1.Красно-чёрное дерево

<https://ru.wikipedia.org/wiki/Красно-чёрное_дерево#:~:text=Красно-чёрное%20дерево%20(англ.%20red-black%20tree%2C,добавление%2C%20удаление%20и%20поиск%20узла>

Красно-чёрное дерево — [двоичное дерево поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0), в котором каждый узел имеет атрибут *цвета*. При этом:

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

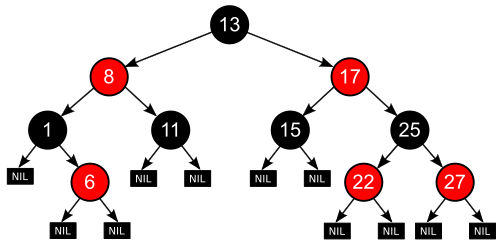
Благодаря этим ограничениям путь от корня до самого дальнего листа не более чем вдвое длиннее, чем до самого ближнего, и дерево примерно сбалансировано. Операции вставки, удаления и поиска требуют в худшем случае времени, пропорционального длине дерева, что позволяет красно-чёрным деревьям в худшем случае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

[ОТРЕДАКТИРОВАНО] Красно-чёрное дерево используется для организации сравнимых данных, таких как фрагменты текста или числа. [Листовые узлы](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)#%D0%9B%D0%B8%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D1%83%D0%B7%D0%BB%D1%8B) 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.



BigO:

Вставка: O(log2n)

Удаление: O(log2n)

Поиск: O(log2n)

2. 2-3 дерево

<https://ru.wikipedia.org/wiki/2-3-дерево>

**2-3 дерево** — [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), являющаяся [B-деревом](https://ru.wikipedia.org/wiki/B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), каждый узел (страница) которого имеет либо два потомка и одно поле, либо три потомка и два поля. Листовые вершины являются исключением — у них нет детей, но есть одно или два поля. 2-3 деревья сбалансированы, то есть все листовые вершины находятся на одной высоте от корня дерева.

[ОТРЕДАКТИРОВАНО] Это дерево используется для организации сравнимых данных, таких как фрагменты текста или числа. [Листовые узлы](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)#%D0%9B%D0%B8%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D1%83%D0%B7%D0%BB%D1%8B) 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. (очень схожее с красным белым, только 2-3 дерево проще понять)



BigO:

Вставка: O(log2n)

Удаление: O(log2n)

Поиск: O(log2n)

3. Расширяющееся дерево?

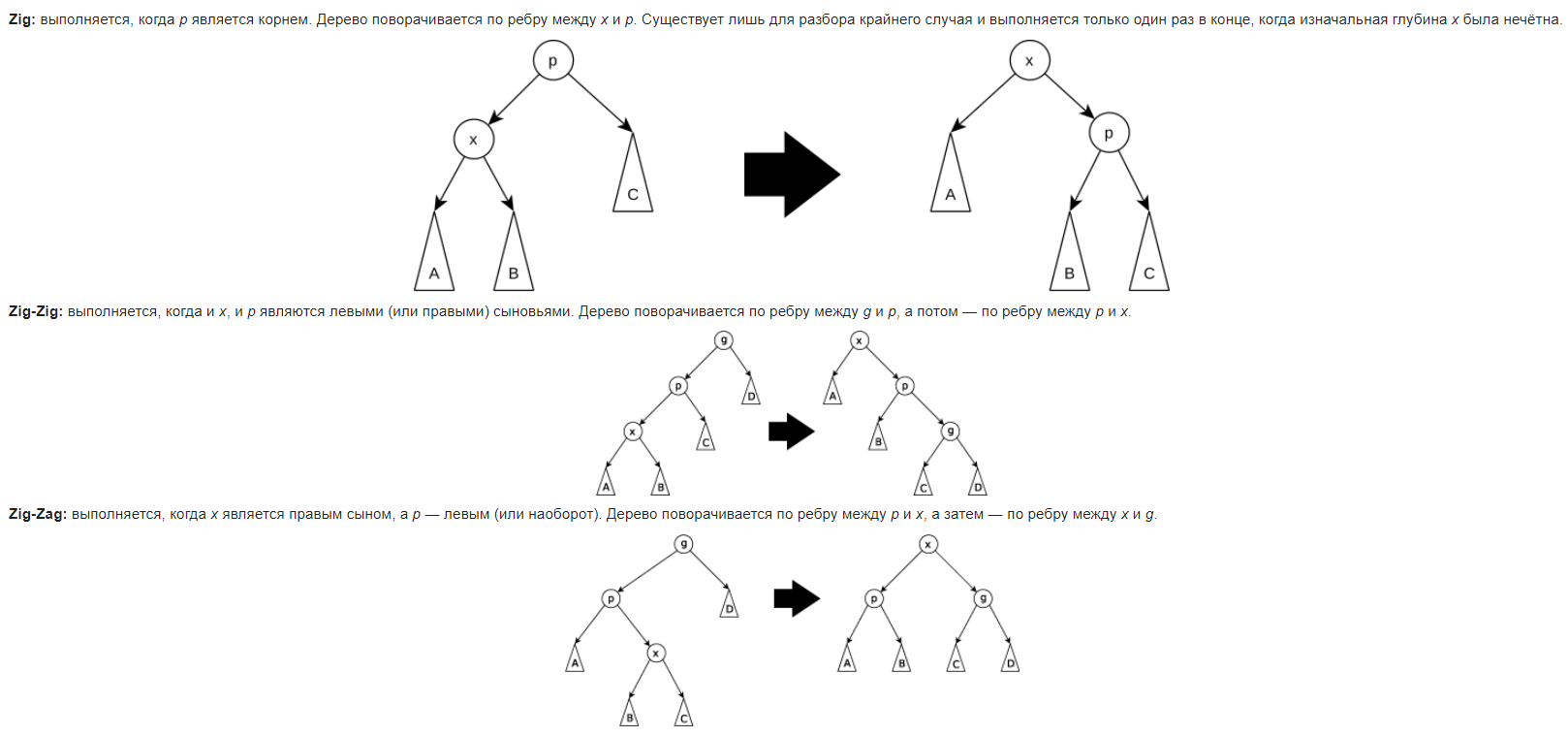
<https://ru.wikipedia.org/wiki/Splay-дерево>

