

# Trabalho Prático 4 - Recuperação da Informação

Matheus Aquino Motta<sup>1</sup>

<sup>1</sup>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 assignment 4 of the subject Information Retrieval. The problem consisted in a follow-up to the assignment 3, where we needed to construct an inverted list with the text words found on a mini-collection of HTML documents. Now, similarly to the last assignment, our goal is to build an inverted list for a larger collection of 1000068 HTML documents, that doesn't necessarily fit in main memory, then the algorithm design has been changed in order to construct the inverted list and other additional files using external sorting techniques. Furthermore was provided an execution flag to load the inverted list dictionary to query for a given term information in the inverted list "efficiently".*

**Resumo.** *Nesse relatório iremos discutir brevemente a implementação do trabalho prático 4 da disciplina Recuperação da Informação. O problema consistia em uma extensão do trabalho prático 3, onde precisávamos construir uma lista invertida a partir das palavras encontradas no texto de uma mini-coleção de documentos HTML. Agora, de modo análogo ao trabalho anterior, o nosso objetivo consiste na construção de uma lista invertida para uma coleção de 1000068 documentos HTML, assim o algoritmo foi modificado de forma que conseguimos construir a lista invertida e arquivos complementares com o uso de técnicas de ordenação em memória externa. Além disso, foi disponibilizada uma tag de execução que carrega em memória principal o dicionário de acesso a termos da lista invertida construída, para a requisição "eficiente" de informações relevantes de um dado termo.*

## 1. Introdução

O problema proposto no trabalho prático 4 consistia em um acompanhamento do trabalho prático 3, no qual a partir de uma mini-coleção de documentos HTML deveríamos construir um indexador para os termos existentes no texto de cada página. De modo análogo ao trabalho anterior, no trabalho prático 4 foi desenvolvido um algoritmo para a construção de uma lista invertida para os termos de uma coleção de 1000068 documentos, onde diferentemente do projeto anterior o algoritmo desenvolvido deveria ser capaz de lidar com um volume de dados que não necessariamente cabe em memória principal.

Com isso, para a indexação apropriada dos documentos da coleção foi utilizado um método de ordenação externa, onde tomamos subconjuntos de documentos menores e realizamos sua indexação, para posterior união em uma lista única. Além disso, uma lista com o vocabulário da coleção, assim como informações relevantes de cada um dos termos únicos também foi criada. Onde ao final, um dicionário de acesso a dados de cada um dos termos únicos da lista invertida foi construído para a requisição "eficiente" de informações dos termos encontrados.

