《Fim 4》命题报告

浙江省杭州第二中学 吴瑾昭

摘 要

本文将介绍作者在集训队互测时命制的一道以字符串匹配为背景的题目《Fim 4》。题目深度挖掘了有关字符串周期的性质，并且将这个性质与大家常见的KMP算法结合起来。本题需要选手对字符串有比较深刻的理解，同时又需要一定代码能力。是一道考察选手思维深度的好题。

除此之外，题目的部分分设置合理，数据强度高，区分度高，能引导选手思考到正解。无论是只写了暴力的选手，还是稍微进行过思考的选手，以及深度研究找到了所有性质的选手，都能获得适度的部分分。

本文同时提供了一种命题思路：对于一种广为人知的算法，我们充分思考它的本质，找到原本算法中利用得较少的性质，从而把这个算法推广到更一般性的问题上。

1 试题大意及数据范围

1.1 原题目

1.1.1 原题目大意

给定 n 个字符串 s_i 和 m 个 $1 \sim n$ 之间的整数 a_i，令母串为 $s_{a_1} + s_{a_2} + \cdots + s_{a_m}$，回答 q 次询问，每次给出一个字符串 t_i，询问这个串在母串中的出现次数。

保证 s_i 和 t_i 都只由字母a,b组成。

1.1.2 原题目数据范围

对于所有的数据，满足：

$1 \le n, m, q \le 5 \times 10^4$

$\sum_{1 \le i \le n} |s_i|, \sum_{1 \le i \le q} |t_i| \le 10^5$ 。

- 子任务1（20分）：

  $\sum_{1 \le i \le m} |s_{a_i}| \le 10^6$

- 子任务2（20分）：

  $\max|t_i| \le \min|s_i|$

- 子任务3（30分）：

  $\max|t_i| \le 200$

- 子任务4（30分）：

  无特殊性质。

但实际上，作者给出的解法能解决比原题目更强的题意。加强版题目大意将在下面给出。不难发现原题目大意的询问严格是加强版题目大意的子集。所以我们只需要解决加强版的题目就能同时解决原题目。本论文也将主要介绍对于加强版题目的做法。

1.2 加强版题目

1.2.1 加强版题目大意

给定一棵有 n 个节点的Trie树。令根到第 i 个节点所形成的字符串为 s_i 。

再给定 m 个在 $1 \sim n$ 之间的整数 a_i 。

回答 q 次询问，每次给出一个字符串 t_i 和两个正整数 L_i, R_i 。令 $S_i = s_{a_{L_i}} + s_{a_{L_i+1}} + \cdots + s_{a_{R_i}}$ ，询问 t_i 在 S_i 中的出现次数。

字符集大小 ≤ 10 。

1.2.2 加强版题目数据范围

对于所有的数据，满足：

$n \le 10^6$

$1 \le m, q \le 5 \times 10^4$

$\sum_{1 \le i \le q} |t_i| \le 10^5$

- 子任务1（20分）：

  $\sum_{1 \le i \le m} \left|s_{a_i}\right| \le 10^5$

- 子任务2（30分）：

  $\max|t_i| \le 200$

- 子任务3（20分）：

  $\max|t_i| \le \min|s_{a_i}|$

- 子任务4（30分）：

  无特殊性质。

2 前置技能

这一部分我们将粗略介绍一些解决本题可能需要用到的定义与算法。

2.1 字符串基本概念

设一个长度为 n 的字符串 $s_1 s_2 \dots s_n$ 。对于 $1 \leqslant i \leqslant j \leqslant n$ ，称 $s_i s_{i+1} \dots s_j$ 为 s 的一个子串，记作 $s[i:j]$ ，记 $|s|$ 为 s 的长度，即 $|s| = n$ 。

对于 $1 \leqslant i \leqslant n$ ，称 $s[i:n]$ 为 s 的一个后缀，记为 $s[i:]$ 。类似地，对于 $1 \leqslant i \leqslant n$ ，称 $s[1:i]$ 为 s 的一个前缀，记为 $s[:i]$ 。

对于 $1 \leqslant i < n$ ，如果 $s[:i] = s[n-i+1:]$ ，则称 $s[:i]$ 是 s 的一个 border 。特别地，空串也是 s 的一个 border 。 s 最长的 border 称作 s 的 Lborder 。

