Технология и методы программирования

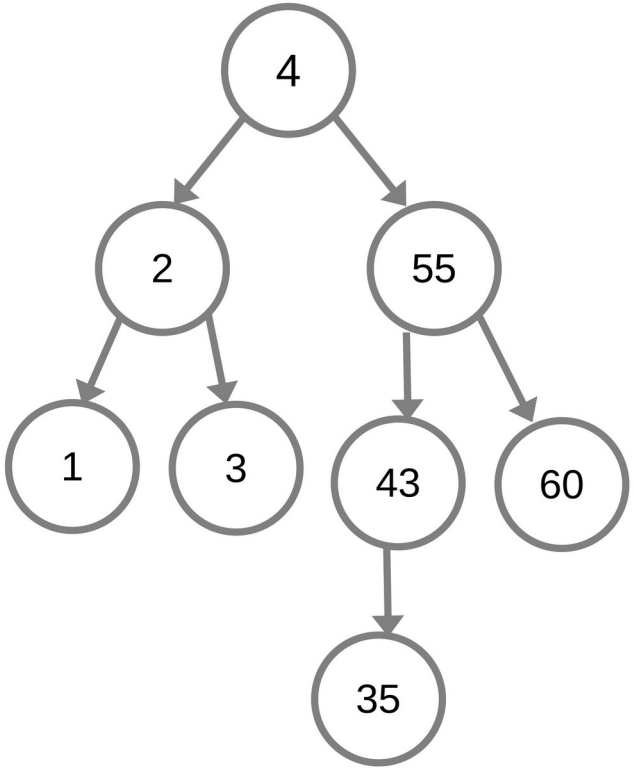
Тема 9. Деревья поиска.

Определения

- **Дерево** — иерархическая структура данных, состоящая из узлов и соединяющих их направленных рёбер.
- **Корень** — начальный узел
- **Лист** — узел, из которого не выходит ни одного ребра

Пример бинарного дерева

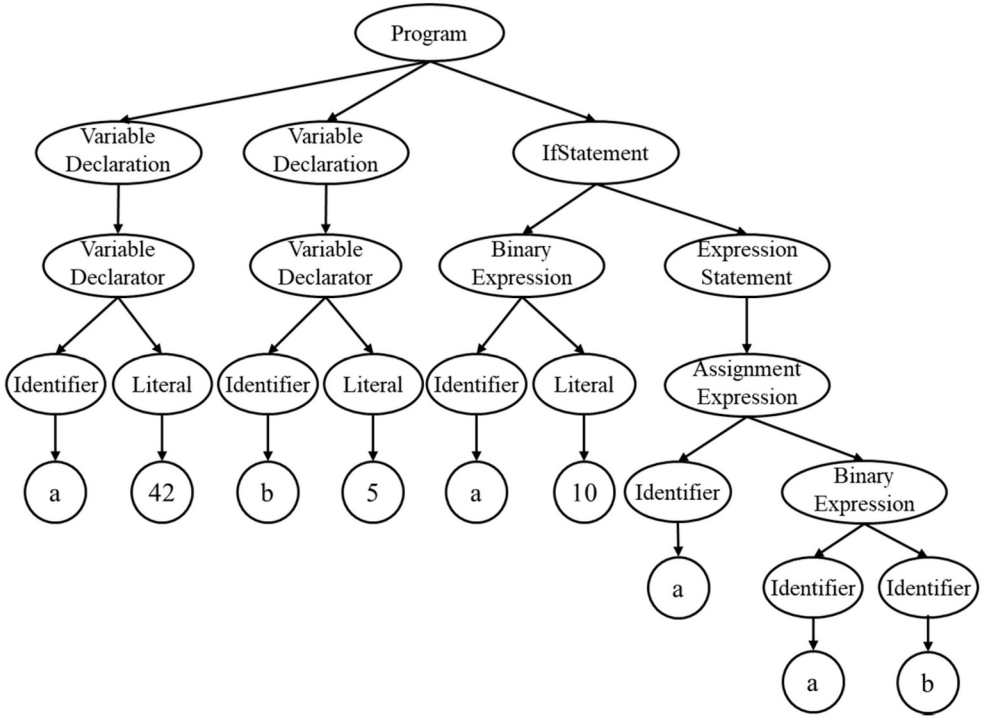
```
struct tree {
        struct tree_node *root;
};

struct tree_node {
        struct tree_node *parent;    //optional
        struct tree_node *left;
        struct tree_node *right;
        T value;
};
```



Пример синтаксического дерева

```
var a = 42
var b = 5
if(a > 10){
        a = a - b
}
```

(**a**)



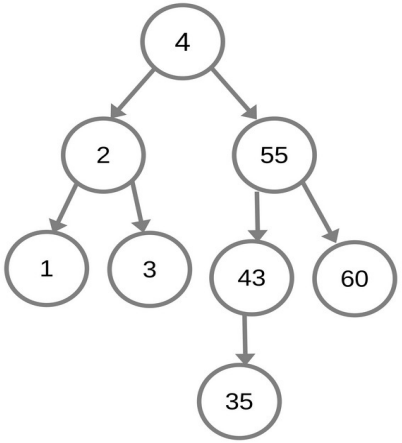
(**b**)

Где применяется?

- Парсеры языков
- Иерархическая файловая система
- В базах данных, для хранения индексов
- Компуктерные сети (таблицы маршрутизации)
- Деревья решений

Обход дерева

```
void tree_walk(struct tree_node *n)
{
        //YOURCODE
}
```




```
struct tree {
        struct tree_node *root;
};

struct tree_node {
        struct tree_node *parent;
        struct tree_node *left;
        struct tree_node *right;
        T value;
};
```

Обход дерева

3 Варианта обхода:

- Pre-order
- In-order
- Post-order

```
void tree_walk(struct tree_node *n)
{
        tree_walk(n->left);
        tree_walk(n->right);
}
```

Pre-order

In-order

Post-order

Обход дерева. Используемая память

- Расход памяти = O(h)
- Если есть указатель на parent => расход памяти O(1)

Бинарное дерево поиска

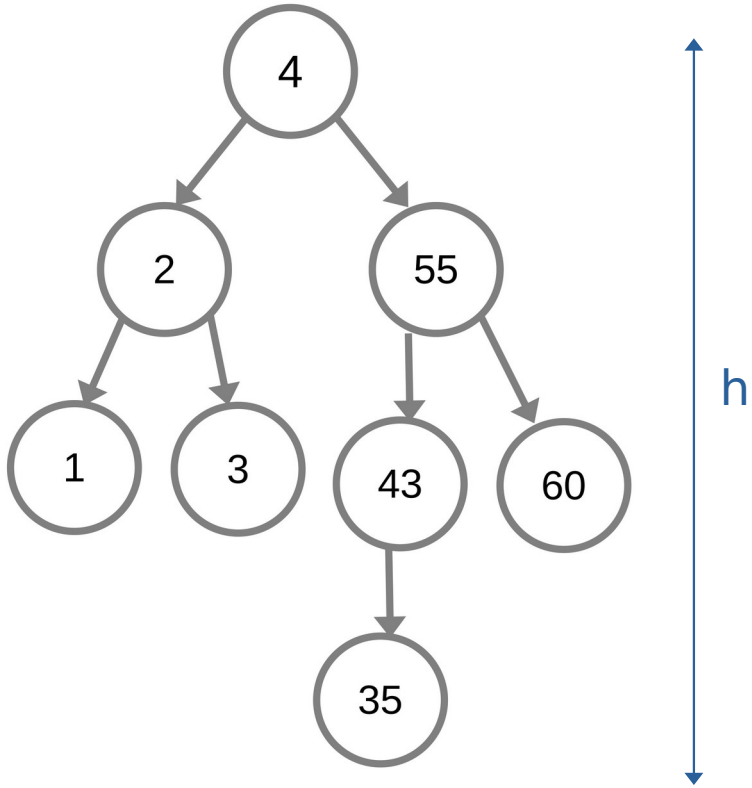
- Бинарное дерево поиска

  Если **x** узел дерева, то все элементы в левом поддереве <= **x**, а все элементы в правом поддереве >= **x**.

