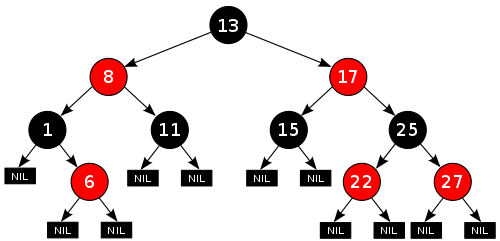
**Красно-чёрное дерево** (англ. *Red-black tree*, *RB-Tree*) — один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева — «цвета». Этот атрибут может принимать одно из двух возможных значений — «чёрный» или «красный».

Изобретателем красно-чёрного дерева считают немца Рудольфа Байера. Название «красно-чёрное дерево» структура данных получила в статье Л. Гимбаса и Р. Седжвика (1978). По словам Гимбаса, они использовали ручки двух цветов. По словам Седжвика, красный цвет лучше всех смотрелся на лазерном принтере.

Красно-чёрное дерево используется для организации сравнимых данных, таких как фрагменты текста или числа. Листовые узлы красно-чёрных деревьев не содержат данных, благодаря чему не требуют выделения памяти — достаточно записать в узле-предке в качестве указателя на потомка нулевой указатель. Однако, в некоторых реализациях для упрощения алгоритма могут использоваться явные листовые узлы.



**Принцип устройства**

**Красно-чёрное дерево — двоичное дерево поиска, в котором каждый узел имеет атрибут *цвета*. При этом:**

* **Узел может быть либо красным, либо чёрным и имеет двух потомков;**
* **Корень — как правило чёрный. Это правило слабо влияет на работоспособность модели, так как цвет корня всегда можно изменить с красного на чёрный;**
* **Все листья — чёрные и не содержат данных.**
* **Оба потомка каждого красного узла — чёрные.**
* **Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов.**

**Благодаря этим ограничениям, путь от корня до самого дальнего листа не более чем вдвое длиннее, чем до самого ближнего и дерево примерно сбалансировано. Операции вставки, удаления и поиска требуют в худшем случае времени, пропорционального длине дерева, что позволяет красно-чёрным деревьям быть более эффективными в худшем случае, чем обычные двоичные деревья поиска.**

**Чтобы понять, как это работает, достаточно рассмотреть эффект свойств 4 и 5 вместе. Пусть для красно-чёрного дерева T число чёрных узлов от корня до листа равно B. Тогда кратчайший возможный путь до любого листа содержит B узлов и все они чёрные. Более длинный возможный путь может быть построен путём включения красных узлов. Однако, благодаря п.4 в дереве не может быть двух красных узлов подряд, а согласно пп. 2 и 3, путь начинается и кончается чёрным узлом. Поэтому самый длинный возможный путь состоит из 2B-1 узлов, попеременно красных и чёрных.**

**Если разрешить нелистовому узлу иметь меньше двух потомков, а листовым — содержать данные, дерево сохраняет основные свойства, но алгоритмы работы с ним усложнятся. Поэтому в статье рассматриваются только «фиктивные листовые узлы», которые не содержат данных и просто служат для указания, где дерево заканчивается. Эти узлы могут быть опущены в некоторых иллюстрациях. Из п.5, также следует, что потомками красного узла могут быть либо два чёрных промежуточных узла, либо два чёрных листа, а с учётом п.3 и 4 — что если у чёрного узла один из потомков — листовой узел, то вторым должен быть либо тоже листовой, либо вышеописанная конструкция из одного красного и двух листовых.**

**Также в литературе встречается трактовка, в которой в красный/чёрный цвета раскрашивают не сами узлы, а ведущие к ним рёбра — но это не имеет большого значения для понимания принципа его работы.**

Вставка

Новый узел в красно-чёрное дерево добавляется на место одного из листьев, окрашивается в красный цвет и к нему прикрепляется два листа (так как листья являются абстракцией, не содержащей данных, их добавление не требует дополнительной операции). Что происходит дальше, зависит от цвета близлежащих узлов. Заметим, что:

* Свойство 3 (Все листья чёрные) выполняется всегда.
* Свойство 4 (Оба потомка любого красного узла — чёрные) может нарушиться только при добавлении красного узла, при перекрашивании чёрного узла в красный или при повороте.
* Свойство 5 (Все пути от любого узла до листовых узлов содержат одинаковое число чёрных узлов) может нарушиться только при добавлении чёрного узла, перекрашивании красного узла в чёрный (или наоборот), или при повороте.

