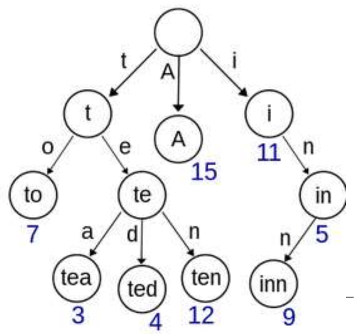
字符串算法选讲

徐沐杰

南京大学

字符串概念

字符串即只有「常数种元素」的序列。



字符串相关概念

字符串：由常数种「字符」构成的一个序列。

字符集：字符串元素的值域，通常认为字符集大小为一小常量。

字典序：如果字符串元素的值域上有全序，那么字典序是以长度为第一关键字，不同的第一个字符为第二关键字的顺序。

前缀：从字符串某个元素开始往前的所有元素构成的序列。

Litrehinn

后缀：从字符串某个元素开始往后的所有元素构成的序列。

Bardisk

真前/后缀：不是原串的前/后缀。

确定有限状态自动机：状态数有限，有一起始状态，且所有状态的后继是确定的，可以根据某一决策/转移序列得到固定的状态。（我们今天会接触到很多自动机的例子）

字符串哈希

不仅仅是哈希：

线性预处理，常数计算任一子串的哈希值。

字符串哈希：实用性介绍

其实就一句话（把字符串看作 b 进制数）： $f(s)=\sum_{i=1}^{l} s[i]\times b^{l-i} \pmod{M}$

其中 s[i] 是字符的序数，b 一般选取一个小质数（如 233），M 是一个大质数（如 998244353，919260817，1e9+7）等。

为了实现方便也可以使用自然溢出（即不取模直接加，计算机自动对 2^64/2^32 取模）。

自然溢出容易被卡，但单哈希也容易被卡，最保险的选择是哈希两次，两个都对上才判断字符串相等。

字符串哈希的算法已经介绍完了，我们这节不是数学课，不讨论哈希的数学细节。

字符串哈希：实现

对于任意子串的哈希值，前缀和就行。

一个 [l, r] 子串的哈希值：
r 前缀哈希值 – (l-1 前缀哈希值 * b^(r-l+1))

不能再简单了吧。

```cpp
typedef unsigned long long ull;
const int B=131;
char a[N];
int n;
ull hs[N],pw[N];
void init(){
  scanf("%s",a+1);
  n=strlen(a+1);
  for(int i=1;i<=n;++i)hs[i]=hs[i-1]*B+a[i];
  pw[0]=1;
  for(int i=1;i<=n;++i)pw[i]=pw[i-1]*B;
}
ull hash(int l,int r){
  return h[r]-h[l-1]*pw[r-l+1];
}
```

过五关斩六将

【题1】求出模式串 S 在文本串 T 中的所有匹配位置。

【解1】由于字符串哈希可以快速实现子串比较，所以只需枚举所有匹配位置 O(1) 比较。

【题2】求出串 S 的最长回文子串。

【解2】枚举回文中心，显然答案单调，我们预处理正着看的 hash 和反着看的，这样就可以 O(1) 确认某点两侧 k 长的部分是否回文。二分答案即可 O(|S| log S) 计算。

【题3】求出总长不超过 n 的若干串的最长公共子串。

【解3】二分答案，把长度为 k 的所有子串全部放到 unordered_map 里求交集就可以检验答案了。复杂度 O(n log n)。

过五关斩六将

【题4】维护一个字符串 Set，支持插入字符串和检查字符串是否存在。

【解4】结合整数的 Set，把哈希值用 set<ull> 维护即可。

【题5】求一个字符串的最短循环节（必须完整/可以不完整）。

【解5】发现循环节为 d 即 S[1 … n-d] = S[d+1 … n]，枚举因数哈希判断即可，如果可以不完整就枚举所有 1~n-1 的整数进行判断。

除了这些基本题外，若允许对字符串进行增删改查，那么需要用数据结构维护字符串哈希的前缀和（和与积信息），数据结构不是这堂课的内容，不多赘述。

KMP

传说中「需要三个小时理解」的算法
本质就是简单的前缀函数。

前缀函数：引入

定义：$\mathrm{l(s)} = \max\limits_{\{i<|S|, pre(i)=suf(i)\}}(i)$，其中 $suf(i)$和 $pre(i)$ 分别是 s 的第 i 个后缀/前缀。

