Trabalho Prático 03

Dicionário

Rafael Francisco Silva Granja

2021019629

Introdução:

O objetivo deste trabalho era a implementação de um TAD Dicionário utilizando duas estruturas estudadas, a arvore balanceada AVL e o Hash, gostaria de já deixar claro desde o início que este trabalho não possui a implementação com o Hash devido a complicações na estrutura da AVL e outras tarefas além.

A solução empregada foi a implementação da árvore balanceada AVL para armazenar cada um dos verbetes e uma estrutura simples de vetor para guardar os significados de cada verbete do dicionário.

Método:

A TAD foi construída utilizando classes para facilitar algumas manipulações e acessos a métodos, porém os métodos principais foram construídos fora das classes para facilitar o uso da recursividade em alguns casos e também pelo fato de ser desnecessário encapsular uma quantidade grande de métodos em uma ou duas classes.

Nesta árvore balanceada foi utilizado o fator de balanceamento padrão, janela de –1 até 1, além disso em todos os momentos foi importante garantir a liberação da memória alocada dinamicamente, assim como a alocação sequencial e correta na criação de novos itens.

O código realiza a leitura de um arquivo TXT para popular a árvore, dicionário, e como proposto pelo enunciado, faz a impressão de todos os verbetes ordenados alfabeticamente, junto com seus significados ordenados pela ordem de entrada, em seguida retira da arvore todos os itens que possuem ao mínimo um significado e após essa exclusão massiva, imprime árvore novamente,

adicionando ao arquivo de saída, todos os verbetes que não possuem nenhum significado relacionado a ele.

Análise de Complexidade:

Em relação a complexidade do TAD, todas as operações individuais possuem complexidade O(log(n)), sendo n o número de verbetes, tanto para o caso médio quanto para o pior caso, acrescento a operação de inserção dos significados no verbete que possui complexidade O(1) para inserir a primeira vez no vetor e O(n) para inserções posteriores no mesmo vetor, sendo n o número de significados do verbete;

Estratégia de Robustez:

O uso de conceitos de Programação Orientada a Objetos, reutilizando funções e adaptando classes para todos os métodos de ordenação garantem a robustez do código para qualquer tipo de manutenção, correção, adição ou remoção no código, da mesma forma que garantem segurança no acesso das informações de cada classe somente pelas classes permitidas.

Além disso, foram implementadas funções para realizar toda a liberação dos espaços de memória da árvore de forma recursiva, tendo certeza da destruição de todos os itens e seus atributos filhos, subsequentes que poderiam ocupar espaço na memória.

Análise Experimental:

Para realizar a análise experimental deste TAD, foram utilizadas duas variáveis para marcação de tempo de execução do programa e foram registrados os seguintes resultados:

Arquivo 100 linhas: TEMPO DE EXECUÇÃO: 0.055881

Arquivo 100 linhas: TEMPO DE EXECUÇÃO: 0.00742

Arquivo 522 linhas: TEMPO DE EXECUÇÃO: 0.068906

Arquivo 2589 linhas: TEMPO DE EXECUÇÃO: 0.347449

Arquivo 4997 linhas: TEMPO DE EXECUÇÃO: 1.0723

*No caso com 4997 linhas, houveram casos de segmentation fault, o que foi possível achar foi corrigido, porém observei que em alguns momentos havia o erro, mas ao rodar o mesmo código e mesmo arquivo, não havia erros de segmentação.

Conclusão:

A partir deste trabalho foi possível perceber o quão complexa e útil pode ser uma árvore de pesquisa e quão eficaz ela pode ser quando balanceada, infelizmente não consegui implementar o hash, talvez fosse até mais simplificado que a árvore, porém preferi focar esforços em entregar uma parte, porém a parte na minha opinião mais complexa e completa do trabalho.

Pensando em análise de algoritmo, esse foi o TP que requisitou maior análise de casos separados, debugação, além de raciocínio lógico para definir as operações a serem realizadas sem afetar a manipulação dos elementos.

Instruções para compilação:

Para utilizar o makefile, existem 2 comandos:

- make clean : limpas os arquivos na pasta bin e obj;
- make all: compila o código main.cpp, gera o objeto main.o na pasta obj, faz a junção do executável tp3 na pasta bin ao main.o na pasta obj;

Após isso é possível rodas comandos como por exemplo:

“bin/tp3 -i entrada.txt -o saida.txt -t arv”, memso exemcplo passado no enunciado do Trabalho