DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

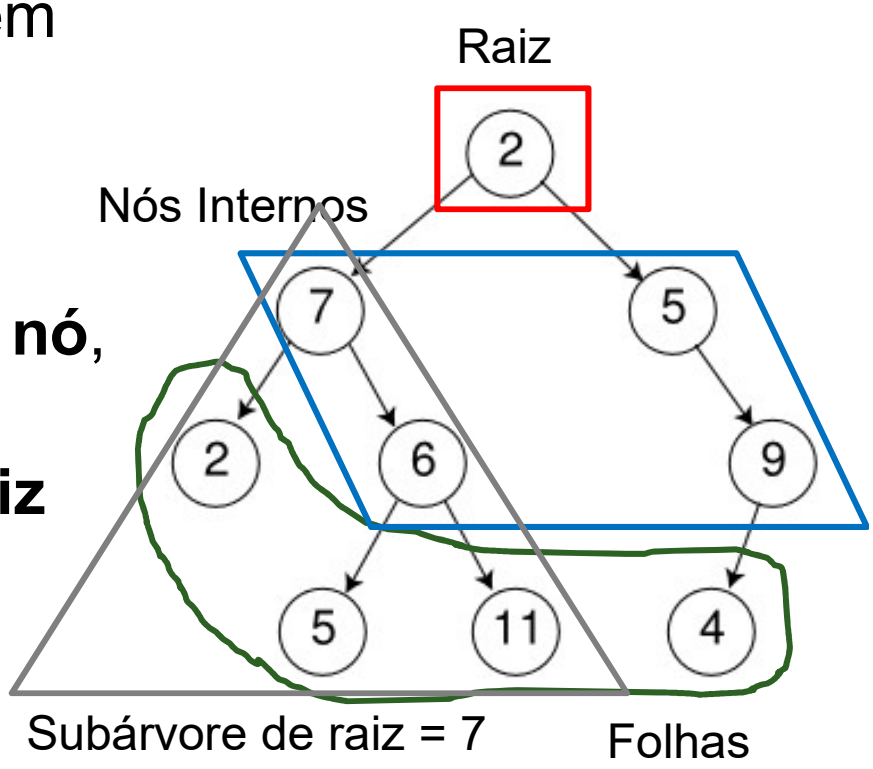
Estruturas de Dados

Árvores

Professores: Luiz Chaimowicz e Raquel Prates

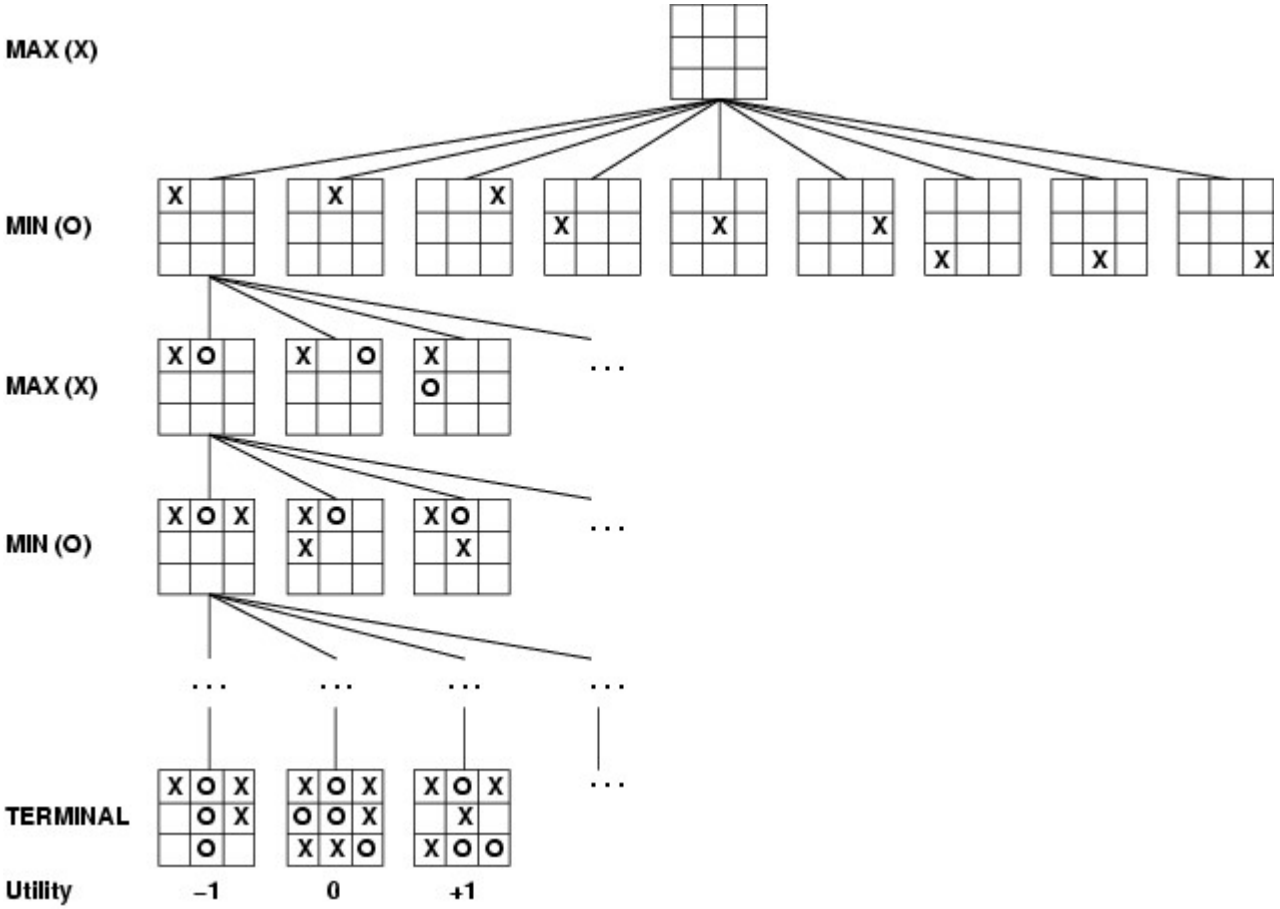
Conceitos básicos

- Arvores organizam os dados em uma estrutura hierárquica
  - *Pais* e *Filhos*
  - *Antecessores* e *Sucessores*
- Cada elemento é chamado de **nó**, e nós são ligados por **arestas**
- O primeiro nó da árvore é a **raiz**
- Os nós **folha** são aqueles que não possuem “filhos”
- Os outros nós são chamados de **nós internos**
- **Recursividade**: o filho de um nó é a raiz de uma outra subárvore