一个词理解：「最大公共真前后缀」的长度。

如：abcacab，最大公共真前后缀是 ab，长度为 2。

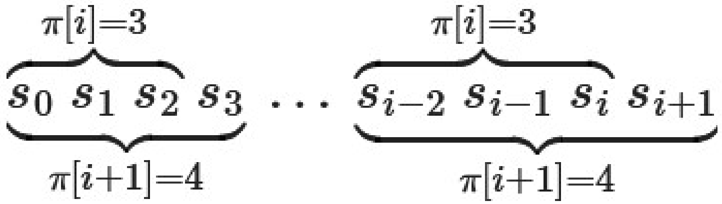
再定义前缀函数 $\pi(i) = l(pre(i))$。

人话：「某一前缀的前缀函数是其的最大公共真前后缀的长度」。

如何计算？

暴力枚举 $O(|\mathrm{S}|^2)$，太慢了。

细细研究一下「最大公共真前后缀」到底是求什么。



计算最大公共真前后缀就像把一个字符串平移到后面和自己匹配，且希望平移量最少。

最大公共真前后缀就是这「匹配上的部分」，于是新加一个字符就可以递推。

观察到 $\pi(i)$ 要么因 $S_{\pi(i-1)+1} = S_i$ 而有 $\pi(i) = \pi(i-1)+1$，要么就必须考虑从 $pre(i-1)$ 的更小的公共前后缀（也就是「失配」），我们只能寻找次小公共前后缀。而是否采纳 $pre(i-1)$ 的 k 公共前后缀取决于是否有$S_{k+1} = S_i$ 。

如何找次小？

容易发现，一个字符串的次长公共真前后缀，就是它的最长公共真前后缀的最长公共真前后缀~~（不行太绕了我要晕了，之后把公共真前后缀称作CPS）~~。

前缀函数：递推

计算最大公共真前后缀就像把一个字符串平移到后面和自己匹配，且希望平移量最少。

最大公共真前后缀就是这「匹配上的部分」，于是新加一个字符就可以递推。

观察到 $\pi(i)$ 要么因 $S_{\pi(i-1)+1} = S_i$ 而有 $\pi(i) = \pi(i-1)+1$，要么就必须考虑从 $pre(i-1)$ 的更小的公共前后缀（也就是「失配」），我们只能寻找次小公共前后缀。而是否采纳 $pre(i-1)$ 的 k 公共前后缀取决于是否有$S_{k+1} = S_i$ 。

如何找次小？

容易发现，一个字符串的次长公共真前后缀，就是它的最长公共真前后缀的最长公共真前后缀~~（不行太绕了我要晕了，之后把公共真前后缀称作CPS）~~。

abcabddbabcab

前缀函数：实现

```cpp
vector<int> prefix_function(string s) {
  int n = (int) s.length();
  vector <int> pi(n); // 预留长度为 n
  for (int i = 1; i < n; i++) { // 递推
    int j = pi[i - 1]; // 第一候选是 i-1 的 LCPS
    while (j > 0 && s[i] != s[j]) j = pi[j - 1]; // 失配，找次大 CPS
    // 匹配成功，计算新的 pi 值
    if (s[i] == s[j]) j++;
    // 否则说明一个都匹配不了，pi(i) = 0
    pi[i] = j;
  }
  return pi;
}
```

