Contest #9 命题报告与总结

南京外国语学校 戴　言 （ diamond_duke ）
南京外国语学校 杨骏昭 （ FizzyDavid ）
南京外国语学校 陈孙立 （ EEEEEEricKKK ）

2018 年 11 月 23 日

1 Roundtrip

Author: EEEEEEricKKK

1.1 算法一

因为每个城市最多经过一次，可以枚举所有可能的路径，复杂度 $\Theta(n!)$。

1.2 算法二

把暴力枚举改为状态压缩，复杂度 $\Theta(2^n)$ 。

1.3 算法三

注意到如果有解，那么一定存在一条从 a 到 b 的简单路径经过 c 。

如果图中 a 到 b 只有 3 条不同的简单路径，就可以通过 dfs 找到这 3 条路径。如果 c 在其中的某一条路径上，就可以沿着这条路径从 a 走到 b，再沿着另一条路径走回 a，否则答案就是 -1 。

1.4 算法四

对于一般的情况，a 到 b 的简单路径个数是指数级别的，不能通过 DFS 求出。

我们可以在原图的基础上建立一个新图。对原图中的每个点 u 建立两个点 u_{in} 和 u_{out} ，然后对于原图每条边 (u, v) ，连接两条新边 (u_{out}, v_{in}) 和 (v_{out}, u_{in}) 。每条边的流量都是 1。可以发现，原图中 k 条不相交的 a 到 b 的路径就对应着新图中从 a_{out} 到 b_{in} 的 k 个单位的流。这样，只要使用最大流算法就能求出 3 条不相交路径。如果 c 在其中某条路径上，按照算法三可以找到解，因此假设 c 不在任意一条路径上。

考虑一条可行的环路，其中一定包含了一条从 a 到 c 的路径和从 c 到 b 的路径，且这两条路径不相交。我们需要保证这两条路径前面求出的三条

a 到 b 的路径中的至多两条有交，这样就可以通过剩下的一条路径走回去。因此，我们需要尝试找到满足条件的这样两条 a 到 c 和 c 到 b 的路径。

我们可以证明，只要存在两条 a 到 c 和 c 到 b 的不相交路径，就一定存在满足条件的解。令在 a 到 b 的第 i 条路径上的点的集合是 S_i ，$S = S_1 \cup S_2 \cup S_3$。我们先找到两条从 a 到 c 和 c 到 b 的不相交路径，令 v_1 表示在 a 到 c 的路径上且在集合 S 中的点中，离 c 最近的一个；类似地，令 v_2 表示在 c 到 b 的路径上且在集合 S 中的点中离 c 最近的一个。由 $c \notin S$ 得到 $v_1 \neq c, v_2 \neq c$。当 $v_1 \neq a, v_2 \neq b$ 且 $v_1 \in S_x, v_2 \in S_y, x \neq y$ 时，我们从 a 沿着 a 到 b 的第 x 条路径走到 v_1，再沿着找到的路径走到 c 然后到 v_2，再沿着 a 到 b 的第 y 条路径到 b，最后沿着剩下的路径走回去即可得到答案。对于其他的情况，可以类比这样的讨论，得到本段开始的结论成立。

最后，就只需要求 a 到 c 和 c 到 b 的两条不相交路径了。我们可以建立新点 v ，并加边 (a, v) 和 (b, v) ，问题就变成找到两条从 c 到 v 的不相交路径，使用原来的最大流算法即可。如果网络流用 Dinic 实现，复杂度 $\Theta(n^2 m)$ 。

1.5 算法五

注意到，我们并不需要求出整个最大流，只需要增广 2 次或 3 次即可，复杂度就被降到了 $\Theta(n + m)$ 。

最后关于程序实现，我们并不需要讨论上面那么多种情况。如果有解，一定存在一个解在 a 到 c 到 b 的过程中只与两条 a 到 b 的路径有交。因此我们可以枚举剩下没有交的路径是哪一条，然后强制这条路径上的点不能经过，去寻找从 a 到 c 和 c 到 b 的路径。这样复杂度不变。

