
Searching for a pattern. Knuth-Morris-Pratt

Lecture 2.

Motivation

“In a very real sense, molecular biology is all about sequences. It tries to reduce complex biochemical phenomena to interaction between defined sequences”

G. Von Heijne. Sequence analysis in molecular biology: treasure trove or trivial pursuit. Academic press, 1987

Examples

- Finding the overlaps during the sequence assembly
 - Finding *STS* – Sequence Tagged Sites – unique sequences used to map the positions of the fragments in the genome
 - Finding *EST* – Expressed Sequence Tags – STSs of protein-coding DNA – to locate genes inside the entire sequenced genome
-

Useful definitions

- A *string* S of length N is an ordered list of N elements written contiguously from left to right
 - The elements are called *symbols* or *characters*
 - $S[i\dots j]$ is a contiguous *substring* of S starting at position i and ending at position j of S
 - $S[1\dots j]$ is a *prefix* of S starting at position **1** and ending at position j
 - $S[i\dots N]$ is a *suffix* of S starting at position i and running till the last character of S
 - $S[i\dots j]$ is an *empty string* if $i > j$
 - A *proper* substring, prefix, suffix of S is respectively a substring, prefix, suffix that is neither the entire string S nor the empty string
-

Pattern matching problem

- Given a string P (of length M) called the *pattern* and a longer string T (of length N) called the *text*, find all occurrences, if any, of pattern P in text T
-

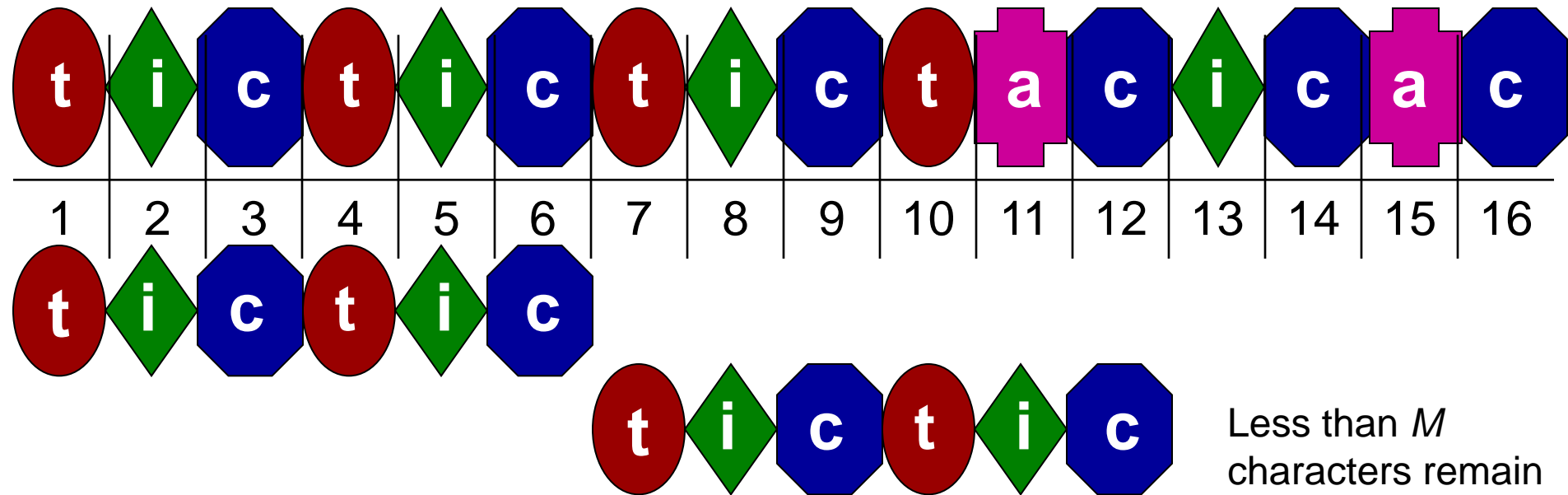
Naïve method – time complexity

- Naïve method is to compare the characters of the pattern starting from each of N positions of the text
- In the worst case, it requires $O(MN)$ character comparisons, exactly $M(N-M+1)$, for example, for $T=aaaaaaaaaa$ ($N=10$) and $P=aaa$ ($M=3$) there are 24 character comparisons

Naïve method – time complexity

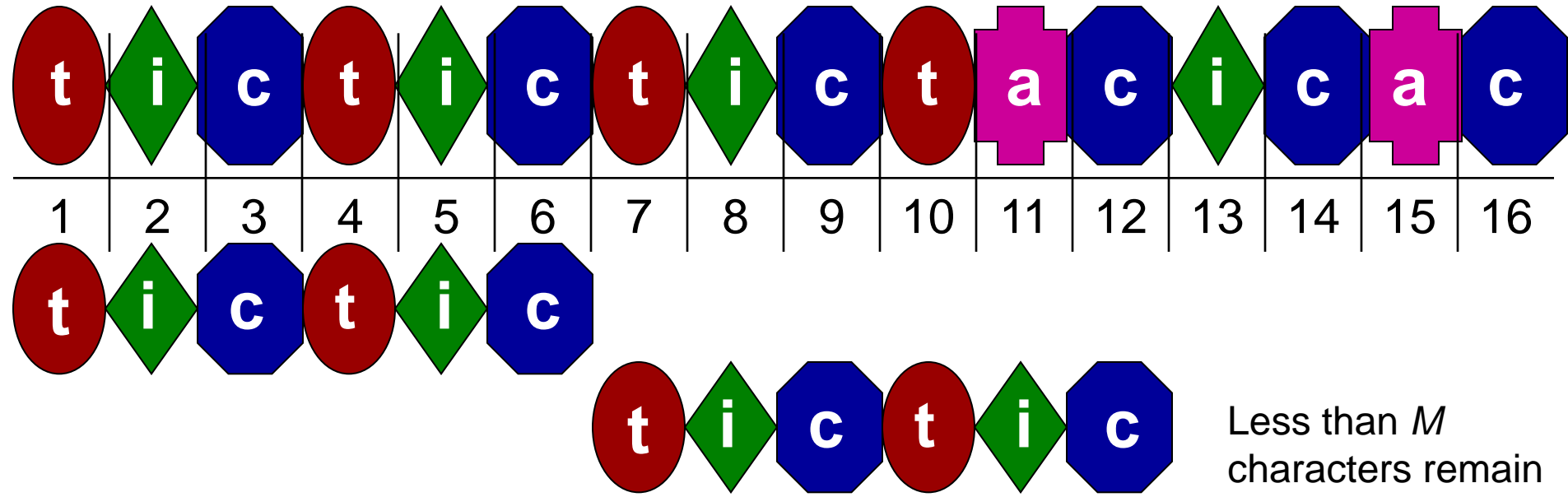
- In the worst case, we start from each position i of T (there are N such positions), and for each such position check, in the worst case, all M characters of P
- A standard fetching time from sequential RAM is 358 MB values per second ([ref](#)).
- If we have 10 random sets of sequenced fragments from the 3 GB-length human genome, then we need to search the text of a total size $3 \cdot 10^{10}$, which can be sequentially accessed with approximately $3 \cdot 10^8$ values per second. We will spend 100 seconds on a linear time algorithm, but for the worst case we need to multiply it by the value of M , which can be as large as 800.
- *Grep* search program (based on a linear-time algorithm), for example, requires about 2 minutes when searching for a string of length 10 in a 3 GB text (on an average desktop machine).
- We want the pattern search algorithm to perform in a *linear time*

Our dream goal: each character of T is accessed only once

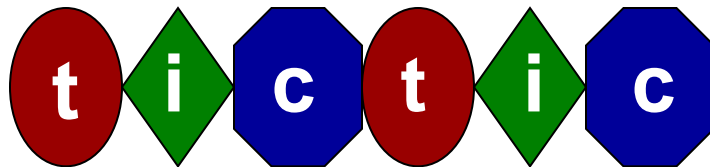


Is this algorithm correct?

Incorrect algorithm



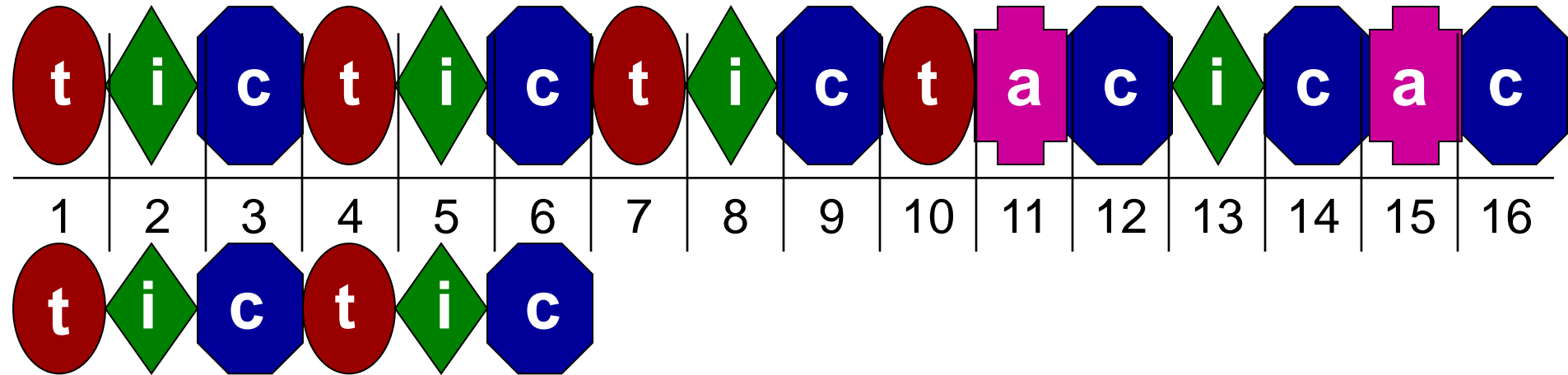
No, we have missed an occurrence of P starting at position 4



Knuth-Morris-Pratt (KMP) idea

- When we have aligned the prefix of P with k characters of T , we know what characters are in T up to the current position (they are equal to those of the prefix $P[1\dots k]$ of P)
 - From this information we can deduce the place where to start the next comparison
-

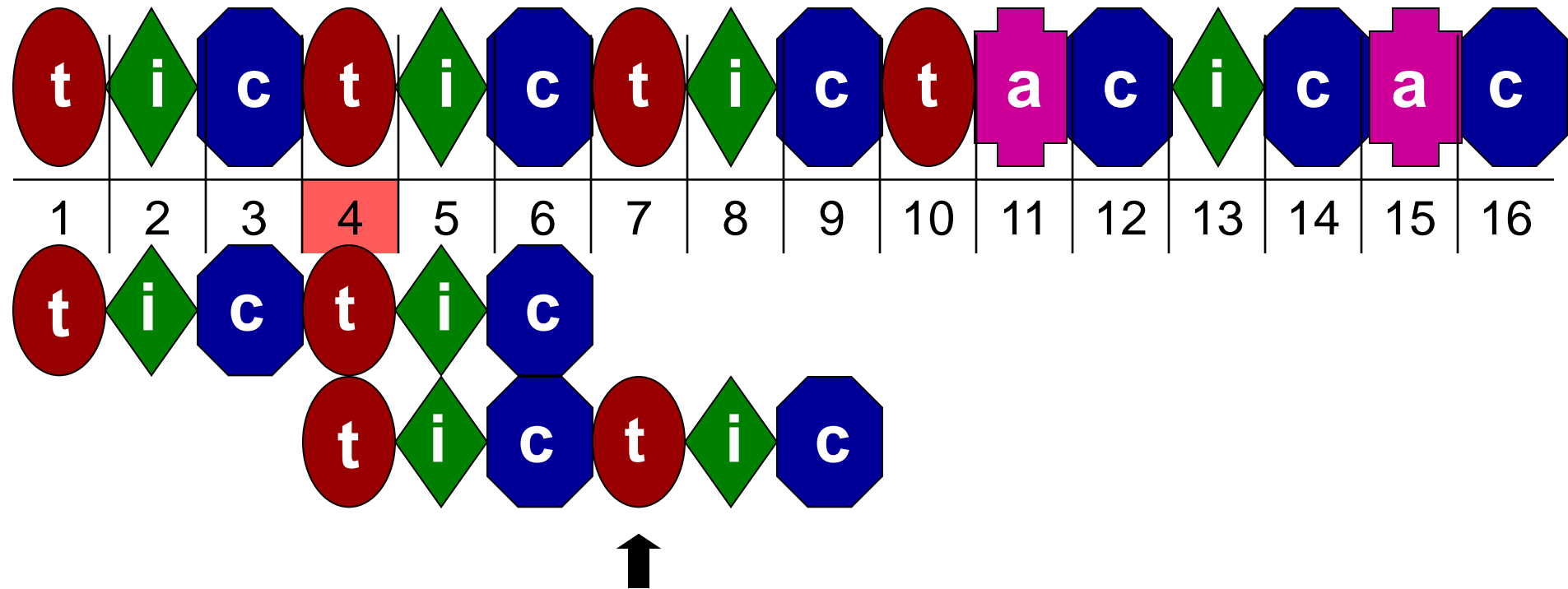
KMP intuition



We have aligned 6 characters

The next occurrence of a pattern has to start with *tic* and we know that the last characters of a match were *tic*, since the suffix of P starting at position 4 is equal to a prefix of P of length 3

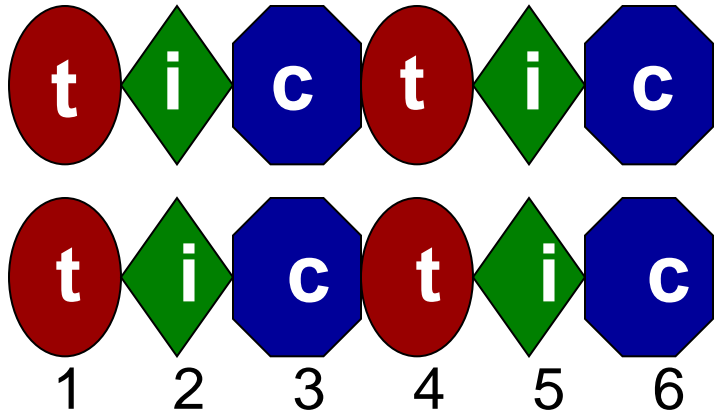
KMP intuition



Therefore we can set a start of the next comparison to 3 positions backwards from the current position (red cell), and we don't need to compare the first 3 characters of P again, since we know that they match

Thus, we can continue the comparison from the next character of P (and T)

KMP intuition – overlap function for P

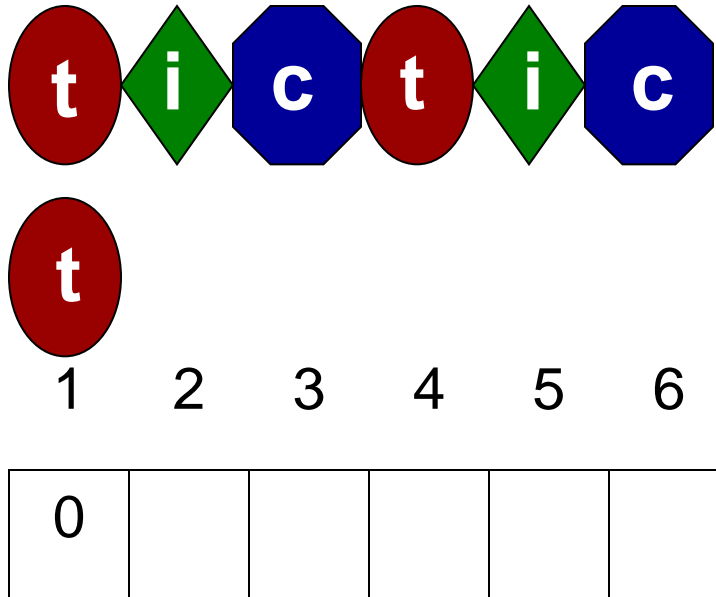


In order to know where to position the start of the next comparison, we need to know the values of an *overlap function* for P , namely:

For each position j in P , the maximal length of a substring which is at the same time a proper prefix of P and the proper suffix of substring $P[1, j]$.

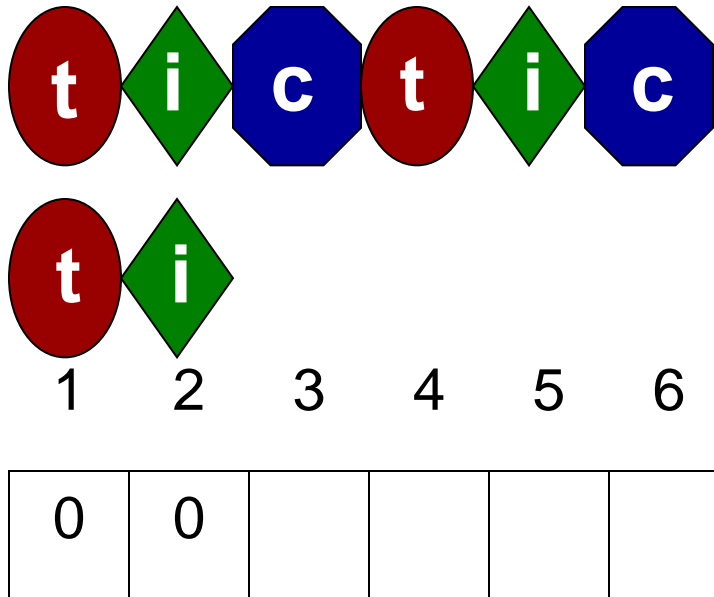
Before we start the search, we need to compute an overlap function for P – we need to preprocess pattern P

KMP intuition – overlap function for P



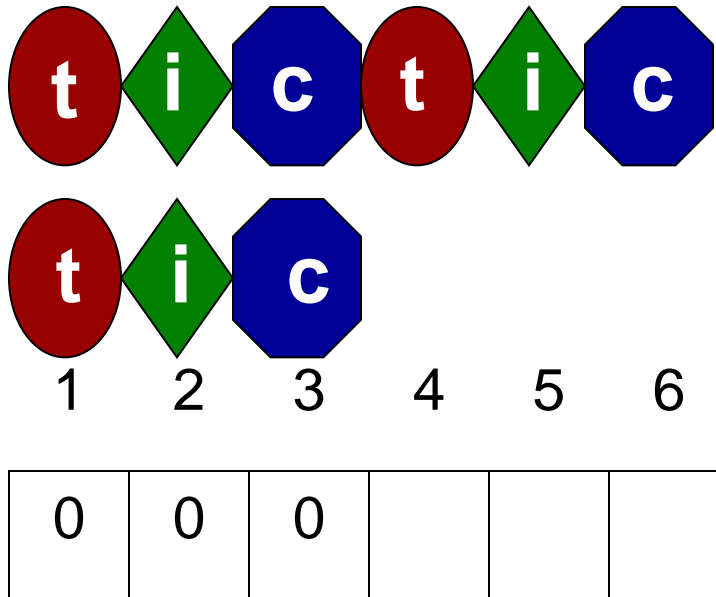
For $j=1$, $OF=0$ (t is not a proper suffix of a substring t , but the entire t)

KMP intuition – overlap function for P



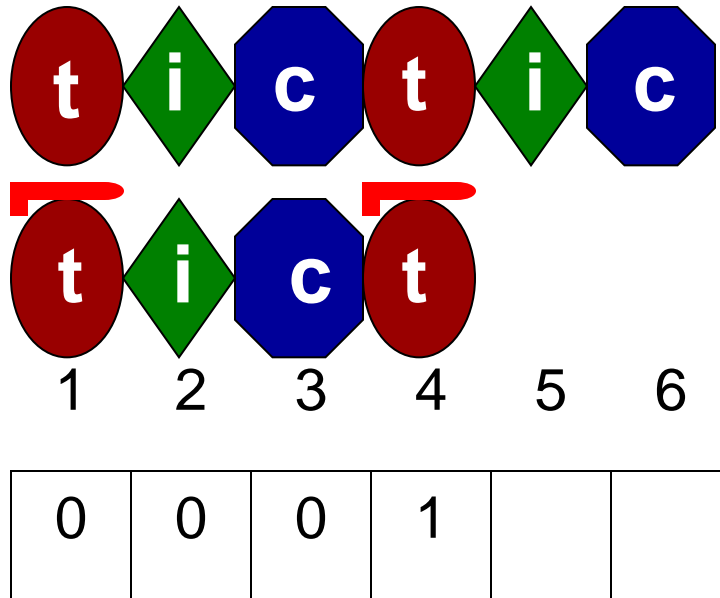
For $j=2$, $OF=0$ (the only proper suffix of ti , the suffix i , does not have any overlap with the prefix t of ti)

KMP intuition – overlap function for P



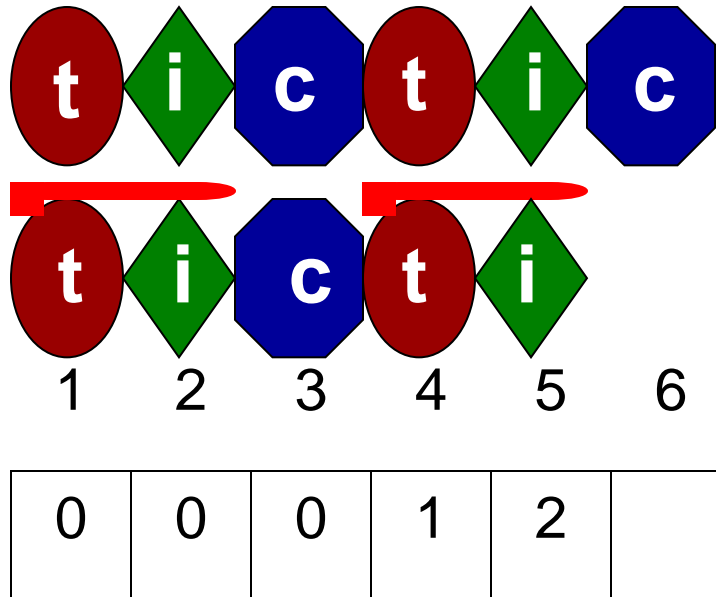
For $j=3$, $OF=0$ (suffixes *ic*, *c* do not have an overlap)

KMP intuition – overlap function for P



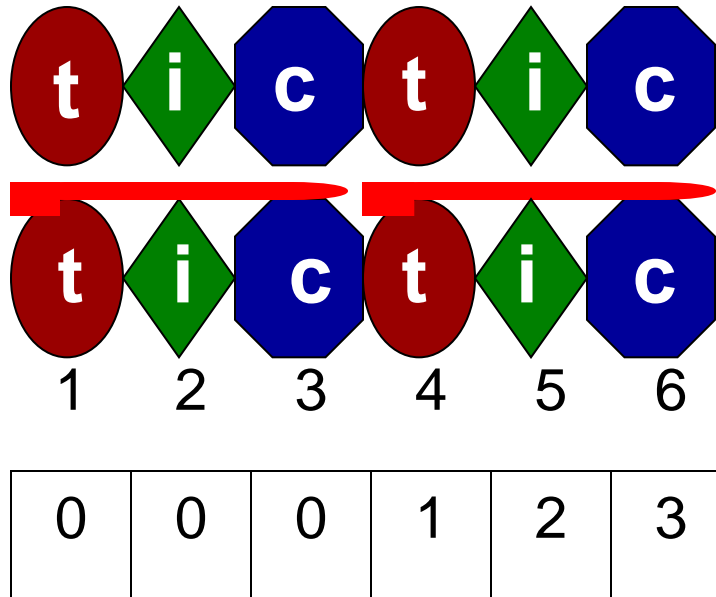
For $j=4$, $OF=1$ (t is a proper suffix of a substring $tict$, and the prefix of P)

KMP intuition – overlap function for P



For $j=5$, $OF=2$ (*ti* is a proper suffix of a substring *ticti*, and the prefix of P)

KMP intuition – overlap function for P

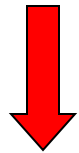
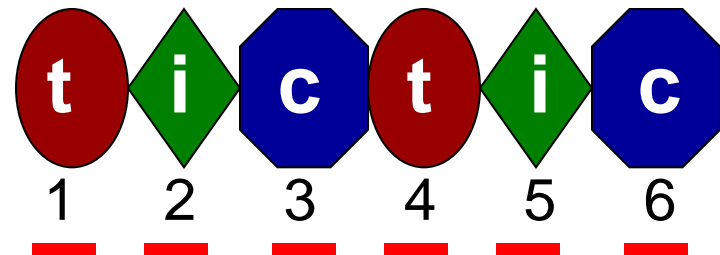
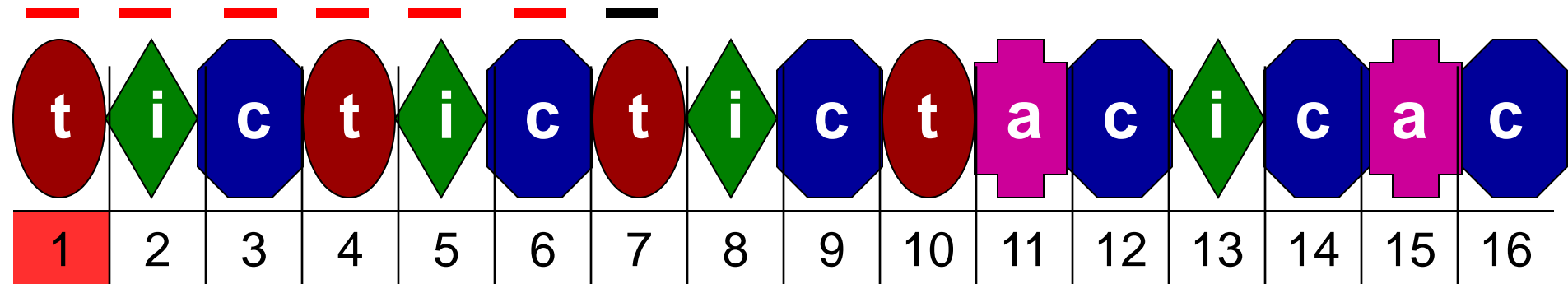


For $j=6$, $OF=3$ (*tic* is a proper suffix of a substring *tictic*, and the prefix of P)

Assume, for now, that the OF values for P are computed

KMP search

$i=7$

$j=7$

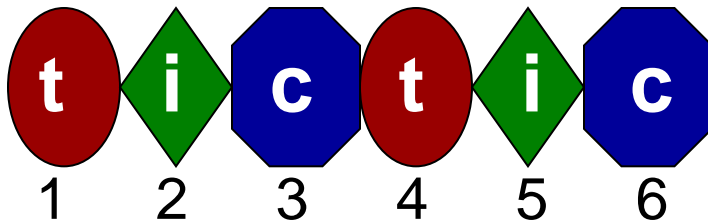
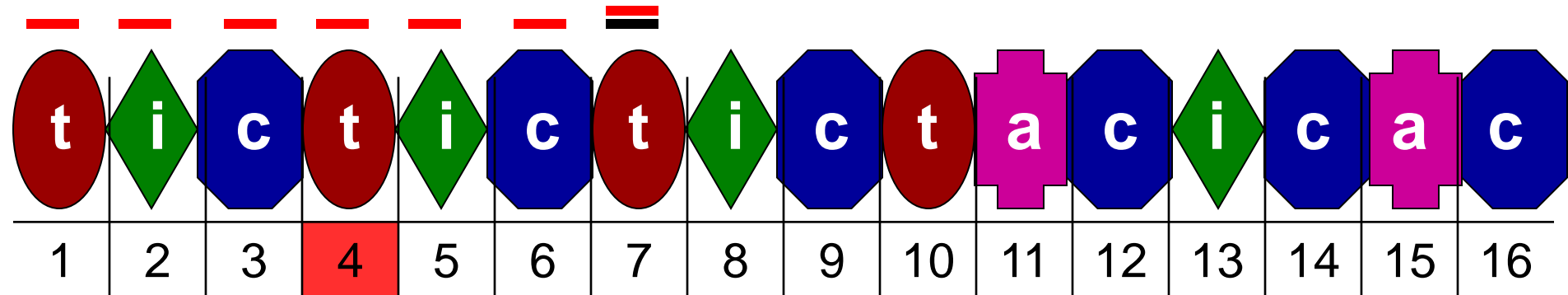


Report 1

0	0	0	1	2	3
---	---	---	---	---	---

Consult $OF(6)=3$ it tells how many positions backward from i the next comparison starts: $k=i-OF(j-1)$

KMP search

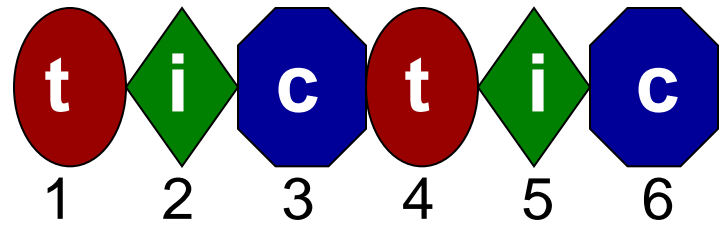
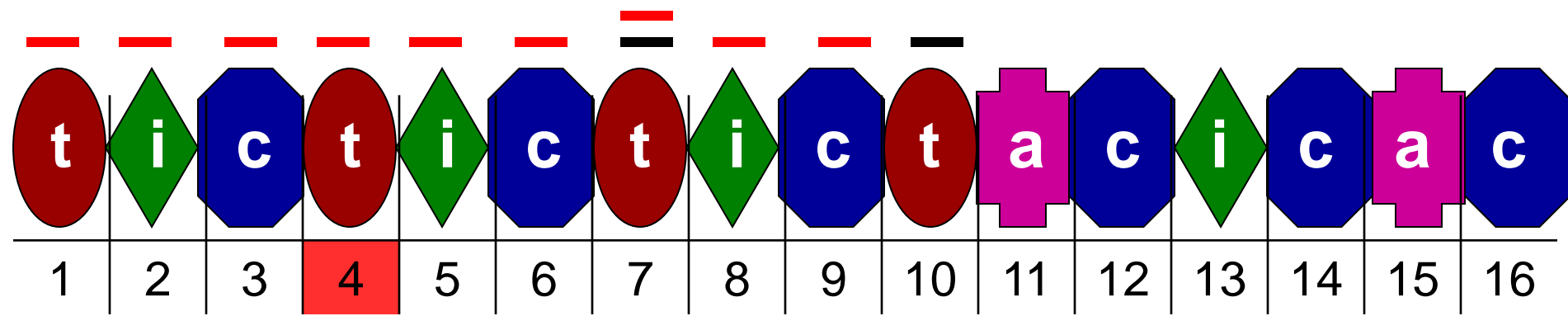


0	0	0	1	2	3
---	---	---	---	---	---

No need to compare these 3 characters, we know that they match – we just compared them

KMP search

$i=10$



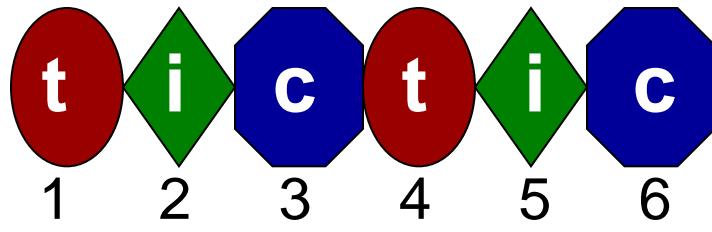
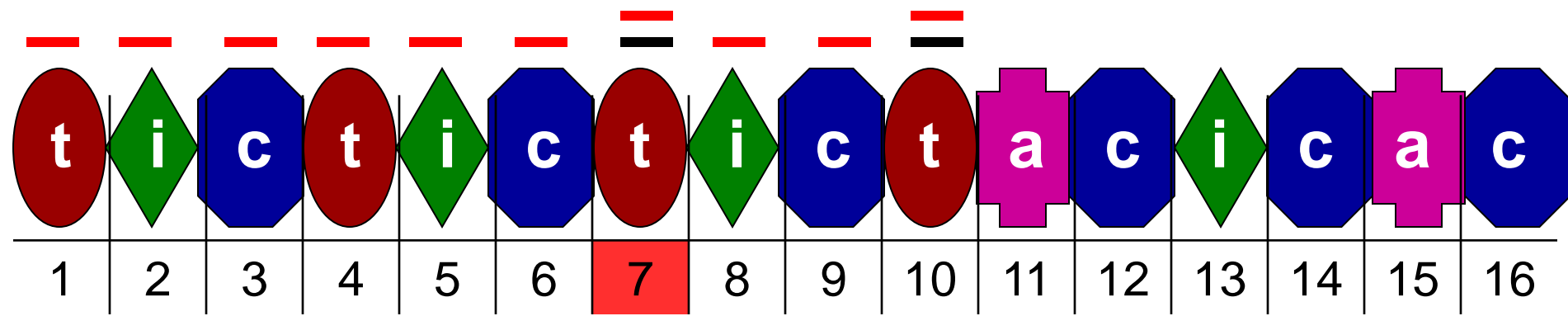
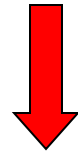
$j=7$

Report **4**

0	0	0	1	2	3
---	---	---	---	---	---

Consult $OF(6)=3$ it tells how many positions backward from i the next comparison starts: $k=i-OF(j)+1$

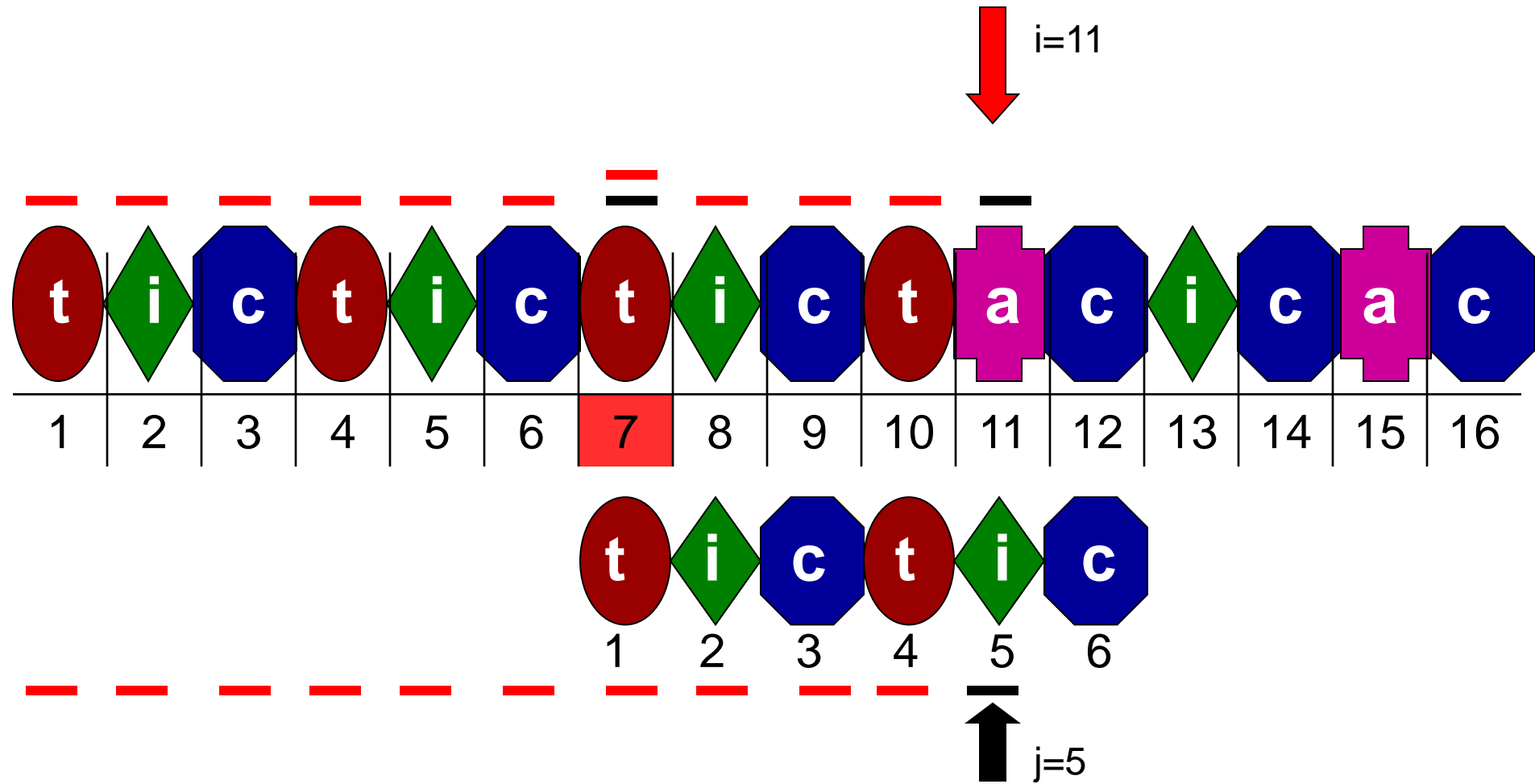
KMP search



0	0	0	1	2	3
---	---	---	---	---	---

Continue comparing T[10] and P[4]

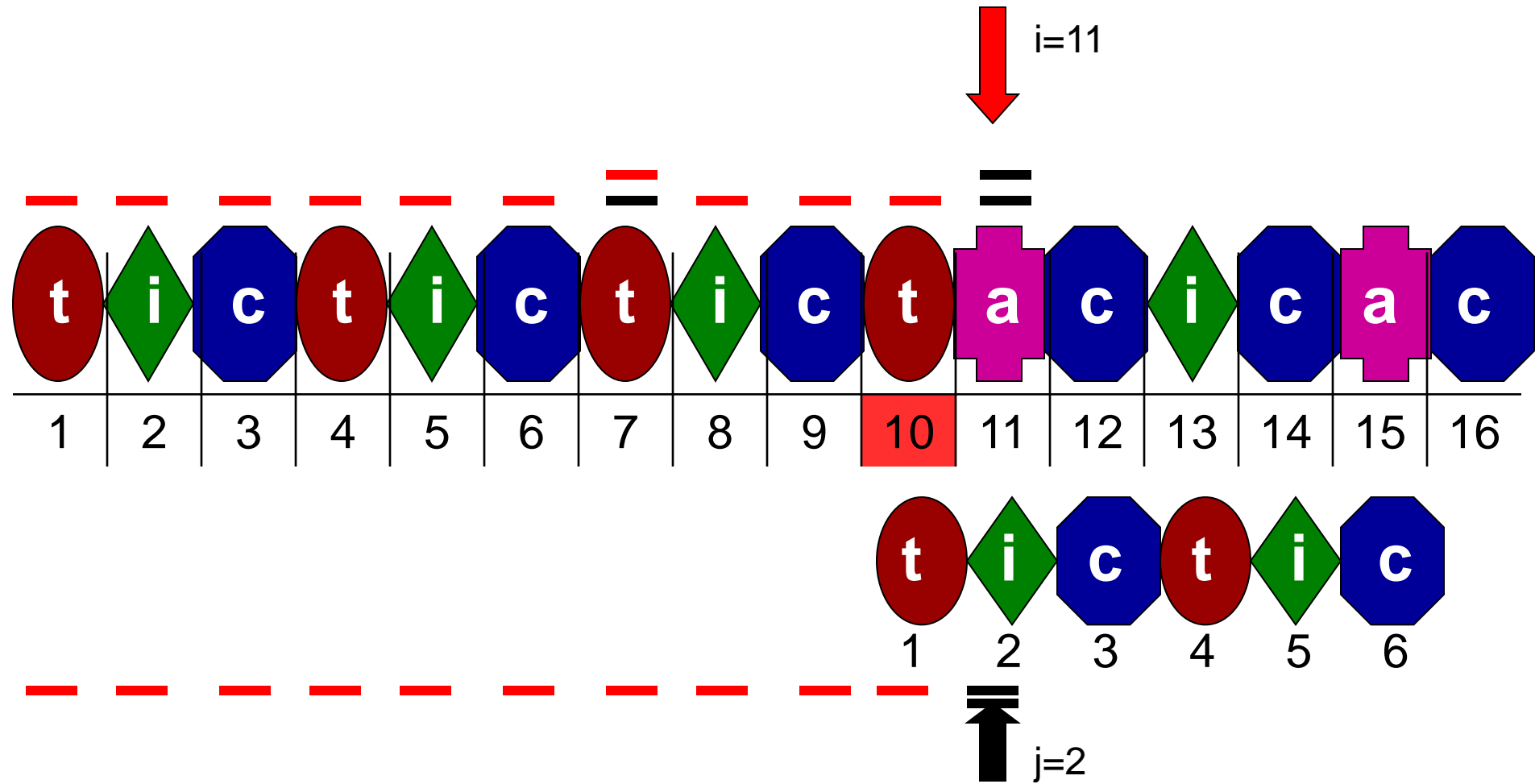
KMP search



0	0	0	1	2	3
---	---	---	---	---	---

T[11] and P[5] do not match. Consult OF(4)=1. next potential match can start at $i - OF(j) = 10$, and the first character is already matched

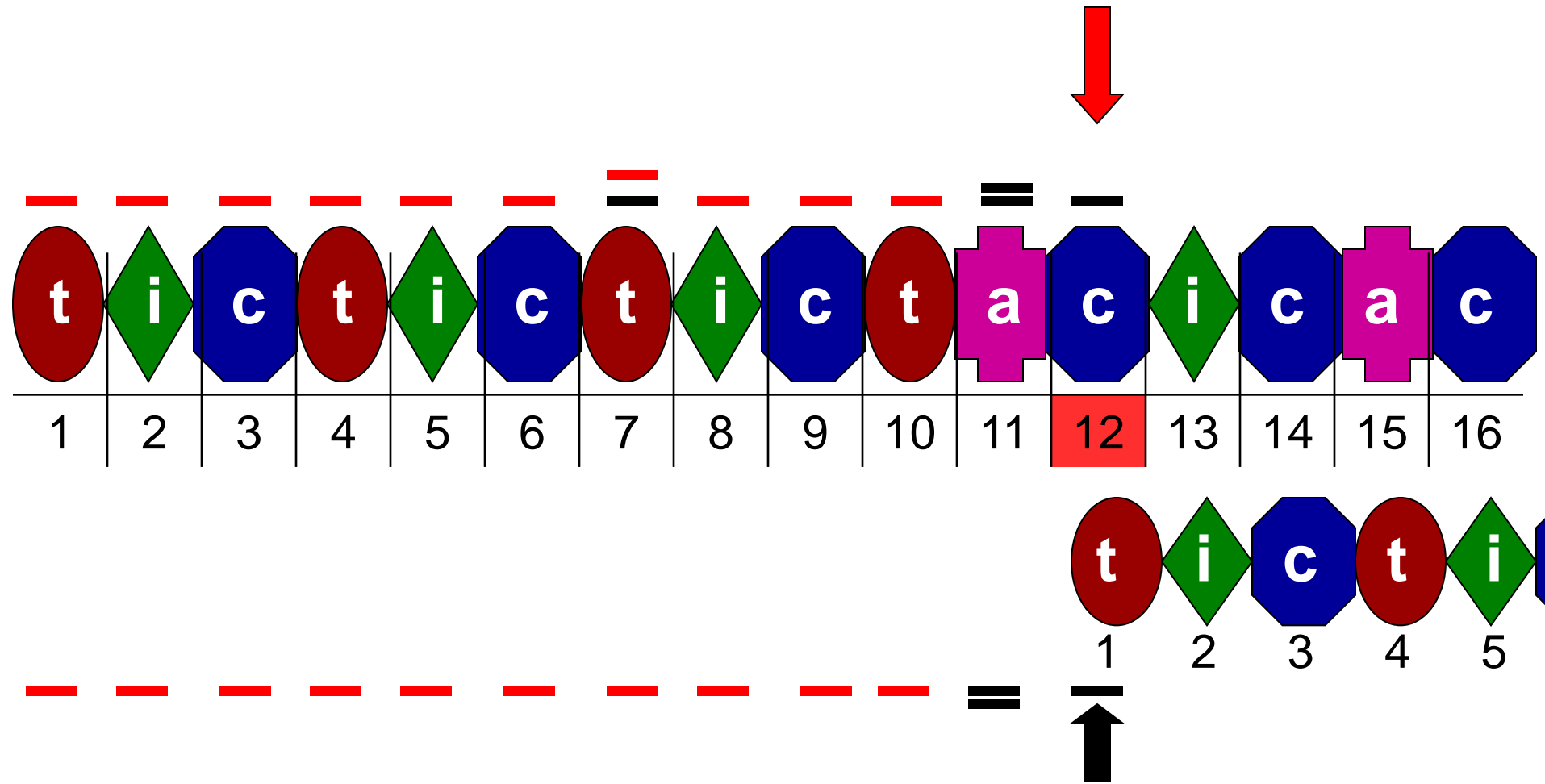
KMP search



0	0	0	1	2	3
---	---	---	---	---	---

Here we only matched till the position $j=2$, the value $OF(1)=0$, therefore we are not shifting the start of the comparison backwards but starting from the next $i=12$ etc...

KMP search



0	0	0	1	2	3
---	---	---	---	---	---

If T would be larger, we continue in a similar manner, never accessing the characters of T more than twice

KMP – from an intuition to the algorithm

We need 3 pointers (3 only for clarity, could work with 2):

- pointer i will point to the current character of text T of length N
- pointer j will point to the current character of pattern P of length M
- pointer k will point to the start of a current comparison in T

in the beginning $i=1, j=1, k=1$

$i:=1 \ j:=1 \ k:=1$

KMP - from an intuition to the algorithm

if we have enough symbols in T to match P starting from position k, then we *continue* to compare the corresponding characters of P and T

```
i:=1 j:=1 k:=1  
while: N-k>=M
```

KMP - from an intuition to the algorithm

we continue matching symbols of P
while they match or until we reached
the end of P

```
i:=1 j:=1 k:=1
while: N-k>=M
    while: j ≤ M and T[i]=P[j]
        i:=i+1
        j:=j+1
```

KMP - from an intuition to the algorithm

If we reached the end of P, we found our match starting at position k of T

```
i:=1 j:=1 k:=1
while: N-k>=M
    while: j ≤ M and T[i]=P[j]
        i:=i+1
        j:=j+1
    if j>M then output k
```

KMP - from an intuition to the algorithm

Now we need to find where to start the next comparison

if there was an overlap $OL(j-1)$, then

- set the start of a new comparison (k) that many steps backwards from the current position in T as the value $OL(j-1)$
- set j to the position $OL(j-1)+1$ in T (we know that the previous characters match)
- i remains unchanged, since now we are going to compare it with the symbol at a different position of P

```
i:=1 j:=1 k:=1
```

```
while: N-k>=M
```

```
    while: j ≤ M and T[i]=P[j]
```

```
        i:=i+1
```

```
        j:=j+1
```

```
    if j>M then output k
```

```
    if  $OF(j-1)>0$ 
```

```
        k:=i- $OF(j-1)$ 
```

```
        j:= $OF(j-1)+1$ 
```

KMP - from an intuition to the algorithm

if the value of an overlap function is zero (do not need to check backwards), then

- advance i to the next position
- set start of a comparison k to i
- set j to 1

```
i:=1 j:=1 k:=1
```

```
while: N-k>=M
```

```
    while: j ≤ M and T[i]=P[j]
```

```
        i:=i+1
```

```
        j:=j+1
```

```
    if j>M then output k
```

```
    if OF(j-1)>0
```

```
        k:=i-OF(j-1)
```

```
        j:=OF(j-1)+1
```

```
    else
```

```
        k:=i
```

```
        j:=1
```


KMP - from an intuition to the algorithm

If fully matched - need to advance i

```
i:=1 j:=1 k:=1
while: N-k>=M
    while: j ≤ M and T[i]=P[j]
        i:=i+1
        j:=j+1
    if j>M then output k
    if OF(j-1)>0
        k:=i-OF(j-1)
        j:=OF(j-1)+1
    else
        if i=k then
            i:=i+1
        k:=i
        j:=1
```

KMP algorithm time complexity

- The number of character comparisons in KMP algorithm is at most $2N$
 - Divide the algorithm into compare/shift phases, where a single phase consists of the comparisons done between 2 successive shifts. During 2 consecutive shifts, at most 2 comparisons are done for each character of T .
 - Since pattern is never shifted left, the total number of character comparisons is bounded by $N+s$, where s is the total number of shifts. But $s < N$, since after N shifts the right end of P is certainly to the right of the right end of T , so the total number of comparisons done is bounded by $2N$

A worst-case example – iterations 1,2

1	1	1	1	1					
a	a	a	a	b	a	a	a	a	a
a	a	a	a	a					

We have aligned pattern P, by performing so far 1 character comparison for each of 5 characters of P

Now we need to restart the comparison from the position 2 of T

1	1	1	1	2					
a	a	a	a	b	a	a	a	a	a
	a	a	a	a	a				

References

- http://en.wikipedia.org/wiki/Knuth-Morris-Pratt_algorithm
 - <http://www.ics.uci.edu/~eppstein/161/960227.html>
 - Dan Gusfield. Algorithms on strings, trees, and sequences. Computer science and computational biology. Cambridge University press, 1999.
-