补天：数据结构与算法

0、写在前面

这一份材料是在数算笔试前夜开始做的，主要的材料来源是郭炜老师上课的PPT（我们班的课件个人认为更加适用于实践而非理论的笔试）。因为当时数算主要是编程实践上的认知，很多笔试要考的内容都没有太多印象，因而把这份材料称为“补天”。一直到考试的时候我也才只写到了拓扑排序，后面的内容是考完试补充的。很多地方因为自己在学编程的时候知道了就没有详细写，严密性和完整性当然比不上同门的各位大佬Orz。

1、数据结构

被描述时需要指明**逻辑结构、存储结构和可进行的操作。**

2、逻辑结构的分类

集合结构：节点之间不存在关系。

线性结构：除头节点外每个节点有唯一前驱，除尾节点外每个节点有唯一后缀。

树结构：根节点无前驱，叶子节点无后缀，其他节点每一个有唯一前驱和一或多个后缀。

图结构：每个节点可以有多个前驱和后缀。

3、存储结构的分类

顺序结构：节点在内存中连续存放，所有节点占据一片连续的内存空间。（list）

链接结构：可不连续存放，每个节点中存有指针指向其前驱结点或者后缀节点。

索引结构：将节点关键字取出单独存储，并且为每一个关键字配一个指针指向对应的节点，便于按照关键字来进行索引。

散列结构：散列函数以结点的关键字为参数算出一个结点的存储位置。

逻辑结构和存储结构无关！一种逻辑结构可以用多种不同的存储结构进行存储

4、二分查找的时间复杂度

O(log(n))，对于有序的列表等可以使用。

5、线性表

有唯一的头元素和尾元素，除头元素外每个元素有唯一前驱，除尾元素外每个元素有唯一后缀；每个元素属于相同的数据类型；分为**顺序表**和**链表**两种。

6、顺序表

元素在内存中**连续**存放，每个元素有唯一下标，根据下标访问元素的时间复杂度O(1).

7、链表

元素**非连续**存放，通过指针来进行链接；每个节点除了元素还有next指针，用来指向自己的后继；访问的时间复杂度O(n)；插入和删除的时间复杂度都是O(1)；有单链表、循环单链表、双向链表、循环双向链表等多种形式。

8、栈

四种操作（push压入一个元素，pop弹出一个元素，top查看栈顶元素，isEmpty查看是否为空），利用list来实现时间复杂度均为O(1).

9、队列

一头进（push）另一头出（pop），且先进先出，要求两种操作的时间复杂度都是O(1).

一般队列实现方法：用足够大的列表来实现，维护一个头指针head和一个尾指针tail.

pop(): head+=1

push(x): array[tail]=x, tail+=1

isEmpty(): return head==tail

循环队列实现方法：用一个容量为capacity的列表来实现，维护头指针head和尾指针tail，tail必须始终指向一个空的元素（或者通过维护size变量来做）

pop(): head=(head+1)%capacity

push(x): array[tail]=x; tail=(tail+1)%capacity

isEmpty(): return head==tail

isFull(): return (tail+1)%capacity==head

10、二叉树

相关概念:

结点：由三部分组成（数据，左子结点指针，右子结点指针）。

结点的度：结点的非空子结点数目，也可以说是节点的子结点数目。

叶结点：度为0.

内部结点：度不为0.

兄弟结点：父结点相同。

结点的层次：**根结点为0。**

二叉树的高度：结点最大层次数，只有一个结点的二叉树高度是0.

特殊类型二叉树:

完美二叉树：每一层的结点数目都达到最大。

满二叉树：没有度为1的结点

完全二叉树：除最后一层外，每一层的节点数目达到最大；若最后一层不满，则所缺节点一定是在右边连续若干个。

二叉树的性质

节点数为n，边数为n-1.

n个结点的非空二叉树至少有$[\log_2(n+1)]$层结点。

任意一棵二叉树中，有以下方程组成立：

$N-1=n_1+2n_2$ （边数关系）

$N=n_0+n_1+n_2$ （节点数关系）

由此可得$n_0=n_2+1$对任意一棵二叉树成立

非空满二叉树叶结点数目等于分支结点数目加1。

非空二叉树中的空子树数目等于其结点数目加1。

完全二叉树的性质

1度节点有0或1个.

