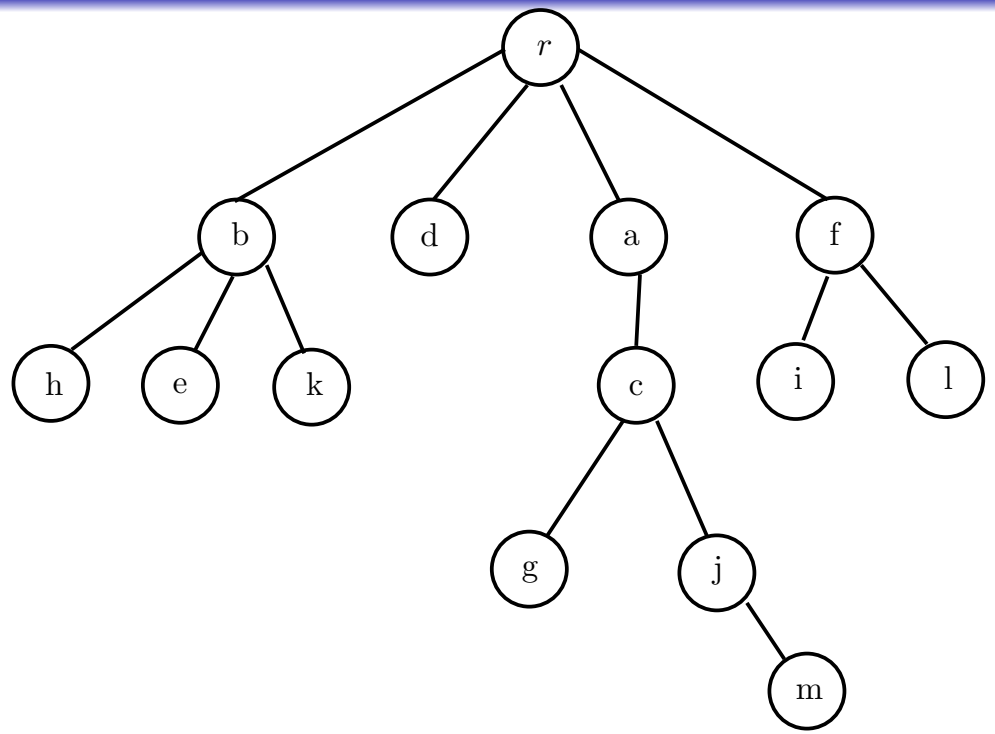
Árvore

Uma **árvore** é um conjunto de nós tal que:

- existe um nó *r* denominado *raiz* que tem uma ou mais subárvores
- os nós destas subárvores são os *filhos* de *r*, que por sua vez podem ter filhos ou não
- nós sem filhos são denominados *folhas* ou *externos*
- nós com filhos são denominados *internos*

Note que a definição é **recursiva!**

Árvore - exemplo



- raiz: r
- nós internos: r, b, a, c, j, f
- folhas: h, e, k, d, g, m, i, l
- filhos do nó b: h, e, k

Árvore Binária

Árvore Binária

Uma **árvore binária** é uma árvore que cada nó tem zero, um ou dois filhos.

Uma árvore binária pode ser:

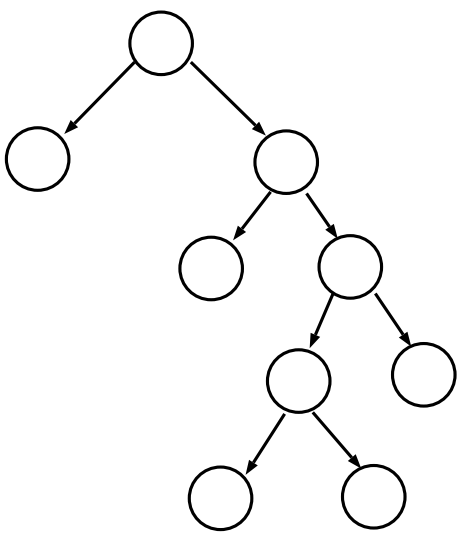
- vazia ou
- uma raiz com duas subárvores (esquerda e direita), ambas árvores binárias.

Árvore Binária

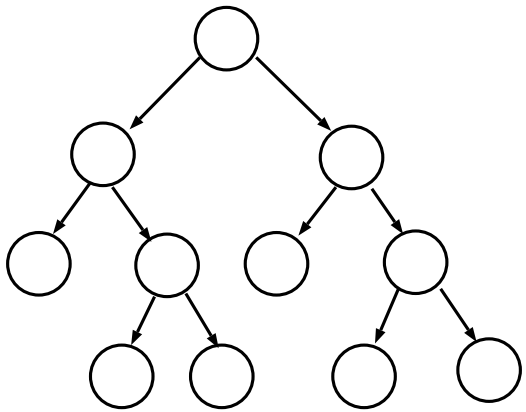
Definições

- A **altura** de um nó x é a distância entre x e seu descendente mais afastado. A altura de uma árvore é a altura da sua raiz.
- A **profundidade** de um nó x é a distância entre a raiz e x.
- **árvore estritamente binária**: cada nó tem exatamente 0 ou 2 filhos.
- **árvore binária cheia**: todas as folhas estão no mesmo nível.
- **árvore binária completa**: nós com filhos vazios estão no último ou penúltimo nível.
- **árvore binária balanceada**: se cada nó tem aproximadamente a mesma altura.