如果 $2|Lborder(s)| \geqslant n$ ，则称 s 是周期串，周期的长度即 $|s| - |Lborder(s)|$ 。

2.2 一些记号

我们规定母串 S 为 $s_{a_1} + s_{a_2} + \cdots + s_{a_m}$ ， lp_i 表示 s_{a_i} 在 S 中的左端点位置， rp_i 表示 s_{a_i} 在 S 中的右端点位置。 L 为询问串最长的长度。 Σ 为字符集。

定义 $[cond]$ 的值当 $cond$ 成立时为 1 ，否则为 0 。

定义 $c(T,S)$ 表示字符串 T 在字符串 S 中的出现次数。

定义 $suf(S)$ 表示 S 的全体后缀组成的集合, $pre(S)$ 表示 S 的全体前缀组成的集合。

2.3 后缀自动机

一个串 S 的后缀自动机是一个有限状态自动机（DFA），它能且仅能接受所有 S 的后缀，并且拥有最少的状态和转移。我们简称为SAM。

对于一个串 S ，我们采用增量法构造，时间复杂度为 $O(|S|)$ 。具体构造方法和复杂度证明与本文主题无关我们不在此赘述，具体可见参考文献[5]。

2.4 SAM on Trie

对于一个Trie树，我们定义它的子串集合是它的根节点到任意一个节点所形成的字符串的后缀所形成的集合。

我们给Trie树建立一个有限状态自动机能识别Trie树上的所有子串，并尽可能的缩小自动机的状态数和转移数，我们称之为SAM on Trie。

SAM on Trie的性质与SAM在序列上的性质几乎完全一致。如：SAM on Trie可以识别一个Trie里所有的子串，SAM on Trie的parent树是一棵倒着的后缀树，每个点的父亲是这个点所代表的字符串里最长的本质不同的后缀（本质不同即在这个Trie里的出现次数不同）所代表的节点。

SAM on Trie的构造同序列上类似。每次每个节点以父亲在SAM上的节点为last来继续扩展。如果采用FIFO顺序构造，则时间复杂度为 $O(n|\Sigma|)$ 。其中 n 为Trie的节点数， Σ 为字符集。对于其复杂度的证明因为与本文主题无关我们不在此赘述，具体可见参考文献[1]。

2.5 KMP

使用 KMP 算法可以在 $O(|s|+|t|)$ 的时间内计算出字符串 t 在字符串 s 中的出现次数。除此之外，它还能计算出所有 $Lborder(t[:i])(1 \leqslant i \leqslant n)$ 。具体算法流程如下：

Algorithm 1 KMP Algorithm

$j \leftarrow 0$
$np \leftarrow 0$
for $i \leftarrow 2$ to $|t|$ **do**
  while $t_i \neq t_{j+1}$ and $j \neq 0$ **do**
    $j \leftarrow p_j$
  end while
  if $t_i = t_{j+1}$ **then**
    $j \leftarrow j+1$
  end if
  $p_i \leftarrow j$
end for
$j \leftarrow 0$
for $i \leftarrow 1$ to $|s|$ **do**
  while $s_i \neq t_{j+1}$ and $j \neq 0$ **do**
    $j \leftarrow p_j$
  end while
  if $s_i = t_{j+1}$ **then**
    $j \leftarrow j+1$
  end if
  if $j = |t|$ **then**
    $pos[np] \leftarrow i$

```
    np ← np + 1
    j ← p_j
  end if
end for
```

2.6 快速求两个串的最长公共前后缀

我们要解决一个这样的问题：给定一个Trie和一个字符串 T ，快速回答：给定Trie上某个节点，求一个最长的字符串 T' ，使得 T' 是 T 的前缀，并且 T' 是这个Trie的根节点到这个节点所形成的字符串的后缀。

我们先建立这个Trie的SAM，考虑对于 T 的每个前缀，我们都在它在Trie上对应的节点里打上一个权值为当前前缀长度的Tag。

当我们询问时，我们只需要找到Trie的根节点到这个节点所代表的字符串在SAM上的对应节点中，具有最大权值Tag的parent树上祖先即可。我们只需要用一棵LCT来维护parent树就可以在$O(\log n)$ 的时间内支持这个操作。

