Estudo de Tecnologias de Busca na *Web*

Eder Eustáquio Alves

Trabalho Final de Mestrado Profissional

Instituto de Computação
Universidade Estadual de Campinas

Estudo de Tecnologias de Busca na *Web*

Eder Eustáquio Alves

Julho de 2004

Banca Examinadora:

- Profa. Dra. Anamaria Gomide (Orientadora)
  Instituto de Computação – UNICAMP
- Prof. Dr. João Paulo Kitajima
  Allelyx Applied Genomics S. A.
- Profa. Dra. Heloisa Vieira da Rocha
  Instituto de Computação – UNICAMP
- Profa. Dra. Célia Picinin de Mello
  Instituto de Computação – UNICAMP

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Alves, Eder Eustáquio

AL87e Estudo de tecnologias de busca na Web / Eder Eustáquio Alves -- Campinas, [S.P. :s.n.], 2004.

Orientadores : Anamaria Gomide, Jorge Stolfi

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Sistemas de recuperação da informação. 2. Recuperação da informação. 3. Engenharia de software. I. Gomide, Anamaria. II. Stolfi, Jorge. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Estudo de Tecnologias de Busca na *Web*

Este exemplar corresponde à redação final do Trabalho Final devidamente corrigida e defendida por Eder Eustáquio Alves e aprovada pela Banca Examinadora.

Campinas, 16 de Julho de 2004

Profa. Dra. Anamaria Gomide (Orientadora)

Prof. Dr. Jorge Stolfi (Coorientador)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para obtenção do título de Mestre em Computação na Área de Engenharia de Software.

TERMO DE APROVAÇÃO

Trabalho Final escrito defendido e aprovado em 16 de julho de 2004, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. João Paulo Fumio Whitaker Kitajima
Allelyx Applied Genomics S. A.

Profª. Drª. Heloísa Vieira Rocha
IC - UNICAMP

Profª. Drª. Célia Picinin de Mello
IC - UNICAMP

Profª. Drª. Anamaria Gomide
IC - UNICAMP

Resumo

O objetivo deste trabalho é estudar as tecnologias de busca na *Web*, enfocando principalmente os algoritmos utilizados. Iniciamos com um breve sumário da evolução dos mecanismos de busca, seguido de uma descrição detalhada de seus principais componentes e algoritmos. Em particular, estudamos dois importantes algoritmos de análise de links: o PageRank (utilizado pelo mecanismo Google) e o HITS (utilizado pelo mecanismo Teoma). Estudamos também em detalhes um exemplo específico, o mecanismo de busca comercial Google. Por fim, relatamos um experimento, conduzido por nós, para avaliar o grau de cobertura da *Web*, alcançado pelo Google.

Abstract

The aim of this work is to study the Web search technologies, focusing mainly on algorithms used. Firstly, we present a brief history of the search engines, followed by a detailed description of a search engine main components and algorithms. Particularly, we studied two important link analysis algorithms: the PageRank (used by Google) and HITS (used by Teoma). We also studied the details of real implementation of the commercial search engine Google. Finally, we describe an experience, performed by us, to check how much of the Web is indexed by Google.

Dedicatória

Dedico este trabalho a minha esposa, Maria Abadia, pelo incentivo e paciência durante todo o mestrado. Esta vitória pertence a nós dois.

Agradecimentos

Agradeço à Professora Anamaria Gomide pela dedicação na orientação deste trabalho, pelo seu respeito e confiança em mim depositados.

Ao Professor Jorge Stolfi pelos inúmeros comentários e sugestões durante a elaboração do trabalho.

Aos colegas do Mestrado Profissional pela agradável convivência durante o curso.

Conteúdo

Índice de Figuras

Capítulo 1 – Introdução

Com a popularização da *World-Wide Web*, os mecanismos de busca se transformaram em importantes ferramentas de trabalho. Muitos usuários da *Web* iniciam sua atividade através de uma busca por palavras-chave em um mecanismo de busca, recebendo como resultado uma lista de *sites* relevantes.

Tradicionalmente, os mecanismos de busca de documentos utilizam técnicas e algoritmos clássicos de Recuperação de Informação (RI) [9]. Porém, essas técnicas foram desenvolvidas para coleções relativamente pequenas e coerentes. A *Web*, pelo contrário, possui um volume imenso de informações, altamente heterogêneo, com uma alta taxa de mudança, e distribuído geograficamente. Desta forma, tornou-se necessária a criação de novas técnicas, ou extensão das técnicas tradicionais, para permitir a coleta eficiente das informações, a criação de índices escaláveis e a melhoria da precisão dos resultados de busca.

Além de apresentar uma grande taxa de crescimento, a *Web* se tornou, a partir da segunda metade da década de 90, cada vez mais comercial. Ao mesmo tempo, muitas das pesquisas em torno dos mecanismos de busca migraram do meio acadêmico para o meio comercial. Em conseqüência, o desenvolvimento de novos mecanismos de busca tem sido realizado principalmente em empresas, que raramente publicam os detalhes técnicos [10]. Desta forma, torna-se importante trazer as discussões em relação às tecnologias de busca na *Web* para o meio acadêmico.

O objetivo deste trabalho é estudar as tecnologias de busca na *Web*, enfocando principalmente os algoritmos utilizados. Como parte deste estudo, realizamos um experimento para avaliar o grau de cobertura da *Web* alcançado por um mecanismo de busca comercial.

O capítulo 2 apresenta um pouco da história dos mecanismos de busca na Internet, destacando os principais acontecimentos até os dias atuais. No capítulo 3, é apresentada uma visão geral dos principais componentes de um mecanismo de busca e são discutidos aspectos algorítmicos relacionados a cada componente. O capítulo 4 aprofunda a discussão dos algoritmos de classificação por análise de *links*, detalhando dois algoritmos importantes (PageRank e HITS). No capítulo 5, é apresentada a arquitetura de um mecanismo de busca comercial. Já no capítulo 6, descrevemos nosso experimento para medir o grau de cobertura da Web pelo Google.

Capítulo 2 - Um pouco de história dos mecanismos de busca na Internet

Neste capítulo será apresentada, em ordem cronológica, um pouco da história dos mecanismos de busca na Internet, destacando os principais acontecimentos até os dias atuais. Como será visto a seguir, os primeiros mecanismos de busca na Internet surgiram ainda antes da *Web*, através de serviços de busca para sites FTP. Mais tarde, com a popularização da *Web*, os mecanismos de busca alcançam a maturidade através de sistemas com grande poder de processamento e cobertura da *Web*, como por exemplo o AltaVista. Por fim, devido a um certo nivelamento em relação a poder de processamento e cobertura, entram em cena sistemas voltados a aumentar a relevância das respostas, como por exemplo o Google, e sistemas com serviços de busca adicionais, como o AllTheWeb. Adicionalmente, começam também a surgir mecanismos específicos voltados para um escopo limitado em termos geográfico ou de assunto.

2.1. Os primeiros mecanismos de busca

O serviço de diretório Archie [1] foi o primeiro mecanismo de busca da Internet. Ele combinava um buscador de dados baseado em *scripts* para criar listagens do conteúdo de *sites* de FTP anônimo, usando expressões regulares para recuperar nomes de arquivo fornecidos pelo usuário. Ele entrou em operação em 1990 e em 1992 já era uma ferramenta popular na Internet.

O sucesso do Archie como indexador de arquivos FTP inspirou a criação de um índice para os menus Gopher, que eram o meio de compartilhamento de arquivos texto pela Internet. Um grupo de pesquisa na Universidade de Nevada desenvolveu o sistema Veronica [1] em 1993.

2.2. Os mecanismos de busca na *Web*

O primeiro sistema de busca para a *Web* foi o "World Wide Web Wanderer" [1]. Inicialmente, o sistema apenas contava o número de servidores *Web* na Internet, mas foi adicionado a ele mais tarde um programa de busca chamado Wandex.

Em outubro de 1993, foi desenvolvido um sistema de busca análogo ao Archie para buscas na *Web*, o Aliweb [1] (Archie-Like Indexing for the *Web*). O Aliweb exigia que cada servidor *Web* contruísse um índice das páginas de seu *site* e que se registrasse no Aliweb. O sistema fornecia um programa desenvolvido em Perl que realizava buscas nos índices fornecidos pelos autores dos sites.

Em dezembro de 1993, mais três mecanismos de busca na Internet foram disponibilizados: o Jumpstation, o "World Wide Web Worm" e o RBSE (Repository-Based Software Enginnering) Spider [1]. O Jumpstation usava um robô *Web*, ou *spider*, para buscar informações sobre o título e os cabeçalhos das páginas *Web* e utilizava um mecanismo de busca simples para recuperar as páginas. O WWW Worm criava índices dos títulos e das URLs, permitindo busca por expressão regular no índice invertido. Ambos os sistemas produziam uma listagem de documentos encontrados na ordem do banco de dados, sem nenhum tipo de classificação baseada na expressão de busca do usuário. Em contraste, o RBSE Spider e o WebCrawler (que entrou em operação em 20 de abril de 1994) foram os primeiros mecanismos de busca a implementarem as respostas baseadas em classificação de relevância, retornando primeiro os documentos mais relacionados a expressão de busca do usuário.

2.3. Os mecanismos de busca alcançam a maturidade

Em abril de 1994, dois estudantes da Universidade de Stanford, David Filo e Gerry Yang criaram uma coleção de páginas que se tornou bastante popular. Foi chamada "Yahoo" [2]. Enquanto o número de *links* crescia e suas páginas começavam a receber milhares de acessos por dia, foram criados meios de melhorar a organização dos dados. Para ajudar na pesquisa de documentos, o "Yahoo" foi transformado em um diretório com capacidade de busca. Como os *links* são adicionados e categorizados manualmente, o "Yahoo" não era

considerado um mecanismo de busca, mas sim um serviço de diretório. Desde então o "Yahoo" automatizou alguns aspectos do processo de busca e classificação de informações, tornando mais difícil a distinção entre mecanismo de busca e diretório.

Em 20 de julho de 1994, o Lycos [1] entrou em operação com um catálogo de 54.000 documentos. Entre suas características estava a classificação por relevância, computada através de parâmetros como a quantidade dos termos de busca presentes no documento (para buscas de mais de uma palavra), quantidade de ocorrências repetidas do termo no mesmo documento, proximidade da ocorrência do termos (para buscas de mais de uma palavra) e posição da ocorrência no documento. Mas o grande diferencial foi o tamanho do catálogo. Em agosto de 1994 ele havia identificado 394.000 documentos; em Janeiro de 1995 o catálogo havia alcançado 1,5 milhão de documentos; e em novembro de 1996, o Lycos havia indexado mais de 60 milhões de documentos – mais que qualquer outro mecanismo de busca da época.

No final de 1994 entra em operação o Infoseek [2]. Inicialmente era apenas mais um mecanismo de busca que usava conceitos do "Yahoo!" e o Lycos. Mas, devido a uma boa interface com o usuário (oferecia um diretório de busca e serviços adicionais como canais de notícias) e um acordo com a Netscape em dezembro de 1995 ele se tornou um dos mecanismos de busca mais famosos. Por acordo, o Infoseek se tornou o mecanismo de busca padrão do *browser* Netscape, acionado pelo botão "Net Search". Antes disso, o "Yahoo!" era o serviço padrão do Netscape.

Em dezembro de 1995, a DEC (Digital Equipment Corporation) lança o mecanismo de busca AltaVista [2]. Ele possuia um conjunto de funcionalidades que rapidamente o levaram ao topo de popularidade. Além disso, possuia o poder de processamento das máquinas DEC Alpha que permitia o processamento de milhões de buscas por dia sem atraso significativo nas respostas. Suas novas funcionalidades tiveram grande impacto na tecnologia de mecanismos de busca. O AltaVista foi o primeiro a utilizar buscas em linguagem natural, o que significa que o usuário poderia buscar utilizando a sentença "What is the weather like in Tokyo?" e não receber milhões de páginas com a palavra

"What". Além disso, foi o primeiro a permitir buscas avançadas, como por exemplo o uso de operadores booleanos (*and, or, not*) e de proximidade (*near*).

Em maio de 1996, o mecanismo de busca HotBot [2] foi lançado pela empresa Inktomi. Ele foi rapidamente licenciado pela revista Wired para ser utilizado em seu *web site*, o HotWired. Isso fez crescer rapidamente a popularidade do HotBot. Na época ele era o mais poderoso dos mecanismos de busca, com capacidade para indexação de 10 milhões de páginas por dia, podendo refazer sua base de dados inteira em dias o que contribuía para diminuir o número de respostas de busca desatualizadas. Além disso ele fazia uso intensivo da tecnologia de *cookies* para registrar preferências de busca dos usuários.

2.4. Últimas novidades

Em setembro de 1998, outros dois estudantes de pós-graduação da Universidade de Stanford, Larry Page e Sergei Brin, lançam em versão beta o mecanismo de busca Google [3]. Ele ganha popularidade rapidamente e já em fevereiro de 1999 processa cerca de 500.000 buscas por dia. Em setembro de 1999 passa de versão beta para a oficial e continua a ganhar popularidade até se tornar o principal mecanismo de busca da *Web*. Entre suas características estão a classificação por relevância baseada na análise dos *links*, *cache* de páginas e alta capacidade de indexação. Em agosto de 2003, o índice do Google havia atingido 3.3 bilhões de documentos, o maior entre todos os mecanismos de busca [4].

Em agosto de 1999, a empresa norueguesa FAST Search lança o mecanismo de busca All TheWeb [5]. Entre as sua características estão a grande capacidade de indexação, com o tamanho de seu índice da mesma ordem do índice do Google [4], e serviços adicionais de busca de notícias, imagens, arquivos de áudio e arquivos de FTP. Atualmente, é um dos principais competidores do Google no mercado de mecanismos de busca genéricos.

Em abril de 2000, é lançado o mecanismo de busca Teoma [6]. Ao contrário de seus concorrentes como o Google e o AllTheWeb, ele não possui o mesmo poder de indexação, mas tenta concentrar esforços no aumento da relevância das respostas de busca fornecidas.

Ao invés de fazer a análise de relevância pela quantidade de páginas que contém *links* para a página em questão - como feito pelo Google – ele criou o conceito de Popularidade Relacionada ao Tema (*Subject-Specific Popularity*), que se baseia na quantidade de links oriundos de páginas sobre o mesmo tema.

2.5. Novas tendências

Com a competição pelos mecanismos de busca genéricos dominada por grandes empresas como Google, AltaVista e "Yahoo", novas empresas têm tentado ganhar espaço criando mecanismos específicos, diminuindo o escopo em termos do tema ou localização geográfica e, desta forma, tentando ganhar um nicho de mercado. Como exemplo de mecanismo para um tema específico, pode-se citar o CiteSeer [7] que é especializado em artigos de literatura científica e é capaz de extrair índices de citações, encontrando artigos citados através da análise de texto do próprio artigo. Já em termos de localização geográfica, existem diversos *sites* especializados em buscas dentro de um país ou língua específica, como por exemplo o TodoBR [8], que cria seu índice através dos sites ".br" e é customizado para buscas em português.

Capítulo 3 – Arquitetura dos mecanismos de busca

Neste capítulo, será apresentada uma visão geral, baseada no artigo de Arvind Arasu et al [9], dos principais componentes de um mecanismo de busca e serão discutidos aspectos algorítmicos relacionados a cada componente.

3.1. Visão geral

A Figura 1 mostra um esquema genérico de um mecanismo de busca [9].

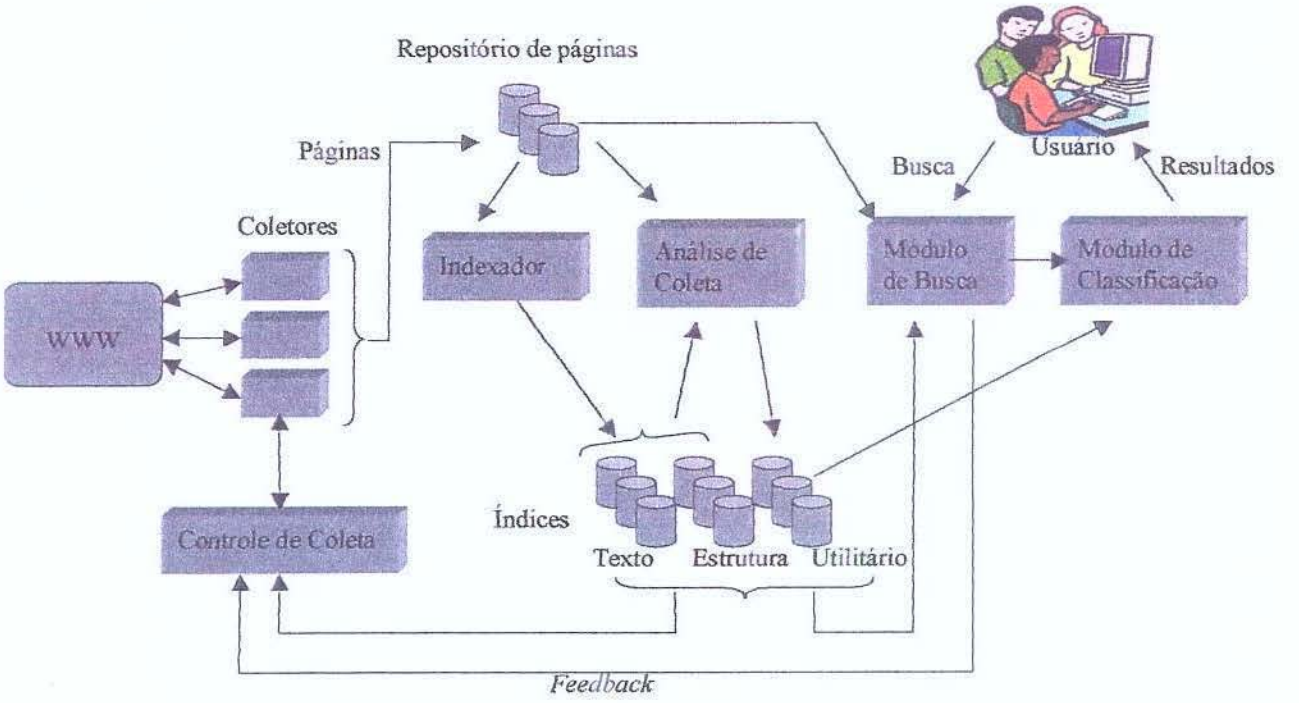

Figura 1 - Arquitetura genérica de um mecanismo de busca

Um mecanismo de busca geralmente possui um *módulo coletor* (também chamado *crawler*, robô ou *spider*) encarregado de obter as páginas a serem indexadas. Os coletores são programas que navegam pela *Web* coletando as páginas a partir de um conjunto de URLs. Eles extraem as URLs das páginas lidas e passam essa informação ao módulo de controle

de coleta. Este módulo determina quais *links* deverão ser visitados pelos coletores em seguida. Os coletores também armazenam as páginas coletadas no repositório de páginas.

O algoritmo básico dos coletores pode ser modificado de várias maneiras para permitir diferentes estratégias baseadas em cobertura ou em tópicos específicos. Por exemplo, os coletores em um determinado mecanismo de busca podem ser direcionados a visitarem o maior número possível de *sites*, deixando de lado as páginas mais periféricas em cada *site*. Já os coletores em um mecanismo de busca específico podem ser direcionados para um certo domínio, por exemplo páginas governamentais. O módulo de controle de coleta é o responsável por direcionar a operação dos coletores.

Os coletores recebem as URLs a buscar do módulo de controle de coleta, que por sua vez otem essa informação como subproduto da formação dos índices. Esse módulo pode utilizar um grafo dos *links* para decidir quais *links* os coletores deverão buscar em seguida. O módulo de controle de coleta pode também se guiar por *feedback* extraído dos padrões de busca do usuário para alimentar o processo de coleta (conexão entre "Módulo de Busca" e "Controle de Coleta" na Figura 1).

O *módulo de indexação* extrai todas as palavras de cada página e guarda a URL e a posição do texto onde cada palavra ocorreu. O resultado é geralmente uma tabela de busca bastante grande, o *índice de texto*, que pode fornecer todas as URLs que apontam para páginas onde uma certa palavra ocorreu. Essa indexação das páginas é uma tarefa difícil, devido ao tamanho e à taxa de mudança da base de páginas indexadas.

Em adição às dificuldades quantitativas, há a necessidade de construção de índices especializados para representar a estrutura da *Web*. Por exemplo, pode-se construir um índice estrutural baseado nos *links* entre as páginas. O *módulo de análise da coleta* é o responsável pela criação destes índices adicionais.

Outros índices utilitários podem ser criados pelo módulo de análise da coleta, por exemplo, para fornecer acesso a páginas de um determinado tamanho, páginas de uma certa importância, ou páginas que apresentem uma certa quantidade de imagens.

Durante as fases de coleta de páginas e indexação, os mecanismos de busca devem guardar as páginas que recuperaram da *Web* num *repositório de páginas*. Este repositório pode ser mantido permanentemente, mesmo depois que a página foi indexada, para servir como um *cache* das páginas. Esse *cache* é um serviço adicional que ajuda a aumentar a satisfação do usuário, através do aumento de velocidade de fornecimento das páginas de resultado e da possibilidade de acesso a páginas que "saíram do ar".

O *módulo de busca* é o responsável por receber e processar os pedidos de busca dos usuários. Para isso, ele depende dos índices (para localizar as páginas) e do repositório de páginas (para mostrar o sumário das páginas). Devido ao tamanho da *Web* e ao fato de que os usuários normalmente fornecem uma ou duas palavras de busca, o conjunto de páginas de resposta tende a ser muito grande. O *módulo de classificação* tem a tarefa de classificar os resultados de forma que as primeiras páginas apresentadas sejam aqueles mais relevantes para o usuário. Este módulo é de interesse especial, porque as técnicas tradicionais de Recuperação de Informação (RI) têm se revelado inadequadas quando aplicadas sem modificação à busca na *Web*. A maioria das técnicas tradicionais de RI se baseiam na similaridade entre o texto da consulta do usuário e o texto das páginas. Porém devido ao volume de páginas da *Web*, essas técnicas não conseguem filtrar, de maneira satisfatória, os resultados irrelevantes. Como será detalhado adiante, há algoritmos que fazem uso da análise de *links* entre as páginas. Esses algoritmos, em conjunto com as técnicas tradicionais de RI melhoram sensivelmente a precisão dos mecanismos de busca.

3.2. Coletando páginas *Web*

Como mostrado na Figura 1, o módulo coletor, junto com os módulos controle de coleta e indexador, são responsáveis pela obtenção das páginas da *Web* a serem indexadas. O módulo de coleta tipicamente recebe um conjunto de URLs a serem coletadas, mantidas em

uma fila e priorizadas. Desta fila, é retirada uma URL, página referenciada é copiada da *Web* e enviada ao indexador. Este extrai todas URL presentes na página coletada e as coloca na fila de URLs. Esse processo continua até que o coletor decida parar. Algumas questões importantes são levantadas:

- Quais páginas devem ser coletadas? Devido ao tamanho da *Web* e à taxa de alteração, o coletor não é capaz de coletar todas as páginas existentes. Mesmo os mecanismos de busca com maior cobertura indexam apenas uma fração da *Web*. Desta forma, é importante que o coletor escolha cuidadosamente as páginas e visite primeiro as páginas mais importantes, de forma que a fração da *Web* coletada seja a mais significativa possível.

- Como o coletor deve atualizar as páginas? Depois da coleta de um número significativo de páginas, o coletor deve começar a revisitar as páginas para detectar mudanças no conteúdo das mesmas. Em particular, o coletor deve ter um mecanismo para detectar e remover as páginas obsoletas, já que o mecanismo de busca não é notificado quando uma página da *Web* é removida. Como as páginas *Web* estão sendo atualizadas com freqüências diferentes, o coletor deve ser cuidadoso ao escolher qual página deve ser revisitada, já que essa decisão pode afetar o nível de atualização da coleção coletada. Por exemplo, se uma página raramente é alterada, o coletor pode escolher revisitá-la com menos freqüência para dar prioridade às que são alteradas freqüentemente.

- Como a carga imposta aos *sites* visitados deve ser minimizada? Quando um coletor coleta as páginas na *Web*, ele consome recursos pertencentes a outras organizações. Por exemplo, quando o coletor coleta uma página de um *site*, o *site* deve recuperar a página em seu sistema de arquivos, consumindo recursos de disco e CPU. Após a página ser recuperada pelo *site*, ela deve ser transferida pela rede, que é outro recurso compartilhado por várias organizações. O coletor deve tentar minimizar os impactos sobre esses recursos. Caso contrário, os administradores dos *sites* ou de uma rede particular podem reclamar ou até mesmo bloquear o acesso do coletor.

- Como o processo de coleta pode ser paralelizado? Devido ao tamanho da Web, os coletores normalmente são executados em múltiplas máquinas para coletarem páginas em paralelo. Esta paralelização é necessária para permitir que uma grande quantidade de páginas seja coletada em um tempo razoável. Fica claro que os coletores em paralelo devem ser coordenados para que diferentes coletores não visitem o mesmo site repetidas vezes.

- Como lidar com *sites* maliciosos (*index spammers*, *black holes*) e *sites* dinâmicos? O coletor deve ser capaz de identificar e descartar sites desse tipo, caso contrário ficaria "preso" e impossibilitado de coletar páginas úteis.

3.3. Armazenamento

O repositório de páginas, mostrado na Figura 1, é um sistema de armazenamento escalável capaz de armazenar grandes coleções de páginas *Web*. Ele precisa fornecer duas funcionalidades básicas: uma interface para que o coletor armazene as páginas, e uma eficiente API (*Application Programming Interface*) de acesso para uso dos módulos indexador e análise de coletas. A seguir serão apresentadas algumas técnicas para este sistema de armazenamento.

O repositório gerencia uma grande coleção de documentos, no caso, as páginas *Web*. Nesse sentido, ele é conceitualmente semelhante a outros sistemas que gerenciam dados, como sistemas de arquivo ou sistemas gerenciadores de banco de dados. Contudo, um repositório *Web* não precisa apresentar muitas das funcionalidades comuns a estes outros sistemas (por exemplo controle de transação, estrutura de diretório e *log*) e pode se concentrar nas seguintes questões:

- Escalabilidade: deve ser possível distribuir o repositório de maneira transparente através de vários *clusters* de computadores e discos, a fim de comportar coleções do tamanho da *Web*.

- Modo duplo de acesso: o repositório deve suportar dois tipos distintos de acesso com a mesma eficiência. O acesso aleatório, que é usado para recuperar uma página específica, dado um identificador da página. E o acesso em fluxo (*streaming*) que é usado para receber ou devolver uma coleção inteira de páginas. O acesso aleatório é usado pelo módulo de busca para retornar as páginas ao usuário. Já o acesso em fluxo é utilizado pelo coletor (escrita) e pelos módulos de indexação e de análise (leitura).

- Atualização e recuperação de espaço: assim como a *Web* muda rapidamente, o repositório deve acomodar uma alta taxa de mudança. Assim que novas versões de páginas são recebidas do coletor, o espaço ocupado pelas versões antigas precisa ser disponibilizado através de compactação e reorganização.

- Sincronização: os conflitos entre os processos de atualização de páginas e os processos de leitura devem ser tratados.

3.4. Indexação

Os módulos de indexação e de análise de coleta, mostrados na Figura 1, criam uma variedade de índices sobre as páginas coletadas. O módulo de indexação cria dois índices básicos: de texto (ou conteúdo) e de estrutura (ou de *links*). Usando esses dois índices e as páginas do repositório, o módulo de análise de coleta cria uma variedade de outros índices. Abaixo, será apresentada uma descrição de cada um desses índices, concentrando em sua estrutura e uso.

Índice de *links*: para construir um índice de links, a porção coletada da *Web* é modelada como um grafo com nós e arestas. Cada nó no grafo é uma página *Web*, e uma aresta do nó

A para o nó B representa um *link* na página A apontando para a página B. Um índice da estrutura de *links* deve ser uma representação eficiente e escalável deste grafo.

Normalmente, a informação estrutural mais utilizada pelos algoritmos de busca é a "informação de vizinhança". Por exemplo, dada uma página P, recuperar as páginas apontadas por P e as páginas que apontam para P.

Construir grafos com centenas ou até milhares de nós pode ser feito com estruturas de dados comuns. Porém, fazer o mesmo com estruturas de milhões de nós representa um desafio.

