3. Exact Matching : A Deeper Look At Classical Method

2015-04-14

Wonyeop Lee

Exact Matching With A Set of Patterns

- **Exact set matching problem**
  - A generalization of the exact matching problem
    - to find all occurrences of any pattern in text T
    - in a set of patterns $\mathcal{P} = \{P_1, P_2, \ldots, P_z\}$
  - It can be solved in $\boldsymbol{O(n + m + k)}$
    - where k is the number of occurrences in T of the patterns from $\mathcal{P}$
    - ***by using keyword tree***

Exact Matching With A Set of Patterns

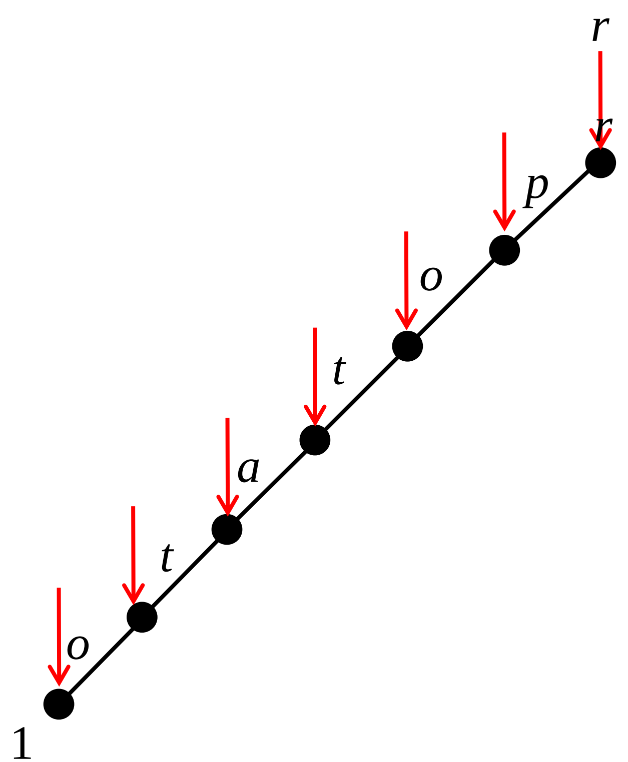
- **Definition of *keyword tree***

The keyword tree for set $\mathcal{P}$ is a rooted directed tree $\mathcal{K}$ satisfying three conditions:

1. Each edge is labeled with exactly one character
2. Any two edges out of the same node have distinct labels
3. Every pattern P_i in $\mathcal{P}$ maps to some node v of $\mathcal{K}$
    - Such that the characters on the path from the root of $\mathcal{K}$ to v exactly spell out P_i
    - Every leaf of $\mathcal{K}$ is mapped to by some pattern in $\mathcal{P}$

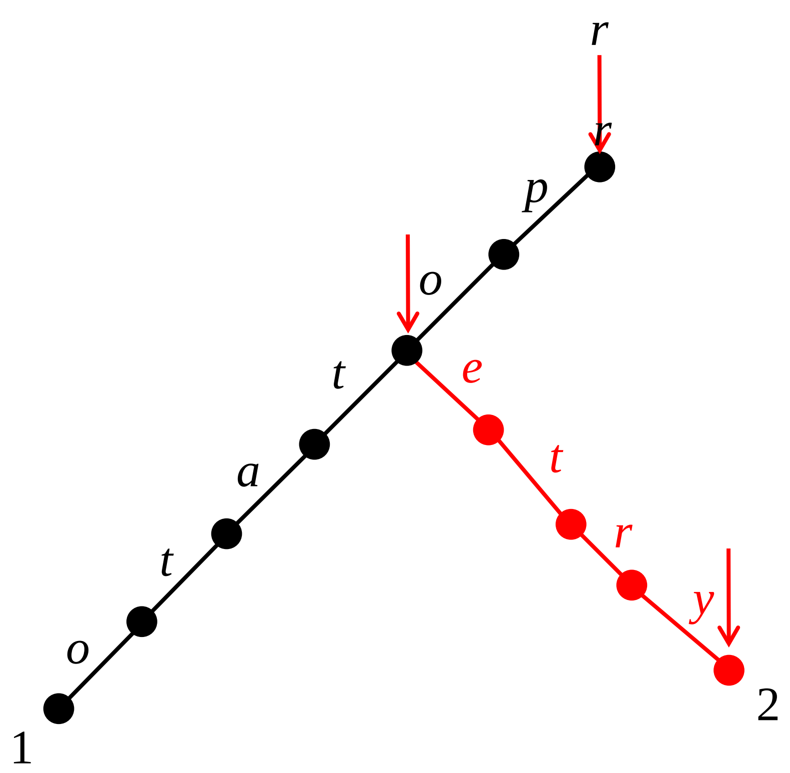
Exact Matching With A Set of Patterns

- **Example(**create ***keyword tree*** **)**
  - for the set of patterns { *potato* , *poetry*, *pot*, *pottery*, *science*, *school* }



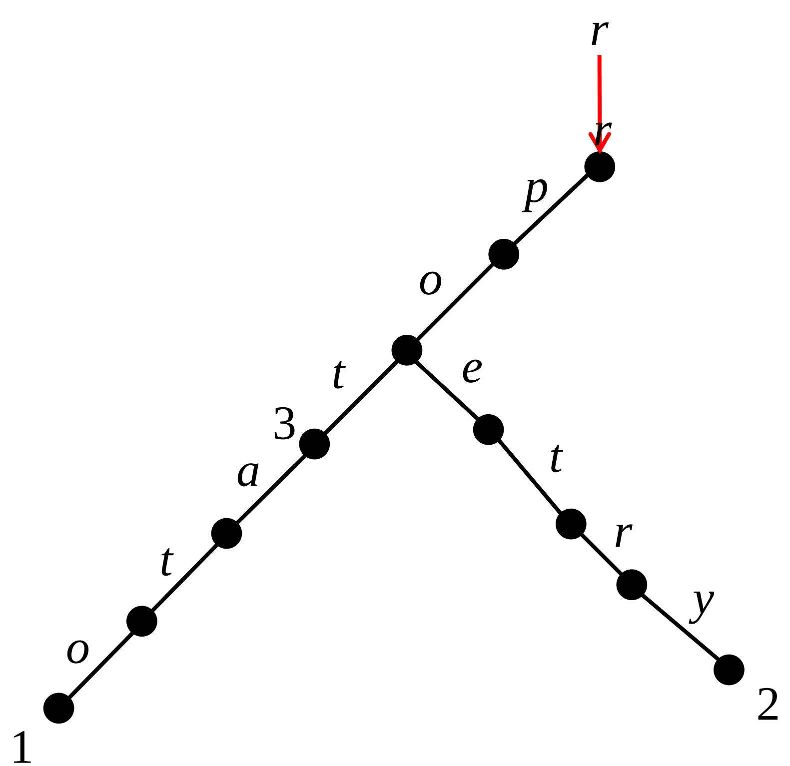
Exact Matching With A Set of Patterns

- **Example(**create ***keyword tree*** **)**
  - for the set of patterns { *potato*, *poetry*, *pot*, *pottery*, *science*, *school* }



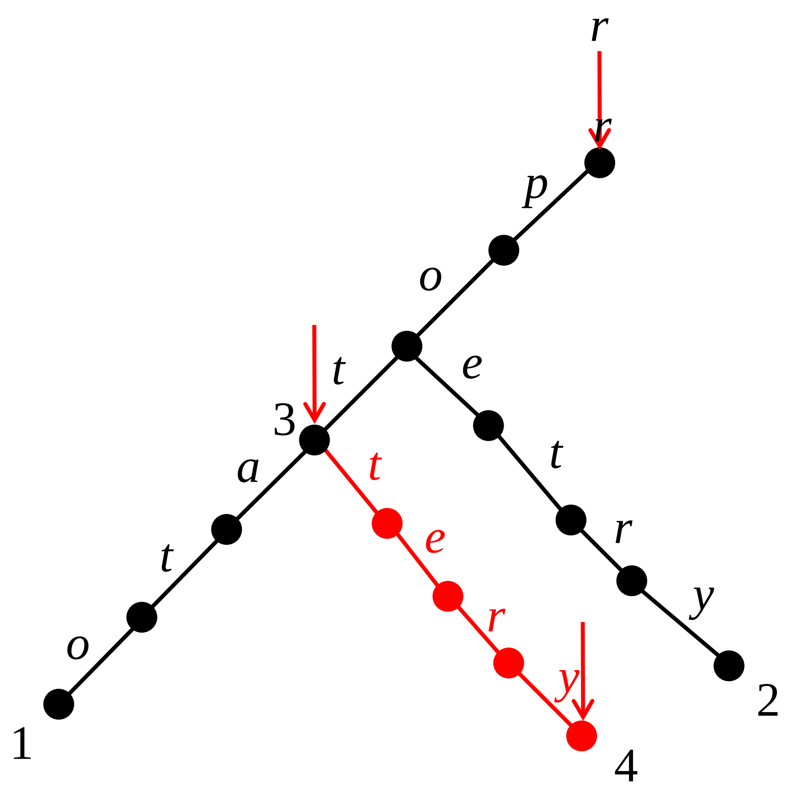
Exact Matching With A Set of Patterns

- **Example(**create ***keyword tree*** **)**
  - for the set of patterns { *potato*, *poetry*, *pot*, *pottery*, *science*, *school* }



