Relatório: Implementação da Red Black

Thamiris Yamate Fischer

I. Objetivo

O seguinte relatório tem como objetivo descrever a implementação de uma das árvores vistas em sala: a Red Black. O programa se concentra principalmente nas operações de inserção e remoção, assim como nas funções auxiliares para esses.

II. Implementação

O código foi baseado no livro Algoritmos: Teoria e Prática, do Thomas H. Cormen e nos pseudocódigos passados em sala.

A. Estrutura

Na implementação há duas estruturas de dados: a `struct no t`, para os nós, e a `struct tree t` que aponta para a raiz da árvore.

- **`struct no t`**: Respeita a estrutura do nó de uma BST, porém tem um atributo que indica se a cor é preta ou vermelha.
- **`struct tree t`**: Há um ponteiro para `root` e um para `nil` (nulo). O ponteiro para nulo representa as folhas nulas e substitui o uso direto de `NULL`, já que, por vezes, é necessário acessar atributos (cor, pai, etc.) de um nó nulo. Cada `t->nil` aponta para si mesmo e possui a cor preta, respeitando a propriedade da red-black.

B. Funções

No código, as seguintes funções foram implementadas:

`cria arvore()`

É alocado espaço para o `root` e `nil`, além disso é criado a primeira folha nula de cor preta, assim respeitando a propriedade da red-black.

`cria no()`

Cria um nó vermelho, cujos ponteiros apontam para folhar nulas.

`inserir()`

Nessa função é criado um nó z com a chave passada e buscado iterativamente onde inserir o novo nó. No loop, x é um auxiliar e y recebe o pai de x a cada iteração. No final da função o novo nó é inserido e a função `inserir fixup()` é chamada.

`inserir fixup()`

O inserir fixup() é um algoritmo guloso, ou seja, as inconsistências são resolvidas de baixo para cima, sempre assumindo que o que está para baixo do nó está correto. Essa função trata três casos:

- Caso 1 - Se o tio for vermelho: Nesse caso o pai e tio são coloridos para preto e o avô para vermelho;
- Caso 2 - Se o pai for vermelho e o tio preto ou inexistente: Nesse caso é feito uma rotação, para a esquerda se z for filho da direita ou para a direita se z for filho da esquerda;
- Caso 3 - Se pai e filho são vermelhos em linha: Esse caso é executado independente se o Caso 2 for executado ou não. Aqui o pai e tio são ”recoloridos”e há uma rotação, para a direita se o pai de z é filho da esquerda do avô de z ou para a esquerda caso contrário.

Após a execução desse algoritmo, a raiz sempre é colorida de preto.

`rot esq() ou rot dir()`

Tem o intuito de mover x e y de modo que x vire filho de y.

`remover()`

Nessa função o nó z a ser removido é buscado a partir da chave e o nó x será utilizado no remove fixup(). São tratados dois casos da deleção:

- Caso 1 - z tem nenhum ou apenas um filho: Se z não tiver filho à esquerda, z é substituído pelo seu filho direito, caso contrário, é substituído pelo seu filho esquerdo;
- Caso 2 - z tem dois filhos: O antecessor de z é buscado, a cor de y é salva e x passa a ser o filho esquerdo de y. Se o antecessor de y é o filho direito de z, atualizamos apenas o pai de x, caso contrário, substituímos y pelo seu filho esquerdo. Após isso, z é substituído, os filhos esquerdo e direito de z são realocados em y e a cor original de z é preservada.

No final dos caos o remove fixup() é chamado se a cor original de y for preta e o nó z é liberado.

`remover()`

Esse algoritmo trata de quatro casos, quando há o duplo preto:

- Caso 1 - o irmão w de x é vermelho: O irmão é colorido para preto e o pai para vermelho. Se x for o filho da esquerda, o pai de x é rotacionado para a esquerda e w passa a ser o filho da direita. Caso contrário, os lados se invertem.
- Caso 2 - o irmão w de x é preto e os filhos de w são pretos: O irmão é colorido para vermelho e x recebe o pai.
- Caso 3 - o irmão w de x é preto e o filho da esquerda de w é vermelho:O filho esquerdo é colorido para preto, w para vermelho, há uma rotação para a direita e w se torna o filho direito do pai, se x for o filho esquerdo, caso contrário isso se inverte.

- Caso 4 - irmão w de x é preto e o filho da direita é vermelho. O nó w recebe a cor do pai, o pai é colorido de preto e o filho da direita de w também, além disso há uma rotação para a esquerda, caso x seja o filho esquerdo, e x se torna a raiz. Se x for o filho direito, a rotação é feita para a direita.

No final do algoritmo, x recebe a cor preta.

busca()

Procura recursivamente a chave passada, respeitando a característica da BST: valores menores estão à esquerda do nó e maiores, à direita.

tree max()

Como na remoção a substituição da raiz é pelo antecessor, no trabalho, esse nó é buscado pelo máximo à direita do lado esquerdo.

transplant()

Tem como objetivo substituir a posição de u na árvore pela de v, ou seja, o pai de u deverá apontar para v.

print em ordem()

Imprime recursivamente os nós da árvore. Conforme se vai descendo na árvore o nível aumenta em uma unidade.

destroi()

Libera recursivamente a memória alocada pelos nós, além de liberar o root e o nil.

III. Conclusão

De acordo com os pseudocódigos passados em sala, com o apoio do livro de referência e adaptando o NULL pelo nil na estrutura, a red-black implementada apresenta o comportamento esperado na inserção e remoção, de acordo com os testes disponibilizados.