**Красно-черное дерево**

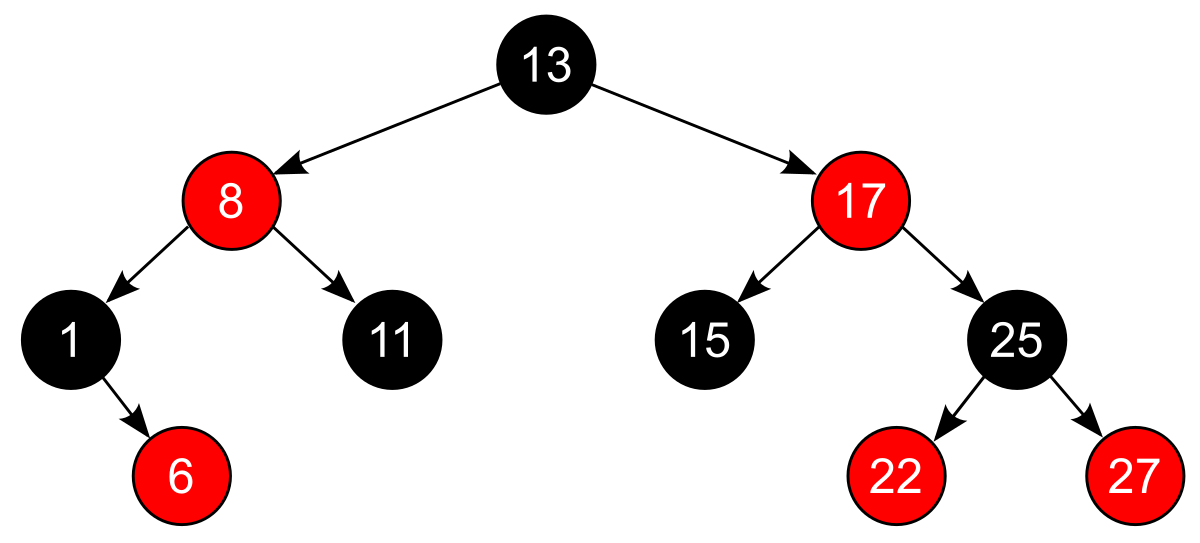
Котов Никита 11-401

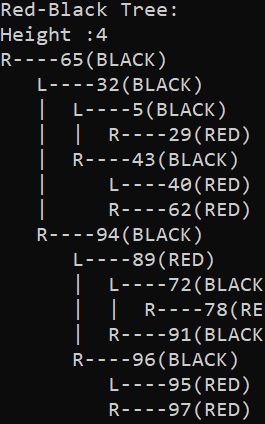
**Красно-черное дерево** - это вид бинарного дерева, основной сутью которого является способность к самобалансировке.

**Сбалансированность** достигается за счет введения дополнительного атрибута узла дерева - ”цвета”. В каждом узле дерева помимо элемента хранится 1 бит информации о том, красный ли узел или черный, при этом это может быть не только цвет, но и любая другая информация позволяющая отличить один тип узла от другого. Например, 1 или 0, true или false и т. п.

* **Принципы организации (свойства) КЧД:**
* Корень дерева **черный**.
* Все листья, не содержащие данных, **черные**
* Оба потомка каждого красного узла – **черные**

Пример красно-черного дерева:



Красно-черное дерево в консоли:

**Таблица сложности:**

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Description | Time Complexity | Space Complexity |
| Insert | Inserts a new element into the tree. | O(log n) | O(1) |
| Delete node | Removes an element from the tree. | O(log n) | O(1) |
| Search | Searches for an element in the tree. | O(log n) | O(1) |
| Left Rotate | Performs a left rotation on a given node. | O(1) | O(1) |
| Right Rotate | Performs a right rotation on a given node. | O(1) | O(1) |

Одно из главных плюсов красно-черного дерева – *логарифмическая* сложность поиска, вставки и удаления даже в худшем случае.

**Алгоритм балансировки кчд**

Необходимость в балансировке подсказывают цвета: родитель и один из его детей красные

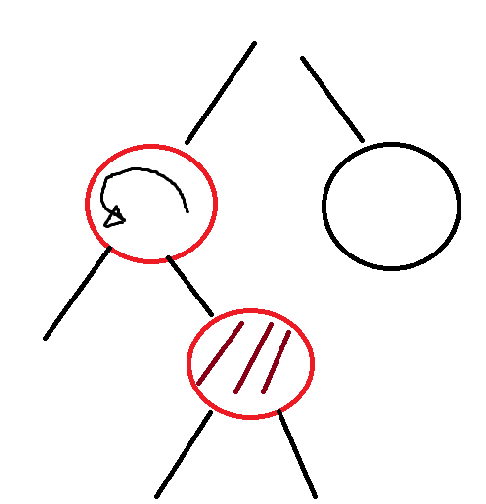
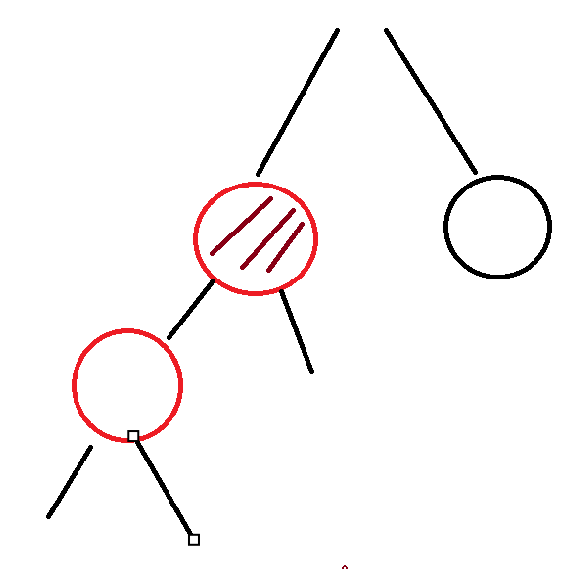
**Алгоритм:**

* если правая нода красная и левая нода черная - левосторонний поворот
* если левая нода красная и левая нода левой ноды красная - правосторонний поворот
* если левая нода красная и правосторонняя нода красная - делаем свап цвета.  
  *(родитель становится красным а его левая и правая ноды черными)*

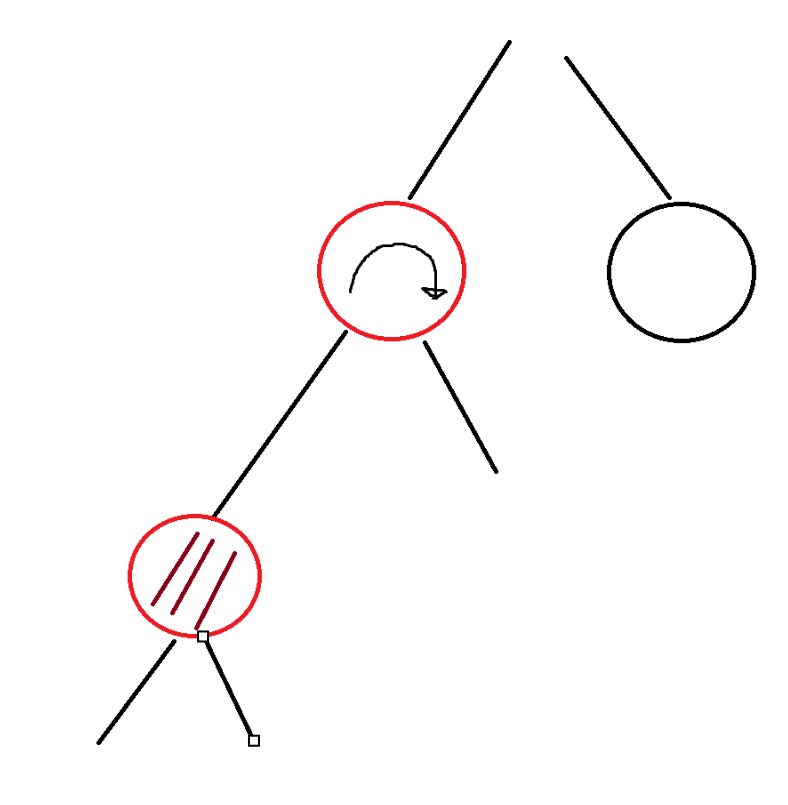
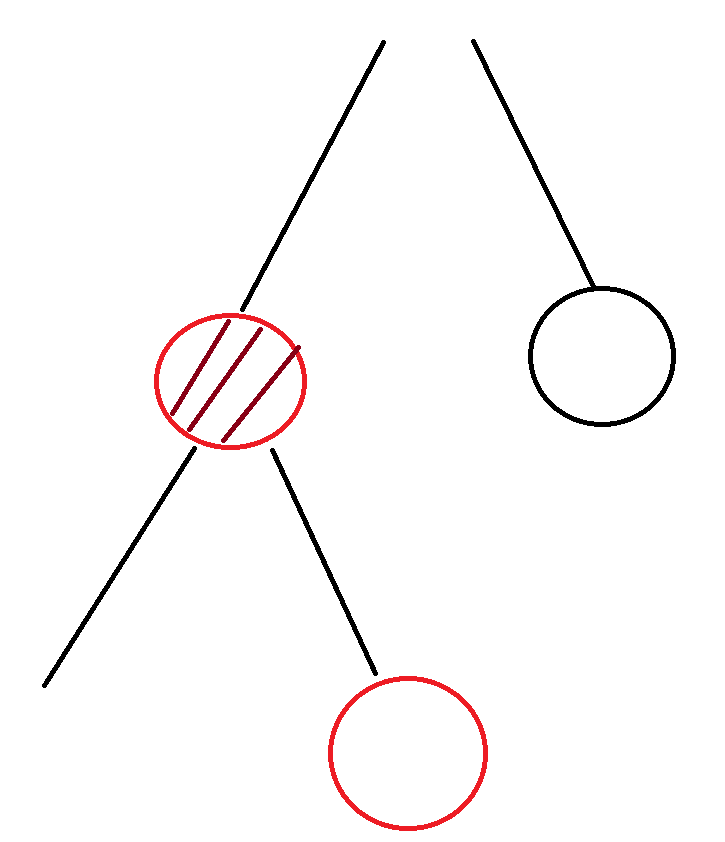
**Повороты:**

