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

Trabalho Prático 3 - Máquina de Busca Avançada

Valor: 15 pontos
Data da devolução: 22/02/2022

Recuperação de Informação é a área responsável por organizar e recuperar documentos de interesse, ordenados pelo grau de relevância com a entrada do usuário. A Universidade Federal de Minas Gerais é mundialmente conhecida pelos trabalhos desenvolvidos nesta área e, recentemente, foi procurada pela XYZ Corporation para uma nova parceria para o desenvolvimento de uma máquina de busca, graças ao sucesso do escalonador de URLs desenvolvido por seus alunos anteriormente.

Uma máquina de busca é composta principalmente por três componentes: *crawler*, que realiza a coleta dos documentos; *indexador*, que lê os documentos e constrói o índice invertido; e *processador de consultas*, que consulta o índice e ordena os documentos de acordo com a relevância com a consulta. O índice invertido é uma estrutura simples que mapeia um conjunto de palavras aos documentos onde elas estão presentes.

Sem perda de generalidade, vamos assumir um vocabulário de 3 palavras (casa, lua e maca) e o seu corpus seja constituído de 6 documentos, apresentados a seguir:

Id do Doc	Conteúdo
1	casa a casa da casa
2	a lua
3	casa casa maca maca maca maca maca lua
4	a maca
5	lua e lua
6	maca

Como já mencionamos, uma forma de organizar os dados de um corpus é através de um índice invertido. Para o corpus apresentado acima, o índice invertido seria:

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

Observe que, por serem *stop words*, os termos "a","da" e "e" não são considerados no índice.

Uma forma de realizar consultas tipo máquina de busca é utilizando o Modelo Espaço Vetorial (MEV). Dada uma consulta q, o MEV permite calcular o grau de similaridade entre q e cada um dos documentos do corpus. Esses graus de similaridade podem ser ordenados, gerando um *ranking* de documentos ordenado pela maior similaridade. O MEV representa a consulta e os documentos como vetores de pesos de termos, e o grau de similaridade consulta-documento corresponde à distância do cosseno entre os vetores da consulta e do documento, originalmente proposto conforme a equação abaixo:

$$sim(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j} * w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} * \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

onde $\mathbf{d_j}$ e $\mathbf{q}$ são vetores do documento j e da consulta q, respectivamente, e cada posição do vetor é ocupado pelo peso do termo i: $w_{i,j}$ ou $w_{i,q}$. *N* representa a quantidade de termos no espaço vetorial.

A equação original possui uma normalização vetorial e produz valores de similaridade entre 0 e 1, diferente de algumas variações propostas na literatura. Neste trabalho, deverá ser usada a variação proposta por Zobel et al (2006) apresentada abaixo:

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

$$W_d = \sqrt{\sum_t w_{d,t}^2}$$

onde W_d normaliza o produto interno dos vetores de pesos em função do tamanho do documento. Esta normalização dispensa percorrer novamente todo o corpus a cada consulta e, neste trabalho, W_d será a raiz quadrada do número de termos distintos no documento. A normalização em função da consulta é desconsiderada por ser constante para todos os documentos.

O peso do termo é comumente computado baseado na função *tf.idf* (Salton, 1988). Assim o peso do termo t no documento d, $w_{t,d}$ é computado da seguinte forma:

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

onde $f_{t,d}$ é a frequência do termo *t* no documento *d*, f_t é 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 valor do *idf* tende a ser muito alto quando o termo é bastante raro, e por isto é amortizado com a função log (adote log natural, base *e*). ~~O peso do termo na consulta $w_{t,q}$ é computado de forma análoga. Neste caso, geralmente, a frequência da palavra tende a ser igual a um e $w_{t,q}$ carrega apenas o *idf*.~~

Já o valor de $w_{i,q}$ é um vetor binário, onde a posição recebe valor 1 se o termo ocorre na consulta e valor 0 caso contrário. Sua máquina de busca deverá tratar consultas de mais de um termo usando o operador OU, ou seja, a consulta "casa maca" deve retornar todos os documentos que possuem casa ou maca.

Aplicando esses conceitos à nossa base exemplo, onde D=6, temos:

	Id do documento					
$w_{t,d}$	**1**	**2**	**3**	**4**	**5**	**6**
casa	3*ln(6/2)=3.295	0	2*ln(6/2)=2.197	0	0	0
maca	0	0	5*ln(6/3)=3.465	1*ln(6/3)=0.693	0	1*ln(6/3)=0.693
lua	0	1*ln(6/3)=0.693	1*ln(6/3)=0.693	0	2*ln(6/3)=1.386	0

	1	2	3	4	5	6
W_d	3.295	0.693	4.160	0.693	1.386	0.693

Considerando a consulta: "casa maca" temos que $w_{i,q}$ é [1,1,0], e "maca lua" $w_{i,q}$ é [0,1,1].