Бинарное дерево поиска

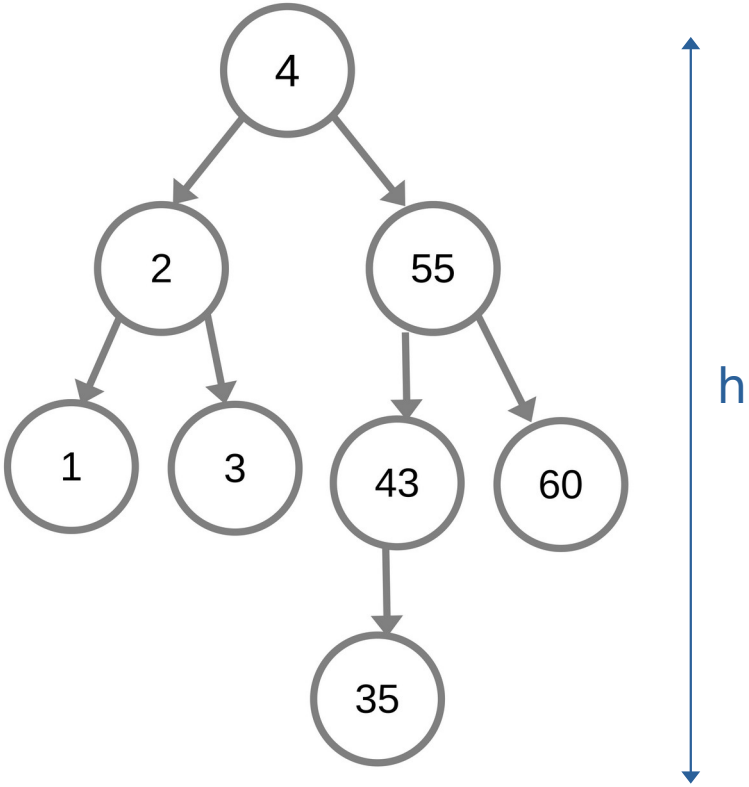
Операции:

- **min()**
- **max()**
- **sort()**
- search(x)
- successor(x)
- predecessor(x)



Бинарное дерево поиска

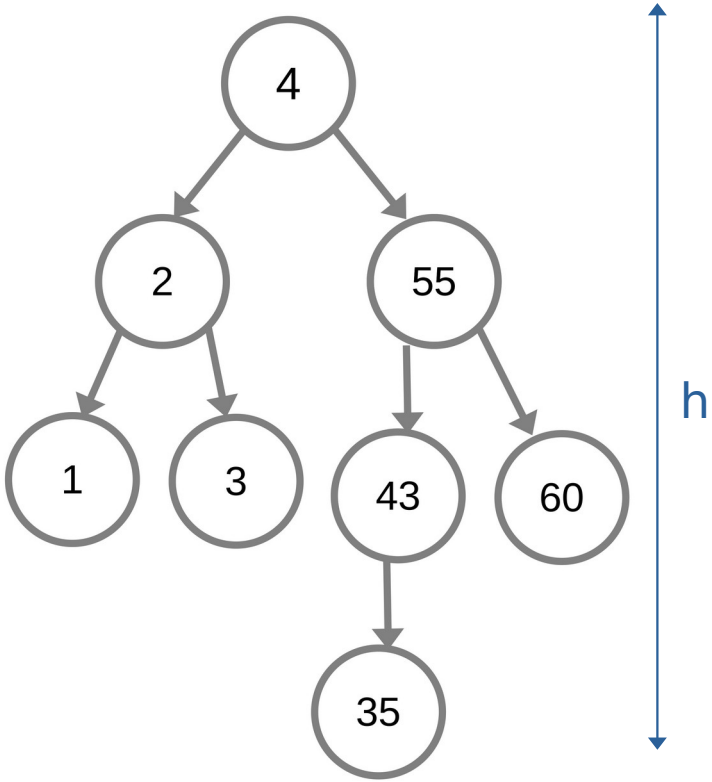
search(x)



Бинарное дерево поиска

search(tree, x)

- Начинаем поиск с корня
- x меньше текущего элемента =>
  идём в левое поддерево
- x больше текущего элемента =>
  идём в правое поддерево
- Совпадение => поиск закончен успешно
- Нет вершины => поиск закончен неуспешно

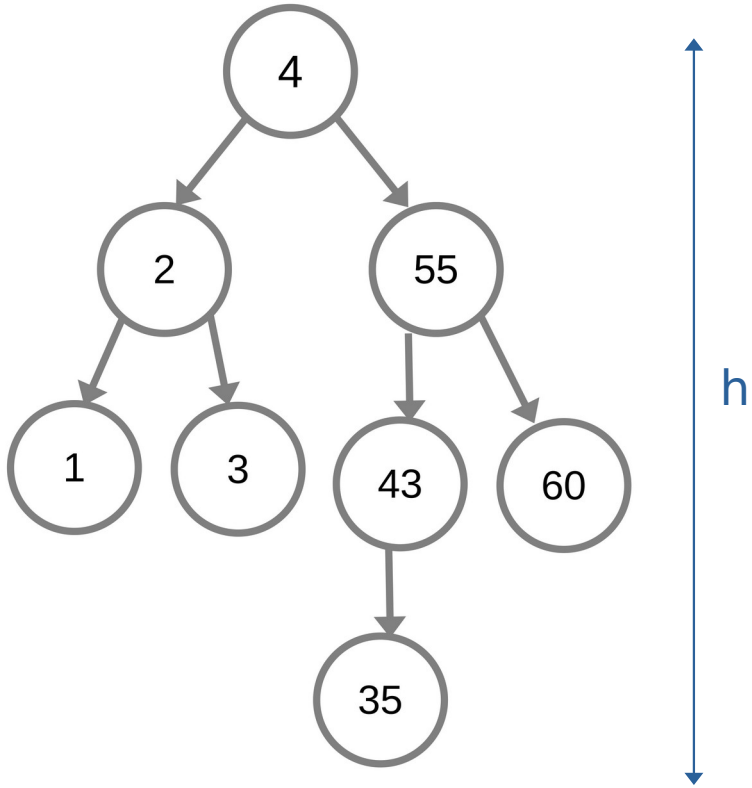


T = O(h)

