Algoritmos e Estruturas de Dados 3
Trabalho Prático 0
Similaridade de Textos

André Taiar Marinho Oliveira

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

taiar@dcc.ufmg.br

Resumo. *Recuperação de Informação (RI) é uma área da computação que lida com o armazenamento de documentos e a recuperação automática de informação associada a eles. É uma ciência de pesquisa sobre busca por informações em documentos, busca pelos documentos propriamente ditos, busca por metadados que descrevam documentos e busca em banco de dados, sejam eles relacionais e isolados ou banco de dados interligados em rede de hipermídia, tais como a World Wide Web.*
Uma boa forma de classificar, definir e reconhecer semelhanças entre textos é definindo o seu assunto de acordo com as suas palavras chaves. Com o crescimento da utilização de sistemas de Recuperação de Informação e sua utilização massiva na atualidade, o problema de se descobrir palavras-chave de um texto tem sido muito explorado.

1. Introdução

Neste trabalho foi desenvolvido um sistema que, dado um conjunto de textos qualquer, encontramos para cada elemento o seu tópico/palavras-chaves. De acordo com a ocorrência das palavras-chaves nos textos, podemos definir algumas métricas de semelhança entre os eles e, assim, identificar textos que têm conteúdo semânticamente relacionado de uma forma automática.

Esse tipo de análise de conteúdo semânticamente relacionado vem sendo amplamente utilizado para conjuntos cada vez maiores de informação. A Web é um exemplo de rede com conteúdo vasto e que crescimento vertiginoso. Organizar e encontrar toda essa informação de forma relevante através em um conteúdo tão vasto e fragmentado em sites, blogs, wikis, redes sociais, etc, é uma importante aplicação deste trabalho e do que é desenvolvido com relação à Recuperação de Informação.

Aqui foram implementados algoritmos simples operando sobre uma estrutura de dados de pesquisa em memória bem eficiente e dinâmica. Foram avaliadas métricas simples de classificar o conteúdo dos textos por palavras-chaves e, posteriormente calcular o quão relacionados estes textos são. Ao final, farei a análise de quão boa foram as métricas propostas para avaliar a semelhança entre os conteúdos e variar as diversas considerações para obter uma visão mais ampla do funcionamento do sistema.

2. Solução Proposta

A solução proposta para os cálculos sobre as frequencias e ocorrencias das palavras utiliza um conjunto de estruturas de dados que contém informações sobre o termo em si (a palavra especificamente), o texto aonde ela ocorreu e quantas vezes ocorreu naquele texto.

Em diferentes partes do algoritmo, esta estrutura está organizada de diferentes formas: às vezes como uma árvore binária, como um vetor de árvores binárias, como uma lista encadeada e como um vetor nos casos em que exigiam ordenação.

Para criar uma estrutura de pesquisa eficiente tanto em termos de espaço quanto custo computacional, optei por armazenar todo o índice de ocorrências dos termos em forma de uma árvore binária. Cada nó dessa árvore tem uma lista encadeada de ocorrências do termo que armazenam em cada célula o texto em que aquele termo apareceu e quantas vezes apareceu nesse texto.

Primeiramete, é criado um índice lendo um por um os arquivos passados como entrada para serem analisados. Cada termo desse arquivo é inserido na estrutura da árvore e servirá como nosso índice/vocabulário.

1: Leitura do índice

```
foreach Documento do
    foreach termo válido do
        if o termo já apareceu em algum texto then
            if o termo já apareceu neste texto then
                incrementa o contador de ocorrências correspondente ao termo no texto;
            end
            else
                insere o texto na lista e contabiliza o contador de ocorrências para este
                termo;
            end
        end
        else
            insere o nó referente ao termo, insere o texto na lista e contabiliza o contador
            de ocorrências para este termo;
        end
    end
end
```

Em uma aplicação real, muitos passos podem ser feitos para determinar efetivamente o que é e o que não é um termo relevante e, dessa forma, poupar processamento e espaço em processos de indexação. Alguns desses passos correspondem à análise de Stop Words [1], remoção de prefixos e sufixos das palavras [2], reconhecimento sintático, entre outras. Em nosso sistema, a única análise feita é quanto ao número de caracteres que a palavra contém. Para uma aplicação padrão da indexação, utilizei o mínimo de 3 caracteres por palavra (parâmetro que será variado nas avaliações experimenais).

