线性dp树形dp

PKU Hzyoi

01 动态规划

DP

动态规划

- 在我们刚开始学习动态规划的时候，看到的教科书基本是长这样的：
- *对于动态规划类题目，可以分为以下解题步骤：*
- *1、具有最优子结构的性质*
- *2、划分阶段*
- *3、建立转移方程*
- 但是面临的题目都非常难，做题时也根本考虑不到这些步骤，纯粹靠“自由发挥”。

动态规划

- 随着做题数的增长，以及对历史经验的总结，才会逐步发现：
- **教科书上的话真是至理名言啊！**
- 首先DP的几个要素：
- **阶段**　可划分，无后效性　　　背包、区间、树形……
- **状态**　状态数少　　　权值 - 体积
- **权值**　能比较（最优子结构）　　　字典序
- **转移**　效率要有保证
- 注：所谓的无后效性是指：“下一时刻的状态只与当前状态有关，而和当前状态之前的状态无关，当前的状态是对以往决策的总结”。

动态规划优化

- 通常的Dp优化也从这些方面入手：
- **阶段** 倍增、正序 - 逆序（从大到小、由外而内）......
- **状态** 状态的“线性相关”、bitset......
- **权值** 前缀和（最大值）、可行性->判定性...
- **转移** 决策单调、斜率优化、数据结构
- 当然很多东西在这个noip的课程中不会涉及，将来继续深入才会碰到，本次课程会给大家介绍一些常见的DP形式，旨在给大家建立一个整体的认识，增加大家对于DP题目的敏感。

老生常谈

- 做Dp题的一般思路：
  - 1、先想一个初步的Dp（可能效率爆炸），把你需要的状态全部记下来，然后建立转移方程。
  - 2、逐步优化
- 如何提高Dp水平？
- **多做题......**

02

线性DP

序列型动态规划

线性DP

- 指阶段呈线性的动态规划，是在线性结构上进行状态转移
- 一维情况下的一般形式为f[i]表示以i结尾的序列信息，高维的矩阵相关的DP也是线性DP
- 另外背包问题这个东西我不确定能不能算线性DP，毕竟算起来它复杂度都不是多项式的，但这个东西显然是必讲的，也会放在这个专题。

经典模型I

- 最长不下降子序列
- 给出N个数，求出其最长不下降子序列的长度

- 30% ($0 < N < 1000$)
- 100% ($0 < N < 100000$)

- 30% 应该都会的DP

- f[n]表示以n结尾的最长不下降子序列的长度最长为多少。
- 转移：$f[i] = max(f[j])+1$ $(a[j]<=a[i]$ && $j<i)$

- 复杂度 ： $O(n^2)$

- 100% 因为这里的数据范围比较大，用经典的n^2做法肯定会超时，所以需要一种更加高效的DP。

- 用$g[n]$表示最长不下降子序列长度为n的序列末尾最小值为多少。
- 然后我们转移的时候就可以二分，二分最长不下降子序列的长度，然后与$g[mid]$比较，得出答案。用答案更新f数组。

- 复杂度：$O(nlogn)$

*[51Nod1218] 最长递增子序列 V2

- 给出一个数列，问哪些数可能出现在最长上升子序列中，哪些数一定会出现在最长上升子序列中。

标准算法：

- 从前往后做一遍最长上升子序列，再从后往前做一遍最长下降子序列，分别求出经过每个元素的最长上升子序列长度，如果这个长度等于全局最长上升子序列长度，那么它就可能出现在最长上升子序列中；如果这个位置的元素唯一，那么它一定出现在最长上升子序列中。
- 总结：
- 最长上升子序列和最长下降子序列，基本可以互相转化，将两者综合起来，能够突破得到结果的前缀性质，而将其转化为全局性质。

经典模型II

- 0/1背包问题
- 给定N个物品，每个物品有一个重量和价值，你现在有一个大小为M的背包，问你最多可以装多少价值的物品，每个物品只能用一次
- 完全背包问题：每个物品能用无限次
- 100% (0 < N,M <3000)
- Memory Limit:
  - 30% 128MB
  - 100% 32MB