Índice de texto: ainda que as técnicas baseadas em análise de *links* sejam usadas para melhorar a qualidade dos resultados de busca, a pesquisa baseada em texto (por exemplo buscar páginas contendo certas palavras chave) continua a ser o método principal para identificar as páginas relevantes à uma determinada busca. Índices para suportar essa pesquisa baseada em texto podem ser implementados usando quaisquer dos métodos tradicionais para pesquisa de texto em coleções de documentos. Os índices invertidos têm sido a estrutura de índices tradicional dos mecanismos de busca *Web*.

Um *índice invertido* de uma coleção de páginas *Web* consiste de um conjunto de *listas invertidas*, uma para cada palavra (ou termo do índice). A lista invertida para um termo é uma lista ordenada, contendo os locais onde o termo aparece na coleção. No caso mais simples, o local será composto de um identificador de página e da posição do termo na página. Contudo, alguns algoritmos de busca normalmente fazem uso de informação adicional sobre a ocorrência dos termos na página *Web*. Por exemplo, termos ocorrendo em negrito (usando a *tag* <B>) ou em cabeçalhos (*tags* <H1> ou <H2>) podem ser classificados com peso diferente em algoritmos de classificação. Para acomodar isso, um campo extra (*payload*) é adicionado nas entradas de locais. Esse campo extra recebe qualquer informação adicional que precisa ser mantida sobre cada ocorrência do termo.

Índices utilitários: o número e o tipo de índices utilitários construídos pelo módulo de análise de coleta dependem das funcionalidades do módulo de busca e do tipo de informação utilizada pelo módulo de classificação. Por exemplo, um módulo de busca que permite que a busca seja restrita a um *site* ou domínio específico se beneficiaria de um índice do *site* que mapearia cada domínio a uma lista de páginas pertencentes aquele domínio. Da mesma forma, usando informação de vizinhança de um índice de *links*, um algoritmo iterativo pode computar um valor de classificação associado a cada página no repositório. Este índice poderia ser usado no momento de busca para ajudar na classificação dos resultados de busca.

3.5. Classificação e análise de *links*

Como mostrado na Figura 1, o módulo de busca recebe os termos de busca do usuário e busca as páginas relevantes. Há duas razões principais pelas quais as técnicas tradicionais de Recuperação de Informação (RI) podem não ser efetivas na classificação dos resultados de busca.

Primeiro, a *Web* é uma coleção muito grande, com muita variação na quantidade, qualidade e tipo de informação presente nas páginas. Desta forma, muitas das páginas que contém os termos de busca podem ser de baixa qualidade ou não relevantes.

Segundo, muitas páginas *Web* não são suficientemente auto descritivas, de modo que as técnicas de RI, que analisam apenas o conteúdo das páginas, podem não funcionar adequadamente. Um exemplo normalmente citado [12], é a questão de que as páginas de entrada de um mecanismo de busca normalmente não contém o texto "mecanismo de busca". Além disso, as páginas *Web* são freqüentemente manipuladas para adicionar termos espúrios de maneira que elas sejam classificadas pelo mecanismo de busca (*spamming*). Um exemplo clássico de *spamming* é o de páginas pornográficas que inserem termos fora de seu domínio para serem classificadas em buscas não relacionadas a pornografia. Desta forma, as técnicas que se baseiam suas decisões apenas no conteúdo das páginas são fáceis de serem manipuladas.

A estrutura de *links* da *Web* contém informações importantes que podem ajudar na filtragem e classificação de páginas. Em particular, um *link* de uma página A para a uma página B pode ser considerado como uma recomendação da página B pelo autor da página A. Alguns novos algoritmos têm sido propostos para explorar essa estrutura de *links*, não só para busca de termos, mas também para construir hierarquias de diretório (tipo Yahoo) automaticamente ou identificar comunidades na *Web*. Dois destes algoritmos serão discutidos no Capítulo 4. O desempenho qualitativo destes algoritmos é normalmente melhor que o dos algoritmos de RI, já que eles usam informação adicional além do conteúdo das páginas. Embora seja possível influenciar a estrutura de links da *Web* localmente, é bastante difícil fazer isto em nível global. Desta forma, os algoritmos de análise de links que trabalham a nível global tendem a ser robustos contra *spamming*.

3.6. Considerações finais

Neste capítulo foi apresentada uma visão geral dos mecanismos de busca, discutindo seus principais componentes e questões algorítmicas relacionadas. Entre os principais desafios técnicos levantados, estão a coleta de páginas *Web*, o armazenamento, a indexação e a classificação dos documentos.

Capítulo 4 – Algoritmos de classificação por análise de *links*

Como explicado na seção 3.5 do capítulo anterior, os algoritmos de classificação por análise de *links* vieram complementar as técnicas tradicionais de RI para buscas na *Web*. Esses algoritmos consideram a estrutura de *links* da *Web* como um grafo dirigido G. Eles partem de suposições simples como:

- Um *link* de uma página A para uma página B pode ser considerado como uma recomendação da página B pelo autor da página A;

- Se as páginas A e B estão conectadas por *links*, a probabilidade de que elas tratem de um mesmo assunto é maior do que se elas não estivessem conectadas.

Em [15], os algoritmos de classificação por análise de *links* estão separados em duas categorias: os algoritmos *independentes de busca* onde uma pontuação é atribuída a cada página independente da busca do usuário; e os algoritmos *dependentes de busca* onde as pontuações das páginas são calculadas dinamicamente a cada busca.

Neste capítulo, serão discutidos em detalhes os dois algoritmos mais conhecidos de análise de links: o PageRank e o HITS, respectivamente classificados como independente de busca e dependente de busca.

4.1. PageRank

Em [11], Page e Brin definem um mecanismo de classificação global, chamado PageRank, que tenta capturar a noção de "importância" de uma página. Por exemplo, a página de entrada do Yahoo pode ser intuitivamente considerada mais importante que uma página pessoal de um aluno da Unicamp. A diferença está no número de páginas que apontam para essas duas páginas, ou seja, mais páginas apontam para o Yahoo que para a página do aluno da Unicamp. O *rank* de uma página A poderia ser definido como o número de páginas na *Web* que apontam para A, e usado para classificar resultados de busca. O problema é que

este critério simples (a *classificação por citação*) não funciona muito bem Um problema prático é o *spamming*, que é a criação artificial de um grupo grande de páginas que apontam para uma certa página.

O PageRank corrige o conceito de classificação por citação levando em consideração também a *qualidade* das páginas que apontam para a página em questão. Desta forma, uma página recebe mais importância se o Yahoo aponta para ela que se uma página desconhecida aponta para ela. Note que a classificação por citação não faz distinção entre estes dois casos.

4.1.1. PageRank simplificado

A definição do PageRank é recursiva, já que a importância de uma página depende de e, ao mesmo tempo, influencia a importância de outras páginas.

Mais precisamente, suponha que as páginas Web são numeradas 1,2,...,m. Seja *N(i)* o número de links saindo de uma determinada página *i* e *B(i)* o conjunto de páginas que aponta para a página *i*. O valor do PageRank simplificado, representado por *r(i)*, para uma página *i* é dado por:

$$r(i) = \sum_{j \in B(i)} r(j) / N(j)$$

Note que a divisão por *N(j)* significa que o *rank* da página *j* é dividido e distribuído, de maneira igual, pelas páginas apontadas por ela. A Figura 2 mostra a propagação do *rank* entre um grupo de páginas.

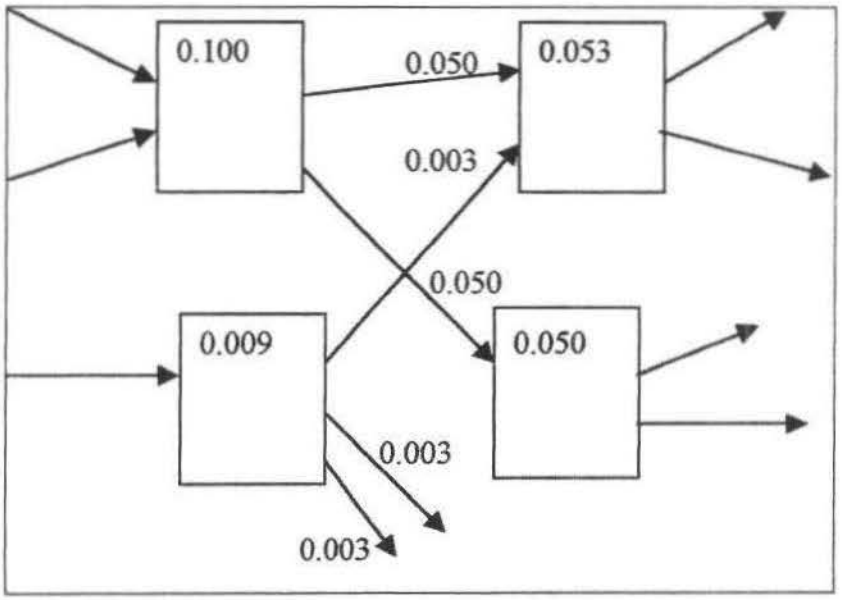

Figura 2 - Cálculo do PageRank simplificado

4.1.2. Modelo do Surfista com Comportamento Aleatório

A definição do PageRank simplificado admite uma interpretação baseada em caminho aleatório em grafos – o Modelo do Surfista com Comportamento Aleatório [11].

Considere uma pessoa que "surfa" pela Web, selecionando aleatoriamente um *link* em cada página visitada. Essa navegação aleatória é equivalente a um caminho aleatório em um grafo. O PageRank de uma página pode ser interpretado como a probabilidade limite de que o surfista esteja visitando essa página ao final de um período tendendo a infinito.

4.1.3. Cálculo do PageRank

Como a equação é recursiva, pode-se começar com um conjunto arbitrário de *ranks* e calcular de forma iterativa até que haja convergência. A Figura 3 mostra uma solução estável (após convergência) para um grupo de páginas.

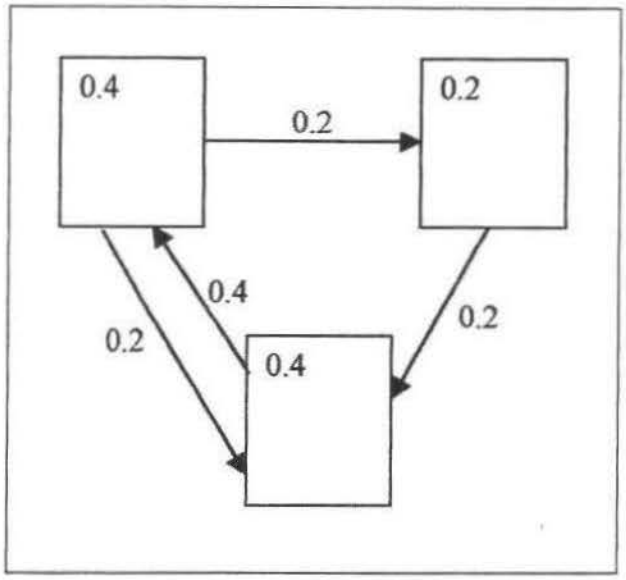

Figura 3 - Resultado estável de cálculo de PageRank simplificado

Em [11], é mostrada também uma abordagem onde o PageRank pode ser modelado e computado através de cálculo de matrizes.

4.1.4. PageRank usado na prática

Há várias dificuldades no cálculo do PageRank simplificado. Por exemplo, suponha que existem duas páginas A e B que apontam uma para outra e que não apontam para nenhuma outra página. Suponha que exista uma outra página C que aponta para a página A. O ciclo A,B irá acumular *rank* infinitamente e nunca distribuirá rank para as outras páginas. Esta formação é chamada de *sorvedouro* (*rank sink*) e é mostrada na Figura 4. Com o tempo, todo o PageRank estará concentrado nas páginas do sorvedouro, (o que obviamente não reflete a importância das mesmas).

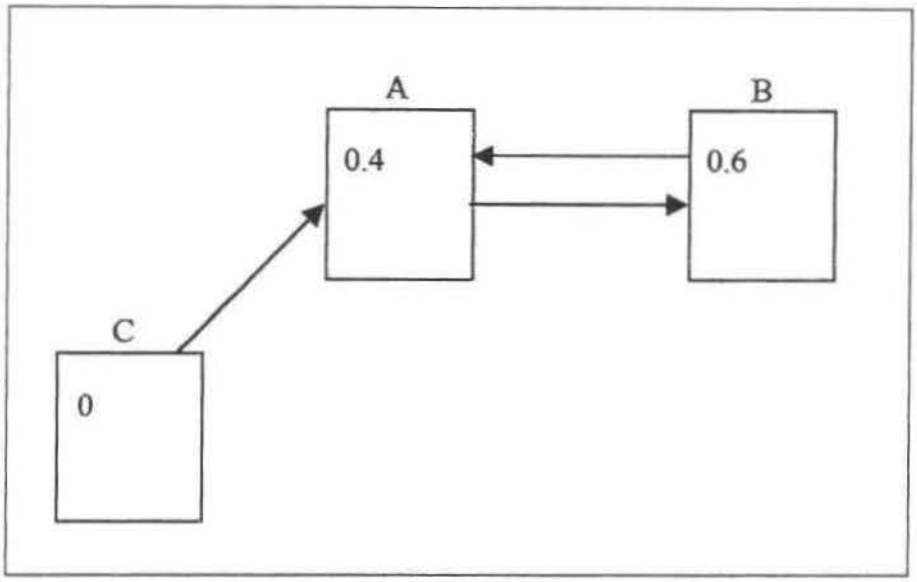

Figura 4 - Ciclo gerando um sorvedouro

Outro problema no cálculo do PageRank simplificado é o das páginas que não possuem nenhum link para outras páginas, chamado de *vazamento de rank* (*rank leak*). Qualquer *rank* for distribuído inicialmente a essas páginas ficará "preso" e não será distribuído para outras páginas.

Page e Brin [11] sugeriram duas formas para eliminar esses problemas. Primeiro, todos os vazamentos de *rank* são removidos do cômputo do PageRank. Segundo, para resolver o problema dos sorvedouros, foi introduzido um fator de correção d ($0 < d < 1$) na definição do PageRank. Desta forma, apenas uma fração d do *rank* de uma página é distribuída pelas páginas apontadas por ela. O resto do *rank* é distribuído igualmente por todas as páginas *Web*. Desta forma, o PageRank modificado fica:

$$r(i) = d * \sum_{j \in B(i)} r(j) / N(j) + (1 - d) / m,$$

onde m é o total de nós no grafo. Note que o PageRank simplificado é um caso especial onde $d = 1$.

