

Dicionário Árvore Rubro Negra

Árvores rubro-negras

Julio Cezar Lossavaro

2018.0743.029-4

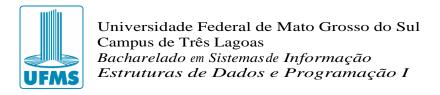
Três Lagoas

2021



Sumário

1. Estrutura das Classes		.3
1.1.	Classe Node	.3
1.1.1.	Variáveis (Atributos)	.3
1.1.2.	Métodos da Classe	.4
1.2.	Classe Tree	.4
1.1.3.	Variáveis (Atributos)	.4
1.1.4.	Métodos da Classe	.4
1.3.	Classe BinaryTreeWords	.6
1.3.1.	Variáveis	.7
1.3.2.	Funções da Classe	.7
. Funcionalidades		.7
2.1.	Desenvolvimento	.7
2.1.1.	Principais mudanças	.7
2.1.1.1.	Novos Métodos da árvore rubro-negra	.8



1. Estrutura das Classes

Nesta sessão será explicado a estrutura de cada classe definindo variáveis e métodos de forma objetiva.

1.1. Classe Node

Classe que será modelo de todos os nós da árvore, tendo como principais atributos referências aos seus filhos e seu valor, que será utilizado como parâmetro para percorrer a estrutura(arvore).

1.1.1. Variáveis (Atributos)

```
char language;
String value;
ArrayList<String> synonyms;

boolean hidden;
Color color;
Node parent;
Node left;
Node right;

public Node(String value, char language) {
    this.color = Color.RED;
    this.hidden = false;
    this.language = language;
    this.value = value;
    this.synonyms = new ArrayList<String>();
}
```

- Node left, right, parent: referências aos filhos e ancestrais do nó em si.
- **ArrayList synonyms**: atributo que armazena os sinônimos da palavra.
- Char language: atributo responsável por armazenar a linguagem da palavra, podendo assumir "e" (en) ou "p" (pt).



OBS: Esse atributo é usado para diferenciar as palavras, já que podemos ter palavras cognatas, como "banana" (inglês) e "banana" (português).

- **String value**: atributo responsável por armazenar a palavra que será associada ao nó.
- Color color: atributo utilizado para definir a cor do nó.
 Obs: Este atributo é definido por um tipo enum Color, que pode assumir os tipos RED(vermelho) e BLACK(preto), inicialmente o atributo é iniciado como RED.
- **Boolean hidden:** atributo que será utilizado para realiza a remoção, assumindo valor true no caso da remoção do nó.

1.1.2. Métodos da Classe

changeColor():

inverte a cor do nó, ou seja, caso preto o transforma em vermelho e caso vermelho o transforma em preto.

1.2. Classe Tree

Esta classe é o modelo da árvore, ela possui um nó que armazena a raiz e métodos responsáveis por retornar, inserir, remover, balancear e atualizar valores.

1.1.3. Variáveis (Atributos)

Node root: Atributo que recebe um nó referenciado a raiz da árvore;

1.1.4. Métodos da Classe

• recursiveAdd(Node, String, char, String):

Método que percorre recursivamente a estrutura com base no valor lexicográfico da palavra, além de verificar se a palavra já existe na estrutura, ao



encontrar um valor nulo cria-se um novo nó e adiciona um sinônimo a sua lista caso necessário, e por fim chama a função updateBalance(node).

Caso 1 (Nó nulo): Adiciona um novo nó e encadeia como sinônimo a próxima palavra.

Caso 2 (Nó encontrado): atribui ao atributo hidden false, já que uma palavra removida pode ser inserida novamente, em seguida verifica se a palavra passada por parâmetro está na lista de sinônimos do nó encontrado e em seguida a encadeia na lista.

Caso 3 (Palavra cognata): Em caso de palavra cognata, ou seja, que sejam iguais porém com linguagem diferente, a recursão é chamada a direita.

• updateBalance(Node):

Este método será chamado para verificar se, após uma inserção os nós subsequentes continuam balanceados e caso não, balanceá-los.

Caso 1 (pai preto): Neste caso não é feito nada, já que quando o pai é preto, seu filho está balanceado.

Caso 2 (pai e filho vermelhos): Verifica-se a cor do tio e aplica uma operação de balanceamento.

Caso 1.1(tio preto): Neste caso aplica-se uma rotação, com base na direção do nó, além de alterar as cores dos nós onde necessário.

Caso 1.2(tio vermelho): Neste caso é chamado o método changeColor() para o tio, pai e caso não seja a raiz, para o seu vô.

rotateRight(Node):

Este método realiza uma rotação a direita em um nó. Executa as devidas trocas de posicionamento entre os nós referenciados, atualiza seu parente e por fim devolve o nó balanceado.

rotateLeft(Node):

Este método realiza uma rotação a esquerda em um nó. Executa as devidas trocas de posicionamento entre os nós referenciados, atualiza seu parente e por fim devolve o nó balanceado.

findWord(String):



Realiza uma busca recursiva na estrutura utilizando uma palavra como parâmetro e por fim, caso a variável hidden não seja verdadeira retorna nó.

