Busca

Fabrício J. Barth

BandTec - Faculdade de Tecnologia Bandeirantes

Outubro de 2011

Tópicos

- Introdução e Objetivo.
- Busca Sequencial.
- Busca Binária.
- Árvore Binária de Busca.

Introdução e Objetivo

Discutir e implementar diferentes estratégias para efetuar a busca de um determinado elemento em um determinado conjunto de dados.

Dado um conjunto de dados, que pode ser armazenado em um vetor v de tamanho n, que algoritmos utizar para encontrar um elemento x de forma eficiente?

Busca Sequencial

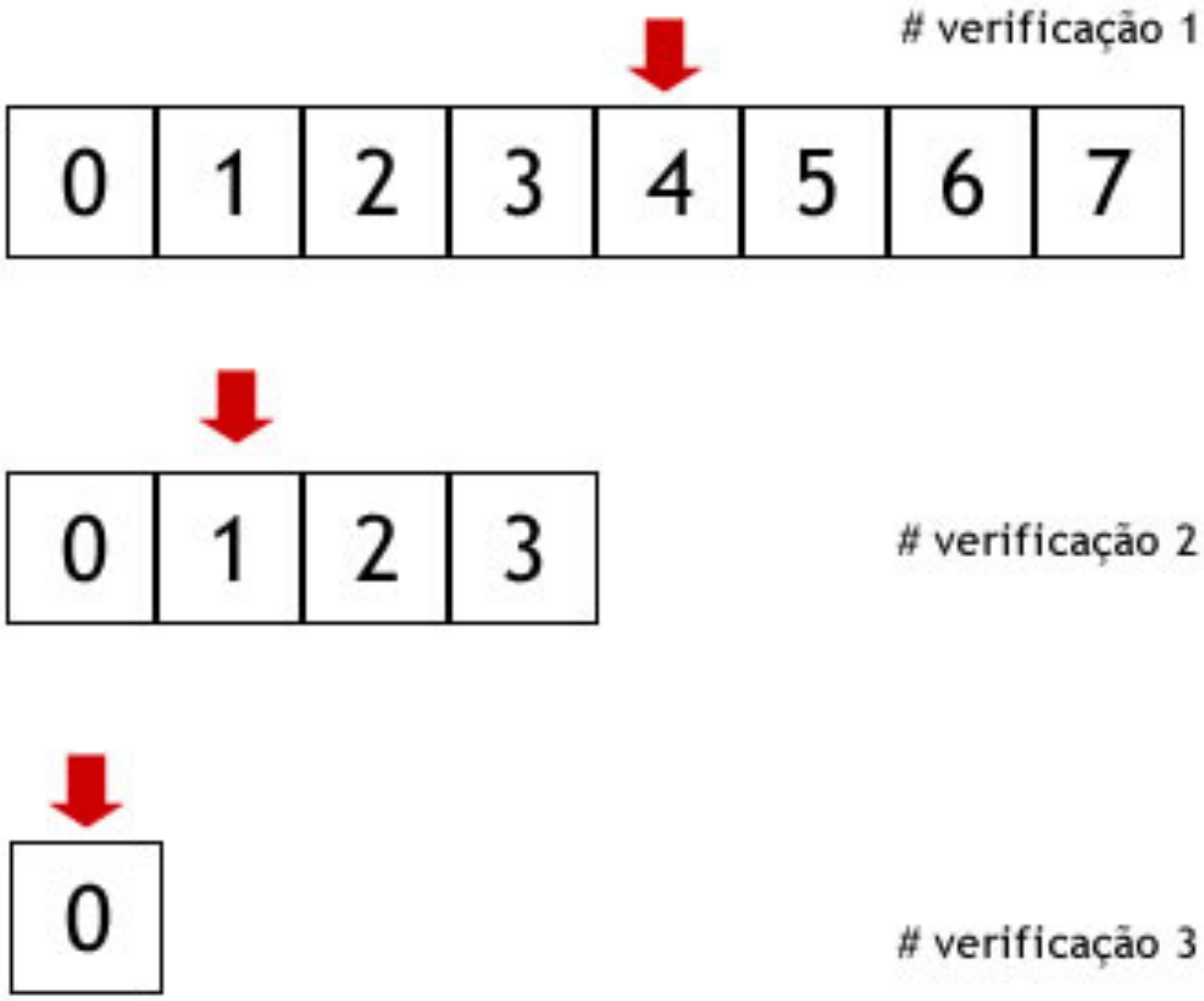
```
1  public static boolean sequencial(int x, int v[]){
2      for(int i =0; i<v.length; i++){
3          if(x==v[i])
4              return true;
5      }
6      return false;
7  }
```

