提高算法班

Hash、MP、KMP

Mas

哈希表

哈希表是又称散列表，是一种以 key – value 形式存储数据的数据结构

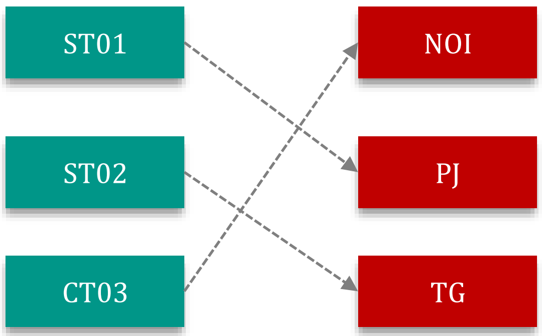
key – value 形式存储数据，是指任意的键值 key 都唯一对应内存中的某个位置

只需根据 key 就可快速找到其对应的 value

根据键值计算索引的函数叫做哈希函数,也称散列函数

一般把键值模一个较大的**质数**作为索引

$$f(\text{key}) = \text{key} \mod p$$



实际应用中， key 可能是更复杂的类型如浮点数、字符串、结构体等

此时需要根据具体情况设计合适的哈希函数

哈希函数应当 **易于计算**，并且尽量使计算出来的索引 **均匀分布**

哈希冲突

对于任意的键值，若哈希函数计算的索引都不同，只需根据索引将 (key, value) 放到对应位置

常会出现多个不同键值其哈希函数计算的索引相同，此时需要解决冲突

开散列法(拉链法)

在各存放数据位置建立链表，若多个键值索引到同一位置，将其放到对应位置的链表中

查询时需要把对应位置的链表扫描，对其中每个数据比较其键值与查询的键值是否一致

闭散列法

闭散列方法把所有记录直接存储在散列表中，若发生冲突则根据某种方式继续进行探查

线性探查法

若在 d 处发生冲突，依次检查 $d \pm 1, d \pm 2, \cdots$

二次探查法

若在 d 处发生冲突，依次检查 $d \pm 1^2, d \pm 2^2, \cdots$

在程序设计竞赛中常用拉链法实现 哈希表

哈希冲突

哈希函数，期望 $0 \le f(\mathrm{key}) \le p-1$ 且分布均匀

以 $f(\mathrm{key}) = \mathrm{key} \bmod p$ 以带余数除法的形式展开

$$\mathrm{key} = \left\lfloor \frac{\mathrm{key}}{p} \right\rfloor p + f(\mathrm{key}) \Rightarrow f(\mathrm{key}) = \mathrm{key} - \left\lfloor \frac{\mathrm{key}}{p} \right\rfloor p$$

令 $p = kx$，$\mathrm{key} = ky$

$$f(\mathrm{key}) = k \times \frac{ky - \left\lfloor \frac{\mathrm{key}}{p} \right\rfloor kx}{k}$$

即 $f(\mathrm{key})$ 必为公因子 k 的倍数

若模数 p 不为质数，那么 k 易取到 > 1 的情况

此时余数分布不均匀，冲突机率比 p 为质数时大

#2777、还是简单哈希

题目描述

给定一系列整型关键字和正整数 P ,用除留余数法定义的散列函数

$$H(\text{Key}) = \text{Key} \bmod P$$

将关键字映射到长度为 P 的散列表中

用链地址法(尾插法插入链表)解决冲突

输入格式

输入第一行首先给出两个正整数 N 和 P ,分别为待插入的关键字总数、以及散列表的长度

第二行给出 N 个正整数关键字不超过 10^9 ,数字间以空格分隔

输出格式

输出 N 行

每行输出在散列表中的索引以及在链表中的索引(如果存在相同的 `key` 仅保留一个)

输入样例1

```
4 5
24 15 61 88
```

输出样例1

```
4 1
0 1
1 1
3 1
```

数据规模

对于全部的数据 $1 \le N \le 1000, P \le 2000$

字符串哈希

字符串映射到整数的函数 f , 称 f 是 hash 函数

一般希望函数 f 可方便判断两个字符串是否相等

- 在 hash 函数值不一样的时候,两个字符串一定不一样
- 在 hash 函数值一样的时候,两个字符串不一定一样 (但有大概率一样,且我们当然希望它们总是一样)

通常采用多项式 hash 的方法

对于字符串 S 定义多项式 hash 函数

$$f(S) = \sum_{i=1}^{|S|} S_i \times p^{|S|-i} \pmod{M}$$

如对于字符串 "xyz" , 其哈希函数值为 $xp^2 + yp + z$ (可理解为 p 进制数)

字符串哈希

一般 p 取不小于 S[i] 的质数（如 ASCII 字符可取 131）

若 M 取 1000000007，有 $n = 1000000$ 次比较

每次的错误率为 $\frac{1}{\mathrm{M}}$，总错误率为

$$1 - \left(1 - \frac{1}{\mathrm{M}}\right)^{n} \approx \frac{1}{1000}$$

不少做法令 M 取 2^{64}

即直接使用 unsigned long long 类型储存 hash 值，产生溢出时相当于对 2^{64} 取模

此时冲突概率约为 0.00000000000005（**但不难构造出哈希冲突的的案例，如 Thue – Morse Sequence**）

建议取两个大质数进行分别取模

#2779、兔子DNA

题目描述

很久很久以前,森林里住着一群兔子

有一天,兔子们想要研究自己的 DNA 序列

我们首先选取一个好长好长的 DNA 序列(小兔子是外星生物, DNA 序列可能包含 26 个小写英文字母)

