Instituto de Matemática e Estatística
Universidade de São Paulo

Recuperação de Informações em Bancos de Dados Textuais

Aluna: Marcela Ortega Garcia
Orientador: Prof. Dr. João Eduardo Ferreira

1 de dezembro de 2009

Sumário

Parte I

Técnica

1 Introdução

Um dos principais objetivos da área de banco de dados é armazenar e recuperar dados de maneira eficiente. O crescimento do uso de bancos para dados sem estrutura definida desencadeou a necessidade de técnicas diferenciadas. Dentre esses tipos de dados, encontram-se os textos e um exemplo que ilustra esse cenário é a pesquisa em páginas Web, um conjunto volumoso corriqueiramente consultado.

1.1 Recuperação de Informação

Recuperação de Informação (RI) é uma área que trata da representação, armazenamento, organização e acesso a informações [BYRN+99]. Em um sistema de RI, a representação e a organização das informações devem ser projetadas para facilitar o acesso do usuário e, assim, ser capaz de relacionar uma consulta a documentos armazenados no banco de dados.

Uma consulta é uma tradução da informação que o usuário necessita de maneira que o sistema de RI possa processar. Por exemplo, considere a seguinte busca:

"Quais as queixas dos pacientes diagnosticados com Distrofias Musculares Congênitas?"

Usualmente não é possível traduzir uma pergunta em uma expressão lógica e, dessa maneira, o sistema não conseguirá encontrar os documentos. A forma mais comum de tradução é selecionar palavras-chave que representem a busca e identifiquem a informação desejada.

1.1.1 Recuperação de Dados versus Recuperação de Informação

A recuperação de dados consiste em apenas encontrar documentos que contenham as palavras-chave utilizadas na consulta. Frequentemente isso não é suficiente para um usuário de um sistema de RI que está interessado em encontrar informações [BYRN+99]. O objetivo do sistema é encontrar documentos que sejam relevantes.

A principal diferença é que um sistema de recuperação de dados, como por exemplo um banco de dados relacional, possui dados bem estruturados. Já o sistema de RI trata textos sem estrutura e com linguagem natural que pode ser ambígua. Sendo assim, é necessário que o sistema "interprete" o conteúdo dos textos e avalie a relevância de cada um em relação à consulta.

1.1.2 Banco de dados textuais

Banco de dados textuais são semelhantes aos convencionais: dada uma consulta, cada documento deve ser comparado aos termos da mesma para determinar se ele é uma possível resposta. Para que esse processo seja eficiente, uma estrutura de dados chamada *índice* é utilizada.

De maneira geral, um índice é qualquer palavra que apareça no texto. A eficiência da recuperação de informação está diretamente relacionada a qual conjunto de termos do texto será utilizado para gerar índices.

É possível definir algumas palavras-chave manualmente ou automaticamente e utilizar apenas estas como índices, ou então indexar automaticamente todas as palavras do texto. Seleção automática de palavras-chave, no geral, não obtém sucesso e indexação automática de todas as palavras apresenta mais eficiência na recuperação quando comparada à técnica de indexação manual. [BTO+97]

1.2 Proposta

Neste trabalho, estudaremos o processo de recuperação de informações em bancos de dados textuais, enfatizando as técnicas de indexação. Além disso, analisaremos o método de ranqueamento utilizado pelo Google e também as técnicas utilizadas pelo Lucene, uma biblioteca de recuperação de informação.

Como estudo de caso, integramos a biblioteca *Ferret*, escrita em Ruby e baseada no Lucene, ao sistema *Web* do Centro de Estudos do Genoma Humano da USP e realizamos testes, objetivando a comparação de buscas indexadas e buscas em SQL.

2 Fundamentos

2.1 O processo de recuperação de informações

O primeiro passo antes de iniciar um processo de recuperação é a definição do banco de dados de textos. Cada texto da coleção deve ser submetido a uma série de operações gerando uma *visão lógica* do mesmo. Uma vez que ela tenha sido criada, o gerenciador de banco de dados poderá construir os índices do texto.

Tendo todos os índices prontos, podemos inciar o processo de recuperação de informações. Para isso, o usuário deve descrever o que deseja e essa especificação também é submetida às operações textuais, criando uma consulta. Ela pode ser reformulada pelo sistema visando melhores resultados ou enviada diretamente para o próximo passo. Com os termos obtidos, é realizada então a busca por documentos e, posteriormente, eles são ranqueados de acordo com sua relevância.

Esses passos são representados pela figura abaixo e são descritos nas próximas seções.

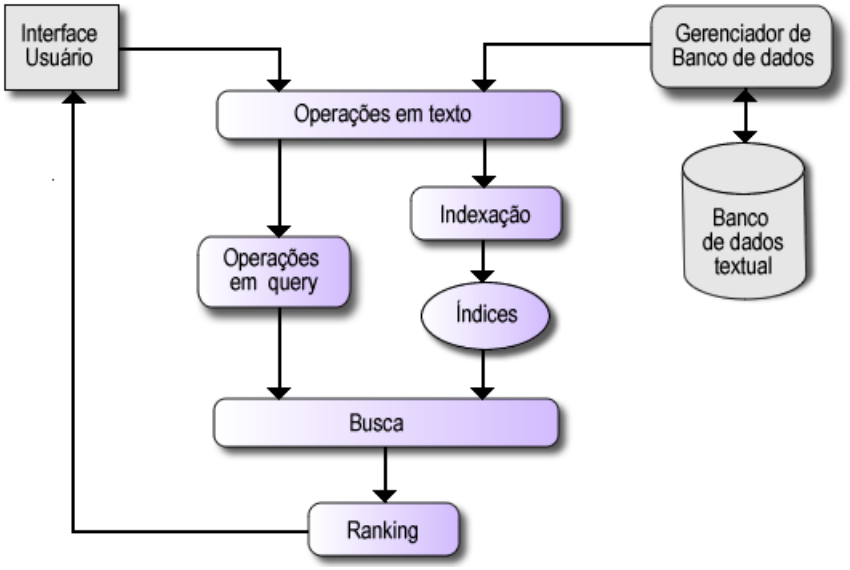

Figura 1: Processo de recuperação de informações.

