



Estruturas de Dados  
Aula 16:  
Árvores com Número Variável  
de Filhos

02/06/2010

## Fontes Bibliográficas

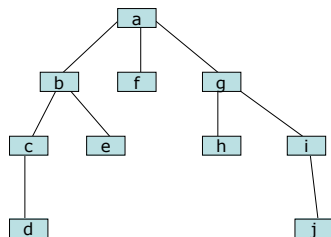


- Livros:
  - Introdução a Estruturas de Dados (Celes, Cerqueira e Rangel): **Capítulo 13**;
  - Projeto de Algoritmos (Nivio Ziviani): **Capítulo 5**;
  - Estruturas de Dados e seus Algoritmos (Szwarcfiter, et. al): **Capítulo 3**;
  - Algorithms in C (Sedgewick): **Capítulo 5**;
- Slides baseados no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

## Árvore com Número Variável de Filhos



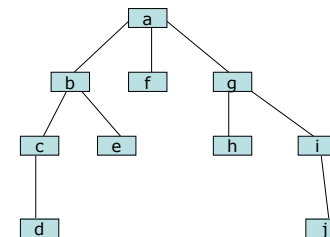
- Árvore com número variável de filhos:
  - cada nó pode ter mais do que duas sub-árvores associadas
  - sub-árvores de um nó dispostas em ordem
    - primeira sub-árvore (*sa1*),
    - segunda sub-árvore (*sa2*), etc.



## Representação



- Em formato textual
  - <raiz sa1 sa2 ... san>
- A árvore exemplo seria representado por
  - <a <b <c <d>> <e>> <f> <g <h> <i <j>>>>>

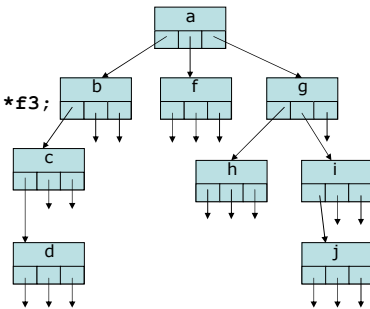


## Representação em C



- São possíveis várias representações em C (dependendo da aplicação)
- Por exemplo, em uma aplicação na qual sabe-se que o número máximo de filhos de um dado nó é 3:

```
struct arv3 {  
    char info;  
    struct arv3 *f1, *f2, *f3;  
};
```



## Representação em C (cont.)



- Função para imprimir

```
void arv3_imprime (Arv3* a) {  
    if (a != NULL) {  
        printf("<%c", a->info);  
        arv3_imprime (a->f1);  
        arv3_imprime (a->f2);  
        arv3_imprime (a->f3);  
        printf(">");  
    }  
}
```

- Note que não há uma maneira sistemática para acessar os nós filhos.
- Impraticável declarar um campo para cada filho (imagine uma árvore com até 100 filhos!)

## Representação em C (cont.)



- Uma outra representação possível

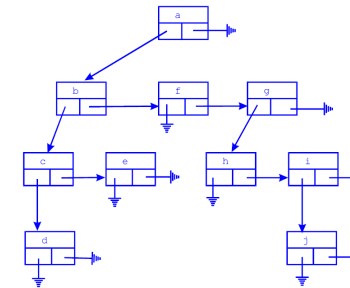
```
#define N 3  
struct arv3 {  
    char info;  
    struct arv3 *f[N];  
}  
void arv3_imprime (Arv3* a) {  
    if (a != NULL) {  
        int i;  
        printf("<%c", a->info);  
        for (i=0; i<N; i++)  
            arv3_imprime (a->f[i]);  
        printf(">");  
    }  
}
```

- Essa abordagem é adequada para aplicações que não se sabe o número de filhos?

## Representação em C Adotada



- Adequada para representar um número variável de filhos
- Filhos de um nó são representados por uma *lista*
  - um nó aponta para o seu primeiro filho (*prim*)
  - cada filho aponta para o próximo (*prox*) irmão



## Representação em C Adotada



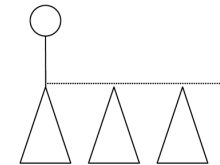
- Representação de um nó da árvore:
  - a informação propriamente dita (exemplo: um caractere)
  - ponteiro para a primeira sub-árvore filha
    - NULL se o nó for uma folha
  - ponteiro para a próxima sub-árvore irmão
    - NULL se for o último filho

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

## Definição



- Para implementações recursivas, usar a seguinte definição:
- Uma árvore é composta de
  - Um nó raiz; e
  - Zero ou mais subárvores.
- Nó folha definido como nó com zero subárvores
  - Diferente da definição de folha na árvore binária (folha era nó com subárvores vazias)
- Funções não consideram o caso de árvore vazias (pré-condição)
- Condições de contorno (parada da recursão) devem considerar essa restrição



## Exemplo de TADArvVar ("TadArvVar.h")



- Conjunto de operações do TAD (usadas como exemplo)
  - Typedef struct arvvar ArvVar;
  - Cria um nó folha, dada a informação a ser armazenada
    - ArvVar\* arvvar\_cria (char c);
  - Insere uma nova subárvore como filha de um dado nó
    - void arvvar\_insere (ArvVar\* a, ArvVar\* sa);
  - Percorre todos os nós e imprime suas informações
    - void arvvar\_imprime (ArvVar\* a);
  - Verifica a ocorrência de uma dada informação na árvore
    - int arvvar\_pertence (ArvVar\* a, char c);
  - Libera toda a memória alocada pela árvore
    - void arvvar\_libera (ArvVar\* a);