3 算法介绍

3.1 算法一

对于Subtask1，注意到母串的总长度在 10^5 范围内。所以直接构造出母串的时间复杂度是可以接受的。

首先我们先构造出母串的后缀自动机。

然后我们用可持久化线段树合并来预处理出parent树上每个节点的Right集合。

每一次询问时，我们只需要找到询问串 t 在parent树上对应的节点，查询它的Right集合里在区间 $[L+|t|-1,R]$ 的个数即可。因为我们之前已经用可持久化线段树预处理了每个节点的Right 集合，所以我们只需要在这个节点对应的线段树上查询这个区间内的和即可。

时间复杂度 $O(\sum |s_{a_i}| \log n + \sum |t_i| + q \log n)$ 。

期望得分 20 分。

3.2 算法二

考虑Subtask2，注意到最长的询问串长度只有 ≤ 200 。

我们考虑到询问串在母串中的匹配，要么是询问串在单个字典串内出现，要么询问串横跨了若干个个字典串，我们对两部分分别处理。

3.2.1 Part 1

我们每次给出一个询问串 t，要计算 $\sum_{i=L}^{R} c(t, S_i)$。这是一个比较常见的问题。我们对给定的Trie建出它的SAM on trie。

类似解法一，我们对于SAM上的每个节点，我们用可持久化线段树合并来预处理出parent树上每个节点所代表的字符串在 s_{a_i} 中的出现次数。

每次询问时找到询问串 t 在parent树上对应的节点，在这个节点上对应的线段树上查询区间 $[L, R]$ 的区间和就可以了。

这一部分的复杂度是 $O(n|\Sigma| + n\log n + \sum |t_i|)$。

3.2.2 Part 2

形式化地说，计算跨越多个字典串的出现次数也就是要求我们来计算 $\sum_{i=L}^{R-1} c(t, S[\max(lp_i, rp_i - |t| + 2) : \min(rp_R, rp_i + |t| - 1)])$。

我们可以把所有 $S[\max(lp_i, rp_i - L + 2) : rp_i + L - 1]$ 拿出来建Trie，并对该Trie建出SAM。对于前缀 $S[\max(lp_i, rp_i - L + 2) : rp_i + k - 1]$ 我们将其标号为 k，且令它的下标为 $rp_i + k - 1$。统计 $\sum_{i=L}^{R-1} c(t, S[\max(lp_i, rp_i - |t| + 2) : \min(rp_R, rp_i + |t| - 1)])$ 也就是相当于在统计 parent树上 t 所代表的节点的子树中有多少节点的标号在 $[2, |t| - 1]$ 中，且下标在 $[lp_L + |t| - 1, rp_R]$ 中，对parent树求dfs序后可以转化为三维数点问题。

我们注意到这个三维数点的“标号”维是 $O(L)$ 级别的。并且总点数是 $O(mL)$ 级别的，询问是 $O(q)$ 级别的。我们可以轻松地通过离线做到加入$O(\log n)$ 询问 $O(L\log n)$，因此这一部分的总复杂度是 $O(mL\log n)$ 的。

综上所述，我们可以在 $O(n\log n + \sum |t_i| + mL\log n)$ 的复杂度内解决这个Subtask，期望得分30分。

3.3 算法三

对于Subtask3，注意到最长的询问串长度也小于等于最短的字典串长度。

这给我们带来了一个这样的性质，即询问串最多只会横跨两个字典串。

我们沿用算法二的想法，还是分别统计字典串内的和横跨字典串的。

我们设置一个 B，对于长度 $\le B$ 的询问我们在用解法二中介绍的算法解决，而对于询问串长度 $> B$ 的算法我们用另一种算法来解决。

我们继续沿用算法二，考虑对于询问串长度 $> B$ 的询问。

定义 $\tau(s, t) = \text{suf}(s) \cap \text{pre}(t)$。

定理 3.1. 令 $\tau(s, t) = \{s[p_i :]\}$，其中 p 是一个单调上升的序列，则 p 可以被表示成 $O(\log n)$ 个等差数列顺次连接起来。

证明. $s[p_{i+1}:]$ 是 $s[p_i:]$ 的前缀，因此只有两种情形：

1. $\left|s[p_{i+1}:]\right| \le \left|s[p_i:]\right|/2$:长度减半，一共只会减半 $O(\log n)$ 次，不需要处理。

