IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Programação Avançada

Implementação do TAD Tree
Programação Avançada 2020-21

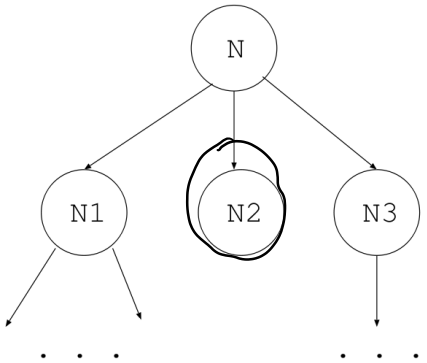
Bruno Silva, Patrícia Macedo

Sumário 

- Especificação do TAD Tree
- Implementação em Java
  - Tipo de Dados Position
  - Interface Tree
  - TreeNode
  - TreeLinked

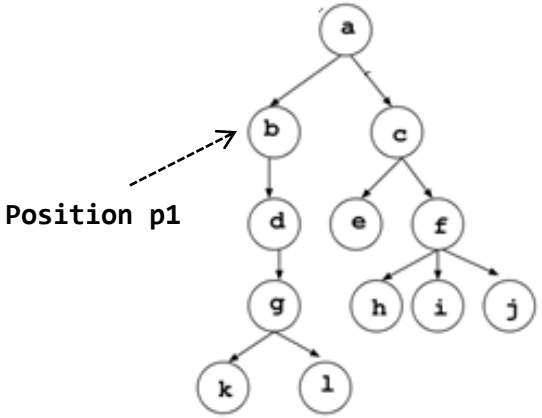
Árvores – Conceitos (revisão)

- Uma árvore é composta por ***nós***;
- No topo da árvore existe um nó especial, chamado ***raiz***;
  - Não possui ascendentes.
- Todos os outros nós têm exatamente um ascendente direto;
- Na Figura, dizemos que N_1, N_2 e N_3 são ***filhos*** de N. Consequentemente, N é ***pai*** de N_1, N_2 e N_3.
- Nós que não têm descendentes são chamados nós ***externos*** ou ***folhas***.
  - Ex.: N_2.
- Nós que não são a raiz nem folhas, são chamados ***internos***.
  - Ex.: N_1 e N_3.

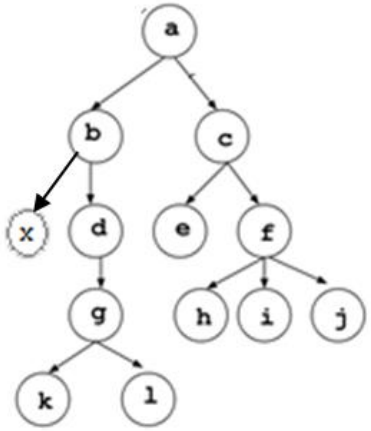


Posição - Conceito

- De forma a podermos facilmente referenciar os nós da árvore sem expormos a usa implementação, introduzimos a noção de Posição (Position).
- A **Posição** modela a noção de **"lugar"** dentro de uma estrutura de dados, onde um único objeto é armazenado.
- É através da referencia para o lugar na árvore que se insere e remove elementos numa árvore.



`insert(x,p1)`



TAD Tree - especificação

- Operações Modificadoras
  - **replace (x, pos):** substitui o elemento que se encontra na posição pos pelo valor **x**, devolve erro se for uma posição que não pertence à árvore.
  - **insert(x,pos):** insere um elemento *x* como sendo filho do nó da árvore que está na posição pos, devolve erro se for uma posição que não pertence à árvore.
  - **insert(x,n,pos):** insere um elemento *x* como sendo o n-ésimo filho do nó da árvore que está na posição pos, devolve erro se for uma posição que não pertence à árvore, ou se n for um numero superior ao numero de filhos daquele nó.
  - **remove(pos):** remove o nó de uma determinada posição pos, devolve erro se for uma posição que não pertence à árvore.

TAD Tree - especificação

- Operações de Verificação
  - **isInternal (pos) :** verifica se a posição **pos** se refere a um nó interno, devolve erro se for uma posição que não pertence à árvore.
  - **isExternal (pos):** verifica se verifica se a posição **pos** se refere a um nó externo, devolve erro se for uma posição que não pertence à árvore.
  - **isRoot(pos)**: verifica se a posição **pos** se refere a um nó do tipo raiz, devolve erro se for uma posição que não pertence à árvore.
  - **isEmpty:** verifica se a árvore está vazia.

TAD Tree - especificação

- Operações Selectoras:
  - **size:** devolve o tamanho da árvore.
  - **elements:** devolve uma coleção iterável com os elementos da árvore.
  - **positions:** devolve uma coleção iterável de posições da árvore.
  - **root**: devolve a raiz da árvore
  - **parent (pos):** devolve a posição do nó pai do nó que se encontra na posição pos, devolve erro se for uma posição que não pertence à árvore.
  - **children (pos)** : devolve a coleção dos filhos de um nó da árvore que se encontra na posição pos, devolve erro se for uma posição que não pertence à árvore.
  - **remove(pos):** remove o nó de uma determinada posição pos, devolve erro se for uma posição que não pertence à árvore.

