SUFFIX TREES

TEXT (T) : SHESELLSSEASHELLSBYTHESEASHORE

PATTERN (P) : ASH

GOAL: see if P is in T

↳ count/find all matches

1) as fast as possible, after pre-processing T

2) but also minimize pre-processing time/space

(1) $\Omega(P)$ → multiple searches : $\Omega(\Sigma(P_i))$

(2) $\Omega(T)$

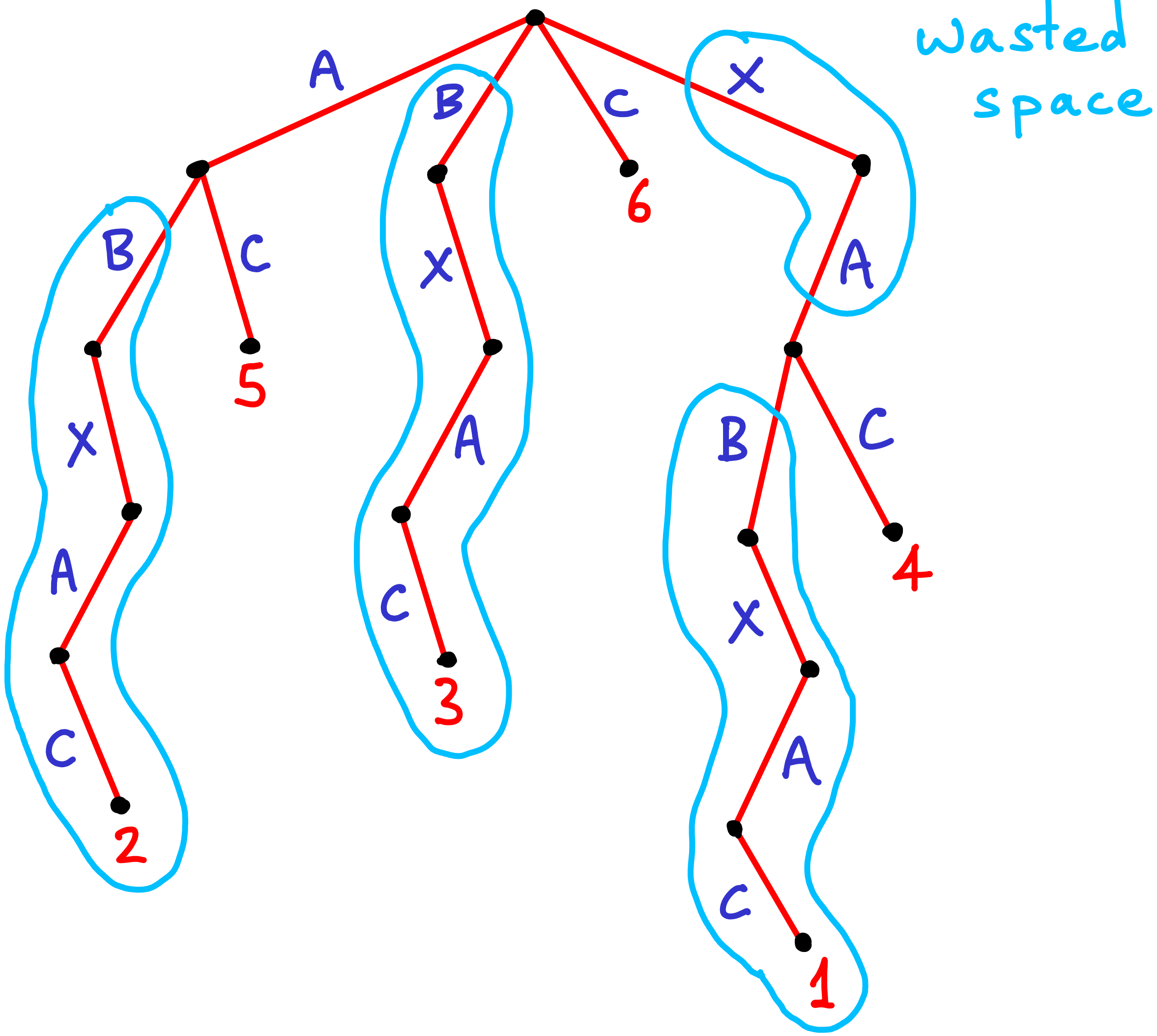
alphabet : A, B, C, X

	1	2	3	4	5	6
Text :	X	A	B	X	A	C

suffixes:

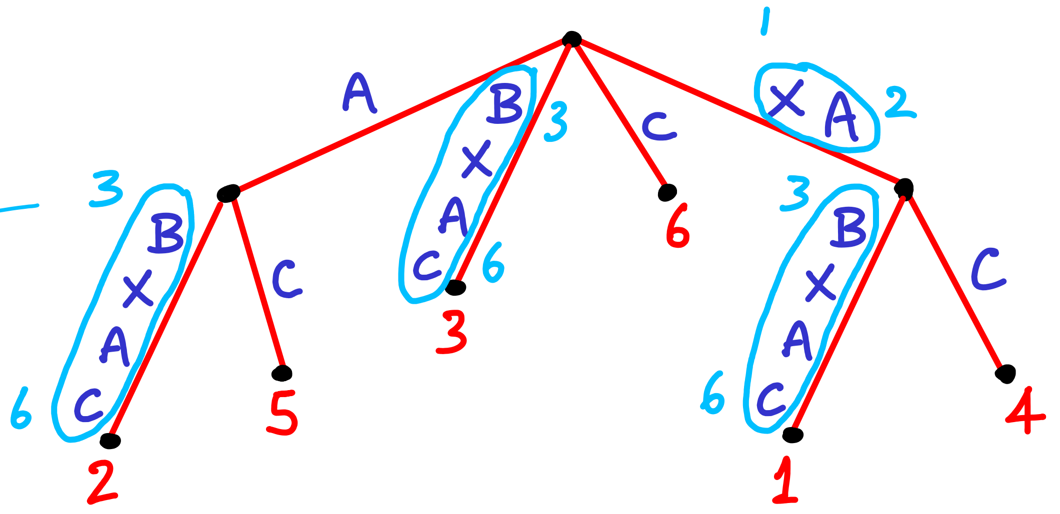
1	XABXAC
2	ABXAC
3	BXAC
4	XAC
5	AC
|T| ~ "T" : 6	C

Search for pattern P:
it starts at some index i,
so is in suffix starting at i.



	1	2	3	4	5	6
Text :	X	A	B	X	A	C

alphabet : A, B, C, X



using start/end indices,

size(tree) = $O(T)$

assuming |alphabet| = const.

search time: $O(P)$

(to find if ∃ match)

(augment: subtree sizes for #matches)

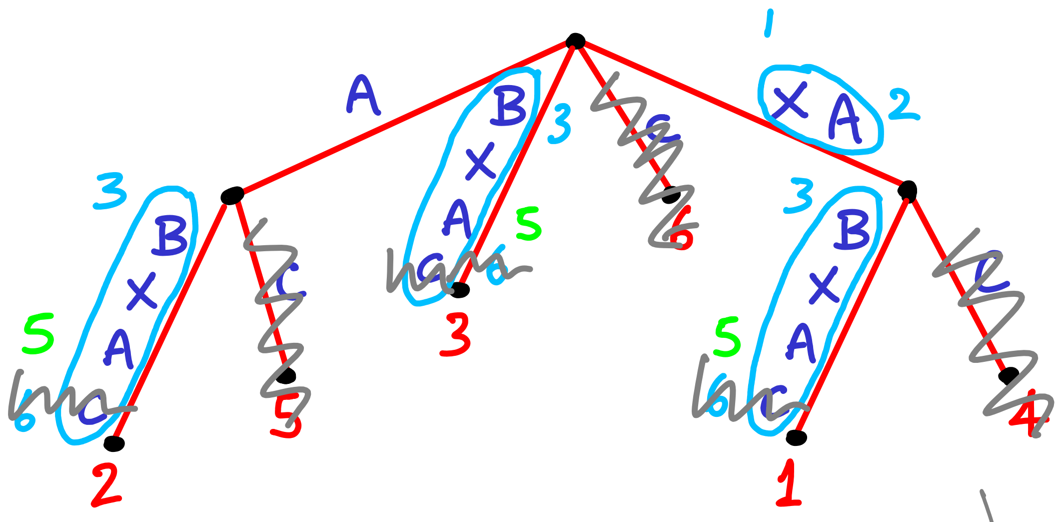
- every suffix is in tree
- # leaves = T ⇒ $\underbrace{\text{size}}_{\#\text{nodes}}$(tree) = $O(T)$

(every internal node has ≥ 2 children)

new

Text : XABXA~~C~~ (positions 1 2 3 4 5 ~~6~~)

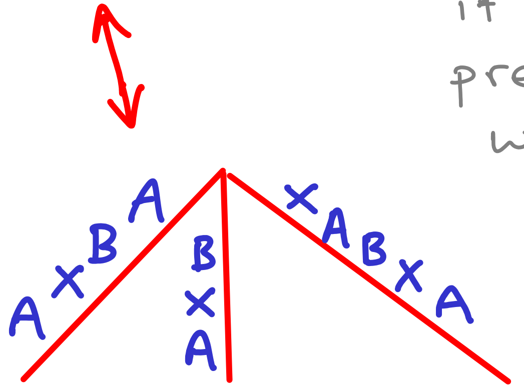
alphabet : A, B, ~~C~~, X



The suffixes are still there,
but we can't enumerate matches.