$sim(d_j,q)$	1	2	3	4	5	6
casa maca	3.295/3.295 = 1	0	(2.197+3.465)/4.160 = 1.361	0.693/0.693 = 1	0	0.693/0.693 = 1
maca lua	0	0.693/0.693 = 1	(3.465+0.693)/4.160 = 0.999	0.693/0.693 = 1	1.386/1.386 = 1	0.693/0.693 = 1
casa	3.295/3.295 =1	0	2.197/4.160 = 0.528	0	0	0

Nesse caso, a saída ordenada da consulta "casa maca" é: 3, 1, 4, 6

Para a consulta "maca lua" é: 2, 4, 5, 6, 3

Enquanto para a consulta "casa" é : 1,3

Note que, em casos onde a similaridade entre 2 ou mais documentos e a consulta for a mesma, os documentos deverão ser ordenados do menor para o maior id.
Os documentos e consultas dos testes estarão em inglês, e portanto os acentos não precisam ser tratados. Porém, caracteres de pontuação (ex: .,!?:;) devem ser tratados. Também é aconselhável que você transforme todo o texto em caixa baixa (*lower case*) para evitar problemas nas comparações.

Neste trabalho prático você deverá desenvolver um **indexador em memória** e um **processador de consultas**. O indexador percorre os documentos da base (*corpus*) e, para cada termo, relaciona os documentos em que eles aparecem e com que frequência, criando o índice invertido. Como já explicado, um índice invertido é um tipo abstrato de dados que mapeia um conjunto de palavras aos documentos onde elas estão presentes, e deverá ser implementado utilizando uma estrutura de *hashing*.

Como mencionado anteriormente, seu processador de consultas deverá permitir buscar por mais de um termo e o operador padrão será OU. Ou seja, se a consulta é *computer science,* documentos com apenas *computer* e/ou *science* deverão ser recuperados. Para os documentos que possuem mais de um termo da consulta, o valor da similaridade será acumulado. Em síntese, o trabalho do processador de consultas é calcular a similaridade de cada consulta com cada documento do corpus, retornando os 10 documentos mais similares à consulta.

Para efeito de validação, seu sistema será testado com consultas e deverá produzir a mesma saída para os top-10 documentos do *ranking*. A entrada será um arquivo texto contendo a consulta na primeira linha. A saída será uma lista ordenada com os IDs dos 10 documentos mais relevantes. Abaixo, apresentamos um exemplo de entrada, saída e como seu programa será executado.

consulta1.txt

```
computer science
```

ranking1.txt

```
10 4 5 2 32 11 15 22 29 10
```

Execução

```
$ ./buscar -i consulta1.txt -o ranking1.txt -c corpus -s stopwords.txt
```

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.

Além disso, 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 stopwords.txt - uma palavra por linha.

O que deve ser entregue:

- Código fonte do programa em C ou C++ (todos os arquivos .c/.cpp e .h), bem identado e comentado.
- 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, compilador utilizado, 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), considerando conjuntos de tamanho n.
d. **Estratégias de Robustez:** Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
e. **Testes:** descrição dos testes realizados e listagem da saída (não edite os resultados).
f. **Análise Experimental:** Apresenta os experimentos realizados em termos de desempenho computacional e localidade de referência, assim como as análises dos resultados.
g. **Conclusão:** comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
h. **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso
i. **Instruções para compilação e execução:** Esta seção deve ser colocada em um apêndice ao fim do documento e em uma página exclusiva (não divide página com outras seções).

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. A submissão será feita pelo minha.ufmg
5. Trabalhos copiados, comprados, doados, etc serão penalizados conforme anunciado.
6. Penalização por atraso: (2^(d – 1)) pontos, onde d é o número de dias de atraso.

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)

ANEXO A: Pseucodódigo em alto nível

```
// Criação do índice
D = ...                    // conjunto de documentos (corpus)
STOP = ...                 // lista de stopwords
Indice = []
Para d em D faça:
  V = []                    // Vocabulário
  Para t em d faça:      // cada termo t do documento d
        Se t em V faça:
          V[t] += 1
        Senão:
          V[t] = 1
   Para t em V faça:
        Se t em STOP: continue
        tf = V[t] /        / frequência do termo no documento
        Se t não em Index faça:
          Indice[t] = []
          Indice[t].add( (d, tf) )

//precomputar de acordo com fórmula definida no TP
W_d = ?                    // norma do documento
```

// Consulta

```
Q = ...                  // consulta
R = []                   // resultado
Wq = 0                   // norma da consulta
Para t em Q faça:        //para cada termo T na consulta Q
  Se t não está no Indice: continue
  docs = Indice[t] // lista de documentos com o termo t
  idf = log( |D|/|docs| )
  w_qt = idf
  W_q += w_qt * w_qt

 Para d em docs:
      tf = d[1]            //doc_id = d[0] e tf = d[1]
      w_dt = tf * idf
      R[d] += w_qt * w_dt

W_q = sqrt(W_q)

// normalizar resultado
Para d em R faça:      R[d] /= W_q*W_d

sort(R) // ordenar decrescente pelo escore

// imprime R
```