Em termos do modelo do surfista com comportamento aleatório, a modificação modela a propabilidade de que o surfista fique "entediado" e pule para uma outra página aleatória, em lugar de simplesmente seguir um dos *links* da página corrente. Desta forma, ele não ficaria encalhado eternamente caso entrasse em um sorvedouro.

4.1.5. Usando o PageRank para buscas de palavras chave

Page e Brin [10], descrevem um protótipo de mecanismo de busca, desenvolvido na Universidade de Stanford, que fazia uso do PageRank. Esse protótipo se tornou mais tarde o mecanismo de busca Google. O Google combina o PageRank com técnicas tradicionais de Recuperação de Informação (RI) para classificar as páginas retornadas por uma busca.

4.2. HITS

O algoritmo HITS (*Hypertext Induced Topic Search*) foi inicialmente proposto por Kleinberg em [12]. Em contraste com o PageRank, que atribui um *rank* global e estático para cada página, o HITS cria o *rank* no contexto de cada busca. Além disso, ao invés de produzir somente um resultado de *rank*, ele produz dois, *o rank de autoridade* a_i e o *rank de diretório* d_i.

Informalmente, as páginas com mais autoridade são aquelas que têm mais chance de serem relevantes para uma determinada busca. Por exemplo, a página da Unicamp é uma autoridade para a busca da "Universidade Estadual de Campinas". As páginas diretório, por outro lado, são aquelas que não são autoridades, mas apontam para várias autoridades. Por exemplo, a página da Capes provavelmente é uma página diretório para a pesquisa "cursos de pós-graduação" já que aponta para *sites* de vários cursos de pós-graduação. A razão para o interesse em se encontrar as páginas diretório é devido ao fato de que o algoritmo HITS as usa para encontrar as páginas de autoridade.

Em resumo, as páginas diretório são aquelas que apontam para várias autoridades. E uma página é uma boa autoridade, quando ela é apontada por vários diretórios. Ou seja, existe uma relação de indicação mútua entre os diretórios e autoridades. Esta noção leva ao algoritmo HITS.

A idéia básica do algoritmo HITS é identificar um pequeno subgrafo da *Web* incluindo várias páginas possivelmente relevantes para a busca feita pelo usuário e aplicar a análise de *links* neste subgrafo para encontrar as autoridades e diretórios. A seleção de um pequeno subgrafo (tipicamente alguns milhares de páginas) não apenas foca a análise de *links* nas páginas mais relevantes, como também reduz a quantidade de processamento para a próxima fase. Como a seleção do subgrafo e sua análise é feita no tempo de busca do usuário, é importante que termine rápido.

4.2.1. Indentificando o subgrafo focalizado

O subgrafo focalizado é gerado formando um grupo raiz R, que é composto por um grupo de páginas contendo os termos de busca, e expandindo este grupo raiz com a inclusão das páginas que estão na vizinhança de R, como mostrado na Figura 5. Um índice de texto comum pode ser utilizado para construir o grupo raiz. A partir daí, o algoritmo para calcular o subgrafo focalizado é o seguinte:

1. $R \leftarrow$ conjunto de t páginas que contém os termos de busca.
2. $S \leftarrow R$
3. para cada página $p \in R$
    a) Inclua em S todas as páginas apontadas por p
    b) Inclua (até um máximo d) em S todas as páginas que apontam para p
4. O grafo produzido por S é o subgrafo focalizado

O algoritmo recebe como entrada os termos de busca e dois parâmetros t e d. O parâmetro t limita o tamanho do conjunto raiz, enquanto o d limita o número de páginas adicionadas ao

subgrafo focalizado. O parâmetro *d* também serve para limitar a influência de uma página extremamente popular (por exemplo Yahoo) caso ela esteja no grupo raiz. O grupo expandido *S* deve ser rico em autoridades, já que é provável que uma autoridade seja apontada por pelo menos uma página do grupo raiz. Da mesma forma, espera-se que um bom número de diretórios estejam em *S*.

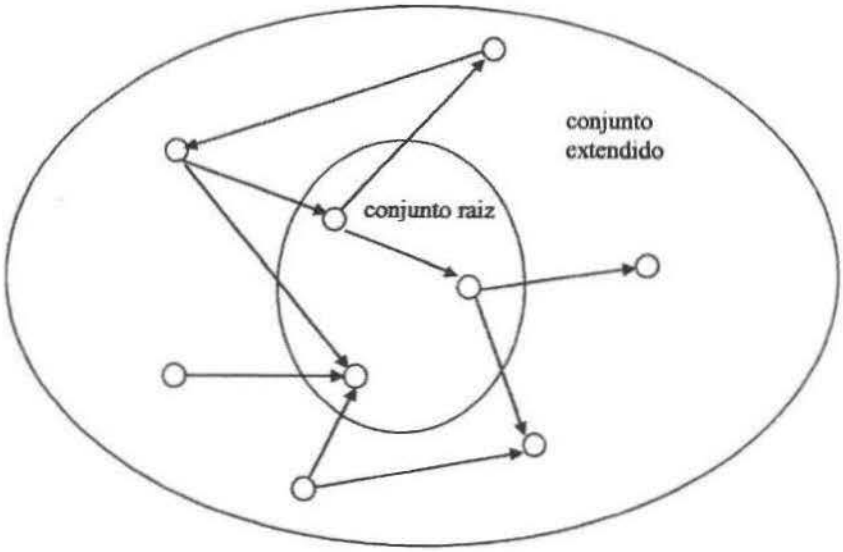

Figura 5 - Extendendo o conjunto raiz

4.2.2. Análise de *links*

A fase de análise de *links* do algoritmo HITS usa a propriedade de indicação mútua para identificar diretórios e autoridades a partir do conjunto extendido *S*. Suponha que as páginas no subgrafo focalizado são indicadas por 1,2,...,*n*. Suponha que *B(i)* representa o conjunto de páginas que aponta para uma página *i* e que *F(i)* representa o conjunto de páginas apontadas por uma página *i*. O algoritmo de análise de links produz uma pontuação de autoridade a_i e uma pontuação de diretório d_i para cada página do conjunto *S*. Para começar, são atribuídos valores arbitrários para as pontuações de diretório e autoridade. O algoritmo é iterativo e, em cada passo, ele realiza duas operações distintas chamadas de *I* e *O*. Na operação *I*, a pontuação de autoridade de cada página é atualizada pela soma da

pontuação de diretório de todas as páginas apontando para ela. Na operação *O*, a pontuação de diretório de cada página é atualizada levando em consideração a soma da pontuação de autoridade de todas as páginas apontadas por ela. Ou seja:

Passo *I*: $$a_i = \sum_{j \in B(i)} d_j$$

Passo *O*: $$d_i = \sum_{j \in F(i)} a_j$$

Os passos *I* e *O* capturam a intuição de que uma boa autoridade é apontada por vários bons diretórios e um bom diretório aponta para várias boas autoridades. Note que algumas vezes uma página pode ser, ao mesmo tempo, um diretório e uma autoridade. O algoritmo HITS calcula as duas pontuações para cada página e repete iterativamente os passos *I* e *O* até que, com normalização, as pontuações de diretório e autoridade atinjam convergência:

1. Inicialize a_i, d_i $(1 \leq i \leq n)$ com valores arbitrários
2. Repita até a convergência
    a) Aplique a operação *I*
    b) Aplique a operação *O*
    c) Normalize $\sum_i a_i^2 = 1$ e $\sum_i d_i^2 = 1$
3. Fim

Note que a normalização a cada iteração é importante para evitar que a_i e d_i cresçam indefinidamente.

Assim como o PageRank, o HITS também pode ser modelado e computado através de cálculo de matrizes. Kleinberg [12] mostra essa abordagem.

Uma resultado interessante da aplicação do algoritmo HITS, é que ele consegue separar as páginas em temas, ou seja grupos de páginas tais que há muitas referências dentro de cada grupo, mas poucas entre os grupos. Kleinberg [12] mostra um exemplo em que os resultados com mais alta citação de uma busca por "java" eram os *sites www.gamelan.com* e *java.sun.com*, ao lado do site da "*Amazon Books*". Os dois primeiros *sites* constituem um grupo (*sites* sobre a linguagem Java) fortemente conexo entre si, mas pouco conexo com o terceiro. O algoritmo HITS consegue isolar apenas um dos grupos (atribuindo pontuação zero para os grupos não relacionados) mesmo que eles tenham alta citação, como mostrado na Figura 6.

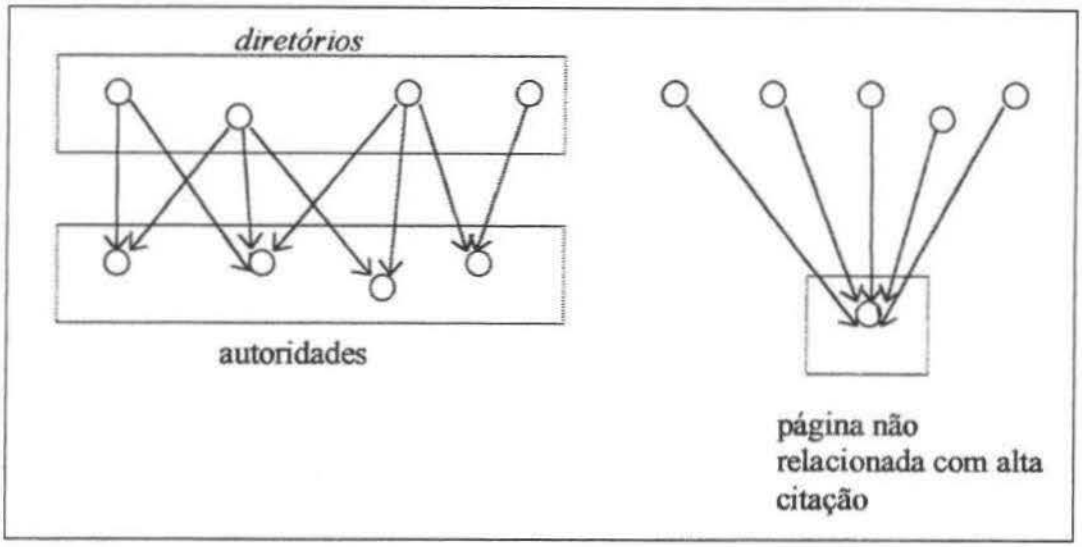

Figura 6 - Conjunto de páginas e *links*, exemplificando diretórios e autoridades

4.2.3. Uso do HITS em mecanismos de busca

Os conceitos do algoritmo HITS foram utilizados na implementação do mecanismo de busca Teoma (*www.teoma.com*), lançado em abril de 2000. O Teoma utiliza o algoritmo HITS para encontrar "comunidades" que compartilham o interesse pelo mesmo assunto e suas respectivas autoridades.

4.3. Avaliação dos algoritmos

Amento e Terveen [16] apresentam uma avaliação de vários algoritmos de classificação por análise de *links*, incluindo o PageRank e o HITS. Os resultados indicaram que ambos algoritmos apresentam desempenho (relativo à precisão das respostas) semelhantes, significativamente melhores que o apresentado por algoritmos que se baseiam somente em busca textual.

Um aspecto positivo do HITS, com sua classificação dinâmica no contexto de cada busca, é a separação de comunidades relacionadas pelo mesmo assunto, tornando-o mais robusto contra *spamming*, comparado a algoritmos estáticos como o PageRank. O exemplo a seguir, apresentado em [17], ilustra essa característica. Foi feita uma busca pela palavra "God", que apresentou como primeiro resultado, no Google, um site de músicas MP3, como mostrado na Figura 7.

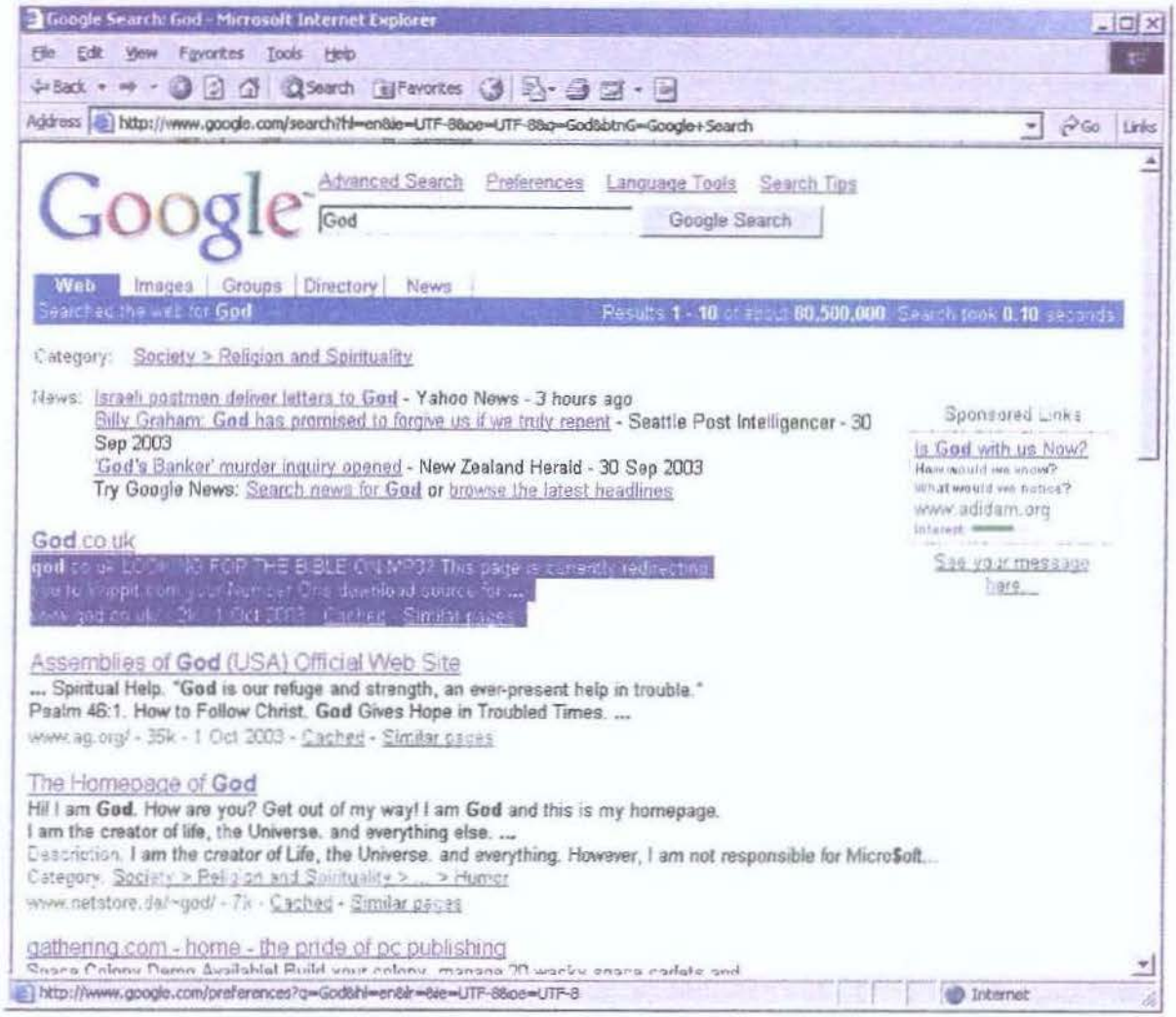

Figura 7 - Busca por "God" no Google

Já o Teoma, implementando o algoritmo HITS foi capaz de separar o site de MP3 de outras comunidades de *sites* religiosos. O resultado pode ser visto na Figura 8.

