Universidade Federal de Minas Gerais **Algoritmos e Estruturas de Dados II**

Instituto de Ciências Exatas **2o Semestre de 2013**

Departamento de Ciência da Computação

Trabalho Prático 2 – Processamento de Texto – Estruturas de Dados
Valor: 20 pontos

Data de entrega no PRATICO (aeds.dcc.ufmg.br): 19/11/2013

O objetivo do presente trabalho é criar um processador de textos para análise da ocorrência de palavras no texto. Essa análise é útil em diversas aplicações tais como indexação de conteúdo, feito nos buscadores google e yahoo e outros. Dentre outas aplicações, também pode ser usado para compressão de dados, onde a análise de ocorrência de palavras é usada para avaliação da entropia do documento posterior aplicação do algoritmo de compressão como Huffman por exemplo.

Uma árvore de pesquisa é uma estrutura de dados eficiente para armazenamento e recuperação de informação. Neste trabalho a árvore de pesquisa será útil devido ao acesso direto e sequencial eficientes a uma estrutura de palavras do texto. Esta estrutura também apresenta baixo custo de inserção.

Objetivo Geral: dado um texto de entrada, o objetivo do Trabalho Prático 2 é a implementação de uma árvore de busca para armazenamento das palavras contidas num texto e sua respectiva ocorrência permitindo também busca de uma palavra e imprimindo ao final do processamento a lista de palavras em ordem alfabética juntamente com o número de sua ocorrência.

Objetivos específicos: construção de uma árvore binária de pesquisa com as seguintes funções: **Vazia**, **Insere**, **Busca, Remove** e **Imprime**.

```
int Vazia(TipoArvore arvore);
```

Verifica se a árvore está vazia, retornando `0`, ou se contém algum elemento, retornando `1`. A árvore está vazia quando o nó raiz é `NIL` como mostrado na Fig. 1.

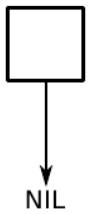

Figura 1: Exemplo de árvore vazia.

```
void Insere(TipoItem *x, Apontador *no)  ;
```

Faz a inserção de um novo nó na árvore binária de busca mantendo seus critérios. Ao inserir uma chave (palavra) que já existe na árvore, a operação deve apenas incrementar a ocorrência desta palavra.

Deverá ser impresso no arquivo de saída `insere <palavra>` caso a operação tenha sucesso na inserção e `incrementa <palavra>` se a palavra já existir, sendo feito um incremento na ocorrência.

Considere o exemplo de inserção do texto a seguir: *Implementação de algoritmos e estruturas de dados*.

Inserção da palavra "*implementação*"

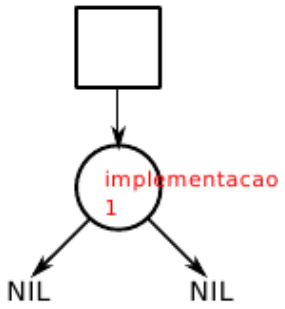

Figura 2: Inserção da palavra *implementação*.

Inserção da palavra "*de*"

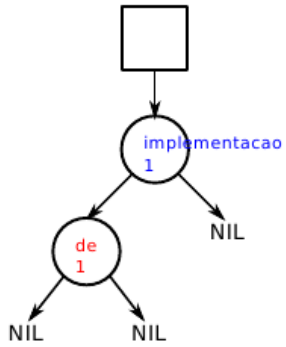

Figura 3: Inserção da palavra *de*.

Inserção da palavra “*algoritmos*”

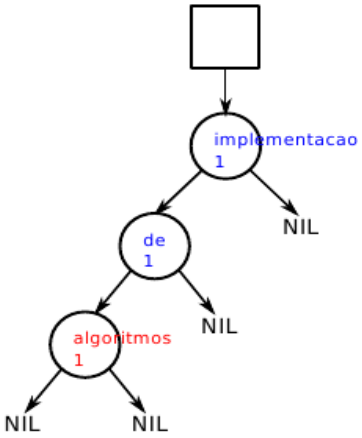

Figura 4: Inserção da palavra *algoritmos*.

