AC 自动机 & Manacher

唐懿宸

清华大学致理书院

AC 自动机 引入

给定 n 个模式串 $T_1, \cdots, T_n$ 和一个文本串 S

求每个模式串 T_i 在 T 中出现的次数。

比 KMP 更多的匹配。

$1 \leqslant n \leqslant 2 \times 10^5,\ \sum |T_i| \leqslant 2 \times 10^5,\ |S| \leqslant 2 \times 10^6$

AC 自动机

AC 自动机是一种支持**多模式串匹配**的数据结构

所谓多模式串匹配，就是用 $O(|S|)$ 的时间求出若干模式串 $T_1, T_2, \cdots, T_n$ 在 S 中出现了几次，出现在哪儿等等

比 KMP 遥遥领先！就是空间复杂度略大一点

直接做 KMP 的复杂度是 $O(n|S|)$ 的，不能接受

AC 自动机

在 KMP 中，每当我们失配，我们将从 $\mathrm{nxt}[p]$ 处重新开始匹配，更具体地，是从 **最长的与后缀相同的前缀** 开始继续匹配

同时我们也知道，Trie 是一种很好的加载多个字符串的数据结构

如果我们给 Trie 上的每个结点加上一个指针，指向最长的与后缀相同的前缀，是不是就能像 KMP 一样匹配?

fail / nxt

我们设 $\mathrm{str}(p)$ 表示 p 代表的字符串，$\mathrm{len}(p)$ 表示 p 代表的字符串的长度

我们设一个节点 p 的 $\mathrm{nxt}[p]$ 是在 Trie 树上满足 $\mathrm{str}(r)[1,\mathrm{len}(r)]=\mathrm{str}(p)[\mathrm{len}(p)-\mathrm{len}(r)+1,\mathrm{len}(p)]$ 中 $\mathrm{len}(r)$ 最大的那个

看着很复杂，其实意思就是 p 的 $\mathrm{nxt}[p]$ 就是串 $\mathrm{str}(p)$ 的**最长的、在 AC 自动机中后缀，所对应的结点**。

fail / nxt

显然 $\mathrm{str}(p)$ 的每一个后缀 $\mathrm{suffix}(i)$，若存在 r 满足 $\mathrm{str}(r) = \mathrm{suffix}(i)$，那么 r 是唯一的。

因此 $\mathrm{nxt}[p]$ 也是唯一的，因为我们想要**最长**的满足条件的后缀。

fail / nxt

举个例子

假设 $\mathrm{str}(p) =$ `aabbccdd`，有 $\mathrm{str}(r_1) =$ `bccdd`，$\mathrm{str}(r_2) =$ `ccdd`，$\mathrm{str}(r_3) =$ `cdd`，可以发现 r_1, r_2, r_3 都满足上述条件

但 $\mathrm{nxt}[p] = r_1$，因为 $\mathrm{len}(r_1)$ 最大

fail / nxt

在 AC 自动机中，我们一般把 KMP 的 nxt 数组称为 fail 数组，表示失配后跳转的位置。

fail / nxt

怎么求?

和 KMP 的 nxt 数组一样，我们可以用类似跳法求出 fail 数组。

只是从序列上跳变成了从 Trie 树上跳，都是跳。

fail / nxt

考虑一个节点 p，现在要算 $\mathrm{fail}[p]$。p 的父亲为 f，$\mathrm{str}[p] = \mathrm{str}[f] + c$，并且此时所有 $\mathrm{len(s)} < \mathrm{len(p)}$ 的结点 s 的 $\mathrm{fail}[s]$ 都已经求出

现在我们考虑 $\mathrm{fail}[f]$，如果存在 $\mathrm{son}[\mathrm{fail}[f]][c]$，那么有 $\mathrm{fail}[p] = \mathrm{son}[\mathrm{fail}[f]][c]$

否则我们令 $q = \mathrm{fail}[f]$ 继续沿着 fail 向上跳，直到达到根或者存在 $\mathrm{son}[q][c]$，此时将 $\mathrm{fail}[p]$ 设置为 $\mathrm{son}[q][c]$。

证明类似 KMP

回到原问题

用 $O(|S|)$ 的时间求出若干模式串 $T_1, T_2, \cdots, T_n$ 在 S 中出现了几次，出现在哪儿。

现在求出来了 fail 数组，怎么匹配呢?