**Расширяющееся** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *splay tree*) или **косое дерево** является [двоичным деревом поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0), в котором поддерживается свойство сбалансированности. Это дерево принадлежит классу «саморегулирующихся деревьев», которые поддерживают необходимый баланс ветвления дерева, чтобы обеспечить выполнение операций поиска, добавления и удаления за логарифмическое время от числа хранимых элементов. Это реализуется без использования каких-либо дополнительных полей в узлах дерева (как, например, в [Красно-чёрных деревьях](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D0%BE-%D1%87%D1%91%D1%80%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) или [АВЛ-деревьях](https://ru.wikipedia.org/wiki/%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), где в вершинах хранится, соответственно, цвет вершины и глубина поддерева). Вместо этого «расширяющие операции» (splay operation), частью которых являются вращения, выполняются при каждом обращении к дереву.

[Учётная стоимость](https://ru.wikipedia.org/wiki/%D0%90%D0%BC%D0%BE%D1%80%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7) в расчёте на одну операцию с деревом составляет O(log⁡n).

Расширяющееся дерево придумали [Роберт Тарьян](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D1%80%D1%8C%D1%8F%D0%BD,_%D0%A0%D0%BE%D0%B1%D0%B5%D1%80%D1%82) и [Даниель Слейтор](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%B0%D0%BD%D0%B8%D0%B5%D0%BB%D1%8C_%D0%A1%D0%BB%D0%B5%D0%B9%D1%82%D0%BE%D1%80&action=edit&redlink=1) в 1983 году.

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*.



[ОТРЕДАКТИРОВАНО] Используется в кэшировании

BigO:

Вставка: O(log n)

Удаление: O(log n)

Поиск: O(log n)

4. Декартово дерево

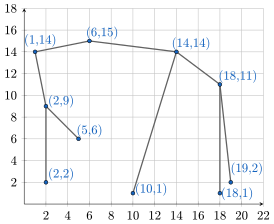
<https://ru.wikipedia.org/wiki/Декартово_дерево>

**Дека́ртово де́рево**, **дуча**, **дерамида** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *treap* от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *tree* «дерево» + [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *heap* «куча») — это структура данных, сочетающая в себе [двоичное дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) и [двоичную кучу](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D0%BA%D1%83%D1%87%D0%B0). Хранит пары *(x, y)*, где для ***ключа*** *x* служит [бинарным деревом поиска](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0), а для ***приоритета*** *y* — двоичной кучей.[[1]](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE#cite_note-_945ba8bb72f78587-1)

Декартово дерево впервые было описано Сесилией Арагон и Раймундом Зиделем в 1989.[[2]](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%B2%D0%BE_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE#cite_note-paper89-2)

