Dicionário e corretor de palavras de textos utilizando retrieval tree

Bruno Rios Sousa, Carlos Eduardo Ramo da Silva, Guilherme Fuzisawa de Sá, Murillo Dias, Rafael Telles, Victor de Oliveira

[1]Centro de Matemática, Computação e Cognição – Universidade Federal do ABC (UFABC)
Caixa Postal 09210-580
Santo André – SP – Brazil

[2]

bruno.rios@aluno.ufabc.edu.br, c.eduardo@aluno.ufabc.edu.br,

guilherme.fuzisawa@aluno.ufabc.edu.br, murillo.dias@aluno.ufabc.edu.br,

rafael.telles@aluno.ufabc.edu.br, victor.oliveira@aluno.ufabc.edu.br

Abstract. *This article aims to explain how the creation of a dictionary and spell checker was made. The fundamentals used were the use of the Trie data structure which is a structure widely used for data recovery, its name coming from the word "reTRIEval". We also use the Python programming language, as it is a simple and effective language, and there is no need to worry about low level details. Defining a key in an attempt not to be a unique value but a present symbol sequence makes TRIE a good choice for dictionary and spell checker creation.*

Resumo. *Este artigo tem o objetivo de explicar sobre como foi feita a criação de um dicionario e corretor ortográfico. Os fundamentos utilizados foram o uso da estrutura de dados Trie que é uma estrutura muito usada para recuperação de dados, sendo seu nome vindo da palavra "reTRIEval". Também utilizamos a linguagem de programação Python, por ser uma linguagem simples e eficaz,e por não ser necessário ter preocupações com detalhes de baixo nivel. Definir uma chave em uma tentativa de não ser um valor único, mas sim uma sequência de símbolos presente, torna TRIE uma boa opção para a criação de dicionário e corretor ortográfico.*

1. Introdução

A grande maioria das estrutura de dados estudadas assumem que a aplicação utilizará números, isto é, onde conseguimos fazer comparações diretas e em tempo constante de chaves, como em árvores binárias de busca. Porém, em processamento de texto, estas estruturas não se comportam de forma adequada, pois textos apresentam estruturas com comprimento variável e sua ordenação pode ser feita de forma lexicográfica, o que pode não representar fielmente a proximidade de duas palavras [Brass 2008]. Surge então a necessidade de abordar estruturas que consigam se moldar a estas características, principalmente em aplicações como dicionário de palavras, corretores de textos, entre outros.

A primeira estrutura de texto foi criada por de la Briandais (1959) e foi denominada de "trie"derivado do nome de retireval trees [De La Briandais 1959]. Diferente de

uma árvore binária, agora cada nó da árvore contém potenciais arestas para cada letra em sequência no seu conjunto de texto, isto é, a árvore é criada de acordo com um dicionário estabelecido, e buscamos uma palavra na árvore lendo cada letra em sequência. Assim, ao invés de nós com dois filhos, podemos ter nós de até A filhos, sendo A o tamanho do alfabeto do dicionário. Portanto, trata-se de uma estrutura onde conseguimos determinar palavras próximas de forma mais adequada do que uma simples ordenação lexicográfica, sendo ideal para estruturas como dicionários e editores de texto. Outra vantagem é a utilização da memória secundária ao carregar a árvore. Para dicionários grande, fica inviável a utilização da memória principal, assim, conseguimos utilizar conceitos de paginação e memória secundária para implementar o algoritmo e facilitar na construção da aplicação.

Dado o que foi descrito nos parágrafos anteriores, este trabalho tem como principal objetivo a construção de uma interface capaz de realizar correção ortográfica de textos utilizando árvores trie. Para este objetivo, escolhemos duas linguagens de programação: JavaScript e Python. A primeira foi motivada pela sua abrangência e fácil implementação ao tratarmos de estruturas de front-end, isto é, na qual o usuário irá interagir, mais especificamente no editor de texto. Já a segunda, foi escolhida para a implementação da árvore por ser uma linguagem de alto nível e com uma diversidade de bibliotecas capaz de auxiliar na construção do mesmo.

