Отчёт 11. Сбалансированные деревья

- Красно-черные деревья
- Splay-деревья
- 2-3-деревья
- B-деревья
- Декартовы деревья

Описать достоинства и недостатки перечисленных деревьев.

Сбалансированное дерево — это дерево поиска, которое не просто поддерживает порядок между узлами. Он также контролирует свою высоту, следя за тем, чтобы она оставалась после вставки или удаления.

Для этого сбалансированное дерево должно заново сбалансировать себя после добавления или удаления узла. Это приводит к вычислительным затратам и усложняет алгоритмы вставки и удаления. Однако это цена, которую мы готовы заплатить за дерево поиска логарифмической высоты с быстрыми операциями поиска, вставки и удаления.

Существует несколько типов таких деревьев. Они требуют, чтобы все их узлы были сбалансированы, но понятие баланса отличается от типа к типу.

Красно-черное дерево

Как бинарное дерево, красно-черное обладает свойствами:

1) Оба поддерева являются бинарными деревьями поиска.
2) Для каждого узла с ключом k выполняется критерий упорядочения

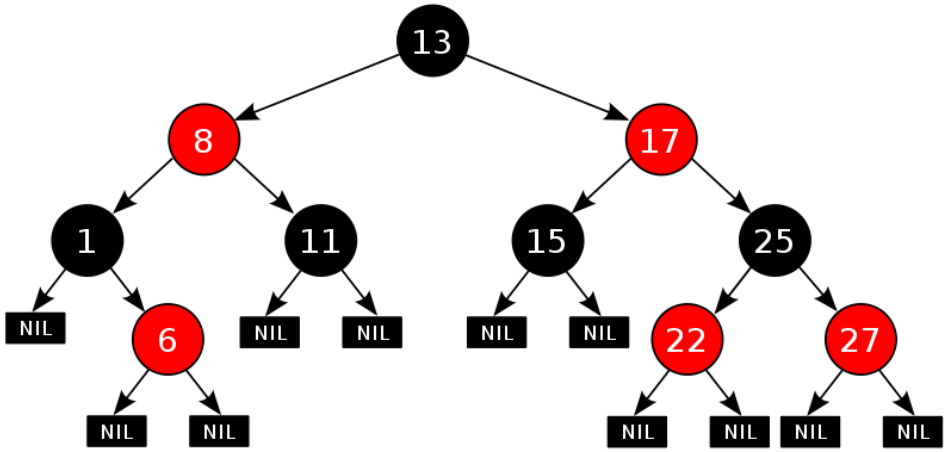
ключи всех левых потомков <= k < ключи всех правых потомков

(в других определениях дубликаты должны располагаться с правой стороны либо вообще отсутствовать).
Это неравенство должно быть истинным для всех потомков узла, а не только его дочерних узлов.

Свойства:

1. Каждый узел окрашен либо в красный, либо в черный цвет (в структуре данных узла появляется дополнительное поле – бит цвета).
2. Корень окрашен в черный цвет.
3. Листья (так называемые NULL-узлы) окрашены в черный цвет.
4. Каждый красный узел должен иметь два черных дочерних узла. Нужно отметить, что у черного узла могут быть черные дочерние узлы. Красные узлы в качестве дочерних могут иметь только черные.

5. Пути от узла к его листьям должны содержать одинаковое количество черных узлов (это черная высота). Здесь мы несколько меняем определение листа, и называем так специальные null-вершины, которые замещают отсутствующих сыновей. Будем считать такие вершины черными.



Преимущества:

- Красно-черные деревья уравновешивают уровень дерева.
- Красно-черное дерево получает некоторый запас для структурирования дерева, восстанавливая уровень дерева.
- Временная сложность поисковой активности составляет O(log n).
- Он имеет такие же низкие константы в широком диапазоне ситуаций.

