UNIVERSIDADE FEDERAL
DE SANTA CATARINA

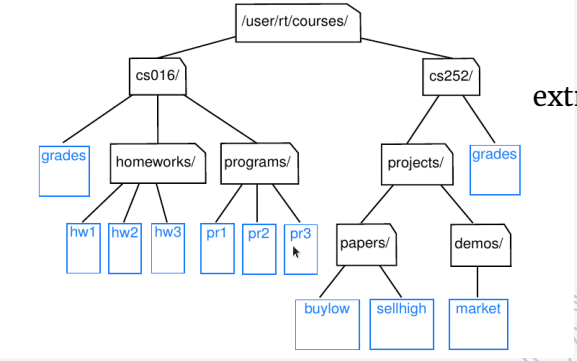
Árvores

Prof. Vinicius Ramos

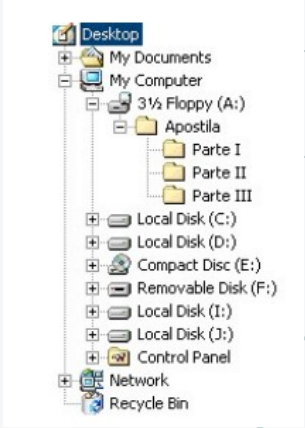
Slides Adaptados do Prof. Cristian Cechinel

Revisão

- As estruturas lineares são muito importantes na representação de dados, mas não são suficientes quando é necessário uma hierarquia entre os dados
- Por exemplo, os arquivos de um computador são armazenados dentro de uma estrutura hierárquica de diretórios



extraído de Goodrich, Tamassia & Goldwasser, 2014



Árvore de diretório (extraído de Celes, Cerqueira e Rangel, 2004)

ÁRVORES

- Listas encadeadas, filas, pilhas e deques são estruturas de dados lineares (sequenciais) e a organização dos dados deve ser feita da mesma maneira: linear
- A ÁRVORE é uma estrutura de dados bidimensional, não-linear, com propriedades especiais
- Por ser não-linear, podemos ordenar os elementos de muitas maneiras
- Este formato nos permite resolver inúmeros problemas muito mais rapidamente

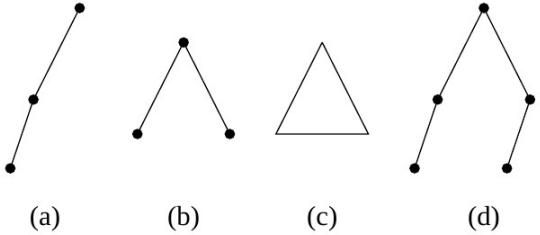
Árvores - Terminologia

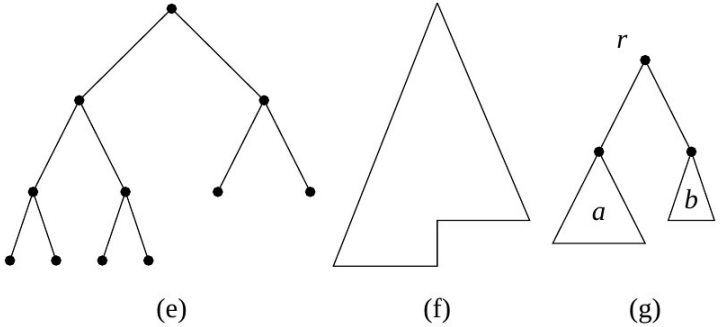
- A árvore possui um conjunto de elementos chamados de NÓS e de um conjunto de relações entre ele, chamadas de arestas
- Normalmente, os nós armazenam os dados da árvore
- Duas árvores são Disjuntas, se não existirem nós ou arestas entre elas
- Apenas um nó isolado, também é uma árvore
- As árvores são, normalmente, construídas de maneira recursiva

Árvores - Terminologia

- Existe um nó que é chamado de raiz (r)
  - Ele contem zero ou mais subárvores, cujas raízes são ligadas diretamente a r
  - Os nós raízes das subárvores de r são ditos filhos do nó pai r

Exemplos de Árvores



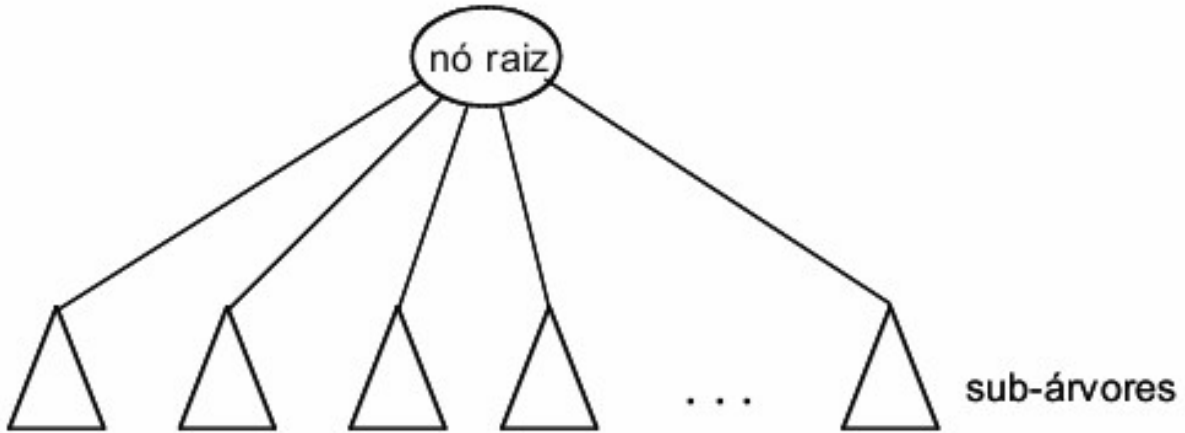


Extraído de BAILEY, D. A., 2007

(a) e (b) são árvores com três nós. (c) uma representação abstrata de árvores. (b) é uma árvore *cheia*, mas (d) não. A árvore (e) não é cheia, mas é *completa*. Árvores completas são representadas como em (f). Uma árvore abstrata (g) tem raiz r e subárvores a e b.

