Da aula passada...

- **Ordenação**

?

“"Vivendo, se aprende; mas o que se aprende, mais, é só fazer outras maiores perguntas...."”

— Guimarães Rosa, *Grande Sertão: Veredas*

Estruturas de Dados (116319, D, 2018/1)

Roteiro da aula:

- Árvores (Conceito e TAD)
- Árvores Binárias
- Ordens de Percurso em Árvores Binárias
- Exemplos

Árvores

- Vetores e listas são ótimos para representar estruturas de dados lineares, mas não para dados hierárquicos

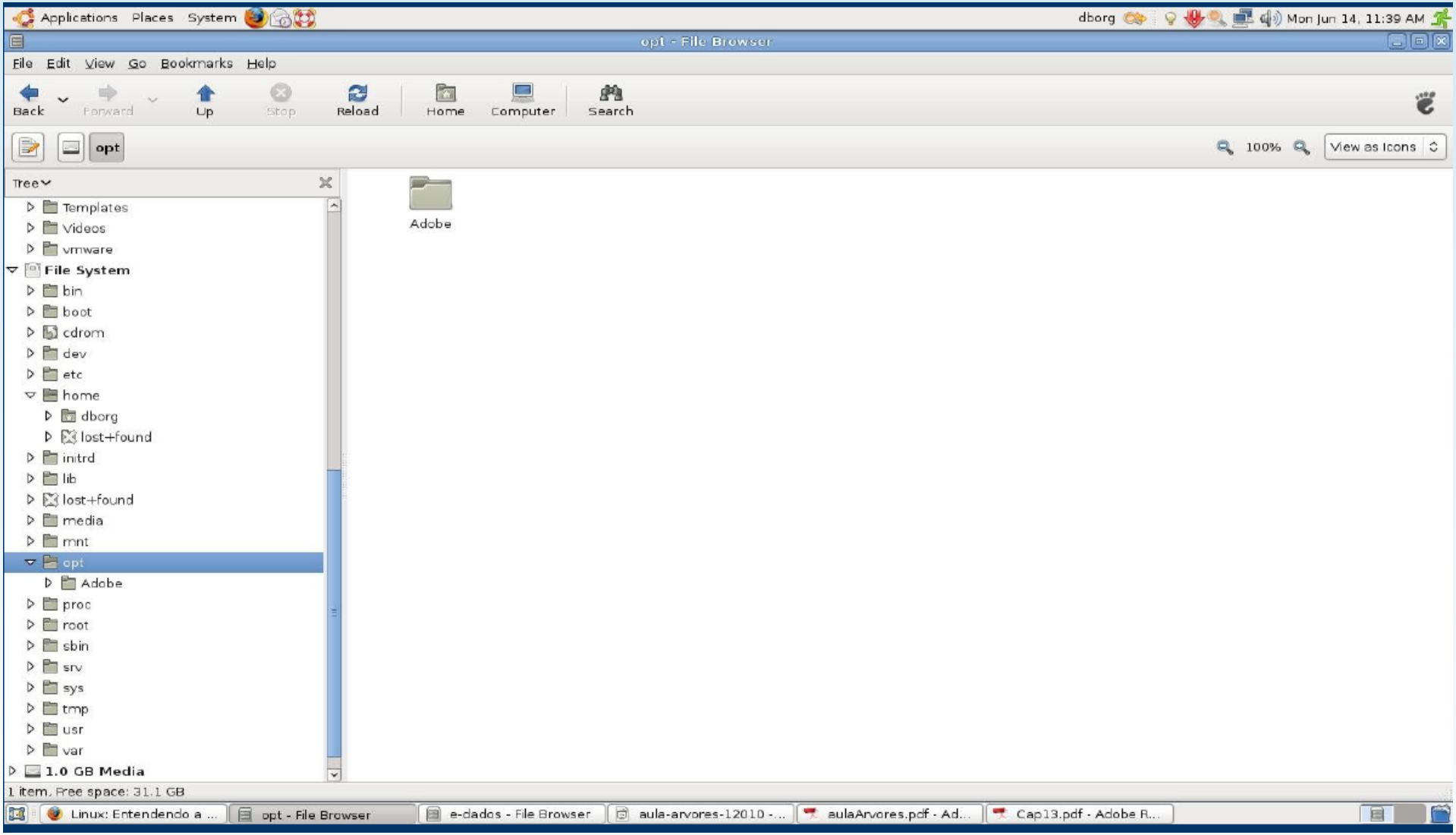
Árvores

- Vetores e listas são ótimos para representar estruturas de dados lineares, mas não para dados hierárquicos
- Exemplo de dados hierárquicos: sistema de arquivos em um computador

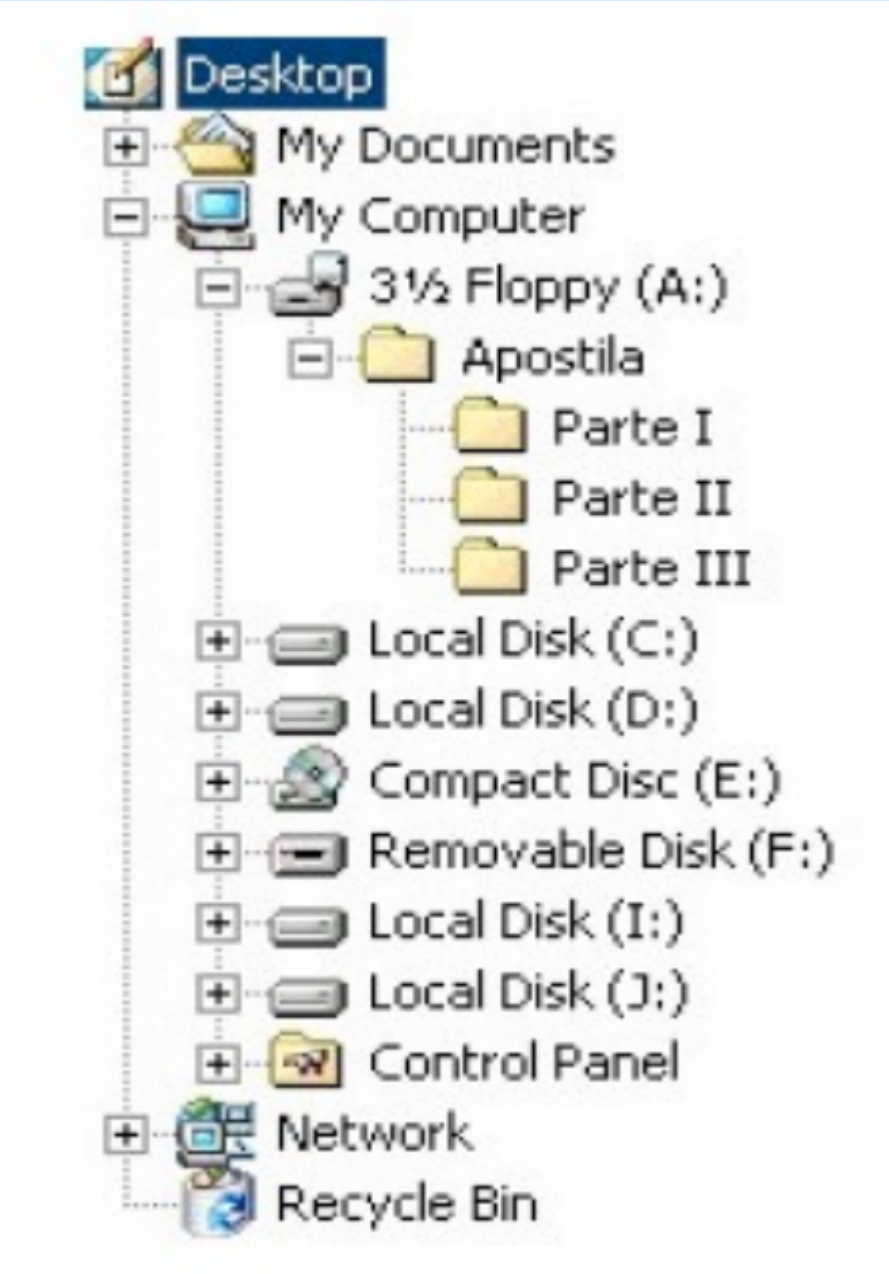
Árvores

- Vetores e listas são ótimos para representar estruturas de dados lineares, mas não para dados hierárquicos
- Exemplo de dados hierárquicos: sistema de arquivos em um computador
- Uma árvore é uma estrutura de dados recursiva, a qual permite representar dados dispostos de maneira hierárquica

Exemplo: Árvore de diretório



Exemplo: Árvore de diretório



Árvores

- **Uma árvore é composta por um conjunto de nós.**

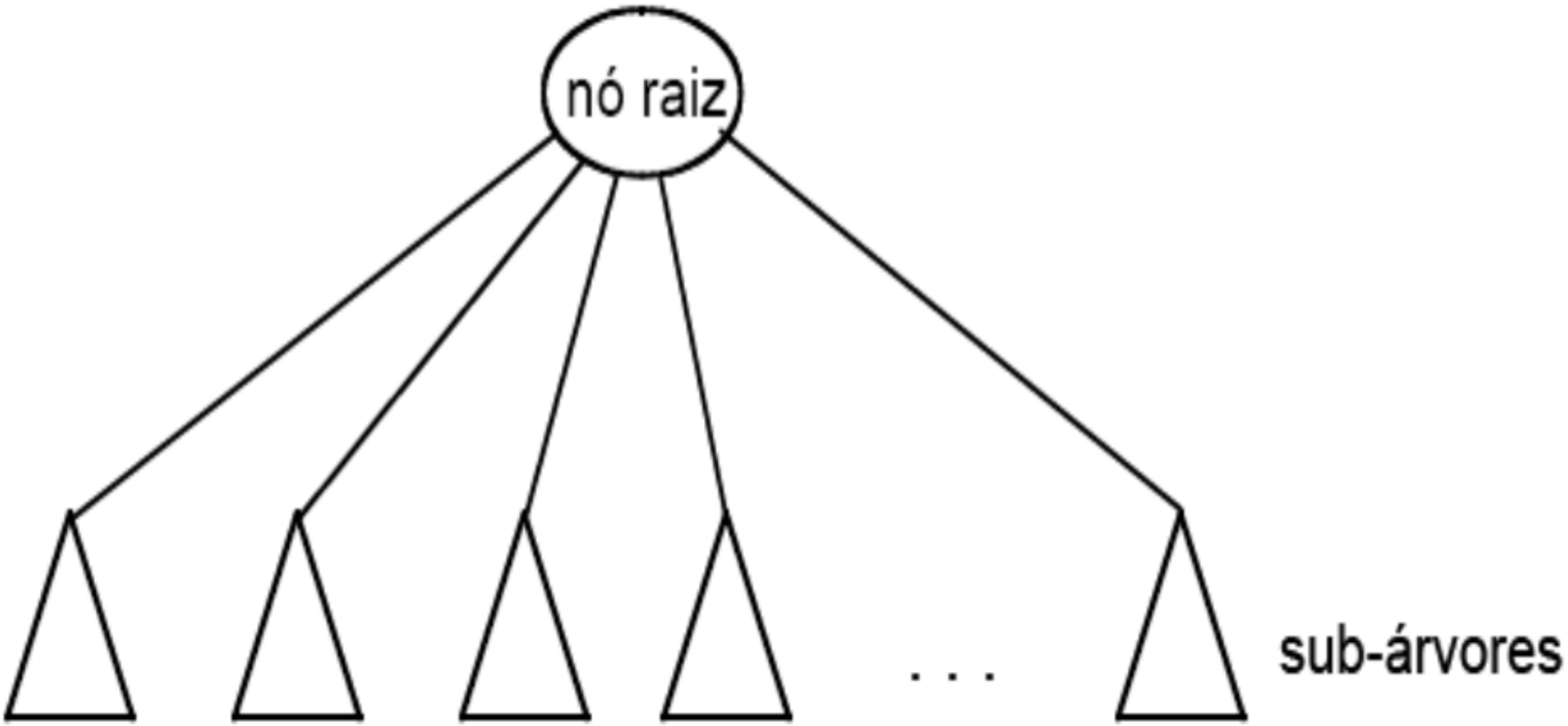
Árvores

- **Uma árvore é composta por um conjunto de nós.**
- **Existe um nó raiz (que contém zero ou mais sub-árvores)**

Árvores

- Uma árvore é composta por um conjunto de nós.
- Existe um nó raiz (que contém zero ou mais sub-árvores).
- Cada sub-árvore possui nós internos que possuem ligações que chegam ao nó raiz.
- Os nós mais externos, ou que não possuem filhos, são chamados de folhas (nós externos).

Árvores



Tipos de Árvores

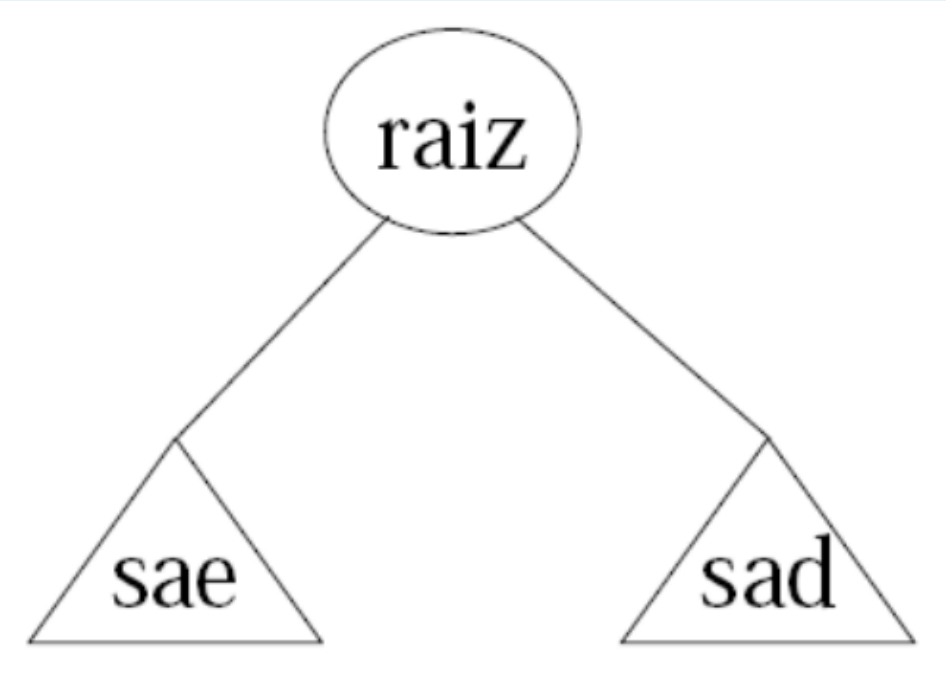
- **Árvores Binárias**
  - **Cada nó possui, no máximo, 2 filhos.**

Tipos de Árvores

- **Árvores Binárias**
  - **Cada nó possui, no máximo, 2 filhos.**
- **Árvores Genéricas**
  - **Cada nó pode ter vários filhos.**

Árvores Binárias

- **Cada nó possui zero, um, ou dois filhos.**
- **Pode ser definida recursivamente, como:**
  - **Árvore vazia, ou**
  - **Um nó raiz, tendo duas sub-árvores: esquerda (sae), e direita (sad).**



Representação textual de Árvores Binárias

- **Para uma árvore vazia <>**
- **Para uma árvore não vazia <raiz sae sad>**

Uso em avaliação de expressões

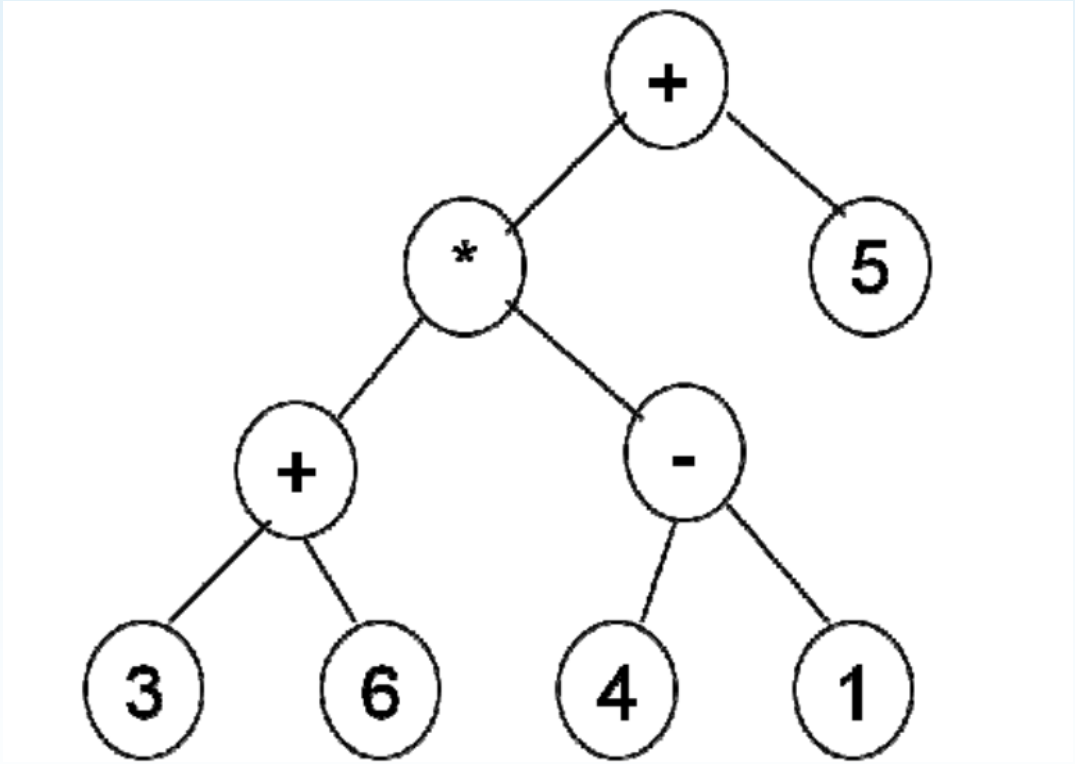
- **Sejam nós folhas representando operandos, e nós internos os operadores.**

Uso em avaliação de expressões

- **Sejam nós folhas representando operandos, e nós internos os operadores.**
- **Exemplo: (3+6)*(4-1)+5**

Uso em avaliação de expressões

- **Sejam nós folhas representando operandos, e nós internos os operadores.**
- **Exemplo: (3+6)*(4-1)+5**



Árvores Binárias (TAD)

- **Criar árvore vazia.**
- **Criar árvore não vazia.**
- **Verificar se árvore está vazia.**
- **Verificar se um elemento pertence à árvore.**
- **Liberar estrutura de árvore.**
- **Imprimir nós da árvore.**

Árvores Binárias (TAD)

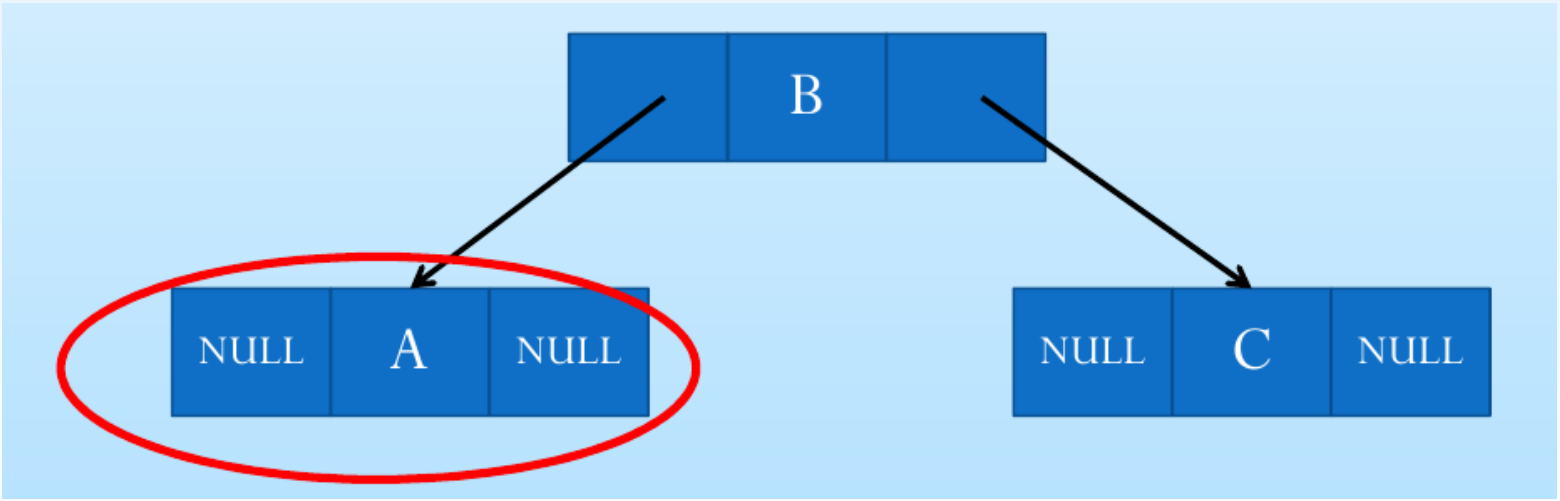
- **Em C: (exemplo)**

```
/* arquivo arvore.h */

typedef struct arv Arv;

Arv* arv_criavazia();

Arv* arv_cria(char c,Arv* e,Arv*d);

int arv_vazia(Arv* a);

int arv_pertence(Arv* a,char c);

Arv* arv_libera (Arv* a);

void arv_imprime (Arv* a);
```

