# Sistema de busca e indexação de arquivos

**Ailson F. dos Santos[1], Larissa G. Melo de Moura[2] e Valmir C. S. Junior[3]**

Instituto Metrópole Digital – Universidade Federal do Rio Grande do Norte (UFRN)
Av. Senador Salgado Filho, 3000 – 59.078-970 – Natal – RN – Brasil

{ailsonforte[1],larissagilliane[2]}@hotmail.com

valmir.correa@ufrn.edu.br[3]

***Abstract.*** *This article deals with a work developed in the discipline of Programming Language 2, taught by Professors Carlos Eduardo da Silva and Rodrigo Teixeira Ramos at the Federal University of Rio Grande do Norte (UFRN), in the second semester of 2017. developed with exploratory research and characteristics educational, to implement a search system for indexing in a text base.*

***Resumo.*** *Este artigo é referente a um trabalho desenvolvido na disciplina de Linguagem de Programação 2, ministrada pelos Professores Carlos Eduardo da Silva e Rodrigo Teixeira Ramos na Universidade Federal do Rio Grande do Norte (UFRN), no segundo semestre do ano de 2017. O trabalho foi desenvolvido com cunho exploratório e educativo, para implementar um sistema de busca por indexação em uma base de textos.*

## 1. Introdução

O problema abordado neste projeto diz respeito a realização de buscas por conteúdo em uma base de textos, de forma eficiente. Para isso, foi idealizado um sistema em linguagem Java para implementar uma solução computacional capaz de realizar a busca textual em arquivos de extensão .txt.

O sistema deverá ser capaz de receber um arquivo de texto indexa-lo no sistema, fazer o tratamento adequado das palavras presentes e armazenar essas palavras em uma estrutura de dados, do tipo Trie. Permitindo assim, uma busca rápida da palavra desejada.

## 2. Funcionalidades

Para o protótipo foram definidas as funcionalidades de inserir arquivo, remover arquivo, atualizar arquivo e listar todos os arquivos contidos na base de dados do sistema tais quais permitem o usuário realizar as operações principais para um simples sistema de busca indexada.

Além de escolher quais arquivos serão indexados, também possível armazenar uma *blacklist*, que neste caso, conterão palavras que não poderão ser pesquisadas, seja por conferir cunho inapropriado, ou apenas para filtrar pesquisas indevidas.

As funcionalidades de busca, restringem-se a dois tipos, a primeira denominada <OR> tem como resultado a ocorrência de uma ou mais palavras pesquisadas presentes em quaisquer arquivos na base de dados, a segunda caracterizada como <AND> trará resultados caso as palavras estejam contidas apenas no mesmo arquivo, ou seja, caso o arquivo contenha uma e a outra palavra buscada.

Para a utilização da interface, as funcionalidades de manipulação de arquivos e adição

da *blacklist* podem ser utilizadas no menu superior, como visto na imagem a seguir:

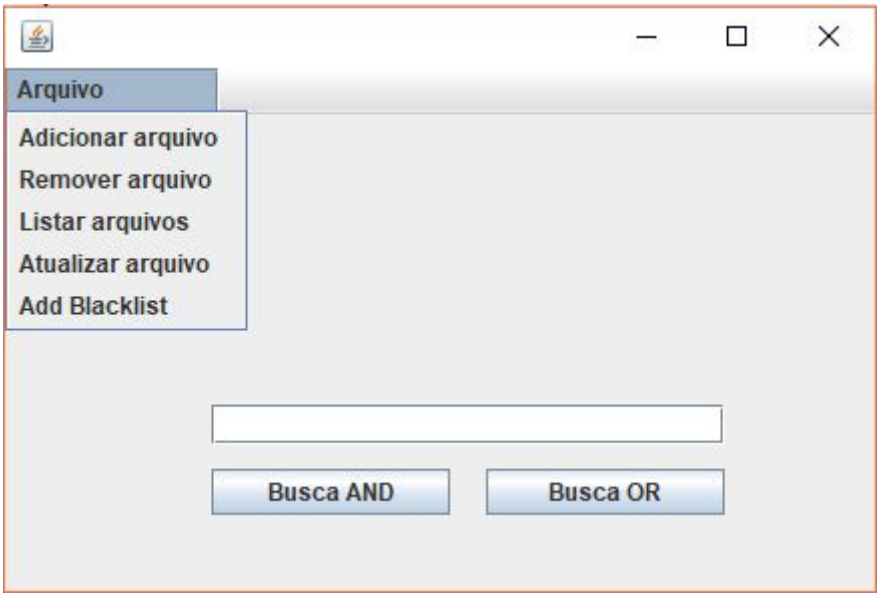


**Figura 1: Visão do menu do sistema.**

Caso o usuário deseje inserir um arquivo no sistema ele poderá, após escolher a opção "Adicionar arquivo", selecionar qualquer arquivo de texto em seu computador e indexa-lo no sistema.

