Trabalho Prático 3 - Recuperação da Informação

Matheus Aquino Motta[1]

[1]Bacharelado em Matemática Computacional
DCC - Universidade Federal de Minas Gerais

matheusaquino199@gmail.com.br 2018046513

Abstract. *In this report we will briefly discuss the implementation of the assignment 3 of the subject Information Retrieval. The problem consisted into constructing an inverted index with the text words found on an mini-collection of the HTML documents crawled in the assignment 2. The algorithm implementation made use of the Gumbo Parser library to parse the documents and clean the text, to posterior word-to-word indexing of each page. Furthermore will be provided an proper statistical analysis about the space and time results achieved.*

Resumo. *Nesse relatório iremos discutir brevemente a implementação do trabalho prático 3 da disciplina Recuperação da Informação. O problema consistia em construir um index invertido a partir das palavras encontradas no texto de uma mini-coleção dos documentos HTML coletados no trabalho Prático 2. O algoritmo foi implementado a partir do uso da biblioteca Gumbo Parser, para realizar o parsing dos documentos, para posterior indexação palavra a palavra de cada página. Além disso, será apresentada uma análise estatística a respeito dos resultados de tempo e espaço alcançados.*

1. Introdução

O problema proposto no trabalho prático 2 consistia no desenvolvimento de um *crawler* focado na *web* brasileira e posterior coleta de 100000 páginas *HTML*. Dentre os resultados obtidos foram coletadas duas coleções de documentos, uma de caráter geral e outra focada especificamente na *web* brasileira, na qual pelos resultados obtidos foi possível inferir que todas as páginas coletadas eram potencialmente de domínios brasileiros. Assim, o objetivo do trabalho prático 3 consiste na implementação de um algoritmo para realizar a construção de um index invertido para as palavras encontradas no texto de documentos *HTML* pertencentes a uma mini-coleção das páginas coletadas no trabalho anterior.

Assim, seja C o cojunto de páginas coletadas no trabalho prático 2, e $S \subseteq C$ um subconjunto (mini-coleção) de C, queremos construir um index invertido $I = \{L_1, \ldots, L_k\}$, onde k é o número de palavras distintas encontradas nas páginas, e L_i, para $1 \leq i \leq k$, a lista que armazena informações acerca de em quais documentos a i-ésima palavra ocorre, quantas vezes ocorre e suas respectivas posições de ocorrência no documento.

De um modo geral o programa foi desenvolvido em torno da construção das listas L_i, onde para cada página $p \in S$ realizamos o *parsing* do seu código *HTML* e extraimos o texto do documento, assim, adicionando ao index informações relativas a cada palavra encontrada de maneira apropriada. Além disso, obtemos informações acerca do tempo de execução do algoritmo e informações estatísticas relativas ao tamanho da mini-coleção utilizada e do index invertido, assim como de suas listas individualmente.

