Métodos Computacionais

Árvores

Árvores

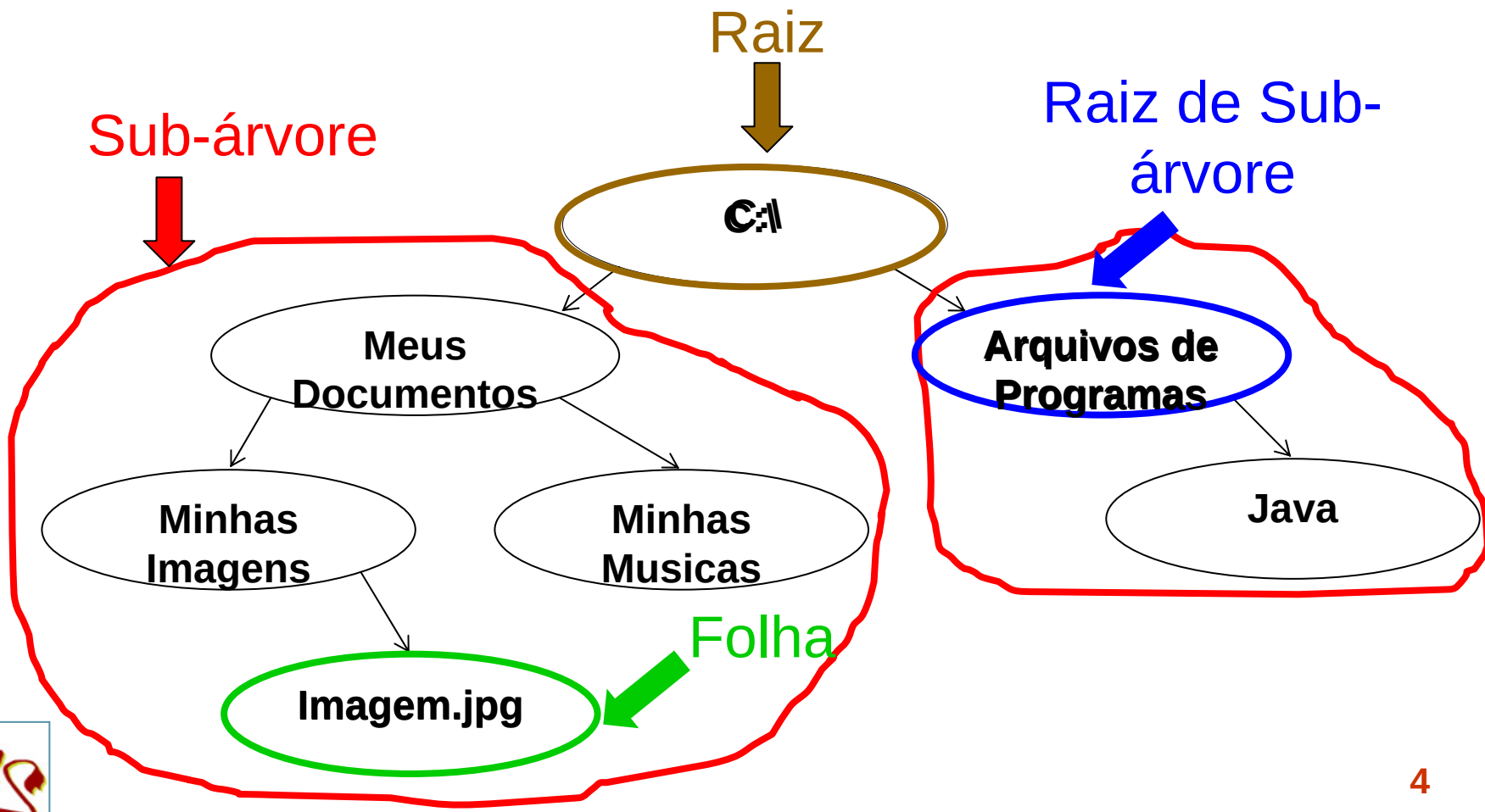
- Vetores e Listas são ótimos para representar estrutura de dados lineares, mas não para modelar dados hierárquicos
- Exemplos de dados hierárquicos: sistema de arquivos de um computador
- Uma árvore é uma estrutura de dados recursiva que permite representar dados dispostos de maneira hierárquica.

