

Estruturas de Dados

Módulo 13 - Árvores



Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,
Introdução a Estruturas de Dados, Editora Campus
(2004)

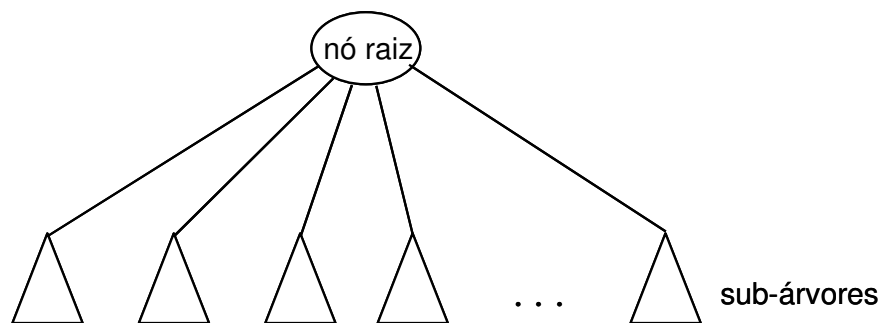
Capítulo 13 – Árvores

Tópicos

- Introdução
- Árvores binárias
 - Representação em C
 - Ordens de percurso em árvores binárias
 - Altura de uma árvore
- Árvores com número variável de filhos
 - Representação em C
 - Tipo abstrato de dado
 - Altura da árvore
 - Topologia binária

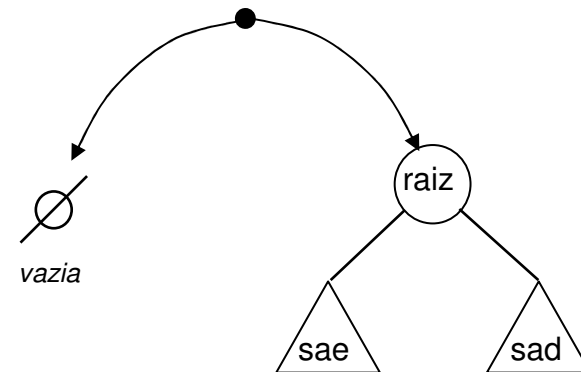
Introdução

- 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



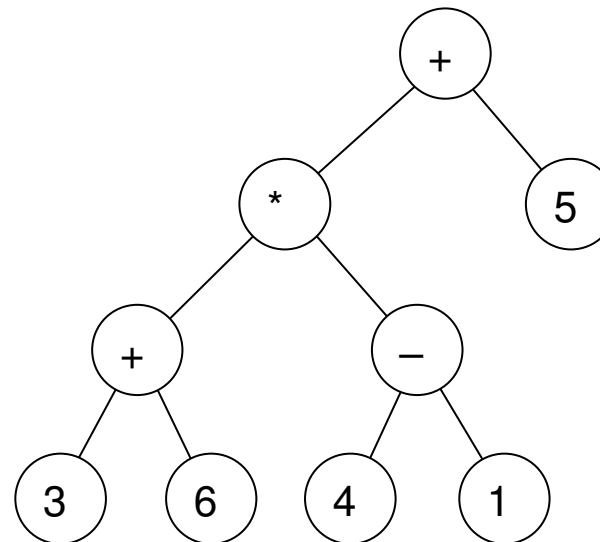
Árvores binárias

- Árvore binária
 - um 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 sub-árvore da direita (sad)
 - a sub-árvore da esquerda (sae)



Árvores binárias

- Exemplo
 - árvores binárias representando expressões aritméticas:
 - nós folhas representam operandos
 - nós internos operadores
 - exemplo: $(3+6)*(4-1)+5$

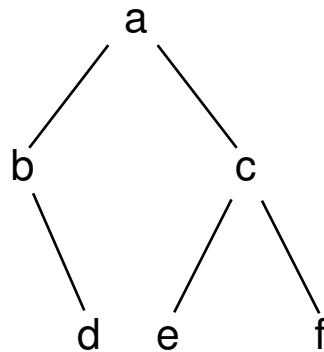


Árvores binárias

- Notação textual:

- a árvore vazia é representada por `<>`
- árvores não vazias por `<raiz sae sad>`
- exemplo:

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



Árvores binárias - Implementação em C

- Representação de uma árvore:
 - através de um ponteiro para o nó raiz
- Representação de um nó da árvore:
 - estrutura em C contendo
 - a informação propriamente dita (exemplo: um caractere)
 - dois ponteiros para as sub-árvores, à esquerda e à direita

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


Árvores binárias - Implementação em C

- Interface do tipo abstrato Árvore Binária: [arv.h](#)