TAD Position (Posição)

- Modela a noção de **"lugar"** dentro de uma estrutura de dados, onde um único objeto é armazenado;
- Possui apenas um método:
  - E element(): retorna o elemento armazenado nessa posição.

```
public interface Position<E>{
    public E element() throws InvalidPositionException;
}
```

TAD Tree - Interface

```
public interface Tree<E> {
    public int size();

    public boolean isEmpty();

    public Iterable<Position<E>> positions();

    public Iterable<E> elements();

    public E replace(Position<E> position, E elem) throws InvalidPositionException;

    public Position<E> root() throws EmptyTreeException;

    public Position<E> parent(Position<E> position)    throws InvalidPositionException, BoundaryViolationException;

    public Iterable<Position<E>> children(Position<E> position) throws InvalidPositionException;

    public boolean isInternal(Position<E> position) throws InvalidPositionException;

    public boolean isExternal(Position<E> position)  throws InvalidPositionException;

    public boolean isRoot(Position<E> position)   throws InvalidPositionException;

    public Position<E> insert(Position<E> parent, E elem, int order)throws InvalidPositionException,BoundaryViolationException;

    public Position<E> insert(Position<E> parent, E elem) throws InvalidPositionException;

    public E remove(Position<E> position)throws InvalidPositionException;

    public int height();
}
```

TAD Tree: Aplicação

```
TREE Animal
  -Mamifero
    -Cao
    -Gato
    -Vaca
  -Ave
    -Papagaio
    -Aguia
      -Aguia Real
```

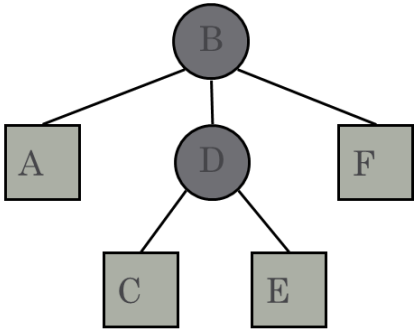
Taxonomia de Animais:

- Construir uma árvore, usando a noção de Posição (Position)
- Após a inserção de um elemento na árvore a posição onde ficou o elemento é devolvida.
- Quando se insere um elemento indica-se qual a posição do pai.

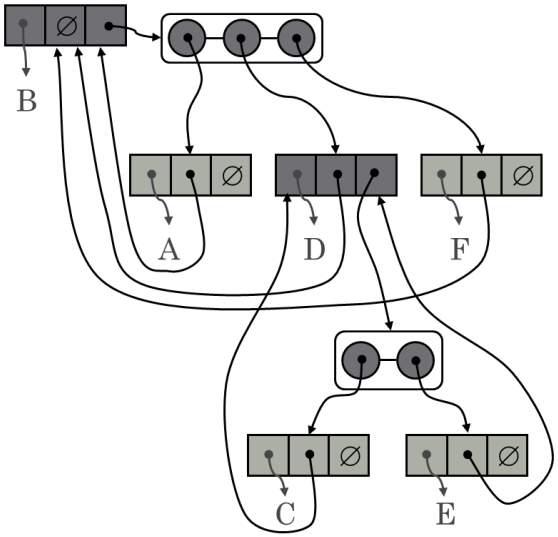
```
public class TADTreeMain {

    public static void main(String[] args) {
        TreeLinked<String> myTree = new TreeLinked("Animal");
        Position<String> root = myTree.root();
        Position<String> posMamifero = myTree.insert(root, "Mamifero");
        Position<String> posAve = myTree.insert(root, "Ave");
        myTree.insert(posMamifero, "Cao");
        Position<String> posGato = myTree.insert(posMamifero, "Gato");
        myTree.insert(posMamifero, "Vaca");
        myTree.insert(posAve, "Papagaio");
        Position<String> posAguia = myTree.insert(posAve, "Aguia");
        myTree.insert(posAguia, "Aguia Real");
        System.out.println("TREE " + myTree);
        System.out.println(myTree);
    }
}
```

TAD Tree Implementação: Usando uma estrutura de nós ligados

- Uma árvore é composta por um conjunto de nós interligados entre si.
- Um nó de uma árvore guarda:
  - element - o elemento
  - parent - referência para o nó pai.
  - children - Uma referencia para a lista de nós filhos



árvore



estrutura de dados da árvore

TAD Tree Implementação: Usando uma estrutura de nós ligados

```
private class TreeNode implements Position<E> {

    private E element;  // element stored at this node
    private TreeNode parent;  // adjacent node
    private List<TreeNode> children;  // children nodes

    TreeNode(E element) {
        this.element = element;
        parent = null;
        children = new ArrayList<>();
    }

    TreeNode(E element, TreeNode parent) {
        this.element = element;
        this.parent = parent;
        this.children = new ArrayList<>();
    }

    public E element() {
        if (element == null) {
            throw new InvalidPositionException();
        }
        return element;
    }
```

Lista de filhos inicializada a vazia. Evita verificações desnecessárias, quando se pretende adicionar um filho

