堆和优先队列

胡船长
初航我带你，远航靠自己

本章题目

1-应试. Leetcode-703：数据流中的第 K 大元素

2-校招. Leetcode-295：数据流的中位数

3-校招. Leetcode-23：合并 K 个升序链表

4-校招. Leetcode-264：丑数 II

5-校招. HZOJ-284：超市卖货

6-竞赛. HZOJ-285：序列 M 小和

7-竞赛. HZOJ-289：生日礼物

本期内容

一. 堆与优先队列

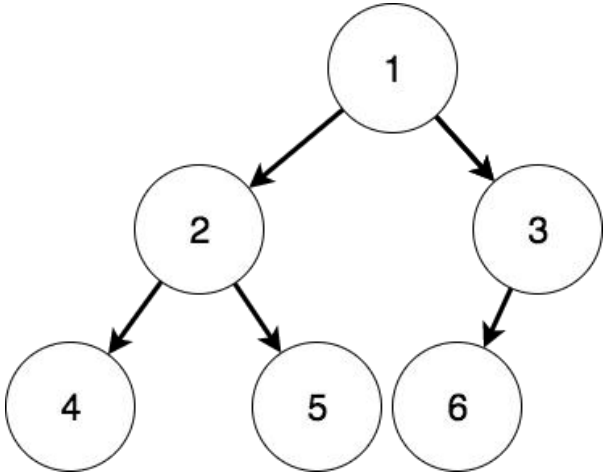
二. 堆排序与线性建堆法

三. 优化：哈夫曼编码

一. 堆与优先队列

二叉树：完全二叉树

完全二叉树
(complete binary tree)



