
KMP & e-KMP

kuangbin

KMP

KMP解决的是**字符串匹配**问题：

一个文本串**S** (主串), 和一个模式串**P**, 求P在S中出现的位置，或者P在S中的出现次数，等等问题。

主串**S**

A	B	C	D	A	B	C	D	A	B	D	E
---	---	---	---	---	---	---	---	---	---	---	---

模式串**P**

A	B	C	D	A	B	D
---	---	---	---	---	---	---

KMP

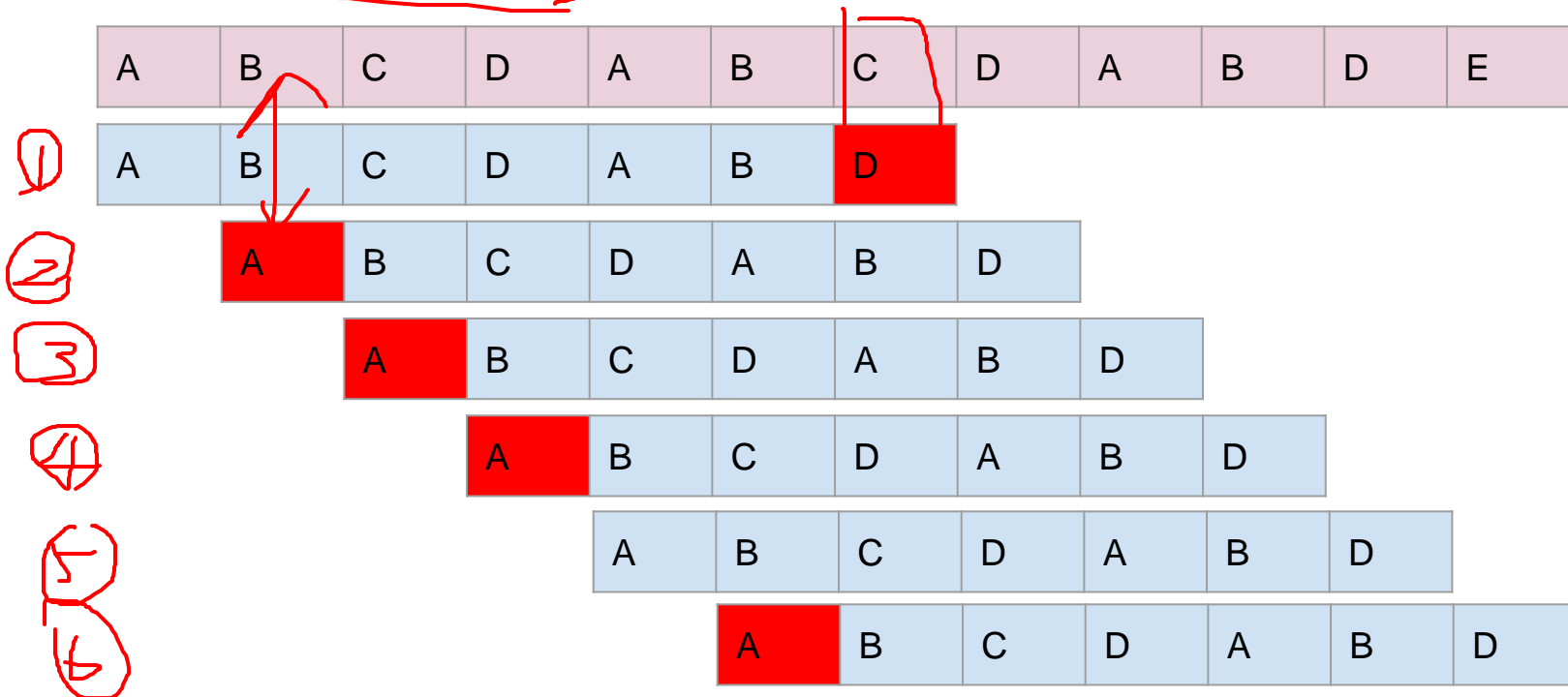
$n + m$

暴力做法，复杂度 $O(n*m)$, n 是P的长度， m 是S的长度。

```
// 返回P在S中的出现次数
int solve(char P[], int n, char S[], int m) { // P是模式串, S是主串
    int ans = 0;
    int i, j;
    for (i = 0; i + n <= m; i++) {
        for (j = 0; j < n; j++) {
            if (S[i+j] != P[j]) {
                break;
            }
        }
        if (j == n) {
            ans++;
        }
    }
    return ans;
}
```

