Практическое занятие №4

Густов Владимир Владимирович
gutstuf@gmail.com

Цитата дня: Вечная весна в одиночной камере (с) Егор Летов

Дерево

- содержит указатель на потомков;
- удаление/добавление элемента либо упорядочено (для бинарных), либо производится напрямую из локального родителя.

Бинарное (двоичное) дерево строится по правилам:

- в левое поддерево добавляются **элементы меньше корня;**
- в правое поддерево добавляются **элементы больше или равные корню;**

Способы обходов дерева:

- поиск в глубину (прямой, симметричный, обратный);
- поиск в ширину.

Бинарное дерево поиска

Свойства:
- вставка/удаление элемента — O(log n)
- поиск элеменета — O(log n);

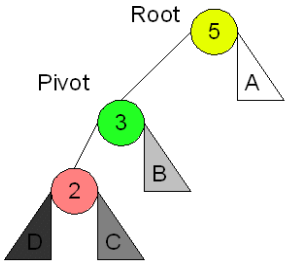
В худшем случае (при вырождении дерева в список): O(n)

- каждый узел имеет не более двух потомков;
- строгое БД – узел имеет степень два или ноль;
- полное БД – каждый узел имеет по 2 дочерних узла;
- сбалансированное БД – для каждой вершины, кол-во вершин в левом и в правом поддереве различаются не более чем на единицу;
- вырожденное БД – каждый узел имеет всего один дочерний узел.

Балансировка

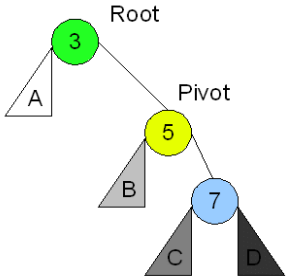
Важно, чтобы разница между высотами левого и правого поддеревьев была не более 1 (AVL).
Соответственно, для сохранения сбалансированности, необходимо поворачивать (перестраивать) дерево.

Left Left Case



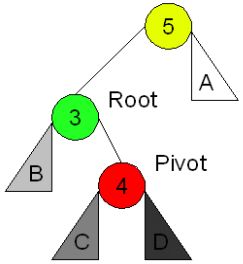
Right Rotation

Right Right Case



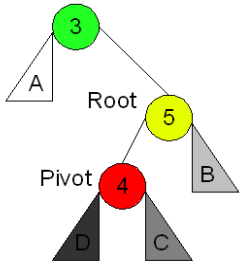
Left Rotation

Left Right Case

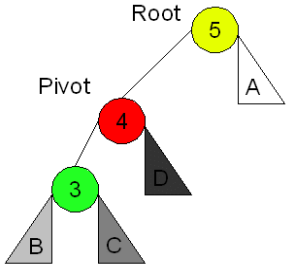


Left Rotation

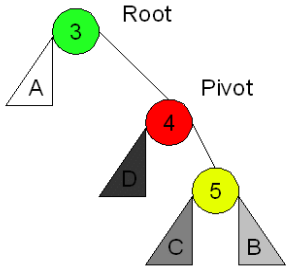
Right Left Case



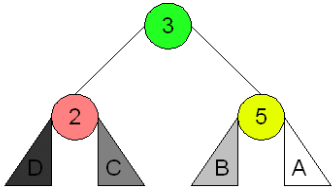
Right Rotation

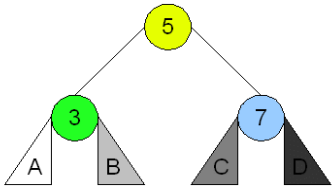


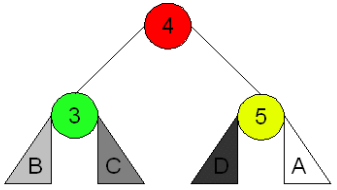
Right Rotation

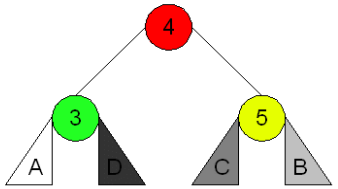


Left Rotation









Давайте повторим

Постройте БДП (бинарное дерево поиска) **и лишь затем сбалансируйте их.**

1) 84, -7 , 38 , -7 , 46 , -94 , -73, 100 , 28 , 56 , -9, 20

2) 18, 3, 63, -42, -23, -50, 97, 53, 80, 70, 5, 21, 41

Постройте сбалансированное бинарное дерево

3) 59, 61, 84, 13, -56, 34, -16, 60, 32, 3, 74, -30, 69, -81

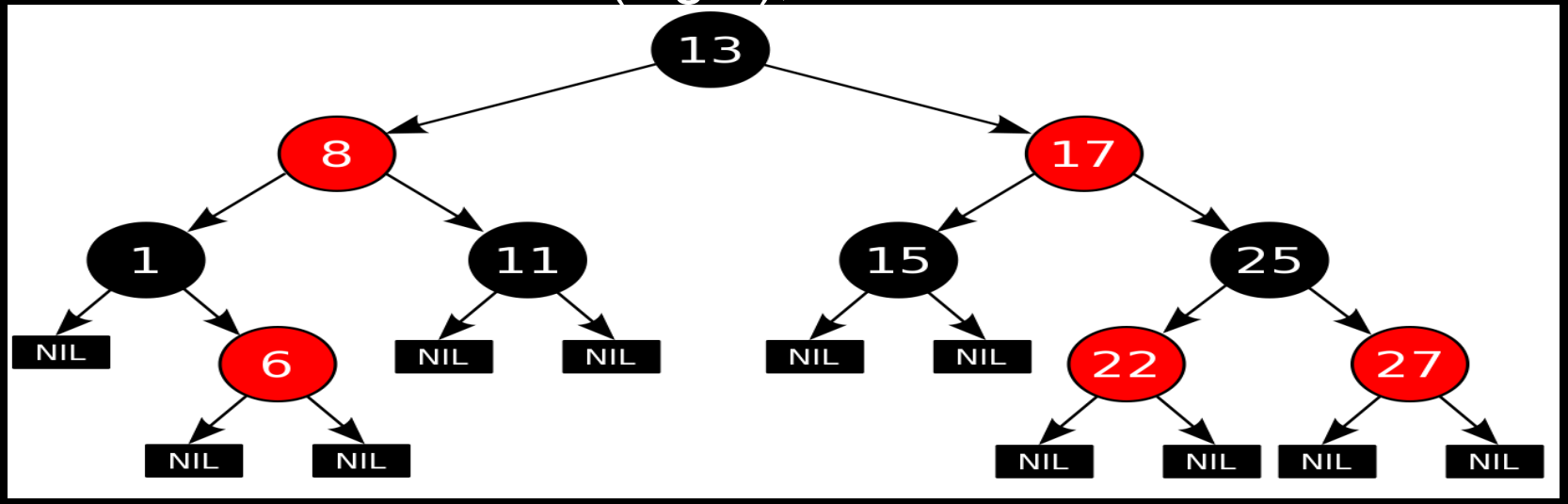
Красно-чёрное дерево

КЧД – сбалансированное бинарное дерево поиска, использующее для балансировки такой атрибут, как «цвет» (красный и чёрный).

Как и любое БДП, используется для организации быстрого поиска данных. (на их основе строятся ассоц. массивы)

Свойства:

- вставка/удаление элемента — O(log n) (даже в худшем случае);
- поиск элемента — O(log n);

