

**UNIVERSIDADE FEDERAL DE MINAS GERAIS**  
**Engenharia de Controle e Automação**

**PROGRAMAÇÃO E DESENVOLVIMENTO DE SOFTWARE II**  
**Trabalho Prático: Máquina de Busca**

Alunos: Augusto Pena Araújo	2018014158
Gabriel Alves da Silva	2018014115
Stéfani Bemfica Soares	2018014190
Professor: Thiago Ferreira de Noronha	

## SUMARIO

Capa.....	1
Sumário.....	2
Introdução.....	3
Implementação.....	3
Normalização das palavras.....	3
Índice Invertido.....	5
Determinação de Coordenadas.....	7
Ranking Cosseno.....	8
Teste de Unidades.....	8
Conclusão.....	9
Referências Bibliográficas.....	9

## 1. Introdução

O trabalho prático consiste em três partes. Inicialmente, serão fornecidos alguns arquivos como parâmetros. Entretanto, não se sabe, a princípio, o conteúdo desses arquivos, que podem conter palavras com letras maiúsculas, minúsculas, com acentos, números, pontuação, etc. Considerando-se que o propósito do trabalho prático consiste em simular uma máquina de buscas, deve-se garantir que esse arquivo de entrada seja convertido em outro que possua, apenas, letras minúsculas a fim de que as palavras sejam encontradas corretamente e sem que variações levem a um resultado diferente daquele almejado. Deve-se notar, ainda, que esse procedimento deve ser realizado em cada arquivo de entrada. Por exemplo, se forem fornecidos dois arquivos de entrada que possuem apenas uma palavra, “SoftWare” e “SofTWaRE”, a busca pela palavra “software” deve identificar que essa palavra está presente em ambos arquivos.

Em seguida, implementamos a estrutura do índice invertido. O índice invertido é o mecanismo chave do programa, responsável por varrer os arquivos, lendo palavra por palavra e ordená-las, havendo assim, a necessidade de normalizar as palavras com a retirada dos caracteres indesejados descritos no parágrafo anterior. Máquinas de busca mais complexas levam em consideração inclusive a localização exata de cada palavra encontrada no arquivo. Porém, neste trabalho, realizaremos até a etapa de identificação de quantas palavras iguais à pesquisada existem em cada arquivo. A composição do índice é definida por vocabulário e ocorrências, ou seja; a partir desse método, relaciona-se uma determinada palavra ao(s) documento(s) onde essa determinada palavra pode ser encontrada.

A última parte do índice é definida pela análise da implementação. Usamos de testes, por enquanto manuais, para verificar o bom funcionamento do módulo. Dessa forma, pedimos a entrada da palavra a ser buscada e de acordo com os resultados devolvidos analisamos se fazem ou não sentido e, se fazem sentido, analisamos se os documentos retornados realmente apresentam a palavra e com a frequência calculada.

## 2. Implementação

### 2.1 Normalização das palavras

Primeiramente, declaramos a estrutura main. Após a devida inicialização, o primeiro passo a ser executado, de acordo com a introdução, é fazer as alterações necessárias no(s) arquivo(s) de texto onde será(ão) realizada(s) a(s) consulta(s). A modificação apenas será realizada se o caractere que está sendo lido no momento não for uma letra minúscula. Sendo assim, consulta-se a Tabela ASCII a fim de

analisar quais são os valores das letras a serem mantidas. A tabela e os valores que permanecerão após a exclusão estão marcados a seguir:

## ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	\$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

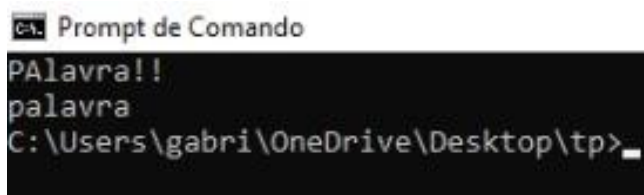
```

9
10 // Transforma as palavras em minusculo e retira os caracteres especiais
11 void minusculo (string& s){
12     for(int i=0; i<s.size();i++){
13         if(s[i]>='A' && s[i]<='Z'){
14             s[i]+=32;
15         }
16         else if(!(s[i]>='a' && s[i]<='z')){
17             s.erase(i);
18         }
19     }
20 }
21

