001-二维数组中的查找

标签: 搜索与排序

思路:

利用二维数组由上到下，由左到右递增的规律，从左下角(或右上角)元素开始搜索，会产生以下几种情况:
(1)当前元素等于目标元素，那么直接返回true;
(2)当前元素大于目标元素，那么向上搜索，因为向上是递减的;
(3)当前元素小于目标元素，那么向右搜索，因为向右是递增的;
重复上述过程直到搜索到目标元素或者搜索坐标超出界限。

002-替换空格

标签: 数组（字符串）

思路：

(1)从前向后记录' '数目；
(2)从后向前替换' '，从后向前替换的时候各个字母的位置要依据空格的个数根据规律求得。

003-从尾到头打印链表

标签: 链表

思路：

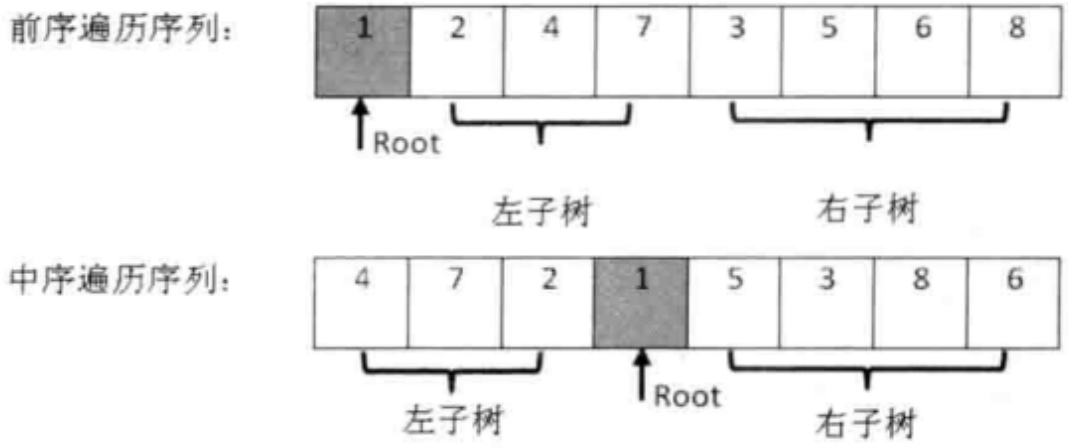
(1) 正向遍历链表，将链表中每个节点的值从结果集的开头处放入结果集中；
(2) 当遍历完链表的时候，此时得到的顺序就是从尾到头。

004-重建二叉树

标签: 二叉树

思路:

(1)在前序序列中越早出现的数字为根节点，首先在先序序列中找到根节点。
(2)根据根节点在中序序列中的位置以及根节点子树的左右范围可以确定其左右子树的长度。
(3)若根节点左子树存在，那么先序数组中根节点元素的下一个元素就是其左孩子，若右子树存在，那么从根节点元素向后移动"左子树长度"个位置的节点就是其右孩子。
(4)重复上述过程直到所有节点都被添加到树中。

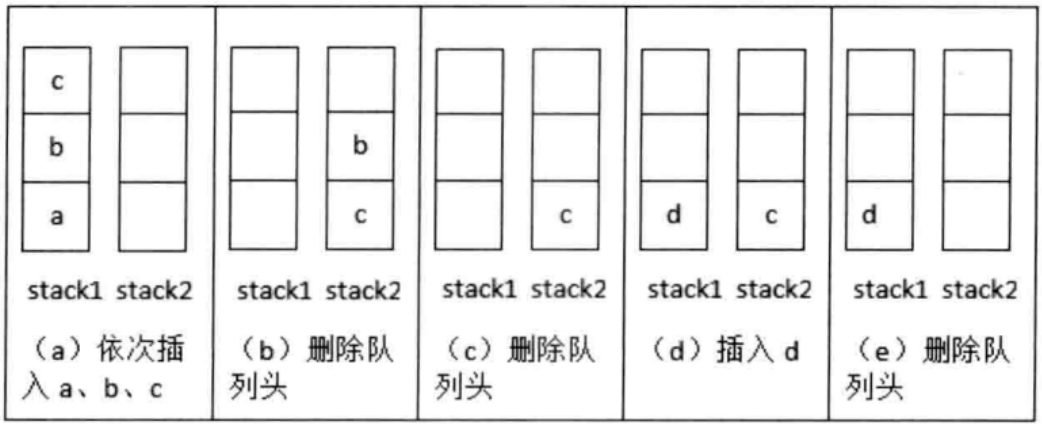


005-用两个栈实现队列

标签: 栈与队列

思路:

(1)入队操作只需将元素入栈stack1即可。
(2)出队操作的总体思想是将stack1中的元素自栈顶向下依次放入stack2中，这样将越早进入stack1的元素，就放在了stack2中越靠近栈顶的位置上，最后将stack2栈顶元素取出。这样就和队列"先进先出"的特点完全一致。



006-旋转数组的最小数字

标签: 搜索与排序

思路:

先将l指向nums[0]，将r指向nums[len-1]，之后求出其中间节点nums[m]和nums[r]进行比较:
(1)如果nums[m]大于nums[r]，那么说明最小值在m之后，此时将l移动到nums[m+1]，继续在m+1到r范围内搜寻最小值;
(2)如果nums[m]小于nums[r]，那么说明最小值在nums[m]之前，或者就是nums[m]，此时将r移动到nums[m]处，在l到m范围内搜寻最小值。
重复上述过程直到l==r。此时l指向的就是最小值。

007-斐波那契数列

标签: 动态规划, 斐波那契数列

思路:

动态规划版的斐波那契数列。

技巧:

（1）斐波那契问题如果使用递归可能会爆栈且会产生过多重复计算。

（2）不使用中间变量进行斐波那契公式：

```
tmp2 += tmp1;

tmp1 = tmp2 - tmp1;
```

008-跳台阶

标签: 动态规划, 斐波那契数列

思路：

动态规划版的斐波那契数列。

技巧：

（1）斐波那契问题如果使用递归可能会爆栈且会产生过多重复计算。

（2）不使用中间变量进行斐波那契公式：

```
tmp2 += tmp1;

tmp1 = tmp2 - tmp1;
```

009-变态跳台阶

标签: 递归, 数学

递归思路:

变形的斐波那契数列，f[n] = f[n-1]+f[n-2]+f[n-3]+...+f[n-(n-1)]+f[n-n]。

数学思路:

f(n-1) = f(0) + f(1)+f(2)+f(3) + ... + f((n-1)-1) = f(0) + f(1) + f(2) + f(3) + ... + f(n-2)
f(n) = f(0) + f(1) + f(2) + f(3) + ... + f(n-2) + f(n-1) = f(n-1) + f(n-1)
可以得出:
f(n) = 2*f(n-1) = 2*2*f(n-2) = 2*2*2*f(n-3) = 2*2*2*2*...*f(1) = 2**(n-1)

还可以想成其实是隔板问题，假设n个台阶，有n-1个空隙，可以用0~n-1个隔板分割，
c(n-1, 0)+c(n-1, 1)+...+c(n-1, n-1)=2^(n-1)。

010-矩形覆盖

标签: 动态规划, 斐波那契数列

思路:

动态规划版的斐波那契数列。

技巧:

（1）斐波那契问题如果使用递归可能会爆栈且会产生过多重复计算。

（2）不使用中间变量进行斐波那契公式：

```
tmp2 += tmp1;

tmp1 = tmp2 - tmp1;
```

011-二进制中1的个数

标签: 位运算

思路：

逐位判断n里面有多少个1。但是要注意一点，最好不要在移位中改变n，使n = n>>1，这样当n是负数时，右移在最高位补得是1，那么会有无数个1，会产生死循环。

012-数值的整数次方

标签: 分治法, 快速求幂

分治思路：

使用分治法，假设幂为5：5->2+3->(1+1)+(1+2)->(1+1)+(1+(1+1))，当幂为0时直接返回1， 当幂为1时直接返回base。
在此基础上使用一个哈希表来记录已经求出幂对应的值，如果再之后的运算中遇到相同的幂，直接返回结果。

快速求幂思路：

快速求幂法：3^11 -> 3^10*3 -> 9^5*3 -> 9^4*9*3 -> 81^2*9*3 -> 6561^1*9*3 -> 43046721^0*6561*9*3