前缀函数：复杂度分析

直觉：这个函数存在两个 For 循环，复杂度应该是 $O(|S|^2)$ 吧？

反直觉：复杂度其实是 $O(|S|)$ 的。

刚刚提到，计算 LCPS 的就是把一个字符串平移到后面和自己匹配，且希望平移量最少。

在递推的过程中，我们不断地往前缀的末尾添加字符，然后看上一次平移到后面的串「还能不能和自己匹配」。如果能匹配最好，如果不能匹配就继续向后匹配。

「失配」就是不能匹配，需要继续向后平移的过程。

abcabddbabcab

前缀函数：复杂度分析

复杂度 = 匹配次数（|S|）+ 失配次数

「失配」会出现多少次？

前缀函数的值可以理解为「匹配上的长度」。

每次失配，这个长度都要缩短；每次匹配，这个长度只会增加 1。

答案呼之欲出：

由于前缀函数的值总大于 0，所以失配不可能超过 |S| 次。

直观理解，就是字符串平移的量不可以超过 |S|。

复杂度 O(|S|)。

abcabddbabcab

Knuth–Morris–Pratt

台下同学：我们不是要讲 KMP 吗？你扯什么前缀函数？RNM 退钱！

然而 KMP = 前缀数组模板。

先考虑 KMP 要解决什么问题。

有一模式串 S，文本串 T，问模式串在文本串中的出现次数，也就是模式串在文本串子串中的出现次数，也就是模式串是文本串多少个前缀的后缀。

于是当我们扫描文本串时，只需要进入一个 S 的「前缀自动机」，对于 T 的每个前缀，我们都相当关心它的后缀和 S 的前缀的匹配情况。

这即是所谓「匹配的部分」，每当后面新增加一个元素，匹配则然，不匹配则将S后移，容易发现和求前缀数组时一样，每次都移动到匹配部分的 LCPS 的位置。

Knuth–Morris–Pratt

这样做，本质上就是在求 S#T 的前缀函数啊！（# 是 S、T都没有的字母）

复杂度 O(|S| + |T|)。

模板直接套用刚刚的前缀函数模板就行。

我们接下来可能会混称 KMP/前缀函数，它们都指这种求所有前缀 CPS 进行匹配的做法。

Censoring

【题目描述】

给出字符串 S, T，每次不断在 T 中删掉 S 的第一次匹配，问得到的最终 S 串。

大家思考三分钟~

提示：难点在于删除时需要重新匹配。

来源：USACO 2015 Feb. Silver

Censoring

【题目解答】

删除的时候怎么回退呢？

正解的想法是在匹配时，把文本串的前缀函数也保存下来，这样就可以方便地回退状态了，这个想法非常自然且优雅，复杂度显然是 O(|S| + |T|) 的。

但是不能更暴力点么？

直接删除的时候回退 |S|，这样复杂度也是 $O(|S| + \frac{|T|}{|S|}|S|)$ 的（逃）。

字符串循环节

【题目描述】

给定字符串 S，求 S 的最小周期。|S| <= 1000000

称字符串 T 是 S 的循环节，当且仅当 S 是 T 无限重复的一个前缀。

周期定义为循环节的长度。

可以拿出纸笔思考五分钟。

来源：[BOI2009] Radio Transmission

字符串循环节

【题目解答】

KMP（或谓前缀数组）求出的是所有前缀的 LCPS，下面我们用 border（边框）来称呼它。

串长减去 border 长是什么呢？

是一个串向后平移能匹配上自己的长度。

由于匹配上了自己，所以再往后平移相同的长度仍然能匹配上自己。

所以串长减去 border 长，就是循环节的长度。

于是，最大 border 就对应了最小循环节。

abcabddbabcab

字符矩阵循环节

【题目描述】

给定 R 行 C 列的字符矩阵 S，求 S 的最小周期。R <= 10000, C <= 75

称字符矩阵 T 是 S 的循环节，当且仅当 S 是 T 无限重复的一个前缀矩形。

周期定义为循环节的面积。

可以拿出纸笔思考三分钟。

来源：[USACO2003NOV] 挤奶阵列

字符矩阵循环节

【题目解答】

行方向的循环节是你选择的每个列循环节的最小公倍数。

列方向的循环节是你选择的每个行循环节的最小公倍数。

注意到每行每列的循环节都可以被求出，同时注意到除了 R，C，所有循环节之间都有倍数关系。如果某行/列的循环节取了 R/C，那么显然其他行/列也取 R/C 为最佳。

如果都不取 R/C，那么都取最小循环节为宜。（注意行列分开考虑）

那么，行方向的循环节就是 min {R, 所有列最小循环节的最小公倍数}，列方向的循环节就是 min {C，所有行最小循环节的最小公倍数}

离散模式匹配

【题目描述】

定义两个串相等当且仅当它们各个位置上的序关系相同。

在此基础上求出 S 在 T 内的所有匹配。 |S|, |T| <= 100000, 字符集大小为 26。

可以拿出纸笔思考五分钟。

来源：[USACO2005DEC] 牛的模式匹配

离散模式匹配

【题目解答】

我们仍然可以用前缀数组/KMP的方式来做，但是存在一个问题：如何判断子串相等？

在 KMP 中递推前缀函数的时候可以利用某位相等来匹配子串，在这里，我们只能采用维护一个子串的字符桶的方式来判定相等。所幸字符集的大小是个常数，于是仍然可以常数级判断两个桶是否表示同一个离散模式。