2 Golf

Author: diamond_duke

2.1 算法一

对于 $n = 0$ 的情况，因为没有障碍物，所以我们至多两次就可以把高尔夫球从起点打到终点。判断一下横、纵坐标是否相等即可知道一次是否可行。

时间复杂度：$\Theta(1)$，期望得分：5 分。

2.2 算法二

因为只能停在整点上，所以我们可以对每个整点建出一个点，然后进行 BFS 求最短路。

转移时枚举下一步击球的方向以及目的地，即可做到转移。

时间复杂度：$\Theta(W^3 \cdot n)$（其中 W 为坐标范围的最大值），期望得分：15 分。

2.3 算法三

我们预处理出哪些点是障碍物内部，这个可以通过二维前缀和方便地求出。

在 BFS 时，把每次击球拆开，让他一步一步地走。如果进行转向，那么我们才把这次转移看做一次新的击球，距离 $+1$，否则距离不变。

每次把距离不变的点放到队头，而距离 $+1$ 的点放到队尾，即可达到队列中所有点的距离单调不增的目的。

时间复杂度：$\Theta(n \cdot W + W^2)$（其中 W 为坐标范围的最大值），期望得分：35 分。

2.4 算法四

可以证明，我们每次停下的坐标一定在两个边界的交点上面（起点和终点也被认为是特殊的障碍物）。证明如下：

> 如果某一步操作我们在中间停下来了，我们可以改为到最近的那个交点再停下，而之后的操作不收影响。
>
> 这是因为初始停下的点不上边界的交点，所以这两个交点之间的范围（一个横向或者竖向的长条）都没有障碍物。

那么我们先把坐标离散化，然后从四个方向分别扫一遍，求出每个点在每个方向上的转移。然后进行 BFS 即可。

时间复杂度：$\Theta(n^2)$，期望得分：50 分。

2.5 算法五

对于满分解法，我们并不能对于每个交点都建出一个点，这样无论时间还是空间都已经爆炸了。

但是我们可以求出：每条极长的，且没有障碍物的线段，并对此进行转移。

考虑先求出每条线段：

> 我们先求出纵向的线段。对于一个矩形 $(x_1, y_1) - (x_2, y_2)$，我们把它变为四个事件：在 x_1 时加入 y_1, y_2，以及在 x_2 时删除 y_1, y_2。在加入前以及删除后，我们求出此时 y_1 所在的线段即可。
>
> 因为矩形个数是 $\Theta(n)$ 的，所以事件个数也是 $\Theta(n)$ 的。每次求出线段只要在 set 中二分即可，因此这一部分的复杂度为 $\Theta(n \log_2 n)$。

在转移时，需要快速求出与当前所在的线段相交，且没有访问过的线段有哪些。考虑使用树套树优化这一过程：

我们在预处理中，先把横向和竖向的线段分别插入两棵树套树中：以竖向的线段 (x, l, r) 为例，我们先把 $[l, r]$ 在外层线段树中分解成 $\Theta(\log_2 n)$ 条线段，然后在这些区间对应的内层线段树中找到 x，并把当前这条线段加入这个节点。

因为一共只有 $\Theta(n)$ 条线段，每个插入的时间复杂度为 $\Theta(\log_2^2 n)$，故这一部分的时间复杂度为 $\Theta(n \log_2^2 n)$。

然后对于每条这样极长的线段，我们进行 BFS。考虑如何进行转移：

我们还是以竖向线段 (x, l, r) 为例：我们先在外层线段树中找到包含 x 的区间，然后在内层线段树中暴力找到 $[l, r]$ 中的每个有内容的叶子，进行转移并删掉这些叶子。

在这个过程中，如果某个节点对应的叶子都没有内容，那么我们直接结束这一次的递归。这样因为一共只有 $\Theta(n \log_2 n)$ 个叶子有内容，每个叶子只会被查询一次，需要 $\Theta(\log_2 n)$ 的复杂度。故这一部分的总复杂度也为 $\Theta(n \log_2^2 n)$。

总复杂度：$\Theta(n \log_2^2 n)$，期望得分：100 分。

3 串串划分

Author: FizzyDavid

3.1 命题背景

这道题是一道原创题。作者之前在训练中遇到了循环串相关的问题，独立思考出了一种处理循环子串的时间复杂度优秀的算法。

子任务 2 的分析中的线性状态数 dp 是这道题的基础做法，这道题的本质是对这个做法的优化。

3.2 一些约定

- 一个字符串 S，它的长度是 $|S|$。
- 字符从 1 开始编号，$S[i]$ 表示第 i 个字符。
- 两个字符串相等当且仅当他们的长度与编号相同的字符都相等。
- 一个 l 到 r 的子串用 $S[l...r]$ 表示，注意为了方便，当 $l \geq r$ 时 $S[l...r]$ 为空串。
- 若 $S = \overline{S_1S_2...S_k}$，且 $S_i = S_{i+1}(1 \leq i < k)$，那么 $S_1, S_2, ..., S_k$ 就是 S 的循环节，它们的长度是 S 的循环节长度，k 称为循环次数。注意一个串可能有多个循环节长度和循环次数，并且至少有一个循环节长度和循环次数。

