

## NaryTree

Generated by Doxygen 1.8.13



# Contents



# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">NaryTree&lt; T &gt;</a>	.....	??
<a href="#">Node&lt; T &gt;</a>	.....	??



## Chapter 2

# Class Documentation

### 2.1 NaryTree< T > Class Template Reference

```
#include <NaryTree.h>
```

#### Public Member Functions

- [NaryTree](#) (unsigned int const &order)
- unsigned int [getOrder](#) () const
- bool [contains](#) (T const &t)
- void [insert](#) (T const &t)
- void [remove](#) (T const &t)

#### Friends

- template<typename U >  
std::ostream & [operator<<](#) (std::ostream &os, const [NaryTree](#)< U > &n)

#### 2.1.1 Detailed Description

```
template<class T>  
class NaryTree< T >
```

Representação de uma árvore de ordem n (árvore enária).

See also

<http://www.inf.ufsc.br/~r.mello/ine5384/13-MetodosPesquisa2.pdf>

#### Template Parameters

<i>T</i>	
----------	--

## 2.1.2 Constructor & Destructor Documentation

### 2.1.2.1 NaryTree()

```
template<class T >
NaryTree< T >::NaryTree (
    unsigned int const & order ) [explicit]
```

Construtor da árvore enária.

#### Parameters

<i>order</i>	ordem da árvore.
--------------	------------------

## 2.1.3 Member Function Documentation

### 2.1.3.1 contains()

```
template<class T >
bool NaryTree< T >::contains (
    T const & t )
```

Verifica se *t* está em algum local da árvore

#### Parameters

<i>t</i>	é a informação a ser buscada.
----------	-------------------------------

#### Returns

true se ela está inserida em algum nó.

### 2.1.3.2 getOrder()

```
template<class T >
unsigned int NaryTree< T >::getOrder ( ) const
```

Retorna a ordem da árvore enária.

#### Returns

a ordem da árvore enária.



## 2.1.3.3 insert()

```
template<class T >
void NaryTree< T >::insert (
    T const & t )
```

Insere na árvore a informação (chave) passada como parâmetro.

## Parameters

<i>t</i>	é a informação a ser inserida.
----------	--------------------------------

## 2.1.3.4 remove()

```
template<class T >
void NaryTree< T >::remove (
    T const & t )
```

Remove na árvore a informação (chave) passada como parâmetro.

## Parameters

<i>t</i>	é a informação a ser removida.
----------	--------------------------------

## 2.1.4 Friends And Related Function Documentation

## 2.1.4.1 operator&lt;&lt;

```
template<class T>
template<typename U >
std::ostream& operator<< (
    std::ostream & os,
    const NaryTree< U > & n ) [friend]
```

overload necessário para poder printar em std::cout

The documentation for this class was generated from the following files:

- NaryTree/NaryTree.h
- NaryTree/NaryTree.inl

## 2.2 Node&lt; T &gt; Class Template Reference

```
#include <Node.h>
```

## Public Member Functions

- [Node](#) (unsigned int const &order)
- [Node](#)< T > \* [next](#) (T const &t)
- unsigned int [indexOfNext](#) (T const &t)
- bool [existsChildAt](#) (unsigned int const &i)
- bool [hasChilds](#) ()
- int [indexOfChild](#) ([Node](#)< T > \*c)
- void [insertKey](#) (T const &t)
- void [removeKey](#) (T const &t)
- void [insertChildAt](#) (int index, [Node](#)< T > \*ch)
- bool [holds](#) (T const &t)
- bool [isLeaf](#) ()
- bool [isFull](#) ()
- bool [isEmpty](#) ()
- int [getOrder](#) ()
- void [setRootState](#) (bool state)
- void [setLeafState](#) (bool state)

### 2.2.1 Detailed Description

```
template<class T>
class Node< T >
```

Representação de um nó de uma árvore enária (árvore de ordem n).

#### Template Parameters

<i>T</i>	é o tipo de informação que o nó guarda.
----------	---

### 2.2.2 Constructor & Destructor Documentation

#### 2.2.2.1 Node()

```
template<class T >
Node< T >::Node (
    unsigned int const & order ) [explicit]
```

Construtor do nó que recebe a ordem da árvore que ele pertence.