然后我们每次选择两个区间,询问如果用两个区间里的 DNA 序列分别生产出来两只兔子,这两个兔子是否一模一样

注意两个兔子一模一样只可能是他们的 DNA 序列一模一样

输入格式

第一行输入一个 DNA 字符串 S

第二行一个数字 m ,表示 m 次询问

接下来 m 行,每行四个数字 l_1, r_1, l_2, r_2 ,分别表示此次询问的两个区间

注意字符串的位置从 1 开始编号

输出格式

对于每次询问,输出一行表示结果

如果两只兔子完全相同输出 `Yes` ,否则输出 `No`

数据范围

对于全部的数据 $1 \leq |S|, m \leq 1000000$

计算单个字符串 S 哈希值复杂度为 $O(|S|)$

若要多次询问一个字符串的子串的哈希值

每次重新计算效率非常低下

#2779、兔子DNA

记 $f_i(S)$ 表示 $f(S_{1\sim i})$ 即原串长度为 i 的前缀的哈希值

按照定义有

$$f_r(S) = \mathbf{S_1\,p^{r-1} + S_2\,p^{r-2} + \cdots + S_{l-1}\,p^{r-l+1}} + \mathbf{S_l\,p^{r-l} + \cdots + S_{r-1}\,p + S_r}$$

那么

$$f(S_{l\sim r}) = S_l\,p^{r-l} + S_{l+1}\,p^{r-l-1} + \cdots + S_{r-1}p + S_r$$

$$f_{l-1}(S) = S_1\,p^{l-2} + S_2\,p^{l-1} + \cdots + S_{l-1}\,p + S_{l-1}$$

对比上述两式发现

$$f(S_{l\sim r}) = f_r(S) - f_{l-1}(S) \times p^{r-l+1}$$

可以 $O(n)$ 的代价预处理 p^i 与 $f_i(S)$

单次询问子串哈希值时间复杂度 $O(1)$

```cpp
int f[2][MAXN], p[2][MAXN];
int MOD[2] = {998244353, 1000000007}, b[2] = {131, 137};
char str[MAXN];
void init() // 双模hash
{
  p[0][0] = p[1][0] = 1;
  for (int i = 0; i < 2; i++)
    for (int j = 1; str[j]; j++)
    {
      f[i][j] = (1LL * f[i][j - 1] * b[i] + str[j]) % MOD[i];
      p[i][j] = 1LL * p[i][j - 1] * b[i] % MOD[i];
    }
}
pair<int, int> get(int l, int r) // 子串hash
{
  int h1 = (f[0][r] - 1LL * f[0][l - 1] * p[0][r - l + 1]) % MOD[0];
  int h2 = (f[1][r] - 1LL * f[1][l - 1] * p[1][r - l + 1]) % MOD[1];
  return {(h1 + MOD[0]) % MOD[0], (h2 + MOD[1]) % MOD[1]};
}
```

#2780、01矩阵

题目描述

给定一个 $n \times m$ 列的 01 矩阵(只包含数字 0 或 1 的矩阵)

再执行 Q 次询问

每次询问给出一个 A 行 B 列的 01 矩阵

求该矩阵是否在原矩阵中出现过

输入格式

第一行四个整数 n, m, A, B

接下来一个 n 行 m 列的 01 矩阵,数字之间没有空格

接下来一个整数 Q

接下来 Q 个 A 行 B 列的 01 矩阵

输出格式

对于每个询问

输出 1 表示出现过, 0 表示没有出现过

数据范围

对于全部的数据 $A \le 100, M, N, B \le 1000, Q \le 1000$

将矩阵各行看作字符串进行 hash

令 $f(i,j)$ 表示 第 i 行前 j 列的 hash 值

$$f(i,j) = \sum_{k=1}^{j}\left(A_{ik} \times b_1^{j-k}\right)$$

令 $F(i,j)$ 表示 前 i 行前 j 列的 hash 值

$$F(i,j) = \sum_{k=1}^{i}\left(f(k,j) \times b_2^{i-k}\right)$$

#2780、01矩阵

考虑求出所有 $a \times b$ 子矩阵 hash 值

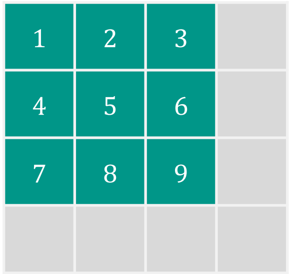
不难得出

$$F(i,j) - F(i-a,j) \times b_2^a - F(i,j-b) \times b_1^b + F(i-a,j-b) \times b_1^b \times b_2^a$$

枚举可能的 i,j 即可

对于询问的矩阵，同样计算其 hash 值，查找是否出现即可

时间复杂度 $O(nm + qab)$



也可将前缀矩阵按照右图形式编号，当作字符串进行 hash

同样预处理出 $f(i,j)$

逐列考虑求出所有 $a \times b$ 子矩阵 hash 值

字符串

实验舱 青少年编程 走近科学 走进名校

字符集

一个字符集 Σ 是一个建立了全序关系的集合，即 Σ 中任意两个不同的元素 α 和 β 都比较大小，要么 $\alpha < \beta$，要么 $\beta < \alpha$

