倍增/矩阵快速幂

倍增

如果后继点是唯一确定的，可以预处理出 $f[i][j]$ 表示从 i 出发走 2^j 步会到哪个点，也可以倍增维护出这一过程的一些信息，这样一个走 k 步的过程就可以用 $\mathcal{O}(\log k)$ 步操作模拟出来

P1081 [NOIP2012 提高组] 开车旅行

如果知道当前位置，小 A 和小 B 下一步会到的位置是确定的，可以预处理出来，此时就可以暴力模拟回答这些问题了，但是显然会 TLE

这是一个后继唯一的模拟过程，可以倍增优化，倍增预处理出 $f[i][j][0/1]$ 表示从 i 出发走 2^j 步，是小 A 还是小 B 先走会走到的点，顺便可以维护出这一过程中两个人走的路径长度分别是多少，注意两个 2^0 步拼成 2^1 步是换人的，2^i（$i>0$）时一定是偶数，这时候拼两段的时候不用换人。

对于问题 1，就枚举出发城市倍增求解，对于问题 2，直接从当前起点倍增模拟

P3509 [POI2010] ZAB-Frog

首先把 k 加 1，算上自己到自己

需要求出从每个点跳一次会跳到哪里，可以发现距离一个点前 k 近的点会形成一个区间，距离 1 号点前 k 近的点就是 $1\sim k$，设距离 $i-1$ 号点距离前 k 近的点形成 $[l,r]$ 区间，那么如果 $p[r+1]-p[i]<p[i]-p[l]$，就说明 l 不再是距离 i 前 k 近的点，就把区间推到 $[l+1,r+1]$，这个区间不可能往左推，只能往右推，当然可能可以接着推，用 `while` 来推，此时距离 i 第 k 近的点就是 l,r 其中一个

因为每个点的后继是唯一的，所以这个跳跃过程可以倍增优化

直接预处理倍增数组可能需要 6×10^7 个 $int=240MB$ 空间，想要优化空间可以在算出从每个点出发走 2^i 步能走到哪里之后看 m 这一位是否为 1 的话，是的话就让所有询问当前的答案跳一下，这样在求出 2^{i+1} 步的数组后，2^i 的信息就没用了，倍增数组就可以用滚动数组优化了

CF1175E Minimal Segment Cover

如果不是询问区间 $[x,y]$，而是询问从头到尾完整覆盖，有一个经典贪心做法，把线段按左端点排序，每次选能选的线段（$l\le$ 当前覆盖到的右边界的线段）中 r 最大的，也就是说如果我们记录以每个 l 为左端点的线段，其对应 r 的最大值是多少，设为 $f[l]$，并对其做前缀 $\max$，我们就可以知道如果当前覆盖到了 l，选择完一条线段后就会覆盖到 $f[l]$（此处的 $f[l]$ 指做完前缀 $\max$ 后的 $f[l]$）

这就是个后继唯一的问题了，可以倍增优化，求出从 x 出发跳多少步能第一次跳到 $>y$ 的位置，这个可以通过求出最后一次 $\le y$ 的步数再加 1 得到

矩阵快速幂

[矩阵 - OI Wiki (oi-wiki.org)]

矩阵乘法没有交换律，只有结合律

单位矩阵：$a[i][j]=[i==j]$

矩阵快速幂常见应用：

- 加速形如 $dp[i]=\sum dp[i-j]\times a[j]$ 的递推，构造转移矩阵使答案矩阵 $[dp[i],dp[i-1],\dots,dp[i-j+1]]$ 乘上一个转移矩阵后得到 $[dp[i+1],dp[i],\dots,dp[i-j+2]]$，有时也许需要在答案矩阵后边再维护常数 1 或一些其它辅助信息
- 加速形如 $dp[i][j]=\sum dp[i-1][k]\times a[k][j]$ 的递推，构造转移矩阵使答案矩阵 $[dp[i][1],\dots,dp[i][n]]$ 乘上一个转移矩阵后得到 $dp[i+1][1],\dots,dp[i+1][n]$

矩阵快速幂时间复杂度 $\mathcal{O}(k^3\log n)$，其中 k 是矩阵行、列数，看到 n 特别大又可能是可以写成矩阵乘法形式的 dp 时，可以往矩阵快速幂上想

构建转移矩阵时，原答案矩阵第 u 项贡献给新答案矩阵第 v 项需要乘的系数填到转移矩阵 $[u][v]$ 这个位置

P5789 [TJOI2017] 可乐（数据加强版）

设 $dp[i][j]$ 表示第 i 秒时到达点 j 的方案数（爆炸可以设为 0 号点），这是一个形如 $dp[i][j]=\sum dp[i-1][k]\times a[k][j]$ 的递推，$a[k][j]$ 为 1 当且仅当有从 k 到 j 的路（要考虑上爆炸）

可以发现 $a[k][j]$ 其实就是考虑上爆炸以后的邻接矩阵

P6569 [NOI Online #3 提高组] 魔法值

如果把异或的结果按位拆开，就能写出一个形如 $dp[i][j]=(\sum dp[i-1][k]\times a[k][j])\mod 2$ 的递推，可以矩阵快速幂