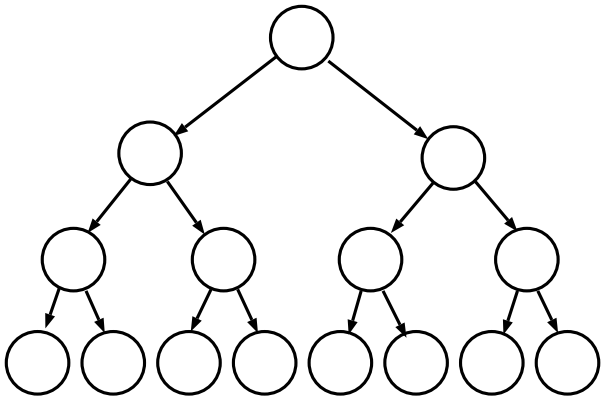
Árvore Binária - exemplo



estritamente binária



completa



cheia

- Toda árvore cheia é completa, estritamente binária e balanceada.
- Toda árvore completa é balanceada.

Árvore Binária

Estrutura:

```
  struct no {
     int info;          // informação armazenada
     struct no * esq; // subárvore esquerda
     struct no * dir; // subárvore direita
  };
 typedef struct no* arvore; //árvore é um ponteiro
                            //  para um nó
// Testa se uma árvore é vazia
 int vazia(arvore r) {
   return (r == NULL);
 }
```

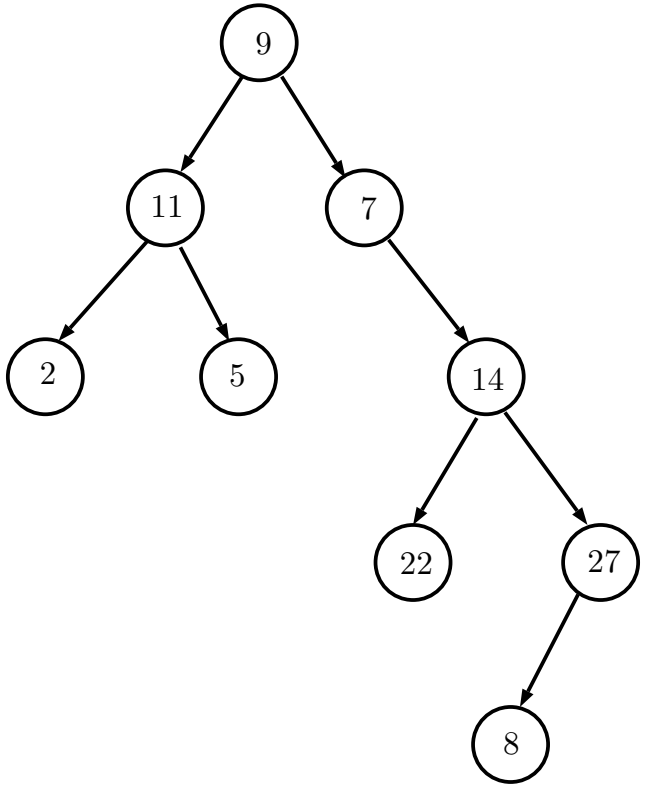
Árvore Binária

Percorrimento

Existem 3 formas de percorrer uma árvore binária:

- *in-ordem*: subárvore esquerda $\rightarrow$ raiz $\rightarrow$ subárvore direita
- *pré-ordem*: raiz $\rightarrow$ subárvore esquerda $\rightarrow$ subárvore direita
- *pós-ordem*: subárvore esquerda $\rightarrow$ subárvore direita $\rightarrow$ raiz

Árvore Binária - exemplo



- in-ordem: 2, 11, 5, 9, 7, 22, 14, 8, 27
- pré-ordem: 9, 11, 2, 5, 7, 14, 22, 27, 8
- pós-ordem: 2, 5, 11, 22, 8, 27, 14, 7, 9

Árvore Binária - percurso in-ordem

```
// imprime os nós fazendo percorrimento in-ordem
 void in_ordem(arvore r){
   if(!vazia(r)){
     in_ordem(r->esq);
     printf("%d ", r->info);
     in_ordem(r->dir);
   }
 }
```

Árvore Binária - percurso pré-ordem

```
// imprime os nós fazendo percorrimento pré-ordem
 void pre_ordem(arvore r){
   if(!vazia(r)) {
      printf("%d ", r->info);
      pre_ordem(r->esq);
      pre_ordem(r->dir);
   }
 }
```

Árvore Binária - percurso pós-ordem

```
// imprime os nós fazendo percorrimento pós-ordem
 void pos_ordem(arvore r){
   if(!vazia(r)) {
      pos_ordem(r->esq);
      pos_ordem(r->dir);
      printf("%d ", r->info);
   }
 }
```

Árvore Binária de Busca

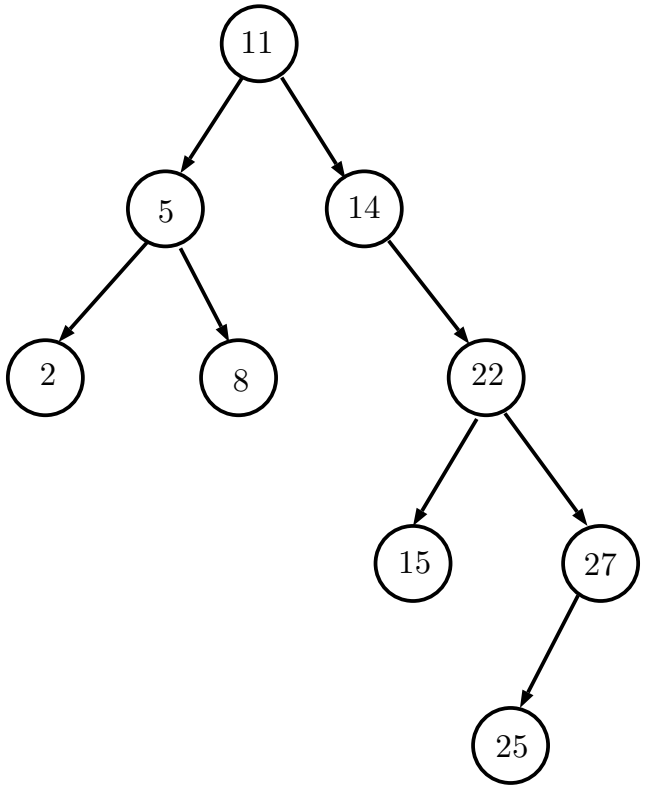
Árvore Binária de Busca

Um **árvore binária de busca** é uma árvore binária que atende à seguinte propriedade:
Seja x um nó da árvore. Se y é um nó na subárvore esquerda de x, então $chave[y] \leq chave[x]$. Se y é um nó na subárvore direita de x, então $chave[x] \leq chave[y]$.

- O percorrimento *in-ordem* de um árvore binária de busca imprime os elementos em ordem crescente.

Observação: daqui para frente, usaremos o termo **árvore binária** para designar uma árvore binária de busca.

Árvore Binária de Busca - exemplo



- percurso *in-ordem*: 2, 5, 8, 11, 14, 15, 22, 25, 27.

Árvore Binária de Busca

Busca do nó de uma chave x:

```
arvore busca_arvore_binaria(arvore r, int x){
  if(vazia(r))
    return NULL;
  if(r->info > x)
    return busca_arvore_binaria(r->esq, x);
  if(r->info < x)
    return busca_arvore_binaria(r->dir, x);
  return r;
}
```

Árvore Binária de Busca

Máximo e mínimo valor de chave da árvore:

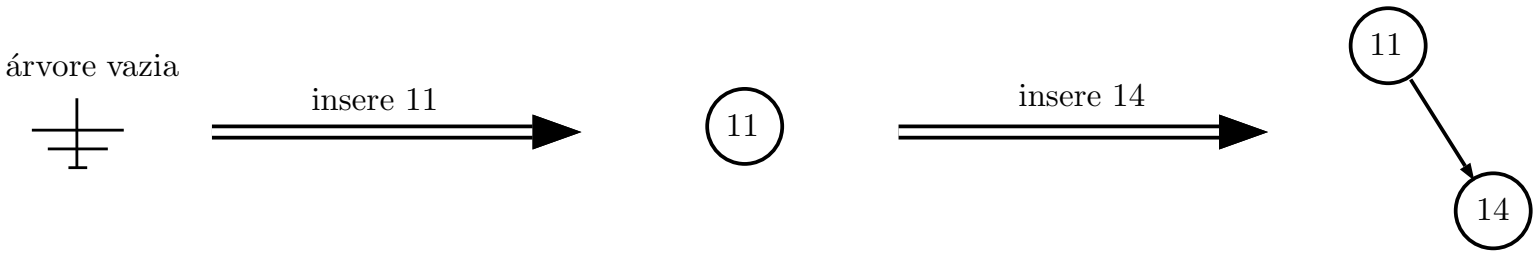
```
// Pré-condição: árvore não vazia
int maximo(arvore r){
  while(r->dir != NULL)
    r = r->dir;
  return r->info;
}

// Pré-condição: árvore não vazia
int minimo(arvore r){
  while(r->esq != NULL)
    r = r->esq;
  return r->info;
}
```

Árvore Binária - Inserção

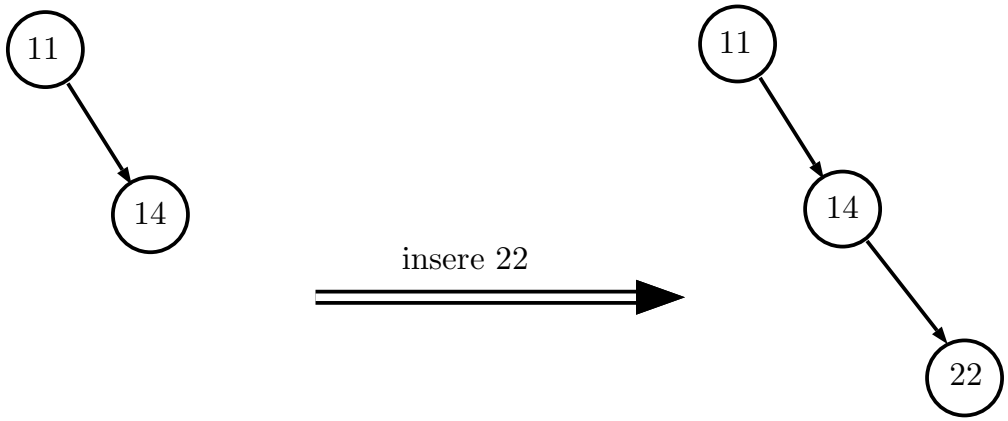
Inserção de uma chave x:

```
arvore insere_arvore_binaria(arvore r, int x){
  if(vazia(r)) {
    r = (struct no*) malloc(sizeof(struct no));
    r->info = x;
    r->esq = NULL;
    r->dir = NULL;
  }
  else if(x < r->info)
        r->esq = insere_arvore_binaria(r->esq, x);
      else // x >= r->info
        r->dir = insere_arvore_binaria(r->dir, x);
  return r;
}
```