3.3 子任务 1

3.3.1 数据范围

$|S| \leq 300$

3.3.2 期望时间复杂度

$\Theta(n^3)$

3.3.3 算法分析

本题对于我们划分的每个子串的要求有两个，第一是相邻的不相同，第二是不循环。若我们预处理出哪些子串是不循环的，就可以用 $\Theta(n^3)$ 的简单 dp 来计算答案。dp 可以记录最后一次划分的子串是什么，状态数 $\Theta(n^2)$，每次转移 $\Theta(n)$。

考虑如何预处理出哪些子串是不循环的。我们可以枚举循环子串最靠左的一个循环节，再从小到大枚举次数，每次判断新增的循环节是否与枚举的循环节的一样，这样就可以获得所有的循环子串。对于一个左端点，枚举的次数是一个调和级数，该部分时间复杂度为 $\Theta(n^2 \log n)$。

注意以上做法涉及到判断两个子串是否相等，可以使用字符串哈希来做到 $\Theta(n)$ 预处理 $\Theta(1)$ 询问的复杂度。总复杂度为 $\Theta(n^3)$。

3.4 子任务 2

3.4.1 数据范围

$|S| \leq 2000$

3.4.2 期望时间复杂度

$\Theta(n^2 \log n)$

3.4.3 算法分析

子任务 1 中的做法的时间复杂度瓶颈在 dp，考虑如何优化。我们发现转移时相邻的相同的串最多只有一个，于是我们只需要计算任意划分的方案数减去相邻的相同的方案数。可以使用简单的前缀和优化做到转移 $\Theta(1)$，于是总时间复杂度可以降到 $\Theta(n^2 \log n)$。

由于这个方法的 dp 状态数高达 $\Theta(n^2)$，要想解决之后 $n \leq 200000$ 的数据规模，需要使用状态数更小的 dp。接下来的算法是后面几个子任务的基础。

我们考虑使用一个线性状态数的 dp，即 $dp(i)$ 表示合法的划分 $s[1...i]$ 的方案数。转移仍然使用枚举最后一个划分的子串是什么的方法，但是为了满足题目中 **相邻两个子串不相等**的条件，可以使用容斥来计算。

我们对题目中的条件 **相邻两个子串不相等**容斥后，就转化为限制某些相邻的子串相等。我们需要对于所有可能的限制，计算满足这些相邻的子串的相等的方案数，并且每个限制将会对方案数贡献 -1 的系数。容斥后的条件将子串序列划分成了若干个 **段**，满足每一段中的相邻两个子串存在限制，且相邻两段的子串之间没有限制。我们 dp 转移就使用 **段**来转移，即枚举最后一个子串所在的段的长度。

一个例子：

假设最后划分的子串序列为：$S_1, S_2, S_3, S_4, S_5, S_6$。

一种容斥后的限制为：$S_1, S_2 = S_3 = S_4, S_5 = S_6$。其中每个等号贡献了 -1 的系数。转移时将从初始状态转移到 S_1，再转移到 $\overline{S_1 S_2 S_3 S_4}$，再转移到原串。

我们可以先枚举最后一个子串是什么，再枚举最后一个 **段**是什么。由于枚举的子串要求是不循环的，所以等同于枚举的子串恰好为这个段所表示的子串的最小循环节。下面是 dp 转移方程：

$$dp_i = \sum_{l=1}^{i} \sum_{k=1}^{\lfloor \frac{i}{l} \rfloor} (-1)^{k-1} dp_{i-lk} [s[i-lk+1...i] \text{的最小循环节长度为} l]$$

其中 $[P]$ 是一个表达式，当 P 为真时值为 1，否则为 0。即枚举要满足最后一段的最小循环节长度恰好为 l。

记字符串 s 的最小循环节长度为 $\text{mnper}(s)$，最大循环次数为 $\text{mxrep}(s)$。这里一个串若不循环，我们定义它的最小循环节长度为串的长度，最大循环次数为 1。注意这里有 $\text{mnper}(s)\text{mxrep}(s) = |s|$。我们不妨改变一下枚举顺序，直接枚举 $j = lk$：

$$dp_i = \sum_{j=1}^{i} \sum_{l=1}^{j} (-1)^{\frac{j}{l}-1} dp_{i-j} [s[i-j+1...i] \text{的最小循环节长度为} l]$$

