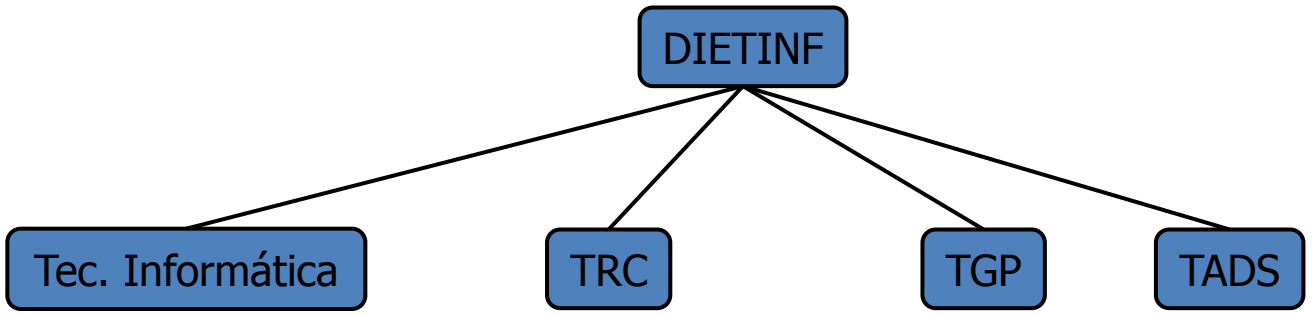
Motivação

- Uma das estrutura de dados não-lineares mais importantes da computação.
- Diversas aplicações necessitam de estruturas mais complexas que as listas estudadas até agora, como listas e filas.
- Diversos problemas podem ser modelados através de árvores.
- Permitem uma gama de algoritmos muito mais rápidos do que no uso de estruturas de dados lineares, tais como listas.

Árvore

- Contêiner onde objetos são armazenados de forma hierárquica (relação pai-filho)
- O elemento no topo da árvore é chamado ***raiz***
- Cada elemento tem ***um pai*** e ***zero ou mais filhos***, com exceção da raiz que não possui pai

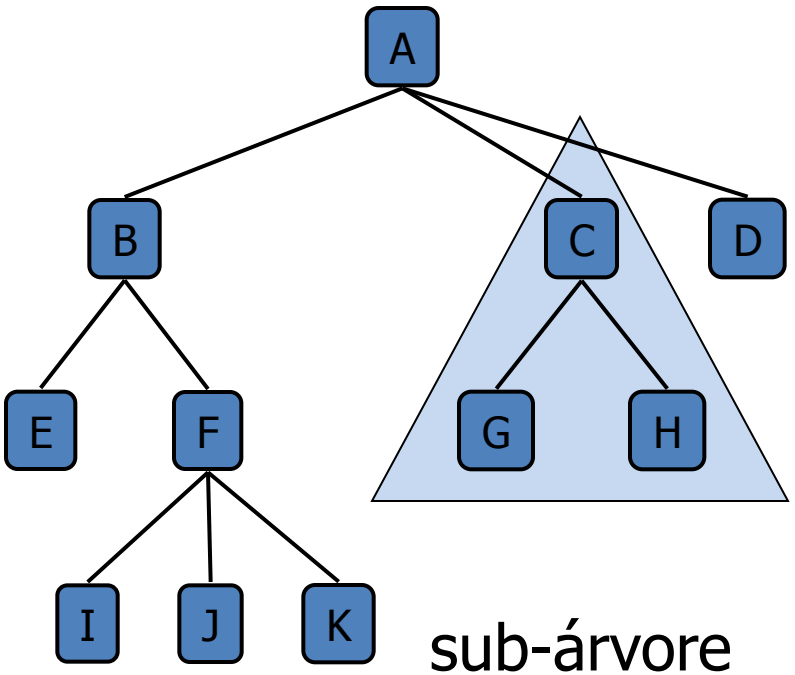


Aplicações

- Representação de hierarquias
  - Relação de hierarquia entre classes Java e .Net
  - Sistemas de arquivos
  - Representação de documentos (livro, capítulo, ...)
  - Respostas a questões sim/não: árvore de decisão
  - Expressões aritméticas
  - Relacionamento entre tabelas em um banco de dados (grupos, subgrupos, produtos)

