Union Training II - Dynamic Programming

A. 雷神之路 （多矩阵快速幂）

> 一条长度 N(N<1e18)的路，一次可以向前走一步、两步或者三步。
> 有些地方有地雷不能走，问走到终点的方案数是多少。

这题 N高达 1e18,反而直接提示了做法。这题非矩阵快速幂不可。
构造两个矩阵，norm表示下一格没有雷，向前递推一步
而 zero表示下一格有雷，向前递推一步
然后对地雷的位置排序，第 i个地雷据上一个地雷的距离为 s
则对答案向量左乘 norm^(s-1)*zero

其中

$$norm = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$zero = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

初始向量为

$$X = \begin{bmatrix} F_0 \\ F_{-1} \\ F_{-2} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

B. Snowdrop修长廊 （斜率DP）

> 使用若干条线段，覆盖坐标轴上的 N个点
> 覆盖 $[i, j]$的代价为$cost(i,j) = W + (x_i - x_j)^2$
> 求覆盖所有点的最小代价

斜率优化入门题，与 HDU - Print Article一致
对坐标排序后，设dp[i]表示已经覆盖前 i个点的最小代价
容易得出DP方程为
$dp[i] = min(dp[j-1] + (x_i - x_j)^2)$
这是 $O(N^2)$ 的转移，所以需要优化

设有 $k < j$，且满足
$dp[k-1] + (x_i - x_k)^2 \geq dp[j-1] + (x_i - x_j)^2$
那么我们选择从 j 转移，把 k优化掉

通过整理上式，可以得到
$\frac{(dp[j-1]+x_j^2)-(dp[k-1]+x_k^2)}{2(x_j-x_k)} \leq x_i$
设 $Y_i = dp[i-1] + x_i^2$，$X_i = 2 * x_i$
这样左边其实就是一个斜率的式子 $\frac{Y_j-Y_k}{X_j-X_k} \leq x_i$
然后右边 x_i 是单调递增的，所以斜率也是单调递增的
意味着不满足单调递增的点 k都可以被优化掉
所以dp过程中需要用到的点连起来实际上构成了一个下凸包（斜率单调递增）

C. TaoSama与煎饼 （序列DP+状态优化）

> 一条长度为 N的道路，其中每个点有个权值
> 有 M个道具，能使煎饼向前跳跃1、2、3或4步
> 保证所有道具使用完时，煎饼落在 N位置
> 求到 N位置的一条路径，使得煎饼沿路获得的权值和最大

首先朴素的想法是用 dp[i][m1][m2][m3][m4]
表示煎饼在 i位置，使用 +1、+2、+3、+4的道具数目
分别为 m1、m2、m3、m4所获得的最大权值

但是显然这样状态爆炸，所以我们可以削减一维状态
由于我比较蠢，所以我的做法并不高明，但是因为数据更蠢，所以我过了
可以把 m1这一维去掉，这是由于 i，m2,m3,m4可以计算出 m1
然后枚举当前要使用的道具，根据这几个状态进行转移即可
然后这样会 MLE，要使用滚动数组，我因为滚动时memset放错位置还 WA了两发
时间复杂度 O(N*K^3)，其中 K为某种道具的最大个数

其实正解的思想是一样的，只不过他削减了 i这一维，

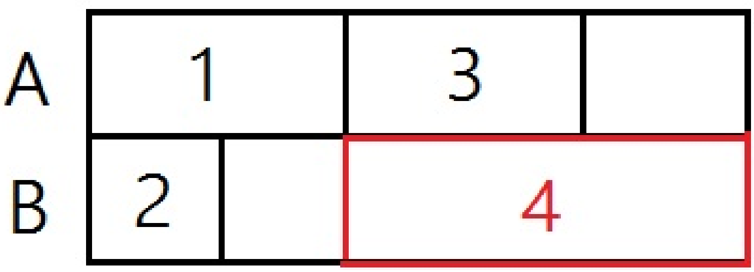
因为知道了 m1，m2,m3,m4同样可以算出 i
这样时间复杂度就变为了 O(K^4)

D. 任务 （状态优化DP）

> 给两个机器安排 N个任务，要求满足
> 1. 任务必须全部被安排到任一机器上
> 2. 同一时间一台机器只能运行一个任务
> 3. 任务 $j(j < i)$ 必须在任务 **i**之前被安排
> 4. 在一台机器上运行的任务不能被打断
>
> 第 i个任务在两台机器上的用时分别为 a_i ，b_i
> 要求所有任务的总用时最小

这题我想了三天三夜，还是没搞明白这题怎么做
直到第四天早上起来，再读一遍题目，才发现我少读了一个条件
第三个条件尤为重要：任务 **j (j < i)**必须在任务 **i**之前被安排

例如之前 A已经被安排了任务 1,用时 2; 任务 3,用时 2
B之前已经被安排了任务 2，用时 1
那么现在我要给 B安排任务 4，用时3， 只能从 2时刻开始安排



我在高数课上把条件三反复咀嚼了一百遍以上，然后回去以后十分钟就秒了这题

所以实际上，任何两个任务被安排的时间差不会超过最大任务时间
所以这题的状态其实和经典的双子塔的 dp状态差不多
设 dp[i][k][j] 为前 i个任务，用时最多的机器是 k，两者用时差为 j
dp里面存的是最少用时
有了这些状态，我们可以知道 A、B的最小用时，
即下一个任务在两边该从什么时刻开始安排
然后就可以进行转移了