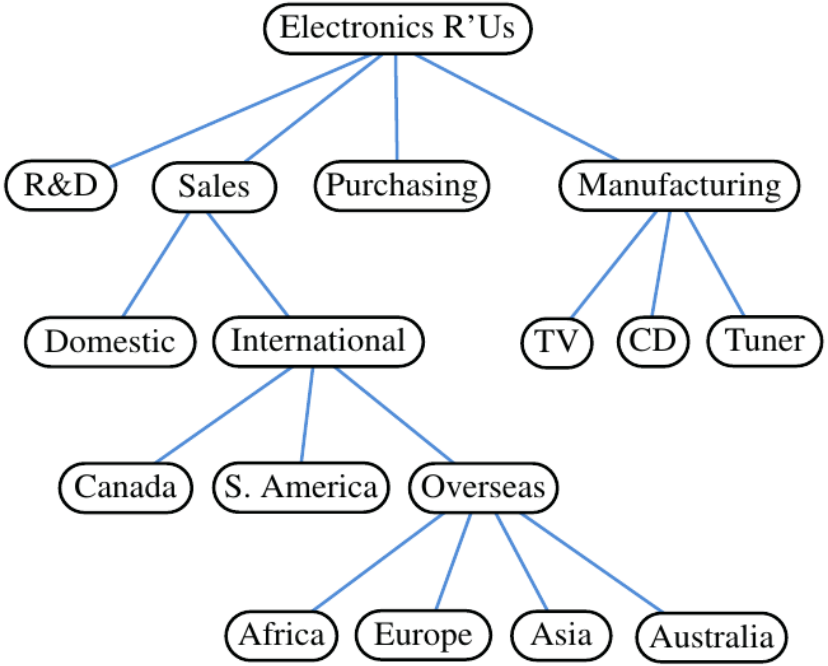
Árvores

- Nós com filhos são chamados de nós internos, e nós sem filhos são chamados de folhas ou externos



extraído de Celes, Cerqueira e Rangel, 2004

Exemplo



extraído de Goodrich, Tamassia & Goldwasser, 2014

Árvores

- A quantidade de filhos permitidos por nós e as informações armazenadas nos mesmos diferenciam os vários tipos de árvores existentes

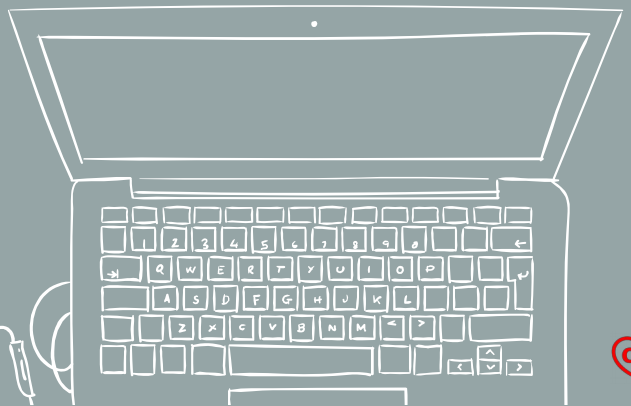
Referências

- CELES, W., R. CERQUEIRA, and JL RANGEL. Introdução a Estruturas de Dados. Rio de Janeiro, RJ, Brasil: Campus, 2004. 294 p. ISBN 85-352-1228-0.
- Sedgewick, Robert, and Kevin Wayne. Algorithms. Addison-Wesley Professional, 2011.
- Deitel, H. M., & Deitel, P. J. (2003). Java, como programar. 4ª Edição. São Paulo.
- BAILEY, D. A.. Java Structures: Data Structures for the Principied Programmer. 7ª Edição. 2007.
- GOODRICH, M. T., TAMASSIA, R. & GOLDWASSER, M. H.. Data Structures and Algorithms in Java. 6ª Edição. 2014

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Obrigado!

Prof. Vinicius Ramos
https://viniciusramos.pro.br
v.ramos@ufsc.br



UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Árvores Binárias

Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel

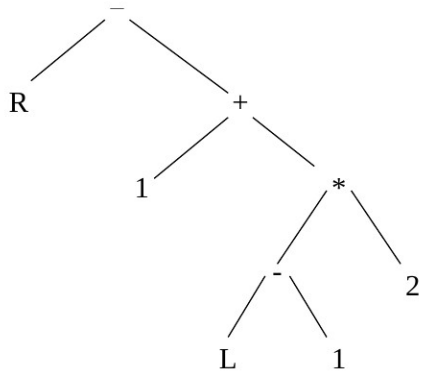
Árvores Binárias

- Em uma árvore binária, cada nó tem zero, um ou dois filhos
- De maneira recursiva, podemos definir uma árvore binária como sendo:
  - Uma árvore vazia
  - Um nó raiz tendo duas subárvores: a subárvore esquerda e a subárvore direita
    - Estas subárvores podem ou não ser vazias (*null*)

Árvores Binárias - Exemplo

- São usadas para avaliação de expressões
- Cada nó tem, no máximo, dois filhos
- Os nós folhas são operandos
- Os nós internos são operadores