Цель: Переместить правого потомка (child) на место текущего узла (node), сохраняя свойства бинарного дерева поиска.

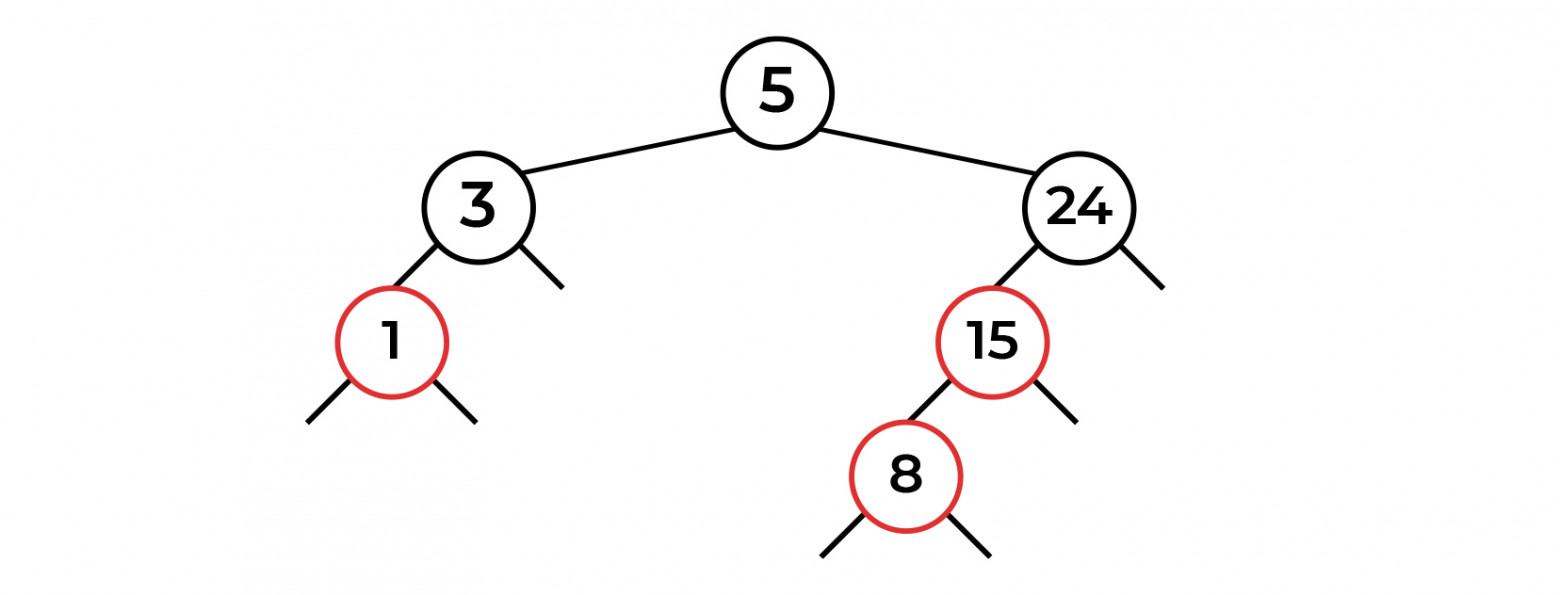
**Левый поворот (rotateLeft):**

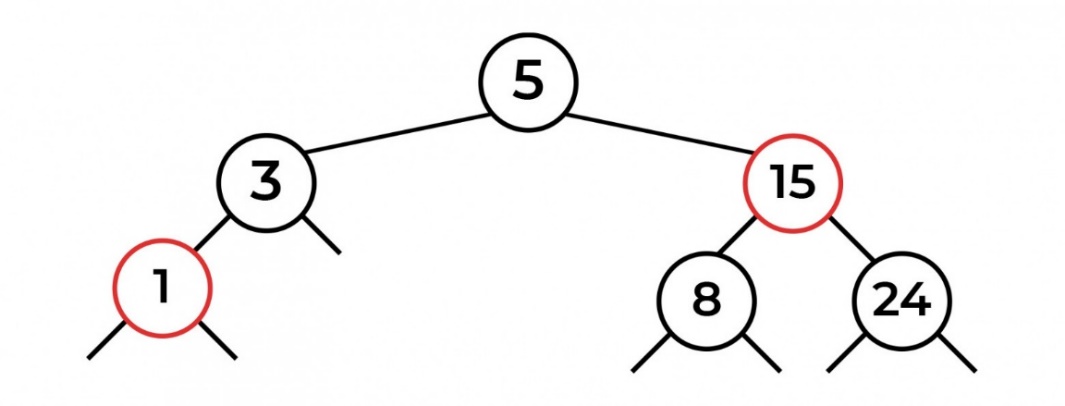
****

**Правый поворот(rotateRight):**



Еще пример:





**Удаление:**

* Если у удаляемого узла нет дочерних узлов, просто удалите его и обновите родительский узел.
* Если у удаляемого узла есть только один дочерний узел, замените его дочерним узлом.
* Если у удаляемого узла есть два дочерних элемента:
  + то замените этот узел на следующий по порядку, который является крайним левым узлом в правом поддереве.
  + Затем удалите следующий по порядку узел, как если бы он имел не более одного дочернего элемента.
* После удаления узла свойства красно-черного могут быть нарушены. Для восстановления этих свойств в узлах дерева выполняются некоторые изменения цвета и повороты. Изменения аналогичны тем, которые выполняются при вставке, но с другими условиями.
* Операция удаления в красно-черном дереве занимает в среднем O(log n) времени, что делает ее хорошим выбором для поиска и удаления элементов в больших наборах данных.

**Исправление нарушений при удалении:**

Когда удаляется черный узел, мы решаем проблему двойного черного, основываясь на цвете дочернего узла и цветах его дочерних элементов:

* Вариант 1: Дочерний узел красный:
  + Поверните родительский узел и перекрасьте его и родительский узел.
* Вариант 2: Дочерний узел черный:
  + Вариант 2.1: Дочерние элементы брата черные:
    - Перекрасьте родного брата и поднимите черный цвет вверх.
  + Вариант 2.2: Один из дочерних элементов брата красный:
    - Если дальний дочерний элемент у брата красный: выполните поворот родительского элемента и дочернего элемента у брата и соответствующим образом перекрасьте.
    - Если ближний дочерний элемент у брата красный: поверните   
      брата и его дочернего элемента, затем выполните описанные выше действия.