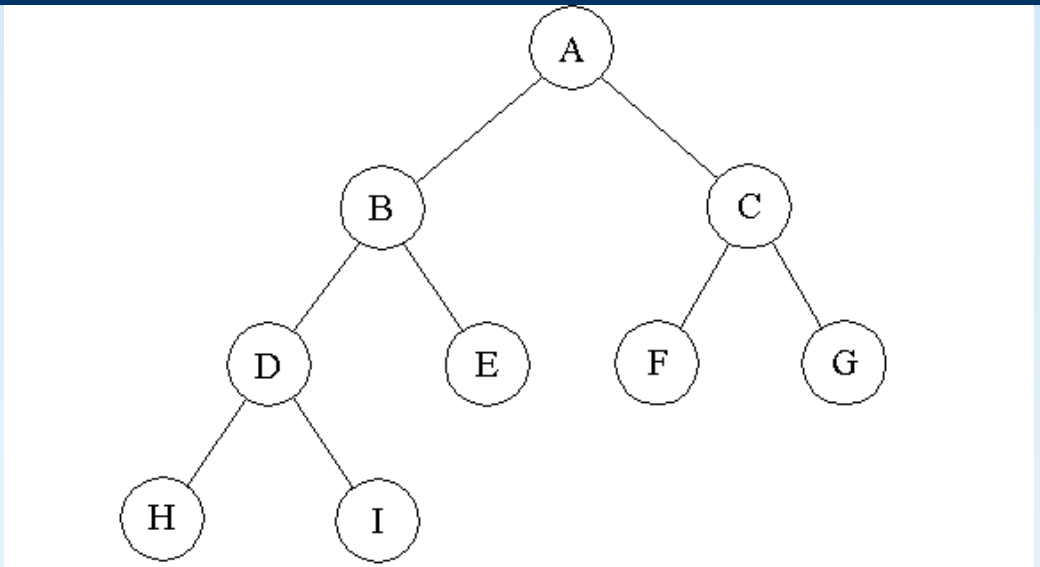
Árvores Binárias (TAD)

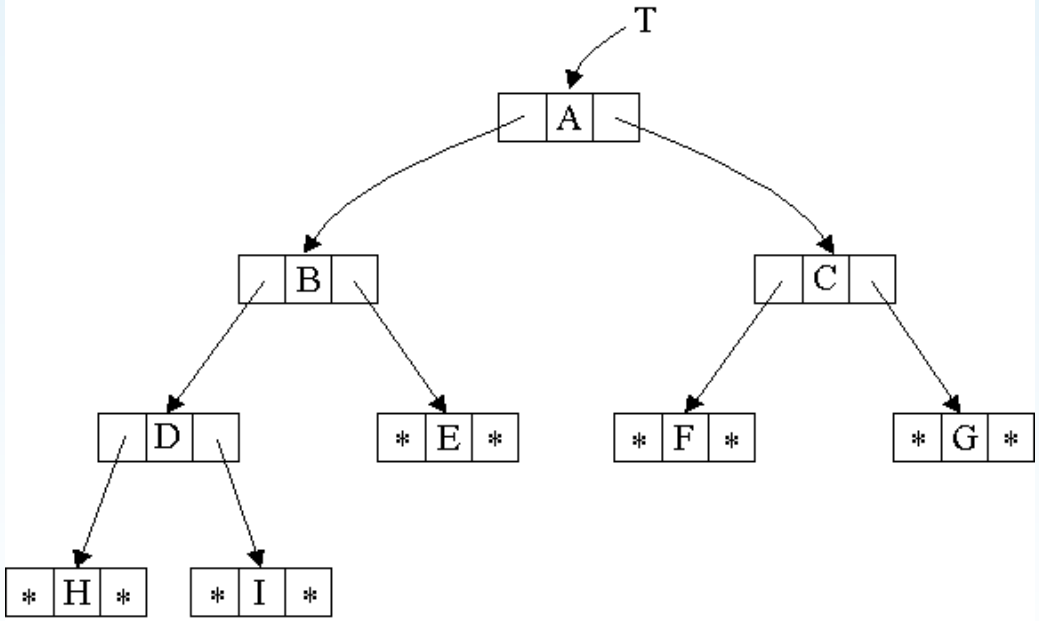
- **Em C: (exemplo)**

```
struct arv {
   char info;
   struct arv* esq;
   struct arv* dir;
} ;
```



Árvores Binárias (TAD)





Árvores Binárias (TAD)

- **Em C: (exemplo)**

```
struct arv {
   char info;
   struct arv* esq;
   struct arv* dir;
} ;
```

- O acesso a uma árvore se dará através de um ponteiro para o nó raiz

Árvores Binárias (TAD)

- **Em C: (exemplo)**

```
struct arv {
   char info;
   struct arv* esq;
   struct arv* dir;
} ;
```

- O acesso a uma árvore se dará através de um ponteiro para o nó raiz
- A estrutura de um nó deve ser composta por: um campo que guarda a informação e dois ponteiros: um para a sub-árvore da esquerda e um para a sub-árvore da direita

Árvores Binárias (TAD)

- **Em C: (exemplo)**

```
struct arv {
   char info;
   struct arv* esq;
   struct arv* dir;
} ;
```

- O acesso a uma árvore se dará através de um ponteiro para o nó raiz
- A estrutura de um nó deve ser composta por: um campo que guarda a informação e dois ponteiros: um para a sub-árvore da esquerda e um para a sub-árvore da direita
- Funções são implementadas utilizando definição recursiva da estrutura

Árvores Binárias (TAD)

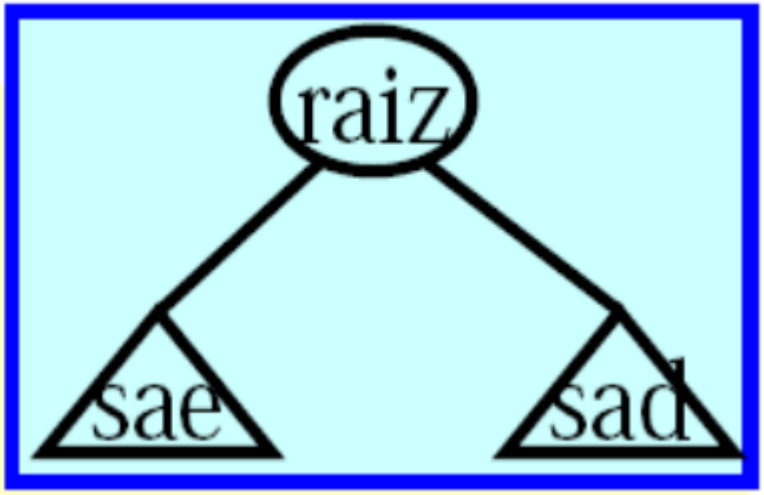
- **Função de criação (vazia)**

```
Arv* arv_criavazia( ) {
    return NULL;
}
```

Árvores Binárias (TAD)

- **Função de criação (não vazia)**

```
Arv* arv_cria (char c, Arv* sae, Arv* sad) {
  Arv* a = (Arv*) malloc (sizeof(Arv));
  a->info = c;
  a->esq = sae;
  a->dir = sad;
  return a;
}
```



Árvores Binárias (TAD)

- **Função que verifica se elemento pertence à árvore.**

```
int arv_pertence (Arv* a, char c) {
   if (arv_vazia(a))
      return 0; /* árvore vazia*/
   else
      return a->info == c ||
             arv_pertence (a->esq, c) ||
             arv_pertence (a->dir,c);
}
```

Árvores Binárias (TAD)

```
int arv_pertence (Arv* a, char c) {
    if (arv_vazia(a))
        return 0; /* árvore vazia*/
    else
        return a->info == c ||
               arv_pertence (a->esq, c) ||
               arv_pertence (a->dir,c);
}
```

Equivalente a:

```
if (c==a->info)
    return 1;
else if(arv_pertence(a->esq,c))
    return 1;
else
    return arv_pertence(a->dir,c);
```

Árvores Binárias (TAD)

- **Função que libera a estrutura da árvore.**

```
Arv* arv_libera (Arv* a) {
   if (!arv_vazia(a)) {
      arv_libera(a->esq); /* libera sae */
      arv_libera(a->dir);  /* libera sad */
      free(a);
   }
   return NULL;
}
```

Deve-se liberar recursivamente todos os elementos das sub-árvores primeiro

Árvores Binárias (TAD)

- **Função que verifica se uma árvore está vazia.**

```
int arv_vazia (Arv* a) {
  return (a==NULL);
}
```

Árvores Binárias (TAD)

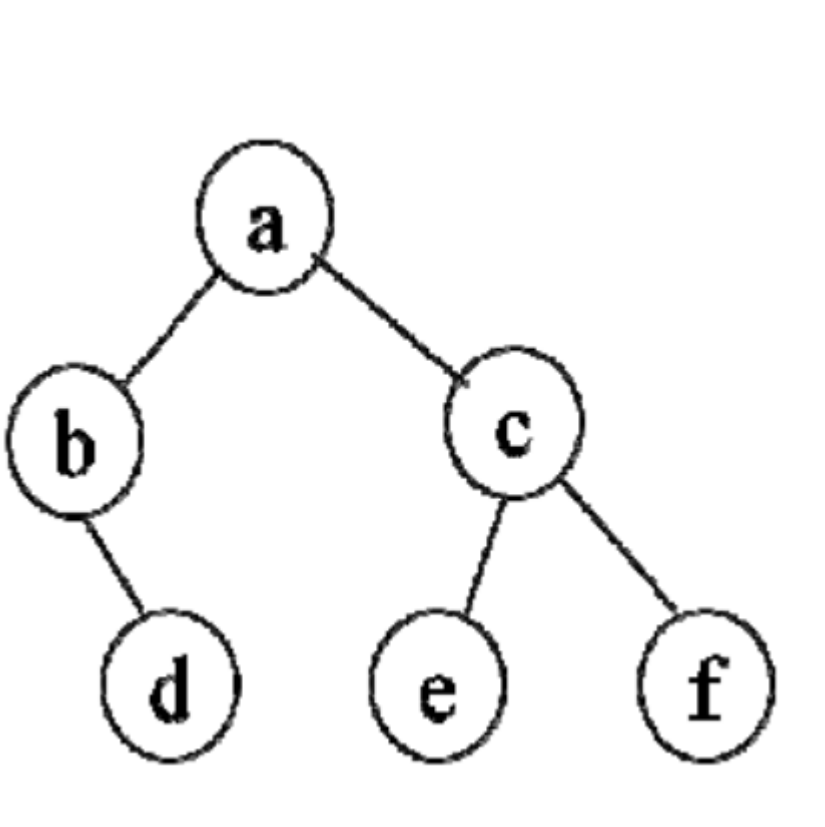
- **Função que imprime todos os elementos de uma árvore.**

```
void arv_imprime (Arv* a) {
  if (!arv_vazia(a)) {
     printf("%c ", a->info); /* mostra raiz */
     arv_imprime(a->esq);  /* mostra sae */
     arv_imprime(a->dir);   /* mostra sad */
   }
}
```