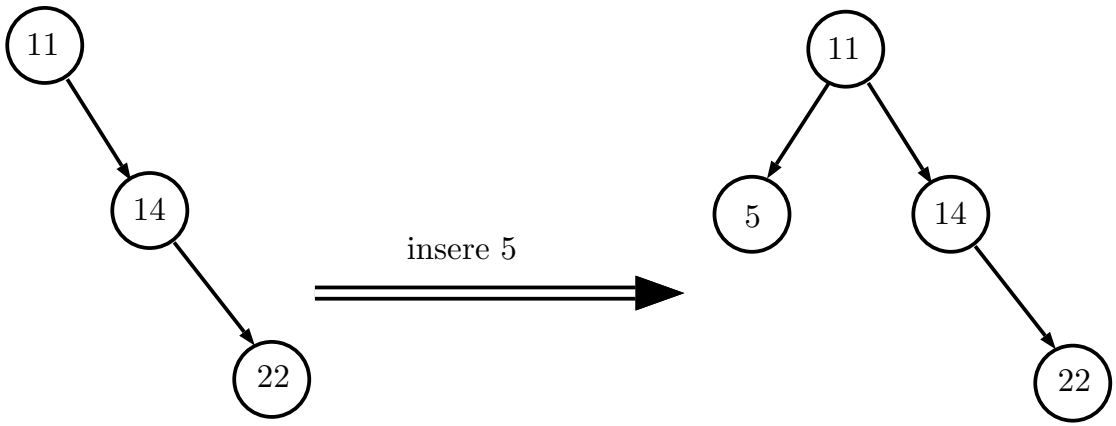
Árvore Binária- exemplo de inserção



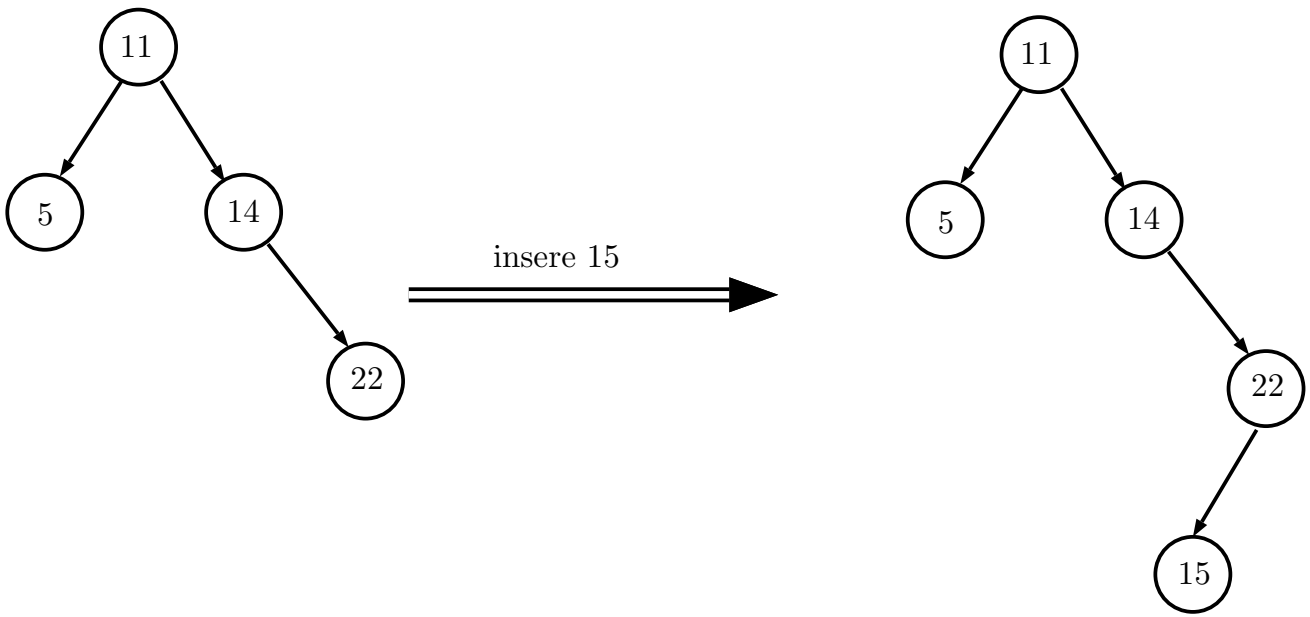
Árvore Binária- exemplo de inserção



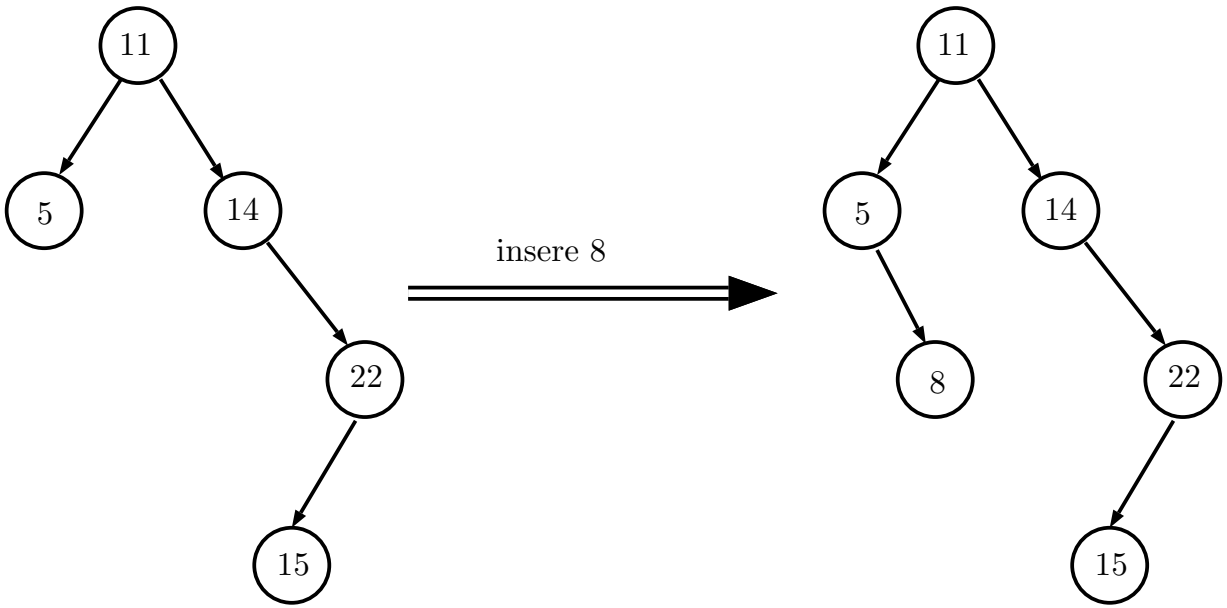
Árvore Binária- exemplo de inserção



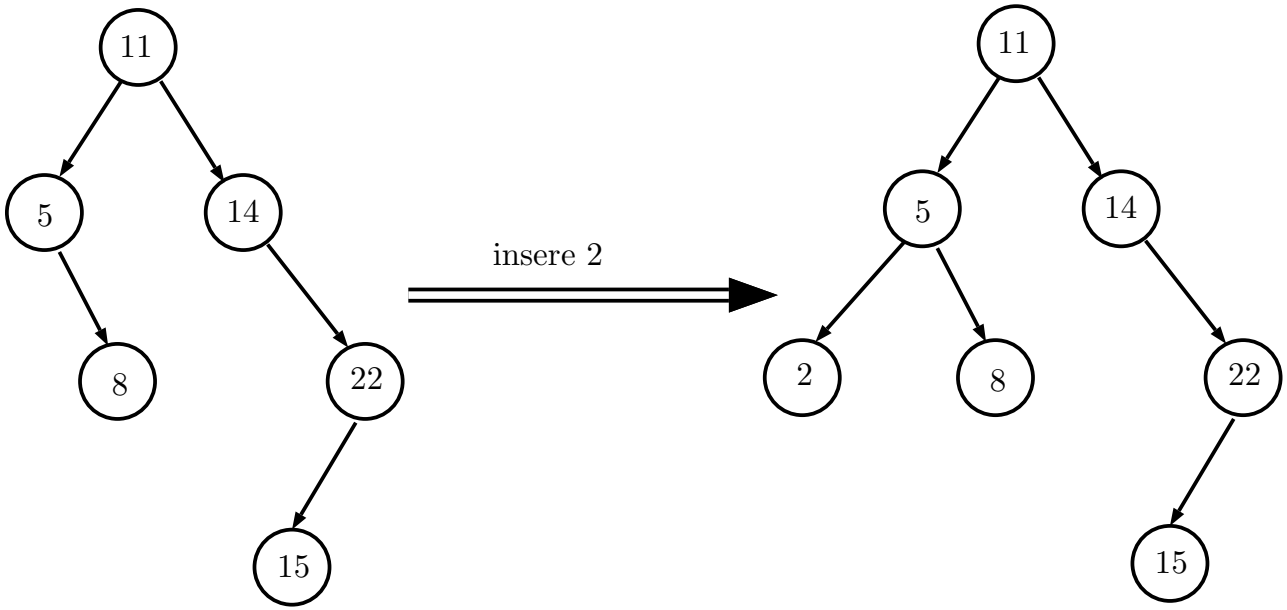
Árvore Binária- exemplo de inserção



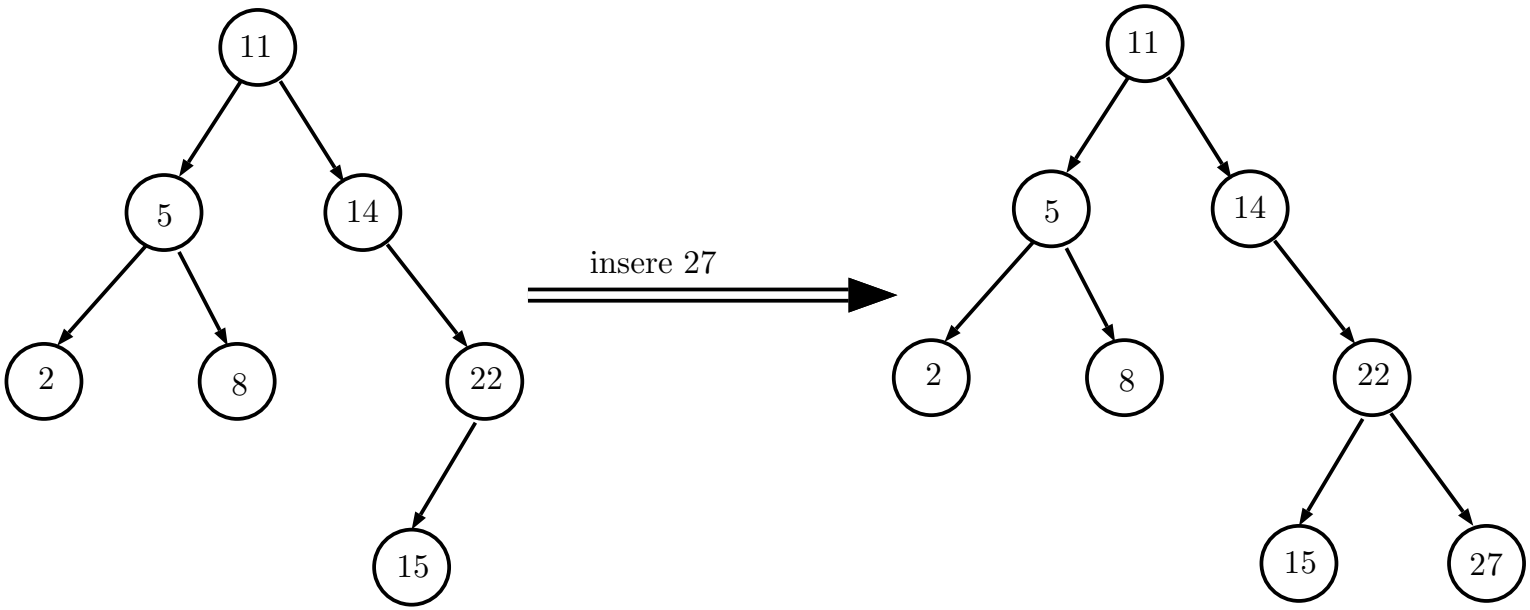
Árvore Binária- exemplo de inserção



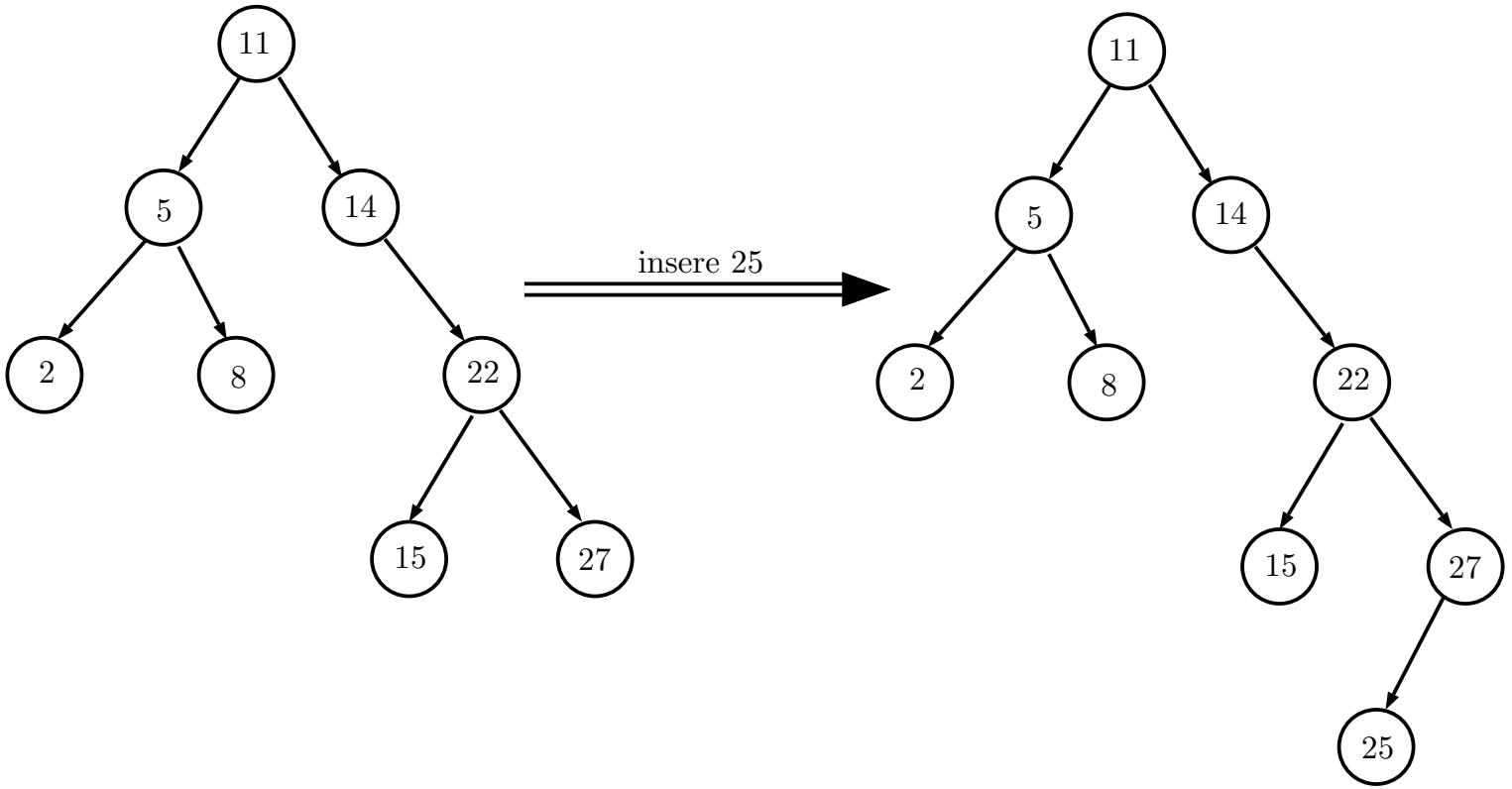
Árvore Binária- exemplo de inserção



Árvore Binária- exemplo de inserção



