17. Red-black trees, definition, asymptotical estimate for the height of a red-black tree.

---

1. Введение

> **Красно-чёрное дерево** (red-black tree) — это тип самобалансирующегося двоичного дерева поиска (Binary Search Tree, BST), в котором каждая вершина помечена цветом (красным или чёрным) с целью поддержания равномерной высоты и обеспечения логарифмической временной сложности для основных операций (поиск, вставка, удаление)..

Основное свойство красно-чёрного дерева в том, что оно *близко к идеально сбалансированному*, то есть высота красно-чёрного дерева растёт пропорционально $\log n$, где n — число элементов (вершин) в дереве.

---

2. Свойства красно-чёрного дерева

Для того чтобы красно-чёрное дерево оставалось сбалансированным, оно должно удовлетворять набору строгих **свойств**:

1. **Каждая вершина либо красная, либо чёрная.**
2. **Корень дерева всегда чёрный.**
3. **Пустые дочерные узлы черные**
4. **У красной вершины оба ребёнка (если они существуют) — чёрные.**
   Иными словами, в красно-чёрном дереве не может быть двух последовательных красных вершин по пути от корня к листу.
5. **Для любой вершины все простые пути от неё до пустых дочерних узлов содержат одинаковое число чёрных вершин.**
   Это свойство часто называют "чёрная высота" (black height): количество чёрных вершин на пути от корня к каждому листу не меняется.

Совокупность этих свойств обеспечивает важный факт: при добавлении и удалении элементов дерево не теряет своей сбалансированности (с точностью до вращений и перекрашиваний).

---

3. Высота красно-чёрного дерева

В общем случае, если бы мы не заботились о балансировке, высота двоичного дерева поиска могла бы достигать $\mathcal{O}(n)$. В **красно-чёрном же дереве** высота гарантированно не превосходит $2\log_2(n+1)$. Более строго:

- **Минимальная высота** (при максимально равномерном распределении узлов) будет порядка $\log n$.
- **Максимально возможная высота** красно-чёрного дерева — тоже $\mathcal{O}(\log n)$. При точных подсчётах получается верхняя оценка $2\log_2(n+1)$ (или близкая к этому, в зависимости от конкретных реализаций и деталей).

Утверждение об оценке (высоты) и сложности красно-чёрного дерева

Пусть:

- L — полная высота красно-чёрного дерева (количество вершин на самом длинном пути от корня к листу).
- bh (black height) — «чёрная высота», то есть число чёрных вершин на любом пути от корня к листу (по свойству красно-чёрного дерева эта величина одинакова для всех путей).

1. Связь между L и bh

① **Чёрная высота в два раза меньше высоты всего дерева (максимум).**
Почему? Потому что по свойству 4 не может быть двух красных вершин подряд. Следовательно, *между двумя чёрными вершинами может находиться не более одной красной*. Таким образом, путь максимальной длины будет чередовать чёрные и красные вершины, а значит, в худшем случае $L \leq 2 \cdot bh$.
   - Если L нечётно, то по сути мы имеем bh чёрных и $bh - 1$ красных вершин на максимальном пути ($L = bh + (bh - 1) = 2\,bh - 1$), следовательно $bh \approx \frac{L+1}{2}$.
   - Если L чётно, то на пути могут чередоваться чёрная-красная-чёрная-красная, и получаем $L = 2\,bh$, т.е. $bh = \frac{L}{2}$.

В любом случае, $L \leq 2\,bh$.

2. Связь между n (числом узлов) и bh

① Минимальное число вершин n при фиксированной чёрной высоте bh оценивается как:

$$n \;\geq\; 2^{\,bh} - 1,$$

поскольку хотя бы на каждом «уровне» по пути мы имеем одну чёрную вершину, и вся «ширина» дерева растёт экспоненциально от числа таких чёрных уровней.

② Из $n \geq 2^{bh} - 1$ следует $bh \leq \log_2(n+1)$.

3. Итоговая оценка высоты L

- Из предыдущих пунктов:
  ① $L \leq 2\,bh$.
  ② $bh \leq \log_2(n+1)$.
- Следовательно,

$$L \;\leq\; 2\,\log_2(n+1) \;=\; \mathcal{O}(\log n).$$

Таким образом, **высота красно-чёрного дерева** растёт пропорционально $\log n$. Отсюда вытекают **логарифмические оценки** на операции вставки, удаления и поиска: каждая из них занимает $\mathcal{O}(\log n)$ времени.

