Recuperação de Informação – Relatório do Trabalho Prático 1

Danilo Ferreira e Silva[1]

[1]Departamento de Ciência da Computação – UFMG

danilofes@gmail.com

Resumo. *Este relatório descreve o desenvolvimento do trabalho prático 1, proposto na disciplina Recuperação de Informação. O trabalho consistiu da implementação de um indexador e processador de consultas simples, aplicando o conhecimento aprendido em aula. Experimentos realizados no sistema desenvolvido mostraram que ele é capaz de gerar índices, na forma de listas invertidas, para coleções da ordem de 1 milhão de documentos em aproximadamente uma hora.*

1. Introdução

Falar alguma bobagem.

2. Arquitetura do sistema

O sistema foi construído na linguagem C++, conforme especificado, e é constituido de dois executáveis. O primeiro deles é o indexador da coleção de documentos, que gera um arquivo de listas invertidas ao final do processamento. O segundo é o processador de consultas, que usa o arquivo gerado pelo indexador para responder a consultas do usuário de forma interativa, via linha de comando.

2.1. Módulos e dependências

Na figura 1 são apresentados os módulos e dependências entre eles.

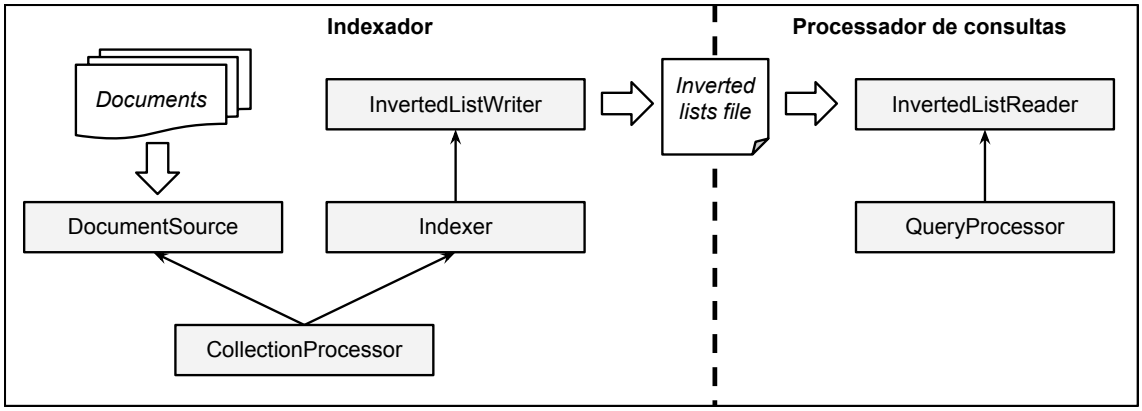

Figura 1. Diagrama de dependências de módulos do sistema

DocumentSource Abstração que representa uma fonte de documentos, identificados por uma URL e cujo conteúdo é uma sequência de caracteres. A implementação lê os documentos usando a RICPlib.

CollectionProcessor Módulo principal que lê um conjunto de documentos fornecidos pelo *DocumentSource*, transforma-os em uma sequência de termos e alimenta o *Indexer*.

Indexer Recebe um conjunto de termos dos documentos e faz a indexação. Ao final escreve as listas invertidas usando o *InvertedListWriter*.

InvertedListWriter Escreve o arquivo de listas invertidas.

InvertedListReader Lê o arquivo de listas invertidas.

QueryProcessor Processa uma consulta e devolve os resultados.

2.2. Dependência a bibliotecas de terceiros

O sistema foi contruído usando a API padrão C++11. Além disso, foram utilizadas as bibliotecas listadas a seguir.

RICPlib Biblioteca fornecida na especificação para ler a coleção de documentos comprimida.

zlib Dependência indireta induzida pela RICPlib.

htmlcxx Parser de documentos HTML.

gmock Biblioteca para criação de *mock objects* (usada apenas em testes unitários).

3. Algoritmos e estruturas de dados utilizados

O processo de indexação pode ser subdividido nas três etapas descritas abaixo:

1. Leitura dos documentos e escrita das triplas no arquivo temporário.
2. Ordenação das triplas, subdivididas em vários blocos de acordo com o tamanho do buffer.
3. *Merge* dos blocos, e geração das listas invertidas.

Na etapa 1, a principal estrutura de dados envolvida é o vocabulário. Ele é representado internamente por uma tabela *hash*, onde a chave é o próprio termo. Cada entrada da mesma também contém um identificador numérico atribuído ao termo e um contador de ocorrências do termo em documentos. Adicionalmente, ao processar um documento, é criado uma estrutura para guardar o vocabulário local, também usando uma tabela *hash*. Nesta a chave é o identificador do termo, e a entrada é um contador de frequência do termo no documento.

Portanto, para cada termo de cada documento, consulta-se primeiro o vocabulário para obter o identificador do termo (ou criar um novo). Em seguida é consultado o vocabulário local, para atualizar a frequência do mesmo no documento. Ambas as operações são $O(1)$, portanto a primeira etapa tem custo de tempo linear com relação ao número total de termos. O custo de memória é proporcional ao número de termos distintos.

3.1. Ordenação externa das triplas

As etapas 2 e 3 do algoritmo são na verdade uma ordenação externa. Para que seja viável indexar uma grande coleção de documentos, o índice não pode ser mantido todo em memória. Isso torna necessário o uso da ordenação externa.

A etapa 2 consiste na primeira etapa da ordenação das triplas no arquivo invertido, onde o conjunto de triplas são divididos em blocos menores de tamanho constante k. Para

cada um desses blocos, é feita uma ordenação em memória usando o algoritmo *heapsort*, cujo custo é $O(n \log n)$.

Consideranto T o número total de triplas, o custo dessa etapa é aproximadamente:

$$\lceil T/k \rceil k \log k$$

que é linear para efeitos práticos, pois $\log k$ é uma constante.

Finalmente na etapa 3, é feita uma intercalação de múltiplos caminhos nos $R = \lceil T/k \rceil$ blocos ordenados em disco. Para isso é utilizado um *heap* de tamanho R. Para cada tripla retirada do mesmo, ele é reconstruído com custo logaritmico. O custo total pode ser aproximado como:

$$T \log \lceil T/k \rceil$$

4. Avaliação

Para avaliar o desempenho do sistema, foram executados experimentos com tamanhos variados de coleções de documento, bem como com a coleção completa fornecida para testes.

Tabela 1. Tempo de execução em segundos para entradas de tamanho variado

Documentos	Etapa 1	Etapa 2	Etapa 3	Total
50000	80	34	23	234
100000				
150000				
200000				
1000000				

Tabela 2. Tempo de execução em segundos para *buffer* de tamanho variado

Documentos	Etapa 1	Etapa 2	Etapa 3	Total
50000	80	34	23	234
100000				
150000				
200000				
1000000				

5. References

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [Knuth 1984], [Boulic and Renault 1991], and [Smith and Jones 1999].

Referências

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.