KMP 算法中 next 数组的计算方法研究

汤亚玲
（安徽工业大学 计算机学院，安徽 马鞍山 243002）

摘 要：next 数组的计算方法是 KMP（Knuth—Morris— Pratt）算法的难点和核心。当前数据结构教材中普遍采用递推的方式来计算 next 数组值。文中给出一种新的采用递归思想设计的计算 next 数组的算法，并对当前数据结构教材中对 next 数组定义的其它一些改进方式进行了讨论与分析。实验数据表明，递归算法的思想正确；并且，从算法设计上考虑，采用递归方法设计的算法具有思路清晰、易于理解和分析的优点。

关键词：KMP；next 数组；递推；递归
中图分类号：TP301.6 文献标识码：A 文章编号：1673—629X(2009)06—0098—04

Research on Design of Next Fuction of KMP Algorithm

TANG Ya-ling
（School of Computer, Anhui University of Technology, Maanshan 243002, China）

Abstract: It usually calculates the next—array's value on the way of recurrence in textbooks of data—structure, which is difficulty and kernel of KMP algorithm. Introduces a new kind of algorithm, which calculates next—array's value by recursion, and finally it does discussion and analysis on some superior definitions of next—array. The experiment data shows its correctness of the recursion algorithm, and the recursion algorithm is also clearer and easier to understand on algorithm designing.

Key words: KMP; next—array; recurrence; recursion

0 引 言

字符串（以下简称串）是一种重要的数据的结构，是一种特殊的线性表，它的数据对象是字符集合。计算机在进行非数值运算时，大量使用串，串广泛应用于文字编辑、汇编和高级语言源程序的编译。在串处理中，串的模式匹配——子串定位是一种重要的串运算，即在给定源串查找特定子串（也称作模式串）的运算。串匹配算法中，经典的有简单模式匹配算法（BF 算法）和改进型 KMP 算法[1~6]。BF 算法在最坏的情况下其时间效率为 $0(n*m)$，其中 n 为主串的长度，m 为模式串的长度，KMP 算法是对简单模式匹配算法的改进型算法，其算法的主要核心是模式串 next 数组值的计算，利用模式串对应的 next 数组值，避免了匹配不成功时不必要的回溯，其时间效率在一般情形下是 $0(n+m)$，效率大大高于 BF 算法。

1 next 数组定义

KMP 算法避免不必要的回溯的关键在于，预先计算出模式串的 next 数组值，而模式串 next 数组值取决于模式串自身的特点，与被匹配的主串无关。一般地，设模式串 $t(t_1t_2t_3\cdots t_m, m\geqslant 1)$，$0<j\leqslant m$，模式中的每一个 t_j 都对应一个 next 值，这个 next 值仅依赖于模式 t 本身字符序列的构成，与主串无关。这里用 next[j] 表示 t_j 对应的 next 值，其值定义如下：

$$\text{next}[j]=\begin{cases}0 & \text{当 } j=1\\ \max\{k \mid 1<k<j \text{ 且“} t_1t_2\cdots t_{k-1}\text{”}\\ =\text{“}t_{j-k+1}t_{j-k+2}\cdots t_{j-1}\text{”}\\ 1 & \text{其它情形}\end{cases} \quad (1)$$

上面是 next 数组在当前主流的数据结构教材（如清华大学严蔚敏教授主编的《数据结构》，北京大学许卓群教授主编的《数据结构》）中普遍采用的定义方式[3,4]，同时对于 next 数组值的计算，均采用了递推（Recurrence）的思想来加以实现。

递推的思路是，先根据定义赋值 next[1]=0，通过 next[1] 计算出 next[2]，在得到 next[1]、next[2]、…next[$i-1$] 的值基础之上，计算 next[i] 的值，它递推的基础如下。

收稿日期：2008—09—21；修回日期：2008—12—01
基金项目：安徽省教育科研重点资助项目（2007jyxm054）；安徽省高校优秀青年人才基金（2009SQRZ076）
作者简介：汤亚玲（1974—），男，硕士，副教授，主要研究方向为智能化信息处理、数据挖掘及网络数据库系统。

设 next[$i-1$] = k, 则下式成立:

"$t_1\ t_2 \cdots t_{k-1}$" = "$t_{i-k+1}\ t_{i-k+2} \cdots t_{i-2}$"　　(2)

