11. String Matching

The problem

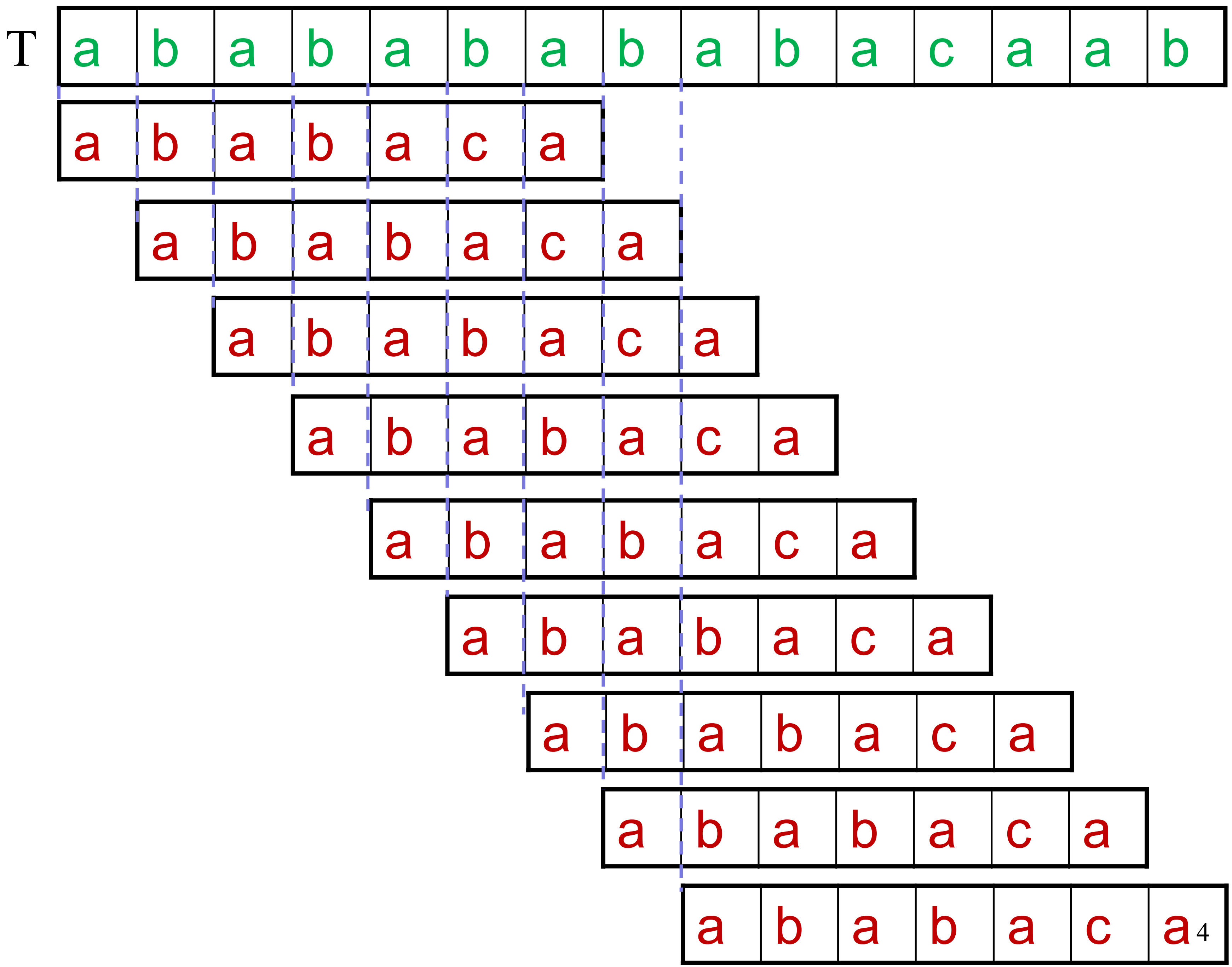
- Input: a text T (very long string) and
  a pattern P (short string).

- Output: the index in T where a copy of P begins.

Example

T:	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P:	a	b	a	b	a	c	a


T
a b a b a b a b a b a c a a b
a b a b a c a
a b a b a c a
a b a b a c a
a b a b a c a
a b a b a c a
a b a b a c a
a b a b a c a
a b a b a c a
a b a b a c a

Brute force method

Basic idea:

1. i=1;
2. do

   Start with T[i] and match P with

$$\begin{array}{ccccc} T[i], & T[i+1], & \ldots & T[i+|P|-1] \\ | & | & & | \\ P[1] & P[2] & & P[|P|] \end{array}$$

3. until a mismatch is found,

   i=i+1 and **goto** 2 until i+|P|-1>|T|.

Analysis

- Step 2 takes O(|P|) comparisons in the worst case.
- Step 2 could be repeated O(|T|) times.
- Total running time is O(|T||P|).

Notations and Terminologies

- **|P|** and **|T|**: the lengths of P and T.
- **P[i]**: the **i**-th letter of P.
- **Prefix** of P: a substring of P starting with P[1].
- **P[1…i]**: the prefix containing the first i letters of P.

Example: abcabbccaa.

prefix: a, ab, abc, abca, abcab, abcabb, ….

Knuth-Morris-Pratt (KMP) Method (linear time algorithm)

A better idea

- In step 3, when there is a mismatch we move the pattern forward by one position (i=i+1).
- We may move more than one position at a time when a mismatch occurs. (carefully study the pattern P).

For example:

```
P: ABABC          ABA
T: ABABABCCA ABABABCCA
       ↑              ↑
```

Property of ABAB: $ABA(frist\ 3) \neq BAB, last\ 3)$

$AB(frist\ 2) = AB\ (last\ 2)$

- **Key:** *When mismatch occurs at P[i+1], we want to find the **longest** prefix of P[1..i] which is also a suffix of P[1..i].*

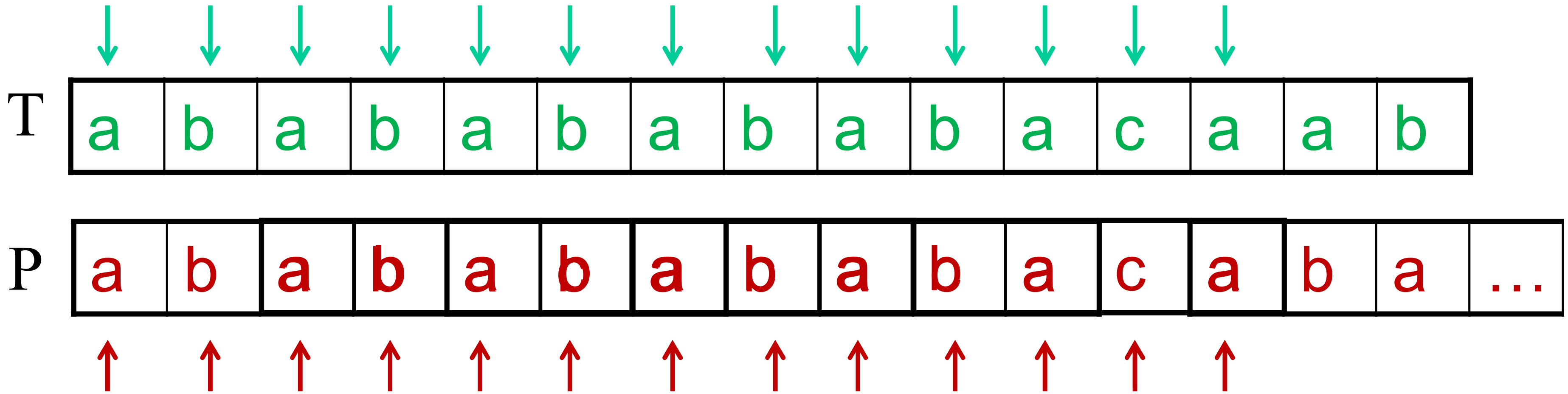
Failure function

- f(i) *is the* ***largest r with (r<i)*** *such that*
  P[1] P[2] ...P[r] = P[i-r+1]P[i-r+2], ..., P[i].

