SIN5013
Análise de Algoritmos e Estruturas de Dados

Árvores

Prof. Luciano Antonio Digiampietri

Árvores

Estrutura de dados composta de um nó chamado raiz, abaixo do qual estão as subárvores que compõem essa árvore.

Cada nó possui um nó pai, exceto o nó raiz que não possui pai.

Árvores

O número de subárvores de cada nó (analogamente, o número de filhos de um nó) é chamado de grau desse nó.

O grau máximo permitido para uma árvore identifica o tipo de árvore (binária, ternária, etc).

Nós que não possuem filhos são chamados de folhas.

Nós abaixo de um determinado nó são seus descendentes.

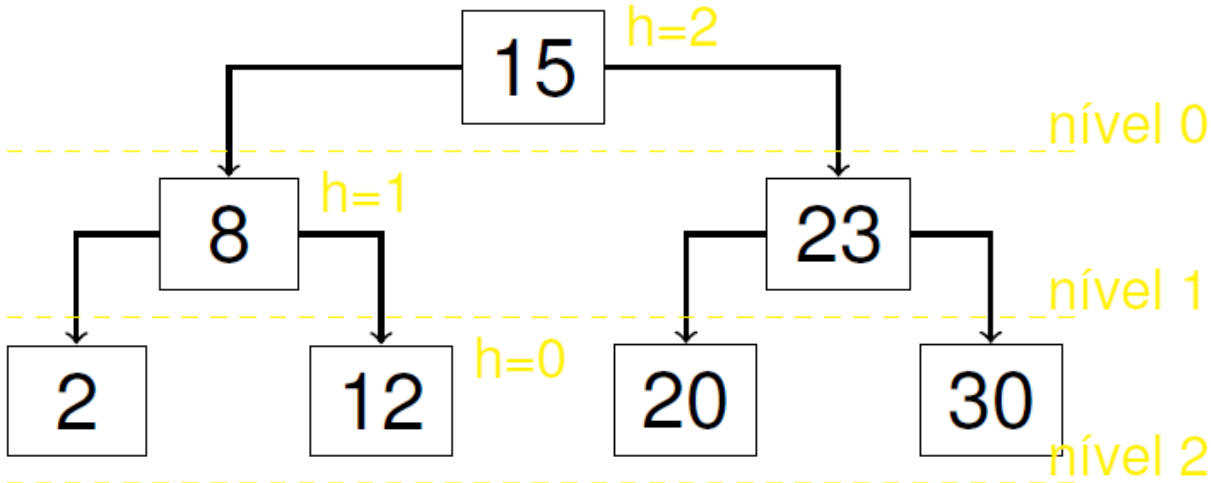
Árvores

O nível do nó raiz é 0.

A altura *h* de um nó é o comprimento do caminho mais longo entre ele e uma folha.

A altura de uma árvore é a altura do nó raiz.

A altura de um nó pode ser calculada como sendo 1 mais o máximo entre a altura de seus filhos.



Árvores Binárias

Uma árvore binária é uma árvore em que, abaixo de cada nó, existem no máximo duas subárvores (cada nó terá, no máximo, dois filhos).

Árvores Binárias - Modelagem

```
#include <stdio.h>
#include <stdlib.h>
#define true 1
#define false 0
typedef int bool;

typedef int TIPOCHAVE;

typedef struct aux {
  TIPOCHAVE chave;
  /* Dados armazenados vão aqui */
  struct aux *esq, *dir;
} NO;

typedef NO* PONT;
```

Árvores Binárias - Altura

```
int altura(PONT p){
  if (!p) return -1;
  else return 1 + max(altura(p->esq),altura(p->dir));
}

int max(int a, int b){
  if (a>b) return a;
  return b;
}
```

Árvores Binárias - Número de Nós

```
int numeroNos(PONT raiz){
  if (!raiz ) return 0;
  return 1 + numeroNos(raiz->esq)
                    + numeroNos(raiz->dir);
}
```

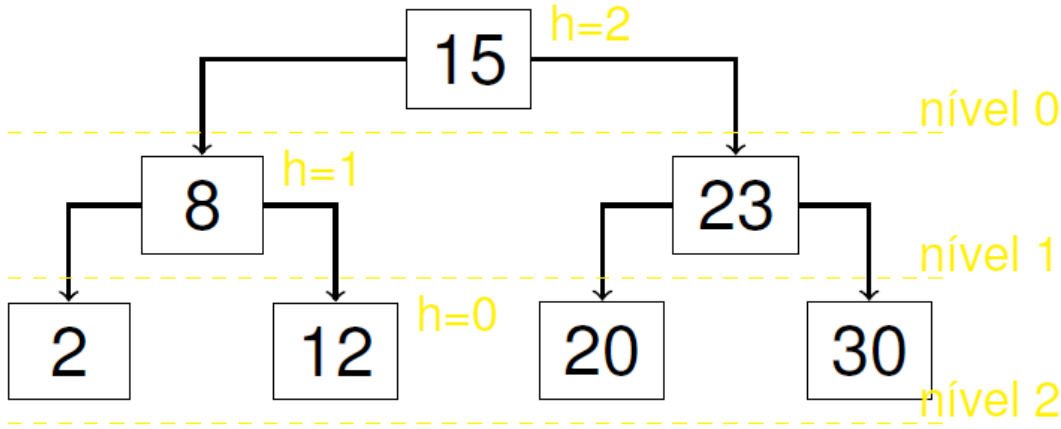
Árvores Binárias - Percursos

Pré-ordem

Em ordem

Pós-ordem

Em nível / largura

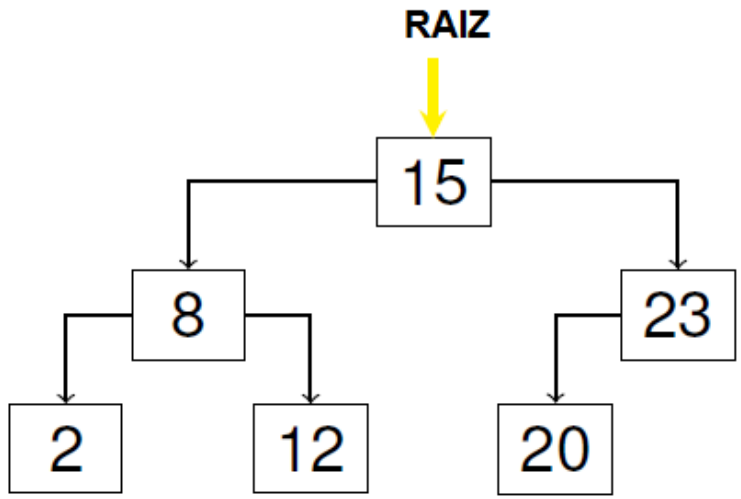


Árvores Binárias - Pré-ordem

```
void exibirArvorePreOrdem(PontNo raiz){
  if (raiz == NULL) return;
  printf("%i ",raiz->chave);
  exibirArvorePreOrdem(raiz->esq);
  exibirArvorePreOrdem(raiz->dir);
}
```

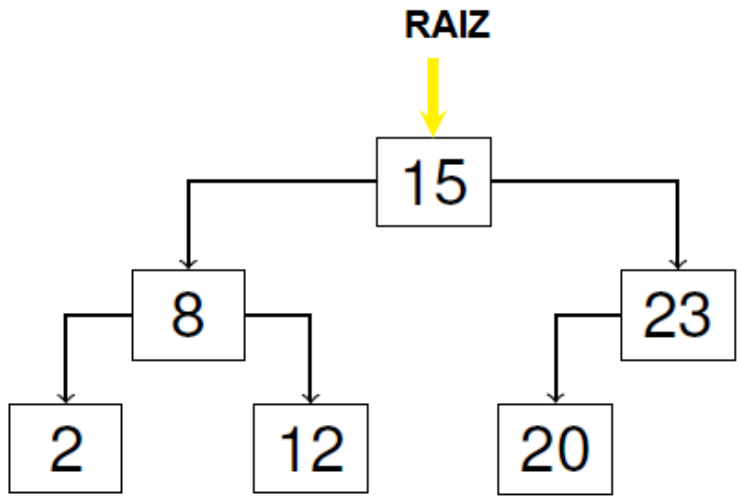
Árvores Binárias - Pré-ordem

```
void exibirArvorePreOrdem(PontNo raiz){
  if (raiz == NULL) return;
  printf("%i ",raiz->chave);
  exibirArvorePreOrdem(raiz->esq);
  exibirArvorePreOrdem(raiz->dir);
}
```



Árvores Binárias - Pré-ordem

```
void exibirArvorePreOrdem(PontNo raiz){
  if (raiz == NULL) return;
  printf("%i ",raiz->chave);
  exibirArvorePreOrdem(raiz->esq);
  exibirArvorePreOrdem(raiz->dir);
}
```