有n个结点的完全二叉树有[(n+1)/2]个叶节点

有n个叶子结点的完全二叉树有2n-1或2n个结点。

有n个结点的非空完全二叉树有$[\log_2(n+1)]$层结点。

二叉树的遍历

前序：根+左+右

中序：左+根+右

后序：左+右+根

仅给定前序和后序遍历序列不能确定该二叉树；给定中序遍历序列和前后序中的一种则可以确定。

最优二叉树（哈夫曼树）

给定n个节点，每个节点有权值w_i，构造一棵二叉树，使得带权路径和最小。

做法：

1)从森林中取出最小的两个节点作为两个子节点建一二叉树，树根的权值为两个节点权值之和。

2)把树根节点加入森林，重复上述过程。

11、堆

定义

一棵完全二叉树，所有节点均满足“自身比子节点优先级高”这个条件。

性质

堆顶元素优先级最高。

堆中任意一棵子树都是堆。

添加元素和删除元素的时间复杂度都是O(log(n))，对于一个无序列表，建堆的时间复杂度O(n).

作用

实现优先级队列，进行堆排序（时间复杂度O(n*log(n))，只需要O(1)的额外空间，递归写法需要O(n)的额外空间）

操作

添加元素（上浮）：往末尾添加新元素，将该元素与父结点元素比较，如果优先级高于父结点就交换两个节点，直至该新元素优先级低于父节点或者成为堆顶。

删除元素（下沉）：弹出堆顶元素，首先将堆顶和堆尾元素交换位置，弹出堆尾，再将新的堆顶不断和儿子中优先级较高且高于新堆顶本身的元素交换位置，直至该新堆顶下沉至叶子节点，或者两个儿子优先级都不高于该节点。

12、树和森林

树的性质

结点度数最多为K的树，第i层最多Ki个结点(i从0开始)

结点度数最多为K的树，高为h时最多有$(K^{h+1}-1)/(k-1)$ 个结点。

n个结点的K度完全树，高度h是$\log_k(n)$向下取整

n个结点的树有n-1条边

树的二叉树表示法：左儿子右兄弟

森林：不相交的树的集合，各树之间有序

森林的二叉树表示法：每个结点及其左子树，是对应树二叉树的表示，右子树根节点及右子树根节点的左子树，是下一棵树对对应的二叉树表示形式。

13、二叉排序树（搜索树，查找树）

对于每个结点，其左子树所有节点均小于该节点，右子树所有节点均大于该节点。一棵二叉树是二叉排序树，当且仅当它的前序遍历序列**递增。**

查找、插入：比较要查找（插入）的结点与当前根节点，如果要查找（插入）的比当前节点值小，则在当前节点的左子树中继续查找（插入）；如果比当前节点值大，则在右子树中进行查找（插入）。