所以维护前缀/后缀桶，每次匹配的时候就往桶里面加数。

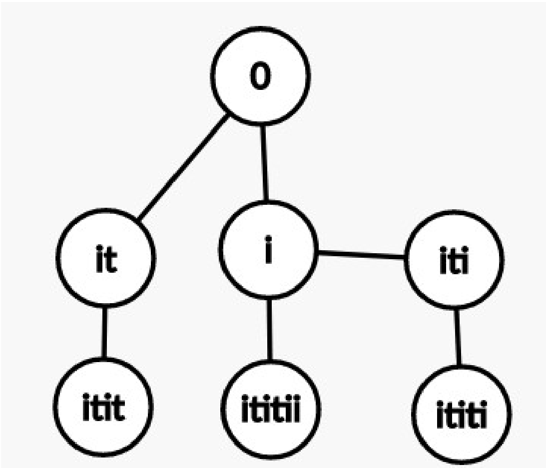
失配的时候怎么办？暴力退回更新前缀/后缀的桶？

没错，不过这不是暴力，因为均摊分析可以证明退回的总距离不超过 |T|（前缀函数总大于 0，只增加 |T| 次 1，所以也只能减少 |T| 次 1）。

失配树

我们发现，对于模式串的所有前缀，它都有唯一的 LCPS，也就是每次失配后的下一个去向。依此我们可以建出一棵树（为什么是一棵树？无环、连通、n-1 条边）

而且我们发现 CPS 关系是传递的，那么，任何一个前缀节点的祖先都是它的 CPS。



ititii

失配树：应用

【题目描述】

给出一个字符串，每次询问一个 p 前缀和 q 前缀，问它们的最长公共 border 的长度。

【题目解答】

很明显，这就是在问失配树上的 LCA。

动物园

【题目描述】

给出一个字符串，询问每个前缀长度小于等于该前缀长度一半的 border 的个数。

大家思考五分钟~

来源：NOI 2014 动物园

动物园

【题目解答】

不考虑限制，对一个前缀所有的 border 计数就是直接问失配树上点的深度。

考虑限制，发现一个前缀最浅的满足条件的点是容易递推维护的：对于每个前缀，维护它的最长的，长度不大于它一半的 border。每个点的答案就是这个 border 的深度。

于是往下递推，要么失配往回跳（失配只能往回跳，因为 border 更深的肯定更不匹配），要么增长，增长超过前缀长度一半后往回跳（由于失配也往回跳，不用考虑往前跳）。

根据均摊分析，复杂度 O(|S|)。

似乎在梦中见过的样子

【题目描述】

给定字符串 S，求该串的所有 ABA 型子串。其中的 A 长度不小于 k，B 不为空。

|S| <= 5000, k <= 100

大家思考五分钟~

似乎在梦中见过的样子

【题目解答】

这题也是 2014 年的题，想来和前面那题差不多（）

数据范围小，对于每个后缀暴力跑 KMP 就行。

ABA 的要求就是：一、不能重叠，那么 border 不能超过串长一半。二、border 要超过 k。

在失配树上维护两个祖先指针就行，同样可以递推达到均摊线性复杂度。

前缀自动机

我们发现 KMP 在计算 S # T 时，「匹配出的部分」其实就一直在 |S| 之内打转，而且 KMP 每次转移状态只看当前状态和后面新加的一个字符。

我们知道状态数只有 |S| 种，于是可以预先算出所有的状态在接受所有下一种字符的转移，这样可以加快匹配的速度。

可是这样还是线性的呀？没有加速。

可以在前缀自动机上 DP/递推/倍增，维护一些极端的匹配情况（下午考）。

前缀自动机：实现

神犇同学：还需要你教？直接枚举每个状态的 26 条出边，看是否匹配就行了。

你说的对，~~但是前缀自动机是一款开放世界……~~失配的过程可能爆复杂度。

比如 aaaaaaaa.... 你每次都要失配跳到最前头，O(|S|^2) 了。

考虑递推，如果已知要失配，那么直接查看失配到的那个点的转移就行了，不必重算。

前缀自动机：实现

下午要考。模板还是端上来罢（有慈悲）

```
void compute_automaton(string s, vector<vector<int>>& aut) {
  s += '#'; // 保证末尾状态失配
  int n = s.size();
  vector<int> pi = prefix_function(s);
  aut.assign(n, vector<int>(26));
  for (int i = 0; i < n; i++) {
    for (int c = 0; c < 26; c++) {
      if (i > 0 && 'a' + c != s[i]) // 失配，递推处理
        aut[i][c] = aut[pi[i - 1]][c];
      else // 匹配，直接跳转；0 就直接是 0
        aut[i][c] = i + ('a' + c == s[i]);
    }
  }
}
```