```

A função responsável por transformar as letras maiúsculas em minúsculas está representada acima. Dentro do for, primeiro verifica-se se o caractere está dentre os caracteres maiúsculo. Se sim, o caractere é convertido para seu correspondente minúsculo. Se não, verifica-se se é um caractere especial. Se sim, esse caractere é excluído. Se nenhum dos if's for executado, o caractere já está normalizado.

Para perceber que esse procedimento está sendo realizado corretamente pelo algoritmo, criou-se uma impressão do texto de saída, onde a palavra já deve estar normalizada:



```
C:\> Prompt de Comando
PALavra!!
palavra
C:\Users\gabri\OneDrive\Desktop\tp>
```

## 2.2 Índice Invertido (Indexação)

Considerando o que foi explicado durante a introdução, para a implementação de um índice invertido foi utilizada uma estrutura que armazenará a palavra normalizada e o nome dos arquivos que a contém, indexando cada palavra com os seus respectivos documentos. Nesta etapa, cada palavra recebe uma indexação para posteriormente podermos associar essa indexação de cada palavra de uma busca e de fato imprimir os documentos mais relevantes para a busca.

A seguir, deve-se atentar à classe criada para o entendimento completo de como foi realizada a execução do código. Utilizou-se a recomendação da explicação do trabalho prático. A estrutura do índice invertido é baseada num map de string e de set de string, conforme pode ser analisado na declaração da classe:

```
19
20 class Dicionario{
21     public:
22         Dicionario();
23         Dicionario(string s);
24         set<string> documentos();
25         set<string> consulta(string palavra);
26         double numdoc();
27         map<string,double> idf();
28     private:
29         map<string,set<string> > iv;
30         int ndoc_;
31     };
32
```

Após o devido tratamento discutido anteriormente da palavra, a classe Dicionário recebe a palavra a ser buscada e associa a ela os documentos onde há ocorrência. Em seu construtor, todos os arquivos são abertos e cada palavra de cada arquivo é inserida no map como uma chave. Após inserir a chave, insere-se

em seu valor (set<string>) o nome do arquivo de texto aberto, já que se a varredura encontrou a palavra no arquivo corrente, este deve ser indexado a palavra. Dessa forma, cada palavra presente nos arquivos de texto terá em seu valor um set com a relação dos documentos nos quais essa palavra está presente.

```
130
131  Dicionario::Dicionario(){
132      ndoc_ = 0;
133      ifstream arquivos;
134      arquivos.open("arquivos.txt");
135      string titulo;
136      while (arquivos>>titulo){
137          ndoc_++;
138          ifstream palavras;
139          string palavra;
140          palavras.open(titulo);
141          while (palavras>>palavra){
142              minuscule(palavra);
143              this->iv[palavra].insert(titulo);
144          }
145      }
146  }
147
```

Na função consulta, recebe-se a palavra a ser buscada e por meio da função “find” nativa da estrutura map, é realizada a busca da chave correspondente a palavra recebida como parâmetro. Caso a palavra seja encontrada (exista nos documentos de texto) o valor dessa chave é impresso. Ou seja, aparecerá na tela o nome dos arquivos nos quais a palavra pode ser encontrada.

```
147
148  set<string> Dicionario::consulta(string palavra) {
149      map<string, set<string> >::iterator it;
150      it = iv.find(palavra);
151      if(it != iv.end()) {
152          return it->second;
153      }
154      else {
155          set<string> bla;
156          return bla;
157      }
158  }
```

A busca pelos arquivos é, portanto, realizada se os arquivos-texto tiverem sido abertos corretamente. Uma vez que o usuário digita um dos vocabulários, é possível buscar se existe algum documento de texto relacionado a essa palavra. Se existir, o nome desse documento será impresso. Se existir mais de um arquivo que possui essa mesma palavra, deve-se imprimir, também, o nome dos outros arquivos que possuem ela. Deve-se observar que a impressão dos arquivos que contém a palavra tem valor de teste, já que o trabalho consiste em uma máquina de busca, que pode receber uma ou mais palavras a cada busca e, não somente uma palavra (como um dicionário).