Este trabalho está organizado da seguinte maneira: Na seção 1.1 detalhamos os objetivos deste trabalho. Na seção 2. trataremos sobre os fundamentos que utilizamos no algoritmo, ja na seção 3 falaremos sobre trabalhos relacionados, que ja foram realizados anteriormente, Nas seções 4, 4.1 e 4.2 ira abranger toda a parte de desenvolvimento, construção da Árvore Trie e Número de acessos respectivamente, a seção 5 traz a conclusão e por ultimo, na seção 6 estão as referências que foram utilizadas para a elaboração do algoritmo.

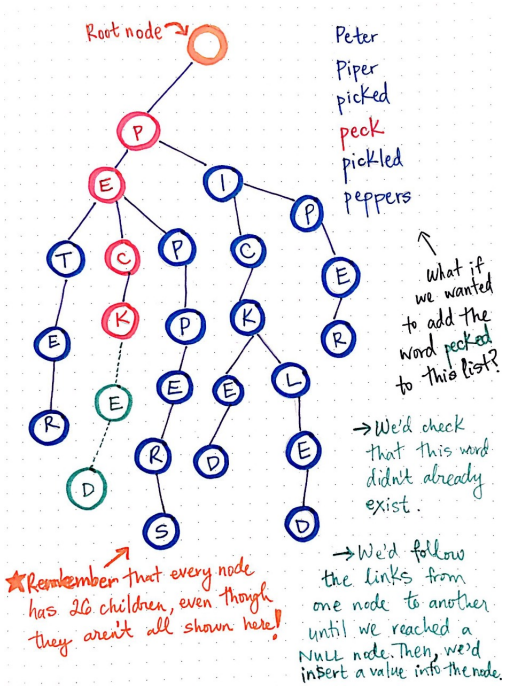
1.1. Objetivos

O objetivo deste trabalho consiste em implementar um editor de texto capaz de realizar correção ortográfica de palavras utilizando retrieval trees. Para isto, os seguintes objetivos específicos são citados.

- Implementação da árvore trie utilizando a linguagem de programação Python.
- Implementação de uma interface front-end utilizando a linguagem de programação JavaScript.
- Implementação de uma API para conexão da interface com a árvore que realizará tarefas de correção ortográfica.

2. Fundamentos

O problema consiste na criação de um dicionário e corretor ortográfico na qual sua base de dados deve ser grande e única , também deve ser localizada em memória secundária de uma maquina remotamente localizada , para esse problema utilizamos TRIE como a estrutura de dados. A escolha da utilização de Tries foi devido a ela ser uma estrutura simples e que torna possivel determinar palavras proximas tornando se ideal para este tipo de problema, isso se deve a estrutura comparar digitos de chaves individualmente ao invéz da chave inteira,essa estrutura tem alguma de suas caracteristicas como sendo:



Exemplo de implementação de TRIE para dicionário.

- Nenhum nó armazenar essa chave inteira,a chave é determinada pela posição na árvore.
- O caminho da raiz para qualquer outro nó é um prefixo de uma String.

Outra vantagem da utilização da Trie é que ela utiliza memoria secundária ao carregar a árvore, facilitando na construção do dicionário devido a ser possível a implementação de dicionarios grandes. Para o desenvolvimento do algoritmo utilizamos a linguagem de programação JavaScript no Front-End e Python para a implementação da Árvore. A construção da árvore foi feita no back-end, ja no front-end a palavra era usada para armazenar as palavras existentes.