Бинарное дерево поиска

successor(x)

Разбор случаев

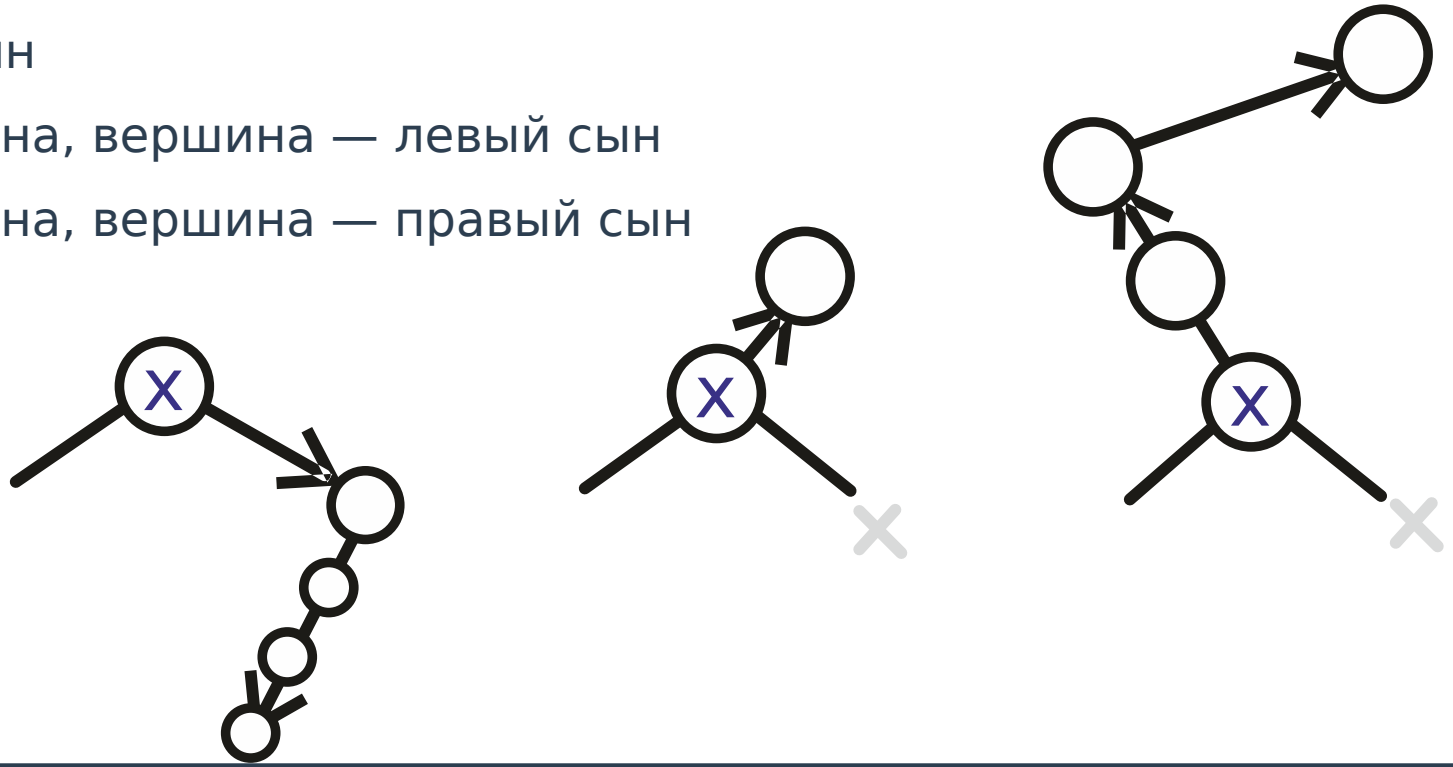


Бинарное дерево поиска

successor(x)

Разбор случаев

- Есть правый сын
- Нет правого сына, вершина — левый сын
- Нет правого сына, вершина — правый сын



Бинарное дерево поиска. Вставка и удаление

```
insert(x) {
    pos = search(x)
    pos.addNode(x)
}
```

```
erase(x) {
    pos = search(x)
    if (pos.r) {
        next = succ(pos)
        swap(next, pos)
    }
    pos.deleteNode(x)
}
```

Сбалансированное дерево поиска

- Требование $h = O(\log(n))$
- Достаточное условие:
  - Для любых вершин x, y: $ch(x) \leq 1$, $ch(y) \leq 1$, верно что $h(x)/h(y) \leq C$

Красно-чёрное дерево

- Каждая вершина имеет цвет {**R**, **B**}
- Добавим «фиктивные» листья там, где нет ребёнка

Свойства:

- Если **x** — лист => c(**x**) = **B**
- Если **y** родитель **x**, c(**x**) = **R** => c(**y**) = **B**
- Для всех листьев число чёрных предков одинаково (обозначим число чёрных предков **bh**)

Красно-чёрное дерево

- Каждая вершина имеет цвет {**R**, **B**}
- Добавим «фиктивные» листья там, где нет ребёнка

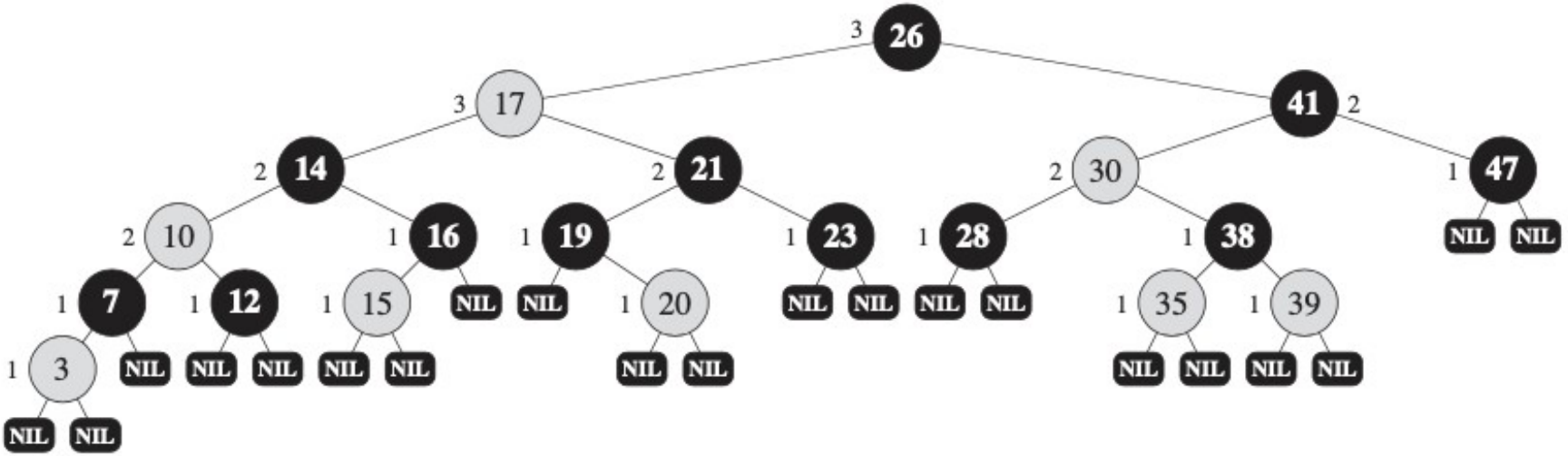
Свойства:

- Если **x** — лист => c(**x**) = **B**
- Если **y** родитель **x**, c(**x**) = **R** => c(**y**) = **B**
- Для всех листьев число чёрных предков одинаково (обозначим число чёрных предков **bh**)

Длина любого пути в дереве от **bh** до 2 * **bh**.