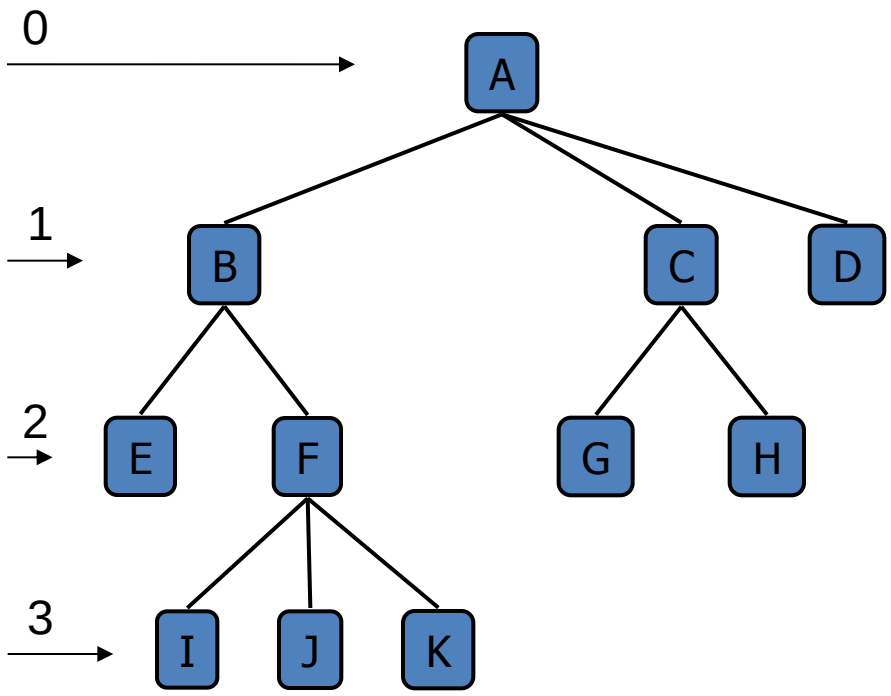
Terminologia

- **Raiz** (root): nó sem pai (A)
- **Nó interno**: nó com pelo menos um filho (A, B, C, F)
- Nó externo ou nó **folha**: nó sem filhos (D, E, G, H, I, J, K)
- **Ancestral** de um nó: pai, avô, bisavô, ...
- **Descendente** de um nó: filho, neto, bisneto, ...
- **Sub-árvore**: árvore formada por um nó e seus descendentes



Terminologia

- **Profundidade** de um nó: número de ancestrais
- **Altura** de um nó: número de níveis de descendentes
  - Altura(A) = 3
  - Altura(B) = 2
  - Altura(C) = 1
  - Altura(D) = 0
- Altura da árvore: profundidade máxima considerando todos os nós folhas



Operações Genéricas

- Métodos genéricos
  - int size()
    - retorna o número de nós da árvore
  - boolean isEmpty
    - indica se a árvore é vazia (nunca é true)
  - Iterator elements()
    - retorna um iterador para os elementos da árvore
  - Iterator Nós()
    - retorna um iterador para os nós da árvore

Operações de Atualização e Acesso

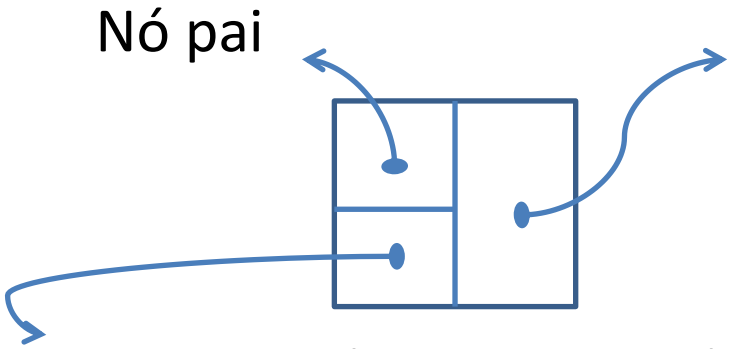
- Métodos de atualização
  - Object replaceElement(Nó v, Object e)
    - substitui o elemento do nó retornando o elemento original
  - void swapElements(Nó v, Nó w)
    - Troca os elementos de dois nós
- Métodos de acesso
  - Nó root()
    - retorna o nó raiz
  - Nó parent(Nó v)
    - retorna o pai de um nó