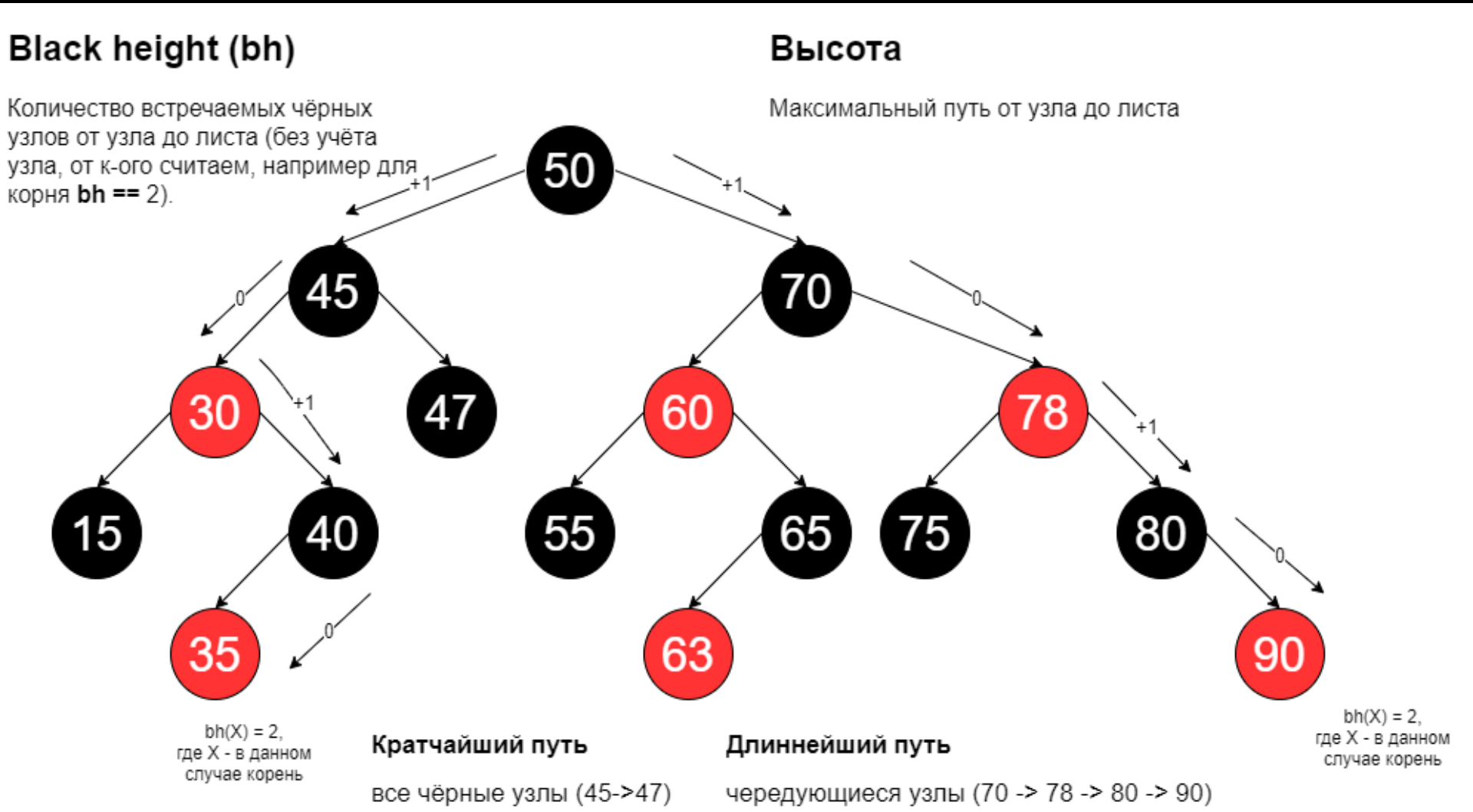


Свойства

- каждый узел окрашен либо в красный, либо в чёрный цвет (цвет – условность, по сути это дополнительное информационное поле в структуре, а-ля bool color);
- корень всегда окрашен в чёрный цвет;
- листья (`null/nil` узлы) всегда окрашены в чёрный цвет;
- каждый красный узел имеет только чёрные дочерние узлы и только чёрного родителя;
- чёрные узлы могут иметь в качестве дочерних чёрные;
- пути от узла к листьям должны содержать одинаковое количество чёрных узлов (так называемая – чёрная высота, **bh**).

Чёрная высота (black height)

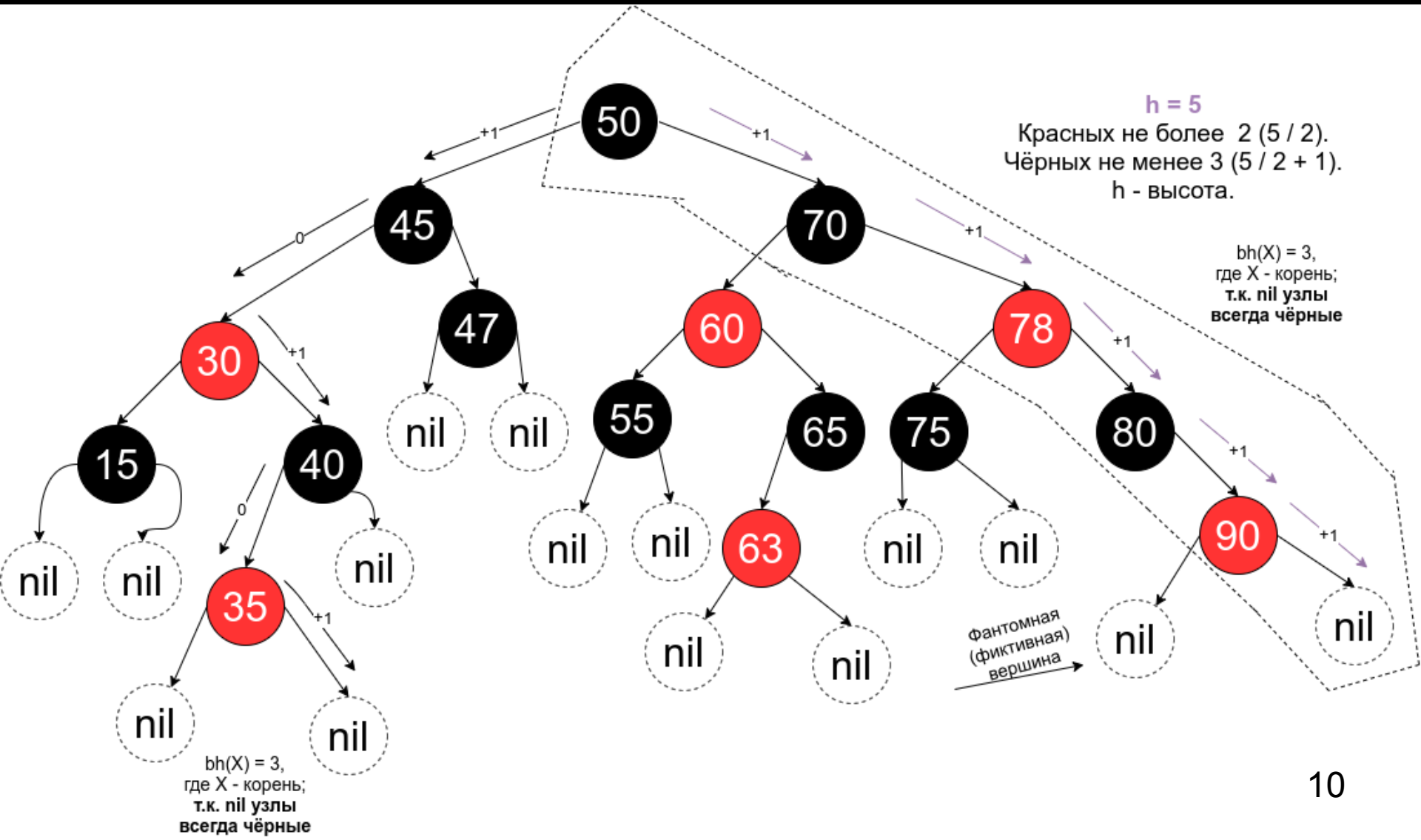
Обратите внимание, **bh** для левого и правого поддерева (относительно корня) одинаков и равен 2ум, т. к. количество встречаемых узлов (на пути к листу) равно 2.



Попробуйте подсчитать **bh** для остальных узлов. Чему он будет равен?

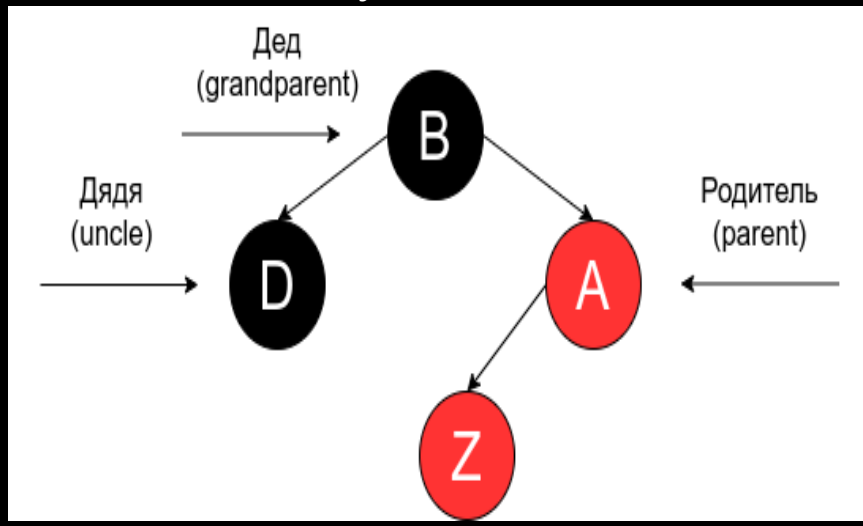
Красных вершин в пути всегда **не более** h/2 (где h — высота).

Чёрных вершин в пути всегда **не менее** h/2 + 1 (где h — высота).



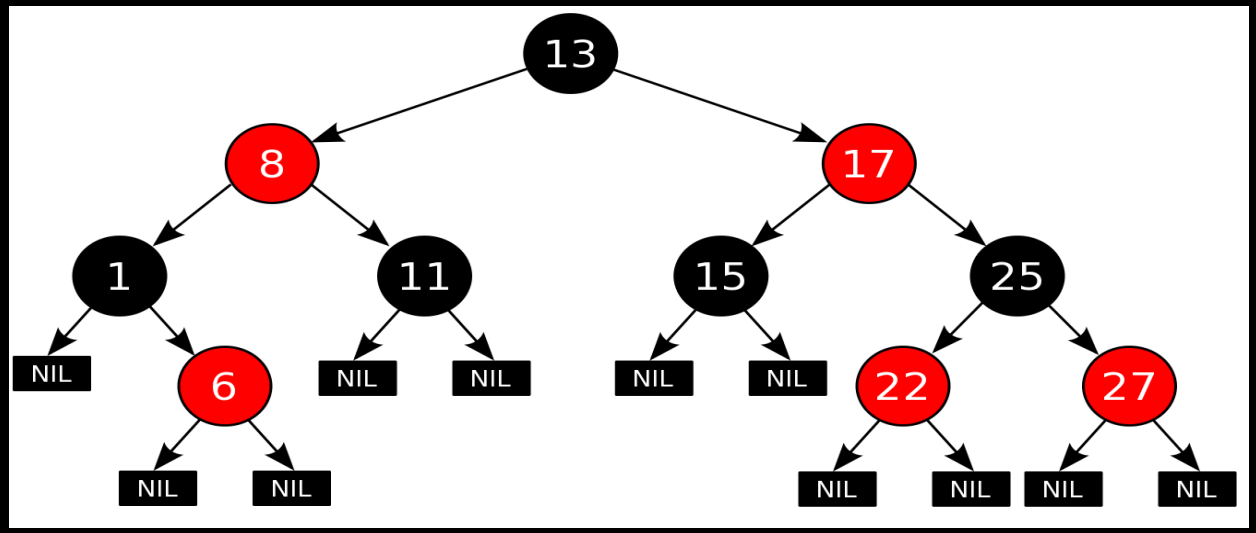
Алгоритм вставки в КЧД

Вставленный узел всегда окрашивается в красный цвет.
В терминах КЧД существует такие понятия, как: дядя (чёрный, красный), дед, родитель; исходя из которых выполняются соответствующие действия при вставке/удалении.