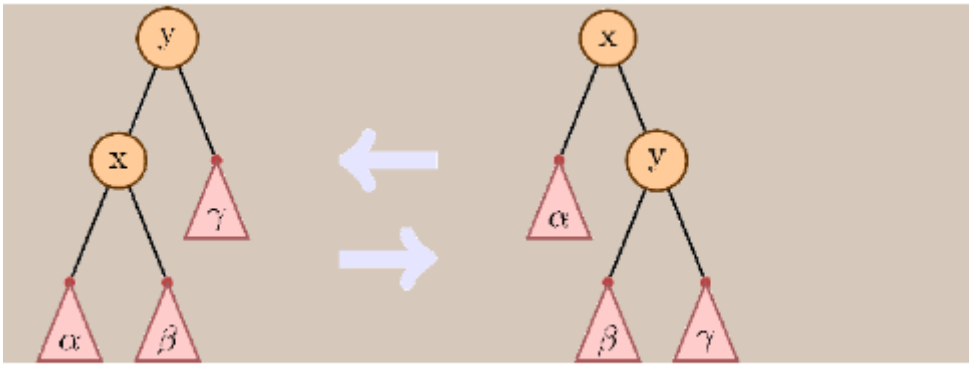
Недостатки:

- Сложно использовать из-за всех крайних случаев активности; как правило, вам нужно использовать стандартное выполнение библиотеки (например, набор STL в C++ и т. д.) вместо того, чтобы выполнять его самостоятельно без какой-либо подготовки.
- Если вы планируете сформировать дерево только один раз, а затем просто выполнять операции чтения, деревья AVL предлагают лучшее исполнение.
- Поскольку B-деревья могут иметь различное количество дочерних элементов, их часто предпочитают красно-черным деревьям за упорядочивание и размещение большого количества данных на пластинах, поскольку они могут быть несколько поверхностными, чтобы ограничить круговые задачи.

Splay-деревья

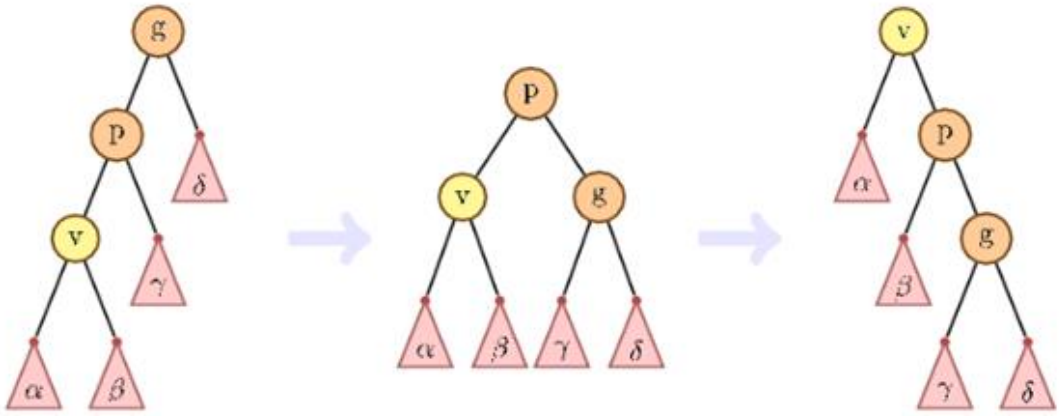
Splay-дерево — это самобалансирующееся бинарное дерево поиска. Дереву не нужно хранить никакой дополнительной информации, что делает его эффективным по памяти. После каждого обращения, даже поиска, splay-дерево меняет свою структуру.

Основная эвристика splay-дерева — move-to-root. После обращения к любой вершине, она поднимается в корень. Подъем реализуется через повороты вершин. За один поворот, можно поменять местами родителя с ребенком, как показано на рисунке ниже.

