

13C2 KMP算法：查询表

#数据结构邓神

制表查询

事先确定t

❖ 构造查询表 $\text{next}[0, m]$: 在任一位置 $P[j]$ 处失败之后, 将 j 替换为 $\text{next}[j]$

❖ 与其说是借助强大的记忆, 不如说是做好充分的预案

$T[i] \neq P[j]$

$T[i - n[j], i]$	X	$T(i, n)$
$T[i - j, i]$		
$P[0, j]$	Y	$P(j, m)$
$P[0, n[j]]$?	$P(n[j], m)$

主算法

KMP算法

```
int match( char * P, char * T ) {  
    int * next = buildNext(P); //构造next表  
    int n = (int) strlen(T), i = 0; //文本串指针  
    int m = (int) strlen(P), j = 0; //模式串指针  
    while ( j < m && i < n ) //自左向右, 逐个比对字符  
        if ( 0 > j || T[i] == P[j] ) { //若匹配  
            i ++; j ++; //则携手共进  
        } else //否则, P右移, T不回退  
            j = next[j];  
    delete [] next; //释放next表  
    return i - j;  
}
```

$T[i] \neq P[j]$

$P[0, n(j)]$?	$P(n(j), m)$
$P[0, j]$	Y	$P(j, m)$
$T[0, i]$	X	$T(i, n)$

D. E. Knuth J. H. Morris V. R. Pratt

```
int match(char * P, char * T){  
    int * next = buildNext(P);  
    int m = (int)strlen(T);  
    int n = (int)strlen(P);  
    int i = 0, j = 0;
```

```

while (i < m && j < n){
    if (0 > j || T[j] == P[j]){
        i ++;
        j ++;
    }
    else {
        j = next[j];
    }
}
delete [] next;
return i - j;
}

```

那么如何去构造一个next 记住这个next表仅仅取决于模式串
在失配情况下更加简明，只要 $j = \text{next}[j]$; 并没有修改变量 i

实例



