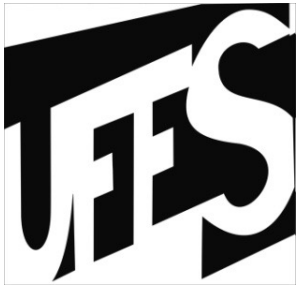


Estruturas de Dados
Árvores

Fontes Bibliográficas

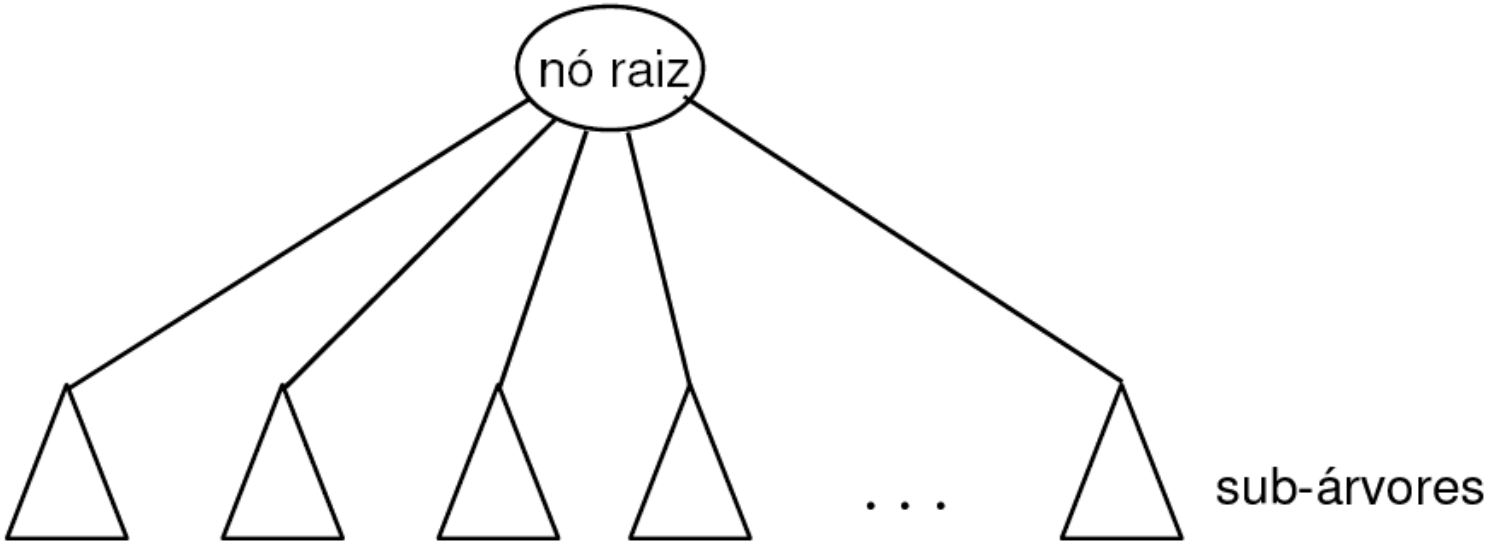
- Livros:
  - Introdução a Estruturas de Dados (Celes, Cerqueira e Rangel): Capítulo 13;
  - Projeto de Algoritmos (Nivio Ziviani): Capítulo 5;
  - Estruturas de Dados e seus Algoritmos (Szwarefiter, et. al): Capítulo 3;
  - Algorithms in C (Sedgewick): Capítulo 5;
- Slides baseados no material da PUC-Rio, disponível em http://www.inf.puc-rio.br/~inf1620/.

Introdução

- Estruturas estudadas até agora não são adequadas para representar dados que devem ser dispostos de maneira hierárquica
  - Ex., hierarquia de pastas
  - Árvore genealógica
- Árvores são estruturas adequadas para representação de hierarquias

