Day16

拓扑排序

仅有向无环图, 过水已隐藏

基环树

外向树无法拓扑排序, 内向树拓扑排序后剩余一个环.

CF1385E

给一个混合图, 既有有向边也有无向边, 定向无向边使得它是一个 DAG.

只看有向边拓扑排序, 无向边连向拓扑序大的点.

DAG 最长链

给一个点带权 DAG, 问选一条链最大权值是多少.

拓扑排序后 DP.

设计状态 f_x 是从 x 出发的最大链权, 转移是用从 x 有一条出边能到达的点 y 更新 x, 使得 $f_x = a_x + \max(f_y)$, 转移顺序需要拓扑序从大到小, 否则会有冲突.

传递闭包

给一个 DAG, 求每个点能到达哪些点.

仍然拓扑排序 + DP, 然后从大到小转移.

边界条件是汇点只能到达自己.

设计状态值 f 是 $\lfloor \frac{n}{64} \rfloor$ 个 64 位整数.

转移是对于任意 x 有出边到达的 y, $f_x = f_x \cup f_y$.

P7480

数轴上有 n 个加油站, 每个加油站 i 对应位置 x_i 和油价 c_i.

一开始没油, 起点有加油站, 油箱容量无限, 一单位油走一单位路, 给一个起点和终点, 求最小花费.

策略是每次到一个有钱更小的加油站, 有单调栈求出每个加油站左右第一个更便宜的加油站, 连边, 边权是 当前油价 $\times$ 距离.

因为一定是向更便宜的点连边, 所以有向图无环, 是 DAG, 跑起点到所有点的最短路后分别更新答案.

最小拓扑序

因为一个 DAG 的拓扑序是不唯一的, 所以一定存在一个方案, 使得拓扑序排出的节点序列字典序最小.

只要在统计时将字典序小的先删除即可.

树上拓扑序计数

统计外向树的拓扑序不同的可能的情况总数.

如果一个点有多个子树, 则两两子树的节点在拓扑序中的先后顺序随意, 所以可以直接在 $Size_x-1$ 个位置中任选 $Size_{Son}$ 个分配给儿子即可, 方案数 $\binom{Size_x-1}{Size_{Son}}$.

对于第二个儿子, 在 $Size_x-1-Size_{Son_1}$ 种中选 $Size_{Son_2}$ 种即可.

所以总的分配方案数是

$$
\begin{aligned}
&\prod_{i=1}^{i\in x_{Son}}\binom{Size_x-1-\sum_{j=1}^{j<i}Size_{Son_j}}{Size_{Son_i}}\\
&=\prod_{i=1}^{i\in x_{Son}}\binom{\sum_{j=i}^{j\in x_{Son}}Size_{Son_j}}{Size_{Son_i}}\\
&=\prod_{i=1}^{i\in x_{Son}}\frac{!\sum_{j=i}^{j\in x_{Son}}Size_{Son_j}}{!Size_{Son_i}\times!\sum_{j=i+1}^{j\in x_{Son}}Size_{Son_j}}\\
&=\frac{!\sum_{i=1}^{i\in x_{Son}}Size_{Son_i}}{\prod_{i=1}^{i\in x_{Son}}!Size_{Son_i}}
\end{aligned}
$$

最后再用乘法原理统计每个儿子内部分配情况即可, 所以 x 子树的总方案数为:

$$f_x = \frac{!\sum_{i=1}^{i\in x_{Son}} Size_{Son_i}}{\prod_{i=1}^{i\in x_{Son}} !Size_{Son_i}} \prod_{i=1}^{i\in x_{Son}} f_{Son_i}$$

拓扑序计数

给一个 n 点 m 边有向图, 每条边可能出现, 对 2^m 种情况, 求每种情况拓扑序方案数之和.

新♂题♂目

对长为 n 的未知正整数序列给 m 个约束条件表示 $a_x \le a_y$, 求 $\min(\max a_i)$.

每个约束条件连边 (x, y), 变成求 DAG 最长链的点数.

拓扑排序, 设计状态 f_i 表示一个点为终点的最长链点数, 按顺序转移, 对于一个点 x, 它可以更新有出边到达的点 y, $f_y = \max(f_y, f_x + 1)$.

答案即为 $\max(f_i)$.

NOI2021

NWRRC2015

给 m 个限制, 描述 y 出现在 x, z 之间.

构造一个 n 的排列至少满足 $\lceil \frac{m}{2} \rceil$ 个限制.

每个限制连接两条有向边 (x, y), (z, y), 每条边有颜色表示是哪个询问的边, 同一个询问的边的颜色相同.

每次选择入度为 0 的点 x, 将它放在排列左端, 然后删除 x 点和所有和它有同样颜色出边的点, 将对应点放到序列右边.

DAG 重链剖分

求一个字符串的字典序第 k 大的子序列的后 p 位.

建子序列自动机, 从后往前转移, 设 f_i 为以第 i 个字符为起点有多少种子序列, DP 求出, 然后 DFS 查找即可找到对应子序列, 然后输出后 p 位.

2-SAT

n 个布尔变量, 每个变量拆成两个点, 表示 `0/1` , 有 m 个条件, 表达了两个变量取 `0` 或 `1` 的情况, 形如 x 取 `0/1` 或 y 取 `0/1` 有且只有一个成立.

针对 x 为 `0` y 为 `1` 至少有一个成立的情况.

x 的 `0` 点向 y 的 `0` 点连边, 表示 x 取 `0` 时 y 取 `0` , 而 x 的 `1` 点向 y 的 `1` 点连边, 另外再连反向边, 以此类推.

对于另一种约束, 即两种情况出现至少一种, 则不连反向边.

对于 x 就是 `0` 这种钦定式约束, 直接将 x 的 `1` 向 `0` 连边, 这样就能代表 x 为 `1` 的情况不合法.

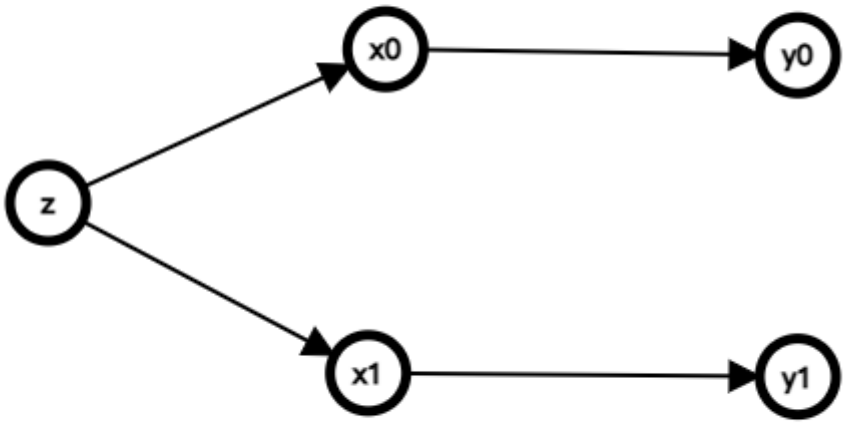
得到了一个有向图, 有向图中任何节点能到达的节点都是只要它本身发生则一定会发生的情况, 所以很显然一个 `1` 点不能到达它对应的 `0` 点, 否则这个点不合法.

得到的图进行缩点, 拓扑排序.

如果某个 `1` 点和对应的 `0` 点在同一个强连通分量内, 则无解.

对于不在一个强连通分量的 `0` 和 `1` , 选择拓扑序大的点所在的强连通分量, 如果这个强连通分量已经不选了, 则不合法.

这时可能会出现一种情况, 可能卡掉算法:



这时假设拓扑序为: $Rank_z = 1, Rank_{x_0} = 2, Rank_{x_1} = 3, Rank_{y_0} = 5, Rank_{y_1} = 4$

这时我们会得出 x_1, y_0 的答案, 不合法. 但是当我们重新审视这组数据, 发现不会存在仅从 x 的点往 y 的点连边的情况, 所以不会出现这种数据.

CF1215F

有 n 个电台, 频段在 $[l(i), r(i)]$ 之间.

由你在 $[1, m]$ 中选频段 f. 区间不包含 f 的电台不能被启用.

给 k 条限制, 有两种:

- 电台 u 和 v 至少选一个
- 电台 u 和 v 不能同时选

输出 f 和选择方案.

区间不交的点不能同时选, 前缀和优化建边, 得到的图跑 2-SAT.

LOJ6036

有 n 个二进制串, 每个串有一位是 ？

你要给 `?` 填上 `0/1`, 使得不存在 i, j 使 s_i 是 s_j 的前缀.

