# Trabalho Prático 3
# Indexador
# Recuperação de Informação

**Daniel Ferreira Abadi**[1]
**2018088062**

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brasil

## 1. Introdução

O problema abordado consiste em construir indexador na linguagem de programação C++ utilizando parte de nossa coleção coletada pelo trabalho prático 2. O indexador deve construir um índice invertido, que nos foi apresentado durante as aulas. E tal processo deve caber na memória principal. O tamanho da mini-coleção escolhida foi de 3096 páginas, sendo as primeiras coletadas.

## 2. Implementação

Para começar o processo de indexação é necessário utilizar uma biblioteca que limpe os HTMLs coletados e, como não nos foi dada nenhuma biblioteca para utilizar, foi escolhida a Gumbo, desenvolvida pela empresa Google.

### 2.1. Limpeza de dados

Como dito anteriormente, foi utilizada a biblioteca Gumbo, que nos provê vários exemplos de uso, incluindo a limpeza total dos elementos HTML, CSS e JavaScript de documentos. A função utilizada para limpar os dados é chamada de ”cleantext”, que deriva de um exemplo passado, como dito anteriormente. Esse método retorna uma ”String”contendo todo o conteúdo sem as marcações utilizadas na WEB.

Dentro da função citada anteriormente é chamado um método para limpar caracteres indesejados, sendo os que estão fora da língua portuguesa, os números e as pontuações (foi mantido o hífen, visto que o mesmo é de grande importância para a língua portuguesa). Há também a conversão de todos os caracteres para o minusculo, visto que letras maiúsculas são diferentes das minusculas de acordo com a tabela Unicode, podendo ocasionar erros. Para fazer tal limpeza foi necessário a utilização da estrutura ”Wstring”e suas funções, e também a tabela Unicode.

### 2.2. Construção do índice

Para a construção do índice que nos foi passado, o mesmo seria uma estrutura ”Hash”cujo elemento chave é uma ”String”e o valor é uma estrutura do tipo Lista contendo dois números inteiros, representando o número do documento e a quantidade de vezes que a palavra da chave aparece no documento, e uma lista de inteiros, que são as posições nas quais aquela palavra ocorre.

Para facilitar a manipulação de toda essa estrutura, foi necessário criar uma estrutura de dados auxiliar, chamada ”struct_aux”, que contém uma estrutura do tipo Lista

que possui uma estrutura ”Pair”. Na primeira posição dessa estrutura também há um ”Pair”que guarda o número de documentos e o número de ocorrências da palavra naquele documento, e a segunda posição do ”Pair”externo é uma lista de inteiros que guarda as posições das ocorrências. Foi escolhido utilizar duas estruturas ”Pair”aninhadas pois é de mais fácil manipulação do que uma estrutura do tipo ”Tuple”, que seria a mais lógica de usar nesse caso. Portanto, a estrutura ”Hash”citada acima foi construída de forma que cada chave possua uma estrutura auxiliar contendo seus dados.

A estrutura auxiliar criada conta com alguns métodos, sendo o mais relevante a função ”add”, que recebe dois inteiros, sendo o número do documento e a posição na qual a palavra associada ocorreu. Tendo esses dados, o método percorre a Lista interna da estrutura procurando pelo número do documento. Caso ache o número do documento, é adicionado um ao número de ocorrências e a lista recebe a nova posição da ocorrência. Caso o número não seja encontrado, então é criada uma nova posição na lista com os números passados, inicializando o número de ocorrências com um.

### 2.3. Detalhes adicionais

O programa recebe um arquivo passado como parâmetro para fazer a limpeza do texto, e após a limpeza, esse texto é salvo em um arquivo gerado chamado ”out.txt”, que depois é reaberto por uma função chamada ”read_doc”, que pega palavra por palavra, chamando um método ”add_hash”que recebe uma ”String”, sendo essa a palavra, e um número inteiro, indicando a posição que a palavra ocorreu no documento. Essa função verifica se a palavra está na estrutura ”Hash”, caso esteja então apenas é chamada a função ”add”, citada na subseção anterior. E se não existir aquela palavra, então é criada uma nova posição no ”Hash”contendo a palavra como chave e é chamada a função ”add”.

O programa também criar um arquivo chamado ”data.txt”, que guarda o número de documentos, o tamanho de todo o índice invertido em Kbytes, as palavras que estão no ”Hash”seguidas pelo número de documentos que as contêm e a média do número de aparições nos documentos.

Para poder utilizar o programa é necessário que a lista de documentos tenha a palavra ”newdoc”antes de cada documento novo, essa foi a forma escolhida para conseguir distinguir os documentos. O arquivo que armazena as páginas deve conter apenas as páginas, sem os respectivos endereços WEB.

A contagem das posições é considerada apenas pelo tamanhos das palavras armazenadas, desconsiderando-se espaços entre as palavras

## 3. Exposição de dados

Foram utilizados 3.061 documentos, o objetivo era utilizar apenas 3 mil, mas esses 62 adicionais não fazem diferença. O tamanho de todo o índice invertido é de 6.818 Kbytes, contando-se o tamanho das palavras adicionadas e os inteiros que se referem ao número do documento, a quantidade de vezes que a palavra ocorreu no mesmo e as posições. O tamanho da coleção sem os tratamentos citados acima é de 416 MB e após o tratamento ela tem o tamanho de 27 MB. A quantidade de termos diferentes encontrados foi de 88.932. Mais detalhes como número de documentos que as palavras ocorrem e a média de aparições estão no arquivo ”data.txt”enviado junto com o código.

## 4. Conclusão

Foi construído um índice invertido, como o que nos foi apresentado em uma das aulas durante o curso. O mesmo tem como componentes uma estrutura ”Hash”com ”String”como chave e uma estrutura auxiliar como valor, que contém uma estrutura interna do tipo Lista que contém um ”Pair”, que contém um ”Pair”de inteiros na primeira posição e uma lista de inteiros na segunda posição.

Para fazer uma boa contagem de palavras foi necessário superar dificuldades como o não reconhecimento de caracteres especiais como ”á”e ”ç”, a diferenciação entre letras maiúsculas e minusculas e a grande dificuldade em converter tudo para minusculo, visto que grande parte dos métodos não conseguiam converter os caracteres especiais citados acima.

Para trabalhos futuros envolvendo caracteres especiais fica o novo conhecimento adquirido sobre a tabela Unicode e a estrutura ”wstring”. Para o próximo trabalho utilizando-se a coleção que nos será provida de um milhão de páginas, será necessário diminuir a complexidade da estrutura utilizada para armazenar os valores, bem como a remoção de escrita em um documento de texto, que faz a mediação entre a limpeza de dados e a construção do índice.