## 2. Implementação

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

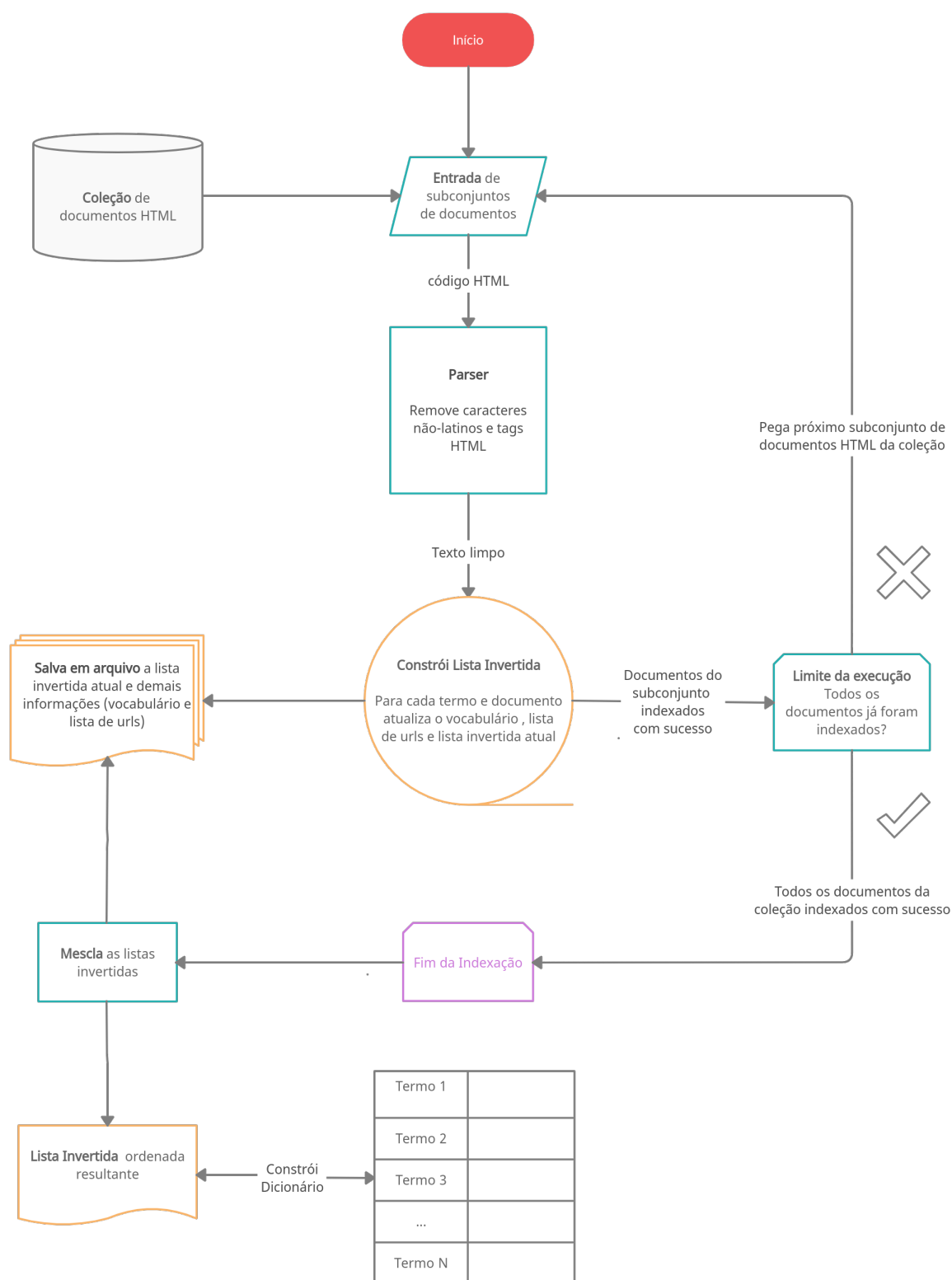


Figure 1. Fluxograma de funcionamento do Indexador.

Assim, para descrever o fluxo de execução e funcionamento de maneira apropriada precisamos entender as funções centrais do indexador: a análise do texto recebido realizada pelo *Parser*, a indexação das palavras encontradas no texto em memória externa e a posterior construção do dicionário de termos do vocabulário.

### 2.1. Análise textual, *Parser*

Assim como no trabalho prático 3 um problema central no desenvolvimento do algoritmo consistia em: tomado o conteúdo *HTML* de uma página, eliminar tags e trechos de código do documento para obter acesso efetivo ao conteúdo textual de cada página, e de forma equivalente esse problema foi tratado diretamente a partir do uso do *parser* da biblioteca *Gumbo*.

Além disso, embora com uso do *Gumbo Parser* tenhamos acesso ao texto limpo de cada um dos documentos, sabemos que as listas invertidas e demais informações são indexadas a partir dos termos únicos encontrados que compõem o vocabulário. Assim, de modo a tentar satisfazer a consistência gramatical e semântica dos termos indexados, de acordo com a Língua Portuguesa, caracteres de pontuação e marcação foram apropriadamente removidos e substituídos por espaços, assim como caracteres que não pertencem ao alfabeto latino e algumas de suas extensões.

Desse modo, garantimos a consistência gramatical das palavras indexadas, mantendo caracteres com sinais diacríticos, isto é, caracteres com sinais gráficos usados para alterar a fonética das palavras como: acentos graves, agudos e circunflexos, til, etc; e garantindo que *substrings* como "hotel", "hotel," e "hotel." encontradas fossem consideradas um termo igual único "hotel".

Com isso, ao fim obtemos uma *string* com o texto "limpo" de palavras devidamente separadas por espaços que serão apropriadamente utilizadas para a construção das listas invertidas.

Entretanto, embora a análise textual garanta a consistência gramatical das palavras, isto é, palavras gramaticalmente diferentes serão consideradas diferentes, o caráter semântico dos termos não é considerado. Dessa forma, palavras com o mesmo significado, mesmo que não escritas de forma gramaticalmente correta, serão consideradas palavras diferentes, como "informação", "INFORMAÇÃO", "Informação" e "Informacao", que embora potencialmente sejam semanticamente iguais são gramaticalmente distintas e portanto consideradas termos diferentes durante a indexação.