Implementação recursiva: primeiro visita a raiz, depois a sub-árvore da esquerda, para finalmente visitar a da direita

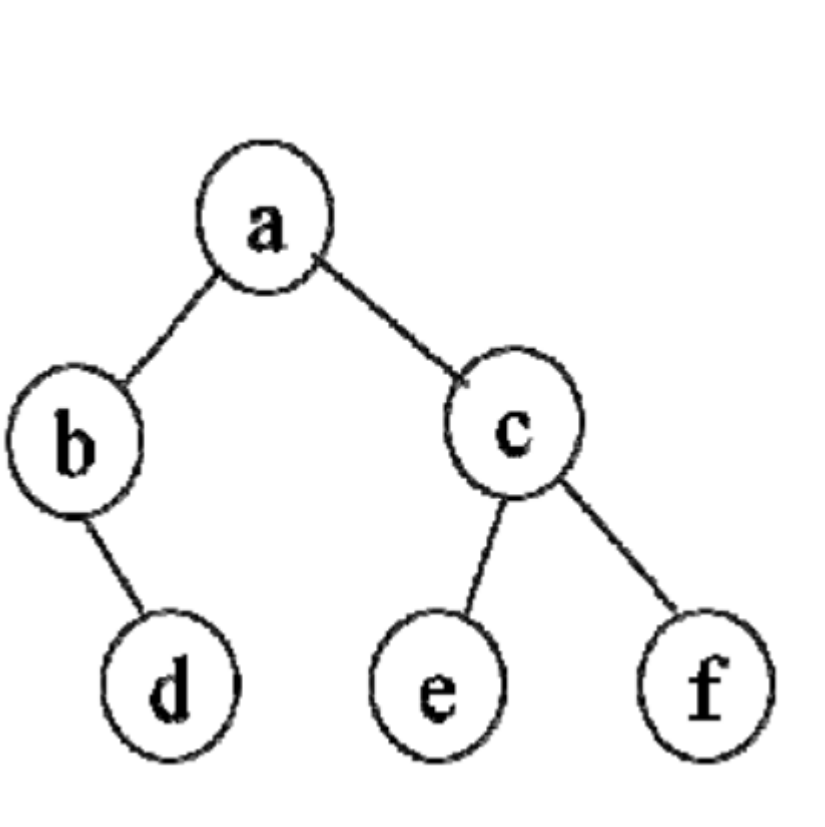
Exemplo: Árvores Binárias

- **Seja a seguinte árvore:**



Exemplo: Árvores Binárias

- **Seja a seguinte árvore:**



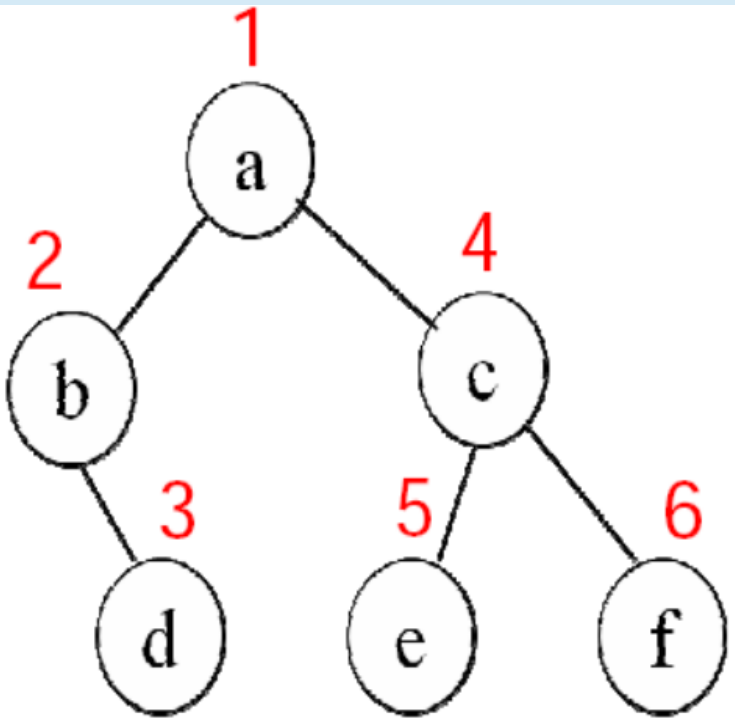
<a<b<><d<><>><c<e<><>><f<><>>>>

Exemplo: Árvores Binárias

- **Usando as funções do TAD**

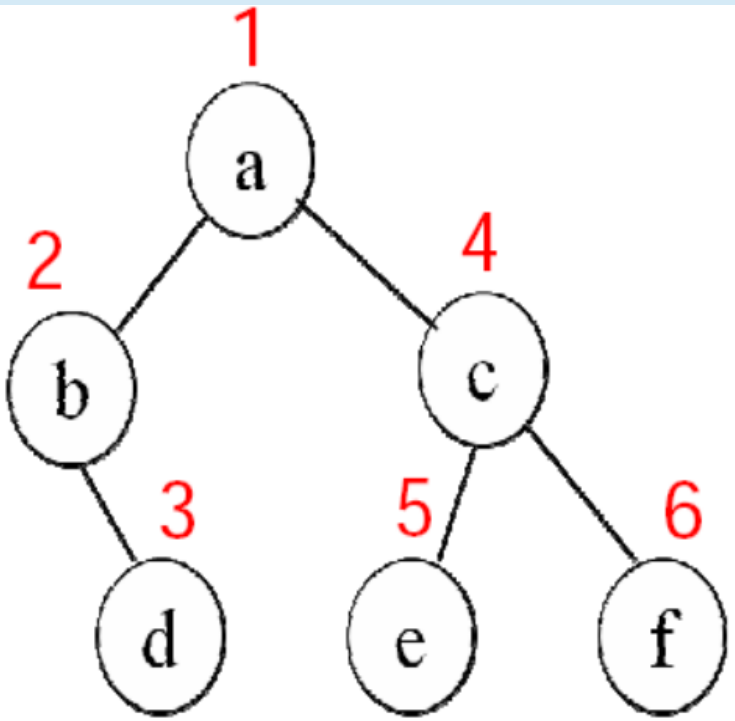
```
/* sub-árvore 'd' */
Arv* a1 = arv_cria('d',
   arv_criavazia(), arv_criavazia());
/* sub-árvore 'b' */
Arv* a2 = arv_cria('b',
   arv_criavazia(), a1);
/* sub-árvore 'e' */
Arv* a3 = arv_cria('e',
   arv_criavazia(), arv_criavazia());
/* sub-árvore 'f' */
Arv* a4 = arv_cria('f',
   arv_criavazia(), arv_criavazia());
/* sub-árvore 'c' */
Arv* a5 = arv_cria('c', a3, a4);
/* Árvore 'a' */
Arv* a = arv_cria('a', a2, a5);
```

Exemplo de impressão: Árvores Binárias




```
void arv_imprime (Arv* a) {
  if (!arv_vazia(a)) {
     printf("%c ", a->info);
     arv_imprime(a->esq);
     arv_imprime(a->dir);
   }
}
```

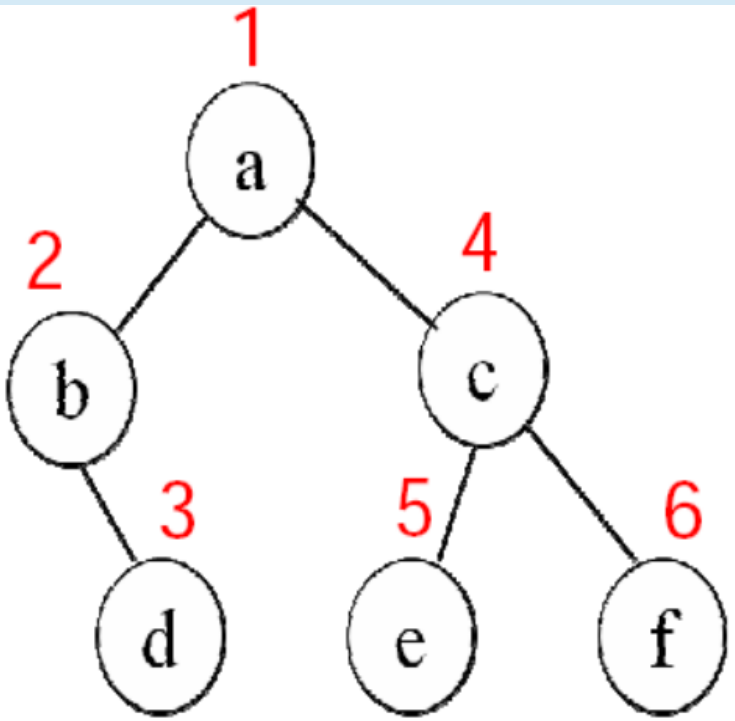
Exemplo de impressão: Árvores Binárias




```
void arv_imprime (Arv* a) {
  if (!arv_vazia(a)) {
     printf("%c ", a->info);
     arv_imprime(a->esq);
     arv_imprime(a->dir);
   }
}
```

O resultado da impressão é

Exemplo de impressão: Árvores Binárias

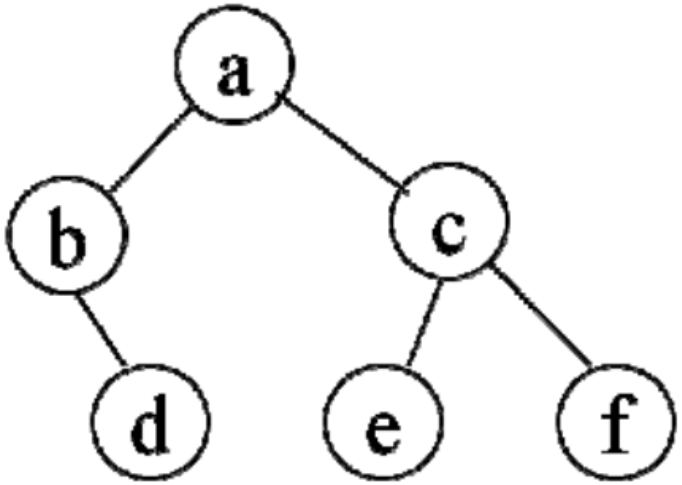



```
void arv_imprime (Arv* a) {
  if (!arv_vazia(a)) {
     printf("%c ", a->info);
     arv_imprime(a->esq);
     arv_imprime(a->dir);
   }
}
```