字符集 Σ 中的元素称为字符

字符串

一个字符串 S 是将 n 个字符顺次排列形成的序列, n 称为 S 的长度表示为 $|S|$

若下标从 0 开始， S 的第 i 个字符表示为 $S[i-1]$

子串

字符串 S 的子串 $S[i..j]$，$i \le j$ 表示 S 串中从 $i \sim j$ 连续一段，即顺次排列 $S[i]$， $S[i+1]$， … ， $S[j]$ 形成的字符串

有时也会用 $S[i..j]$， $i > j$ 来表示空串

子序列

字符串 S 的子序列是从 S 中提取若干元素并不改变相对位置形成的序列

即 $S[p_1]$， $S[p_2]$， … ， $S[p_k]$， $0 \le p_1 < p_2 < \cdots < p_k < |S|$

字符串

后缀

指从某个位置 i 开始到串末尾的一个特殊子串

字符串 S 的从 i 开头的后缀表示为 $\mathrm{Suffix}(S,i)$ 即 $S[i..|S|-1]$

真后缀指除了 S 本身的 S 的后缀

前缀

指从串首开始到某个位置 i 的一个特殊子串

字符串 S 的以 i 结尾的前缀表示为 $\mathrm{Prefix}(S,i)$ 即 $S[0..i]$

真前缀指除了 S 本身的 S 的前缀

字典序

以第 i 个字符作为第 i 关键字进行大小比较，空字符小于字符集内任何字符

回文串

满足 $\forall\, 0 \le i < |S|,\ S[i] = S[\,|S|-1-i\,]$ 的字符串 S

字符串匹配

字符串匹配又称模式匹配 (pattern matching)

该问题可概括为给定字符串 S 和 T , 在 S 中寻找子串 T 字符

其中 S 称为 **主串** ， T 称为 **模式串**

单串匹配

给定一个模式串和一个待匹配串,找出前者在后者中的所有位置

多串匹配

给定多个模式串和一个待匹配串,找出这些模式串在后者中的所有位置

其他类型

匹配一个串的任意后缀

匹配多个串的任意后缀

……

字符串匹配

暴力做法

从主串 S 的第一个字符开始和模式串 T 的首个字符进行比较

- 若相等，则继续比较二者的后续字符
- 否则，模式串 T 回退到第一个字符，重新和主串 S 的下一字符进行比较

直到 S 或 T 中所有字符比较完毕

在最坏情况下，每趟不成功的匹配都发生在模式串的最后一个字符

时间复杂度为 $O(|S||T|)$

哈希

求出模式串的哈希值，求出主串各长度为模式串长度的子串哈希值

分别与模式串的哈希值比较，时间复杂度 $O(|S| + |T|)$

可能产生 hash 冲突

前缀函数

给定一个长度为 n 的字符串 S

其前缀函数被定义为一个长度为 |S| 的数组 π

- 子串 $S[0 \dots i]$ 仅有一对相等真前缀与真后缀 $S[0 \dots k-1]$ 和 $S[i-(k-1) \dots i]$

  那么 $\pi[i]$ 为相等 **真**前/后缀长度，即 $\pi[i] = k$

- 不止有一对相等的

  那么 $\pi[i]$ 为其中最长那对长度

- 没有相等的

  那么 $\pi[i] = 0$

	0	1	2	3	4	5	6
S	A	B	C	A	B	C	D
π	0	0	0	1	2	3	0

即

$$\pi[i] = \max_{0 \le k \le i}\{\, k : S[0 \dots k-1] = S[i-(k-1) \dots i] \,\}$$

规定 $\pi[0] = 0$

前缀函数

朴素求解 $\pi[i]$

在一个循环中以 $i = 1 \to n - 1$ 的顺序计算 $\pi[i]$

令变量 j 从最大的真前缀长度 i 开始尝试

若当前长度下真前缀和真后缀相等，则此时长度为 $\pi[i]$

否则令 j 自减 1，继续匹配直到 $j = 0$

若 $j = 0$ 并且仍没有任何一次匹配，令 $\pi[i] = 0$

时间复杂度 $O(|S|^3)$

$$\overbrace{\underbrace{S_0S_1S_2}_{\pi[i]=3}\,S_3}^{\pi[i+1]=4}\cdots\overbrace{\underbrace{S_{i-2}S_{i-1}S_i}_{\pi[i]=3}\,S_{i+1}}^{\pi[i+1]=4}$$

不难发现相邻前缀函数值至多增加 1

当取一个尽可能大的 $\pi[i+1]$ 时，必然要求新增的 $S[i+1]$ 也与之对应的字符匹配

前缀函数

即 $S[i+1] = S[\pi[i]]$, 此时 $\pi[i+1] = \pi[i] + 1$

当 i 增大时前缀函数的值要么增加1 、要么维持不变 , 要么减少

最好情况为首次比较就完成了匹配 , 那么首次次比较就成功至多为 $|S|$

由于限制了最大前缀的上界 , 当进行超过一次比较时 $\pi[i]$ 不增大

即每多进行一次回退时的比较就消耗了后续 $\pi[i]$ 的增长空间

总回退至多发生 |S| 次

那么至多共进行 2|S| 次比较 , 单次匹配比较时间复杂度 $O(|S|)$

总时间复杂度 $O(|S|^2)$

```
vector<int> prefix_function2(string s)
{
  vector<int> pi(s.size());
  for (int i = 1; i < s.size(); i++)
    for (int j = pi[i - 1] + 1; j >= 0; j--)
      if (s.substr(0, j) == s.substr(i - j + 1, j))
      {
        pi[i] = j;
        break;
      }
  return pi;
}
```