### 2.2. Indexador

Agora com o acesso apropriado aos termos de cada documento podemos realizar a construção da lista invertida e demais arquivos complementares, como o vocabulário da coleção, a lista de URLs e o dicionário as posições de cada termo na lista invertida final.

A lista invertida será composta por 3-uplas, onde o primeiro elemento será relativo ao identificador do termo, o segundo, o identificador do documento e por fim o terceiro, a posição em que o termo ocorre no documento. Assim, seja  $d$  o identificador do documento atual, percorrendo o texto, para cada termo  $t \in d$ , encontrado associamos à  $t$  um identificador  $id_t$  com o uso de uma tabela *hash*, que mapeia cada termo do vocabulário da coleção a um identificador inteiro único, adicionando a 3-upla  $(id_t, d, p)$  na lista invertida, onde  $p$  é a posição em que o termo ocorre em  $d$ .

Como já discutido, devido à potenciais limitações *hardware* a lista invertida da coleção será construída a partir de técnicas de ordenação em memória externa, onde iremos construir listas invertidas para subconjuntos de documentos da coleção, que podem ser armazenados na memória principal, que serão ordenadas e devidamente salvos em memória externa.

Isto é, seja  $C$  o conjunto de documentos HTML, iremos particionar  $C$  em  $2^k$  subconjuntos de documentos  $S$  de tamanho aproximado, tais que  $2^k|S| \approx |C|$ , os subconjuntos sejam mutuamente disjuntos,

$$S_i \cap S_j = \emptyset$$

para  $i \neq j$ , e que a união dos subconjuntos seja igual a coleção original

$$\bigcup_{i=1}^{2^k} S_i = C$$

Nesse sentido, iremos construir a lista invertida para cada um dos subconjuntos de acordo o processo descrito acima, e ao final da construção de cada uma delas iremos ordená-las em memória principal em ordem crescente de acordo com as 3-uplas  $(id_t, d, p)$ , respectivamente em relação ao  $id_t$ ,  $d$  e  $p$ . Dessa forma, ao fim teremos  $2^k$  listas invertidas ordenadas de acordo com os mesmo critérios, que podem ser visualizadas termo a termo por meio de sequências ou blocos de 3-uplas relativas a um determinado termo.

Assim, com as listas invertidas devidamente construídas e ordenadas iremos uni-las iterativamente tomando-as duas-a-duas, em um processo similar ao do *Merge Sort*. Isto é, inicialmente iremos contar com  $2^k$  listas que tomadas duas-a-duas serão unidas em uma nova lista resultante mantendo a ordenação relativa das 3-uplas, haja vista que iremos percorrer os dois arquivos linha-a-linha para satisfazer as limitações da memória externa. Com isso, ao fim iremos obter  $2^{k-1}$  listas também ordenadas com informações das listas iniciais. Desse modo, executando esse processo  $k$  vezes a partir da  $2^k$  listas originais iremos obter uma lista invertida final resultante ordenada para a coleção  $C$  como queríamos.

### 2.3. Dicionário

O último passo da execução principal consiste na construção de um dicionário de acesso às informações relativas a um determinado termo na lista invertida, haja vista que precisamos de uma forma minimamente eficiente para realizar alguma pesquisa para um dado termo. Como sabemos a lista invertida final pode ser visualizada por divisões em blocos ou sequências de 3-uplas relativas às ocorrências de um mesmo termo.

Partido dessa informação iremos percorrer a lista invertida linha-a-linha, verificando se o termo da linha se o  $id_t$  da linha  $l_i$  é igual ao  $id_t$  da linha  $l_{i+1}$ , caso sejam iguais seguimos para linha seguinte, caso contrário adicionamos no dicionário o intervalo de ocorrência do termo  $id_t$  passado e definimos um novo limite inicial para o novo termo encontrado.

Esses intervalos são tomados de acordo com a posição em *bytes* da linha em (em relação ao início do arquivo) que as informações de um determinado termo iniciam e terminam no arquivo, e pode ser obtido por meio da função da STL do C++ *tellg*. O que posteriormente permite que realizemos uma busca direta nas posições exatas em que um termo ocorre na lista invertida por meio de seus limites de início e fim no dicionário a partir da função *seekg* também da STL do C++.