AC 自动机上匹配

类似 KMP 的匹配思路

若在一个节点 p 没有找到出边 S_i，我们就跳回 $\mathrm{fail}[p]$ 继续尝试找出边 S_i，直到跳回根

如果走到一个被标记为末尾的节点，记录一个匹配

这样就完了吗?

AC 自动机上匹配

当你走到一个 $\mathrm{str}(p) =$ `aabb` 的点时，需要注意的是，此时若 Trie 树上也存在一个点 q 使得 $\mathrm{str}(q) =$ `abb` 并且 q 是终止节点，那么这个终止节点**不会被标记**

因为你走到 p 的路上是不会经过 q 的。

AC 自动机上匹配

但是，注意到 p 不断跳 $\mathrm{fail}(p)$ 一定能到达这类 q

所以可以在维护 fail 数组时，就进行结束标记下传：如果 $\mathrm{fail}(p)$ 是终止节点，那么 p 也是终止节点（伪终止结点）。

AC 自动机上匹配复杂度

类似 KMP

考虑将整个 Trie 树分层，跳 fail 相当于是将整个 Trie 在匹配串 S 上向右移动。

否则向右匹配一位相当于是挪动 S 串上的指针

因此时间复杂度为 $O(|S|)$

够快吗?

注意，我们说这里 $O(|S|)$ 的时间复杂度是基于“每次匹配，都只会耗去 $O(1)$ 的时间”

如果说有大量的 `a` 和一个 `aaaa..aaa` 是模式串，给定 `aaaa..aaaa` 去匹配。

那么每个 `a` 都会将所有的 `a` 匹配一遍，给出现次数 +1，该次移动就会产生 $O(\text{a 的个数})$ 次计算。

瓶颈是需要优化遍历末尾标记的次数。

优化

每一个点只有一个 fail 指针，根代表空串没有 fail（或者说根的 fail 就是自己）

那么如果我们给每个点都建一条 从 fail 到自己的边，最后会构成一棵**树**。

就叫 fail 树。

优化

那么就很简单了，我们知道如果一个点被匹配到了，那么它在 fail 树上的所有祖先都会被匹配到。

我们记录每个点被经过的次数（被匹配到的次数），然后在 fail 树上 DFS 一遍，统计每个点子数经过次数和，就是这个点被匹配到的次数。

优化

在实际的应用中，我们常常将 fail 树上祖先的儿子合并到当前节点上，具体来说：

如果结点 p 不存在边 c，那么 $\mathrm{son}[p][c]$ 会被记为 $\mathrm{son}[\mathrm{fail}[p]][c]$。

此时我们建立的 AC 自动机是可以做到：

1. 任何一个串都可以在这个 AC 自动机上走一遍
2. 如果给定的串中包含模式串，那么会在遍历时走到一个终止节点/能通过跳 fail 到终止节点的点上

KMP 练习题

对于字符串 S，对 S 的前 i 个字符构成的子串，既是它的后缀又是它的前缀的字符串中（它本身除外），最长的长度记为 $\mathrm{nxt}[i]$

现在想要求一个 num 数组，对于 S 的前 i 个字符构成的子串，满足

1. 既是它的后缀同时又是它的前缀，
2. 该后缀与该前缀不重叠

这种字符串的数量纪录为 $\mathrm{num}[i]$

$|S| \leqslant 2 \times 10^6$

分析

根据 nxt 数组的性质，我们可以发现，对于一个串 $str[1:i]$，$\mathrm{nxt}[i]$ 是他的前缀=后缀，$\mathrm{nxt}[\mathrm{nxt}[i]]$，$\mathrm{nxt}[\mathrm{nxt}[\mathrm{nxt}[i]]]$ 也是他的前缀=后缀。