如 $t_k = t_{i-1}$, 则表明在模式串 t 中:

"$t_1\ t_2 \cdots t_k$" = "$t_{i-k+1}\ t_{i-k+2}\ \cdots\ t_{i-1}$"　　(3)

此时 next[i] = $k+1$, 即:

next[i] = next[$i-1$] + 1　　(4)

如 $t_k \neq t_{i-1}$, 则表明在模式串中

"$t_1\ t_2 \cdots t_k$" ≠ "$t_{i-k+1}\ t_{i-k+2} \cdots t_{i-1}$"　　(5)

此时将求 next[i] 函数值的问题看成是一个模式匹配问题, 模式串 t 既是主串又是模式串, 而当前在匹配的过程中, 已有(2)式成立, 则当 $t_k \neq t_{i-1}$时应将模式向右滑动, 使得第next[k] 个字符和"主串" 中的第 $i-1$ 个字符相比较。若 next[k] = k', 且 $t_{k'} = t_{i-1}$, 则说明在主串中第 i 个字符之前存在一个最大长度为k'的子串, 使得:

"$t_1\ t_2 \cdots t_{k'}$" = "$t_{i-k+1}\ t_{i-k+2} \cdots t_{i-1}$"　　(6)

因此: next[i] = next[k] + 1　　(7)

同理若 $t_{k'} \neq t_{i-1}$, 则将模式继续向右滑动至使第 next[k'] 个字符和 t_{i-1} 对齐, 依此类推, 直至 t_{i-1} 和模式中的某个字符匹配成功或者不存在任何 k'($1 < k' < k < \cdots < i-1$) 满足(6), 则: next[i] = 1。

通过递推思想得到的计算 next 数组值算法用 C 语言描述如下(算法 1):

```
void GetNext_Recurrence (char *t, int next[], int L)
/*L 表示模式串的长度, 字符串 t 从数组下标 1 开始存储 */
/*求模式 t 的 next 值并存入 next 数组中 */
{ int i=1, j=0;
next[1]=0;
while(i<t[0])
{
if(j=0||t[i]=t[j])
{++i;
++j;
next[i]=j;
}
else
j=next[j];
}/*end while*/
}/*end GetNext_Recurrence */
```

通过先计算 next[1] 的值, 找出 next[$i-1$] 与 next[i] 值之间的关系, 然后在 next[1] 值的基础之上, 逐步求出 next[2] 、next[3] 、next[4] …next[m] 的值。此种方法优点是简单易行, 计算效率高, 但技巧性强, 编程思路不太清晰, 比较难以理解, 在当前的数据结构的教材中基本上都采用递推的思想编写了相应的算法[3,4,5]。

2　next 数组的递归算法

在计算理论上, 递推和递归的计算方法有着密切的联系, 如果考察基于自然数集合的计算问题, 递归可以看成是递推的一个逆过程[7~9]。递归是从一个较大的自然数 N 向一个较小的自然数进行计算, 而递推是在较小自然数子集计算结果的基础之上, 向较大自然数 N 计算和推进的; 并且, 在计算机内部解释执行上, 递归算法最终还是系统栈的支持下通过递推方式来计算实现的[7,8,10]。两种不同的计算方式对应了两种不同的算法设计思想。从逻辑角度来看, 递归更便于理解, 编程的思路也更为清晰, 从效率上, 递推直接用循环实现, 其计算效率高于递归[3,5,10]。

理解了计算 next 数组递推的过程之后, 逆置 next 数组的递推计算过程, 即把计算对应自然数 N 的函数 $F(N)$ 问题转化为求解 $F(N-1)$ 的问题, 把计算 $F(N-1)$ 转化为计算 $F(N-2)$, 直到某个特定自然数 $n0$, $F(n0)$ 可解, 即得到求解问题 $F(N)$ 的递归计算方法。