Prefix of length r *Suffix of P[1]P[2]...P[i] of length r*

Example: P=abcabbabcabbaa

P[1] =a,	f(1)=0;
P[1..2] =ab,	f(2)=0;
P[1..3] =abc,	f(3)=0;
P[1..4] =abca,	f(4)=1;
P[1..5] =abcab,	f(5)=2;
P[1..6] =abcabb,	f(6)=0;
P[1..7] =abcabba,	f(7)=1;
P[1..8] =abcabbab,	f(8)=2;
P[1..9] =abcabbabc,	f(9)=3;
P[1..10]=abcabbabca,	f(10)=4;
P[1..11]=abcabbabcab,	f(11)=5;
P[1..12]=abcabbabcabb,	f(12)=6;
P[1..13]=abcabbabcabba,	f(13)=7;
P[1..14]=abcabbabcabbaa,	f(14)=1;



f(1)=0, f(2)=0, f(3)=1, f(4)=2, f(5)=3, f(6)=0, f(7)=1

The Scan Algorithm

- i : indicates that T[i] is the next character in T to be compared (green arrow).
- q: indicates that P[q+1] is the next character in P to be compared with T[i] (red arrow-1).

1. i=1 and q=0;
2. compare T[i] with P[q+1]
   case 1: T[i]==P[q+1]
   i=i+1;q=q+1;
   if q==|P| then print "P occurs at i-|P|", and q=f(|P|).
   case 2: T[i]≠P[q+1] and q≠0
   q=f(q); % the pattern shifts forward
   case 3: T[i]≠P[q+1] and q==0
   i=i+1; % the pattern shifts one position forward
3. Repeat step2 until i==|T|.

<u>Illustration:</u> given a String ‘S’ and pattern ‘P’ as follows:

T	b	a	c	b	a	b	a	b	a	b	a	c	a	c	a

P	a	b	a	b	a	c	a

Let us execute the KMP algorithm to find whether ‘P’ occurs in ‘T’.

the failure function, f was computed previously and is as follows:

q	1	2	3	4	5	6	7
f	0	0	1	2	3	0	1

Initially: n = size of T = 15;

m = size of P = 7

Step 1: i = 1, q = 0

comparing P[0+1] with T[1]

T	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P	a	b	a	b	a	c	a

P[1] does not match with T[1]. P will be shifted one position to the right.

Step 2: i = 2, q = 0

comparing P[1] with T[2]

T	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P	a	b	a	b	a	c	a

P[1] matches T[2]. Since there is a match, P is not shifted.

Step 3: i = 3, q = 1

Comparing P[2] with T[3]   P[2] does not match with T[3]

T	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P	a	b	a	b	a	c	a

Backtracking on p, comparing P[1] and T[3]

Step 4: i = 4, q = 0

comparing P[1] with T[4]   P[1] does not match with T[4]

T	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P	a	b	a	b	a	c	a

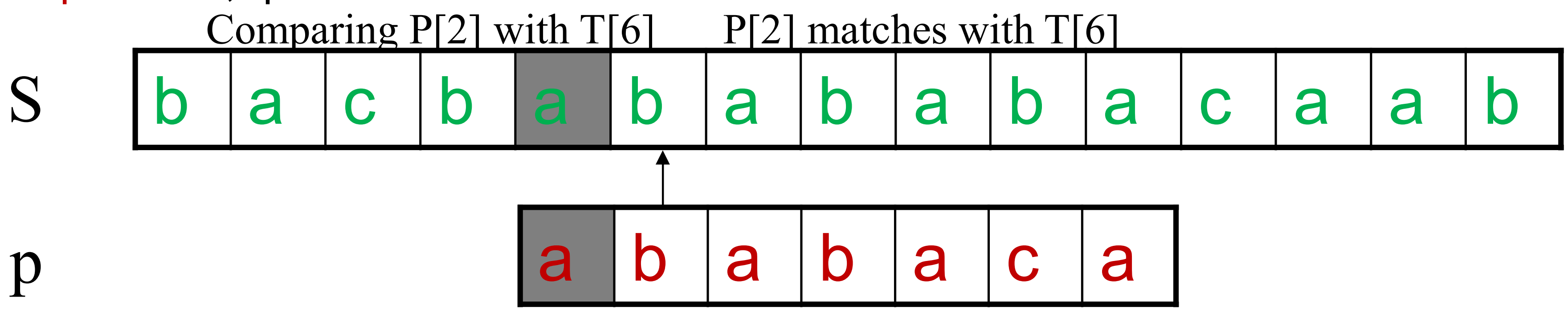
Step 5: i = 5, q = 0

comparing P[1] with T[5]   P[1] matches with T[5]

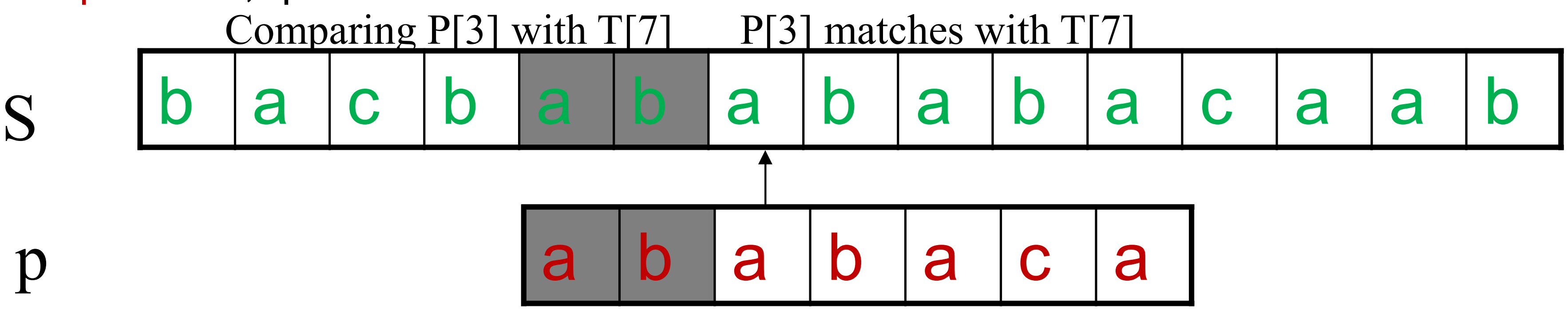
T	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

p	a	b	a	b	a	c	a

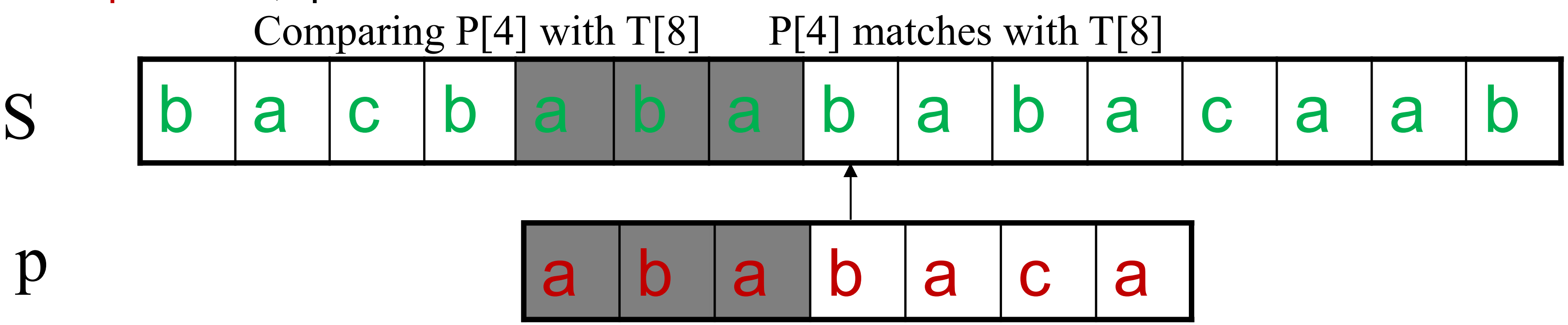
Step 6: i = 6, q = 1



Step 7: i = 7, q = 2



Step 8: i = 8, q = 3



Step 9: i = 9, q = 4

Comparing P[5] with T[9] P[5] matches with T[9]

T	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P	a	b	a	b	a	c	a

Step 10: i = 10, q = 5

Comparing P[6] with T[10] P[6] doesn't match with T[10]

T	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P	a	b	a	b	a	c	a

Mismatch. Backtracking on p, q=f(5)=3, comparing P[3+1] with T[10].