## 2.4. Fluxo de execução

Entendidas as partes centrais do algoritmo, o fluxo de execução segue de maneira direta o processo de Indexação dos subconjuntos de documentos, união dos documentos e construção do dicionário.

Assim inicialmente tomamos os  $2^k$  subconjuntos  $S$  da coleção  $C$  um-a-um e para cada documento  $d \in S$  realizamos a o *parsing* do documento, adicionando a URL de  $d$  em um arquivo com a lista de todas as URLs da coleção e seus respectivos identificadores. Nesse sentido, para cada termo  $t$  encontrado no texto após a análise textual atualizamos o vocabulário da coleção, caso seja a sua primeira ocorrência, adicionamos  $t$  ao vocabulário associando a ele um identificador inteiro único  $id_t$  e inicializando o  $n_t$ , número de documentos em que  $t$  ocorre, como  $n_t = 1$ . Caso  $t$  já tenha ocorrido antes, associamos a  $t$  o seu respectivo  $id_t$  e se  $t$  ainda não ocorreu em  $d$  incrementamos o seu  $n_t$ . Por fim, adicionamos a tupla relativa ao termo  $t$  na lista invertida atual na forma  $(id_t, d, p)$  e seguimos para o próximo termo.

Realizando esse processo para cada subconjunto  $S \in C$  obtemos uma complexidade de tempo de execução para a construção das listas invertidas iniciais da ordem de

$$\mathcal{O}(2^k(|S|(P + \alpha|T|) + n \log n))$$

, onde  $P$  é o custo análise textual,  $T$  é o texto limpo obtido em cada documento e  $n$  o tamanho da lista invertida criada.

Esse resultado é obtido haja vista que para cada um dos  $2^k$  subconjuntos  $S$  temos aproximadamente  $|S|$  documentos, cujo o texto  $T$  será obtido através de um da análise textual, *parsing*, de custo  $P$ , e posteriormente  $T$  será percorrido com o auxílio de contadores linearmente, entretando devido às verificações realizadas em tabelas *hash* durante a varredura podemos obter uma variação  $\alpha$  no tempo de pesquisa, haja vista que o custo de acesso às tabelas *hash* utilizadas é em média  $\mathcal{O}(1)$ , totalizando em um custo da ordem de  $\mathcal{O}(P + \alpha|T|)$ . E por fim, para cada uma das listas invertidas criadas de tamanho  $n$ , temos um custo de  $\mathcal{O}(n \log n)$  para a ordenação das 3-uplas.

Com todas as listas devidamente criadas o próximo passo consiste na união das listas em um arquivo único, que como já discutido é baseado na ideia do *Merge Sort* e possui uma complexidade de tempo da ordem de  $\mathcal{O}(N \log k)$ , onde  $k$  é o número de iterações de união das  $2^k$  listas e  $N$  é o tamanho da lista invertida final. Esse resultado é obtido haja vista que para cada iteração de união percorremos todas 3-uplas da lista invertida linearmente linha-a-linha.

A partir da lista invertida final, a construção do dicionário é feita em tempo  $\mathcal{O}(N)$ , haja vista que esse processo consiste apenas em uma varredura com marcadores da lista invertida de tamanho  $N$ .