2. Implementação

A estrutura do algoritmo pode ser visualizada a partir do fluxograma abaixo,

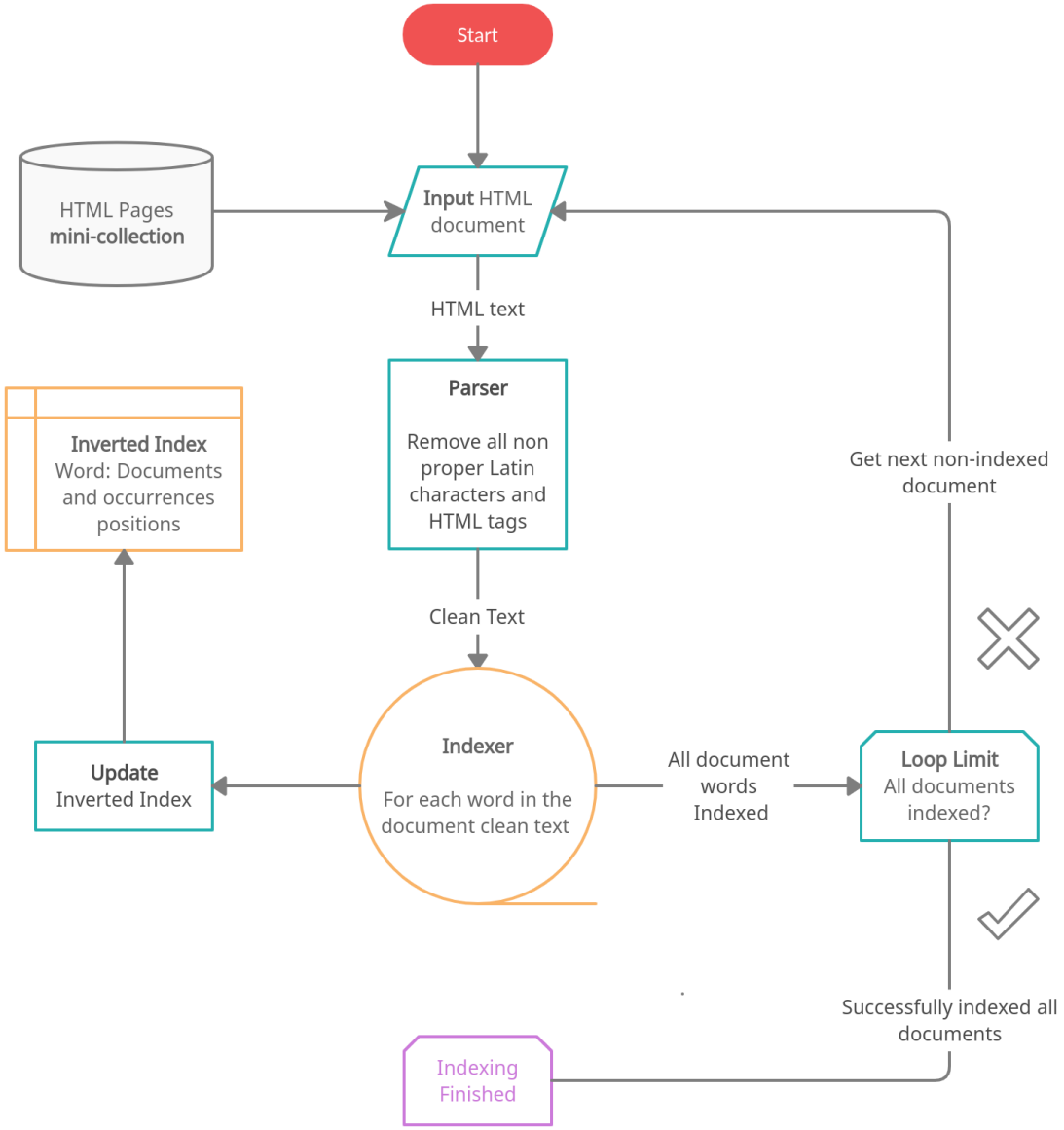

Figure 1. Fluxograma de funcionamento do Indexador.

onde para descrever o fluxo de execução e funcionamento de maneira apropriada precisamos de entender as funções centrais do coletor: a análise do texto recebido, realizada pelo *Parser*, e o indexador das palavras encontradas no texto.

2.1. Análise textual, *Parser*

Um problema central no desenvolvimento do algoritmo consistia em, tomado o conteúdo *HTML* de cada página, eliminar de forma efetiva tags e trechos de código do documento, para que tivessemos acesso ao conteúdo efetivo de cada página, problema que foi tratado diretamente a partir do uso do *parser* da biblioteca *Gumbo*.

Entretanto, além disso, sabemos que as listas invertidas são indexadas a partir das palavras encontradas, assim, precisamos definir quais caracteres seriam mantidos no texto que será utilizado para construir o index, de modo à garantir consistência semântica e gramatical de acordo com a língua portuguesa das palavras que serão anexadas ao index. Para fins práticos durante a análise textual todos os caracteres de pontuação e marcação foram removidos e apropriadamente substituidos por espaços, mantendo apenas caracteres do alfabeto Latino, assim como algumas de suas extensões, e algarismos arábicos, em uma *string* de palavras separadas por espaços únicos.

Assim, garantindo a consistência gramatical das palavras, onde caracteres com sinais diacríticos foram mantidos, isto é, caracteres com sinais gráficos usados para alterar a fonética das palavras como: acentos graves, agudos e circunflexos, til, etc; e assim, *substrings* do texto original como "algoritmo", "algoritmo," e "algoritmo." foram consideradas como uma palavra única, "algoritmo", haja vista que essas representam a mesma palavra a menos de caracteres de pontuação.

