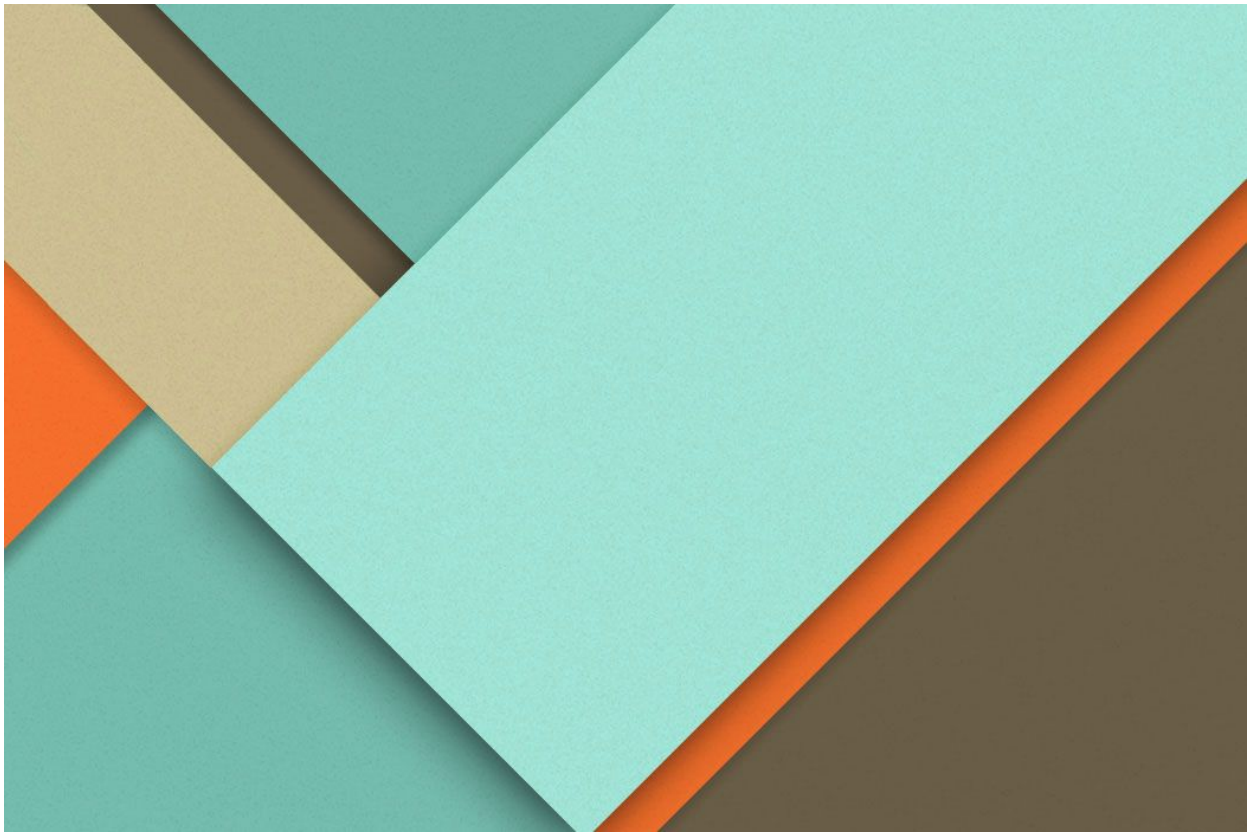


Máquina de Busca

22/11/2019

Nome: Thiago Henrique Moreira Santos

Matrícula: 2019007074

Belo Horizonte,Minas Gerais

Introdução

A ideia de criar uma máquina busca foi proposta na disciplina de Programação e Desenvolvimento de Software II da Universidade Federal de Minas Gerais, onde nós, os alunos, tínhamos que criar uma máquina de busca.

