Da aula passada...

- **Árvores (TAD)**
- **Árvores Binárias**

?

“"Vivendo, se aprende; mas o que se aprende, mais, é só fazer outras maiores perguntas...."”

— Guimarães Rosa, *Grande Sertão: Veredas*

Estruturas de Dados (116319, D, 2018/1)

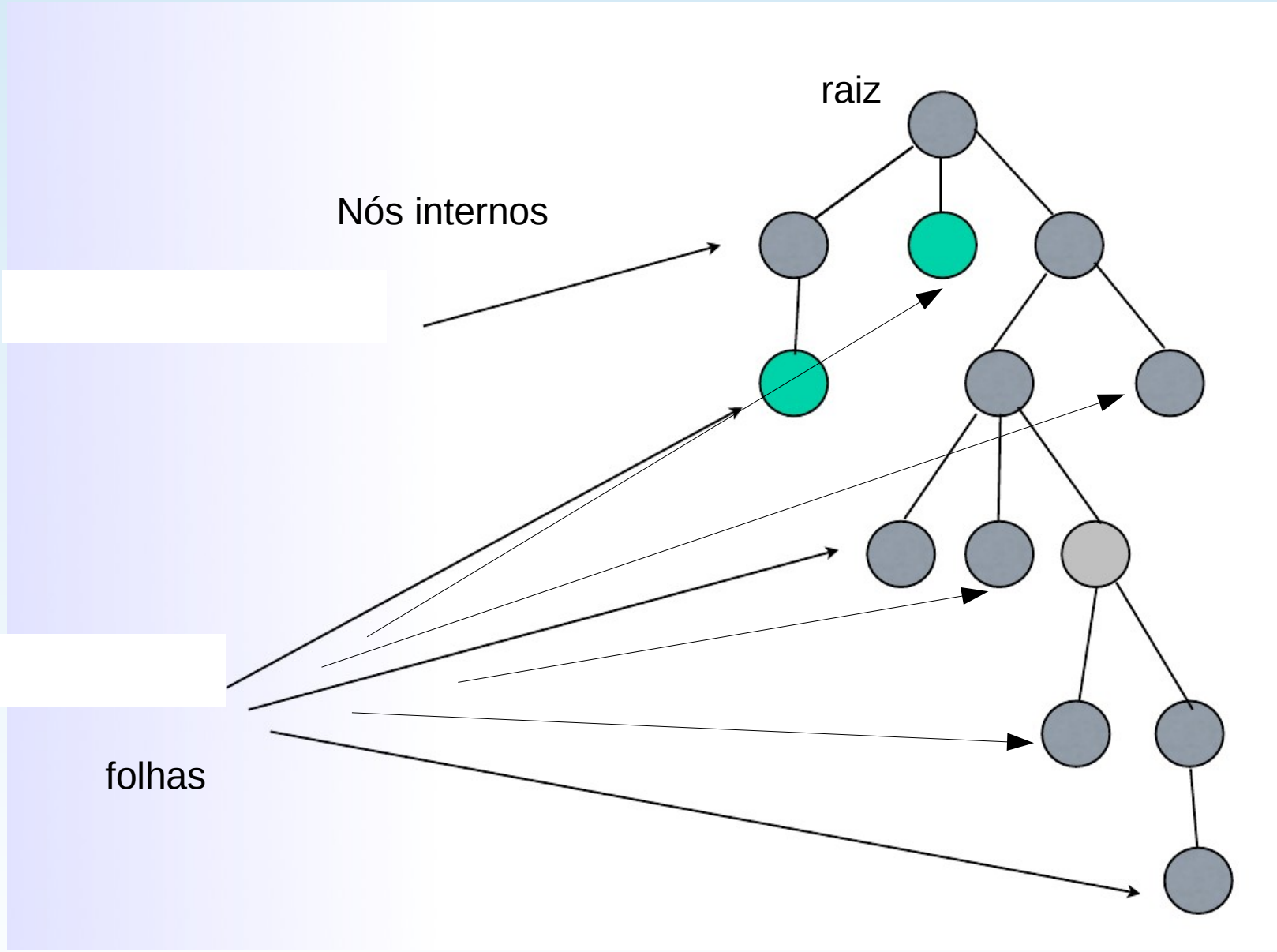
Roteiro da aula:

- Árvores Genéricas
- Exemplos

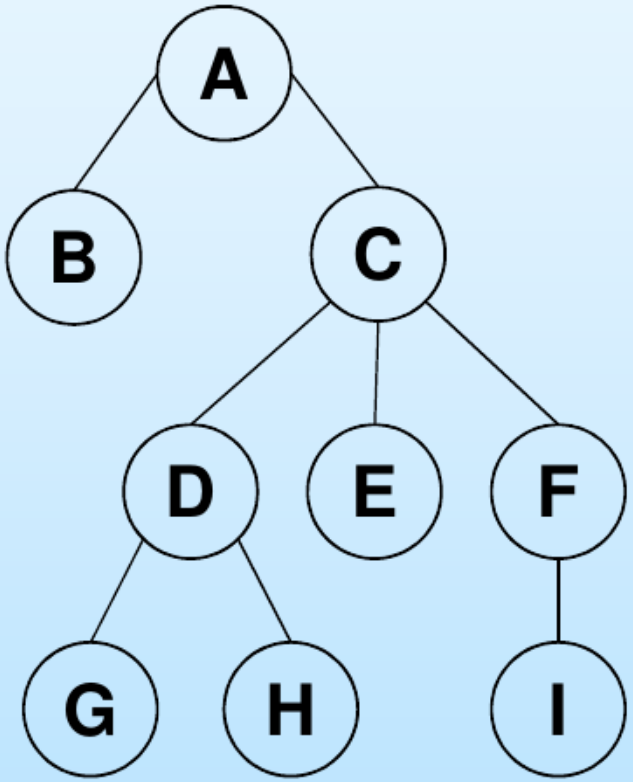
Árvores Genéricas

- **Em uma Árvore Genérica cada nó pode ter um número arbitrário de filhos.**
- **Pode ser constituída, portanto, com:**
  - **Um nó raiz.**
  - **Zero, ou mais sub-árvores.**
  - **Nessa definição não há árvore vazia, e sim uma árvore com nós folhas com zero sub-árvores.**

Árvores Genéricas



Árvores Genéricas



- **Graus dos nós**
  - G(A)=2
  - G(B)=0
  - G(C)=3
  - G(D)=2
  - G(E)=0
  - G(F)=1
  - G(G)=0
  - G(H)=0
  - G(I)=0