2. $\left|s[p_{i+1}:]\right| > \left|s[p_i:]\right|/2$: $s[p_i:]$ 必然是周期串，我们可以把形如 $p_k = p_i + (k-i)l$ 的位置压成一个等差数列，其中 l 是 period 长度。则处理后 $\left|s\left[p_j:\right]\right| < \left|s[p_i:]\right|/2$ （其中 $j > i$ 是第一个不在等差数列里的下标），同样也只会减半 $O(\log n)$ 次。 □

我们要计算每个串在 s_{a_i} 与 $s_{a_{i+1}}$ 之间的出现次数，可以先算出 $\tau(s_{a_i}, t)$ 和 $\tau(t, s_{a_{i+1}})$ ，然后把两部分结果合并起来。注意到这 $\log n$ 段等差数列的值域一定是互不相交的，因此我们可以用类似归并排序的办法来合并，每次合并复杂度 $O(\log^2 n)$ 。

于是问题来到怎么求 $\tau(s_{a_i}, t)$ 。（求 $\tau(t, s_{a_{i+1}})$ 是对称的，用同样的做法就可以了）

我们对于字典串建出Trie，然后再对这个Trie建出SAM。在 $O(m \log n)$ 的时间内对于每个 i 求出 $\max\{\text{LCP}(s_{a_i}[j:], t)\}$ （具体计算方法在前置技能中有介绍）。这就是我们要计算的 p_1 。我们再用KMP预处理出 t 的next数组。注意到 $s_{a_i}[p_i:]$ 一定是 $s_{a_i}[p_1:]$ 的border。所以我们就可以利用这个next数组来构造出整个 p 数组。

当 B 取到 $\sqrt{m}\log^{1.5} n$ 的时候，整个算法的时间复杂度最小，为 $O(n|\Sigma| + n\log n + m\sqrt{m}\log^{1.5} n)$ 。

期望得分20分,结合算法一，二可以获得70分。

3.4 解法四

我们继续沿用解法二，我们考虑对于询问串 $> B$ 的询问。

我们先介绍两个引理：

引理 3.1. *如果 p, q 都是字符串 S 的周期，且 $p + q \le |S|$,那么 $\gcd(p, q)$ 也是字符串 S 的周期。*

证明. 令 $d = q - p(p < q)$ ，则由 $i - p > 0$ 或 $i + q \le |S|$ 均可推出 $S_i = S_{i+d}$ 。

□

引理 3.2. *令 l 是周期长度，对于 $i > 2l, next_i = i - l$ 。*

证明. 这等价于对于 $i > 2l$ 的前缀，它的最短周期是 l 。我们使用反证法：如果存在比 l 更短的周期 T ，那么 $T + l < i$,所以由引理3.1可得 $\gcd(T, l)$ 也是这个串的周期，注意到 $\gcd(T, l) | l$ ，这与 l 是整个串的最短周期矛盾，故得证。

□

考虑优化KMP算法。我们考虑把询问串放在母串上跑KMP。我们注意到我们只需要计算串之间的匹配就可以了。

我们令 i 表示当前KMP算法指向字典串的指针， j 表示当前KMP算法指向询问串的指针, L 表示询问串的长度。

如果存在某个 k，使得 $i \in [lp_k, rp_k]$ 且 $i-j+1 \in [lp_k, rp_k]$,那么说明当前正在匹配的是串内的，是不需要被计算的。所以我们直接令 j 等于最长的询问串前缀的长度，并且这个询问串的前缀是 s_{a_k} 的一个后缀，再把 i 设为那个后缀开始的地方再继续KMP即可。

由于我们可以快速求出两个后缀的 LCP，因此我们可以很容易地计算下一次失配的位置。因此我们只需要优化跳 next 的过程即可。

在失配之后如果 $next_j < \frac{j}{2}$ 我们就直接执行 $j := next_j$，否则：

如果执行 $j := next_j$ 后还会失配，根据引理3.2，我们可以直接跳到 $l + j \bmod l$ 的位置（l 是周期长度）(Case (1))。

否则我们统计接下来 $S[i+1:]$ 还能继续匹配多少个 $t[:j]$ 的周期并找到失配时的位置，此时有两种可能的情况，第一种情况是 $S[i+1] = t_j$ (Case (2))，此时我们继续向下匹配，第二种情况同之前 Case (1)。