Após indexar todos os textos, conseguimos obter vários parâmetros úteis para expressar a relevância das palavras-chave neste contexto. O primeiro aspecto a ser levado em consideração é o número de ocorrências de uma palavra. Como sugerido na especificação, assumi que, para um termo ser considerado relevante ele deveria aparecer na minoria dos textos apresentados (aparecer em menos de 50% dos textos) aumentando o índice de discriminação que ele potencialmente tem. Esse parâmetro foi colocado (assim

[1] Stop Words são palavras consideradas sem valor semântico para a análise de tópico de um texto (como artigos e preposições, por xemplo). Referência.

[2] É um processo que simplifica as palavras removendo seu prefixo e sufixo (caso tenha) e dessa forma reconhece o padrão de formação daquela palavra.

como o tamanho mínimo dos termos) de forma a ser facilmente modificado para obtermos melhores análises experimentais.

Após termos uma referência sobre quais palavras serão potencialmente relevantes para analisarmos as semelhanças entre os textos, podemos reconstruir novamente um índice. Dessa vez, eu analiso cada texto e incorporo ao seu índice apenas as palavras relevantes, colocando cada índice em um vetor que terá o índice de um texto em cada posição.

Com estes elementos calculados podemos partir para a finalização do trabalho. Para retornar as palavras chaves de cada texto, basta percorrer o índice individual de cada texto. Porém, como as palavras chaves precisam ser retornadas devemos primeiramente ordená-las de acordo com as suas ocorrências naquele texto. Para isso, lemos o índice e armazenamos os termos e suas ocorrências um um vetor que será posteriormente ordenado decrescentemente, obtendo dessa forma as palavras-chave de um texto ordenadas por sua relevância.

O segundo item a ser retornado pelo programa é um arquivo contendo a lista dos textos analisados se os texos mais semelhantes à ele. O número de textos a ser retornado é arbitrário. Nas execuções do trabalho, retornamos quantidades de 5 a 8 indicações de textos.

2: Identificação de textos semelhantes

```
foreach Documento do
    PalavrasChave ⟵ recebe as palavras-chave do Documento e o seu número de
    ocorrências naquele texto;
    OrdenaOcorrenciaDecrescente(PalavrasChave);
    Imprime lista de palavras-chave;
    foreach palavras-chave do Documento do
        Ocorrencias ⟵ vetor de textos e numero de ocorrencias da palavra-chave por
        texto;
        foreach Documento em que a palavra-chave ocorreu do
            PotencialDeSemelhanca[DocumentodeOcorrencia]+ ⟵ ocorrencias da
            palavra-chave no Documento de Ocorrencia;
        end
    end
    OrdenaPotenciaDecrescente(PotencialDeSemelhanca);
    Imprime lista de N Documentos mais similares;
end
```

Encerrada a explicação sobre os principais algoritmos e as estruturas de dados utilizados, e como eles se combinaram na solução do problema, segue abaixo uma breve análise sobre a análise de complexidade dos principais algoritmos e mais alguma explicação sobre as estruturas que eventualmente tenha faltado acima.

2.1. Análise da solução

2.1.1. Ordenação

O algoritmo de ordenação utilizado foi o Quicksort que tem ordem de complexidade $n(\log(n))$ para casos médios.

2.1.2. Dicionário

Armazena cada termo, o documento em que ocorreu e quantas vezes ocorreu naquele documento. O Dicionário foi organizado como uma árvore binária de pesquisa em que cada termos e suas ocorrências estão em um nó da árvore. Dessa forma, cada nó ainda conta com 2 ponteiros (um que aponta para a sub-árvore da direita e um que aponta para a sub-árvore da direita).

