Guilherme Barboza Mendonça
João Antonio Oliveira Pedrosa
Larissa Aline Fernandes Vieira

PSE: Python Search Engine

Ferramenta de busca de documentos em Python

Trabalho desenvolvido no segundo período de Ciência da Computação para a disciplina de Programação e Desenvolvimento de Software II

Belo Horizonte

2019

1. Introdução

A ferramenta de busca da Google realiza, em média, 3.5 bilhões de pesquisas por dia, retornando milhões de resultados ordenados por relevância e, em muitas das vezes, em menos de um segundo, para cada busca, o que só é possível com a união de um hardware extremamente poderoso com um algoritmo de busca altamente eficiente. Em vista dessa necessidade, nos foi proposto o desenvolvimento de uma máquina de busca, capaz de receber uma coleção de documentos de texto e computar quais desses se relacionam melhor com uma determinada *query* de pesquisa.

2. Objetivo

O objetivo do sistema a ser construído é, a partir de um conjunto de documentos e um conjunto de palavras dado, dizer quais documentos são mais compatíveis com a busca. Para tal, é construído um índice invertido: Um índice padrão relacionaria o documento às palavras que ele contém. O índice invertido, por sua vez, associa individualmente cada palavra aos documentos onde ela aparece. A partir desse índice, poderemos construir um sistema de coordenadas, de tal forma que cada documento pode ser interpretado como um vetor no espaço gerado pelo sistema.

Uma query de consulta também está associada a um vetor desse espaço, logo, a máquina de busca pode comparar o vetor de um documento com o da busca e encontrar a sua similaridade: quanto menor o ângulo entre eles (ou, mais fácil de calcular, quanto maior o cosseno desse ângulo), melhor é a relação entre a busca e esse documento. Por fim, com todas as similaridades calculadas, podemos relacionar os melhores documentos com a consulta realizada.

3. Implementação

3.1 Linguagem e Módulos

O programa foi desenvolvido na linguagem Python em sua versão 3.7.3. Os módulos utilizados foram:

- *NumPy*
- *re — Regular expression operations*
- *time — Time access and conversions*
- *pathlib — Object-oriented filesystem paths*
- *argparse — Parser for command-line options, arguments and sub-commands*

- *os — Miscellaneous operating system interfaces*
- *glob — Unix style pathname pattern expansion*

3.2 Classes

Foram criadas duas classes. A classe Indice_Invertido e a classe consulta.

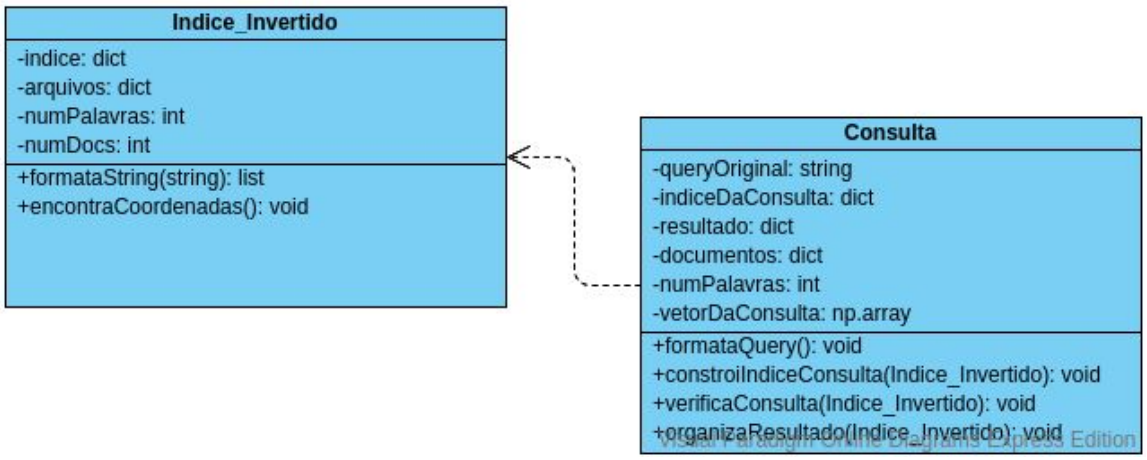

Figura 1- Diagrama de Classes

Em Python, um dicionário é uma variável que associa chaves a valores, o equivalente a um *map* em *C++*.

A classe Indice_Invertido tem, como seu atributo principal "indice", um dicionário. As chaves do dicionário são todas as palavras presentes nos textos e os valores são também dicionários. Estes últimos, por sua vez, tem como chaves o nome de cada documento e, como valor, o número de vezes em que a palavra aparece no respectivo documento. Já o atributo "arquivos" é um dicionário que associa o nome de cada arquivo a um identificador arbitrário gerado pelo algoritmo, que será usado no final do processo para relacionar os documentos corretos à consulta feita.

Nesta classe também estão as "coordenadas", uma matriz do *NumPy* na qual cada linha representa uma palavra do banco de dados, cada coluna representa um documento e os valores são a importância da palavra em relação ao documento, de acordo com a equação "W(dj, Px)" dada no PDF de especificação do trabalho prático. Os demais atributos são inteiros que guardam o número de palavras e o número de documentos do índice.

