



Universidade Federal de Mato Grosso do Sul
Campus de Três Lagoas
Bacharelado em Sistemas de Informação
Estruturas de Dados e Programação I

Árvore Binária de Busca

Árvores Binárias de Busca

Julio Cezar Lossavaro

2018.0743.029-4

Três Lagoas

2021



Sumário

1. Estrutura das Classes	2
1.1. Classe Node	2
1.1.1. Variáveis (Atributos)	3
1.1.2. Funções da Classe	3
1.2. Classe Tree	4
1.1.3. Variáveis (Atributos)	4
1.1.4. Funções da Classe	4
1.3. Classe BinaryTreeWords	8
1.3.1. Variáveis	8
1.3.2. Funções da Classe	8

1. Estrutura das Classes

1.1. Classe Node

Classe que será modelo de todos os nós da árvore, tendo como principais atributos nós referenciando seu pai e seus filhos.

1.1.1. Variáveis (Atributos)

```
char language;  
String value;  
ArrayList<String> synonyms;  
Node left;  
Node right;  
Node parent;  
  
public Node(String value, char language) {  
    this.language = language;  
    this.value = value;  
    this.synonyms = new ArrayList<String>();  
}
```

- **Node left, right:** referências aos filhos do nó em si.
- **Node parent:** referência ao pai do nó, este atributo foi criado com o intuito de facilitar a remoção do nó referenciado, no caso de precisarmos subir na árvore em busca de um sucessor.
- **ArrayList synonyms:** atributo que armazena os sinônimos da palavra.
- **Char language:** atributo responsável por armazenar a linguagem da palavra, podendo assumir “en” ou “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.

1.1.2. Funções da Classe

- **hasOnlyLeftChild():**

Função que retorna verdadeiro caso o nó tenha apenas um filho a direita.

- **hasOnlyRightChild():**

Função que retorna verdadeiro caso o nó tenha apenas um filho a esquerda.

- **isLeaf():**

Função que retorna verdadeiro caso o nó não tenha filhos.

- **compareTo(String):**

Essa função tem grande importância no projeto, já que é por ela que organizamos toda a árvore, ela compara o valor lexicográfico da primeira letra do atributo value e retorna 0, caso seja igual, 1, caso maior e -1, caso seja menor.

1.2. Classe Tree

Esta classe é o modelo da árvore, ela possui um nó que armazena a raiz e funções responsáveis por retornar, inserir, remover e atualizar informações na mesma.

1.1.3. Variáveis (Atributos)

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

1.1.4. Funções da Classe

- **isEmpty():**

Função que verificar se a raiz está vazia.

- **recursiveAdd(Node, String, char, String):**

Função que percorre recursivamente a estrutura com base no valor lexicográfico do primeiro caractere da String, indo a esquerda caso seja menor e a direita caso maior, ao encontrar um valor nulo cria-se um novo nó e encadeia ao mesmo pai, sinônimo, String(palavra) e um char(linguagem).

- **recursiveAdd(String[]):**

Função que recebe um array de Strings com os argumentos informados pelo usuário, faz uma busca e em seguida trata os seguintes casos:

Caso 1 (Raiz Vazia): Adiciona um nó a raiz, encadeia como sinônimo a segunda palavra e em seguida chama o método RecursiveAdd(Node, String, char, String) com os parâmetros da segunda palavra.

Caso 2 (Nó Não Encontrado): Invoca o método RecursiveAdd(Node, String, char, String) adicionando a palavra a estrutura.

Caso 3(Nó Encontrado): verifica se a palavra está na lista de sinônimos do nó encontrado na busca e em seguida a encadeia na lista.

- **min(Node):**

Função que percorre recursivamente todos os filhos de um nó a esquerda e assim consequentemente retornando o seu menor valor.

Caso 1: retorna nulo caso a raiz esteja vazia.

- **max(Node):**

Função que percorre recursivamente todos os filhos de um nó a direita e por fim retornando o último elemento, consequentemente retornando o seu menor valor.

Caso 1: retorna nulo caso a raiz esteja vazia.