前缀函数

在上个优化中，讨论了计算 $\pi[i+1]$ 的最好情况

$S[i+1] = S[\pi[i]]$ 时 $\pi[i+1] = \pi[i] + 1$

当 $S[i+1] \neq S[\pi[i]]$ 时

若能找到子串 $S[0 \dots i]$ 仅次于 $\pi[i]$ 的第二长度 j，使得 i 的前缀性质仍能保持

即 $S[0 \dots j-1] = S[i-j+1 \dots i]$

仅需再比较 $S[i+1]$ 和 $S[j]$

若它们相等有 $\pi[i+1] = j+1$

否则需要找到子串 $S[0 \dots i]$ 仅次于 j 的第二长度 j_2，使得前缀性质得以保持

不断重复直到 $j = 0$

若 $S[i+1] \neq S[0]$ 则 $\pi[i+1] = 0$

前缀函数

$$\overbrace{S_0\ S_1 \cdots S_j}^{j} \cdots \underbrace{\overbrace{S_{i-j+1}\ \cdots S_{i-1}\ S_i}^{j}}_{\pi[i]}\ S_{i+1}$$

观察上图可以发现

由于 $S[0 \dots \pi[i]-1] = S[i-\pi[i]+1 \dots i]$

对于 $S[0 \dots i]$ 的第二长度 j 存在如下性质

$$S[0\ \dots j\ -\ 1] = S[i\ -\ j\ +\ 1\ \dots i] = S[\,\pi[i]-j \dots \pi[i]-1\,]$$

j 等价于子串 $S[0 \dots \pi[i]-1]$的前缀函数值，即 $j = \pi[\,\pi[i]-1\,]$

同理次于 j 的第二长度等价于 $S[0 \dots j-1]$ 的前缀函数值 $j_2 = \pi[j-1]$

不难得出关于 j 的转移方程

$$j_n = \pi[j_{n-1}-1]\,, \qquad (j_{n-1} > 0)$$

前缀函数

	0	1	2	3	4	5	6	7	8	9
S	A	B	A	C	A	B	A	B	A	B
π	0	0	1	0	1	2	3	2	\	\

	0	1	2	3	4	5	6	7	8	9
S	A	B	A	C	A	B	A	B	A	B
π	0	0	1	0	1	2	3	2	\	\

j

```
vector<int> prefix_function(string s)
{
    vector<int> pi(s.size());
    for (int i = 1, j = 0; i < s.size(); i++)
    {
        while (j > 0 && s[i] != s[j])
            j = pi[j - 1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

借助上述性质，可得出无需字符串比较的算法

类似的 j 减少的次数取决于前缀数组取值，且前缀数组始终非负

时间复杂度 $O(|S|)$

#640、Seek the Name, Seek the Fame

题目描述

给定若干字符串(这些字符串总长 $\leq 4 \times 10^5$)

在每个字符串中求出所有既是前缀又是后缀的子串长度

例如: $\mathrm{ababcababababcabab}$ 既是前缀又是后缀的:

$$\mathrm{ab, abab, ababcabab, ababcababababcabab}$$

输入格式

输入若干行,每行一个字符串

输出格式

对于每个字符串,输出一行

包含若干个递增的整数,表示所有既是前缀又是后缀的子串长度

样例输入

```
ababcababababcabab
aaaaa
```

样例输出

```
2 4 9 18
1 2 3 4 5
```

字符串本身长度 $|S|$ 必然为一个答案

根据前缀函数定义

$\pi[\,|S|-1\,]$ 为 S 最长真公共前后缀长度

$\pi\big[\,\pi[\,|S|-1\,]\big]$为次二长

以此类推

将结果逆序输出即可

时间复杂度 $O(|S|)$

MP Algorithm

abcabcabd
abcabd

abcabcabd
abcabd

朴素算法中出现失配

目标串指针回溯,模式串指针归零

abcabd

模式串中对于失配字符 d

它前面所有字符已经匹配

存在真前缀 ab 和 真后缀 ab 为公共部分

abcabcabd
abcabd

目标串指针可保持不移动

仅将模式串指针左移

回退到公共部分的下一位置

MP Algorithm

MP 算法(Morris – Pratt Algorithm) 是一种快速串匹配算法

由詹姆斯·莫里斯(James Morris)、沃恩·普莱特(Vaughan Pratt)在 1970 年提出

过程如下

求出模式串 T 前缀函数 $\pi[i]$

当模式串 T 的 $T[j]$ 与 $S[i]$ 发生失配时，此时 $\mathrm{S}[i-j+1 \dots i-1] = \mathrm{T}[0 \dots j-1]$

i 并不回溯

j 尽可能保留最右的位置，使得 $\mathrm{S}[i-j'+1 \dots i-1] = \mathrm{T}[0 \dots j']$ 依然成立

即令 $j' = \pi[j-1]$

由于 $\pi[i]$ 为最长真公共前后缀，i 不回溯并不会产生遗漏

类似的 j 每次至多增加 1 而 $\pi[i]$ 始终非负，那么 j 变化次数不超过 $2|\mathrm{T}|$

时间复杂度 $O(|\mathrm{S}| + |\mathrm{T}|)$