O resultado da impressão é a b d c e f

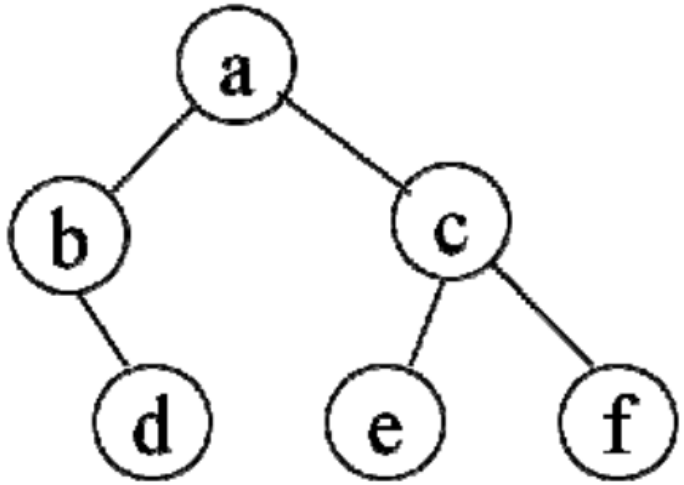
Exemplo inserindo elementos: Árvores Binárias

- Dada a árvore

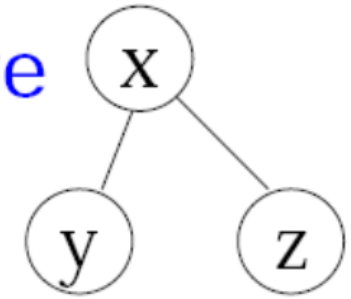


Exemplo inserindo elementos: Árvores Binárias

- Dada a árvore



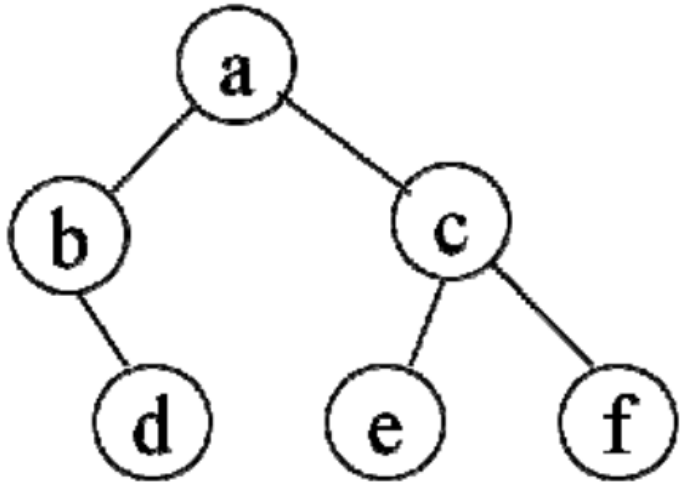
- Inserir a sub-árvore



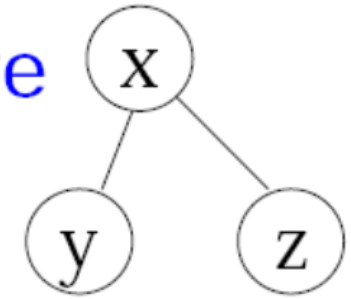
à esquerda do nó b.

Exemplo inserindo elementos: Árvores Binárias

- Dada a árvore



- Inserir a sub-árvore

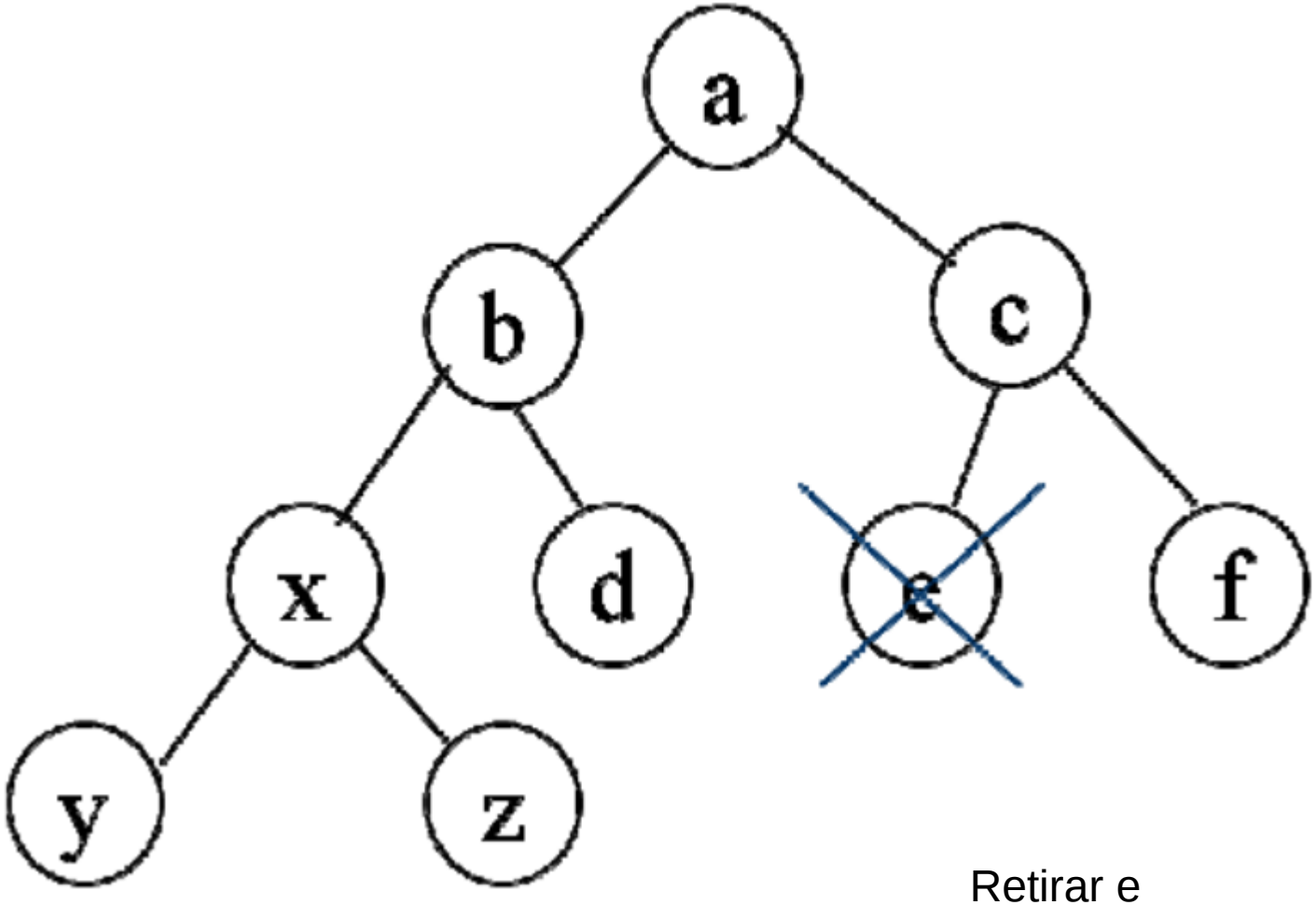


à esquerda do nó b.

```
a->esq->esq = arv_cria('x',

        arv_cria('y',
                arv_criavazia(),
                arv_criavazia()),

        arv_cria('z',
                arv_criavazia(),
                arv_criavazia())
                                );
```

