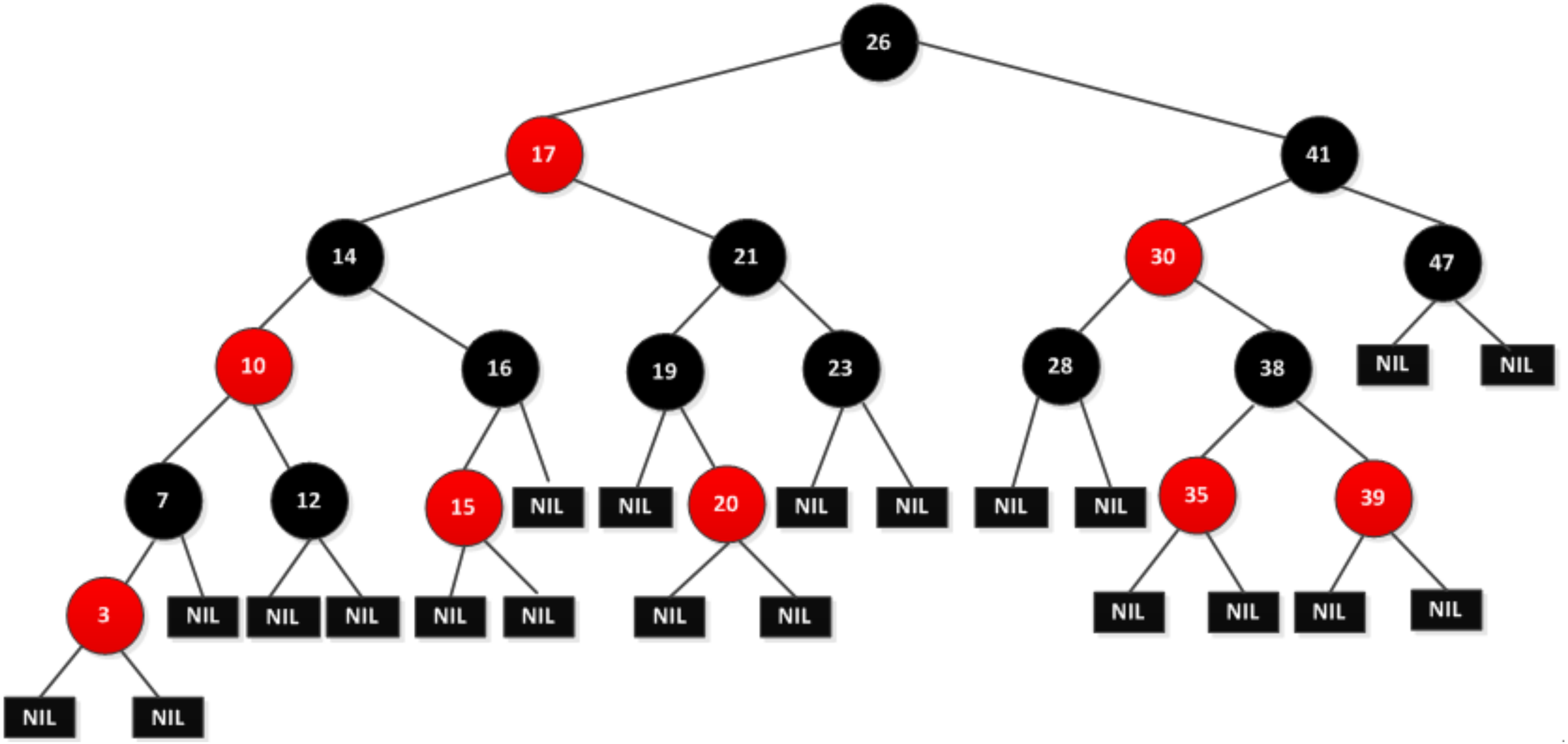
玩儿转数据结构

liuyubobobo

红黑树

什么是红黑树

红黑树


26
17
41
14
21
30
47
10
16
19
23
28
38
NIL
NIL
7
12
15
NIL
NIL
20
NIL
NIL
NIL
NIL
35
39
3
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL

《算法导论》中的红黑树

A red-black tree is a binary tree that satisfies the following ***red-black properties***:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