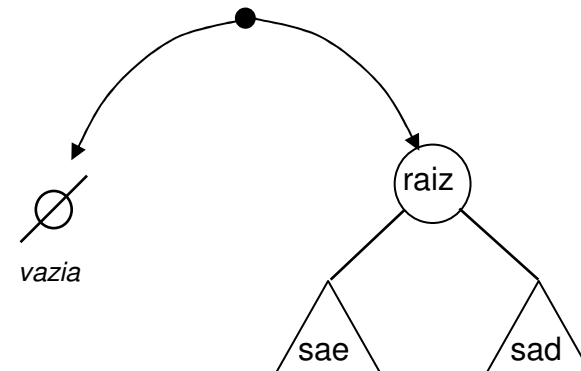
```
typedef struct arv Arv;  
  
Arv* arv_criavazia (void);  
Arv* arv_cria (char c, Arv* e, Arv* d);  
Arv* arv_libera (Arv* a);  
int  arv_vazia (Arv* a);  
int  arv_pertence (Arv* a, char c);  
void arv_imprime (Arv* a);
```

Árvores binárias - Implementação em C

- Implementação das funções:
 - implementação recursiva, em geral
 - 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)



Árvores binárias - Implementação em C

- função `arv_criavazia`
 - cria uma árvore vazia

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

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;
}
```

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);`

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;  
}
```

Árvores binárias - Implementação em C

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

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

Árvores binárias - Implementação em C

- função `arv_pertence`
 - verifica a ocorrência de um caractere `c` em um de 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);  
}
```


Árvores binárias - Implementação em C

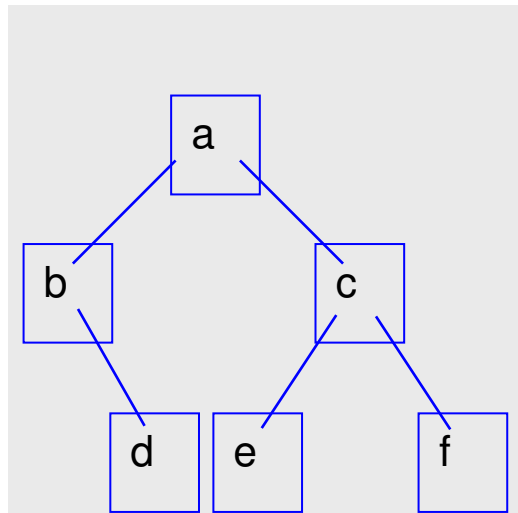
- função `arv_imprime`
 - percorre recursivamente a árvore, visitando todos os nós e imprimindo sua informação

```
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 */
    }
}
```

Árvores binárias - Implementação em C

- Exemplo: <a <b <> <d <><> > <c <e <><> > <f <><> > > >

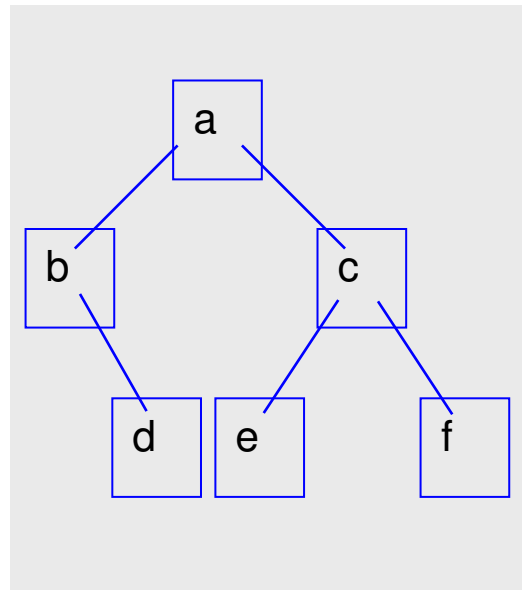
```
/* sub-árvore 'd' */
Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
/* sub-árvore 'b' */
Arv* a2= arv_cria('b',arv_criavazia(),a1);
/* sub-árvore 'e' */
Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
/* sub-árvore 'f' */
Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
/*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



Árvores binárias - Implementação em C

- Exemplo: $\langle a \langle b \langle \rangle \langle d \langle \rangle \langle \rangle \rangle \rangle \langle c \langle e \langle \rangle \langle \rangle \rangle \langle f \langle \rangle \langle \rangle \rangle \rangle$