3. Trabalhos Relacionados

Trabalhos com propostas similares a este já foram realizados no meio acadêmico, foram utilizadas diversas técnicas e ferramentas para os estudos de caso de cada estudo. Dentre elas podemos citar o Processamento de Linguagem Natural(PLN), programação paralela e aprendizado de máquina. No geral, a correção ortográfica pode ser dividida em dois passos: Primeiro identificar a palavra errada e, segundo, substituir pela palavra correta. Abaixo é exposto alguns aspectos de trabalhos que possuem tais características.

Em [Nunes and Oliveira Jr 2000], é apresentado os aspectos do desenvolvimento do ReGra, um revisor gramatical em português, que possui como atributos três módulos principais: o módulo estático, mecânico e o módulo gramatical. O módulo estático tem como sua principal função retornar o nível de legibilidade do texto. O módulo mecânico busca detectar erros facilmente detectáveis por pessoa, mas não por um corretor ortográfico, por exemplo: Alguma pontuação isolada no texto. Por fim, o módulo gramatical, que procura detectar erros de concordância verbal/nominal e acentuação incorreta. No geral, erros que fogem da Norma Culta. A atuação desses módulos se apro-

Língua	Nº de palavras	Tempo médio de execução (s)
inglês	80004	7.67
chinês	802	7.38
croata	28110	7.49
italiano	360832	8.54
latin	154194	7.90
alemão	339409	8.57
húngaro	17981	10.54
grego	111035	8.00
japonês	231180	8.07

Tabela 1. Medidas para construção da árvore Trie em cada língua

ximava de uma abordagem baseada em erros, assim o corretor se limitava à problemas de combinações de palavras. Dessa forma, o ReGra se propôs a analisar o itens lexicais sintaticamente, utilizando etiquetamento morfossintático, com isto foi possível categorizar, abstrair as palavras para analisar e, por fim, realizar o processamento.

Outro trabalho que tange o problema de corretor ortográfico é o Certografia [Pittol and Rigo 2015], este estudo apresenta uma proposta utilizando o aprendizado de máquina como seu mecanismo para realizar a verificação ortográfica. Utilizando o aprendizado supervisionado para alimentar o corpus, a ferramenta se baseia em outros textos para expelir a sugestão. Vale notar que o cálculo probabilístico para a palavra correta não se baseia em todo conjunto de exemplos, mas sim nas N palavras anteriores à palavra que será corrigida. Este aspecto, nos sistemas atuais, é chamado de N-gram onde a principal característica é levar em conta a forma flexionada da palavra.

O trabalho apresentado em [Andrade et al. 2012] utiliza dois dicionários devido ao seu contexto, corretor ortográfico de páginas web, sendo eles respectivamente um dicionário de palavras em português e o outro com hábitos de linguagem(”Internetês”), estes dois armazenados em memória secundária. Aqui os autores utilizaram programação paralela e tabelas hash, pois o foco era ter o menor custo de pesquisa possível. Por fim, este trabalho se mostrou bastante eficiente quando o volume de palavras é muito grande.

4. Desenvolvimento

Nesta seção vamos apresentar como implementamos o corretor ortográfico, bem como suas funcionalidades, testes de performance, e análise de complexidade do algoritmo proposto.

4.1. Construção da Árvore Trie

Ao inicializar, o algoritmo precisa construir a árvore Trie de um corpus de dicionário, isto pode ser de uma única lingua ou de várias. Foram realizados alguns experimentos para analisarmos o tempo de execução para diferentes corpus, entre eles: português, inglês, chinês, croata, italiano, latim, alemão, húngaro e grego. A tabela 1 detalha o tamanho do dicionário, bem como a figura 1 mostra graficamente com os erros de execução para cada iteração. No total, sete experimentos foram realizados para cada língua.

