2021 江苏省大学生程序设计大赛 题解

2021 Jiangsu Collegiate Programming Contest Tutorial

电子科技大学

UESTC

2021 年 12 月 25 日

Problem A - Spring Couplets

题目大意

- 给一副对联，判断是否满足平仄规则。
- 关键词：模拟

Problem A - Spring Couplets

题目大意

- 给一副对联，判断是否满足平仄规则。
- 关键词：模拟

- 样例：

Problem A - Spring Couplets

题目大意

- 给一副对联，判断是否满足平仄规则。
- 关键词：模拟

- 样例：
  - 千门万户曈曈日，总把新桃换旧符。
  - 平平仄仄平平仄，仄仄平平仄仄平。
  - 苟
  - 哼哼哼，啊啊啊。

Problem A - Spring Couplets

题目大意

- 给一副对联，判断是否满足平仄规则。
- 关键词：模拟

- 样例：
  - 千门万户曈曈日，总把新桃换旧符。
  - 平平仄仄平平仄，仄仄平平仄仄平。
  - 苟
  - 哼哼哼，啊啊啊。
- 按照题目模拟判断即可。时间复杂度 $\mathcal{O}(n)$。

Problem A - Spring Couplets

题目大意

- 给一副对联，判断是否满足平仄规则。
- 关键词：模拟

- 样例：
  - 千门万户曈曈日，总把新桃换旧符。
  - 平平仄仄平平仄，仄仄平平仄仄平。
  - 苟
  - 哼哼哼，啊啊啊。
- 按照题目模拟判断即可。时间复杂度 $\mathcal{O}(n)$。
- 提前祝大家新春快乐！

Problem B - Among Us

题目大意

- 给出一张带边权的无向图，边权可被视为长度。最开始有 2 个内鬼和至多 8 个船员，内鬼起始位置给出，内鬼只能沿着边走。内鬼知道一系列形如“船员 p 会在第 t 秒出现在顶点 x”的信息。若内鬼与船员在同一时间和同一地点出现，则内鬼可以把船员给砍了。内鬼只能在前面给出的时间/地点砍人。你需要为 2 个内鬼分别规划路径使得它们能够尽快将所有船员都给砍了。若方案存在则输出最短时间，否则输出 -1。
- 关键词：图论，最短路，状压 DP

Problem B - Among Us

- 首先把两个内鬼分开，问题转化为给出某个内鬼的起始点，对每个船员集合 S 求出将 S 中所有人给砍了最短需要多少时间，然后 2^k 扫一次 S 即可合并出答案。

Problem B - Among Us

- 首先把两个内鬼分开，问题转化为给出某个内鬼的起始点，对每个船员集合 S 求出将 S 中所有人给砍了最短需要多少时间，然后 2^k 扫一次 S 即可合并出答案。
- 针对该问题不难直接设计出一个 $\mathcal{O}(2^k nt)$ 的 DP，即设状态 $dp[S][v][t]$ 为是否有可能在第 t 秒的时候将 S 中所有人都给砍了并站在顶点 v。

Problem B - Among Us

- 首先把两个内鬼分开，问题转化为给出某个内鬼的起始点，对每个船员集合 S 求出将 S 中所有人给砍了最短需要多少时间，然后 2^k 扫一次 S 即可合并出答案。
- 针对该问题不难直接设计出一个 $\mathcal{O}(2^k nt)$ 的 DP，即设状态 $dp[S][v][t]$ 为是否有可能在第 t 秒的时候将 S 中所有人都给砍了并站在顶点 v。
- 然后我们发现可以把时间这一维压到状态里，即设 $dp[S][v]$ 为将 S 中所有人都给砍了之后到达顶点 v 最少需要多少时间。

Problem B - Among Us

- 转移有两种：走路和砍人。

Problem B - Among Us

- 转移有两种：走路和砍人。
- 走路即固定 S 对所有 v 进行转移，可对每个 S 分别跑一遍最短路完成转移。

Problem B - Among Us

- 转移有两种：走路和砍人。
- 走路即固定 S 对所有 v 进行转移，可对每个 S 分别跑一遍最短路完成转移。
- 砍人即固定 v，对每个 S 查找每个不在 S 中的船员下一次出现在 v 是什么时候，然后在顶点 v 蹲到该船员出现为止。

Problem B - Among Us

- 转移有两种：走路和砍人。
- 走路即固定 S 对所有 v 进行转移，可对每个 S 分别跑一遍最短路完成转移。
- 砍人即固定 v，对每个 S 查找每个不在 S 中的船员下一次出现在 v 是什么时候，然后在顶点 v 蹲到该船员出现为止。
- 最后复杂度大概是 $\mathcal{O}(2^k(m \log n + nk \log t))$。

Problem C - Magical Rearrangement

题目大意

- 使用给定的数字集合构造一个尽可能小的相邻数字不同的数，不允许有前导零（单独一个 0 不算前导零）。
- 数据范围：$1 \leq n = \sum_{i=0}^{9} a_i \leq 10^5$
- 关键词：贪心，分类讨论

Problem C - Magical Rearrangement

题目大意

- 使用给定的数字集合构造一个尽可能小的相邻数字不同的数，不允许有前导零（单独一个 0 不算前导零）。
- 数据范围：$1 \le n = \sum_{i=0}^{9} a_i \le 10^5$
- 关键词：贪心，分类讨论

- 从高位开始贪。每次选择一个使得接下来有解的最小数字加到后面。

Problem C - Magical Rearrangement

题目大意

- 使用给定的数字集合构造一个尽可能小的相邻数字不同的数，不允许有前导零（单独一个 0 不算前导零）。
- 数据范围：$1 \le n = \sum_{i=0}^{9} a_i \le 10^5$
- 关键词：贪心，分类讨论