《算法导论》中的红黑树

1．每个节点或者是红色的，或者是黑色的

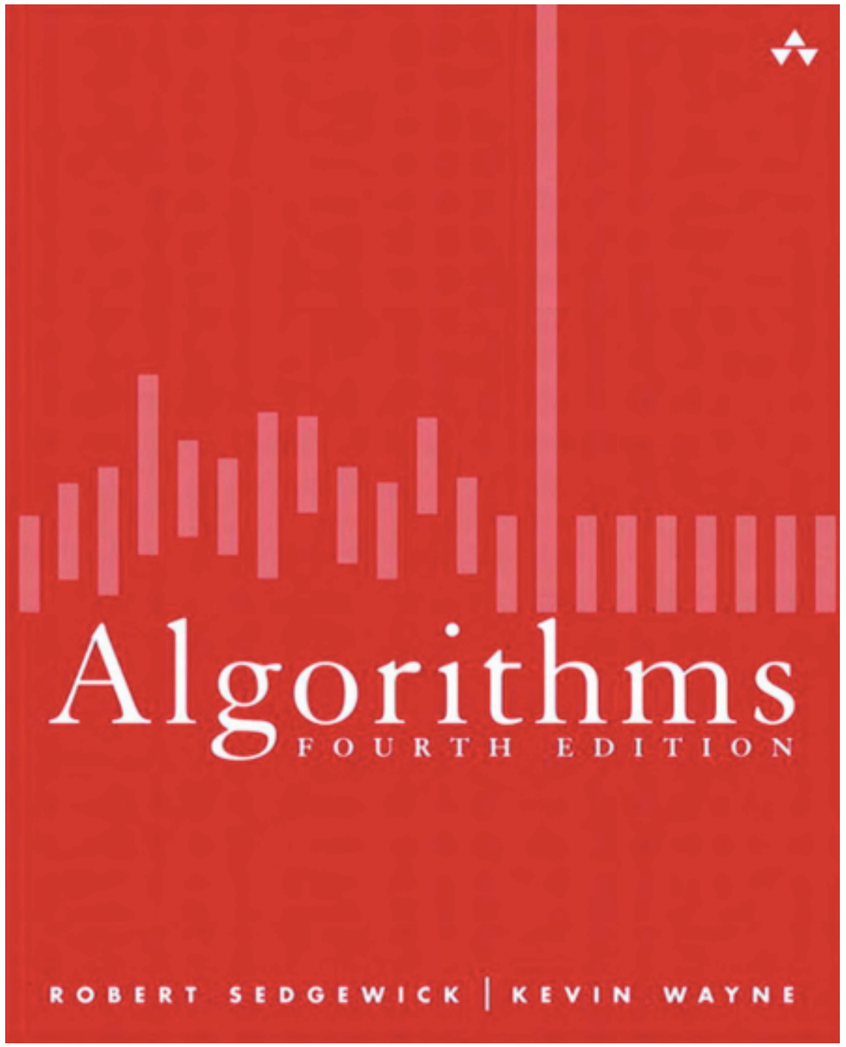
2．根节点是黑色的

3．每一个叶子节点（最后的空节点）是黑色的

4．如果一个节点是红色的，那么他的孩子节点都是黑色的

5．从任意一个节点到叶子节点，经过的黑色节点是一样的

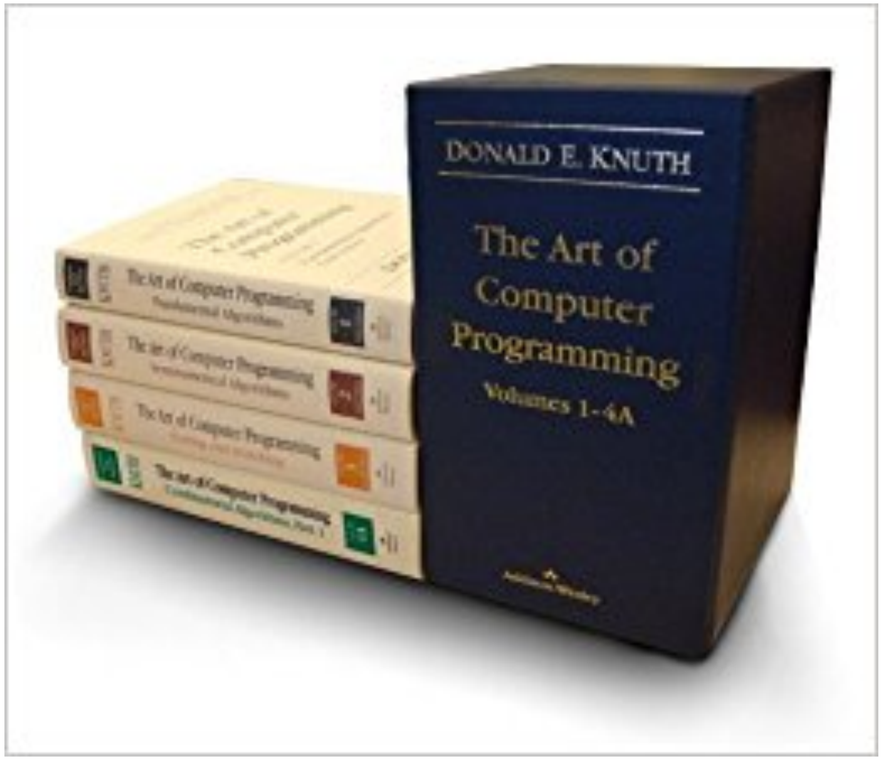
算法4



Robert Sedgewick

红黑树的发明人

算法4



Robert Sedgewick　　Donald Knuth

红黑树的发明人　　现代计算机科学的前驱

红黑树

Robert Sedgewick

红黑树的发明人

红黑树与2-3树的等价性

红黑树

红黑树与2-3树的等价性

理解了2-3树和红黑树之间的关系

红黑树并不难！

2-3 树

2-3 树

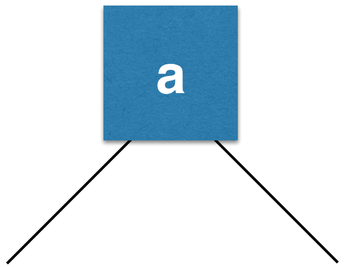
学习2-3树，不仅对于理解红黑树有帮助

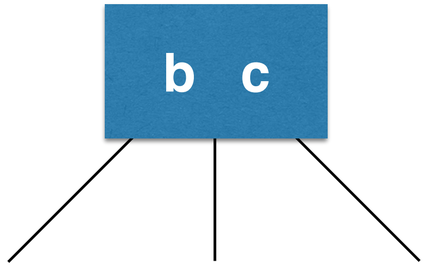
对于理解B类树，也是有巨大帮助的！

2-3树

满足二分搜索树的基本性质

节点可以存放一个元素或者两个元素



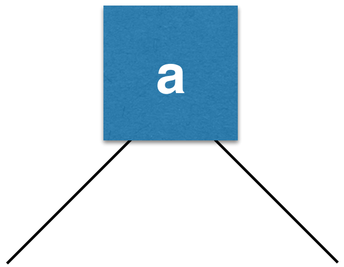


每个节点有2个或者3个孩子 —— 2-3树

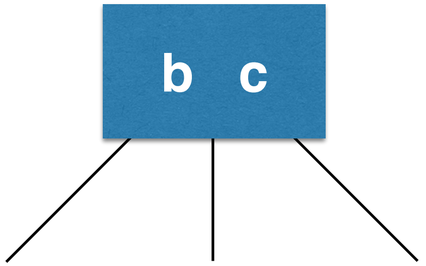
2-3树

满足二分搜索树的基本性质

节点可以存放一个元素或者两个元素

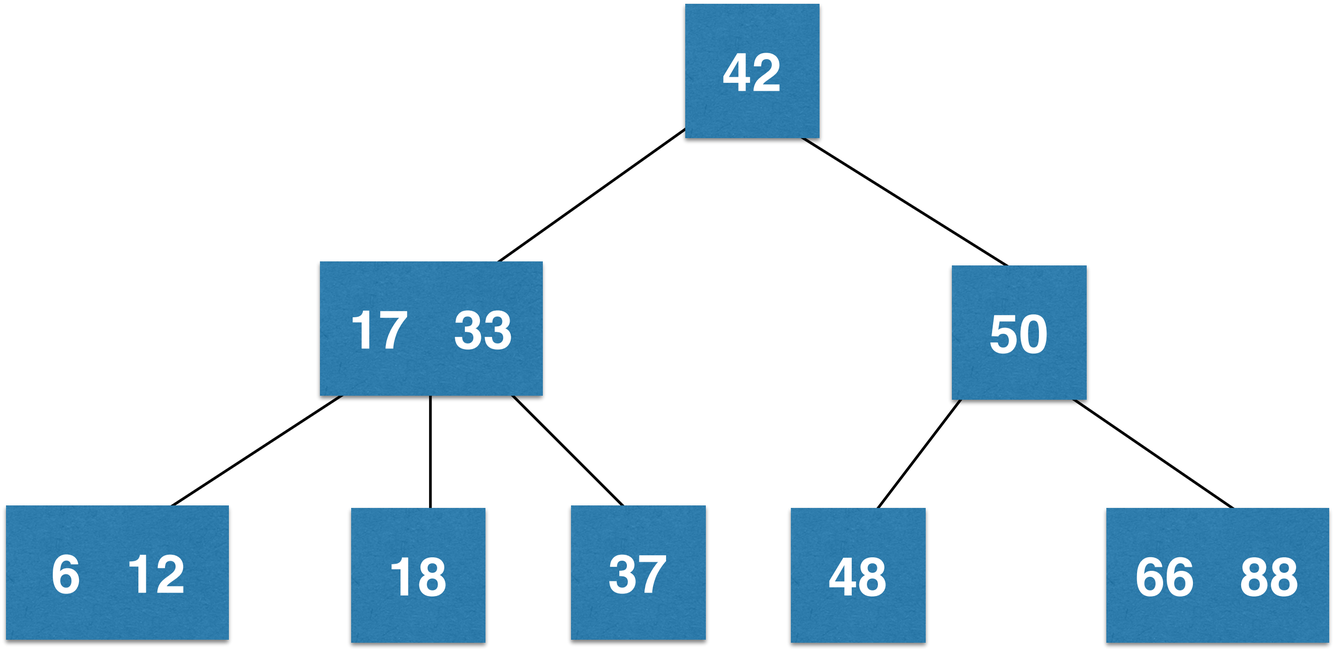


2节点



3节点

2-3树



2-3树是一棵绝对平衡的树

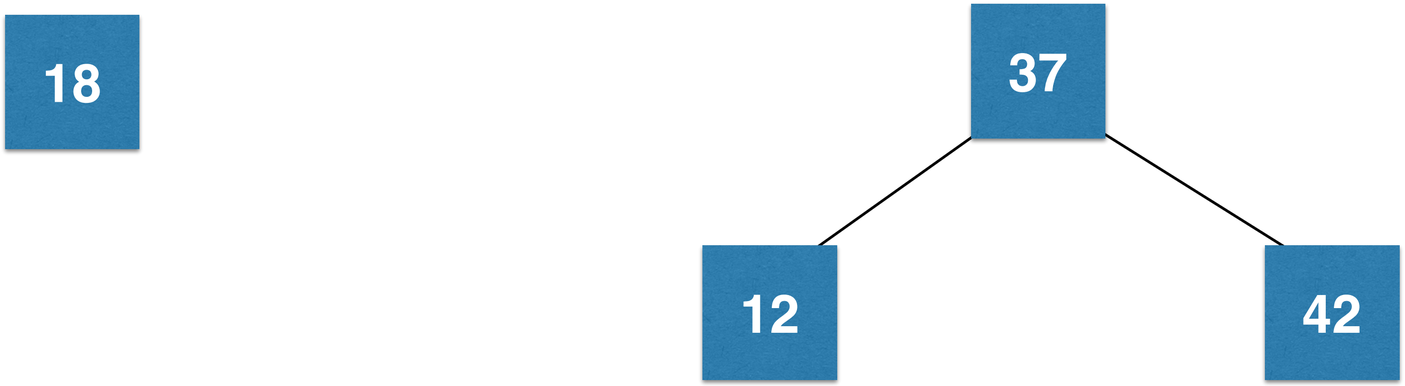
2-3 树如何维持绝对的平衡

2-3树

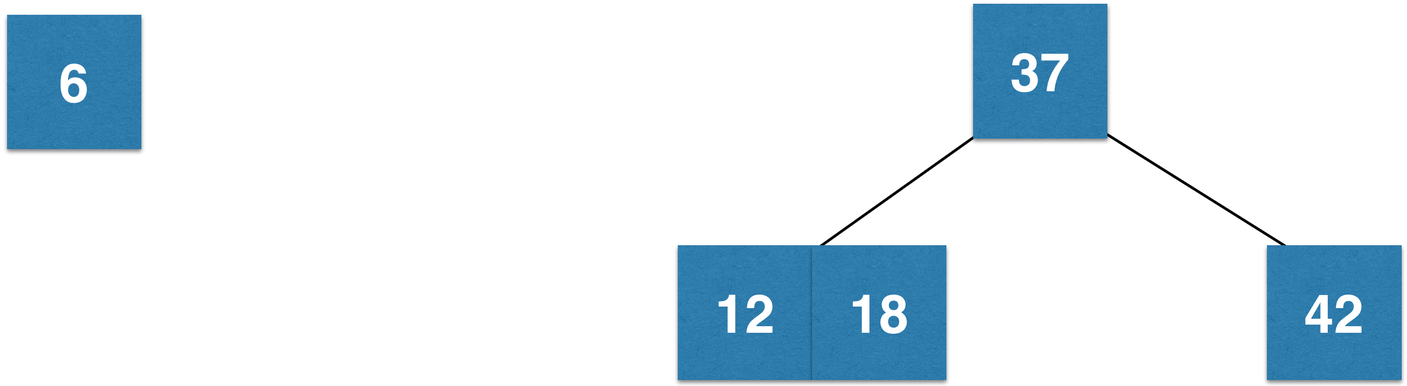
2-3树

2-3树

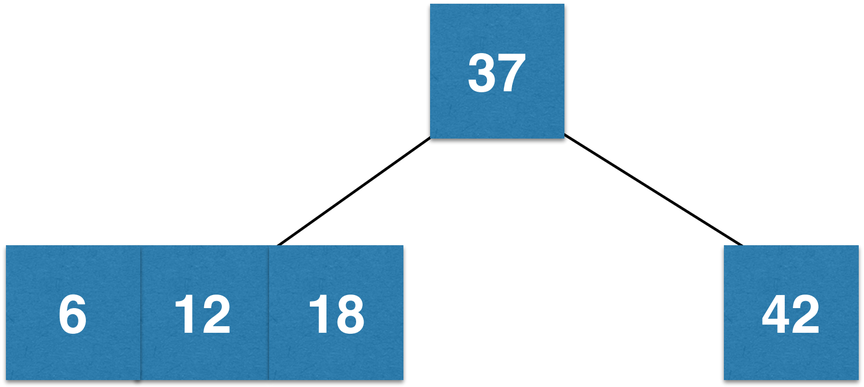
2-3树



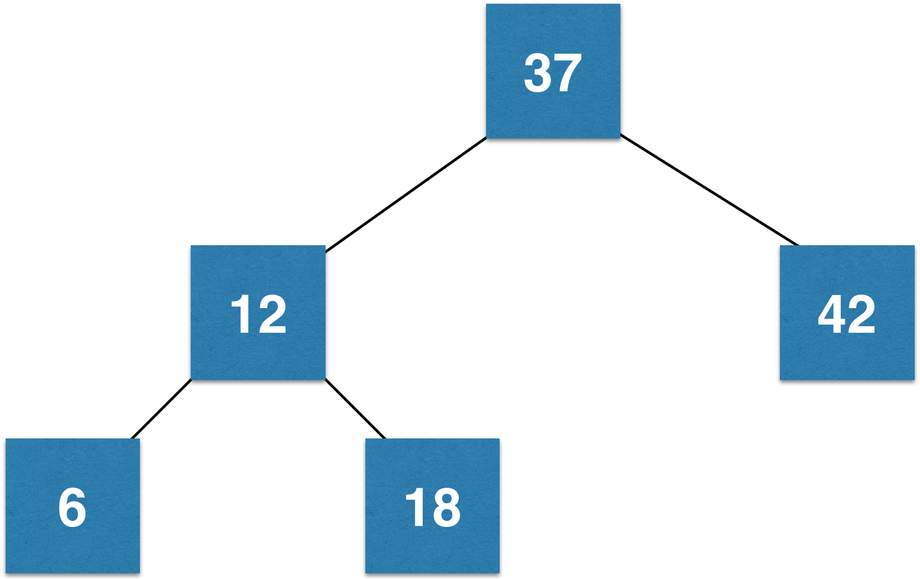
2-3树



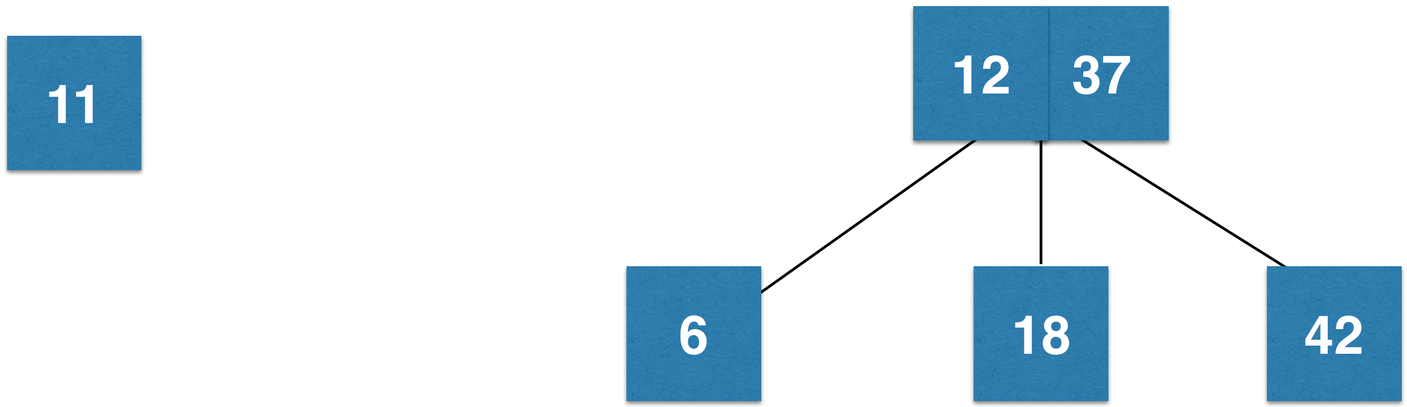
2-3树



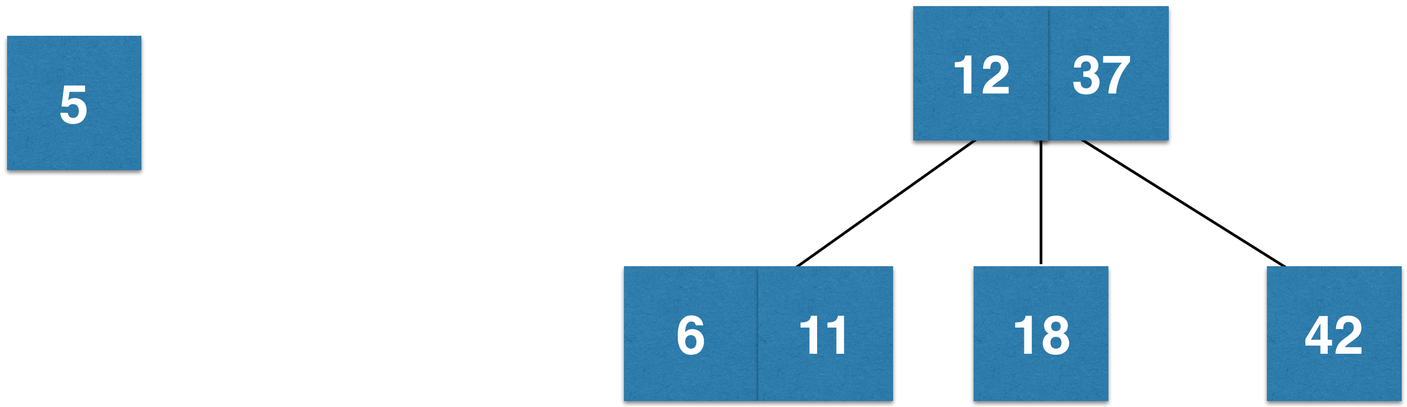
2-3树



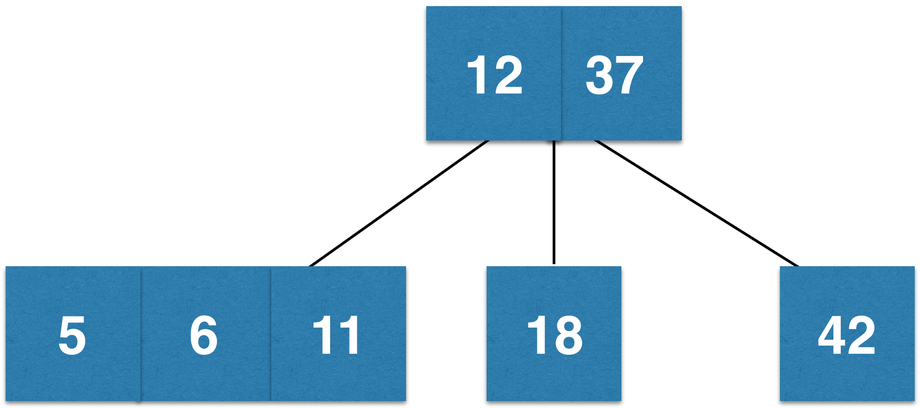
2-3树



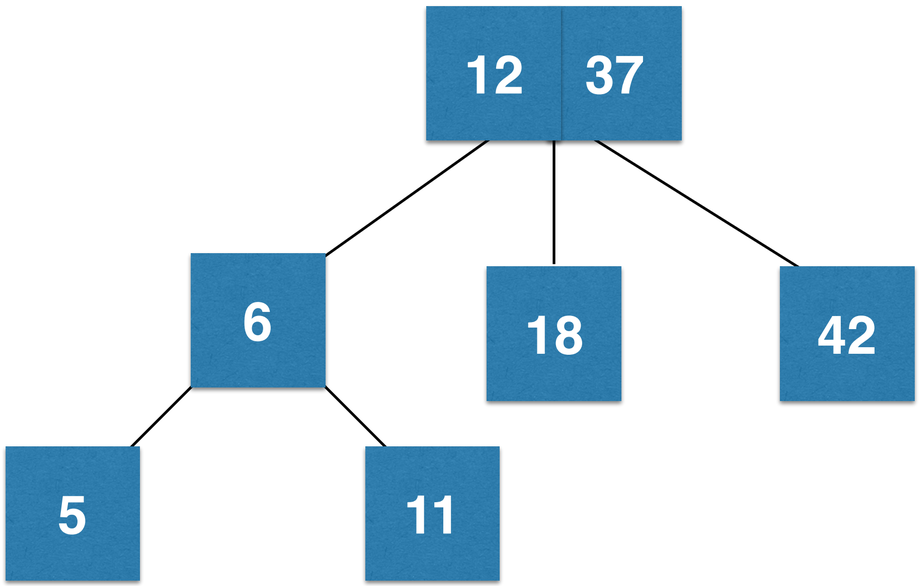
2-3树



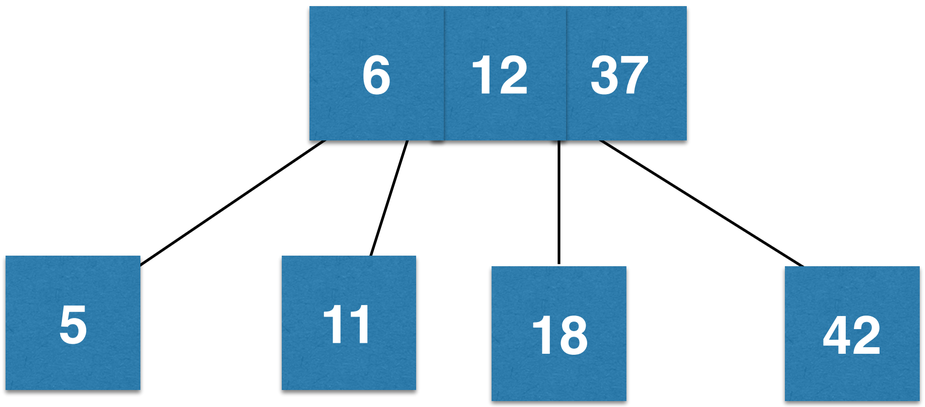
2-3树



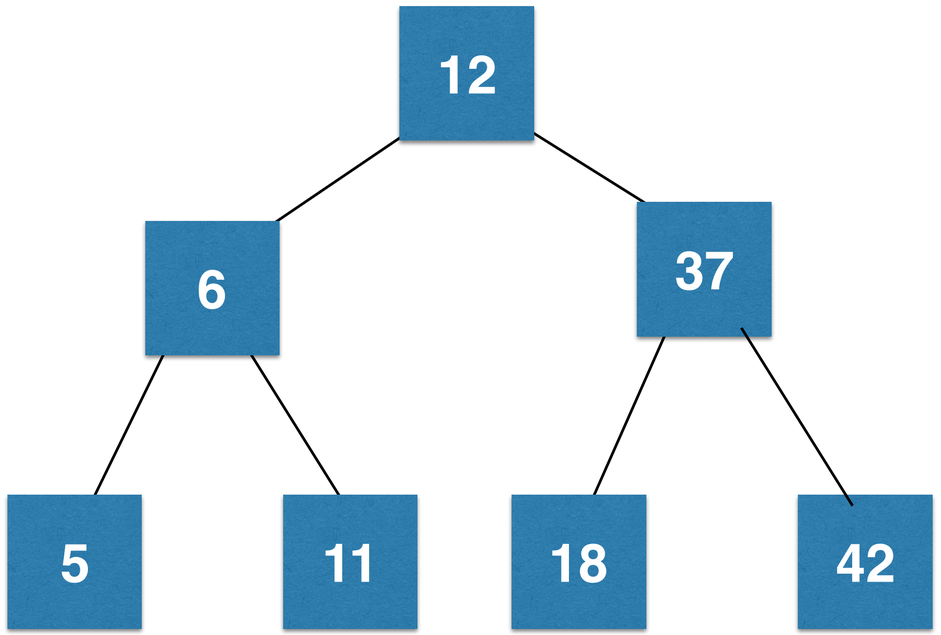
2-3树



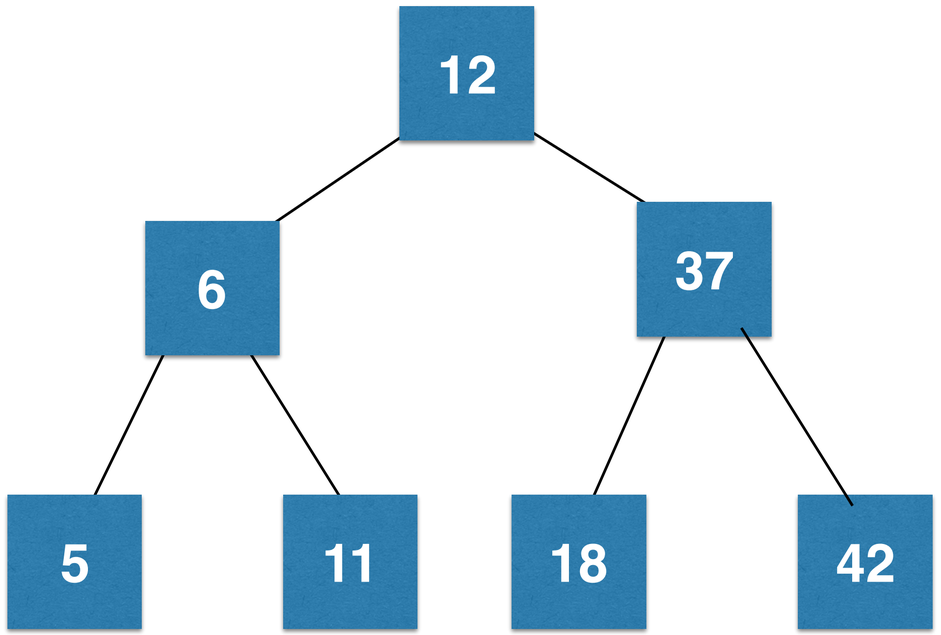
2-3树



2-3树



2-3树

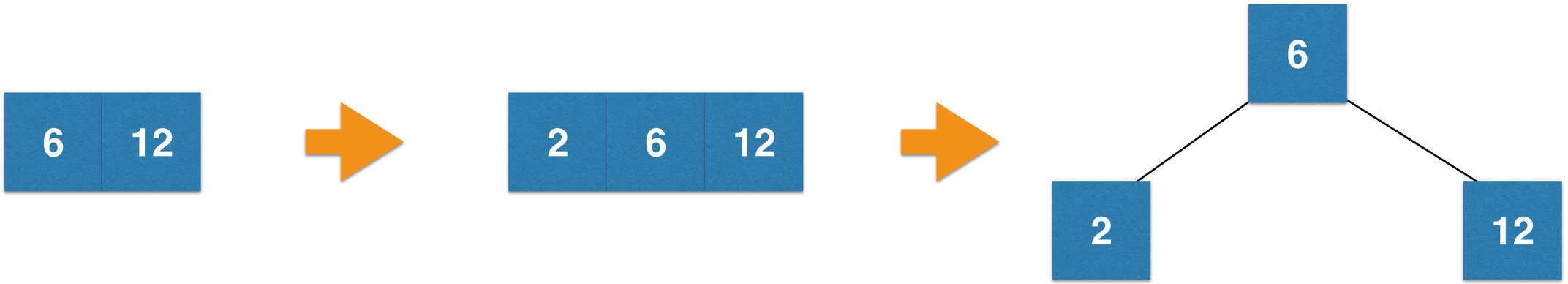


2-3树

如果插入2-节点

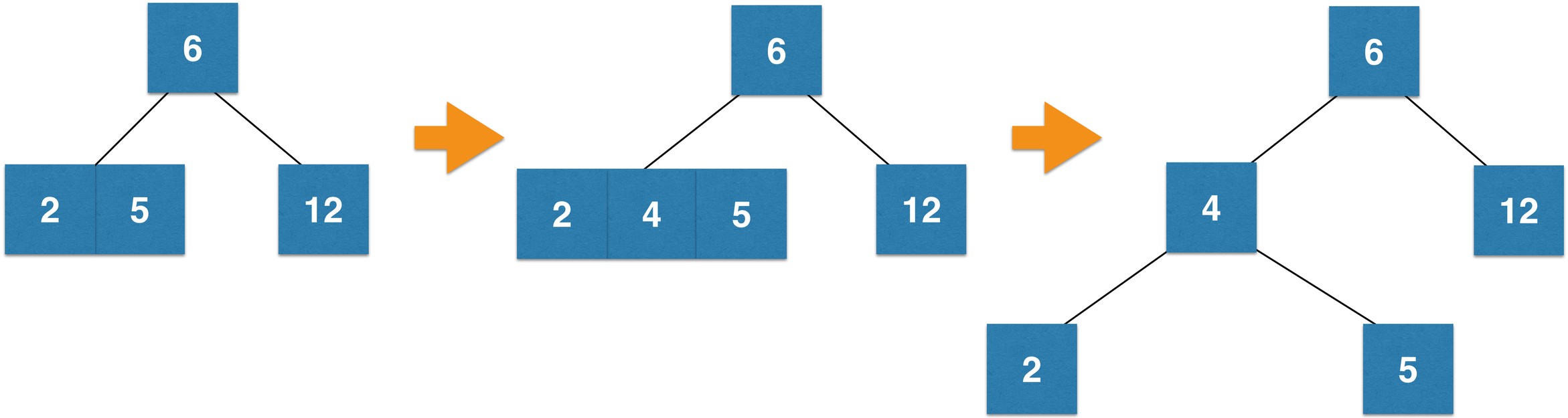
2-3树

如果插入3-节点

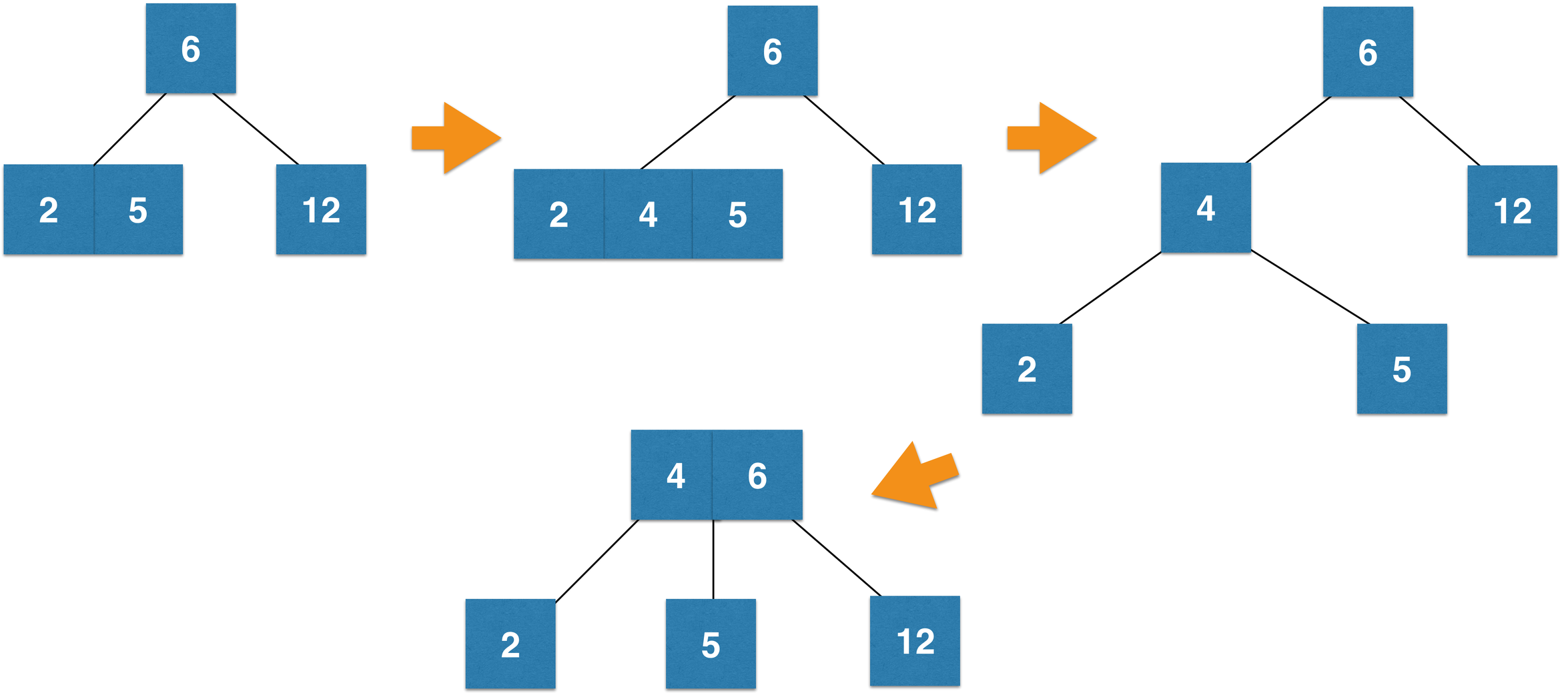


2-3树

如果插入3-节点　　父亲节点为2-节点

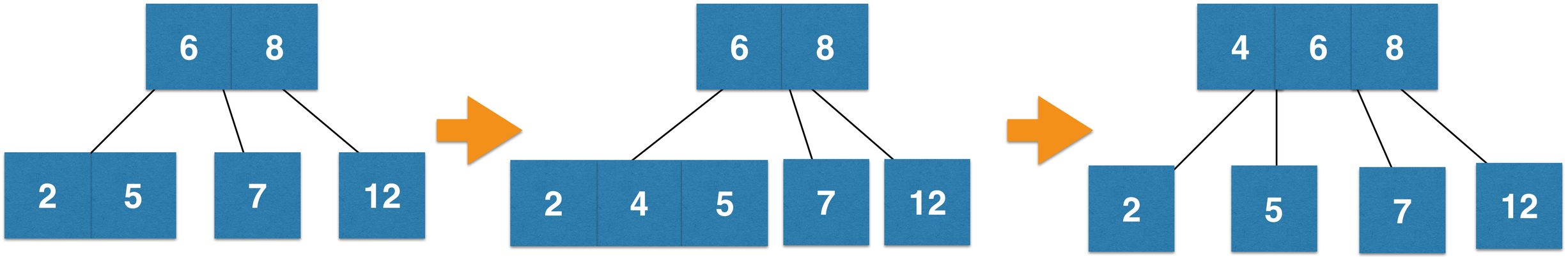