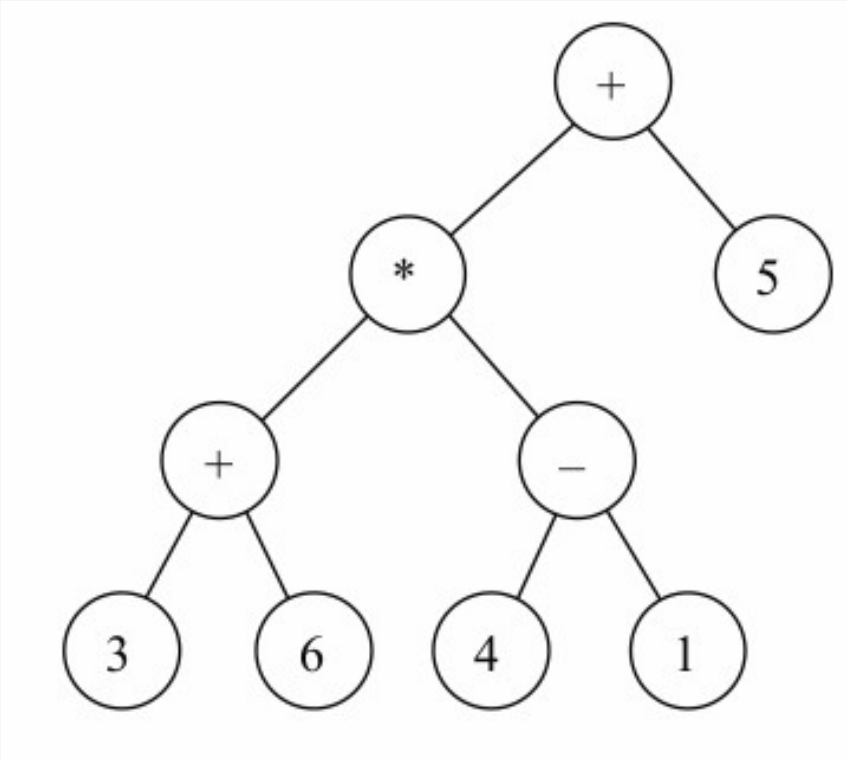
$$R = 1 + (L - 1) * 2$$



Extraído de BAILEY, D. A., 2007

Árvores Binárias - Exemplo

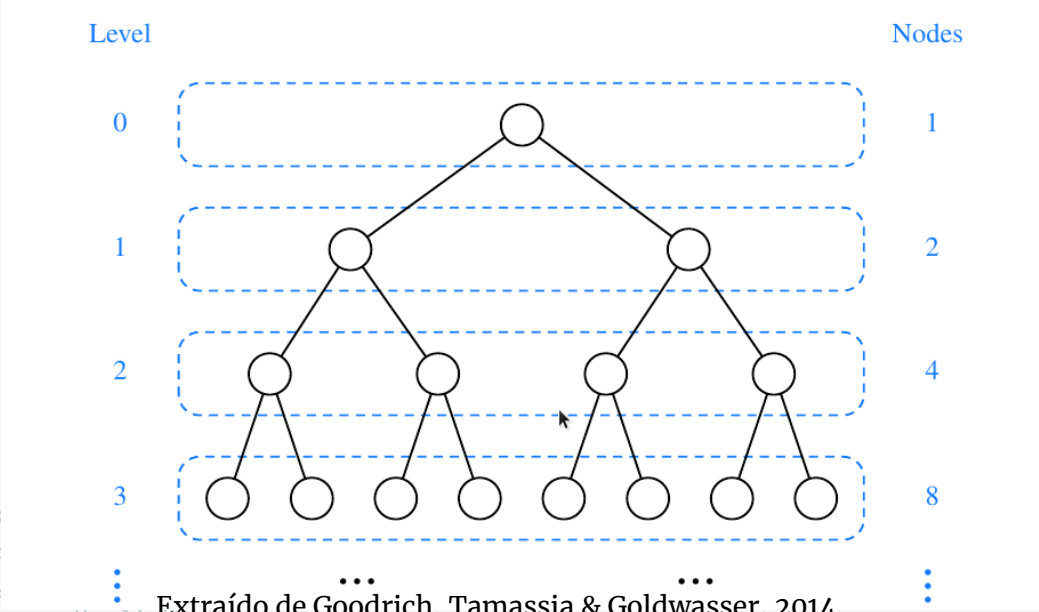
(3 + 6) * (4 – 1) + 5



Extraído de Celes, Cerqueira e Rangel, 2004

Árvores Binárias - Propriedades

- Dizemos que uma árvore tem altura/nível h
- O número de nós dado uma altura h é: $N = 2^h$



Extraído de Goodrich, Tamassia & Goldwasser, 2014

Árvores Binárias

* Implementações

Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel

Árvores Binárias

- Definimos uma Árvore a Classe Arvore

```
public class Arvore{
    private NodoArvore raiz;

    Arvore(){
        raiz = null;
    }
```

- Veja que essa definição pressupõe a criação de um Nó raiz para a árvore. Temos que criar essa classe!

Árvores Binárias

- Faremos todas as implementações no NodoArvore e poderemos chamar os seus métodos porque o Nó raiz será do tipo NodoArvore
- Como já foi comentado, precisaremos de três atributos para o Nodo: nodo esquerda, nodo direita e o valor armazenado
- Existem algumas implementações que consideram o campo pai (parent) como um atributo do Nó
  - Isso é feito para facilitar a navegação entre os nós e para voltar ao nó pai ou anterior

Árvores Binárias

- Definimos um Nó usando a Classe NodoArvore

```
class NodoArvore {
    private NodoArvore nodoEsquerda;
    private Integer valor;
    private NodoArvore nodoDireita;


    NodoArvore(){
    }

    NodoArvore(int valor, NodoArvore esquerda, NodoArvore direita){
        this.valor = valor;
        this.nodoEsquerda = esquerda;
        this.nodoDireita = direita;
    }
```

Árvores Binárias

- São 3 atributos: nodo esquerda, nodo direita e o valor a ser armazenado

```
class NodoArvore {
    private NodoArvore nodoEsquerda;
    private Integer valor;
    private NodoArvore nodoDireita;


    NodoArvore(){
    }

    NodoArvore(int valor, NodoArvore esquerda, NodoArvore direita){
        this.valor = valor;
        this.nodoEsquerda = esquerda;
        this.nodoDireita = direita;
    }
```