Busca Sequencial

- Algoritmo extremamente simples;
- Muito ineficiente quando o número de elementos no vetor for muito grande.
- No pior caso, o algoritmo de busca sequencial precisa realizar n comparações para encontrar o elemento x.
- Portanto, o desempenho computacional desse algoritmo varia **linearmente** em relação ao tamanho do problema ($O(n)$).

Busca Binária

- Considerando que o vetor v já está ordenado, então pode-se aplicar outra estratégia para busca um elemento x.
- Ua estratégia que particiona o vetor para realizar menos comparações.


verificação 1
0
1
2
3
4
5
6
7
0
1
2
3
verificação 2
0
verificação 3

```
1  public static boolean binaria(int x, int v[]){
2      int inicio = 0; int fim = v.length-1;
3      int meio;
4      /* enquanto a parte restante for maior que zero*/
5      while(inicio <= fim){
6          meio = (inicio + fim) / 2;
7          if(x < v[meio])
8              fim = meio - 1;
9          else if(x > v[meio])
10             inicio = meio + 1;
11         else
12             return true;
13     }
14     return false;
15 }
```

Análise do Algoritmo de Busca Binária

- O desempenho deste algoritmo é muito superior ao de busca sequencial[a].
- Quantas vezes precisamos repetir o procedimento de subdivisão para concluir que o elemento não está presente no vetor?

[a] o pior caso caracteriza-se pela situação do elemento que buscamos não estar no vetor.

Implementação recursiva do Algoritmo de Busca Binária

O algoritmo de busca binária iterativo tem um desempenho superior a versão recursiva do mesmo algoritmo.

Mesmo assim, vale a pena dar uma olhada na versão recursiva.

```
1  public static boolean binariaR(int x, int v[], int n){
2      if(n <= 0)
3          return false;
4      else{
5          /* deve buscar o elemento do meio*/
6          int meio = n / 2;
7          if(x < v[meio])
8              return binariaR(x,v,meio);
9          else if(x > v[meio]){
10             return binariaR(x,copyV(v,meio),n-1-meio);
11         }else
12             return true;
13     }
14 }
```

Busca em uma Lista Encadeada

- Se tivermos os dados armazenados em uma lista encadeada, só temos a alternativa de implementar um algoritmo de **busca linear**, mesmo se os elementos estiverem ordenados.
- Portanto, a lista encadeada não é uma boa opção para estruturar nossos dados, se desejarmos realizar muitas operações de busca.

Árvore Binária de Busca

Uma árvore binária de busca é uma estrutura de dados de árvore binária onde todos os nós da sub-árvore **esquerda** possuem um valor numérico **inferior** ao nó raiz e todos os nós da sub-árvore **direita** possuem um valor **superior** ao nó raiz.

