Knuth–Morris–Pratt(KMP) 算法 *

张晴川
qzha536@aucklanduni.ac.nz

April 19, 2020

1 题意

设文本串为 $T(1 \le |T| \le 10^5)$，模式串为 $S(1 \le |S| \le 10^5)$，求 S 在 T 中的所有出现位置。下标均从 1 开始。

例：$S = \texttt{14}, T = \texttt{114514}$，那么匹配的末尾位置为 $\{3, 6\}$

2 算法

首先我们考虑用一个没有出现过的字符 `#` 把 `S` 和 `T` 拼成一个串 `S#T`，例如 `14#114514`。

现在考虑对于位置 `i`，有哪些以它结尾的串等于相同长度的**前缀**，以 `i` 结尾的前缀本身除外。例如对于串 `aba#ababa` 的最后一个位置 `i` = 9，匹配的长度为 $\{3, 1, 0\}$，分别对应 $\{\texttt{aba}, \texttt{a}, \epsilon\}$[1]，定义集合里的串为以 `i` 结尾的 **border**。

border：既是前缀也是后缀

问题转化 我们用 `match(i,len)` = `True`/`False` 表示以 `i` 结尾，长度为 `len` 的串是否是 `border`，用 `borders[i]` 表示以 `i` 结尾，所有 `border` 的集合。原问题可以转化为：对于多少位置 `i`，`match(i,|S|)` = `True`。

.....

如何求 `borders[i]` 首先我们可以发现 `match(i,len)` = `True` 成立有两个条件：

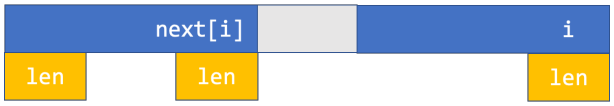
1. `match(i-1,len-1)` = `True`
2. `S[len]` = `S[i]`

于是我们可以枚举 `borders[i-1]` 里的元素，如果下一位恰好和 `S[i]` 匹配，那么就 `+1` 就可以得到 `borders[i]` 里的一个元素。最后加入 0 即可。设总串长 $n = |S| + 1 + |T|$，由于一共有 n 个位置，每次最多枚举 n 个元素，复杂度为 $O(n^2)$，需要加速。

*更多内容请访问：`https://github.com/SamZhangQingChuan/Editorials`

[1] ϵ 表示空串

更高效的表示方法 我们用 next[i] 表示 borders[i] 里最长的 border。假设 len 也是 i 的一个 border，不难发现 len 也是 next[i] 的 border。(参考下图，想想为什么)



所以除了 next[i] 外，其余元素都是 next[i] 的 border，于是可以得到：

$$\texttt{borders[i]} = \{\texttt{next[i]}\} \cup \texttt{borders[next[i]]}$$

展开写的话就是：

$$\texttt{borders[i]} = \{\texttt{next[i]}, \texttt{next[next[i]]}, \ldots, 0\}$$

另外根据定义，borders[1] $= \{0\}$。

递推求解 由上述解释，只需要求出 next[i-1]，就可以得到整个 borders[i-1]。如果知道了 borders[i-1]，next[i] 就等于 borders[i-1] 里从大到小第一个可以转移到 i 的元素再加 1 。如果全部都无法转移，那么 next[i] $= 0$。

完全解决 在求出所有 next[i] 之后，只需要计算有哪些位置的 next[i] $= |S|$ 即可。

总结 一言以蔽之，KMP 算法的本质就是：

以增量法递推求出每个前缀的 border 集合

3 复杂度分析

寻找转移时，每失败一次就会减少一个 border，而每向右移一位最多添加一个，所以复杂度是线性的。

4 习题

1. 实现 KMP 算法
2. 「NOI2014」动物园
3. 以同样思路分析 AC 自动机的算法。

5 核心代码

```cpp
int KMP(string S, string T) {
    string s = " " + S + "#" + T; // 在前面添加空格使得下标从 1 开始
    vector<int> next(s.size());
    next[1] = 0;
    for(int i = 2; i < s.size(); i++) {
        for(int len = next[i - 1];; len = next[len]) {
            if(s[len + 1] == s[i]) {  // 如果找到了可以转移的 border
                next[i] = len + 1;
                break;
            } else {
                if(len == 0) { // 如果没有可以转移的
                    next[i] = 0;
                    break;
                }
            }
        }
    }
    // 枚举 T 的部分有多少位置 next[i] = |S|
    int count = 0;
    for(int i = 1; i <= T.size(); i++) {
        count += next[i + S.size() + 1] == S.size();
    }
    return count;
}
```