Also we can't answer other queries
e.g. "find words starting with AB..."

if we follow previous rules we get



new

Text : 1 2 3 4 5 ~~6~~

XABXA~~C~~

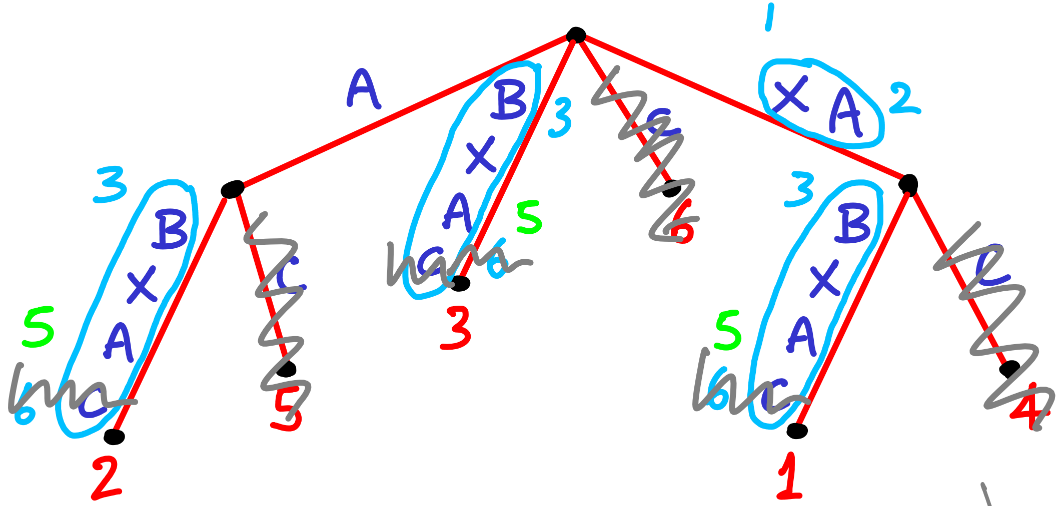
solution

1 2 3 4 5 6

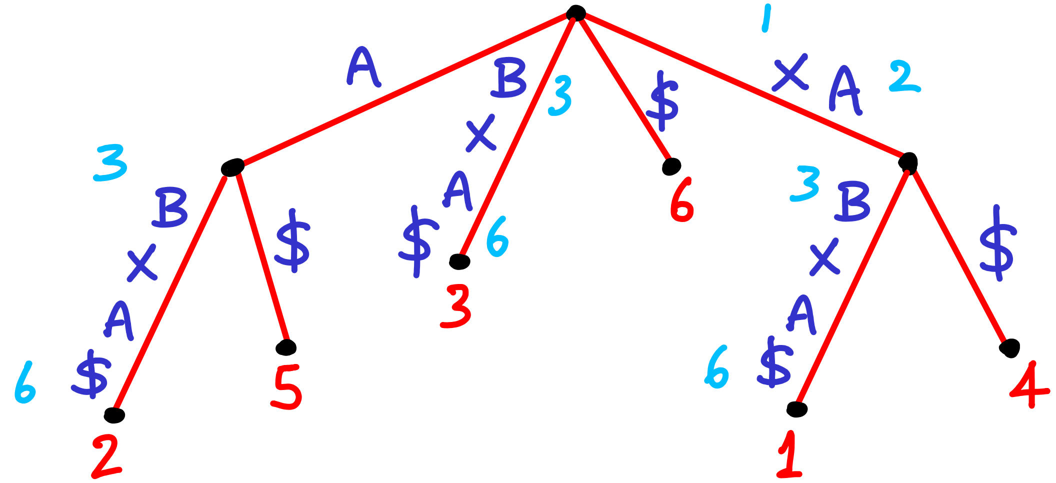
XABXA$

alphabet : A, B, X, $

alphabet : A, B, ~~C~~, X

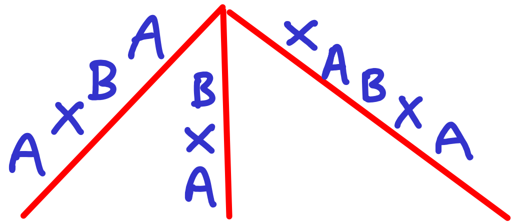


3



6

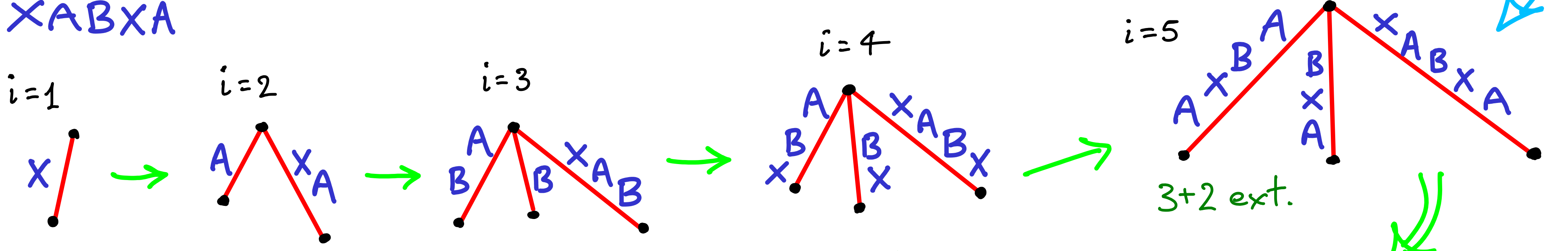
if we follow previous rules we get



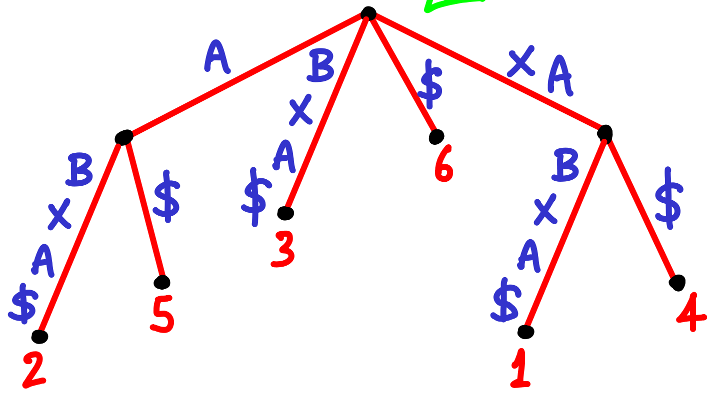
UKKONEN'S ALGORITHM to build a suffix tree in $O(T)$ time

- iteratively build suffix tree: iteration i builds tree on $T[1 \ldots i]$
  - note: we actually build the "implicit" tree (& make it proper at last iteration)

XABXA



1 ext. 2 ext. 3 ext. 3+1 ext. 3+2 ext.

