Relatório da Implementação da Árvore Rubro-Negra

Antonio Joaquim Brych Mendes De Souza

1 Introdução

Este relatório descreve a implementação de uma Árvore Rubro-Negra (RBT), uma estrutura de dados fundamental na ciência da computação para operações eficientes de inserção, remoção e busca.

Árvores Rubro-Negras (RBTrees)

Árvores Rubro-Negras (RBTrees) são estruturas de dados balanceadas usadas principalmente para armazenar e gerenciar conjuntos de dados ordenados.

Estrutura de Nó

- Cada nó na árvore possui uma chave (valor) e informações adicionais que ajudam a estruturar a árvore, como ponteiros para os filhos (esquerdo e direito), um ponteiro para o pai e uma cor (vermelho ou preto).

Propriedades

- **Propriedade das Cores**: Cada nó é colorido de vermelho ou preto.
- **Propriedade da Raiz**: A raiz da árvore é sempre preta.
- **Propriedade das Folhas**: Todas as folhas (ponteiros nulos onde os nós filhos terminam) são consideradas pretas.
- **Propriedade de Cor Red-Black**: Qualquer caminho da raiz até uma folha deve ter o mesmo número de nós pretos.

Operações Principais

- **Inserção**: Ao inserir um novo nó, ele é inicialmente colorido de vermelho. A árvore é então ajustada para manter suas propriedades de cor e estrutura através de rotações e recolorações.
- **Rotações**: São operações que mantêm o equilíbrio da árvore, permitindo que nós mudem de posição para preservar as propriedades da RBT.
- **Remoção**: Ao remover um nó, a árvore é reestruturada para garantir que todas as propriedades da RBT sejam mantidas.

2 Implementação

A implementação da RBT consiste nos seguintes arquivos:

- `RBT.hpp`: Definição das estruturas de dados e protótipos das funções.
- `RBT.cpp`: Implementação das funções da Árvore Rubro-Negra.
- `main.cpp`: Programa principal para testar a funcionalidade da RBT.

3 Descrição das Funções

Aqui estão as funções principais implementadas na RBT.cpp:

- **createNode(int key)**: Cria um novo nó com a chave especificada.
- **rotateLeft(RBTreeNode *&root, RBTreeNode *&node)**: Realiza uma rotação à esquerda na árvore.
- **rotateRight(RBTreeNode *&root, RBTreeNode *&node)**: Realiza uma rotação à direita na árvore.
- **fixInsertRBTree(RBTreeNode *&root, RBTreeNode *&node)**: Corrige a árvore após a inserção de um novo nó.
- **insertRBTree(RBTreeNode *&root, int key)**: Insere um novo nó com a chave especificada na árvore.
- **inorderHelper(RBTreeNode *node, bool verbose)**: Realiza um percurso em ordem simétrica (inorder) na árvore.
- **searchRBTree(RBTreeNode* node, int key)**: Realiza busca por um nó específico na árvore.
- **minimumRBTree(RBTreeNode* node)**: Encontra o nó mínimo na árvore.
- **maximumRBTree(RBTreeNode* node)**: Encontra o nó máximo na árvore.
- **heightRBTree(RBTreeNode* node)**: Calcula a altura da árvore.
- **deleteRBTree(RBTreeNode* node)**: Libera a memória alocada pela árvore.

4 Testes Realizados

A seguir estão os resultados dos testes realizados com a implementação da RBT:

- **Inserção de Elementos**: Tempo de inserção de 10.000 elementos: 1.776.583 nanosegundos.
- **Buscas**:
  - Nó 5.000 encontrado em 375 nanosegundos.
  - Nó 10.001 não encontrado em 333 nanosegundos.
- **Mínimo e Máximo**:
  - Nó mínimo: 1 (Tempo: 292 nanosegundos).
  - Nó máximo: 10.000 (Tempo: 250 nanosegundos).
- **Altura da Árvore**: Altura da árvore: 24 (Tempo: 98.084 nanosegundos).
- **Liberação de Memória**: Tempo para liberar a memória: 594.667 nanosegundos.

5 Conclusão

A implementação da Árvore Rubro-Negra mostrou-se eficiente para operações de inserção, busca e remoção, mantendo a estrutura balanceada e respeitando as propriedades da RBT. A utilização de rotações e correções adequadas garantiu o bom desempenho das operações, como evidenciado pelos testes realizados.