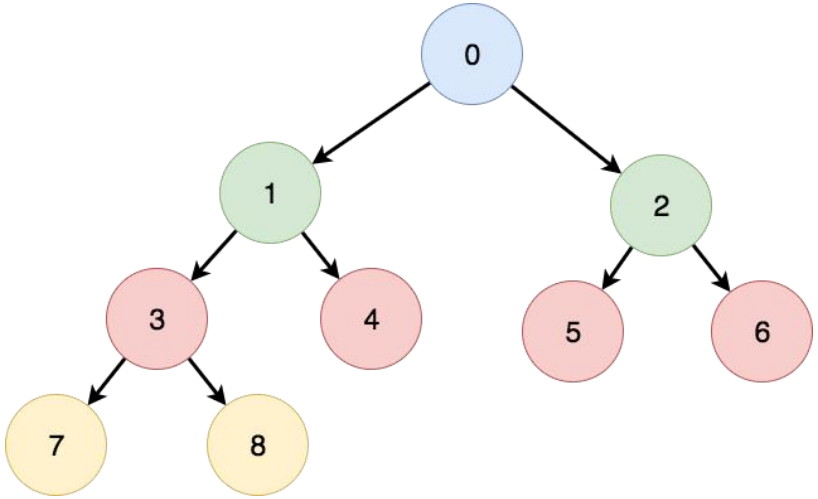
1、编号为 i 的子节点：

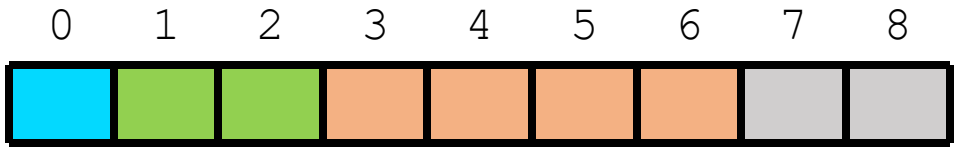
左孩子编号：2 * i

右孩子编号：2 * i + 1

2、可以用连续空间存储（数组）

二叉树：完全二叉树



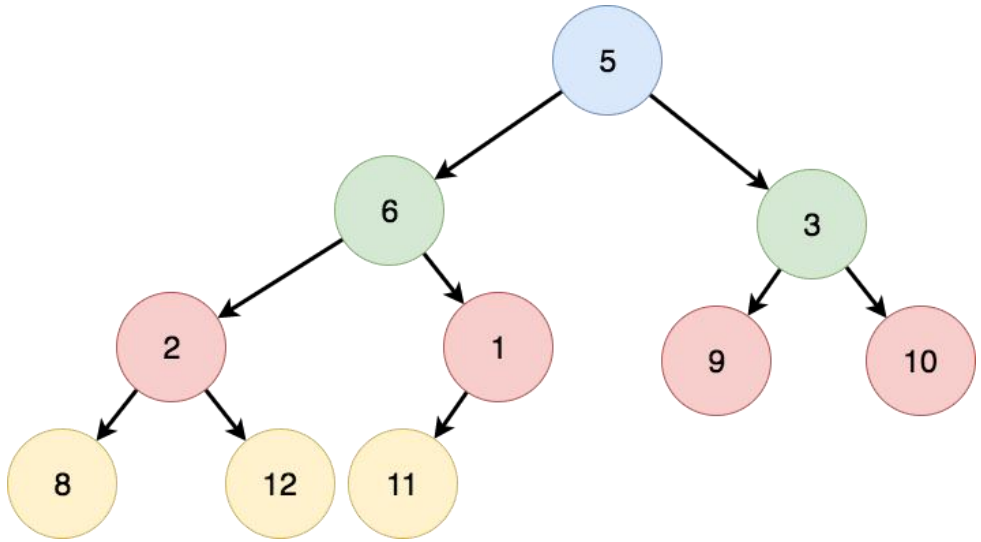


二叉树：完全二叉树

画出以下数组代表的完全二叉树

0	1	2	3	4	5	6	7	8	9
5	6	3	2	1	9	10	8	12	11

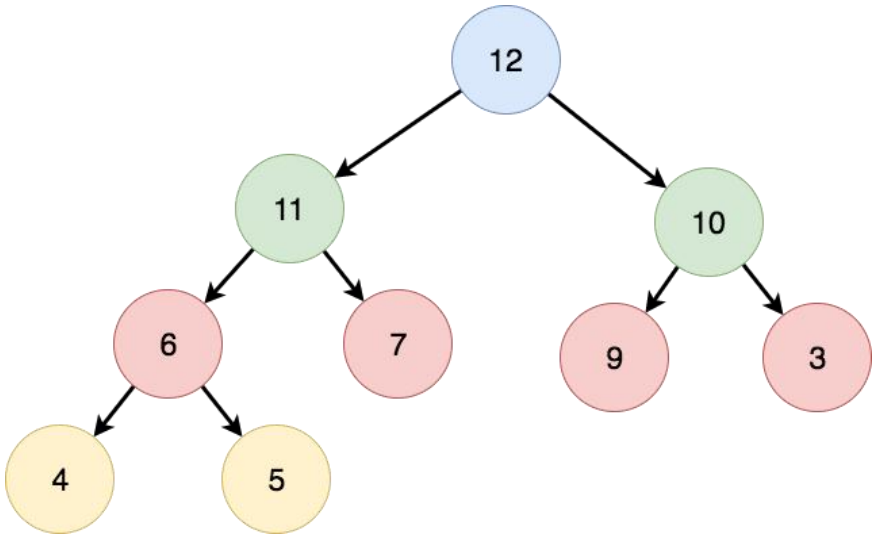
二叉树：完全二叉树



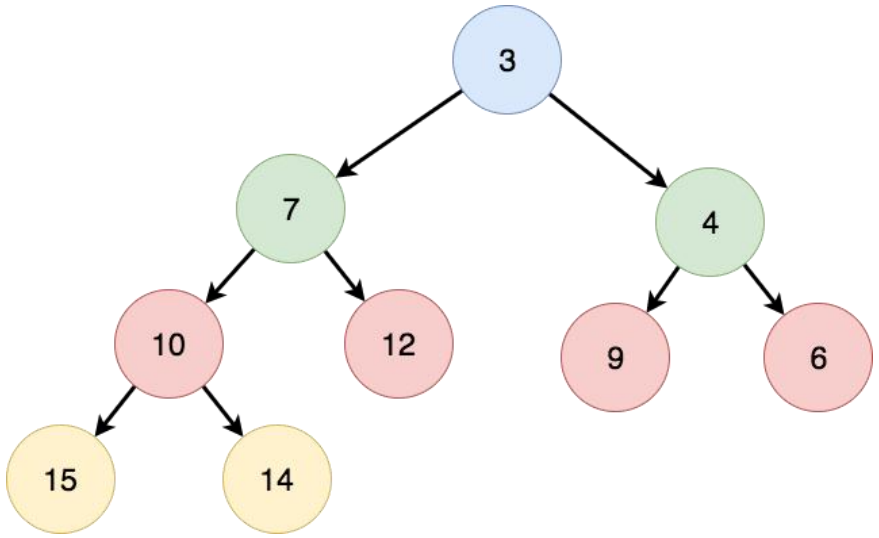
画出以下数组代表的完全二叉树

0	1	2	3	4	5	6	7	8	9
5	6	3	2	1	9	10	8	12	11

堆：结构讲解

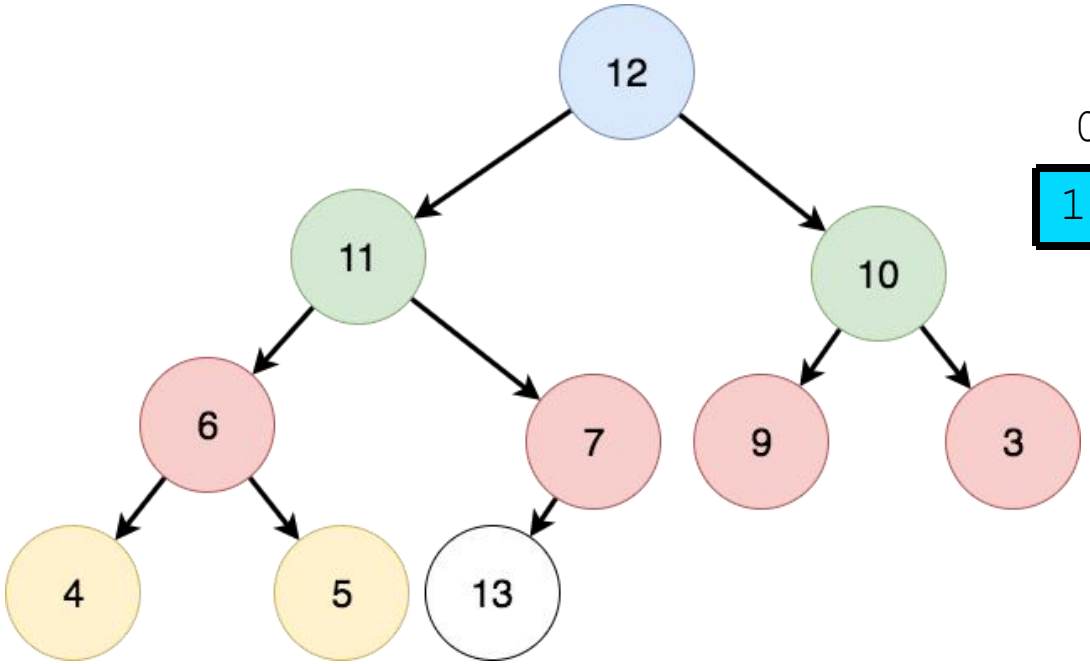