Um detalhe de extrema importância que se deve dar a devida atenção é o fato de que a condição de parada do programa é o primeiro if desta parte do loop, que segue o programa enquanto a variável for diferente de 'STOP'. Ou seja, para terminar a busca e encerrar é necessário digitar esse comando de parada.

```
23
24 //Cria um arquivo txt com a consulta
25 int criar_consulta() {
26     ofstream consulta;
27     consulta.open("consulta.txt");
28     string palavra;
29     cout<<"Digite o texto que deseja buscar (digite 'pause' para terminar): ";
30
31     while(palavra != "pause"){
32         cin>>palavra;
33         if(palavra == "pause"){
34             break;
35         }
36         if(palavra == "STOP"){
37             return 0;
38         }
39         consulta<<palavra<<" ";
40     }
41     consulta.close();
42     return 1;
43 }
```

### 2.3 Determinação de Coordenadas

Na parte de determinação de coordenadas é realizada a ligação, ou interseção, dos arquivos que contem cada uma das palavras da busca. Após cada palavra ser devidamente indexada, um auxiliar de `set<string>` é criado para receber o set(conjunto de documentos de texto) de cada palavra existente na busca. A partir da segunda iteração outro auxiliar é usado para efetuar a interseção do conjunto documentos de texto de cada palavra. Ou seja, para obtermos os documentos que serão resultado da busca, é necessário que todas

as palavras existam nesse documento resultado. Para isso, cada vez que o iterator (mecanismo de varredura dos conjuntos de cada palavra) assume o nome de um documento que contém uma palavra, mas não contém uma se quer das outras, o mesmo é retirado do auxiliar, significando que esse documento não pode ser resultado para a busca.

```
45
46 //Cria set<string> com a interseccão dos documentos que contem a c_consulta
47 set<string> Dicionario::documentos() {
48     ifstream p;
49     p.open("consulta.txt");
50     int a=0;
51     set<string> docs;
52     string palavra;
53     while(p>>palavra){
54         minusculo(palavra);
55         if(a==0){
56             docs = consulta(palavra);
57             a++;
58             continue;
59         }
60         set<string> aux = consulta(palavra);
61         for(set<string>::iterator it = docs.begin(); it != docs.end(); it++){
62             if(aux.count(*it)==0){
63                 docs.erase(*it);
64             }
65         }
66     }
67     return docs;
68 }
```

## 2.4 Ranking Cosseno

De acordo com a descrição do trabalho prático, deve ser utilizado um método para definir ordens de relevância para os documentos resposta. O método denominado Ranking Cosseno é baseado no conceito de vetor e suas relações. Utiliza-se o cosseno do ângulo entre os vetores como unidade de medida de proximidade, ou seja, quanto maior o cosseno (menor o ângulo) mais próximos estão os vetores. Quando dizemos vetores, nos referimos a estrutura efetuada nos tópicos anteriores e, mais precisamente, aos vetores palavras associadas a documentos. Após a determinação de coordenadas, temos pronto um vetor, representando a importância de cada palavra no texto da busca. No ranking, comparamos esse vetor com o vetor das relevâncias de cada palavra de cada um dos documentos resultado. Desse modo, o documento que contém as relevâncias mais parecidas com as da busca é o documento mais próximo.

