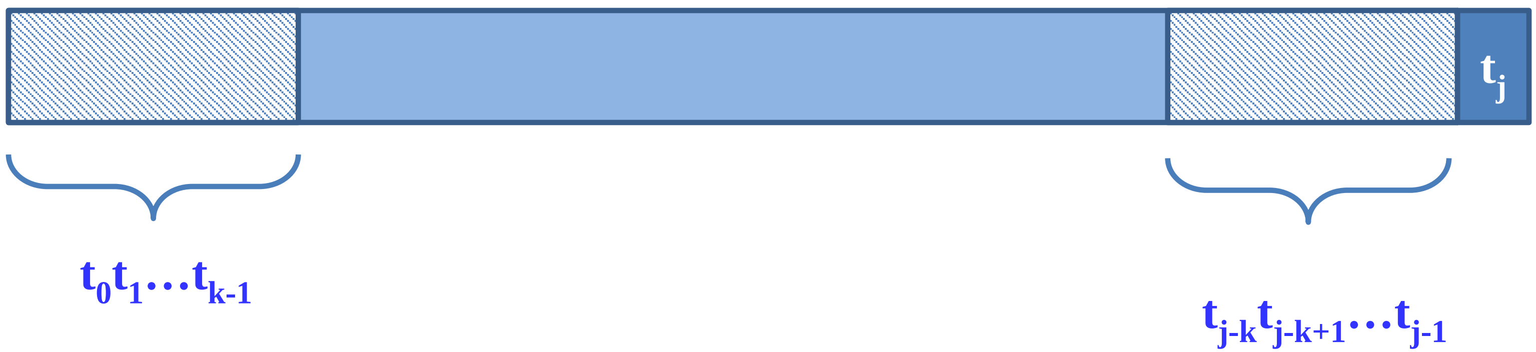
Knuth–Morris–Pratt Algorithm
KMP 算法

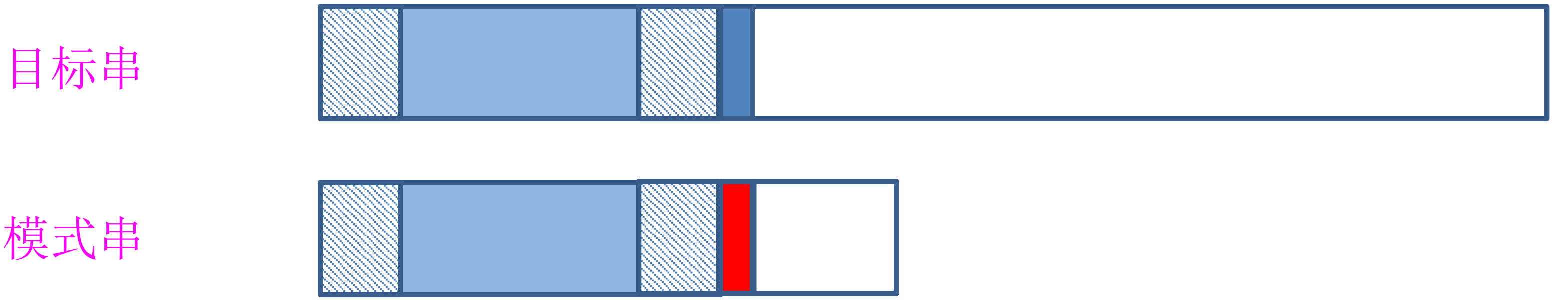
最大子串的概念

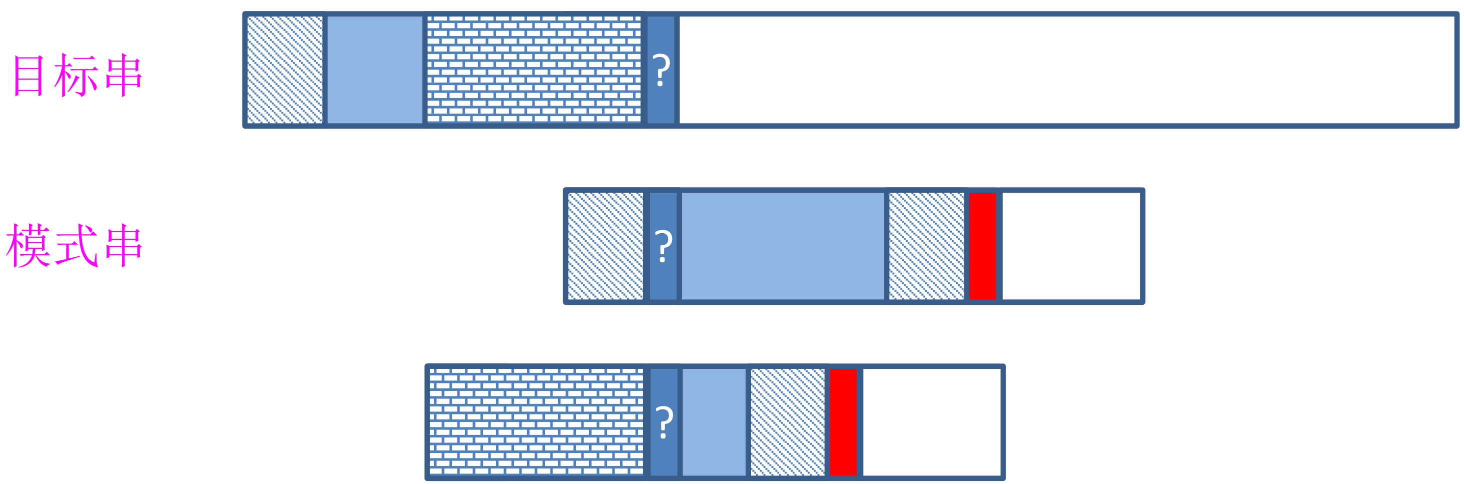
- 所谓**真子串**是指模式串t存在某个k（$0<k<j$），使得“$t_0t_1\ldots t_{k-1}$” = “ $t_{j-k}t_{j-k+1}\ldots t_{j-1}$ ”成立。



所有真子串中最长的称为**最大子串**，其长度k记为next[j]

j	0	1	2	3	4	5	6	7	8
t[j]	A	B	A	C	A	B	A	B	C
next[j]	-1	0	0	1	0	1	2	3	2


目标串
模式串


目标串
模式串
?
?
?

j	0	1	2	3	4	5	6
t[j]	A	B	C	D	A	B	D
next[j]	-1	0	0	0	0	1	2

```
i: 01234567890123456789012
S: ABC_ABCDAB_ABCDABCDABDE
t: ABCDABD
j: 0123456
```

```
i: 01234567890123456789012
S: ABC_ABCDAB_ABCDABCDABDE
t:         ABCDABD
j:         0123456
```

```
i: 01234567890123456789012
S: ABC_ABCDAB_ABCDABCDABDE
t:    ABCDABD
j:    0123456
```

```
i: 01234567890123456789012
S: ABC_ABCDAB_ABCDABCDABDE
t:           ABCDABD
j:           0123456
```

```
i: 01234567890123456789012
S: ABC_ABCDAB_ABCDABCDABDE
t:     ABCDABD
j:     0123456
```

```
i: 01234567890123456789012
S: ABC_ABCDAB_ABCDABCDABDE
t:            ABCDABD
j:            0123456
```

```
i: 01234567890123456789012
S: ABC_ABCDAB_ABCDABCDABDE
t:                ABCDABD
j:                0123456
```

归纳起来，定义next[j]函数如下:

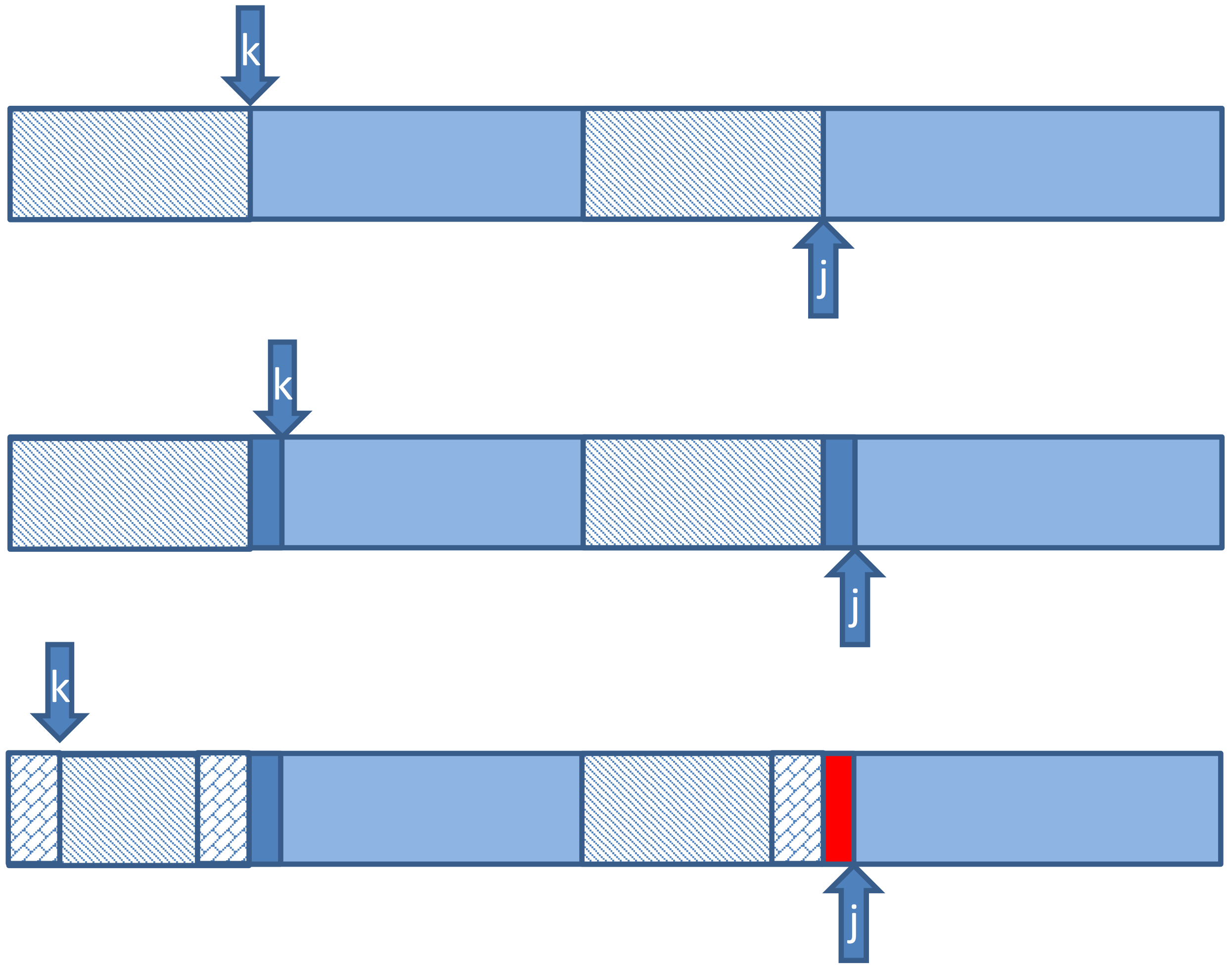
$$
\text{next}[j]=\begin{cases}\max\{k \mid 0<k<j,\text{且“}t_0 t_1 \ldots t_{k-1}\text{”=“}t_{j-k} t_{j-k+1} \ldots t_{j-1}\text{”}\} & \text{当此集合非空时} \\ -1 & \text{当}j=0\text{时} \\ 0 & \text{其他情况}\end{cases}
$$

t=“abab”对应的next数组如下:

j	0	1	2	3
t[j]	*a*	*b*	*a*	*b*
next[j]	-1	0	0	1

由模式串t求出next值：

```
void GetNext(SqString t,int next[])
{  int j,k;
   j=0;k=-1;next[0]=-1;
   while (j<t.length-1)
   {  if (k==-1 || t.data[j]==t.data[k])
      {   j++;k++;
          next[j]=k;
      }
      else k=next[k];
   }
}
```


k
j
k
j
k
j

KMP算法：

```
int KMPIndex(SqString s,SqString t)
{  int next[MaxSize],i=0,j=0;
   GetNext(t,next);
   while (i<s.length && j<t.length)
   {  if (j==-1 || s.data[i]==t.data[j])
      {  i++;
         j++;                      //i,j各增1
      }
      else j=next[j];              //i不变,j后退
   }
   if (j>=t.length)
      return(i-t.length);          //返回匹配模式串的首字符下标
   else
      return(-1);                  //返回不匹配标志
}
```

设主串s的长度为n，子串t长度为m。

在KMP算法中求next数组的时间复杂度为O(m)，在后面的匹配中因主串s的下标不减即不回溯，比较次数可记为n，所以KMP算法总的时间复杂度可以认为是O(n+m)。