Desse modo, ao final teremos uma *string* texto de palavras separadas por espaços únicos, que serão utilizadas para a construção do index invertido.

Entretanto, a análise textual realizada garante apenas a distinção gramatical das palavras para o indexador, isto é, palavras gramaticalmente diferentes serão consideradas diferentes no vocabulário resultante. Com isso, palavras com mesmo valor semântico, mesmo que erroneamente escritas, serão consideradas palavras distintas, como "Brasil", "BRASIL" e "brasil" que semanaticamente podem possuir a mesma interpretação, mas são gramaticalmente distintas.

2.2. Indexador

Agora com o texto pronto para ser indexado iremos construir as listas invertidas para cada palavra a partir do uso de tabelas *hash*, a partir de uma iteração sobre o texto, que irá designar a cada nova palavra um id de acesso para um vetor que irá armazenar as informações relativas a ocorrência das palavras.

Desse modo, seja w uma palavra pertencente ao texto t_d do documento d, iremos adicionar w ao index invertido $I = \{L_1, \ldots, L_k\}$ da seguinte forma:

- Se w já foi encontrada anteriormente e possui um id dado pela tabela *hash*, tal que, hash[w] $= j$, para $1 \leq j \leq k$, então adicionamos informações relativas a ocorrência de w na lista L_j. Caso contrário, adicionamos uma nova lista L_{k+1} para armazenar informações acerca das ocorrências das palavras w nos textos da mini-coleção, e adicionamos w à tabela *hash* como hash[w] $= k+1$.

- Assim, a cada vez que adicionamos uma nova ocorrência de w a uma lista L_i, armazenamos, ou atualizamos informações relativas a quantas vezes w ocorre no documento d, assim como suas respectivas posições de ocorrência no texto t_d.

Com isso, ao fim obtemos um index invertido, onde para cada palavra w pertencente ao index I, temos uma lista L_j, onde $j =$ hash[w], com um valor n_i associado que determina o número de documentos em que w corre, e listas nas quais para cada documento $d \in L_j$, armazenamos quantas vezes w ocorre, assim como suas respectivas posições de ocorrência.

Podemos visualizar o resultado da indexação das palavras a partir do texto dos documentos dados como exemplo em aula (Aula 7).

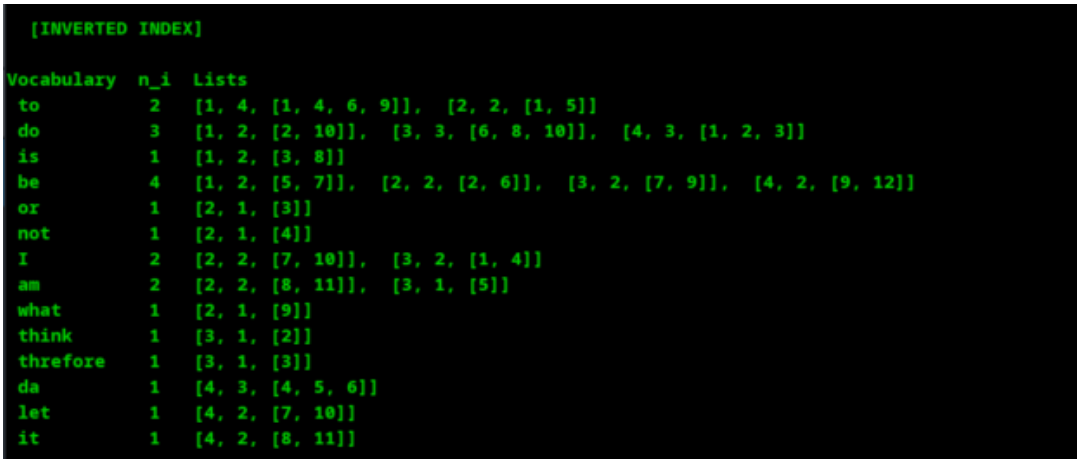

Figure 2. Resultado do index invertido usando documentos texto exemplo dados em aula.