013-调整数组顺序使奇数位于偶数前面

标签:　　数组（字符串）

思路：

(1)遍历原数组，将原数组中的奇数和偶数按照顺序分别放入a1和a2两个队列中。
(2)重构原数组，先将奇数放入原数组的前半部分，之后再将偶数放入原数组的后半部分。

014-链表中倒数第k个结点

标签: 链表

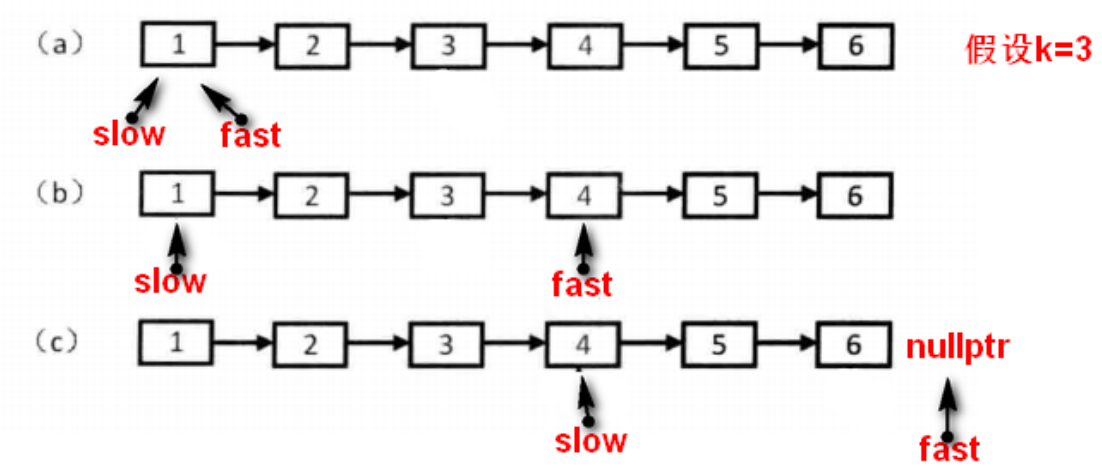
思路:

(1)将两个指针slow和fast均指向头结点，之后先将fast指针向后移动k个节点。

(2)两指针同时向后移动，当fast指针指向NULL时，slow指针正好指向倒数第k个节点。

技巧:

输入的参数k为0时，由于k是一个无符号整数(unsigned int)，那么k-1得到的将不是-1,而是4294967295(无符号的0xFFFFFFFF),要注意该种情况。

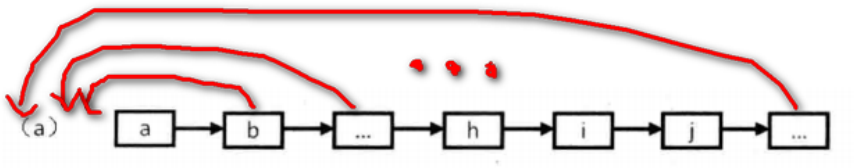


015-反转链表

标签: 链表

思路:

遍历链表，依次将链表的节点从原链表中删除并且插入链表的开头。

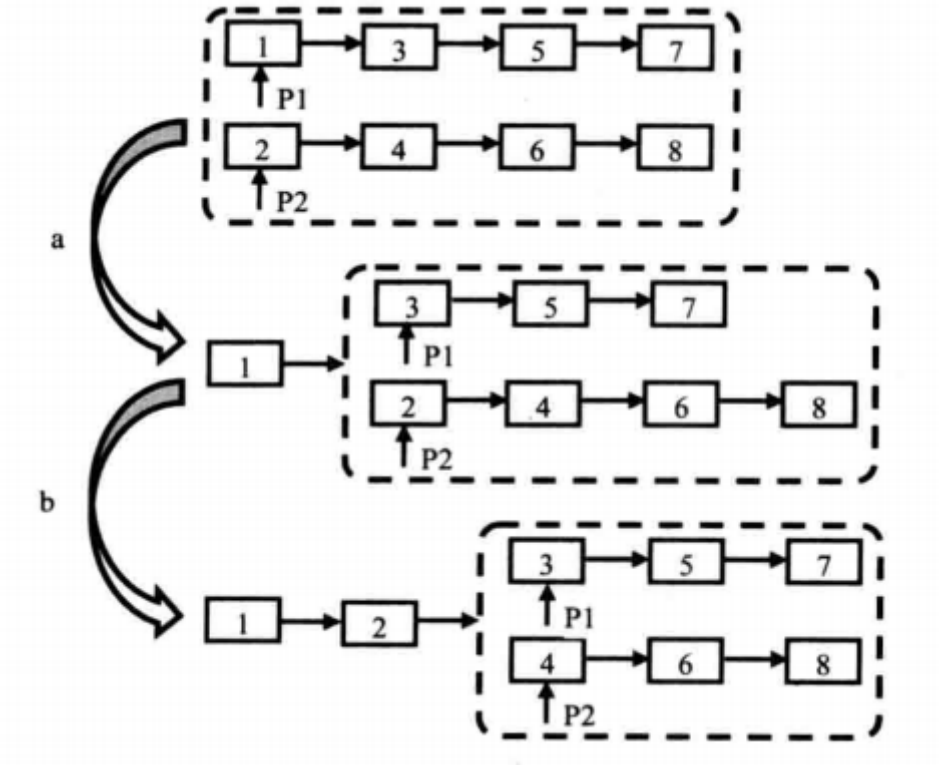


016-合并两个排序的链表

标签: 链表

思路:

(1)用两个指针p1和p2分别指向pHead1和pHead2，并比较p1->val和p2->val的大小，将较小节点添加入新链表，并且将相应指针向后移动一个节点，另一个指针不动。
(2)若一条链表先遍历完成，那么就继续遍历另一条链表，并将另一条链表剩余部分添加到新链表中。
(3)重复上述过程直到两个链表所有节点全部加入新链表。

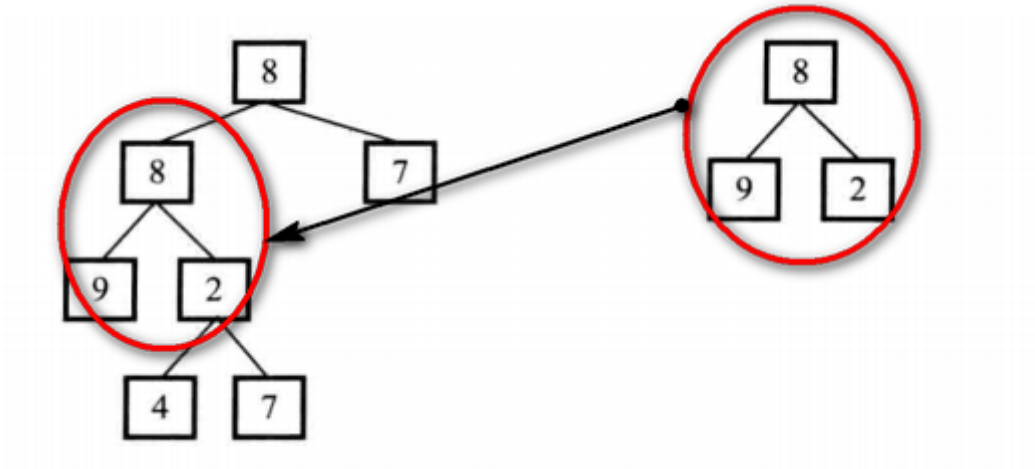


017-树的子结构

标签: 二叉树

思路:

(1)在A树中寻找和B树根节点相同的节点N，如果找不到N直接返回false，如果找到了N则执行(2)。
(2)同时遍历B树和以N为根节点的子树T，比较各自对应节点是否相等。要注意一下结构的定义，即B树是以N为根节点子树T的一部分，T可以有B没有的节点，但是B不可以有T没有的节点。
(3)如果(2)中找到了子结构则返回true，否则返回(1)中继续寻找N，直到找到子结构或者A树被遍历完。

