Health Search

Gerson Ferreira dos Anjos Neto

Graduando em Engenharia da Computação - Universidade Estadual de Feira de Santana (UEFS) - Feira de Santana - Bahia - Brasil

gersonferreiradosanjosneto@gmail.com

Resumo. *Este relatório tem como objetivo descrever o desenvolvimento de uma ferramenta de busca por termos em arquivos de texto, voltada para a varredura de grandes bancos de dados em hospitais. Os conceitos de Inverted Index e CLI – neste descritos – foram utilizados para garantir uma implementação simples e eficiente.*

1. Introdução

No que se diz respeito ao significado, o dicionário Merriam-Webster define a palavra *data* (dados) em três aspectos essenciais [1]:

- Informações factuais (*e.g.* medições ou estatísticas) abundante e facilmente disponíveis usadas como base para o raciocínio e a lógica;
- Informação emitida por um dispositivo ou processada;
- Informação emitida por um dispositivo ou órgão sensor que inclui informações úteis e irrelevantes ou redundantes e deve ser processada para ser significativa.

Diante de tais podemos afirmar que os dados estão diretamente ligados a troca de informações e/ou o acúmulo dessas, e esse potencial já era explorado pela humanidade desde os seus primórdios – por volta de 19.000 a.c. [2].

Na modernidade, os dados são de extrema importância e utilidade no âmbito da computação, podendo ser coletados de diversas formas e armazenados em diferentes formatos. A informação que eles representam alimentam os motores que nos movimentam tanto no campo digital como no social, ao passo que cada vez mais nos tornamos dependentes delas e da forma como são manipuladas [2].

Em face a esta crescente demanda, foi idealizado um sistema que fosse capaz de suprir as necessidades de consulta de informações de redes hospitalares. Para tal, os conceitos de *Inverted Index* e operação por *CLI* (*Command Line Interface*), foram utilizados para garantir a eficiência e simplicidade na utilização do mesmo. O produto proposto é capaz de percorrer um arquivo de texto – em formato ".txt" – e armazenar as palavras nesse contidas em uma estrutura que permita à sua eventual consulta.

O desenvolvimento dessa aplicação foi feito utilizando a linguagem de programação *Python* em sua versão 3.8.10 dentro da *IDE* (Ambiente de Desenvolvimento Integrado) *PyCharm Community* 2022. Todo o desenvolvimento e testes foram realizados dentro do *OS* (Sistema Operacional) *Ubuntu* 20.04.4 *LTS*.

2. Metodologia

Nesta seção será descrito o processo de criação da supracitada aplicação, desde a sua idealização até o *source code* (código-fonte). Os tópicos a seguir descritos, foram estruturados a fim de facilitar a compreensão do processo criativo.

2.1 Fundamentação Teórica

Primordialmente, faz-se necessário compreender os conceitos que foram utilizados durante o processo. A seguir, será feita uma breve descrição sobre as principais estruturas e ferramentas utilizadas durante o desenvolvimento.

A linguagem de programação *Python*, criada por Guido Van-Rossum em 1989, é uma linguagem de alto nível (*i.e.* sua sintaxe se aproxima mais da linguagem humana) onipresente capaz de ser utilizada nas mais diversas situações – independente da dificuldade das mesmas. Algumas das estruturas nativas utilizadas foram [3]:

- Estruturas Condicionais: Permitem o redirecionamento do fluxo de dados dentro do código, dado uma determinada condição. Os comando utilizados são *if, elif* e *else* (se, senão se, senão);

- Estruturas de Repetição: Também chamados de laços, são capazes de manter um determinado bloco em execução dado uma quantidade de ciclos (*for*) ou uma condição (*while*).

- Dicionários e Listas: Utilizadas para armazenar informações dentro do código. As listas armazenam os dados em sequência atribuindo automaticamente para cada um índice iniciado em 0, já nos dicionários é possível atribuir manualmente uma chave para cada informação armazenada;

- Funções: Permitem a definição de um determinado bloco (*i.e.* um porção do código) que poderá ser reaproveitado a qualquer momento dentro do código.

Ademais, foi feita a utilização das seguintes bibliotecas:

- *sys*: Fornece funções e variáveis usadas para manipular diferentes partes do ambiente em tempo de execução do *Python*, tais como: obter o caminho atual do arquivo, verificar o nome do sistema, etc [4].

- *os:* Permite a utilização de funcionalidades do próprio sistema operacional, tais como: a manipulação de arquivos e diretórios, navegação dentro do sistema de arquivos, etc [5].

2.2 Funcionamento Geral

Tendo definido os conceitos utilizados na idealização de tal aplicação, partirmos para a compreensão da lógica por trás de seu funcionamento. O sistema foi projetado visando atender os seguintes critérios do usuário:

- Poder indicar um diretório/arquivo para ser indexado e/ou removido;
- Permitir a buscar – rápida e eficiente – de um termo dentro do índice;
- Exibir, quando solicitado, todo o índice.