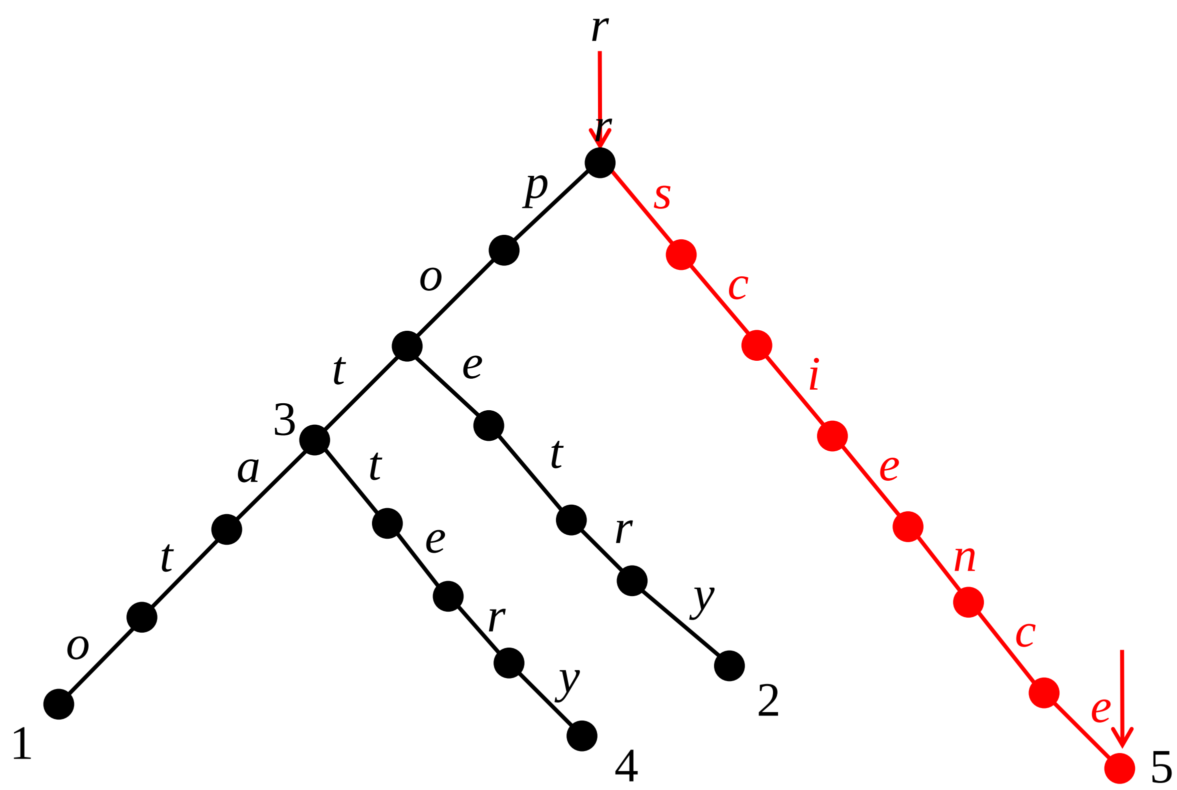
Exact Matching With A Set of Patterns

- **Example(**create ***keyword tree*** **)**
    - for the set of patterns { *potato*, *poetry*, *pot*, *pottery*, *science*, *school* }



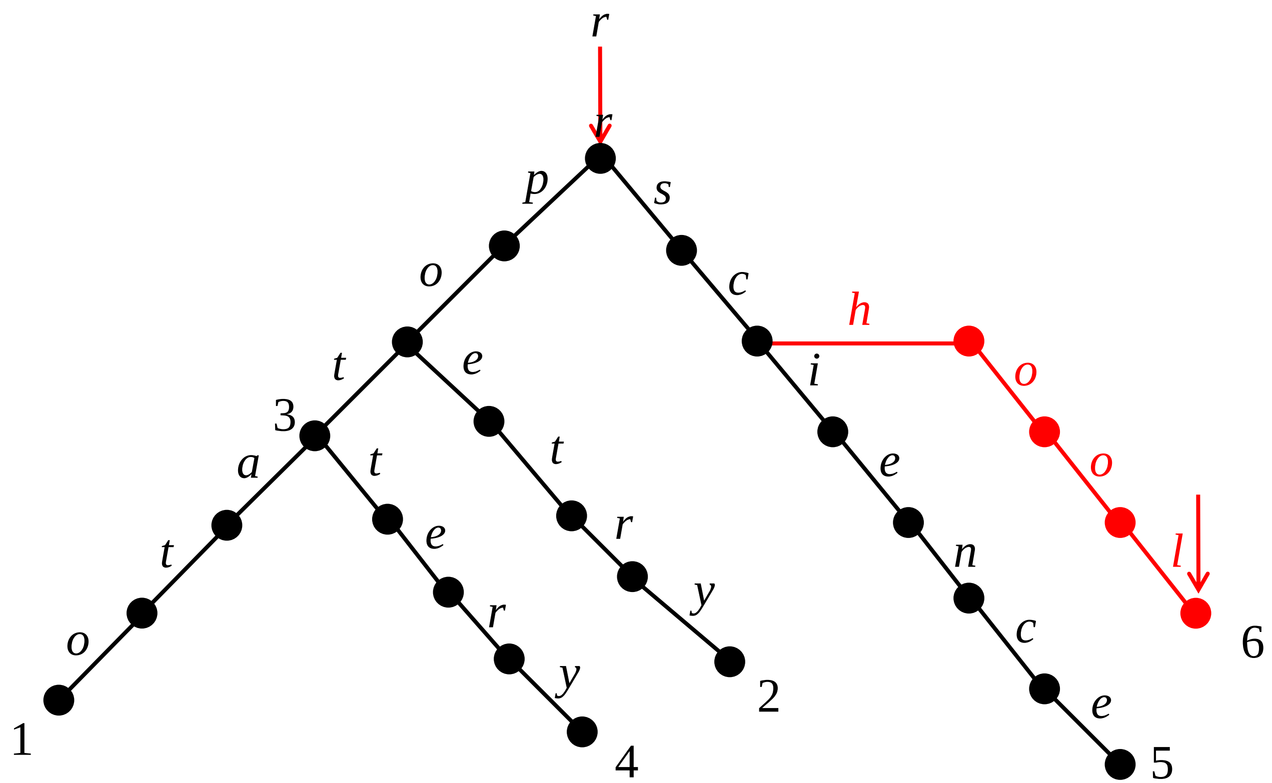
Exact Matching With A Set of Patterns

- **Example(**create ***keyword tree*** **)**
    - for the set of patterns { *potato*, *poetry*, *pot*, *pottery*, *science* , *school* }



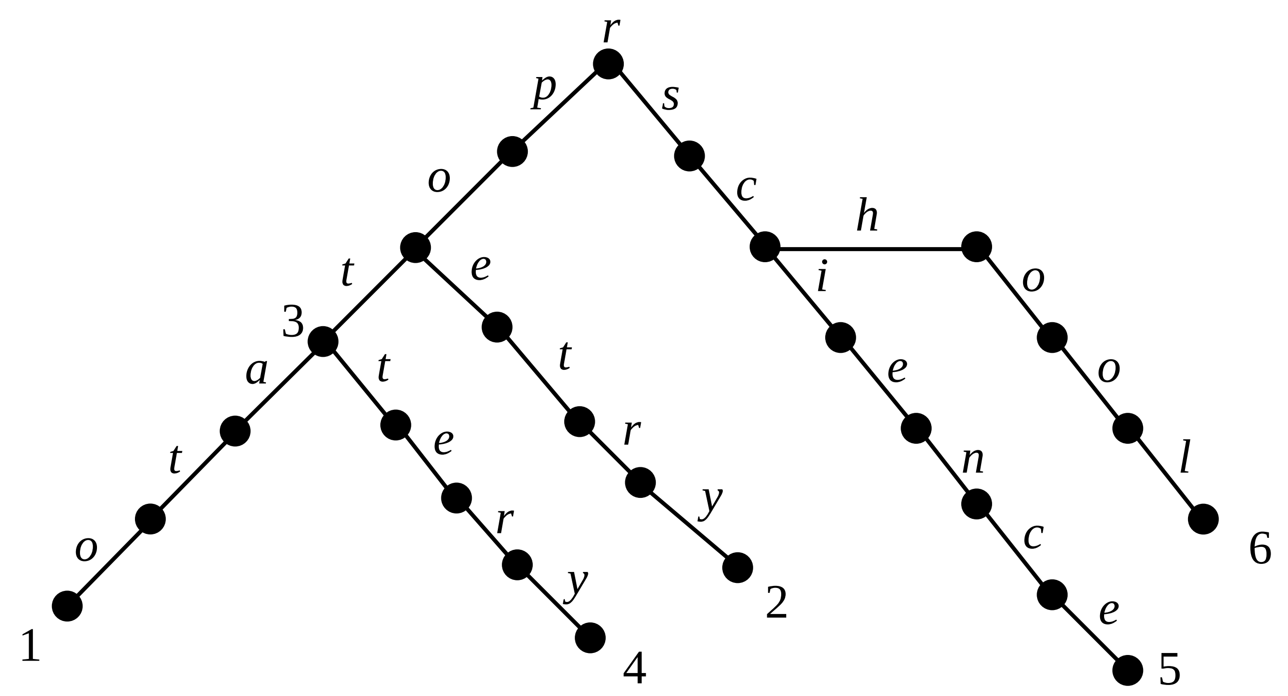
Exact Matching With A Set of Patterns

- **Example(**create ***keyword tree*** **)**
  - for the set of patterns { *potato*, *poetry*, *pot*, *pottery*, *science*, *school* }



Exact Matching With A Set of Patterns

- **Example of *keyword tree***
  - for the set of patterns { *potato*, *poetry*, *pot*, *pottery*, *science*, *school*}



Exact Matching With A Set of Patterns

- **Time complexity of construct keyword tree**
  - During the insertion of P_{i+1}, it takes $O(|P_{i+1}|)$ time
  - So the time to construct the entire keyword tree is $O(n)$

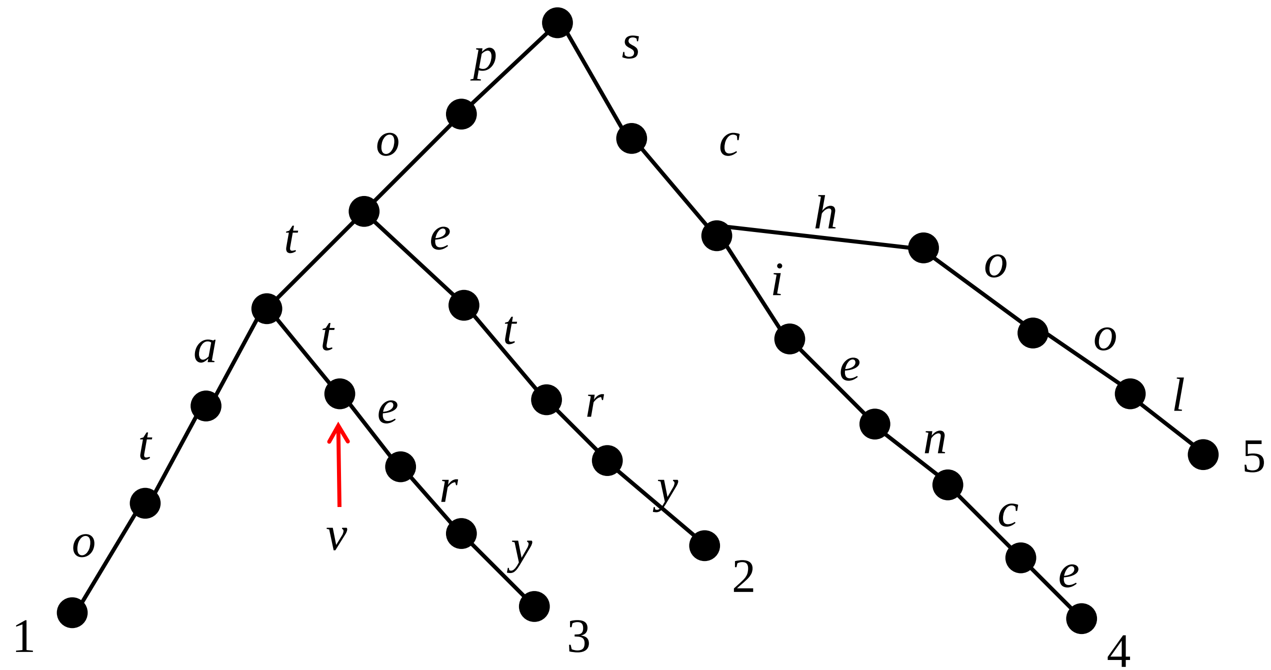
Aho-Corasick Algorithm

- **Assumption**
  - No pattern in $\mathcal{P}$ is a proper substring of any other pattern in $\mathcal{P}$
  - It is useful to solve the *exact set matching problem easily*

Aho-Corasick Algorithm

- **Definition of $\mathcal{L}(v)$**
  - The node pointed to by the arrow is labeled with the string *pott*

$\mathcal{L}(v) =$

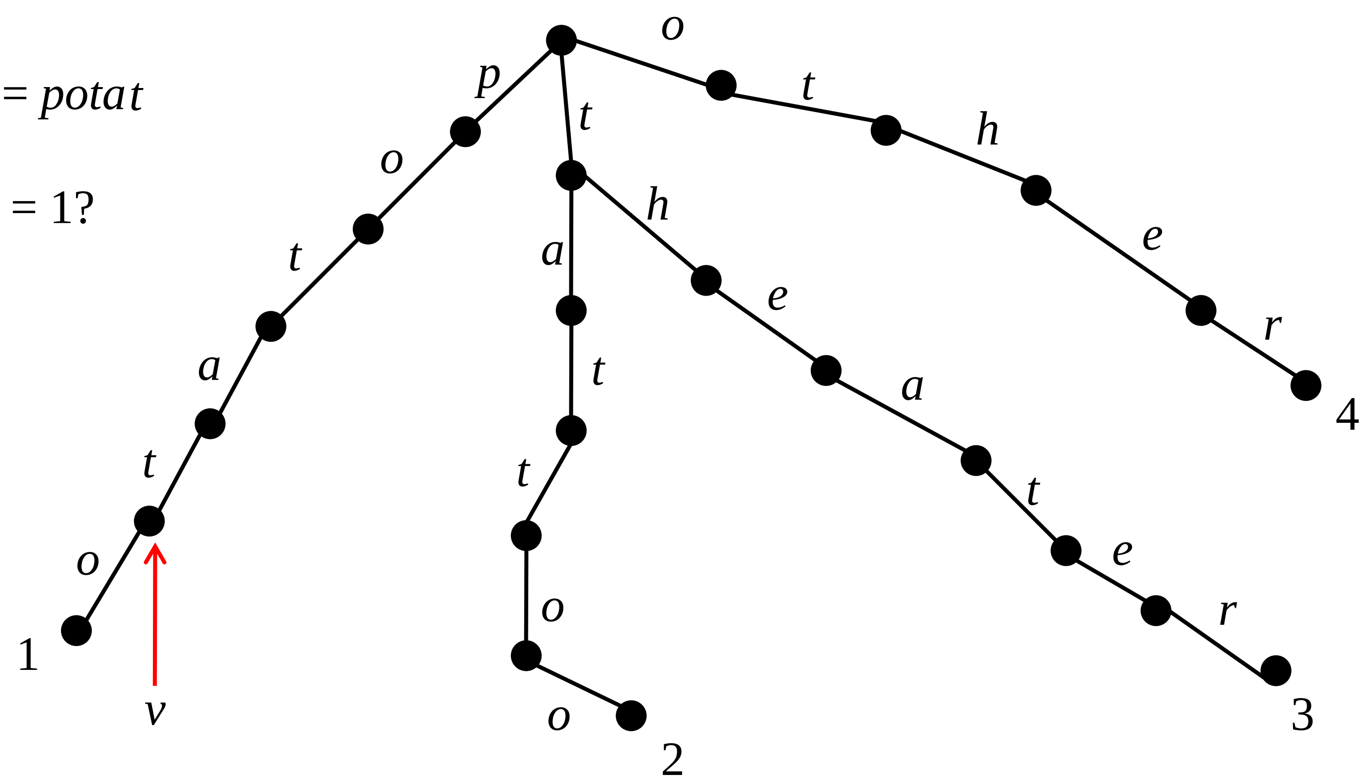


Aho-Corasick Algorithm

- **Definition of *lp*(*v*)**

$\mathcal{L}(v) = potat$

$lp(v) = 1?$

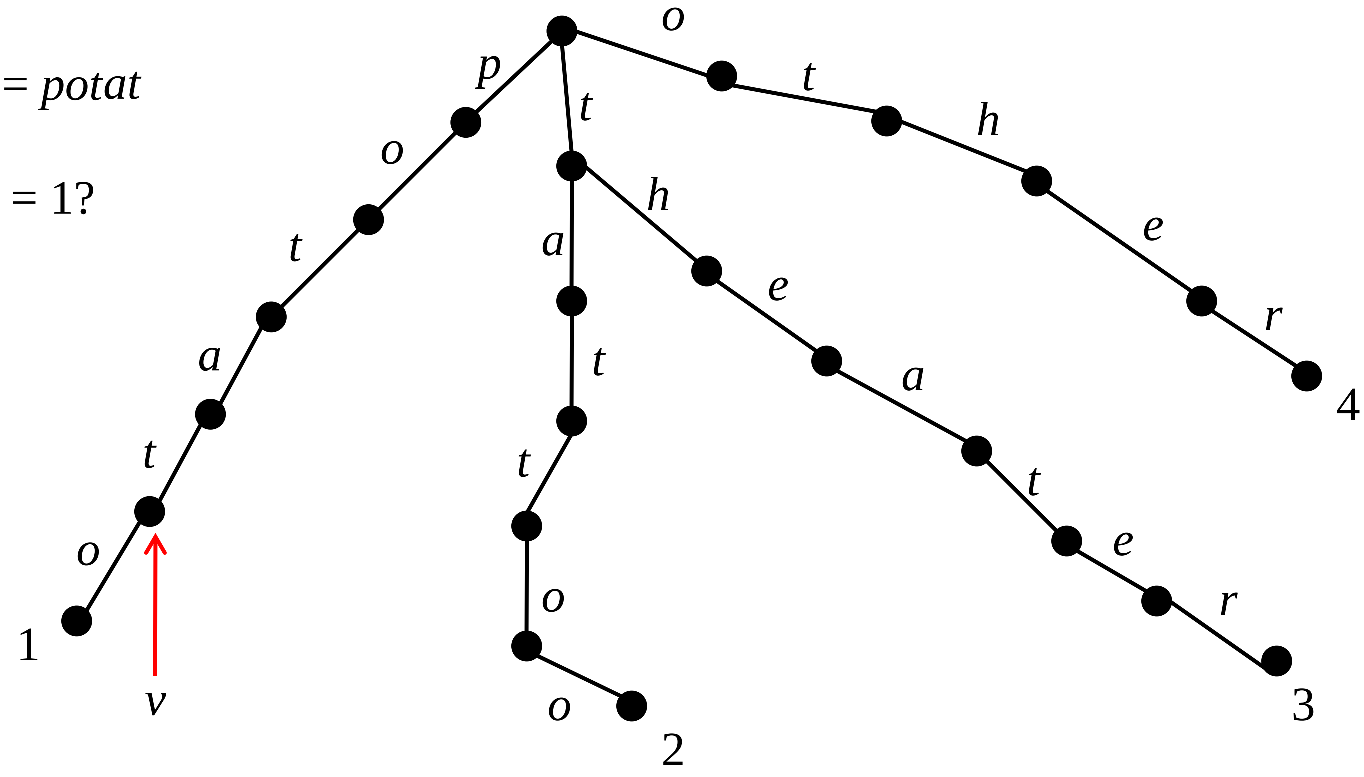


Aho-Corasick Algorithm

- **Definition of *lp*(v)**

$\mathcal{L}(v) = potat$

$lp(v) = 1$?

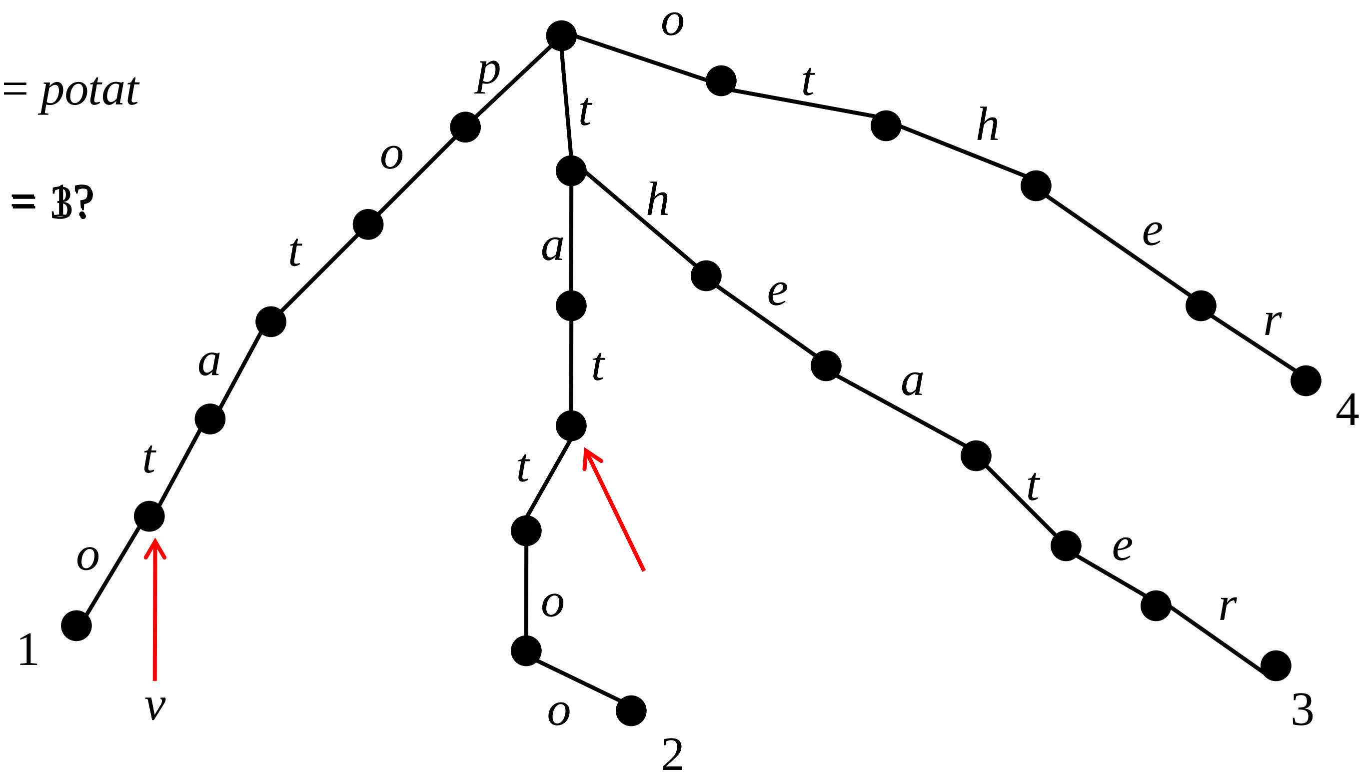


Aho-Corasick Algorithm

- **Definition of *lp*(*v*)**

$\mathcal{L}(v)$ = *potat*

lp(*v*) = 3?

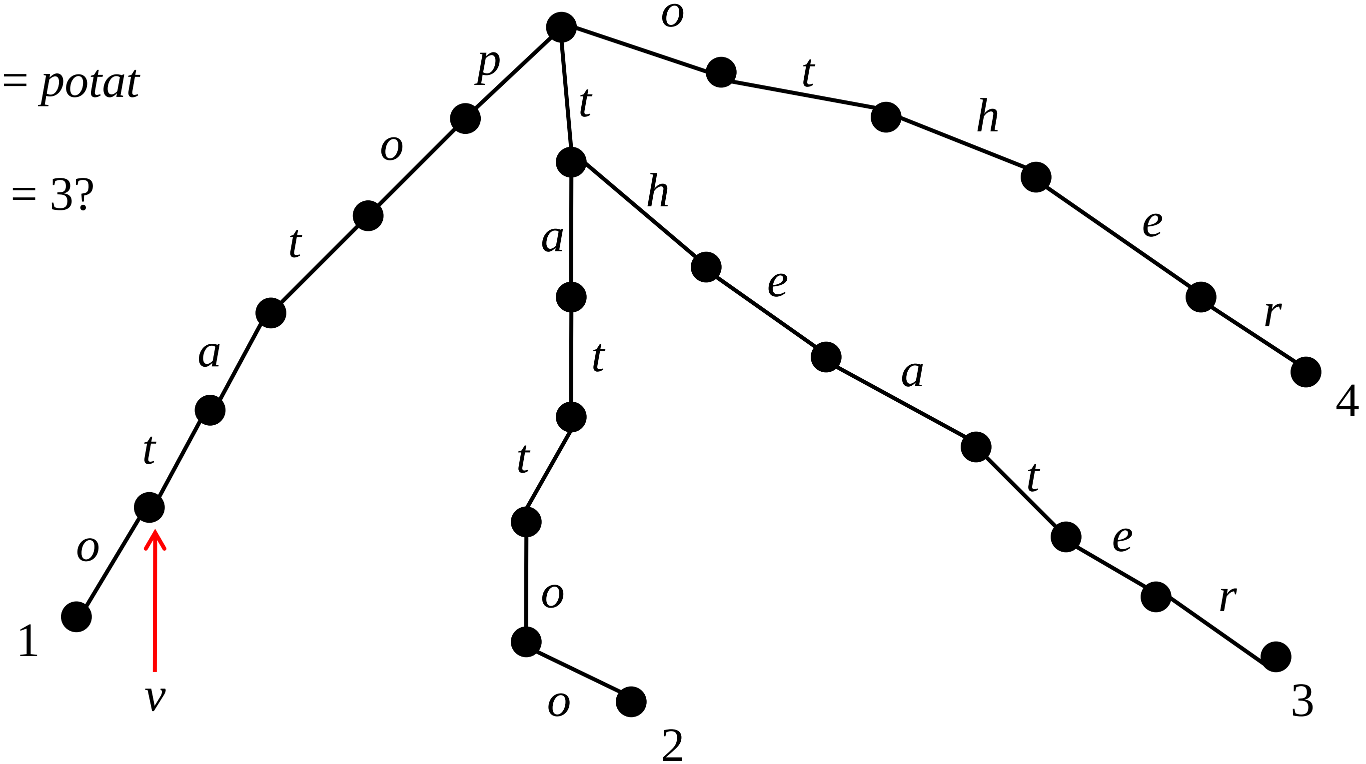


Aho-Corasick Algorithm

- **Definition of *lp*(*v*)**

$\mathcal{L}(v)$ = *potat*

$lp(v) = 3$?

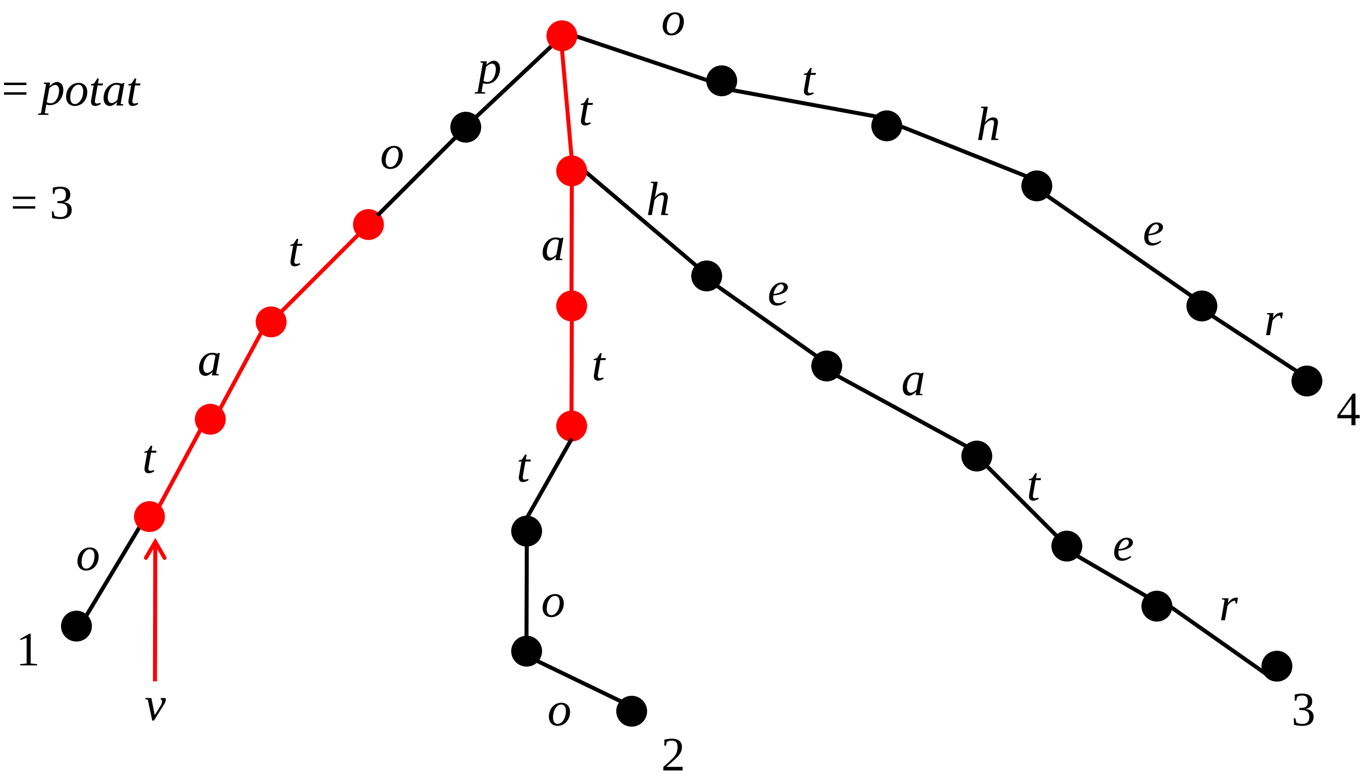


Aho-Corasick Algorithm

- **Definition of *lp*(*v*)**

$\mathcal{L}(v) = potat$

$lp(v) = 3$



Aho-Corasick Algorithm

- **Definition**

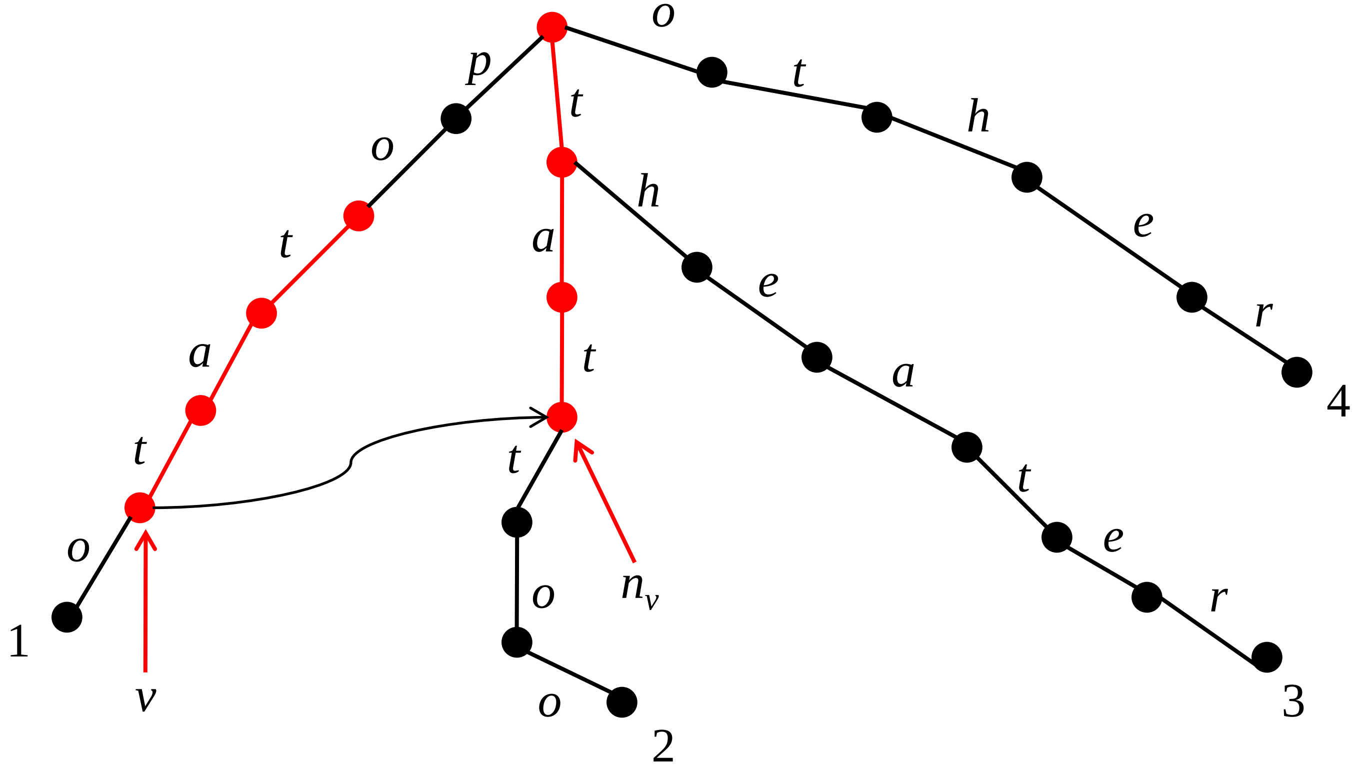
  For a node v of $\mathcal{K}$, let n_v be the unique node in $\mathcal{K}$ labeled with the suffix of $\mathcal{L}(v)$ of length $lp(v)$. When $lp(v) = 0$ then n_v is the root of $\mathcal{K}$

- **Definition**

  We call the order pair (v, n_v) a **failure link**

Aho-Corasick Algorithm

- **Example**

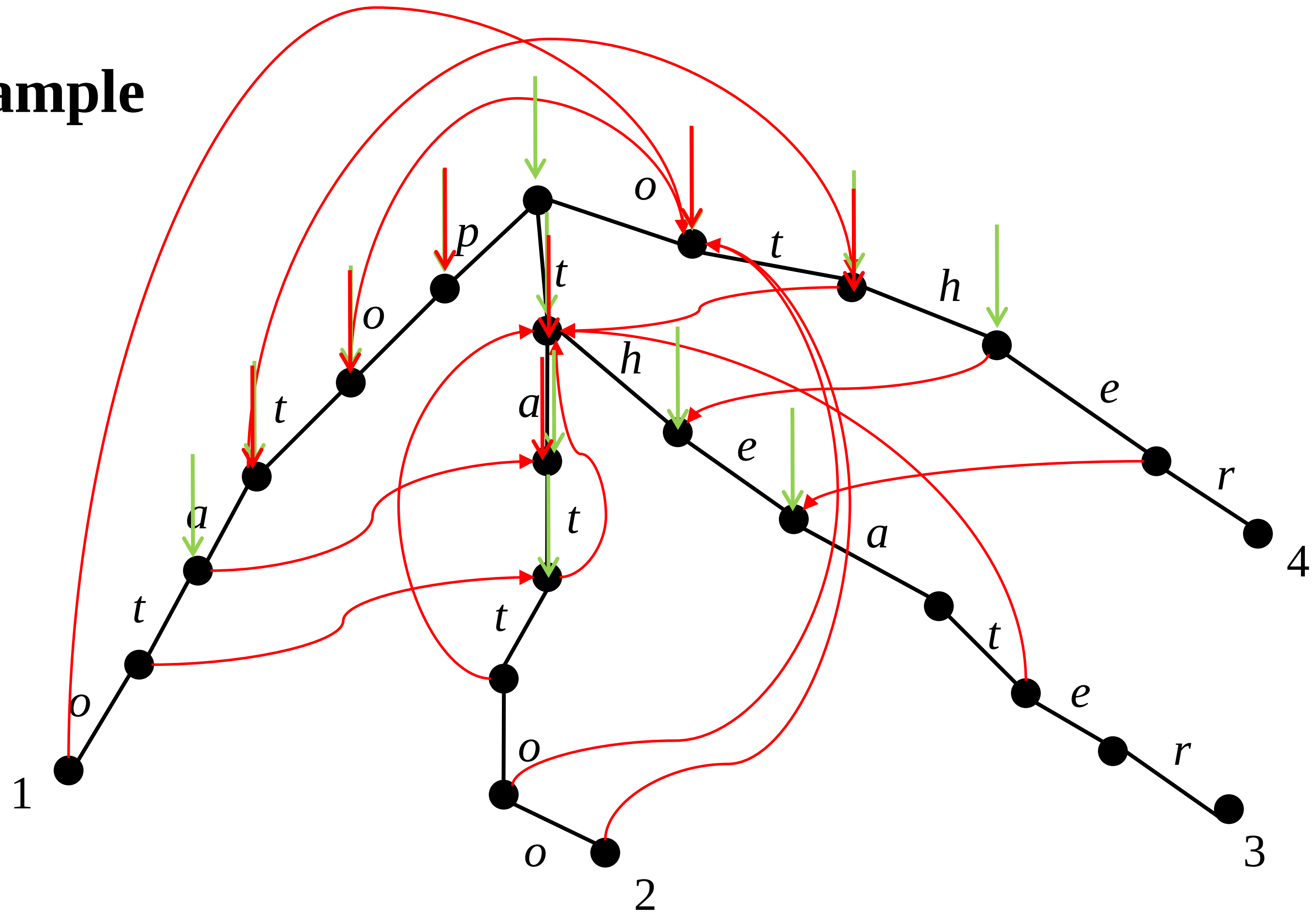


Aho-Corasick Algorithm

- **Algorithm n_v**
  - To find n_v, for every node v, repeatedly apply the algorithm to the nodes in $\mathcal{K}$ in a **breadth-first manner** starting at the root

Aho-Corasick Algorithm

- **Example**



Aho-Corasick Algorithm

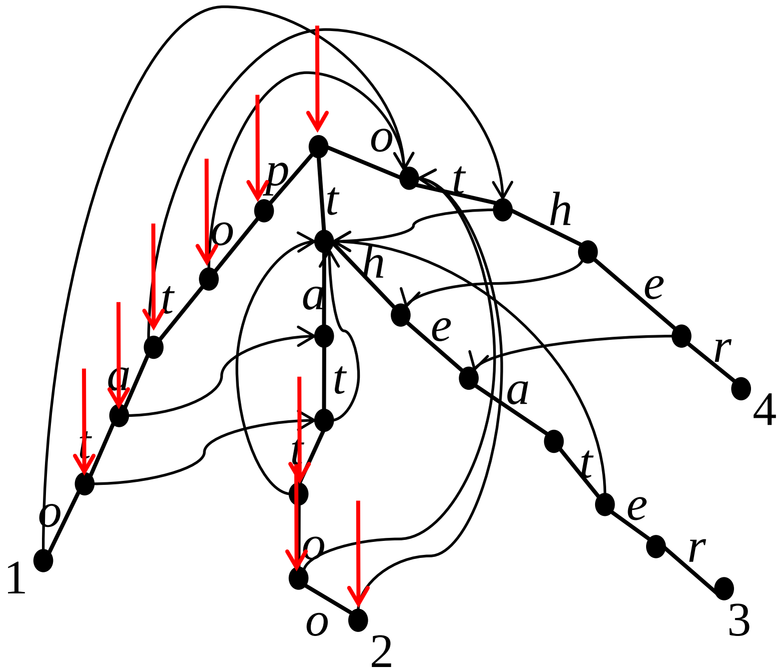
- **Theorem 3.4.1**
  - Let n be the total length of all the patterns in $\mathcal{P}$.
  - The total time used by Algorithm n_v when applied to all nodes in $\mathcal{K}$ is $O(n)$
    - because we find failure link at once for each node
    - and traverse of failure link for parents at most length of pattern -1

Aho-Corasick Algorithm

- **Algorithm AC search**
  - *l* indicates the starting position of the patterns in *T*
  - Point *c* into *T* indicates the current character of *T*
    - to be compared with a character on $\mathcal{K}$

Aho-Corasick Algorithm

- **Algorithm AC search**



1	2	3	4	5	6	7	8	9	10	11	12
x	*x*	*p*	*o*	*t*	*a*	*t*	*t*	*o*	*o*	*x*	*x*

$l = 3$ $l = 5$ $c = 8$ $c = 11$

$l = 10$

Report that P_1 occurs in T starting at 5

$lp(v) = 3$, $l = 8 - 3 = 5$

$lp(v) = 1$, $l = 11 - 1 = 10$

Aho-Corasick Algorithm

- **Algorithm AC search**
  - Search time is $O(m)$
    - because no back jump occurs during the search

Aho-Corasick Algorithm

- **The full AC algorithm : relaxing the substring assumption**

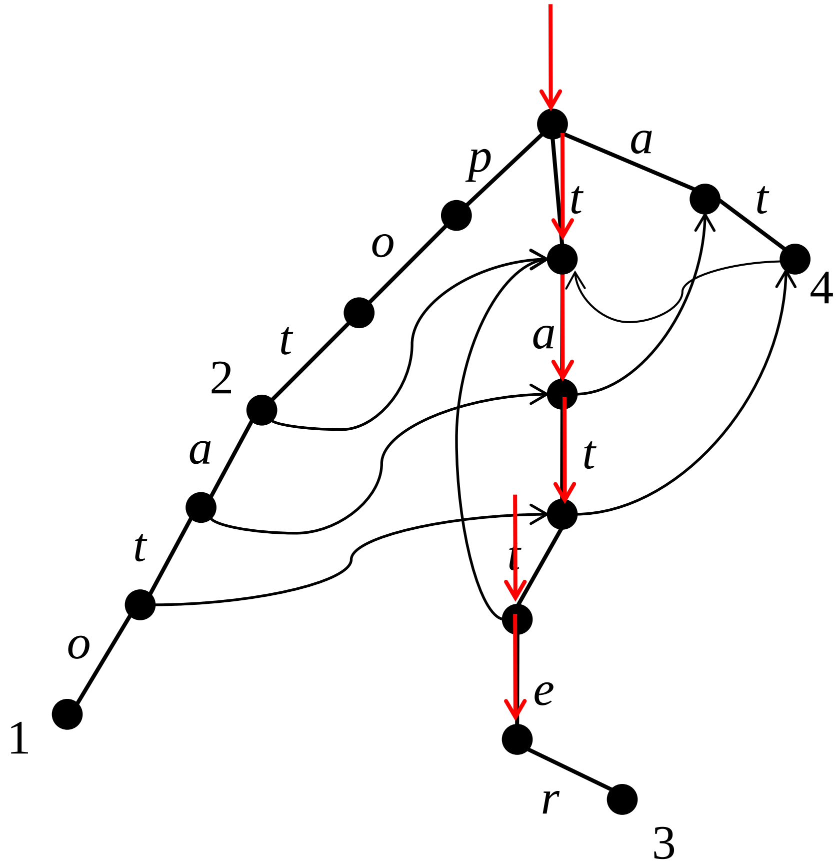
  - Until now we have assumed that no pattern in $\mathcal{P}$ is a substring of another pattern in $\mathcal{P}$

  - **If one pattern is a substring of another**, and yet *Algorithm AC search* uses the same keyword tree as before, then the algorithm has a problem

Aho-Corasick Algorithm

- **The full AC algorithm : relaxing the substring assumption**
  - $\mathcal{P}$ = { *potato*, *pot*, *tatter*, *at* }
  - *T* = *x t a t t e x*

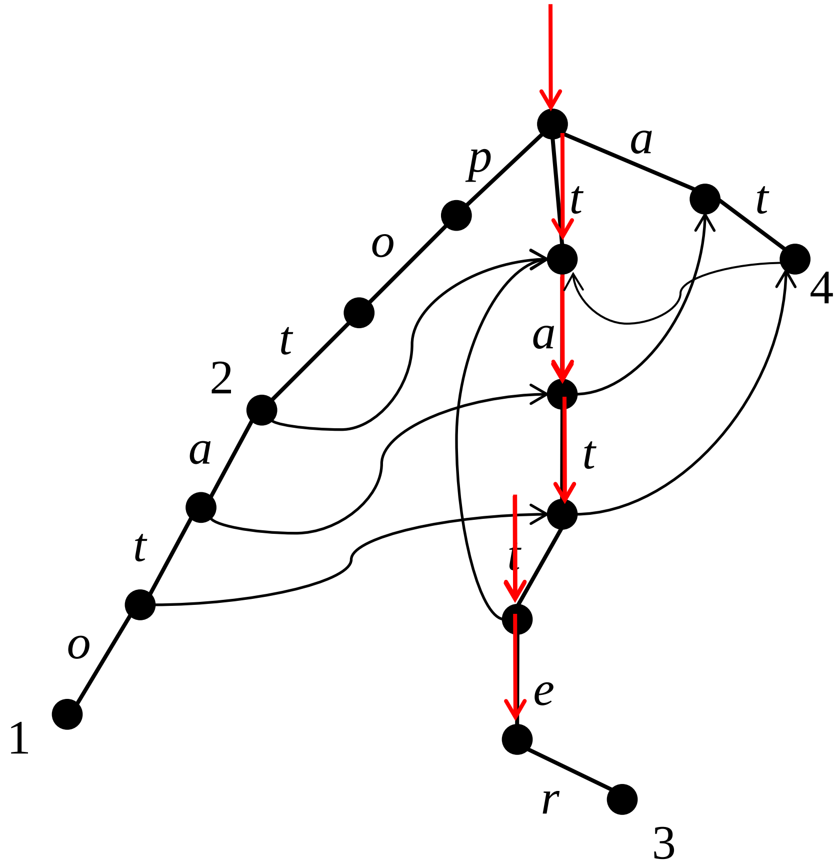
Pattern '*at*' occurs in text!
But, does not reported!



Aho-Corasick Algorithm

- **The full AC algorithm : relaxing the substring assumption**
  - $\mathcal{P}$ = { *potato*, *pot*, *tatter*, *at* }
  - $T = x\ t\ a\ t\ t\ e\ x$

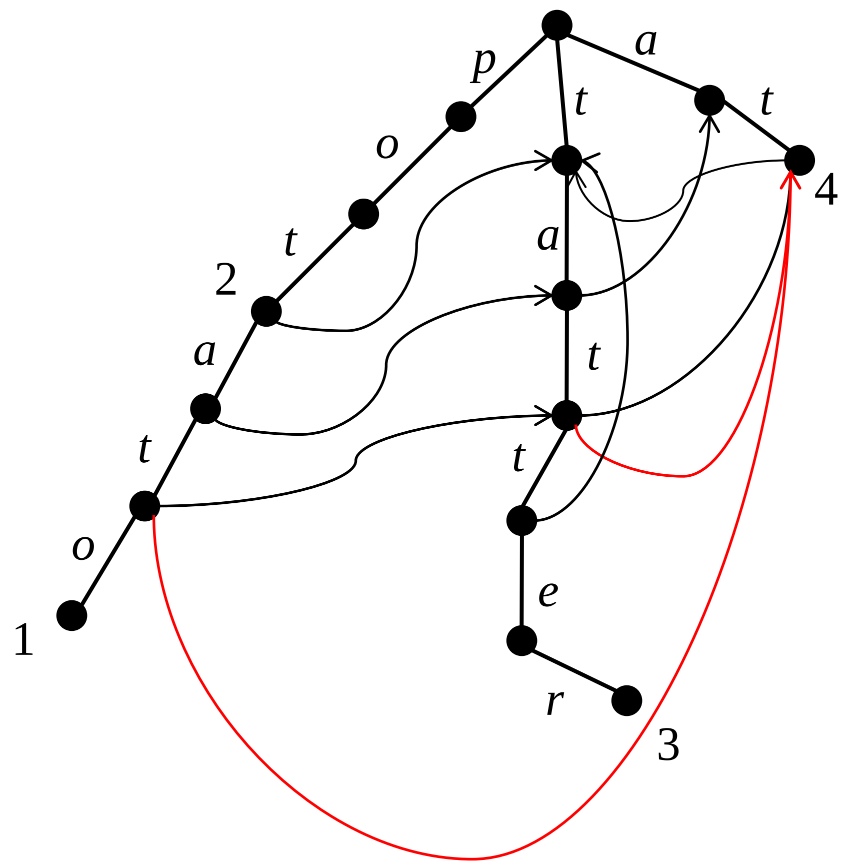
Report that pattern
'*at*' occurs in text



Aho-Corasick Algorithm

- **Lemma 3.4.2**
  - Suppose, in a keyword tree $\mathcal{K}$, there is a directed path of failure links from a node *v* to a node that is **numbered** with pattern *i*.
  - Then pattern P_i must **occur in *T*** ending at position *c* (the current character) whenever node *v* is reached during the search phase of the *Aho-Corasick algorithm*

Aho-Corasick Algorithm



Aho-Corasick Algorithm

- **Lemma 3.4.3**
  - Suppose a node v has been reached during the algorithm.
  - Then pattern P_i occurs in T ending at position c only if v is numbered i or there is a directed path of failure links from v to the node numbered i

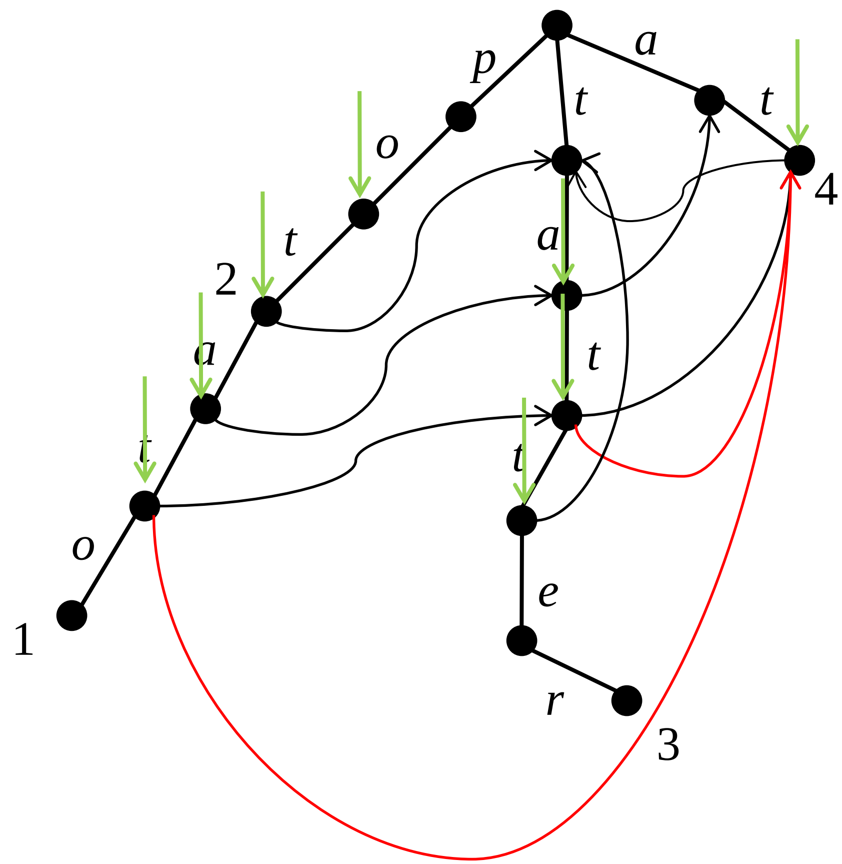
Aho-Corasick Algorithm

- **Output link**
  - The output link at a node v points to that numbered node other than v that is reachable from v by the fewest failure links
  - The output links can be determined in $O(n)$ time
    - during the running of the preprocessing algorithm n_v

Aho-Corasick Algorithm

- **Output link**
  - When the n_v value is determined, the possible output link from node v is determined
    - If n_v is numbered node then the output link from v point to n_v
    - If n_v is not numbered but has an output links to a node w, then the output link from v points to w
    - Otherwise v has no output link

Aho-Corasick Algorithm



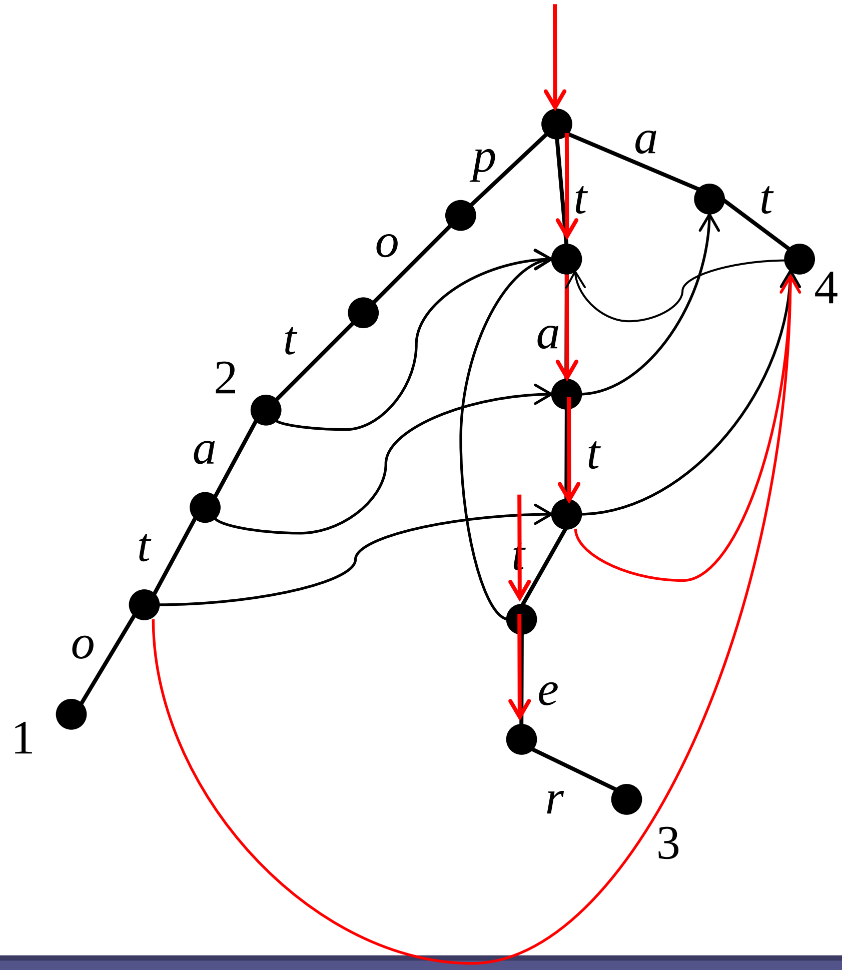
Aho-Corasick Algorithm

- **Output link**
  - An output link points only to a numbered node
  - With the output links, all occurrences in T of patterns of $\mathcal{P}$ can be detected in $O(m + k)$ time

Aho-Corasick Algorithm

- **The full AC algorithm : relaxing the substring assumption**
  - $\mathcal{P}$ = { *potato*, *pot*, *tatter*, *at* }
  - $T = x\ t\ a\ t\ t\ e\ x$

Report that pattern
'*at*' occurs in text



Aho-Corasick Algorithm

- **Time complexity of Algorithm full AC search**
  - It takes $O(n)$ time to construct a keyword tree.
  - And the preprocessing task to find all failure links and output links takes $O(n)$ time
  - The time complexity to execute the full AC search algorithm is $O(m + k)$ time
  - So the entire time complexity for the full AC search algorithm is **O(n + *m* + *k*) time**

Aho-Corasick Algorithm

Thanks