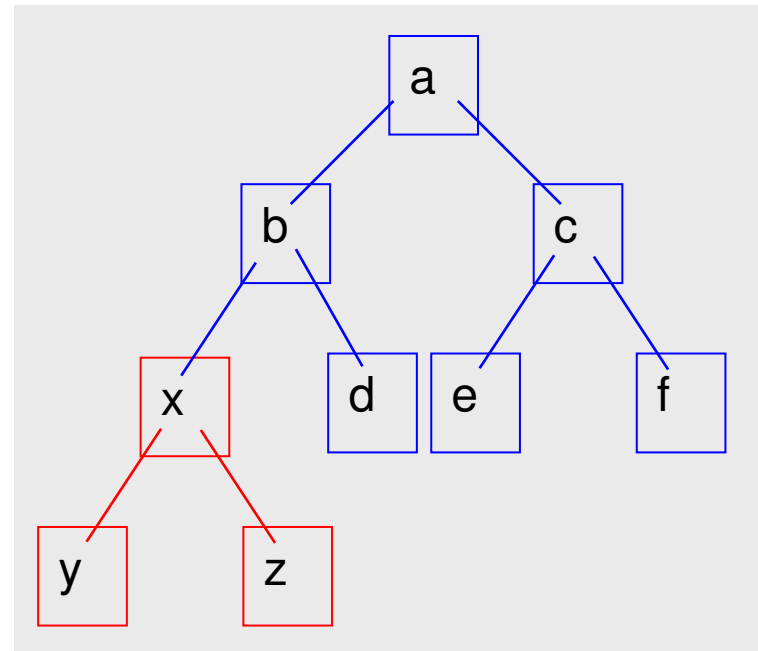
```
Arv* a = arv_cria('a',  
    arv_cria('b',  
        arv_criavazia(),  
        arv_cria('d', arv_criavazia(), arv_criavazia())  
    ),  
    arv_cria('c',  
        arv_cria('e', arv_criavazia(), arv_criavazia()),  
        arv_cria('f', arv_criavazia(), arv_criavazia())  
    )  
);
```



Árvores binárias - Implementação em C

- Exemplo - acrescenta nós

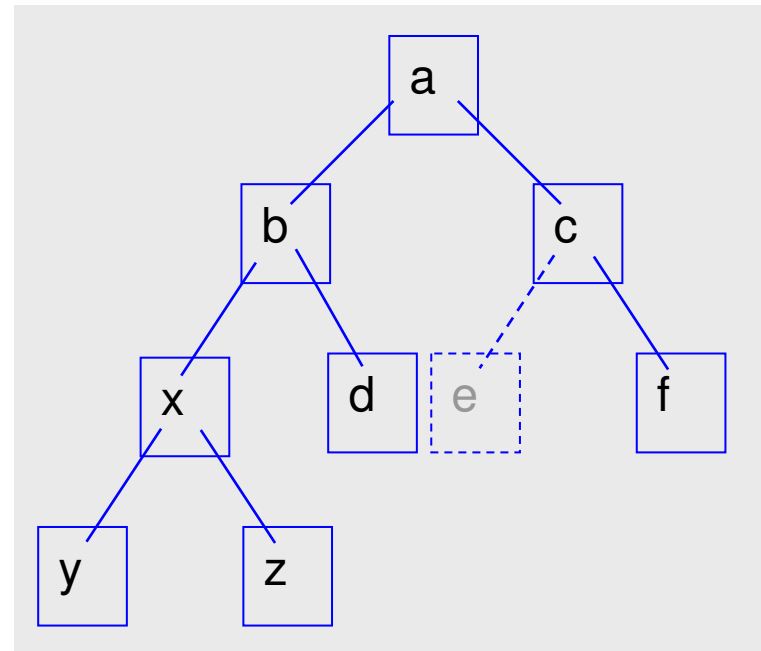
```
a->esq->esq =  
    arv_cria('x',  
            arv_cria('y',  
                    arv_criavazia(),  
                    arv_criavazia()),  
            arv_cria('z',  
                    arv_criavazia(),  
                    arv_criavazia())  
    );
```



Árvores binárias - Implementação em C

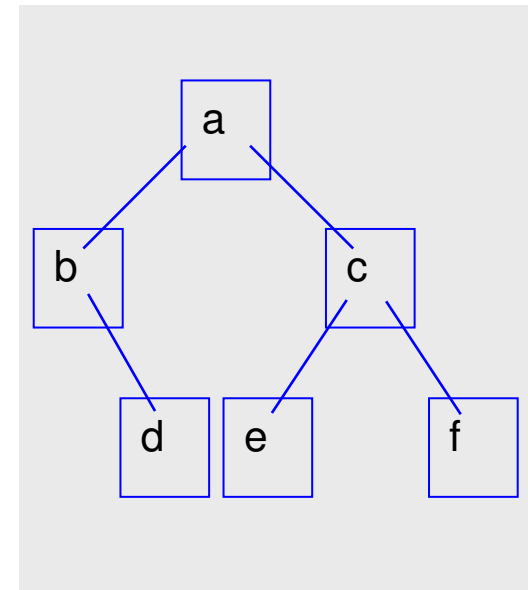
- Exemplo - libera nós

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



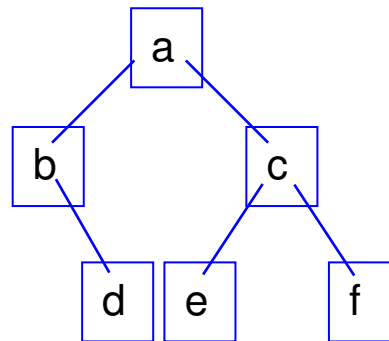
Árvores binárias - Ordens de percurso

- Ordens de percurso:
 - *pré-ordem*:
 - trata *raiz*, percorre *sae*, percorre *sad*
 - exemplo: a b d c e f
 - *ordem simétrica*:
 - 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