2-3树

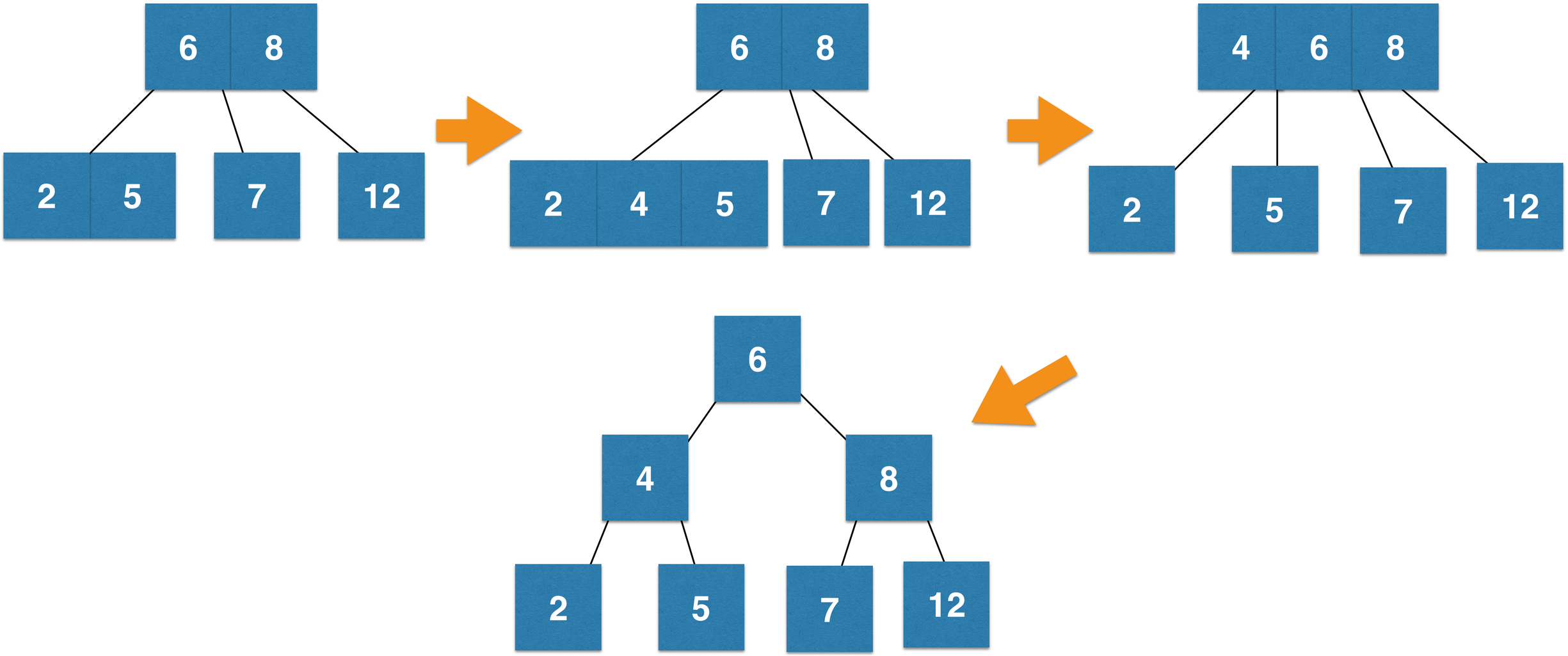


2-3树

如果插入3-节点　　父亲节点为3-节点



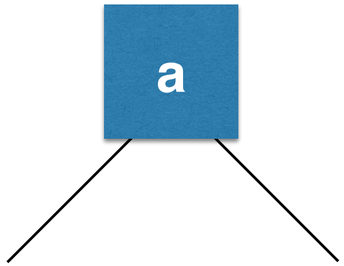
2-3树

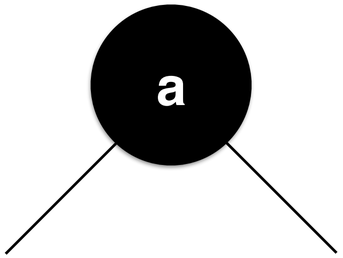


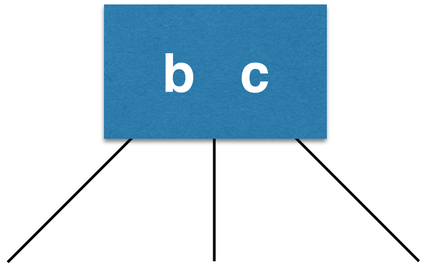
红黑树和 2-3 树的等价性

红黑树和2-3树

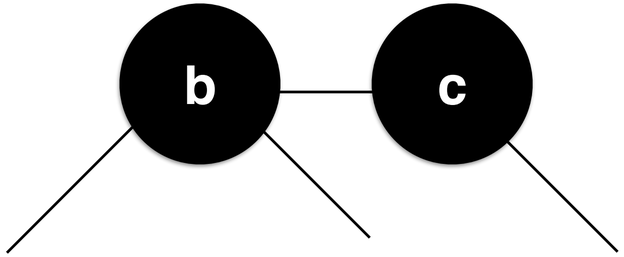
2节点





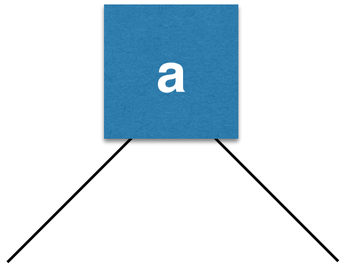


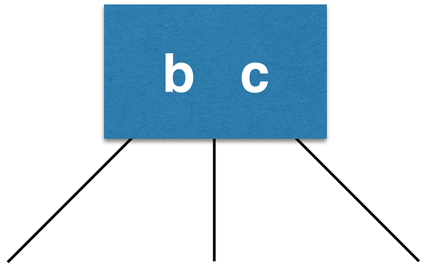
3节点



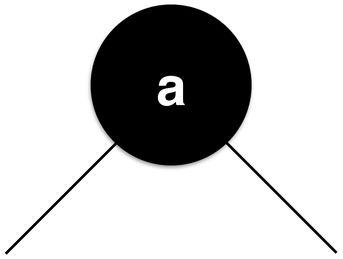
红黑树和2-3树

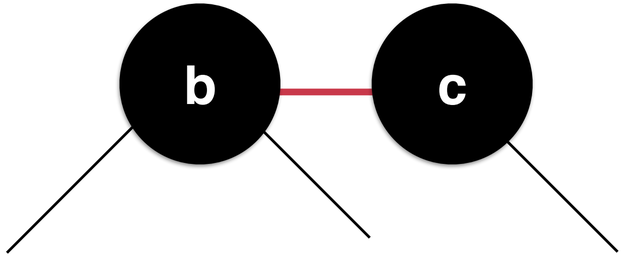
2节点





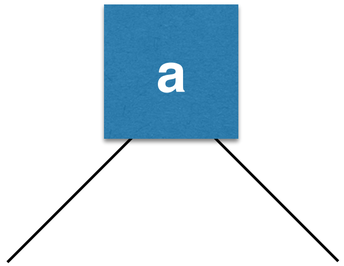
3节点

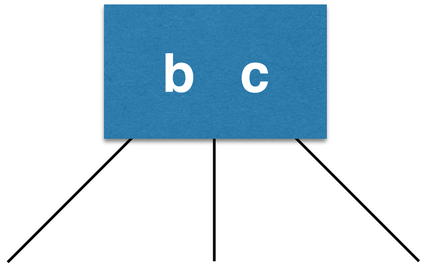




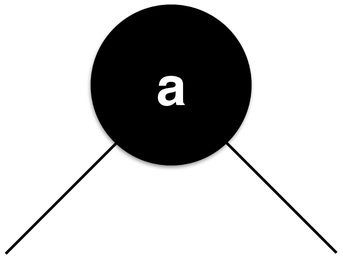
红黑树和2-3树

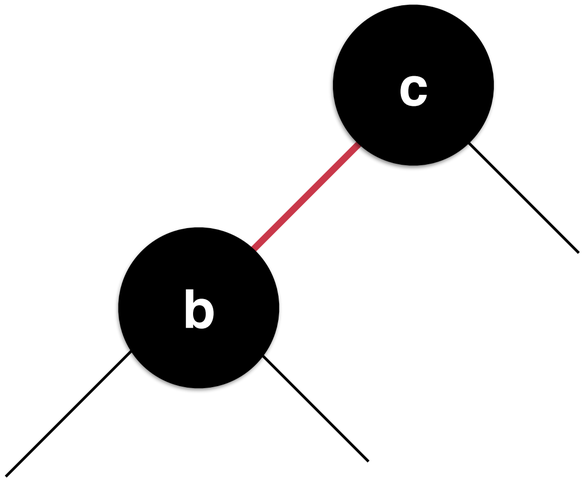
2节点





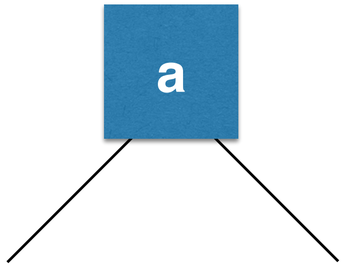
3节点

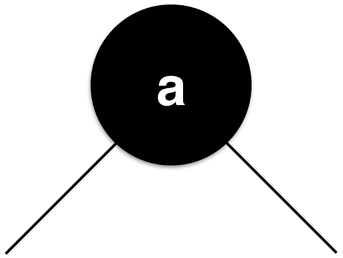


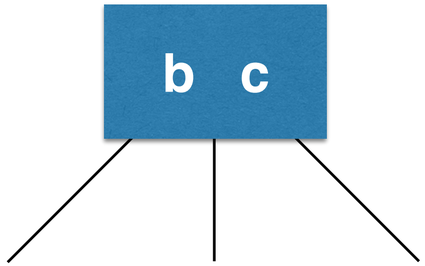


红黑树和2-3树

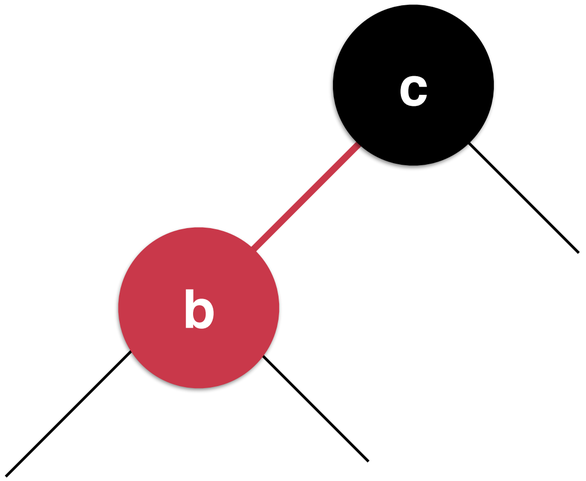
2节点





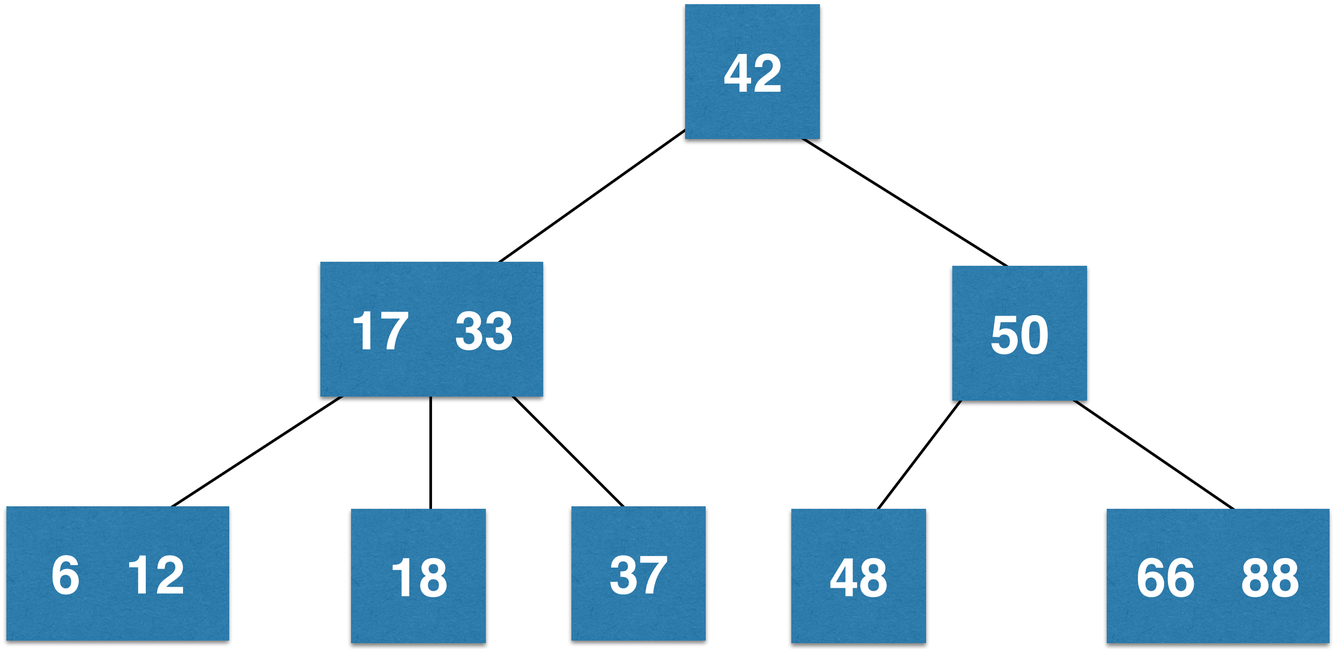


3节点

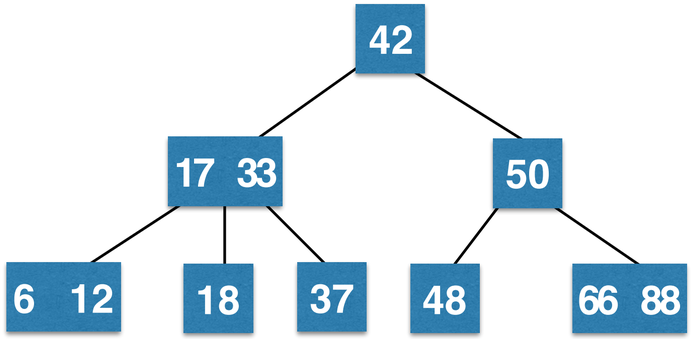


所有的红色节点都是左倾斜的

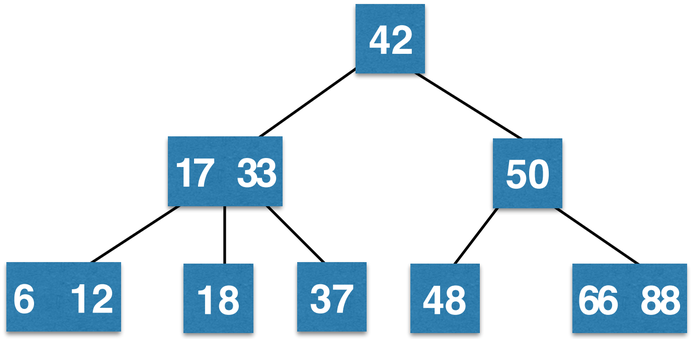
红黑树和2-3树

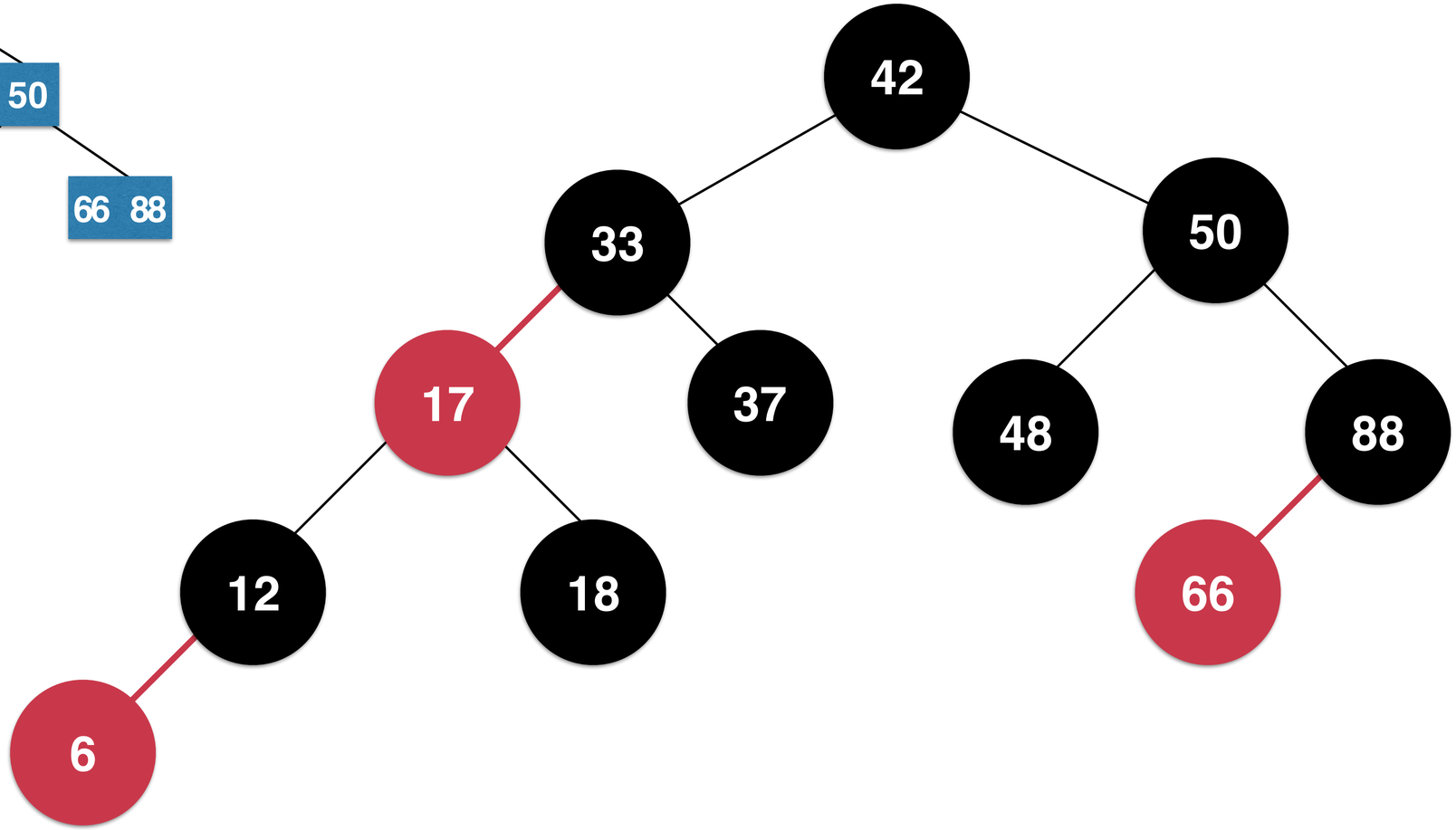


红黑树和2-3树

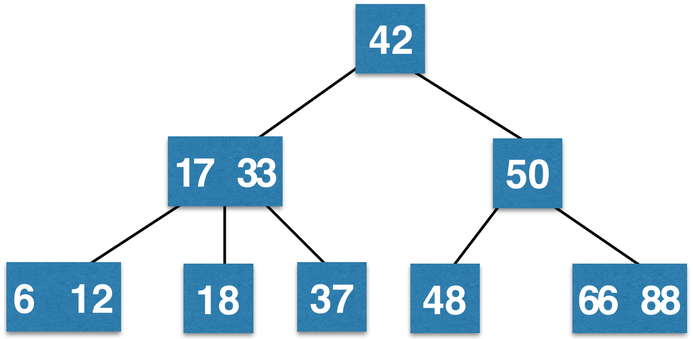


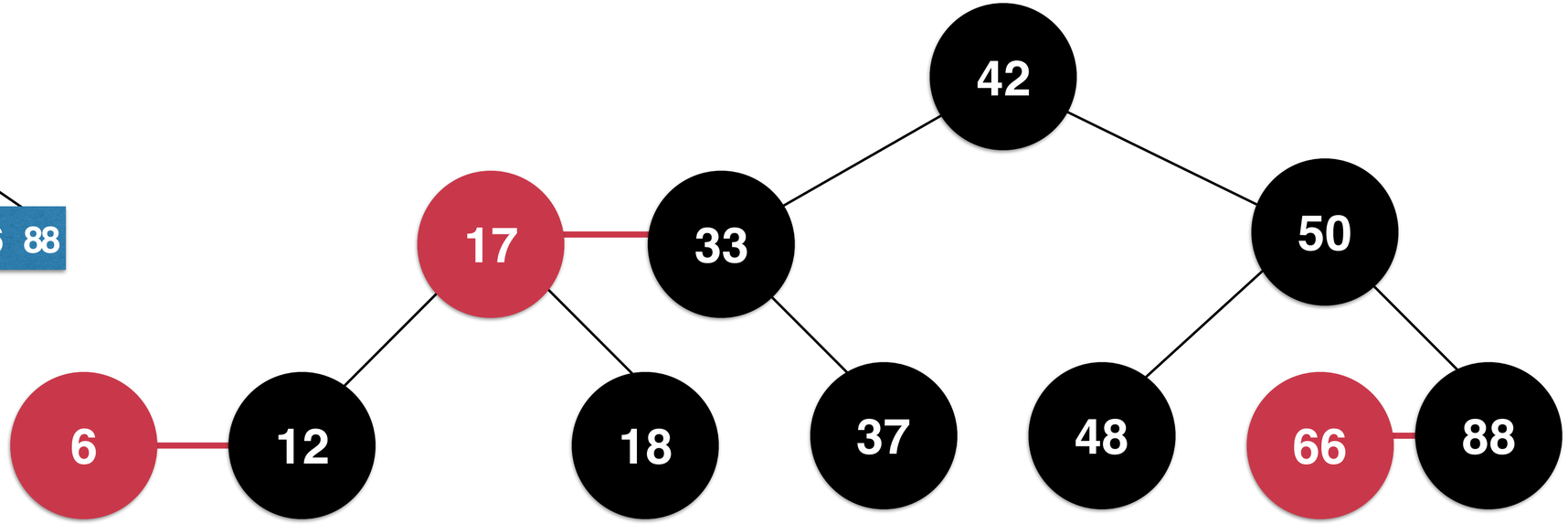
红黑树和2-3树





红黑树和2-3树



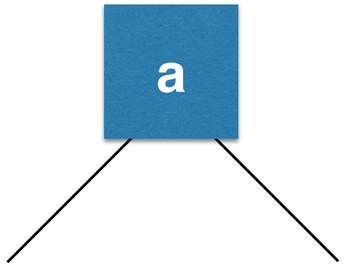


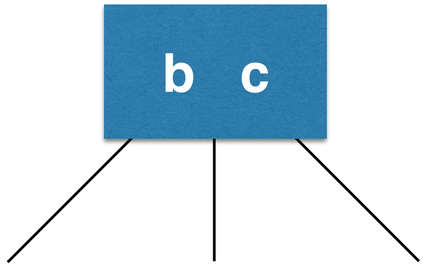
实践：红黑树的节点

红黑树的重要性质

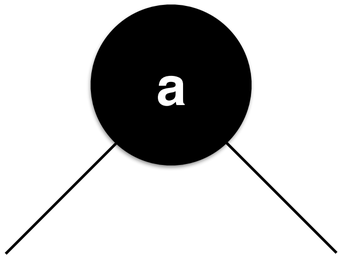
红黑树和2-3树的等价性

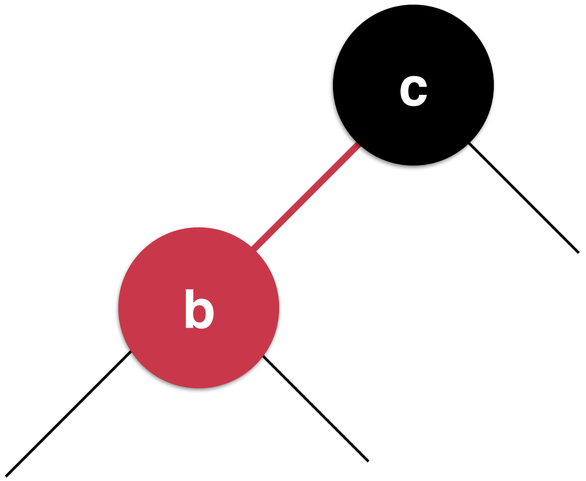
2节点





3节点





《算法导论》中的红黑树

A red-black tree is a binary tree that satisfies the following ***red-black properties***:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