对于 next[i] 与 next[$i-1$], 先假设 next[1] 、next[2] …next[$i-1$] 已经通过递归计算得到, 如果 t[next[$i-1$]] = t[$i-1$] , 则next[i] = next[$i-1$] + 1; 否则, 回溯, 考察 t[next[next[$i-1$]]] 是否等于 t[$i-1$] , 如果相等, 则 next[i] 的值应该为 next[next[$i-1$] + 1, 反之如果 t[next[next[$i-1$]]] 不等于 t[$i-1$] , 则判断 t[next[next[next[$i-1$]]]] 是否等于 t[$i-1$] , 如此一直下去, 直至存在某个 k = next[…next[$i-1$] …] > 0, 使得 t[k] = t[$i-1$] , 此时 next[i] = $k+1$, 否则当 k 等于 0 时, 则 next[i] 值为 1 ((1)式定义中的其它情形)。

其次, 考虑到递归的结束条件: next[1] = 0;

根据上面的分析, 写出如下的 C 语言描述的递归算法(算法 2)。

```
void GetNext_Recursion(char *t, int next[], int L)
{/*参数 t 为模式串, L 为模式串的长度, 算法结束时, 字符串 t 的 next 数组值保存在数组 next[] 中, 从下标为 1 开始存储 */
  if(L=1){next[1]=0;
    return;
   }/*L=1 时, 递归出口 */
  GetNext_Recursion(t, next, L-1);/*递归求 next[L-1], 为求 next[L] 作准备 */
  int k=next[L-1];
  while(true)/*循环直到 next[L] 值计算完毕 */
  { if(t[k]==t[L-1]){next[L]=k+1; /*满足 if 语句的条件, next[L]=k+1 */
    return;
```

```
    }
  k= next[ k] ; / *回溯 */
  if( k=0){ next[ L] = 1; / *不存在最大相等的前缀和后缀子串, next[ L] 赋值 1 */
    return;
    }
  }/ *end while */
  }/ *end GetNext_Recursion */
```

运用算法 GetNext_ Recursion 对字符串“abaabcac”, 求出的 next 数组值为 0, 1, 1, 2, 2, 3, 1, 2 与清华大学数据结构(严蔚敏版)中的递推算法 get_next 得到的结果是一致的[3]。

3 next 数组定义的改进及实现

有一些数据结构教材中对模式串 t 的 next 数组值的定义作了如下的改进, 如文献[5] 中 P101 给出的 next 数组的定义如下:

$$\text{next\_}r[j]=\begin{cases}0 & \text{当 } j=1\\ \max\{k \mid 1<k<j \text{ 且“} t_1t_2\cdots t_{k-1}\text{”}\\ =\text{“}t_{j-k+1}t_{j-k+2}\cdots t_{j-1}\text{”}\\ 1 & \text{当不存在上面的 } k \text{ 且 } t_1\neq t_j\\ 0 & \text{当不存在上面的 } k \text{ 且 } t_1=t_j\end{cases} \tag{8}$$

此定义讨论了这样一种特殊情形, 当模式串 t 中某个字符 t_j 前不存在真子串“$t_1\ t_2\cdots t_{k-1}$”=“$t_{j-k+1}\ t_{j-k+2}\cdots t_{j-1}$”时, 如 $t_1=t_j$, 则 next[j] = 0, 否则 next[j] = 1。作了这样的改进之后, 如果求出了某个 $t_j(j>1)$ 对应的 next 值为 0, 因为此时 $t_1=t_j$; 模式匹配时, 当 t_j 与主串中某个字符 s_i 不相等时, 可以直接开始用 t_1 与 s_{i+1} 进行新一轮的匹配, 而此时如果采用(1)式中的定义, 则 next[j] 的值等于 1, 将会用 t_1 与 s_i 进行新一轮的匹配; 所以, 采用(8)式中定义 next 数组值的方式避免了一次不必要的匹配过程。

对于(8)式定义的 next 数组, 其计算方法只需在(1)式定义对应的算法上稍作修改, 主要是考虑到(8)式定义中情形四: 当不存在上面的 k, 且 $t_1=t_j$ 这种特殊情形就可以了; 具体算法实现上, 可以看出算法 1 对于 next 数组值的计算, 除 next[1] 的值为 0 外, 其余字符对应的 next 值都是大于 0 的, 因为在算法 1 中的 while 循环中有这样几行代码:

```
++i;
++j;
next[ i] = j;
```