大顶堆



小顶堆

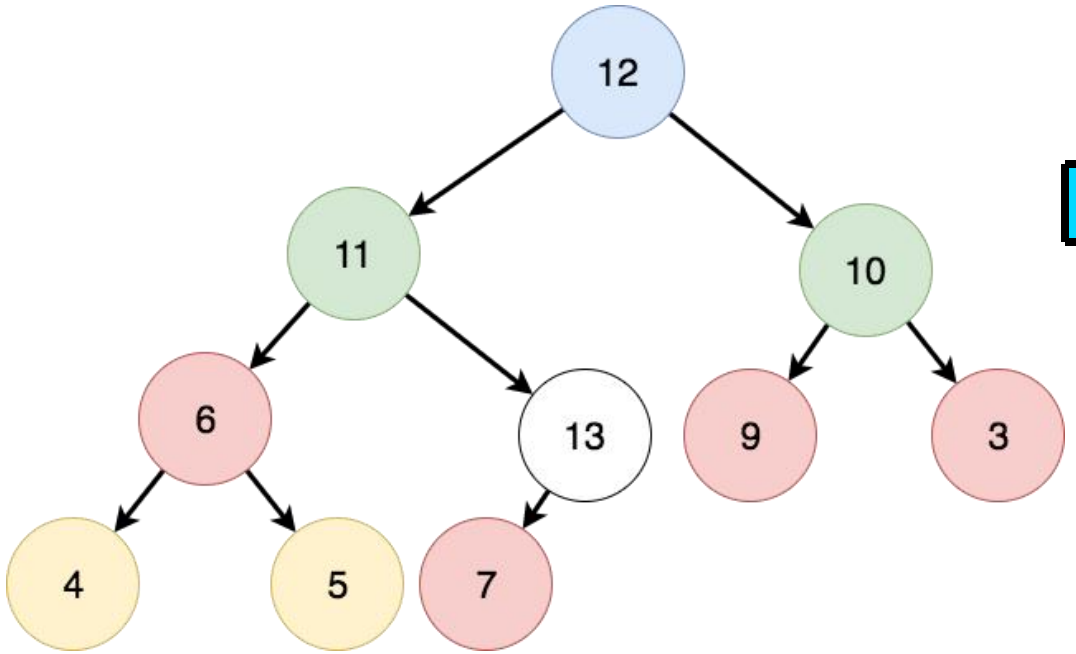
堆：尾部插入调整



大顶堆

0	1	2	3	4	5	6	7	8	9
12	11	10	6	7	9	3	4	5	13

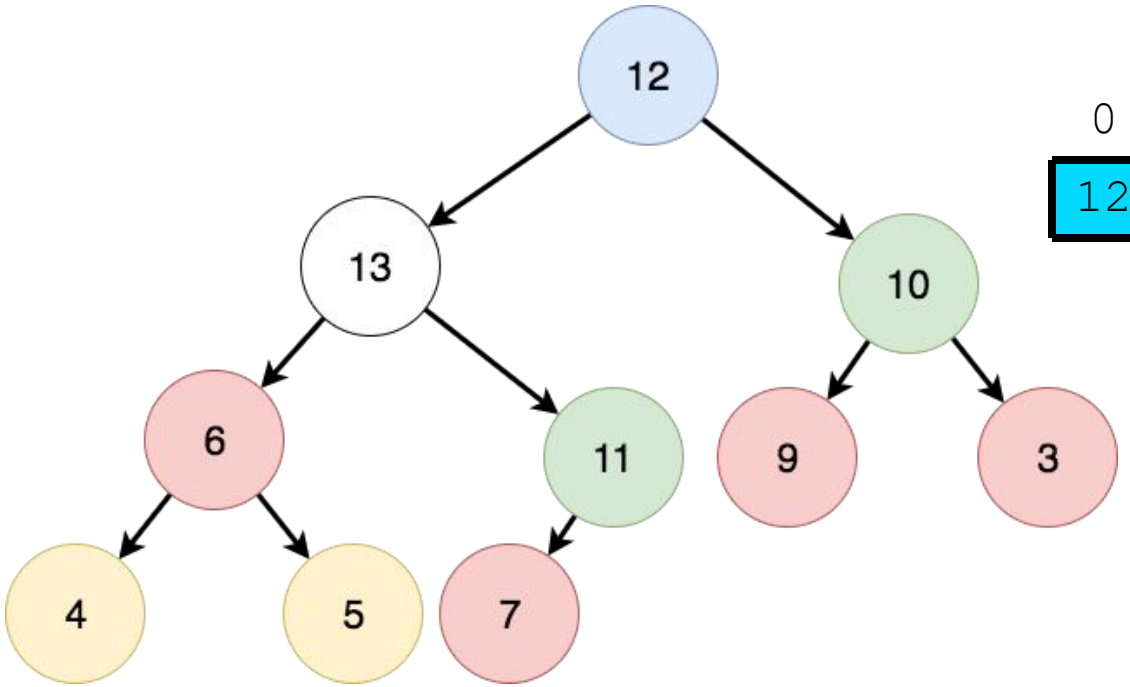
堆：尾部插入调整



大顶堆

0	1	2	3	4	5	6	7	8	9
12	11	10	6	13	9	3	4	5	7

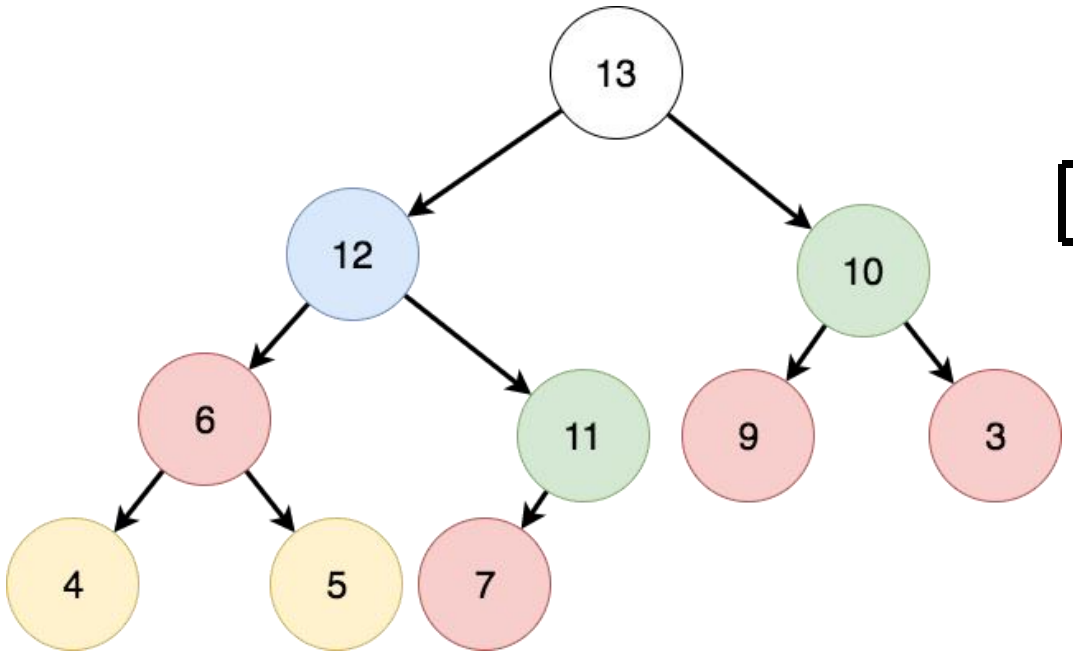
堆：尾部插入调整



大顶堆

0	1	2	3	4	5	6	7	8	9
12	13	10	6	11	9	3	4	5	7

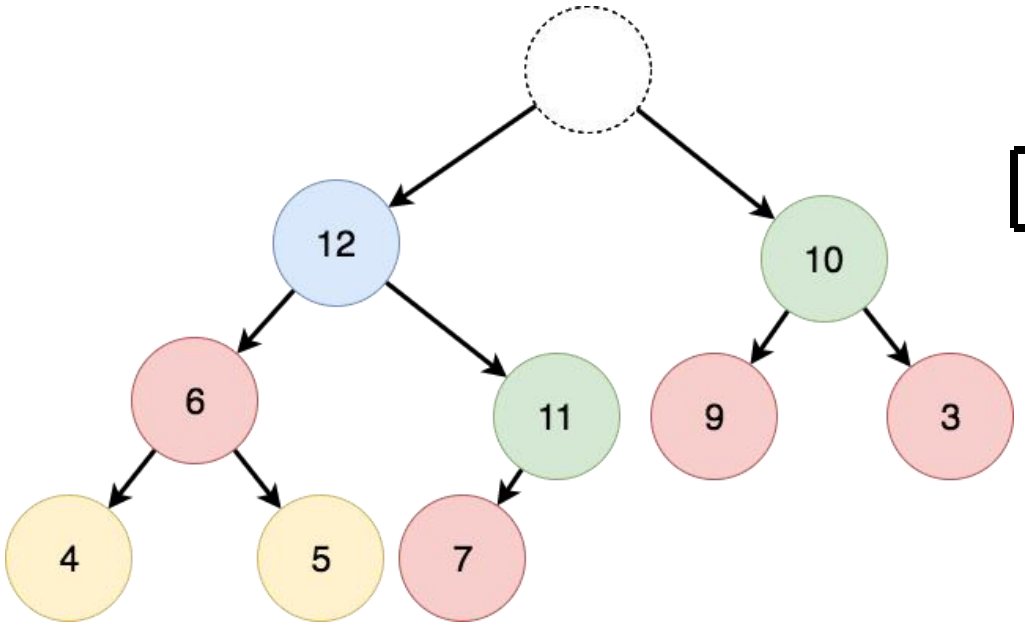
堆：尾部插入调整



大顶堆

0	1	2	3	4	5	6	7	8	9
13	12	10	6	11	9	3	4	5	7

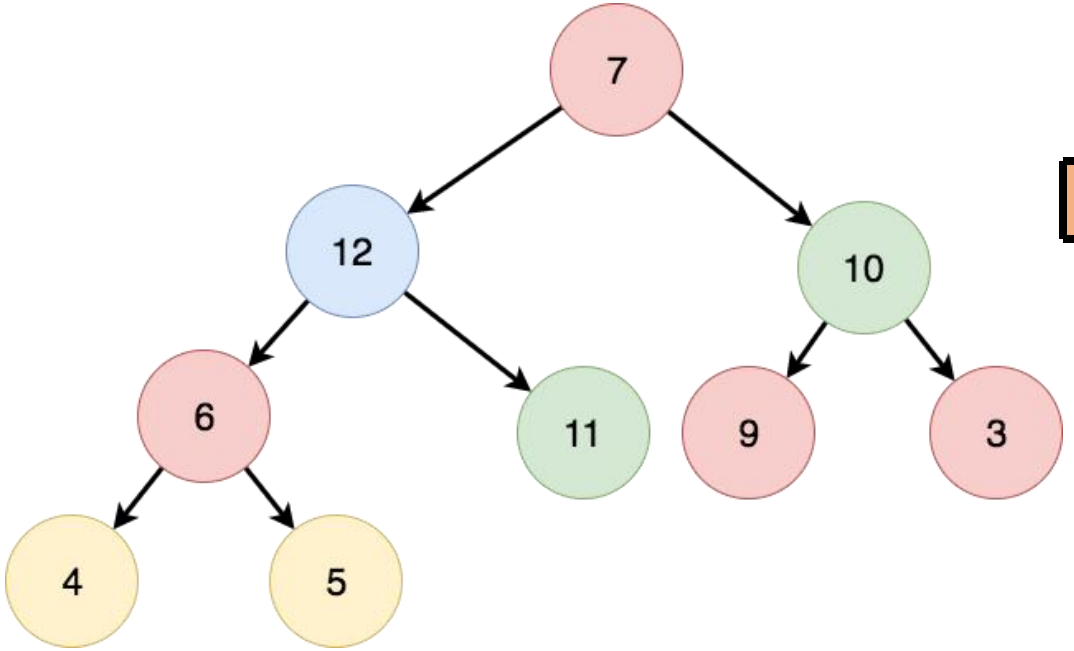