有一个特殊情况是我们匹配到了串末。这时候我们可以计算 $S[i+1:]$ 还能匹配多少个 $t[:j]$ 的周期，能匹配的周期数就是对答案的贡献。之后我们和失配情况一样处理即可。

定理 3.2. *之前所述的算法的时间复杂度是 $O(m \log n)$ 的。*

证明. 对于Case (1)，考虑 $j-(i-lp_k)$ 的值。显然当 $j-(i-lp_k) \le 0$ 时，我们就会重置 i 和 j 的值。

首先，求LCP时，$j-(i-lp_k)$ 的值并不会变。

而在跳next时，对于 $next_j \le \frac{j}{2}$ 时，执行 $j := next_j$ 会让 j 变小 $\frac{1}{2}$。对于 $next_j > \frac{j}{2}$ 时，当我们执行 $j := l + j \bmod l$ 时，$l + j \bmod l \leqslant \lceil \frac{2l}{3} \rceil$，$j$ 会变小 $\frac{1}{3}$。所以每次Case（1）操作就会让 $j-(i-lp_k) \le 0$ 变小至少 $\frac{1}{3}$。

所以我们发现Case (1)的总复杂度是 $O(m \log n)$ 的。

对于Case (2)，我们先引入一个关于层的定义：

我们令 $b_i = \left[next_i \geqslant \frac{i}{2} \right]$，将 b 序列中从左到右第 i 段连续的 1 称作第 i 层。

我们再证明一个有关层的性质：

定理 3.3. *令第 i 层的前缀的周期为 per_i，那么有 $\frac{3}{2}|per_i| \le |per_{i+1}|$*

证明. 令 pos_i 表示第 i 层最右边的位置，那么显然 per_i 和 per_{i+1} 都是 $S[:pos_i]$ 的周期。

如果 $|per_i|+|per_{i+1}| \le pos_i$，那么根据引理3.2,可以知道 $\gcd(|per_i|, |per_{i+1}|)$ 也是 $S[:pos_i]$ 的周期。注意到$\gcd(|per_i|, |per_{i+1}|)$ 整除 $|per_{i+1}|$，这与 per_{i+1} 是第 $i+1$ 层的最短周期矛盾，故我们得到 $|per_i| + |per_{i+1}| > pos_i$

又有 $pos_i \ge 2|per_i|$，联立两式可得 $\frac{3}{2}|per_i| \le |per_{i+1}|$。

□

那么对于Case (2)，而根据定理3.3显然一共只有 log 层。注意到只有两种操作，而Case 1的操作每次只会倒退一层,Case 2的操作每次只会增加一层。所以Case (2)的操作最多只有 $m \log n$ 次。

因此我们的总失配次数是 $O(m\log n)$ 的，而对于求LCP部分，我们建出字典串的SAM on Trie后直接查询对应他们在SAM上节点的LCA就做到查询他们的LCP。注意到LCA问题可以转化成欧拉序上的RMQ问题，是可以做到 $O(1)$ 回答的。而对于重置 i, j 部分，我们已经在前置技能中已经介绍了如何在 $O(\log n)$ 的时间内求两个串的最长公共前后缀，所以我们总的时间复杂度也是 $O(m\log n)$ 的。 □

综上所述，我们取 $B = \sqrt{m}\log n$ 时复杂度最低，为 $O((q+m)\sqrt{m}\log n)$ 。

4 数据生成方式

因为这是与字符串匹配有关的相关问题，直接随机的话会导致整个串几乎没有period，会让暴力轻松通过，所以需要一种比较精细的构造办法。

4.1 字典串的构造方式

基本思路是先随机出若干个比较小的串，使得它的period尽量小。之后把这个小串复制若干份得到一个大串。然后把这个大串切片切成若干个字典串。注意切的时候要让最长的字典串不能太小，并且字典串个数也不能太少。之后我们再生成一些字典串，它们也是同一个period的若干倍，并且随机加入若干噪点。之后我们把生成的串塞入一个Trie内即可。

