Search Engine

Cristiana Aparecida Nogueira Couto
Hanna Rodrigues Ferreira
Marcos Antônio Alves

Junho de 2020

1 Resumo

O projeto consiste de um mecanismo de busca de palavras em uma coleção de documentos, onde os resultados são exibidos no console ou na interface web, ordenados por ordem alfabética de títulos.

Criamos uma interface web para efetuar pesquisas e acessar resultados; o objetivo é oferecer uma dinâmica melhor entre o usuário e o programa. Além disso, há também um sistema de sugestões de palavras,tanto para pesquisas de palavras únicas quanto para frases.

2 Instruções

Para executar o programa é preciso fazer o download do repositório em https://github.com/Cristiananc/Search_Engine. O compilador usado é o C++ 11 GNU. Os pacotes utilizados estão carregados nas funções. Para executar a interface web localmente é preciso ter instalado o pacote boost. Maiores detalhes serão dados no arquivo "README.md".

3 Descrição

3.1 Estrutura de Dados

A estrutura de dados utilizada para o trabalho foi uma *Trie* . Cada nó possui um array com trinta e seis ponteiros, onde os primeiros dez índices indicam os algarismos de 0 à 9. Os demais são as letras do alfabeto indo de a até z.

index do array children	0	1	2	...	8	9	10	11	...	34	35	36
elemento do alfabeto	0	1	2	...	8	9	a	b	...	w	y	z

Além disso, cada nó possui um vetor de inteiros. Se o caminho feito até um certo nó representa uma palavra, no vetor desse nó se encontram os *Ids* em ordem crescente dos textos onde essa palavra aparece. *Ids* são os identificadores dos textos; cada texto possui um *Id* diferente. Para realizar uma pesquisa, basta caminhar nos arrays de ponteiros usando os índices de acordo com os caracteres.

Para realizar uma pesquisa, basta caminhar nos arrays de ponteiros usando os índices de acordo com os caracteres.

3.2 Pré-processamento

A seguir encontram-se os detalhes das atividades do pré-processamento:

- Remoção de caracteres especiais como acentuação, pontuação, etc. Isto simplificou a entrada de dados da árvore; pois muitos caracteres não fazem parte do conteúdo, são apenas identificadores presentes em todos os textos.

- Ordenação em ordem alfabética por títulos. Essa parte foi essencial para podermos indexar os Ids dos textos de cada palavra em tempo constante; pois desta forma o id que estivermos inserindo será sempre maior ou igual aos anteriores, logo, o adicionamos no final do vetor.

- Remoção de palavras repetidas num mesmo texto. Reduz o espaço de memória e melhora o tempo de execução para a construção da árvore. Com estes procedimentos; todos os 164 arquivos foram organizados em 136 arquivos.

Nossa proposta é exibir como resultado os textos originais e deixar os textos processados apenas para a construção da árvore.

Com o pré-processamento feito também foi possível realizar a interseção dos vetores de ids em tempo linear. A interseção é realizada dois a dois e funciona para pesquisas com multi palavras.

3.3 Funcionalidades extras

3.3.1 Sugestão de palavras

A sugestão de palavras funciona da seguinte forma: ao digitar uma palavra e não obter resultados, são oferecidas 5 opções de palavras semelhantes que se encontram em pelo menos 1 texto do corpus. No caso de uma frase oferecemos outra como sugestão, onde todas as palavras se encontram em algum texto.

Não encontrar um resultado significa que o vetor de inteiros dos ids é nulo. Para o caso de uma frase há a possibilidade de não existir textos com a interseção dessas palavras, o que não implica que elas estejam escritas de forma incorreta. Por isso precisamos separar em dois casos. Observamos cada palavra individualmente, caso haja uma que não exista fazemos a sugestão de palavras para ela. Retornamos uma frase como sugestão, sendo o primeiro elemento do conjunto de sugestões, de modo que o usuário, no terminal pode optar por refazer a pesquisa.

O número de sugestões foi escolhida de modo arbitrário. No caso das frases, optou-se por apenas uma sugestão pois a depender da quantidade n de palavras escritas erradas numa frase, existem 5^n combinações possíveis de sugestões de frases.

As sugestões são calculadas considerando que as palavras sugeridas possuem até duas edições de distância, sendo considerada a distância de Damerau–Levenshtein. Essa distância inclui transposição entre dois caracteres adjacentes, exclusão, interseção e substituição de um caracterer. O algortimos foi feito com base em [1].