堆：头部弹出调整



大顶堆

0	1	2	3	4	5	6	7	8	9
	12	10	6	11	9	3	4	5	7

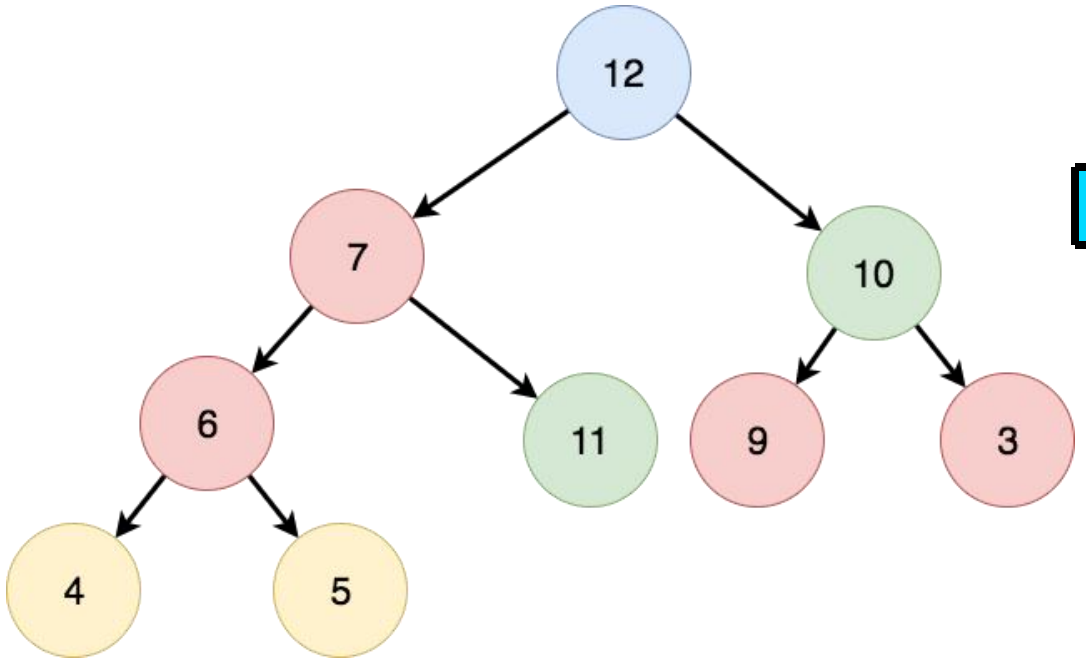
堆：头部弹出调整



大顶堆

0	1	2	3	4	5	6	7	8	9
7	12	10	6	11	9	3	4	5	

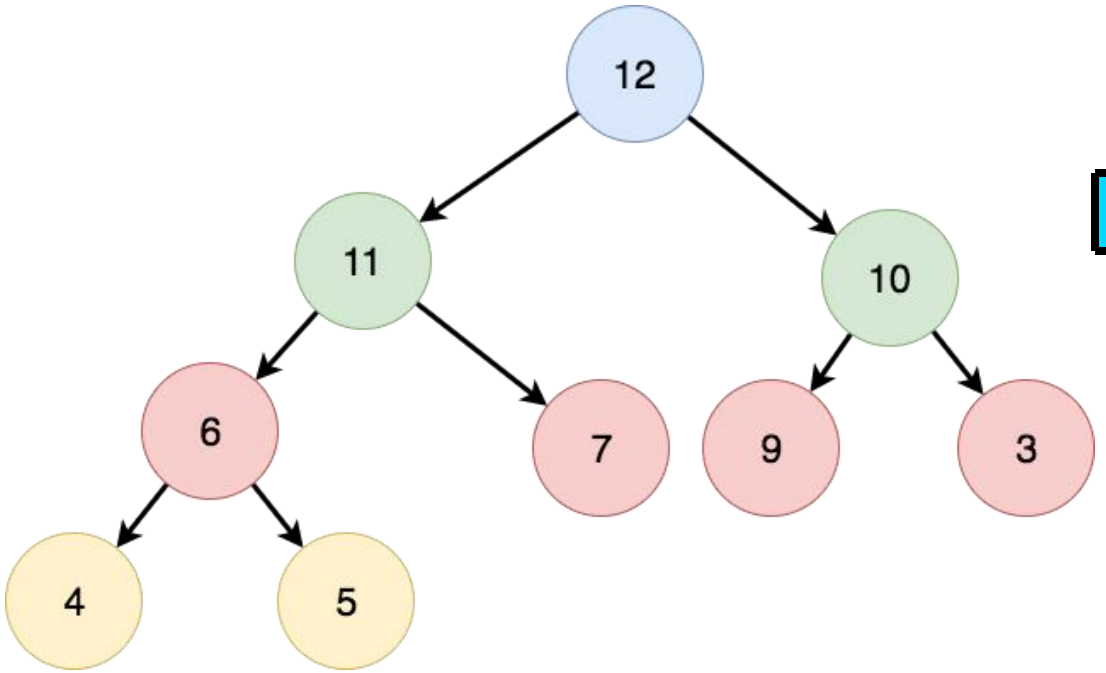
堆：头部弹出调整



大顶堆

0	1	2	3	4	5	6	7	8	9
12	7	10	6	11	9	3	4	5	

堆：头部弹出调整



大顶堆

0	1	2	3	4	5	6	7	8	9
12	11	10	6	7	9	3	4	5	

堆：优先队列

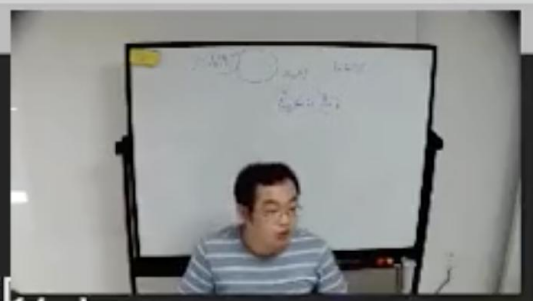
普通队列	（最大/最小）堆
尾部入队	尾部可以插入
头部出队	头部可以弹出
先进先出	每次出队权值（最大/最小的元素）
数组实现	数组实现，**逻辑上看成一个堆**