2.3. Fluxo de execução

Assim, entendidas as partes centrais do código, o fluxo de execução é realizado de modo que definimos em nosso algoritmo o número de documentos que serão indexados. Desse modo, para cada documento realizamos o sua análise textual e *parsing* e posterior indexação, construindo o index invertido a partir de cada uma das palavras existente no texto.

Assim, o index é construído conforme as especificações dadas em aula com uma complexidade tempo de execução da ordem $O(|S|(P + \alpha n))$, isto é, para cada documento d pertencente a mini-coleção S realizamos o parsing do texto t_d de custo $O(P(t_d))$ (realizado pela biblioteca *Gumbo*), e posterior indexação da *string* s resultate do *parsing* de tamanho $n = |s|$, de custo $O(\alpha n)$, onde iteramos pelo pelo texto uma única vez e realizamos diversos acessos a estruturas da STL do C++ *unordered map*, usadas como tabelas *hash*, com operações de custo médio $O(1)$, o que pode resultar em variações no tempo de execução representadas por um custo α.

2.4. Bibliotecas e Estruturas de Dados

Para a implementação do trabalho foram utilizadas diversas estruturas da biblioteca STL do C++, como a estrutura *vector* para representação das listas criadas devido a maior praticidade no acesso das informações em $O(1)$ e alocação dinâmica de memória, com complexidade de espaço da ordem de $O(n)$, onde n é o tamanho do vetor.

Além disso, foi feito amplo uso da estrutura *unordered map* para realizar o *hashing* das palavras e compressão de *strings* em ids inteiros, para maior dinamicidade no acesso de posições de memória em estruturas e outras informações relevantes, onde cada operação realizada possui custo de tempo médio de $O(1)$ e complexidade de espaço $O(n)$, onde n é o número de elementos inseridos na estrutura.

Outras estruturas como *pair* e *string* também foram utilizadas para maior praticidade na realização de operações, acesso e armazenamento de informações ao longo da execução do algoritmo. E como já mencionado, para a realização da análise textual e *parsing* dos documentos *HTML* foi utilizada a biblioteca open source da Google, *Gumbo Parser*.

3. Resultados

Os resultados apresentados pelo algoritmo foram obtidos a partir de uma mini-coleção de 5000 documentos HTML retirada das 100000 páginas coletadas no dia 06 de janeiro de 2021 durante a realização do trabalho prático 2.

A mini-coleção utilizada possui um tamanho de aproximadamente 0.8 Gigabytes (799788.298 Kilobytes), como pode ser vizualizado pelo gráfico abaixo, que mostra o tamanho total da mini-coleção em função do número de documentos.

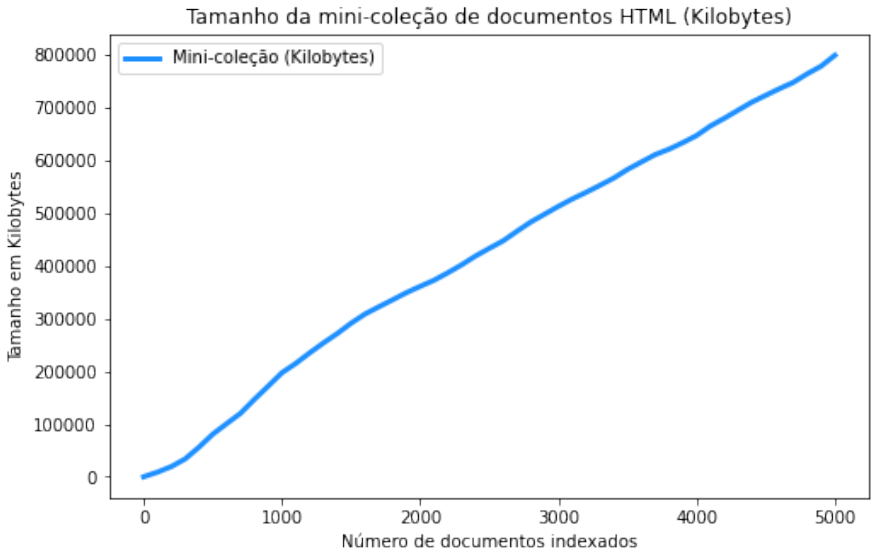

Figure 3. Tamanho da mini-coleção (Kilobytes) utilizada em função do número de documentos.

