2020 多校训练第八场题解

华东师范大学

2020 年 8 月 13 日

花絮

- 曾用名: 2020 多校训练第~~六~~七场
- 曾用名 2: 2020 多校训练第~~七~~八场
- 预估难度:
  - 签到: C F
  - 简单: A G H I
  - 中等: B D J K L
  - 难: E
- 实际情况: C > F > H > I > D > K > L > B > G > J > A > E
- 这是敝校第二次出多校. 由于出题人都很忙 (摸), 导致出题过程断断续续拖了好久 (如曾用名所示). 好在最后并没有夭折.
- ~~比赛前一天晚上还在修锅, 似乎成了我们出题的传统艺能.~~ 比赛延期, 比赛前夜只说不存在了.
- 实际上由于出题人习惯了肆无忌惮的出题方式, 写了一大堆 SPJ 题. 而 hdu 老旧的 Special Judge 机制, 导致花费很多时间在题目的转移上.
- 有几个题目是从昨晚的 Codeforces Round 筛选下来后回收利用的.
- 如果你观察到了 K 题的首字母缩写, 你可能能够获得提示加成.

Auto-correction

首先不要想着把相邻段合并起来得到答案序列, 因为这是错的.

将标注数据应用到原序列 s 上后会得到一个目标序列 t, 问题就转化成一个只有替换操作的编辑距离问题. 记 $f(i,j)$ 为 $s[1..i]$ 变换为 $s[1..j]$ 的最少替换个数和最少分段数组成的二元组, 则有,

$$f(i,j) = \min \begin{cases} \min_{1\le i'<i, 1\le j'<j}\{f(i',j') + (i-i',1)\} \\ f(i-1,j-1)+1 \text{ for } a_i = b_j \end{cases}$$

朴素实现复杂度为 $O(n^4)$. 但是这个式子很容易使用前缀 min 优化, 记

$$G(i,j)=\min_{1\le i'\le i,1\le j'\le j}\{f(i',j')+(-i',0)\}$$

$G(i,j)$ 就是 $g(i,j)=f(i,j)-(i,0)$ 的前缀 min. 那么,

$$f(i,j)=\min\begin{cases}G(i-1,j-1)+(i,1)\\ f(i-1,j-1)+1 \text{ for } a_i=b_j\end{cases}$$

复杂度即降低到 $O(n^2)$. 转移的时候可以把对应的坐标也带上, 求方案应该还是比较方便的.

Tips

如果觉得这题的输入很恶心难以处理的话, 说明读入水平有待提高, 需要多加练习.

Breaking Down News

考虑用 dp 来做. 令 $f[i]$ 表示前 i 个数分成若干个区间的最大答案.

很容易得到转移方程:

$$f[i]=\max\{f[j]+(s[i]>s[j])-(s[i]<s[j])\}$$

, 其中 $s[i]$ 表示前缀 i 个位置的前缀和

维护的方式, 可以考虑按照 s 建立线段树, 需要在线段树的每一个节点上存一个单调队列

考虑 dp 转移的过程中, 每次都需要将 $f[i-l]$ 加入线段树, 更新 $f[i]$, 然后将 $f[i-r]$ 删除.

时间复杂度 $O(n\log n)$

Clockwise or Counterclockwise

显然, 如果点 C 在向量 $\vec{AB}$ 的右侧, 则方向为顺时针, 否则为逆时针. C 在 $\vec{AB}$ 的右侧, 当且仅当 $\vec{AB}\times\vec{BC}<0$. 故我们只需要判断这两条向量的叉积的正负性即可.

Discovery of Cycles

我们可以给每一条边 i 求出以这条边作为左端点, 在最短的区间内能形成环的最小右端点, 标记为 R_i, 如果不存在这样的右端点（即从当前到结尾所有边都不能组成环）, 则让 $R_i=m+1$, 显然暴力求这个 R_i 是不允许的.