Definição de Árvore

- Uma árvore é composta por um conjunto de nós
- Existe um nó raiz que contém zero ou mais sub-árvores, cujas raízes(nós internos) estão ligadas diretamente à raiz.
- Os nós que não têm filhos são chamados de folhas (nós externos)

Exemplo de Árvore: Sistema de Arquivos

- Exemplo de árvore: Sistema de Arquivos

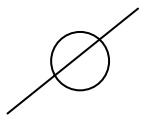


Principais tipos de árvores

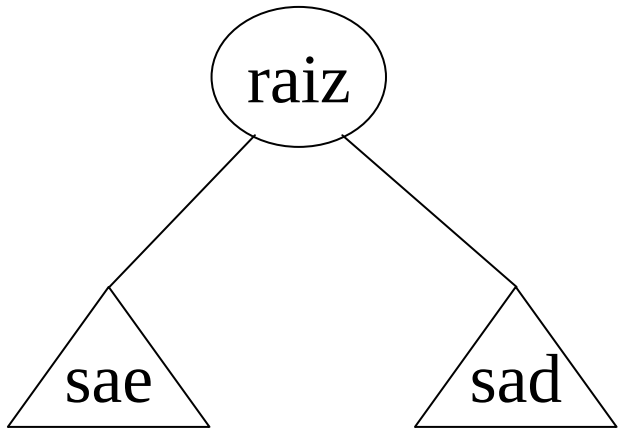
- Árvores binárias
  - Cada nó tem no máximo dois filhos
- Árvore com número variável de filhos
  - Cada nó pode ter vários filhos

Definindo Árvores Binárias

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

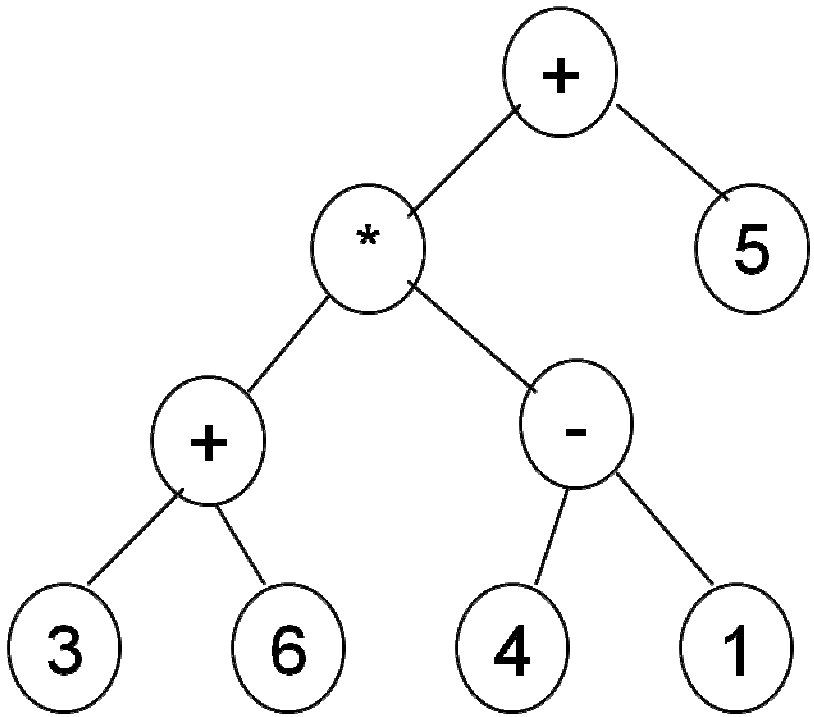


Ou



Exemplo de Uso de Árvores Binárias

- Avaliação de expressões aritméticas.
  - Folhas representando operandos e nós internos operadores
  - Ex: (3+6)*(4-1)+5