**Плюсы и минусы:**

**Преимущества красно-черных деревьев:**

* **Сбалансированный:** Красно-черные деревья самобалансируются, то есть они автоматически сохраняют баланс между высотами левых и правых поддеревьев. Это гарантирует, что операции поиска, вставки и удаления займут O(log n) времени в худшем случае.
* **Эффективный поиск, добавление и удаление:** Благодаря своей сбалансированной структуре, красно-черные деревья предлагают эффективные операции. Поиск, вставка и удаление - все это занимает O(log n) времени в худшем случае.
* **Простота в реализации:** Правила для сохранения свойств красно-черного дерева относительно просты и прямолинейны в реализации.
* **Широко используется:** красно-черные деревья являются популярным выбором для реализации различных структур данных, таких как map, set и priority queue.

**Недостатки красно-черных деревьев:**

* **Более сложные, чем другие сбалансированные деревья:** По сравнению с более простыми сбалансированными деревьями, такими как AVL-деревья, красно-чёрные деревья имеют более сложные правила вставки и удаления.
* **Постоянная надбавка:** Сохранение свойств Red-Black Tree добавляет небольшую надбавку к каждой операции вставки и удаления.
* **Не оптимальна для всех случаев использования:** Хотя красно-черные деревья эффективны для большинства операций, они могут не быть лучшим выбором для приложений, где частые вставки и удаления необходимы, так как постоянная накладная работа может стать значительной.

**Применение красно-черных деревьев:**

* **Реализация map и set**: Красно-чёрные деревья часто используются для реализации map и set, где эффективный поиск, вставки и удаление являются решающими.
* **Очереди приоритетов:** Красно-черные деревья могут быть использованы для создания очередей приоритетов, в которых элементы упорядочены по их приоритетам.
* **Файловые системы:** Красно-черные деревья используются в некоторых файловых системах для управления структурами файлов и каталогов.
* **Базы данных в памяти:** Красно-черные деревья иногда используются в базах данных в памяти для хранения и извлечения данных эффективно.
* **Графика и разработка игр:** Red-Black Trees могут быть использованы в графике и разработке игр для таких задач, как обнаружение столкновений и поиск путей.