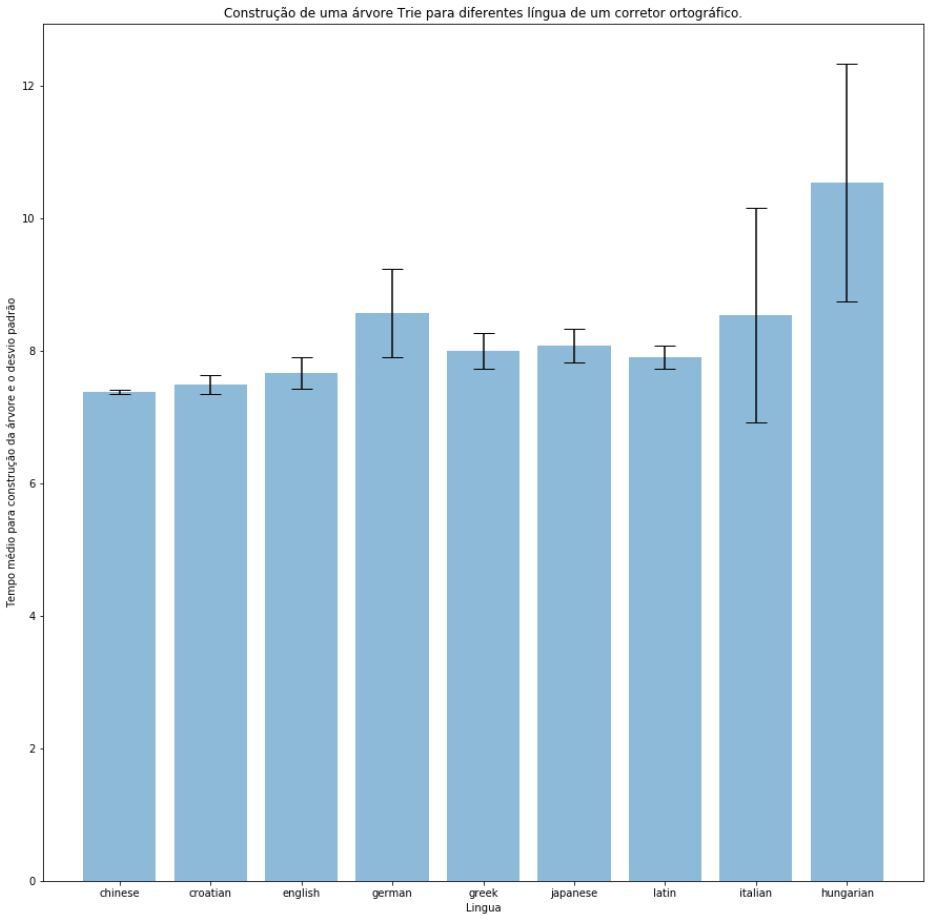


Figura 1. Tempo médio de execução da construção da árvore para cada língua

Também analisamos o comportamento do algoritmo para o tamanho do texto, como mostra a figura 3 abaixo.

Interessante notar que conforme o corpus cresce, temos um crescimento linear no tempo médio que o algoritmo leva para construir a árvore, porém com um ponto de outlier que corresponde a língua húngara. Isto pode acontecer pelo número de letras em palavras típicas da língua húngara, assim cada nó teria no caso mais opções, o que não ocorre muito em línguas dominadas por poucas letras como o inglês, por exemplo.

4.2. Número de Acessos

Após a construção da árvore, o arquivo está pronto e o corretor começa a funcionar de forma dinâmica. A construção do arquivo se dá na plataforma backend da nossa aplicação, conforme detalhada na seção de Fundamentos. Pelo JavaScript, é armazenado o conjunto de palavras existente no editor e se uma nova palavra é inserida, o mesmo entra e é checado. Se a palavra estiver incorreta, procura-se a mesma na árvore, vamos analisar o

comportamento da aplicação nesta fase com alguns textos e línguas diferentes. Para os testes, utilizamos a biblioteca nltk do python que contém diversos textos exemplos, de diversos tamanhos. Então, para cada texto, rodar a busca para cada palavra que o corretor supostamente deveria analisar. A figura 2 ilustra o experimento.