Interface do Tipo Árvore

- Criar uma árvore vazia
- Criar uma árvore não vazia
- Verificar se a árvore está vazia
- Verificar se um elemento pertence a árvore
- Liberar uma árvore
- Imprimir os nós da árvore

- Em C:

```
/* 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);
```

Implementação de Árvores Binárias

- 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

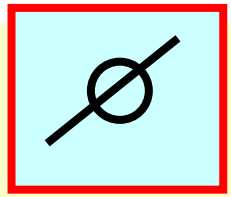
- Em C:

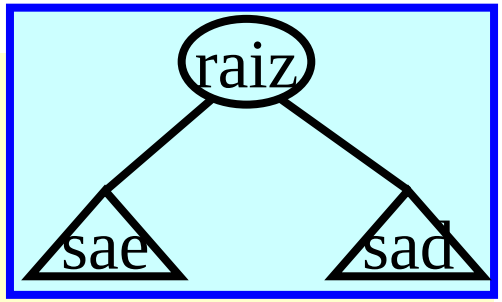
```
struct arv {
  char info;
  struct arv* esq;
  struct arv* dir;
} ;
```

Implementação de Árvores Binárias

- Funções de Criação

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






```
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;
}
```

Duas funções representam os 2 casos da definição recursiva de árvore binária

Implementação de Árvores Binárias

- Função que verifica se elemento pertence a 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);
}
```

Equivalente a:

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

Implementação de Árvores Binárias

- 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

- Função que verifica se uma árvore está vazia

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

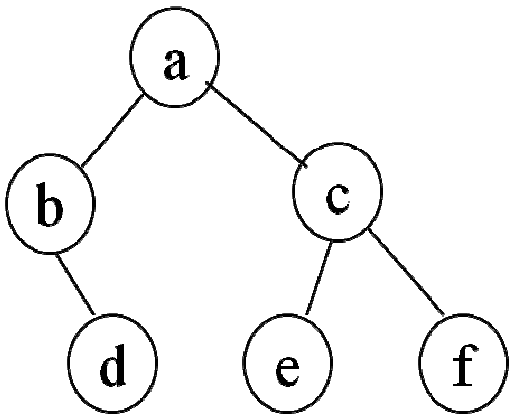
Implementação de Árvores Binárias

- Função que imprime elementos da 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 */
   }
}
```

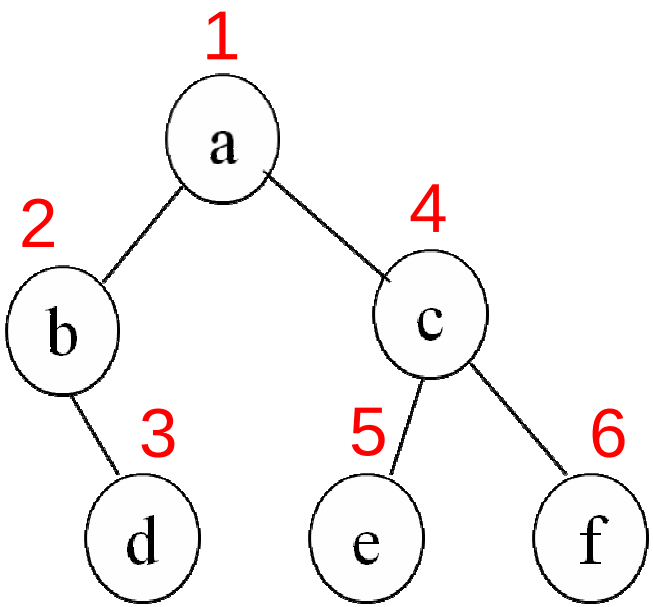
Implementação recursiva: primeiro visita a raiz, depois a sub-árvore da esquerda, para finalmente visitar a da direita

Exemplo de Criação de uma Árvore




```
/* 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 Uso da Impressão

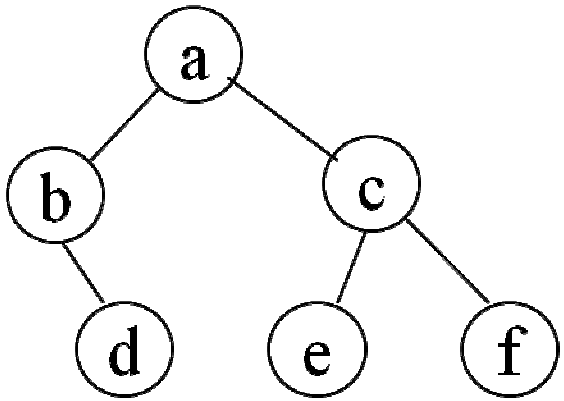



```
void arv_imprime (Arv* a) {
  if (!arv_vazia(a)) {
     printf("%c ", a->info);
     arv_imprime(a->esq);
     arv_imprime(a->dir);
   }
}
```

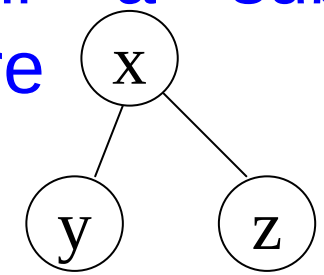
O resultado da impressão é a b d c e f

Inserindo uma Sub-árvore Numa Árvore

- Dada a árvore



- Inserir a sub-árvore



à esquerda do nó b.

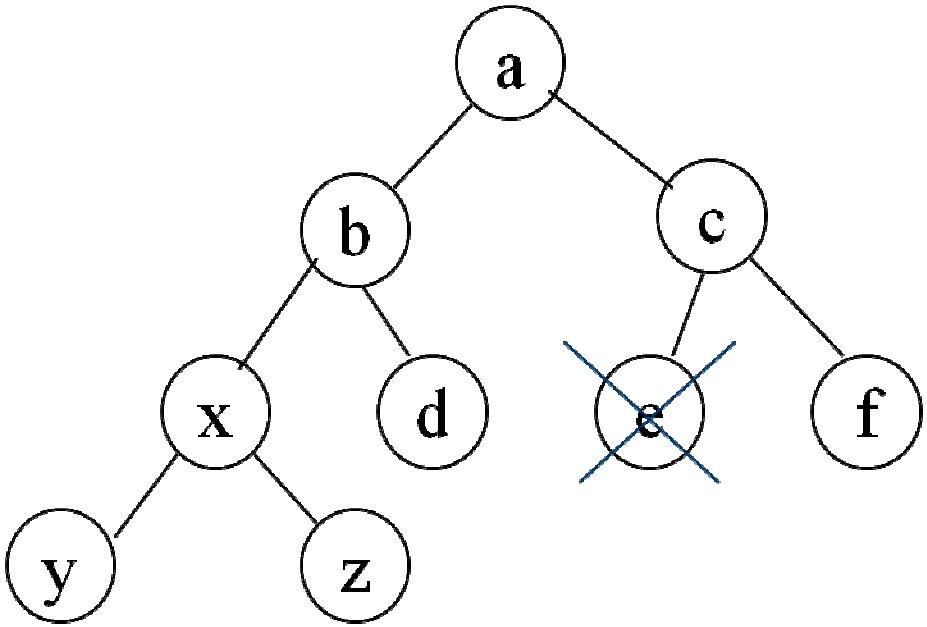
```
a->esq->esq = arv_cria('x',

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

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

Liberando uma Sub-árvore Numa Árvore

- Dada a árvore



- Liberar a sub-árvore e:

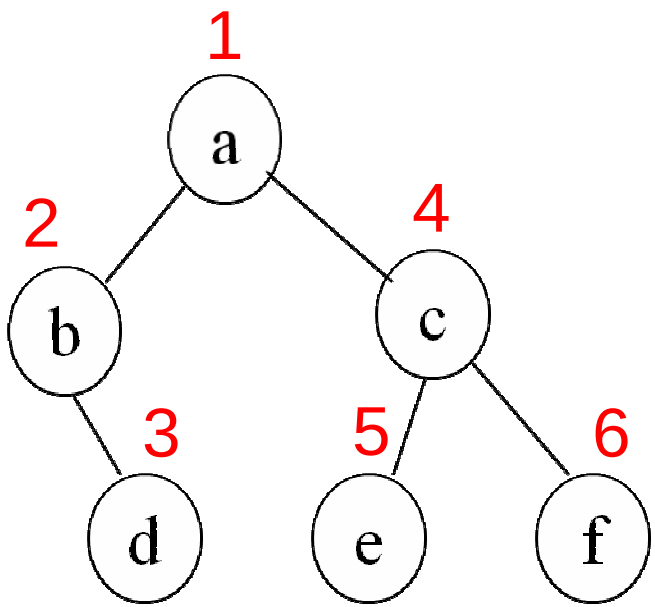
```
a->dir->esq =
        arv_libera (a->dir->esq);
```