所以 $\mathrm{num}[i]$ 本质就是 $str[1:i]$ 能跳 i 几次 nxt。

分析

为了保证前缀和后缀不重叠，暴力的方法是先跳到一个长度小于一半的 nxt，取这个的 $\mathrm{num}[i]$

但是全 `a` 串会非常非常慢

因此我们在跳的时候可以控制它只能在 $1/2$ 之前

分析

```
int tmp = 0;
for (int i = 0; i < len; i++)
{
    while (tmp >= 0 && str[i] != str[tmp])
        tmp = nxt[tmp];
    tmp++;
    while (tmp * 2 > i + 1) // 其实这里 tmp 最多只会跳一次 nxt
        tmp = nxt[tmp];
    // 这里算一下答案
}
printf("%lld\n", ans);
```

AC 自动机练习题-1

如果某段代码中不存在任何一段病毒代码，那么我们就称这段代码是安全的。现在委员会已经找出了所有的病毒代码段，试问，是否存在一个无限长的安全的二进制代码。

例如如果 $\{011, 11, 00000\}$ 为病毒代码段，那么一个可能的无限长安全代码就是 $010101\cdots$；如果 $\{01, 11, 000000\}$ 为病毒代码段，那么就不存在一个无限长的安全代码。

现在给出所有的病毒代码段，判断是否存在无限长的安全代码。

病毒代码个数 $n \leqslant 2000$，所有病毒代码的长度和不超过 3×10^4

分析

建立 AC 自动机，目标就是能够在 AC 自动机上一直走，但是不能走到一个终止结点。

注意，对于 AC 自动机上的一个结点，如果他的 fail 链上存在一个终止结点，那么它也被视为终止结点。

这样的 AC 自动机就可以开始匹配了，从根出发，判断是否存在一个没有终止结点的环就好了。

AC 自动机练习题-2

阿狸喜欢收藏各种稀奇古怪的东西，最近他淘到一台老式的打字机。打字机上只有 28 个按键，分别印有 26 个小写英文字母和 `B`、`P` 两个字母。经阿狸研究发现，这个打字机是这样工作的：

- 输入小写字母，打字机的一个凹槽中会加入这个字母(这个字母加在凹槽的最后)。
- 按一下印有 `B` 的按键，打字机凹槽中最后一个字母会消失。
- 按一下印有 `P` 的按键，打字机会在纸上打印出凹槽中现有的所有字母并换行，但凹槽中的字母不会消失。

例如，阿狸输入 `aPaPBbP`，纸上被打印的字符如下：

```
a
aa
ab
```

我们把纸上打印出来的字符串从 1 开始顺序编号，一直到 n。

m 个询问，每次给定 $x, y (1 \leqslant x, y \leqslant n)$，求第 x 个打印的字符串在第 y 个打印的字符串中出现了多少次。

$1 \leqslant n \leqslant 10^5$，$1 \leqslant m \leqslant 10^5$，输入给打印机的总长度小于等于 10^5。

