First Applications of Suffix Trees
(7.12 ~ 7.13)

한양대학교
2015101331
Ko Daejin

발표자료 : 조윤성

Index

- **7.12 APL 11 : Finding all maximal repetitive structures in linear time.**

- **7.13 APL 12 : Circular string linearization.**

Finding all maximal repetitive structures

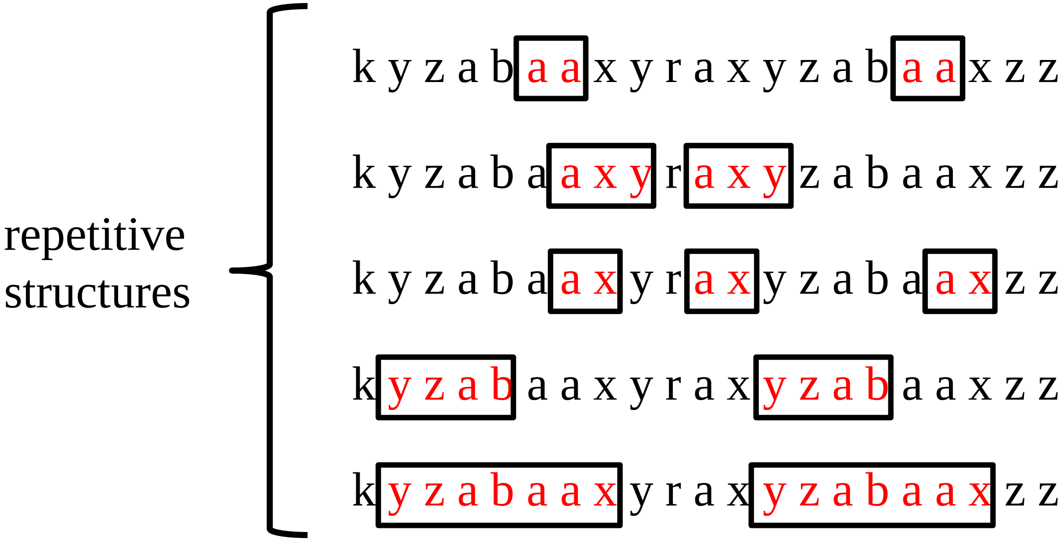
- **Finding all maximal repetitive structures in linear time**

  - Ex) S = k y z a b a a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = k y z a b a a x y r a x y z a b a a x z z



repetitive structures

k y z a b a a x y r a x y z a b a a x z z

k y z a b a a x y r a x y z a b a a x z z

k y z a b a a x y r a x y z a b a a x z z

k y z a b a a x y r a x y z a b a a x z z

k y z a b a a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = k y z a b a a x y r a x y z a b a a x z z

Naive $\boxed{\text{k y}}$ z a b a a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = k y z a b a a x y r a x y z a b a a x z z

Naive k y z a b a a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**
  - Ex) S = kyzabaaxyraxyzabaaxzz

Naive ky$\boxed{\text{za}}$baaxyraxyzabaaxzz

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = k y z a b a a x y r a x y z a b a a x z z

Naive k y z $\boxed{\text{a b}}$ a a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = kyzabaaxyraxyzabaaxzz

Naive k y z a $\boxed{b\ a}$ a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = k y z a b a a x y r a x y z a b a a x z z

Naive k y z a b a a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = kyzabaaxyraxyzabaaxzz

Naive kyzabaaxyraxyzabaaxzz

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = k y z a b a a x y r a x y z a b a a x z z

Naive k y z a b a a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = kyzabaaxyraxyzabaaxzz

Naive kyzabaaxyraxyzabaaxzz

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = k y z a b a a x y r a x y z a b a a x z z

Naive k y z a b a a x y r a x y z a b a a x z z

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**
  - Ex) S = kyzabaaxyraxyzabaaxzz

Naive kyzabaaxyraxyzabaaxzz

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**
  - Ex) S = k y z a b a a x y r a x y z a b a a x z z

Naive k y z a b a a x y r a x y $\boxed{\text{z a b a a x z z}}$

Consider $\Theta(n^4)$ pairs !!

Finding all maximal repetitive structures

- **Finding all maximal repetitive structures in linear time**

  - Ex) S = kyzabaaxyraxyzabaaxzz

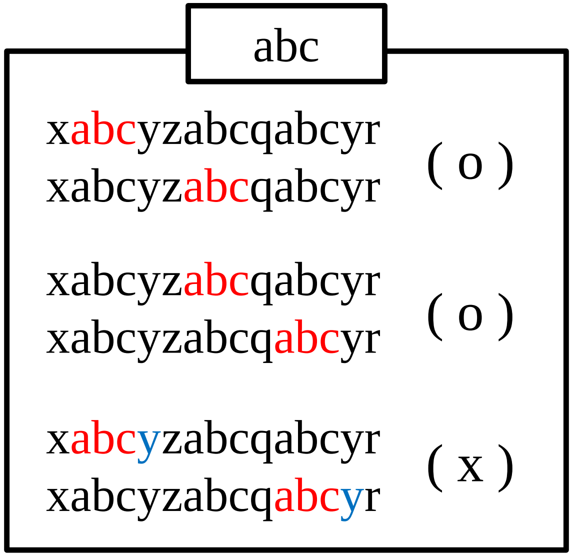
Naive kyzabaaxyraxy$\boxed{\text{zabaaxzz}}$

Consider $\Theta(n^4)$ pairs !!

➔ We can make it linear time

Finding all maximal repetitive structures

- **Maximal pair**
  - Identical substrings *α* and *β* in *S* such that the character to the immediate left(right) of *α* is different from the character to the immediate left(right) of *β*.
  - Ex) S = xabcyzabcqabcyr

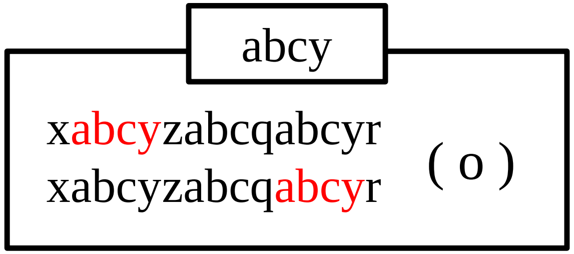


abc

xabcyzabcqabcyr
xabcyzabcqabcyr (o)

xabcyzabcqabcyr
xabcyzabcqabcyr (o)

xabcyzabcqabcyr
xabcyzabcqabcyr (x)



abcy

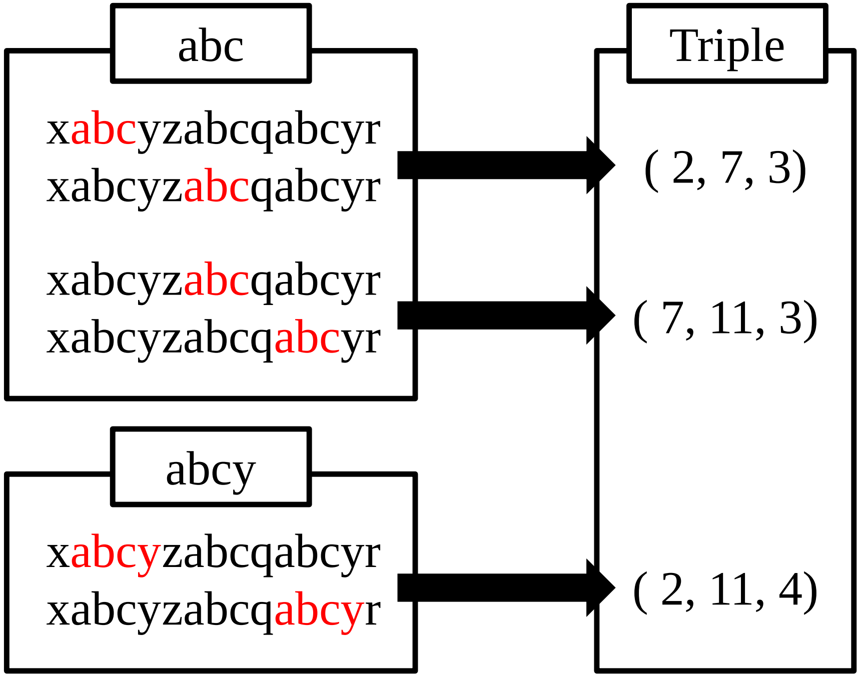
xabcyzabcqabcyr
xabcyzabcqabcyr (o)

Finding all maximal repetitive structures

- **Triple (*p*1, *p*2, *n*')**
  - A maximal pair is represented by the triple.
  - *p*1, *p*2 : starting positions of the two substrings
  - *n'* : substring length

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Ex) S = xabcyzabcqabcyr



abc

xabcyzabcqabcyr
xabcyzabcqabcyr

xabcyzabcqabcyr
xabcyzabcqabcyr

abcy

xabcyzabcqabcyr
xabcyzabcqabcyr

Triple

(2, 7, 3)

(7, 11, 3)

(2, 11, 4)

Finding all maximal repetitive structures

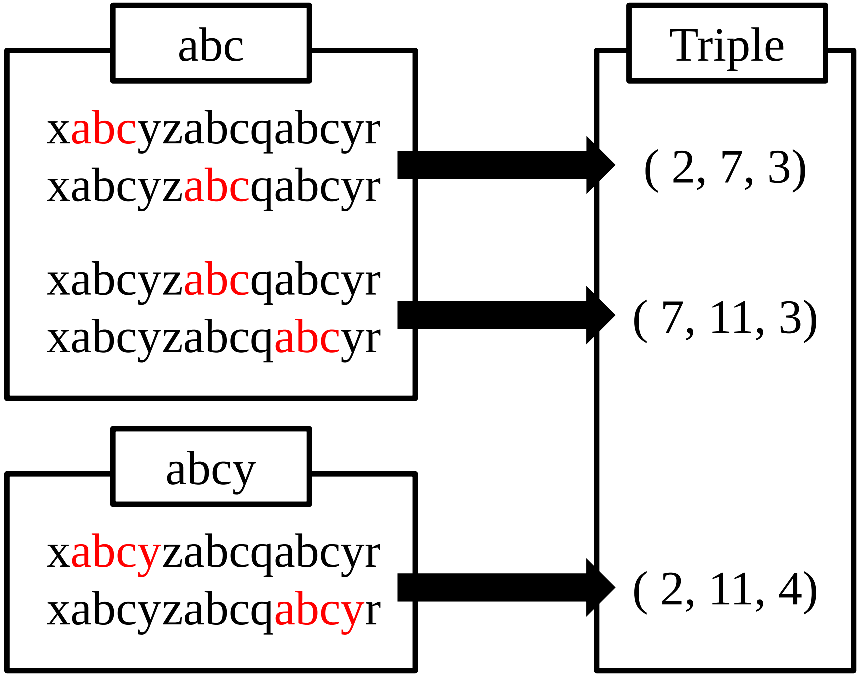
- **Triple (*p*1, *p*2, *n*')**
  - A maximal pair is represented by the triple.
  - *p*1, *p*2 : starting positions of the two substrings
  - *n'* : substring length

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Ex) S = xabcyzabcqabcyr

- **R(S)**
  - Set of all triples

  R(S) = { (2,7,3), (7,11,3), (2,11,4) }



abc

xabcyzabcqabcyr
xabcyzabcqabcyr

xabcyzabcqabcyr
xabcyzabcqabcyr

abcy

xabcyzabcqabcyr
xabcyzabcqabcyr

Triple

(2, 7, 3)

(7, 11, 3)

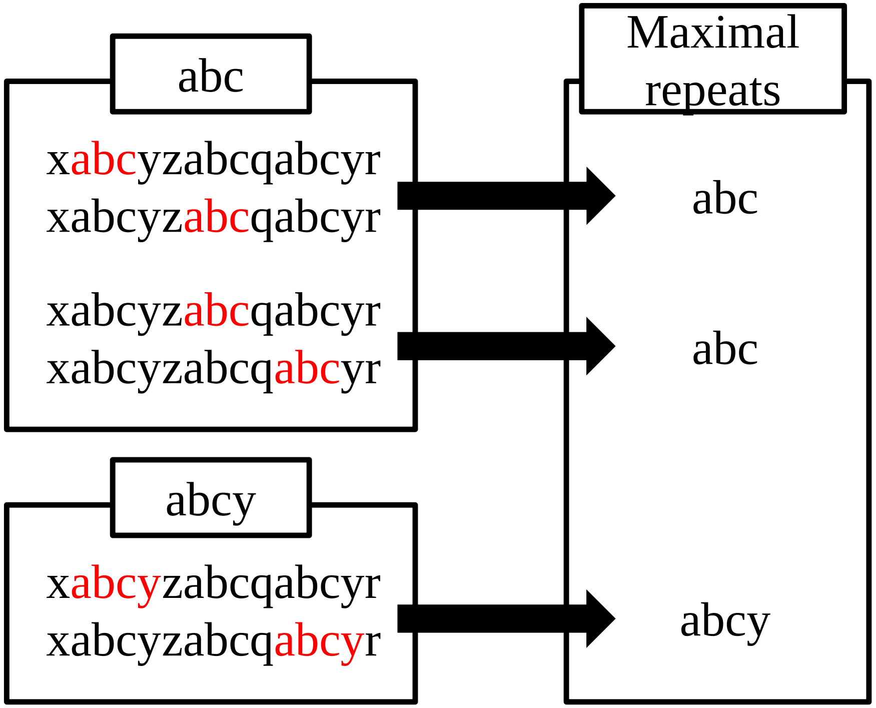
(2, 11, 4)

Finding all maximal repetitive structures

- **Maximal repeats α**
  - Substring of S that occurs in a maximal pair in S.
  - α is maximal repeat in S if there is a triple (p1, p2, |α|) ∈ R(S) and α occurs in S starting at position p1 and p2.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Ex) S = x a b c y z a b c q a b c y r



abc

xabcyzabcqabcyr
xabcyzabcqabcyr

xabcyzabcqabcyr
xabcyzabcqabcyr

abcy

xabcyzabcqabcyr
xabcyzabcqabcyr

Maximal repeats

abc

abc

abcy

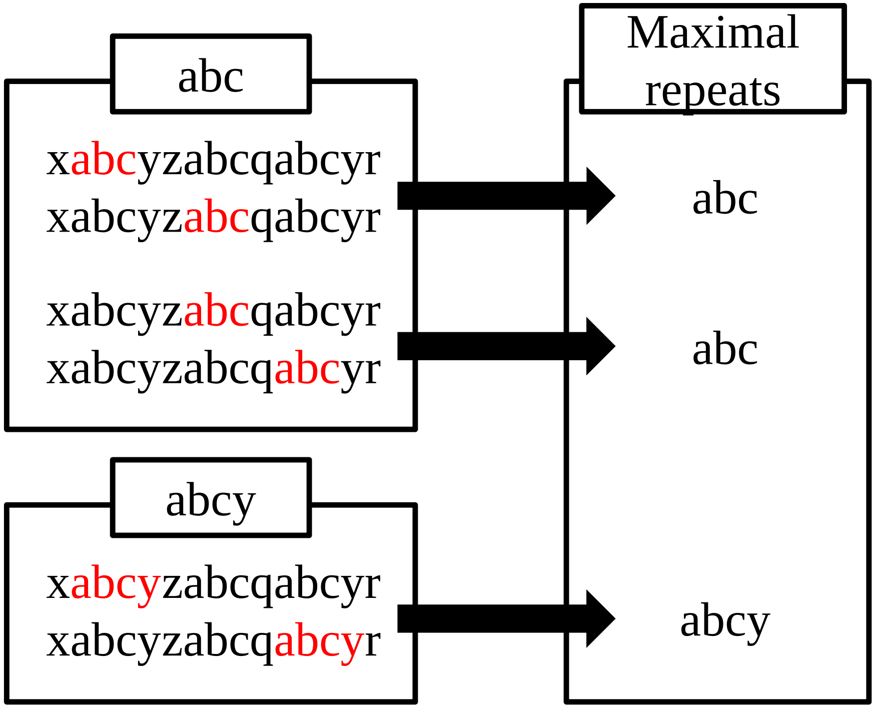
Finding all maximal repetitive structures

- **Maximal repeats α**
  - Substring of S that occurs in a maximal pair in S.
  - α is maximal repeat in S if there is a triple (p1, p2, |α|) ∈ R(S) and α occurs in S starting at position p1 and p2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ex) S =	x	a	b	c	y	z	a	b	c	q	a	b	c	y	r

- **R'(S)**
  - Set of maximal repeats

  R’(S) = { abc, abcy }



abc

xabcyzabcqabcyr
xabcyzabcqabcyr

xabcyzabcqabcyr
xabcyzabcqabcyr

abcy

xabcyzabcqabcyr
xabcyzabcqabcyr

Maximal repeats

abc

abc

abcy

Finding all maximal repetitive structures

- **Supermaximal repeat**
  - Maximal repeat that never occurs as a substring of any other maximal repeat.

R(S) = { (2,7,3), (7,11,3), (2,11,4) }

R'(S) = { abc, abcy }

Supermaximal repeat of S = 'abcy'

A linear time algorithm to find all maximal repeats

- **T = Suffix tree for string S**

- **If a string α is maximal repeat in S then α is the path-label of a node v in T.**

Ex) S = x α y α z (α = substring)

A linear time algorithm to find all maximal repeats

- **T = Suffix tree for string S**

- **If a string α is maximal repeat in S then α is the path-label of a node v in T.**

Ex) S = **x α y α z** (α = substring)

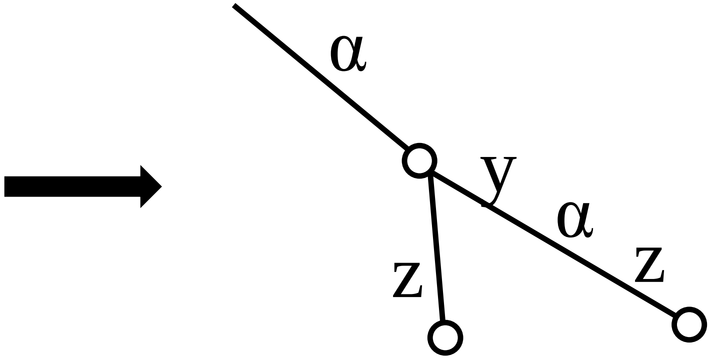
A linear time algorithm to find all maximal repeats

- **T = Suffix tree for string S**
- **If a string α is maximal repeat in S then α is the path-label of a node v in T.**

Ex) S = x α y α z (α = substring)

x α y α z

x α y α z



A linear time algorithm to find all maximal repeats

- **S(i-1), left character**
  - The left character of a leaf of T is the left character of the suffix position represented by that leaf.