Inserção da palavra “*e*”

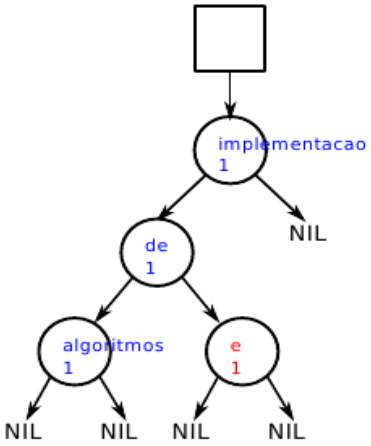

Figura 5: Inserção da palavra *e*.

Inserção da palavra “*estruturas*”

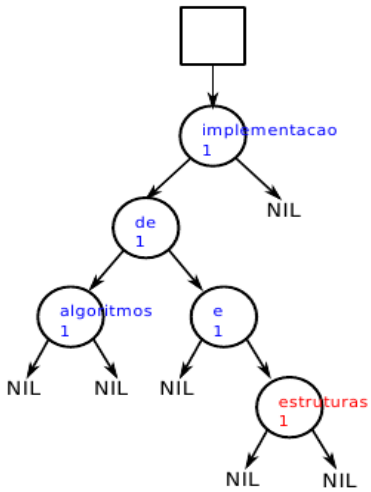

Figura 6: Inserção da palavra *estruturas*.

Inserção da palavra “*de*”. Note que essa palavra já existe na árvore, de modo que terá apenas o efeito de incremento na sua ocorrência. Ver fig. 7.

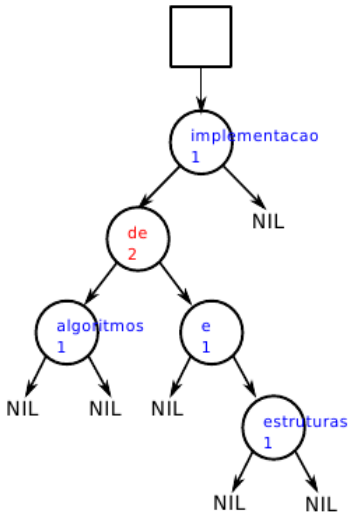

Figura 7: Inserção da palavra *de*.

Inserção da palavra “*dados*”

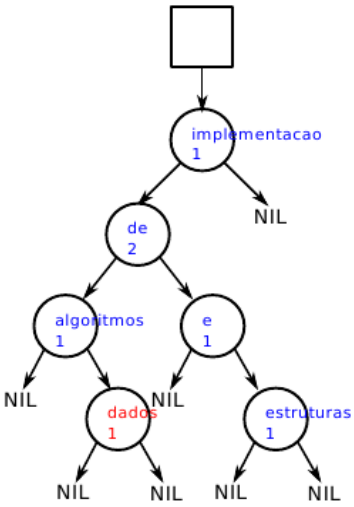

Figura 8: Inserção da palavra *dados*.

Nesta sequência de inserção, seria gerada a seguinte saída no arquivo de saída:

```
insere implementacao
insere de
insere algoritmos
insere e
insere estruturas
incrementa de
insere dados
```

```
int Busca(TipoChave c, Apontador *no) ;
```

Faz a busca de uma determinada palavra-chave retornando `0` caso não encontre e `1` caso contrário. Também deverá ser impresso no arquivo de saída `true` caso a palavra seja encontrada e `false` caso contrário.

Na busca, a função verifica se o `no` contém a palavra-chave, caso não tenha, verifica se a palavra-chave é menor que a chave do `no` corrente, se sim, a pesquisa continua pelo filho esquerdo, caso contrário, pelo filho direito e assim sucessivamente para cada `no` visitado, até que a chave seja encontrada ou uma folha diferente da palavra-chave seja alcançada.

As buscas serão realizadas com palavras encontradas no texto que comecem com o caracter especial `#`. Por exemplo, caso ocorra no texto em qualquer posição `#estruturas`, então a palavra estruturas não sera inserida e sim buscada na árvore.