Exemplo retirando elementos: Árvores Binárias

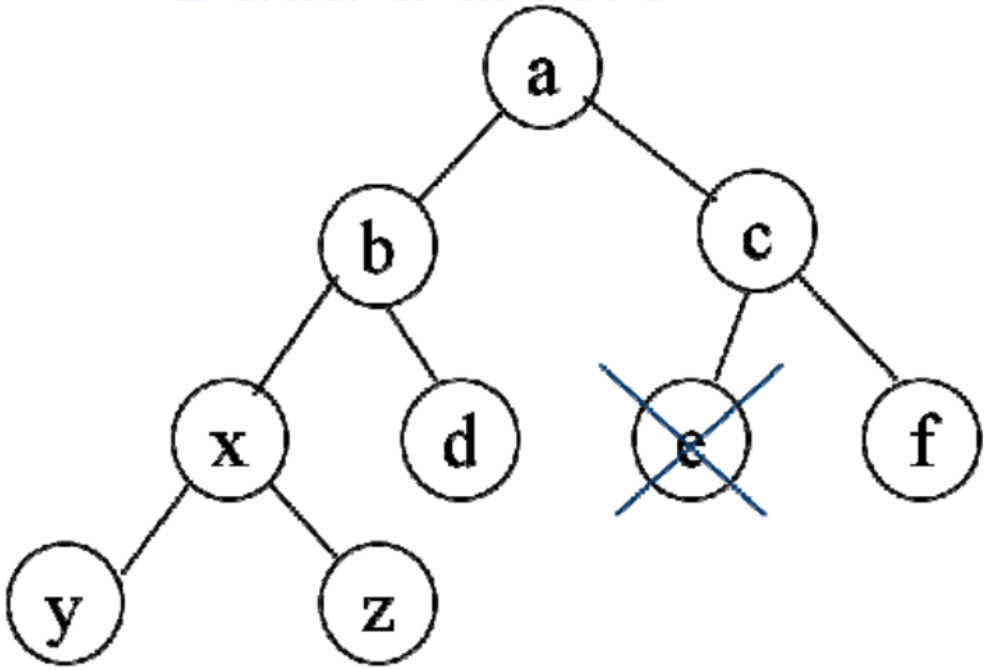
- Dada a árvore



Retirar e

Exemplo retirando elementos: Árvores Binárias

- Dada a árvore



Retirar e

```
a->dir->esq =
        arv_libera (a->dir->esq);
```

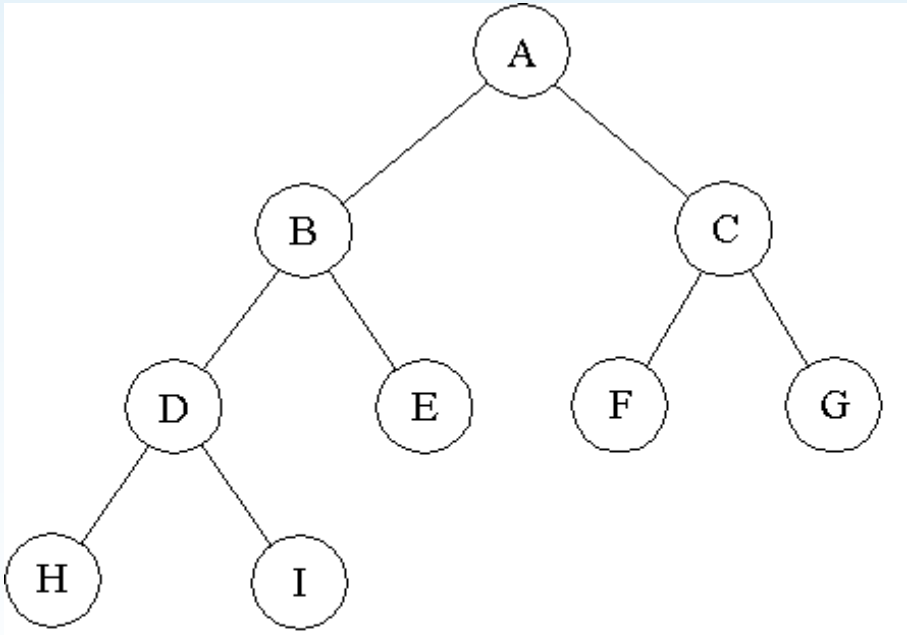
Ordens de Percurso em Árvores Binárias

- **Pré-ordem:**

  raiz, sae, sad

- **Ordem Simétrica:**

- **Pós-ordem:**



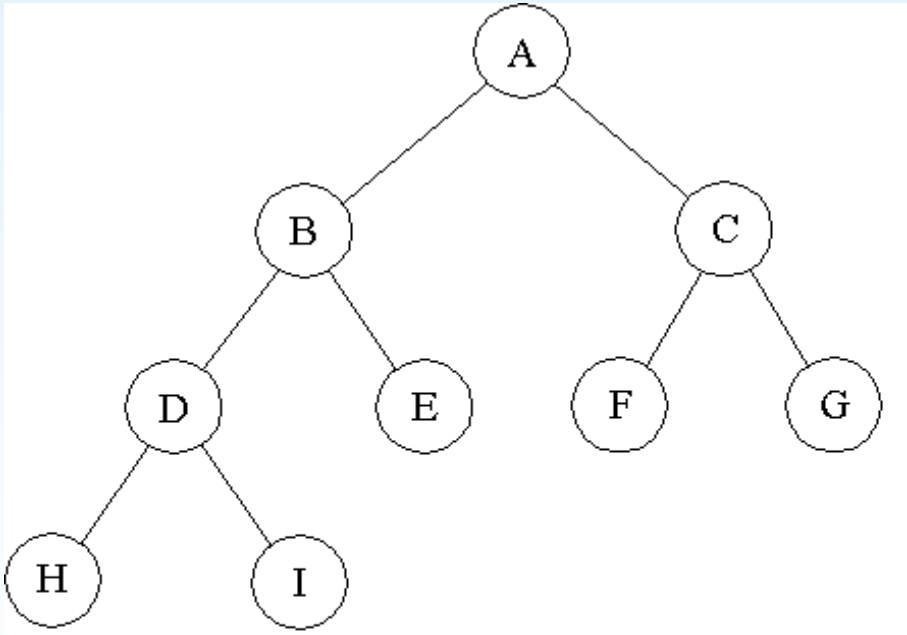
Ordens de Percurso em Árvores Binárias

- **Pré-ordem:** ABDHIECFG

  raiz, sae, sad

- **Ordem Simétrica:**

- **Pós-ordem:**



Ordens de Percurso em Árvores Binárias

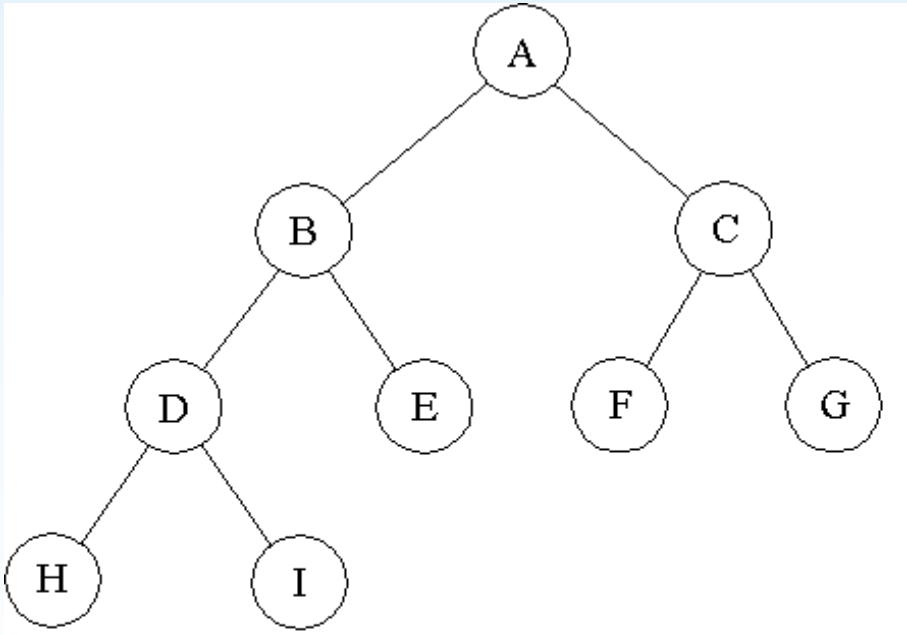
- **Pré-ordem:** ABDHIECFG

  raiz, sae, sad

- **Ordem Simétrica:**

  sae, raiz, sad

- **Pós-ordem:**



Ordens de Percurso em Árvores Binárias

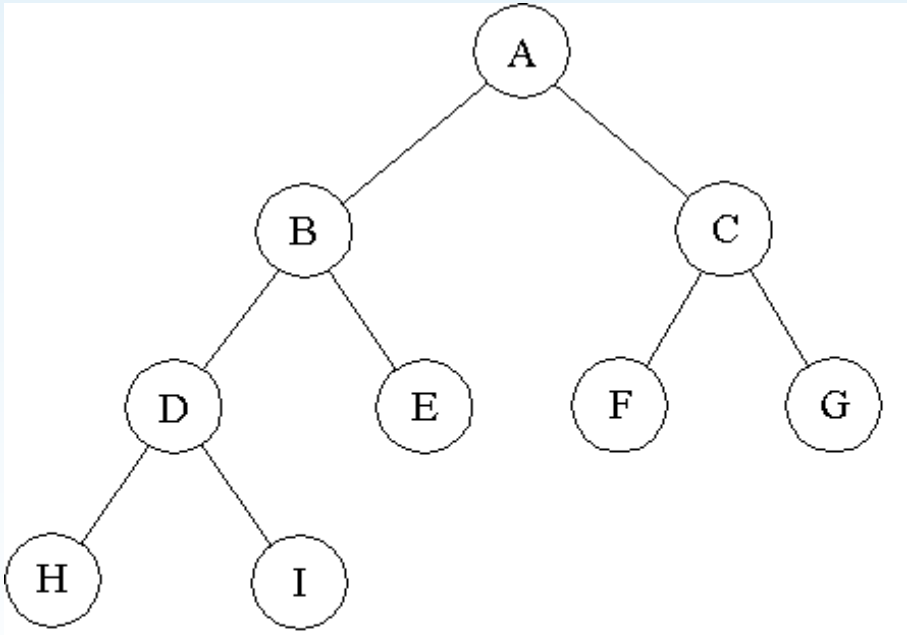
- **Pré-ordem:** ABDHIECFG

  raiz, sae, sad

- **Ordem Simétrica:** HDIBEAFCG

  sae, raiz, sad

- **Pós-ordem:**



Ordens de Percurso em Árvores Binárias

- **Pré-ordem:** ABDHIECFG

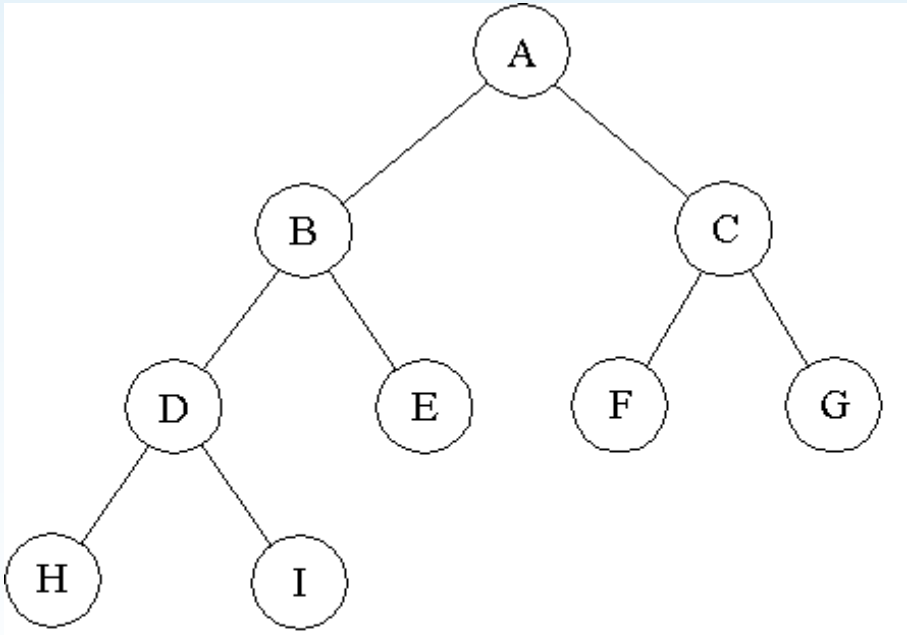
  raiz, sae, sad

- **Ordem Simétrica:** HDIBEAFCG