建 Trie, 每个字符串插入, 分别保存 `?` 取 `0` 和 `1` 的情况对应的结尾节点, 一个节点选择后, 它的子树上的节点都不能选, 于是从下往上连边, 跑 2-SAT 即可.

AT2336

数轴上插 n 面旗, 第 i 面要么插 x_i, 要么插 y_i. 使相邻旗子最小间隔最大.

设计每个旗子两个点分别表示选 x_i 和 y_i, 然后二分答案 Ans, 这样每个点 x_i 左后各 Ans 位区间内的点的对应点都应该和 y_i 连边, 跑 2-SAT 判断可行性.

P6898

给定 n 个点的无向带权图, 将点集剖分成 L, R, 最小化 L 内部最大边权与 R 内部最大边权的和.

假设 L 的最大边权更大, 枚举 L 的最大边权 x, 二分 R 的最大边权 y. 对于一个二分到的 (x, y), 对所有大于 x 的边 (u_i, v_i), 连接 u_{iL} 和 v_{iR}, u_{iR} 和 v_{iL}, 但是这样的复杂度达到了 $O(n \log n(n+m))$.

预处理处可行的 x, `random_shuffle()`, 然后二分 y, 假设这个时候答案是 Ans, 对每个 a 先判断 $b = Ans - a$ 的时候是否可行, 否则即使二分 b 得到的结果也不足以更新答案. 这样期望复杂度是 $O(n(n+m))$.

Day17

Ford_Fulkerson

每次从源点 DFS 找增广路, 因为 DFS 可能会每次只找 1 个流量, 而且可能会遍历整个边集, 可能会被卡到 $O(mMax)$.

Edmond_Karp

将 DFS 改成 BFS, 每次找边数最少的最短路,

Dinic

BFS 将图分层, DFS 找增广路, 一次 DFS 走多条增广路. 为了避免走得太远, 选择只能从一个点走到它下一层的点. 复杂度 $O(n^2m)$

但是仅仅是 Dinic 还远远不够, 因为 HLPP 的复杂度非常优秀, 但是由于代码复杂度, 常数还有网络流算法一般不会跑到复杂度上界等原因, Dinic 仍是算法竞赛的首选, 而且不会被卡.

最小割

最小割大小即为最大流大小, 这便是最小割最大流定理.

最大权闭合子图

一个有向图, 选择一个点就必须选择其后继点, 且选择每个点有一个花费或者奖励. 求总奖励最大值.

建 S 连向所有正权点, 负权点连向 T, 权值都是对应点权的绝对值, 假设初始全选所有正权点, 跑最大流.

减掉最小割就是答案.

这样做正确性的保证是一开始假设所有的正权点都选了, 而所有花费也都使用了, 如果有的正权点使用入不敷出, 花费也顶多和收获相等, 这时总的收获中就会将这个点减去, 如果有的点有富余, 那么总收获中减去的值就会剩下净赚. 所以我们的策略相当于将亏的正权点都不选, 只选有净赚的点.

最小费用最大流

过水已隐藏

上下界可行流

每条边有流量上下界, 求最大流.

新建超级源汇 SS, ST.

强制流满每条边的下界, 然后把每条反向边余量设为 0.

如果某个点流量不平衡 (流出 $\neq$ 流入), 则往超级源汇连边, 流入多几个单位就从 SS 往这个点连几个单位容量的边; 流出多几个单位就往 ST 连几个单位的边.

原来的 T 往 S 连容量无穷的边, 在新的图上跑最大流即可找到一个可行解.

上下界最小流

跑一遍可行流, 忽略每条边下界以下的流, 然后从 T 往回反向推流, 最后得到的就是上下界最小流.

上下界最大流

同样的, 从 S 推流即可.

最小费用上下界可行流

仍然是类似于普通可行流, 往超级源汇连边

二分图匹配

匈牙利 $O(mn)$, Dinic $O(m\sqrt{n})$.

过水已隐藏

最大独立集

指图上找出最多的点使得点集中没有任何点之间有边.

二分图的最大独立集大小就是 $n-2\times$ 最大匹配.

最大权匹配

在最大流跑最大匹配的时候给边加权, 跑最小费用最大流 $O(nm)$ 解决. 存在 DFS 的 $O(n^4)$ 的 KM 算法, 还存在 $O(n^3)$ 的 BSF 版本的 KM 算法.

P4001

给一个边带权网格图, 和一般网格图不同的是, 每个网格的左上角和右下角也有连边, 求左上角为源, 右下角为汇的路径上的最小割.

而网格图是一个平面图, 平面图是边两两不相交的图, 通俗地说, 可以画在平面上, 保证两两边不相交的图.

平面图的性质就是以每个边分割出来的区域为点, 相邻的区域之间连边, 边权就是跨过的那条网格图边的边权, 在新图上跑单源最短路即可得到原图的最小割. (原图汇到源连无穷大的边)

p2762

n 个实验, m 个仪器, 每个实验用到的仪器是 m 个仪器的子集, 每个仪器可以用于多个实验, 每个实验有 c_i 收益, 求如何安排实验, 使得去掉仪器花费后的净收益最大.

每个实验和仪器都视为点, 实验的权值为正, 是它的收益, 仪器的收益为负, 是它的花费.

每个实验向对应的仪器连边, 然后建超级源汇 SS, ST, 将 SS 往实验上连边, 仪器的点往 ST 连边, 跑最大闭合子图即可.

P2763

n 道题, 每道题属于至少一种类型, 要求选 m_0 道题组卷, 每个类型要求存在 m_i 道题, $m_0 = \sum m_i$, 求一组可行方案.

每个题和类型分别看成左部点和右部点, 每个题属于哪些类型, 就拆成几个点, 每个点分别向对应类型连边, 每个类型需要多少题, 就拆成多少相同的点, 跑二分图最大匹配即可.

P2764

有向图, 分成若干路径, 每条路径是简单路径且顶点不相交. 求一个划分方案使路径数量尽可能小.

一个划分合法的条件是每个点至多有一个出度, 最多有一个入度, 所以我们只要一个点拆成两个, 一个代表出度, 在左部, 一个代表入度, 在右部. 跑二分图最大匹配即可.

P2765

n 个栈, 依次将正整数压入栈中, 要求栈中任何相邻数字的和为完全平方数. 计算最多压入多少数, 并且给出方案.

二分答案, 暴力枚举所有相加为完全平方数的数连边, 从小到大连单向边.

判断答案可行的时候统计图上的路径条数, 这样就和上一题一样.

P2766

给一个序列.

- 计算最长不降子序列长度 s
- 求最多同时取出多少长度为 s 的不下降子序列
- 如果第一个和最后一个元素可以随意无限使用, 求最多同时取出多少长度为 s 的不同的不下降子序列

对于第一个问题, 直接 DP 即可.

设 DP 时每个点为结尾的最长不下降子序列为 f_i.

第二个问题, 每个点拆成两个, 中间连一条容量为 1 的边, 自己的第二个点往自己后面不比自己小的数 j 并且 $f_j = f_i + 1$ 的第一个点连边, 这样就得到了一个分层图.

新建源点 S, 向每个深度为 1 的点连边, 新建汇点 T, 每个深度为 s 的点向汇点连边.

为了防止一个点两次被用, 我们将一个点拆成两个点, 中间用容量为 1 的边连接, 这样再跑最大流即可.

第三个问题, 为了复用第一个和最后一个元素, 我们只要将这两个元素的点之间的边的容量改成无限即可.

P2774

给一个方格图, 每个格子有权值, 要求选出互不相邻的格子使得权值和最大, 求权值.

先在每对相邻的点之间连边, 得到一个二分图, 然后跑二分图最大权独立集.

例题

有 n 个物品和两个集合 A, B, 如果将一个物品放入 A 集合会花费 a_i, 放入 B 集合会花费 b_i

还有若干个形如 U_i, V_i, W_i 限制条件, 表示如果 U_i 和 V_i 同时不在一个集合会花费 W_i. 每个物品必须且只能属于一个集合, 求最小的代价.

BZOJ3232

一个网格, 边上格子上都有权值, 求一个闭合不相交的回路, 求一个回路使得路径上围起来的格子的权值和的权值和的比值.

发现一个不相交的简单回路围起来的格子一定是四联通的, 所以尝试每个格子之间连边,

二分这个答案, 尝试判断一个答案合不合法, 将格子默认一开始选, 每个格子被删除的花费是它本身的价值, 每个边的价值被计算, 当且仅当它的两个端点一个选一个不选.

