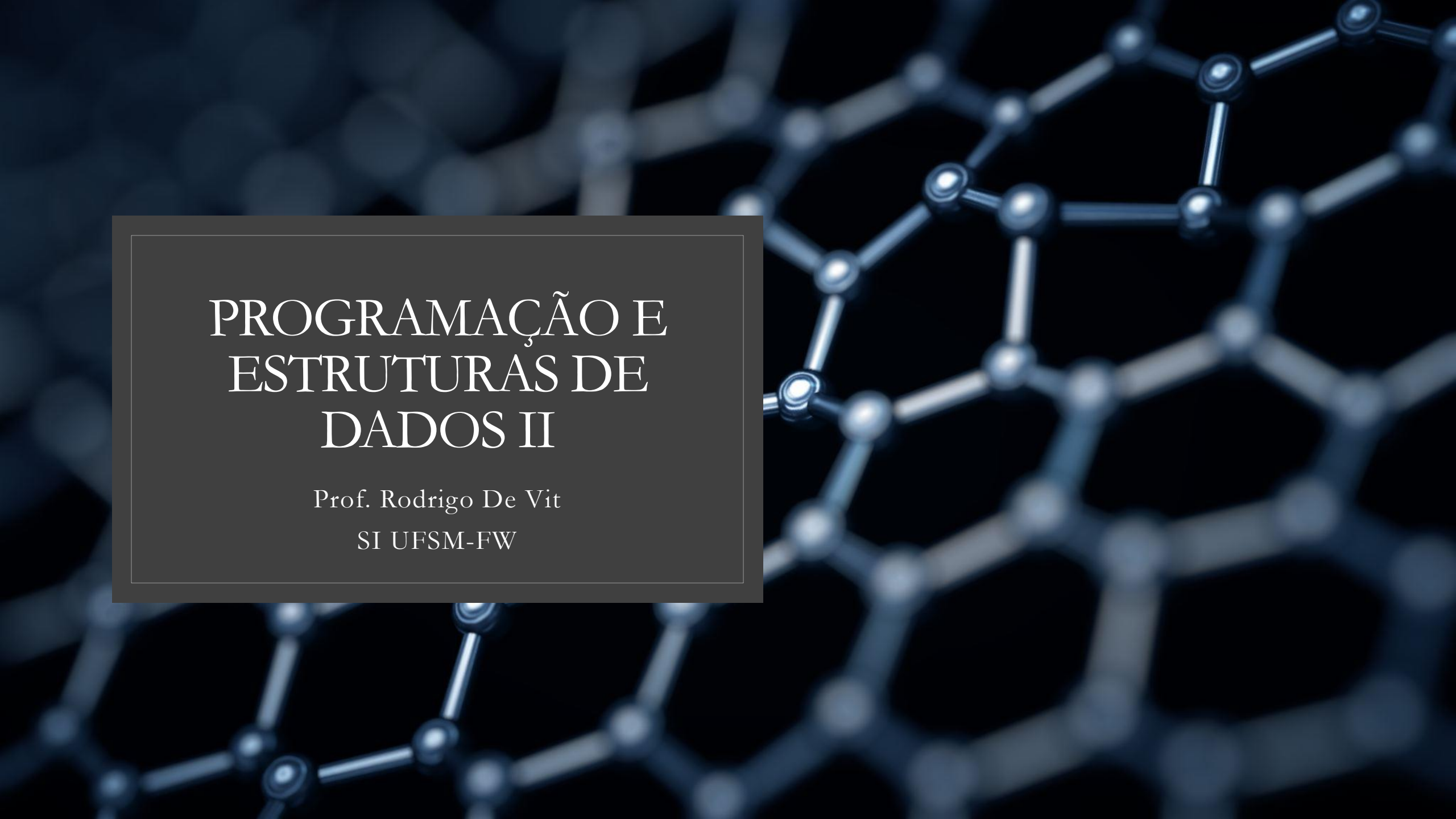


PROGRAMAÇÃO E ESTRUTURAS DE DADOS II

Prof. Rodrigo De Vit

SI UFSM-FW

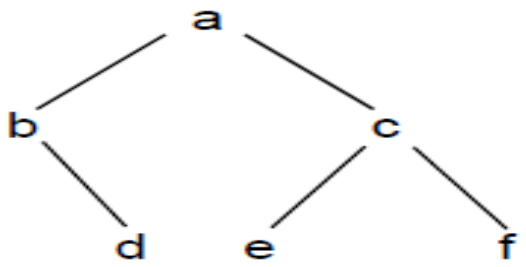


Figura 13.5: Exemplo de árvore binária

Figura 13.6: Duas árvores binárias distintas.