- 也是一个很经典的问题
- 用f[i][j]表示前i个物品使用了j的空间的最大收益
- 转移方程:f[i][j] = max(f[i-1][j-cost[i]]+val[i],f[i][j])
- 但是对于100%的内存限制如果数组开满O(N*M)显然是会MLE的，所以我们可以对第一维滚存，空间就变成O(M)了。
- 滚存时维护一个f[i-1]和一个f[i]就没问题
- 但如果只用一个f，为了满足转移方程，转移时枚举j需要从大到小枚举。

关于完全背包

- 状态设计相同，但转移方程变化
- 转移方程:$f[i][j] = max(f[i][j-cost[i]]+val[i],f[i][j])$
- 滚存时如果只用一个f，为了满足转移方程，转移时枚举j需要从小到大枚举

多次背包问题：

- 物品个数有限制了，最简单的做法是想到把第i个物品拆成K_i个。但是出题人是不会让你这样A掉的。
- 所以我们需要别的做法来做。
- 展开想象的翅膀→记得怎么设计邮票面值吗？

- 把物品拆成2^k次个（即按照二进制拆分）能使件数达到最小，还可以凑出所有个数个。

- 接下来就用普通的01背包处理。

- 时间复杂度$O(C*\sum(\log_2 K_i))$

单调队列优化

- 继续多次背包的问题
- 这个问题我们也可以用单调队列来进行优化
- 对于第i种物品来说，已知体积v，价值w，数量k，那么可以根据当前枚举的体积j对v的余数把整个动归数组分成v份。

- 像这样：

编号j	0	1	2	3	4	……
对应体积	d	d+v	d+v*2	d+v*3	d+v*4	……

- 现在看到分组以后，编号j可以从j-k到j-1的任意一个编号转移而来(因为相邻的体积正好相差v)，这看上去和区间和的最大值是不是很像？
- 于是就用求区间最大值的单调队列进行优化

[51nod]有限背包计数问题

- 有一个大小为n的背包，有N种物品，第i种物品有i个， 体积为i。 • 问装满背包有多少种方案。
- N<=100000

- 直接背包显然不能做。
- 大于根号的一定用不完。而且总数不超过根号个。
- 所以可以直接dp[i][j]表示当前已经用了i个，体积之和为j
- 每次新增一个大小为根号的物品或者将所有物品size+1
- 每个状态只有两种转移，状态总数为n*sqrt(n)

- 小于根号的套上面的多重背包

*Noip2014 飞扬的小鸟

- Flappy Bird 是一款风靡一时的休闲手机游戏。玩家需要不断控制点击手机屏幕的频率来调节小鸟的飞行高度，让小鸟顺利通过画面右方的管道缝隙。如果小鸟一不小心撞到了水管或者掉在地上的话，便宣告失败。
- 为了简化问题，我们对游戏规则进行了简化和改编：
  - 1 游戏界面是一个长为n，高 为m的二维平面，其中有k个管道（忽略管道的宽度）。
  - 2 小鸟始终在游戏界面内移动。小鸟从游戏界面最左边任意整数高度位置出发，到达游戏界面最右边时，游戏完成。
  - 3 小鸟每个单位时间沿横坐标方向右移的距离为1，竖直移动的距离由玩家控制。如果点击屏幕，小鸟就会上升一定高度X，每个单位时间可以点击多次，效果叠加；如果不点击屏幕，小鸟就会下降一定高度Y。小鸟位于横坐标方向不同位置时，上升的高度X和下降的高度Y可能互不相同。
  - 4 小鸟高度等于0或者小鸟碰到管道时，游 戏 失 败 。小 鸟 高 度 为m时，无法再上升。
- 现在，请你判断是否可以完成游戏。如果可以，输出最少点击屏幕数；否则，输出小鸟最多可以通过多少个管道缝隙。
- 5≤n≤10000，5≤m≤1000，0≤k<n，0<X<m，0<Y<m，0<P<n，0≤L<H ≤m，L+1<H。

题解：

- 用f[i][j]表示小鸟在(i,j)时最少点击屏幕的次数。
- f[i][j]=min{f[i-1][j+y[i-1], f[i-1][j-k*x[i-1]+k}。
- 最暴力的方法是从1开始枚举k，然而这样会T。
- 观察一下可以发现转移方程式可以简化为f[i][j]=min{f[i-1][j+y[i-1], f[i][j-x[i-1]+1}(类似于无限背包问题)
- 时间复杂度O(nm)

更加高维的问题

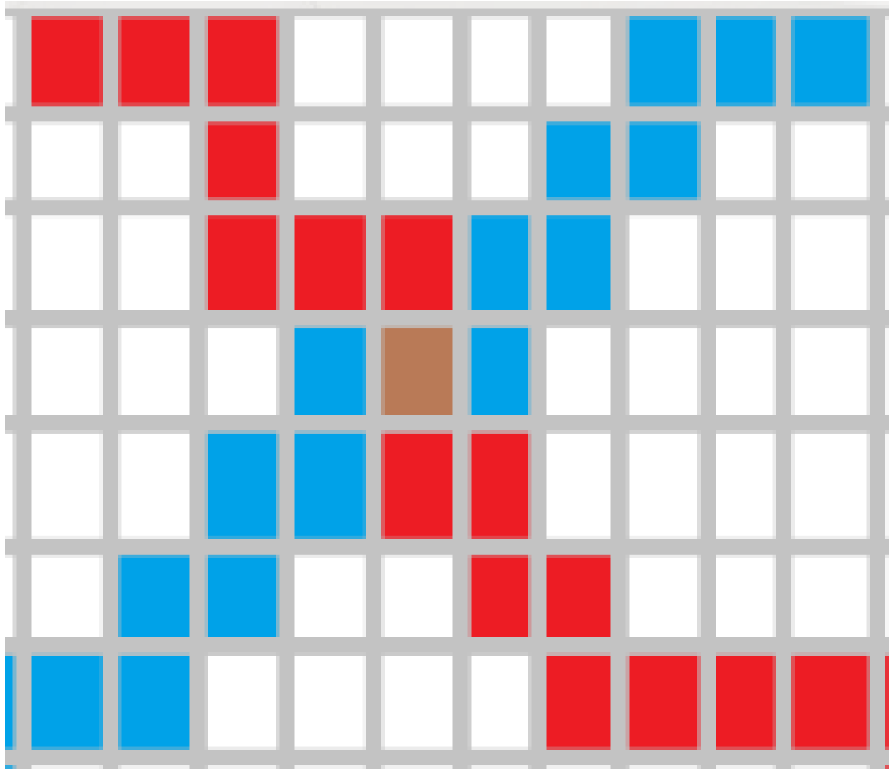
- 直观上就是高维的
- 数字三角形
- 在一个数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。路径上的每一步都只能往左下或右下走。只需要求出这个最大和即可，不必给出具体路径。

```
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

【245DIV1B】B. Working out

- 有一个n*m的方格图，每个方格有一定量的钱，现在JPY从左上角出发，只能往右或往下走，目的地是右下角。YPJ从右上角出发，只能往左或往下走，目的地是左下角。他们都很厉害，每经过一个格子会把这个格子的钱给捡走，由于JPY和YPJ都是追求完美的人，所以他们要求只有一个格子被他们两人都经过，现在求他们两个人最多能捡到多少钱。
- 注意如果一个格子被两个人同时经过的话钱只能捡一次
- n,m<=1000

- 我们可以枚举交点，然后我们可以发现，JPY的路径被分成了两段，一段是从左上角到交点上面或者左边，一段是从交点下面或者右边 到右下角。YPJ的路径也同理
- 于是我们DP出，
- 从四个角落出发到某些点能得到的
- 最多的钱，最后枚举交点算一下即可



高维的状态设计

- [loi2007]Miners 矿工配餐
- 你有N波食物，食物有三种类型，你有两头牛，每次你获得一波食物后可以选择一头牛喂给他，牛吃完后获得的兴奋值为：这次食物与前两次食物中，类型不同的食物的种数，现在给你食物的类型，让你求你最多可以获得多少兴奋值

- $1 < N <= 100000$

- F[i][k1][k2][p1][p2]表示，前i波食物后，第一头牛前两次吃的食物的种类分别为k1,k2；第二头牛前两次吃的食物种类分别为p1,p2，所能获得的最大兴奋值，转移时只要枚举喂给哪只JPY就行了。

03 树形DP

树形DP

- 树形DP，顾名思义，就是在树上进行的DP。
- 普通线性DP转移方程大致形式为$f[i]=g(f[j])$，$1\leqslant j<i$（这里$g(x)$表示一个函数，如max，min等）。
- 而树形DP的转移方程为$f[i]=g(f[s])$，s为i的子节点（son）。
- 我们来看一道例题。

hdu1520

- 【问题描述】
- 给定一棵N个点的关系树，每个节点有个权值，子节点和父节点不能同时选，问最后能选的最大价值是多少？
- 【数据范围】
- $N \le 6000$

- 这题很简单，而且题面描述也很清楚，就不放样例了。
- 设f[i][0/1]表示第i个点选（1）或不选（0），以i为根的这棵子树能选的最大价值。易得转移方程为：
- $f[i][0]=\sum(\max(f[s][0],f[s][1]))$
- $f[i][1]=\sum(f[s][0])$

- 然而，线性和树形两个看似差不多（甚至完全一样）的DP方程对于初学者来说却有着天壤之别。