P1251

有 n 天, 每天需要 r_i 块餐巾, 每天可以送去快/慢洗部一些餐巾, 花费一定 $morn$ 天是间和 $fors$ 金钱洗完, 也可以新买一批餐巾, 花费金钱 p 立刻得到餐巾.

求保证每天有足够干净餐巾的情况下的最小花费.

每天表示一个点, 源点 S 向每天的点连一条权值为 p, 容量无限的边, 每个点向后面大于等于 $i+morn$ 的天连权值为 $fors$ 的容量无穷的边.

但是为了控制每天的餐巾, 我们把每天的点拆成两个, 中间用免费的容量无穷的边连接, 然后末尾的点连向汇点一条免费的容量 r_i 的点.

为了将脏毛巾留下来, 我们从第二个点往下一天的第二个点连免费无穷边, 这样每天只要往 $i+m$ 和 $i+n$ 连边就可以了.

AT2689

无限硬币, 有 n 枚正面朝上, 每次翻转长度为奇数质数的一个区间, 求使硬币都朝下的最少操作数.

我们将原序列需要翻转的区间情况差分, 得到一个含有偶数个 1 的差分数组.

发现奇质数可以凑出任意偶数 (奇质数的差可以凑出 2, 4, 之和可以凑出很可观的正整数范围内的所有偶数), 而一个奇合数可以被 3 和一个偶数凑出, 一共用了 3 个奇质数.

由此发现每次可以将两个 1 同时消去, 而两个 1 坐标之差决定了消去的花费.

- $j-i$ 是奇质数

花费为 1, 优先考虑, 在差为奇质数的 1 之间连边, 跑二分图最大匹配.

- $j-i$ 是偶数

花费为 2, 在剩余的差为偶数的 1 之间连边, 跑二分图最大匹配, 乘 2.

- $j-1$ 是奇合数

花费为 3, 在剩余的差为奇合数的 1 之间连边, 跑二分图最大匹配, 乘 3.

Hall

???

AT2645

BZOJ2138

一个序列的石子, 每分钟选一个区间 $[L_i, R_i]$ 取 K_i 个石子扔掉, 如果不足 K_i 的

CF1288F

P3980

n 类志愿者, 每类可以在 $[l_i, r_i]$ 日期工作, 价格 c_i 每天. 每天需要 a_i 个志愿者, 求满足所有条件的志愿者数量的最小花费.

Day18: 模拟赛

期望 360 炸成 210, 真不是我假, 考完试就说我挂了 TMD $100+$.

A

一个 ST 题, 放在 T1 还算合理.

但是...

样例诈骗不讲武德, ST 表写炸了样例全过, 结果爆 0.

```
unsigned Lst[2000005], a[2][1000005][20], Pos[1000005][2][2], Bin[1000005], Log[1000005], m, n,
inline unsigned Find(unsigned L, unsigned x, unsigned y) {
  register unsigned Tmp(Log[y - x + 1]);
  return max(a[L][x][Tmp], a[L][y - Bin[Tmp] + 1][Tmp]);
}
int main() {
  n = RD();
  for (register unsigned b(0); b < 2; ++b) {
    for (register unsigned i(1); i <= n; ++i) {
      Lst[++Cnt] = a[b][i][0] = RD();
    }
  }
  sort(Lst + 1, Lst + Cnt + 1);
  unique(Lst + 1, Lst + Cnt + 1);
  for (register unsigned b(0); b < 2; ++b) {
    for (register unsigned i(1); i <= n; ++i) {
      a[b][i][0] = lower_bound(Lst + 1, Lst + n + 1, a[b][i][0]) - Lst;
      if(Pos[a[b][i][0]][0][1]) {
        Pos[a[b][i][0]][1][0] = b;
        Pos[a[b][i][0]][1][1] = i;
      } else {
        Pos[a[b][i][0]][0][0] = b;
        Pos[a[b][i][0]][0][1] = i;
      }
    }
  }
  for (register unsigned i(1), j(0); i <= n; i <<= 1, ++j) {
    Bin[j] = i, Log[i] = j;
  }
  for (register unsigned i(2); i <= n; ++i) {
    Log[i] = max(Log[i], Log[i - 1]);
  }
  for (register unsigned b(0); b < 2; ++b) {
    for (register unsigned i(1); i <= Log[n]; ++i) {
      for (register unsigned j(1); j + Bin[i] <= n + 1; ++j) {
        a[b][j][i] = max(a[b][j][i - 1], a[b][j + Bin[i - 1]][i - 1]);
      }
    }
  }
  for (register unsigned i(n); i; --i) {
    if(Pos[i][0][0] ^ Pos[i][1][0]) {
      Ans = max(Ans, i);
      continue;
    }
    if(Pos[i][0][1] > Pos[i][1][1]) {
      swap(Pos[i][0][1], Pos[i][1][1]);
    }
    if(Pos[i][0][1] + 1 == Pos[i][1][1]) {
      continue;
    }
    Ans = max(min(i, Find(Pos[i][0][0], Pos[i][0][1] + 1, Pos[i][1][1] - 1)), Ans);
```

```
  }
  printf("%u\n", Lst[Ans]);
  return Wild_Donkey;
}
```

B

求一个数 x 和序列中所有数的差的平方和.

这个答案对 x 成二次函数关系, 所以只要用所有数的平方和和总和求出解析式然后 $O(1)$ 求答案即可.

大水题, 本来可以 $O(n)$, 但是数据非要开 3000.

所以我们 $O(n)$ 枚举这个数即可, 总复杂度 $O(n^2)$.

```
unsigned a[10005], m, n, Cnt(0), Sum(0);
unsigned long long Ans(0x3f3f3f3f3f3f3f3f), Tmp, A;
char b[10005];
inline void Clr() {}
int main() {
  n = RD();
  for (register unsigned i(1); i <= n; ++i) {
    Sum += (a[i] = RDsg() + 4000);
  }
  Sum /= n;
  for (register unsigned i(1000); i <= 7000; ++i) {
    Tmp = 0;
    for (register unsigned j(1); j <= n; ++j) {
      A = max(a[j], i) - min(a[j], i);
      Tmp += A * A;
    }
    Ans = min(Ans, Tmp);
  }
  printf("%llu\n", Ans);
  return Wild_Donkey;
}
```

C

我到 AC 都不知道暴力怎么写./kk

线段树 + 二分答案事实上可以优化成 ST + 二分答案, 这样就能将 $O(n\log^2 n)$ 优化到 $O(n\log n)$.

然后我就看着老师三年前的代码拿着 $O(n^3)$ 的复杂度跑得比我 $O(n\log n)$ 都快.

26 个字母的出现情况状态压缩成一个整数, 线段树查询区间的出现情况, $O(26)$ 找出出现字母数量, 然后直接 DP 即可.

```
unsigned f[200005][20], b[200005], Bin[205], Log[200005], m, n, Cnt[30], BinL, BinR, BinMid, A,
char ap[200005], *a(ap);
struct Node {
  Node *LS, *RS;
  unsigned Val;
}N[500005], *CntN(N);
void Build(Node *x, unsigned L, unsigned R) {
  if(L == R) {
    x->Val = b[L];
    return;
  }
  register unsigned Mid((L + R) >> 1);
  Build(x->LS = ++CntN, L, Mid);
  Build(x->RS = ++CntN, Mid + 1, R);
  x->Val = (x->LS->Val) | (x->RS->Val);
  return;
}
inline unsigned Count (unsigned x) {
  register unsigned Ctmp(0);
  for (register char i(0); i < 26; ++i) {
    Ctmp += (x & (1 << i)) ? 1 : 0;
  }
  return Ctmp;
}
void Qry(Node *x, unsigned L, unsigned R) {
  if((A <= L) && (R <= B)) {
    C |= x->Val;
    return;
  }
  register unsigned Mid((L + R) >> 1);
  if(B > Mid) {
    Qry(x->RS, Mid + 1, R);
  }
  if(A <= Mid) {
    Qry(x->LS, L, Mid);
  }
  return;
}
char Judge1 (unsigned x) {
  A = x, B = D, C = 0, Qry(N, 1, n);
  return Count(C) <= m;
}
char Judge2 (unsigned x) {
  A = x, B = D, C = 0, Qry(N, 1, n);
  return Count(C) >= m;
}
inline unsigned Find(unsigned x, unsigned y) {
  register unsigned Tmp(Log[y - x + 1]);
  return min(f[x][Tmp], f[y - Bin[Tmp] + 1][Tmp]);
}
int main() {
```

```
m = RD();
fread(ap + 1, 1, 200002, stdin);
while (a[1] < 'a') ++a;
while (a[n + 1] >= 'a') ++n;
for (register unsigned i(1); i <= n; ++i) {
  b[i] = 1 << (a[i] - 'a');
}
Build(N, 1, n);
for (register unsigned i(1), j(0); i <= n; i <<= 1, ++j) {
  Bin[j] = i, Log[i] = j;
}
for (register unsigned i(1); i <= n; ++i) {
  Log[i] = max(Log[i], Log[i - 1]);
}
NowL = NowR = 0;
for (register unsigned i(1); i <= n; ++i) {
  D = i, BinL = 1, BinR = i;
  while (BinL ^ BinR) {
    BinMid = (BinL + BinR) >> 1;
    if(Judge1(BinMid)) {
      BinR = BinMid;
    } else {
      BinL = BinMid + 1;
    }
  }
  NowL = BinL;
  BinL = 0, BinR = i;
  while (BinL ^ BinR) {
    BinMid = (BinL + BinR + 1) >> 1;
    if(Judge2(BinMid)) {
      BinL = BinMid;
    } else {
      BinR = BinMid - 1;
    }
  }
  NowR = BinL;
  f[0][0] = 0;
  if(NowR <= 0) {
    f[i][0] = 0x3f3f3f3f;
  } else {
    if(!NowL) NowL = 1;
    f[i][0] = Find(NowL - 1, NowR - 1) + 1;
  }
  for (register unsigned j(1); j <= Log[i]; ++j) {
    f[i - Bin[j] + 1][j] = min(f[i - Bin[j - 1] + 1][j - 1], f[i - Bin[j] + 1][j - 1]);
  }
  if(f[i][0] >= 0x3f3f3f3f) {
    printf("-1\n");
  } else {
    printf("%u\n", f[i][0]);
  }
```

```
    }
    return Wild_Donkey;
}
```

