Árvore Rubro-Negra

Nathan Endo e Nicolas Cosenza

Objetivo:

Desenvolver um projeto com código em C que implemente a estrutura de uma árvore Red-Black (Rubro-Negra), com o intuito de entrar com chaves para inserção e remoção, assim como imprimir o resultado dessas operações no formato solicitado (chave, nível, cor). Em resumo, nesse modelo proposto por Rudolf Bayer (1972), cada nó recebe uma cor (preto ou vermelho), essa atribuição de cores deve respeitar estas regras:

1. O filho de um nó vermelho deve ser preto;
2. A quantidade de nós pretos de um lado da árvore deve ser igual ao do outro;
3. A raiz deve ser preta;
4. Os nós NIL devem ser pretos.

Com base nessas regras, a estrutura de dados tem como objetivo manter o balanceamento da árvore, através de algoritmos que visam a manutenção das características da Red-Black. Como referência para o desenvolvimento das funções do projeto foi utilizada a bibliografia "*Algoritmos – Teoria e Prática*" dos autores Cormen, Leiserson, Rivest e Stein.

Desenvolvimento:

Primeiramente, foi definido a estrutura da árvore e dos nodos, além de todas as funções a serem utilizadas em um arquivo de cabeçalho denominado "functions.h". A estrutura da árvore consiste em um ponteiro para a raiz e outro para o nodo NIL, já os nós consistem em chave, cor, e ponteiros para: filho da esquerda, filho da direita, e pai. Em seguida, foram implementadas algumas funções iniciais, como: create_tree, inicia a árvore e o nodo nil; create_node, cria um nó; destroy_tree, destroi uma árvore; e destroy_node, destroi um nó. Posteriormente, foram feitas funções auxiliares às de inserção e remoção, sendo elas: pesquisa, achar mínimo, achar máximo, antecessor, sucessor, rotação e transplante. Ademais, foi realizada a implementação das funções insert (insere) e insert_fixup (conserta insere). A função insert é idêntica ao da BST, porém realiza a chamada da função insert_fixup no final da sua execução. O novo nó inserido é sempre da cor vermelha, dessa forma a árvore não precisará de ajustes caso o pai do novo nó seja preto. A função insert_fixup trata o nó problemático "X" e aborda 3 casos para cada lado da árvore ("pai = filho da esquerda" e "pai = filho da direita"), ou seja, casos idênticos, porém espelhados, eles sendo:

1. O tio do novo nó é vermelho:
    a. Recolore pai, tio e avô;
    b. X = avô.

2. Zig-Zag (se pai = filho esquerda/direita, X = filho direita/esquerda, respectivamente):
    a. X = pai;
    b. Rotação para esquerda/direita;

    c. Segue para o caso 3.

3. Zig-zig (se pai = filho esquerda/direita, x = filho esquerda/direita, respectivamente):
    a. Pai = preto;
    b. Avô = vermelho;
    c. Rotação para direita/esquerda.

Em seguida, prosseguimos com a implementação das funções delete_node (remove nodo) e delete_fixup (conserta remoção). A função delete_node, assim como a insert, é idêntica ao da BST, porém realiza a chamada da função delete_fixup no final da sua execução se o nó removido ou o antecessor transplantado seja preto. A função delete_fixup trata o nó problemático “X” e aborda 4 casos para cada lado da árvore (“X = filho da esquerda” e “X = filho da direita”), ou seja, casos idênticos, porém espelhados, eles sendo:

1. Irmão vermelho:
    a. irmão = preto;
    b. pai = vermelho;
    c. rotação (esquerda ou direita) de x;
    d. irmão = filho(direita ou esquerda);
    e. segue os outros casos;
2. Irmão e sobrinhos pretos:
    a. irmão = vermelho;
    b. x = pai.
3. Sobrinho direito/esquerdo preto, esquerdo/direito vermelho:
    a. sobrinho esquerdo/direito = preto;
    b. irmão = vermelho;
    c. rotação direita/esquerda do irmão;
    d. irmão é filho da direita/esquerda;
    e. segue para o caso 4.
4. sobrinho direito/esquerdo vermelho:
    a. irmão = cor do pai;
    b. cor do pai = preto;
    c. cor do sobrinho da direita/esquerda = preto;
    d. rotação esquerda/direita de X;
    e. x = raiz.

Por fim, foram feitos o arquivo main.c e a função para imprimir a árvore de acordo com o modelo proposto pelo professor, a função faz chamadas recursivas para a saída ser em ordem crescente de acordo com a chave (travessia em ordem) e calcular durante as chamadas recursivas os níveis correspondentes de cada nó. O arquivo principal foi feito para receber uma entrada de arquivo e ler elementos dois a dois, sendo o primeiro elemento um char, podendo ser i (inserir) ou r (remover), e o segundo elemento é a chave do nó para realizar a operação. Esse arquivo também evita entradas erradas ou chaves repetidas.

Conclusão:

A implementação da árvore Rubro-Negra em linguagem C foi um desafio para ambos os membros do grupo, o projeto estimulou raciocínio lógico e trabalho em equipe,

principalmente nas funções características insert_fixup e remove_fixup. O projeto demonstrou as dificuldades de garantir uma árvore balanceada, que necessita de várias verificações e procedimentos. Em geral, foi um trabalho satisfatório, afinal conseguimos entregar os resultados esperados e entender perfeitamente o funcionamento da árvore Red-Black.