3.3.2 Interface web

O código do servidor local foi disponibilizado pelo monitor. Foi implementada a interação entre o Java Script, C++ e Html. A função *query* no JavaScript recebe o texto submetido no quadrado de busca e chama a função de limpeza do input e pesquisa na árvore. Feito isso retorna uma lista com os primeiros

20 títulos. Cada título está associado a um hiperlink, o qual quando clicado abre o texto correspondente ao título. Há também um hiperlink com a palavra *next* que chama a função que retorna os títulos para os próximos 20 títulos da lista e assim sucessivamente.

4 Resultados

Todos os 136 arquivos foram indexados. Usamos um espaço de memória adicional de tamanho linear de 1,7 GB (na verdade é menos da metade do tamanho dos arquivos originais que ocupam 3,5 GB). Procuramos otimizar este espaço ao eliminar palavras repetidas no pré-processamento. O tempo de execução da árvore oscila entre 80 e 120 segundos em nossos computadores como mostra a figura 1.
Na figura 3 está indicado o espaço de RAM; em um computador com 16Gb de memória RAM; gastou cerca de 4Gb.

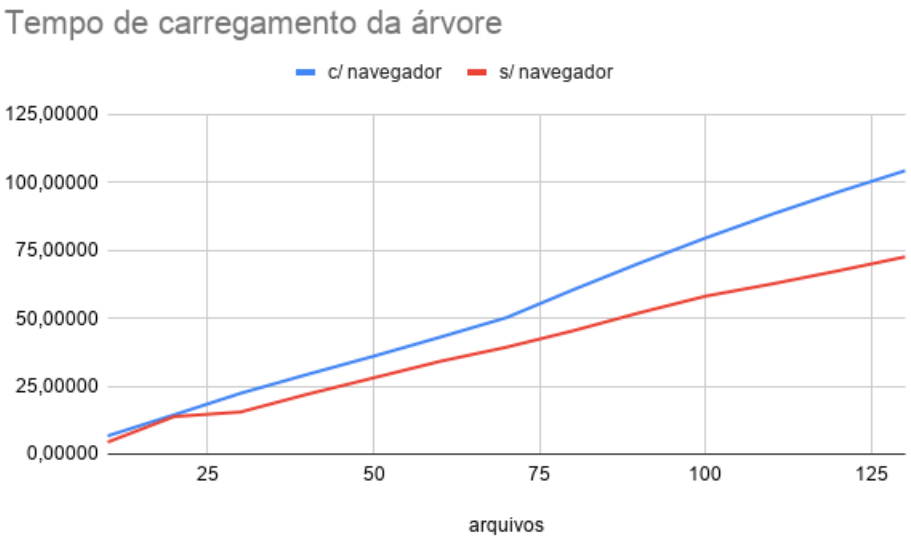

Figure 1: Tempo de construção

Na figura 2, os dados foram obtidos em uma máquina com as configurações:

Intel Core i5-10210U CPU @ 1.60GHz x 8.

Na figura 2 estão algumas pesquisas com seus tempos de busca indicados. Foram pesquisadas diferentes quantidades de palavras, todas tendo seis caracteres, as quais obtivemos através de [4] , que gera palavras aleatórias.

entrada	tempo de busca	resultados
future	0,001082	57043
health patrol	0,003129	450
scrape gutter affect	0,000235	1
crutch endure arrest mobile	0,000731	0
clique finger marine hunter arrest	0,000893	0