2.2 Operações em texto

Para o processo de recuperação de informações, é necessário que cada texto possua uma visão lógica, ou seja, tenha uma representação. Computadores modernos permitem

a visão lógica de *texto completo*, ou seja, todas as palavras do texto são utilizadas para representa-lo. Porém, como as coleções de textos estão cada vez maiores, é necessária uma redução desse vocabulário.

Na linguagem escrita, algumas palavras carregam mais significado e representam melhor o conteúdo de um texto do que outras. Por exemplo, artigos "o" ou "um" não acrescentam conteúdo ao texto e não são úteis no processo de recuperação já que, possivelmente, são encontrados em qualquer texto [BYRN+99]. Pré processar o texto em busca dessas palavras é considerado uma boa técnica para controlar o tamanho do vocabulário utilizado como índice e, assim, melhorar o desempenho da busca.

2.2.1 Análise léxica

A análise léxica é o primeiro passo que possibilita a geração automatica de índices. O principal objetivo dessa fase é a identificação de palavras que serão utilizadas como índices. A princípio, parece razoável considerar palavras como sequências de letras limitadas por espaços, porém, como o objetivo de um sistema de RI é "entender" o conteúdo do texto, alguns casos especiais devem ser tratados com cuidado: números, hifens, pontuação, letras maiúsculas ou minúsculas. [FBY92]

Números geralmente não são bons índices pois, sem analisar o contexto, eles não denotam significado. Por isso, usualmente eles não são utilizados como índices. Contudo, existem casos que eles necessitam de atenção como, por exemplo, em um texto sobre vitaminas, "B12" é uma palavra relevante.

No caso do hífen, a dúvida consiste em utilizá-lo ou não como separação de palavras. A quebra em palavras permite tratar por exemplo, "cor-de-rosa" e "cor de rosa" da mesma maneira, podendo ajudar a encontrar documentos importantes. Entretanto, existem palavras significativas que contêm o hífen, como "F-1", "14-Bis". Tanto para os hifens como para os números, a melhor abordagem é criar uma regra geral e especificar uma lista de exceções. [BYRN+99]

Pontuações geralmente são inteiramente removidas para a criação de índices. A única ressalva é sobre pontuações que estão contidas em uma palavra e a ausência delas altera o significado. Por exemplo, em códigos de programas, "x.id" e "xid" não devem ser tratados da mesma maneira.

Letras maiúsculas ou minúsculas também não costumam influenciar no significado e, por isso, normalmente o texto é inteiramente convertido em minúsculas (ou maiúsculas)

para a criação dos índices assim como as consultas são transformadas para realizar a busca.

Como podemos notar, não há uma solução clara para esses casos. Não há dificuldades na implementação dessas operações [FBY92], porém, é necessário que sejam bem pensadas durante a criação de um sistema de RI, já que elas podem causar um grande impacto na coleção de resultados. Alguns *Web search engines* evitam utilizar todas as operações juntas porque elas podem dificultar o entendimento do usuário ao receber os textos recuperados.[BYRN+99]

2.2.2 Eliminação de *Stopwords*

Desde o início dos estudos sobre recuperação de informações, palavras que são frequentemente encontradas em quaisquer textos são consideradas inúteis ao serem utilizadas como índices [FBY92]. Essas palavras são chamadas de *stopwords* e usualmente são eliminadas na indexação automática de textos. Como elas são encontradas na maioria dos textos, a eliminação não altera a eficácia da recuperação e ainda possibilita a redução do tamanho da estrutura de indexação.

Assim como na análise léxica, não é muito claro quais palavras devem entrar em uma lista de *stopwords*. Artigos, preposições e conjunções são candidatos potenciais, entretanto, dependendo do banco de dados, até substantivos podem ser incluídos. Considere, por exemplo, um banco de dados de textos sobre computação: palavras como "computador", "programa" ou "código" possivelmente são bem frequentes.

Em [FBY92] podemos encontrar uma lista de 425 *stopwords* para o inglês. Apesar do benefício da redução de vocabulário, a eliminação de palavras diminui a especificidade da busca e, por isso, alguns sistemas de RI acabam considerando pouquíssimas palavras como *stopwords*, por exemplo, na língua inglesa, "the", "of", "and", "with" e "an".

2.2.3 *Stemming*

Uma palavra possui variações sintáticas como plural, gerúndio, tempo verbal e etc. Ao elaborar uma consulta, o usuário pode especificar uma palavra e, possivelmente, estar interessado também em flexões da mesma.

Stemming é o processo de aplicar operações em uma palavra para encontrar sua raíz gramatical. Por exemplo, "recuperar" é raiz de "recuperação", "recuperações", "recuperam" e "recuperado". Além de ser útil para encontrar possíveis textos relevantes, esse

processo também ajuda a reduzir o tamanho da estrutura de indexação, já que diminui o número de índices distintos.

O método mais utilizado é o de remoção de prefixos e/ou sufixos que é simples e eficientemente implementado [FBY92]. A remoção de sufixos é a parte mais importante, já que a maioria das variações de palavras são realizadas por meio da adição de um sufixo. Um dos algoritmos mais populares é o de Porter (1980), que utiliza uma lista de sufixos e um conjunto de regras. Segue abaixo, como exemplo, a regra de remoção do plural para o inglês [BYRN+99]:

```
select rule with longest suffix {
     sses → ss;
     ies → i;
     ss → ss;
     s → φ;
}
```

A sentença "*select rule with longest suffix*" indica que a regra escolhida será a que tiver o maior sufixo que coincidir com a palavra e ϕ indica uma *string* vazia. Assim, a palavra "dresses" vira "dress" e "books" é transformado em "book".

2.3 Indexação

Os desafios da recuperação de informações em textos estão no desenvolvimento de algoritmos e estruturas de dados eficientes. Isso inclui representação e construção de índices e avaliação de consultas para busca [ZM06]. A estrutura mais utilizada é a de arquivos invertidos. Além dela, outras também serão abordadas a seguir.