*Примечание*: Буквой **N** будем обозначать текущий узел (окрашенный красным). Сначала это новый узел, который вставляется, но эта процедура может применяться рекурсивно к другим узлам (смотрите случай 3). **P** будем обозначать предка **N**, через **G** обозначим дедушку **N**, а **U** будем обозначать дядю (узел, имеющий общего родителя с узлом **P**). Отметим, что в некоторых случаях роли узлов могут меняться, но, в любом случае, каждое обозначение будет представлять тот же узел, что и в начале. Любой цвет, изображенный на рисунке, либо предполагается в данном случае, либо получается из других соображений.

* **Случай 1:** Текущий узел **N** в корне дерева. В этом случае, он перекрашивается в чёрный цвет, чтобы оставить верным Свойство 2 (Корень — чёрный). Так как это действие добавляет один чёрный узел в каждый путь, Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается.
* **Случай 2:** Предок **P** текущего узла чёрный, то есть Свойство 4 (Оба потомка каждого красного узла — чёрные) не нарушается. В этом случае дерево остаётся корректным. Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается, потому что текущий узел **N** имеет двух чёрных листовых потомков, но так как **N** является красным, путь до каждого из этих потомков содержит такое же число чёрных узлов, что и путь до чёрного листа, который был заменен текущим узлом, так что свойство остается верным.
* **Случай 3:** Если и родитель **P,** и дядя **U** — красные, то они оба могут быть перекрашены в чёрный, и дедушка **G** станет красным (для сохранения свойства 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов)). Теперь у текущего красного узла **N** чёрный родитель. Так как любой путь через родителя или дядю должен проходить через дедушку, число чёрных узлов в этих путях не изменится. Однако, дедушка **G** теперь может нарушить свойства 2 (Корень — чёрный) или 4 (Оба потомка каждого красного узла — чёрные) (свойство 4 может быть нарушено, так как родитель **G** может быть красным). Чтобы это исправить, вся процедура рекурсивно выполняется на **G** из случая 1.
* **Случай 4:** Родитель **P** является красным, но дядя **U** — чёрный. Также, текущий узел **N** — правый потомок **P**, а **P** в свою очередь — левый потомок своего предка **G**. В этом случае может быть произведен поворот дерева, который меняет роли текущего узла **N** и его предка **P**. Тогда, для бывшего родительского узла **P** в обновленной структуре используем случай 5, потому что Свойство 4 (Оба потомка любого красного узла — чёрные) все ещё нарушено. Вращение приводит к тому, что некоторые пути (в поддереве, обозначенном «1» на схеме) проходят через узел **N**, чего не было до этого. Это также приводит к тому, что некоторые пути (в поддереве, обозначенном «3») не проходят через узел **P**. Однако, оба эти узла являются красными, так что Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается при вращении. Однако Свойство 4 всё ещё нарушается, но теперь задача сводится к Случаю 5.
* **Случай 5:** Родитель **P** является красным, но дядя **U** — чёрный, текущий узел **N** — левый потомок **P** и **P** — левый потомок **G**. В этом случае выполняется поворот дерева на **G**. В результате получается дерево, в котором бывший родитель **P** теперь является родителем и текущего узла **N** и бывшего дедушки **G**. Известно, что **G** — чёрный, так как его бывший потомок **P** не мог бы в противном случае быть красным (без нарушения Свойства 4). Тогда цвета **P** и **G** меняются и в результате дерево удовлетворяет Свойству 4 (Оба потомка любого красного узла — чёрные). Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) также остается верным, так как все пути, которые проходят через любой из этих трех узлов, ранее проходили через **G**, поэтому теперь они все проходят через **P**. В каждом случае, из этих трёх узлов только один окрашен в чёрный.

Удаление

При удалении узла с двумя нелистовыми потомками в обычном двоичном дереве поиска мы ищем либо наибольший элемент в его левом поддереве, либо наименьший элемент в его правом поддереве и перемещаем его значение в удаляемый узел. Затем мы удаляем узел, из которого копировали значение. Копирование значения из одного узла в другой не нарушает свойств красно-чёрного дерева, так как структура дерева и цвета узлов не изменяются. Стоит заметить, что новый удаляемый узел не может иметь сразу два дочерних нелистовых узла, так как в противном случае он не будет являться наибольшим/наименьшим элементом. Таким образом, получается, что случай удаления узла, имеющего два нелистовых потомка, сводится к случаю удаления узла, содержащего как максимум один дочерний нелистовой узел. Поэтому дальнейшее описание будет исходить из предположения существования у удаляемого узла не более одного нелистового потомка.