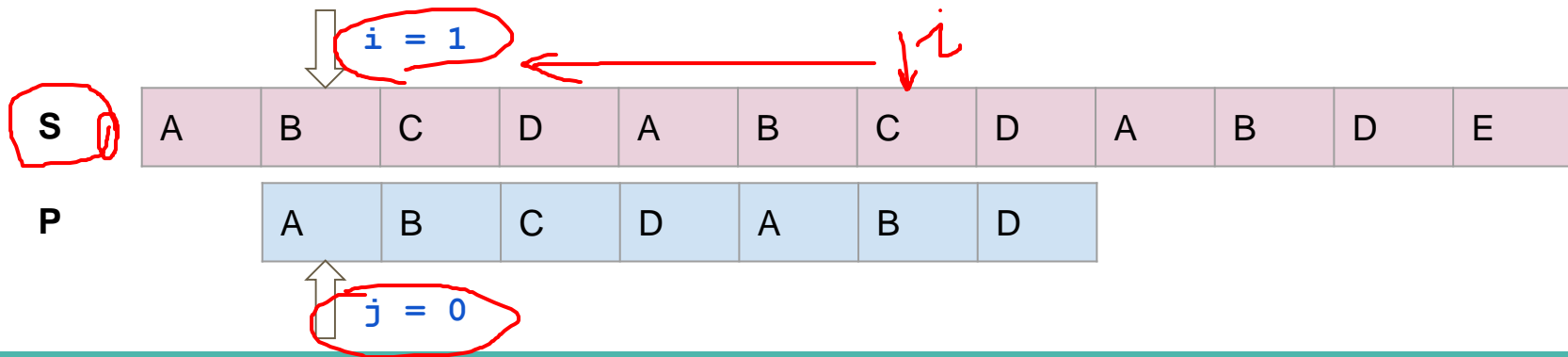
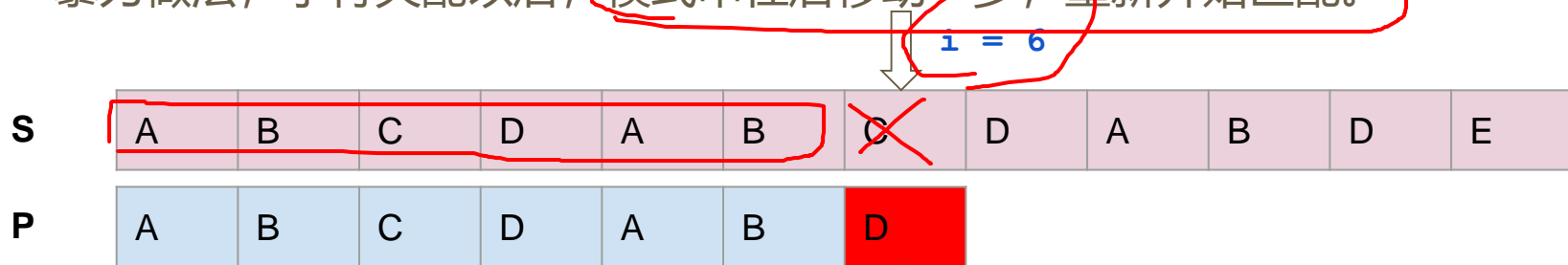
KMP