Step 11: i = 11, q = 4

Comparing P[5] with T[11] P[5] matches with T[11]

T	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P	a	b	a	b	a	c	a

Step 12: i = 12, q = 5

Comparing P[6] with T[12]   P[6] matches with T[12]

T

b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P

a	b	a	b	a	c	a

Step 13: i = 13, q = 6

Comparing P[7] with T[13]   P[7] matches with T[13]

T

b	a	c	b	a	b	a	b	a	b	a	c	a	a	b

P

a	b	a	b	a	c	a

Pattern ‘P’ has been found to completely occur in string ‘T’. The total number of shifts that took place for the match to be found are: i – m = 13 – 7 = 6 shifts.

b	a	c	b	a	b	a	b	a	b	a	c	a	c	a

a	b	a	b	a	c	a

q	1	2	3	4	5	6	7
f	0	0	1	2	3	0	1

Solution:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	b	a	c	b	a	b	a	b	a	b	a	c	a	c	a
	a														
		a	b												
			a	b	a	b	a	c							
					a	b	a	b	a	c					
							a	b	a	b	a	c	a		

P occurs at i=7 in T.

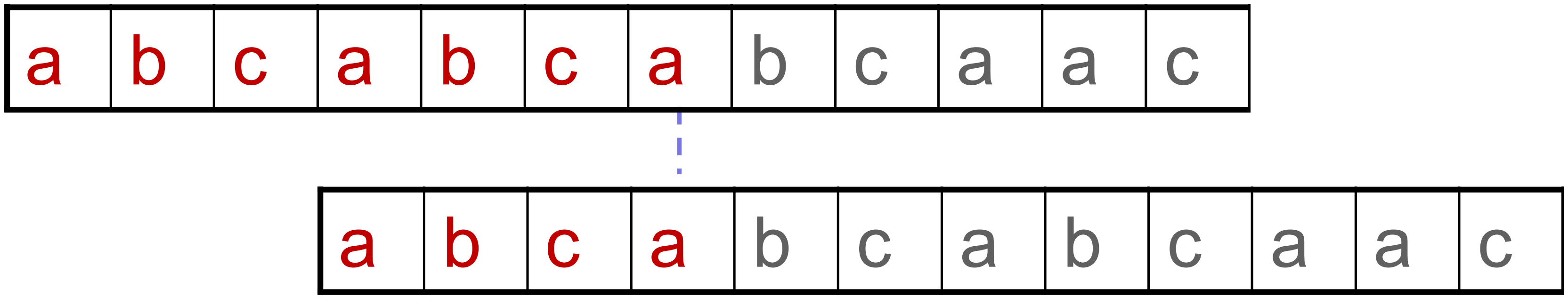
Running time complexity(hard)

- The running time of the scan algorithm is O(|T|).
- Proof:
  - After each comparison.
    either i increases by 1 (i.e, the pointer moves one step forward)
    or the pattern moves at least one postion forward.

Another version of scan algorithm (code)