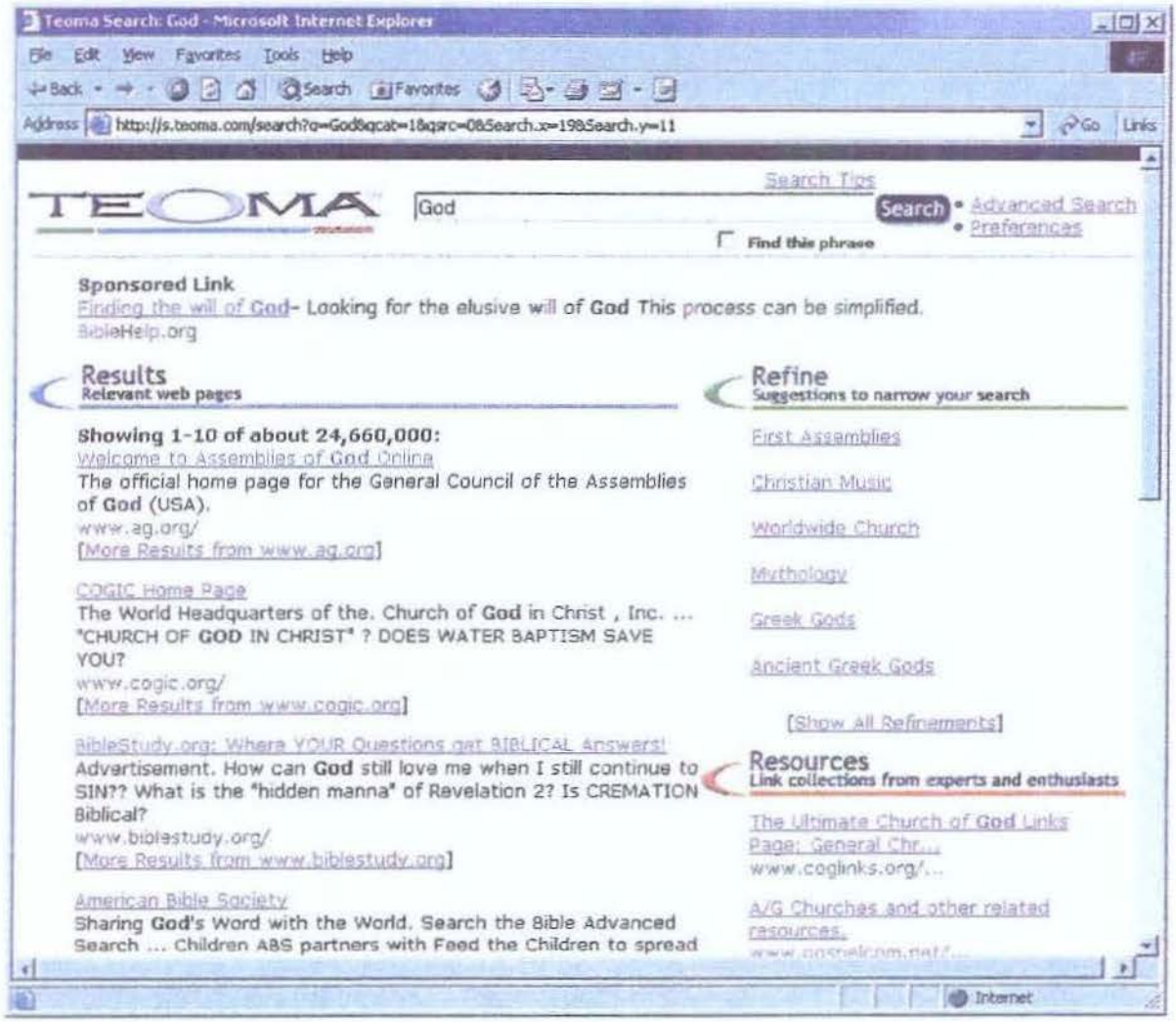

Figura 8 - Busca por "God" no Teoma

Por outro lado, os algoritmos dinâmicos como o HITS possuem o problema da escalabilidade. O HITS, por exemplo, tem que encontrar um meio-termo entre um conjunto raiz pequeno o suficiente para permitir a procura dos diretórios e autoridades em tempo razoável para o usuário, e grande o suficiente para aumentar a chance de encontrar os diretórios e autoridades mais significativos.

4.4. Considerações Finais

Neste capítulo foram estudados dois importantes algoritmos de classificação por análise de *links*. Esses algoritmos podem ser usados em conjunto com as técnicas tradicionais de RI,

para melhorar significativamente a precisão das respostas oferecidas pelos mecanismos de busca. Além disso, eles podem ser usados para resolver outros problemas como escolher a ordem de coleta das páginas *Web* e categorizar *sites* por assuntos.

Capítulo 5 – Exemplo de arquitetura de um mecanismo de busca comercial

Neste capítulo será apresentada a arquitetura do mecanismo de busca comercial Google, baseada em [10], um dos primeiros trabalhos acadêmicos a descrever detalhes de implementação de tais mecanismos. Através da arquitetura apresentada a seguir, poderemos verificar as soluções práticas para os problemas inerentes à construção de um mecanismo de busca, apresentados no Capítulo 3, além de acompanhar o uso do algoritmo de classificação PageRank, descrito no Capítulo 4.

O objetivo dos mecanismos de busca é fornecer, de maneira eficiente, resultados de busca com qualidade. Muitos dos mecanismos de busca comercial atingiram grande progresso em termos de eficiência. O Google, por sua vez, deu enfoque a melhoria da qualidade dos resultados de busca.

5.1. Visão geral da arquitetura do Google

A Figura 9 apresenta uma visão geral do funcionamento do Google.

Dois conceitos fundamentais utilizados no texto adiante são o identificador de documento (*docId*), que é um número único associado a cada URL absoluto e o identificador de palavra (*wordId*), outro número unicamente associado a cada palavra já encontrada.

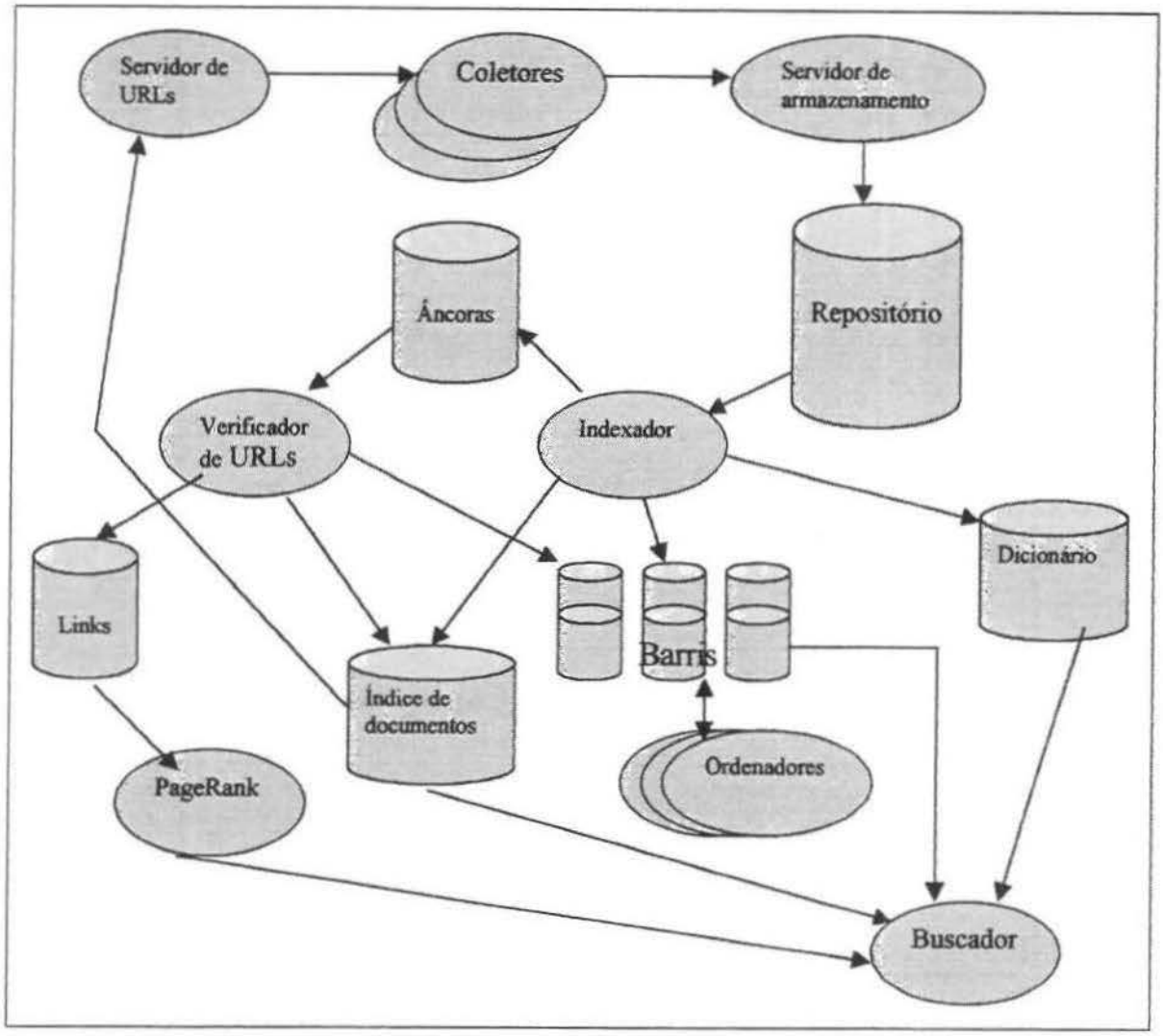

Figura 9 - Arquitetura do Google

Os principais componentes do Google são:

- Estruturas de dados
    1. Repositório: armazena as páginas coletadas.
    2. Barris: armazena os índices de texto (direto e invertido).
    3. Arquivo de âncoras: guarda informações sobre as URLs encontradas nos documentos.
    4. Base de dados de links: armazena os links entre os documentos.
    5. Índice de documentos: guarda informações sobre cada documento coletado.
    6. Dicionário: guarda um conjunto de palavras e seu respectivos *wordIds*

- Processos
    1. Coletor: coleta os documentos da *Web*.
    2. Servidor de URLs: distribui URLs para os coletores.
    3. Servidor de armazenamento: gerencia o repositório de documentos.
    4. Indexador: extrai palavras dos documentos e constrói o índice direto.
    5. Ordenador: constrói o índice invertido a partir do índice direto.
    6. Verificador de URLs: converte URLs relativas em absolutas, atribui *docIds* a elas, e gera a base de dados de *links*.
    7. Gerador de PageRank: associa um PageRank a cada documento.
    8. Buscador: recebe e processa consultas de usuários do Google.

A coleta de páginas é feita através de vários coletores distribuídos. Há um servidor de URLs que envia listas de URLs a serem buscadas pelos coletores. As páginas são coletadas e enviadas a um servidor de armazenamento. Esse servidor comprime e guarda as páginas em um repositório. As funções de indexação são realizadas pelo indexador e pelo ordenador. O indexador realiza várias tarefas. Ele acessa o repositório, descomprime os documentos e os lê. Cada documento é convertido em um conjunto de ocorrências de palavras. Cada ocorrência contém a palavra identificadora, a posição no documento, o tamanho dos caracteres e a caixa (maiúscula/minúscula). O indexador distribui essas ocorrências em um conjunto de barris, criando um índice direto parcialmente ordenado. Outra importante função do indexador é ler os *links* presentes em cada página e guardar as informações relevantes (texto visível do *link*, URL da página onde o *link* está presente e URL da página apontada) no arquivo de âncoras. Desta forma, esse arquivo é capaz de identificar a origem e destino de um *link* e seu texto.

O verificador de URLs lê o arquivo de âncoras, converte URLs relativas em URLs absolutas e finalmente em *docIds*. Ele adiciona ao índice direto o texto dos *links* (do arquivo de âncoras) associado com o *docId* apontado pelo *link*. Por fim, o verificador de URLs gera uma base de dados de *links* (representados por pares de *docIds*), que será utilizada para o cálculo do PageRank (algoritmo descrito na seção 4.1) para todos os documentos.

O ordenador acessa os barris, que são ordenados por *docId* e reordena-os segundo palavras (*wordId*) para gerar um índice invertido. O ordenador também cria uma lista de *wordIds* e sua localização no índice invertido. Um programa chamado DumpLexicon faz uso dessa lista, juntamente com o dicionário produzido pelo indexador e cria um novo dicionário, para ser utilizado pelo buscador. O buscador é executado a partir de um servidor *Web* e usa o dicionário criado pelo DumpLexicon junto com a lista invertida e o PageRank para responder às buscas de usuário.

5.2. Principais estruturas de dados

Embora as taxas de processamento (CPU) e leitura/escrita em bloco tenham crescido bastante durante os últimos anos, uma procura em disco ainda requer cerca de 10 ms. Portanto, as estruturas de dados foram escolhidas de forma a evitar, sempre que possível, buscas em disco. Isto teve impacto na escolha das estruturas de dados.