《算法导论》中的红黑树

1．每个节点或者是红色的，或者是黑色的

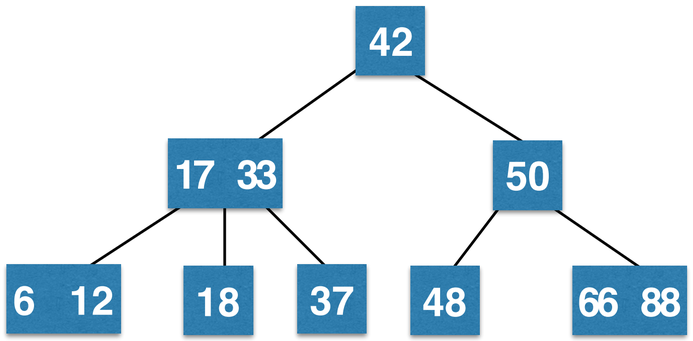
2．根节点是黑色的

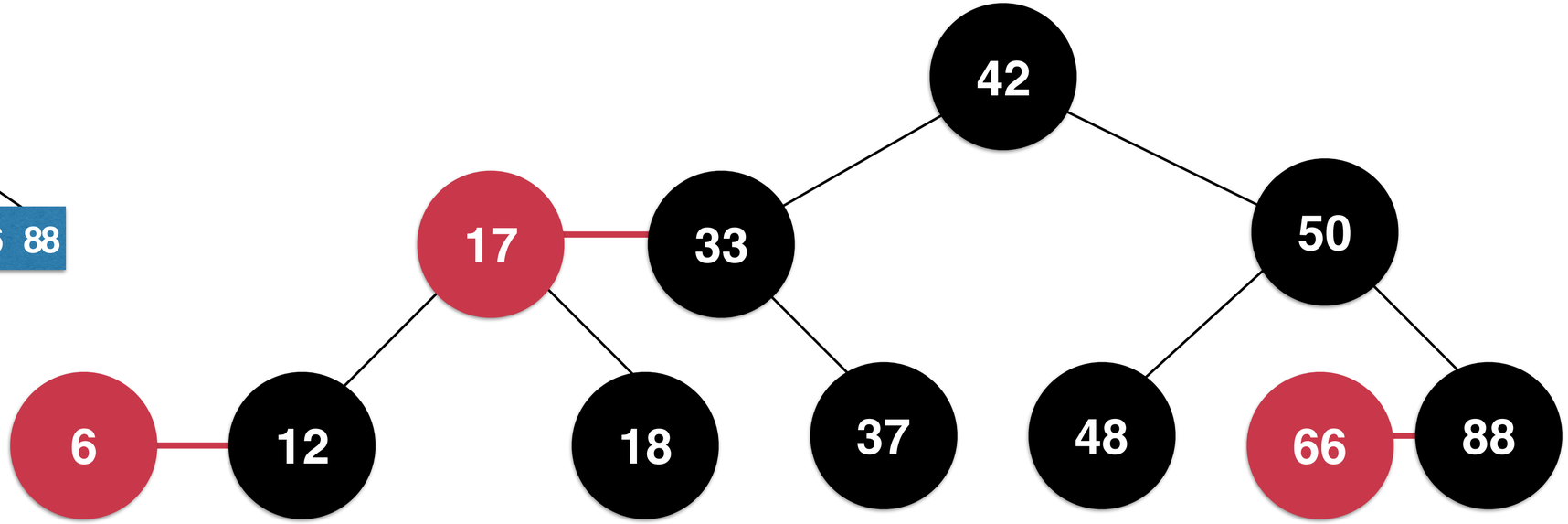
3．每一个叶子节点（最后的空节点）是黑色的

4．如果一个节点是红色的，那么他的孩子节点都是黑色的

5．从任意一个节点到叶子节点，经过的黑色节点是一样的

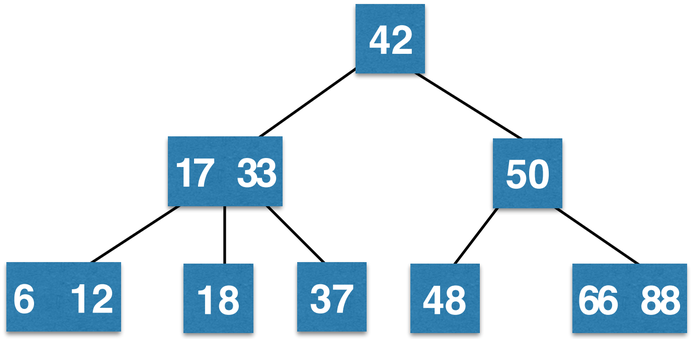
红黑树和2-3树

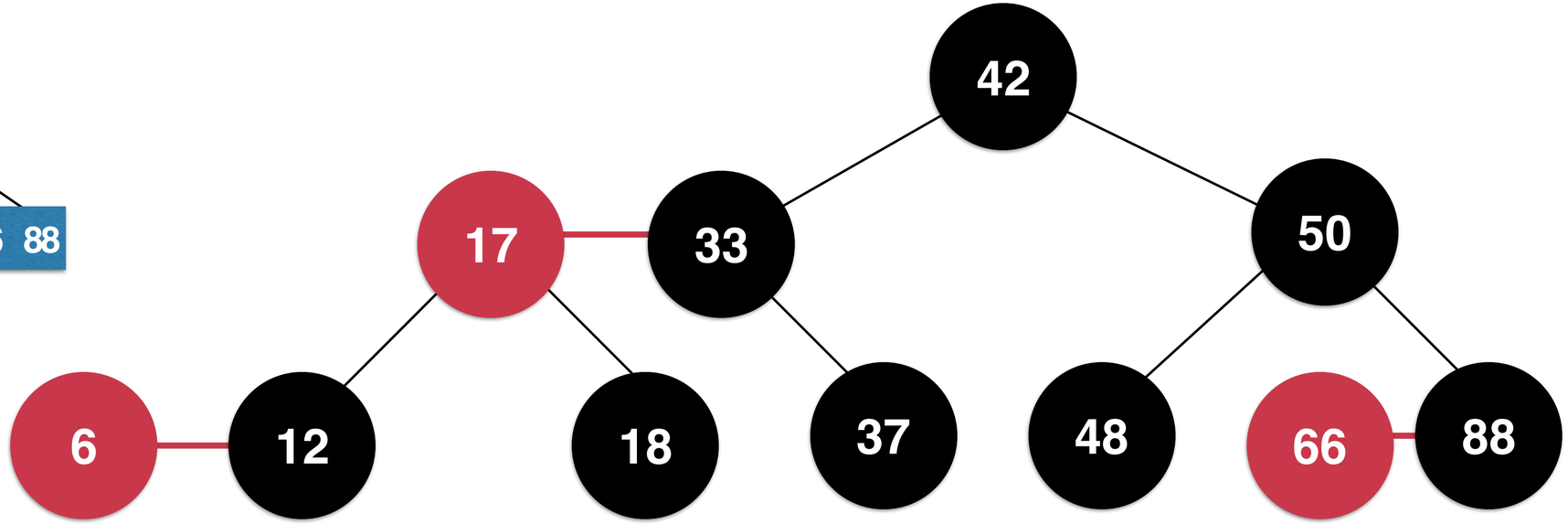




1．每个节点或者是红色的，或者是黑色的

红黑树和2-3树

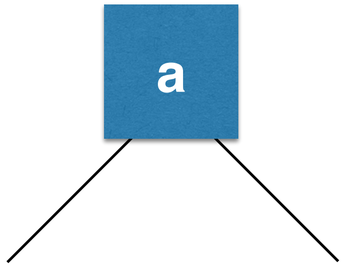


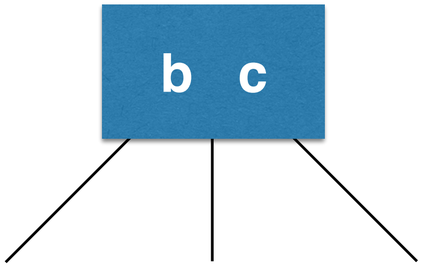


2．根节点是黑色的

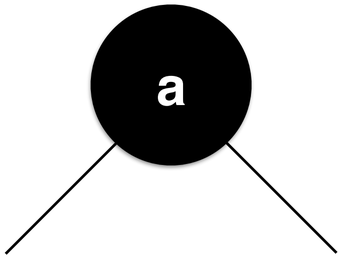
红黑树和2-3树

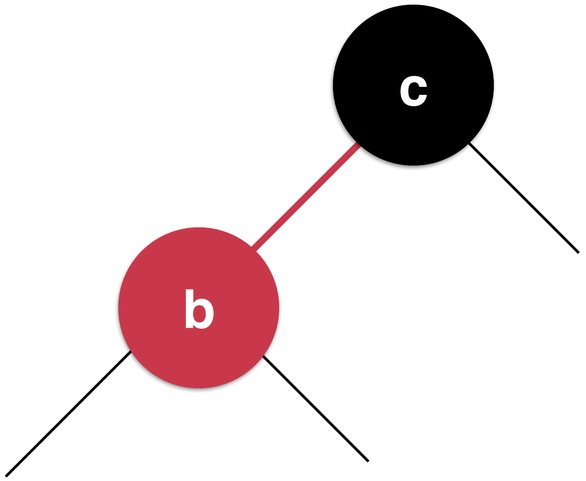
2节点





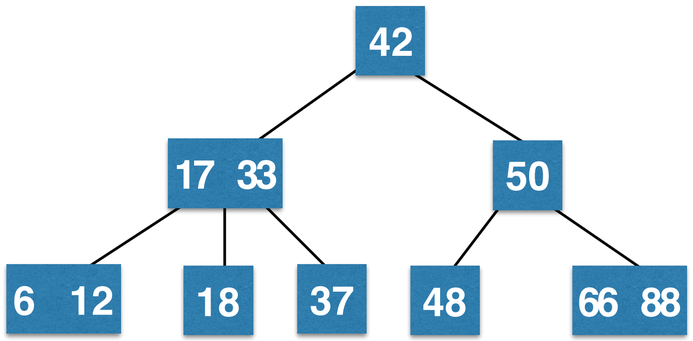
3节点

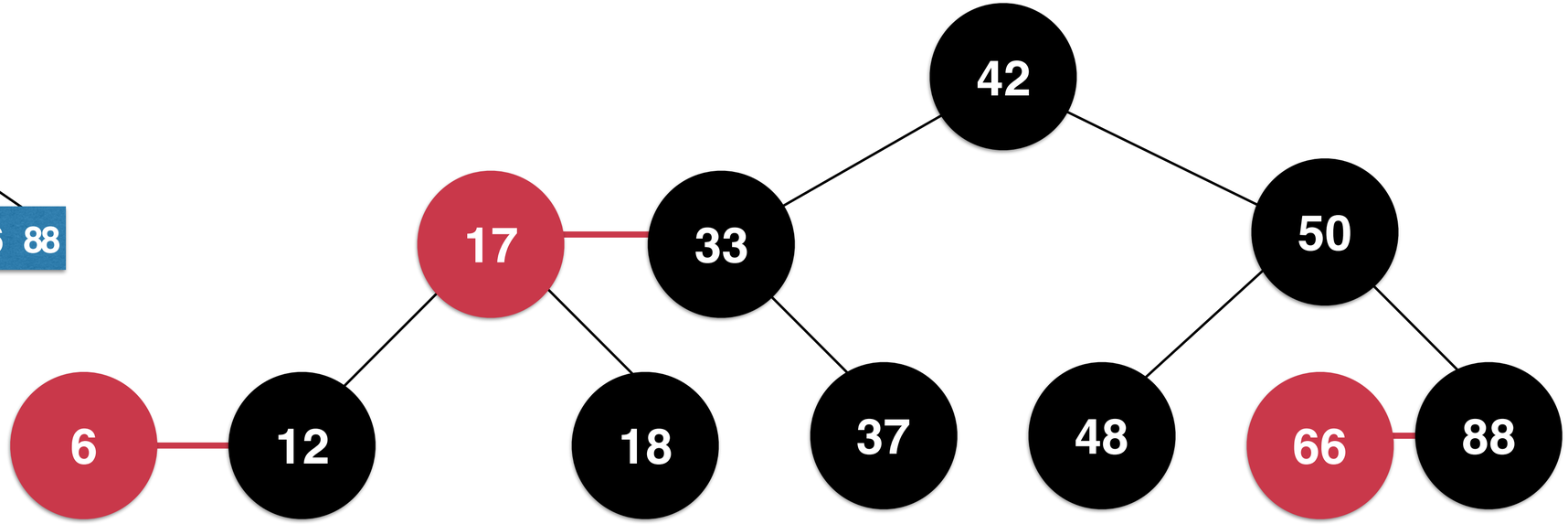




2．根节点是黑色的

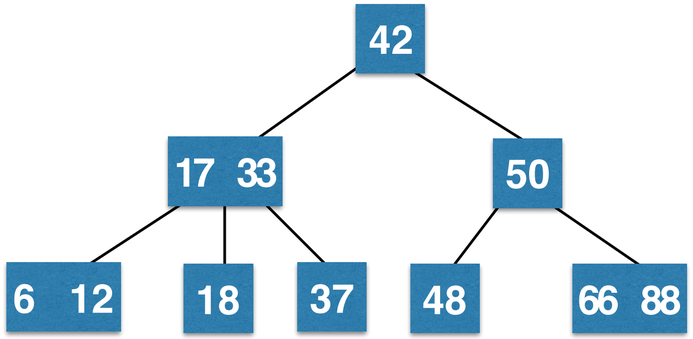
红黑树和2-3树

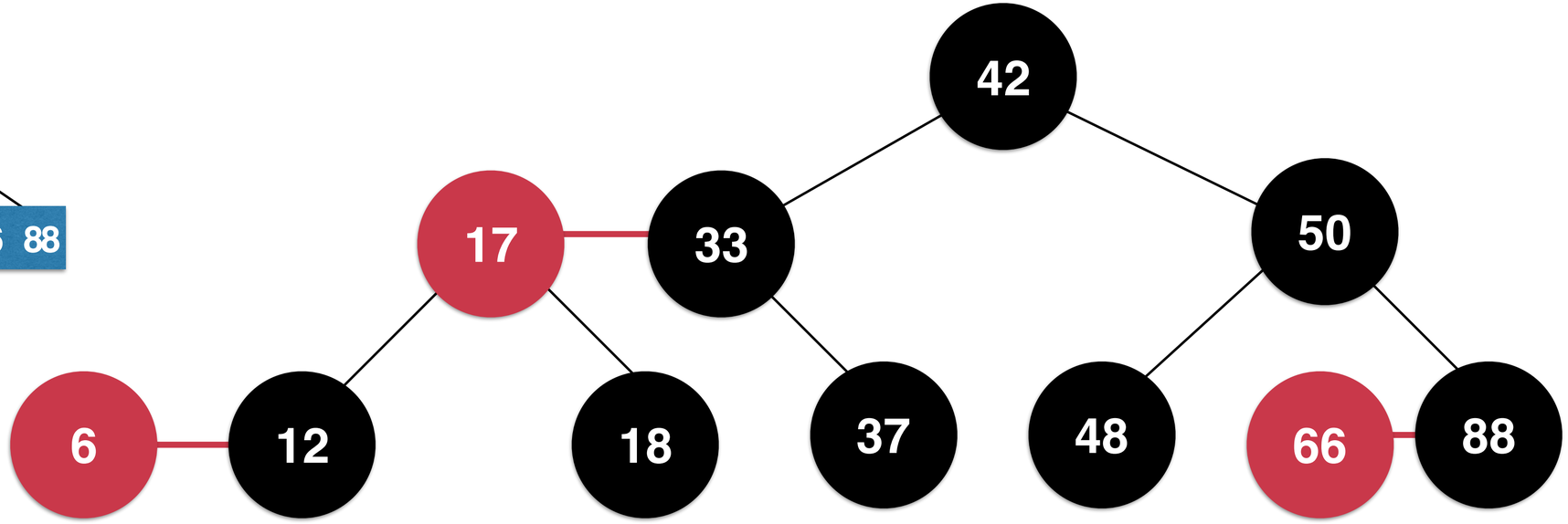




3．每一个叶子节点（最后的空节点）是黑色的

红黑树和2-3树

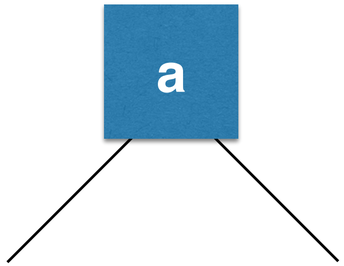


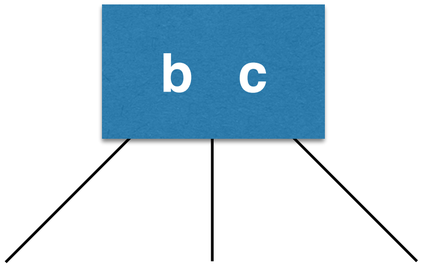


4．如果一个节点是红色的，那么他的孩子节点都是黑色的

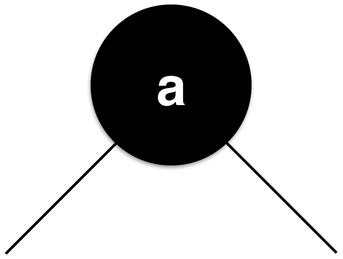
红黑树和2-3树

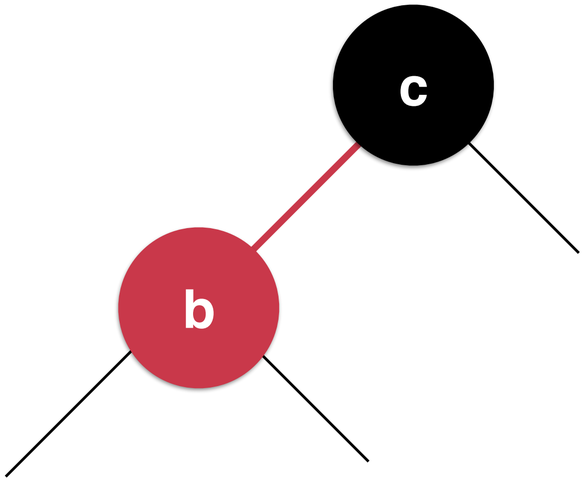
2节点





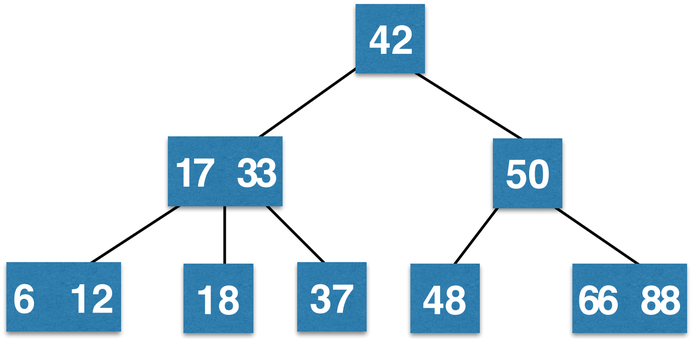
3节点

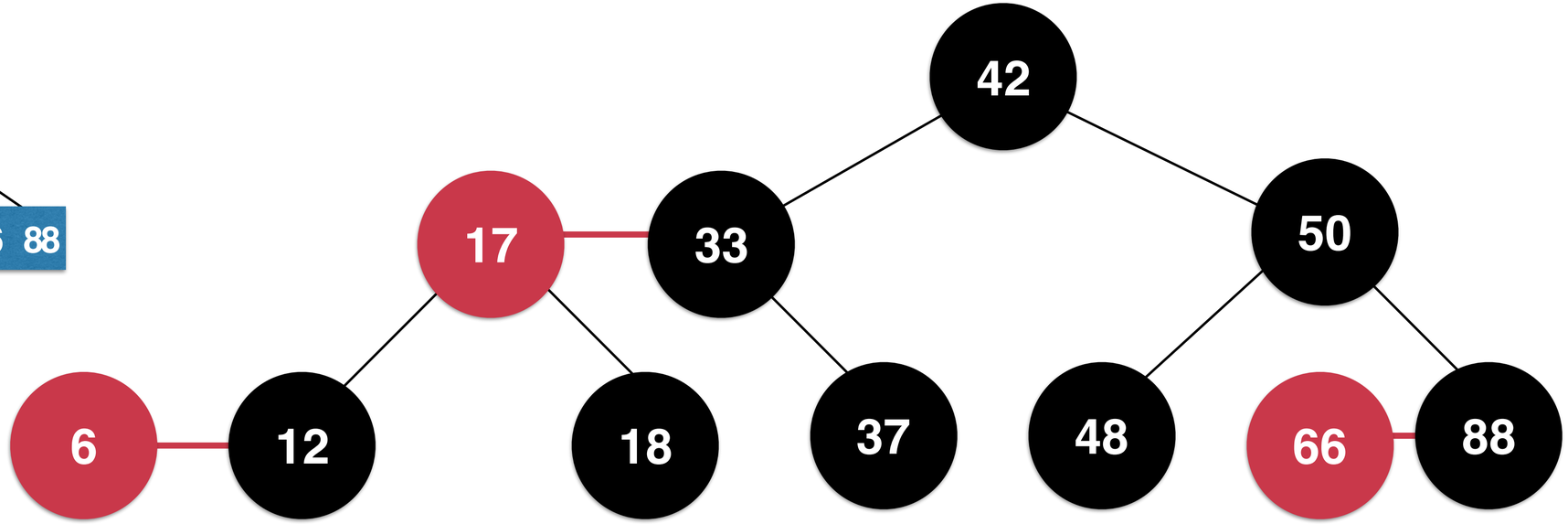




4．如果一个节点是红色的，那么他的孩子节点都是黑色的

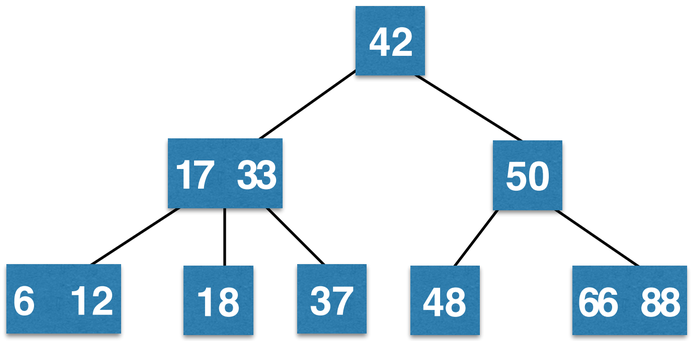
红黑树和2-3树

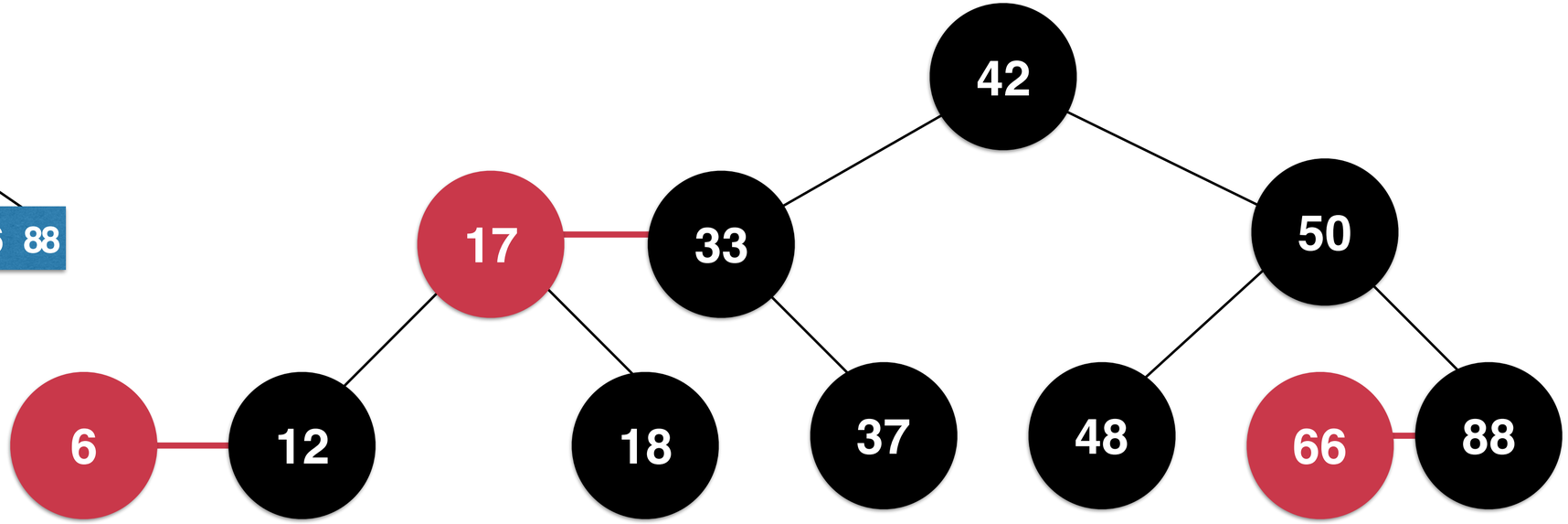




5．从任意一个节点到叶子节点，经过的黑色节点是一样的

红黑树和2-3树





红黑树是保持“黑平衡”的二叉树

严格意义上，不是平衡二叉树　　最大高度：2logn　　O(logn)

《算法导论》中的红黑树

A red-black tree is a binary tree that satisfies the following ***red-black properties***:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