Árvores Binárias

- Um Construtor para criar um nodo com os atributos inicializados

```
class NodoArvore {
    private NodoArvore nodoEsquerda;
    private Integer valor;
    private NodoArvore nodoDireita;


    NodoArvore(){
    }

    NodoArvore(int valor, NodoArvore esquerda, NodoArvore direita){
        this.valor = valor;
        this.nodoEsquerda = esquerda;
        this.nodoDireita = direita;
    }
```

Árvores Binárias

- Método para verificar se a árvore está vazia

```
boolean arv_vazia(){
    return this == null;
}
```

Árvores Binárias

- Método para buscar um elemento na árvore recursivamente

```
19      boolean busca (NodoArvore nodo, int v){
20          if (nodo == null) return false;
21          else
22              if (nodo.valor == v) return true;
23              else {
24                  return busca(nodo.nodoDireita, v) ||
25                         busca(nodo.nodoEsquerda, v);
26          }
27      }
```

Árvores Binárias

- Verificar se a árvore está vazia e, nesse caso, retornar falso

```
19      boolean busca (NodoArvore nodo, int v){
20          if (nodo == null) return false;
21          else
22              if (nodo.valor == v) return true;
23              else {
24                  return busca(nodo.nodoDireita, v) ||
25                         busca(nodo.nodoEsquerda, v);
26          }
27      }
```

Árvores Binárias

- Verificar se o valor do nodo atual é igual ao elemento buscado

```
19      boolean busca (NodoArvore nodo, int v){
20          if (nodo == null) return false;
21          else
22              if (nodo.valor == v) return true;
23              else {
24                  return busca(nodo.nodoDireita, v) ||
25                         busca(nodo.nodoEsquerda, v);
26          }
27      }
```

Árvores Binárias

- Se o elemento não foi encontrado, a busca prossegue pelos nodos esquerdo e direito

```
19      boolean busca (NodoArvore nodo, int v){
20          if (nodo == null) return false;
21          else
22              if (nodo.valor == v) return true;
23              else {
24                  return busca(nodo.nodoDireita, v) ||
25                         busca(nodo.nodoEsquerda, v);
26          }
27      }
```

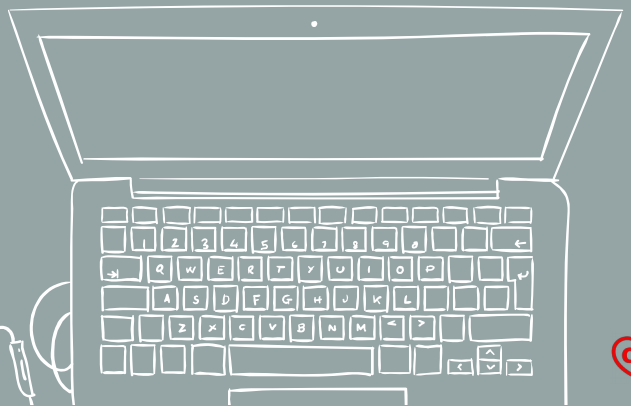
Referências

- CELES, W., R. CERQUEIRA, and JL RANGEL. Introdução a Estruturas de Dados. Rio de Janeiro, RJ, Brasil: Campus, 2004. 294 p. ISBN 85-352-1228-0.
- Sedgewick, Robert, and Kevin Wayne. Algorithms. Addison-Wesley Professional, 2011.
- Deitel, H. M., & Deitel, P. J. (2003). Java, como programar. 4ª Edição. São Paulo.
- BAILEY, D. A.. Java Structures: Data Structures for the Principied Programmer. 7ª Edição. 2007.
- GOODRICH, M. T., TAMASSIA, R. & GOLDWASSER, M. H.. Data Structures and Algorithms in Java. 6ª Edição. 2014

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Obrigado!

Prof. Vinicius Ramos
https://viniciusramos.pro.br
v.ramos@ufsc.br



Árvores Binárias

* Ordem de Percurso

Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel

Árvores Binárias

- Muitas operações em árvores binárias envolvem o percurso de todas as subárvores
- Normalmente, usam-se três possíveis ordens:
  - Pré-ordem: trata a raiz, percorre a esquerda, percorre a direita
  - Ordem simétrica: percorre a esquerda, trata a raiz, percorre a direita
  - Pós-ordem: percorre a esquerda, percorre a direita, trata a raiz

Árvores Binárias - Pré-ordem

- A visita/impressão do nó ocorre no início:

```
29      void imprimePre(NodoArvore nodo){
30          if (nodo != null){
31              System.out.print("<");
32              System.out.print(nodo.valor);
33              imprimePre(nodo.nodoEsquerda);
34              imprimePre(nodo.nodoDireita);
35              System.out.print(">");
36          }
37      }
```

Árvores Binárias – ordem Simétrica

- A visita/impressão do nó ocorre sempre após visitar toda a subárvore esquerda:

```
void imprimeOrdem(NodoArvore nodo) {
    if (nodo != null) {
        System.out.println("<");
        imprimeOrdem(nodo.nodoEsquerda);
        System.out.println(nodo.valor);
        imprimeOrdem(nodo.nodoDireita);
        System.out.println(">");
    }
}
```

Árvores Binárias - Pós-ordem

- A visita/impressão do nó ocorre sempre após visitar toda a subárvore esquerda e direita:

```
void imprimePosOrdem(NodoArvore nodo) {
    if (nodo != null) {
        System.out.println("<");
        imprimeOrdem(nodo.nodoEsquerda);
        imprimeOrdem(nodo.nodoDireita);
        System.out.println(nodo.valor);
        System.out.println(">");
    }
}
```