3: Nó da Árvore
termo;
lista de ocorrencias;

2.1.3. Lista de Ocorrências

Precisamos de uma lista de ocorrências para guardar todas as ocorrências que variavam por termo e documentos. Cada nó do dicionário contém uma lista de ocorrências.

Tabela 1. Análise de Complexidade do Dicionário

Operação	Custo (casos médios)
Inserção na Árvore	$O(\log(n))$
Pesquisa na Árvore	$O(\log(n))$
Caminhamento na Árvore	$O(n)$

4: Item da lista
identificador do documento;
ocorrências no documento;

2.2. Cdigo

2.2.1. Arquivos .c

- **principal.c:** Arquivo principal do programa que implementa o simulador do elevador.
- **simulador.c:** Define as funes relacionadas simulao do elevador.
- **lista.c:** Define funes relacionadas a manipulao de um TAD lista implementada atravs de apontadores.
- **fila.c:** Define as estruturas de dados e cabealhos de funes relacionadas a manipulao de um TAD fila implementada atravs de apontadores.

2.2.2. Arquivos .h

- **simulador.h:** Define as estruturas de dados e cabealhos de funes relacionadas a simulao do elevador

Tabela 2. Análise de Complexidade da Lista

Operação	Custo
Inserção na Lista	$O(n)$
Remocao da Lista	$O(n)$
Pesquisa na Lista	$O(n)$

- **lista.h:** Define as estruturas de dados e cabealhos de funes relacionadas manipulao de um TAD lista implementada atravs de apontadores.
- **fila.h:** Define as estruturas de dados e cabealhos de funes relacionadas a manipulao de um TAD fila implementada atravs de apontadores.

2.3. Compilao

O programa deve ser compilado atravs do compilador GCC atravs de um makefile ou do seguinte comando:

```
gcc principal.c simulador.c fila.c lista.c -o tp0
```

2.4. Execuo

A execuo do programa tem como parmetros:

- Um arquivo de descrio do ambiente.
- Um arquivo de requisies.
- Um arquivo de sada.

O comando para a execuo do programa da forma:

```
./tp0 -a <arquivo de ambiente> -r <arquivo de requisicoes> -s <arquivo de saida>
```

2.4.1. Formato da entrada

O arquivo de descrio do ambiente contm apenas uma linha onde so definidos o nmero de andares do prdio atendido pelo elevador e a capacidade de elevador, separados por espao. J o arquivo de requisies contm o nmero de requisies na primeira linha seguido de cada uma das requisies.

Cada requisio contm um identificador, o instante de chamada do elevador (que tem como base um temporizador sequencial), o andar de origem do passageiro e o andar desejado. Um exemplo de arquivo de requisies dado a seguir:

```
3
0 0 0 10
1 3 0 3
2 5 13 15
```

2.4.2. Formato da sada

A sada do programa, armazenada em um arquivo de sada, contm informaes sobre todas as requisies atendidas. Para cada requisio as informaes de interesse so o identificador da

requisio, o tempo de espera de carga (at que a pessoa entre no elevador) e o tempo de espera dentro do elevador. A seguir, a sada correspondente entrada apresentada na seo anterior.

```
id :0
espera carga :1
espera elevador :11

id :1
espera carga :27
espera elevador :4

id :2
espera carga :11
espera elevador :31
```

3. AVALIAO EXPERIMENTAL

Nesta seo ns avaliamos a estratégia de controle de elevador proposta e o simulador implementado em termos do tempo de execuo (em segundos) e do tempo total de espera dos usurios (em Jepslons) para diferentes nmeros de andares, capacidades do elevador, nmeros de requisies e popularidades de trajetos. O tempo total de espera a soma dos tempos de espera para entrar no elevador e para chegar ao andar desejado. Os experimentos foram executados em um computador pessoal com sistema operacional Suse Linux, processador AMD Athlon 3500+ 64 bits e 1GB de memria principal.

