UNIVERSIDADE FEDERAL DE MINAS GERAIS

JOÃO PEDRO RIBEIRO JUNHO

MARCELO SARTORI LOCATELLI

MÁQUINA DE BUSCA

Trabalho Prático

Belo Horizonte – Minas Gerais

2019

1-Introdução:

O projeto desenvolvido constitui-se principalmente de um mecanismo de busca e recuperação de dados a partir de uma dada base de dados. O sistema é construído de uma maneira que são lidas e convertidas, para uma forma sem letras maiúsculas e sem símbolos especiais, todas as palavras dos documentos desejados, que depois são armazenadas na memória do computador juntamente com os respectivos documentos nas quais elas se encontram formando o chamado índice invertido desses arquivos, que mapeia palavras a documentos.

A partir disso, o programa utiliza essas informações para realizar buscas nesses documentos, calculando a frequência, isto é, o número de vezes que uma palavra aparece, de cada palavra em cada documento, e também o idf de cada palavra, que é dado pelo logaritmo da razão entre o número total de documentos e o número de documentos nos quais a palavra apareceu.

Utilizando dessas duas características das palavras, o sistema calcula, a partir delas, a coordenada de cada palavra relativa a um plano que contém todos os vetores resultantes. Essas coordenadas são posteriormente comparadas entre si para determinar aquela que mais se assemelha aos termos presentes na busca realizada pelo usuário. Essa comparação é dada pela similaridade, ou seja, o cosseno, dos documentos à pesquisa, isso é: o produto interno das coordenadas de cada documento da pesquisa com as coordenadas do que foi pesquisado, dividido pela norma do vetor que representa o documento multiplicado pela norma do vetor que representa a pesquisa.

Utilizando esse método, também chamado de *cosine ranking*, o programa consegue, então, ordenar os documentos a partir do número de similaridade, sendo que maiores números representam documentos que mais se assemelham à pesquisa, isto é, documentos que tem mais chance de ser aquilo desejado pelo usuário.

Por fim, o programa imprime uma tabela que mostra as posições dos documentos relativos à pesquisa, expondo ao usuário onde ele deve olhar primeiro para tentar encontrar aquilo que ele deseja.

2-Implementação:

2.1 Interface principal:

Ao abrir o programa, algumas escolhas são apresentadas ao usuário. São elas:

1-*Adicionar arquivo:*

Ao selecionar essa opção, é pedido que o usuário forneça o nome de um arquivo, que será adicionado ao índice.

2-*Remover arquivo:*

Ao selecionar essa opção, é pedido que o usuário forneça o nome de um arquivo, que será adicionado ao índice.

3-*Consultar arquivos do índice:*

Ao selecionar essa opção, os nomes de todos os arquivos que compõem o índice invertido são apresentados ao usuário.

4-*Fazer uma busca:*

Ao selecionar essa opção, é pedido que o usuário forneça os termos a serem buscados. Depois que ele os insere, o resultado da busca é apresentado.