Árvores binárias

- Uma propriedade fundamental de todas as árvores é que só existe um caminho da raiz para qualquer nó. Com isto, podemos definir a *altura* de uma árvore como sendo o comprimento do caminho mais longo da raiz até uma das folhas. Por exemplo, a altura da árvore da Figura 13.5 é 2, e a altura das árvores da Figura 13.6 é 1. Assim, a altura de uma árvore com um único nó raiz é zero e, por conseguinte, dizemos que a altura de uma árvore vazia é negativa e vale -1.

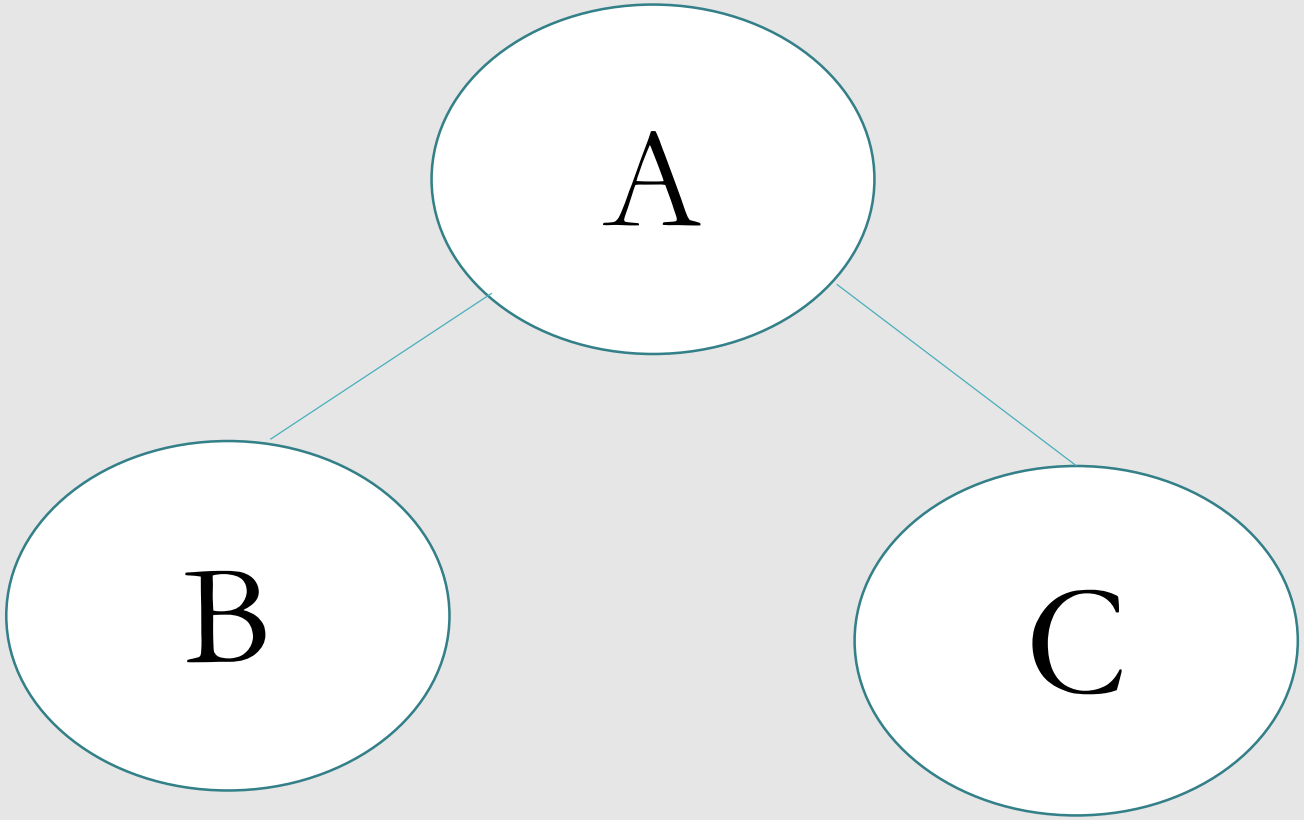
Representação em C

- Árvore binária em C.
- Enxerto e poda em árvores. (vide `main()`)

