第二章
串

1 串的基本概念及存贮结构

2 串的运算

3 模式匹配

基本概念

假设V是程序设计语言所使用的字符集，由字符集V上的字符所组成的任何有限序列，称为字符串（简称为串）。

空串：不包含任何字符。

串的长度：一个串所包含的字符个数。

一个串的子串：这个串中任一个连续的子序列。

可以把串看成一种特殊的线性表，这种线性表是由单个字符依次排列而成的。

串的存储

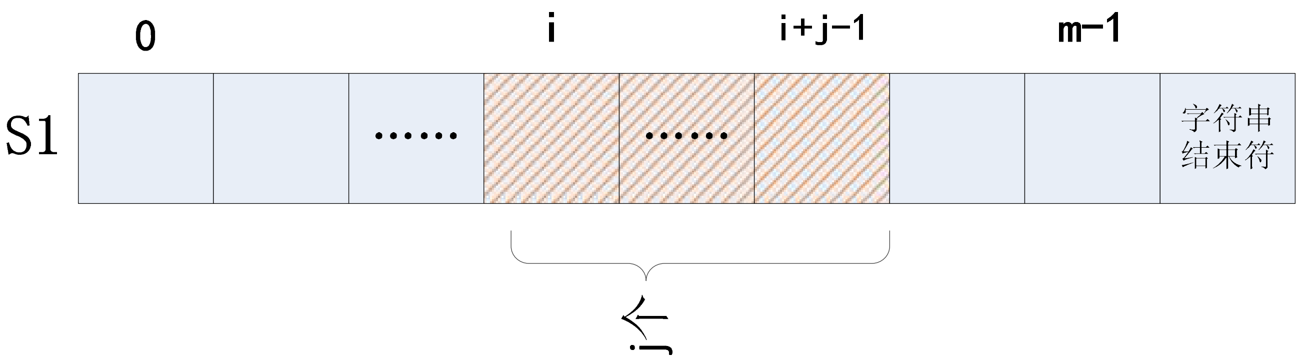
串的顺序存储：把串中的字符依次放在一组连续的存储空间中，字符数组。

串的链接存储：用结点值为字符的链表表示字符串。

常见串的运算及实现

P62

Strsub(s1,i,j,s2)，从串s1中位置i开始取长度为j的子串构成串s2



非法情况：

$i<0$;

$i>m-1$;

$J<0$;

$i+j-1>m-1$;

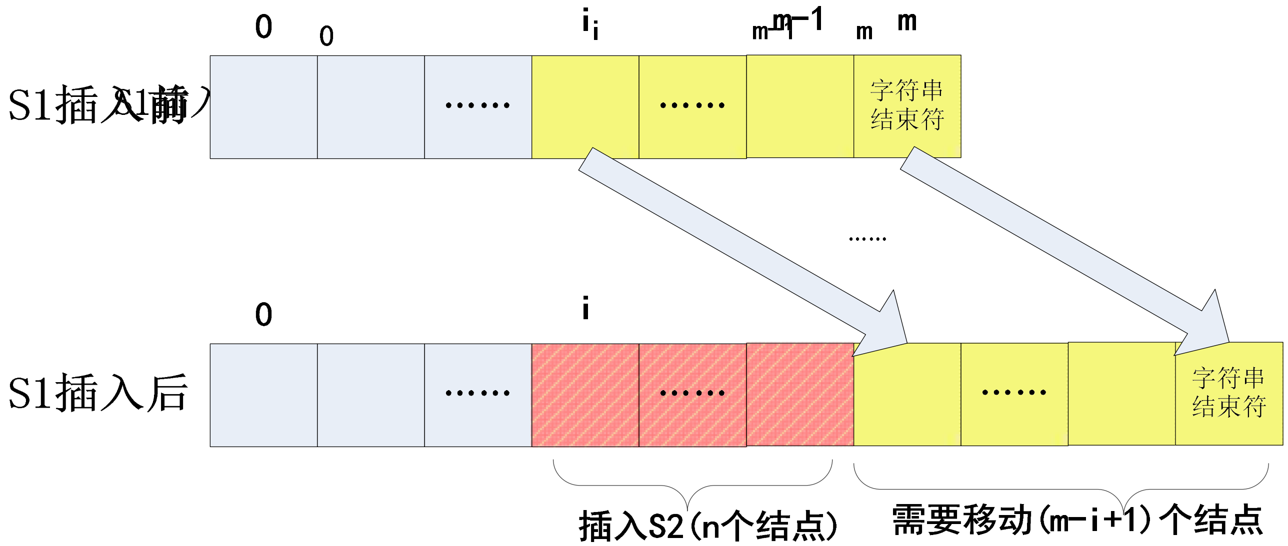
```
int strsub(char s1[], int i, int j, char s2[])
{//从串s1中位置i开始取长度为j的字串构成s2
   int m, k;
    if(i < 0 || i >= (m = strlen(s1))) //i越界
        return fail;                   //操作失败
    if(j < 0 || i + j > m)             //j越界
        return fail;

    for(k = 0; k < j; k ++)    //取长度为j的字串
        s2[k] = s1[i + k];

        s2[k] = '\0';
    return success;
}
```

