MAC323/BCC QUARTO EXERCÍCIO-PROGRAMA

INDEXAÇÃO DE TEXTOS

Y. KOHAYAKAWA

Data de entrega. Você deve entregar este EP até as 18:00 do dia 5/7/2001 (quinta-feira), na secretaria do Departamento de Ciência da Computação, sala 256, bloco A. Você tem exatamente 3 semanas para fazer este EP.

1. Introdução

Neste exercício-programa, você implementará um indexador de textos. Você deve usar o Capítulo 15 do Sedgewick como fonte da base teórica (estude especialmente a Seção 15.5).

2. O problema a ser resolvido

Suponha que temos um arquivo-texto grande, 'mais ou menos estático' (por exemplo, um dicionário ou mesmo uma enciclopédia, ou uma lista telefônica, que, digamos, é atualizada uma vez por dia). Queremos criar um esquema em que a busca de texto neste arquivo seja extremamente rápida, e também queremos que o esquema não seja custoso do ponto de vista de memória. Queremos permitir que o usuário possa buscar frases ("Revolução Francesa" ou "Santo Tomás de Aquino") e também frases com o caracter especial `"*"` para indicar um (exatamente um) caracter qualquer.

Como os exemplos acimas ilustram, a ênfase nesta aplicação é a seguinte: supomos que o texto a ser indexado altera-se pouco, mas fazemos buscas freqüentemente. Desta forma, o tempo para se montar o *índice* pode ser maior, mas as buscas devem ser extremamente rápidas.

Dadas as hipóteses acima, é natural que armazenemos o índice em um arquivo (em memória secundária), para que posteriormente possamos fazer buscas no texto sem gerar o índice novamente. O índice de que estamos falando acima deve ser uma estrutura de dados que nos permita fazer buscas eficientes de frases (e padrões com `"*"`). *Você pode supor que os textos a serem processados são tais que os índices cabem na memória pricipal* (desde que seu esquema de indexação seja razoável). Você pode supor que o esquema de memória virtual de seu sistema operacional será suficiente para tratar arquivos maiores.

O arquivo a ser armazenado para buscas posteriores deve ser um arquivo binário, que é uma "cópia" da estrutura de dados acima (o índice). Este arquivo binário deve ser tal que o seu sistema pode recuperar o índice facilmente a partir dele.

Date: Versão de 14 de junho de 2001.

3. Os programas que você deve escrever

É natural que você escreva dois programas neste EP: `ep4a` e `ep4b`. O primeiro programa deve ter como entrada um arquivo texto, digamos `foo.txt`, e deve produzir uma estrutura de dados que permita as buscas discutidas acima. Ele deve também produzir um arquivo binário, digamos `foo.idx`, que é uma "imagem" desta estrutura de dados (esta é a única saída do `ep4a`).

O seu programa `ep4b` deve ler `foo.idx` e também deve receber no `stdin` uma lista de strings. A saída do `ep4b` deve ser assim: para cada string s dado, o `ep4b` deve produzir um inteiro N e uma seqüência de inteiros $n_0 < n_1 < \dots$ tais que a i-ésima ocorrência daquele string s no texto (`foo.txt`) começa no n_i-ésimo caracter do texto. Por exemplo, se s é o string `"a"`, então N deve ser o número de ocorrências de `"a"` no texto dado; a primeira ocorrência desse caracter é na posição n_0 do texto, a segunda é na posição n_1, etc.

Você deve implementar uma opção de linha de comando para o `ep4b` (digamos `-l`) que faz com que, em vez dos inteiros n_i acima, o `ep4b` com a opção `-l`m imprima m linhas antes e depois da linha em que o string procurado s ocorre (isto é, o `ep4b` fornece os "contextos" em que s ocorre).

Ambos os programas `ep4a` e `ep4b` devem ter acesso ao arquivo-texto original: isto é claro no caso do `ep4a`; no caso do `ep4b`, a idéia é que `foo.idx` seja apenas um índice para facilitar acesso ao texto dado, isto é, o arquivo `foo.idx` não precisa "conter" o arquivo `foo.txt`. (Veja a Seção 12.7 do Sedgewick).

3.1. **Scripts.** É natural que queiramos minimizar o tamanho de `foo.idx`. Você pode portanto escrever um pequeno script que executa o `ep4a` para produzir `foo.idx` e depois chama um programa de compressão (por exemplo, o `gzip` ou o `bzip2`) para comprimir `foo.idx`. Você pode escrever um segundo script para as buscas: esse script primeiro descomprimiria o índice e chamaria o `ep4b`. Você precisaria disponibilizar apenas estes scripts para o usuário.

Uma medida razoável para a *eficiência de espaço* de seu sistema é a razão dos tamanhos do índice comprimido e do texto original (digamos, comprimido).

4. Um sistema de exemplo

Um sistema sofisticado que pode servir como um exemplo (ou inspiração) é o sistema `MG` do *Managing Gigabytes*, um excelente livro sobre compressão e indexação de textos e imagens. Veja o endereço http://www.cs.mu.OZ.AU/mg/

5. Observações

1. *Este EP pode ser feito em grupos de três.*
2. Seja cuidadoso com sua programação (correção, documentação, apresentação, clareza do código, etc), dando especial atenção a suas estruturas de dados. A correção será feita levando isso em conta.
3. Comparem entre vocês o desempenho de seus programas.
4. Entregue o seu EP seguindo os moldes usuais: listagem impressa, cabeçalho claro, disquete com os arquivos (coloque um `Makefile`) e eventuais arquivos de teste.
5. Não deixe de entregar um *relatório* para discutir o seu EP. Por exemplo, faça testes e determine a razão típica entre o tamanho do arquivo `foo.idx` comprimido e do arquivo `foo.txt` (original e comprimido). Quanto demora

o seu sistema para processar um número grande de buscas (algumas centenas e alguns milhares—imagine que seu sistema está sendo usado via rede por milhares de usuários). Quão sensível é o tempo de busca em função do comprimento do string procurado? Tipicamente, quantos caracteres do string procurado é examinado em cada busca?

Quanto tempo o seu sistema demora para produzir o arquivo índice?

Observação final. Enviem dúvidas para a lista de discussão da disciplina. Eventualmente, ajustes no enunciado ocorrerão ao discutirmos este EP na lista.