Definição Recursiva de Árvore

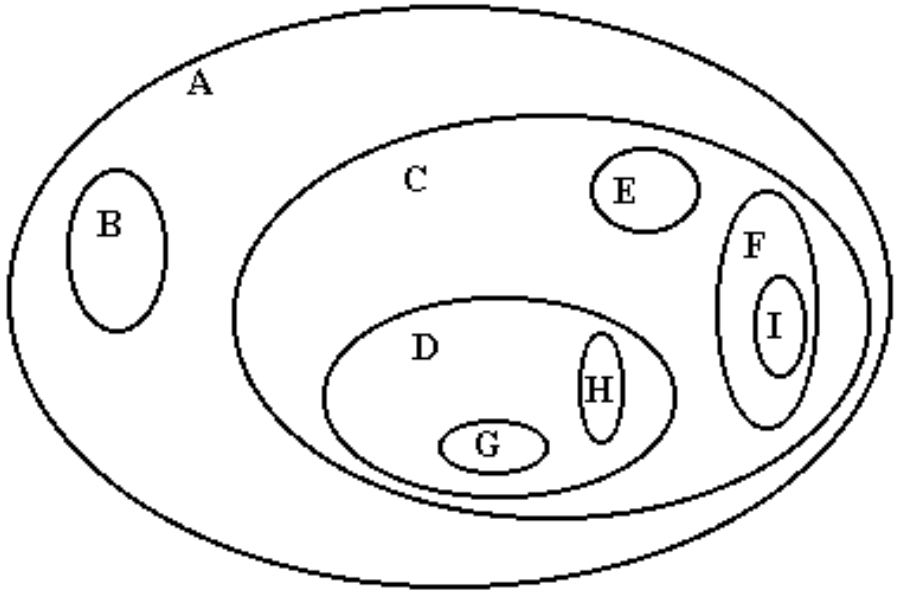
- Um conjunto de nós tal que:
  - existe um nó r, denominado *raiz,* com zero ou mais sub-árvores, cujas raízes estão ligadas a r
  - os nós raízes destas sub-árvores são os *filhos* de r
  - os *nós internos* da árvore são os nós com filhos
  - *as folhas* ou *nós externos* da árvore são os nós sem filhos



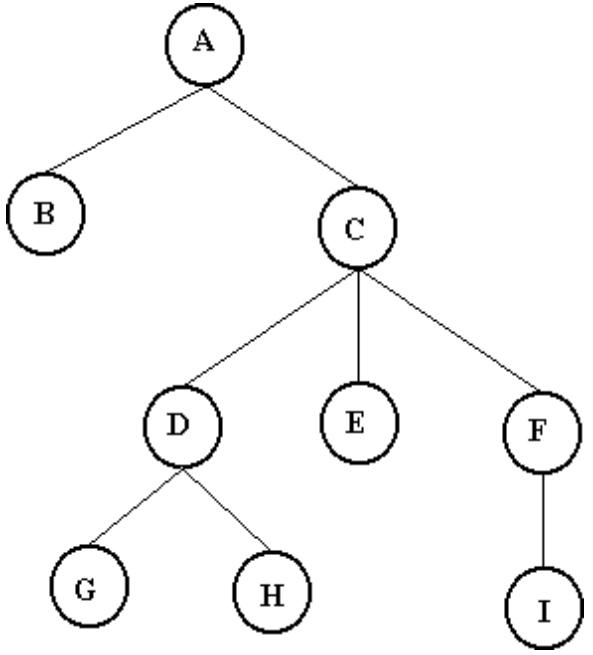
Formas de representação

- Representação por parênteses aninhados
  - (A (B) (C (D (G) (H)) (E) (F (I))))

Diagrama de Inclusão



Representação Hierárquica



Conceitos Básicos

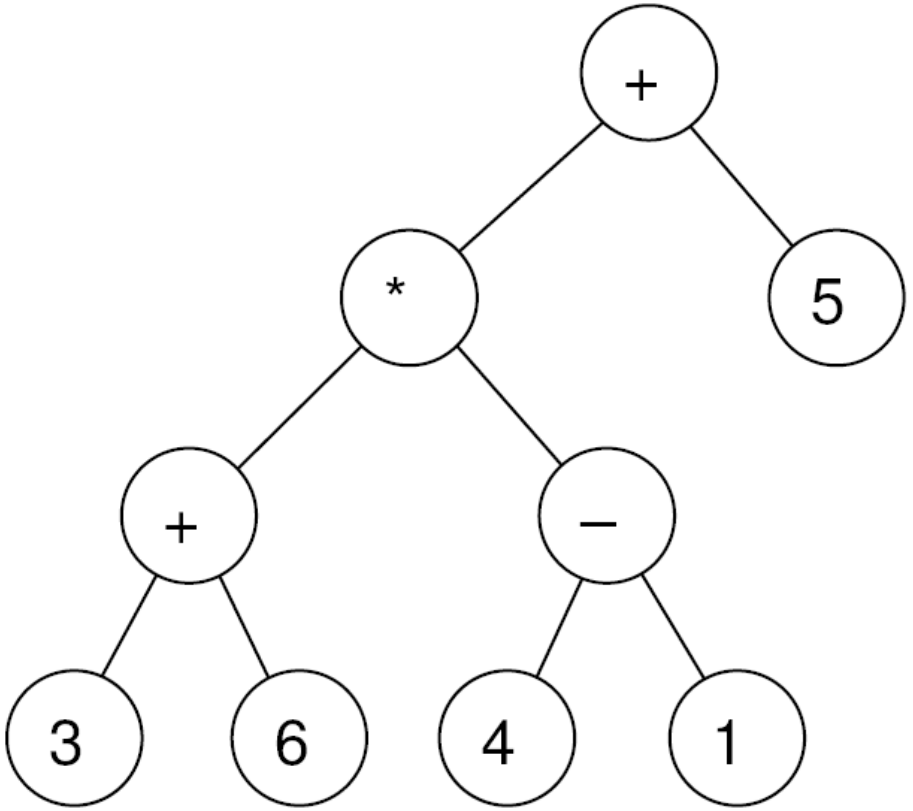
- Nós filhos, pais, tios, irmãos e avô
- Grau de saída (número de filhos de um nó)
- Nó folha (grau de saída nulo) e nó interior (grau de saída diferente de nulo)
- Grau de uma árvore (máximo grau de saída)
- Floresta (conjunto de zero ou mais árvores)

