Árvores Continuação...

- Sérgio Carlos Portari Júnior

Árvores Genéricas

- Como vimos, numa árvore binária o número de filhos dos nós é limitado em no máximo dois.
- No caso da árvore genérica, esta restrição não existe.
- Cada nó pode ter um número arbitrário de filhos.
- Essa estrutura deve ser usada, por exemplo, para representar uma árvore genealógica.

Árvores Genéricas

- Como veremos, as funções para manipularem uma árvore genérica também serão implementadas de forma recursiva, e serão baseadas na seguinte definição:

- uma árvore genérica é composta por:
  - um nó raiz; e
  - zero ou mais sub-árvores.

Árvores Genéricas

- Estritamente, segundo essa definição, uma árvore não pode ser vazia, e a árvore vazia não é sequer mencionada na definição.
- Assim, uma folha de uma árvore não é um nó com sub-árvores vazias, como no caso da árvore binária, mas é um nó com *zero* subárvores.
- Em qualquer definição recursiva deve haver uma “condição de contorno”, que permita a definição de estruturas finitas, e, no nosso caso, a definição de uma árvore se encerra nas *folhas*, que são identificadas como sendo nós com zero sub-árvores.

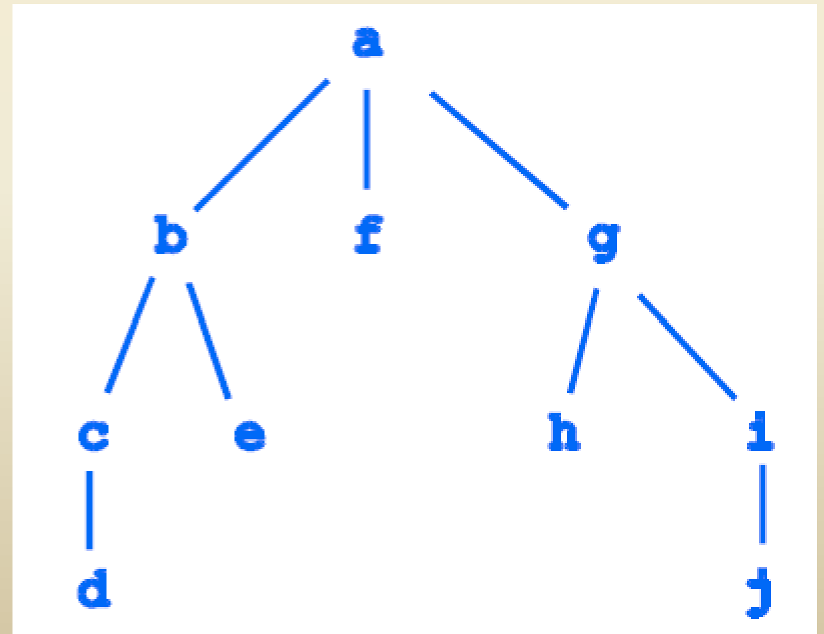
Árvores Genéricas

- Como as funções que implementaremos nesta seção serão baseadas nessa definição, não será considerado o caso de árvores vazias.
- Esta pequena restrição simplifica as implementações recursivas e, em geral, não limita a utilização da estrutura em aplicações reais.
- Uma árvore de diretório, por exemplo, nunca é vazia, pois sempre existe o diretório base – o diretório raiz.

Árvores Genéricas

- Como as sub-árvores de um determinado nó formam um conjunto linear e são dispostas numa determinada ordem, faz sentido falarmos em primeira sub-árvore (*sa1*), segunda sub-árvore (*sa2*), etc.