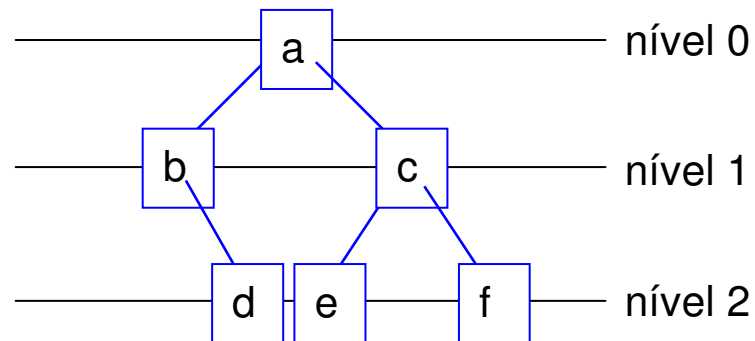
Árvores binárias - Altura

- Propriedade fundamental de árvores
 - só existe 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
 - exemplo:
 - $h = 2$



Árvores binárias - Altura

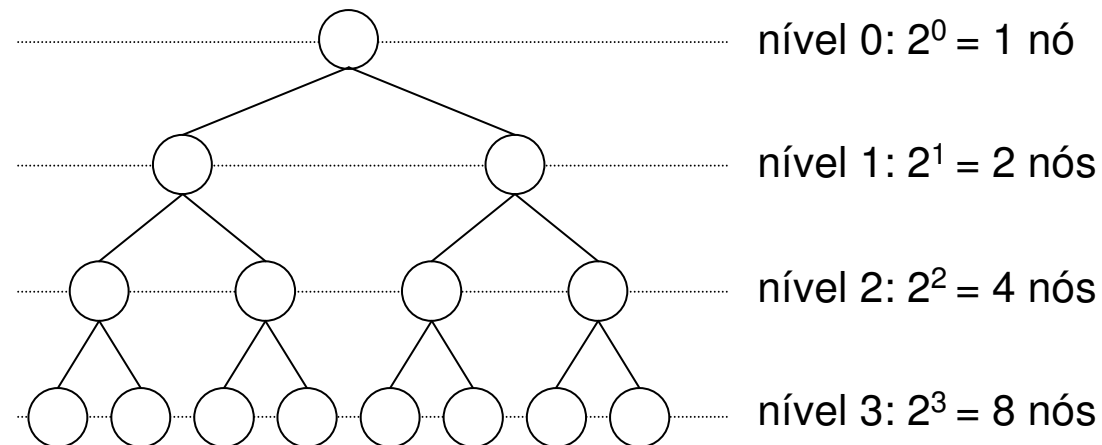
- Nível de um nó
 - a raiz está no nível 0, seus filhos diretos no nível 1, ...
 - o último nível da árvore é a altura da árvore



Árvores binárias - Altura

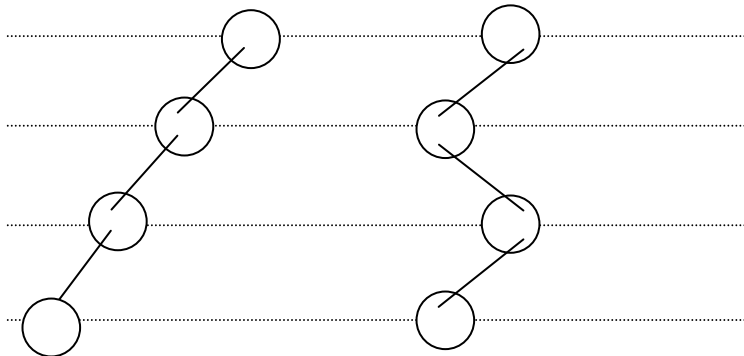
- Árvore cheia
 - todos os seus nós internos têm duas sub-árvores associadas
 - número n de nós de uma árvore cheia de altura h

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



Árvores binárias - Altura

- Árvore degenerada
 - todos os seus nós internos têm uma única sub-árvore associada
 - número n de nós de uma árvore degenerada de altura h
 $n = h + 1$

