Faculty of Computer Science
Data Science and Business Analytics (DSBA)

Algorithms and Data Structures

Seminar 2 – Module 2

Aho-Corasick

October 2021

J.C. Carrasquel

The Problem

Given a text and multiple patterns,
find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

"sin" found in position 0

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

"sing" found in position 0

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

"in" found in position 1

Idea:
Try to search the patterns in the text ***in parallel***

The Problem

Given a text and multiple patterns, find all the patterns in the text.

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

text:

s	i	n	g	a	s	o	n	a	r

Idea:
Try to search the patterns in the text ***in parallel***

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

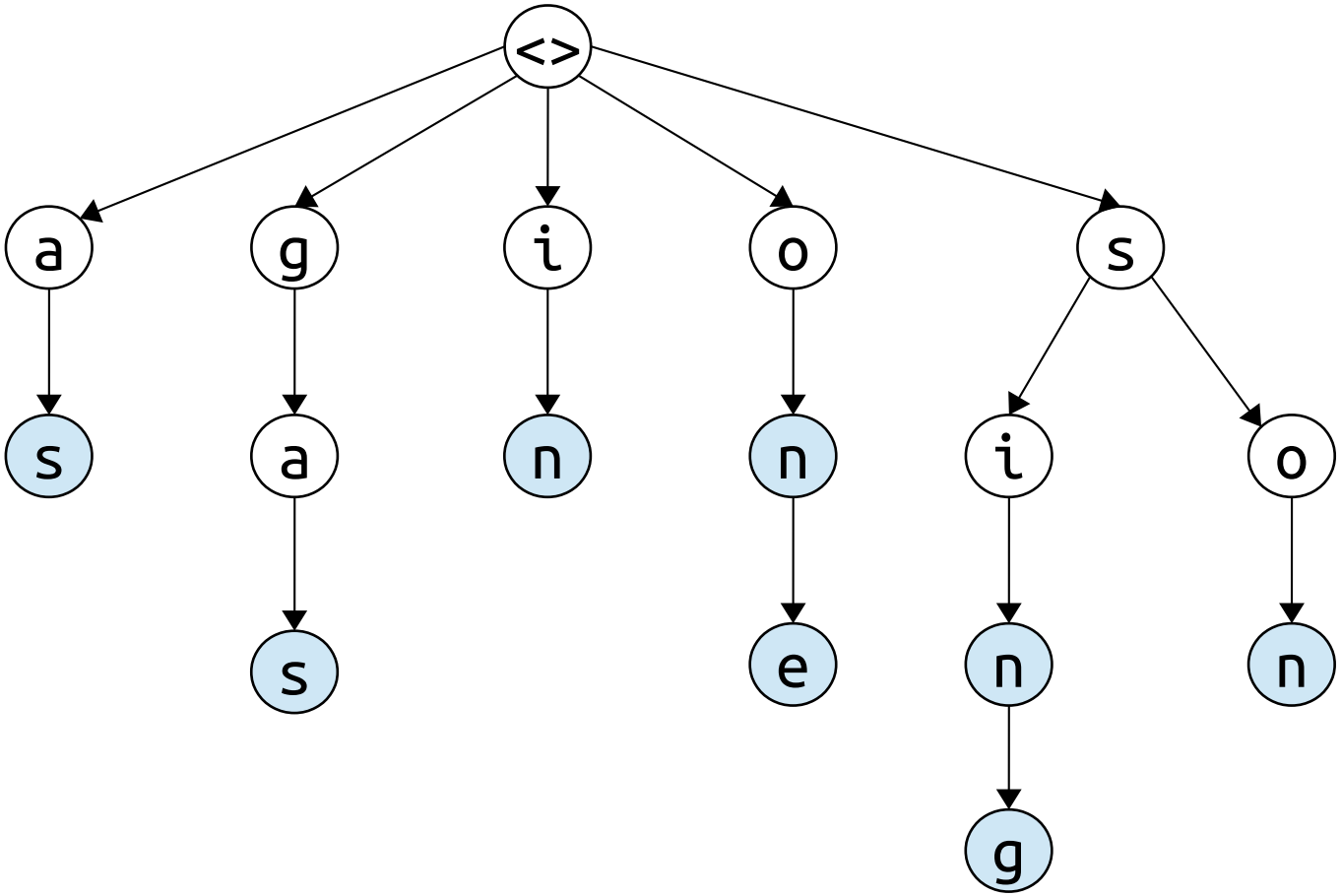
a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	

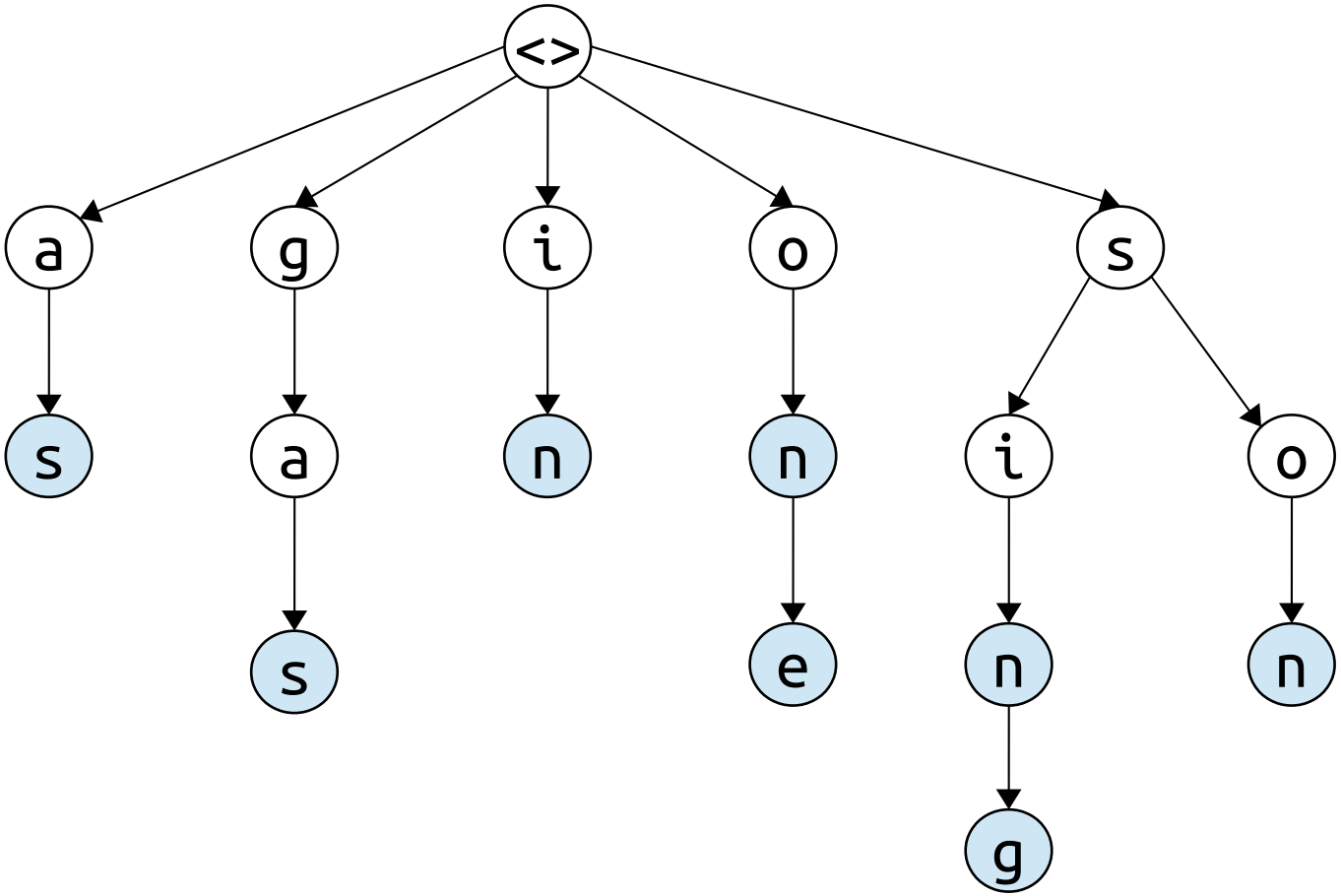


Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



text:

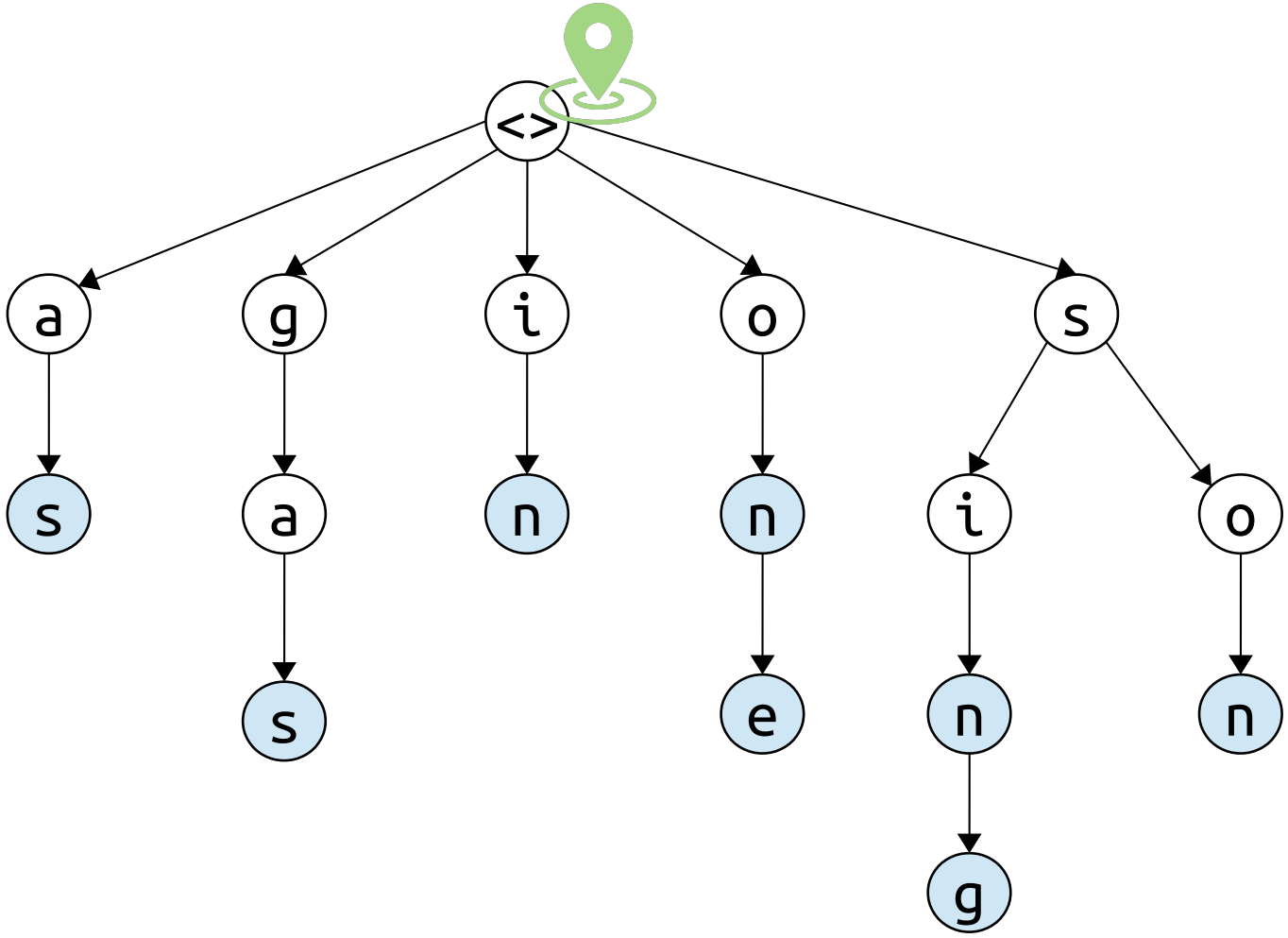
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



text:

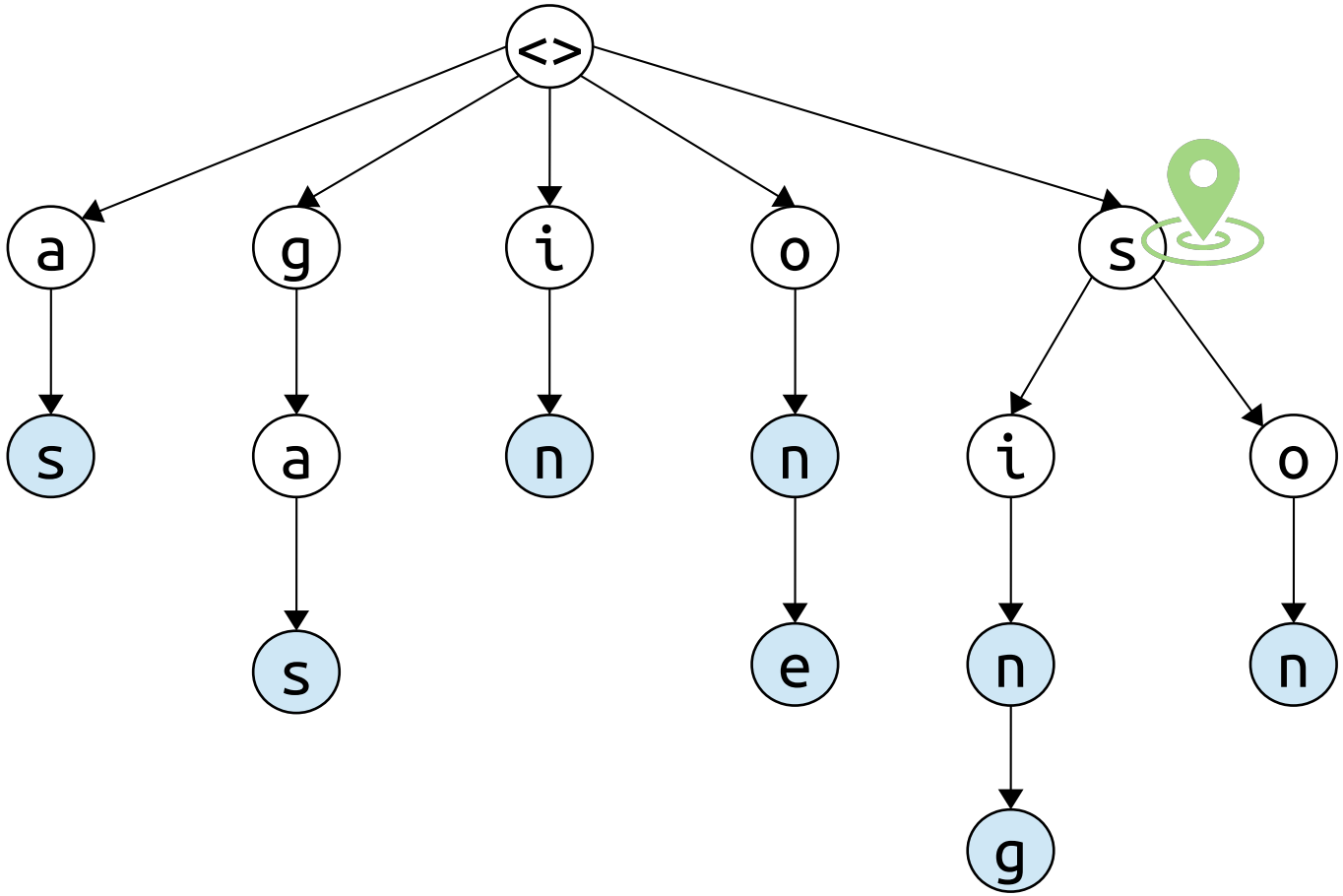
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



text:

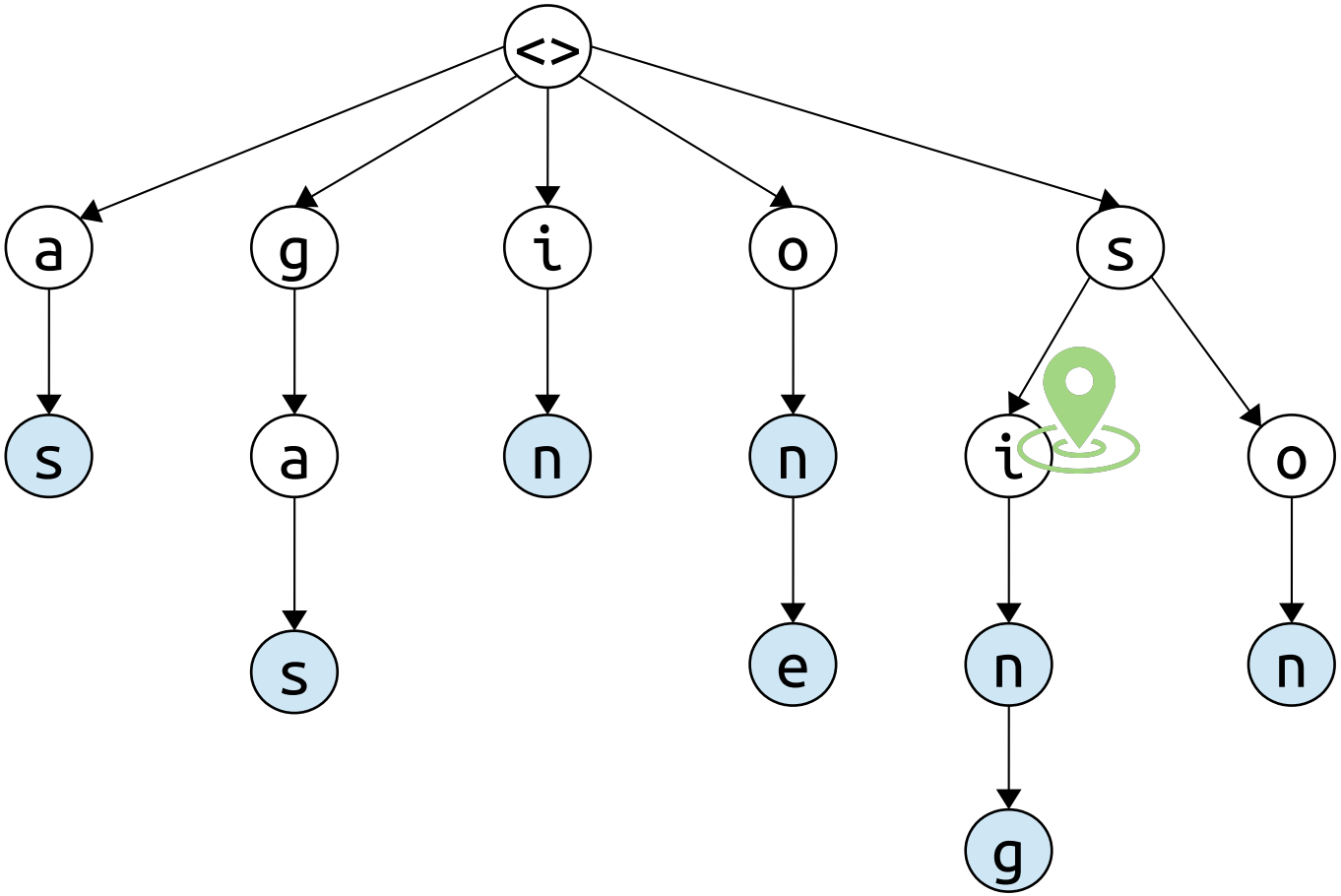
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



text:

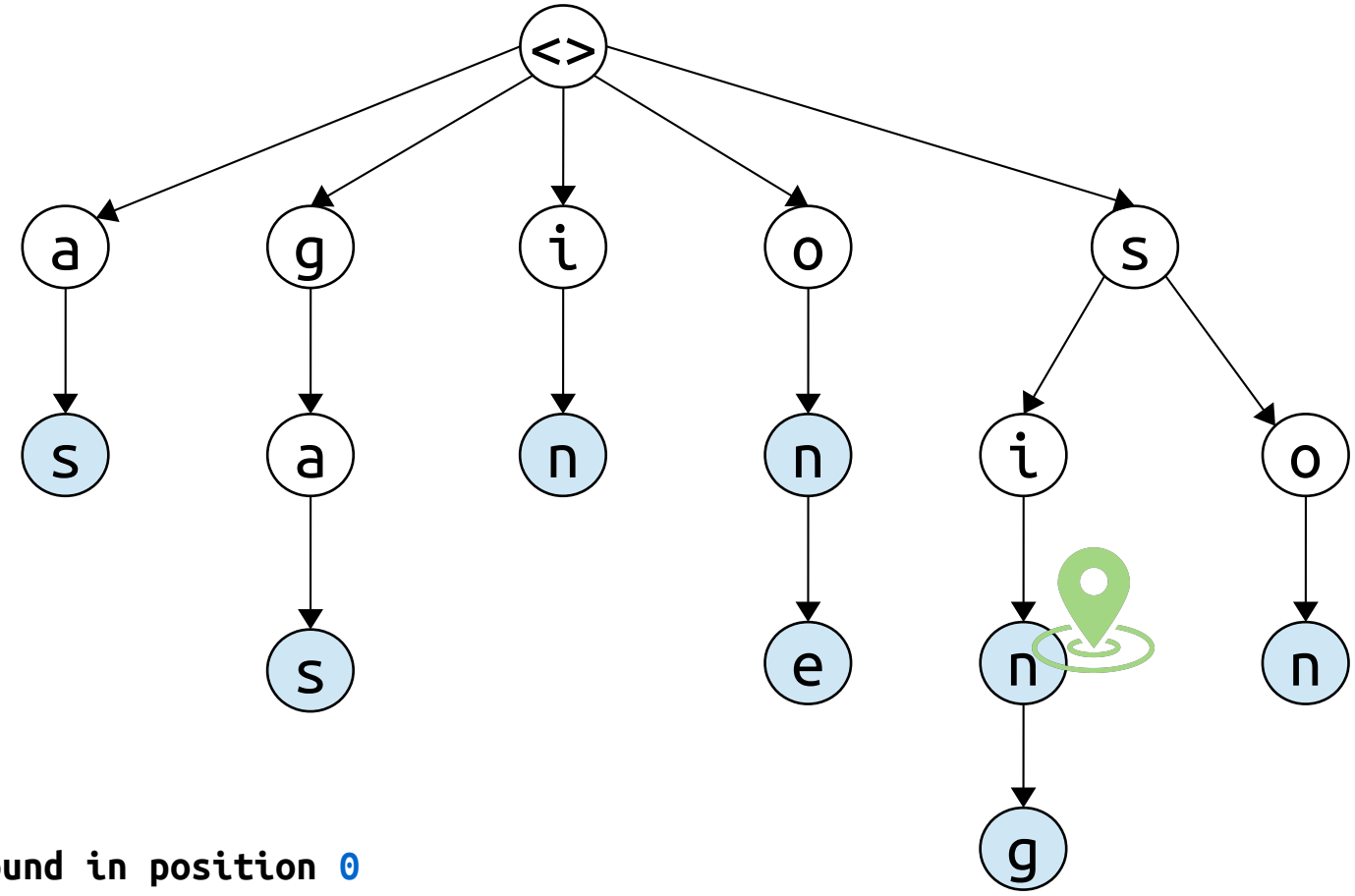
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



“sin” found in position 0

text:

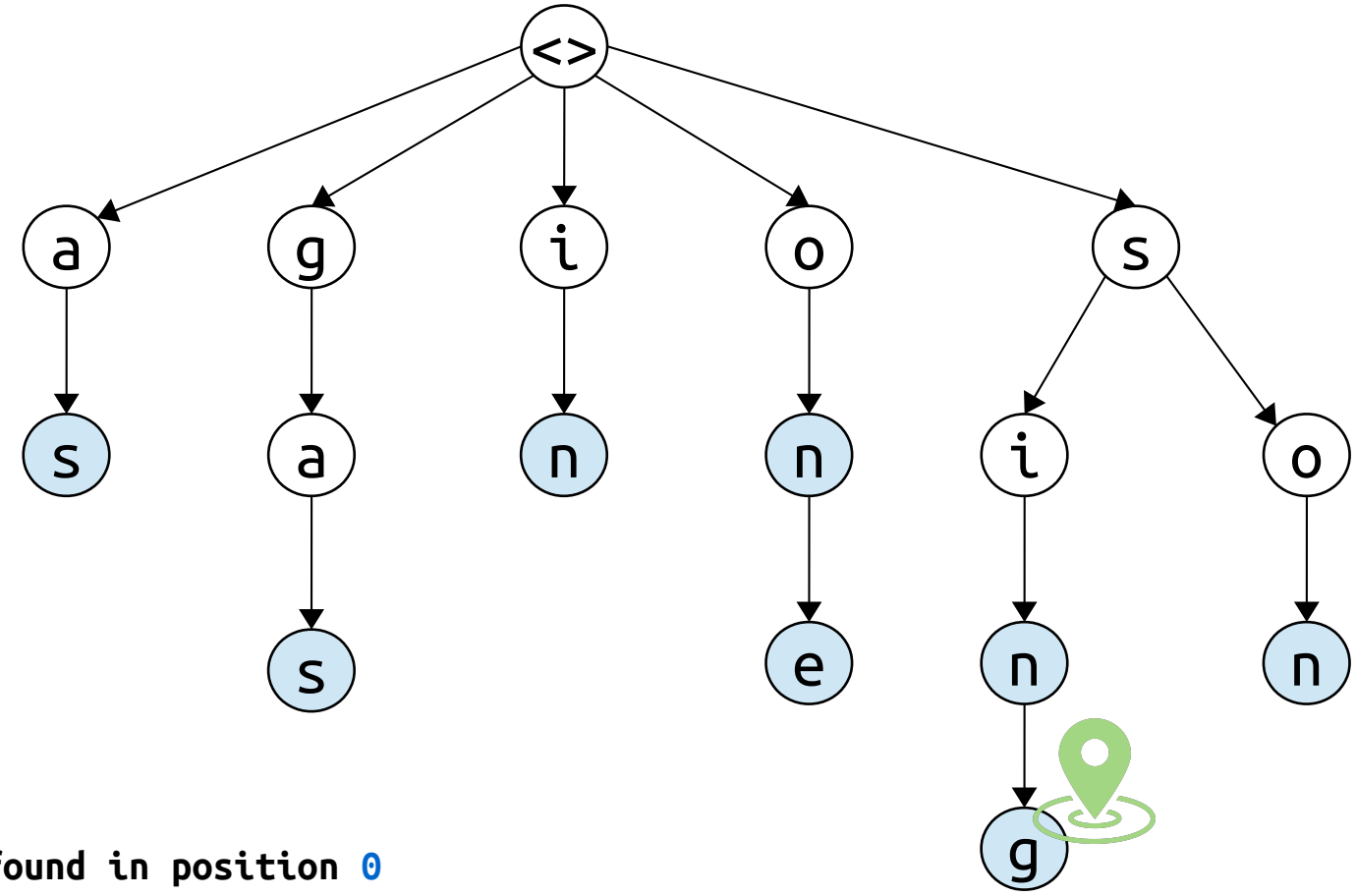
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



