Árvores binárias de busca

Prof. Marcel Hugo
Estruturas de Dados

Departamento de Sistemas e Computação
Universidade Regional de Blumenau – FURB

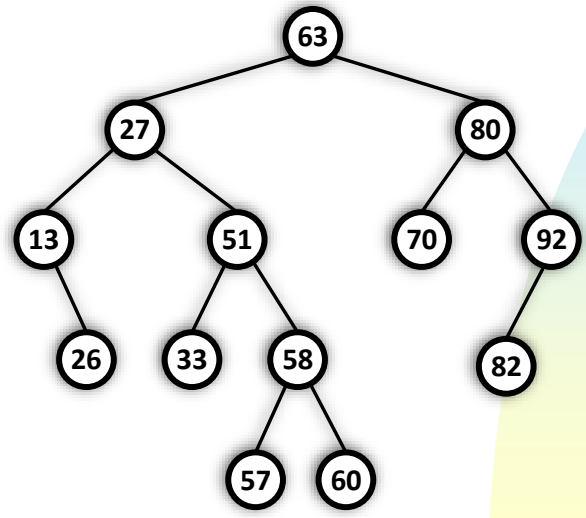
Slides criados a partir do material Profa. Patricia Dockhorn Costa, disciplina de Estrutura de Dados (UFES); Prof. David Menotti, disciplina de Algoritmos e Estruturas de Dados I, DECOM – UFOP; e do Prof. Paulo Rodacki Gomes, Disciplina de Algoritmo e Estrutura de Dados, DSC - FURB

Árvores binárias de busca
(Binary Search Trees – BST)

- É uma árvore binária que possui as seguintes características:
  - O filho à esquerda de um nó possui chave de busca menor que seu pai
  - O filho à direita de um nó possui uma chave de busca maior ou igual ao seu pai
- Chave de busca
  - É um valor utilizado para buscar um item numa árvore
  - A chave de busca deve ser comparável (igual, maior, menor)

Se usando tipos genéricos, deve-se marcar ***T extends Comparable***. O tipo concreto obrigatoriamente deve implementar *Comparable*.

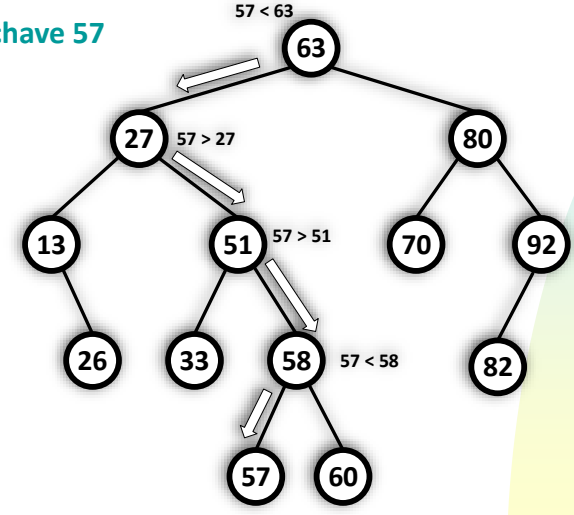
Exemplo de árvore de busca



3

Busca na árvore binária

Buscar chave 57



7

Localização de um nó na árvore
(algoritmo recursivo)

```
Algoritmo NóArvoreBST : buscar(int valor)
se (valor = no.info) então
  retornar no;
senão
   se (valor < no.info) então
      retornar no.esquerda.buscar(valor);
   senão
      retornar no.direita.buscar(valor);
   fim-se
fim-se
```

8

```
Algoritmo ArvoreBST: inserir(int valor)

Se (raiz = null) então
  raiz ← new NoArvoreBST(valor);
senão
  raiz.inserir (valor);
fim-se


Algoritmo NóArvoreBST: inserir(int valor)

Se (valor < no.info) então
   Se (no.esquerda = null) então
         no.esquerda = new NoArvoreBST(valor);
   senão
         no.esquerda.inserir(valor);
   fim-se
senão
   Se (no.direita = null) então
         no.direita = new NoArvoreBST(valor);
   senão
         no.direita.inserir(valor);
   fim-se
fim-se;
```

Inserção de nó

Algoritmo recursivo para inserir um nó numa árvore binária de busca:

Localiza em que posição deve ser adicionado o dado, similar ao algoritmo de busca

9

equals, Comparable, Comparator

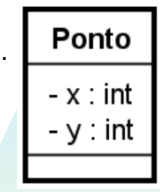
- Como se trabalha com tipos genéricos, os algoritmos utilizam métodos auxiliares de comparação dos objetos, ou seja, os algoritmos não realizam a comparação direta entre os objetos:
- equals (herdado de Object) – Igualdade de objetos
  public boolean equals(Object o)
- Comparable (interface) – ordem natural
  public interface Comparable<T> {
  public int compareTo(T o); }
- Comparator (interface)
  public interface Comparator<T> {
  int compare(T o1, T o2);}

10

Igualdade de Objetos

- **Igualdade de referência**
  - quando as variáveis fazem referência ao mesmo objeto.

```
Ponto p1 = new Ponto(2, 10);
Ponto p2 = p1;
```



  - a igualdade de objetos é verificada com o operador **==**
- **Igualdade lógica**
  - quando os atributos de dois objetos possuem os mesmos valores.

```
Ponto p1 = new Ponto(3, 5);
Ponto p2 = new Ponto(3, 5);
```

  - A igualdade de valores é verificada com o método **equals**

```
if (p1.equals(p2))
    System.out.print("Mesmo ponto");
```

11

11

Igualdade de Objetos

- Método `equals()` está na classe Object e sua implementação é:

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- para que seja possível realizer a igualdade lógica, devemos sobrescrever o método `equals()`.

12

Igualdade de Objetos

- A implementação do **equals** deve seguir um padrão (usando classe Ponto como exemplo):

```
1  public boolean equals(Object obj) {
2      if (this == obj)
3          return true;
4      if (obj == null)
5          return false;
6      if (getClass() != obj.getClass())
7          return false;
8      Ponto other = (Ponto) obj;
9      if (getX() != other.getX())
10         return false;
11     if (getY() != other.getY())
12         return false;
13     return true;
14 }
```

13

13

Ordem natural

- Comparable (interface) – ordem natural

  public interface Comparable<T> {

  public int compareTo(T o);

  }

  Retorno: um inteiro negativo, zero, ou um inteiro positivo se este objeto for menor que, igual a, ou maior que o objeto especificado como parâmetro.

- **Fortemente recomendado que ordenação natural seja consistente com equals :**
- `e1.compareTo(e2) == 0` mesmo valor booleano de `e1.equals(e2)`

14

14

Ordem natural

- De acordo com a natureza dos dados da classe, o desenvolvedor vai escolher quais atributos definem o critério de comparação e a maneira de compará-los.
- Por exemplo: classe Ponto

```
public int compareTo(Ponto outro) {
    if (this.getX() == outro.getX()) {
      return (this.getY() - outro.getY());
    }
    else {
      return (this.getX() - outro.getX());
    }
}
```

15

15

Implementação de Árvore Binária de Busca (BST)

- Lista 6

Mãos à obra !

16

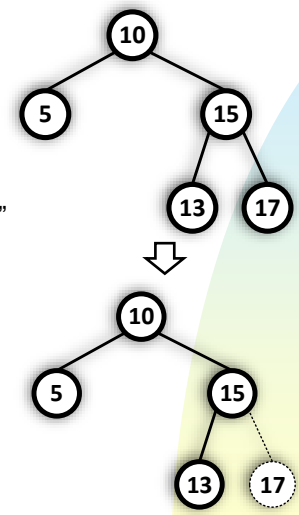
Exclusão de dados em árvores binárias de busca

- Consiste em duas etapas:
  - Utilizar um algoritmo para localizar o nó que contém o dado a ser removido, guardando quem é seu pai
  - Ao encontrar o nó, haverá três casos para considerar:
    - Caso 1 - O nó a ser removido é uma folha
    - Caso 2 - O nó a ser removido tem apenas um filho
    - Caso 3 - O nó a ser removido tem dois filhos

17

Caso 1 – Remover uma folha

- Ao identificar que o nó a ser removido é uma folha, basta remover a sua ligação com o nó pai
- Exemplo:
  - Remover nó que tem chave “17”



