UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INF01203 Estruturas de Dados - Turma C

Engenharia de Computação

CONTADOR DE FREQUÊNCIAS DE PALAVRAS

Giordano Souza de Paula 308054

Sabryna Trindade 302133

Professora: Renata Galante

Porto Alegre, 10 de dezembro de 2019

SUMÁRIO

1. INTRODUÇÃO

O objetivo do dado trabalho é a criação um algoritmo utilizando a linguagem de programação C, que a partir de uma linha de comando, deve receber três arquivos de entrada, e a partir desses contabilizar a frequência em que palavras aparecem em um dos arquivos que deve conter um texto. Essa contabilização deve ser feita de forma a considerar definidas operações solicitadas no arquivo texto de operações.

O desenvolvimento do trabalho visa desenvolver os conhecimentos obtidos ao longo do semestre na cadeira de Estrutura de Dados - lecionada pela professora Renata Galante - como listas, pilhas, filas e diferentes tipos de árvores binárias, e com isso constatar através de um experimento prático, os diferentes desempenhos obtidos (como tempo de execução, e instruções necessárias) utilizando diferentes tipos de estruturas de dados para realizar uma mesma operação.

2. FUNÇÕES

Rotinas que foram criadas para operar, simplificar e organizar melhor o código do trabalho realizado como um todo, explicadas com detalhes abaixo:

2.1. imprime_apresentacao()

Imprime na tela as informações sobre o trabalho. Incluindo nome da professora, dos alunos, número de matrícula da dupla e uma ligeira explicação do que o trabalho faz.

2.2. imprime_encerramento()

Informa que está fechando os arquivos e encerrando o programa.

2.3. abre_arquivo(FILE **, char *, char *)

Abre um arquivo texto, o qual foi enviado o nome (para que possa abrir mais de um arquivo com a mesma função). Retorna 1 se o arquivo estiver vazio.

2.4. contador (FILE*, avlNode *, int, int)

Função exigida na definição do trabalho. Percorre a árvore, salvando no arquivo de saída as palavras que possuírem K1 >= frequência <= K2.

2.5. compara_palavras(char, char)

É uma função auxiliar criada para que se torne mais fácil ordenar a árvore, mais especificamente em ordem alfabética. A compara_palavras recebe duas palavras e as compara.

Retorna:

- 0 caso a primeira palavra venha antes da segunda na ordem alfabética;
- 1 caso a segunda palavra venha antes da primeira na ordem alfabética;
- 2 caso as duas palavras sejam iguais, ou seja, a palavra foi inserida na árvore anteriormente. É usada para contar a frequência;

- 404 caso uma ou ambas as palavras esteja vazia. É usada para tratamento de erros;

2.6. Funções para rotação da Árvore AVL:

Recebem o nome da rotação que executam, são chamadas pelas funções Caso1 e/ou Caso2.

rotacao_direita, rotacao_esquerda, rotacao_dupla_esquerda, rotacao_dupla_direita

2.7. Caso1 e Caso2

Funções auxiliares, são chamadas pela função insertAVL e definem qual o tipo de rotação a ser executada.

2.8. checa_altura_AVL(avlNode *)

Verifica a altura de certo nó da Árvore, este que é recebido por referência.

2.9. checa_AVL(avlNode *)

Essa função confere se a árvore está devidamente balanceada, seguindo as definições de Árvore AVL.

2.10. insertAVL(avlNode *, char *, int *, int *, int *, int *)

Insere um nó na Árvore AVL, usando a compara_palavras para ordenar e chamando as funções Caso1 e Caso2 se for necessário efetuar algum tipo de rotação.

2.11. calcula_FB(avlNode *)

Calcula o Fator de Balanceamento da Árvore AVL ao comparar duas sub-árvores.

2.12. move_cursor(int, int)

Move o cursor para posição desejada, utilizada para a organização da interface.

2.13. apaga_caracteres(int, int, int)

Apaga uma definida quantidade de caracteres a partir de definida posição do display, utilizada para a organização da interface.