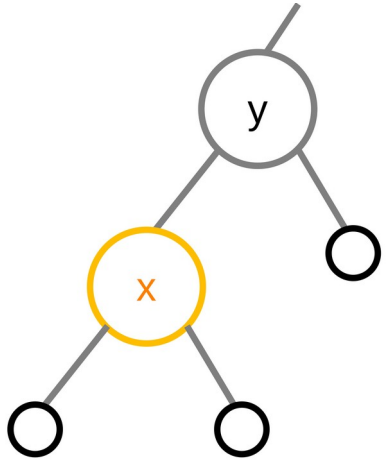
  - Следовательно h = O(log(n))

Красно-чёрное дерево. Пример



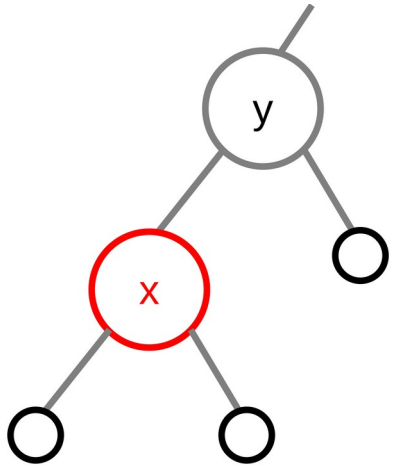
Красно-чёрное дерево. Вставка

- Добавляем к **y** сына **x**
  - Как покрасить **x**?



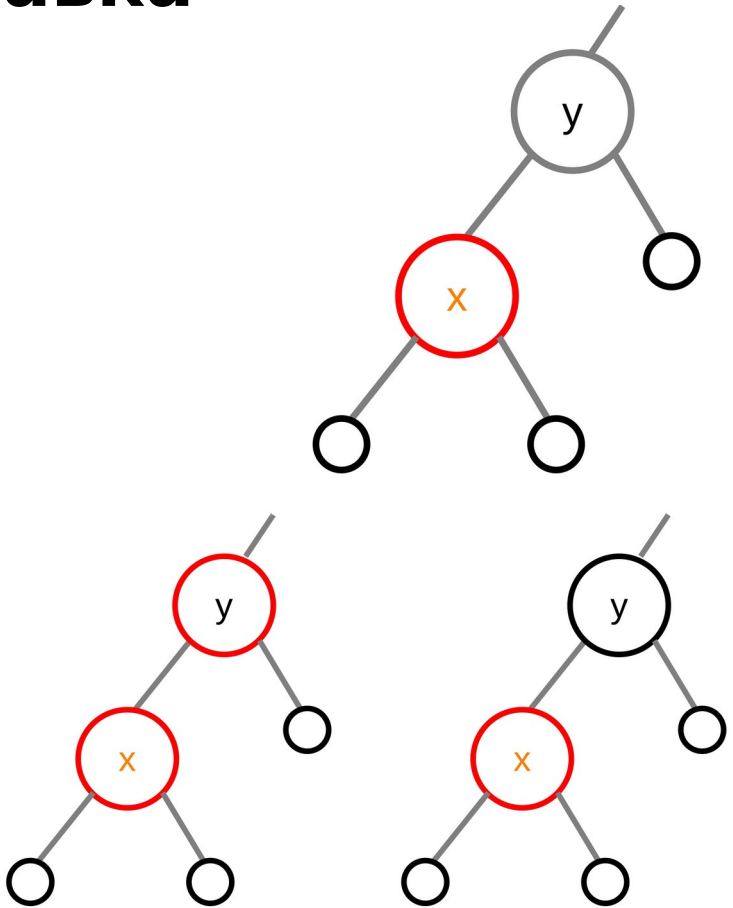
Красно-чёрное дерево. Вставка

- Добавляем к **y** сына **x**
  - c(**x**) = **R**



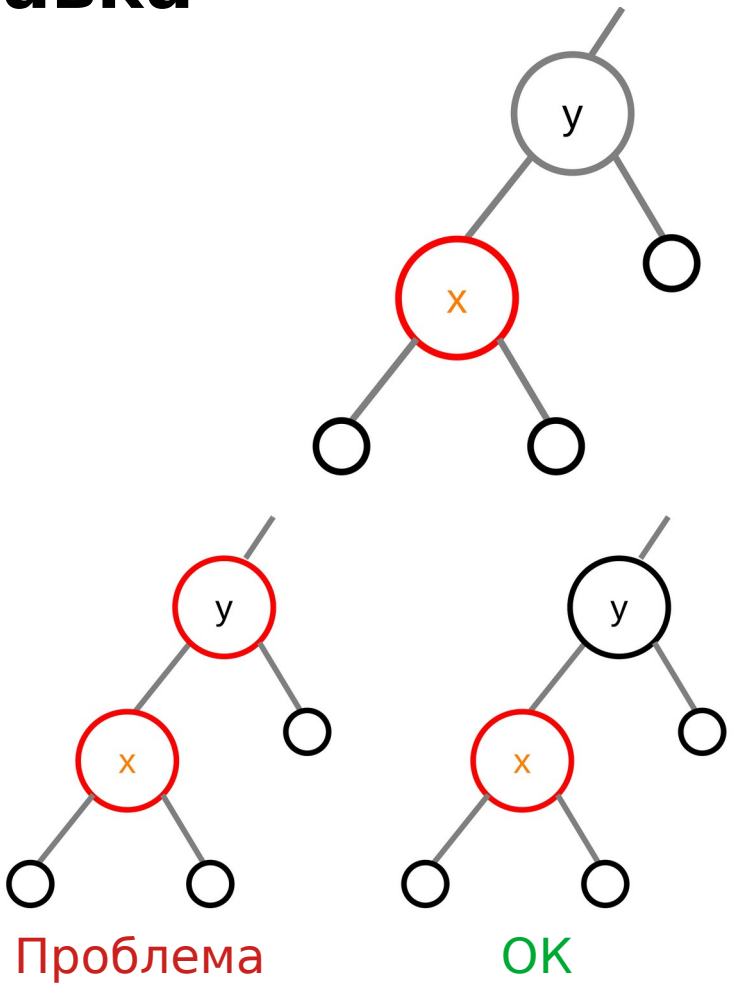
Красно-чёрное дерево. Вставка

- Добавляем к **y** сына **x**
  - c(**x**) = **R**
- Два варианта c(**y**)