Após a construção index invertido podemos observar uma diminuição significativa de espaço em memória utilizada, de aproximadamente 0.036 Gigabytes (36468.476 Kilobytes), cerca de 4.5% do espaço ocupado originalmente.

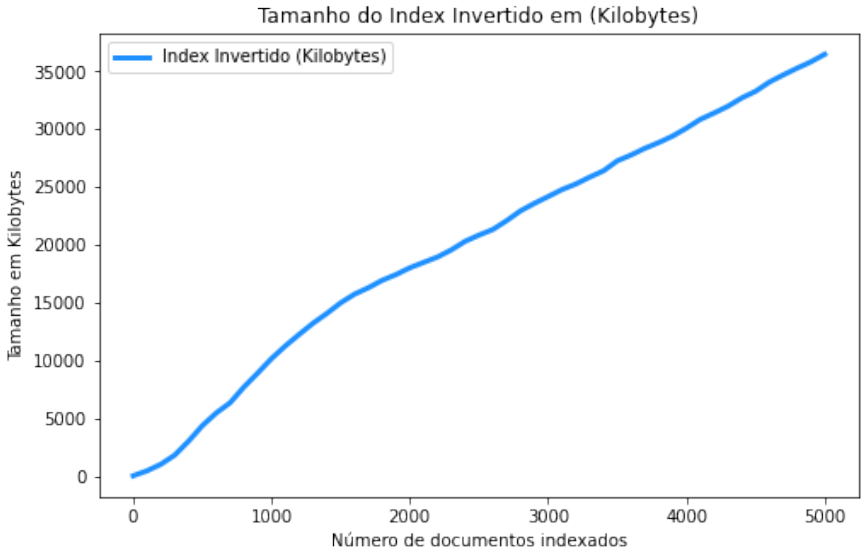

Figure 4. Tamanho do index invertido (Kilobytes) construído em função do número de documentos.

Essa variação pode ser observada a partir do gráfico comparativo abaixo, que demonstra o ganho computacional em termos de espaço após a indexação do texto presente nos documentos.

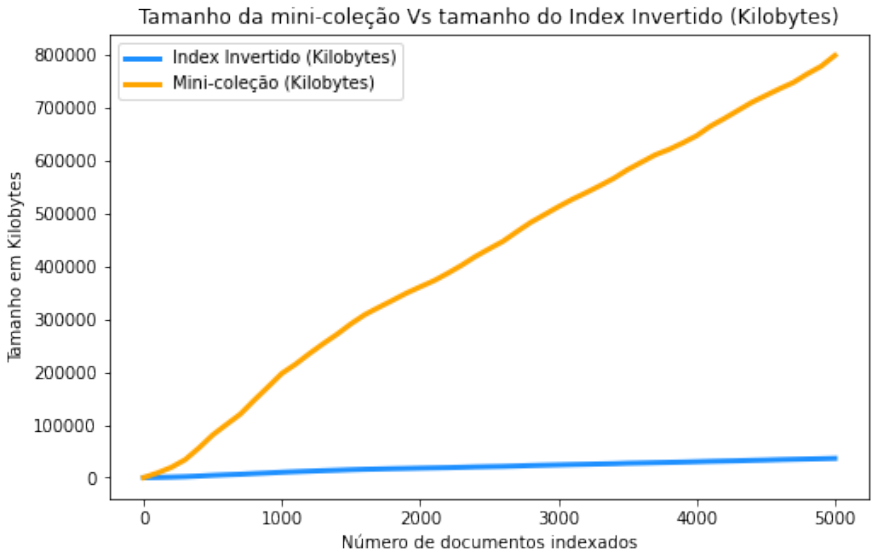

Figure 5. Comparação de espaço ocupado pela mini-coleção e index invertido

Além disso, temos que o tempo necessário para a construção do index invertido reflete a complexidade teórica esperada, como uma função quase linear a menos de pequenas variações, que podem ser observadas na curva observada do gráfico de tempo de execução do algoritmo. Onde ao final obtemos em diferente testes para a mesma mini-coleção que o algoritmo levou em média $\mu_t = 1.954$ minutos para ser executado com um desvio padrão de $\sigma_t = 0.074$.