Na busca é preciso imprimir no arquivo de saída o percurso, ou seja, a sequência de palavras dos nós visitados.

Por exemplo, na busca pela palavra `estruturas`, deverá ser impressa a seguinte sequência de busca com a respectiva ocorrência da palavra até o dado momento:

```
implementacao 1
de 2
e 1
estruturas 1
true
```

A fig. 9 ilustra a busca pela palavra `estruturas`.

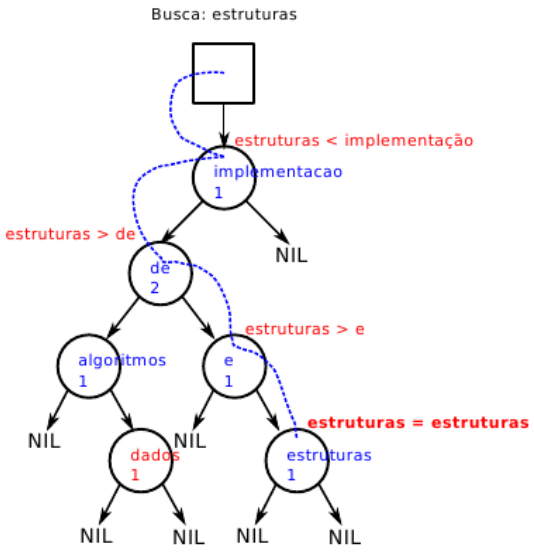

Figura 9: Busca pela palavra *estruturas*.

Caso a palavra buscada não seja encontrada, deverá ser impresso o caminho de busca até a folha visitada. Por exemplo, para a busca da palavra *abacaxi*, na árvore apresentada na fig. 9, deverá ser impressa a seguinte sequência:

```
implementação 1
de 2
algoritmos 1
false
```

```
void Remove(TipoChave c, Apontador *no);
```

Faz a remoção de uma palavra-chave c.

O procedimento de remoção será da seguinte forma: se o nó que contém a palavra-chave a ser retirado possui no máximo um filho, então a operação é simples. Caso o nó contenha dois filhos, nó a ser retirado deve ser substituído pelo registro mais à esquerda na sua subárvore direita.

A remoção deverá imprimir no arquivo de saída se a operação teve sucesso ou não, sendo `remove true <palavra-chave>` para sucesso e `remove false <palavra-chave>` caso contrário, seguindo da palavras-chave.

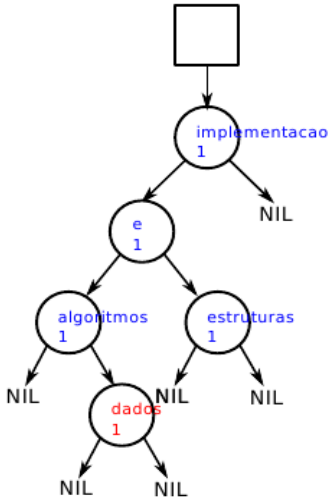

Figura 10: Remoção da palavra *de*.

```
void Imprime(Apontador *no);
```

Imprime todas as palavras da árvore em ordem alfabética e sua ocorrência no texto. A impressão ocorrerá sempre que for encontrado o símbolo `&` em qualquer parte do texto. A ocorrência do símbolo de impressão, implicará na impressão da árvore em ordem alfabética com a respectiva ocorrência de cada palavra, tal como a árvore se encontra no momento da impressão.

Dica: Para fazer a impressão da árvore binária de busca em ordem alfabética, use o caminhamento central.

Para o exemplo da árvore mostrada anteriormente construída com o texto: *Implementação de*

algoritmos e estruturas de dados (Fig 8). Teríamos a seguinte impressão na saída:

```
algoritmos 1
dados 1
de 2
e 1
estruturas 1
implementação 1
```

Tratamento do texto.

Para extrair as palavras de um texto, você pode utilizar o procedimento `ExtraiPalavra` mostrado na página 205 (2ª Ed.) do livro "Projeto de Algoritmos", Nívio Ziviani.

