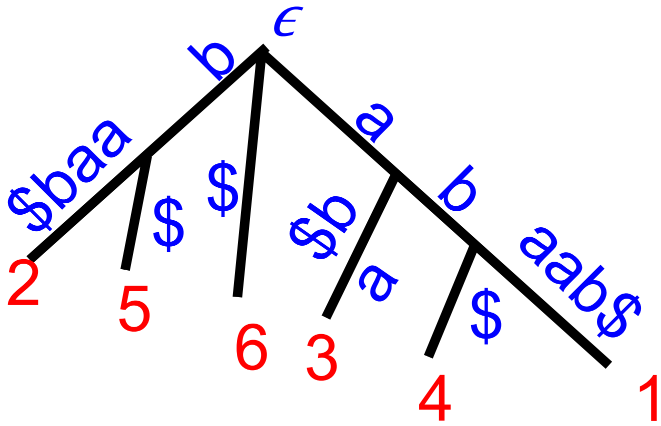
McCreight's suffix tree construction algorithm



String Algorithms; 12 Nov 2007

Motivation

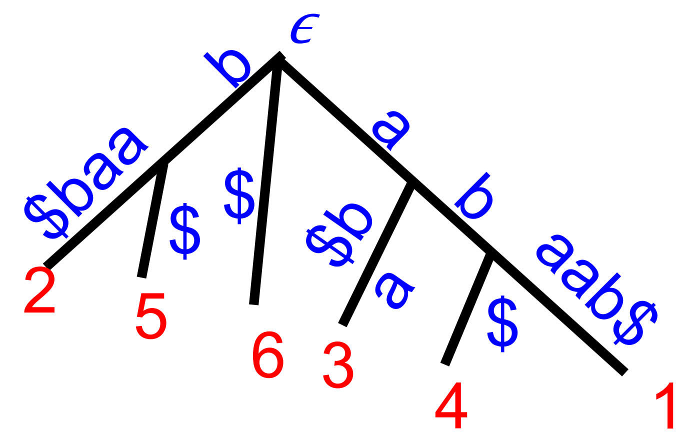
Recall: the suffix tree is an extremely useful data structure with space usage and construction time in $O(n)$.

Today we see the first algorithm for constructing a suffix tree in time $O(n)$.

Suffix trees

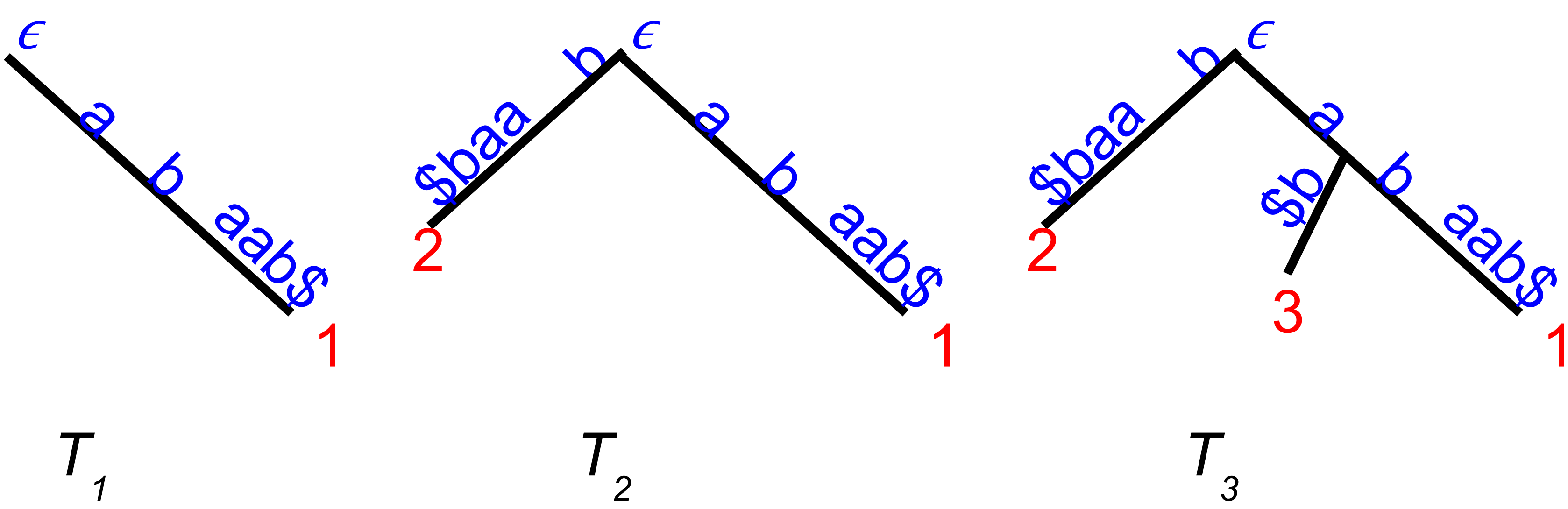
A suffix tree of a sequence, *x*, is a *compressed trie* of all suffixes of the sequence *x*$.

x=abaab
1: abaab$
2: baab$
3: aab$
4: ab$
5: b$
6: $



McCreight's algorithm

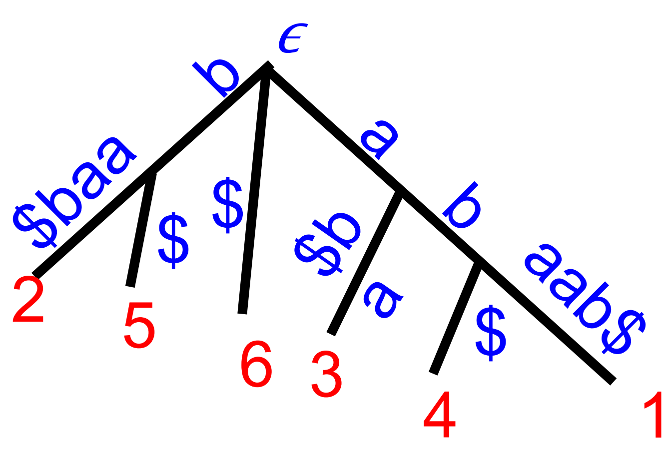
Iteratively, for i=1,...,n+1 build tries, T_i, where ... is a trie of sequences $x\$[1..n+1]$, $x\$[2..n+1]$, ..., $x\$[i..n+1]$



McCreight's algorithm

Iteratively, for i=1,...,n+1 build tries, T_i, where ... is a trie of sequences x\$[1..n+1], x\$[2..n+1], ..., x\$[i..n+1]

For i=n+1, T_i is the suffix tree for x.



McCreight's algorithm

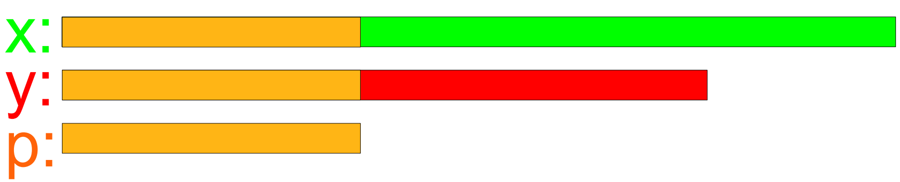
Iteratively, for i=1,...,n+1 build tries, T_i, where ... is a trie of sequences x\$[1..n+1], x\$[2..n+1], ..., x\$[i..n+1]

For i=n+1, T_i is the suffix tree for x.

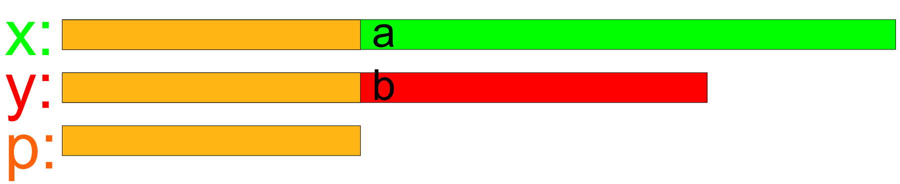
The essential trick is being clever in how we insert $x[i..n]\$$ into T_i so we don't spend $O(n^2)$ all in all.

Terminology

- A *common prefix* of x and y is a string, p, that is a prefix of both:

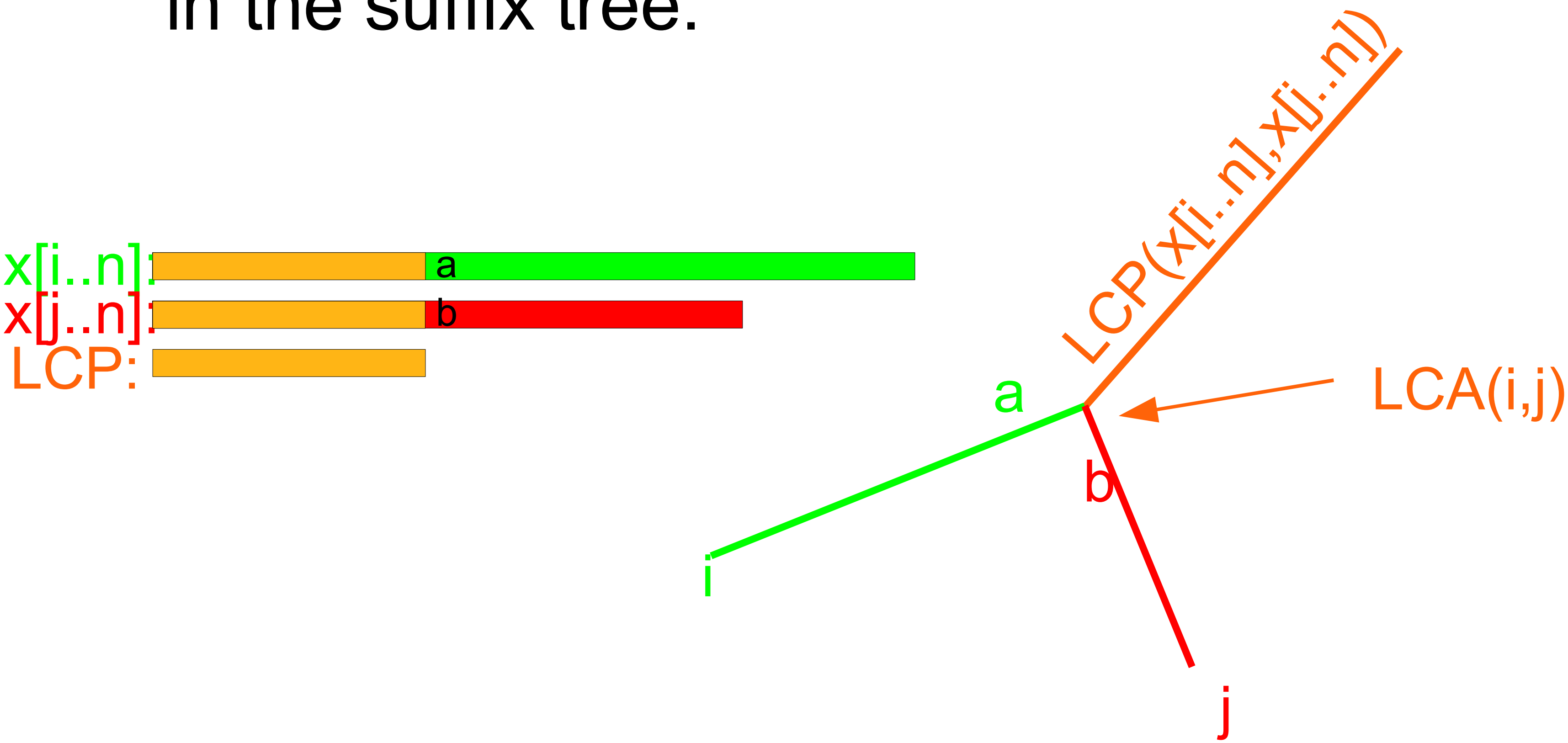


- The *longest* common prefix, p=LCP(x,y), is a prefix such that: $x[|p|+1] \neq y[|p|+1]$



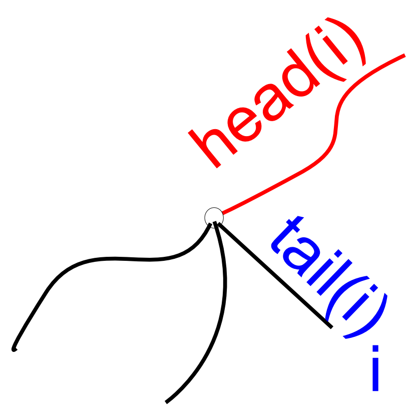
LCP and LCA

- For suffixes of x, x[i..n], x[j..n], their longest common prefix is their *lowest common ancestor* in the suffix tree:



Head and tail

- Let head(i) denote the longest LCP of $x[i..n]\$$ and $x[j..n]\$$ for all $j < i$
- Let tail(i) be the string such that $x[i..n]\$=$head($i$)tail($i$)
- Iteration i in McCreight's algorithm consist of
  - finding (or inserting) the node for head(i),
  - and appending tail(i)

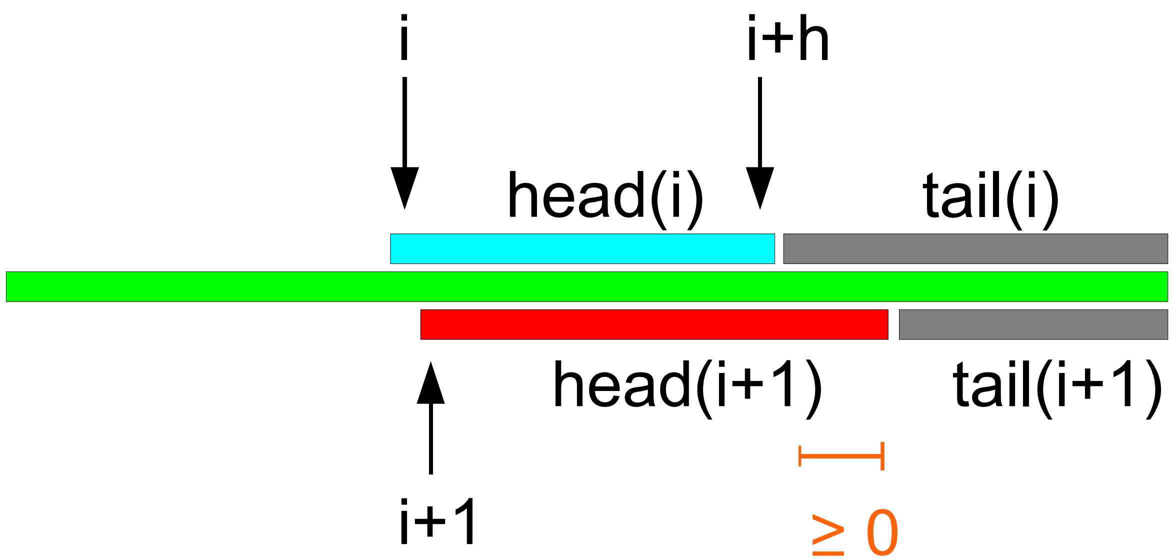


“The Trick”

The trick in McCreight's algorithm is a clever way of finding head(i)

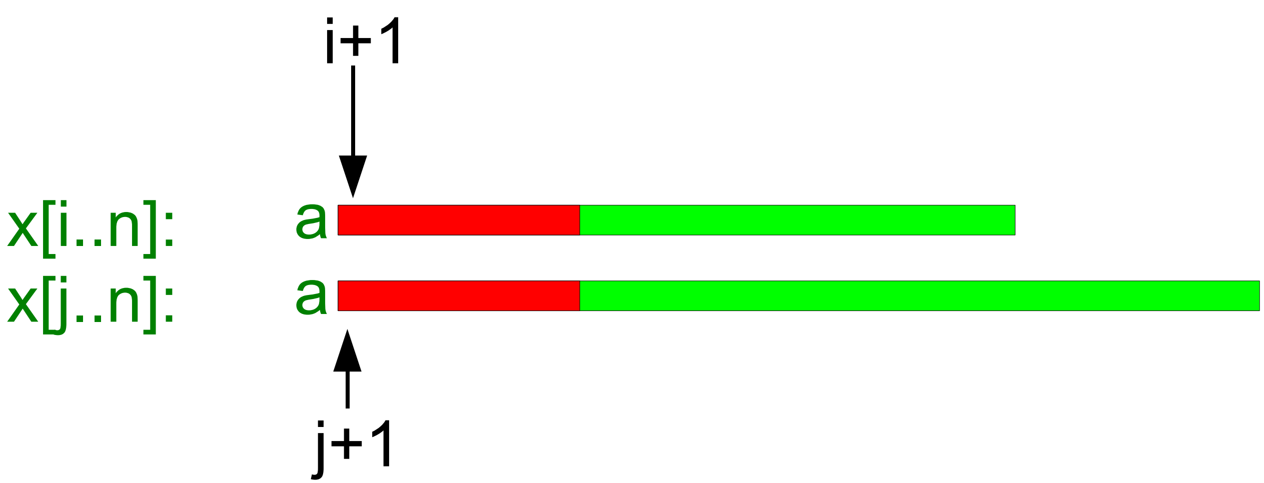
Lemma 5.2.1

Let head(i) = $x[i..i+h]$. Then $x[i+1..i+h]$ is a prefix of head(i+1)



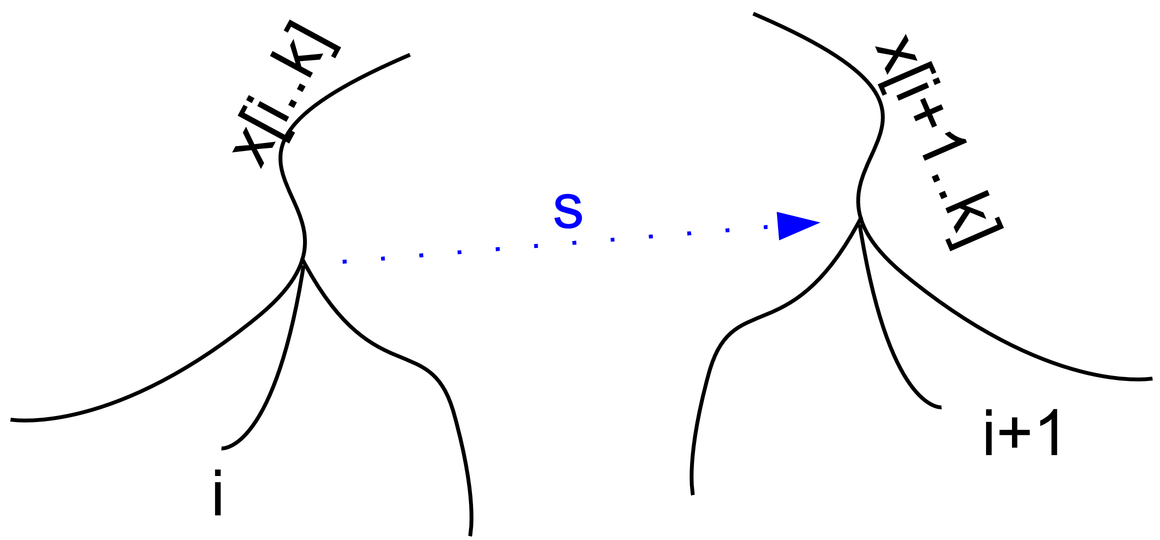
Proof

- Trivial for h=0 (head(i) empty), so assume h>0:
  - Let head(i) = ay: a
  - By def. ∃ j<i such that LCP(i,j)=ay
  - Thus suffix j+1 and i+1 share prefix y
  - Thus y is a prefix of LCP(i+1,j+1)
  - Thus y is a prefix of head(i+1)



Suffix link

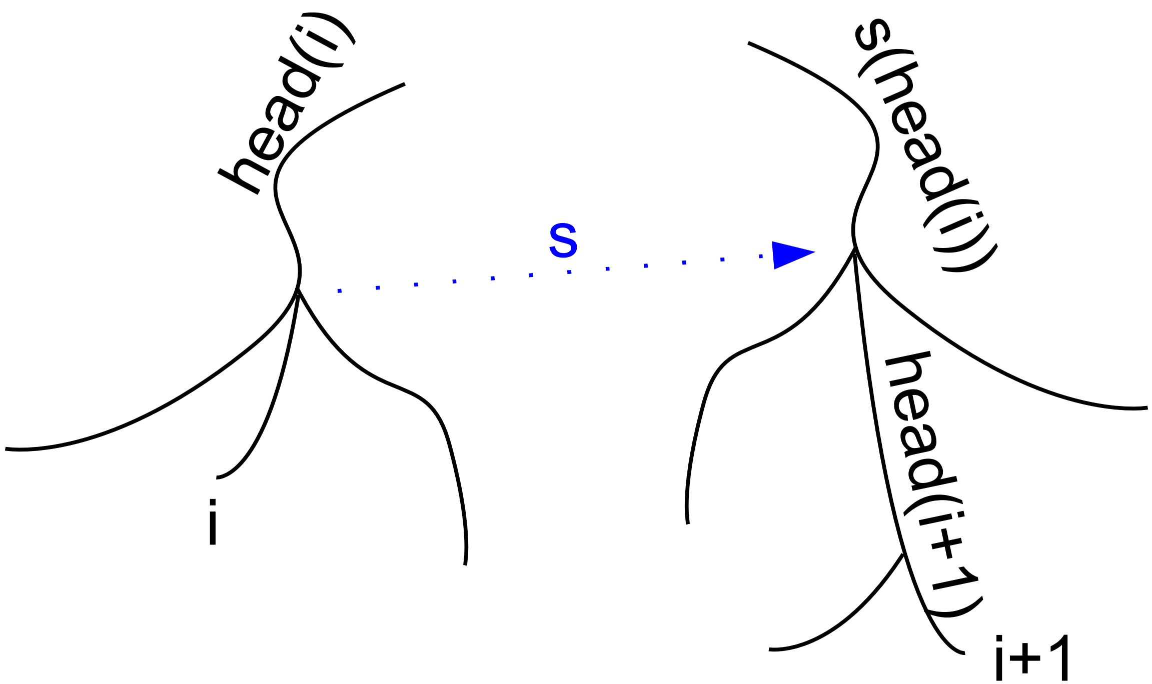
- Define s(u) = ϵ if u=ϵ, v if u=av
- As a pointer from x[i..k] to x[i+1..k]:



- (ex 5.2.3: if u is a node, so is s(u))

Corollary of Lemma 5.2.1

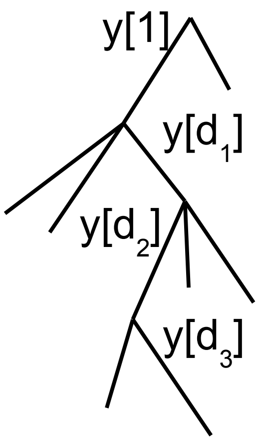
- s(head(i)) is a prefix of head(i+1)
- Thus: s(head(i)) is an ancestor of head(i+1)



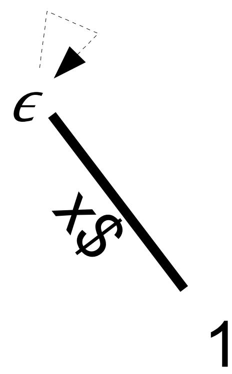
- ***s(head(i)) can be used as a shortcut!***

Slowscan and fastscan

- **Slowscan**: if we do not know if string *y* is in T_i, we must search character by character
- **Fastscan**: if we ***do know*** that *y* is in *x*, we can jump directly from node to node
  - At node u at (path-)depth d, follow the edge with label starting with *y*[*d*]
  - Continue until we reach the end of *y*
    - On a node (if *y* is in T_i)
    - Or on an edge (if *y* is a prefix of a string in T_i)

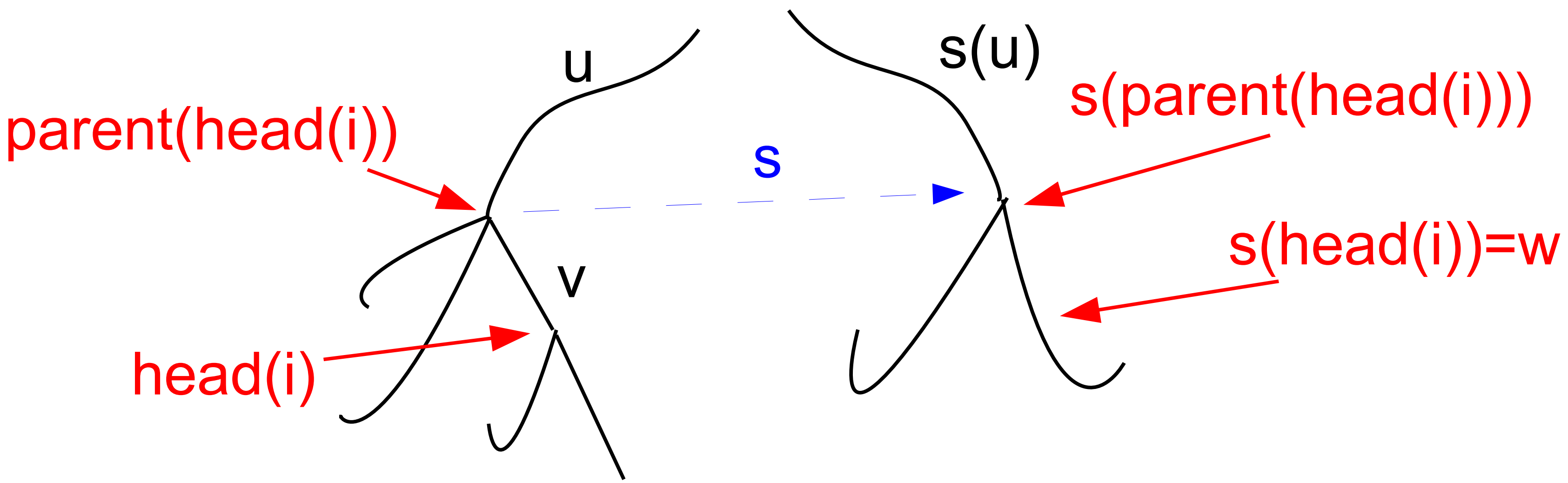


Sketch of McCreight's algorithm



- Begin with the tree T_1:
- For i=1,...,n, build tree T_{i+1} satisfying:
  - T_{i+1} is a compressed trie for x[j..n]$, $j \leq i+1$
  - All non-terminal nodes (with the possible exception of head(i)) have a suffix link s(-)
- Each iteration must:
  - Add node i+1
  - Potentially add head(i+1)
  - Add tail(i+1)
  - Add suffix link head(i) → s(head(i))

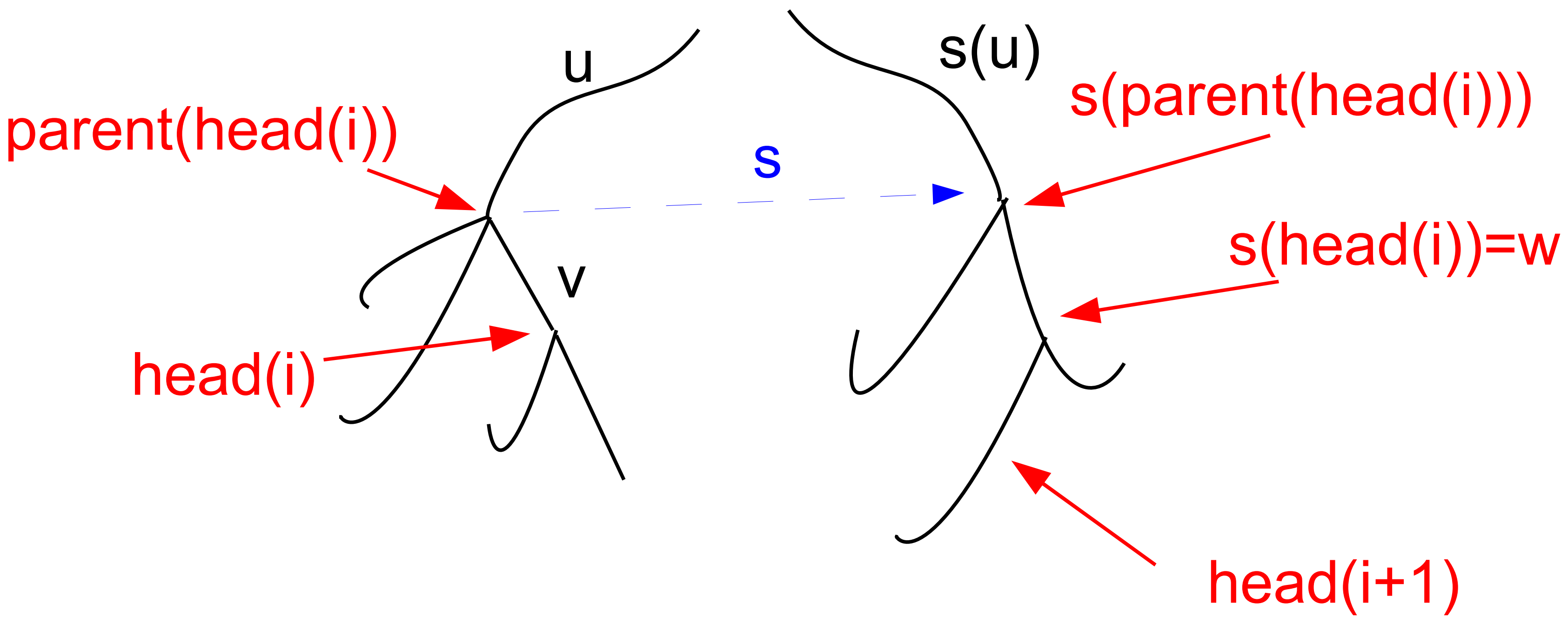
At the beginning of iteration *i*...



Let head(i)=uv and parent(head(i))=u and w=s(u)v=s(head(i))

By the invariant, s(parent(head(i))) and the suffix link exists;
by the lemma, w is an ancestor of head(i+1)

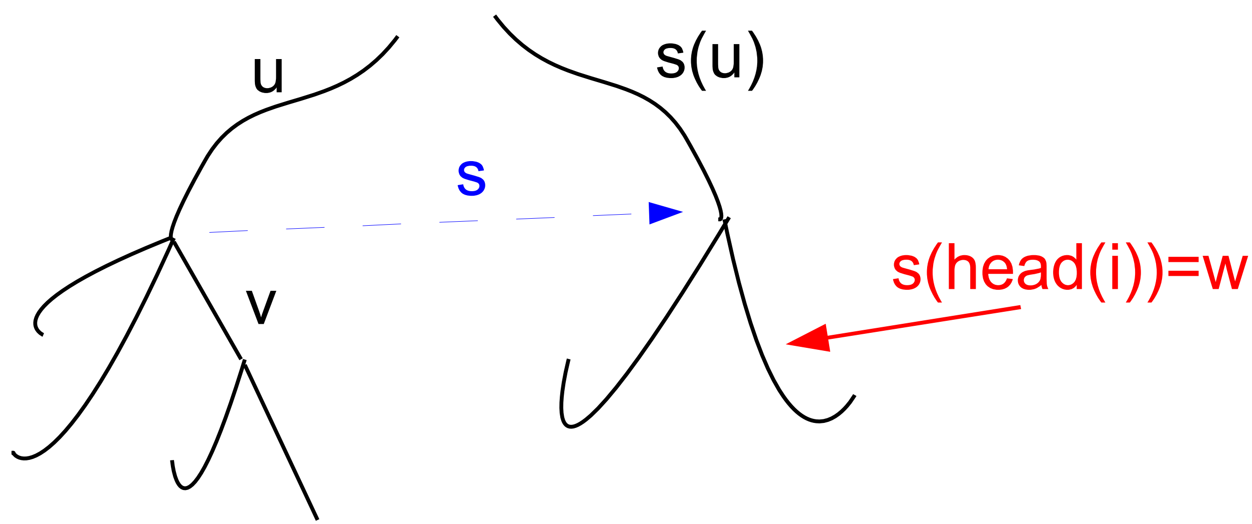
Steps in iteration *i*...



Move quickly to w, then search for head(i+1) starting there

Observe: w is in T_i

- head(i) is a prefix of x[j..n] for some j < i
- Thus w is a prefix of x[j+1..n] for some j < i
  - i.e w is a prefix of some suffix j ≤ i
  - i.e. w is in T_i
- **Consequently**: we can search for w from s(u) using **fastscan**!



If *w* is a node

- Update s(head(i)) := w
- Then search for head(i+1) using **slowscan**

If *w* is on an edge

- If w is not a node, then all suffix j<i with prefix w agree on the next letter
- By definition of head(i) there is j<i such that suffix x[i..n] and x[j..n] differs after head(i)
  - x[i+1..n] must also disagree at that character
  - Thus head(i+1) must be w

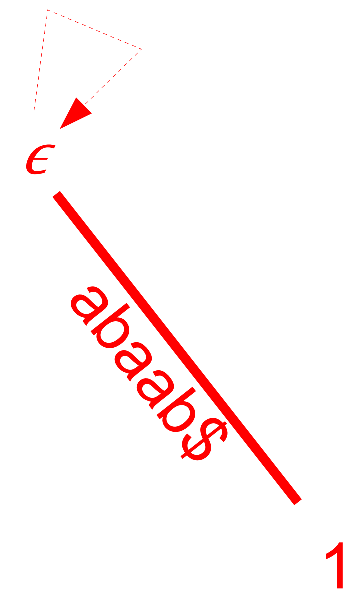
- Add node w, update head(i):=w and set head(i+1)=w

McCreight's algorithm

```
Construct tree for x[1..n]
for i = 1 to n do
     if head(i)=ϵ then
          head(i+1) = slowscan(ϵ,s(tail(i)))
          add i+1 and head(i+1) as node if necessary
          continue
     u = parent(head(i)) ; v = label(u,head(i))
     if u≠ϵ then w = fastscan(s(u),v)
     else        w = fastscan(ϵ,v[2..|v|])
     if w is an edge then
          add a node for w
          head(i+1) = w
     else if w is a node then
          head(i+1) = slowscan(w,tail(i))
          add head(i+1) as node if necessary
     s(head(i)) = w
     add leaf i+1 and edge between head(i+1) and i+1
```

Example: *x* = *abaab*

```
Construct tree for x[1..n]
for i = 1 to n do
     if head(i)=ϵ then
          head(i+1) = slowscan(ϵ,s(tail(i)))
          add i+1 and head(i+1) as node if necessary
          continue
     u = parent(head(i)) ; v = label(u,head(i))
     if u≠ϵ then w = fastscan(s(u),v)
     else        w = fastscan(ϵ,v[2..|v|])
     if w is an edge then
          add a node for w
          head(i+1) = w
     else if w is a node then
          head(i+1) = slowscan(w,tail(i))
          add head(i+1) as node if necessary
     s(head(i)) = w
     add leaf i+1 and edge between head(i+1) and i+1
```

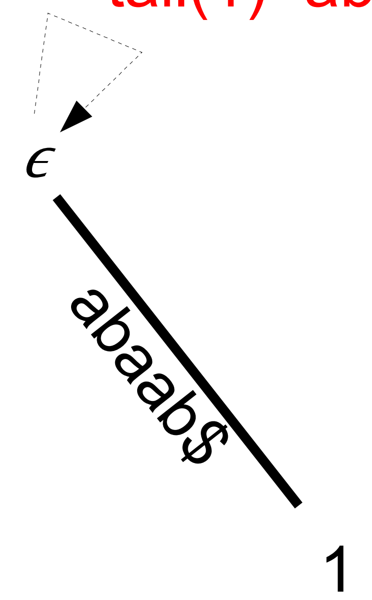


Example: *x* = *abaab*

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ε then
        head(i+1) = slowscan(ε,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ε then w = fastscan(s(u),v)
    else        w = fastscan(ε,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```

i=1

head(1)=ε
tail(1)=abaab$

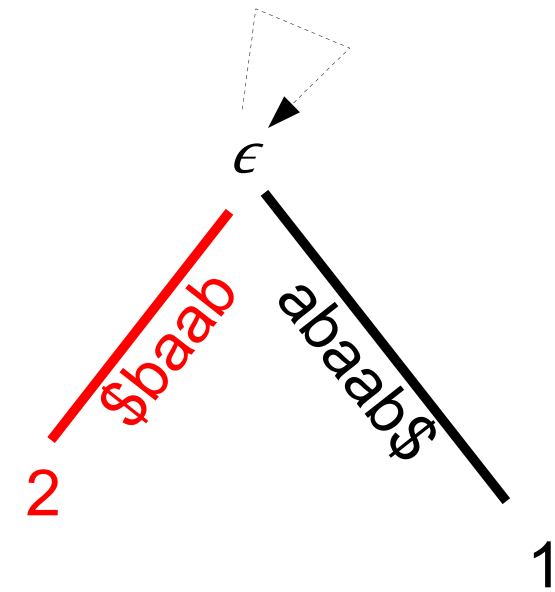


Example: x = *abaab*

i=1

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ϵ then
        head(i+1) = slowscan(ϵ,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ϵ then w = fastscan(s(u),v)
    else        w = fastscan(ϵ,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```

head(2)=ϵ
tail(2)=baab$

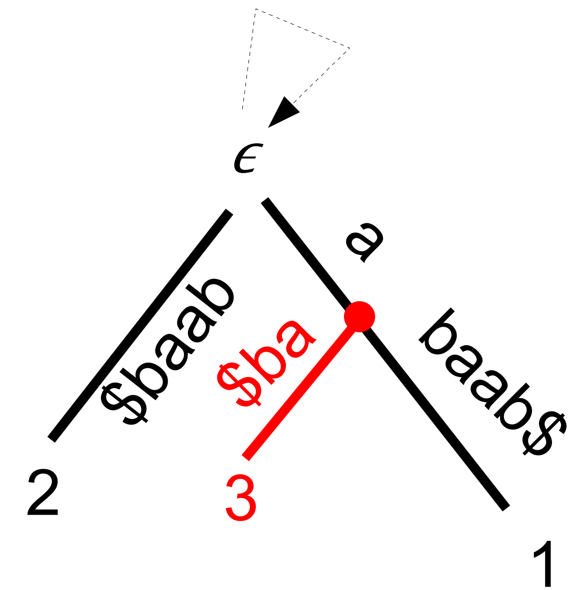


Example: *x* = *abaab*

head(2)=ϵ
tail(2)=baab$

i=2

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ϵ then
        head(i+1) = slowscan(ϵ,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ϵ then w = fastscan(s(u),v)
    else        w = fastscan(ϵ,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```



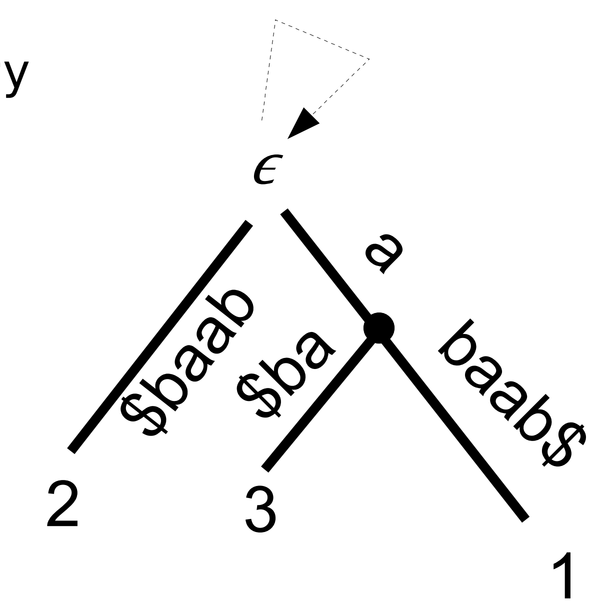
head(3)=a
tail(3)=ab$

Example: *x* = *abaab*

i=3

head(3)=a
tail(3)=ab$

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ϵ then
        head(i+1) = slowscan(ϵ,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ϵ then w = fastscan(s(u),v)
    else        w = fastscan(ϵ,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```

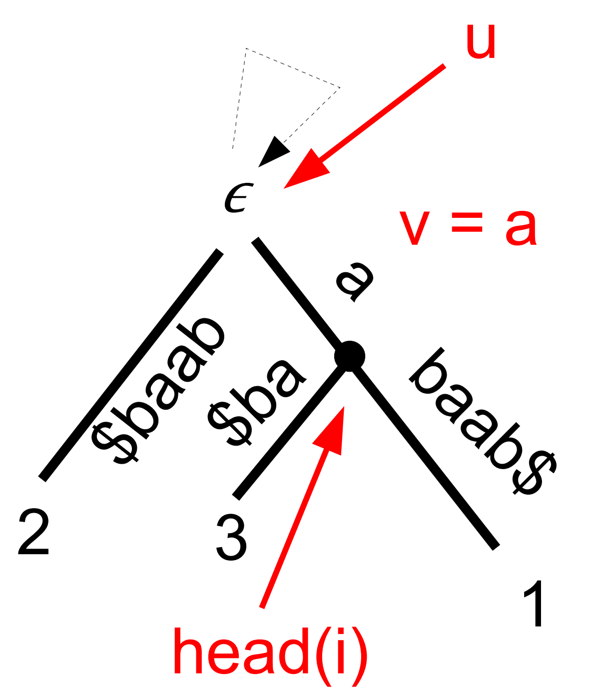


Example: *x* = *abaab*

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ε then
        head(i+1) = slowscan(ε,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ε then w = fastscan(s(u),v)
    else        w = fastscan(ε,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```

i=3

head(3)=a
tail(3)=ab$



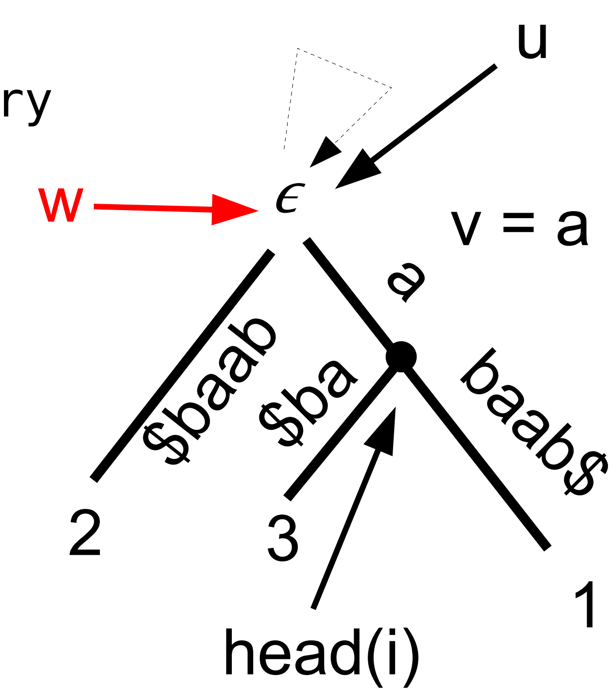
Example: *x* = *abaab*

i=3

head(3)=a
tail(3)=ab$

```
Construct tree for x[1..n]
for i = 1 to n do
     if head(i)=ε then
        head(i+1) = slowscan(ε,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
     u = parent(head(i)) ; v = label(u,head(i))
     if u≠ε then w = fastscan(s(u),v)
     else        w = fastscan(ε,v[2..|v|])
     if w is an edge then
        add a node for w
        head(i+1) = w
     else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
     s(head(i)) = w
     add leaf i+1 and edge between head(i+1) and i+1
```

v[2..|v|]=ε

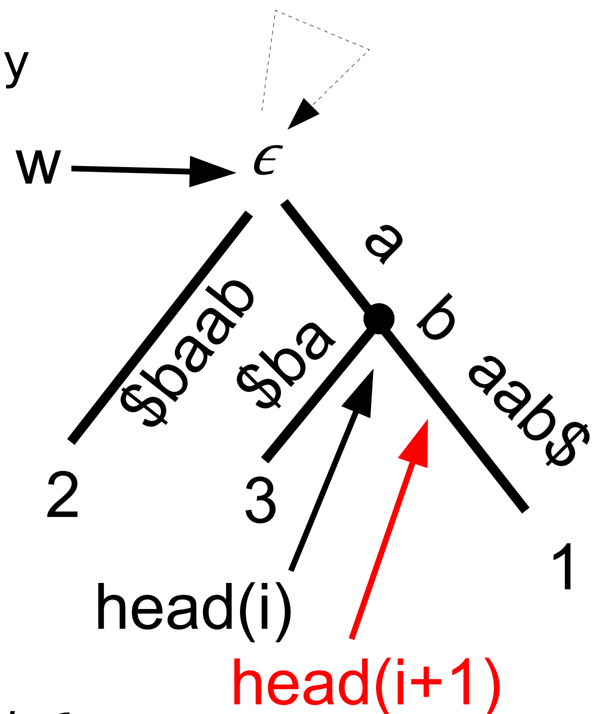


Example: *x* = *abaab*

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ϵ then
        head(i+1) = slowscan(ϵ,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ϵ then w = fastscan(s(u),v)
    else        w = fastscan(ϵ,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```

i=3

head(3)=a
tail(3)=ab$

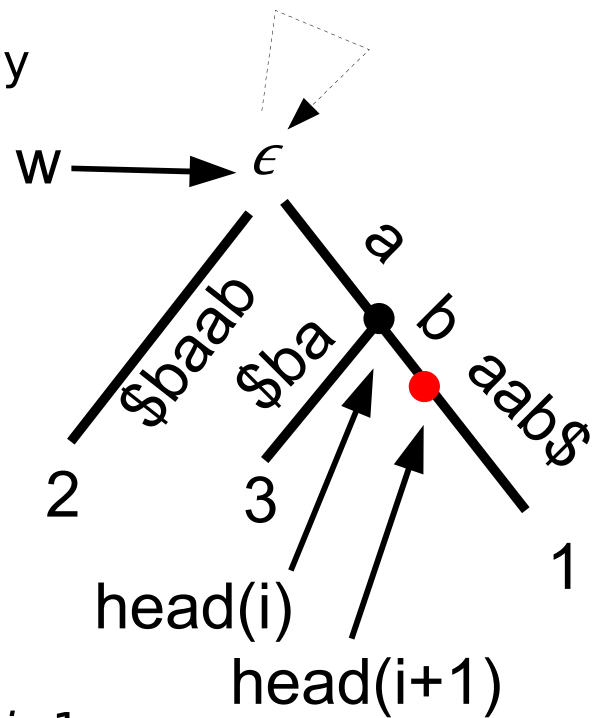


Example: *x* = *abaab*

i=3

head(3)=a
tail(3)=ab$

```
Construct tree for x[1..n]
for i = 1 to n do
     if head(i)=ϵ then
          head(i+1) = slowscan(ϵ,s(tail(i)))
          add i+1 and head(i+1) as node if necessary
          continue
     u = parent(head(i)) ; v = label(u,head(i))
     if u≠ϵ then w = fastscan(s(u),v)
     else         w = fastscan(ϵ,v[2..|v|])
     if w is an edge then
          add a node for w
          head(i+1) = w
     else if w is a node then
          head(i+1) = slowscan(w,tail(i))
          add head(i+1) as node if necessary
     s(head(i)) = w
     add leaf i+1 and edge between head(i+1) and i+1
```

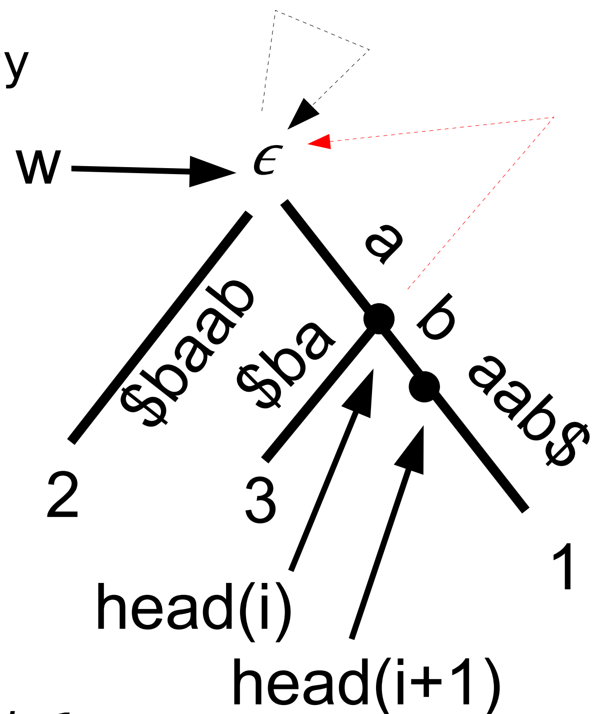


Example: *x* = *abaab*

i=3

head(3)=a
tail(3)=ab$

```
Construct tree for x[1..n]
for i = 1 to n do
     if head(i)=ϵ then
          head(i+1) = slowscan(ϵ,s(tail(i)))
          add i+1 and head(i+1) as node if necessary
          continue
     u = parent(head(i)) ; v = label(u,head(i))
     if u≠ϵ then w = fastscan(s(u),v)
     else         w = fastscan(ϵ,v[2..|v|])
     if w is an edge then
          add a node for w
          head(i+1) = w
     else if w is a node then
          head(i+1) = slowscan(w,tail(i))
          add head(i+1) as node if necessary
     s(head(i)) = w
     add leaf i+1 and edge between head(i+1) and i+1
```

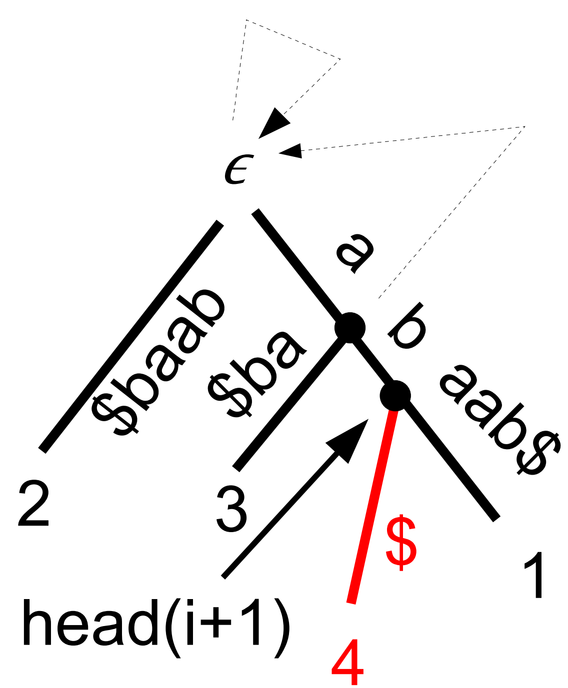


Example: *x* = *abaab*

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ϵ then
        head(i+1) = slowscan(ϵ,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ϵ then w = fastscan(s(u),v)
    else        w = fastscan(ϵ,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```

i=3

head(3)=a
tail(3)=ab$

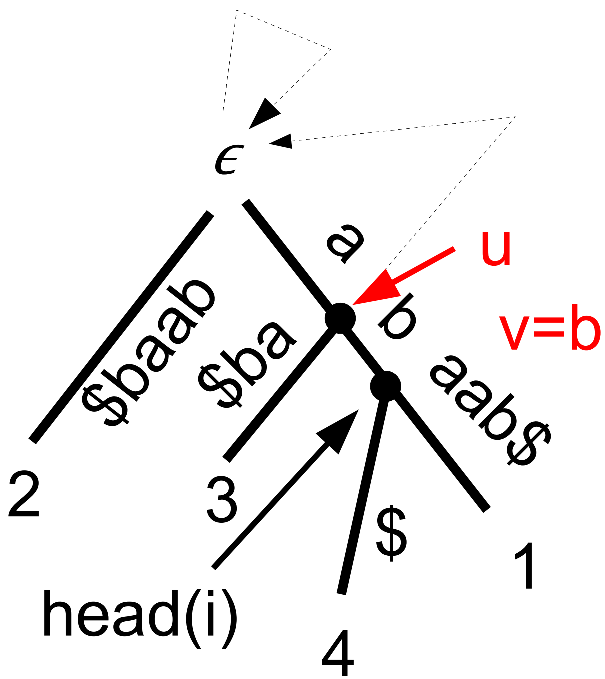


Example: x = *abaab*

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ϵ then
        head(i+1) = slowscan(ϵ,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ϵ then w = fastscan(s(u),v)
    else        w = fastscan(ϵ,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```

i=4

head(4)=ab
tail(4)=$

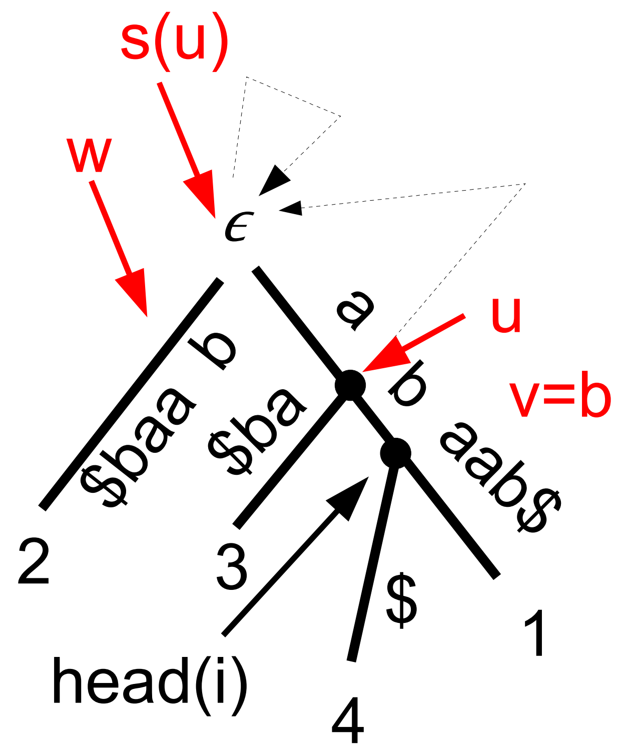


Example: *x* = *abaab*

i=4

head(4)=ab
tail(4)=$

```
Construct tree for x[1..n]
for i = 1 to n do
     if head(i)=ϵ then
          head(i+1) = slowscan(ϵ,s(tail(i)))
          add i+1 and head(i+1) as node if necessary
          continue
     u = parent(head(i)) ; v = label(u,head(i))
     if u≠ϵ then w = fastscan(s(u),v)
     else        w = fastscan(ϵ,v[2..|v|])
     if w is an edge then
          add a node for w
          head(i+1) = w
     else if w is a node then
          head(i+1) = slowscan(w,tail(i))
          add head(i+1) as node if necessary
     s(head(i)) = w
     add leaf i+1 and edge between head(i+1) and i+1
```

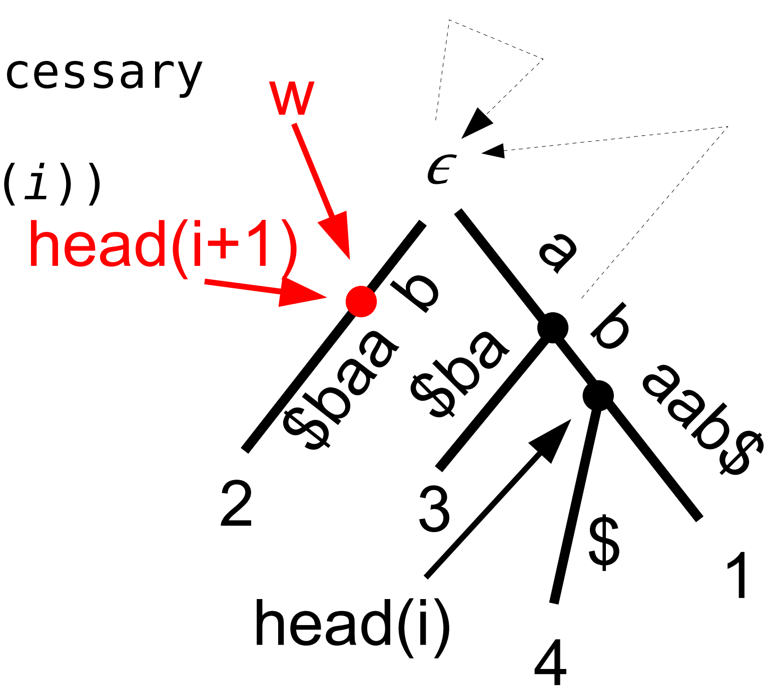


Example: *x* = *abaab*

i=4

head(4)=ab
tail(4)=$

```
Construct tree for x[1..n]
for i = 1 to n do
    if head(i)=ϵ then
        head(i+1) = slowscan(ϵ,s(tail(i)))
        add i+1 and head(i+1) as node if necessary
        continue
    u = parent(head(i)) ; v = label(u,head(i))
    if u≠ϵ then w = fastscan(s(u),v)
    else        w = fastscan(ϵ,v[2..|v|])
    if w is an edge then
        add a node for w
        head(i+1) = w
    else if w is a node then
        head(i+1) = slowscan(w,tail(i))
        add head(i+1) as node if necessary
    s(head(i)) = w
    add leaf i+1 and edge between head(i+1) and i+1
```

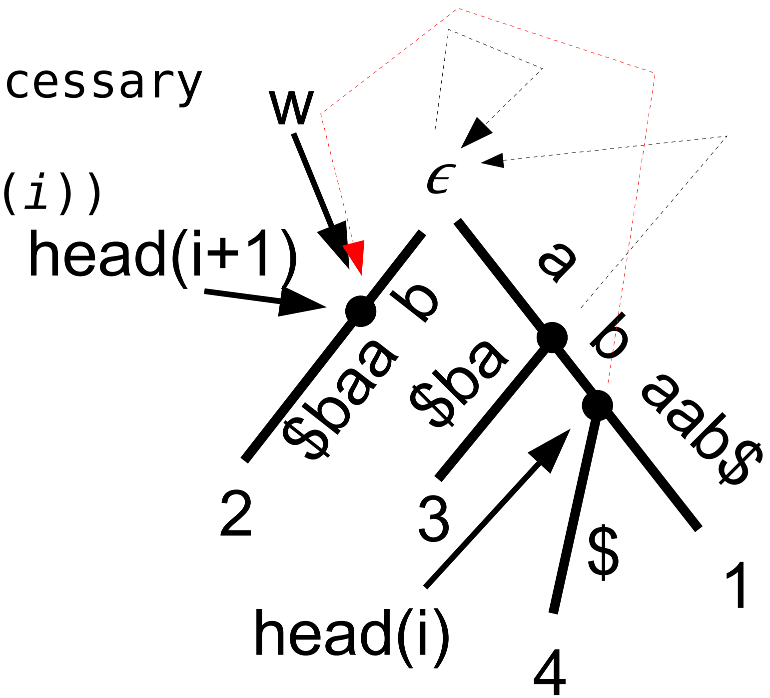


Example: *x* = *abaab*

```
Construct tree for x[1..n]
for i = 1 to n do
     if head(i)=ϵ then
          head(i+1) = slowscan(ϵ,s(tail(i)))
          add i+1 and head(i+1) as node if necessary
          continue
     u = parent(head(i)) ; v = label(u,head(i))
     if u≠ϵ then w = fastscan(s(u),v)
     else         w = fastscan(ϵ,v[2..|v|])
     if w is an edge then
          add a node for w
          head(i+1) = w
     else if w is a node then
          head(i+1) = slowscan(w,tail(i))
          add head(i+1) as node if necessary
     s(head(i)) = w
     add leaf i+1 and edge between head(i+1) and i+1
```

i=4

head(4)=ab
tail(4)=$

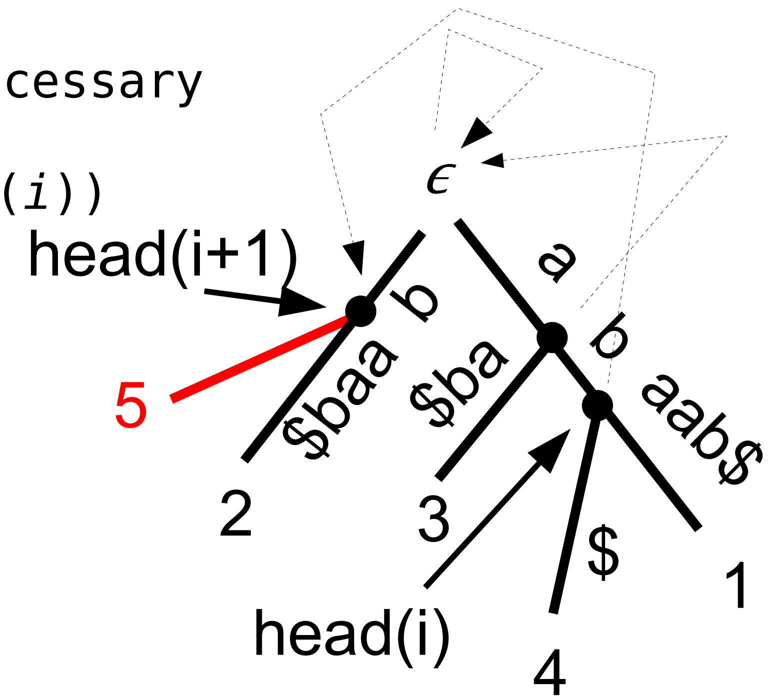


Example: *x* = *abaab*

```
Construct tree for x[1..n]
for i = 1 to n do
     if head(i)=ϵ then
          head(i+1) = slowscan(ϵ,s(tail(i)))
          add i+1 and head(i+1) as node if necessary
          continue
     u = parent(head(i)) ; v = label(u,head(i))
     if u≠ϵ then w = fastscan(s(u),v)
     else         w = fastscan(ϵ,v[2..|v|])
     if w is an edge then
          add a node for w
          head(i+1) = w
     else if w is a node then
          head(i+1) = slowscan(w,tail(i))
          add head(i+1) as node if necessary
     s(head(i)) = w
     add leaf i+1 and edge between head(i+1) and i+1
```

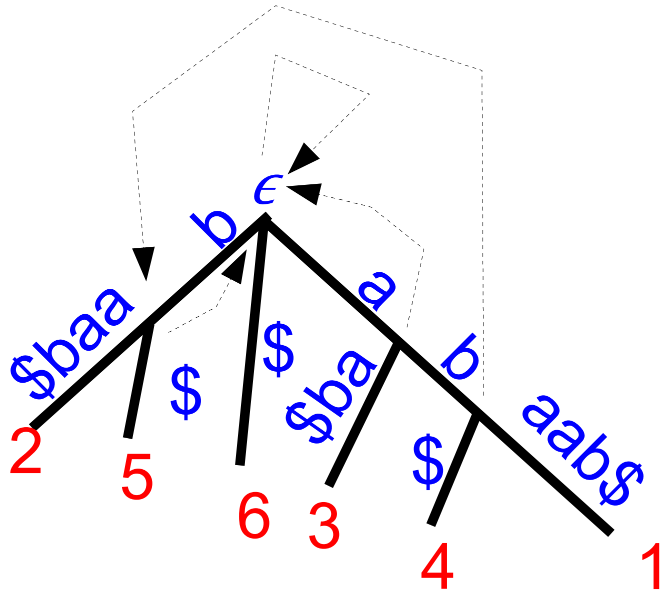
i=4

head(4)=ab
tail(4)=$



Done

- Nothing new for i=5, inserting $...



Correctness

Correctness follows from the invariant:

- At iteration i we have a trie of all suffixes $j < i$.
- After the final iteration we have a trie of all suffixes of $x\$$, i.e. we have the suffix tree of x.

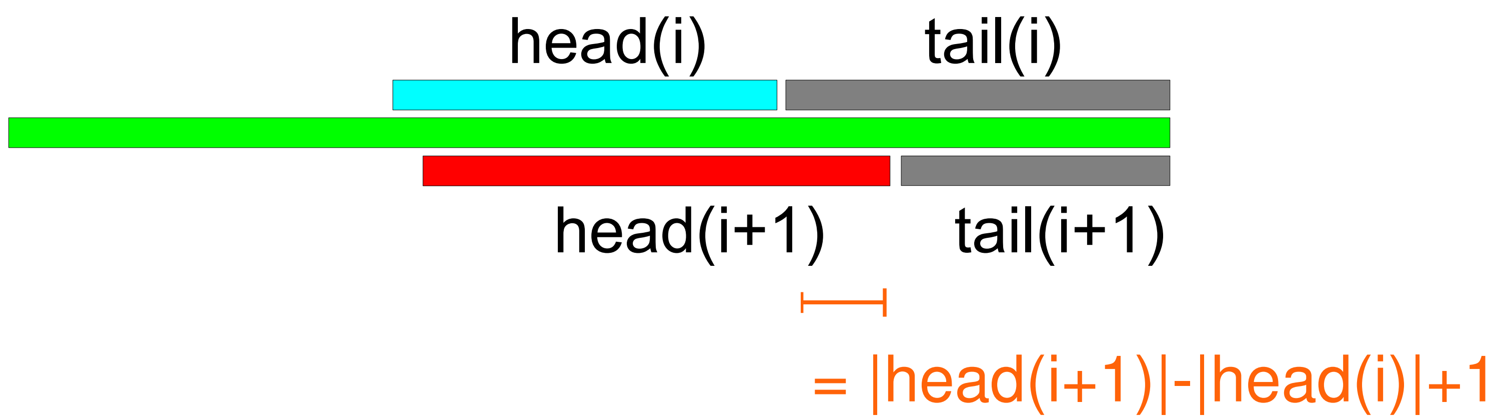
Time and space usage

Everything but searching is constant time per suffix, so the running time is $O(n$ + "slowscan" + "fastscan").

We are not using more space than time, so the space usage is the same.

Slowscan time usage

- We use slowscan to find head(i+1) from w=s(head(i)), i.e. time |head(i+1)|-|head(i)|+1 for iteration i



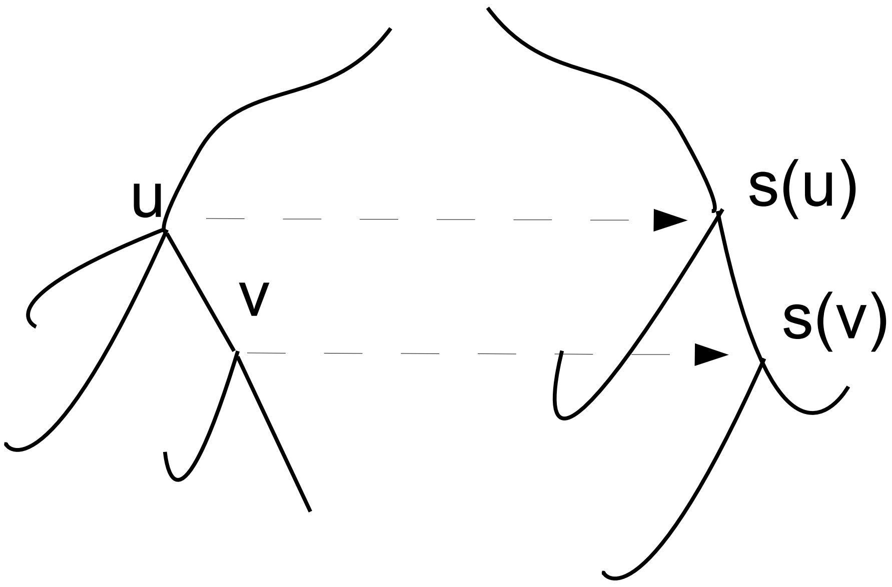
- A telescoping sum
  - Sum = |head(n+1)|-|head(1)|+n
  - Equal to n since head(n+1)=head(1)=ϵ

Fastscan time usage

- Fastscan uses time proportional to the number of nodes it process
- Define d(v) as the (node-)depth of node v
  - Fastscan increases the node depth
  - Following parent and suffix pointers decreases the node depth
- Time usage of fastscan is bounded by the total depth-increase (Amortized analysis)

Proposition

- $d(v) \leq d(s(v))+1$
- Proof:
  - For any ancestor u of v, s(u) is an ancestor of s(v)
  - Except for the empty prefix and the single letter prefix of v, the s(u)'s are different

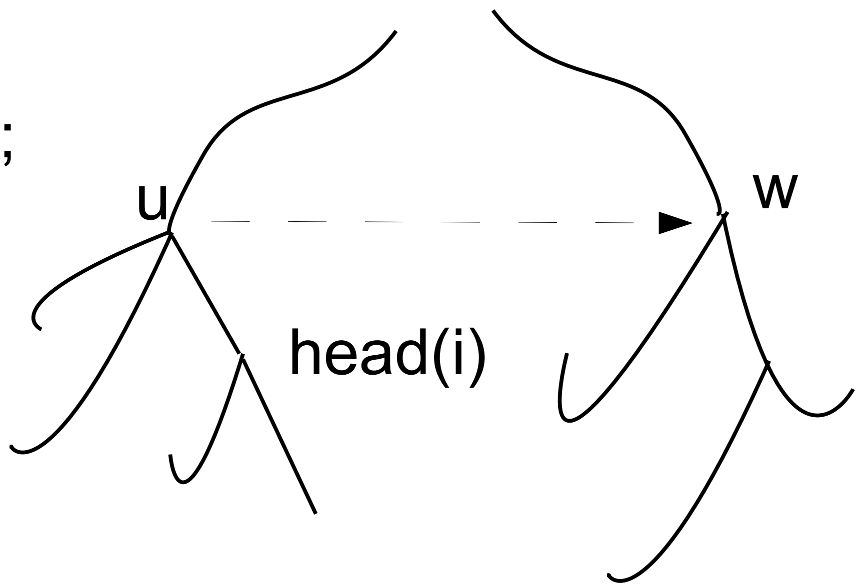


Corollary

- In each step, before calling fastscan, we decrease the depth by at most 2:

$d(u) = d(head(i))-1;$

$d(w) \geq d(u)-1$



- The total decrease is thus 2n

Time usage of fastscan

- The time usage of fastscan is bounded by n plus the total decrease of depth,
  - i.e. the time usage of fastscan is O(n)

Summary

We iteratively build tries of suffixes of *x*.

Using *suffix links* and *fastscan* we can quickly find where to insert the next suffix in our current trie.

By amortized analysis, the total running time becomes linear.