2.3.1 Arquivos invertidos

Um arquivo invertido é um mecanismo de indexação orientado à palavra e sua estrutura é composta de duas partes: o vocabulário e as ocorrências [BYRN+99].

O vocabulário é um conjunto de palavras distintas, selecionadas por meio das operações de texto descritas no capítulo anterior. As ocorrências são listas que fornecem informações sobre cada palavra do vocabulário. Essas informações são definidas de acordo com a necessidade da aplicação, podendo ser, por exemplo, documentos nos quais a palavra aparece e frequência e posição da palavra no texto.

Nos índices em nível de documento, as listas mais simples armazenam apenas números que identificam os documentos em que a palavra é encontrada. Porém, como apenas esses dados não são suficientes para alguns tipos de ranqueamentos, a frequência da palavra em cada documento é adicionada. Assim, as listas são sequências de pares $< d, f_{d,t} >$, nos quais t representa o termo, d o documento e $f_{d,t}$ a frequência do termo t no documento d.

Já em índices em nível de palavra, além da frequência também é armazenada a posição de cada palavra. Assim, é simples modificar a lista e adicionar $f_{t,d}$ números que indiquem a posição da palavra no texto: $< d, f_{d,t}, p_1, ..., p_{f_{d,t}} >$.

Segue abaixo um exemplo retirado e adaptado de [ZM06]:

1. The old night keeper keeps the keep in the town.
2. In the big old house in the big old gown.
3. The house in the town had the big old keep.

Tabela 1: Exemplo de documentos.

Palavra	Nível de documento	Nível de palavra
big	$< 2, 2 >, < 3, 1 >$	$< 2, 2, 3, 8 >, < 3, 1, 8 >$
gown	$< 2, 1 >$	$< 2, 1, 10 >$
had	$< 3, 1 >$	$< 3, 1, 6 >$
house	$< 2, 1 >, < 3, 1 >$	$< 2, 1, 5 >, < 3, 1, 2 >$
in	$< 1, 1 >, < 2, 2 >, < 3, 1 >$	$< 1, 1, 8 >, < 2, 2, 1, 6 >, < 3, 1, 3 >$
keep	$< 1, 1 >< 3, 1 >$	$< 1, 1, 7 >< 3, 1, 10 >$
keeper	$< 1, 1 >$	$< 1, 1, 4 >$
keeps	$< 1, 1 >$	$< 1, 1, 5 >$
night	$< 1, 1 >$	$< 1, 1, 3 >$
old	$< 1, 1 >, < 2, 2 >, < 3, 1 >$	$< 1, 1, 2 >, < 2, 2, 4, 9 >, < 3, 1, 9 >$
the	$< 1, 3 >, < 2, 2 >, < 3, 3 >$	$< 1, 3, 1, 6, 9 >, < 2, 2, 2, 7 >, < 3, 3, 1, 4, 7 >$
town	$< 1, 1 >, < 3, 1 >$	$< 1, 1, 10 >, < 3, 1, 5 >$

Tabela 2: Exemplo de listas de ocorrências para os documentos da Tabela 1.

Estruturas e busca

O vocabulário deve ser organizado de maneira que facilite a busca por uma palavra e cada lista deve prover processamento eficiente das ocorrências.

Uma das estruturas utilizadas no vocabulário é a *trie*: uma árvore enária capaz de armazenar um conjunto de *strings* e possibilitar a recuperação de uma delas em tempo

proporcional ao seu tamanho, independente do número e tamanho de *strings* que a árvore possui [BYRN+99]. Cada aresta da árvore é rotulada com uma letra e o caminho da raíz até um nó que contenha uma a lista de ocorrência representa uma palavra no vocabulário.

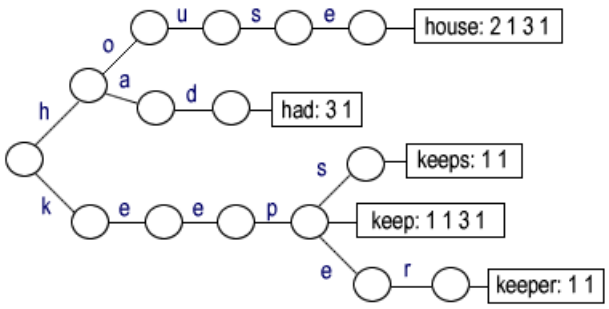

Figura 2: *Trie* para as palavras "had"', "house", "keep", "keeper" e "keeps" dos documentos da Tabela 1.

Já a lista de ocorrências deve ser armazenada de maneira contínua para reduzir o tempo gasto na leitura em disco [BTO+97].

Os índices são geralmente divididos em dois arquivos: um armazena as listas de ocorrências e em outro, o vocabulário é mantido em ordem lexicográfica e, para cada palavra, um ponteiro para sua lista no primeiro arquivo.

Sendo assim, a busca em arquivos invertidos é dividida em três passos: pesquisa no vocabulário, recuperação de ocorrências e manipulação das ocorrências encontradas.

Construção de índices

O que torna a construção de índices desafiadora é o volume de dados. Existem várias abordagens que podem ser classificadas como de "uma-passagem" ou de "duas-passagens", que indicam o número de vezes que o texto é lido durante a construção. As melhores soluções utilizam uma estrutura dinâmica que contém as palavras do vocabulário e também listas de ocorrências dinâmicas.

Existem dois métodos rápidos de construção de índices que utilizam um *buffer* na memória como armazenamento temporário. O de "uma-passagem", é descrito em [BTO+97]:

1. Enquanto o buffer interno não está cheio, pegar documentos; para cada documento d, extrair as palavras distintas e, para cada palavra t:
   a) Se t já apareceu anteriormente, adicione d na lista de documentos de t.
   b) Senão, adicione t à estrutura de palavras e crie uma lista para t contendo d.
2. Quando o buffer estiver cheio, escreva-o no disco, criando um índice parcial. Limpe o buffer e retorne ao passo 1.
3. Junte os índices parciais para criar o índice final.