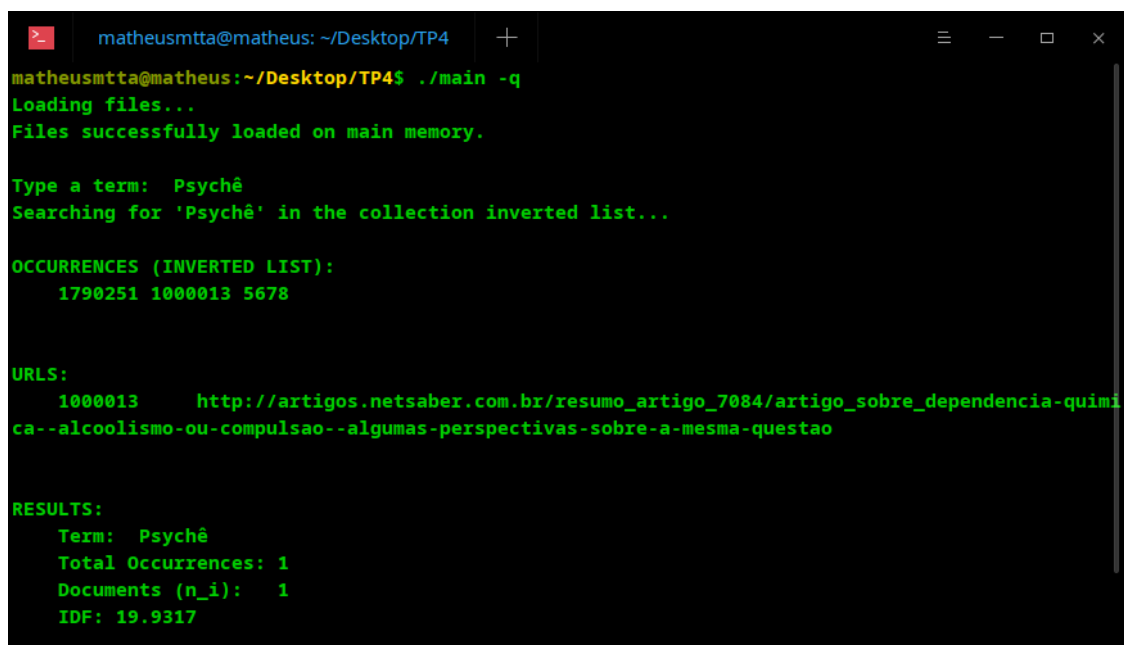
Por fim, o resultado do algoritmo será 4 arquivos:

- Lista Invertida, construída a partir de 3-uplas  $(id_t, d, p)$ , respectivamente o identificador do termo  $t$ , o documento relativo à ocorrência e a posição no documento em que  $t$  ocorre. Para todos os termos do texto da coleção.

- Vocabulário, contruído a partir de 3-uplas  $(\text{termo}, id_t, n_t)$ , respectivamente o termo  $t$ , o identificador numérico do termo e o número de documentos em que o termo ocorre. Para todos os termos únicos do texto da coleção.
- Lista de URLs, construída a partir de 2-uplas  $(id_d, \text{url})$ , respectivamente o identificador do documento e sua URL. Para todos os documentos da coleção.
- Dicionário, construído a partir de 3-uplas  $(id_t, \text{início}_t, \text{fim}_t)$  respectivamente o identificador do termo, e o limite inicial e final de suas 3-uplas de ocorrência na lista invertida. Para todos os termos únicos da coleção, isto é, o vocabulário.

## 2.5. Requisições

Agora com todos os arquivos devidamente contruídos, podemos realizar a requisição de um determinado termo na coleção de forma "eficiente", carregando o Dicionário e arquivos auxiliares de Vocabulário e Lista URLs em memória principal. Assim, podemos fazer a requisição das informações relativas a algum termo a partir dos dados de acesso presentes nesse arquivo, como pode ser visualizado no exemplo abaixo



```

matheusmtta@matheus: ~/Desktop/TP4
matheusmtta@matheus:~/Desktop/TP4$ ./main -q
Loading files...
Files successfully loaded on main memory.

Type a term: Psychê
Searching for 'Psychê' in the collection inverted list...

OCCURRENCES (INVERTED LIST):
  1790251 1000013 5678

URLS:
  1000013      http://artigos.netsaber.com.br/resumo_artigo_7084/artigo_sobre_dependencia-quimi-
ca--alcoolismo-ou-compulsao--algumas-perspectivas-sobre-a-mesma-questao

RESULTS:
  Term: Psychê
  Total Occurrences: 1
  Documents (n_i): 1
  IDF: 19.9317
  
```

Figure 2. Exemplo de requisição na Lista Invertida.

O exemplo acima é apenas ilustrativo, que tomamos um termo de identificador 1790251 que ocorre apenas uma vez na coleção no documento de número 1000013, na posição 5678. Além da ocorrência é mostrada a URL em que o termo ocorre e informações quantitativas do termo na coleção, como seu  $n_i$  (número de documentos em que o termo ocorre) e o seu  $IDF$ .

O tempo de requisição após o carregamento dos arquivos na lista invertida é da ordem do número de ocorrências do termo requisitado na lista, haja vista que percorremos todas as linhas do arquivo relativas ao termo dado, a partir das informações do dicionário.

## 2.6. 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 identificadores 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*. Ademais, para esse trabalho especificamente foi utilizada a biblioteca *rapidjson* para o *parsing* do arquivo *JSON Line* de documentos *HTML* disponibilizado para o trabalho.