Como destacado anteriormente, o tempo de execuo prov, alm do tempo para que toda a simulao seja realizada, uma aproximao sobre o quo demorado o atendimento de todas as requisies do elevador (j que cada instante de tempo simulado em uma iterao). J o tempo de espera pelo elevador uma maneira simples de avaliar a qualidade do servio prestado. Outros critrios como utilizao do elevador, distncia total percorrida pelo elevador, dentre outros, poderiam ser avaliados. Alm disso, tambm seria interessante considerar como o tempo entre as requisies afeta o elevador, no entanto, devido restries de tempo e espao, no avaliaremos todos os parmetros da simulao.

As entradas para a simulao do elevador foram geradas a partir de um gerador sinttico de requisies. Esse gerador recebe como parmetros: (1) o nmero de andares, (2) a capacidade do elevador, (3) um atributo que determina a popularidade dos trajetos e (4) o nmero de requisies. Como sada, o gerador cria um arquivo de ambiente e um arquivo de requisies. O atributo que determina a popularidade dos trajetos pode ser: (1) escolha aleatria, ou seja, trajetos escolhidos aleatoriamente, (2) probabilidade do trajeto proporcional ao tamanho, ou seja, trajetos mais longos so mais frequentes, e (3) probabilidade do trajeto inversamente proporcional ao tamanho do trajeto, o que resulta em maior frequncia de trajetos curtos.

Em todos os experimentos ns variamos um parmetro e mantivemos os outros constantes. Cada experimento foi executado 5 vezes e foram tomados os valores mdios. Nosso objetivo reduzir o impacto de propriedades de uma entrada especfica produzida pelo gerador de entradas. Alm do tempo de execuo, foi medido tambm o tempo de usurio, que o tempo gasto pelo processo do programa. O tempo de sistema, utilizado na realizao de tarefas no nvel do kernel, tambm foi medido, mas no mostrado nos resultados pois muito

menor que os tempos de usurio e execuo. O tempo mdio entre as requisies do elevador foi mantido constante, igual a 3 Jepslons.

A Tabela 3 mostra como o aumento do nmero de andares afeta os tempos de execuo, usurio e espera dos usurios. O nmero de andares varia de 25 a 40. O nmero de requisies 50000, a capacidade do elevador de 10 usurios e a escolha dos trajetos aleatria. importante notarmos que, ao aumentarmos o nmero de andares, o tamanho mdio dos trajetos tende a aumentar. O nmero de andares afeta significativamente os tempos de execuo, usurio e espera. Experimentos com nmero de andares maior que 40 requerem tempos de execuo muito longos. Nossa hiptese de que esse impacto se deve ao aumento dos trajetos pode ser explicada pelo grande aumento do tempo de espera, ou seja, os usurios esto permanecendo no elevador por um longo perodo de tempo.

#andares	Tempo execuo(seg.)	Tempo usurio(seg.)	Tempo de espera (Jepslons)
25	2,03	1,85	56,19
30	3,00	2,77	76,13
35	6,96	6,69	140,37
40	2068,28	1952,83	1947,29

Tabela 3. Tempo de simulao e de espera para diferentes nmeros de andares

A Tabela 4 mostra impacto da capacidade do elevador sobre o tempo de simulao e o tempo de espera. A capacidade varia de 10 a 13 lugares. O nmero de andares 40, o nmero de requisies 50000 e a escolha dos trajetos aleatria. Como era esperado, o tempo de simulao e o tempo de espera so inversamente proporcionais capacidade do elevador.

capacidade	Tempo execuo(seg.)	Tempo usurio(seg.)	Tempo de espera (Jepslons)
10	2068,28	1952,83	1947,29
11	17,63	17,16	234,37
12	5,43	5,20	121,51
13	4,31	4,07	101,04