红黑树添加新元素

保持根节点黑色和左旋转

红黑树添加新元素

2-3树中添加一个新元素

或者添加进2-节点，形成一个3-节点

或者添加进3-节点，暂时形成一个4-节点

永远添加红色节点

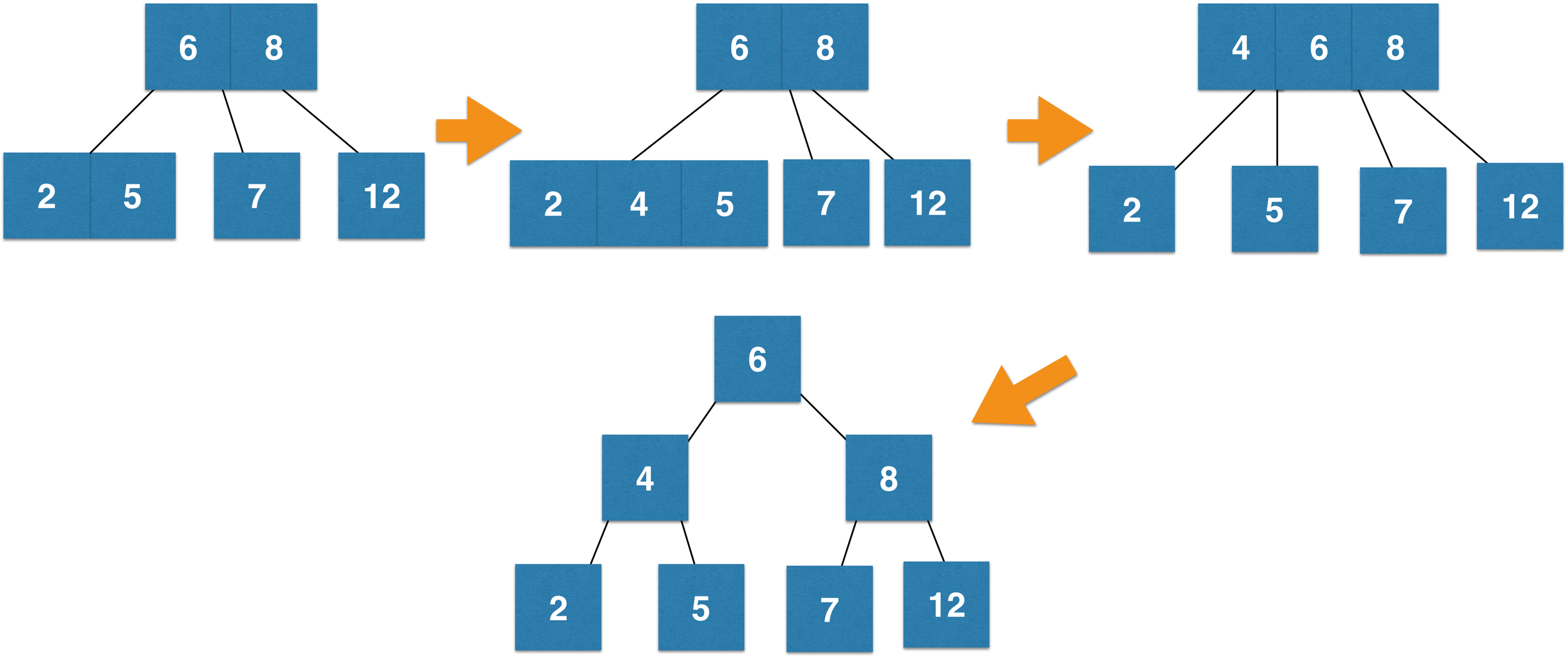
红黑树添加新元素

42

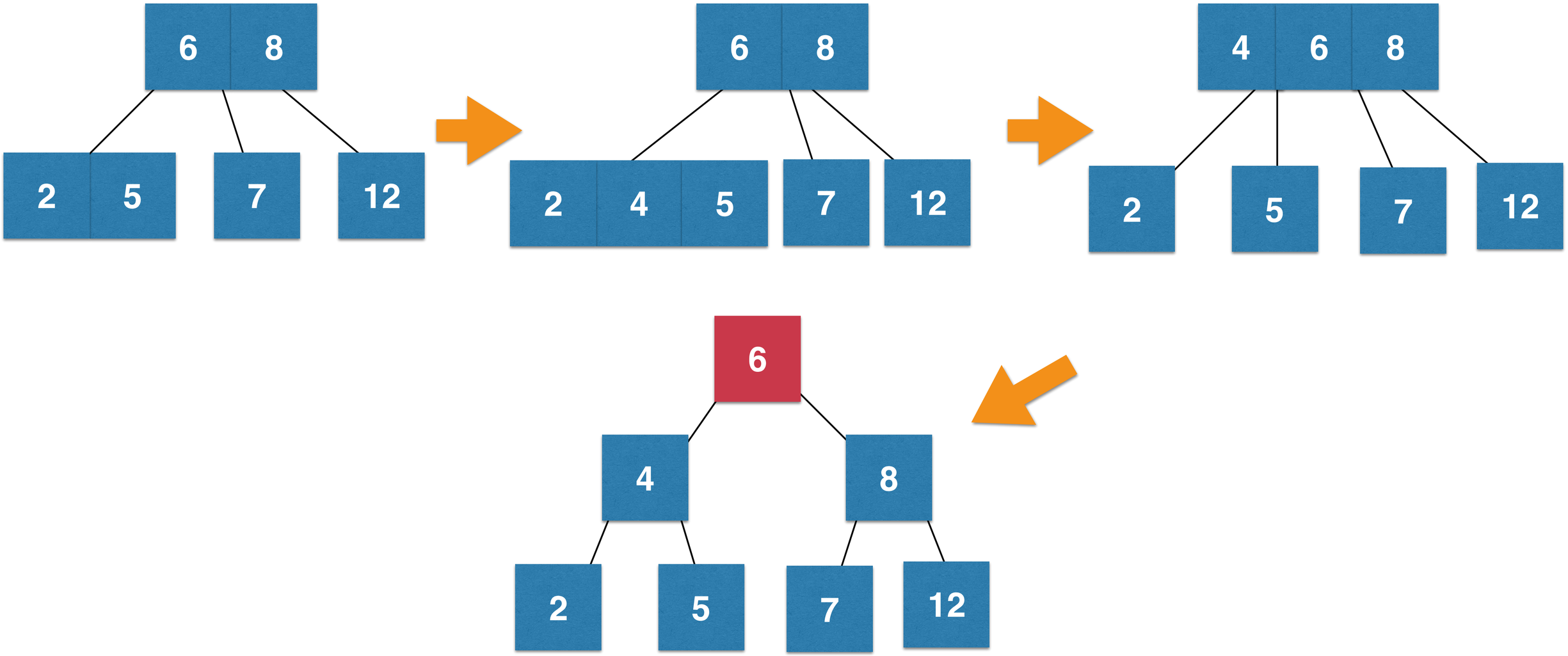
红黑树添加新元素

红黑树添加新元素

2-3树



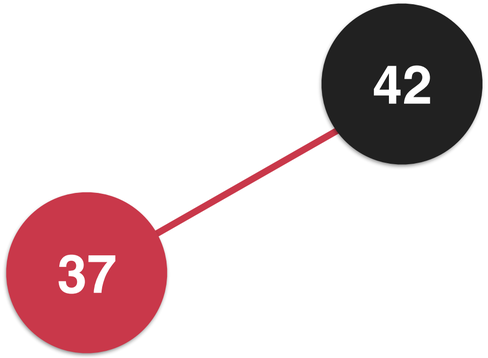
2-3树



实践：保持根节点为黑色节点

红黑树添加新元素

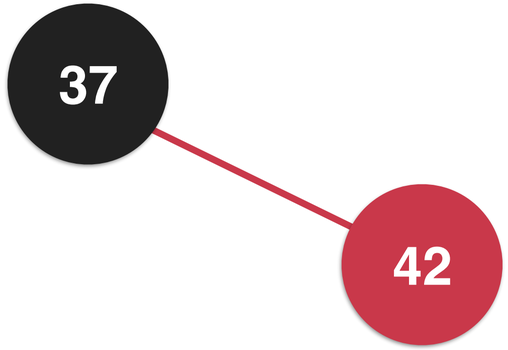
红黑树添加新元素



红黑树添加新元素

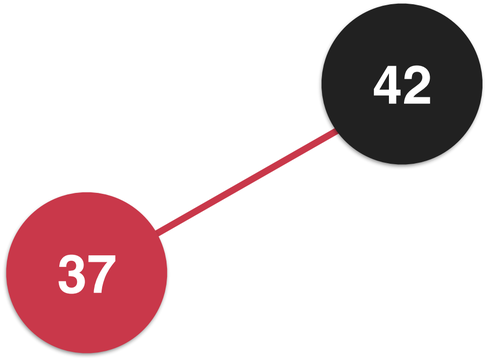
红黑树添加新元素

此时需要进行左旋转



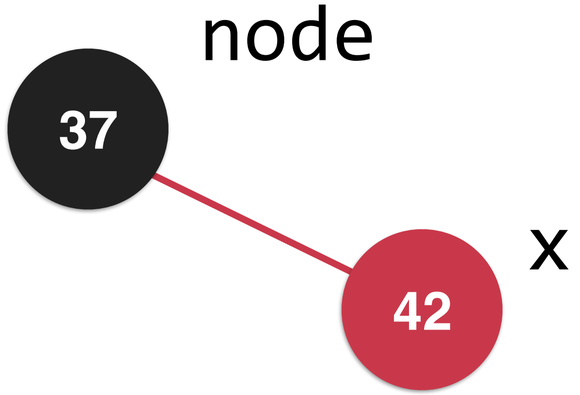
红黑树添加新元素

此时需要进行左旋转

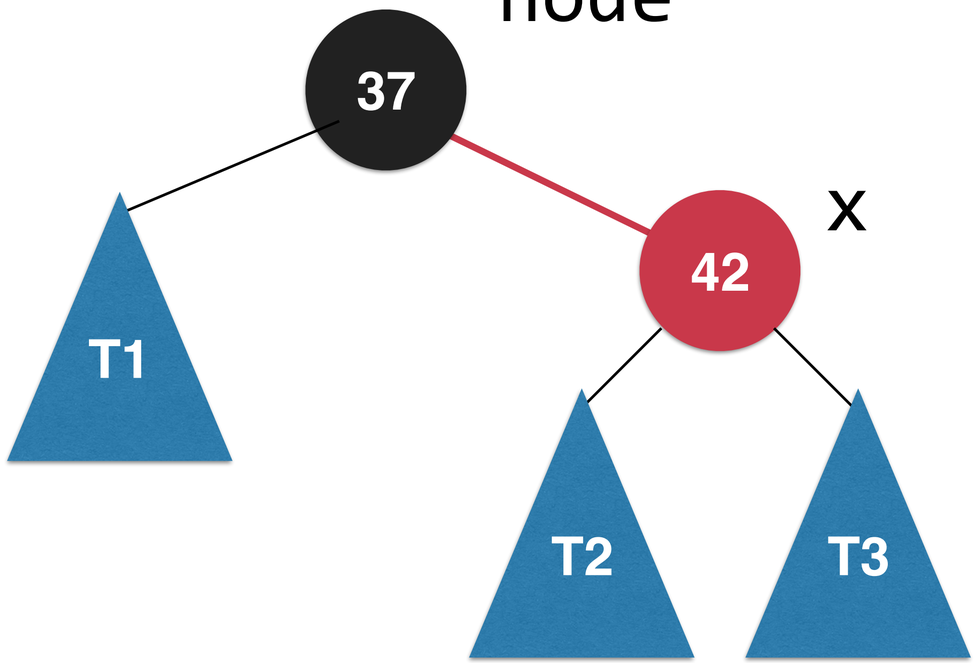


左旋转

此时需要进行左旋转



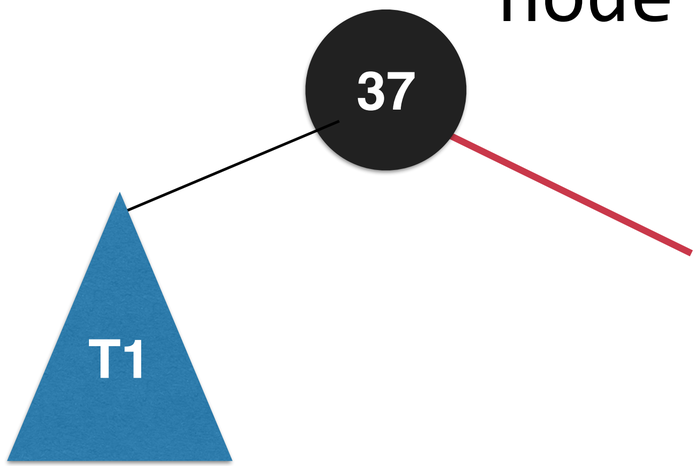
左旋转




```
node.right = x.left
```

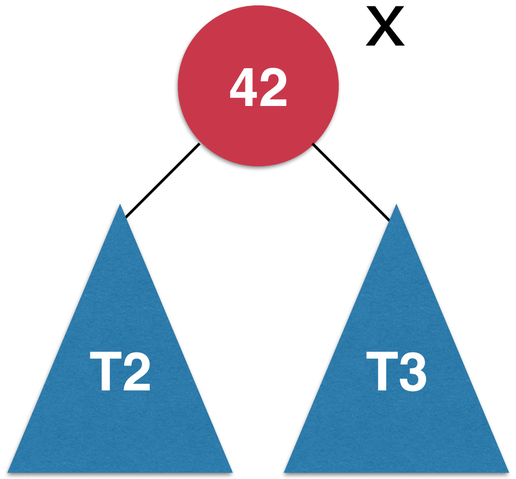
左旋转

node

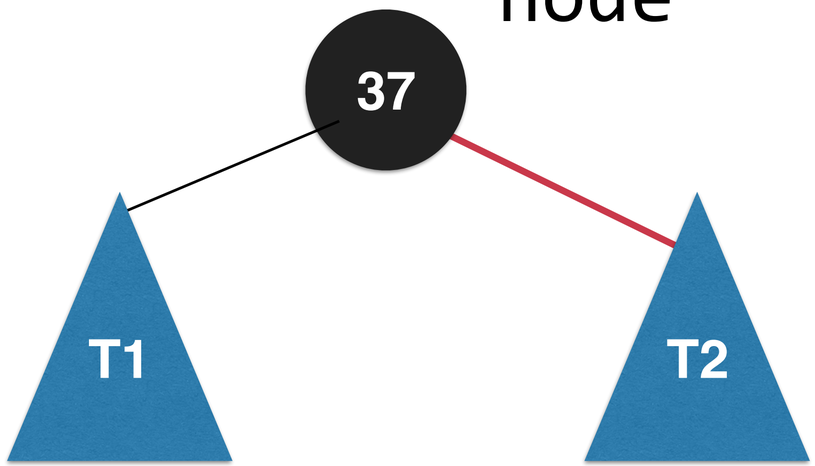


node.right = x.left

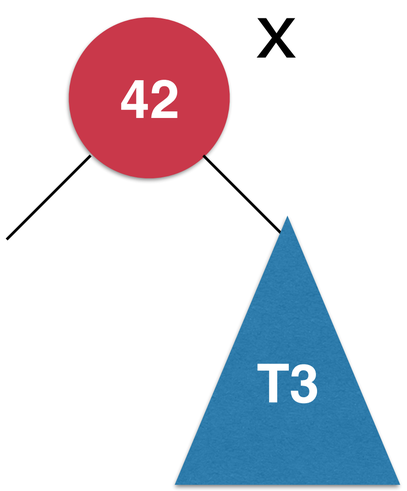
x



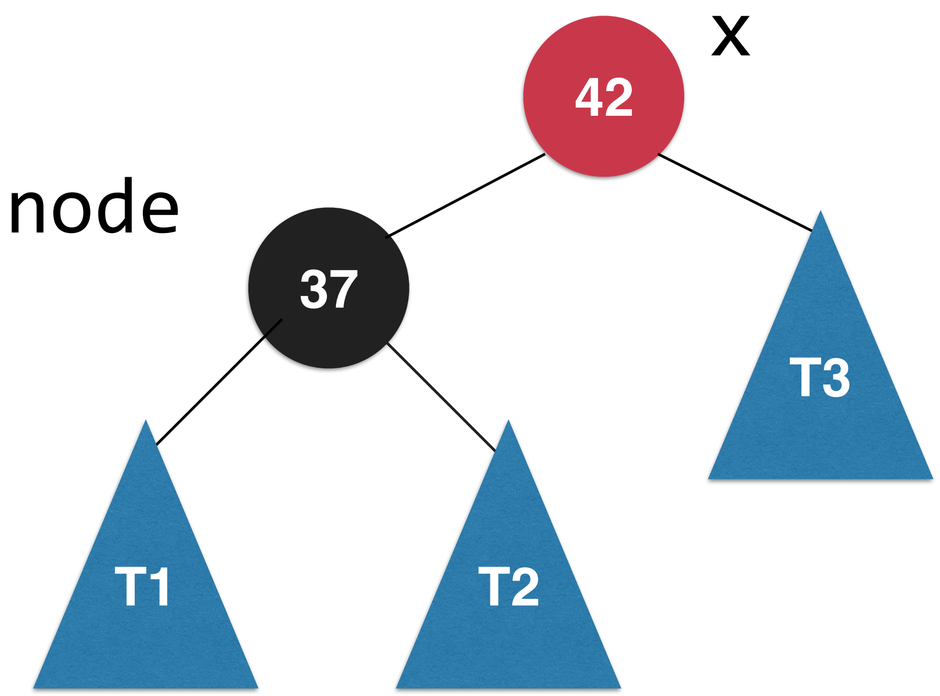
左旋转




```
node.right = x.left
x.left = node
```

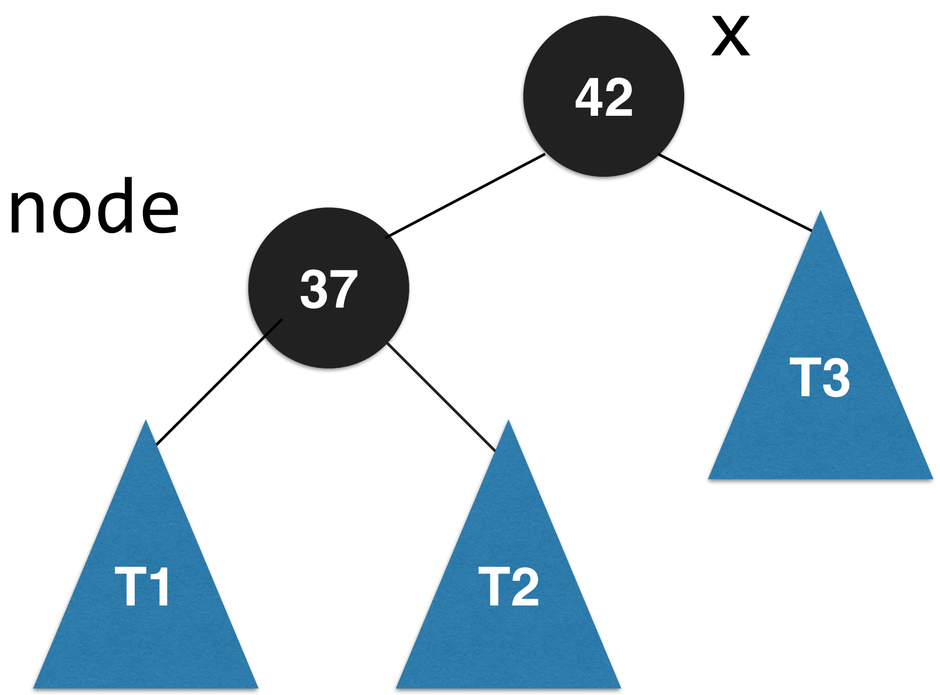


左旋转



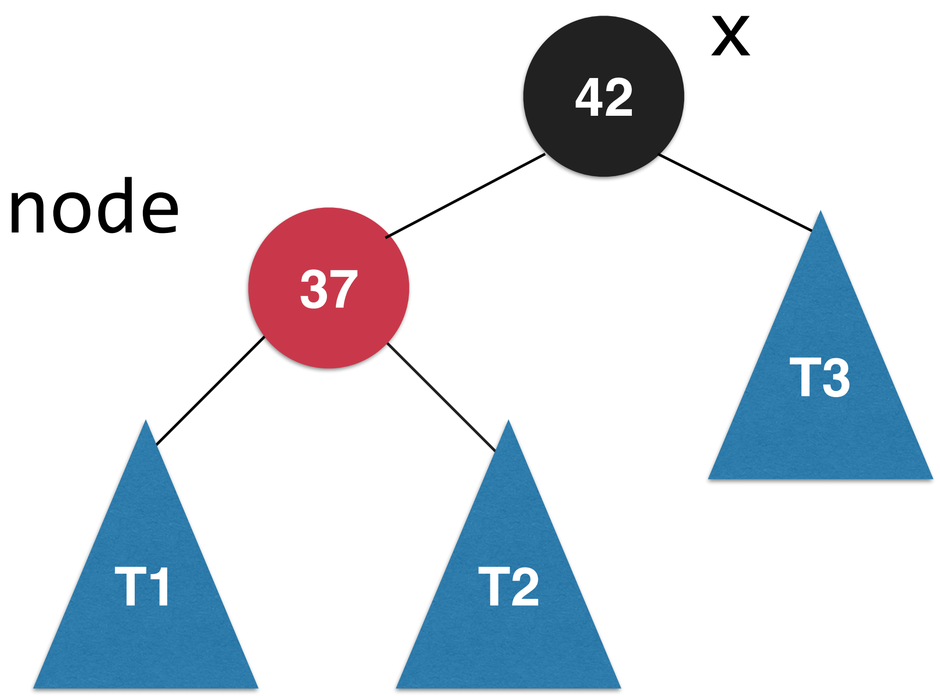

```
node.right = x.left
x.left = node
x.color = node.color
```

左旋转




```
node.right = x.left
x.left = node
x.color = node.color
```

左旋转




```
node.right = x.left
x.left = node
x.color = node.color
node.color = RED
```