Após ter todos os índices parciais no disco, o índice I_1 é mesclado ao I_2, o I_3 ao I_4 e assim por diante. Depois, no próximo nível, o índice $I_{1,2}$ ao $I_{3,4}$ e sucessivamente até a obtenção do índice final. Juntar os índices parciais consiste em agrupar os vocabulários e quando existir uma palavra que esteja contida nos dois, agrupar suas listas em uma única lista de ocorrências.

A desvantagem desse método na prática é o uso temporário de espaço para os índices parciais. Como uma palavra pode aparecer em mais de um índice parcial, o espaço total utilizado por eles é maior do que o espaço para o índice final e além disso, o processo de junção também requer espaço. Assim, esse método não é considerado adequado para bancos de dados extensos.

O outro método, classificado como de "duas-passagens", é considerado eficiente e não possui esses problemas. Segue abaixo o algoritmo também encontrado em [BTO+97]:

1. Extrair as palavras distintas de cada documento e, para cada palavras, contar o número de documentos em que ela aparece. (Estatísticas adicionais devem ser armazenadas para um índice a nível de palavra.)
2. Utilize o vocabulário e contadores obtidos para criar um *template* vazio de índices invertidos em disco. O *template* deve conter cada palavra do vocabulário e espaço continuo para sua respectiva lista de ocorrências.
3. Inicialize a segunda passagem criando, em um *buffer* interno, uma lista vazia para cada palavra do vocabulário.
4. Enquando o *buffer* não estiver cheio, pegar documentos; para cada documento d, extrair as palavras distindas e, para cada palavra t, adicionar d à lista de ocorrências de t.
5. Quando o buffer estiver cheio, escreva o índice parcial nos lugares apropriados do *template*. Limpe o buffer e retorne ao passo 4.

Nesse método, o *template* de índices invertidos é progressivamente preenchido até a obtenção do índice final. A cada escrita do *buffer* em disco, é necessário apenas um acesso a ele, minimizando a movimentação da cabeça de disco e aumentando o desempenho.

2.3.2 Vetores de sufixo

Este método de indexação considera o texto como uma grande *string*. Cada posição nessa *string* forma um sufixo do texto, por exemplo, a *substring* de início na posição p até o fim do texto. Como cada posição gera um sufixo diferente, o mesmo é unicamente identificado por sua posição. As posições escolhidas são chamadas de pontos de índice.

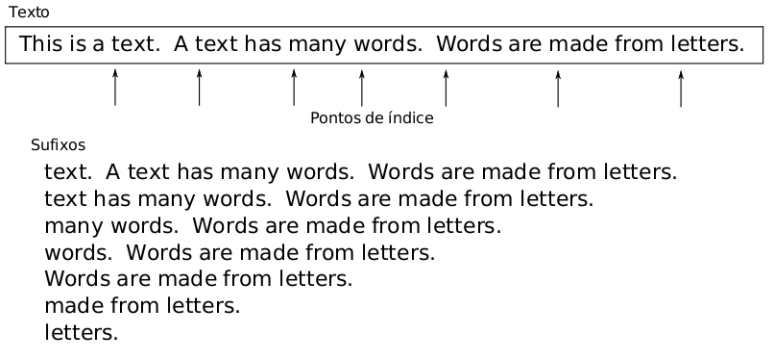

Figura 3: Exemplo de sufixos de texto. (Retirado de [BYRN+99])

Um array de sufixo contém todos os pontos de índice de um texto, organizados de maneira lexicográfica. Assim, essa estrutura permite busca binária realizada por meio da comparação do conteúdo de cada ponto.

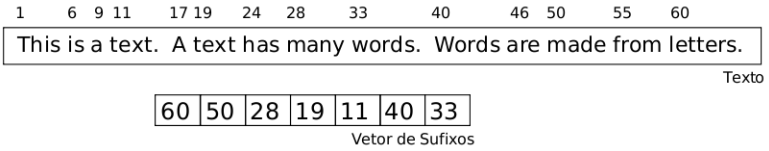

Figura 4: Vetor de sufixos para o exemplo da figura 3. (Retirado de [BYRN+99])

Para executar uma consulta por um padrão nessa estrutura, dois "padrões limites", P_1 e P_2, devem ser criados e a partir deles, procura-se algum sufixo S tal que $P_1 \leq S < P_2$. Por exemplo, para encontrar a palavra "text" no exemplo da figura 3, podemos usar P_1 como "text" e P_2 como "texu" e assim, obtemos a porção do vetor que contém os ponteiros 19 e 11 (figura 4).

2.3.3 Arquivos de assinatura

Arquivo de assinatura é uma estrutura de índice orientada à palavra e baseada em *hashing* [BYRN+99]. Cada documento é representado por uma *bitstring* de tamanho fixo e é dividido em blocos. As palavras são colocadas em um *hash* que indica a assinatura de cada uma, ou seja, quais *bits* devem ser 1 quando a mesma estiver presente.

Se uma palavra está presente em um bloco, os *bits* definidos como 1 em sua assinatura também devem ser 1 na assinatura do bloco.

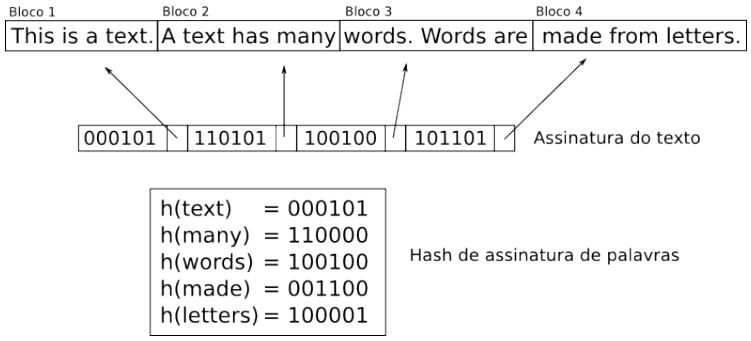

Figura 5: Exemplo de arquivo de assinatura. (Retirado de [BYRN+99])

É possível que os *bits* correspondentes a uma palavra estejam definidos como 1 na assinatura do texto, embora a palavra não apareça. O desenvolvimento da técnica de assinatura de arquivos deve garantir que a probabilidade dessa ocorrência seja baixa.