(Относительно узла Z)

Так же есть такое понятие как фантомный узел или nil узел, который **всегда считается чёрным узлом** и строится в конце узла (как на картинке ниже), но при этом по сути отсутствует. Воспринимайте его всегда как лист (или не вставленный узел листа). Порой **в реализации** это всего лишь 1 пустой узел, на который все ссылаются.



Алгоритм вставки в КЧД

- вставка производится по правилам БДП (слева строго меньше, справа больше или равно);
- после вставки необходимо производить балансировку дерева (посредством поворотов и перекраски узлов) **с учётом свойств КЧД**;
- возможны 4 ситуации (случая) при вставки (для узла X):

1) X — корень;
2) дядя узла X — красный;
3) дядя узла X — чёрный (вставка образует угол);
4) дядя узла X — чёрный (вставка образует линию);

Важно: данные «сценарии» работают в купе (совместно), далее рассмотрим каждый из них **отдельно** и их совместное применение.

1 случай

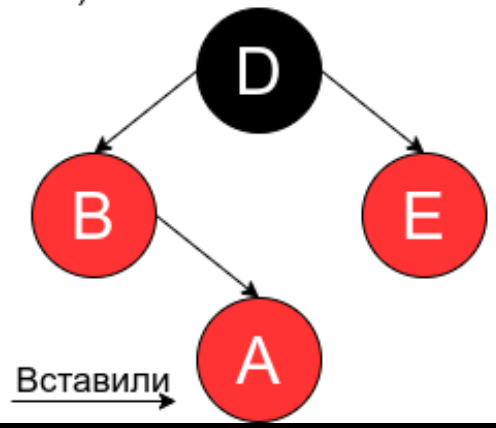
1) 

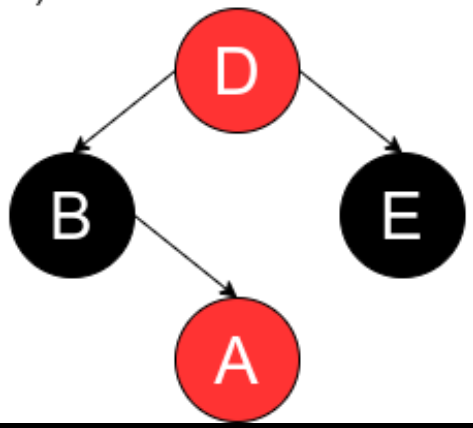
2) 

1) вставленный узел (A) красный;

2) нет родственников - корень, соответственно красим в чёрный.

2 случай

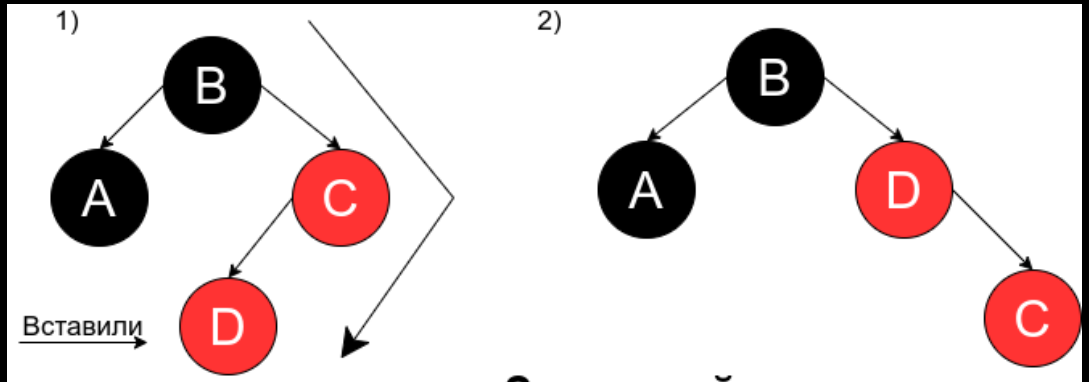
1) 

2) 

1) вставленный узел (A) красный;

2) дядя (узел E) красный - отдаём цвет деда (D) родителю и дяде, дед становится красным

(TL;DR: родственники перекрашиваются)

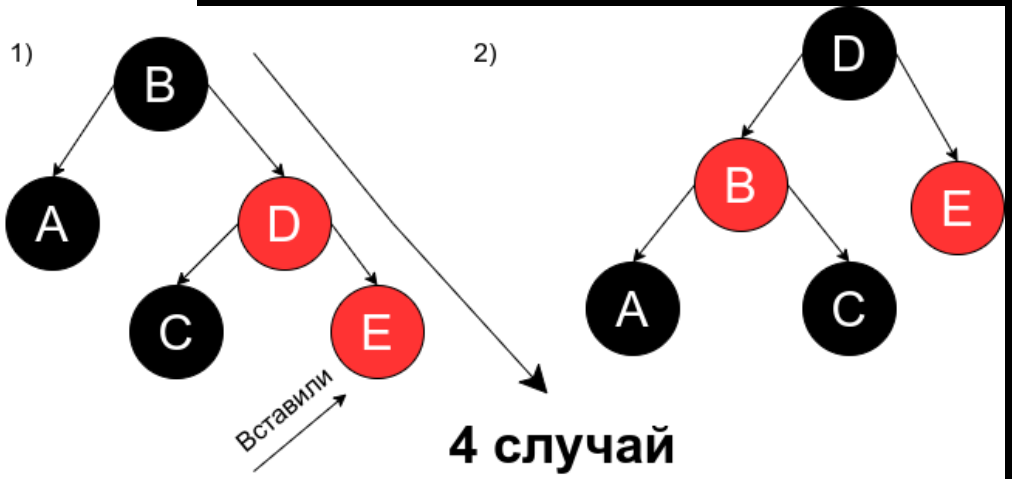


3 случай

1) вставленный узел (D) красный;

2) дядя (узел A) чёрный - вставка образовала угол, делаем правый поворот

(TL;DR: производим правый поворот родителя)



4 случай

1) вставленный узел (E) красный;

2) дядя (узел A) чёрный - вставка образовала прямую, делаем левый поворот и перекрашиваем деда и родителя

(TL;DR: производим левый поворот **и** перекраску деда)

Построим КЧД

по набору: 8, 5, 12, 9, 15, 13, 19, 10, 11.

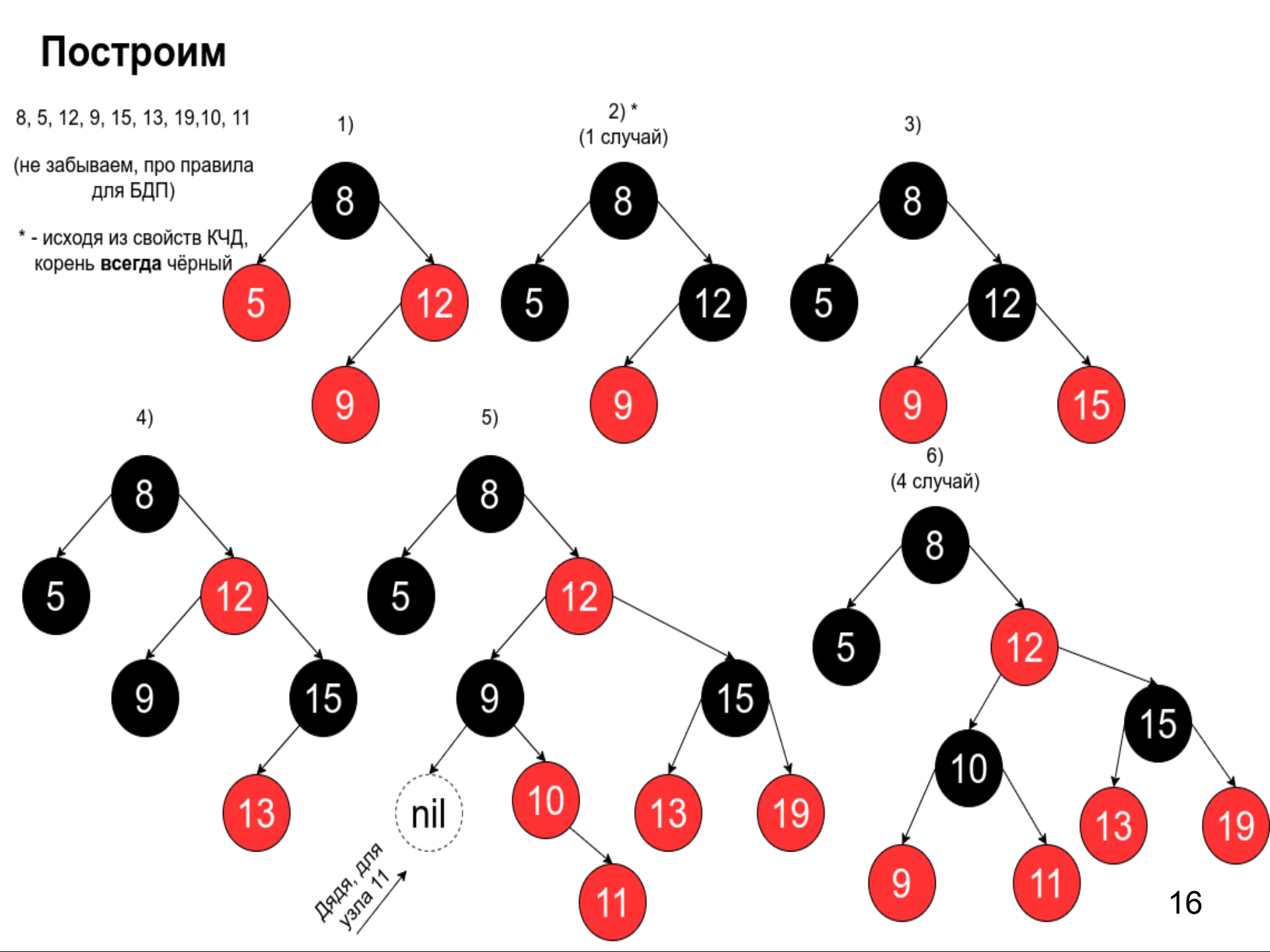
Некоторые шаги опущены (e.g. создание корня) для сокращения места

