Tries (Árvores Digitais)

Estruturas de Dados Básicas II Instituto Metrópole Digital

2019.2

1 Introdução

Neste trabalhos, você deverá implementar uma estrutura de dados chamada **trie**, também conhecida como árvore digital.

2 Descrição

Uma árvore digital é uma árvore genérica onde cada nó pode ter quaisquer quantidade de filhos. Ela é utilizada para armazenar palavras que podem ser pesquisadas a partir de um prefixo comum. Esta propriedade faz dela uma estrutura eficiente para completar palavras. De fato, algumas das *IDEs* e editores que vocês utilizam, usam uma **trie** internamente no seu *autocomplete*. Outro uso comum também é em gerenciadores de contatos pessoais.

2.1 Funcionamento

Supondo que queremos armazenar as seguintes palavras: `a`, `ama`, `amar`, `ame`, `amei`, `ameixa`, `oi`, `oito`, `eu`, `ele`, `ela`, `eles`, `elas`. Ao invés de armazenar as palavras diretamente, uma *trie* armazena o caminho da raiz até a palavra dividindo-a em seus caracteres. A Figura 1 mostra uma representação visual de uma *trie* com as palavras mencionadas anteriormente.

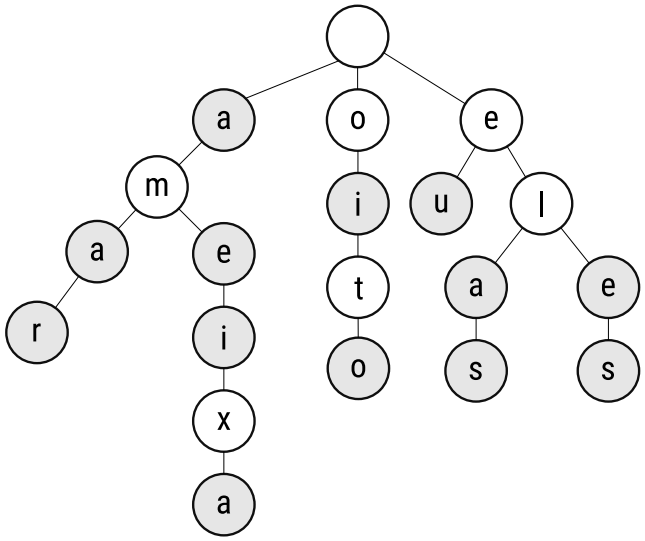

Figura 1: Representação de uma *trie*. Os nós cinzas representam palavras.

2.2 Inserção

O método de inserção em uma *trie* se dá da seguinte maneira:

1. Seja p uma palavra, n uma posição de caractere em p e $p[n]$ o caractere na posição n da palavra p.
2. Para cada caractere em palavra p, verificamos se ele já existe na árvore.
    - Se não existir criamos o nó com a chave $p[n]$.
3. Temos que criar a árvore de tal forma que um caractere $p[n]$ seja pai de $p[n+1]$.
4. Marcamos que o último caractere da palavra p como formando uma palavra. Este passo é importante porque uma palavra pode ser prefixo de uma palavra maior. Por exemplo, a palavra `ame` é prefixo das palavras `amei` e `ameixa`.

2.3 Busca

Para sabermos se uma palavra existe na árvore, utilizamos cada letra de uma palavra como o caminho a partir da raiz. Se conseguirmos chegar até o último caractere de uma palavra na árvore, isto quer dizer que ela existe dentro da nossa estrutura.

2.4 Autocompletar

A *trie* é uma estrutura boa para pesquisarmos elementos a partir de prefixos de sua chave. Por exemplo ao pesquisar pelo prefixo `am`, podemos encontrar facilmente as palavras: `ama`, `amar`, `ame`, `amei` e `ameixa`.

- Primeiro vamos até o nó que contém o prefixo
- A partir dele, você deve percorrer os nós e ir armazenando todos que estão marcados como sendo palavra em uma lista

3 Remoção

A remoção de um elemento em uma árvore digital é bem *trievial* e deve ser elaborado por você.

4 Detalhes de Implementação

Para a criação da árvore, você deve utilizar um nó com a seguinte definição:

```
class TrieNode {
    boolean isWord;
    ???? children;

    //opcional -- deve ser preenchido apenas quando isWord é
    ↪  true
    String text;
}
```

A estrutura utilizada para armazenar os filhos — representada por `????` — deve ser definida por você. A sua *trie* deve possuir as seguintes funcionalidades:

- Um método para inserir uma palavra
- Um método para remover uma palavra
- Um método para checar se uma palavra existe
- Um método para retornar sugestões de palavras a partir de um prefixo. Este método deve ter duas sobrecargas:

  - Uma onde eu passo só prefixo e ele me retorna todas as palavras com este prefixo
  - Uma onde eu passo o prefixo e a quantidade máxima de sugestões e ele retorna no máximo uma lista com esta quantidade de elementos

Também será necessário criar um programa de linha de comando com a seguinte interface:

```
java -jar programa.jar palavras prefixo [quantidade]
```

Onde:

- `programa.jar` é o nome do seu programa.
- `palavras` é um arquivo de texto contendo várias palavras que devem ser adicionadas na árvore.
- `prefixo` é uma string que deve ser utilizada na busca.
- `quantidade` é a quantidade máxima de elementos que deve ser retornado a partir do prefixo (este parâmetro é opcional).

A saída deve ser as palavras que começam com o prefixo informado em ordem crescente a partir do tamanho. Exemplo:

```
java -jar trie.jar palavras.txt am 4
ama
ame
amar
amei
```

Lembrando que o parâmetro do tamanho é opcional e se não for informado deve exibir todas as palavras com o prefixo:

```
java -jar trie.jar palavras.txt am
ama
ame
amar
amei
ameixa
```

5 Pontuação

Este trabalho possui uma pontuação de 70 pontos e pode ser feito **individualmente**. A pontuação está distribuída da seguinte maneira:

- Implementação da inserção — 10 pontos
- Implementação da busca — 10 pontos
- Implementação da remoção — 10 pontos
- Implementação do autocompletar — 20 pontos
- Implementação do programa — 20 pontos

5.1 Bonificações e Penalidades

Além da pontuação normal, este trabalho pode ter seu total alterado nas seguintes condições:

- Criação dos testes unitários — até 10 pontos
- Criar uma interface gráfica em Java e implementar uma funcionalidade de autocompletar a partir do seu código. A interface deve ter um componente de texto e ao digitar deve aparecer uma caixa de sugestões com no máximo 10 palavras — até 20 pontos

6 Colaboração e Plágio

Ajudar o colega de classe é legal, mas copiar seu trabalho, não. Por isso, cópias de trabalhos **não** serão toleradas, resultando em nota **zero** para **todos** os envolvidos. O mesmo vale para códigos copiados da internet.

7 Entrega

O trabalho deve ser entregue em um único arquivo compactado contendo:

- O código-fonte do seu programa zipado ou um link para um repositório git.

O arquivo deve ser enviado **apenas** pelo *SIGAA* através da opção *Tarefas* até a data divulgada no sistema.