暴力做法, 复杂度 $O(n*m)$, n 是P的长度, m 是S的长度。



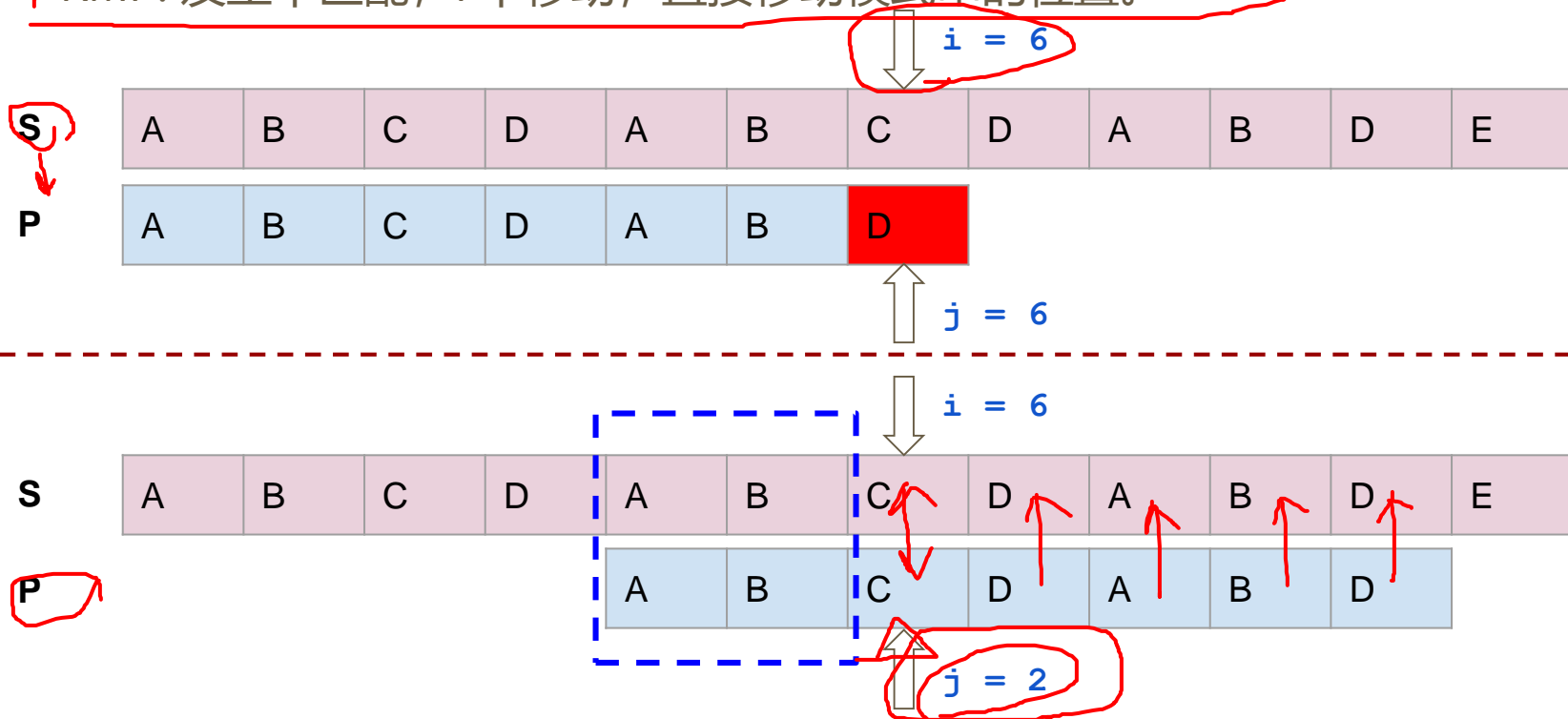
KMP

暴力做法，字符失配以后，模式串往后移动一步，重新开始匹配。



KMP

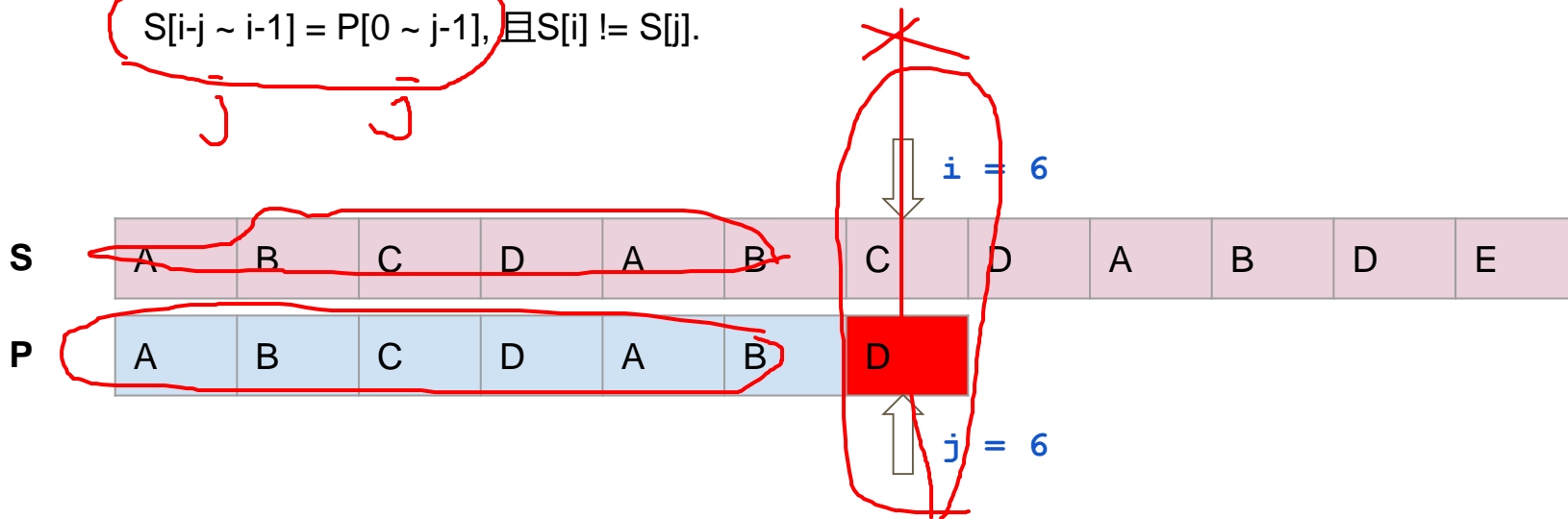
KMP: 发生不匹配, i 不移动, 直接移动模式串的位置。



KMP

在模式串j处失配以后，如何移动？

$S[i-j \sim i-1] = P[0 \sim j-1]$, 且 $S[i] \neq S[j]$.