Os arquivos maiores são gerenciados por um pacote especial que permite arquivos virtuais (*BigFiles*) que se extendem por múltiplos sistemas de arquivos e são endereçados usando 64 bits. A alocação dos múltiplos sistemas de arquivos é tratada automaticamente. O pacote de implementação de arquivos grandes também trata a alocação e desalocação de descritores de arquivos e opções de compressão rudimentares.

5.2.1. Repositório

O repositório armazena o conteúdo HTML completo de cada página coletada. Cada página é comprimida utilizando zlib [13], que apresenta uma alta velocidade associada a uma taxa de compressão razoável. No repositório, os documentos são guardados um após outro e prefixados pelo *docId*, tamanho e URL. O repositório não requer nenhuma outra estrutura de dados para ser utilizado. Isto permite que todas as outras estruturas de dados possam ser reconstruídas a partir apenas do repositório.

5.2.2. Índice de documentos

O índice de documentos guarda informações sobre cada documento. Ele possui entradas de tamanho fixo, ordenadas pelo *docId*. As informações guardadas em cada entrada incluem o estado do documento, um ponteiro para o repositório, o *checksum* do documento e várias estatísticas. A decisão de *design* foi guiada pelo desejo de se criar uma estrutura razoavelmente compacta, de forma que a leitura de uma entrada seja feita em apenas uma busca de disco.

Adicionalmente, há um arquivo que é usado para converter URLs em *docIds*. É formado por uma lista de *checksums* das URLs com os correspondentes *docIds*, ordenada pelo *checksum*. Para encontrar o *docId* de uma determinada URL, o *checksum* da URL é computado e uma busca binária é feita no arquivo. As URLs podem ser convertidas em *docIds* através de processamento em lote. Esta técnica é utilizada pelo verificador de URLs para evitar uma busca em disco para cada *link*.

5.2.3. Dicionário

O dicionário contém cerca de 14 milhões de palavras e é implementado em duas partes: uma lista de palavras e uma tabela *hash* de ponteiros para o índice invertido. Uma mudança importante em relação aos sistemas anteriores é que o dicionário pode caber na memória principal (a implementação atual cabe em 256 MB de memória).

5.2.4. Listas de ocorrência

As listas de ocorrência correspondem às ocorrências de uma determinada palavra em um documento, incluindo a posição dentro do documento, o tamanho dos caracteres e caixa (letras maiúsculas ou minúsculas). As listas de ocorrência consomem a maioria do espaço usado nos índices diretos e invertidos. Desta forma, é importante representá-las da maneira mais eficiente possível. Foi escolhida uma codificação compacta, com 2 *bytes* para cada

ocorrência, onde os *bits* são alocados manualmente entre os campos, como mostrado na Figura 10.

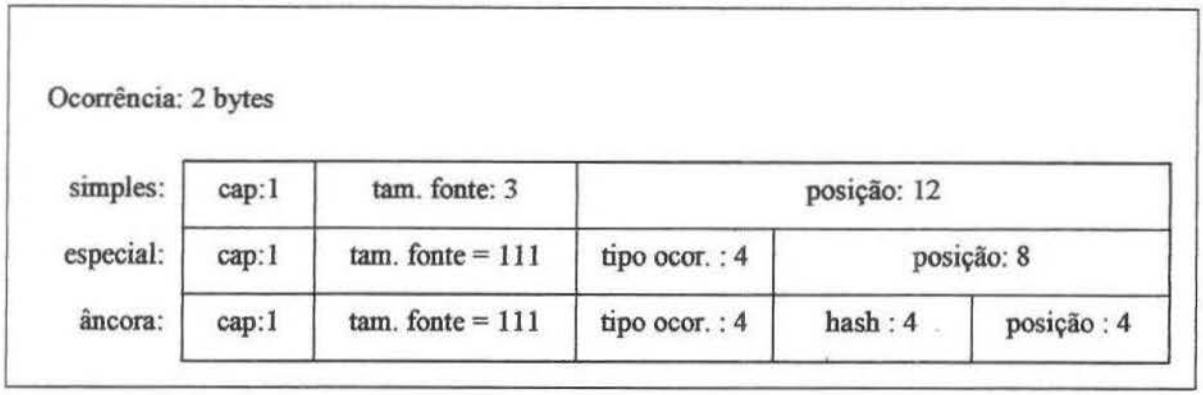

Figura 10 – Estrutura das ocorrências

São considerados dois tipos de ocorrências: as ocorrências simples e as especiais. As segundas incluem as ocorrências da palavra em URLs, título do documento ou texto de *link*. As simples, incluem todas as outras ocorrências. Uma ocorrência simples consiste em 1 *bit* de capitalização, 3 *bits* para tamanho da fonte (representada de forma relativa em relação ao resto do documento) e 12 *bits* para a posição da palavra no documento (todas as posições maiores que 4095 são representadas por 4096). As ocorrências especiais possuem 1 *bit* para capitalização, o tamanho da fonte representado por 111 (*flag* que sinaliza ocorrência especial), 4 *bits* para sinalizar o tipo da ocorrência especial e 8 *bits* para a posição no documento. Para o caso das ocorrências em *links* (âncoras) os 8 *bits* de posição são divididos em 4 para a posição da palavra no texto do *link* e 4 para o código *hash* do *docId* apontado pelo *link*.

O tamanho da lista de ocorrência é guardada antes das ocorrências. Para economizar espaço, o tamanho da lista é combinado com o *wordId* no índice direto e com o *docId* no índice invertido.

5.2.5. Índice direto

O índice direto é criado parcialmente ordenado. Ele é guardado em um número de barris (são usados 64). Cada barril guarda um intervalo de *wordIds*. Se um documento possui palavras mapeadas em determinado barril, o *docId* é guardado no barril, seguido da lista de *wordIds* e das listas de ocorrência, relativas às palavras mapeadas. Este esquema gasta um pouco mais de espaço de armazenamento, devido a duplicação de *docIds* em diferentes barris, mas consegue diminuir o tempo de processamento do ordenador na fase final de processamento.

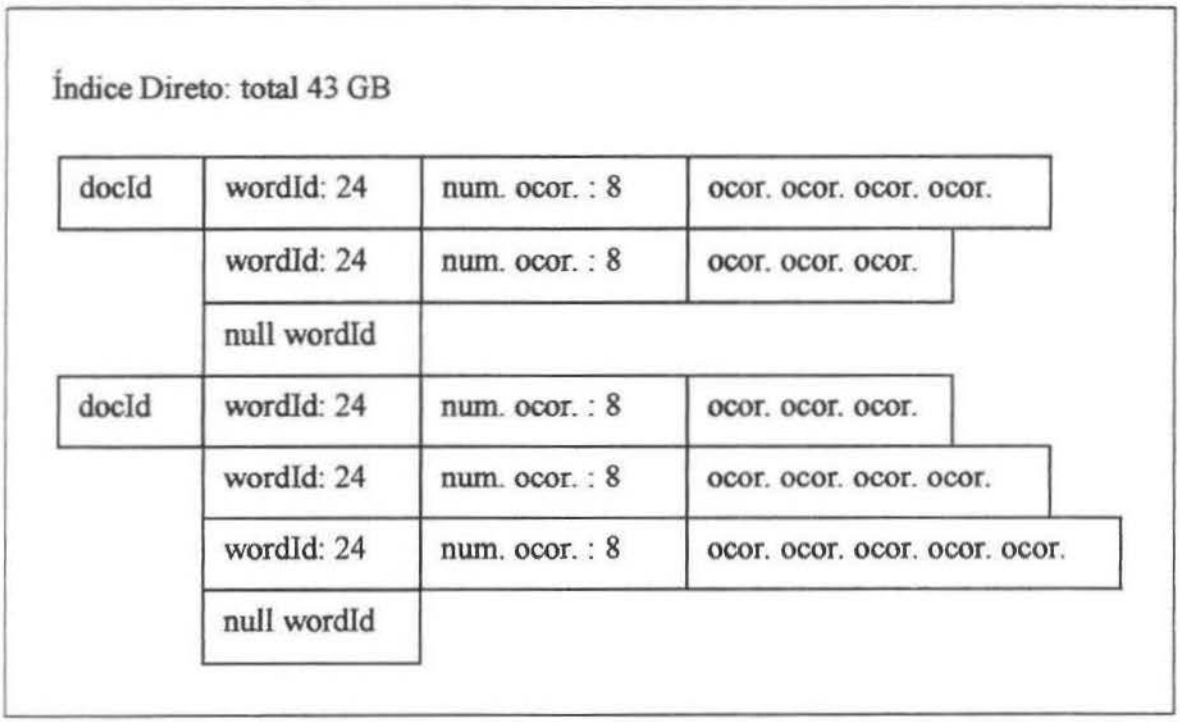

Figura 11 - Estrutura do índice direto

5.2.6. Índice invertido

O índice invertido consiste dos mesmos barris que o índice direto, exceto pelo fato de que eles já foram processados pelo ordenador. Para cada *wordId* válida, o dicionário contém um apontador para o barril em que a *wordId* está armazenada. Esse apontador aponta para uma

lista de *docIds* e as listas de ocorrência correspondentes. Desta forma, a lista de documentos representa as ocorrências de uma determinada palavra em todos os documentos.

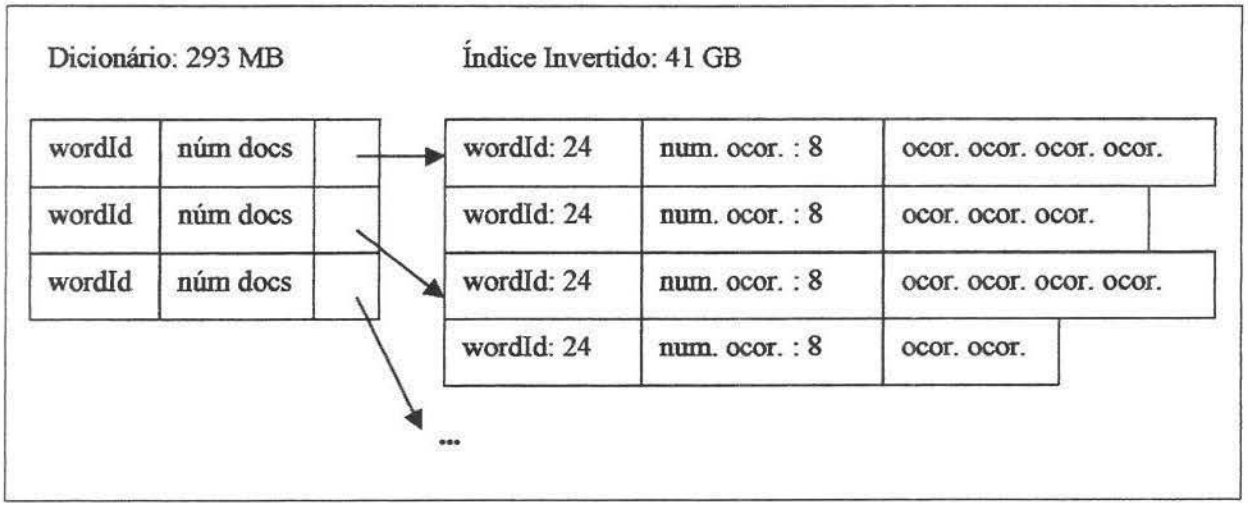

Figura 12 - Estrutura do dicionário e índice invertido

Uma questão importante é como ordenar os *docIds* na lista de documentos de uma palavra. Uma solução simples é armazená-los ordenados por *docId*. Isso facilita a junção de listas de documentos para buscas de múltiplas palavras. Outra opção é em ordem decrescente da relevância das ocorrências da palavra. Isto torna as respostas às buscas de uma palavra só triviais, porém torna a junção das listas mais difícil. Além disso, exigiria que o índice fosse refeito a cada mudança na função de classificação. Foi escolhido um meio-termo entre as duas opções, guardando dois conjuntos de barris: um conjunto para as ocorrências especiais e outro conjunto para as ocorrências simples. Desta forma, pode-se pesquisar inicialmente no primeiro conjunto de barris e passar para o segundo conjunto somente se não forem encontrados resultados suficientes.

5.3. Coletando páginas *Web*

Colocar em operação um coletor de páginas *Web* é uma tarefa difícil. Há que considerar uma série de questões de desempenho e confiabilidade e, principalmente, questões sociais.

O coletor é a aplicação mais frágil do sistema, já que ele interage com milhares de servidores *Web* e servidores de nomes que estão fora do controle do sistema.

Para poder coletar milhões de páginas *Web*, o Google usa um sistema de coleta distribuído. Um servidor de URLs fornece uma lista de URLs para um certo número de coletores (tipicamente 3). Cada coletor mantém cerca de 300 conexões abertas, para poder ler as páginas em uma velocidade razoável. O sistema chega a atingir picos de leitura (usando 4 coletores) onde coleta 100 páginas *Web* por segundo, que equivale na média, a 600 Kb por segundo. Um ponto crítico para o desempenho é a busca de nomes em DNS (*Domain Name Server*). Para evitar uma busca no servidor a cada página coletada, os coletores possuem um *cache* de DNS. Cada uma das centenas de conexões podem estar em diferentes estados: procurando no DNS, se conectando ao *host*, enviando requisições e recebendo respostas. Os coletores usam E/S assíncrona para gerenciar os eventos, e filas para mover as coletas de página de um estado para outro.

Na prática, executar um coletor que se conecta a meio milhão de servidores gera uma grande quantidade de reclamações. Muitos administradores confundem o coletor com um usuário comum, outros gostariam que seus *sites* não fossem indexados, mas tentam indicar isso incluindo textos de *copyright*, em lugar de usar o protocolo para exclusão de coletores [14]. Devido a imensa variação de conteúdo das páginas *Web* e dos servidores, é quase impossível testar um coletor sem executá-lo contra um "pedaço" significativo da Internet. Existem inúmeros problemas obscuros que podem ocorrer, interrompendo a execução do coletor ou causando comportamento não esperado. Desta forma, o operador do serviço de coleta deve estar preparado para enfrentar esses problemas durante a execução dos coletores.

5.4. Indexando páginas *Web*

Cada página que é coletada precisa ser indexada Esta tarefa envolve os seguintes passos: análise, indexação e ordenação.