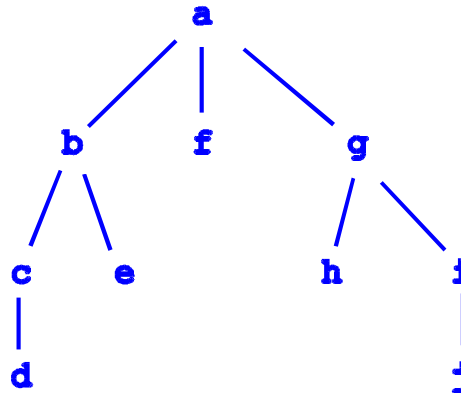


Árvores binárias - Altura

- Esforço computacional necessário para alcançar qualquer nó da árvore
 - proporcional à altura da árvore
 - altura de uma árvore binária com n nós
 - mínima: proporcional a $\log n$ (caso da árvore cheia)
 - máxima: proporcional a n (caso da árvore degenerada)

Árvores com número variável de filhos

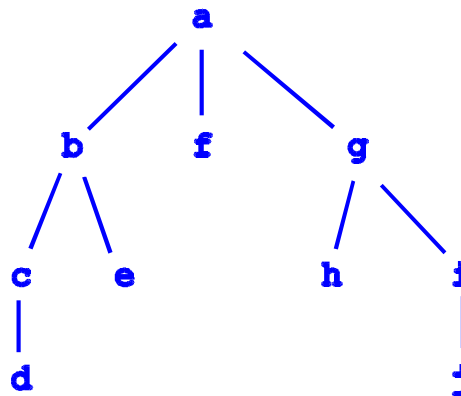
- Árvore com número variável de filhos:
 - cada nó pode ter mais do que duas sub-árvores associadas
 - sub-árvores de um nó dispostas em ordem
 - primeira sub-árvore (*sa1*), segunda sub-árvore (*sa2*), etc.



Árvores com número variável de filhos

- Notação textual: <raiz sa1 sa2 ... san>

- exemplo:

$$\alpha = \langle a \, \langle b \, \langle c \, \langle d \rangle \rangle \, \langle e \rangle \, \langle f \rangle \, \langle g \, \langle h \rangle \, \langle i \, \langle j \rangle \rangle \rangle \rangle$$


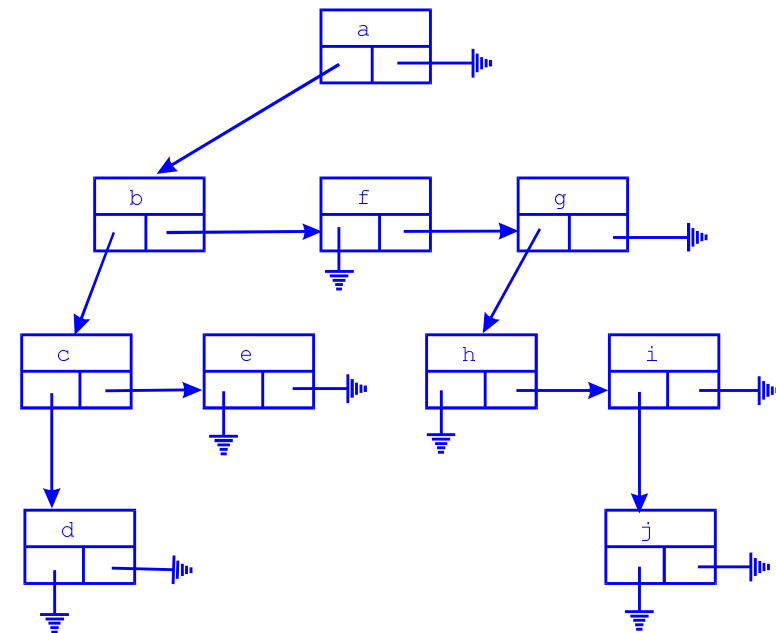
Árvores com número variável de filhos

Representação em C

- Representação de árvore com número variável de filhos:

- utiliza uma “lista de filhos”:

- um nó aponta apenas para seu primeiro (prim) filho
 - cada um de seus filhos aponta para o próximo (prox) irmão



Árvores com número variável de filhos

Representação em C

- Representação de um nó da árvore:
 - estrutura em C contendo
 - a informação propriamente dita (exemplo: um caractere)
 - ponteiro para a primeira sub-árvore filha
 - NULL se o nó for uma folha
 - ponteiro para a próxima sub-árvore irmão
 - NULL se for o último filho

```
struct arvvar {  
    char info;  
    struct arvvar *prim;    /* ponteiro para eventual primeiro filho */  
    struct arvvar *prox;    /* ponteiro para eventual irmão */  
};  
  
typedef struct arvvar ArvVar;
```