- **Sucessor(Node):**

Função que procura um nó que possa substituir um determinado nó.

Caso 1: Inicialmente percorre a direita a procura do menor valor (utilizando a função min) e retorna o nó encontrado.

Caso 2: Percorre os nós acima utilizando como referência parent (referência do pai do nó) a procura de um valor maior que o do nó em si e o devolve.

- **recursiveSearch(Node, String, char):**

Percorre árvore com os parâmetros de busca de uma árvore binária de busca, ou seja, a esquerda caso o valor seja menor a direita, caso maior e por fim, retorna o valor do nó, se encontrado.

Caso 1: o valor do nó coincide com a String e o char informado, retornando o nó em si;

Caso 2: o valor do nó é maior que o da String informada, é feita a chamada recursiva da função passando o seu nó a esquerda como parâmetro.

Caso 3: o valor do nó é menor que o da String informada, é feita a chamada recursiva da função passando o seu nó a direita como parâmetro.

Caso 4: Ao encontrar um valor nulo, retorna um nó nulo, sinalizando que o nó não foi encontrado.

- **search(String, char):**

Função utilizada para chamar a função recursiveSearch, passando a raiz da árvore e as informações do nó a ser encontrado.

OBS: Essa função foi criada com o intuito de especializar ao máximo o método que percorre a árvore, para que possamos chama-la em outros métodos/funções.

- **findWord(String):**

Invoca a função search(String, Char) e procura o nó em ambas as linguagens possíveis, retornando um nó caso a palavra exista na estrutura.

OBS: Essa função foi criada por conta da implementação do “Buscar”, que busca uma palavra na estrutura a partir de um String única, vale ressaltar que podemos ter palavras iguais, tomando como exemplo a palavra “banana” ou “animal”, portanto ele pode retornar uma palavra tanto em inglês tanto português.

- **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 retorna o valor do nó junto aos seus sinônimos.

- **inOrder(char):**

método que chama o método `inOrder(Node, char)` e passa a raiz como parâmetro de nó e o `char(linguagem)` como parâmetro.

- **`inOrderBetwen(Node, char, char[])`:**

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 e esteja entre os valores de `char[0]` e `char[1]`, retorna o valor do nó junto aos seus sinônimos.

- **`inOrderBetwen(String[])`:**

Método que cria um array de chars alimentado o mesmo com os argumentos informados por linha de comando, em seguida chama o método `inOrderBetwen(Node, char, char[])`.

- **`remove(Node)`**

Método que remove um nó e o troca por um sucessor, caso necessário.

Caso 1(Nó folha): Invoca o método `isLeaf()` do nó, verificando se é uma folha, em seguida verifica se é uma raiz, caso sim ela apontará para nulo, simbolizando que a árvore esta vazia, em seguida, caso o nó não seja a raiz, subimos a árvore e fazemos o seu pai apontar para nulo no lado em que a ser removido está.

Caso 2(Apenas um Filho): Utilizamos os métodos `hasOnlyLeftChild()` e `hasOnlyRightChild()` para saber em qual lado o filho está, em seguida temos que ligar o filho ao pai do nó que será removido.

Caso 3(Dois Filhos): Invoca a função `sucessor(Node)` que irá retornar um nó para realizar a substituição, em seguida trocamos o valor do nó para os valores do nó sucessor, e por último chamamos recursivamente `remove(Node)` passando o nó sucessor como parâmetro para a remoção.

- **`Menu(String[])`:**



Método responsável por chamar as demais funções/métodos de acordo com o que o usuário informar, seguindo os seguintes casos.

Caso 1('i', insere): Chama o método RecursiveAdd(String[]).

Caso 2('l', listar): Chama o método inOrder(char) ou inOrderBetwen(String[]) de acordo com o número de argumentos passados.

Caso 3('b', buscar): Chama o método find(String[]).

Caso 4('r', remove): Chama o método remove(String[]);

1.3. Classe BinaryTreeWords

Essa classe 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 a função menu(String[]).

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, em seguida invoca o método menu(String[]) dentro de um laço de repetição, e assim permanece, até que o usuário envie a instrução de parada