堆：优先队列

普通队列	优先队列
尾部入队	尾部可以插入
头部出队	头部可以弹出
先进先出	每次出队权值（最大/最小的元素）
数组实现	数组实现，**逻辑上看成一个堆**

1. vim

vim ⌘1 bash ⌘2 bash ⌘3



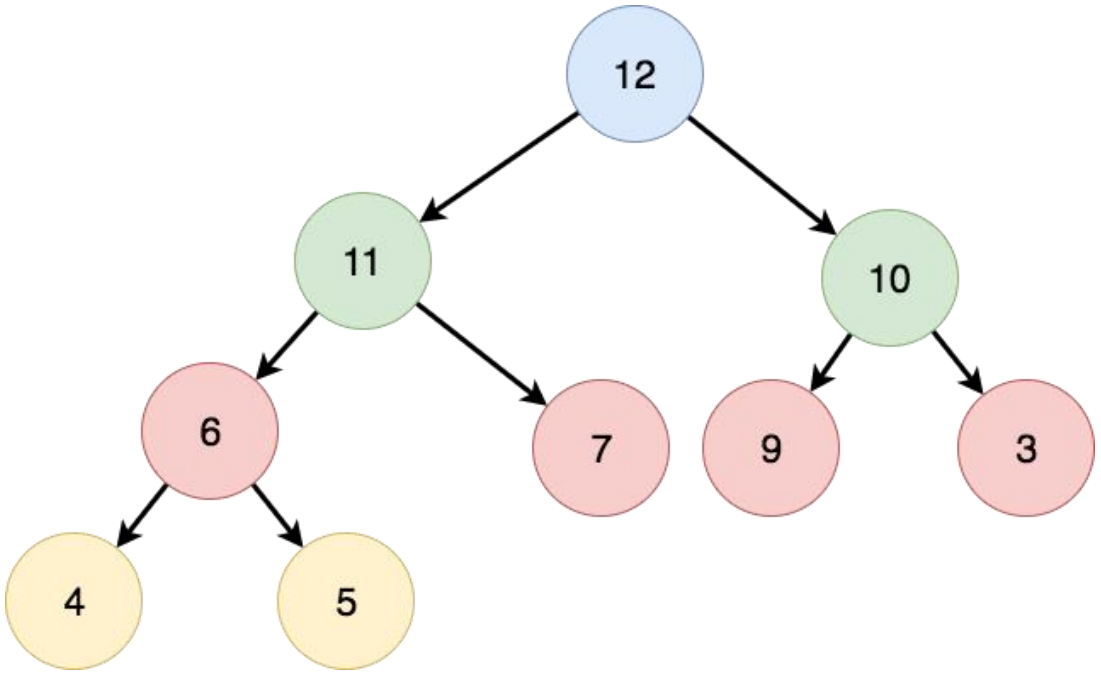
```
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED) {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52
53     } else {
54         if (!hasRedChild(root->rchild)) return root;
55
56     }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

优先队列：代码演示

<-6班资料/X.现场撸代码/15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21

二. 堆排序与线性建堆法

堆排序



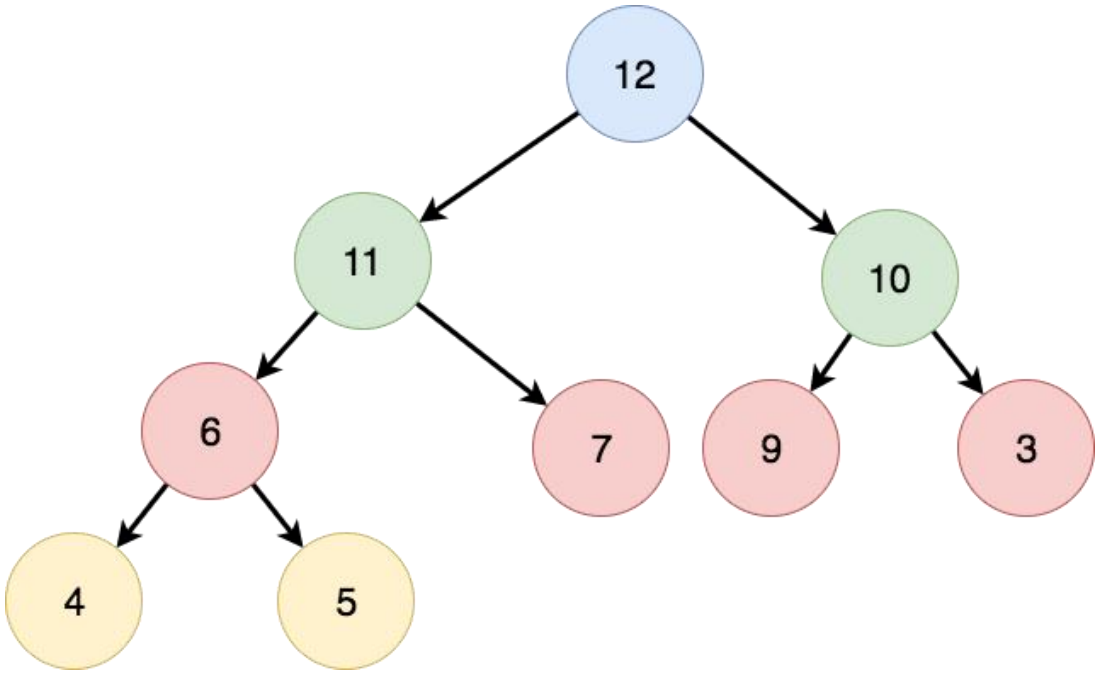
大顶堆

0	1	2	3	4	5	6	7	8
12	11	10	6	7	9	3	4	5

口诀：

1、将堆顶元素与堆尾元素交换
2、将此操作看做是堆顶元素弹出操作
3、按照头部弹出以后的策略调整堆

堆排序



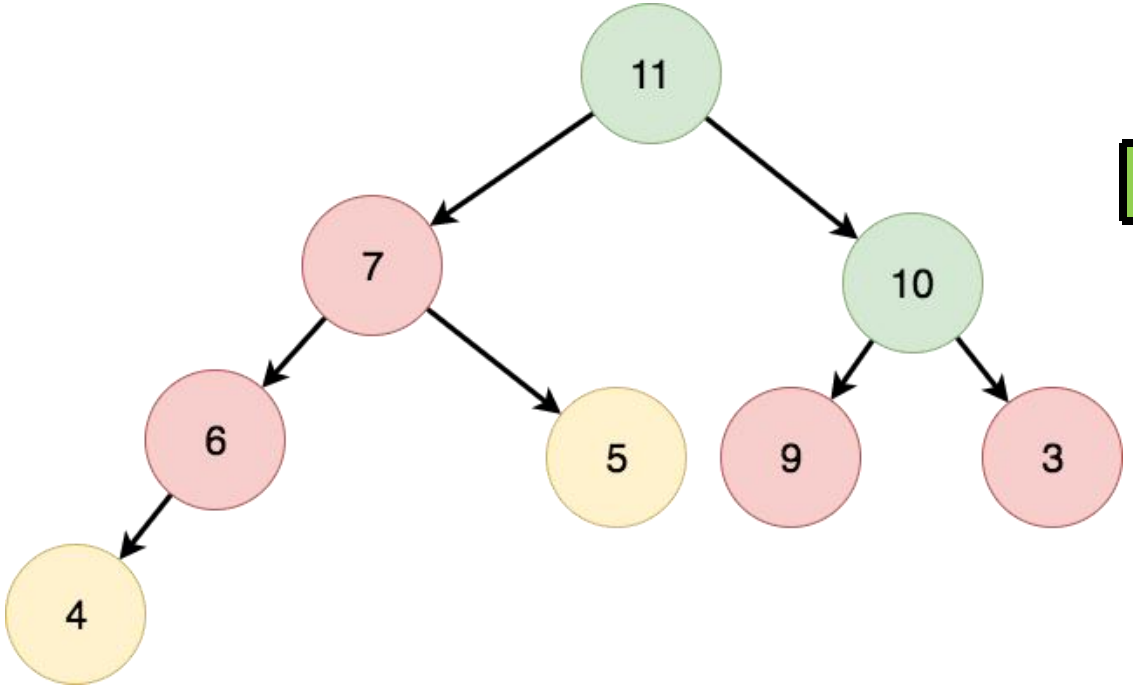
大顶堆

0	1	2	3	4	5	6	7	8
12	11	10	6	7	9	3	4	5

练习题：

1、请画出弹出一次以后的堆以及数组
2、请画出弹出三次以后的堆以及数组

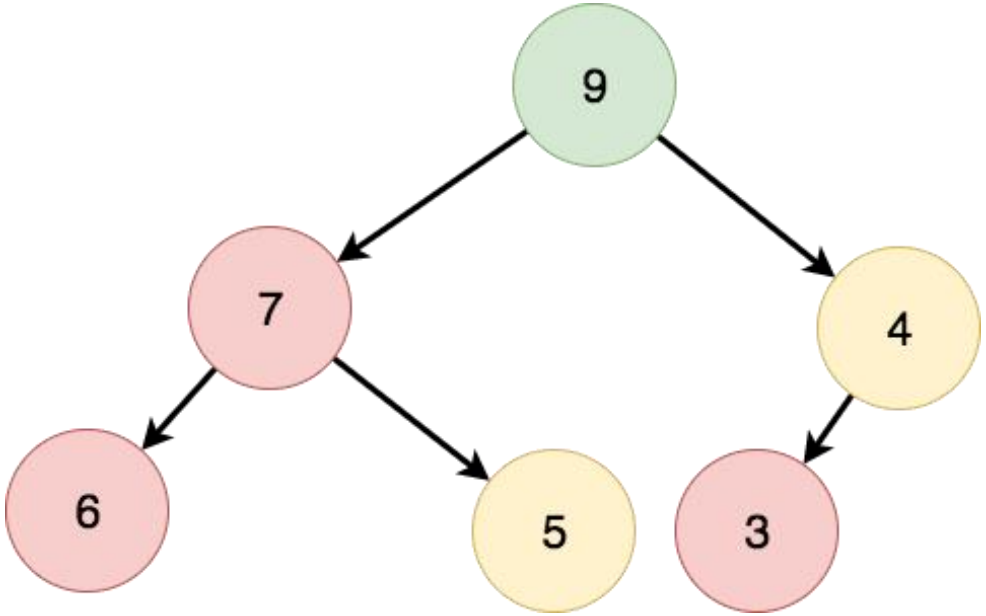
堆排序：弹一次



大顶堆

0	1	2	3	4	5	6	7	8