删除：找到要删除节点的前序遍历序列的后继（或前驱），用该结点的值覆盖要删除节点，然后删除该节点（前驱或后继）本身。

时间复杂度：建树时可以有O(n*log(n))的时间复杂度，平均建好的树深度为log(n)，但无法保证查找插入删除的O(log(n))时间复杂度。

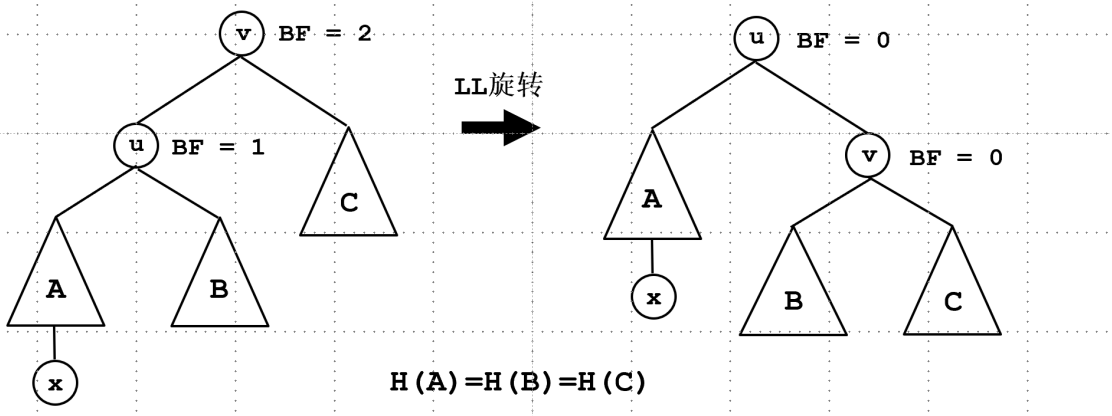
14、AVL树

四种旋转：

LL型：左子节点成为根节点，原根节点成为新根节点的右子节点，原左子节点的右子节点成为原根节点的左子节点

➢ LL旋转rotateLL

适用场景：新增的结点x位于祖父v的左子树的左子树



H(A)=H(B)=H(C)

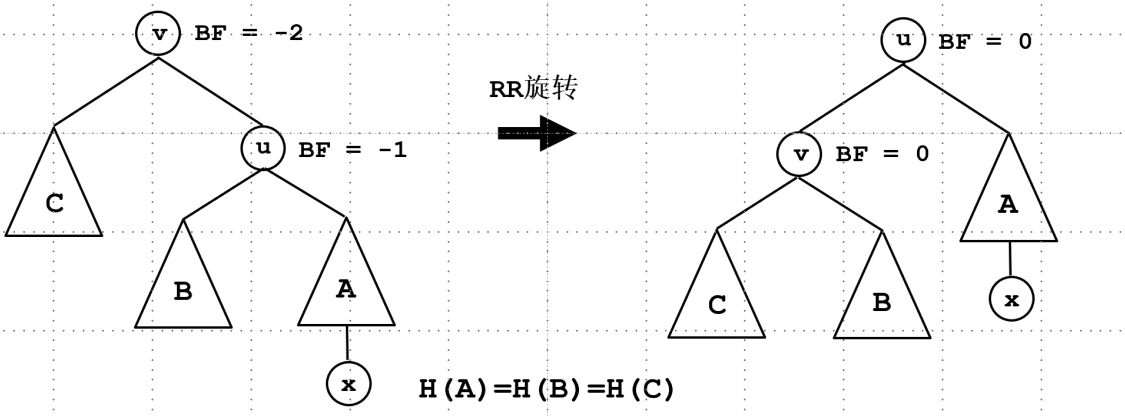
v.BF==2且u.BF==1即可断定要LL旋转，此时必有 H(A) = H(B)=H(C)

34

RR型：右子节点成为根节点，原根节点成为新根节点的左子节点，原右子节点的左子节点成为新左子节点的右子节点。

➢ RR旋转rotateRR

适用场景：新增的结点x位于祖父v的右子树的右子树



H(A)=H(B)=H(C)