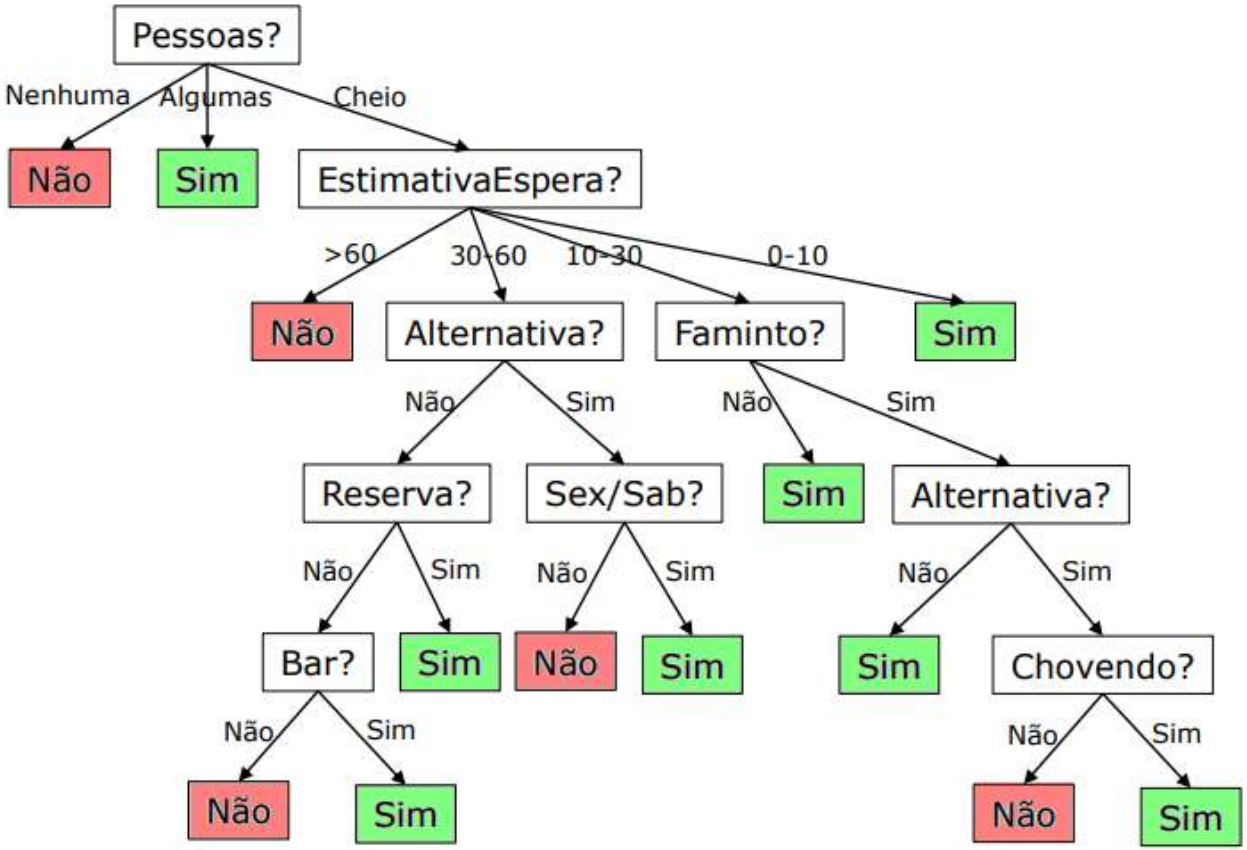
Exemplo

- Jogos: Árvore Minimax



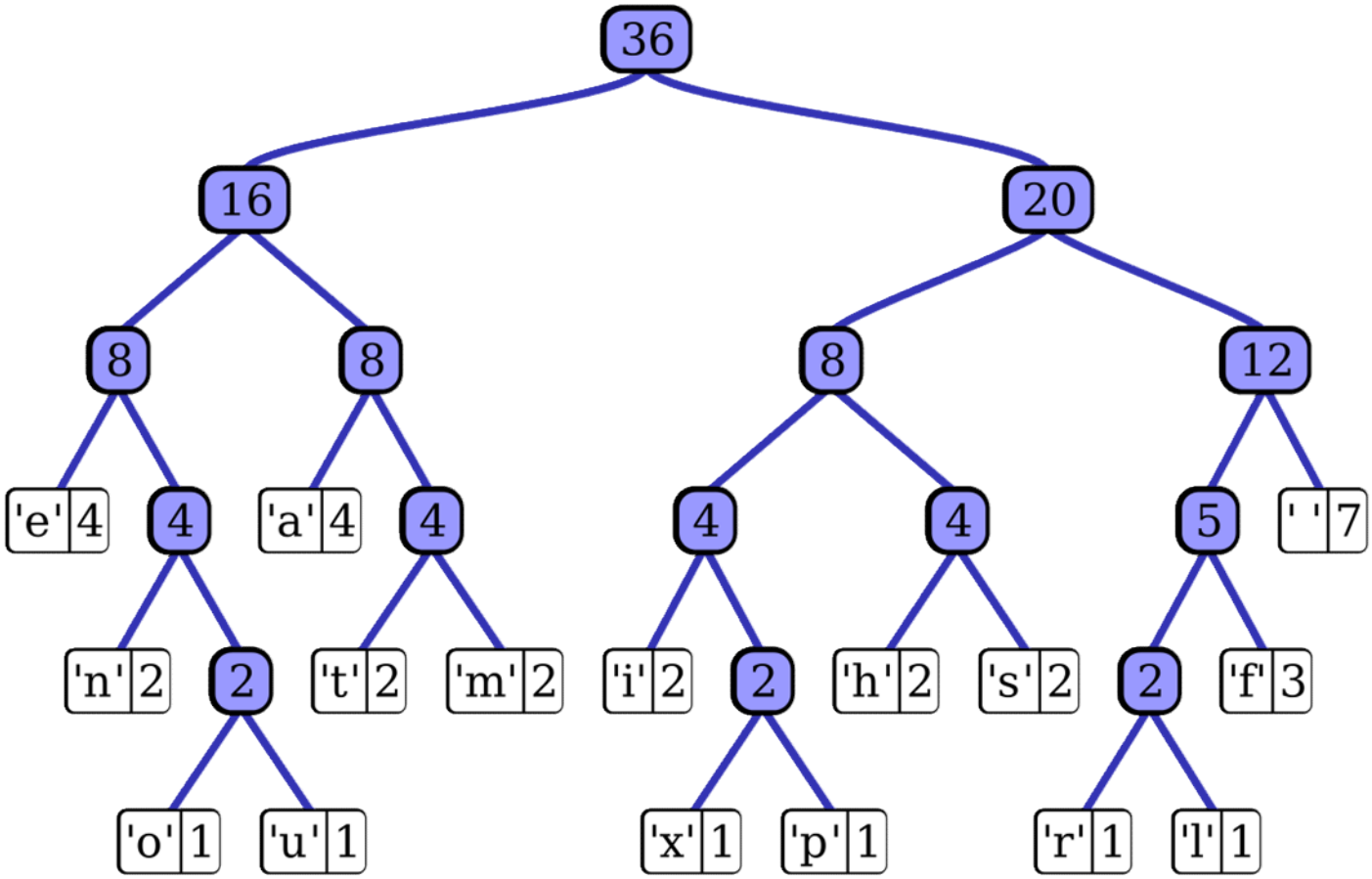
Exemplo

- IA: Árvore de Decisão



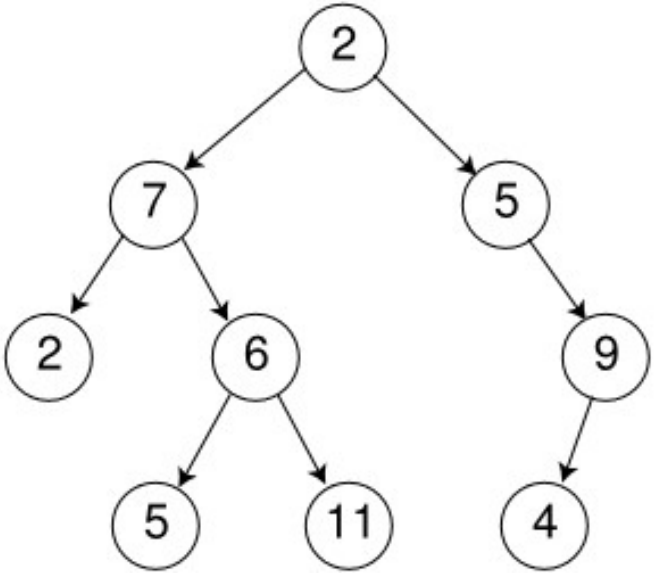
Exemplo

- Compressão de dados: Árvore de Huffman



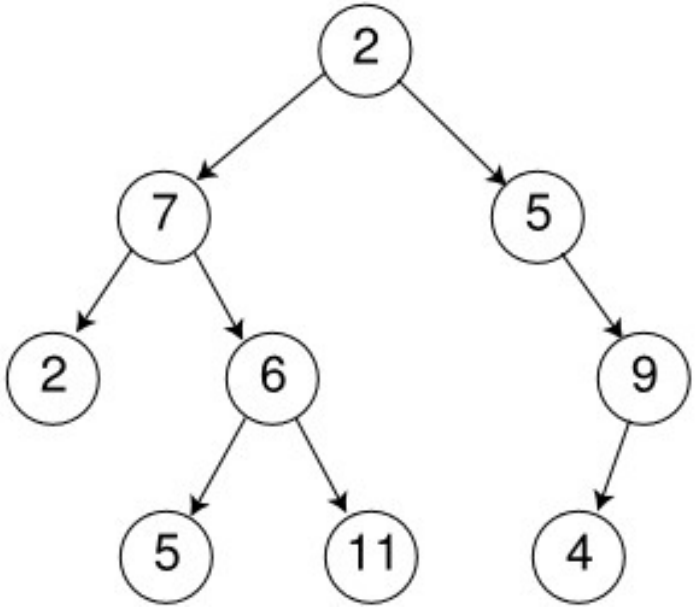
Mais Conceitos

- Níveis
  - A raiz da árvore está no nível 0
  - Se um nó está no nível i, os seus filhos estão no nível $i+1$
- Caminho
  - É a sequência de nós percorrida entre quaisquer 2 nós
  - Em uma árvore, só existe um único caminho entre quaisquer 2 nós
  - Tamanho ou Comprimento do caminho é igual ao número de arestas percorridas (que é igual ao número de nós -1)



Altura / Profundidade

- A **profundidade** de um nó é o comprimento do caminho entre a raiz e aquele nó
- A **altura** de um nó é o comprimento do caminho mais longo desse nó até uma folha
- **Altura da árvore** é igual a altura da raiz que é igual à sua maior profundidade

