Красно черное дерево это структура которая основана на структуре простого бинарного дерева. У этой структуры такие операции как поиск вставка или удаление имеют логарифмическую сложность. Кроме того эта структура гарантирует несколько важных условий: 1) листы черные (NULL листы) 2) корень черный 3) любой путь до листа имеет одинаковое кол-во черных вершин 4) у красной вершины оба ребенка чёрные.

**Вставка:**

Теперь поговорим о том как работает вставка в красно черном дереве. В целом она работает на основе вставки в простое бинарное дерево: когда мы хотим добавить новый лист нам следует начать смотреть наше дерево с корня если корень пуст то добавляем его иначе если элемент меньше корня то мы переходим в левого ребенка и делаем с ним все тоже что и с корнем. Когда мы дойдем до вершины которой нет то мы добавляем ее и красим в красный. При этом важно понимать что нарушится условие красно черного дерева если родитель созданной вершины тоже красный.

Нам седeдует провести проверку после того как мы вставили новую вершину. В случаи если родитель рассматриваемого объекта красный то мы должны узнать дядя красный или нет. Если да то нам следует покрасить родителя и дядю в черных а дедушку в красный и тогда проверить тоже самое для дедушки потому что там теперь могла возникнуть нарушение условия о 2 черных сыновьях у красного родителя (т е мы берем родителя дедушки и смотрим его). Когда мы дойдем до корня нам следует завершить вышеописанную процедуру. Потом нужно покрасить корень корень в черный чтобы гарантировать условие 2.

Если родитель – левый сын дедушки а дядя не красный нам следует сделать поворот дерева: если наша вершина правый сын то мы делаем левый поворот относительно ее родителя иначе красим родителя в черный дедушку в красный и делаем правый поворот.

Если родитель – правый сын дедушки а дядя не красный нам следует сделать поворот дерева: если наша вершина левый сын то мы делаем правый поворот относительно ее родителя иначе красим родителя в черный дедушку в красный и делаем левый поворот.

Левый поворот по x где y – сын x. A(left), b(right) – сыны y. После поворота x(left), b(right) - сыны y. a(right) – сын x.

Правый поворот по x где y – сын x. A(left), b(right) – сыны y. После поворота a(left), x(right) - сыны y. b(right) – сын x.

**Поиск:**

Задача: Нужно найти x

Решение:

0) у = корень

1) Если х больше у то y = правый ребёнкок если меньше у = левый иначе если равен то мы нашли х

2) Если у = NULL то х нет в дереве

**Сравнение с AVL Tree :**

Деревья AVL более сбалансированы по сравнению с красно-черными деревьями, но они могут вызывать больше поворотов во время вставки и удаления. Поэтому, если ваше приложение подразумевает частые вставки и удаления, то следует отдать предпочтение красно-черным деревьям. А если вставки и удаления происходят реже, а поиск — более частая операция, то дерево AVL следует отдать предпочтение красно-черному дереву.