数据结构与算法（四）

张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008. 6 （ “十一五”国家级规划教材）

http://www.jpk.pku.edu.cn/pkujpk/course/sjjg

主要内容

- 字符串基本概念
- 字符串的存储结构
- 字符串运算的算法实现
- 字符串的模式匹配
  - 朴素算法
  - **KMP算法**

无回溯匹配

- 匹配过程中，一旦 p_j 和 t_i 比较不等时，即

```
P.substr(1,j-1) == T.substr(i-j+1,j-1)
```

但 $p_j \neq t_i$

  - 该用 P 中的哪个字符 p_k 和 t_i 进行比较？
  - 确定右移的位数
  - 显然有 $k < j$, 且不同的 j, 其 k 值不同
- Knuth-Morrit-Pratt (KMP)算法
  - k 值仅仅依赖于模式 P 本身，而与目标对象T无关

KMP算法思想

T = a b c d e f a b c d e f f

P = a b c d e f f

a b c d e f f

$T \quad t_0\ t_1\ \ldots\ t_{i\text{-}j\text{-}1}\ t_{i\text{-}j}\ t_{i\text{-}j+1}\ t_{i\text{-}j+2}\ \ldots\ t_{i\text{-}2}\ t_{i\text{-}1}\ t_i\ \ldots\ t_{n\text{-}1}$

$\| \quad \| \quad \| \quad \| \quad \| \quad \times$

$P \quad p_0\ p_1\ p_2\ \ldots\ p_{j\text{-}2}\ p_{j\text{-}1}\ p_j$

则有 $t_{i\text{-}j}\ t_{i\text{-}j+1}\ t_{i\text{-}j+2}\ \ldots\ t_{i\text{-}1} = p_0\ p_1\ p_2 \ldots p_{j\text{-}1}$ (1)

朴素下一趟 $p_0\ p_1\ \ldots\ p_{j\text{-}2}\ p_{j\text{-}1}$

如果 $p_0\ p_1 \ldots p_{j\text{-}2} \neq p_1\ p_2 \ldots p_{j\text{-}1}$ (2)

则立刻可以断定

$p_0\ p_1 \ldots p_{j\text{-}2} \neq t_{i\text{-}j+1}\ t_{i\text{-}j+2}\ \ldots\ t_{i\text{-}1}$

(朴素匹配的)下一趟一定不匹配，可以跳过去

$p_0\ p_1\ \ldots\ p_{j\text{-}2}\ p_{j\text{-}1}$

4.3字符串的模式匹配

T = a b c d e f a b c d e f f

P = a b c d e f f

a b c d e f f

同样，若 $p_0\ p_1\ \ldots p_{j-3} \neq p_2\ p_3 \ldots p_{j-1}$

则再下一趟也不匹配，因为有

$$p_0\ p_1\ \ldots p_{j-3} \neq t_{i-j+2}\ t_{i-j+3}\ \ldots\ t_{i-1}$$

直到对于某一个“k”值（首尾串长度），使得

$$p_0\ p_1\ \ldots p_k \neq p_{j-k-1}\ p_{j-k} \ldots p_{j-1}$$

且 $p_0\ p_1\ \ldots p_{k-1} = p_{j-k}\ p_{j-k+1} \ldots p_{j-1}$

模式右滑 **j-k** 位

$$\begin{array}{ccccc}
t_{i-k} & t_{i-k+1} & \ldots & t_{i-1} & t_i \\
\| & \| & & \| & \times \\
p_{j-k} & p_{j-k+1} & \ldots & p_{j-1} & p_j \\
\| & \| & & \| & ? \\
p_0 & p_1 & \ldots & p_{k-1} & p_k
\end{array}$$

则 $p_0\ p_1\ \ldots p_{k-1} = t_{i-k}\ \ t_{i-k+1}\ \ \ldots\ \ t_{i-1}$

字符串的特征向量N

设模式 P 由 m 个字符组成，记为

$$P = p_0\ p_1 p_2 p_3 \ldots\ldots p_{m-1}$$

令 特征向量 N 用来表示模式 P 的字符分布特征，简称 N 向量由m个特征数 $n_0 \ldots n_{m-1}$ 整数组成，记为

$$N = n_0 n_1 n_2 n_3 \ldots\ldots n_{m-1}$$

N 在很多文献中也称为 next 数组，每个 n_j 对应 next 数组中的元素 next[j]