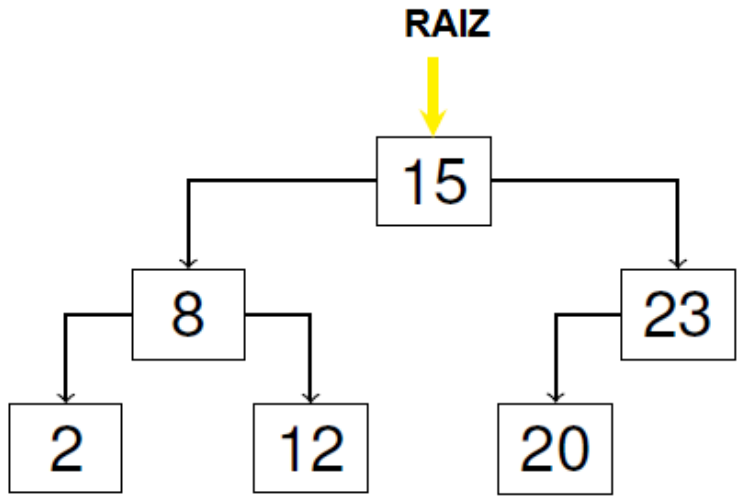
Resultado da execução: 15 8 2 12 23 20

Árvores Binárias - Em ordem

```
void exibirArvoreEmOrdem(PontNo raiz){
  if (raiz == NULL) return;
  exibirArvoreEmOrdem(raiz->esq);
  printf("%i ",raiz->chave);
  exibirArvoreEmOrdem(raiz->dir);
}
```

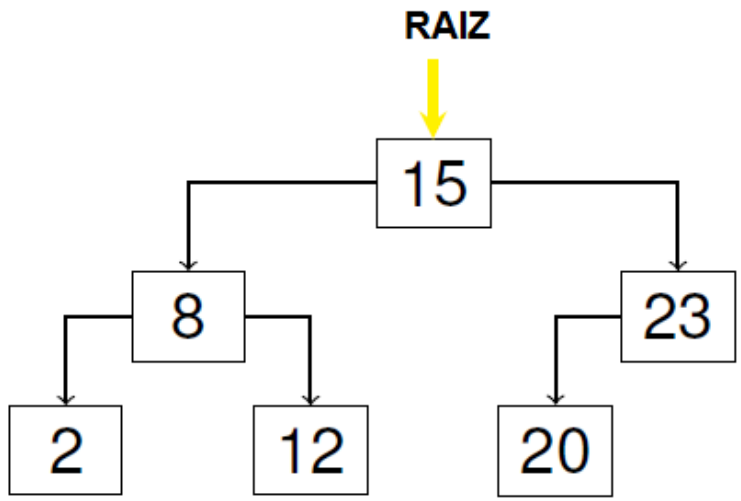
Árvores Binárias - Em ordem

```
void exibirArvoreEmOrdem(PontNo raiz){
  if (raiz == NULL) return;
  exibirArvoreEmOrdem(raiz->esq);
  printf("%i ",raiz->chave);
  exibirArvoreEmOrdem(raiz->dir);
}
```



Árvores Binárias - Em ordem

```
void exibirArvoreEmOrdem(PontNo raiz){
  if (raiz == NULL) return;
  exibirArvoreEmOrdem(raiz->esq);
  printf("%i ",raiz->chave);
  exibirArvoreEmOrdem(raiz->dir);
}
```



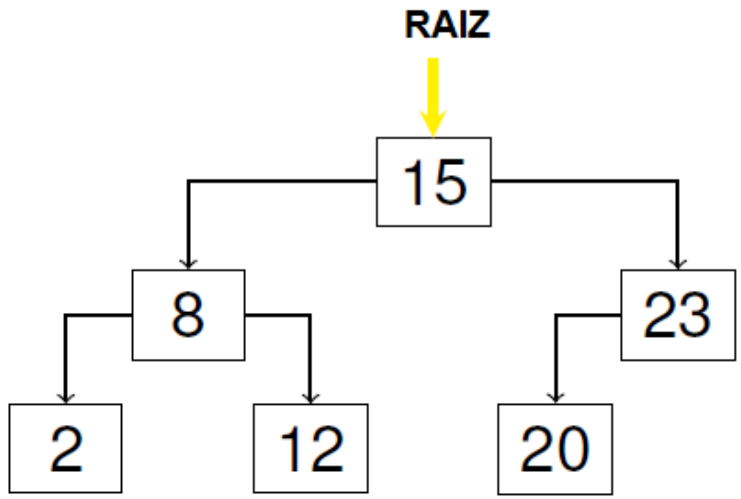
Resultado da execução: 2 8 12 15 20 23. Resultado ordenado **apenas** porque esta árvore é uma árvore **de busca** binária.

Árvores Binárias - Pós-ordem

```
void exibirArvorePosOrdem(PontNo raiz){
  if (raiz == NULL) return;
  exibirArvorePosOrdem(raiz->esq);
  exibirArvorePosOrdem(raiz->dir);
  printf("%i ",raiz->chave);
}
```

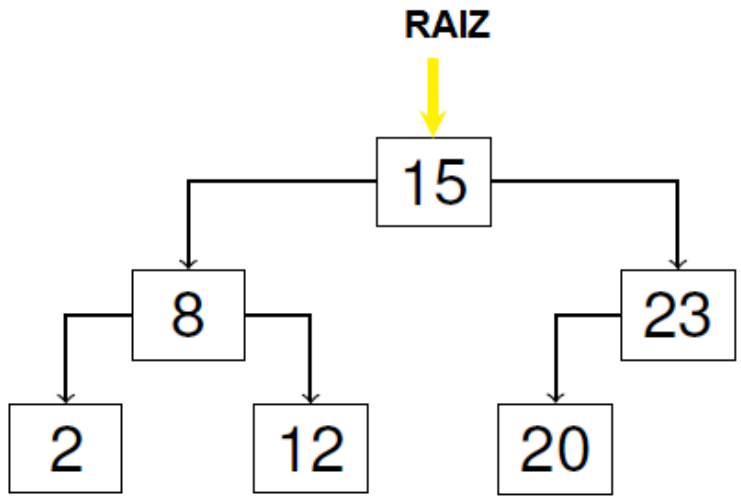
Árvores Binárias - Pós-ordem

```
void exibirArvorePosOrdem(PontNo raiz){
  if (raiz == NULL) return;
  exibirArvorePosOrdem(raiz->esq);
  exibirArvorePosOrdem(raiz->dir);
  printf("%i ",raiz->chave);
}
```



Árvores Binárias - Pós-ordem

```
void exibirArvorePosOrdem(PontNo raiz){
  if (raiz == NULL) return;
  exibirArvorePosOrdem(raiz->esq);
  exibirArvorePosOrdem(raiz->dir);
  printf("%i ",raiz->chave);
}
```



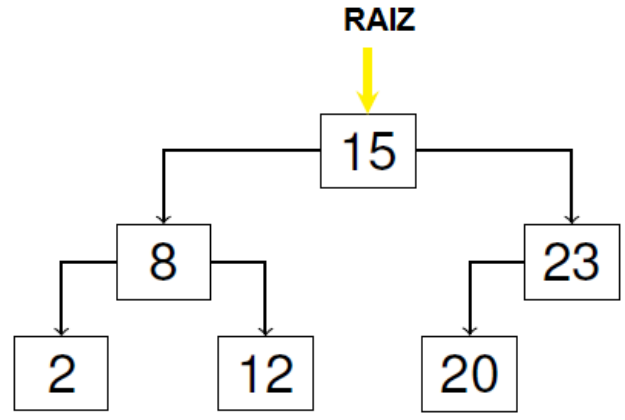
Resultado da execução: 2 12 8 20 23 15

Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```

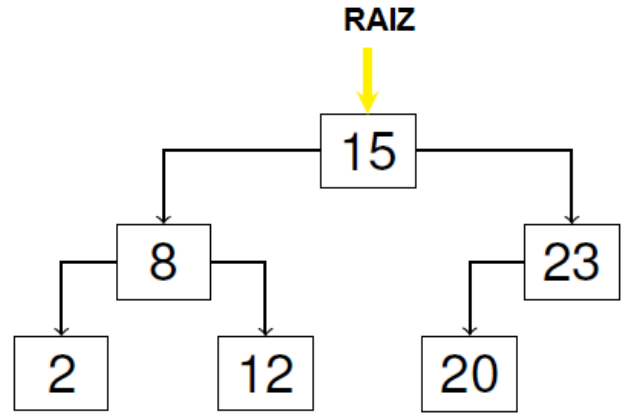
Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```



Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```

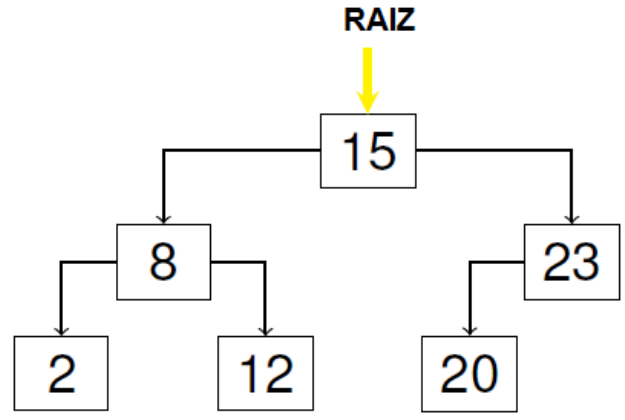


Elementos na Pilha - topo:

Valores impressos:

Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```

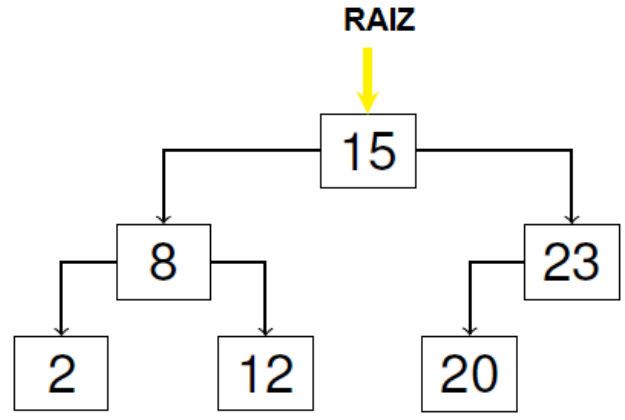


Elementos na Pilha - topo: 15

Valores impressos:

Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```

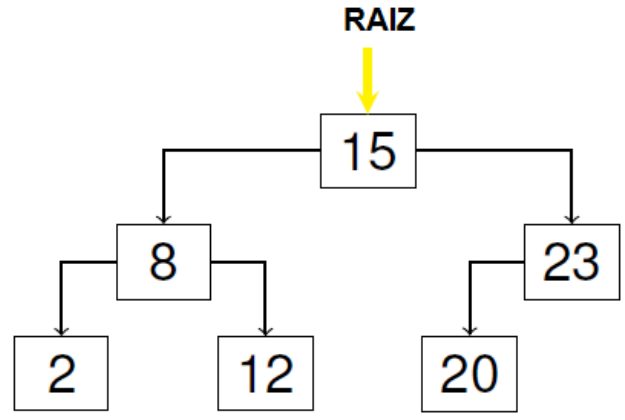


Elementos na Pilha - topo: 8 23

Valores impressos: 15

Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```

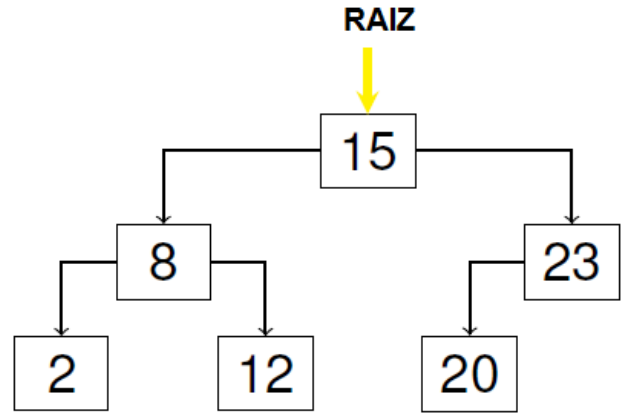


Elementos na Pilha - topo: 2 12 23

Valores impressos: 15 8

Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```

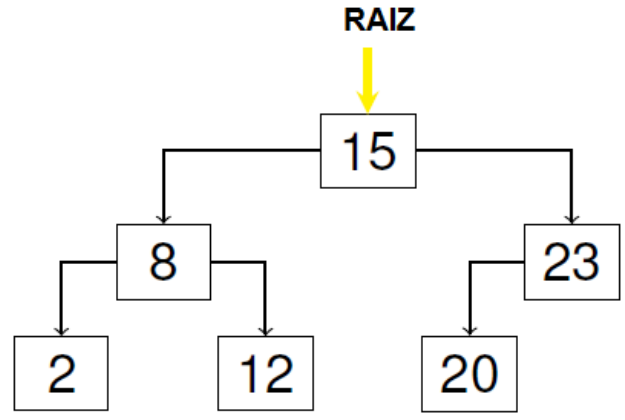


Elementos na Pilha - topo: 12 23

Valores impressos: 15 8 2

Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```

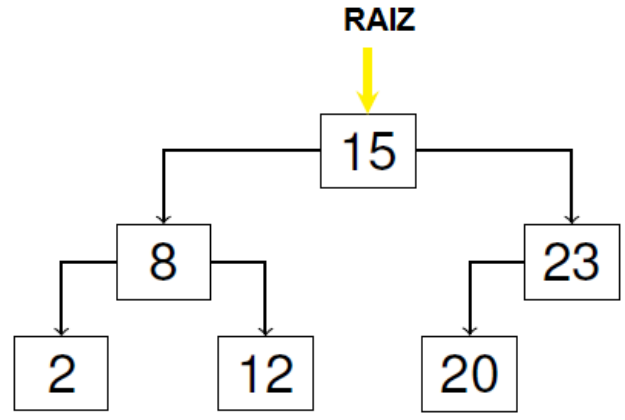


Elementos na Pilha - topo: 23

Valores impressos: 15 8 2 12

Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```

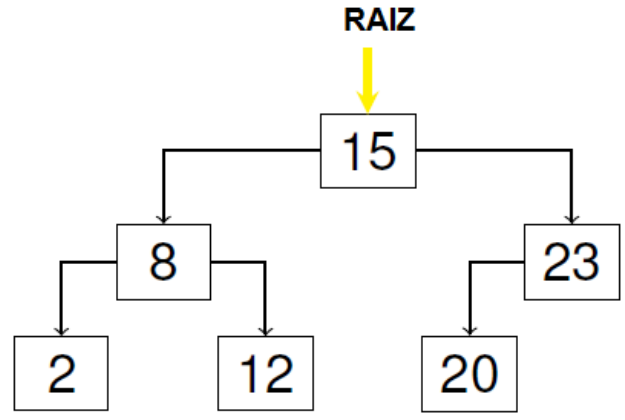


Elementos na Pilha - topo: 20

Valores impressos: 15 8 2 12 23

Árvores Binárias - Pré-ordem - iterativo

```
void preOrdemIterativo(PONT raiz){
  if (!raiz) return;
  PILHA pi;
  PONT p;
  inicializarPilha(&pi);
  push(raiz, &pi);
  while (!isEmpty(&pi)) {
     p = pop(&pi);
     printf("%i ",p->chave);
     if (p->dir) push(p->dir, &pi);
     if (p->esq) push(p->esq, &pi);
  }
  printf("\n");
}
```



Elementos na Pilha - topo:

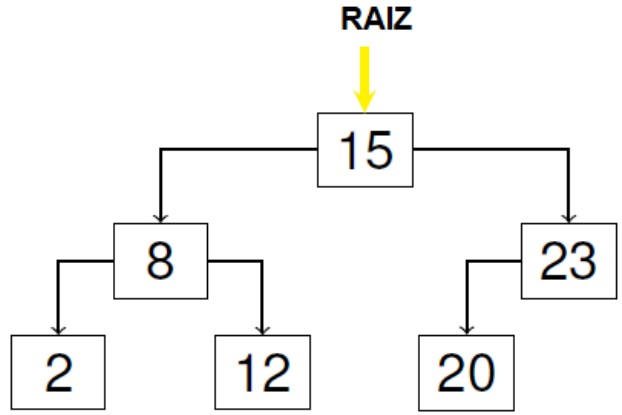
Valores impressos: 15 8 2 12 23 20

Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
     entrarFila(p,&f);
     while((f.inicio)){
         p = sairFila(&f);
         printf("%i ", p->chave);
         if(p->esq) entrarFila(p->esq, &f);
         if(p->dir) entrarFila(p->dir, &f);
     }
  }
}
```

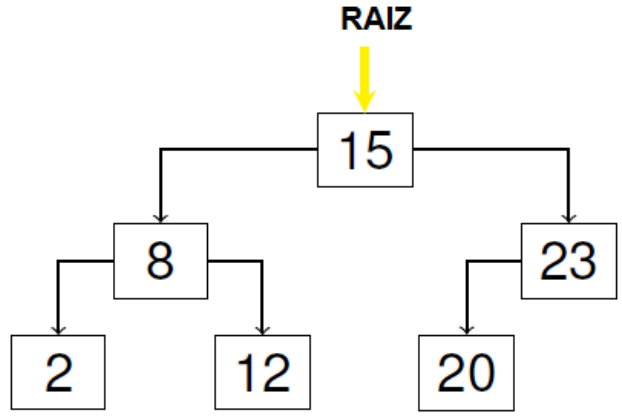
Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```



Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```

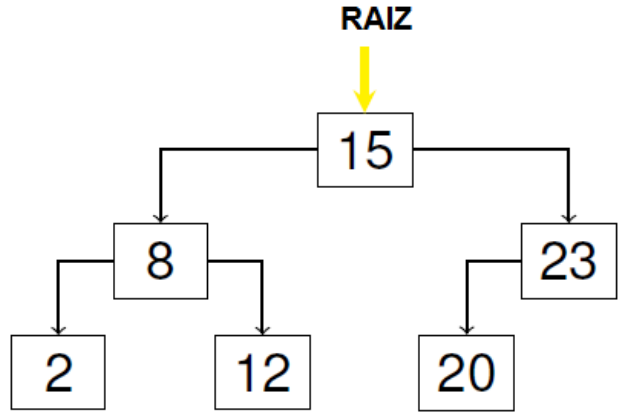


Elementos na Fila - inicio:

Valores impressos:

Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```

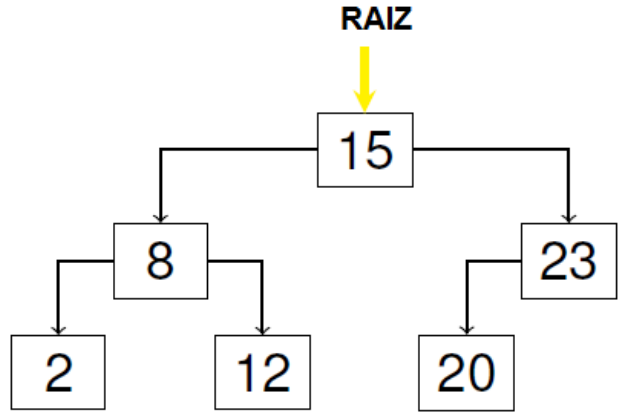


Elementos na Fila - inicio: 15

Valores impressos:

Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```

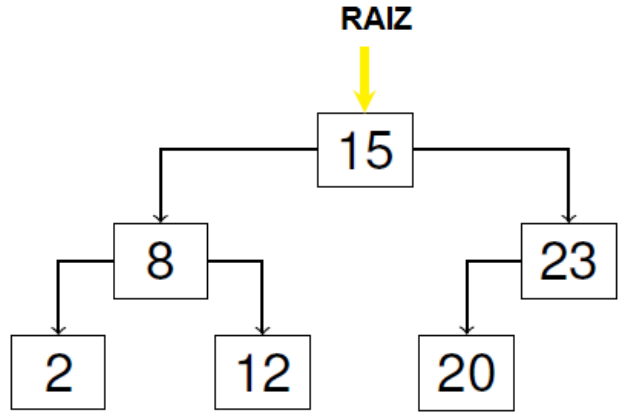


Elementos na Fila - inicio: 8 23

Valores impressos: 15

Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```

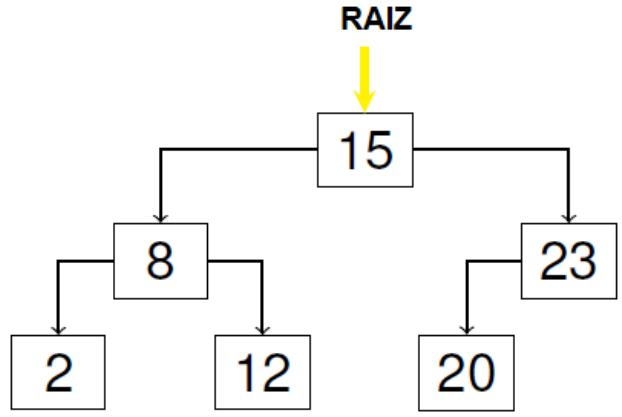


Elementos na Fila - inicio: 23 2 12

Valores impressos: 15 8

Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```

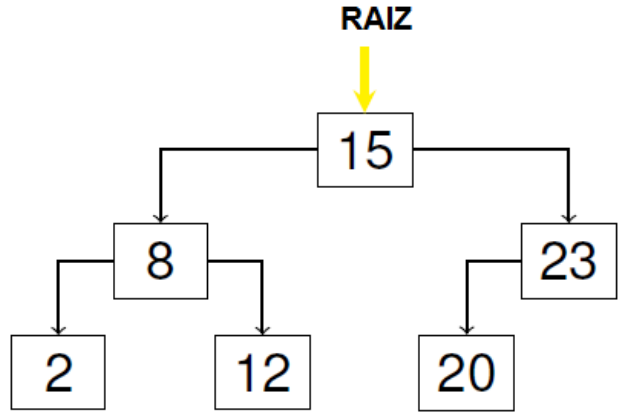


Elementos na Fila - inicio: 2 12 20

Valores impressos: 15 8 23

Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```

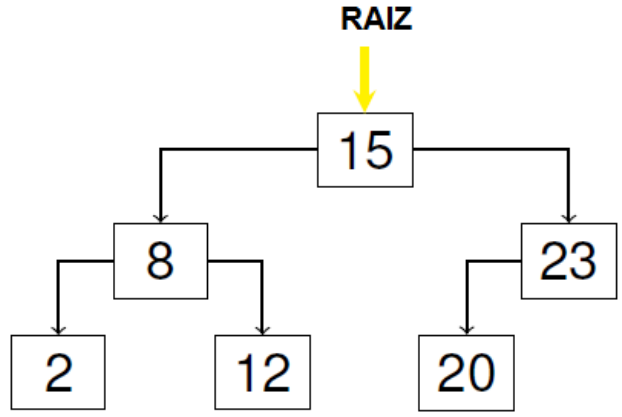


Elementos na Fila - inicio: 12 20

Valores impressos: 15 8 23 2

Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```

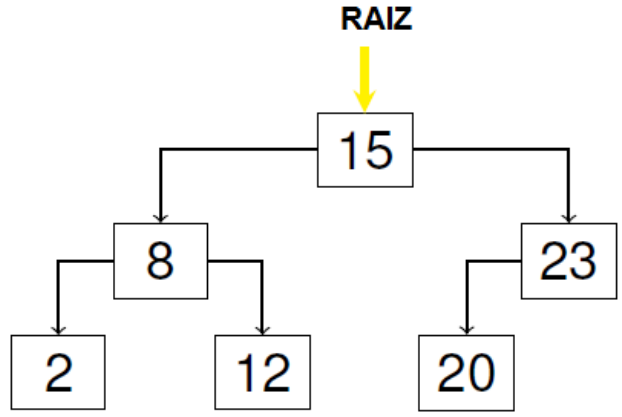


Elementos na Fila - inicio: 20

Valores impressos: 15 8 23 2 12

Árvores Binárias - Em nível

```
void exibirArvoreEmNivel(PONT p) {
  FILA f;
  inicializarFila(&f);
  if (p){
    entrarFila(p,&f);
    while((f.inicio)){
       p = sairFila(&f);
       printf("%i ", p->chave);
       if(p->esq) entrarFila(p->esq, &f);
       if(p->dir) entrarFila(p->dir, &f);
    }
  }
}
```



Elementos na Fila - inicio:

Valores impressos: 15 8 23 2 12 20

Árvores Binárias - Busca

A busca em uma árvore "**apenas**" binária (sem nenhuma ordenação no valor de seus nós) precisará, no pior caso, percorrer todos os nós. A princípio, usando qualquer um dos percursos que vimos.

Implementaremos, sem perda de generalidade, uma busca que utilize a pré-ordem: investigue primeiro o nó atual, se não encontrar o nó buscado, procure, recursivamente, à esquerda e se novamente não encontrar, procure, recursivamente à direita.

Árvores Binárias - Busca

```
PONT buscarChave(TIPOCHAVE ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

Árvores Binárias - Inserção

A inserção em uma árvore binária (que não tenha uma ordenação de chaves a ser cumprida) é tipicamente feita de acordo com a indicação do usuário. Isto é, usuário diz qual nó será o pai de um novo nó e a criação da raiz é, tipicamente, feita a parte.

Não há uma definição padrão do que como será feita a inserção se o nó pai já tiver um filho no local onde o usuário solicitou a inserção (pode-se não inserir, inserir e excluir o nó antigo, transformar o nó antigo em filho do novo nó, etc [é algo que precisa ser definido na especificação do problema]).

Árvores Binárias - Inserção

```
PONT criarNovoNo(TIPOCHAVE ch){
    PONT novoNo = (PONT)malloc(sizeof(NO));
    novoNo->esq = NULL;
    novoNo->dir = NULL;
    novoNo->chave = ch;
    return novoNo;
}

void criarRaiz(PONT * raiz, TIPOCHAVE novaChave){
     *raiz = criarNovoNo(novaChave);
}
```

Árvores Binárias - Inserção

```
typedef enum{esquerdo, direito} LADO;

bool inserirFilho(PONT raiz, TIPOCHAVE novaChave,
                          TIPOCHAVE chavePai, LADO lado){
  PONT pai = buscarChave(chavePai,raiz);
  if (!pai) return false;
  PONT novo = criarNovoNo(novaChave);
  if (lado == esquerdo){
     novo->esq = pai->esq;
     pai->esq = novo;
  }else{
     novo->esq = pai->dir;
     pai->dir = novo;
  }
  return true;
}
```

Árvores de Busca Binária (ABB)

Ou Árvores Binárias de Busca (ABB) ou Árvores Binária de Pesquisa (ABP).

Uma árvore binária de pesquisa é uma árvore binária em que, a cada nó, todos os registros com chaves menores (ou menores e iguais) que a deste nó estão na subárvore da esquerda, enquanto que os registros com chaves maiores estão na subárvore da direita.