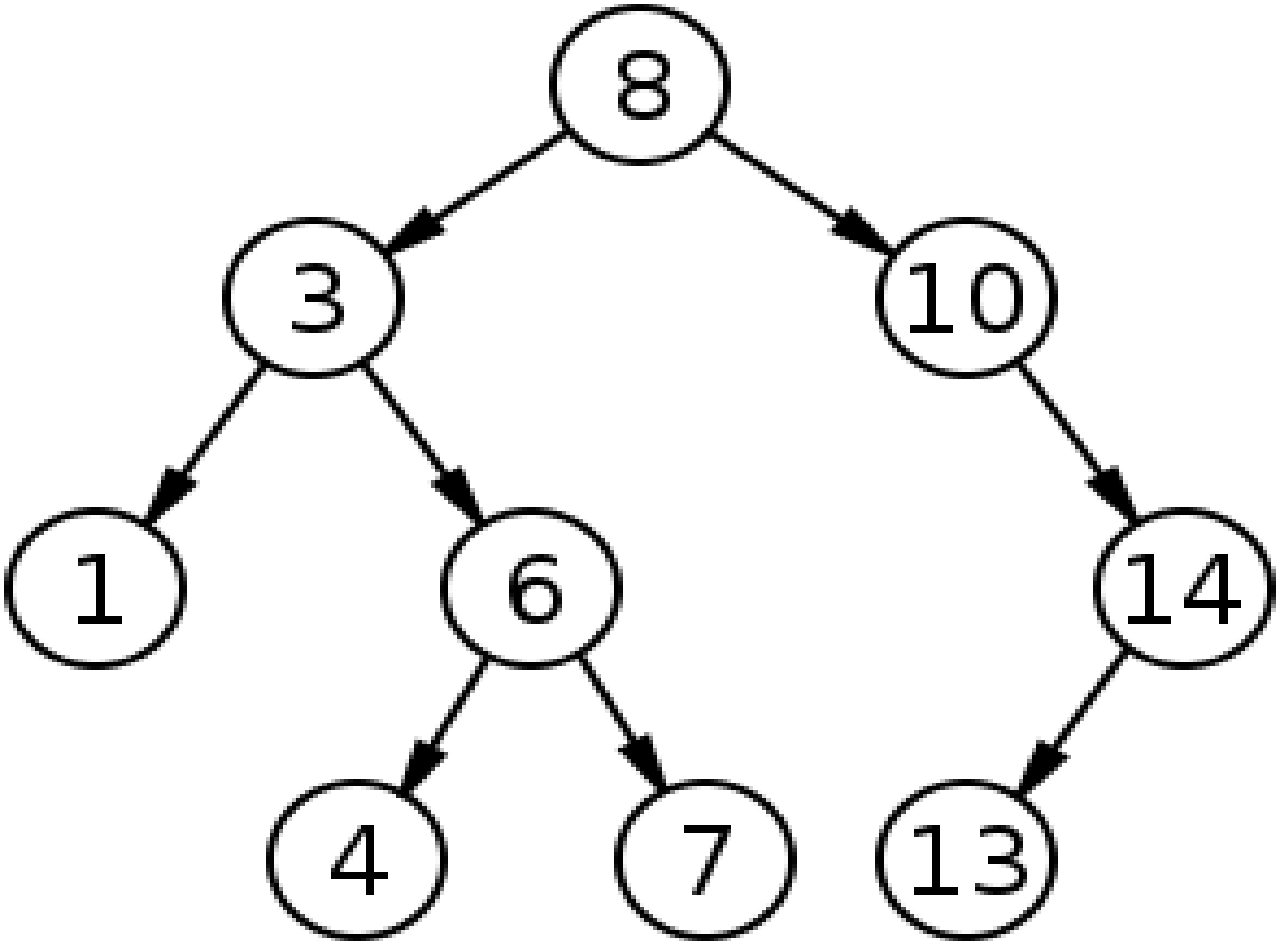


Figura 1: Exemplo de árvore binária de busca

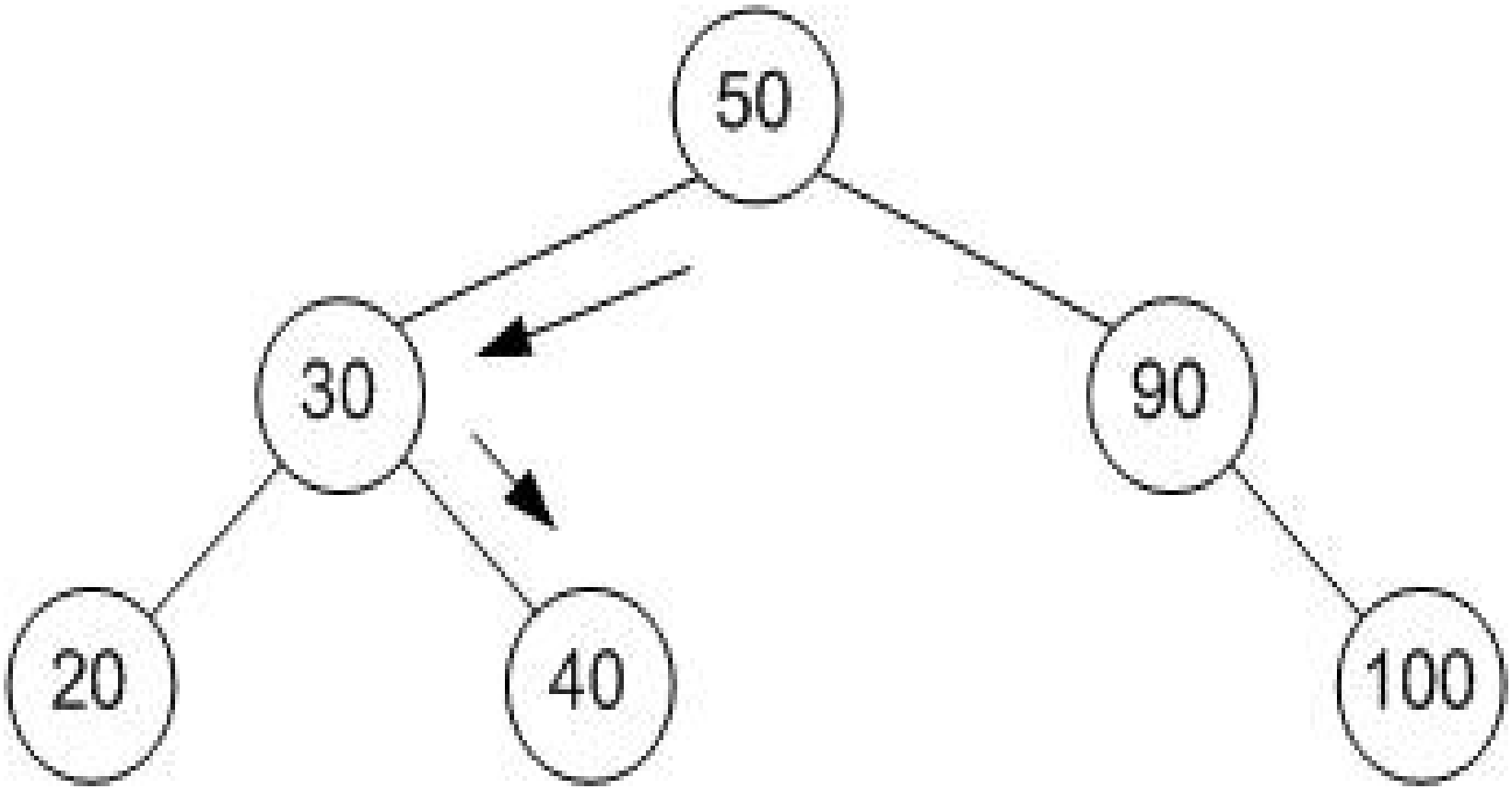
Métodos de uma Árvore Binária de Busca

- void imprimir(Nodo n);
- **Nodo inserir(Nodo n, char c);**
- **boolean procura(Nodo n, char c);**
- **Nodo retirna(Nodo n, char c);**

Imprimindo o conteúdo ordenado

```
1  private void printOrdenado(Nodo n){
2      if(n!=null){
3          printOrdenado(n.getEsq());
4          System.out.print(n.getInfo()+" - ");
5          printOrdenado(n.getDir());
6      }
7  }
```

Buscando por conteúdo na árvore



Buscando por conteúdo na árvore

```
1  private boolean procura(Nodo n, char c){
2      if(n != null){
3          if((int)n.getInfo() > (int)c)
4              return procura(n.getEsq(),c);
5          else if((int)n.getInfo() < (int)c)
6              return procura(n.getDir(),c);
7          else
8              return true;
9      }else{
10         return false;
11     }
12 }
```

Inserindo conteúdo na árvore

```
1  private Nodo inserir(Nodo n, char c){
2      if(n != null){
3          if((int)n.getInfo() > (int)c)
4              n.setEsq(inserir(n.getEsq(),c));
5          else if((int)n.getInfo() < (int)c)
6              n.setDir(inserir(n.getDir(),c));
7          return n;
8      }else{
9          n = new Nodo(); n.setInfo(c);
10         n.setDir(null); n.setEsq(null);
11         return n;
12 }}
```