此处 next[i] = j, 是算法中除 next[1] = 0 赋值语句外唯一的对 next[i] 赋值的语句, 而在对 next[i] 赋值 j 前执行了 ++ j; 同时 j 能取得的最小值是通过回溯得到的, 并且其最小值是 0, 所以 next[i] ($i>1$) 的最小值是 1; 因此必须考虑到对于第字符 t[i], 不存在:“$t_1\ t_2\cdots t_{k-1}$”=“$t_{i-k+1}\ t_{i-k+2}\cdots t_{i-1}$”, 也就是字符 t[i] 前不存在相等的非空最大前缀和后缀子串时, 如果 t[1] = t[i], 则 t[i] = 0, 否则 t[i] = 1; 根据上述思想, 可以写出对应定义(8)的改进型计算 next 数组的算法(算法 3), 如下所示:

```
void GetNext_ Revise ( char *t, int next[] , int L)
/ *L 表示模式串 t 的长度, t 从下标 1 开始存储 */
    { int i= 1, j= 0;
    next[ 1] = 0;
    int flag_ once= 1; / /第一次进入循环标记置 true
    while( i< L)
    { if( j= 0 && flag_ once){ next[ ++ i] = ++ j; flag_ once= 0; }
    if( j= 0 && ! flag_ once) if( t[ i+ 1] = t[ 1] ){ next[ ++ i] = j++; }
    else next[ ++ i] = ++ j; / * 定义(8)中的情形 3 和情形 4 */
    if( t[ i] = t[ j] )
    {
    ++ i;
    ++ j;
    next[ i] = j;
    }
    else j= next[ j] ;
    }/ *end while */
}/ *end GetNext_ Revise */
```

上面的算法中用 flag_ once 表示是否是第一次进入循环, flag_ once 值为 1 时表示第一进入 while 循环, flag_ once 为 0 时表示不是第一次进入 while 循环。当第一次进入 while 循环时, j 的值不会回溯, 因此在 while 循环中, 当 flag_ once = 1 && j = 0 成立时, 执行 next[++ i] = ++ j; 而当 flag_ once = 0 && j = 0 && t[i+1] = t[1] 成立时, 执行 next[++ i] = j++。

清华大学版数据结构对于(1) 式的定义[3], 考虑到了这样一种情形, 当计算得到 next[i] 的值等于 j 时, 如果 t[i] 的值等于 t[j], 则将 next[j] 的值赋给 next[i], 即文献[3] 中定义模式串 t 的 next_ val 数组, 如下面(9) 式所示[3]:

$$\text{next}[j]=\begin{cases}0 & \text{当 } j=1\\ \max\{k \mid 1<k<j \text{ 且“} t_1t_2\cdots t_{k-1}\text{”}\\ =\text{“}t_{j-k+1}t_{j-k+2}\cdots t_{j-1}\text{”}\\ 1 & \text{其它情形, 如 } t_j=t_{\text{next}[j]},\\ & \text{则 next}[j]=\text{next}[\text{next}[j]]\end{cases} \tag{9}$$

此时 next[j] 的定义, 也称作 nextval[j]。(9) 式的

定义,给出了这样一种清形,当 next[j] 的值为 k 时,此时如果 $t_j = t_k$,则可将 next[k] 的赋值给 next[j] ,这样在模式匹配时,最大程度地避免了不必要的回溯;同时(9)式的定义包含了(8)式定义中的第四种情形,因此从定义上考虑,(9)式对模式串的 next 数组值的定义是最优的。同时,只需对算法 GetNext_ Recursion 稍作修改,使之符合定义(9),如算法 Get_ NextVal 所示(C 语言描述)(算法 4)。

```
void Get_NextVal(char *t, int next[], int i)
{/*求解模式 t 的 nextval 数组的递归算法*/
GetNext_Recursion(t, next, i);/*调用算法 GetNext_Recursion*/
for(int j=2; j<=i; j++)
    { int k=next[j];
    if(t[j]==t[k])next[j]=next[k];/*对模式 t 的 next 数组值进行修正,得到 nextval 值*/
    }
}
```

表 1 给出了用文中算法 GetNext_ Recursion、算法 Get_ NextVal 和算法 GetNext_ Revise 对模式串“abcaababc”求解 next 数组值的计算结果。

表 1　next 数组计算对照表

模式 t	a	b	c	a	a	b	a	b	c
下标	1	2	3	4	5	6	7	8	9
next[]	0	1	1	1	2	2	3	2	3
nextval[]	0	1	1	0	2	1	3	1	1
next_ rev[]	0	1	1	0	2	2	3	2	3