  sae, raiz, sad

- **Pós-ordem:**

  sae, sad, raiz



Ordens de Percurso em Árvores Binárias

- **Pré-ordem:** ABDHIECFG

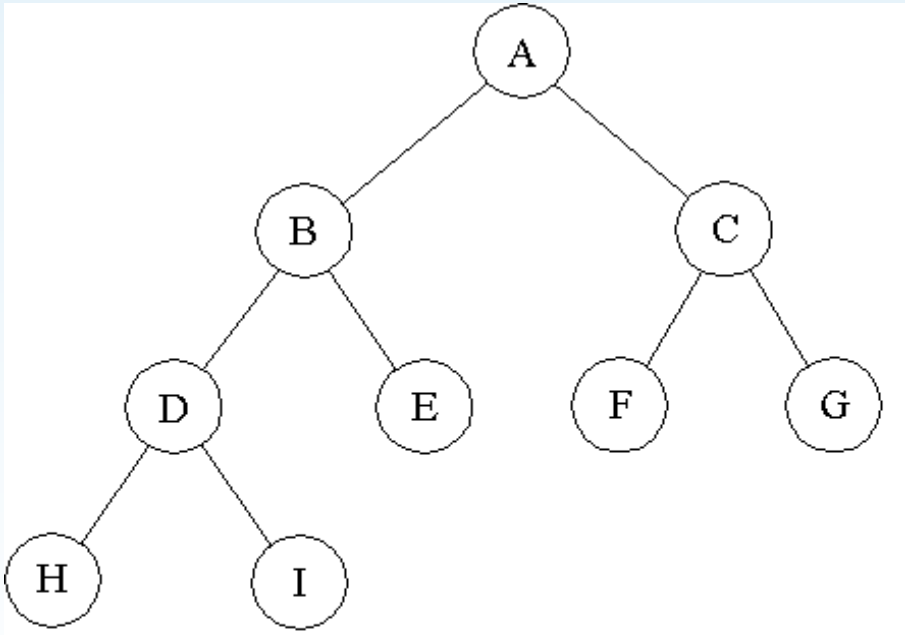
  raiz, sae, sad

- **Ordem Simétrica:** HDIBEAFCG

  sae, raiz, sad

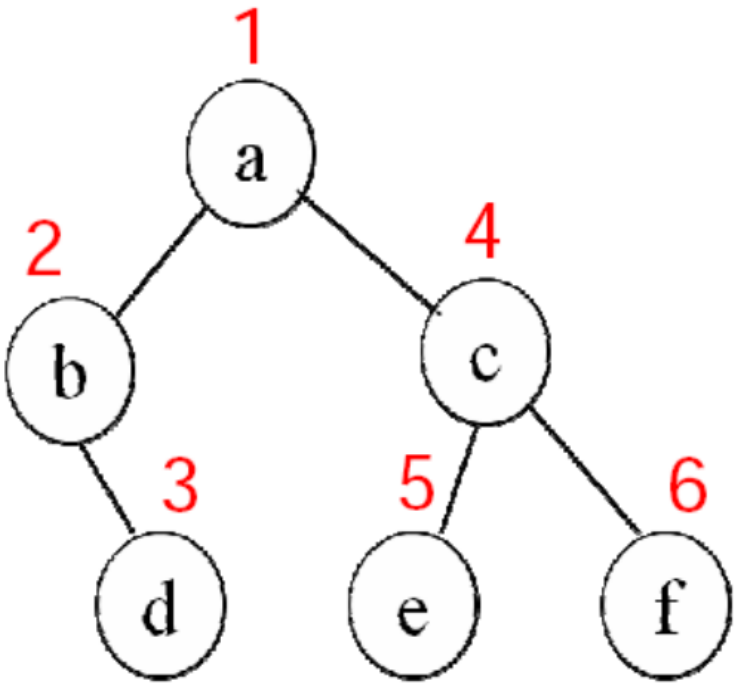
- **Pós-ordem:** HIDEBFGCA

  sae, sad, raiz



Exemplo: Ordens de Percurso em Árvores Binárias

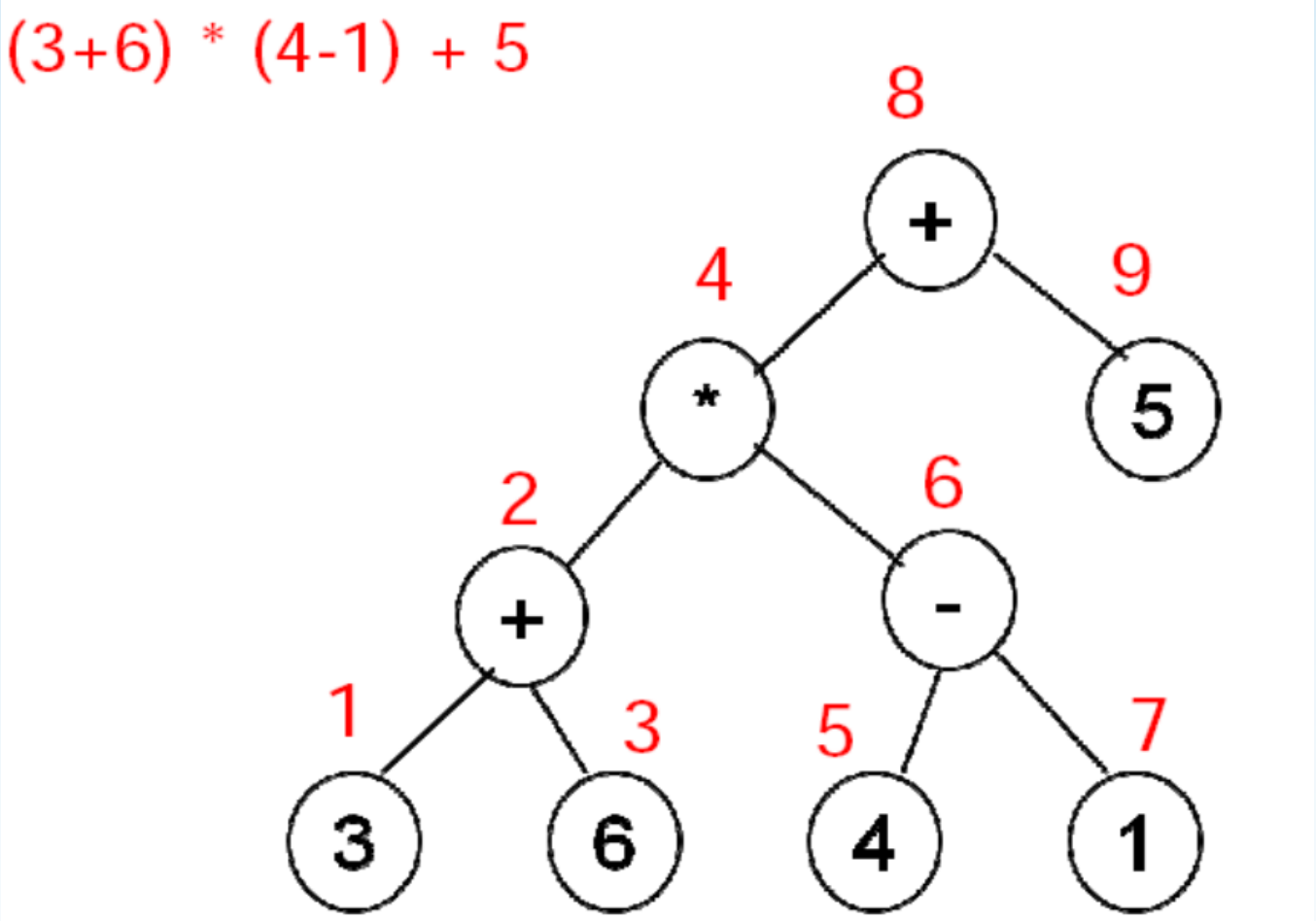
- **Pré-ordem:** Exemplo: função `arv_imprime(Arv* a)`



- **a b d c e f**

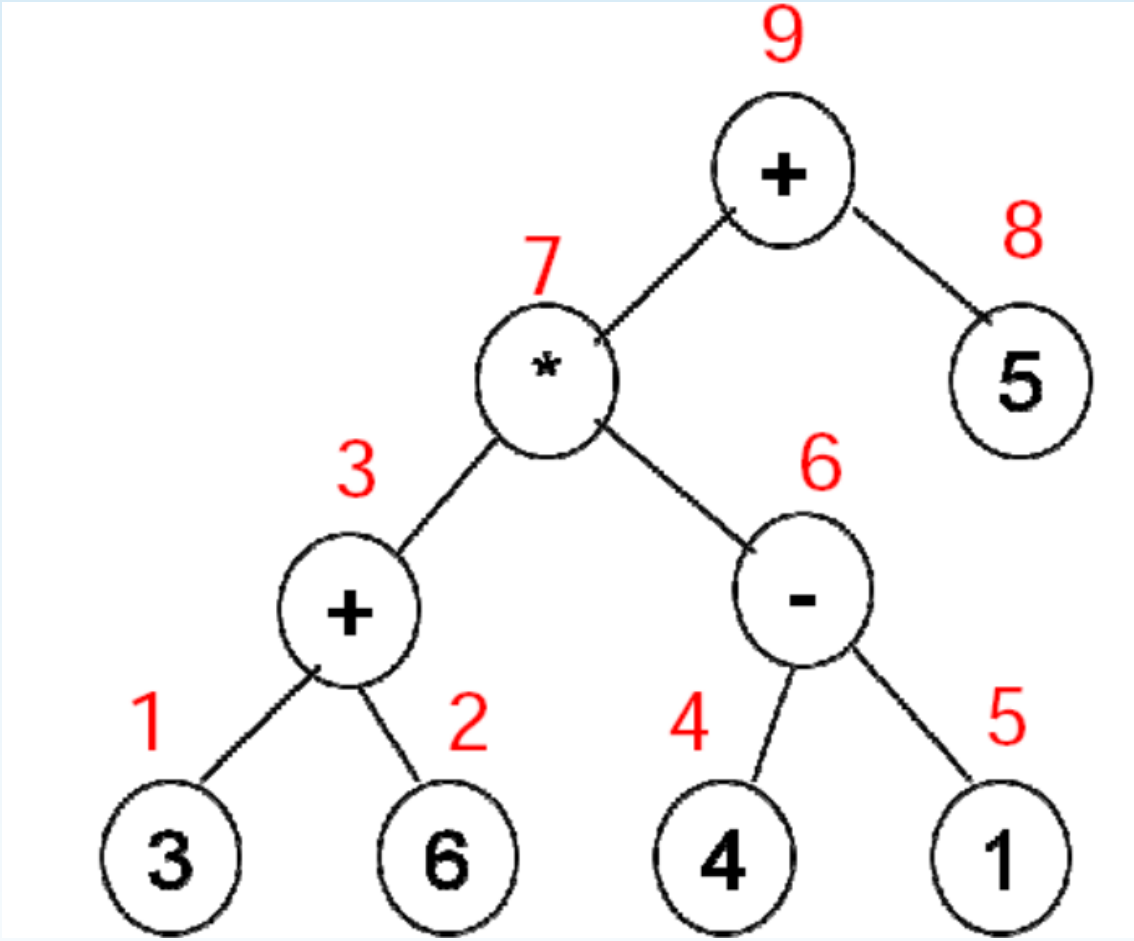
Exemplo: Ordens de Percurso em Árvores Binárias

- **Simétrica:** (3+6) * (4-1) + 5



Exemplo: Ordens de Percurso em Árvores Binárias

- **Pós-ordem:**



3 6 + 4 1 - * 5 +

Exemplo: Impressão dos nós em Árvores Binárias

- **Pós-ordem**

```
imprime(a->esq);                    /* mostra sae */
imprime(a->dir);                    /* mostra sad */
printf("%c ", a->info);          /* mostra raiz */
```

Outras definições sobre Árvores

- **Árvore é dita cheia, se todos os seus nós internos possuem 2 sub-árvores associadas, e todos os nós folhas estão no último nível.**