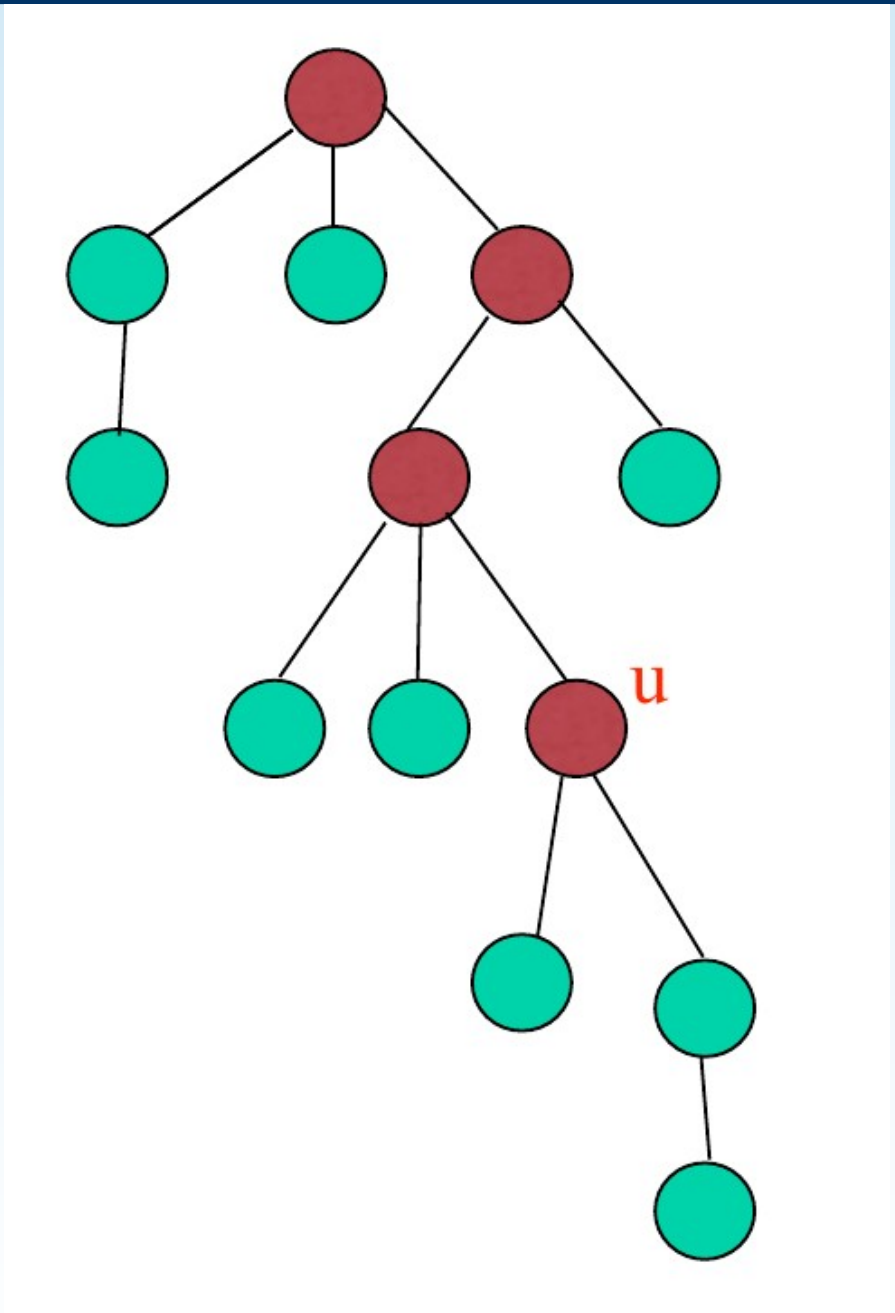
Nós Internos

Folhas

Grau(T) = 3

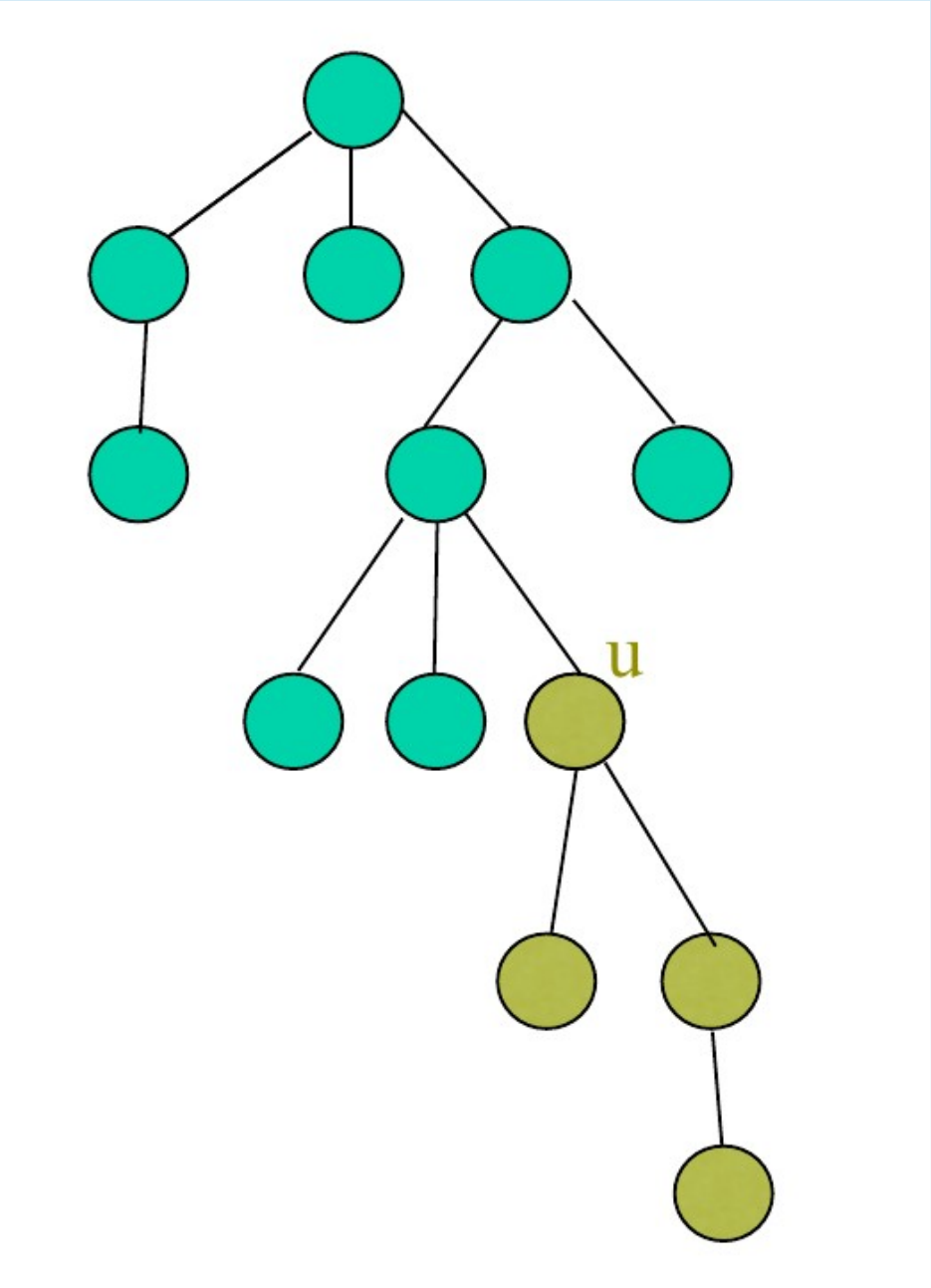
Árvores Genéricas

Ancestrais de u

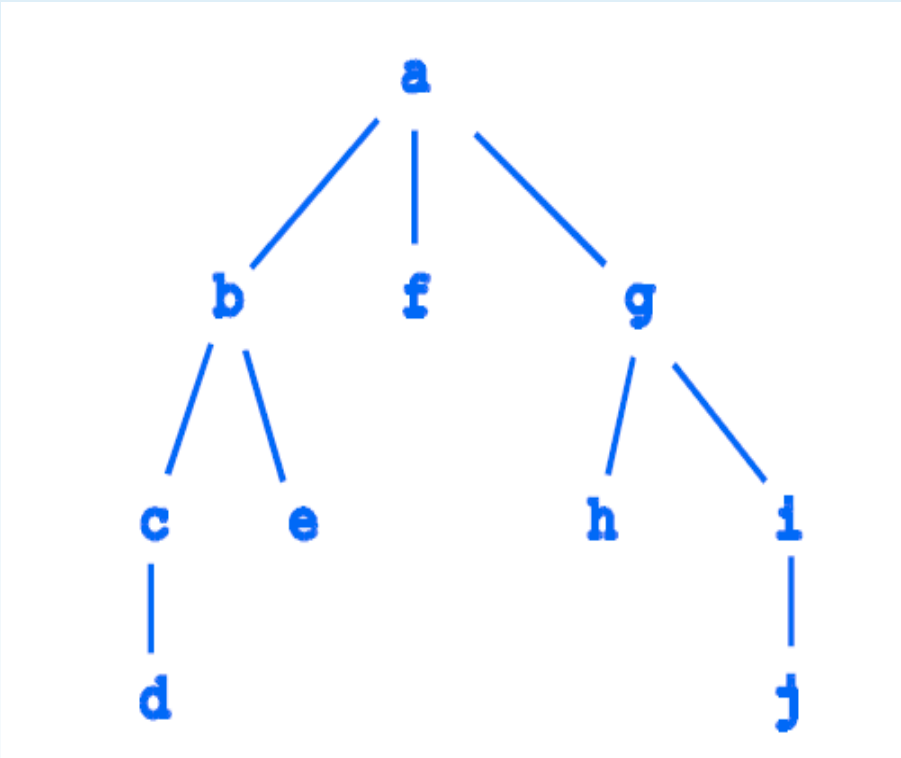


Árvores Genéricas

