动态规划专题 难点部分批注

术语部分

阶段 状态 决策

通过dp的思路我们可以将原本独立的，整体的问题分解为一些有顺序关系的阶段。我们按照这种顺序，一步一步的考虑每一个阶段，求解出每一个阶段中包含的状态。在求解每个阶段的过程之中，我们可以充分利用前面的阶段已经计算出的答案，进而将指数级别的枚举转换为阶段数，每个状态的状态数，决策数的乘积。最终，我们就在多项式的时间复杂度内解决了问题。

状态转移方程

其实就是一个描述每一个状态是如何从前面的状态之中产生的公式。

目标函数与最优化

其实就是用来修饰状态转移方程的一个修饰语。用来规定和对比不同的状态转移之间的优劣。
我们通过这种修饰来从众多的转移中找到我们想要的那一个。

无后效性

一个状态的表示（转移方程之中）确定的约束了每一个状态，后无效性的意思大概就是一个确定的状态表示，只能对应一个确定的状态，不会因为之前的一些多余的信息（例如如何到达这个状态）就表示另一个状态。

记号约定

方便大家记忆和讨论每一种不同形式的dp转移方程对应的不同的优化的方法。

第一道例题 录制唱片

第一种想法

想法是：第i首歌的状态有第i-1首歌的状态转移而来。
当前的决策是：是否要加入第i首歌。
如果当前考虑的这首歌的长度小于等于k，那么说明，之前没有加新的唱片。
反之，则说明之前加入过新的唱片。

第二种想法

想法是：每次任然是考虑当前的这首歌应不应该被加入。
但是将时间这种东西放到了状态里面了。
而且，通过之前的讲解，大家应该可以看到，这种状态表示，也是能够对应唯一的一种状态的，
也就是说，是无后效性的。

具体的做法和之前的相似，但是就不用去枚举时间了

总结

总之，我们对于每一个dp问题，在设计状态的时候，都应当尽量选择状态数，转移数尽可能少的状态表示，（当然，前提是要满足后无效性）。

第二道例题 划分大理石

初始的状态表示

其实想法就是考虑当前这种石头能要几个，然后将问题就转化为了之前的阶段之中已经解决了的问题。

递推方法的优势

注意到，我们靠着每次枚举选取石头的个数，创造出的子问题，其实不一定是true，
而我们不太需要不是true的状态。也就是说，我们枚举了一些不存在的状态，浪费了时间
所以，我们这里采用了一种递推的方式，仅仅从当前阶段true的状态来推出下一个阶段true的状态，这样就减少了时间的开销。

双向广度优先搜索

递推的方法，可以理解为广度优先搜索。
相信大家都知道广度优先搜索，那么什么是双向广度优先搜索呢?
比如，假设你不知道魔方的所有的公式，让你判断一个魔方能否从状态A，扭到状态B，这种问题。
我们同时知道了起始的状态和结束的状态，所以，要给出这个问题的答案，我们自然不难想到两种方法。
1: A -> B
2: B -> A
想到这两种方法之后，聪明的同学就会想到，为什么不能同时从A和B xjb扭,然后看它们能不能同时扭到一个中间状态C，呢？所以，这种思路大概就是双向广搜的思路。

双向广搜图形的解释

用圆表示的话，显得十分的直观。
但是，这个图，看上去是一个两边的面积，左边是右边的两倍的关系，实际上却不是。
实际上，可以理解为，圆的质量不是均匀分布的。
对于大多数广度优先搜索问题，随着搜索深度的增加，状态数往往成指数级别增长。
所以这里的每个质点所代表的质量随着半径的增大，成指数级别增长。
所以左边的面积要开根号才和右边的面积是同一个量级的。

