做人不能太贪心

今日目标：

1：能够说出贪心算法是如何做选择的

2：完成实战面试题目

1、贪心算法

贪心算法（英语：greedy algorithm），又称**贪婪算法**，是一种在每一步选择中都采取在当前状态下最好或最优（即最有利）的选择，从而希望导致结果是最好或最优的算法。

> 贪心：当下做局部最优判断，不会回退
>
> 回溯：能够回退撤销选择

看这个算法的名字：贪心，贪婪，两个字的含义最关键，好像一个贪婪的人所有事情都只想到眼前，看不到长远，也不为最终的结果和将来着想，贪图眼前局部的利益最大化。

贪心算法可以解决一些最优化问题，比如：求图的最小生成树，求哈夫曼编码等等。然后对于工程和生活中的一些问题，贪心法一般不能得到我们想要的最终答案，因为每一步都找局部最优并不一定导致全局最优。

> 比如leetcode：322. 零钱兑换
>
> 场景1：假设当前可选硬币集合为：coins=[20，10，5，1]，求可以凑成总金额amount=36 所需的最少的硬币个数?
>
> 这时就可以用贪心法，每次都先选最大的面额

基于贪心的思路我们可以写出第一版代码：

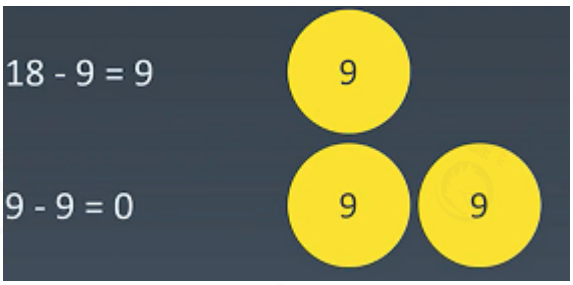
```java
class Solution {
    //贪心思路:每次都找局部最优解
    public int coinChange(int[] coins, int amount) {
        //特殊
        if (amount==0) {
            return 0;
        }
        //定义组合总数
        int count = 0;
        Arrays.sort(coins);
        for (int i=coins.length-1;i>=0;i--) {
            //计算该面额能用几次
            int currentCount = amount / coins[i];
            if (currentCount==0) {
                continue;
            }
            //该面额用完后剩余的面额
            amount = amount % (currentCount * coins[i]) ;
            //统计所用硬币个数
            count+= currentCount;

            if (amount ==0) {
                return count;
            }
        }
        return -1;
    }
}
```

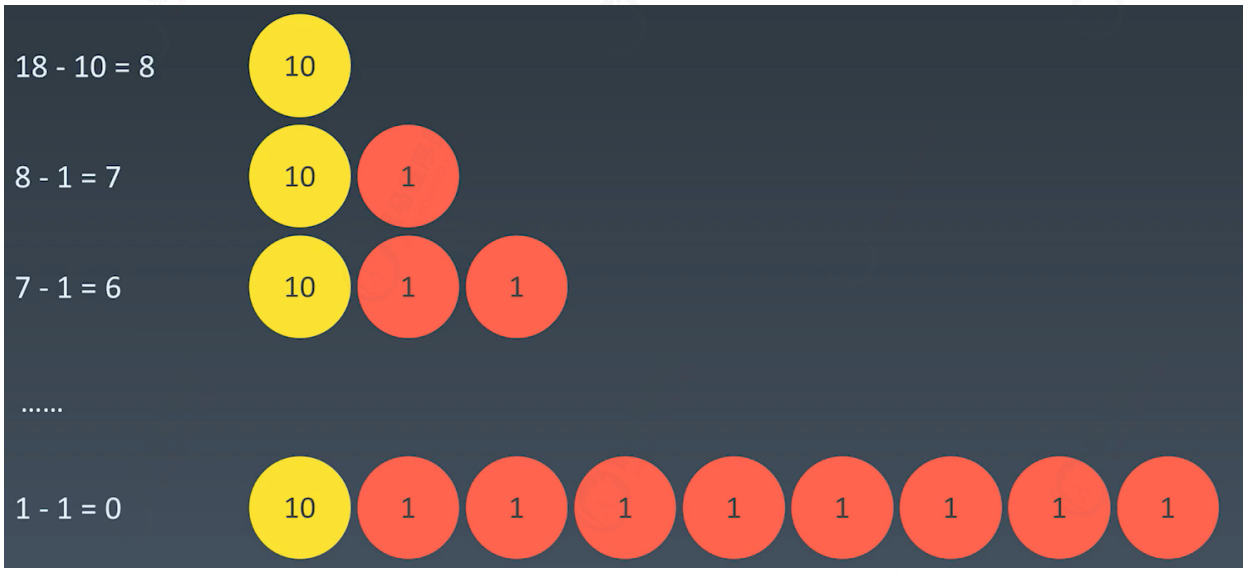
提交测试，有些测试用例能通过，但有些不行，譬如：输入如下测试用例

```
[10,9,1]
18
```