常见串的运算及实现

int strins(char s1[], int i, char s2[])，把串s2插在串s1的位置i上。



非法情况：
i<0;
i>m;
（m+n）>MAXN

在用数组表示的串中，判断位置i(即数组下标）是非法的：i<0或i>=m;
（串的长度为m）

3 模式匹配

假设 T 和 P 是两个给定的串，在T中寻找等于P的子串的过程称为模式匹配。

称T为正文（text），P为模式（pattern）。通常T的长度远远大于P的长度。

如果在T中找到等于P的子串，那么匹配成功；否则，匹配失败。

简单算法：

对于i=0，1，2，……，n-m，依次执行下面的匹配步骤：
用p[0]～p[m-1]依次与t[i]~t[i+m-1]进行比较，如果每一个都相等，那么匹配成功，整个算法结束；

否则，一定存在某个整数k，使得p[k]不等于t[i+k]，一旦出现这种情况，立即中断后面的比较，然后执行下一次匹配操作。

简单算法：

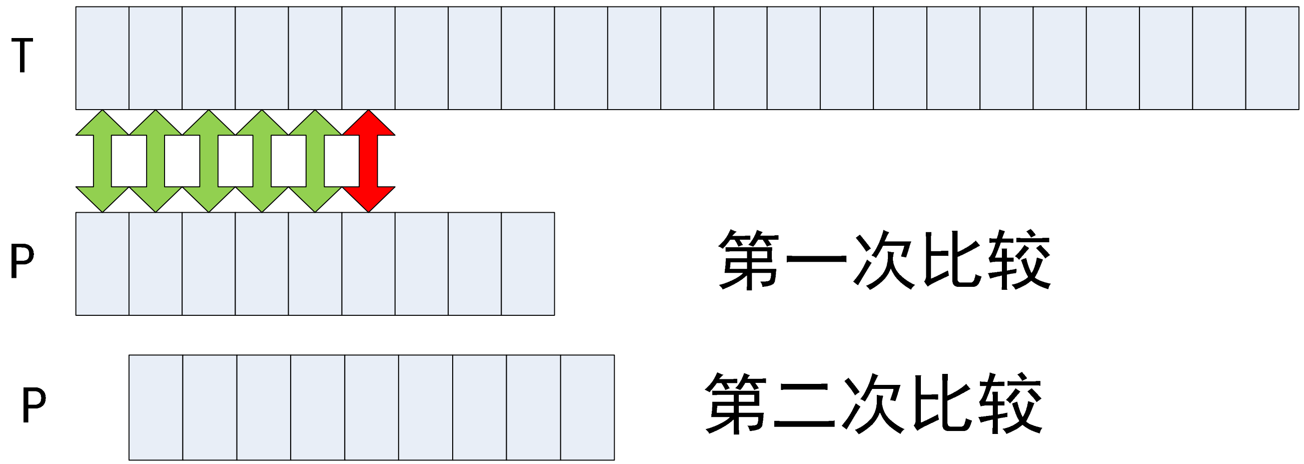
```
int simple_match(char t[], char p[], int n, int m) //简单模式匹配
{
    int i, j, k;
    for(i = 0; i <= n - m; i ++)  //从正文下标i开始匹配
     {
        for(j = 0, k = i; j < m && t[k] == p[j]; k ++, j ++);  //比较
        if(j == m) return i;  //匹配成功，返回模式串首次出现的起始下标
    }
      return -1;              //匹配失败
}
```

简单算法:

在最坏情况下， 匹配步骤最多执行多少次?

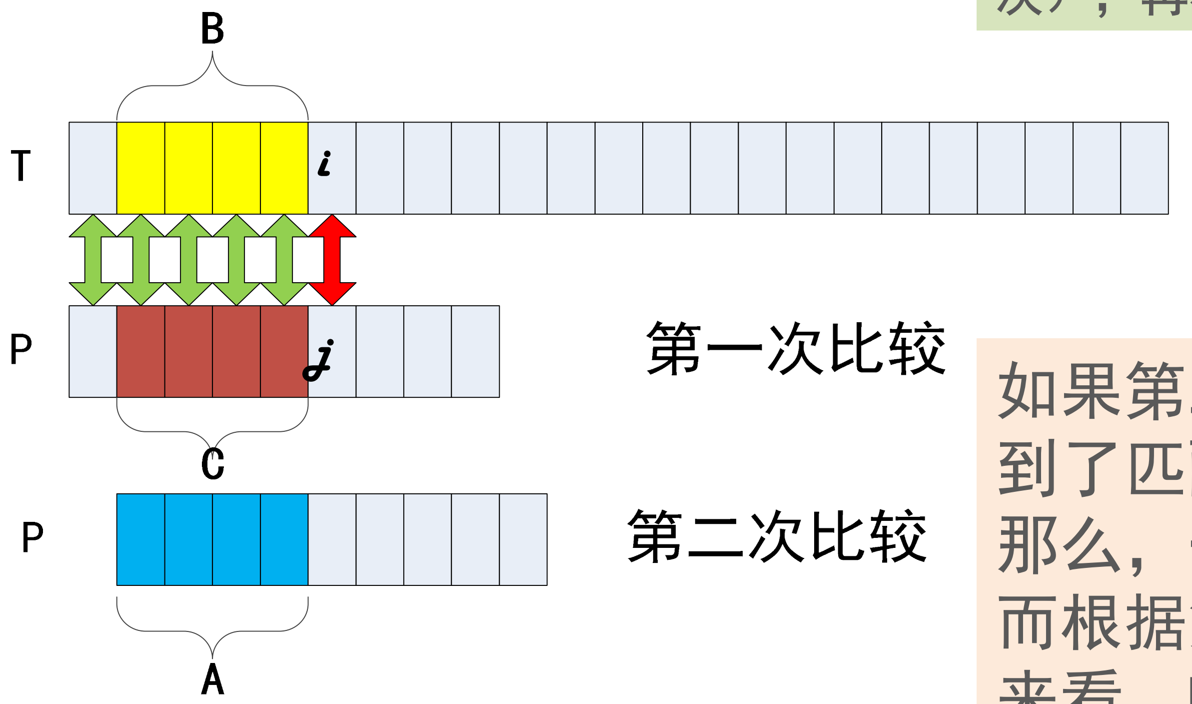
程序执行的字符串比较的总次数为m(n-m+1)。如果n>>m，则运行的时间为O（mn）。

