Trabalho ALG III - Implementação da Red-Black

Aluno: Victor Hugo Weigmann Chequer Maia

Obs: O trabalho foi implementado sozinho.

Objetivo:

Criar uma Árvore Red-Black no estilo do livro do Cormen

A árvore foi implementada com base nas estruturas e algoritmos apresentados no livro do Cormen. Por esse motivo, optei por manter os mesmos nomes de variáveis e convenções utilizadas no livro, incluindo o uso do inglês no código, para evitar confusões durante o desenvolvimento.

Seguindo esse padrão, eu utilizei uma estrutura de árvore que inclui um nó chamado `NIL`, que representa os nós nulos, além do nó raiz. Todos os novos nós são inseridos como vermelhos, conforme a convenção do Cormen. Como o trabalho inteiro trata exclusivamente dessa estrutura, optei por concentrar toda a implementação em um único arquivo, por questão de simplicidade.

A utilização de um nó genérico para representar os nulos faz sentido em termos de uso de memória, pois evita a alocação de diversos nós desnecessários. Como o número de nós nulos em uma árvore binária balanceada tende a ser próximo de *n/2*, o uso de ponteiros para um nó `NIL` único resulta em uma economia significativa de memória.

Na minha implementação, os casos de “duplo preto” que surgem durante a remoção são tratados de forma implícita, sem a necessidade de flags ou marcações adicionais. Isso é possível porque, ao analisar os casos do `deleteFixup`, é garantido que, se o nó passado à função for preto, ele já pode ser considerado duplamente preto no contexto da operação.

Uma diferença importante em relação à abordagem do Cormen é que, no processo de remoção, utilizei o **antecessor** para substituição, enquanto o livro opta pelo **sucessor**. Assim, implementei uma função que localiza o maior valor da subárvore esquerda, em vez de buscar o menor valor da subárvore direita.

Outra escolha foi definir a estrutura da árvore rubro-negra como uma variável global, o que facilita a chamada das funções e torna o código mais direto. Como o foco principal do programa é justamente operar sobre essa árvore, essa simplificação me pareceu razoável.

Para a leitura das entradas, configurei o código para interromper quando o `scanf` não conseguir mais ler dois valores do arquivo. Optei por essa abordagem em vez

de utilizar diretamente a verificação de `EOF`, pois já enfrentei problemas com ela anteriormente.

A função de busca segue a lógica tradicional de uma árvore binária de busca (BST), percorrendo a árvore à esquerda ou à direita com base na comparação entre as chaves. Caso existam múltiplos nós com a mesma chave, o novo nó é inserido à direita do último nó com aquele valor.

A impressão da árvore ao final da execução é feita de forma recursiva, utilizando um percurso em ordem (in-order), conforme solicitado no enunciado. Também implementei uma função para liberar a memória de toda a árvore, que percorre os nós recursivamente pela esquerda e direita até alcançar os nós nulos (`NIL`).

O Makefile é bem simples: ele apenas compila o arquivo `redblack.c` e gera o executável `myrn`, sem gerar nenhum código objeto intermediário.

Em freeTree, percorro pós-ordem, para primeiro eliminar todas as subárvores antes da raiz.

Conclusão:

A implementação da Rubro-Negra do trabalho seguiu os conceitos e algoritmos do livro do Cormen, deixando tanto a mesma nomenclatura quanto a mesma estrutura. As decisões de projeto, como a centralização do código em um único arquivo, o uso do nó NIL único e a substituição por antecessor na remoção, foram tomadas para otimizar a clareza, a eficiência e a simplicidade do programa.