11	7	10	6	5	9	3	4	12

堆排序：弹三次



大顶堆

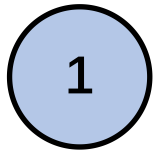
0	1	2	3	4	5	6	7	8
9	7	4	6	5	3	10	11	12

普通建堆：向上调整

0	1	2	3	4	5	6
1	2	3	4	5	6	7

大顶堆

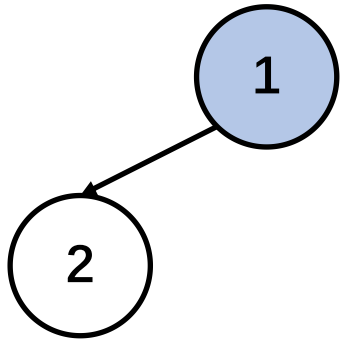
普通建堆：向上调整



0	1	2	3	4	5	6
1	2	3	4	5	6	7
0						

大顶堆

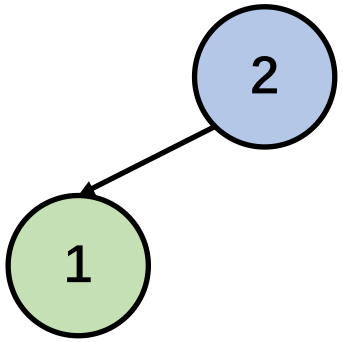
普通建堆：向上调整



0	1	2	3	4	5	6
1	2	3	4	5	6	7
0						

大顶堆

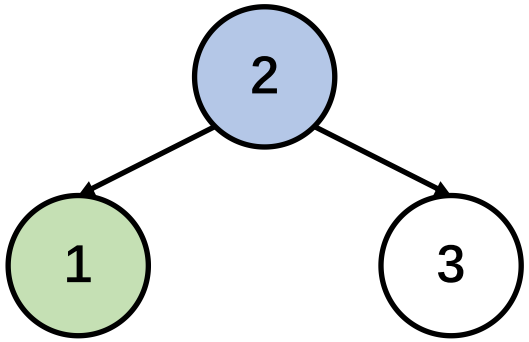
普通建堆：向上调整



0	1	2	3	4	5	6
2	1	3	4	5	6	7
0	1					

大顶堆

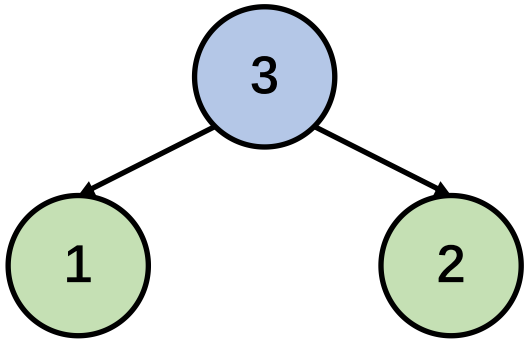
普通建堆：向上调整



0	1	2	3	4	5	6
2	1	3	4	5	6	7
0	1					

大顶堆

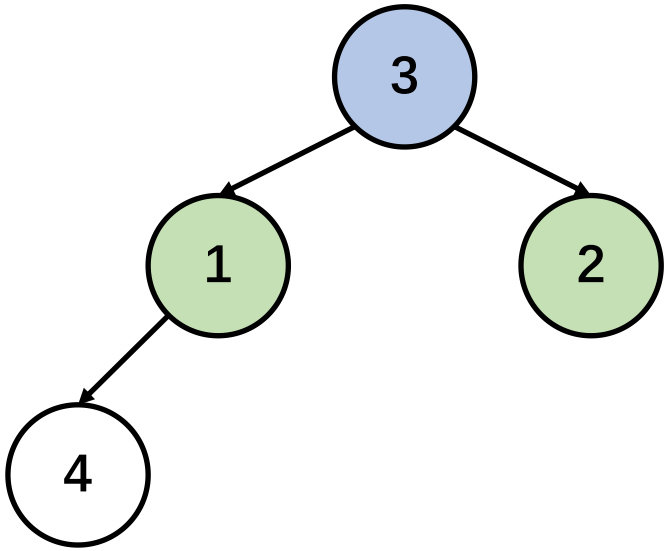
普通建堆：向上调整



0	1	2	3	4	5	6
3	1	2	4	5	6	7
0	1	1				

大顶堆

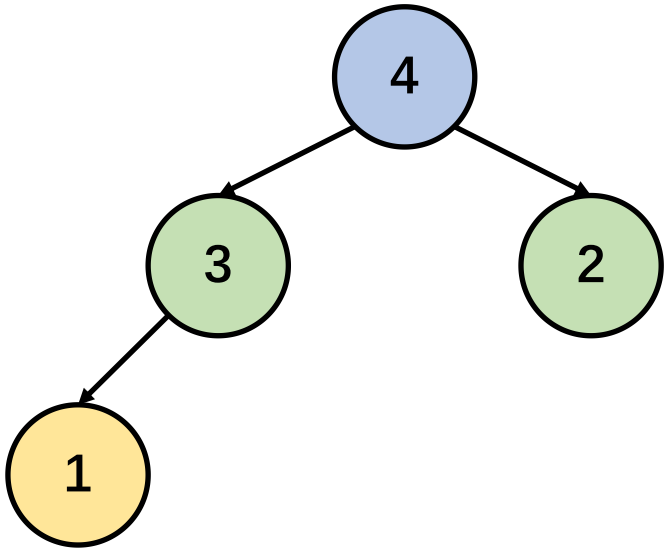
普通建堆：向上调整



0	1	2	3	4	5	6
3	1	2	4	5	6	7
0	1	1				

大顶堆

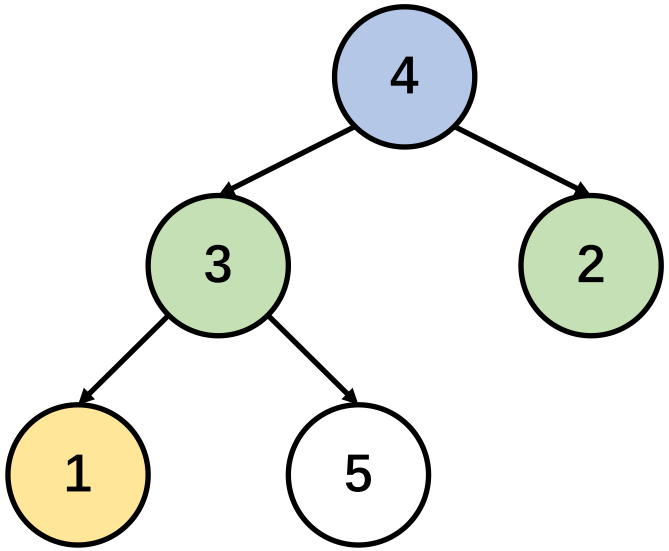
普通建堆：向上调整



0	1	2	3	4	5	6
4	3	2	1	5	6	7
0	1	1	2			

大顶堆