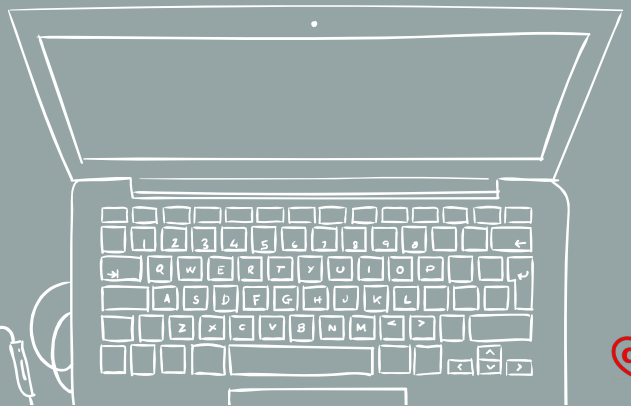
Referências

- CELES, W., R. CERQUEIRA, and JL RANGEL. Introdução a Estruturas de Dados. Rio de Janeiro, RJ, Brasil: Campus, 2004. 294 p. ISBN 85-352-1228-0.
- Sedgewick, Robert, and Kevin Wayne. Algorithms. Addison-Wesley Professional, 2011.
- Deitel, H. M., & Deitel, P. J. (2003). Java, como programar. 4ª Edição. São Paulo.
- BAILEY, D. A.. Java Structures: Data Structures for the Principied Programmer. 7ª Edição. 2007.
- GOODRICH, M. T., TAMASSIA, R. & GOLDWASSER, M. H.. Data Structures and Algorithms in Java. 6ª Edição. 2014

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Obrigado!

Prof. Vinicius Ramos
https://viniciusramos.pro.br
v.ramos@ufsc.br



UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Árvores Binárias de Busca

* Implementações

Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel

Árvores Binárias de Busca

- Essas árvores possuem UMA propriedade fundamental:
  - O valor associado à raiz é sempre MAIOR do que o valor associado a QUALQUER NÓ da subárvore esquerda e é sempre MENOR do que o valor associado a QUALQUER NÓ da subárvore direita
- Essa propriedade garante que, quando a árvore é percorrida em ORDEM (esquerda, raiz, direita), os valores são encontrados em ordem crescente
- Isso permite uma busca eficiente nas árvores

Árvores Binárias de Busca

* Inserção

Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel

Árvores Binárias de Busca

Inserir Elemento

- É importante verificar se existem elementos na árvore

```
31  NodoArvore insere(NodoArvore atual, int v){
32      if (atual == null)
33          atual = new NodoArvore(v, null, null);
34      else if (v < atual.valor)
35          atual.nodoEsquerda = insere(atual.nodoEsquerda, v);
36      else atual.nodoDireita = insere(atual.nodoDireita, v);
37      return atual;
38  }
```

Árvores Binárias de Busca

Inserir Elemento

- Lembre-se que os valores menores do que no nó atual estão na subárvore à esquerda (recursivo)

```
31  NodoArvore insere(NodoArvore atual, int v){
32      if (atual == null)
33          atual = new NodoArvore(v, null, null);
34      else if (v < atual.valor)
35          atual.nodoEsquerda = insere(atual.nodoEsquerda, v);
36      else atual.nodoDireita = insere(atual.nodoDireita, v);
37      return atual;
38  }
```

Árvores Binárias de Busca

Inserir Elemento

- Lembre-se que os valores maiores do que no nó atual estão na subárvore à direita (recursivo)

```
31  NodoArvore insere(NodoArvore atual, int v){
32      if (atual == null)
33          atual = new NodoArvore(v, null, null);
34      else if (v < atual.valor)
35          atual.nodoEsquerda = insere(atual.nodoEsquerda, v);
36      else atual.nodoDireita = insere(atual.nodoDireita, v);
37      return atual;
38  }
```

Árvores Binárias de Busca

Inserir Elemento

- Ao final, retorna-se o nó atual

```
31      NodoArvore insere(NodoArvore atual, int v){
32          if (atual == null)
33              atual = new NodoArvore(v, null, null);
34          else if (v < atual.valor)
35              atual.nodoEsquerda = insere(atual.nodoEsquerda, v);
36          else atual.nodoDireita = insere(atual.nodoDireita, v);
37          return atual;
38      }
```

Árvores Binárias de Busca

* Implementando a Classe Ávore Binária

Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel

Árvores Binárias de Busca

- A classe Arvore Binária tem um atributo raiz, do tipo NodoArvore
- Podemos fazer chamadas aos métodos implementados anteriormente por conta do nós raiz ser do tipo NodoArvore

```
public class Arvore{
    private NodoArvore raiz;

    Arvore(){
        raiz = null;
    }
```

Árvore Binária de Busca

```
3   public class ArvoreBinaria {
4       private NodoArvore raiz;
5
6       ArvoreBinaria(){
7           this.raiz = null;
8       }
9
10      boolean buscaBinaria(int v){
11          if (raiz == null) return false;
12          else return raiz.busca(raiz, v);
13      }
14
15      void insere(int v){
16          if (raiz == null){
17              raiz = new NodoArvore(v, null, null);
18          }
19          else raiz.insere(raiz, v);
20      }
21      void imprimePre(){
22          if (raiz != null) {
23              raiz.imprimePre(raiz);
24              System.out.println();
25          }
26      }
27  }
```

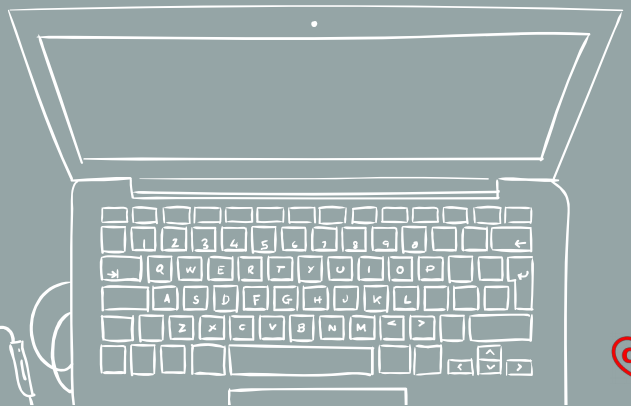
Referências

- CELES, W., R. CERQUEIRA, and JL RANGEL. Introdução a Estruturas de Dados. Rio de Janeiro, RJ, Brasil: Campus, 2004. 294 p. ISBN 85-352-1228-0.
- Sedgewick, Robert, and Kevin Wayne. Algorithms. Addison-Wesley Professional, 2011.
- Deitel, H. M., & Deitel, P. J. (2003). Java, como programar. 4ª Edição. São Paulo.
- BAILEY, D. A.. Java Structures: Data Structures for the Principied Programmer. 7ª Edição. 2007.
- GOODRICH, M. T., TAMASSIA, R. & GOLDWASSER, M. H.. Data Structures and Algorithms in Java. 6ª Edição. 2014

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Obrigado!