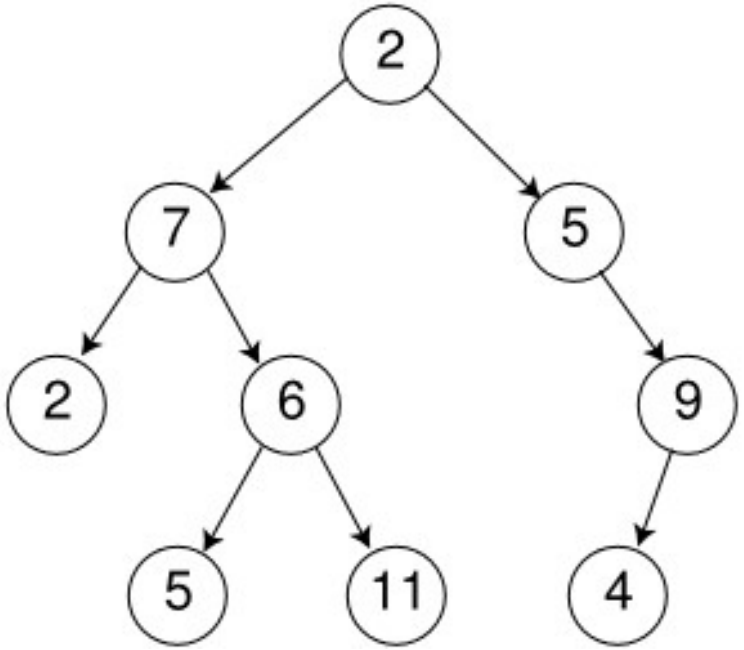


Profundidade do nó 7: **1**
Altura do nó 7: **2**
Altura da Árvore: **3**

Árvores Binárias

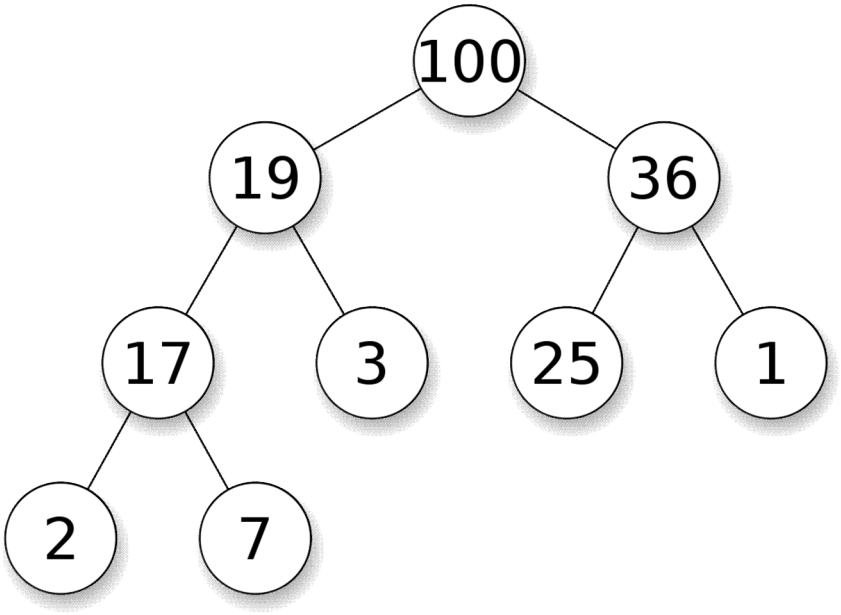
- Em uma árvore binária, cada nó pode ter no máximo 2 filhos (subárvores da esquerda e da direita)

- Algumas aplicações impõem restrições na organização desses nós
  - Heap
  - Árvore Binária de Pesquisa
  - …

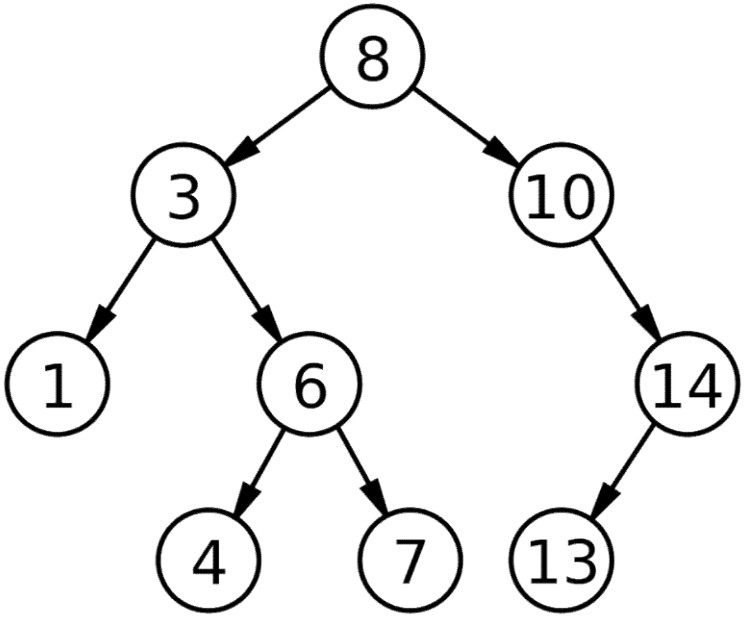


Exemplos: Árvores Binárias

- Heap
  - Pai maior que os filhos

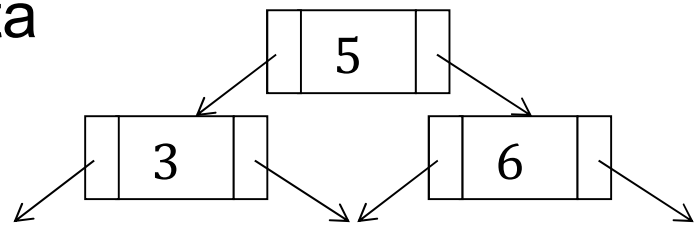


- Árvore Binária de Pesquisa
  - Esq < Raiz < Dir



TAD Árvore Binária

- Implementação usando apontadores
  - Cada nó vai armazenar um item e apontadores para os filhos da esquerda e direita



- Operações Comuns
  - Criar uma árvore
  - Inserir Itens
  - Remover Itens
  - Pesquisar por um item
  - "Percorrer" ou "Caminhar" na árvore
    - Para imprimir todos os itens, por exemplo

Essas operações dependem da organização desejada e serão estudas com detalhes no contexto dos algoritmos de ordenação e pesquisa

Class ArvoreBinaria

```
class ArvoreBinaria
{
    public:
        ArvoreBinaria();
        ~ArvoreBinaria();

        void Insere(TipoItem item);
        void Caminha(int tipo);
        void Limpa();

    private:
        void InsereRecursivo(TipoNo* &p, TipoItem item);
        void ApagaRecursivo(TipoNo* p);
        void PorNivel();
        void PreOrdem(TipoNo* p);
        void InOrdem(TipoNo* p);
        void PosOrdem(TipoNo* p);

        TipoNo *raiz;
};
```

Class TipoNo

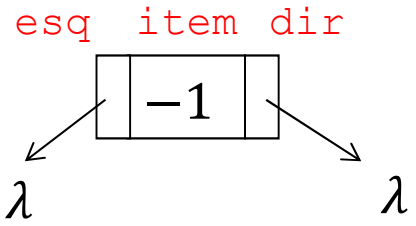
- Classe para representar os Nós da Árvore
  - **TipoItem item**: armazena o item
  - **Esq** e **Dir**: apontadores para as subárvores da direita e esquerda
  - Permite o acesso de atributos privados pela classe ArvoreBinaria
    - *Friend class*

```
class TipoNo
{
    public:
        TipoNo();

    private:
        TipoItem item;
        TipoNo *esq;
        TipoNo *dir;

    friend class ArvoreBinaria;
};
```

```
TipoNo::TipoNo()
{
    item.SetChave(-1);
    esq = NULL;
    dir = NULL;
}
```



Class ÁrvoreBinária
Construtor e Destrutor

- Construtor
  - Inicializa o apontador raiz com null.

```
ArvoreBinaria::ArvoreBinaria()
{
    raiz = NULL;
}
```

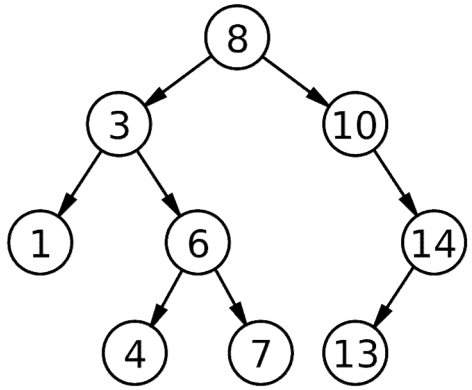
raiz $\longrightarrow \lambda$

- Destrutor
  - Chama o método Limpa que remove todos os nós da árvore

```
ArvoreBinaria::~ArvoreBinaria()
{
    Limpa();
}
```

Inserção

- O método de inserção da árvore depende da organização desejada para os itens
- Aqui apresentamos o método de inserção em uma **árvore binária de pesquisa**
  - Esq < Raiz < Dir

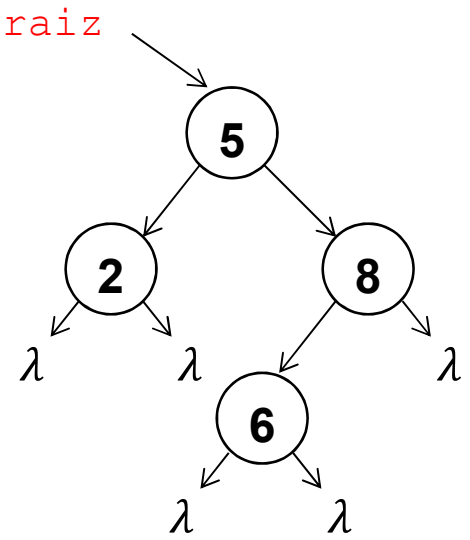


- Método Recursivo para Inserção
  - Compara item com o elemento da Raiz
    - Se menor: insere na subárvore da esquerda
    - Se maior: insere na subárvore da direita
    - Quando a raiz for nula, insere item.
  - Raiz deve ser passada por referência