Conceitos Básicos (2)

- Caminho
  - Uma sequência de nós distintos v1, v2, ..., vk, tal que existe sempre entre nós consecutivos (isto é, entre v1 e v2, entre v2 e v3, ... , v(k-1) e vk) a relação "é filho de" ou "é pai de" é denominada um caminho na árvore.
- Comprimento do Caminho
  - Um caminho de vk vértices é obtido pela sequência de k-1 pares. O valor k-1 é o comprimento do caminho.
- Nível ou profundidade de um nó
  - número de nós do caminho da raiz até o nó.

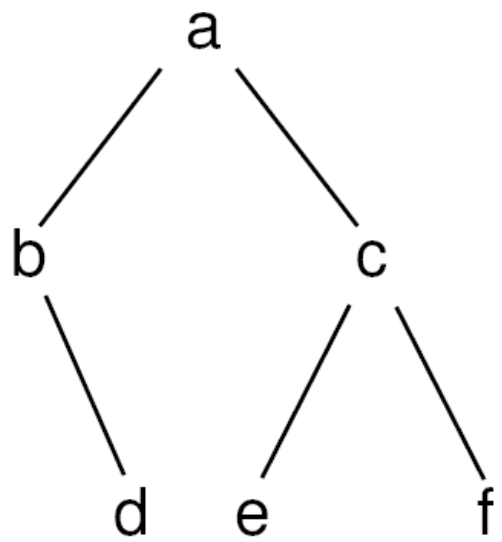
Exemplo

- Árvore binária representando expressões aritméticas binárias
  - Nós folhas representam os operandos
  - Nós internos representam os operadores
  - (3+6)*(4-1)+5



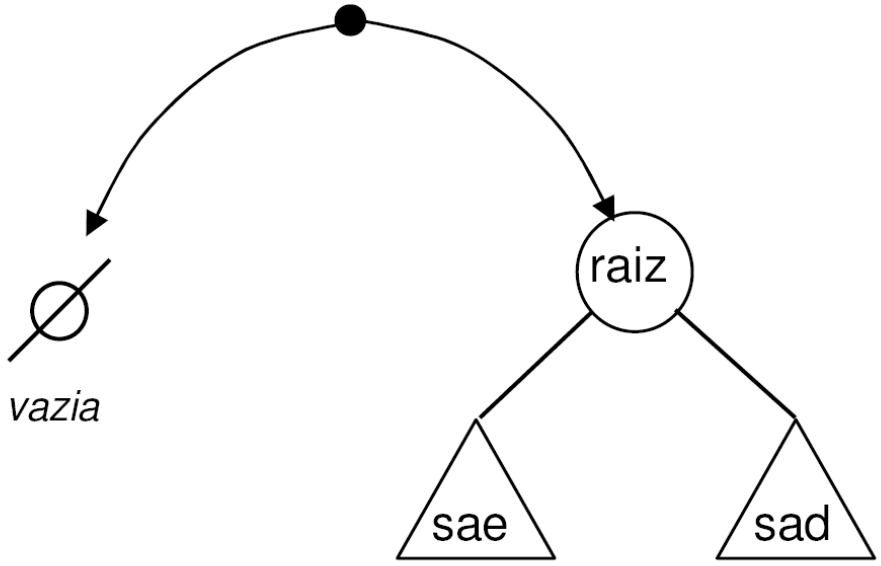
Árvores Binárias

- Notação textual
  - a árvore vazia é representada por <>
  - árvores não vazias por <raiz sae sad>
- Exemplo:
  - <a <b <> <d<><><>> > <c <e<><><>> <f<><><>>> >