Cabe ressaltar que:

a) Uma palavra é considerada como uma sequência de letras e dígitos, começando com uma letra. Portanto, ignore sinais de pontuação. Você pode assumir que os textos não terão acentuação.

b) Palavras com letras em maiúsculas devem ser primeiramente transformadas para minúsculas antes da inserção na árvore de busca. Desta forma, a mesma palavra apresentada com letras minúsculas ou maiúsculas não serão diferenciadas.

c) Uma palavra pode ocorrer múltiplas vezes na mesma linha de um documento, ou mesmo em múltiplas linhas de um mesmo documento.

Modelo de Entrada

O programa principal deverá ler, da entrada padrão, o arquivo de entrada. Quando executado na linha de comando, usar:

```
./programa < entrada.txt
```

A saída deverá ser direcionada para o dispositivo de saída padrão. (`stdout`).

Os arquivos de entrada serão arquivos de texto contendo caracteres especiais para dar comandos de busca (#), remoção (@) e impressão (&).

Um exemplo de arquivo de entrada é mostrado a seguir:

```
Entrada para o trabalho pratico dois. #trabalho & @o @para @facil &
```

Modelo de saída

Cada saída deverá ser escrita em uma nova linha. Em outras palavras, não é admitido mais que uma saída por linha. As possíveis saídas são listadas a seguir:

`insere <palavra>`	Para inserção de uma palavra nova.

`incrementa <palavra>`	Para o incremento da ocorrência de uma palavra já existente.
`remove true <palavra>`	Para operação de remoção executada com sucesso.
`remove false <palavra>`	Para operação de remoção executada com algum erro ou chave não encontrada.
`palavra x`	Palavra e sua ocorrência x.

Use EXATAMENTE a saída conforme especificado para manter a conformidade com o PRATICO. A saída referente a entrada mostrada nesse texto é apresentada a seguir:

```
insere entrada
insere para
insere o
insere trabalho
insere pratico
insere dois
entrada 1
para 1
trabalho 1
true
dois 1
entrada 1
o 1
para 1
pratico 1
trabalho 1
remove true o
remove true para
remove false facil
dois 1
entrada 1
pratico 1
trabalho 1
```

Note que o programa principal não poderá acessar diretamente a estrutura interna do TAD. Se necessário, acrescente novas funções ao seu TAD detalhando-as na documentação do trabalho.
O programa criado não deve conter "menus interativos" ou "paradas para entrada de comandos" (como o system ("PAUSE") por exemplo). Ele deve apenas ler os arquivos de entrada, processá-los e gerar os arquivos de saída.

Os TPs serão corrigidos em um ambiente Linux. Portanto, o uso de bibliotecas do Windows está PROIBIDO.

O que deve ser entregue:

- Código fonte do programa em C (todos os arquivos .c e .h), bem identado e comentado.
- Arquivo executável.
- Documentação do trabalho. Entre outras coisas, a documentação deve conter, sucintamente:

1. **Introdução:** descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
2. **Implementação:** descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, compilador utilizado, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
3. **Estudo de Complexidade:** estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O), considerando conjuntos de tamanho *n*.
4. **Testes:** descrição dos testes realizados e listagem da saída (não edite os resultados).
5. **Conclusão:** comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
6. **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso

Obs: Apesar desse trabalho ser relativamente simples, a documentação pedida segue o formato da documentação que deverá ser entregue nos próximos trabalhos. Um exemplo de documentação está disponível no Moodle.
Comentários Gerais:

1. Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.
2. Clareza, indentação e comentários no programa também serão avaliados.
3. O trabalho é individual.
4. A submissão será feita pelo sistema online (`http://aeds.dcc.ufmg.br`).
5. Trabalhos copiados, comprados, doados, etc. serão penalizados conforme anunciado.
6. Na prova 3, um dos exercícios poderá ser sobre a implementação do trabalho. A nota do trabalho poderá ser ponderada pela nota desse exercício.
7. Penalização por atraso: $(2^d - 1)$ pontos, onde d é o número de dias de atraso.