Построим

8, 5, 12, 9, 15, 13, 19,10, 11

(не забываем, про правила для БДП)

* - исходя из свойств КЧД, корень **всегда** чёрный



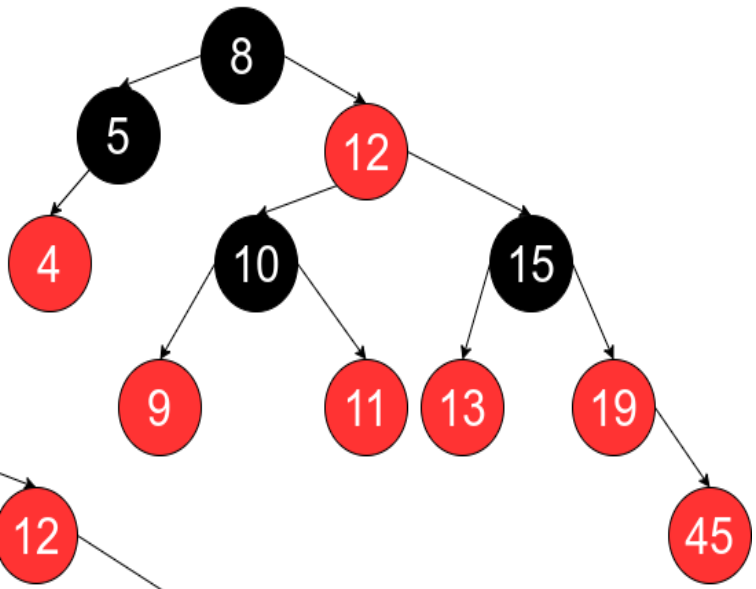
Добавим ещё

4, 45, 43

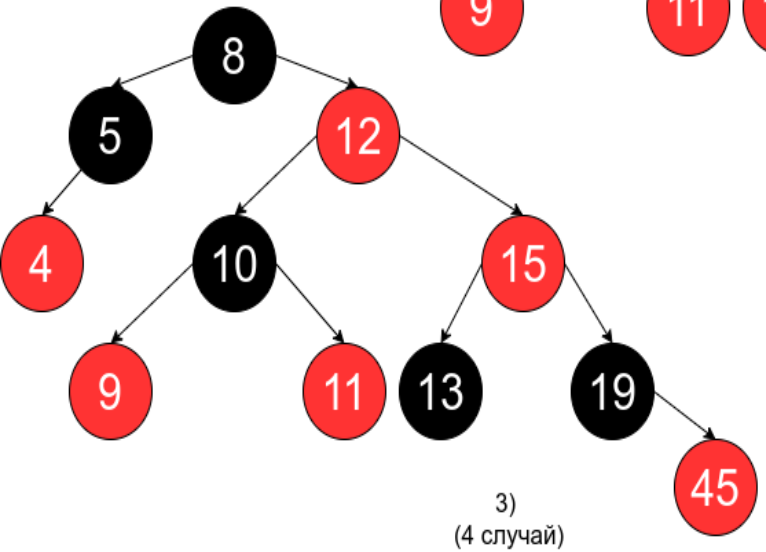
(не забываем, про правила для БДП)

* - исходя из свойств КЧД, корень **всегда** чёрный

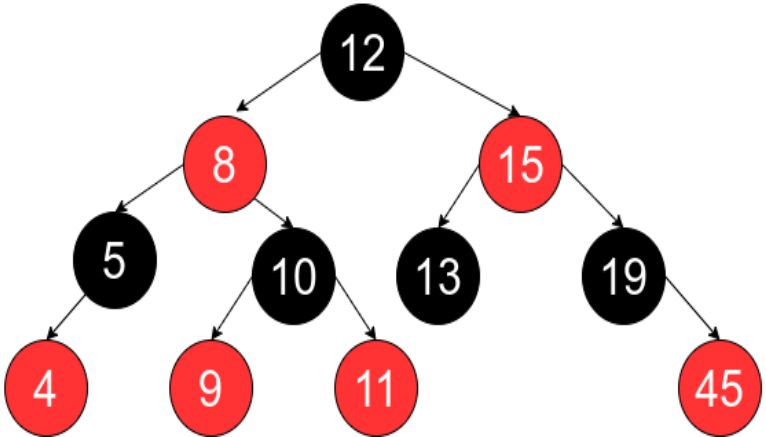
1)



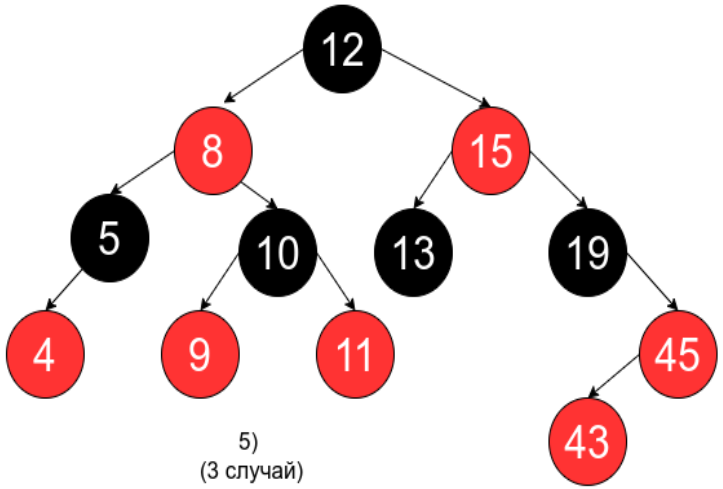
2)
(2 случай)



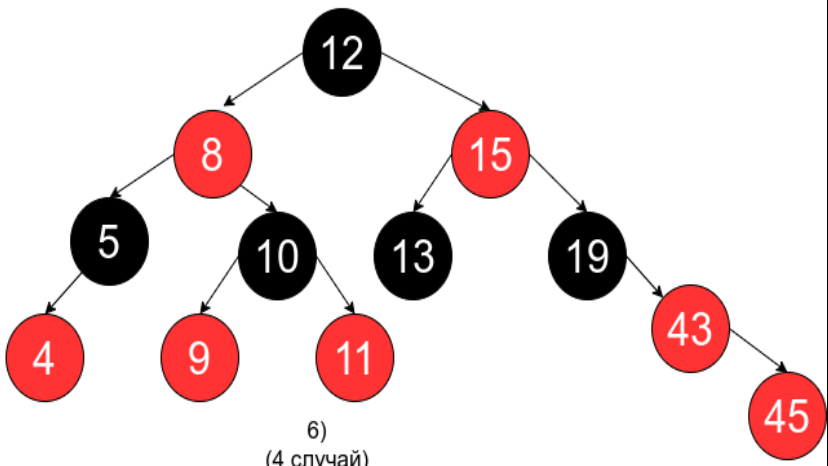
3)
(4 случай)



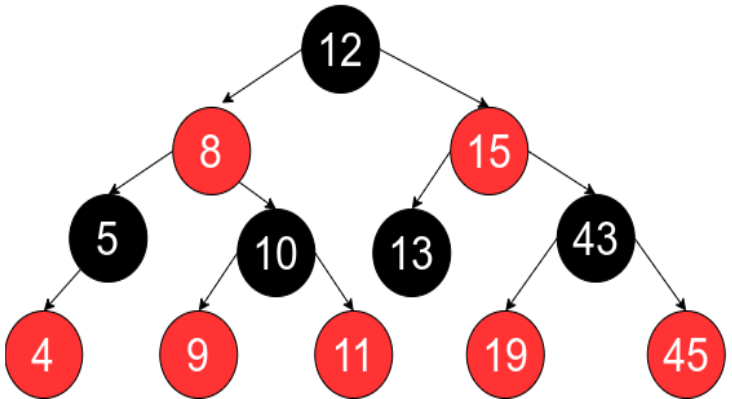
4)



5)
(3 случай)



6)
(4 случай)



Резюме

1. красно-чёрное дерево обладает теми же свойствами, что и обычное бинарное дерево поиска;
2. имеет дополнительный бит информации (цвет);
3. имеет фантомный чёрный лист;
4. балансировка производится посредством поворотов и перекраски;
5. после балансировки должны выполняться/сохраняться все свойства красно-чёрного дерева;