Além disso, toda a interação com o sistema deve ser feita por *CLI*. Dessa forma, a cada nova chamada do programa é necessário armazenar os dados da execução na forma de um *cache* que será lido na chamada seguinte – tudo isso automaticamente.

Aplicações baseadas em *CLI* (*i.e.* Interface de Linha de Comando) foram os primeiros meios de interação com computadores, sendo muito utilizados entre os anos 60 e 80. São manipuladas por meio de comandos na forma de texto dentro de um terminal ou *shell* [6].

No que tange a interação por linha de comando, a função *sys.argv* da biblioteca *sys* permite captar os argumentos e parâmetros passados juntos a chamada do programa.

Para atender aos critérios de manipulação de arquivos e diretórios, foram utilizadas funções da biblioteca *os* tais como:

- *os.path.isdir()* / *os.path.isfile()*: Permitem, respectivamente, a identificação de um diretório ou arquivo, dado o seu referente caminho;
- *os.path.basename()* / *os.path.dirname()*: Retornam, respectivamente, o nome de um arquivo ou o diretório onde esse está armazenado, dado o seu caminho;
- *os.walk()*: Permite a varredura dos diretórios e arquivos de um determinado local.

Visando garantir que a busca atenda aos requisitos estipulados, foi utilizada a estrutura de dados *Inverted Index* (Índice Invertido). Tal estrutura, como o nome sugere, é baseada em índices que fazem um mapeamento entre o conteúdo de um arquivo – seja números e/ou palavras – e o seu local em um banco de dados ou dentro de um arquivo.

Para construção de tal, as frases presentes em um texto são divididas em *tokens* (*i.e.* chaves, neste caso representadas pelas próprias palavras) e para cada um é atribuída uma referência ao seu documento de origem. Nesse processo são desconsideradas as palavras conhecidas como *Stop Words*, que não possuem peso semântico (*e.g.* tinhas, nosso, ela, está).

É possível ainda, realizar a passagem de palavras derivadas para a sua versão original (*e.g.* casarão → casa) ou até mesmo considerar sinônimos [7]. Todavia, para nossa aplicação, desconsideramos estes dois últimos métodos para garantir a simplicidade da implementação.

2.3 Entrada de Dados

Tendo esclarecido a lógica por trás do seu funcionamento, partirmos para o entendimento do processo de recepção de dados no sistema.

Como já foi dito, toda a interação com a ferramenta é feita por linha de comando (*i.e.* via terminal), portanto não está presente qualquer tipo de menu. Para cada operação que deseje realizar, o usuário deverá fazer a chamada do programa passando um parâmetro referente ao comando relacionado à operação que deseja realizar (*e.g.* "-a" ou "--add" → Adicionar um diretório/arquivo ao índice) e um complemento referente a dita operação, seja o caminho de um arquivo indexado e/ou removido ou um termo a ser buscado.

Esses comandos e complementos que o usuário deve fornecer são validados antes da execução, garantindo o bom funcionamento do programa. Para garantir um boa experiência ao usuário, é fornecido um menu de ajuda contendo a sintaxe de cada comando e o seu respectivo funcionamento.

Vale ressaltar que, além dos dados fornecidos pelo usuário durante a chamada, o *script* automaticamente faz a leitura do seu *cache* antes de qualquer execução, garantindo a recuperação dos dados referentes à última operação. Da mesma forma que ao fim de sua execução, o mesmo reescreve o *cache* com as suas informações atualizadas.

2.4 Manipulação dos Dados

Após o usuário fornecer os dados coerentes e referentes à operação que deseja realizar, o programa executa a tarefa solicitada como descrito no seguinte algoritmo de indexação de um diretório:

I. Em primeira instância, o programa realiza a leitura do *cache* – por meio de uma função – garantindo a recuperação das informações no caso de uma busca, ou nesse caso, a atualização do índice. Essas informações serão armazenadas em um dicionário;

II. Logo após, é feita a varredura do diretório indicado utilizando a função *os.walk()* e a indexação das palavras de cada arquivo por meio de um laço *for* que as passa por uma função de normalização e faz a contagem de todas;

III. Neste momento, são desconsiderados, além das *Stop Words* (que estão armazenadas em uma lista), números e qualquer tipo de pontuação.

IV. As palavras são armazenadas no *Index*, que – como dito – está estruturado como um dicionário, da seguinte forma:

   - InvertedIndex["Palavra"]["Diretório"] = {local_do_arquivo: quant_de_ocorrências};

- Dessa forma para cada chave/palavra no dicionário, há um outro dicionário menor contendo os diretórios indexados e este, por sua vez, contém um outro dicionário com os nomes dos arquivos e a quantidade de vezes que determinada palavra aparece.

