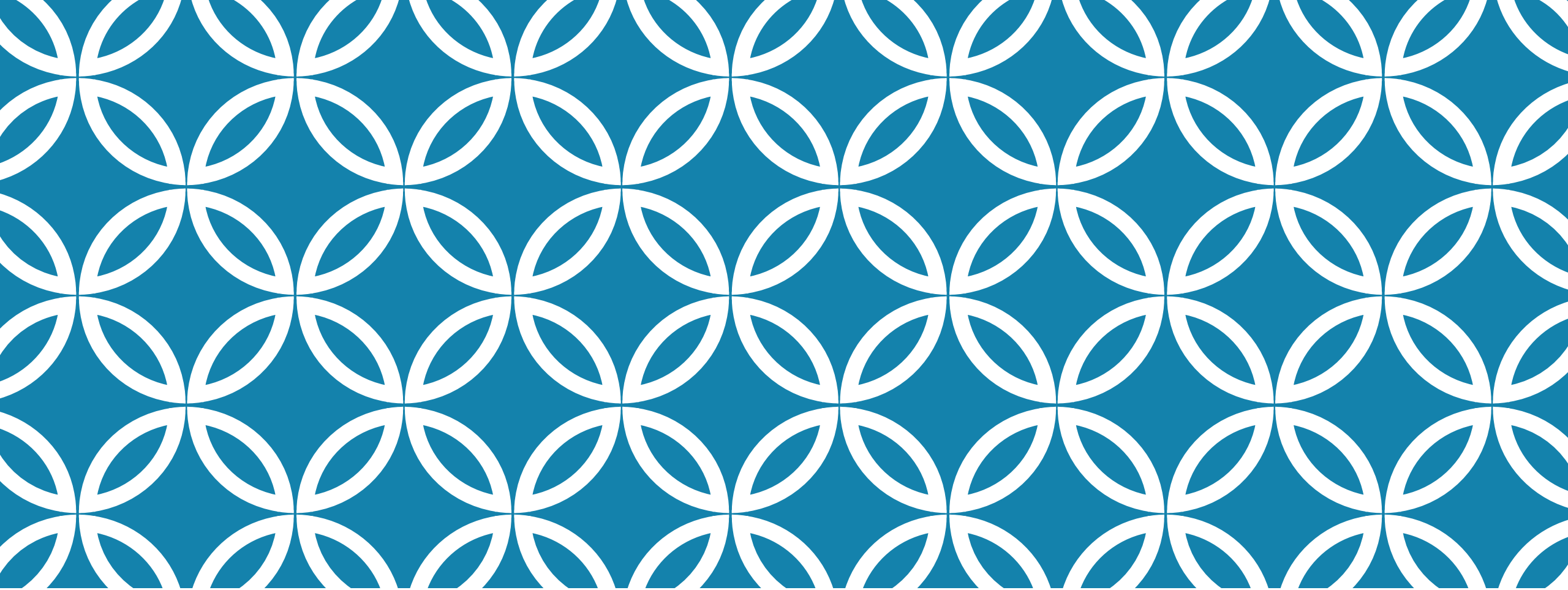

区间DP+树形DP

前言

区间dp，一般需要把一个区间的左右端点作为状态，然后枚举区间的断点将两个相邻的区间合并，原理就是断点分割的两个区间的答案互不干扰，和其他dp没有本质不同。

树形dp，需要对树进行遍历，枚举一个点的子树进行转移，其他的点如果不经过这个点就无法到达这个点子树的节点，这也是一个天然的“隔断”，可以保证状态的答案互不干扰。

树形dp，计数类的问题居多，有时和博弈、树的性质、组合数学等相结合，由于枚举方案数量较大，一般要求取模，而树形dp转移时经常使用除法，注意要使用逆元。

石子归并

有 n 堆石子排成一列，各堆石子个数分别为 $ai(1 \leq i \leq n)$

玩家每次可将两堆相邻石子合并为一大堆，并消耗等于新堆石子个数的代价

求将所有石子合为一堆后，玩家消耗的最小代价。

解题

线性状态dp为何不可行？每次合并的时候不一定前面的石头全都合并完了。

答案与合并顺序有关，但每次合并都是相邻的两堆连续的区间内的石头进行合并。

dp[l][r]表示合并区间[l,r]的石子所需要消耗的最小代价

dp[l][r] = min(dp[l][k]+dp[k+1][r]+sum[r]-sum[l-1])(l<=k<r)

时间复杂度:O(n^3)

选点问题

给定数轴上n个整数点ai，你需要把其中m个点打上标记，每个点所产生的贡献为它到距离它最近的标记点的距离，问这n个点所产生的贡献最小值是多少。

1<=ai<=1e9

1<=m<=n<=500

解题

每个标记点所辐射的范围一定在它周围一段连续的区间内。

dp[i][j][k]表示[i,j]使用k个标记点所产生贡献的最小值。

dp[i][j][k] = min(dp[i][t][k-1]+dp[t+1][j][1])(i<=t<j,2<=k<=m)

对于dp[i][j][1]来说，可以进行预处理，枚举区间内的所有点，然后取一个最小值。

复杂度: O(n^3)

树形DP

以下是树形dp部分。

其实树形dp与区间dp有相通的地方，将树上各点打上dfs序，则一颗子树可以看成是一个dfs序的区间。

最小点覆盖

对于图G=(V,E)来说，**最小点覆盖**指的是从V中取尽量少的点组成一个集合，使得E中所有的边都与取出来的点相连。

也就是说，设V‘是图G的一个顶点覆盖，则对于图中的任意一条边(u,v)，要么u属于集合V’，要么v属于集合V‘。

称G的所有顶点覆盖中顶点个数最少的覆盖为最小点覆盖。

最小点覆盖

为每个节点设立两种状态。

dp[i][0]：表示点i属于点覆盖，并且以点i为根的子树中所连接的边都被覆盖情况下点覆盖集中所包含最少点的个数。

dp[i][1]：表示点i不属于点覆盖，并且以点i为根的子树中所连接的边都被覆盖的情况下点覆盖集中所包含最少点的个数。

dp[i][0]=1+Σmin(dp[u][0],dp[u][1])

dp[i][1]=Σdp[u][0]

最大独立集

对于图G=(V,E)来说，**最大独立集**指的是从V中取尽量多的点组成一个集合，使得这些点之间没有边相连。

也就是说，设V'是图G的一个独立集，则对于图中任意一条边(u,v)，u和v不能同时属于集合V'，甚至可以u和v都不属于集合V'。

称G的所有顶点独立集中顶点个数最多的独立集为最大独立集。

最大独立集

设立两种状态。

dp[i][0]：表示点i属于独立集的情况下，最大独立集中点的个数。

dp[i][1]：表示点i不属于独立集的情况下，最大独立集中点的个数。

$dp[i][0]=1+\Sigma dp[u][1]$

$dp[i][1]=\Sigma max(dp[u][0],dp[u][1])$

最小支配集

对于图G=(V,E)来说，**最小支配集**指的是从V中取尽量少的点组成一个集合，使得对于V中剩余的点都与取出来的点有边相连。

也就是说，设V‘是图G的一个支配集，则对于图中的任意一个顶点u，要么属于集合V’，要么与V‘中的顶点相邻。

称G的所有支配集中顶点个数最少的支配集为最小支配集，最小支配集中顶点的个数称为支配数。

最小支配集

运用树形dp求解，每个点设计三种状态。

dp[i][0]：表示点i属于支配集，并且以点i为根的子树都被覆盖了的情况下支配集中所包含的的最少点的个数。

dp[i][1]：i不属于支配集，且以i为根的子树都被覆盖，且i被其中不少于1个子节点覆盖的情况下支配集中所包含最少点的个数。

dp[i][2]：i不属于支配集，且以i为根的子树都被覆盖，且i没被子节点覆盖的情况下支配集中所包含最少点的个数。

最小支配集

对于第一种状态，子树的根节点无论是什么情况都可以，保证最小就可以。

dp[i][0]=1+Σmin(dp[u][0],dp[u][1],dp[u][2])

对于第二种状态，他的子树中至少有一个要取状态0，但又不知道要取哪个，可以先让每个子树取状态0和1的最小值，如果有一个取了状态0就作罢，否则就找一个替换代价最小的进行替换。需要注意，没有子树的节点该状态要设为INF。

dp[i][1]=Σmin(dp[u][0],dp[u][1]) + (min(dp[u][1]-dp[u][0]) if dp[i][1]==Σdp[u][0])

对于第三种状态，子树的根节点不能是支配点，但必须要被子树的子树支配，因为每个点都要被支配。

dp[i][2]=Σdp[u][1]

树的重心

树的重心也叫树的质心。

找到一个点,其所有的子树中最大的子树节点数最少,那么这个点就是这棵树的重心,删去重心后，生成的多棵树尽可能平衡。

这个东西也是类似于树形dp的求法，你只需要记录每个子树的节点数量，遍历每个节点，它子树的节点数量和除去以它为根节点其他节点的数量取最大值，让这个值尽量小就可以了。

树的直径

树上两个最远的点的距离就是树的直径。

一种做法是随便找一个点u，找到距离u最远的点v，再从v找距离v最远的点w，v到w的距离即是直径。

另一种做法是树形dp，对于每个节点，维护出距离它最远的两个点，这样的两点间距离取最大值就是直径。

没有上司的舞会

有一群职工要参加一个聚会，职工之间的等级关系呈树形结构（每个人可能有多个下属，但至多有一个上司）。

每个职工具有一个“活跃度”表示其参加聚会的快乐程度。

为了聚会气氛轻松，希望有直接上下级关系的两个职工不同时出现在聚会中。

求参加聚会的职工的活跃度之和的最大值。

没有上司的舞会

把员工的关系看成是树，这就是个最大独立集的问题。

把dp[i][0]递推时的1改成这个人的欢乐度就行了。

选课

学校开了N门课，学生可以选M门，这些课学分各不同，但是可能有0到1门先修课，学生学某门课之前必须先把它的先修课学了，问最多可以学多少学分。

每一门课的先修课可以看成是它的父节点，这样就可以看成是一个树形dp的问题了。

现在有一个问题，该如何给某个根节点的子树分配“资源”？

你当然可以弄一个背包现推，但是我们有更省力的办法。

左儿子右兄弟

分配子树资源之所以棘手，是因为这是一颗多叉树，如果是一颗二叉树就好办多了。

要把一颗多叉树转换成二叉树，同时又不破坏原有的节点关系。

重新建立一颗二叉树，某个节点的左儿子指向它原来的儿子节点的其中一个，右儿子指向它下一个兄弟节点（即，它父亲的另一个儿子节点）

这样，如果一个节点选了，他可以分配若干个给它的左儿子节点，一个给它自己，剩下都给他右儿子（也就是它的兄弟）

如果一个节点没选，全都给它右儿子就是了。

`f[i][j]=f[b[i]][j]`（`b[i]`表示右儿子）

`f[i][j]=f[c[i]][k]+f[b[i]][j-k-1]+a[i]`（`c[i]`表示左儿子）

苹果树

有n颗苹果被n-1个树枝连接，有若干个苹果被标记，现在要剪去一些树枝，这样就把整个苹果树分成若干个部分，要求每个部分刚好有1个点被标记，求方案数。

答案对1e9+7取模

N<=1e5

苹果树

按照苹果是不是标记点分类讨论。

令dp[i][0]表示i点及其子树没有标记点的方案数

dp[i][1]表示i点及其子树有标记点的方案数。

i是标记点:

dp[i][0]=0

dp[i][1]=Π(dp[u][0]+dp[u][1])

i是非标记点:

dp[i][0]=Π(dp[u][0]+dp[u][1])

dp[i][1]=Σ(dp[i][0]*dp[u][1]/(dp[u][0]+dp[u][1]))

求解技巧

注意，dp[u][0]+dp[u][1]，这个东西在模运算的意义下可能为0，这样式子进行递推的时候就会出现除以0的情况，你对一个0求乘法逆元，就会求出一堆很奇怪的东西，你的程序就会疯狂WA。

这种情况，就需要我们在不改变方程原意的情况下，改变一下计算顺序。

如果一个节点是标记节点，则不需要考虑这个问题。

如果是非标记节点，每次遍历儿子节点时

dp[i][1] = dp[i][1]*dp[u][0] + dp[i][0]*dp[u][1]

然后

dp[i][0] = dp[i][0]*dp[u][0]

遍历结束后，令

dp[i][0] = dp[i][0] + dp[i][1]