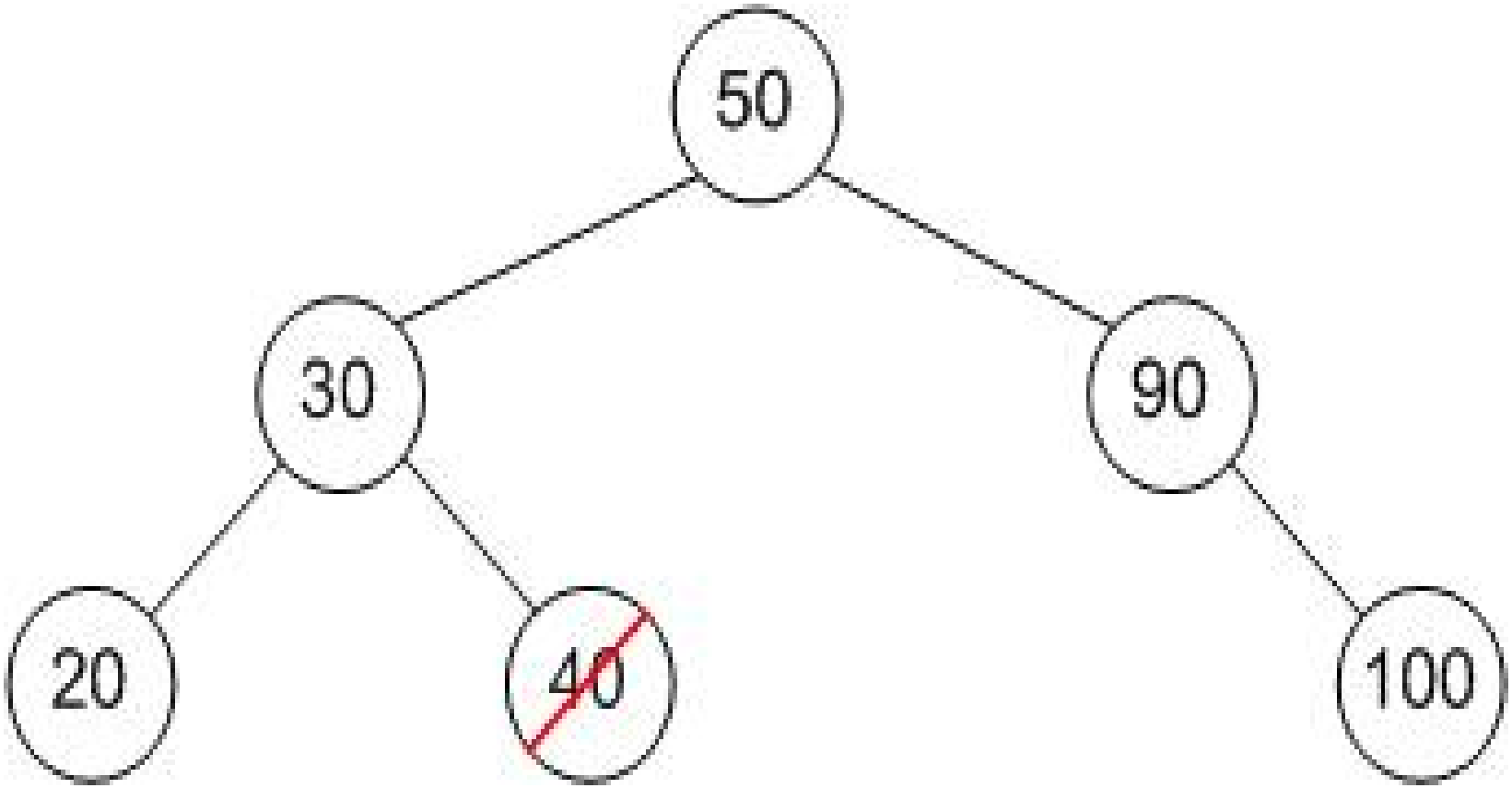
Estrutura de uma árvore binária de busca

```
1   public class ArvoreBinaria {
2       private Nodo raiz = null;
3
4       public void inserir(char c){
5           raiz = inserir(raiz,c);}
6
7       public void printOrdenado(){
8           this.printOrdenado(raiz);}
9
10      public boolean procura(char c){
11          return procura(raiz,c);}
12  ...
```