4.2 a数组的构造方式

构造a数组时，可以以某个概率来决定接下里的一段是带不带噪点的串。这样可以既避免了一个询问串只会跨越几个相邻的串，也避免了一个串直接匹配到底的情况。

除此之外，也要保证 $\sum_{i=1}^{m}\left|s_{a_i}\right|$ 要足够大。

4.3 询问串的构造方式

之后，对于询问串，我们也让它的period等于我们最开始随机的那若干个个小串之一。然后以一个较低概率的加入一个噪点。

对于询问串的长度。也要做到长的询问串和短的询问串都不能太少。同样的我们有三种随机方式：第一种是全都是比较长的串，第二种是全都是比较短的串，第三种是长度比较均匀，长度不同的串要尽量多。

4.4 其他构造方法

为了更好地覆盖到一些的Corner Case，还有一些手构的数据。比如全A串，随机串等等。

5 命题契机和过程

这道题是我与张宇博同学一起讨论并命题的。

张宇博同学首先想到了这道题最开始的题面。我们认为这样一道题意简洁的题一定也会有一个优美简单的做法。在最开始的思考过程中，我和张宇博同学首先想到的做法是解法三。但后来我们发现如果没有每个询问串都只会跨过两个匹配串这个性质的话，我们并不会在很好的复杂度内求出 $\tau(t, S[i:])$ 。

之后我们又回顾了我们所学过的字符串匹配算法，希望从中得到启发。最终我们发现了KMP算法中next数组与字符串period是互补关系这一良好的性质，这可以极快地加速我们匹配的过程。所以我们在KMP算法的基础上加以优化，添加了若干补丁之后得到了解法四。

在子任务的设置方面，我们也特意地给标算的若干个部分以及我们在思考这题的过程中所拐过的弯路都分别设置了部分分，来引导选手思考到标算。

在集训队互测中，因为考虑到代码复杂程度的原因，我们在把原题削弱了三个版本之后放到了集训队互测里。

对于解法三，我和张宇博同学都只会在每个询问串都只会跨过两个匹配串这个性质下解决这个题目。对于这个解法有没有更好的扩展我们都没有一个较好的结论，欢迎大家和我讨论。

6 总结

题目深度挖掘了有关字符串周期的性质，并且将这个性质应用到了有关字符串匹配中去。

对于题目的四个子任务，子任务一是朴素算法，知识、思维、代码难度都很低。子任务二需要利用SAM on trie的相关知识，但对题目稍加分析后也不难得到这个解法。子任务三则需要选手对于border性质的理解，难度稍加提升了一些。子任务四除了要掌握传统算法之外，还要求了选手能够在观察性质之后在传统算法上创新的能力，对选手的字符串水平是一个比较高的考验，无论是只写了暴力的选手，还是稍微进行过思考的选手，以及深度研究找到了所有性质的选手，都能获得适度的部分分。

而在实际考试中,有4位同学获得了20分，1位同学通过了此题。我在赛后与若干同学交流了他们有关这题的想法。发现有大多数同学想到了算法二，有少数几个同学想到了算法

三。但是因为各种原因只写了最基本的暴力。可见本题具有良好的区分度。

总体来说，这是一道既需要选手掌握了一些字符串中常见性质，又需要选手熟练掌握基本字符串算法。同时，这题还有一个比较大的代码难度，总体难度接近近年的CTSC。

就我个人而言，我非常喜欢算法四的有关想法。KMP是一个经典的算法。我们在仔细挖掘了KMP的性质之后把KMP算法应用在了这样新的场景上。考察了大家对于这个基本算法的掌握和理解。这提醒我们我们对于某个耳熟能详的算法，我们也许可以继续挖掘它的性质，从而出出来一些有趣的题。

7 感谢

感谢张宇博同学对撰写本文提供的巨大帮助，也同时感谢他在OI路上对我一直以来的帮助与陪伴。

感谢中国计算机学会提供学习和交流的平台。

感谢国家集训队教练张瑞喆教练的指导。

感谢李建老师以及父母多年来的关心和指导。

感谢王修涵、吴越、董炜隽、李宁远同学为本文验稿。

参考文献

[1] 刘研绎,《后缀自动机在字典树上的扩展》, 2015年国家集训队论文。

[2] wikipedia，Knuth–Morris–Pratt algorithm，https://en.wikipedia.org/wiki/KMP_algorithm

[3] 金策，《字符串算法选讲》，2017年NOI冬令营。

[4] 王鉴浩，《浅谈字符串匹配的几种方法》，2015年国家集训队论文。

[5] 张天扬，《后缀自动机及其应用》，2015年国家集训队论文。