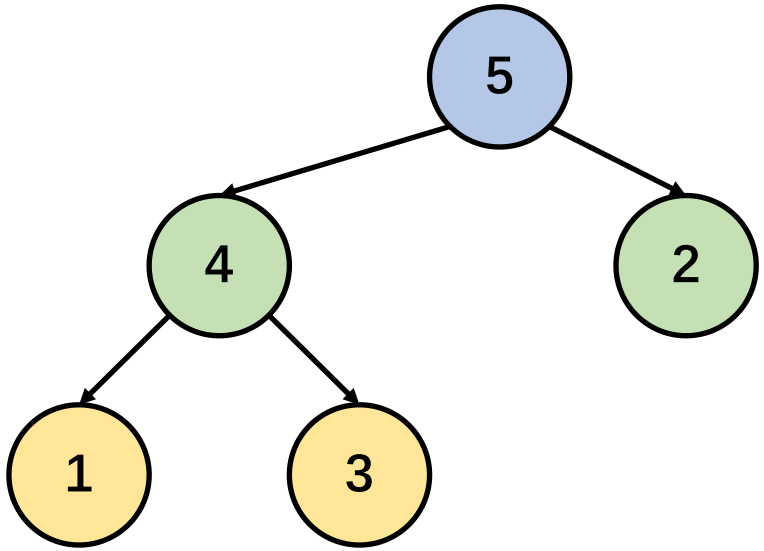
普通建堆：向上调整



0	1	2	3	4	5	6
4	3	2	1	5	6	7
0	1	1	2			

大顶堆

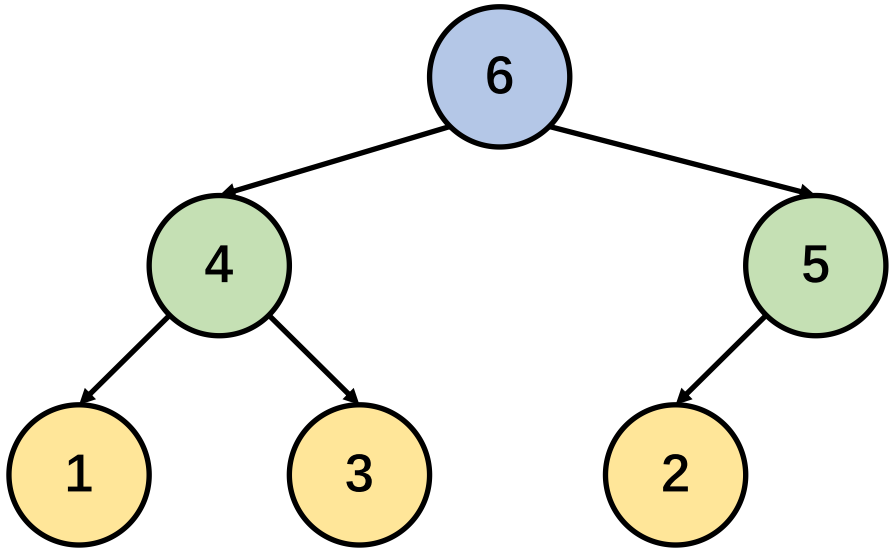
普通建堆：向上调整



0	1	2	3	4	5	6
5	4	2	1	3	6	7
0	1	1	2	2		

大顶堆

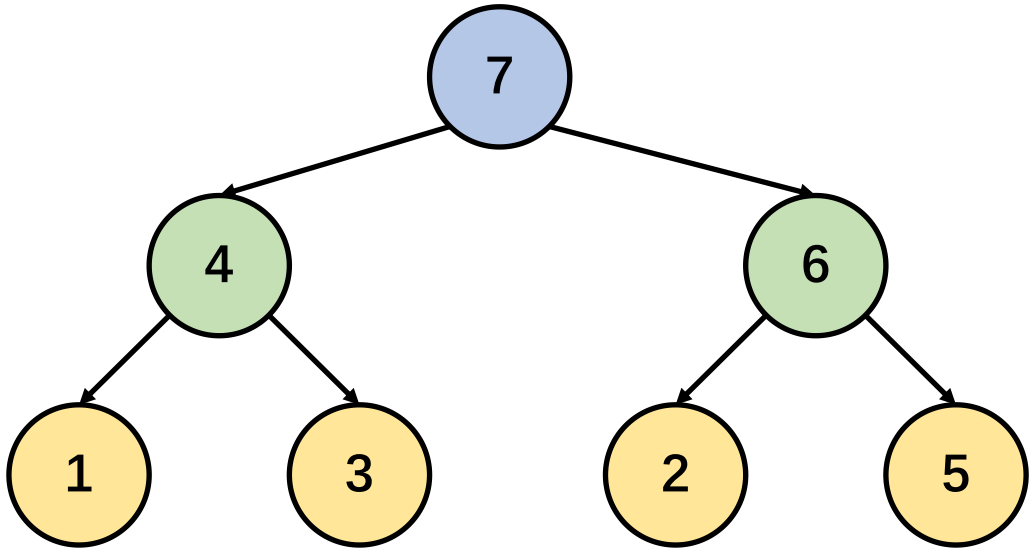
普通建堆：向上调整



0	1	2	3	4	5	6
6	4	5	1	3	2	7
0	1	1	2	2	2	

大顶堆

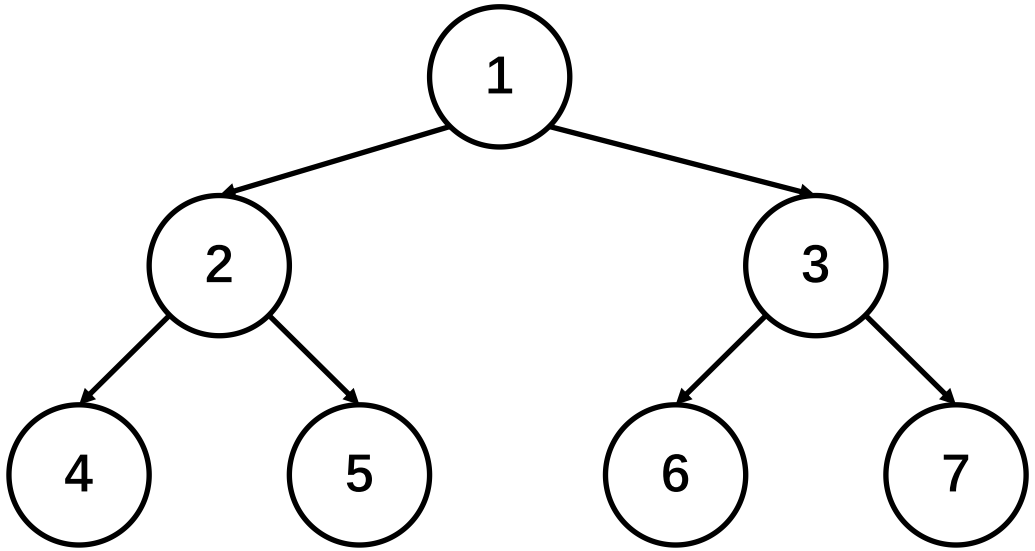
普通建堆：向上调整



0	1	2	3	4	5	6
7	4	6	1	3	2	5
0	1	1	2	2	2	2

大顶堆

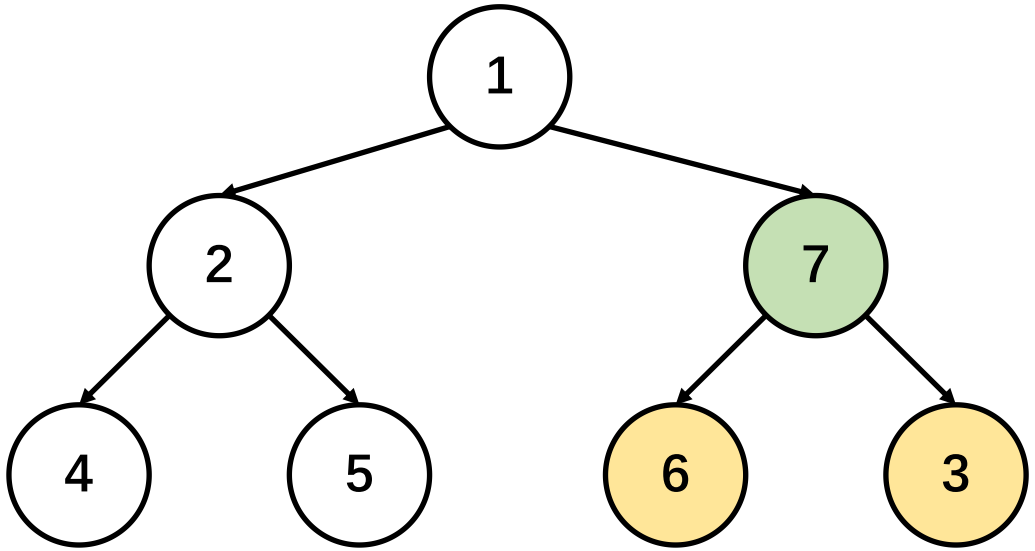
线性建堆：向下调整



0	1	2	3	4	5	6
1	2	3	4	5	6	7

大顶堆

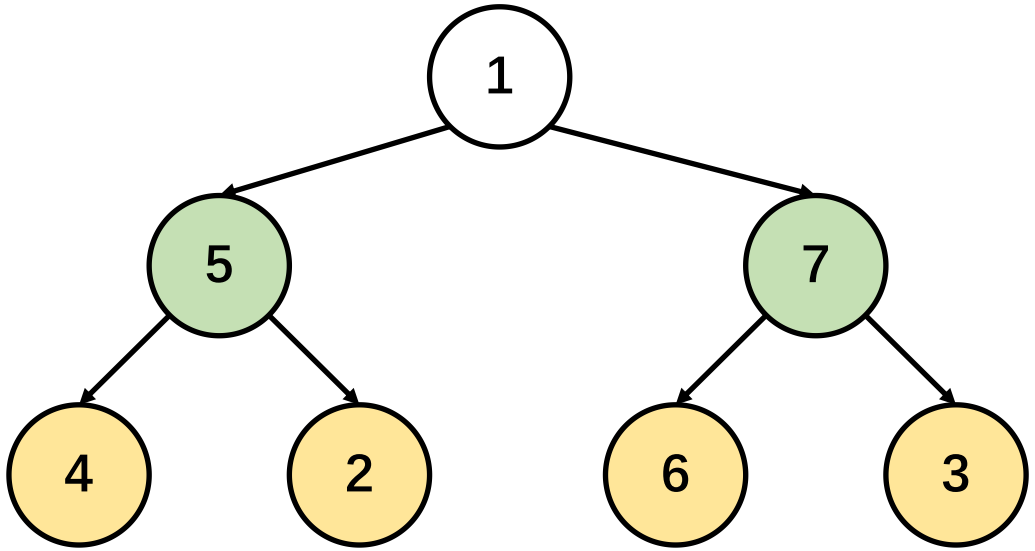
线性建堆：向下调整



0	1	2	3	4	5	6
1	2	7	4	5	6	3
		1			0	0

大顶堆

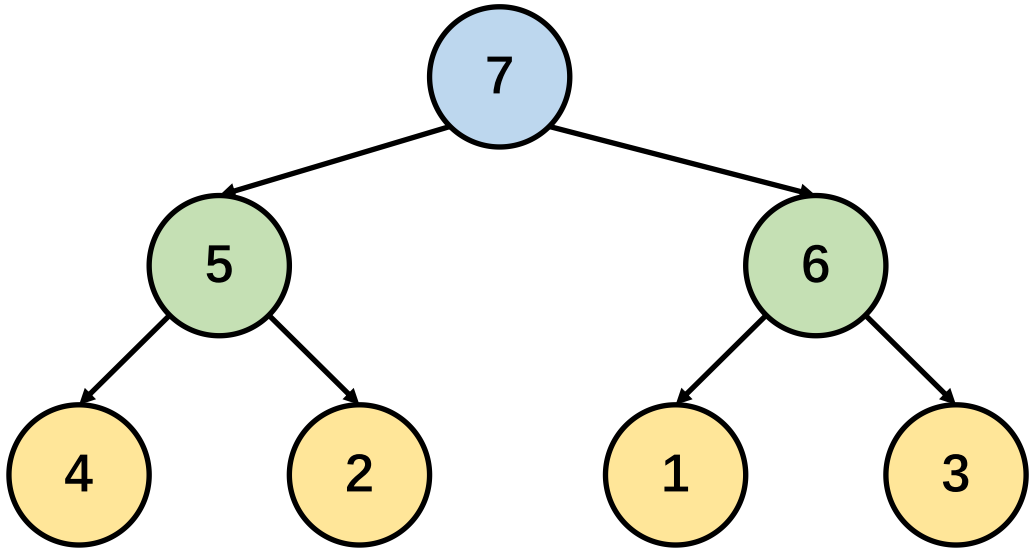
线性建堆：向下调整



0	1	2	3	4	5	6
1	5	7	4	2	6	3
	1	1	0	0	0	0

大顶堆

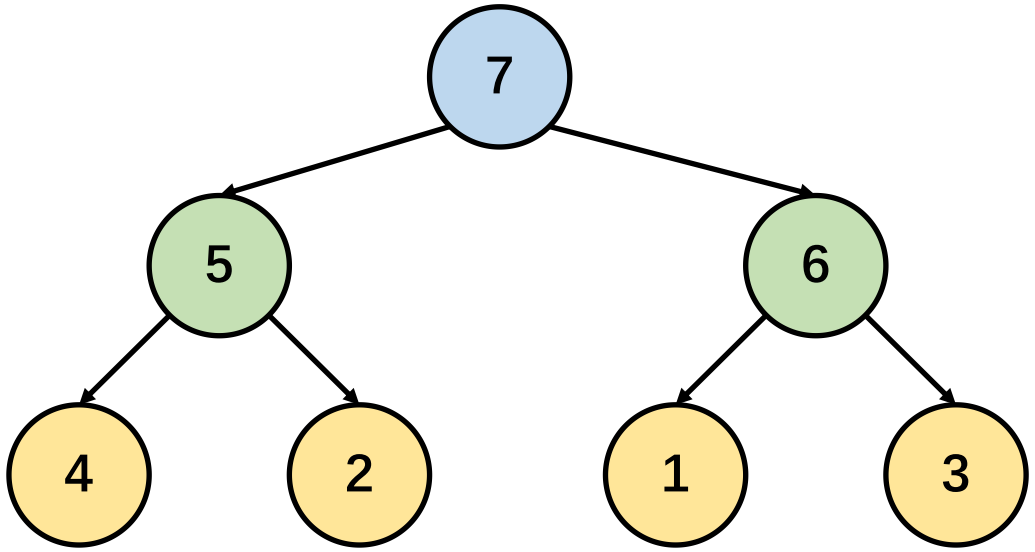
线性建堆：向下调整



0	1	2	3	4	5	6
7	5	6	4	2	1	3
2	1	1	0	0	0	0

大顶堆

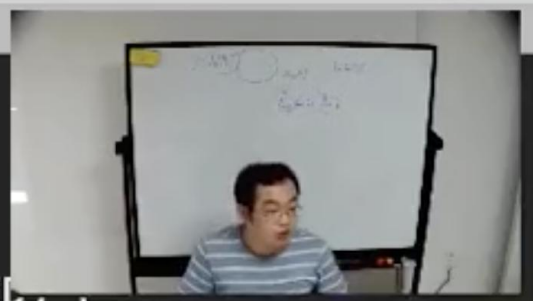
线性建堆法复杂度推导



大顶堆

1. vim

vim ⌘1 bash ⌘2 bash ⌘3