Ex)

1	2	3	4	5	6
x	a	b	x	a	$

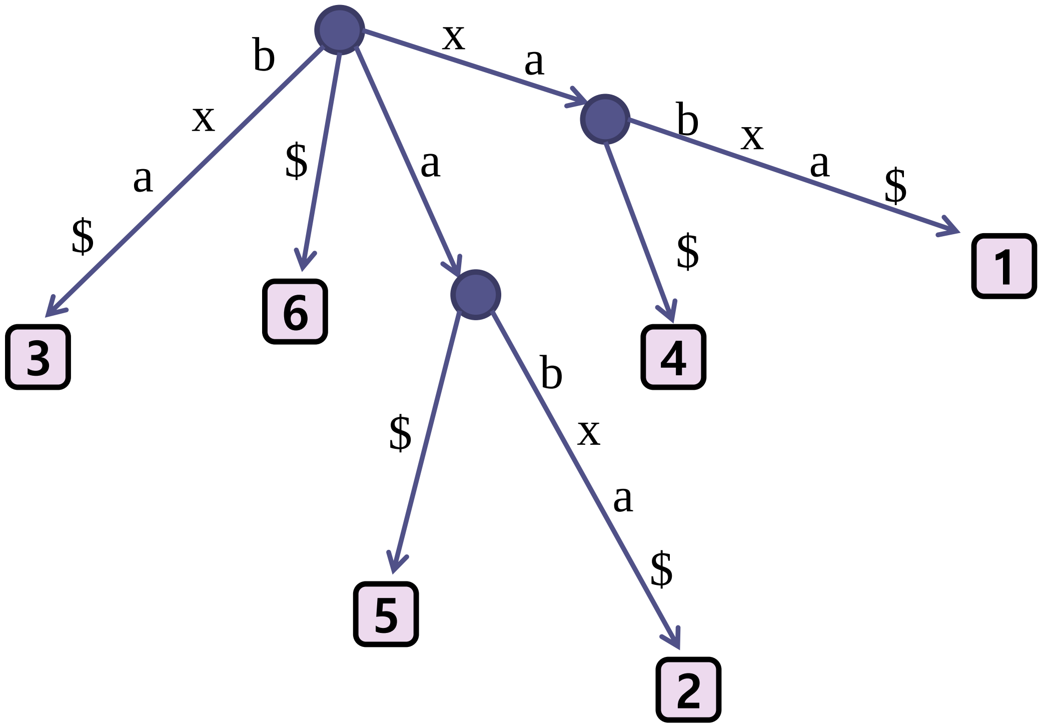
S(1) = x
S(2) = a
S(3) = b
S(4) = x
S(5) = a
S(6) = $

A linear time algorithm to find all maximal repeats

- **S(i-1), left character**
  - The left character of a leaf of T is the left character of the suffix position represented by that leaf.

Ex)

1	2	3	4	5	6
x	a	b	x	a	$

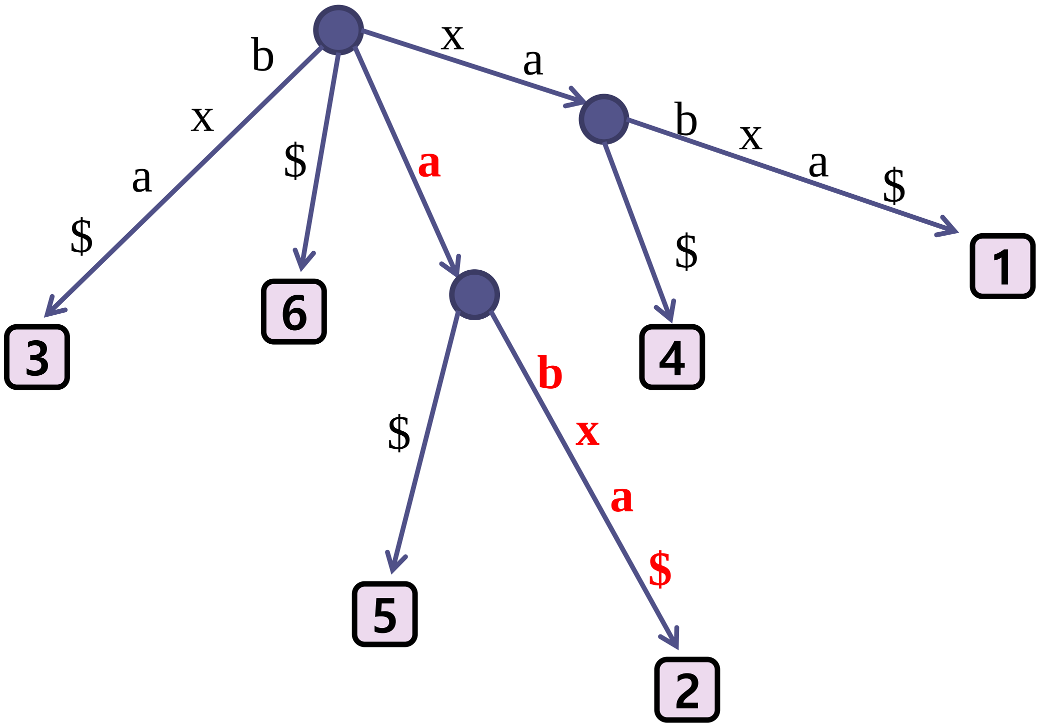


A linear time algorithm to find all maximal repeats

- **S(i-1), left character**
  - The left character of a leaf of T is the left character of the suffix position represented by that leaf.

Ex)

1	2	3	4	5	6
x	a	b	x	a	$



Left character of $\boxed{2}$ is ‘**x**’

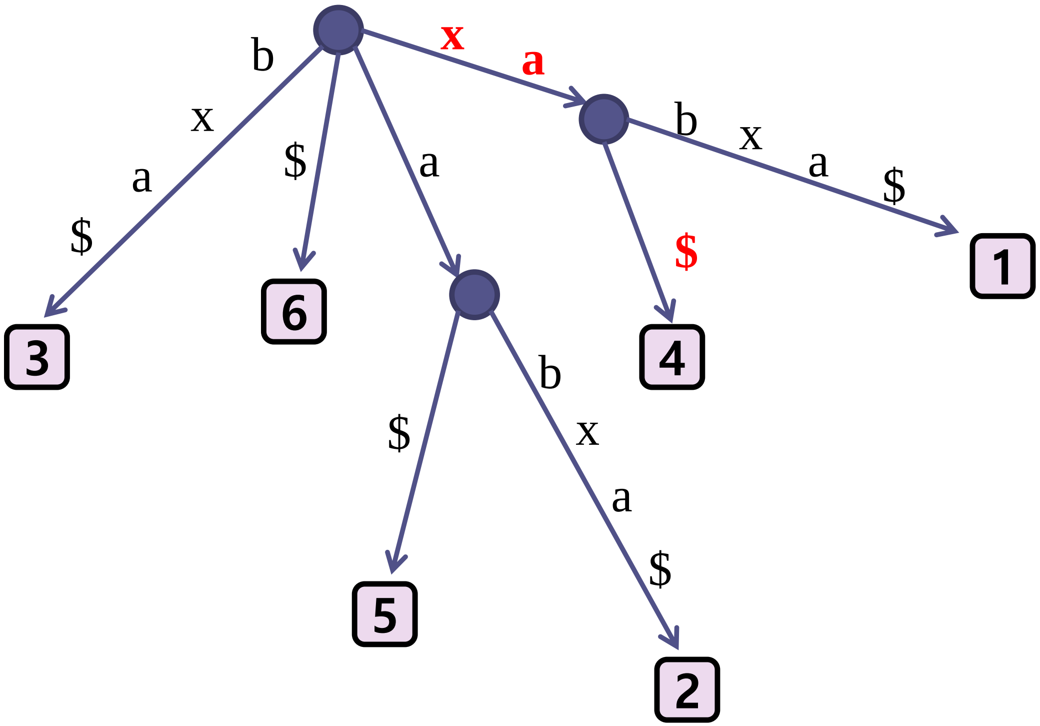
A linear time algorithm to find all maximal repeats

- **S(i-1), left character**
  - The left character of a leaf of T is the left character of the suffix position represented by that leaf.

1 2 3 4 5 6

Ex) x a b x a $

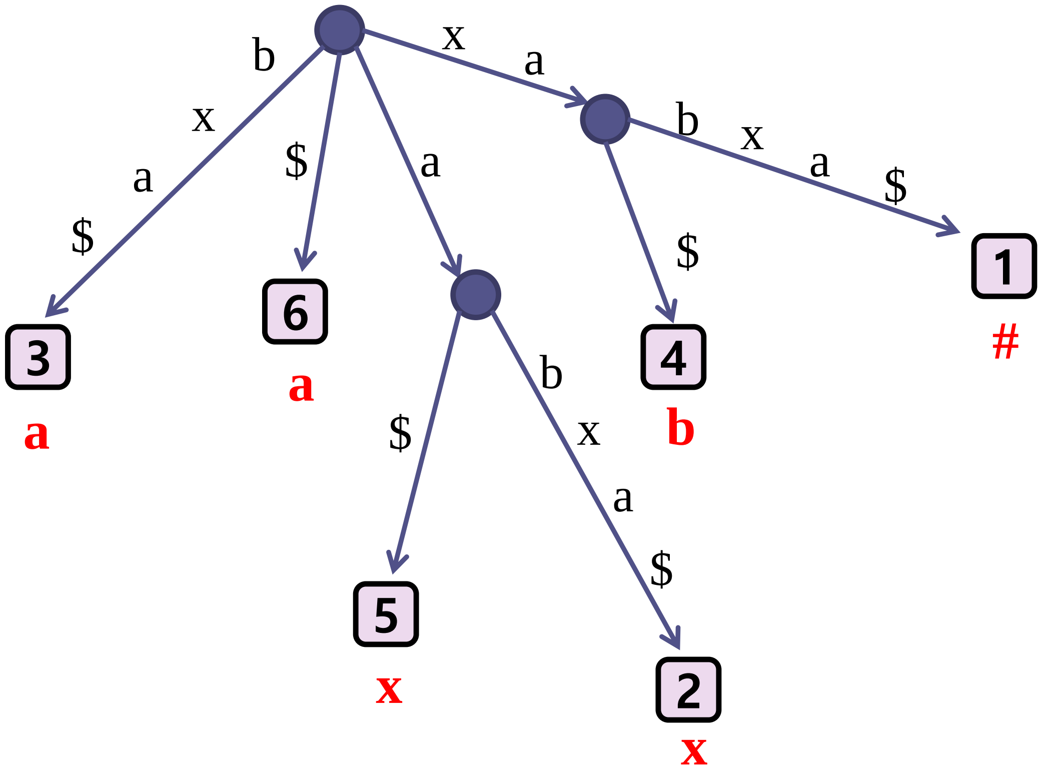
Left character of 4 is ‘**b**’



A linear time algorithm to find all maximal repeats

- **S(i-1), left character**
  - The left character of a leaf of T is the left character of the suffix position represented by that leaf.

	1	2	3	4	5	6
Ex)	x	a	b	x	a	$



A linear time algorithm to find all maximal repeats

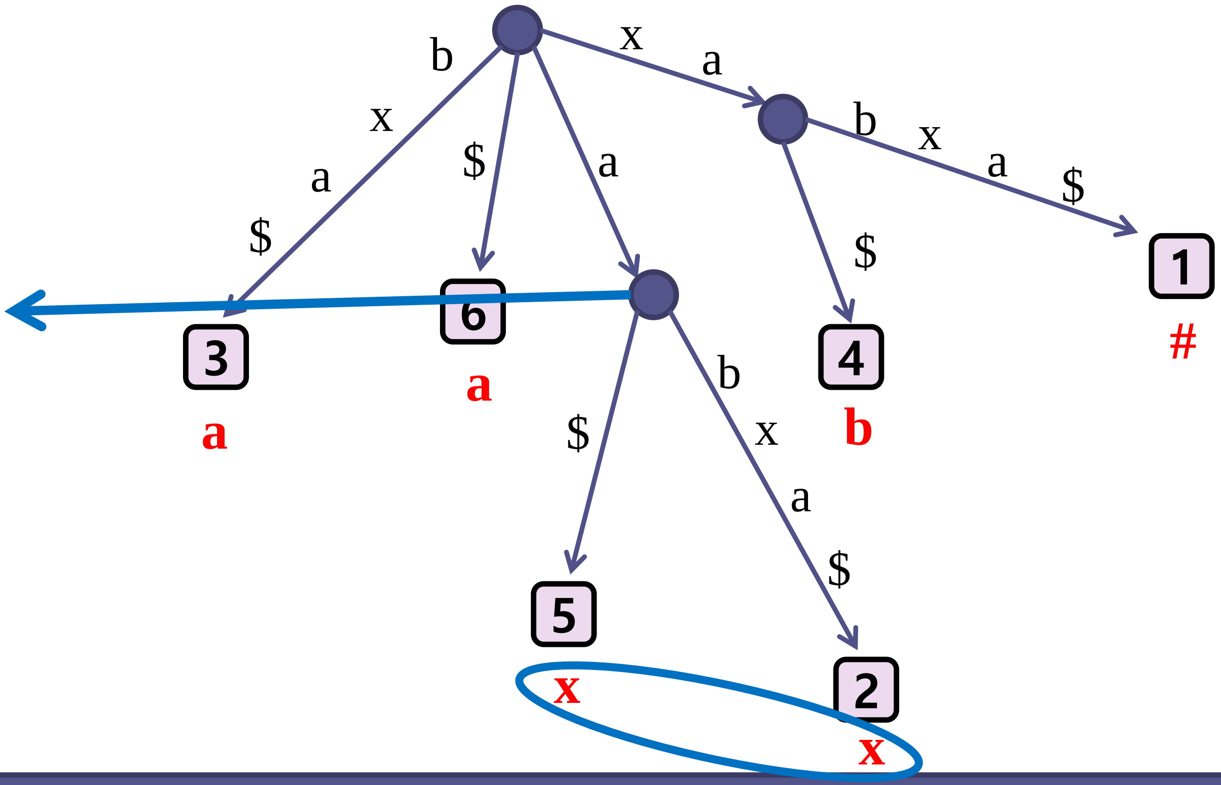
- **Left diverse**
  - A node *v* is called left diverse if at least two leaves in *v*’s subtree have different left characters.

A linear time algorithm to find all maximal repeats

- **Left diverse**
  - A node *v* is called left diverse if at least two leaves in *v*'s subtree have different left characters.

1 2 3 4 5 6

Ex) x a b x a $



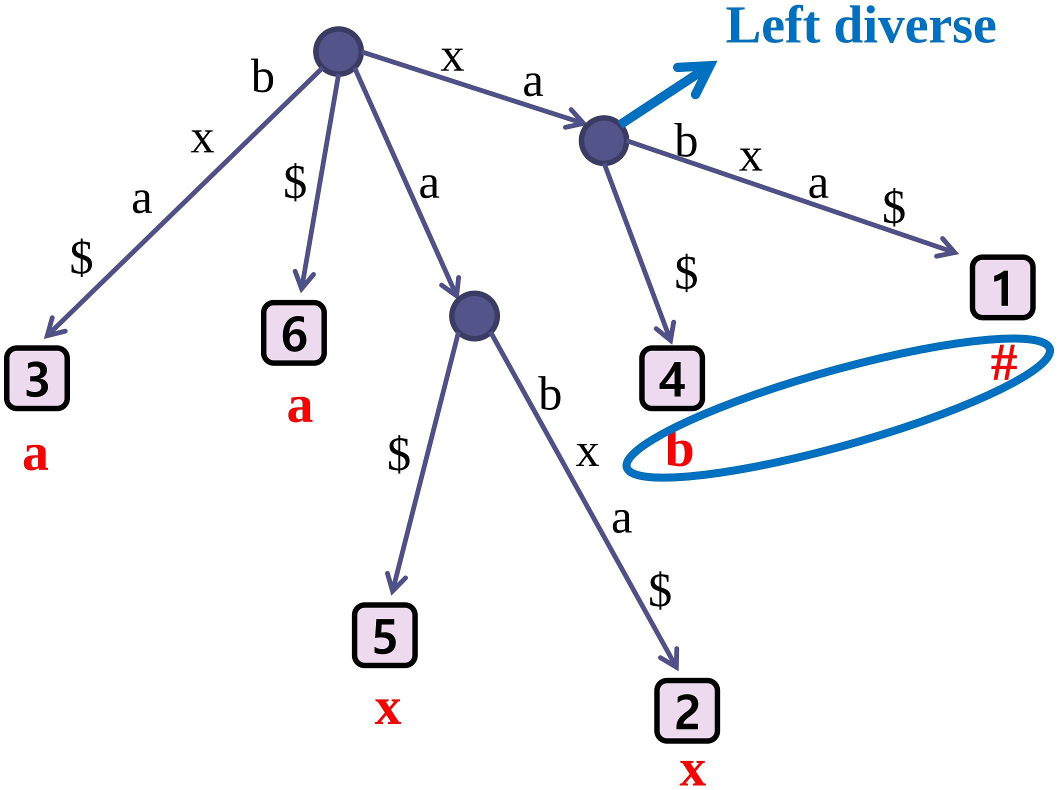
It is not a Left divers

A linear time algorithm to find all maximal repeats

- **Left diverse**
  - A node *v* is called left diverse if at least two leaves in *v*'s subtree have different left characters.