No momento da busca, apenas os *bits* definidos como 1 nas assinaturas dos termos da consulta devem ser avaliados para concluir se o documento é ou não uma possível resposta.

Esse método não é utilizado nas aplicações atuais pois, além dos textos possuirem tamanhos variáveis, a estrutura não permite diferentes abordagens de consulta.

2.4 Modelos de Recuperação de Informação

A recuperação de informações utilizando índices baseia-se na ideia de palavras-chave para encontrar um documento. Porém, frequentemente os termos de uma consulta estão presentes em um documento que não é de interesse do usuário.

A noção de relevância é o centro da recuperação de informação. O algoritmo de ranqueamento é o responsável pela decisão de qual documento é relevante e, por meio dele,

os resultados obtidos são ordenados de maneira que os do topo da lista são considerados os mais relevantes.

Diferentes abordagens para ranqueamento definem diferentes modelos de recuperação de informações. De acordo com Baeza-Yates e Ribeiro-Neto em [BYRN+99], um modelo de RI é uma quádrupla $[D, Q, F, R(q_i, d_j)]$, na qual:

- D é um conjunto de representações para os documentos de uma coleção.
- Q é um conjunto de representações para as informações que um usuário precisa; Conjunto de consultas.
- F é um *framework* para modelar representações de documentos, consultas e seus relacionamentos.
- $R(q_i, d_j)$ é uma função de ranqueamento que associa um número real com uma consulta $q_i \in Q$ e uma representação de documento $d_j \in D$. Tal ranqueamento definirá uma ordenação entre os documentos relacionados à consulta.

Analisando os índices extraídos de um documento, é fácil notar que eles não possuem o mesmo valor semântico para o texto, ou seja, alguns termos são mais relevantes que outros. Tendo isso em vista, podemos atribuir pesos a cada termo, isto é, um valor numérico $w_{i,j}$ que indica o grau de relevância do termo k_i no documento d_j. Caso o termo não esteja presente, $w_{i,j} = 0$. Assim, para cada documento d_j, sendo t o número de índices no sistema, definimos um vetor $\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$ para representar tais pesos e uma função $g_i(x)$ que devolve o peso do termo k_i em um vetor de pesos x.

2.4.1 Modelo Booleano

O modelo booleano de recuperação de informação é baseado na álgebra booleana. Neste modelo, dizemos simplesmente que um termo está presente ou ausente em um documento e assim, o peso de cada termo é binário, ou seja, $w_{i,j} \in \{0, 1\}$. Além disso, uma consulta tem formato de uma expressão booleana, a qual é composta por palavras que são ligadas pelos operadores booleanos *e*, *ou* e *não* e pode ser representada na forma normal disjuntiva $\vec{q}_{dnf}$.

Sendo $\vec{q}_{cc}$ um componente de $\vec{q}_{dnf}$, [BYRN+99] define a similaridade de um documento d_j a uma consulta q como:

$$sim(d_j, q) = \begin{cases} 1 & \text{Se } \exists \vec{q_{cc}} \mid (\vec{q}_{cc} \in \vec{q}_{dnf}) \wedge (\forall k_i, g_i(\vec{d_j}) = g_i(\vec{q}_{cc}) \\ 0 & \text{Caso contrário.} \end{cases}$$

Considere, por exemplo, o documento 1 da tabela 1 *"The old night keeper keeps the keep in the town."* e a consulta $q = old \wedge (night \vee \neg big)$. Podemos escrever q na forma normal disjuntiva como $\vec{q}_{dnf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)$, na qual cada componente é um vetor de pesos associado à tupla $(old, night, big)$. Assim, como $\vec{d_1} = (1,1,0)$ e existe $\vec{q_{cc}} = (1,1,0)$, temos $sim(d_1, q) = 1$, ou seja, o documento 1 é relevante para a consulta q.

Apesar da simplicidade do modelo, a similaridade booleana não permite a ideia de parcialidade, ou seja, um documento é relevante ou irrelevante.

2.4.2 Modelo Vetorial

Considerando a similaridade booleana limitada, o modelo vetorial propõe a ideia de parcialidade permitindo valores não booleanos aos pesos dos termos. Assim, podemos avaliar o grau de relevância do termo.

Neste modelo, além dos pesos em relação aos documentos, é necessário calcular o grau de relevância do termo nas consultas, definindo então, o vetor $\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{t,q})$, no qual t é o número total de índices do sistema.

De acordo com [BYRN+99], a similaridade entre um documento e uma consulta pode ser calculada como a correlação entre os vetores d_j e q e quantificada pelo cosseno do ângulo formado por eles:

$$sim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}}$$

Existem diversas maneiras de calcular o peso de um termo. Uma delas é baseada na frequência do mesmo: $w_{i,j} = f_{i,j} \times idf_i$.

• **fator *tf*:** Sendo $freq_{i,j}$ a frequência do termo k_i no documento d_j:

$$f_{i,j} = \frac{freq_{i,j}}{max_l freq_{l,j}}$$

• **fator *idf*:** frequência inversa dos documentos; Sendo N o número total de documentos e n_i a quantidade de documentos em que o termo k_i aparece, temos:

$$idf_i = log \frac{N}{n_i}$$

Já em relação à consulta, o peso de um termo pode ser calculado por meio da fórmula

$$w_{i,q} = (0.5 + \frac{0.5 freq_{i,q}}{max_l freq_{l,q}}) \times log \frac{N}{n_i}.$$

A grande vantagem do modelo vetorial é a possibilidade de calcular o grau de relevância de um documento em relação à consulta do usuário e, assim, ordenar o resultado de acordo com a similaridade.

3 Alternativas tecnológicas

3.1 Apache Lucene

O *Lucene* [Luc] é um projeto de código aberto da Apache implementado em Java. Trata-se de um biblioteca robusta de Recuperação de Informação que possibilita a indexação e busca de maneira eficiente e simples.