“sing” found in position 0

text:

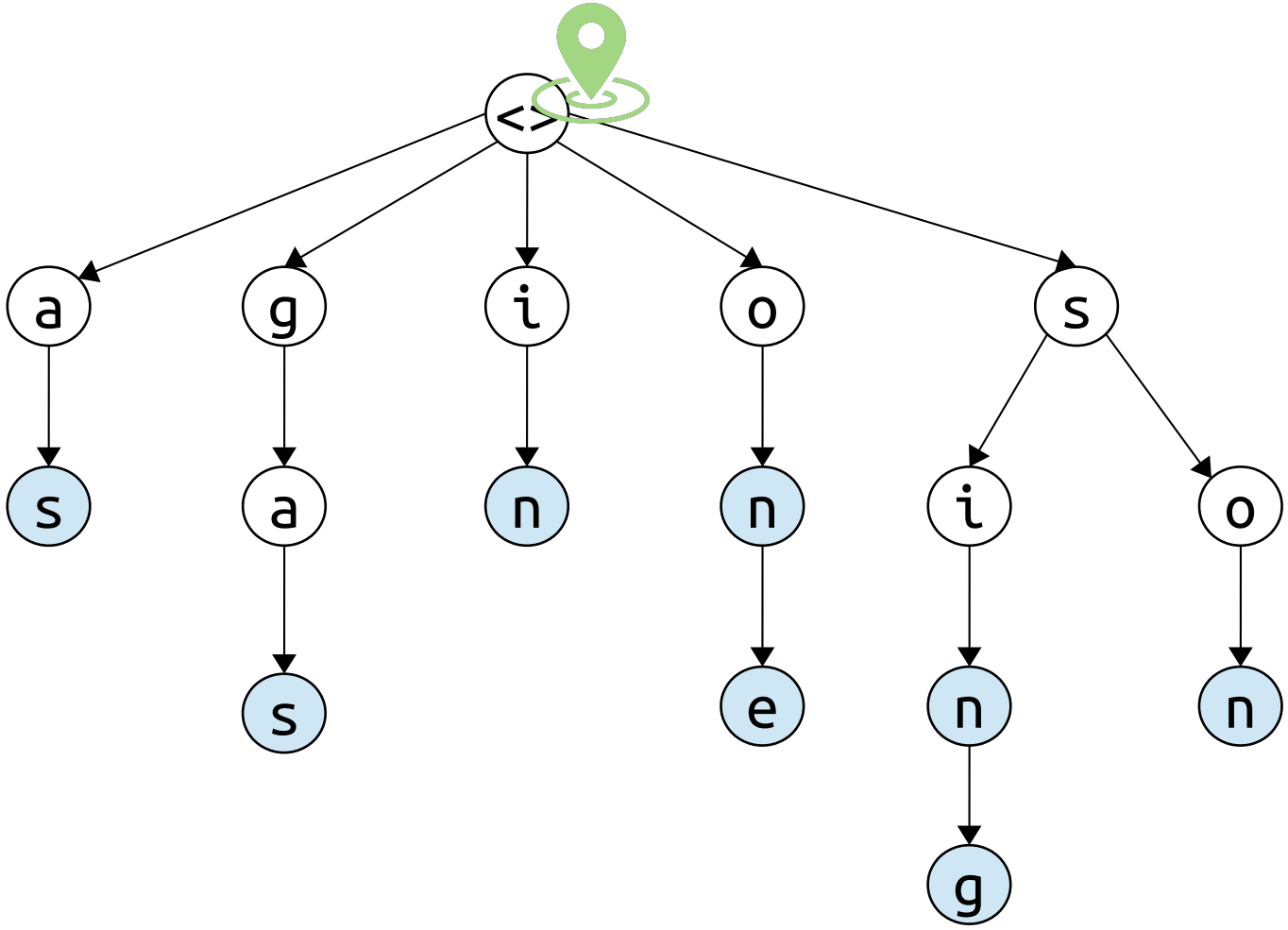
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



text:

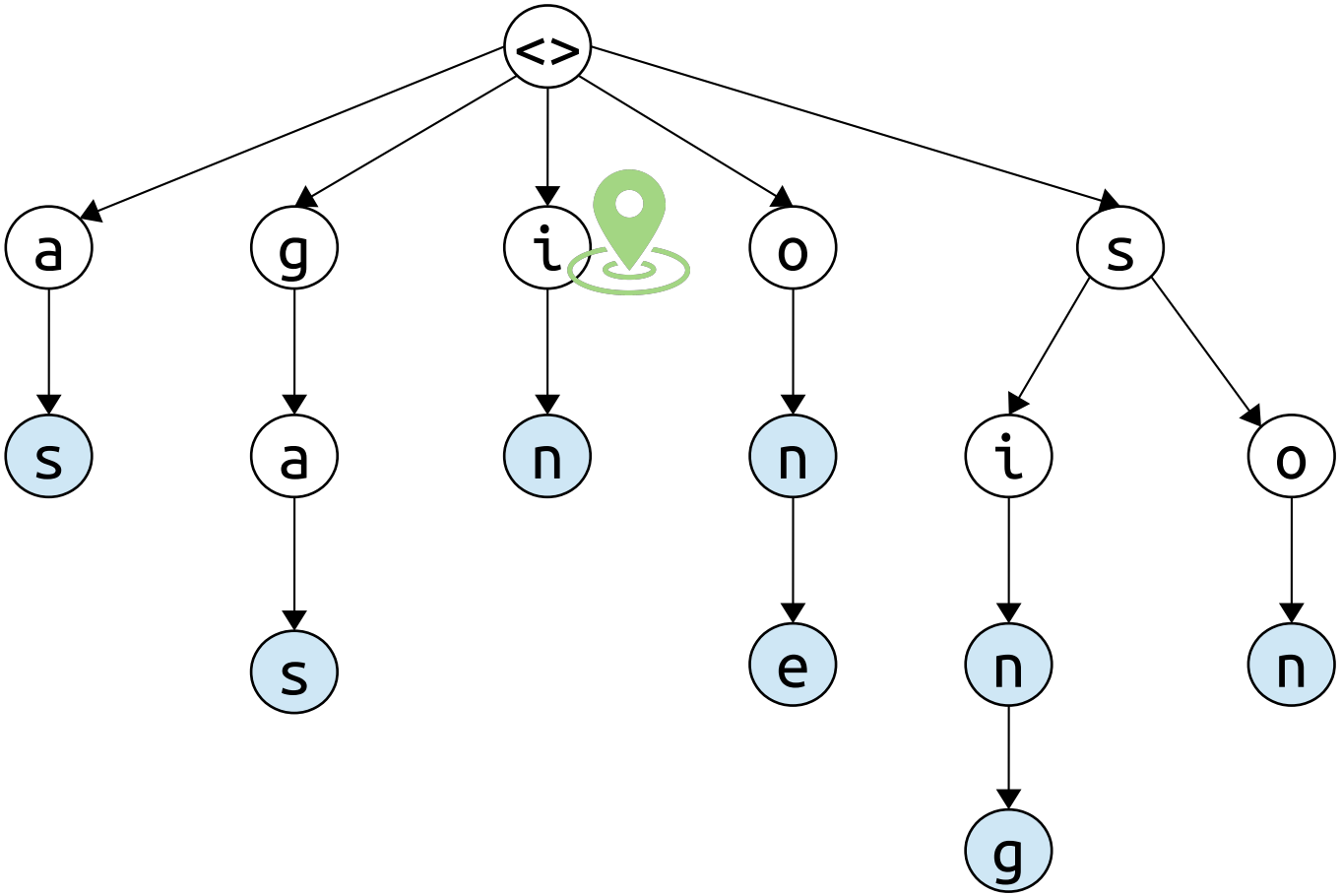
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



text:

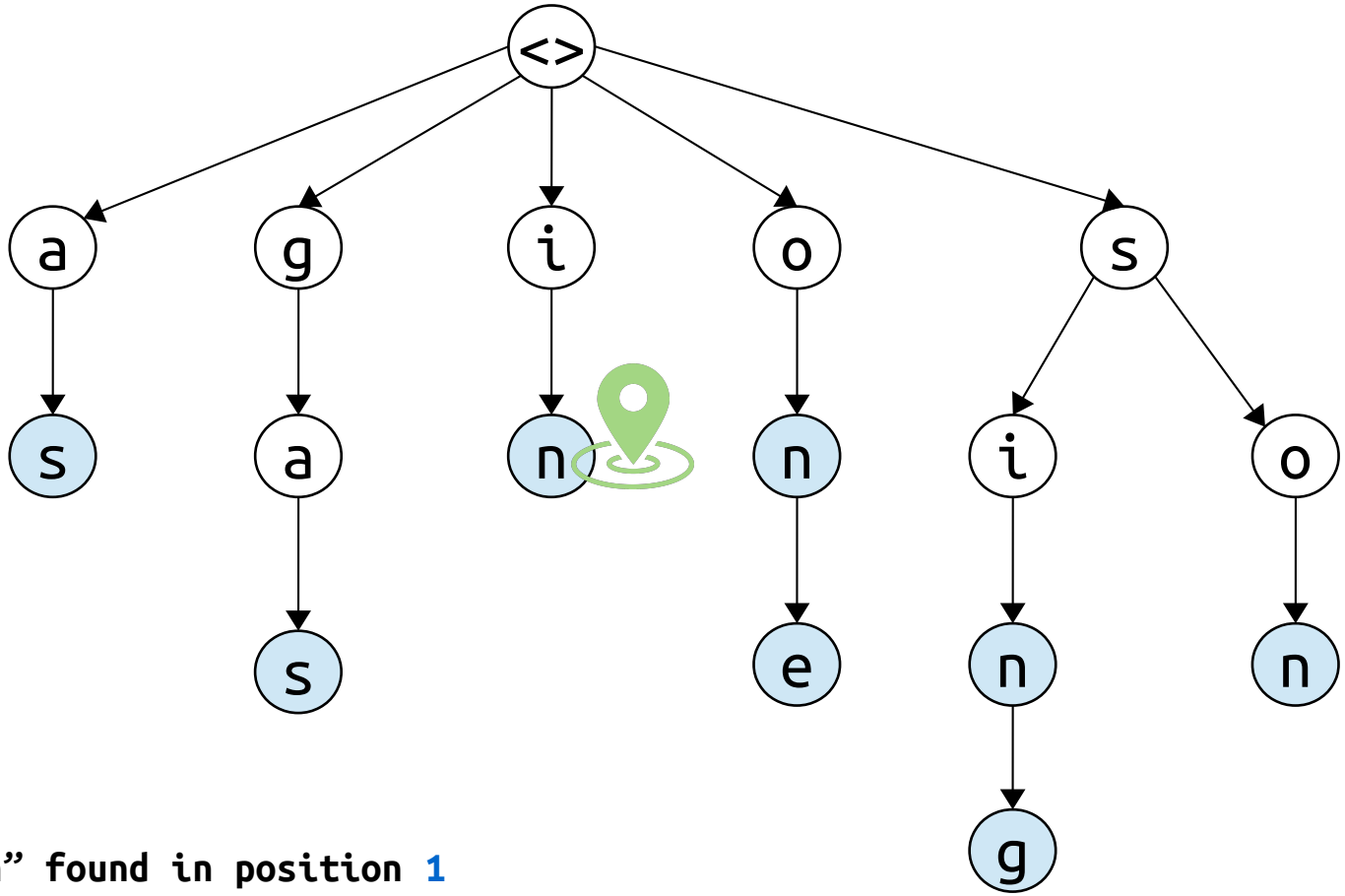
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



"in" found in position 1

text:

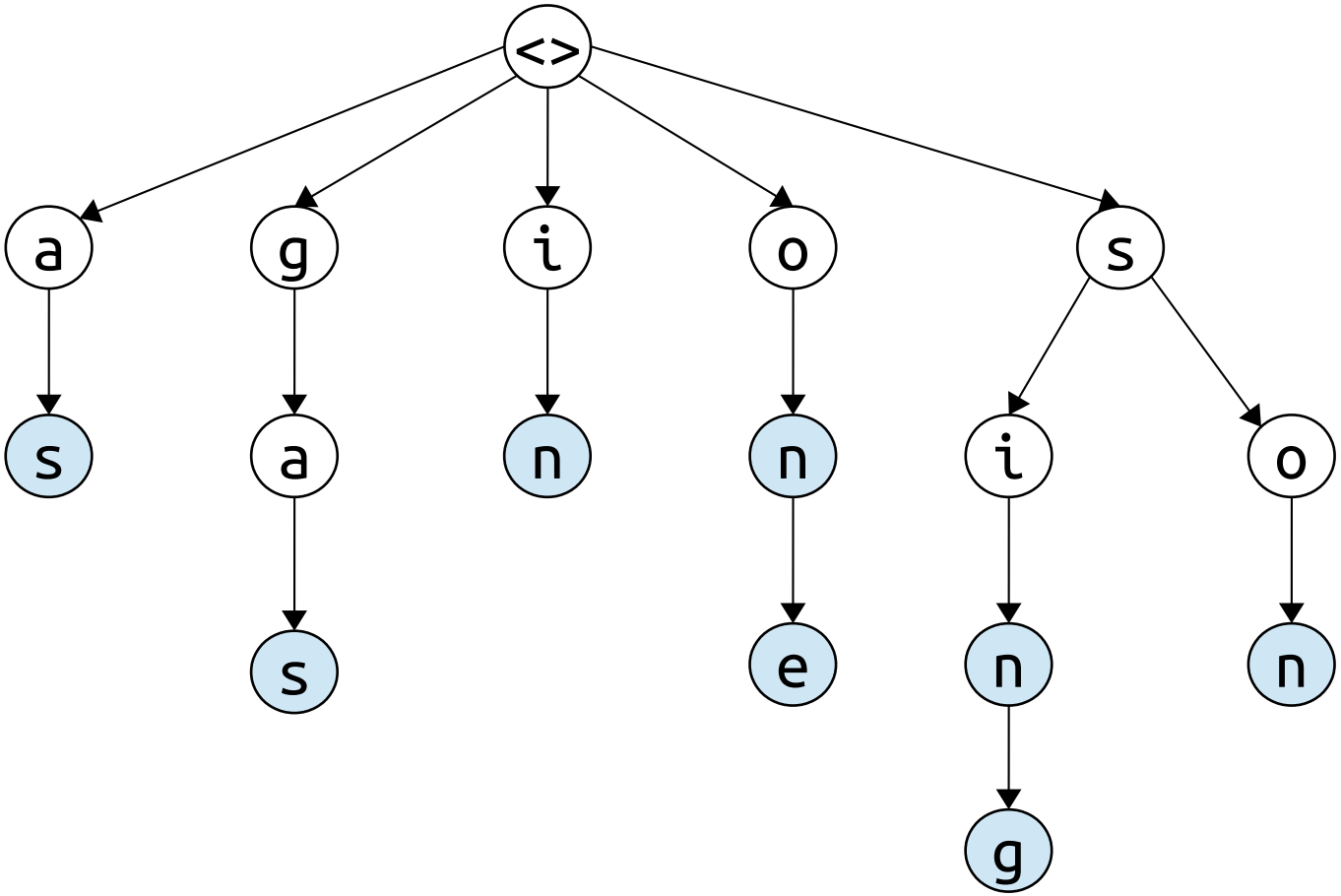
s	i	n	g	a	s	o	n	a	r

Aho-Corasick

Given a text and multiple patterns stored in a trie
find all the patterns in the text

patterns:

a	s		
g	a	s	
i	n		
o	n		
o	n	e	
s	i	n	g
s	i	n	
s	o	n	



AHO-CORASICK BASIC: TIME COMPLEXITY

O(M * Lmax) where

M : size of the text

Lmax : size of the longest pattern

Aho-Corasick – Suffix Links

In Aho-Corasick Tries (also called Aho-Corasick Automata)
Suffix links are used to *speed up* the search.

Using suffix links,

We can make Aho-Corasick faster
and take down

O(M * Lmax)

To

O(M)

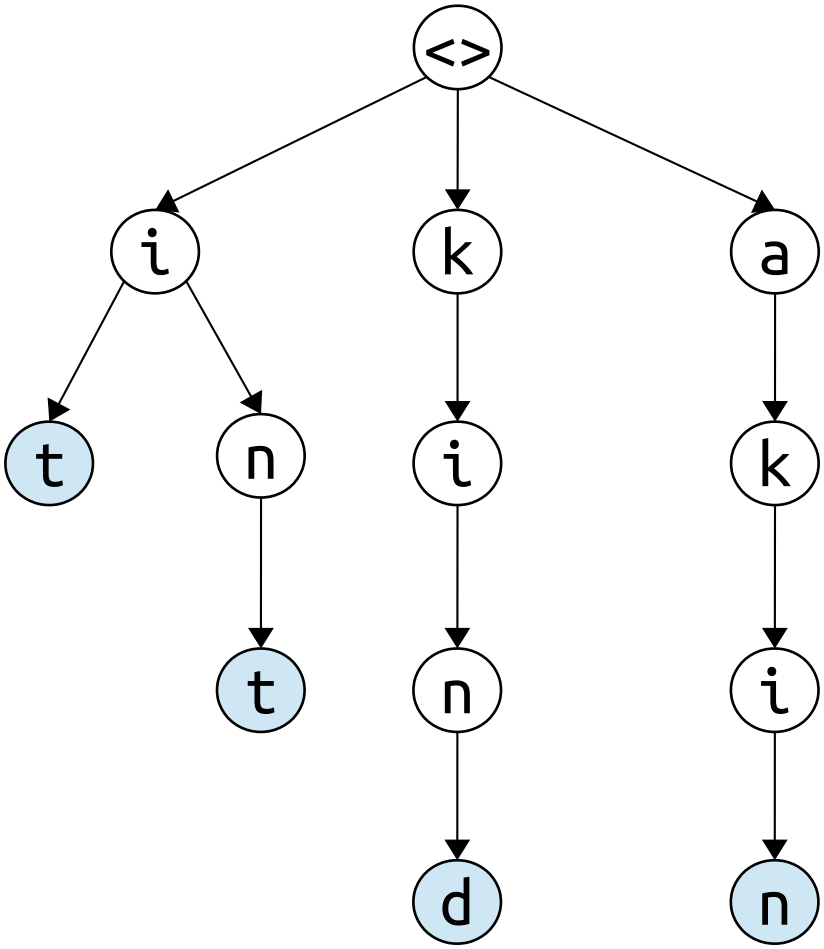
M : size of the text

Lmax : size of the longest pattern

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



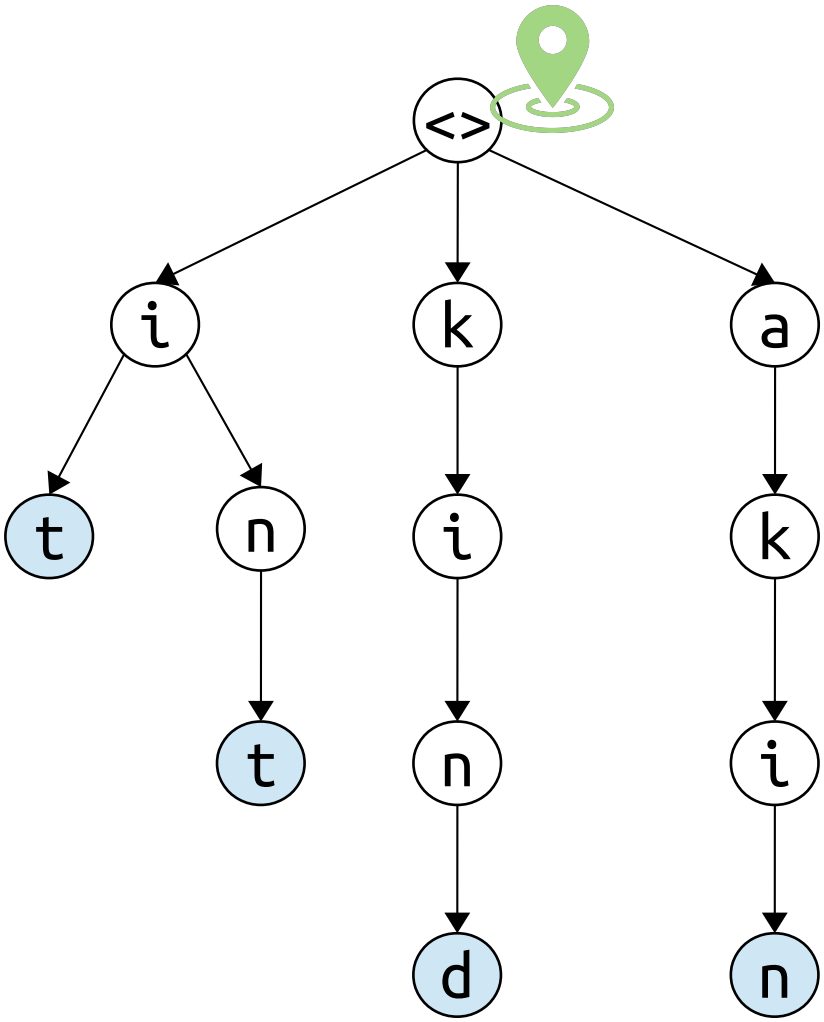
text:

a	k	i	n	d

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



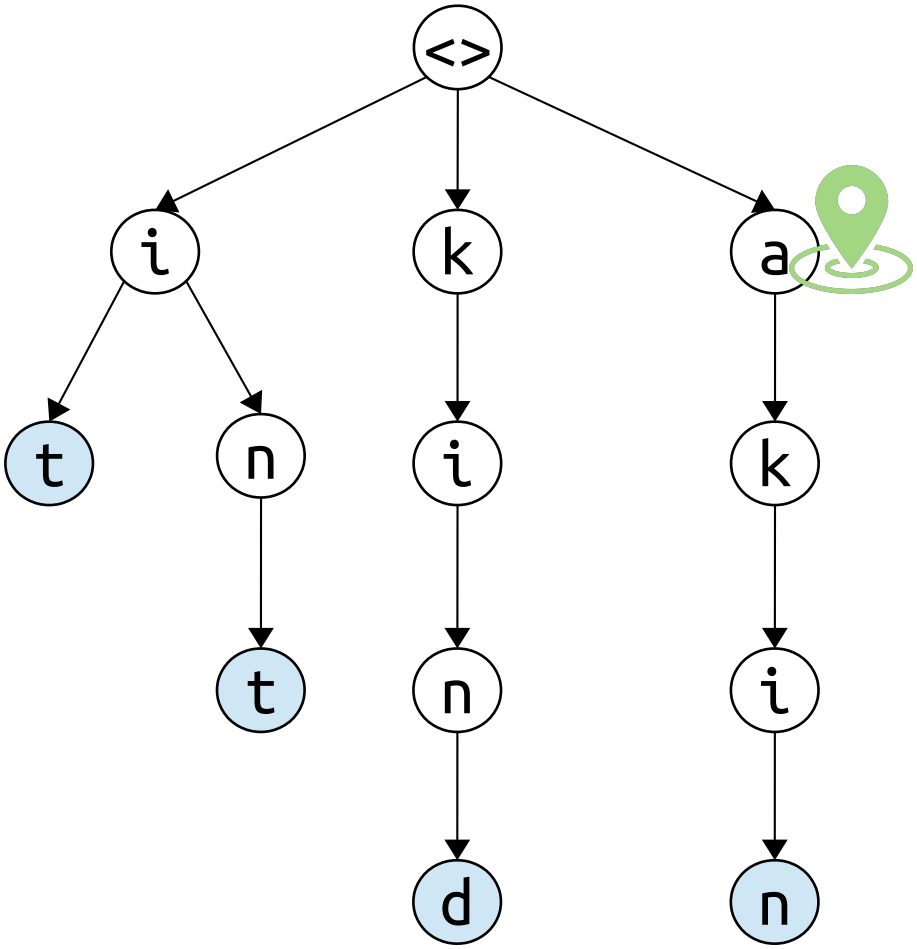
text:

a	k	i	n	d

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



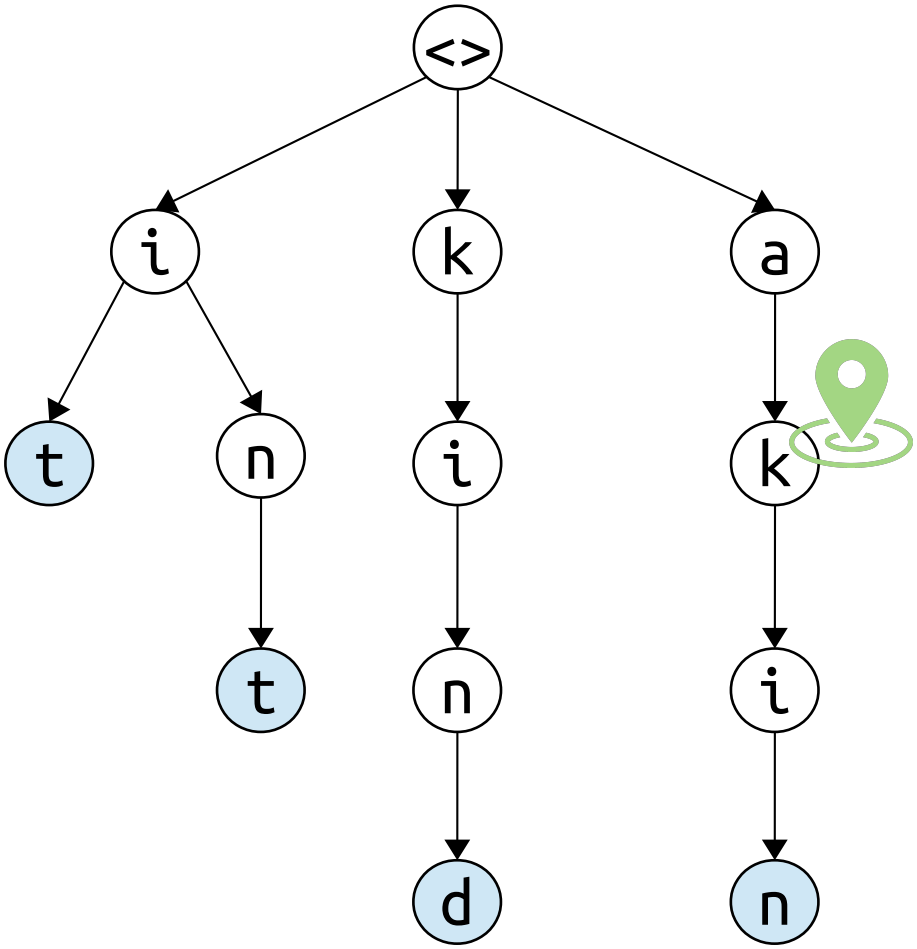
text:

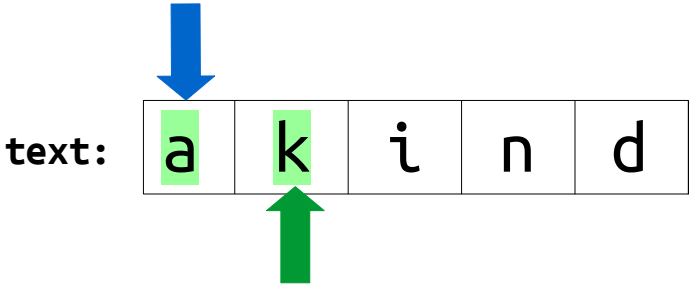
a	k	i	n	d

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n

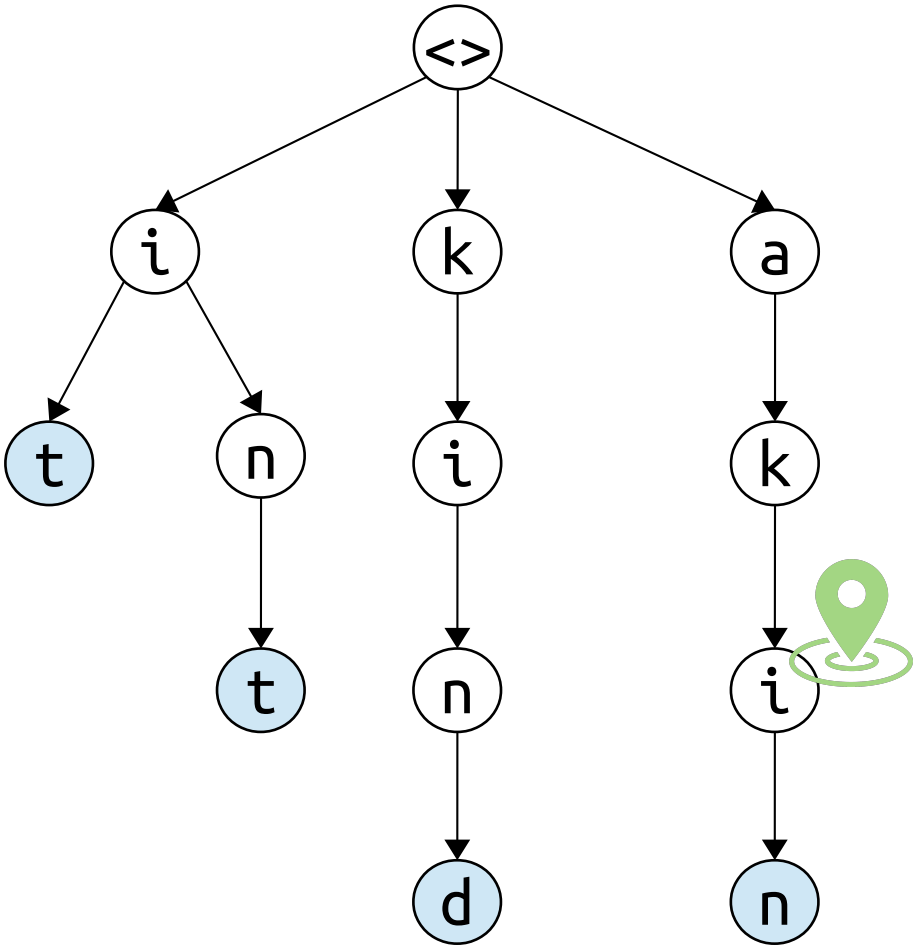




Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



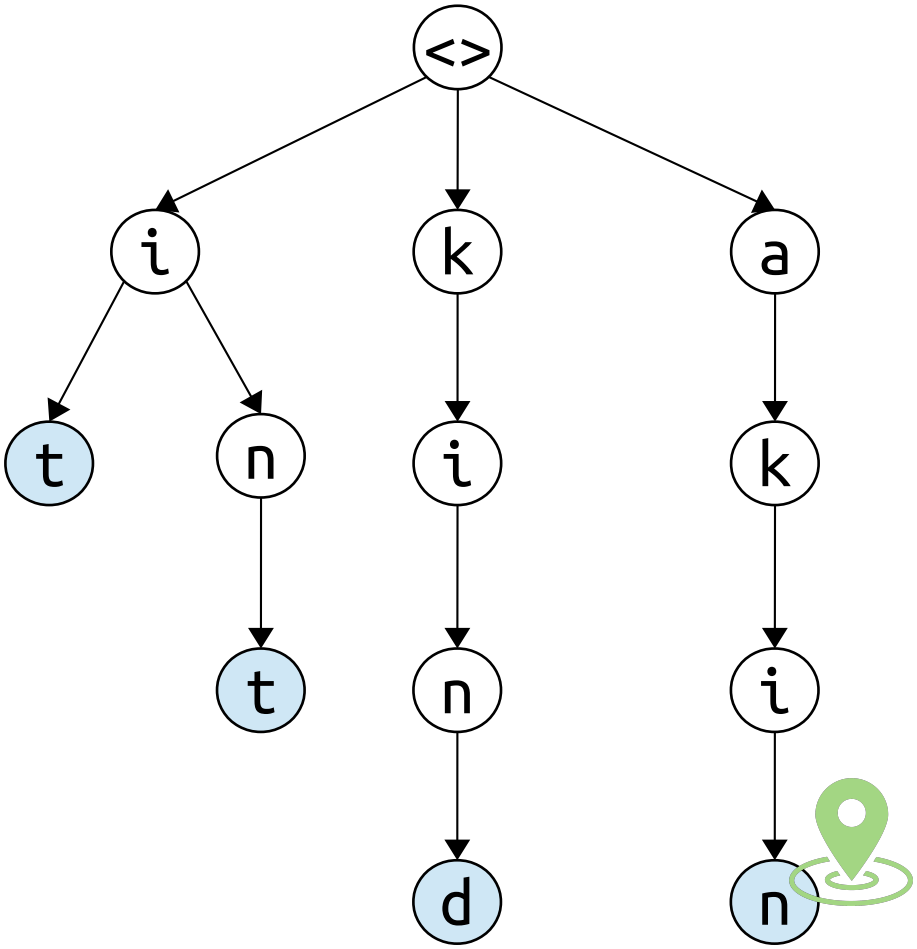
text:

a	k	i	n	d

Aho-Corasick – Suffix Links

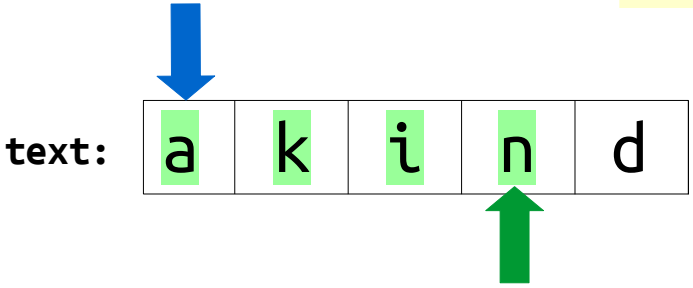
patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



This node has no children, so we should return, and re-start the search from position 1 of the text

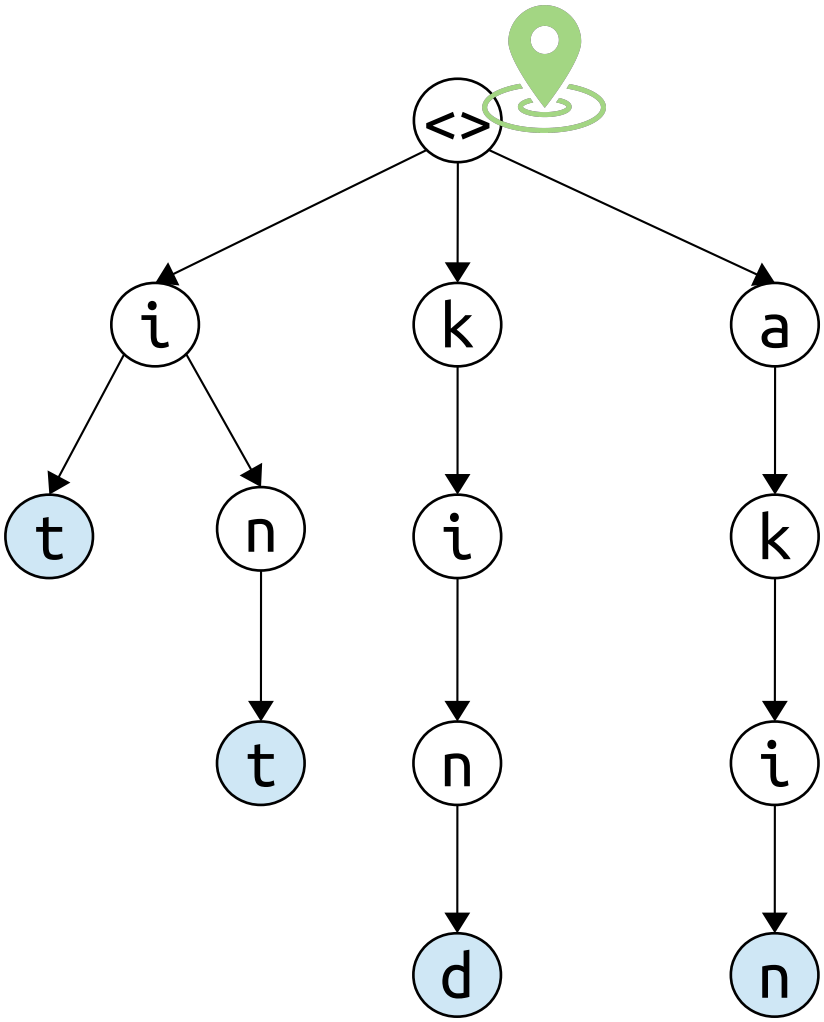
"akin" found in position 0



Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



This node has no children, so we should return, and re-start the search from position 1 of the text

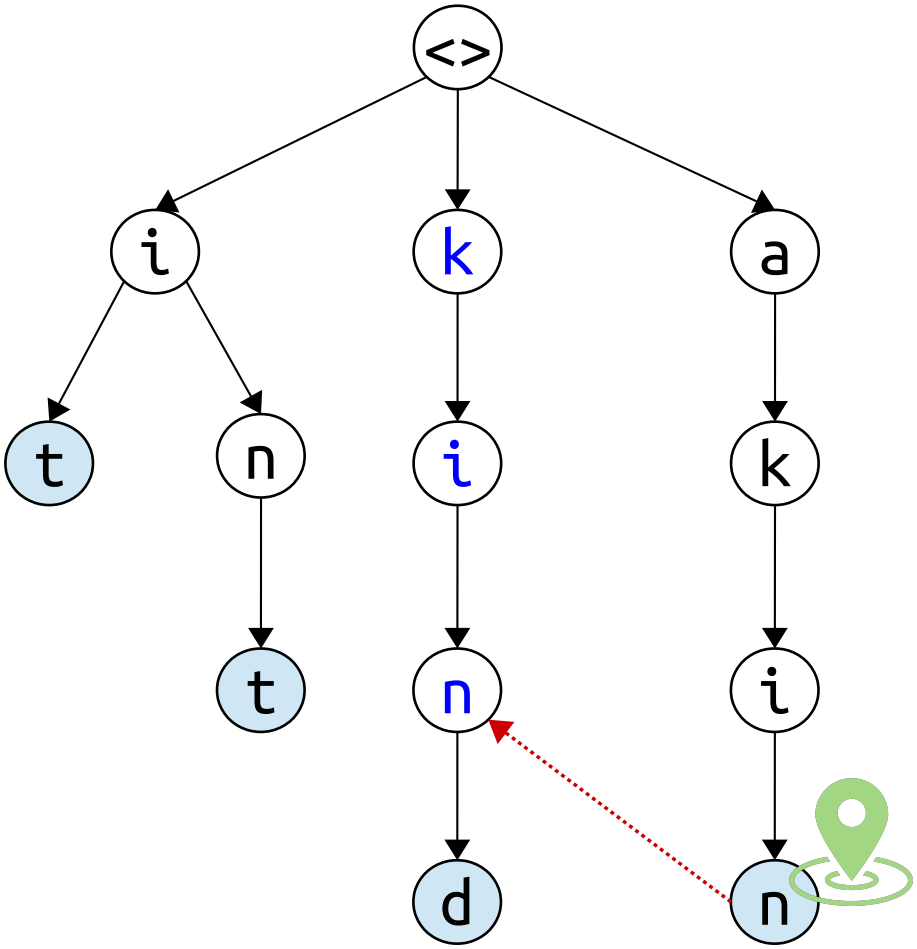
text:

a	k	i	n	d

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



BUT,
A suffix of the sequence we visited appears in another branch of the trie.

We use suffix links to skip steps.

(we will see later how to construct them)

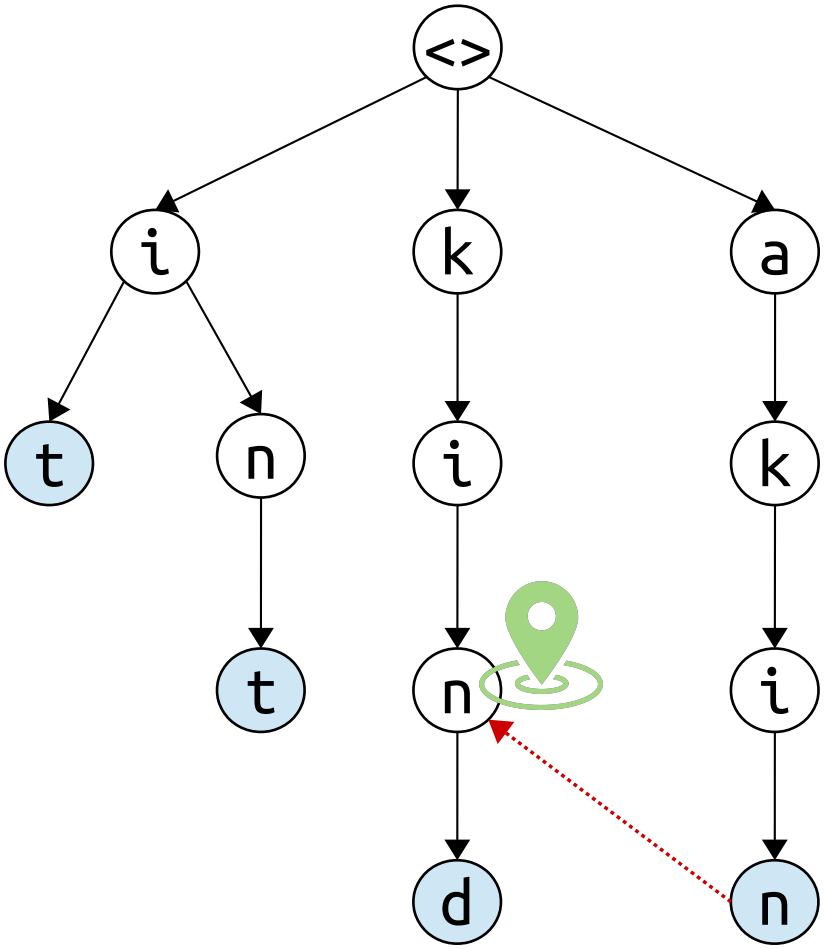
text:

a	k	i	n	d

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



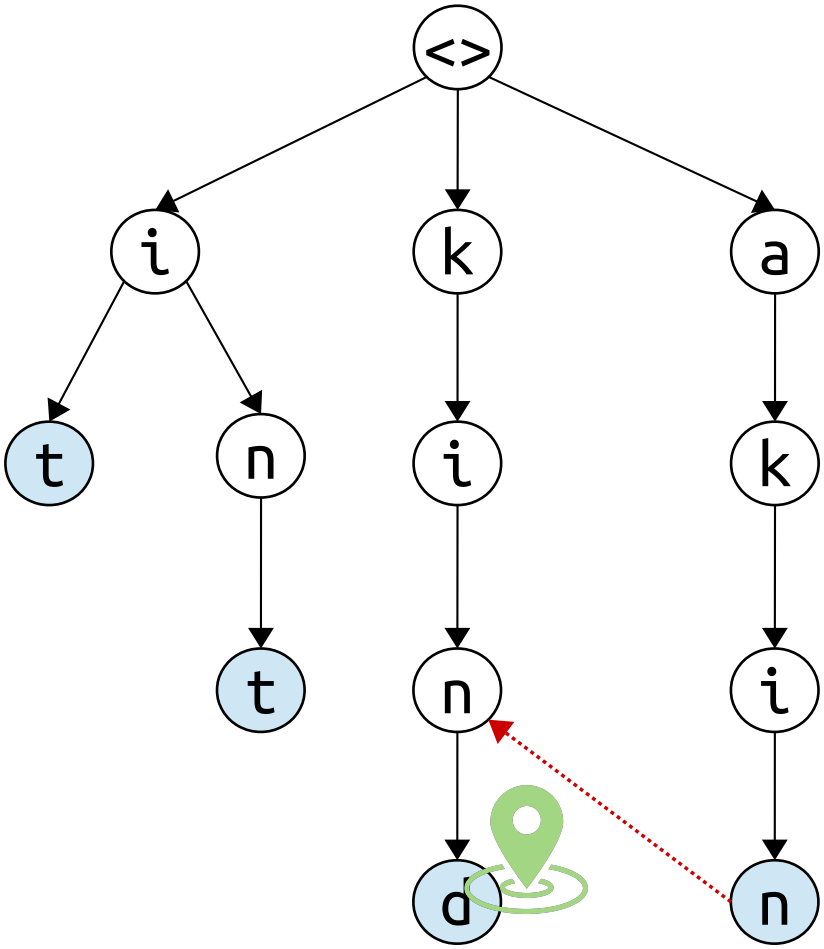
text:

a	k	i	n	d

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



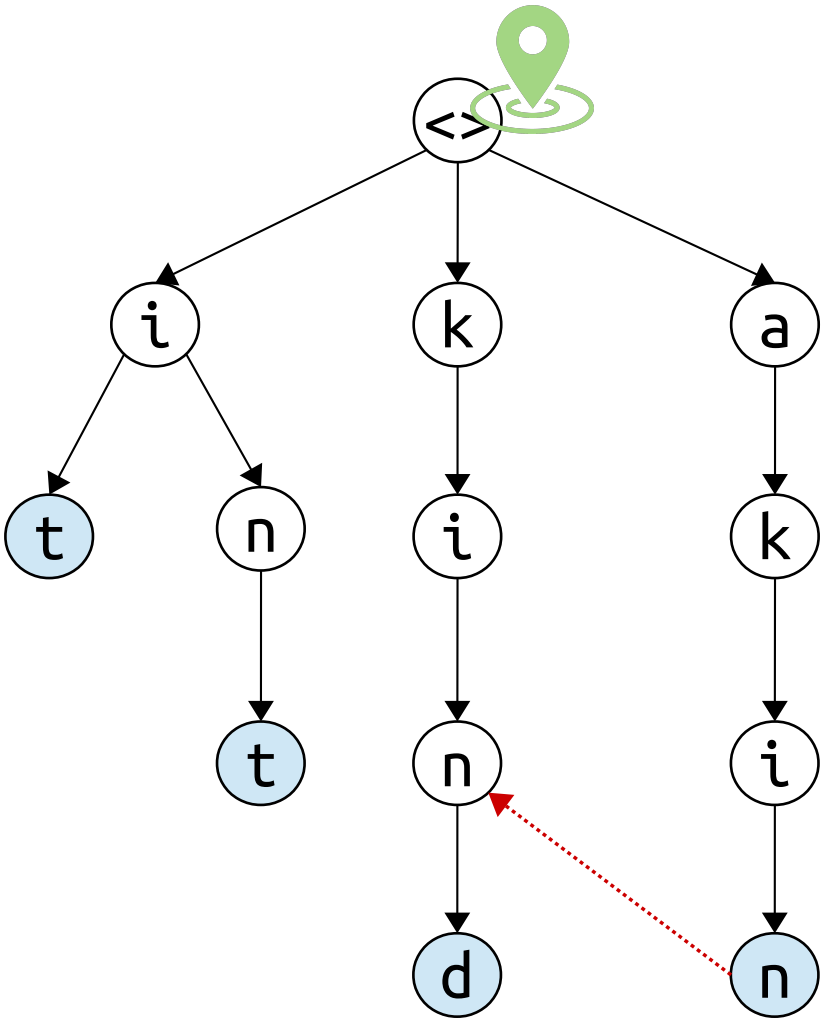
text:

a	k	i	n	d

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



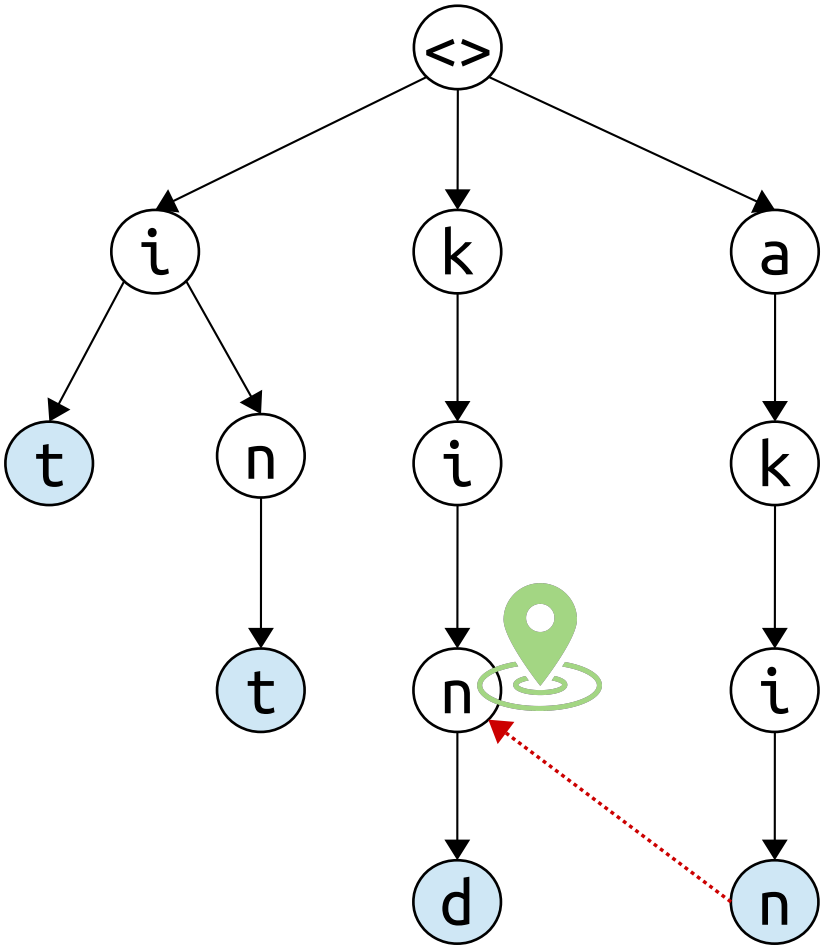
text:

k	i	n	t

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



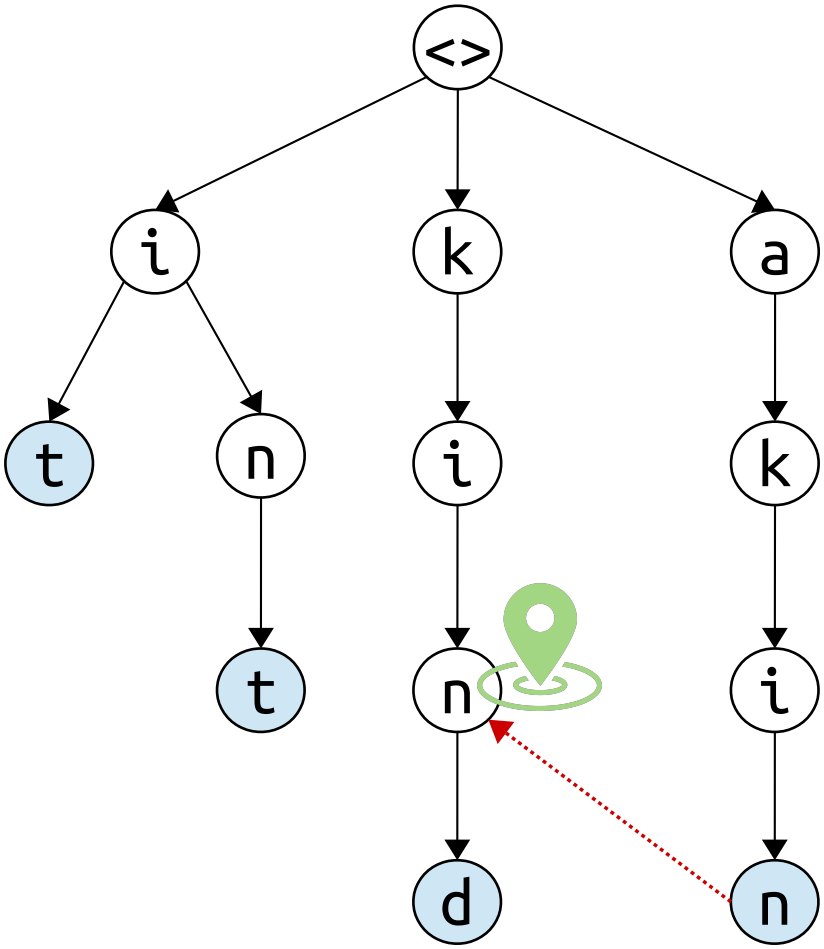
text:

k	i	n	t

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



text:

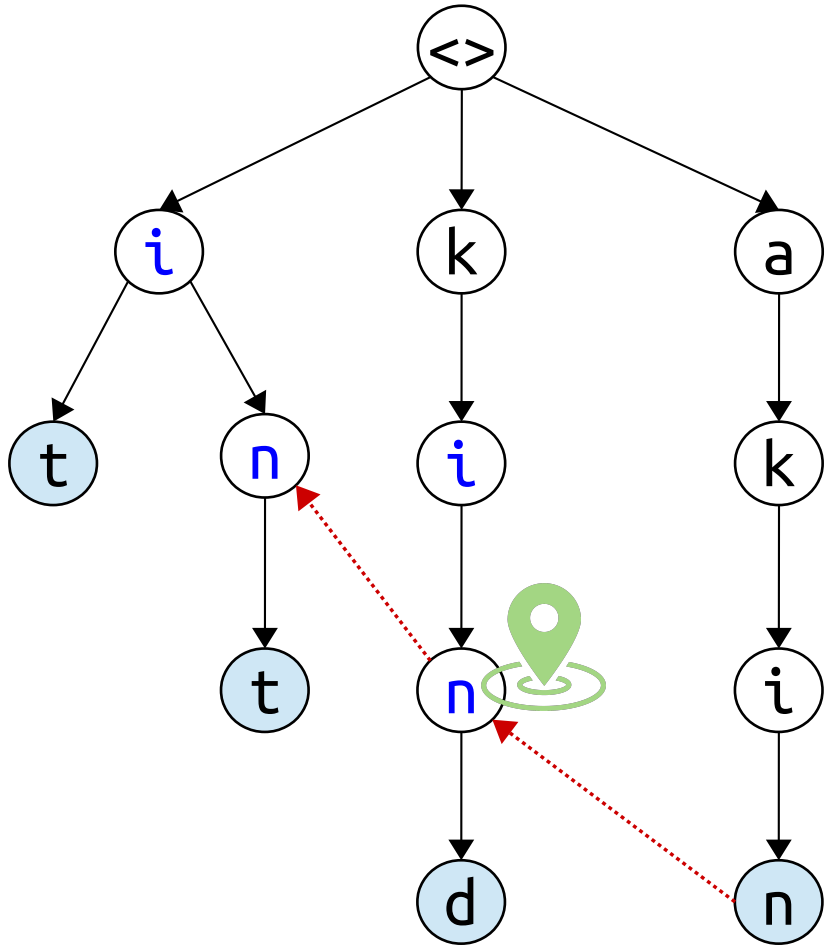
k	i	n	t

No pattern has found in the current branch.
Node “n” has not child “t”

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



text:

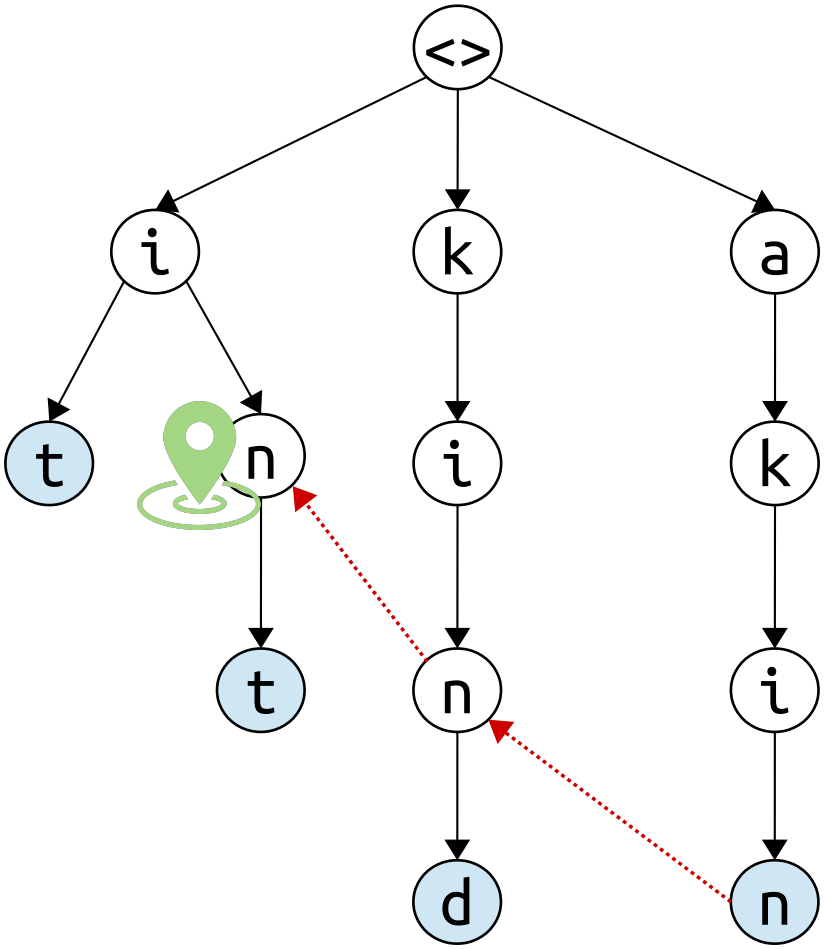
k	i	n	t

BUT, suffix "i n" of the sequence we have visited appears in another branch of the trie.