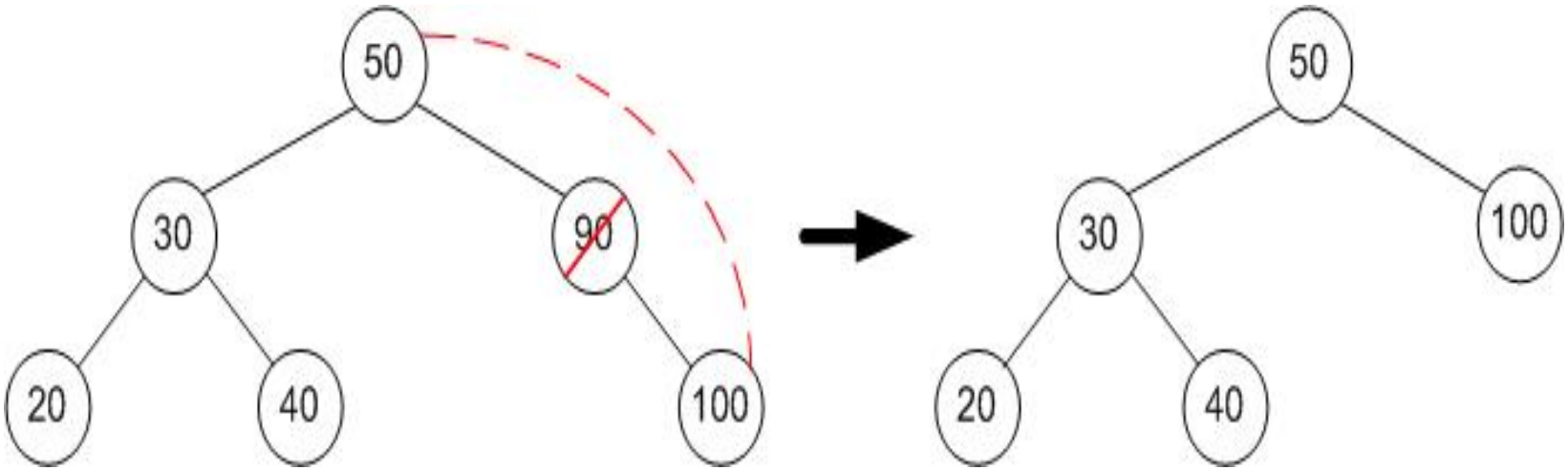
Função que retira um elemento da árvore

- Exclusão na folha.
- Exclusão de um nó com um filho.
- Exclusão de um nó com dois filhos.