Итоговые формулы для количества узлов n

Ранее мы получили, что при чёрной высоте bh минимальное число узлов в красно-чёрном дереве можно оценить как $2^{bh} - 1$. Однако в дереве также могут быть красные вершины. По свойствам красно-чёрного дерева:

- bh — число чёрных вершин от корня до любого листа.
- L — общее число уровней (высота) дерева.

Так как на пути из корня к листу между двумя чёрными вершинами может быть **не более одной красной**, то максимальное количество "лишних" (красных) узлов на самом длинном пути есть $L - bh$. Прикидывая грубо сверху (с учётом потенциальных красных вершин и структуры дерева), можно записать:

$$n \;\approx\; \bigl(2^{bh} - 1\bigr) \;+\; (L - bh).$$

Здесь $2^{bh} - 1$ отвечает за минимально возможное число узлов при данных чёрных уровнях, а $(L - bh)$ отражает верхнюю оценку количества красных вершин вдоль самого длинного пути. Формально можно сказать, что

$$n = 2^{bh} - 1 + (L - bh)$$

(это нестрогая модель, однако она показывает, как линейная поправка $(L - bh)$ складывается с экспоненциальной частью 2^{bh}).

1. $n = \Theta(2^{bh})$

Так как 2^{bh} есть экспоненциальная составляющая, а $(L - bh)$ — линейная, то при больших bh член 2^{bh} будет доминировать:

$$n \;=\; 2^{bh} - 1 + (L - bh) \;=\; \Theta\bigl(2^{bh}\bigr).$$

2. $n = \Theta\bigl((\sqrt{2})^{L}\bigr)$

Мы также знаем, что $L \le 2\,bh$ и, наоборот, $bh \ge \frac{L}{2}$. Тогда:

$$2^{bh} \;=\; 2^{\,\frac{L}{2}} \;=\; (\sqrt{2})^{L}.$$

Отсюда получается:

$$n \;=\; \Theta\bigl(2^{bh}\bigr) \;=\; \Theta\bigl((\sqrt{2})^{L}\bigr).$$

Таким образом, обе записи для n взаимосвязаны через соотношение между bh и L, и в любом случае получаем экспоненциальную зависимость от чёрной высоты bh или от половины полной высоты L.

Следствие

В любом красно-чёрном дереве, содержащем n узлов и имеющем L уровней, выполняется

$$\Theta(\log_2 n) \;\le\; L \;\le\; \Theta(\log_{\sqrt{2}} n).$$

Так как

$$\log_{\sqrt{2}} n \;=\; \frac{\log_2 n}{\log_2(\sqrt{2})} \;=\; \frac{\log_2 n}{\frac{1}{2}} \;=\; 2\log_2 n,$$

то обе оценки — нижняя и верхняя — являются константными множителями $\log n$.

Следовательно,

$$L \;=\; \Theta(\log n).$$

Именно поэтому высота (число уровней) красно-чёрного дерева с n узлами растёт пропорционально $\log n$.

---

4. Основные операции в красно-чёрном дереве

4.1. Поиск (Search)

Поиск элемента в красно-чёрном дереве не отличается от поиска в обычном двоичном дереве поиска:

① Сравниваем ключ искомого элемента с ключом текущей вершины.
② Если ключ меньше — переходим в левое поддерево, если больше — в правое, если равен — нашли вершину.

③ Процедура продолжается рекурсивно (или итеративно) до тех пор, пока не найдём элемент или не достигнем пустого узла.

Время работы: $\mathcal{O}(\log n)$ — благодаря тому, что высота дерева $\mathcal{O}(\log n)$.

4.2. Вставка (Insertion)

① Новый узел добавляется, как в обычном **бинарном дереве поиска (BST)**:
- Если значение меньше текущего узла, идем в **левое поддерево**.
- Если значение больше текущего узла, идем в **правое поддерево**.

② Новый узел всегда окрашивается в **красный**.

После вставки возможно нарушение свойств красно-черного дерева:

- Если родитель нового узла тоже **красный**, возникает **нарушение свойства 4**.

Исправление выполняется с помощью:

- **Перекраски** узлов.
- **Поворотов (левый или правый)**.

Возможны три случая:

① **Дядя узла красный**
- **Условие**: Дядя нового узла (брат его родителя) — **красный**.
- **Действия**:
  - Перекрасить родителя и дядю в **черный**.
  - Перекрасить дедушку в **красный**.
  - Переместить фокус на дедушку и повторить проверку.