Ordens de Percurso em Árvores Binárias

- Muitas operações em árvores envolvem o percurso de todas as sub-árvores, com a execução de alguma ação de tratamento em cada nó.
- Dependendo da aplicação, uma determinada ordem de percurso da árvore é mais adequada do que uma outra
- É comum percorrer uma árvore em uma das seguintes ordens:
  - Pré-ordem: raiz,sae, sad
  - Ordem simétrica: sae, raiz, sad
  - Pós-ordem: sae, sad, raiz

Ordens de Percurso em Árvores Binárias

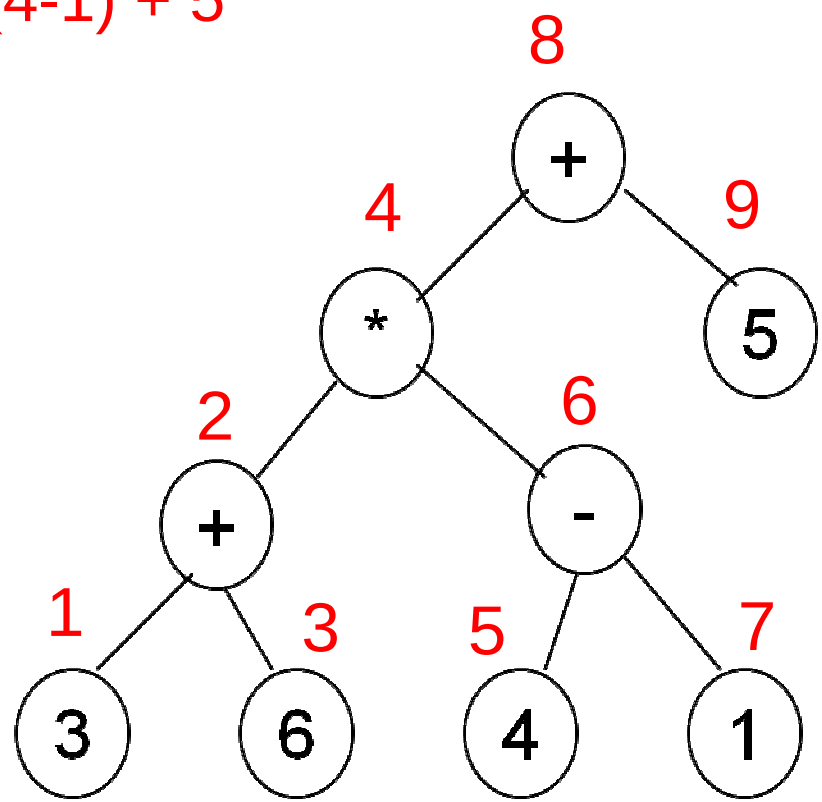
- Pré-Ordem
  - Exemplo: função **`arv_imprime(Arv* a)`**