Ex)

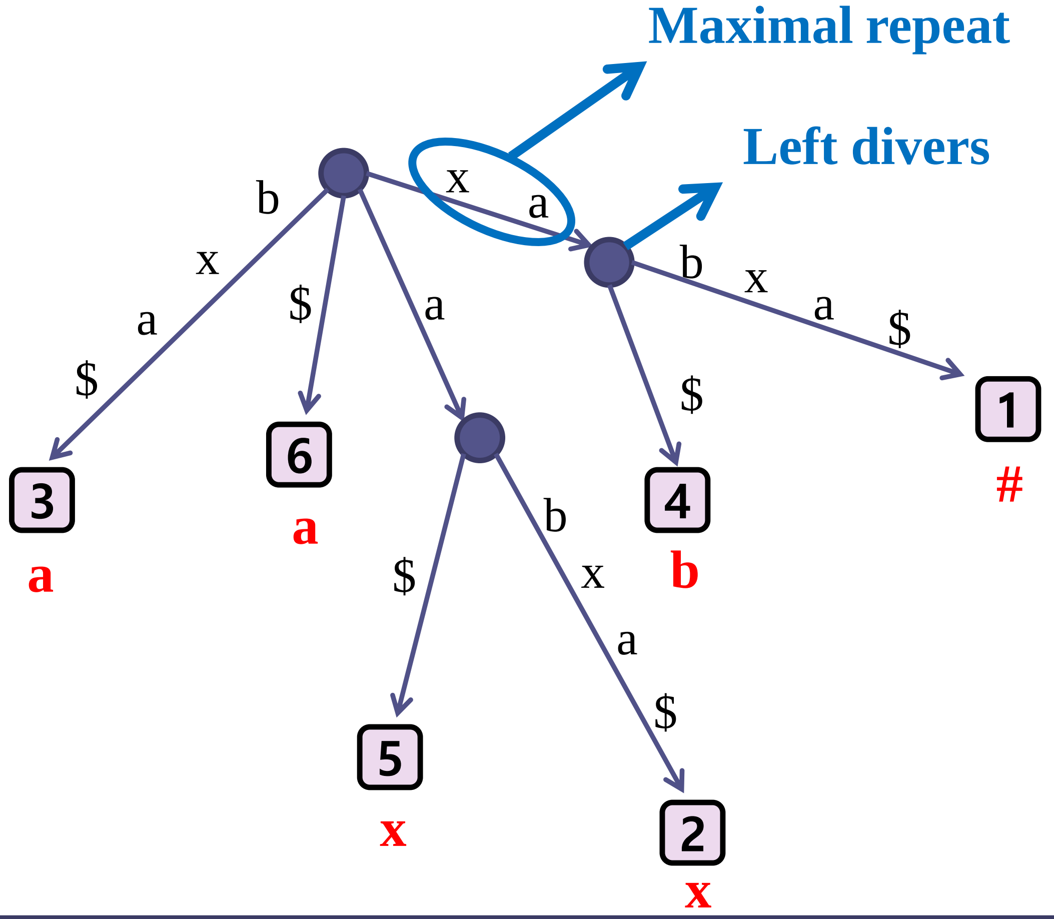
1	2	3	4	5	6
x	a	b	x	a	$



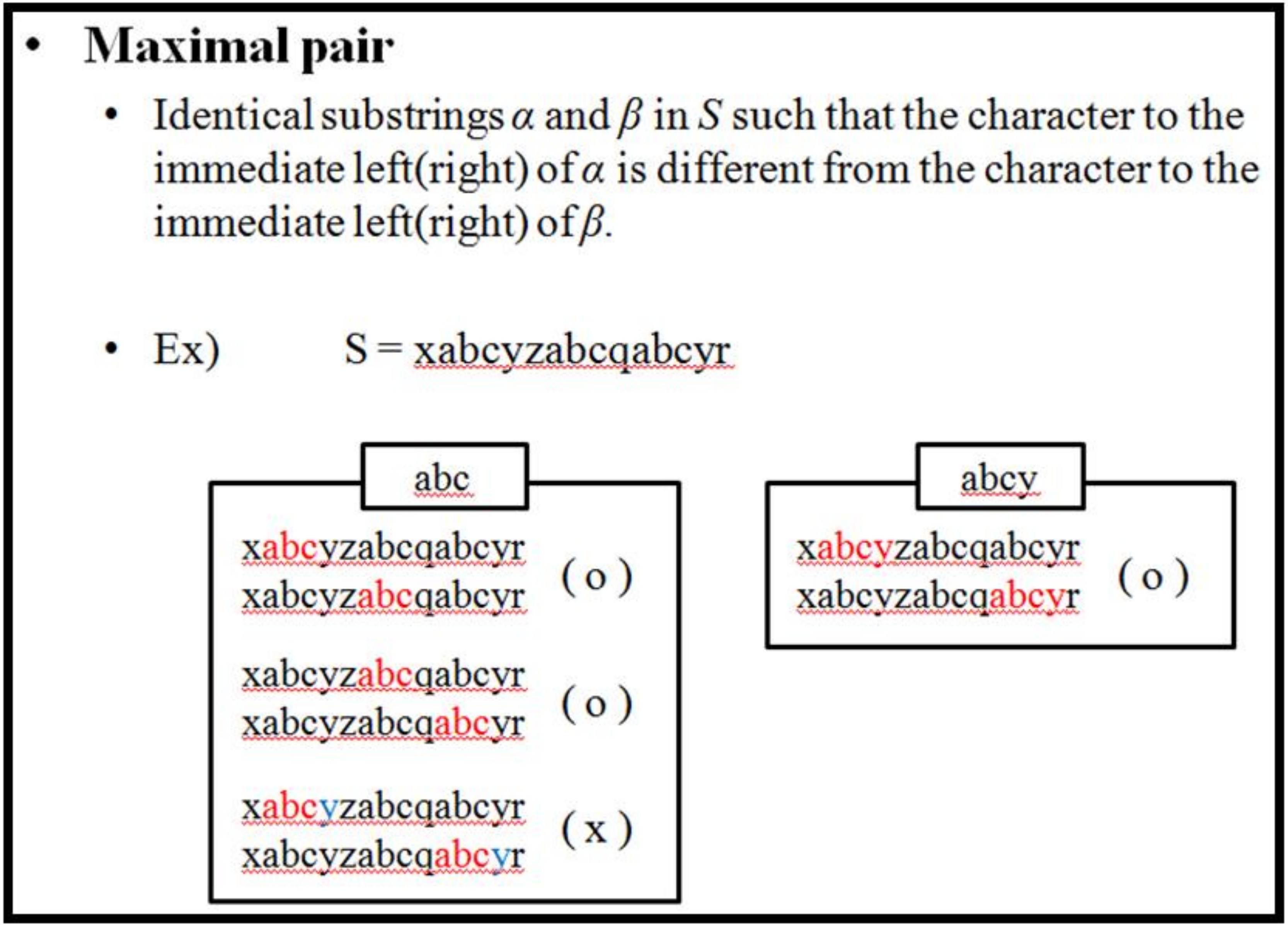
Left diverse

A linear time algorithm to find all maximal repeats

- **Theorem**
  - The string α labeling the path to a node *v* is a maximal repeat if and only if *v* is left diverse.



A linear time algorithm to find all maximal



- **Maximal pair**
  - Identical substrings *α* and *β* in *S* such that the character to the immediate left(right) of *α* is different from the character to the immediate left(right) of *β*.
  - Ex) S = xabcyzabcqabcyr

abc

xabcyzabcqabcyr
xabcyzabcqabcyr (o)

xabcyzabcqabcyr
xabcyzabcqabcyr (o)

xabcyzabcqabcyr
xabcyzabcqabcyr (x)

abcy

xabcyzabcqabcyr
xabcyzabcqabcyr (o)

t

1

#

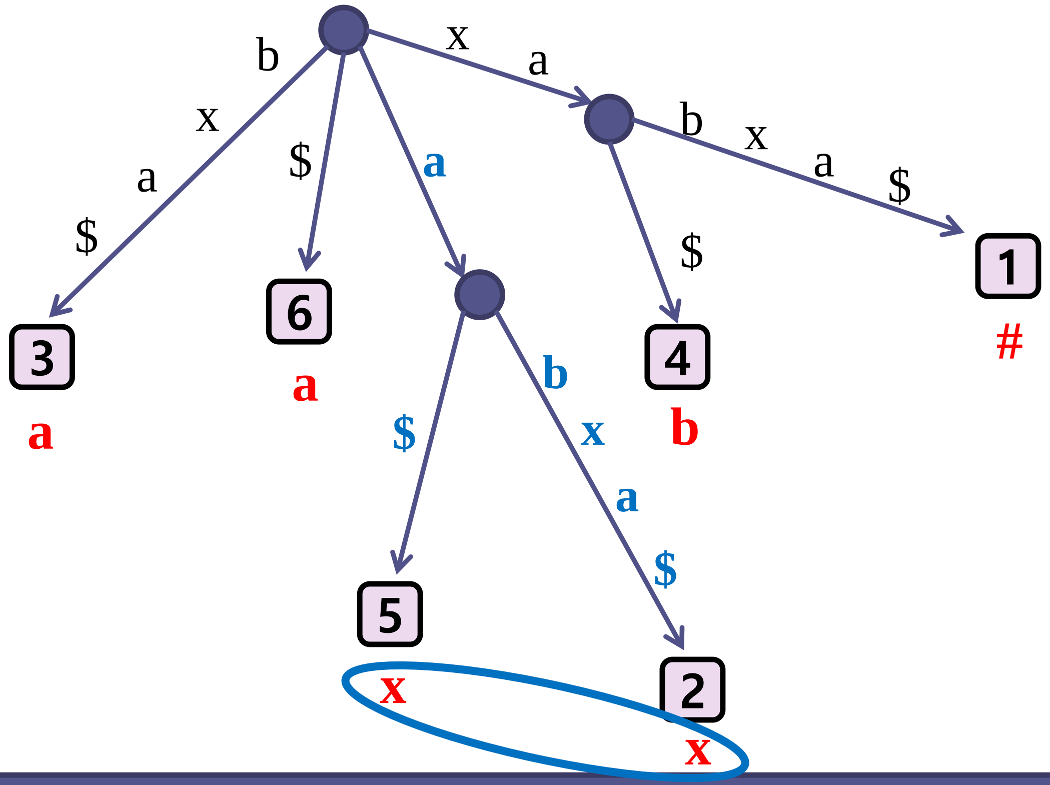
x

A linear time algorithm to find all maximal repeats

- **Theorem**
  - The string α labeling the path to a node *v* is a maximal repeat if and only if *v* is left diverse.

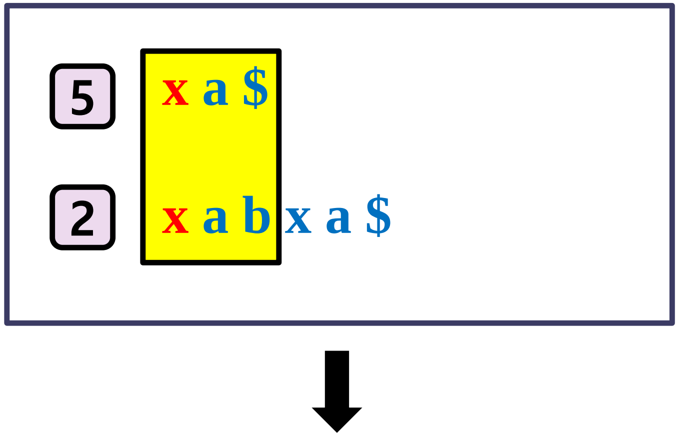
5 x a $

2 x a b x a $

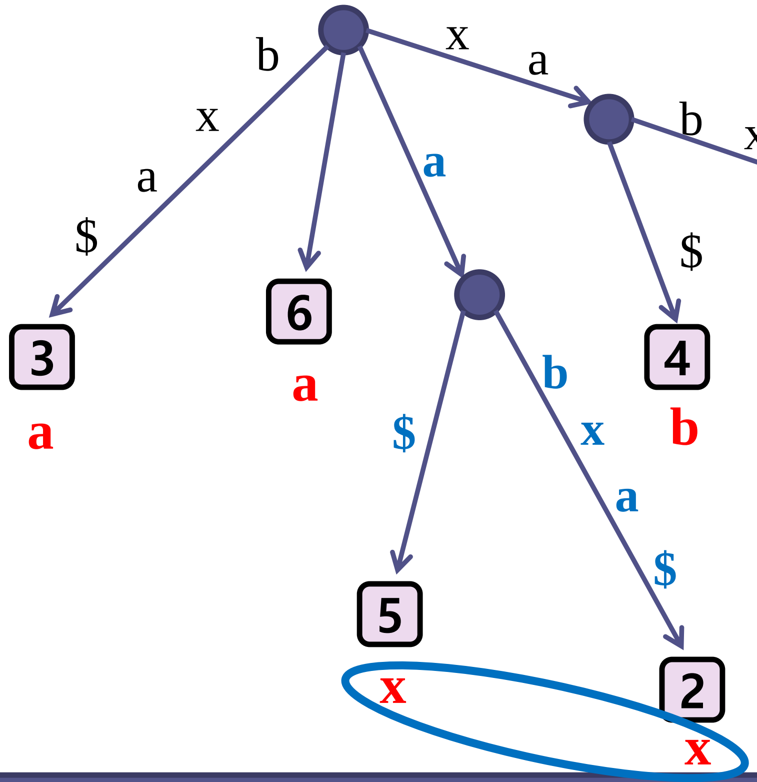


A linear time algorithm to find all maximal repeats

- **Theorem**
  - The string α labeling the path to a node *v* is a maximal repeat if and only if *v* is left diverse.



Not a maximal repeat

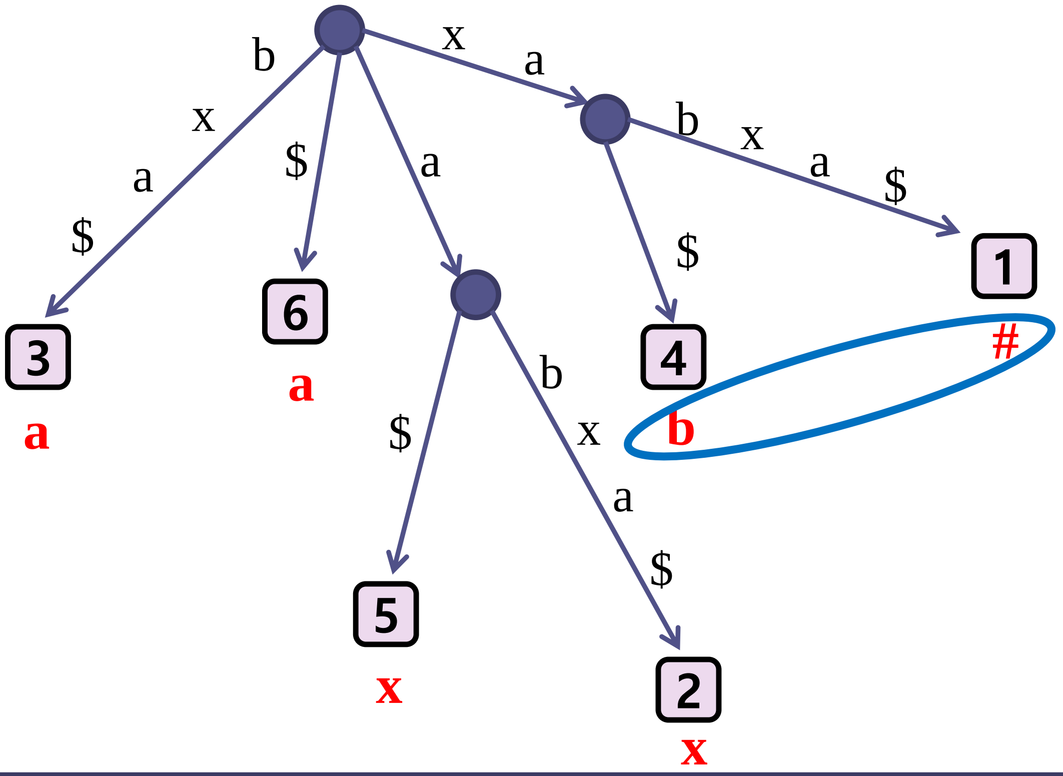


A linear time algorithm to find all maximal repeats

- **Theorem**
  - The string α labeling the path to a node *v* is a maximal repeat if and only if *v* is left diverse.