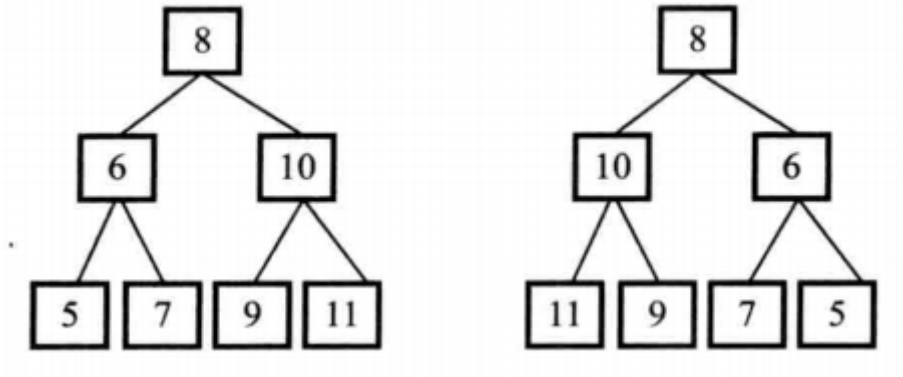


018-二叉树的镜像

标签: 二叉树

思路:

前序遍历二叉树，将每个节点的左右子节点交换即可。



019-顺时针打印矩阵

标签: 模拟

思路:

顺时针打印实质上就是从外向里一圈一圈打印，每一圈之间的长和宽均相差2，其中每一圈的开头元素的x, y相等，每一圈x, y均相差1。例如：

```
 1   2   3   4   5
 6   7   8   9  10
11  12  13  14  15
16  17  18  19  20
```

第一圈为1, 2, 3, 4, 5, 10, 15, 20, 19, 18, 17, 16, 11, 6；开头元素为1([0, 0])

第二圈为7, 8, 9, 14, 13, 12；开头元素为7([1, 1])

当遍历到某圈时，该圈长或宽小于等于0，则说明打印已经完成。

在每一圈的打印过程中要注意右下到左下， 左下到左上这两条边:

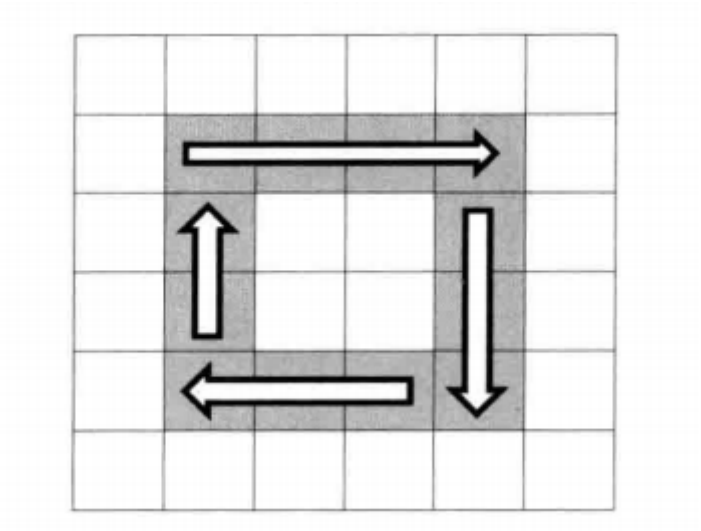
(1)右下到左下这条边一定要保证宽度大于1的时候，即不为单行的时候才打印这条边，否则在单行情况下会导致重复打印。

(2) 左下到左上这条边一定要保证长度大于1的时候，即不为单列的时候才打印这条边，否则在单列情况下会导致重复打印。

特例:

(1)单行：{{1, 2, 3, 4, 5}}

(2)单列：{{1}, {2}, {3}, {4}, {5}}



020-包含min函数的栈

标签: 栈与队列

思路：

本题的关键在于如何处理序列中的最小值，用一个辅助栈来保存序列中的最小元素：

(1)入栈时，如果辅助栈为空，或者辅助栈栈顶元素大于等于入栈元素，则将新元素也放入辅助栈。辅助栈中的元素不是原始栈元素的排序序列，而是视原始栈的入栈情况来保存最小元素，例如：原始栈入栈为3 2 5 1 4，那么辅助栈的情况为3 2 1。

(2)出栈时判断原始栈的出栈元素是否和辅助栈的栈顶元素相等，若相等则将辅助栈的栈顶元素也出栈。

(3)返回栈中最小值时，直接将辅助栈的栈顶元素返回。

021-栈的压入、弹出序列

标签: 栈与队列

思路:

用一个栈来模拟入栈出栈过程:
(1)将入栈序列的元素依次入栈，在入栈过程中将栈顶元素和出栈序列中index位置元素进行对比;
(2)如果相等，那么栈顶元素出栈，index向后移动一位;之后继续对两者进行比较，直到不相等则执行(1);
重复上述过程直到所有入栈序列中的元素都入过栈，如果此时辅助栈为空说明出栈顺序正确，辅助栈不为空则出栈顺序错误。

022-从上往下打印二叉树

标签: 二叉树

思路：

层次遍历二叉树，并按照顺序保存每个节点的值：
(1)从队头取出最早进入队列的结点，将其保存在结果集中；
(2)如果该结点有子结点，则将其子节点(如果左右子节点均存在，则按照先左子节点后右子节点的顺序)入队，之后继续执行(1)；
重复上述操作，直到队列为空，即所有节点都被遍历完。

023-二叉搜索树的后序遍历序列

标签: 二叉树

思路1(判断后序序列节点在中序序列中的位置是否正确):

(1)根据二叉搜索树的特点，中序序列为从小到大排列的有序序列。先将后序序列排序得到中序序列。
(2)在后序序列中越晚出现的数字为根节点，首先在后序序列中找到根节点。
(3)之后判断根节点在中序序列中的位置是否在其左右子树范围之间。若不在直接返回false，若在则判断其子节点的正确性，执行(4)。
(4)若根节点右子树存在，那么后序数组中根节点元素的上一个元素就是其右孩子；若左子树存在，那么从根节点元素向前移动"右子树长度"个位置的节点就是其左孩子。找到子节点之后判断子节点位置是否正确。
(5)重复上述过程直到所有节点都被判断完。

思路2(判断二叉搜索树左右子树的数值大小):

根据二叉搜索树的特点，左子树均小于根节点，右子树均大于根节点。二叉搜索树的后序遍历序列S，最后一个元素是x(也就是根节点)，如果去掉最后一个元素的序列为T，那么T满足：T可以分成两段，前一段(左子树)均小于x，后一段(右子树)均大于x，而前序序列反之。

024-二叉树中和为某一值的路径

标签: 二叉树

思路：

(1)前序遍历所给定的树，并且记录遍历的路径；
(2)如果一条路径上的和等于目标值就放入结果集之后向上还原路径，继续遍历其他路径；如不等于目标值或者在一条路径的遍历过程中该路径之和已经大于目标值则直接向上还原路径，继续遍历其他路径；
(3)重复上述过程直到所有路径都被遍历完。

025-复杂链表的复制

标签: 链表

思路:

(1) 复制每个节点，并将复制节点放到原节点之后
(2) 根据原有节点的随机指针为新节点的随即指针赋值
(3) 将链表拆分为原有链表和复制链表

具体分为三步：

（1）在旧链表中创建新链表，此时不处理新链表的兄弟结点

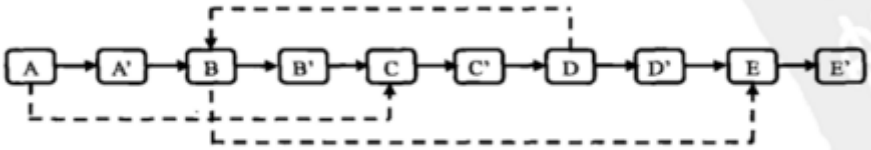


图 4.9 复制复杂链表的第一步

（2）根据旧链表的兄弟结点，初始化新链表的兄弟结点

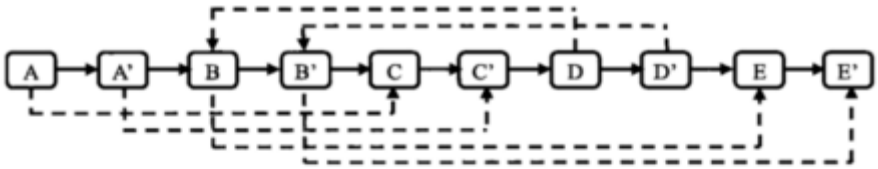


图 4.10 复制复杂链表的第二步

（3）从旧链表中拆分得到新链表

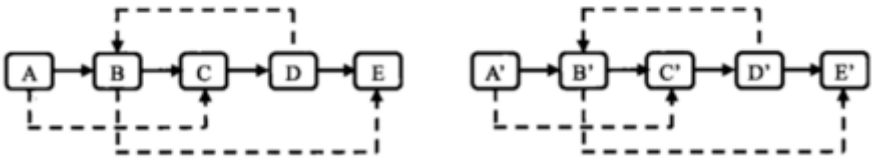


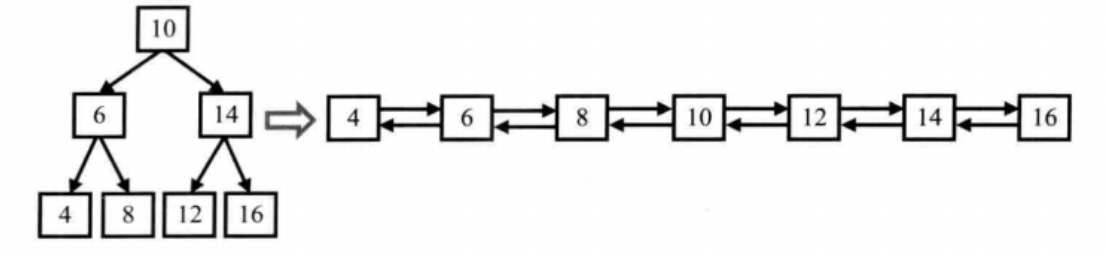
图 4.11 复制复杂链表的第三步

026-二叉搜索树与双向链表

标签: 二叉树

思路:

二叉搜索树的特点是中序遍历二叉搜索树会得到一个有序序列:
(1)采用中序遍历的方法来遍历二叉搜索树，在遍历过程中记录每个节点在中序遍历过程中前一个被遍历的节点front;
(2)每当遍历到一个节点root，将root的左指针指向front，将front的右指针指向root(每个节点的left指针指向中序遍历序列中其前一个节点，right指针指向中序遍历序列中其后一个节点)。之后将front指向root，继续遍历下一个节点;
(3)重复(1)(2)的过程，直至中序遍历结束，双向链表即构建完成。



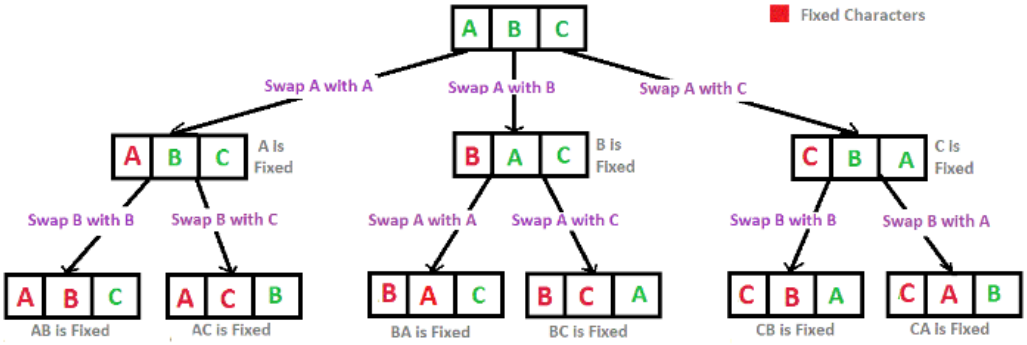
027-字符串的排列

标签: 递归

思路:

(1) 全排列字符串，当一个字符串中有重复字母时，在全排列过程中要做去重处理。

(2) 用sort函数对全排列集进行排序，让各种情况按照字典序排序。



Recursion Tree for Permutations of String "ABC"

028-数组中出现次数超过一半的数字

标签: 排序与搜索, 数组（字符串）

哈希表思路：

遍历数组记录每个数字出现次数：
(1)当某个数字出现次数已经超过一半则立刻返回该数；
(2)如果没有出现次数超过一半的数则返回0。

快排思路：

数组中有一个数字出现的次数超过了数组长度的一半。如果把这个数组排序，那么排序之后位于数组中间的数字一定就是那个出现次数超过数组长度一半的数字。也就是说，这个数字就是统计学上的中位数,即长度为n的数组中第n/2大的数字。我们是用快排思想可以很快找到第k大的数（具体算法见029-最小的K个数）。

029-最小的K个数

标签: 堆, 排序与搜索

快排思路：

(1)利用快速排序的思想获取每一次排序后的分割(中轴)点位置。根据每一次得到的中轴点位置来调整数组；
(2)如果得到的中轴点位置index比k-1(第k个数)大，则继续执行(1)向index之前寻找第k个数； 如果得到的中轴点位置index比k-1(第k个数)小，则继续执行(1)向index之后寻找第k个数；
(3)反复执行(1)(2),直到得到的中轴点位置index等于k-1，此时比第k个数字小的所有数字都位于中轴点的左边，比第k个数字大的所有数字都位于中轴点的右边。位于数组左边的k个数字就是最小的k个数字(这k个数字不一定是排序的)。

大顶堆思路：

使用一个大顶堆来筛选出最小的k个数。首先遍历input数组：
(1)如果堆中数据个数小于k则直接将input[i]入堆；
(2)如果堆中数据个数已经等于k，则将input[i]与堆顶数据(即堆中的最大值)比较，如果比堆顶元素小，则将堆顶元素移出堆，input[i]入堆，反之则继续遍历input，直到将input数组遍历完。要注意当k==0时这种特殊情况。

030-连续子数组的最大和

标签: 动态规划

思路:

用dp[i]表示以array[i]结尾的子序所能得到的最大子序和，得到以array[i]结尾的最大子序有两种可能：
(1)一种是将array[i]加到array[i-1]结尾的最大子序上形成一个新子序；
(2)一种是array[i]本身自成一个子序；
所以当遍历到第i个元素时，将把array[i]加到以array[i-1]结尾的最大子序上所形成新的以array[i]为结尾的子序同array[i]本身作比较，将二者中的较大值赋给dp[i]。，由此可以得到递推公式：
dp[i] = max(dp[i-1]+array[i],array[i])；
并且可以知道以第一个数字为结尾的子序就是其本身，所以dp[0]就是array[0]，由此可以推导得到所有array[i]，选出其中最大值即为最大子序和。

031-整数中1出现的次数（从1到n整数中1出现的次数）

标签: 数学

思路:

设定整数点(如1、10、100等等)作为位置点i(对应n的各位、十位、百位等等),分别对每个数位上有多少包含1的点进行分析根据设定的整数位置，对n进行分割，分为两部分，高位n/i，低位n%i：
(1)当i表示百位，且百位对应的数>=2,如n=31456,i=100，则a=314,b=56，此时百位为1的次数有a/10+1=32（最高两位0~31），每一次都包含100个连续的点，即共有(a/10+1)*100个点的百位为1；
(2)当i表示百位，且百位对应的数为1，如n=31156,i=100，则a=311,b=56，此时百位对应的就是1，则共有a/10(最高两位0-30)次是包含100个连续点，当最高两位为31（即a=311），本次只对应局部点00~56，共b+1次，所有点加起来共有（a/10*100）+(b+1)，这些点百位对应为1；
(3)当i表示百位，且百位对应的数为0,如n=31056,i=100，则a=310,b=56，此时百位为1的次数有a/10=31（最高两位0~30）；
综合以上三种情况，当百位对应0或>=2时，有(a+8)/10次包含所有100个点，还有当百位为1(a%10==1)，需要增加局部点b+1，之所以补8，是因为当百位为0，则a/10==(a+8)/10，当百位>=2，补8会产生进位位，效果等同于(a/10+1)。
PS：求的是各位上面的1的出现次数，不需要担心重复，比如就算1100在百位和千位时均检测了一次，但是在百位时算的是百位上1出现的次数，在千位时算的是千位上1出现的次数，互不影响，况且1100中有两个1，本就应该计数两次。

032-把数组排成最小的数

标签: 数组（字符串）

思路：

(1)根据题目的要求，两个数字m和n能拼接成数字mn和nm。如果mn<nm，那么m应该排在n的前面，反之则n排在m前面。在拼接数字时需要考虑到一个潜在的问题就是m和n都在int能表达的范围内，但把它们拼起来的数字mn和nm用int表示就有可能溢出了，所以这还是一个隐形的大数问题。一个非常直观的解决大数问题的方法就是把数字转换成字符串之后再进行拼接。按照上述规律来对原数组进行排序；

(2)将排序好的数组中的元素依次添加到结果字符串上。

033-丑数

标签: 数学

思路:

如果p是丑数，那么p=2^x*3^y*5^z,那么只要赋予x,y,z不同的值就能得到不同的丑数。如果要顺序找出丑数，要知道下面几个特点。
对于任何丑数p：
(1)那么2*p,3*p,5*p都是丑数，并且2*p<3*p<5*p
(2)如果p<q,那么2*p<2*q,3*p<3*q,5*p<5*q
现在来看算法思想：
由于1是最小的丑数，那么从1开始，把2*1，3*1，5*1，进行比较，得出最小的就是1的下一个丑数，也就是2*1， 这个时候，多了一个丑数'2'，也就又多了3个可以比较的丑数，2*2，3*2，5*2，这个时候就把之前'1'生成的丑数和'2'生成的丑数加进来也就是(3*1,5*1,2*2，3*2，5*2)进行比较，找出最小的。如此循环下去就会发现，每次选进来一个丑数，该丑数又会生成3个新的丑数加入比较队列。
上面的暴力方法也应该能解决，下面说一个O(n)的算法:
在上面的特点中，既然有p<q， 那么2*p<2*q，那么越小的数乘以2,3,5得到的值越小，再在其中选出最小的值就是下一个丑数。其实每次我们只用比较3个值：乘2的最小的数*2、乘3的最小的数*3，乘5的最小的数*5。也就是比较2*x,3*y,5*z(x>=y>=z)就可以了。

034-第一个只出现一次的字符

标签:　　数组（字符串）

思路：

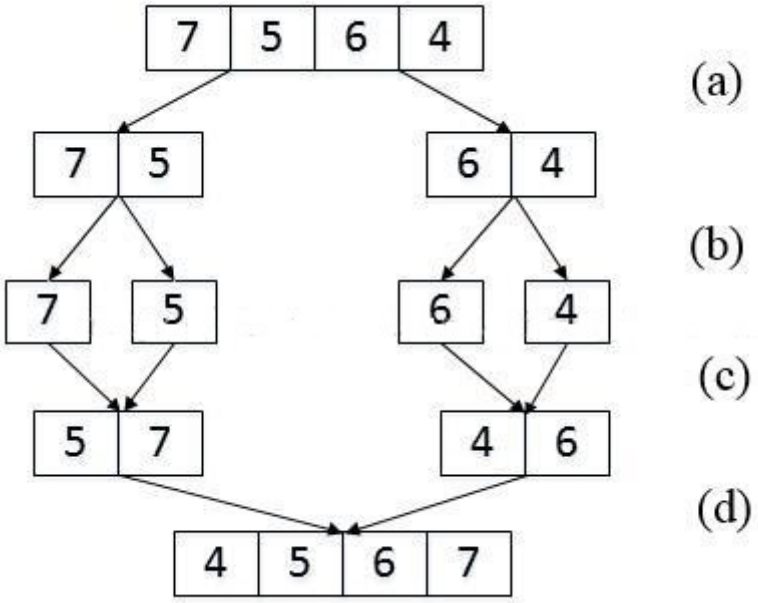
(1)遍历字符串并记录每个字符出现次数；
(2)再次遍历字符串，每遍历一个字符就去查看其出现次数，若出现次数为1那么直接返回该字符下标，否则继续向后遍历直到字符串被遍历完返回-1。

035-数组中的逆序对

标签: 排序与搜索

思路:

基于归并排序思想求逆序对个数。 先把数组分割成子数组，先统计出子数组内部的逆序对的数目，然后再统计出两个相邻子数组之间的逆序对的数目。在统计逆序对的过程中，还需要对数组进行排序，以免在以后的统计过程中再重复统计。



PS：

```
                                  形参：原数组          拷贝数组
     7 5 6 4                            data              copy
     /     \
   7 5      6 4                         copy              data
  /   \     /   \
 7     5   6     4                       copy              data
```

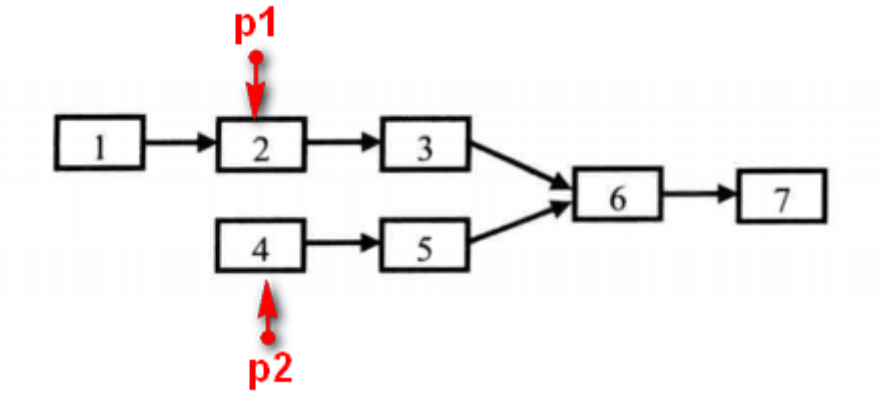
在每一层处理中，形参data位置数组是前半部分元素和后半部分元素均已经有序(但整个数组不是有序)的数组，形参copy位置是要保存本层的前后两段data的排序后的情况的辅助数组，从而构成一个范围更大前半部分元素和后半部分元素均已经有序的数组。在递归过程中一直交换data位置数组和copy位置数组，是为了使下一层的copy是上一层的data，是为了使下一层的data是上一层的copy，这样的话保证了每一层形参data位置数组是一个前半部分元素和后半部分元素均已 经有序的数组，这样可以省略了常规归并排序中还需要将每层分段排序好的数组再赋给原数组的操作。

036-两个链表的第一个公共结点

标签: 链表

思路:

(1) 先分别得到两个链表的长度;
(2) 如果两链表长度不相等，那么就要将长链表的指针移动到剩余节点长度和短链表相同的节点处，因为当两个链表的长度不一样长时，若将两个指针均指向两个链表头结点，那么这两个指针永远不会相遇，指向长链表的指针永远在指向短链表指针的后面。所以要先将两个指针对齐;
(3) 将p1和p2同时移动，判断两指针是否会相遇。



037-数字在排序数组中出现的次数

标签: 排序与搜索

思路:

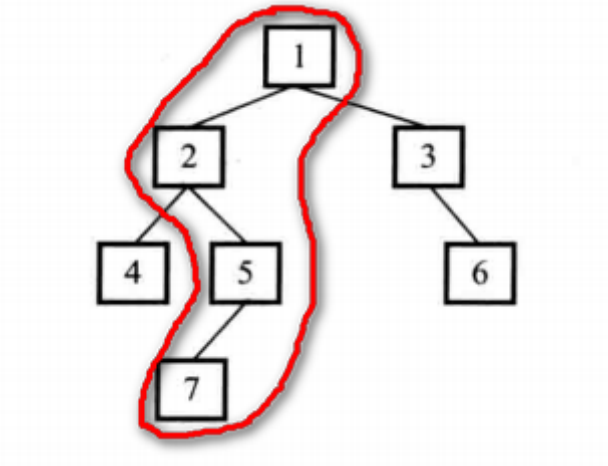
(1)使用二分查找找到第一个k的位置，即在找到了一个k的位置之后继续向前查找是否还有k，如果找到了k则执行(2)，否则直接返回0;
(2)继续使用二分查找找到最后一个k的位置，即在找到了一个k的位置之后继续向后查找是否还有k;
(3)因为数组有序，所以第一个k到最后一个k之间全部都是k，就可以得到k出现次数。

038-二叉树的深度

标签: 二叉树

思路:

(1) 首先分别得到左右子树的长度;
(2) 之后选取左右子树长度中的最大值再加上根节点就是树的深度。



039-平衡二叉树

标签: 二叉树

思路：

(1)首先分别得到每个节点左右子树的长度；
(2)之后判断左右子树的高度差，如果高度差大于1则返回false，反之返回true。

040-数组中只出现一次的数字

标签: 位运算

思路:

位运算中异或的性质：两个相同数字异或等于0，一个数和0异或还是它本身：
(1)首先对数组中所有元素进行异或，得到两个只出现一次的数（a,b）异或的结果；
(2)所得结果的二进制中的1表示的是a和b的不同的位。找到所得结果中从右向左第一个1所在的位数n，之后把原数组分成两组，分组标准是第n位是否为1；
(3)分组之后相同的数肯定在一个组，因为相同数字所有位都相同，而不同的数，肯定不在一组。然后依次异或这两个组，得到的两个异或结果就是a和b。