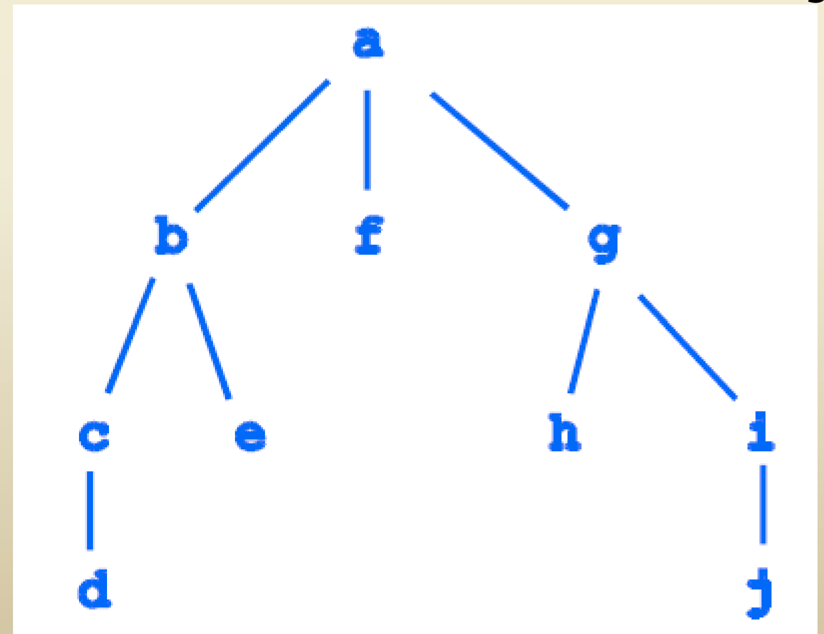
- Um exemplo de uma árvore genérica é Ilustrado na Figura ao lado.



Árvores Genéricas

- Nesse exemplo, podemos notar que o a árvore com raiz no nó **a** tem 3 sub-árvores, ou, equivalentemente, o nó **a** tem 3 filhos.
- Os nós **b** e **g** tem dois filhos cada um; os nós **c** e **i** tem um filho cada, e os nós **d**, **e**, **f**, **h** e **j** são folhas,

  e tem zero filhos.



Árvores Genéricas em C

- Dependendo da aplicação, podemos usar várias estruturas para representar árvores, levando em consideração o número de filhos que cada nó pode apresentar.
- Se soubermos, por exemplo, que numa aplicação o número máximo de filhos que um nó pode apresentar é 3, podemos montar uma estrutura com 3 campos para apontadores para os nós filhos, digamos, f1, f2 e f3.