Basicamente, o *Lucene* pode ser dividido em dois grupos de classes: as de indexação e as de busca [HG04].

No grupo de indexação encontram-se, por exemplo, a classe *IndexWriter*, responsável pela criação e manutenção dos índices, a *Analyzer*, que possibilita a configuração das operações em texto descritas na seção 2.2 e a *Field* que indica como cada parte de um documento deve ser tratada.

Entre as classes de busca estão a *IndexSearcher*, responsável pela busca nos índices, a *Query*, que indica o tipo de consulta e a classe *Hits*, que representa o resultado de uma consulta.

3.1.1 Índices

O *Lucene* utiliza a estrutura de arquivos invertidos. A biblioteca permite que o índice seja mantido inteiramente na memória, por meio da classe *RAMDirectory* ou armazenado em disco, utilizando a *FSDirectory*.

Em ambos os casos, o índice é formado por um conjunto de arquivos que contêm informações sobre os termos. Como o *Lucene* suporta diferentes tipos de consultas, é necessário armazenar diversos dados como, por exemplo, a quantidade de documentos em que um termo aparece, quais são esses documentos, a frequência em cada um e também as posições de ocorrência.

3.1.2 *Score*

Para cada documento presente no resultado de uma busca é atribuido um *score* que representa a similaridade de tal documento com a consulta. Esse valor é calculado por meio da fórmula:

$$s(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{\text{t in d}} (tf(\text{t in d}) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t,d))$$

Fator	**Significado**
coord(q,d)	Quantos termos da consulta q aparecem no documento d.
queryNorm(q)	Utilizado para fazer comparações entre consultas. Não afeta o ranqueamento.
tf(t in d)	Frequência do termo t no documento d.
idf(t)	Número de documentos que contêm o termo t.
t.getBoost()	Peso atribuído ao termo t no momento da busca.
norm(t,d)	Encapsula alguns fatores de *boost* definidos no momento da indexação.

Mais detalhes do cálculo podem ser encontrados em [Sim].

3.2 Google

A *search engine* da Google foi desenvolvida com o objetivo de criar algo que "entenda exatamente o que você diz e que te devolva exatamente o que você quer", de acordo com Larry Page [Goo].

O servidor de índices utilizado pela empresa segue a ideia de arquivos invertidos, analisando o conteúdo dos textos e criando uma lista de documentos que contenham determinada palavra.

3.2.1 PageRank

A inovação do sistema da Google está na maneira de avaliar a importância de uma página na *Web*. Um dos fatores utilizados no cálculo da relevância de um documento em relação a uma consulta é chamado de *PageRank* e é obtido a partir da estrutura de *links* utilizada na Internet. A definição desse fator é encontrada em [BP98] :

> *Considere que as páginas $T_1, T_2, ..., T_n$ possuam links para a página A e que $C(X)$ seja o número de links presentes na página X. O PageRank de A é definido por:*
>
> $$PR(A) = (1-d) + d\sum_{i=1}^{n} \frac{PR(T_i)}{C(T_i)}$$
>
> *O parâmetro d é um fator de "amortecimento", entre 0 e 1, geralmente definido como 0,85.*

Nesse mesmo artigo, os autores dizem que o cálculo pode ser entendido ao se pensar no comportamento de um usuário que vai seguindo *links* nas páginas e em um determinado momento cansa e reinicia o processo a partir de uma outra página qualquer. A probabilidade do usuário chegar em uma página A é $PR(A)$ e a probabilidade dele se cansar enquanto estiver visitando a página A é d. O uso do fator de "amortecimento" é importante para evitar que o sistema seja enganado por páginas que objetivam melhor classificação no *PageRank*.

Outra abstração para o cálculo é a ideia de uma votação (um voto é representado por um *link*), exemplificada na figura 6. Quanto mais votos significantes uma página recebe, mais importante ela é. O peso de um voto é definido pela importância de quem está votando.

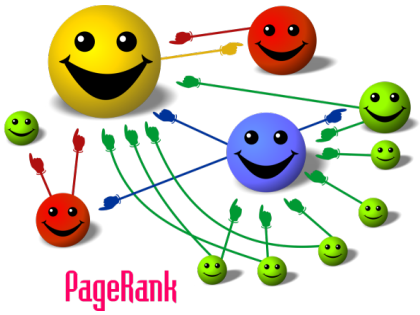

Figura 6: Cada bola representa uma página e o tamanho, sua importância. (Fonte: Wikipedia)

4 Estudo de caso: CEGH

O Centro de Estudos do Genoma Humano (CEGH) [ceg] possui um sistema *web* desenvolvido pelo grupo de banco de dados do IME-USP para cadastro de pacientes, consultas, solicitação de exames entre outras funcionalidades.

Para o desenvolvimento, utilizamos a linguagem Ruby, o *framework* Ruby on Rails e o gerenciador de banco de dados relacional PostgreSQL. O banco contém diversas entidades e relacionamentos mas, neste estudo de caso, utilizaremos apenas os descritos abaixo:

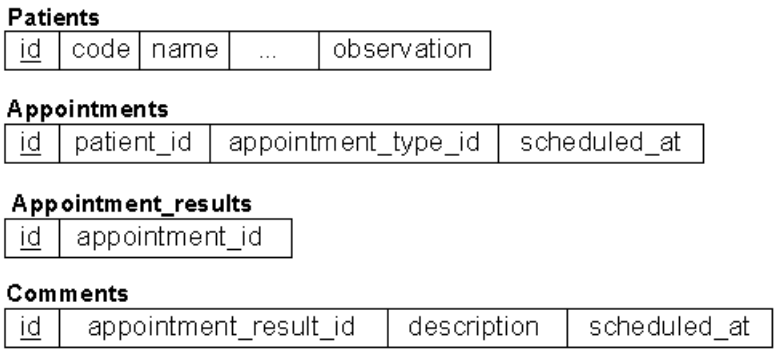

Figura 7: Entidades do banco de dados. (Alguns atributos irrelevantes a este trabalho foram omitidos)

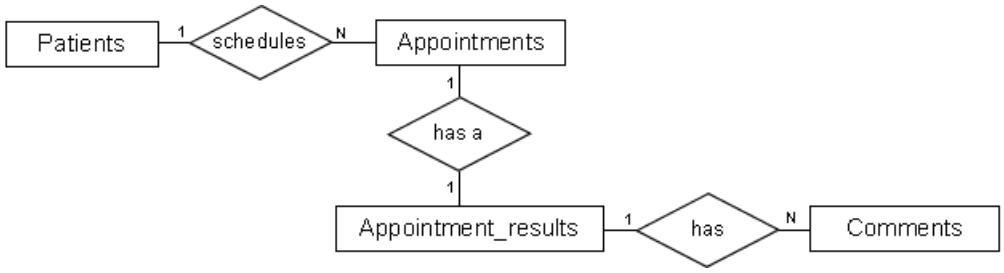

Figura 8: Diagrama entidade relacionamento.

Ao realizar o cadastro de um paciente, os dados pessoais são armazenados na entidade "Patients" e quaisquer observações podem ser registradas no atributo "observation". Tendo o paciente cadastrado, podemos, então, agendar consultas médicas para o mesmo e, durante o atendimento, o médico pode realizar diversas anotações que serão salvas na entidade "Comments", no campo "description".

Para atender as exigências da dinâmica de atendimentos do CEGH, os atributos "observation" e "description" são do tipo texto e recebem dados sem estrutura definida. Com a necessidade de realizar busca nesses campos, integramos a biblioteca *Ferret* [Bal], implementada em *Ruby* e inspirada no *Apache Lucene*, e também o plugin *Acts as Ferret*.

4.1 Ferramentas e implementações

No sistema do CEGH, uma "Anotação de Consulta" é formada, entre outros dados, por um conjunto de comentários. Apesar destes serem individualmente armazenados na entidade "Comments", ao realizar uma busca, o usuário espera como resposta uma tupla da entidade "Appointment_results", que representa a anotação de consulta, ou então da entidade "Patients", caso os termos da consulta apareçam na observação. Assim, o primeiro passo antes da indexação foi a inclusão do atributo *all_comments* no modelo "Appointment_results" para unir os comentários.

Utilizando o plugin *Acts as Ferret* criamos um índice único para os dados dos atributos *observation* e *all_comments*.

```
ActsAsFerret::define_index('obs',
  :models => {
     Patient  => {:fields => [:observation]},
     AppointmentResult => {:fields => [:all_comments]},
 },
 :ferret   => {
    :default_fields => [:observation, :all_comments],
    :analyzer => Analyzer.new,
    :store_class_name => true
 }
)
```

Figura 9: Arquivo aaf.rb - Configuração de índice único.

Inspirado no *Lucene*, a biblioteca *Ferret* é dividida em módulos de acordo com a função. Segue uma breve descrição da utilização no sistema do CEGH.

- **Analyses:** especificação das operações que serão aplicadas nos textos;
- **QueryParser:** recebe a consulta do usuário e identifica o tipo de busca;
- **Search:** recuperação de documentos.

4.2 Resultados obtidos

Com o *Ferret*, implementamos uma página de busca que devolve uma lista de resultados encontrados ordenados de acordo com sua relevância. A biblioteca permite diversos tipos de consultas:

Consulta: fendas AND parciais **Tradução do *QueryParser*:** +(observation:fendas all_comments:fendas) +(observation:parciais all_comments:parciais)

Exemplo 1: Consulta utilizando o operador booleano AND.

Consulta: gravidez NOT cesária **Tradução do *QueryParser*:** (observation:gravidez all_comments:gravidez) -(observation:cesária all_comments:cesária)

Exemplo 2: Consulta utilizando o operador booleano NOT.

Consulta: observation:DMC **Tradução do *QueryParser*:**(observation:dmc)

Exemplo 3: Consulta no campo *observation*.

4.3 Análise dos Resultados

Com aproximadamente 15 mil pacientes cadastrados (nem todos possuem alguma observação cadastrada) e 6 mil comentários feitos em anotações de consultas, realizamos testes comparativos à busca SQL.

O gráfico da figura 6 exibe o tempo médio de busca e também a comparação entre diferentes tipos de consulta.

Teste 1: busca por palavra rara na coleção. **Teste 2:** busca por palavra frequente na coleção.

Tipos de testes

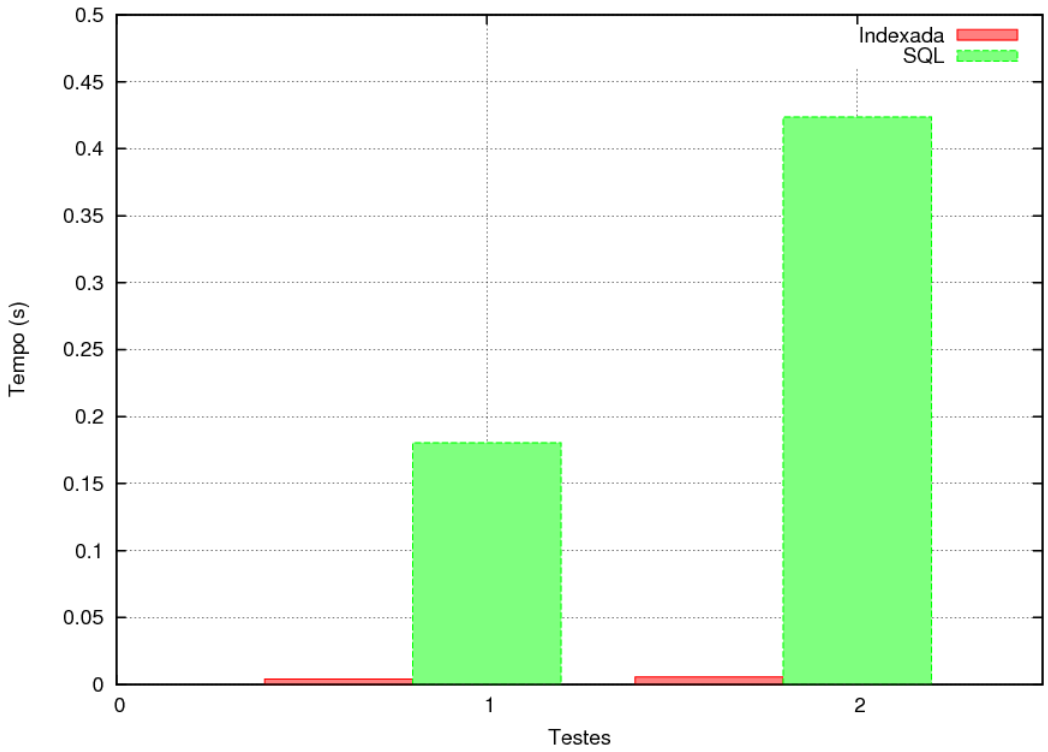

Figura 10: Gráfico de comparação. SQL x Busca Indexada

No gráfico, podemos notar a significativa melhora de desempenho. Podemos observar também que a diferença entre os tempos de busca indexada para os testes 1 e 2 é mínima, porém, quanto mais frequente a palavra, mais demorada será a consulta SQL.

5 Conclusão

Com este trabalho pudemos entender a importância de cada passo do processo de Recuperação de Informação. Vimos que existem diversas técnicas e maneiras de implementá-los e com isso, podemos personalizar o sistema de acordo com as necessidades da aplicação objetivando melhores resultados.

Percebemos também, inclusive empiricamente, que a utilização de índices é imprescindível para a busca textual. Pela seção 4.3, a melhora de desempenho é clara e, com os testes realizados, podemos notar que, com o crescimento do volume de dados, a busca SQL ficaria ainda pior.

Além da melhoria no desempenho, o uso de técnicas de RI permite novas funcionalidades como a avaliação de relevância dos documentos e dos diversos tipos de consulta. Neste estudo, a partir da distinção entre dado e informação, percebemos o quanto a noção de relevância pode afetar a listagem dos resultados. Quando trata-se de um banco volumoso, muitas vezes um documento que possui todas as palavras da consulta não é de interesse do usuário.

Referências

[Bal] Dave Balmain. Ferret. `http://www.davebalmain.com`.

[BP98] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 1998.

[BTO+97] E. Bertino, K.L. Tan, B.C. Ooi, R. Sacks-Davis, J. Zobel, and B. Shidlovsky. *Indexing techniques for advanced database systems*. Kluwer Academic Publishers Norwell, MA, USA, 1997.

[BYRN+99] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*. Addison Wesley Reading, MA, 1999.

[ceg] Cegh. `http://genoma.ib.usp.br`.

[FBY92] W.B. Frakes and R. Baeza-Yates. *Information retrieval: data structures and algorithms*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1992.

[Goo] Technology overview. `http://www.google.com/corporate/tech.html`.

[HG04] E. Hatcher and O. Gospodnetic. *Lucene in action*. Manning Publications, 2004.

[Luc] Apache lucene. `http://lucene.apache.org`.

[Sim] Lucene - class similarity. `http://lucene.apache.org/java/2_3_2/api/org/apache/lucene/search/Similarity.html`.

[ZM06] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys (CSUR)*, 2006.

Parte II

Subjetiva

1 Desafios e Frustrações

Comecei a trabalhar no desenvolvimento do sistema do CEGH em setembro de 2008 e no início de 2009 fiquei responsável por ele. Foi um grande desafio e pude aprender como organizar e desenvolver um projeto grande e também mais sobre *Ruby on Rails*. Além do aprendizado "técnico", adquiri experiência em reuniões com os usuários e por meio delas, tive a dimensão da importância do projeto.

O sistema é robusto e, além de conter muita informação sobre os pacientes, possui conhecimento sobre as doenças estudadas no centro. Em conversas com o Jef, percebemos que não era suficiente apenas guardar informações, o próximo passo seria recuperá-las.

A partir da necessidade do CEGH, definimos o tema deste trabalho e partimos para o estudo do processo de RI, enfatizando as técnicas de indexação e os modelos mais utilizados. Procuramos também uma biblioteca para integrar ao sistema do CEGH e encontrar no *Lucene* a implementação de tudo que havíamos estudado foi motivante.

A área de RI é extensa e durante o estudo encontramos muitos tópicos interessantes, porém, tivemos que escolher os que eram mais importantes para este trabalho e não pudemos nos aprofundar tanto nos outros. Além disso, não conseguimos iniciar o estudo sobre busca semântica (analisar o significado das palavras para encontrar documentos relevantes) e acredito que essa seja minha maior frustração. Em contrapartida, saber que meu trabalho agregará uma nova funcionalidade ao sistema do CEGH e que ela será de fato utilizada é gratificante.

2 Disciplinas revelantes para o trabalho

MAC 110 - Introdução à Computação,
MAC 122 - Princípios de Desenvolvimento de Algoritmos

Tive um contato superficial com programação antes do curso, mas estas matérias foram fundamentais para o aprendizado ao longo da graduação.

MAC 323 - Estrutura de Dados

Aprendi muito nesta disciplina. Considero-a uma das mais importantes e para este trabalho, foi imprescíndivel ter um conhecimento prévio de estruturas como lista, árvore e *hash*.

MAC 426 - Sistemas de Bancos de Dados

Obviamente, matéria fundamental para este trabalho.

MAC 342 - Laboratório de Programação Extrema

Além de ter utilizado *Rails* no projeto desta disciplina, ela me ajudou muito no sentido de organizar e desenvolver um sistema grande. Foi importante também pois tivemos contato direto com os usuários e acredito que foi uma boa experiência.

3 Futuro

A área de RI é extensa e diversificada. Restringindo a textos, uma possibilidade para continuação deste trabalho é o estudo de busca semântica, tema o qual acho muito interessante.

Este trabalho e esse possível caminho contribuiu para minha decisão de prosseguir os estudos e aprimorar o sistema do CEGH. Tendo esses objetivos, pretendo ingressar no programa de mestrado do IME logo após o término da graduação.