举个例子，假设输入数组｛2，4，3，6，3，2，5，5｝。
(1)当我们依次对数组中的每一个数字做异或运算之后，得到的结果用二进制表示是0010；
(2)异或得到结果中的倒数第二位是1,于是我们根据数字的倒数第二位是不是1分为两个数组。第一个子数组（2，3，6，3，2｝中所有数字的倒数第二位都是1,而第
二个子数组｛4，5，5｝中所有数字的倒数第二位都是0；
(3)接下来只要分别对这两个子数组求异或，就能找出第一个子数组中只出现一次的数字是6,而第二个子数组中只出现一次的数字是4。

041-和为S的连续正数序列

标签: 数组（字符串）

思路：

(1)用两个指针l和r分别表示滑动窗口的的最大值和最小值，首先将l初始化为1，r初始化为2；
(2)如果从l到r的和大于sum，我们就从序列中去掉较小的值(即增大l)；如果从l到r的和小于sum，只需要再向序列中增加一个较大的数(即增大r)；如果从l到r的和等于sum，则将该序列保存进结果集，之后序列向后移动(即增大l)；
(3)重复上述过程直到l与r重合。

042-和为S的两个数字

标签: 数组（字符串）

思路：

(1)使用双指针，l指向第一个元素，r指向第二个元素；
(2)若指针指向元素的和小于sum，说明太小了，l右移寻找更大的数；如果和大于sum，说明太大了，r左移寻找更小的数； 若和相等，把l和r指向的数返回。
PS：
证明两个数相差越大乘积越小找到两组满足条件的数组对（x，y）（x+a,y-a），其中（x+y=S, 0<a<y-x）
x*y-[(x+a)(y-a)]
=x*y-x*y-(y-x)a+a^2
=a[a-(y-x)]
因为0<a<y-x,所以a-(y-x)<0,所以a[a-(y-x)]<0，因此(x,y)乘积一定比(x+a,y-a)小。

043-左旋转字符串

标签: 数组（字符串）

思路：

假设字符串abcdef，n=3，设X=abc，Y=def，所以字符串可以表示成XY，如题干，问如何求得YX。假设X的翻转为X.T，X.T=cba，同理Y.T=fed，那么YX=(X.T Y.T).T，三次翻转后可得结果。

044-翻转单词顺序列

标签: 数组（字符串）

思路：

(1)首先将整个句子进行翻转，但是此时每个单词也是翻转的；
(2)之后将每个单词再次翻转成正常顺序，将每两个空格之间的单词逆置，将最后一个单词单独处理，因为最后一个单词不在两个空格之间。 这样每个单词在翻转字符串中就是一个正常的顺序。

045-扑克牌顺子

标签: 模拟

思路：

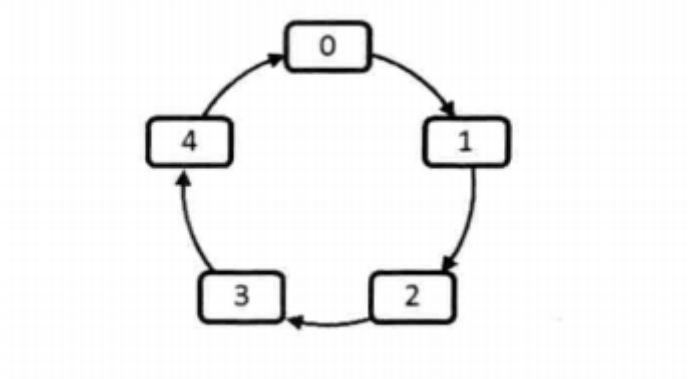
(1)先对数组进行排序；
(2)之后得到大小王的个数；
(3)最后遍历除大小王之外的其他牌，看看是否有空缺的牌，所有的空缺位置能否用大小王补上，如果能则返回true，不能则返回false；如果有相同牌说明不能构成顺子，直接返回false。

046-孩子们的游戏(圆圈中最后剩下的数)

标签: 模拟, 数学

模拟思路:

用一个链表(因为便于删除节点)模拟整个报数过程：在这n个数字中，经过报数之后被删除的数字是从当前指向节点节点向后数(m-1)%n次得到的数字，然后在链表中按照上述顺序依次删除对应节点，直到链表删到只剩下一个人。



数学思路:

约瑟夫环公式(丢手绢问题)。

$$f(n,m)=\begin{cases}0 & n=1\\ [f(n-1,m)+m]\%n & n>1\end{cases}$$

047-求1+2+3+...+n

标签: 递归

思路：

利用短路&&来实现if的功能；利用递归来实现循环while的功能。

048-不用加减乘除做加法

标签: 位运算

思路:

二进制的竖式计算中存在两个逻辑，分别是进位和合并。进位指对列同为1的执行逢二进一，合并指对列不同数值的执行为1。以上逻辑将分别用二进制运算来取代，前者可以视为按位与(&)和左移一位，后者可以视为两数执行异或。

```
    1011
  + 1001
---------
   00010  // 先 1011 ^ 1001 合并得到未进位的结果（ 1011 ^ 1001 = 00010 ）
   10010  // 1011 & 1001 = 1001 不为 0 存在进位，故执行 1001 << 1 得到进位后的结果 1001
---------
   10000  // 通过 00010 ^ 10010 合并得到未进位的结果（ 00010 ^ 10010 = 10000 ）
   00100  // 00010 & 10010 = 00010 不为 0 存在进位，故执行 00010 << 1 得到进位后的结果 00100
---------
   10100  // 通过 10000 ^ 00100 合并得到未进位的结果（ 10000 ^ 00100 = 10100 ）
   00000  // 10000 & 00100 = 00000 ，此时结束运算。
---------
```

PS：减法位运算实现->两个数num1，num2，num1-num2=num1+(-num2)；对于一个数的负数在计算机中时取反再加1，即m_num2=add(~num2,1)；然后add(num1,m_num2)。

049-把字符串转换成整数

标签: 数组（字符串）

思路：

(1)首先要根据第一个字符来确定数值绝对值的最大值，正数绝对值的最大值是2147483647，负数绝对值的最大值是2147483648；
(2)之后遍历字符串，如果遇到非数字字符，直接返回0；否则先判断新得到的数字是否溢出，若溢出则返回0，不溢出则更新res；
(3)最后如果str[0]为'-'，那么要对res乘以-1。

050-数组中重复的数字

标签: 数组（字符串）

思路：

不需要额外的数组或者hash table来保存，题目里写了数组里数字的范围保证在0 ~ n-1 之间，所以可以利用现有数组设置标志，如numbers[3]表示3是否有重复的数字：

(1)当一个数字被访问过后，可以设置以该数字为下标的位上的数+n；

(2)之后再遇到相同的数时，会发现以该数字为下标的位位上的数已经大于等于n了，那么直接返回这个数即可。

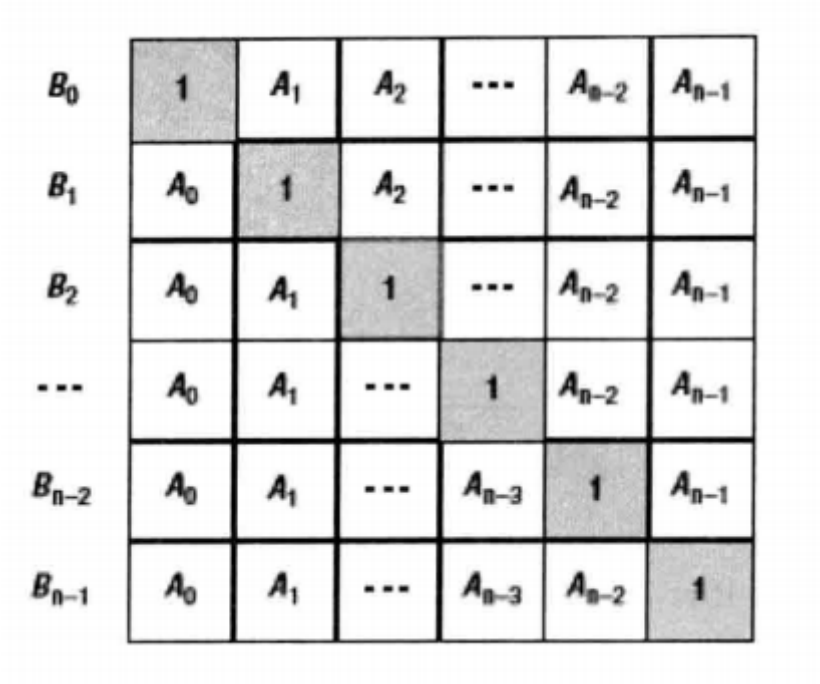
051-构建乘积数组

标签: 数组（字符串）

思路：

(1)由下图可知，res[i]就是每一行的乘积，最简单的方法时用两重循环求每一行的值。

(2)换一种思路来看，先算上三角中的连乘，即我们先算出res[i]中的一部分，然后倒过来按下三角中的分布规律，把另一部分也乘进去。



B_0	1	A_1	A_2	---	A_{n-2}	A_{n-1}
B_1	A_0	1	A_2	---	A_{n-2}	A_{n-1}
B_2	A_0	A_1	1	---	A_{n-2}	A_{n-1}
---	A_0	A_1	---	1	A_{n-2}	A_{n-1}
B_{n-2}	A_0	A_1	---	A_{n-3}	1	A_{n-1}
B_{n-1}	A_0	A_1	---	A_{n-3}	A_{n-2}	1