Árvore Binária- exemplo de inserção



Árvore Binária - Remoção

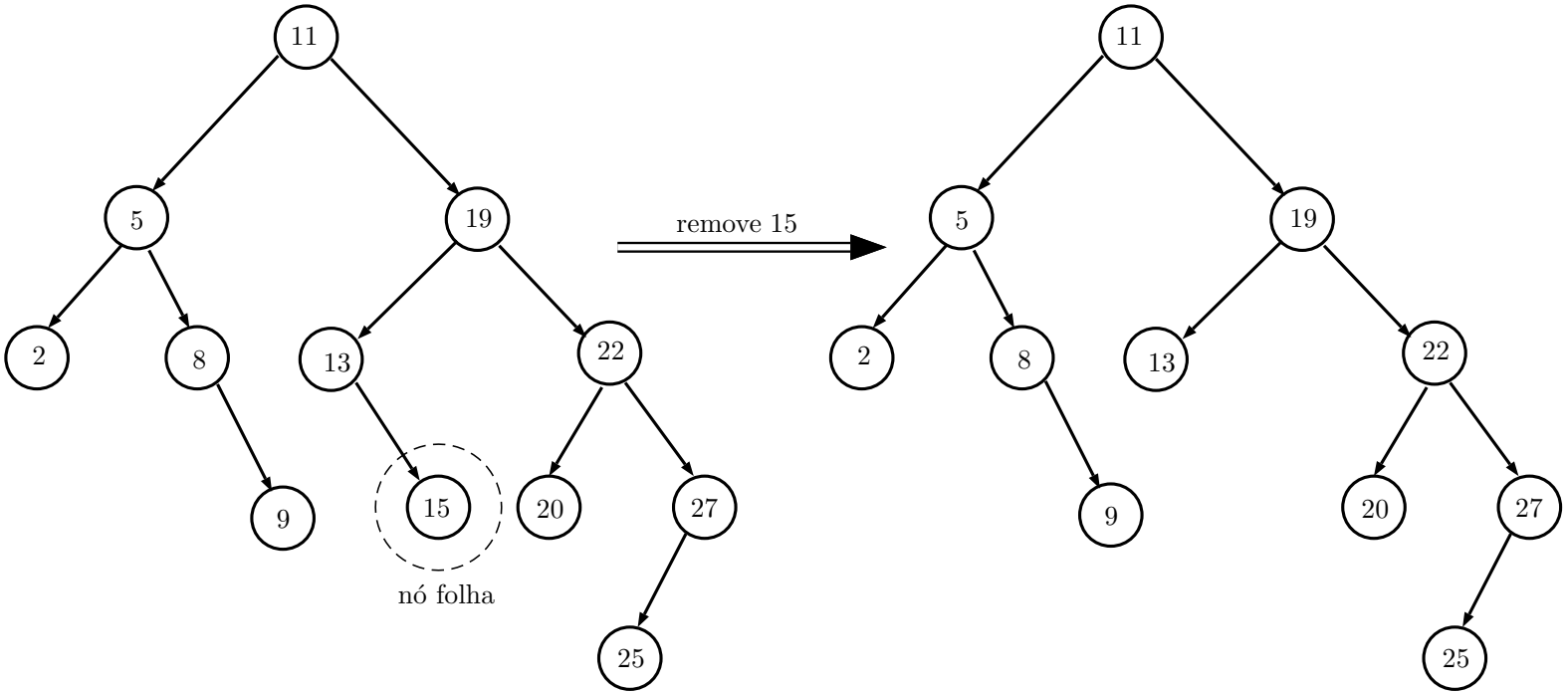
A remoção de uma chave x tem os seguintes casos:

- **caso** 1: a árvore é vazia então retorne NULL
- **caso 2**: a árvore não é vazia
  - **caso 2.1**: se x é menor que a chave da raiz, retire o elemento na subárvore esquerda
  - **caso 2.2**: se x é maior que a chave da raiz, retire o elemento na subárvore direita
  - **caso 2.3**: se x é igual a chave da raiz
    - **caso 2.3.1**: se x está em nó-folha, liberar o nó de x e retornar NULL
