Advanced Data Structures

String Pattern Matching/Te t Search

What is Pattern Matching?

Definition:

given a te t string T and a pattern string P, find the pattern inside the te t

T: the rain in spain sta s mainl on the plain

P: n th

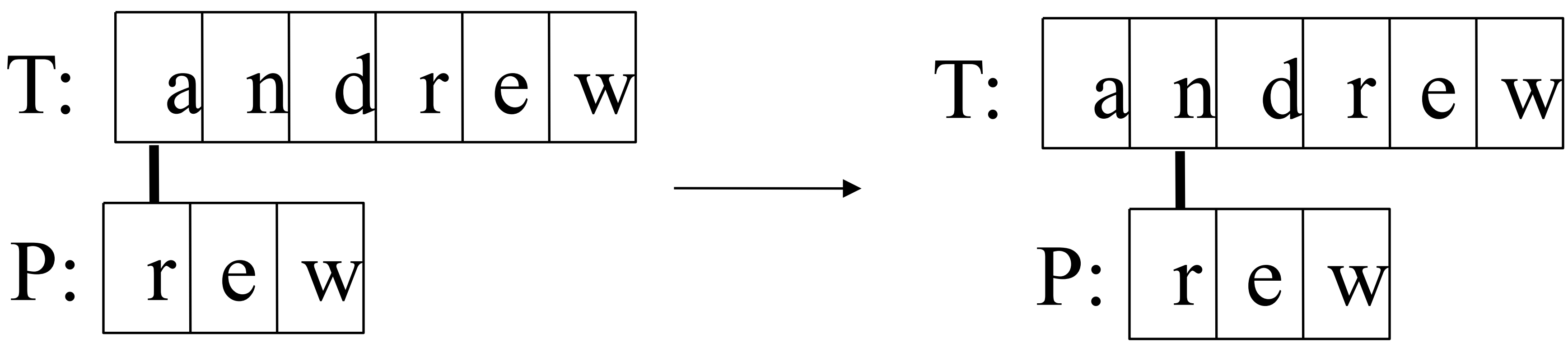
Te t search

- Pattern matching directl
  - Brute force
  - KMP
  - BM
- Regular e pressions (Not in this course)
- Indices for pattern matching
  - Inverted files

and

The Brute Force Algorithm

Check each position in the te t T to see if the pattern P starts in that position



P moves 1 char at a time through T

....

The brute force algorithm is fast when the alphabet of the te t is large

e.g. A..Z, a.. , 1..9, etc.

It is slower when the alphabet is small

e.g. 0, 1 (as in binar   files, image files, etc.)

continued

E ample of a worst case:

T: "aaaaaaaaaaaaaaaaaaaaaaaaaaaah"

P: "aaah"

E ample of a more average case:

T: "a string searching e ample is standard"

P: "store"

The KMP Algorithm

The Knuth-Morris-Pratt (KMP) algorithm looks for the pattern in the te t in a *left-to-right* order (like the brute force algorithm).

But it shifts the pattern more intelligentl than the brute force algorithm.

continued

Summar

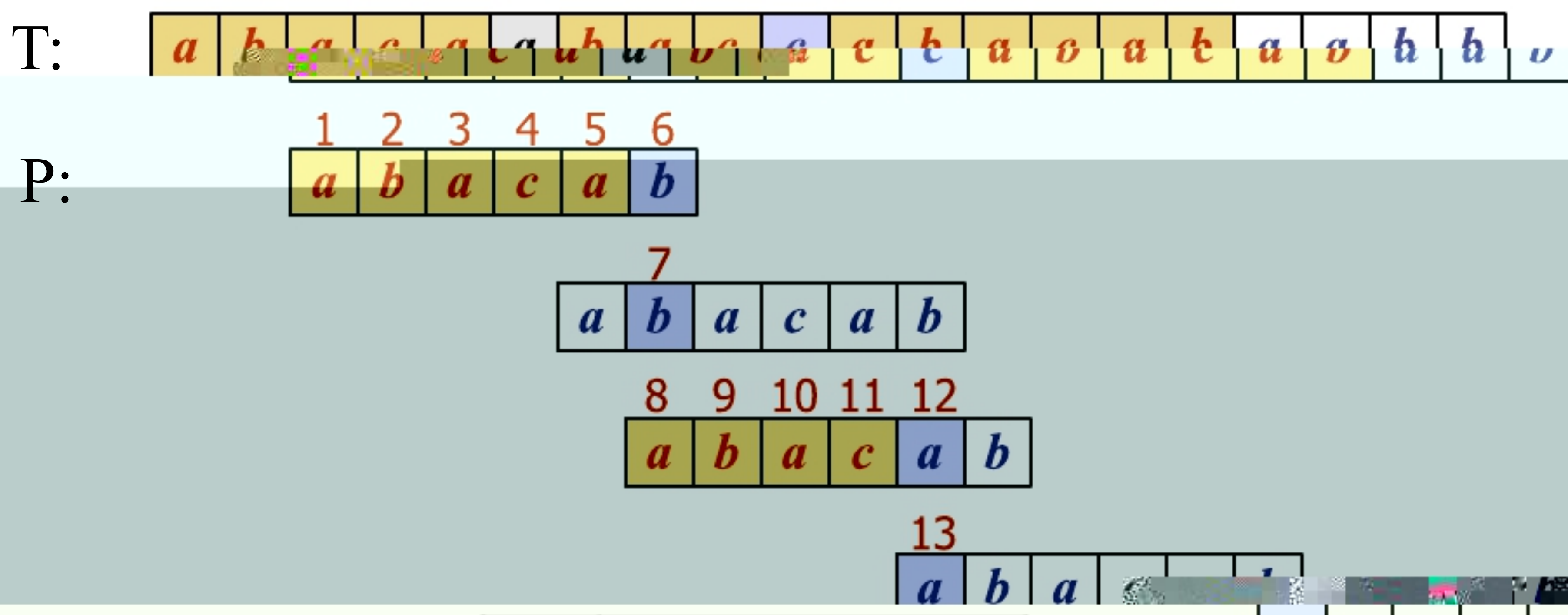
If a mismatch occurs between the te t and pattern P at P[j], what is the *most* we can shift the pattern to avoid wasteful comparisons?

Summar

If a mismatch occurs between the te t and pattern P at P[j], what is the *most* we can shift the pattern to avoid wasteful comparisons?

Answer: the largest prefi   of P[0 .. j-1] that is a suffi   of P[1 .. j-1]

E ample



k	0	1	2	3	4
F(*k*)	0	0	1	0	1

KMP Advantages

KMP runs in optimal time: O(m+n)

- ver   fast

The algorithm never needs to move backwards in the input te t, T

- this makes the algorithm good for processing ver   large files that are read in from e ternal devices or through a network stream

KMP Disadvantages

KMP doesn t work so well as the si e of the alphabet increases

- more chance of a mismatch (more possible mismatches)
- mismatches tend to occur earl   in the pattern, but KMP is faster when the mismatches occur later

A fast string searching algorithm. *Communications of the ACM*. Vol. 20 p.p. 762-772, 1977.

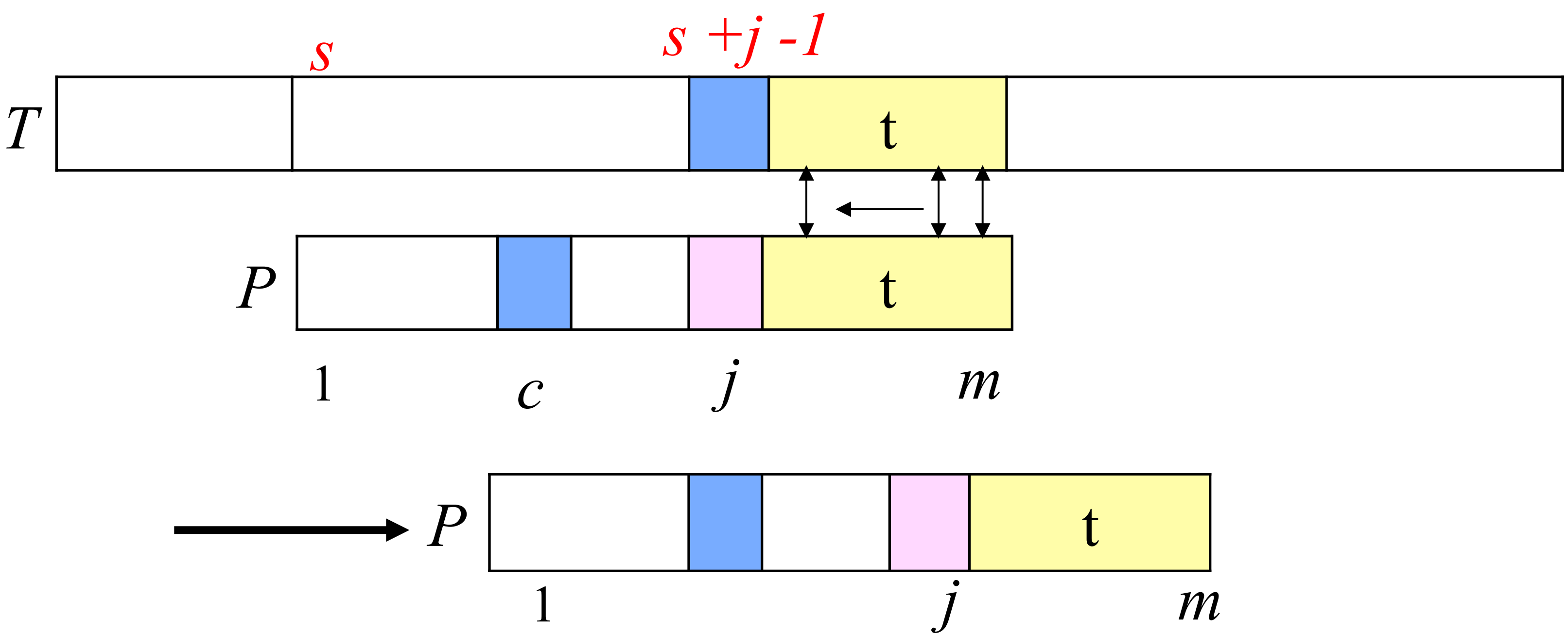
, . . and , . .

The algorithm compares the pattern *P* with the substring of sequence *T* within a sliding window in the - - .

The and are used to determine the movement of sliding window.

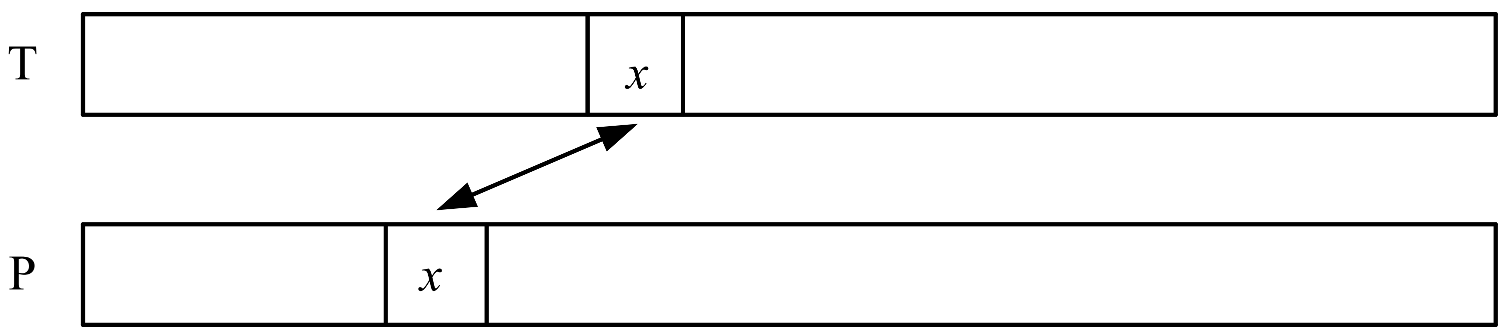
Suppose that P_1 is aligned to T_s now, and we perform a pair-wise comparing between te t T and pattern P from right to left. Assume that the first mismatch occurs when comparing T_{s+j-1} with P_j .

Since $T_{s+j-1} \neq P_j$, we move the pattern P to the right such that the largest position c in the left of P_j is equal to T_{s+j-1}. We can shift the pattern at least (j-c) positions right.

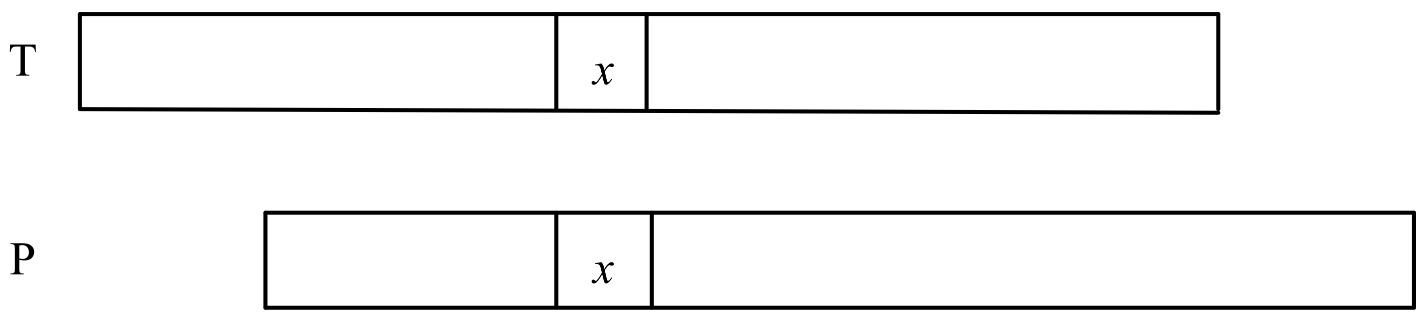


Bad character rule uses Rule 2-1 (Character Matching Rule).

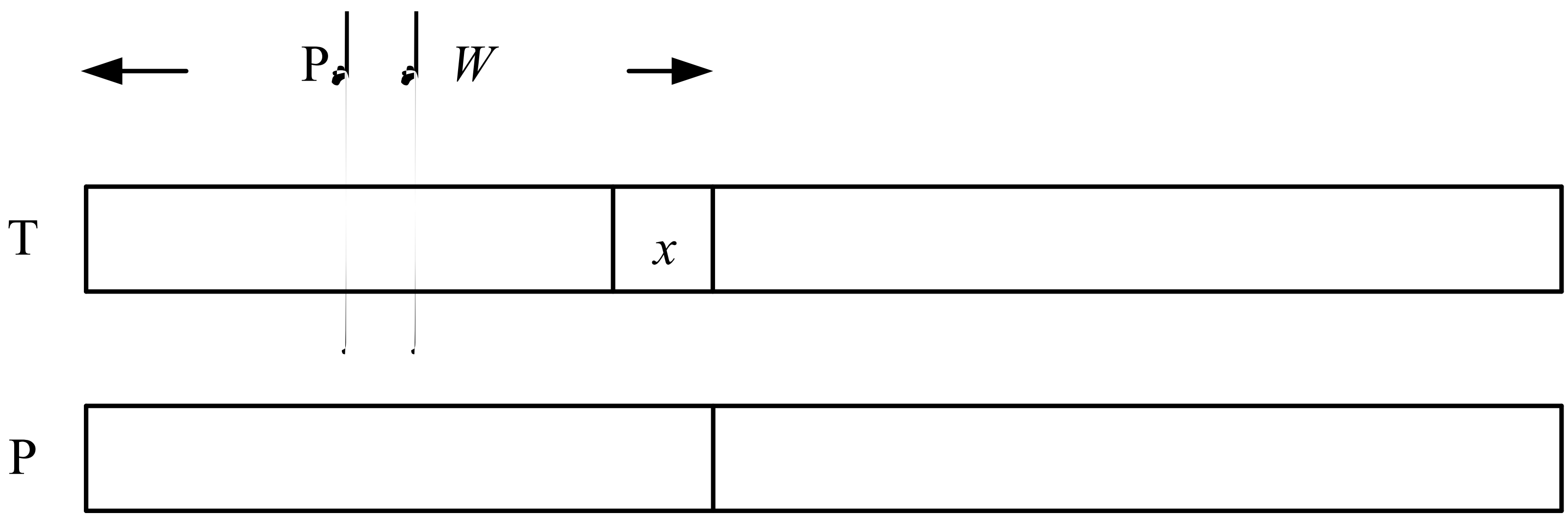
For an character *x* in *T*, find the nearest *x* in *P* which is to the left of *x* in *T*.



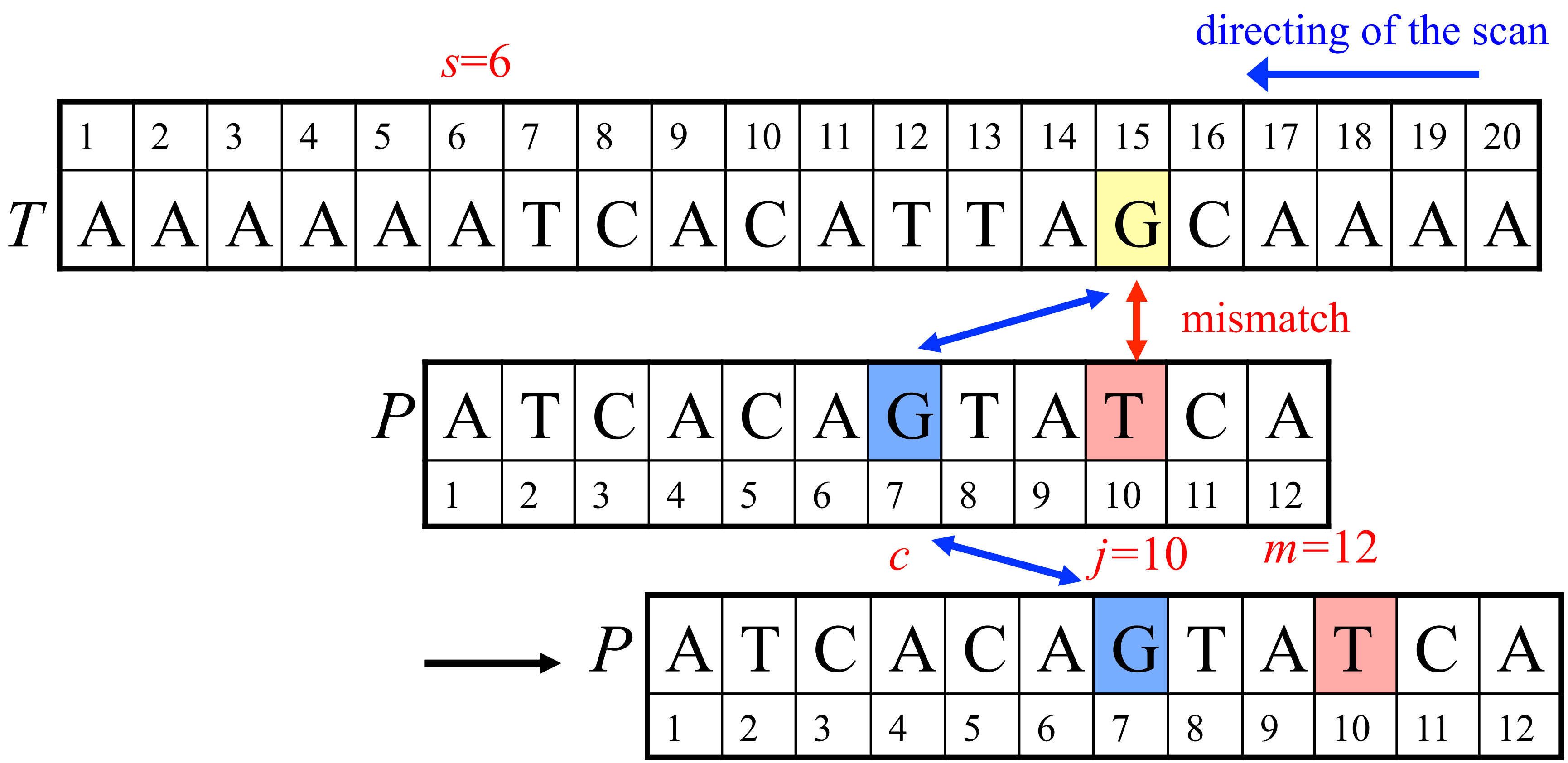
Case 1. If there is a *x* in *P* to the left of *T*, move *P* so that the two s match.



Case 2: If no such a x e ists in P, consider the partial window defined b x in T and the string to the left of it.

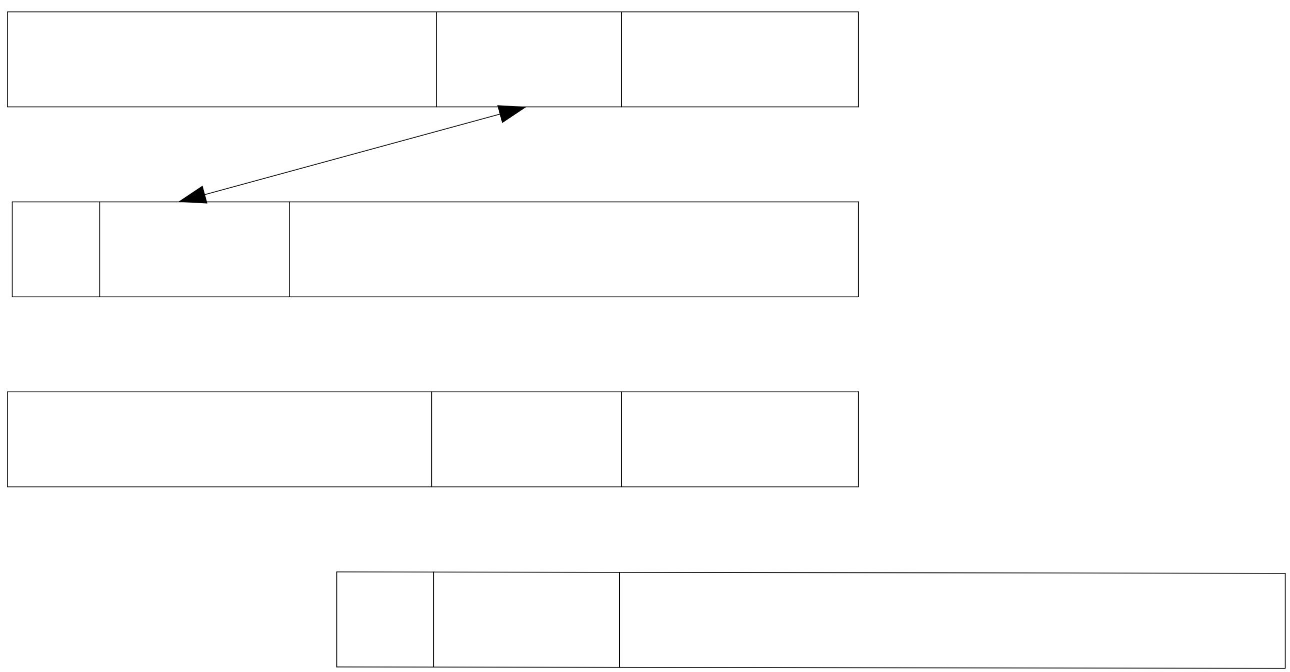


E : Suppose that P_1 is aligned to T_6 now. We compare pair-wise between T and P from right to left. Since $T_{16,17} = P_{11,12} =$ "CA" and $T_{15} =$ "G" $\neq P_{10} =$ "T". Therefore, we find the rightmost position c=7 in the left of P_{10} in P such that P_c is equal to "G" and we can move the window at least (10-7=3) positions.

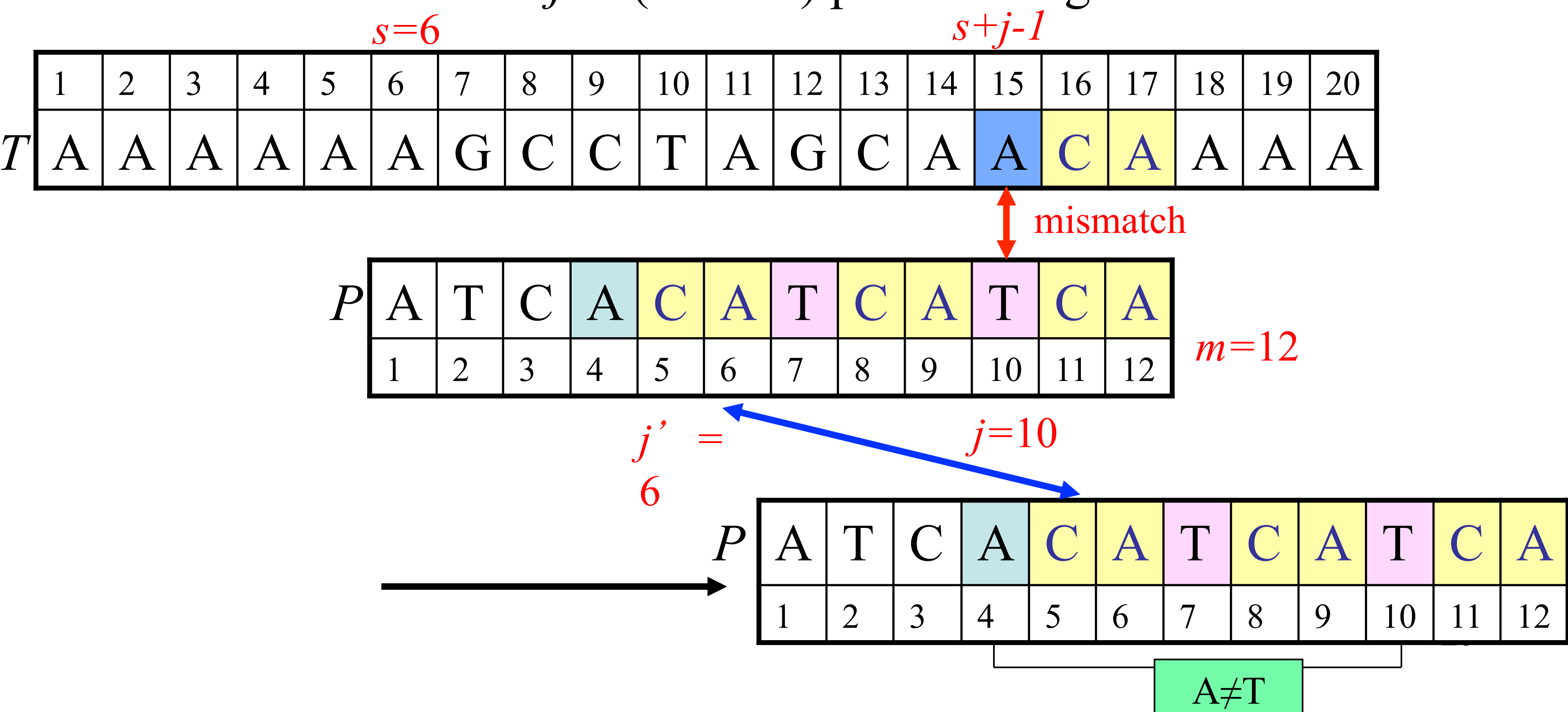


If a mismatch occurs in T_{s+j-1}, we match T_{s+j-1} with $P_{j'-m+j}$, where j' (m-j+1 ≦ j' < m) is the

For an substring *u* in T, find a nearest *u* in P which is to the left of it. If such a *u* in P e ists, move P; otherwise, we ma define a new partial window.

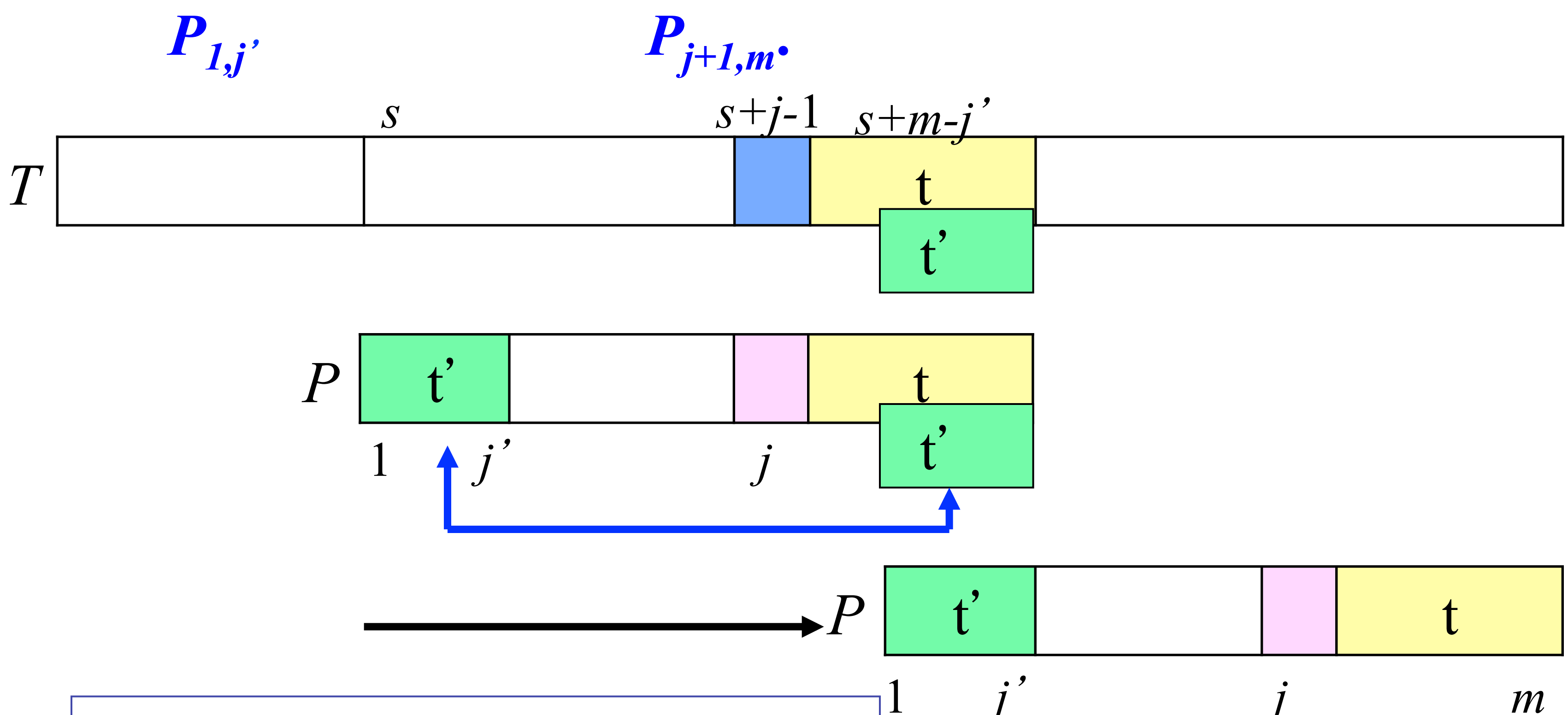


E : Suppose that P_1 is aligned to T_6 now. We compare pair-wise between P and T from right to left. Since $T_{16,17}$ = "CA" = $P_{11,12}$ and T_{15} = "A" $\neq P_{10}$ = "T". We find the substring "CA" in the left of P_{10} in P such that "CA" is the suffi of $P_{1,6}$ and the left character to this substring "CA" in P is not equal to P_{10} = "T". Therefore, we can move the window at least m-j' (12-6=6) positions right.



Good Suffi Rule 2 is used onl when Good Suffi Rule 1 can not be used. That is, t does not appear in P(1, j). Thus, t is in P.

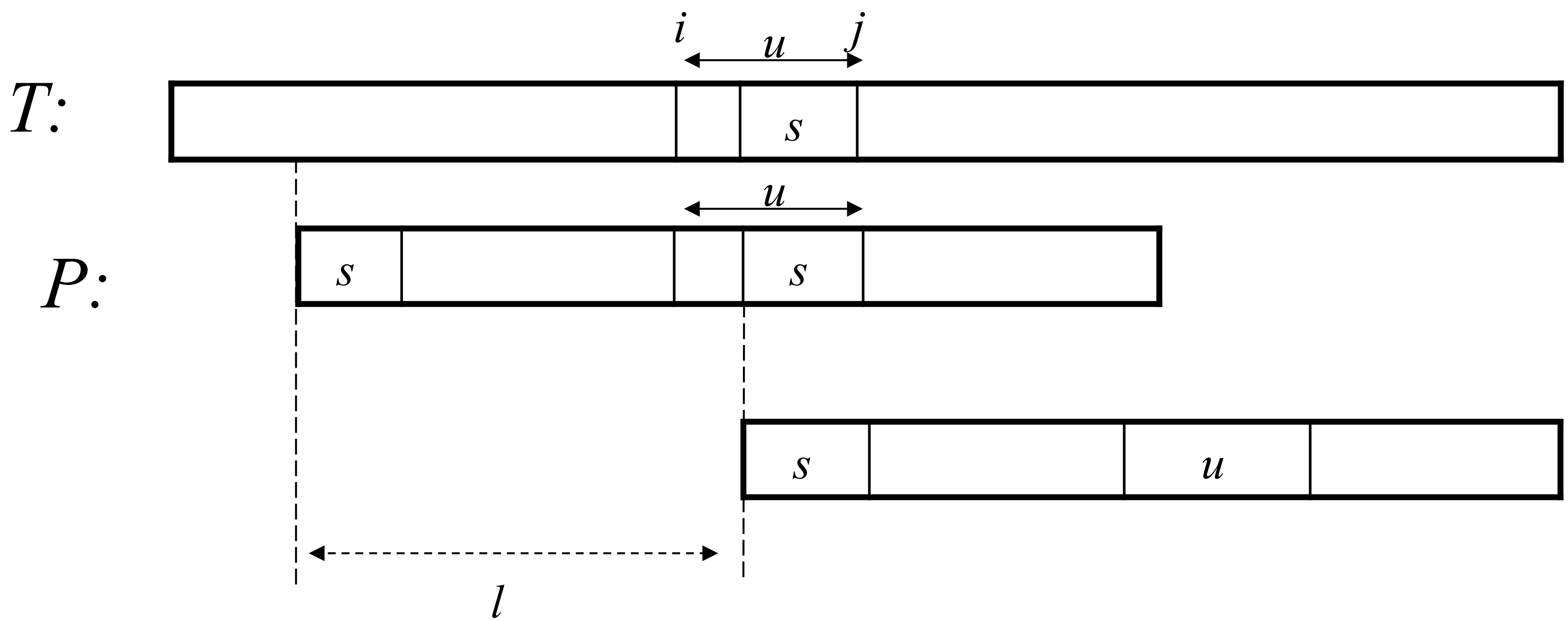
If a mismatch occurs in T_{s+j-1}, we match $T_{s+m-j'}$ with P_1, where j' ($1 \leqq j' \leqq m-j$) is such that



P.S. : t' is suffi of substring t.

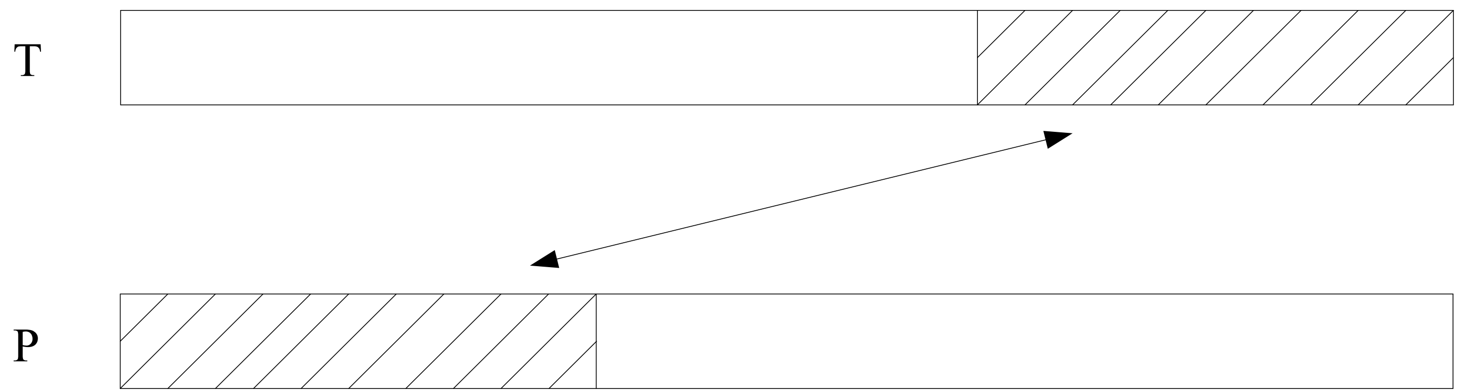
The substring u appears in P e actl   once.

If the substring u matches with $T_{i,j}$, no matter whether a mismatch occurs in some position of P or not, we can slide the window b l.



The string s is the longest prefi   of P which equals to a suffi   of u.

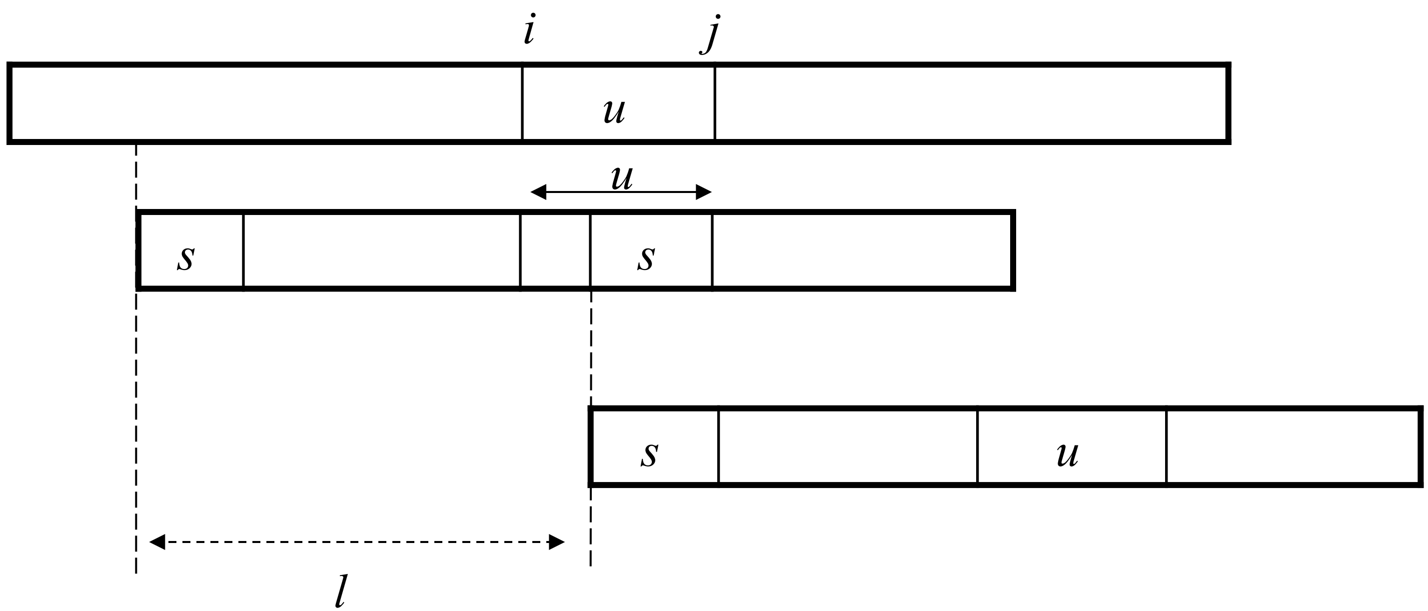
For a window to have an   chance to match a pattern, in some wa , there must be a suffi   of the window which is equal to a prefi   of the pattern.



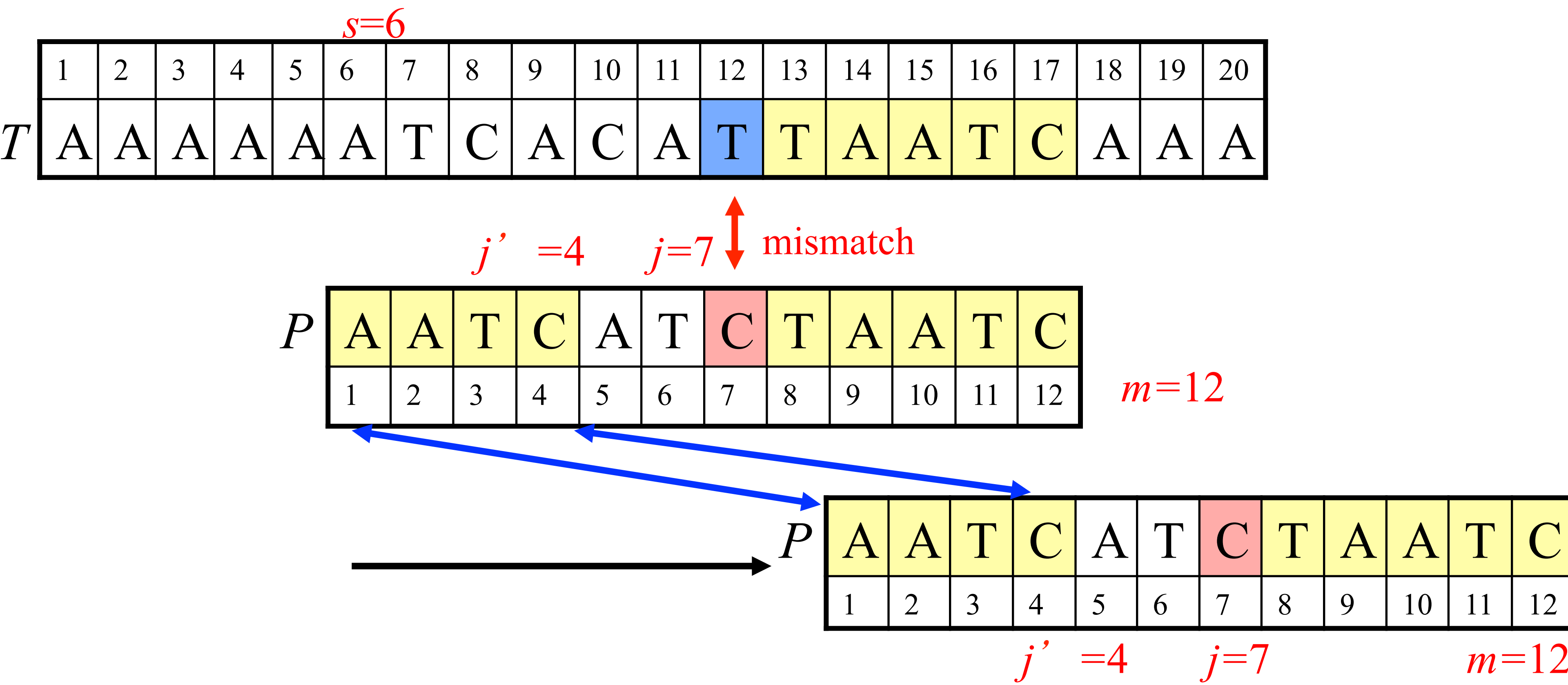
Note that the above rule also uses Rule 1.

It should also be noted that the unique substring is the shorter and the more right-sided the better.

A short *u* guarantees a short (or even empt) *s* which is desirable.



E : Suppose that P_1 is aligned to T_6 now. We compare pair-wise between P and T from right to left. Since $T_{12} \neq P_7$ and there is no substring $P_{8,12}$ in left of P_8 to e actl match $T_{13,17}$. We find a longest suffi "AATC" of substring $T_{13,17}$, the longest suffi is also prefi of P. We shift the window such that the last character of prefi substring to match the last character of the suffi substring. Therefore, we can shift at least 12-4=8 positions.



Let ***Bc*(*a*)** be the rightmost position of *a* in *P*. The function will be used for appl ing ***bad character rule***.

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A

Σ	A	C	G	T
B	12	11	0	10

We can move our pattern right $\boldsymbol{j\text{-}B(T_{s+j-}\ )}$ position b above ***Bc*** function.

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	A	G	C	T	A	G	C	C	T	G	C	A	C	G	T	A	C	A

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A

Move
10-$B(G)$ = 10 positions

Let ***Gs(j)*** be b ***good suffix rule*** when a mismatch occurs for comparing P_j with some character in T.

$\boldsymbol{gs_1(j)}$ be the largest $\boldsymbol{k}$ such that $\boldsymbol{P_{j+1,m}}$ $\boldsymbol{P_{1,k}}$

$\boldsymbol{P_{k-m+j}} \neq \boldsymbol{P_j}$, where $m-j+1 \leq k<m$; 0 if there is no such k.

(gs_1 is for Good Suffi Rule 1)

$\boldsymbol{gs_2(j)}$ be the largest $\boldsymbol{k}$ such that $\boldsymbol{P_{1,k}}$ $\boldsymbol{P_{j+1,m}}$,

where $1 \leq k \leq m-j$; 0 if there is no such k.

(gs_2 is for Good Suffi Rule 2.)

$Gs(j)$ m gs_1, gs_2 , if $j = m$, $Gs(j)$=1.

j	1	2	3	4	5	6	7	8	9	10	11	12
P												
gs_1	0	0	0	0	0	0	9	0	0	6	1	0
gs_2	4	4	4	4	4	4	4	4	1	1	1	0
Gs	8	8	8	8	8	8	3	8	11	6	11	1

$gs_1(7)=9$

$\because P_{8,12}$ is a suffi of $P_{1,9}$

and $P_4 \neq P_7$

$gs_2(7)=4$

$\because P_{1,4}$ is a suffi of $P_{8,12}$

How do we obtain $\boldsymbol{gs_1}$ and $\boldsymbol{gs_2}$?

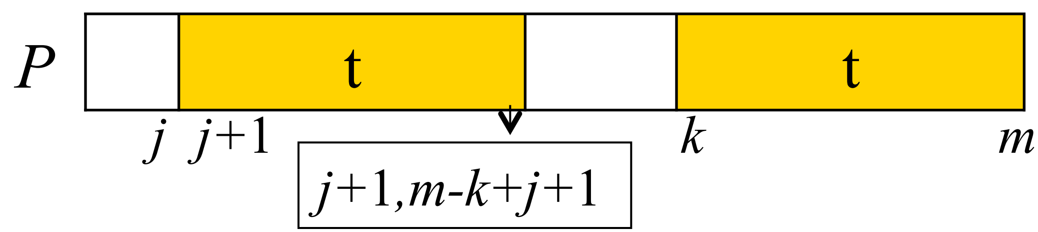
In the following, we shall show that b constructing the , we can kill two birds with one arrow.

f'

For $1 \leqq j \leqq m-1$, let the suffi function $f'(j)$ for $\boldsymbol{P_j}$ be the $\boldsymbol{k}$ such that $\boldsymbol{P_{k,m}}$ $\boldsymbol{P_{j+1,m-k+j+1}}$; ($\boldsymbol{j}$+ $\leqq k \leqq m$)

If there is no such k, we set f' $= m+1$.

If $j=m$, we set f' $(m)=m+2$.



E :

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14

- (4)=8, it means that $P_{f'(4),m} = P_{8,12} = P_{5,9} = P_{4+1,4+1+m-f'(4)}$

Since there is no k for $13=j+2 \leqq k \leqq 12$, we set f' $(11)=13$.

Suppose that the Suffi   is obtained. How can we use it to obtain $\boldsymbol{gs_1}$ and $\boldsymbol{gs_2}$?

$\boldsymbol{gs_1}$ can be obtained b   scanning the Suffi function from right to left.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
T	G	A	T	C	G	A	T	C	A	A	T	C	A	T	C	A	C	A	T	G	A	T	C	A

P	A	T	C	A	C	A	T	C	A	T	C	A
	1	2	3	4	5	6	7	8	9	10	11	12

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14

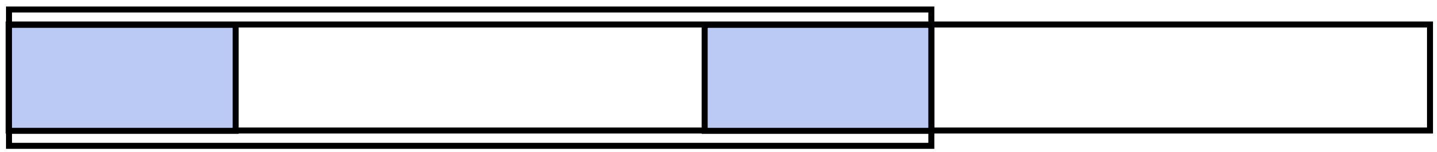
As for Good Suffi Rule 2, it is relativel easier.

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14

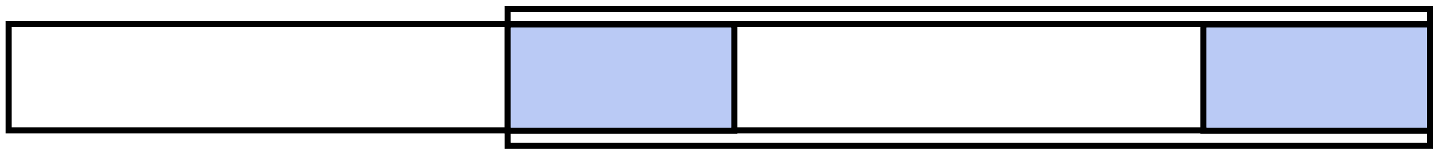
Question: How can we construct the Suffi function?

To e plain this, let us go back to the prefi function used in the KMP Algorithm.

The following figure illustrates the prefi function in the KMP Algorithm.



The following figure illustrates the suffi function of the BM Algorithm.



We now can see that actuall   the suffi function is the same as the prefi . The onl difference is now we consider a suffi . Thus, the recursive formula for the prefi   function in KMP Algorithm can be slightl   modified for the suffi   function in BM Algorithm.

The formula of suffi function f' as follows :

L $f'^{x}(y) = f'(f'^{x-1}(y)) \quad x > 1 \qquad f'^{1}(y) = f'(y)$

$$f'(j) = \begin{cases} m+2, & j = m \\ f'^{k}(j+1)-1, & 1 \le j \le m-1 \\ & k \ge 1 \qquad P_{j+1} = P_{f'^{k}(j+1)-1}; \\ m+1, & \end{cases}$$

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'												14

j ,
f' +

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'											13	14

k
P + ***P*** $_{f'}$ $_{k(j+\ )-}$,
f' ***m*** + +

$k = 1 \rightarrow$

$P_{12} \neq P_{13}$

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'								12	13	13	13	14

$\because P_{j+1} = P_{f'(j+1)-1} \Rightarrow P_9 = P_{12}$,

f' f' $(j+1)$ - -

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'							11	12	13	13	13	14

$\because P_{j+1} = P_{f'(j+1)-1} \Rightarrow P_8 = P_{11}$,

f' f' $(j+1)$ - -

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'				8	9	10	11	12	13	13	13	14

$\because P_{j+1} = P_{f' \quad (j+1)-1} \Rightarrow P_5 = P_8$,
f' f' $(j+1)$ - -

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'			12	8	9	10	11	12	13	13	13	14

$\because P_{j+1} = P_{f' \quad (j+1)-1} \Rightarrow P_4 = P_{f' \quad (4)-1} = P_{12}$,
f' f' $(j+1)$ - -

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'		11	12	8	9	10	11	12	13	13	13	14

∵ $P_{j+1} = P_{f'(j+1)-1}$ => $P_3 = P_{f'(3)-1} = P_{11}$,
f ***f*** **(*j*+*1*) -** **-**

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14

∵ $P_{j+1} = P_{f'(j+1)-1}$ => $P_2 = P_{f'(2)-1} = P_{10}$,
f ***f*** **(*j*+*1*) -** **-**

Let $G'(j)$, $1 \leqq j \leqq m$,to be the largest number of shifts b good suffi rules.

First, we set $G'(j)$ to eros as their initiali ations.

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14
G'	0	0	0	0	0	0	0	0	0	0	0	0

We scan from right to left and $gs_1(j)$ is determined during the scanning, then $gs_1(j) >= gs_2(j)$

If $P_j = P_4 \neq P_7 = P_{f'(j)-1}$, we know $\boldsymbol{gs}$ $(\boldsymbol{f'}$ $(\boldsymbol{j})-$ $)$ $\boldsymbol{m+j-f'(j)+}$ $=9.$
$\boldsymbol{t}$

- When $j=12$, $t=13$. $t > m$.
- When $j=11$, $t=12$. Since $P_{11}=$ 'C' $\neq$ 'A' $= P_{12}$,
  G' $(t) = m$ ma $gs_1(t), gs_2(t)$ $= m$ $\underline{gs_1(t)}$
  $= f'$ $(j) - 1 - j$
  $\Rightarrow G'$ $(12) = 13-1-11 =$ 1.

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14
G'	0	0	0	0	0	0	0	0	0	0	0	1

t f (j)- ≦ m $P_j \neq P_t$, G' (t)=f' (j) – 1 – j.
f' (k)(x)=f' (k-1)(f' (x) 1), k ≥

- When j=10, t=12. Since P_{10}= ‘T’ ≠ ‘A’ =P_{12}, G' (12) ≠0.
- When j=9, t=12. P_9= ‘A’ =P_{12}.
- When j=8, t=11. P_8= ‘C’ =P_{11}.
- When j=7, t=10. P_7= ‘T’ =P_{10}
- When j=6, t=9. P_6 = ‘A’ =P_9
- When j=5, t=8. P_5 = ‘C’ =P_8
- When j=4, t=7. Since P_4= ‘A ≠ P_7= T , $G'(7) = 8 – 1 – 4= 3$

Besides, $t = f'^{(2)}(4)$ 1=f (f (4) 1) – 1=10. Since P_4 = A ≠ P_{10} = ‘T’ , G' (10) =f' (7) 1 – j= 11 1 4 = 6.

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14
G'	0	0	0	0	0	0	3	0	0	6	0	1

$t \quad f' \quad (j)- \quad \leqq m \qquad P_j \neq P_t, G'(t)=f'(j) - 1 - j.$
$f'^{(k)}(x)=f'^{(k-1)}(f'(x) \quad 1), k \geq$

- When j=3, t=11. P_3= ‘C’ =P_{11}.
- When j=2, t=10. P_2= ‘T’ =P_{10}
- When j=1, t=9. P_1= ‘A’ =P_9.

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14
G'	0	0	0	0	0	0	3	0	6	0	0	1

B the above discussion, we can obtain the values using the Good Suffi Rule 1 b scanning the pattern from right to left.

Continuousl , we will tr to obtain the values using ***Good Suffix Rule 2*** and those values are still eros now and scan from left to right.

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14
G'	0	0	0	0	0	0	3	0	0	6	0	1

Let k' be the k in 1, ,m such that $P_{f'^{(k)}( )-}$ P and $f'^{(k)}(1)-1<=m$.

If $G'(j)$ is not determined in the first scan and $1<=j<=f'^{(k')}(1)-2$, thus, in the second scan, we set $G'(j)=m$ - ma $gs_1(j)$, $gs_2(j)$ $= m - gs_2(j)=f'^{(k')}(1) - 2$. If no such k e ists, set each undetermined value of G to m in the second scan.

k ***k***

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14
G'	8	8	8	8	8	8	3	8	0	6	0	1

Let z be $f'^{(k')}(1)$-2. Let k'' be the **k** such that $f'^{(k)}(z)-1 <= m$.

Then we set $G'(j) = m - gs_2(j) = m - (m - f'^{(i)}(z) - 1) = f'^{(i)}(z) - 1$, where $1 <= i <= k''$ and $f'^{(i-1)}(\ ) < j <= f'^{(i)}(z)-1$ and $f'^{(0)}(z) = z$.

For e ample, z=8 :

- $k=1, f'^{(1)}(8)-1=11 \leqq m=12$
- $k=2, f'^{(2)}(8)-1=12 \leqq m=12$ **=> $k''=2$**
- $i=1, f'^{(0)}(8)-1 = 7 < \boldsymbol{j} \leqq f'^{(1)}(8)-1=11$.
- $i=2, f'^{(1)}(8)-1 =11< \boldsymbol{j} \leqq f'^{(2)}(8)-1=12$.
- We set $G(9)$ and $G(11)=f'^{(1)}(8)$ 1= 12-1 = 11.

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14
G'	8	8	8	8	8	8	3	8	11	6	11	1

We essentiall have to decide the ma imum number of steps. We can move the window right when a mismatch occurs. This is decided b the following function:

ma G' $(j), j\text{-}B(T_{s+j-1})$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
T	G	A	T	C	G	A	T	C	A	C	A	T	A	T	C	A	C	A	T	C	A	T	C	A

mismatch

P	A	T	C	A	C	A	T	C	A	T	C	A
	1	2	3	4	5	6	7	8	9	10	11	12

P	A											

We compare *T* and *P* from right to left. Since T_{12}= "T" $\neq P_{12}$= "A" , the largest movement = ma G' $(j), j\text{-}B(T_{s+j-1})$ = ma G' $(12), 12\text{-}B(T_{12})$ = ma 1,12-10 = 2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
T	G	A	T	C	G	A	T	C	A	C	A	T	A	T	C	A	C	A	T	C	A	T	C	A

mismatch

P	A	T	C	A	C	A	T	C	A	T	C	A
	1	2	3	4	5	6	7	8	9	10	11	12

→

P	A	T	C	A	C	A	T	C	A	T	C	A
	1	2	3	4	5	6	7	8	9	10	11	12

j	1	2	3	4	5	6	7	8	9	10	11	12
P	A	T	C	A	C	A	T	C	A	T	C	A
f'	10	11	12	8	9	10	11	12	13	13	13	14
G'	8	8	8	8	8	8	3	8	11	6	11	1

Σ	A	C	G	T
B	12	11	0	10

After moving, we compare *T* and *P* from right to left. Since T_{14}= "T" $\neq P_{12}$= "A", the largest movement = ma *G'* (*j*), *j*-*B*(*Ts*+*j*-1) = ma *G'* (*12*), *12*-*B*(T_{14})
= ma 1,12-10 = 2.

The preprocessing phase in O(m+Σ) time and space comple it   and searching phase in ***O***(mn) time comple it .

The worst case time comple it   for the ***Boyer-Moore*** method would be $O((n\text{-}m+1)m)$.

It was proved that this algorithm has $O(m)$ comparisons when P is not in T. However, this algorithm has $O(mn)$ comparisons when P is in T.

, AHO, A.V.,
,Volume A ,Chapter 5 Elsevier , Amsterdam , 1990, pp. 255-300.

, Jun-ichi, A. , IEEE
Computer Societ Press , 1994.

, BAASE, S. and
VAN GELDER, A. , - , Chapter 11 , 1999.

, BAEZA-YATES, R. , NAVARRO, G. and RIBEIRO-
NETO, B. , , Chapter 8 , 1999 , pp. 191-228.

' , BEAUQUIER, D., BERSTEL, J. and CHR TIENNE,
P., Masson Paris , Chapter 10 , 1992 , pp. 337-377.

, BOYER R.S. and MOORE J.S. ,
, Vol 20 , 1977 , pp. 762-772 .

-
, COLE, R., , Vol 23 , 1994 , pp.
1075-1091.

, CORMEN, T.H. , LEISERSON, C.E. and RIVEST,
R.L. , , Chapter 34 , 1990 , pp. 853-885.

- , CROCHEMORE, M. ,
, Chapter 1 , 1997 , pp 1-53

, CROCHEMORE, M. and HANCART, C. ,
, Chapter 11, 1999 , pp.
11-1-11-28.

, CROCHEMORE, M. and LECROQ, T. , , Chapter 8 , 1996 , pp. 162-202.

, CROCHEMORE, M. and RYTTER, W., , 1994.

, GONNET, G.H. and BAEZA-YATES, R.A. , - , Chapter 7 , 1991 , pp. 251-288,.

, GOODRICH, M.T. and TAMASSIA, R. , & , Chapter 11 , 1998 , pp. 441-467.

, , GUSFIELD, D. , , 1997.

' ' , HANCART, C., Universit Paris 7, France , 1993.

, KNUTH, D.E. , MORRIS, J.H. and PRATT, V.R. , , 1977 , pp.323-350.

LECROQ, T., 1992, *Recherches de mot*,. Thesis, Universit of Orl ans, France.

, LECROQ, T. , - & , Vol 25 , 1995 , pp. 727-765.

, SEDGEWICK, R. , - , Chapter 19 , 1988 , pp. 277-292.

, SEDGEWICK, R. , - , Chapter 19 , 1988.

, STEPHEN, G.A. , World Scientific , 1994.

, WATSON, B.W. , , 1995.

& , WIRTH, N. , Prentice-Hall , Chapter 1 , 1986 , pp. 17-72.

Suffi trees and suffi arra s

String/Pattern Matching

- You are given a source string S.
- Answer queries of the form: is the string p_i a substring of S?
- Knuth-Morris-Pratt (KMP) string matching.
  - O(S + p_i) time per quer .
  - O(n S + S_i p_i) time for n queries.
- Suffi tree solution.
  - O(S + S_i p_i) time for n queries.

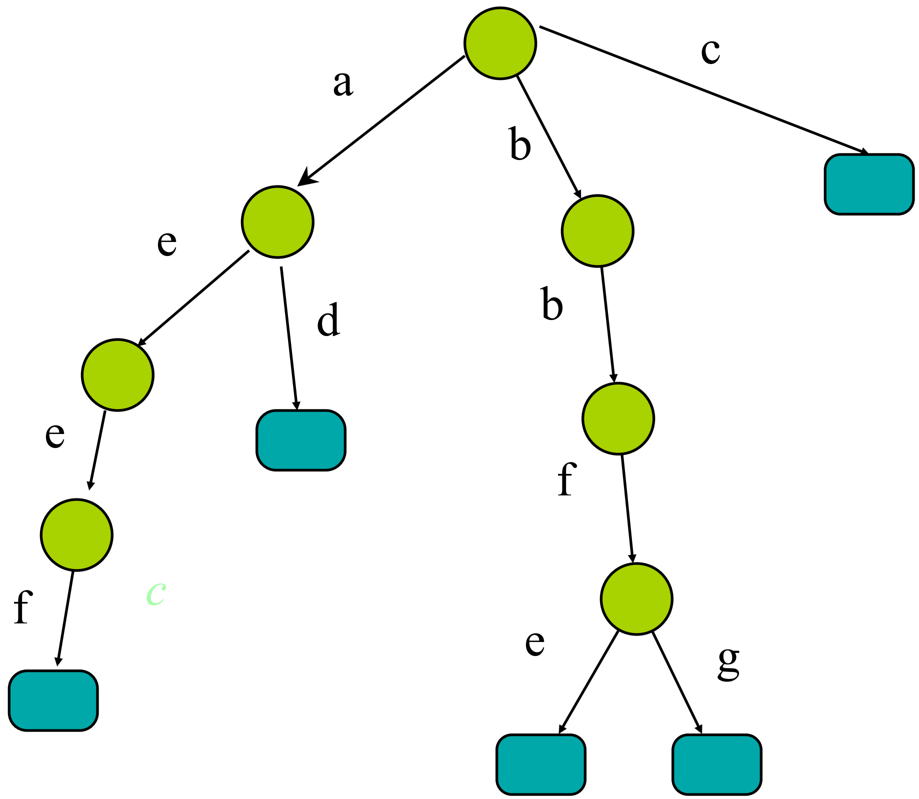
String/Pattern Matching

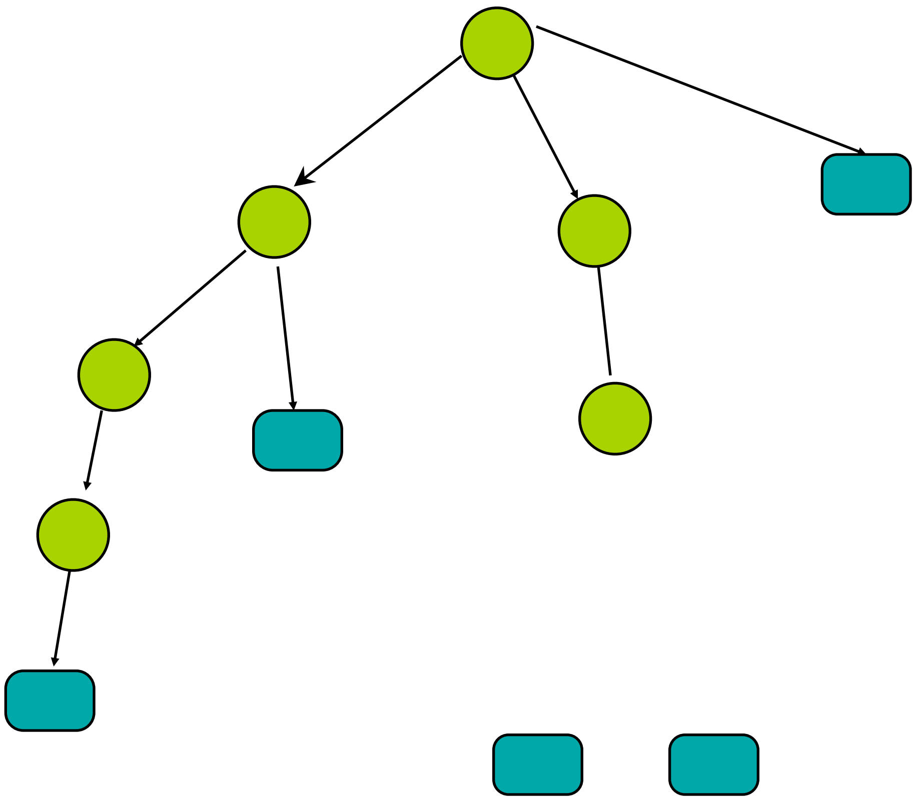
- KMP/BM preprocesses the quer   string p_i, whereas the suffi   tree method preprocesses the source string S.

Trie

A tree representing a set of strings.

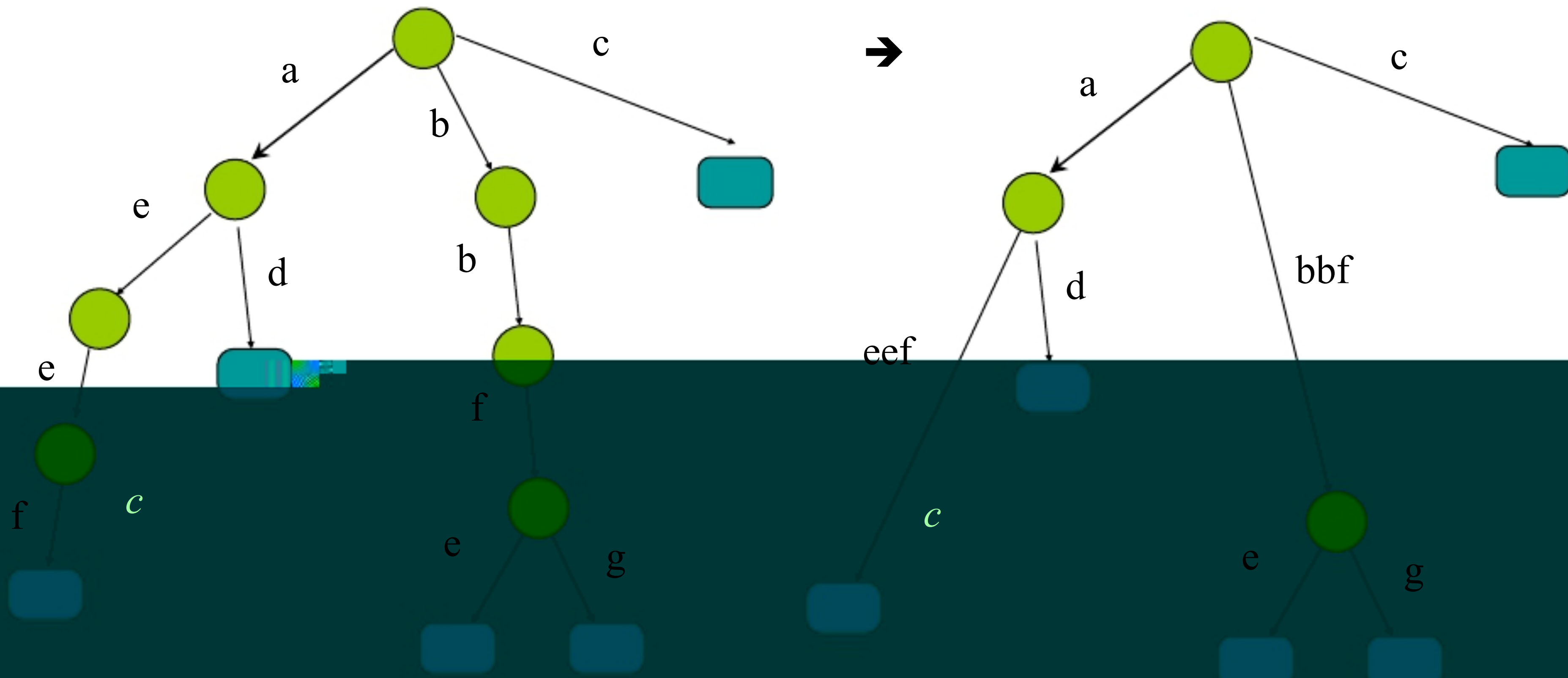
aeef
ad
bbfe
bbfg
c





Compressed Trie

Compress unar   nodes, label edges b   strings



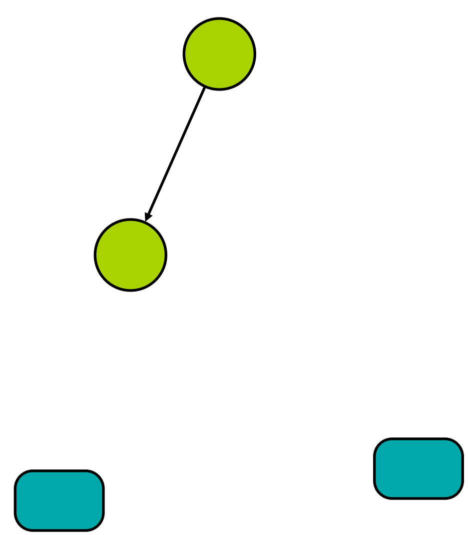
Suffi tree

Given a string s a suffi tree of s is a compressed trie of all suffi es of s

To make these suffi es prefi -free we

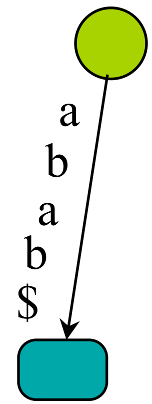
The suffi tree Tree(T) of T

- data structure suffi tree, Tree(T), is compacted trie that represents all the suffi es of string T
- linear si e: Tree(T) = O(T)
- can be constructed in linear time O(T)
- has *myriad virtues* (A. Apostolico)
- is well-known: Google hits

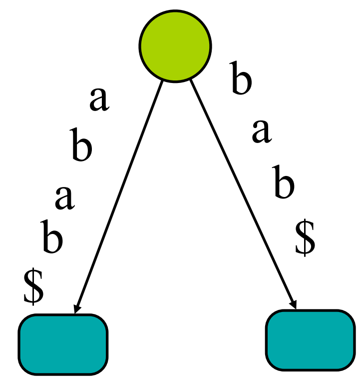


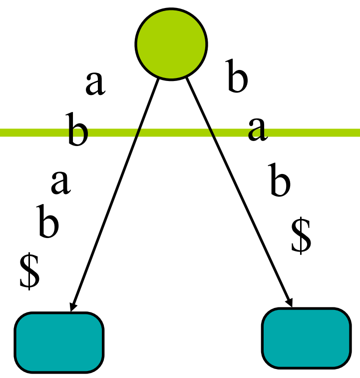
Trivial algorithm to build a Suffi tree

Put the largest suffi in

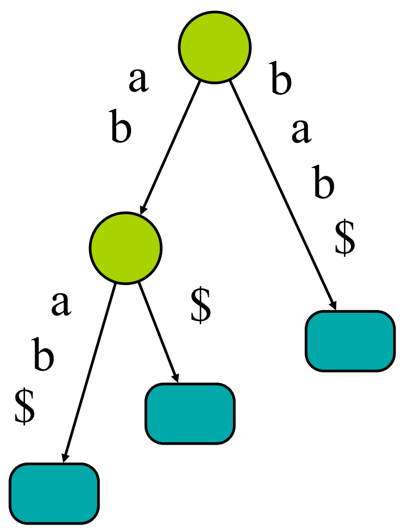


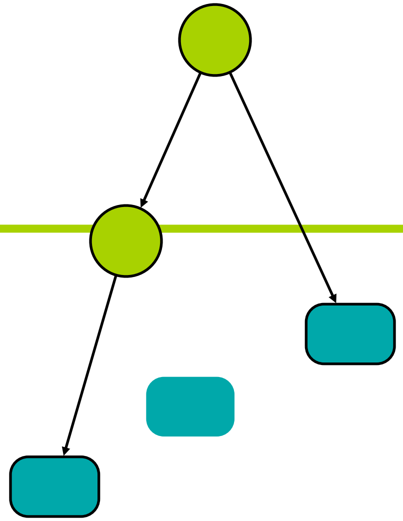
Put the suffi bab$ in

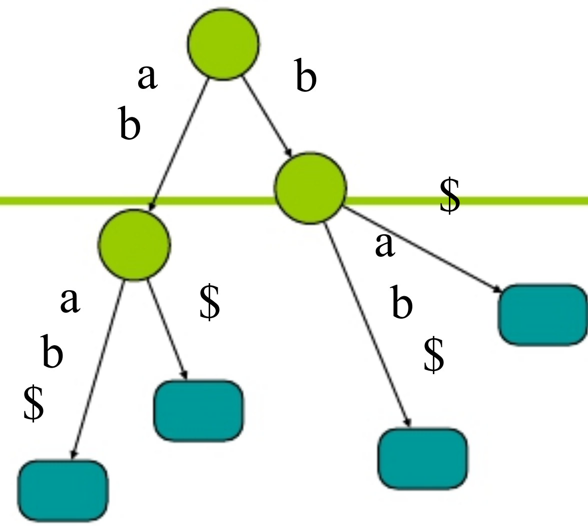




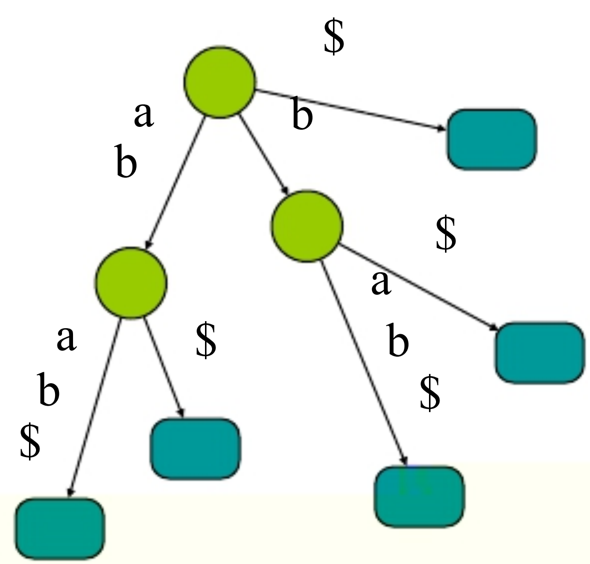
Put the suffi ab$ in

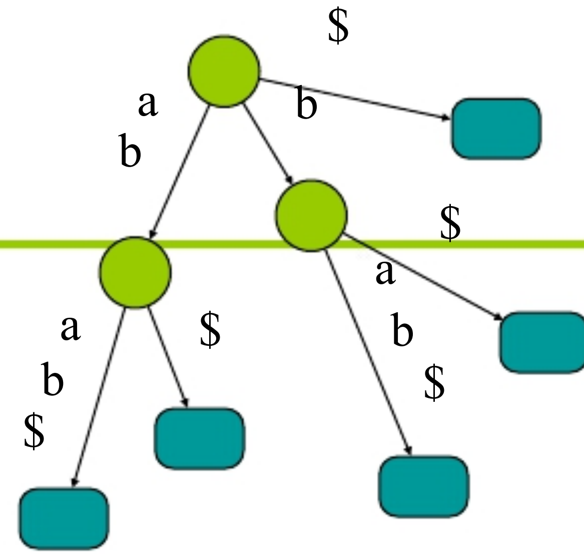




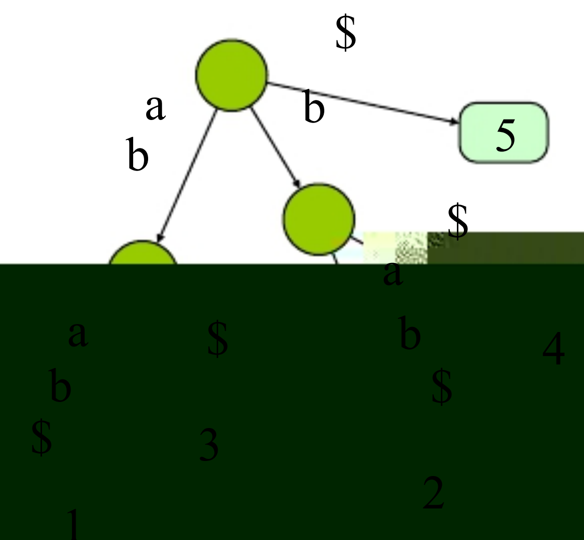


Put the suffi $ in





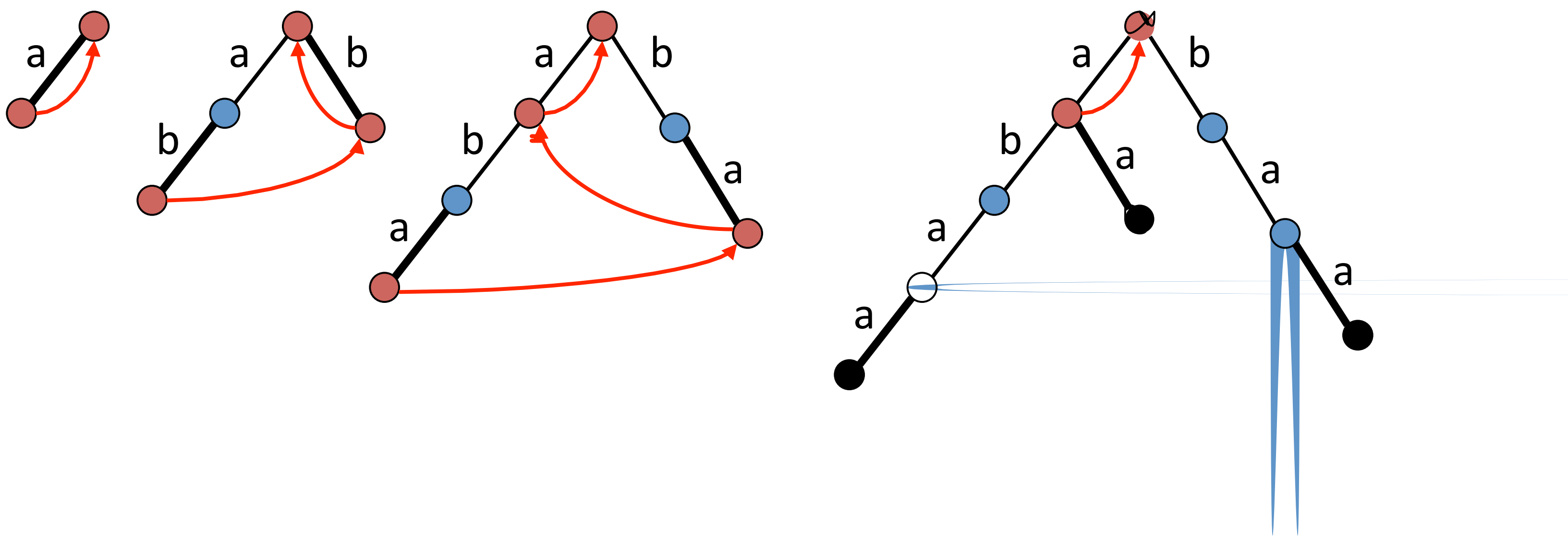
We will also label each leaf with the starting point of the corres. suffi .



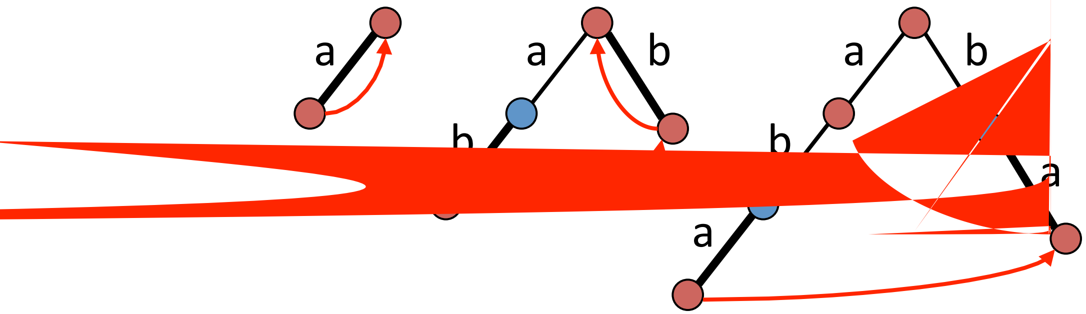
On-line construction of Trie(T)

- $T = t_1 t_2 \dots t_n\$$
- $P_i = t_1 t_2 \dots t_i$ i:th prefix of *T*
- <u>on-line idea</u>: update *Trie(P_i)* to *Trie(P_{i+1})*
- => very simple construction

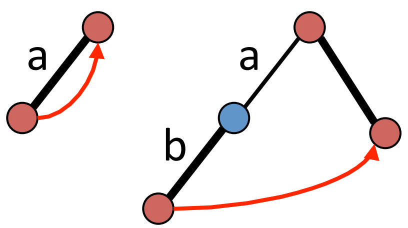
Trie(abaab)



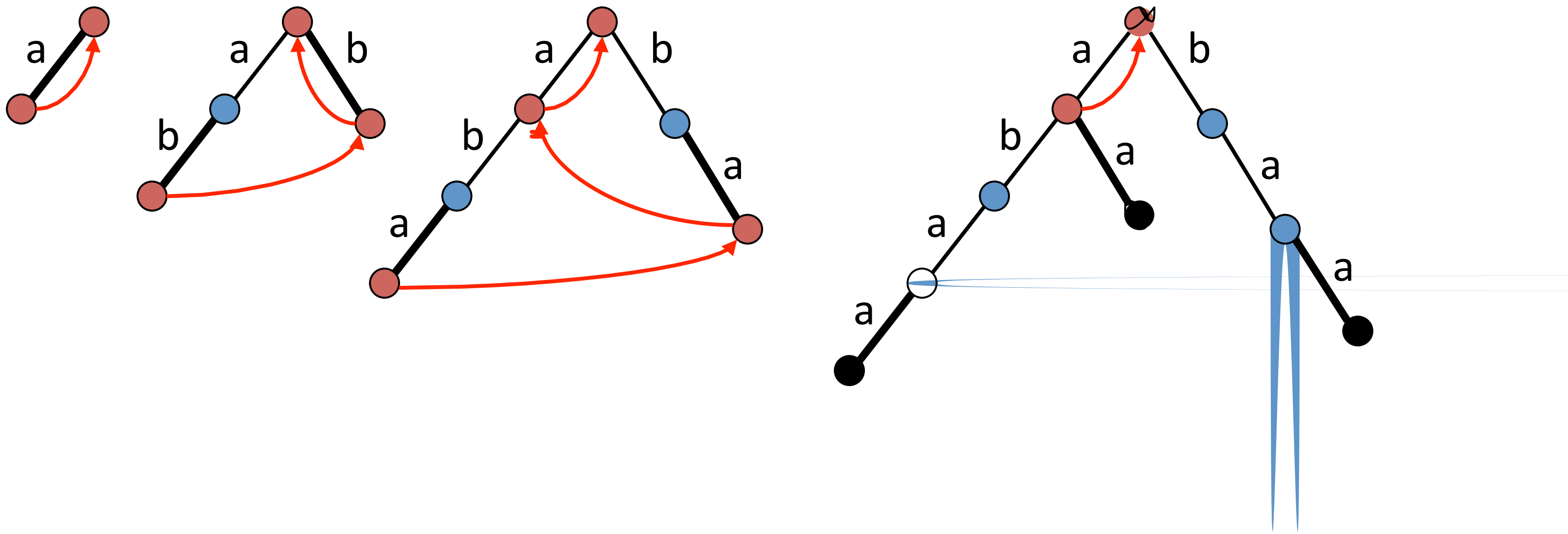
Trie(abaab)



Trie(abaab)



Trie(abaab)



What happens in *Trie*(P_i) => *Trie*(P_{i+1}) ?

- time: O(size of Trie(T))
- suffix links:

  slink(node($a\alpha$)) = node(α)

What can we do with it ?

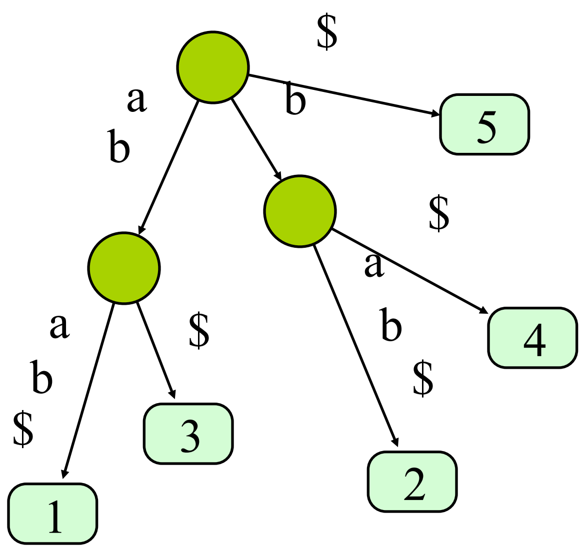
E act string matching:

Given a Text T, |T| = n, preprocess it such that when a pattern P, |P|=m, arrives you can quickly decide when it occurs in T.

W e may also want to find all occurrences of P in T

E act string matching

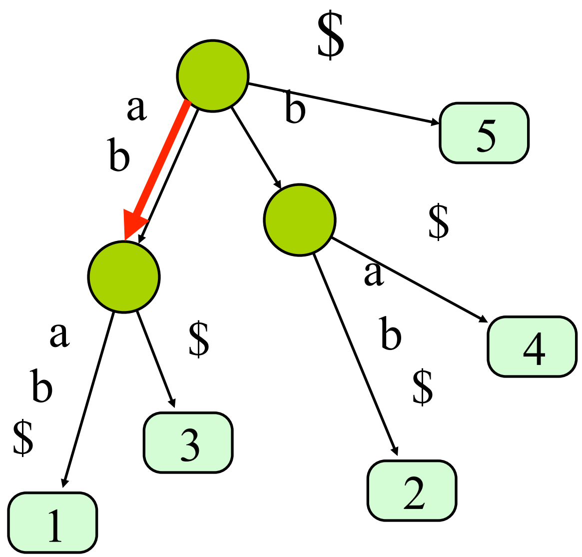
In preprocessing we just build a suffi tree in O(n) time



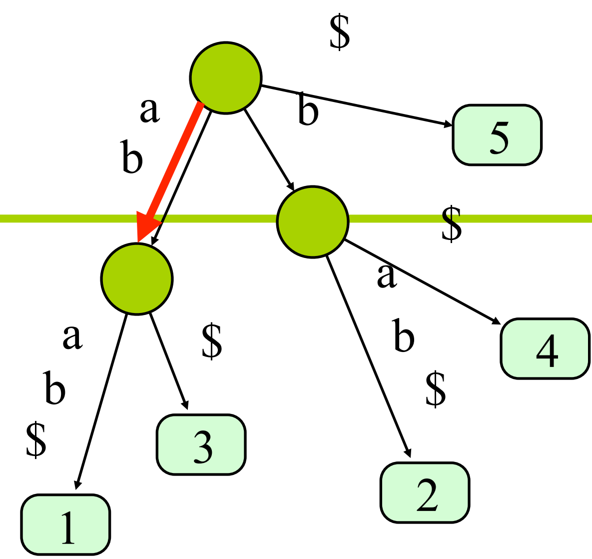
Given a pattern P = ab we traverse the tree according to the pattern.

E act string matching

In preprocessing we just build a suffi   tree in O(n) time



Given a pattern P = ab we traverse the tree according to the pattern.



If we did not get stuck traversing the pattern then the pattern occurs in the te t.

Each leaf in the subtree below the node we reach corresponds to an occurrence.

B   traversing this subtree we get all k occurrences in O(n+k) time

Generali ed suffi   tree

Given a set of strings S a generali ed suffi tree of S is a compressed trie of all suffi es of s ∈ S

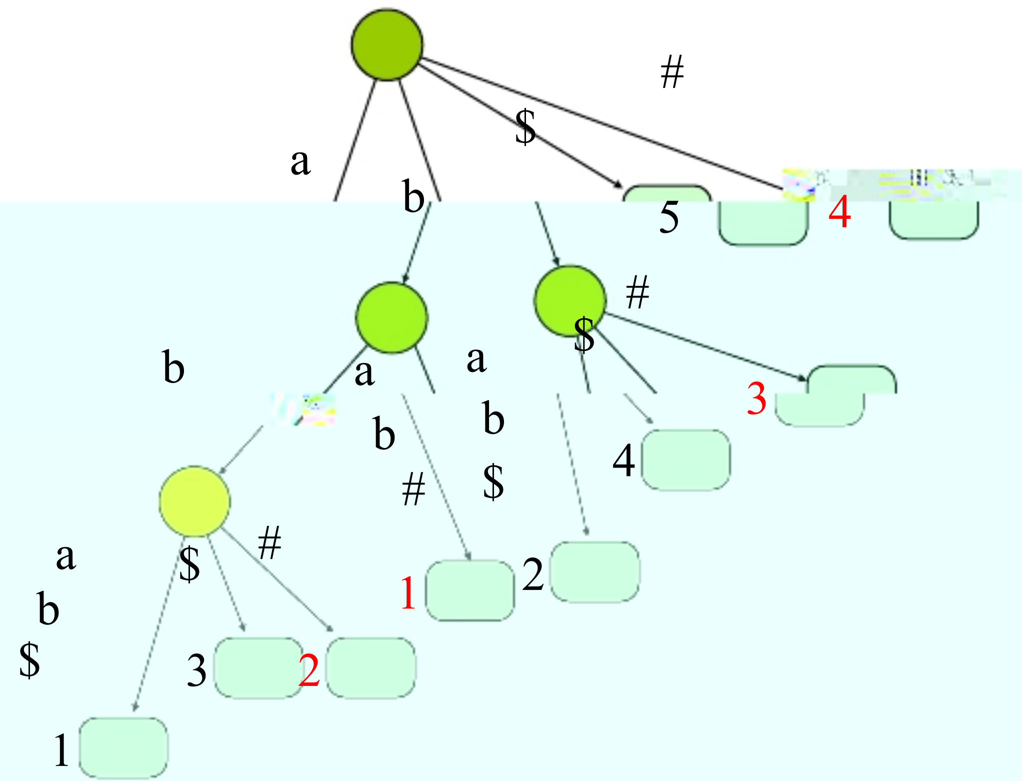
To make these suffi es prefi -free we add a special char, sa $, at the end of s

To associate each suffi   with a unique string in S add a different special char to each s

Generali ed suffi tree (E ample)

Let s_1=abab and s_2=aab here is a generali ed suffi tree for $s_1$and s_2

$	#
b$	b#
ab$	ab#
bab$	aab#
abab$	



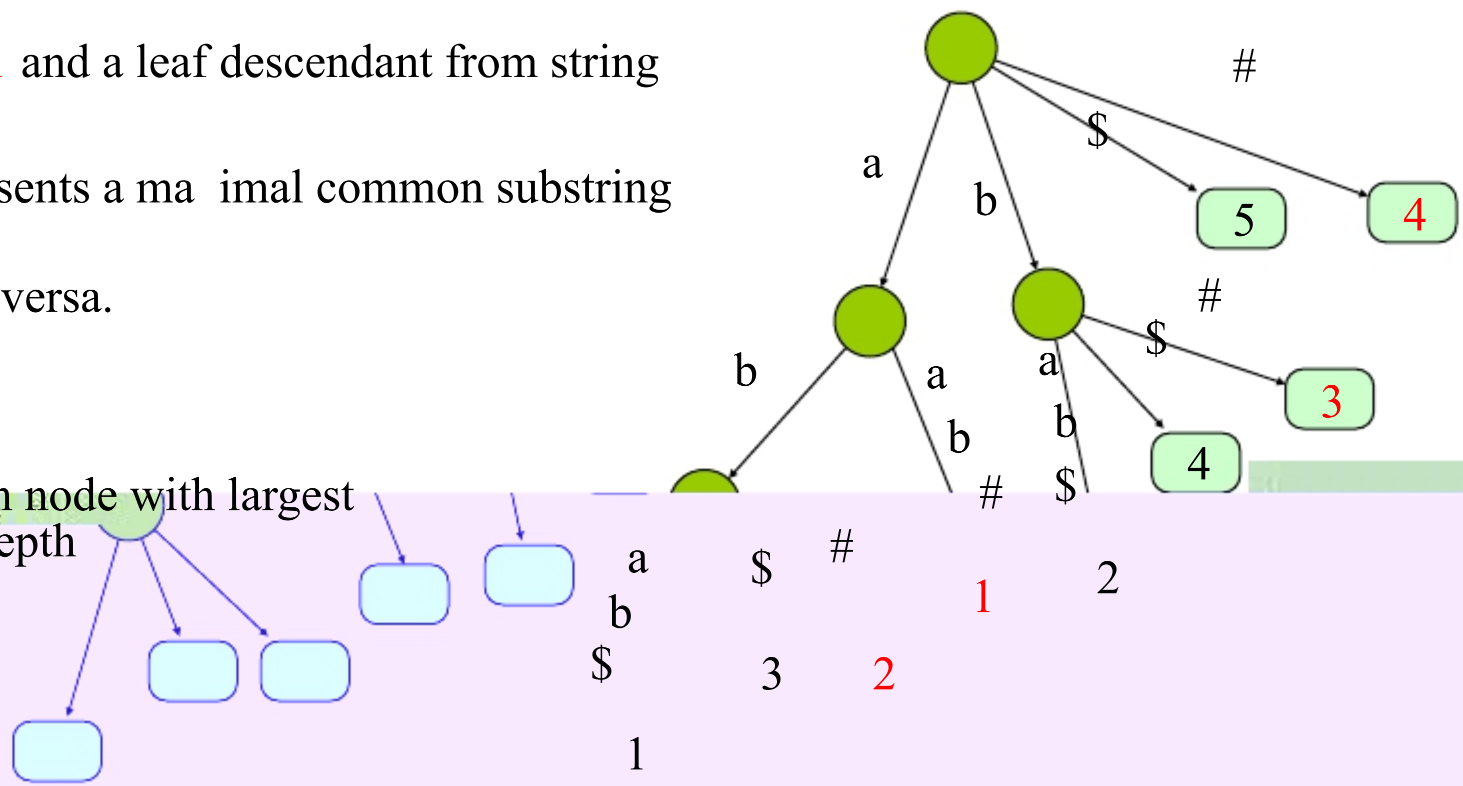
So what can we do with it ?

Matching a pattern against a database of strings

Longest common substring (of two strings)

Ever node with a leaf descendant from string $S1$ and a leaf descendant from string $S2$ represents a ma imal common substring and vice versa.

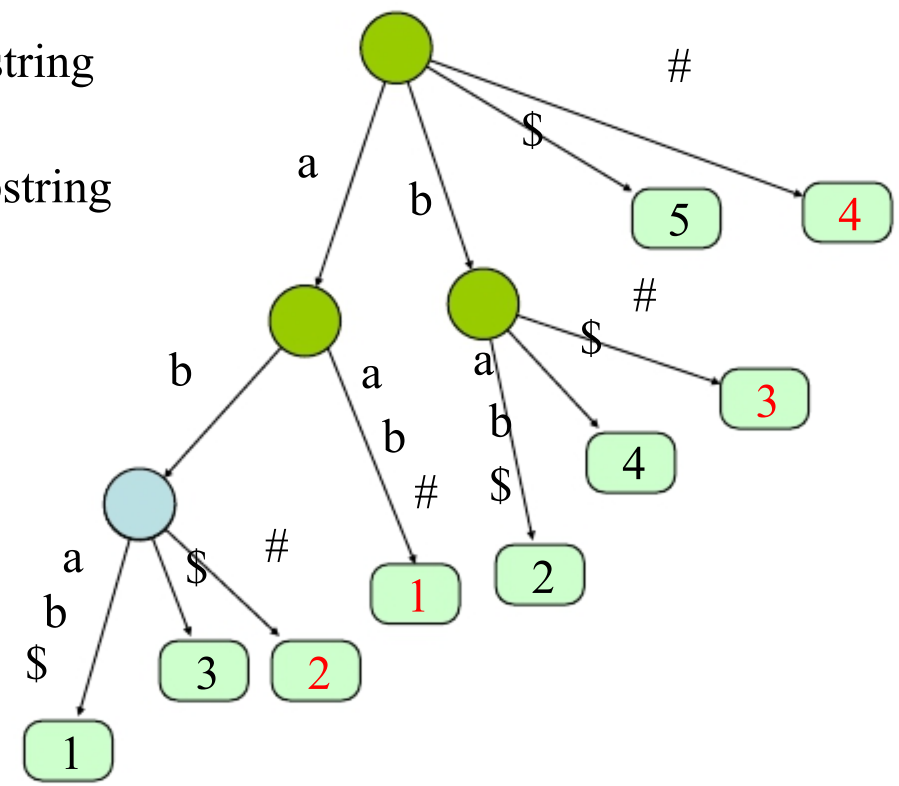
Find such node with largest string depth



Longest common substring (of two strings)

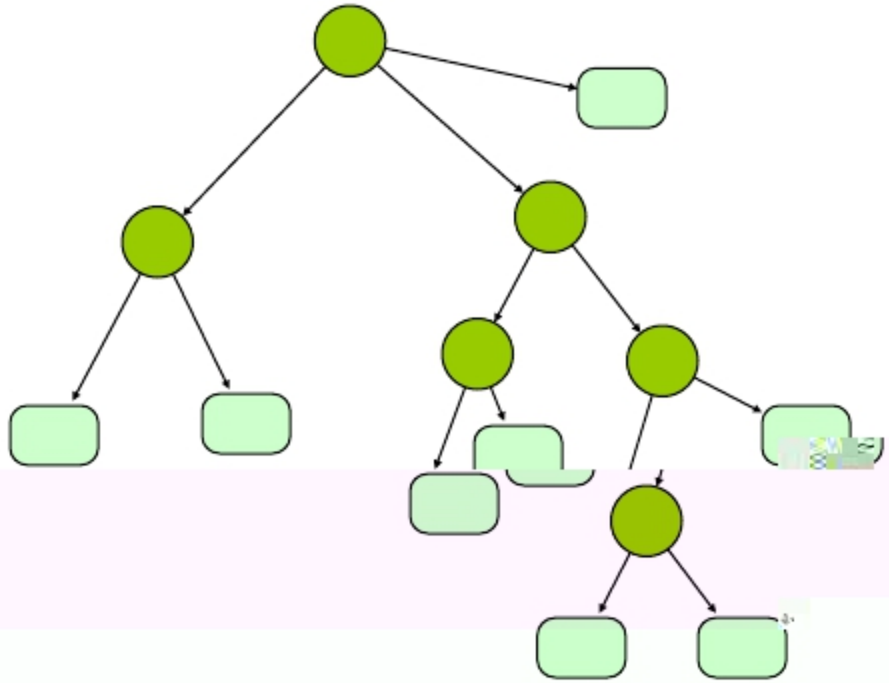
Ever node with a leaf descendant from string S1 and a leaf descendant from string S2 represents a ma imal common substring and vice versa.

Find such node with largest string depth



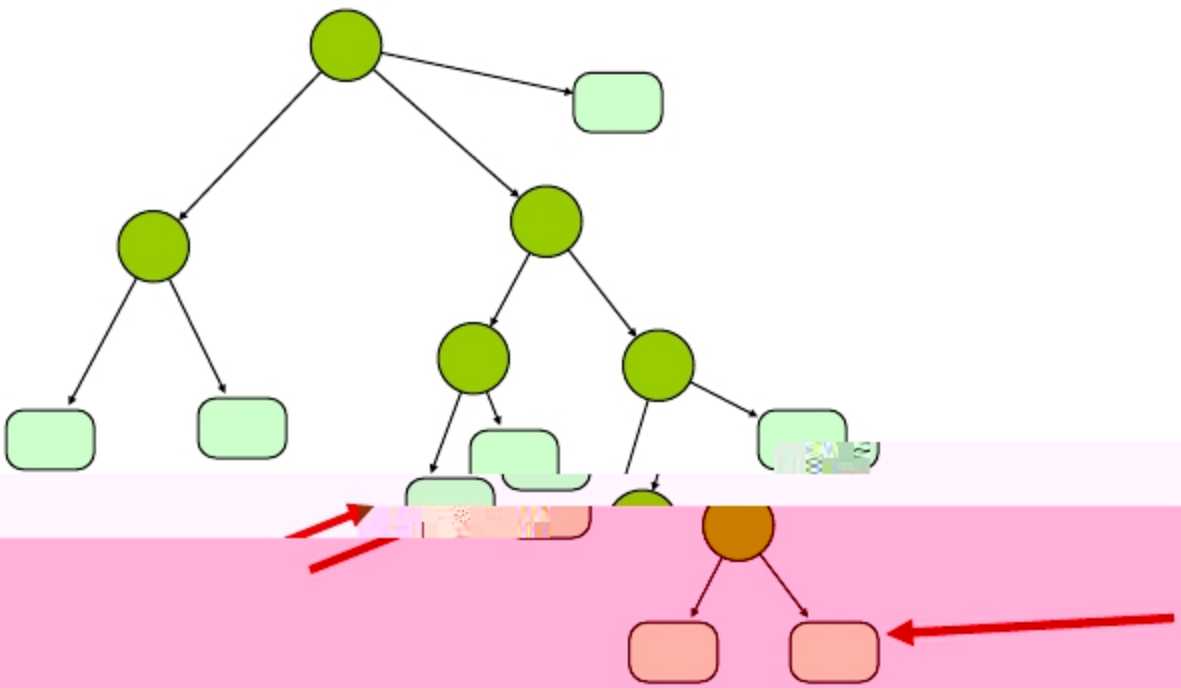
Lowest common ancestor

A lot more can be gained from the suffi tree if we preprocess it so that we can answer LCA queries on it



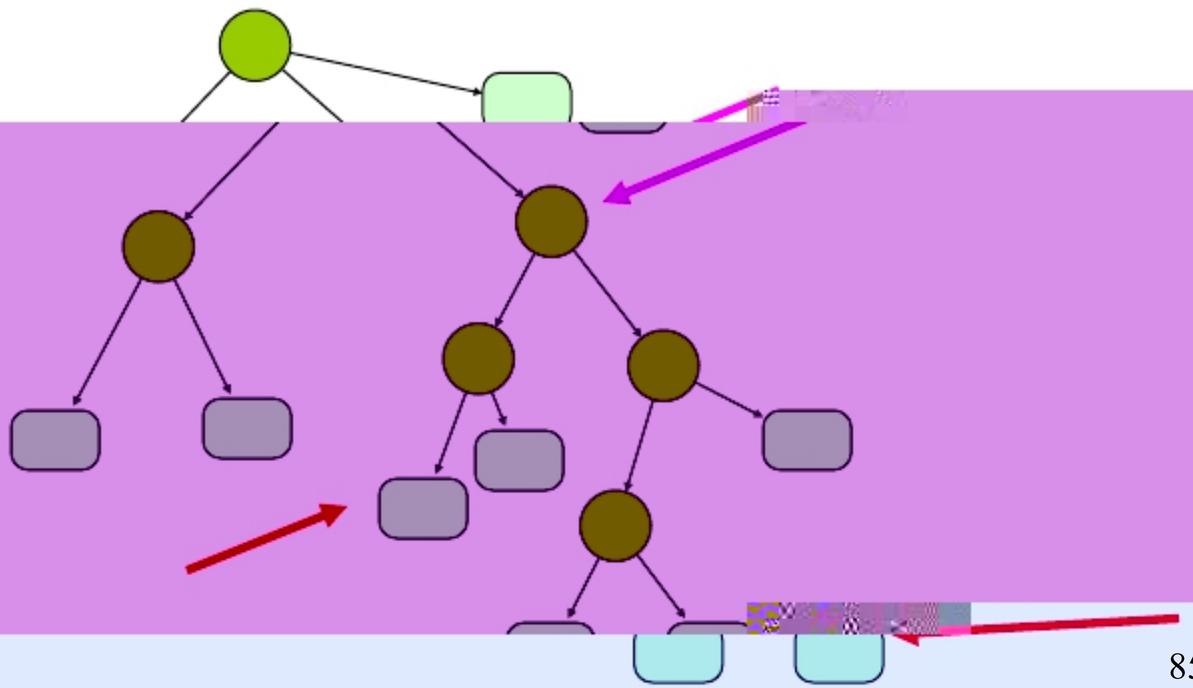
Lowest common ancestor

A lot more can be gained from the suffi   tree if we preprocess it so that we can answer LCA queries on it



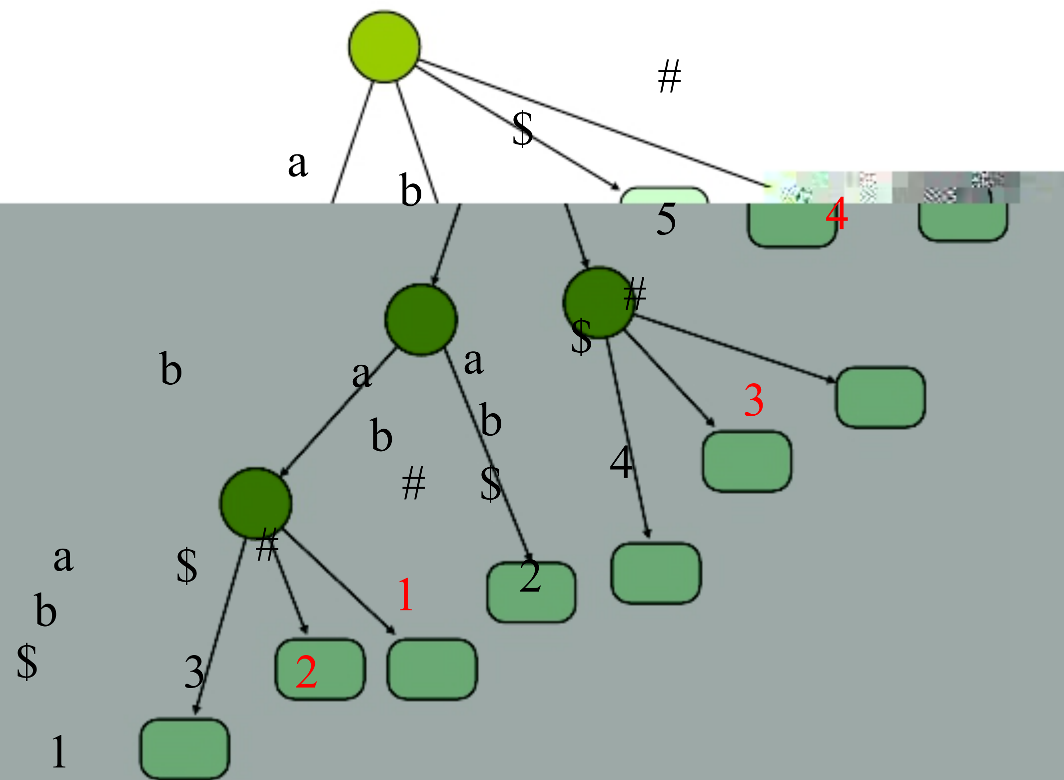
Lowest common ancestor

A lot more can be gained from the suffi   tree if we preprocess it so that we can answer LCA queries on it



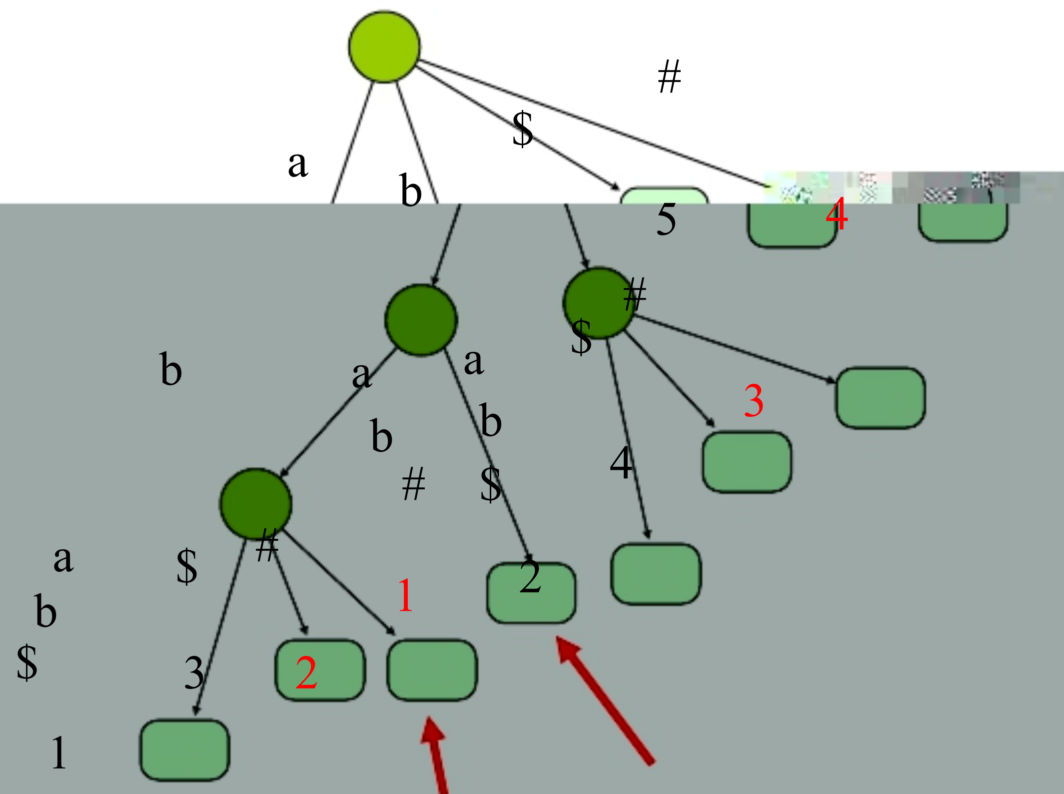
Wh ?

The LCA of two leaves represents the longest common prefi (LCP) of these 2 suffi es



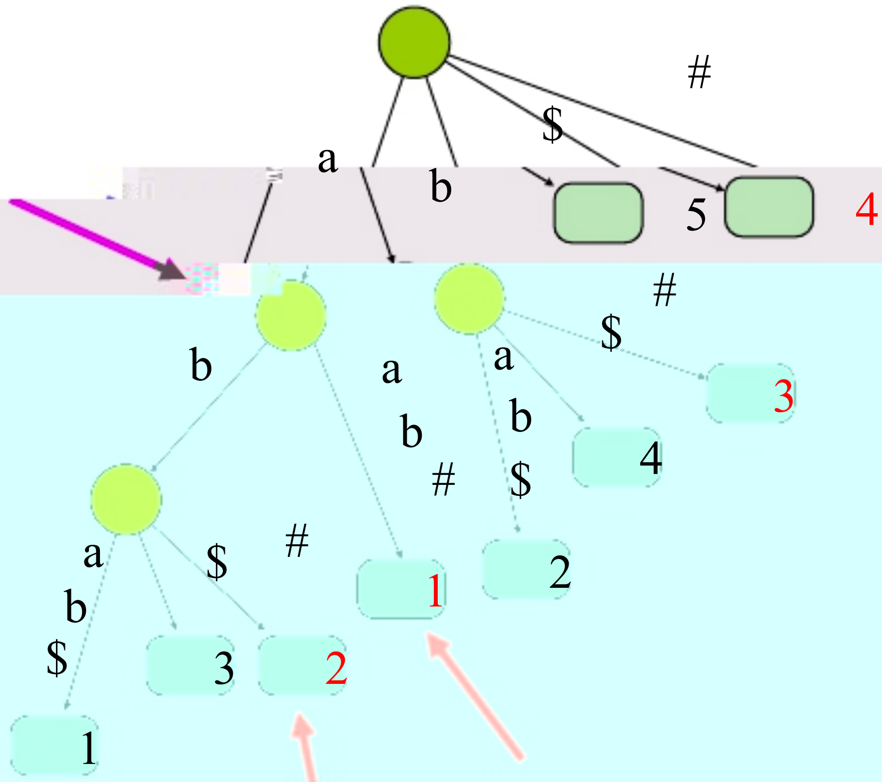
Wh ?

The LCA of two leaves represents the longest common prefi (LCP) of these 2 suffi es



Wh ?

The LCA of two leaves represents the longest common prefi (LCP) of these 2 suffi es

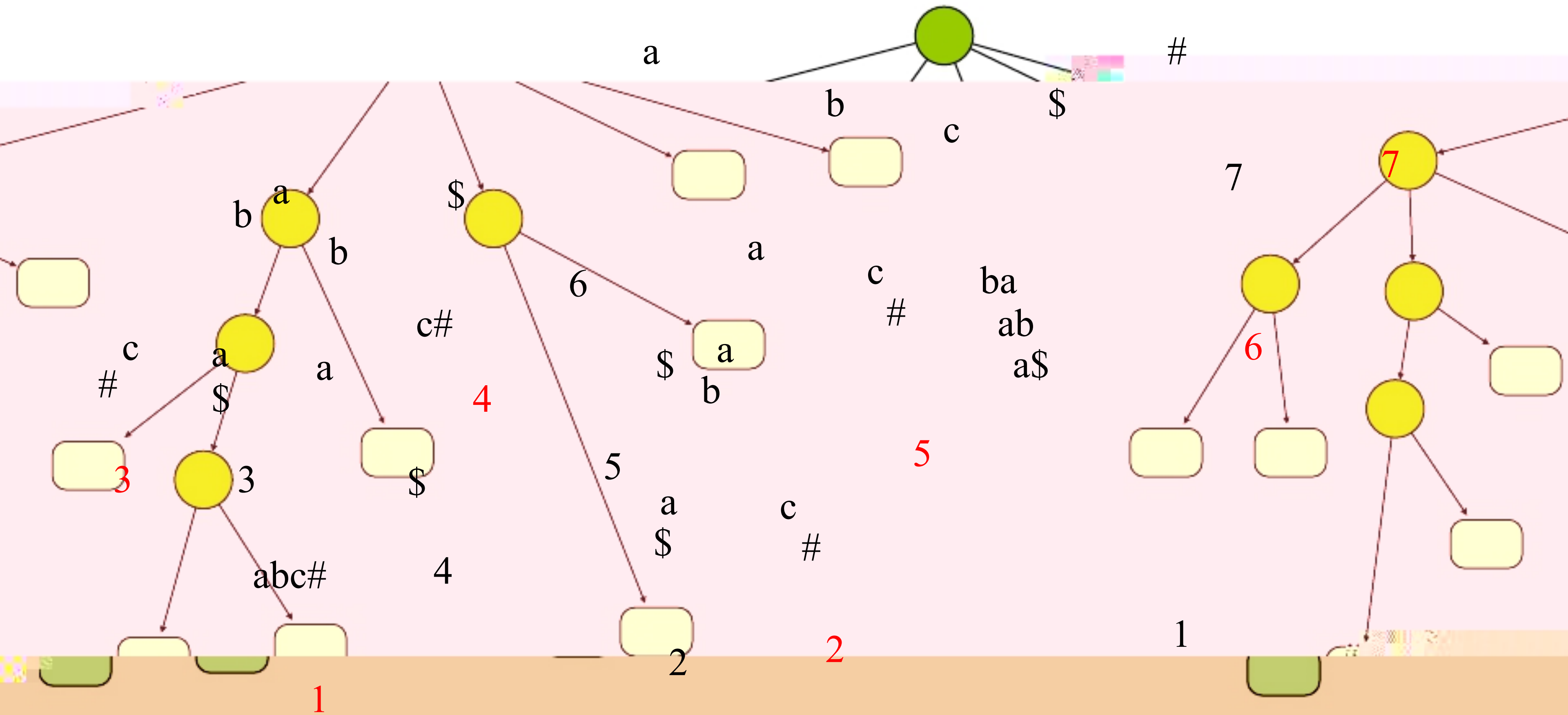


Ma imal palindromes algorithm

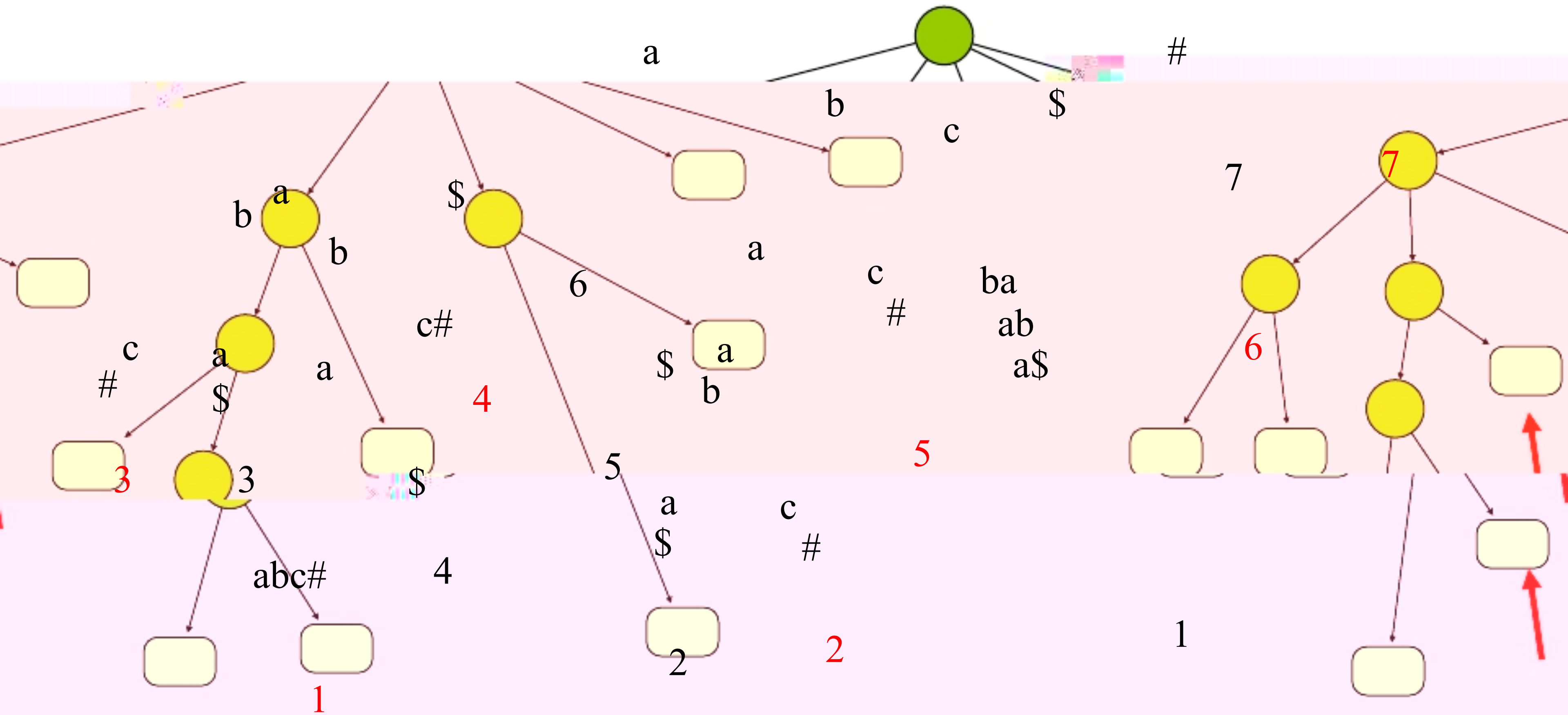
Prepare a generali ed suffi   tree for
s = cbaaba$ and s_r = abaabc#

For ever   i find the LCA of suffi   i of s and suffi   m-i+1 of s_r

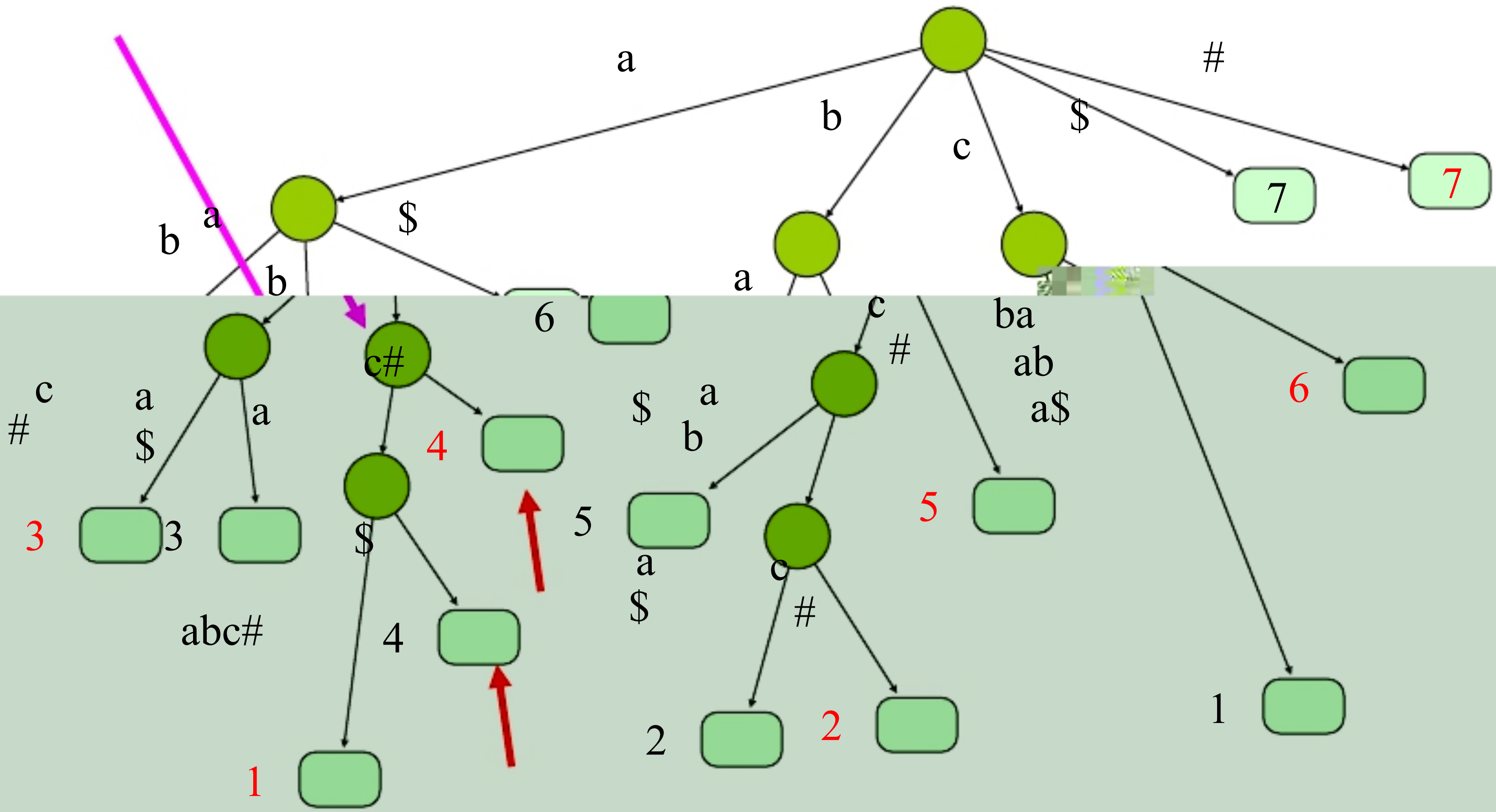
Let s = cbaaba$ then sr = abaabc#



Let s = cbaaba$ then s_r = abaabc#



Let s = cbaaba$ then s_r = abaabc#



Anal sis

O(n) time to identif all palindromes

Drawbacks

Suffi   trees consume a lot of space

It is O(n) but the constant is quite big

Notice that if we indeed want to traverse an edge in O(1) time then we need an arra   of ptrs. of si e Σ in each node

Suffi arra

We loose some of the functionalit but we save space.

Let s = abab

Sort the suffi es le icographicall :

ab, abab, b, bab

The suffi arra gives the indices of the suffi es in sorted order

3	1	4	2

How do we build it ?

Build a suffi tree

Traverse the tree in DFS, le icographicall picking edges outgoing from each node and fill the suffi arra .

O(n) time

How do we search for a pattern ?

If P occurs in T then all its occurrences are consecutive in the suffi   arra .

Do a binar   search on the suffi   arra

Takes O(mlogn) time

E ample

Let S = mississippi

Let P = issa

11	i
8	ippi
5	issippi
2	ississippi
1	mississippi
10	pi
9	ppi
7	sippi
4	sisippi
6	ssippi
3	ssissippi

Supra inde

Structure

are implementation of

.

Simpl an arra to the te t suffi es listed in le icographical order.

-

If the suffi arra is , this binar search can perform because of the number of random disk accesses.

Suffi arra s are designed to allow done b comparing the contents of each pointer.

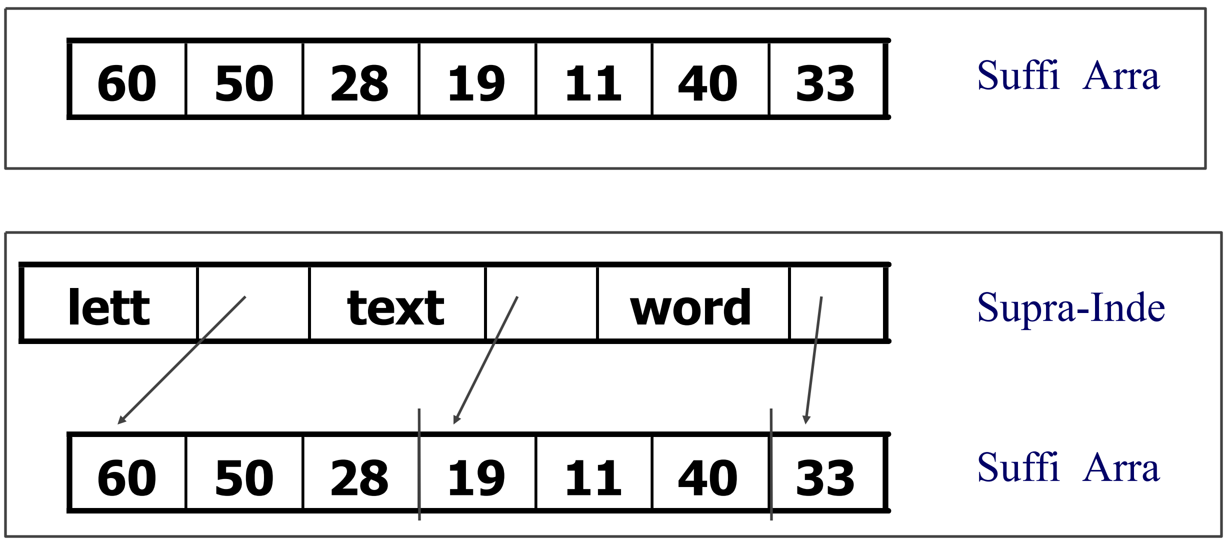
To remed this situation, the use of - over the suffi arra has been proposed.

Supra inde

E ample

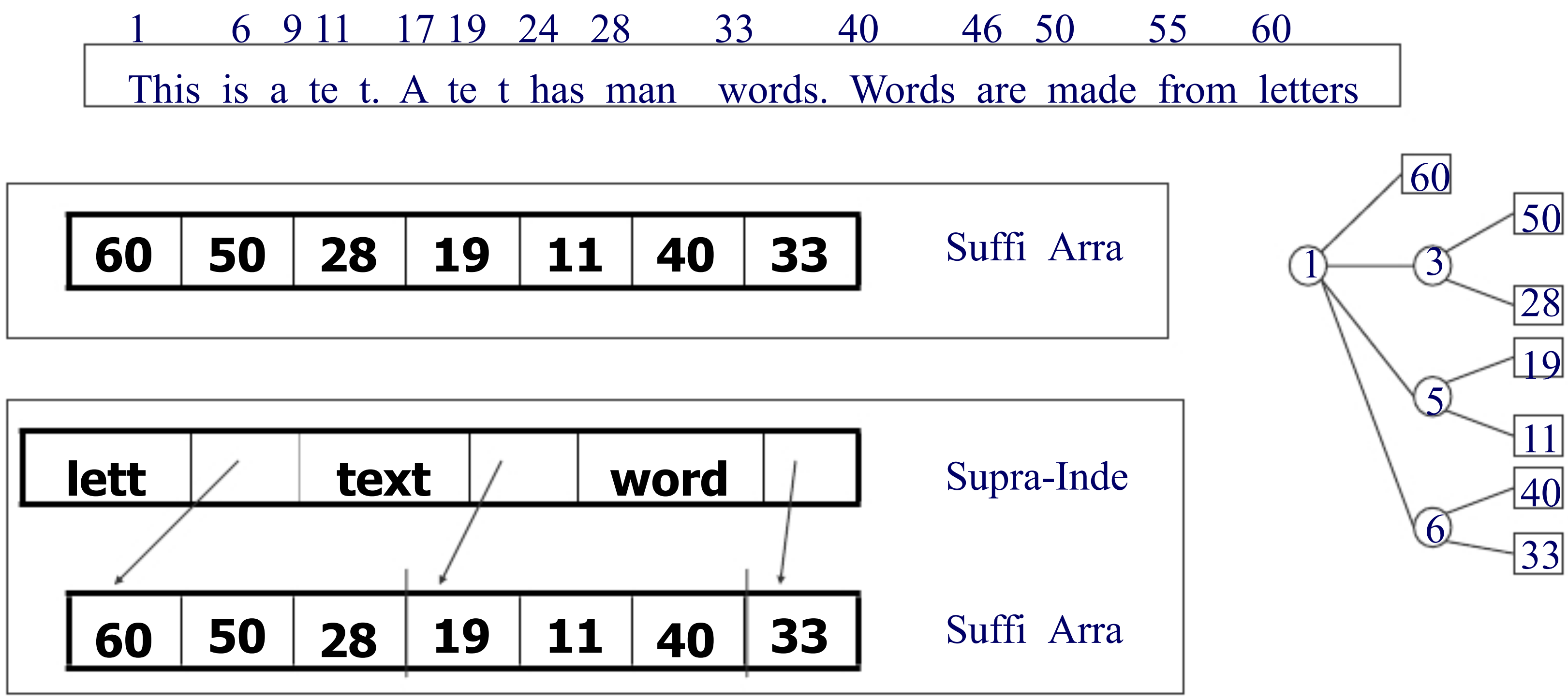
1 6 9 11 17 19 24 28 33 40 46 50 55 60

This is a te t. A te t has man words. Words are made from letters



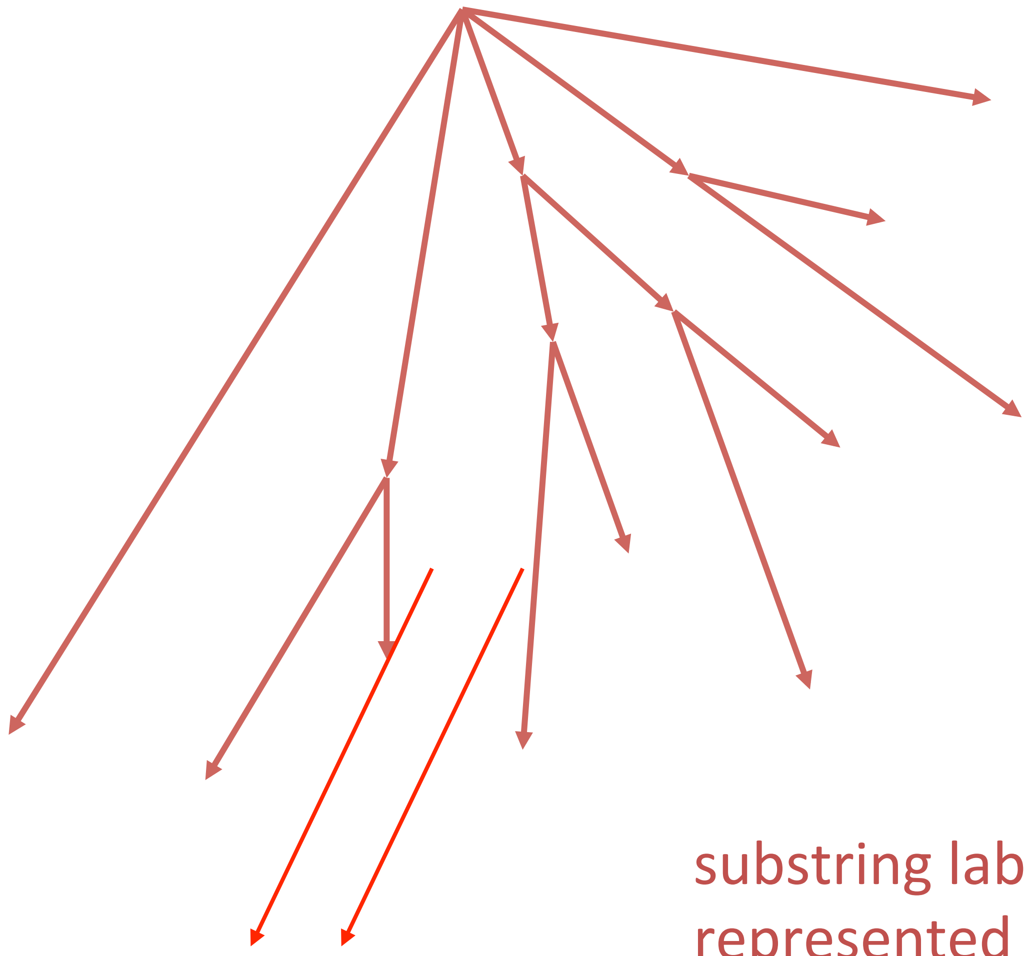
Supra inde

E ample



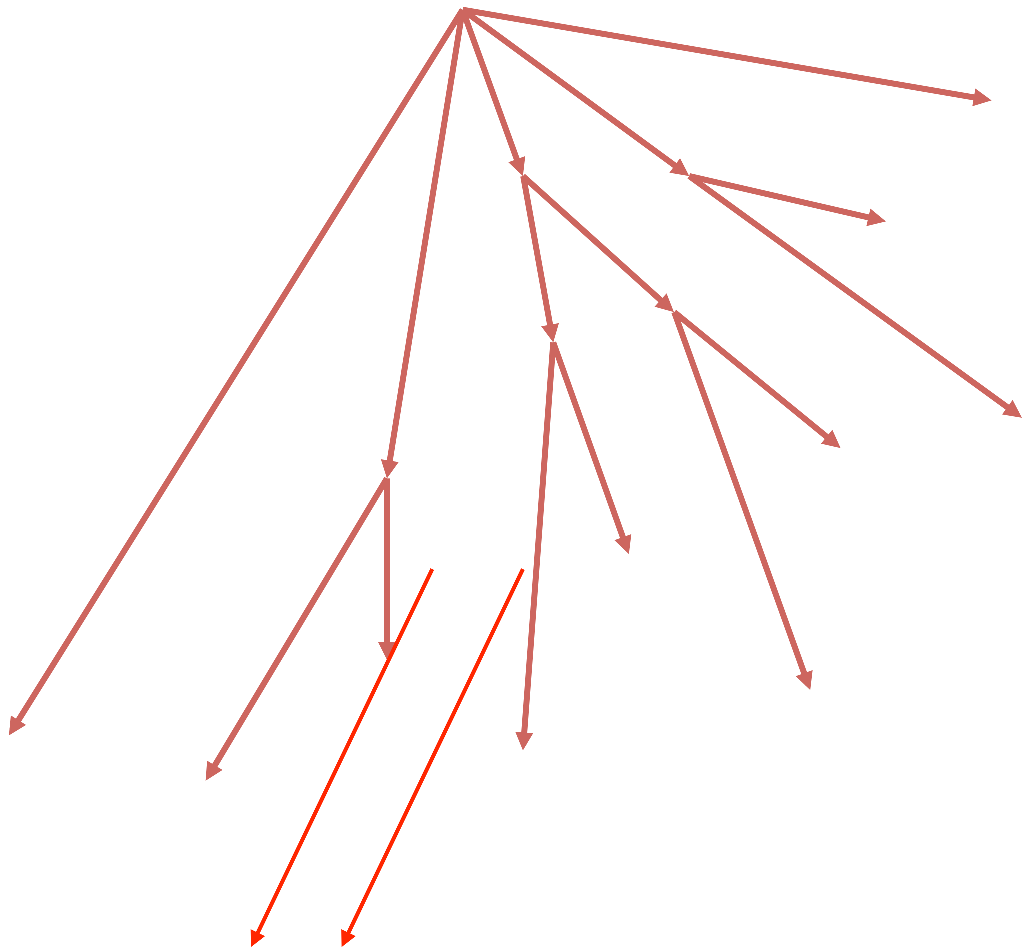
Tree(hattivatti)

Tree(hattivatti)

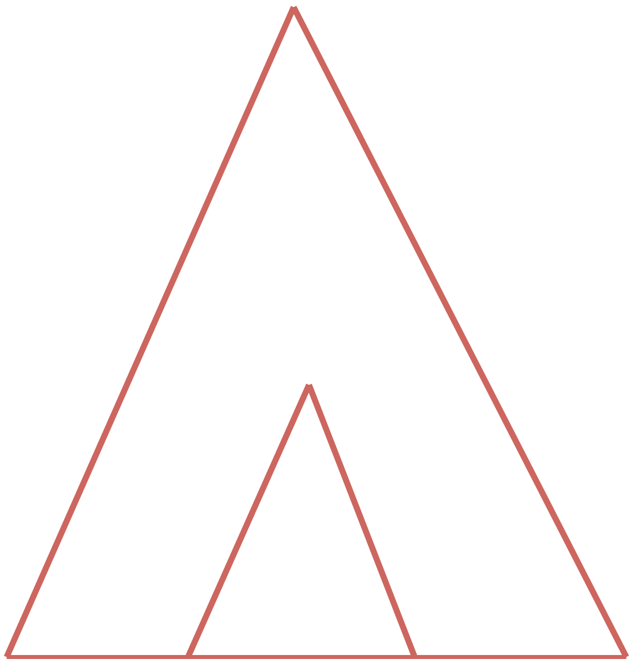


substring labels of edges represented as pairs of pointers

Tree(hattivatti)



Tree(T) is *full* text index



Find att from Tree(hattivatti)

Linear time construction of Tree(T)

Weiner (1973),

’algorithm of the year’

McCreight (1976)

’on-line’ algorithm (Ukkonen 1992)