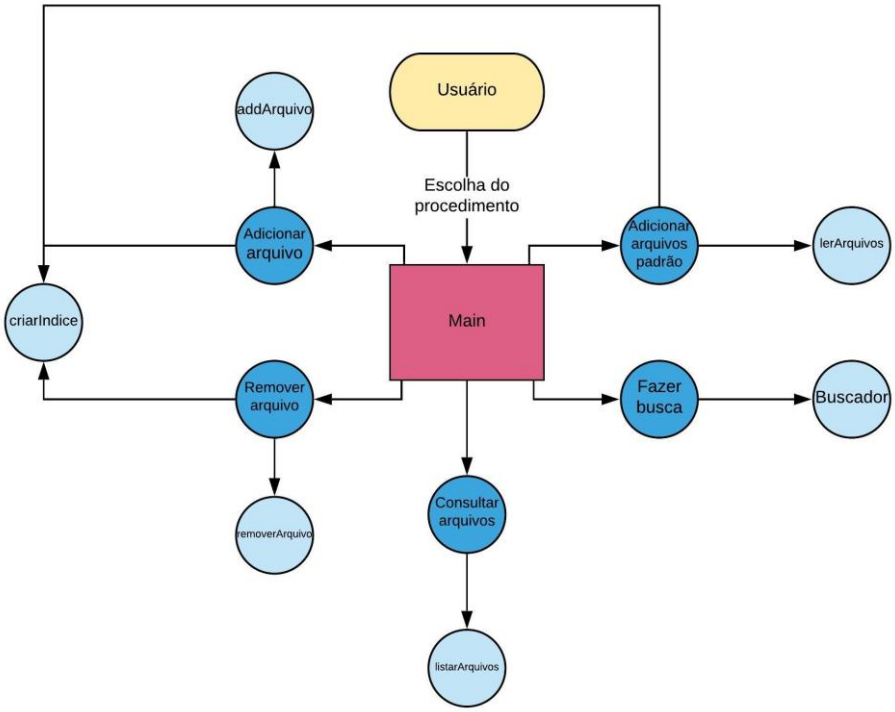
5-*Adicionar os arquivos padrão:*

Ao selecionar essa opção, todos os arquivos da pasta “20-newsgroups” são adicionados ao índice.

6-*Sair:*

Ao selecionar essa opção, o programa é encerrado.

Assim que o usuário encerra o uso de uma das opções, pergunta-se a ele se ele deseja continuar ou sair do programa. Caso deseje continuar, a mesma tela inicial é exibida.



2.2 Índice invertido:

O índice invertido foi representado em uma classe chamada indiceInvertido. Para calculá-lo, o programa percorre cada palavra de cada documento, já criando um mapa com a frequência de cada palavra em cada documento. Nesse mapa o primeiro elemento é uma *string* e o segundo elemento é um mapa de *string* e int. Essa estrutura foi usada, pois é necessário armazenar-se documento, palavra e número de repetições.

Durante esse mesmo processo, é criado também o arqPalavras, um mapa que relaciona cada palavra do vocabulário dos documentos aos arquivos em que elas ocorrem. Nesse mapa o primeiro elemento é uma *string* e o segundo elemento é um set de strings, que contém o nome de todos os documentos que contém as palavras. Nesse caso, foi usado o set para evitar que documentos repetidos sejam associados a uma palavra, impedindo, por exemplo, que um documento apareça várias vezes caso ele possua várias palavras iguais.

2.3 Algumas observações importantes:

O construtor da classe percorre os arquivos registrados em enderecos.txt e indexa eles em tempo de execução, caso já não haja o arquivo indice.txt, que será

explicado posteriormente. Para adicionar ou remover arquivos de enderecos.txt, usa-se as funções addArquivo e removerArquivo, respectivamente. Note que essas funções não recriam o índice em tempo de execução. Isso foi feito para evitar que, em caso de se adicionar muitos arquivos de uma só vez, o índice seja repetidamente recriado, gerando desperdício de tempo. Caso deseje-se recriar o índice, em tempo de execução, depois de adicionar-se arquivos, deve-se usar a função criarIndice. Essa função recria os mapas, de forma semelhante ao construtor. Além disso, ela armazena esses mapas em dois arquivos de texto: freq.txt e indice.txt, que armazenam a frequência e o índice, respectivamente.

Para representar esses dados nos arquivos de texto, utilizou-se a seguinte estratégia:

No freq.txt, para cada arquivo imprime-se o endereço do arquivo, seguido de uma alternância entre palavras e frequências. Assim que as palavras e frequências de um documento acabam, é impresso um ponto de exclamação, para ficar evidente que a próxima linha já é um outro endereço de arquivo.

Já no indice.txt, imprime-se uma palavra, seguida do endereço de todos os documentos que contém ela. Ao final dos documentos, é impresso um ponto de exclamação, para ficar evidente que a próxima linha já é outra palavra.