TAD Tree Implementação: Usando uma estrutura de nós ligados

```
private class TreeNode implements Position<E> {

    private E element;  // element stored at this node
    private TreeNode parent;  // adjacent node
    private List<TreeNode> children;  // children nodes

    TreeNode(E element) {
        this.element = element;
        parent = null;
        children = new ArrayList<>();
    }

    TreeNode(E element, TreeNode parent) {
        this.element = element;
        this.parent = parent;
        this.children = new ArrayList<>();
    }

    public E element() {
        if (element == null) {
            throw new InvalidPositionException();
        }
        return element;
    }
```

Lista de filhos inicializada a vazia. Evita verificações desnecessárias, quando se pretende adicionar um filho

TreeLinked : Atributos e Construtor

```
public class TreeLinked<E> implements Tree<E> {


    private TreeNode root;

    public TreeLinked() {
     this.root=null;
    }

    public TreeLinked(E root) {
        this.root = new TreeNode(root);
    }
```

Conversão Position ->TreeNode

- Se observarmos a interface, verifica-se que a maior parte dos métodos tem um parâmetro do tipo Position, parâmetro esse que se refere indiretamente a um nó da árvore.
- A *inner* classe TreeNode é do tipo Position, logo é possível fazer um "cast" para o tipo Position.
- O método *checkPosition* é um método auxiliar que recebe uma posição valida-a e "converte-a" num TreeNode.

```
private TreeNode checkPosition(Position<E> position)
        throws InvalidPositionException {
    if (position == null) {
        throw new InvalidPositionException();
    }

    try {
        TreeNode treeNode = (TreeNode) position;
        if (treeNode.children == null) {   // a lista de filhos nunca pode ser null
            throw new InvalidPositionException("The position is invalid");
        }
        return treeNode;
    } catch (ClassCastException e) {
        throw new InvalidPositionException();
    }
}
```

TreeLinked : insert

- Para inserir um elemento na árvore genérica, é necessário indicar a posição do nó pai.
- Adiciona-se **um nó** à lista de nós filhos, na ordem especificada no parâmetro order.

```
public Position<E> insert(Position<E> parent, E elem, int order)
        throws BoundaryViolationException, InvalidPositionException {
    if(isEmpty()){
        if( parent!= null) throw new InvalidPositionException("Pai não é nulo");
        if (order != 0 ) throw new BoundaryViolationException("Fora de limites");
        this.root = new TreeNode(elem);
        return root;
    }
    TreeNode parentNode = checkPosition(parent);
    if (order < 0 || order > parentNode.children.size()) {
        throw new BoundaryViolationException("Fora de limites");
    }
    TreeNode node = new TreeNode(elem, parentNode);
    parentNode.children.add(order, node);
    return node;

}
```

TreeLinked: métodos recursivos

- A implementação de métodos recursivos seguem a mesma estratégia usada com a implementação de métodos recursivos na BST.
- Uso de um método auxiliar para implementar o mecanismo de recursão a partir de um nó.
- **Exemplo**: Devolver todas os elementos da árvore numa coleção iterável

```
@Override
public Iterable<E> elements() {
    ArrayList<E> lista = new ArrayList<>();
    if (!isEmpty()) {
        elements(root, lista);
    }
    return lista;
}

private void elements(Position<E> position, ArrayList<E> lista) {

    lista.add(position.element()); // visit (position) primeiro, pre-order
    for (Position<E> w : children(position)) {
        elements(w, lista);
    }

}
```

ADT Tree | Exercícios de implementação

1. Faça *clone* do projeto base **ADTTree_Template** (projeto **IntelliJ**) do *GitHub*:

   https://github.com/pa-estsetubal-ips-pt/ADTTree_Template

2. Forneça o código dos métodos por implementar, i.e., os que estão a lançar `NotImplementedException` ;

   Nota: Em relação ao método size, sugere-se que reveja a implementação realizada para a BST

3. Compile e teste o programa fornecido.

ADT Tree | Exercícios Adicionais

1. Considere o algoritmo Breath-first para percorrer a árvore em largura, por níveis.

```
BFS(arvore)
   Coloque a raiz da árvore na fila
   Enquanto a fila não está vazia faça:
           seja n o primeiro nó da fila
           processe n
           para todo o f nó filho de n
                       coloque f na fila
```

Implemente na classe TreeLinked o método Breath-first que devolve uma Coleção Iteravel com os elementos da árvore ordenados segundo o algoritmo Breath-first.

ADT Tree | Exercícios Adicionais

2. O método `checkPosition` fornecido não está a validar se à posição fornecida pertence à árvore.
   2.1 Implemente um método auxiliar (denominado `belongs` que dado um nó verifica se este pertence à arvore T.
   Sugere-se a utilização do seguinte algoritmo:

```
belongs(tree,node)
    Enquanto  parent(node) <> NULL faça
            node<-parent(node)
    Se node=root(tree)
            retorna verdade
     senão
            retorna falso
```

   2.2 Altere o método checkPosition, para incluir esta validação.

Estudar e Rever

Tree

- Páginas 280-296