红黑树的应用

the Application of Red-Black Tree

张浩宇

2020.9.9

1. 红黑树与AVL树的异同及其适用的场景

发明、节点信息、平衡方式、平衡效果、时间开销

2. 红黑树在C++中STL的使用

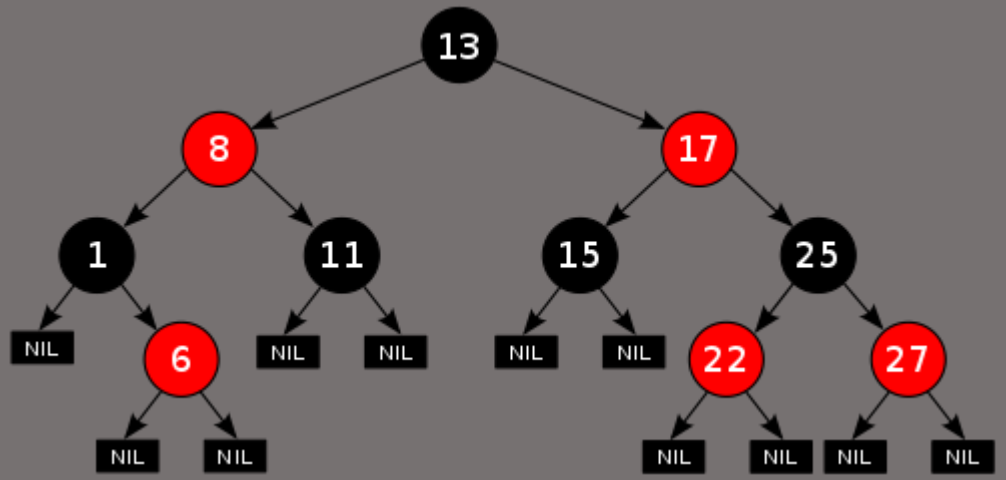
包括set、map、multiset、multimap（以set为主）的设计思路。基本用法，注意事项等

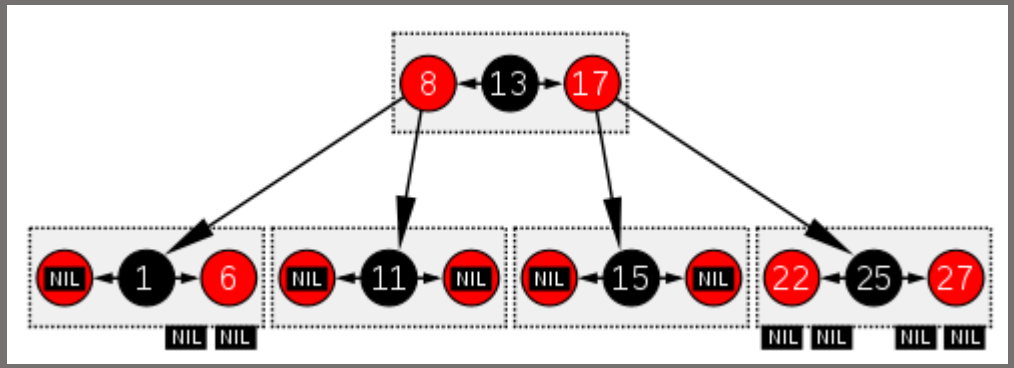
1. 红黑树与AVL树的异同及其适用的场景

1.1（发明）

红黑树：1972年 Rudolf Bayer基于B树设计的2-3-4树

1978年 Leonidas J. Guibas and Robert Sedgewick 从上述数据结构中获得了红黑树。


13
8
17
1
11
15
25
NIL
6
NIL
NIL
NIL
NIL
22
27
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL


8
13
17
NIL
1
6
NIL
11
NIL
NIL
15
NIL
22
25
27
NIL
NIL
NIL
NIL
NIL
NIL

1. 红黑树与AVL树的异同及其适用的场景

1.1（发明）

红黑树：1972年 Rudolf Bayer基于B树设计的2-3-4树

1978年 Leonidas J. Guibas and Robert Sedgewick 从上述数据结构中获得了红黑树。

AVL树：1962年 Адельсóн-Вéльский 和 Лáндис合作发明了该数据结构，以其名字命名（Adelson-Velsky 和 Landis），是最早被发明的平衡树

同：两者最终定型均为合作产物

异：红黑树发明者为美国人，AVL树发明者为苏联人

1. 红黑树与AVL树的异同及其适用的场景

1.2（节点信息）

红黑树：除二叉搜索树本身带有的parent, left-son, right-son, key 之外，额外维护 颜色 属性，只需要1bit额外空间。

AVL树：除二叉搜索树本身带有的属性外， 需要维护子树高度，需要占用的额外空间取决于可能最大高度。

同：两者均为子平衡的二叉搜索树，且需要在节点中维护额外信息

异：额外信息不同，至少需要的空间也不同。

1. 红黑树与AVL树的异同及其适用的场景

1.3（平衡方式）

红黑树：

1.红色节点儿子必须是黑色

2.任意一条根到叶子的路径上黑色节点数量一致

AVL树：任意一个节点，其左右子树的高度之差（即最深的叶子节点深度之差）为+1, 0 或 -1

同：两者均为高度平衡，而非重量平衡

异：AVL树直接对高度做限制，而红黑树用“黑高”来限制树的高度

1. 红黑树与AVL树的异同及其适用的场景

1.4（平衡效果）

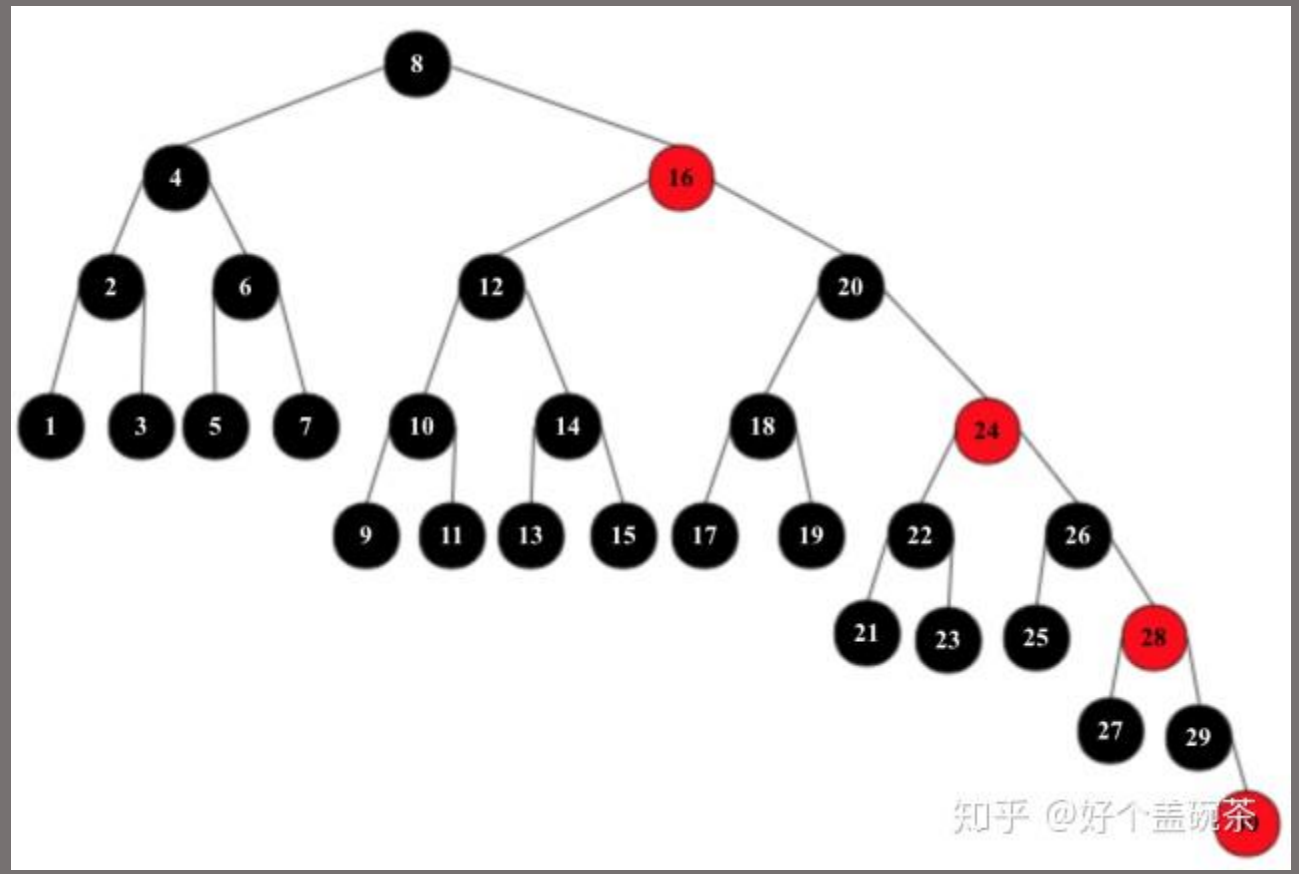
红黑树：

黑高为 b 的红黑树内部节点数为 n ，假设 n >= 2^b-1 ，利用数学归纳法：

当 b = 0 时，有 n = 0 >= 2^b-1 = 2^0-1 = 0 成立；当 b > 0 时，考虑根节点最少也要拥有两棵黑高为 b-1 的子树，所以它的节点数必然满足关系 n >= 1 + 2^(b-1)-1 + 2^(b-1)-1 = 2^b-1 。

由于红节点的孩子必须是黑节点，故而得到红黑树的高度上界：

$$h \leq 2\lfloor \log(n+1) \rfloor \approx 2\log(n)$$


8
4
16
2
6
12
20
1
3
5
7
10
14
18
24
9
11
13
15
17
19
22
26
21
23
25
28
27
29
知乎 @好个盖碗茶

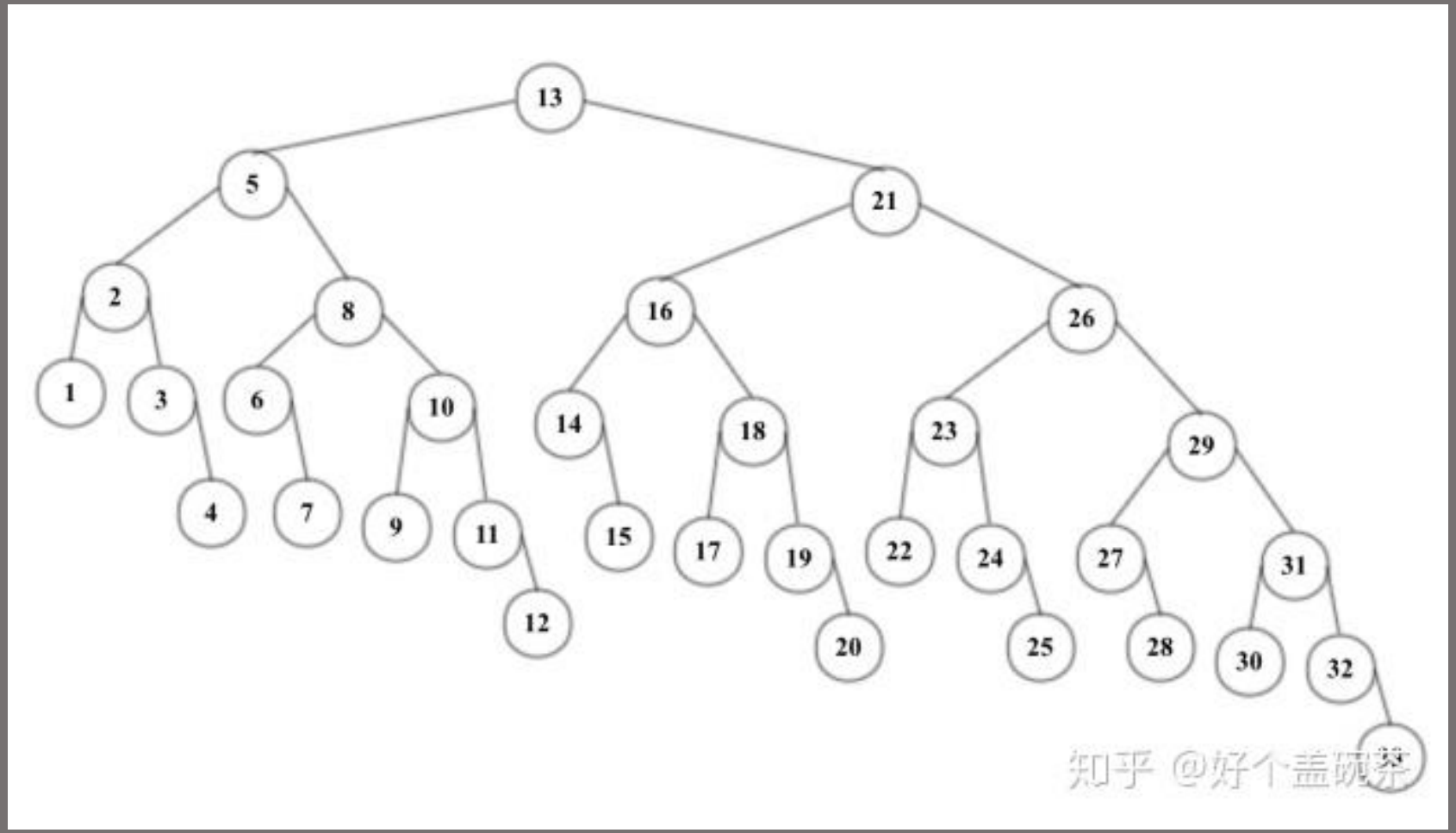
1. 红黑树与AVL树的异同及其适用的场景

1.4（平衡效果）

AVL树：假设高度为 h 的AVL树最少包含 f(h) 个内部节点，因为AVL树本身的定义决定了左右子树高度差不超过1，所以容易得出关系 f(h) = 1 + f(h-1) + f(h-2)，再结合初始条件 f(0) = 0, f(1) = 1 可以反解出AVL树高度的上界：

$$h \leq f^{-1}(n) \approx 1.44\log(n)$$

（f(h)的递推关系可用矩阵乘法表示，由于转移矩阵是满秩的，故矩阵可以对角化，然后计算出特征向量，构成矩阵并求其逆矩阵，之后就能得到f(h)的封闭形式，随后求反函数）


13
5
21
2
8
16
26
1
3
6
10
14
18
23
29
4
7
9
11
15
17
19
22
24
27
31
12
20
25
28
30
32

1. 红黑树与AVL树的异同及其适用的场景

1.4（平衡效果）

红黑树： 2log(n)

AVL树： 1.44log(n)

同：两者均为对数级，且常数较小

异：AVL树的常数更加优秀

1. 红黑树与AVL树的异同及其适用的场景

1.5 时间开销

操作类型	查找次数	调整次数	变色次数	旋转次数
AVL插入	$1.44\log(n)$	$1.44\log(n)$	$1.44\log(n)$	2
R-B插入	$2\log(n)$	$2\log(n)$	$3\log(n)$	2
AVL删除	$1.44\log(n)$	$1.44\log(n)$	$1.44\log(n)$	$0.72\log(n)$
R-B删除	$2\log(n)$	$\log(n)$	$\log(n)$	3

调整开销。插入和删除都有可能打破原来的平衡约束，因此需要一层一层地向上调整。

a. 变色开销，即修改节点的颜色(或者平衡因子)带来的开销；

b. 旋转开销，即旋转操作中修改各种指针指向带来的开销。

1. 红黑树与AVL树的异同及其适用的场景

1.5时间开销（除查找的 均摊复杂度）

操作类型	查找次数	调整次数	变色次数	旋转次数
AVL插入	-	O(1)	O(1)	O(1)
R-B插入	-	O(1)	O(1)	O(1)
AVL删除	-	O(1)	O(1)	O(1)
R-B删除	-	O(1)	O(1)	O(1)

以任意序列连续插入 n 次的最坏平均开销和以任意序列连续删除 n 次的最坏平均开销。

其中AVL树还有一个更强的上界，插入后的均摊调整次数不大于3.618，删除后的均摊调整次数不大于2.618。

1. 红黑树与AVL树的异同及其适用的场景

1.5 均摊时间开销（除查找的 均摊复杂度）

任意序列的插入或者删除混合进行：

红黑树：均摊复杂度依然保持在 O(1) 。

AVL树：均摊复杂度退化到 O(log(n))

1. 红黑树与AVL树的异同及其适用的场景

1.6应用场景

(1)空间

红黑树：额外空间更小，可以将其压缩入其他变量 -> 使得其无额外空间

AVL树：需要的额外空间取决于可能的最大树高，如3*10^6插入时可能需要5bit，10^7时可能需要6bit. 不一定能够压缩入其他变量 -> 有额外空间消耗

(2)缓存及分支预测

红黑树：能够写成2-3-4树的形式，即B树的一种特例，在每个节点维护更多的信息从而加快运行

AVL树：单个节点只能存储单个信息，对缓存的命中并不友好

1. 红黑树与AVL树的异同及其适用的场景

1.6应用场景

(3)时间开销

红黑树：在插入和删除较为频繁的情况下较为优秀

AVL树：在查找较为频繁的情况下更为优秀

AVL树与红黑树的性能表现差距较小，没有哪个能完胜另一方

虽然在工业界使用红黑树更为常见，但也有不少软件使用AVL树且并没有因此而有问题。

2. 红黑树在C++中STL的使用

(1)简单介绍

set 、 map、 multiset、 multimap

set: 维护有序的非重集合->元素需要构成全序关系，且不保存多个等价元素。 在c++ 中 使用红黑树实现。

map: 几乎和数学的“映射”一致，相当于在set的每个元素（函数的自变量）之后挂一个对应的函数的因变量。只不过允许修改，可以看作任意类型坐标的数组。与set同样，等价的自变量仅能出现一次，多次赋值会修改因变量。对于等价不相等的因变量，以首次出现的因变量为准。

multiset: 可以保存多个等价（甚至相等）元素，但对于删除操作，若以元素为参数，则会删除全部等价元素，若以迭代器为参数，则仅会删除该元素。

multimap: 等价的自变量可对应多个因变量，但无法使用[]运算符。

2. 红黑树在C++中STL的使用

(2)设计思路

均为利用红黑树维护动态有序序列，只不过set维护若干个元素，map维护若干对元素，multi前缀表示可以有重复元素。

之所以要用平衡树，因为要支持快速的插入、删除、取首元素、取尾元素，取前驱后继。

(3) 使用介绍（以set为主）

声明：

```
44 struct P {
45     int a, b;
46     bool operator < (const P &dst) const {
47         return a < dst.a;
48     }
49 }p;
50 set<int> tree;
51 map<P, string> mp;
```

2. 红黑树在C++中STL的使用

```
tree.clear();
tree.insert(15);
tree.insert(17);
tree.insert(15);
cout << tree.size() << endl;
set<int>::iterator it1, it2;
for (it1 = tree.begin(); it1 != tree.end(); it1++) {
    cout << *it1 << endl;
}
/*
for (it1 = tree.begin(); it1 != tree.end(); it1++) {
    cout << *it1 << endl;
    it2 = it1++;
    tree.erase(it2);
}
*/
/*
for (it1 = tree.begin(); it1 != tree.end(); it1++) {
    cout << *it1 << endl;
    tree.erase(it1);
}
*/
```

```
2
15
17
15
17
0
段错误 (核心已转储)
```

2. 红黑树在C++中STL的使用

```
tree.clear();
tree.insert(15);
tree.insert(17);
tree.insert(15);

set<int>::iterator it1, it2;

it1 = tree.find(15);
it2 = tree.find(16);
cout << (it1 == tree.end()) << " " << (it2 == tree.end());
return;
```

```
0 1
```

2. 红黑树在C++中STL的使用

```
map<P, string> mp;
map<P, string, less<P> > mp_;

p.a = 10; p.b = 1;
map<P, string>::iterator itm;
string tmp = "1453";
mp[p] = tmp;
for (int i = 0; i < 17; i++) {
    tmp[2] = '1' + (char) (i - 10);
    p.b = i;
    mp[p] = tmp;
}
itm = mp.begin();
cout << itm->first.b << endl;
cout << itm->second << endl;
p.a = -1;
cout << mp[p] << endl;
cout << mp.size();


return;
```

```
gbakkk5951@gbakkk5951-Precision-3530:~$ ./std
1
1473

2gbakkk5951@gbakkk5951-Precision-3530:~$
```

2. 红黑树在C++中STL的使用

(4)应用

(4.1) 按照数学概念使用

例1.维护一个可重元素的集合，支持多次插入1个元素和删除1个元素。

(4.2) set作为优先队列使用

例1.对于队列中的人按照优先级排序，若干次取出最优先/加入一个人

(4.3) map作为任意类型下标数组使用

例1.以哈希值为下标存储对应的原数据

例2.以姓名查找/修改学号

感谢马骏老师提供的帮助和建议

参考资料

https://blog.csdn.net/qq_25343557/article/details/89110319

https://zhuanlan.zhihu.com/p/93369069

https://en.wikipedia.org/wiki/AVL_tree

https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

《算法导论》（第三版）第13章 红黑树

谢谢大家！