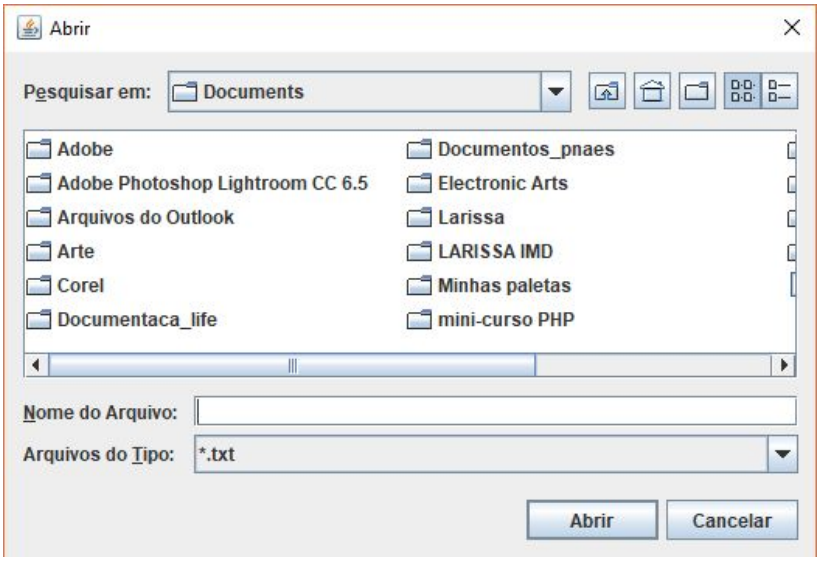


**Figura 2: Escolher arquivo**

Para remover algum arquivo presente na base de dados, é possível digitar o nome do

arquivo correspondente e este será removido do sistema. Caso o arquivo não esteja contido na base de dados o usuário será avisado que a escolha do dado foi incorreta.

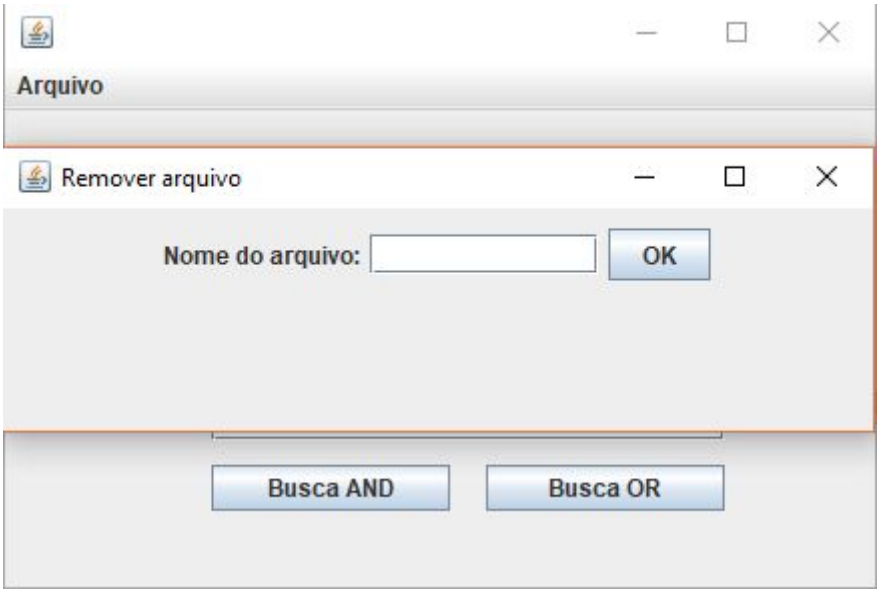


**Figura 3: Remover arquivo**

Quando o usuário desejar atualizar algum arquivo no sistema, este poderá escolher o arquivo a ser atualizado em seu computador e caso o dado não esteja presente no sistema o usuário será notificado que algum engano pode ter ocorrido.

