UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE QUIXADÁ

ESTRUTURA DE DADOS

Relatório sobre o Trabalho de Árvore de Ordenação

PEDRO **BOTELHO**

GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO - S2

ORIENTADOR: ATÍLIO GOMES LUIZ

QUIXADÁ - CE
2019

1 Introdução

O trabalho proposto traz como problemática a idéia de ordenar um vetor de inteiros com o uso de uma árvore. Essa árvore precisa estar cheia, ou seja, com todas suas folhas preenchidas, e ter sua altura $h = [log_2 n] + 1$.

A árvore é criada de baixo pra cima, e o processo foi automatizado para suprir as necessidades do usuário de ordenar os elementos. Em resumo, os menores elementos são mandados à raiz, e depois subistituídos pelos ϵ.

Desta forma, os dados inseridos por meio de arquivos de texto, são processados pelo programa e devolvidos ao usuário em um outro arquivo de texto.

2 Listagem dos Programas em C++

Foram criados três divisões do projeto:

2.1 ArvoreOrdenacao.h

Contém o protótipo das funções programados no outro arquivo *"ArvoreOrdenacao.cpp"*. Esse arquivo serve para "ligar" o arquivo anteriormente citado a função principal, para que suas funções sejam acessíveis lá.

2.2 ArvoreOrdenacao.cpp

Conta com todas as funções do protótipo já programadas (com excessão da *main*) para serem usadas para gerenciar a árvore.

2.3 principal.cpp

Conta com a função *main*, a função que vai operar as outras funções a fim de criar a árvore e ordenar seus nós. Também será responsável por receber informações do arquivo e escreve-los em outro arquivo, facilitando a comunicação com o usuário.

3 Listagem dos Testes Executados

Foram executados os testes propostos no PDF do trabalho, e ainda foram executados alguns outros testes que achei que seriam importantes, como por exemplo números repetidos, para mostrar como eles não eram repetidos também na saída. Segue abaixo o arquivo "vetores.txt" e "ordenados.txt", as entradas e saídas:

vetores.txt:

10

23 3 45 6 1 9 37 99 0 30

3

345 2 1

5

98 34 2 1 76

8

1 2 3 6 23 9 37 99

4

20 17 5 0

5

2 4 26 0 7

3

1 1 2

4

1 1 2 2

0

ordenados.txt:

0 1 3 6 9 23 30 37 45 99

1 2 345

1 2 34 76 98

1 2 3 6 9 23 37 99

0 5 17 20

0 2 4 7 26

1 2

1 2

Bom lembrar que na entrada, a primeira linha é o tamanho do vetor, e a segunda é o vetor em si. Na saída apenas é retornado o vetor.

4 Estruturas de Dados Usadas

Todas essas estruturas foram implementadas em "ArvoreOrdenacao.cpp":

struct NoArvore: representa a estrutura do nó da árvore.

NoArvore* criarArvore: essa função cria uma árvore com seus nós e retorna o nó raiz. Essa funçaõ cria um vetor para receber os inteiros digitados pelo usuário, com tamanho 2^{h-1}, onde h é a altura da árvore. Logo, se sobrar espaços, serão preenchidos com epsilons, um número ligeiramente maior que o maior número dentre os digitados pelo usuário. Com o auxílio de uma fila cria as folhas, e após isso cria o restante da árvore até chegar à raiz.

NoArvore* percorreArvore: essa função percorre a árvore até achar as folhas, para então substituir os menores valores por ϵ, e atribuir ao nó pai de dois filhos, o menor valor entre eles.

NoArvore* liberaArvore: desaloca espaço alocado na memória pela árvore, e apaga o nó raiz ao final.

int valorNo: retorna o valor numérico - chave - um nó qualquer.

int alturaArvore: calcula a altura da árvore cheia por meio da fórmula $log_2 n + 1$, com n igual ao tamanho passado pelo usuário.

int maiorElemento: retorna o maior elemento de um vetor de inteiros.

int eMin: simplesmente retorna o menor valor da árvore, que está na raiz.

int main: a função *main* irá receber os dados informados pelo cliente no arquivo, se utilizar de funções externas e internas para criar e ordenar a arvore, e mostrar em uma arquivo os valores ordenados (de sua raiz). Em resumo, o programa verifica se o arquivo está aberto, pega o tamanho do vetor e o vetor no arquivo, faz as devidas modificações, cria a árvore a partir desse vetor, percorre a árvore, ordenando os elementos e escrevendo no arquivo o valor da raiz, e depois libera a árvore. No caso, isso se repetirá pra toda entrada até o programa encontrar apenas um 0.

5 Tempo de Execução das Funções

:

NoArvore* criarArvore: essa função tem complexidade $O(n)$, é iterativa, já que tem vários loops finitos dentro, três, precisamente. E fora as criações e atribuições de variavel, com tempo de pior caso $f(n) = 3n$, $O(n)$ é sua complexidade.

NoArvore* percorreArvore: essa função tem complexidade $O(n)$, já que vai percorrer a árvore algumas vezes, porém nunca aninhados. No pior caso ele vai percorrer a árvore inteira três vezes, ou seja, $f(n) = 3n$, logo, $O(n)$. As atribuições do final da função também atrasam um pouco, mas a ordem de $O(n)$ permanece.

NoArvore* liberaArvore: essa função tem complexidade $O(n)$, pois percorre a árvore no máximo duas vezes para libera-la, e depois deleta a raiz. Logo, $f(n) = 2n$, $O(n)$.

int valorNo: essa função tem complexidade $O(1)$, pois apenas retorna o valor numérico de um nó.

int alturaArvore: essa função tem complexidade $O(1)$, pois faz um cálculo matemático simples e retorna o seu resultado.

int maiorElemento: essa função tem complexidade $O(n)$, pois verifica um vetor de n elementos no máximo n vezes se um "valor" é maior que o atual elemento do vetor.

int eMin: essa função tem complexidade $O(1)$, pois retorna apenas um valor independentemente de qualquer outra coisa.

int main: Essa função executa em sua maioria dentro de um loop, dentro desse loop, tem-se mais loops. No caso, aninhamento de repetições diz respeito a uma complexidade $O(n^2)$, já que teremos $f(n) = 2n^2$, logo, $O(n^2)$.