Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência de Computação

Algoritmos e Estrutura de Dados II
2º semestre de 2018
Profs. Luiz Chaimowicz, Gisele Pappa
Antônio Loureiro, Cristiano Arbex

Trabalho Prático 3 - Máquina de Busca

Valor: 15 pontos
Data da devolução: 02/12/2018

Uma máquina de busca é um programa que procura por palavras-chave em documentos de uma base de dados. Ela é composta usualmente por três componentes: *crawler*, realiza a coleta dos documentos; *indexador*, lê os documentos e constrói um índice invertido; e *processador de consultas*, consulta o índice e ordena os documentos de acordo com a relevância com a consulta.

Neste trabalho prático você deverá desenvolver um **indexador em memória** e um **processador de consultas**. O indexador percorre os documentos da base (*corpus*) e relaciona os termos encontrados aos documentos que os contém, criando o índice invertido. Um índice invertido é uma estrutura simples que mapeia um conjunto de palavras aos documentos onde elas estão presentes, e deverá ser implementado utilizando uma estrutura de *hashing*. Você poderá escolher a função de *hashing* que achar mais adequada, lembrando de justificar sua escolha na documentação. Conflitos poderão ser tratados com endereçamento aberto.

O índice invertido é acessado pelo processador de consultas para recuperar rapidamente a lista dos documentos associados à palavra, além do peso da palavra no documento (frequência da palavra no documento). Este índice pode ser representado em memória na forma de uma lista invertida, conforme o exemplo abaixo:

```
[termo]  => (id do documento, frequência do termo no documento), ...
-----------------------------------
[casa]   => (1,3),(3,2)
[maca]   => (3,5),(4,1),(6,1)
[lua]    => (2,3),(3,2),(5,1)
```

Exemplo 1. Índice invertido contendo 3 termos, indicando os documentos onde estão presentes e sua respectiva frequência.

Neste exemplo a palavra (termo) “casa” aparece nos documentos 1 e 3, com frequências 3 e 2, respectivamente. Assim, sua primeira tarefa será ler um conjunto de documentos e contar a frequência das palavras no documento para criar o índice invertido.

A base com documentos (*corpus*) será uma pasta onde os arquivos contidos correspondem aos seus documentos, e o nome de cada arquivo corresponde ao ID do documento. Algumas palavras são muito frequentes e carregam pouca informação (tais como artigos, conjunções, sujeitos, etc), e são conhecidas como *stopwords*. Estas palavras deverão ser descartadas na indexação e na consulta, e lidas do arquivo *stopword_file.txt*.

Seu programa deverá **construir o índice invertido** com base no *corpus* fornecido e **imprimir em um arquivo** o índice invertido, assim como no Exemplo 1. Note que para cada termo, a lista de documentos é **ordenada** de acordo com a frequência do termo. Após a construção do índice, **exiba na tela** quantos conflitos houveram durante a criação da tabela hash.

Seu programa deverá também **prover uma função de busca** no índice invertido. Dado um termo de consulta, ele deverá retornar todos os documentos em que o termo está presente. Para verificar a eficiência da busca, **meça o tempo gasto** por esta função. O Exemplo 2 mostra uma das formas de medir tempo na linguagem C. O termo de busca deverá ser lido a partir de um arquivo contendo a string na primeira linha. Seu programa deverá abrir este arquivo (fornecido via argv) e realizar a consulta.

```
/* Program to demonstrate time taken by function foo() */
#include <stdio.h>
#include <time.h>

// A function that terminates when enter key is pressed
int foo(){
    //Do a lot of interesting stuff
    return 1;
}

// The main program calls foo() and measures time taken by foo()
int main(){
    // Calculate the time taken by fun()
    clock_t t;
    t = clock();
    foo();
    t = clock() - t;
    double time_taken = ((double)t)/CLOCKS_PER_SEC; // in seconds

    printf("foo() took %f seconds to execute \n", time_taken);
    return 0;
}
```

Exemplo 2. Como medir o tempo gasto na linguagem C.