发现二分得到的 l, r 单调递增, 所以直接双指针扫描即可, 可以被进一步优化掉, 但是由于查询时仍然需要动态 ST 表, 所以仍然是 $O(n \log n)$.

D

$O(nm2^m)$ 暴力给了 $50'$, 据说 $10min$ 码量, 我没写.

转化为在一个 m 维超空间内的 n 个点中找出每维坐标为 0 或 1 的点使得它到最近的点的曼哈顿距离最小.

既然是找最短曼哈顿距离, 考虑多源 BFS (我竟然没想到, 在我喊出这 5 个字符后一秒钟老师就喊出了这个 5 个字符).

```
unsigned Now(0), Ways(0), Q[1500005], Dist[1500005], Hd(0), Tl(0), m, N, n, Cnt(0), A, B, C, D,
char Faq[104];
int main() {
  n = RD() - 1, m = RD(), N = (1 << m);
  memset(Dist, 0x3f, ((N + 1) << 2));
  for (register unsigned i(1), Tmpi(0); i <= n; ++i) {
    scanf("%s", Faq);
    Tmpi = 0;
    for (register unsigned j(0); j < m; ++j) {
      Tmpi <<= 1, Tmpi += Faq[j] - '0';
    }
    Dist[Q[++Tl] = Tmpi] = 0;
  }
  while (Hd < Tl) {
    Now = Q[++Hd];
    for (register unsigned i(0), Nowi(0); i < m; ++i) {
      Nowi = Now ^ (1 << i);
      if(Dist[Nowi] >= 0x3f3f3f3f) {
        Dist[Nowi] = Dist[Now] + 1;
        Q[++Tl] = Nowi;
        if(Dist[Nowi] > Ans) {
          Ans = Dist[Nowi];
          Ways = 1;
        } else {
          ++Ways;
        }
      }
    }
  }
  printf("%u %u\n", m - Ans, Ways);
  return Wild_Donkey;
}
```

E

写的 $O(n^3m4^n)$ 暴力, 期望 $60'$, 但是发现 $60'$ 的极限数据 `8 64` 本地跑了 $2s$, 所以用打表来卡常. (以表代卡的操作我自己都佩服我能这么不要脸)

实际 $10'$, 原因是代码中有一个 j 写成了 i.

这就是 $60'$ 代码:

```
const unsigned long long _1(1);
const unsigned long long MOD(998244353);
unsigned f[70005], m, n, N, Cnt(0), A, B, C, D, t, Ans(0);
unsigned long long Tmp(0), List[100005];
inline void Print(unsigned long long x) {
  for (register unsigned i(0); i < (n << 1); ++i){
    putchar(((_1 << i) & x) ? '1' : '0');
  }
  putchar('\n');
  return;
}
inline unsigned long long To(unsigned long long x, unsigned long long y) {
  register unsigned long long L;
  while (x) {
    L = Lowbit(x);
    if(y & L) {
      y ^= (L | (L << 1));
    } else {
      y ^= L;
    }
    x ^= L;
  }
  return y;
}
unsigned Ans8[70] = {0, 93, 8649, 804357, 69214125, 112083636, 213453520, 809812580, 188050035,
int main() {
  n = RD(), m = RD();
  if(n > m) swap(n, m);
  if(n == 8) {
    printf("%u\n", Ans8[m]);
    return 0;
  }
  for (register unsigned i(0); i < n; ++i) {
    List[++Cnt] = (_1 << i);
  }
  for (register unsigned i(0); i < n; ++i) {
    for (register unsigned j(i + 1); j < n; ++j) {
      List[++Cnt] = (_1 << i) | (_1 << j);
    }
  }
  for (register unsigned i(0); i < n; ++i) {
    for (register unsigned j(i + 1); j < n; ++j) {
      for (register unsigned k(j + 1); k < n; ++k) {
        List[++Cnt] = (_1 << i) | (_1 << j) | (_1 << k);
      }
    }
  }
  for (register unsigned i(1); i <= Cnt; ++i) {
    for (register unsigned j(n - 1); j; --j) {
      if((_1 << j) & List[i]) {
        List[i] ^= (_1 << j);
```

```
        List[i] ^= (_1 << (j << 1));
      }
    }
  }
  N = 1 << (n << 1);
  f[0] = 1;
  for (register unsigned i(1); i <= m; ++i) {
    for (register unsigned long long k(N - 1); k < 0x3f3f3f3f3f3f3f3f; --k) {
      for (register unsigned j(1); j <= Cnt; ++j) {
        if((List[j] & k) & (((List[j] << 1) & k) >> 1)) {
          continue;
        }
        Tmp = To(List[j], k);
        f[Tmp] += f[k];
        if(f[Tmp] >= MOD) f[Tmp] -= MOD;
      }
    }
  }
  for (register unsigned i(0); i < N; ++i) {
    Ans += f[i];
    if(Ans >= MOD) Ans -= MOD;
  }
  printf("%u\n", Ans);
  return Wild_Donkey;
}
```

然后是正解: 发现每一列的位置不一定要特别讨论, 只要知道当前状态有多少列有 0 个 1, 多少列有 1 个 1, 多少列有 2 个 1, 多少列有 3 个 1 即可, 因为列数是 n 所以这 4 个量知 3 推 1.

设计状态 $f_{i,j,k,l}$ 表示讨论到第 i 列, 有 j 列没有 1, k 列有 1 一个 1, l 有 2 个 1.

转移也比较简单, 只要枚举本层的 1 有几个放在 j 表示的列中, 几个放在 k 表示的列中, 几个在 l 表示的列中, 用排列组合求出不同的情况数, 总共有大约 20 种分配方式.

另可以用滚动数组滚掉第一维, 这样空间复杂度就是 $O(n^3)$, 时间复杂度是巨大常数的 $O(n^4)$.

总结

这次考场代码离 $360'$ 只差 5 个字符, 有 4 个是 T1 的 `+1` 和 `-1`, 还有一个是 T5 把 `i` 改成 `j`.

悔恨的泪水, 流了下来. (所以为什么样例这么菜啊???)

Day19: 模拟赛

A

无向图, 求点 1 到点 n 的, 强制经过某个点或某条边的最短路.

分别从 1 和 n 跑单源最短路, 然后 $O(1)$ 回答询问.

对于强制走点 i, 答案就是 1 到 i 的最短路加 n 到 i 的最短路.