E. Goozy的积木 （状态优化DP）

> 有若干个积木，你可以选择将他放在 A塔，放在 B塔，或者不放
> 求在两塔高度相等的情况下，所能达到的最大高度

朴素的想法是 dp[i][H1][H2]表示使用前 i个积木
A塔高度为 H1，B塔高度为 H2，dp存的是能否到达这个状态的一个 bool值
首先这样状态爆炸，所以依旧要合理调整状态位置
我们可以把最高塔的高度移到 dp存的最优值里，
然后状态里表示一下两塔的高度差，因为根据高度差能算出次高塔的高度
这样一来状态表示就变为了二维，
即dp[i][d]表示使用前 i个积木，两塔高度差为 d，较高塔的高度最高为多少
最后答案即为 dp[N][0]

F. 先锋看烟花 （数据结构优化DP）

> 一条路上有 N个房子，一共有 M个烟花
> 在 t_i时刻，第 a_i个房子会放一个价值为b_i的烟花
> 对在 cur位置的先锋会产生 $b_i - abs(a_i - cur)$的幸福度
> 其中幸福度可以为负数
> 先锋每个单位时间最多可以移动 D的距离
> 问所有烟花放完之后，先锋的幸福度最大为多少

首先朴素的想法是暴力模拟这个过程
用 dp[i][j] 表示第 i 秒，处于 j位置的先锋能获得的最大幸福度
在没有烟花的时刻，先锋可以左右移动 D的距离
所以向左右走2*D，以最优答案来更新周围的值

经过模拟可以发现，在没有烟花的时候
某点的答案会其实就等于某段区间内的最大值
并且这段区间随着时间地增长会变大

所以对放烟花的时间排序，在放烟花的时刻，扫一遍数组更新答案
在没放烟花的空闲时刻，算出到下一个烟花绽放时刻之前，
从任意一点出发，先锋能移动的最大范围

用稀疏表求出此范围内的最优答案，然后再暴力扫一遍数组更新答案

总的复杂度$O(NMlog(N))$
如果使用单调队列维护最大值，还可以优化到$O(NM)$

G. Simple dp （XJBLG法）

> 给定 N个点，每个点的权值表示以此点为根的子树中节点的个数
> 每个非叶子节点至少有两个儿子，问给定的 N个点能否组成满足条件的树

这题我是贪心构造，虽然正确性无法保证
但由于N比较小，最多24，所以难以构造数据把我卡掉

直观感觉权值大的点在树上的位置尽量靠上
先将点从小到大排序，然后从大到小选择根
然后在小于它的为标记的点中选择它的儿子，
保证儿子的权值尽可能大，然后打上标记
然后依次循环，如果中间某点为根无法构造出子树，则返回

以上方法并不能够通过，这是因为这个办法是错误的
但是可以反过来做，也就是说从小到大选择根
然后在小于它的为标记的点中选择它的儿子，其余操作基本相同

以上方法依旧不可能通过，但是如果你两种贪心顺序都判一下
如果两个顺序都无解，就输出无解，否则输出有解，就能够得到 AC
时间复杂度$O(N^2)$

H. 又见背包 （可行性背包DP）

> 有 N个大小不同的数字，第 i种数字为 a_i，每种有 m_i个
> 求问能否从中选出若干个数字，使他们的和为 K

背包九讲 2.0的例题，用多重背包的二进制能过

根据 lyb dalao所述
因为 $K < 1e5$，所以其实最后用到的物品数量不会超过 1e5
所以 $m_i = min(m_i, K/a_i)$，所以用二进制优化能过

背包九讲上的做法如下：
dp[i][j] 表示前 i个物品，组成背包容量为 j时，
所剩下的最多第 i个物品的数量，不能到达的状态设为 -1
然后检查能否转移到 dp[N][K]即可

I. Mingo's Game （斜率DP）

> 有 N个关卡，可以分为 K块，每个关卡都有个权值 t_i
> 每次选择最早没有通关的关卡块，设这个关卡包含了 $[i, j]$ 的游戏
> 选到最早没有通关的关卡是k, 选到 k的概率是 $P = \frac{t_k}{\sum_{x=i}^{j} x}$
> 选到一个关卡一定能通关，花费一小时
> 求合理分块的情况下，通关所有关卡块的期望时间最小是多少

原题是 CodeForces - 643C Levels and Regions ， 做法是斜率优化DP。
概率公式的推导过程一脸懵逼，反正我看了这个题解才会做的 0.0

> http://m.blog.csdn.net/article/details?id=51346853

L. 来签个到吧 （GCD + 期望）

> 盒子里有若干个球，每个球上面都有一个数字，数字各不相同
> 每次从中选两个数字 x，y，设 z=$|x-y|$
> 若 z不在盒子中，则加入这个数
> 反复执行操作，直到无法再向盒子里加数
> 随机从盒子中摸出一个球，反复执行这个操作直到所有球都被摸出来过
> 问最后的期望步数

第一部分的构造：
设所有数的最大公因数是D
则所有数可以表示为$x=k*D$
所以所有的$|y-x|=k'*D$，必然是D的倍数
实际上这个相减的过程是更相减损法的再现
所以保证一定能构造出最大公因数D
构造出D后，用最大的数不断减去D，
就能构造出小于最大数的所有D的倍数

第二部分的期望：
设$dp[i]$为摸到i个求的期望步数
$dp[i]=\frac{N-(i-1)}{N*dp[i-1]}+\frac{(i-1)}{N*dp[i]+1}$
移项整理后可得$dp[i]=dp[i-1]+\frac{N}{N-(i-1)}$

最后的期望步数要加上第一部分构造时所用的步数