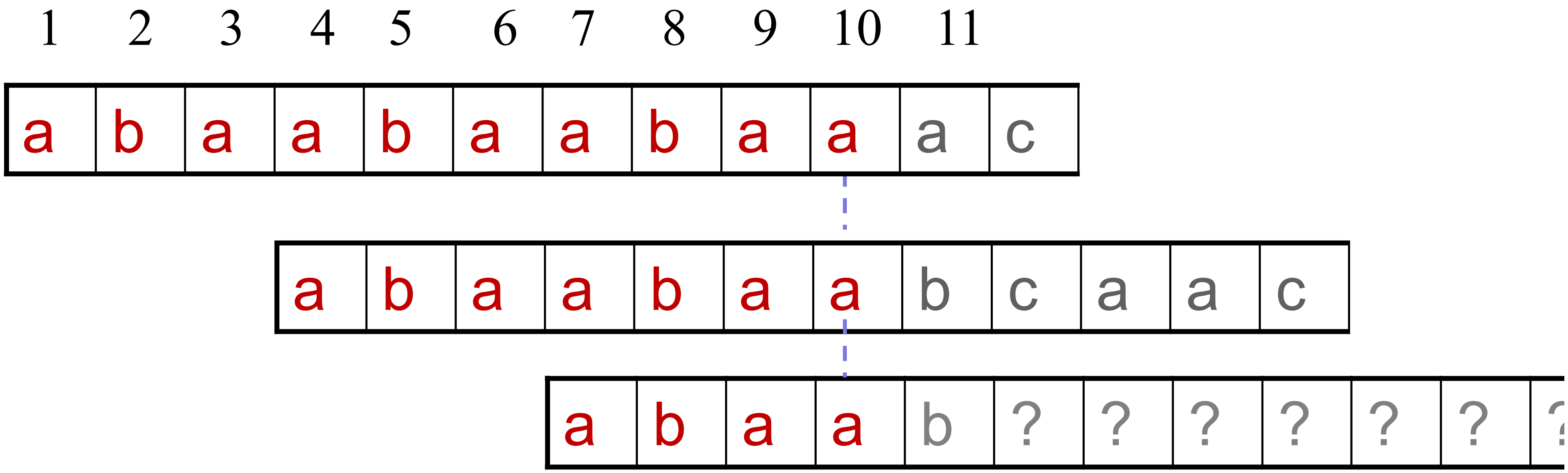
```
n=|T|
m=|P|
q=0
for i=1 to n
{
    while q>0 and P[q+1]≠T[i] do
    {
        q=f(q)
    }
    if P[q+1]==T[i] then
        q=q+1
    if q==m then
    {
        print "pattern occurs at i-m+1"
        q=f(q)
    }
}
```

Failure Function



f(7)=4, f(8)=f(7)+1

Failure Function



f(7)=4, f(8)=f(7)+1=5, f(8)=f(9)+1=6, f(10)=f(9)+1=7

f(11): try to see if P[1..f(f(10))] is good!

Failure Function

Case 1: f(1) is always 0.

Case 2: if P[q]==P[f(q-1)+1] then f(q)=f(q-1)+1.

Example: p=abcabcc

abc

f(1)=0; f(2)=0; f(3)=0; f(4)=1; f(5)=2; f(6)=3; f(7)=0;

P[4]= P[f(4-1)+1], f(4)=f(4-1)+1=1.

P[5]= P[f(5-1)+1], f(5)=f(5-1)+1=1+1=2.

P[6]= P[f(6-1)+1]. F(6)=f(6-1)+1=2+1=3.

Case 3: if $P[q] \neq P[f(q-1)+1]$ and $f(q-1) \neq 0$ then consider $P[q]$?= $P[f(f(q-1))+1]$ (Do it recursively)

Case 4: if $P[q] \neq P[f(q-1)+1]$ and $f(q-1)==0$ then $f[q]=0$.

Example : abc abc abb

abc abc   f(8)=5

abc   f(5)=2

a   f(2)=0

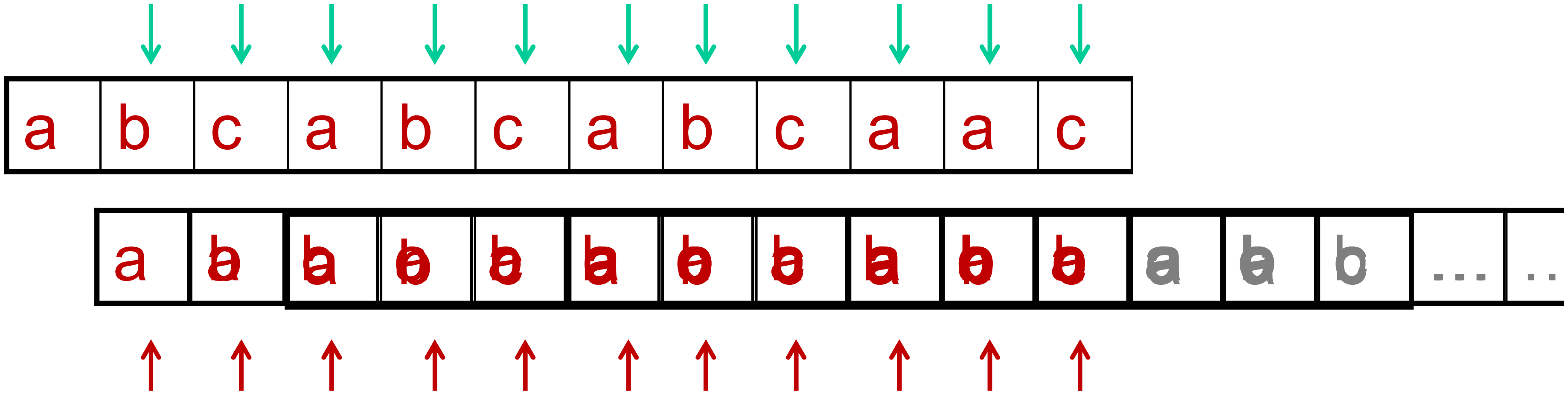
i:	1	2	3	4	5	6	7	8	9
f(i):	0	0	0	1	2	3	4	5	0

