Implementacja drzewa AVL

Arkadiusz Janus

1. **Złożoność obliczeniowa poszczególnych metod.**

   - `Node* right_rotation(Node* parent);`
     O(1), ponieważ jest to tylko kilka przypisań.
     Operacja dominująca: przypisanie
   - `Node* left_rotation(Node* parent);`
   - **O(1),** ponieważ jest to tylko kilka przypisań.
     Operacja dominująca: przypisanie
   - `const int & findMin() const;`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: current->left != nullptr
   - `const int & findMax() const;`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: current->right != nullptr
   - `Node* findMin(Node *t) const;`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: current->left != nullptr
   - `Node* findMax(Node *t) const;`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: current->right != nullptr
   - `bool contains(const int& element) const;`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: current->left != nullptr
   - `bool contains(const int& element, Node* t) const;`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: node->element == element
   - `Node* copy(Node* node);`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: rekurencja
   - `bool isEmpty();`
   - **O(1),** ponieważ mamy bezpośredni dostęp do **root**.
     Operacja dominująca: return
   - `Node* insert(Node *node, int element);`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: porównania
   - `Node* remove(int element, Node* t);`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: porównaina
   - `void printTreeInOrder(Node *t);`
   - **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
     Operacja dominująca: rekurencja
   - `void printTreePreOrder(Node *t);`

- **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
  Operacja dominująca: rekurencja
- `void printTreePostOrder(Node *t);`
- **O(log(n)),** ponieważ musimy przejść przez całą wysokość drzewa.
  Operacja dominująca: rekurencja
- `AVLTree & operator=(const AVLTree& t);`
- **O(log(n)),** ponieważ kopiujemy.
  Operacja dominująca: copy
- `AVLTree & operator=(AVLTree&& t);`
- **O(log(n)),** ponieważ kopiujemy.
  Operacja dominująca: copy