In [7]:

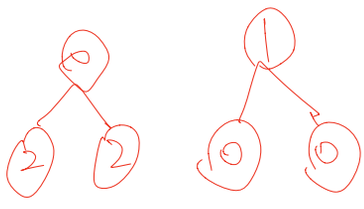
```python
# 455 饼干
def findContentChildren(g, s):
    # 大饼干喂饱大胃口先
    g.sort()  # 胃口
    s.sort()  # 饼干
    gpoint = len(g)-1
    spoint = len(s)-1
    res = 0
    while gpoint >=0 and spoint>=0:
        # 不能while gpoint and spoint 这样会忽略gpoint=0的情况
        if g[gpoint] <= s[spoint]:
            res += 1
            spoint -= 1
        gpoint -= 1
    return res

# 376 摆动序列
def wiggleMaxLength(nums):
    curDiff = 0 # 右-中
    preDiff = 0 # 左-中
    res = 1
    for i in range(len(nums)-1):
        curDiff = nums[i+1]-nums[i]
        if curDiff * preDiff <= 0 and curDiff != 0:
            res += 1
            preDiff = curDiff
    return res

# 53 最大子序和
def maxSubArray(nums):
    res = float('-inf')
    cur = 0
    for i in range(len(nums)):
        cur += nums[i]
        if cur <= nums[i]: cur = nums[i]  # 加进去反而变小了，重新开始
        if cur > res: res = cur
    return res

# 122 买卖股票II
def maxProfit(prices):
    # 捕获上坡那一段即可
    res = 0
    for i in range(1,len(prices)):
        res += max(prices[i]-prices[i-1],0)
    return res

# 55 跳跃游戏I
def canJump(nums):
    cover = 0
    for i in range(len(nums)):
        if i > cover : return False # 出现断层
        cover = max(nums[i]+i,cover)
        if cover > len(nums): return True # 早停
    return True

# 45 跳跃游戏II
def jump(nums):
```

```
    end = 0
    cover = 0
    res = 0
    for i in range(len(nums)-1):
        # 记得减1,实际上,cover会比end快到达终点,end会比i先到达终点
        cover = max(nums[i]+i,cover)
        # if end == len(nums)-1: return res
        if cover >= len(nums)-1: return res+1 # 少走了一步
        if i == end:
            end = cover
            res += 1
    return res

# 1005 k次反后最大化数组
def largestSumAfterKNegations(nums, k):
    nums.sort()
    for i in range(len(nums)):
        # 处理完所有负数
        if nums[i] < 0 and k > 0:
            nums[i] *= -1
            k -= 1
        else:
            break
    if k > 0:
        nums.sort()
        nums[0] *= (-1)**k
    return sum(nums)

# 134 加油站
def canCompleteCircuit(gas, cost):
    # 暴利求解 index = (index + 1) % n 适合环形遍历
    n = len(gas)
    for i in range(n):
        res = gas[i]-cost[i]
        index = (i+1)%n
        while res > 0 and index != i:
            res += gas[index]-cost[index]
            index = (index+1) % n
        if res >= 0 and index == i:
            return i
    return -1

    # 贪心算法
    if sum(gas) < sum(cost) : return -1  # 不可能跑一圈
    n = len(gas)
    curSum = 0
    start = 0
    for i in range(n):
        curSum += gas[i]-cost[i]
        if curSum < 0:
            curSum = 0  # 突然不够油了
            start = i+1 # 重新开始
    return start

# 435 无重叠子区间
def eraseOverlapIntervals(intervals):
    n = len(intervals)
    intervals.sort(key=lambda x:x[1])
    count = 1  # 记录没有交集的区间数
    end = intervals[0][1]
    for i in intervals:
```

```
        if i[0] >= end:
            count += 1
            end = i[1]
    return n-count # 去掉没有交集的就是有重合的

# 986 区间列表的交集
def intervalIntersection(firstList, secondList):
    i,j = 0,0
    n,m = len(firstList),len(secondList)
    res = []
    while i < n and j < m:
        # 无交集的情况取反 a2 < b1 or a1 > b2
        if firstList[i][1] >= secondList[j][0] and firstList[i][0] <= secondList[j][1]:
            res.append([max(firstList[i][0],secondList[j][0]),min(firstList[i][1],secondList[j]
        if secondList[j][1] > firstList[i][1]: i += 1
        else: j += 1
    return res

# 135 分发糖果
def candy(ratings):
    # 需要一个存放糖果nums