其中, next[] 表示基本 next 数组定义(1)对应的计算结果, nextval[ ] 、和 next_ rev[ ] 分别表示文献[5] 和文献[3] 对于 next 数组的改进型定义(定义(8)、定义(9))所对应的算法的计算结果。特别的,对于模式串 t 中的第四个字符 a,根据定义(1)计算得到的 next 值为 1,而由定义(8)和定义(9)计算得出的 next 值则为 0,其原因就是定义(1)没有考虑到定义(8)中所描述的第四种特殊情形,而 nextval 的定义(定义(9))则涵盖了此种情形。此外,从整个模式串 t 的 next 数组值计算结果上,可以看出下式成立:

$$\text{nextval}[i] \leqslant \text{next\_rev}[i] \leqslant \text{next}[i],\ i > 0 \quad (10)$$

因此,采用 nextval 数组执行 KMP 算法回溯的次数最少,模式匹配的效率在三者中是最高的。

4　结束语

字符串作为一种特殊的线性表,由于它要处理的一般是非数值对象,如程序的编译,文本性质的数据编辑和查询等,其算法具有其特殊性。字符串算法中,模式匹配的 KMP 算法是一个经典算法,也是一个具有一定难度的算法[2];同时,KMP 算法中的 next 数组值的计算是 KMP 算法的核心。通过计算 next 数组值,避免了模式匹配时不必要的字符比较,大大提高了匹配的效率。当前,一般的算法教科书都给出采用递推思想设计的 next 数组的计算算法,以递推方法编写的 next 数组计算算法效率高,但思路没有递归清晰。在文中,给出了一种新的计算 next 数组和 nextval 数组的递归算法,便于对 next 数组和 nextval 数组计算方法的学习,从递推和递归两个不同的角度和计算过程去理解算法;同时讨论了对于 next 数组的三种定义方式,并比较了这三种定义中的各自的特点,给出了其相应的计算算法和要点分析。

参考文献:

[1] 甘学士. 改进的模式匹配算法及在入侵检测中的应用[J]. 计算机技术与发展, 2006, 16(7): 150－152.

[2] 陆建军. KMP 模式匹配算法在串行通讯中的应用[J]. 工业控制计算机, 2005, 18(2): 30－32.

[3] 严蔚敏, 吴伟民. 数据结构[M]. 北京: 清华大学出版社, 2008: 81－84.

[4] 许卓群, 张乃孝, 杨冬青, 等. 数据结构[M]. 北京: 高等教育出版社, 1993: 90－91.

[5] 秦　锋, 汤文兵, 章曙光, 等. 数据结构[M]. 合肥: 中国科学技术大学出版社, 2007: 101－102.

[6] 刘玉龙, 刘　啸. 一种模式匹配快速算法[J]. 计算机科学, 2008, 35(1): 219－220.

[7] Milner R. Functions as processes[M]//Mathematical Structures in Computer Science. Berlin / Heidelberg: Springer, 1992: 167－180.

[8] Sangiorgi D. An investigation into functions as processes[C]//In: Proc. Math. Foundations of Program Semantics' 93. Berlin / Heidelberg: Springer, 1993: 143－159.

[9] Hamilton. 数学家的逻辑[M]. 北京: 科学出版社, 1989.

[10] 汤亚玲, 崔志明. 基于 C++ 递归方法在应用问题中的设计与实现[J]. 微机发展(现更名: 计算机技术与发展), 2003, 13(8): 90－92.

(上接第 97 页)

2007.

[5] 张　舸, 刘利强, 周细义. 快速层次移动 IPv6 切换性能分析及优化[J]. 无线电通信技术, 2008(4): 25－26.

[6] 孔祥松, 贾卓生. 移动 IPv6 的切换技术[J]. 计算机工程与设计, 2006, 27(8): 1453－1455.

[7] 沈庆伟, 张　霖. 基于隧道的 IPv4/IPv6 过渡技术分析[J]. 计算机技术与发展, 2007, 17(5): 176－178.

[8] 官　俊, 陈　健, 陈　炯. 一种基于转交地址池的层次移动 IPv6 改进协议[J]. 计算机应用, 2006, 26(2): 300－302.