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



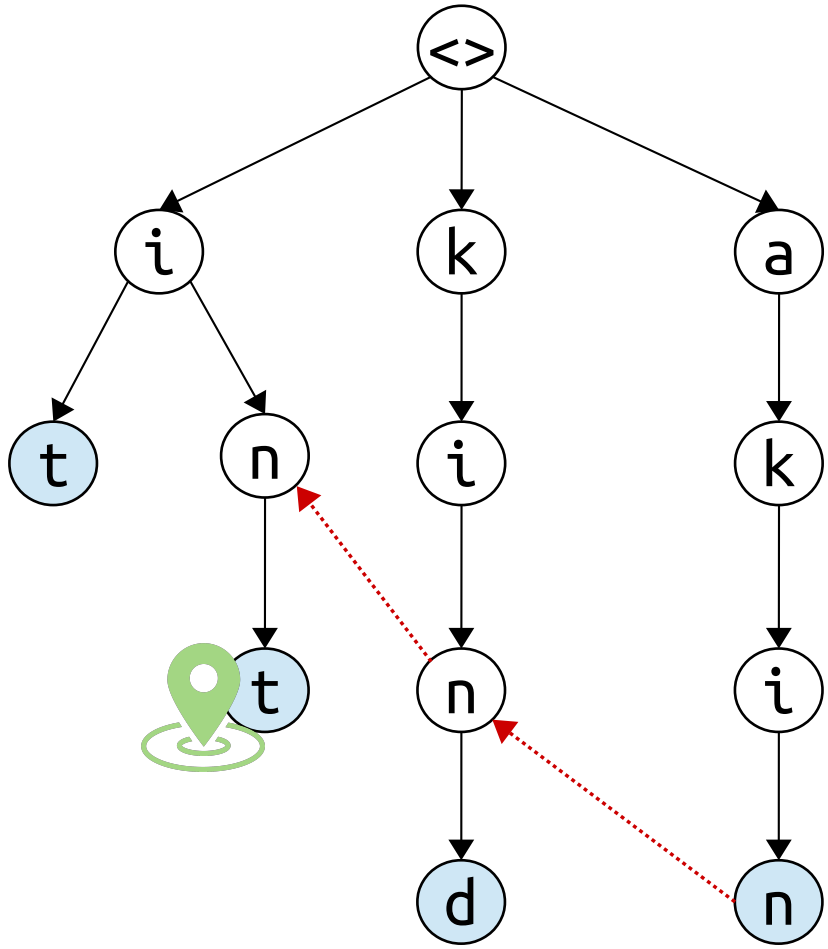
text:

k	i	n	t

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



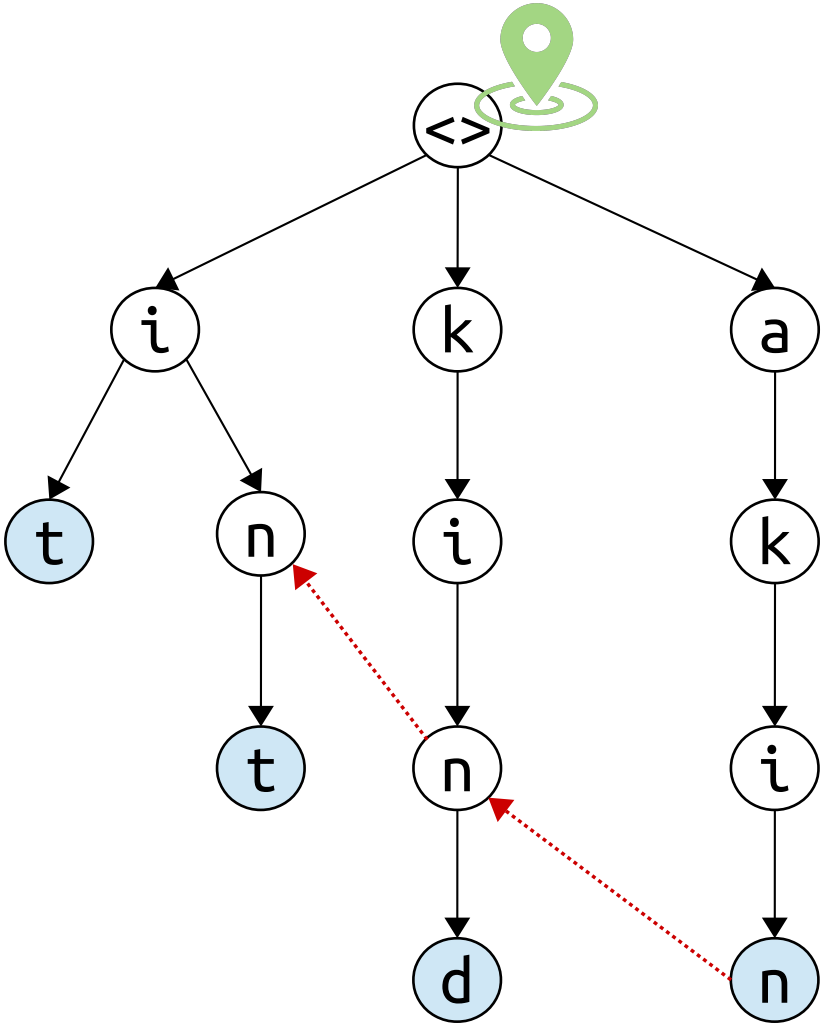
text:

k	i	n	t

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



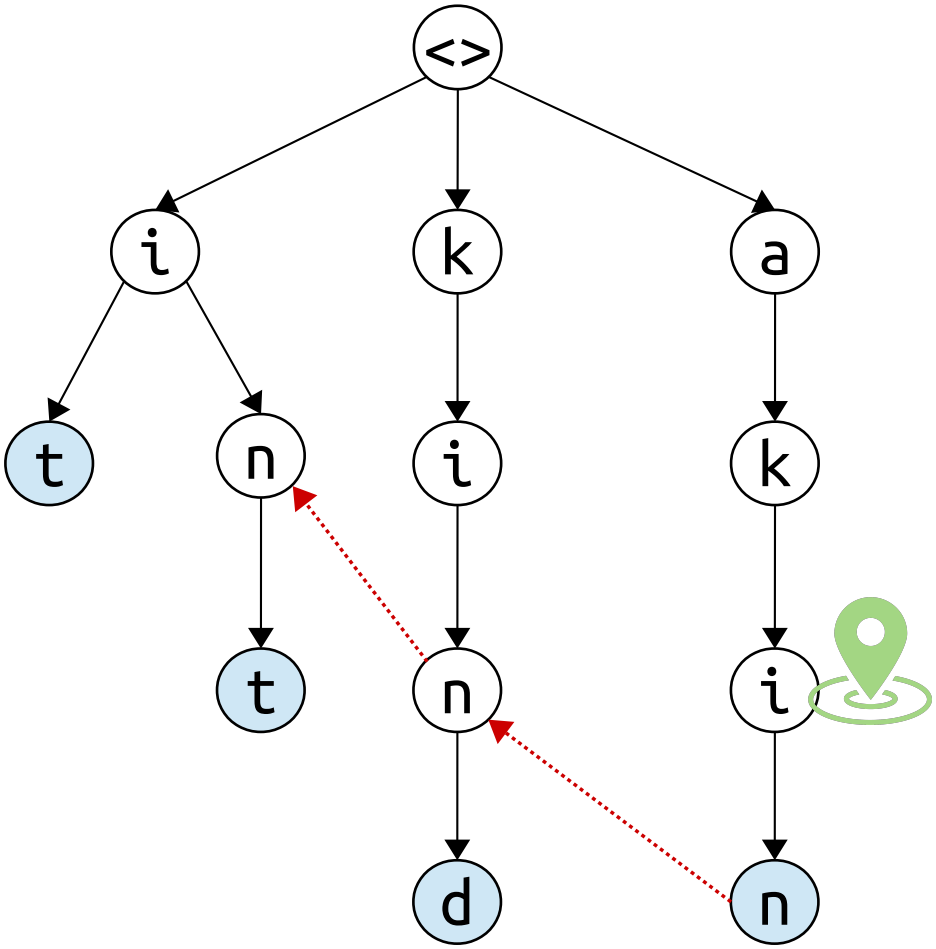
text:

a	k	i	t

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



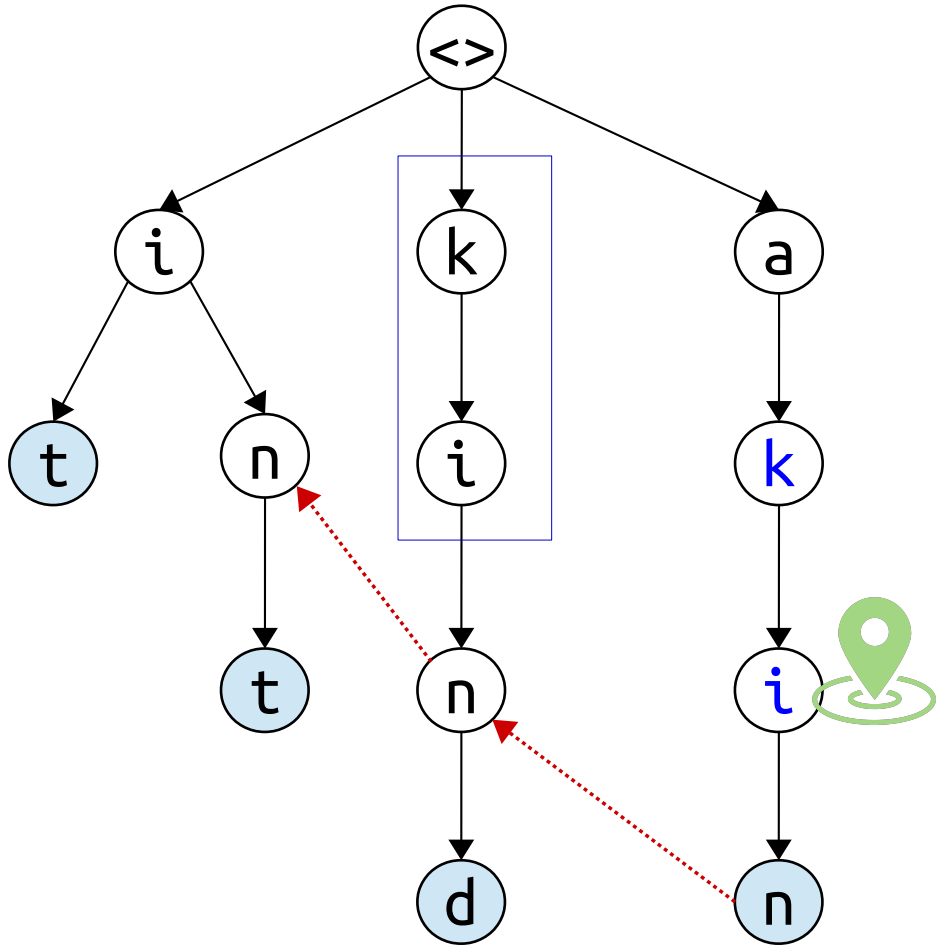
text:

a	k	i	t

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



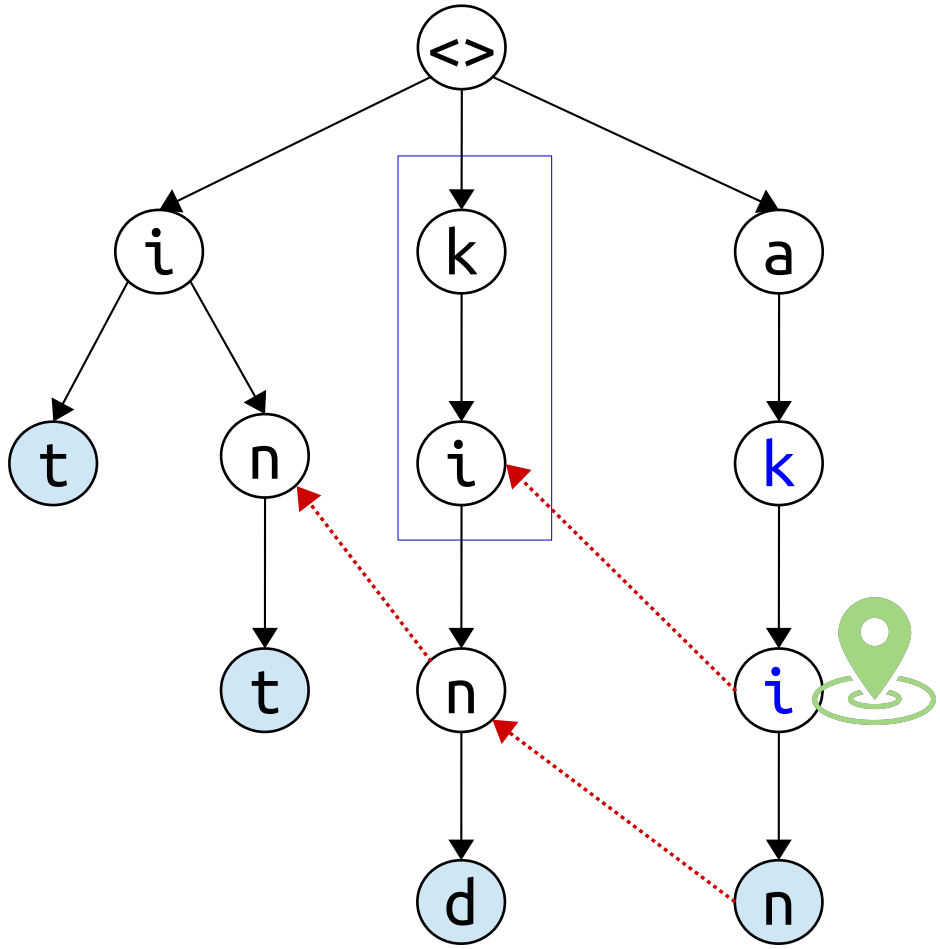
text:

a	k	i	t

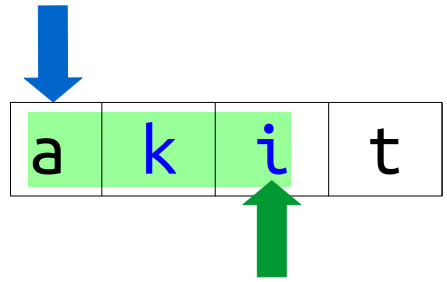
Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



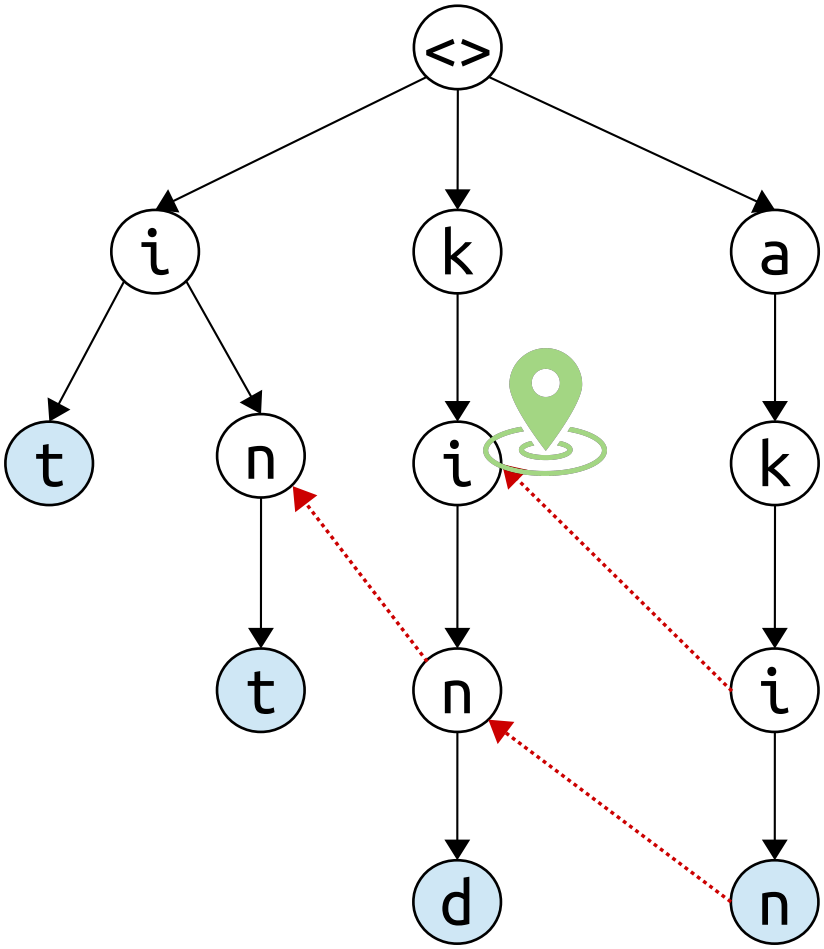
text:



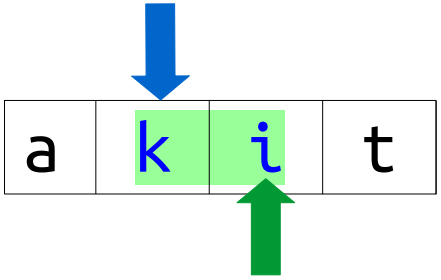
Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



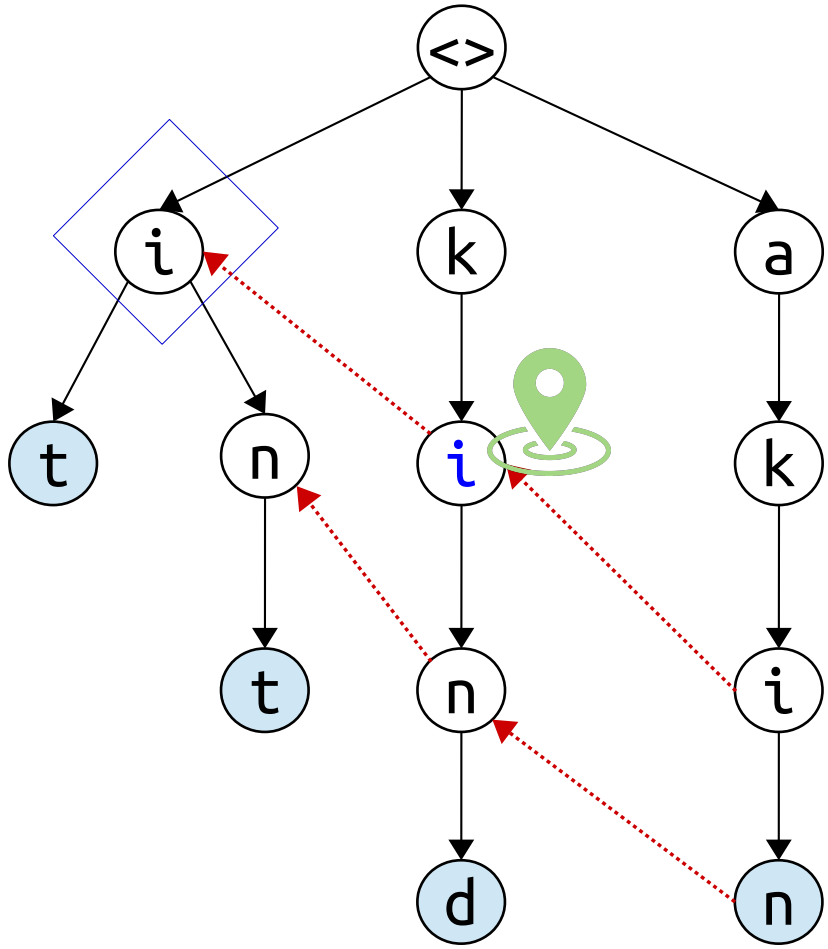
text:



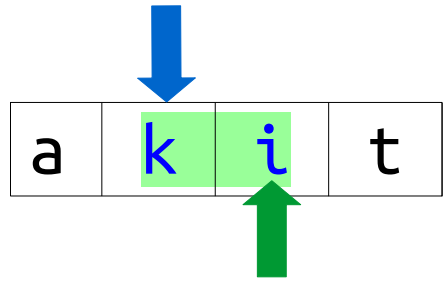
Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



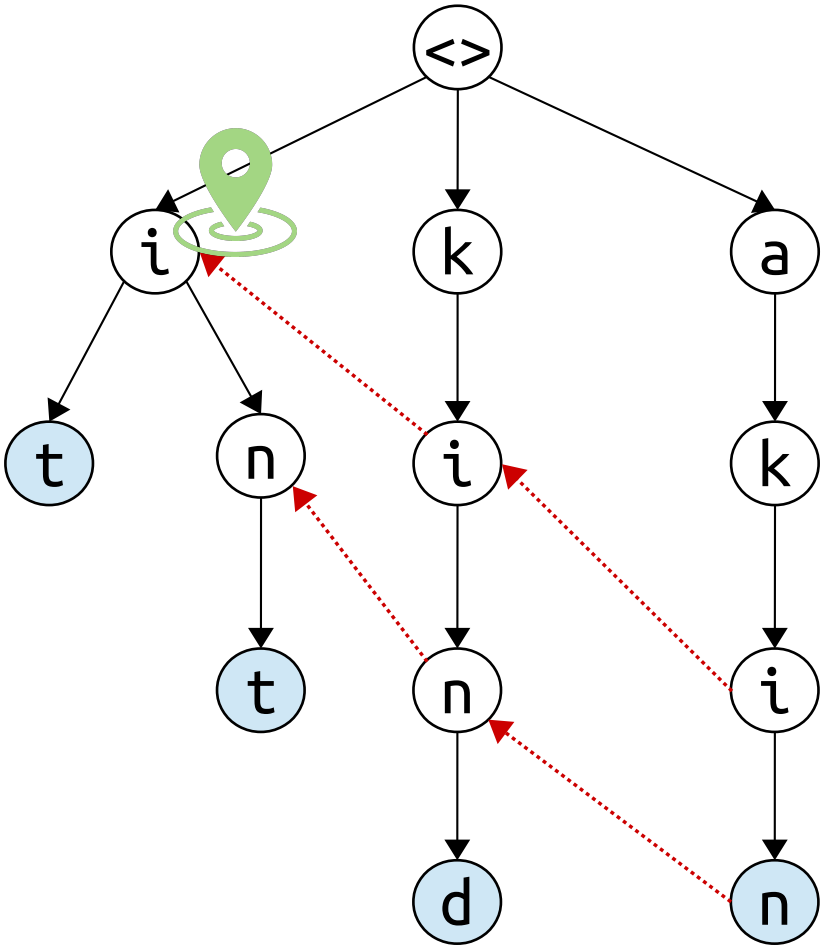
text:



Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



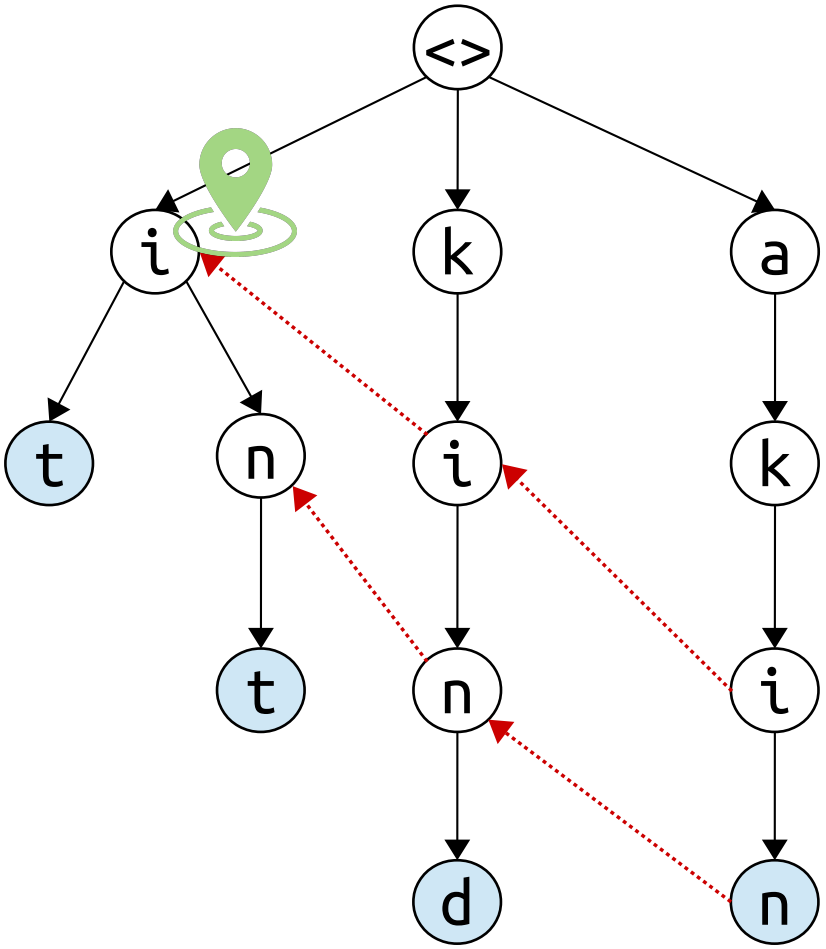
text:

a	k	i	t

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



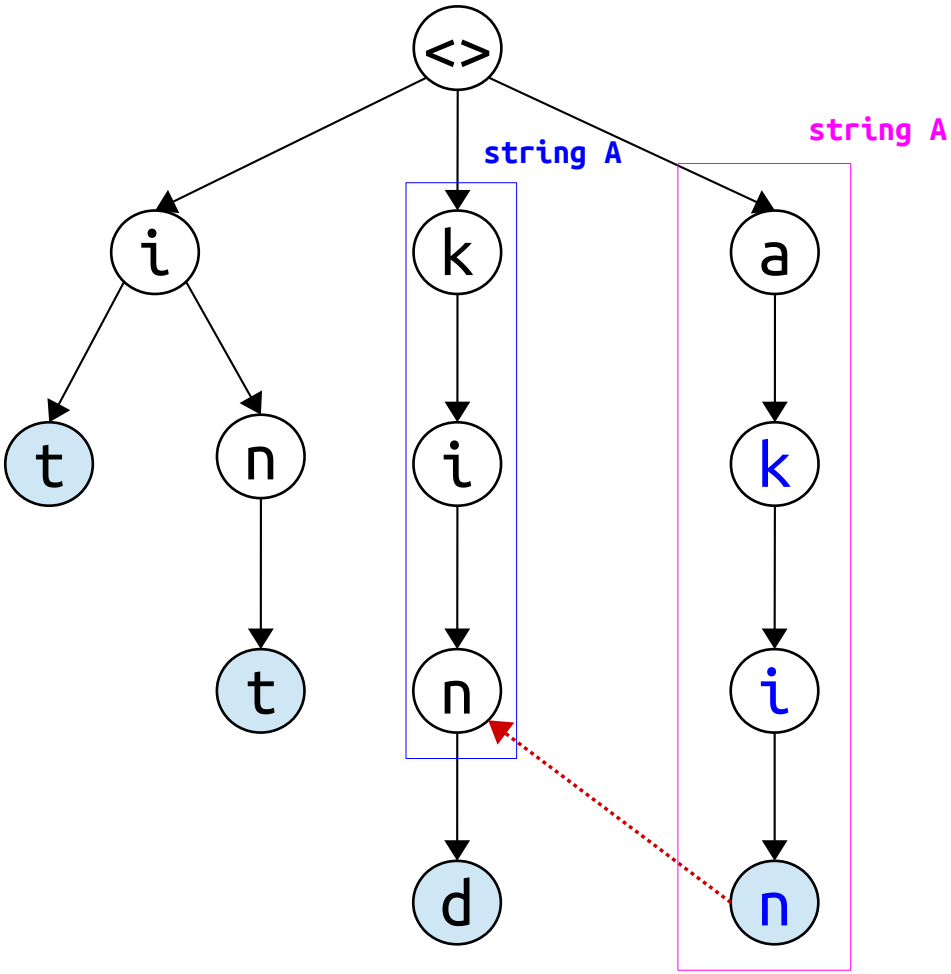
text:

a	k	i	t

Aho-Corasick – Suffix Links

patterns:

i	t		
i	n	t	
k	i	n	d
a	k	i	n



Suffix link leads from a node corresponding to string A to the node corresponding to string B

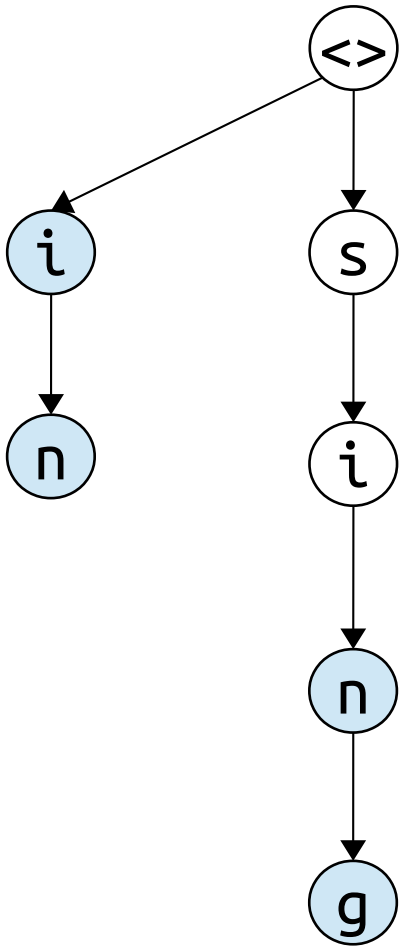
string B is in a branch of the trie and it is the *longest (proper) suffix* of string A.

Every node in the trie has a non-null suffix link (except the root).

Aho-Corasick – Output Links

patterns:

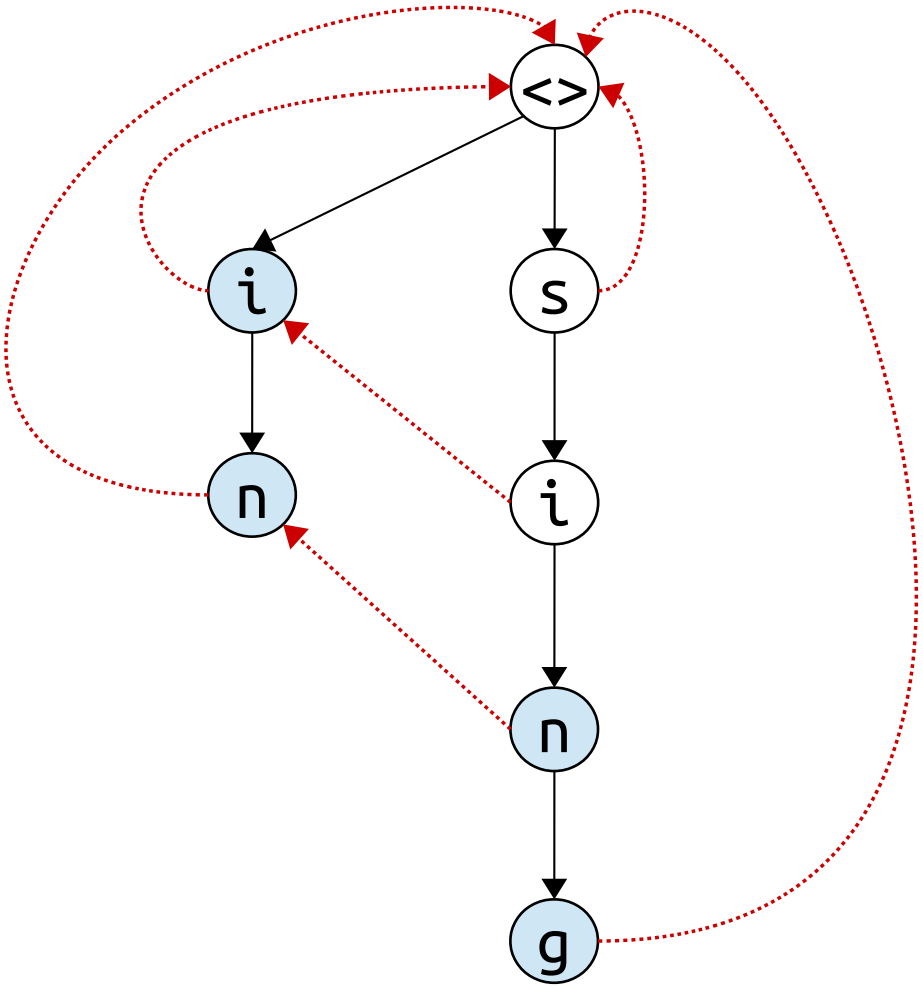
i			
i	n		
s	i	n	
s	i	n	g



Aho-Corasick – Output Links

patterns:

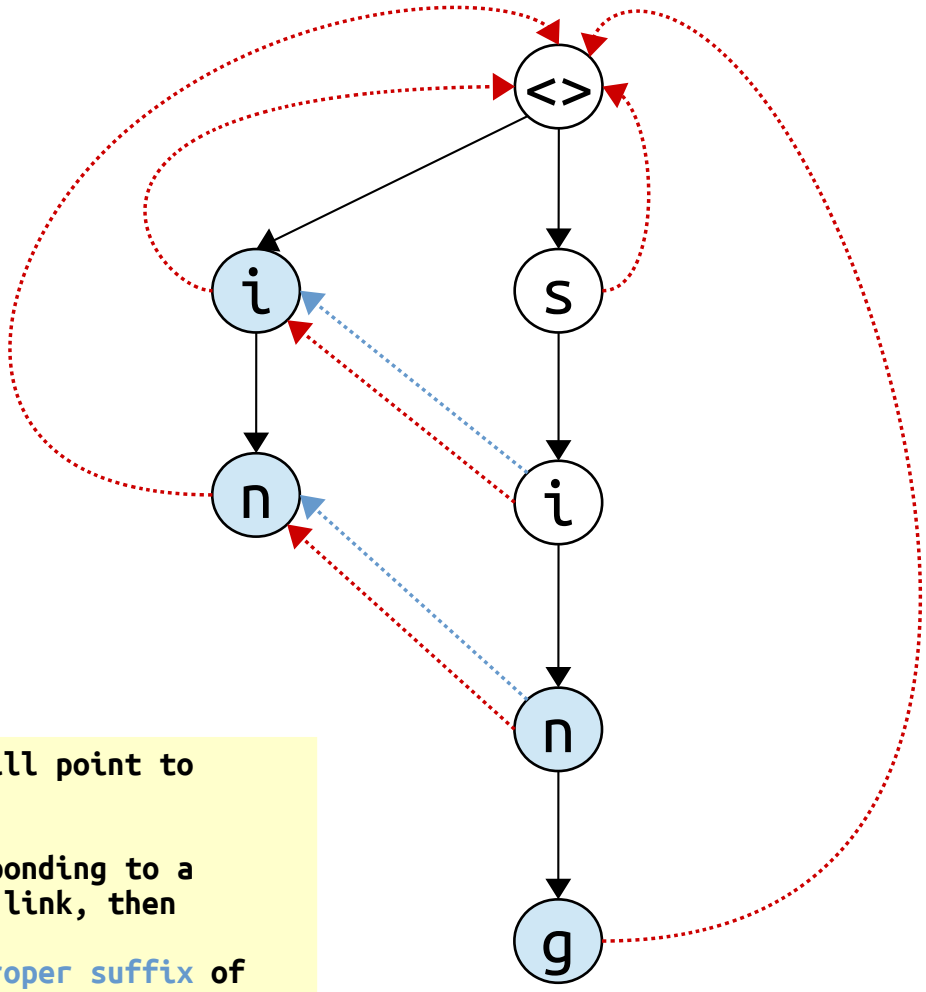
i			
i	n		
s	i	n	
s	i	n	g



Aho-Corasick – Output Links

patterns:

i			
i	n		
s	i	n	
s	i	n	g



The output link of a node will point to
A endOfWord node.

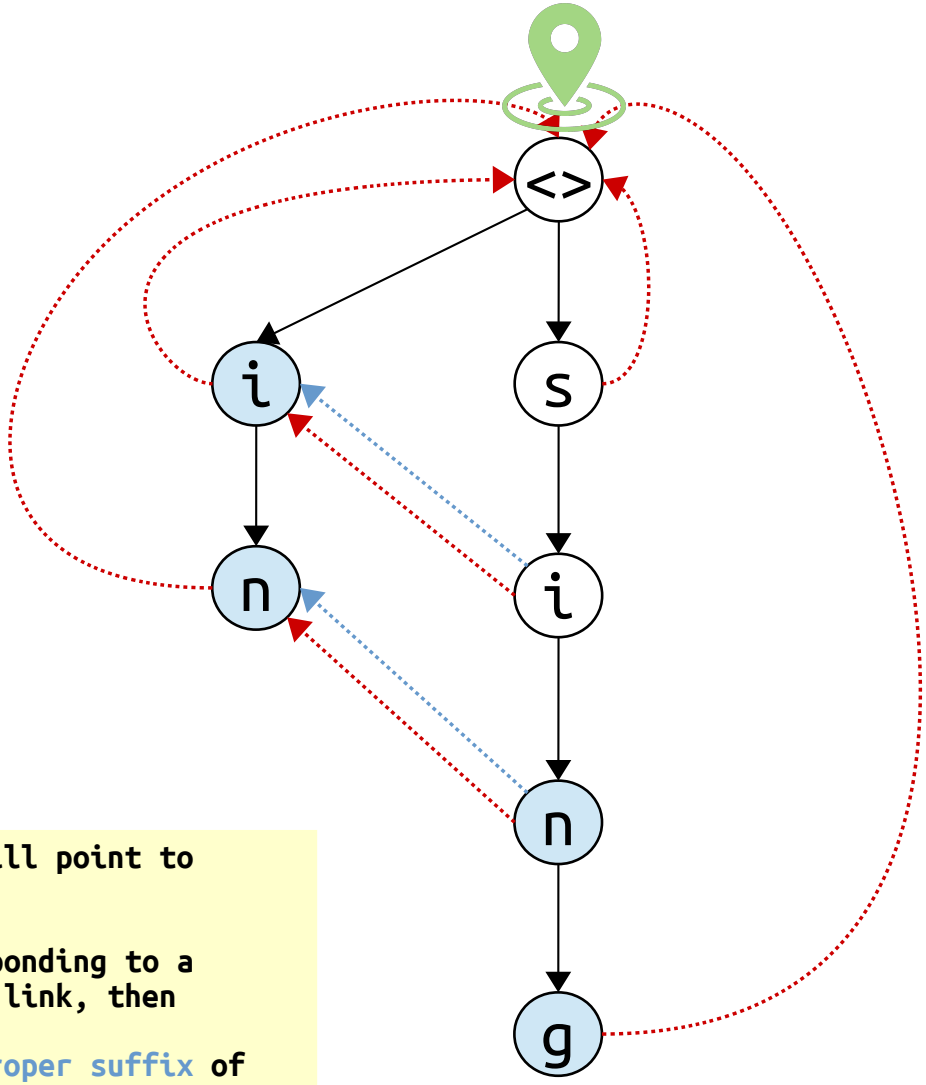
When we visit a node corresponding to a
string A that has an output link, then

It means that the longest proper suffix of
string A is a pattern in the trie, so we
“print” it.

Aho-Corasick – Output Links

patterns:

i			
i	n		
s	i	n	
s	i	n	g



The output link of a node will point to A endOfWord node.

When we visit a node corresponding to a string A that has an output link, then

It means that the longest proper suffix of string A is a pattern in the trie, so we "print" it.

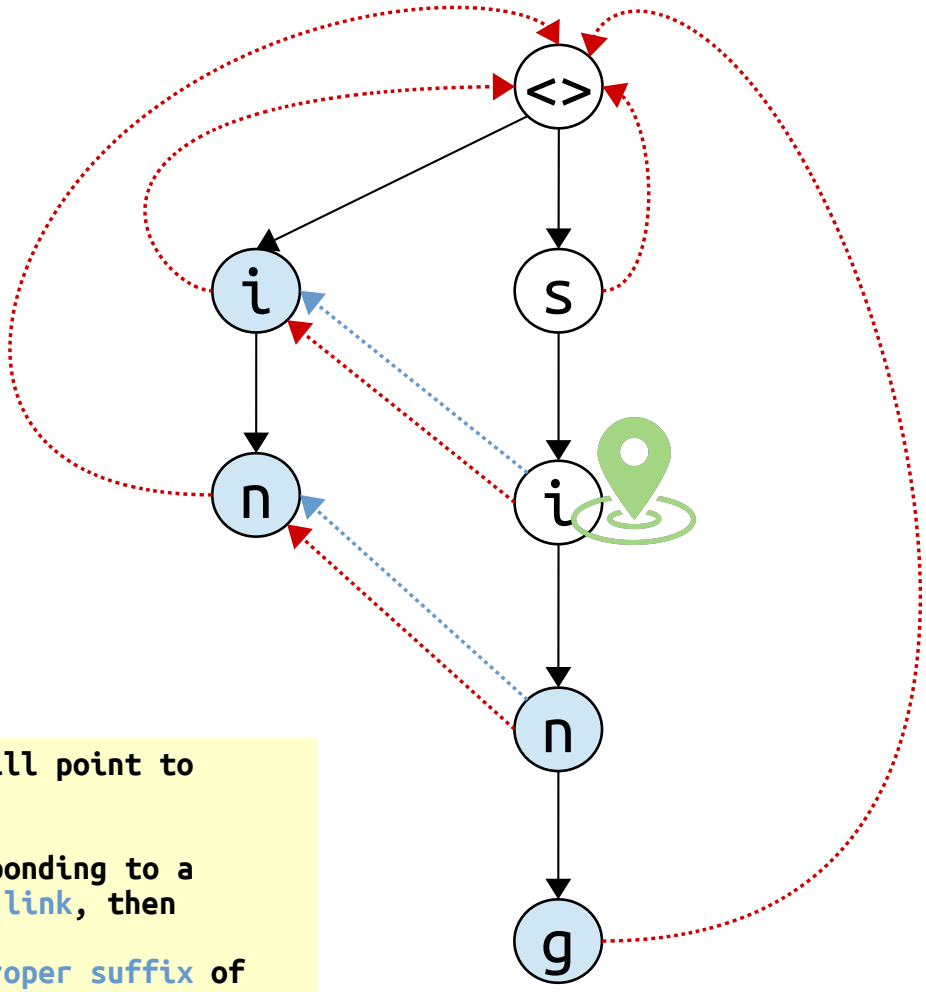
text:

s	i	n	g

Aho-Corasick – Output Links

patterns:

i			
i	n		
s	i	n	
s	i	n	g



The output link of a node will point to
A endOfWord node.

When we visit a node corresponding to a
string A that has an output link, then

It means that the longest proper suffix of
string A is a pattern in the trie, so we
“print” it.

pattern “i” found in the text

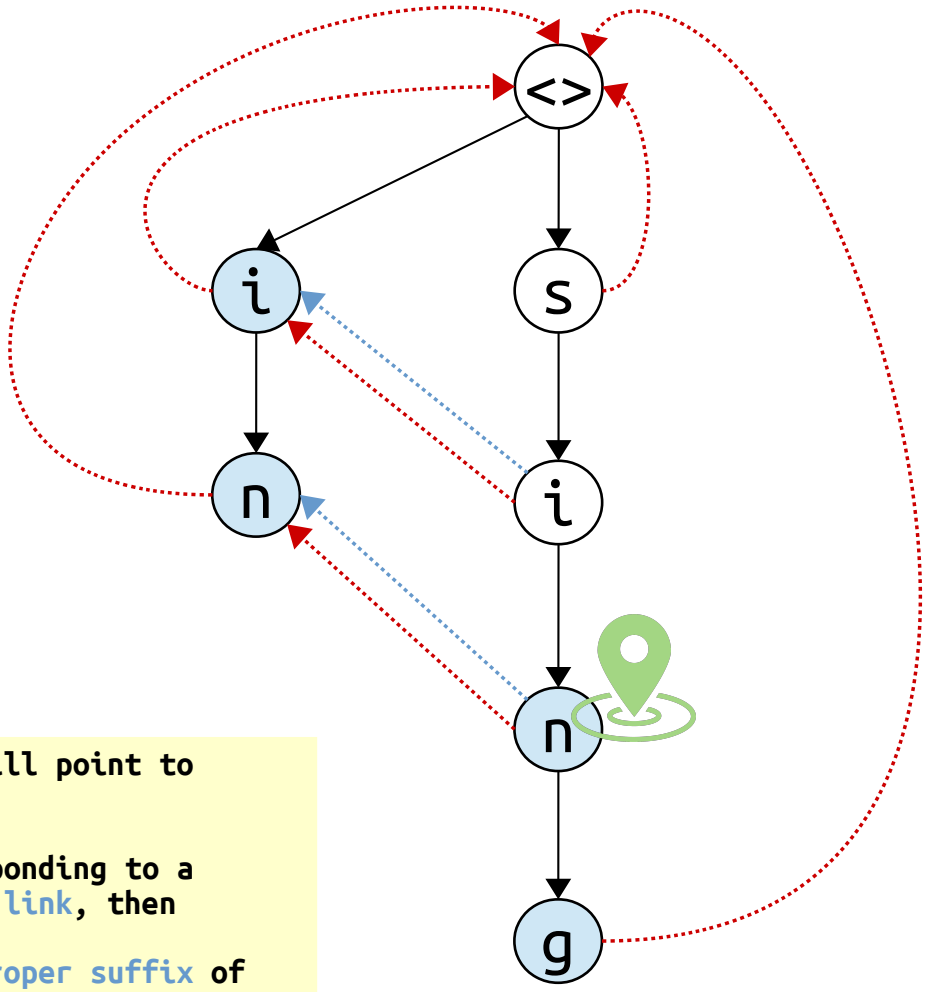
text:

s	i	n	g

Aho-Corasick – Output Links

patterns:

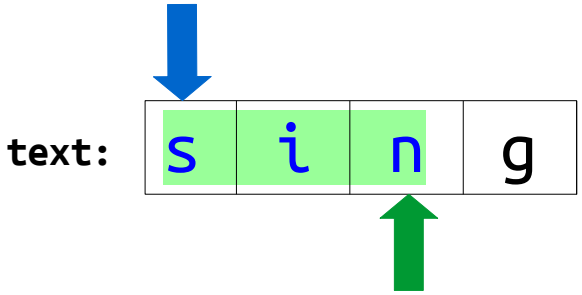
i			
i	n		
s	i	n	
s	i	n	g



The output link of a node will point to
A endOfWord node.

When we visit a node corresponding to a
string A that has an output link, then

It means that the longest proper suffix of
string A is a pattern in the trie, so we
"print" it.



pattern "i" found in the text

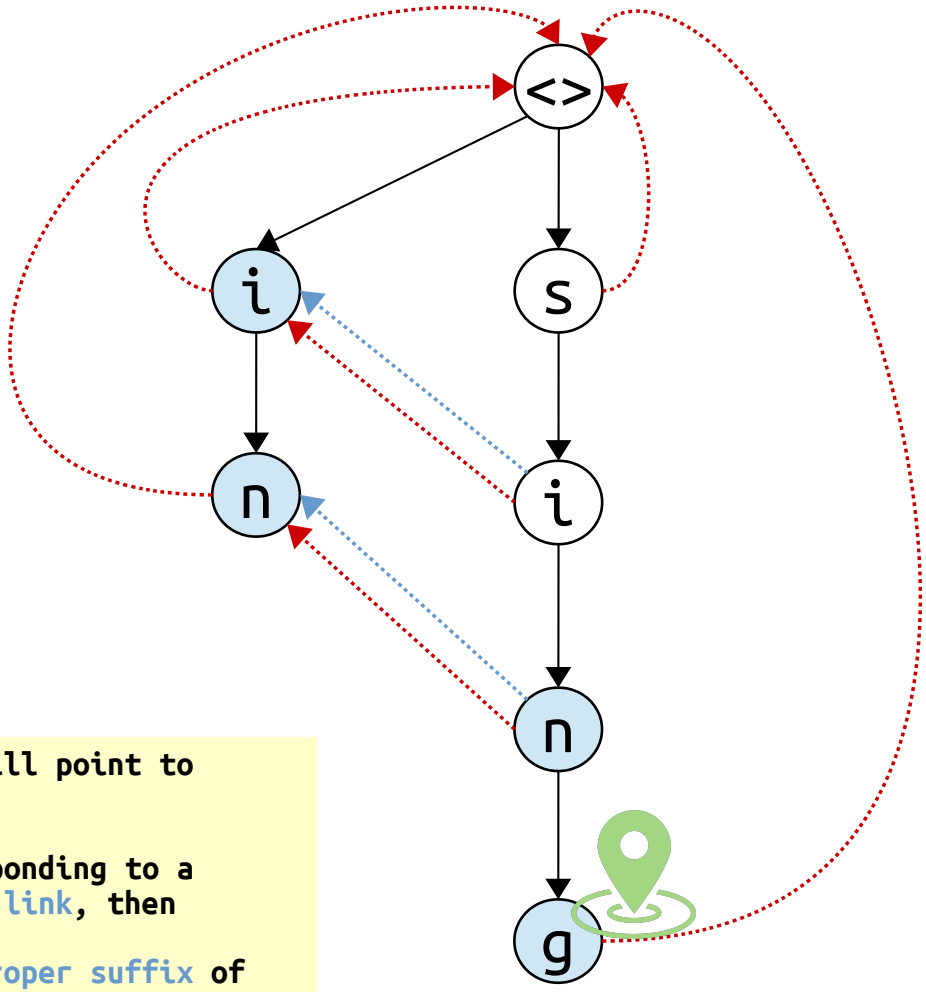
pattern "in" found in the text

pattern "sin" found in the text

Aho-Corasick – Output Links

patterns:

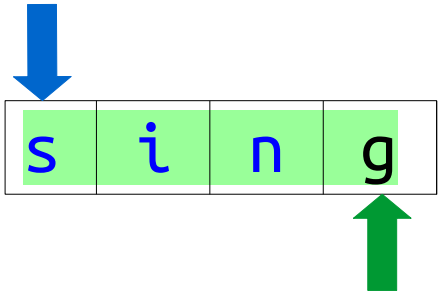
i			
i	n		
s	i	n	
s	i	n	g



The output link of a node will point to A endOfWord node.