对于强制走边 i, 它的两个端点为 u,.v, 则讨论经过这条边的方向, 取 $min(Dis_{(1,u)} + Dis_{(n,v)}, Dis_{(1,v)+Dis_{n,u}}) + Val_i$ 作为答案.

```
unsigned Pnts[200005][2], m, n, Cnt(0), C, D, t, Ans(0), Tmp(0);
char b[10005];
struct Edge;
struct Node {
  Edge *Fst;
  unsigned long long Dist1, Dist2;
  bool InQue;
}N[100005], *A, *B;
struct Edge {
  Node *To;
  Edge *Nxt;
  unsigned long long Val;
}E[400005], *CntE(E);
struct Pnt{
  Node *P;
  unsigned long long Dist;
  const inline char operator <(const Pnt &x) const{
    return this->Dist > x.Dist;
  }
}TmpP;
void Link (Node *x, Node *y) {
  (++CntE)->Nxt = x->Fst;
  x->Fst = CntE;
  CntE->To = y;
  CntE->Val = C;
}
priority_queue<Pnt> Q;
int main() {
  n = RD(), m = RD();
  for (register unsigned i(1); i <= m; ++i) {
    A = N + RD(), B = N + RD(), C = RD();
    Link(A, B), Link(B, A);
    Pnts[i][0] = A - N, Pnts[i][1] = B - N;
  }
  for (register unsigned i(1); i <= n; ++i) {
    N[i].Dist1 = N[i].Dist2 = 2147483647;
  }
  TmpP.P, N[1].Dist1 = 0, TmpP.P = N + 1, Q.push(TmpP);
  while (Q.size()) {
    register Node *Now((Q.top()).P); Q.pop();
    if(Now->InQue) continue;
    Now->InQue = 1;
    Edge *Sid(Now->Fst);
    while (Sid) {
      if(Sid->To->Dist1 > Now->Dist1 + Sid->Val) {
        Sid->To->Dist1 = Now->Dist1 + Sid->Val;
        TmpP.Dist = Sid->To->Dist1, TmpP.P = Sid->To, Q.push(TmpP);
      }
      Sid = Sid->Nxt;
    }
  }
```

```
  for (register unsigned i(1); i <= n; ++i) {
    N[i].InQue = 0;
  }
  TmpP.P, N[n].Dist2 = 0, TmpP.P = N + n, Q.push(TmpP);
  while (Q.size()) {
    register Node *Now((Q.top()).P); Q.pop();
    if(Now->InQue) continue;
    Now->InQue = 1;
    Edge *Sid(Now->Fst);
    while (Sid) {
      if(Sid->To->Dist2 > Now->Dist2 + Sid->Val) {
        Sid->To->Dist2 = Now->Dist2 + Sid->Val;
        TmpP.Dist = Sid->To->Dist2, TmpP.P = Sid->To, Q.push(TmpP);
      }
      Sid = Sid->Nxt;
    }
  }
  for (register unsigned i(1); i <= n; ++i) {
    printf("%llu\n", N[i].Dist1 + N[i].Dist2);
  }
  for (register unsigned i(1); i <= m; ++i) {
    printf("%llu\n", E[i << 1].Val + min(N[Pnts[i][0]].Dist1 + N[Pnts[i][1]].Dist2, N[Pnts[i][1]
  }
  return Wild_Donkey;
}
```

B

给一个矩阵, `#` 不能走, 走一个和坐标轴平行的线段算一步, 求在每个 `.` 最少需要多少步能走到一个 `+`.

又是多源 BFS, 从 `+` 往外搜, 保证每个点只进一次队, 复杂度 $O(n^2)$.

```cpp
const int maxn = 2005;
const int inf = 0x3f3f3f3f;
char s[maxn][maxn];
int n, m, a[maxn][maxn];
unsigned Que[4000005][2], Hd(0), Tl(0);
int main() {
  memset(a, 0x3f, sizeof(a));
  input::read(n), input::read(m);
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
      input::read(s[i][j]);
      if(s[i][j] == '+') {
        a[i][j] = 0;
        Que[++Tl][0] = i, Que[Tl][1] = j;
      }
      if(s[i][j] == '#') {
        a[i][j] = -1;
      }
    }
  }
  for (register unsigned i(0); i <= n + 1; ++i) {
    a[i][0] = a[i][m + 1] = -1;
  }
  for (register unsigned i(0); i <= m + 1; ++i) {
    a[0][i] = a[n + 1][i] = -1;
  }
  register unsigned Nowx, Nowy;
  while (Hd < Tl) {
    Nowx = Que[++Hd][0], Nowy = Que[Hd][1];
    register unsigned i(Nowx), j(Nowy);
    while(a[Nowx][Nowy] < a[i + 1][j]) {
      if(a[Nowx][Nowy] + 1 < a[i + 1][j]) {
        a[++i][j] = a[Nowx][Nowy] + 1;
        Que[++Tl][0] = i, Que[Tl][1] = j;
      } else {
        ++i;
      }
    }
    i = Nowx, j = Nowy;
    while(a[Nowx][Nowy] < a[i - 1][j]) {
      if(a[Nowx][Nowy] + 1 < a[i - 1][j]) {
        a[--i][j] = a[Nowx][Nowy] + 1;
        Que[++Tl][0] = i, Que[Tl][1] = j;
      } else{
        --i;
      }
    }
    i = Nowx, j = Nowy;
    while(a[Nowx][Nowy] < a[i][j + 1]) {
      if(a[Nowx][Nowy] + 1 < a[i][j + 1]) {
        a[i][++j] = a[Nowx][Nowy] + 1;
```

```
        Que[++Tl][0] = i, Que[Tl][1] = j;
      } else{
        ++j;
      }
    }
    i = Nowx, j = Nowy;
    while(a[Nowx][Nowy] < a[i][j - 1]) {
      if(a[Nowx][Nowy] + 1 < a[i][j - 1]) {
        a[i][--j] = a[Nowx][Nowy] + 1;
        Que[++Tl][0] = i, Que[Tl][1] = j;
      } else{
        --j;
      }
    }
  }
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
      if (a[i][j] == -1) {
          output::print('#');
      } else if (a[i][j] == inf) {
          output::print('X');
      } else {
          output::print(a[i][j]);
      }
      output::print(" \n"[j == m]);
    }
  }
  output::flush();
  return 0;
}
```

C

给一个矩阵, 对于点 (x, y), 如果 $x > y$, 我们说它是危险的, 另外给出 k 个点是危险的.

假设从 $(1, 1)$ 到 (n, m) 的走法有 x 种, 求 $233^x \% 999911659$.

很容易想到 $O(n^2)$ DP, 用欧拉定理和快速幂求答案即可, 可以处理 $10000 * 10000$ 的网格, 可得 $30'$.

```
const unsigned MOD(999911659), MOD2(999911658);
unsigned f[10005][10005], Dan[10005][2], Min, m, n, q, Cnt(0), A, B, C, D, t, Ans(0), Tmp(0);
char b[10005][10005];
unsigned Power(unsigned x, unsigned y) {
  unsigned long long Tmpx(x), Tmpa(1);
  while (y) {
    if(y & 1) {
      Tmpa = Tmpa * Tmpx % MOD;
    }
    y >>= 1;
    Tmpx = Tmpx * Tmpx % MOD;
  }
  return Tmpa;
}
int main() {
  n = RD(), m = RD(), q = RD(), Min = min(n, m);
  if(n == 10000000) {
    printf("539030724\n");
    return 0;
  }
  for (register unsigned i(1); i <= q; ++i) {
    Dan[i][0] = RD() + 1, Dan[i][1] = RD() + 1;
    b[Dan[i][0]][Dan[i][1]] = 1;
  }
  for (register unsigned i(1); i <= Min + 1; ++i) {
    b[i][i - 1] = 1;
  }
  f[1][1] = 1;
  for (register unsigned i(1); i <= n; ++i) {
    for (register unsigned j(1); j <= m; ++j) {
      if(b[i][j]) continue;
      if(!(b[i - 1][j])) {
        f[i][j] += f[i - 1][j];
        if(f[i][j] >= MOD2) f[i][j] -= MOD2;
      }
      if(!(b[i][j - 1])) {
        f[i][j] += f[i][j - 1];
        if(f[i][j] >= MOD2) f[i][j] -= MOD2;
      }
    }
  }
  printf("%u\n", Power(233, f[n][m]));
  return Wild_Donkey;
}
```

接下来看后面的 $20'$, 这时没有额外的危险的点, 只考虑 $x > y$ 的点.

如果之前了解卡特兰数的各种应用, 可以一眼看出是求卡特兰数. 但是很遗憾, 基础数论没学扎实的我怎么可能会组合数学.

卡特兰数, 表示的是在 $n * n$ 的网格上, 从左上角到右下角不越过对角线的走法数量, 也就是本题让我们求的东西. 其通项公式是:

$$C_i = \binom{2i}{i} - \binom{2i}{i+1}$$

接下来考虑证明:

我们把原图旋转 45°, 然后等比例放大 $\sqrt{2}$ 倍, 就相当于每次可以往 45° 方向走 $\sqrt{2}$, 或往 315° 方向走 $\sqrt{2}$, 求从 $(0,0)$ 走到 $(2n,0)$, 且不跨 x 轴的走法数量.

