Data Structures and Algorithms (4)

Instructor: Ming Zhang

Textbook Authors: Ming Zhang, Tengjiao Wang and Haiyan Zhao

Higher Education Press, 2008.6 (the "Eleventh Five-Year" national planning textbook)

https://courses.edx.org/courses/PekingX/04830050x/2T2014/

Outline

- The Basic Concept of Strings
- The Storage Structure of Strings
- The Implementation of Strings' Operations
- Pattern Matching for Strings
  - Naïve algorithm
  - **KMP algorithm**

Match without Backtracking

- In matching process , once p_j is not equal to t_i , that's:

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

But $p_j \neq t_i$

  - Which character p_k should be used to compare with t_i in p ?
  - Determine the number of right-moving of digits
  - It is clear that $k < j$, and when j changes, k will change too
- Knuth-Morrit-Pratt (KMP) algorithm
  - The value of k only depends on pattern P itself, it doesn't have relations with target string T

KMP algorithm

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

‖ ‖ ‖ ‖ ‖ ×

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

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

naïve for next trip p_0 p_1 … $p_{j\text{-}2}$ $p_{j\text{-}1}$

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

You can immediately conclude:

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

(naive matching) Next trip will not match, jump over

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

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

P = a b c d e f f X

X a b c d e f f

It's same , if $p_0\ p_1\ \ldots p_{j-3} \neq p_2\ p_3 \ldots p_{j-1}$

It doesn't match in the next step , because

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

Until , "k" appears (the length of head and tail string) , it makes

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

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

Pattern moves j-k bits right

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

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

String feature vector: N

Assume that P is consisting of m chars , noted as

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

Use Feature Vector N to represent the distribution characteristic of pattern P, which is consisting of m feature numbers $n_0 \ldots n_{m-1}$, noted as

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

N is also called as the next array, each element n_j corresponds to next[j]

Feature vector for String N： Constructor

- The feature number of the j-th position of P is n_j, The longest head and tail string is k
  - Head string： $p_0 \ p_1 \ \ldots \ p_{k-2} \ p_{k-1}$
  - Tail String： $p_{j-k} \ p_{j-k+1} \ \ldots \ p_{j-2} \ p_{j-1}$

$$
\text{next}[j] = \begin{cases} -1, & \text{j==0} \\ \max\{k: 0 < k < j \ \& \ P[0\ldots k\text{-}1] = P[j\text{-}k\ldots j\text{-}1]\}, & \text{If K exists} \\ 0, & \text{otherwise} \end{cases}
$$

4.3 Pattern Matching for Strings

	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 (should be 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	X a	a	b	a	a	a	a	c						
				a	a	a	a	X 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

Miss it !

Example for Pattern Matching for Strings

	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 matching algorithm

```
int KMPStrMatching(string T, string P,  int *N, int start) {
   int j= 0;                    // subscript variable of pattern
   int i = start;               // subscript variable of target string
   int pLen = P.length( );      // length of pattern
   int tLen = T.length( );      // length of target string
   if (tLen - start < pLen)     // if the target is shorter than
                                // the pattern, matching can not succeed
       return (-1);
   while ( j < pLen  &&  i < tLen)  {// repeat comparisons to match
      if ( j == -1  ||  T[i] == P[j])
         i++,  j++;
      else j = N[j];
   }
   if (j >= pLen)
      return (i-pLen);                    // be careful with the subscript
   else return (-1);
}
```

Algorithm Framework for Seeking the Feature Value

- Feature value n_j ($j>0$, $0\leq n_{j+1}\leq j$) is recursively defined , defined as follows：

1. $n_0 = -1$, for n_{j+1} with $j>0$, assume that the feature value of the previous position is n_j , let $k = n_j$ ；
2. When $k\geq0$ and $p_j \neq p_k$, let $k = n_k$; let Step 2 loop until the condition is not satisfied.
3. $n_{j+1} = k + 1$; // $k == -1$ or $p_j == p_k$

Feature vector of String: N — non-Optimized version

```
int findNext(string P) {
    int j, k;
    int m = P.length( );        // m is the length of pattern P
    assert( m > 0);             // if m=0, exit
    int *next = new  int[m];    // open up an integer array in dynamic storage area.
    assert( next != 0);         // if opening up integer array fails , exit
    next[0] = -1;
    j = 0;  k = -1;
    while (j < m-1) {
        while (k >= 0 && P[k] != P[j])// ff not equal, use kmp to look for head and
tail substring
            k = next[k];               // k recursively looking forward
         j++;   k++;  next[j] = k;
    }
    return next;
}
```

