UFC

Relatório - Árvore de Ordenação
Estrutura de Dados

Professor: Atílio Gomes Luiz

Aluno: Fábio Luz Duarte Filho

Matrícula: 474027

Ciência da Computação

Universidade Federal do Ceará - Quixadá/CE - Brasil

1 Listagem dos programas em C++

1.1 Tree.h

Contém o struct TreeNode e os cabeçalhos das funções que envolvem a árvore.

1.2 Tree.cpp

Contém as implementações das funções iniciadas em Tree.h.

1.3 Main.cpp

Contém a função main, a qual pertence o código executável que utiliza as funções da Tree.h para realizar a ordenação em árvore.

2 Listagem dos testes executados

2.1 Entradas

.

```
5
8 20 41 7 2
10
23 3 45 6 1 9 37 99 0 30
3
345 2 1
5
98 34 2 1 76
0
```

2.2 Saídas

.

```
2 7 8 20 41
0 1 3 6 9 23 30 37 45 99
1 2 345
1 2 34 76 98
```

3 Descricões

3.1 Estruturas

Nós de árvore (TreeNode's) implementados em struct.

1 vetor de inteiro para conter as entradas múltiplas.

2 vetores de ponteiro de TreeNode (TreeNode*) para manipular os nós da árvore.

3.2 Decisões e especificações

O programa captura, para cada caso, o tamanho do vetor, checa se esse é uma potência de dois (já que o número de folhas para que a árvore seja cheia precisa ser uma potência de dois). O vetor é criado com tamanho igual à potência de dois, mais próxima, maior ou igual ao tamanho captado. O vetor é preenchido com a segunda linha captada. Caso o tamanho do vetor captado não seja uma potência de dois, o vetor é preenchido com um epsilon(E) que é maior que todos os números do vetor.

Vetores de TreeNode* são criados para conter a árvore e manipular seus nós. Um deles sofrerá alterações de acordo com a comparação de nós. O outro, que serve de backup, sofre apenas uma alteração por ciclo, a qual substitui a menor folha por epsilon. A cada ciclo, o primeiro vetor recebe os nós do vetor backup. Ao fim de cada ciclo, o menor valor da árvore é imprimido no arquivo de saída.

Ao fim de cada caso passado na entrada, a árvore criada é liberada e busca-se o próximo caso, até que a entrada do tamanho do vetor seja igual a zero, caso no qual o programa encerra.

4 Complexidades

4.1 Funções

4.1.1 TreeNode* createNode

Implementada em Tree.cpp. Possui complexidade O(1), visto que apenas cria um nó alocado e "seta"os campos da struct.

4.1.2 TreeNode* freeTree

Implementada em Tree.cpp. Possui complexidade O(n), visto que percorre a árvore recursivamente deletando os nós de baixo para cima.

4.1.3 int treeHeight

Implementada em Tree.cpp. Possui complexidade O(1), visto que apenas calcula a altura da árvore cheia de acordo com a quantidade de folhas dela.

4.1.4 int getKey

Implementada em Tree.cpp. Possui complexidade O(1), visto que apenas retorna a chave de certo nó repassado com parâmetro.

4.1.5 void setKey

Implementada em Tree.cpp. Possui complexidade O(1), visto que apenas "seta"uma chave em um nó, ambos passados como parâmetro.

4.1.6 TreeNode* nodesComparisonToCreate

Implementada em Tree.cpp. Possui complexidade O(1), visto que apenas compara as chaves de dois nós passados como parâmetros e cria um nó que será pai desses dois nós contendo a menor chave entre os nós.

4.1.7 TreeNode* whichNodeIsSmaller

Implementada em Tree.cpp. Possui complexidade O(n), visto que percorre um vetor de TreeNode's passado como parâmetro a fim de encontrar e retornar o nó que possui menor chave.

4.1.8 int whichIsBigger

Implementada em Main.cpp. Possui complexidade O(n), visto que percorre um vetor de inteiros passado como parâmetro a fim de encontrar e retornar o índice que possui o inteiro de maior valor.

4.2 Programa

O programa possui complexidade O(n^2), visto que possui vários for's e while's aninhandos em um único while "global"para cada caso de entrada. Ainda que sejam alguns casos, o programa permanece O(n^2).