Indexação e busca de documentos

1. Problema 1: Construção do Índice Invertido para Máquinas de Busca

A base para o funcionamento de uma máquina de busca, tais como Google e Yahoo!, é a construção de um índice invertido para uma coleção de documentos. Dada a coleção de documentos, um índice invertido é uma estrutura contendo uma entrada para cada palavra (termo) que aparece em pelo menos um documento. Esta entrada associa à palavra pares do tipo <count, doc_id>, onde doc_id identifica um documento e count é o número de vezes em que a palavra em questão apareceu no documento doc_id. Documentos que não contém a palavra não precisam ser indicados.

Como exemplo, suponha uma coleção com apenas os dois documentos seguintes:

<doc1> Quem casa quer casa. Porem ninguem casa. Ninguem quer casa tambem. Quer apartamento.

<doc2> Ninguem em casa. Todos sairam. Todos. Quer entrar? Quem? Quem?

Supondo os identificadores 1 e 2 para <doc1> e <doc2>, respectivamente, o índice invertido seria:

apartamento	1 1
casa	4 1 1 2
em	1 2
entrar	1 2
ninguem	2 1 1 2
porem	1 1
quem	1 1 2 2
quer	3 1 1 2
sairam	1 2
tambem	1 1
todos	2 2

Note que, adjacente a cada palavra do índice está uma lista de pares de números representando ocorrências da palavra nos documentos. O primeiro número de cada par representa o número de vezes que a palavra ocorre em um documento e o segundo representa o identificador do documento em questão. Note que a lista está ordenada por identificadores de documentos.

O sistema deverá processar cada um dos documentos, lendo palavra após palavra e construindo o índice invertido. Ele deverá também associar a cada documento um doc_id único e associar, em memória, este identificador com o nome do documento. Cabe ressaltar que:

a) Uma palavra é considerada como uma sequência de letras e dígitos, começando com uma letra. Portanto, ignore sinais de pontuação.

b) Palavras com letras em maiúsculas devem ser primeiramente transformadas para minúsculas antes da inserção no índice. Desta forma, a mesma palavra apresentada com letras minúsculas ou maiúsculas não serão diferenciadas. Além disso, palavras com acentos serão normalizadas para palavras sem acento. Ou seja: “ação” virará “acao”.

c) Apenas os primeiros 20 caracteres devem ser retidos nas chaves. Assim, duas palavras que não diferem nos primeiros 20 caracteres são consideradas idênticas.

d) Palavras constituídas por menos do que 20 caracteres devem ser preenchidas por um número apropriado de brancos.

e) Uma palavra pode ocorrer múltiplas vezes na mesma linha de um documento, ou mesmo em múltiplas linhas de um mesmo documento.

2. Problema 2: Consultas sobre o índice invertido

Nesta parte do projeto, você utilizará o índice invertido construído para realizar consultas de um ou mais termos. O objetivo é, dado uma consulta, retornar uma lista de documentos, ordenados pela **relevância** para a consulta.

Existem vários métodos de estimar a relevância de documentos de uma coleção para uma consulta. Vocês utilizarão um simples, baseado na frequência dos termos da consulta em cada documento (TF – *term* frequency) da coleção bem como na frequência inversa dos documentos (IDF – *inverse document frequency*). O componente IDF estima o quão bem um termo ajuda a discriminar os documentos entre relevantes e não relevantes: um termo que aparece em muitos documentos (valor de IDF baixo) não é um bom discriminador, enquanto que um termo que aparece em poucos documentos (IDF alto) é um bom discriminador dos mesmos.

Em outras palavras, dada uma consulta com *q* termos, t_1, t_2, ..., t_q, a relevância de um documento i, *r(i)*, é computada como:

$$r(i) = \frac{1}{n_i} \sum_{j=1}^{q} w_j^i$$

onde n_i é o número de termos distintos do documento i e w^i_j é o peso do termo t_j no documento i, que é calculado como:

$$w_j^i = f_j^i \frac{\log(N)}{d_j}$$

onde f^i_j é o número de ocorrências do termo t_j no documento i, d_j é o número de documentos na coleção que contém o termo t_j e *N* é o número de documentos na coleção. Se o termo t_j não aparece no documento i, $f^i_j = 0$.

3. Entrada de dados

A Wikipedia é uma enciclopédia escrita a milhões de mãos e atualizada diariamente, sendo uma ótima fonte de dados para diversos tipos de aplicações como reconhecimento de domínio, melhoria de métodos de processamento de linguagem natural, entre outros. A DBPedia, por sua vez, é um projeto que visa estruturar a informação contida na Wikipedia e entregá-la num formato de fácil consumo para desenvolvedores. Diversos conteúdos da Wikipedia podem ser encontrados através do DBPedia no endereço http://wiki.dbpedia.org/Downloads2015-04.