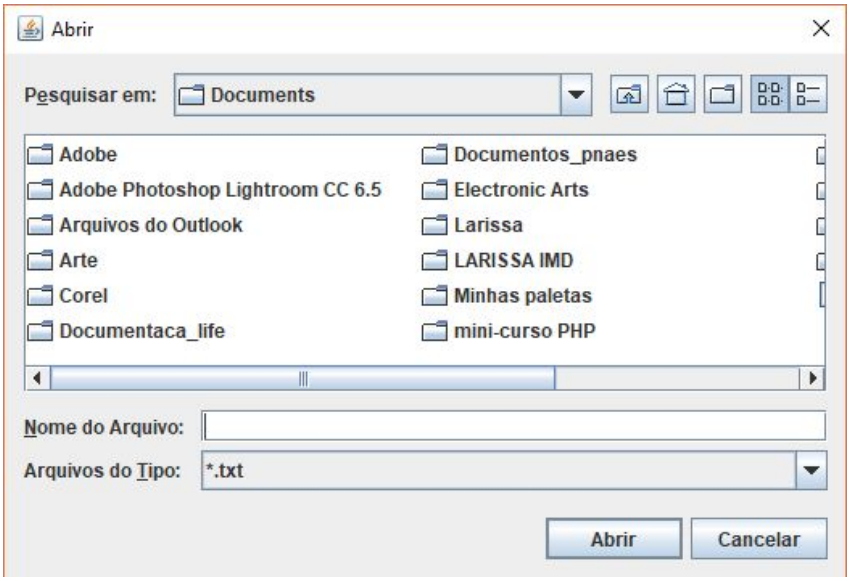


**Figura 4: Escolher arquivo para atualizar**

Para a listagem de arquivo, basta a opção “Listar arquivos” ser selecionada e o sistema criará uma nova janela listando os arquivos presentes no sistema.

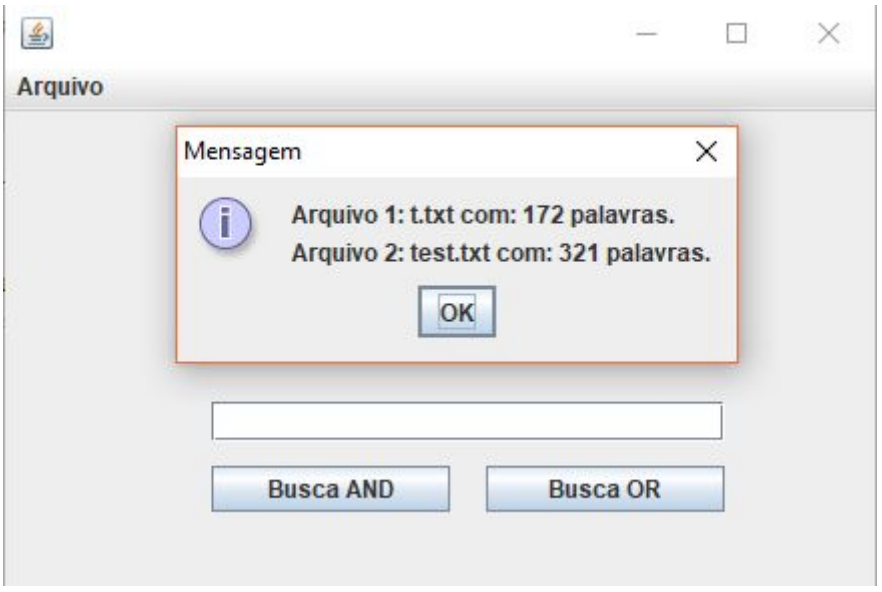


**Figura 5: Listagem de arquivos**

Quando se desejar realizar a busca de uma ou mais palavras, basta o usuário digitar o dado desejado e selecionar o tipo de busca que preferir, logo uma janela será exibida com as informações textuais referentes aos resultados obtidos.

