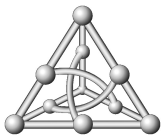

UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
Sistemas de Informação – CPAN

Índice baseado em Árvores Rubro-Negras

Trabalho 1

Estruturas de Dados e Programação — 2017

1 Descrição

No contexto de buscas por palavras, a indexação de termos (ou palavras) em textos é uma operação muito comum. Seja para buscas em páginas web ou arquivos em um disco, a indexação está entre as tarefas pré-processamento de dados necessárias para consultas eficientes.

Figura 1: Árvore Rubro-Negra?

Contudo, ao buscar uma palavra, o armazenamento desse índice em uma estrutura linear pode resultar em um longo tempo de processamento, principalmente se tratando de textos com um grande volume de palavras. Portanto, precisamos de uma estrutura de dados eficiente para tal, ou seja, *balanceada*.

Sua tarefa é escrever um programa que, dado um arquivo de texto qualquer, seja capaz de criar um índice de termos, armazenando-o em uma árvore rubro-negra. Aqui, cada nó representa uma palavra, que é sua **chave**. Além disso, precisaremos de **dados satélite** que armazenem em quais linhas a palavra aparece, mais especificamente uma **lista encadeada** de inteiros mantidos em **ordem crescente**, onde cada inteiro representa uma linha onde a palavra apareceu.

Queremos também, a fim de verificar as propriedades da árvore rubro-negra, imprimir a árvore por níveis. Obviamente precisaremos para isso de uma estrutura de dados auxiliar que você já conhece: a **fila**.

Ao final da execução do programa queremos visualizar a estrutura da árvore rubro-negra construída, conforme descrito acima, e o índice criado à partir do texto, em ordem alfabética. Esta última exigência pode ser satisfeita utilizando algum **percurso** na árvore.

Você deve implementar algum procedimento para, sistematicamente, remover todos os nós da árvore e seus dados satélite ao finalizar o programa. Porém, observe que, como isso pode ser feito com algum **percurso** na árvore, não é preciso implementar a operação de remoção da árvore rubro-negra.

2 Entrada e saída


Figura 2: Lista Encadeada?

O nome do arquivo de entrada é dado pela linha de comando como o primeiro parâmetro, por exemplo `./programa texto.txt`. Você deve inserir no índice todas palavras com 3 ou mais caracteres, juntamente com a linha onde cada palavra ocorre. Cada termo deve aparecer apenas uma vez no índice, por isso cada nó guarda uma lista de inteiros que representam as linhas.

A leitura do arquivo deverá desprezar espaços em branco e sinais de pontuação, que serão considerados separadores de palavras. Além disso, a leitura deverá converter todas as letras maiúsculas em minúsculas, armazenando todas palavras em letras minúsculas, portanto versões maiúsculas ou minúsculas de termos são consideradas idênticas. Você pode considerar que cada palavra contém no máximo 20 letras e que não haverá caracteres acentuados.

A saída referente à árvore deve ser escrita da seguinte maneira: para cada nível da árvore, começando pela raiz, escreva em uma linha cada nó do nível em questão, do mais à esquerda para o mais à direita na árvore, separados por espaço, no formato `E(PALAVRA [COR])D_` onde:

- `E` deve ser `/` caso haja filho esquerdo e `*` caso contrário
- `D` deve ser `\` caso haja filho direito e `*` caso contrário
- `PALAVRA` deve ser a palavra que o nó representa, em letras minúsculas
- `COR` é a cor do nó, `R` para vermelho e `N` para preto
- _ é um espaço

, por exemplo `/(teste [R])*`

A saída referente ao índice deve ser escrita da seguinte maneira: cada linha deve conter uma palavra do índice, seguida de um espaço e das linhas onde a palavra ocorre, separadas por vírgula. Observe que, caso um termo ocorra mais de uma vez na mesma linha, essa linha não deve constar mais de uma vez na lista ocorrências. As palavras com as respectivas ocorrências devem ser exibidas em ordem lexicográfica.

3 Exemplo de entrada

```
Um, dois, tres, testando.
Testando tres tres tres vezes.
Um teste final.
```

4 Exemplo de saída

```
/(testando [N])\
*(dois [N])\ /(tres [N])\
*(final [R])* *(teste [R])* *(vezes [R])*
dois 1
final 3
testando 1,2
teste 3
tres 1,2
vezes 2
```

A saída acima representa a seguinte árvore:

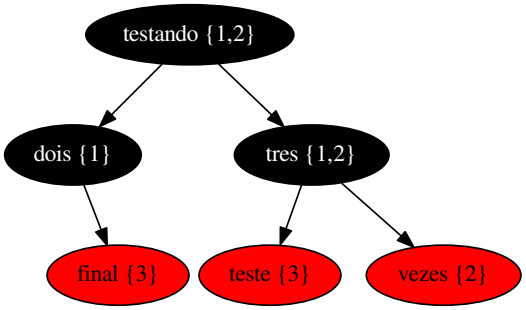

Figura 3: Os números entre chaves representam as linhas onde cada palavra ocorre.

5 Exigências

Você DEVE passar apenas uma vez pelo texto. À medida que as palavras são lidas do arquivo, o seu algoritmo deve pesquisar a árvore para ver se a palavra já está presente. Se estiver, adicione o novo número de linha à lista dessa palavra. Se não estiver presente, cria um novo nó na árvore e inicia a lista de linhas com a linha atual.