```cpp
int mathch(string s, string t)
{
    int len1 = s.size(), len2 = t.size();
    for (int i = 0, j = 0; i < len1; i++)
    {
        while (j && (s[i] != t[j]))
            j = pi[j - 1];
        if (s[i] == t[j])
            j++;
        if (j == len2)
            ;//一次匹配成功
    }
    return -1;
}
```

KMP Algorithm

KMP 算法(Knuth – Morris – Pratt Algorithm)

由 唐纳德·克努特(Donald Knuth)、詹姆斯·莫里斯(James Morris) 和 沃恩·普莱特(Vaughan Pratt) 在 1977 年共同发布

KMP 算法是对 MP 算法复杂度常数上的改进

ababab**d**
ababab**c**

ababab**d**
ababa**a**bc

ababab**d**
ab**a**babc

上图案例需三次才能发现匹配失败，而回退毫无意义

考虑更早发现无意义的回退

KMP Algorithm

	0	1	2	3	4	5	6
S	a	b	a	b	a	b	c
π	0	0	1	2	3	4	0
nxt	0	0	0	0	0	4	0

```cpp
void getNxt(string t)
{
    nxt[0] = 0;
    for (int i = 1, j = 0; t[i]; i++)
    {
        while (j && t[i] != t[j])
            j = nxt[j - 1];
        if (t[i] == t[j])
            if (t[i + 1] == t[j + 1])
                nxt[i] = nxt[j++];
            else
                nxt[i] = ++j;
        else
            nxt[i] = j;
    }
}
```

若 S[i] 与 T[j] 失配，j 将利用 $\pi[j-1]$ 进行回退

在计算 $\pi[x]$ 时，若利用了$\pi[x]$ 说明在模式串第 $x+1$ 位出现失配

若此时 $T[x+1]$ 与 $T[y+1]$ 相同，回退到 $y+1$ 位依然失配

不妨令 $\pi[x]$ 直接等于 $\pi[y]$

无论是 MP 算法还是 KMP 算法，其总时间复杂度都为 $O(|S|+|T|)$

字符串匹配

前缀函数求解过程，事实上就是模式串自匹配

	0	1	2	3	4	5	6	
S	a	b	a	b	a			
		a	b	a	b	a		
			a	b	a	b	a	
			a	b	a	b	a	a
			a	b	a	b	a	a
π	0	0	1	2	3			

另一个视角理解匹配

构造字符串 T + # + S，其中 # 为 主串 S 与模式串中一定不出现的字符

求出字符串 T + # + S 的前缀函数 $\pi[i]$

由于 # 的存在 $\pi[\,0 \sim |T|\,]$ 的值必然小于 $|T|$

若在 $\pi[\,|T|+1 \sim |S|+|T|\,]$ 中 发现 $\pi[i] = |T|$ 说明 T 在 S 中出现

	0	1	2	3	4	5	6	7	8	9	10
S'	a	b	a	#	a	b	a	c	a	b	a
π	0	0	1	0	1	2	3	0	1	2	3

一个人能走多远不在于他在顺境时能走多快，而在于他在逆境时多久能找到曾经的自己

——KMP

#2788、字符串匹配

题目描述

给出两个字符串 S_1 和 S_2

若 S_1 的区间 $[l,r]$ 子串与 S_2 完全相同,则称 S_2 在 S_1 中出现了,其出现位置为 l

现在请你求出 S_2 在 S_1 中所有出现的位置

定义一个字符串 S 的 border 为 S 的一个非 S 本身的子串 t,满足 t 既是 S 的前缀,又是 S 的后缀

对于 S_2,你还需要求出对于其每个前缀 S' 的最长 border t' 的长度

输入格式

第一行为一个字符串,即为 S_1

第二行为一个字符串,即为 S_2

输出格式

首先输出若干行,每行一个整数,**按从小到大的顺序**输出 S_2 在 S_1 中出现的位置

最后一行输出 $|S_2|$ 个整数,第 i 个整数表示 S_2 的长度为 i 的前缀的最长 border 长度

数据规模

对于全部的测试点,保证 $1 \le |S_1|, |S_2| \le 10^6, S_1, S_2$ 中均只含大写英文字母

```cpp
void mathch(string s, string t)
{
  int len1 = s.size(), len2 = t.size();
  for (int i = 0, j = 0; i < len1; i++)
  {
    while (j && (s[i] != t[j]))
      j = nxt[j - 1];
    if (s[i] == t[j])
      j++;
    if (j == len2)
    {
      printf("%d\n", i - len2 + 2);
      j = nxt[j - 1];
    }
  }
}
```

当一次匹配成功后 i 需要递增

将 j 视作一次失配

令 $j = \pi[j-1]$ 即可

#649、Censoring

题目描述

给出两个字符串 S 和 T

每次从前往后找到 S 的一个子串 $A = T$ 并将其删除

空缺位依次向前补齐,重复上述操作多次,直到 S 串中不含 T 串

输出最终的 S 串

输入格式

第一行包含一个字符串 S

第二行包含一个字符串 T

输出格式

输出处理后的 S 串

数据范围

对于全部数据 $1 \leq |T| \leq |S| \leq 10^6$

保证字符串中只出现小写字母

样例输入