Ordens de Percurso em Árvores Binárias

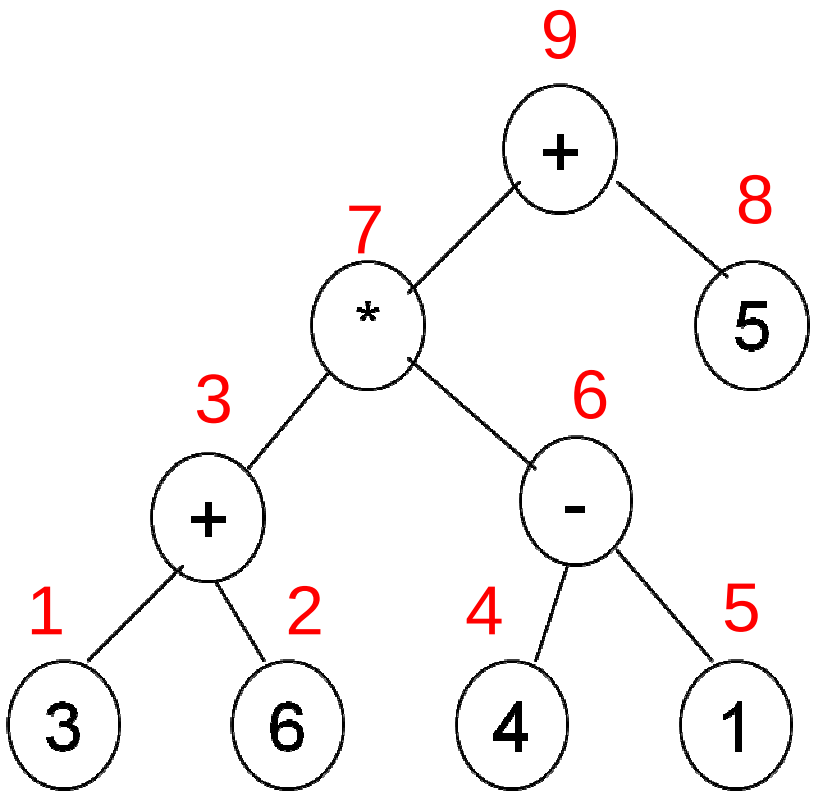
- Ordem Simétrica
  - Exemplo: função para avaliar expressões

(3+6) * (4-1) + 5



Ordens de Percurso em Árvores Binárias

- Pós-ordem
  - Exemplo: função para avaliar expressões pós-fixas 3 6 + 4 1 - * 5 +