实践：左旋转

颜色翻转和右旋转

红黑树添加新元素

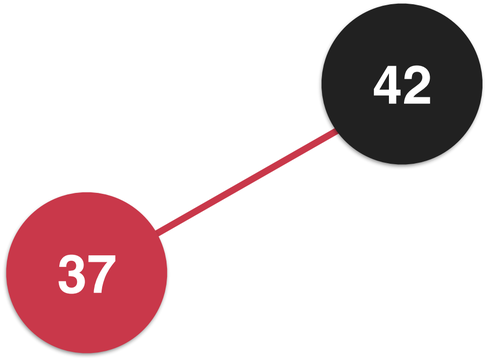
42

红黑树添加新元素

红黑树添加新元素

红黑树添加新元素

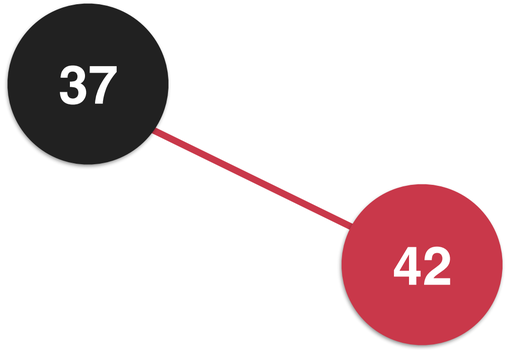
红黑树添加新元素



红黑树添加新元素

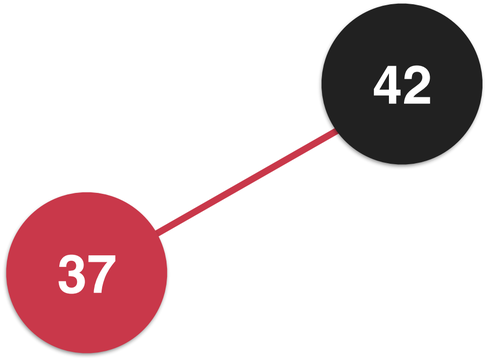
红黑树添加新元素

此时需要进行左旋转



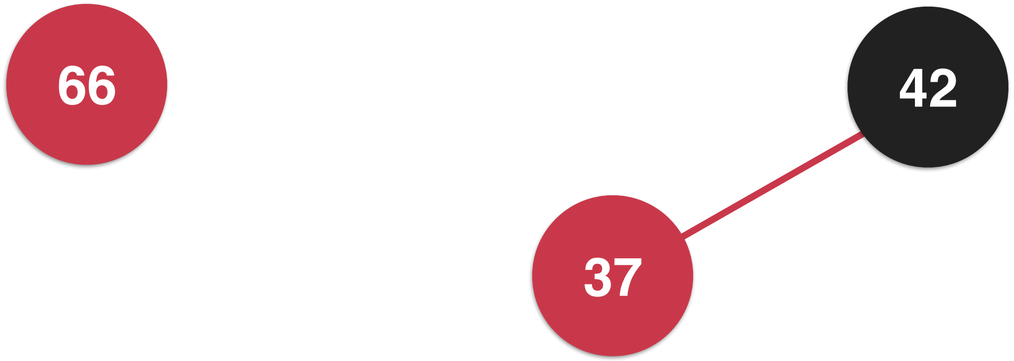
红黑树添加新元素

此时需要进行左旋转

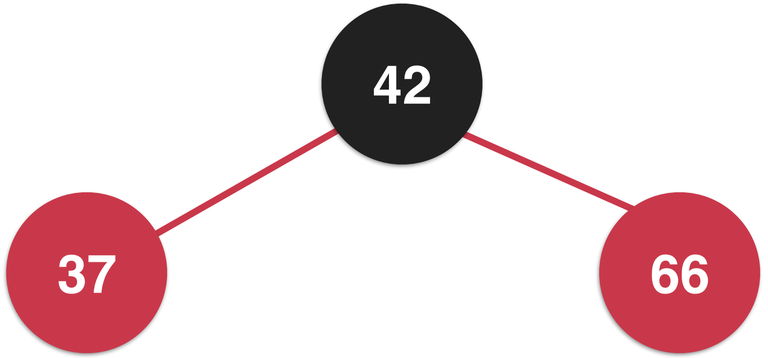


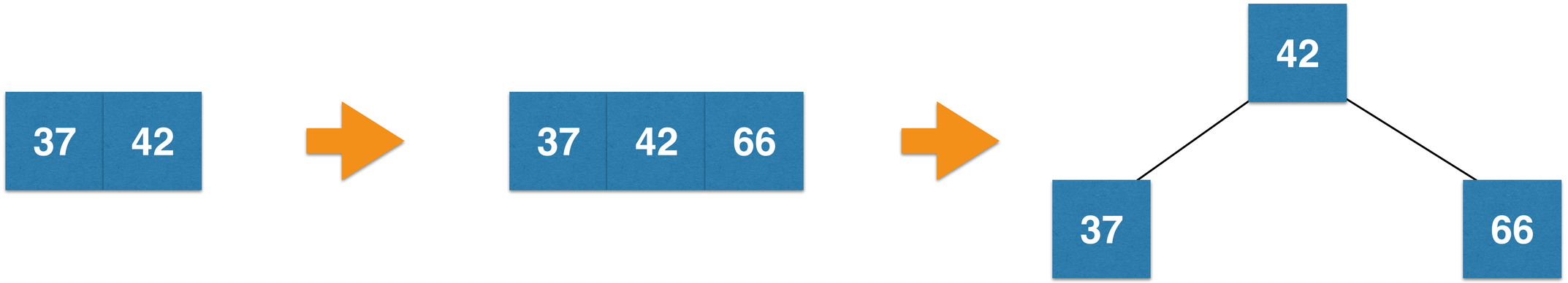
向红黑树中的“3-node”添加元素

红黑树添加新元素

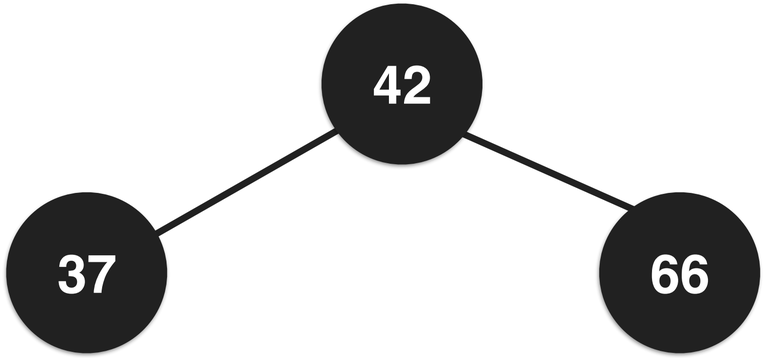


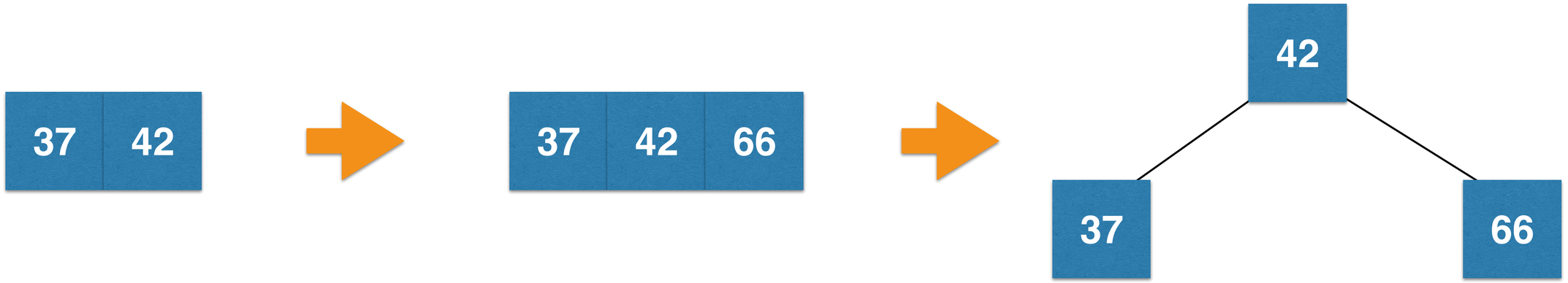
红黑树添加新元素





红黑树添加新元素

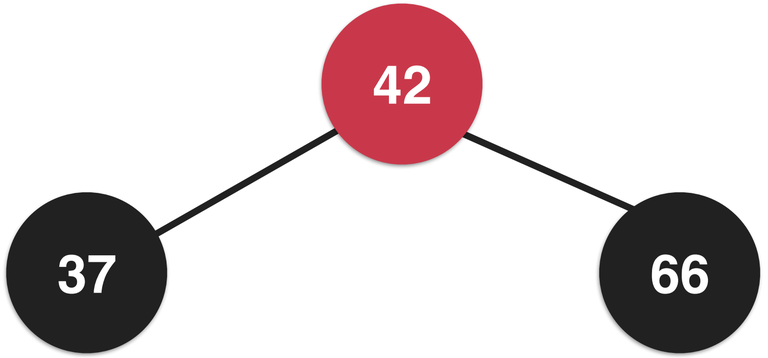


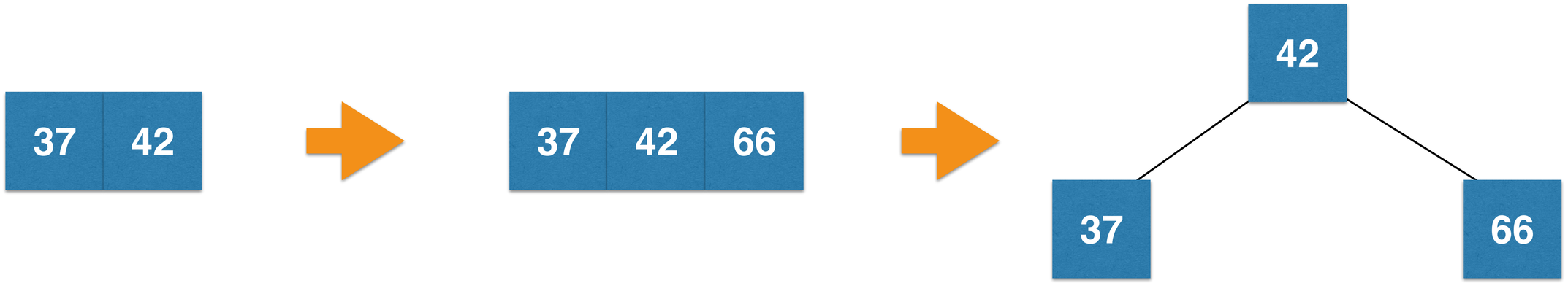


红黑树添加新元素

颜色翻转

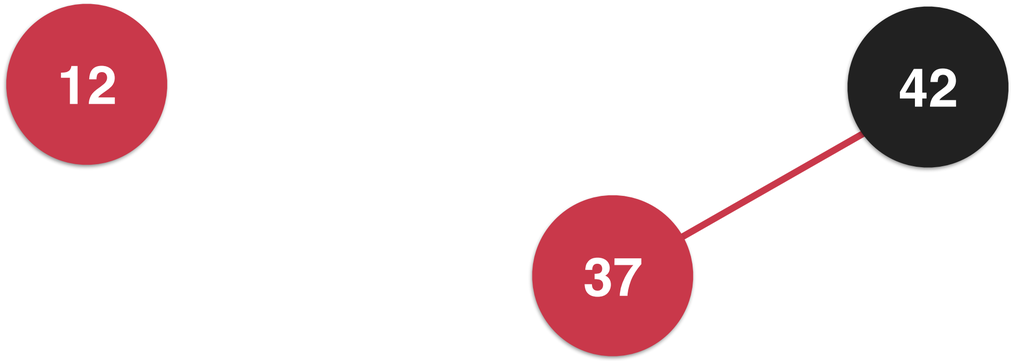
flipColors



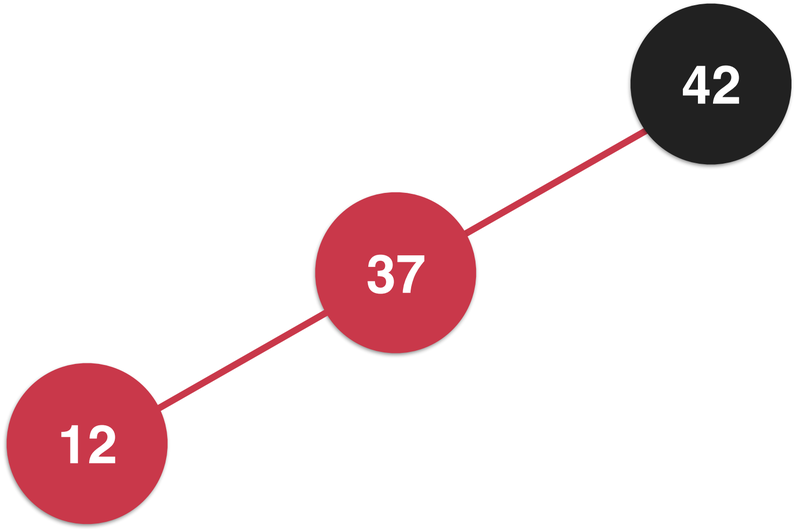


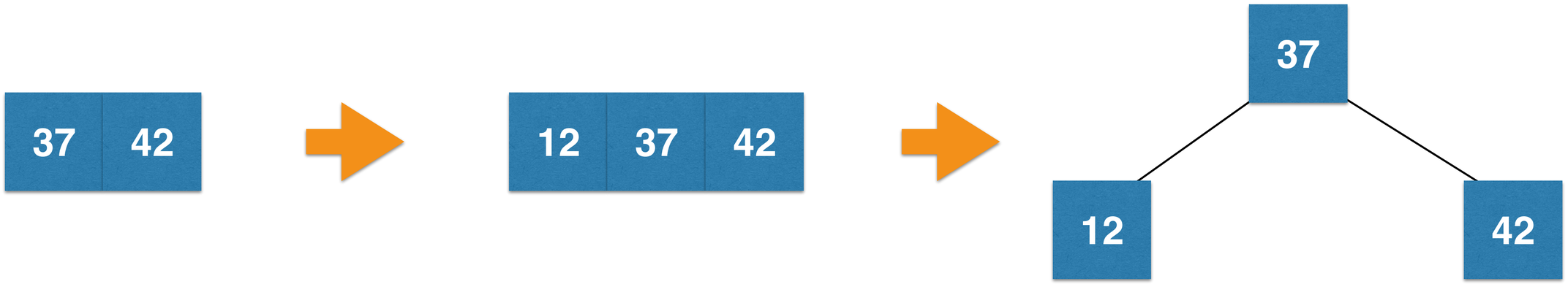
实践：颜色翻转

红黑树添加新元素

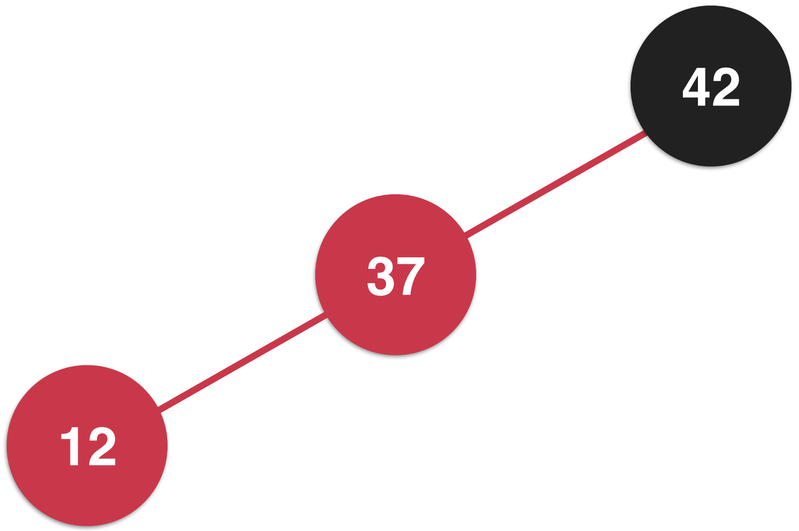


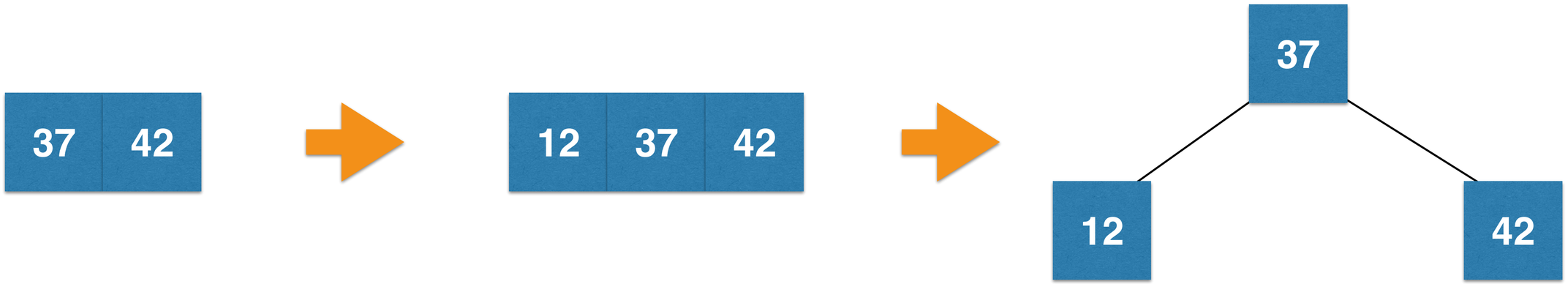
红黑树添加新元素



