为 AI（和学过 DLCO 的）学生准备的 KMP 算法介绍

Magical Girl YunHua Zhong

July 4 2022

1 介绍

KMP 算法是处理字符串搜索问题的重要算法。在信息竞赛和算法课中，它也是一个有趣但困难的算法。KMP 算法的代码异常的短，“前缀”字符串的概念也并不难理解，但算法中的技巧却很难想通。
但是，KMP 算法的核心其实不难理解，因为我们已经在**数字电路和计算机组成**课程中学过了相近的算法。
这篇文章是写给所有学过 DLCO 这门课程的人的（比如 AI 的学生），在这篇文章中，我将借助 DLCO 中的知识来讲清 KMP 算法及其本质。
无论算法课中是否考到该算法，我希望这篇文章可以作为一个有趣的思考体验。

2 算法介绍

首先我们来介绍一下 KMP 算法所解决的问题，即在长字符串中搜索特定的字符串。不难想到，解决这个问题的一个暴力的方法就是从每一个位置开始枚举，以此来判断子字符串的位置。
若长字符串的长度为 n，子字符串的长度为 m，可以知道，在最坏的情况下，暴力算法的复杂度为 $O(mn)$。而 KMP 则是利用前缀数组来记录已知的信息，从而将算法的复杂度优化到 $O(m+n)$。

3 前置知识：有限状态机

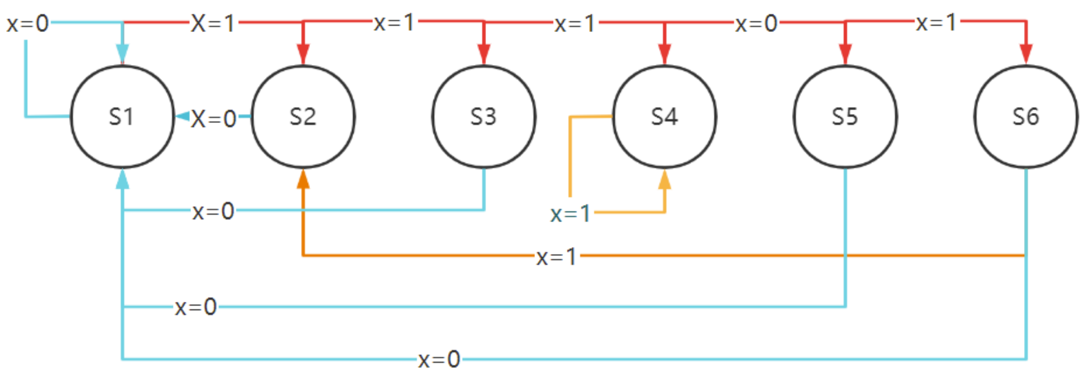
3.1 总括

在开始分析之前，我将首先回顾一下 DLCO 中所学的知识。
在时序逻辑电路这一章中，我们曾遇到了一个问题，即“设计一个电路来识别特定的二进制串”。
假设在识别出来的时候，这个数字电路可以将当前的位置记录下来，那么如果我输入一个长序列，这个自动机可以解决特定序列的“字符串匹配”问题。

让我们回忆一下，这是怎么实现的。
假设我们想设计一个自动机，使它可以识别“11101”字符串。那停下并拿起你手中的笔，作为数字电路的复习也好，让我们一起动手画出一个不需要化简状态的自动机。
(此处你应该已经在想并画完一个自动机了，但如果你忘了也没关系，让我们继续看下去吧。)

3.2 前进和回退

那我们来分析电路图。假设我把六个状态分别用 S1-S6 表示，X 代表输入的数，那么简单的电路图如下：



电路图没有化简，我们可以看到，红色箭头是正确的输入，随着正确的输入我们会一步一步走到终点；蓝色和橙色的箭头是回退（错误）的输入，会让暂时的状态电路图往回走。
这时让我们思考一个问题：**为什么橙色的箭头没有回退到底？**
此外，还有一个更重要的事实。
让我们模拟一遍这个电路（算法）的过程：可以发现，这个电路（算法）的时间复杂度也是 $O(m+n)$！因为每个时刻电路都会输入一次，而这个算法在**线性时间**中就可以输出所有的特定匹配，这和我们想要的结果是一样的！

那，这个电路（算法）做了哪些事情呢？只有匹配和回退。
如果匹配的位数相同，电路向前走；否则电路回退跳转到**合适的位置**。至于**合适的位置是哪里？**带着这两 k 个问题，让我们继续我们的分析。
如果我们借鉴这个思路的话，那么字符串匹配算法也是相似的：如果匹配，就向前移动一格，否则就后退至**合适的位置**。

4 回到 KMP

4.1 新定义 1：后缀和前缀

后缀是指从某个位置 i 开始到整个串末尾结束的一个特殊子串。字符串 S 的从 i 开头的后缀表示为 *Suffix(S,i)*，也就是 *Suffix(S,i)* $= S[i..(|S|-1)]$。
真后缀指除了 S 本身的 S 的后缀。
举例来说，字符串 'abcabcd' 的所有后缀为 'd, cd, bcd, abcd, cabcd, bcabcd, abcabcd'，而它的真后缀为 'd, cd, bcd, abcd, cabcd, bcabcd'。
前缀是指从串首开始到某个位置 i 结束的一个特殊子串。字符串 S 的以 i 结尾的前缀表示为 *Prefix(S,i)*，也就是 *Prefix(S,i)* $= S[0..i]$。
真前缀指除了 S 本身的 S 的前缀。
'abcabcd' 的真前缀作为练习，请读者自行写出。