Seeking feature vector N

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

j= 9 k = 0

Head substring→ a

Head substring→ a a

Head substring→ a a a

Head substring→ a

Head substring→ a a

Head substring→ a a a

Head substring→ a a a a

Pattern moves j-k bit right

$$t_{i-j} \quad t_{i-j+1} \quad t_{i-j+2} \quad \ldots \quad t_{i-k} \quad t_{i-k+1} \quad \ldots \quad t_{i-1} \quad t_i$$

$$\| \; \| \; \| \qquad \| \quad \| \qquad \| \qquad \times$$

$$p_0 \quad p_1 \quad p_2 \quad \ldots \quad p_{j-k} \quad p_{j-k+1} \quad \ldots \quad p_{j-1} \quad p_j$$

$$\| \quad \| \qquad \| \qquad ?$$

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

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

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

KMP Matching

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

Target *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

This line is redundant *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*

P[3]==P[0], P[3] ≠ T[4], one more time comparison is redundant

Feature vector of String: N — Optimized version

```
int findNext(string P) {
    int j, k;
    int m = P.length( );                 // m is the length of pattern P
    int *next = new  int[m];             // open up an integer array in dynamic storage area.
    next[0] = -1;
    j = 0;  k = -1;
    while (j < m-1) {                    // if  j<m, the code will be out of the border
        while (k >= 0 && P[k] != P[j])//if not equal, use kmp to look for head and tail substring
            k = next[k];                 // k looks forward recursively
        j++;   k++;
        if (P[k] == P[j])
            next[j] = next[k];           // finding value k isn’t affected by the optimization
        else next[j] = k;                // no optimization if you cancel the “if” judgment
    }
    return next;
}
```

Comparison of next arrays

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	Non-optimized version
$p_k{=}{=}p_j$?		≠	≠	==	≠	==	≠	==	==	
next[j]	-1	0	0	-1	1	0	2	0	0	Optimized version

Time Analysis for the KMP algorithm

- The statement “j = N[j];” in the loop will not execute more than n times. Otherwise,
  - Because every time the statement “j = N[];” is executed, j decreases inevitably (minus at least by one)
  - Only “j++” can increase j
  - Thus, if the statement “j==N[j]” is executed more than n times, j will become smaller than -1. It’s impossible (sometimes j becomes -1, but it will be increased by 1 and becomes 0 immediately)
- The time for constructing the N array is O(m)

Therefore, the time complexity of KMP is O(n+m)

Summary: Single-matching algorithm

Algorithm	*Time efficiency for preprocessing*	*Time efficiency for matching*
Naïve matching algorithm	0	Θ(n m)
KMP	Θ(m)	Θ(n)
BM	Θ(m)	Best (n/m), Worst Θ(nm)
shift-or, *shift-and*	Θ(m+|Σ|)	Θ(n)
Rabin-Karp	Θ(m)	Average (n+m), WorstΘ(nm)
Finite state automaton	Θ(m |Σ|)	Θ(n)

Different Versions of the Feature Vector

If match fails on the j-th character, let j=next[j]

$$
next[j] = \begin{cases} -1, & \text{If } j==0 \\ \max\{k: 0 < k < j \ \ \&\& \ P[0\ldots k\text{-}1] = P[j\text{-}k\ldots j\text{-}1]\ \}, & \text{If K exists} \\ 0, & \text{else} \end{cases}
$$

If match fails on the j-th character, let j=next[j-1]

$$
next[j] = \begin{cases} 0, & \text{If } j==0 \\ \max\{k: 0 < k < j \ \ \&\& \ P[0\ldots k] = P[j\text{-}k\ldots j]\ \}, & \text{If K exists} \\ 0, & \text{else} \end{cases}
$$

Reference materials

- **Pattern Matching Pointer**
  - http://www.cs.ucr.edu/~stelo/pattern.html
- **EXACT STRING MATCHING ALGORITHMS**
  - http://www-igm.univ-mlv.fr/~lecroq/string/
  - Description and complexity analysis of pattern matching for strings and the C source code

Data Structures and Algorithms

Thanks

the National Elaborate Course (Only available for IPs in China)
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

Ming Zhang, Tengjiao Wang and Haiyan Zhao
Higher Education Press, 2008.6 (awarded as the "Eleventh Five-Year" national planning textbook)