Красно-чёрное дерево. Вставка

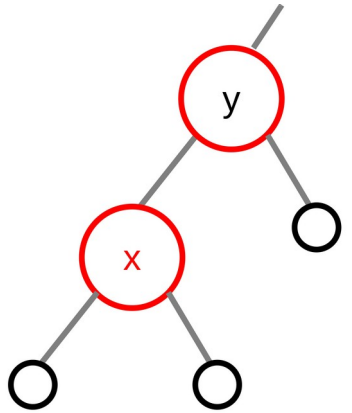
- Добавляем к **y** сына **x**
  - c(**x**) = **R**
- Два варианта c(**y**)
  - c(**y**) = **B** — OK
  - c(**y**) = **R** — Проблема



Красно-чёрное дерево. Вставка

Устранение проблемы

c(**x**) == c(**y**) == **R**



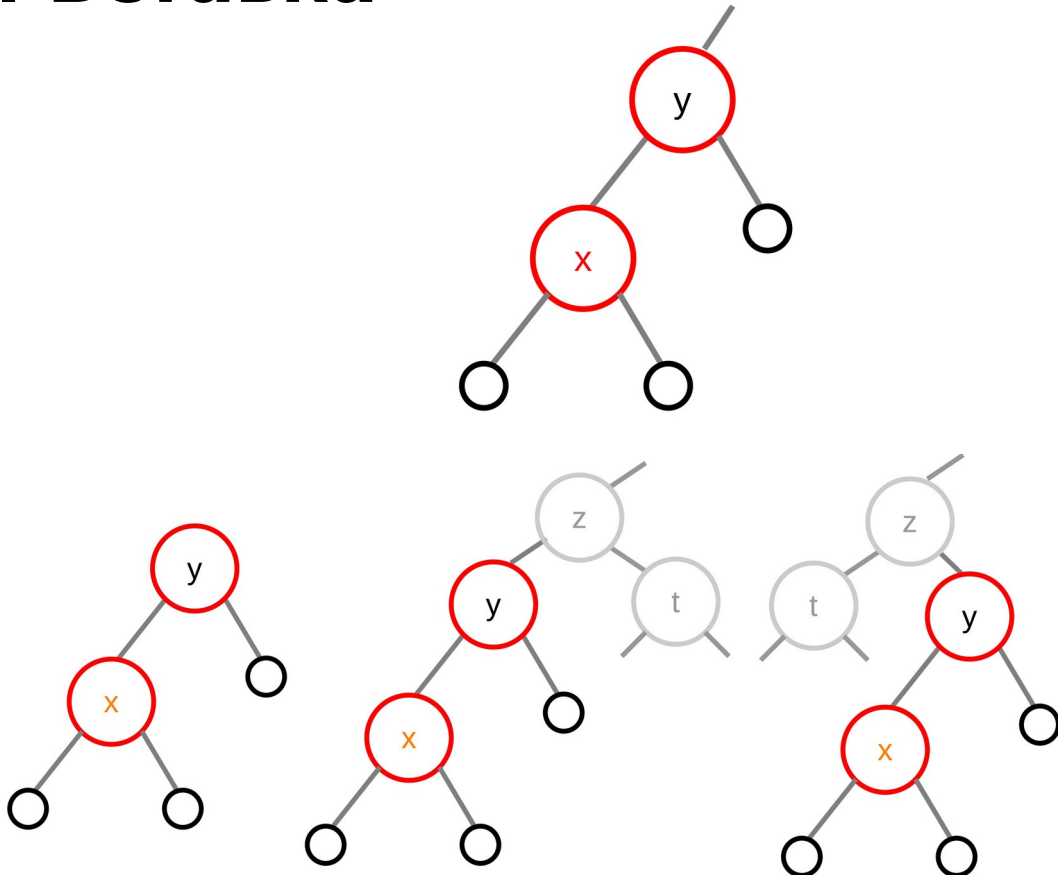
Красно-чёрное дерево. Вставка

Устранение проблемы

c(**x**) == c(**y**) == **R**

Варианты:

- y — корень
- y — левый или правый сын

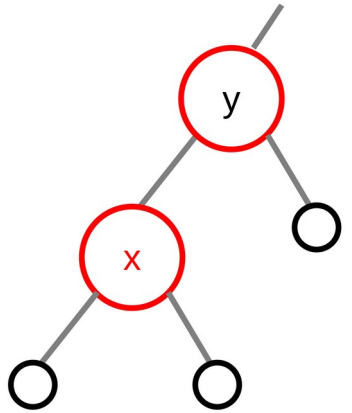


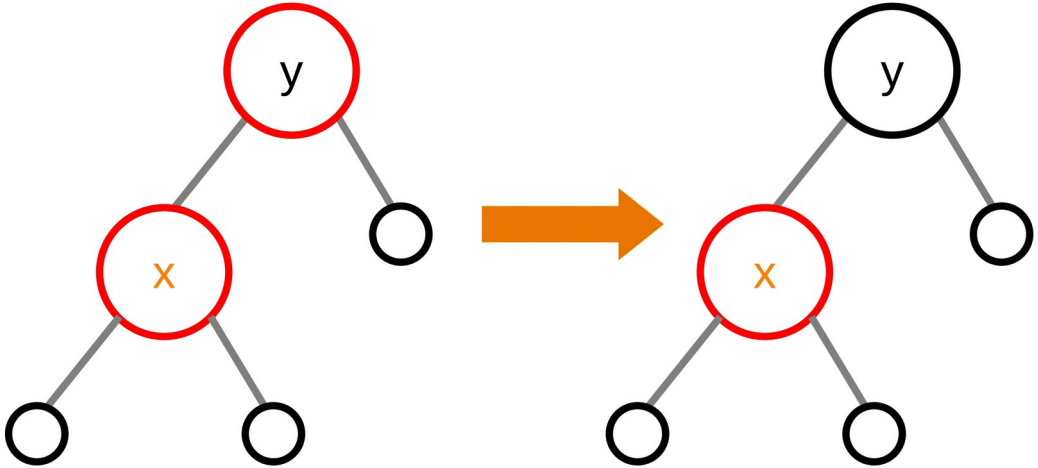
Красно-чёрное дерево. Вставка

Устранение проблемы

c(**x**) == c(**y**) == **R**, y — корень

- C(**y**) = **B**



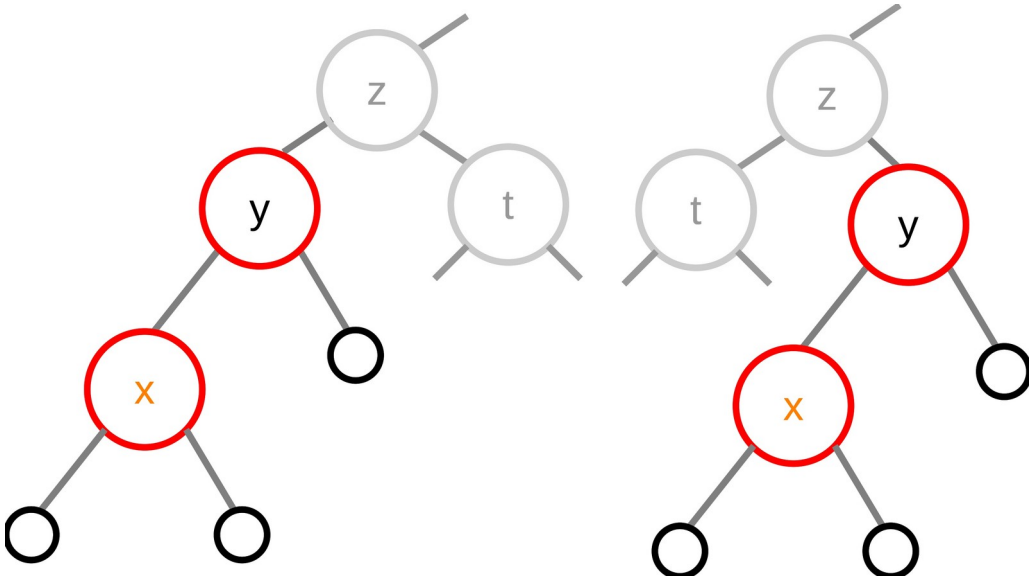


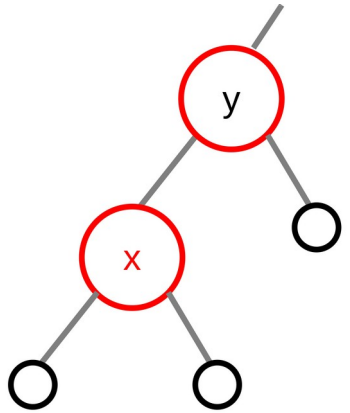
Красно-чёрное дерево. Вставка

Устранение проблемы

c(**x**) == c(**y**) == **R**, y — не корень

z — дед **x**, c(**z**) == ?