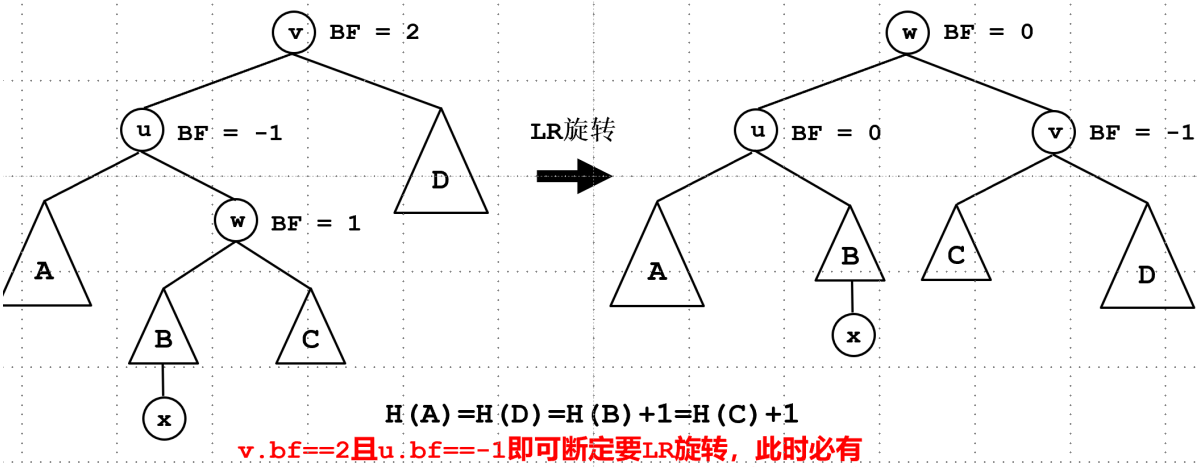
v.BF==-2且u.BF==-1即可断定要RR旋转，此时必有 H(A) = H(B)= H(C)