```
whatthemomooofun
moo
```

样例输出

```
whatthefun
```

#649、Censoring

令 j 为已匹配字符数量

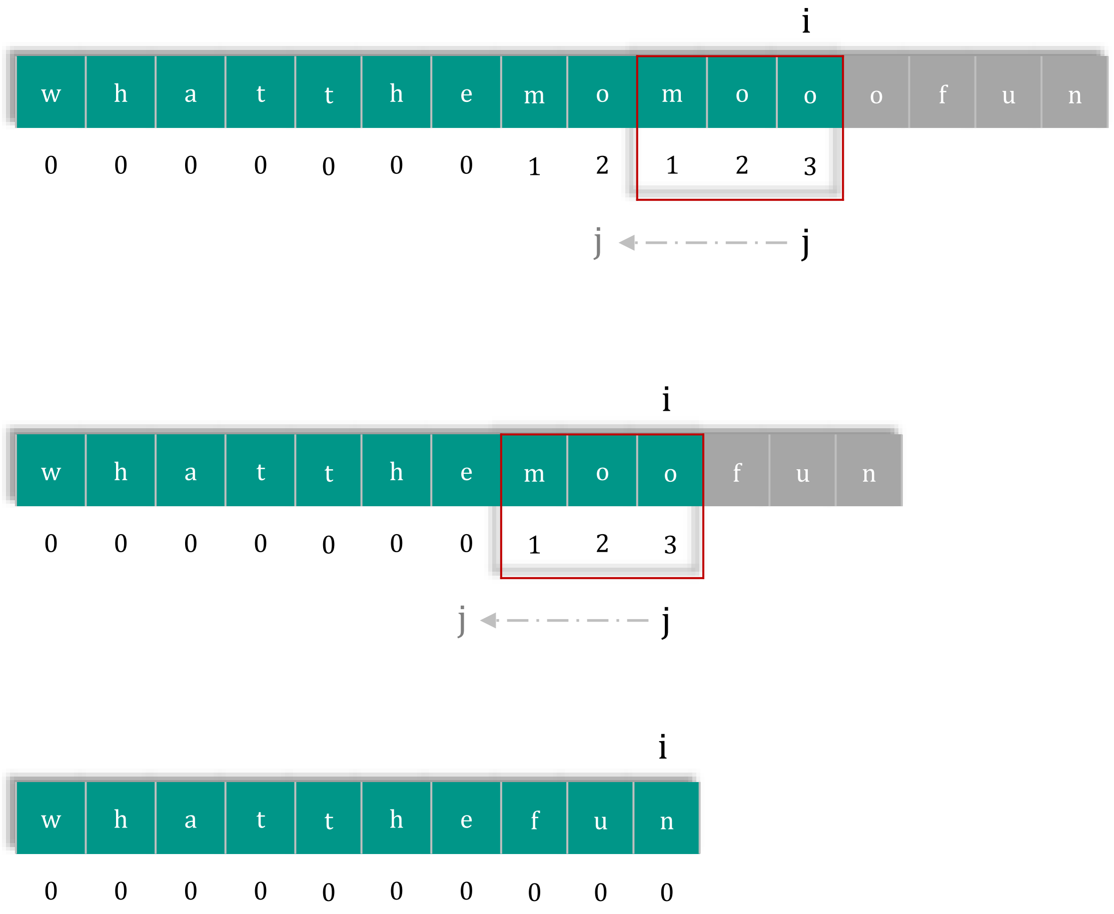
使用栈记录每个字符以及当前位置匹配字符数量

上述信息不难在匹配过程中维护

对于一次成功匹配，弹出栈中 $|T|$ 个字符

j 修改为当前栈顶记录的匹配字符数量

时间复杂度 $O(|S| + |T|)$



#639、Power Strings

题目描述

给定若干个长度 $\leq 10^6$ 的字符串

询问每个字符串最多是由多少个相同的子字符串重复连接而成的

如：`ababab` 则最多有 3 个 `ab` 连接而成

输入格式

输入若干行,每行有一个字符串

特别的,字符串可能为 `.` 即一个半角句号,此时输入结束

样例输入