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

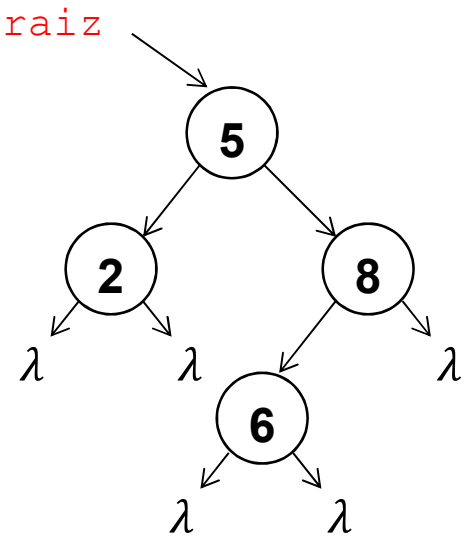



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

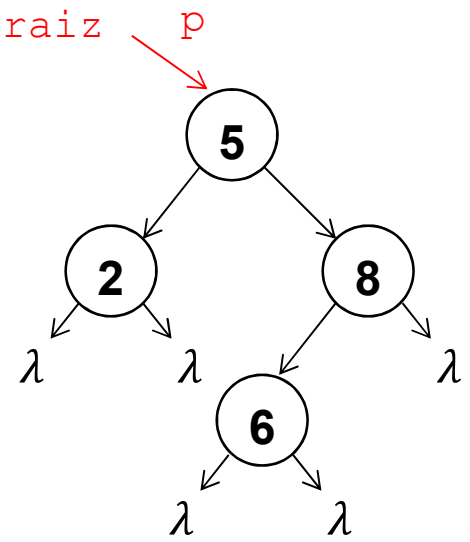



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

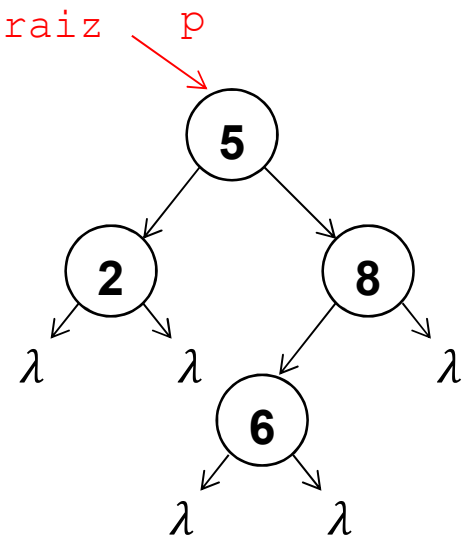



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

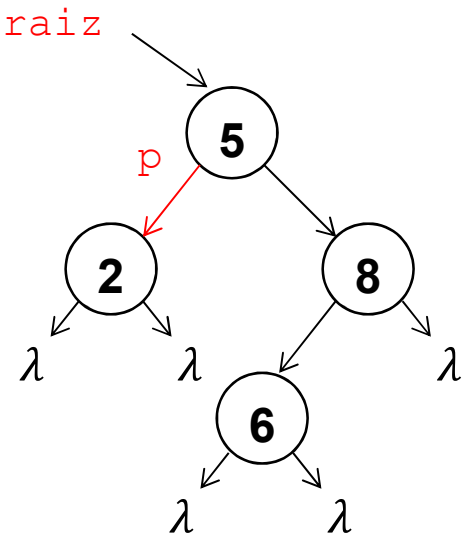



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

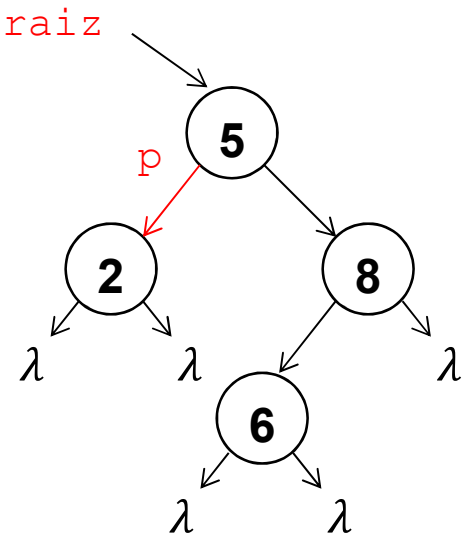



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

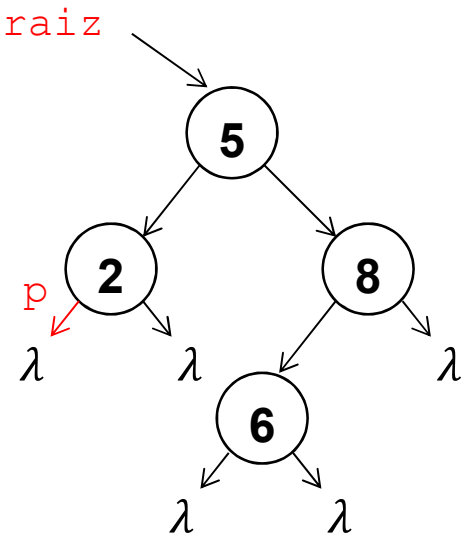



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

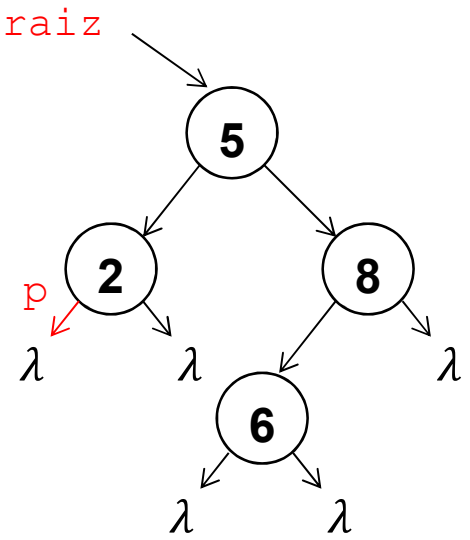



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

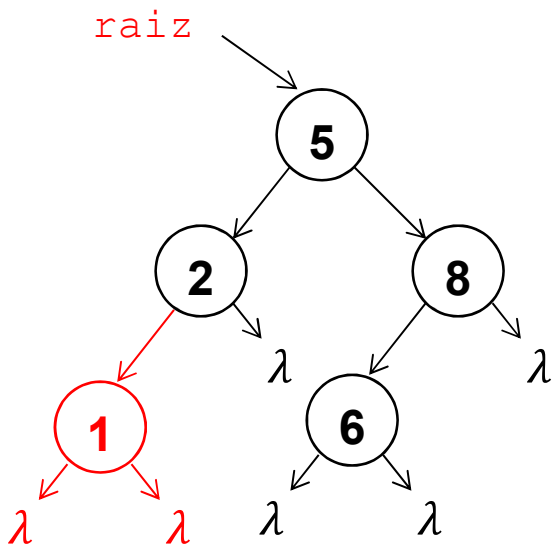



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

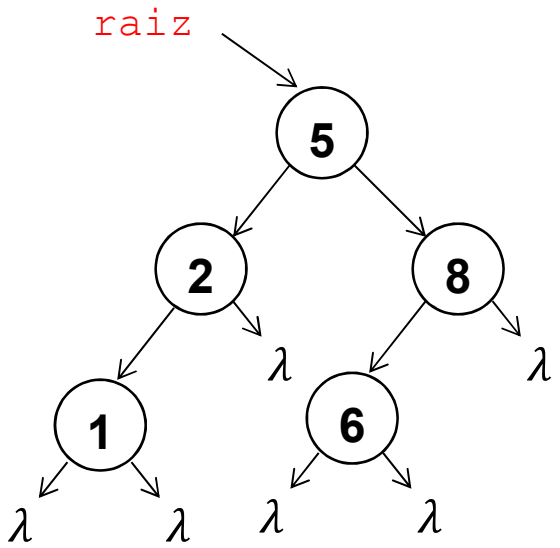



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

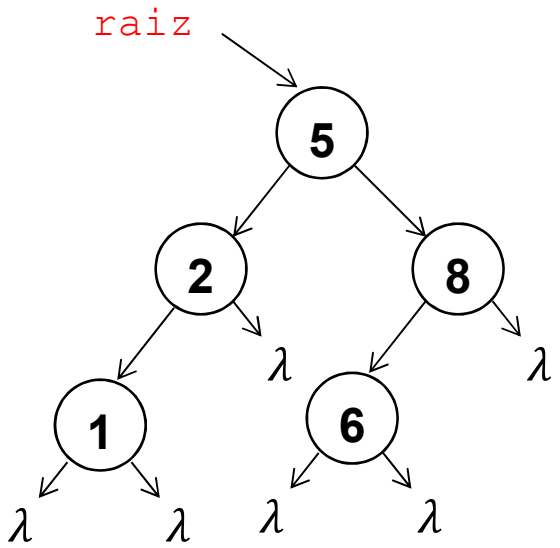



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

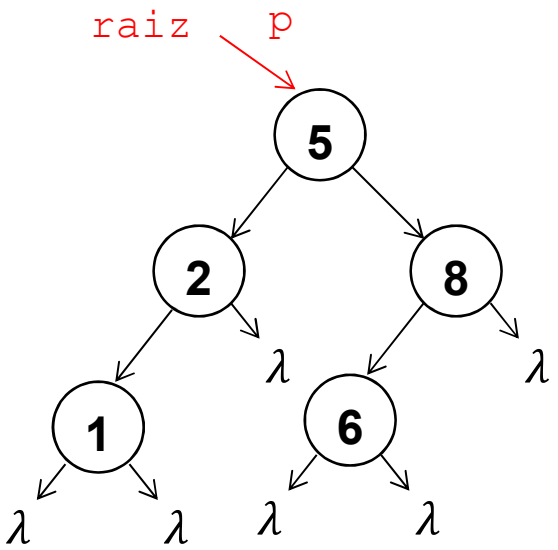



