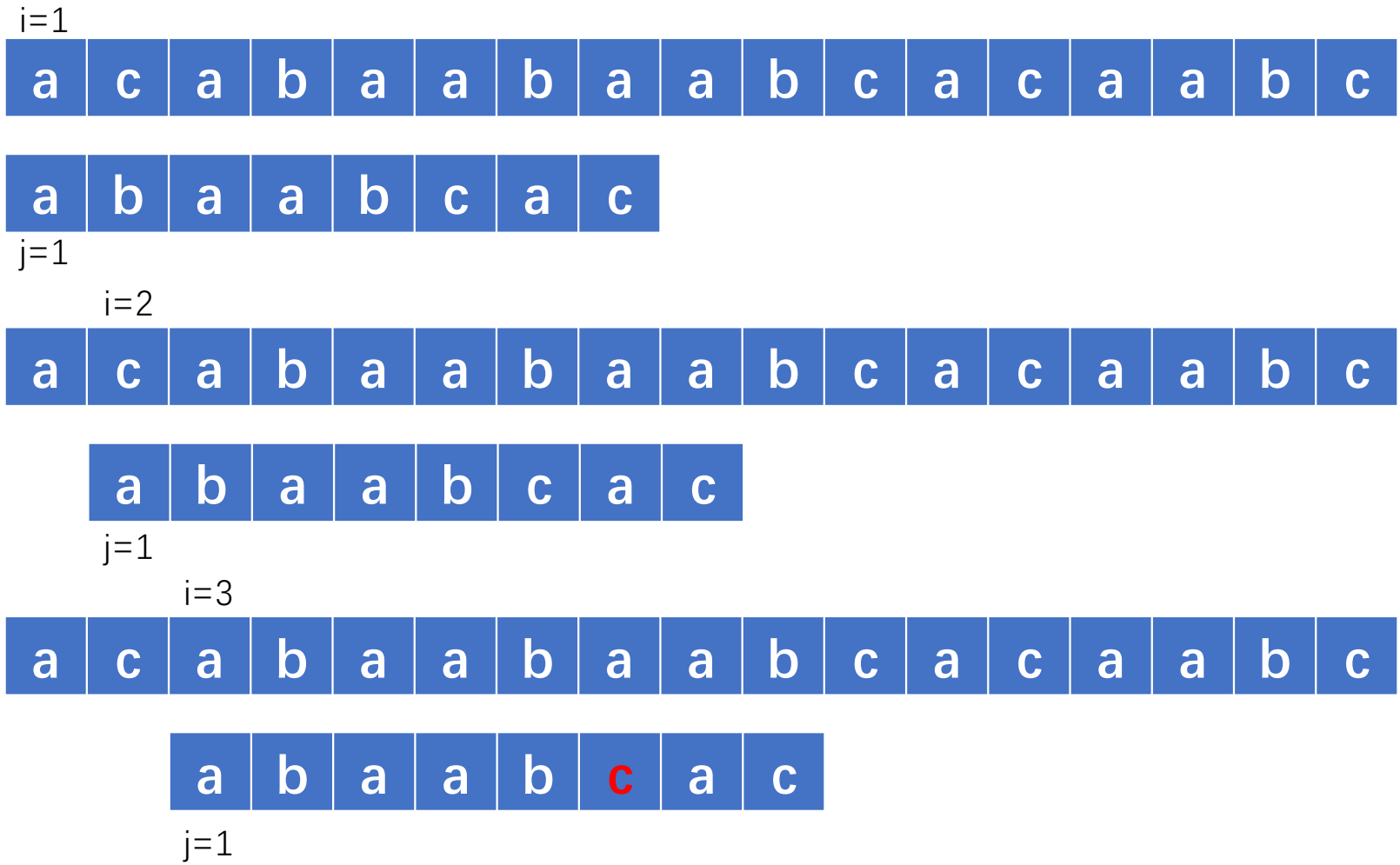
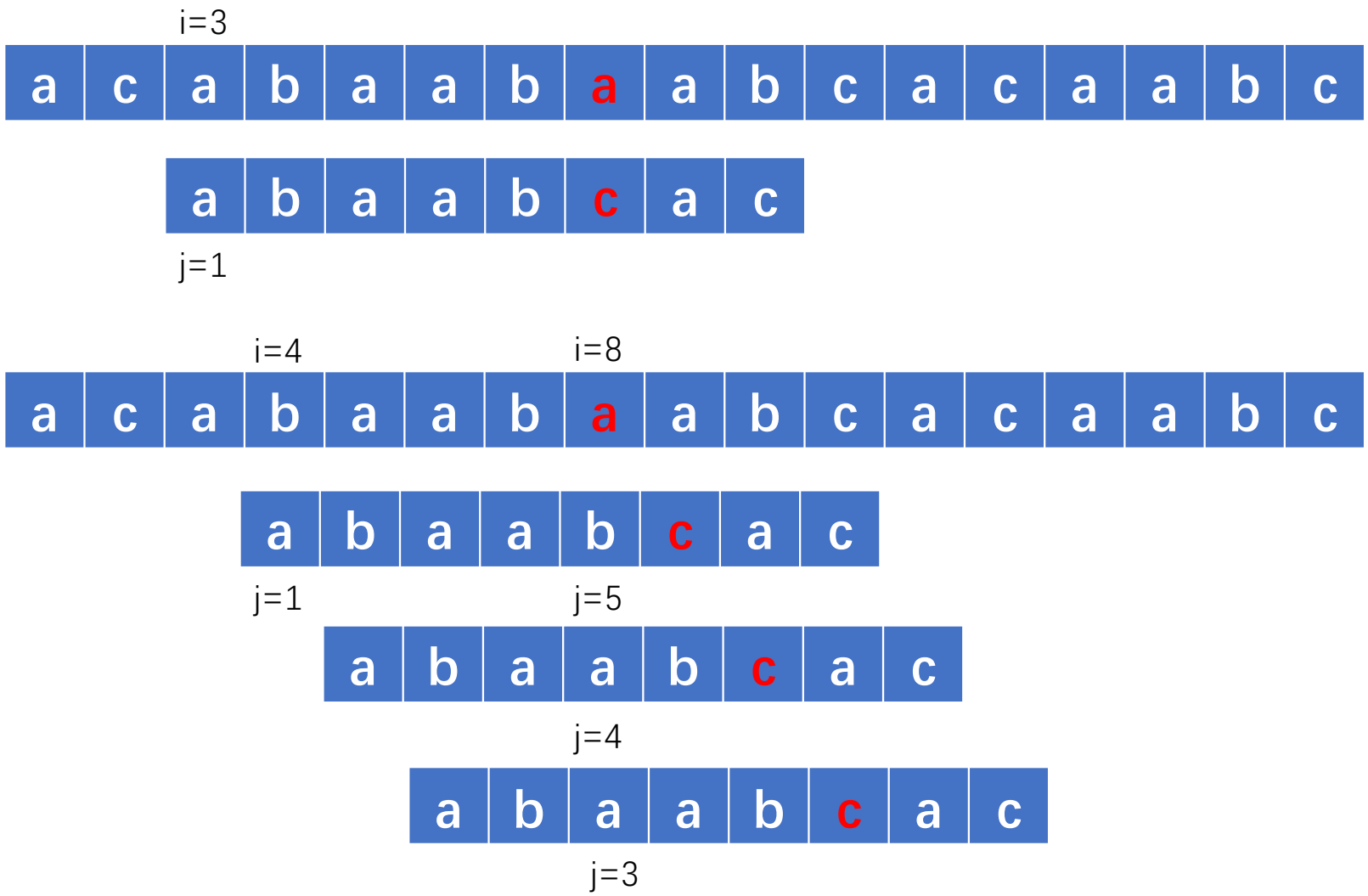