```
abcd
aaaa
ababab
.
```

样例输出

```
1
4
3
```

对于字符串 S 和 $0 < p \leq |S|$

若 $S[i] = S[i + p]$ 对 $i \in [0, |S| - p - 1]$ 成立

那么称 p 为 S 的周期 (period)

用周期不断复制可以得到原串 S (最后一次复制可仅复制一部分)

显然 S 为 S 的一个周期

若周期 $p \mid |S|$ 那么该周期称为循环节

显然 S 也为 S 的一个循环节

本题要求最多重复次数

若能求出最小循环节 p 那么 $\frac{|S|}{p}$ 即为答案

#639、Power Strings

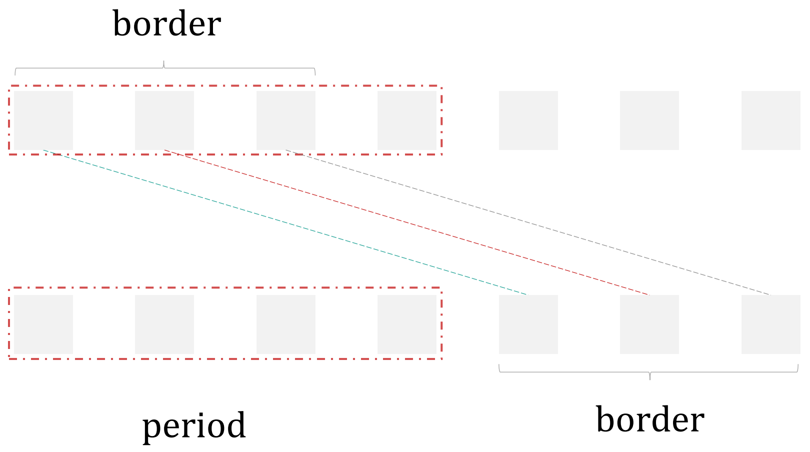
对于字符串 S 和 $0 \le r < |S|$

若 S 长度为 r 的前缀与后缀相等，那么称 r 为 S 的 border

不难发现 $\pi[|S|-1]$ 为 S 的一个 border

对于字符串 S 若其存在 border 长度为 r，那么 $|S|-r$ 是 S 的周期

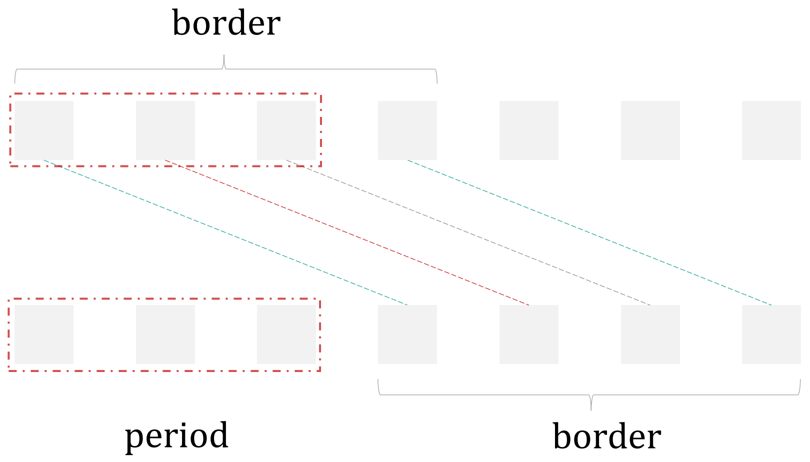
当 border 不存在交集时



period 显然可完整覆盖 S

#639、Power Strings

当 border 存在交集时



重合部分为原串相同位置，根据 border 性质前后缀相同

逐字符讨论不难得出上述结论

形式化的说

border 与 period 存在双射关系

#639、Power Strings

实验舱 青少年编程 走近科学 走进名校

period → border

任取周期 p , 根据定义 $\forall\, 0 \le i < |S| - p$ 都有 $S[i] = S[i+p]$

那么 $\forall\, p \le i < |S|$ 都有 $S[i] = S[i-p]$

即 $\mathrm{Prefix}(S, |S|-p) = \mathrm{Suffix}(S, |S|-p)$

border → period

任取 border b , 根据定义 $\mathrm{Prefix}(S, |b|) = \mathrm{Suffix}(S, |b|)$

即 $\forall\, 0 \le i < |b| = |S| - (|S| - |b|)$ 都有 $S[i] = S[\, i + |S| - |b|\,]$

不难发现 $|S| - |b|$ 即为周期

容易发现 $\pi[|S|-1]$ 即为 S 最大 border , 那么 $|S| - \pi[|S|-1]$ 即为最小周期

弱周期引理

弱周期引理 (Weak Periodicity Lemma)

若 p, q 都为 S 的周期 且 $p + q \leq |S|$ 那么 $\gcd(p, q)$ 也为 S 的周期

证明

不妨设 $p < q$, 记 $d = q - p$

由于 $p + q \leq |S|$ 若 $0 \leq i < |S| - d$ 那么 $i + q < |S|$ 或 $i - p \geq 0$ 必然满足其中一个

若 $i + q < |S|$

那么 $S[i] = S[i + q] = S[i + q - p] = S[i + d]$

若 $i - p < |S|$

那么 $S[i] = S[i - p] = S[i - p + q] = S[i + d]$

即 d 也为 S 的周期

不难发现这能形成一个辗转相减的过程 , 不难得到 $\gcd(p, q)$ 为 S 的周期

弱周期引理

不难得出推论

字符串 S 的最小周期为 p , 若周期 $q \le \left\lfloor \frac{|S|}{2} \right\rfloor$ 都有 $p \mid q$

证明

根据弱周期引理 $\gcd(p, q)$ 也为周期

根据 gcd 性质有 $\gcd(p, q) \le p$, 由于 p 为最小周期也有 $\gcd(p, q) \ge p$

那么 $\gcd(p, q) = p$ 即 $p \mid q$

得证

由于 $p \mid q$ 若 $p \nmid |S|$, 若周期 $q \le \left\lfloor \frac{|S|}{2} \right\rfloor$ 也有 $q \nmid |S|$

记 $q = kp$ 若 $q \mid |S| \Rightarrow kp \mid |S| \Rightarrow p \mid |S|$ 矛盾

这意味着最小周期不为循环节 , 那么最小循环节即为串本身

border 结构

对于字符串 S , 记 Border(S) 为 S 的 border 集合

border 的 border 还是 border

若 $|\mathrm{Border(S)}| > 1$ 且有 $v \in \mathrm{Border}(u) \land u \in \mathrm{Border(S)}$ 那么 $v \in \mathrm{Border(S)}$

设 $u \in \mathrm{Border(S)}$ ， $v \in \mathrm{Border}(u)$ 显然 $|v| < |u|$

根据定义有 $v = \mathrm{Prefix}(u, |v|) = \mathrm{Prefix}(S, |v|)$, 同时也有 $v = \mathrm{Suffix}(u, |v|) = \mathrm{Suffix}(S, |v|)$

得证

u 为 S 的最大 border , 那么 $\mathbf{Border(S)} = \mathbf{Border}(u) \cup u$

若存在 $v \in \mathrm{Border(S)}$ 且 $v \neq u \land v \notin \mathrm{Border}(u)$, 根据 u 的性质 $|v| < u$

根据定义有 $v = \mathrm{Prefix}(S, |v|) = \mathrm{Prefix}(u, |v|)$, 同时也有 $v = \mathrm{Suffix}(S, |v|) = \mathrm{Suffix}(u, |v|)$

即 $v \in \mathrm{Border}(u)$, 与假设矛盾 , 原命题得证

该性质为求解 **前缀函数** 的关键依据

#646、Radio Transmission

题目描述

给你一个字符串,它是由某个字符串不断自我连接形成的

但是这个字符串是不确定的,现在只想知道它的最短长度是多少

输入格式

第一行给出字符串的长度 L

第二行给出一个字符串,全由小写字母组成

输出格式

输出最短的长度

样例输入