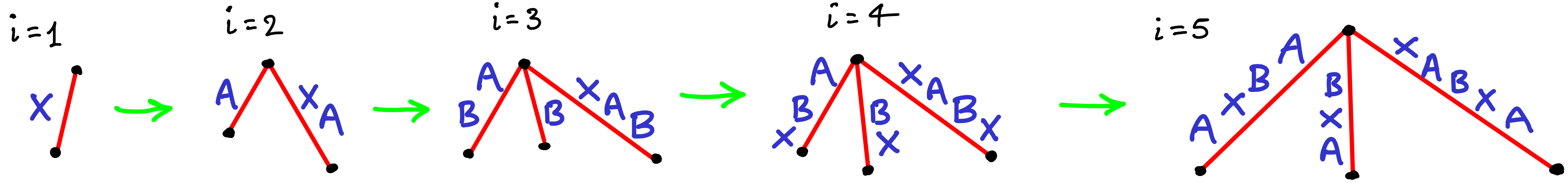


At iteration i, we make i "extensions" $\Big\}$ $\sum_{i=1}^{|T|} i$ extensions

we must reduce this

XABXA

At iteration i, we make i "extensions"

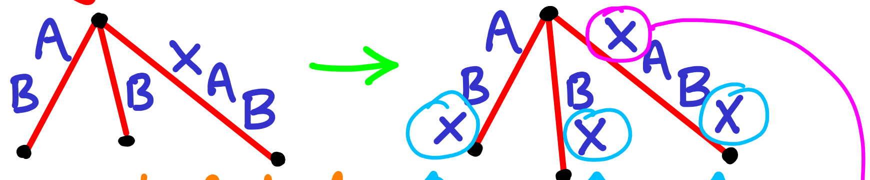


Each extension handles one of the existing suffixes (including empty suffix)

XABX → XABXA
ABX → ABXA
BX → BXA
X → XA
∅ → A

3 cases depending on how current suffix "ends"

1) at a leaf



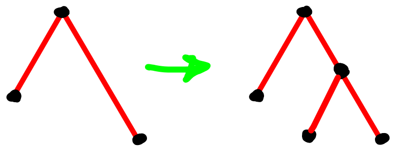
↳ extend current label

2) not at a leaf, but new character is already there

↳ do nothing

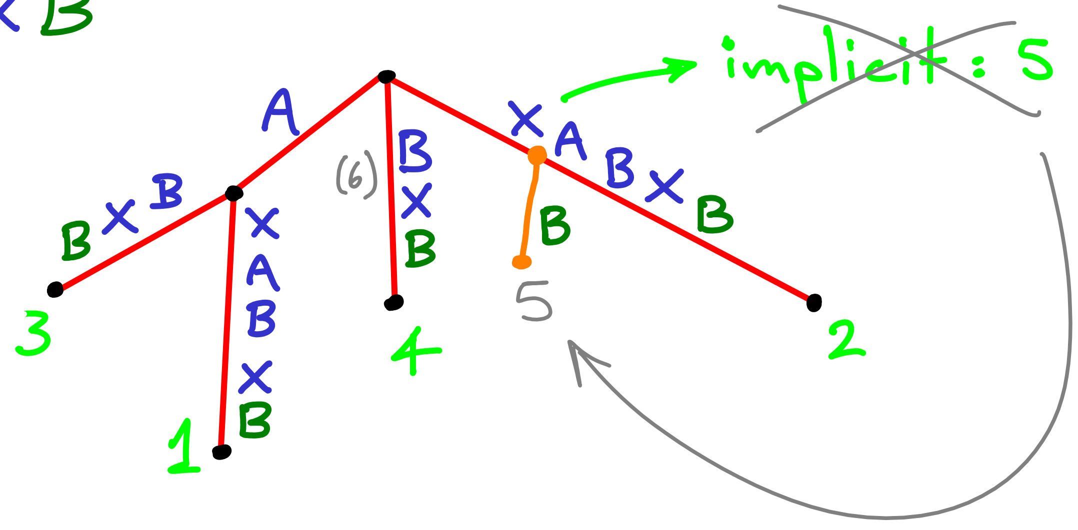
3) not at a leaf, new char. not there

↳ make new edge w/ new character



Another example : AXABXB

suffix		
	1	AXABXB
	2	XABXB
	3	ABXB
	4	BXB
	5	XB
	6	B



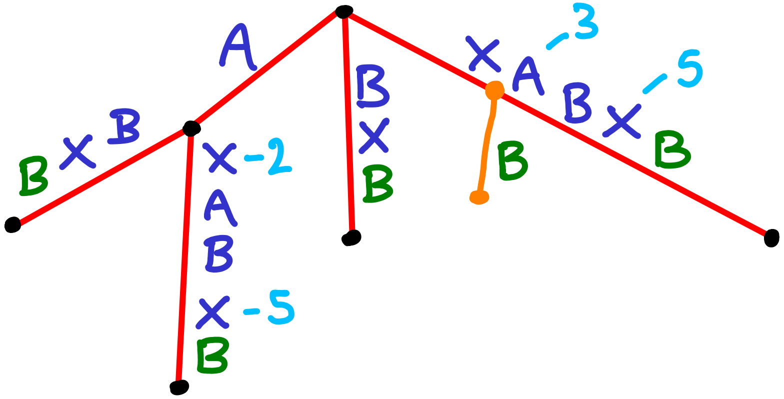
Let T[6] = B
(iteration 6)

extension	1	2	3	4	5	6
case	1	1	1	1	3	2

1) at a leaf → extend current label
2) not at a leaf, but new character is already there → do nothing
3) not at a leaf, new char. not there → make new edge w/ new character

At iteration i, we make i "extensions"

For each extension: $\underbrace{\text{3 cases}}_{\text{each looks easy \& quick}}$ depending on $\underbrace{\text{how current suffix "ends"}}_{\text{how do we determine this?}}$



* Scanning whole suffix → $O(i^2)$ per iteration $\left(\sum_{j=1}^{i} i-j\right)$

* Using indexing doesn't help (yet)

AXABXB

1) at a leaf → extend current label

2) not at a leaf, but new character is already there → do nothing

3) not at a leaf, new char. not there → make new edge w/ new character

How do we determine where a current suffix "ends"? (quickly)

Suffix Links

Say we just found the end of a suffix, XABCDE

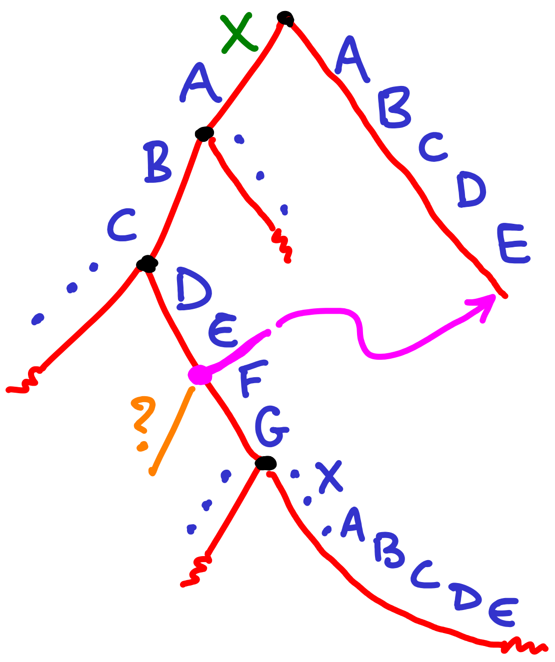
(maybe while extending it w/ F or something else)

↓

Then ABCDE is also a suffix, so it is also in the tree

and it is the next extension

so it would help to have a link from every $x\alpha$ to α ...except maybe there was no node at $x\alpha$