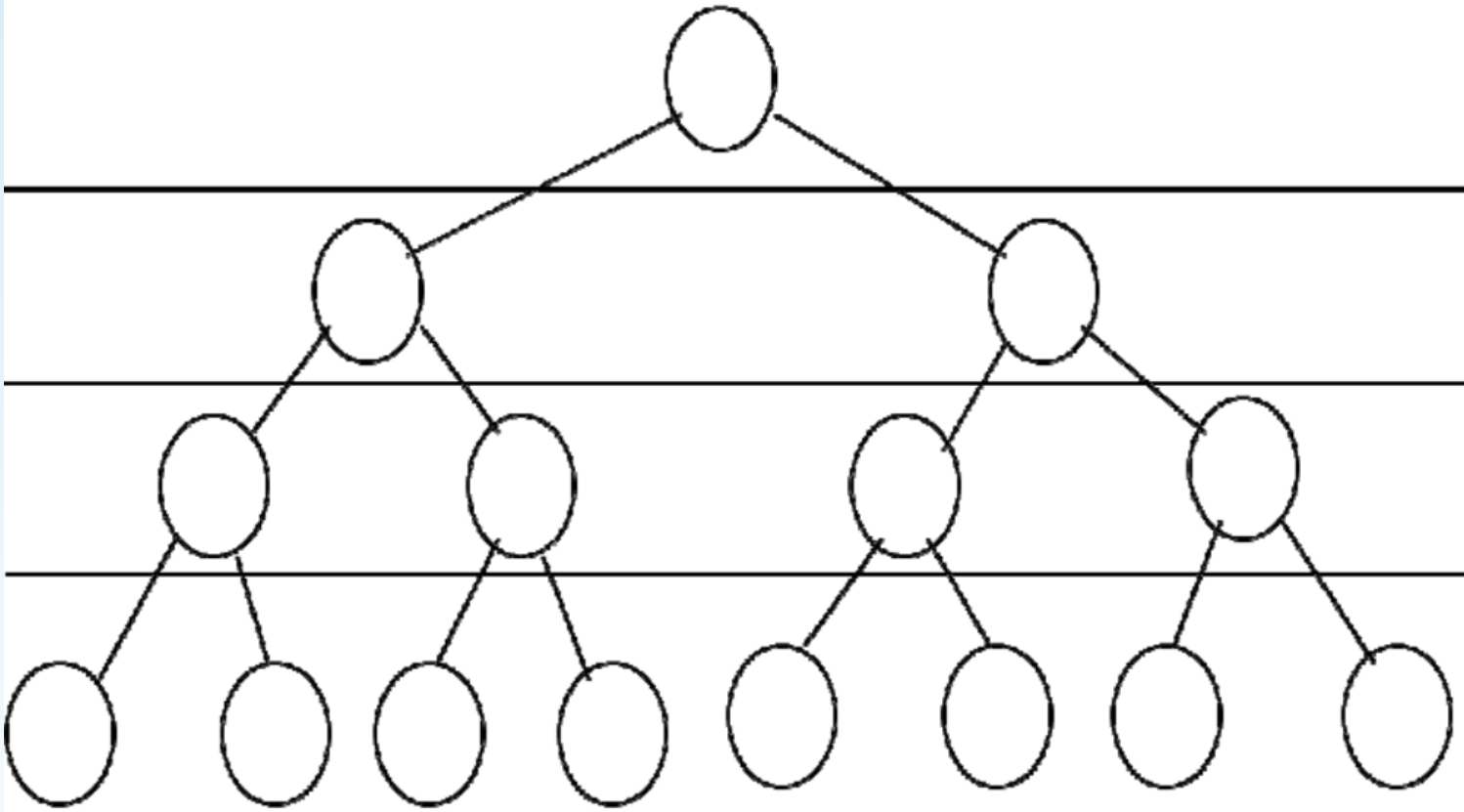
Outras definições sobre Árvores

- **Árvore é dita cheia, se todos os seus nós internos possuem 2 sub-árvores associadas, e todos os nós folhas estão no último nível.**
- **O número total de nós de uma árvore cheia é dado por** $2^{h+1} - 1$

Outras definições sobre Árvores

- **Árvore é dita cheia, se todos os seus nós internos possuem 2 sub-árvores associadas, e todos os nós folhas estão no último nível.**
- **O número total de nós de uma árvore cheia é dado por** $2^{h+1} - 1$
- **Uma árvore binária cheia, com n nós, tem uma altura proporcional à** $\log n$

Exemplo: Árvore Binária cheia



Nivel 0 : 2^0
= 1 nó

Nivel 1: 2^1
= 2 nós

Nivel 2:2^2
= 4 nós

Nivel 3:2^3
= 8 nós

Número de nós = $2^{3+1} - 1 = 15$

Outras definições sobre Árvores

- **Árvore é dita degenerada, se todos os seus nós internos possuem uma única sub-árvore associada.**
- **A estrutura hierárquica se degenera em uma estrutura linear.**

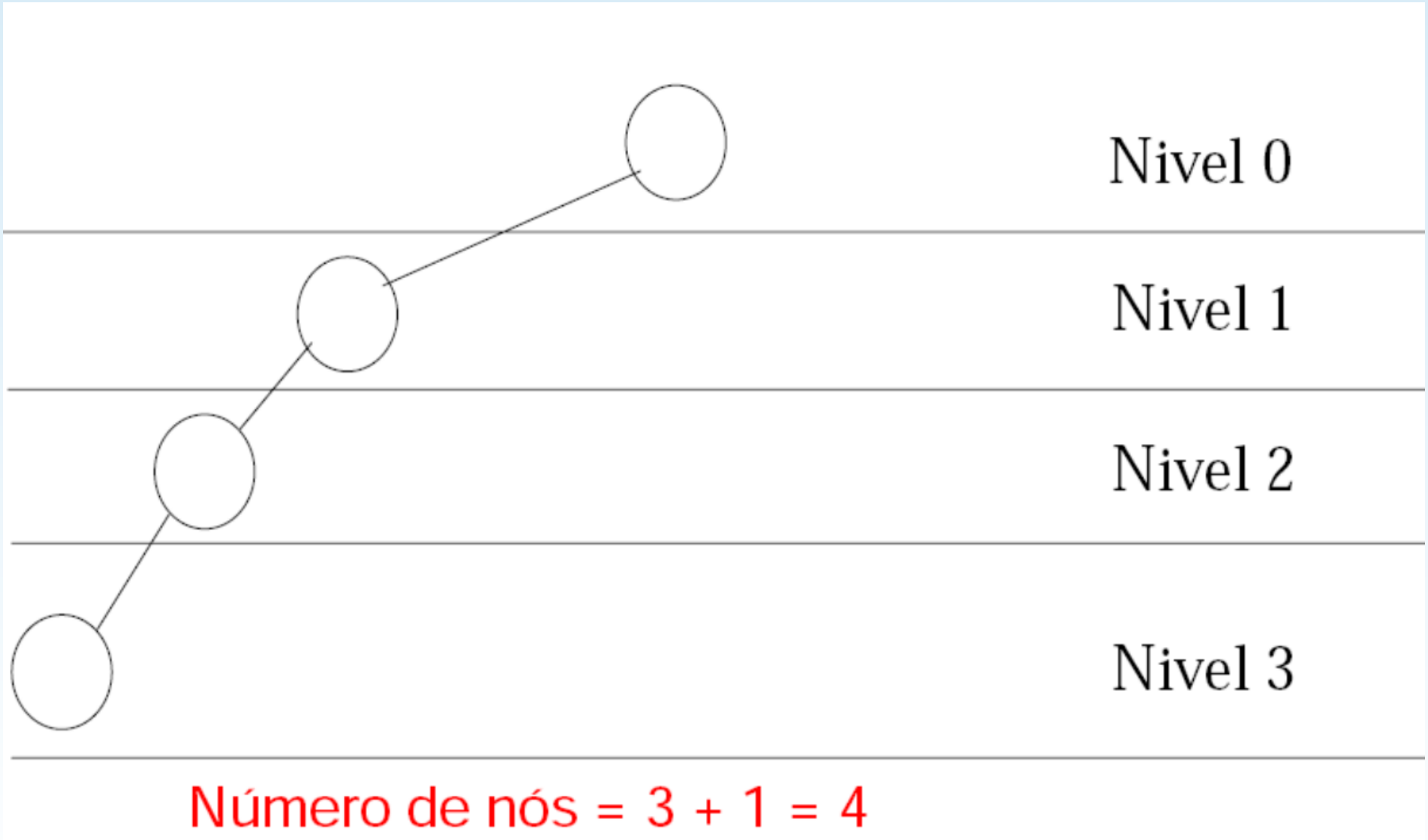
Outras definições sobre Árvores

- **Árvore é dita degenerada, se todos os seus nós internos possuem uma única sub-árvore associada.**
- **A estrutura hierárquica se degenera em uma estrutura linear.**
- **Uma árvore degenerada de altura h tem h+1 nós.**
- **A altura de uma árvore degenerada com n nós é proporcional a n.**

Outras definições sobre Árvores

- **Árvore é dita degenerada, se todos os seus nós internos possuem uma única sub-árvore associada.**
- **A estrutura hierárquica se degenera em uma estrutura linear.**
- **Uma árvore degenerada de altura h tem h+1 nós.**
- **A altura de uma árvore degenerada com n nós é proporcional a n.**

Exemplo: Árvore Binária degenerada



Outras definições sobre Árvores

- A altura de uma Árvore Binária com um único nó é 0.
- A altura de uma árvore vazia é -1.
- A raiz está no nível 0, e seus filhos no nível 1, e assim por diante.
- O último nível é o h (altura da árvore).

Outras definições sobre Árvores

- **A altura de uma árvore é uma medida de avaliação da eficiência com que visitamos os nós de uma árvore.**
- **Uma árvore binária com n nós temuma altura mínima proporcional a log n (e.g. cheia), e uma altura máxima proporcional a n (e.g. degenerada)**

Exemplo: Função em C para calcular altura de uma árvore binária

```
int  arv_altura (Arv* a) {
   if (arv_vazia (a))
      return -1;
   else
      return 1 + max2(arv_altura(a->esq),
                      arv_altura(a->dir));
```

- **Sendo que max2 é**

```
int max2 (int a, int b) {
  return (a>b)? a:b;
}
```

Referências Bibliográficas

- Cormen, T. & Leiserson, C. & Rivest, R. & Stein, C. *Introduction to Algorithms.* MIT Press. 2009.

- Sedgewick, R. *Algorithms in C* (Parts 1-4, Part 5), 2nd ed., Addison Wesley, 1997.

- Rangel, J.; Cerqueira, R.& Celes, W. *Estruturas de dados: uma introd. com téc. de programação em C*, Elsevier, 2004.

- Tenenbaum, A.; Langsam, Y. & Augenstein, M. *Estruturas de dados usando C*, Makron Books, 1995.

- Ziviani, N. *Projeto de Algoritmos: com implementações em Pascal e C,* 2a. ed., Thompson, 2007.