这样我们就有了一个 $\mathcal{O}(qn^3log^2a)$ 的做法

如果把所有位的矩阵运算放一起做，其实是在做 $dp[i][j]=dp[i][j]\oplus(dp[i-1][k]\times a[k][j])$，其中 $\oplus$ 表示异或，其中 a 矩阵是个 01 矩阵，用该运算法则计算 a^x 都会得到 01 矩阵

用这个方法做矩阵快速幂看上去很有道理，但严格证明结合律还是有点难度，需要依赖 a 是 01 矩阵，此时也可以把 $dp[i-1][k]\times a[k][j]$ 写成 $dp[i-1][k]\&a[k][j]$

可以做 2×2 的矩阵乘法 $(AB)C$ 和 $A(BC)$ 看每一位的结果是否一样，可以发现确实有结合律

也可以推这样的式子，$(ABC)_{i,j}=\oplus_{x=1}^{q}(\oplus_{y=1}^{p}A_{i,y}\times B_{y,x})\times C_{x,j}$，一般情况下没法把括号拆掉，但如果 C 是 01 矩阵，就可以发现无论 $C_{x,j}$ 是 0 还是 1 都有 $(ABC)_{i,j}=\oplus_{x=1}^{q}\oplus_{y=1}^{p}(A_{i,y}\times B_{y,x}\times C_{x,j})$

同理 $(A(BC))_{i,j}=\oplus_{x=1}^{q}A_{i,x}\times(\oplus_{y=1}^{p}B_{x,y}\times C_{y,j})$ 当 B,C 都是 01 矩阵时也可以拆出这个式子

设 $B_{x,y}\times C_{y,j}=D_y$，其实就是看 $A\times(D_1\oplus D_2)$ 是否等于 $(A\times D_1)\oplus(A\times D_2)$，因为 B,C 都是 01 矩阵所以这两个式子是等的，如果 B,C 不是 01 矩阵就不行

因此在 C 是 01 矩阵的情况下，该矩阵运算方式具有结合律

因此我们可以不按位拆开，直接用这个运算法则构造转移矩阵（其实 $a[k][j]$ 就是邻接矩阵）做矩阵快速幂

此时的时间复杂度是 $\mathcal{O}(qn^3\log a)$ 还是有些风险

对于这种做 q 次矩阵快速幂，且每次做快速幂时转移矩阵都一样的问题，我们倍增可以预处理出该转移矩阵的 2^i 次方，这个过程是 $\mathcal{O}(n^3\log a)$ 的，每次询问只需使用一个 $1\times n$ 的答案矩阵（向量）乘上相应的矩阵完成转移即可，这个过程则是 $O(n^2\log a)$ 的，因此这样做可以使总复杂度变为 $\mathcal{O}(n^3\log a+qn^2\log a)$，就比较安全了

P6772 [NOI2020] 美食家

设 $dp[i][j]$ 表示在第 i 天到了城市 j 的最优解，在不考虑美食节的情况下，有状态转移方程 $dp[i][j]=\max(dp[i-w][k]+c_j)$，遇到美食节就在转移计算完再单独加一下，这样可以直接过 40 分

看到 n 是 10^9，可能要想一下这个式子能不能矩阵快速幂

首先 w 很小，最大只有 5，因此可以构造答案矩阵为一个 $1\times 5n$ 的矩阵 $[dp[i][1],\dots,dp[i][n],dp[i-1][1],\dots,dp[i-1][n],\dots,dp[i-4][1],\dots,dp[i-4][n]]$

乘一个转移矩阵得到

$$[dp[i+1][1],\dots,dp[i+1][n],dp[i][1],\dots,dp[i][n],\dots,dp[i-3][1],\dots,dp[i-3][n]]$$

我们只需要考虑这个内层 $+$ 外层取 $\max$ 的矩阵运算是不是满足结合律，可以手推一下矩阵行、列比较小的情况，发现是满足的，从图论角度理解，其实这个矩阵运算是在推经过 i 步的最长路，而从 u 到 v 经过 i 步的最长路显然可以通过枚举中间点 k 拆成 u 到 k 经过 x 步的最长路拼上从 k 到 v 经过 $i-x$ 步的最长路，具有结合律，同理外层取 $\min$ 也是可以的，相当于是最短路

那转移矩阵 mat 就是首先有 $mat[i][i+n]=0$，然后对于一条从 u 指向 v 边长为 w 的边，有 $mat[u+(w-1)\times n][v]=c[v]$，其它位置是负无穷

可以通过这个运算机制推出相应的单位矩阵，但其实可以不用单位矩阵

接下来考虑美食节，其实就是把这个 dp 过程打断，然后对当前的答案数组（矩阵）某个位置单独加一个数

因此可以每次只用矩阵快速幂推到美食节的日期，加完后再继续推，k 个美食节就把整个过程切成了 $k+1$ 段

此时 $\mathcal{O}(k\times(5n)^3\log t)$ 的时间复杂度算下来就有点爆炸

可以用预处理转移矩阵的 2^i，并把矩阵快速幂修改成一个 $1\times 5n$ 的矩阵乘上预处理出的相应转移矩阵的这种优化方法，把时间复杂度降到 $\mathcal{O}((5n)^3\log t+k\times(5n)^2\log t)$

这样就可以通过了

如果加强一下，让美食节在一段时间里都开，那按切成的段一段一段推矩阵快速幂的时候，就需要在进入美食节时修改一下转移矩阵，美食节结束后再修改一下转移矩阵了