字典树*

「字符串状态搜索树」

字典树：引入

什么是字典树（Trie）？它的作用和字典很像：索引字符串。

如果让你搜索所有可能的字符串，你会怎么做？

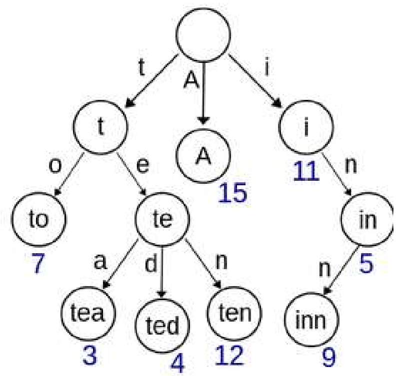
从空串状态开始，每次转移枚举地往末尾添加一个字符……

这就会形成一个搜索树，其中树上的每个状态节点都是一个字符串，字符串中从前往后的字符就是状态的转移序列。

但是我们有时只关心一个特定字符串集合。

那就是「已知状态集合，建出能搜出所有状态的搜索树（状态机）」。

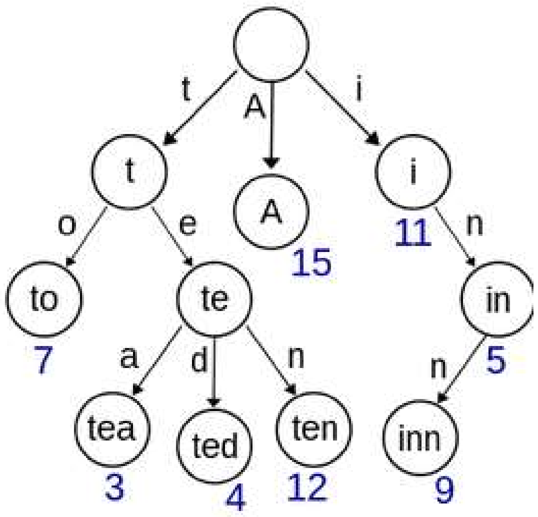
每次沿着状态转移路径走一遍，不够就加。



字典树：实现

每次沿着状态转移路径走一遍，不够就加。

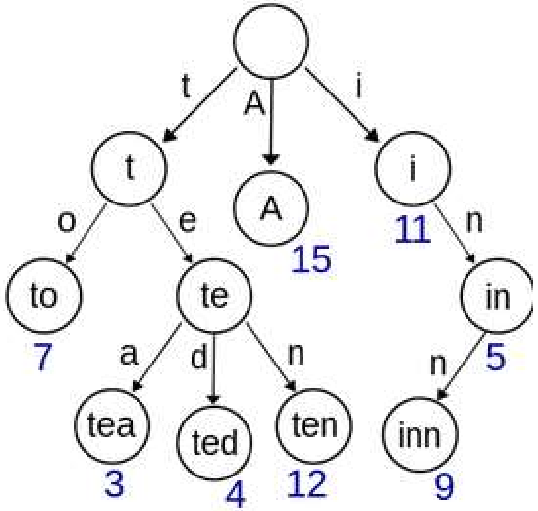
```
int nex[100000][26], cnt;
bool exist[100000];  // 该结点结尾的字符串是否存在

void insert(char *s, int l) {  // 插入字符串
  int p = 0;
  for (int i = 0; i < l; i++) {
    int c = s[i] - 'a';
    if (!nex[p][c]) nex[p][c] = ++cnt;  // 如果没有，就添加结点
    p = nex[p][c];
  }
  exist[p] = 1;
}
```



字典树：实现

每次沿着状态转移路径走一遍，不够就加。

```
bool find(char *s, int l) {  // 查找字符串
  int p = 0;
  for (int i = 0; i < l; i++) {
    int c = s[i] - 'a';
    if (!nex[p][c]) return 0;
    p = nex[p][c];
  }
  return exist[p];
}
```