Декартово дерево не является [самобалансирующимся](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B0%D0%BC%D0%BE%D0%B1%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D1%83%D1%8E%D1%89%D0%B5%D0%B5%D1%81%D1%8F_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE&action=edit&redlink=1) в обычном смысле, и применяют его по следующим причинам:

* Проще реализуется по сравнению, например, с настоящими самобалансирующимися деревьями вроде [красно-чёрного](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D0%BE-%D1%87%D1%91%D1%80%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE).
* Хорошо ведёт себя «в среднем», если ключи *y* раздать случайно.
* Типичная для сортирующего дерева операция «разделить по ключу *x* на „меньше *x*0“ и „не меньше *x*0“» работает за *O*(*h*), где *h* — высота дерева. На красно-чёрных деревьях придётся восстанавливать балансировку и окраску узлов.
* 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](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B).



[ОТРЕДАКТИРОВАНО] Используется для хранения изображений

BigO:

Вставка: O(log n)

Удаление: O(log n)

Поиск: O(n)

5. M-арное дерево

<https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/M-ary_tree>

В [теории графов](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/Graph_theory), ***m*-арное дерево** (для неотрицательных целых чисел *m*) (также известное как ***n*-арное**, ***k*-арное** или ***k*-way** дерево) - это [древовидное](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/Arborescence_(graph_theory)) (или, для некоторых авторов, [упорядоченное дерево](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/Ordered_tree))[[1]](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/M-ary_tree#cite_note-1)[[2]](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/M-ary_tree#cite_note-2), в котором каждый узел имеет не более *m* дочерних элементов. [Двоичное дерево](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/Binary_tree) является важным случаем, когда *m* = 2; аналогично, [троичное дерево](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/Ternary_tree) - это то, где *m* = 3.

* **Полное *m*-арное дерево** - это *m*-арное дерево, где внутри каждого уровня каждый узел имеет 0 или *m* дочерних элементов.
* **Полное *м*-арное дерево**[[3]](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/M-ary_tree#cite_note-3)[[4]](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/M-ary_tree#cite_note-4) (или, реже, **совершенное *м*-арное дерево**[[5]](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/M-ary_tree#cite_note-5)) - это полное *м*-арное дерево, в котором все [листовые узлы](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/Leaf_node) находятся на одинаковой глубине.
* Для *m*- ary дерева высотой *h* верхняя граница максимального количества листьев равна mh.
* Высота *h m*-ary дерева не включает корневой узел, при этом дерево, содержащее только корневой узел, имеет высоту 0.
* Высота дерева равна максимальной глубине *D* любого узла в дереве.
* Общее количество узлов N в полном *m*-ary дереве равно ∑i=0hmi=mh+1−1m−1, а высота *h* равна

mh+1−1m−1≥N>mh−1m−1mh+1≥(m−1)⋅N+1>mhh+1≥logm⁡((m−1)⋅N+1)>hh≥⌈logm⁡((m−1)⋅N+1)−1⌉.По определению Big-Ω, максимальная глубинаD=h≥⌈logm⁡((m−1)⋅N+1)−1⌉=O(logm⁡n)=O(log⁡n/log⁡m).

* Высота полного *m*-ary дерева с *n* узлами равна ⌊logm⁡((m−1)⋅n)⌋.
* Общее количество возможных *m*-ary деревьев с *n* узлами равно Cn=1(m−1)n+1⋅(m⋅nn) (что является [каталонским числом](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/Catalan_number)).[[6]](https://translated.turbopages.org/proxy_u/en-ru.ru.9f722ecb-671726de-73d6703e-74722d776562/https/en.wikipedia.org/wiki/M-ary_tree#cite_note-6)