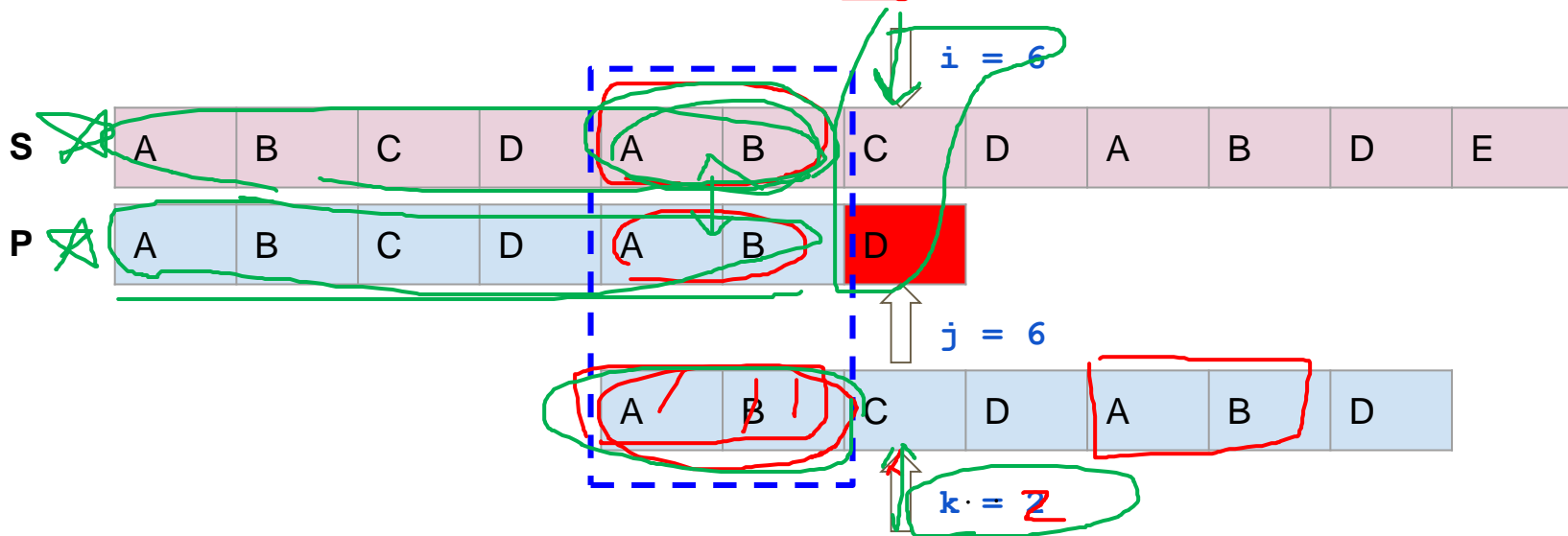
KMP

ABC PAB 2

假如需要将模式串移动到k处。则需要：

$$S[i-k \sim i-1] = P[0 \sim k-1]$$

等价于 $P[0 \sim k-1] = P[j-k \sim j-1]$, 而且k越大越好