## 3. Resultados

O algoritmo foi desenvolvido com base na coleção de documentos *HTML* para a construção da Lista Invertida e demais arquivos com informações acerca da coleção disponibilizada. O arquivo é um *JSON Line* composto por 1000068 linhas de 2-uplas (url, html), de aproximadamente 81.78 GB.

O algoritmo possui duas formas de execução, sendo elas

- \$ ./main -b para a construção dos arquivos.
- \$ ./main -q para a requisição de algum termo na lista invertida.

Para a construção da Lista Invertida, Dicionário e demais arquivos o algoritmo levou aproximadamente 3 horas 27 minutos e 58 segundos para executar.

```
60 DONE
61 DONE
62 DONE
63 DONE

Merging Inverted Lists on file...
Final Inverted List saved at 'output/invertedList.txt.'

Building dictionary on file...
Dictionary saved at 'output/dictionary.txt'.

Running approximate time 3h 27m 58s
matheusmta@matheus:~/Desktop/TP4$
```

Figure 3. Terminal fim da construção dos arquivos.

O algoritmo foi executado em uma máquina de 64 BITS com processador INTEL(R) CORE(TM) i5-6400 CPU @ 2.70GHz x 4 e memória RAM de 7.7 GB.

O tamanho dos arquivos de entrada e saída pode ser visualizado na tabela abaixo, onde podemos observar um ganho significativo de memória quando acessamos informações da lista invertida a partir do dicionário, vocabulário e lista de URLs.

	Coleção	Lista Invertida	Dicionário	Vocabulário	Lista de URLs
Tamanho	81.78 GB	12.02 GB	53.54 MB	33.78 MB	97.45 MB

Table 1. Tamanho dos arquivos de entrada e saída.

O vocabulário da coleção indexada conta com 1790353 termos únicos, que por sua vez possuem uma distribuição de *IDFs*, *inverse document frequency*, dada pelo gráfico abaixo.

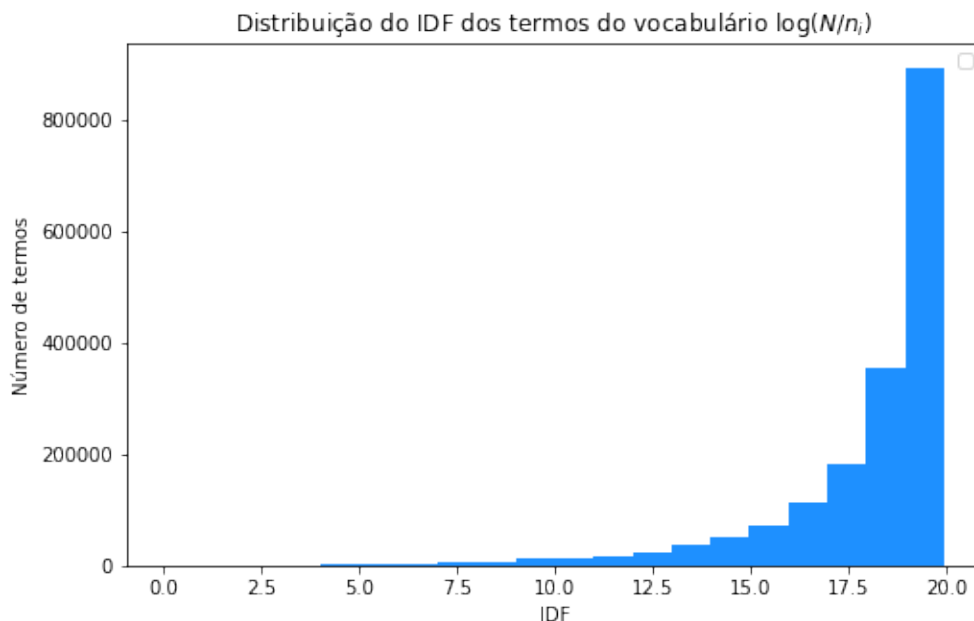


Figure 4. Gráfico de distribuição dos IDFs dos termos do vocabulário obtido.

Distribuição obtida através do cálculo do  $\log \frac{N}{n_i}$  de cada termo, onde  $n_i$  é o número de documentos em que o termo  $i$  ocorre e  $N$  o número total de documentos.