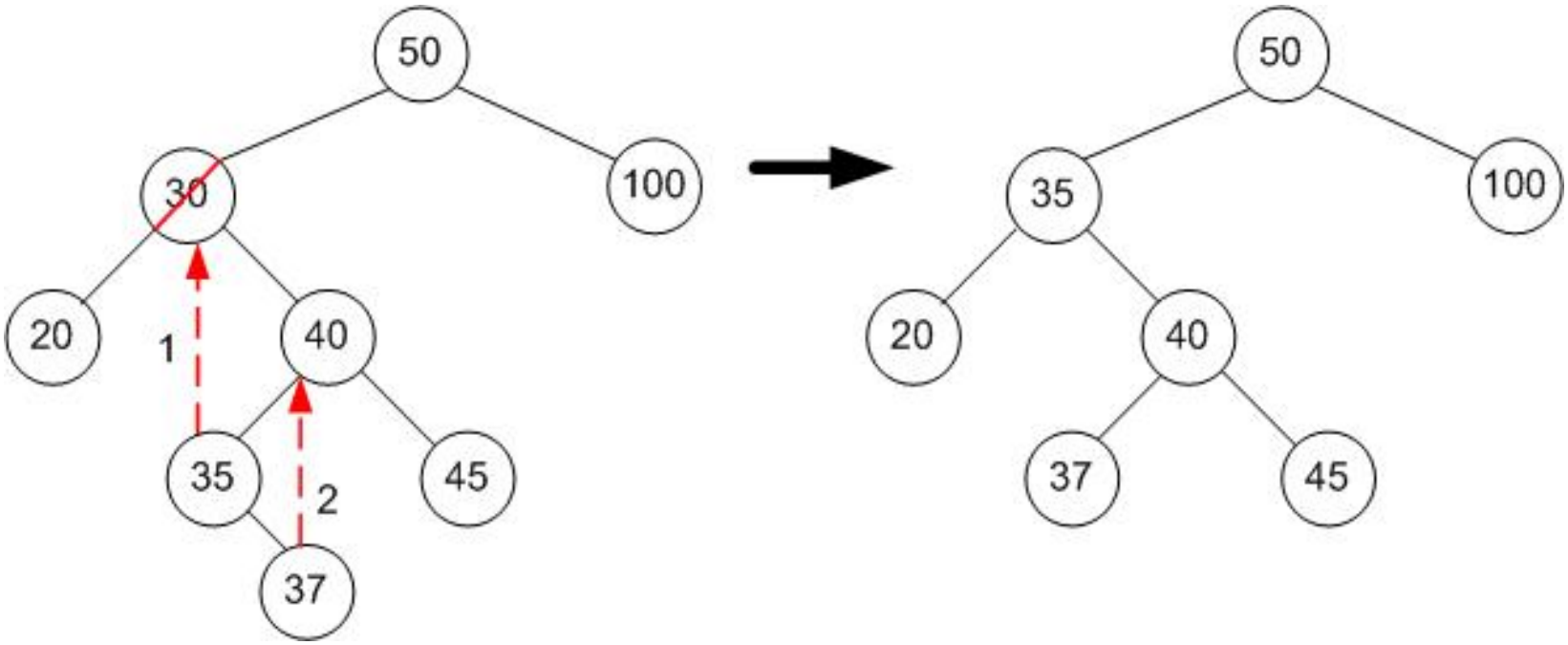
Exclusão na folha



Exclusão de um nó com um filho



Exclusão de um nó com dois filhos



Retirando um elemento da árvore

```
1   private Nodo retira(Nodo n, char c){
2       if(n==null){
3           return null;
4       }else if((int)n.getInfo() > (int)c){
5           n.setEsq(retira(n.getEsq(),c));
6       }else if((int)n.getInfo() < (int)c){
7           n.setDir(retira(n.getDir(),c));
8       }else{
9           /*achou o elemento*/
10          if(n.getEsq()==null && n.getDir()==null){
11              n = null; /*elemento sem filhos*/
12          }else if(n.getEsq() == null){
13              /*so tem filho a direita*/
14              n = n.getDir();
15          }else if(n.getDir() == null){
16              /*so tem filho a esquerda*/
17              n = n.getEsq();
18          }else{
19              /*TODO tem os dois filhos */
20          }
21      }
22      return n;
23  }
```

Retirando um elemento da árvore com dois filhos

```
1   ...
2   }else{
3         /*tem os dois filhos*/
4         Nodo temp = n.getEsq();
5         while(temp.getDir()!=null){
6             temp = temp.getDir();
7         }
8         n.setInfo(temp.getInfo());
9         temp.setInfo(c);
10        n.setEsq(retira(n.getEsq(),c));
11  }
12  ...
```

Considerações Finais

Nesta aula foram vistos os seguintes algoritmos de busca:

- **Busca sequencial ou linear**:
  - ⋆ Método que consome **muito** tempo para encontrar um elemento.
- **Busca binária**:
  - ⋆ Método que consome **pouco** tempo ao procurar por um elemento.
  - ⋆ No entanto, não pode ser utilizado em estruturas dinâmicas.

- **Árvore de Busca Binária**:
  - ⋆ Método que pode ser utilizado no armazenamento ordenado e recuperação de elementos em uma **estrutura dinâmica**;
  - ⋆ Consome **pouco** tempo para procurar por um elemento.

Material de **consulta** e **referência**

- Capítulo 17 do livro: "Introdução a Estruturas de Dados" do Waldemar Celes, Renato Cerqueira e José Lucas Rangel.
- As imagens utilizadas nestes slides foram obtidas no site http://pt.wikipedia.org/wiki/Arvore_binaria_de_busca.