STL: C++ 标准模板----通俗来说:对常见数据结构的封装+增加一些通用类型的算法
通用:算法可以操作任意类型的数据+与具体的数据结构无关

六大组件:
容器:就是用来装数据的---每个容器实际对应了某种特定的数据结构
分类:序列式容器和关联式容器
序列式容器: --线性结构
string/vector/list
deque
array/forward_list
关联式容器:红黑树+哈希结构
红黑树结构关联式容器: map/set/multimap/multiset
哈希结构关联式容器: unordered_map/set/multimap/multiset

string/vector/list
1.熟悉常用接口
2.底层结构必须熟悉
3.涉及到的在线OJ
4.应用场景

vector和list的不同--顺序表和链表):
vector和list都是STL提供通用类型的序列式容器
底层：vector是一段连续空间，list是链式结构(带头结点的双向循环链表)
元素访问:vector支持随机访问--- >0(1)；list不支持随机访问-->访问任意位置的元素时必须要遍历-->0(N)
插入删除：vector任意位需插入或者删除元素，需要搬移大量的元素，效率，比较低--->0(N)；在插入时，可能需要扩容:1.申请新空间2.拷贝元素3.释放旧空间；因此:在插入时比如push_back如果没有提前将容量给出，效率更低
list任意位置插入/删除元素--->只需要改变指针的指向，效率: 0(1)；插入不需要扩容
使用场累:vector高效存储+频繁的访问；list在任意位置插入/删除操作比较多
迭代器:vector迭代路- - SGI-迭代器的类型实际就是原生态指针的别名，因为原生态的指针可以去遍历到空间中的每个元素；list实际就是对原生态指针进行了封装--可以将该迭代器按照指针的方式类进行使用---方便遍历
迭代器失效: vector只要底层空间发生改变目前所定义的迭代器都会失效；
push_back/insert/resize/reserve等；it = erase(it)

list：erase(it)
接口不同:vector支持随机访---operator[]，与容量相关的接口；list：有一些特殊接口: merge、sort...
接口不同:支持随机访---operator[]
与容量相关的接口
string--1. string中的常用接口2. 刷题3. 深浅拷贝--传统版||简洁版- >写时拷贝
vector--1.熟忠vecotr中的接口2. 熟悉vectorr容机制3. 刷题
list---> 1. list中的常用接口2. 不带头结点的单随表中基本操作(注意:在面试期间，面试官让写链表相关的代码，如果没有特殊说明，一般按照不带头结点的单链---问一下)
3.双向链表反复去写-->任意位置插入和删除
4.链表先关的面试题
deque:了解---将《STL》源码剖析--deque

关联式容器:
树形结构的关联式容器---->底层结构都是红黑树----->1.用迭代器遍历的结果一定是关于key有序的
2.查找元素的效率--->log(N)

map: key-value, key必须是唯一 的
multimap: key-value, key是可以重复的
set: key--key必须是唯一---->可以去重
multiset: key--key是可以重复---相当于排序

底层数据结构:
1.二叉例中的基本操作:创建、拷贝、删除，求高度，求叶子，求K层叶子、查找
遍历:按照某种规则对二叉树中的每个节点进行相应的操作(打印，节点的值+1),并且每个节点只操作一次；前序、中序，后序遍历(递归遍历/非递归)层序遍历
在线OJ:根据前序中序，中序后序还原二叉树，最近公共祖先
2.二叉搜索树
二叉搜索树概念：二叉树搜索树特性:中序遍历结果是有序　最左侧和最右侧节点一定是最小或者最大的节点
二叉树搜索树的实现:插入、查找、删除(复杂)
二叉搜索树的极端情况:如果插入的序列是有序的或者接近有序----单支树-->在其上进行查找时间复杂度就是O(N)
二叉搜索树查找的效率是O(N)

3. AVL树----概念:二叉树搜索树+平衡因子的限制---->每个节点左右子树高度差的绝对值不超过1===>平衡二叉搜索树，AVL树的查找效率: O(logN)

a.对于AVL树中的旋转一定要然----面试期间比较喜欢考

b. AVL树的插入过程

4.红黑树

概念:二叉搜索树+给节点增加颜色&节点进行约定===>最长路径中节点个数一定不会超过最短路径中节点个数的两倍，近似平衡的二叉搜索树，虽然说近似平衡，大量的使用结果都表明-->性能还是非常好的

约定/性质

a.每个节点都有颜色---不是红色就是黑色

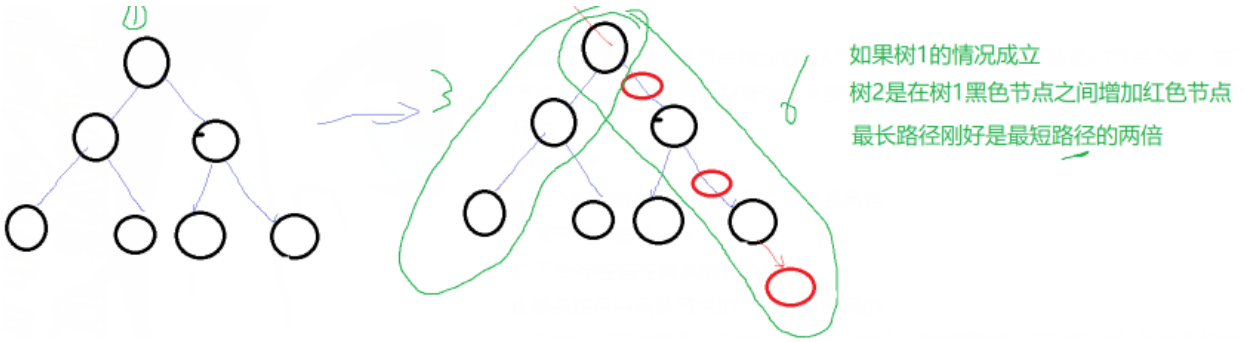
b.根节点一定是黑色

C.不会存在连在有其的红色节点

d.每条路径中黑色节点的个数都是相同的

e.所有叶子节点都是黑色的(注意:此处的叶子与常规的叶子节点不同--指的是树中的空指针域)

最终就可以保证:最长路径中节点个数一定不会超过最短路径中节点个数的两倍---近似平衡的二叉搜索树，查找效率: O(logN)



但是图一是不可能存在---因此图2也一定不会存在

红黑树插入代码的实现

1.先按照二叉搜索树的方式，将cur新节点插入

2.检测cur的双亲parent是否为红色----给节点的默认颜色红色,parent如果是红色的，才需要进行更新

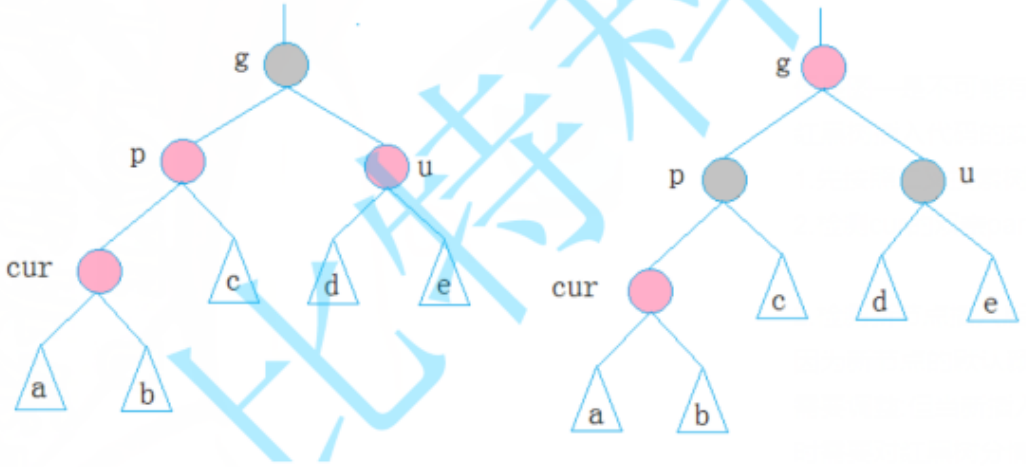
2.检测新节点插入后，红黑树的性质是否造到破坏

因为新节点的默认颜色是红色，因此:如果其双亲节点的颜色是黑色，没有违反红黑树任何性质，则不需要调整;但当新插入节点的双亲节点颜色为红色时，就违反了性质三不能有连在一起的红色节点，此时需要对红黑树分情况来讨论:

约定:cur为当前节点，p为父节点，g为祖父节点，u为叔叔节点

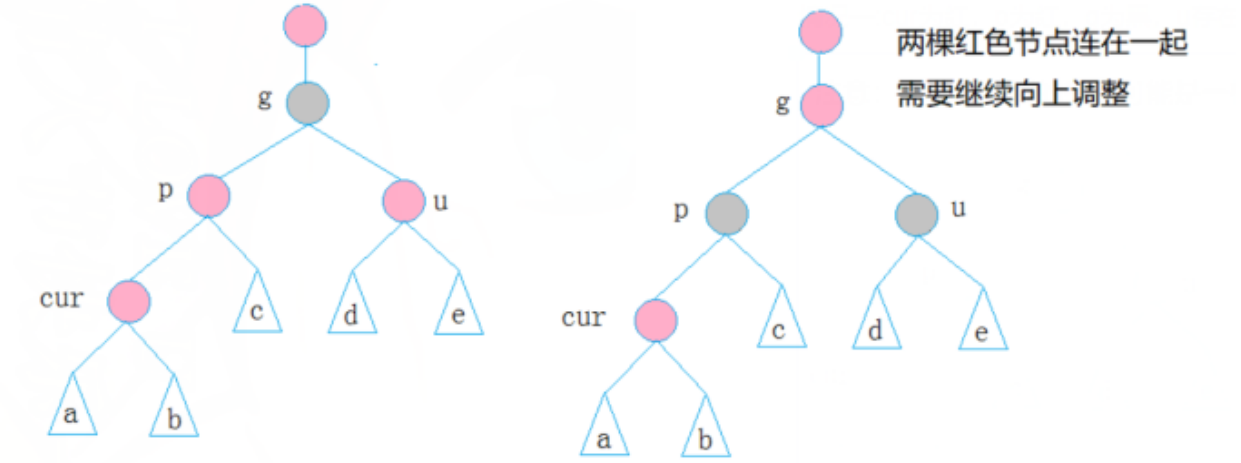
情况一:cur为红，p为红，g为黑，u存在且为红

注意：此处所看到的树，可能是一棵完整的树，也可能是一棵子树



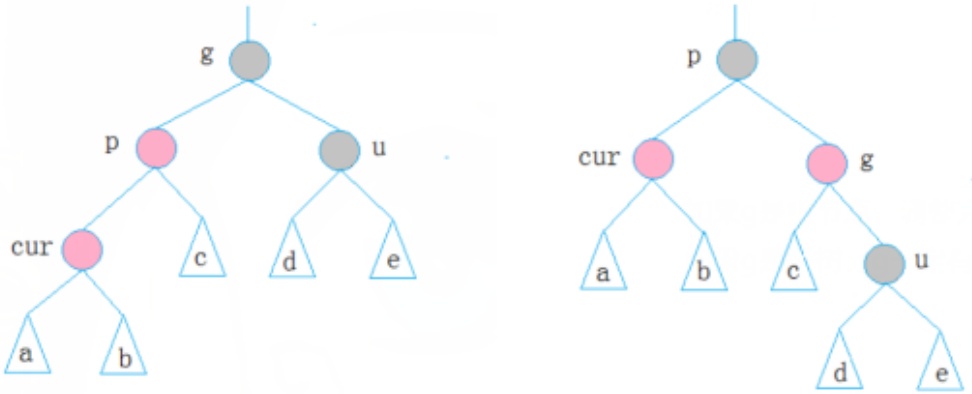
如果g是根节点，调整完成后，需要将g改为黑色

如果g是子树，g一定有双亲，且g的双亲如果是红色，需要继续向上调整



cur和p均为红，违反了性质三，此处能否将p直接改为黑？

- 情况二: **cur为红，p为红，g为黑，u不存在/u为黑**



说明：u的情况有两种

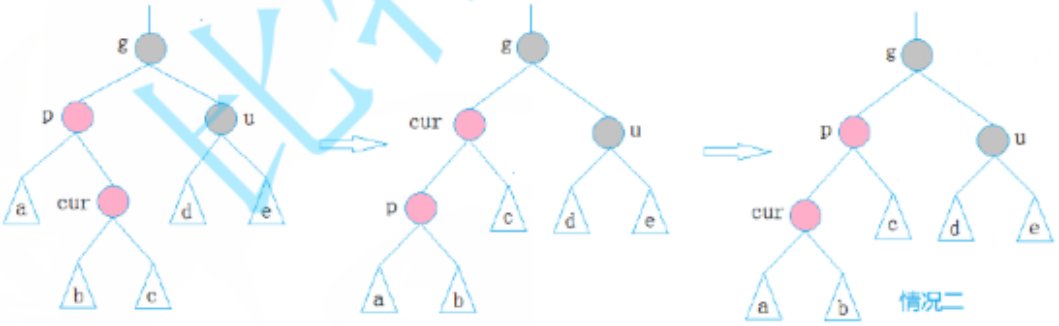
1. 如果u节点不存在，则cur一定是新插入节点，因为如果cur不是新插入节点，则cur和p一定有一个节点的颜色是黑色，就不满足性质4：每条路径黑色节点个数相同。
2. 如果u节点存在，则其一定是黑色的，那么cur节点原来的颜色一定是黑色的，现在看到其是红色的原因是因为cur的子树在调整的过程中将cur节点的颜色由黑色改成红色。

p为g的左孩子，cur为p的左孩子，则进行右单旋转；相反，

p为g的右孩子，cur为p的右孩子，则进行左单旋转

p、g变色--p变黑，g变红

- 情况三: **cur为红，p为红，g为黑，u不存在/u为黑**



p为g的左孩子，cur为p的右孩子，则针对p做左单旋转；相反，

p为g的右孩子，cur为p的左孩子，则针对p做右单旋转

则转换成了情况2

红黑树的应用:
哈希结构的关联式容器:
unordered_ map/set/multimap/multiset区别 和树形结构的区别
要熟悉其使用

到底选择树形结构还是选择哈希结构:
1.如果需要有序--树形结构关联式容器
2如果对有序不关心，高的查询效率----统计ip次数

哈希:
通过某种方式，将元素与其在表格中的存储位置之间建立一一对应的关系
比如插入:
1.先通过哈希函数计算元素在表格中的位置
2.插入元素
比如查找:
1.先通过哈希函数计算元素在表格中的位置
2.检测是否为待查找的元素
因为哈希的方式:可以直接通过哈希定位到元素在表格中的存储位置，因此在查找时不需要进行遍--->哈希效率: 0(1)
哈希冲突(碰撞):不同元素通过相同的哈希函数计算出相同的哈希地址
哈希冲突的解决方式:
1.直接定址法
2.除留余数法
了解:叠加法、平方取中法、随机数法、数学分析法
几个常见的哈希函数:
检查哈希函数的设计是否合理，如果不合理(设计不好)--可能会使冲突概率增大
重新设计哈希函数，考虑哪些因素:
a.哈希函数设计应该尽可能简单
b.哈希函数计算的哈希地址要尽可能的均匀---例子:到教室去上课，假设都集中坐在某个范围中
C.哈希函数的值域哈希地址必须要在哈希表格的范图之内----例子:让你坐在教室外听课

注意:不论哈希函数设计的有多精妙,都不可能绝对的解决哈希冲突,只可能使发生哈希冲突的概率降低

解决哈希冲突的方式:

1.闭散列:从发生哈希冲突的位置开始，找"下一个"空位置

找"下一个"空位置的方式:

a.线性探测:逐个挨着依次往后查找---注意:如果走到末尾，从头再来

优点:探测方式简单

缺陷:容易产生数据的堆积；冲突的数据容易堆积在一起

b.二次探测:假如:首次计算的哈希地址是: H0--->冲突-->向后探测，假设第i次探测的方式: Hi= HO + i^2； Hi= H0—i^2

优点:解决了线性探测中数据堆积的问题

缺陷:如果表格中空余位置比较少，可能需要探测多次

实现注意:必须要给一些状态比较 EMPTY、 EXIST、DELETE

哈希表中:随着元素的不断增多，哈希表发生冲突的概率会不断提升，会影响哈希表的性能

例子:教室有40个座位，10个人在教室中.上课和30个人在教室中上课

哈希负载因子:表格中的元素/哈希表的容量

闭散列缺陷:空间利用率太低了，处理有点麻烦

2.开散列开链法链地址法 哈希桶

扩容时机:开散列最佳状态----如果每个桶中刚好都挂了一个节---->该场景之后再插入元素---->一定会发生哈希冲突元素个数= =桶的个数(表格的容量)

文件: 100亿个ip地址，找到出现次数最多的前K条地ip地址---->哈希切割----类似哈希桶:将相同的元素文件到一个文件中unordered_map统计次数,借助优先级队列，对次数建立一个小堆

位图:用一个比特位表示存在与否的状态信息,如果笔试时，需要用到位图时一-般不需要自己创建--bitset

面试题:

位图的应用: 40亿个不重复的元素，快速查找某个元素是否在该集合中

位图的变形应用: 100亿个整形数据， 快速找只出现一次的数据，100亿个整形数据，快速找只出现次数不超过2次的数据

位图和多哈希

布隆过滤器:位图+多个哈希函数

紧促类型的快速查找的数据结构

注意:数据不存在--->一定不存在

数据存在--->可能存在