When we visit a node corresponding to a string A that has an output link, then

It means that the longest proper suffix of string A is a pattern in the trie, so we "print" it.



pattern "i" found in the text

pattern "in" found in the text

pattern "sin" found in the text

pattern "sing" found in the text

Important

Aho-Corasick – sketch of the find() method

Find in a text all occurrences of patterns stored in a trie

```
Start with the root of the trie.
• For each character a in the text string:
    - While the current node has no child labeled a:
        • if the current node is the root, break out the loop;
        • otherwise, move to the current node to the node pointed by its suffix link.
    - If the current node has child a, move the current node to a.
    - If the current node is a pattern string, output the pattern.
    - Output all patterns in the chain of output links starting from the current node.
```

Aho-Corasick – The Complete Algorithm

```
Step 1. Build a trie from pattern strings

Step 2. Add suffix links to the trie

Step 3. Add output links to the trie

Step 4.Execute Aho-Corasick find() method with an input text
```

Our Plan

(1) The rest of this seminar

- How to construct suffix links in a trie

(2) Homework for next Tuesday 02.11.2021
(Optional, full value of 1 seminar)

- Code the algorithm for constructing suffix links in a trie.

(3) Seminar of Tuesday 02.11.2021

- Review the (optional) homework

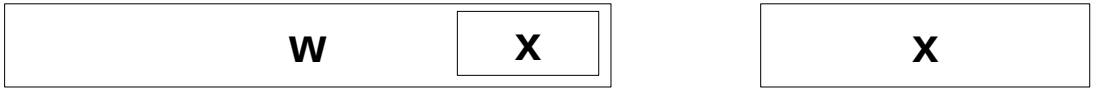
- Include output links in the code

- Code Aho-Corasick find() method (slide 58)

Step 2 -
Add Suffix Links in a Trie

Aho-Corasick – Algorithm to Build Suffix Links

- Suppose we know the suffix link for a node corresponding to **string w**, which leads to node corresponding to string **x**.

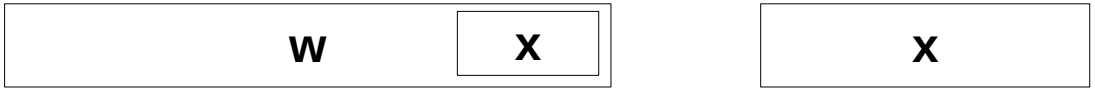


- Assume that exists node **wa**
  Consider to assign a suffix link to node **xa** (if exists)

We have two cases:

Aho-Corasick – Algorithm to Build Suffix Links

- Suppose we know the suffix link for a node corresponding to **string w**, which leads to node corresponding to string **x**.

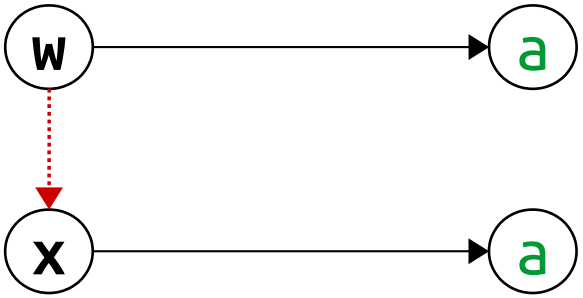


- Assume that exists node **wa**
  Consider to assign a suffix link from node **wa** to node **xa** (if exists)

We have two cases:

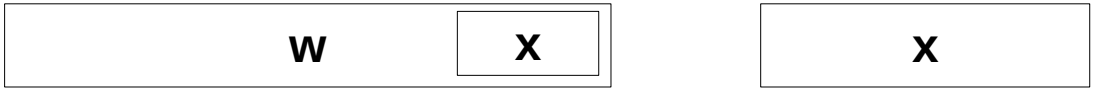
Case 1: string **xa** *exists* in the trie.
(Node for word **x** has child **a**)

Add **suffix link** from node **wa** to node **xa**



Aho-Corasick – Algorithm to Build Suffix Links

- Suppose we know the suffix link for a node corresponding to **string w**, which leads to node corresponding to string **x**.

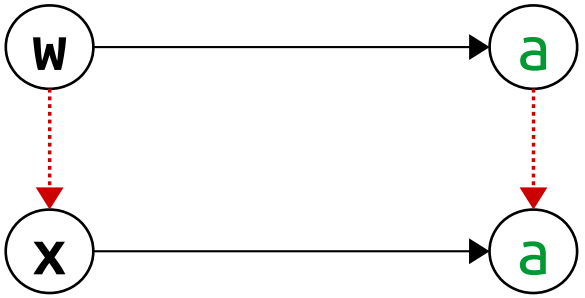


- Assume that exists node **wa**
  Consider to assign a suffix link from node **wa** to node **xa** (if exists)

We have two cases:

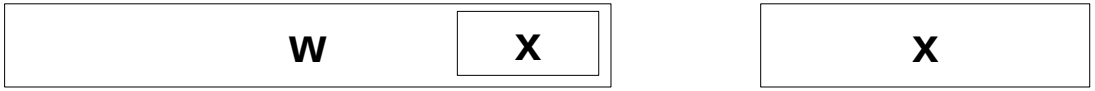
Case 1: string **xa** *exists* in the trie.
(Node for word **x** has child **a**)

Add **suffix link** from node **wa** to node **xa**



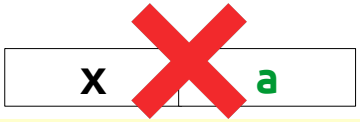
Aho-Corasick – Algorithm to Build Suffix Links

- Suppose we know the suffix link for a node corresponding to **string w**, which leads to node corresponding to string **x**.



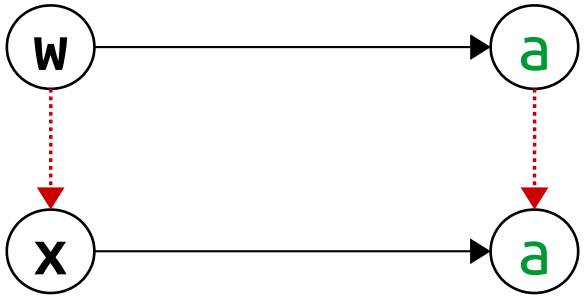
- Assume that exists node **wa**
  Consider to assign a suffix link from node **wa** to node **xa** (if exists)

We have two cases:



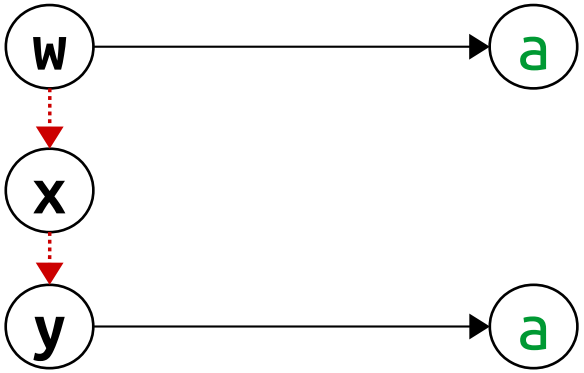
Case 1: string **xa** *exists* in the trie.
(Node for word **x** has child **a**)

Add **suffix link** from node **wa** to node **xa**



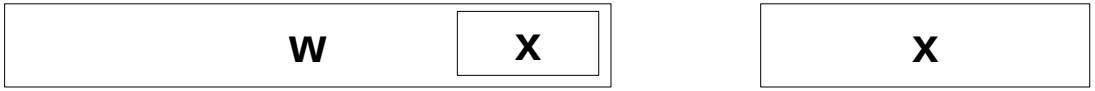
Case 2: string **xa** doesn't exist in the trie.
(Node for word **x** has not child **a**)

Check if **y** = **x**'s **suffix link** has node **ya**



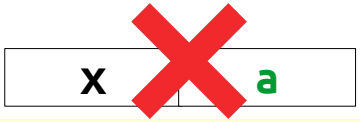
Aho-Corasick – Algorithm to Build Suffix Links

- Suppose we know the suffix link for a node corresponding to **string w**, which leads to node corresponding to string **x**.



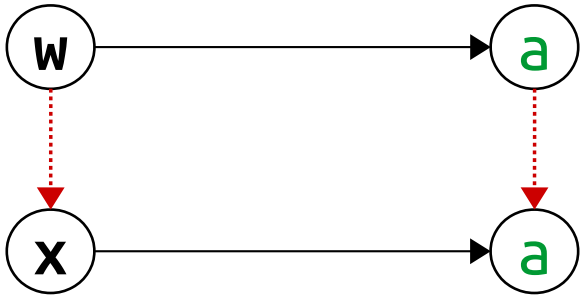
- Assume that exists node **wa**
  Consider to assign a suffix link from node **wa** to node **xa** (if exists)

We have two cases:



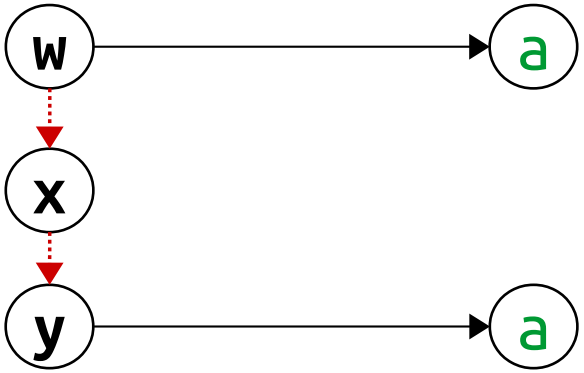
Case 1: string **xa** *exists* in the trie.
(Node for word **x** has child **a**)

Add **suffix link** from node **wa** to node **xa**



Case 2: string **xa** doesn't exist in the trie.
(Node for word **x** has not child **a**)

Check if **y** = **x**'s **suffix link** has node **ya**



Important

Aho-Corasick – Algorithm to Build Suffix Links

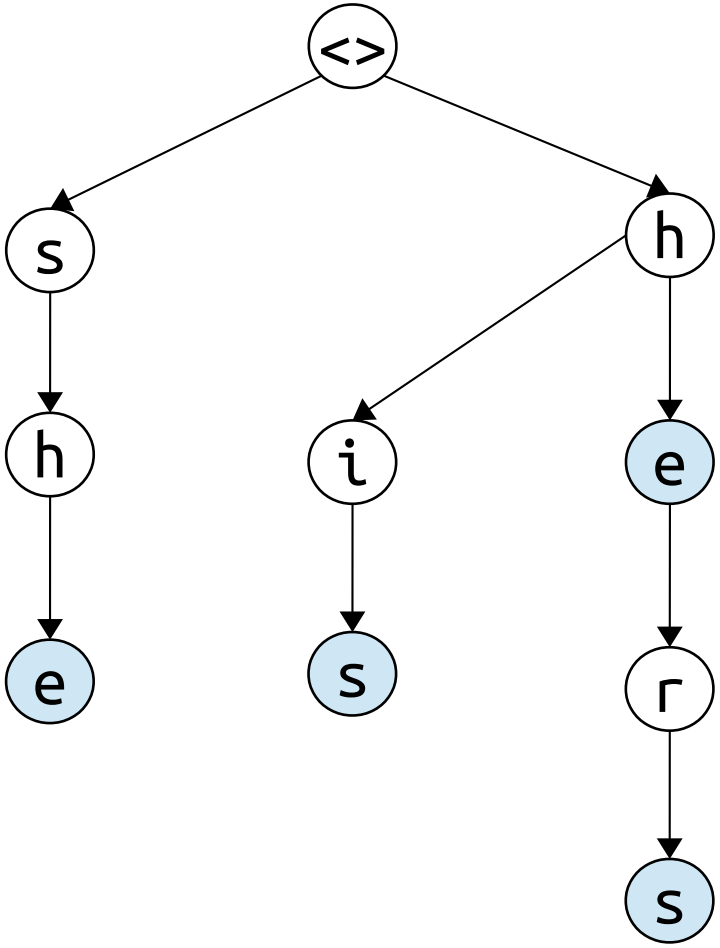
```
suffix link of the root is nullptr.

For each child i of the root
    suffix link of root->child[i] is the root.

Visit each node w of the trie in "level-order" (except the root)

    For each non-null child a of node w:

        /*** Create suffix link for node wa ***/

        Let node x be the suffix link of w

        While (node x is not nullptr) and (x has not child a):
            x = suffix link of x;

        If x is nullptr:
            suffix link of node wa = root of the trie
        Else:
            suffix link of node wa = child a of x
```

Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

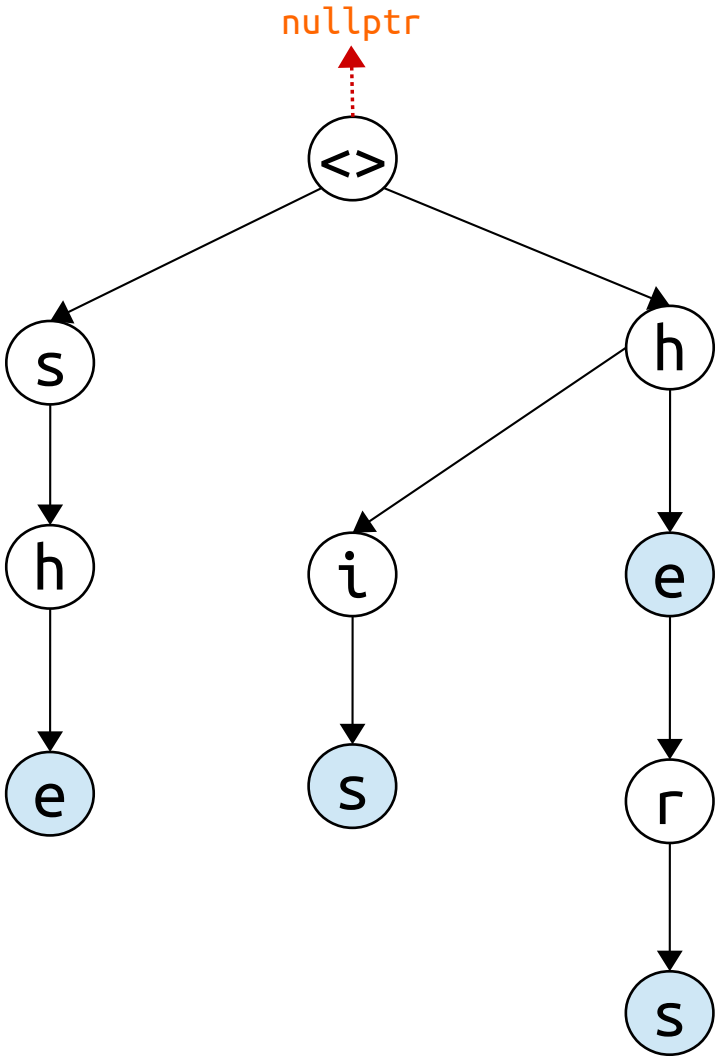


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

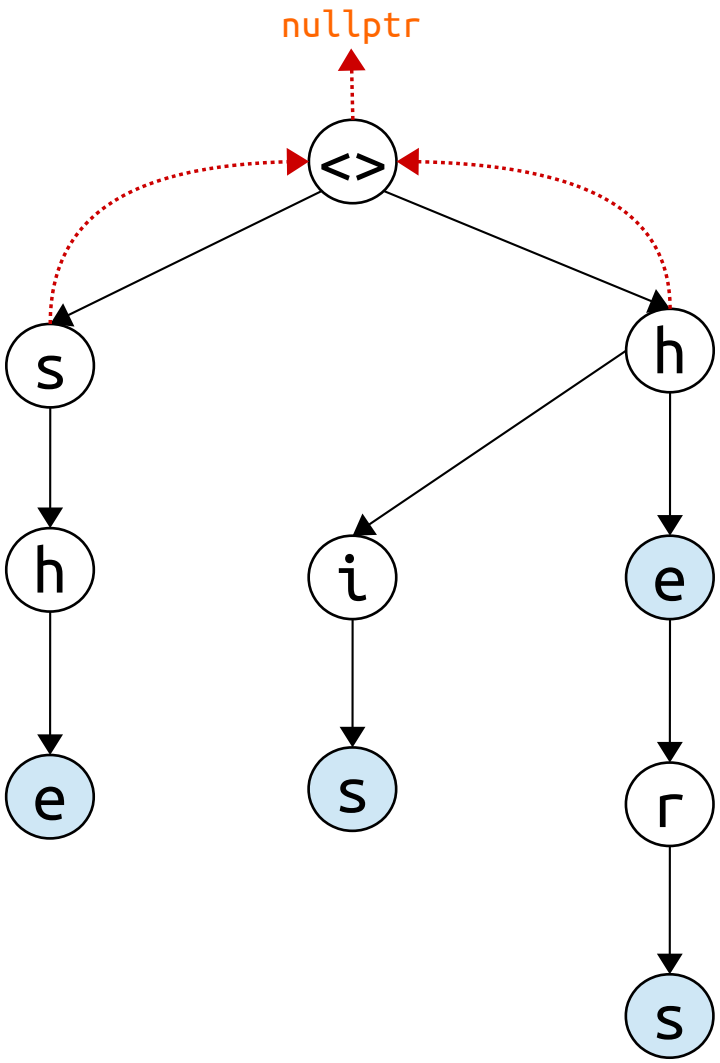


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

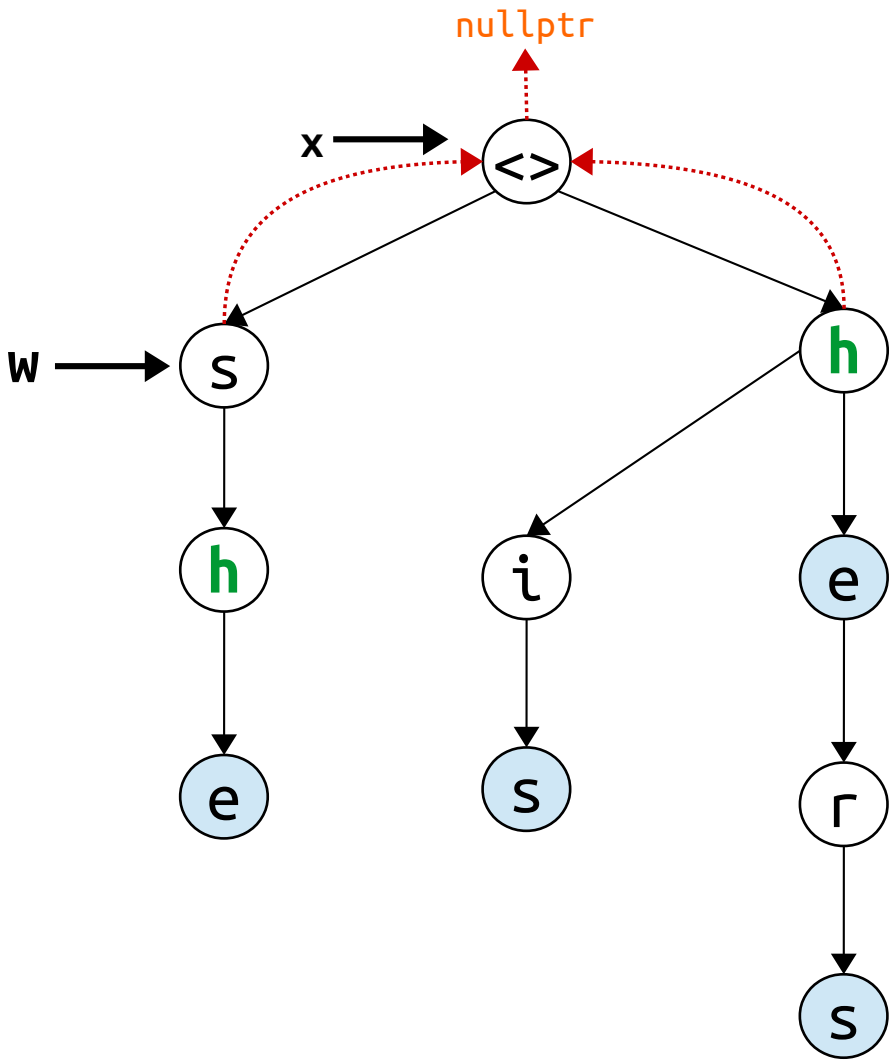


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

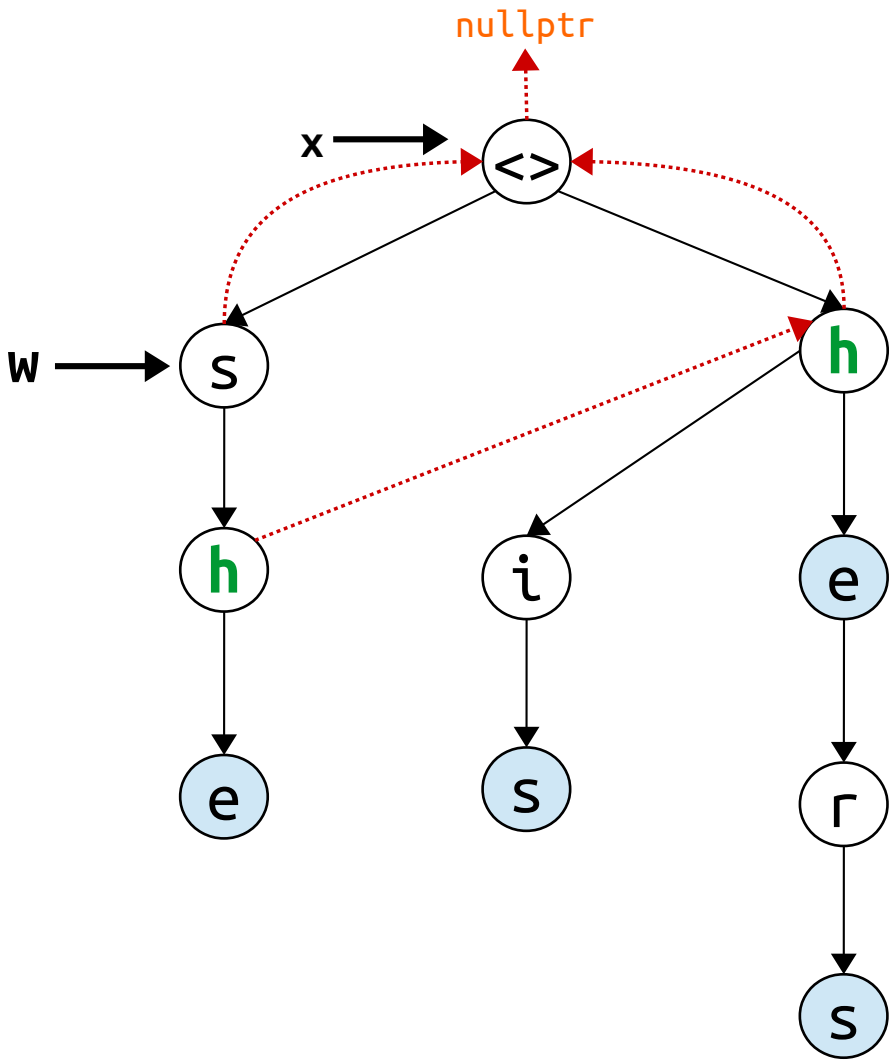


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

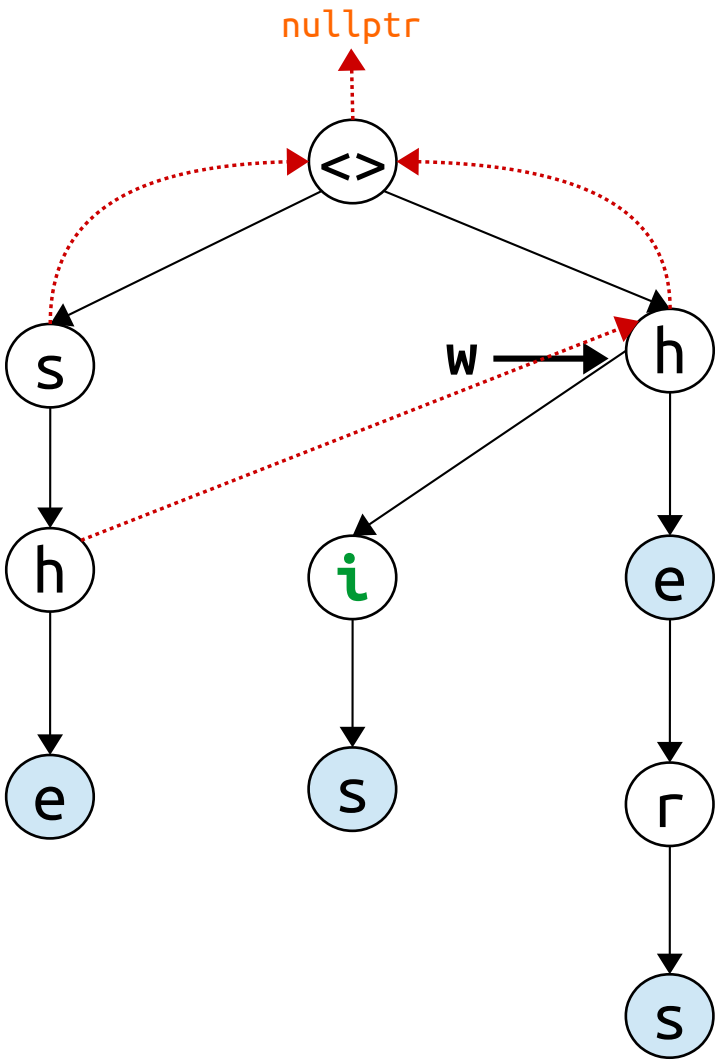


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

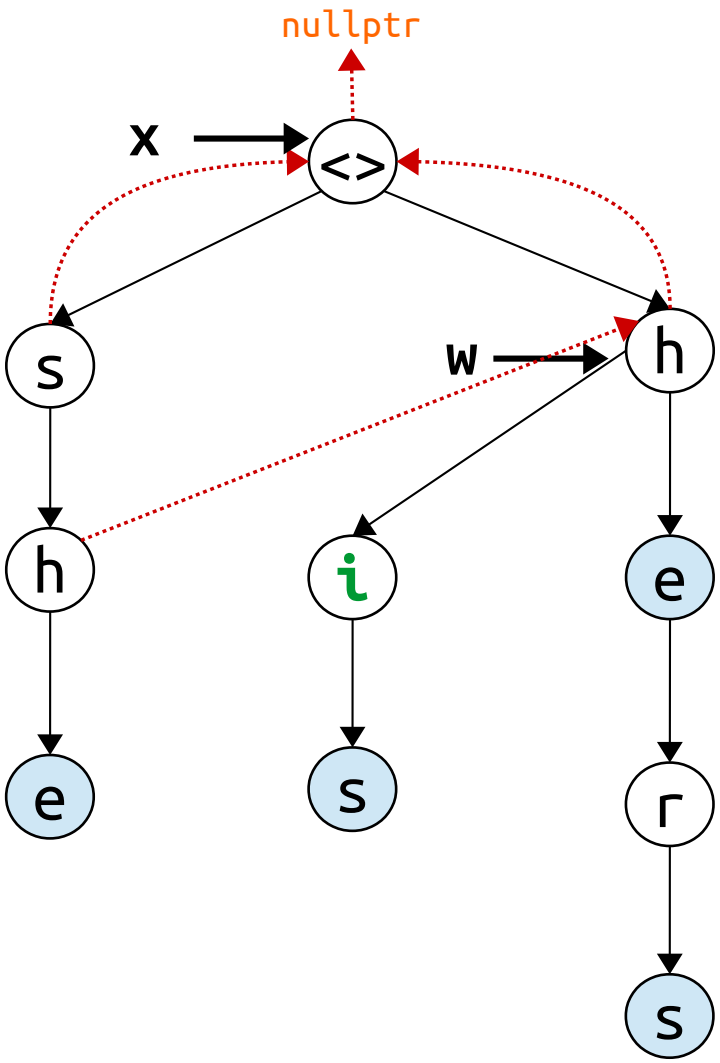


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

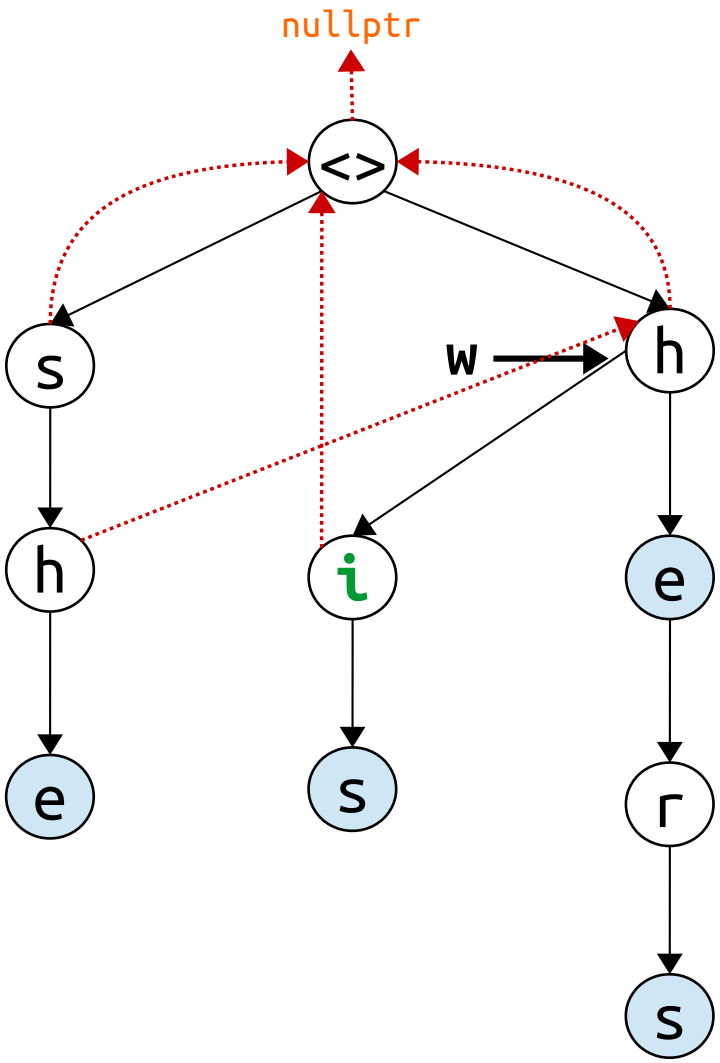


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

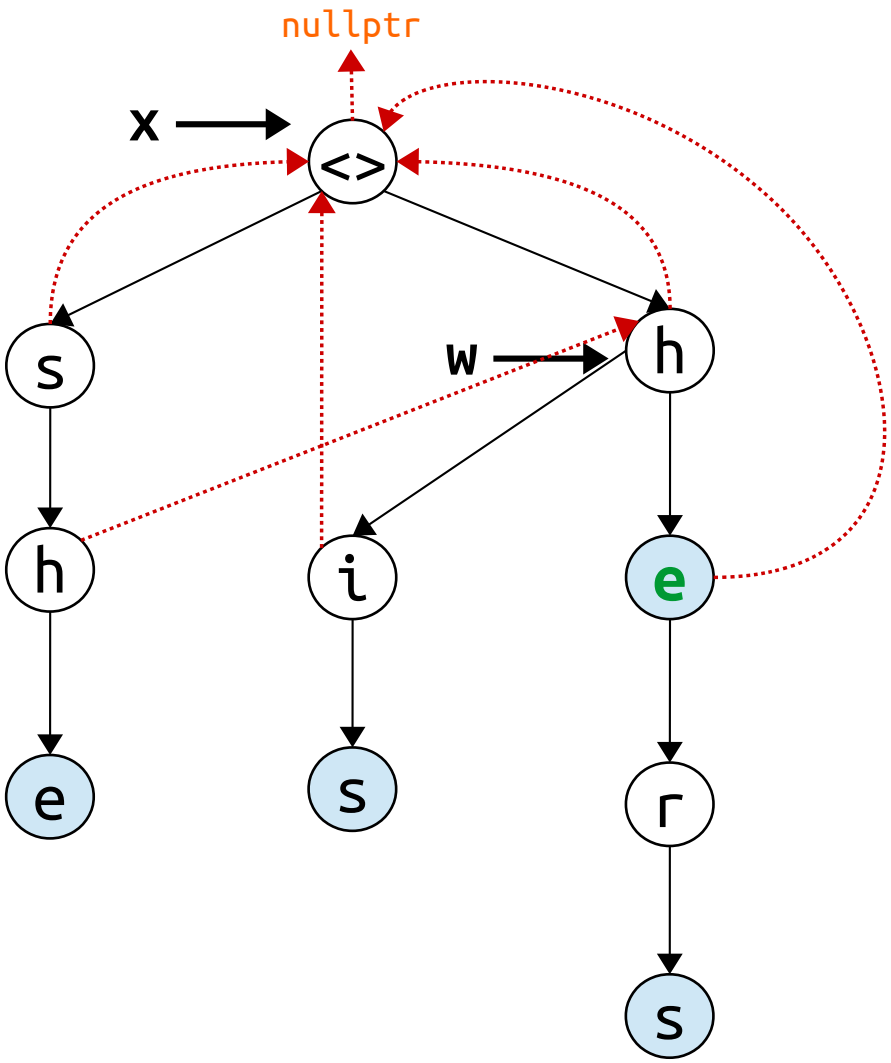


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

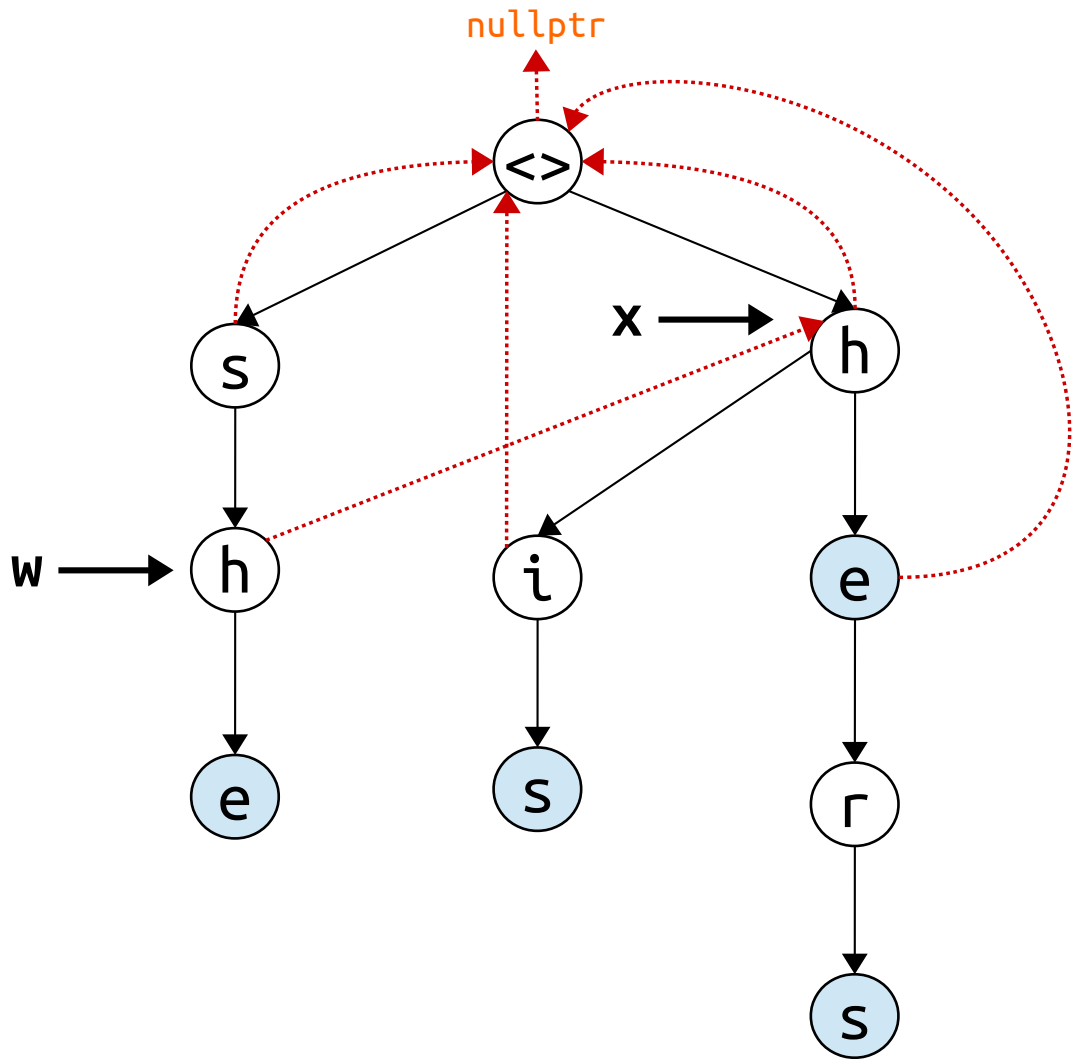


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

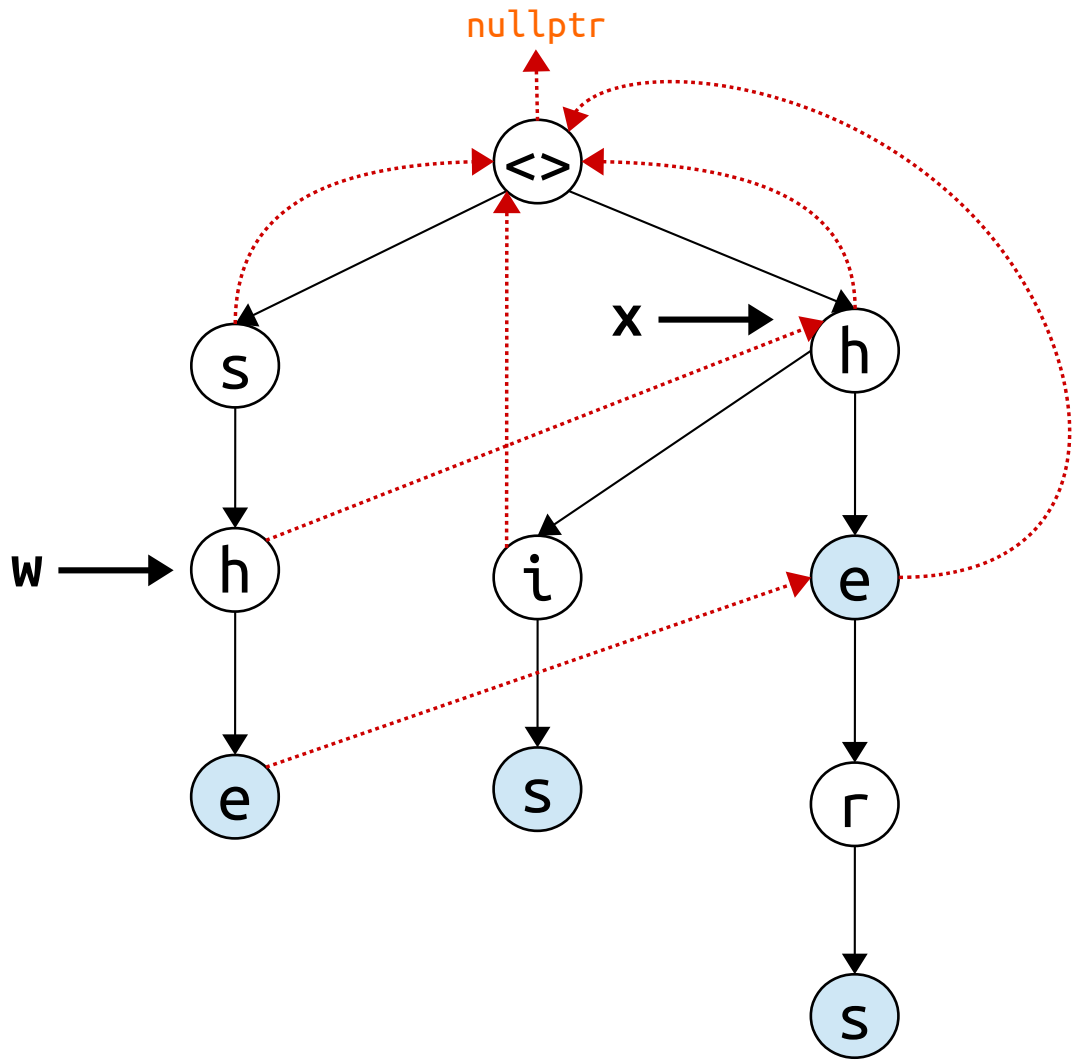


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

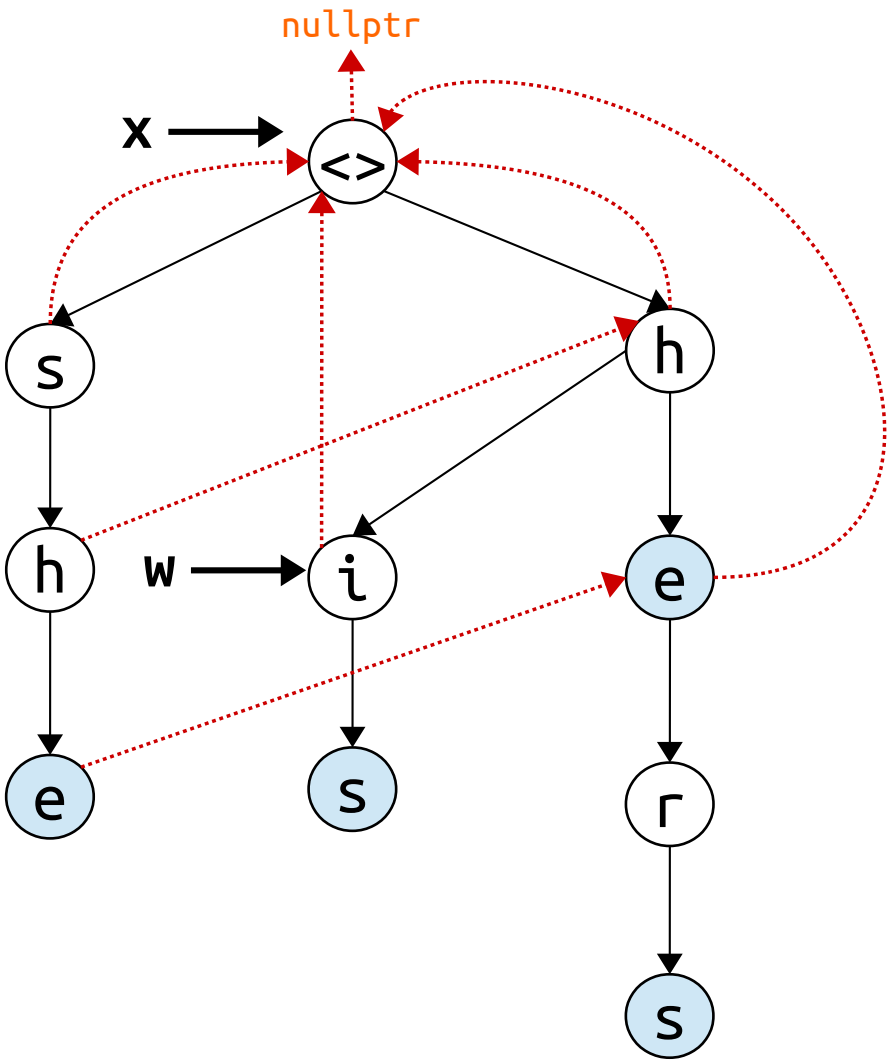


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

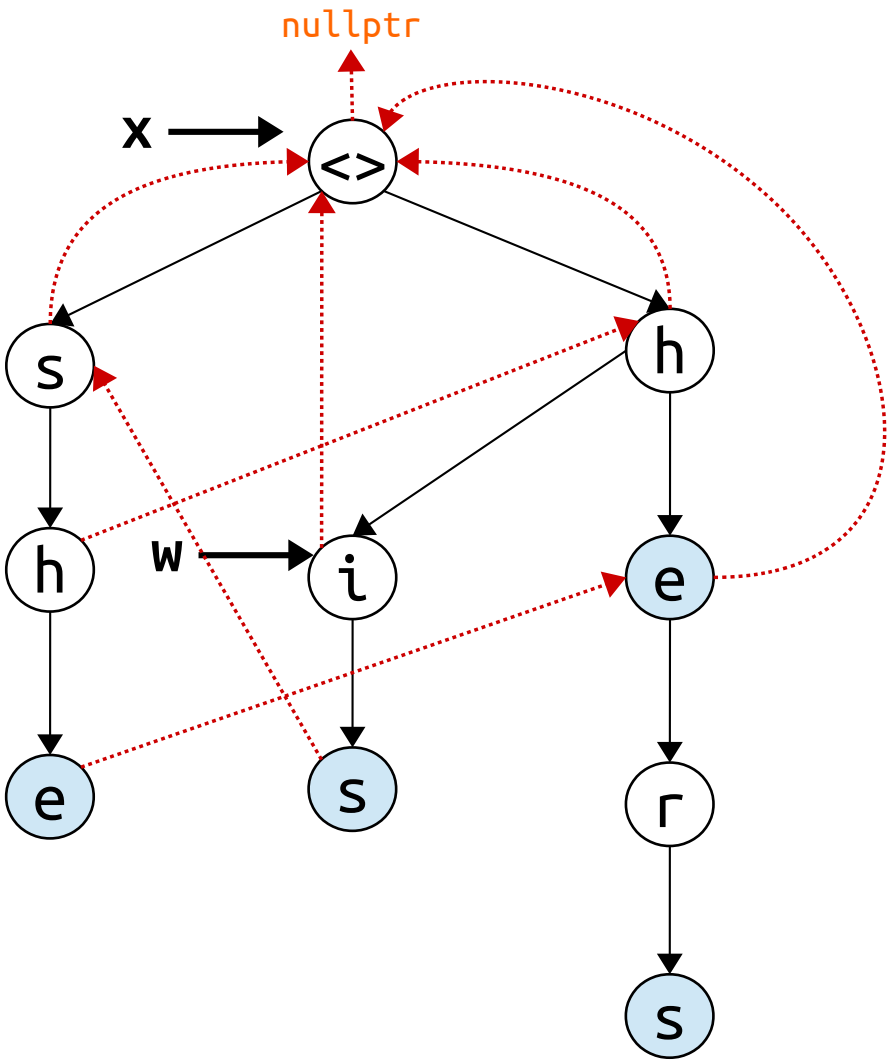