Árvore Binária

- Uma árvore em que cada nó tem zero, um ou dois filhos
- Uma árvore binária é:
  - uma árvore vazia; ou
  - um nó raiz com duas sub-árvores:
    - a subárvore da direita (sad)
    - a subárvore da esquerda (sae)



Árvores Binárias – Implementação em C

- Representação: ponteiro para o nó raiz
- Representação de um nó na árvore:
  - Estrutura em C contendo
    - A informação propriamente dita (exemplo: um caractere, ou inteiro)
    - Dois ponteiros para as sub-árvores, à esquerda e à direita

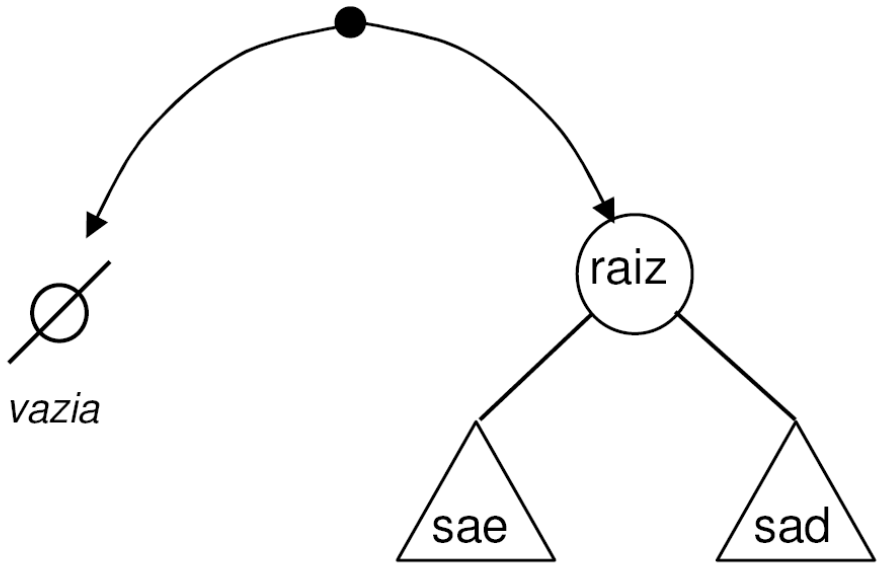
```
struct arv {
  char info;
  struct arv* esq;
  struct arv* dir;
};
```

TAD Árvores Binárias – Impl. em C (arv.h)

```
typedef struct arv Arv;
//Cria uma árvore vazia
Arv* arv_criavazia (void);
//cria uma árvore com a informação do nó raiz c, e
//com subárvore esquerda e e subárvore direita d
Arv* arv_cria (char c, Arv* e, Arv* d);
//libera o espaço de memória ocupado pela árvore a
Arv* arv_libera (Arv* a);
//retorna true se a árvore estiver vazia e false
//caso contrário
int arv_vazia (Arv* a);
//indica a ocorrência (1) ou não (0) do caracter c
int arv_pertence (Arv* a, char c);
//imprime as informações dos nós da árvore
void arv_imprime (Arv* a);
```

TAD Árvores Binárias – Implementação em C

- Implementação das funções:
  - implementação em geral recursiva
  - usa a definição recursiva da estrutura
- Uma árvore binária é:
  - uma árvore vazia; ou
  - um nó raiz com duas sub-árvores:
    - a sub-árvore da direita (sad)
    - a sub-árvore da esquerda (sae)



TAD Árvores Binárias – Implementação em C

- função arv_criavazia
  - cria uma árvore vazia

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

TAD Árvores Binárias – Implementação em C

- função arv_cria
  - cria um nó raiz dadas a informação e as duas sub-árvores, a da esquerda e a da direita
  - retorna o endereço do nó raiz criado

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

TAD Árvores Binárias – Implementação em C

- arv_criavazia e arv_cria
  - as duas funções para a criação de árvores representam os dois casos da definição recursiva de árvore binária:
    - uma árvore binária Arv* a;
      - é vazia a=arv_criavazia()
      - é composta por uma raiz e duas sub-árvores a=arv_cria(c,sae,sad);