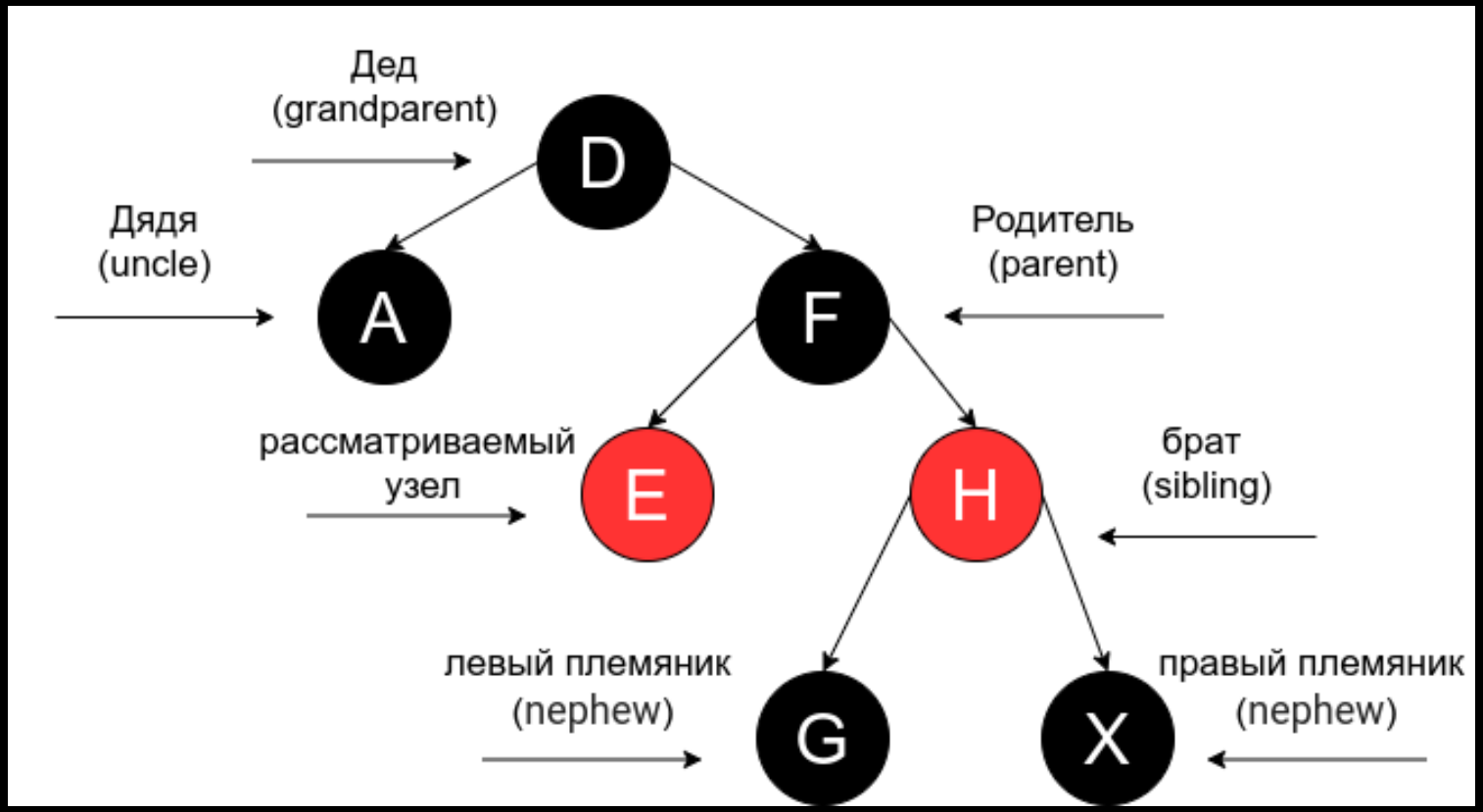
Попробуйте построить сами

1) 34, 25, -28, 10, 93, -25, 36, 90
2) -39, 37, -48, 19, -93, -96, -60, 97, 89
3) 50, 53, -29, 44, -51, 6, 79, 96, 70
4) -52, 28, -29, 31, 63, 46, 67, 70
5) 98, 33, 15, 85, 100, 92, 1, 9, 91, 12

Проверить себя можете здесь (плохо работает с отрицательными числами):
https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Алгоритм удаления из КЧД

К известным нам терминам КЧД добавляются такие понятия, как: брат, (левый/правый) племянник; исходя из которых выполняются соответствующие действия при удалении.



(Относительно узла E)

Алгоритм удаления из КЧД

- удаление производится по правилам БДП (нет детей — удаляем; 1 дитё — заменяем; два и более — берём либо максимальный из левого поддерева, либо минимальный из правого);
- после удаления необходимо производить балансировку дерева (посредством поворотов и перекраски узлов) **с учётом свойств КЧД**;
- при нарушении черной высоты, производится балансировка (т.е. балансировка потребуется только при удалении/смещении чёрного узла);
- **замещаемый узел (дочерний узел, которым заменяем удалённый) первоначально сохраняет цвет удалённого узла;**

Алгоритм удаления из КЧД

nil – (пустой!) листовой узел, всегда чёрный. **Участвует** в алгоритме удаления (и в родственных связях) **как** и **все остальные узлы**. Не удаляем.

При удалении узла Z (с замещением на узел X) возможны следующие **исходы**:

1) Z – красный, X – красный или *nil* -> балансировка не нужна;
2) Z – красный, X – чёрный -> необходима балансировка;
3) Z – чёрный, X – красный -> балансировка не нужна;
4) Z – чёрный, X – чёрный или *nil* -> необходима балансировка;

Для балансировки используются 4 **случая** (описаны после примеров):

1) узел X чёрный и его брат красный;
2) узел X чёрный и его брат чёрный, и оба племянника чёрные;
3) узел X чёрный и его брат чёрный, один из племянников красный **И** второй чёрный;
4) узел X чёрный и его брат чёрный, а один из племянников красный.

Note: если забыли как выглядит родство, посмотрите на 20ый слайд

Поудаляем из КЧД

По рассмотренному дереву из слайда 17.

Некоторые моменты опущены (e.g. буду указывать *nil* узлы, только по необходимости) для сокращения места.

Удалим

4, 10, 12, 5, 13, 43, 19

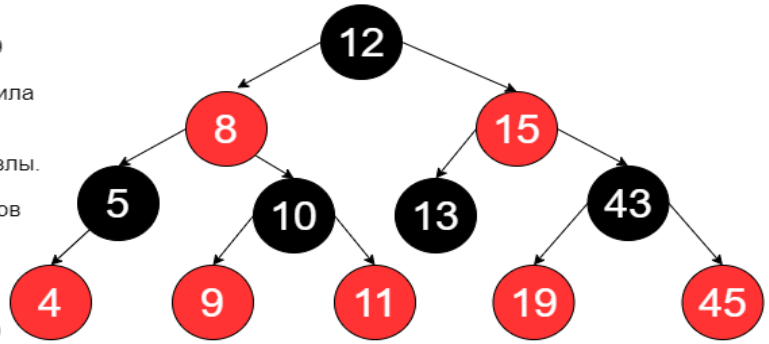
(не забываем, про правила для БДП)

* - не забываем про *nil* узлы.

** - описание подпунктов случаев будут в их пояснении.

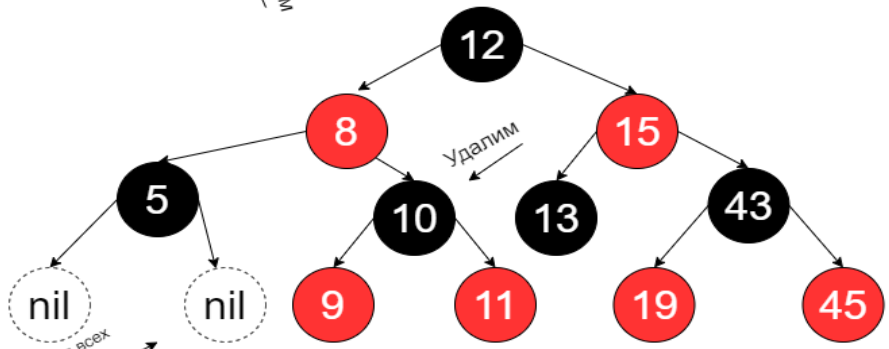
В описании, брат будет отмечаться как *S* (*sibling*)

1)



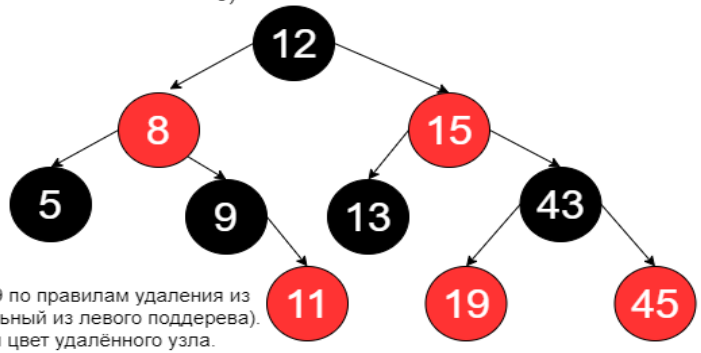
Удалим

2)



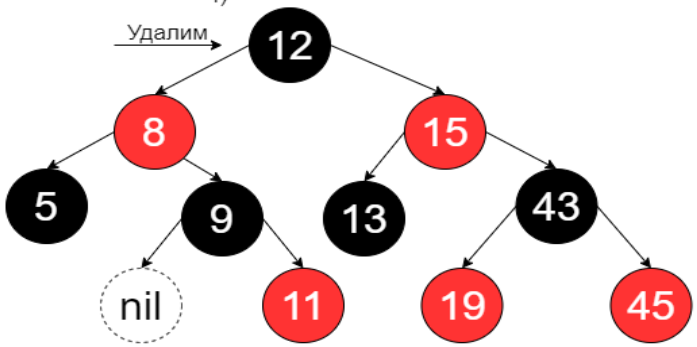
Существуют для всех концевых узлов

3)