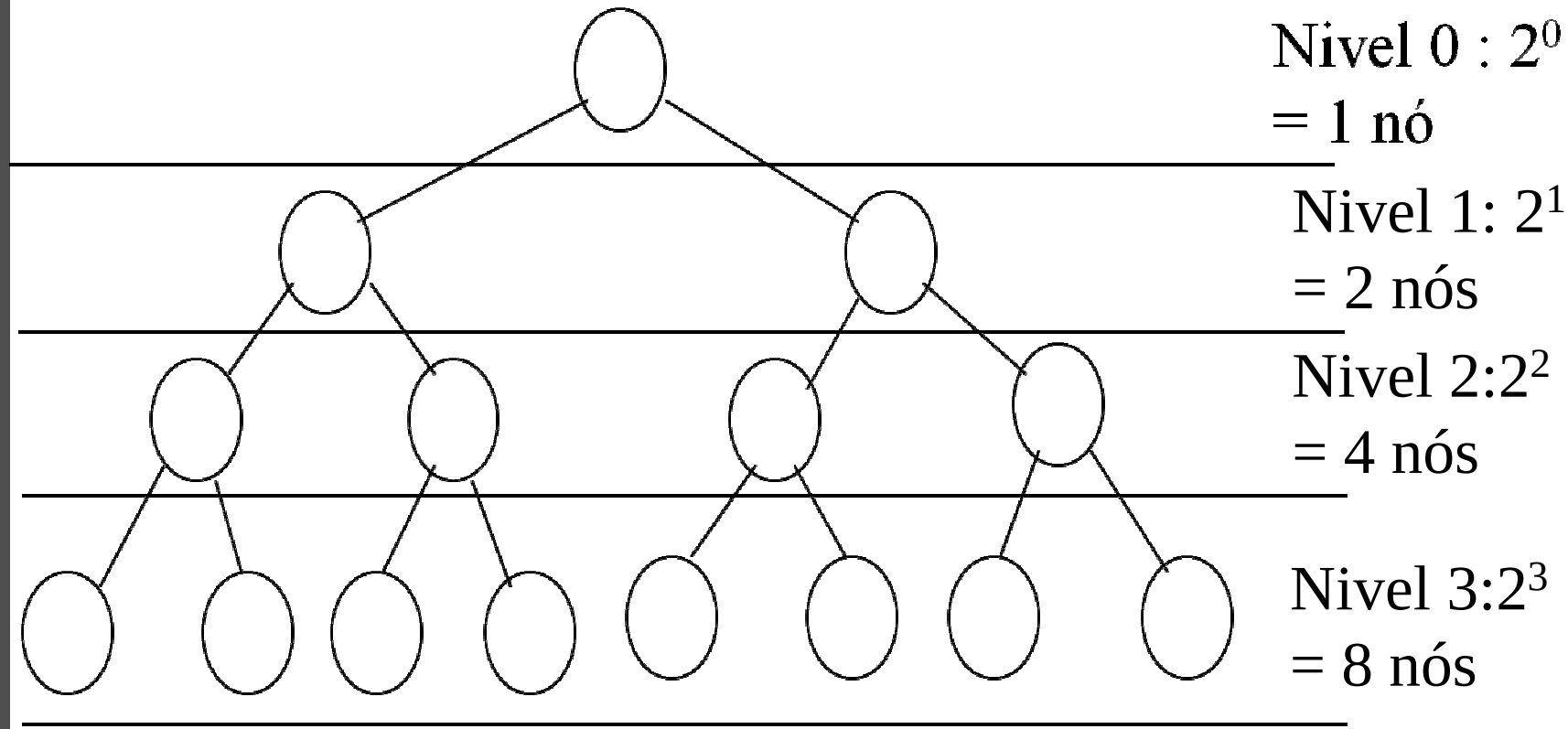
Altura de uma Árvore

- Propriedade Fundamental de Árvores
  - Só existe um caminho da raiz para qualquer nó
- Definição de Altura de Árvore
  - É o comprimento do caminho mais longo da raiz até uma das folhas.
    - A altura de uma árvore com um único nó é 0;
    - A altura da árvore vazia é –1;
    - A raiz está no nível 0 e seus filhos diretos no nível 1, e assim por diante;
    - O último nível é o *h* (que é a altura da árvore)

Árvore Cheia

- Árvore é dita cheia se todos os seus nós internos têm 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 a log n