- 从高位开始贪。每次选择一个使得接下来有解的最小数字加到后面。
- 无解当且仅当余下的数字个数为 $2k+1$ 且出现次数最多的次数出现了 $k+1$ 次且你上一次才选了这个数。

Problem C - Magical Rearrangement

题目大意

- 使用给定的数字集合构造一个尽可能小的相邻数字不同的数，不允许有前导零（单独一个 0 不算前导零）。
- 数据范围：$1 \le n = \sum_{i=0}^{9} a_i \le 10^5$
- 关键词：贪心，分类讨论

- 从高位开始贪。每次选择一个使得接下来有解的最小数字加到后面。
- 无解当且仅当余下的数字个数为 $2k+1$ 且出现次数最多的次数出现了 $k+1$ 次且你上一次才选了这个数。
- 特判掉一些简单情况即可。

Problem D - Pattern Lock

题目大意

- n 行 m 列的点阵图，用一条折线经过所有的点各一次，要求每条线段不经过除端点外的其他点，且形成的角都是锐角。
- 数据范围：$2 \leq n, m \leq 500$
- 关键词：构造

Problem D - Pattern Lock

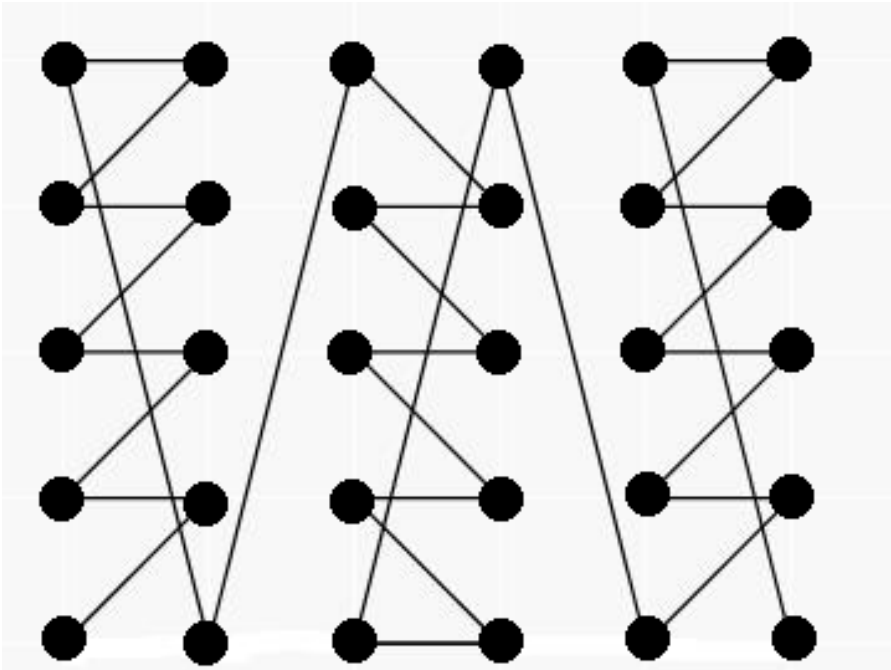
- 解法不唯一，这里给出其中一种思路，欢迎分享其他方法。

Problem D - Pattern Lock

- 解法不唯一，这里给出其中一种思路，欢迎分享其他方法。
- 当行数或列数至少一个为偶数时，可以每两行/列为一组如下构造：

Problem D - Pattern Lock

- 解法不唯一，这里给出其中一种思路，欢迎分享其他方法。
- 当行数或列数至少一个为偶数时，可以每两行/列为一组如下构造：

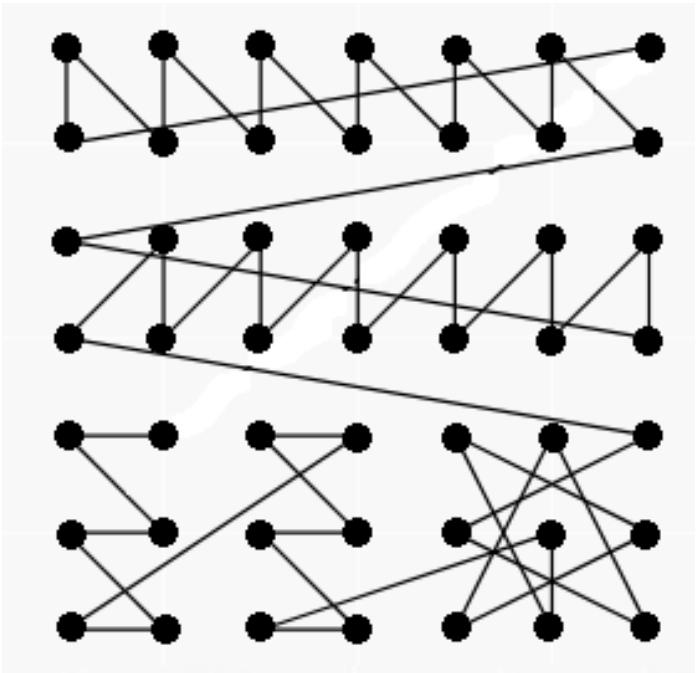


Problem D - Pattern Lock

- 当行数和列数都为奇数时，可以前 $n-3$ 行用上面的方法进行构造，然后剩下的 $3 \times m$ 又可以分成 $3 \times (m-3)$ 和 3×3 两部分，如下构造：