Indo além (Ponto Extra)

O processador de consultas usará o índice invertido para retornar os documentos mais relevantes para consulta. O processador de consultas desenvolvido usará o modelo espaço vetorial, que computa o grau de similaridade entre a consulta e os documentos, e gera um *ranking* de resposta ordenado pela maior similaridade. Este modelo representa a consulta e os documentos como vetores de peso dos termos. Neste trabalho, usaremos a similaridade normalizada proposta por Zobel et al (2006) apresentada abaixo:

$$sim(d_j, q) = \frac{\sum_{i=1}^{N} w_{i,j} * w_{i,q}}{W_d}$$

onde $\mathbf{d_j}$ e **q** são vetores representando o documento e a consulta, respectivamente, e cada posição do vetor é ocupado pelo peso do termo: $w_{i,j}$ ou $w_{i,q}$. *N* representa a quantidade de termos no espaço vetorial, W_d é uma normalização da similaridade em função do tamanho documento. W_d é definido como a **raiz quadrada do número de termos distintos no documento**. O peso do termo no documento $w_{t,d}$ é computado da seguinte forma:

$$w_{t,d} = f_{t,d} \cdot \log \frac{|D|}{f_t}$$

$f_{t,d}$ é a frequência do termo *t* no documento *d*, f_t é o número de documentos na base que possuem o termo *t* e |D| é o número total de documentos na base. O segundo fator na equação corresponde à raridade do termo, que seria o inverso da frequência nos documentos da base (*idf: inverse document frequency*). O peso do termo na consulta $w_{t,q}$ é computado de forma análoga.

Seu processador de consultas deverá permitir buscar por mais de um termo. Se a consulta é "*computer science",* documentos com *computer* e/ou *science* deverão ser recuperados. Para os documentos que possuem mais de um termo da consulta, o valor da similaridade será acumulado.

Assim, seu programa deverá computar a similaridade da busca com todos os documentos, utilizando para isso o índice invertido, e retornando os 10 documentos mais similares. A entrada será um arquivo texto contendo a consulta na primeira linha. A saída serão os IDs dos 10 documentos ordenados por relevância.

Abaixo, apresentamos um exemplo de entrada, saída e como seu programa será executado.

consulta1.txt:

```
Science
```

Execução

```
$ ./tp3 /corpus/consulta1.txt /corpus/stopword_file.txt
```

Saída:

```
science: 0.2 seconds
10 4 5 2 32 11 15 22 29 10 (Caso seja feito o ponto extra)
```

inverted_list.txt

```
[hackers]  => (1,8),(12,2)
[justice]  => (3,5),(2,4),(8,1)
[magazine] => (1,4),(10,2),(15,1),(12,1)
```

O que deve ser entregue:

- Código fonte do programa em C (todos os arquivos .c e .h), bem identado e comentado. Não inclua o executável ou arquivos do projeto da IDE.
- Documentação do trabalho. Entre outras coisas, a documentação deve conter:
    a. Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
    b. Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
    c. Estudo de Complexidade: estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O).
    d. Testes: descrição dos testes realizados e listagem da saída (não edite os resultados).
    e. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
    f. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso

Submissão:

O trabalho (código e documentação) deverá ser entregue no Moodle. Uma tarefa será criada para a entrega.

Atenção: O trabalho será avaliado em um ambiente Linux e compilado com o seguinte

comando:

```
gcc -Wall -std=c99 -lm *.c -o tp3
```

Evite incluir bibliotecas que funcionam apenas em ambientes Windows, como `windows.h`. Se tiver alguma dúvida, pergunte a nós que podemos te sugerir alguma mudança caso necessário.

Comentários Gerais:

1. Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
2. Clareza, indentação e comentários no programa também serão avaliados.
3. O trabalho é individual.
4. Trabalhos copiados serão penalizados conforme anunciado.
5. Penalização por atraso: $(2^d - 1)$ pontos, onde d é o número de dias de atraso, contando a partir da data de entrega e não do fim da tolerância.

Referências

Gerard Salton and Christopher Buckley. 1988. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.* 24, 5 (August 1988), 513-523.

Justin Zobel and Alistair Moffat. 2006. Inverted files for text search engines. *ACM Comput. Surv.* 38, 2, Article 6 (July 2006)