33

LR型：先对u进行一次左旋，后对v进行一次右旋。

➢ LR旋转rotateLR

适用场景：新增的结点x位于v的左子树的右子树（x在C下面类似）

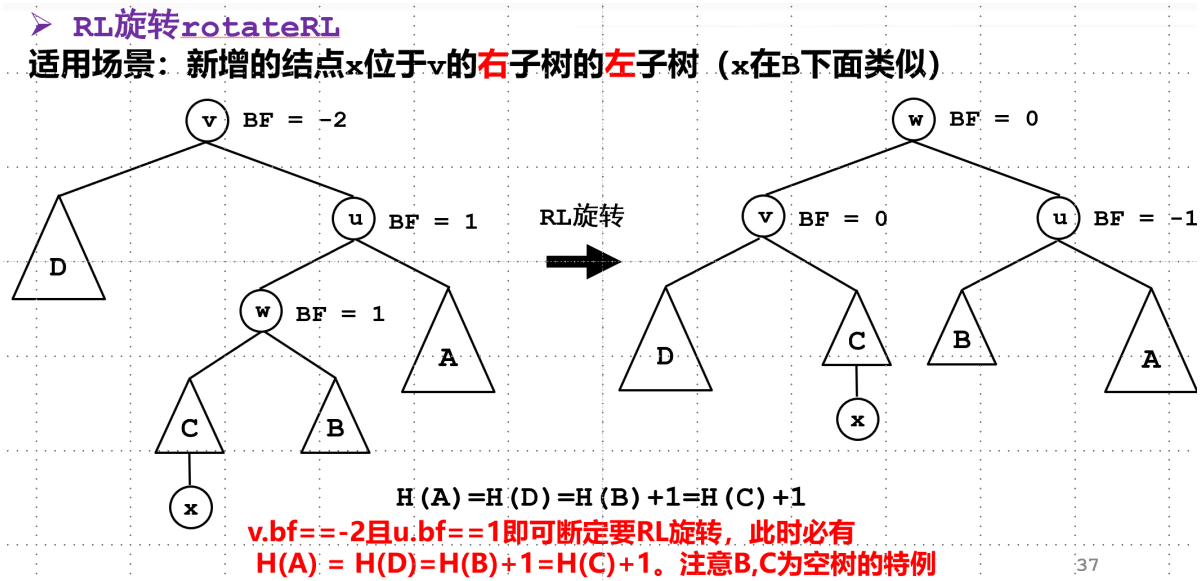


H(A)=H(D)=H(B)+1=H(C)+1

v.bf==2且u.bf==-1即可断定要LR旋转，此时必有
H(A) = H(D)=H(B)+1=H(C)+1。要注意B,C为空树的特例

36

RL型：先对u进行一次右旋，后对v进行一次左旋。



AVL具有**O(log(n))**的插入复杂度，其旋转操作复杂度为O(1)，每一次添加节点只会做一次。

15、图的相关概念

顶点的度数：和顶点相连的边的数目。

顶点的入边出边、入度出度：**有向图**中，顶点的入边是以该顶点为重点的边，入度即为该顶点入边的条数。出边和出度依此类推。

简单路径：除了起点和终点可能相同以外，其他顶点均不相同。

完全无向图和完全有向图：完全无向图指任两个点均有边相连的无向图，完全有向图指任两个顶点均有双向边相连的有向图。

连通无向图：图中任意两个顶点u和v互相可达。

强连通有向图：图中任意两个顶点u和v互相可达。

连通分量（极大连通子图）：无向图的一个子图，是连通的，且再添加任何一些原图中的顶点和边，新子图都不再连通。

强连通分量：有向图的一个子图，是强连通的，且再添加任何一些原图中的顶点和边，新子图都不再强连通。

相关性质

边数=顶点度数和的一半

n个顶点的连通图至少有n-1条边

图的遍历

常用算法：BFS，DFS

对于邻接表形式给出的图，时间复杂度O(V+E)

对于邻接矩阵形式给出的图，时间复杂度O(V^2)

16、拓扑排序

概念：在有向图中求一个顶点的序列使其满足以下条件：

1）每个顶点出现且只出现一次

2）若存在从A到B的路径，则在拓扑排序中A必然出现在B之前

充要条件：能够进行拓扑排序的图是**有向无环图(DAG)**

算法实现：

1）从有向图中寻找一个入度为0的点作为起始。

2）把该点加入序列中，将该点和所有以它为起点的边从图中删除。

3）重复以上两步，直至图中不存在入度为零的点。

(说明：如果结束后，序列中的点少于图中的点，说明该有向图存在环)

17、最小生成树

生成树：取一无向图中所有顶点和一部分边构成一子图，使得该子图具有以下性质：

1）所有顶点连通

2）不存在回路

则称该子树是原图的一棵生成树

性质：

1）n个顶点的图生成树有n-1条边

2）无向图的**极小连通子图**就是生成树

最小生成树：对于一无向连通带权图，可能得到多个不同权值的生成树，其中，权值最小的生成树称为**最小生成树**

Prim算法

思路：1）初始时设置所有点到起点的距离为无穷大，成一列表

2）从图中拿出起始点，加入“已访问集合”中

3）考察该点到所有邻接点的距离，若该距离小于列表中现有的距离，则更新

4）选择一个当前距离最小的点，重复以上2）和3）两个步骤直至得到最小生成树。

使用邻接矩阵来存放图并且不加优化，prim算法的时间复杂度为$O(n^2)$；使用邻接表存放图且使用堆来选取最小距离，时间复杂度可以降低至O(E*log(V))

Kruskal算法

思路：1）将所有边按照权值从小到大排序

2）选取当前未被选取的最小边，若该边加入生成树中不构成回路，保留；反之舍弃

3）重复上述步骤，直至生成树中包含n-1条边

使用并查集来判断是否存在环路，时间复杂度O(E*log(E))

18、最短路径

Dijkstra算法

适用条件：无负权边的带权有向图或无向图的单源最短路径问题。

思路：1）使用一个列表记录所有边到起点的最小值，初始时，这个列表里面的“最小值”均为无穷大。

2）取出起点，加入已访问集合，并且更新各邻接点到起点的距离。

3）选择一个当前距离起点最短的点，把它加入已访问集合，并且**若：**

该点到邻接点距离+该点到起点距离<列表记录值

则将列表中邻接点的距离更新。

4）重复3，直到走到终点。

使用堆来实现的优先级队列可以将时间复杂度降低至O(E*log(V))