分析

字符串重叠很多，暴力找出来所有的字符串是不太可行的。

看看能不能建立 Trie 树：可以发现，`B` 就是回到父亲结点，从父亲结点开始往下；`P` 是给当前结点加终止标记。

这样就快速的将所有的字符串建立了 Trie 树，也就可以建立 AC 自动机来匹配了。

分析

问题是想要求 串 x 在 串 y 中出现了几次。

如果 y 串中有结点可以通过 fail 指针走到 x 的终止结点，那么 x 就在 y 中出现过。

如果 y 在 Trie 树上的链有 n 个结点可以通过 fail 走到 x 的终止结点，那么 x 就在 y 中出现过 n 次。

问题就变成了，在 x 的终止结点的 fail 树的子树中，有多少个 y 这条链上的点。

分析

问题的瓶颈是，要标记 Trie 树上 y 对应的链，然后找他们有多少个在 fail 树的 x 终止结点子树内。

这是两棵树，不能直接做树链剖分。

分析

离线。

还是利用 `dfs` 序，fail 树上 x 终止结点的子数是一个区间。用线段树，线段树区间就是 fail 树上 dfs 序区间。

我们在 Trie 树上 dfs，每走到一个点就做标记，离开了就去掉它的标记，也就是在线段树上这个点对应的节点 +1/-1。

走到 y 的终止节点的时候，考虑所有 $(x_1, y), (x_2, y), \cdots$ 的询问，在线段树上查这些询问的区间和就好了。

分析

线段树上操作就是单点修改区间求和。可以改成树状数组。

时间复杂度：

- 建立 AC 自动机 $O(|S|)$
- 对 fail 树求 dfs 序 $O(|S|)$
- 遍历 Trie 树并查询 $O(|S|\log|S|)$

总时间复杂度 $O(|S|\log|S|)$，空间复杂度 $O(|S|)$

AC 自动机练习题-3

文本生成器可以随机生成一些文章——总是生成一篇长度固定且完全随机的文章。生成的文章中每个字符都是完全随机的。

如果一篇文章中至少包含使用者们了解的一个单词，那么我们说这篇文章是可读的。

给定 n 个认识的单词，生成文章的长度 m，求所有可能生成的文章中有多少个文章是可读的。

$n \leqslant 60, m \leqslant 100$，单词长度 $\leqslant 100$，所有字符串只包含大写字母。

分析

「存在至少一个」单词是认识的

等价于

所有字符串数量 m^{26} - 「一个单词都不包含」的数量

这种思路叫「正难则反」，很常用。

「一个单词都不包含」的数量 显然是会更好做的。

分析

将所有认识的单词建立 AC 自动机。

如果我们在 AC 自动机上跑，什么意味着「一个单词都不包含」？

分析

对于一个结点，如果他跳 fail 能到一个终止结点，那么他也被视为终止结点。

所以「一个单词都不包含」等价于：从根出发，不能走到一个终止结点。

分析

令 $dp_{len,p}$ 表示当前串长度为 len，目前走到 AC 自动机上的结点 p，并且当前没有经过任何一个终止节点（也就意味着生成的串中不包含任何一个单词），有多少种走法。

转移时枚举 p 的所有儿子 $son[p][c]$：

$$dp_{len+1,son[p][c]} += dp_{len,p}$$

要求 p 和 $son[p][c]$ 都不是终止节点。

分析

时间复杂度 $O(m \times |S| \times 26)$。

空间复杂度 $O(m \times |S|)$。

AC自动机 / KMP 练习题

给定 m 个不吉利的数字 $a_1, \cdots, a_m (0 \leqslant a_i \leqslant 9)$

n 位数字 $x_1, \cdots, x_n (0 \leqslant x_i \leqslant 9)$ 中如果没有长度为 m 的一段恰好等于 $a_1, \cdots, a_m$，就称这个 $x_1, \cdots, x_n$ 是吉利的。

求所有的 n 位数字中有多少个是吉利的。

注意 $n \leqslant 10^9, m \leqslant 20$。

分析

使用 KMP/AC 自动机建图

然后跑矩阵乘法。

Manacher

问题定义：

给定一个长度为 n 的字符串 S，请找到所有对 (i, j) 使得子串 $S[i \dots j]$ 为一个回文串。

Manacher 是一种在 $O(n)$ 的时间复杂度内找到一个串所有回文子串的算法。

Manacher

理论上回文子串最多有 n^2 个。

但是关于回文串的信息可用一种更紧凑的方式表达：对于每个位置 $i = 0 \ldots n-1$，用 $d_1[i]$ 和 $d_2[i]$ 分别表示以位置 i 为**中心**的长度为**奇数**和长度为**偶数**的回文串个数。