Isso funciona bem, visto que símbolos especiais não são considerados palavras pelo índice, assim não há o risco de o programa interpretar uma palavra como separador e vice-versa.

Para ler essas informações do arquivo para o programa, em uma execução posterior, basta usar o construtor.

2.4 Coordenadas:

Segundo a fórmula para o cálculo das coordenadas, é necessário primeiro encontrar tf, isto é, a frequência, e idf, isto é a importância de uma palavra em um dado documento, seguindo a formula:

W (dj,Px) = tf (dj ,Px) × idf(Px),

Encontrar a frequência é simples, dado que todas as frequências já são anteriormente armazenadas na estrutura frequências.

O idf, por sua vez, é encontrado por meio da fórmula:

idf (x) = log(N /nx) ,

onde N é o número total de documentos e Nx é o número de documentos nos quais a palavra está presente. A própria classe do índice invertido já armazena automaticamente quantos arquivos estão sendo pesquisados utilizando o int nArquivos. Nx, por sua vez, é obtido a partir do tamanho do *set<string>* mapeado a uma dada palavra em arqPalavras, visto que esse set representa quantos documentos apresentam essa palavra.

Com isso, são obtidos todos os dados necessários para calcular a coordenada completa de todos os documentos do vocabulário do acervo. Essa coordenada é armazenada na estrutura coordenadas, que é um *map<string,map<string,double>>*, pois isso permite uma associação do tipo:

coordenadas [documento][palavras] = w(dj,px),

que pode ser interpretado como um vetor:

coordenadas[documento] = (P^1,..,Pi,..,Pn)

coordenada relativa
à palavra 1 em dj

De modo que ao se escrever coordenadas[documentos][Pi], se acessa a coordenada Pi do vetor.

Além disso, é necessário calcular as coordenadas das palavras da própria pesquisa, para isso foi utilizado um *map<string,double>* Wq, visto que nesse caso só precisa-se associar uma palavra à sua coordenada pois existe apenas um vetor pesquisa.

Para que se crie uma coordenada por palavra, existe uma função separarPalavras, que recebe a *string* da pesquisa e devolve um vector com uma palavra (já convertida para os padrões da pesquisa, sem caracteres e símbolos especiais) ocupando cada posição.

Essa variável é somente criada quando necessário para se economizar espaço na memória ram.

2.5 Cosseno:

A função responsável por encontrar o cosseno recebe Wq, que foi obtida anteriormente e utiliza isso, juntamente com as coordenadas dos documentos para calcular o cosseno entre o documento e pesquisa, isto é, a similaridade entre cada documento e cada pesquisa. Isso é feito de acordo com a formula fornecida:

$$sim(d_j, q) = \cos(\theta) = \frac{\sum_{i=1}^{t}(W(d_j, P_i) \times W(q, P_i))}{\sqrt{\sum_i W(d_j, P_i)^2} \times \sqrt{\sum_i W(q, P_i)^2}}$$

Para armazenar esses números foi usado um *map<string,double>* pois ele permite associar a chave relativa ao documento à sua similaridade com a pesquisa.

O cálculo do cosseno no sistema, é realizado por documento, o que permite que após cada similaridade ser calculada, seja limpado todas as coordenadas armazenadas relativas àquele arquivo, o que deixa o programa consideravelmente mais lento, mas ao mesmo tempo foi a solução encontrada para impedir um gasto exorbitante de memória, que estava causando erros na execução do programa.