假设有一个跨过 x 轴的走法, 将它从第一次纵坐标为 -1 的点到右端的部分关于 $y=-1$ 翻转, 这时得到了一个从 $(0,0)$ 到 $(2n,-2)$ 的路径.

发现所有从 $(0,0)$ 到 $(2n,-2)$ 的路径翻转后和不合法的从 $(0,0)$ 到 $(2n,0)$ 的路径一一对应, 所以用所有的路径减去不合法的路径就是合法的路径.

没有限制的 $(0,0) \to (2n,0)$ 路径有 $\binom{2n}{n}$ 种; 没有限制的 $(0,0) \to (2n,-2)$ 路径有 $\binom{2n}{n+1}$ 种, 所以就有了一开始的式子.

由于欧拉定理, 我们需要求 $C_n \% 999911659 - 1$, 但是 999911658 是个合数, 所以不能像一般情况一样直接用 Lucas 求, 需要对 999911658 的质因数 2, 3, 4679, 35617 分别求对应的答案, 然后用 ExCRT 合并.

对于 Lucas 定理, 就是将 $\binom{m}{n} \% p$ 中的 n 和 m 进行了 p 进制分解, 然后对每一个 p 进制位的答案进行分别计算, 最后乘起来即可在 $O(p)$ 预处理的前提下 $O(\log_p m)$ 的复杂度计算组合数.

对于 ExCRT, 假设有一些同余方程, 可以通过两两合并得到同余方程组的解.

这里就拿两个同余方程举例:

$$\begin{aligned} x &\equiv r_1 \pmod{b_1} \\ x &\equiv r_2 \pmod{b_2} \end{aligned}$$

转化为一般的方程形式

$$\begin{aligned} x &= kb_2 + r_2 \\ x &= k'b_1 + r_1 \end{aligned}$$

移项联立, 得到:

$$kb_2 - k'b_1 = r_1 - r_2$$

发现可以用 $Exgcd(b_1, b_2)$ 解决这个问题.

这样就可以求出 k, k', 算出 $x = kb_2 + r_2$, 新的 b 是 $lcm(b_1, b_2)$, 新的 $r = x\%b$, 也就合并了两个同余方程.

加上之前的 $30'$ 便是 $50'$.

接下来的 $20'$ 是不保证 $n = m$ 的情况, 显然, $n \leq m$, 因为如果 $n > m$, 则终点就变得危险了.

这时结合 $n = m$ 的证明, 总路径数变成 $\binom{n+m}{n}$, 而不满足条件的条数也很容易表示: $\binom{n+m}{n-1}$. 所以答案是 $\binom{n+m}{n} - \binom{n+m}{n-1}$.

这时已经得到了 $70'$, 而得 $70'$ 却全然没有这么复杂. 因为数据保证随机, 所以几乎不会出现 n 太小的情况, 组合数很大的时候, 很可能是 999911658 倍数, 所以最后对 999911658 后很大概率是 0, 所以对于 n 较大的情况, 只要输出 $233^0 = 0$ 即可.

接下来是最后 $30'$.

在做最后 $30'$ 之前, 我们需要掌握求任意两点间只考虑对角线, 不考虑特殊危险点的情况数, 假设需要求以 (a, b) 为左上角, (c, d) 为右下角的矩形之间的路径数.

这时的总方案数仍然是 $\binom{c-a+d-b}{c-a}$, 因为这个数字只和矩形的长宽有关. 不合法的方案数则需要讨论矩形和对角线的位置关系, 矩形左下角的直角边为 $c - b$ 的等腰直角三角形是危险点, 不合法的方案数应该是 $\binom{c-a+d-b}{d-a+1}$. 这时我们就可以 $O(\log n)$ 地求任何矩形从左上角走到右下角只考虑对角线的合法路线数量, 封装为 $Path_{a,b,c,d}$.

考虑容斥, 设 f_i 表示从 $(0, 0)$ 走到第 i 个特殊危险点的, 不经过其它危险点的走法数量.

对于转移, 取所有路径 $Pa_{0,0,x_i,y_i}$, 然后对所有满足 $x_j \leq x_i \ \& \ y_j \leq y_i$ 减去一个 $f_j \times Pa_{x_j,y_j,x_i,y_i}$ 即可.

为了使转移正常进行, 我们需要对特殊危险点根据坐标排序, 使得调用某个 f 值的时候它已经求出来了.

说明为什么所有 $f_j \times Pa_{x_j,y_j,x_i,y_i}$ 覆盖了所有从 $(0, 0)$ 经过了至少一个其他的特殊危险点到达第 i 个特殊危险点的路径数.

一个点可以到另一个点, 是一个二维的偏序关系, 所以一定不存在一个点在路径中多次出现的情况, 且一个路径经过 x 个特殊危险点的顺序是一定的.

所以我们可以将所有这些不合法路径分成 $0 \to j_1 \to ... \to i$, $0 \to j_2 \to ... \to i$, $0 \to j_2 \to ... \to i ... 0 \to j_{Last} \to ... \to i$.

其中, $0 \to j$ 这部分的路径数是 f_j, f 的定义保证了这其中不会经过除 j 以外的危险点. 而 $j \to ... \to i$ 这部分是不保证经过多少特殊危险点的, 也就是说覆盖了所有不越线的路径, 也就是 Pa_{x_j,y_j,x_i,y_i}, 通过乘法原理合并即可.

这些路径是互不相同的, 所以这样统计不会重复, 而每个要在结果中删除的路径有都能表示成 $0 \to j \to \ldots \to i$ 的形式, 所以也不会漏数. 所以这种统计方式是不重不漏的.

接下来是常数无敌代码 `Super_Mega_Fast.cpp` (是在 AB 班考试的时候的评测高峰期交的, 我也不知道为什么这么快):