Árvore Cheia

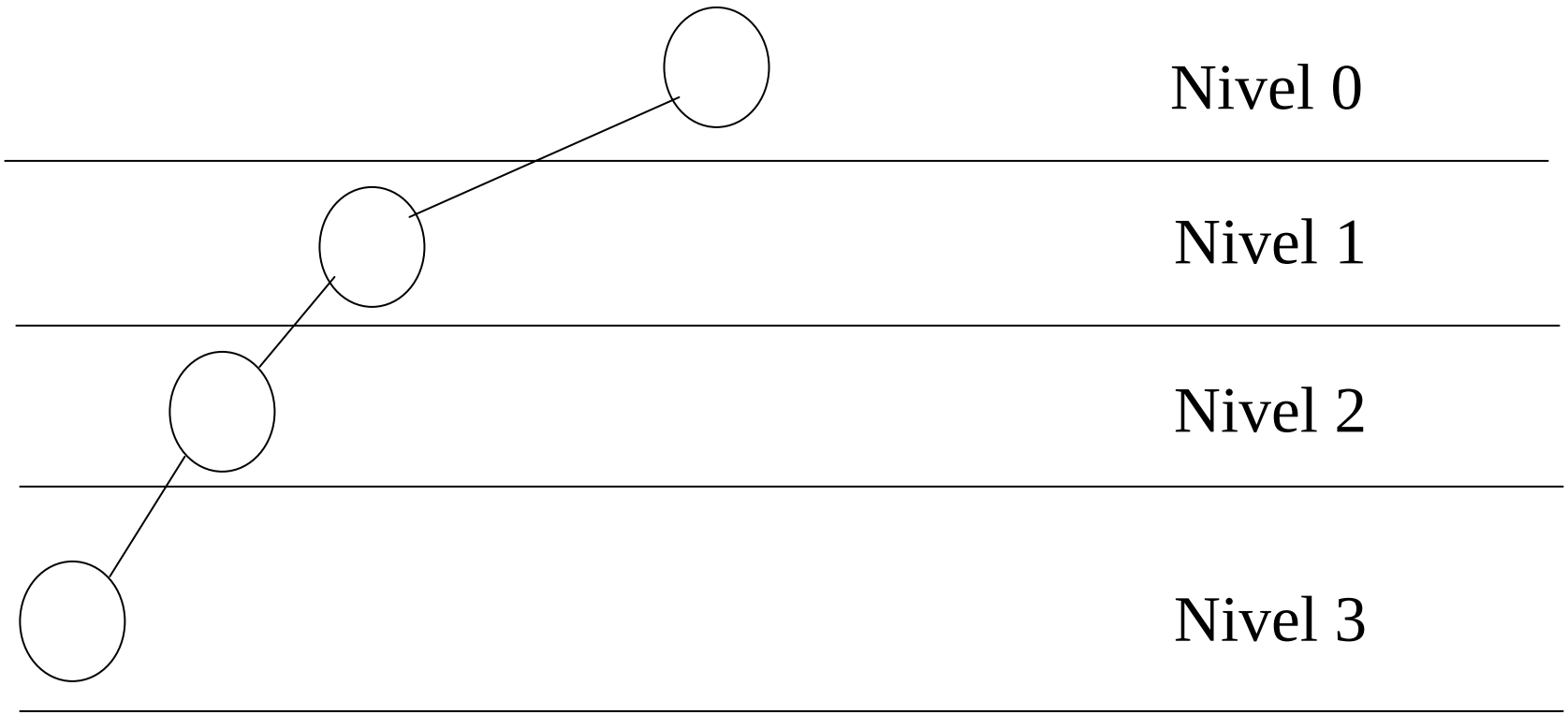


Número de nós = $2^{3+1} - 1 = 15$

Árvore Degenerada

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

Árvore Degenerada



Número de nós = 3 + 1 = 4

Para quê Calcular a Altura?

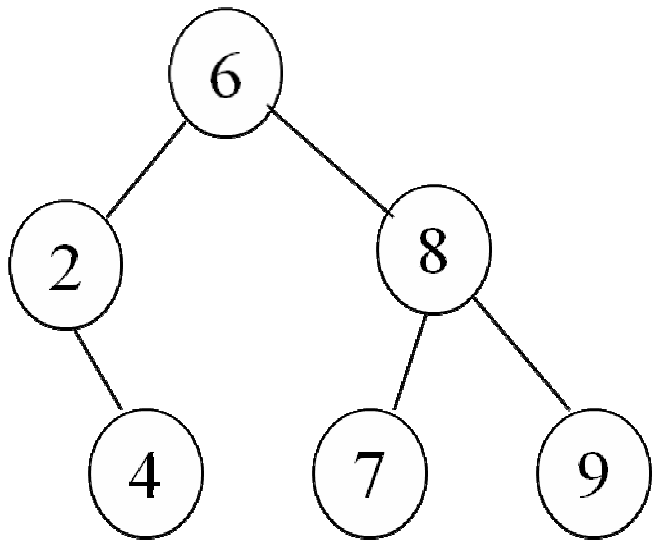
- A altura de uma árvore é uma medida de avaliação da eficiência com que visitamos os nós de uma árvore
- Uma árvore binária com *n* nós tem uma altura mínima proporcional a *log n* (caso a árvore seja cheia) e uma altura máxima proporcional a *n* (caso a árvore seja degenerada)

Para quê Calcular a Altura?

- A altura indica o esforço computacional necessário para alcançar qualquer nó da árvore;
- Em árvores binárias de busca, é importante manter a altura pequena, ou seja, manter a árvore com distribuição dos nós próxima à da árvore cheia.

Para quê Calcular a Altura?

- Exemplo de árvore binária de busca
  - Valor da raiz sempre maior que o da sub-árvore da esquerda e menor que o da sub-árvore da direita
  - Busca de elemento é otimizada



Função para Cálculo da Altura

- Função auxiliar para calcular o máximo entre dois inteiros

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

- Função recursiva para calcular altura

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