The algorithm (code) to compute failure function

```
1.   m=|P|;
2.   f(1)=0;
3.   k=0;
4.   for q=2 to |P| do
     {
5.       k=f(q-1);
6.       if(k>0 and P[k+1]!=P[q])
         {       k=f(k);   goto 6; }
7.       if(k>0 and P[k+1]==P[q])
         {       f[q]=k+1;         }
8.       if(k==0)
         {  if(P[k+1]==P[q] f[q]=1; else f[q]=0;}
     }
```

Another version

```
m=|P|;
f(1)=0;
k=0;
for q=2 to |P| do
{
    k=f(q-1);
    while(k>0 and P[k+1]!=P[q]) do
    {
        k=f(k);
    }
    if(P[k+1]==P[q]) then k=k+1;
    f[q]=k;
}
```



$f(1)=0$; $f(2)=0$; $f(3)=0$ $f(4)=1$;

$f(5)=2$; $f(6)=3$; $f(7)=4$; $f(8)=5$;

$f(9)=6$; $f(10)=7$; $f(11)=1$ $f(12)=0$

Running time complexity (Fun Part, not required)

The running time of failure function construction algorithm is $O(|P|)$. (The proof is similar to that for scan algorithm.)

Total running time complexity

The total complexity for failure function construction and scan algorithm is $O(|P|+|T|)$.

- P=a b c a b c a b c a a c. Compute its failure function

q	1	2	3	4	5	6	7	8	9	10	11	12
	a	b	c	a	b	c	a	b	c	a	a	c

- Solution:
- q= 1, $f(1) = 0$
- q= 2, P(q)=b ≠P(f(q-1)+1)=P(0+1)=a, f(2)=0
- q= 3, P(q)=c ≠P(f(q-1)+1)= P(0+1)=a, f(3)=0
- q= 4, P(q)=a=P(f(q-1)+1)= P(0+1)= a, f(4)=1
- q= 5 P(q)=b =P(f(q-1)+1)=P(1+1)=b, f(5)=2
- q= 6 P(q)=c =P(f(q-1)+1)=P(2+1)=b, f(6)=3
- …
- q=10, …. f(10)=7
- q=11, $P(q) = a \neq P(f(q-1)+1) = P(7+1) = b$,
- $P(q) = a \neq P\big(f\big(f(q-1)\big)+1\big) = P(f(7)+1) = P(4+1) = b$

  $P(q) = a \neq P\big(ff\big(f(q-1)\big)+1\big) = P(f(4)+1) = P(1+1) = b$

  $P(q) = a = P\big(fff\big(f(q-1)\big)+1\big) = P(f(1)+1) = P(0+1) = a$

  f(11)=1
- q=12, $P(q) = c \neq P(f(q-1)+1)$=P(1+1)=b
- $P(q) = c \neq P(ff(q-1)+1)$=P(f(1)+1)=P(0+1)=a.
- f(12)=0.

Summary

- String Matching Problem
- KMP algorithm O(|P|+|T|)
- Failure Function

Case 1: f(1) is always 0.

Case 2: if P[q]==P[f(q-1)+1] then f(q)=f(q-1)+1.

Case 3: if P[q]≠P[f(q-1)+1] and f(q-1)≠0 then consider P[q] ?= P[f(f(q-1))+1] (Do it recursively)

Case 4: if P[q] ≠ P[f(q-1)+1] and f(q-1)==0 then f[q]=0.