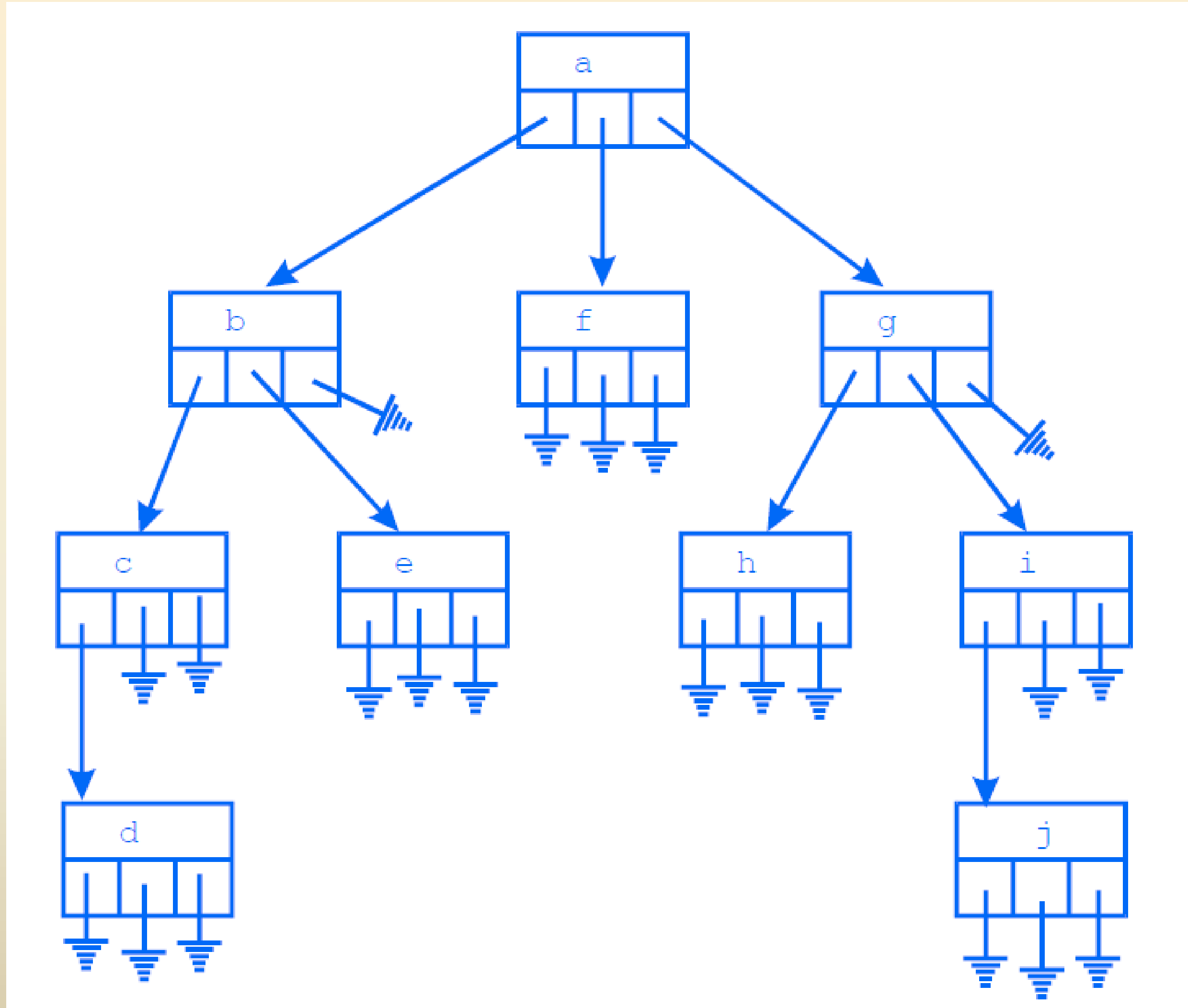
Árvores Genéricas em C

- Os campos não utilizados podem ser preenchidos com o valor nulo NULL, sendo sempre utilizados os campos em ordem.
- Assim, se o nó n tem 2 filhos, os campos f1 e f2 seriam utilizados, nessa ordem, para apontar para eles, ficando f3 vazio.
- Prevendo um número máximo de filhos igual a 3, e considerando a implementação de árvores para armazenar valores de caracteres simples, a declaração do TAD que representa o nó da árvore poderia ser:

Árvores Genéricas em C

```
typedef struct arv3 {
   char val;
   struct arv3 *f1, *f2, *f3;
} Arvore3;
```

