UNIVERSIDADE FEDERAL DE OURO PRETO
INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTAÇÃO E SISTEMAS
CAMPUS JOÃO MONLEVADE

ALEXANDRE MORAIS
GABRIEL DE MELO
LEANDRO GERALDO LINHARES COTA
MATHEUS DE CASTRO
MILENA ESTHER DE SÁ
NICOLE PEDROSA
THIAGO DIAS MAGALHÃES

RELATÓRIO – TRABALHO PRÁTICO PARTE - 1

ALGORITMO E ESTRUTURA DE DADOS I

Estudo de caso apresentado à Universidade Federal de Ouro Preto, como requisito parcial da Disciplina Algoritmo e Estrutura de Dados I, do Curso Sistemas de Informação, 2º Período.

Docente: Prof. Erik de Brito e Silva

JOÃO MONLEVADE

2018

1 Introdução

A proposta do trabalho foi confeccionar um programa em linguagem C que complementasse as funções do Trabalho prático 1, que gerenciava os dados dos países (lidos de um arquivo .txt) em um Lista simples. Para o trabalho 2, além das funções mencionadas anteriormente, foi pedido que os mesmos dados fossem armazenados e gerenciados em uma árvore de busca balanceada (AVL), e em uma HEAP máxima, executando buscas e remoções através da chave “idioma” (char[12]), tarefa que apesar de trabalhosa, foi executada corretamente por nosso grupo, como vamos detalhar a seguir.

2- Solução Proposta

Main – A função principal deste código foi propositalmente implementada de forma a deixar a responsabilidade das execuções do que se pede às sub-rotinas, deixando a main responsável apenas por exibir os menus e de acordo com a entrada do usuário alternar entre as funções através de alguns “switch-case”, além, é claro, de declarar as variáveis necessárias.

lista2AVL – Função alimentadora da arvore AVL, como seu nome pressupõe, passa os dados da lista para a arvore AVL. Um loop percorre a lista da primeira posição, até a ultima, com o auxilio de um ponteiro *p igualado inicialmente à cabeça da lista, e um laço while que executa ate que esse ponteiro seja igual a NULL, a cada interação, o p recebe p->prox.

struct No_avl- Tipo de dado do elemento da arvore AVL, gostaríamos de chama-lo apenas de “No”, mas já havíamos usado este nome para o nó da lista.Possui 4 variáveis, uma do tipo “Pais” chamada info, um inteiro que guarda a altura, e duas variáveis do próprio tipo “No_avl” “*esq” e “*dir”

cria_ArvAVL– Função básica da árvore AVL, usa um malloc para aloca a memória necessária para a raiz da árvore, faz uma conferencia se o espaço realmente pode ser alocado e retorna o ponteiro para a raiz.

libera_NO – Função recursiva que libera um nó da árvore, além de ser chamada por ela própria (recursão), também é chamada pela função que libera a árvore.

libera_ArvAVL– Faz uma pequena conferencia, para o caso da arvore já ser nula ao chegar à função, se não for, chama a função libera_NO passando a raiz, por fim dá “free” passando a raiz.

altura_NO- Função que não é exibida ao usuário, apenas é chamada pelas outras funções da árvore, que tem como objetivo retornar em que nível da árvore está o nó passado, tarefa fácil, pois “altura” é um dos elementos do “No-avl”.

consulta_ArvAVL–Função útil para retornar a busca do usuário, tem como parâmetros a raiz da àrvore, e a string referente ao idioma que se deseja buscar. Cria-se uma variável auxiliar de nome “atual” do tipo “struct No_avl”, com o auxilio dessa, percorremos toda a árvore em busca de um valor que de atual->info.idioma que seja igual à string passada por parametro

imprime- Função bem simples, recebe um Pais por parâmetro e imprime os seus elementos.

fatorBalanceamento_NO– Mais uma função específica da árvore, que retorna o resultado da função labs passando por parâmetro a subtração do nó a esquerda e à direita do nó passado por parâmetro.

maior– Simples função que retorna o maior entre dois inteiros x e y.

estaVazia_ArvAVL- Função usada pelas outras funções da árvore, retorna se a arvore cuja raíz é passada por parâmetro está vazia ou não.

totalNO_ArvAVL– Função retorna a quantidade de nós contidos na árvore, através da contagens de todos os nós à esquerda mais os nós à direita da raiz, mais 1 (a própria raiz), utiliza recursão.

emOrdem_ArvAVL– Função que tem como objetivo fazer a impressão em ordem da árvore, essa função é chamada na main, quando o usuário pede para que seja exibida toda a árvore AVL.

RotacaoLL– Faz a rotação esquerda-esquerda na árvore.

RotacaoRR– Faz a rotação direita-direita na árvore.

RotacaoLR– Faz a rotação esquerda-direita na árvore.

RotacaoRR– Faz a rotação direita-esquerda na árvore.

Insere_ArvAVL– Função que recebe como parâmetros a raiz da árvore e o Pais a ser inserido, e tem como objetivo inserir na arvore o país passado, obviamente, são feitas muitas comparações e recursões para certificar que o elemento está sendo inserido no lugar correto de acordo com a chave de ordenação (IDIOMA), além disso, caso a inserção desse novo elemento desbalanceie a árvore, é chamada a rotação correta para o desbalanceamento ocorrido.

getcomp– função retorna o numero de comparações que foram contadas na inserção anterior, e zera a variável de comparações usada, para que ela esteja pronta para outra contagem, na próxima inserção.

remove_ArvAVL–Função recebe dois parâmetros, a raiz da árvore e o valor que se deseja remover da árvore, no caso um idioma a ser removido, a arvore vai ser percorrida em busca do idioma passado, e quando encontrado, removeremos este, da mesma forma que a inserção, a remoção também pode desbalancear a árvore, e para cada caso de desbalanceamento esta função chama a função que sana este problema.

lista2HEAP–Similar à função “lista2avl”, ela percorre a lista e para cada elemento da lista, cria um elemento na HEAP.