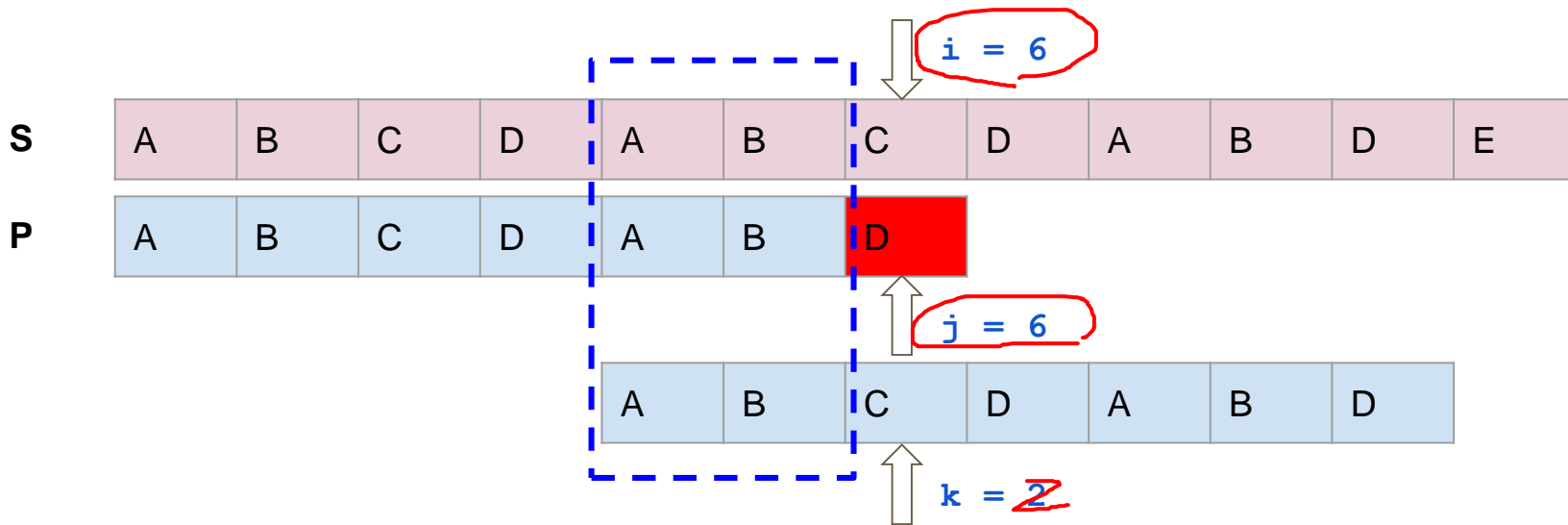


KMP



使用一个next数组:

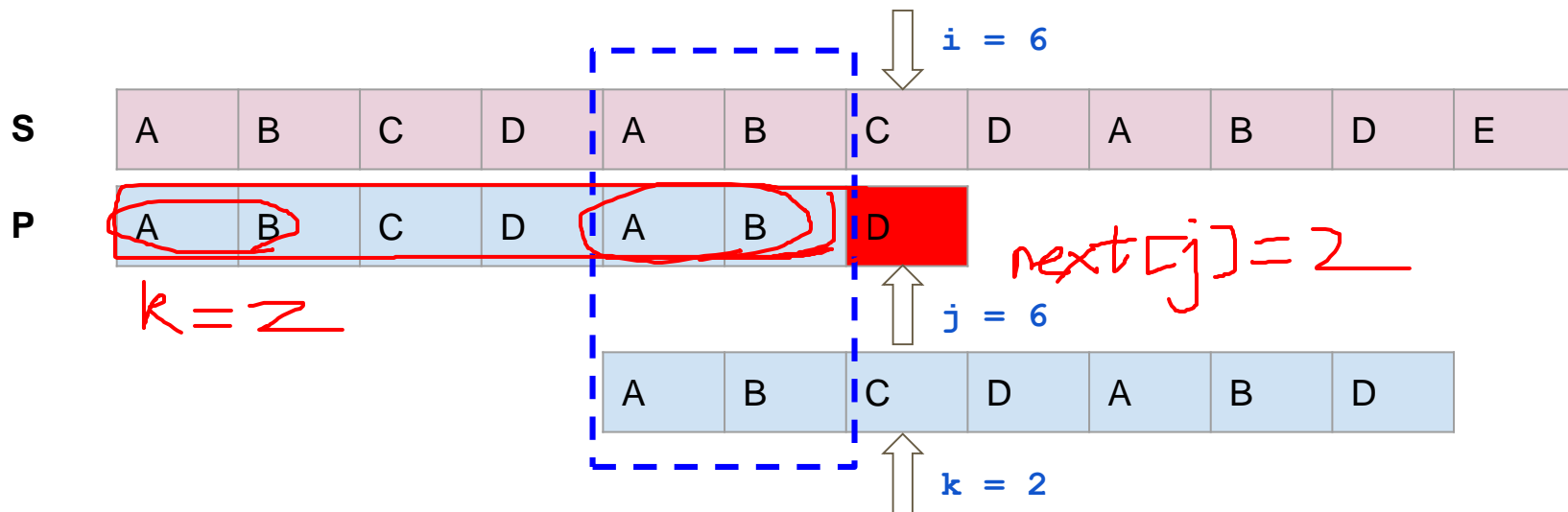
$\text{next}[j] = k$ 表示到 $S[i] \neq P[j]$ j 指针需要指向的位置。