descendentes de u

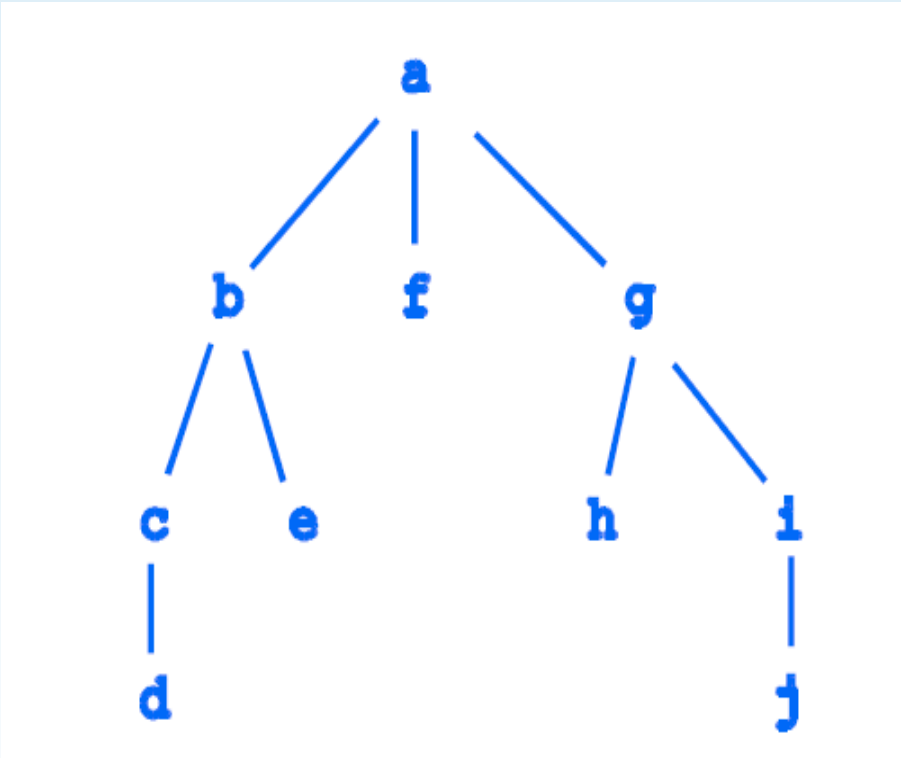


Exemplo: Árvores Genéricas



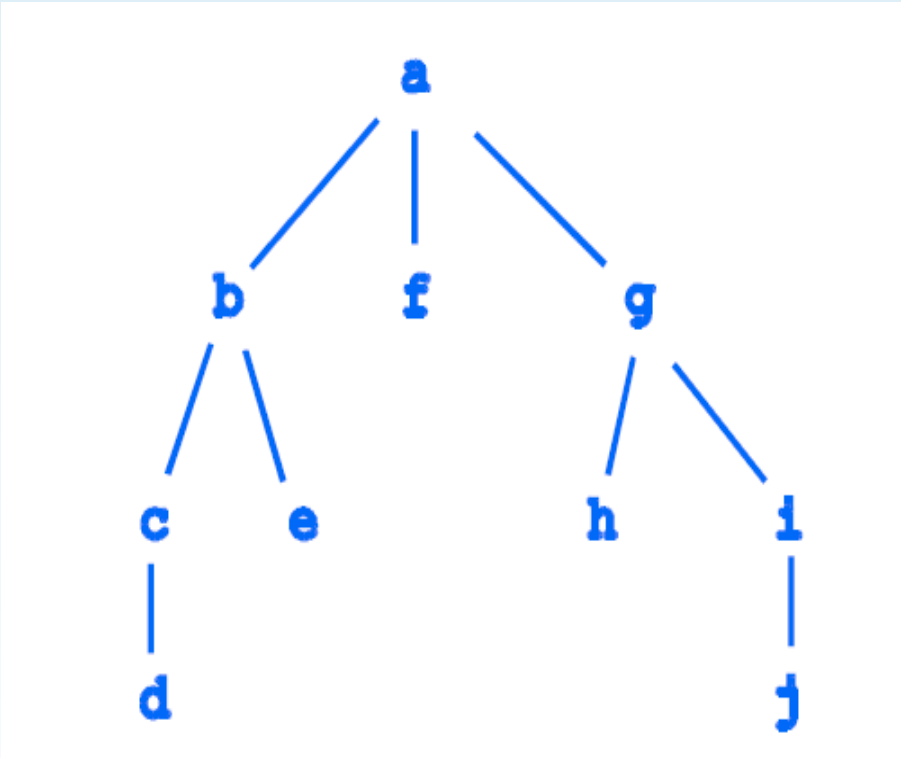
Exemplo: Árvores Genéricas

- **As sub-árvores podem ser denominadas, da esquerda para a direita, sub-árvore 1 (sa1), sub-árvore 2 (sa2), ...**



Exemplo: Árvores Genéricas

- **As sub-árvores podem ser denominadas, da esquerda para a direita, sub-árvore 1 (sa1), sub-árvore 2 (sa2), ...**