Struct HEAP–Tipo de dado criado para a HEAP, tem 3 variáveis, uma do tipo Pais “*A”, duas variáveis do tipo inteiro: “tamanhoAtual” e “tamanhoMaximo”, a estrutura não tem um nome inicial e através do typedef foi batizada de HEAP.

inicializarHeap–Função inicializadora da heap, recebe como parâmetro, a heap, e o tamanho máximo suportado por ela, e aloca o espaço para as variáveis, depois disso, armazena em h-> tamanhoAtual o valor 0, e em h-> tamanhoMaximo, o valor passado por parâmetro .

destruirHeap–Função chamada ao fim do programa para desalocar o espaço alocado para a heap.

deleta–Função chamada quando se deseja excluir um elemento, como foi pedido nesse trabalho a exclusão com impressão do que foi excluído, incluímos alguns printf’s para atender à esse pedido, a função recebe como parâmetro o ponteiro de referência para a heap e a posição a ser excluída.

heapify_down–Função específica da heap, é usada para o ajuste da heap após uma exclusão, o que evita que a heap fique com “buracos” depois que elementos foram exluídos.

pai–Função específica da heap, recebe um i parâmetro e logo na primeira linha da função retorna este i/2.

filhoEsquerda– Função específica da heap, recebe um i parâmetro e logo na primeira linha da função retorna este i*2.

filhoDireita– Função específica da heap, recebe um i parâmetro e logo na primeira linha da função retorna este i*2+1.

maxHeapify–Função específica da heap, é usada para construir o heap máximo, caso essa heap passada já não o seja.

construirHeapMaximo–Função específica da heap, que trabalha junto com o maxHeapify na missão de transformar a heap em uma heap máxima, nesta função toda a metade inferior da heap é submetida à um loop, e para cada elemento dessa primeira metade da heap, é chamada a função anteriormente comentada, a maxHeapify.

imprimirArranjo–Função utilizada quando o usuário quer que seja exibida toda a heap, simplesmente percorre todos os elementos e os imprime um em cada linha na tela, com o auxilio de um “for” que vai de i=0 ate i<tamanhoMaximo.

imprimirIdioma–Parecida com a função anterior, com a diferença de percorrer a heap e imprimir com a condição de que só será impresso ta tela o país que tiver sua variável idioma igual à string passada por parâmetro.

heapSort –Mais uma função para impressão na tela do que se deseja, como no caso deste trabalho, o idioma é o parâmetro usado para pesquisa.

inserirHeap –Função utilizada para inserir um novo elemento (struct pais) na heap, a cada comparação incrementa a variável “comparacoes” para que seja exibido para o usuário após a inserção.

3 – Analise de Complexidade

Utilizei a abordagem do pior caso (big O) para analizar a complexidade desse programa .

Função	Complexidade
Main	O(1)
Lista2avl	O(N)
cria-ArvAvl	O(1)
Libera-No	O(1) .
altura_NO	O(1).
consulta_ArvAVL	O(N).
imprime	O(1).
fatorBalanceamento_NO	O(1)
maior	O(1)
estaVazia_ArvAVL	O(1)
totalNO_ArvAVL	O(logN)
emOrdem_ArvAVL	O(N)
RotacaoLL	O(1)
RotacaoLR	O(1)
RotacaoRL	O(1)
RotacaoRR	O(1)
Insere_ArvAVL	O(1)
getcomp	O(1)
remove_ArvAVL	O(1)
lista2HEAP	O(N)
inicializarHeap	O(1)
destruirHeap	O(1)
deleta	O(1)
heapify_down	O(logN)
pai	O(1)
filhoEsquerda	O(1)
filhoDireita	O(1)
maxHeapify	O(logN)
construirHeapMaximo	O(N)
imprimirArranjo	O(N)
imprimirIdioma	O(N)
heapSort	O(NlogN)
inserirHeap	O(N)

Como o algoritmo tem menus que vem e vão, o tempo e complexidade do programa depende muito do usuário, de quantos clientes são cadastrados e quantas operações sao efetuadas.

4 – Maquina Utilizada

A aplicação foi inteiramente escrita e compilada em um notebook HP 1000, com processador i3 (2,2GHz), 2GB de memoria primária , 500GB de memória secundaria. Sistema operacional windows 10 – 64bits.

5 - Conclusão

Nossa maior dificuldade foi manipular as árvores pela chave que nos foi passada, por ser um valor do tipo char[12] as comparações de valores não podiam acontecer com simples “==” “<” “>”, etc, tivemos que aprender a comparar strings, e mais importante, comparar no lugar certo para que a função das árvores acontecesse da forma correta. Mas com um mapa do programa feito à mão pudemos entender melhor, e conseguimos (orgulhosamente) realizar tudo que foi pedido no trabalho.

6 - Referências

- https://www.scriptbrasil.com.br/forum/topic/117251-duvida-como-usar-a-fun%C3%A7%C3%A3o-strcpy/

- Slides disponibilizados pelo professor

- www.cprogressivo.net