KMP

next数组是模式串P的特性，只由P决定，和S无关：

$\text{next}[i]$ 为满足 $P[i-z \sim i-1] = P[0 \sim z-1]$ 的最大z值



KMP

1 ~ n

next数组:

next[i] 为满足 $P[i-z \sim i-1] = P[0 \sim z-1]$ 的最大z值

next数组的含义: 当前字符以前的字符串中, 有多大长度的相同前缀和后缀。

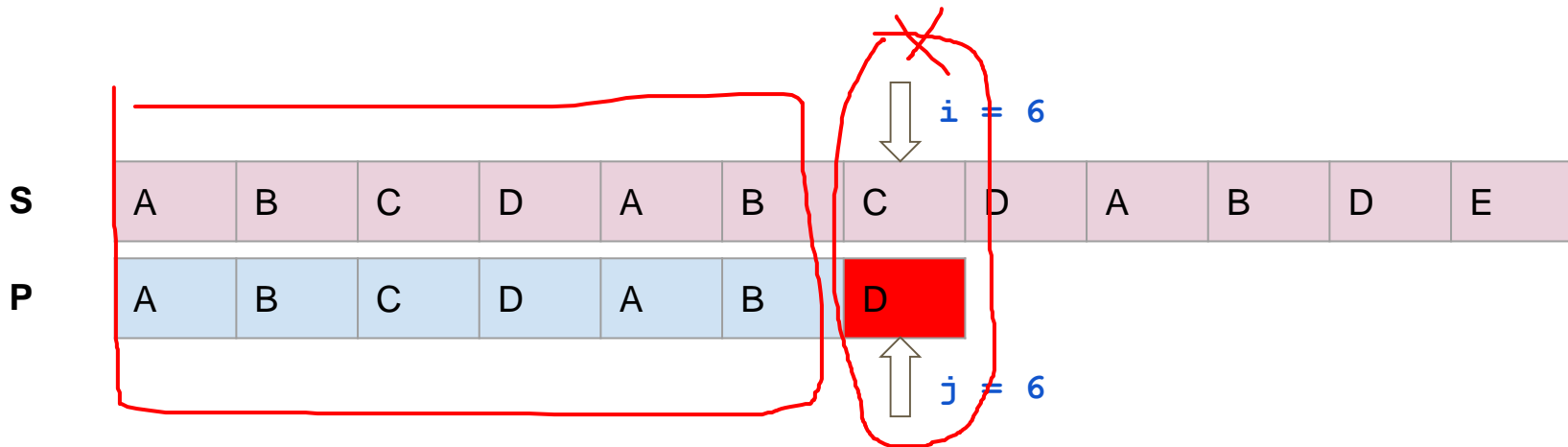
$P[i-\text{next}[i] \dots i-1] = P[0 \dots \text{next}[i]-1]$

	0	1	2	3	4	5	6	7
P	A	B	C	D	A	B	D	
next	-1	0	0	0	0	1	2	0

KMP

回到刚才失配问题：在模式串j处失配以后，如何移动？ $S[i-j \sim i-1] = P[0 \sim j-1]$, 且 $S[i] \neq S[j]$.