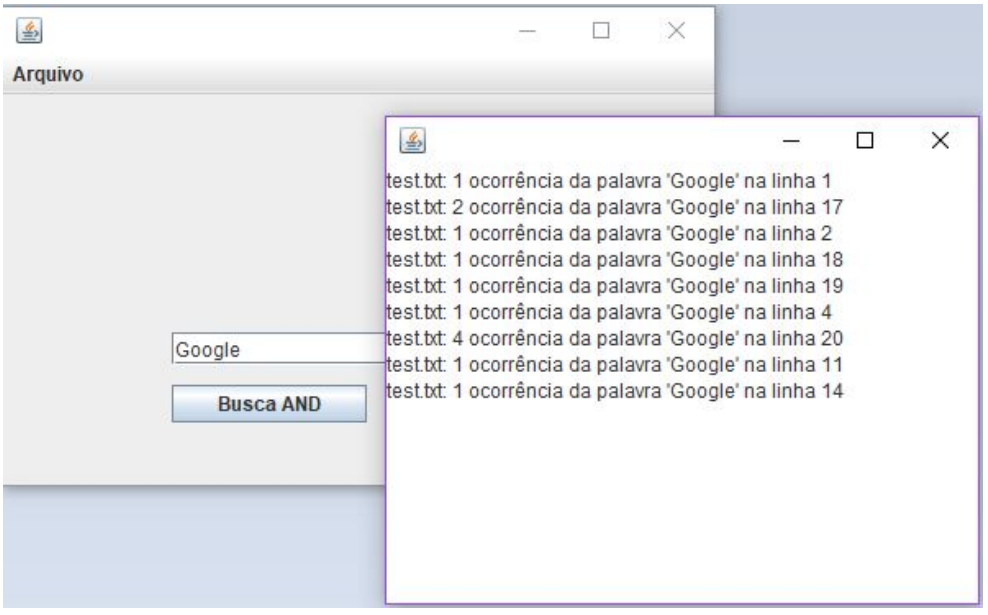


**Figura 6: Resultados de Busca <AND> sendo exibidos**

## 3. Solução escolhida

Durante o desenvolvimento do projeto ocorreram variâncias entre as decisões acerca da solução, entretanto foi possível visualizar com mais clareza, pelos membros envolvidos, o funcionamento do projeto através da escolha das classes: *Reader* - responsável pela leitura de arquivos .txt -, *Writer* - responsável pela escrita em arquivos .txt - , *Trie* - estrutura de árvore que contém as palavras armazenadas -, *TrieNode* - representa nos "nós" da árvore onde cada nó representa um caractere da palavra (letra) -, *Parser* - faz o processamento dos arquivos e armazena as palavras contidas nele na árvore (*Trie*), é caracterizada por ser a principal classe de controle do sistema-, *Indexer* - responsável pela indexação de arquivos no sistema e pela busca de palavras na base de dados - e a classe *Window* - que representa a classe de fronteira com o usuário contendo a interface da aplicação.

É possível observar a relação das classes através do diagrama de classes a seguir:

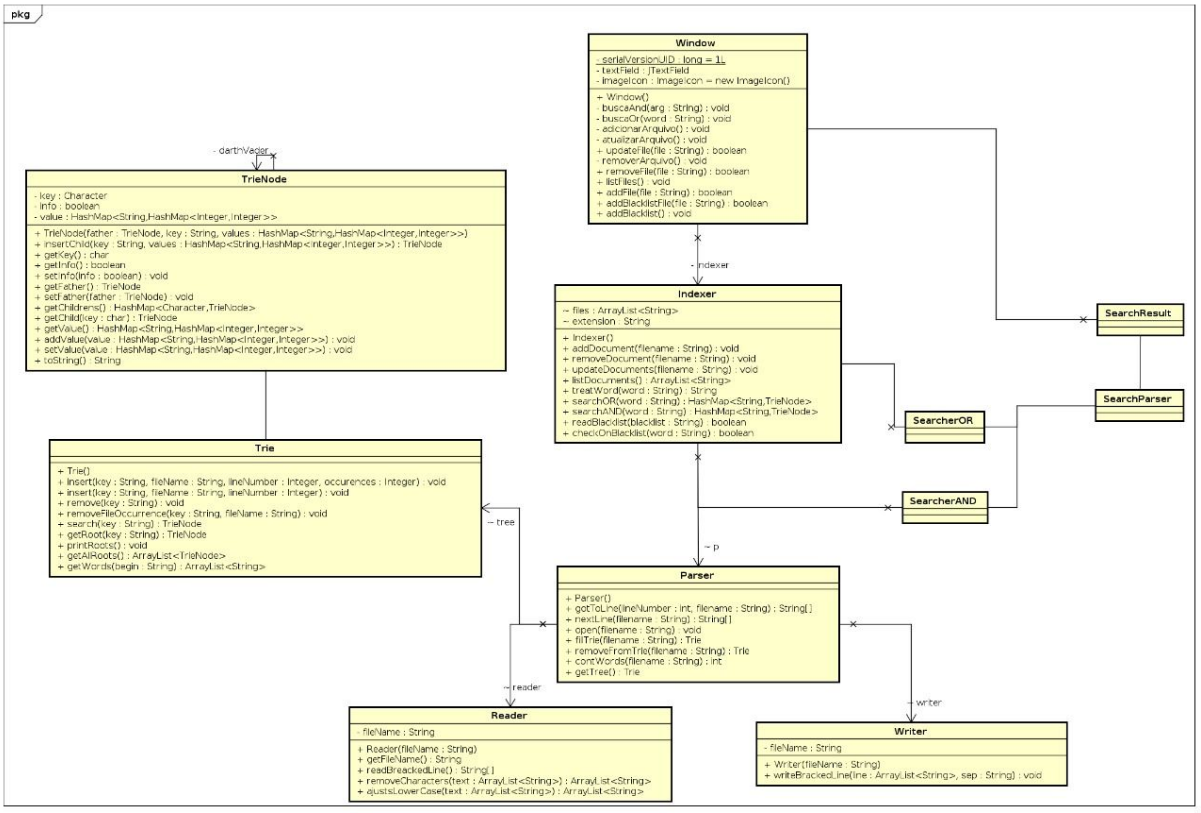


**Figura 7: Diagrama de classe do projeto**

Observando a figura 7 acima, representando o diagrama de classes do sistema, o funcionamento se inicia na classe *Main* que começa seu o funcionamento invocando um objeto da classe *Window*, que por sua vez receberá as ações do usuário e suas respectivas escolhas de dados. Ao inserir um arquivo no sistema, este será enviado para a classe *Index* que iniciará o processo de indexação utilizando um objeto *Parser.* A classe *Parser,* principal classe do sistema, fará a leitura desse arquivo recebido através de objetos do tipo *Reader* e armazenará as palavras retornadas numa estrutura de dados *Trie,* está que interage com os

objetos *TrieNode* armazenando neles os "nós" de cada palavra. Durante o processo de busca o objeto *Indexer* utiliza o *Parser* para percorrer os "nós" da *Trie* e assim retorna os resultados da palavra buscada.

Na etapa de implementação, as principais decisões foram a do uso de *HashMap* nas classes *Indexer, Trie* e *TrieNode* para otimizar a busca nas estruturas de dados. Essa Classe possui uma busca indexada por chaves com complexidade de acesso (get e put) constante. Isso facilitou a implementação das classes *Trie* e *TrieNode* na construção das sub-árvores(chave e nó filho) e da indexação dos valores(nome do arquivo, linha e número de ocorrências). Na *Indexer*, foi utilizado o *HashMap* como instanciador de objetos no retorno das buscas, para manter uma ligação entre as palavras buscadas e o "nó" pertencente a elas.

## 4. Estrutura de dados

Como estrutura de dados, foi escolhida a árvore digital, ou árvore de prefixos, para realizar o processo de armazenamento das palavras e facilitar o processo de busca. A escolha se deu pelo fato da árvore ser uma estrutura ordenada que pode armazenar arrays associativos, o que se relaciona com o intuito do projeto que é encontrar caracteres relacionados.

A estrutura apresenta busca mais rápida quando comparada a outras estruturas - árvore de busca binária, por exemplo -, além de que, ocupa menos espaço quando contém cadeias curtas, pois os nós das chaves iniciais comuns são compartilhados. Na imagem 8 abaixo, é possível visualizar a organização da estrutura.

