Algoritmos e Estruturas de Dados I

Árvores Binárias de Busca

versão 2.3

Fabiano Oliveira
fabiano.oliveira@ime.uerj.br

ArvoreBusca <TChave, TElem>

procedimento Enumera(**ref** T: ArvoreBusca)

Enumera (lista) cada chave de L ordenadamente

função Busca(**ref** T: ArvoreBusca, c: <TChave>): <TElem>

Obtém o elemento de T com chave c ou NULO se inexistente

procedimento Insere(**ref** T: ArvoreBusca, c: <TChave>, x: <TElem>)

Insere x com chave c em T

função Remove(**ref** T: ArvoreBusca, c: <TChave>): <TElem>

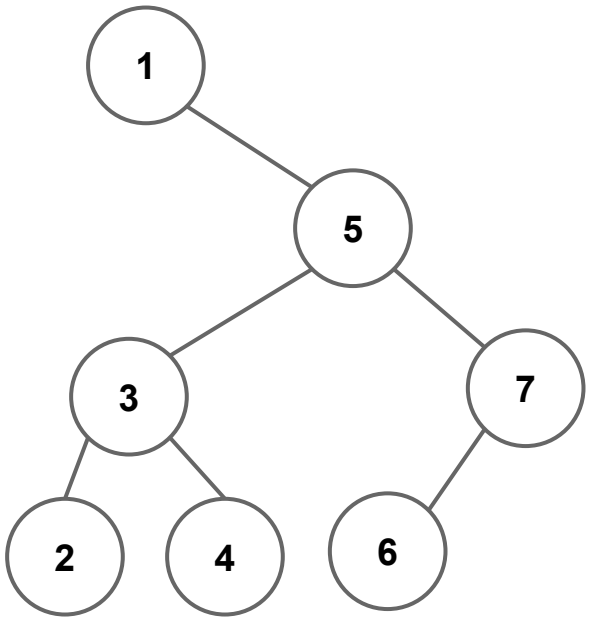
Remove e retorna o elemento de T com chave c

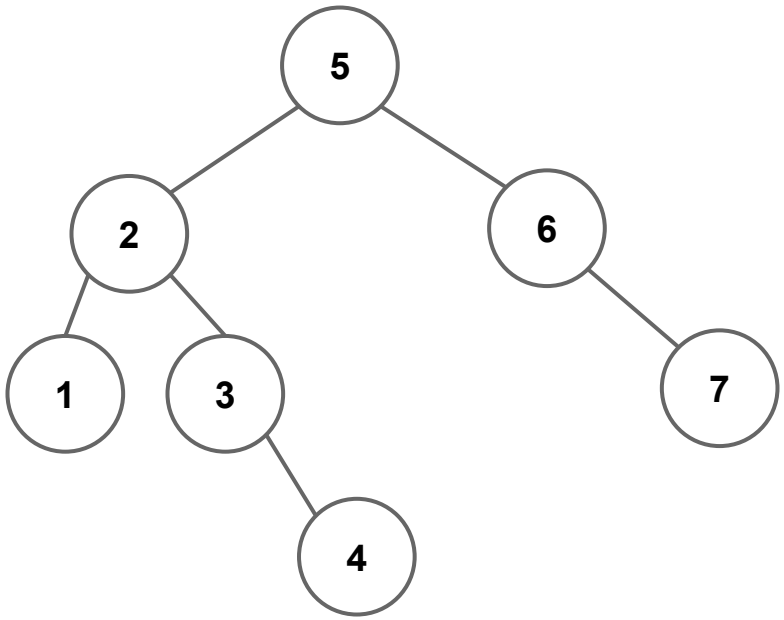
Árvores Binárias de Busca

- Uma ***árvore binária de busca*** T é uma árvore binária tal que cada nó de T possui uma chave distinta e se R é nó raiz de T, então:
  - para todo nó Q da subárvore esquerda de R, Q.Chave < R.Chave
  - para todo nó P da subárvore direita de R, Q.Chave > R.Chave
  - subárvores à esquerda e à direita de R são árvores de busca

Árvores Binárias de Busca

- Existem diversas árvores binárias de busca para um mesmo conjunto não-unitário de chaves:





Árvores Binárias de Busca

- **estrutura** ArvoreBusca = **estrutura** Arvore

Árvores Binárias de Busca

- Enumeração

procedimento Enumera(**ref** T: ArvoreBusca)
    PercursoInOrdem(T.Raiz)

Tempo:
$\theta(N)$

Árvores Binárias de Busca

- Busca

```
função Busca(ref T: ArvoreBusca, c: <TChave>): <TElem>
    var Q: ^No ← T.Raiz
    enquanto Q ≠ NULO faça
        se Q^.Chave = c então
            retornar (Q^.Elem)
        senão se c < Q^.Chave então
            Q ← Q^.Esq
        senão
            Q ← Q^.Dir
    retornar ("NULO")
```