Problem D - Pattern Lock

- 当行数和列数都为奇数时，可以前 $n-3$ 行用上面的方法进行构造，然后剩下的 $3 \times m$ 又可以分成 $3 \times (m-3)$ 和 3×3 两部分，如下构造：



Problem E - Stone Ocean

题目大意

- 定义一个长为 n 串的权重 $w(s)$ 为有多少个排列 p 使得 $s[p_1]s[p_2]...s[p_n]$ 是一个回文，现在有 n 个串，你需要从每个串随机选一个位置的字符，连接成一个长为 n 的新串，求这个新串的 w 的期望，用分数取模 998244353 的方式表示。
- 数据范围：$n \leq 30$, 串长 $\leq 5 \times 10^4$
- 关键词：期望，状压 DP

Problem E - Stone Ocean

- 计算期望，我们选择统计权重 w 的总和，最后通过除以总方案数量得到期望，总方案数量为所有串的串长的乘积这个不需要解释，重点在于如何计算 w 的总和。

Problem E - Stone Ocean

- 计算期望，我们选择统计权重 w 的总和，最后通过除以总方案数量得到期望，总方案数量为所有串的串长的乘积这个不需要解释，重点在于如何计算 w 的总和。
- 若 n 为偶数，我们可以直接枚举数对 (u, v) 在排列后有 $p_u = n - p_v + 1$ 对总 w 的贡献，这里规定 $u < v$，其中 (u, v) 要满足能对 w 产生贡献，从原串 S_u 和 S_v 中选取的字符要相同，仅有这一个条件，这个方案数很好统计，我们计为 $val(u, v)$。

Problem E - Stone Ocean

- 计算期望，我们选择统计权重 w 的总和，最后通过除以总方案数量得到期望，总方案数量为所有串的串长的乘积这个不需要解释，重点在于如何计算 w 的总和。
- 若 n 为偶数，我们可以直接枚举数对 (u, v) 在排列后有 $p_u = n - p_v + 1$ 对总 w 的贡献，这里规定 $u < v$，其中 (u, v) 要满足能对 w 产生贡献，从原串 S_u 和 S_v 中选取的字符要相同，仅有这一个条件，这个方案数很好统计，我们计为 $val(u, v)$。
- 我们会枚举出 $\frac{n}{2}$ 个数对，其中所有数字各不相同，这些数对中满足条件的选取字符的方案数为 $\prod_{i=1}^{\frac{n}{2}} val(u_i, v_i)$，对 w 的贡献再考虑数对在新串中摆放的位置即可，这是一个排列，贡献为 $2^{\frac{n}{2}} \cdot (\frac{n}{2})! \cdot \prod_{i=1}^{\frac{n}{2}} val(u_i, v_i)$。则 w 即为所有数对选择方案的贡献和。

Problem E - Stone Ocean

- 容易联想到状压来进行转移，对于一个状态 $status$，以二进制位上的 1 为已匹配过且与其他数字形成了数对，0 为未匹配。每次枚举两个未匹配过的数字进行匹配，为了保证统计方案不重复，我们可以选取 0 位中的最小数字与其他数字匹配。转移如下

Problem E - Stone Ocean

- 容易联想到状压来进行转移，对于一个状态 $status$，以二进制位上的 1 为已匹配过且与其他数字形成了数对，0 为未匹配。每次枚举两个未匹配过的数字进行匹配，为了保证统计方案不重复，我们可以选取 0 位中的最小数字与其他数字匹配。转移如下

DP

1. `next_status = status | (1 << u) | (1 << v);`
2. `dp[next_status] += dp[status] * val(u, v);`

Problem E - Stone Ocean

- 但是可能 n 的范围能达到 30，所以可能有的队伍会放弃状压的想法，可是如果来计算一下状态数其实远远没有达到 2^{30} 那么恐怖。

Problem E - Stone Ocean

- 但是可能 n 的范围能达到 30，所以可能有的队伍会放弃状压的想法，可是如果来计算一下状态数其实远远没有达到 2^{30} 那么恐怖。
- 我们设 $S(n)$ 为长度为 n 的总状态数，有

Problem E - Stone Ocean

- 但是可能 n 的范围能达到 30，所以可能有的队伍会放弃状压的想法，可是如果来计算一下状态数其实远远没有达到 2^{30} 那么恐怖。
- 我们设 $S(n)$ 为长度为 n 的总状态数，有

$$S(n) = \sum_{k=0}^{n} \binom{n-k}{k}$$

$$S(n-1) + S(n-2) + 1$$

$$= \sum_{k=0}^{n-1} \binom{n-k-1}{k} + \sum_{k=0}^{n-2} \binom{n-k-2}{k} + 1$$

$$= \binom{n-1}{0} + \sum_{k=1}^{n-1} \binom{n-k-1}{k} + \binom{n-k-1}{k-1} + 1$$

$$= \binom{n-0}{0} + \sum_{k}^{n-k} \binom{n-k}{k} + \binom{n-n}{0} = S(n)$$

Problem E - Stone Ocean

- 状态数为一个斐波那契数第 n 项的级别，可看作 1.618^n，考虑转移时候枚举匹配，则总复杂度为 $\mathcal{O}(n \cdot 1.618^n)$，暴力状压即可通过此题，注意需要跳过没有出现过的状态，实现的时候可以类似于在对于一个 DAG 图进行 BFS，使用队列保存出现过的状态进行转移。

Problem E - Stone Ocean

- 状态数为一个斐波那契数第 n 项的级别，可看作 1.618^n，考虑转移时候枚举匹配，则总复杂度为 $\mathcal{O}(n \cdot 1.618^n)$，暴力状压即可通过此题，注意需要跳过没有出现过的状态，实现的时候可以类似于在对于一个 DAG 图进行 BFS，使用队列保存出现过的状态进行转移。
- 考虑 n 为奇数怎么做，我们可以添加一个虚点，与虚点匹配上的数字固定作为中点即可，其余转移不变，复杂度不变。