    n = len(ratings)
    nums = [1]*n
    # 从前往后确定i值大于i-1
    for i in range(1,n):
        if ratings[i] > ratings[i-1]: nums[i] = nums[i-1]+1 # 比前面的大1
    # 从后往前确定i-1大于i的值
    for i in range(n-2,-1,-1):
        if ratings[i] > ratings[i+1]: nums[i] = max(nums[i],nums[i+1] + 1) # 取两边都满足的最
    return sum(nums)

# 860 柠檬水找零
def lemonadeChange(bills):
    five,ten,twenty = 0,0,0
    for bill in bills:
        if bill == 5: five += 1
        if bill == 10:
            if five <= 0: return False
            ten += 1
            five -= 1
        if bill == 20:
            if ten >= 1 and five > 0:
                twenty += 1
                ten -= 1
                five -= 1
            elif  five >= 3:
                twenty += 1
                five -= 3
            else: return False
    return True

# 406 身高排序
def reconstructQueue(people):
    # 优先按身高高的people的k来插入，后序插入节点也不会影响前面已经插入的节点
    queue = []
    people.sort(key=lambda x:(-x[0],x[1])) # 一维降序，二维升序，（）中谁先先排谁
    for i in people:
        if i[1] > len(queue): queue.append(i)
        else:
            queue.insert(i[1],i)
    return queue
```

```
180
181 # 452 最少的气球——按end排序
182 def findMinArrowShots(points):
183     # 求区间的交集,[1,2],[2,3]不相交但是count += 1
184     # 重叠的情况很多种，求不重叠情况！ start > end
185     points.sort(key=lambda x:x[1])
186     count = 1
187     end = points[0][1]
188     for i in points:
189         start = i[0]
190         if start > end :
191             count += 1
192             end = i[1]
193     return count
194
195 # 763 划分字母区间
196 def partitionLabels(s):
197     curfarther = 0
198     s = list(s)
199     res = []
200     j = -1
201     dic = {s[i]:s.index(s[i],i) for i in range(len(s))}  # 记录最远的位置
202     for i in range(len(s)):
203         curfarther = max(curfarther,dic[s[i]])
204         if  i == curfarther:
205             res.append(i-j)
206             j = i
207     return res
208
209 # 56 区间合并——按start排序
210 def merge(intervals):
211     # 融合区间
212     intervals.sort(key=lambda x:x[0])
213     res = []
214     res.append(intervals[0])
215     for interval in intervals:
216         start = interval[0]
217         if start <= res[-1][1]:  # 衡量标准，最大end
218             res[-1][1] = max(interval[1],res[-1][1])
219         else:
220             res.append(interval)
221     return res
222
223 # 738 单调递增数字
224 def monotoneIncreasingDigits(n):
225     # 单调递增的数字
226     # 从后向前遍历
227     nums = [int(i) for i in str(n)]
228     length = len(nums)
229     for i in range(length-2,-1,-1):
230         if nums[i] > nums[i+1]:
231             nums[i] -= 1
232             nums[i+1:] = '9'*len(nums[i+1:]) # 后面赋值为9
233     nums = [str(i) for i in nums]
234     return int(''.join(nums))
235
236 # 968 监控二叉树
237 def minCameraCover(root):
238     # 0 表示无摄像头+有覆盖
239     # 1 表示摄像头
240     # 2 表示有覆盖
```





```
241     self.res = 0
242     def traversal(cur):
243         if not cur: return 2  # 标记为2
244         left = traversal(cur.left)
245         right = traversal(cur.right)
246         if left == 2 and right == 2:
247             # 叶子节点,左右节点都覆盖，那么这个个应该什么都没有或者是叶子
248             return 0
249         elif left == 0 or right == 0:
250             # 有一个孩子没有覆盖，父节点就应该放摄像头
251             self.res += 1
252             return 1
253         elif left == 1 or right == 1:
254             # 有一个孩子有摄像头了，父节点就应该被覆盖了
255             return 2
256         else: return
257     if traversal(root) == 0: self.res += 1  # 这里已经进行了方法修改，特殊情况
258     return self.res
```