走出迷宫的人们，有的是认识路；有的是莽撞碰巧出来的；有的则是一路做着标记出来的；也有的是走遍了整个迷宫。   
——证明了的贪心算法、没有证明的贪心算法、动态规划、暴力搜索的区别。

**所有的贪心算法问题都能用DP求解，更可以归结为一个搜索问题，反之不成立**。

暴力搜索——记忆搜索——动态规划

相同点：  
动态规划和贪心算法都是一种递推算法   
均有局部最优解来推导全局最优解，均满足最优子结构和子问题重叠。  
  
不同点：

贪心：A最优+B最优  
动态规划：（A+B）最优  
就单步而言  
贪心的A最优是前一步的结果，B最优需要遍历找到  
动态规划的A为前一步的可行解，需要选择一个B后再去找A  
贪心算法：   
1.贪心算法中，作出的每步贪心决策都无法改变，因为贪心策略是由上一步的最优解推导下一步的最优解，而上一步之前的最优解则不作保留。   
2.由（1）中的介绍，可以知道贪心法正确的条件是：每一步的最优解一定包含上一步的最优解。

该算法存在问题：     
1.   不能保证求得的最后解是最佳的；   （需要先证明）  
2.   **不能用来求最大或最小解问题；**     
3.   只能求满足某些约束条件的可行解的范围。  
  
动态规划算法（基本原理：动态规划英文名dynamic programming。其中**pogramming指的是表格法**，而非编写计算机程序。因此，可以初步得出动态规划的基本思想：将一个具有最优子结构性质的问题分成若干个子问题，在求解过程中，记录下子问题的结果，存储在一个表格中，使得公共的子问题只需要计算一次。）：   
1.全局最优解中一定包含某个局部最优解，但**不一定包含前一个局部最优解**，因此需要记录之前的所有最优解 ；

2.动态规划的关键是状态转移方程，即如何由以求出的局部最优解来推导全局最优解   
3.边界条件：即最简单的，可以直接得出的局部最优解  
==============================================================================  
贪心算法与动态规划   
贪心法的基本思路：     
      
从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快的地求得更好的解。当达到某算法中的某一步不能再继续前进时，算法停止。     
该算法存在问题：     
1.   不能保证求得的最后解是最佳的；     
2.   不能用来求最大或最小解问题；     
3.   只能求满足某些约束条件的可行解的范围。实现该算法的过程：     
从问题的某一初始解出发；  
     
while   能朝给定总目标前进一步   do     
求出可行解的一个解元素；     
由所有解元素组合成问题的一个可行解   
  
贪心算法最经典的例子，给钱问题。     
比如中国的货币，只看元，有1元2元5元10元20、50、100     
      
如果我要16元，可以拿16个1元，8个2元，但是怎么最少呢？     
如果用贪心算，就是我每一次拿那张可能拿的最大的。     
比如16，我第一次拿20拿不起，拿10元，OK，剩下6元，再拿个5元，剩下1元     
也就是3张   10、5、1。     
      
每次拿能拿的最大的，就是贪心。     
      
但是一定注意，贪心得到的并不是最优解，也就是说用贪心不一定是拿的最少的张数     
贪心只能得到一个比较好的解，而且贪心算法很好想得到。     
再注意，为什么我们的钱可以用贪心呢？因为我们国家的钱的大小设计，正好可以使得贪心算法算出来的是最优解（一般是个国家的钱币都应该这么设计）。如果设计成别的样子情况就不同了     
比如某国的钱币分为   1元3元4元     
如果要拿6元钱   怎么拿？贪心的话   先拿4   再拿两个1     一共3张钱     
实际最优呢？   两张3元就够了     
  
  
  
**求最优解的问题，从根本上说是一种对解空间的遍历**。最直接的暴力分析容易得到，最优解的解空间通常都是以指数阶增长，因此暴力穷举都是不可行的。  
最优解问题大部分都可以拆分成一个个的子问题，把解空间的遍历视作对子问题树的遍历，则以某种形式对树整个的遍历一遍就可以求出最优解，如上面的分析，这是不可行的。  
**贪心和动态规划本质上是对子问题树的一种修剪。**两种算法要求问题都具有的一个性质就是“子问题最优性”。即，组成最优解的每一个子问题的解，对于这个子问题本身肯定也是最优的。如果以自顶向下的方向看问题树（原问题作根），则，我们每次只需要向下遍历代表最优解的子树就可以保证会得到整体的最优解。形象一点说，**可以简单的用一个值（最优值）代表整个子树，而不用去求出这个子树所可能代表的所有值。**  
**动态规划方法代表了这一类问题的一般解法。**我们自底向上（从叶子向根）构造子问题的解，对每一个子树的根，求出下面每一个叶子的值，并且以其中的最优值作为自身的值，其它的值舍弃。动态规划的代价就**取决于可选择的数目（树的叉数）和子问题的的数目**（树的节点数，或者是树的高度？）。  
**贪心算法是动态规划方法的一个特例。**贪心特在，可以证明，每一个子树的根的值不取决于下面叶子的值，而只取决于当前问题的状况。换句话说，**不需要知道一个节点所有子树的情况，就可以求出这个节点的值**。通常这个值都是对于当前的问题情况下，显而易见的“最优”情况。因此用“贪心”来描述这个算法的本质。由于贪心算法的这个特性，它对解空间树的遍历不需要自底向上，而只需要自根开始，选择最优的路，一直走到底就可以了。这样，与动态规划相比，它的**代价只取决于子问题的数目，而选择数目总为1**。

**两个实例**：最小生成树算法和单源最短路径算法，以及集合覆盖问题的贪心启发式算法。

prim算法：将集合A看成是一棵树，每次选择剩余的节点中与这棵树形成的边的权值最小的点加入到集合A中形成新的树，循坏调用该过程，知道所有的点都已经放入到集合A中。初始时随机选择一个节点放入到集合A中。

kruskal算法：在所有连接森林中两颗不同树的边里面，找到权重最小的边（u，v），并将其加入到集合A中，循环调用该过程，直到所有的点已经放入到集合A中