Ordens de percurso em árvores binárias

- A programação da operação imprime, vista anteriormente, seguiu a ordem empregada na definição de árvore binária para decidir a ordem em que as três ações seriam executadas. Entretanto, dependendo da aplicação em vista, esta ordem poderia não ser a preferível, podendo ser utilizada uma ordem diferente desta, por exemplo:
  - `imprime(a->esq); /* mostra sae */`
  - `imprime(a->dir); /* mostra sad */`
  - `printf("%c ", a->info); /* mostra raiz */`
- Muitas operações em árvores binárias envolvem o percurso de todas as sub-árvores, executando alguma ação de tratamento em cada nó, de forma que é comum percorrer uma árvore em uma das seguintes ordens:
  - *pré-ordem*: trata *raiz*, percorre *sae*, percorre *sad*;
  - *ordem simétrica*: percorre *sae*, trata *raiz*, percorre *sad*;
  - *pós-ordem*: percorre *sae*, percorre *sad*, trata *raiz*.

Ordens de percurso em árvores binárias



Pré: ABC
Central: BAC
Pós: BCA

Árvores Genéricas

- Cada nó pode ter um número arbitrário de filhos. Essa estrutura deve ser usada, por exemplo, para representar uma árvore de diretórios.
- 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, h e j são folhas, e tem zero filhos.
- De forma semelhante ao que foi feito no caso das árvores binárias, podemos representar essas árvores através de notação textual, seguindo o padrão: <raiz sa1 sa2 ...san>. Com esta notação, a árvore da Figura 13.7 seria representada por:

α = <a <b <c <d>> <e>> <f> <g <h> <i <j>>>>>

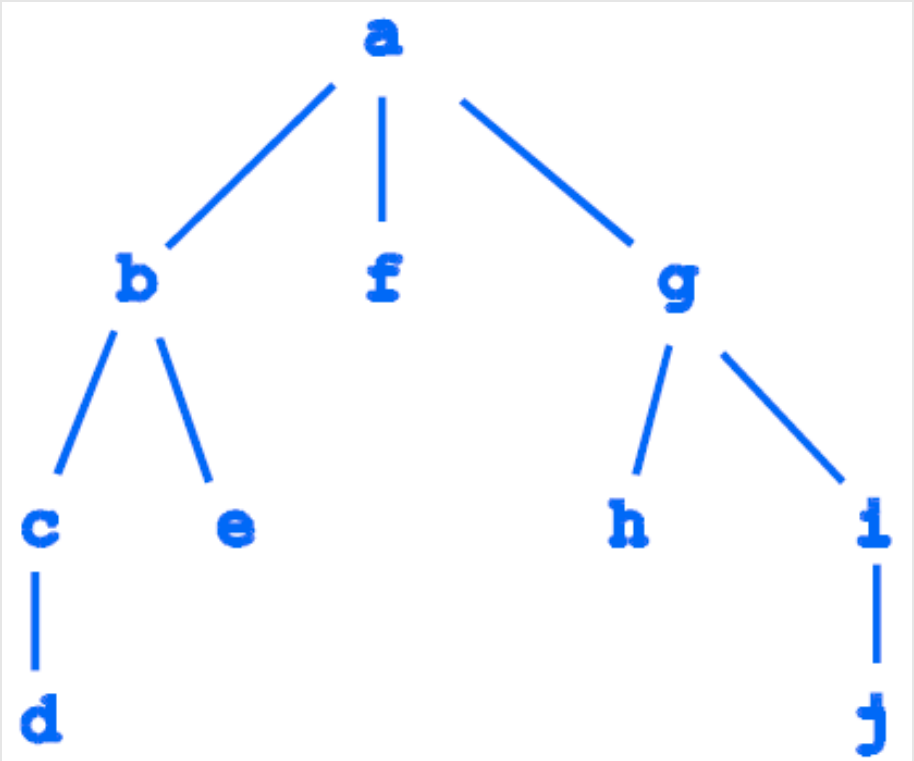


Figura 13.7: Exemplo de árvore genérica.

