Universidade Federal do Amazonas - UFAM
Instituto de Computação - ICOMP
Programa de Pós Graduação em Informática - PPGI
Disciplina: Tópicos em Recuperação de Informação.
Professor: Davi Fernandes.
Aluna: Luísa dos Reis e Silva. **Matrícula:** 2150363

Relatório Trabalho Prático

Link GitHub: https://github.com/LuisaReis07/TPRI_2015_2.git

Objetivo do Trabalho:

O objetivo principal desse trabalho é implementar uma máquina de busca para uma coleção de documentos utilizando o modelo vetorial como função de recuperação de informação. Então, ao se executar uma consulta, o que se deseja ter como retorno são os documentos que melhor atendem a resposta da consulta.

Implementação:

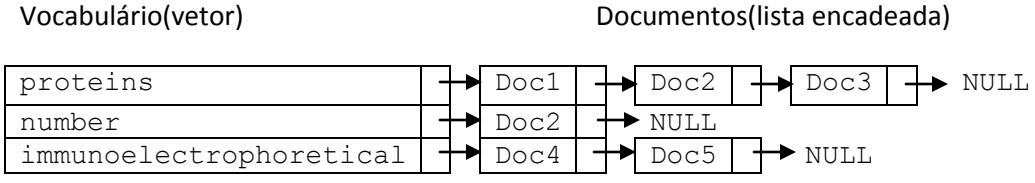
O trabalho foi implementado na linguagem de programação C++, por possuir mais recursos e funções que a linguagem C. As principais bibliotecas bibliotecas utilizadas foram *"math.h"*, para realizar cálculos matemáticos; *"time.h"*, para calcular tempos de execução; *"iostream"*, para leituras dos arquivos.

Para desenvolver o modelo vetorial precisamos de calcular o tf (freqüência do termo em cada documento), idf, frequencia inversa do documento, o peso do termo no documento, o vetor correspondente ao documento no espaço vetorial e, por fim, a similaridade de cada documento com a consulta.

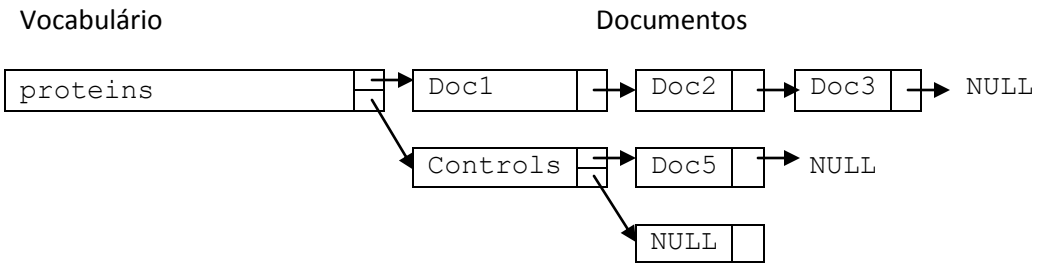
Inicialmente, o código lê os arquivos com as informações dos documentos e armazena essas as informações em um hash que representa o vocabulário (utilizado para o calculo do modelo vetorial). Foram excluídos do vocabulário as stopWords, que são termos que se repetem com muita freqüência em qualquer documento e podem influenciar negativamente no cálculo do modelo vetorial. Além disso, os termos também são normalizados, retirando acentos, espaços em branco e etc. Essas funções estão no código com os nomes *"normalizaTermo"* e *"retiraStopWord"*, respectivamente.

O vocabulário deve armazenar as informações: tf, idf e peso dos documentos. Ele foi estruturacomo como um hash, onde o a chave representa um termo e ela aponta para os documentos onde são feitas referencias aos termos.

Exemplificando:



Se ocorrer colisão, será criada uma lista encadeada de termos. O hash ficará como segue:



Universidade Federal do Amazonas - UFAM
Instituto de Computação - ICOMP
Programa de Pós Graduação em Informática - PPGI
Disciplina: Tópicos em Recuperação de Informação.
Professor: Davi Fernandes.
Aluna: Luísa dos Reis e Silva. **Matrícula:** 2150363

Os structs de termos é o *"Termo"* e o de documentos é o *"Documento"*. O vocabulário está como variável global *"vocabulario"*.

A função *"LeArquivo(Arquivo)"* le os arquivos e os indexa armazena no vetor de vocabulário, já fazendo a contabilização do tf e idf. Após a indexação, o programa calcula o peso de cada termo em cada documento com função *"calcularPeso()"*.

Outra informação importante para calcular a similaridade é o valor do vetor normal do documento. Essa informação é calculada na função *"calcularVetorDoc()"* e os valores são armazenados no vetor *"vetDocs"*, conforme o struct *"VetorDocInfo"*. Para facilitar a pesquisa. A posição do vetor em que o documento foi armazenado equivale ao RecordNumber do documento.

É feito um vocabulário para os termos da query semelhante ao dos documentos (com a diferença de que não precisa adicionar as informações sobre o documento). A função *"LeQuery(arqQuery)"* executa a leitura do arquivo de query e encaminha para a indexação nesse vocabulário (termosQuery). Esse vocabulário é reiniciado ao final de cada consulta.

Por fim, com todas as informações necessárias, é calculada a similaridade de todos os documentos com a consulta pela função *"calculaSimilaridades()"*. Depois de calculadas as similaridades, nessa mesma função, são selecionados os top 10 documentos com a maior similaridade (que é a informação que queremos).

Os resultados são apresentados na tela de execução, onde a primeira linha representa o numero da consulta e a segunda linha apresenta os documentos similares, cada um deles separados por um tab. No final também é apresentado o tempo de execução do algoritmo.

Tutorial de Compilação e Execução:

O projeto possui dois arquivos: tp_ri.h, contendo os structs utilizados no programa; e Main_tp_ri.cpp, contendo o código principal. Para executar o programa, é só deixar os dois arquivos na mesma pasta e rodar o Main_tp_ri.cpp.

O programa irá pegar os arquivos em uma pasta com o nome "cfc" contendo os documentos e o arquivo de query, então recomenda-se colocar os arquivos em uma pasta com esse nome, ou então alterar o nome das variáveis "Arquivo" (para os arquivos com os documentos) e "arqQuery", para o arquivo com consultas.

Executando o arquivo principal, ele irá apresentar na tela do terminal as etapas de execução do código que o programa está percorrendo e terminará mostrando o tempo de execução do programa.

Resultados Obtidos:

Os experimentos foram realizados em um notebook com processador Intel Core i3 (2,4GHz, 3MB cache, 4GB Ram) e Sistema Operacional Windows 8.1.

O sistema está apenas apresentando os documentos retornados pela consulta. Com essa informação, sabemos que o tempo de processamento para as 100 consultas foi e 2.88 segundos.