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

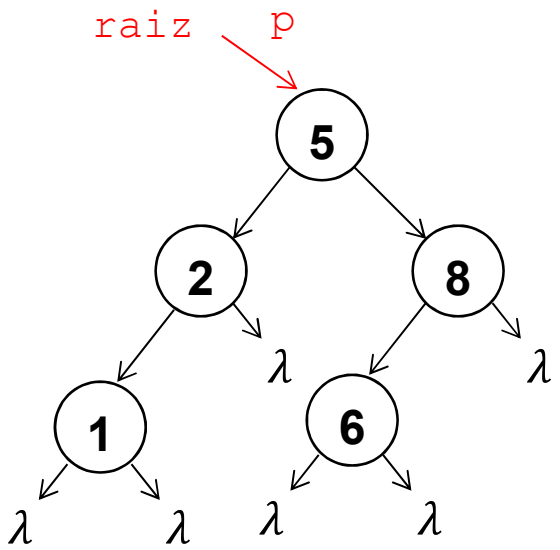



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

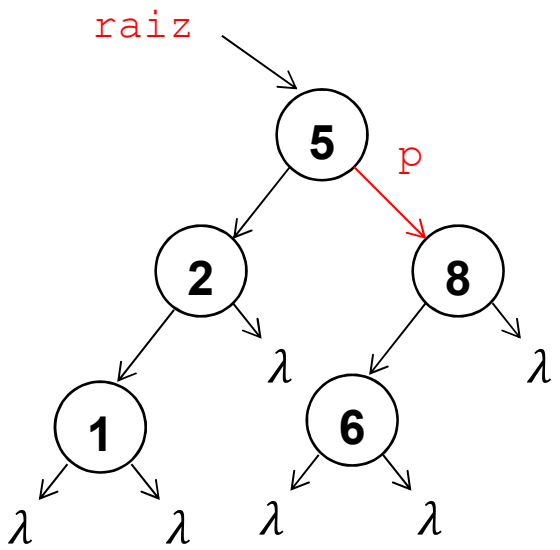



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

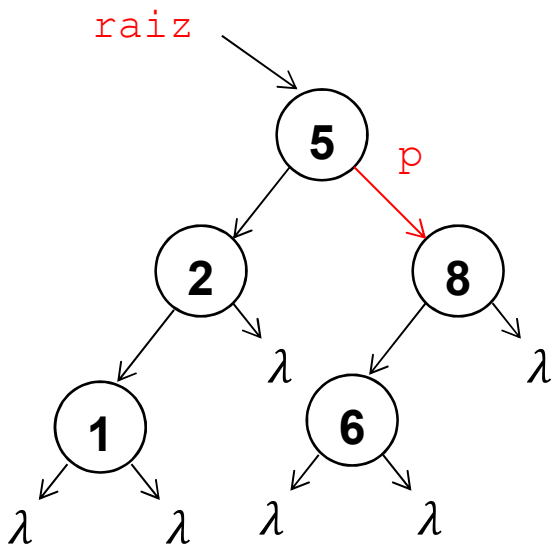



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

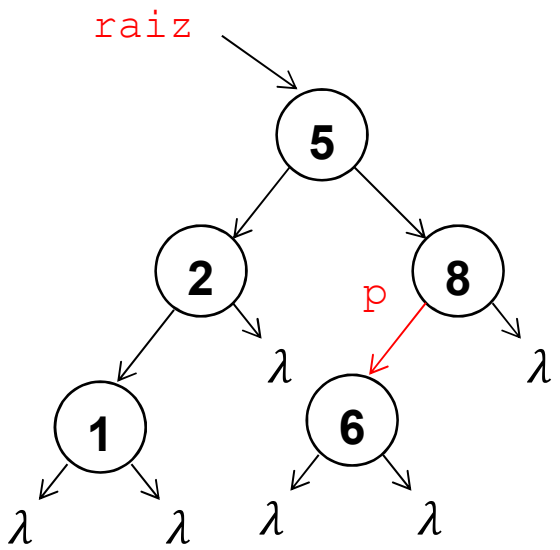



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

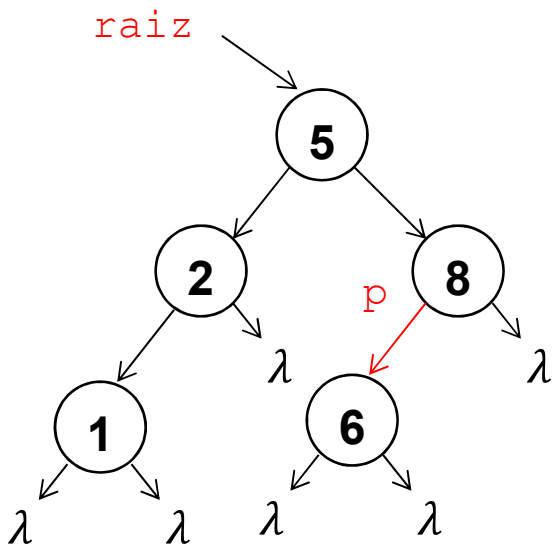



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

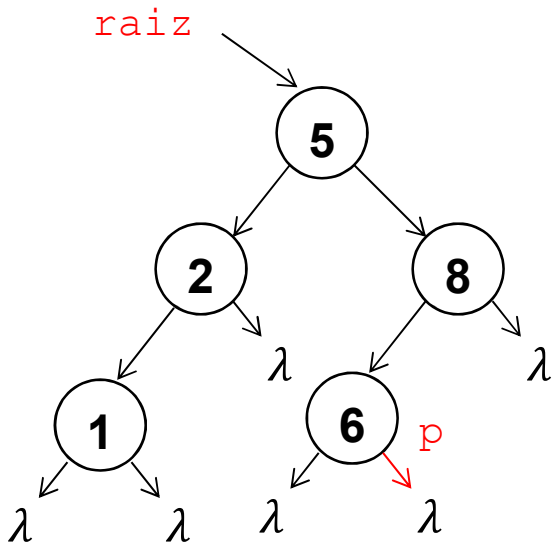



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

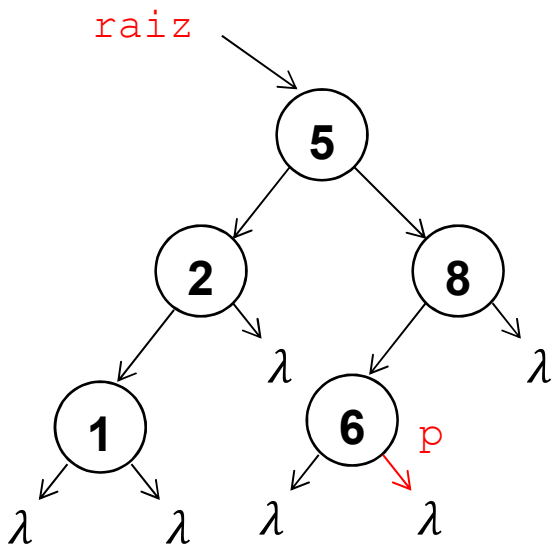



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```



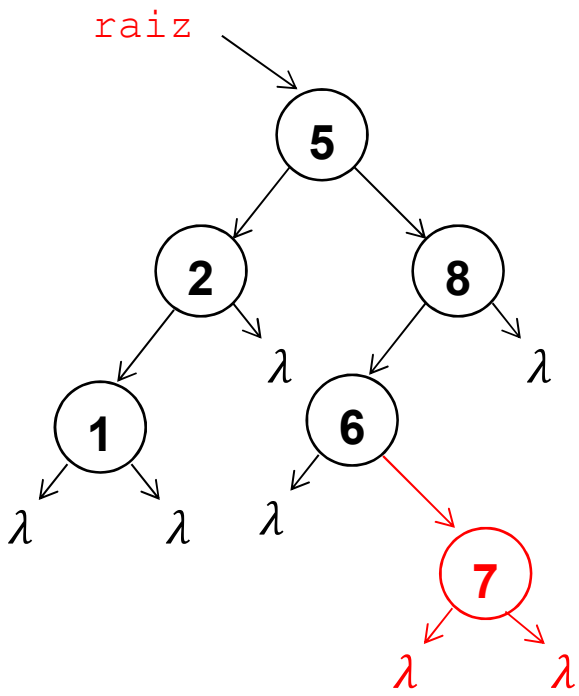

```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

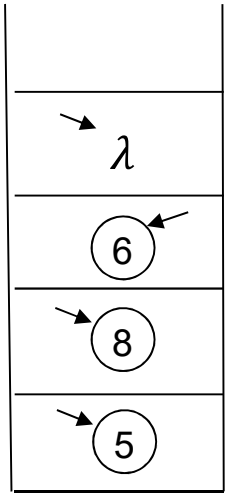


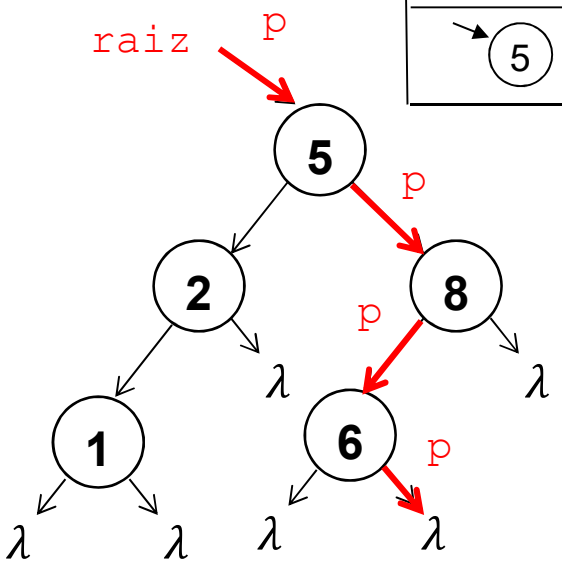
Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

Pilha de Execução





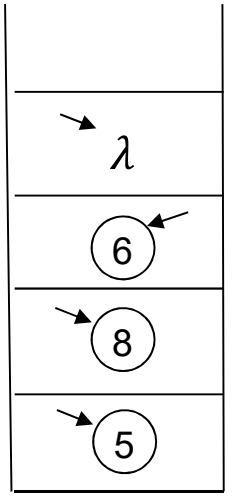

```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