Iremos utilizar o arquivo Short Abstracts para indexar as entradas da Wikipedia da seguinte forma: o documento será identificado através do seu titulo (URI). A descrição da página será dado pelo texto que descreve o documento. Por exemplo, veja a linha abaixo do arquivo Short Abstracts:

```
<http://dbpedia.org/resource/An_American_in_Paris> <http://www.w3.org/2000/01/rdf-schema#comment> "An American in Paris is a jazz-influenced symphonic poem by the American composer George Gershwin, written in 1928. Inspired by the time Gershwin had spent in Paris, it evokes the sights and energy of the French capital in the 1920s and is one of his best-known compositions.Gershwin composed An American in Paris on commission from the conductor Walter Damrosch. He scored the piece for the standard instruments of the symphony orchestra plus celesta, saxophones, and automobile horns."@en .
```

O arquivo é dividido em 3 colunas. A primeira (azul) possui o identificador do documento a ser indexado (URL da página). A segunda coluna (vermelho) contém o tipo de dado da terceira coluna (comentário) e é igual para toda linha do arquivo, não sendo de interesse para o nosso sistema. A terceira coluna (verde) contém parte do texto da página, a qual descreve o conteúdo do documento.

4. Requisitos da parte 1 do trabalho

DATA DE ENTREGA: 08 de Novembro de 2015.

Na primeira parte do trabalho, deverá entregar o sistema que:

- Leia o arquivo .nt do DBPedia e crie o índice invertido
- O índice invertido será criado através da técnica de hashing com encadeamento exterior
- Deverá experimentar o uso de 5 funções de hashing e analisar a eficácia de cada uma delas antes de escolher a função definitiva para o sistema. Faça uma pesquisa bibliográfica para se embasar em quais funções de hashing implementar. Variações simples da mesma estratégia não serão consideradas.
- Após o índice ser criado, o usuário pode fazer buscas no sistema. Para a busca, o usuário digitará uma palavra e o sistema informará todos os documentos que contém a palavra, ordenado pelo peso (valor de *w*, conforme equação na seção 2). Será informada uma palavra por vez.

Será entregue um relatório contendo a pesquisa bibliográfica sobre as funções de hashing, os experimentos (ver Apêndice 1) e demais considerações sobre o sistema.
Pequenas melhorias podem ajudar o sistema a ser mais eficiente. Pense nessas melhorias, seja criativo e informe as melhorias implementadas.

. 5. Requisitos da parte 2 do trabalho

DATA DE ENTREGA: 05 de Dezembro de 2015.

Na segunda parte do trabalho, deverá entregar o sistema que:

• Crie o índice invertido usando uma estrutura de TRIE ao invés da tabela de hash.
• Permita que o usuário realize buscas com mais de uma palavra e retorne os documentos mais relevantes (segundo o TF-IDF, ou seja, r(i) da seção 2).

Será feito um experimento para analisar o desempenho e gasto de memória da aplicação ao utilizar hashing ou TRIE. Para tal, analise o tempo gasto para construção do índice invertido (usando o arquivo completo em inglês e em português), gasto de memória após o índice criado, tempo gasto para processar 100 consultas aleatórias em cada abordagem. Interprete os dados gerados pelo experimento e fundamente a sua argumentação de acordo com o conteúdo discutido em sala de aula e estudado por você.

Novamente, diversas melhorias podem ajudar o sistema a ser mais eficiente nesta segunda parte do trabalho. Analise seu problema e proponha ajustes visando um melhor desempenho e/ou gasto de memória.

O trabalho será avaliado em 4 dimensões: completude/corretude, pontualidade, qualidade da solução e relatório. Os pesos de cada dimensão serão definidos posteriormente. Dentre os critérios, estão:

- Completude/Corretude: o sistema deve ter todas as funcionalidades pedidas e retornar os resultados esperados.
- Pontualidade.
- Relatório bem redigido. Experimentos e discussão bem fundamentada.
- Qualidade da solução: Código documentado e boa prática de programação (o mínimo necessário de variáveis globais, variáveis e funções com nomes de fácil compreensão, soluções elegantes de programação, código bem modularizado, etc). Boas soluções para melhorar o desempenho do sistema (tempo e/ou espaço). Boa fundamentação para escolha das soluções de melhoria.

Apêndice I – Testando funções de hash

. Uma forma de testar funções de hash é analisar a quantidade de colisões e a distribuição das colisões na tabela. Assim, o pode-se pensar em analisar:

- A Quantidade Média de Colisões Totais = (Total de Colisões) / n.

- A Distribuição Média das Colisões = (Total de Colisões) / (Total de pastas que colidiram ao
  aplicar h(k)
  Imagine que, ao inserir registros em uma tabela utilizando duas funções distintas de *hashing* (h1 e h2), são gerados os seguintes resultados:
  • h1 gerou 5 colisões totais na tabela, porém todas as 5 colisões aconteceram em somente 2 posições da tabela (ou seja, 5 chaves distintas, ao aplicar h1(k), foram direcionadas para uma das duas posições de colisão).
  • h2 gerou 7 colisões totais na tabela, porém as 7 colisões aconteceram em 5 posições distintas na tabela.
  Sendo assim, temos (para somente 1 execução do algoritmo):

- Quantidade de Colisões Totais de h1 = 5

- Quantidade de Colisões Totais de h2 = 7

- Distribuição das Colisões de h1 = 5 / 2 = 2,5

- Distribuição das Colisões de h2 = 7 / 5 = 1,4
  Analisando os resultados, verificamos que a função h1 gera menos colisões do que a função h2. Contudo, as colisões geradas pela função h2 são melhor distribuídas que as colisões geradas pela função h1. Em algumas situações, pode ser mais interessante escolher uma função que distribua melhor as colisões, mesmo que ela gere algumas colisões a mais que outra função qualquer, pois permite um resultado melhor ao aplicar técnicas de tratamento de colisões como, por exemplo, na abordagem de encadeamento.