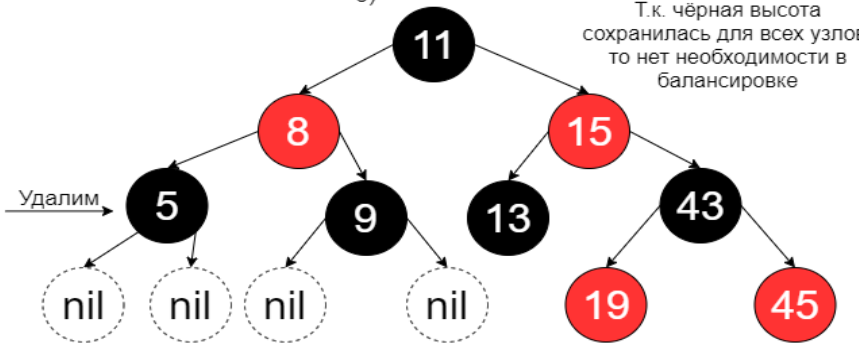
Заменили на 9 по правилам удаления из БДП (максимальный из левого поддерева). Сохранили цвет удалённого узла. (по сути, 3 исход)

4)*



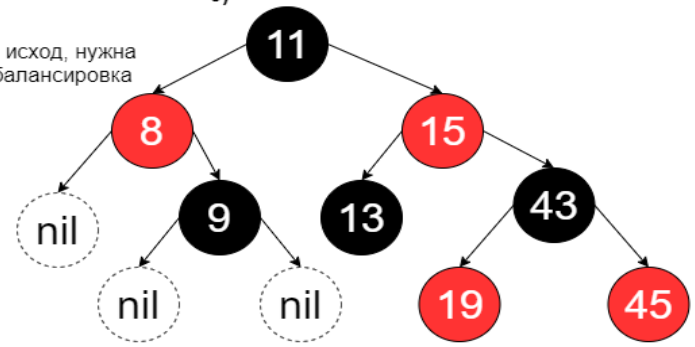
5)

Т.к. чёрная высота сохранилась для всех узлов, то нет необходимости в балансировке



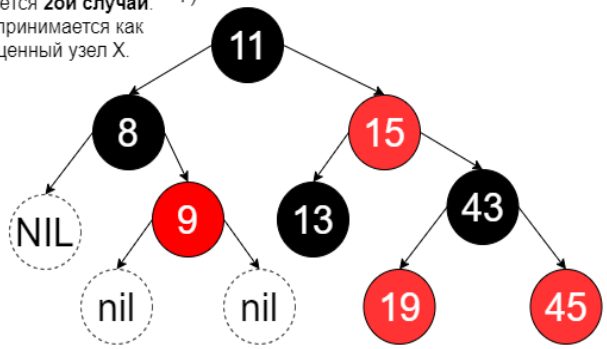
6)

4 исход, нужна балансировка



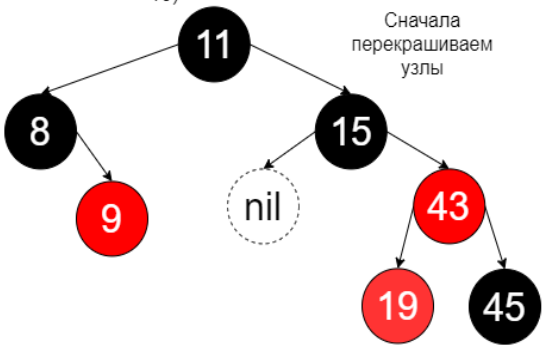
Для балансировки используется **2ой случай**. *NIL* воспринимается как полноценный узел X.

7)

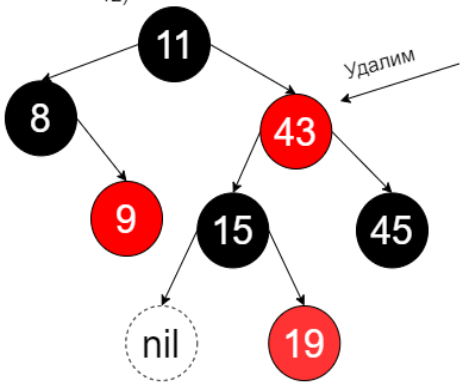


10) **

Сначала перекрашиваем узлы

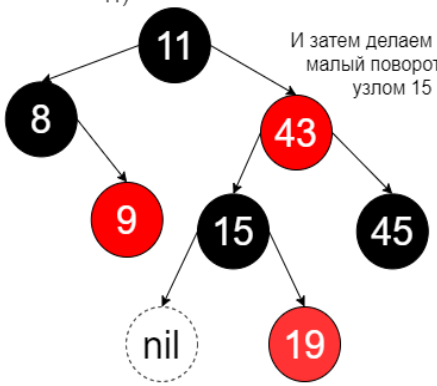


12)



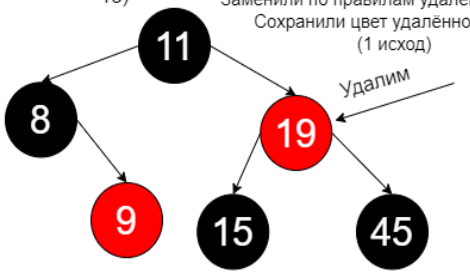
11)

И затем делаем левый малый поворот над узлом 15

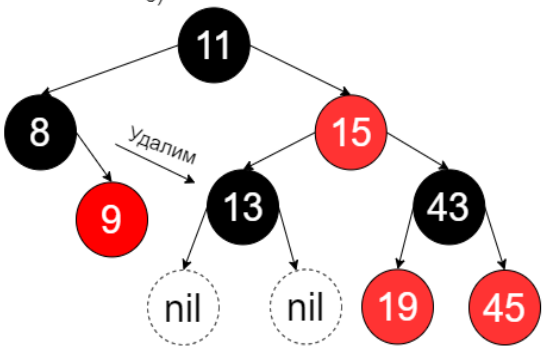


13)

Заменили по правилам удаления из БДП
Сохранили цвет удалённого узла.
(1 исход)

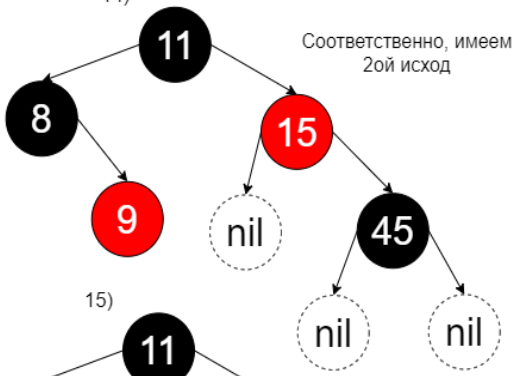


8)



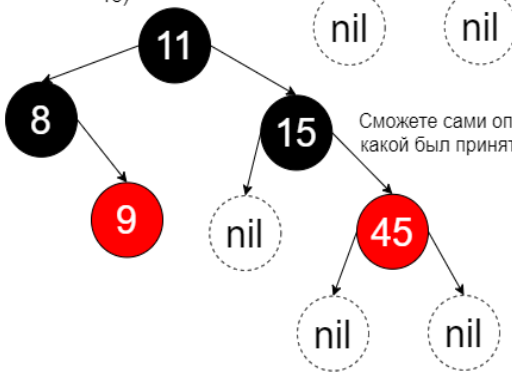
14)

Соответственно, имеем 2ой исход



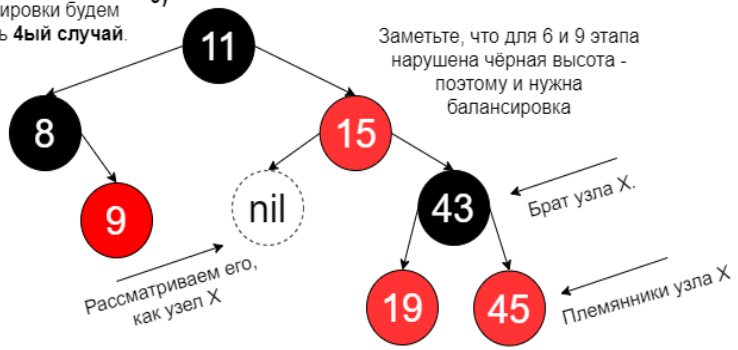
15)

Сможете сами определить, какой был принят **случай**?



Для балансировки будем использовать **4ый случай**.

9)