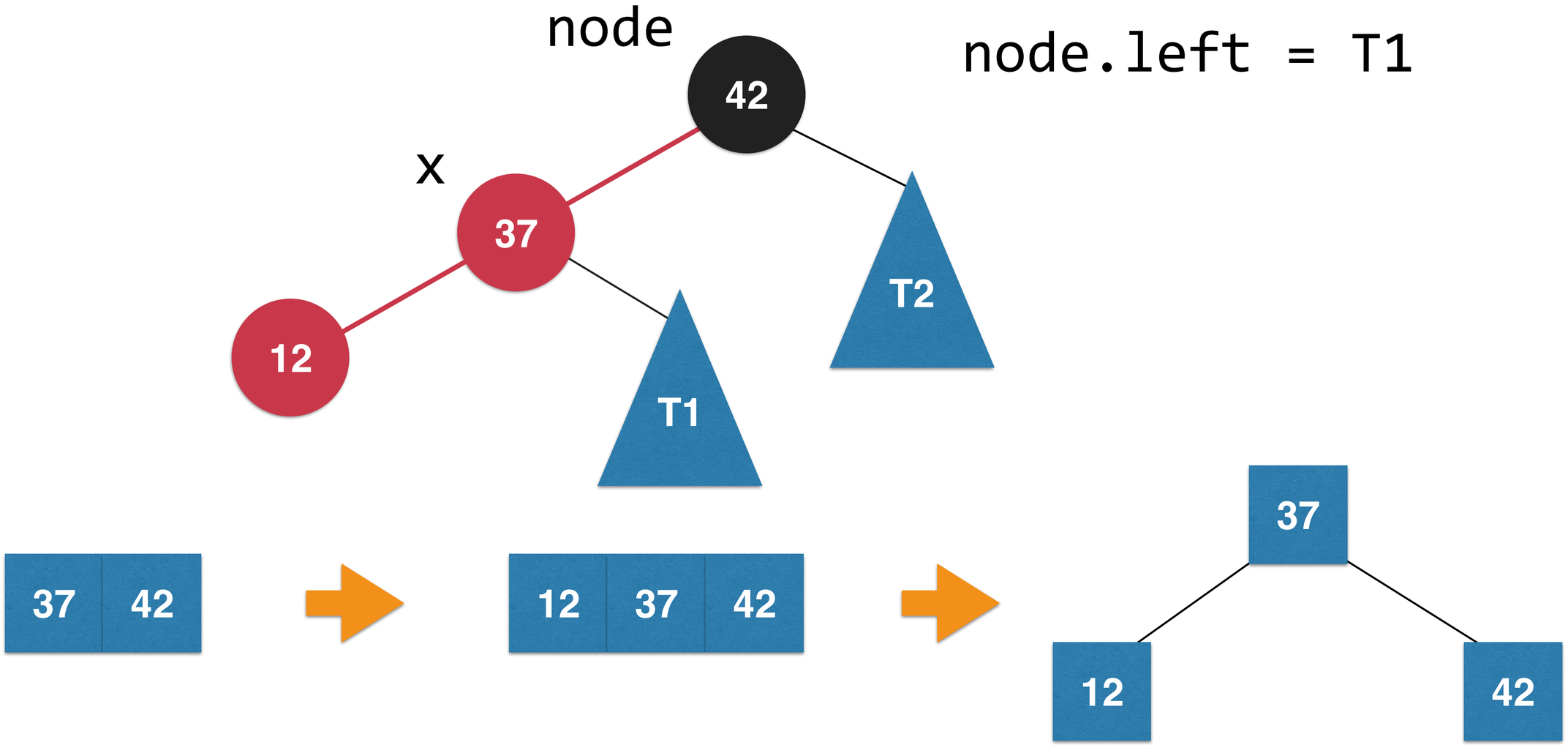


右旋转

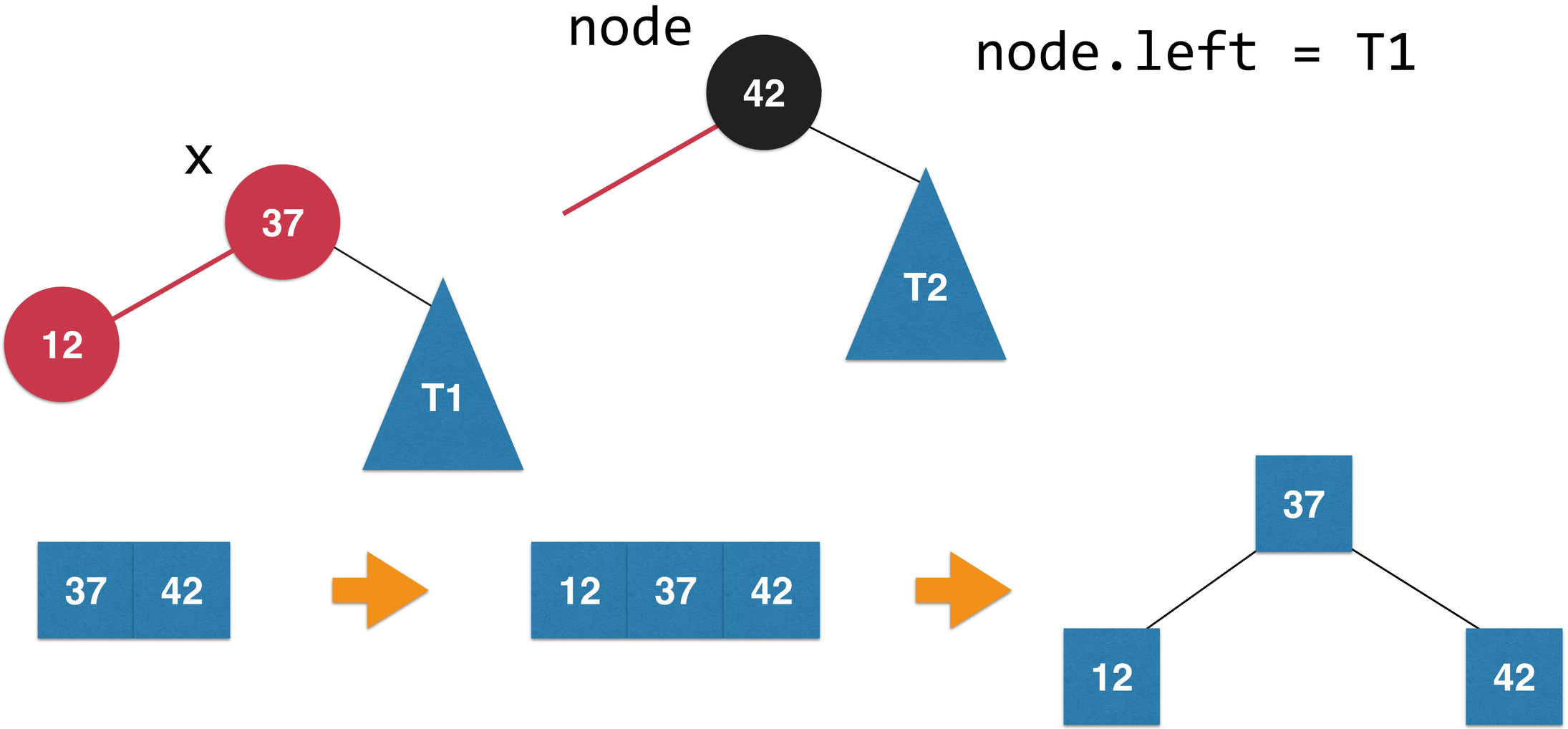




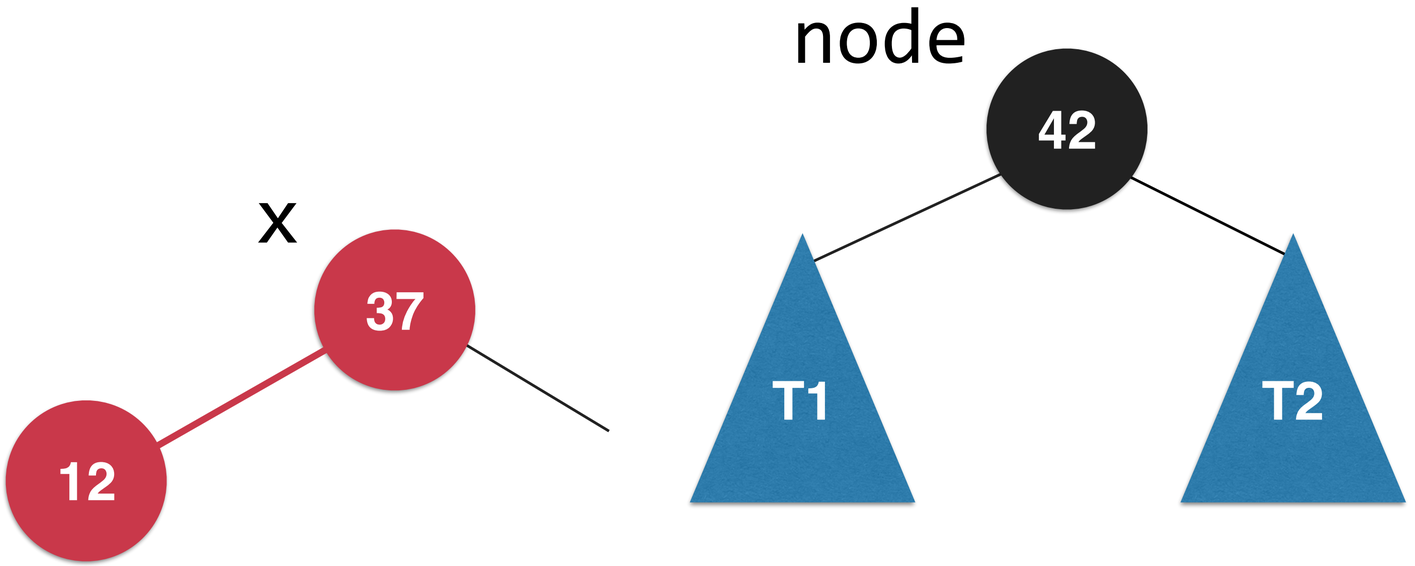
右旋转



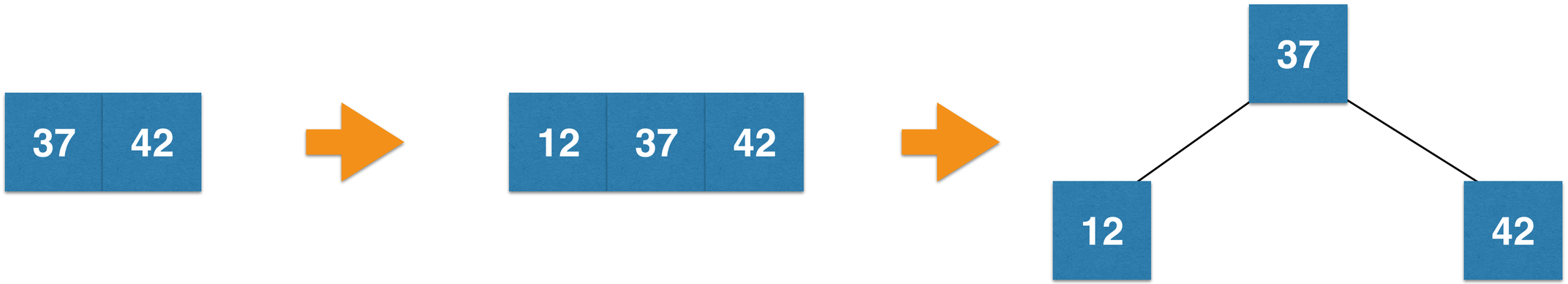
右旋转



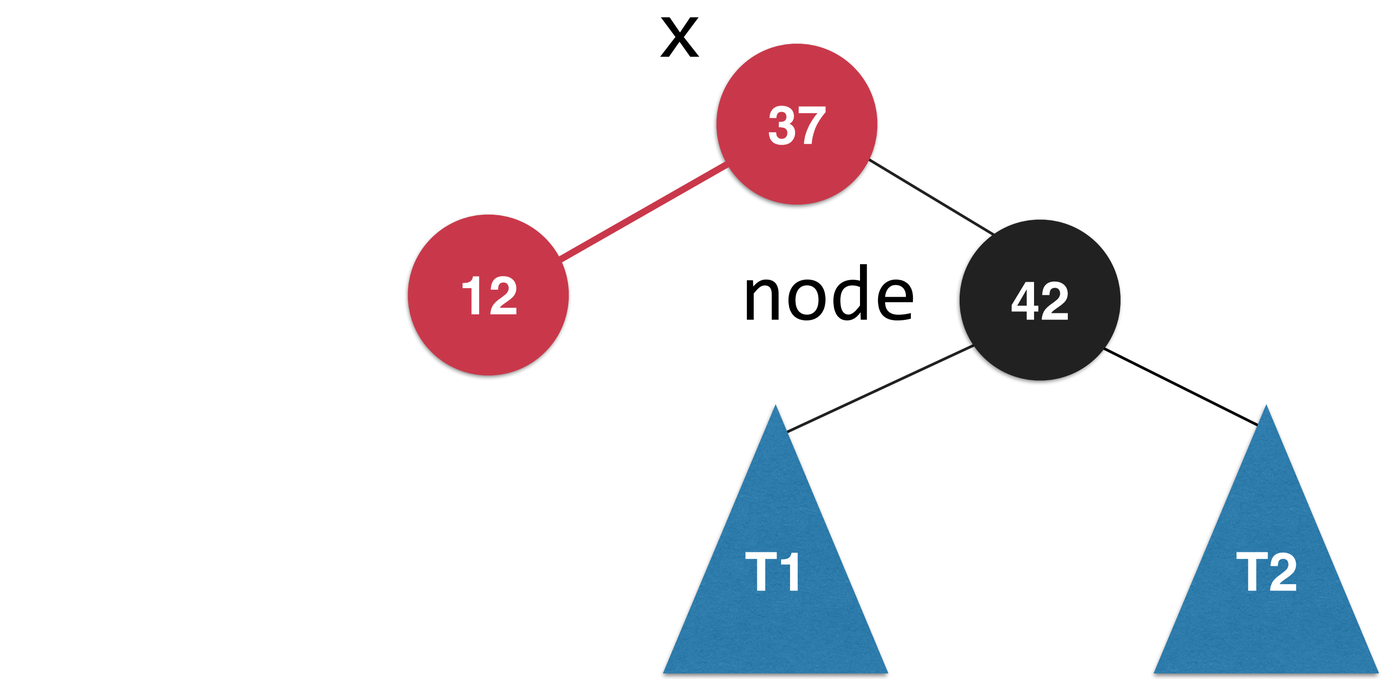
右旋转



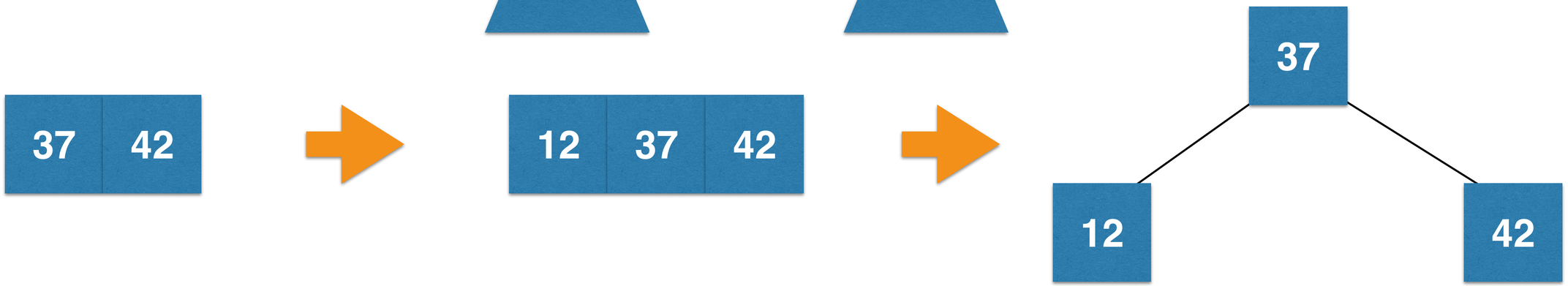

```
node.left = T1

x.right = node
```



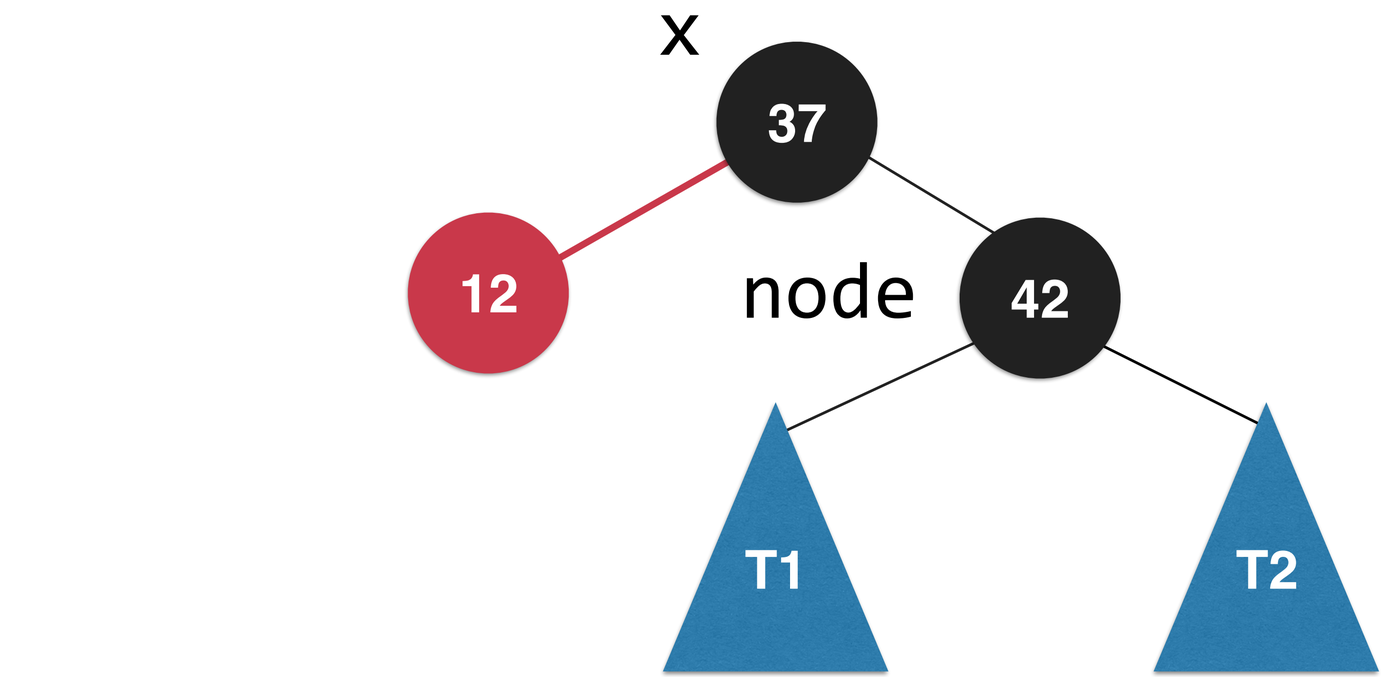
右旋转



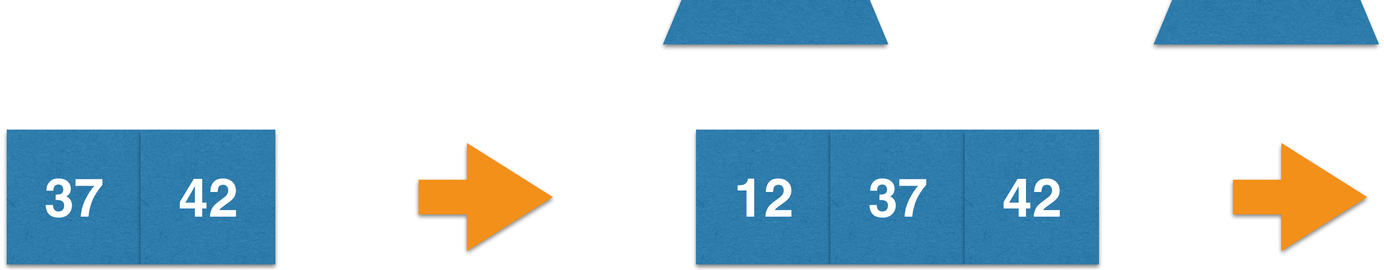

```
node.left = T1

x.right = node

x.color = node.color
```

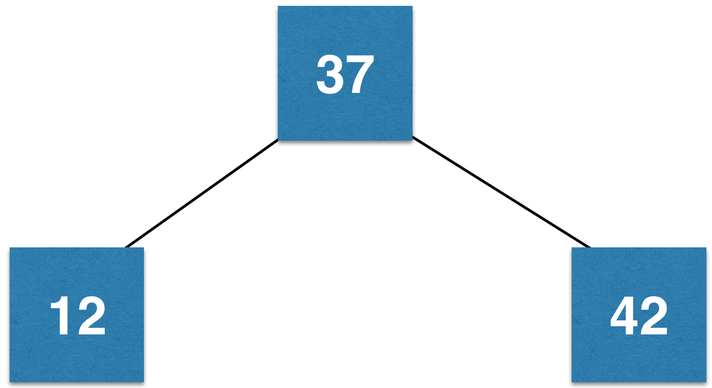


右旋转

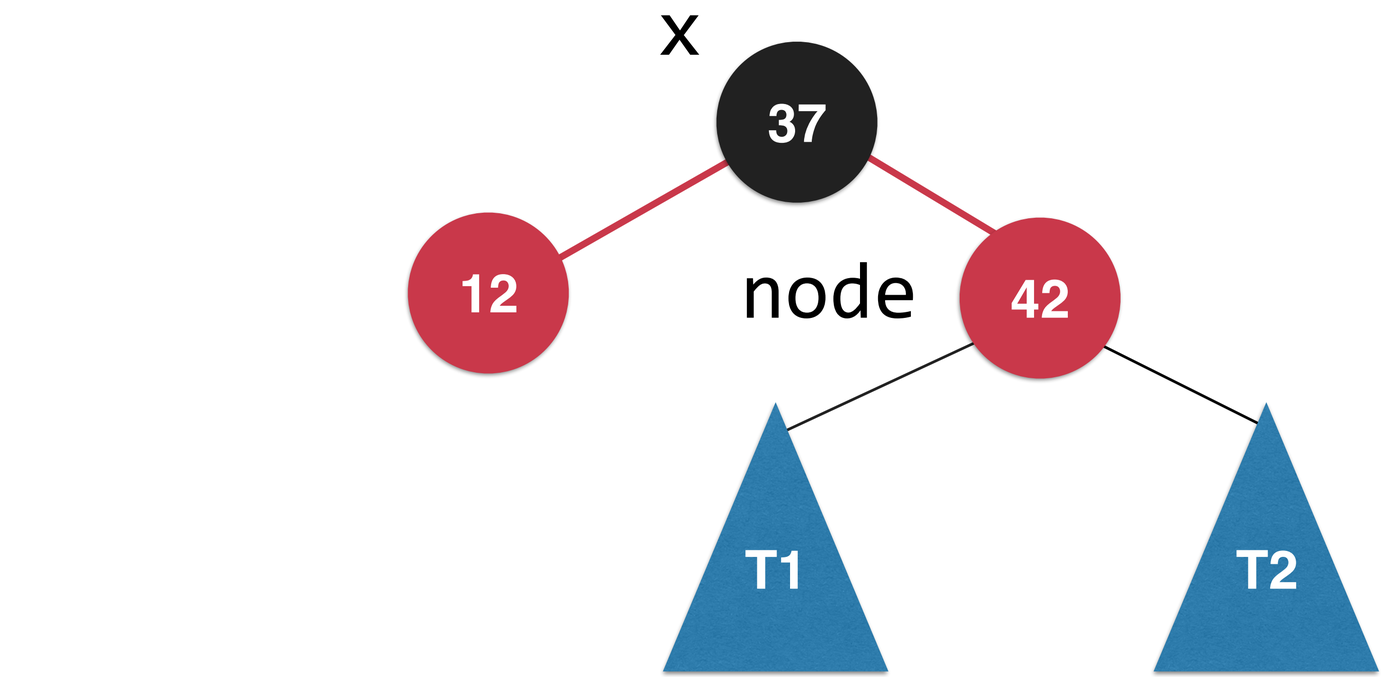



```
node.left = T1

x.right = node

x.color = node.color

node.color = RED
```

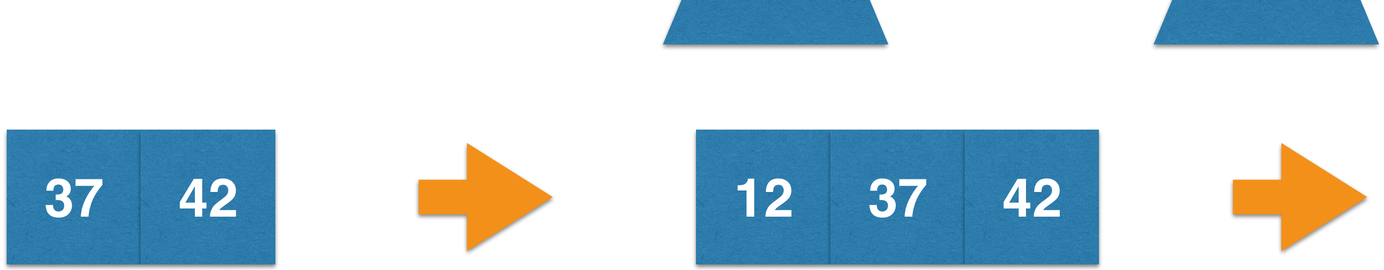


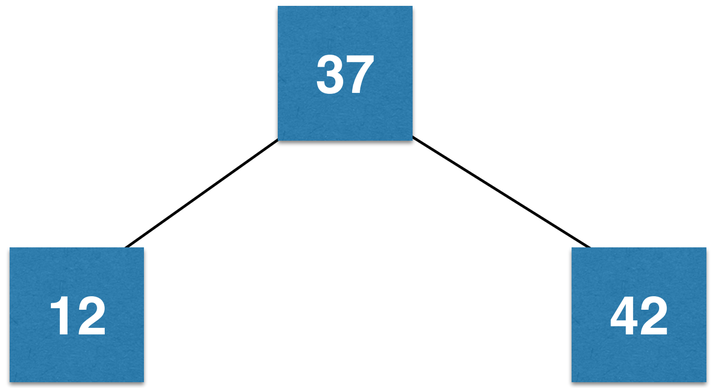


右旋转

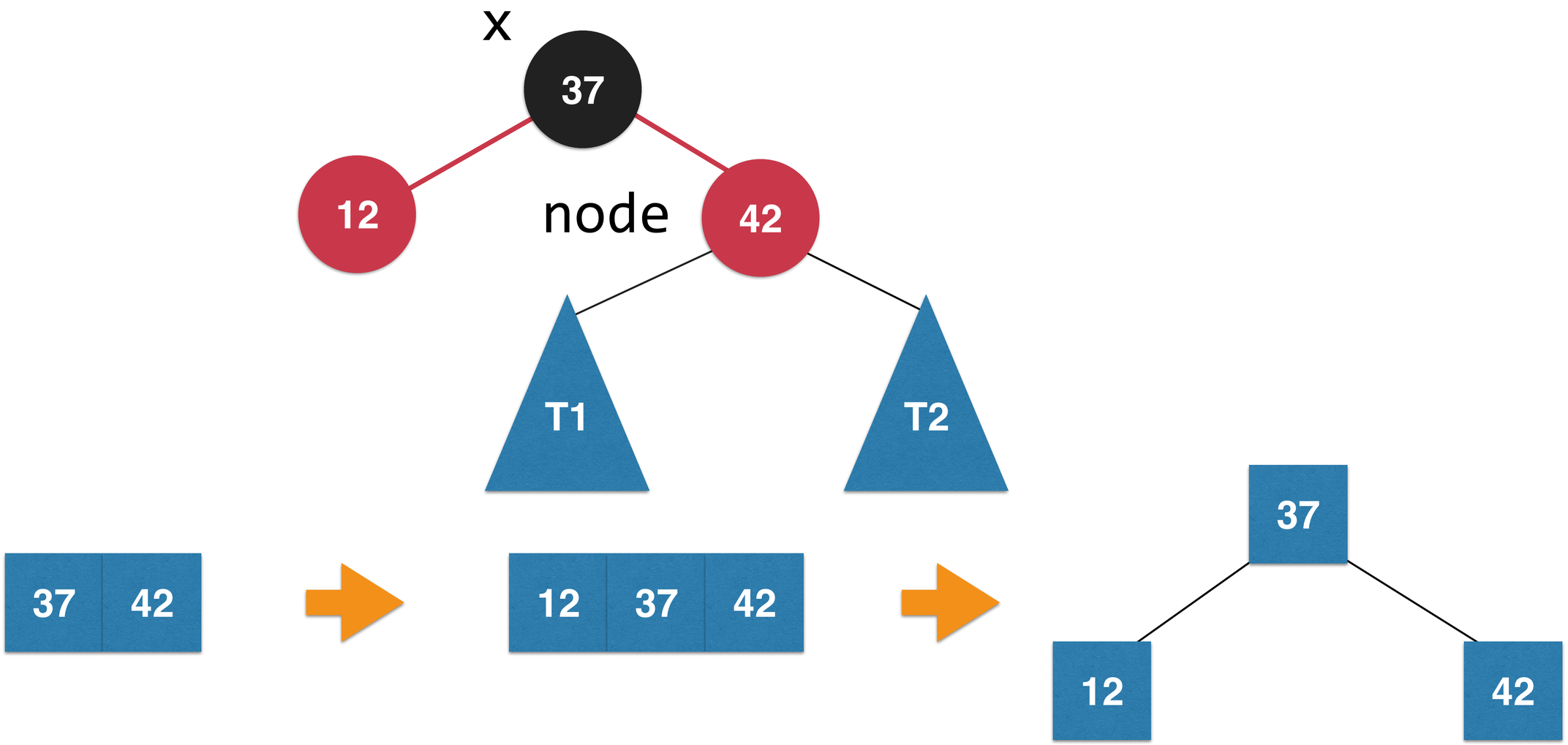



```
node.left = T1

x.right = node

x.color = node.color

node.color = RED
```