Tempo:
O(h(T)) = O(N)

Árvores Binárias de Busca

- Busca

```
função Busca(ref T: ArvoreBusca, c: <TChave>,
             ref PosIns: ^^No): <TElem>
    var Q: ^No ← T.Raiz
    PosIns ← @(T.Raiz)
    enquanto Q ≠ NULO faça
        se Q^.Chave = c então
            retornar (Q^.Elem)
        senão se c < T^.Chave então
            Q, PosIns ← Q^.Esq, @(Q^.Esq)
        senão
            Q, PosIns ← Q^.Dir, @(Q^.Dir)
    retornar ("NULO")
```

Esta variante é interessante para informar onde o elemento procurado deveria estar no caso em que a chave não é encontrada

Árvores Binárias de Busca

- Inserção

```
procedimento Insere(ref T: ArvoreBusca, c: <TChave>,
                                        x: <TElem>)
   var PosIns: ^^No, Q: ^No, v: <TElem>
   v ← Busca(T, c, PosIns)
   se v = "NULO" então
      alocar(Q)
      PosIns^, Q^.Chave, Q^.Elem, Q^.Esq, Q^.Dir ←
         Q, c, x, NULO, NULO
   senão
      Exceção("Chave existente")
```

Tempo:
O(h(T)) = O(N)

Árvores Binárias de Busca

- Construção
  - O algoritmo consiste de inserir as N chaves uma a uma, até que a árvore esteja construída. Este algoritmo tem complexidade $O(N^2)$.

Árvores Binárias de Busca

- Remoção

```
função Remove(ref T: ArvoreBusca, c: <TChave>): <TElem>
    var Q: ^No, pontQ: ^^No, x: <TElem>
    x ← Busca(T, c, pontQ)
    se x ≠ "NULO" então
        Q ← pontQ^
        //remoção de Q -- slide seguinte
        desalocar (Q)
        retornar (x)
    senão
        Exceção("Chave inexistente")
```

Árvores Binárias de Busca

```
//remoção de Q
se Q^.Esq ≠ NULO então
    se Q^.Esq^.Dir = NULO então
        Q^.Esq^.Dir, pontQ^ ← Q^.Dir, Q^.Esq
    senão
        var Qmenor, paiQmenor: ^No ← Q^.Esq^.Dir, Q^.Esq
        enquanto Qmenor^.Dir ≠ NULO faça
            Qmenor, paiQmenor ← Qmenor^.Dir, Qmenor
        paiQmenor^.Dir, Qmenor^.Esq, Qmenor^.Dir, pontQ^
            ← Qmenor^.Esq, Q^.Esq, Q^.Dir, Qmenor
senão
    pontQ^ ← Q^.Dir
```

Exercícios

Exercícios

1. Quantas árvore binárias de busca podem representar o conjunto de chaves {1, 2, 3, 4, 5, 6}?

2. Escreva um algoritmo que ordene um vetor A de N elementos com o seguinte método; (a) criar uma árvore binária de busca com os elementos como chaves, inserindo as chaves A[1], A[2], …, A[N] nesta ordem na árvore binária de busca; (b) faça um percurso inordem, onde a visita a um nó X corresponde à escrita da chave de X. Qual a complexidade de espaço e de tempo deste algoritmo?

3. Escreva uma versão de Busca() para árvores binárias de busca que, no caso de não encontrar o elemento, retorne em duas variáveis passadas por referência: (i) o ponteiro para o elemento onde a chave buscada deveria ser filha (NULO se a chave buscada deveria ser a raiz) e (ii) um *flag* para indicar se a chave buscada deveria ser inserida como filho à esquerda ou á direita do nó retornado em (i)

Exercícios

4. Criar versões dos algoritmos Insere() e Busca() que sejam recursivos.

5. Dado um vetor ordenado, criar uma árvore binária de busca de altura mínima em tempo $\theta(N)$.

6. Dados um par de ponteiros A e B para nós de uma árvore binária T e um ponteiro para a raiz de T, determinar o ancestral comum mais perto de A e B com espaço auxiliar O(h(T)):
   a. se T é uma árvore binária de busca
   b. se T é uma árvore binária

7. Dada uma árvore binária de busca T com inteiros como chave e dois inteiros a, b, determinar os nós de T que possuem a chave entre a e b. O algoritmo deve ter complexidade O(h(T)+R), onde R é o número de nós que estão no resultado.

Exercícios

8. Elabore um algoritmo com complexidade de tempo O(N) que verifique se uma dada árvore binária é de busca.