MAC323 - Estruturas de Dados

Primeiro semestre de 2006

Segundo Exercício-Programa – Devolução: 28 de maio

Verificador Ortográfico

1. Introdução

Neste exercício-programa, você estudará o uso de *árvores binárias de busca* (ABBs) na implementação de tabelas de símbolos, dando especial atenção à eficiência de tais implementações.

2. O problema a ser resolvido

O programa que você escreverá neste exercício deve resolver o seguinte problema: (*i*) a entrada de seu programa é um texto e uma coleção de dicionários (listas de palavras) e (*ii*) a saída de seu programa deve ser uma lista contendo as palavras do texto que não ocorrem em nenhum dos dicionários fornecidos.

2.1. **Um programa exemplo.** Inicialmente, você deve estudar cuidadosamente o programa `wordtest` escrito por Knuth, disponível em

`http://www.ime.usp.br/~cris/mac323/eps/wordtest`

O programa do Knuth está escrito em sua linguagem de programação favorita, o CWEB (`http://www-cs-staff.Stanford.EDU/~knuth/cweb.html`). Veja a página

`http://www.ime.usp.br/~pf/CWEB/`

do professor Paulo Feofiloff sobre o CWEB.

O programa do Knuth usa uma estrutura chamada *treap*. Você não precisa estudar o funcionamento desta estrutura. Você deve, entretanto, entender que um treap é usado no programa do Knuth para implementar uma tabela de símbolos. Você também deve estudar os pontos finos da funcionalidade do programa do Knuth. Por exemplo, o que são as opções de linha de comando de `wordtest`?

3. O seu programa

A sua tarefa principal neste exercício-programa é escrever *um clone do programa do Knuth*. Do ponto de vista do usuário, seu programa deve se comportar exatamente da mesma forma que o `wordtest`.

As diferenças entre seu programa, digamos `ep3`, e o programa do Knuth devem ser as seguintes:

(1) O `wordtest` supõe que as palavras do texto de entrada vêm uma por linha. Você deve implementar uma opção de linha de comando adicional, digamos `-t`, para permitir que o texto de entrada seja um arquivo texto "normal", com várias palavras por linha. Assim, quando fazemos

```
ep2 -t [outras opções e argumentos]
```

o seu programa deve primeiro isolar as palavras do texto antes de processá-las.
Observação. Você pode achar interessante estudar o programa `tr` do Unix; em particular, estude o que faz a chamada

```
tr -cs a-zA-Z '\n'
```

Os programas `tr` e `wordtest` em conjunto podem simular o seu programa com a opção `-t`.

(2) A tabela de símbolos em seu programa deve ser implementada em um módulo separado, digamos `ST.c`, e o acesso a ela deve ser estritamente através de uma interface, digamos `ST.h`.

(3) Você deve escrever duas implementações de `ST.c`. Uma delas deve usar uma ABB sem balanceamento. A segunda deve usar *alguma* ABB balanceada (fica a seu critério qual—escolha uma boa!). O seu sistema precisa ser tal que ele possa ser compilado com qualquer uma das duas implementações de `ST.c` sem qualquer modificação. Organize seus arquivos em dois conjuntos (diretórios). A única diferença entre os dois conjuntos deve ser o arquivo `ST.c`.

(4) O seu programa deve ser "portável". Observe que no mínimo o seu programa deve compilar sem problemas com o `gcc`.

4. Análise de desempenho

Faça uma análise empírica de desempenho, comparando seus dois programas e o programa `wordtest`. Faça um pequeno relatório com os resultados e conclusões.

4.1. **Dados.** Procure implementar sua tabela de símbolos de forma que os seus dois programas possam manipular arquivos de tamanho 'razoavelmente grandes' como, por exemplo, o *King James' Bible* (`ftp://ibiblio.org/pub/docs/books/gutenberg/etext92/bible11.txt`), que contém algo como 800.000 palavras.

Uma boa fonte de arquivos para testar o seu programa é o Projeto Gutenberg (`http://promo.net/pg/`).

4.2. **Dicionários.** Você pode usar os dicionários do Unix para processar textos em inglês. Para o português, você pode usar o dicionário

(1) `portugues_br` (2.9M): uma lista de palavras do português brasileiro.

Esses e outros dois dicionários para inglês estão disponíveis em

`http://www.ime.usp.br/~cris/dicionarios/`

O arquivo `br` foi gerado a partir do trabalho de Ricardo Ueda (`http://www.ime.usp.br/~ueda`). Veja, em particular, a sua página sobre `br.ispell` (`http://www.ime.usp.br/~ueda/br.ispell`). Este trabalho é distribuído de acordo com a licença GNU GPL (veja a licença GNU GPL (`http://www.ime.usp.br/~ueda/br.ispell/gpl`)). Portanto note que uma condição que devemos respeitar é que qualquer coisa que produzirmos com a ajuda deste arquivo deve ficar livremente redistribuível. Para entender a motivação e filosofia dos termos da licença GNU GPL, veja a página do Projeto GNU (`http://www.gnu. ai.mit.edu/`).

5. Observações

(1) *Este EP é individual.*
(2) Preste atenção na modularização descrita neste enunciado.
(3) 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.
(4) *Não imponha restrições arbitrárias sobre a entrada.* Quando usado de forma inesperada, o seu programa deve parar de forma 'graciosa'.
(5) Organizem-se na turma para fazer bons testes. Comparem entre vocês o desempenho de seus programas.
(6) Seja criativo. Se seu programa fizer algo a mais do que foi pedido você poderá ganhar algum bônus na nota. Por outro lado, você *deve* respeitar as especificações dadas neste enunciado.
(7) Entregue o seu EP seguindo os moldes usuais, no panda. Inclua, se quiser, um `Makefile` e eventuais arquivos de teste.

Observação final. Enviem dúvidas para o fórum da disciplina. Eventualmente, ajustes no enunciado ocorrerão ao discutirmos este EP no fórum.