```
8
cabcabca
```

样例输出

```
3
```

答案为 $|L| - \pi[\,|L| - 1\,]$

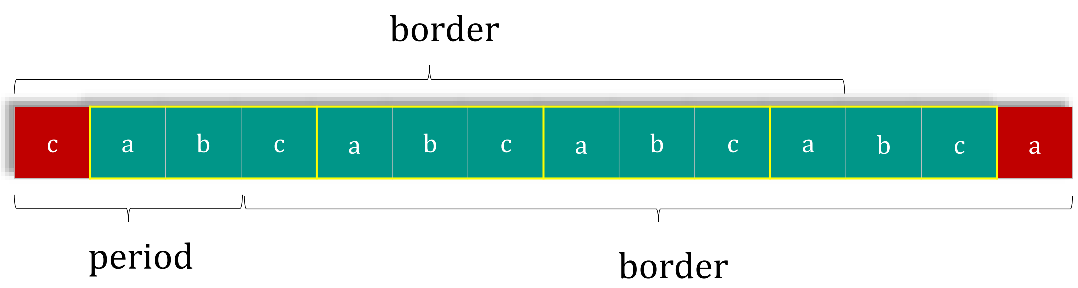
样例说明

对于样例,我们可以利用 abc 不断自我连接得到 $\mathrm{abcabcabc}$,读入的 $\mathrm{cabcabca}$ 是它的子串

数据范围

对于全部数据, $1 \leq L \leq 10^6$

#646、Radio Transmission



以字符串 t 为最小周期重复至少两次得到的字符串 S ，其子串 S′ 的最小周期长度也为 |t|

证明

记 S′ 首字母为 x 末尾字母为 y

显然 border 必以 x 开头且以 y 结尾

S′ 其最大 border 以 x 开头以 S’ 中最后一个完整周期中的 y 结尾

那么 border 右边界距 S’ 有边界恰为 $|t|$

得证

#647、 OKR-Periods of Words

题目描述

串是有限个小写字符的序列,特别的一个空序列也可以是一个串

一个串 P 是串 A 的前缀，当且仅当存在串 B ,使得 $A = PB$

如果 $P \neq A$ 并且 P 不是一个空串，那么我们说 P 是 A 的一个 proper 前缀

定义 Q 是 A 的周期，当且仅当 Q 是 A 的一个 proper 前缀并且 A 是 QQ 的前缀(不一定要是 proper 前缀)

比如串 `abab` 和 `ababab` 都是串 `abababa` 的周期

串 A 的最大周期就是它最长的一个周期或者是一个空串(当 A 没有周期的时候),比如说, `ababab` 的最大周期是 `abab`

串 `abc` 的最大周期是空串

给出一个串，求出它所有前缀的最大周期长度之和

输入格式

第一行一个整数 k ,表示串的长度

接下来一行表示给出的串

输出格式

输出一个整数表示它所有前缀的最大周期长度之和

每个前缀找出最大周期长度

累加所有最大周期长度

样例输入

```
8
babababa
```

样例输出

```
24
```

数据范围

对于全部数据, $1 < k < 10^6$

#647、 OKR-Periods of Words

对于 $\mathrm{Pre}(S,i)$ 找出其最小非零 border 长度 j 那么 $i-j$ 即为该前缀的最大周期

根据定义 $\pi[i]$ 为 $\mathrm{Pre}(S,i)$ 的最大 border 长度 $\pi[\pi[i]-1]$ 即为次大

求出最小非零 border 长度即可求出最大周期

为处理方便令 $\mathrm{nxt}[i+1]=\pi[i]$，从 $1\sim|S|$ 考虑前缀长度 i

不断令 $i\leftarrow \mathrm{nxt}[i]$ 直到 $\mathrm{nxt}[i]=0$ 时停止

对于每个前缀求解最坏情况下时间复杂度 $O(|S|)$，总时间复杂度 $O(|S|^2)$

可采用类似并查集的思想

对于某次 i 的求解，若最终 最小非零 border 长度为 j，令 $\mathrm{nxt}[i]=j$

上述优化可避免后续重复回退

时间复杂度约为 $O(|S|)$

实验舱

青少年编程

走近科学 走进名校

谢谢观看