显然 R_i 一定是递增的, 所以如果我们可以快速删边的话, 那么这个就能快速的求解了. 于是就考虑到可以用 Link-Cut-Tree 来优化这个过程, 即只需要维护左端点和右端点, 肯定会尽量拓展右

端点, 拓展不了就删掉左端点所在的边然后移动左端点. 不断重复这个过程就可以求解所有的 R_i 了.

时间复杂度 $O(m\log n+q)$.

Easy NPC Problem

无解

n, m 均为奇数 在对网格图黑白染色之后, 被挖掉的格子必须和角落上的格子同色. 即被挖掉的格子到网格图上边界、左边界的距离奇偶性必须相同.

n, m 其中至少一个是偶数 同样将网格图黑白染色之后, 起点和终点颜色必须不同, 否则必定无解.

构造

n, m 均为奇数 除了 $n=1$ 的情况, 其余情况其实均能构造出有哈密顿回路（就无需考虑起点在哪）. 于是可以首先特判 $n=1$ 的情况.

考虑被挖掉的格子不在边界上的情况: 则把网格图除了被挖掉的格子外的所有格子分为四个矩形, 每个矩形均为奇数 $\times$ 偶数, 然后再考虑将矩形之间串起来即可. 而被挖掉的格子在边界上的情况, 就相对好处理的多了.

n, m 其中至少一个是偶数 对于起点 (S_x,S_y) 和被挖掉的位置 (N_x,N_y) , 不妨假设 $N_x\le S_x$, m 是偶数（其余的情况, 均可以通过旋转、对称等规约到这种情况）

由于 m 是偶数, 所以可以对于 $x<N_x$ 的部分跑出一个哈密顿回路, 对 $x>S_x$ 的部分也跑出一个哈密顿回路. 那么接下来的问题就是考虑对 $[N_x,S_x]$ 这些行跑出一个哈密顿路, 最后将上述的三部分连接起来.

连接的方法比较简单, 假设可以在分界的边界上找到这样的四个位置满足 $(x,y)-(x,y+1)$ 且 $(x+1,y)-(x+1,y+1)$, 只需要将它们的连接改为 $(x,y)-(x+1,y)$ $(x,y+1)-(x+1,y+1)$, 这样就能使两部分连接起来了. 因为对于想要连接起来的两部分, 一定有其中一部分是回路, 所以连接起来的结果肯定是一条哈密顿通路.

在偶数行的情况下构造哈密顿回路的方式是简单的. 于是现在考虑对于 $[N_x,S_x]$ 部分构造哈密顿通路的方法, 这里考虑用增量法构造. 考虑每次可以使用两行连接不同断的通路, 即 $(x,y)-(x+1,y)-(x+1,y+1)-\cdots-(x+1,n)-(x+2,n)-(x+2,n-1)-\cdots-(x+2,y)-(x+2,y-1)-(x+1,y-1)-(x+1,y-2)-\cdots-(x+1,1)-(x+2,1)-(x+2,2)-\cdots-(x+2,y-2)$

而显然用这种方法, 在构造出 x 行的情况下, 可以顺利的推到 $x+2$ 行的情况, 而对于 x 比较小的时候, 需要用插头 dp 来得到一个合法的解.

不过需要注意的是, 当 m 较小的时候, 不一定能找出将上下接起来的那种边, 故需要把 n m 互换, 拿 n 小的时候的方法来做.

Fluctuation Limit

如果 i 是在 $[l,r]$ 范围内, 那么 $i+1$ 必须要在 $[l-k,r+k]$ 范围内. 这是因为如果 $i+1$ 选了范围外的值, i 就无解了.

这样可以从左往右, 把左边的约束带到右边. 再从右往左做一遍. 最后剩下的区间应该就是可行域. 因为题目只要求一种方案, 全部选最低的即可. 复杂度 $O(n)$.

Gaming of Co-prime Disallowance

Tips

中间的过程是可以用计数的, 所以如果中间就用了分数计算的话, 会导致精度爆炸.