该场景下，真正的全局最优解应该是两个9，如下



但是如果用贪心算法来求解的话，应该是如下的情况：



显然这并不是最优解。

进阶：当贪心算法失效了怎么办？如何优化？

思路：可以通过回溯来解决“过于贪心”的问题，简单的说就是加入一个递归搜索过程，穷举所有结果，通过记录所有解中最小的那个得到最优解，即使用硬币数最少。

```
1：每种硬币有[0,maxCount]个选择，其中maxCount= amount /  coins[i]，可以看作是选择列表

2：从选择列表中选择一个，然后递归到下一个硬币，搜索所有组合结果

3：整个过程中记录组合结果中使用硬币最少的解，即为题目要求的解。

```

直接使用DFS回溯的话肯定是能够得到准确答案的，但是效率会非常低；因此在这里使用贪心+回溯相配合的方式，先尽可能的使用大面值的coins，如果此路不通则慢慢减少大面值coins的数量直至为0；这里还是用的一个剪枝的技巧：尽早的判断这条路走下去会不会可能是最优解。这里就是时刻判断：如果使用这种coin，则使用coins的总数量是不是已经超过了之前的方案，如果超过的话说明这条路即便是通了也不可能是最优解。

```
class Solution {

    public int coinChange(int[] coins, int amount) {
        //对coins从大到小排序
        sort(coins);
        dfs(coins,amount,0,0);
        if (minCount == Integer.MAX_VALUE) {
            return -1;
        }
        return minCount;
    }
```

```
    public void sort (int[] coins) {
        //为了方便先Arrays.sort,然后倒置
        Arrays.sort(coins);
        int n = coins.length-1;
        int temp;
        for (int i=0;i<=(n-1)/2;i++) {
            temp = coins[i];
            coins[i] = coins[n-i];
            coins[n-i] = temp;
        }
    }
    //全局最优解使用的最少硬币数量
    int minCount = Integer.MAX_VALUE;

    public void dfs (int[] coins,int amount,int selectedCount,int startIndex) {

        //如果amount==0则终止
        if (amount==0) {
            if (selectedCount < minCount) {
                minCount = selectedCount;
            }
            return;
        }

        if (startIndex >= coins.length) {
            return;
        }

        //计算当前硬币最多能使用多少次
        int maxCount = amount / coins[startIndex];
        //从选择列表[0,maxCount]中选择,然后递归到下一个硬币
        for (int i=maxCount;i>=0 && i+selectedCount < minCount ;i--) {
            //选择i个该硬币后剩余的额度
            int resAmount = amount - i*coins[startIndex];
            //drill down
            dfs(coins,resAmount,selectedCount+i,startIndex+1);
            //这里无需显示的撤销选择,for循环继续走选择下一个就相当于在回溯了,我们无需记走过的路径和每种走法,我们只是在走的过程中记步数。
        }
    }

}
```

贪心算法的局限性：

贪心算法总是以**局部最优**来解决问题，只考虑“当前”的最大利益，既不向前多看一步，也不向后多看一步，导致每次都只用当前阶段的最优解，那么如果纯粹采用这种策略我们就永远无法达到**整体最优**，有时候也就无法求得题目最终的答案了。

虽然纯粹的贪心算法作用有限，但是这种求解局部最优的思路在方向上肯定是对的，毕竟所谓的整体最优肯定是从很多个局部最优中选择出来的，因此所有**最优化问题**的基础都是贪心算法。

贪心算法的适用场景：

一旦一个问题可以通过贪心算法来解决，那么贪心算法一般是解决这个问题的最好办法。因此一般遇到一个问题的时候重点是要能证明该问题能够用贪心算法来求解。

求最值等系列问题，而求最值的核心思想是**穷举**。这是因为只要我们能够找到所有可能的答案，从中挑选出最优的解就是算法问题的结果。在没有优化的情况下，穷举从来就不算是一个好方法。使用贪心算法来解题是一种使用**局部最优思想**解题的算法（即从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快的速度去求得更好的解，当达到算法中的某一步不能再继续前进时，算法停止）。

我们往往需要使用回溯（搜索）来优化贪心算法，否则就会导致算法失效。因此，在求解最值问题时，我们需要更好的方法来解。在后面课程讲到递归和穷举优化问题的时候，会讲到解决最值问题的正确思路和方法：**考虑整体最优**的问题。

由于贪心法的高效性以及所求得的答案比较接近最优结果，贪心法也可以用作辅助算法或者直接解决一些要求结果不是特别精确的问题。

2、面试实战

455. 分发饼干

字节跳动，亚马逊最近面试题，455. 分发饼干

```java
class Solution {
    //此处贪心的整体思路是：最小尺寸的饼干发给最小胃口的孩子，即保证：给每个孩子分配的饼干既能满足他的胃口又还不浪费饼干，
    public int findContentChildren(int[] g, int[] s) {
        int max = 0 ;
        //对胃口值和饼干尺寸排序，
        Arrays.sort(g);
        Arrays.sort(s);
        //定义两个数组下标-也是双指针思想的体现
        int i=0;
        int j=0;
        while (i<g.length && j<s.length) {
            if (s[j]>= g[i]) {//j饼干能满足i孩子,然后看下一块饼干和下一个孩子
                j++;
                i++;
                max++;
            }else {//j饼干不能满足i孩子，看下一块饼干是否能满足
                j++;
            }
        }
        return max;

    }
}
```