吉吉游西藏 统计

满分提交

最快 最短

ID	题目	提交者	结果	用时	内存	语言	文件大小	提交时间
#334208	#303. 吉吉游西藏	Wild_Donkey	100	738ms	1460kb	C++	3.1kb	2021-08-05 10:15:05
#334109	#303. 吉吉游西藏	justin	100	2787ms	1492kb	C++	1.8kb	2021-08-04 23:20:33
#316602	#303. 吉吉游西藏	xuanyiming	100	2819ms	1476kb	C++11	1.5kb	2021-03-26 08:04:53
#89838	#303. 吉吉游西藏	20190127	100	3069ms	1732kb	C++11	1.5kb	2019-02-04 21:14:37
#334098	#303. 吉吉游西藏	rjr	100	4482ms	1652kb	C++11	2.3kb	2021-08-04 22:56:26
#333704	#303. 吉吉游西藏	LiMengtong	100	4535ms	1636kb	C++	2.5kb	2021-08-04 17:45:45
#334005	#303. 吉吉游西藏	jinzhan	100	4541ms	1632kb	C++11	1.7kb	2021-08-04 21:00:09
#333823	#303. 吉吉游西藏	ink	100	4569ms	1636kb	C++	2.3kb	2021-08-04 19:00:43
#334039	#303. 吉吉游西藏	xmz	100	4801ms	3776kb	C++	2.2kb	2021-08-04 21:34:19
#31256	#303. 吉吉游西藏	gaolinxiang	100	5156ms	956kb	C++	2.4kb	2018-08-20 23:08:46

```
#define C(x,y,z) ((((Fac[x]*Inv[y])%(MOD[z]))*Inv[(x)-(y)])%MOD[z])
using namespace std;
inline unsigned RD() {
  unsigned intmp(0);
  char rdch(getchar());
  while (rdch < '0' || rdch > '9') rdch = getchar();
  while (rdch >= '0' && rdch <= '9') {
    intmp = (intmp << 3) + (intmp << 1) + rdch - '0';
    rdch = getchar();
  }
  return intmp;
}
unsigned MOD[5] = {2, 3, 4679, 35617, 999911659}, n, m, k;
unsigned Fac[36000], Inv[36000], f[1005], Ans[4];
struct Pnt{
        unsigned X, Y;
        const inline char operator <(const Pnt &x) const{
                return (this->X ^ x.X) ? (this->X < x.X) : (this->Y < x.Y);
        }
}Pn[1005];
unsigned Power(unsigned long long base, unsigned p, char Pr){
        unsigned long long res(1);
        while(p){
                if(p & 1) res = (res * base) % MOD[Pr];
                base = (base * base) % MOD[Pr];
                p >>= 1;
        }
        return res;
}
void Prework(char Pr){
        Fac[0]=1;
        for(int i=1;i < MOD[Pr];++i) Fac[i] = (Fac[i-1] * i) % MOD[Pr];
        Inv[MOD[Pr] - 1] = Power(Fac[MOD[Pr] - 1], MOD[Pr] - 2, Pr);
        for (register unsigned i(MOD[Pr] - 2); i < 0x3f3f3f3f; --i) Inv[i] = (Inv[i + 1] * (i +
        return;
}
unsigned Lucas (unsigned x, unsigned y, unsigned Pr) {
        register unsigned TmpL(1);
        if(!(x | y)) return 1;
        if(x % MOD[Pr] < y % MOD[Pr]) return 0;
        return Lucas(x / MOD[Pr], y / MOD[Pr], Pr) * C(x % MOD[Pr], y % MOD[Pr], Pr) % MOD[Pr];
}
unsigned Pa(unsigned x, unsigned y, unsigned x2, unsigned y2, unsigned Pr) {
        if((x > y) || (x2 > y2)) return 0;
        if(x2 <= y) return Lucas(x2 - x + y2 - y, x2 - x, Pr);
        register unsigned TmpP(MOD[Pr] + Lucas(x2 - x + y2 - y, x2 - x, Pr) - Lucas(x2 - x + y2
        if(TmpP >= MOD[Pr]) TmpP -= MOD[Pr];
        return TmpP;
}
unsigned Sol(unsigned Pr) {
        Prework(Pr);
```

```
        for (register unsigned i(1); i <= k; ++i) {
                f[i] = Pa(0, 0, Pn[i].X, Pn[i].Y, Pr);
                for (register unsigned j(1); j < i; ++j) {
                        if(Pn[i].Y >= Pn[j].Y) {
                                f[i] += MOD[Pr] - (f[j] * Pa(Pn[j].X, Pn[j].Y, Pn[i].X, Pn[i].Y,
                                if(f[i] >= MOD[Pr]) f[i] -= MOD[Pr];
                        }
                }
        }
        return f[k];
}
void Exgcd(int &x, int &y, unsigned a, unsigned b) {
        if(b == 0) {
                x = 1, y = 0;
                return;
        }
        Exgcd(y, x, b, a % b);
        y -= (a / b) * x;
  return;
}
int main(){
        n = RD() - 1, m = RD() - 1, k = RD();
        if(n > m) {printf("1\n"); return 0;}
        for (register unsigned i(1); i <= k; ++i) {
    Pn[i].X = RD(), Pn[i].Y = RD();
    if(Pn[i].X > Pn[i].Y) --i, --k;
  }
  sort(Pn + 1, Pn + k + 1);
        Pn[++k].X = n, Pn[k].Y = m;
        for (register unsigned i(0); i < 4; ++i) Ans[i] = Sol(i);
        for (register unsigned i(1); i < 4; ++i) {
                int Tmpa, Tmpb;
    Exgcd(Tmpa, Tmpb, MOD[i], MOD[i - 1]);
    Tmpb = (Tmpb * ((long long)Ans[i] - Ans[i - 1]) % MOD[i]) + MOD[i];
    if(Tmpb >= MOD[i]) Tmpb -= MOD[i];
                MOD[i] *= MOD[i - 1];
                Ans[i] = ((long long)MOD[i - 1] * Tmpb + Ans[i - 1]) % MOD[i];
        }
        printf("%u\n", Power(233, Ans[3], 4));
        return 0;
}
```

Day20: 区间 & 树形 DP

区间 DP

一个区间的状态可以通过更短的区间状态转移而来.

P1880

合并石子.

破环为链, 跑区间 DP.

$$f_{i,j} = \min / \max(f_{i,k} + f_{k+1,j} + Sum_{i,j})$$

状态 $O(n^2)$, 转移 $O(n)$, 时间复杂度 $O(n^3)$

SCOI2003

区间 DP, 设计状态 $f_{i,j}$ 表示区间 $[i,j]$ 的最小折叠长度.

$$f_{i,j} = \min(f_{i,k} + f_{k+1,j})$$
$$f_{i,j} = \min(f_{i,i+t-1} + \lfloor \log_{10}(\frac{j-i+1}{t}) \rfloor + 3)(t是[i,j]的整区间)$$

P1220

因为任意时刻关闭的路灯一定是一个连续的区间, 所以设计状态 $f_{i,j,0/1}$ 表示一个区间内的灯都关闭, 最后人在左/右边的花费.

$$f_{i,j,0} = min(f_{i+1,j,0} + Dis_{i,i+1} * (Sum_{1,n} - Sum_{i+1,j}),$$
$$f_{i+1,j,1} + Dis_{i,j} * (Sum_{1,n} - Sum_{i+1,j}),$$
$$f_{i,j-1,0} + Dis_{i,j} * (Sum_{1,n} - Sum_{i+1,j} + Sum_{1,n} - Sum_{i+j}),$$
$$f_{i,j-1,1} + Dis_{j-1,j} * (Sum_{1,n} - Sum_{i+1,j}) + Dis_{i,j} * (Sum_{1,n} - Sum_{i+j}))$$

发现对于第二种情况, 她一定不能得到比第一种情况优的答案, 因为如果它比第一种优 $Dis_{i+1,j} * (Sum_{1,n} - Sum_{i+1,j})$ 那么 $f_{i+1,j,0}$ 的值就不是正确的, 因为我们可以找到一个方案比它更优, 也就是在 $f_{i+1,j,1}$ 的基础上移动到 $i+1$ 上去.

所以我们可以简化转移:

$$f_{i,j,0} = \min(f_{i+1,j,0} + Dis_{i,i+1} * (Sum_{1,n} - Sum_{i+1,j}),$$
$$f_{i,j-1,1} + Dis_{j-1,j} * (Sum_{1,n} - Sum_{i+1,j}) + Dis_{i,j} * (Sum_{1,n} - Sum_{i+j}))$$

对于 $f_{i,j,1}$ 的转移同理.

P4766

NOI2009

将节点按数据值排序, 则 BST 上的一棵子树的节点就是一个连续的区间.

设计状态 $f_{i,j}$ 表示包含 $[i,j]$ 内所有节点的子树的总代价.

$$f_{i,j}=min(K[Arr_k=ArrMin]+f_{i,k-1}+f_{k+1,j}+SumVis_{i,j}*(j-i+1))$$

其意义是选择一个点为根, 如果这个点本来不是根, 就花费 K 让他变成根, 然后统计上左右子树的代价和, 最后所有点的深度 $+1$. (取最小值的时候用 ST 表 $O(1)$ 查询)

状态 $O(n^2)$, 转移 $O(n)$, 预处理 $O(n\log n)$, 复杂度 $O(n^3+n\log n)$.

NOI2017

首先考虑从小到大 DP. 假设 $f_{i,j}$ 表示宽度至多为 i, 高度至少是 j, 可以圈出的最大面积不大于 K 的概率. 然后用较小的 i 和满足面积 $<K$ 的状态转移即可.

树形 DP

在树上, 每个点状态是它所在子树的信息, 然后通过儿子的值转移.

P1352

给一棵树, 求它的最大独立集.

每个点 i 维护 $f_{i,0/1}$, 分别表示 i 的子树选或不选 i 的最大独立集大小.

$$f_{i,0}=\sum_{j}^{j\in Son_i}max(f_{j,0},f_{j,1})$$
$$f_{i,1}=\sum_{j}^{j\in Son_i}f_{j,0}$$

CTSC1997

需求可以组成一个森林结构. 建一个超级根, 连向每一个根, 把选课数量限制 $+1$.

每个点 i 维护一个值 $f_{i,j}$ 表示它所在子树学 j 门课能学到的学分.

$$f_{i,0}=0$$
$$f_{i,1}=a_i$$
$$f_{i,j}=\max(\sum_{j}^{j\in Son_i}f_{j,k_j})(\sum k=j-1)$$

发现转移就是跑完全背包, 所以这是在树形 DP 里套背包, 简称 "树上背包".

P2015

可以将问题转化为选课问题. 然后就可以用上一个题的方法解决了.

