11.串

(c4) KMP算法:构造next[]表

邓俊辉

deng@tsinghua.edu.cn

递推

- ❖ 根据已知的 next[0, j],如何高效地计算 next[j + 1]?
- ❖ 所谓 next(j) , 即是在 P[0, j) 中 , 最大 自匹配 的 真前缀 和 真后缀 的长度
- ❖故: next[j + 1] ≤ next[j] + 1

特别地,当且仅当P[j] == P[next[j]]时取等号

P[0, j) X P(j, m)

P[0, n(j)) X P(n(j), m)

j + 1

❖一般地, P[j] != P[next[j]]时, 又该如何得到 next[j + 1]?

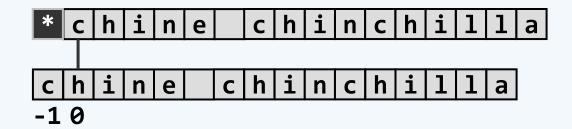
算法 ❖ next[j + 1]的候选者 P(j, m) P[0, j) X 依次应该是: P(n(j), m) P[0, n(j)) 1 + next[j] P[0, n(n(j)))1 + |next[|next[j]] P(n(n(j)), m) Z next[next[j] 1 + |next[P[0, n(n(n(j)))) X P(n(n(j))), m). . . ❖ 这个序列严格递减,且 必收敛于 1 + next[0] ≡ 0 P(-1, m)❖以上递推过程,即是P的 自匹配 过程,故只需对KMP框架略做修改...

```
实现)
```

```
❖int * <u>buildNext(</u> char * P ) { //构造模式串P的next[]表
     size_t m = strlen(P), j = 0; //"主" 串指针
     int * |N| = new int[m]; //next[]表
    int t = N[0] = -1; //模式串指针(P[-1])通配符)
    while (j < m - 1)
                                                        * P( -1, m ) = P[ 0, m )
       if ( 0 > t || P[j] == P[t] ) //匹配
           N[ ++j ] = ++t;
                                    P[0, n(n(j)))
                                                        X = P(n(n(j))), m)
        else //失配
                                                        \mathbf{Z} \mid P(n(n(j)), m)
                                        P[0, n(n(j)))
           t = N[t];
                                                       Y P( n(j), m )
                                          P[ 0, n(j) )
     return N;
                                             P[ 0, j ) X P( j, m )
                                                   Data Structures (Spring 2014), Tsinghua University
```



```
* c h i n e c h i n c h i l l a | c h i n c h i l l a | c h i n c h i l l a | c h i n c h i l l a | c h i n c h i l l a | c h i n c h i l l a | c h i n c h i l l a | c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n c h i n
```



```
* c h i n e c h i n c h i l l a

c h i n e c h i n c h i l l a

-100
```