5.4.1. Análise

A análise da página é feita com o objetivo de extrair da mesma o texto a indexar e os *links* de novas páginas a recuperar. No mínimo, o analisador deve ser capaz de lidar com as páginas HTML. O Google também indexa arquivos em outros formatos como PDF, *PostScript*, *MS Word*, etc. Qualquer analisador destinado a analisar páginas arbitrárias da *Web* deve estar preparado para uma grande quantidade de erros possíveis, como caracteres não ASCII ou erros nas *tags* HTML. O analisador HTML do Google foi implementado através da ferramenta *flex*.

5.4.2. Indexação

Depois que cada documento é analisado, ele é codificado em diferentes barris. Cada palavra do texto extraído é convertida em *wordId* usando uma tabela *hash* em memória primária (o dicionário). As adições ao dicionário são armazenadas em um arquivo. Assim que as palavras são convertidas em *wordId*, as suas ocorrências no documento corrente são traduzidas em listas de ocorrência e são armazenadas nos índices diretos.

A maior dificuldade para a indexação em paralelo é atualizar o dicionário, pois ele é compartilhado. Em lugar disso, a abordagem utilizada foi fornecer a cada processo indexador um dicionário fixo com 14 milhões de palavras. Os termos encontrados nesse dicionário são indexados imediatamente, os demais são repassados a um único processo separado que usa um dicionário mutável para atribuir a eles novos *wordIds*. Desta forma, múltiplos indexadores podem ser executados em paralelo.

5.4.3. Ordenação

Para gerar um índice invertido, o ordenador deve processar cada barril contendo um índice direto e ordená-lo por *wordId* para produzir um barril contendo o índice invertido correspondente. Note que as palavras (*wordIds*) são colocadas em dois barris: um para as ocorrências especiais e outro para as ocorrências simples. Esse processo é realizado com um barril por vez, necessitando, desta forma, de pouco espaço de armazenamento

temporário. Além disso, a ordenação é feita em paralelo, com a execução de múltiplos ordenadores.

5.5. Realizando buscas

As buscas solicitadas pelos usuários consistem de um conjunto de palavras a buscar (possivelmente consecutivas). O resultado da busca é uma lista de páginas $P_1,P_2,...P_i$, ordenada por uma pontuação $S(P_i)$.

O processo de busca do Google está descrito a seguir:

1. Leia a expressão de busca e separe as palavras.
2. Converta as palavras em *wordIds*.
3. Para cada palavra da expressão de busca, localize a lista de documentos do índice invertido que contém ocorrências especiais dessa palavra. Processe essa lista de documentos coletando todos os documentos que contenham ocorrências de todas as palavras da expressão de busca.
4. Repita o passo anterior utilizando as lista de documentos do índice invertido de ocorrências simples correspondentes as palavras procuradas.
5. Calcule a pontuação $S(P_i)$ para cada documento encontrado nos dois passos anteriores.
6. Ordene (pela pontuação calculada em 5) todos os documentos encontrados nos passos 3 e 4 e retorne um número máximo *k* para o usuário.

5.5.1. Classificação

O Google mantém mais informação sobre os documentos *Web* que os mecanismos de busca típicos. Como mostrado na seção 5.2, cada lista de ocorrência inclui a posição, o tamanho da fonte e informação de caixa. Todas essas informações afetam a pontuação $S(P_i)$, que é usada para ordenar as páginas antes de apresentá-las ao usuário. Além disso, a pontuação

$S(P_i)$ leva em consideração, o tipo da ocorrência (texto comum, título do documento, URL ou texto de *link*) e o PageRank do documento.

Combinar toda essa informação em uma classificação relevante para o usuário é difícil. Foi tomado cuidado especial para que nenhum fator tenha influência demasiada na função de classificação.

Primeiro, será analisado o caso mais simples – uma busca de apenas uma palavra. Seja $C_1,C_2,...,C_i$ a lista de ocorrências daquela palavra nesse documento. Para cada ocorrência, é atribuído um peso $W(C_i)$ relativo ao seu tipo. Os pesos são somados para formar o peso total $W(P_i)$ daquela palavra no documento. Por fim, essa pontuação é combinada com a pontuação *PageRank* $P(P_i)$ para dar a pontuação final $S(P_i)$ do documento.

No caso de uma busca de múltiplas palavras, a situação é mais complicada. Agora, várias listas de ocorrência devem ser verificadas para encontrar documentos em que todas as palavras ocorrem simultaneamente. Além disso, as ocorrências $C_1,C_2,...,C_i$ dessas palavras num documento P_i devem ser classificadas de forma que aquelas onde as palavras da lista estejam próximas tenham pontuação maior que as demais ocorrências (na busca por frase, aliás, é preciso que as palavras ocorram em posições consecutivas e na ordem especificada). Desta forma, são atribuídos pesos $W_d(C_i)$ diferentes para níveis diferentes de distância (classificados desde ocorrências em sequência na frase até ocorrências distantes). Assim, para cada ocorrência, é atribuído um peso $W_t(C_i)$ para seu tipo e $W_d(C_i)$ para sua proximidade (em relação a outras ocorrências). Esses pesos são somados para gerar um peso $W(P_i)$, que, assim como no caso das ocorrências simples, é combinado com a pontuação *PageRank* $P(P_i)$ para dar a pontuação final $S(P_i)$ do documento.

5.5.2. Sistema de realimentação

A função que calcula $S(P_i)$ possui vários parâmetros, em particular os pesos relativos atribuídos às ocorrências (tipo, proximidade, etc) e o PageRank. Calibrar esses parâmetros é uma tarefa complicada. Para isto, foi criado um mecanismo de realimentação (*feedback*)

que permite customizar esses parâmetros. Um usuário especial pode avaliar os resultados retornados. O sistema irá salvar o *feedback* e usá-lo como base de comparação quando são realizadas mudanças na função de classificação. Desta forma, pode-se ter uma idéia de como as mudanças na função de classificação afetam os resultados de busca.

5.6. Considerações finais

Neste capítulo foi apresentada a descrição do mecanismo de busca comercial Google. A intenção foi mostrar como alguns dos desafios técnicos apresentados no Capítulo 3 foram resolvidos nessa implementação específica, além de ilustrar o uso do algoritmo PageRank descrito no Capítulo 4.

Uma característica interessante nas soluções apresentadas é a preocupação com questões não funcionais, como desempenho, otimização de espaço em disco e memória, e escalabilidade. Pode-se entender essas preocupações tendo-se em vista o volume de informações a ser processado. É interessante ressaltar que, na época da divulgação da descrição feita em [10] (1998), o Google possuía uma base de dados de 24 milhões de páginas. Atualmente (Fevereiro de 2004), a base de dados conta com mais de 4 bilhões de páginas. Desta forma, a preocupação com os aspectos não funcionais apresentados anteriormente deve continuar presente e até ter aumentado.

Capítulo 6 – Estudo de caso: grau de cobertura da *Web* por um mecanismo de busca comercial

Neste capítulo vamos tentar avaliar, de forma empírica, o grau de cobertura da Web por um mecanismo de busca comercial. Foi escolhido o Google, por ser atualmente o mecanismo com maior base de dados (cerca de 4,2 bilhões de páginas em Fevereiro de 2004), além de disponibilizar APIs que permitem a automatização de buscas, facilitando o experimento.

6.1. Motivação

Embora muitos dos mecanismos de busca divulguem o tamanho de sua base de dados, não é possível ter uma idéia clara se esse tamanho é suficiente para oferecer uma boa cobertura da *Web*. O tamanho da *Web* é uma grandeza que não tem muito sentido nem muita utilidade. Não tem muito sentido pois há um grande número de páginas que podem ser acessadas apenas por meios não convencionais como *forms*, *scripts*, endereços privados, através de senhas, etc. Incluir tais páginas no cálculo de cobertura seria no mínimo questionável. Uma pergunta mais útil nesse contexto é: "Dada uma página qualquer da *Web* procurada pelo usuário, qual é a probabilidade de que ela esteja na base de dados de um determinado mecanismo de busca?"

Formalmente, seja D o conjunto de todas as URLs possíveis, atribuímos um peso ou probabilidade p(i) para cada URL i, que significa a probabilidade da URL i ser escolhida numa consulta aleatória. Neste modelo, as páginas secretas e inacessíveis terão pesos muito menores que as páginas acessíveis por caminhos naturais. Podemos então definir uma "URL aleatória" como sendo uma URL i escolhida com probabilidade P(i). Queremos então saber qual é a probabilidade de que uma "URL aleatória" tenha sido indexada pelo serviço de busca (Google). Se X é o conjunto de URLs indexadas, então queremos saber:

$$p = \sum_{i \in X} p(i)$$

6.2. Metodologia

A idéia básica do experimento é reunir uma amostra de URLs aleatórias e verificar qual fração delas foi coletada e indexada pelo mecanismo de busca em questão. Para isso, é necessário resolver os seguintes problemas:

- Escolha do modelo probabilístico: decidir o modelo de probabilidade "p" a ser usado para as URLs.

- Amostragem: conseguir uma lista S de URLs, onde cada URL i é escolhida com probabilidade proporcional p(i), a ser verificada contra a base de dados do mecanismo de busca.

- Verificação: para cada página da lista S, verificar se ela se encontra na base de dados do mecanismo de busca.

6.2.1. Escolha do modelo probabilístico

Pelas mesmas razões discutidas acima, não é nem possível nem interessante usar um modelo em que todas as URLs tenham a mesma probabilidade. Outra abordagem (que é a única factível) é usar p(i) proporcional à freqüência com que a URL é acessada normalmente (sem uso de um coletor/indexador). Esse valor poderia ser obtido por uso de *logs* da *Web* global ou estimado por modelos como o do "surfista aleatório" (PageRank).

A solução encontrada foi partir de um conjunto arbitrário de páginas e navegar pelos seus *links*, de forma semelhante ao "Modelo do Surfista com Comportamento Aleatório", utilizado pelo PageRank. Após um certo número de iterações, podemos selecionar a página corrente e utilizá-la na fase de verificação. Desta forma, cada página i da Web será testada com probabilidade P(i), igual a seu PageRank.

6.2.2. Amostragem

O ponto de partida para o conjunto inicial de páginas foi a base de dados do serviço de diretório dmoz [24] que contém cerca de 4 milhões de páginas. O seu catálogo ASCII de URLs foi processado por um *script* que escolheu ao acaso 10000 URLs. O número de URLs escolhidas representa um meio-termo entre conseguir uma amostra representativa da Web e ser capaz de processar as páginas em tempo razoável.

Para cada URL do conjunto é aplicado o algoritmo do surfista, e a página corrente é selecionada após 10 iterações.

Na tabela a seguir, é mostrado um conjunto de 25 páginas, com a página inicial escolhida do dmoz e a página alcançada após 10 iterações.

Página inicial do dmoz	Página após 10 iterações
http://www.screenit.com/	http://www.amazon.com/exec/obidos/tg/detail/-/B00005JKZY?%5Fencoding=UTF8
http://www.lofidelityallstars.com/	http://purl.org/rss/1.0/
http://www.georgespubngrub.com/eventcal.htm	http://www.netzero.net/support/getting-started.html
http://www.suntimes.com/ebert/ebert_reviews/1990/11/577651.html	http://www.starnewspapers.com/index/s-biz-za.html
http://www.all-reviews.com/videos-5/house-of-dead.htm	http://www.netflix.com/Default?mqso=60141305
http://forum.jurassicisland.ch/	http://www.at-nite.com/USA/s_OR/co_Wallowa/Wallowa.htm
http://www.digitalpostproduction.com/Htm/Articles/IndieFilms/One/One_1.htm	http://digitalmedianet.dealtime.com/xCP-NEC_MultiSync_LCD1760V_17_in~PD-20658632~FD-9006~kworg-lcd_monitor~linkin_id-3010840~DMT-

	3~VK-
http://www.rockymusic.org/cds/meatloaf/backfromhell.html	http://www.moock.org/asdg/
http://www.geocities.com/stegallery	http://kwiki.ffii.org/SwpatcninoEn
http://www.usaloans4u.com/	http://www.omniture.com/privacy/optout.html
http://spazioinwind.libero.it/bjj/	http://www.three-stroke.co.uk/home.html
http://p.dimayuga.free.fr/shingitai/fr/	http://www.redhat.com/apps/response/sub_thankyou.html
http://www.angelfire.com/nj/persephone/index.html	http://dir.webring.com/rw?forum=y;d=Government___Politics
http://www.ong-biz.com/elizwalkeracct	http://shop.areatravelpackets.com/listing.asp?mscssid=B9MJ65DKD8089MDQDRDLUELP8EJ1BK58
http://www.textlibrary.com/TITLE/gambler/index.htm	http://www.ebookmall.com/knowledge-collection/instructions.htm
http://www.ihomedecor.com/	http://demo.sitebrand.com/section.jsp?s=3&sblid=header4
http://florawww.eeb.uconn.edu/acc_num/199500142.html	http://ctdata.lib.uconn.edu/
http://www.geocities.com/youandwhosearmy/nintendo.html	http://www.nintendo.com/contact
http://www.mandogroup.com/	http://www.omniture.com/policy.html
http://gyroscope.iwarp.com/	http://pages.ebay.com/help/policies/privacy-policy.html
http://www.techfore.co.uk	http://www.engnet.co.uk/
http://www.a1toolresharpening.com	http://www.4cdg.com/portfolio/index.html
http://www.hhonline.net	http://www.safesurf.com/
http://www.vicnet.net.au/~tvpc	http://e-newsletters.internet.com/samples/networkin

	g-news-html.html
http://httpd.chello.nl/hw.eghuizen/	http://www.inria.fr/multimedia/

Acreditamos que este número de passos é suficiente para que a página resultante esteja "distante" da página original. Um número muito grande tornaria o processamento muito demorado, e um número muito pequeno faria com que a probabilidade de que cada página *i* esteja na amostra fosse muito influenciada pela amostra de partida.