HNOI2003

用 $f_{i,0/1/2/3/4}$ 分别表示:

- i 和儿子都需要父亲的消防站覆盖
- i 需要被爷爷的消防站覆盖
- i 和儿子都没有消防站, 被孙子覆盖
- i 上没有消防站, 被儿子覆盖
- 一个节点 i 上有消防站

的最少数目.

$$f_{i,4}=1+\sum_{j}^{j\in Son_i}f_{j,0}$$

$$f_{i,3}=\min(f_{i,4},\min(f_{k,4}-f_{k,1}+\sum_{j}^{j\in Son_i}f_{j,1})(k\in Son_i))$$

$$f_{i,2}=\min(f_{i,3},\min(f_{k,3}-f_{k,2}+\sum_{j}^{j\in Son_i}f_{j,2})(k\in Son_i))$$

$$f_{i,1}=\min(f_{i,2},\sum_{j}^{j\in Son_i}f_{j,2})$$

$$f_{i,0}=\min(f_{i,1},\sum_{j}^{j\in Son_i}f_{j,1})$$

放代码:

```
unsigned a[10005], m, n, Cnt(0), A, B, C, D, t, Ans(0), Tmp(0);
char b[10005];
struct Node {
  Node *Son, *Bro;
  unsigned f[5];
}N[1005];
void DFS(Node *x) {
  register Node *Now(x->Son);
  if(!Now) {
    x->f[0] = x->f[1] = 0;
    x->f[2] = x->f[3] = x->f[4] = 1;
    return;
  }
  register unsigned Sum2(0), Sum1(0);
  while (Now) {
    DFS(Now);
    x->f[4] += Now->f[0];
    x->f[0] += Now->f[1];
    x->f[1] += Now->f[2];
    Sum2 += Now->f[2];
    Sum1 += Now->f[1];
    Now = Now->Bro;
  }
  x->f[2] = x->f[3] = 0x3f3f3f3f;
  Now = x->Son;
  while (Now) {
    x->f[2] = min(x->f[2], Sum2 - Now->f[2] + Now->f[3]);
    x->f[3] = min(x->f[3], Sum1 - Now->f[1] + Now->f[4]);
    Now = Now->Bro;
  }
  register unsigned Min(0x3f3f3f3f);
  ++(x->f[4]);
  for (register char i(4); i >= 0; --i) {
    x->f[i] = min(x->f[i], Min);
    Min = min(x->f[i], Min);
  }
  return;
}
int main() {
  n = RD();
  for (register unsigned i(2), j; i <= n; ++i) {
    j = RD();
    N[i].Bro = N[j].Son;
    N[j].Son = N + i;
  }
  DFS(N + 1);
  printf("%u\n", min(min(N[1].f[2], N[1].f[3]), N[1].f[4]));
  return Wild_Donkey;
}
```

P3177

树上背包复杂度

假设要求 n 个点的树, 选 k 个节点的树上背包问题.

每次合并所有 x 个子树看作将子树两两合并共合并 $x-1$ 次.

每次合并 x 和 y, 枚举 x 中选 $min(Size_x,k)$ 个, 所以每次最多是 $O(k)$ 单次合并.

而每个点的子树最多被合并 1 次, 所以总复杂度是 $O(nk)$, 而 $k\leq n$, 所以较松的界是 $O(n^2)$ 的复杂度.

ZJOI2007

给一个带权树, 每次将一条边的边权 $+1$, 要求根到每个叶子的距离相等最少的操作数.

每个节点存 f_i 表示它到子树中所有叶子的距离相等的最少操作数, g_i 表示一个节点的子树的最远的叶子到它的距离.

$$f_i=\sum_{j}^{j\in Son_i} Max_g-g_j+f_j$$

```cpp
unsigned a[10005], m, n, Cnt(0), A, B, C, D, t;
unsigned long long Ans(0);
struct Edge;
struct Node {
  Node *Fa;
  Edge *Fst;
  unsigned long long Dep;
}N[500005], *S;
struct Edge {
  Node *To;
  Edge *Nxt;
  unsigned Val;
}E[1000005], *CntE(E);
inline void Link(Node *x, Node *y) {
  (++CntE)->Nxt = x->Fst;
  x->Fst = CntE;
  CntE->To = y;
  CntE->Val = C;
}
void DFS(Node *x) {
  register Edge *Sid(x->Fst);
  register unsigned long long Max(0);
  while (Sid) {
    if(Sid->To != x->Fa) {
      Sid->To->Fa = x;
      DFS(Sid->To);
      Max = max(Sid->To->Dep + Sid->Val, Max);
      x->Dep = max(x->Dep, Sid->To->Dep + Sid->Val);
    }
    Sid = Sid->Nxt;
  }
  Sid = x->Fst;
  while (Sid) {
    if(Sid->To != x->Fa) {
      Ans += Max - Sid->To->Dep - Sid->Val;
    }
    Sid = Sid->Nxt;
  }
}
int main() {
  n = RD(), S = N + RD();
  for (register unsigned i(1); i < n; ++i) {
    A = RD(), B = RD(), C = RD();
    Link(N + A, N + B);
    Link(N + B, N + A);
  }
  DFS(S);
  printf("%llu\n", Ans);
  return Wild_Donkey;
}
```

NOIO2020

Day21: 状压 DP

状态压缩

状态是一个 k 进制数, 将每个元素的状态用每个 k 进制位的值表示.

USACO 06 Nov

一个矩形, 分成 $n * m$ 个格子, 有的格子能选, 有的不能选, 要求选择的格子两两不相邻, 求合法的选取方式数对 10^8 取模的余数.

设计状态 $f_{i,j}$, 表示从上到第 i 行, 第 i 行的状态是 j 的二进制表示的.

我们可以先预处理出合法的行状态, $List_i$ 表示第 i 个没有两个格子相邻的行状态.

$$f_{i,j} = \sum_{k}^{k \in List, k\&j=k\&a_i=0} (f_{i-1,k})$$

注: a_i 是棋盘第 i 行状态.

NOI2001

仍然先预处理出合法状态集合 $List$, 然后设计状态 $f_{i,j,k}$ 表示算到第 i 行, 第 i 行状态为 j, 第 $i-1$ 行状态为 k 的情况.

$$f_{i,j,k} = \sum_{l}^{l \in List, l\&j=l\&k=l\&a_{i-2}=0} f_{i-1,k,l}$$

SDOI2009

设计状态 $f_{i,j,k}$ 表示在 i 前面, j 表示的人先拿饭, 最后拿到饭的人是 $i-7+k$ 的最少时间.

$$f_{i,j,k} = min(f_{i-7+k,j<<(7-k),l}) \ (k < 7)$$
$$f_{i,j,k} = min() \ (k > 7)$$

AHOI2009

在棋盘上放置中国象棋中的炮, 使得没有炮能互相攻击, 求方案数.

问题转化为往棋盘上放点, 使得不存在三个点在一行或一列中.

状态 $f_{i,j,k}$ 表示到第 i 行, 有 j 列没有炮, k 列有 1 个炮.

$$f_{i,j,k}=\sum_{a,b}^{a+b\le k}\binom{j+a}{a}\binom{k+b-a}{b}f_{i-1,j+a,k+b-a}$$

P5005

在棋盘上放置中国象棋中的马, 使得没有马能互相攻击或单向攻击, 求方案数.

仍然状压两位, $f_{i,j,k}$ 表示到第 i 行, 第 i 行状态为 j, 第 $i-1$ 行状态为 k 的状态.

$$f_{i,j,k}=\sum_{l}f_{i,k,l}$$
$$(j<<1\ \&\ l)|(j\ \&\ l>>1)|k=k,$$
$$(j>>1\ \&\ l)|(j\ \&\ l<<1)|k=k,$$
$$((j<<2\ \&\ k)>>1)|k=k,$$
$$((j<<2\ \&\ k)>>1)|j=j,$$
$$((j>>2\ \&\ k)<<1)|k=k,$$
$$((j>>2\ \&\ k)<<1)|j=j,$$

TJOI2015

每个点的范围是 $3*p$, 仍然求放点使其不能攻击的方案数.

因为转移规则是一个矩阵, 所以不方便使用位运算判断, 用转移矩阵对状态进行转移.

新建一个 2^n*2^n 的矩阵, 用矩阵快速幂加速转移.

NOIP2017

给一个无向图, 求一个生成树使得代价最小, 一个生成树的代价是边的代价之和, 每条边的代价是边的权值和边下端点深度的积.

这个题 Prim 被 Hack 了, 详情见 RQY 的数据.

P3943

一个长为 n 的 `0/1` 串, 有 k 个 0, 每次允许取反特定长度的区间, m 种区间长度. 求最少取反几次得到 n 个 1.

SCOI2005

往棋盘上放国际象棋国王, 使其不能互相攻击, 求方案数.

仍然是状态 $f_{i,j}$ 表示到第 i 行, 第 i 行状态为 j 时的方案数.

$$f_{i,j} = \sum_{k}^{(k|(k<<1)|(k>>1))\&j=0} f_{i-1,k}$$

Day22: