On-line Linear-time Construction of Word Suffix Trees

Shunsuke Inenaga

(Japan Society for the Promotion of Science & Kyushu University)

Masayuki Takeda

(Kyushu University & JST)

Pattern Searching Problem

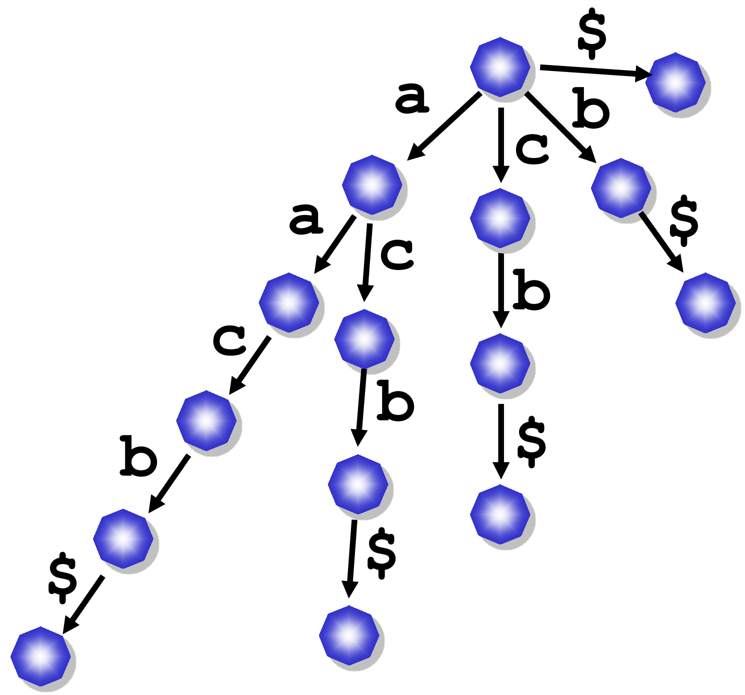
- Given : text ***T*** in Σ^* and pattern ***P*** in Σ^*
- Find : an occurrence of ***P*** in ***T***
  - Σ: *alphabet*
  - Σ^*: set of *strings*
- Using an indexing structure for ***T***, we can solve the above problem in $O(|\boldsymbol{P}|)$ time.

Suffix Trie

- A trie representing **all** suffixes of ***T***

T = `aacb$`

```
aacb$
 acb$
  cb$
   b$
    $
```



Introducing Word Separator #

- # : word separator - special symbol *not* in Σ
- $\boldsymbol{D} = \Sigma^* \#$: dictionary of *words*
- Text $\boldsymbol{T}$: an element of $\boldsymbol{D}^+$
  ($\boldsymbol{T}$ is a sequence $\boldsymbol{T}_1\boldsymbol{T}_2\ldots\boldsymbol{T}_k$ of k words in $\boldsymbol{D}$)
- e.g., $\boldsymbol{T}$ = This#is#a#pen#
  - $\Sigma = \{A,\ldots,z\}$
  - $\boldsymbol{D}$ = {...,This#,...,a#,...is#,...pen#,...}

Word-level Pattern Searching Problem

- Given: text ***T*** in ***D***$^+$ and pattern ***P*** in ***D***$^+$
- Find: an occurrence of ***P*** in ***T*** which immediately follows #

e.g.

The#space#runner#is#not#your#good#pace#runner#

Word-level Pattern Searching Problem

- Given: text ***T*** in ***D***$^+$ and pattern ***P*** in ***D***$^+$
- Find: an occurrence of ***P*** in ***T*** which immediately follows #

e.g.

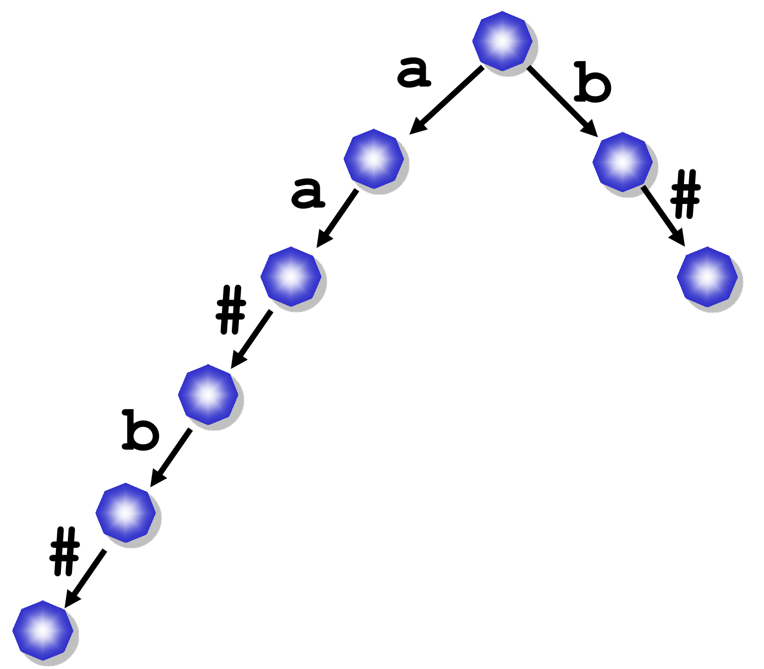
The#s<u>pace#runner</u>#is#not#your#good#**<u>pace#runner#</u>**

Word Suffix Trie

- A trie representing the suffixes of ***T*** which immediately follows **#** (and ***T*** itself).

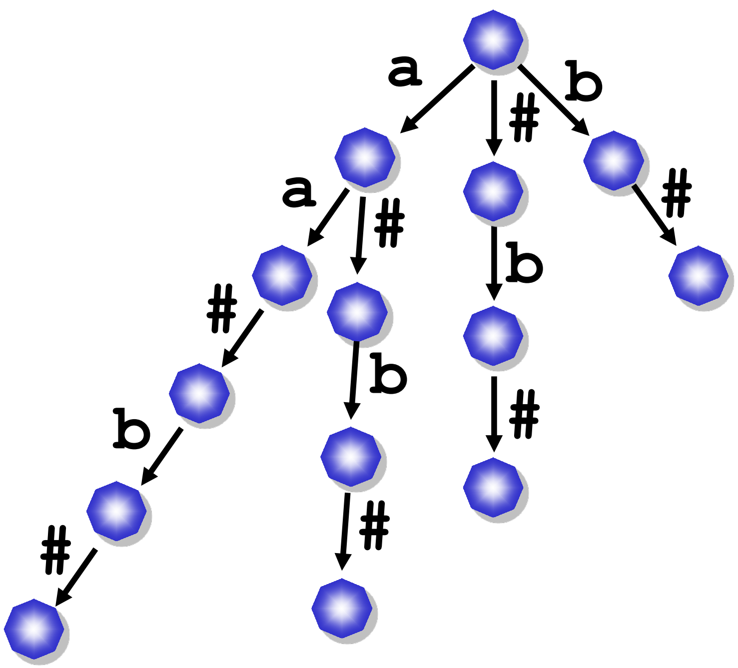
T = **aa#b#**

aa#b#
a#b#
#b#
b#
#

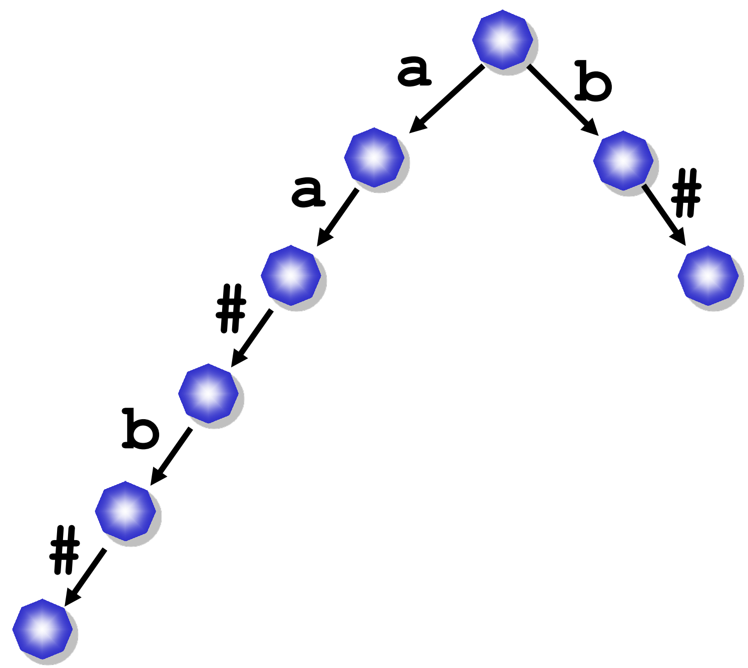


Comparison

T **= `aa#b#`**



Suffix Trie



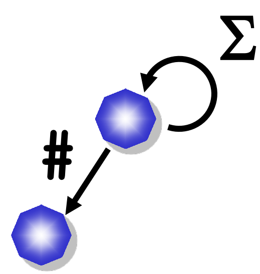
Word Suffix Trie

Construction

- Suffix Trie：Ukkonen's on-line algorithm（1995）
- Word Suffix Trie：We modify Ukkonen's algorithm by:
  - Using *minimum DFA accepting dictionary* ***D***
  - Redefining *suffix links*

Minimum DFA

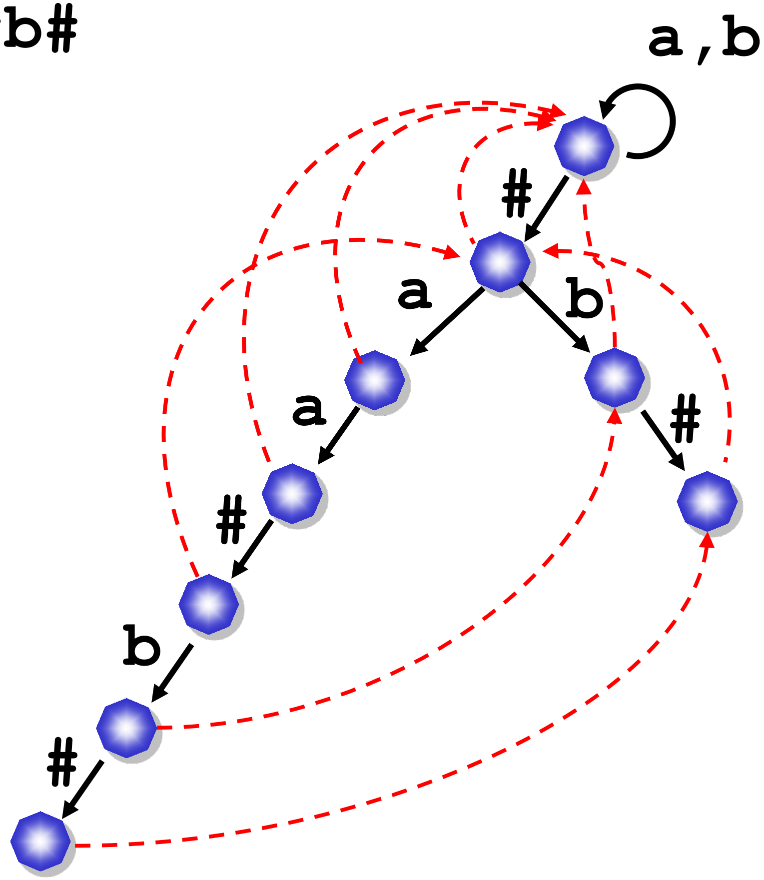
- The minimum DFA accepting $\boldsymbol{D} = \Sigma^* \#$ clearly requires constant space (for fixed Σ).



- *We replace the root node of the suffix trie with the final state of the DFA.*

Suffix Links

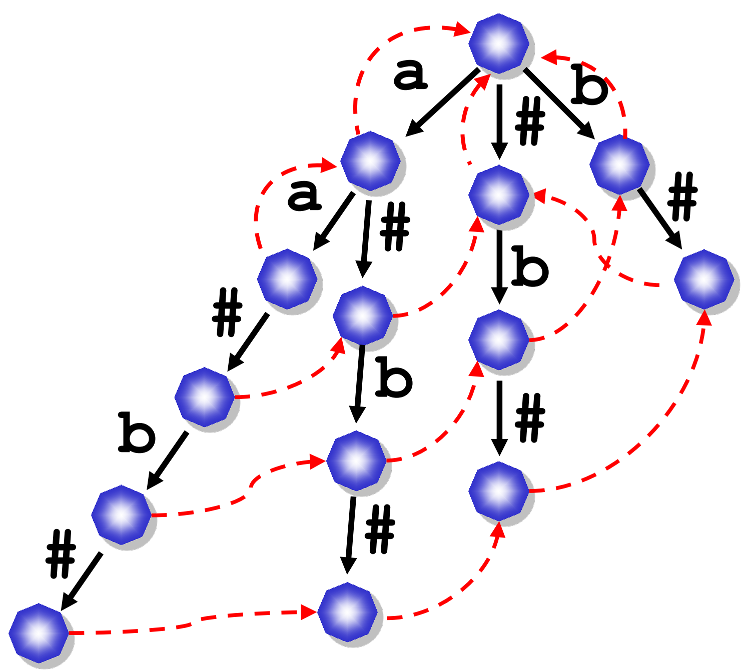
T **= `aa#b#`**



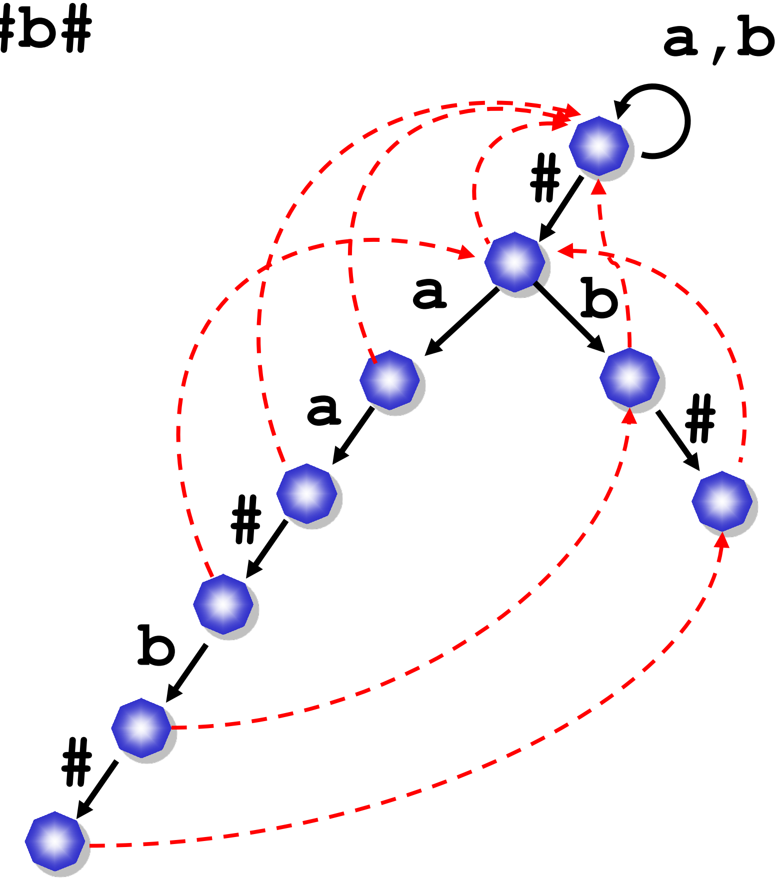
Word Suffix Trie

Suffix Links [cont.]

T = **aa#b#**



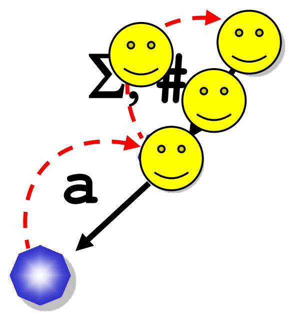
Suffix Trie

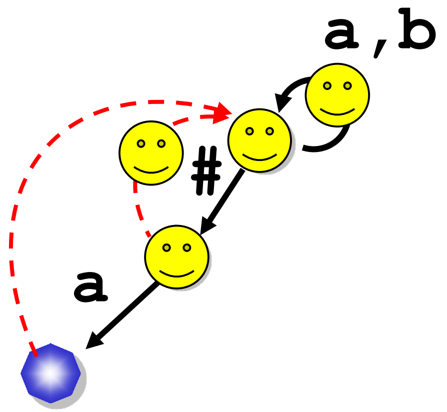


Word Suffix Trie

On-line Construction

T = `aa#b#`



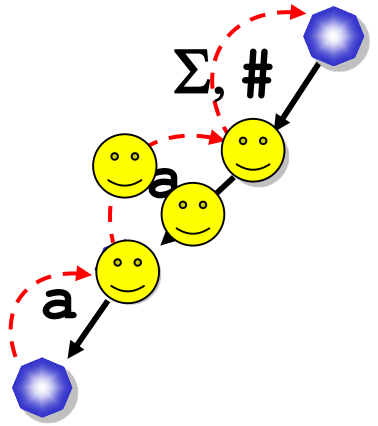


Suffix Trie

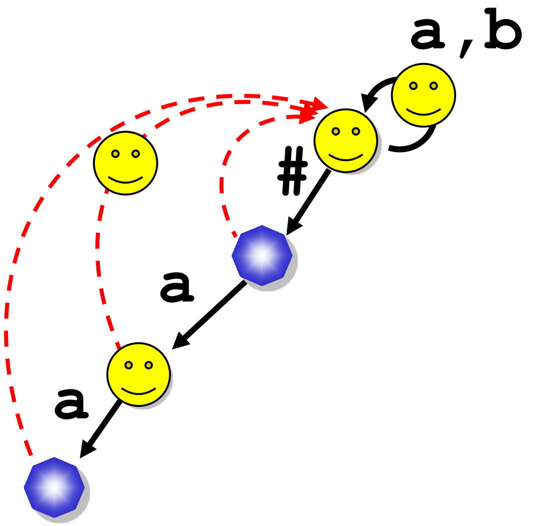
Word Suffix Trie

On-line Construction

T = **`aa#b#`**



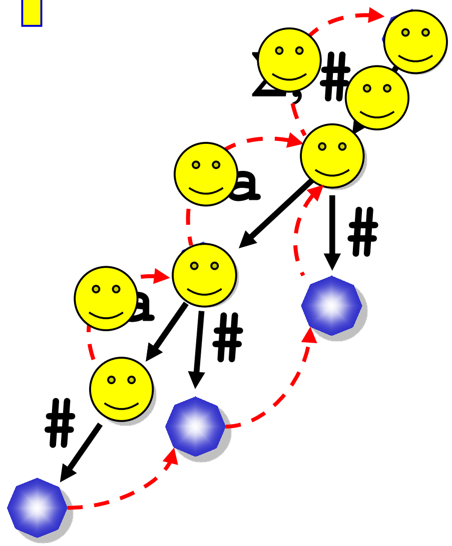
Suffix Trie

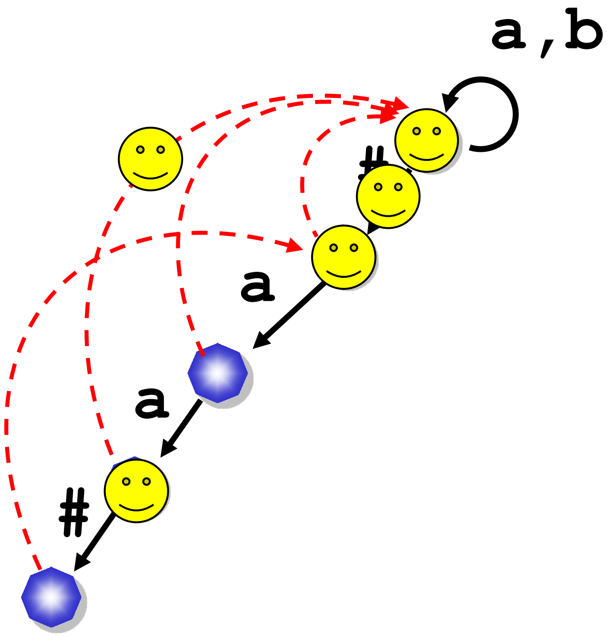


Word Suffix Trie

On-line Construction

T **=** **`aa#b#`**



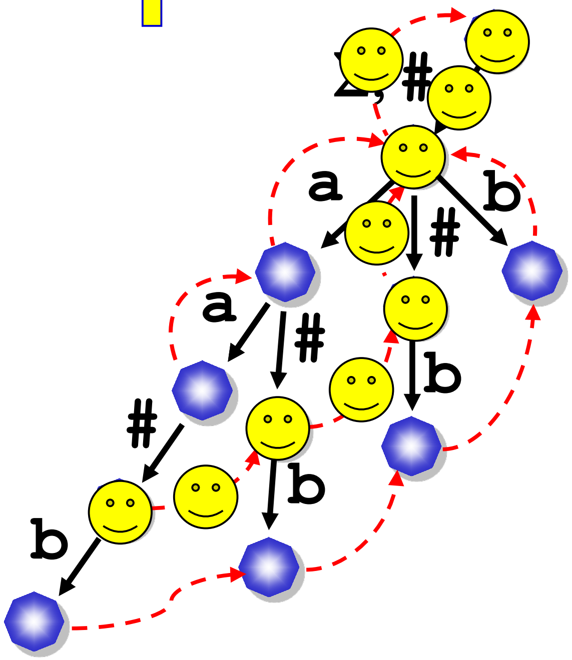


Suffix Trie

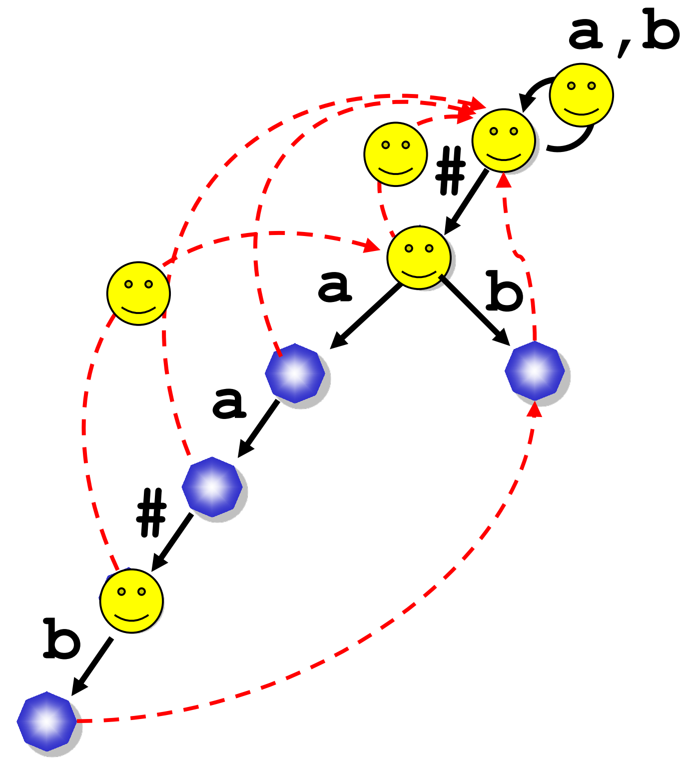
Word Suffix Trie

On-line Construction

T = **`aa#b#`**



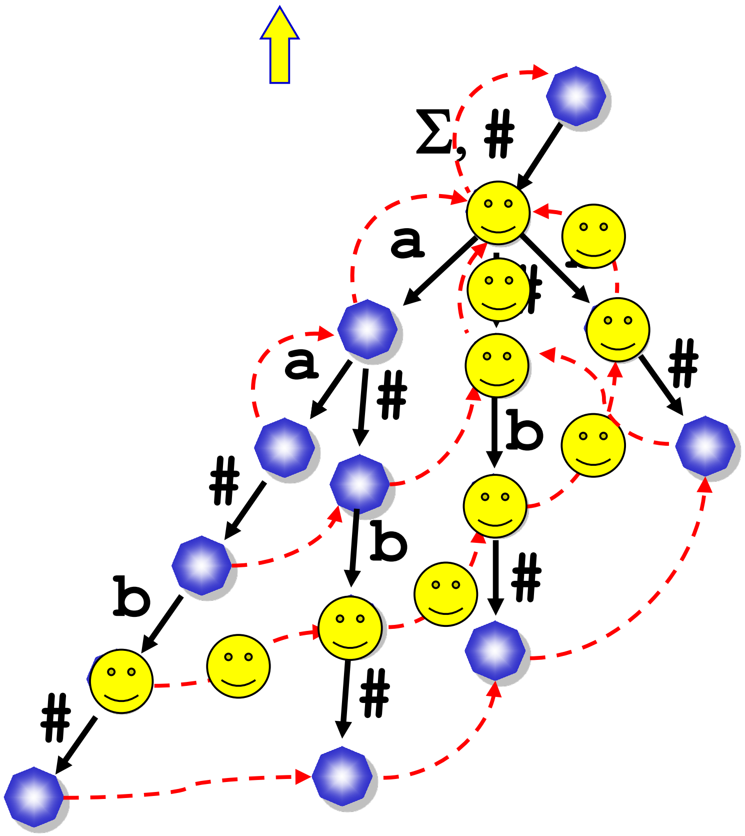
Suffix Trie



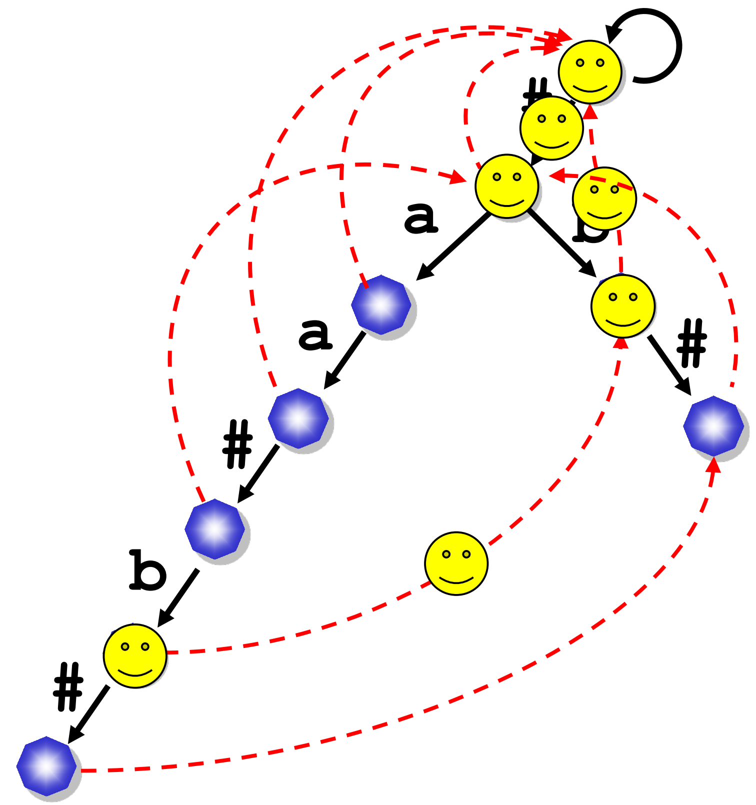
Word Suffix Trie

On-line Construction

T = `aa#b#`



Suffix Trie



Word Suffix Trie

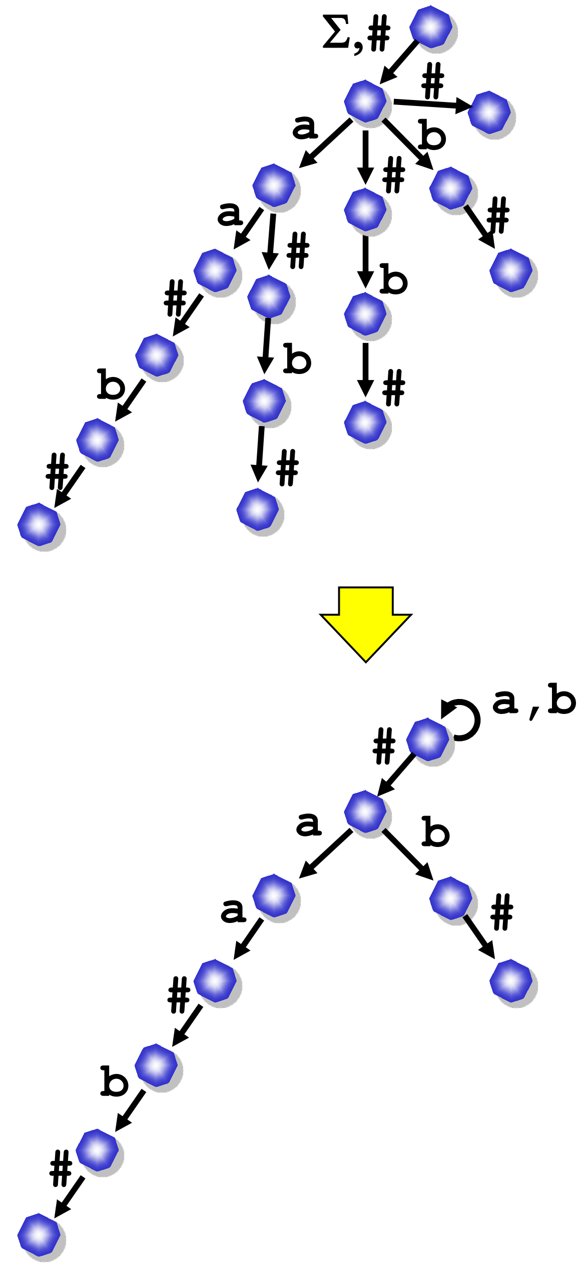
Pseudo Code

Just change here!!

Input: $w = w[1..n] \in D^+$ and auxiliary DFA M_D.
Output: Word suffix trie of w w.r.t. D.

{
    $root$ = the final state of M_D;
    $Suf(root)$ = the initial state of M_D;
    $top = root$;
    for $(i = 1;\ i \leq n;\ i++)$ $top = Update(top, w[i])$;
}

node $Update(top, c)$ {
    $newtop = CreateNewNode()$;
    create a new edge $top \xrightarrow{c} newtop$;
    $prev = newtop$;
    for $(t = Suf(top)$; no c-edge from t; $t = Suf(t))$ {
        $new = CreateNewNode()$;
        create a new edge $t \xrightarrow{c} new$;
        $Suf(prev) = new$;
        $prev = new$;
    }
    $Suf(prev)$ = the initial node of the c-edge from t;
    return $newtop$;
}



Drawback of Word Suffix Trie

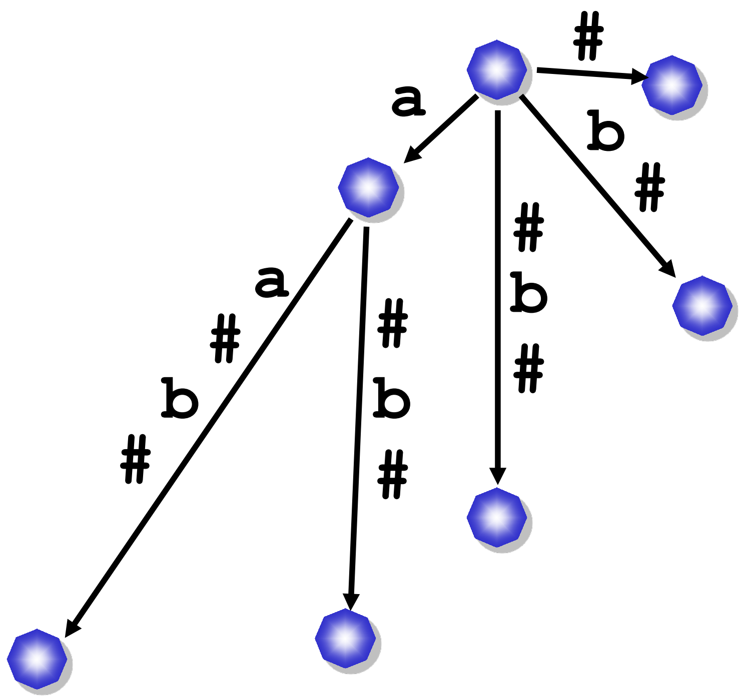
- Word suffix tries require $O(k|\boldsymbol{T}|)$ space.
- Andersson et al. introduced word suffix trees which can be implemented in $O(k)$ space.

Construction of Word Suffix Trees

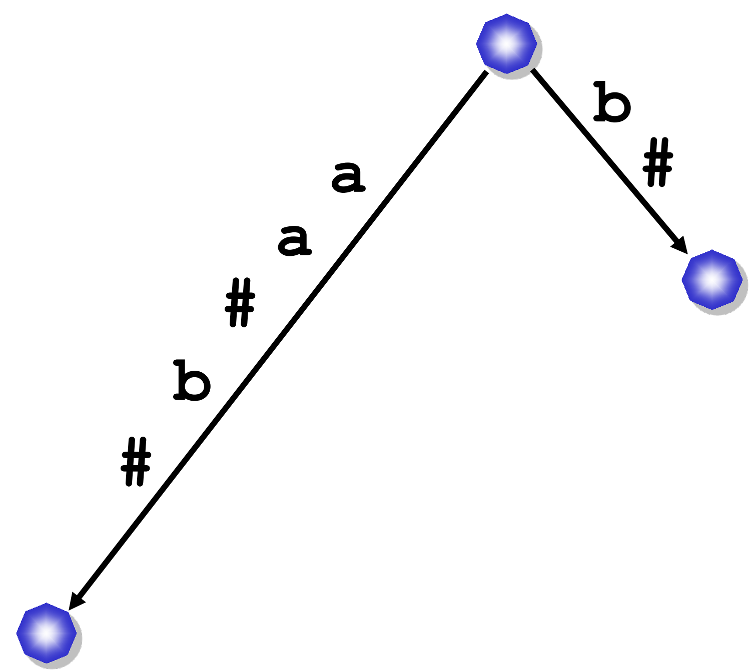
- Algorithm by Andersson et al.（1996）
  - for text $\boldsymbol{T} = \boldsymbol{T_1 T_2 \ldots T_k}$, constructs word suffix trees in $O(|\boldsymbol{T}|)$ expected time with $O(k)$ space.
- Our algorithm
  - simulates the on-line word suffix trie algorithm on word suffix trees.
  - runs in $O(|\boldsymbol{T}|)$ time in the worst cases, with $O(k)$ space.

Normal and Word Suffix Trees

T = **aa#b#**



Suffix Tree



Word Suffix Tree

Construction Algorithm

Just change here!!

Input: $w = w[1..n] \in D^+$ and auxiliary DFA M_D.
Output: Word suffix tree of $w[1..n]$ w.r.t. D.

{
/* *We assume* $\Sigma = \{w[-1], w[-2], \ldots, w[-m]\}$ */.
/* *Replace the edge labels of* M_D *with appropriate integer pairs* */.
$\boxed{\begin{array}{l} root = \text{the final state of } M_D; \\ Suf(root) = \text{the initial state of } M_D; \end{array}}$
$(s, k) = (root, 1)$;
for $(i = 1; i \le n; i{+}{+})$ {
$oldr =$ **nil**;
while $(CheckEndPoint(s, (k, i-1), w[i]) ==$ **false**$)$ {
if $(k \le i-1)$ /* $(s, (k, i-1))$ *is implicit.* */
$r = SplitEdge(s, (k, i-1))$;
else /* $(s, (k, i-1))$ *is explicit.* */
$r = s$;
$t = CreateNewNode()$;
create a new edge $r \xrightarrow{(i,\infty)} t$;
if $(oldr \ne$ **nil**$)$ $Suf(oldr) = r$;
$oldr = r$;
$(s, k) = Canonize(Suf(s), (k, i-1))$;
}
if $(oldr \ne$ **nil**$)$ $Suf(oldr) = s$;
$(s, k) = Canonize(s, (k, i))$;
}
}

boolean $CheckEndPoint(s, (k, p), c)$ {
if $(k \le p)$ { /* $(s, (k, p))$ *is implicit.* */
let $s \xrightarrow{(k',p')} s'$ be the $w[k]$-edge from s;
return $(c == w[k' + p - k + 1])$;
} **else return** (there is a c-edge from s);
}

(node,integer)-pair $Canonize(s, (k, p))$ {
if $(k > p)$ **return** (s, k); /* $(s, (k, p))$ *is explicit.* */
find the $w[k]$-edge $s \xrightarrow{(k',p')} s'$ from s;
while $(p' - k' \le p - k)$ {
$k \mathrel{+}= p' - k' + 1$; $s = s'$;
if $(k \le p)$ find the $w[k]$-edge $s \xrightarrow{(k',p')} s'$ from s;
}
return (s, k);
}

node $SplitEdge(s, (k, p))$ {
let $s \xrightarrow{(k',p')} s'$ be the $w[k]$-edge from s;
$r = CreateNewNode()$;
replace this edge by edges $s \xrightarrow{(k',k'+p-k)} r$ and $r \xrightarrow{(k'+p-k+1,p')} s'$;
return r;
}

Conclusions

- We first proposed an on-line word suffix trie construction algorithm.
  - The keys to the algorithm are *the minimal DFA accepting **D** and the re-defined suffix links*.
- Further, we introduced an on-line algorithm to build word suffix trees that works *with O(k) space and in O(|**T**|) time in the worst cases.*

Further Work

- **“Sparse Directed Acyclic Word Graphs”**
  by *Shunsuke Inenaga and Masayuki Takeda*
  Accepted to SPIRE’06

- **“Sparse Compact Directed Acyclic Word Graphs”**
  by *Shunsuke Inenaga and Masayuki Takeda*
  Accepted to PSC’06