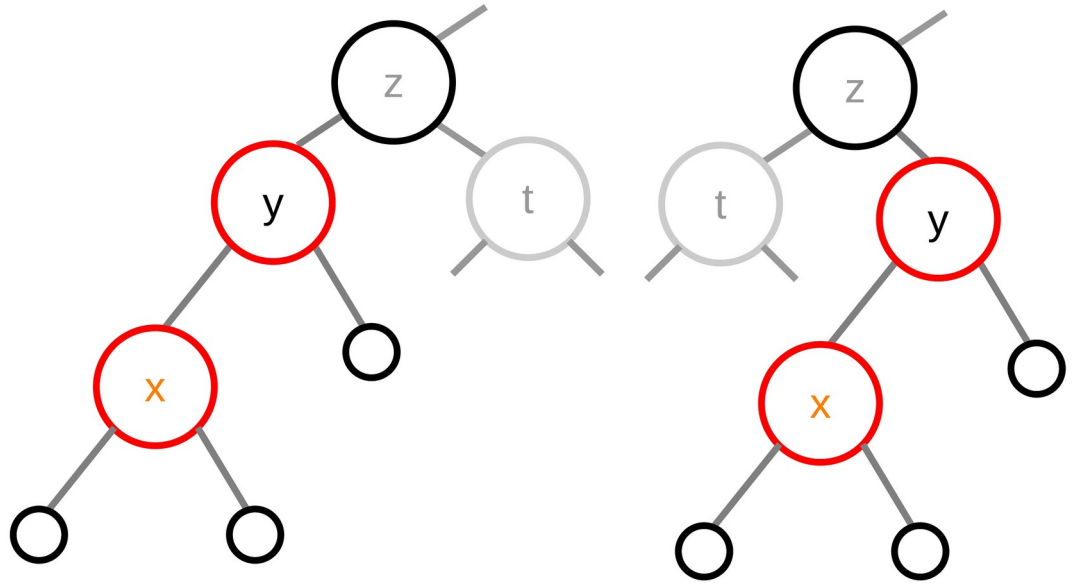


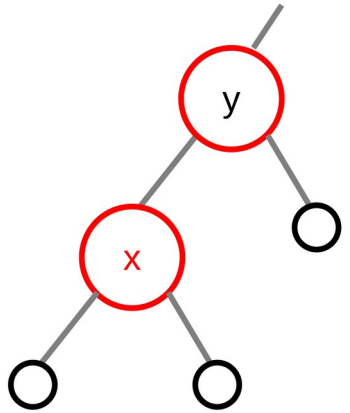
Красно-чёрное дерево. Вставка

Устранение проблемы

c(**x**) == c(**y**) == **R**, y — не корень

z — дед **x**, c(**z**) == **B**





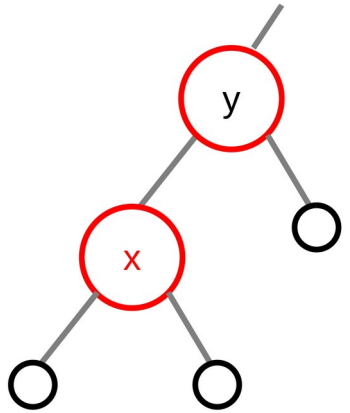
Красно-чёрное дерево. Вставка

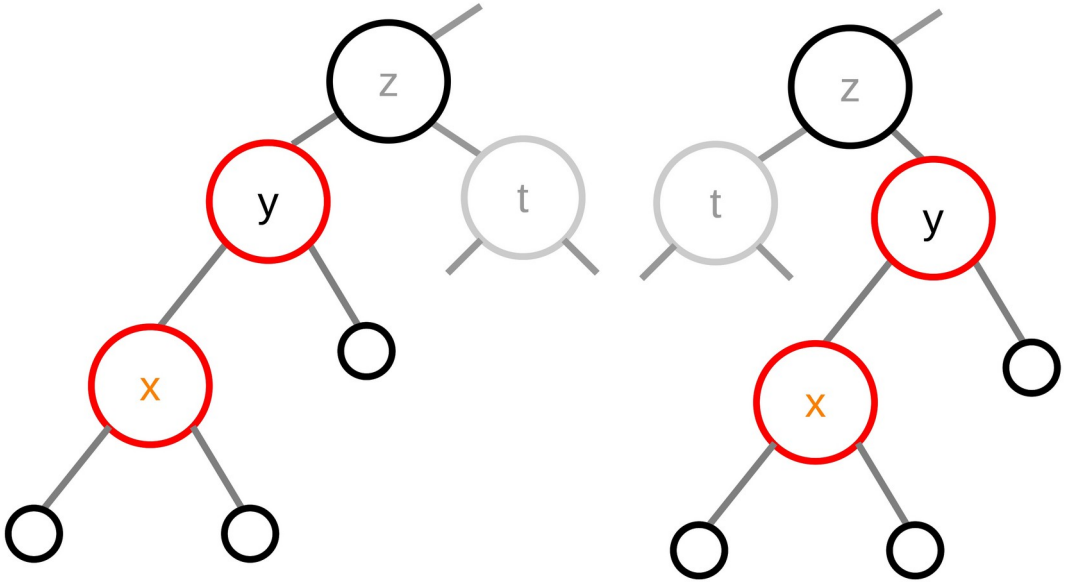
Устранение проблемы

c(**x**) == c(**y**) == **R**, y — не корень

z — дед **x**, c(**z**) == **B**

То, как сбалансировать дерево зависит от **t**





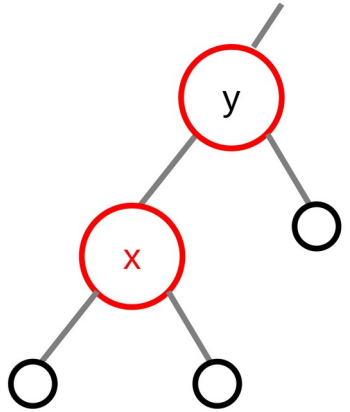
Красно-чёрное дерево. Вставка

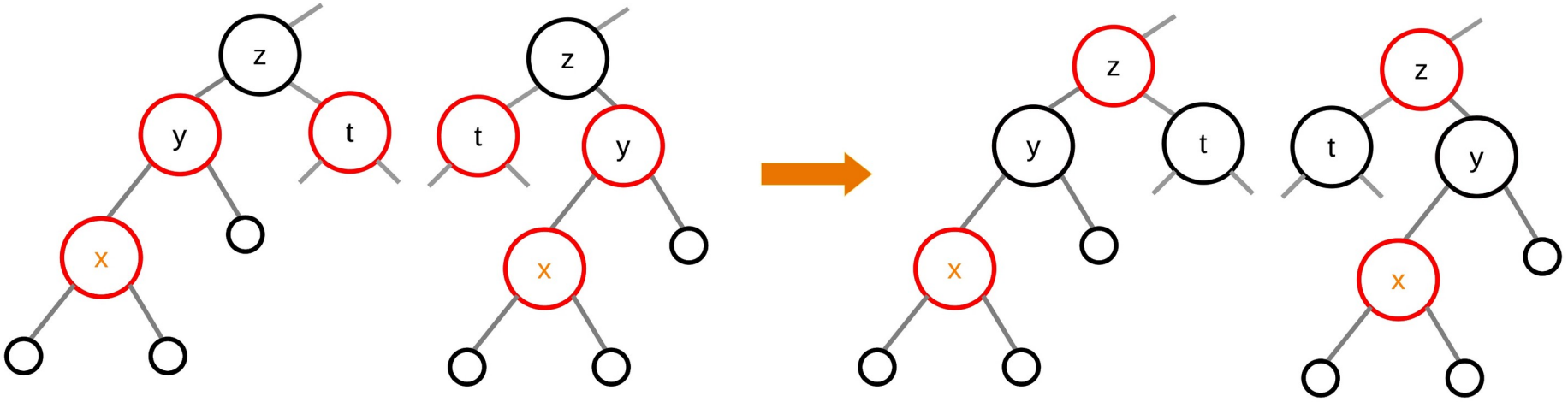
Устранение проблемы

c(**x**) == c(**y**) == **R**, y — не корень

z — дед **x**, c(**z**) == **B**, c(**t**) == **R**

- Перекрасим c(**z**) = **R**; c(**y**) = c(**t**) = **B**



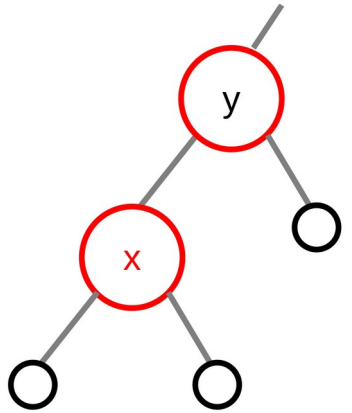


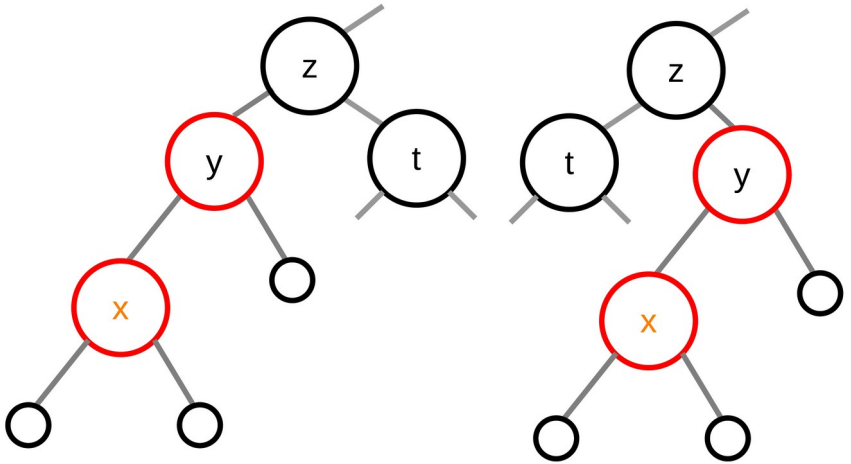
Красно-чёрное дерево. Вставка

Устранение проблемы

c(**x**) == c(**y**) == **R**, y — не корень

z — дед **x**, c(**z**) == c(**t**) == **B**





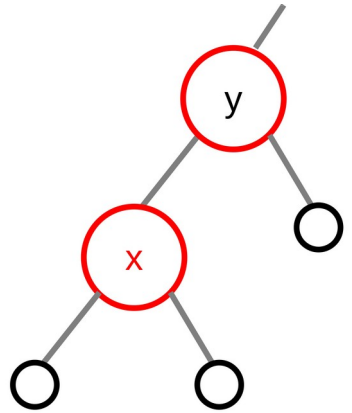
Красно-чёрное дерево. Вставка

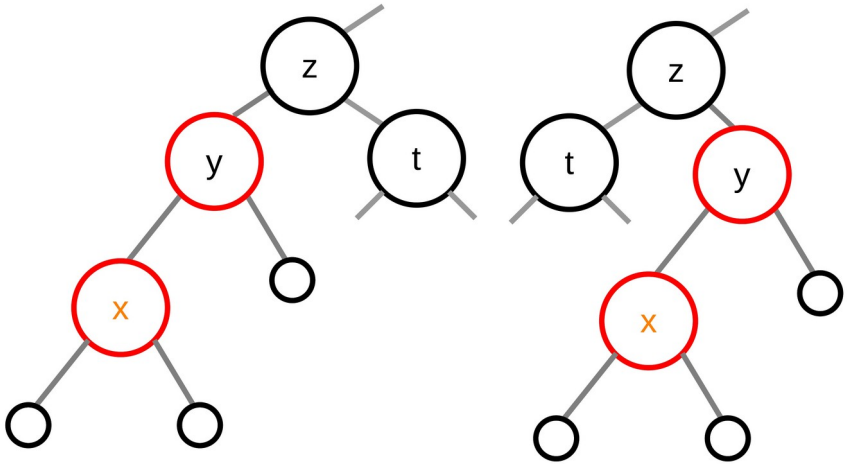
Устранение проблемы

c(**x**) == c(**y**) == **R**, y — не корень

z — дед **x**, c(**z**) == c(**t**) == **B**

- В этом случае перекраска вершин не поможет
- Нужно «повернуть» поддерево!