Árvores com número variável de filhos

Representação em C

- Interface do tipo abstrato Árvore Variável: [arvvar.h](#)

```
typedef struct arvvar ArvVar;  
  
ArvVar* arvvar_cria (char c);  
void    arvvar_insere (ArvVar* a, ArvVar* sa);  
void    arvvar_imprime (ArvVar* a);  
int     arvvar_pertence (ArvVar* a, char c);  
void    arvvar_libera (ArvVar* a);
```

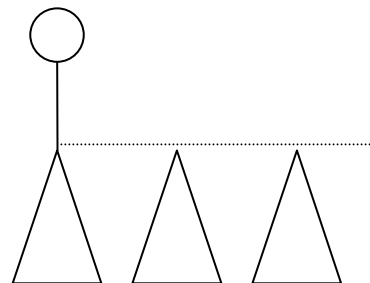

Árvores com número variável de filhos

Representação em C

- Implementação das funções:
 - implementação recursiva, em geral
 - usa a definição recursiva da estrutura

Uma árvore é composta por:

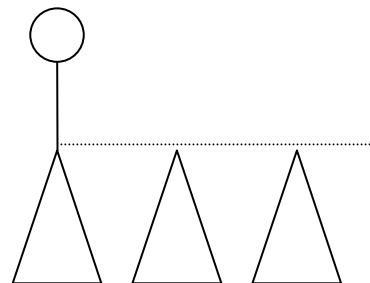
- um nó raiz
- zero ou mais sub-árvores



Árvores com número variável de filhos

Representação em C

- Implementação das funções (cont.):
 - uma árvore não pode ser vazia
 - uma folha é identificada como um nó com zero sub-árvores
 - uma folha não é um nó com sub-árvores vazias, como nas árvores binárias
 - funções não consideram o caso de árvores vazias



Árvores com número variável de filhos

Representação em C

- função `arvv_cria`
 - cria uma folha
 - aloca o nó
 - inicializa os campos, atribuindo NULL aos campos `prim` e `prox`

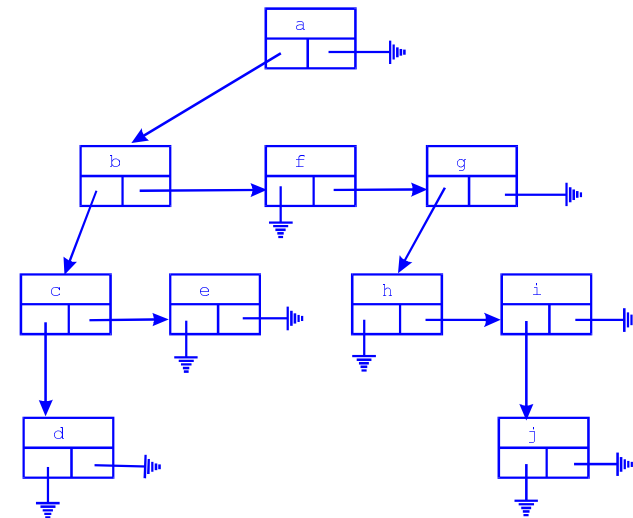
```
ArvVar* arvv_cria (char c)
{
    ArvVar *a =(ArvVar *) malloc(sizeof(ArvVar));
    a->info = c;
    a->prim = NULL;
    a->prox = NULL;
    return a;
}
```

Árvores com número variável de filhos

Representação em C

- função `arvv_inserere`
 - insere uma nova sub-árvore como filha de um dado, sempre no início da lista, por simplicidade

```
void arvv_inserere (ArvVar* a, ArvVar* sa)
{
    sa->prox = a->prim;
    a->prim = sa;
}
```

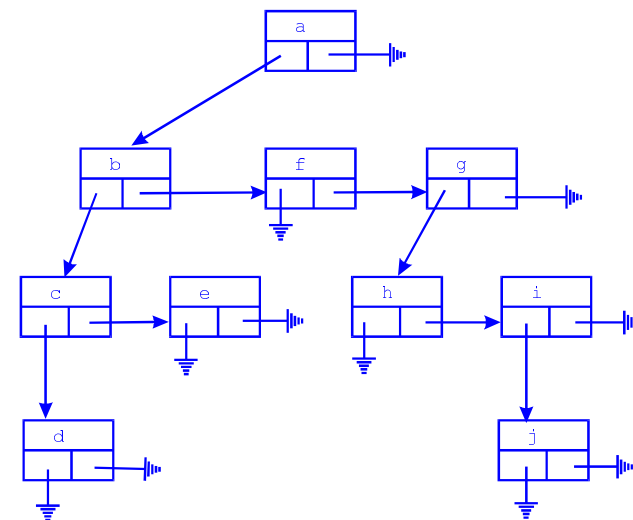


Árvores com número variável de filhos

Representação em C

- função `arvv_imprime`
 - imprime o conteúdo dos nós em pré-ordem

```
void arvv_imprime (ArvVar* a)
{
    ArvVar* p;
    printf("<%c\n",a->info);
    for (p=a->prim; p!=NULL; p=p->prox)
        arvv_imprime(p); /* imprime filhas */
    printf(">");
}
```