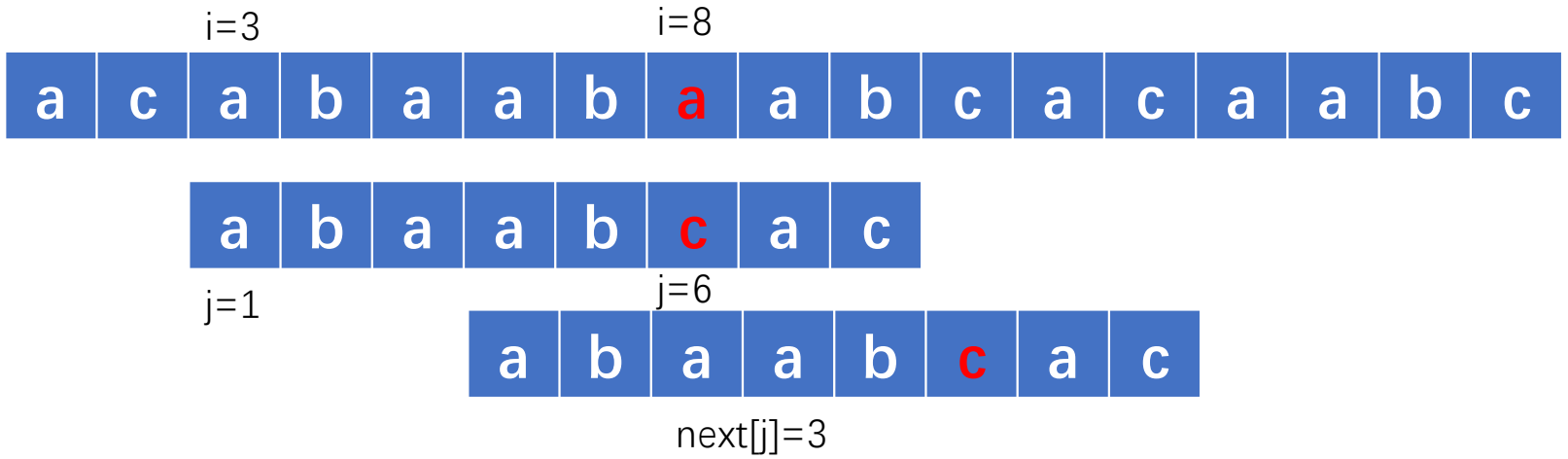
KMP算法