Vemos na figura claramente um comportamento linear, isto é, conforme o número de palavras no editor aumenta, a busca das mesmas leva um tempo proporcional linearmente. Isto mostra uma certa eficiência ao carregar o editor de textos com escritas grandes, pois conseguimos um tempo de execução próximo de ótimo.

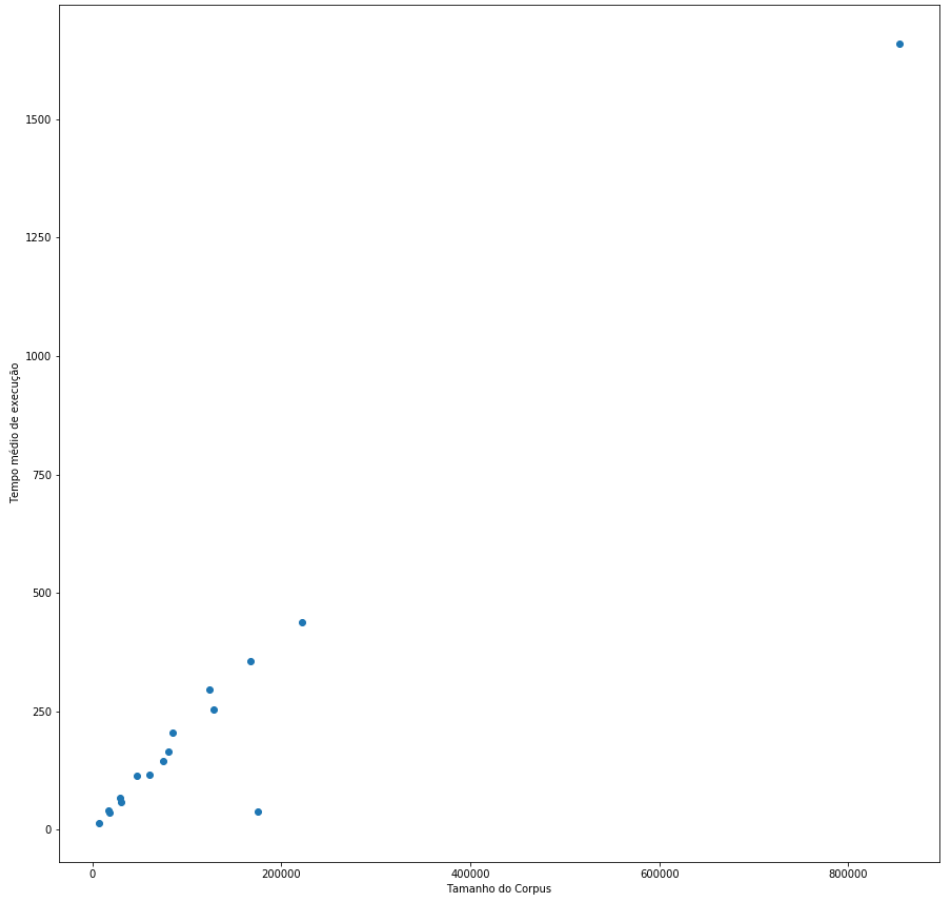


Figura 2. Análise de execução de tempo para search trie

5. Conclusão e Trabalhos Futuros

Com esse trabalho pudemos concluir que o algoritmo atendeu as expectativas do grupo e se mostrou eficiente na construção de dicionarios e de corretor ortográfico utilizando Tries. A utilização da linguagem Python contribuiu para a construção da árvore devido a