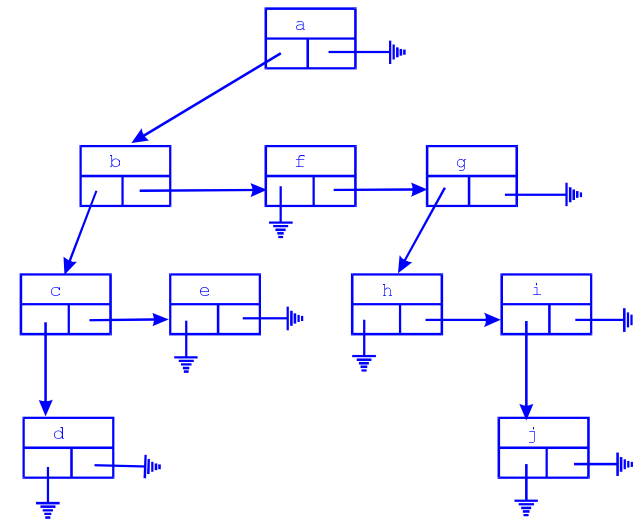


Árvores com número variável de filhos

Representação em C

- função `arvv_pertence`
 - verifica a ocorrência de uma dada informação na árvore

```
int arvv_pertence (ArvVar* a, char c)
{
    ArvVar* p;
    if (a->info==c)
        return 1;
    else {
        for (p=a->prim; p!=NULL; p=p->prox) {
            if (arvv_pertence(p,c))
                return 1;
        }
        return 0;
    }
}
```

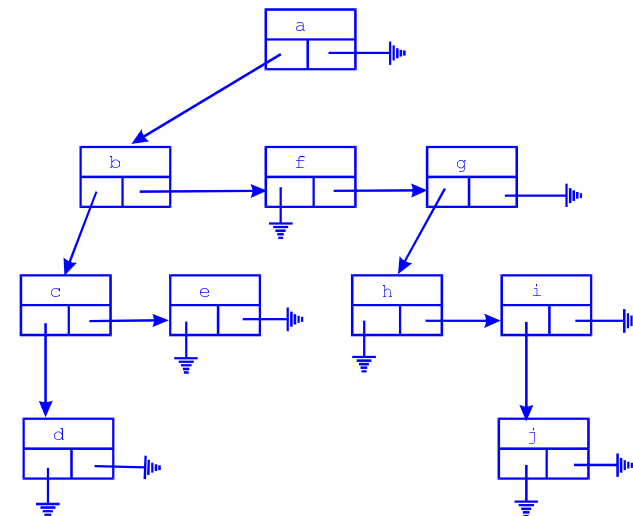


Árvores com número variável de filhos

Representação em C

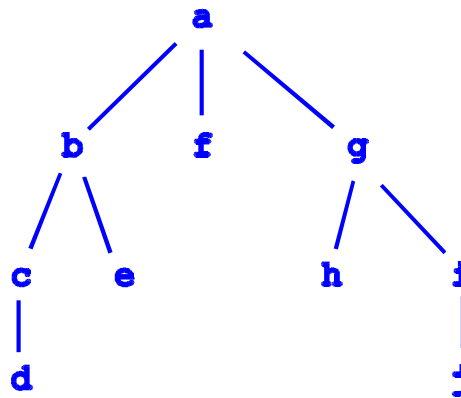
- função `arvv_libera`
 - libera a memória alocada pela árvore
 - libera as sub-árvores antes de liberar o espaço associado a um nó (libera em pós-ordem)

```
void arvv_libera (ArvVar* a)
{
    ArvVar* p = a->prim;
    while (p!=NULL) {
        ArvVar* t = p->prox;
        arvv_libera(p);
        p = t;
    }
    free(a);
}
```



