1. 求最大连续子序列和，输出最大的和以及这个子序列的开始值和结束值
   1. Sum为要求的最大和，tem临时的最大和，left和right标记要求子序列的下标，temindex标记left的临时下标。
   2. 一个子序列为结果，子序列的集合必然是有限的。确定左边，不断增加右边，当子序列和小于零了，更新左边，再继续增加右边；
   3. tem=tem+v[i]；tem变大更新right，变小right不更新；
   4. tem大于sum，更新tem；
   5. tem小于0了之后就更新temindex，从temindex继续计算子序列和，当新的子序列大于sum之后，更新right，left。

动态规划的本质不在于是递推或是递归，也不需要纠结是不是内存换时间。

理解动态规划并不需要数学公式介入，只是完全解释清楚需要点篇幅…首先需要明白哪些问题不是动态规划可以解决的，才能明白为神马需要动态规划。不过好处时顺便也就搞明白了递推贪心搜索和动规之间有什么关系，以及帮助那些总是把动规当成搜索解的同学建立动规的思路。当然熟悉了之后可以直接根据问题的描述得到思路，如果有需要的话再补充吧。

动态规划是对于 某一类问题 的解决方法！！重点在于如何鉴定“某一类问题”是动态规划可解的而不是纠结解决方法是递归还是递推！

怎么鉴定dp可解的一类问题需要从计算机是怎么工作的说起…计算机的本质是一个状态机，内存里存储的所有数据构成了当前的状态，CPU只能利用当前的状态计算出下一个状态（不要纠结硬盘之类的外部存储，就算考虑他们也只是扩大了状态的存储容量而已，并不能改变下一个状态只能从当前状态计算出来这一条铁律）

当你企图使用计算机解决一个问题是，其实就是在思考如何将这个问题表达成状态（用哪些变量存储哪些数据）以及如何在状态中转移（怎样根据一些变量计算出另一些变量）。所以所谓的空间复杂度就是为了支持你的计算所必需存储的状态最多有多少，所谓时间复杂度就是从初始状态到达最终状态中间需要多少步！

太抽象了还是举个例子吧：

比如说我想计算第100个非波那契数，每一个非波那契数就是这个问题的一个状态，每求一个新数字只需要之前的两个状态。所以同一个时刻，最多只需要保存两个状态，空间复杂度就是常数；每计算一个新状态所需要的时间也是常数且状态是线性递增的，所以时间复杂度也是线性的。

上面这种状态计算很直接，只需要依照固定的模式从旧状态计算出新状态就行（a[i]=a[i-1]+a[i-2]），不需要考虑是不是需要更多的状态，也不需要选择哪些旧状态来计算新状态。对于这样的解法，我们叫递推。

非波那契那个例子过于简单，以至于让人忽视了阶段的概念，所谓阶段是指随着问题的解决，在同一个时刻可能会得到的不同状态的集合。非波那契数列中，每一步会计算得到一个新数字，所以每个阶段只有一个状态。想象另外一个问题情景，假如把你放在一个围棋棋盘上的某一点，你每一步只能走一格，因为你可以东南西北随便走，所以你当你同样走四步可能会处于很多个不同的位置。从头开始走了几步就是第几个阶段，走了n步可能处于的位置称为一个状态，走了这n步所有可能到达的位置的集合就是这个阶段下所有可能的状态。

现在问题来了，有了阶段之后，计算新状态可能会遇到各种奇葩的情况，针对不同的情况，就需要不同的算法，下面就分情况来说明一下：

假如问题有n个阶段，每个阶段都有多个状态，不同阶段的状态数不必相同，一个阶段的一个状态可以得到下个阶段的所有状态中的几个。那我们要计算出最终阶段的状态数自然要经历之前每个阶段的某些状态。