对于这个 $d_1[i]$，实际上意味着，所有以 i 为中心的，向左向右延伸 $1, 2, 3, \ldots, d_1[i]-1$ 个字符，都会构成回文串。

插入特殊字符

偶数回文串我们不太喜欢，因为中心在两个字符之间。

我们在每个字符之间插入一个特殊字符，比如 `$`，这样所有长度为偶数的字符串变成了以 `$` 为中心的奇数回文串。

为了防止越界，我们在字符串的开头和结尾也插入一个特殊字符。
但是我一般的习惯会再在开头插入一个 `#`，反正不和任何字符匹配。

综上，如果我们原本的字符串是 `aba`，那么我们处理后的字符串变成了 `#a$b$a$`，长度是 $2n+2$。

此时，对于我们实际上存在的回文串：

- 如果长度是奇数，那么会以原本的字符为中心，如果中心是 i，那么会属于 $d_1[i]$。
- 如果长度是偶数，那么会以特殊字符为中心，如果中心是 j，那么会属于 $d_1[j]$。

这样就只要记录 d_i 了：d_i 表示以第 i 个字符（包括特殊字符）为中心，有多少个长度为奇数的回文串。

换句话，d_i 也表示以第 i 个字符为中心，向左向右延伸 $1, 2, 3, \ldots, d_i - 1$ 个字符，都会构成回文串。但是延伸 d_i 个字符后不会构成回文串。

得到 d_i

现在假设我们要对下一个 i 计算 d_i，而 $d_1, d_2, \cdots, d_{i-1}$ 已计算完毕。

对于前面的 $d_1, d_2, \cdots, d_{i-1}$，我们记 mid 和 r 表示目前所有的回文串中，能达到最右的回文串的中心和最右边的字符的位置。

现在要算 d_i，我们有两种情况：

- 如果 $i > r$，那么我们暴力以 i 为中心向左向右扩展，直到不能扩展为止，这个时候 d_i 就是扩展的长度。

- 如果 $i \leq r$，说明 i 此时在以 mid 为中心的一个回文串内，我们可以利用这个信息：

  i 关于 mid 的对称点是 $2 \times mid - i$，记录为 p，我们之前已经算出来以 p 为中心的回文串的长度 d_p

  以回文的性质，此时的 d_p 等价于以 i 为中心的回文串

  但是，以 i 为中心扩展 d_p 个字符后，可能会超过 r

  所以实际上，我们能通过 mid, r 的信息，得到 d_i 的一个下界：

$$d_i = \min(d_{2 \times mid - i}, r - i)$$

在这个基础上，我们再暴力扩展 d_i，直到不能扩展为止。

```
void manacher()
{
    int r = 0, mid = 0;
    for (int i = 1; i < n; i++)
    {
        if (i < r)
            d[i] = min(d[(mid << 1) - i], r - i);
        else
            d[i] = 1;
        while (1 <= i - d[i] && i + d[i] <= n && s[i - d[i]] == s[i + d[i]])
            d[i]++;
        if (i + d[i] > r)
            r = i + d[i], mid = i;
    }
}
```

时间复杂度

事实上我们可以发现

- 如果 $d_i < r - i$，那么其实不会暴力扩展
- 如果 $d_i = r - i$，那么每一次暴力扩展都会使 r 增加 1

r 在整个算法运行过程中从不减小，最多增加 n 次，所以总时间复杂度是 $O(n)$。

Manacher 基础题

给定一个只有小写字母组成的字符串 S，求 S 中的最长回文子串的长度。

$|S| \leqslant 10^7$

你已经知道了 d_i 怎么算，那么答案怎么表示?

分析

答案是 $\max d_i$，这里 i 可以是原有字符串的位置，也可以是插入的特殊字符的位置。

Manacher 应用

给定字符串 S，求 S 的最长双回文串 T，即 T 可以分成两部分 X 和 Y，均不为空串，且 X 和 Y 都是回文串。

$|S| \leqslant 10^5$

分析

在求出来 d 后，我们可以进一步求出 $l[i]$、$r[i]$ 分别表示以 i 为左端点和右端点的最长回文串的长度。

需要注意，如果 $l[1] = 7$，那么其实以 3 为左端点的回文串长度至少为 $7 - 2 = 5$，但是我们用 d 求出来的 $l[3]$ 可能不够大，所以需要递推更新一下 l（r 同理）。

为了保证不重叠，我们选择 `$` 的位置作为 X 和 Y 的分割点，答案就是 $\max_{S[i]=\$} l[i] + r[i]$

谢谢！