Para começar, buscamos da determinação de coordenadas a relevância das palavras em cada um dos documentos, denominada idf, e a colocamos em um map de string e double dentro do construtor da classe do ranking. Dentro da



função inserir da classe, que tem esse map e um set com o nome dos documentos como parâmetros, realizamos as contas do ranking cosseno de acordo com a fórmula presente na descrição do trabalho. Essas operações são realizadas tanto para os arquivos de texto quanto para a própria busca. Para compararmos o resultado da busca com os dos arquivos, usamos a função similaridade, que retorna um vetor com o valor das similaridades entre a busca e cada documento. Para organizar e de fato construir um ranking usamos a função link, que cria um map para ligar o valor da similaridade com cada documento, tudo isso em ordem crescente. E por fim, para organizar e imprimir o ranking de forma correta (decrecente), usamos a função imprimir ranking.

### 3. Teste de Unidades

Entende-se a necessidade da criação de tratamento de erro como a existência de possíveis situações em que o programa não desempenha a função destinada por ele a partir da utilização do usuário. Dessa maneira, percebe-se, por exemplo, que se determinado arquivo não existe e, ainda assim, ele é passado como um dos parâmetros, o algoritmo não é capaz de realizar a consulta, sendo esta a sua finalidade principal. Além disso, tendo em vista que esta situação (sem o tratamento de erro) não estava prevista pelo algoritmo, não é possível determinar, a princípio, qual será o comportamento do código. Dessa forma, deve-se considerar todas as opções viáveis que possam prejudicar o funcionamento ideal do algoritmo para que ele possa ser considerável confiável por seus usuários.

### 4. Conclusão

Para realizarmos o trabalho, usamos o conhecimento obtido durante todo o semestre da disciplina de Programação e Desenvolvimento de Software 2, passando por tipo abstrato de dados, versionamento de código, contêineres, etc. Com isso, pudemos colocar na prática todo o conhecimento teórico e aprender um pouco de como realmente é programar. O mais interessante e satisfatório foi poder conceber algo útil, que tem real valor prático e que nos aproxima da realidade fora do ambiente de sala de aula.

O trabalho nos ensinou valores como trabalho em equipe, persistência, paciência e dedicação. Mas, acima de tudo, nos fez aprender com nossos erros e, dessa forma, ir em busca do desenvolvimento pessoal e profissional.

Exemplo de uma pesquisa no programa desenvolvido no trabalho prático em questão:

```
ca Prompt de Comando - a
Digite o texto que deseja buscar (digite 'pause' para terminar ou 'STOP' para fechar o programa): eu sei
pause

Ranking
Posicao 1: Serenata.txt
Posicao 2: Juntos.txt
Posicao 3: Boate_Azul.txt
Posicao 4: Vou_Ter_Que_Superar.txt
Posicao 5: Evoluiu.txt
Posicao 6: Voce_Nao_Soube_Me_Amar.txt
Posicao 7: Entre_Tapas_E_Beijos.txt

Digite 1 para realizar outra busca e 0 para fechar o programa
```

## 5. Referências Bibliográficas

[http://www.ufjf.br/jairo\\_souza/files/2012/11/9-Indexa%C3%A7%C3%A3o-invertida.pdf](http://www.ufjf.br/jairo_souza/files/2012/11/9-Indexa%C3%A7%C3%A3o-invertida.pdf) <https://homepages.dcc.ufmg.br/~nivio/cursos/ri09/tp1/ri09tp1.pdf>

[https://www.researchgate.net/figure/Figura-3-Exemplo-de-Indice-Invertido-O-Indice-Sequencial-consiste-de-uma-lista-de-pares\\_fig3\\_307693724](https://www.researchgate.net/figure/Figura-3-Exemplo-de-Indice-Invertido-O-Indice-Sequencial-consiste-de-uma-lista-de-pares_fig3_307693724)

<https://pt.slideshare.net/VanessaBiff/arquivo-invertido>

[http://www.sbmec.org.br/eventos/cnmac/xxxi\\_cnmac/PDF/219.pdf](http://www.sbmec.org.br/eventos/cnmac/xxxi_cnmac/PDF/219.pdf)

<http://ptcomputador.com/P/computer-programming-languages/86551.html>