Quando o nó removido está à direita:
pai.direita ← null;

Quando o nó removido está à esquerda:
pai.esquerda ← null;

18

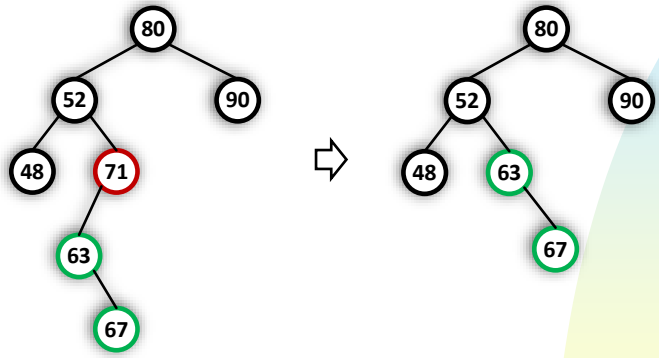
Caso 1 – Remover uma folha

- Caso especial
  - A folha a ser removida é a raiz da árvore

19

Caso 2 – Remover um nó com um filho

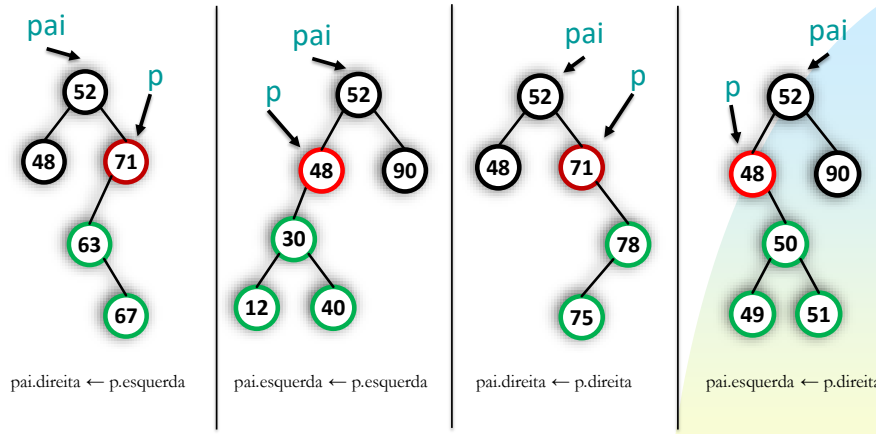
- Exemplo: remover chave “71”



O filho ocupa o lugar do nó removido.

20

Caso 2 – Remover um nó com um filho



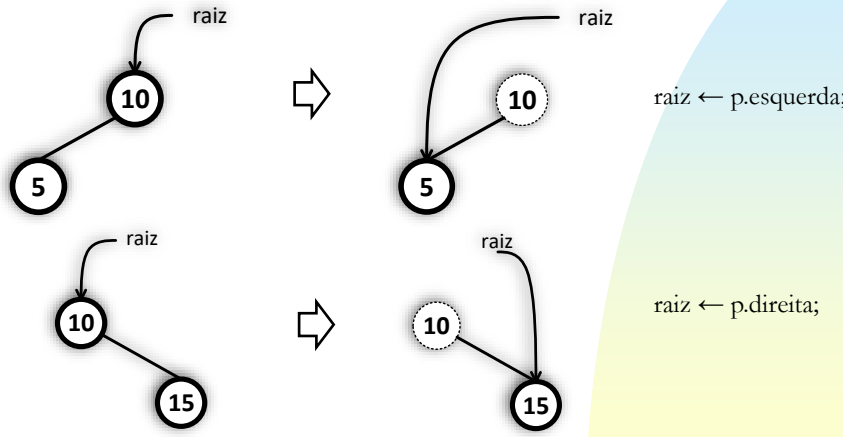
p.direita = null

p.esquerda = null

21

Caso 2 – Remover um nó com um filho

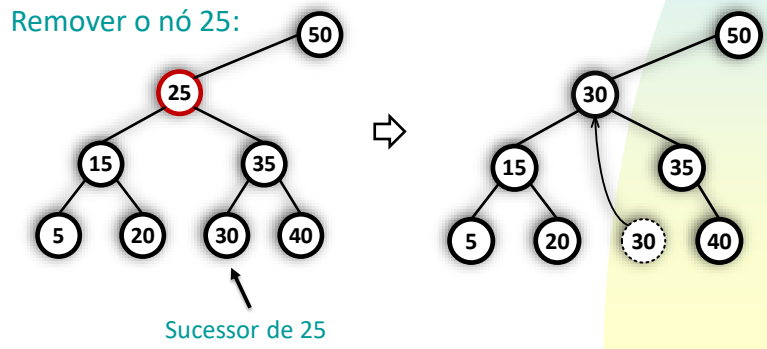
- **Caso especial**: O nó a ser removido é a raiz da árvore