Prof. Vinicius Ramos
https://viniciusramos.pro.br
v.ramos@ufsc.br



Árvores Binárias

* Altura da Árvore

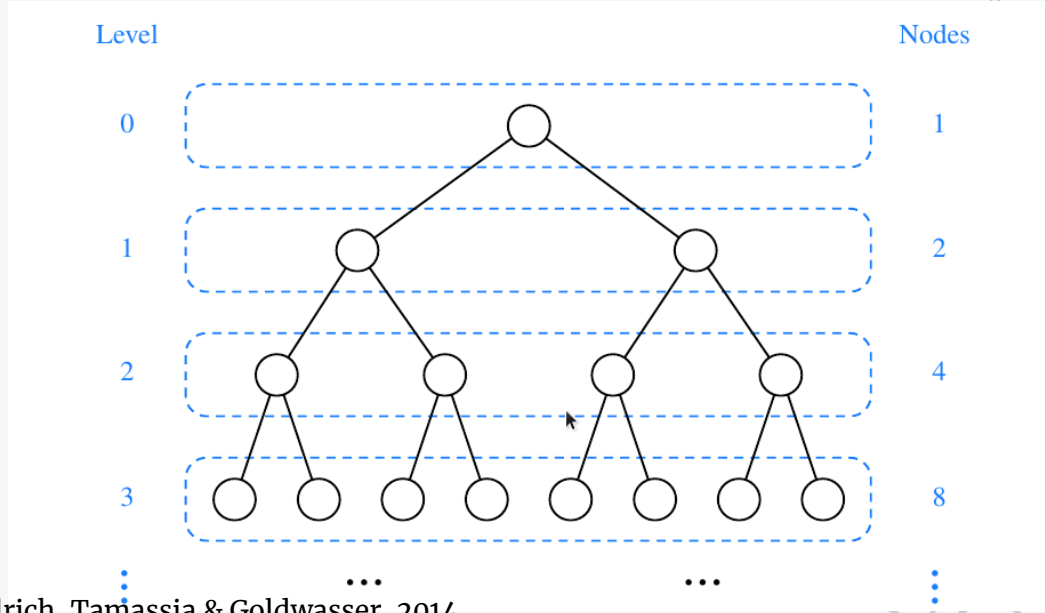
Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel

Árvores Binárias - Propriedades

- Dizemos que uma árvore tem altura/nível *h*
- O número de nós dado uma altura h é: $N = 2^h$

- A altura pode ser definida como o comprimento do caminho mais longo da raiz até uma das folhas
- Como pode ser visto, um único nó tem altura h = 0
- A altura de uma árvore vazia (*null*) é -1



Extraído de Goodrich, Tamassia & Goldwasser, 2014

Árvore Estritamente Binária, Cheia e Completa

- **Estritamente binária:** é uma árvore binária em que cada nó possui 0 ou 2 filhos
- **Binária completa:** é aquela que se v é um nó tal que alguma subárvore de v é vazia, então v se localiza ou no último (maior) ou no penúltimo nível da árvore
- **Binária cheia:** é aquela em que, se v é um nó com alguma de suas subárvores vazias, então v se localiza no último nível

Árvore Estritamente Binária, Cheia e Completa

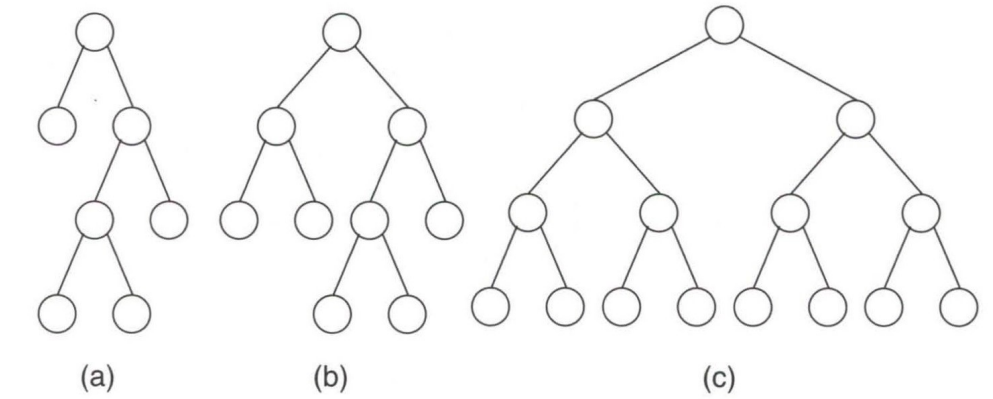


Figura 3.7: Árvores estritamente binária, binária completa e cheia

Extraído de Szwarcfiter & Markezon, 2010

Árvore Estritamente Binária, Cheia e Completa

- Isso traz propriedades importantes para as aplicações
- As árvores completas SEMPRE apresentam uma altura mínima
- Falaremos de árvores ótimas e balanceadas que usam o conceito de árvores completas e são muito eficientes em termos de acesso e manutenção

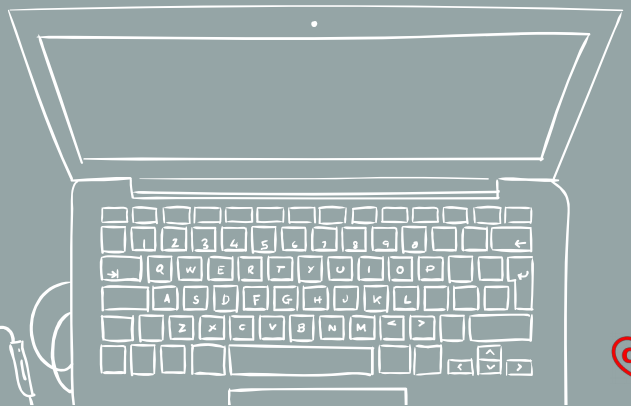
Referências

- GOODRICH, M. T., TAMASSIA, R. & GOLDWASSER, M. H.. Data Structures and Algorithms in Java. 6ª Edição. 2014
- SZWARCFITER, J. L., MARKEZON, L.. Estruturas de Dados e Seus Algoritmos. 3 Edição. 2015.