Árvores Genéricas

- Podemos verificar que a representa a árvore do exemplo seguindo a sequência de definição a partir das folhas:

α 1 = <d>

α 2 = <c α 1> = <c <d>>

α 3 = <e>

α 4 = <b α 2 α 3> = <b <c <d>> <e>>

α 5 = <f>

α 6 = <h>

α 7 = <j>

α 8 = <i α 7> = <i <j>>

α 9 = <g α 6 α 8> = <g <h> <i <j>>>

α = <a α 4 α 5 α 9> =

<a <b <c <d>> <e>> <f> <g <h> <i <j>>>>

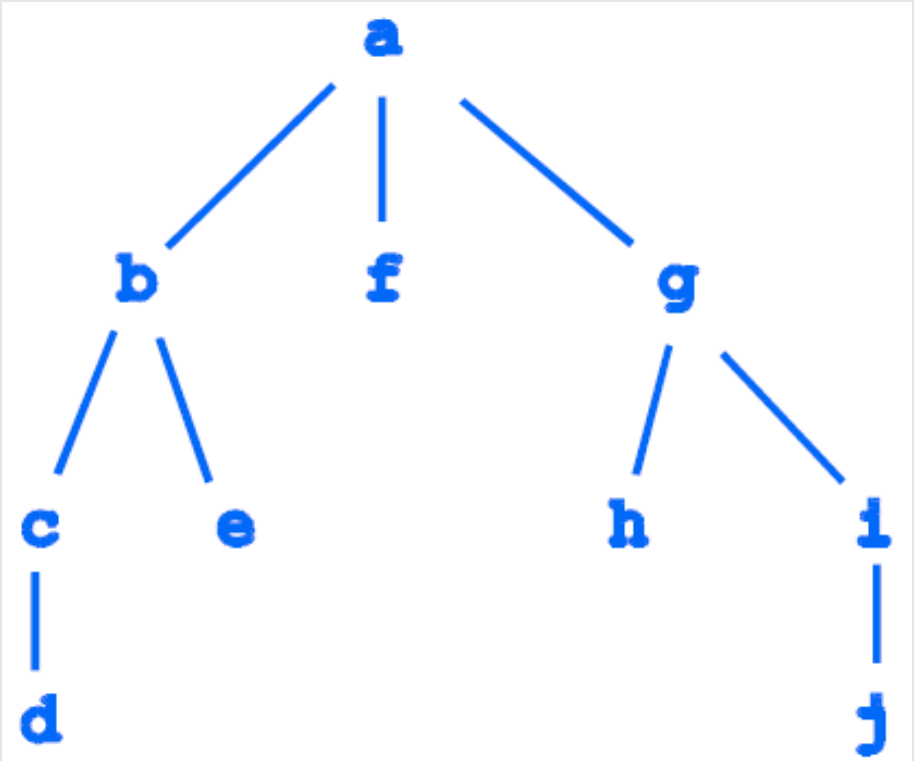


Figura 13.7: Exemplo de árvore genérica.

Representação em C

```
struct arv3 {
   char val;
   struct no *f1, *f2, *f3;
};
```

Desperdício...