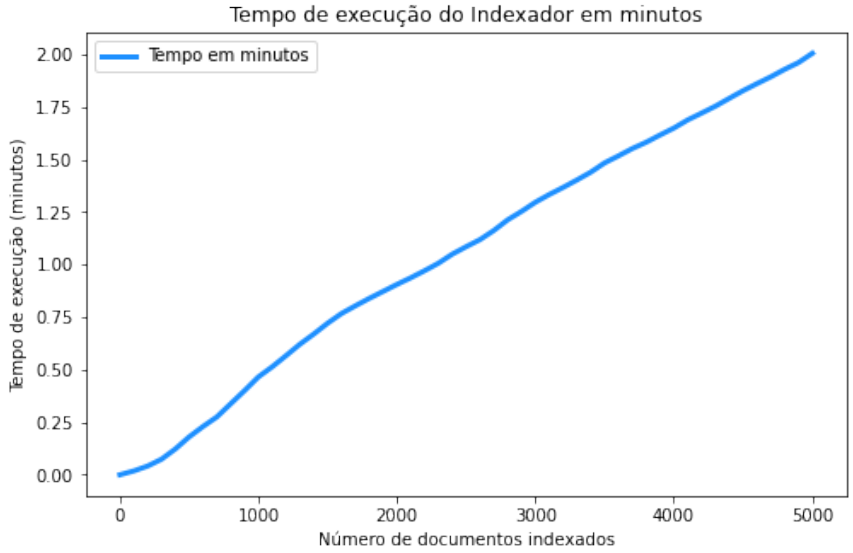

Figure 6. Tempo de execução do algoritmo em minutos.

Dados acerca de resultados de particularidades do index invertido construído também foram obtidos, como o tamanho do vocabulário (número de termos distintos) encontrado

no textos da mini-coleção. Esse que por sua vez possui um crescimento assim como esperado por métricas dadas em aula, de que o crescimento do vocabulário em função de uma coleção de n documentos é dado por uma função $O(n^{\beta})$, onde β está definido em um intervalo de aproximadamente $0.4 \leq \beta \leq 0.6$.

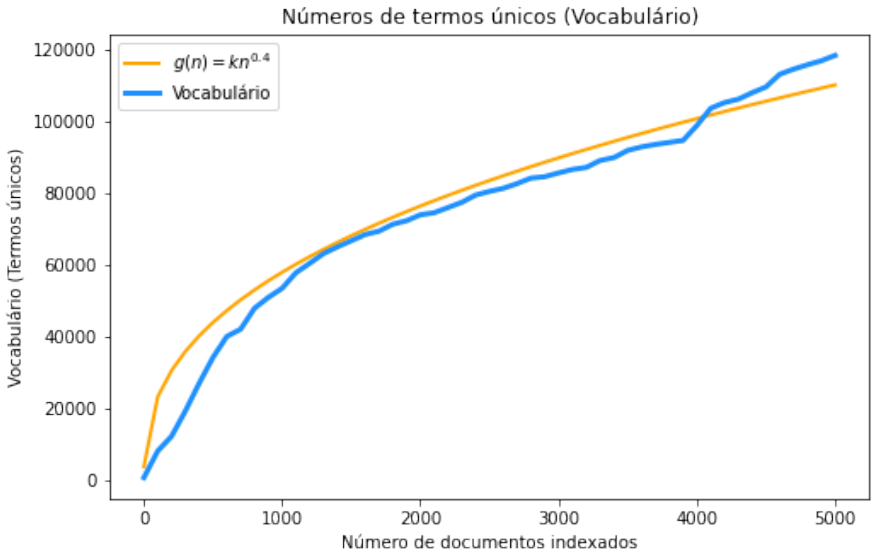

Figure 7. Tamanho do vocabulário (termos distintos) da coleção.

O crescimento sublinear do número de palavras é explicado devido a grande repetição dos termos encontrados no texto, o que é ainda evidenciado pela comparação entre o número total de termos na coleção com o número de termos que compõem o vocabulário, onde foi obtido respectivamente ao final um total de 6376292 palavras e um vocabulário de 118362 termos.

