2° Trabalho de Algoritmos e Estruturas de Dados I

Murilo Leandro Garcia
Glauco Fleury

1 Introdução

Neste projeto foi implementado um TAD de conjuntos na linguagem C. Seguindo as especificações do projeto, foram criadas as funções básicas de uma ED: criação, deleção, inserção de elementos, remoção de elementos, e impressão. Além disso, as seguintes funções especificas de conjuntos: união e intersecção.

Foram utilizadas duas estruturas de dados para armazenamento dos elementos do conjunto: Árvore AVL e Left-Leaning Red Black Tree, em decorrência de sua eficiência para inserção e remoção de elementos $O(\log n)$.

2 Conjunto

2.1 Inserção de elementos

Como ambas as estruturas de dados são baseadas em árvores binárias de busca (ABB), a inserção de um elemento terá complexidade $O(\log n)$, onde n é o tamanho do conjunto sendo inserido. Isso porque a cada nó visitado, ao escolher ir para a esquerda ou para a direita, o número de nós que poderia ser inserido, em um caso ideal, é cortado pela metade, ou seja, se visitam $\approx \log_2 n$.

Todavia, isso é apenas no caso ideal. É possível que no processo de inserção em uma ABB a árvore fique degenerada e se torne algo próximo de uma linked list (no pior caso, igual uma linked list). Por isso, se usaram duas árvores que se balanceiam automaticamente, a AVL e a LLRBT. O processo de balanceamento aumenta ligeiramente a complexidade do algoritmo, mas torna em geral o processo mais rápido.

Na árvore AVL, a altura h de uma árvore com n nós satisfaz:

$$\log_2(n+1) \leq h < \log_\phi(n+2) - \frac{\log_2 5}{2\log_2 \phi} + 2$$

Isso ocorre por conta do rebalanceamento que é feito após cada operação de inserção, que garante que a árvore não se torne degenerada ou muito desbalanceada. Dessa forma, é garantido no mínimo $(1,44\log_2 n)$ de etapas para chegar até o nó onde será inserido. Posteriormente, anda-se de volta à raíz para rebalancear nós desequilibrados. No pior dos casos, você terá que andar todos os $(1,44 \log_2 n)$ nós até balancear, onde a operação de balanceamento é de complexidade constante $O(1)$. Assim, em pior caso a complexidade é $O(2 \cdot 1,44 \log_2 n + 1)$ $= O(\log n)$.

Já na árvore Left Leaning Red Black Tree (LLRBT), a lógica é a mesma, porém com pior caso da altura sendo $2 \log_2 n$. Há mais ainda um 'ColorFlip' que se propaga recursivamente nos $2 \log_2 n$. Assim, há no máximo $O(3 \cdot 2 \log_2 n + 1) = O(\log n)$ para inserir o nó e rebalancear.

2.2 Remoção de elementos

2.3 União

O algoritmo tanto para o caso da AVL quanto para o caso da LLRBT baseia-se em: Criar um conjunto C e inserir todos os elementos dos conjuntos de entrada (A e B). Para inserir todos elementos de um conjunto, a árvore é percorrida em-ordem e para cada nó visitado, é inserido sua chave na árvore do conjunto C.

Assim, chamando $n = |A|$ e $m = |B|$, percorrer-se-ão todos os $n + m$ nós, e para cada um dos nós, será feita uma inserção de complexidade $O(\log(n + m))$. A complexidade total do algoritmo será portanto $O((n + m) \log(n + m))$.

2.4 Intersecção