i=1
a c a b a a b a a b c a c a a b c
a b a a b c a c
j=1
i=2
a c a b a a b a a b c a c a a b c
a b a a b c a c
j=1
i=3
a c a b a a b a a b c a c a a b c
a b a a b c a c
j=1


i=3
a c a b a a b a a b c a c a a b c
a b a a b c a c
j=1
i=4
i=8
a c a b a a b a a b c a c a a b c
a b a a b c a c
j=1
j=5
a b a a b c a c
j=4
a b a a b c a c
j=3



- 找到一个无回溯的算法，即i指针不回溯
- 问题是：模式串右滑多远可以继续和i指针比较
- 与主串没有关系，是模式串P内的部分匹配问题，仅依赖于P的前m个字符与失效位置前m个字符的构成（m=k-1）
- 令next[j]为与失效位置j对应的k值，即滑动后从哪个字符开始比较，那么P中前k-1个字符与失效位置j前的k-1个字符是相同的

KMP算法：

```
int Index_KMP(SString S, SString T, int pos) {
        i=pos; j=1;
        while(i<=S[0] && j<=T[0]) {
                if (j==0 || S[i]==T[j]) {++i; ++j}
                else j=next[j];
        }
        if (j>T[0]) return i-T[0];//T[0]为T长度
        else return 0;
}//Index_KMP
```

- 如何计算next[j]？
- 令next[j]为与失效位置j对应的k值，即滑动后从哪个字符开始比较，那么P中前k-1个字符与失效位置j前的k-1个字符是相同的

$$\text{“}p_1p_2\cdots p_{k-1}\text{”}=\text{“}p_{j-k+1}p_{j-k+2}\cdots p_{j-1}$$

$$\text{next}[j]=\begin{cases}0 & \text{当 } j=1 \text{ 时}\\ \text{Max}\{k \mid 1<k<j \text{ 且“}p_1p_2\cdots p_{k-1}\text{”}=\text{“}p_{j-k+1}p_{j-k+2}\cdots p_{j-1}\text{”}\} & \\ 1 & \text{其它情况}\end{cases}$$

a	b	a	a	b	c	a	c

$$next[j]=\begin{cases}0 & \text{当 } j=1 \text{ 时}\\ Max\{k \mid 1<k<j \text{ 且“} p_1p_2\ldots p_{k-1}\text{”}=\text{“}p_{j-k+1}p_{j-k+2}\ldots p_{j-1}\text{”}\} & \\ 1 & \text{其它情况}\end{cases}$$

j	1	2	3	4	5	6	7	8
next[j]	0	1	1	2	2	3	1	2

计算next函数值算法---在S中找前缀和后缀相同的最大真子串，目标串和模式串是同一个串

假设next[j]=k，j是失效位置

- 如果 p（j）=p（k），则next（j+1）=k+1=next（j）+1
- 如果 p（j）！=p（k），匹配是否存在一个比长度k更小的匹配串，由于前k个位置已经匹配，等同于匹配k’ =next[k]
- 如果p（j）=p（k’ ）,则next（j+1）=k’ +1=next（k）+1
- 以此类推，一直到找不到匹配的更小子串，则next（j+1）=1

计算next函数值算法---在S中找前缀和后缀相同的最大真子串，目标串和模式串是同一个串

```
void get_next(SString S, int next[ ]) {
        i=1; next[1]=0; j=0;
        while(i<T[0]) {
                if (j==0 || T[i]==T[j]) {++i; ++j; next[i]=j; }
                else j=next[j];
        }
}//get_next
```

KMP算法：

```
int Index_KMP(SString S, SString T, int pos) {
        i=pos; j=1;
        while(i<=S[0] && j<=T[0]) {
                if (j==0 || S[i]==T[j]) {++i; ++j}
                else j=next[j];
        }
        if (j>T[0]) return i-T[0];//T[0]为T长度
        else return 0;
}//Index_KMP
```

- next[j]的修正值

匹配在j处失效，k = next[j], 如果p（j）=p（k），则k处必然失效，k=next[k], 以此类推，直至p（j）！=p（k）

a	b	a	a	b	c	a	c

j	1	2	3	4	5	6	7	8
next[j]	0	1	1	2	2	3	1	2
nextval[j]	0	1	0	2	1	3	0	2

计算nextval函数值算法

```
void get_nextval(SString S, int next[ ]) {
        i=1; nextval[1]=0; j=0;
        while(i<T[0]) {
                if (j==0 || T[i]==T[j]) {
                ++i; ++j;
                if(T[i]!=T[j])   nextval[i]=j;
                else  nextval[i]=nextval[j];
                }
                else j=nextval[j];
        }
}//get_nextval
```

Next函数值算法

```
void get_next(SString S, int next[ ]) {
        i=1; next[1]=0; j=0;
        while(i<T[0]) {
                if (j==0 || T[i]==T[j])
                  {++i; ++j; next[i]=j; }
                else j=next[j];
        }
}//get_next
```

练习

	a	b	c	a	a	b	b	a	b	c	a	b
j	1	2	3	4	5	6	7	8	9	10	11	12
next[j]	0	1	1	1	2	2	3	1	2	3	4	5
nextval[j]	0	1	1	0	2	1	3	0	1	1	0	5