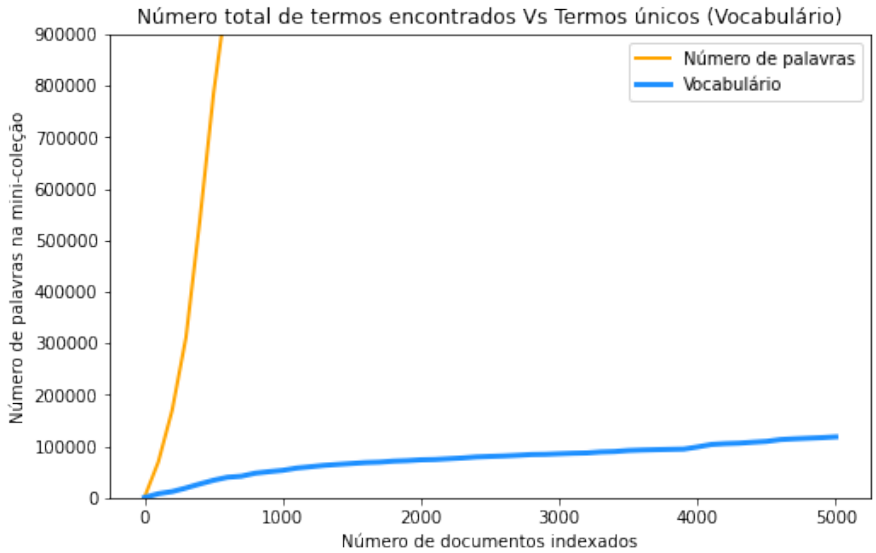

Figure 8. Comparação entre o número de termos encontrados e número de palavras que compõem o vocabulário.

Uma relação similar de crescimento também pode ser observada no gráfico relativo ao número médio de documentos n_i associado a cada palavra do indexador (*Average number of postings per term.*), como ilustrado pelo gráfico abaixo.

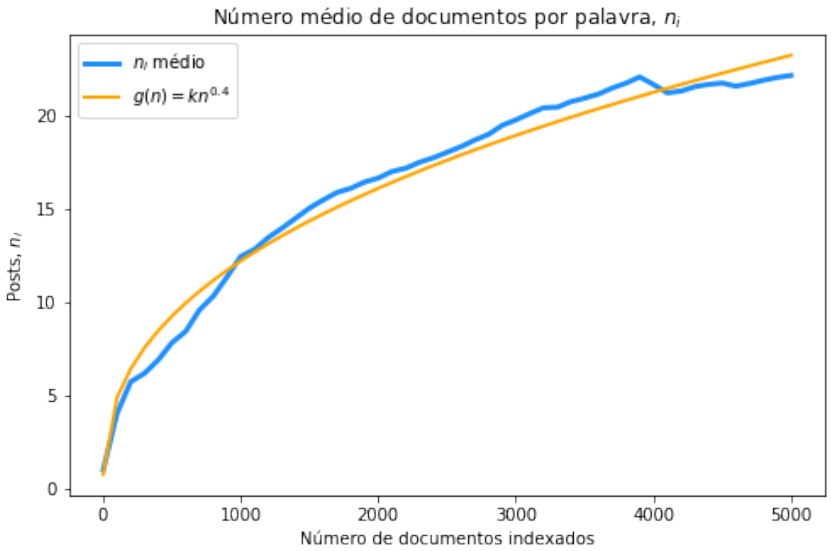

Figure 9. Número médio de *postings* por termo na coleção.

Por fim, uma análise individual qualitativa e quantitativa acerca dos termos mais frequentes também foi realizada, na qual foi possível observar a prevalência de artigos e preposições entre as palavras que mais ocorrem na mini-coleção.