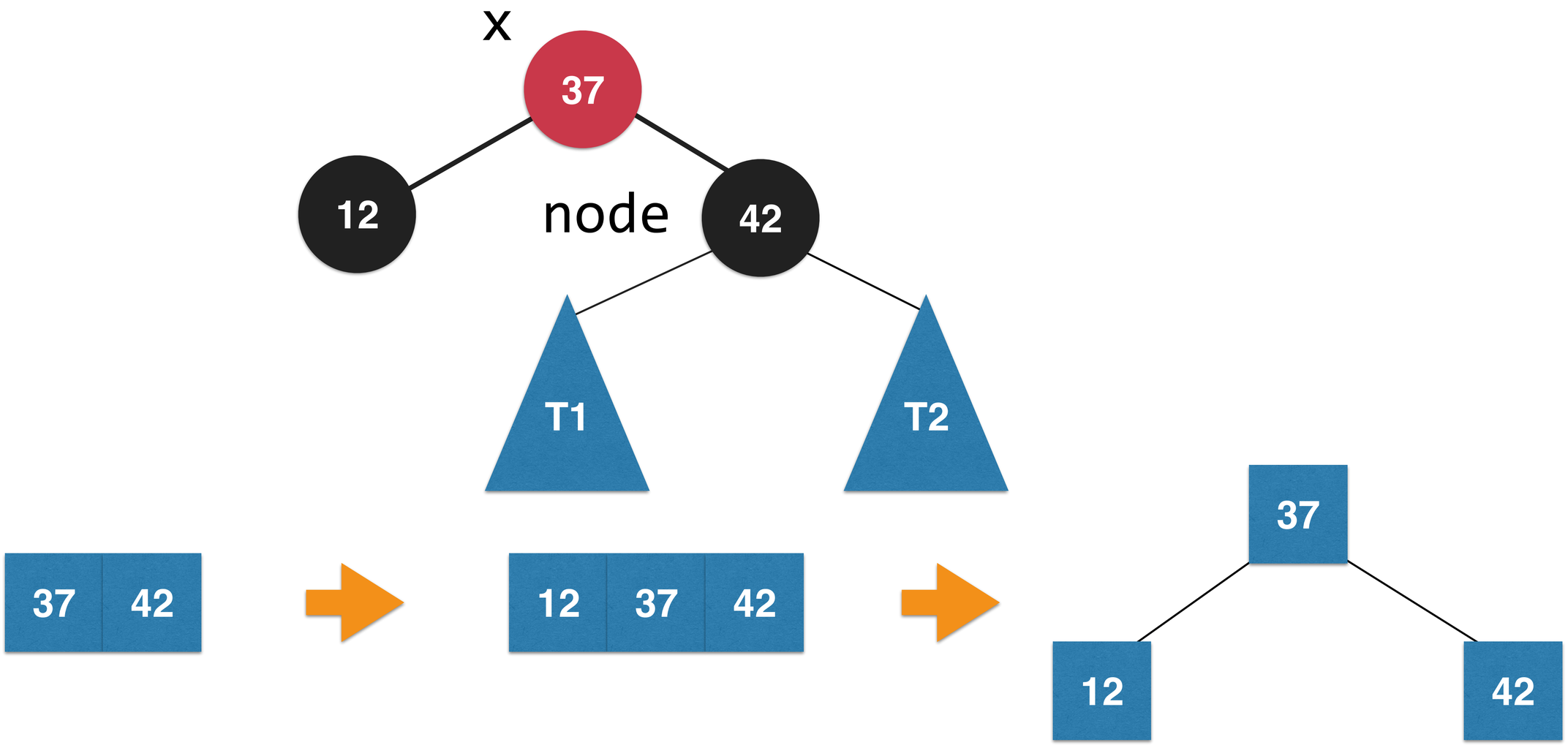




颜色翻转



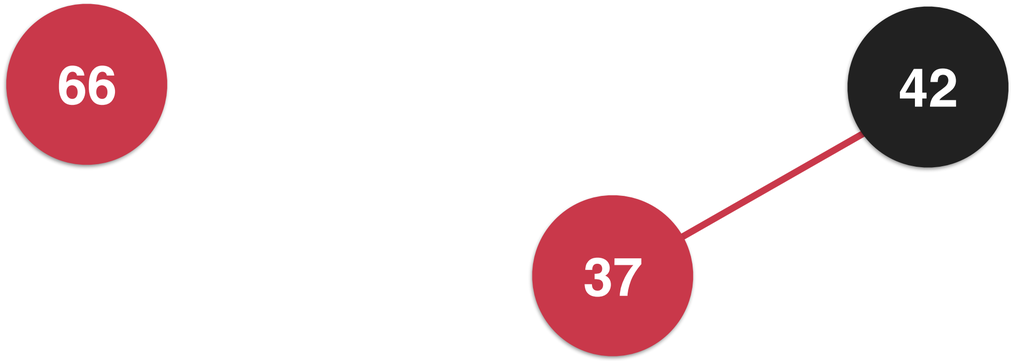
颜色翻转



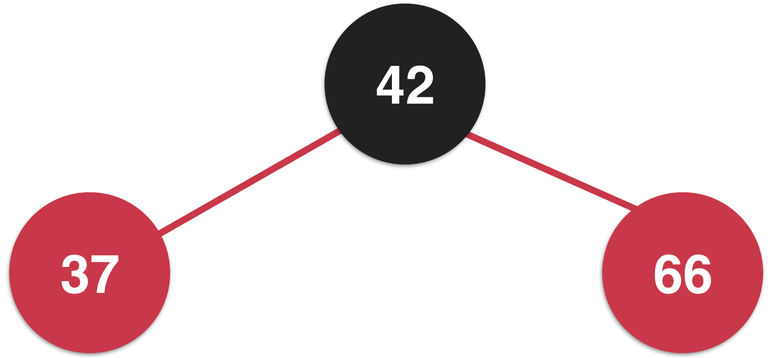
实践：右旋转

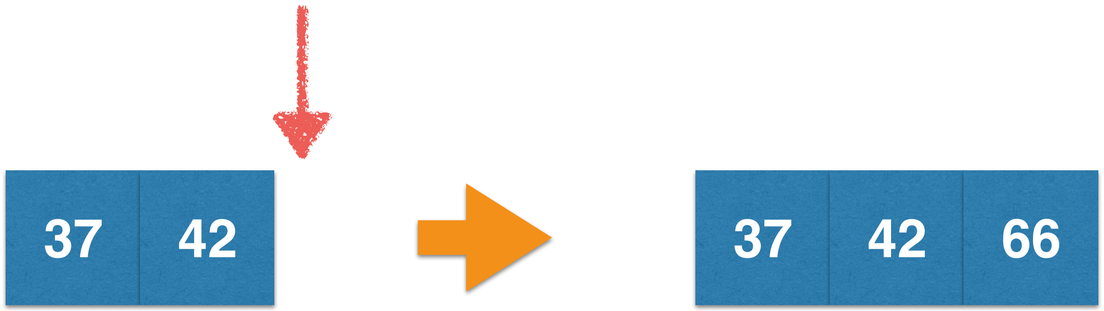
向红黑树中添加新节点

红黑树添加新元素

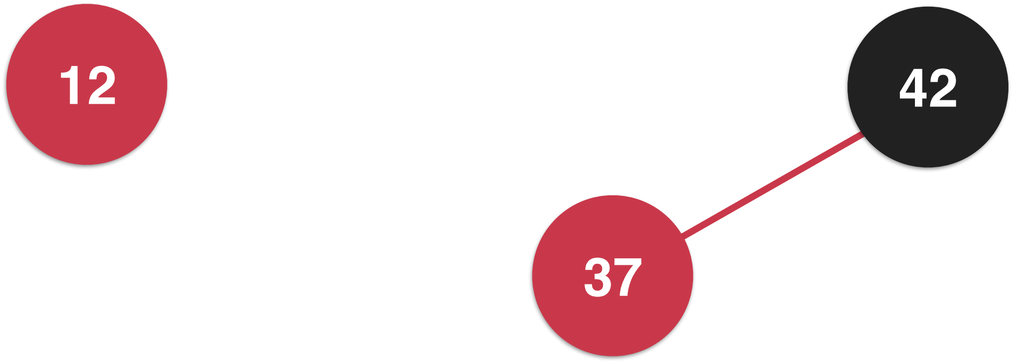


红黑树添加新元素

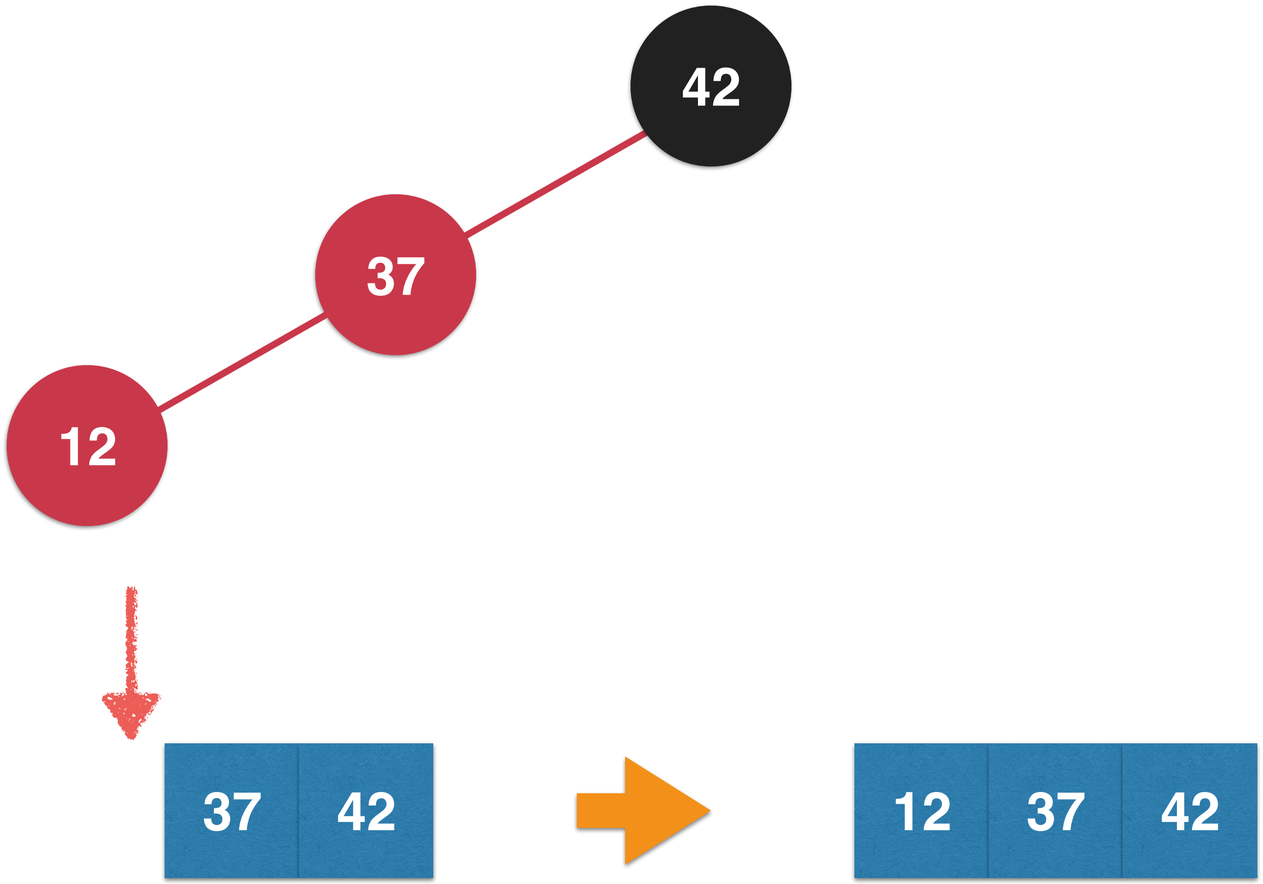




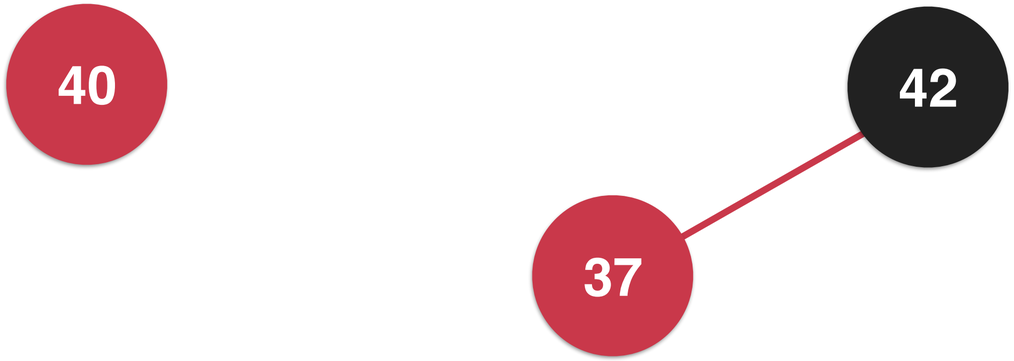
红黑树添加新元素



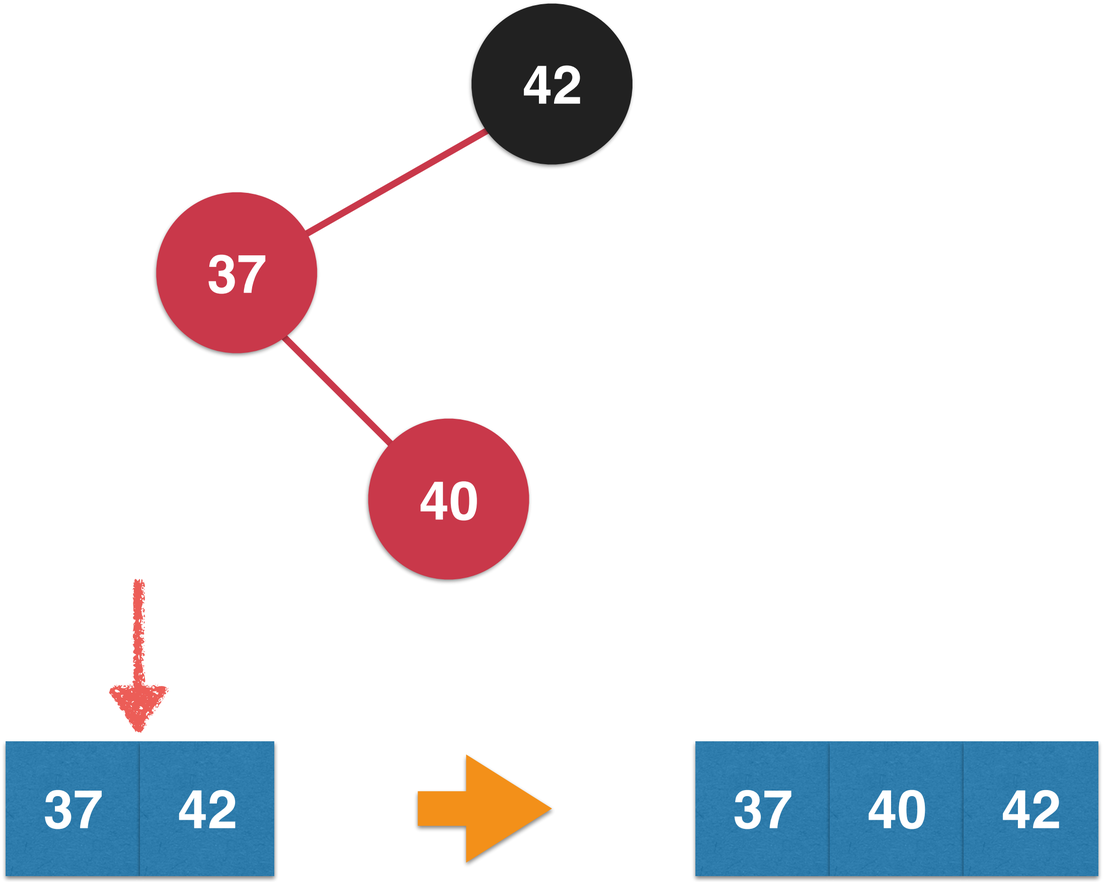
红黑树添加新元素



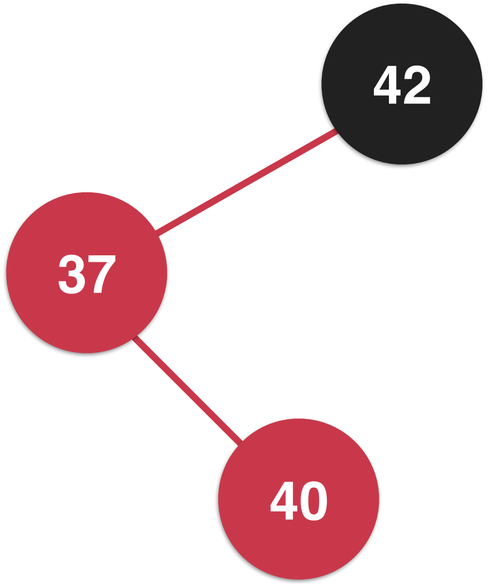
红黑树添加新元素



红黑树添加新元素

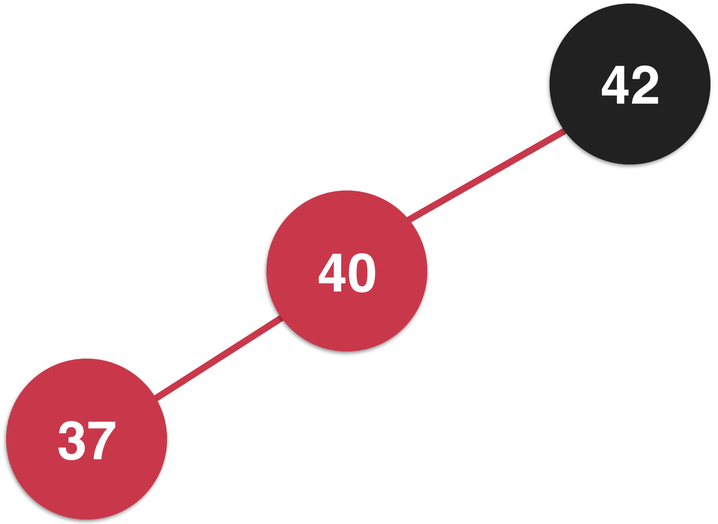


红黑树添加新元素



左旋转

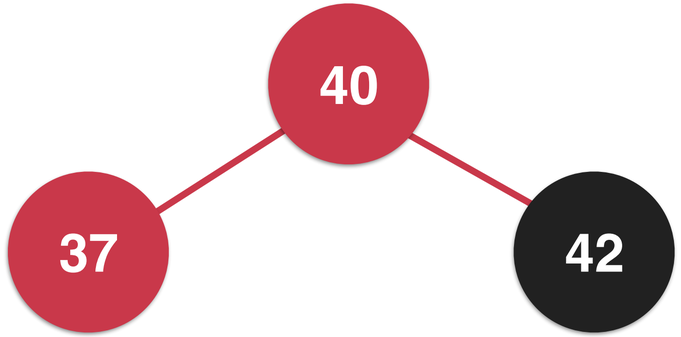
红黑树添加新元素



左旋转

右旋转

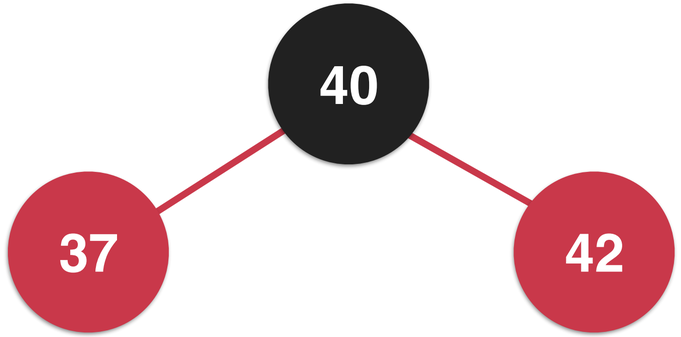
红黑树添加新元素



左旋转

右旋转

红黑树添加新元素

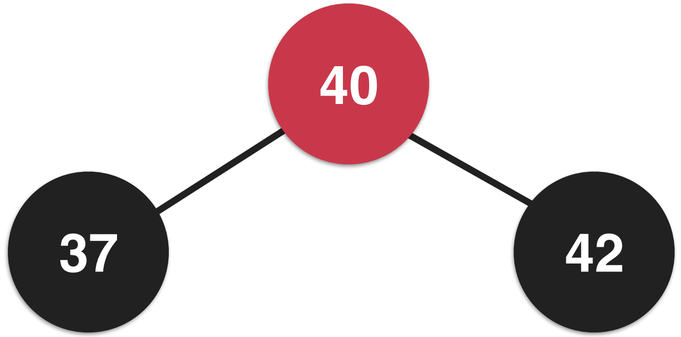


左旋转

右旋转

颜色翻转

红黑树添加新元素



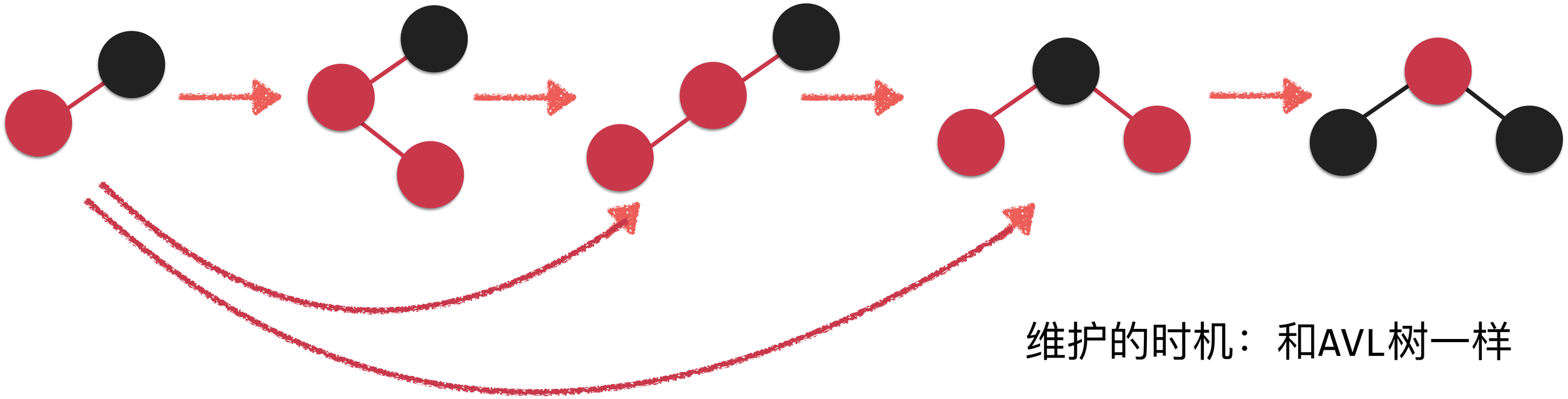
左旋转

右旋转

颜色翻转

红黑树添加新元素

左旋转　　右旋转　　颜色翻转



维护的时机：和AVL树一样

添加节点后回溯向上维护

实践：向红黑树中添加新节点

红黑树的性能

实践：红黑树的性能

红黑树的性能总结

对于完全随机的数据，普通的二分搜索树很好用！

缺点：极端情况退化成链表（或者高度不平衡）

对于查询较多的使用情况，AVL树很好用！

红黑树牺牲了平衡性（2logn的高度）

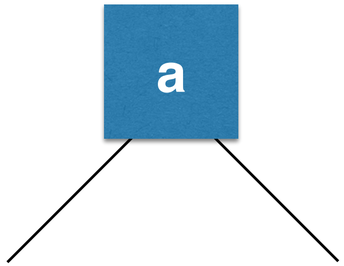
统计性能更优（综合增删改查所有的操作）

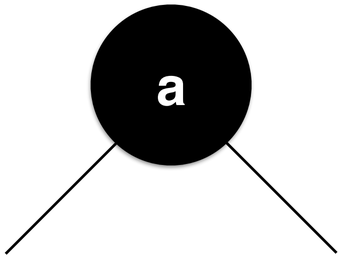
更多和红黑树相关的问题

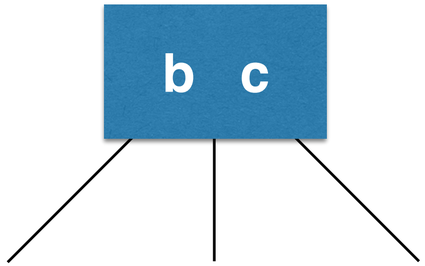
红黑树中删除节点

左倾红黑树

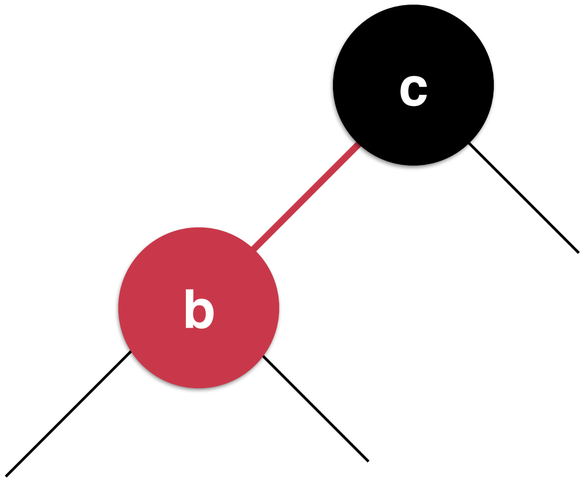
2节点





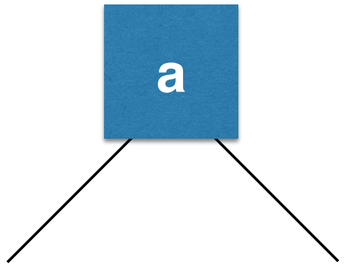


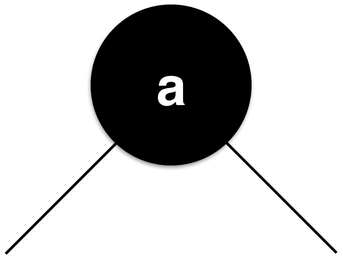
3节点

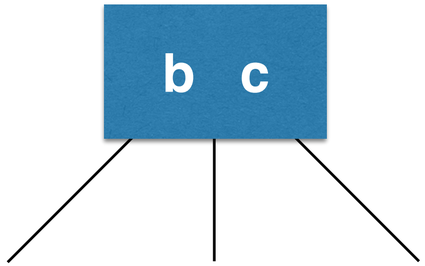


右倾红黑树

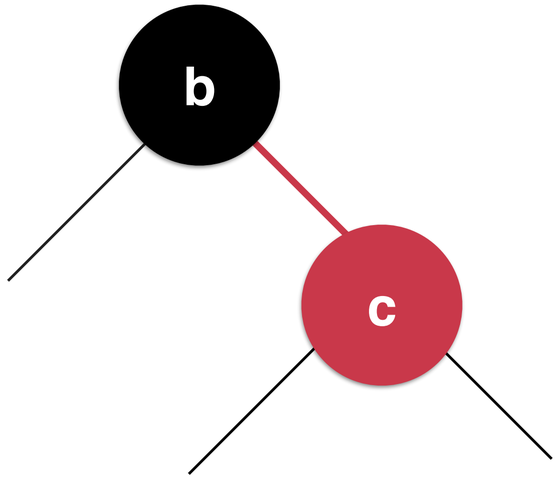
2节点







3节点



红黑树统计性能更优

另一种统计性能优秀的树结构：Splay Tree（伸展树）

局部性原理：刚被访问的内容下次高概率被再次访问

基于红黑树的Map和Set

java.util中的TreeMap和TreeSet基于红黑树：）

红黑树的其他实现

算法导论中红黑树的实现

红黑树

其他

欢迎大家关注我的个人公众号：是不是很酷

玩儿转数据结构

liuyubobobo