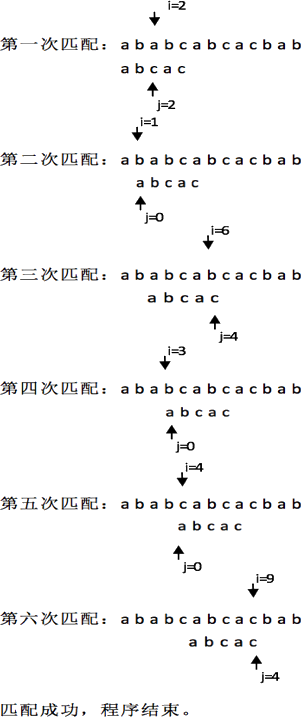
# **[数据结构19: BF算法(普通模式匹配算法)](https://www.cnblogs.com/ciyeer/p/9033187.html)**

判断两个串之间是否存在主串与子串的关系，这个过程称为串的模式匹配。

在串的模式匹配过程，子串 T 通常被叫做“模式串”。

## **普通的模式匹配（“BF”算法）**

判断两个串是否存在子串与主串的关系，最直接的算法就是拿着模式串，去和主串从头到尾一一比对，这就是“BF”算法的实现思想。  
  
将提供的模式串（例如 “abcac” ）从主串的第一个字符开始，依次判断相同位置的字符是否相等，如果全部相等，则匹配成功；反之，将子串向后移动一个字符的位置，继续与主串中对应的字符匹配。  
  
算法运行过程：（图中，i 和 j 表示匹配字符在数组中的位置下标）



如图所示，第一次匹配，模式串和主串匹配到第三个字符时，匹配失败；模式串向右移动一个字符的位置，还是从第一个字符 ‘a’ 和主串的第二个字符 ‘b’ 相匹配，匹配失败；模式串继续后移一个字符的位置，继续匹配。  
  
实现代码：

[IMG_257](javascript:void(0);)

#include <stdio.h>

#include <string.h>  
int sel(char \*S, char \*T)  
{

　　int i = 0, j = 0;

　　while (i<strlen(S) && j<strlen(T))   
　　{

　　　　if (S[i] == T[j])   
　　　　{

　　　　　　i++;

　　　　　　j++;

　　　　}  
　　　　else  
　　　　{

　　　　　　i = i-j+1;

　　　　　　j = 0;

　　　　}

　　}

　　//跳出循环有两种可能，i=strlen(S)说明已经遍历完主串；j=strlen(T),说明模式串遍历完成，在主串中成功匹配

　　if (j == strlen(T))   
　　{

　　　　return i-strlen(T)+1;

　　}

　　//运行到此，为i==strlen(S)的情况

　　return 0;

}  
int main()   
{

　　int add = sel("ababcabcacbab", "abcac");

　　printf("%d", add);

　　return 0;

}

运行结果：

　　6

## **“BF”算法的时间复杂度**

“BF” 算法在最理想的情况下的时间复杂度为O(m)( m 是模式串的长度，也就是第一次匹配就成功的情况)。  
  
一般情况下，"BF"算法的时间复杂度为O(n+m)(n是主串的长度，m是模式串的长度)。  
  
最坏的情况下的时间复杂度为O(n\*m)（例如主串 S 为“000000000001”，模式串 T ”001”,每次匹配时，直到匹配最后一个元素，才得知匹配失败，运行了 n\*m 次）。

## **总结**

“BF”算法在进行模式匹配时，从主串的第一个字符开始，每次失败，模式串向后移动一个字符的位置，继续匹配，无脑式操作。但是整个算法受测试数据的影响非常大，在解决实际问题时，由于数据量庞大，时间复杂度往往会很高。  
  
所以，对在“BF”算法的基础上，对其做了改进，就是下一节要讲的“KMP”算法。