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

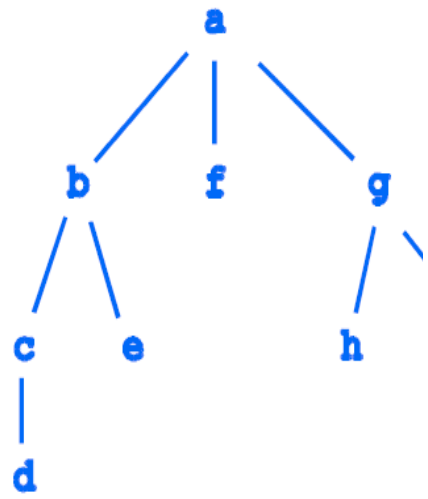
Possível representação: Árvores Genéricas

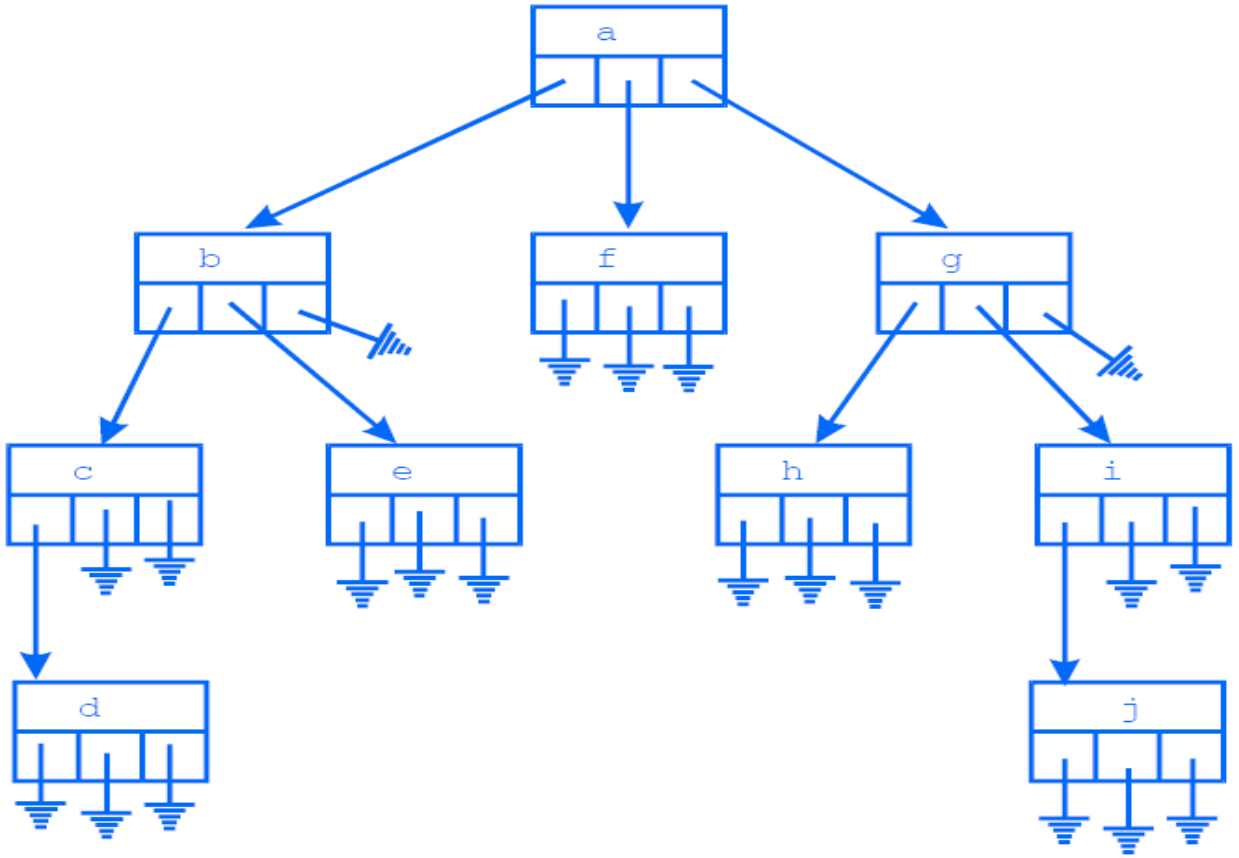
- **Exemplo prevendo um número máximo de filhos 3, e com elementos tipo char**