Além disso, podemos destacar dentre os resultados os termos mais frequentes, que são majoritariamente artigos e preposições da Língua Portuguesa, como pode ser visualizado pelo gráfico abaixo, assim como seus respectivos *IDFs*.

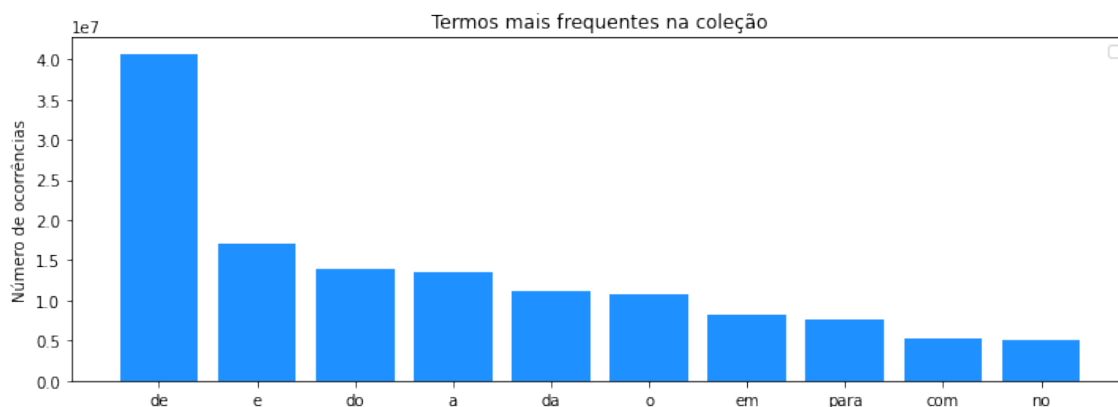


Figure 5. Termos mais frequentes da coleção.



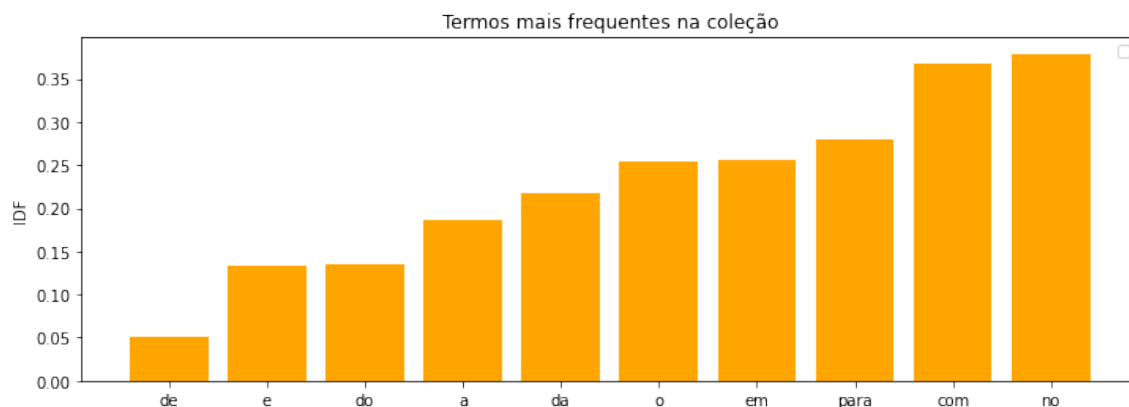


Figure 6. IDF dos termos mais frequentes da coleção.

#### 4. Conclusões

Nesse relatório discutimos as decisões de implementação, abordagens e resultados do desenvolvimento de um algoritmo para a construção de uma lista invertida para uma coleção de 1000068 documentos HTML. Além disso, para verificação de corretude do algoritmo e acesso a informação dos termos que compoem a lista, um sistema de requisições também foi implementado, a partir da construção de um dicionário de acesso à lista invertida e arquivos de auxílio para armazenamento de informações relevantes sobre os termos e documentos.

As maiores dificuldades encontradas centraram-se em como lidar com um volume de dados que superava a capacidade da memória principal do computador. Porém, tais dificuldades forma devidamente solucionadas e foi extremamente satisfatório visualizar o funcionamento do algoritmo e os resultados obtidos.

Além disso, é importante mencionar que otimizações podem ser feitas na forma em que os dados são armazenados na lista invertida para a diminuição no custo de memória, como por exemplo a compressão dos dados obtidos.