Будем использовать обозначение **M** для удаляемого узла; через **C** обозначим потомка **M**, который также будем называть просто «его потомок». Если **M** имеет нелистового потомка, возьмем его за **C**. В противном случае за **C** возьмем любой из листовых потомков.

Если **M** является красным узлом, заменим его своим потомком **C**, который по определению должен быть чёрным. (Это может произойти только тогда, когда **M** имеет двух листовых потомков, потому что если красный узел **M** имеет чёрного нелистового потомка с одной стороны, а с другой стороны — листового, то число чёрных узлов на обеих сторонах будет различным, таким образом дерево станет недействительным красно-чёрным деревом из-за нарушения Свойства 5.) Все пути через удаляемый узел просто будут содержать на один красный узел меньше, предок и потомок удаляемого узла должны быть чёрными, так что Свойство 3 («Все листья — чёрные») и Свойство 4 («Оба потомка красного узла — чёрные») все ещё сохраняется.

Другим простым является случай, когда **M** — чёрный и **C** — красный. Простое удаление чёрного узла нарушит Свойство 4 («Оба потомка красного узла — чёрные») и Свойство 5 («Всякий простой путь от данного узла до любого листового узла, содержит одинаковое число чёрных узлов»), но если мы перекрасим **С** в чёрный, оба эти свойства сохранятся.

Сложным является случай, когда и **M** и **C** — чёрные. (Это может произойти только тогда, когда удаляется чёрный узел, который имеет два листовых потомка, потому что если чёрный узел **M** имеет чёрного нелистового потомка с одной стороны, а с другой — листового, то число чёрных узлов на обеих сторонах будет различным и дерево станет недействительным красно-чёрным деревом из-за нарушения Свойства 5.) Мы начнём с замены узла **M** своим потомком **C**. Будем называть этого потомка (в своем новом положении) **N**, а его «брата» (другого потомка его нового предка) — **S**. (До этого **S** был «братом» **M**.) На рисунках ниже мы также будем использовать обозначение **P** для нового предка **N** (старого предка **M**), **SL** для левого потомка **S** и **SR** для правого потомка **S** (**S** не может быть листовым узлом, так как если **N** по нашему предположению является чёрным, то поддерево **P**, которое содержит **N**, чёрной высоты два и поэтому другое поддерево **P**, которое содержит **S** должно быть также чёрной высоты два, что не может быть в случае, когда **S** — лист).

Если **N** и его текущий отец чёрные, тогда удаление отца приведет к тому, что пути, которые проходят через **N** будут иметь на один чёрный узел меньше, чем пути, которые не проходят через него. Так как это нарушает свойство 5 (все пути из любого узла к его листовым узлам содержат одинаковое количество чёрных узлов), дерево должно быть перебалансировано. Есть несколько случаев для рассмотрения:

**Случай 1:** **N** — новый корень. В этом случае, все сделано. Мы удалили один чёрный узел из каждого пути и новый корень является чёрным узлом, так что свойства сохранены.

**Случай 2:** **S** — красный. В этом случае мы меняем цвета **P** и **S**, и затем делаем вращение влево вокруг **P**, ставя **S** дедушкой **N**. Нужно заметить, что **P** должен быть чёрным, если он имеет красного потомка. Результирующее поддерево всё равно имеет черных узлов на единицу меньше, поэтому на этом мы ещё не закончили. Теперь **N** имеет чёрного брата и красного отца, поэтому мы можем перейти к шагу 4, 5 или 6. (Его новый брат является чёрным потому, что он был потомком красного **S**.)Далее через **S** будет обозначен новый брат **N**.

**Случай 3:** **P**, **S**, и дети **S'** — чёрные. В этом случае мы просто перекрашиваем **S** в красный. В результате все пути, проходящие через **S**, но не проходящие через **N**, имеют на один чёрный узел меньше. Так как удаление отца **N** приводит к тому, что все пути, проходящие через **N**, содержат на один чёрный узел меньше, то такие действия выравнивают баланс. Тем не менее, все проходящие через **P** пути теперь содержат на один чёрный узел меньше, чем пути, которые через **P** не проходят, поэтому свойство 5 (все пути из любой вершины к её листовым узлам содержат одинаковое количество чёрных узлов) все ещё нарушено. Чтобы это исправить, мы применяем процедуру перебалансировки к **P**, начиная со случая 1.

**Случай 4:** **S** и его дети — чёрные, но **P** — красный. В этом случае мы просто меняем цвета **S** и **P**. Это не влияет на количество чёрных узлов на путях, проходящих через **S**, но добавит один к числу чёрных узлов на путях, проходящих через **N**, восстанавливая тем самым влияние удаленного чёрного узла.