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

Possível representação: Árvores Genéricas

- **Exemplo prevendo um número máximo de filhos 3, e com elementos tipo char**





Possível representação: Árvores Genéricas

- **Desvantagem dessa representação?**

Possível representação: Árvores Genéricas

- **Desvantagem dessa representação?**
  - **Desperdício nos nós com menos de 3 filhos**

Possível representação: Árvores Genéricas

- **Desvantagem dessa representação?**
  - **Desperdício nos nós com menos de 3 filhos**
- **Como fazer implementação mais eficiente?**

Mais eficiente representação: Árvores Genéricas

- **Um nó aponta para o primeiro filho (prim), e cada um de seus filhos (exceto o último) aponta para o próximo (prox). Ou seja, uma lista de filhos.**

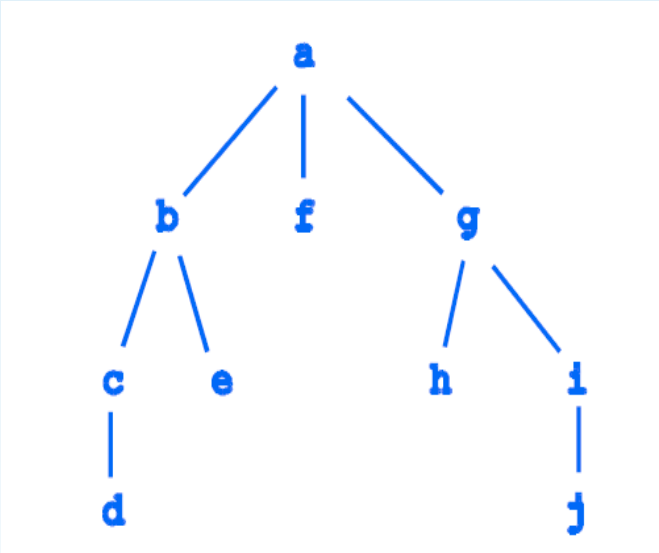
Mais eficiente representação: Árvores Genéricas

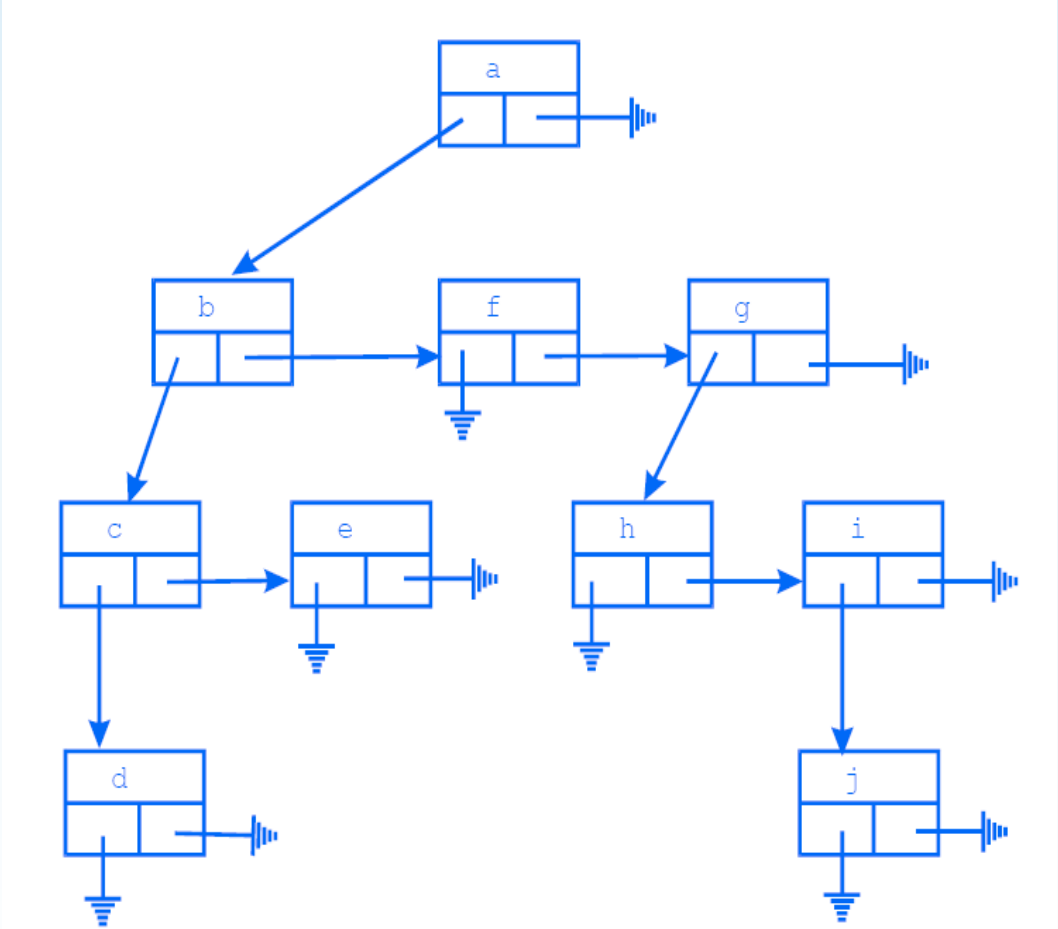
- **Um nó aponta para o primeiro filho (prim), e cada um de seus filhos (exceto o último) aponta para o próximo (prox). Ou seja, uma lista de filhos.**