字符串的特征向量N：构造方法

- P 第 j 个位置的特征数 n_j，首尾串最长的 k
  - 首串： $p_0 \quad p_1 \quad \ldots \quad p_{k-2} \quad p_{k-1}$
  - 尾串： $p_{j-k} \quad p_{j-k+1} \quad \ldots \quad p_{j-2} \quad p_{j-1}$

$$
next[j] = \begin{cases} -1, & j = 0\text{时候} \\ \max\{k: 0 < k < j \ \ \& \ P[0\ldots k\text{-}1] = P[j\text{-}k\ldots j\text{-}1]\}, & \text{首尾配串最长k} \\ 0, & \text{其他} \end{cases}
$$

4.3 字符串的模式匹配

	0	1	2	3	4	5	6	7	8	9
P =	a	a	a	a	b	a	a	a	a	c
N =	-1	0	1	2	1	0	1	2	3	4

X (应为3)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
T =	a	a	b	a	a	a	a	a	a	b	a	a	a	a	c	b
P =	a	a	~~a~~	a	b	a	a	a	a	c						
				a	a	a	a	~~b~~	a	a	a	a	c			
							a	a	a	a	b	a	a	a	a	c

i=2, j=1, N[j]=0

i=7, j=4, N[4]=1 X

错过了！

KMP模式匹配示例

	0	1	2	3	4	5	6
P =	a	b	a	b	a	b	b
N =	-1	0	0	1	2	3	4

	0	1	2	3	4	5	6	7	8	9	10	11	12	
T =	a	b	a	b	a	b	a	b	a	b	a	b	b	
P =	a	b	a	b	a	b	~~b~~ X							i=6,j=6, N[j]=4
			a	b	a	b	a	b	~~b~~ X					i=8,j=6, N[j]=4
					a	b	a	b	a	b	~~b~~ X			i=10, j=6, j’=4
							a	b	a	b	a	b	b	√

KMP模式匹配算法

```
int KMPStrMatching(string T, string P,  int *N, int start) {
    int j= 0;                          // 模式的下标变量
    int i = start;                     // 目标的下标变量
    int pLen = P.length( );            // 模式的长度
    int tLen = T.length( );            // 目标的长度
    if (tLen - start < pLen)           // 若目标比模式短，匹配无法成功
        return (-1);
    while ( j < pLen  &&  i < tLen)  {  // 反复比较，进行匹配
        if ( j == -1  ||  T[i] == P[j])
            i++,  j++;
        else j = N[j];
    }
    if (j >= pLen)
        return (i-pLen);                // 注意仔细算下标
    else return (-1);
}
```

对应的求特征向量算法框架

- 特征数 n_j ($j>0$, $0\le n_{j+1}\le j$)是递归定义的，定义如下：
  1. $n_0 = -1$，对于$j > 0$的 n_{j+1}，假定已知前一位置的特征数n_j，令 $k = n_j$；
  2. 当 $k\ge 0$ 且 $p_j \ne p_k$ 时，则令 $k = n_k$；让步骤2循环直到条件不满足
  3. $n_{j+1} = k + 1$；// 此时，$k == -1$或 $p_j == p_k$

字符串的特征向量N ——非优化版

```
int findNext(string P) {
    int j, k;
    int m = P.length( );                  // m为模式P的长度
    assert( m > 0);                       // 若m＝0，退出
    int *next = new  int[m];              // 动态存储区开辟整数数组
    assert( next != 0);                   // 若开辟存储区域失败，退出
    next[0] = -1;
    j = 0;  k = -1;
    while (j < m-1) {
        while (k >= 0 && P[k] != P[j])// 不等则采用 KMP 自找首尾子串
            k = next[k];                  // k 递归地向前找
         j++;   k++;  next[j] = k;
    }
    return next;
}
```

求特征向量N

N =	-1	0	1	2	3	0	1	2	3	4
	0	1	2	3	4	5	6	7	8	9
P =	a	a	a	a	b	a	a	a	a	c

j= 9 k = 0

首串→ a

首串→ a a

首串→ a a a

首串→ a

首串→ a a

首串→ a a a

首串→ a a a a

模式右滑j-k位

$$
\begin{array}{cccccccc}
t_{i-j} & t_{i-j+1} & t_{i-j+2} & \cdots & t_{i-k} & t_{i-k+1} & \cdots\ t_{i-1} & t_i \\
\| & \| & \| & & \| & \| & \| & \times \\
p_0 & p_1 & p_2 & \cdots & p_{j-k} & p_{j-k+1} & \cdots\ p_{j-1} & p_j \\
 & & & & \| & \| & \| & ? \\
 & & & & p_0 & p_1 & \cdots\ p_{k-1} & p_k
\end{array}
$$