**Случай 5:** **S** — чёрный, левый потомок **S** — красный, правый потомок **S** — чёрный, и **N** является левым потомков своего отца. В этом случае мы вращаем дерево вправо вокруг **S**. Таким образом левый потомок **S** становится его отцом и новым братом **N**. После этого мы меняем цвета у **S** и его нового отца. Все пути по прежнему содержат одинаковое количество чёрных узлов, но теперь у **N** есть чёрный брат с красным правым потомком, и мы переходим к случаю 6. Ни **N**, ни его отец не влияют на эту трансформацию. (Для случая 6 мы обозначим через **S** нового брата **N**.)

**Случай 6:** **S** — чёрный, правый потомок **S** — красный, и **N** является левым потомком своего отца **P**. В этом случае мы вращаем дерево влево вокруг **P**, после чего **S** становится отцом **P** и своего правого потомка. Далее мы меняем местами цвета у **P** и **S** (**P** принимает цвет **S**, **S** принимает цвет **P**), и делаем правого потомка **S** чёрным. Поддерево по прежнему имеет тот же цвет корня, поэтому свойства 4 (Оба потомка каждого красного узла — чёрные) и 5 (все пути из любой вершины к её листовым узлам содержат одинаковое количество чёрных узлов) не нарушаются. Тем не менее, у **N** теперь появился дополнительный чёрный предок: либо **P** стал чёрным, или он был чёрным и **S** был добавлен в качестве чёрного дедушки. Таким образом, проходящие через **N** пути проходят через один дополнительный чёрный узел.

Между тем, если путь не проходит через **N**, то есть 2 возможных варианта:

* Он проходит через нового брата **N**. Тогда, он должен проходить через **S** и **P**, которые просто поменяли цвета и места. Поэтому путь содержит то же количество чёрных узлов.
* Он проходит через нового дядю **N**, правого потомка **S**. Когда-то он проходил через **S**, отца **S** и правого потомка **S** (который был красным), но теперь он проходит только через **S**, который принял на себя цвет своего прежнего родителя, и правого потомка **S**, который был перекрашен из красного в чёрный (Предполагаем, что цвет **S**: чёрный). Эффект заключается в том, что этот путь проходит через такое же количество чёрных узлов.

В любом случае, число чёрных узлов на этих путях не изменится. Поэтому, мы восстановили свойства 4 (Оба потомка каждого красного узла — чёрные) и 5 (все пути из любой вершины к её листовым узлам содержат одинаковое количество чёрных узлов). Белый узел на диаграмме может быть как красным так и чёрным, но должен указывать на тот же цвет как в начале, так и в конце трансформации.

Время работы

Красно-чёрное дерево с N ключами имеет высоту h = O(log N)

Рассмотрим красно-чёрное дерево с высотой h. Так как у красной вершины чёрные дети (по свойству 3) количество красных вершин не больше h2h2. Тогда чёрных вершин не меньше, чем h2−1h2−1.

Т.к в красно-чёрном дереве с чёрной высотой hb количество внутренних вершин не менее , для количества внутренних вершин в дереве N выполняется неравенство:

Прологарифмировав неравенство, имеем:

log(N+1)⩾h/2

2log(N+1)⩾h

h⩽2log(N+1)

Вывод, использование, плюсы и минусы

Красно-чёрные деревья являются одними из наиболее активно используемых на практике самобалансирующихся деревьев поиска. В частности, контейнеры set и map в большинстве реализаций библиотеки STL языка C++, класс TreeMap языка Java, так же, как и многие другие реализации ассоциативного массива в различных библиотеках, основаны на красно-чёрных деревьях.

Красно-чёрные деревья более популярны, чем идеально сбалансированные деревья, т.к. в последних может тратиться слишком много ресурсов на операции удаления из дерева и поддержание необходимой сбалансированности. После вставки или удаления требуется операция перекраски, требующая (O(log *n*) или O(1)) смен цветов (что на практике довольно быстро) и не более чем трёх поворотов дерева (для вставки — не более двух).

Хотя вставка и удаление сложны, их трудоёмкость остается O(log *n*)

Список использованной литературы

<https://en.wikipedia.org/wiki/Red%E2%80%93black_tree>

<https://neerc.ifmo.ru/wiki/index.php?title=%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D0%BE-%D1%87%D0%B5%D1%80%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE>

Код программы и все входные данные – в репозитории: https://github.com/LoDi-bot/ITIS-AiSD-c/tree/main/semestrovaya1