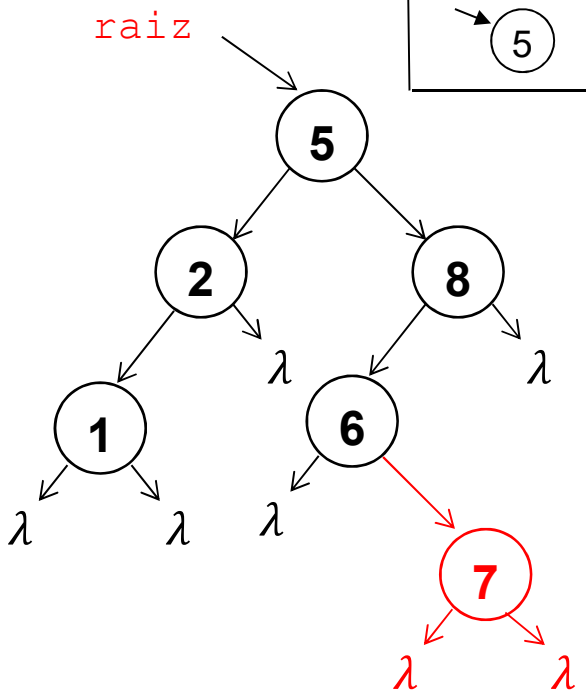
Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

Pilha de Execução



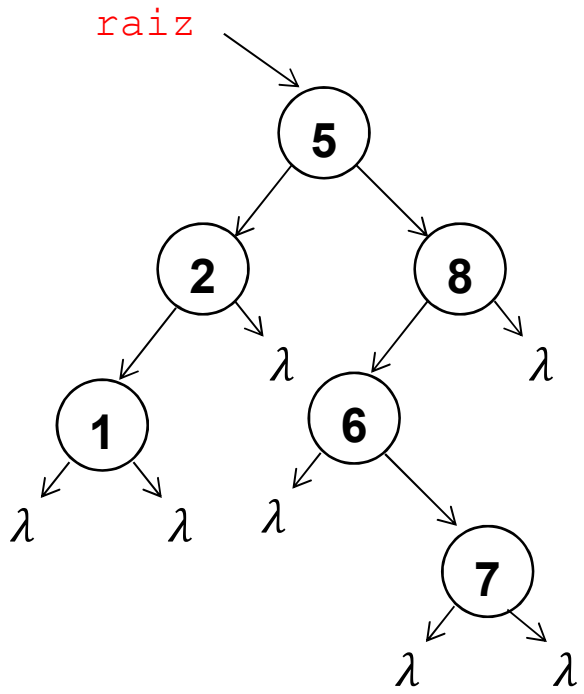



```
ArvoreBinaria T;
TipoItem x;
...
x.SetChave(1);
T.Insere(x);
x.SetChave(7);
T.Insere(x);
```

Inserção

```
void ArvoreBinaria::Insere(TipoItem item){
    InsereRecursivo(raiz,item);
}

void ArvoreBinaria::InsereRecursivo(TipoNo* &p, TipoItem item){
    if(p==NULL){
        p = new TipoNo();
        p->item = item;
    }
    else{
        if(item.GetChave() < p->item.GetChave())
            InsereRecursivo(p->esq, item);
        else
            InsereRecursivo(p->dir, item);
    }
}
```

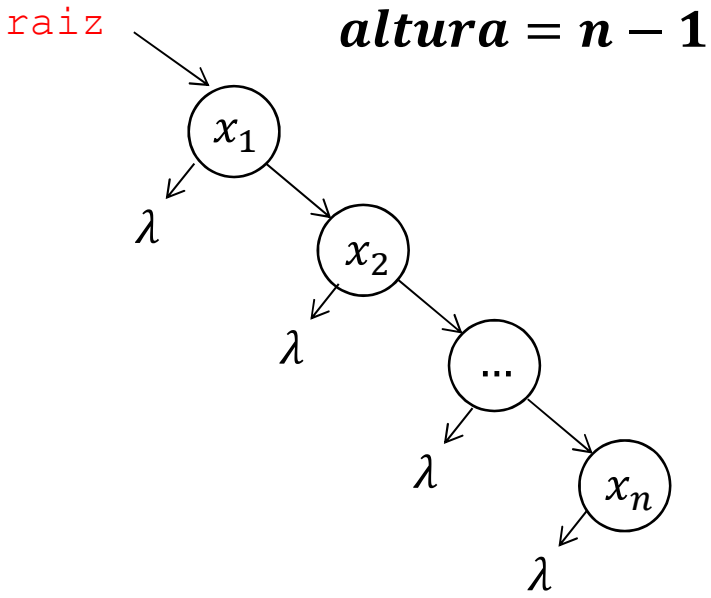


Qual a ordem de Complexidade do método Insere?

Inserção

- A ordem complexidade do Inserção é relacionada com a altura da árvore…
- Qual é a árvore binária de **maior** altura com *n* elementos?

- Árvore binária "degenerada"
  - cada nó tem apenas um filho
  - gerada, por exemplo, com a inserção dos elementos em ordem crescente

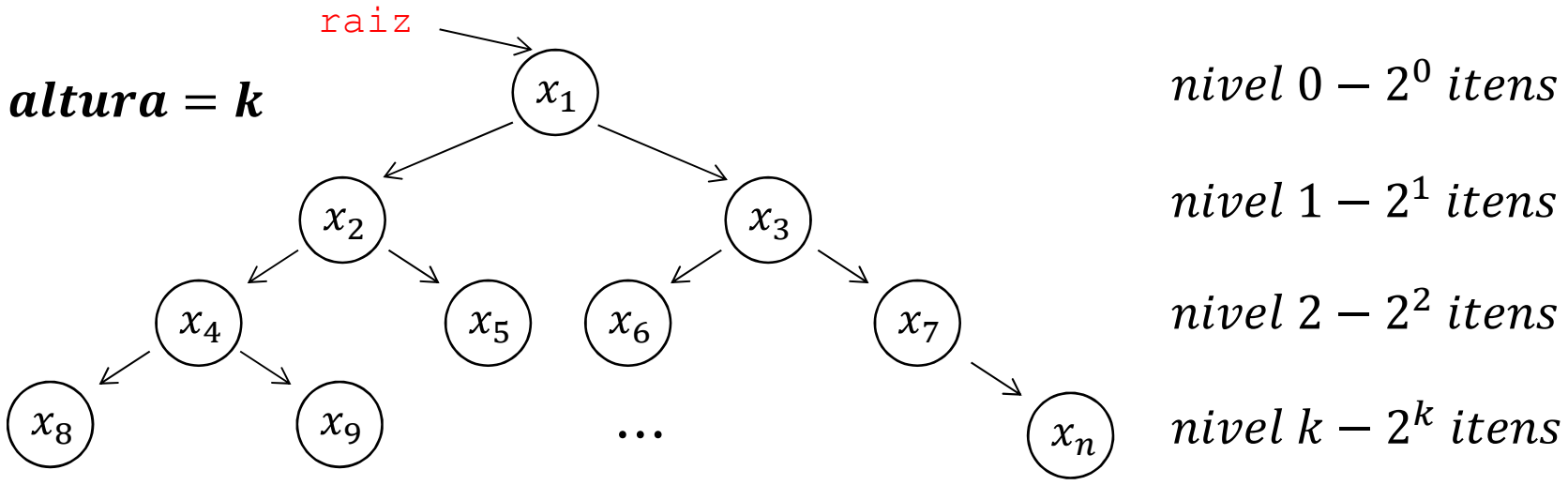


- Pior caso do método Insere:

$$O(n)$$

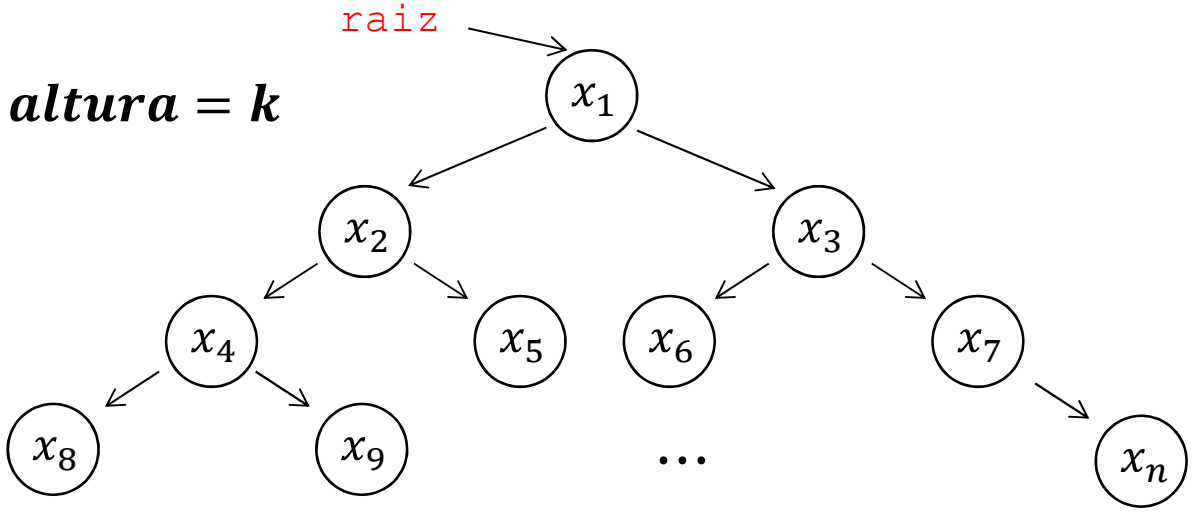
Inserção

- Qual é a árvore binária de **menor** altura com *n* elementos?
- Árvore binária "balanceada": nós folha em no máximo 2 níveis em sequência



Inserção

- Qual é a árvore binária de **menor** altura com *n* elementos?
- Árvore binária "balanceada": todos os nós possuem 2 filhos



$$\boldsymbol{altura = k}$$

$$n = 2^0 + 2^1 + \cdots + 2^k$$

$$n = \sum_{i=0}^{k} 2^i = \frac{2^{k+1} - 1}{2 - 1}$$

$$n = 2^{k+1} - 1$$

$$k + 1 = \log(n - 1)$$

$$k = \log(n + 1) - 1$$

- Melhor caso do método Insere:

$$O(\log n)$$

Caminhamento em Árvores

- Em alguns casos, pode ser necessário caminhar na árvores “visitando” (por exemplo imprimindo) todos os seus nodos
- Diversas formas de percorrer ou caminhar em uma árvore listando seus nós
- Tipos mais comuns:
  - Caminhamento em Profundidade
    - Pré-ordem (Pré-fixada)
    - In-ordem (Central ou Infixada)
    - Pós-ordem (Pós-fixada)
  - Caminhamento por nível

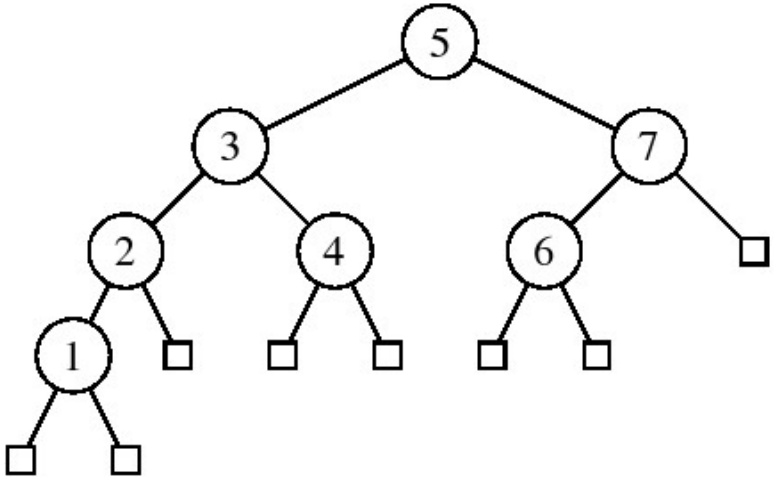
Caminhamento em Árvores

- Pré-ordem (Pré-fixada)
  - Visita o nó e depois os filhos da esquerda e da direita
- In-ordem (Central ou Infixada)
  - Visita o filho da esquerda, o nó, e depois o filho da direita
- Pós-ordem (Pós-fixada)
  - Visita os filhos da esquerda e da direita e depois o nó
- Caminhamento por nível
  - Visita os nodos de cada nível em sequência

Pré-Ordem

- Imprime o item, e depois visita recursivamente as árvores da esquerda e da direita

```
void ArvoreBinaria::PreOrdem(TipoNo *p){
     if(p!=NULL){
          p->item.Imprime();
          PreOrdem(p->esq);
          PreOrdem(p->dir);
     }
}
```



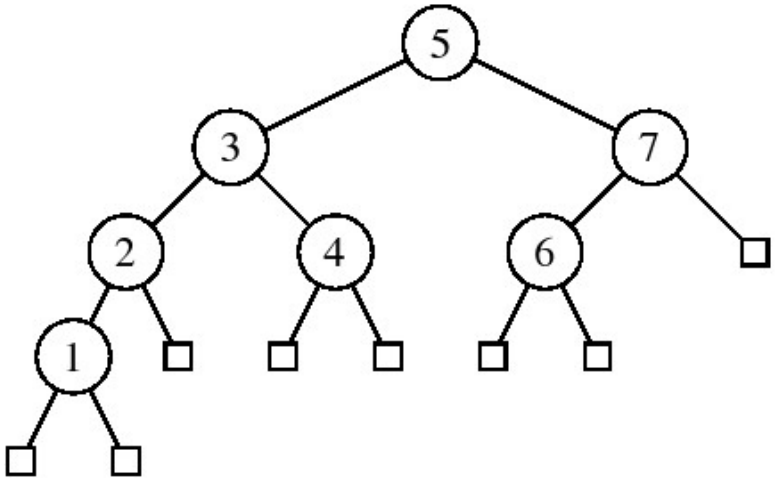
5 3 2 1 4 7 6

In-Ordem (ou central)

- Visita recursivamente a árvore da esquerda, imprime o item, e depois visita a subárvore da direita

```
void ArvoreBinaria::InOrdem(TipoNo *p){
    if(p!=NULL){
        InOrdem(p->esq);
        p->item.Imprime();
        InOrdem(p->dir);
    }
}
```

1 2 3 4 5 6 7

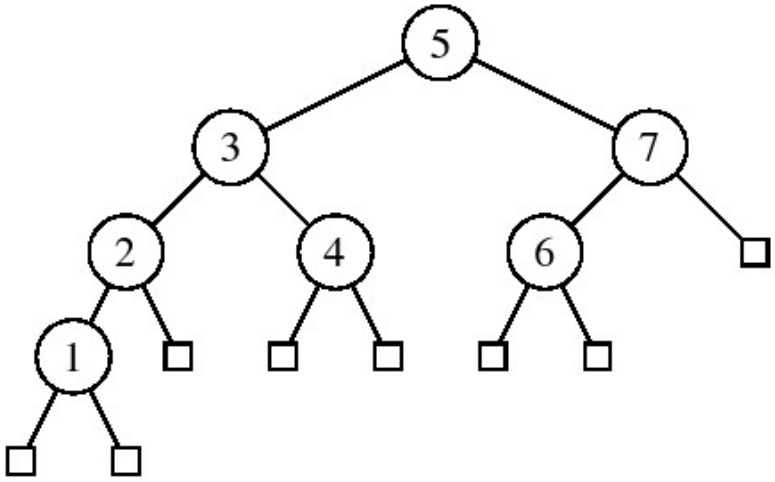


Pos-Ordem

- visita recursivamente as árvores da esquerda e da direita e depois imprime o item

```
void ArvoreBinaria::PosOrdem(TipoNo *p){
    if(p!=NULL){
        PosOrdem(p->esq);
        PosOrdem(p->dir);
        p->item.Imprime();
    }
}
```