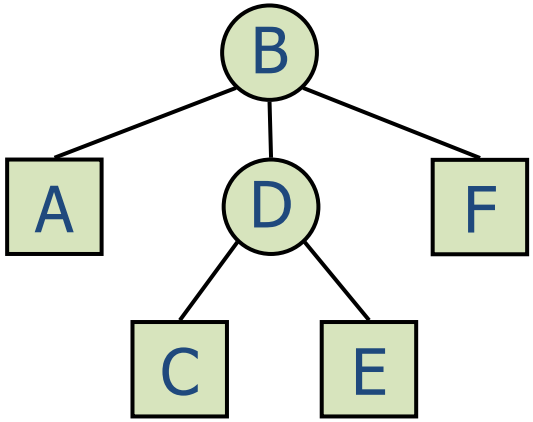
Operações de Acesso e Expansão

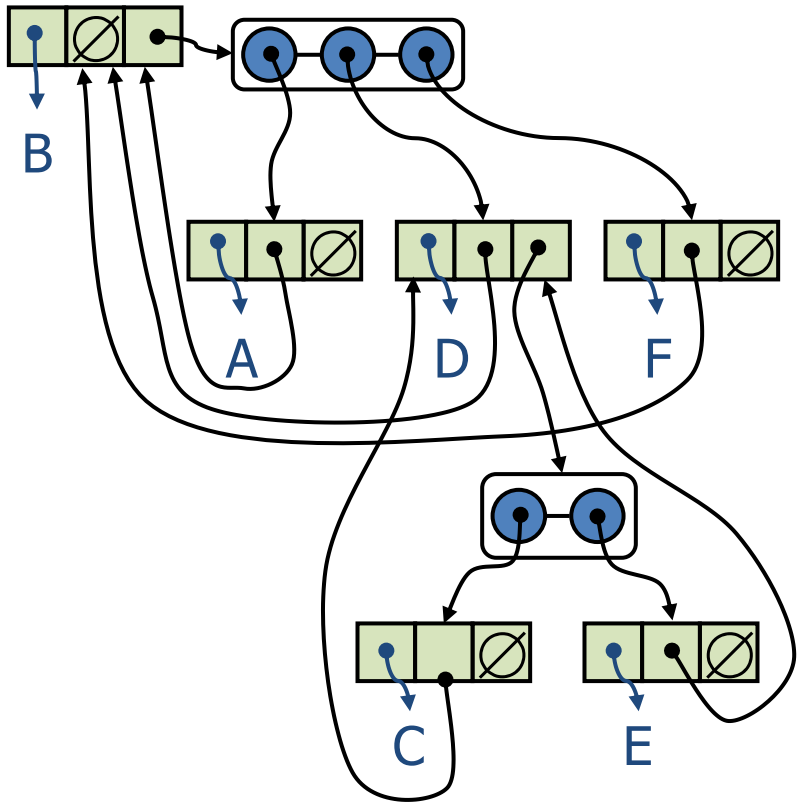
- Métodos de acesso - continuação
  - Iterator children(Nó v)
    - retorna um iterador para os filhos de um nó
  - boolean isInternal(Nó v)
    - testa se um nó é interno
  - boolean isExternal(Nó v)
    - testa se um nó é externo
  - boolean isRoot(Nó v)
    - testa se um nó é raiz
- Métodos de expansão
  - Definidos de acordo com a aplicação da árvore

Estrutura Encadeada para Árvores Genéricas

- Um nó armazena referências para:







Interface Árvore

```
import java.util.Iterator;
public interface Árvore
{
        /* Métodos genéricos */
        public int size(); /** Retorna o número de nós da árvore */
        public boolean isEmpty(); /** retorna se a árvore está vazia */
        public int height(); /** Retorna a altura da árvore */
        public Iterator elements(); /** Retorna um iterator com os elementos */
        public Iterator Nós(); /** Retorna um iterator com os nós */
        /* Métodos de acesso*/
        public Nó root(); /** Retorna a raiz da árvore */
        public Nó parent(Nó v); /** Retorna o nó pai de um nó */
        public Iterator children(Nó v); /** Retorna os filhos de um nó */
        /* Métodos de consulta */
        public boolean isInternal(Nó v); /** Testa se um nó é interno */
```

Interface Árvore

```
    public boolean isExternal(Nó v); /** Testa se um nó é externo*/
    public boolean isRoot(Nó v); /** Testa se um nó é a raiz */
    public int depth(Nó v); /** Retorna a profundidade de um nó */
    /* Métodos de atualização */
    public Object replace(Nó v, Object o);
}
```

Interface ÁrvoreGenérica

```
public interface ArvoreGenerica extends Arvore
{
        public void addChild(Nó v, Object o);
        public Object remove(Nó v) throws InvalidNóException;
}
```

Como implementar uma árvore genérica?

- Criar a Interface **Árvore (ver acima)**
- Criar a classe **Nó**
  - Deve armazenar uma referência para o pai (parent), uma para o elemento e para uma coleção para armazenar as conexões com os nodos filhos.
- Criar interface **ÁrvoreGenérica** que implementa **Árvore**
- Criar a classe **ÁrvoreSimples** que implementa **ÁrvoreGenérica**.
  - Deve armazenar uma referência para a raiz da árvore e o número de nodos da árvore

Algoritmos

- Muitos algoritmos utilizam recursividade
  - Profundidade
  - Altura
  - Caminhamento prefixado – pré-ordem
  - Caminhamento pós-fixado – pós-ordem

Profundidade

- A profundidade (n° de ancestrais) de um nó *v* pode ser definida recursivamente como:
  - Se *v* for raiz, então a profundidade é 0
  - Senão, a profundidade é 1 mais a profundidade do pai de *v*

Algoritmo depth (Árvore T, Nó v)

se (isRoot(v)) retorne 0

senão retorne 1 + depth (T, parent(v))

Altura

- A altura (n° de níveis descendente) de um nó *v* pode ser definida recursivamente como:
  - Se *v* for externo, então a altura é 0
  - Senão, a altura é 1 mais a maior altura de um filho de *v*

Algoritmo height (Tree T, Nó v)

```
se (T.isExternal(v)) retorne 0
senão
    h = 0
    para cada w em T.children(v) faça h = max(h, height (T, w))
    retorne 1 + h
```

Caminhamento

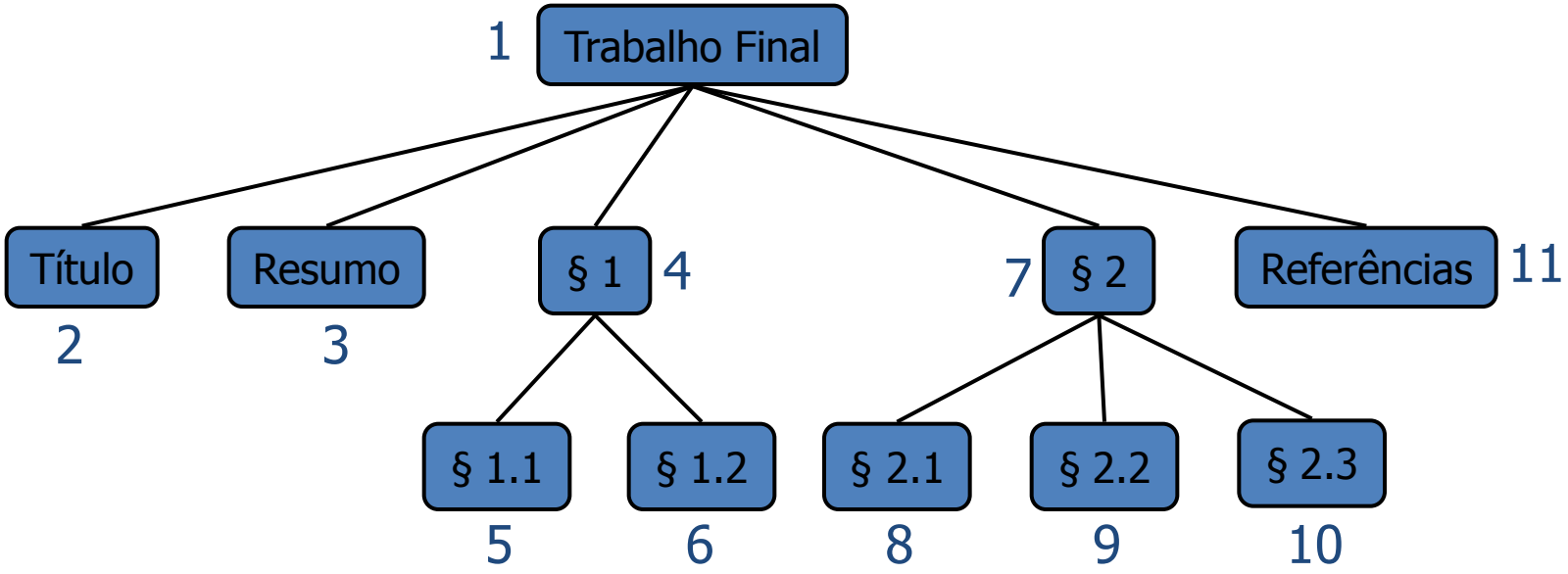
- O caminhamento é uma visita a todos os nós de uma árvore de forma sistemática
- Caminhamento prefixado: cada nó é visitado antes de seus descendentes
  - Aplicação: imprimir um documento estruturado
- Caminhamento pós-fixado: cada nó é visitado depois de seus dependentes
  - Aplicação: computar o espaço utilizado em uma árvore de diretórios

Caminhamento Prefixado

Algoritmo preOrder(T, v)

execute a ação para o nó v

para cada w em T.children(v) faça preOrder (T, w)



Caminhamento Pós-fixado

Algoritmo postOrder(T, v)

para cada w em T.children(v) faça postOrder (T, w)

execute a ação para o nó v