KMP (Kunth-Morris-Pratt)算法



KMP（Kunth-Morris-Pratt)算法

如果在P中，我们预先通过某种判断已经得知A不等于C，则这次比较将不必进行（可以跳过这一次），再将P向前推进。



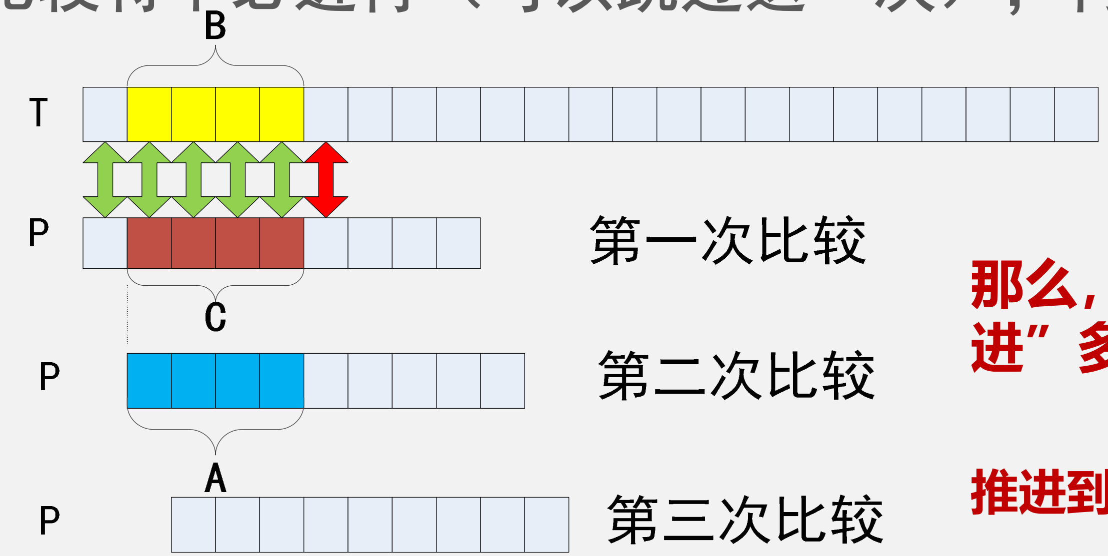
第一次比较

第二次比较

如果第二次比较是一次找到了匹配的比较，
那么，一定有A=B，
而根据第一次比较的情况来看，B=C，
那么，必然有 A=C

KMP（Kunth-Morris-Pratt)算法

A和C都是P的子串；
如果在P中，我们预先通过某种判断已经得知A不等于C，则这次比较将不必进行（可以跳过这一次），再将P向前推进。



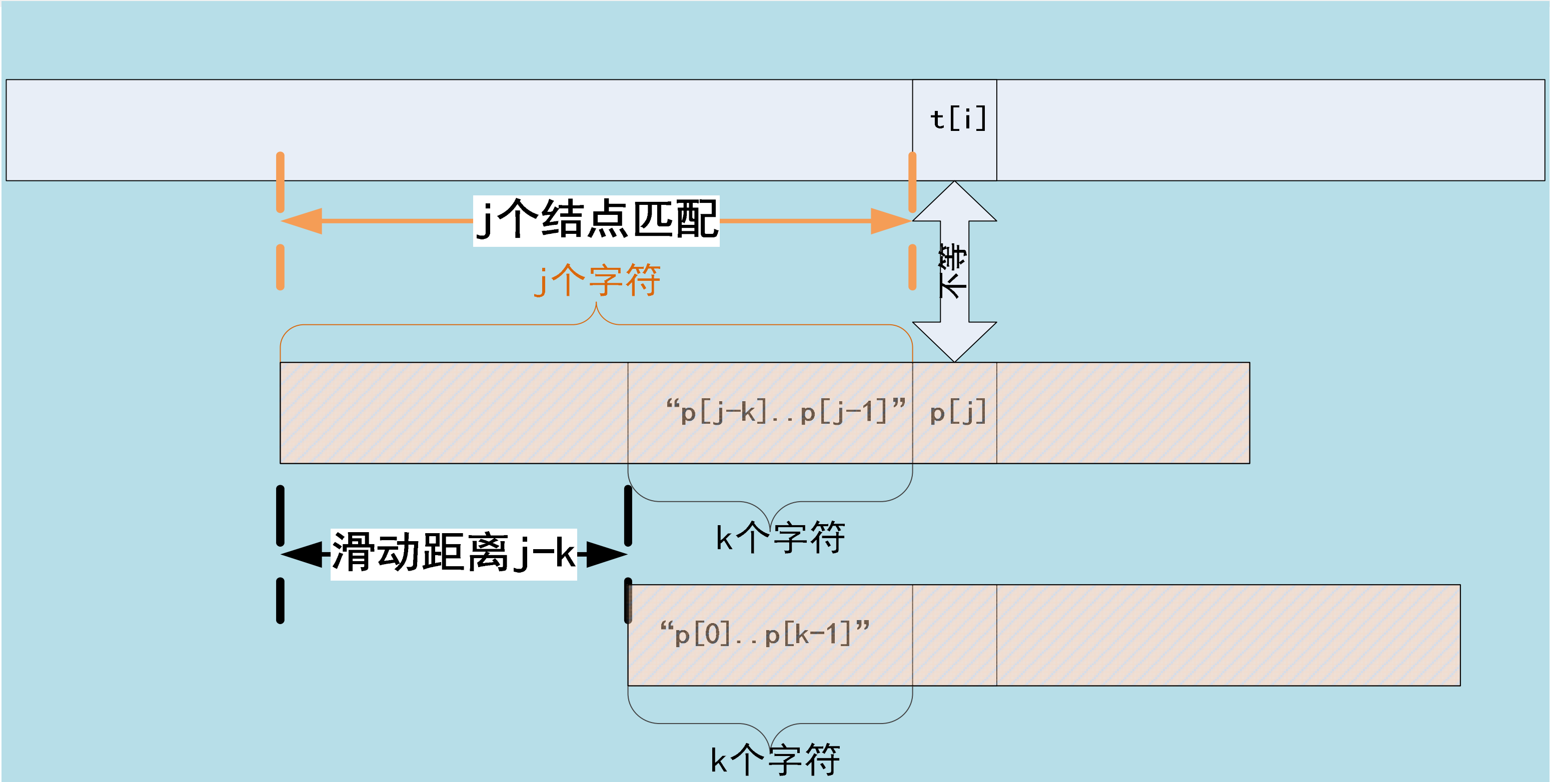
那么，是否可以判断出要“向前推进”多少?