② **Дядя узла черный, и узел находится "напротив" родителя (зигзаг)**
- **Условие**: Дядя нового узла — **черный**, и новый узел находится в противоположной ветви (лево-право или право-лево).
- **Действия**:
  - Выполнить **поворот**:
    - Если новый узел — левый ребенок правого родителя, выполнить **левый поворот**.
    - Если новый узел — правый ребенок левого родителя, выполнить **правый поворот**
    - После поворота случай превращается в **Случай 3**.

③ **Дядя узла черный, и узел находится "в одной линии" с родителем**
- **Условие**: Дядя нового узла — **черный**, и новый узел и родитель находятся в одной ветви (лево-лево или право-право).
- **Действия**:
  ① Выполнить **поворот** вокруг дедушки:
    - Если узел в ветви лево-лево, выполнить **правый поворот**.
    - Если узел в ветви право-право, выполнить **левый поворот**.
  ② Перекрасить:
    - Родителя в **черный**.
    - Дедушку в **красный**.
  ③ **Проверка корня**
  Если после всех исправлений корень стал **красным**, перекрасить его в **черный**.

пример:

```
// Вставка значения 15:
1. Изначальное дерево:
   20(B)
  /    \
10(B)  30(B)

2. Шаг 1: Добавляем 15 (красный):
   20(B)
  /    \
10(B)  30(B)
   \
   15(R)

3. Шаг 2: Проверяем свойства:
- Узел 15 красный, родитель 10 черный → нарушений нет.
```

Сложность:

- **Вставка:** $O(\log n)$.
- **Исправления:** $O(\log n)$, так как высота красно-черного дерева ограничена $2 \log n$.

4.3. Удаление (Deletion)

При **удалении** узла из красно-чёрного дерева:

1. Удаляем вершину как в обычном BST (заменяя её либо минимальным элементом из правого поддерева, либо максимальным из левого поддерева, или просто удаляя, если у неё меньше двух детей).
2. Если мы удалили/заменили *чёрную* вершину, то могли нарушиться свойства чёрной высоты. При этом может возникнуть необходимость *дополнительных вращений* и *перекрашиваний*, чтобы восстановить все красно-чёрные свойства.
3. Процедура исправления обычно сложнее, чем при вставке, но асимптотически тоже укладывается в $\mathcal{O}(\log n)$ вращений и перекрашиваний.

---

5. Асимптотические оценки

- **Вставка**: $\mathcal{O}(\log n)$.
- **Удаление**: $\mathcal{O}(\log n)$.
- **Поиск**: $\mathcal{O}(\log n)$.
- **Высота дерева**: $\mathcal{O}(\log n)$.

Таким образом, красно-чёрные деревья эффективны для поддержания отсортированных динамических наборов данных: операции (вставка, удаление, поиск) выполняются за логарифмическое время.

---

6. Выводы

1. Красно-чёрное дерево — **самобалансирующееся** двоичное дерево поиска.
2. Благодаря ограничениям по "красным" и "чёрным" вершинам достигается равномерная высота, и операции выполняются за $\mathcal{O}(\log n)$.

③ Главный показатель сбалансированности — **чёрная высота**, общая для всех путей от корня до листьев.
④ Максимальная высота красно-чёрного дерева не превышает $2\log_2(n+1)$, что упрощённо даёт оценку $\mathcal{O}(\log n)$.

- Связь AVL и Red/Black trees:

  В общем случае *высота красно-чёрного дерева* может быть чуть больше, чем у *AVL-дерева*, и поэтому (при одинаковом числе узлов n) *поиск* в AVL-дереве может оказаться немного эффективнее. Однако за это приходится «платить» более сложными механизмами балансировки при вставке и удалении в AVL.

  В *красно-чёрном дереве* ребалансировка при добавлении/удалении устроена проще (меньше ограничений на разницу высот поддеревьев), из-за чего операции *вставки* и *удаления* обычно выполняются быстрее, чем в AVL.

Итог:

- *AVL* даёт более жёсткую балансировку (меньшая высота, быстрее поиск), но при этом сложнее обрабатывает вставки и удаления.
- *Красно-чёрное дерево* допускает немного большую высоту (потенциально чуть медленнее поиск), но упрощает операции вставки и удаления.

---

7. Применение

Благодаря хорошему балансу, красно-чёрные деревья используются в качестве фундаментальной структуры данных во многих языках и библиотеках:

- В стандартной реализации **std::map** и **std::set** в C++ обычно используются красно-чёрные деревья.