052-正则表达式匹配

标签: 递归

思路:

首先，考虑特殊情况：
(1)两个字符串都为空，返回true；
(2)当匹配字符串不空，而正则表达式字符串空了，返回false（因为这样，就无法匹配成功了,而如果第一个字符串空了，第二个字符串非空，还是可能匹配成功的，比如第二个字符串是"a*a*a*a*",由于'*'之前的元素可以出现0次，所以有可能匹配成功）。
之后就开始匹配字符，这里有两种可能pattern下一个字符为'*'或不为'*':
(1)pattern下一个字符不为'*'：这种情况比较简单，直接匹配当前字符。如果匹配成功，继续匹配下一个；如果匹配失败，直接返回false。注意这里的“匹配成功”，除了两个字符相同的情况外，还有一种情况，就是pattern的当前字符为'.',同时str的当前字符不为'\0'。
(2)pattern下一个字符为'*'时，稍微复杂一些，因为'*'可以代表0个或多个。这里把这些情况都考虑到：
a>当'*'匹配0个字符时，str当前字符不变，pattern当前字符后移两位，跳过这个'*'符号；匹配不到按匹配0个处理；
b>当'*'匹配1个或多个时，str当前字符移向下一个，pattern当前字符不变。(这里匹配1个或多个可以看成一种情况，因为当匹配一个时，由于str移到了下一个字符，而pattern字符不变，就回到了上边的情况a；当匹配多于一个字符时，相当于从str的下一个字符继续开始匹配)。

053-表示数值的字符串

标签: 数组（字符串）

思路：

将整个字符串划分为两部分来判断：'e'之前的数字段，'e'之后的数字段；
(1)首先找到字符串中'e'第一次出现的位置，如果'e'出现在字符串的开头或结尾直接返回false；
(2)根据'e'的位置先判断'e'之前的数字段，这个数字段可以是小数，可以是整数，所以在前半段只能出现'+'，'-'，'.'，'0'到'9'；'+'或者'-'只能出现在前半段的开头，否则返回false；'.'只能出现在前半段的开头(类似.09是正确写法)和数字之间，也只能出现一次，否则返回false；其余字符不能超过'0'到'9'这个范围；
(3)根据'e'的位置先判断'e'之后的数字段，这个数字段只能是整数，在前半段只能出现'+'，'-'，'0'到'9'，不能出现'.'；'+'或者'-'只能出现在后半段的开头，否则返回false;其余字符不能超过'0'到'9'这个范围。
PS：如果字符串不存在'e'或者'E'，那么直接判断整个字符串，这个过程与判断'e'之前的数字段相同。

054-字符流中第一个不重复的字符

标签: 数组（字符串）

思路：

(1)插入字符：插入字符时，将字符插入字符串中，之后将其出现次数加一；
(2)寻找字符：遍历字符串中的每一个字符，如果字符出现次数为1则返回该字符，如果不存在出现次数为1的字符则直接返回'#'。

055-链表中环的入口结点

标签: 链表

思路:

(1)先判断链表中是否有环:使用快慢指针，如果快慢指针相遇则有环，如果快指针最终指向NULL则说明没有环；
(2)如果有环，则在链表头与相遇点分别设一个指针，每次各走一步，两个指针必定相遇，且相遇第一点为环入口点。

056-删除链表中重复的结点

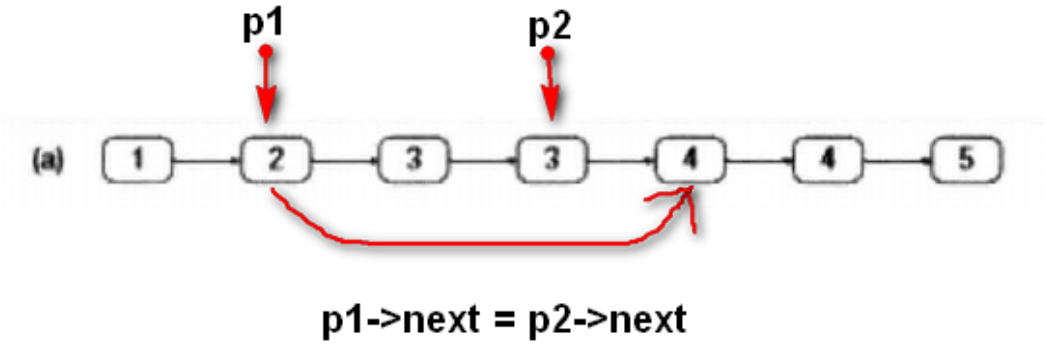
标签: 链表

思路:

先为原始链表生成一个空的头结点，便于删除重复节点，之后用p1指向当前节点node的前一个节点，p2用于寻找和当前节点node相等的节点：

(1)如果当前节点node不为重复节点，那么将p1指向node，继续进行判断；

(2)如果当前节点node为重复节点，那么将p2指针指向后面链表中最后一个与node相等的节点，之后将p1->next指向p2->next(相当于删除这些节点)，继续判断；重复上述判断过程直到判断完整条链表。



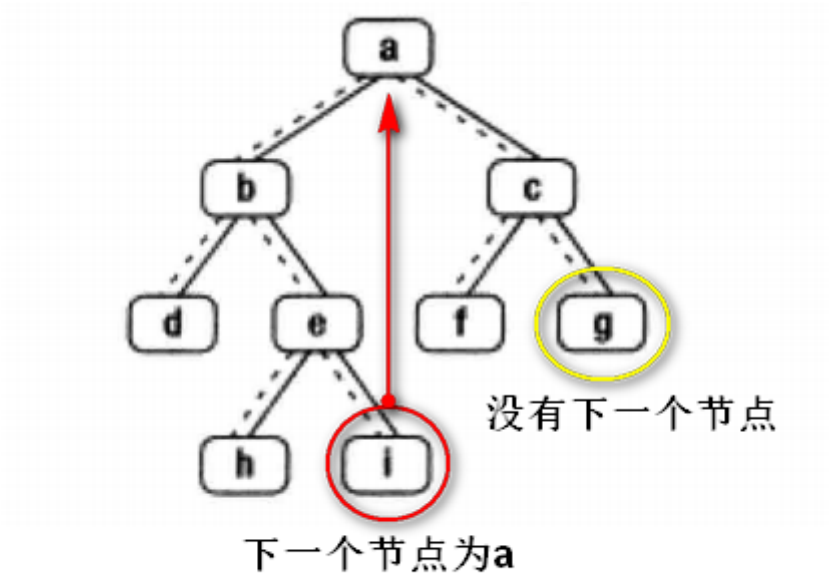
p1->next = p2->next

057-二叉树的下一个结点

标签: 二叉树

思路:

分为以下几种情况处理:
1. pNode节点有右子树，那么pNode的下一个节点为其右子树上最左侧的节点。
2. pNode节点没有右子树，又分为以下几种情况:
(1)pNode为整棵树的根节点，此时没有下一个节点;
(2)pNode为其父节点的左子树，那么pNode下一个节点为其父节点;
(3)pNode为其父节点的右子树，则向上寻找，看pNode是否在某个节点的左子树上，如果存在这样一个节点那么pNode的下一个节点为该节点，否则，pNode没有下一个节点(在整棵树的最右侧，所以其在中序序列中没有下一个节点)。