推进到首次满足 A′（=B′）=C′

如果在P中存在某个k（k>=1，k<j），使得 “p[0]..p[k-1]” = “p[j-k]..p[j-1]” 这样，我们就直接向前推进w（w=j-k），并且可以不进行前k次的比较。

KMP（Kunth-Morris-Pratt)算法

在执行匹配比较的过程中，当不匹配的字符是p[j]时（即t[i]≠p[j]），



K要尽量大，则滑动的距离就尽量小，不错过可能的匹配。K只和串P有关。

KMP（Kunth-Morris-Pratt)算法

可以看到，k只和串P有关。我们可以预先针对P的每个j值，求出对应的k。记为flink[j]。 J的取值范围：0～m-1

当t[i]不等于p[j]时，把P”向前滑动” j-flink[j]个字符，并且直接从t[i]和P[flink[j]]是否相等开始比较。

J=0，第一个元素，如果不匹配，往后移动w(w=j-k),k=-1;

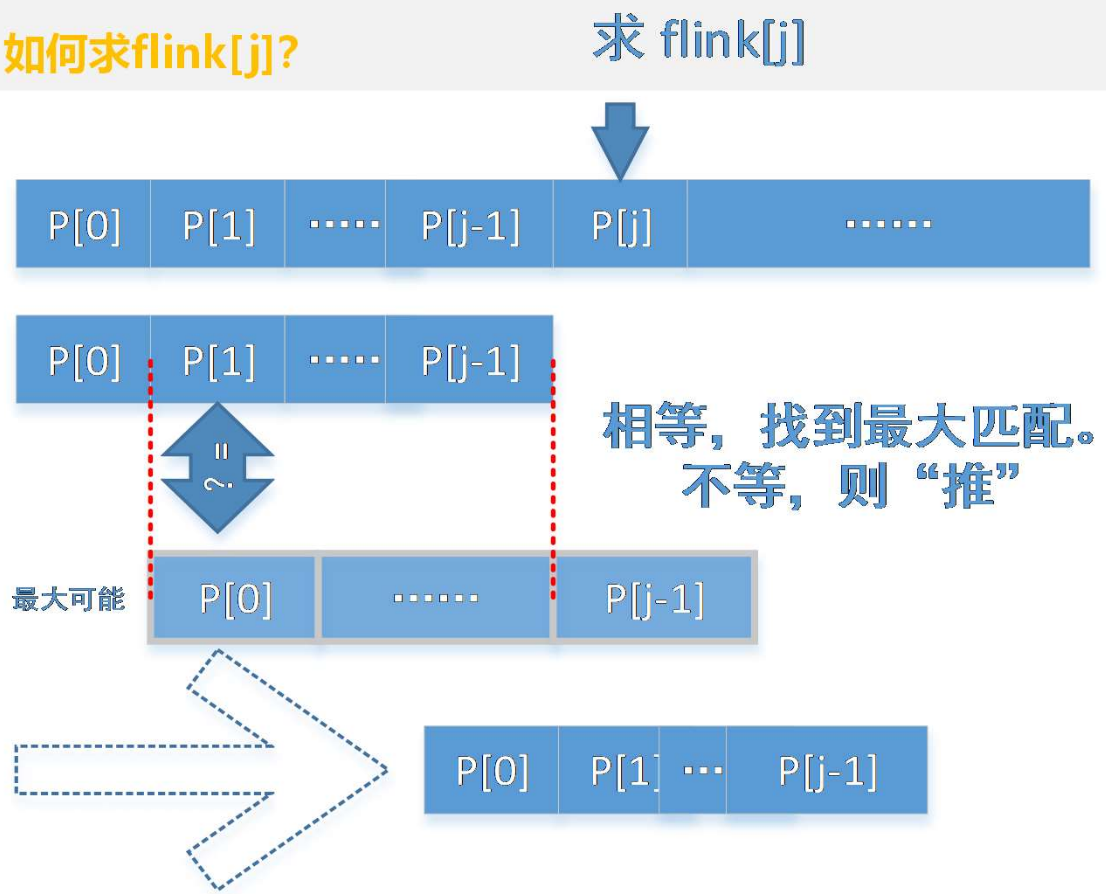
0<j<m, 无匹配，k取0; (j=1时，k=0)

0<j<m，max{k|0<k<j，‘p[0]..p[k-1]’ = ‘p[j-k]..p[j-1]’ }

K要尽量大，则滑动的距离就尽量小，不错过可能的匹配。

KMP（Kunth-Morris-Pratt)算法

在P中，对每个j，如何求flink[j]?