由于最小循环节长度唯一，所以推出式子：

$$dp_i = \sum_{j=1}^{i} dp_{j-1} (-1)^{\text{mxrep}(s[j...i])-1}$$

我们仍然可以用子任务 1 中的 $\Theta(n^2 \log n)$ 算法预处理出每个子串的最大循环次数。之后我们可以直接用这个转移方程在 $\Theta(n^2)$ 的时间求得答案。

3.5 子任务 3

3.5.1 数据范围

$|S| \leq 8000$

3.5.2 期望时间复杂度

$\Theta(n^2)$

3.5.3 算法分析

考虑优化子任务 2 中预处理的复杂度。这里我们需要使用最小循环节长度的性质：

- 设一个长度为 n 的字符串 s 的最大 border 长度为 p，若 $n-p$ 整除 n，那么就有 $\text{mnper}(s) = n-p$，否则 $\text{mnper}(s) = n$。

如果有 $0 \leq p < |s|$，$s[1...p] = s[n-p+1...n]$，那么 p 就是 s 的一个 border 长度，$n-p$ 就是 s 的一个 period。最大 border 长度就是满足条件的最大的 p。

KMP 算法可以对于每个前缀计算出它的最大 border 长度，也就是说可以对每个前缀计算出它的最小循环节长度和最大循环次数。那么我们在 dp 转移时，对前缀倒着跑一遍 KMP 就可以计算出该前缀的每个后缀的最大循环次数。总复杂度就降到了 $\Theta(n^2)$。

3.6 子任务 4

3.6.1 数据范围

$|S| \leq 200000$ 且 S 的每个字符从指定的某个字符集中随机生成。

3.6.2 期望时间复杂度

$\Theta(np)$，其中 p 为一个 60 左右的常数。

3.6.3 算法分析

首先特判掉字符全相等的情况。注意到一个随机串，在长度大的时候有极大的概率最大循环次数为 1。我们使用下面的转移方程:

$$dp_i = (\sum_{j=1}^{i} dp_{j-1}) - 2(\sum_{j=1}^{i} dp_{j-1}[mxper(s[j...i]) \bmod 2 == 0])$$

可以使用前缀和优化。由于 $[mxper(s[j...i]) \bmod 2 == 0]$ 在 $i - j + 1$ 大的时候几乎不可能为 1，所以我们设置参数 p，只枚举 $j \geq i - p$ 的情况。p 可以设为 60 左右。

3.7 子任务 5

3.7.1 数据范围

$|S| \leq 200000$ 且保证对于 S 的任意子串，要么它是不循环的，要么它的循环次数不会超过 10。

3.7.2 期望时间复杂度

$\Theta(n \log n)$ 或 $\Theta(n \log^2 n)$

3.7.3 算法分析

由子任务 4 中的做法启发，我们考虑找到那些循环次数大于等于 2 的子串。我们先枚举循环节长度 L。我们认为下标是 L 的倍数的位置是一个关键点。由于一个循环次数大于等于 2 的子串必然覆盖至少两个相邻的关键点，所以我们对于 kL 和 $(k+1)L$，计算它们向前的 lcp 与向后的 lcp。即我们找到最大的 $l1$ 与 $l2$，使得 $s[kL-l1+1...kL] = s[(k+1)L-l1+1...(k+1)L]$ 且 $s[kL...kL+l2-1] = s[(k+1)L...(k+1)L+l2-1]$。若 $l1+l2 \geq L+1$，那我们就找到了至少一个循环次数大于等于 2 的子串。

具体来说，设 $l = kL - l1 + 1$，$r = (k+1)L + l2 - 1$，那么对于满足下列条件的 l' 和 r'，$s[l'...r']$ 都有一个长度为 L 的循环节：

$$l \leq l' \leq r' \leq r,\ r' - l' + 1 \geq L,\ (r' - l' + 1) \bmod L = 0$$

我们可以用一个三元组 (L, l, r) 来表示循环节长度以及我们找到的区间。由于该子任务限制所有子串的循环次数不超过 10，所以循环次数大于等于 2 的子串个数不会很多（其实下面我们可以证明在该限制下不会超过 $\Theta(10n \log n)$ ），我们考虑直接记录所有循环次数大于等于 2 的子串，dp 转移时直接用我们记录的子串转移就好了。

接下来分析时间复杂度。关键点的个数是 $\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + ... = \Theta(n \ln n)$。如果我们使用后缀数组与 ST 表 $\Theta(n \log n)$ 预处理 $\Theta(1)$ 询问 lcp，那么求相邻关键点的 lcp 这一部分就可以做到 $\Theta(n \log n)$。