解法一

很容易想到用状态压缩 dp 来做, 即记录当前已经取的牌情况. 此时的时间复杂度是 $O(n2^n)$.

但是可以发现, 一个状态只和当前已经取的牌的 gcd 和已选牌数量有关, 于是考虑 dp 状态只记录这两个信息进行转移. 此时的时间复杂度是 $O(n^2 max\{a_i\})$.

而实际上这样做 dp 的状态个数远远不到 mn, 真正涉及到的状态个数是 $\sum c(a_i)$, 其中 $c(x)$ 表示数 x 的约数个数. 我们可以将 a_i 中所有相同的质因数都去掉, 因为 $a_i \leq 10^5$, 所以实际涉及到的状态个数最多只有 $2^6 n$. 于是我们可以将时间复杂度做到 $O(2^6 n^2)$

解法二

首先, 如果 $\gcd(a_1,\dots,a_n) \neq 1$, 那么可以简单处理. 因此考虑 $\gcd(a_1,\dots,a_n) = 1$ 的情况.

一局进行了 m 轮的游戏可以用一个长度为 m 的序列 $b_1,b_2,\dots,b_m$ 表示, 其中 b_i 表示, 第 i 轮, 操作的人拿了 a_{b_i}. i 为奇数, 就是 QQ 拿的, 否则是 LF 拿的. 如果这个游戏要 QQ 赢, 则要求:

- m 是奇数.
- $\gcd(b_1,\dots,b_m) = 1$ 且 $\gcd(b_1,\dots,b_{m-1}) \neq 1$.

因此这就转化为一个序列计数的问题. 我们统计所有合法的序列, 把这些序列出现的概率加起来, 就是 QQ 赢的概率. 而一个长度为 m 的序列的出现概率显然是 $\frac{1}{n^{\underline{m}}}$. 因此我们现在有一个暴力搜索序列然后统计概率和的做法.

注意到 $\gcd(b_1,\dots,b_{m-1}) \neq 1$ 的条件, 而 a_i 的范围很小. 考虑容斥. 设一个长度为 m 的序列的权值为

$$f(m) = \frac{1}{(n-1)^{\underline{m}}}$$

那么我们可以统计所有 $\gcd(b_1, \ldots, b_m) = k \quad (k > 1)$ 且 m 是偶数的序列的 f 值的和, 假设为 $g(k)$. 那么我们再统计满足 $a_i \perp k$ 的 i 的个数, 设为 c_k. 那么这部分序列对答案的贡献就是 $\frac{1}{n}g(k)c_k$. 怎么理解这个过程: 相当于你选了 m 个数, gcd 为 k. 然后你最后选一个与 k 互质的数, 就终结了游戏. 而这个 f 值乘上 $\frac{1}{n}$ 正好就是长度为 $m+1$ 的序列出现的概率.

因此问题转化成了, 如何计算 $g(k)$: $\gcd(b) = k$ 且 $|b|$ 为偶数的序列的 f 值的和.

套路容斥, 设 $G(k) = \sum_{k|x} g(x)$. $G(k)$ 的含义等价于: b_i 都是 k 的倍数, 且 $|b|$ 为偶数的序列的 f 值的和. 这就很好求了. 设 a 序列中 k 的倍数有 t_k 个, 那么有

$$G(k) = \sum_{i=1}^{\frac{t_k}{2}} f(2i)(t_k)^{\underline{2i}}$$

计算 t_k 的复杂度是 $O(w \log w)$ 的, 计算 $G(k)$ 的复杂度是 $O(w)$ 的.

在计算完 G 后, 根据 $g(k) = G(k) - \sum_{k|x, k<x} g(x)$, 就可以从大到小枚举 k, 计算出 $g(k)$. 复杂度是调和级数 $O(w \log w)$. 算出 $g(k)$ 了, 然后就可以计算答案了.

注意, 不要直接计算 $g(1)$, 因为它没有保证你在拿最后一张牌的时候让 gcd 变成 1.

Hexagon

构造方式如图 1 所示.

Isomorphic Strings

哈希.

枚举 n 的约数 k, 那么你可以 $O(\frac{n}{k})$ 求出 $s_1, \ldots, s_k$ 的哈希值. 同时你可以 $O(k)$ 求出 s_1 的循环同构串的哈希值.

因此问题转化为, 给出两个数集 S, T, 判断是否有 $S \subseteq T$. 这并不难, 把哈希模数设成 10^7 级别的, 开一个数组就可以 $O(|S| + |T|)$ 判了.

总复杂度 $O(\sigma(n)C)$. 其中 $\sigma(n) = \sum_{d|n} d$, C 为哈希模数个数.

Jumping on a Cactus

考虑给边定向. 对于一条边 $(u, v) \in E$, 如果 $d(u, e) < d(v, e)$, 我们就定向为从 u 到 v 的有向边. 否则就是从 v 到 u 的有向边. 那么原问题可以描述为, 有向偶环仙人掌的拓扑序计数问题.

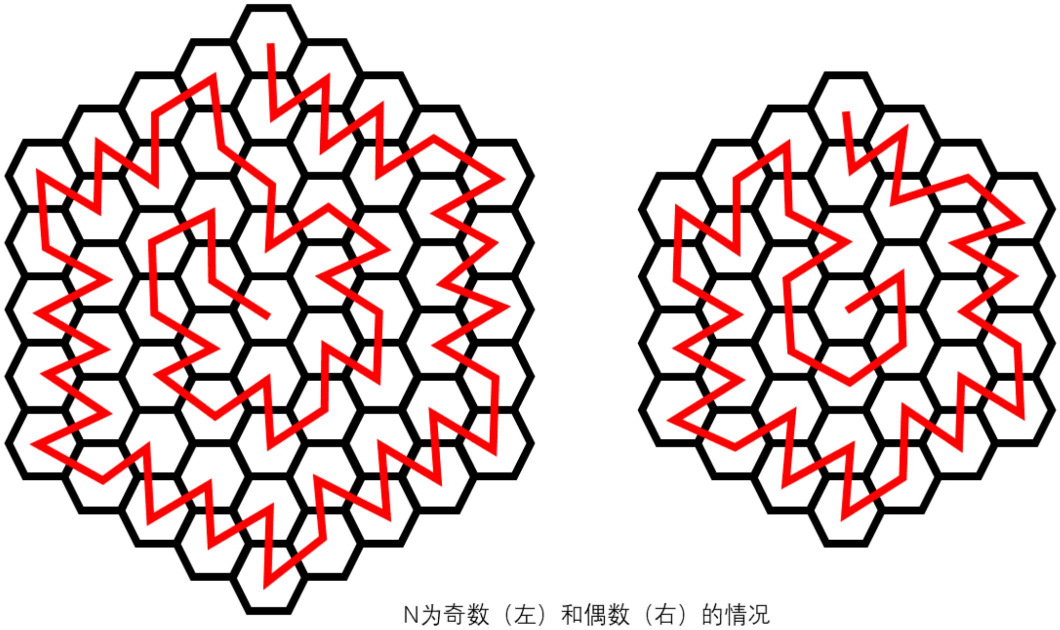

图 1: Solution of Hexagon

考虑弱化情况. 如果 G 是一棵以 e 为根的有根树, 那么如何计算它的拓扑序? 可以树形 DP 在 $O(n)$ 的时间内计算. 但计数算法的细节较多, 因此我们考虑概率算法. 换言之, 我们想求的是:

- 从 $n!$ 个排列中等概率随机一个排列 $p_1, p_2, \ldots, p_n$, 求 p 是 G 的合法拓扑序的概率.

如果我们算出了这个概率, 假设为 P. 那么合法拓扑序的个数就是 $n!P$. 接下来考虑怎么求 P.

经验丰富的选手可以很快知道 $P = \prod_u \frac{1}{S_u}$, 其中 S_u 表示 u 的子树大小. 如果已经知晓, 就可以可以跳过下面的部分.

有根树求 P

一个合法拓扑序可以理解为是给有根树 G 的每个节点标号, 使得

- 对于任何节点 u, u 的标号都是 u 子树中的标号的最小值.

因此, 等概率随机排列, 等价于等概率随机标号, 且每个节点标号互不相同. 但是给 n 个节点随机标号显然不是 n 个互不相关的事件, 因为它要求标号互不相同. 不妨考虑, 把标号由 n 的排列改为 n 个 $[0,1]$ 中的**实数**. 为什么这样做是对的? 因为你在 $[0,1]$ 的实数中随机两个数 x, y, $x = y$ 的概率是 0.

这样, 我们就摆脱了标号互不相同的限制. 问题转化为: 从 $[0,1]$ 中随机 k 个实数 $x_1, \ldots, x_k$, 求 x_1 是这 k 个数的最小值的概率.

显然, 它等于 $\frac{1}{k}$. 因为这 k 个数是平等的, 每个数都有概率成为最小值.

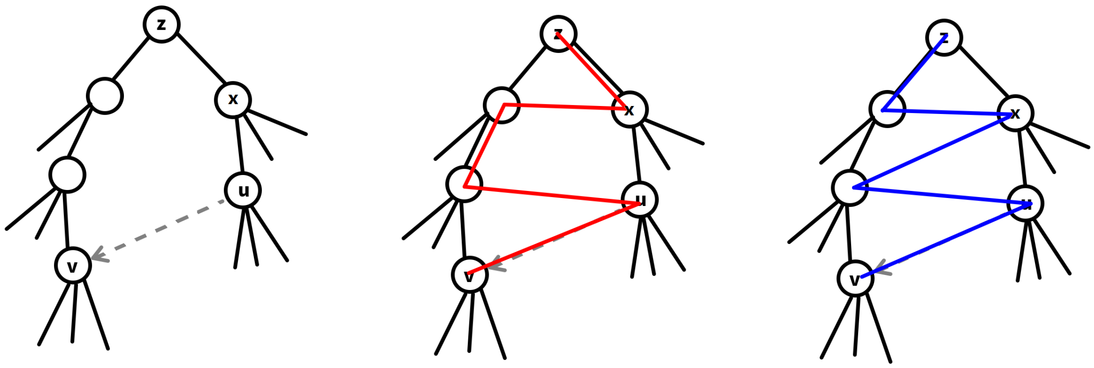

图 2: Solution of Cactus: Loop-based Tree

所以要求 P, 只需要把每个结点的概率乘起来即可.

基环树求 P

对于 G 是有根基环树的情况, 仍考虑计算概率.

如果节点 u 不是环上的点, 或者它是环上深度最浅的点, 它的概率仍为 $\frac{1}{S_u}$. 这里我们规定有向无环图的 S_u 表示, 从 u 出发, 能到达的点的数量.

考虑环上的点的概率怎么求.

如图 2. 虚线边表示不在 DFS 生成树上的边, 它对应了以 z 为顶点的那个环. 假设我们要求节点 x 的概率. 那么首先我们得搞清楚, 哪些节点在 x 之后被访问. 首先 x 的子树里的节点显然在它之后被访问, 但与此同时, 环上另一些点所代表的子树与 x 的先后关系是不确定的. 因此我们做的事情相当于是把这个环给当作两个序列, 将其合并成一个序列, 这样就变环为链, 可以按照树的方式计算概率了. 而变环为链的方案由很多种, 所以我们把每种情况的概率加起来就是这个环上的节点的概率.

注意, 与树不同的是, 树的情况我们可以每个节点分别统计概率然后相乘. 但是对于一个环, 我们统计的是这个环上的节点的概率, 也就是说我们把环当作一个整体（缩点）, 但也仅把环当作一个整体（环上节点的子树的和环是不相关的）. 更通俗地说, 我们关心的只是图上画出来的点的概率, 没被画出来的子树部分的点的概率与环是独立的.

好的, 那么怎么求这个环的概率呢? $O(n^2)$ DP 即可. 我们知道一个环可以用三元组 (u, v, z) 标记, 表示 (u, v) 这条非树边对应的环, 且环上深度最浅的点是 z. 那么不妨设 z 到 u 路径上的点按深度从小到大排序分别为 $a_1, a_2, \ldots, a_l$（不含 z）, 设 z 到 v 路径上的点按深度从小到大排序分别为 $b_1, b_2, \ldots, b_k$（不含 z）.

设 $f(i, j)$ 表示把 $a_i, \ldots, a_l$ 和 $b_j, \ldots, b_k$ 合并成链的概率和. 那么有

$$f(i,j) = \frac{1}{S_{a_i} + S_{b_j}}(f(i,j-1) + f(i-1,j))$$

边界部分是小的细节问题, 不赘述. 则这个环的概率就是 $f(1,1)$.

仙人掌求 P

会基环树了, 自然就会仙人掌了. 仙人掌的环互不相交, 因此可以分别计算概率然后乘起来.

算法复杂度 $O(n^2)$.

计数 DP 的本质与此相同.

Kidnapper's Matching Problem

考虑如何判断 $x \oplus y \in 2^S_\oplus$. 先求出 S 的线性基 B.

断言. 假设 x, y 消去 B 中的位后得到的数分别为 x', y', 那么 $x \oplus y \in 2^S_\oplus$ 的充要条件是 $x' = y'$.

充分性: $x \oplus y$ 必然能写成 $x' \oplus y'$ 再异或上 B 中的一些数的形式. 在 $x' = y'$ 时即 $x \oplus y \in 2^S_\oplus$.

必要性: 考虑反证. 因为 x', y' 都不包含 B 中的位, 所以 $x' \oplus y'$ 不包含 B 中的位. 又因为 $x' \neq y'$, 所以 $x' \oplus y'$ 非零, 那么 $x' \oplus y'$ 一定包含 B 无法表示的位, 所以 $x \oplus y \notin 2^S_\oplus$.

接下来问题就变简单了. 我们只需要把序列 a, b 都消去 B 中的位, 然后做 KMP 即可. 时间复杂度 $O((N+M+K)\log V)$.

Linuber File System

考虑 DP. 设 $F_u(x)$ 表示 u 的所有真祖先执行子树加操作对 u 的贡献和为 x 时, u 的子树内部至少要几次子树加操作. 我们把 $F_u(x)$ 看作关于 x 的函数.

我们设在实际方案中 u 和它的所有祖先的贡献和为 y. 那么对于 u 所有孩子, 总的代价是 $\sum_{v\ u} F_v(y)$. 现在把 y 看作变量, 总代价的函数就是 $G_u(y) = \sum_{v\ u} F_v(y)$.

然而, y 会比 x 多出结点 u 自己产生的贡献（如果 u 自己执行子树加操作）, 也就是说如果 $x \neq y$ 就会产生 1 的额外代价. 那么, 由于要最小化代价, 从 G_u 到 F_u 的转移方程就可以写成:

$$F_u(x) = [x \neq y] + \min_{l_u \le y \le r_u} G_u(y)$$

有了转移方程, 我们就可以归纳地证明: $F_u(x)$ 是一个分段函数. 它的取值只有两种——在若干段区间中它的取值为 w_u, 在其余部分中它的取值为 $w_u + 1$. 这样 G_u 也可以表示成若干段常值函数. 在用 G_u 更新 F_u 时, $F_u(x)$ 的最小值就是 $G_u(y)$ 的最小值, 取到最小值的若干段区间就是 $G_u(y)$ 中取到最小值的那些区间, 其余部分的取值就是最小值 +1.

这样, 我们对于每个 u 维护 $F_u(x)$ 的分段函数即可. 在合并儿子求 G_u 时需要对函数的分段点排序, 所以总时间复杂度为 $O(Tn^2 \log n)$.