Problem F - Jumping Monkey II

题目大意

- 给一棵树，带有点权。对每个点，以他为起点出发时，求所有简单路径形成的序列中最长上升子序列的长度最长是多少。
- 数据范围：$1 \leq n \leq 2 \times 10^5$
- 关键词：点分治，DP

Problem F - Jumping Monkey II

- 树上 LIS 问题的一个经典做法是用启发式合并
或线段树合并 来维护 DP 数组从子结点向父节点的转移。
但这样我们只能统计每个结点向其子树方向的 LIS。

Problem F - Jumping Monkey II

- 树上 LIS 问题的一个经典做法是用启发式合并或线段树合并 来维护 DP 数组从子结点向父节点的转移。但这样我们只能统计每个结点向其子树方向的 LIS。
- 考虑点分治，我们可以将从 u 出发的路径按其经过分治中心分为 $\log n$ 组。对每个分治中心，维护其每棵子树的最长下降子序列的 DP 数组，将这些 DP 数组合并到分治中心，再贡献给每棵子树的每个结点（注意去掉当前子树和加上分治中心）。

Problem F - Jumping Monkey II

- 树上 LIS 问题的一个经典做法是用启发式合并或线段树合并 来维护 DP 数组从子结点向父节点的转移。但这样我们只能统计每个结点向其子树方向的 LIS。
- 考虑点分治，我们可以将从 u 出发的路径按其经过分治中心分为 $\log n$ 组。对每个分治中心，维护其每棵子树的最长下降子序列的 DP 数组，将这些 DP 数组合并到分治中心，再贡献给每棵子树的每个结点（注意去掉当前子树和加上分治中心）。
- 时间复杂度 $\mathcal{O}(n \log^2 n)$。

Problem F - Jumping Monkey II

- 树上 LIS 问题的一个经典做法是用启发式合并或线段树合并 来维护 DP 数组从子结点向父节点的转移。但这样我们只能统计每个结点向其子树方向的 LIS。
- 考虑点分治，我们可以将从 u 出发的路径按其经过分治中心分为 $\log n$ 组。对每个分治中心，维护其每棵子树的最长下降子序列的 DP 数组，将这些 DP 数组合并到分治中心，再贡献给每棵子树的每个结点（注意去掉当前子树和加上分治中心）。
- 时间复杂度 $\mathcal{O}(n \log^2 n)$。
- P.S. 线段树合并的常数太大，在卡一种错解的时候大概是把线段树合并也卡掉了。

Problem G - Five Phases

题目大意

- 在五维空间中从 $(0,0,0,0,0)$ 出发使用给定的方向集合走 k 步，有多少种走法可以走到 (a,b,c,d,e)。
- 数据范围：$1 \leq k \leq 10^5$
- 关键词：生成函数

Problem G - Five Phases

- 生成函数。由题意得答案为

Problem G - Five Phases

- 生成函数。由题意得答案为

$$\left[v^a w^b x^c y^d z^e\right] \left(\begin{aligned} & v+w+x+y+z+\frac{1}{v}+\frac{1}{w}+\frac{1}{x}+\frac{1}{y}+\frac{1}{z} \\ & +vwx+wxy+xyz+yzv+zvw \\ & +\frac{1}{vwx}+\frac{1}{wxy}+\frac{1}{xyz}+\frac{1}{yzv}+\frac{1}{zvw} \\ & +\frac{vw}{y}+\frac{wx}{z}+\frac{xy}{v}+\frac{yz}{w}+\frac{zv}{x} \\ & +\frac{y}{vw}+\frac{z}{wx}+\frac{v}{xy}+\frac{w}{yz}+\frac{x}{zv} \\ & +vwxyz+\frac{1}{vwxyz} \end{aligned} \right)^k$$

Problem G - Five Phases

- 生成函数。由题意得答案为

$$\left[v^a w^b x^c y^d z^e\right] \left(\begin{array}{r} v+w+x+y+z+\frac{1}{v}+\frac{1}{w}+\frac{1}{x}+\frac{1}{y}+\frac{1}{z} \\ +vwx+wxy+xyz+yzv+zvw \\ +\frac{1}{vwx}+\frac{1}{wxy}+\frac{1}{xyz}+\frac{1}{yzv}+\frac{1}{zvw} \\ +\frac{vw}{y}+\frac{wx}{z}+\frac{xy}{v}+\frac{yz}{w}+\frac{zv}{x} \\ +\frac{y}{vw}+\frac{z}{wx}+\frac{v}{xy}+\frac{w}{yz}+\frac{x}{zv} \\ +vwxyz+\frac{1}{vwxyz} \end{array} \right)^k$$

$$= \left[v^a w^b x^c y^d z^e\right] \left[\frac{(1+vw)(1+wx)(1+xy)(1+yz)(1+zv)}{vwxyz}\right]^k$$

Problem G - Five Phases

- 设 vw, wx, xy, yz, zv 的幂次分别为 m, n, o, p, q，得

Problem G - Five Phases

- 设 vw, wx, xy, yz, zv 的幂次分别为 m, n, o, p, q，得

$$
\begin{aligned}
a &= q + m - k \\
b &= m + n - k \\
c &= n + o - k \\
d &= o + p - k \\
e &= p + q - k
\end{aligned}
$$

