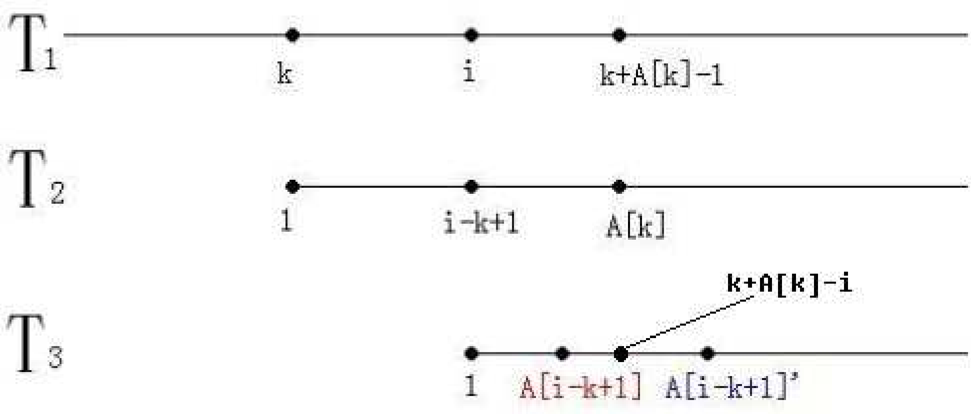
设：
S－主串，长度为 m
T－匹配串，长度为 n
A_i－T[i..n]和 T[1..n]的最长公共前缀，即从 T 的第 i 位用 T 去匹配的最大长度。
B_i－S[i..m]和 T[1..n]的最长公共前缀，即从 S 的第 i 位用 T 去匹配的最大长度。

普通的 KMP 算法只是在判断是否存在整数 i 满足 B[i]=m，求出所有 B[1…n]的函数值并保持线性复杂度就是扩展 KMP 算法（Extended KMP）。



函数 A[1…n]的求解过程：
1. A[1]=n；A[2]通过逐个字符比较得出。
2. 假设已经求出函数 A[2], A[3] … A[i-1]，并且 k 满足 1<k<i 且 k+A[k]最大。现在我们希望通过 A[2] ... A[i-1]求出 A[i]。
3. 如图，当前求 A[i]，对于任意 k<i，A[k]已求出
   在 k 用 T_2 去匹配，则匹配到 A[k]，对应与 T_1 中的 k+A[k]-1
   T_1 中的 i 对应于 T_2 中的 i-k+1，有 T_1[i ...k+A[k]-1]=T_2[i-k+1...A[k]]
   在 i-k+1 用 T_3 去匹配，则匹配到 A[i-k+1]，分情况讨论：
   如果 A[i-k+1] < k+A[k]-i（图中红色），那么 A[i]=A[i-k+1]
   如果 A[i-k+1]⩾k+A[k]-i（图中蓝色），那么 A[i]⩾A[i-k+1]，从 T_1 中的 k+A[k]开始逐个字符的匹配，计算出 A[i]
   从上诉过程中可知，为了尽量减少重复运算，应选择使 k+A[k]-1 最大的 k，特殊的如果 k+A[k]-1<i，那么 A[i]无法借助之前的运算，需要逐个字符的匹配。
   如果求 A[i]需要逐个字符匹配，那么在求完 A[i]后，对于以后的计算，i+A[i]-1 最大，所以置 k=i
   因为 i+A[i]-1 增加时才进行匹配，所以总的匹配数是 O(n)的。
   求 B 的方法于求 A 类似。

求 A：
```
 j:=0;
 while T[1+j]=T[2+j] do
   inc(j);
 a[2]:=j;
 k:=2;
```

```
for i:=3 to length(T) do
begin
  max:=k+a[k]-1;
  now:=a[i-k+1];
  if now<max-i+1 then
    a[i]:=now
  else
  begin
    if max-i+1>0 then
      j:=max-i+1
    else
      j:=0;
    while T[i+j]=T[1+j] do
      inc(j);
    a[i]:=j;
    k:=i;
  end;
end;
```

求 B：

```
j:=0;
while T[1+j]=S[1+j] do
  inc(j);
b[1]:=j;
k:=1;
for i:=2 to length(S) do
begin
  max:=k+b[k]-1;
  now:=a[i-k+1];
  if now<max-i+1 then
    b[i]:=now
  else
  begin
    if max-i+1>0 then
      j:=max-i+1
    else
      j:=0;
    while S[i+j]=T[1+j] do
      inc(j);
    b[i]:=j;
    k:=i;
  end;
end;
```