- Ao fim da indexação, uma outra função é chamada para verificar se houve alguma remoção de arquivo e/ou diretório do armazenamento, removendo o mesmo do *Index*.

V. Por fim, o *cache* é reescrito por meio de uma função com as informações atualizadas do *Index*.

2.5 Saída de Dados

É de se esperar que em um programa, após o processamento dos dados, os resultados sejam exibidos para o usuário, mas na nossa aplicação isso nem sempre é o caso. Baseando-se na Regra do Silêncio (“*Silence is Golden”)* presente nos sistemas *UNIX* desde a sua criação [8].

A aplicação em questão não apresenta retorno evidente para as funções de adição e remoção de arquivos do *Index*, a menos que ocorra algum erro de sintaxe ou o caminho para o arquivo/diretório informado esteja incorreto, não exista ou esteja corrompido.

Para os outros casos, os resultados respeitam o seguinte formato:

- Mostrar o *Index*: Exibe palavra por palavra, o nomes e locais dos arquivos onde estão presentes e a quantidade de ocorrências dessas;

- Buscar Palavra: Exibe a quantidade de arquivos encontrados, os seus respectivos nomes e locais onde estão armazenados ordenados decrescentemente pela quantidade de ocorrências.

3. Resultados e Discussões

Por fim, nesta seção será feito um breve resumo de como o programa deve ser utilizado.

Ao tentar executar o programa sem passar nenhum argumento, é apresentado ao usuário uma mensagem de ajuda contendo instruções sobre a utilização do mesmo Essa mesma mensagem também pode ser exibida caso o usuário passe como opção os parâmetros “-h” ou “--help”.

Para cada operação que deseje realizar o usuário deve escolher a opção referente

e o seu respectivo complemento, como na relação a seguir:

- “-h” ou “--help” → Não requer complementos, exibe a mensagem de ajuda;
- “-a” ou “--add” → Requer o local do arquivo/diretório a ser adicionado ao *index*, respeitando as normas do sistema. Não apresenta retorno, a menos que haja erro de sintaxe;
- “-r” ou “--remove” → Requer o local do arquivo/diretório a ser removido do *index*, respeitando as normas do sistema. Não apresenta retorno, a menos que haja erro de sintaxe;
- “-s” ou “--searchIndex” → Requer uma palavra a ser buscada no *index*, retorna os nomes e locais dos arquivos como descrito na seção anterior. Caso a palavra não seja encontrada o usuário também é informado;
- “-S” ou “--showIndex” → Não requer complementos, exibe todo o *index*.

Caso o usuário utilize alguma das funções incorretamente ou chame alguma inexistente, as mensagens de erro de sintaxe serão exibidas exemplificando a utilização correta do comando.

4. Conclusão

Por fim, conclui-se que o produto atende a todos os critérios estipulados. Testes foram feitos para garantir que o mesmo suporte a manipulação de grandes volumes de dados no menor tempo possível, mantendo a qualidade das informações.

Para futuras implementações, poderá ser feito o aprimoramento da utilização do *Inverted Index*, permitindo realizar uma indexação e busca mais rápidas e eficientes ou até mesmo poder buscar por mais de uma palavra. Não obstante, poderá ser feita a implementação de uma versão com *GUI* (*i.e.* uma interface gráfica) para facilitar a utilização por usuários inaptos a operar por linha de comando.

5. Referências

[1] Merriam-Webster Dictionary “Data”, https://www.merriam-webster.com/dictionary/datautm_campaign=sd&utm_medium=serp&utm_source=jsonld, Junho de 2022.

[2] Stephanie Shen (2020) “What is Data?”, https://towardsdatascience.com/what-is-data-ade94b37204a, Junho de 2022.

[3] Swaroop C. H. (2008) “A Byte of Python”, Version 2.1

[4] Python Official Documentations “os - Miscellaneous operating system interfaces”, https://docs.python.org/3/library/sys.html, Junho de 2022.

[5] Python Official Documentation “sys – System-specific parameters and functions” https://docs.python.org/3/library/os.html, Junho de 2022.

[6] Oyetoke Tobi Emmanuel (2018) “Building Beautiful Command Line Interfaces with Python”, https://codeburst.io/building-beautiful-command-line-interfaces-with-python-26c7e1bb54df, Junho de 2022.

[7] Ihor Kopanev (2019) “A brief explanation of the Inverted Index”, https://medium.com/@igorkopanev/a-brief-explanation-of-the-inverted-index-f082993f8605, Junho de 2022.

[8] Henrique Marques Fernandes (2019) “The Rule of Silence (Silence is Golden) – Linux”, https://marquesfernandes.com/en/technology/the-rule-of-silence-silence-and-gold-linux/ , Junho de 2022.