> 进阶：想想如何去论证贪心算法在该场景中，每次使用局部最优解能保证最终的最优解呢？
>
> 可参考题解：https://leetcode-cn.com/problems/assign-cookies/solution/tan-xin-jie-fa-by-cyc2018/ 的论述。

122. 买卖股票的最佳时机 II

亚马逊，字节跳动最近面试题，122. 买卖股票的最佳时机 II

股票买卖系列问题典型的解法是动态规划，但是针对这道题的特殊解法可以是用贪心。

> 可以参考精选题解：
>
> https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/solution/tan-xin-suan-fa-by-liweiwei1419-2/的方法三
>
> https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/solution/best-time-to-buy-and-sell-stock-ii-zhuan-hua-fa-ji/

```java
class Solution {
    //贪心算法来求解问题
    public int maxProfit(int[] prices) {
        int maxProfit = 0 ;
        if (prices.length < 2) {
            return maxProfit;
        }
        // 如果今天价格比前一天高，则在前一天买入,今天卖出
        for (int i=0;i<prices.length;i++) {
            if (i>0 && prices[i]>prices[i-1]) {
                maxProfit+= prices[i] - prices[i-1];
            }
        }
        return maxProfit;
    }
}
```

55. 跳跃游戏

字节，华为，腾讯最近面试题，55. 跳跃游戏

> **解题思路：**
>
> 1：如果某一个作为 起跳点 的格子可以跳跃的距离是 3，那么表示后面 3 个格子都可以作为 起跳点。
>
> 2：可以对每一个能作为 起跳点 的格子都尝试跳一次，把 **能跳到最远的距离** 不断更新。
>
> 3：如果可以一直跳到最后，就成功了。

```java
class Solution {
```

```
    public boolean canJump(int[] nums) {
        int farthest = 0;//跳跃的最远距离
        for (int i=0;i<nums.length;i++) {
            //farthest到某个位置后再走不下去了,[3,2,1,0,4]
            if (i>farthest) {
                return false;
            }

            farthest = Math.max(farthest,i+nums[i]);

            if (farthest >=nums.length-1) {
                return true;
            }
        }
        return false;
    }
}
```

45. 跳跃游戏 II

字节，小米，百度面试题，45. 跳跃游戏 II

典型的贪心算法，通过局部最优解得到全局最优解

> 请参考官方题解：https://leetcode-cn.com/problems/jump-game-ii/solution/tiao-yue-you-xi-ii-by-leetcode-solution/中的方法二

```
class Solution {
    public int jump(int[] nums) {
        //边界位置
        int boundary = 0;
        //最远位置的点
        int maxPoint = 0;
        //步数
        int stepCount = 0;

        /*
            在遍历数组时，我们不访问最后一个元素，这是因为在访问最后一个元素之前，我们的边界一定大于等于最后一个位置，
            否则就无法跳到最后一个位置了。
            如果访问最后一个元素，在边界正好为最后一个位置的情况下，我们会增加一次「不必要的跳跃次数」
        */
        for (int i=0;i<nums.length-1;i++) {

            //找最远位置的点

            maxPoint = Math.max(maxPoint,i+nums[i]);
```

```
19
20          //i走到边界后，更新边界并增加跳跃次数
21          if (i== boundary ) {
22              boundary  = maxPoint;
23              stepCount++;
24          }
25      }
26
27      return stepCount;
28  }
29  }
```

860. 柠檬水找零

华为半年面试题，860. 柠檬水找零

> 参照官方题解：https://leetcode-cn.com/problems/lemonade-change/solution/ning-meng-shui-zhao-ling-by-leetcode/

```
1   class Solution {
2       public boolean lemonadeChange(int[] bills) {
3           //定义5元，10元钞票的张数
4           int five,ten;
5           five = ten =0;
6           for (int bill:bills) {
7               if (bill == 5) {
8                   //不需找零，
9                   five ++;
10              }else if (bill == 10) {
11                  //需要找零5块,没有5块则找零失败
12                  if (five <=0) {
13                      return false;
14                  }
15                  five--;
16                  ten++;
17              }else {
18                  //需要找零15,按照贪心思想，先找面额大的组合,这样最有利 10+5
19                  if (ten>0 && five >0) {
20                      ten--;
21                      five--;
22                  }else if (five >=3) {// 如果没有10+5，找三个5块也行
23                      five -=3;
24                  }else {
25                      //找零失败
26                      return false;
27                  }
28              }
29          }
30          return true;
31      }
32  }
```