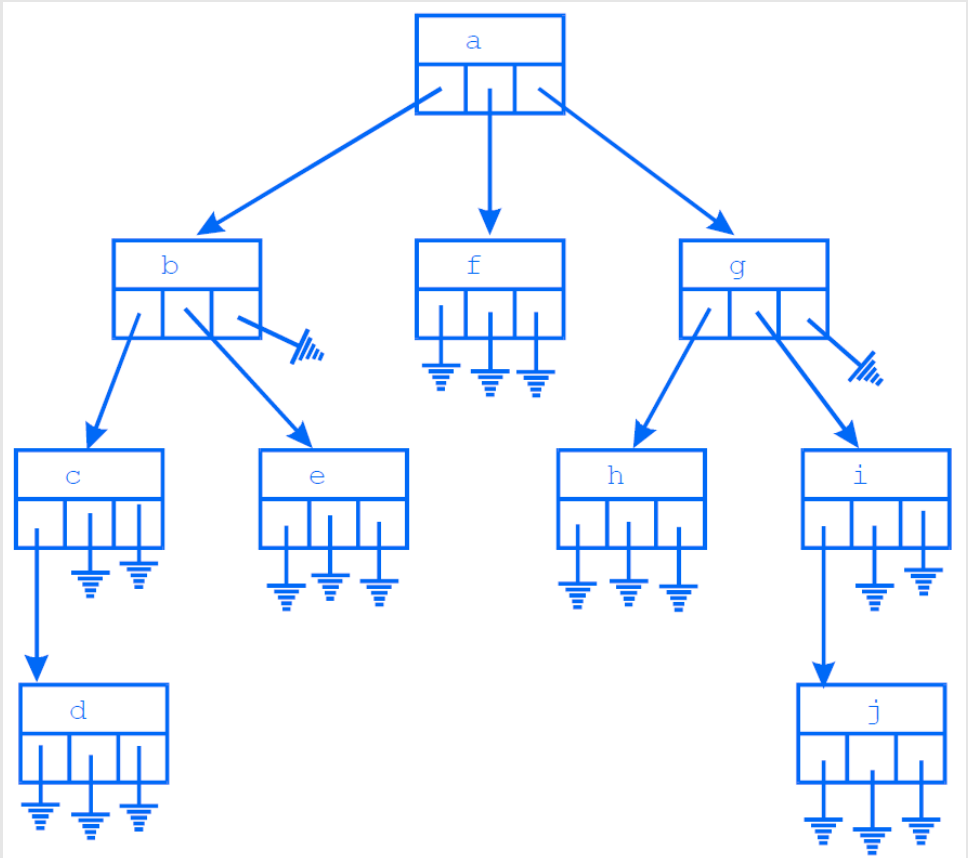


Figura 13.8: Árvore com no máximo três filhos por nó.

Representação em C

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

- 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.
- TAD:

cria: cria um nó folha, dada a informação a ser armazenada;

insere: insere uma nova sub-árvore como filha de um dado nó;

imprime: percorre todos os nós e imprime suas informações;

busca: verifica a ocorrência de um determinado valor num dos nós da árvore;

libera: libera toda a memória alocada pela árvore.

ArvGen.c

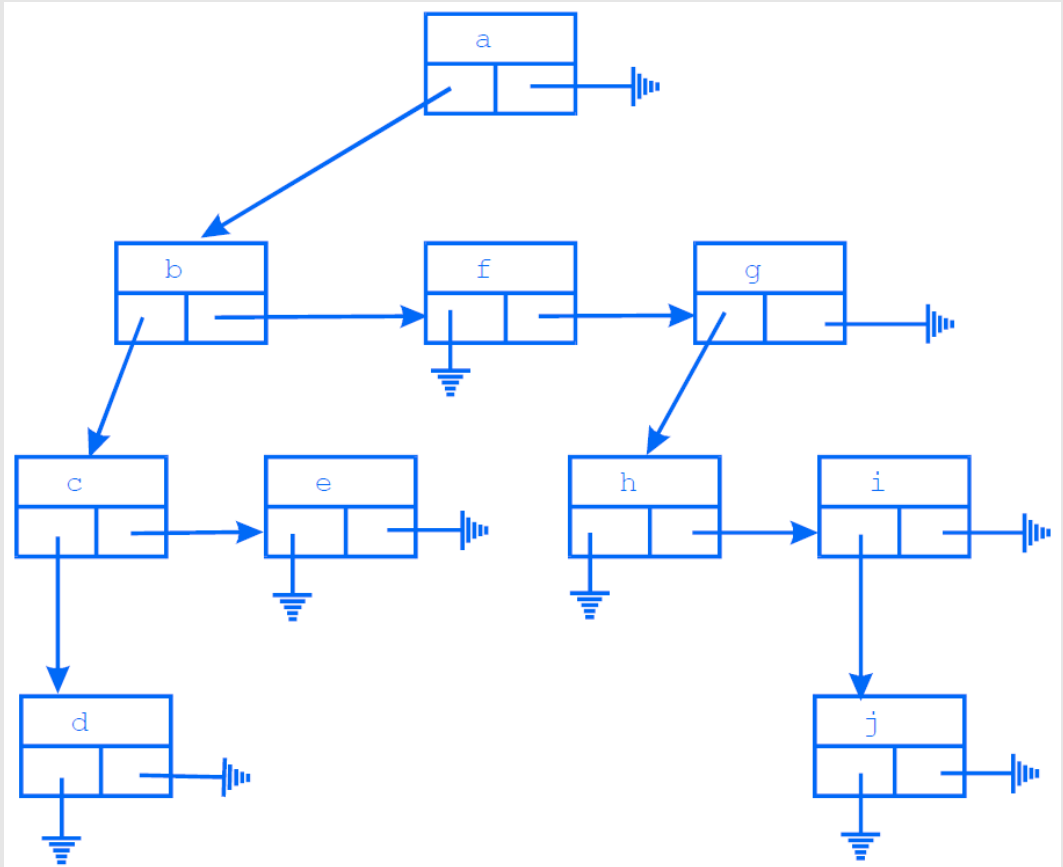


Figura 13.9: Exemplo usando "lista de filhos".