1 2 4 3 6 7 5

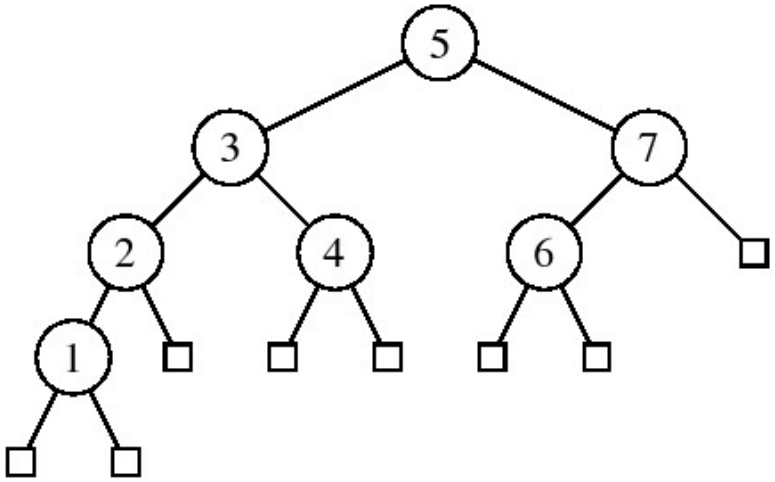


Caminhamento por Nível

- Visita os nós por nível usando uma fila auxiliar

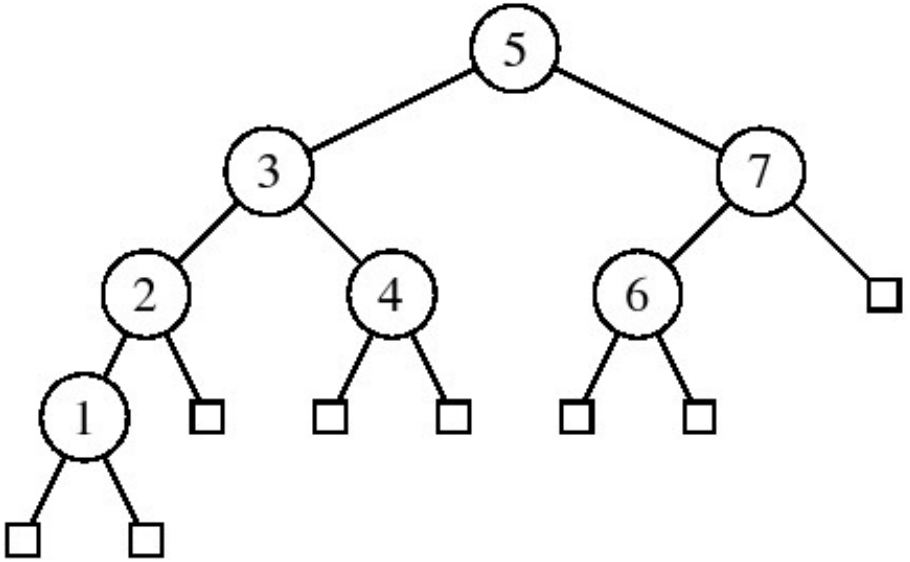
```
void ArvoreBinaria::CaminhaNivel(){
    FilaArranjo F;  // fila de apontadores para nos
    TipoNo *p;

    F.Enfileira(raiz);
    while(!F.Vazia()) {
        p = F.Desenfileira();
        if(p!=NULL) {
            p->item.Imprime();
            F.Enfileira(p->esq);
            F.Enfileira(p->dir);
        }
    }
}
```



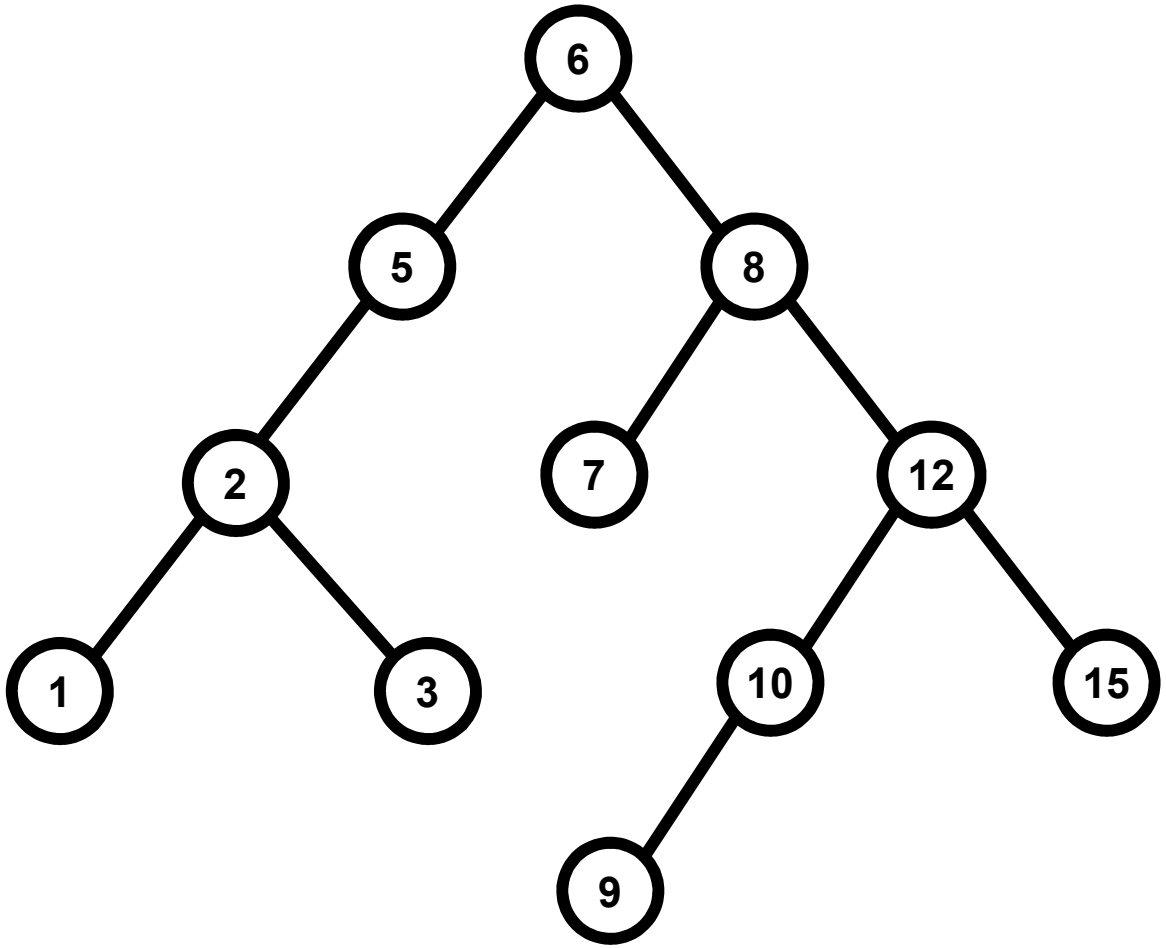
5 3 7 2 4 6 1

Exemplo de Caminhamento



- Pré-Ordem: 5, 3, 2, 1, 4, 7, 6
- Central: 1, 2, 3, 4, 5, 6 , 7
- Pós-Ordem: 1, 2, 4, 3, 6, 7, 5
- Por nível: 5, 3, 7, 2, 4, 6, 1

Exemplo 2 de Caminhamento



- Pré-ordem:

6 5 2 1 3 8 7 12 10 9 15

- Central:

1 2 3 5 6 7 8 9 10 12 15

- Pós-ordem:

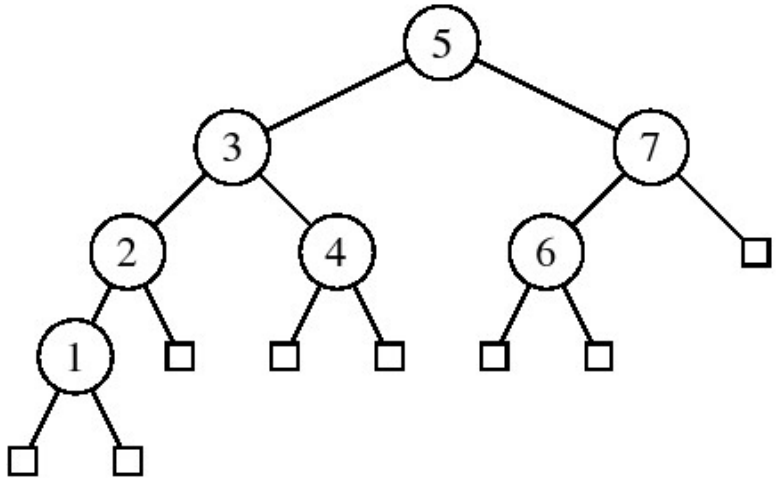
1 3 2 5 7 9 10 15 12 8 6

- Por nível

6 5 8 2 7 12 1 3 10 15 9

ArvoreBinária: Limpa

- Faz o caminhamento PosOrdem removendo os nodos




```
void ArvoreBinaria::Limpa(){
    ApagaRecursivo(raiz);
    raiz = NULL;
}
```

```
void ArvoreBinaria::ApagaRecursivo(TipoNo *p){
    if(p!=NULL){
        ApagaRecursivo(p->esq);
        ApagaRecursivo(p->dir);
        delete p;
    }
}
```