```
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED) {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52
53     } else {
54         if (!hasRedChild(root->rchild)) return root;
55
56     }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

堆排序：代码演示

<-6班资料/X.现场撸代码/15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21

三. 优化：哈夫曼编码

哈夫曼编码

哈夫曼编码生成过程 ：

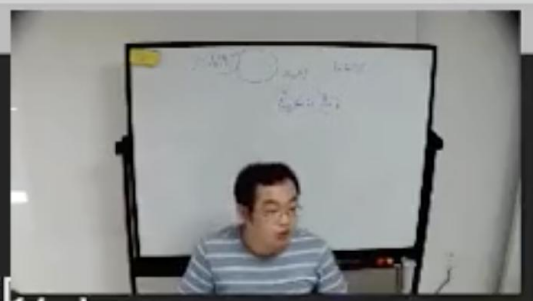
1.首先，统计得到每一种字符的概率
2.每次将最低频率的两个节点合并成一棵子树
3.经过了 n-1 轮合并，就得到了一棵哈夫曼树
4.按照左0，右1的形式，将编码读取出来

(a,0.5),(b,0.2)

(c,0.1),(d,0.2)

1. vim

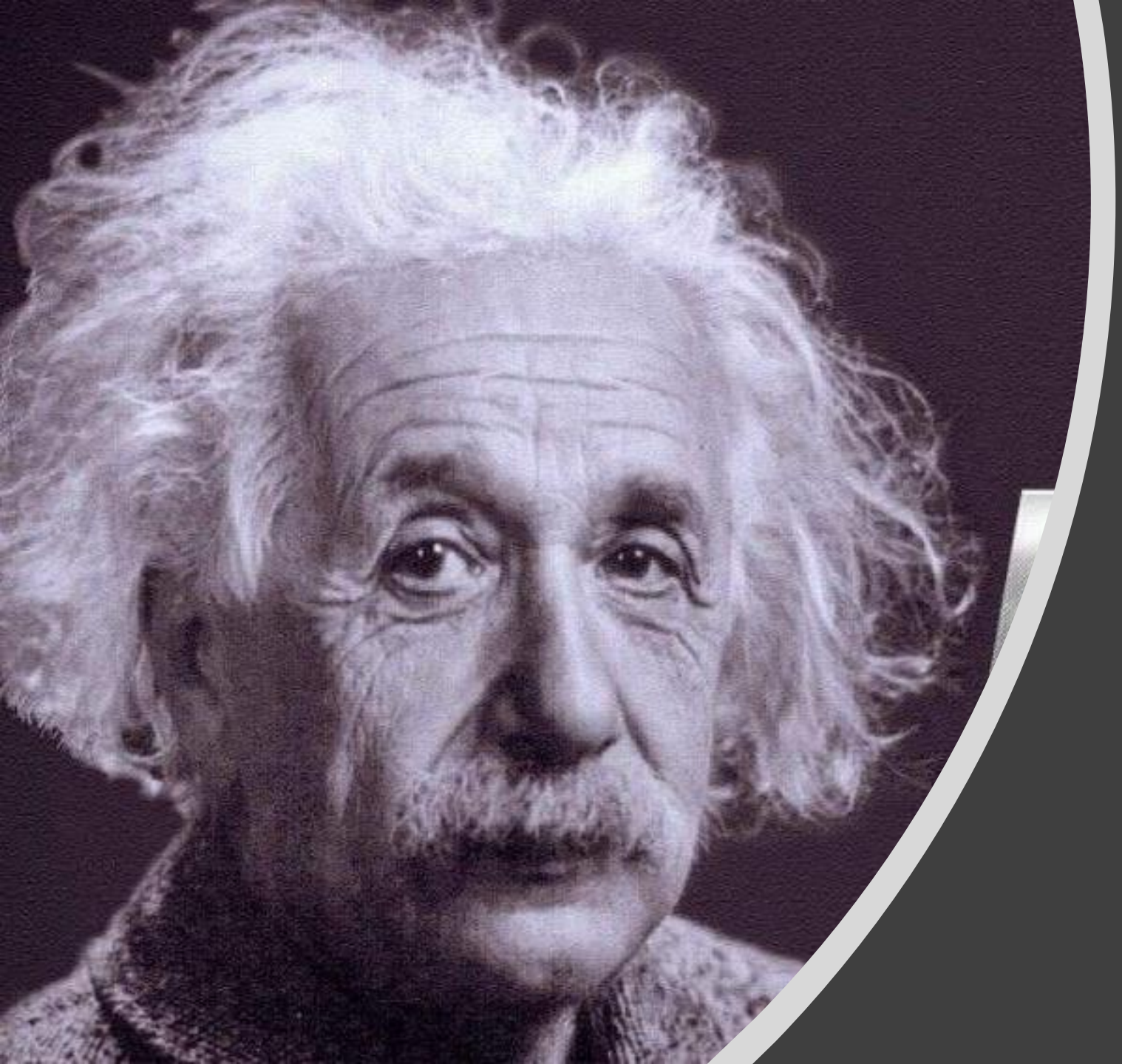
vim ⌘1 bash ⌘2 bash ⌘3



```
39 }
40
41 Node *insert_maintain(Node *root) {
42     if (!hasRedChild(root)) return root;
43     if (root->lchild->color == RED && root->rchild->color == RED) {
44         if (!hasRedChild(root->lchild) && !hasRedChild(root->rchild)) return root;
45         root->color = RED;
46         root->lchild->color = root->rchild->color = BLACK;
47         return root;
48     }
49     if (root->lchild->color == RED) {
50         if (!hasRedChild(root->lchild)) return root;
51
52
53     } else {
54         if (!hasRedChild(root->rchild)) return root;
55
56     }
57
58 }
59
60
61 Node *__insert(Node *root, int key) {
62     if (root == NIL) return getNewNode(key);
```

哈夫曼编码：代码优化

<-6班资料/X.现场撸代码/15.RBT.cpp [FORMAT=unix] [TYPE=CPP] [POS=54,30][62%] 21/09/19 - 20:21



为什么

会出一样的题目？

《船说：算法与数据结构》
第5章-堆与优先队列