    - **caso 2.3.2**: se x está em nó interno, buscar o maior elemento entre os menores que x ou o menor elemento entre os maiores que x, e copiá-lo em cima de x e remover recursivamente o elemento copiado.

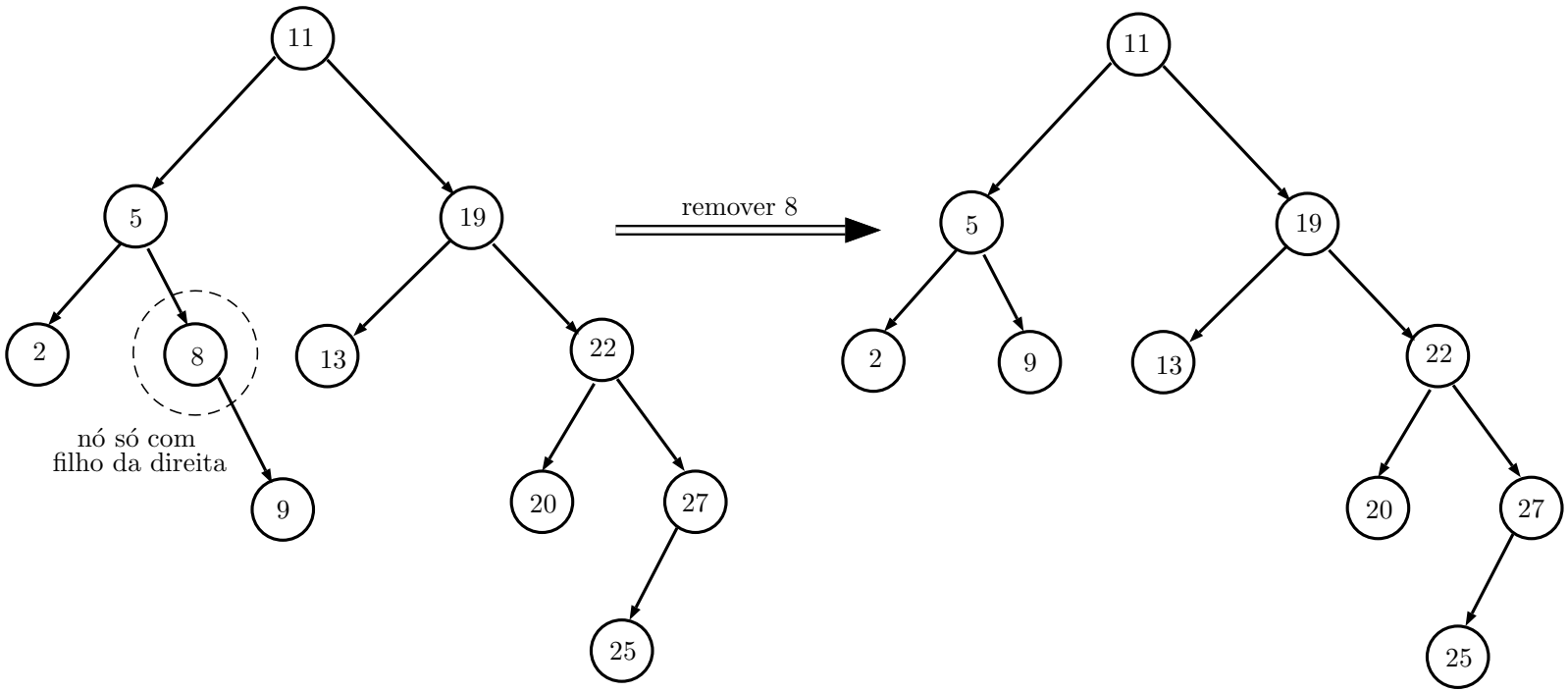
Árvore Binária - Remoção

```
arvore remove_arvore_binaria(arvore r, int x){
  if(vazia(r))
    return NULL;
  if(x < r->info)
     r->esq = remove_arvore_binaria(r->esq, x);
  else if(x > r->info)
         r->dir = remove_arvore_binaria(r->dir, x);
       else // x == r->info
            if(r->esq == NULL && r->dir == NULL) { // é nó folha
              free(r);
              r = NULL;
            }
            else if(r->esq == NULL) { // só tem filho da direita
                   r->info = minimo(r->dir);
                   r->dir = remove_arvore_binaria(r->dir,r->info);
                 }
                 else { // tem 2 filhos ou só o da esquerda
                        r->info = maximo(r->esq);
                        r->esq = remove_arvore_binaria(r->esq,r->info);
                      }
  return r;
}
```