4.2 新定义 2：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 n 的数组 π。
其中 $\pi[i]$ 的定义是：
1. 如果子串 $s[0\dots i]$ 有一对相等的真前缀与真后缀：$s[0\dots k-1]$ 和 $s[i-(k-1)\dots i]$，那么 $\pi[i]$ 就是这个相等的真前缀（或者真后缀，因为它们相等）的长度，也就是 $\pi[i]=k$
2. 如果不止有一对相等的，那么 $\pi[i]$ 就是其中最长的那一对的长度
3. 如果没有相等的，那么 $\pi[i]=0$
简单来说，$\pi[i]$ 就是子串 $s[0\dots i]$ 最长的相等真前缀与真后缀的长度，描述如下：

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

特别地，规定 $\pi[0]=0$。

举例来说，对于字符串 'abcabcd'，
$\pi[0]=0$，因为 'a' 没有真前缀和真后缀，根据规定为 0
$\pi[1]=0$，因为 'ab' 无相等的真前缀和真后缀
$\pi[2]=0$，因为 'abc' 无相等的真前缀和真后缀
$\pi[3]=1$，因为 'abca' 只有一对相等的真前缀和真后缀：'a'，长度为 1
$\pi[4]=2$，因为 'abcab' 相等的真前缀和真后缀只有 'ab'，长度为 2
$\pi[5]=3$，因为 'abcabc' 相等的真前缀和真后缀只有 'abc'，长度为 3
$\pi[6]=0$，因为 'abcabcd' 无相等的真前缀和真后缀
同理可以计算字符串 'aabaaab' 的前缀函数为 $[0,1,0,1,2,2,3]$。

计算前缀函数的算法并不是本文的重点，可以查看：https://oi-wiki.org/string/kmp/

4.3 寻找回退

然后，我们解决最核心的问题：**合适的位置是哪个位置**？
首先我们需要明白为什么要回退。回退的原因是，当下这一位和子字符串失配，因此我们需要回退到合适的位置，看看能否再一次匹配。
注意到上图中 S4 处的回退。当检测到 “111” 后再检测到 “1” 时，虽然不能往前进到 “1110”，但仍然可以保留在 “111” 处。

回到字符串匹配问题。不失一般性，假设字符串匹配到了 a_k，子字符串已经匹配到了 s_i，但 a_{k+1} 和 s_{i+1} 失配了，如下图所示：

$$\overbrace{\underbrace{s_0\ s_1}_{j}\ s_2\ s_3}^{\pi[i]}\ \ \cdots\ \ \overbrace{s_{i-3}\ s_{i-2}\ \underbrace{s_{i-1}\ s_i}_{j}}^{\pi[i]}\ s_{i+1}$$

那么，当我们根据前缀函数的定义，意识到子字符串 s[0..i] 的前后 $\pi[i]$ 位一样时，我们可以**首先**回退到 $\pi[i]$ 处，在上图中是 s_3 处，继续进行匹配。
至于为什么可以呢，如果 $\pi[i+1]$ 实配，那次长的匹配就是从 s$[\pi[i+1]]$ 开始，因为定义中前缀函数是前后缀相同时**最长的公共序列**。即我们试着匹配 a_{k+1} 和 $s_{\pi[i]+1}$。
若仍然无法匹配，我们继续查看 $\pi[\pi[i]]$，再次回退匹配（$\pi[\pi[i]]$ 在图中所示的对应位置是 j）。
若匹配到最后都不行，则我们不得不遗憾的说，这一位无法和任何一位匹配，匹配必须重新开始。

4.4 代码与讲解

接下来展示 KMP 算法的代码，附有详细的注释（C 语言）：
尽管如此，代码不是本文章的核心，仅供参考。

```c
char lb[1000005],la[1000005];
//数组名为la和lb，其中la是主字符串，lb为子字符串
int kmp[1000005];   //kmp数组即为前缀数组
int main(){
        scanf("%s%s",la+1,lb+1);
        int lena=strlen(la+1),lenb=strlen(lb+1);

        /*此段为计算前缀数组的算法，本文章中没有细讲*/
        int u=0;
        for(int i=2;i<=lenb;i++){
                while(u&&lb[i]!=lb[u+1]){
                        u=kmp[u];
```

```
                }
                if(lb[u+1]==lb[i])u++;
                kmp[i]=u;
        }

        /*此段为KMP算法的核心*/
        int j=0;
        for(int i=1;i<=lena;i++){
                while(j&&lb[j+1]!=la[i])j=kmp[j]; //回退到合适的位置
                if(lb[j+1]==la[i])j++; //匹配
                if(j==lenb){//匹配成功
                        printf("%d\n",i-lenb+1);//输出位置
                        j=kmp[j];//继续跳转匹配
                }
        }
    return 0;
}
```

5 总结

应该来说，文章到这里就结束了。希望这可以让大家更清晰的明白 KMP 算法。
当然,写成这篇文章对我的启示主要是:学的事物总归会在某些方面产生作用,即使是看上去以后不太用得到的数字电路。~~(数字电路刚刚出分，这话可不能让数电老师听到)~~
当然，这也是我第一次用 Latex 写作，这也是难忘的经历。

参考资料与注释

本文章的 4.1 和 4.2 部分选自 oi-wiki，版权协议为 CC-BY-SA 4.0。