#### Parameters

<i>order</i>	ordem da árvore que o nó pertence.
--------------	------------------------------------

### 2.2.3 Member Function Documentation

#### 2.2.3.1 existsChildAt()

```
template<class T >
bool Node< T >::existsChildAt (
    unsigned int const & i )
```

Verifica se existe um filho no índice indicado.

##### Parameters

<i>i</i>	é o índice a ser consultado.
----------	------------------------------

##### Returns

true se existe filho (ou seja, diferente de NULL).

#### 2.2.3.2 getOrder()

```
template<class T >
int Node< T >::getOrder ( )
```

##### Returns

ordem do nó e consequentemente ordem da árvore que ele está.

#### 2.2.3.3 hasChilds()

```
template<class T >
bool Node< T >::hasChilds ( )
```

Verifica se o nó possui filhos (ou seja, no mínimo 1).

##### Returns

true se número de filhos > 0, caso contrário false.

#### 2.2.3.4 holds()

```
template<class T >
bool Node< T >::holds (
    T const & t )
```

Verifica se t (informação) existe no vetor de informações do nó.

**Parameters**

<i>t</i>	é a informação a ser buscada.
----------	-------------------------------

**Returns**

true se ele existe no vetor de informações  
false se ela NÃO existe no vetor de informações

**2.2.3.5 indexOfChild()**

```
template<class T >
int Node< T >::indexOfChild (
    Node< T > * c )
```

**Parameters**

<i>c</i>	é filho a ser utilizado para consultar seu índice no vetor.
----------	---

**Returns**

o índice que o filho passado como parâmetro está no vetor de filhos.  
-1 se o filho não existe

**2.2.3.6 indexOfNext()**

```
template<class T >
unsigned int Node< T >::indexOfNext (
    T const & t )
```

Retorna o índice ideal para a criação do filho

**Parameters**

<i>t</i>	
----------	--

**Returns**

### 2.2.3.7 insertChildAt()

```
template<class T >
void Node< T >::insertChildAt (
    int index,
    Node< T > * ch )
```

Insere o filho no índice passado como parâmetro.

#### Parameters

<i>index</i>	índice a ser utilizado para inserção.
<i>ch</i>	filho utilizado na inserção.

### 2.2.3.8 insertKey()

```
template<class T >
void Node< T >::insertKey (
    T const & t )
```

Insere no vetor de informações (também chamado de chaves) a chave como parâmetro.

#### Parameters

<i>t</i>	é a informação a ser inserida.
----------	--------------------------------

### 2.2.3.9 isEmpty()

```
template<class T >
bool Node< T >::isEmpty ( )
```

Verifica se o nó está vazio.

#### Returns

true se ele está vazio.

### 2.2.3.10 isFull()

```
template<class T >
bool Node< T >::isFull ( )
```

#### Returns

true se o nó está cheio.

### 2.2.3.11 isLeaf()

```
template<class T >
bool Node< T >::isLeaf ( )
```

#### Returns

true se é um nó folha.

### 2.2.3.12 next()

```
template<class T >
Node< T > * Node< T >::next (
    T const & t )
```

Retorna o "sub nó" que a chave t poderia estar

#### Parameters

<i>t</i>	é a chave que deseja ser "buscada" ou localizada possível nó de inserção
----------	--

#### Returns

nó que a chave poderia estar

### 2.2.3.13 removeKey()

```
template<class T >
void Node< T >::removeKey (
    T const & t )
```

Remove no vetor de informações (também chamado de chaves) a chave como parâmetro.

#### Parameters

<i>t</i>	é a informação a ser removida.
----------	--------------------------------

### 2.2.3.14 setLeafState()

```
template<class T >
void Node< T >::setLeafState (
    bool state )
```

Seta se o nó é uma folha.

**Parameters**

<i>state</i>	
--------------	--

**2.2.3.15 setRootState()**

```
template<class T >
void Node< T >::setRootState (
    bool state )
```

Seta se o nó é uma raiz.

**Parameters**

<i>state</i>	
--------------	--

The documentation for this class was generated from the following files:

- NaryTree/Node.h
- NaryTree/Node.inl