Árvore Genérica em C



Árvore Genérica em C

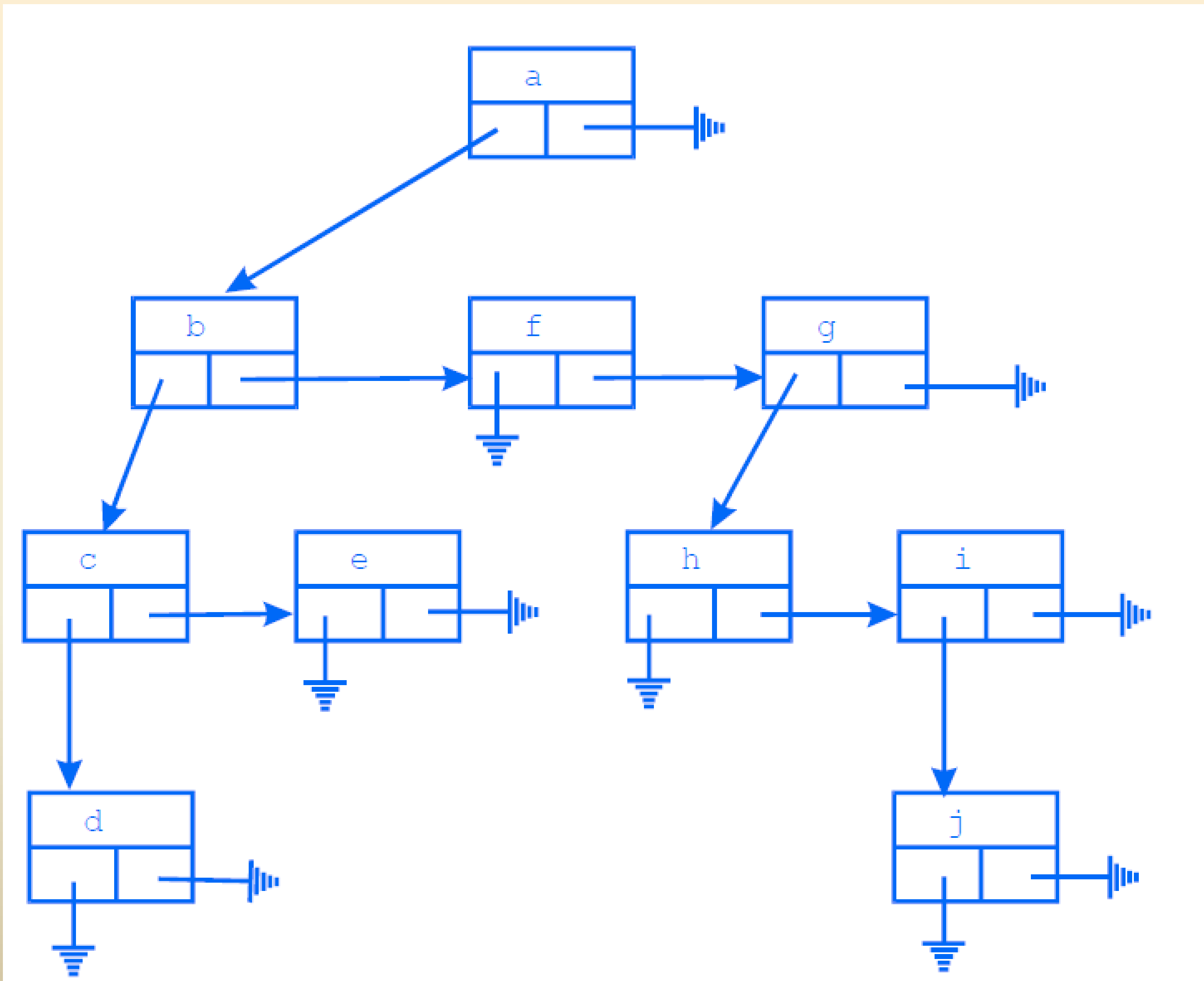
- Como se pode ver no exemplo, em cada um dos nós que tem menos de três filhos, o espaço correspondente aos filhos inexistentes é desperdiçado.
- Além disso, se não existe um limite superior no número de filhos, esta técnica pode não ser aplicável.

Árvore Genérica em C

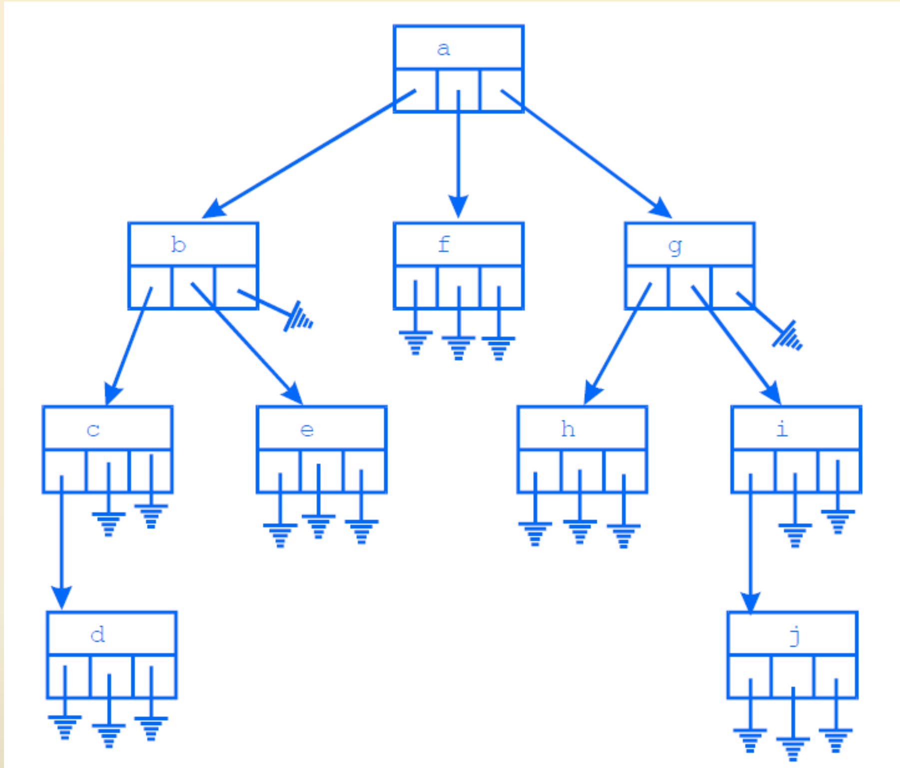
- Uma solução que leva a um aproveitamento melhor do espaço utiliza uma "lista de filhos": um nó aponta apenas para seu primeiro (prim) filho, e cada um de seus filhos, exceto o último, aponta para o próximo (prox) irmão. A declaração de um nó pode ser:

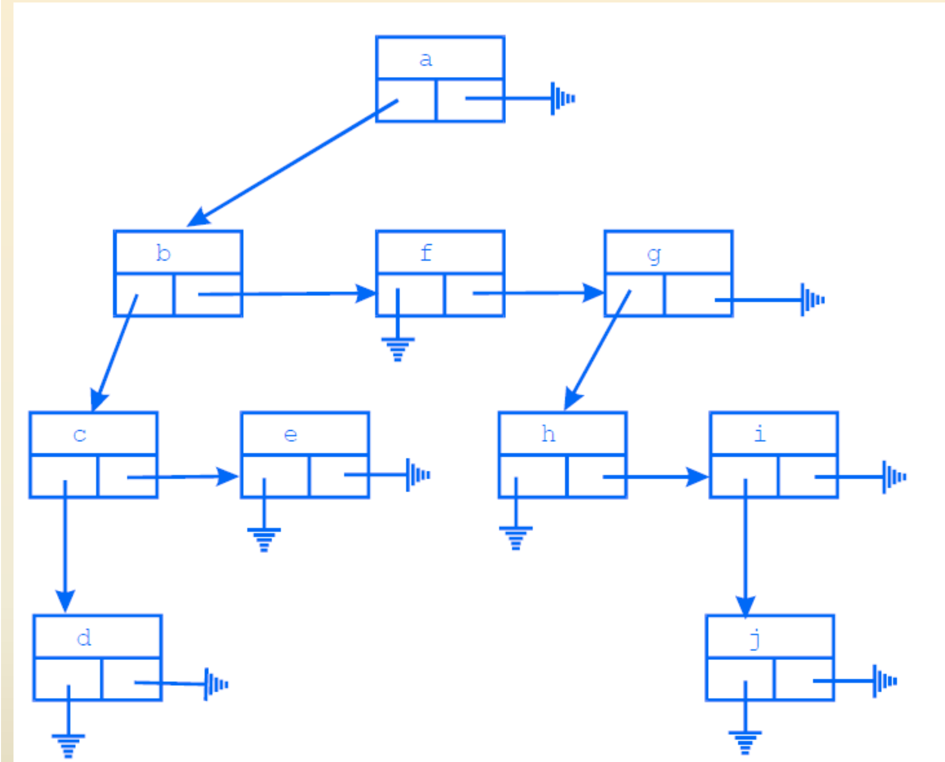
```
typedef struct arvgen {
    char info;
    arvgen *prim;
    arvgen *prox;
} ArvGen;
```

Árvore Genérica em C



Árvore Genérica em C





Árvore Genérica em C

- Uma das vantagens dessa representação é que podemos percorrer os filhos de um nó de forma sistemática, de maneira análoga ao que fizemos para percorrer os nós de uma lista simples.
- Com o uso dessa representação, a generalização da árvore é apenas conceitual, pois, concretamente, a árvore foi transformada em uma árvore binária, com filhos esquerdos apontados por prim e direitos apontados por prox.

Árvore Genérica em C

- Seguindo este exemplo da lista de filhos

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

Árvore Genérica em C

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

Árvore Genérica em C

```
/* cria nós como folhas */
ArvGen* a = cria('a');
ArvGen* b = cria('b');
ArvGen* c = cria('c');
ArvGen* d = cria('d');
ArvGen* e = cria('e');
ArvGen* f = cria('f');
ArvGen* g = cria('g');
ArvGen* h = cria('h');
ArvGen* i = cria('i');
ArvGen* j = cria('j');
```

Árvore Genérica em C

```
/* montar a hierarquia */
insere(c,d);
insere(b,e);
insere(b,c);
insere(i,j);
insere(g,i);
insere(g,h);
insere(a,g);
insere(a,f);
insere(a,b);
```

Árvore Genérica em C

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

Árvore Genérica em C

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

Árvore Genérica em C

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