就是要找到一个最长的P的前缀来匹配P[0 ~ j-1] 的后缀



KMP

next数组的求法:

```
1  /*
2  * next[] 的含义:  $x[i-\text{next}[i] \dots i-1] = x[0 \dots \text{next}[i]-1]$ 
3  * next[i] 为满足  $x[i-z \dots i-1] = x[0 \dots z-1]$  的最大 z 值 (就是 x 的自身匹配)
4  */
5  void kmp_pre(char x[], int m, int next[]){
6      int i, j;
7      j = next[0] = -1;
8      i = 0;
9      while(i < m){
10         while(-1 != j && x[i] != x[j]) j = next[j];
11         next[++i] = ++j;
12     }
13 }
```

P A B C D A B D

next	-1	0	0	0	0	1	2	0
------	----	---	---	---	---	---	---	---

KMP

MAX

next数组的深入理解和应用:


沿着next数组一直往前后, 得到的是所有前缀, 也是当前后缀的字符串。

ABCABCABCAB -> ABCABCAB -> ABCAB -> AB

P	A	B	C	A	B	C	A	B	C	A	B	
next	-1	0	0	0	1	2	3	4	5	6	7	8

KMP

周期性字符串: $n \% (n - \text{next}[n]) == 0$, 循环节长度是 $n - \text{next}[n]$

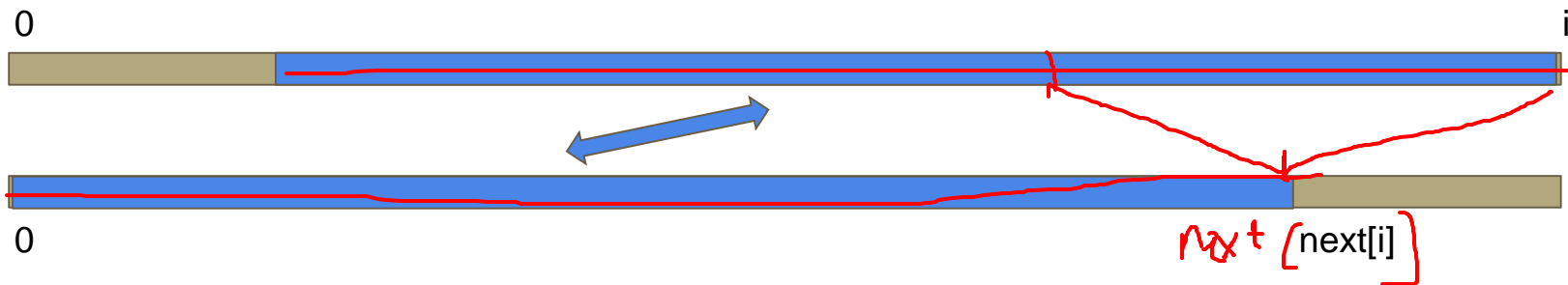
P													
	A	B	C	A	B	C	A	B	C	A	B	C	
next	-1	0	0	0	1	2	3	4	5	6	7	8	9

$$12 - 9 = 3$$

KMP

next数组往前跳的步长是一样, 除了最后一次。

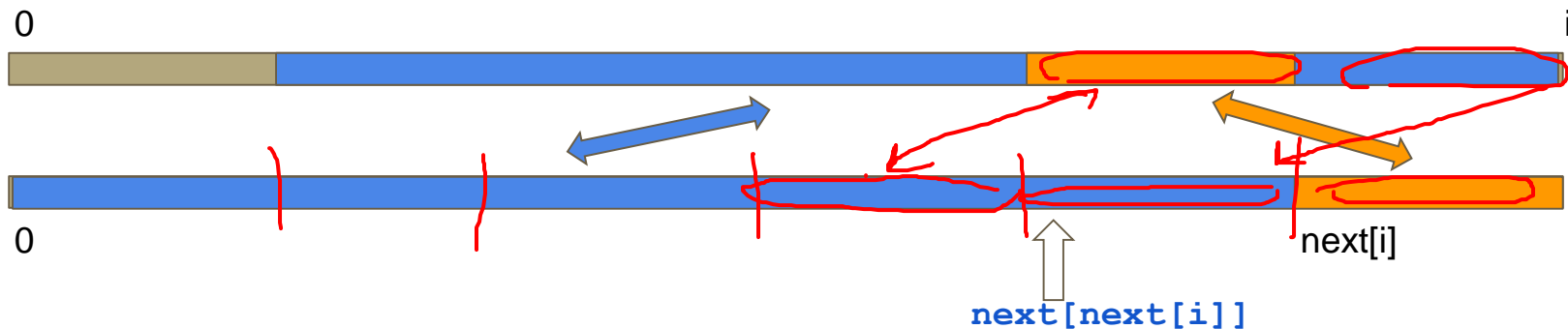
$$P[i - \text{next}[i] \dots i - 1] = P[0 \dots \text{next}[i] - 1]$$