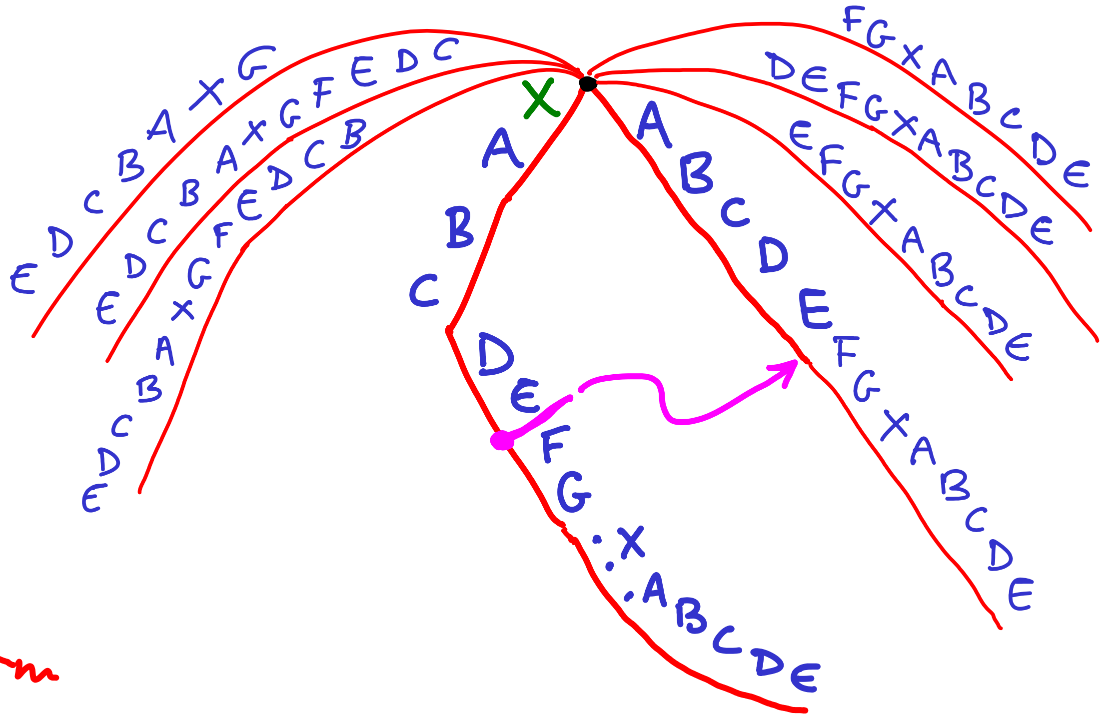


T[1...i-1] = XABCDEFGXABCDE $\boxed{?}$

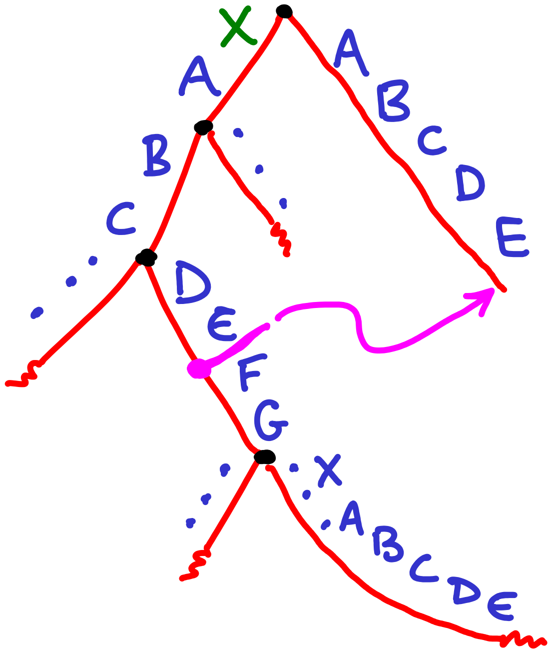
T[i]

T[1...i-1] = XABCDEFGXABCDE

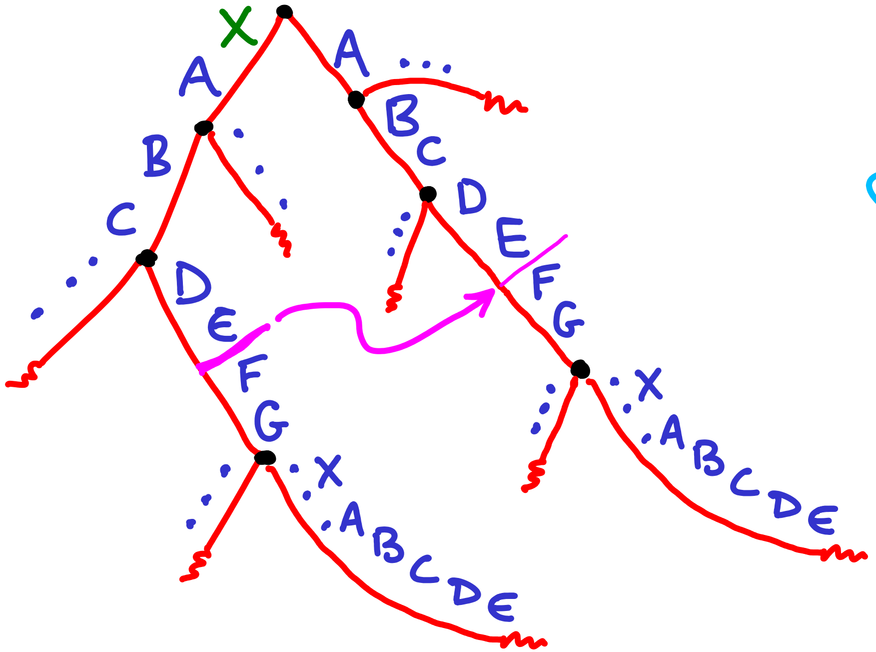
actual suffix tree



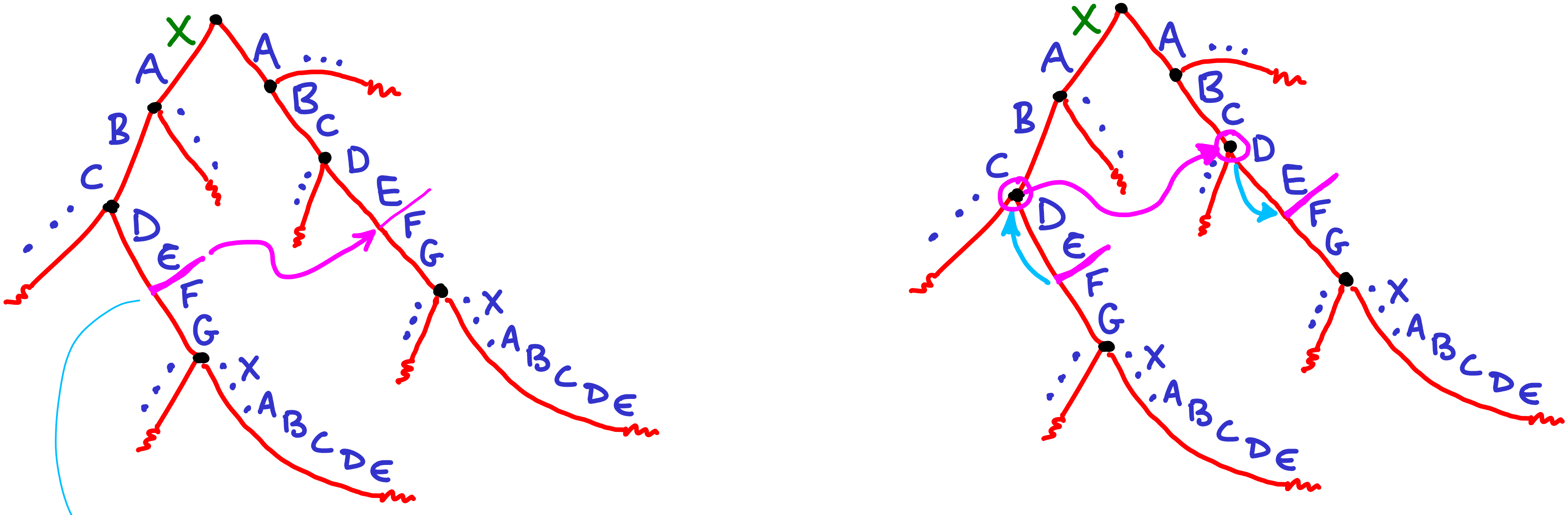
(now enjoy this unused slide)



for illustration



lots of repeated structure



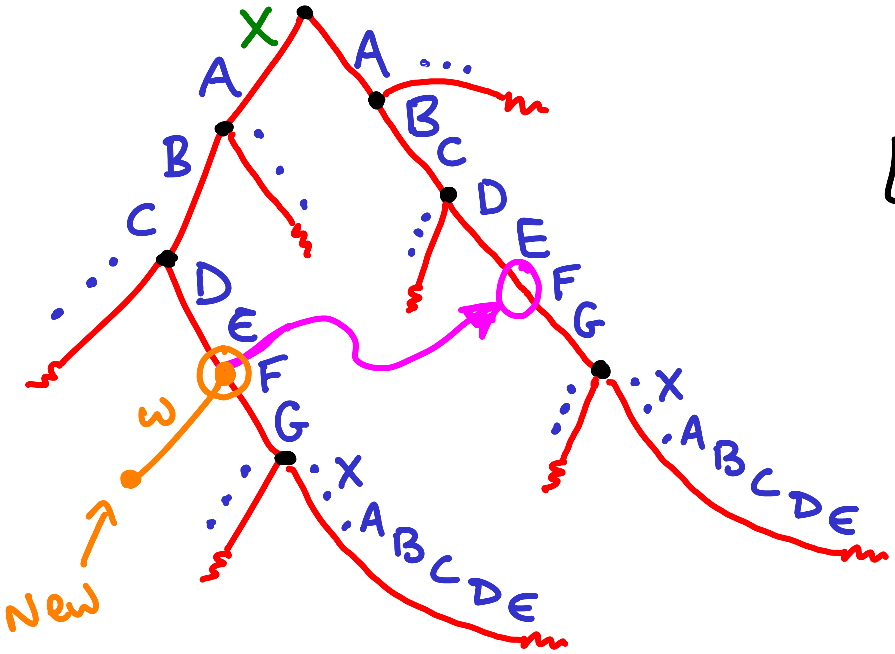
no node here

Instead move up until a node is found,
use a suffix link from node to node,
then "move down"

deal w/ this later

$O(1)$

// next issue:
how to make suffix links



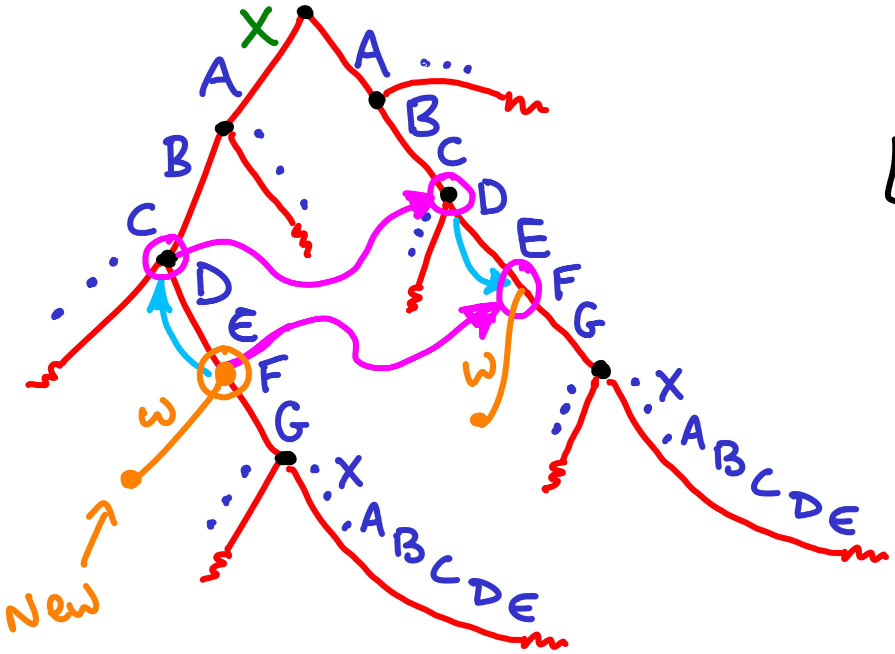
When do we create a new node?

↳ case where we process $x\alpha$ to extend ω and $\exists\ x\alpha z$ $(z \neq \omega)$

(end of $x\alpha$ is in mid-edge)

When do we create a suffix link from $x\alpha$ to α?

↳ First we need to know that a node exists at α.



When do we create a new node?

↳ case where we process $x\alpha$ to extend w and $\exists\ x\alpha z$ $(z \neq w)$

(end of $x\alpha$ is in mid-edge)

When do we create a suffix link from $x\alpha$ to α?

↳ First we need to know that a node exists at α.

↳ either it does already, or

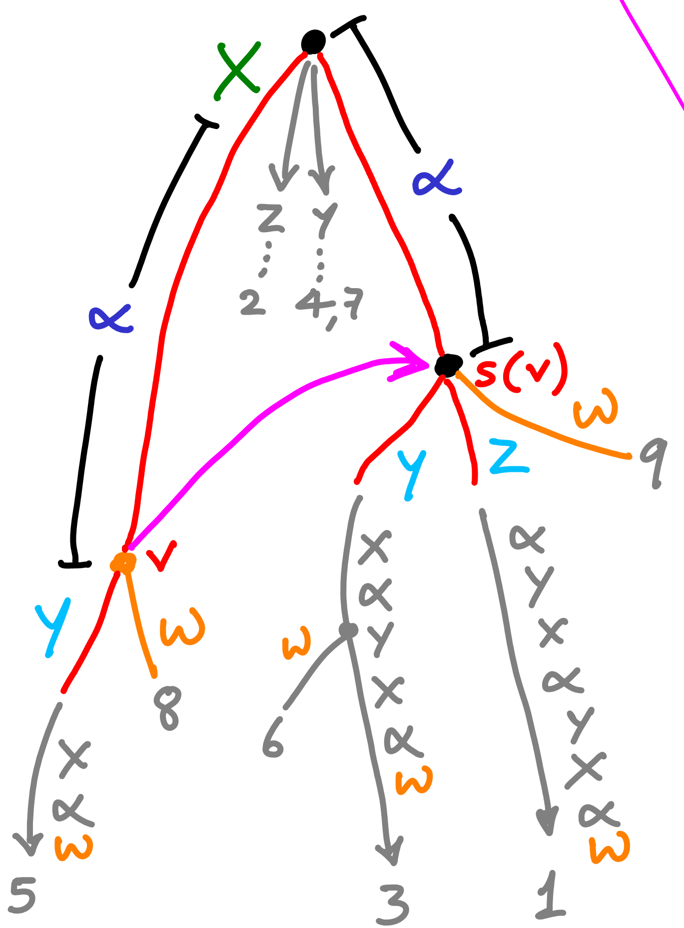
it will be created at next extension (we will process α)

So we can assume that all "old" nodes have outgoing suffix links

Example of suffix link target node existing before source node

$\text{Example of suffix link } \underbrace{\text{target node}}_{s(v)} \text{ existing before } \underbrace{\text{source node}}_{v}$

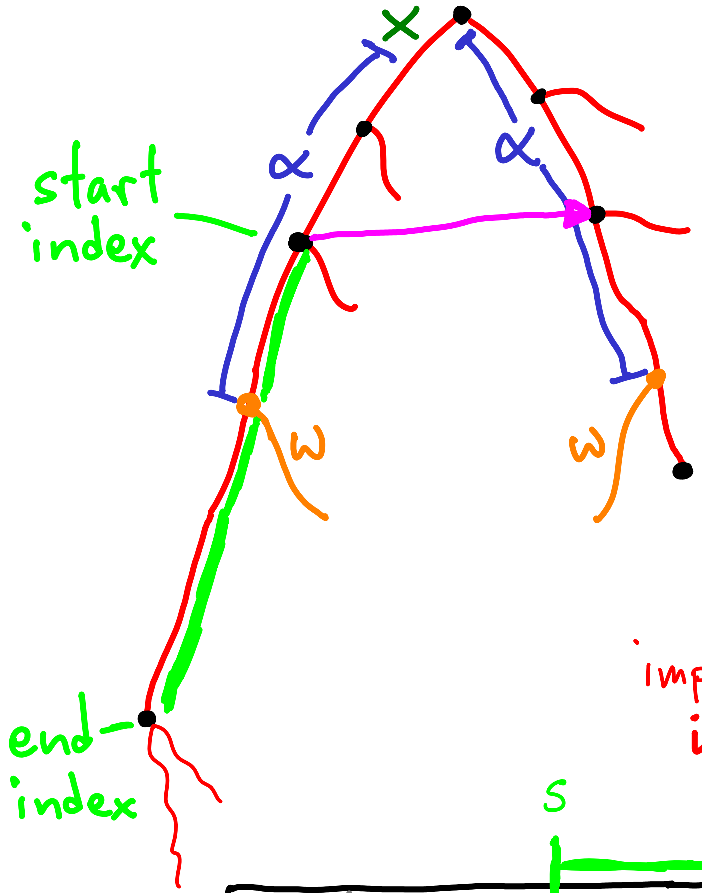
to be created at extension 9



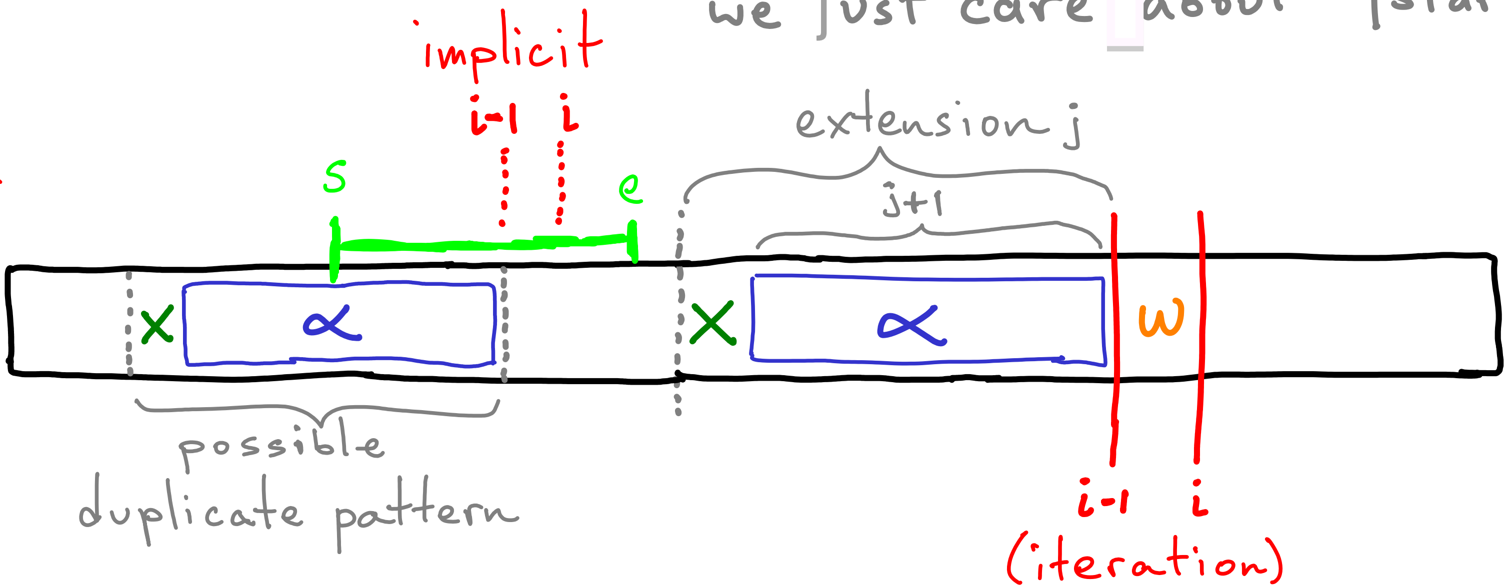
1	2	3	4	5	6	7	8	9	
α	Z	α	Y	X	α	Y	X	α	ω

currently processing extension 8

Technical details



For each edge we store the start/end indices.
While processing a suffix that is the prefix of a larger suffix we would be using indices implicitly (to find actual chars from T). Besides that, we just care about |start-end| = length



WHAT MATTERS:

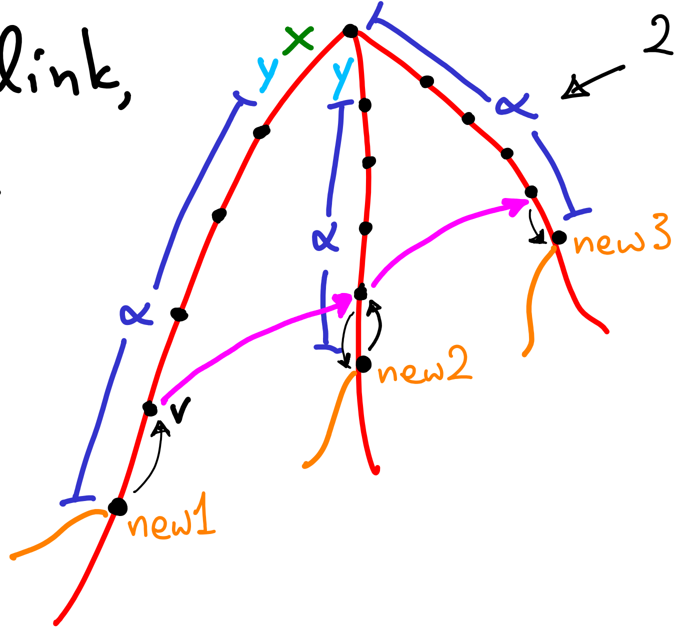
using indices we can move between nodes in $O(1)$ time

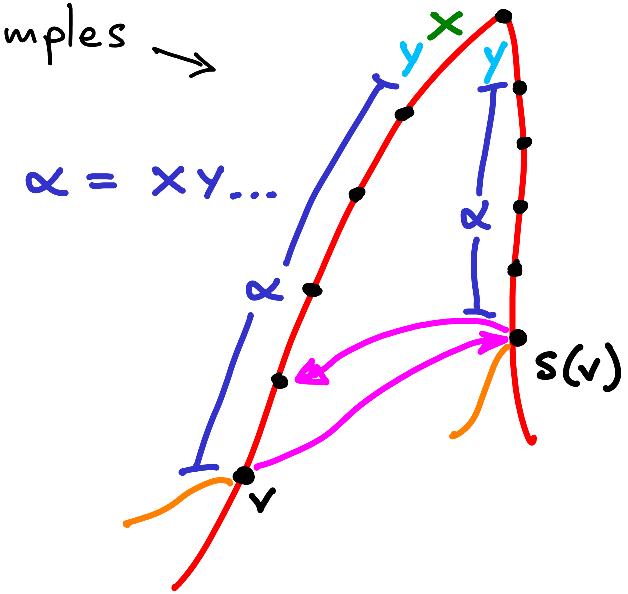
When following a suffix link, node depth (from root) can

- stay same
- increase (down to ~leaf)
- decrease
  (i.e. move up)

example to follow

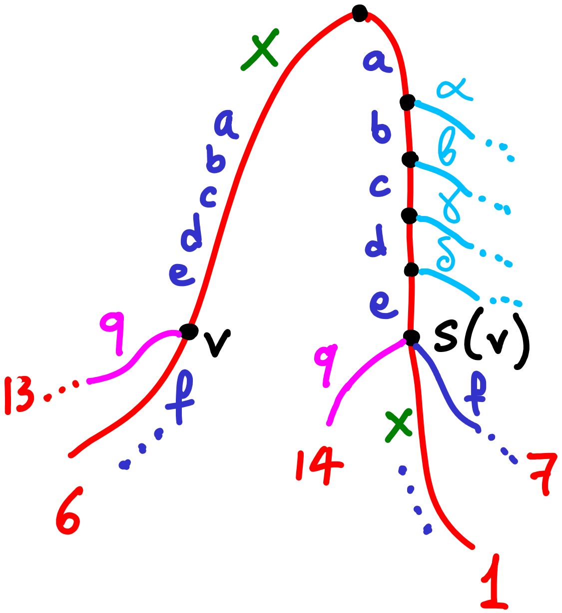
2 examples





example of v with suffix link to s(v)

where depth(v) << depth(s(v))



1

abcdexabcdefxabcdeqaαabβabcγabcdδ

6

xabcdefxabcdeqaαabβabcγabcdδ

7

abcdefxabcdeqaαabβabcγabcdδ

13

xabcdeqaαabβabcγabcdδ

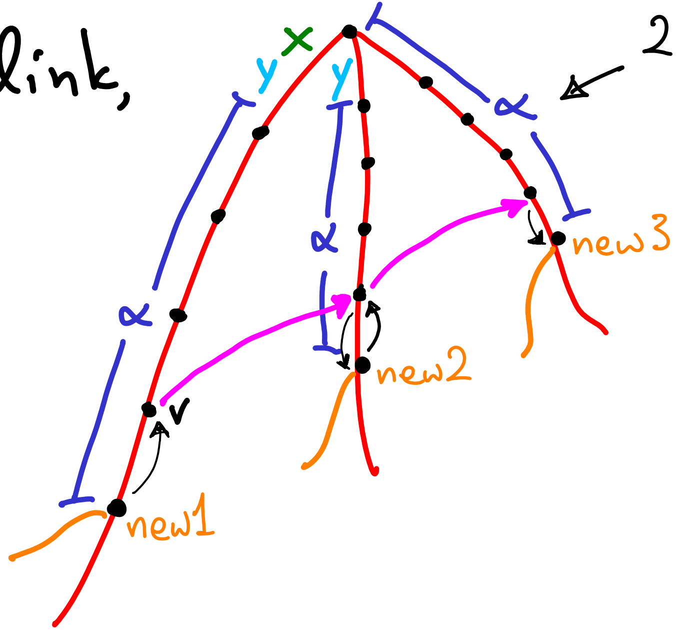
14

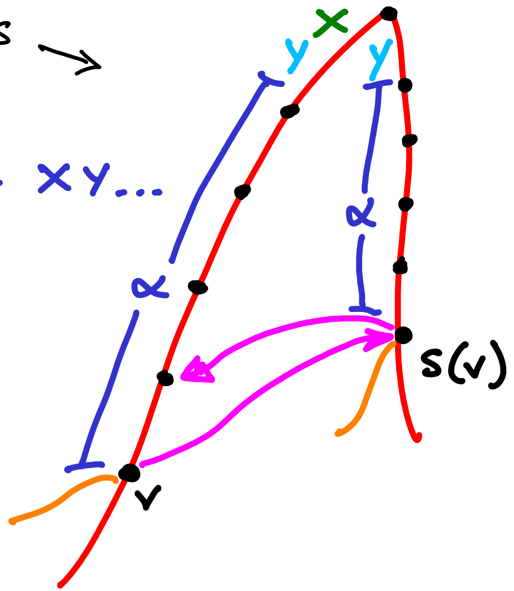
abcdeqaαabβabcγabcdδ

When following a suffix link, node depth (from root) can

- stay same
- increase (down to ~leaf)
- decrease (only by 1) (i.e. move up)

2 examples





α = xy...

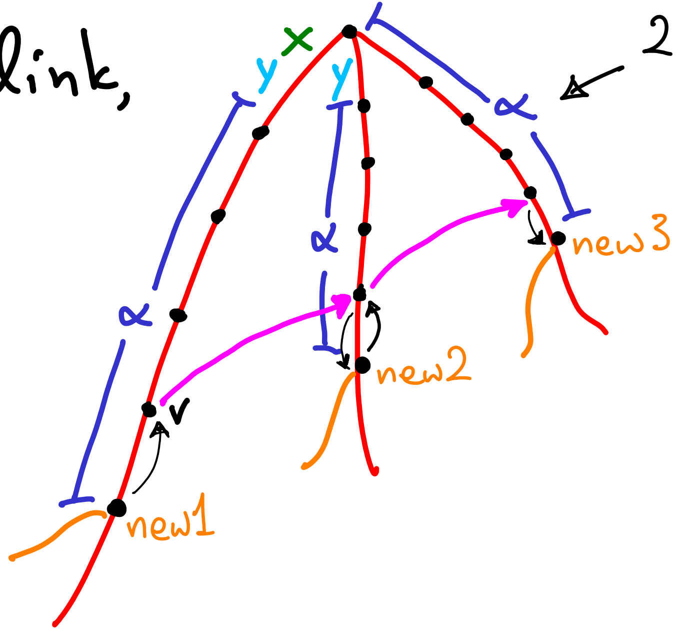
we've seen: if xy leads to v then y is in tree, leading to s(v)

So every node on path from x to v has a "mirror" node on path from root to s(v)

(reverse not necessarily true)

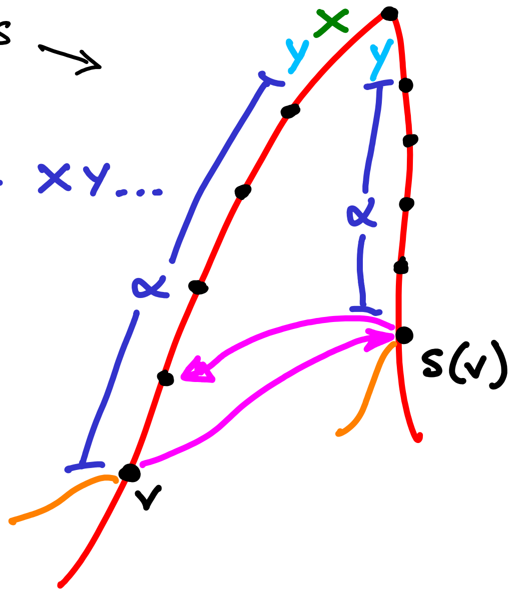
When following a suffix link, node depth (from root) can

- stay same
- increase (down to ~leaf)
- decrease (only by 1) (i.e. move up)



2 examples

$\alpha = xy\ldots$



amortize: total node hops per iteration = $O(i)$

[total upward = $O(i)$ & max depth = $O(i)$]

CONCLUSION

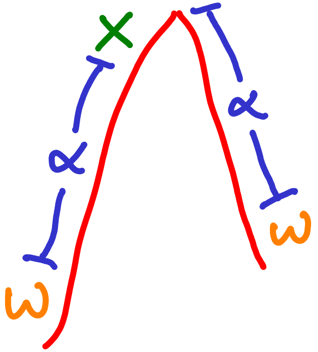
During iteration i, we visit $O(i)$ nodes

...spending $O(i)$ time to find each, excluding the first one: total work $O(i)$

which is always the longest suffix, i.e same node

Story so far: find "end" of next extension using suffix link from end of current extension (plus a bit of up/down)

What if current extension makes no new node?



stop current iteration

extend xα + ω
but xαω already in tree
(αω also in tree)

All future extensions in current iteration will also follow same case

extension

iteration

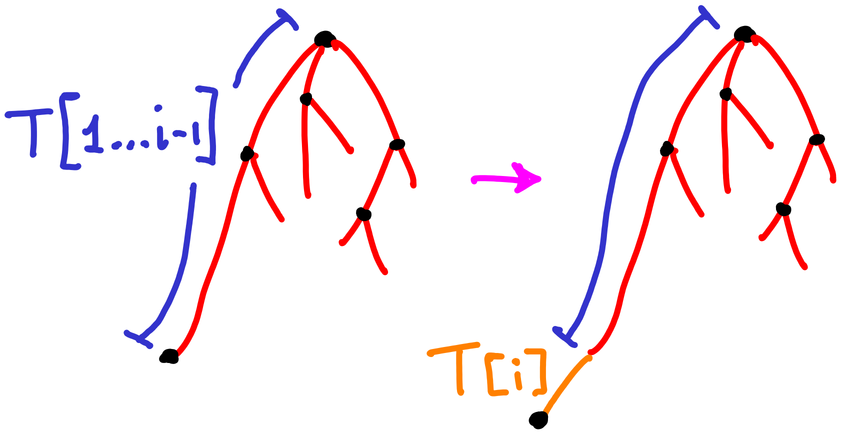
	1	2	3 … i …				|T|
1	2						
2	3	1					
3	1	1	1				
⋮	3	2	—	—			
	1	3	1	3	3		
	3	3	3	1	1	3	
(i) |T|	1	3	1	2	—	—	—

Table of cases applied in each extension of each iteration
↳ nothing right of 2

1) at a leaf → extend current label
2) not at a leaf, but new character is already there → do nothing
3) not at a leaf, new char. not there → make new edge w/ new character

Story so far: find "end" of next extension using suffix link from end of current extension (plus a bit of up/down)

What if current extension makes no new node?



extension

iteration (i)	1	2	3	... i ...			|T|
1	1						
2	1	1					
3	1	1	1				
	1	2	—	—			
⋮	1	3	1	3	3		
	1	3	3	1	1	3	
|T|	1	3	1	2	—	—	—

First extension of every iteration: case 1 (longest suffix ends at leaf)

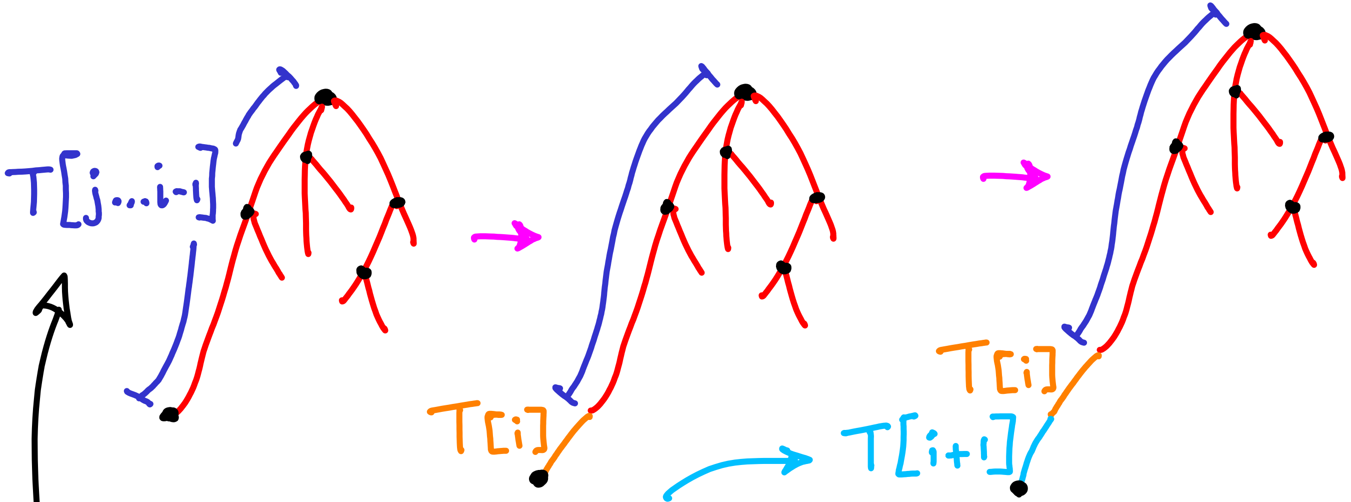
1) at a leaf → extend current label

2) not at a leaf, but new character is already there → do nothing

3) not at a leaf, new char. not there → make new edge w/ new character

Story so far: find "end" of next extension using suffix link from end of current extension (plus a bit of up/down)

What if current extension makes no new node?



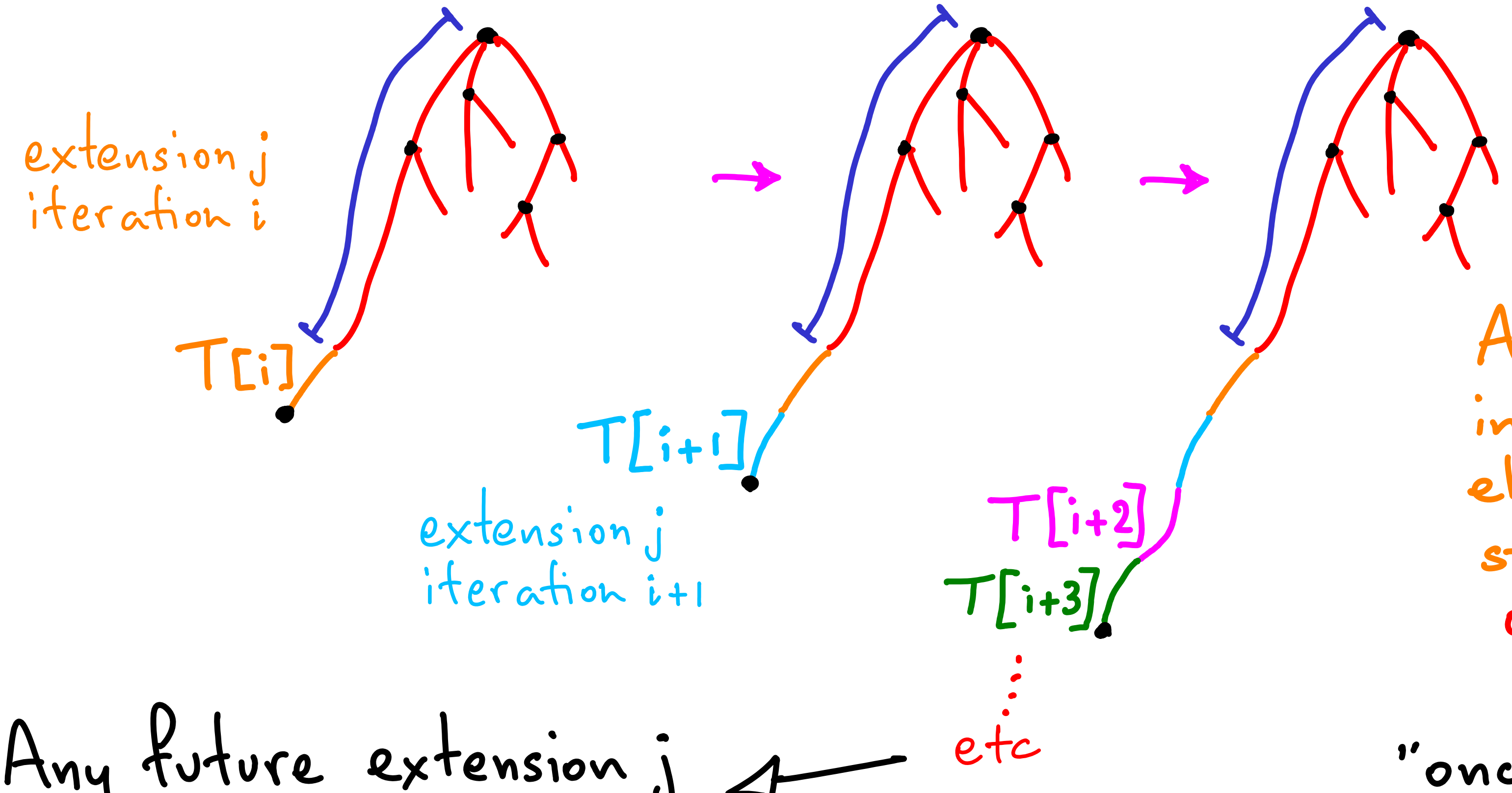
Extension j of iteration i+1

All suffixes considered in the future will divert elsewhere in the tree, or stop short on this path, or $\underbrace{\text{extend this path}}$

"once a leaf, always a leaf"

What if arbitrary extension j of iteration i uses case 1?

1) at a leaf → extend current label
2) not at a leaf, but new character is already there → do nothing
3) not at a leaf, new char. not there → make new edge w/ new character



[for future iterations extension k<j diverts; extension k>j too short]

All suffixes considered in the future will divert elsewhere in the tree, or stop short on this path, or extend this path

"once a leaf, always a leaf"

Any future extension j has same prefix, so will follow same path & use same rule

1) at a leaf → extend current label
2) not at a leaf, but new character is already there → do nothing
3) not at a leaf, new char. not there → make new edge w/ new character

what if extension j uses case 3 ? → creates a leaf

... same logic, must have 1's below

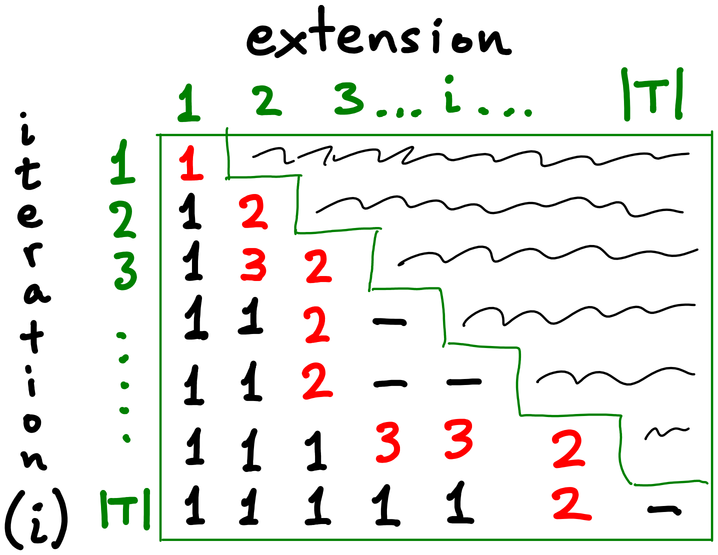
We've determined that below any 1 we can have only 1's

iteration (i) \ extension	1	2	3	...	i	...	|T|
1	1						
2	1	2					
3	1	3	2				
⋮	1	1	2	–			
⋮	1	1	2	–	–		
⋮	1	1	3	1	3	3	
|T|	1	1	1	1	1	1	2

1) at a leaf → extend current label

2) not at a leaf, but new character is already there → do nothing

3) not at a leaf, new char. not there → make new edge w/ new character



if 1 or 3 → all 1's below

if 2 → empty to right

$\leq 2|T|$ "interesting" entries

(still $O(T^2)$ work!)

How to handle the 1's: use a global variable = iteration #

- recall, the 1's involve leaf (edge) extensions
- edges are represented by indices in T.
- the "end" index just gets incremented: $i\text{-}1 \to i$

Last step: add $: makes all implicit suffixes proper

conclusion: suffix tree construction → $O(T)$ time & space