Figure 2: Amostra de pesquisas

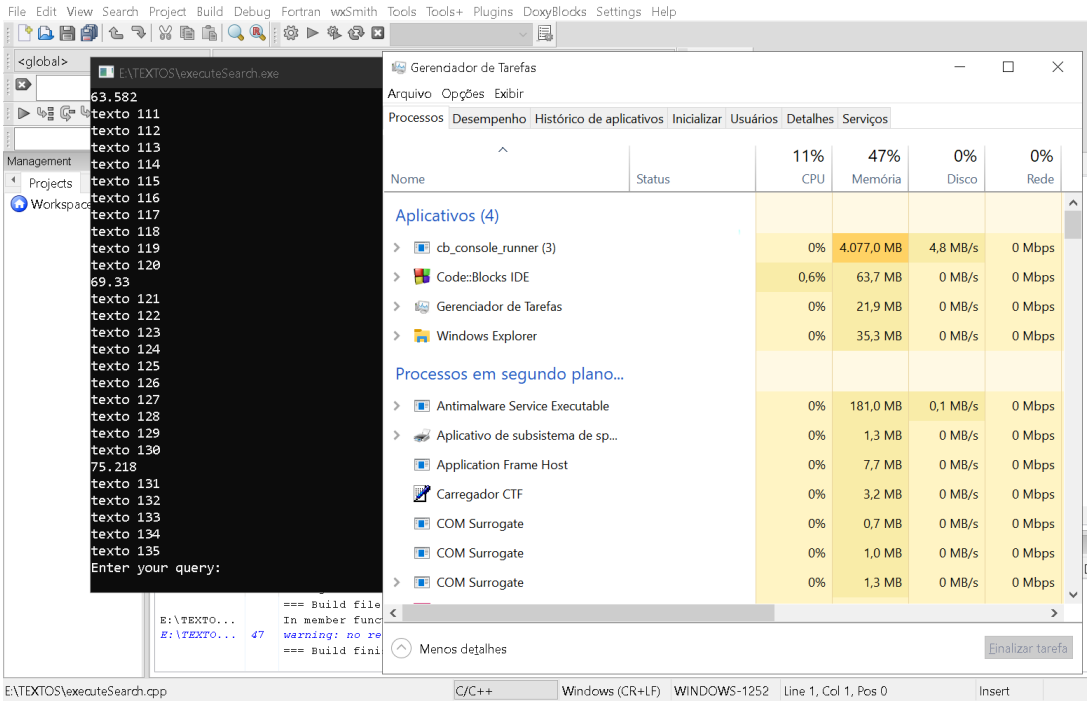

Figure 3: RAM utilizada

5 Limitações do trabalho

O programa não diferencia caracteres maiúsculos e minúsculos e caracteres que não sejam letras ou algarismos são ignorados. A interface web funciona apenas localmente e está num formato básico de funcionamento. A sugestão de palavras para frases retorna apenas uma sugestão. Nosso programa lida apenas com palavras e não com a estrutura das frases, ou seja; cada palavra é tratada como uma palavra chave até mesmo artigos e preposições como the, a, in, to etc.

6 Trabalhos futuros

Como trabalhos futuros a sugestão de palavras poderia ser feita utilizando pesos, ou seja; verificar o número de ocorrências de uma palavra no texto e a partir disso ordenar os resultados de acordo com o número de ocorrências.

A interface web poderia ser um pouco mais elaborada, melhorar o design da interface, colocar mais opções como pesquisa por intersecção ou união de Ids, gráficos etc.

Além disso, um sistema de sugestão de palavras mais complexo para frases, apresentando mais do que uma sugestão. Também seria interessante não considerar apenas as palavras numa frase, mas também sua estrutura; ou seja, considerar um texto apenas se as palavras no texto aparecem na mesma ordem da frase.

7 Conclusões

Concluímos que apesar das limitações conseguiu-se chegar a um resultado satisfatório.

O tempo de execução das pesquisas na maioria dos casos não chega a 10^{-3} segundos. Além disso, conseguimos um tempo razoavelmente rápido para a inserção das 250 milhões de palavras na árvore.

Cabe agora, analisar as atividades futuras e implementar novas modificações que possam diminuir ainda mais esse tempo. Implementar um design para a interface web e investir em novas ideias para a melhoria do projeto.

8 Distribuição das atividades no grupo

Todos os membros participaram da tomada de decisão sobre a estrutura de dados selecionada e ideias sobre as implementações. A seguir estão listadas as atividades feitas por cada um mais detalhadamente:

- Cristiana:
  - Auxiliou na implementação da estrutura de dados;
  - Implementação da função de pesquisa;
  - Implementação da pesquisa via web;
  - Implementação da sugestão de palavras.
- Hanna:
  - Implementação da estrutura de dados;
  - Serialização e Desserialização dos dados carregados na árvore, inacabada.
  - Edição do video de apresentação.
- Marcos Antônio:
  - Pré-processamento dos dados: limpeza; ordenação dos textos; eliminação de repetições;
  - Carregamento dos dados para a árvore e inserção em tempo otimizado;
  - Exibição dos resultados selecionados pelo usuário.

References

[1] How to write a spelling corrector. `http://norvig.com/spell-correct.html`. Acesso: 2020-06-04.

[2] Introduction to tries. `http://people.cs.ksu.edu/~rhowell/DataStructures/trees/tries/intro.html`. Acesso: 2020-05-09.

[3] Radix tree. `https://en.wikipedia.org/wiki/Radix_tree`. Acesso: 2020-04-30.

[4] Random words generator. https://randomwordgenerator.com/. Acesso: 2020-06-17.

[5] re — regular expression operations. https://docs.python.org/3/library/re.html. Acesso: 2020-04-30.

[6] Serialization. https://en.wikipedia.org/wiki/Serialization. Acesso: 2020-04-30.

[7] Session 16: Strings. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-851-advanced-data-structures-spring-2012/lecture-videos/session-16-strings/. Acesso: 2020-04-30.

[8] Trie — (insert and search). https://www.geeksforgeeks.org/trie-insert-and-search/. Acesso: 2020-04-30.

[9] Tries (árvores digitais). https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/tries.html. Acesso: 2020-04-30.