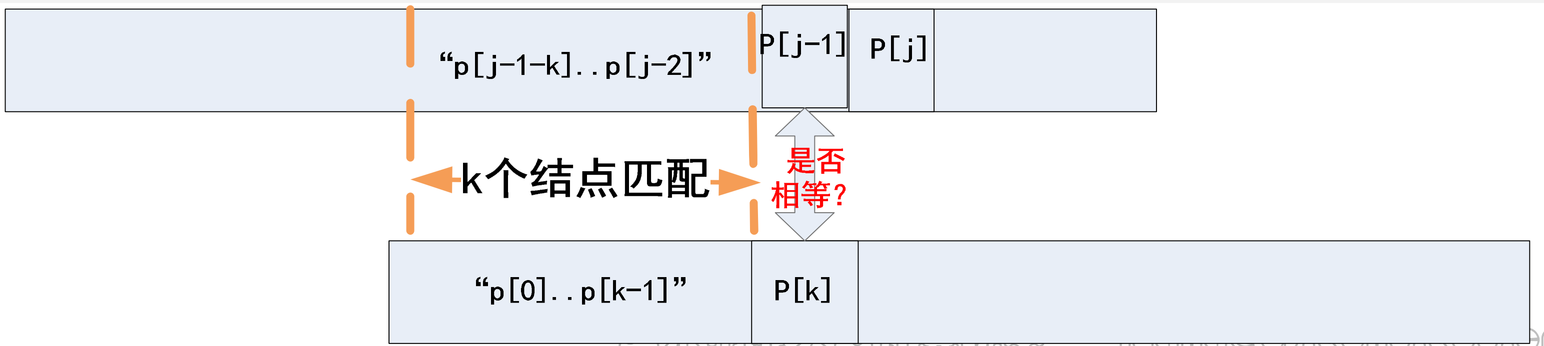
KMP（Kunth-Morris-Pratt)算法

在P中，对每个j，如何求flink[j]?

flink[0]=-1，然后依次求flink[1]...flink[m-1].

假设现在求flink[j]，之前的位置的flink已求出。如果flink[j-1]值为k,
那么，有　　**‘p[0]..p[k-1]’ ==‘p[j-k-1]..p[j-2]’**

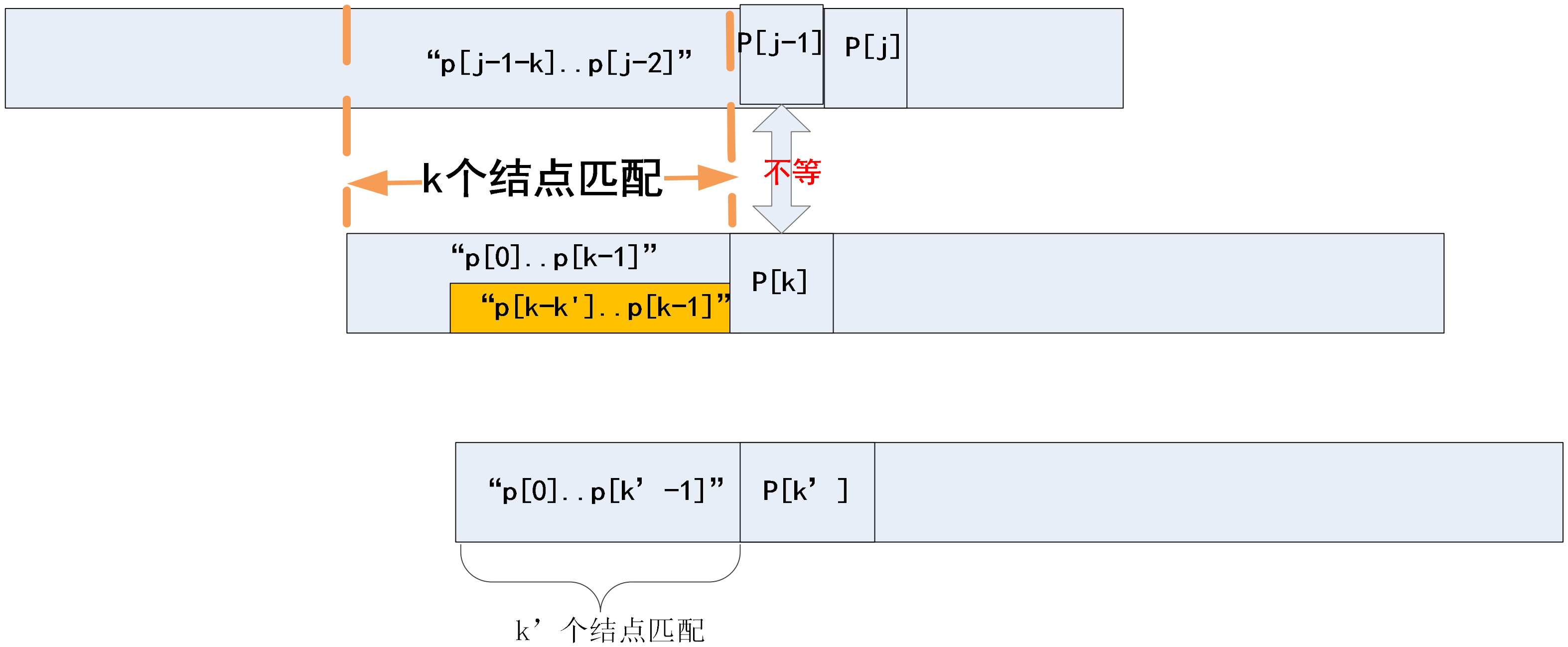
如果p[k]==p[j-1],
那么**p[0]..p[k-1]’+p[k]==‘p[j-k-1]..p[j-2]’+p[j-1]，则**flink[j]=k+1 .