Árvores com número variável de filhos - Altura

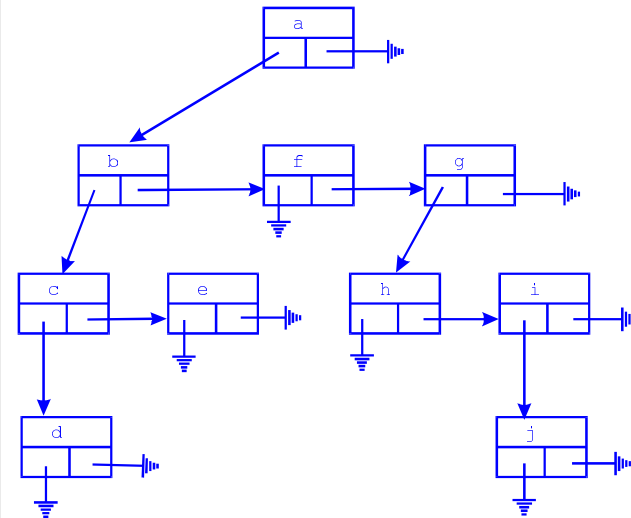
- nível e altura
 - (definidos de forma semelhante a árvores binárias)
 - exemplo:
 - $h = 3$



Árvores com número variável de filhos - Altura

- função `arvv_altura`
 - maior altura entre as sub-árvores, acrescido de uma unidade
 - caso o nó raiz não tenha filhos, a altura da árvore deve ser 0

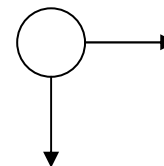
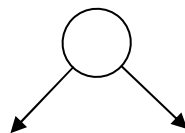
```
int arvv_altura (ArvVar* a)
{
    int hmax = -1; /* -1 para arv. sem filhos */
    ArvVar* p;
    for (p=a->prim; p!=NULL; p=p->prox) {
        int h = arvv_altura(p);
        if (h > hmax)
            hmax = h;
    }
    return hmax + 1;
}
```



Árvores com número variável de filhos

Topologia Binária

- Topologia binária:
 - representação de um nó de uma árvore variável \equiv representação de um nó da árvore binária
 - nó possui informação e dois ponteiros para sub-árvores
 - árvore binária:
 - ponteiros para as sub-árvores à esquerda e à direita
 - árvores variável:
 - ponteiros para a primeira sub-árvore filha e para a sub-árvore irmã



Árvores com número variável de filhos

Topologia Binária

- Topologia binária:
 - redefinição de árvore com número variável de filhos:
 - árvore vazia, ou
 - um nó raiz tendo duas sub-árvores, identificadas como a sub-árvore filha e a sub-árvore irmã
 - re-implementação das funções:
 - pode se basear na nova definição
 - o caso da árvore vazia agora deve ser considerado

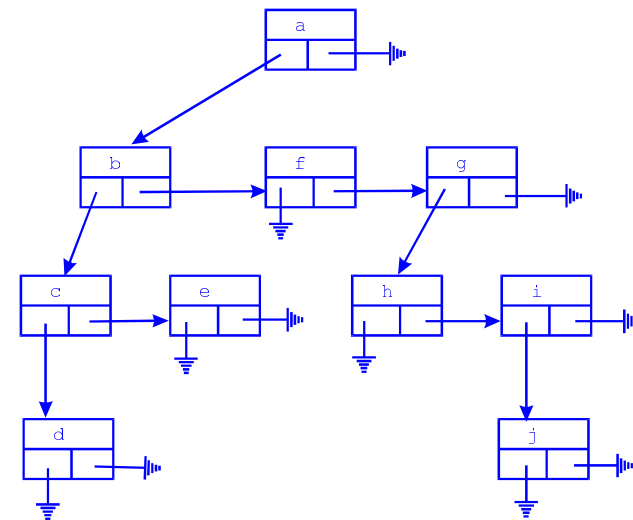
Árvores com número variável de filhos

Topologia Binária

- função para calcular altura:
 - a altura da árvore será o maior valor entre
 - a altura da sub-árvore filha acrescido de uma unidade e
 - a altura da sub-árvore irmã

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

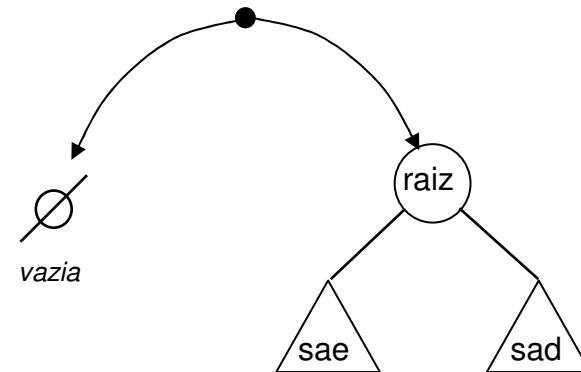
int arv_v_altura (ArvVar* a)
{ if (a==NULL)
  return -1;
  else
    return max2(1+arv_v_altura(a->prim),
                arv_v_altura(a->prox));
}
```



Resumo

Árvore binária

- uma árvore vazia; ou
- um nó raiz com duas sub-árvores:
 - a sub-árvore da direita (sad)
 - a sub-árvore da esquerda (sae)



Árvore com número variável de filhos

- um nó raiz
- zero ou mais sub-árvores