所以，对于同一个搜索任务，双向广度，往往能将时间开根号。

母函数的解释

母函数其实是通过多项式的计算来模拟这类计数问题的求解过程的一种数学工具。
多项式的计算结果中，每一项的指数表示了最终得到的价值，每一项前面的系数则表示了得到这个价值，一共有多少种不同的方法。如果某一项前面的系数为0的话，就表示这一项的次数所对应的价值是
我们搞不出来的。

母函数的效果和状态数都和我们这里使用的递推是同一个效果。

优化的点

可以看到，通过我们这里的这种操作，我们就不用将最后剩下的两个括号乘起来了。
那将会是令人难受的操作。

---

hash表的介绍。如果你估算出来两边的括号内的多项式的最高项次数可能会取得比较高。然而，实际的状态数可能又比较小的话，这个时候可以考虑set，但是有些时候set的速度会让我们感到难受，

所以我们可以考虑使用hash表，这里只提一下，不继续深入。

矩阵加速

什么是线形递推

ppt中的那种形式的递推公式就是线性递推。

快速幂

我们能够快速的计算出一个数的2^n。
然后我们再来看，原来题目中要求的指数。
搞清楚这个数是由那些2的次幂加起来求的的。
然后由于乘法的结合律，我们将这些2的次幂找出来，给他乘进去。

显然，这样是非常快的。

石子合并

转移方程的思路

转移的时候，按照区间的长度来划分阶段。
对于当前的i -> j 的这样一个区间，我们找一个中间的点K，将这个区间划分为两个
长度较小的区间（对应的状态都被算过了）。

四边形不等式的第二种理解

其实这种理解还不是十分区域本质。
但是，我在讲这个不等式的时候，希望大家能够从粗浅的理解，最终到达深入的理解。
因为这种理解确实就是，国内各大论坛，博客上的一种最常用的理解。
我最开始研究这个方面的时候也吃了很多的亏，希望大家能更多的从这次的讲座之中，
学习到一种自学的方法。能够通过不停的在网络和书籍上搜索资料来充实自己的理解。
最后，希望大家在学习的过程中，能够更多的关注算法的英文名字，这样在你搜索资料的时候，会带来很大的便利。

这里我只是稍微画了一个图形，希望大家能够更清楚的记住上面的不等式大概要表达的意思。

区间包含关系单调的理解

其实就是保证一个序列的每一项大概是非负的就好了。

转移方程的简化表示

利用前缀和就可以很快的算出来w(i,j)

四边形不等式，定理一的解释

这个东西其实也不能叫做定理，大概只是个结论吧。只是我懒得去改ppt了。

四边形不等式，定理一证明

需要分情况讨论。其实只是因为四个值的取值会导致得到的式子的结构不太相同，
但是本质都是一样的。大家不必过多的纠结如何分情况。因为，之后会证明一个比较重要的结论，然后就不用分情况了。有些时候甚至都不用证明了。

这些证明看起来会比较复杂，其实都是套路，只要证明了第一个，之后大家自己都能证明出来。

套路：利用数学归纳法，假设小于一个长度的所有取法都是具有凸性的，将每次从中间选取一个点，将原来的不等式左边的两个区间打开，然后不停的利用w(i,j)的凸性。或者利用m(i,j)在长度较小的时候的凸性，不停的放缩不等式，从而将不等式的右边推出来。

四边形不等式，定理二证明

这个结论是优化的关键所在。
证明转化：其实意思就是说：如果当前满足$K_1 \geq K \wedge m_{k_1}(i,j) \leq m_k(i,j)$，
那么当j变成j+1之后也一定满足这个结论。不会突然从比K_1小的数中出现比m_{K_1}更小的值，
当j变大的时候，只有比k_1还要大的数才可能成为新的决策。
这说明了什么呢？ 说明了当j变大的时候，决策不会变小。所以和我们要证明的东西是同一个意思。
这个东西一般的行话叫做决策单调性。

合并石子， n^2时间复杂度分析解释

大致的意思就是，对于每一个位置固定的i（左端点），我们枚举的所有的j（右端点）的时候，通过决策单调性枚举的中间点K，的次数，是刚好和n是同意个量级的。最后就消掉了。
所以最后就是n^2的了。