A seguir farei uma breve introdução do conteúdo que será apresentado no decorrer do projeto. Quando o assunto é um sistema de busca ao meu ver o principal aspecto a ser levado em conta é a velocidade da consulta, por conta disso um dos pilares utilizados nesse projeto foi o **Índice Invertido.** O índice invertido, de forma simplificada, é uma estrutura de dados que mapeia uma palavra a um conjunto de documentos nos quais ela aparece, essa estrutura de dados é extremamente eficiente para nosso propósito(buscas rápidas), embora ela traga consigo uma ineficiência quando formos inserir, retirar ou deletar o banco de dados, ela nos permite fazer consultas de forma muito acelerado e também é fácil de ser implementada, comparado a outras formas.

Com índice invertido em mãos estaremos aptos a realizar consultas no nosso banco de dados. Nesse projeto coloquei “duas formas” de consultas, uma que chamei de **Consulta Simples** e outra que chamei de **Consulta Exata.** A primeira é mais abrangente, dada uma entrada do usuário o sistema retorna para ele os documentos que aparecem pelo menos uma palavra da entrada(seguindo um processo de rank que será apresentado em breve). O segundo é um método um pouco mais específico, ele retorna os documentos que contém a entrada “exata” que foi escrita para consulta, ambas as formas serão apresentadas de forma mais profunda mais a frente.

E por fim nosso objetivo é desenvolver uma forma de classificar os documentos e retornar o que mais se aproxima da consulta. Neste projeto utilizei o método **tf-idf** que basicamente trata da frequência de termos e da frequência inversa dos documentos em relação a algum documento. Por fim, para calcular a similaridade da consulta e o banco de dados utilizaremos o **Similaridade Cosseno**, que basicamente realiza o produto escalar entre dois vetores e divide pelo produto da norma dos dois, isso vai resultados no cosseno do ângulo entre esses dois vetores (veremos que esses vetores em questão são o vetor consulta e o vetor do documento). Com a similaridade cosseno nosso sistema terá plenas condições de retornar os documentos de maior similaridade com a consulta.

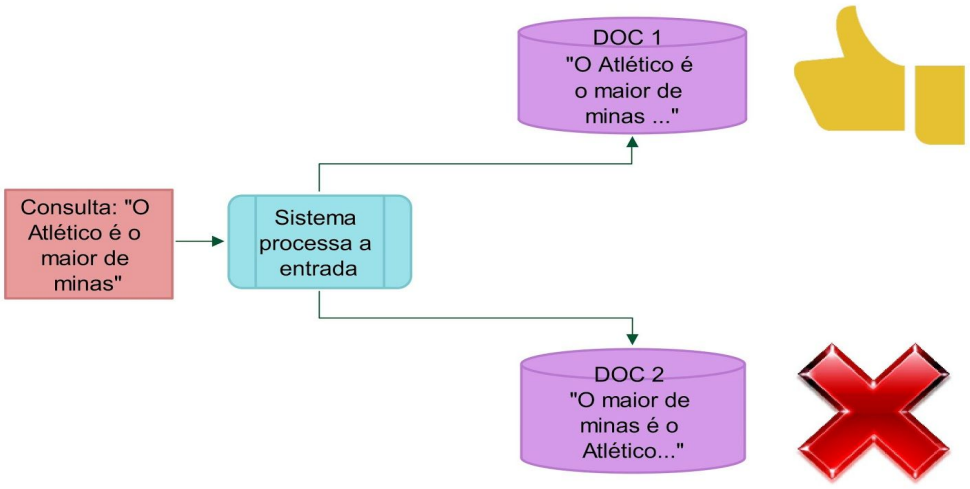
Nota de rodapé: Todas as estruturas foram implementadas seguindo o “Paradigma de Programação Orientada a Objetos”, essa documentação não vai entrar em detalhes quanto à essa forma de modelagem do projeto.

Implementação

Nesta seção vou expor, com maior profundidade, como foi implementado os recursos apresentados na introdução.