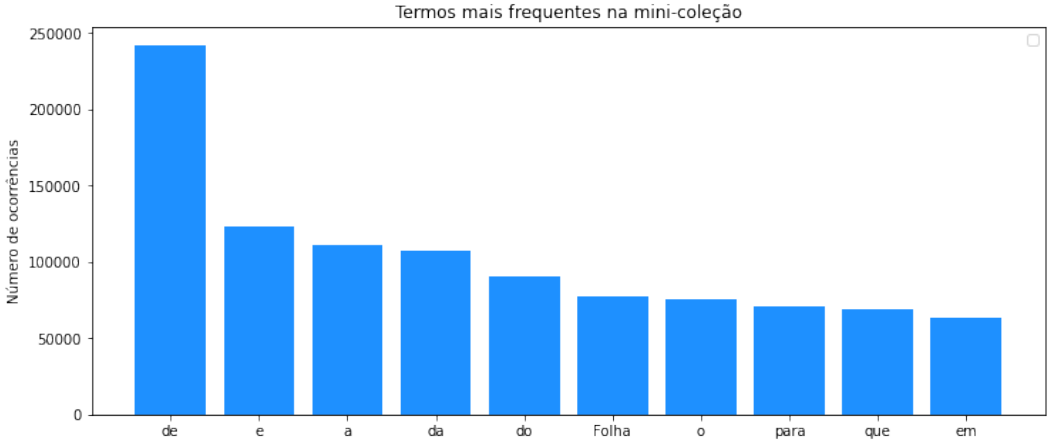

Figure 10. 10 termos do vocabulário mais frequentes na coleção.

A presença da palavra ”Folha” pode ser explicada devido ao fato de que uma das *seed URLs* utilizadas para a coleta dos documentos na *Web* foi a *https://www.folha.uol.com.br/*.

Por fim, dentre o conjunto das 100 palavras do vocabulário mais frequentes no texto, algumas podem ser notabilizadas pela sua relevância quando analisadas em perspectiva do cenário sócio-político do período no qual a coleta foi realizada.

Termos que refletem a alta disseminação de informação relacionada a pandemia do COVID-19 e personagens políticos de alta proeminência midiática.

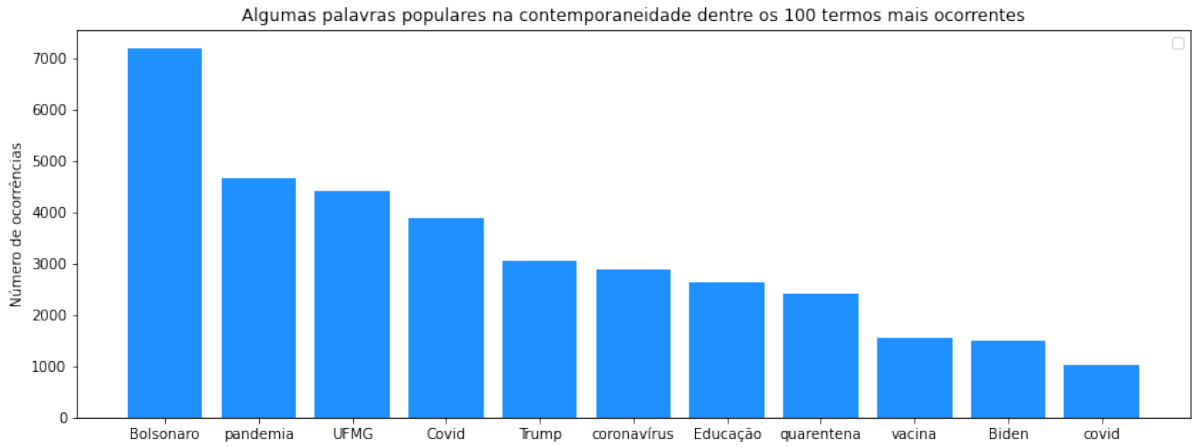

Figure 11. Termos frequentes e popularmente relevantes do vocabulário.

4. Conclusões

Nessa documentação apresentamos o desenvolvimento e resultados da implementação de um algoritmo para a construção de um index invertido para uma mini-coleção de páginas *web* coletadas no trabalho anterior.

Algumas dificuldades encontradas centraram-se principalmente em como tratar o código *HTML* coletado para obter acesso ao texto e termos individualmente de forma apropriada. As decisões tomadas trataram o problema de modo a garantir a consistência gramatical de termos da língua portuguesa e os resultados foram significativamente positivos, demonstrando que tais problemas foram devidamente tratados.

Algumas alternativas para melhora na eficiência do algoritmo podem ser feitas, como a compressão dos dados finais do index, para uma economia ainda maior do espaço ocupado em memória. Além disso, o uso de técnicas para tratar a inconsistência semântica do indexador para propósitos de busca, haja vista que para o indexador implementado palavras como ”Informação” e ”informação” são distintas devido a diferença gramatical, apesar de possuirem o mesmo significado semântico.