[ОТРЕДАКТИРОВАНО] Обыкновенная файловая система

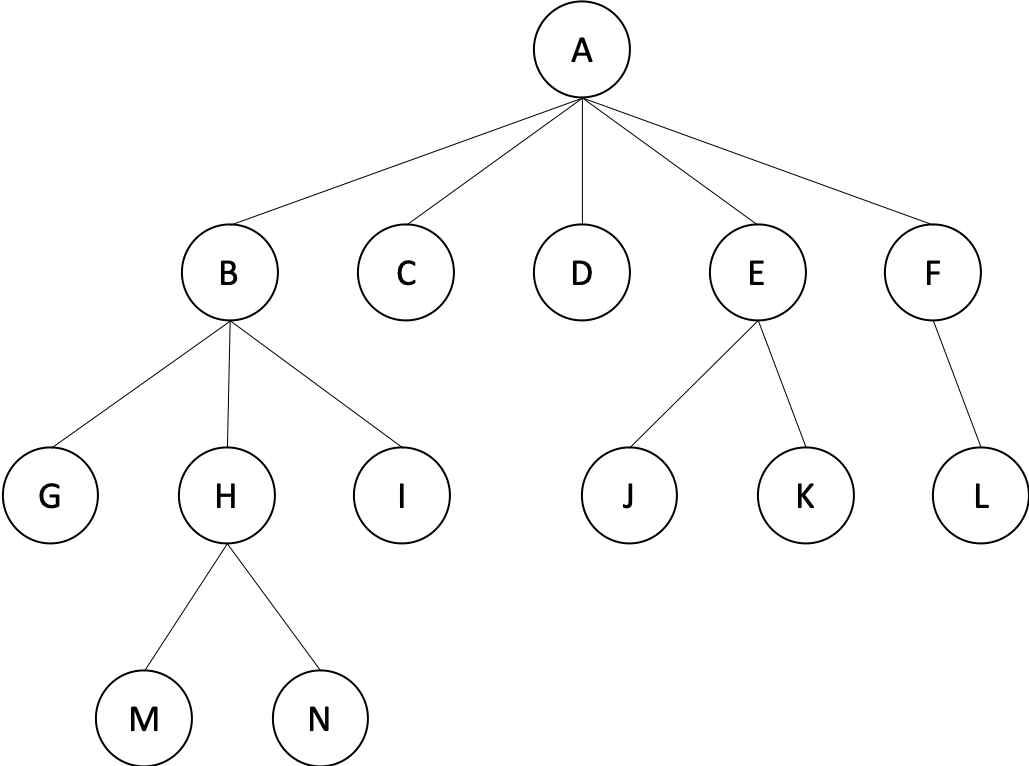
BigO:

Вставка: O(n)

Удаление: O(n)

Поиск: O(n)

6. B-дерево



<https://ru.wikipedia.org/wiki/B-дерево>

**B-дерево** — [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), дерево поиска. С точки зрения внешнего логического представления — [сбалансированное](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), сильно ветвистое [дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)). Часто используется для хранения данных во внешней памяти[[*источник не указан 599 дней*](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B8_%D0%BD%D0%B0_%D0%B8%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8)].

Использование B-деревьев впервые было предложено Р. Бэйером ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *R. Bayer*) и Э. МакКрейтом ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *E. McCreight*) в [1970 году](https://ru.wikipedia.org/wiki/1970_%D0%B3%D0%BE%D0%B4).

*Сбалансированность* означает, что длины любых двух путей от корня до листьев различаются не более, чем на единицу.

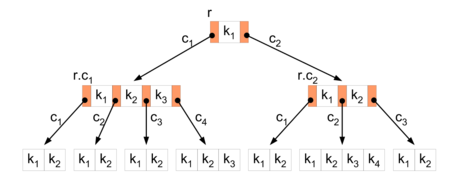
*Ветвистость дерева* — это свойство каждого узла дерева ссылаться на большое число узлов-потомков.

С точки зрения физической организации B-дерево представляется как мультисписочная структура страниц памяти, то есть каждому узлу дерева соответствует блок памяти (страница). Внутренние и листовые страницы обычно имеют разную структуру.

Структура B-дерева применяется для организации [индексов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) во многих современных [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94).

B-дерево может применяться для структурирования (индексирования) информации на жёстком диске (как правило, метаданных). Время доступа к произвольному блоку на жёстком диске очень велико (порядка миллисекунд), поскольку оно определяется скоростью вращения диска и перемещения головок. Поэтому важно уменьшить количество узлов, просматриваемых при каждой операции. Использование поиска по [списку](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) каждый раз для нахождения случайного блока могло бы привести к чрезмерному количеству обращений к диску вследствие необходимости последовательного прохода по всем его элементам, предшествующим заданному, тогда как поиск в B-дереве, благодаря свойствам сбалансированности и высокой ветвистости, позволяет значительно сократить количество таких операций.

Относительно простая реализация алгоритмов и существование готовых библиотек (в том числе для [C](https://ru.wikipedia.org/wiki/C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))) для работы со структурой B-дерева обеспечивают популярность применения такой организации памяти в самых разнообразных программах, работающих с большими объёмами данных.



[ОТРЕДАКТИРОВАНО] Используется в СУБД

BigO:

Вставка: O(log2n)

Удаление: O(logn)

Поиск: O(log2n)

~~~~~~~~~~Выполнил Стич Назар ИВТ-22~~~~~~~~~~