22

Caso 3 – Remover nó com dois filhos

- Deve-se executar as seguintes etapas:
  - Localizar o próximo nó do nó a ser removido (denominado de "nó sucessor")
  - O nó sucessor deve tomar o lugar do nó a ser removido

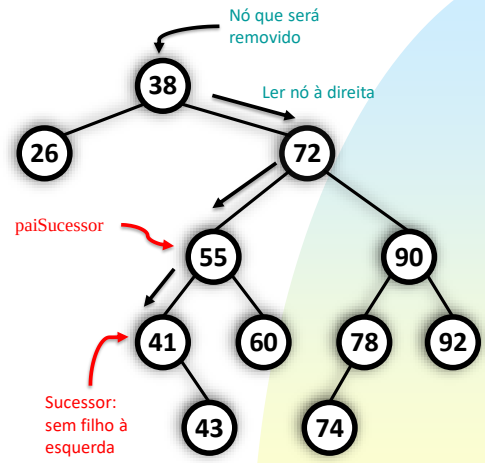


23

Caso 3 – Remover nó com dois filhos

Localizando sucessor

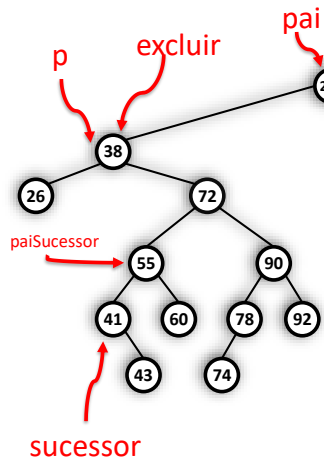
- Para localizar o nó sucessor, deve-se:
  1) Acessar o primeiro nó à direita e
  2) Caminhar até encontrar o último nó a esquerda

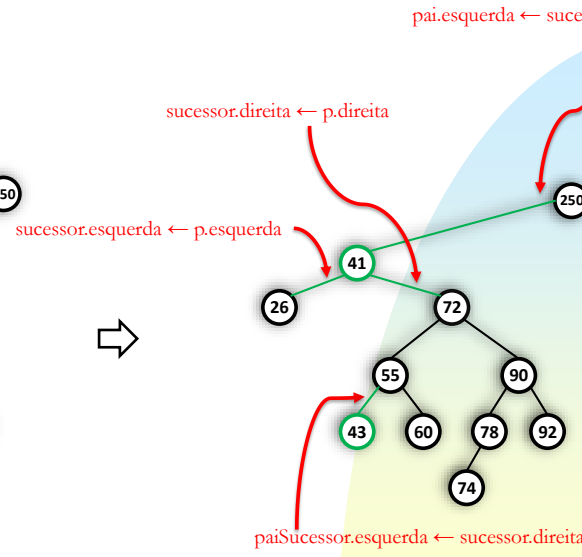


24

Exemplo

- remover chave “38”

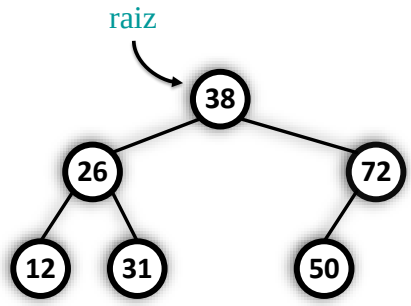




25

Caso 3 – Caso especial

- O nó a ser removido é a raiz:



26

Implementação de Exclusão em Árvore Binária de Busca (BST)

- Lista 7

Mãos à obra !

27

Desafios em Árvore Binária de Busca (BST)

- Lista 8

Mãos à obra !

28