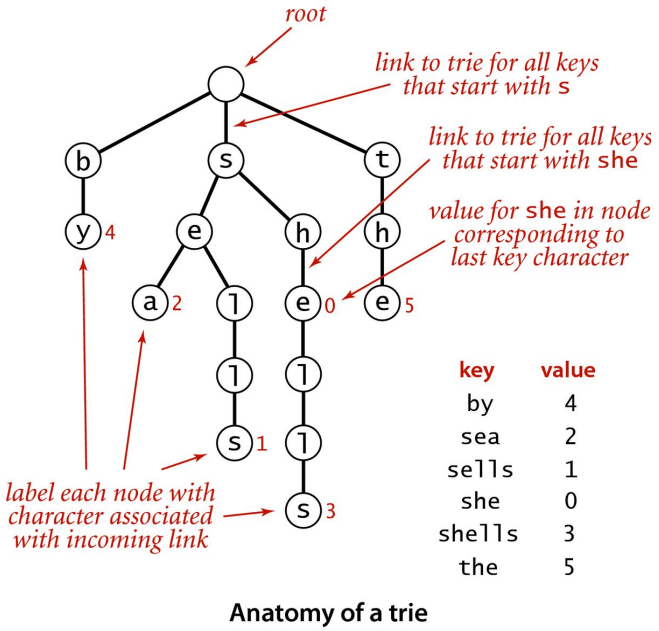


**Figura 8: Exemplo de funcionamento da árvore digital**

Dentro do sistema a árvore está representada na classe *Trie* e seus "nós" na classe *TrieNode* , cada nó possui um carácter como chave e o local onde ele se encontra no texto caso seja uma palavra completa, ou seja, o nível do nó indica a posição da letra na palavra a ser buscada, caso a palavra esteja na árvore ela possuirá um valor contendo em quais arquivos

ela se encontra, quais linhas e quantas vezes ocorre a palavra.

## 5. Reflexão

### 5.1 Qualidade do código

Levando em consideração a qualidade no processo de desenvolvimento, foi elaborado um diagrama de classe logo no início do processo de prototipação, entretanto, ao longo da implementação o diagrama sofreu diversas modificações, seguindo o modelo iterativo e incremental, o que facilitou a visualização do problema durante a evolução do projeto.

A princípio as responsabilidades foram bem definidas entre as classes, porém, uma extrema responsabilidade foi dedicada a classe *Parser* o que dificultou em alguns momentos a manutenção do código fonte, onde as demais classes tiveram uma grande dependência do objeto *Parser,* principalmente a classe *Indexer*.

Foi possível perceber a real importância de um código fonte com baixo acoplamento e alta coesão, pela facilidade de manutenibilidade e consequentemente um menor custo de tempo gasto para desenvolver o sistema, muitos dos bugs e problemas encontrados se deveu a algumas anomalias de código como a repetição de códigos, à exemplo os trechos dos métodos *searchOr* e *searchAnd* da classe *Indexer* e à classes extensas - *God class* - como a classe *Parser*.

### 5.2 Bugs e problemas encontrados

Dentre os problemas percebidos, estão no tratamento de caracteres especiais, o sistema não diferencia as palavras com caracteres especiais de outras e trata ambas como iguais.

Quanto ao desafio proposto, que se baseava na tentativa de implementar o algoritmo da distância de Levenshtein, ele não pode ser concluído a tempo.

### 5.3 Aprendizagem

Relacionando o que foi ensinado durante a disciplina com a elaboração do projeto final, houveram habilidades adquiridas que puderam ser aprimoradas e melhor compreendidas durante a conclusão da disciplina.

Como sugestão, um estudo mais detalhado referentes a elaboração de diagramas de classe e implementação, acompanhando as etapas do processo de desenvolvimento, traria provavelmente melhores resultados na elaboração do projeto.

## 6. Conclusão

O projeto de sistema de busca e indexação de arquivos proporcionou aperfeiçoamento

nos conceitos como os de acoplamento, coesão, responsabilidade. O aprofundamento em estruturas de dados foi importante tanto para entender melhor seu funcionamento quanto para a sua aplicação computacional.

Contudo, os resultados obtidos foram satisfatórios dentre as expectativas.

## 7. Referência


Trie.Disponívelem <https://pt.wikipedia.org/wiki/Trie#Vantagens_e_desvantagens_com_rela.C3.A7.C3.A3o_a_.C3.A1rvore_de_busca_bin.C3.A1ria>. Acessado em dezembro de 2017.

Estruturas de Dados:Tries. Disponível em <https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/tries.html>. Acessado em dezembro de 2017.

Projeto final LP2. Disponível em <https://github.com/ailsonfds/Trabalho-Final-LP2>. Acessado em dezembro de 2017.