四边形不等式，定理三作用的解释

有了这个定理之后，你在证明形如定理一的这种结论的时候，就可以不用再分几种情况讨论了。
更加偷懒的方法是，先写一发大暴力，打个表，然后定理三给了我们一种通过人力大致判断一个矩阵是不是满足四边形不等式的一个高效的方法。

注意，在这之后的讨论之中，为了加深大家的理解，我会统一将四边形不等式这种叫法，换做Monge array，这种通用叫法，他表示了一种具有四边形不等式性质的矩阵。
在矩阵中，四边形不等式可以更加直观的表示为：对于矩阵的任意的一个二阶子式，都满足，主对角线上的元素和小于反对角线上的元素的和。
四边形不等式其实是一种Monge array的特殊情况。

这种优化方法和四边形不等式的概念是姚期智教授的妻子储枫教授在1980年首先提出的。
算是一种Monge array的应用吧。

深入理解Monge array

下面给大家讲一下Monge array

凸完全单调性的证明

主要是利用反证法。
其实这个东西，非常的显然。
因为如果不这个样子的话，就会很矛盾。会不符合Monge array的基本性质。
这个性质不仅仅局限于这个公式，大家也不用死记，只要知道是怎么推的。
就能在行和列两个方向上都灵活的使用这个性质。

最小值行编号性质的证明

同样是利用反证法。
这里是把j处理成了关于i的一个函数。

辅助定理二的解释

证明辅助定理二的时候，我们给出一种能够$O(n)$解决这个问题的算法。
因为每一行只可能有唯一的一个列成为答案，所以我们分行计算，然后每次排除一些不可能成为答案

的列。

对于删掉n-m列的解释

加了一行之后，并不会影响答案。
那么根据最后一行的性质，如果队列中已经有m+1个元素的话，那么再加入新的元素的时候，一定会有出栈的操作。至少会把之前的那一列给出栈。
最后留在栈里面的都是没有被排除的。

对于$m \leq n$的解释

如果不是这样的话，我们的辅助定理二就不是很有用了。

斜率

对于convex hull trick名字的解释

因为这个东西维护的实际上不能叫做一个凸包，只能是叫做一个上或下凸壳。所以名字后面加了个trick

对于Query的解释

这种做法在查询n次之后，消耗的时间是$O(nlogn)$的。
如果我们查询的点是有序的话，可以使用双端队列，使得查询的复杂度降为$O(n)$。

例题的转移方程的解释

枚举当前区间的开头，进行转移。

斜率无序的情况的解释

就是用set维护斜率和交点的横坐标，两个键值。
加入新的直线的时候，用斜率为键值，找到这个直线对应的位置。
查询的时候用横坐标为键值，找到这个点具体对应那个直线。
还是比较好理解的。

转移方程的做法

一般来说，斜率优化之有可能出现在$1D/1D$这种形式的方程上面。
一般的话，如果你确定一个题可以用斜率优化的话，那么你一定可以找到一个方法
将它的方程化简为斜率优化的一般方程的形式。

减少状态转移花费的时间复杂度 解释

这一部分的问题在于高效的维护，查询一些我们想要的信息，比如快速的求一个区间中的最小值之类的。
主要是属于数据结构的内容，相信大家的数据结构都学得不错。
只要善于观察，在适当的时候使用合适的数据结构就可以了。

今天我所讲的东西

今天给大家讲的东西比较偏理论，也不太容易消化。是dp的难点。
dp还有很多类型和技巧，我认为，以大家的能力，都是可以轻松自学的。

但是要自学Monge array和 斜率优化 的话，却是一件极其痛苦的事。
今天相当与带着大家熟悉了一下这些比较困难的概念，让大家在之后的学习中，对它们有情切感。加快大家之后看这方面资料的速度。