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Obrigado!

Prof. Vinicius Ramos
https://viniciusramos.pro.br
v.ramos@ufsc.br



Árvores Binárias de Busca

* Remoção

Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel

Árvores Binárias de Busca

- A remoção de um elemento é um pouco mais complexa do que a inclusão
- Existem três situações possíveis:
  - Remoção de elemento folha
  - Remoção de elemento que possui um único filho único
  - Remoção de elemento que possui dois filhos

Árvores Binárias de Busca

Remover Elemento Folha

- Situação simples
- Basta retirar o elemento da árvore e atualizar o pai, pois o filho não existe mais

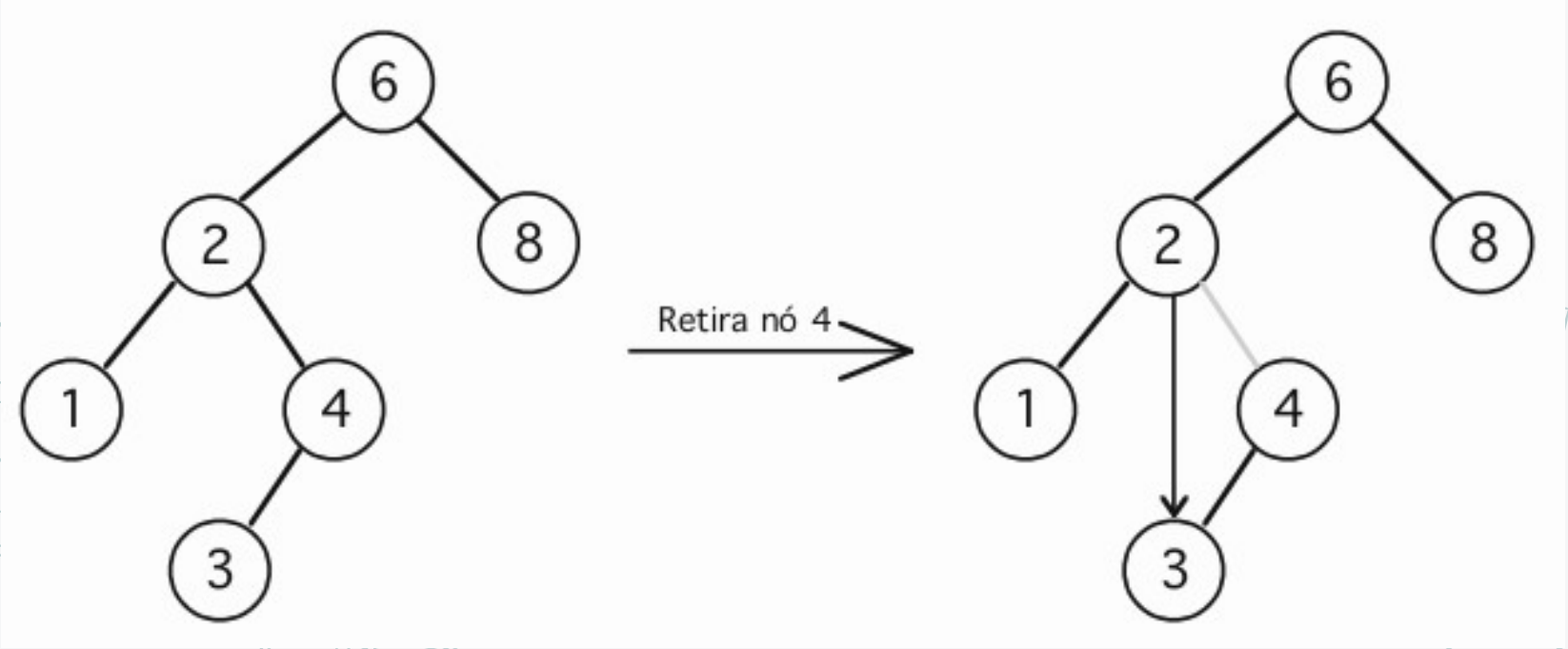
Árvores Binárias de Busca

Remover Elemento com um Único Filho

- Primeiro, é necessário acertar o ponteiro pai
  - Pular o nó atual do pai para o filho
- O único neto passa a ser o filho direto