Você DEVE inserir elementos e balancear a árvore rubro-negra conforme visto em sala, de forma que, para cada conjunto de dados inseridos, só haverá uma topologia de árvore possível.

NÃO É PERMITIDO utilizar qualquer estrutura de dados já implementada em bibliotecas, portanto você deve implementar todas as estruturas de dados necessárias para seu programa.

Você DEVE, ao final do seu programa, liberar toda memória alocada dinamicamente (lista, fila, árvore, e outras) e também garantir que seu programa não realiza acessos inválidos à memória. Para verificar essa questão, será utilizado o utilitário *Valgrind*, com o comando `valgrind --leak-check=full --show-reachable=yes --track-fds=yes ./programa texto.txt`. Embora execute mais lentamente, você pode compilar seu programa com a opção de depuração `-g`, permitindo que o valgrind detalhe mais a saída, incluindo os números das linhas de seu programa com eventuais problemas.

Fique atento às especificações de estruturas de dados utilizadas e da saída.

6 Entrega

Instruções para entrega do seu trabalho:

1. **Cabeçalho**

   Seu trabalho deve ter um cabeçalho com o seguinte formato:

```
/*************************************************
 *
 * Nome do(a) estudante
 * Trabalho 1
 * Professor(a): Nome do(a) professor(a)
 *
 */
#include <stdio.h>
```

2. **Compilador**

   Os(as) professores(as) usam o compilador da linguagem C/C++ da coleção de compiladores GNU `gcc/g++`, com as opções de compilação `-Wall -pedantic -std=c99` para C e `-Wall -pedantic -std=c++03` para C++ ao corrigir os programas. Opcionalmente, você pode utilizar a flag `-g` para compilar seu programa Se você usar algum outro compilador para desenvolver seu programa, antes de entregá-lo verifique se o seu programa tem extensão `.c` / `.cpp`, compila sem mensagens de alerta e executa corretamente.

3. **Forma de entrega**

   A entrega será realizada diretamente no Sistema de Suporte a Disciplinas (SSD/FACOM), na disciplina de Estruturas de Dados e Programação. Um fórum de discussão deste trabalho já se encontra aberto. Para entrega do trabalho, você deve estar cadastrado na página `http://ead.facom.ufms.br` na disciplina e na turma que você está matriculado. Após abrir uma sessão digitando seu *login* e sua senha, vá até o tópico "Trabalhos", e escolha "T1 - Entrega". Você pode entregar o trabalho quantas vezes quiser até às **23 horas e 55 minutos** do dia **26 de junho de 2017**. A última versão entregue é aquela que será corrigida. Encerrado o prazo, não serão mais aceitos trabalhos.

4. **Atrasos**

   Trabalhos atrasados não serão aceitos. Não deixe para entregar seu trabalho na última hora. Para prevenir imprevistos como queda de energia, problemas com o sistema, falha de conexão com a internet, sugerimos que a entrega do trabalho seja feita pelo menos um dia antes do prazo determinado.

5. **Erros**

   Trabalhos com erros de compilação receberão nota ZERO. Faça todos os testes necessários para garantir que seu programa está livre de erros de compilação.

6. **O que entregar?**

   Você deve entregar um único arquivo contendo APENAS o seu programa fonte com o mesmo nome de seu login no moodle, como por exemplo, `fulano_silva.cpp`. NÃO entregue qualquer outro arquivo, tal como o programa executável, já compilado.

7. **Verificação dos dados de entrada**

   Não se preocupe com a verificação dos dados de entrada do seu programa. Seu programa não precisa fazer consistência dos dados de entrada. Isto significa que se, por exemplo, o seu programa pede um número entre 1 e 10 e o usuário digita um número negativo, uma letra, um cifrão, etc, o seu programa pode fazer qualquer coisa, como travar o computador ou encerrar a sua execução abruptamente com respostas erradas.

8. **Arquivo com o programa fonte**

   Seu arquivo contendo o programa fonte na linguagem C/C++ deve estar bem organizado. Um programa na linguagem C/C++ tem de ser muito bem compreendido por uma pessoa. Verifique se seu programa tem a indentação adequada, se não tem linhas muito longas, se tem variáveis com nomes significativos, entre outros. Não esqueça que um programa bem descrito e bem organizado é a chave de seu sucesso. Não esqueça da documentação de seu programa e de suas funções.

   Dê o nome do seu usuário do moodle para seu programa e adicione a extensão `.c` / `.cpp` a este arquivo. Por exemplo, `fulano_silva.c` / `fulano_silva.cpp` é um nome válido.

9. **Conduta Ética**

   O trabalho deve ser feito INDIVIDUALMENTE. Cada estudante tem responsabilidade sobre cópias de seu trabalho, mesmo que parciais. Não faça o trabalho em grupo e não compartilhe seu programa ou trechos de seu programa. Você pode consultar seus colegas para esclarecer dúvidas e discutir ideias sobre o trabalho, ao vivo ou no fórum de discussão da disciplina, mas NÃO copie o programa!

   Trabalhos considerados plagiados terão nota ZERO. Estudante que se envolver em DOIS CASOS DE PLÁGIO estará automaticamente REPROVADO na disciplina.