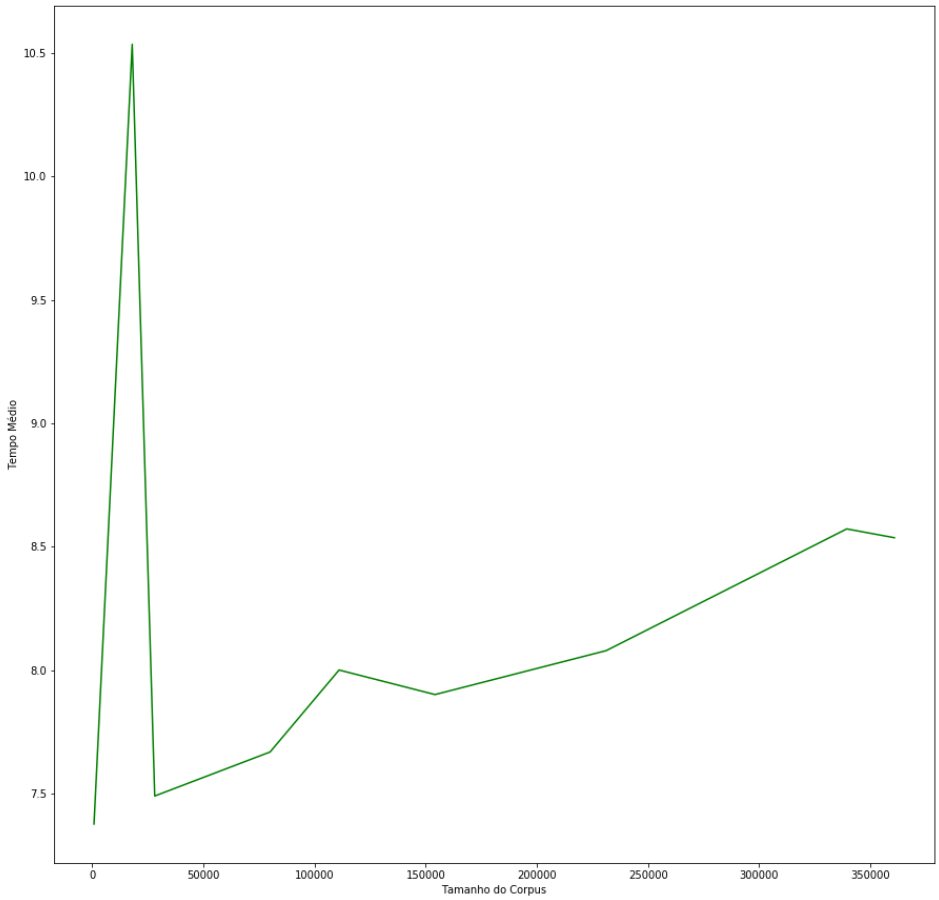


Figura 3. Tamanho do dicionário versus tempo de execução para criação da árvore

sua simplicidade e suas bibliotecas tendo como exemplo a biblioteca counter. Já o uso de Javascript atendeu aquilo que o grupo esperava ao fornecer uma tela de interação para o usuário, fazendo com que o uso do corretor possa ser mais simples. Para trabalhos futuros propomos fazer uma implementação que use menos espaço em disco, tornando as buscas mais eficientes, e tambem seria possivel deixar o dicionario ao lado do cliente ao invez de ser remoto.

Referências

Andrade, G., Teixeira, F., Xavier, C., Oliveira, R., Rocha, L., and Evsukoff, A. (2012). Hasch: High performance automatic spell checker for portuguese texts from the web. *Procedia Computer Science*, 9:403–411.

Brass, P. (2008). *Advanced data structures*, volume 193. Cambridge University Press Cambridge.

De La Briandais, R. (1959). File searching using variable length keys. In *Papers presented at the the March 3-5, 1959, western joint computer conference*, pages 295–298. ACM.

Nunes, M. d. G. V. and Oliveira Jr, O. (2000). O processo de desenvolvimento do revisor gramatical regra. In *Anais do XXVII SEMISH (XX Congresso Nacional da Sociedade Brasileira de Computação)*, volume 1, page 6.

Pittol, E. and Rigo, S. J. (2015). Certografia: um corretor ortográfico automático para português e resultados de um estudo de caso aplicado na área jurídica. *Revista Brasileira de Computação Aplicada*, 7(3):31–42.