- 我们来看二维DP的情况：
- 线性：$f[i][j]=g(f[i-1][k]),1 \le k<j$
- 树形：$f[i][j]=g'[\ g(f[s][k])\ 1 \le k<j\ ]$, s是i的孩子
- 很显然，线性DP只有一个函数，而树形DP却有两个。

- 正是树形DP和线性DP这样一个小小的不同，使得树形DP与线性DP相比，难度有了一定的提高。
- 在f[i][j]=g'(g(f[s][k]))中，g'(x)和g(x)往往相似但不同。
- 有时为避免重复计算，要先另开一数组计算g(x)，再对g(x)进行一遍DP（一般是背包），计算g'(x)。

树形依赖背包问题：

- 给定 n 件物品和一个背包。第 i 件物品的价值是 W_i,其体积为Vi,但是依赖于第X_i件物品（必须选取Xi后才能取i，如果无依赖则 X_i=0），依赖关系形成森林，背包的容量为C。可以任意选择装入背包中的物品，求装入背包中物品的最大总价值。

- 直接DP显然是n*C^2的
- 这就是上面说的g’所带来的的不便利之处
- 0/1背包数组中可以以O(C)的复杂度加入一个物品，但在树上只能合并两个dp数组，需要C^2
- 考虑怎么样才能像0/1背包一样只有加入一个物品
- 在往孩子节点dp时，把现有的dp数组直接传入进去，并强制选择父亲节点的物品（树形依赖，实现中体现为数组复制时的下标位移），于是到每个叶节点上只需要O(C)加入一个物品，总复杂度降为O(n*C)
- 不过在接触dfs序之后，这种方法和在dfs序上dp的本质完全相同，那个更好理解

不知道算不算DP的经典应用

- 给出一棵N个节点的树，求出树上到其他所有节点距离和最大的点。
- 随便取根，深度优先搜索计算出以各点为根的子树的节点数与根到所有节点的距离和；广度优先搜索根据根到所有节点的距离和，推算出各点到所有节点的距离和，结果取最大值。

不知道算不算DP的经典应用二

- 给出一棵N个节点的树，求这棵树的直径
- 记f[x]为x这课子树内的直径
- 考虑从f[y]中取max，然后剩下的情况肯定是这条路径的两端分别在x的不同子树中
- 于是再记g[x]为x向下的最长链，在遍历y时边维护g[x]边取max

- 不用dp的做法：任取一个点，dfs找出离它最远的点，再以这个点为根dfs一次

一个树形DP的常用优化技巧

- 一个例题
- 一棵n个节点的有根树，每个点有点权ai，现在要从这n个点中选出k个点，满足n个限制：
- 对于节点x有一个长度为n 的01串Sx表示限制，Sx的第i位为1表示x的子树中可以选取i个点，否则不行。
- 求权值最大和
- N,k<=5000

- 直观想法f[x][k]表示x子树内取k个点的最大值，不合法的赋为-oo
- 直接做的n*k*k的，过不了
- 加个优化
- 枚举k的时候把上限从k改成min(k,sz[x])，sz是x子树内节点个数
- 复杂度就变成n^2的啦
- 证明的话，考虑任意两个节点会在他们的lca处贡献一次复杂度

非常感谢您的观看

THANK YOU FOR YOUR WATCHING