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

Dicionário Árvore Digital

Árvores digitais

Julio Cezar Lossavaro
2018.0743.029-4

Três Lagoas
2021

Sumário

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, além dos valores que serão utilizados para identificar uma palavra.

Para implementar esta classe foi optado por utilizar um HashMap, como estrutura para armazenar os filhos de um dado nó, já que podemos utilizar um dado caractere para identificar um nó e assim diminuindo a complexidade de busca e de uso de memória, já que por se tratar de um mapa, sua complexidade para todas as operações é equivalente a O(1).

1.1.1. Variáveis (Atributos)

```
public class Node {

    boolean isFinal;
    char lang;
    String word;
    HashMap<Character, Node> letras;
    ArrayList<String> synonyms;


    public Node() {
        this.isFinal = false;
        this.letras = new HashMap<>();
    }
```

- **HashMap<Character, Node> letras:** este atributo armazena um nó associado a um caractere, que será utilizado para percorrer a arvore, dado o seu caractere correspondente.
- **ArrayList synonyms**: atributo que armazena os sinônimos da palavra, este atributo existira apenas para nós onde isFinal(final da palavra) for verdadeiro.

- **Char language**: atributo responsável por armazenar a linguagem da palavra, podendo assumir “e” (en) ou “p” (pt).
- **String word**: atributo responsável por armazenar a palavra que será associada ao nó, este atributo existira apenas para nós onde isFinal(final da palavra) for verdadeiro..
- **Boolean isFinal:** atributo que será utilizado para identificar quando um nó corresponde ao final de alguma palavra.

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 valores.

1.1.2. Variáveis (Atributos)

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

1.1.3. Métodos da Classe

- **recursiveAdd(Node, char[], char, String, int):**
  Método que percorre recursivamente a estrutura com base em uma cadeia de caracteres, e para cada caractere, o adiciona, caso não exista ou atribui o valor isFinal ao caractere final da cadeia como verdadeiro, caso seja prefixo de alguma outra palavra já existente, além de encadear um sinônimo a mesma, caso necessário.

  Caso 1 (Cadeia de caracteres existente):

  Caso a cadeia já exista na estrutura, atribui os valores correspondentes ao nó do ultimo caractere, em seguida adiciona seu sinônimo, caso necessário.

Caso 2 (Cadeia de caracteres não existente):

Cria um novo nó com seus respectivos valores e em seguida o adiciona ao mapa hash, passando seu caractere como chave e o nó como valor.

- **findWord(Node, char[]):**

  Método que ao receber uma cadeia de caracteres, retorna o valor do seu respectivo nó final, caso exista.

- **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: Retorna o valor do nó, caso isFinal seja verdadeiro e a linguagem seja a mesma passada por parâmetro.

- **Remove(Node, char[], int):**

  Método que percorre a estrutura utilizando um array de caracteres como parâmetros, caso consiga consumir todos os caracteres realiza os seguintes casos:

  Caso 1(Sem filhos): neste caso, apenas é atribuído nulo ao nó.

  Caso 2(tem filhos): neste caso, apenas retornamos o nó, já que a palavra é prefixo de outra.

- **RemoveSyns(String, String):**
  Este método recebe duas palavras e as remove dos seus respectivos sinônimos, caso não haja mais sinônimos em sua lista, chama-se o método remove() para o seu nó correspondente, e assim as removendo, caso necessário.

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. Métodos 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.

- **Definição classe Node**: Ao definir a estrutura foi levado em consideração o uso de um HashMap para mapear as letras ao dado nó, já que ao utilizar esta estrutura, a complexidade de todas as operações envolvendo o mesmo, seria O(1), além de facilitar a busca e remoção, já que podemos iterar pelo mapa hash utilizando os caracteres.
- **RecursiveAdd**: fazer uma função recursiva utilizando um iterador e um array de caracteres foi um desafio, mas ao estudar a fundo a biblioteca HashMap do próprio java, a função replace foi de grande ajuda, já que facilitava muito substituir o nó retornado pela função(recursivamente).
- **Remove**: Implementar o remove foi relativamente fácil, uma vez que bastava verificar se o mapa hash estava vazio(folha) e utilizar a mesma lógica já empregada em RecursiveAdd para percorrer a estrutura.