Floyd算法

适用条件：有向图或无向图每一对顶点之间的最短路径，允许有向图存在负权边，但不能存在负权回路。

思路：1）使用一个二维矩阵记录某两点之间的最短距离，初始化时仅将直接存在的边记录，其他边均为无穷大

2）从某一个点开始，以该点为中间点，逐个更新矩阵中每两个点之间的最短路径长度，即：

存在两点A和B，此时以V为中间点：

若distance[A->B]>distance[A->V]+distance[V->B]

则将A->B的最短距离更新为A->V和V->B的最短距离之和

3）依次对每一个点进行如此操作，即可得到一个最短路径构成的矩阵

该算法的时间复杂度为$O(n^3)$

19、散列表

一种直接通过关键字来索引值的数据结构，查询、插入和删除的时间复杂度均为O(1).

散列表中最重要的要素是散列函数，它能把值对应到一定的关键字上去，建立起关键字到值的映射。

散列函数的设计要求包括以下几点：

1）函数简单，计算速度快；

2）结果尽量均匀分布

3）结果尽量减少冲突

4）结果可以覆盖整个存储区，避免有些存储单元被浪费

冲突消解

难以避免不同值哈希后对应的关键字相同，因此需要冲突消解

内消解：只使用散列表基本存储区来解决冲突

线性探查：遇到冲突时挨个挨个往后找，把新的值映到第一个空的关键字中。

双散列探查：设置另一个散列函数，在遇到冲突的时候使用另一个散列函数来消解冲突。

外消解：使用散列表基本存储区之外的额外方法来解决冲突

溢出区方法：用另一个列表顺序存放冲突的数据，查找时先看散列值的位置，没找到则到溢出区去查找.

桶散列：关键字不直接存值而是存列表的开头指针。所有散列值相同的元素，都放在一个链表或列表便于查找。

20、排序算法

类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	Shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
归并排序		$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定

插入排序

思路：1）将数据分为有序区和无序区两个区段，有序区在左端无序区在右端

2）每一次从无序区的左端选取一个元素，插入到有序区的合适位置

3）重复直至有序

希尔排序

思路：1）按照初始步长，逐步比较当前元素和加上步长后元素的大小，不符合要求的就交换位置

2）进行一次这样的交换以后，步长减少一，再一次进行这个操作

3）直至步长减少至1进行最后一次排序使得该数组有序

选择排序

思路：1）将序列分成有序部分和无序部分，有序在左无序在右，初始时，有序部分无元素

2）每次找到无序部分最小的元素，和无序元素的最左边进行交换

3）重复直至有序

堆排序

思路：1）先将待排序列变成一个堆

2）依次弹出最小（大）值完成排序

冒泡排序

思路：逐次比较相邻两个元素的大小，不符合要求的交换位置，重复直至有序。

快速排序

思路：1）在数组中选定一个基准元素（一般选第一个元素），使两个指针（左指针和右指针）分别指在两端。此时左指针指向的位置为空

2）移动右指针，碰到第一个小于基准元素的元素时停下，将该元素插入左指针所指向的空位置，此时右指针所指的位置变为空

3）移动左指针，碰到第一个大于基准元素的元素时停下，将该元素插入右指针所指向的空位置，此时左指针所指的位置变为空

4）重复进行以上两个步骤，直至左右指针相遇，把基准元素插入相遇的空位置中

5）对左区间和右区间递归地进行以上步骤，直至数组有序

归并排序

思路：1）将前一半排序

2）将后一半排序

3）将排好序的两半进行合并

实际操作的过程当中经常递归地做这件事来排序

其他排序算法由于考试和做题较少涉及，此处不赘述。