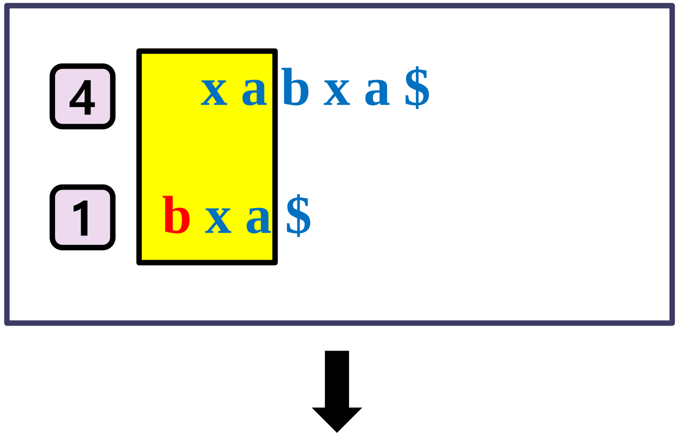
4 x a b x a $

1 b x a $

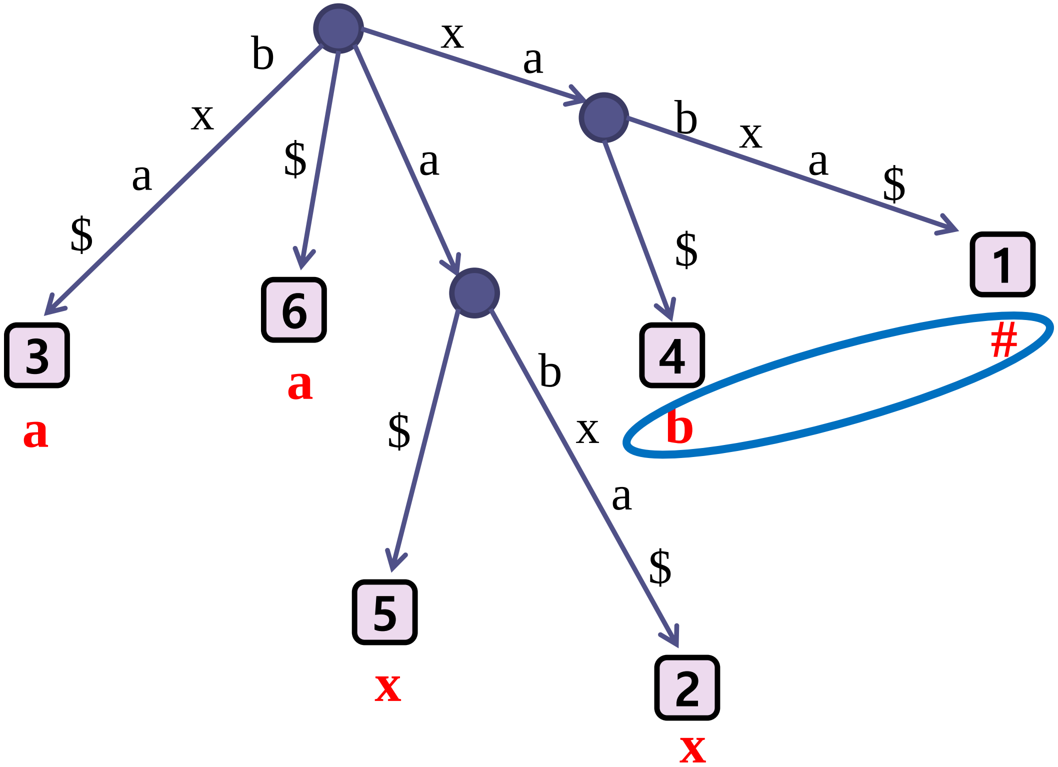


A linear time algorithm to find all maximal repeats

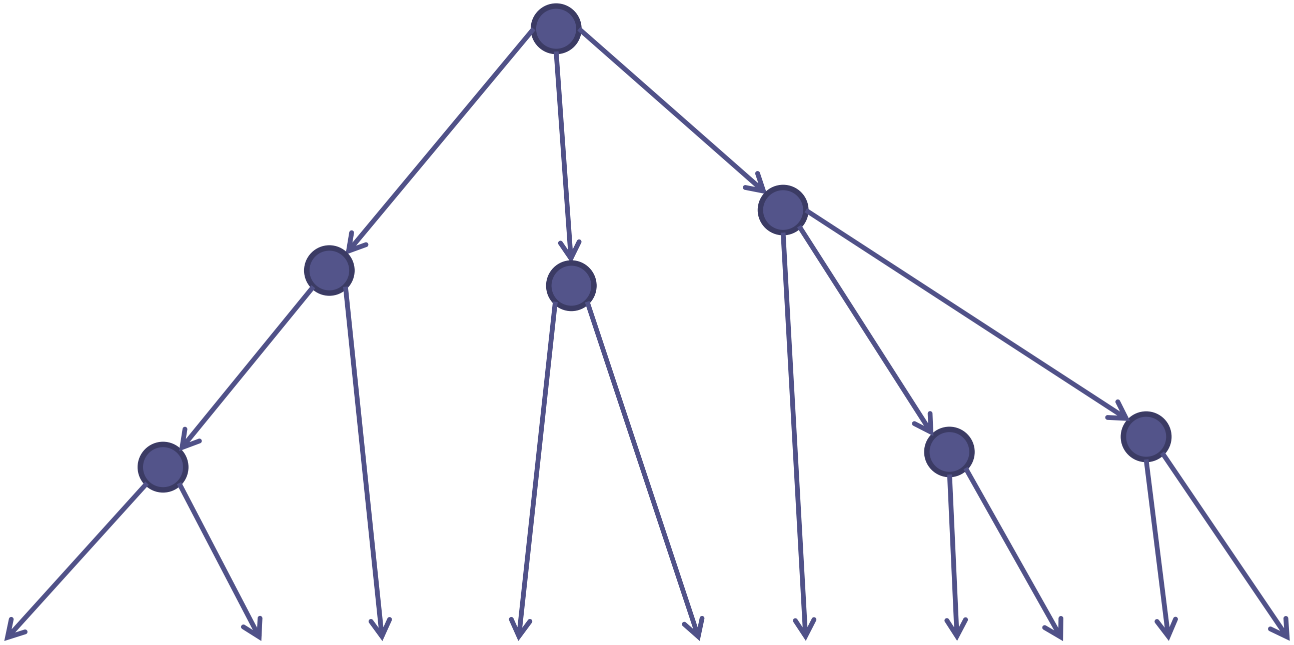
- **Theorem**
  - The string α labeling the path to a node *v* is a maximal repeat if and only if *v* is left diverse.



maximal repeat

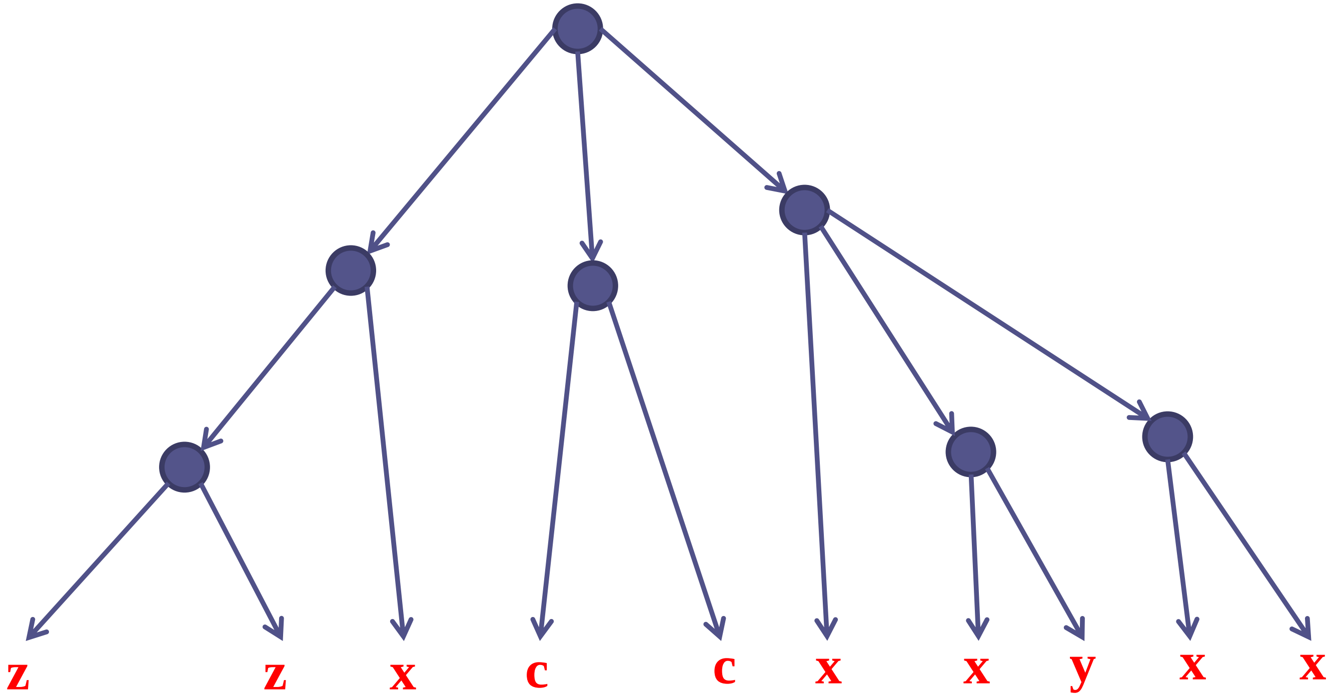


Find left diverse nodes in linear time



Find left diverse nodes in linear time

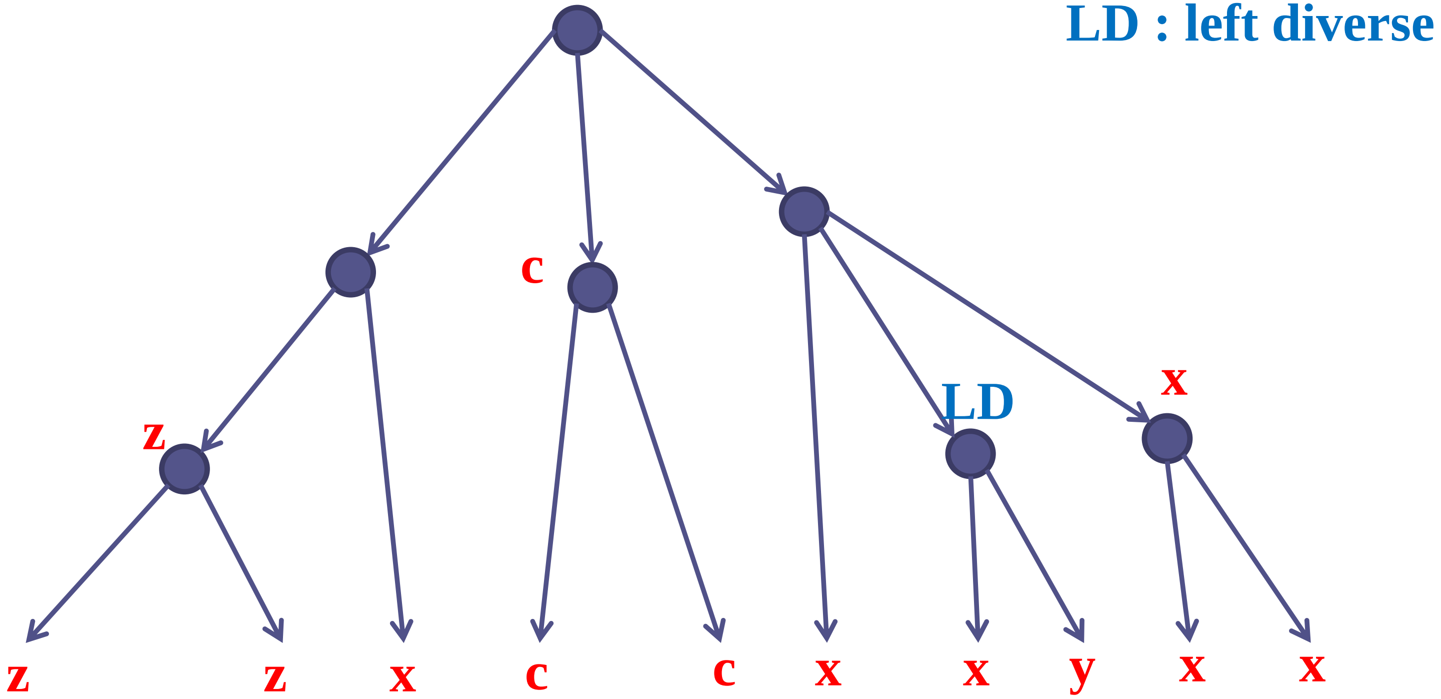
1. Records the left character of every leaf



Find left diverse nodes in linear time

2. (a) If any child of *v* has been identified as being left diverse, it records that v is left diverse

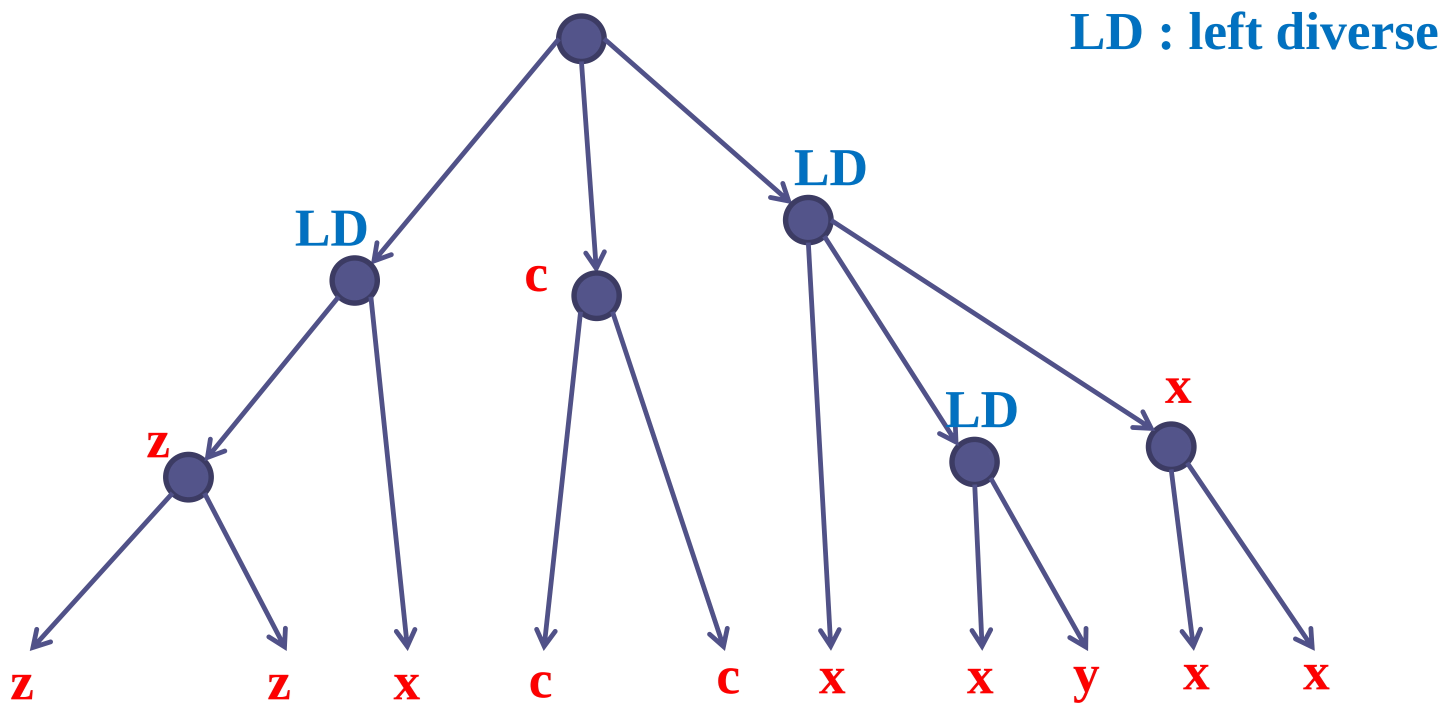
   (b) else, examines the characters recorded at *v*'s children



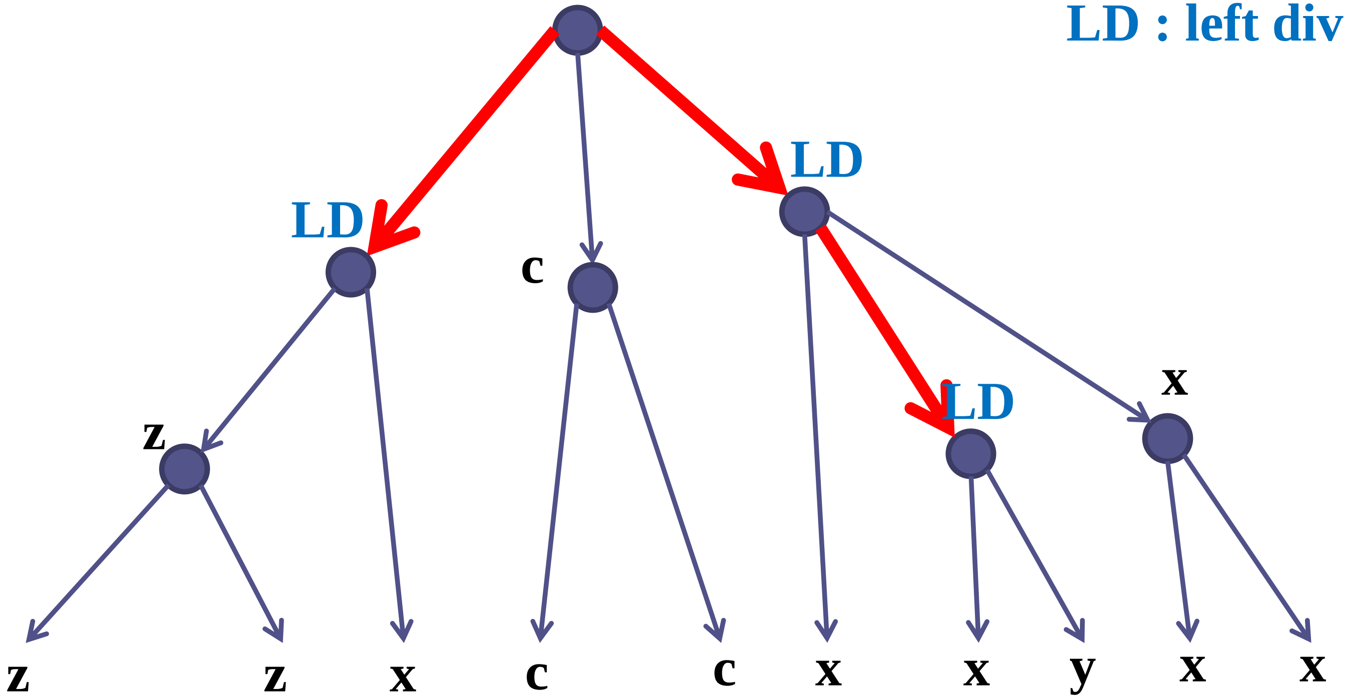
Find left diverse nodes in linear time

2. (a) If any child of *v* has been identified as being left diverse, it records that v is left diverse

   (b) else, examines the characters recorded at *v*'s children



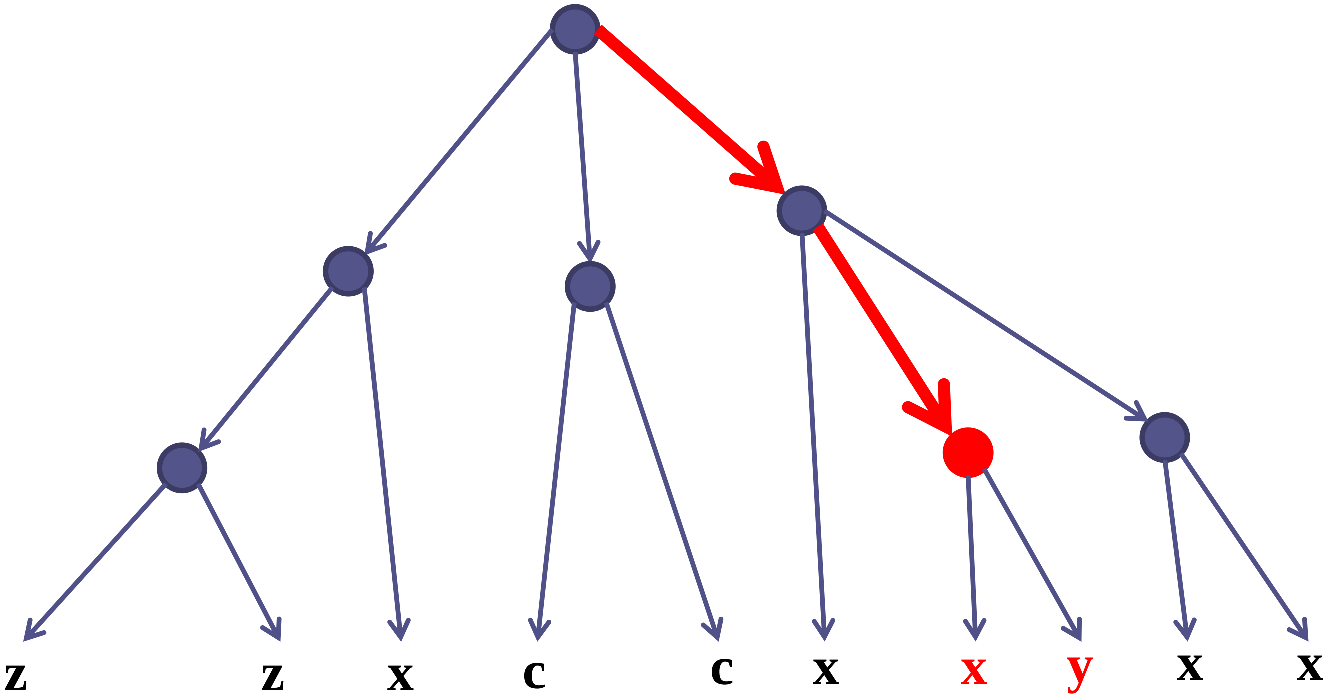
Find left diverse nodes in linear time



LD : left diverse

Find supermaximal repeats in linear time

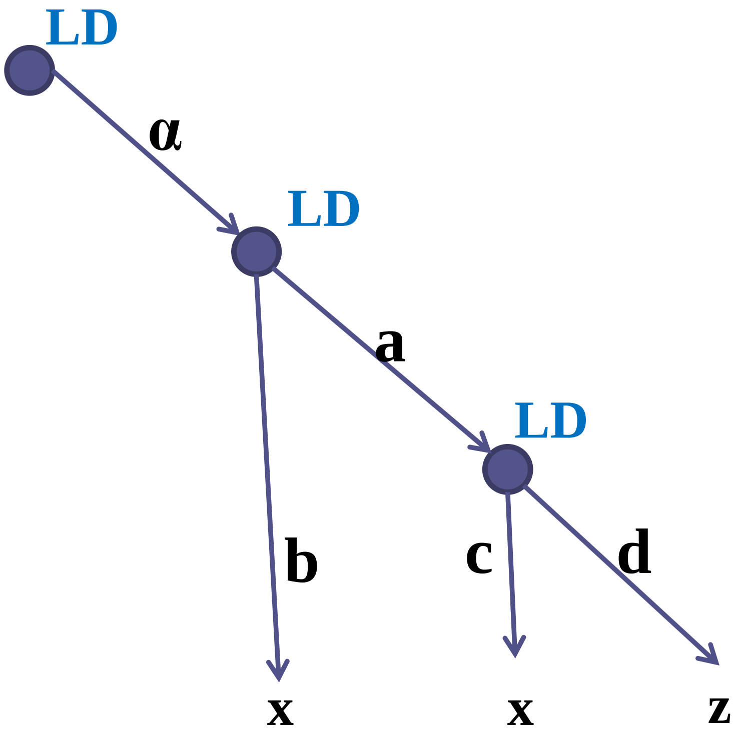
- **Supermaximal repeat**
  - Maximal repeat that is not a substring of any other maximal repeat.
  - Node v represents a supermaximal repeat α if and only if..
    1. Each children has a distinct left character.
    2. All of v’s children are leaves



Find supermaximal repeats in linear time

2. All of v’s children are leaves

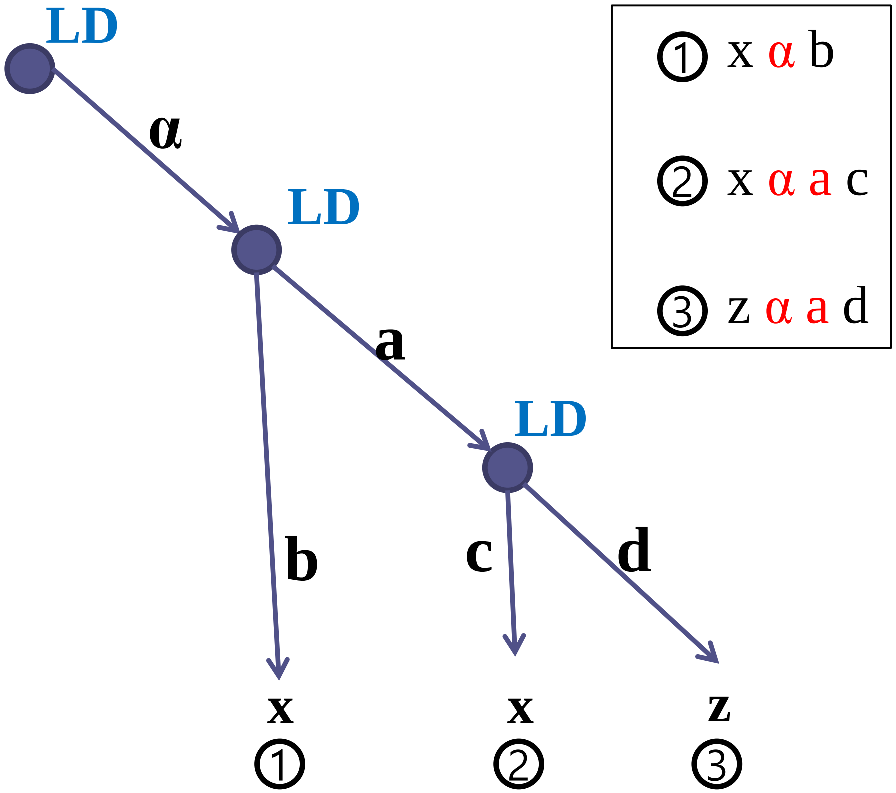
LD : left diverse



Find supermaximal repeats in linear time

2. All of v’s children are leaves

LD : left diverse



① x α b

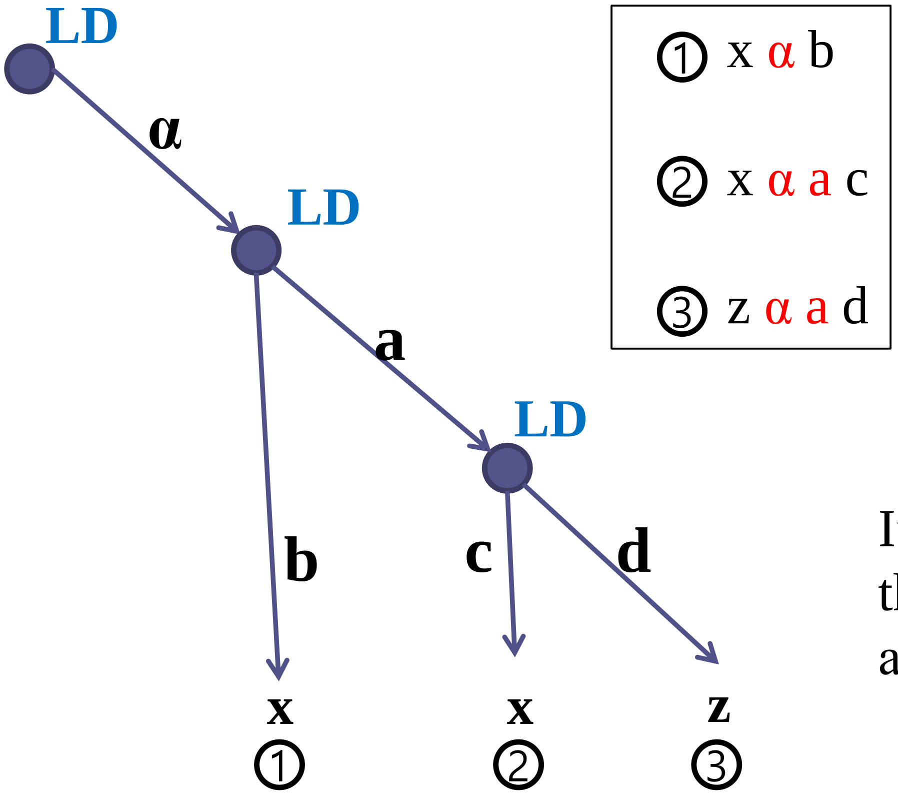
② x α a c

③ z α a d

Find supermaximal repeats in linear time

2. All of v’s children are leaves

LD : left diverse



① x α b

② x α a c

③ z α a d

⇒ Supermaximal repeat : α a

If v’s children are not leaves,
there must exist sharing branch
and it can’t be Supermaximal repeat.

Find all the maximal pairs in linear time

- The algorithm is an extension of the method given earlier to find all maximal repeats.

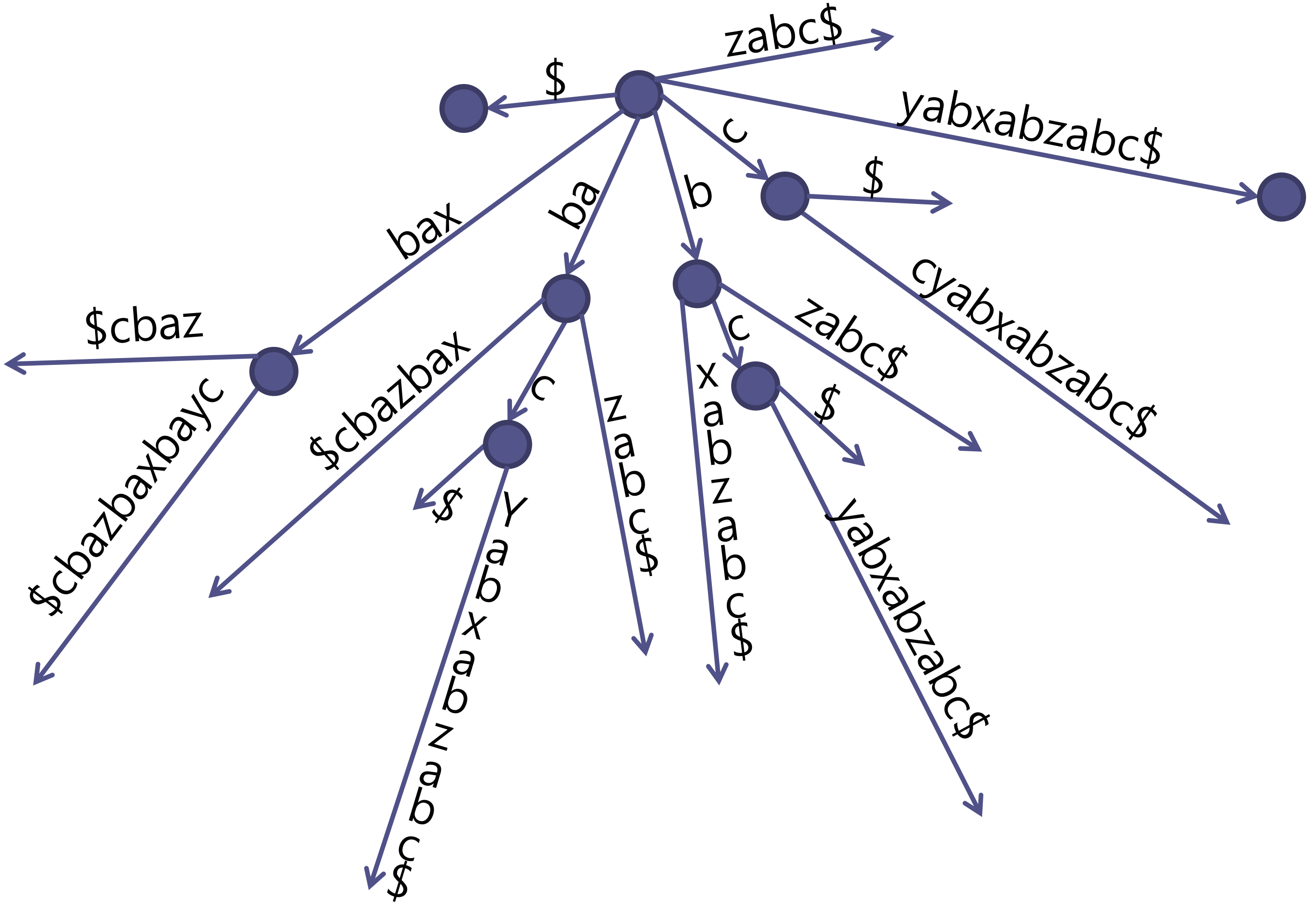
- Save left character with it’s postion.

- Working bottom up

- When calculate node *v*, Save the information about *v*'s children

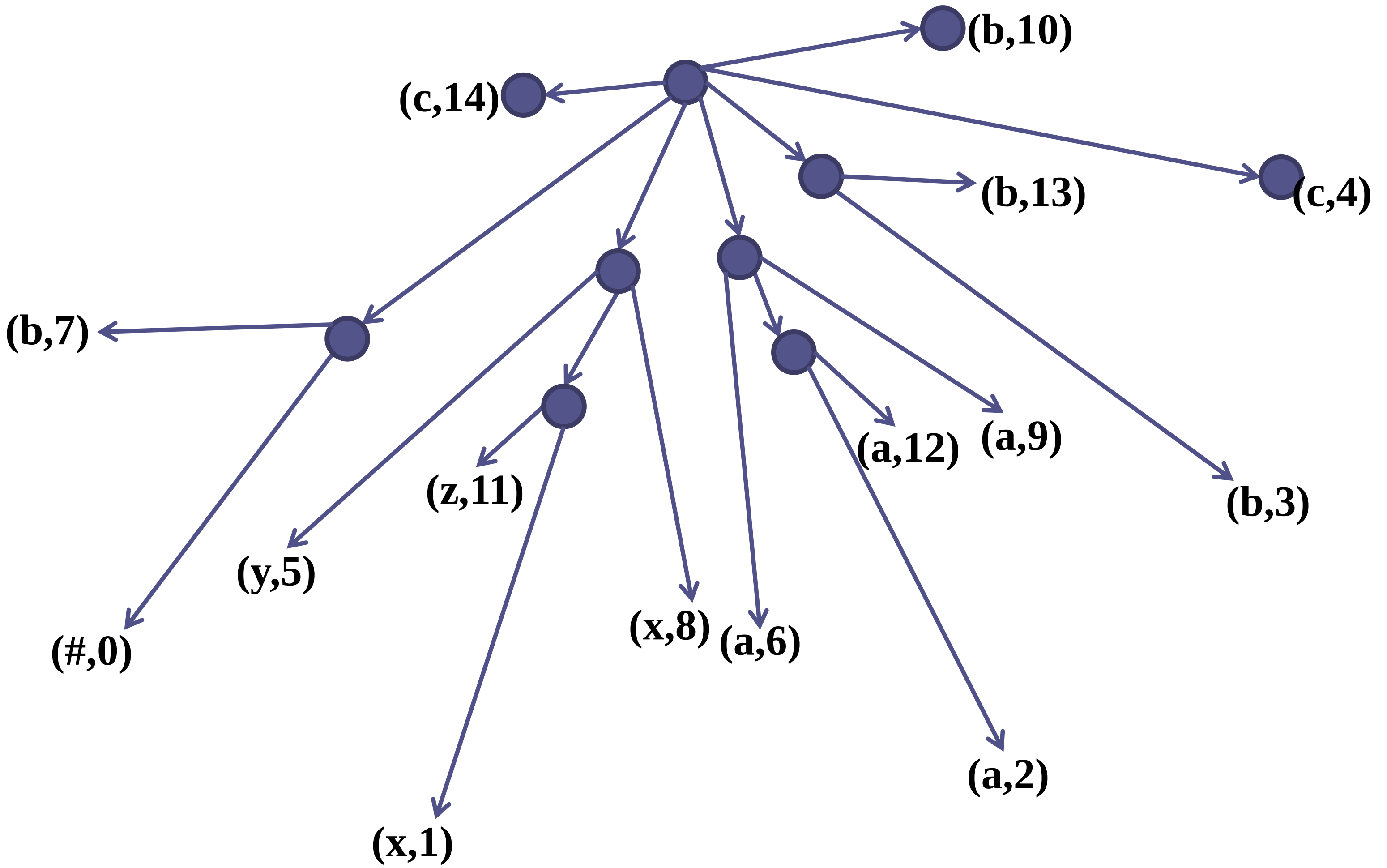
Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$



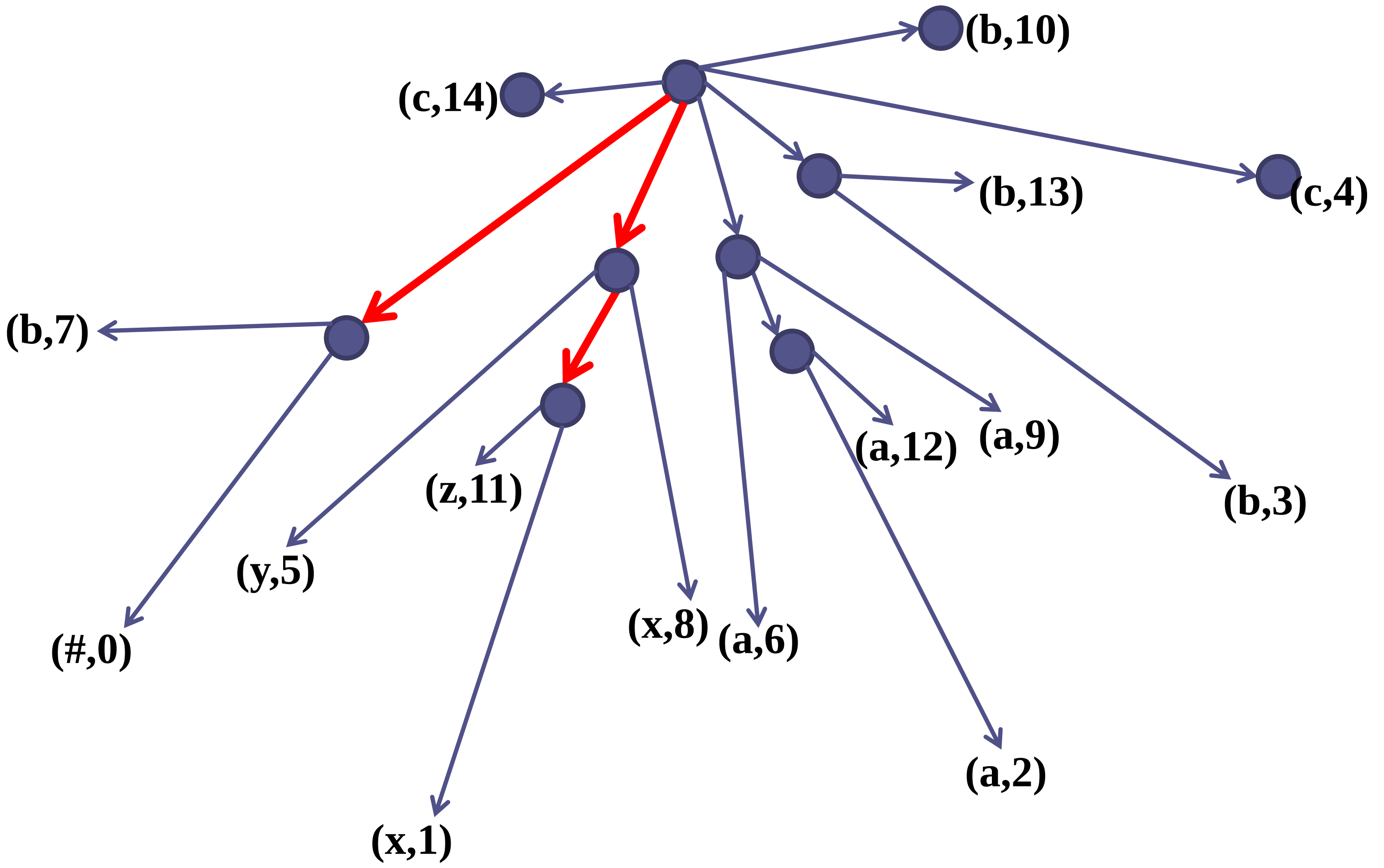
Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$



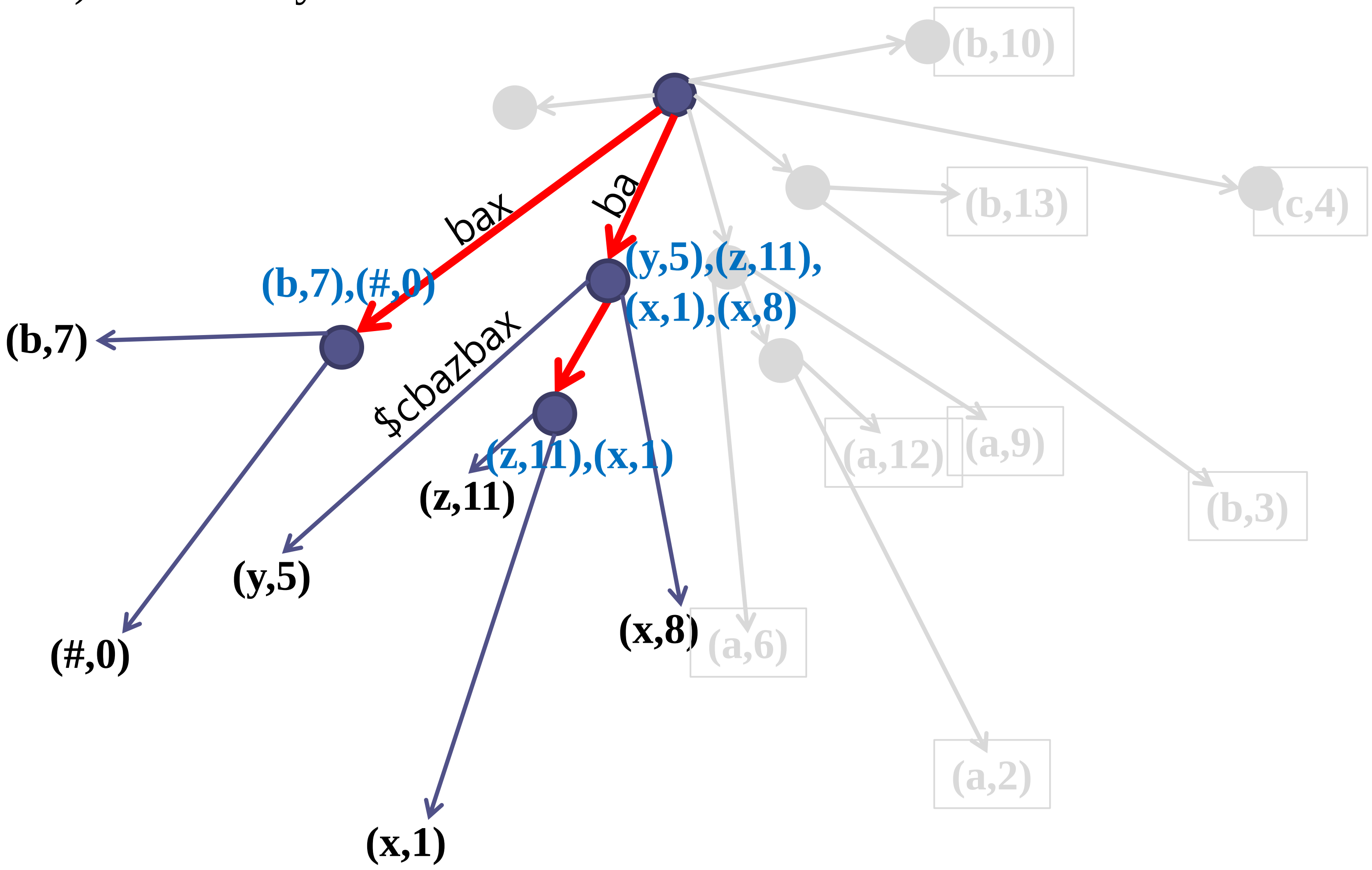
Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$



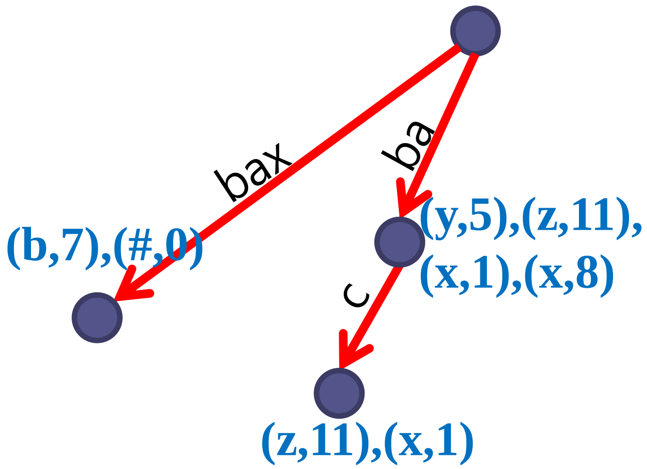
Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$



Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$



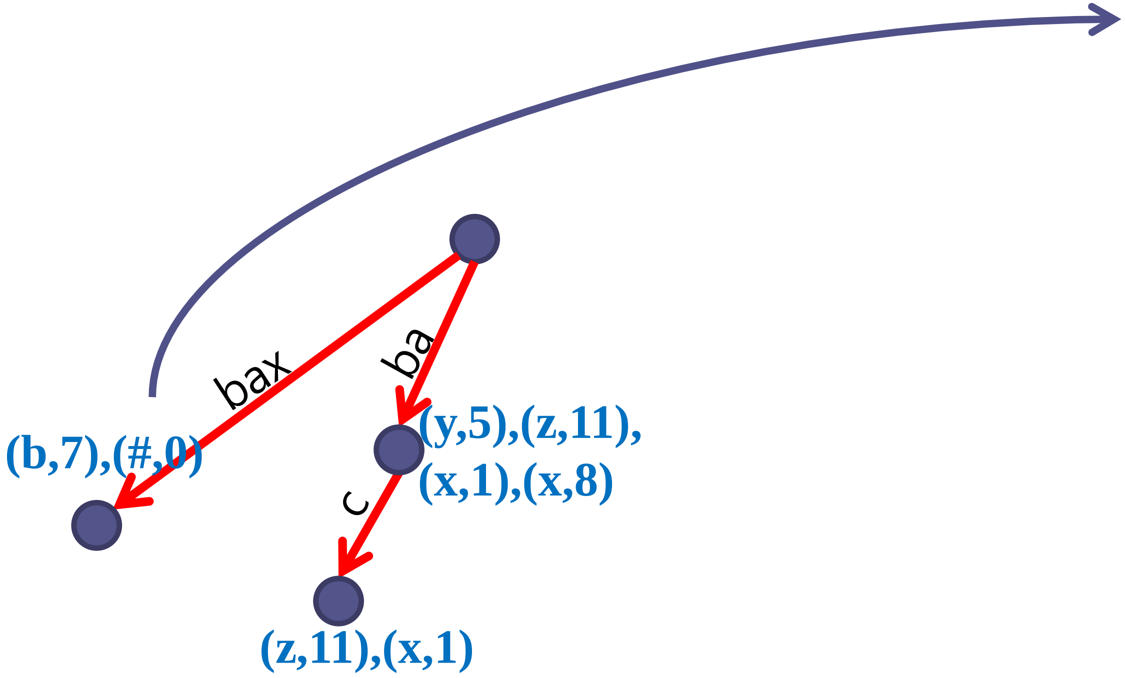
Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$

S = xabcyabxabzabc$

Maximal pair

(1, 8, ‘xab’)



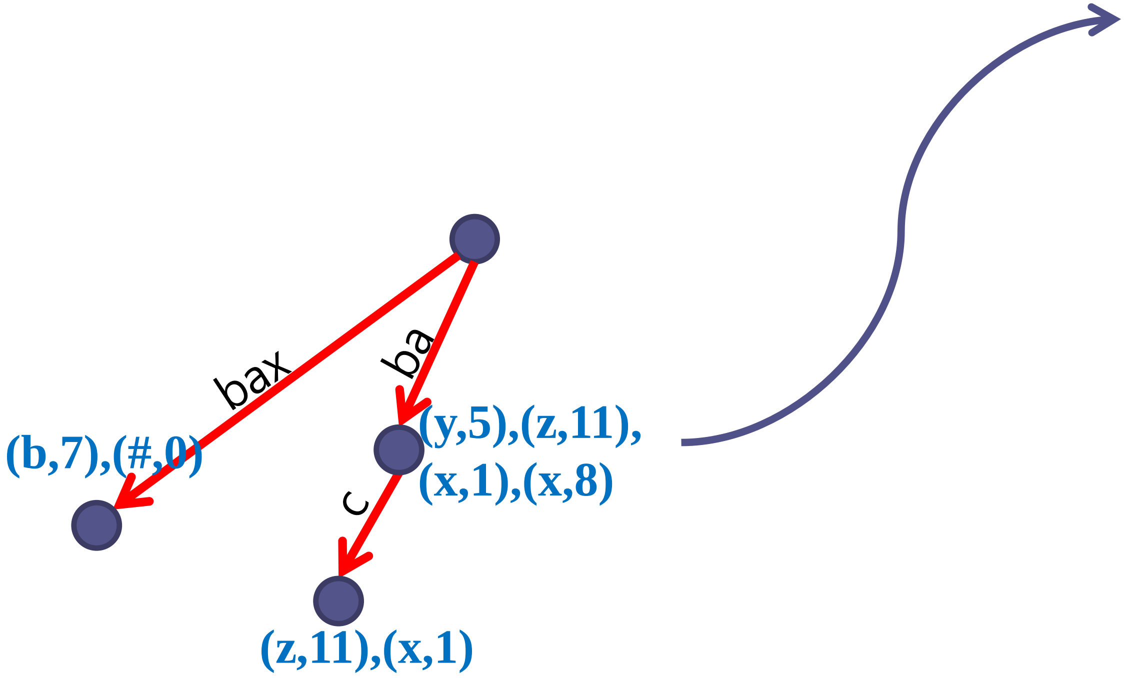
Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$

S = xabcyabxabzabc$

Maximal pair

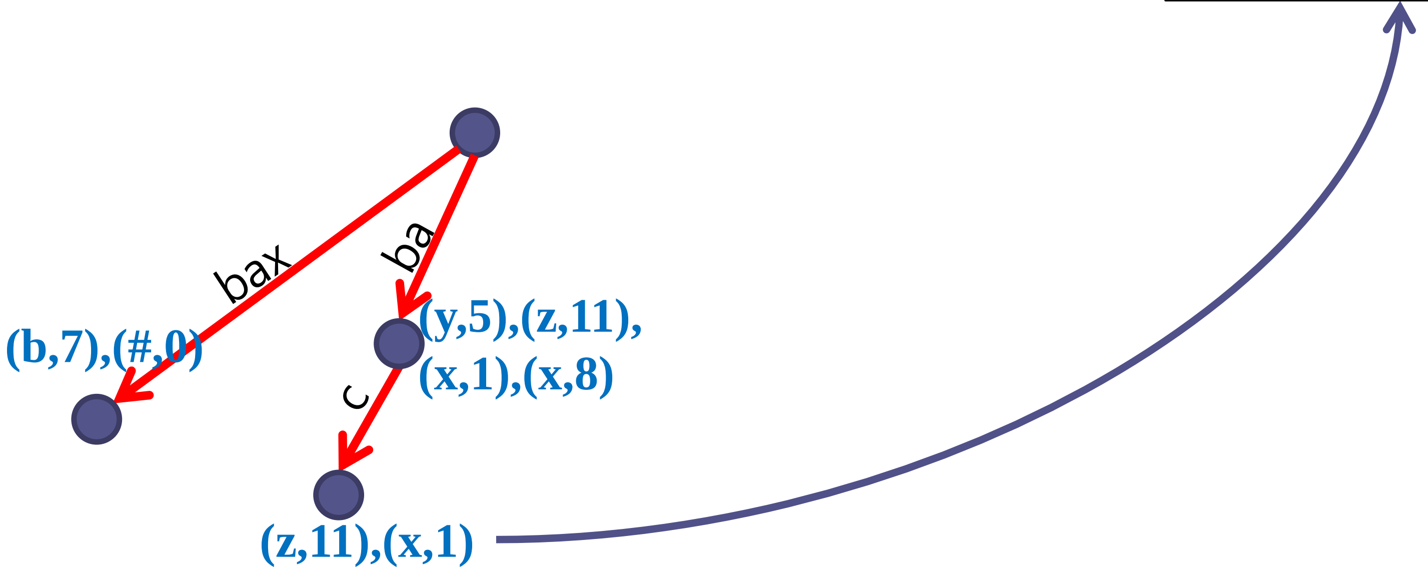
(2, 6, ‘ab’) (6, 9, ‘ab’)
(6, 12, ‘ab’) (9, 12, ‘ab’)



Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$

S = xabcyabxabzabc$



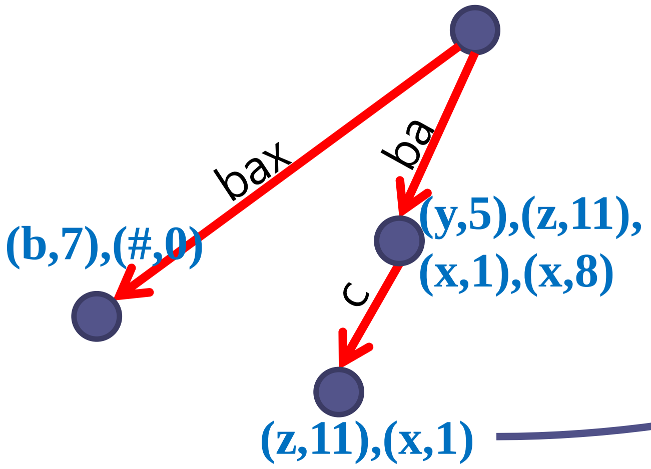
Maximal pair

(2, 12, 'abc')

Find all the maximal pairs in linear time

Ex) S = xabcyabxabzabc$

S = xabcyabxabzabc$



Maximal pair

(2, 12, ‘abc’)

Time complexity

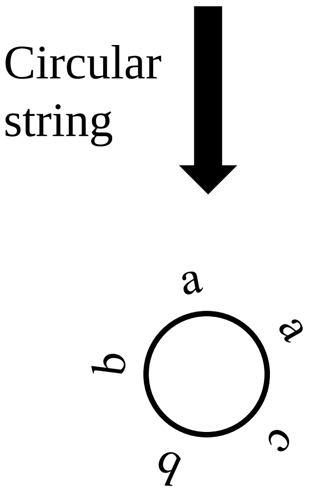
O (n + k) time

k : number of maximal pair

Circular string linearization

- **Circular string**

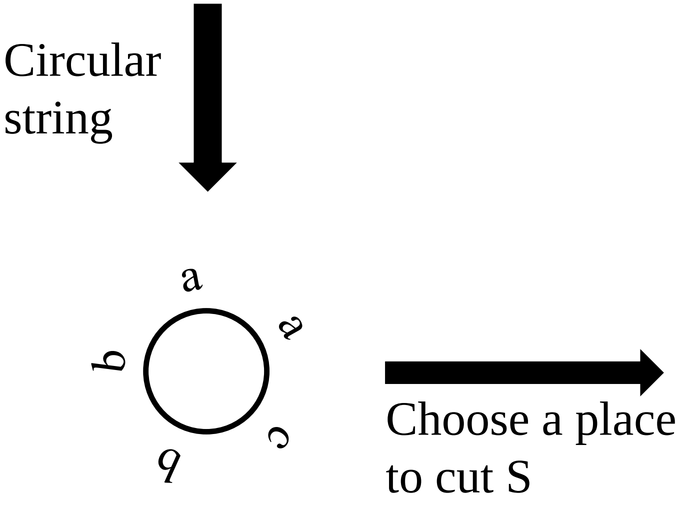
  - Ex) S = b a a c b



Circular string

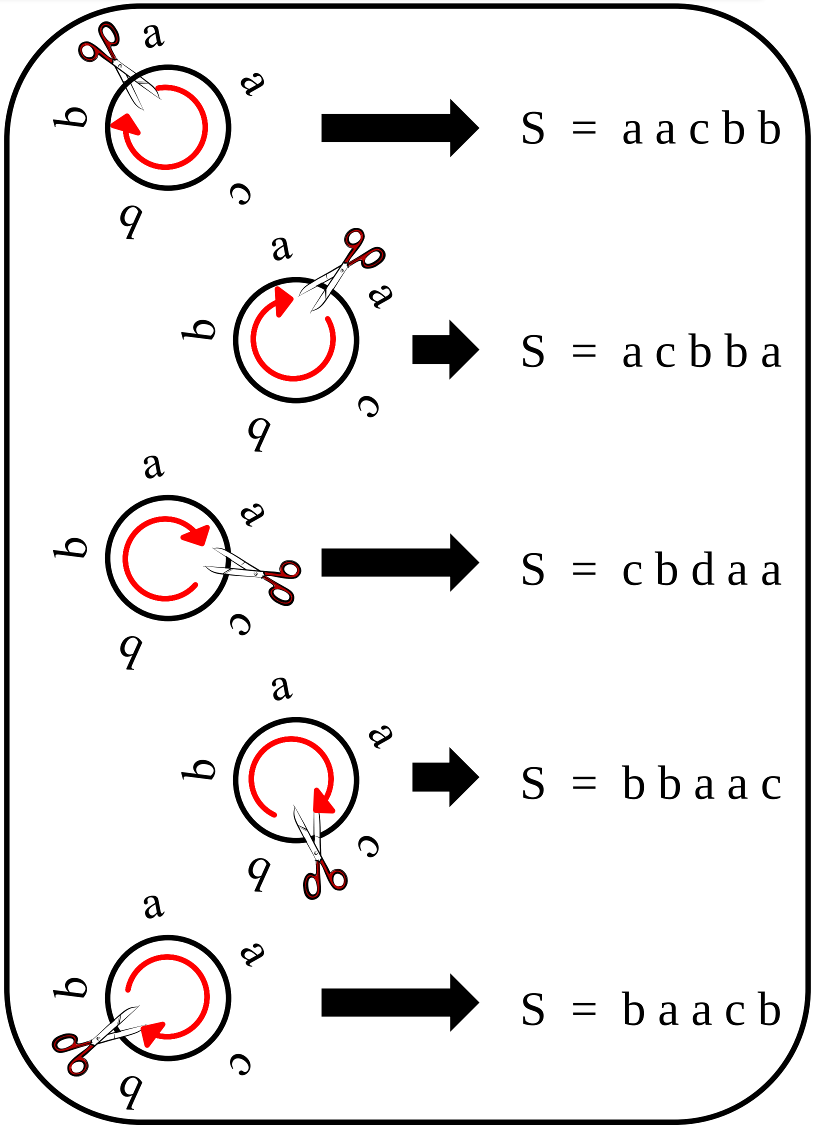
Circular string linearization

- **Circular string**
  - Ex) S = b a a c b



Circular string

Choose a place to cut S



S = a a c b b

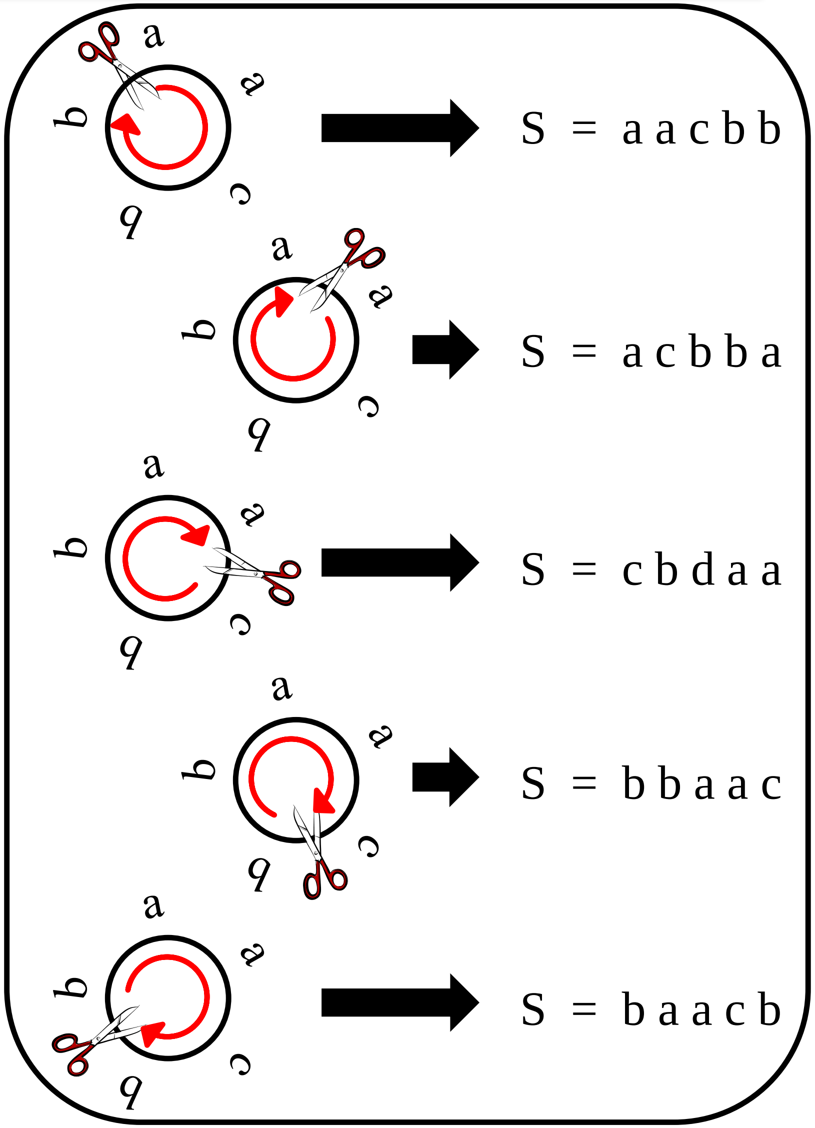
S = a c b b a

S = c b d a a

S = b b a a c

S = b a a c b

Circular string linearization



S = a a c b b

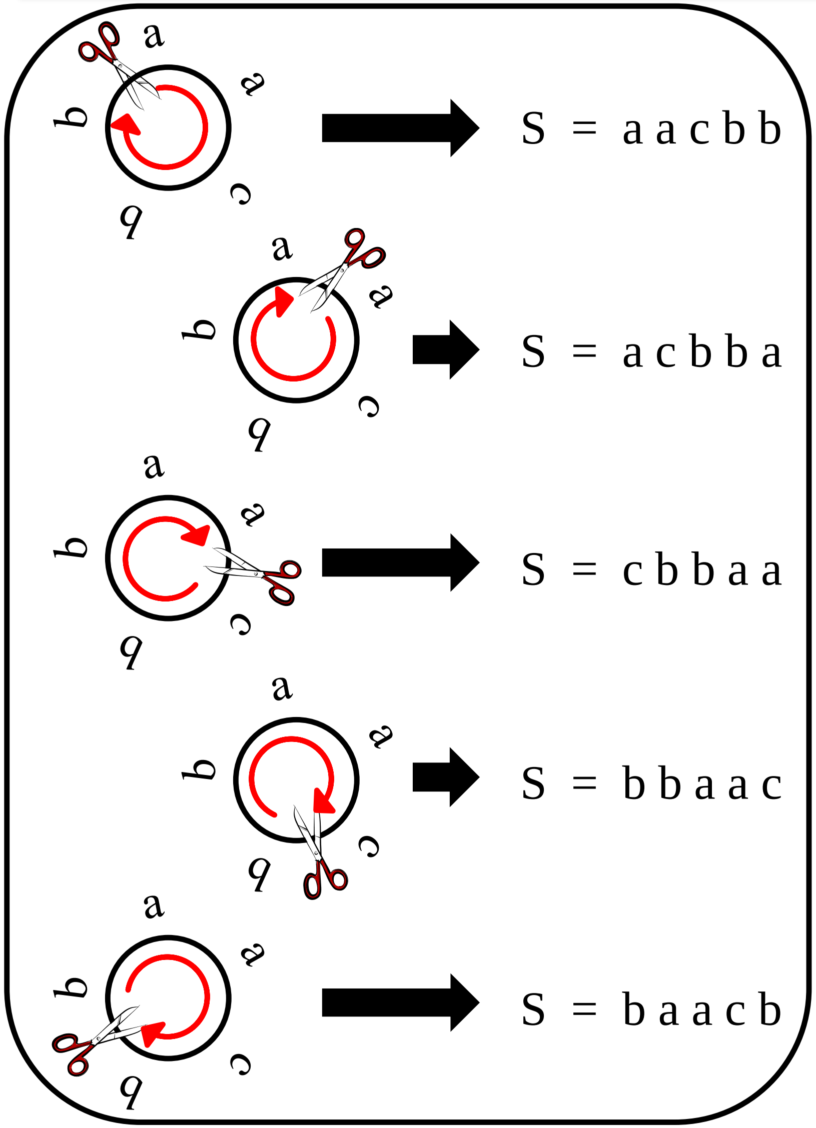
S = a c b b a

S = c b d a a

S = b b a a c

S = b a a c b

Circular string linearization



S = a a c b b

S = a c b b a

S = c b b a a

S = b b a a c

S = b a a c b

We want to Find the lexically smallest of all the *n* possible linear strings

ex) S = b a a c b

1) a a c b b

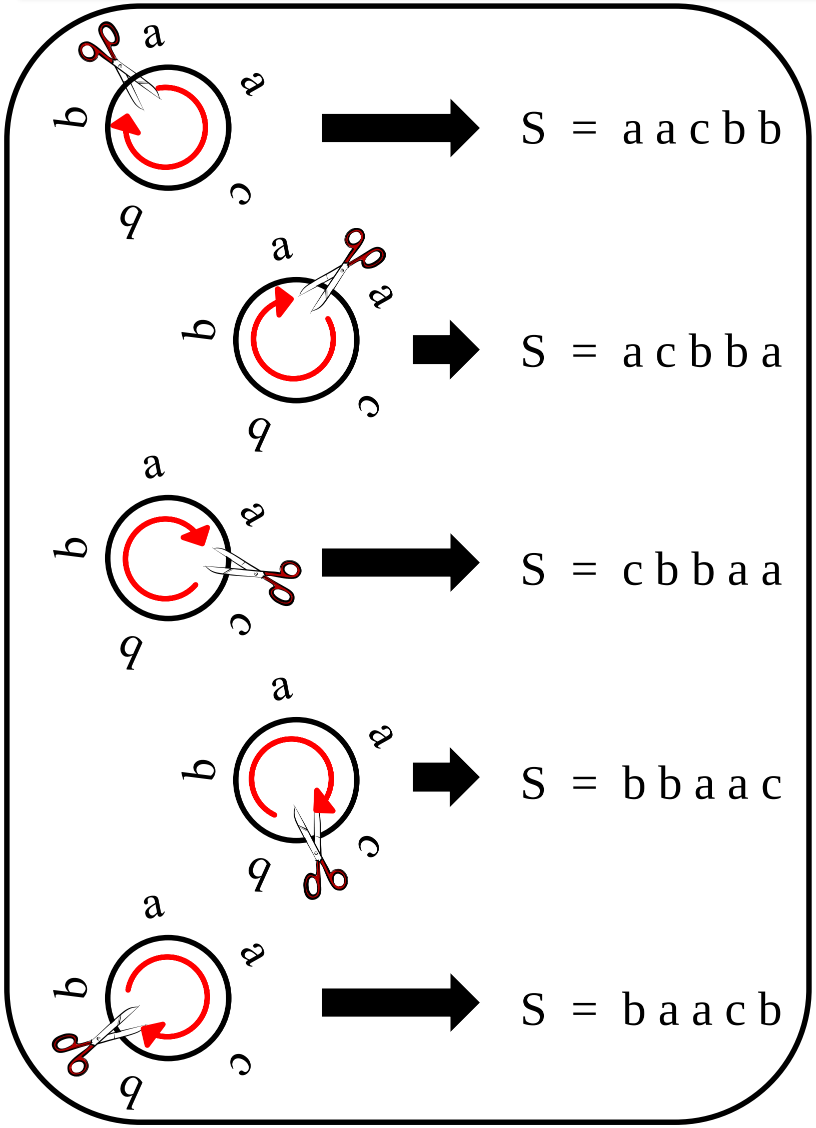
2) a c b b a

3) b a a c b

4) b b a a c

5) c b b a a

Circular string linearization



S = a a c b b

S = a c b b a

S = c b b a a

S = b b a a c

S = b a a c b

We want to Find the lexically smallest of all the *n* possible linear strings

ex) S = b a a c b

1) **a a c b b** → Lexically (dictionary order) smallest string

2) a c b b a

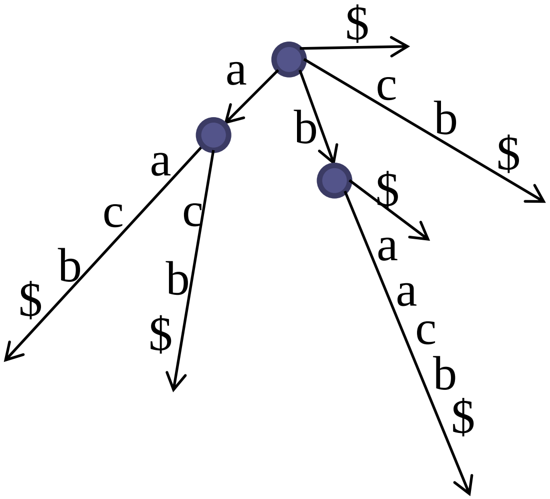
3) b a a c b

4) b b a a c

5) c b b a a

Circular string linearization

Ex) S = b a a c b $



1) a a c b b

2) a c b b a

3) b a a c b

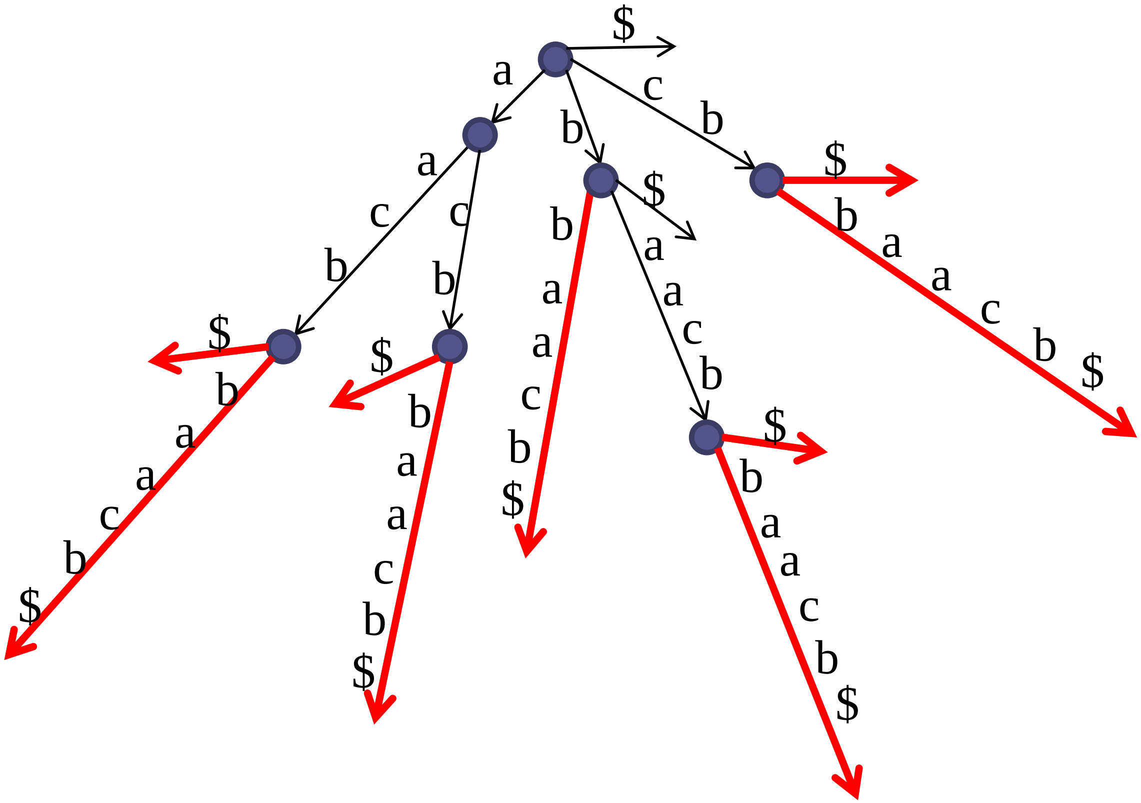
4) b b a a c

5) c b b a a

Can’t find these strings by suffix tree of *S*.

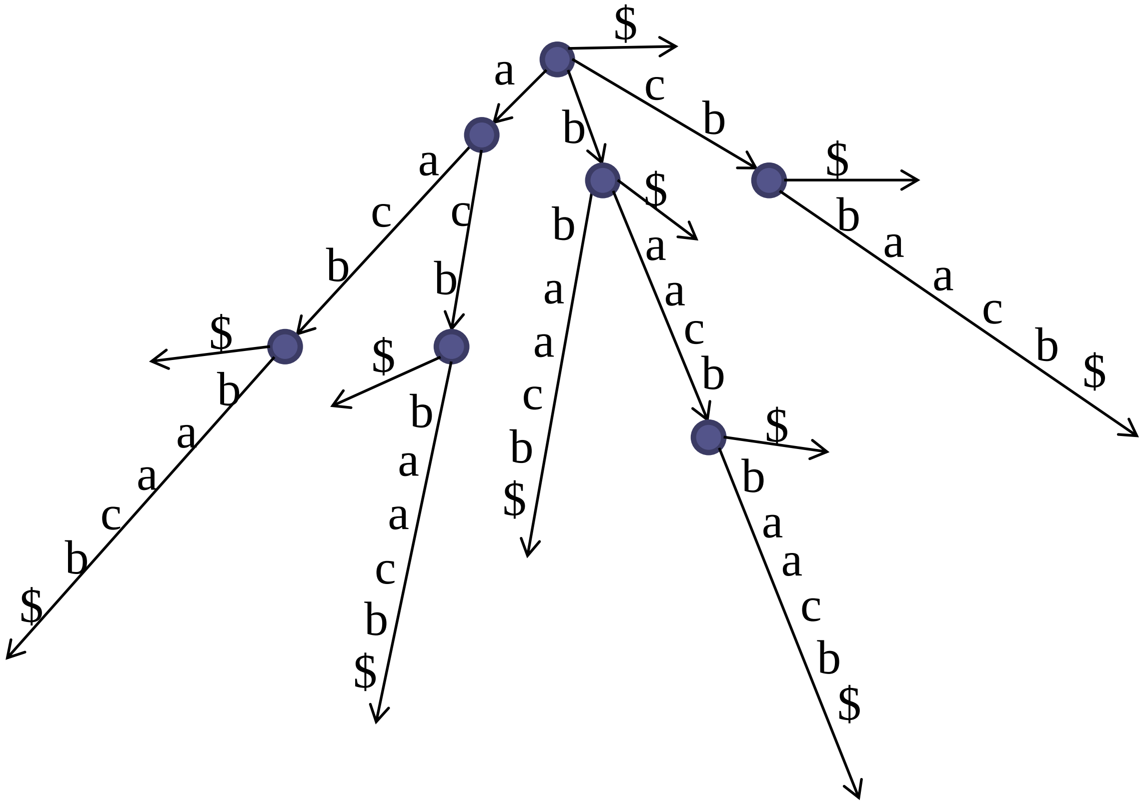
Circular string linearization

Ex) S' = b a a c b b a a c b $



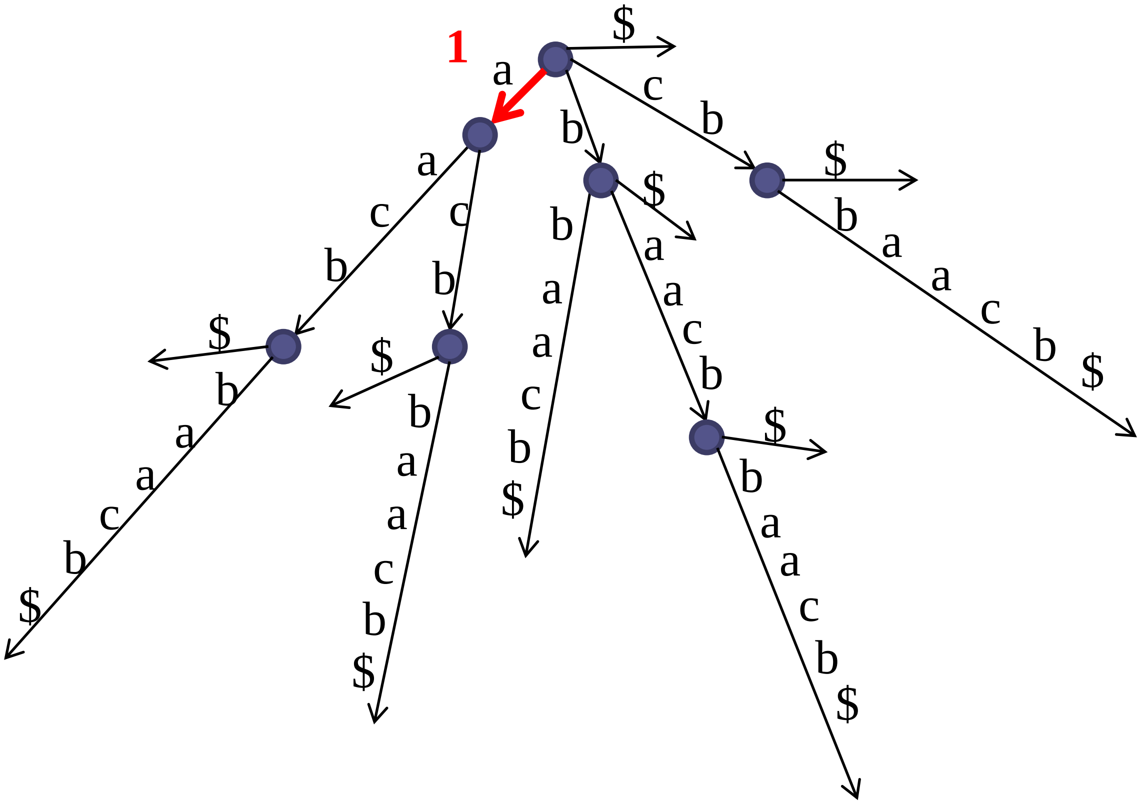
Circular string linearization

Ex) S' = b a a c b b a a c b $



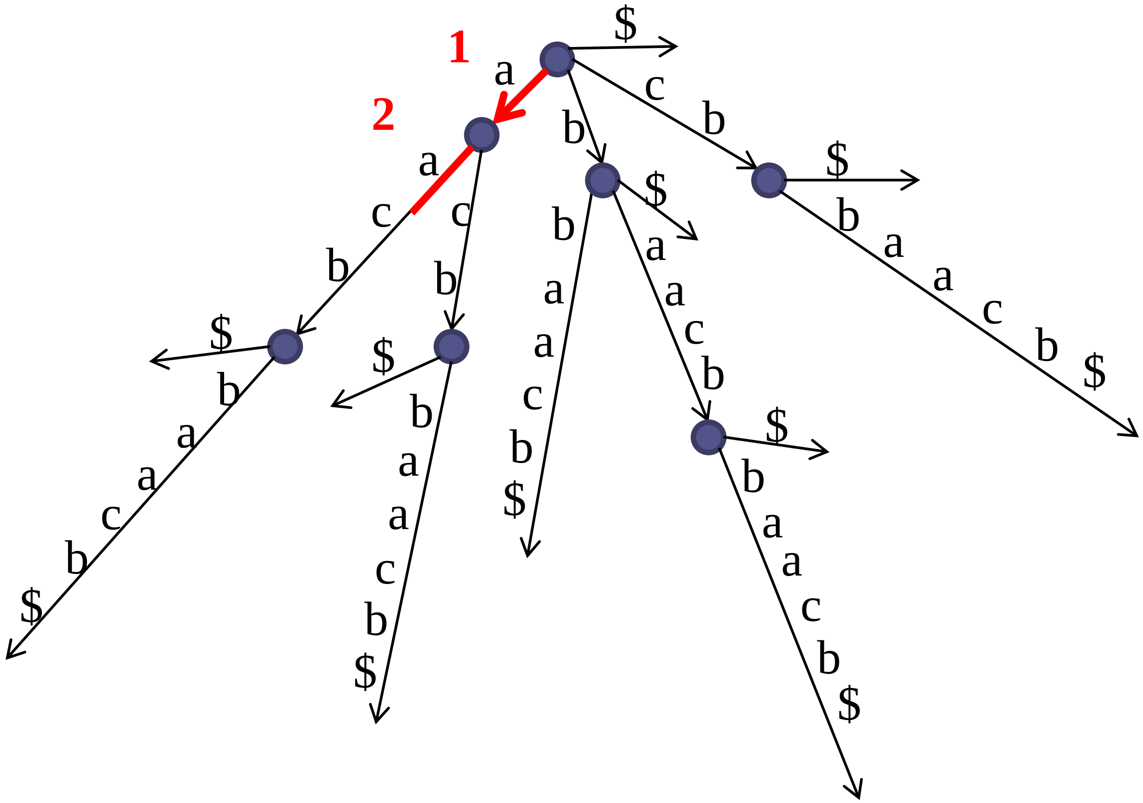
Circular string linearization

Ex) S' = b a a c b b a a c b $



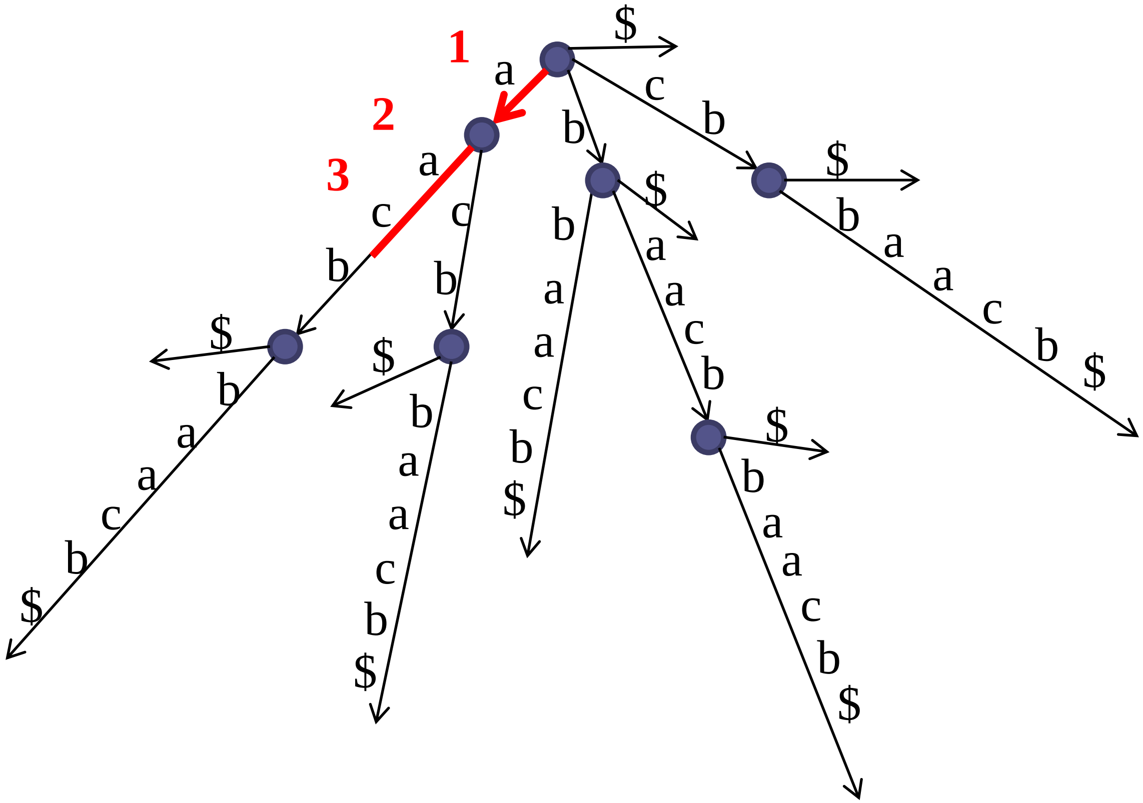
Circular string linearization

Ex) S' = b a a c b b a a c b $



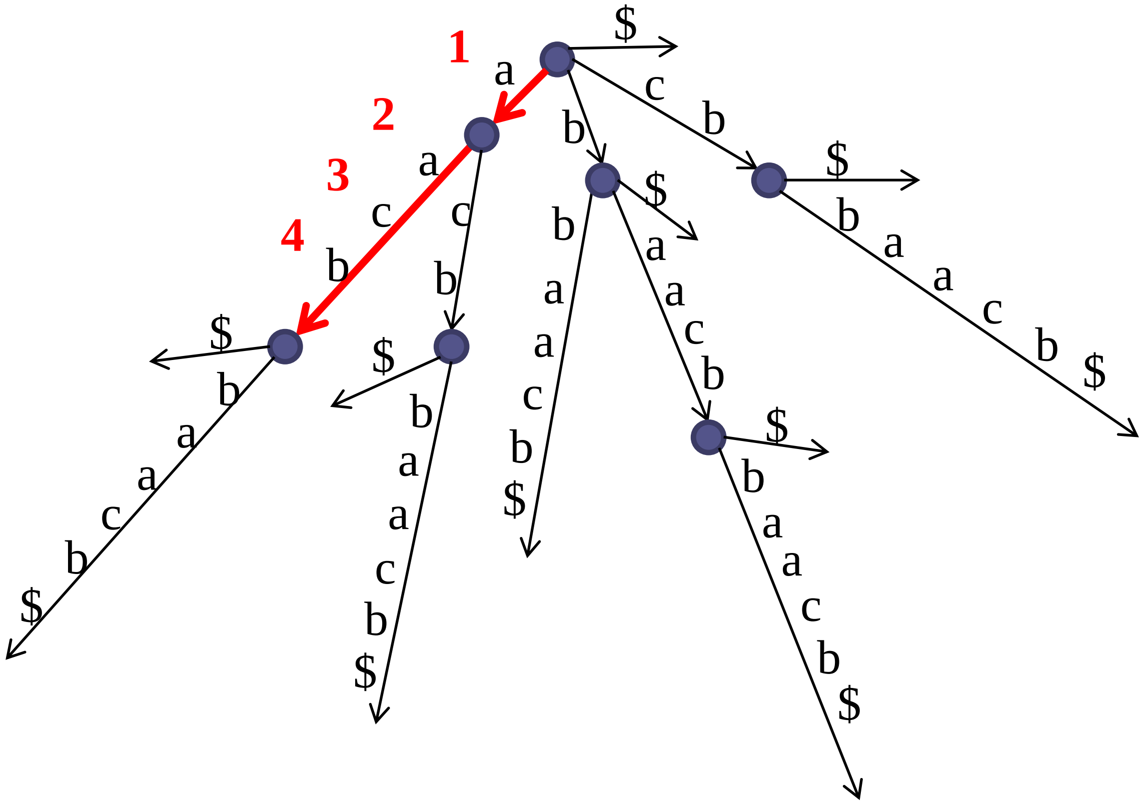
Circular string linearization

Ex) S' = b a a c b b a a c b $



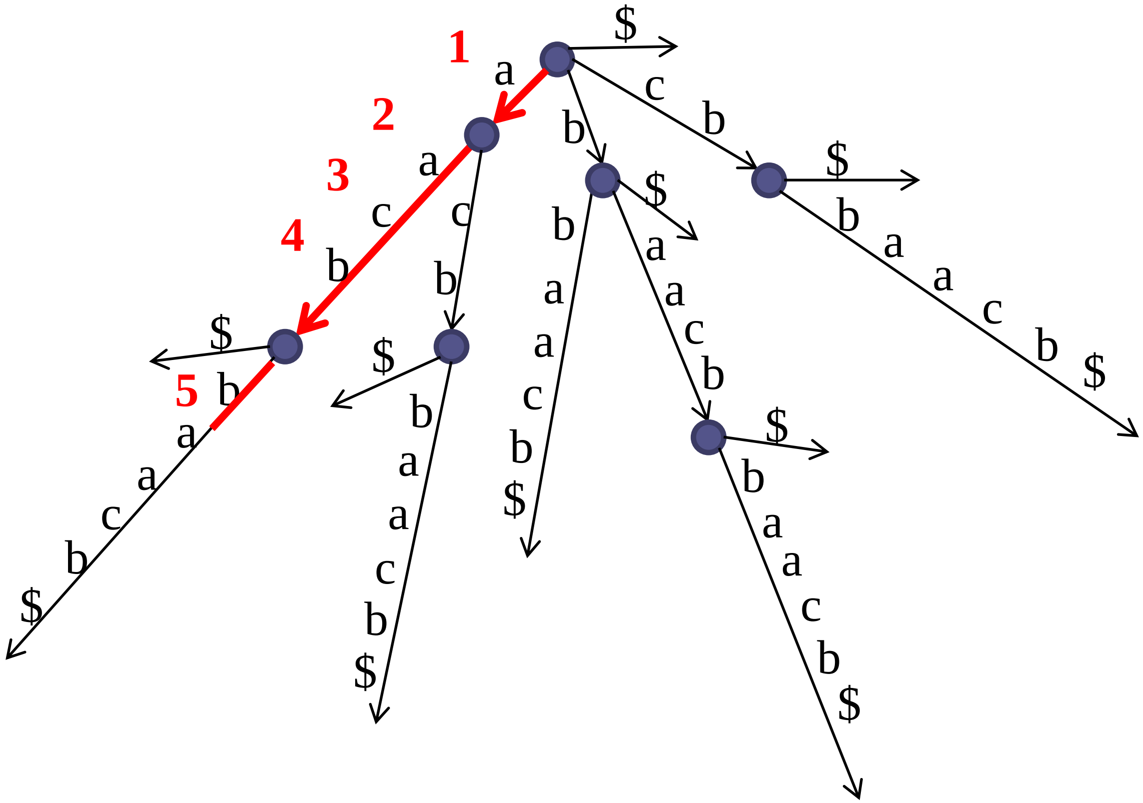
Circular string linearization

Ex) S' = b a a c b b a a c b $



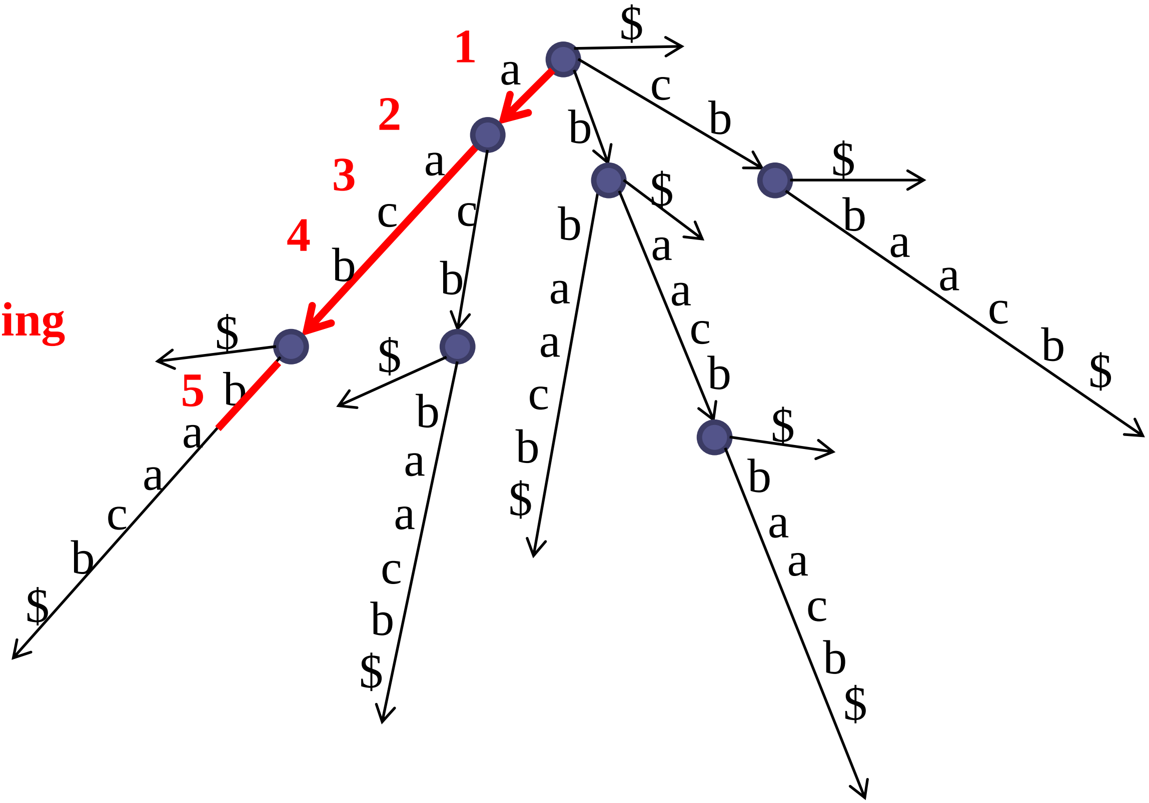
Circular string linearization

Ex) S' = b a a c b b a a c b $



Circular string linearization

Ex) S' = b a a c b b a a c b $



Lexically smallest string
➔ a a c b b