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

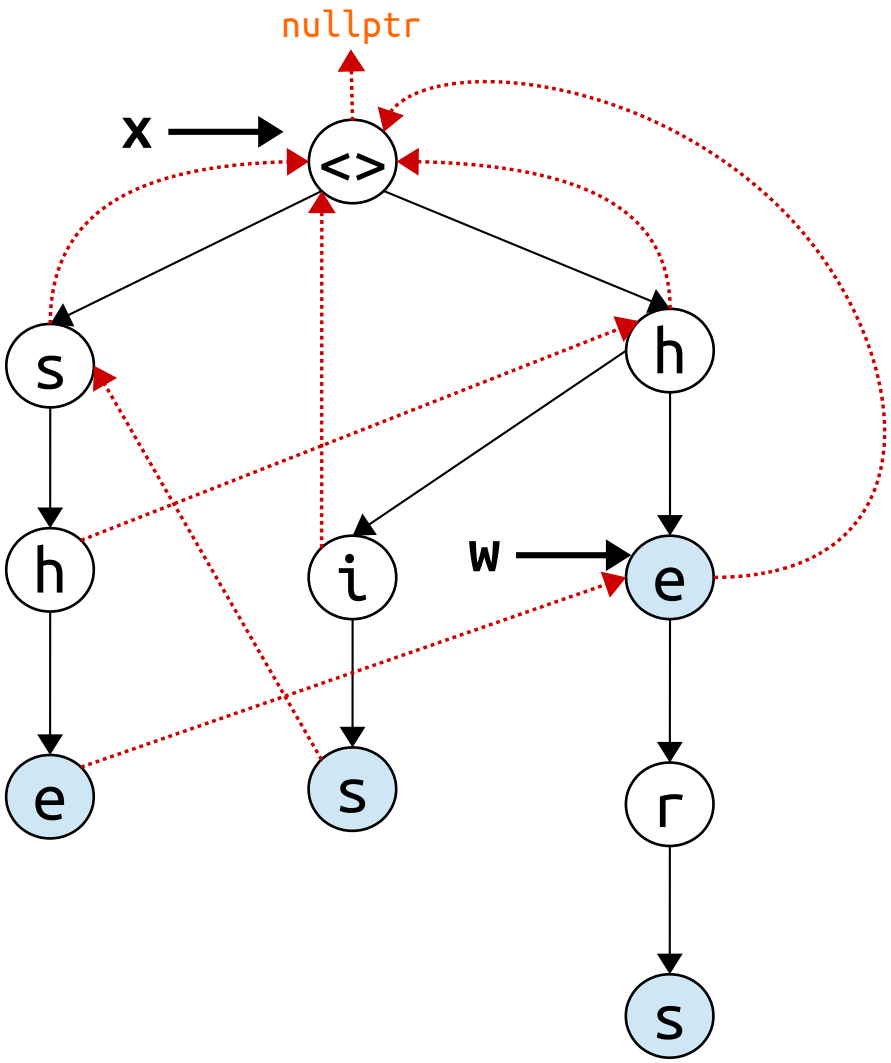


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

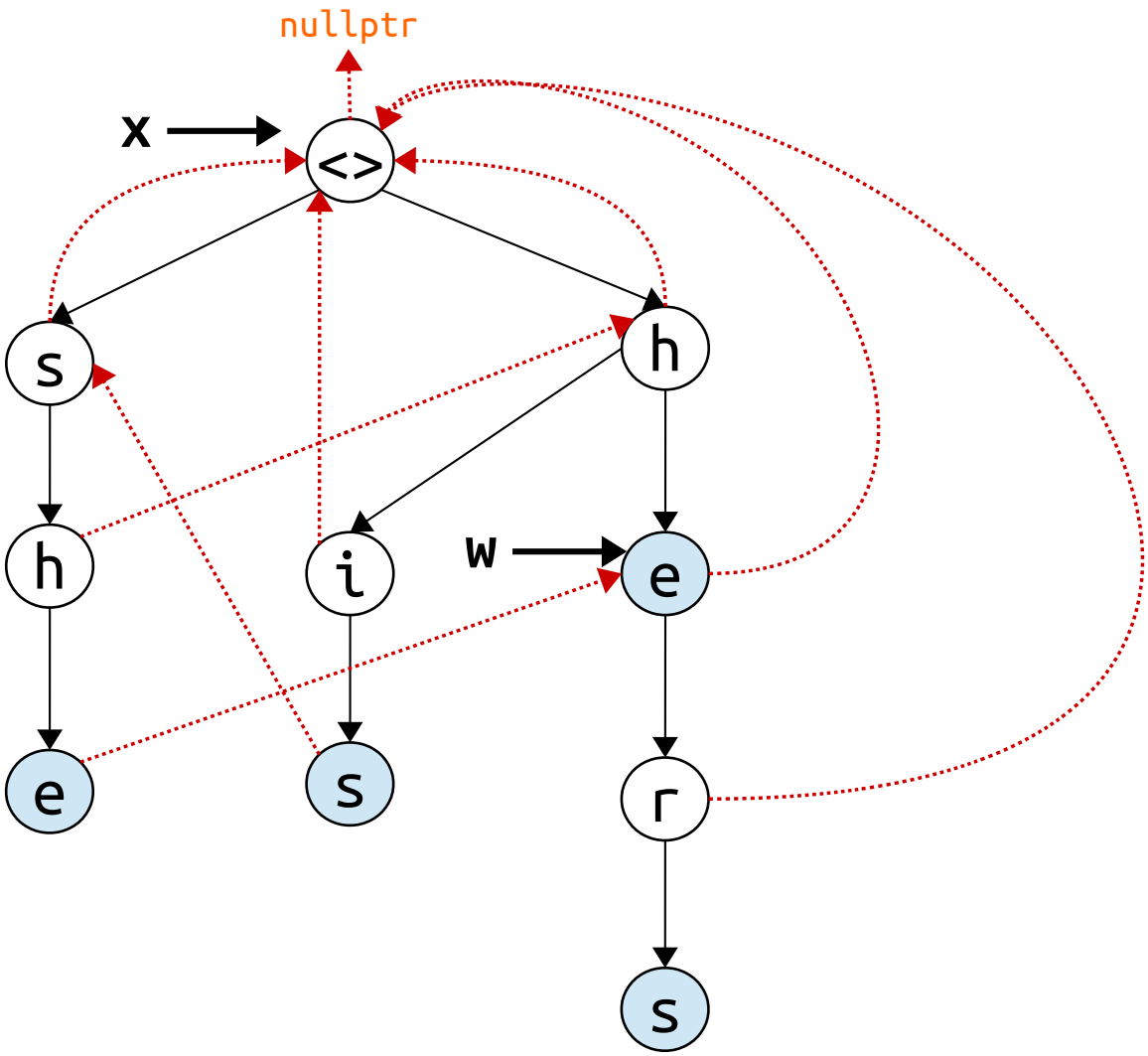


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

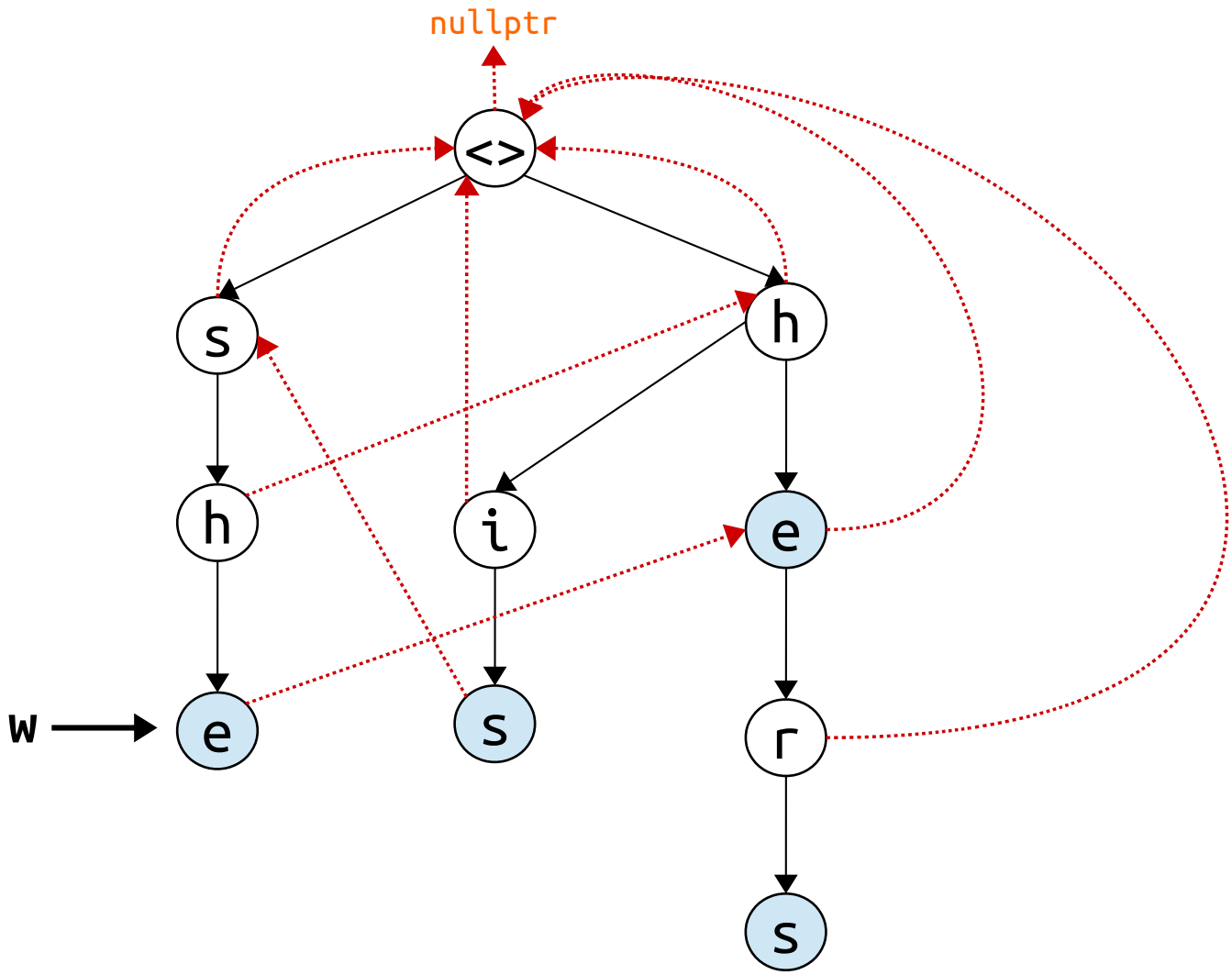


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

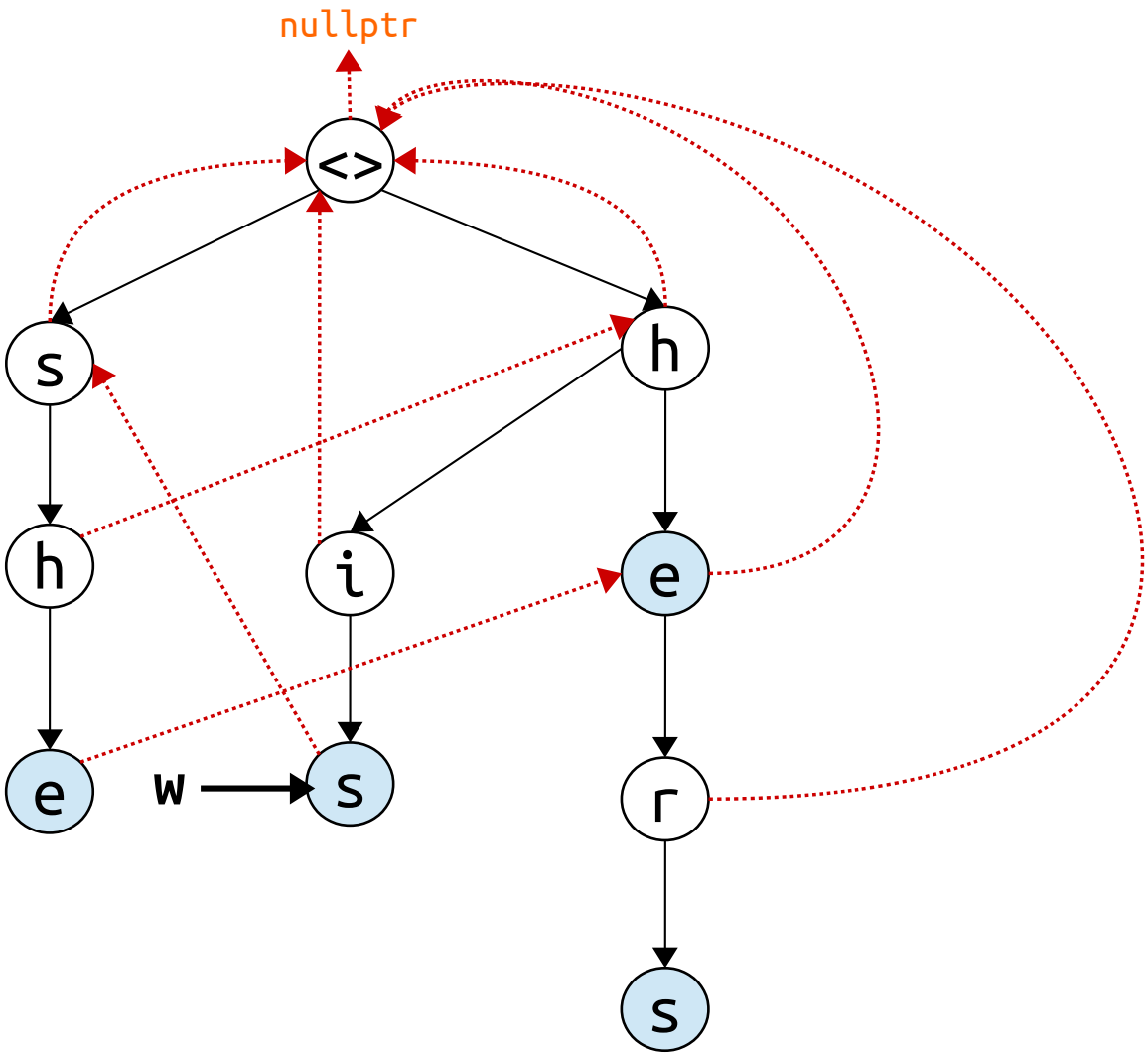


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

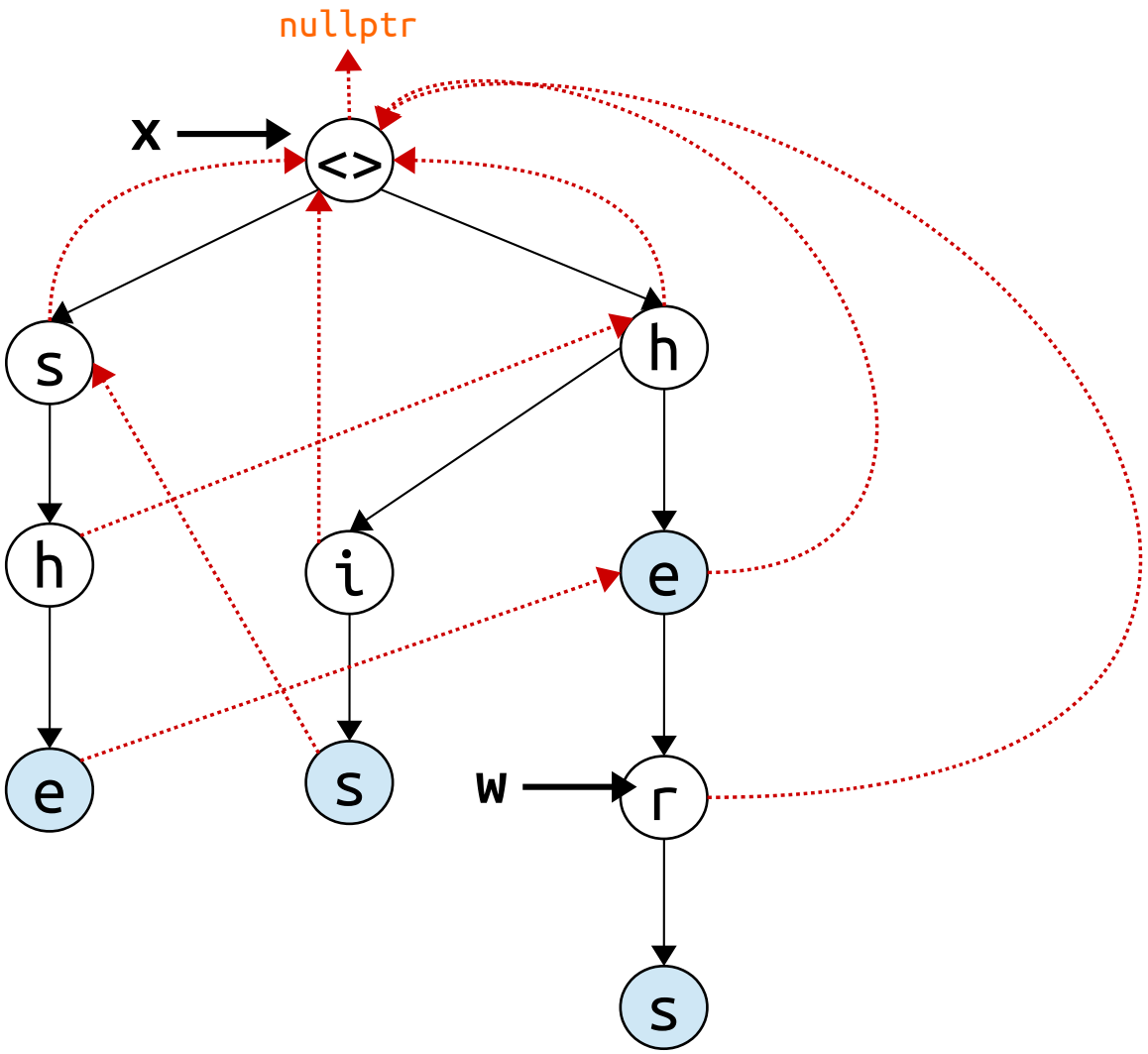


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

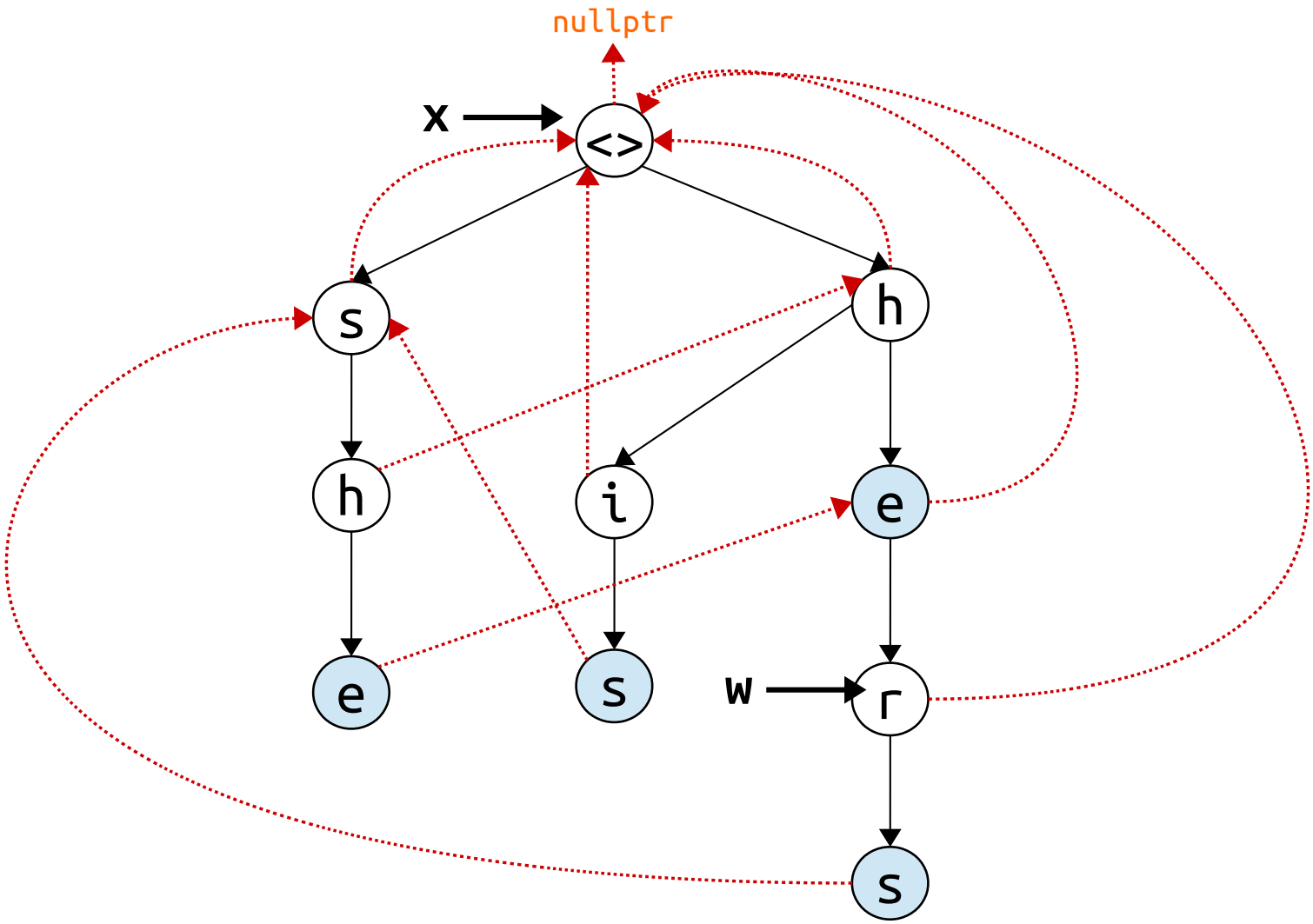


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

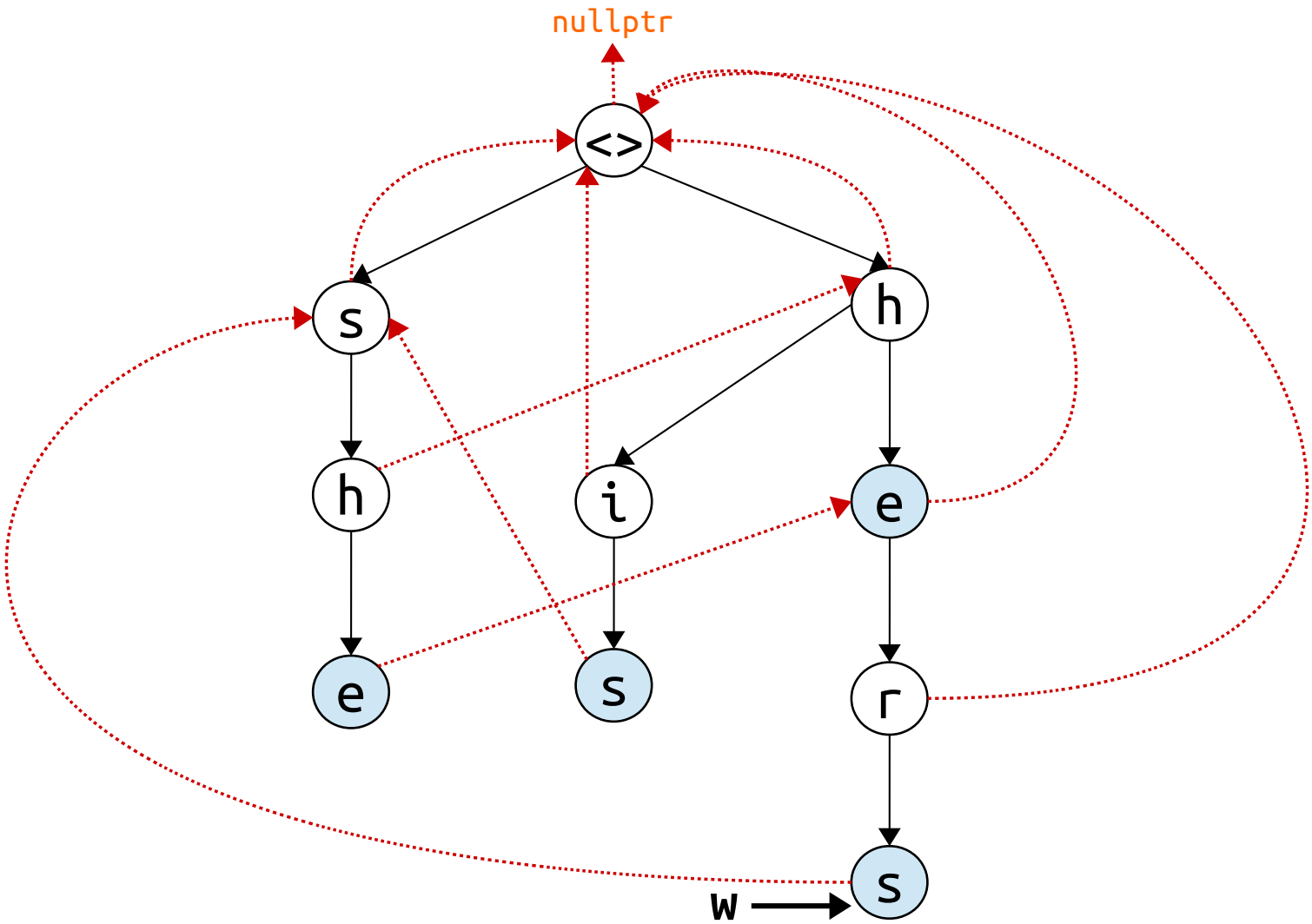


Aho-Corasick – Algorithm to Build Suffix Links

Example

patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	

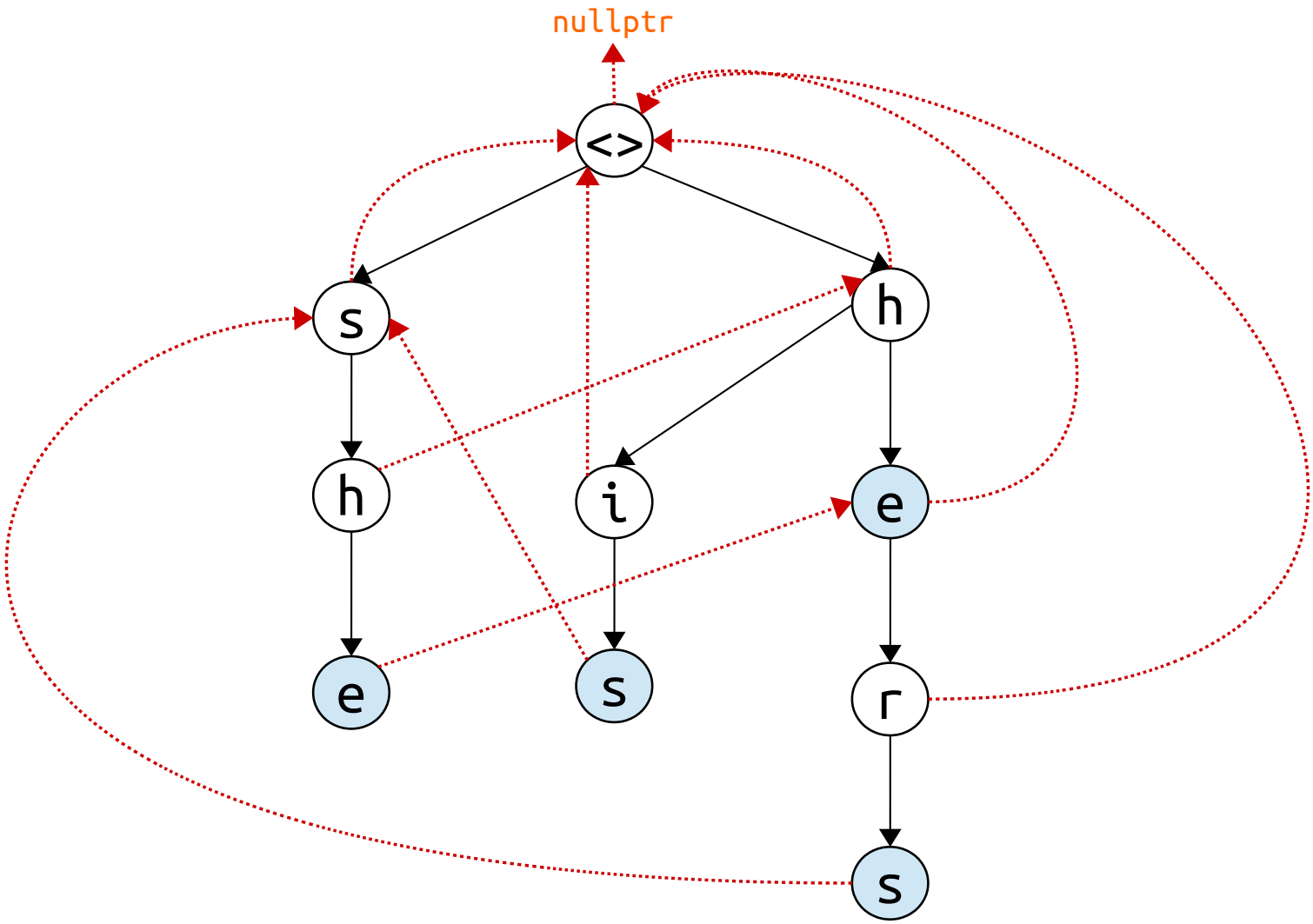


Aho-Corasick – Algorithm to Build Suffix Links

Example

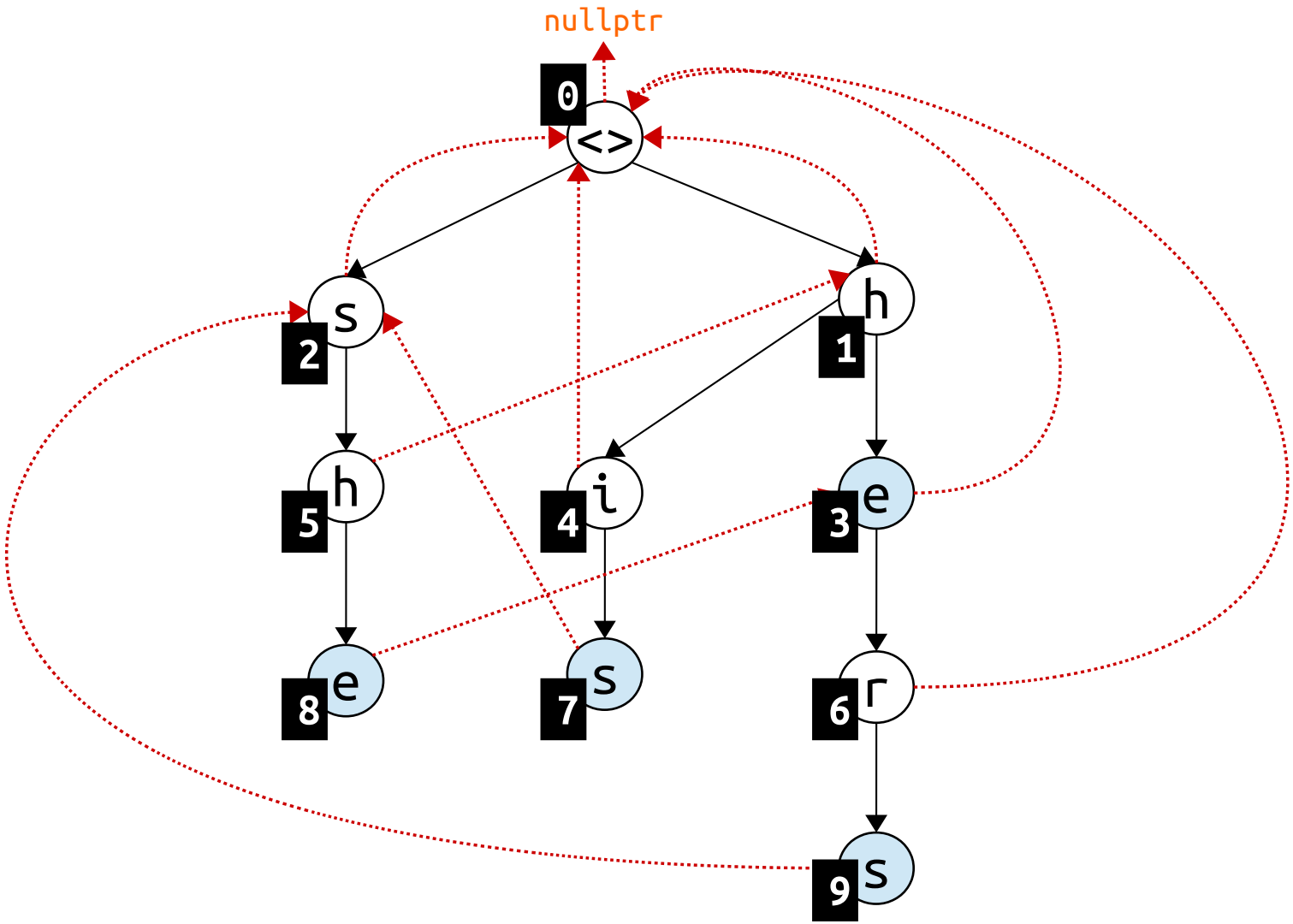
patterns:

h	e		
h	i	s	
h	e	r	s
s	h	e	



Homework (see slide 60)

Output example



SUFFIX LINKS

```
1 : h --> 0 : _

2 : s --> 0 : _

3 : e --> 0 : _

4 : i --> 0 : _

5 : h --> 1 : h

6 : r --> 0 : _

7 : s --> 2 : s

8 : e --> 3 : e

9 : s --> 2 : s
```