$$p_0\ p_1\ \ldots p_{k-1} = t_{i-k}\quad t_{i-k+1}\quad \cdots\quad t_{i-1}$$

$$t_i \neq p_j\ ,\quad p_j == p_k?$$

KMP匹配

j	0	1	2	3	4	5	6	7	8
P	a	b	c	a	a	b	a	b	c
K		0	0	0	1	1	2	1	2

目标 *a a b c b a b c a a b c a a b a b c*

a ~~b~~ c a a b a b c

a b c ~~a~~ a b a b c

这行冗余 *~~a~~ b c a a b a b c*

a b c a a b ~~a~~ b c

a b c a a b a b c √

$N[1]= 0$

$N[3] = 0$

$N[0] = -1$

$N[6]= 2$

上面 P[3]==P[0], P[3] ≠ T[4]，再比冗余

字符串的特征向量N ——优化版

```
int findNext(string P) {
    int j, k;
    int m = P.length( );                   // m为模式P的长度
    int *next = new  int[m];               // 动态存储区开辟整数数组
    next[0] = -1;
    j = 0;  k = -1;
    while (j < m-1) {                      // 若写成 j < m 会越界
        while (k >= 0 && P[k] != P[j])// 若不等，采用 KMP 找首尾子串
            k = next[k];                   // k 递归地向前找
        j++;   k++;
        if (P[k] == P[j])
            next[j] = next[k];             // 前面找 k 值，没有受优化的影响
        else next[j] = k;                  // 取消if判断，则不优化
    }
    return next;
}
```

next数组对比

序号j	0	1	2	3	4	5	6	7	8	
P	a	b	c	a	a	b	a	b	c	
k		0	0	0	1	1	2	1	2	非优化版
$p_k==p_j$?		≠	≠	==	≠	==	≠	==	==	
next[j]	-1	0	0	-1	1	0	2	0	0	优化版

KMP算法的效率分析

- 循环体中” j = N[j];” 语句的执行次数不能超过 n 次。否则，
  - 由于“j = N[]; ”每执行一次必然使得j减少(至少减1)
  - 而使得 j 增加的操作只有“j++ ”
  - 那么，如果“j = N[j]; ”的执行次数超过n次，最终的结果必然使得 j 为**比-1小很多的**负数。这是不可能的（j有时为-1,但是很快+1回到0）。
- 同理可以分析出求N数组的时间为O(m)

故，KMP算法的时间为O(n+m)

总结：单模式的匹配算法

算法	*预处理时间效率*	*匹配时间效率*
朴素匹配算法	0 (无需预处理)	$\Theta(n\ m)$
KMP算法	$\Theta(m)$	$\Theta(n)$
BM算法	$\Theta(m)$	最优 (n/m), 最差 $\Theta(nm)$
位运算算法 （*shift-or*, *shift-and*)	$\Theta(m+\|\Sigma\|)$	$\Theta(n)$
Rabin-Karp算法	$\Theta(m)$	平均 $(n+m)$, 最差$\Theta(nm)$
有限状态自动机	$\Theta(m\ \|\Sigma\|)$	$\Theta(n)$

思考：不同版本特征值定义

j 位匹配错误，则 j=next[j]

$$\text{next}[j]=\begin{cases}-1, & \text{对于 } j=0\\ \max\{k:0<k<j\ \ \&\&\ P[0...k\text{-}1]=P[j\text{-}k...j\text{-}1]\ \}, & \text{如果k存在}\\ 0, & \text{否则}\end{cases}$$

j 位匹配错误，则 j=next[j-1]

$$\text{next}[j]=\begin{cases}0, & \text{对于 } j=0\\ \max\{k:0<k<j\ \ \&\&\ P[0...k]=P[j\text{-}k...j]\ \}, & \text{如果k存在}\\ 0, & \text{否则}\end{cases}$$

参考资源

- **Pattern Matching Pointer**
  - http://www.cs.ucr.edu/~stelo/pattern.html
- **EXACT STRING MATCHING ALGORITHMS**
  - http://www-igm.univ-mlv.fr/~lecroq/string/
  - 字符串匹配算法的描述、复杂度分析和C源代码

数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材