Documentação da AVL

Alunos:

Alessandra Souza da Silva - GRR 20182526

Luan Reno Cretella - GRR 20182561

Conceito do Trabalho

Implementar uma árvore binária AVL com funções de inserção e exclusão de nodos com propriedades de uma árvore BST.

Estrutura Nó:

typedef struct nodo {

int chave; -> Variável que armazena o dado retirado do arquivo;

int altura; -> Variável que armazena a altura na qual o nodo se encontra;

struct nodo* direita; -> Ponteiro para o filho da direita;

struct nodo* esquerda; -> Ponteiro para o filho da esquerda;

struct nodo* pai; -> Ponteiro para o pai;

} nodo;

Funções:

main:

A função **int main()** cria um ponteiro do tipo nodo para ser a raiz, que a princípio recebe NULL, para indicar que a árvore está vazia. A ação (i para inserir ou r para retirar) e a chave (valor inteiro) são lidos do arquivo teste.in e executados.

Inserir:

Ao entrar na função **nodo* insert(int chave, nodo *no, int altura_h)** , verifica-se se o nodo é NULL. Se sim, um novo ponteiro aponta para o espaço criado para receber esse

novo valor e os valores preenchidos são a chave e a altura, enquanto os ponteiros apontam para NULL. Caso contrário, ocorre uma comparação entre o valor chave do teste.in e o valor que já se encontra na árvore, se a chave do teste.in for maior a função insert é chamada de novo indo para o filho da direita, do contrário, a função irá para a esquerda. Toda vez que a função insert é chamada, a o valor altura é incrementado em 1.

Balanceamento:

A função **nodo *balanceamento(nodo *no)** verifica se a altura da árvore ou subárvore está balanceada (no caso da AVL, as alturas do lado direito e do esquerdo podem diferenciar em no máximo um na altura). Existem quatro casos em que a árvore está desbalanceada:

1. Se estiver desbalanceado profundamente a esquerda se chama **nodo* LL(nodo * no);**
2. Se estiver desbalanceado a esquerda e direita se chama **nodo* LR(nodo * no);**
3. Se estiver desbalanceado direita e esquerda se chama **nodo* RL(nodo * no);**
4. Se estiver desbalanceado profundamente a direita se chama **nodo* RR(nodo * no);**

Busca:

A função **nodo *busca(nodo* raiz, nodo* no, int chave)** procura onde na árvore está o valor chave do input, seguindo o mesmo método de comparação da função inserir. Quando o valor é encontrado, o processo de exclusão começa.

Exclusão:

A função **nodo* excluir(nodo *raiz, nodo* no)** verifica se o no que se deseja retirar:

- É uma folha: Neste caso, o no é excluido;
- Se for raiz: Neste caso, **excluir_chave_raiz(no, antecessor);**
- Tem dois filhos: Neste caso, **excluir_dois(no, antecessor);**
- Tem um filho à esquerda: Neste caso, **excluir_um(no, no -> esquerda);**
- Tem um filho à direita: Neste caso, **excluir_um(no, no -> direita);**

Funções de exclusão:

1. Caso o no seja uma folha, é criado um ponteiro para o pai do nó e verifica-se se o no está a direita ou à esquerda do pai. Se o no está a direita o ponteiro para o pai que aponta para a direita recebe NULL e se o no está a esquerda o contrário acontece.

2. Caso o no seja a raiz, se cria um ponteiro para o antecessor do no e a função **excluir_chave_raiz(no, antecessor)** é chamada. Nela se copia a chave do antecessor para a raiz e verifica-se existe filho a esquerda, se sim o pai do antecessor recebe a esquerda o filho do antecessor, senão o pai do antecessor recebe a direita NULL.

3. Caso o no tenha dois filhos, se cria um ponteiro para o antecessor do no e a função **excluir_dois(no, antecessor)** é chamada. Nela se copia a chave do antecessor para o nó e verifica-se o antecessor possui filho a esquerda, se sim o pai do antecessor recebe o filho do antecessor. Nos dois próximos ifs verifica-se se no está a direita ou à esquerda do pai e se o antecessor não tiver filhos o pai do antecessor recebe a esquerda NULL. A seguir se no tiver filho a esquerda o antecessor a es querda recebe o que no tinha a esquerda. E por fim atualiza os ponteiros de pai, do antecessor e do pai do antecessor, e direita.

4. Caso o no tenha um filho , se for a esquerda **excluir_um(no, no -> esquerda)** ou se for à direita **excluir_um(no, no -> direita)**, se ocorrer o primeiro o no -> esquerda substitui o lugar do pai do no -> esquerda e se ocorrer o segundo o no -> direita substitui o lugar do pai do no -> direita.

Conclusão do Trabalho

Na primeira tentativa da programação da AVL, o método de exclusão utilizado foi a rotação. Depois de inúmeros erros e complicações, decidimos em usar o código do transplante do Cormen como base e conseguimos chegar na AVL esperada.

Ao final do trabalho, aprendemos a criar uma árvore AVL funcional que realiza duas ações: inserção e exclusão. A manutenção do balanceamento da árvore também segue o esperado e, com exceção dos desafios já mencionados, só tivemos maiores problemas com os ponteiros, por serem muitos.