设目标串s=“*aaabaaaab*”，模式串t=“*aaaab*”。KMP模式匹配过程。

求t的next：

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
next[j]	−1	0	1	2	3

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
next[j]	−1	0	1	2	3

0 1 2 3 4 5 6 7 8

s: *a a a b a a a a b*

t: *a a a a b*

0 1 2 3 4

失败：

i=3

j=3，*j*=next[3]=2

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
next[j]	−1	0	1	2	3

i=3
j=2

0 1 2 3 4 5 6 7 8

s: *a a a b a a a a b*

t: *a a a a b*

0 1 2 3 4

失败:
i=3
j=2，j=next[2]=1

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
next[j]	−1	0	1	2	3

i=3
j=1

0 1 2 3 4 5 6 7 8

s: *a a a b a a a a b*

t: *a a a a b*

0 1 2 3 4

失败:
i=3
j=1，j=next[1]=0

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
next[j]	−1	0	1	2	3

i=3
j=0

0 1 2 3 4 5 6 7 8
s: *a a a b a a a a b*

t: *a a a a b*
0 1 2 3 4

失败：
i=3
j=0，j=next[0]=−1

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
next[j]	−1	0	1	2	3

因为*j*=−1：
i++;
j++;

0 1 2 3 4 5 6 7 8
s: *a a a b a a a a b*

t: *a a a a b*
0 1 2 3 4

成功：
返回4

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
next[j]	−1	0	1	2	3

前面的匹配过程：

i=3
j=3
将s[i]与
t[3]匹配

i=3
j=2
将s[i]与
t[2]匹配

i=3
j=1
将s[i]与
t[1]匹配

i=3
j=0
将s[i]与
t[0]匹配

i=3
j=−1
将s[i+1]与
t[0]匹配

因为t[3]=t[2]=t[1]=t[0]='*a*'

是不必要的

i=3
j=3

i=3
j=−1

将next改为nextval：

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
next[j]	−1	0	1	2	3
nextval[j]	−1	−1	−1	−1	3

next[1]=0

t[1]=t[next[1]]=t[0]='*a*'

∴ nextval[1]=nextval[0]=−1

t[4]='*b*' ≠ t[next[4]]='*a*'

∴ nextval[4]=next[4]

- nextval[0]=−1
- 当t[*j*]=t[next[*j*]]时： nextval[*j*]=nextval[next[*j*]]
- 否则： nextval[*j*]=next[*j*]

用nextval取代next，得到改进的KMP算法。

使用改进后的KMP算法示例：

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
nextval[j]	−1	−1	−1	−1	3

0 1 2 3 4 5 6 7 8

s: *a a a b a a a a b*

t: *a a a a b*

0 1 2 3 4

失败：

i=3

j=3，*j*=nextval[3]=−1

j	0	1	2	3	4
t[j]	*a*	*a*	*a*	*a*	*b*
nextval[j]	−1	−1	−1	−1	3

因为*j*=−1：
i++;
j++;

0 1 2 3 4 5 6 7 8
s: *a a a b a a a a b*

t: *a a a a b*
0 1 2 3 4

成功：
返回4

改进后的KMP算法进一步提高模式匹配的效率。

由模式串t求出nextval值：

```
void GetNextval(SqString t,int nextval[])
{  int j=0,k=-1;
   nextval[0]=-1;
   while (j<t.length)
   {  if (k==-1 || t.data[j]==t.data[k])
      {  j++;k++;
         if (t.data[j]!=t.data[k])
            nextval[j]=k;
         else
            nextval[j]=nextval[k];
      }
      else
         k=nextval[k];
   }
}
```

修改后的KMP算法:

```
int KMPIndex1(SqString s,SqString t)
{  int nextval[MaxSize],i=0,j=0;
   GetNextval(t,nextval);
   while (i<s.length && j<t.length)
   {  if (j==-1 || s.data[i]==t.data[j])
      {  i++;
         j++;
      }
      else
         j=nextval[j];
   }
   if (j>=t.length)
      return(i-t.length);
   else
      return(-1);
}
```

数据结构经典算法的启示

BF算法

↓ 利用模式串中部分匹配信息

KMP算法

课堂练习：目标串s=“abcaabbcaaabababaabca”

模式串t=“babab”

求KMP算法匹配过程

求改进的KMP算法的匹配过程

作业：第4章，5，6

——本讲完——