• Find(String):

Chama a função findWord, se a palavra for encontrada, lista seus sinônimos, caso não, retorna "hein?".

• inOrder(Node, char):

Função que percorre a árvore em ordem de maneira recursiva, percorrendo primeiro a esquerda e em seguida a direita.

Caso 1: Caso o char (linguagem) informado seja igual ao do nó e o atributo hidden seja falso, retorna o valor do nó junto aos seus sinônimos.

• inOrderBetwen(Node, char, String[]):

Método que percorre a árvore em ordem de maneira recursiva, percorrendo primeiro a esquerda e em seguida a direita.

Caso 1: Caso o char (linguagem) informado seja igual ao do nó, esteja entre os valores de char[0] e char[1] e o atributo hidden seja falso, retorna o valor do nó junto aos seus sinônimos.

• Remove(Node, String):

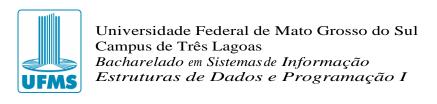
Método que percorre a estrutura em busca de um nó, caso o encontre seta a variável hidden como verdadeira, simbolizando para o algoritmo que o valor do nó será ignorado.

• RemoveSyns(String, String):

Este método recebe duas palavras e as remove dos sinônimos de ambas as palavras correspondentes, caso não haja mais sinônimos em sua lista, chama-se o método remove() para o seu nó correspondente.

1.3. Classe BinaryTreeWords

Essa classe é contém o método principal do programa, ele é responsável por receber as informações passadas pelo sistema e armazenar em um array de strings, e em seguida os passando por parâmetro para os métodos da árvore.



1.3.1. Variáveis

1.3.2. Funções da Classe

main(): método principal que cria um objeto Scanner para ler o fluxo de entrada passado pelo usuário e um objeto Tree, que será a árvore propriamente dita. Um laço é criado com a finalidade de ler todos os parâmetros passados e invocar os métodos da classe Tree ou finalizar o programa.

2. Funcionalidades

Nesta sessão será descrito as funcionalidades do projeto de maneira teórica, explicando de maneira abrangente, esclarecendo decisões e também destacando as principais mudanças do projeto.

2.1. Desenvolvimento

Durante o desenvolvimento não houve grandes dificuldades, uma vez que ao entender o conceito de árvores rubro-negras e de como elas funcionam, realizar as operações de balanceamento se tornou pura lógica de programação.

2.1.1. Principais mudanças

• Classe Node: Foi implementada a classe Node três novos atributos, sendo eles: Node parent: Este atributo foi implementado com o objetivo de percorrer os ancestrais de um dado nó, tendo como principal função verificar a cor de um dado tio, para em seguida utilizar como parâmetro de balanceamento.

Boolean hidden: Este atributo será utilizado na remoção, para simbolizar se um dado nó está "escondido" na estrutura.

Color color: Este atributo é definido por um tipo enum, podendo assumir RED ou Black, assim definindo a cor de um dado nó.



- recursiveAdd(): O método de inserção continua o mesmo, porém agora leva em consideração que, caso precise inserir um nó que já foi previamente removido, ele atribui ao atributo hidden false, demonstrando ao algoritmo que agora é uma palavra a ser mostrada.
- inOrder() e inOrderBetwen: Agora verifica se o nó está com o atributo hidden como false, antes de mostrá-lo.
- rotateLeft() e rotateRight(): Agora leva em consideração os parentes de um dado nó, fazendo as trocas necessárias.
- **findWord():** Agora verifica se o nó está com o atributo hidden como false, antes de devolver um dado nó.

2.1.1.1. Novos Métodos da árvore rubro-negra

- **updateBalance():** Método que verifica se os filhos de um dado nó estão balanceados, verificando e aplicando as devidas operações para manter as propriedades da árvore.
 - Foi implementado com o objetivo de balancear a arvore após uma inserção.
- Remove(): Para realizar a remoção foi escolhido o método preguiçoso, já que dado o contexto do projeto não há necessidade de realizar muitas operações de remoção, pois ao adicionar uma palavra dificilmente haverá necessidade de retirar a mesma da estrutura.
 - Foi implementado o atributo hidden na classe Node, para "esconder" uma palavra na estrutura e assim possibilitando realizar a remoção preguiçosa de maneira simples.
 - O método basicamente percorre a estrutura em busca de uma referenciada palavra e ao encontrá-la atribui ao seu atributo hidden como true, assim sinalizando para o algoritmo que aquele nó foi "removido".