## Exemplo de TADArvVar



- Conjunto de operações do TAD (usadas como exemplo)
  - Cria um nó folha, dada a informação a ser armazenada
    - aloca o nó
    - inicializa os campos, atribuindo NULL aos campos prim e prox

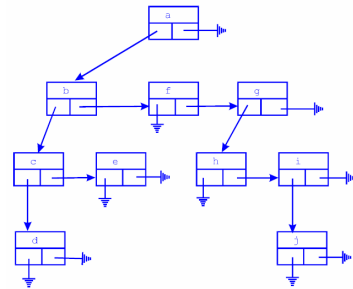
```
ArvVar* arvvar_cria (char c) {
    ArvVar *a =(ArvVar *) malloc(sizeof(ArvVar));
    a->info = c;
    a->prim = NULL;
    a->prox = NULL;
    return a;
}
```

## Exemplo de TADArvVar



- Insere uma nova subárvore como filha de um dado nó
  - insere uma nova sub-árvore como filha de um dado nó, sempre no início da lista, por simplicidade

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

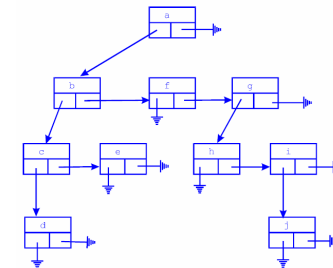


## Exemplo de TADArvVar



- Percorre todos os nós e imprime suas informações
  - 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(">");  
}
```

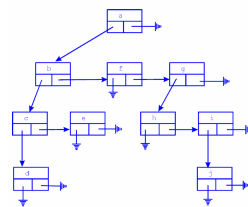


## Exemplo de TADArvVar



- 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;  
}
```

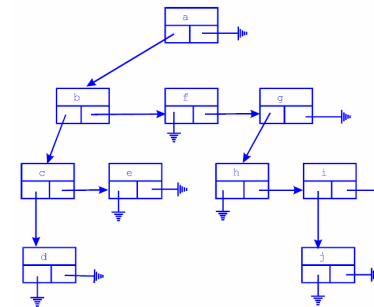


## Exemplo de TADArvVar



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

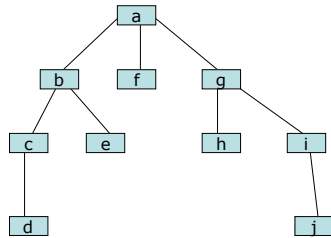
```
void arvv_libera (ArvVar* a) {  
    ArvVar* p = a->prim;  
    while (p!=NULL) {  
        ArvVar* t = p->prox;  
        arvv_libera(p);  
        p = t;  
    }  
    free(a);  
}
```



## Altura



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

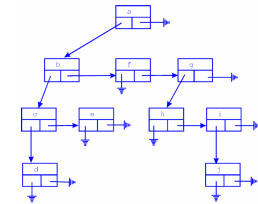


## Altura



- 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;
}
```



## Exercício



- Implemente uma função que retorne a quantidade de folhas de uma árvore com número variável de filhos. Essa função deve obedecer ao protótipo:
  - `int folhas (ArvVar* a);`
- Implemente uma função que compare se duas árvores são iguais (apenas em estrutura). Essa função deve obedecer ao protótipo:
  - `int igual (ArvVar* a, ArvVar* b);`

## Respostas



```
int folhas (ArvVar* a) {
    ArvVar* p;
    int n = 0;
    if (a->prim == NULL)
        return 1;
    for (p=a->prim; p!=NULL; p=p->prox) {
        n = n + folhas(p);
    }
    return n;
}
```

## Respostas (cont.)



```
int igual (ArvVar* a, ArvVar* b) {
    ArvVar* p;
    ArvVar* q;
    if (a == NULL && b == NULL)
        return 1;
    for (p=a->prim, q=b->prim; p!=NULL && q!=NULL;
         p=p->prox, q=q->prox) {
        if (!igual(p,q))
            return 0;
    };
    return 1;
}
```

## Exercícios



- Considerando as seguintes declarações de uma árvore genérica:

```
struct arvvar {
    int info;
    struct arvvar* prim;
    struct arvvar* prox;
};
typedef struct arvvar ArvVar;
```

- implemente uma função que, dada uma árvore, retorne a quantidade de nós que guardam valores maiores que um determinado valor x (também passado como parâmetro). Essa função deve obedecer o protótipo:

```
int maiores (ArvVar* a, int x)
```

## Exercícios



- Considerando as seguintes declarações de uma árvore genérica

```
struct arvvar {
    int info;
    struct arvvar* prim;
    struct arvvar* prox;
};
typedef struct arvvar ArvVar;
```

- implemente uma função que, dada uma árvore, retorne a quantidade de nós que possuem apenas um filho. Essa função deve obedecer o protótipo:

```
int um_filho (ArvVar* a);
```