TAD Árvores Binárias – Implementação em C

- função arv_vazia
  - indica se uma árvore é ou não vazia

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

TAD Árvores Binárias – Implementação em C

- função arv_libera
  - libera memória alocada pela estrutura da árvore
    - as sub-árvores devem ser liberadas antes de se liberar o nó raiz
  - retorna uma árvore vazia, representada por NULL

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

TAD Árvores Binárias – Implementação em C

- função arv_pertence
  - verifica a ocorrência de um caractere c em um dos nós
  - retorna um valor booleano (1 ou 0) indicando a ocorrência ou não do caractere na árvore

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

TAD Árvores Binárias – Implementação em C

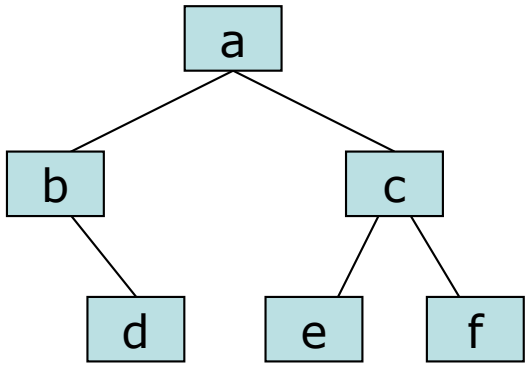
- função arv_imprime
  - percorre recursivamente a árvore, visitando todos os nós e imprimindo sua informação

```
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 */
  }
}
```