字典树：复杂度

字典树的建树时空复杂度显然都是 O(|S|) 的，其中 |S| 为所有字符串的长度总和。

一次状态匹配（查找）的复杂度是 O(|S|) 的，其中 |S| 为单次字符串的长度。

这一页太空了，多讲两句。

字典树把字符串本身变成了一种状态，字符序列变成了状态转移序列。这样的状态机本来将有指数级大小，但字典树只保留了本字符串集合存在的状态，使得复杂度变为线性。

但是却保留下了状态机的好处：可以在线性时间内接受一个状态序列，并维护大量相似状态的信息（在字典树中，「相似性」就是串间的公共前缀）

可以看出，字典树和字符串哈希虽然都可以实现串查找，但是字典树的结构性更好。

字典树，DFS 序与树状数组

【题目描述】

一组字符串，每组字符串有权值，有两种操作：

一种是修改某个特定字符串的权值。

一种是查询所有以一个给定串为前缀的所有串的权值和。

【题目解答】

很明显所谓「以一个给定串为前缀的所有串」就是该给定串状态点的子树。

那么我们只需要把树建好，展平成 DFS 序利用树状数组求和即可。

（下午会考）

01-字典树（01-Trie）

01 序列/二进制数是不是字符串？（回去看定义）

既然是字符串，那么也可以建字典树！

对异或/与/或等按位操作，可以有效维护相关信息。

最长异或路径

【题目描述】

给你一棵带边权的树，求 (u, v) 使得 u 到 v 的路径上的边权异或和最大，输出这个最大值。

两分钟想想。

提示：异或具有消去律。即异或上两个相同的数等于异或上 0。

最长异或路径

【题目解答】

很明显，每条路径的价值等于选择两条根节点出发的链异或起来。

那么把每个点到根的路径上的异或和计算出来，那么我们只需要计算其中异或和最大的一对数就好了，因为相同的数互相异或是 0，问题又可以转化为对每个数寻找和它异或结果最大的一个数。

建 0/1 字典树，将每个数从高位到低位插进去，然后对于查询的数，从高位起贪心地尝试和该位不同的状态转移路径（即前缀/高位优先匹配）。

AC 自动机初步*

Trie 都是选讲，再讲 AC 自动机是不是有点……~~可是知识点太少了讲不满三小时啊~~

考虑多个模式串的匹配问题。

可以把多个模式串建成 Trie，文本串也直接上这个 Trie 进行转移，匹配到了就记录答案。

失配了怎么办？暴力回退？太慢了！还是考虑找匹配过的一个最大公共前/后缀。

等等，这里的前缀是什么意思？

Trie 树能接受的状态都可以被称作这里的一个前缀。

还是把失配指针指向 LCPS，这样又可以得到一个失配树（由于有多个模式串，匹配的时候需要检查失配树），这里的失配树仍然通过类似前缀函数的方法进行递推。

后记&杂谈

要下班咯

字符串的世界还很广大！

SA/SAM，BM算法，Z函数（exKMP），回文树，回文自动机，后缀平衡树……不一而足。

讲解完全部这些是时间不允许的，亦非本人能力之所能及，需要大家课下自行好好锻炼。

字符串算法很实用，也很有意思，因为常常涉及自动机和树形结构可以很好的结合多种算法（如 DP/数据结构）锻炼思维能力。

漫谈 OI/ICPC/升学 & 经验分享*

今天知识点太少了，如果时间有剩，就和大家闲聊会吧，介绍点我自己的竞赛经验。

关于下午的题……

听说有人和昨天的教练抱怨我出题很水（？）~~呜呜呜被 D 了，那这次还是难一点吧~~

三道题（知识点比较少，我也出不动了），不过我还是尽量提早开始。

课件/题面将放在第一题的附加文件里。

题目部分分很多，230 out of 300 都是送给大家的（背好模板！）。

重点：
字典树（150）、KMP / 字符串哈希（80）、失配树（40）、前缀自动机（20）、树状数组（10）（括号内为涉及的分值数）。

重在练习，不要登别人的号，不要抄袭，不要浪费自己的时间/精力做无意义的事。

最后一天了~~（My last day, not yours）~~，大家有缘再见~~（下午还要再见）~~。

感谢垂听

祝大家用餐愉快