好消息是，有时候我们并不需要真的计算所有状态，比如这样一个弱智的棋盘问题：从棋盘的左上角到达右下角最短需要几步。答案很显然，用这样一个弱智的问题是为了帮助我们理解阶段和状态。某个阶段确实可以有多个状态，正如这个问题中走n步可以走到很多位置一样。但是同样n步中，有哪些位置可以让我们在第n+1步中走的最远呢？没错，正是第n步中走的最远的位置。换成一句熟悉话叫做“下一步最优是从当前最优得到的”。所以为了计算最终的最优值，只需要存储每一步的最优值即可，解决符合这种性质的问题的算法就叫贪心。如果只看最优状态之间的计算过程是不是和非波那契数列的计算很像？所以计算的方法是递推。

既然问题都是可以划分成阶段和状态的。这样一来我们一下子解决了一大类问题：一个阶段的最优可以由前一个阶段的最优得到。

如果一个阶段的最优无法用前一个阶段的最优得到呢？

什么你说只需要之前两个阶段就可以得到当前最优？那跟只用之前一个阶段并没有本质区别。最麻烦的情况在于你需要之前所有的情况才行。

再来一个迷宫的例子。在计算从起点到终点的最短路线时，你不能只保存当前阶段的状态，因为题目要求你最短，所以你必须知道之前走过的所有位置。因为即便你当前再的位置不变，之前的路线不同会影响你的之后走的路线。这时你需要保存的是之前每个阶段所经历的那个状态，根据这些信息才能计算出下一个状态！

每个阶段的状态或许不多，但是每个状态都可以转移到下一阶段的多个状态，所以解的复杂度就是指数的，因此时间复杂度也是指数的。哦哦，刚刚提到的之前的路线会影响到下一步的选择，这个令人不开心的情况就叫做有后效性。

刚刚的情况实在太普遍，解决方法实在太暴力，有没有哪些情况可以避免如此的暴力呢？

契机就在于后效性。

有一类问题，看似需要之前所有的状态，其实不用。不妨也是拿最长上升子序列的例子来说明为什么他不必需要暴力搜索，进而引出动态规划的思路。

假装我们年幼无知想用搜索去寻找最长上升子序列。怎么搜索呢？需要从头到尾依次枚举是否选择当前的数字，每选定一个数字就要去看看是不是满足“上升”的性质，这里第i个阶段就是去思考是否要选择第i个数，第i个阶段有两个状态，分别是选和不选。哈哈，依稀出现了刚刚迷宫找路的影子！咦慢着，每次当我决定要选择当前数字的时候，只需要和之前选定的一个数字比较就行了！这是和之前迷宫问题的本质不同！这就可以纵容我们不需要记录之前所有的状态啊！既然我们的选择已经不受之前状态的组合的影响了，那时间复杂度自然也不是指数的了啊！虽然我们不在乎某序列之前都是什么元素，但我们还是需要这个序列的长度的。所以我们只需要记录以某个元素结尾的LIS长度就好！因此第i个阶段的最优解只是由前i-1个阶段的最优解得到的，然后就得到了DP方程（感谢

[@韩曦](/people/5fe6048f8cb6d4a0fa7df5f9a8e24c40" \t "_blank)

指正）  
IMG_256

所以一个问题是该用递推、贪心、搜索还是动态规划，完全是由这个问题本身阶段间状态的转移方式决定的！

每个阶段只有一个状态->递推；  
每个阶段的最优状态都是由上一个阶段的最优状态得到的->贪心；  
每个阶段的最优状态是由之前所有阶段的状态的组合得到的->搜索；  
每个阶段的最优状态可以从之前某个阶段的某个或某些状态直接得到而不管之前这个状态是如何得到的->动态规划。

每个阶段的最优状态可以从之前某个阶段的某个或某些状态直接得到

这个性质叫做最优子结构；

而不管之前这个状态是如何得到的

这个性质叫做无后效性。

另：其实动态规划中的最优状态的说法容易产生误导，以为只需要计算最优状态就好，LIS问题确实如此，转移时只用到了每个阶段“选”的状态。但实际上有的问题往往需要对每个阶段的所有状态都算出一个最优值，然后根据这些最优值再来找最优状态。比如背包问题就需要对前i个包（阶段）容量为j时（状态）计算出最大价值。然后在最后一个阶段中的所有状态种找到最优值。