Árvores de Busca Binária - Busca

```
PONT buscaBinaria(TIPOCHAVE2 ch, PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  if (raiz->chave<ch)
    return buscaBinaria(ch,raiz->dir);
  return buscaBinaria(ch,raiz->esq);
}
```

Busca x Busca Binária

Busca

```
PONT buscarChave(TIPOCHAVE ch,
                            PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  PONT aux = buscarChave(ch,raiz->esq);
  if (aux) return aux;
  return buscarChave(ch,raiz->dir);
}
```

Busca Binária

```
PONT buscaBinaria(TIPOCHAVE ch,
                            PONT raiz){
  if (raiz == NULL) return NULL;
  if (raiz->chave == ch) return raiz;
  if (raiz->chave<ch)
    return buscaBinaria(ch,raiz->dir);
  return buscaBinaria(ch,raiz->esq);
}
```

Árvores de Busca Binária - Busca

Implementação Iterativa

```
PONT buscaBinariaIterativa(PONT raiz, TIPOCHAVE ch){
  PONT atual = raiz;
  while (atual) {
    if(atual->chave == ch) return atual;
    if(ch < atual->chave) atual = atual->esq;
    else atual = atual->dir;
  }
  return NULL;
}
```

Árvores de Busca Binária - Inserção

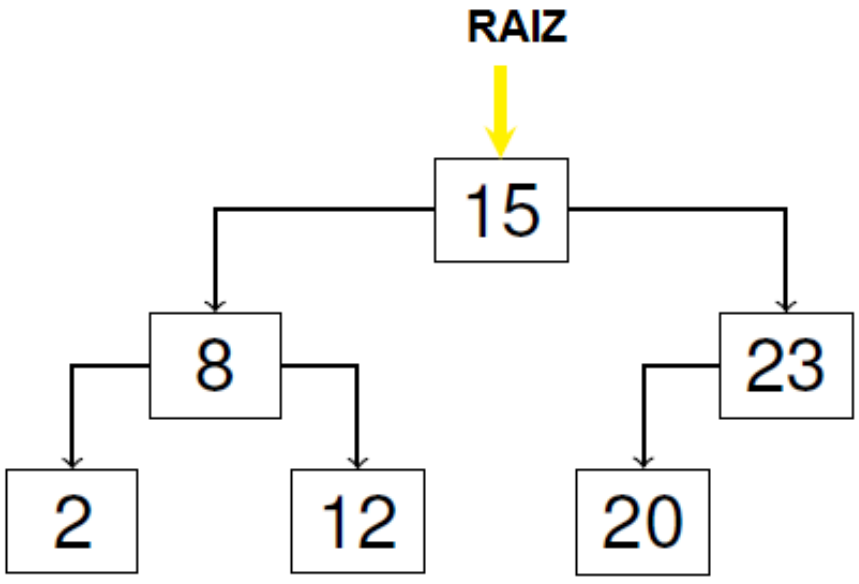
A inserção em uma árvore de busca binária se dá pelo valor da chave do registro que será inserido.

A implementação pode ou não permitir a inserção de chaves repetidas.

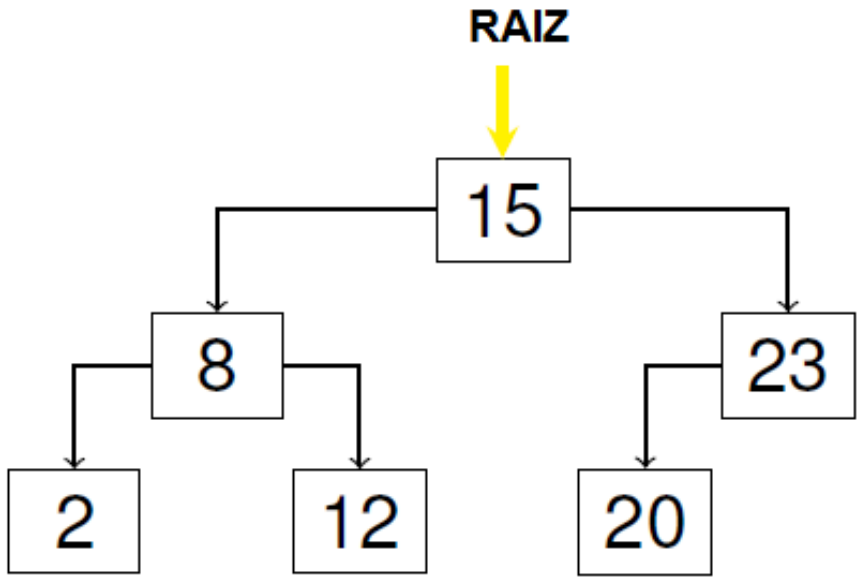
A regra de organização é a seguinte: chaves menores (ou iguais) devem ser inseridas à esquerda do nó atual; chaves maiores devem ser inseridas à direita.

Vamos inserir um nó apenas quando encontrarmos "um espaço livre" para ele, seguindo a regra de organização apresentada.

Árvores de Busca Binária - Inserção

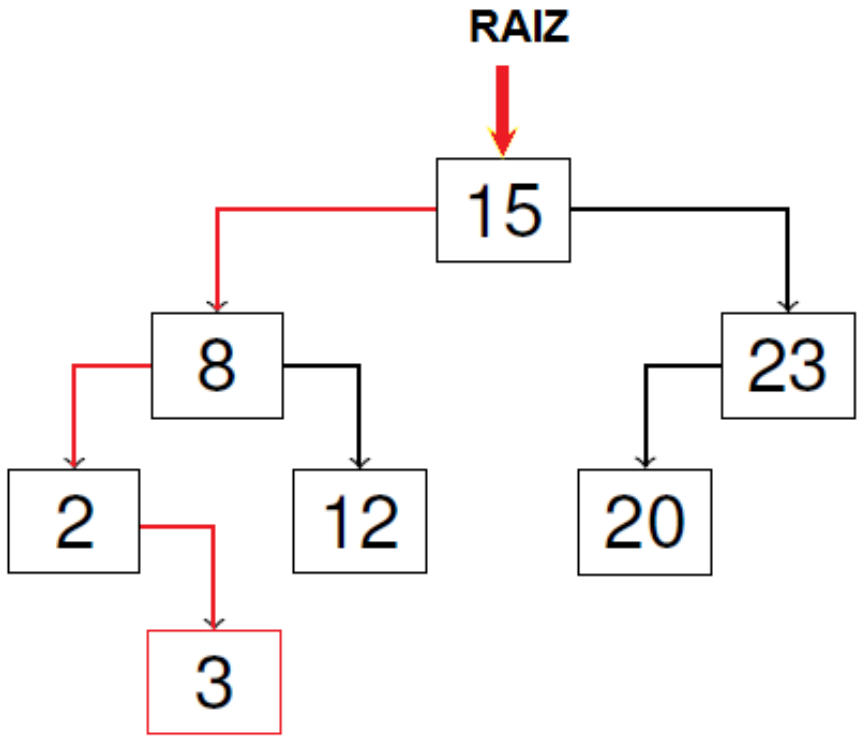


Árvores de Busca Binária - Inserção



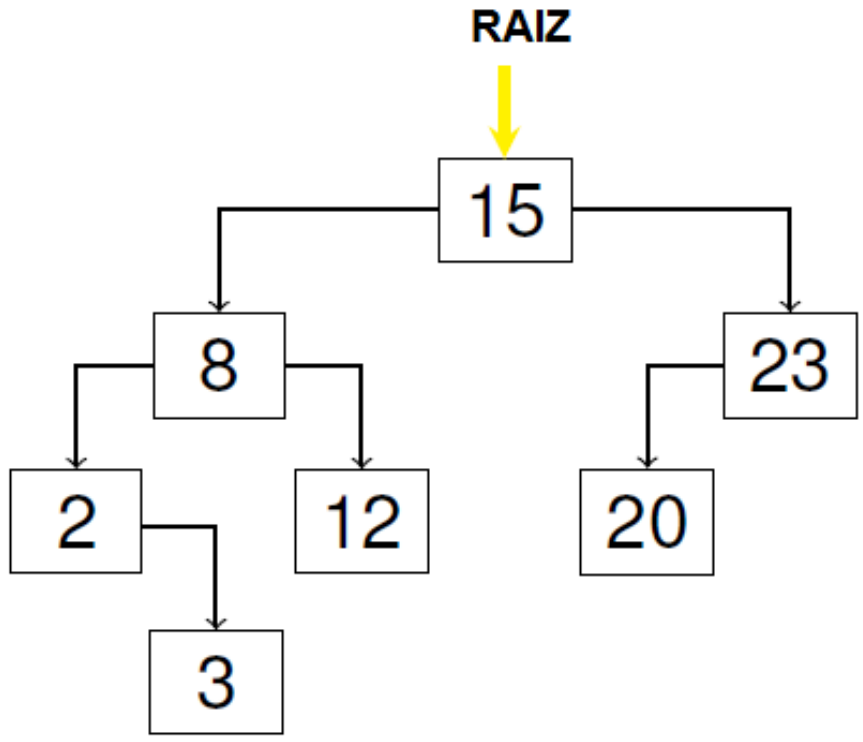
Inserção do elemento com chave igual a 3.

Árvores de Busca Binária - Inserção



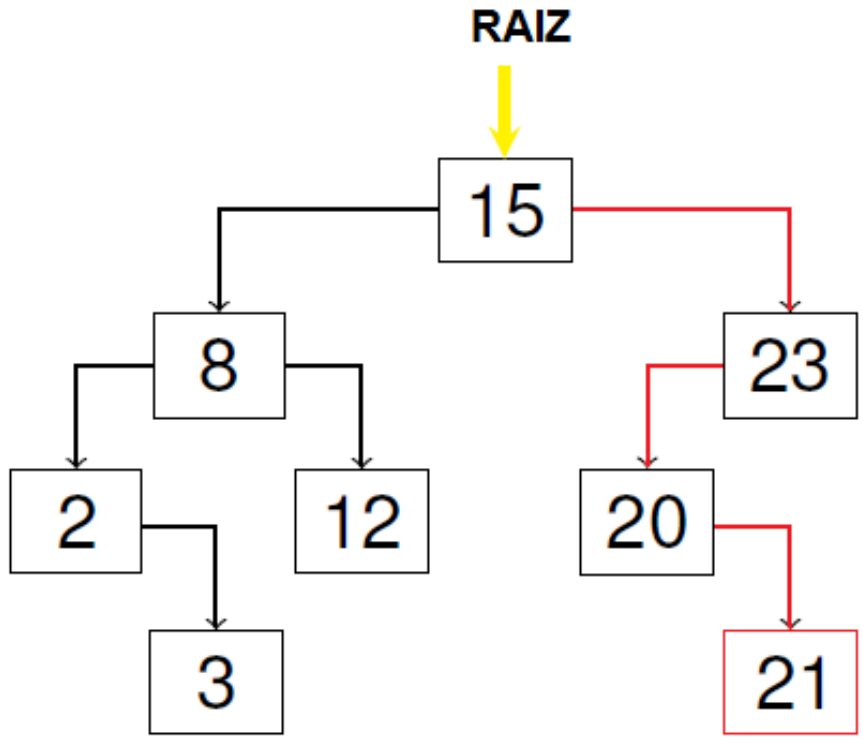
Inserção do elemento com chave igual a 3.

Árvores de Busca Binária - Inserção



Inserção do elemento com chave igual a 21.

Árvores de Busca Binária - Inserção



Inserção do elemento com chave igual a 21.

Árvores de Busca Binária - Inserção

```
bool inserirNo2(PONT* raiz, TIPOCHAVE novaChave){
  if (*raiz == NULL){
    *raiz = criarNovoNo(novaChave);
    return true;
  }
  else return inserirAux(*raiz, novaChave);
}
```

Árvores de Busca Binária - Inserção

```
bool inserirAux(PONT raiz,
                      TIPOCHAVE ch){
  if (raiz->chave >= ch) {
    if (raiz->esq == NULL) {
      raiz->esq = criarNovoNo(ch);
      return true;
    }else
   return inserirAux(raiz->esq, ch);
  }else{
    if (raiz->dir == NULL) {
      raiz->dir = criarNovoNo(ch);
      return true;
    }else
   return inserirAux(raiz->dir, ch);
  }
}
```

Inserção x Inserção em Árvore de Busca

Árvore Binária

```
bool inserirFilho(PONT raiz,
                  TIPOCHAVE ch,
       TIPOCHAVE chPai, LADO lado){
  PONT pai = buscarChave(chPai,raiz);
  if (!pai) return false;
  PONT novo = criarNovoNo(ch);
  if (lado == esquerdo){
     novo->esq = pai->esq;
     pai->esq = novo;
  }else{
     novo->esq = pai->dir;
     pai->dir = novo;
  }
  return true;
}
```

Árvore de Busca Binária

```
bool inserirAux(PONT raiz,
                    TIPOCHAVE ch){
  if (raiz->chave >= ch) {
    if (raiz->esq == NULL) {
      raiz->esq = criarNovoNo(ch);
      return true;
    }else
   return inserirAux(raiz->esq, ch);
  }else{
    if (raiz->dir == NULL) {
      raiz->dir = criarNovoNo(ch);
      return true;
    }else
   return inserirAux(raiz->dir, ch);
  }
}
```

ABB - Inserção Iterativa

```
bool inserirNoIterativo(PONT* raiz, TIPOCHAVE ch){
  if (*raiz == NULL){
    *raiz = criarNovoNo(ch);
    return true;
  } else {
    PONT atual = *raiz;
    while (atual){
      if (atual->chave >= ch) {
        if (atual->esq == NULL) {
          atual->esq = criarNovoNo(ch);
          return true;
        }else atual = atual->esq;
      }else{
        if (atual->dir == NULL) {
          atual->dir = criarNovoNo(ch);
          return true;
        }else atual = atual->dir;
      }
    }
  }
  return false;
}
```

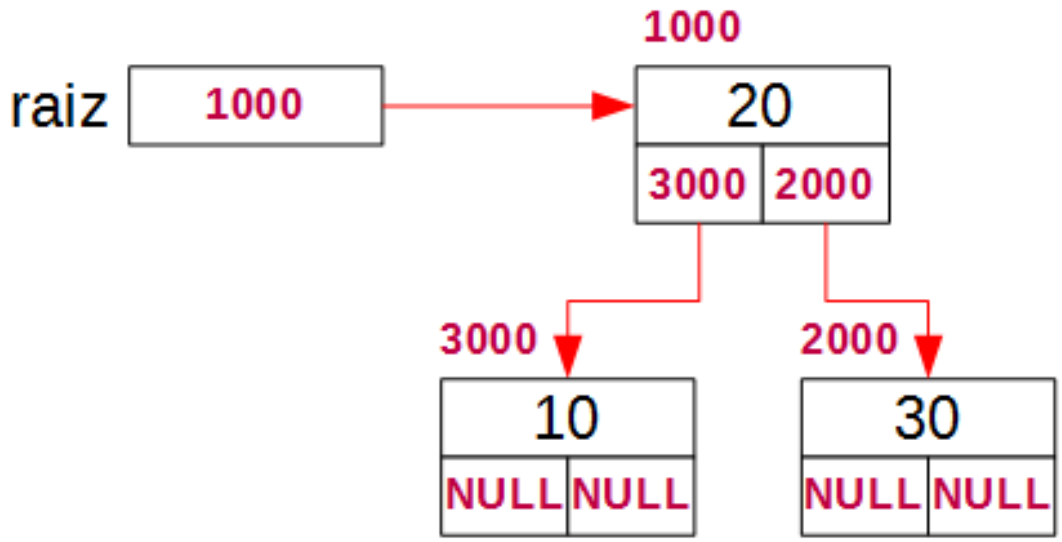
Árvores de Busca Binária - Inserção

Algoritmo Recursivo Único (insere raiz ou qualquer outro nó)

```
bool inserirNo(PONT* raiz, TIPOCHAVE novaChave){
  if (*raiz == NULL){
    *raiz = criarNovoNo(novaChave);
    return true;
  }
  else if ((*raiz)->chave >= novaChave){
        return inserirNo(&(*raiz)->esq, novaChave);
      }else return inserirNo(&(*raiz)->dir, novaChave);
}
```

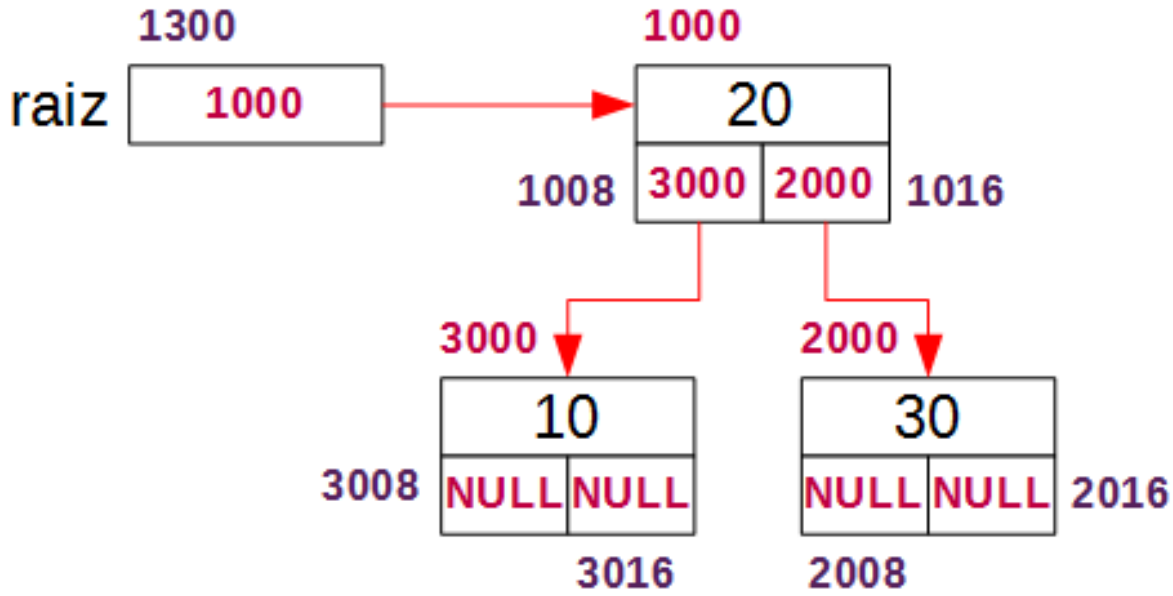
Árvores

```
bool inserirNo(PONT* raiz,
               TIPOCHAVE novaChave){
  if (*raiz == NULL){
    *raiz = criarNovoNo(novaChave);
    return true;
  }
  else if ((*raiz)->chave >= novaChave){
         return inserirNo(&(*raiz)->esq, novaChave);
       }else return inserirNo(&(*raiz)->dir, novaChave);
}
```



Árvores

```
bool inserirNo(PONT* raiz,
               TIPOCHAVE novaChave){
  if (*raiz == NULL){
    *raiz = criarNovoNo(novaChave);
    return true;
  }
  else if ((*raiz)->chave >= novaChave){
         return inserirNo(&(*raiz)->esq, novaChave);
       }else return inserirNo(&(*raiz)->dir, novaChave);
}
```



Árvores de Busca Binária - Exclusão

O usuário informa a chave do nó que deverá ser excluído.

Se o nó existir, há três possibilidades:

1. Nó que está sendo excluído não tem filhos: o nó será excluído e quem apontava para ele (pai ou endereço que apontava para a raiz) deve apontar para *NULL*.
2. Nó que está sendo excluído tem um único filho: o nó será excluído e quem apontava para ele (pai ou endereço que apontava para a raiz) deve apontar para o filho do nó a ser excluído.
3. Nó que está sendo excluído tem dois filhos: para não desorganizar a árvore será necessário encontrar um "substituto" para ele, antes de realizar a exclusão.

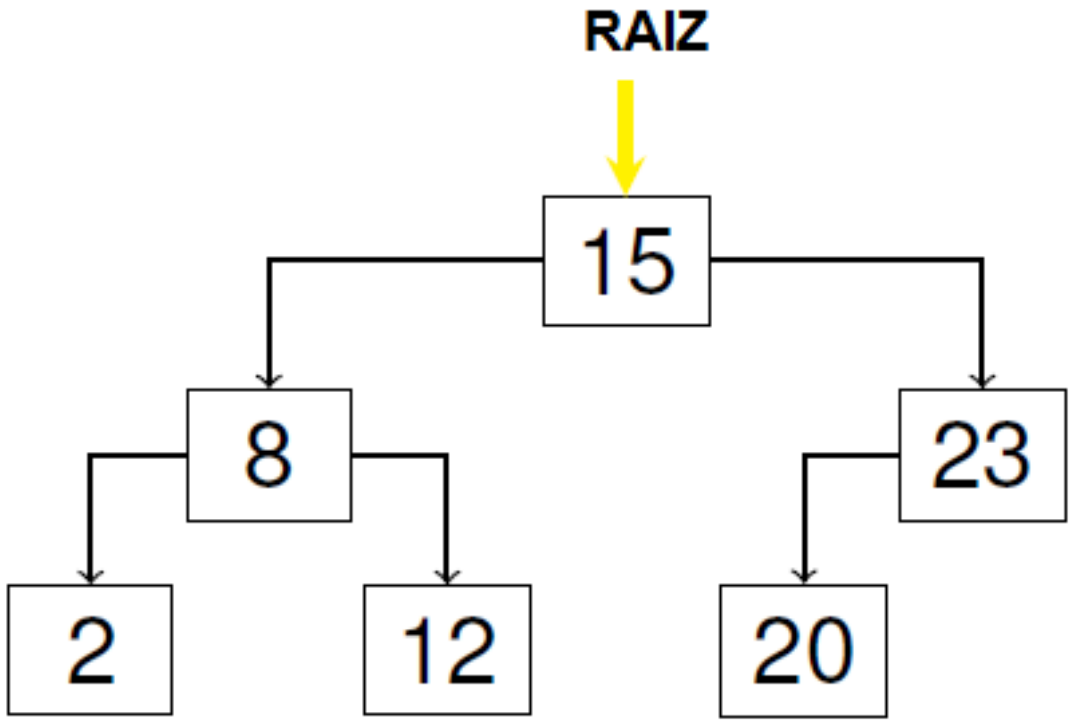
Árvores de Busca Binária - Exclusão

3. Nó que está sendo excluído tem dois filhos: para não desorganizar a árvore será necessário encontrar um "substituto" para ele, antes de realizar a exclusão.

Na árvore há outros dois nós que poderiam estar no local do nó que será excluído. O maior entre os elementos menores do que ele (ou menores iguais a ele) ou o menor entre os elementos maiores do que ele.

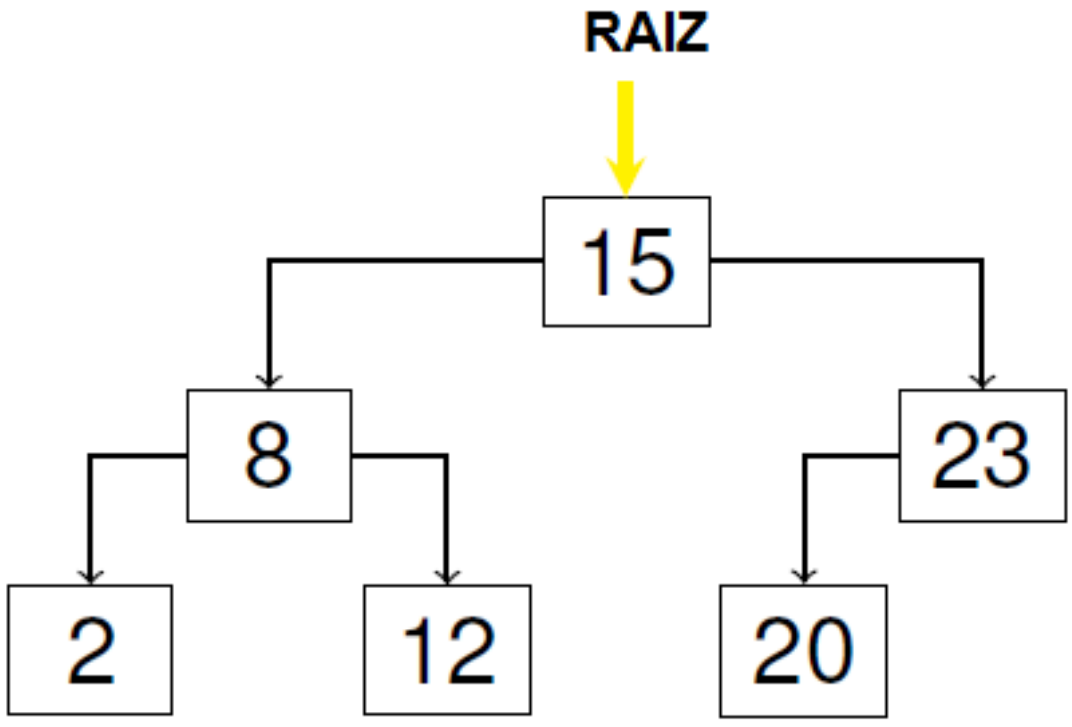
Utilizaremos o maior entre os menores (ou iguais) a ele.

Árvores de Busca Binária - Exclusão



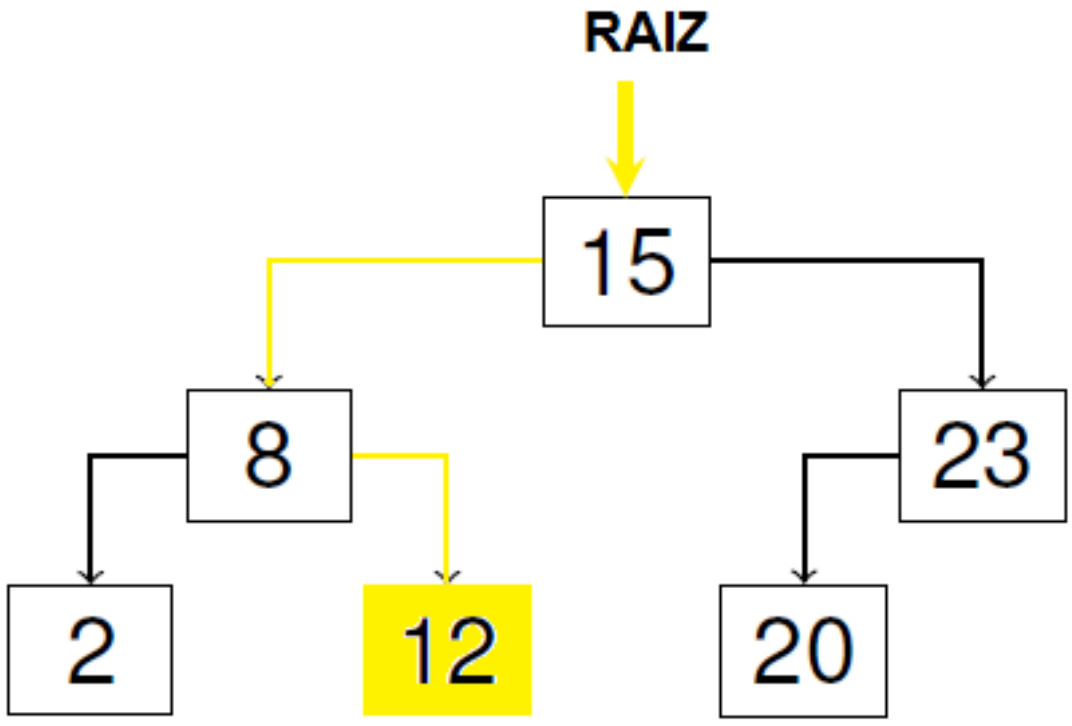
Exclusão do elemento com chave igual a 15.