1 Случай

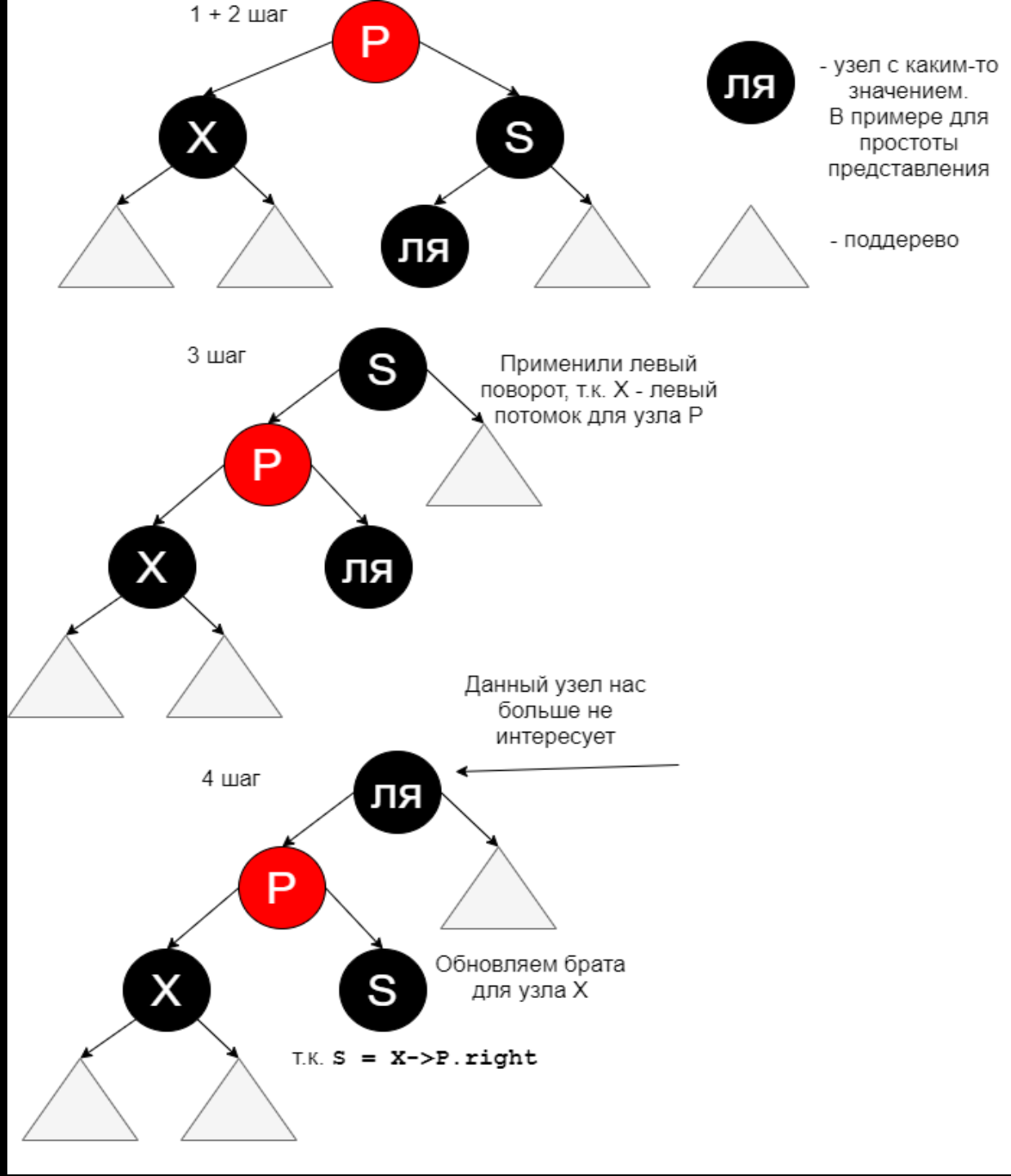
Возникает, когда узел X чёрный и его брат красный.

Здесь и далее, братские узлы будем отмечать как S (*sibling*), а родительский узел (у X и S один родитель) как P (*parent*).

Алгоритм решения:

1) красим узел S в чёрный;
2) красим узел P в красный;
3) применяем над узлом P:
   a) левый поворот, если X – левый потомок;
   b) правый поворот, если X – правый потомок;
4) т.к. после поворота узел P смещается вниз, а узел S наоборот поднимается, для узла X необходимо обновить брата, поэтому:
   a) если X – левый потомок, то узлом S будет правый узел родителя (`S = X->P.right`);
   b) если X – правый потомок, то узлом S будет левый узел родителя (`S = X->P.left`);
5) теперь с нашим X и новым S выбираем подходящий случай (2, 3 или 4)

Сжатая иллюстрация первого случая



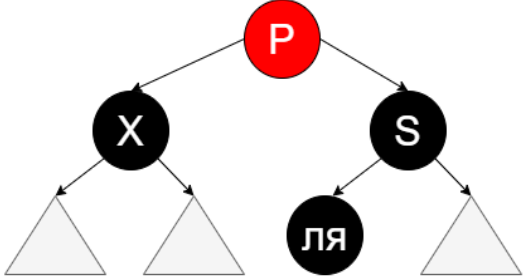
2 Случай

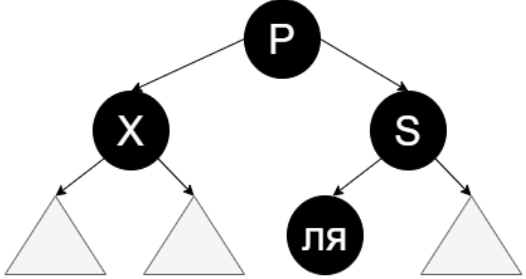
Возникает, когда узел X чёрный и его брат чёрный, и оба племянника чёрные.

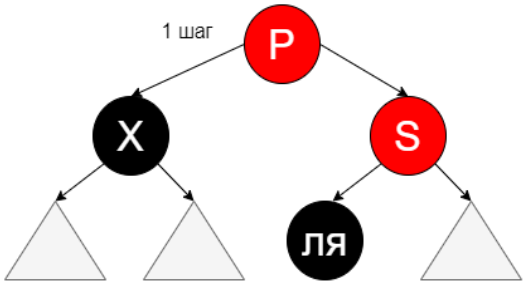
Алгоритм решения:

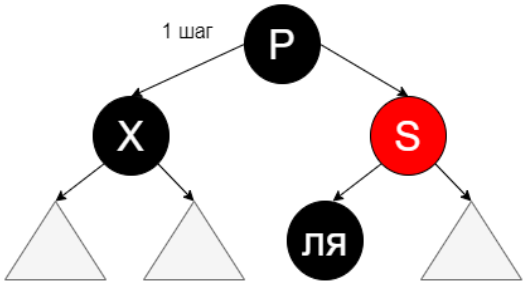
1) красим узел S в красный;
2) обновляем («поднимаем») узел X, поднимая его к родителю P (X = X->P);
   a) если X теперь красный, то красим в чёрный. Конец алгоритма.
   b) если X теперь чёрный, то выбираем подходящий случай (1, 2, 3 или 4). Заметьте, в таком случае мы будем иметь и новый S (как и P).