KMP（Kunth-Morris-Pratt)算法

在P中，对每个j，如何求flink[j]? **(如果flink[j-1]值为k)**

如果p[k]≠p[j-1]，那么令k’ = flink[k]



KMP（Kunth-Morris-Pratt)算法

在P中，对每个j，如何求flink[j]?

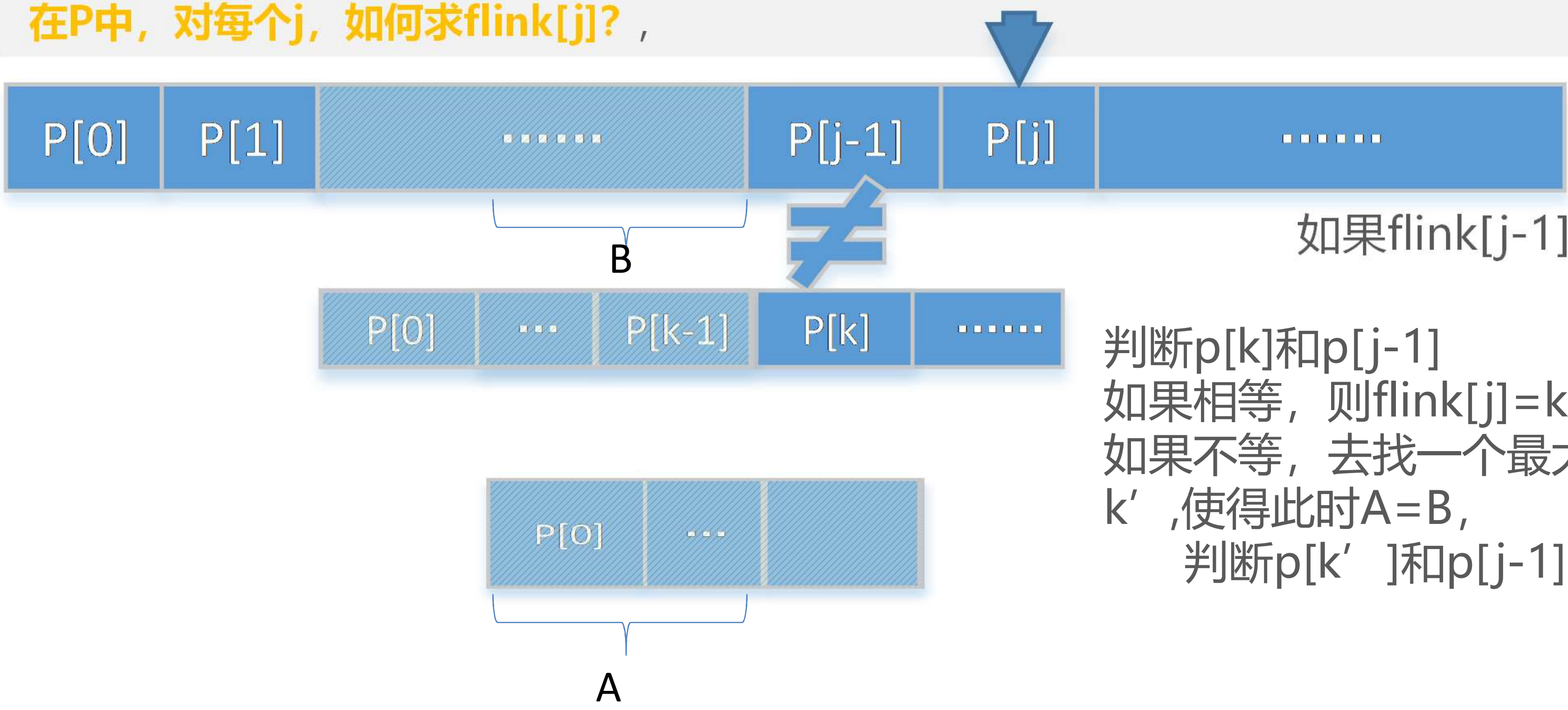
如果p[k]≠p[j-1]，那么令k’ = flink[k]，如果p[k’]=p[j-1]，flink[j]=k’+1;