3.8 子任务 6

3.8.1 数据范围

$|S| \leq 200000$。

3.8.2 期望时间复杂度

$\Theta(n \log n)$ 或 $\Theta(n \log^2 n)$

3.8.3 算法分析

我们继续改进子任务 5 中的做法。对于上述做法中的一个三元组 (L, l, r)，表示 $s[l...r]$ 中所有长度是 L 的倍数的子串都存在一个长度为 L 的循环节，但是我们的 dp 转移式要求我们算出它的最小循环节长度（最大循环次数），而不是任意一个循环节。我们尝试修正我们的做法：

- 若 $s[l...l+L-1]$ 不是不循环的，那么 $s[l...r]$ 中长度是 L 的倍数的所有子串一定拥有一个更小的循环节。我们在遇到这种情况时就不处理 (L, l, r) 这个三元组。

这样我们就可以保证 L 是其中长度是 L 的倍数的子串的最小循环节长度。$s[l...r]$ 中可能会包含很多循环次数大于等于 2 的子串，但是由于右端点固定时，左端点是一个公差为 L 的等差数列，不难发现可以使用前缀和的技巧来优化转移，其中每个位置使用 s[l...r] 中的子串进行 dp 转移都可以做到 $\Theta(1)$。注意由于只有在长度大于等于 $2L$ 是循环次数才会大于等于 2，所以我们实际上需要转移的位置数量为 $r-l+1-2L$。

接下来我们来分析复杂度。关键点与 lcp 的部分的复杂度仍是 $\Theta(n \log n)$，我们只需要分析转移的复杂度，即 $r-l+1-2L$ 的和。我们发现 $r-l+1-2L$ 恰好为 $s[l...r]$ 中长度为 $2L$ 的子串的个数，即最大循环次数恰好为 2 且最小循环节长度为 L 的串的个数。我们对所有的 L 求和后，这个和就变成了最大循环次数恰好为 2 的子串个数。我们接下来证明这种子串个数最多只有 $\Theta(n \log n)$ 个：

对于所有左端点相同的子串，我们考虑那些满足条件的子串，并且列出他们的循环节长度 $l_i, l_2...l_k$，并且将其从小到大排好序。一个字符串若存在两个 period p_1, p_2，那么必然存在一个长度为 $\gcd(p_1, p_2)$ 的 period（$p_1, p_2 \leq |S|/2$）。而对于一个前缀 $s[1...len]$，如果存在一个长度为 p 的 period，那么 $l_i (l_i \leq len)$ 中不会出现大于 p 的倍数，不然就不满足条件了。

我们从后往前考虑 l_i 及 $s[1...2l_i]$ 的 period，首先对于 $s[1...2l_k]$ 肯定存在一个长度为 l_k 的 period，设我们知道的 $s[1...2l_i]$ 的 period 长度为 p，我们先将 p 设为 l_k。

当我们考虑到 $i(i < k)$ 时，有下面几种情况：

- 情况 1：$2l_i < p$ ： 将 p 设为 l_i。
- 情况 2：$l_i \neq p$ ： 将 p 设为 $\gcd(p, l_i)$。此时由于 l_i 不可能是 p 的倍数，所以 $2\gcd(p, l_i) \leq p$。
- 情况 3：$l_i = p$。

注意到情况 1、2 中 p 都减少了至少一半，而当 $p = 1$ 时则不可能有新的 l_i 了，情况 1、2 只会出现 $\Theta(\log n)$ 次。那么不同的 p 只有 $\Theta(\log n)$ 个，所以第三种情况只会出现 $\Theta(\log n)$ 次。而每个 $l_i(i < k)$ 都对应着某一种情况，所以 k 是 $\Theta(\log n)$ 的。所以总共只有 $\Theta(n \log n)$ 个最大循环次数恰好为 2 的子串。

至此，我们就分析出算法的总复杂度为 $\Theta(n \log n)$。

顺带一提，其实在这个算法中处理的三元组 (L, l, r) 都对应到不同的 run，根据 Runs Theorem，run 的个数是 $\Theta(n)$ 的，可以知道三元组的个数也是 $\Theta(n)$ 的。

4 总结

本场比赛:

- 题目难度适中，符合集训队选手的水平。
- 题面描述清晰易懂，没有歧义，题目背景贴近生活。
- 对于代码能力与思维能力的考察较为均衡，能够全面地考察选手水平。
- 部分分多样，有效区分了不同水平的选手，选手们都可以得到与自己水平相符的分数。

希望选手们都能从这场比赛中发现自己的优缺点并针对性地加以训练。出题人相信，这场比赛会为选手们准备即将到来的集训提供有力的援助。