Problem G - Five Phases

- 解得

Problem G - Five Phases

- 解得

$$m = \frac{b - c + d - e + a + k}{2}$$
$$n = \frac{c - d + e - a + b + k}{2}$$
$$o = \frac{d - e + a - b + c + k}{2}$$
$$p = \frac{e - a + b - c + d + k}{2}$$
$$q = \frac{a - b + c - d + e + k}{2}$$

Problem G - Five Phases

- 解得

$$m = \frac{b - c + d - e + a + k}{2}$$
$$n = \frac{c - d + e - a + b + k}{2}$$
$$o = \frac{d - e + a - b + c + k}{2}$$
$$p = \frac{e - a + b - c + d + k}{2}$$
$$q = \frac{a - b + c - d + e + k}{2}$$

- 于是答案是

Problem G - Five Phases

- 解得

$$m = \frac{b - c + d - e + a + k}{2}$$
$$n = \frac{c - d + e - a + b + k}{2}$$
$$o = \frac{d - e + a - b + c + k}{2}$$
$$p = \frac{e - a + b - c + d + k}{2}$$
$$q = \frac{a - b + c - d + e + k}{2}$$

- 于是答案是

$$\binom{k}{m}\binom{k}{n}\binom{k}{o}\binom{k}{p}\binom{k}{q}$$

Problem H - Reverse the String

题目大意

- 给你一个由小写字母组成的字符串 s，让你对至多一个子串进行区间翻转，求字典序最小的结果。
- 数据范围：$|s| \leq 100000$
- 关键词：hash，SA

Problem H - Reverse the String

- 首先考虑在什么情况下一个字符串不进行翻转操作是字典序最小的，显然，让我们给这个字符串当作字符数组后排序，如果没有改变顺序的话，就说明当前字符串已经是字典序最小的了。

Problem H - Reverse the String

- 首先考虑在什么情况下一个字符串不进行翻转操作是字典序最小的，显然，让我们给这个字符串当作字符数组后排序，如果没有改变顺序的话，就说明当前字符串已经是字典序最小的了。
- 然后考虑如果要翻转，从哪里开始翻转。从上面的结论可以看出，翻转和排序之后的字符数组匹配的前缀是没有意义的，因此翻转开头的位置必然是原字符串第一个和排序后的字符数组不匹配的位置。

Problem H - Reverse the String

- 首先考虑在什么情况下一个字符串不进行翻转操作是字典序最小的，显然，让我们给这个字符串当作字符数组后排序，如果没有改变顺序的话，就说明当前字符串已经是字典序最小的了。
- 然后考虑如果要翻转，从哪里开始翻转。从上面的结论可以看出，翻转和排序之后的字符数组匹配的前缀是没有意义的，因此翻转开头的位置必然是原字符串第一个和排序后的字符数组不匹配的位置。
- 然后可以直接枚举翻转结束的位置，求字符串字典序最小最简单的方法当然是直接通过二分答案 +hash 来求出两个字符串的 lcp 后比较下一个字符即可。

Problem H - Reverse the String

- 原因如下：

Problem H - Reverse the String

- 原因如下：
- 因为对一个字符串的字串翻转后的字符串相当于原字符串的一个前缀连上一个选定翻转的字符串的逆序连上原字符串的后缀，

Problem H - Reverse the String

- 原因如下：
- 因为对一个字符串的字串翻转后的字符串相当于原字符串的一个前缀连上一个选定翻转的字符串的逆序连上原字符串的后缀，
- 我们只需要预处理原字符串的正序和逆序的 hash 值即可 $\mathcal{O}(1)$ 来得到新字符串的一个子串的 hash 值。

Problem H - Reverse the String

- 原因如下：
- 因为对一个字符串的字串翻转后的字符串相当于原字符串的一个前缀连上一个选定翻转的字符串的逆序连上原字符串的后缀，
- 我们只需要预处理原字符串的正序和逆序的 hash 值即可 $\mathcal{O}(1)$ 来得到新字符串的一个子串的 hash 值。
- 二分复杂度为 $\mathcal{O}(\log n)$，翻转的字符串开头固定结尾个数为 $\mathcal{O}(n)$，因此本题的总复杂度为 $\mathcal{O}(Tn \log n)$，此外使用 SA 或者许多其他算法也可以完成该题。

Problem I - Fake Walsh Transform

题目大意

- 用 0 到 2^m-1 范围内（包括两端）尽可能多的数的异或和表示 n，求最多可以用多少个数来表示。
- 数据范围：$1 \le m \le 60, 0 \le n < 2^m$。
- 关键词：分类讨论

Problem I - Fake Walsh Transform

- 注意到当 $m \geq 2$ 时，0 到 $2^m - 1$ 中所有数的异或和为 0，这就说明了此时每个二进制位上出现的 1 的次数为偶数，因此我们想用最多的 0 到 $2^m - 1$ 之内的数的异或和得到 n，我们只需不异或 n 即可。但当 $n = 0$ 时，我们仍可异或 n 并保持结果不变。

Problem I - Fake Walsh Transform

- 注意到当 $m \geq 2$ 时，0 到 $2^m - 1$ 中所有数的异或和为 0，这就说明了此时每个二进制位上出现的 1 的次数为偶数，因此我们想用最多的 0 到 $2^m - 1$ 之内的数的异或和得到 n，我们只需不异或 n 即可。但当 $n = 0$ 时，我们仍可异或 n 并保持结果不变。
- 当 $m = 1$ 时，由于只能取 0 和 1，当 $n = 0$ 时只能取 0 这一个数，而当 $n = 1$ 时我们可以取 $\{0, 1\}$ 两个。