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

Mais eficiente representação: Árvores Genéricas

- **Um nó aponta para o primeiro filho (prim), e cada um de seus filhos (exceto o último) aponta para o próximo (prox). Ou seja, uma lista de filhos.**





Como uma árvore binária, mas com significado de genérica.

Árvores Genéricas (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 em um dos nós da árvore.**
- **Libera: libera toda a memória alocada pela árvore.**

Árvores Genéricas (TAD)

- **Em arvgen.h, poderia ser, por exemplo:**

```
typedef struct arvgen ArvGen;

ArvGen* cria (char c);
void    insere (ArvGen* a, ArvGen* sa);
void    imprime (ArvGen* a);
int     busca (ArvGen* a, char c);
void    libera (ArvGen* a);
```

Árvores Genéricas (TAD)

- **Função de criação**

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

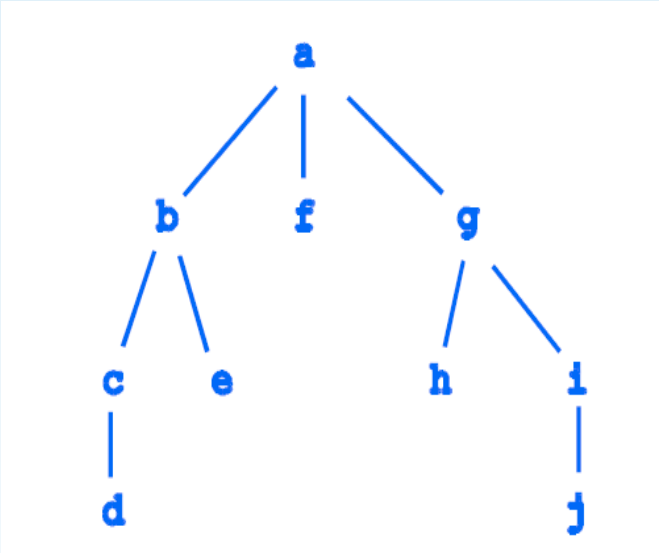
Árvores Genéricas (TAD)

- **Função de inserção (sempre no início)**

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

Árvores Genéricas (TAD)

- **Exemplo:**




```
/* 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');
/* monta 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);
```

Árvores Genéricas (TAD)

- **Impressão (pré-ordem)**

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

Árvores Genéricas (TAD)

- **Buscar um elemento na árvore**

```
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;
}
```

Árvores Genéricas (TAD)

- **Liberar espaço de árvore (sub-árvores antes, e usando pós-ordem)**

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

Referências Bibliográficas

- Cormen, T. & Leiserson, C. & Rivest, R. & Stein, C. *Introduction to Algorithms.* MIT Press. 2009.
- Sedgewick, R. *Algorithms in C* (Parts 1-4, Part 5), 2nd ed., Addison Wesley, 1997.
- Rangel, J.; Cerqueira, R.& Celes, W. *Estruturas de dados: uma introd. com téc. de programação em C*, Elsevier, 2004.
- Tenenbaum, A.; Langsam, Y. & Augenstein, M. *Estruturas de dados usando C*, Makron Books, 1995.
- Ziviani, N. *Projeto de Algoritmos: com implementações em Pascal e C,* 2a. ed., Thompson, 2007.