Árvores Binárias de Busca

Remover Elemento com um Único filho



Extraído de Celes, Cerqueira e Rangel, 2004

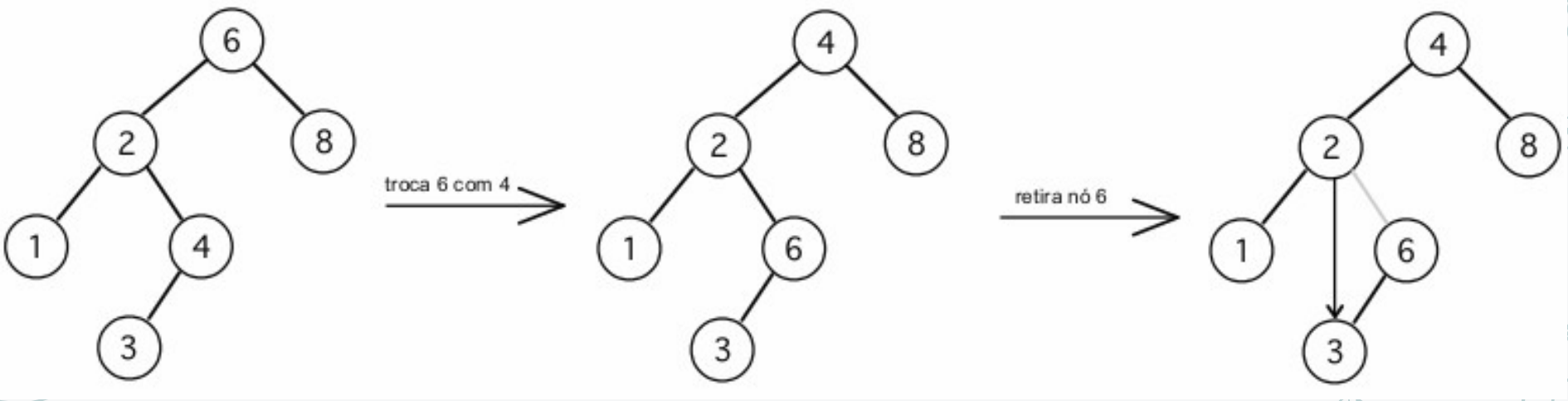
Árvores Binárias de Busca

Remover Elemento com Dois filhos

- Deve-se encontrar o elemento que precede o elemento a ser retirado
  - Ou seja, encontrar o elemento mais à direita da subárvore à esquerda (ou o contrário, elemento mais a esquerda da subárvore da direita)
- Trocar a informação do nó a ser retirado com a informação do nó encontrado no passo anterior
- Retirar o nó encontrado
  - Retirar o nó mais a direita é trivial, pois esse é um nó folha ou com um único filho (no caso, o filho da direita nunca existe)

Árvores Binárias de Busca

Remover Elemento com Dois Filhos



Extraído de Celes, Cerqueira e Rangel, 2004

Método para Remover Elemento da Árvore

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

O método recebe um nodo e um número a ser removido
Se houver elemento para remoção, o método retorna um NodoArvore

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Caso o nodo que se deseja remover esteja VAZIO, o retorno é *NULL*

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Caso contrário, a busca pelo elemento deve continuar

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Se o valor a ser removido for menor do que o valor do nodo atual, a remoção deve continuar pelo nodo da esquerda. Se for menor, ele continua pelo nodo da direita

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Caso não seja maior nem menor, significa que o valor foi encontrado no nodo atual

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Então, é necessário verificar qual a situação desse nodo e realizar a remoção

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Se ambos os filhos forem Nulos, basta retornar *Null*

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Caso apenas um dos filhos seja Nulo, deve-se retornar o nodo filho não Nulo

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Caso o nodo possua dois filhos, é preciso localizar o nodo a direita do filho da esquerda e trocar o valor desse nodo com o valor do nodo atual. Depois, pedir para o nodo da esquerda retirar o valor

```
72  NodoArvore retiraValor(NodoArvore nodo, int numero){
73      if (nodo == null) return null;
74      else if (numero < nodo.valor)
75          nodo.nodoEsquerda =
76                  retiraValor(nodo.nodoEsquerda, numero);
77      else if (numero > nodo.valor)
78          nodo.nodoDireita =
79                  retiraValor(nodo.nodoDireita, numero);
80      else {
81          if (nodo.nodoDireita == null && nodo.nodoEsquerda == null)
82              return null;
83          if (nodo.nodoDireita == null)
84              return nodo.nodoEsquerda;
85          if (nodo.nodoEsquerda == null)
86              return nodo.nodoDireita;
87
88          NodoArvore temp = nodo.nodoEsquerda;
89          while (temp.nodoDireita != null) {
90              temp = temp.nodoDireita;
91          }
92          nodo.valor = temp.valor;
93          temp.valor = numero;
94          nodo.nodoEsquerda =
95                  retiraValor(nodo.nodoEsquerda, numero);
96      }
97      return nodo;
98  }
```

Na classe árvore, o método verifica se a raiz é não nula (ou seja, existe um nó na árvore) e pede para a raiz retirar o elemento

```
void retiraNodo(int valor){
    if (raiz != null){
  raiz = raiz.retiraValor(raiz, valor);
    }
}
```

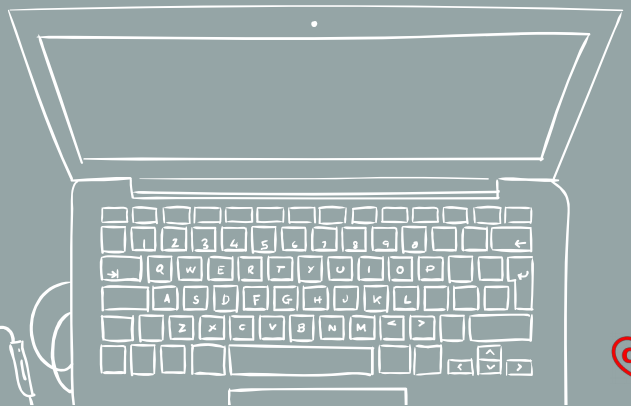
Referências

- CELES, W., R. CERQUEIRA, and JL RANGEL. Introdução a Estruturas de Dados. Rio de Janeiro, RJ, Brasil: Campus, 2004. 294 p. ISBN 85-352-1228-0.
- Sedgewick, Robert, and Kevin Wayne. Algorithms. Addison-Wesley Professional, 2011.
- Deitel, H. M., & Deitel, P. J. (2003). Java, como programar. 4ª Edição. São Paulo.

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Obrigado!

Prof. Vinicius Ramos
https://viniciusramos.pro.br
v.ramos@ufsc.br



Árvores Binárias de Busca

* Implementações

Prof. Vinicius Ramos

Slides Adaptados do Prof. Cristian Cechinel