**Описание структуры данных**

Структура данных представляет собой модифицированное **красно-черное дерево**, дополненное поддержкой весов узлов. Она сочетает в себе свойства самобалансирующегося дерева поиска и эффективное управление весами, что позволяет выполнять операции вставки, удаления и поиска с учетом вероятностей, основанных на весах.

**Методы:**

1. **add(key, weight) — добавление/обновление узла:**
   * Если узел с данным ключом уже существует, обновляется его вес.
   * Иначе вставляется новый узел с указанным весом. Балансировка выполняется через механизм исправления дерева (insert\_fix).
2. **delete(key) — удаление узла:**
   * Удаляет узел с данным ключом, если он существует.
   * Выполняет перестройку дерева с учетом свойств красно-черного дерева.
3. **get() — взвешенный выбор:**
   * Возвращает ключ узла, выбранного пропорционально его весу.
   * Используется генерация случайного числа в диапазоне от 0 до total\_weight корневого узла.
4. **Балансировка:**
   * Методы **left\_rotate** и **right\_rotate** реализуют вращения, которые используются при балансировке.
   * Исправления после вставки (insert\_fix) и удаления (delete\_fix) поддерживают свойства красно-черного дерева.
5. **update\_total\_weight(node) — обновление веса:**
   * Пересчитывает total\_weight для узла на основе его собственного веса и весов дочерних узлов.

**Сложность операций:**

* **Добавление и обновление(add), удаление(delete), взвешенный выбор (get):** O(log(n))

**Пример использования:**

rbt = RedBlackTree()

rbt.add(1, 1) # Добавить узел с ключом 1 и весом 10

rbt.add(1,10) # Вес для узла с ключом 1 меняется на 10

rbt.add(2, 20) # Добавить узел с ключом 2 и весом 20

rbt.add(3, 30) # Добавить узел с ключом 3 и весом 30

print(rbt.get()) # Случайный выбор узла с учетом весов (с большей вероятностью вернет 3)

rbt.delete(2) # Удаление узла с ключом 2