Problem I - Fake Walsh Transform

- 注意到当 $m \geq 2$ 时，0 到 $2^m - 1$ 中所有数的异或和为 0，这就说明了此时每个二进制位上出现的 1 的次数为偶数，因此我们想用最多的 0 到 $2^m - 1$ 之内的数的异或和得到 n，我们只需不异或 n 即可。但当 $n = 0$ 时，我们仍可异或 n 并保持结果不变。
- 当 $m = 1$ 时，由于只能取 0 和 1，当 $n = 0$ 时只能取 0 这一个数，而当 $n = 1$ 时我们可以取 $\{0, 1\}$ 两个。
- 综上，答案为：

$$\begin{cases} 1 & n = 0, m = 1 \\ 2 & n = 1, m = 1 \\ 2^m - 1 & n \neq 0, m \geq 2 \\ 2^m & n = 0, m \geq 2 \end{cases}$$

Problem I - Fake Walsh Transform

- 注意到当 $m \geq 2$ 时，0 到 $2^m - 1$ 中所有数的异或和为 0，这就说明了此时每个二进制位上出现的 1 的次数为偶数，因此我们想用最多的 0 到 $2^m - 1$ 之内的数的异或和得到 n，我们只需不异或 n 即可。但当 $n = 0$ 时，我们仍可异或 n 并保持结果不变。
- 当 $m = 1$ 时，由于只能取 0 和 1，当 $n = 0$ 时只能取 0 这一个数，而当 $n = 1$ 时我们可以取 $\{0, 1\}$ 两个。
- 综上，答案为：

$$\begin{cases} 1 & n = 0, m = 1 \\ 2 & n = 1, m = 1 \\ 2^m - 1 & n \neq 0, m \geq 2 \\ 2^m & n = 0, m \geq 2 \end{cases}$$

- 时间复杂度：$\mathcal{O}(1)$，空间复杂度：$\mathcal{O}(1)$。

Problem J - Anti-merge

题目大意

- 对于一个数表，对于相同的单元格进行合并，优先合并相同内容的行，再合并高度相同且行内容相同的列。现在想让所有的单元格都不会合并，方法是添加一些冗余内容。问需要至少要给多少单元格添加冗余内容，至少添加多少种冗余内容。
- 数据范围：$1 \leq n, m \leq 500$。
- 关键词：BFS，黑白染色

Problem J - Anti-merge

- 注意到最终的局面应该是：对于任意单元格内容，将不添加标签看成 0 号标签。对于一个单元格，由单元格内容和单元格标签组成的 pair 对应该与它四连通的所有单元格互不相同。

Problem J - Anti-merge

- 注意到最终的局面应该是：对于任意单元格内容，将不添加标签看成 0 号标签。对于一个单元格，由单元格内容和单元格标签组成的 pair 对应该与它四连通的所有单元格互不相同。
- 考虑黑白棋盘染色，可知最少所用颜色数不超过 1，对于方案的求解，可以考虑使用 BFS，从每个位置开始对 pair 相同的单元格区域进行这种黑白染色，取黑白颜色数较少的颜色填充标签即可。

Problem J - Anti-merge

- 注意到最终的局面应该是：对于任意单元格内容，将不添加标签看成 0 号标签。对于一个单元格，由单元格内容和单元格标签组成的 pair 对应该与它四连通的所有单元格互不相同。
- 考虑黑白棋盘染色，可知最少所用颜色数不超过 1，对于方案的求解，可以考虑使用 BFS，从每个位置开始对 pair 相同的单元格区域进行这种黑白染色，取黑白颜色数较少的颜色填充标签即可。
- 时间复杂度：$\mathcal{O}(nm)$，空间复杂度：$\mathcal{O}(nm)$。

Problem J - Anti-merge

- 注意到最终的局面应该是：对于任意单元格内容，将不添加标签看成 0 号标签。对于一个单元格，由单元格内容和单元格标签组成的 pair 对应该与它四连通的所有单元格互不相同。
- 考虑黑白棋盘染色，可知最少所用颜色数不超过 1，对于方案的求解，可以考虑使用 BFS，从每个位置开始对 pair 相同的单元格区域进行这种黑白染色，取黑白颜色数较少的颜色填充标签即可。
- 时间复杂度：$\mathcal{O}(nm)$，空间复杂度：$\mathcal{O}(nm)$。
- Special thanks to Menci for markdown-it-cells-merge, and contributions to SYZOJ and Lyrio Platform.

Problem K - Longest Continuous 1

题目大意

- 将 $0, 1, 2, \ldots$ 不含前导 0 的二进制形式依次拼接成一个字符串，求长度为 k 的前缀中最长连续 1 的长度。
- 数据范围：$1 \leq k \leq 10^9$。
- 关键词：二分，打表

Problem K - Longest Continuous 1

- 结论是：前缀中最长连续 1 的长度是满足 $(n-1)\cdot 2^n+2\geq k$ 的最小的 n。

Problem K - Longest Continuous 1

- 结论是：前缀中最长连续 1 的长度是满足 $(n-1)\cdot 2^n+2\geq k$ 的最小的 n。
- 下面我们来说明这一结论。对于单独的一个数字，其二进制中连续的 1 最长的情况为这个数字为某个 2 的幂次减 1。我们可以采取构造法证明这一结论。

Problem K - Longest Continuous 1

- 结论是：前缀中最长连续 1 的长度是满足 $(n-1)\cdot 2^n+2\geq k$ 的最小的 n。
- 下面我们来说明这一结论。对于单独的一个数字，其二进制中连续的 1 最长的情况为这个数字为某个 2 的幂次减 1。我们可以采取构造法证明这一结论。
- 对于将这些字符串拼接起来，我们需要考虑两点：