如果p[k’] ≠p[j-1]，那么依据flink[k’]再试一次。这个过程一直做下去，直到找到一个k*，使得p[k*] =p[j-1]，或者k*=-1。

最后，置 flink[j]=k* + 1。
程序见教材：P67

KMP（Kunth-Morris-Pratt)算法

在P中，对每个j，如何求flink[j]?，



如果flink[j-1]=k

判断p[k]和p[j-1]
如果相等，则flink[j]=k+1
如果不等，去找一个最大的k’，使得此时A=B，
判断p[k’]和p[j-1]

K’：是位置k的失败链接值

求flink数组

```
void faillink(char p[], int flink[], int m) {
  int  j,k;
  flink[0]=-1; j=1;
  While (j<m) {
          k =flink[j-1];
          while(k!=-1 && p[k] !=p[j-1])    k=flink[k];
          flink[j]=k+1;
          j++;
  }
}
```

KMP匹配算法

```
int kmp_match(.......................){
   int i,j;
   i=0;j=0;
   While(i<n)
   {
            while(j!=-1  && t[i] ! = p[j]))   j = flink[j];
        // (在这个while循环中， “i不动”,一旦 t[i] == p[j] 则结束这个循环 )
        //一旦在P串的P[j]处出现失败，则不断寻找下一个可能成功的匹配。

             if (j == m-1 ) return(i-m+1);  //匹配成功

             i++;
             j++;
   }
   return (-1);
}
```

KMP匹配算法

实例:
T：baadaabc
P：aabc　　　　flink = (-1,0,1,0)

i	j	k	J-k

BM (Boyer-Moore)算法

（1）匹配时从右到左进行。
（2）**初始条件增强，**预先知道在T和P中会出现的字符的集合，预先计算出正文T中出现的字符在模式P中出现的位置的有关信息，用这个信息指导每次匹配失败时P的移动位置。

例：T，P 如下。假设：字符串由英文小写字母组成。

T ： t h e n e l s e
P: r e t u r n

从右到左开始匹配。

T ： t h e n e l s e ……
P: r e t……

华东师范大学计算机系 jyang@cs.ecnu.edu.cn

BM (Boyer-Moore)算法

例：T，P 如下。假设：字符串由英文小写字母组成。

T ： t h e n e l s e t u r n f a t h e r e t u r n

P: r e t u r n

从右到左开始匹配。

T ： t h e n e l s e t u r n f a t h e r e t u r n

P: r e t u r n

T ： t h e n e l s e t u r n f a t h e r e t u r n

P: r e t u r n

华东师范大学计算机系 jyang@cs.ecnu.edu.cn

BM（Boyer-Moore)算法

例：T，P 如下。假设：字符串由英文小写字母组成。

T ： t h e n e l s e t u r n f a t h e r e t u r n
P: r e t u r n

T ： t h e n e l s e t u r n f a t h e r e t u r n
P: r e t u r n

T ： t h e n e l s e t u r n f a t h e r e t u r n
P: r e t u r n

BM（Boyer-Moore)算法

BM算法的关键：对于T中所有出现的单个字符x，根据其在P中的位置，定义一个函数d（x）。
d（x）的值域为1～m。

d(x)=m，若x不在p中，或（x=p[m-1]且x≠p[i],0≤i≤m-2)
d(x)=m-i-1，其余情况，i=max{i| x=p[i], 0≤i≤m-2}

处理思想：假设在执行正文T中自位置i起，与模式P的自右至左的匹配检查中，一旦出现不匹配（不管在什么位置），则去执行P[m-1]与t[i+d(x)]开始的自右向左的匹配。（这里的x即字符t[i]）.
效果：类似于P“向右滑动” d（x）的距离。

BM（Boyer-Moore)算法

参考代码段。