KMP

next数组往前跳的步长是一样，除了最后一次。

$$P[i - \text{next}[i] \dots i - 1] = P[0 \dots \text{next}[i] - 1]$$



KMP

KMP解决的是**字符串匹配**问题：

一个文本串**S** (主串), 和一个模式串**P**, 求P在S中的出现次数。

```
1  /*
2   *  `返回x在y中出现的次数, 可以重叠`
3   */
4  int Next[100010];
5  int KMP_Count(char x[],int m,char y[],int n){`//`x是模式串, y是主串`
6      int i,j;
7      int ans=0;
8      kmp_pre(x,m,Next);
9      i=j=0;
10     while(i<n){
11         while(-1!=j && y[i]!=x[j])j=Next[j];
12         i++;j++;
13         if(j>=m){
14             ans++;
15             j=Next[j];
16         }
17     }
18     return ans;
19 }
```

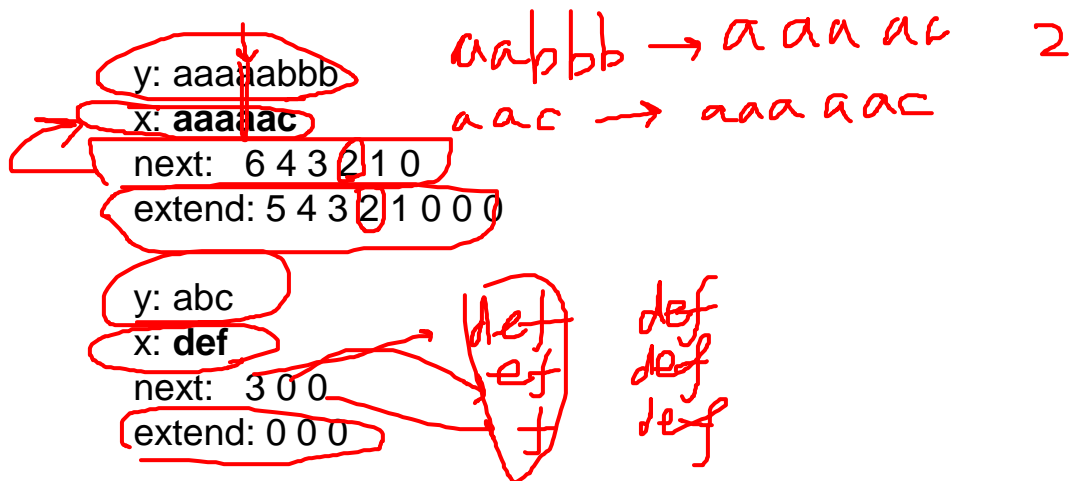
KMP

专题练习: <https://vjudge.net/contest/278481>

扩展KMP

$\text{next}[i]: x[i \dots m-1]$ 与 $x[0 \dots m-1]$ 的最长公共前缀

$\text{extend}[i]: y[i \dots n-1]$ 与 $x[0 \dots m-1]$ 的最长公共前缀



扩展KMP

HDU2594: <http://acm.hdu.edu.cn/showproblem.php?pid=2594>

given strings s1 and s2, finds the longest prefix of s1 that is a suffix of s2.

扩展KMP

HDU2594: <http://acm.hdu.edu.cn/showproblem.php?pid=2594>

given strings s1 and s2, finds the longest prefix of s1 that is a suffix of s2.

KMP 或者 e-KMP 都可以。

e-KMP: 找到 $i + \text{extend}[i] == \text{length}(s2)$.

扩展KMP

HDU6153: <http://acm.hdu.edu.cn/showproblem.php?pid=6153>

Thank you!

博客: <https://kuangbin.github.io/>

知识星球:



QQ交流群: 181826055