Problem K - Longest Continuous 1

- 结论是：前缀中最长连续 1 的长度是满足 $(n-1)\cdot 2^n+2\geq k$ 的最小的 n。
- 下面我们来说明这一结论。对于单独的一个数字，其二进制中连续的 1 最长的情况为这个数字为某个 2 的幂次减 1。我们可以采取构造法证明这一结论。
- 对于将这些字符串拼接起来，我们需要考虑两点：
  - 对于 2^m-1 类型的字符串，拼接后会不会连续的 1 变长；

Problem K - Longest Continuous 1

- 结论是：前缀中最长连续 1 的长度是满足 $(n-1)\cdot 2^n+2\geq k$ 的最小的 n。
- 下面我们来说明这一结论。对于单独的一个数字，其二进制中连续的 1 最长的情况为这个数字为某个 2 的幂次减 1。我们可以采取构造法证明这一结论。
- 对于将这些字符串拼接起来，我们需要考虑两点：
  - 对于 2^m-1 类型的字符串，拼接后会不会连续的 1 变长；
  - 对于不是这种类型的字符串，会不会存在拼接后连续的 1 比 2^m-1 这种类型字符串拼接后还要长。

Problem K - Longest Continuous 1

- 对于第一个问题，我们考虑 $2^m - 2$ 为偶数，而 2^m 的二进制表示中第一个位置是 1。那么按顺序拼接到 2^m 后由 $2^m - 2, 2^m - 1, 2^m$ 组成的字符串中连续最长 1 的长度即为 $m + 1$。

Problem K - Longest Continuous 1

- 对于第一个问题，我们考虑 $2^m - 2$ 为偶数，而 2^m 的二进制表示中第一个位置是 1。那么按顺序拼接到 2^m 后由 $2^m - 2, 2^m - 1, 2^m$ 组成的字符串中连续最长 1 的长度即为 $m + 1$。
- 对于第二个问题，我们可以证明不存在这样的情况。如果出现这样的情况，说明一个长度小于等于 $m - 2$ 且均为 1 的后缀（因为如果这个后缀长度为 $m - 1$，那么它就会有前导零，不符合题意）和一个长度大于 3 的 1 前缀拼接在一起，而当最差情况下，若前一个数二进制的后缀连续 1 长度为 $m - k\ (k \geq 2)$，则下一个数二进制的前缀连续 1 最大将为 k，加在一起长度为 m，并不会更新第一个问题中讨论出的答案 $m + 1$。

Problem K - Longest Continuous 1

- 最差情况出现在：

$$\underbrace{11\ldots111}_{k-1}0\underbrace{111\ldots111}_{m-k}$$

Problem K - Longest Continuous 1

- 最差情况出现在：

$$\underbrace{11\ldots111}_{k-1}0\underbrace{111\ldots111}_{m-k}$$

- 那么下一个数将为：

$$\underbrace{11\ldots1111}_{k}\underbrace{00\ldots000}_{m-k}$$

Problem K - Longest Continuous 1

- 最差情况出现在：

$$\underbrace{11\ldots111}_{k-1}0\underbrace{111\ldots111}_{m-k}$$

- 那么下一个数将为：

$$\underbrace{11\ldots1111}_{k}\underbrace{00\ldots000}_{m-k}$$

- 这样最长连续的 1 仍然长度为 m。

Problem K - Longest Continuous 1

- 因此，每当达到 2^m 的二进制表示中的 1 时，前缀中最长的 1 的答案就会更新为 $m+1$。这些位置是将 0 到 2^m-1 所有数的二进制表示都写出来的长度加 1。下面的问题就转化成 0 到 2^m-1 所有数的二进制依次写出的长度。

Problem K - Longest Continuous 1

- 因此，每当达到 2^m 的二进制表示中的 1 时，前缀中最长的 1 的答案就会更新为 $m+1$。这些位置是将 0 到 2^m-1 所有数的二进制表示都写出来的长度加 1。下面的问题就转化成 0 到 2^m-1 所有数的二进制依次写出的长度。
- 考虑 $i \geq 2$，若某个数的二进制长度为 i，则它的范围将会是 $[2^{i-1}, 2^i-1]$，但是 $i=1$ 时范围为 $[0,1]$。记 L_m 为 0 到 2^m-1 所有数的二进制依次写出所组成的字符串长度，则有

$$L_m = \begin{cases} 2 & m = 1 \\ L_{m-1} + m \cdot (2^m - 1 - 2^{m-1} + 1) & m \geq 2 \end{cases}$$

Problem K - Longest Continuous 1

- 因此，每当达到 2^m 的二进制表示中的 1 时，前缀中最长的 1 的答案就会更新为 $m+1$。这些位置是将 0 到 2^m-1 所有数的二进制表示都写出来的长度加 1。下面的问题就转化成 0 到 2^m-1 所有数的二进制依次写出的长度。
- 考虑 $i \geq 2$，若某个数的二进制长度为 i，则它的范围将会是 $[2^{i-1}, 2^i-1]$，但是 $i=1$ 时范围为 $[0,1]$。记 L_m 为 0 到 2^m-1 所有数的二进制依次写出所组成的字符串长度，则有

$$L_m = \begin{cases} 2 & m = 1 \\ L_{m-1} + m \cdot (2^m - 1 - 2^{m-1} + 1) & m \geq 2 \end{cases}$$

- 由于 $L_m - L_{m-1} = m \cdot 2^{m-1}$，因此使用累加法后裂项相消即可求出 $L_m = (m-1) \cdot 2^m + 2$。

Problem K - Longest Continuous 1

- 也就是说，当 $k \in (L_{m-1}, L_m]$ 时，答案都是 $m-1$。打出 $m \leq 30$ 的表之后，二分或直接查找怎么做都行。
- 时间复杂度：$\mathcal{O}(\log k)$ 或 $\mathcal{O}(\log \log k)$，空间复杂度：$\mathcal{O}(\log k)$。

Problem L - Tree Game

题目大意

- 给一棵树，每个点有权值，权值是一个排列，你要运行一个程序将这些权值排序，程序的 dfs 这棵树（1 是根），每次重新排列 u 和所有 $ch[u]$，然后向上递归。排序的结果是权值等于结点编号，现在你忘记了每个结点原来的权值，只记得其中一些，问有多少可能的权值排列可以让你排序成功。
- 数据范围：$2 \leq n \leq 2 \times 10^5$
- 关键词：DP

Problem L - Tree Game

- 由于整个过程是一个向上递归的过程，所以对于一个子树，我们必须要保证子树内数字排序成功，再递归向上考虑子树根结点与其兄弟结点和父节点的重新排列。根据这个定义我们容易得到一个结论，子树的根结点上的值在递归向上的过程中一定是固定的，只需要随着 dfs 去维护每个结点在以其为根结点的子树完成重新排列后这个结点上的数字是什么即可。

Problem L - Tree Game

- 由于整个过程是一个向上递归的过程，所以对于一个子树，我们必须要保证子树内数字排序成功，再递归向上考虑子树根结点与其兄弟结点和父节点的重新排列。根据这个定义我们容易得到一个结论，子树的根结点上的值在递归向上的过程中一定是固定的，只需要随着 dfs 去维护每个结点在以其为根结点的子树完成重新排列后这个结点上的数字是什么即可。
- 在这里我们是将所有未知的数看作同一个值 0 来考虑这个问题，这意味着我们可以只在同一层中考虑多个 0 应当被分配给哪个数字，设当前结点 u，所有以它的孩子结点为根结点的子树已经完成重新排列，$cnt0$ 为此刻它以及它的全部孩子结点上的 0 的数量，则转移后方案应当乘以 $(cnt0)!$。

Problem L - Tree Game

- 设 $dp[u]$ 为以 u 为根节点的子树能完成重新排列的方案数，转移式如下：

Problem L - Tree Game

- 设 $dp[u]$ 为以 u 为根节点的子树能完成重新排列的方案数，转移式如下：

$$dp[u] = (cnt0)! \cdot \prod_{v \text{ is a direct child of } u} dp[v]$$

Problem L - Tree Game

- 设 $dp[u]$ 为以 u 为根节点的子树能完成重新排列的方案数，转移式如下：

$$dp[u] = (cnt0)! \cdot \prod_{v \text{ is a direct child of } u} dp[v]$$

- 最后注意判断当前层是否能成功重新排列，如果存在一层无法成功重新排列，则总方案数一定为 0。可以使用一个 unordered_set 来查找当前层的重新排列中有没有无法互换的情况，并且维护根结点最后的定值是什么。

Problem L - Tree Game

- 详细来讲：

Problem L - Tree Game

- 详细来讲：
  - 我们用 unordered_set 来记录 u 的所有儿子编号，

Problem L - Tree Game

- 详细来讲：
  - 我们用 unordered_set 来记录 u 的所有儿子编号，
  - 初始我们可以看作 u 最后向上递归时的值是 0。

Problem L - Tree Game

- 详细来讲：
  - 我们用 unordered_set 来记录 u 的所有儿子编号，
  - 初始我们可以看作 u 最后向上递归时的值是 0。
  - 若 $a[u]$ 在这个 unordered_set 中无法找到，则 u 最后的值一定是 $a[u]$；

Problem L - Tree Game

- 详细来讲：
  - 我们用 unordered_set 来记录 u 的所有儿子编号，
  - 初始我们可以看作 u 最后向上递归时的值是 0。
  - 若 $a[u]$ 在这个 unordered_set 中无法找到，则 u 最后的值一定是 $a[u]$；
  - 若 u 的一个儿子 v 发现自己当前的结点上的值 $a[v]$ 在这个 unordered_set 中无法找到，结点 u 将被赋予的值为 $a[v]$，但如果结点 u 已被赋予了其他值则冲突，视为重新排列失败。

Problem L - Tree Game

- 详细来讲：
  - 我们用 unordered_set 来记录 u 的所有儿子编号，
  - 初始我们可以看作 u 最后向上递归时的值是 0。
  - 若 $a[u]$ 在这个 unordered_set 中无法找到，则 u 最后的值一定是 $a[u]$；
  - 若 u 的一个儿子 v 发现自己当前的结点上的值 $a[v]$ 在这个 unordered_set 中无法找到，结点 u 将被赋予的值为 $a[v]$，但如果结点 u 已被赋予了其他值则冲突，视为重新排列失败。
  - 否则视为成功重新排列，向上递归即可。

Problem L - Tree Game

- 详细来讲：
  - 我们用 unordered_set 来记录 u 的所有儿子编号，
  - 初始我们可以看作 u 最后向上递归时的值是 0。
  - 若 $a[u]$ 在这个 unordered_set 中无法找到，则 u 最后的值一定是 $a[u]$；
  - 若 u 的一个儿子 v 发现自己当前的结点上的值 $a[v]$ 在这个 unordered_set 中无法找到，结点 u 将被赋予的值为 $a[v]$，但如果结点 u 已被赋予了其他值则冲突，视为重新排列失败。
  - 否则视为成功重新排列，向上递归即可。
- 总复杂度 $\mathcal{O}(n)$。