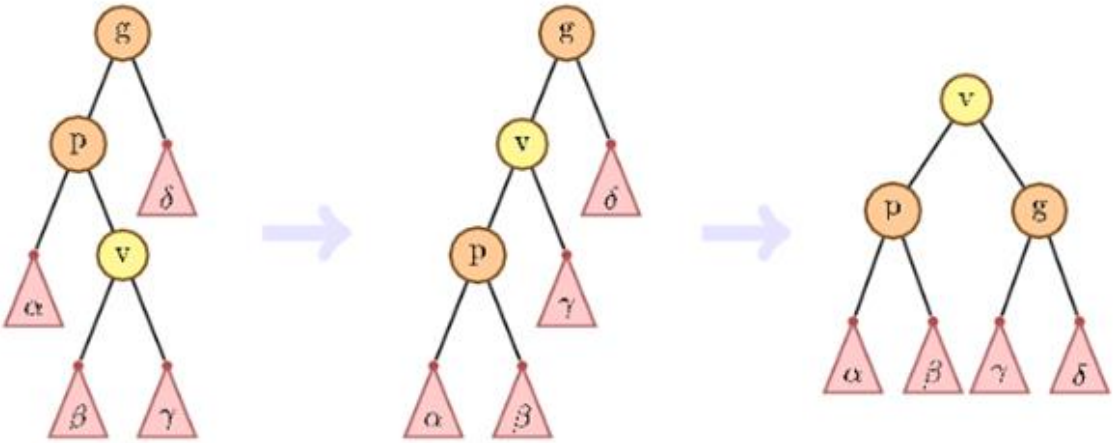


Но просто поворачивать вершину, пока она не станет корнем, недостаточно. Хитрость splay-дерева в том, что при продвижении вершины вверх, расстояние до корня сокращается не только для поднимаемой вершины, но и для всех ее потомков в текущих поддеревьях. Для этого используется техника zig-zig и zig-zag поворотов.

Основная идея zig-zig и zig-zag поворотов, рассмотреть путь от дедушки к ребенку. Если путь идет только по левым детям или только по правым, то такая ситуация называется zig-zig. Как ее обрабатывать показано на рисунке ниже. Сначала повернуть родителя, потом ребенка.



В противном случае, мы сначала меняем ребенка с текущим родителем, потом с новым.



Если у вершины дедушки нет, делаем обычный поворот:

Splay-деревья ведут себя хорошо при последовательной обработке большого набора запросов с произвольным распределением. Они экономичны в плане количества информации, которую требуется хранить в узлах.

Преимущества:

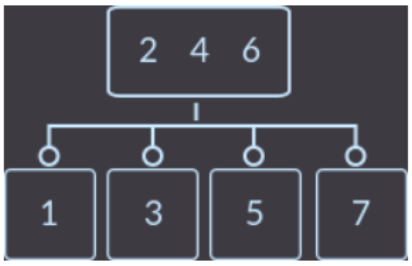
- Сопоставимая производительность: производительность в среднем случае такая же эффективная, как и у других деревьев.
- Небольшой объем памяти: деревьям Splay не нужно хранить какие-либо бухгалтерские данные.
- Сбалансировано время операций

Недостатки:

- Могут вырождаться в линеный список
- Например, это будет иметь место после доступа ко всем *n* элементам в порядке неубывания. Поскольку высота дерева соответствует наихудшему времени доступа, это означает, что фактическая стоимость одной операции может быть высокой. Однако амортизированная стоимость доступа в этом наихудшем случае является логарифмической, O (log *n*). Кроме того, ожидаемая стоимость доступа может быть уменьшена до O (log *n*) с помощью рандомизированного варианта.[4]
- Представление splay-деревьев может измениться, даже если к ним осуществляется доступ "только для чтения" (т. Е. с помощью операций *поиска*). Это усложняет использование таких splay деревьев в многопоточной среде. В частности, требуется дополнительное управление, если нескольким потокам разрешено выполнять операции *поиска* одновременно. Это также делает их непригодными для общего использования в чисто функциональном программировании, хотя даже там они могут использоваться ограниченными способами для реализации очередей приоритетов.

2-3 деревья

2-3-дерево — структура данных, страницы которой могут содержать только 2-вершины (вершины с одним полем и 2 детьми) и 3-вершины (вершины с 2 полями и 3 детьми). Листовые вершины являются исключением — у них нет детей (но может быть одно или два поля). 2-3-деревья сбалансированы, то есть, каждое левое, правое, и центральное поддерево имеет одну и ту же высоту, и, таким образом, содержат равные (или почти равные) объемы данных.



Свойства:

- Все нелистовые вершины содержат одно поле и 2 поддерева или 2 поля и 3 поддерева.
- Все листовые вершины находятся на одном уровне (на нижнем уровне) и содержат 1 или 2 поля.
- Все данные отсортированы (по принципу двоичного дерева поиска).
- Нелистовые вершины содержат одно или два поля, указывающие на диапазон значений в их поддеревьях. Значение первого поля строго больше наибольшего значения в левом поддереве и меньше или равно наименьшему значению в правом поддереве (или в центральном поддереве, если это 3-вершина); аналогично, значение второго поля (если оно есть) строго больше наибольшего значения в центральном поддереве и меньше или равно, чем наименьшее значение в правом поддереве. Эти нелистовые вершины используются для направления функции поиска к нужному поддереву и, в конечном итоге, к нужному листу. (*прим. Это свойство не будет выполняться, если у нас есть одинаковые ключи. Поэтому возможна ситуация, когда равные ключи находятся в левом и правом поддеревьях одновременно, тогда ключ в нелистовой вершине будет совпадать с этими ключами. Это никак не сказывается на правильности работы и производительности алгоритма.*).

<u>Преимущества:</u>

- Для наборов данных, которые помещаются в основную память, 2-3 дерева и BST обычно являются лучшим выбором (хотя были некоторые исследования, показывающие, что B-деревья низкого порядка могут превосходить BST в основной памяти из-за эффектов кэша.)

- Если вы планируете хранить огромный объем данных, которые не могут поместиться в основную память, B-деревья - отличный выбор для структуры данных.

Недостатки:

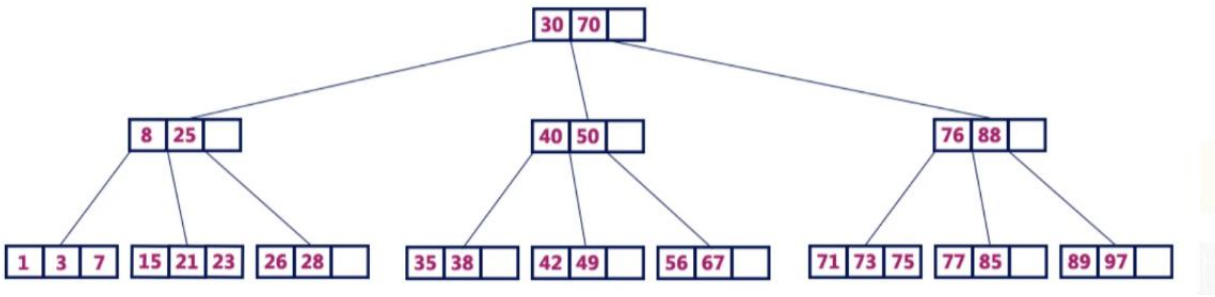
- В основной памяти B-деревья с очень большим коэффициентом ветвления будут медленнее, чем BSTS или 2-3 дерева, потому что для каждой вставки или удаления B-дерева может потребоваться большое количество переназначений указателей.

B-дерево

B-дерево (англ. *B-tree*) — сильноветвящееся сбалансированное дерево поиска, позволяющее проводить поиск, добавление и удаление элементов за $O(\log n)$. B-дерево с n узлами имеет высоту $O(\log n)$.

Количество детей узлов может быть от нескольких до тысяч (обычно степень ветвления B-дерева определяется характеристиками устройства (дисков), на котором производится работа с деревом).

B-деревья также могут использоваться для реализации многих операций над динамическими множествами за время $O(\log n)$



tt — параметр дерева, называемый *минимальной степенью* B-дерева, не меньший 2.

Свойства:

- Каждый узел, кроме корня, содержит не менее $t-1$ ключей, и каждый внутренний узел имеет по меньшей мере t дочерних узлов. Если дерево не является пустым, корень должен содержать как минимум один ключ.
- Каждый узел, кроме корня, содержит не более $2t-1$ ключей и не более чем $2t$ сыновей во внутренних узлах
- Корень содержит от 1 до $2t-1$ ключей, если дерево не пусто и от 2 до $2t$ детей при высоте большей 0.

- Каждый узел дерева, кроме листьев, содержащий ключи k1,...,knk1,...,kn, имеет n+1n+1 сына. ii-й сын содержит ключи из отрезка [ki−1;ki],k0=−∞,kn+1=∞[ki−1;ki],k0=−∞,kn+1=∞.
- Ключи в каждом узле упорядочены по неубыванию.
- Все листья находятся на одном уровне.

Преимущества:

- Во всех случаях полезное использование памяти для хранения индексов составляет свыше 50%. Обеспечивается динамическое распределение и использование памяти. С ростом степени полезного использования памяти не происходит снижения качества обслуживания пользовательских запросов.
- Произвольный доступ к записи реализуется посредством небольшого количества подопераций (обращений к физическим блокам) и по эффективности сопоставим с методами перемешивания.
- В среднем достаточно эффективно реализуются операции включения и добавления записей. При этом сохраняется естественный порядок ключей с целью последовательной обработки, а также соответствующий баланс дерева для обеспечения быстрой произвольной выборки.
- Упорядоченность по ключу обеспечивает возможность эффективной пакетной обработки.

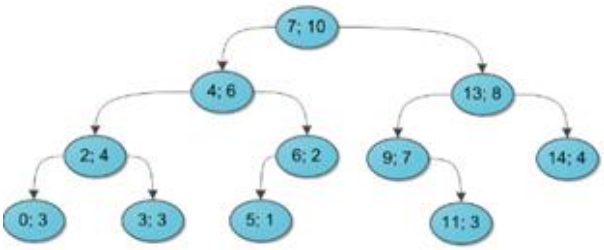
Недостатки:

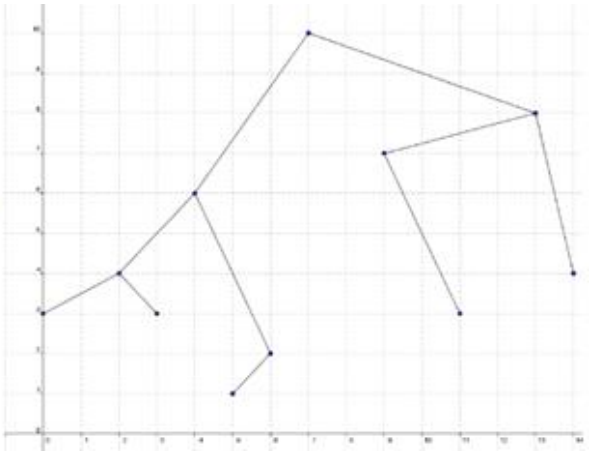
- Основной недостаток B-деревьев состоит в отсутствии для них средств выборки данных по вторичному ключу.

Декартово дерево

Декартово дерево — красивая и легко реализующаяся структура данных, которая с минимальными усилиями позволит вам производить многие скоростные операции над массивами ваших данных.

Почему дерево называется декартовым? Это сразу станет ясно, как только мы попробуем его нарисовать. Возьмем какой-нибудь набор пар «ключ-приоритет» и расставим на координатной сетке соответствующие точки (x, y). А потом соединим соответствующие вершины линиями, образуя дерево. Таким образом, декартово дерево отлично укладывается на плоскости благодаря своим ограничениям, а два его основных параметра — ключ и приоритет — в некотором смысле, координаты. Результат построения показан на рисунке: слева в стандартной нотации дерева, справа — на декартовой плоскости.





- обладает почти гарантированно логарифмической высотой относительно количества своих вершин;
- позволяет за логарифмическое время искать любой ключ в дереве, добавлять его и удалять;
- исходный код всех её методов не превышает 20 строк, они легко понимаются и в них крайне сложно ошибиться.

Преимущества:

- В том применении, которое мы рассматриваем (мы будем рассматривать дерамиды, поскольку декартово дерево - это фактически более общая структура данных), X'ы являются ключами (и одновременно значениями, хранящимися в структуре данных), а Y'и - называются приоритетами. Если бы приоритетов не было, то было бы обычное бинарное дерево поиска по X, и заданному набору X'ов могло бы соответствовать много деревьев, некоторые из которых являются вырожденными (например, в виде цепочки), а потому чрезвычайно медленными (основные операции выполнялись бы за O (N)).
- В то же время, приоритеты позволяют однозначно указать дерево, которое будет построено (разумеется, не зависящее от порядка добавления элементов) (это доказывается соответствующей теоремой).
- Проще реализуется, по сравнению, например, с настоящими самобалансирующимися деревьями вроде красно-черного.
- Хорошо ведет себя «в среднем», если ключи *y* раздать случайно.

Недостатки:

- Большие накладные расходы на хранение: вместе с каждым элементом хранятся два-три указателя и случайный ключ *y*.
- Скорость доступа $O(n)$ в худшем, хотя и маловероятном, случае. Поэтому декартово дерево недопустимо, например, в ядрах ОС.

Источники:

1) https://habr.com/ru/post/66926/
2) https://habr.com/ru/post/330644/
3) https://habr.com/ru/post/303374/
4) https://habr.com/ru/post/114154/
5) https://neerc.ifmo.ru/wiki/index.php?title=B-дерево
6) https://otus.ru/nest/post/1801/
7) https://habr.com/ru/post/101818/