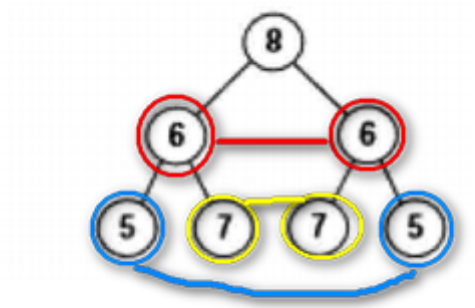


058-对称的二叉树

标签: 二叉树

思路:

(1)首先判断根节点是否为空节点，如果为空节点直接返回true，不为空则判断其左右节点是否对称；
(2)若两节点均为空则直接返回true；若两节点只有一个节点为空或者两节点的值不相等，则直接返回false；
(3)如果判断两个节点相等，则将root1的左节点和root2的右节点比较，root1的右节点和root2的左节点比较。这样就相当于一直将每层第一个节点和最后一个节点，第二个节点和倒数第二个节点等等相对应的节点进行比较；
(4)重复(2)(3)的操作，直到遇到不对称节点或者将整个树对应节点比较完。

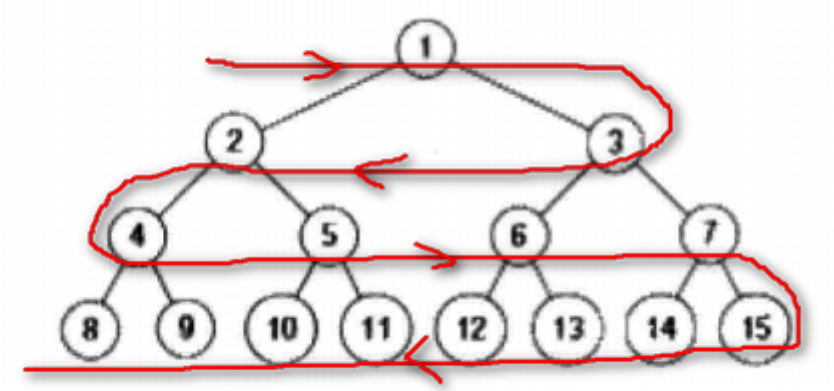


059-按之字形顺序打印二叉树

标签: 二叉树

思路:

(1) 层次遍历给定的树，之后按照从左到右的顺序保存各层节点；
(2) 每遍历一层的节点之后，判断这层为奇数层还是偶数层。如果为奇数层按照从左往右的顺序放入结果集；为偶数层则按照从右往左的顺序放入结果集。



060-把二叉树打印成多行

标签: 二叉树

思路：

层次遍历二叉树，并按照顺序保存每层节点的值：
(1)每次遍历完上一层节点之后，队列中剩余的节点数count，就是这一层节点数；
(2)从队头取出最早进入队列的结点，将其保存在结果集中；如果该结点有子结点，则将其子节点(如果左右子节点均存在，则按照先左子节点后右子节点的顺序)入队；
(3)继续执行(2)count次将该层节点全部放入结果集中；
重复上述操作，直到队列为空，即每一层的节点都被遍历完。

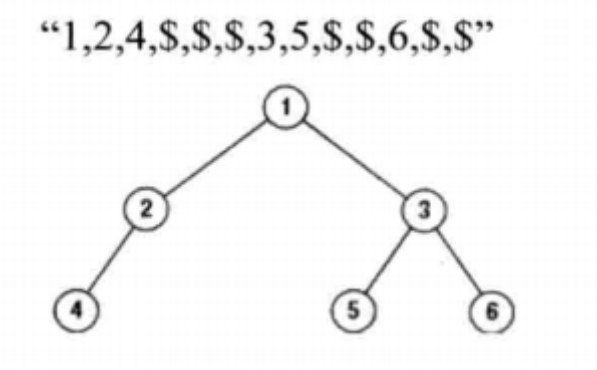
061-序列化二叉树

标签: 二叉树

思路:

(1)序列化二叉树:采用前序遍历的方法遍历二叉树，并将二叉树转化为字符序列;

(2)反序列化二叉树:再根据前序遍历的顺序来解析字符串。



062-二叉搜索树的第k个结点

标签: 二叉树

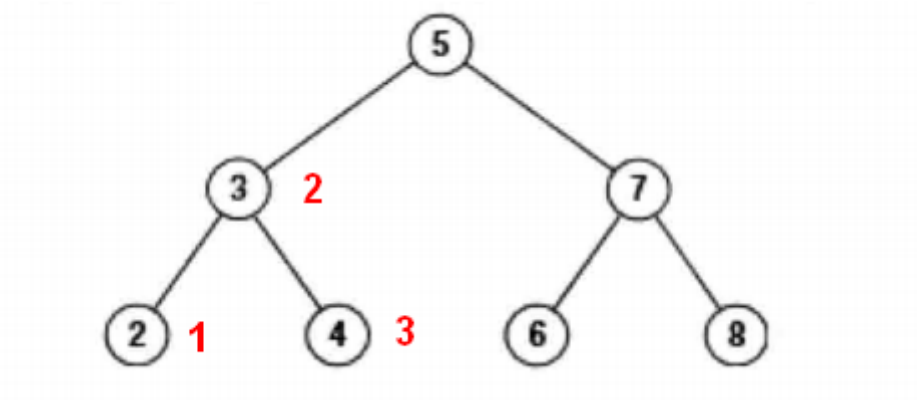
思路:

根据二叉搜索树的特点，中序遍历二叉搜索树得到的序列即为从小到大排列的有序序列:

(1) 中序遍历二叉搜索树，并在遍历过程中记录遍历过节点的个数;

(2) 当遍历到第k个节点时，该节点为第k小的节点，返回该节点即可。

例如在下面这颗树中找第三小的节点:



063-数据流中的中位数

标签: 堆

思路:

将数据流中的数字用一个大顶堆和一个小顶堆来保存，大顶堆用于保存较小的前一半数，堆顶是前半部分最大的数；小顶堆用于保存较大的后一半数，堆顶是后半部分最小的数。如果数据流中的数据总数为偶数，那么大顶堆和小顶堆中的数据量相等；如果数据流中的数据总数为奇数，那么大顶堆比小顶堆中的数据量大1：

(1)插入数据时，如果插入数据小于等于前半部分数据的最大值，那么将其插入前半部分，否则插入后半部分；在插入数据完成之后，要对两个堆进行调整，使得两个堆数据量相等(偶数情况)或者大顶堆比小顶堆中的数据量大1(奇数情况)；

(2)求中位数时，如果数据总量为偶数，那么中位数就是两堆顶的平均数；如果数据总量为奇数，那么中位数就小顶堆的堆顶，因为在奇数情况下，大顶堆比小顶堆中的数据量大1，那么中位数存在于大顶堆之中。

064-滑动窗口的最大值

标签: 队列

思路:

滑动窗口应当是队列，但为了得到滑动窗口的最大值，队列应当可以从两端删除元素，因此使用双端队列：
对新来的元素k，将其与双端队列中的元素相比较
(1)前面比k小的，直接移出队列(因为不再可能成为后面滑动窗口的最大值了)。
(2)前面比k大的X，比较两者下标，判断X是否已不在窗口之内，不在了，直接移出队列。
(3)将k入队(因为k可能成为后面滑动窗口中的最大值)。
队列的第一个元素始终是滑动窗口中的最大值。

065-矩阵中的路径

标签: 图

思路:

(1)遍历矩阵，寻找和目标字符串第一个字符相等的格子，找到之后执行(2)；
(2)根据找到的格子的坐标向四周搜索，搜索过程中要对搜索过的格子做标记，否则会出现重复搜索一个格子的情况。如果可以搜索到一条包含目标字符串的路径则直接返回true；反之则返回false，将路径上格子被遍历过的标记清除，并继续执行(1)。
重复上述过程直到找到一条正确的路径或者将所有格子都遍历完。

066-机器人的运动范围

标签: 图

思路:

(1)从起点向四周搜索节点。如果节点坐标没有越界，坐标各位之和小于目标值，且该坐标没有被遍历过，那么该坐标为可到达坐标。之后将该坐标记录为已遍历坐标。
(2)重复上述过程直到遍历完全部可到达坐标。

067-剪绳子

标签: 动态规划

思路:

(1)用res[i]代表长度为i时的最大乘积，长度为0和1时最大乘积为1;
(2)长度为i时的最大乘积(res[i])等于长度为i-j(j<=i)时的最大乘积(res[i-j])与所剪去的长度j之积。由此可得递推公式：res[i] = max(res[i],res[i-j]*j)。但是要注意一点除了目标长度之外其余长度都可以一刀不剪。