Exemplo

- Criar a árvore <a <b <> <d <><<>> > <c <e <><<> > <f <><<> > > >

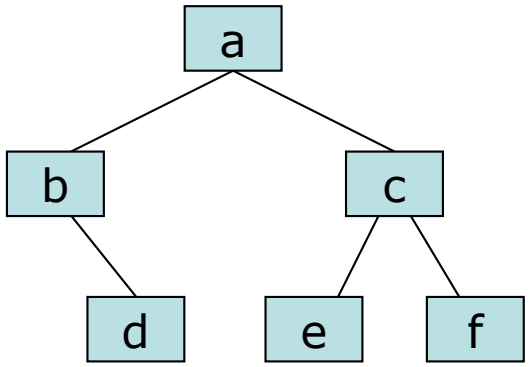
```
/* sub-árvore 'd' */
Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
/* sub-árvore 'b' */
Arv* a2= arv_cria('b',arv_criavazia(),a1);
/* sub-árvore 'e' */
Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
/* sub-árvore 'f' */
Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
/*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



Exemplo

- Criar a árvore <a <b <> <d <><>> > <c <e <><> > <f <><> > > >

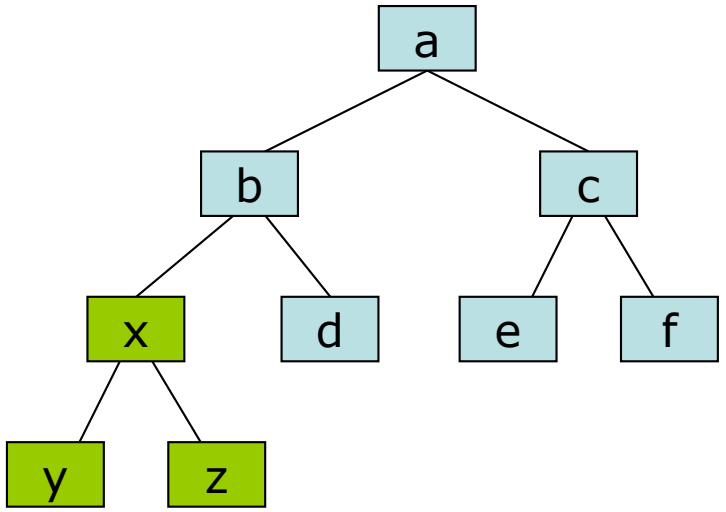
```
Arv* a = arv_cria('a',
           arv_cria('b',
               arv_criavazia(),
               arv_cria('d', arv_criavazia(), arv_criavazia())
           ),
           arv_cria('c',
              arv_cria('e', arv_criavazia(), arv_criavazia()),
              arv_cria('f', arv_criavazia(), arv_criavazia())
           )
);
```



Exemplo

- Acrescenta nós x, y e z

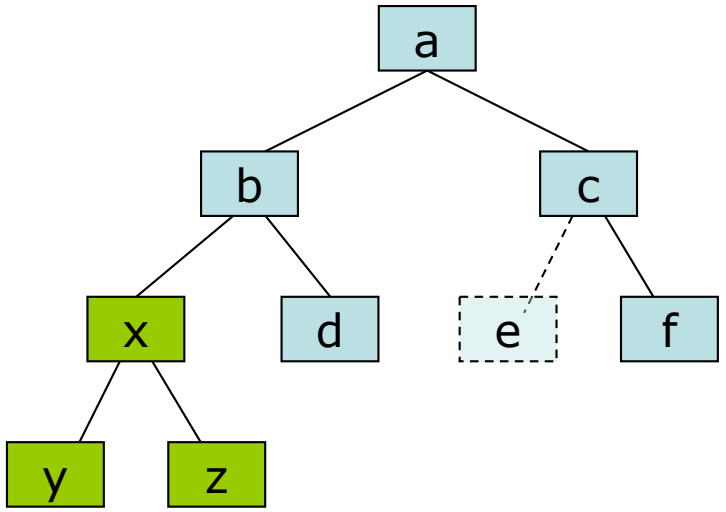
```
a->esq->esq =
   arv_cria('x',
       arv_cria('y',
            arv_criavazia(),
            arv_criavazia()),
       arv_cria('z',
            arv_criavazia(),
            arv_criavazia())
   );
```



Exemplo

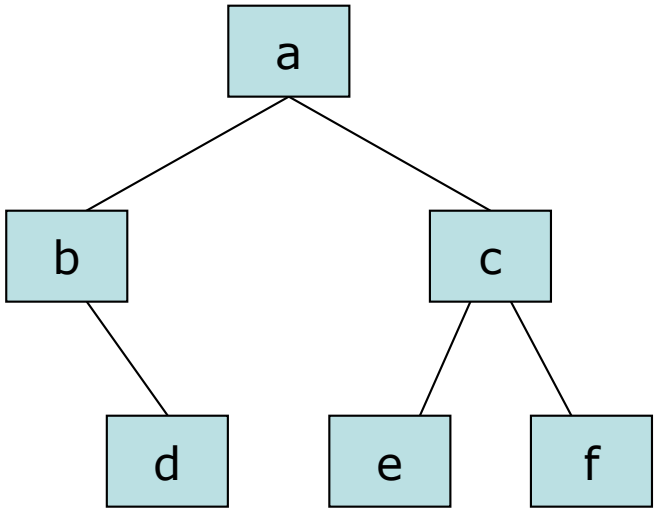
- Libera nós

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



Ordem de Percurso (ou travessia) – Árvores Binárias

- *Pré-ordem*:
  - trata *raiz*, percorre *sae*, percorre *sad*
  - exemplo: a b d c e f
- *Ordem simétrica (ou In-Ordem)*:
  - percorre *sae*, trata *raiz*, percorre *sad*
  - exemplo: b d a e c f
- *Pós-ordem*:
  - percorre *sae*, percorre *sad*, trata *raiz*
  - exemplo: d b e f c a



Pergunta

- função arv_pertence
  - Pré-ordem, pós-ordem ou in-ordem?

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

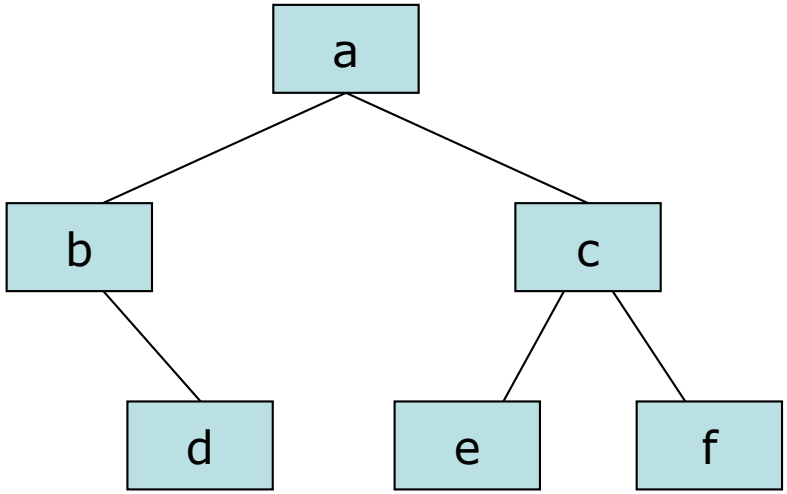
Pergunta

- função arv_libera
  - Pré-ordem, pós-ordem ou in-ordem?

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

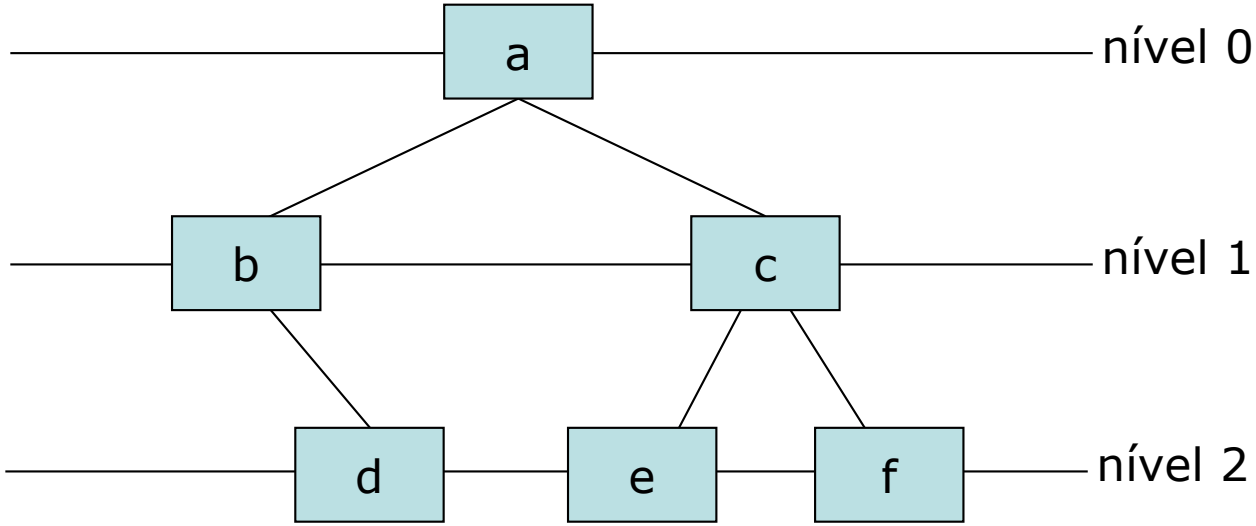
Árvores Binárias - Altura

- Propriedade das árvores
  - Existe apenas um caminho da raiz para qualquer nó
- Altura de uma árvore
  - comprimento do caminho mais longo da raiz até uma das folhas
  - a altura de uma árvore com um único nó raiz é zero
  - a altura de uma árvore vazia é -1
- Esforço computacional necessário para alcançar qualquer nó da árvore é proporcional à altura da árvore
- Exemplo:
  - h = 2



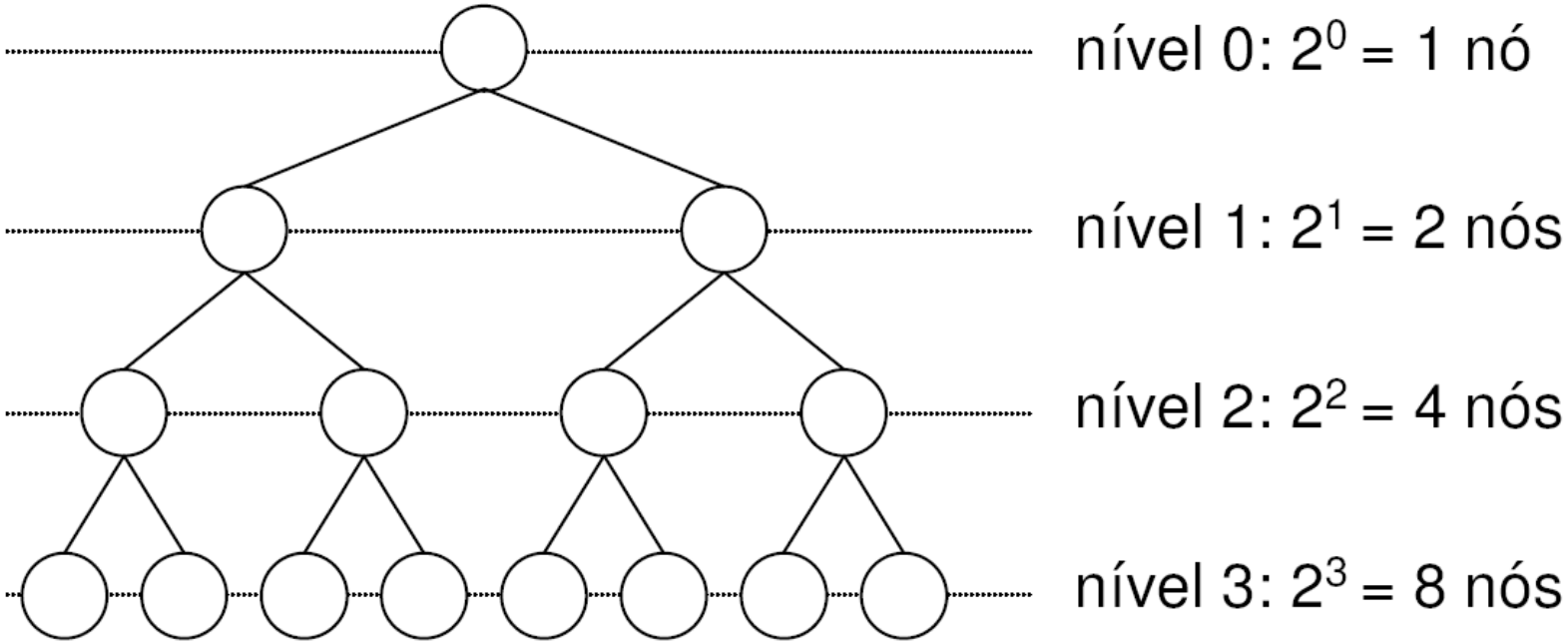
Árvores Binárias - conceitos

- Nível de um nó
  - a raiz está no nível 0, seus filhos diretos no nível 1, ...
  - o último nível da árvore é a altura da árvore



Árvores Binárias - conceitos

- Árvore Cheia
  - todos os seus nós internos têm duas sub-árvores associadas
  - número *n* de nós de uma árvore cheia de altura *h*
  - $n = 2^{h+1} - 1$



Altura (slides prof. Mutz)

Afirmação: A altura **h** de uma árvore com **n** elementos é $\lceil log_2^{n+1} \rceil$.

Prova:

1. Inicialmente, vamos assumir uma árvore binária balanceada **completa**.
2. Podemos escrever o número de elementos como a soma das quantidades de itens nos diferentes níveis:

$$1 + 2^1 + \cdots + 2^{h-1} = n$$

3. A soma da esquerda pode ser reescrita como $2^h - 1$ (prova a seguir):