Árvores de Busca Binária - Exclusão



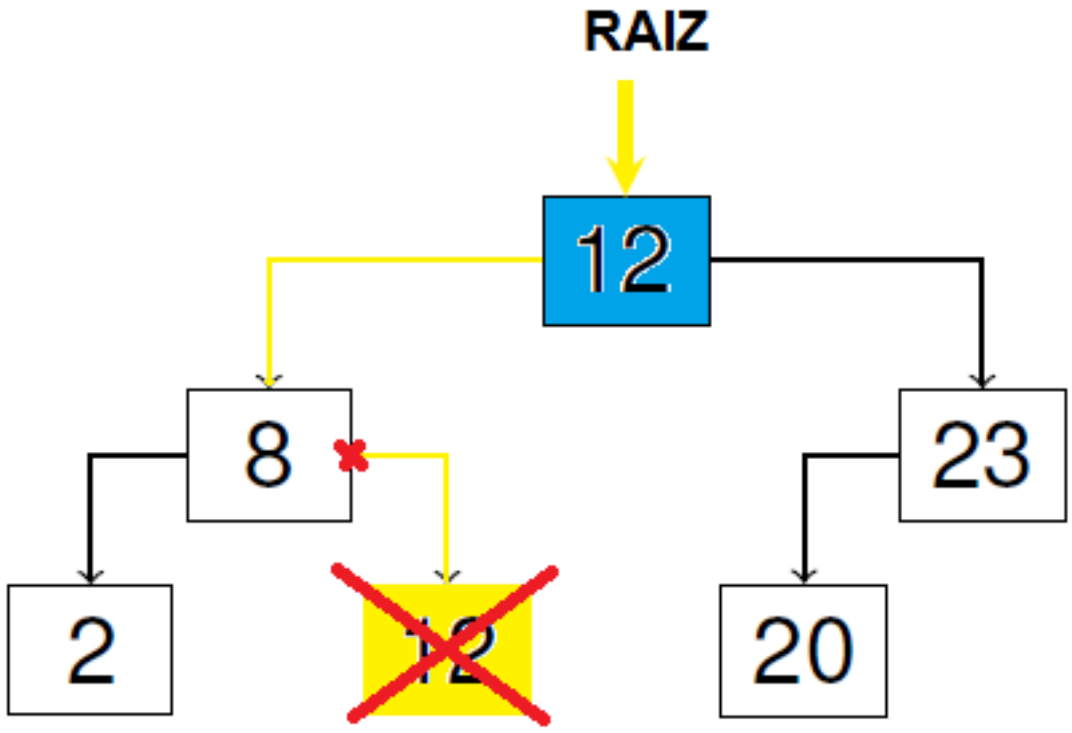
Exclusão do elemento com chave igual a 15: encontrar o maior à esquerda.

Árvores de Busca Binária - Exclusão



Exclusão do elemento com chave igual a 15: encontrar o maior à esquerda.

Árvores de Busca Binária - Exclusão



Exclusão do elemento com chave igual a 15: realizar a substituição.

Árvores de Busca Binária - Exclusão

```
PONT maior_esquerdo(PONT p, PONT* ant){
  *ant = p;
  p = p->esq;
  while (p->dir) {
    *ant = p;
    p = p->dir;
  }
  return(p);
}
```

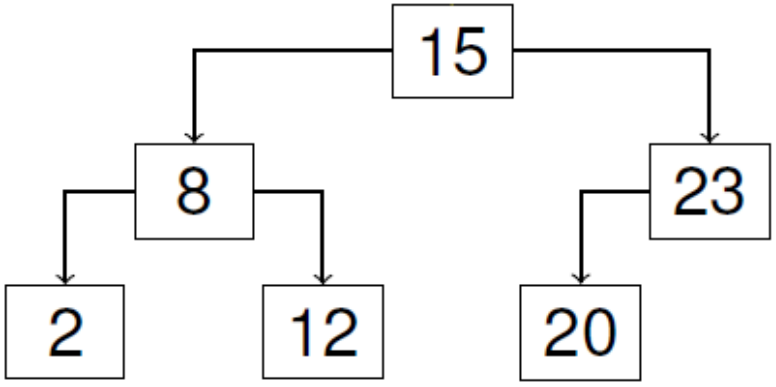
Árvores de Busca Binária - Exclusão

```
bool excluirNo(PONT *raiz, TIPOCHAVE ch){
  PONT atual, no_pai, substituto, pai_substituto;
  atual = buscaNo(*raiz, ch, &no_pai);
  if(atual == NULL) return false;
  if(!atual->esq || !atual->dir) { // tem zero ou um filho
    if(atual->esq) substituto = atual->esq;
    else substituto = atual->dir;
    if(!no_pai) { // no excluido eh raiz
       *raiz = substituto;
    }else {
      if(no_pai->esq == atual) no_pai->esq = substituto;
      else no_pai->dir = substituto;
    }
    free(atual);
  } else {   // tem dois filhos
    substituto = maior_esquerdo(atual, &pai_substituto);
    atual->chave = substituto->chave;
    if (pai_substituto->esq == substituto)
              pai_substituto->esq = substituto->esq;
    else pai_substituto->dir = substituto->esq;
    free(substituto);
  }
  return true;
}
```

Árvores Binárias - Complexidade

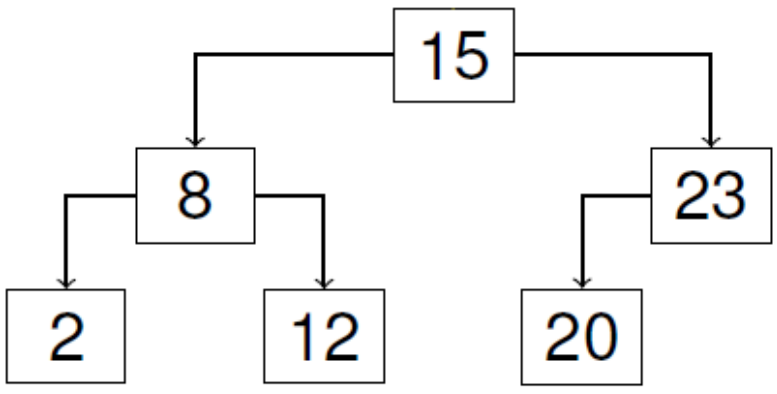
	Árvore Binária	Árvore de Busca Binária
Busca	$O(n)$	$O(h); h \in O(n)$
Inserção	$O(n)$	$O(h); h \in O(n)$
Exclusão	$O(n)$	$O(h); h \in O(n)$

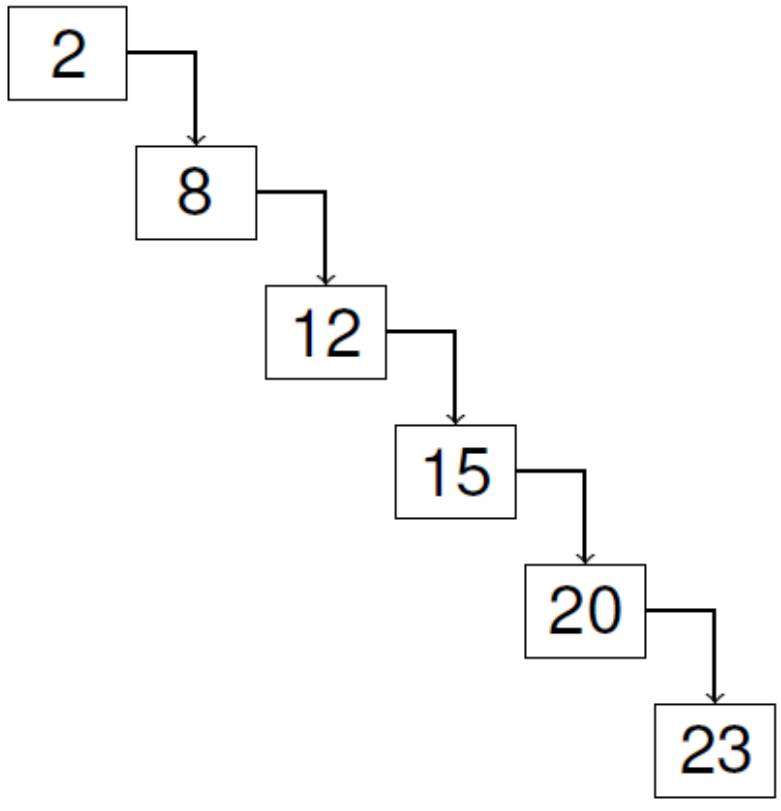
Árvores de Busca Binária - Desbalanceamento



Árvores de Busca Binária - Desbalanceamento

Duas árvores com os mesmos nós, inseridos em ordens diferentes.





SIN5013
Análise de Algoritmos e Estruturas de Dados

Árvores

Prof. Luciano Antonio Digiampietri