2.14. intro_arquivo_saida_avl(FILE **, avlNode*, int, int, clock_t, int)

Preenche o arquivo de saída com todos os dados que foram obtidos ao decorrer do programa.

2.15. checa_linhas(FILE **)

Retorna a quantidade de linhas que um arquivo possui. O arquivo é recebido por referência para que a função possa ser usada para mais de um arquivo.

2.16. checa_operacoes(FILE *, char *, char *)

Lê o arquivo de operações, conferindo quantas vezes foram chamadas as funções F e C.

2.17. error_texto()

É executada se houve erro ao abrir o arquivo com o texto a ser lido, informando ao usuário.

2.18. error_opera()

É executada se houve erro ao abrir o arquivo com as operações a serem executadas, informando ao usuário.

2.19. error_saida()

É executada se houve erro ao criar o arquivo de saída, informando ao usuário.

2.20. Desenha(avlNode *, int)

Imprime um rascunho da árvore AVL.

2.21. get_operacao (FILE **, int, int, int, char, char, int *, int *)

Lê o arquivo de operações, inserindo na matriz somente a parte útil de cada linha.

2.22. le_texto_avl (FILE **, avlNode **, int, int *, int *, int *, int *)

Lê o arquivo que contém o texto a ser analisado, tratando a pontuação e chamando a insertAVL para inserir a palavra lida na árvore.

2.23. frequencia(char*, avlNode *)

Função exigida na definição do trabalho. Percorre a árvore, procurando as palavras chamadas pela operação F e checando suas respectivas frequências.

2.24. fim_arquivo_saida_AVL(FILE **, avlNode *, int, int, clock_t, clock_t *, int, int, int,int, int, int, char, char)

Faz o tratamento de uma série de variáveis a serem salvas no arquivo de saída, incluindo o tempo.

2.25. ContaNodos(avlNode *)

Conta a quantidade de nodos contidos na árvore.

2.26. InsereRN(RBNode*, char *, int *, int *)

Insere nodos na árvore Rubro-Negra, chamando a VerificaRN para tratar o balanceamento.

2.27. VerificaRN(RBNode*,char, int *)

Verifica se é necessária a execução de rotações, conforme as regras da Árvore Rubro-Negra. Chama as funções RotacaoSimplesDir ou RotacaoSimplesEsq conforme o resultado.

2.28 Funções para rotação da Árvore AVL:

Recebem o nome da rotação que executam, são chamadas pela função VerificaRN.

RotacaoSimplesDir, RotacaoSimplesEsq

3 CONCLUSÃO

Operações simples, como comparativos IF, geralmente tinham seu tempo de execução desconsiderados nos trabalhos anteriores por serem algoritmos simples, mas quando essas operações são utilizadas em programas mais complexos (como entre diversas funções recursivas) seu tempo de execução deve ser considerado pois ele torna-se significativo para o tempo de execução total do programa.

Neste trabalho, por lidar com textos com grande números de palavras, teve o processo de simplificação do algoritmo (remoção de operações desnecessárias e variáveis não utilizadas) priorizado para melhorar o desempenho do programa como um todo.

REFERÊNCIAS BIBLIOGRÁFICAS

ALGORITMOS e Estruturas de Dados/Árvores AVL. [*S. l.*], 24 jan. 2011. Disponível em: https://pt.wikibooks.org/wiki/Algoritmos_e_Estruturas_de_Dados/%C3%81rvores_AVL. Acesso em: 5 dez. 2019.

ÁRVORE AVL. [*S. l.*], 30 set. 2019. Disponível em: https://pt.wikipedia.org/wiki/%C3%81rvore_AVL. Acesso em: 5 dez. 2019.

BSTS rubro-negras. [*S. l.*], 17 fev. 2018. Disponível em: https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/st-redblack.html. Acesso em: 4 dez. 2019.

ÁRVORE rubro-negra. [*S. l.*], 20 out. 2019. Disponível em: https://pt.wikipedia.org/wiki/%C3%81rvore_rubro-negra. Acesso em: 4 dez. 2019.