Como dito no início, o primeiro passo para a criação da máquina de busca é a implementação de um **índice invertido**. Embora essa seja a primeira coisa impactante que temos que implementar ainda há algo a se fazer antes disso, temos que fazer uma "limpeza" no nosso banco de dados. A nossa base de dados (nesse caso, arquivos) provavelmente vai conter palavras coisas como: palavras com acento, pontuações de caráter linguístico(vírgula, ponto final e etc), letras maiúscula e minúsculas. Para fim práticos vamos "ignorar" todas essas especificidades, dessa forma temos que fazer uma "filtragem" dos documentos antes da criação do índice invertido.

Feito isso estamos prontos para implementar o índice invertido. Como dito anteriormente, o objetivo principal é mapear palavras aos documentos que elas aparecem, mas também disse na introdução que implementaria uma "busca exata", para isso além de mapear a palavra com o documento que ela aparece vou precisar da posição que ela se encontra dentro do documento, o objetivo disso é apresentado no esquema abaixo:



A ideia aqui é que o "match" só ocorra se documento possuir a consulta na "exatamente" como a consulta foi solicitada (basicamente mesma ordem).

Dado essa explicação criamos então uma estrutura de mapeamento de palavras para suas respectivas posições no documento. Basicamente construímos uma estrutura

Nota de rodapé: Todas as estruturas foram implementadas seguindo o "Paradigma de Programação Orientada a Objetos", essa documentação não vai entrar em detalhes quanto à essa forma de modelagem do projeto.

de “chave e valor” com o dicionário do python onde a chave é a palavra e o valor é uma lista com as posições iniciais da palavra em um dado documento. Vamos utilizar esse retorno para criar um dicionário aninhado, que será nosso índice invertido final. No fim teremos um ‘dicionário externo’ que terá como chave as palavras distintas, ou seja o vocabulário da aplicação, e terá como valor um outro dicionário, que tem como chave os documentos em que as palavras aparecem e como valor a posição dessas palavras nos documentos. Segue um exemplo, de um trecho do índice invertido final.

```
'atletico': {'doc3.txt': [1]}, 'maior': {'doc3.txt': [4]}, 'de': {'doc3.txt': [5]}, 'minas': {'doc3.txt': [6]}
```

Tendo terminado o índice invertido completo em mãos podemos começar a pensar como funcionarão as consultas, como citei anteriormente vão haver dois tipo de pesquisas: “Consulta Simples” e “Consulta Exata”.

Consulta Simples: Assim como na criação do índice invertido há um passo prévio na implementação da consulta e ele é o mesmo do passo prévio do índice invertido, temos que “limpar” as palavras. A ideia geral da implementação é a seguinte: sempre vou buscar segmentar a entrada por palavras e fazer múltiplas chamadas de uma função que verifica se a palavra se encontra nas chaves do índice invertido final(lembrando que as chaves do dicionário equivalem a uma coisa como uma lista chamada vocabulário,eu não criei essa lista pois o python já retira as palavras repetidas, até porque não faz sentido ter chaves repetidas, e isso já é tudo o que eu preciso. Então optei por consultar as chaves ao invés de um vetor/lista chamado vocabulário). Essa função que citei anteriormente recebe *uma única palavra* e retorna uma lista com os documentos que ela aparece. Para finalizar a consulta simples basta unir as listas que são retornadas por cada palavra, isso é feito em outra função.

Consulta Exata: Essa forma de consulta é de certa forma parecida com a anterior, mas precisa levar em conta a ordem das palavras da consulta/entrada. De início é tudo muito semelhante à consulta simples, limpamos as palavras, enviamos cada palavra para a função ... A diferença começa agora, vamos criar uma lista que contém as listas com as posições das palavras nos documentos em que elas aparecem, o objetivo é encontrar uma intersecção entre essas listas de posições. Um argumento plausível poderia ser, “Uma intersecção? Como você vai conseguir isso? Até onde eu sei duas palavras distintas não podem ocupar o mesmo índice.”. Isso de fato é verdade, mas como já dizia o professor Thiago “tem uma roubadinha que podemos fazer”, brincadeiras à parte de fato tem uma forma bem interessante de verificar se há a intersecção ou não, mostrarei o funcionamento com um exemplo:

Imagine a seguinte consulta: `"O Atlético é o maior de Minas."`

e que temos as seguintes posições para as palavras:

Nota de rodapé: Todas as estruturas foram implementadas seguindo o “Paradigma de Programação Orientada a Objetos”, essa documentação não vai entrar em detalhes quanto à essa forma de modelagem do projeto.

(essas linhas pretas podem ficar desproporcionais, mas não afetam a ideia central do exemplo)

'O':[52, 89, 120, 123, 163] → 'O':[59, 96, 127, 130, 170,] soma 7

'Atletico':[2, 121, 330, 440] → 'Atletico':[8, 127, 336, 406] soma 6

'é':[3, 8, 12, 15, 38, 49, ..., 122, 430] → 'E':[8, 13, 17, 45, 54, ..., 127, 435] soma 5

'o':[52, 89, 120, 123, 163] → 'O':[56, 93, 124, 127, 167] soma 4

'maior':[5, 14, ..., 124, 164] → 'Maior':[8, 17, ..., 127, 167] soma 3

'de':[23, 45, ..., 125] → 'De':[25, 47, ..., 127] soma 2

'Minas':[126] → 'Minas':[127] soma 1

(lembrando que essas são as listas das posições das palavras no documento, e uma pesquisa real essa análise é feita em *uma lista que contém todas as listas de posições das palavras*)

O "macete" que citei a pouco consiste em somar n, n-1, n-2, ..., 1 a cada lista de posições, onde n é o número de palavras da consulta. Depois de todas as iterações as listas ficarão como descritas depois da seta ('-->').

Sempre que há uma intersecção (em outras palavras, *palavras aparecem na mesma ordem da consulta*) esse algoritmo vai conseguir encontrar a intersecção, nesse caso nas posições [120, 121, 122, 123, 124, 125, 126]. (OBS: Este é um documento genérico, criado apenas com o objetivo de exemplo).

Classificando Documentos: Essa é a parte em que classificamos os a relevância do documento de acordo com a consulta, iremos utilizar o tf-idf, que é um método de classificação que leva em conta a frequência da palavra e a frequência inversa do documento em relação a palavra. Para fazer isso vamos representar todos os documentos como um vetor de tamanho n, onde n é o número de palavras distintas que existem na nossa base de dados, no nosso caso corresponde às chaves do índice invertido final. Não

Nota de rodapé: Todas as estruturas foram implementadas seguindo o "Paradigma de Programação Orientada a Objetos", essa documentação não vai entrar em detalhes quanto à essa forma de modelagem do projeto.

utilizaremos apenas a frequência das palavras (tf), pois apenas ela não é um parâmetro bom o suficiente, pois as vogais (separadas) aparecerão muito mais que as palavras compostas, o que indicaria que as vogais são mais relevantes e não é isso que queremos. Para isso utilizamos a *frequência inversa dos documentos*. A frequência inversa de um documento (idf) é dada pela fórmula log(N/nx), onde N é o número de documentos e nx é a quantidade de vezes que a palavra em questão apareceu naquele documento (a função logaritmo aparece para pois o "acerto" de N/nx seria muito "rude"). Fazemos o cálculo dessa quantidade para todos os documentos e todas as palavras. Para calcular a coordenada W do documento, basta fazer o produto tf x idf de cada documento. Também fazemos isso para consulta, transformamos ela em um vetor e calculamos o tf-idf para as palavras que lá aparecem.

Calculando a similaridade entre as consultas e os documentos: O último passo que falta é calcular o quão próxima cada documento está da consulta e apresentar o resultado em ordem que o documento mais similar a entrada seja mostrado primeiro e assim sucessivamente. Como dito na introdução vamos utilizar a similaridade cosseno, que nada mais é do que o produto interno entre dois vetores dividido pelo produto de suas respectivas normas, com isso vamos obter o cosseno do ângulo entre eles. Com o cálculo da similaridade cosseno teremos "pontuações" para cada documento de acordo com uma consulta específica e poderemos exibir os documentos de modo que o documento com maior pontuação, isto é maior proximidade, seja exibido primeiro e assim por diante.

Funcionamento da classificação: Primeiramente calculamos o tf-idf para cada palavra e construímos o vetor dos documentos com tamanho n, onde n é a quantidade de palavras distintas. Depois disso, no segundo passo, obtemos a consulta e "processamos" ela de modo que neste estágio ocorre a busca e temos o retorno de uma lista contendo os documentos que apresentam as palavras ou frases que estavam na consulta. Em um terceiro momento, também vetorizamos as consultas (para podermos comparar com os vetores de documentos), também terá tamanho n. No quarto passo, utilizamos a similaridade cosseno para obter uma pontuação para cada documento que basicamente diz o quão próximo aquele documento está da consulta, e está pronto basta retornar os documentos na ordem das pontuações.

Observações Importantes:

Professor, quando o cálculo da similaridade cosseno é igual para dois documentos, infelizmente não pude fazer a apresentação dos documentos no mesmo espaço. Por exemplo, na apresentação que o senhor fez no pdf, o resultado

era mostrado da seguinte forma, ['d4.txt', 'd1.txt', 'd3.txt, d4.txt'], ou seja, os documentos 'd3.txt' e 'd4.txt' são representados em uma mesma string. Eu infelizmente, não pude fazer dessa forma pois a minha função que calcula o vetor para um documento

Nota de rodapé: Todas as estruturas foram implementadas seguindo o "Paradigma de Programação Orientada a Objetos", essa documentação não vai entrar em detalhes quanto à essa forma de modelagem do projeto.

recebe o nome do documento, se eu fizer dessa forma o que seria 'doc3.txt' passaria a ser 'doc3.txt, doc4.txt' e obviamente esse documento não existe. Eu poderia sim na função splitar a entrada que chega em duas strings, mas isso não resolver o meu problema pois minha função

também necessita do tamanho(quantidade de documentos) que existem, se eu fizer dessa forma, nesse exemplo, ficaria n-1 documentos. É impossível "trackear" **cada caso de** quantidades de documentos então, a minha saída é expressa da seguinte forma, uma simples ordenação dos documentos baseados nas suas pontuações, sem colocar documentos com mesma pontuação na mesma string.

O método de pesquisa que foi pedido no pdf, eu o chamei de “consulta_simples”.

Eu tinha feito o trabalho em um repositório privado e coloquei tudo em um público no último dia.

Conclusão

Este foi trabalho muito interessante em que vimos da melhor forma o “tradeoff” que tanto foi comentado no curso. Abrindo de mão de velocidade na inserção, remoção e até mesmo edição no “banco de dados” (no nosso caso foram apenas documentos), temos um ganho absurdo de velocidade pesquisa, que para o objetivo do TP vem bem a calhar. Mas é claro que não é algo tão determinístico assim, você não simplesmente escolhe abrir mão de um artifício computacional para obter melhoria em outra apenas por vontade, temos que conhecer métodos que nos proporciona isso e é exatamente por isso que utilizamos conceito clássicos como: índice invertido, classificação por similaridade cosseno, utilizar o método do tf-idf dentre outros. Enfim, foi um trabalho extremamente interessante, em que pude ter um conhecimento mais próximo do funcionamento de um buscador.

Nota de rodapé: Todas as estruturas foram implementadas seguindo o “Paradigma de Programação Orientada a Objetos”, essa documentação não vai entrar em detalhes quanto à essa forma de modelagem do projeto.