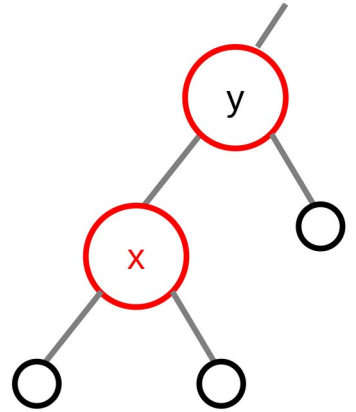
Красно-чёрное дерево. Вставка

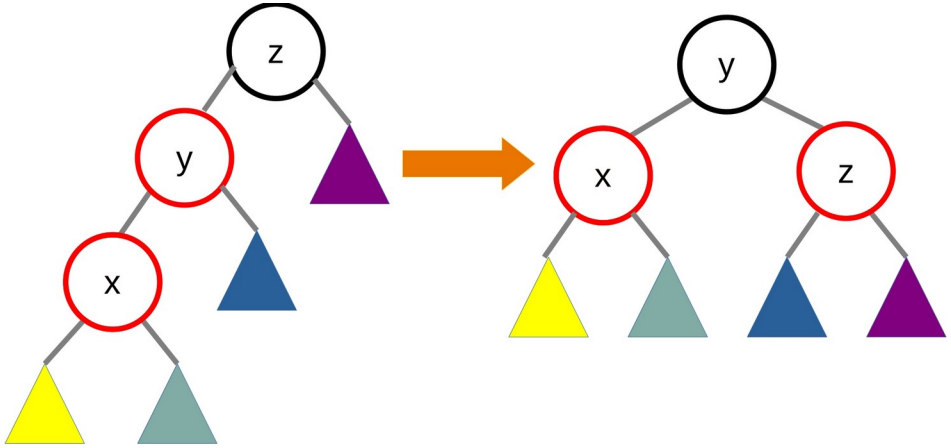
Устранение проблемы

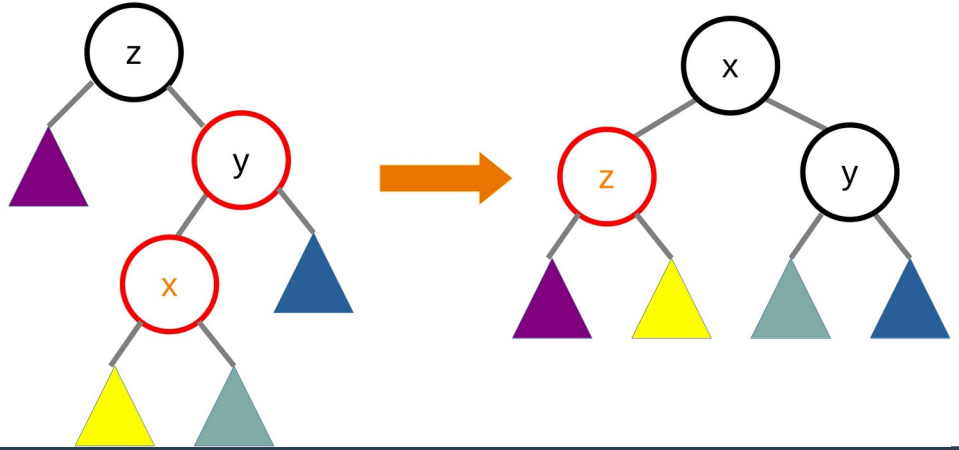
c(**x**) == c(**y**) == **R**, y — не корень

z — дед **x**, c(**z**) == c(**t**) == **B**

- В этом случае перекраска вершин не поможет
- Нужно «повернуть» поддерево!







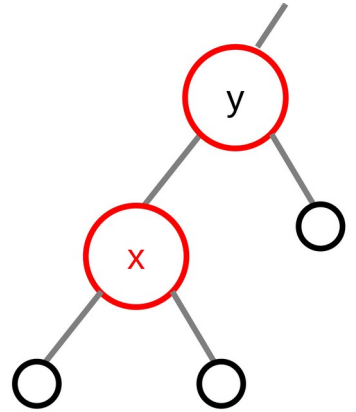
Красно-чёрное дерево. Вставка

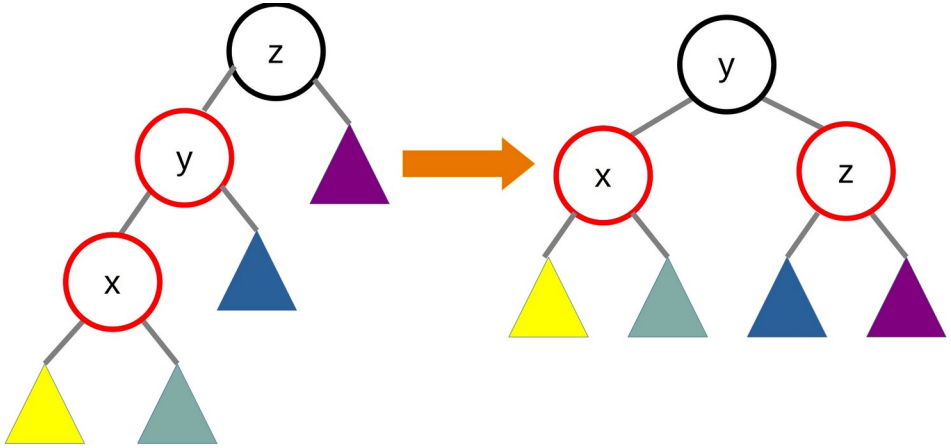
Устранение проблемы

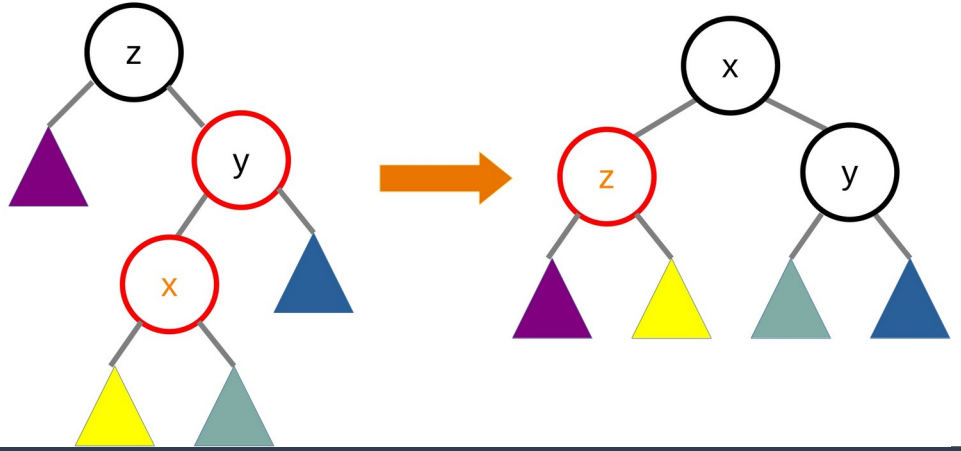
c(**x**) == c(**y**) == **R**, y — не корень

z — дед **x**, c(**z**) == c(**t**) == **B**

- В этом случае перекраска вершин не поможет
- Нужно «повернуть» поддерево!
- После поворота новых конфликтов выше не образуется.







Красно-чёрное дерево. Сложность

- T(insert)= O(h) = O(log(n))
- T(remove) = O(h) = O(log(n))

Дополнительные материалы

- Деревья поиска (youtube лекция)