Tabela 4. Tempo de simulao e de espera para diferentes capacidades do elevador

A Figura **??** avalia o tempos de simulao e de espera em termos do nmero de requisies. O nmero de requisies varia de 30000 a 50000. O nmero de andares 40, a capacidade do elevador 10 e a escolha dos trajetos aleatria. Como indicado na anlise de complexidade, o custo computacional do algoritmo polinomial em relao ao nmero de requisies. O crescimento do tempo de execuo se aproxima de uma funo quadrtica, tambm de acordo com a anlise de complexidade. J o tempo de espera aumenta linearmente com o nmero de requisies, uma hiptese para esse resultado que quanto maior o nmero de requisies, mais tempo necessrio para que o simulador gere todas as requisies para o elevador. A partir do momento que todas as requisies foram geradas, a tendncia que a fila de requisies se reduza gradativamente. No entanto, um longo perodo com gerao constante de requisies pelo simulador congestiona a fila de registros de requisio do elevador, aumentando o tempo mdio de espera.

Como ltima parte de nossa avaliao experimental, ns analisamos como a popularidade dos trajetos afeta os tempos de execuo e espera. Trs relaes entre as frequncias dos trajetos e seus tamanhos foram consideradas: (1) trajetos escolhidos aleatoriamente,

(2) probabilidade do trajeto diretamente proporcional sua distncia e (3) probabilidade do trajeto inversamente proporcional sua distncia. O nmero de andares 40, a capacidade do elevador 10 e o nmero de requisies 15000. A Tabela 5 mostra os resultados obtidos. Podemos notar que a distribuio das requisies afeta significativamente os tempos de execuo e espera, principalmente o tempo de execuo. Quando trajetos longos se tornam mais frequentes, o tempo de execuo cresce, no somente devido ao aumento do nmero de iteraes de simulador, mas tambm do custo da manipulao de um grande nmero de requisies. J o tempo de espera, menos afetado, aumenta devido uma maior carga de trabalho para o elevador.

Frequncia tamanho dos trajetos	Tempo de execuo(seg.)	Tempo de espera(seg.)
Aleatria	73,52	778,48
Proporcional distncia	9.011,42	7.011,46
Inv. Proporcional distncia	2,24	100,61

Tabela 5. Tempos de simulao e de espera para diferentes frequncias de tamanhos de trajeto

4. CONCLUSO

Neste trabalho ns descrevemos um simulador discreto de eventos e uma estratégia de controle de um elevador. O simulador gera requisies a serem atendidas ao longo do tempo de simulao e o elevador atende essas requisies de acordo com uma estratégia similar utilizada pelos elevadores convencionais.

O trabalho atingiu seus principais objetivos: a prtica da linguagem de programao C e o estudo de problemas complexos. O autor do trabalho j havia implementado alguns programas nessa linguagem e, portanto, no teve grandes problemas na implementao do trabalho. J a soluo do problema demandou a modelagem de diversos cenrios, de acordo com diferentes estados do elevador. Uma importante deciso foi a simulao de cada Jepslon e no de instantes especficos associados a eventos, o que facilitou o projeto da soluo.

A anlise de complexidade da soluo proposta no foi uma tarefa simples, j que o custo computacional da soluo est sujeito diversas caractersticas da entrada. Por outro lado, o impacto de diferentes fatores associados entrada pde ser avaliado atravs de diversos experimentos realizados.

Algumas melhorias que poderiam ser consideradas neste trabalho so:

- Uma modelagem mais complexa do tempo de simulao em termos da popularidade dos trajetos, do tempo entre as requisies, dentre outros;
- A anlise do comportamento do elevador considerando caractersticas como a distncia percorrida e a taxa de utilizao do elevador.

Referências

[Cormen 2001] Cormen, T. (2001). *Introduction to algorithms*. MIT press.

[Jain 1991] Jain, R. (1991). *The art of computer systems performance analysis*. John Wiley & Sons New York.

[Kernighan et al. 1988] Kernighan, B., Ritchie, D., and Ejeklint, P. (1988). *The C programming language*. Prentice-Hall Englewood Cliffs, NJ.

[Sedgewick 2001] Sedgewick, R. (2001). Algorithms in C.

[Ziviani 2007] Ziviani, N. (2007). *Projeto de Algoritmos com implementações em Pascal e C*. Pioneira Thomson Learning.