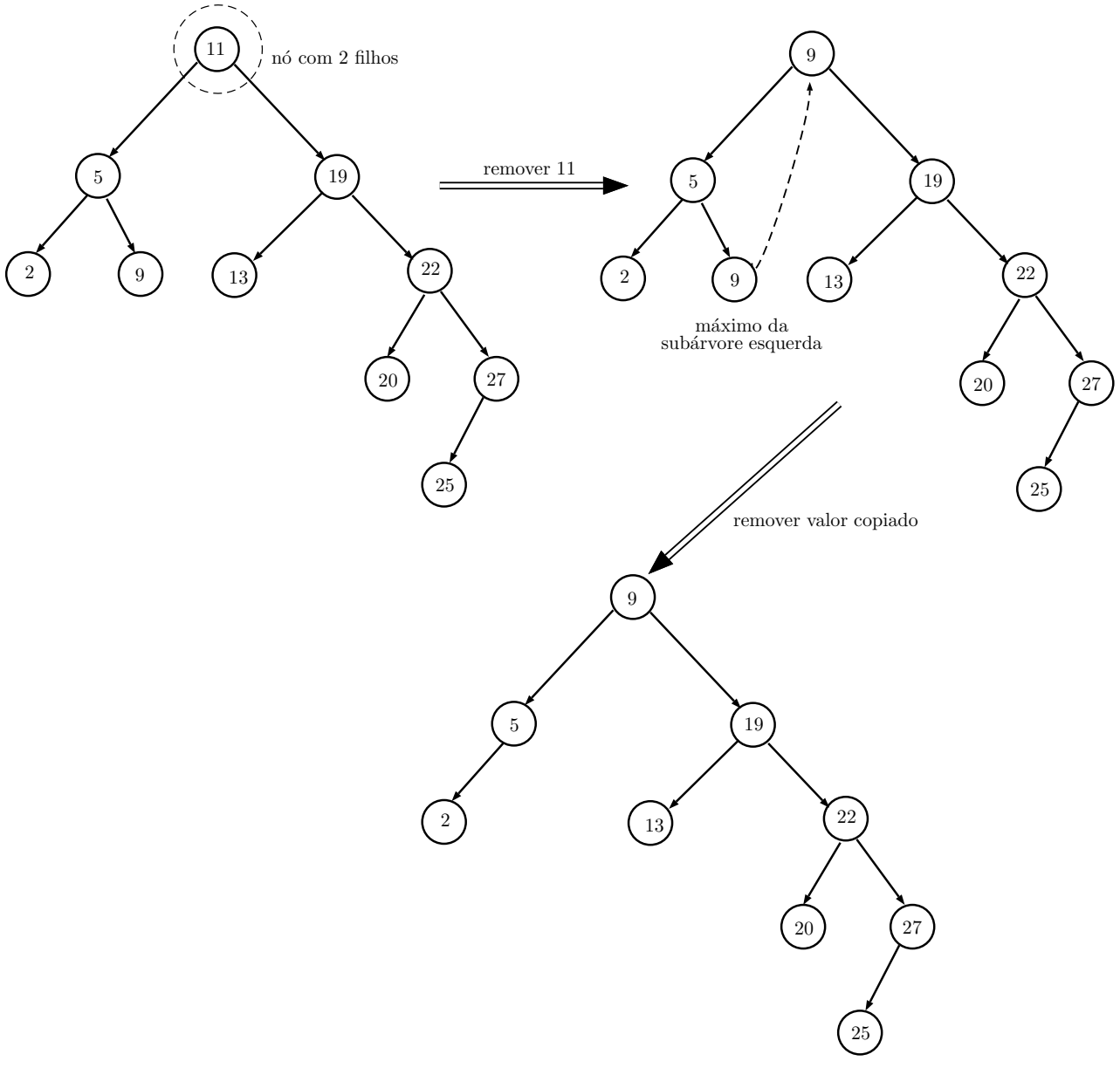
Árvore Binária- exemplo de remoção



Árvore Binária- exemplo de remoção



Árvore Binária- exemplo de remoção



Bibliografia I

[Cormen 1997] Cormen,T.; Leiserson, C.; Rivest, R.
Introduction to Algorithms. McGrawHill, New York, 1997.

[Feofiloff 2009] Paulo Feofiloff.
Algoritmos em linguagem C. Elsivier, Rio de Janeiro, 2009.