A classe Consulta, por sua vez, é responsável por manipular e encontrar a resposta final: A partir de uma consulta, é construído um novo índice invertido da *query*, porém, relacionado com o índice original, resultando, em termos geométricos, em um vetor que representa a posição da pesquisa no espaço vetorial dos documentos. A partir dessas informações, o algoritmo pode calcular a similaridade entre os vetores dos documentos com o da consulta. É interessante citar que uma das razões para a escolha da linguagem Python para se desenvolver o projeto foi a sua grande ênfase ao cálculo matemático (em especial, álgebra linear), portanto, com o auxílio do *NumPy*, o cálculo da similaridade entre dois vetores é:

$$sim(u, v) = cos(u, v) = \frac{<u \,.\, v>}{|u| \,.\, |v|}$$

Ou seja, a similaridade entre dois vetores é o cosseno do seu ângulo, que pode ser calculado fazendo o produto vetorial deles, dividido pelo produto de suas normas. Quanto maior a similaridade, melhor a aproximação do documento com a consulta, logo, uma vez calculadas todas as similaridades, basta ordenar os documentos da maior para a menor similaridade e, finalmente, apresentar esses dados para o usuário.

3.3 Entrada do Programa e Arquivo Principal

O caminho para o banco de dados é fornecido durante a compilação do programa. Já o arquivo principal do programa "Maquina_de_Busca.py" possui apenas a interface de interação com o usuário e a exibição dos resultados. É perguntado ao usuário qual a busca que ele deseja fazer e os dados são enviados para serem processados pelas classes. Após isso, são exibidos os resultados e uma interface na qual é possível visualizar os arquivos encontrados e fazer novas buscas.

4. Testes de Unidade

Os Testes de Unidade "isolam" partes do programa, a fim de mostrar que estão funcionando corretamente. Dessa forma, foram desenvolvidos testes unitários para todas as funções do sistema, considerando diversos cenários e situações.

Para tal feito, o *assert*, que verifica uma determinada condição em tempo de execução, foi usado. Já que, somente se a condição for falsa, o "AssertionError" acontece, parando o programa, foram feitos avisos personalizados, explicando melhor sobre onde o erro deve estar (há uma boa precisão, uma vez que fazem parte de algum teste unitário).

Ademais, podem ser encontrados testes para construtores e funções com ou sem retorno. Para os construtores, foram verificados os valores iniciais de cada uma das variáveis após a inicialização da variável e, para as funções com retorno, foram construídos vários testes contemplando diferentes contextos, comparando os valores retornados com os esperados.

5. Conclusão

Como aprendizado retirado do trabalho, podemos citar 3 pontos principais. O fato do trabalho ser em grupo, forçou os integrantes a aprenderem a desenvolver códigos juntos e a usarem ferramentas como o *GitHub*, habilidades altamente requisitadas no mercado de trabalho e no desenvolvimento de software em geral. A complexidade do algoritmo serviu como motivação para que os integrantes aprendessem a otimizar o código, o que resultou em uma versão final do software 20 vezes mais eficiente do que a versão inicial. E a escolha da linguagem Python para o desenvolvimento da aplicação gerou uma aproximação dos integrantes com uma linguagem com a qual eles não possuíam intimidade.

Ao final do desenvolvimento do software, pudemos compreender que o objetivo de uma máquina de busca é praticamente o mesmo que o de vários outros softwares: encontrar a melhor dentre uma grande gama de possíveis soluções para um problema. Porém, o que destaca esse tipo de algoritmo em relação a outros é o seu alto grau de complexidade, delicadeza e importância, afinal, bilhões de consultas são realizadas todo dia e, paralelamente, milhões são investidos anualmente para manter e desenvolver melhores algoritmos de busca.

Além disso, concluímos que a validade de uma máquina de busca gira em torno ter três eixos principais: Corretude, espaço e tempo. O algoritmo deve conseguir extrair e interpretar o máximo tanto do documento quanto da consulta: sintaxe, semântica, erros

ortográficos, palavras isoladas, frases compostas, o peso de cada um desses fatores, entre outros, visando fornecer a melhor solução possível.

Ademais, a tecnologia de hardware desenvolvida até hoje, por mais que seja relativamente poderosa, ainda tem capacidade de armazenamento limitado, logo, é importante que a máquina de busca seja capaz de gerenciar memória e alocar recursos da forma mais dinâmica e eficiente possível, pois, em cenários reais, bilhões de arquivos deverão ser processados e manipulados. Podemos ainda estender esse conceito a outros custos, tais como energia e manutenção.

Por fim, o terceiro eixo - talvez o mais importante deles - é o tempo levado para encontrar a solução ideal. Novamente, a tecnologia atual tem eficiência limitada, porém, uma das principais características da sociedade moderna é a sua constante exigência por rapidez, fazendo com que a maioria das atividades, das mais cotidianas e simples às mais complexas e desafiadoras, sejam realizadas no menor tempo possível. Portanto, seguindo essa necessidade, os algoritmos de busca devem ser modelados de tal forma que possam fornecer soluções coerentes em um tempo aceitável.

Assim, conclui-se que o trabalho, assim como a disciplina de PDS2, trouxe vários novos conhecimentos para os integrantes do grupo.

6. Referências

Informações:

- https://www.internetlivestats.com/google-search-statistics/
- https://www.ateomomento.com.br/uml-diagrama-de-classes/
- https://www.visual-paradigm.com/

Módulos do Python:

- https://numpy.org/
- https://docs.python.org/3/library/time.html
- https://docs.python.org/3/library/re.html
- https://docs.python.org/3/library/pathlib.html
- https://docs.python.org/3/library/argparse.html

- https://docs.python.org/3/library/os.html
- https://docs.python.org/2/library/glob.html