## Splay-дерево

Splay-дерево — это самобалансирующееся бинарное дерево поиска. Дереву не нужно хранить никакой дополнительной информации, что делает его эффективным по памяти.

Вместо этого после каждого обращения, даже поиска, splay-дерево меняет свою структуру, благодаря splay operation, частью которых являются вращения.

Представим, что мы уже построили дерево на https://habrastorage.org/getpro/habr/post_images/c7f/092/d48/c7f092d481acb49c8a0f96178ceb3119.gif ключах и теперь нам нужно отвечать на запросы, лежит ли заданный ключ в дереве. Может так оказаться, что пользователя интересует в основном один ключ, и остальные он запрашивает только время от времени. Если ключ лежит далеко от корня, то  запросов могут отнять https://habrastorage.org/getpro/habr/post_images/076/63d/c3c/07663dc3c790b8c5e111597f71b68abc.gif времени.

Эти деревья не являются перманентно сбалансированными и на отдельных запросах могут работать даже линейное время. Однако, после каждого запроса они меняют свою структуру, что позволяет очень эффективно обрабатывать часто повторяющиеся запросы. Более того, амортизационная стоимость обработки одного запроса у них http://habrastorage.org/getpro/habr/post_images/84c/07b/cc9/84c07bcc99d5fc8ab9086ace521ed96a.gif, что делает splay-деревья хорошей альтернативой для перманентно сбалансированных собратьев.

* Пример использования - сетевой маршрутизатор. Сетевой маршрутизатор получает сетевые пакеты с высокой скоростью от входящих подключений и должен быстро решить, по какому исходящему проводу отправлять каждый пакет, на основе IP-адреса в пакете. Маршрутизатору нужна большая таблица (карта), которую можно использовать для поиска IP-адреса и определения того, какое исходящее соединение использовать. Если IP-адрес использовался один раз, он, вероятно, будет использоваться снова, возможно, много раз. В этой ситуации Splay-деревья могут обеспечить хорошую производительность.

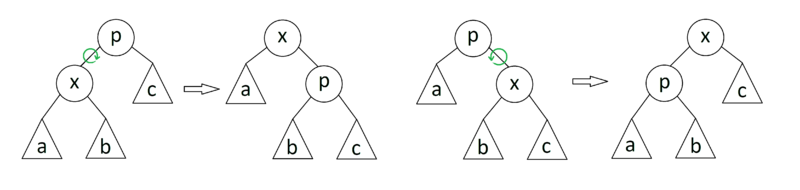
**Операции со splay-деревом**

### Splay (расширение)

Основная операция дерева. Заключается в перемещении вершины в корень при помощи последовательного выполнения трёх операций: Zig, Zig-Zig и Zig-Zag. Обозначим вершину, которую хотим переместить в корень за *x*, её родителя — *p*, а родителя *p* (если существует) — *g*.

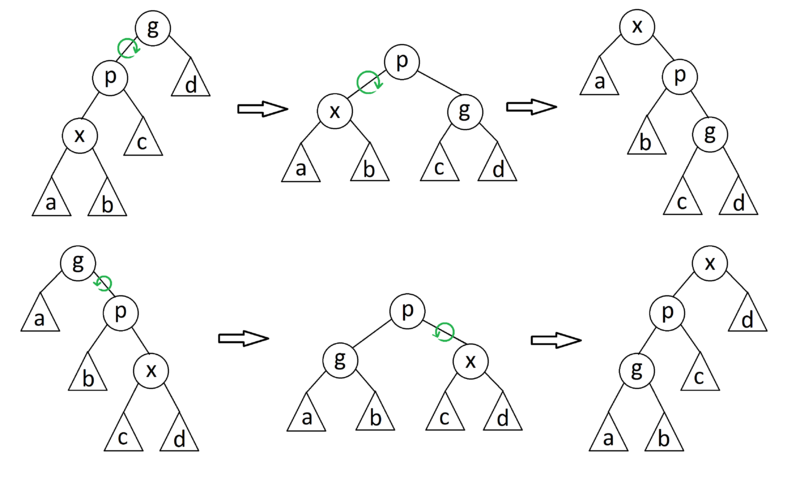
#### **zig**

Если p — корень дерева с сыном x, то совершаем один поворот вокруг ребра (x,p), делая x корнем дерева. Данный случай является крайним и выполняется только один раз в конце, если изначальная глубина x была нечетной.

[](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:%D0%97%D0%B8%D0%B3.png)

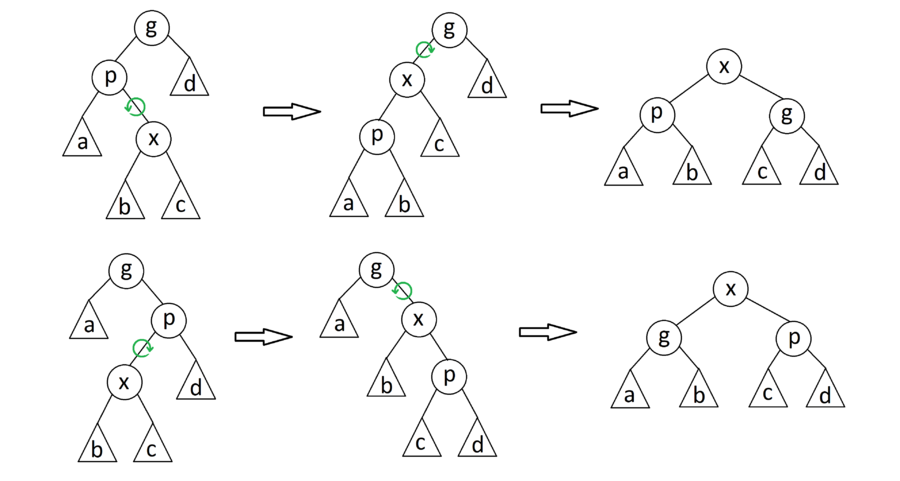
#### **zig-zig**

Если p — не корень дерева, а x и p — оба левые или оба правые дети, то делаем поворот ребра (p,g), где g отец p, а затем поворот ребра (x,p).

[](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:%D0%97%D0%B8%D0%B3_%D0%B7%D0%B8%D0%B3.png)

#### **zig-zag**

Если p — не корень дерева и x — левый ребенок, а p — правый, или наоборот, то делаем поворот вокруг ребра (x,p), а затем поворот нового ребра (x,g), где g — бывший родитель p.

[](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:%D0%97%D0%B8%D0%B3_%D0%B7%D0%B0%D0%B32.png)

Данная операция занимает O(d) времени, где d— длина пути от x до корня.

1. **Search (поиск элемента)** Поиск выполняется как в обычном двоичном дереве поиска. При нахождении элемента запускаем Splay для него.
2. **Split (разделение дерева на две части)** Для разделения дерева найдем наименьший элемент, больший или равный x, и сделаем для него Splay. После этого отрезаем у корня левого ребёнка и возвращаем 2 получившихся дерева.
3. **Merge (объединение двух деревьев)** У нас есть два дерева tree1 и tree2, причём подразумевается, что все элементы первого дерева меньше элементов второго. Запускаем splay от самого большого элемента в дереве tree1 (пусть это элемент i). После этого корень tree1 содержит элемент i, при этом у него нет правого ребёнка. Делаем tree2 правым поддеревом i и возвращаем полученное дерево.
4. **Insert (добавление элемента)** Вставка происходит как в обычном бинарном дереве поиска, после, запускаем Split() от добавляемого элемента и подвешиваем получившиеся деревья за него.
5. **Remove (удаление элемента)** Находим элемент в дереве, делаем Splay для него, делаем текущим деревом Merge его детей.

* Сложность операций и требование по памяти:
* Затраты по памяти - O(n)
* Splay - O(log(n))
* Search - O(log(n))
* Split - O(log(n))
* Merge - O(log(n))
* Insert - O(log(n))
* Delete - O(log(n))

Расширяющееся дерево предоставляет самоизменяющуюся структуру — структуру, характеризующуюся тенденцией хранить узлы, к которым часто происходит обращение, вблизи верхушки дерева, в то время как узлы, к которым обращение происходит редко, перемещаются ближе к листьям. Таким образом время обращения к часто посещаемым узлам будет меньше, а время обращения к редко посещаемым узлам — больше среднего.

Расширяющееся дерево не обладает никакими явными функциями балансировки, но процесс скоса узлов к корню способствует поддержанию дерева в сбалансированном виде.

Источники:

[Splay-дерево — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE)

[Splay-деревья / Хабр (habr.com)](https://habr.com/ru/companies/JetBrains-education/articles/210296/)

[Algorithms-and-Data-Structures-2021/semester-work-splay-tree (github.com)](https://github.com/Algorithms-and-Data-Structures-2021/semester-work-splay-tree?ysclid=lp9ppqm4cg52559515)

[Splay-дерево — Викиконспекты (ifmo.ru)](https://neerc.ifmo.ru/wiki/index.php?title=Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE)