ЛЕКЦИЯ 08
AVL-ДЕРЕВЬЯ И КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

ЛЕКТОР ФУРМАВНИН С.А.

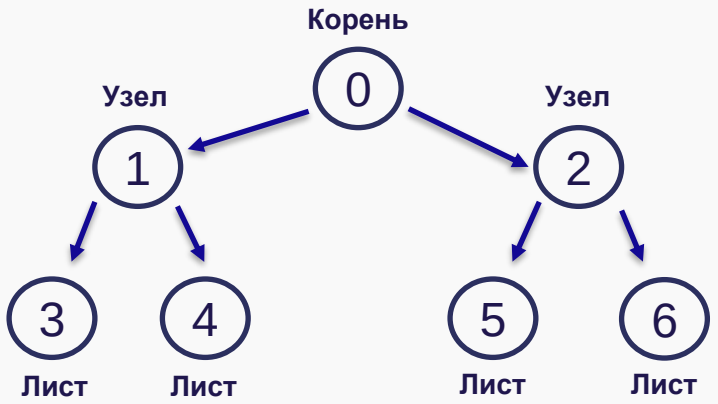
ПРЕЗЕНТАЦИЯ ПОДГОТОВЛЕНА СТУДЕНТОМ ГРУППЫ 31ПРД4310 МОШХОЕВЫМ А.С.

ВВЕДЕНИЕ В СТРУКТУРЫ ДАННЫХ ОСНОВЫ ДЕРЕВЬЕВ

Дерево — это абстрактная структура данных, которая представляет собой иерархическую организацию элементов, называемых узлами. Каждый узел может содержать данные и ссылки на другие узлы. Основные компоненты дерева:

Узлы: Основные элементы дерева, которые могут содержать данные и ссылки на другие узлы.
Корень: Узел, который находится на верхнем уровне дерева. Он не имеет родительского узла и является начальной точкой для всех операций с деревом.
Листья: Узлы, которые не имеют дочерних узлов. Они находятся на самом нижнем уровне дерева и представляют собой конечные элементы структуры.

Деревья используются для представления иерархических данных, таких как файловые системы, организационные структуры и многие другие.



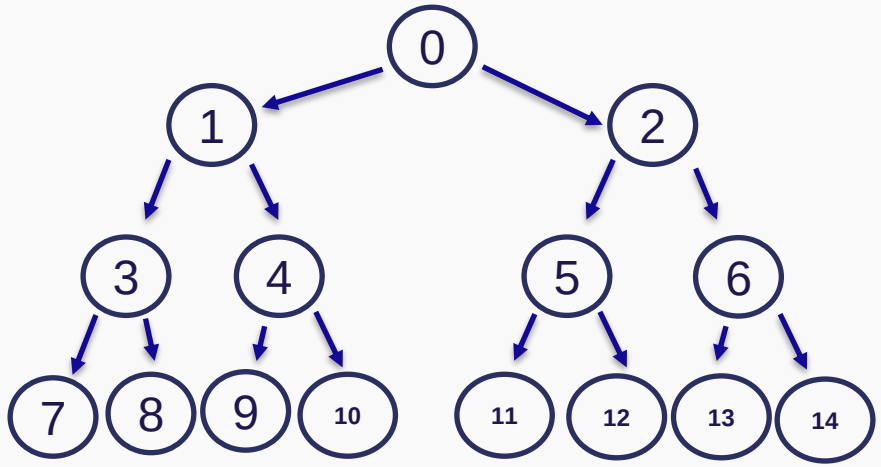
СВОЙСТВА БИНАРНЫХ ДЕРЕВЬЕВ

Бинарное дерево — это особый вид дерева, в котором каждый узел может иметь не более двух дочерних узлов, которые обычно называются левым и правым. Основные свойства бинарных деревьев:

Высота: Высота бинарного дерева определяется как максимальное количество узлов на пути от корня до самого глубокого листа. Высота дерева влияет на эффективность операций поиска, вставки и удаления.

Сбалансированность: Сбалансированные бинарные деревья, такие как **AVL-деревья и красно-чёрные деревья**, обеспечивают логарифмическое время выполнения операций, что делает их более эффективными для работы с динамическими наборами данных.

Обход: Существует несколько способов обхода бинарного дерева, включая прямой (**preorder**), симметричный (**inorder**) и обратный (**postorder**) обход, каждый из которых имеет свои применения.



ВИДЫ ДЕРЕВЬЕВ

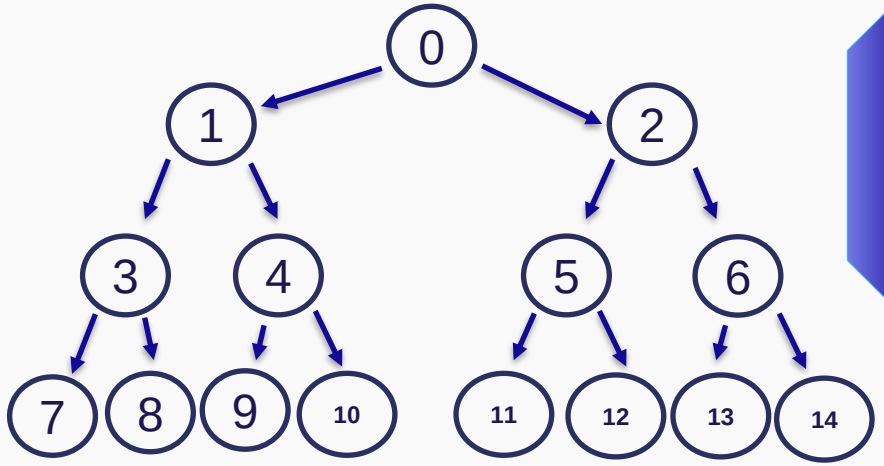
Сбалансированные деревья, такие как **AVL-деревья** и **красно-чёрные деревья**, используются в компьютерных науках для обеспечения эффективной работы с динамическими наборами данных. Вот некоторые из основных причин, почему они нужны:

1. Эффективность операций

- Оба типа деревьев обеспечивают логарифмическое время выполнения основных операций (поиск, вставка, удаление) — O(log n). Это значительно быстрее, чем в несбалансированных бинарных деревьях, где время выполнения может достигать O(n) в худшем случае.

2. Поддержка динамических наборов данных

- **AVL-деревья** и **красно-чёрные деревья** позволяют эффективно управлять динамическими наборами данных, где элементы могут часто добавляться или удаляться. Они автоматически поддерживают балансировку, что предотвращает ухудшение производительности.



ВИДЫ ДЕРЕВЬЕВ

3. Строгая балансировка (AVL) и гибкая балансировка (красно-чёрные)

AVL-деревья обеспечивают более строгую балансировку, что делает их более эффективными для операций поиска.

Красно-чёрные деревья, в свою очередь, обеспечивают более гибкую балансировку, что делает их более эффективными для операций вставки и удаления, особенно в сценариях с частыми изменениями.

4. Упорядоченное хранение данных

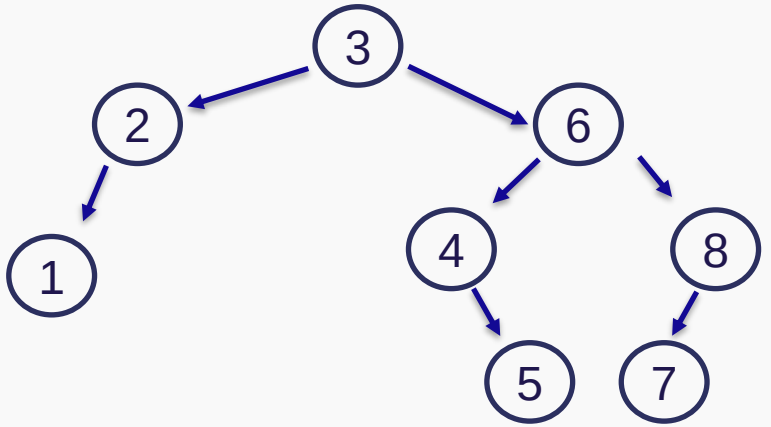
Оба типа деревьев хранят данные в отсортированном порядке, что позволяет легко выполнять операции диапазона и находить минимальные или максимальные значения. Это делает их полезными для реализации различных структур данных, таких как множества и ассоциативные массивы.

5. Применение в реальных системах

AVL-деревья и **красно-чёрные деревья** широко используются в реальных приложениях, таких как базы данных, файловые системы и системы управления памятью. Например, **красно-чёрные деревья** используются в стандартной библиотеке **C++ (STL)** для реализации контейнеров.

AVL-ДЕРЕВЬЯ

Эти деревья поддерживают строгую балансировку с помощью условия "разности высот" — разница высот левого и правого поддеревьев для любого узла не должна превышать 1. Это гарантирует, что операции будут оставаться быстрыми.

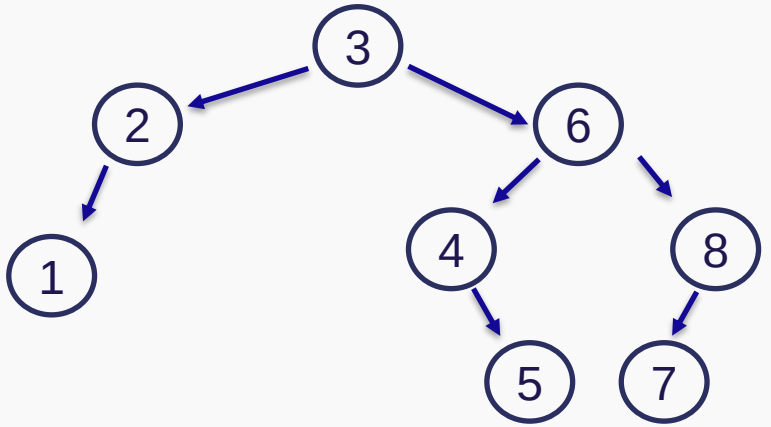


Они были предложены в 1962 году советскими математиками Георгием Адельсоном-Вельским и Евгением Ландисом. Название "AVL" происходит от первых букв их фамилий.

AVL-ДЕРЕВЬЯ

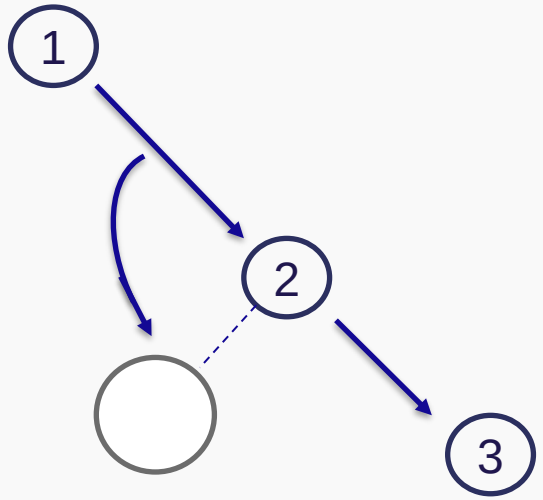
Правила AVL дерева:

1. Каждый узел имеет значение и два указателя: на левое и правое поддеревья.
2. Для каждого узла высота левого и правого поддеревьев может отличаться не более чем на 1. Это условие называется **балансировкой**.
3. Высота дерева определяется как максимальная длина пути от корня до самого глубокого листа.
4. При добавлении или удалении узлов дерево может стать несбалансированным. В этом случае выполняются операции вращения (левое или правое) для восстановления баланса.
5. **AVL-деревья** обеспечивают логарифмическое время выполнения операций поиска, вставки и удаления, что делает их эффективными для хранения и поиска данных.



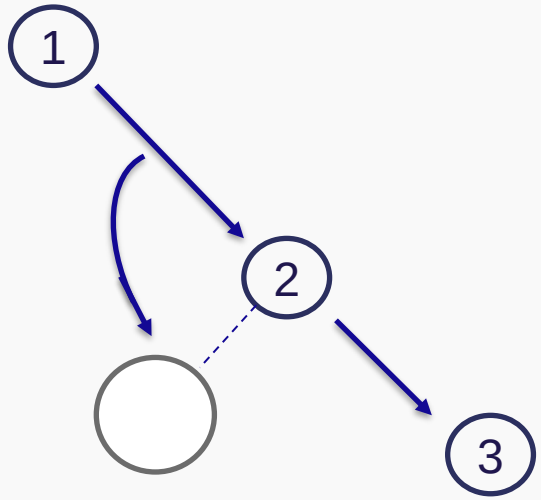
ВСТАВКА В AVL-ДЕРЕВО

Процесс вставки в **AVL-дерево** начинается так же, как и в обычном бинарном дереве поиска. После добавления нового узла необходимо проверить, не нарушены ли условия балансировки. Если дерево становится несбалансированным, выполняются ротации для восстановления баланса. Это может потребовать одного или нескольких вращений, в зависимости от ситуации.



ОПРЕДЕЛЕНИЕ «БАЛАНСА»

Балансировка в **AVL-деревьях** определяется с помощью коэффициента баланса, который рассчитывается как разница высот левого и правого поддеревьев для каждого узла. AVL-условия требуют, чтобы этот коэффициент находился в диапазоне от -1 до 1. Это означает, что высота левого и правого поддеревьев может отличаться не более чем на единицу. Соблюдение этих условий позволяет поддерживать сбалансированность дерева и обеспечивает эффективное выполнение операций.

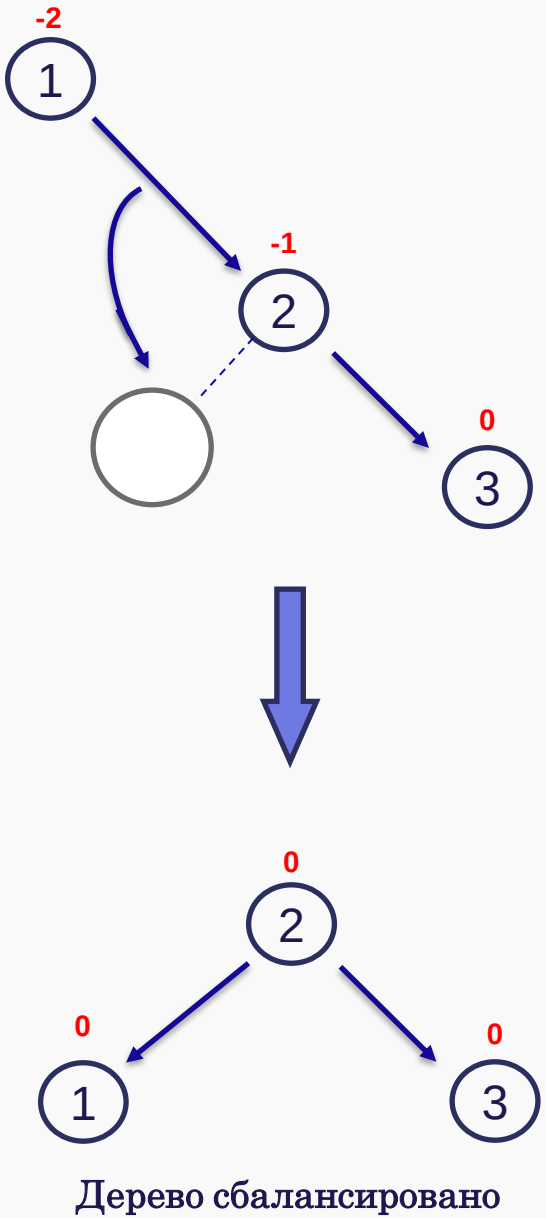


AVL-ДЕРЕВЬЯ

Балансировка:

Левое вращение: Применяется, когда правое поддерево узла выше левого. Это позволяет переместить узел вниз и сделать его левым потомком.

Поворачиваем ребро, связывающее корень и его правый дочерний узел, влево



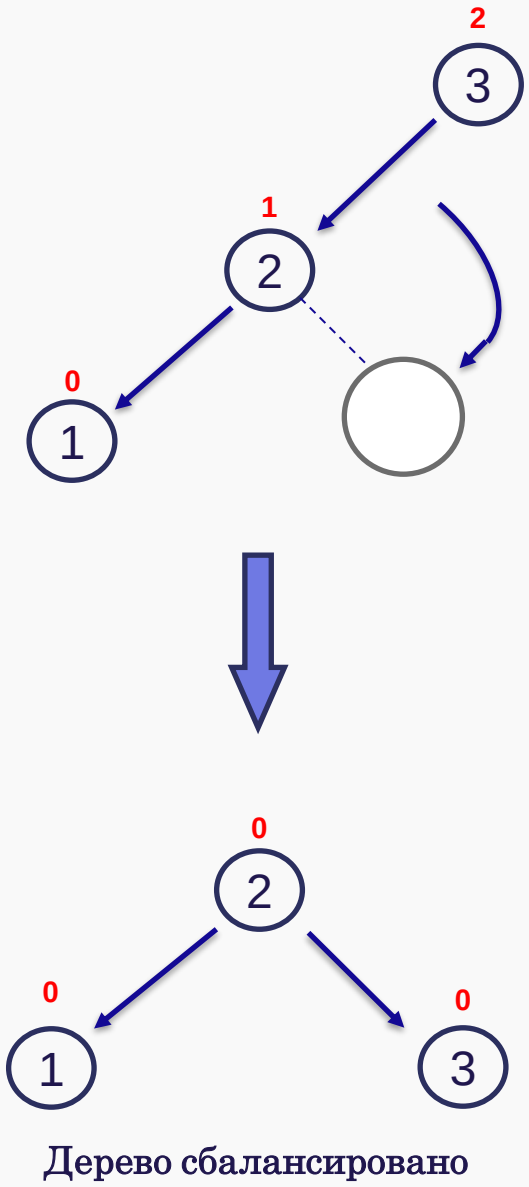
Дерево сбалансировано

AVL-ДЕРЕВЬЯ

Балансировка:

Правое вращение: Применяется, когда левое поддерево узла выше правого. Это позволяет переместить узел вниз и сделать его правым потомком.

Поворачиваем ребро, связывающее корень (3) и его левый дочерний узел (2), вправо.

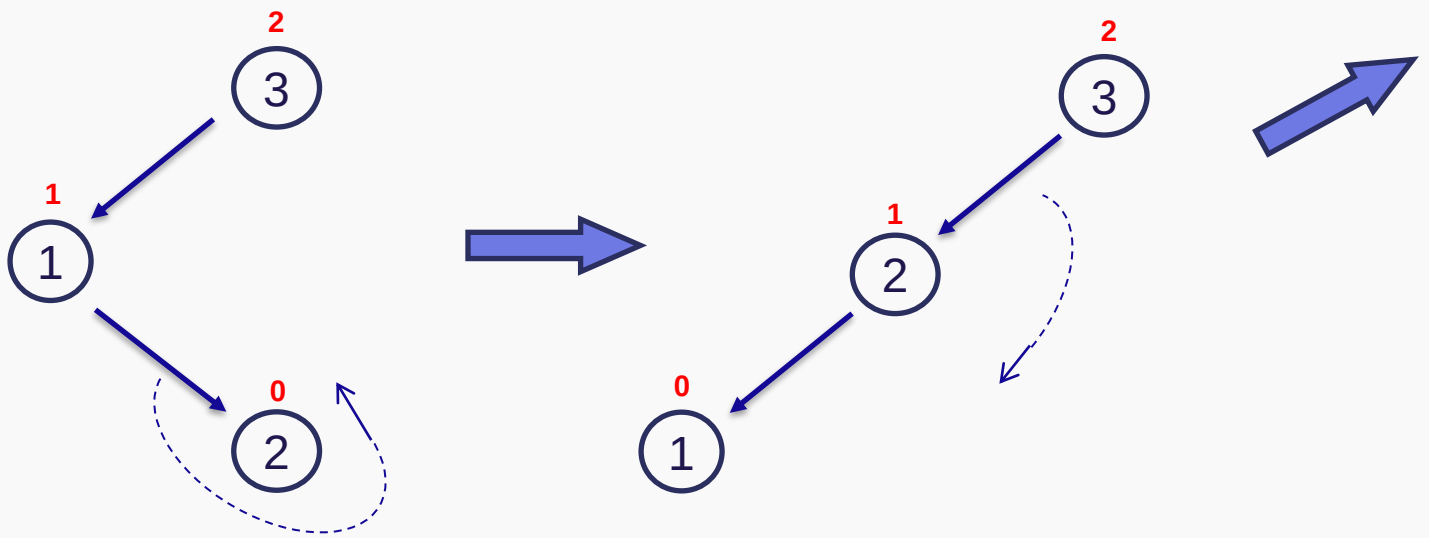


Дерево сбалансировано

AVL-ДЕРЕВЬЯ

Балансировка:

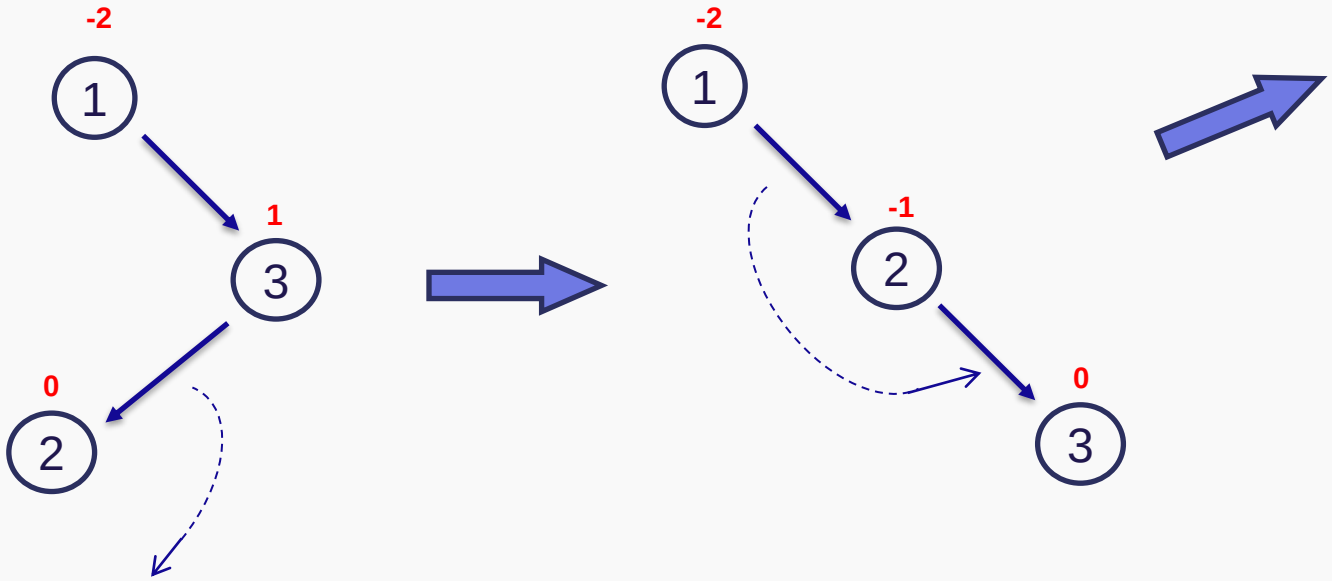
Левое-правое вращение: Применяется, когда левое поддерево правого узла выше. Сначала выполняется левое вращение на левом поддереве, затем правое вращение на текущем узле.

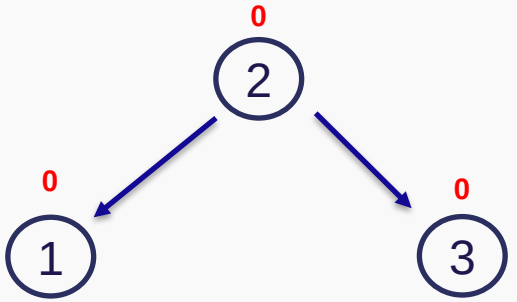


AVL-ДЕРЕВЬЯ

Балансировка:

Правое-левое вращение: Применяется, когда правое поддерево левого узла выше. Сначала выполняется правое вращение на правом поддереве, затем левое вращение на текущем узле.



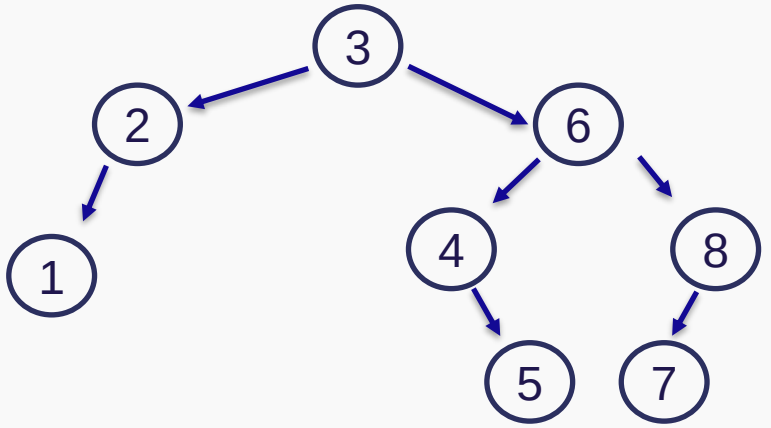


Дерево сбалансировано

AVL-ДЕРЕВЬЯ

Удаление из AVL-дерева:

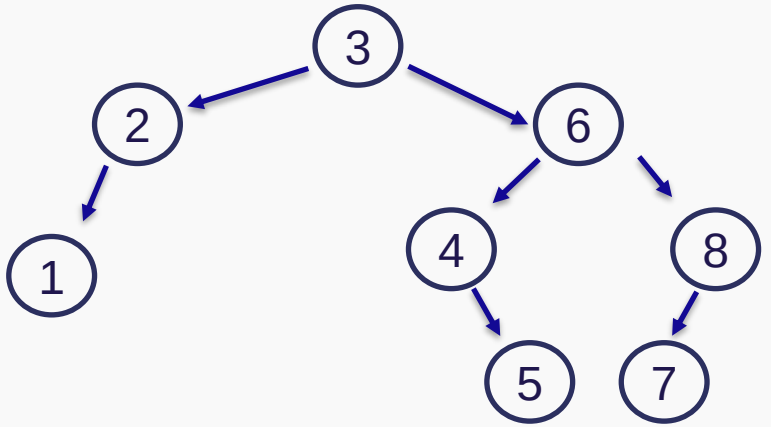
Процесс удаления узла из **AVL-дерева** аналогичен удалению из обычного бинарного дерева поиска. После удаления узла необходимо проверить, не нарушены ли условия балансировки. Если дерево становится несбалансированным, выполняются ротации для восстановления баланса. Это может потребовать одного или нескольких вращений, в зависимости от ситуации.



AVL-ДЕРЕВЬЯ

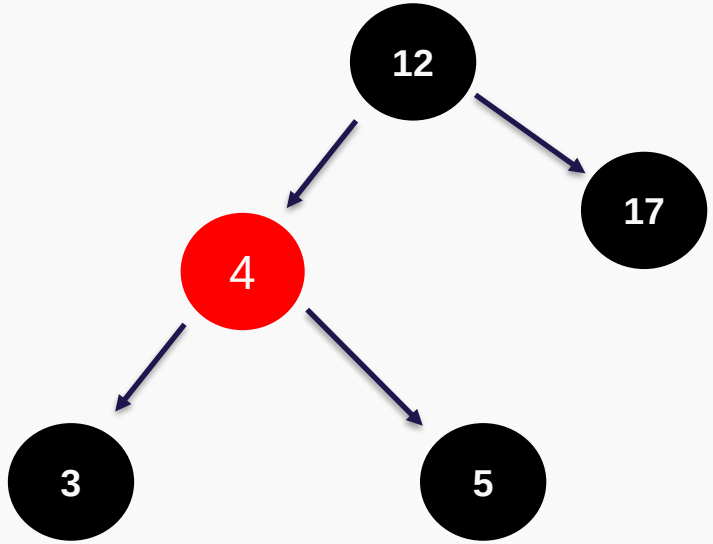
Применение:

AVL-деревья часто используются в приложениях, где важна быстрая вставка, удаление и поиск данных, таких как базы данных и системы управления памятью. Они обеспечивают более строгую балансировку по сравнению с другими структурами данных, такими как красно-чёрные деревья, что делает их более эффективными для операций, требующих частого поиска.



КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Они обеспечивают более гибкий подход к балансировке, где высота левого и правого поддеревьев может различаться более чем на 1, но при этом соблюдаются определенные правила окраски узлов и родительских/дочерних отношений, которые обеспечивают общую сбалансированность дерева.



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.

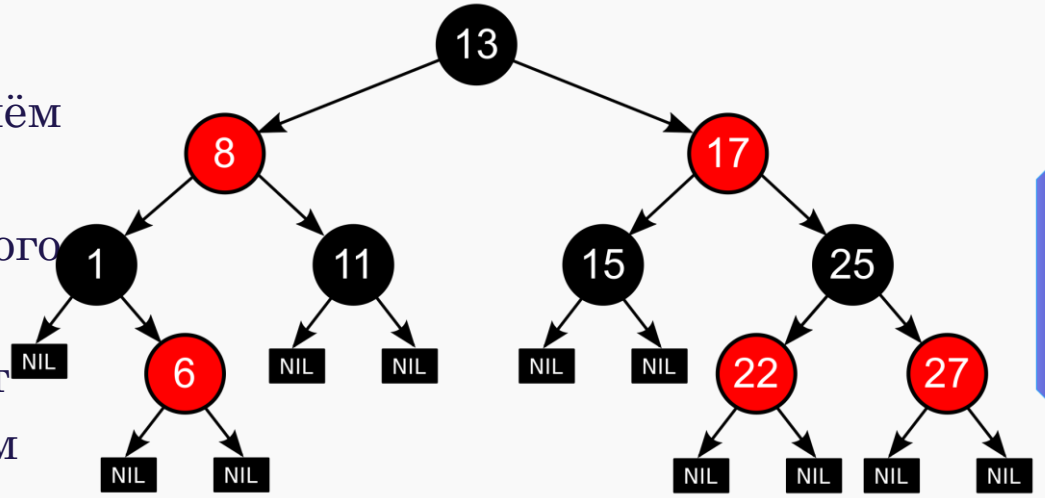
КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Правила красно-чёрного дерева:

1. Каждый узел либо красный, либо чёрный
2. Корень - чёрный.
3. Нулевые указатели считаются листьями, причём листья - чёрные.
4. Потомки красного узла - чёрные. (Потомки чёрного узла не обязательно красные).
5. Любой путь от корня поддерева до листа содержит одинаковое число чёрных узлов. (Глубина по чёрным узлам).

Грубая оценка на основании правил 4 и 5 показывает, что длины двух соседних поддеревьев отличаются не более, чем в 2 раза.

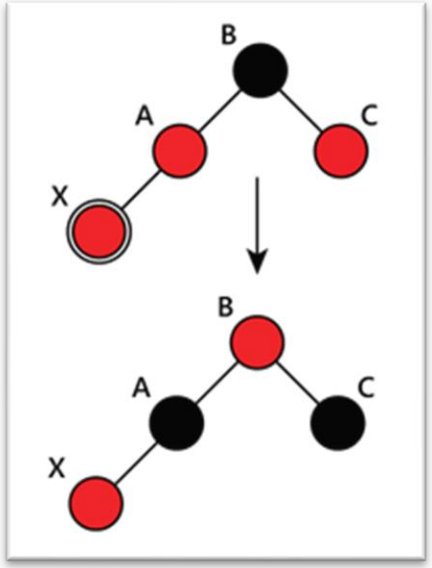
Каждый новый узел изначально считается красным. Если это нарушает одно из правил, обычно 4 или 5, производится балансировка...



КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Вставка в красно-чёрное дерево:

Вставка нового узла в **красно-чёрное дерево** начинается так же, как и в обычном бинарном дереве поиска. Новый узел всегда добавляется как красный. После вставки проверяются свойства **красно-чёрного дерева**. Если какие-либо свойства нарушены, необходимо выполнять операции ротации и перекраску узлов для восстановления красно-чёрных свойств.



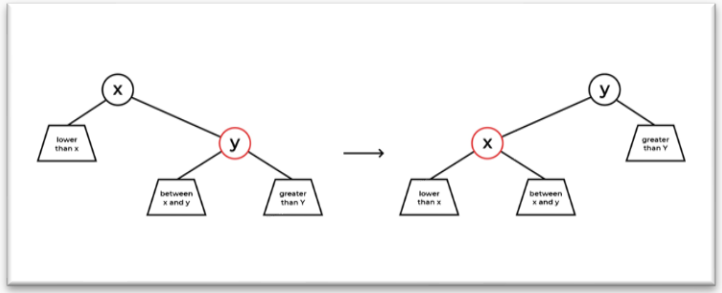
КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

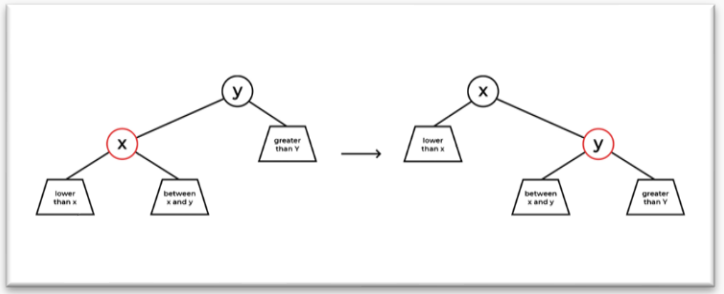
Ротации в красно-чёрных деревьях:

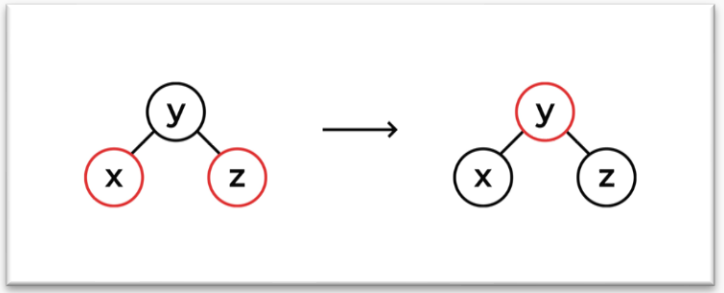
Левое вращение: Применяется для перемещения узла вниз и его правого поддерева вверх.

Правое вращение: Применяется для перемещения узла вниз и его левого поддерева вверх.

Перекраска: Когда родительский узел и дядя (сосед) красного узла также красные, их необходимо перекрасить в чёрный, а деда перекрасить в красный, чтобы восстановить баланс.



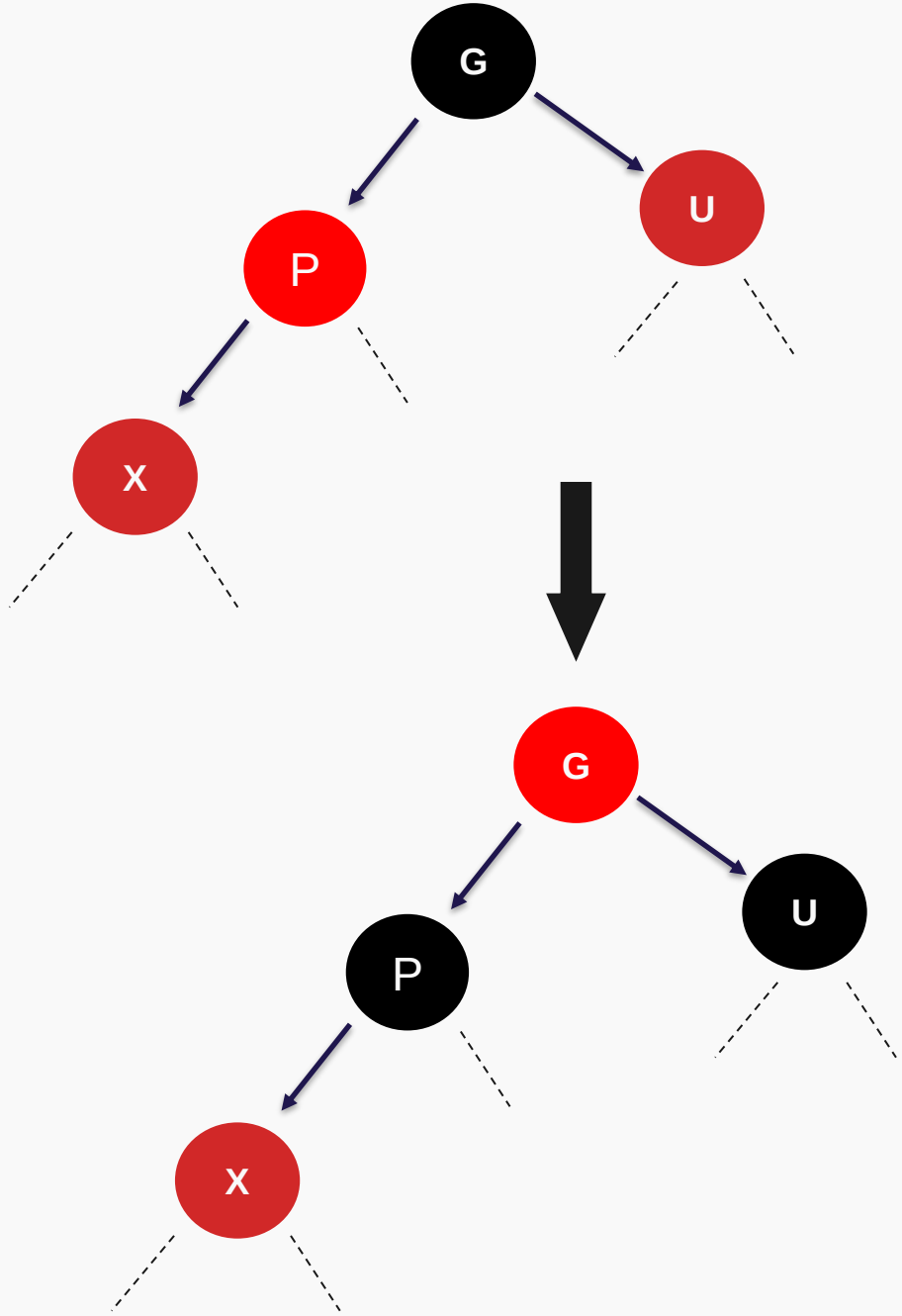




КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Вставка в красно-чёрное дерево. Случай первый:

Если дядя **красного цвета**, то мы можем «спустить» чёрный цвет с уровня деда на уровень отца и перекрасить узлы, как показано на рисунке. В этом случае «чёрная высота» останется прежней, однако возможно нарушение 4 правила для элемента G, поэтому необходимо рекурсивно вызвать дальнейшую балансировку для этого узла.

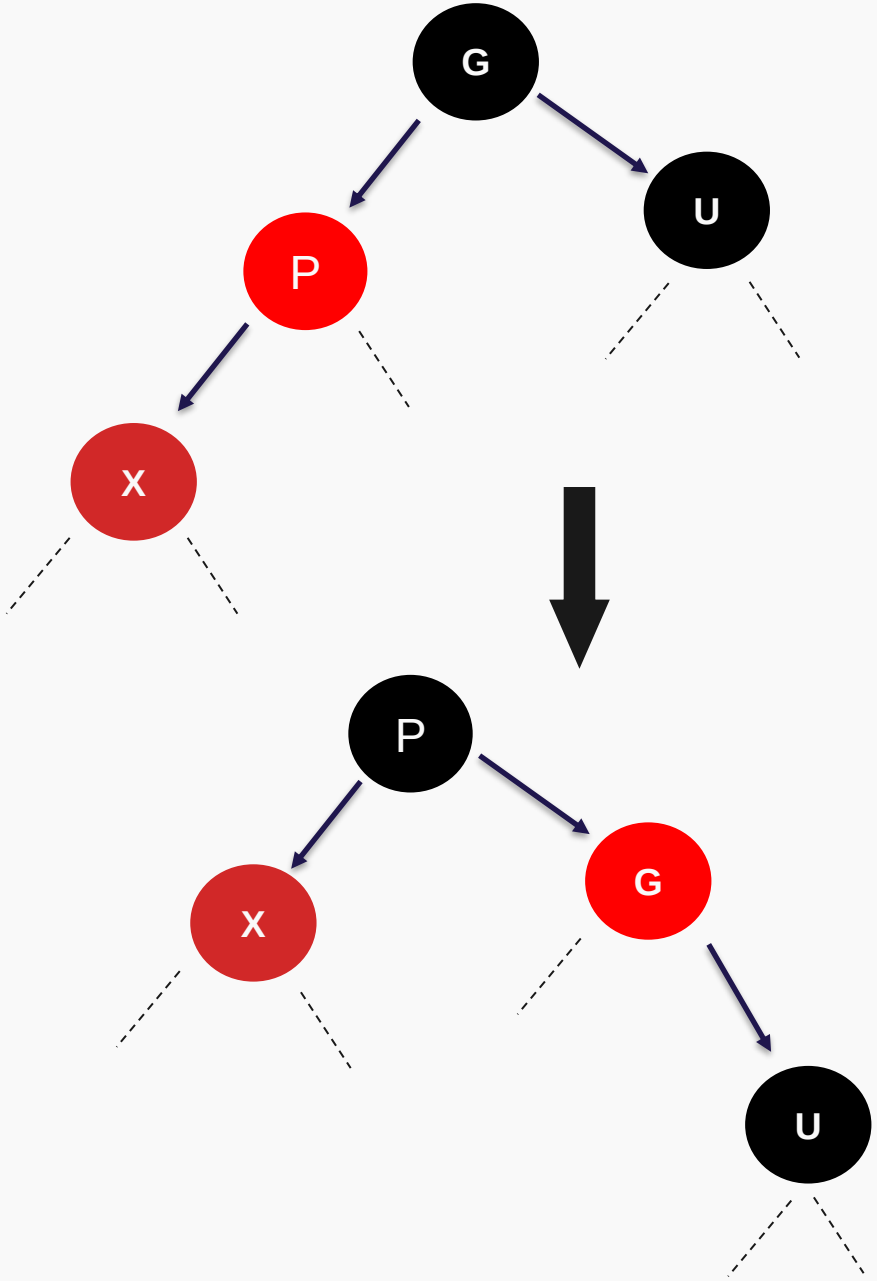


X — добавленный элемент, который нарушает 4 пункт правил.
P — папа элемента X
G — дедушка элемента X, папа элемента P
U — дядя элемента X, брат элемента P, второй сын элемента G.

КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Вставка в красно-чёрное дерево. Случай второй:

Если дядя **чёрного цвета**, то мы уже можем совершить большой поворот от отца через деда к чёрному дяде и перекрасить P в чёрный, а G в красный. В результате этого поворота 4 правило будет выполнено.



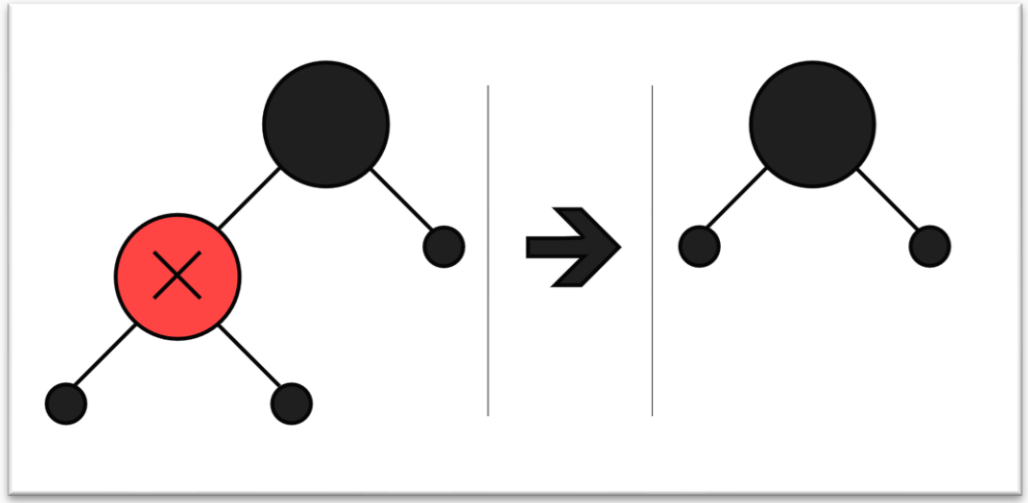
X — добавленный элемент, который нарушает 4 пункт правил.
P — папа элемента X
G — дедушка элемента X, папа элемента P
U — дядя элемента X, брат элемента P, второй сын элемента G.

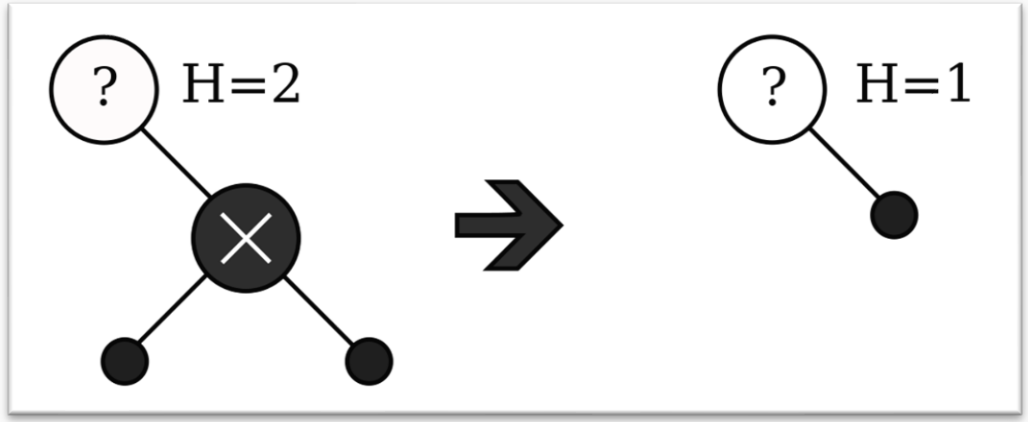
КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Удаление из красно-чёрного дерева:

Удаление узла из **красно-чёрного дерева** может быть более сложным, чем его вставка, из-за необходимости поддержания свойств дерева. При удалении узла:

1. Удаляется узел, как в обычном бинарном дереве поиска.
2. Если удаляемый узел **чёрного цвета**, это может нарушить количество чёрных узлов на пути от корня до листьев.
3. Для решения возникших проблем могут потребоваться ротации и перекраски, аналогичные тем, что применяются при вставке.





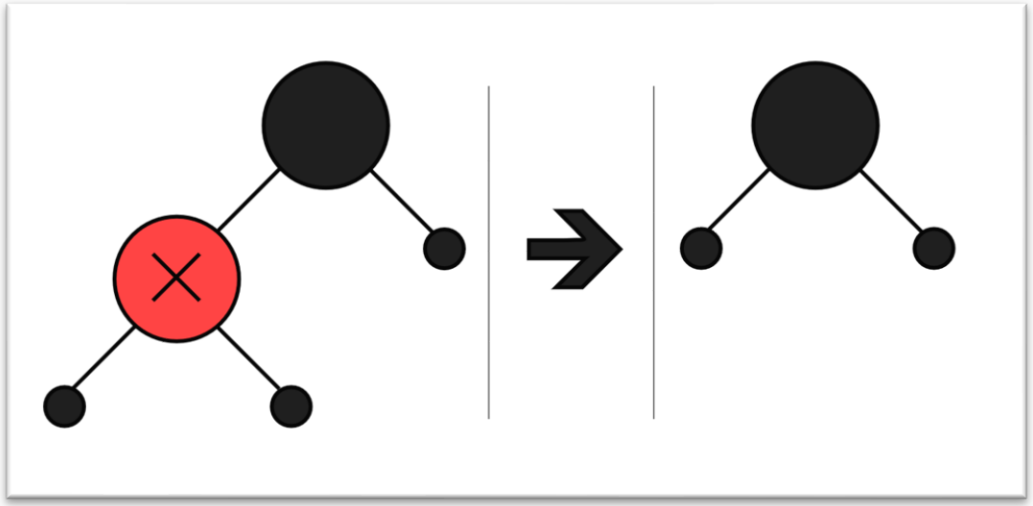
КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Удаление из красно-чёрного дерева:

1) Удаление красной вершины с 0 детей:

В этом случае мы просто заменяем данную вершину на **NULL**-вершину. Никакие пункты из вышеописанных правил не нарушены (черная высота не нарушается), дерево все еще корректно.

Удаляемую вершину обозначим крестом.

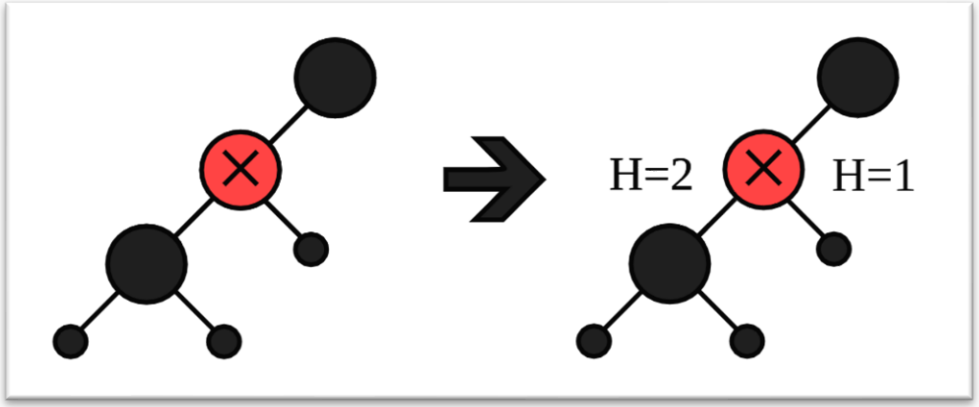


КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Удаление из красно-чёрного дерева:

2) Удаление красной вершины с 1 ребенком:

Ребенок красной вершины может быть только чёрным. Но если такая вершина является единственным ребенком, то происходит нарушение чёрной высоты. Таким образом, еще перед удалением у нас имеется некорректное дерево. Если балансировка при добавлении написана верно, то такой случай невозможен.

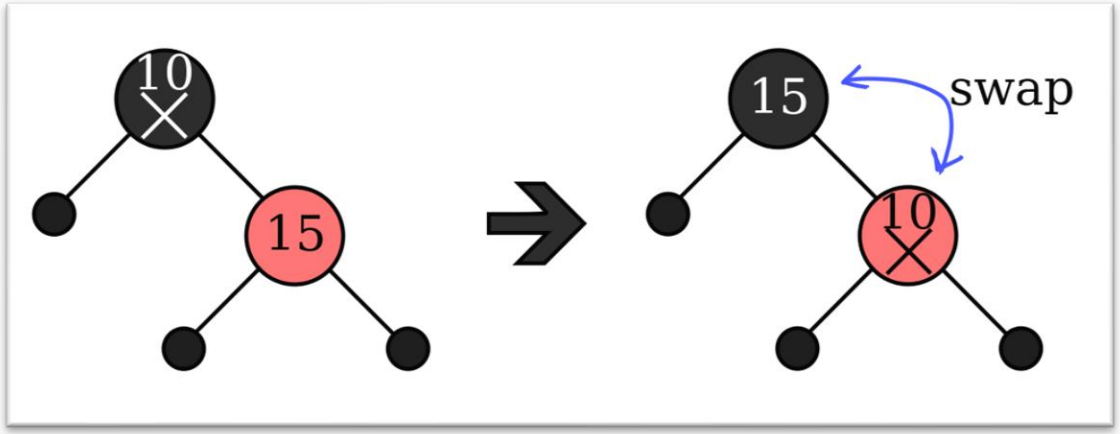


КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Удаление из красно-чёрного дерева:

3) Удаление чёрной вершины с 1 ребенком:

При удалении узла с одним или нулем детей в **красно-чёрном дереве** мы можем заменить его на узел-потомок, сохраняя цвета узлов. Это позволяет сохранить черную высоту, не нарушая балансировку дерева.



КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

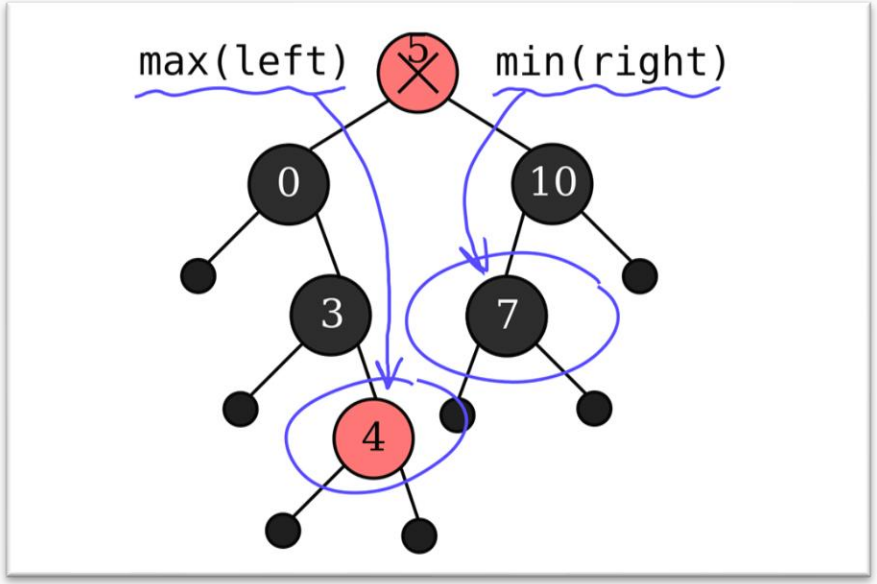
Удаление из красно-чёрного дерева:

4) Удаление красной или чёрной вершины с 2 детьми:

Здесь мы рассматриваем не только красные вершины, но и чёрные. Все из-за метода, который здесь можно применить:

Так как у вершины есть два ребенка, стоит найти замену. Эту замену мы можем найти в этих же поддеревьях: из левого мы можем искать максимальное, а из правого — минимальное.

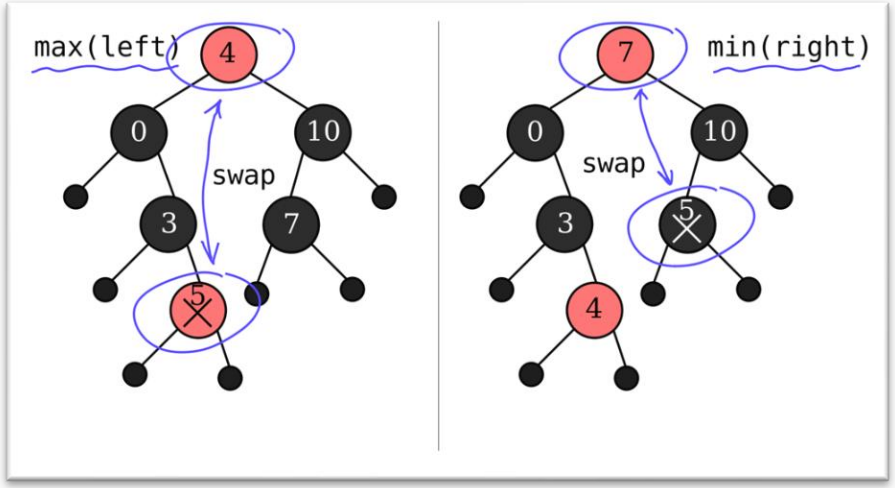
Чтобы найти максимальный элемент в поддереве, из-за свойств бинарного дерева нам достаточно просто постоянно спускаться по правой ветви. То же и для минимального — спускаться до конца по левой ветви.



КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Удаление из красно-чёрного дерева:

Все же выбрав нужную вершину, нам необходимо сделать обмен значений с ней. Это очень важно, ведь если мы поменяем цвета, иначе может нарушиться черная высота, а замена значений пройдет очень гладко (мы же специально выбрали максимальный или минимальный — то есть порядок значений не нарушится).

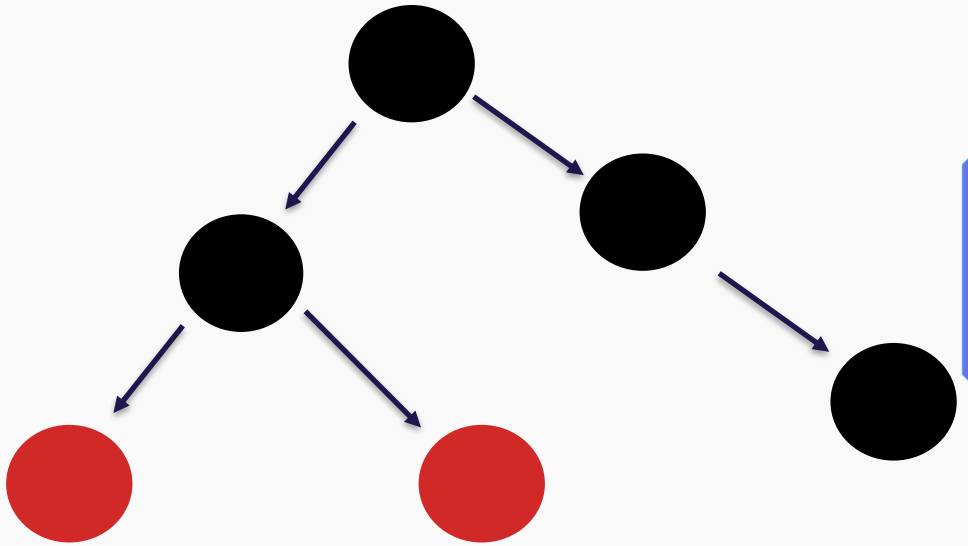


Таким образом, нам останется удалить вершину, с которой мы произвели замену, а у нее уже точно один или ноль детей.

КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Проблема дважды чёрного узла:

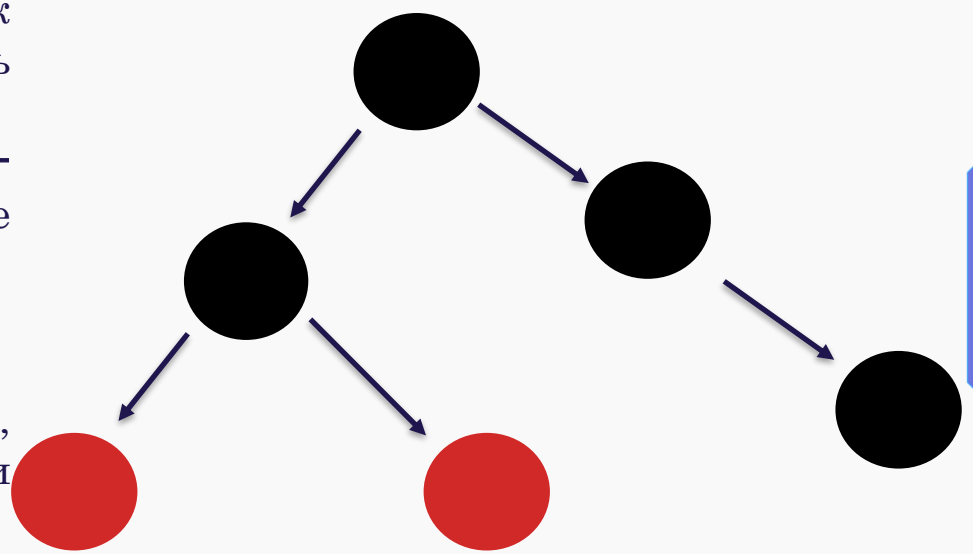
Дважды черный узел в красно-черном дереве — это узел, который считается чёрным дважды из-за операций вставки или удаления. Это может произойти, когда удаляется черный узел, и его цвет передается вниз по дереву, что приводит к нарушению свойств дерева.



КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

Проблема дважды чёрного узла:

1. Удаление чёрного узла: При удалении чёрного узла, если у него нет двух чёрных детей, его цвет передается к родителю, что может привести к ситуации, когда родитель становится дважды черным.
2. Нарушение свойств: Это нарушает свойства **красно-чёрного дерева**, так как в любом пути от корня до листа не должно быть двух чёрных узлов подряд.



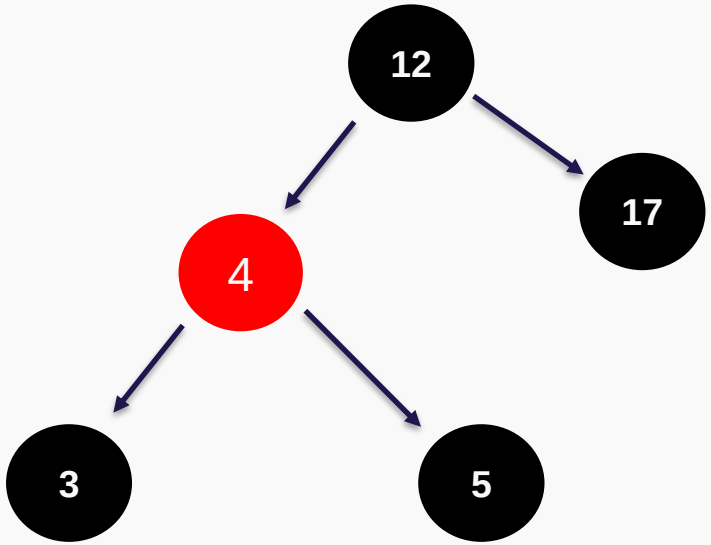
Решение проблемы:

1. Перекраска: Если у родителя есть красный брат, перекрасьте родителя в красный, а брата в чёрный, и поднимитесь к дедушке.
2. Ротация: Если брат черный, выполните ротацию (левую или правую) вокруг родителя, чтобы перераспределить чёрный цвет.
3. Рекурсивное исправление: Если после этих операций проблема не решена, повторите процесс для родителя.

КРАСНО-ЧЁРНЫЕ ДЕРЕВЬЯ

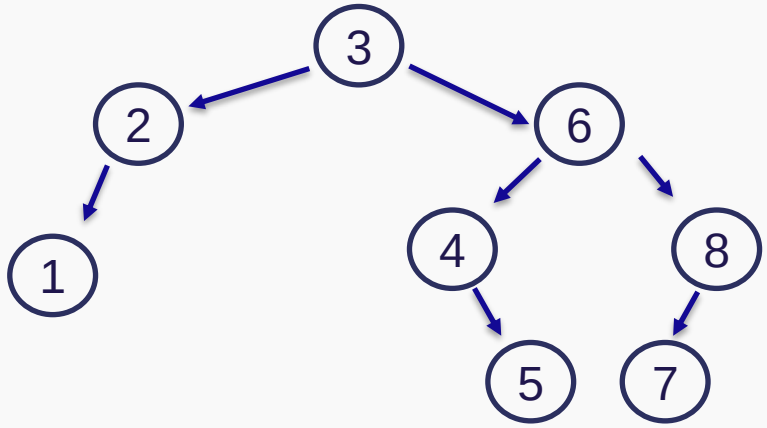
Применение:

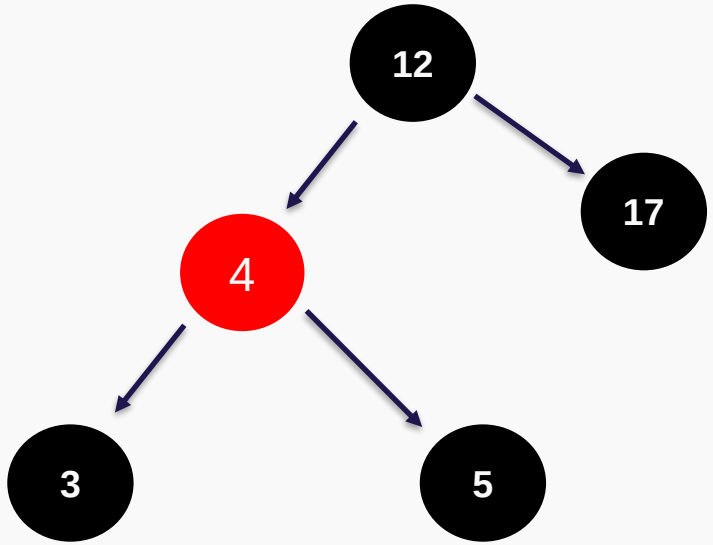
Красно-чёрные деревья широко применяются в различных сферах программирования и компьютерных наук благодаря их эффективности и поддержанию сбалансированности. Они используются для реализации ассоциативных массивов и наборов в стандартных библиотеках языков программирования, таких как **C++** (STL) и **Java** (TreeMap). Кроме того, **красно-чёрные деревья** находят свое применение в системах управления базами данных для создания индексов, что обеспечивает быстрый доступ к данным.



СРАВНИТЕЛЬНЫЙ АНАЛИЗ

Красно-чёрные и AVL-деревья — это оба типа самосбалансированных бинарных деревьев поиска, но они имеют разные механизмы балансировки. **AVL-деревья** обеспечивают более строгую балансировку, поддерживая разницу высот между поддеревьями не более единицы, что приводит к более быстрому поиску, однако это требует частых вращений при вставке и удалении узлов. В отличие от этого, **красно-чёрные деревья** допускают большую разницу в высоте между поддеревьями, что упрощает операции вставки и удаления, но может приводить к менее эффективному поиску по сравнению с **AVL-деревьями.**





ЗАКЛЮЧЕНИЕ

Выбор между **AVL-деревьями** и **красно-чёрными деревьями** зависит от конкретных потребностей приложения. **AVL-деревья** обеспечивают лучшую скорость поиска благодаря строгой балансировке, что делает их идеальными для задач с частыми операциями поиска. В то же время **красно-чёрные деревья** предлагают более эффективные вставки и удаления, что делает их предпочтительными в сценариях с высокой динамикой изменения данных. Понимание этих характеристик поможет разработчикам выбрать подходящую структуру данных для достижения оптимальной производительности в своих приложениях.