2.6 Buscador:

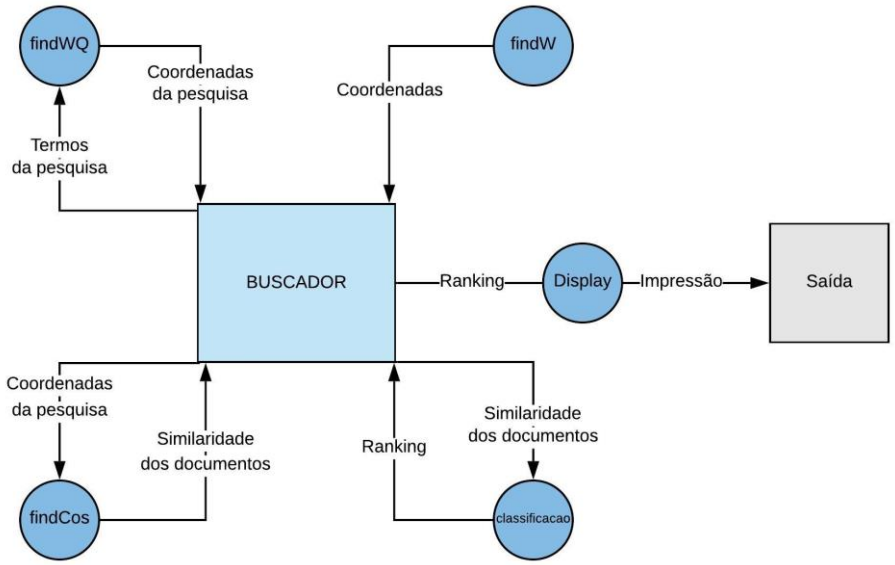
O buscador é a principal função do sistema e da classe índice invertido e é o responsável por chamar em sequência todas as funções responsáveis por encontrar as coordenadas, passando esses dados para a função que encontra a similaridade entre os documentos e a pesquisa, por fim classificando os resultados e imprimindo-os na tela. Caso o número de arquivos ou o tamanho do vocabulário dos arquivos seja muito elevado, essa função pode ser bem lenta.

Para realizar essa classificação, são utilizados 2 *vectors* criados em uma função específica chamada classificação. Um desses *vectors* recebe o valor das similaridades(valor), enquanto o outro recebe o nome dos documentos que tem tal similaridade(ordem).

As posições de um valor e um documento associada a ele são as mesmas nos vetores, o que permite que ao ordenar valor pelos números contidos nele, ordene-se simultaneamente a ordem dos documentos, montando a classificação final.

Além disso, ao se fazer essa ordenação, para garantir que documentos com igual similaridade ocupem as mesmas posições, a função, caso um valor seja igual ao outro, concatena as *strings* de ordem dessas posições, adicionando “ e “ entre elas.

A última função que é chamada nessa classe é a função display, que a partir dos dados obtidos anteriormente realiza a impressão da tabela de classificação na tela.



2.7 Leitura do dataset:

Existe também um arquivo chamado leitor_arquivos, que visa à leitura e incorporação ao índice invertido de todos os arquivos do *dataset* de teste. A única função desse arquivo é o procedimento lerArquivos, que recebe um indiceInvertido como parâmetro. Essa função percorre todas as pastas e arquivos do *dataset*, e adiciona eles ao índice recebido por meio da função addArquivo. Para isso, é usado um dirent. Note que no inicio dessa função há uma *string* com o endereço do *dataset*. Sendo assim, é primordial que a pasta “20_newsgroups” esteja no mesmo diretório do executável.

3-Conclusão:

Com os métodos descritos acima, um resultado satisfatório foi alcançado. Ao testar o programa com diversos arquivos, percebeu-se que o resultado correto foi obtido. Todos os documentos foram devidamente ordenados.

Entretanto, com o uso de bancos de dados maiores, percebeu-se certa lentidão na função *findCos*. Contudo, o custo computacional das outras funções foi suficientemente eficiente. Para bancos de dados com menos arquivos, a execução se deu de forma impecável.

Com isso, pode-se concluir que, embora o tempo de execução não seja o ideal para enormes bancos de dados, o sistema obteve êxito, pois mesmo nessas situações ele é capaz de, após algum tempo, realizar a classificação conforme os padrões requisitados pela proposta de trabalho.