$$2^h - 1 = n$$

4. Somando 1 nos dois lados e, em seguida, usando o $\log_2$:

$$\log_2(2^h) = \log_2(n+1)$$

5. Usando a propriedade $\log_k(a^b) = b\log(a)$ e o fato que $\log_2(2) = 1$:

$$h = \log_2(n+1)$$

Altura (slides prof. Mutz)

Afirmação: $1 + 2^1 + 2^2 + \cdots + 2^{h-1} = 2^h - 1.$

Temos uma pendência!

Prova:

Observe que $1 + 2^1 + 2^2 + \ldots + 2^{h-1}$ é uma soma de progressão geométrica cujo valor é dado por $S_n = \frac{a_1(q^n - 1)}{q-1}$, com $a_1 = 1$ e $q = 2$. Portanto, a soma será $S_h = \frac{1(2^h - 1)}{2-1} = (2^h - 1)$.

Dedução da Soma de PGs:

$$S_n = a_1 + a_1 q^1 + a_1 q^2 + \ldots + a_1 q^{n-1}$$

$$qS_n = a_1 q^1 + a_1 q^2 + \ldots + a_1 q^{n-1} + a_1 q^n$$

Multiplicando os dois lados da equação acima por q e mostrando as parcelas de uma forma conveniente.

Subtraindo as duas equações:

$$S_n(1 - q) = a_1 - a_1 q^n$$

Portanto:

$$S_n = \frac{a_1(q^n - 1)}{(q - 1)}$$

Árvores Binárias - conceitos

- Árvore Degenerada
  - Nós internos têm uma única subárvore associada
  - Vira uma estrutura linear
  - Arvore de altura h tem n = h+1

- Altura de uma árvore
  - Importante medida de eficiência (visitação do nó)
  - Árvore com n nós:
  - Altura mínima proporcional a log n (árvore binária cheia)
  - Altura máxima proporcional a n (árvore degenerada)

Exercícios

- Escrever uma função recursiva que calcule a altura de uma árvore binária dada. A altura de uma árvore é igual ao máximo nível de seus nós.

Respostas

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


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