A seguir é mostrado um exemplo de 10 iterações, tendo a página *www.ic.unicamp.br* como ponto de partida. Como pode-se notar, a página resultante é "distante" da página inicial. A coleta foi realizada por *script*, executando o utilitário *wget* da GNU.

```
--15:34:29--  http://www.ic.unicamp.br:80/
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: 28,398 [text/html]

    0K -> .......... .......... .......                      [100%]


--15:34:32--  http://www.ic.unicamp.br:80/mata04
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 301 Moved Permanently
Location: http://www.dcc.unicamp.br/mata04/ [following]
--15:34:35--  http://www.dcc.unicamp.br:80/mata04/
          => `output
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: 5,185 [text/html]

    0K -> .....                                              [100%]

--15:34:38--  http://www.sbc.org.br:80/
          => `output
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]

    0K -> .......... .......

--15:34:41--  http://www.computerworld.com.br:80/AdPortalV3/
          => `output
Connecting to 146.250.159.233:3128... connected!
```

```
Proxy request sent, awaiting response... 200 OK
Length: 49,758 [text/html]

    0K -> ......... ......... ......... ......... ........    [100%]


--15:34:46--  http://aventura.terra.com.br:80/
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 301 Moved Permanently
Location: http://360graus.terra.com.br [following]
--15:34:48--  http://360graus.terra.com.br:80/
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: 47,742 [text/html]

    0K -> ......... ......... ......... ......... ......      [100%]

--15:34:52--  http://www.terra.com.br:80/sebastiaosalgado/
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: 541 [text/html]

    0K ->                                                      [100%]

--15:34:55--  http://www.exploratorium.edu:80/
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: 23,108 [text/html]

    0K -> ......... .........  ..                              [100%]

--15:35:01--  http://www.exploratoriumstore.com:80/
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]

    0K -> ......... .........  .

--15:35:04--  http://order.store.yahoo.com:80/cgi-bin/wg-order?explo
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]

    0K -> ......... ...

--15:35:08--  http://store.yahoo.com:80/explo/exstorin.html
          => `output'
Connecting to 146.250.159.233:3128... connected!
Proxy request sent, awaiting response... 200 OK
Length: unspecified [text/html]
```

```
OK -> .......... .......... .......... ......
```

Vale lembrar que, neste caso, o conjunto resultante de páginas escolhidas será uma amostra da Web, amostrada com probabilidades proporcionais aos PageRanks. Ou seja, o PageRank de uma página, é a propabilidade de que ela faça parte do conjunto amostrado.

6.2.3. Verificação

De posse das páginas selecionadas na fase de amostragem, deve-se verificar, para cada uma delas, se elas estão na base de dados do Google. Para isso, a abordagem escolhida foi montar, para cada página, uma expressão de busca que possa identificá-la o mais unicamente possível, e verificar as URLs dos resultados recebidos, para ver se pelo menos uma delas é equivalente à URL da página em questão.

Para a execução das buscas, foi utilizada uma API Java fornecida pelo Google [19], que possibilitou a automatização do experimento. A parte principal desta API é uma função que, dada uma cadeia de busca, efetua a consulta ao Google pela rede, e devolve uma lista das URLs encontradas (limitada em 10 entradas).

Como expressão de busca, foi escolhido o título da página (o trecho da mesma delimitado pelas chaves <title> e </title>). Esse título é submetido à pesquisa do Google com a opção de "busca pela frase exata", indicada colocando-se o título entre aspas (").

Para cada consulta feita, utilizando o título da página como frase de busca, as URLs encontradas pelo Google devem ser comparados com a URL inicial. A API utilizada só retorna os 10 primeiros resultados da busca. Ou seja, se a URL não estiver nos 10 primeiros resultados, a consulta é considerada "não conclusiva". De qualquer forma, como a expressão de busca utilizada (frase formada pelo título da página) tende a identificar fortemente a página, verifica-se que a chance é pequena de que ela esteja na base de dados e não seja classificada entre as 10 primeiras, fazendo com que a fração se consultas não conclusivas seja pequena.

6.3. Resultados

A tabela seguinte ilustra os resultados encontrados para uma pesquisa com base em 10000 páginas:

Páginas pesquisadas	Páginas encontradas na base do Google	Páginas **não** encontradas na base do Google	Consulta não conclusiva
10000	5961	3731	308

Foi verificado que em cerca de 3 % das buscas o número de resultados encontrados era maior que 10, e a URL não foi encontrada porque não estava nas 10 primeiras respostas (consulta não conclusiva). Portanto, na tabela de resultados, as porcentagens obtidas para páginas "encontradas" e "não encontradas" são, respectivamente, entre 60 % 63 %, e entre 37% e 40 %.

Podemos notar que entre 60 % e 63 % das páginas testadas estão na base de dados do Google e somam entre de 60 % e 63% do PageRank de todas as páginas Web. Ou seja, dada uma página qualquer, escolhida com probabilidade proporcional ao PageRank, há uma probabilidade de 60 % a 63 % de que ela esteja na base de dados do Google.

Lembramos mais uma vez que essa percentagem refere-se a probabilidade (PageRank) e não ao número de páginas. Nosso experimento não permite concluir qual a porcentagem das páginas da Web está presente na base do Google, mas apenas a probabilidade de que uma página esteja presente.

6.3.1. Precisão dos resultados

A amostra de 10000 páginas utilizada pode parecer muito pequena, pois corresponde a cerca de 0,0001 % do tamanho divulgado da base de dados do Google (cerca de 4 bilhões de páginas). Porém, não seria possível aumentar muito essa amostra por limitações de tempo e recursos. Em particular, o Google impõe um limite de uso para sua API de 1000 buscas diárias por usuário. Por outro lado, a amostra não é tão pequena assim. A teoria mostra que em experimentos utilizando "Simulação de Monte Carlo", como é o caso deste, o erro de amostragem na probabilidade calculada é proporcional a $1/\sqrt{N}$, onde N é o número de amostras. Ou seja, estima-se que a percentagem calculada (60 %) tenha um erro da ordem de 1 %.

6.4. Considerações finais

Neste capítulo avaliamos o grau de cobertura da *Web* pelo mecanismo de busca Google. Apesar das limitações apresentadas, foi possível ter uma idéia de que há uma porção razoável da Web (cerca de 40 %, em termos de probabilidade) que não é coletada pelo referido mecanismo. Devido à alta taxa de crescimento da *Web*, que dobra de tamanho a cada ano segundo [20], o problema pode se agravar. Dada a impossibilidade de se cobrir toda a Web, uma estratégia interessante pode ser tentar coletar as páginas mais "importantes". Isso aumentaria a probabilidade de que uma página procurada estivesse na base de dados, mesmo sem aumentar o número de páginas da mesma. Em [15], é mencionado que os algoritmos de análise de *links* podem ser usados para selecionar quais páginas devem ser coletadas primeiro. Desta forma, os algoritmos de análise de *links* poderiam, ajudar a minimizar os efeitos da baixa cobertura da *Web*, garantindo a coleta das páginas mais "importantes".

Capítulo 7 – Conclusão

Neste trabalho foi apresentada uma visão geral das tecnologias de busca na *Web*, enfocando as técnicas e algoritmos utilizados pelos mecanismos de busca.

A partir da cronologia dos mecanismos de busca, apresentada no Capítulo 2, foi possível perceber a evolução gradual desses mecanismos, que foram ganhando novas funcionalidades e melhorando a qualidade de suas respostas à medida em que aumentava seu uso comercial. O desenvolvimento recente desses mecanismos indica um bom potencial para pesquisas futuras.

A visão geral dos mecanismos de busca, apresentada no Capítulo 3, serviu para apresentar e discutir seus principais componentes e questões algorítmicas relacionadas. Foram levantadas várias questões técnicas, que devem ser alvo de estudos mais detalhados, como a coleta de páginas *Web*, o armazenamento, a indexação e a classificação dos documentos. Em especial, discutimos no Capítulo 4 o problema da classificação de documentos, utilizando os algoritmos de análise de *links*. Esses algoritmos foram os principais responsáveis pela melhoria da qualidade das respostas dos mecanismos de busca recentes, como o Google. Foram detalhados os dois mais importantes algoritmos de classificação por análise de *links*, o PageRank e o HITS.

A descrição detalhada da implementação do mecanismo de busca Google, realizada no Capítulo 5, mostrou os problemas práticos enfrentados na implementação, como por exemplo escalabilidade, e serve como referência para a construção de futuros mecanismos. Além disso, foram mostradas soluções práticas para questões tratadas de forma genérica no Capítulo 3, e foi possível também acompanhar a aplicação do algoritmo PageRank.

No Capítulo 6, foi feito um experimento para se avaliar quanto da *Web* é coberta por um mecanismo de busca comercial, no caso o Google que possui a maior base de dados atualmente. Pode-se perceber que mesmo esse mecanismo ainda deixa um porção razoável

da *Web* descoberta. Aqui, novamente, os algoritmos de análise de *links* podem ajudar a minimizar o problema garantindo a coleta das páginas mais importantes.

Uma questão interessante lançada pelo experimento de cobertura da *Web* é sobre a possibilidade de os mecanismos de busca genéricos como o Google continuarem a ser capazes de oferecer uma boa cobertura da *Web*, tendo em vista o crescimento contínuo desta. O experimento poderia ser repetido regularmente para apontar tendências. Se a tendência mostrada for a redução da cobertura, é possível que uma solução seja a segmentação (geográfica ou de assunto) desses mecanismos. Aliás, essa segmentação parece ser uma tendência, como comentado no Capítulo 2.

Entre as várias áreas promissoras para pesquisas futuras, gostaria de apontar três: o suporte à buscas de conteúdo multimídia (como imagens e áudio), a personalização das buscas por usuário e categorização automática de páginas por assuntos.

Em relação às buscas de conteúdo multimídia, podemos notar que alguns mecanismos (por exemplo o Google e o AllTheWeb) estão começando a oferecer busca de imagens e áudio. O problema é que a indexação e classificação desses arquivos é feita usando basicamente indexação de texto, por exemplo indexando o próprio nome do arquivo ou da referência na *tag* HTML, que será comparado à expressão de busca do usuário. Os resultados apresentados nem sempre são satisfatórios, o que demanda novas técnicas além das de busca em texto.

Outro desafio para os mecanismos de busca é personalizar as buscas de seus usuários, de acordo com seus perfis. Podemos identificar o início dessa preocupação quando os mecanismos começaram a utilizar *cookies* para guardar informações sobre as buscas realizadas pelos usuários. Porém, essa personalização pode e deve ir bem além disso. Pode-se, por exemplo, descobrir os interesses dos usuários a partir de buscas passadas e utilizar essa informação para classificar os resultados das novas buscas. Um exemplo interessante é o do Google que recentemente adquiriu um site especializado em redes de relacionamento

social, o Orkut [25]. Especula-se que o Google planeja usar os perfis de usuário do Orkut para personalizar suas buscas [26].

A categorização automática de páginas por assunto representa outra área importante de pesquisa para os mecanismos de busca. Desde a origem dos primeiros mecanismos, houve uma separação entre os serviços de diretório que ofereciam listas de navegação por assunto, em geral catalogadas manualmente, e os mecanismos de busca propriamente ditos, que buscam uma expressão em um repositório de dados gerado de forma autônoma. A questão é que por mais que os mecanismos de busca tenham evoluído, é muitas vezes mais eficiente/confortável para o usuário realizar a busca navegando em uma lista de assuntos. Isso explica a grande popularidade de serviços de diretório como o Yahoo. O problema aqui é que o crescimento da *Web* torna inviável a manutenção de diretórios catalogados manualmente. Uma possível solução tem sido apresentada por mecanismos como o Vivissimo [27], que trabalham basicamente como um mecanismo de busca normal, porém agrupam as respostas das buscas em categorias, oferecendo uma funcionalidade semelhante à dos serviços de diretório. Isso é possível, através do uso de algoritmos de análise de links, como o HITS descrito no Capítulo 4, que permitem a separação de tópicos a partir de um conjunto de páginas.

Referências

1. Michael L. Mauldin, "Lycos: Design choices in an Internet search service", IEEE Expert, Jan-Feb 1997, p. 8-11.

2. "A History of Search Engines", www.wiley.com/legacy/compbooks/sonnenreich/history.html

3. "Google History", http://www.google.com/corporate/history.html

4. "Google Tweaks Its Number to 3.3 Billion and Adds Supplemental Results", http://www.searchengineshowdown.com/newsarchive/000713.shtml

5. "FAST History", http://www.clubi.ie/webserch/engines/fast/index.htm

6. "The History of Teoma", http://sp.teoma.com/docs/teoma/about/developmentteamhistory.html

7. CiteSeer, http://citeseer.com/

8. TodoBR, http://www.todobr.com.br

9. Arasu, Arvind; Cho, Junghoo; Garcia-Molina, Hector; Paepcke, Andreas; Raghavan, Sriram; "Searching the Web"; Standford University, 2000.

10. Sergey Brin and Lawrence Page, "The Anatomy of a large-scale hypertextual web search engine", Proceedings of 7th World Wide Web Conference, 1998.

11. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. "The PageRank citation ranking: Bringing order to the web". Technical report, Computer Science Department, Stanford University, 1998.

12. Jon Kleinberg. "Authoritative sources in a hyperlinked environment". *Journal of the ACM*, 46(5):604-632, November 1999.

13. RFC 1950 – "ZLIB Compressed Data Format Specification version 3.3", http://www.ietf.org/rfc/rfc1950.txt

14. A Standard for Robot Exclusion, Koster, M., "A Standard for Robot Exclusion", http://www.robotstxt.org/wc/norobots.html, June 1994.

15. Henzinger, M. "Link Analysis in Web Information Retrieval". In: IEEE Data Engineering Bulletin, 23(3):3-8, 2000.

16. B. Amento, L. Terveen, and W. Hill. "Does authority mean quality? Predicting expert quality ratings of web documents". In *Proceedings of the 23rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'00)*, pages 296–303.

17. "Google is most popular but others may do it better", http://www.myrtlebeachonline.com/mld/sunnews/2003/08/24/business/6605978.htm

18. "Análise Comparativa de Máquinas de Busca para a Web brasileira", Akwan Information Technologies, http://www.akwan.com.br/mix_analise.shtml

19. Google Web APIs, http://www.google.com/apis/

20. Z. Smith, The truth about the Web; crawling towards eternity, *Web Techniques Magazine*, May 1997, http://www.newarchitectmag.com/documents/s=6600/new1013637947/sidebar.htm

21. Search Engine Showdown, http://www.searchengineshowdown.com

22. Search Engine Watch, http://www.searchenginewatch.com

23. Search Engine Guide, http://www.searchengineguide.com

24. Dmoz Open Directory Project, http://www.dmoz.org

25. Orkut, www.orkut.com

26. Sites de busca e lojas on-line investem na personalização de serviços para atrair clients. Revista Época, edição 322, 19/07/2004 http://revistaepoca.globo.com/Epoca/0,,EPT760582-1664,00.html

27. Vivissimo Clustering Engine, www.vivissimo.com