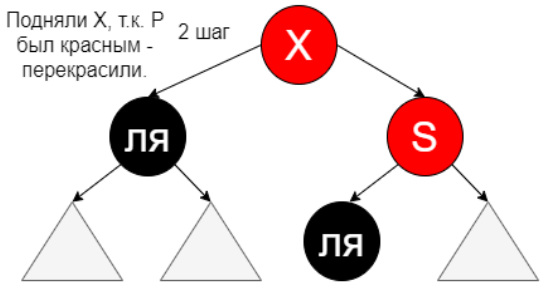
Сжатая иллюстрация второго случая

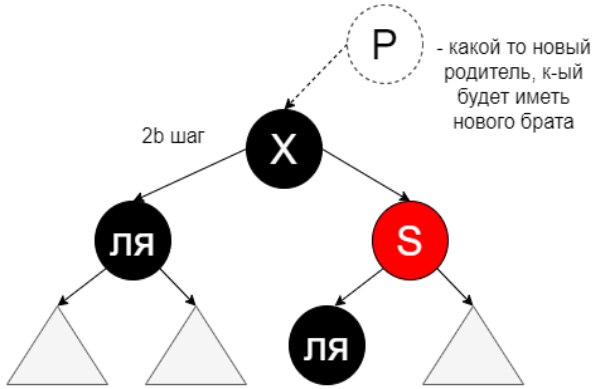


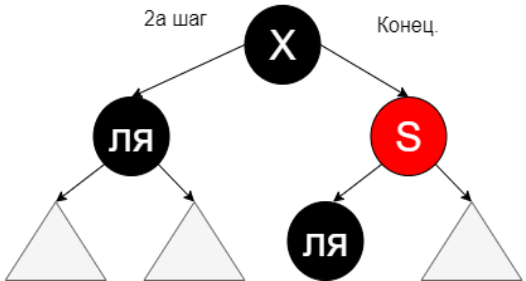


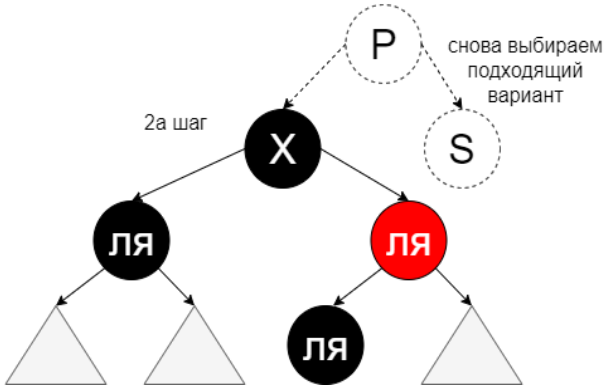












3 Случай

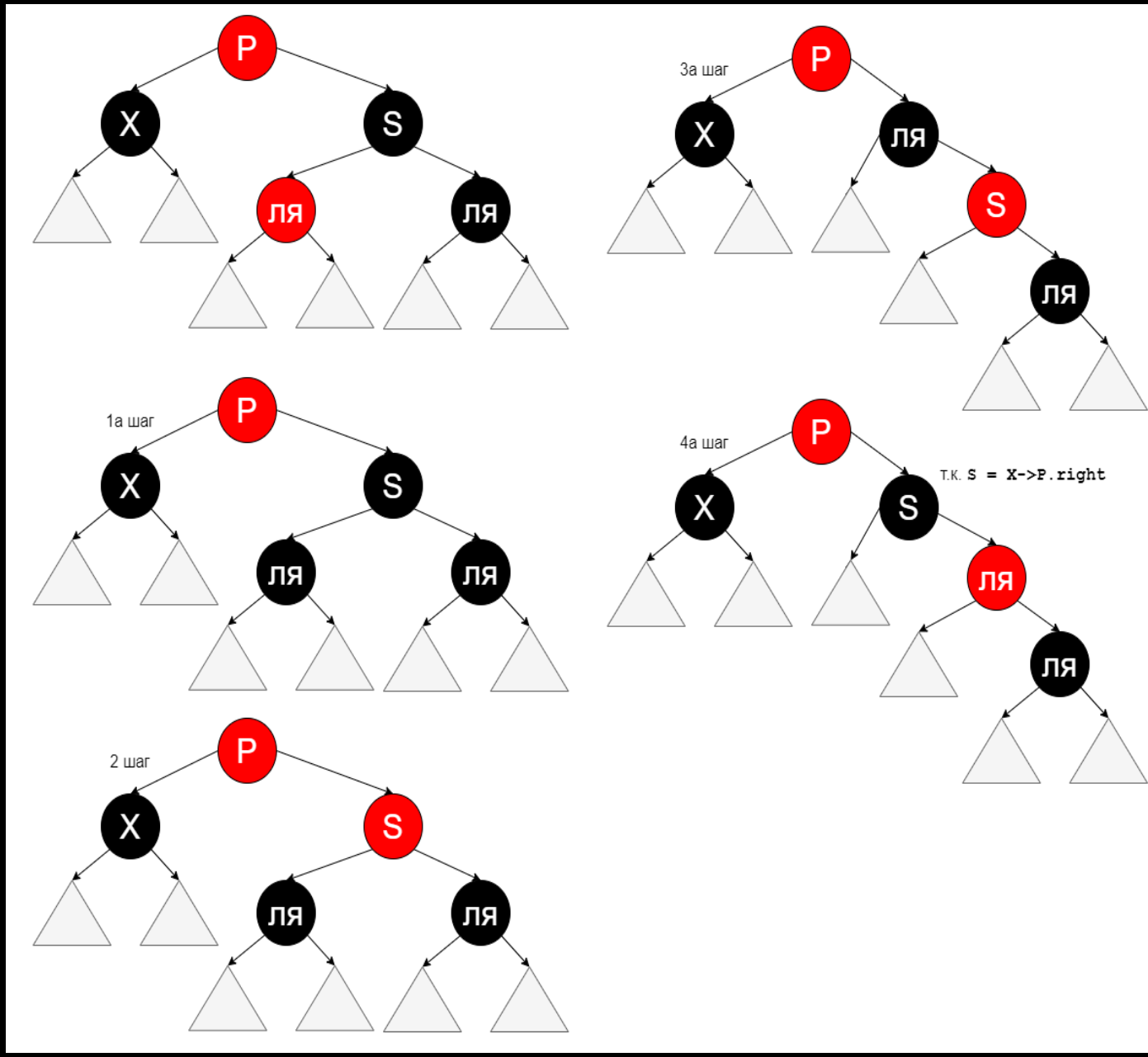
Возникает, когда узел X чёрный и его брат чёрный, при этом:
a) X левый потомок, имеет красного левого племянника и правого чёрного;
b) X правый потомок, имеет чёрного левого племянника и красного правого.

Алгоритм решения:

1) красим (здесь и далее, в зависимости от условия выше, выбираем соответствующий вариант):
   a) левого племянника в чёрный;
   b) правого племянника в красный;
2) красим S в красный;
3) применяем над узлом S:
   a) правый поворот;
   b) левый поворот;
4) обновляем S, т.к. узел X остаётся на месте, ему нужен актуальный брат:
   a) `S = X->P.right;`
   b) `S = X->P.left;`
5) выполняем случай (вариант/условие) 4.

Сжатая иллюстрация третьего случая

(для варианта a)



4 Случай

Возникает, когда узел X чёрный и его брат чёрный, при этом:
a) X левый потомок, имеет красного правого племянника;
b) X правый потомок, имеет красного левого племянника.

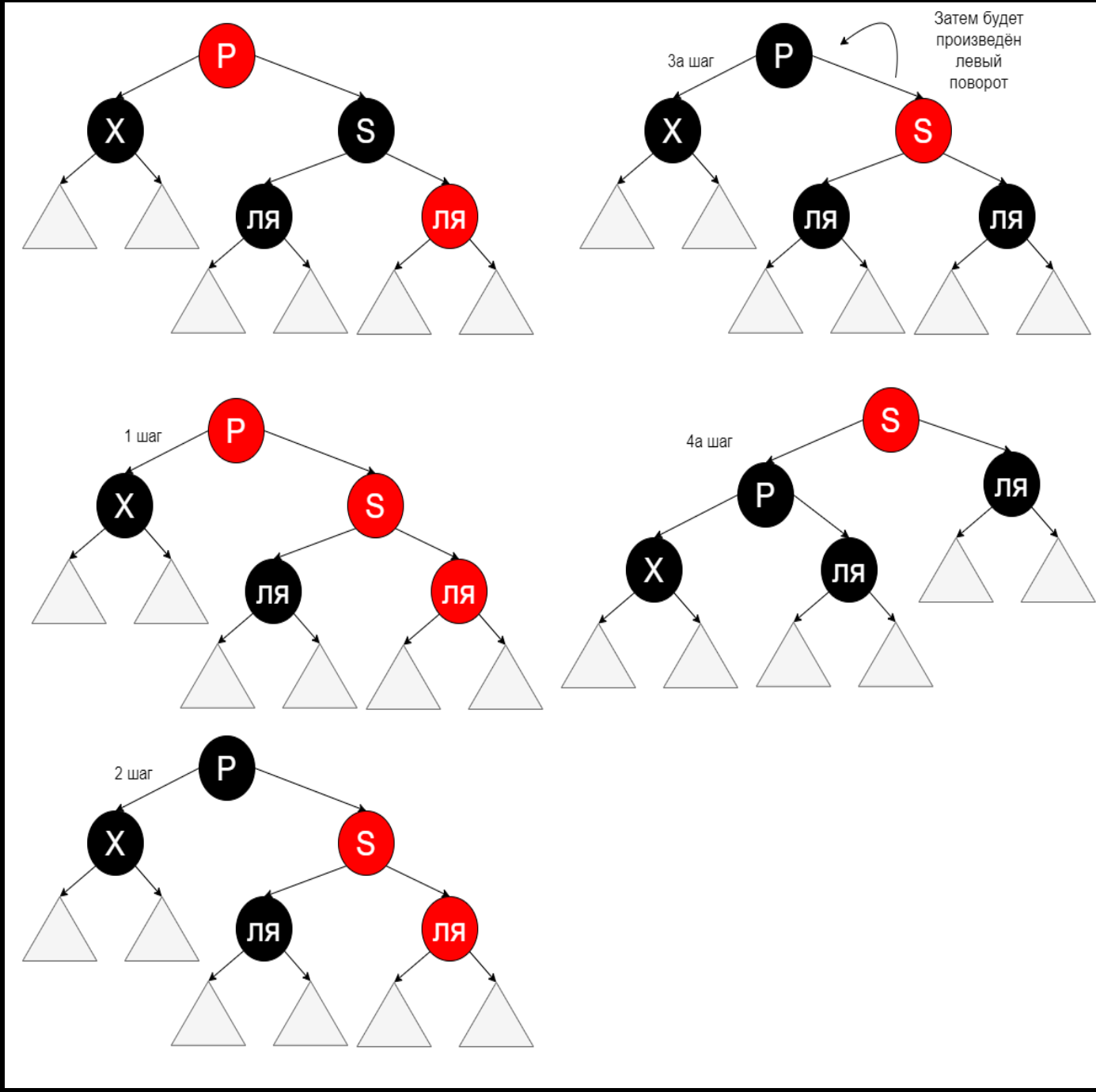
Алгоритм решения:

1) красим узел S в тот же цвет, что и узел P;
2) красим узел P в чёрный;
3) красим:
   a) правого племянника в чёрный;
   b) левого племянника в чёрный;
4) применяем над узлом P:
   a) левый поворот;
   b) правый поворот;
5) Конец.

В основном встречается, когда оба племянника – красные (но необязательно).

Сжатая иллюстрация четвёртого случая

(для варианта а)



Резюме

1. при добавлении оперируем цветом дяди;
2. при удалении оперируем цветом племянников и брата;
3. при удалении красной вершины (зачастую) нет необходимости в балансировке.

Попробуйте удалить

Из представленных ранее вариантов:
1) все чётные;
2) все отрицательные;
3) все нечётные;
4) все узлы, что больше 46;
5) все узлы, что меньше 90.

Проверить себя можете здесь (плохо работает с отрицательными числами):
https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

А что будет дальше?

Я как то вам уже говорил, что важнейшей частью вашего обучения (становления как профессионалов) является – самообучение.

Вы должны сами проявить желание в постижении (изучении) различных сфер вашей будущей деятельности, к сожалению, за вас это никто не сможет сделать.

Здесь я приведу примерный список тем, который вам стоит изучить в рамках курса СПО (и отдельно, в рамках изучения C++), по возможности так же приведу ссылки на ресурсы, где об этом можно почитать.

Возможно я и дальше буду составлять подобные небольшие презентации по СПО, но не стоит надеяться, что в них будет полностью раскрыта каждая из тем. В первую очередь, я рассчитываю на вашу самостоятельность.

Так же не забываем про лабораторные работы!

Темы

СПО

- куча (Heap), AVL-дерево, красно-чёрное дерево, алгоритмы балансировки;
- словарь , хеш-таблицы, свойства и реализация, коллизии и способы их решения;
- Формальные грамматики, контекстно-свободная грамматика, форма Бэкуса-Наура (БНФ, РБНФ);
- лексический анализатор, таблица лексем (идентификаторов).

C++

- lvalue, rvalue, prvalue, etc;
- move семантика (rvalue reference);
- умные указатели (smart pointers);
- итераторы (input/output/forward/bidirectional/random);
- STL контейнеры;
- static члены (методы и поля);
- constexpr.

Ссылки

- коротко о КЧД;
- разжёвано про КЧД;
- про КЧД с примерами кода на C;
- курс по структурам данных (почти must have);
- для визуализации изучаемых алгоритмов (деревьев);
- ответы на возникшие вопросы
- про типы (C++, стоит учесть, что понятия различаются в зависимости от стандарта);
- про move семантику (просто, без погружения).