MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT2004
Algorithms and Data Structures

Ian Wern Han Lim
lim.wern.han@monash.edu

Referencing materials by
Nathan Companez, Aamir Cheema, Arun Konagurthu and Lloyd Allison

GROUP OF EIGHT AUSTRALIA

Faculty of Information Technology, Monash University

Ready?

Agenda

- String retrieval

Agenda

- String retrieval
- Tries and suffix tries

Let us begin…

- String retrieval is one of the oldest retrieval task in the world…

- String retrieval is one of the oldest retrieval task in the world…

- Anything can be represented as a string

- String retrieval is one of the oldest retrieval task in the world…

- Anything can be represented as a string
  - DNA sequence
  - Images (RGB)
  - Keys
  - … and many more!

- So how can we search for string very fast?

- So how can we search for string very fast?
  - Sort the strings
  - Binary search for what you want

- So how can we search for string very fast?
  - Sort the strings
  - Binary search for what you want

  - What is our complexity?
    - N = number of strings
    - M = average length the string (instead of the longest)

- So how can we search for string very fast?
  - Sort the strings
    - O(MN log N) using merge sort… because O(M) for string comparison
    - O(MN) using radix sort
  - Binary search for what you want

  - What is our complexity?
    - N = number of strings
    - M = average length the string (instead of the longest)

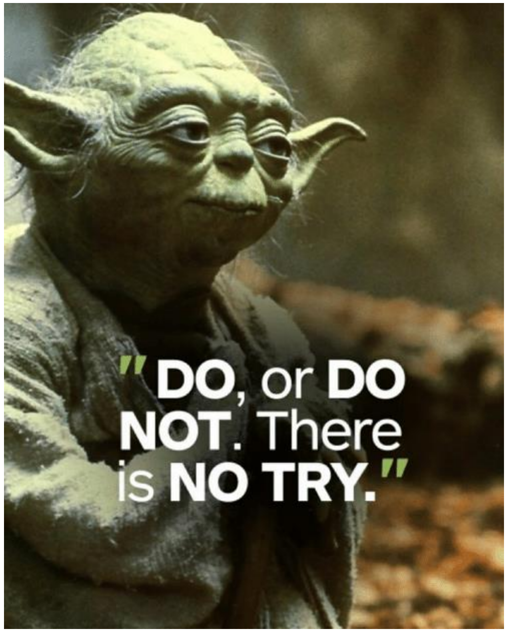
- So how can we search for string very fast?
  - Sort the strings
    - O(MN log N) using merge sort… because O(M) for string comparison
    - O(MN) using radix sort
  - Binary search for what you want
    - O(M log N)… again O(M) for string comparison
  - What is our complexity?
    - N = number of strings
    - M = average length the string (instead of the longest)

Questions?

- When we search, we would need to go through every character of a string. Thus, we can use a special data structure that organise it according to characters…

- When we search, we would need to go through every character of a string. Thus, we can use a special data structure that organise it according to characters…

- We use reTRIEval tree

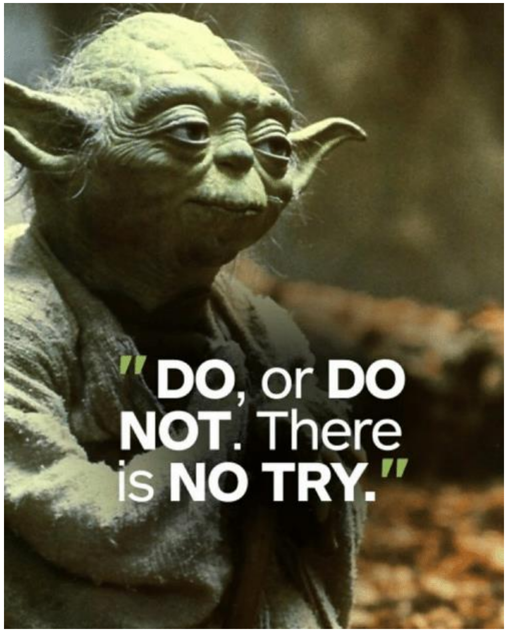


Tries

Efficient string retrieval

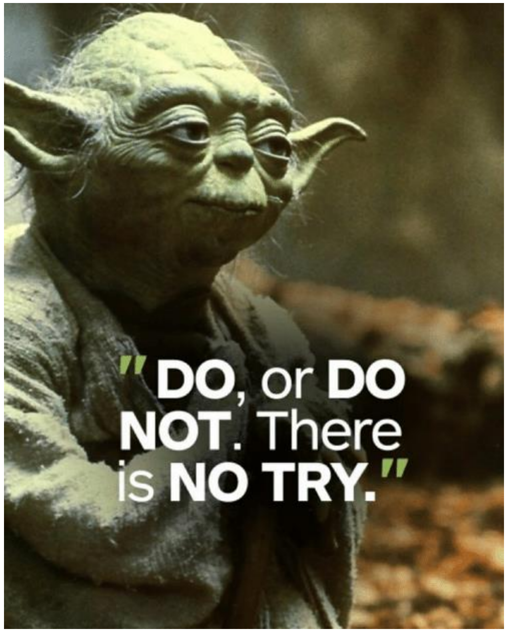
- When we search, we would need to go through every character of a string. Thus, we can use a special data structure that organise it according to characters…

- We use reTRIEval tree
  - A tree
  - M-child per node



- When we search, we would need to go through every character of a string. Thus, we can use a special data structure that organise it according to characters…

- We use reTRIEval tree
  – A tree
  – M-child per node 26 childs
    M = number of unique character



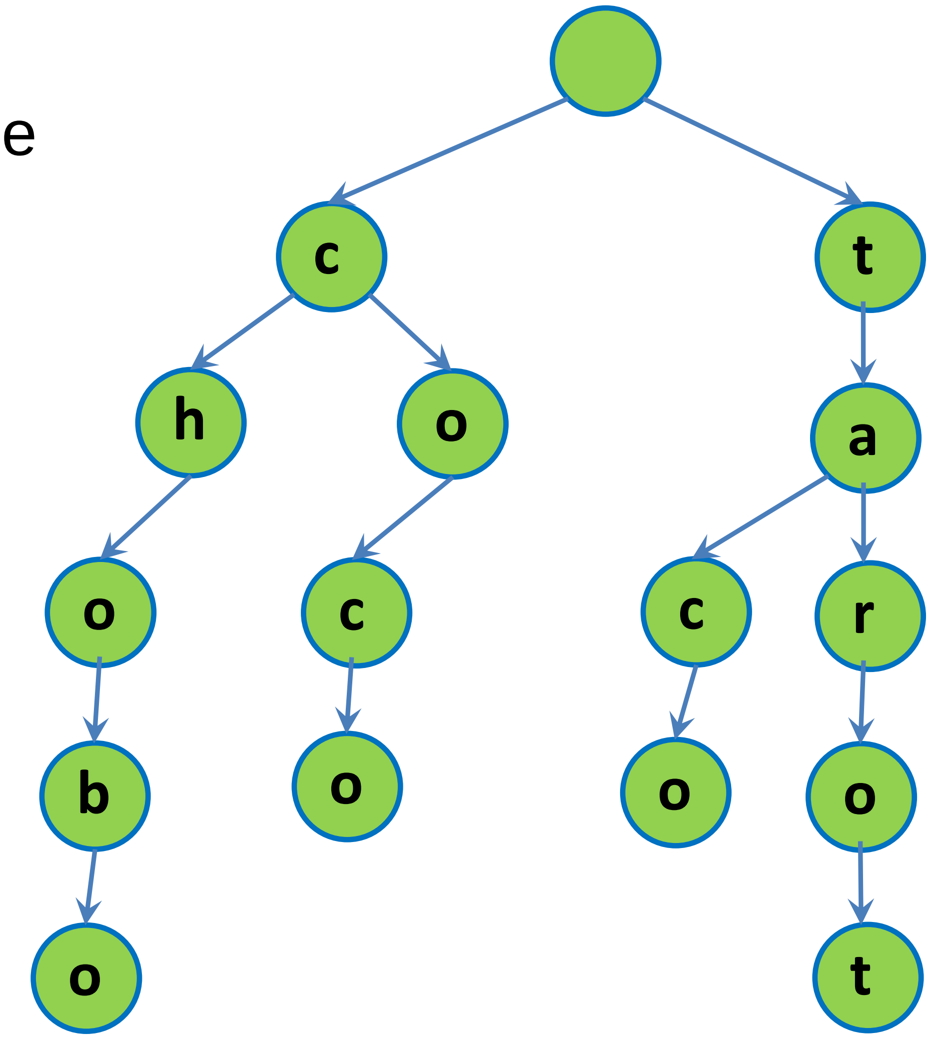
- Let assume we have the following words:
  - Taco
  - Taro
  - Tarot
  - Coco
  - Chobo

- Let assume we have the following words:
  - Taco
  - Taro
  - Tarot
  - Coco
  - Chobo
  - What is the trie?

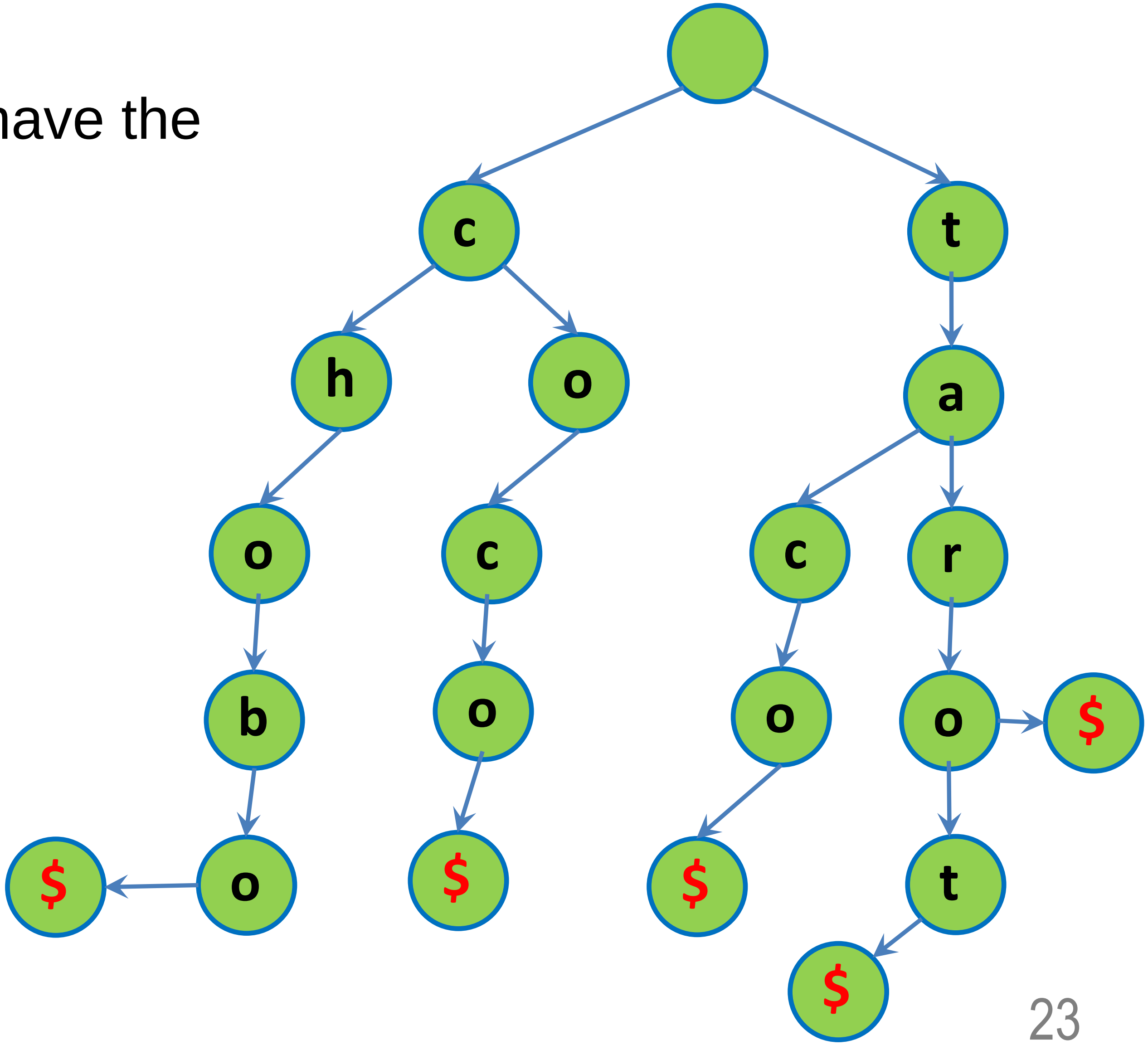
Tries

Efficient string retrieval

- Let assume we have the following words:
  - Taco
  - Taro
  - Tarot
  - Coco
  - Chobo
  - What is the trie?



- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$ terminal character
  - Coco$
  - Chobo$
  - What is the trie?

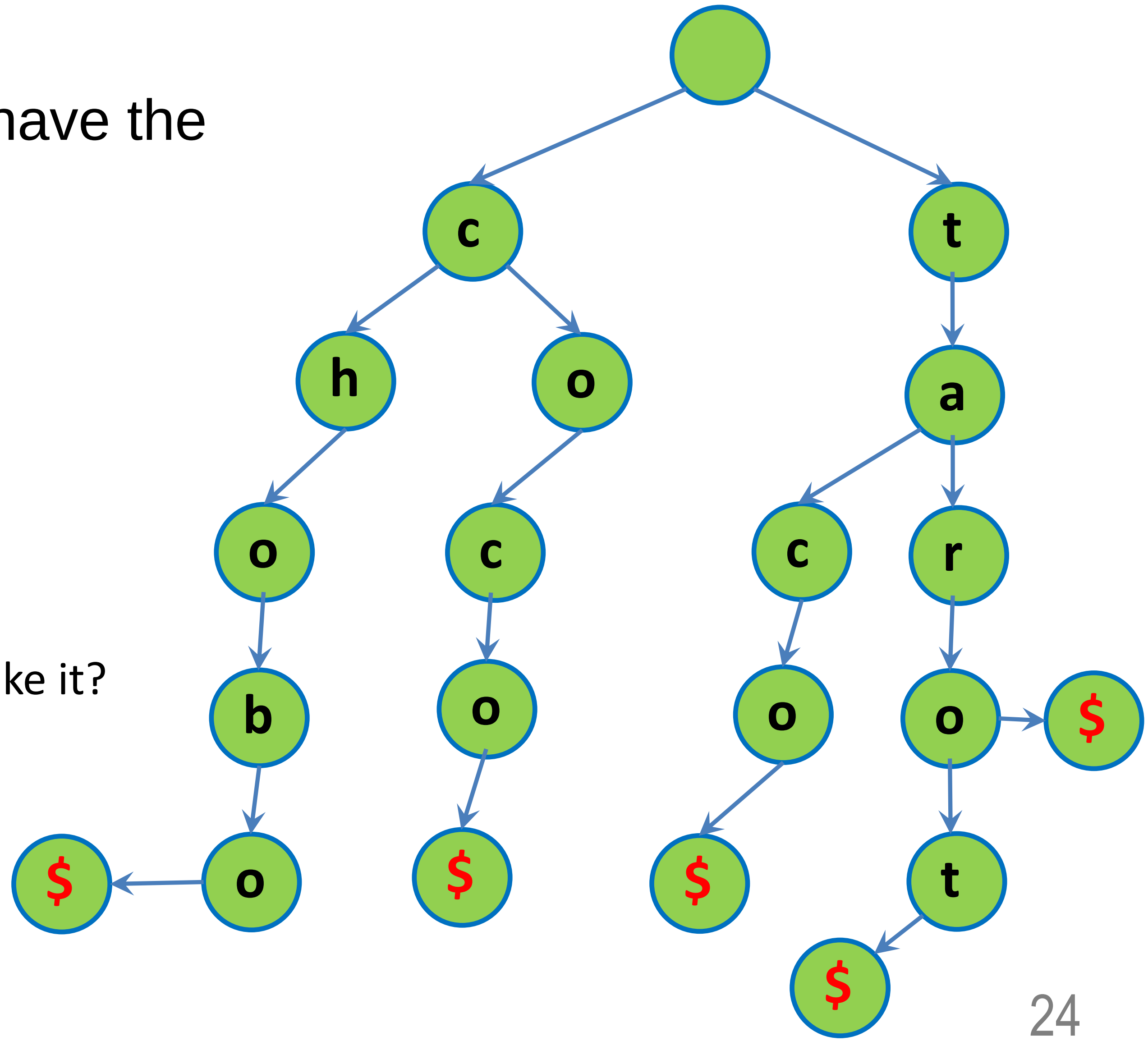


Tries

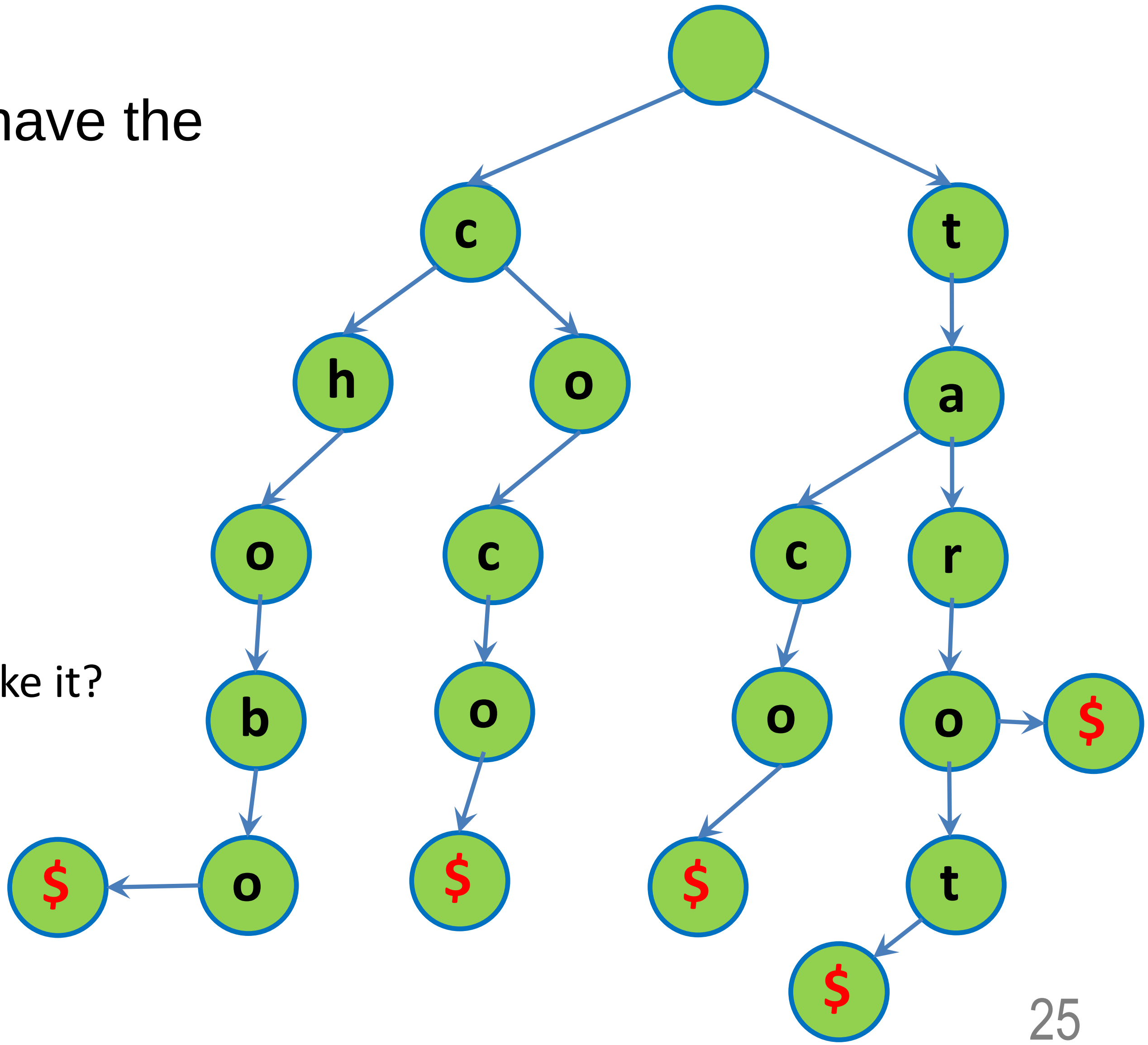
Efficient string retrieval

MONASH University

- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?



- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…

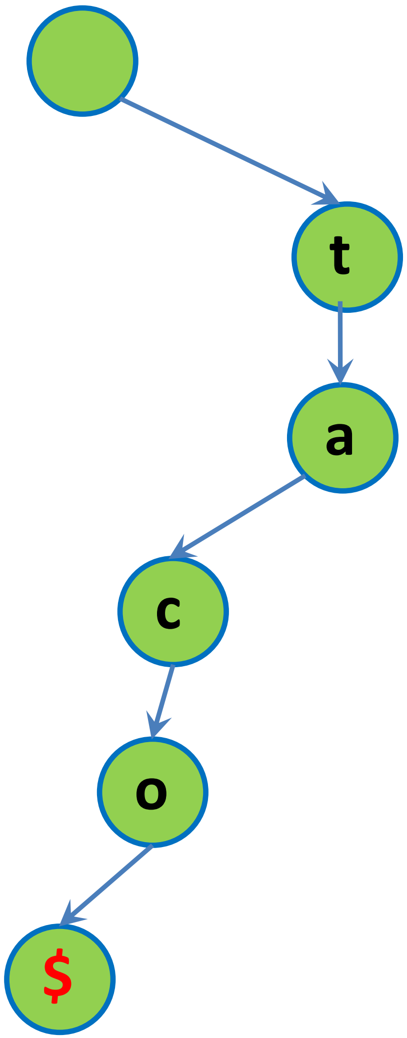


Tries

Efficient string retrieval

- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…

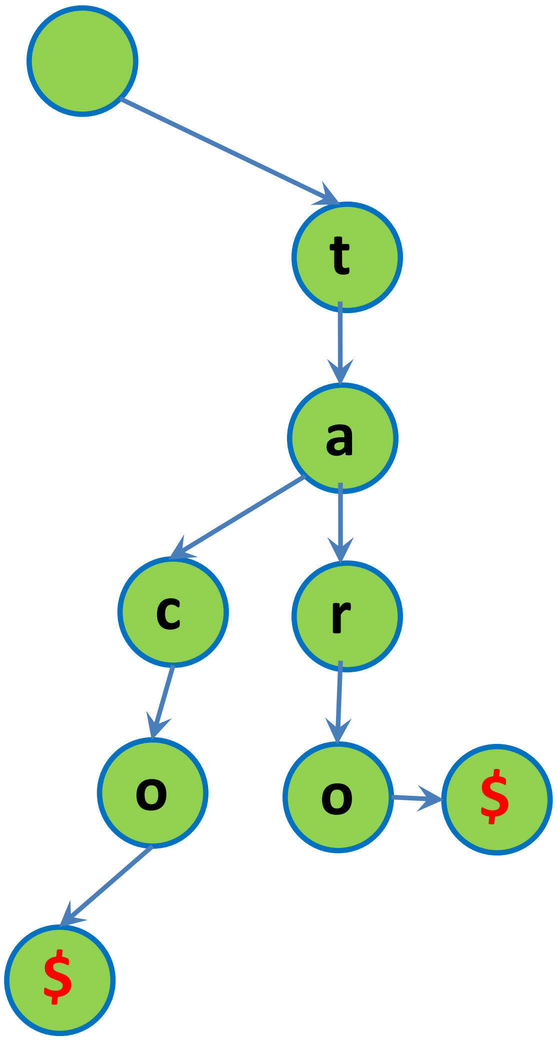
a..z
A..Z
52 characters
+ $
53 characters



Tries

Efficient string retrieval

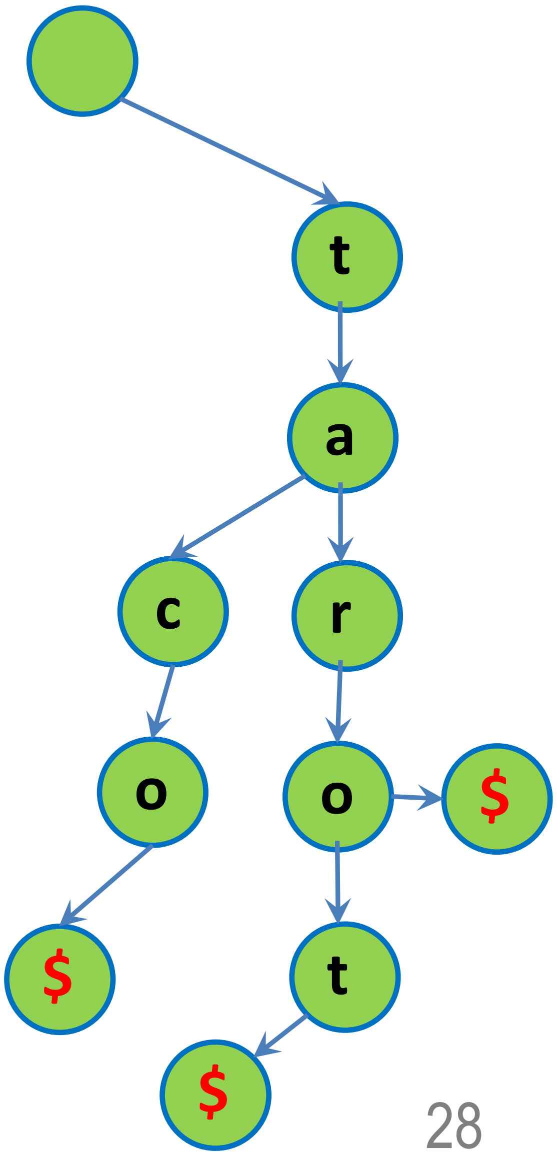
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…



Tries

Efficient string retrieval

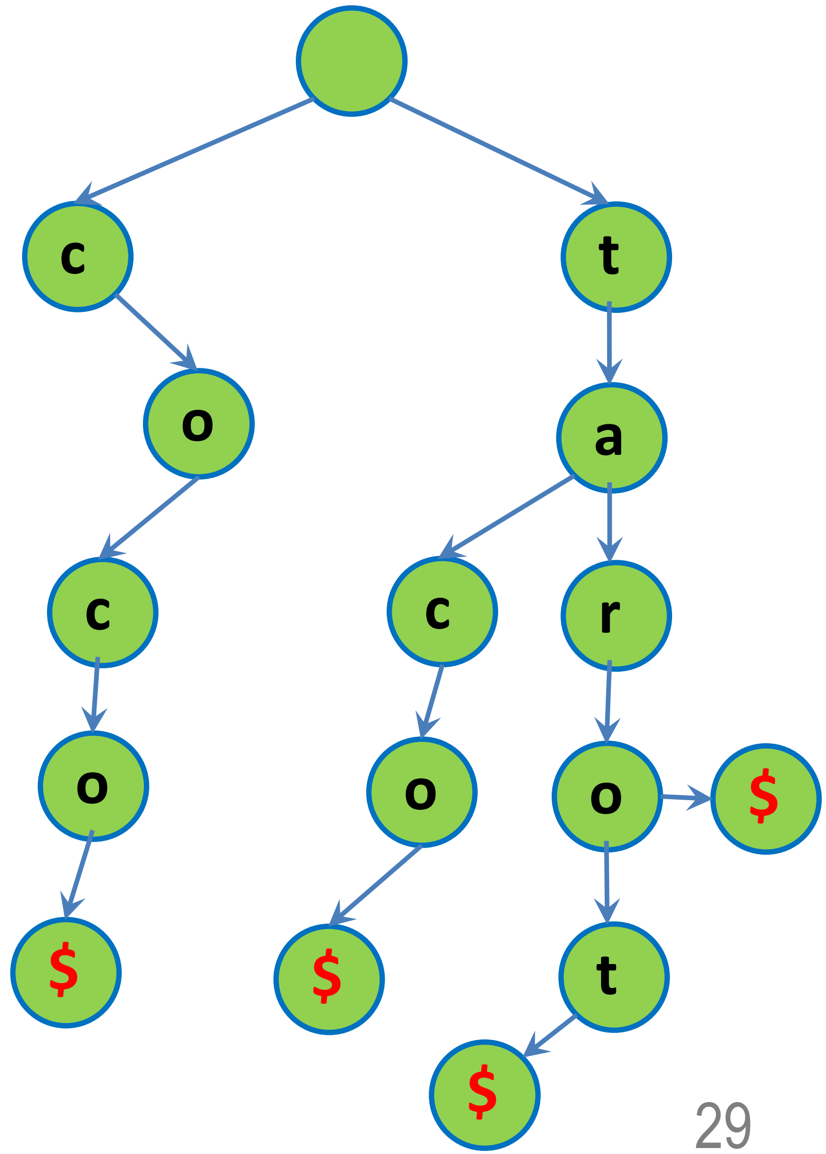
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…



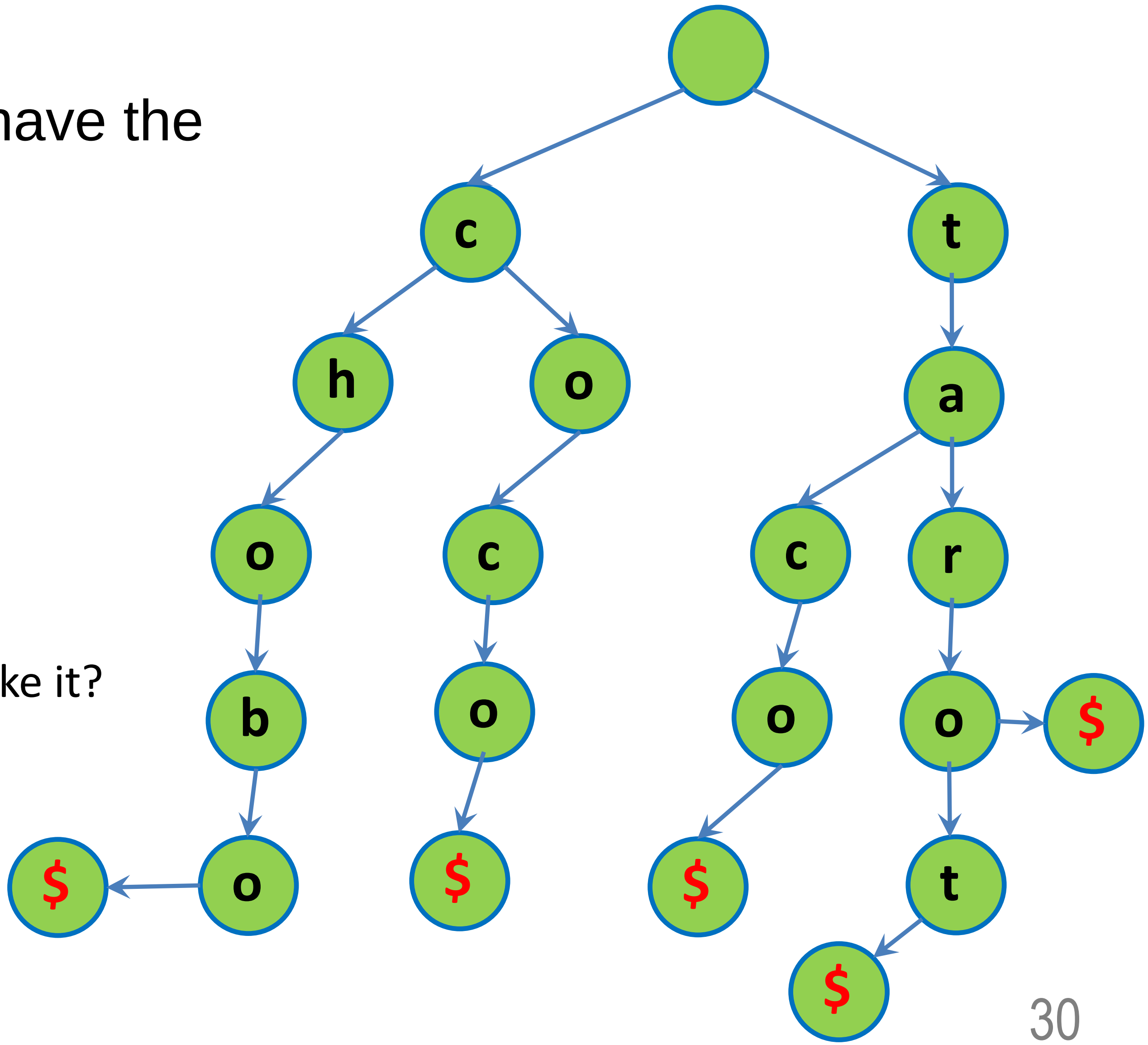
Tries

Efficient string retrieval

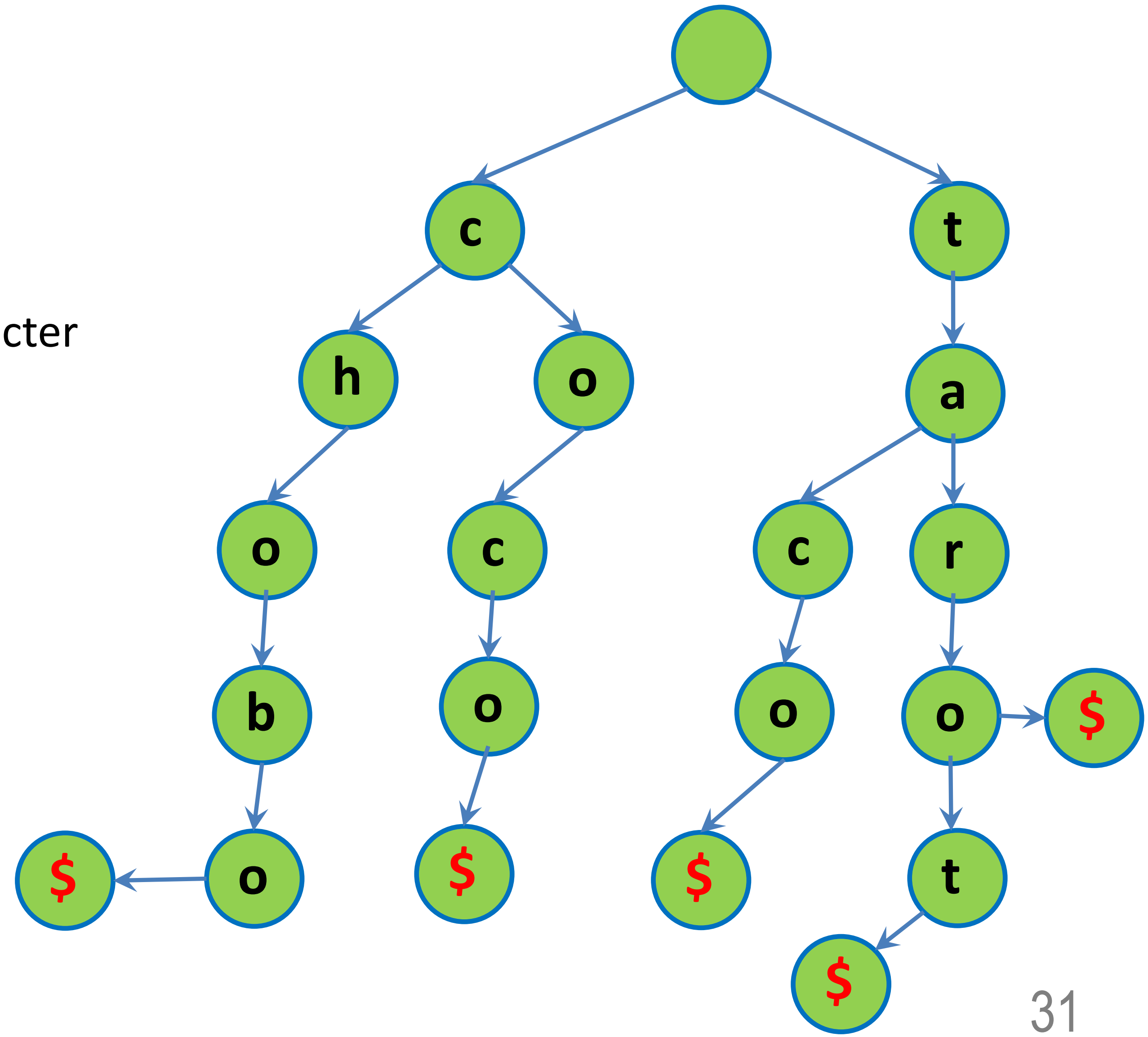
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…



- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…

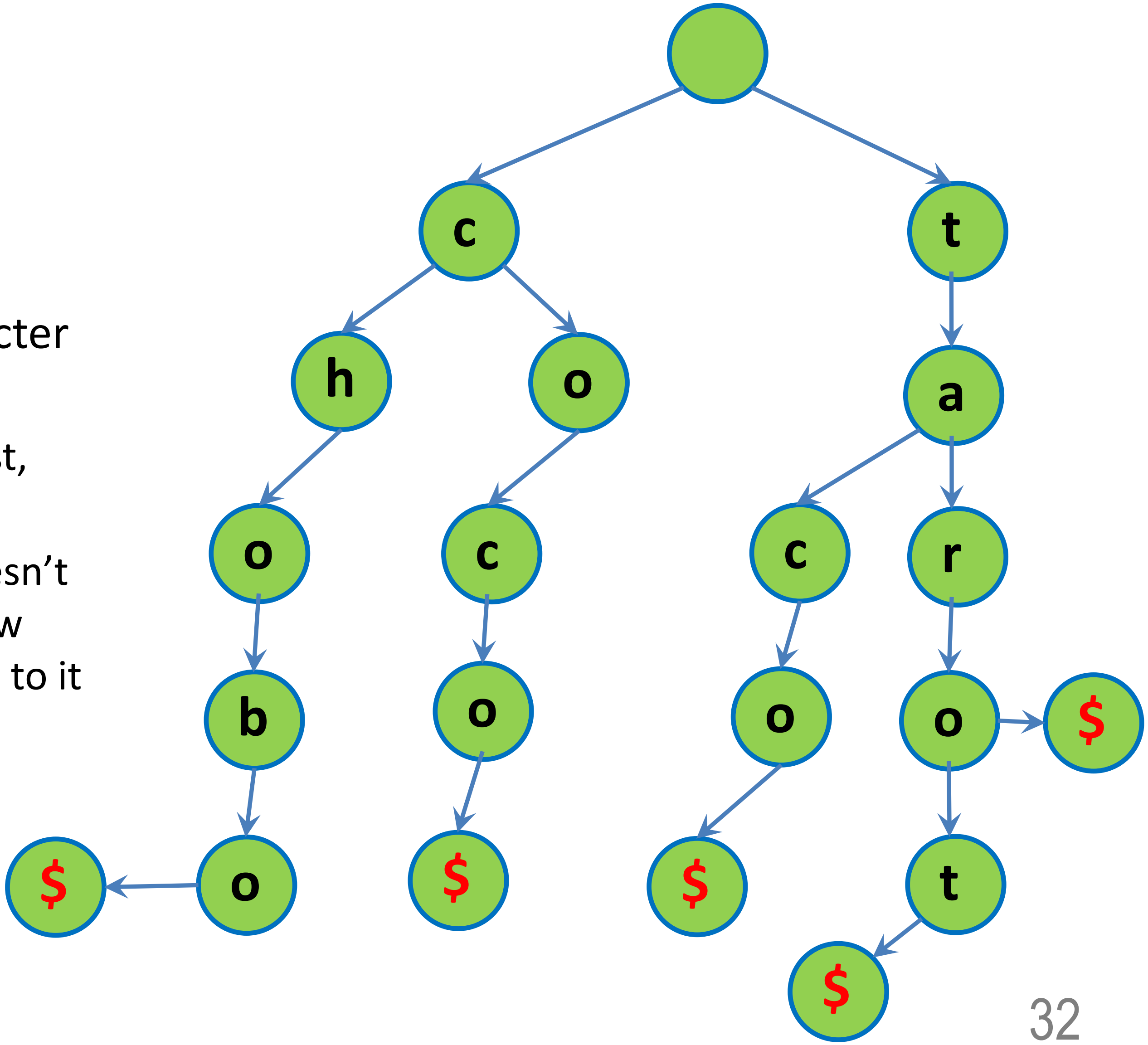


- So steps?
  - For each word
  - Start from root
  - Go through character by character



Tries
Efficient string retrieval

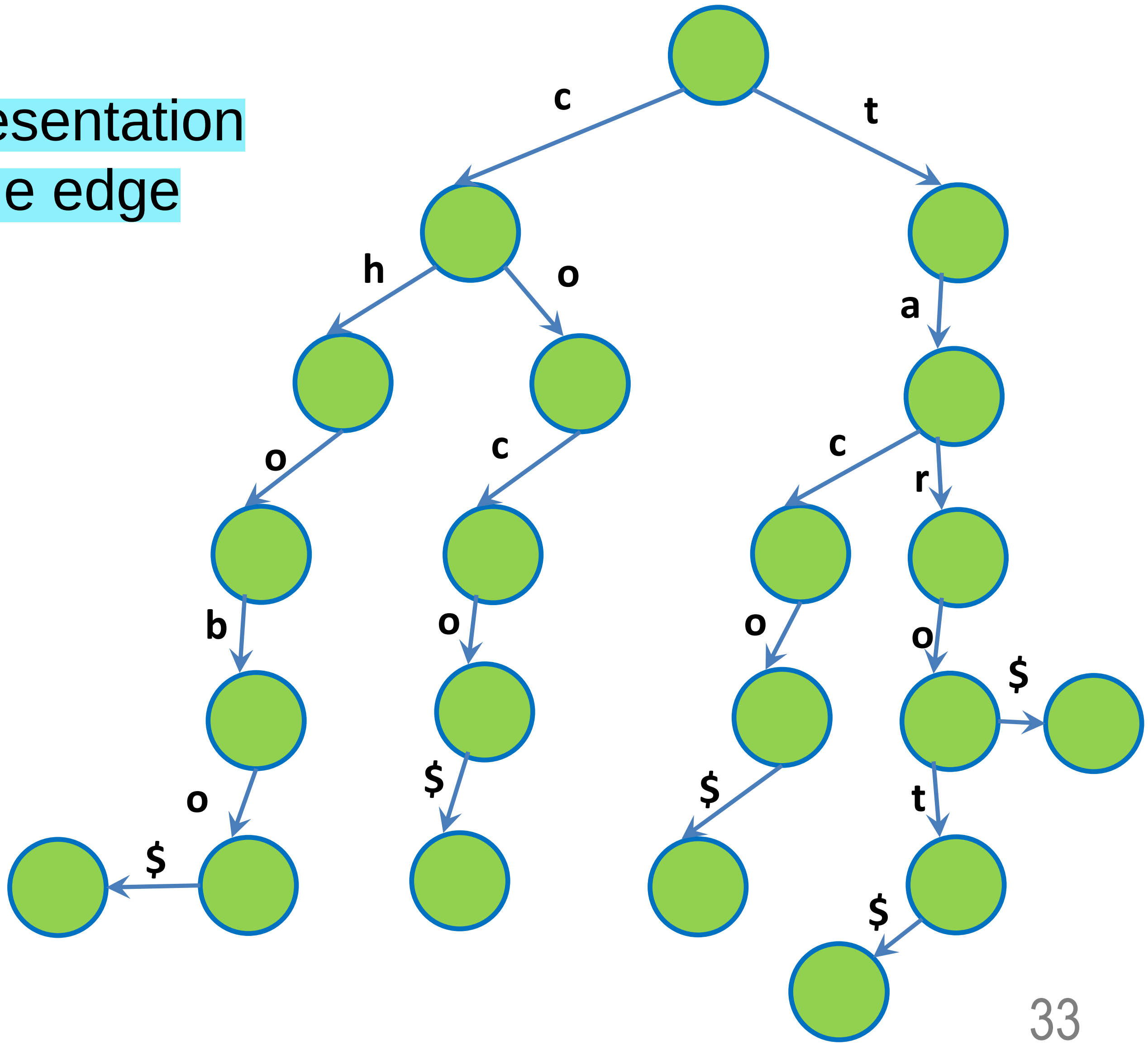
- So steps?
  - For each word
  - Start from root
  - Go through character by character
    - If character exist, follow through
    - If character doesn't exist, create new node and move to it



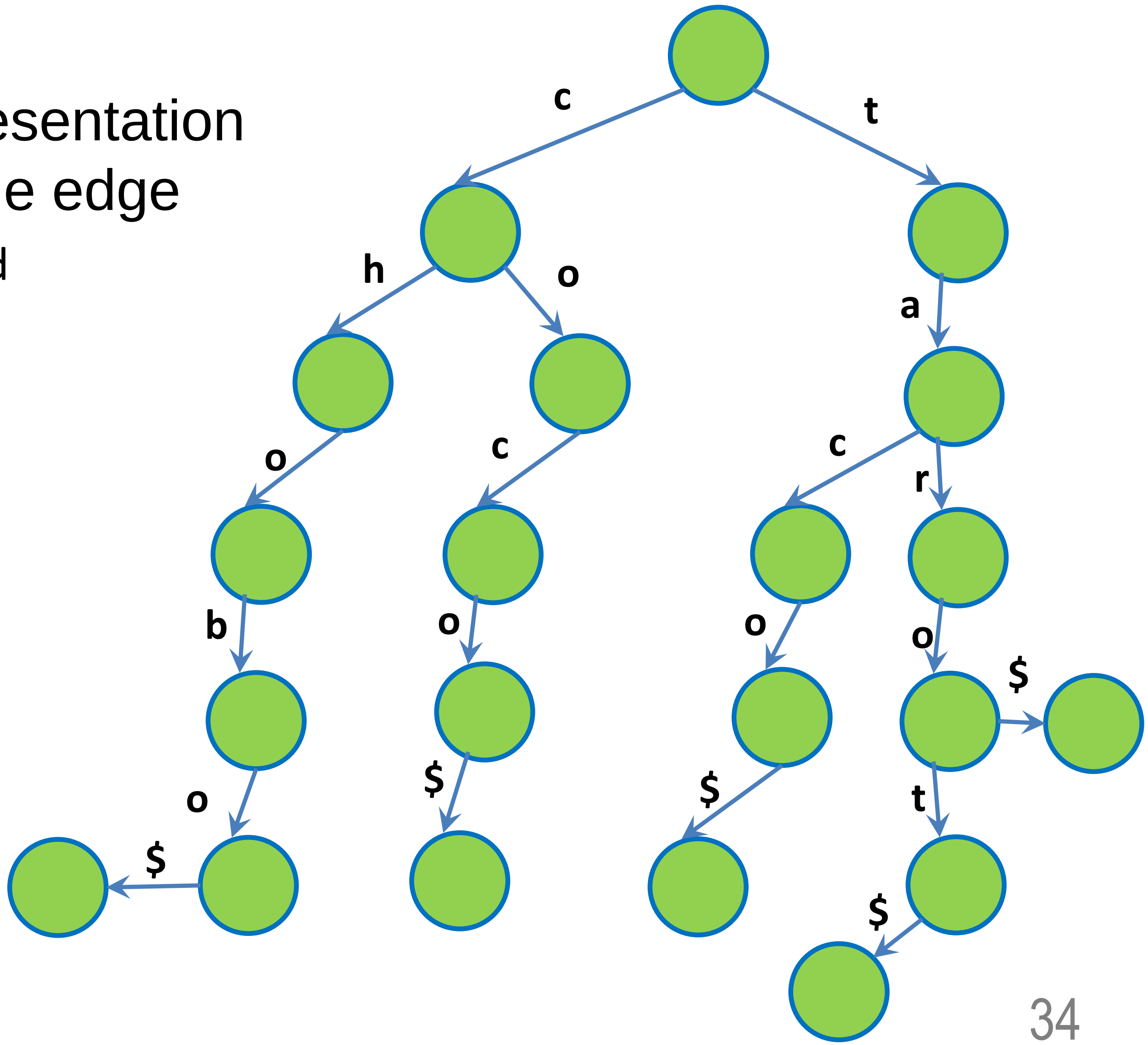
Tries

Efficient string retrieval

- The proper representation is character at the edge



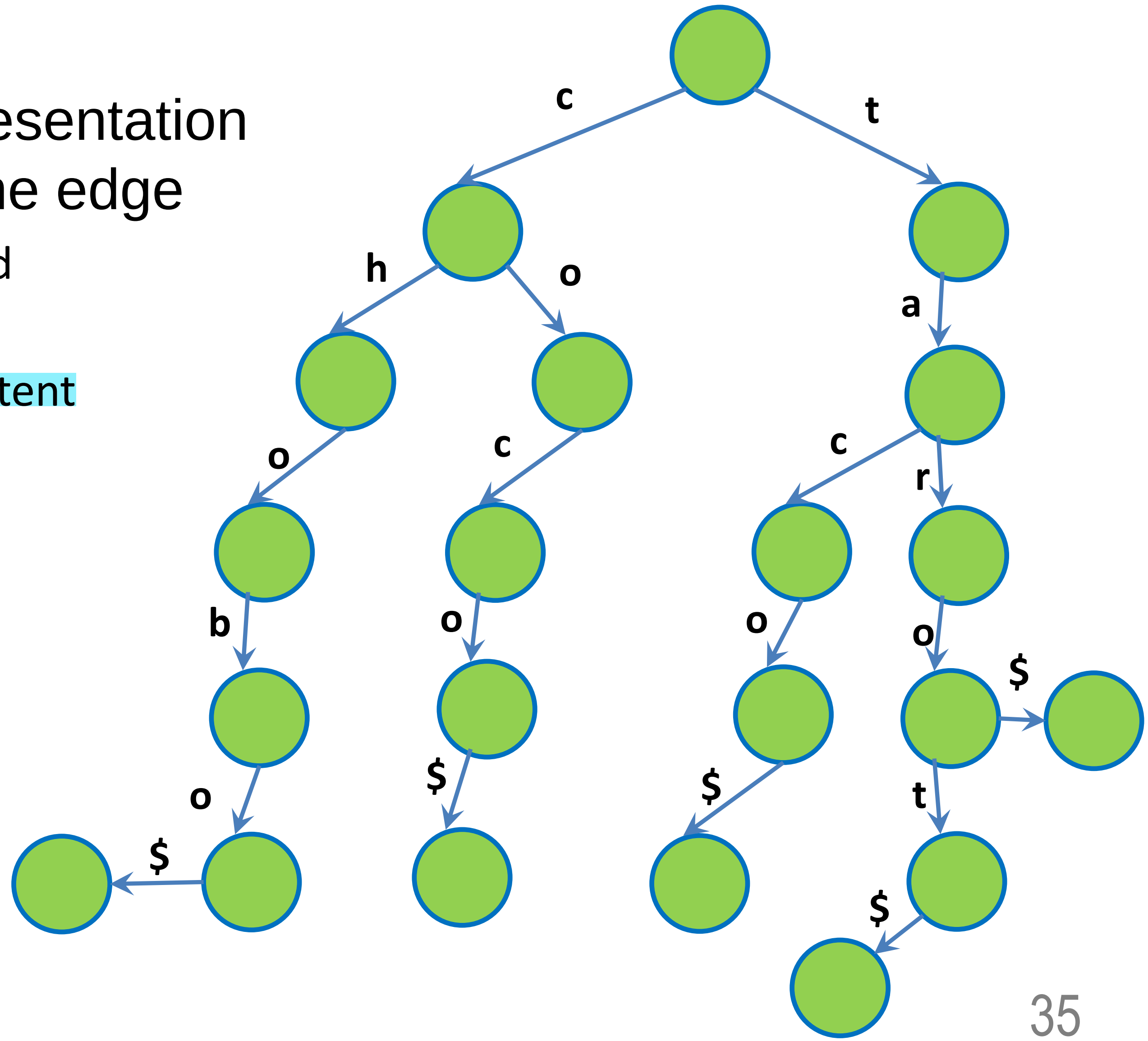
- The proper representation is character at the edge
  - Both are accepted for your exam!



Tries
Efficient string retrieval

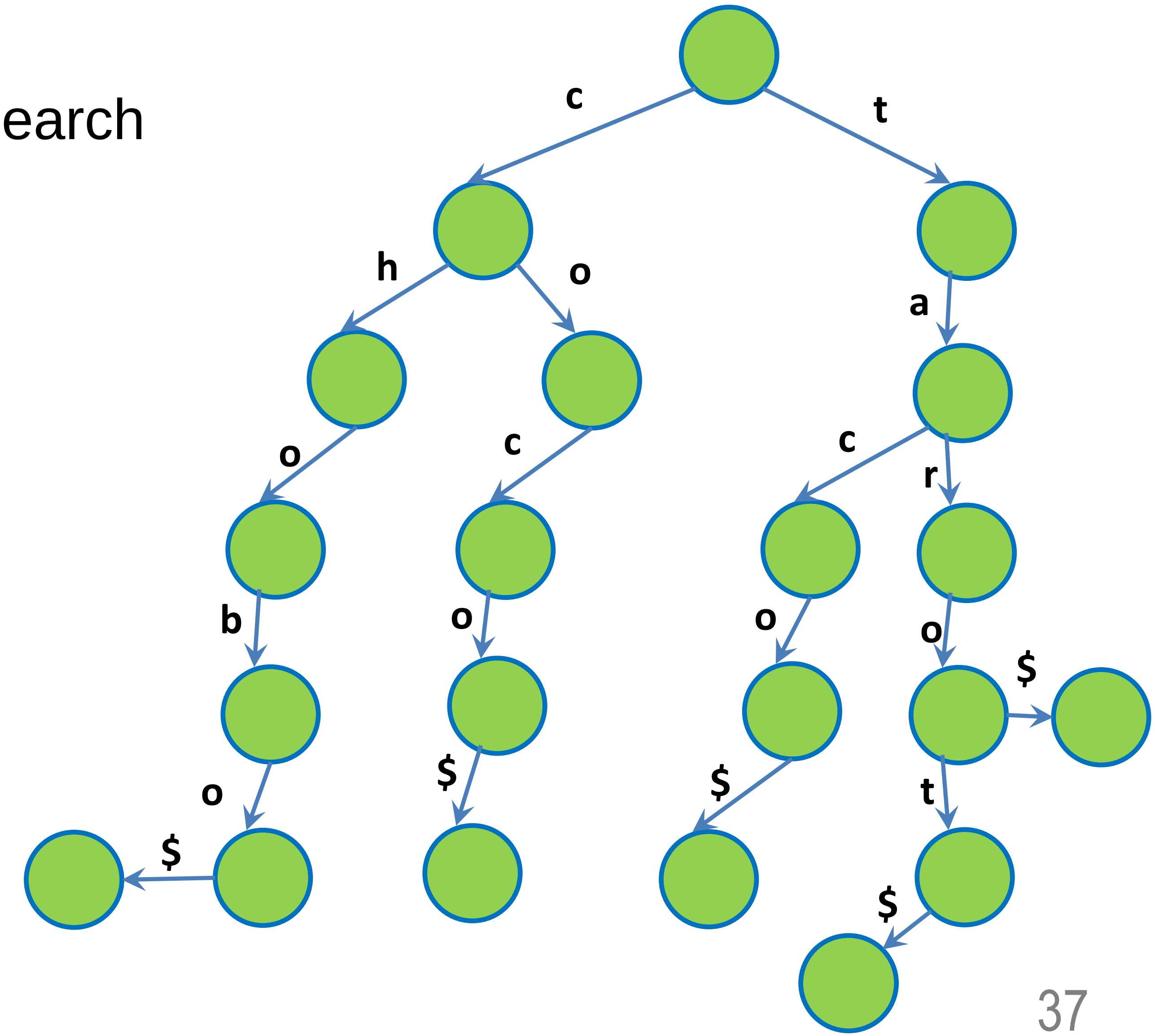
MONASH University

- The proper representation is character at the edge
  – Both are accepted for your exam!
  – This is also consistent with the graph representation



Questions?

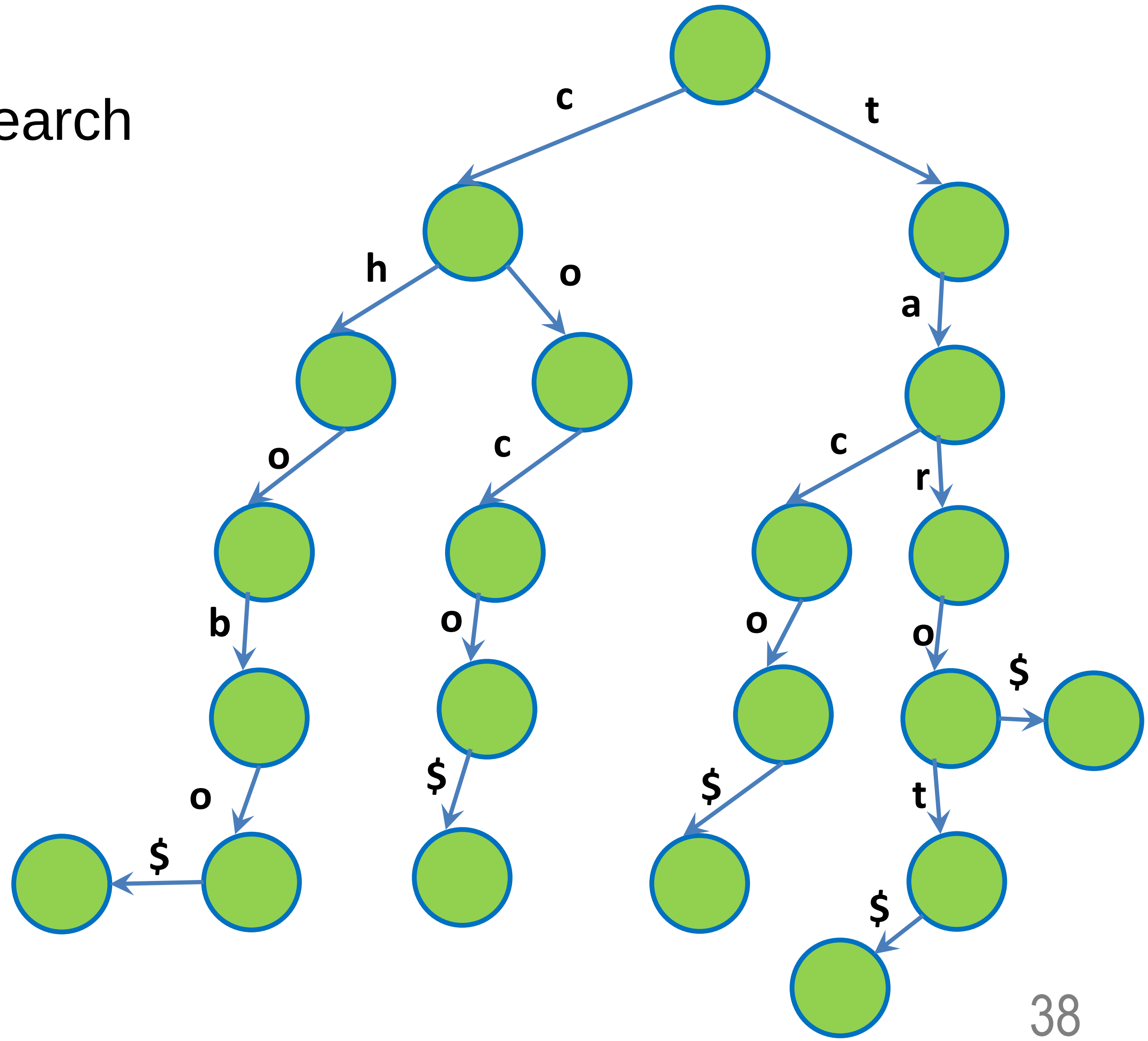
- So how do we search for retrieval?



Tries

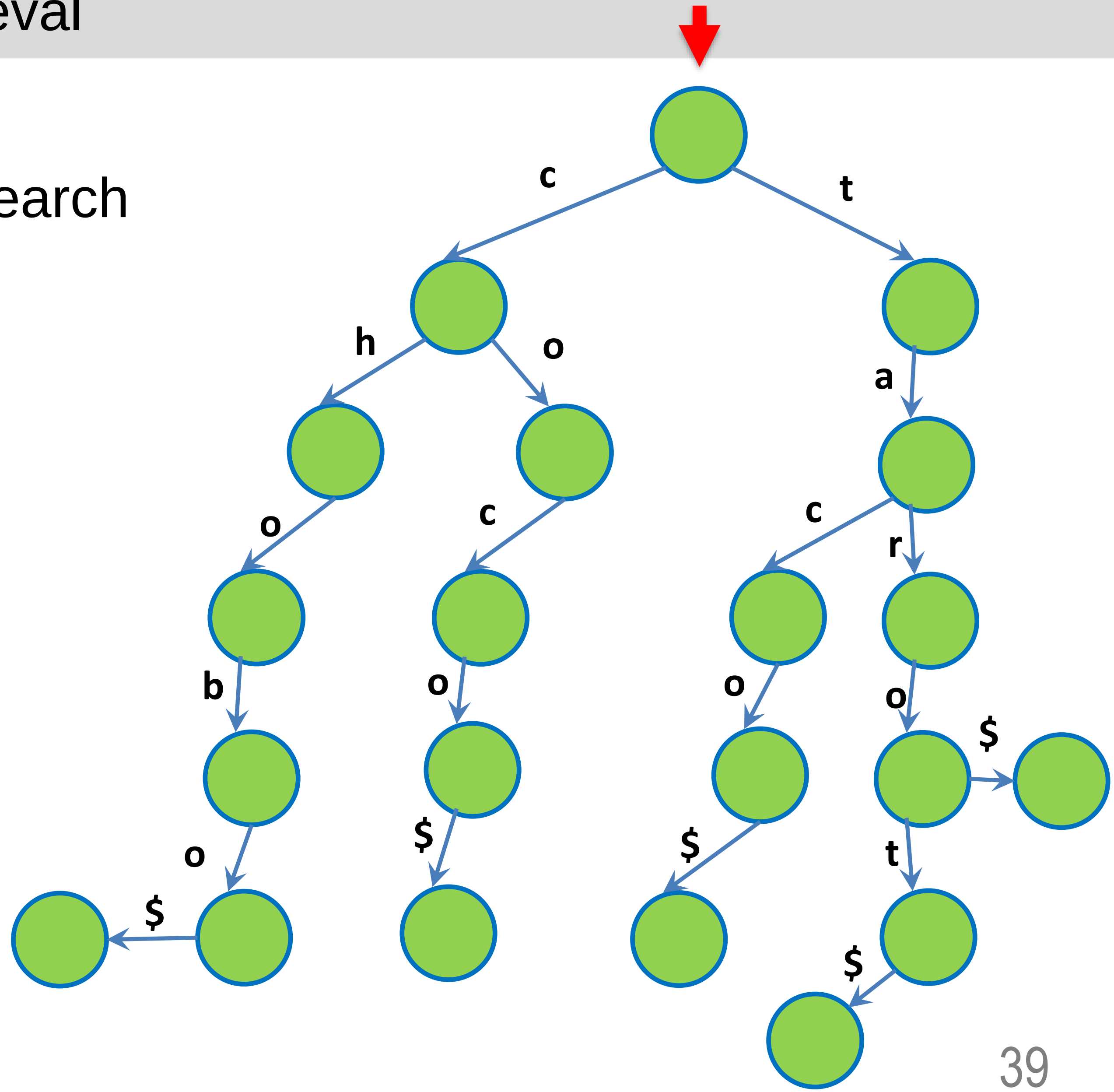
Efficient string retrieval

- So how do we search for retrieval?
  - Search for “coco”



Tries
Efficient string retrieval

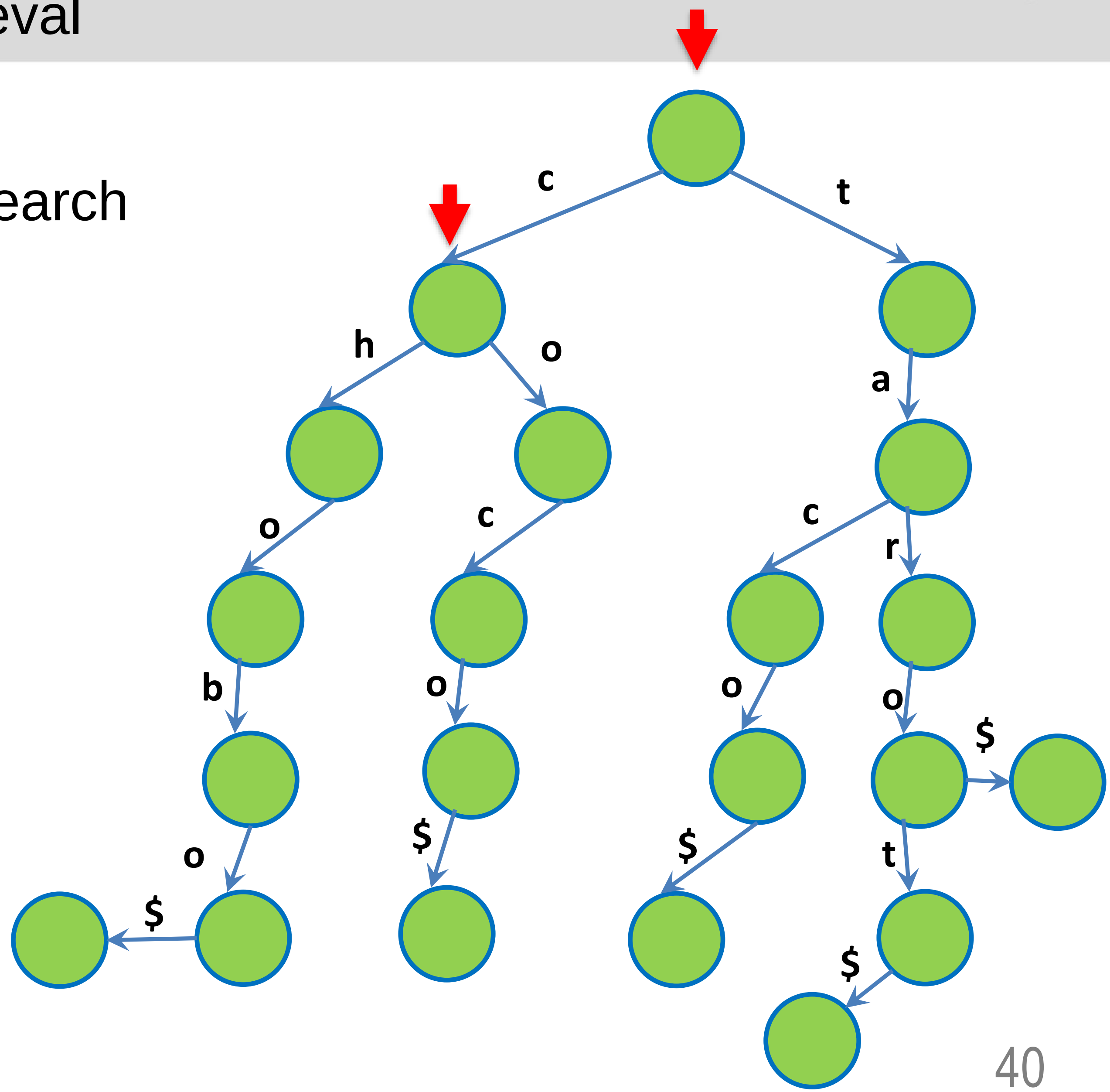
- So how do we search for retrieval?
  - Search for “coco”



Tries

Efficient string retrieval

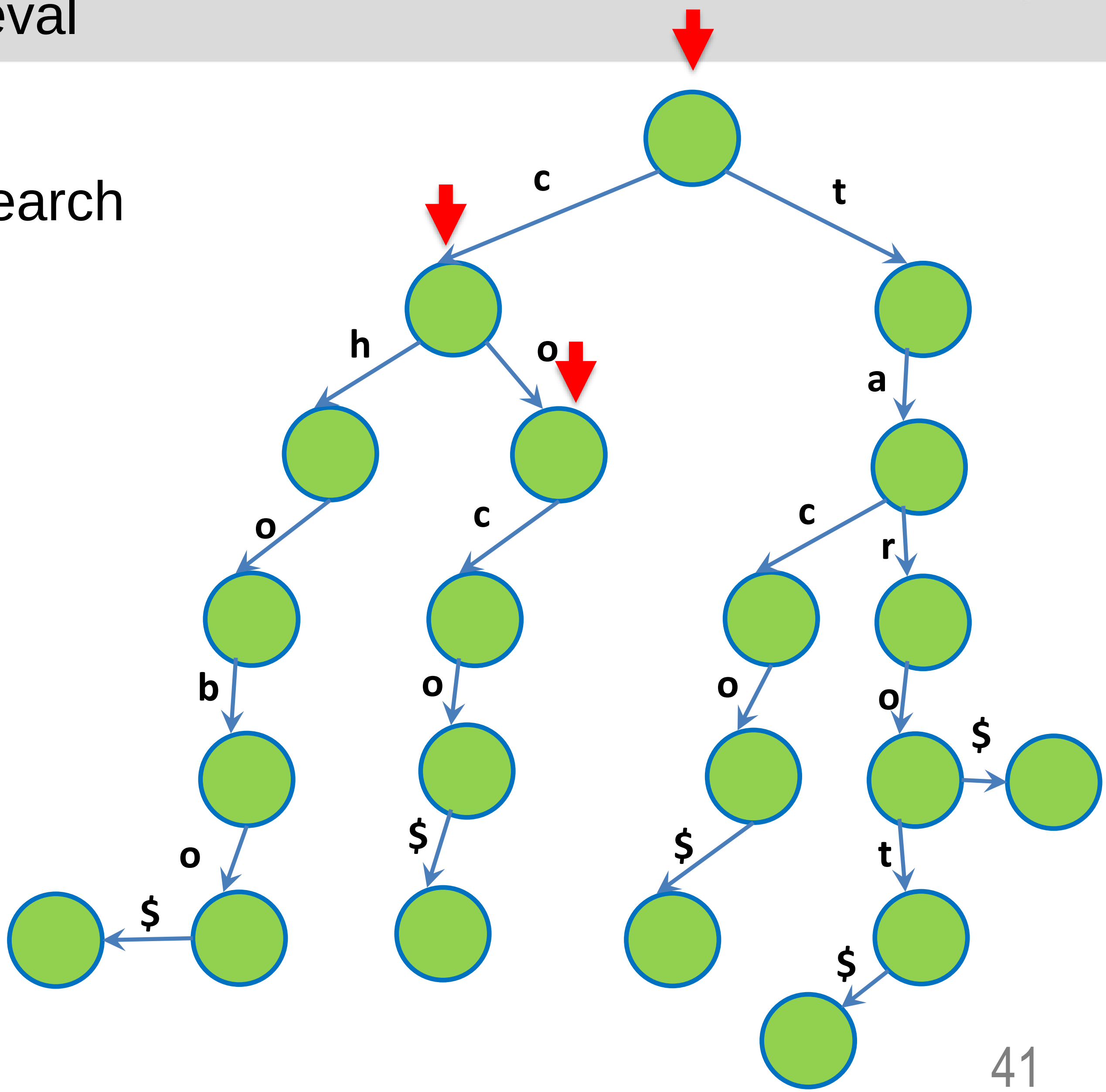
- So how do we search for retrieval?
  - Search for “coco”



Tries

Efficient string retrieval

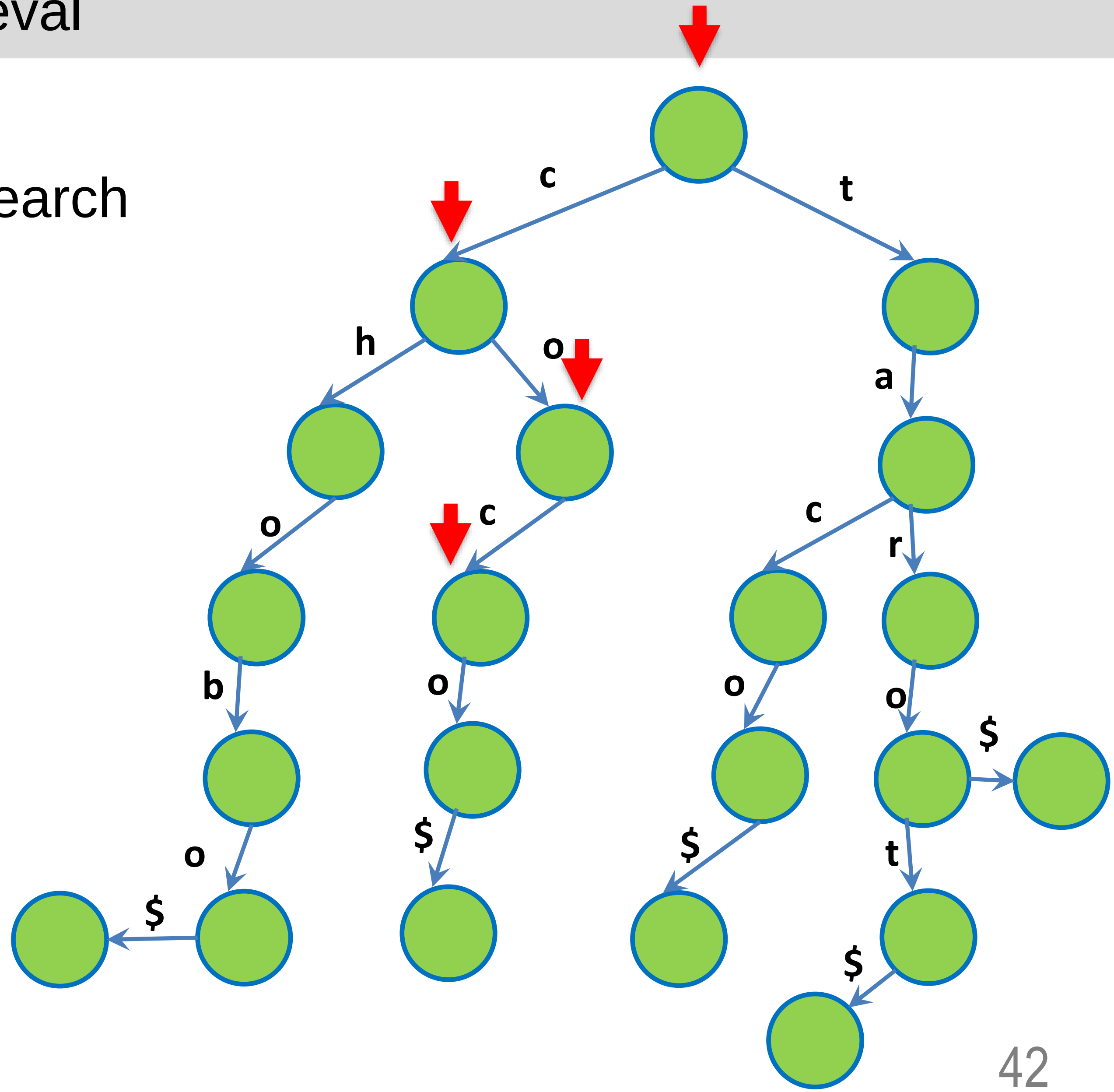
- So how do we search for retrieval?
  - Search for "coco"



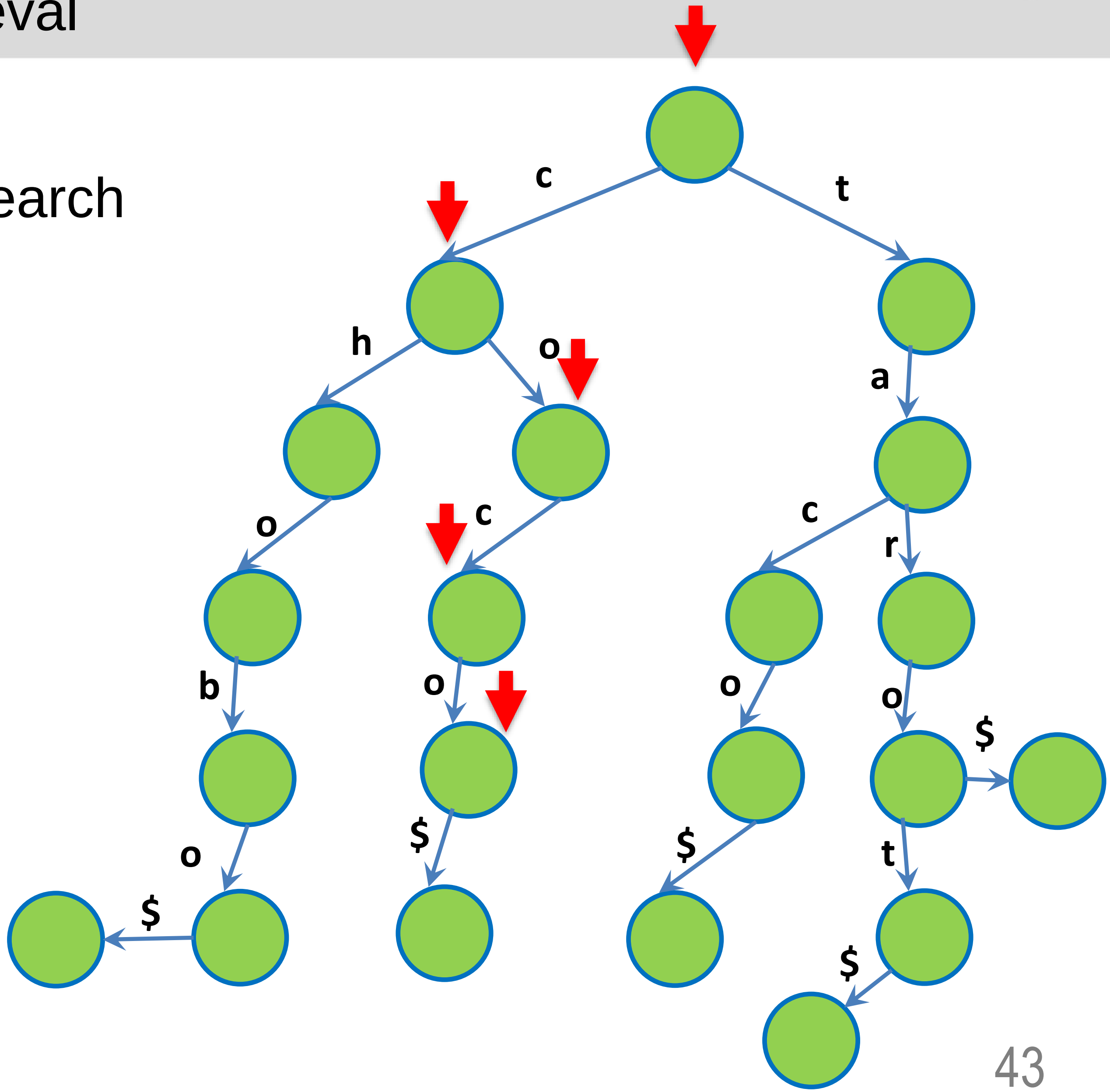
Tries

Efficient string retrieval

- So how do we search for retrieval?
  - Search for “coco”

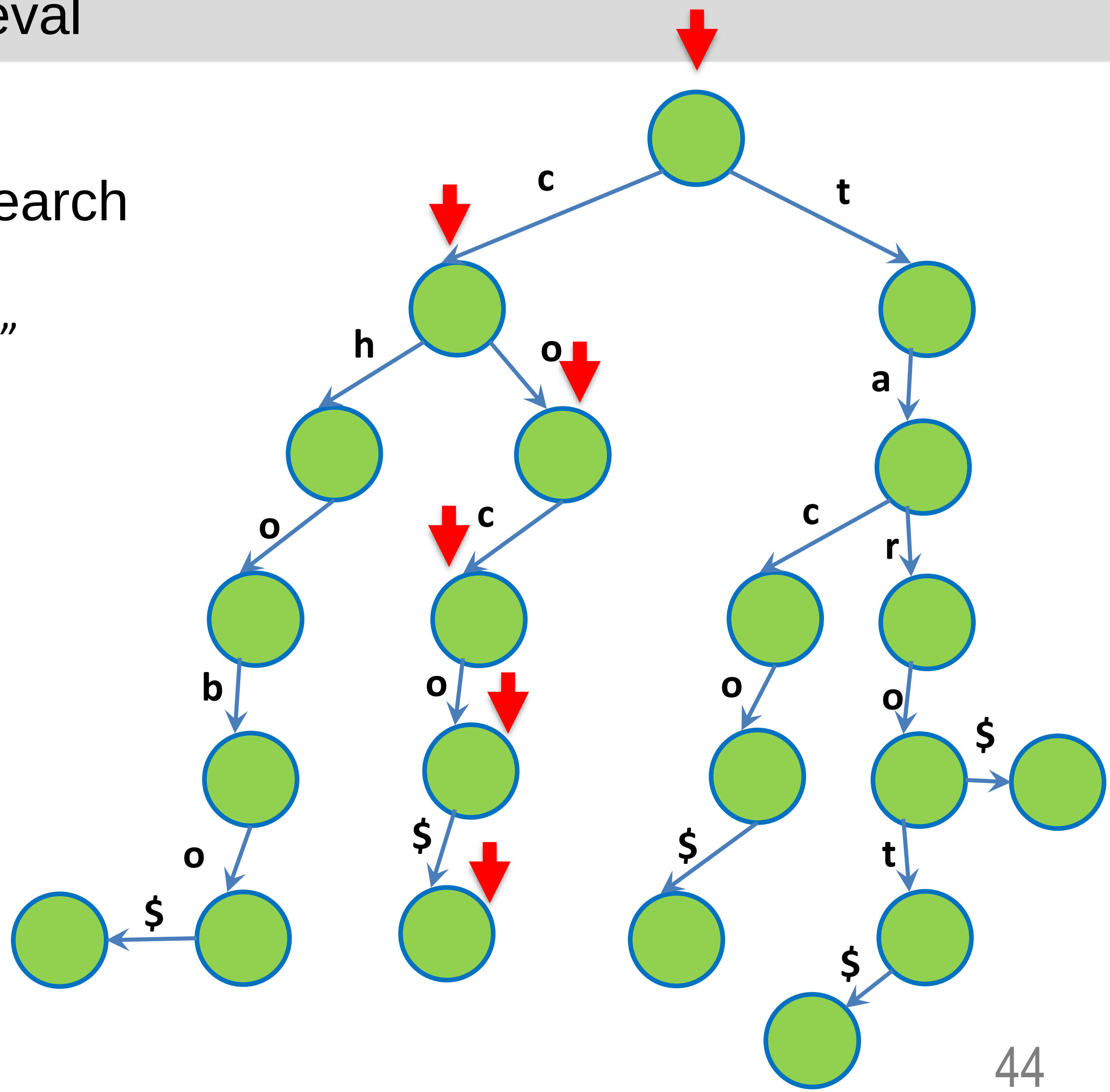


- So how do we search for retrieval?
  - Search for "coc<u>o</u>"

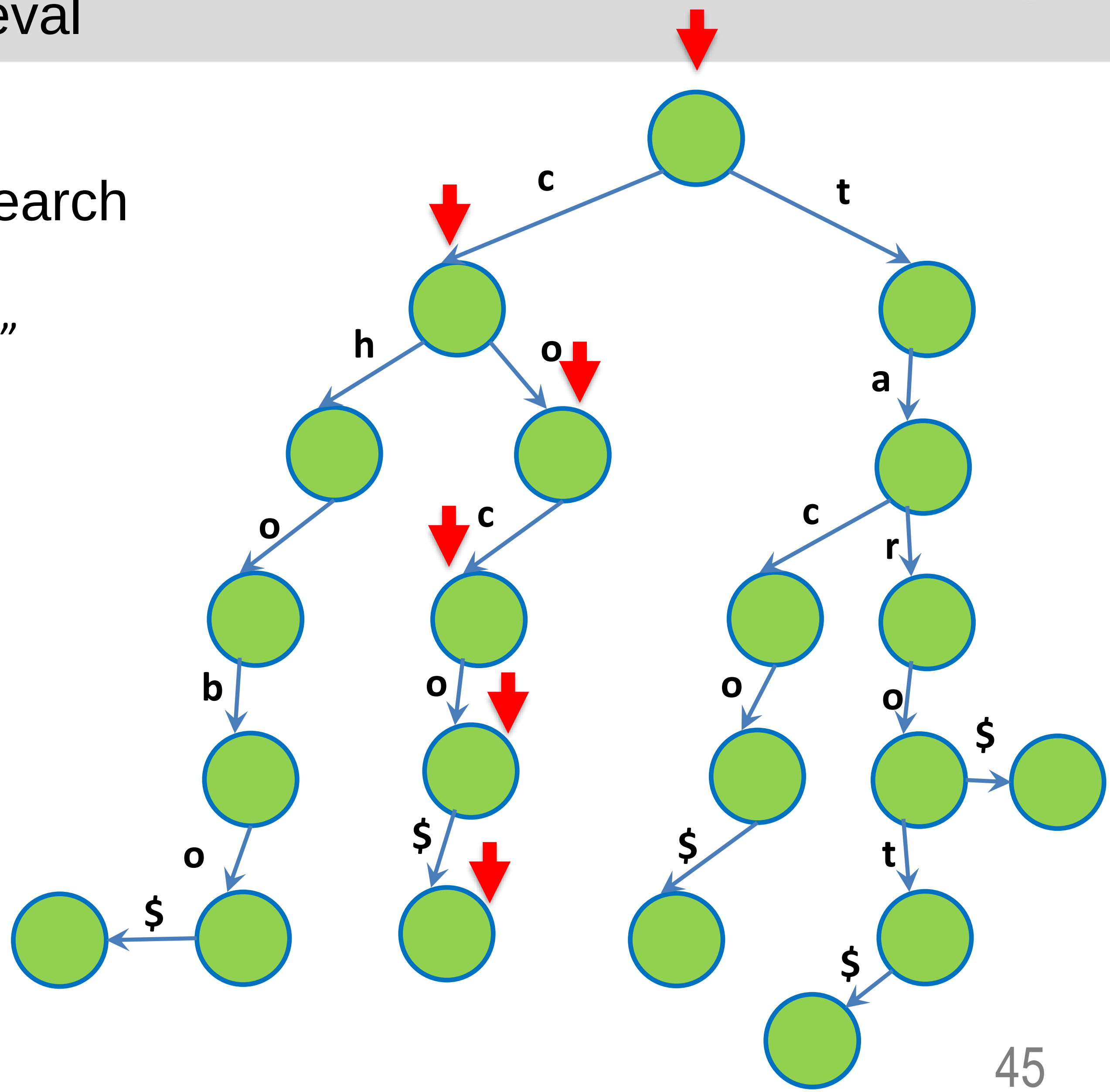


Tries
Efficient string retrieval

- So how do we search for retrieval?
  - Search for “coco$”



- So how do we search for retrieval?
  - Search for “coco$”
    so we found it!



MONASH University

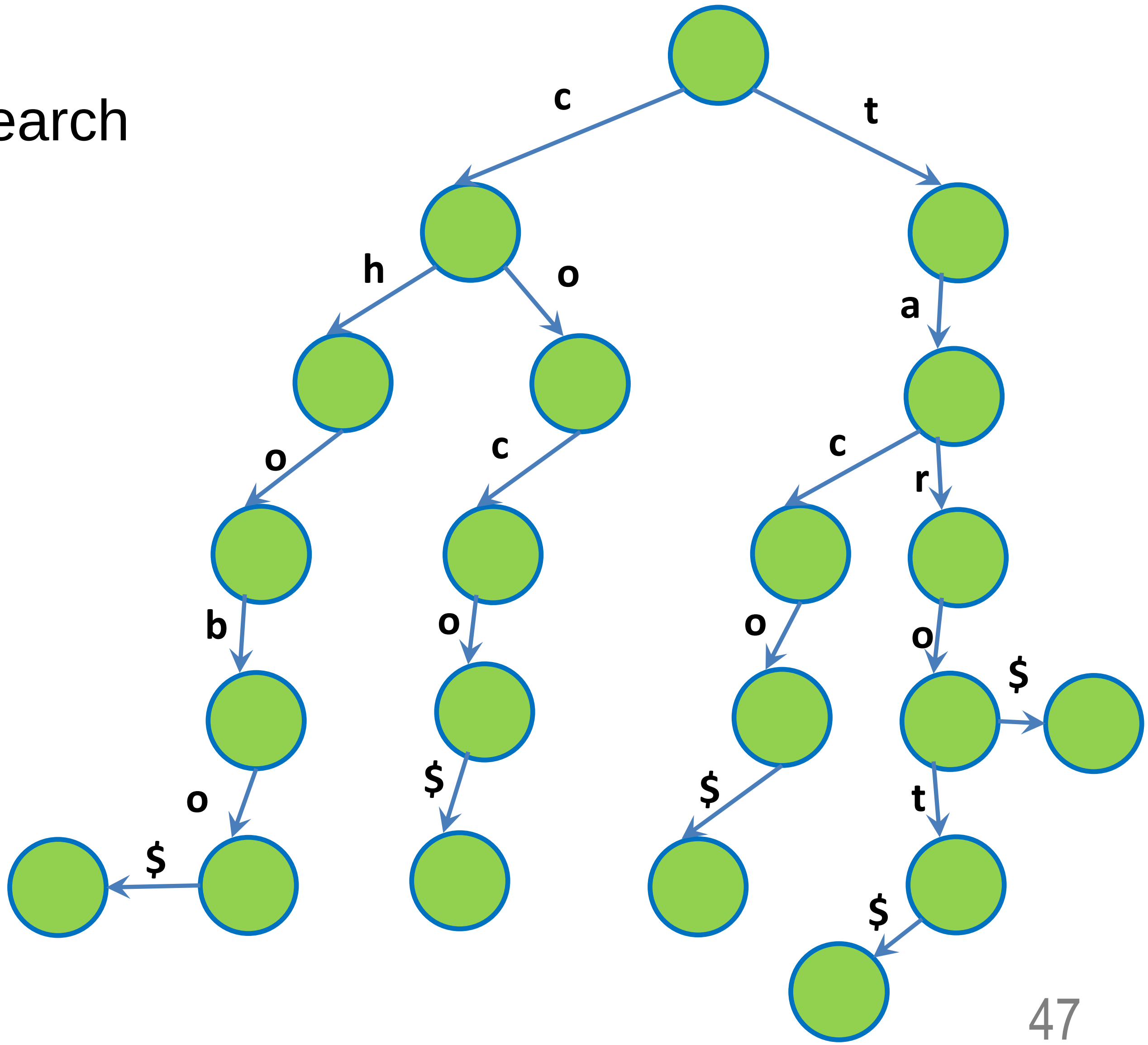
Questions?

Tries

Efficient string retrieval

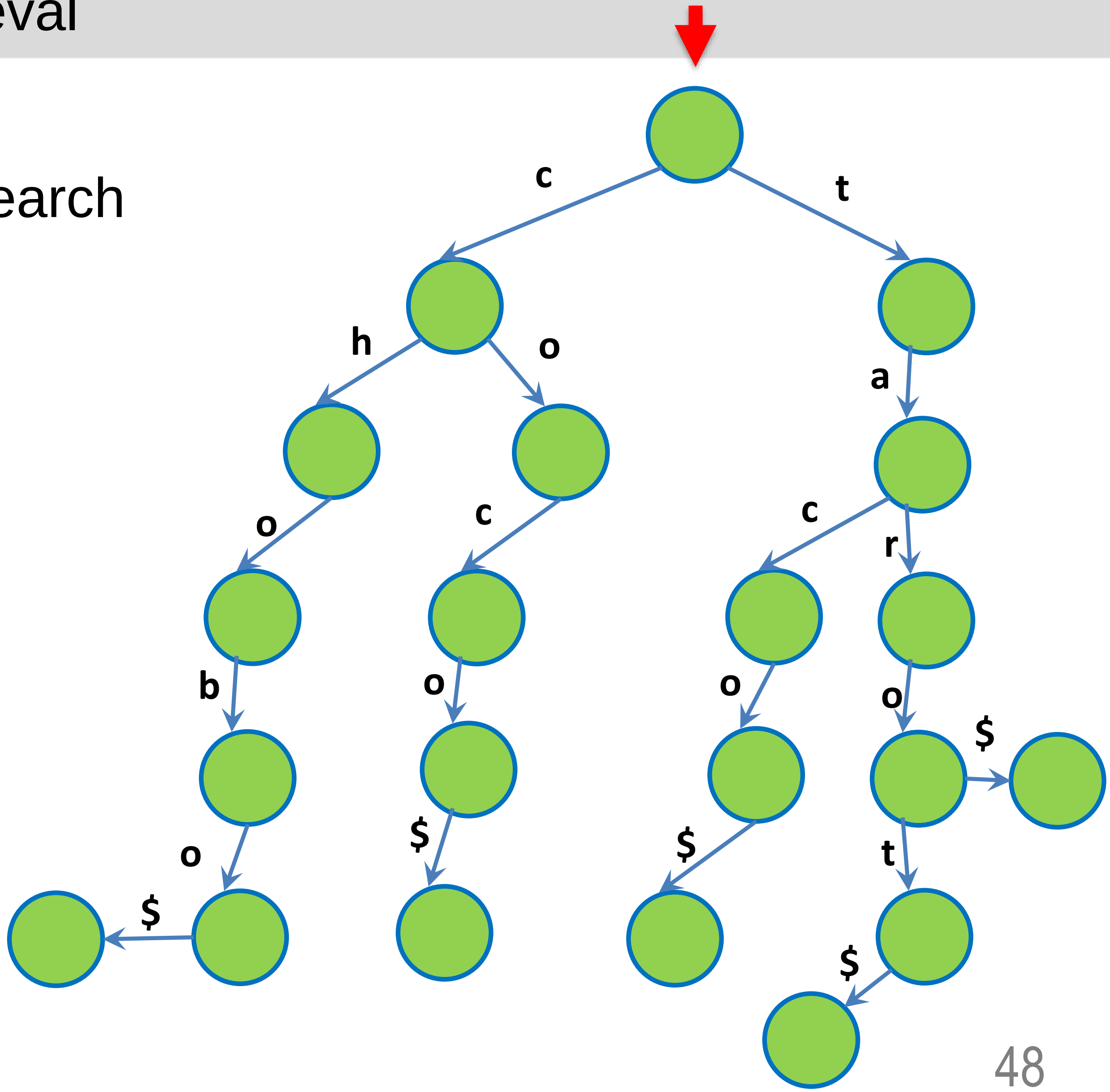
MONASH University

- So how do we search for retrieval?
  - Search for “tarot”



Tries
Efficient string retrieval

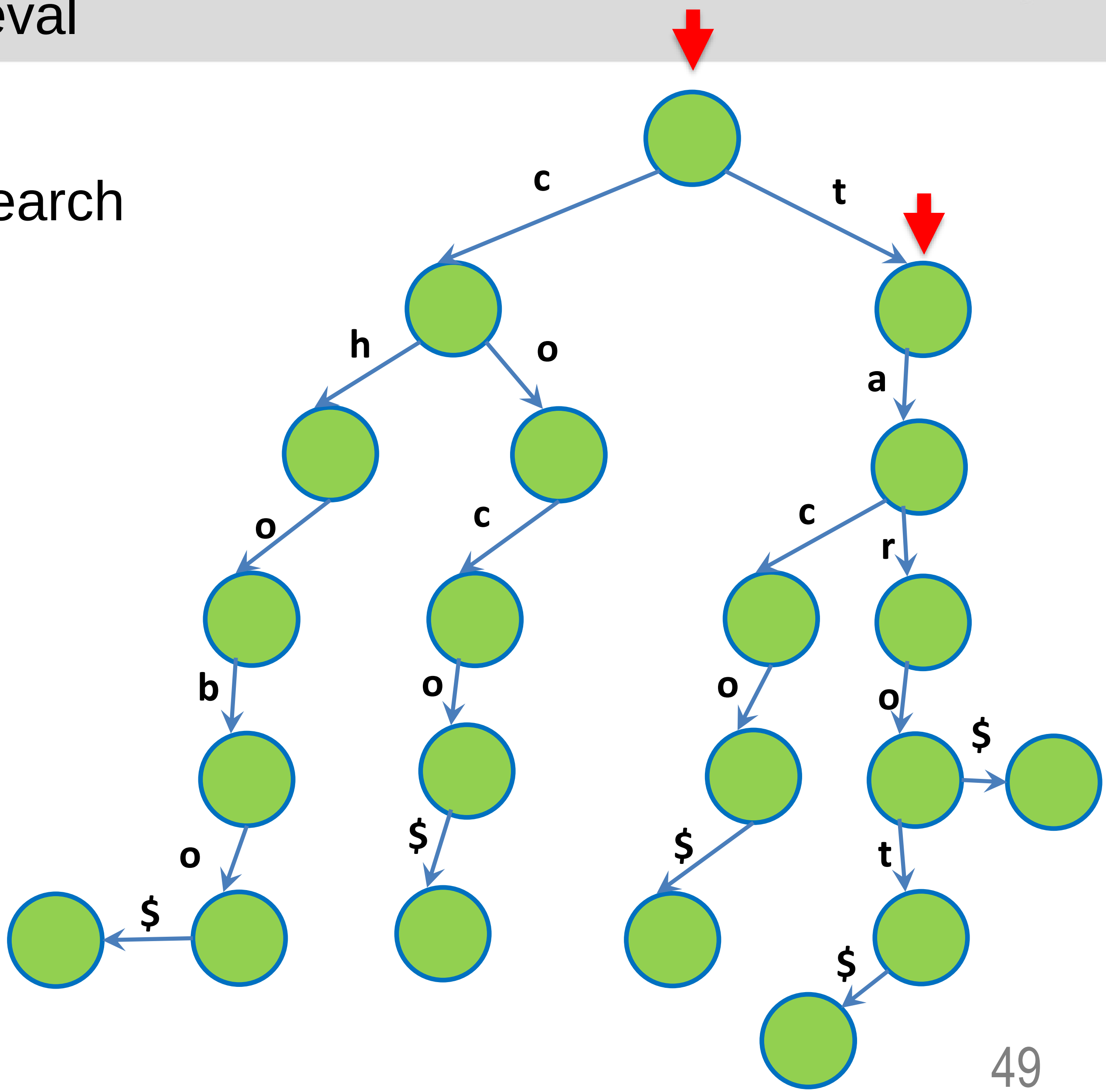
- So how do we search for retrieval?
  - Search for “tarot”



Tries

Efficient string retrieval

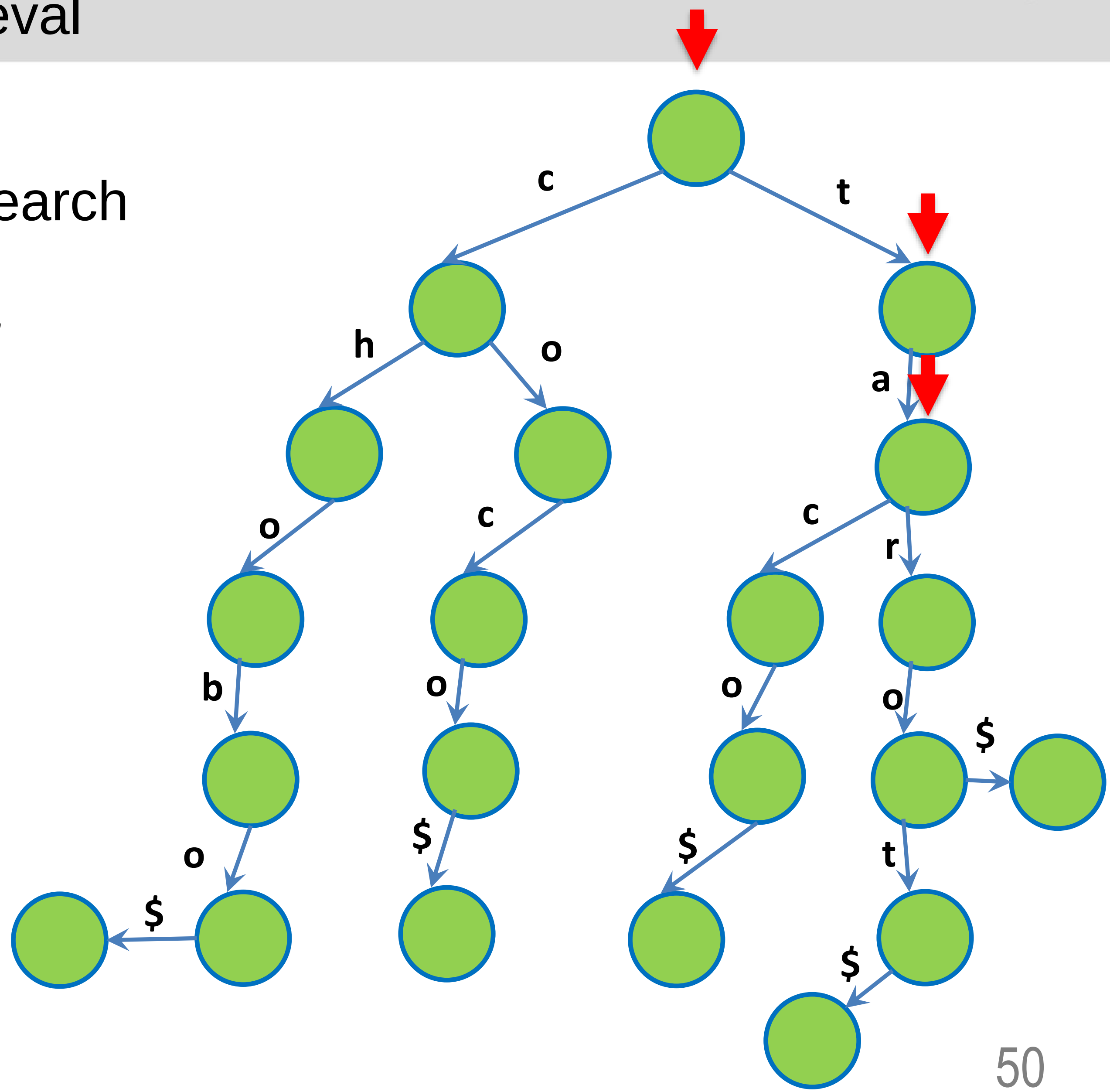
- So how do we search for retrieval?
  - Search for "t̲arot"



Tries

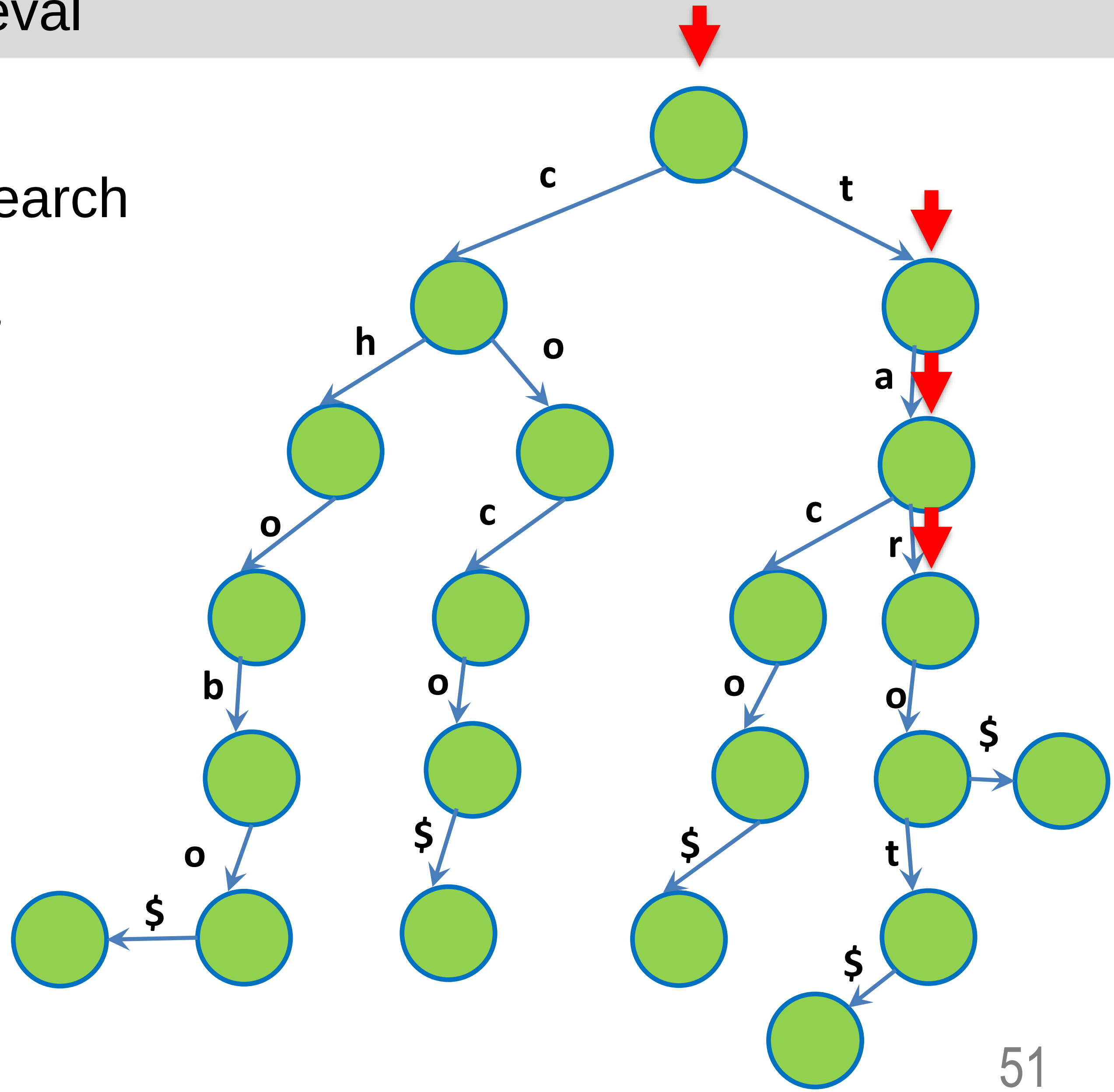
Efficient string retrieval

- So how do we search for retrieval?
  - Search for “tarot”

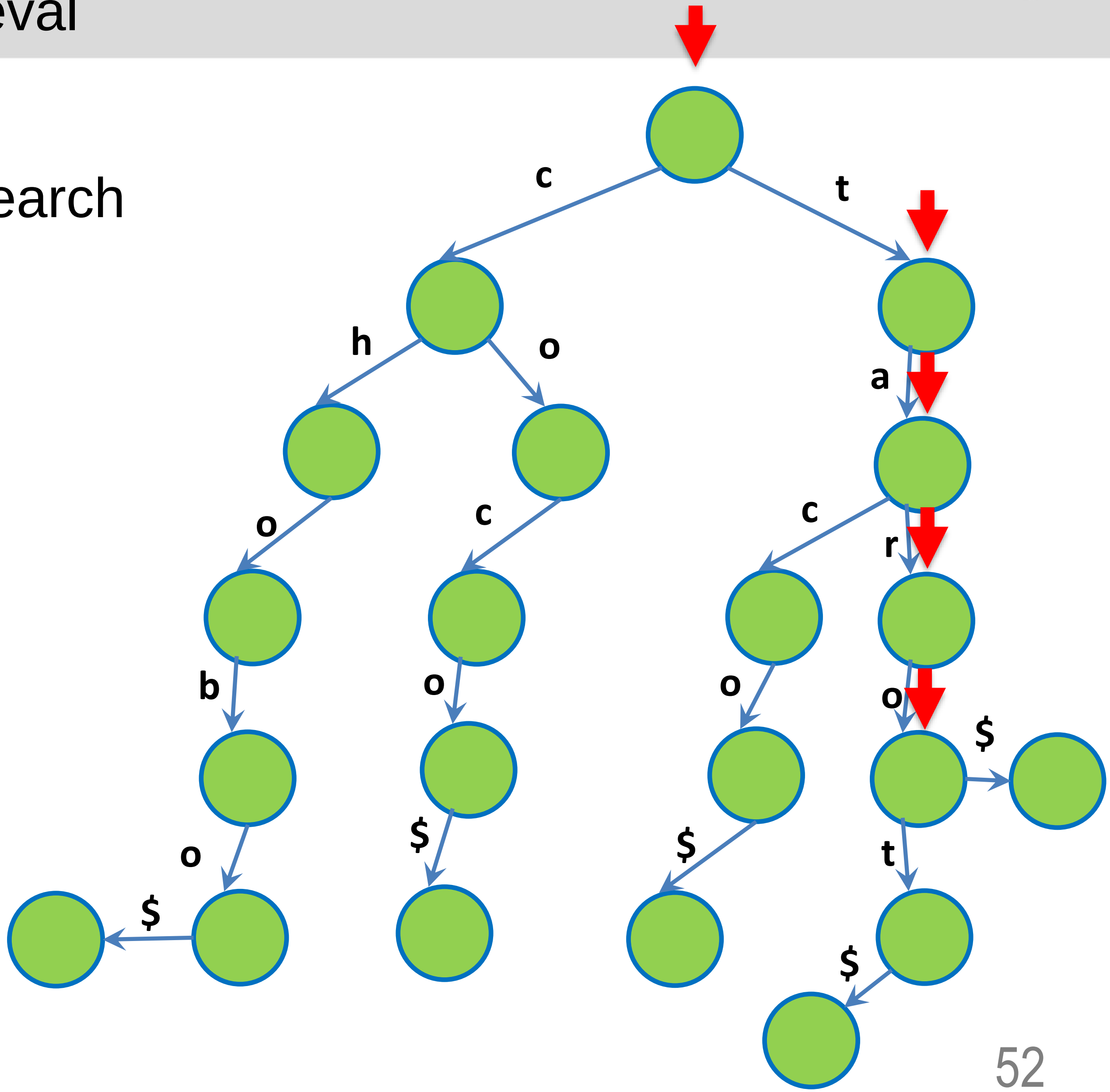


Tries
Efficient string retrieval

- So how do we search for retrieval?
  - Search for “tarot”



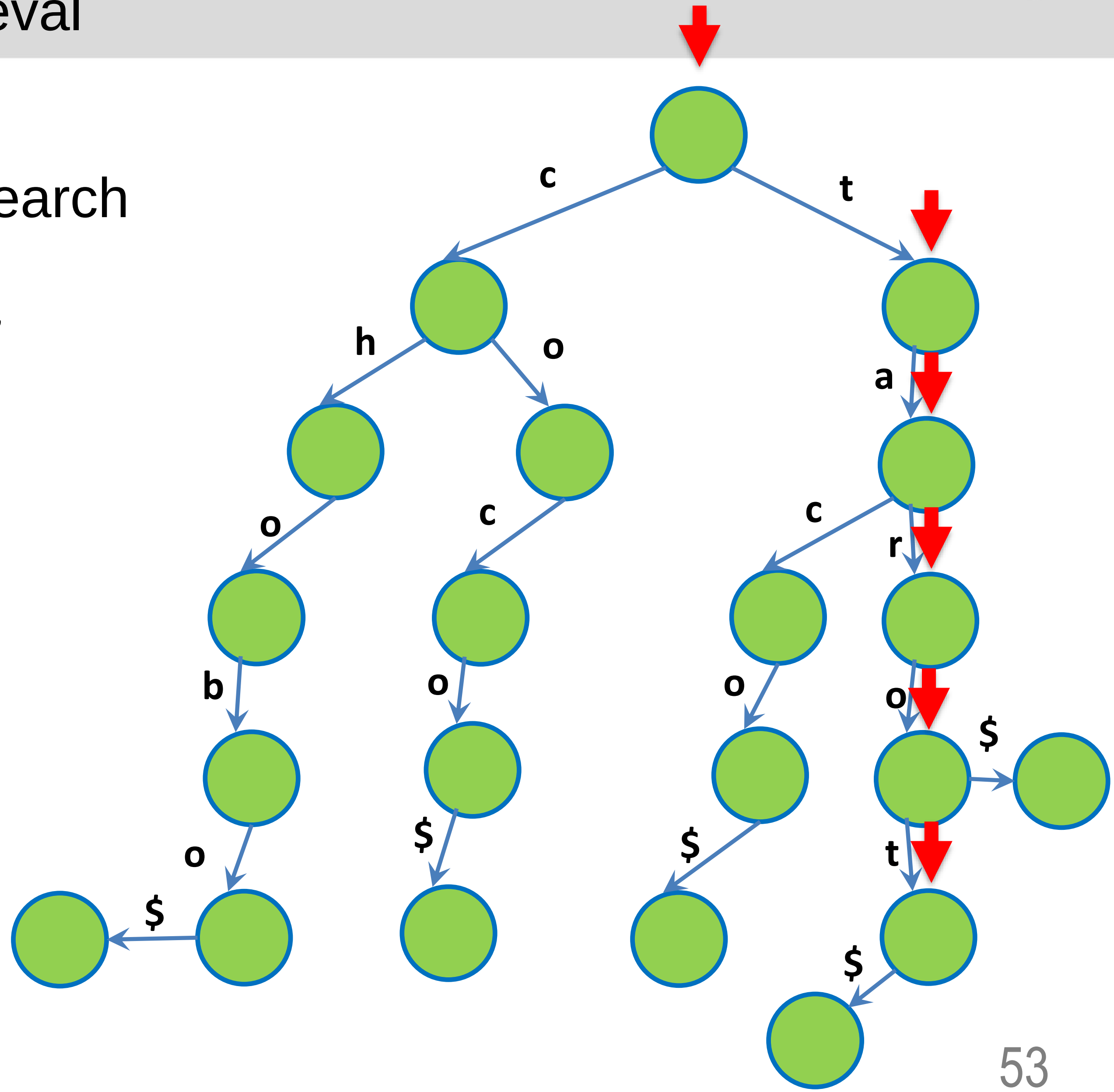
- So how do we search for retrieval?
  - Search for "tarot"



Tries

Efficient string retrieval

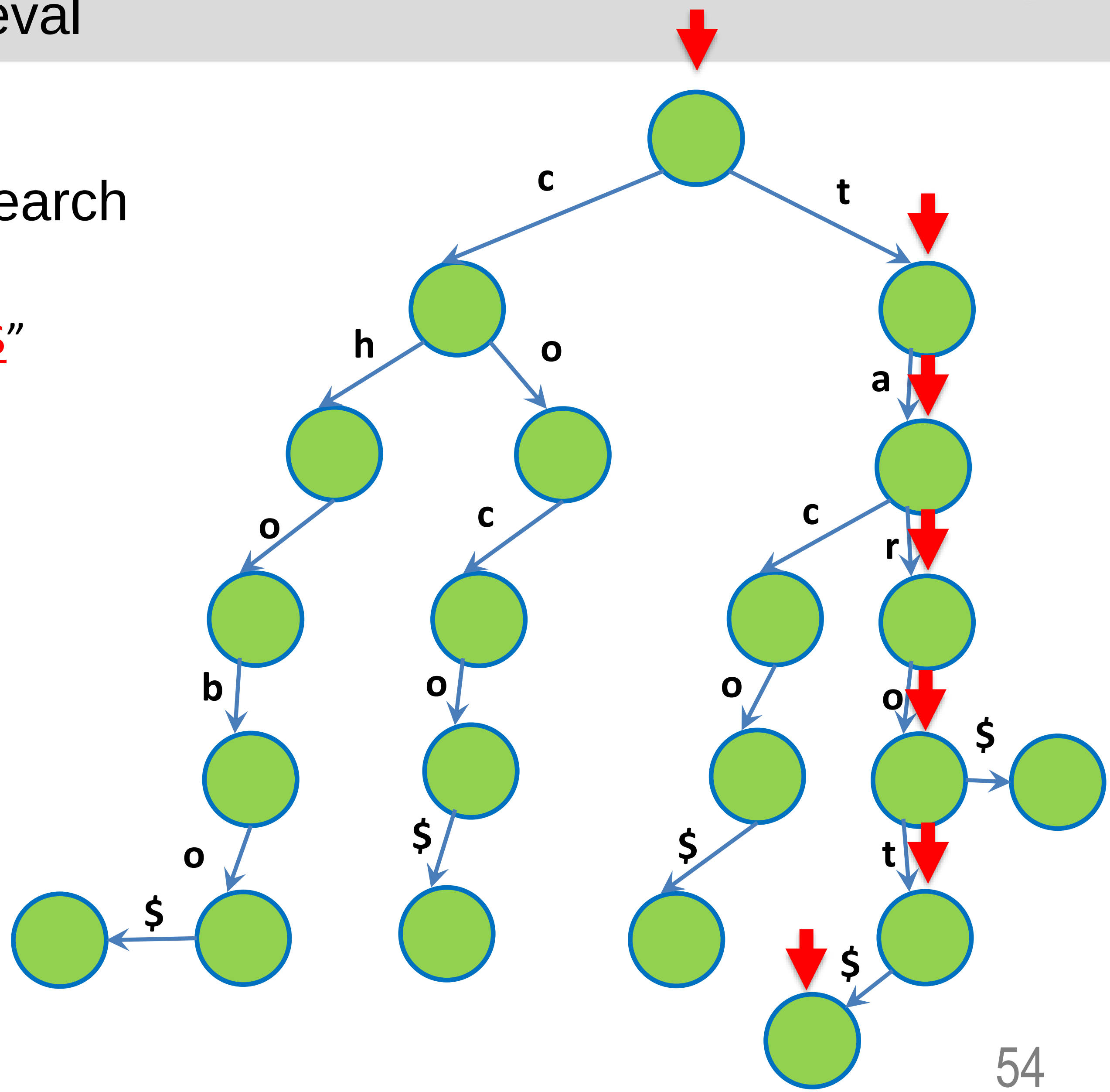
- So how do we search for retrieval?
  - Search for “taro<u>t</u>”



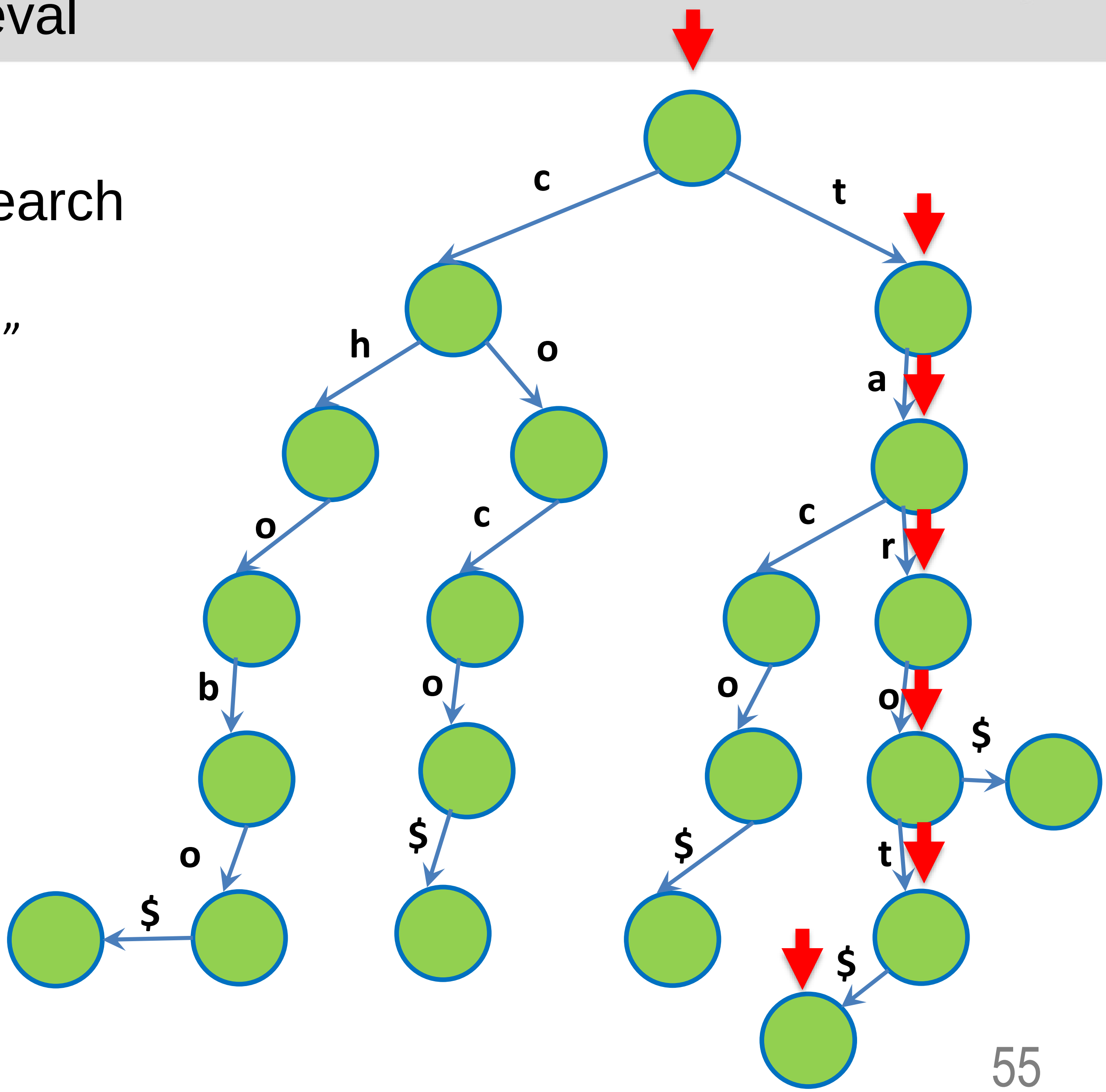
Tries

Efficient string retrieval

- So how do we search for retrieval?
  - Search for “tarot$”

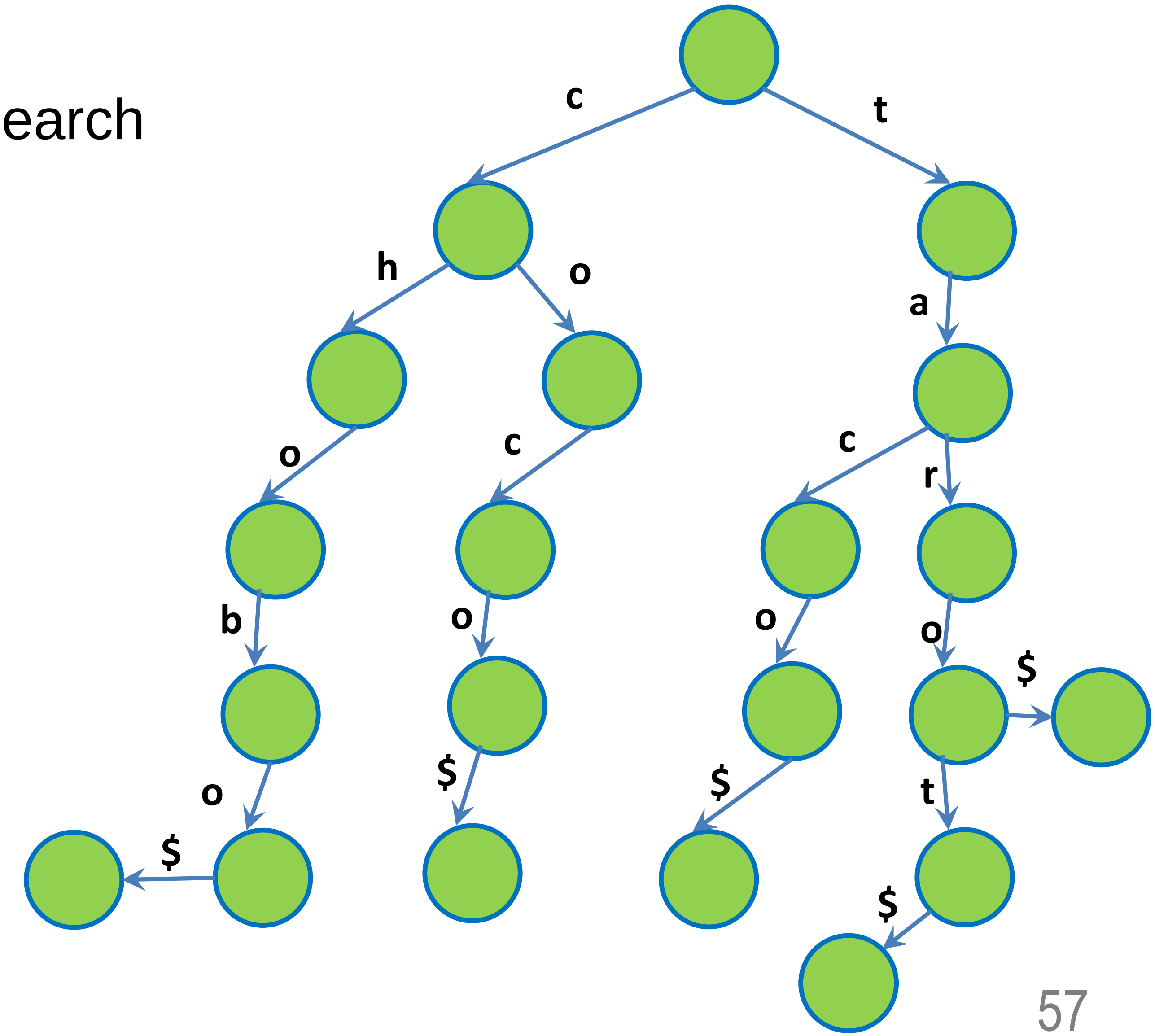


- So how do we search for retrieval?
  - Search for “tarot$”
    Found!



Questions?

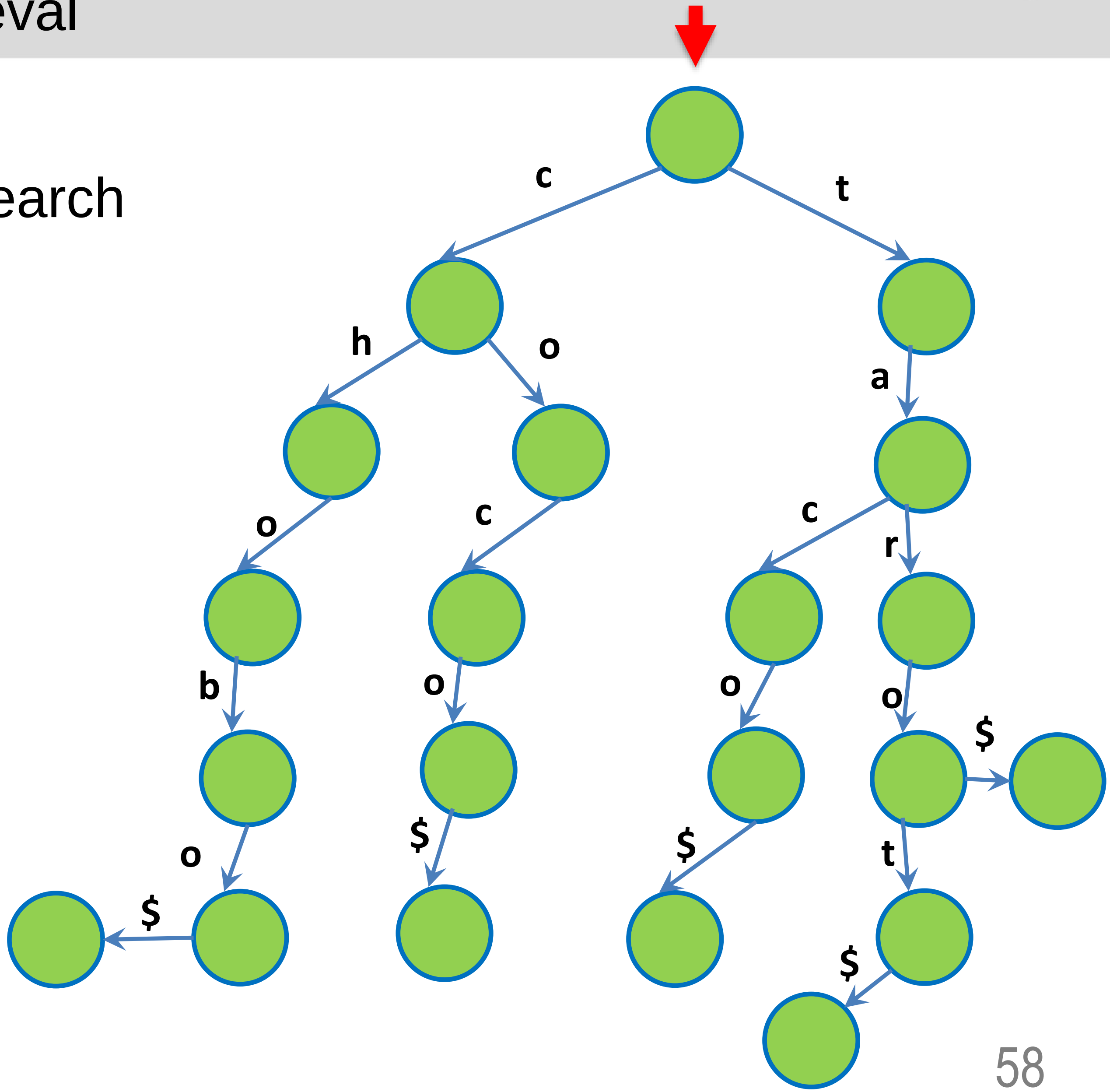
- So how do we search for retrieval?
  - Search for “cow”



Tries

Efficient string retrieval

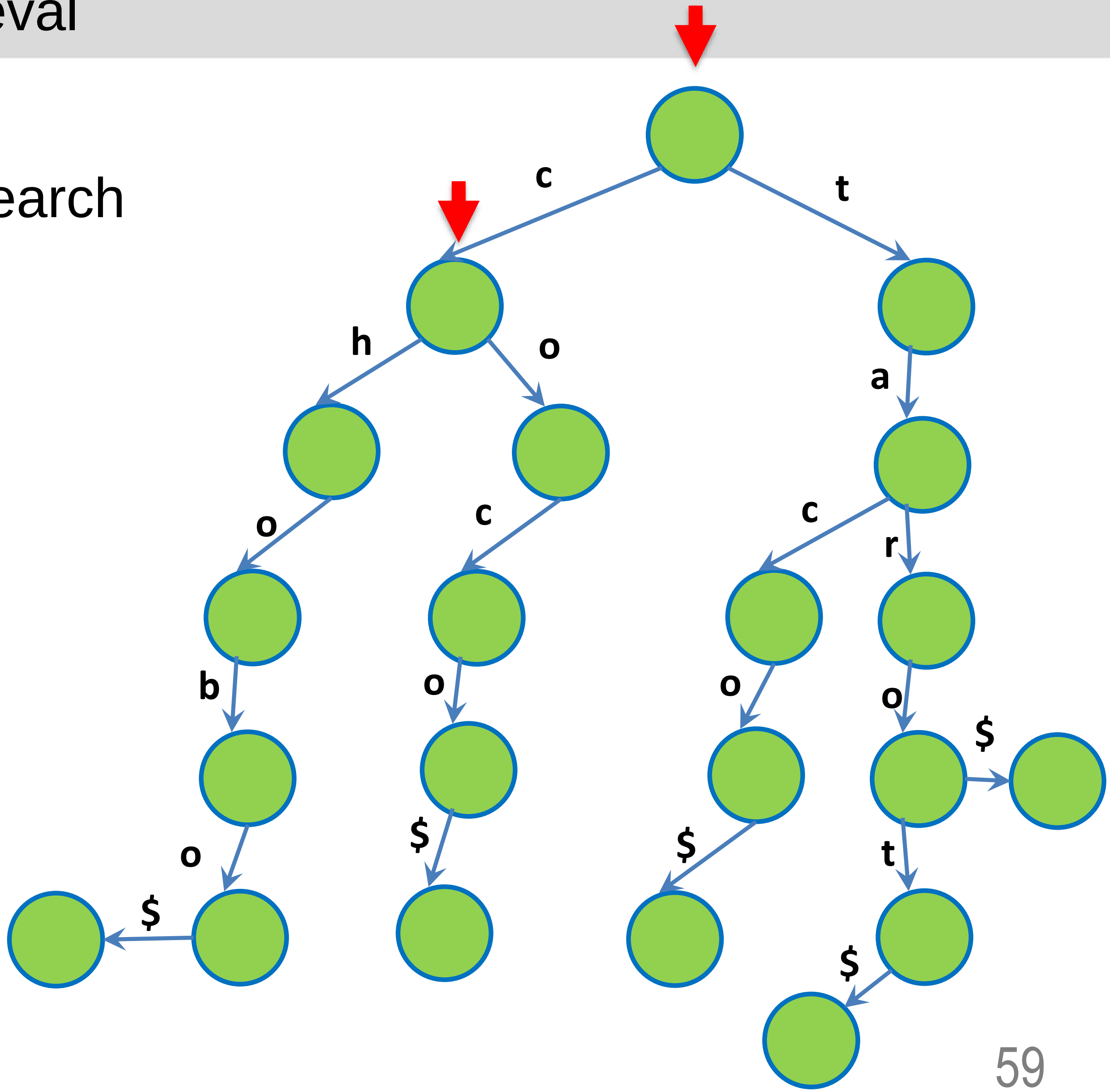
- So how do we search for retrieval?
  - Search for "cow"



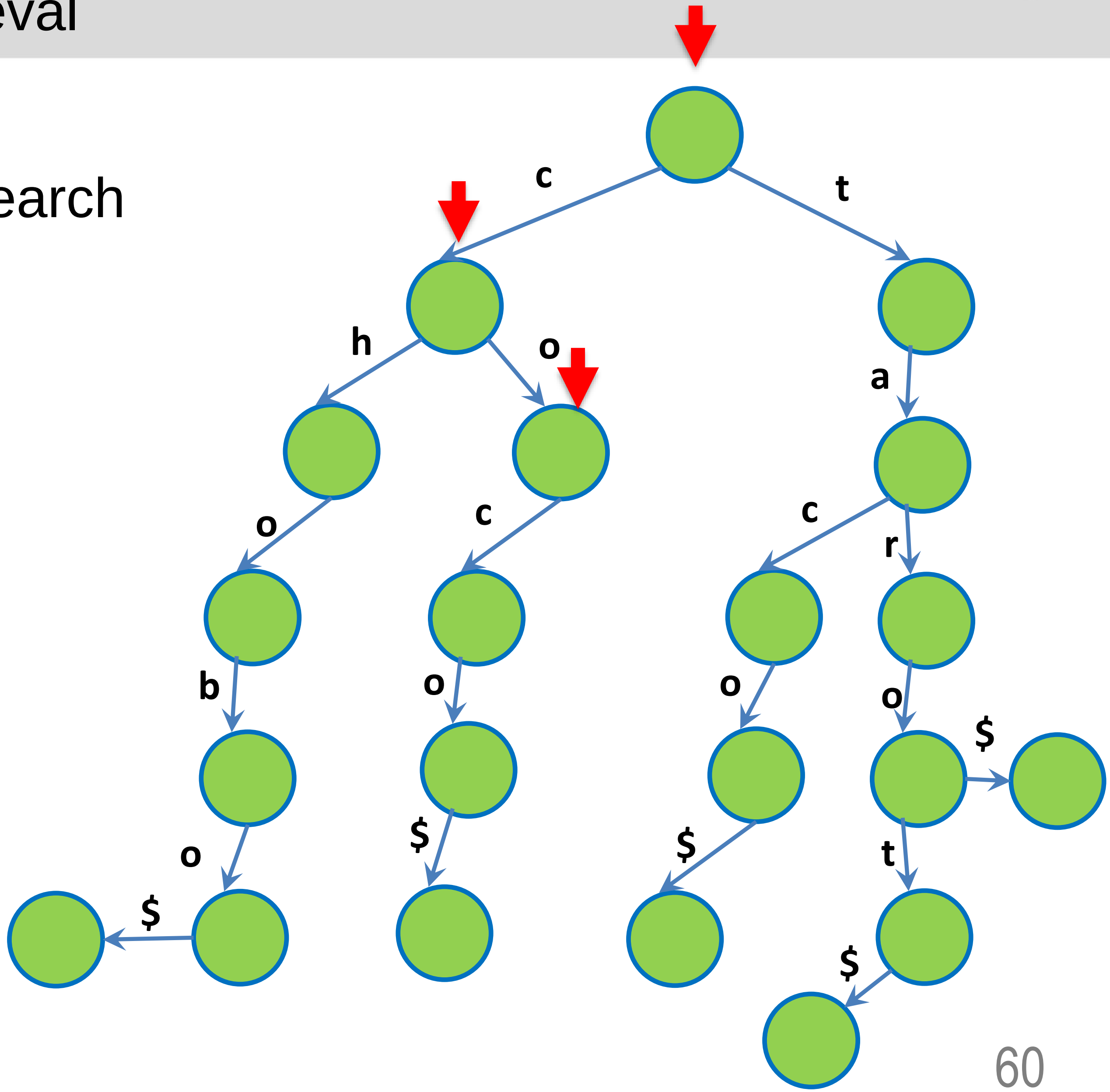
Tries

Efficient string retrieval

- So how do we search for retrieval?
  - Search for “c̲ow”



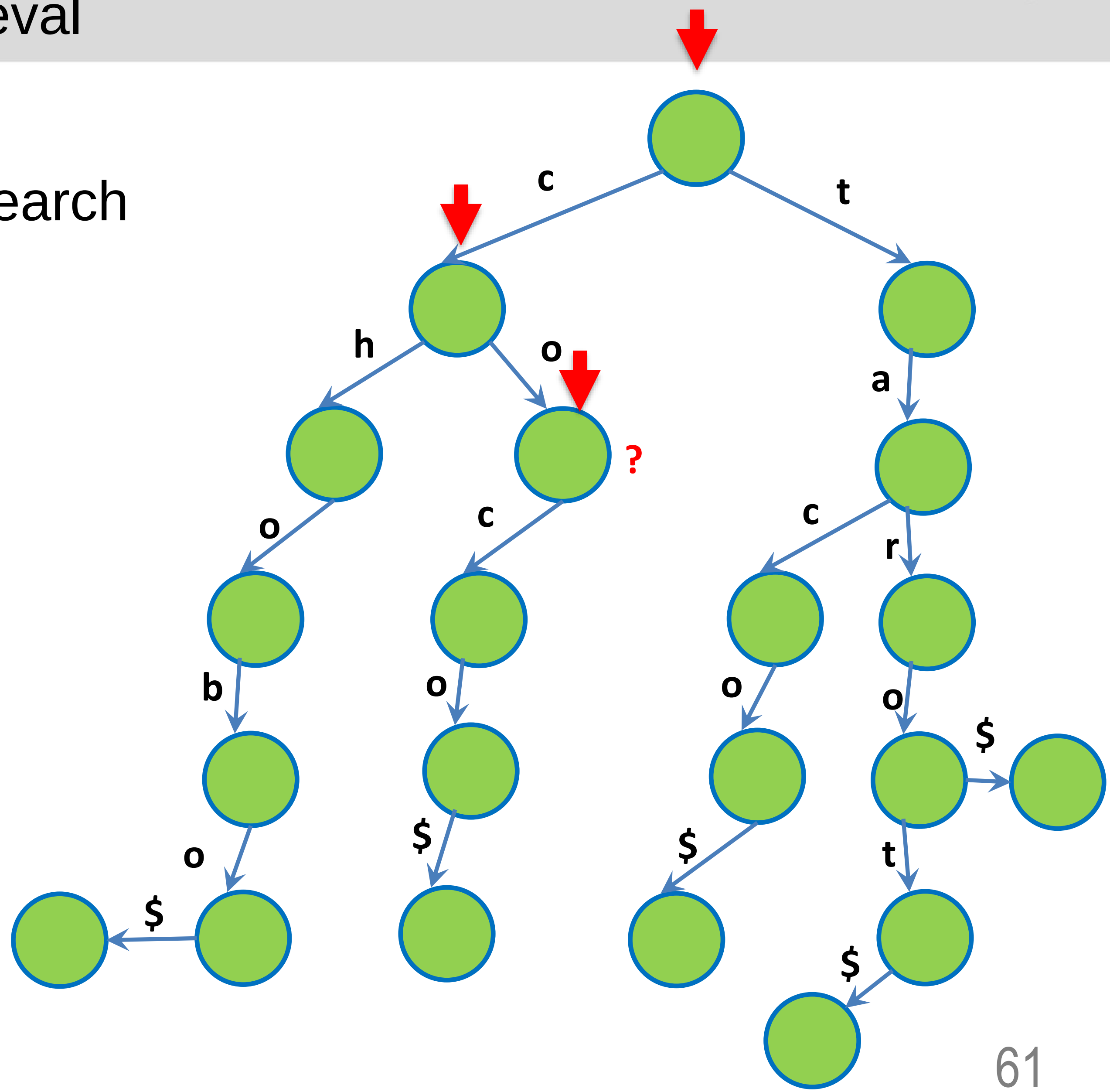
- So how do we search for retrieval?
  - Search for "c<u>o</u>w"



Tries

Efficient string retrieval

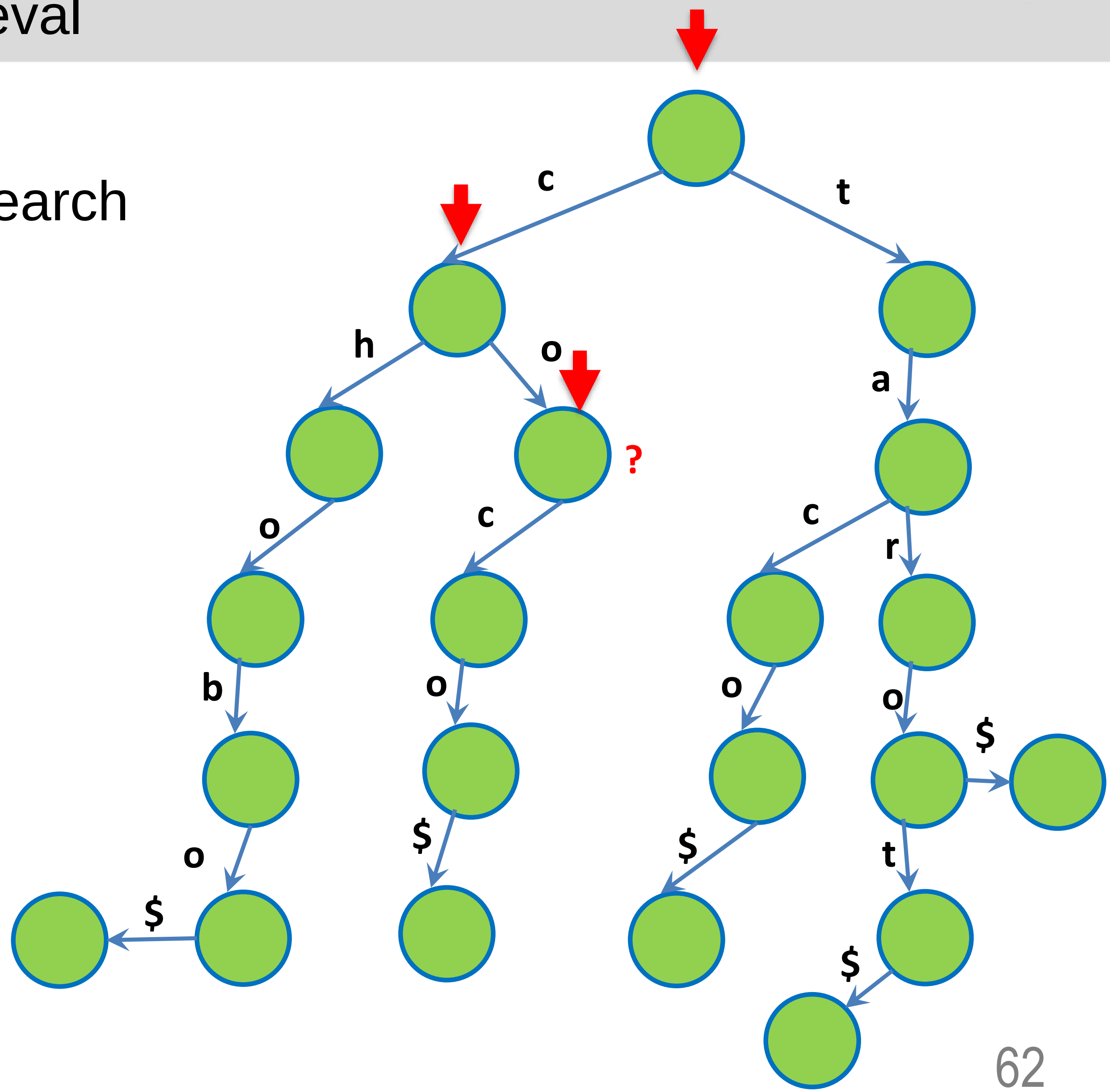
- So how do we search for retrieval?
  - Search for "co<u>w</u>"



Tries

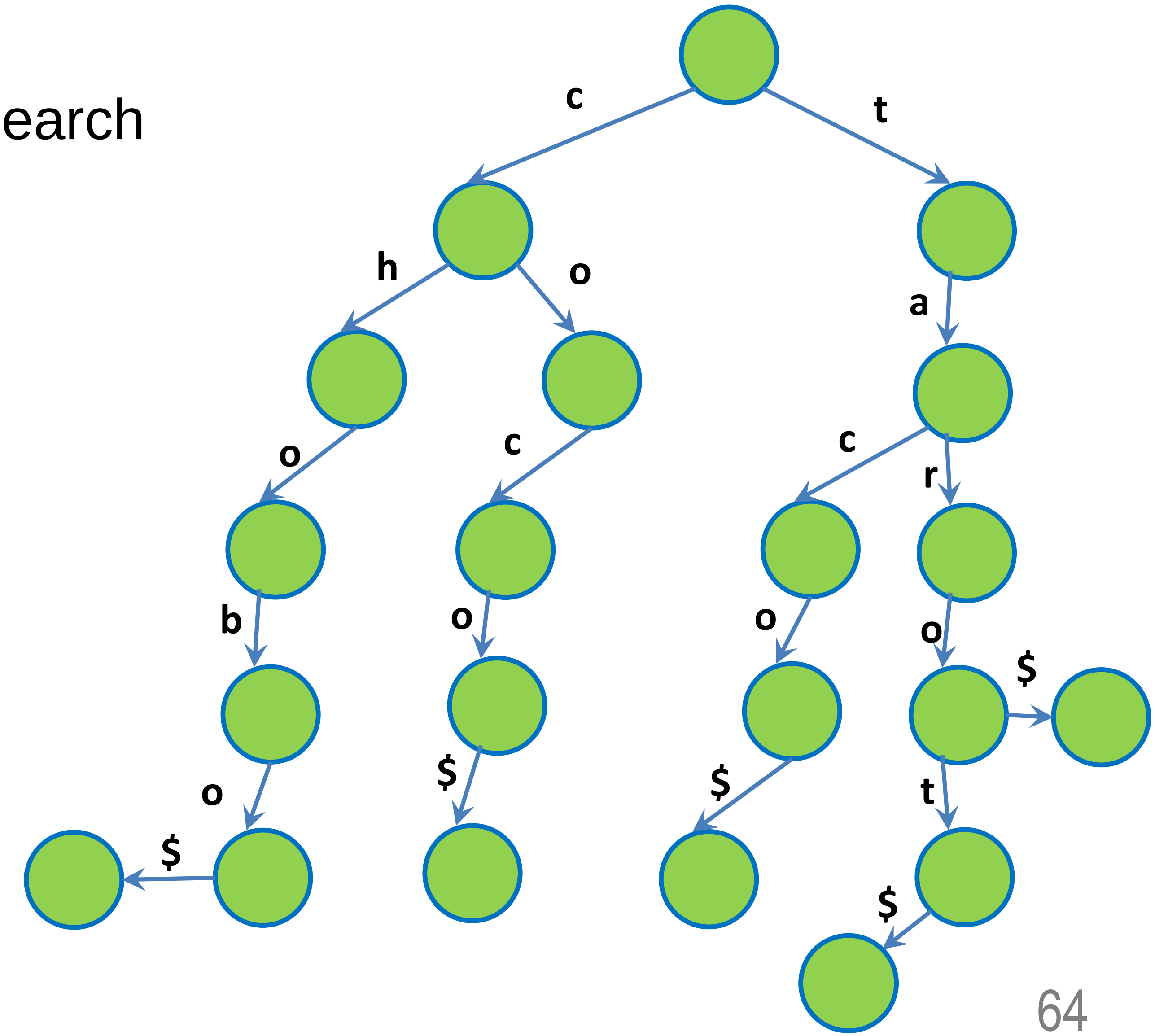
Efficient string retrieval

- So how do we search for retrieval?
  - Search for “cow”
    Not found T.T

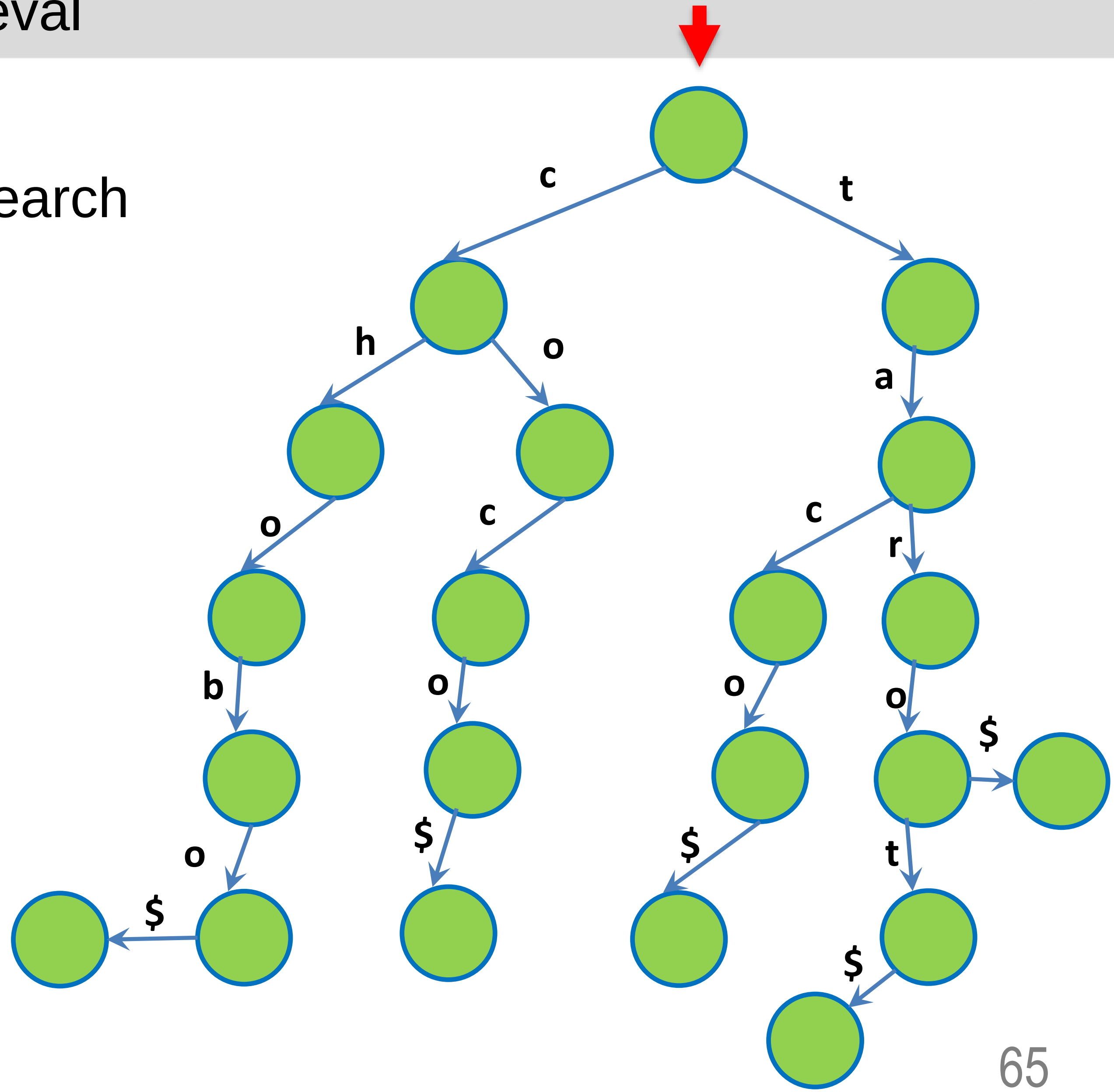


Questions?

- So how do we search for retrieval?
  – Search for “tar”

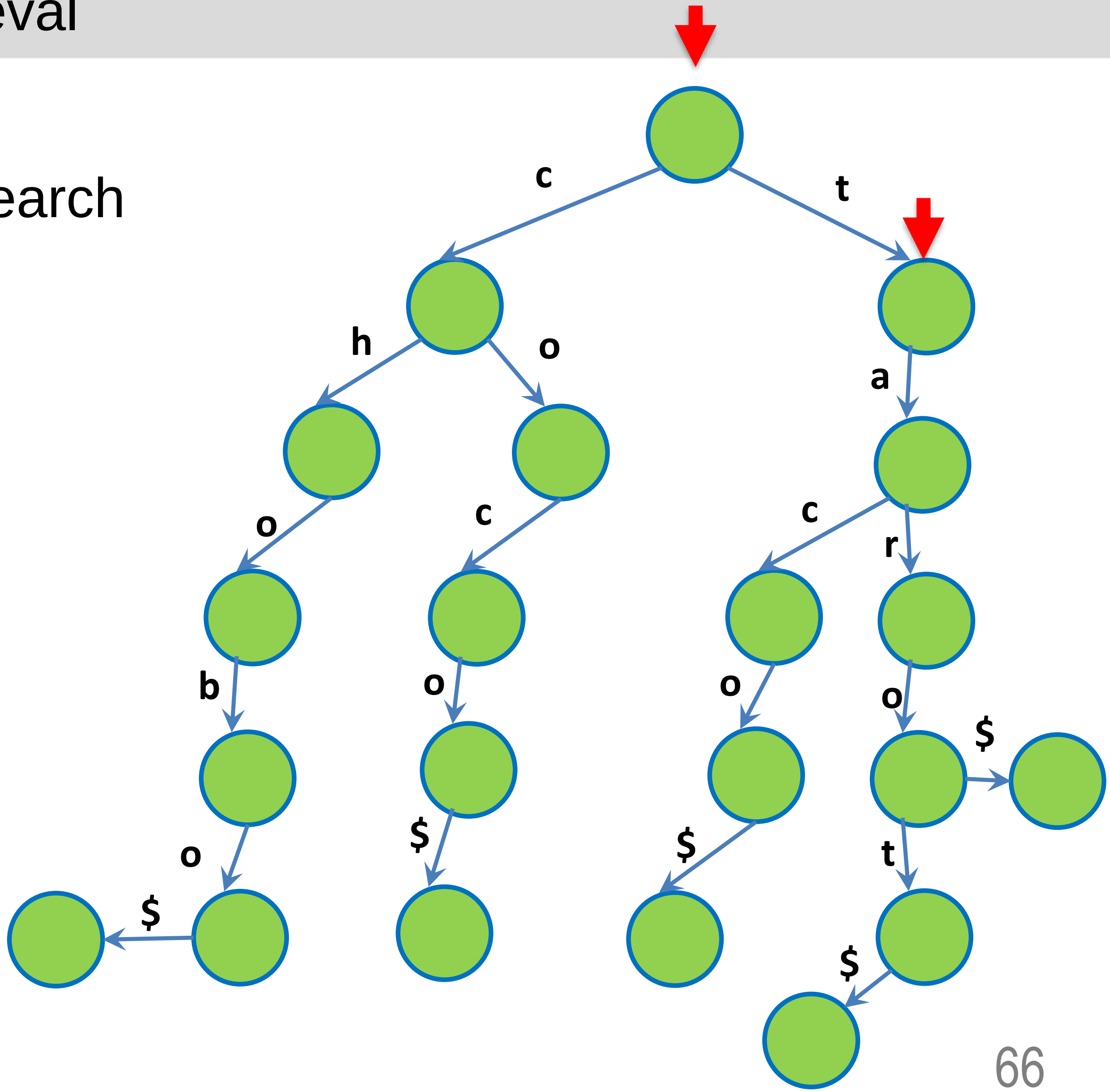


- So how do we search for retrieval?
  - Search for "tar"

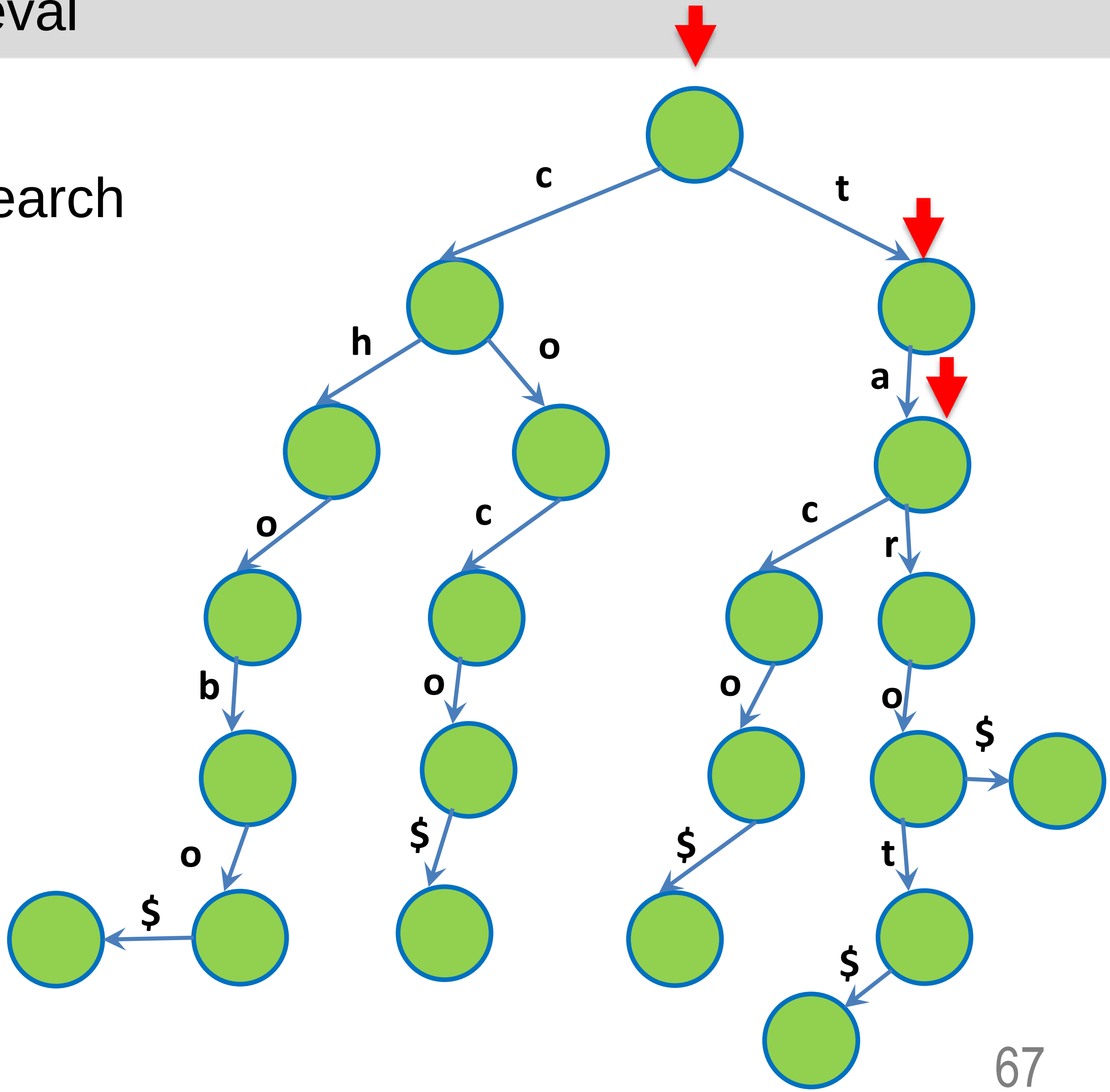


Tries
Efficient string retrieval

- So how do we search for retrieval?
  - Search for “tar”

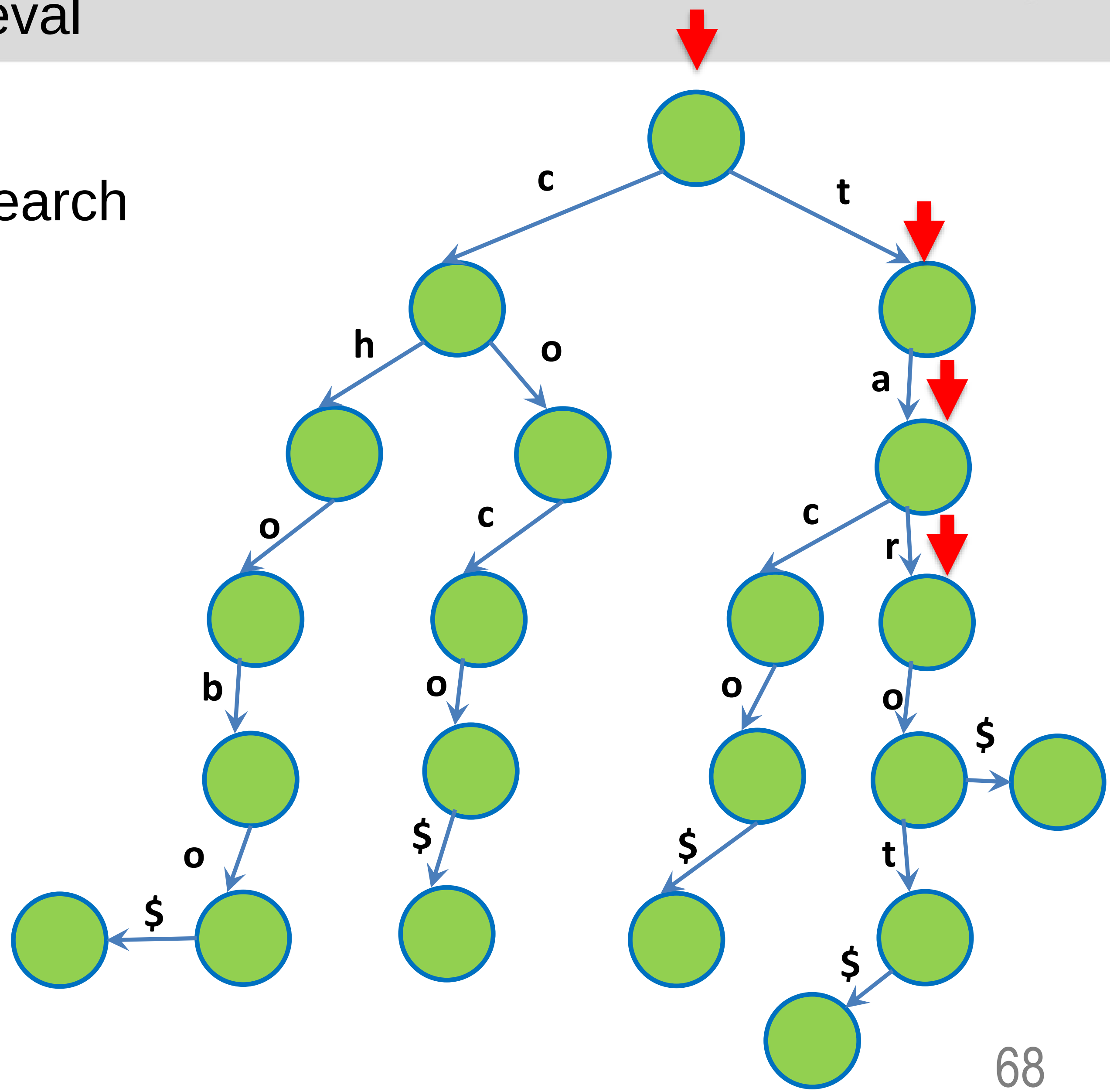


- So how do we search for retrieval?
  - Search for "t<u>a</u>r"

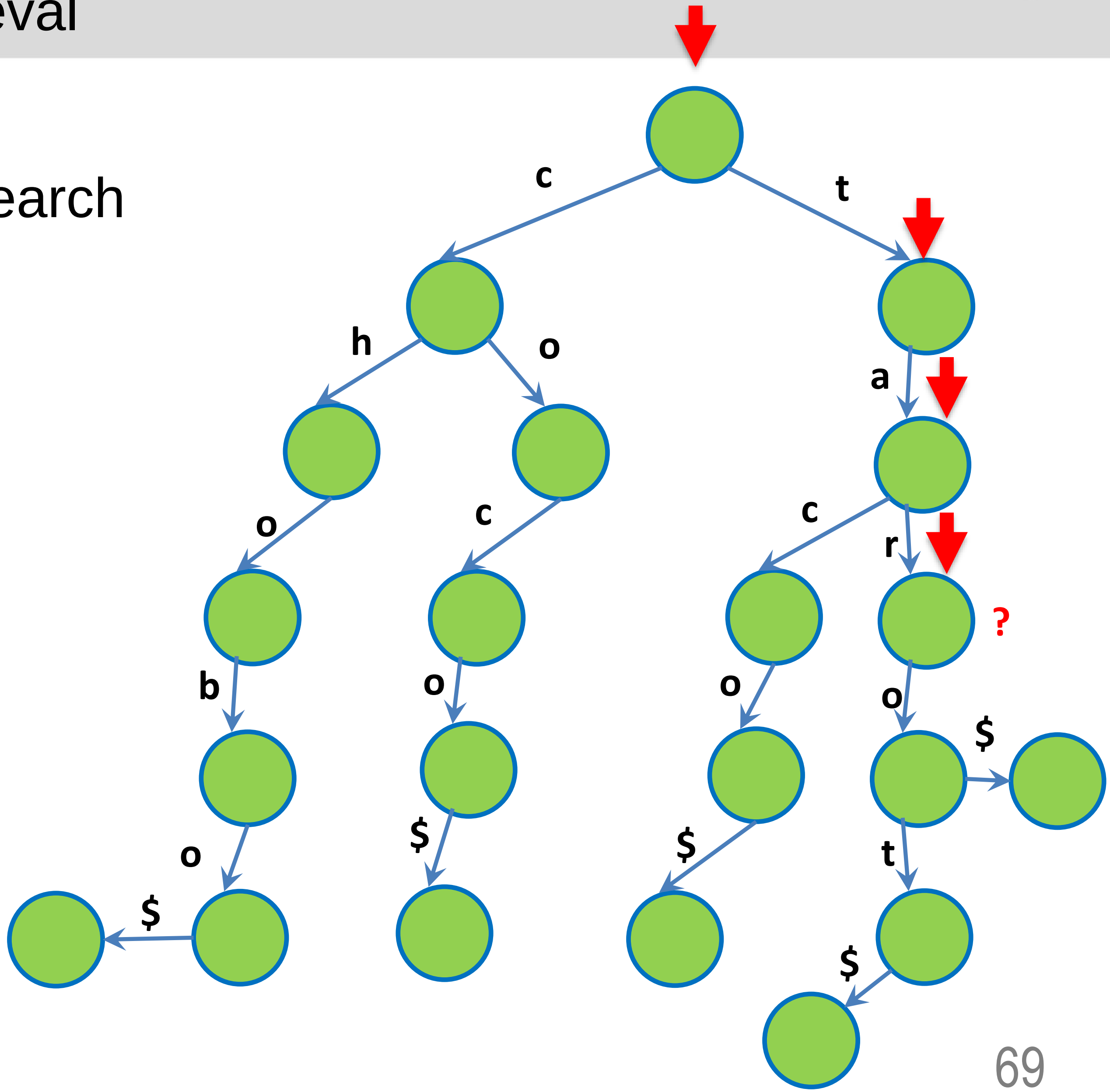


Tries
Efficient string retrieval

- So how do we search for retrieval?
  - Search for “tar”



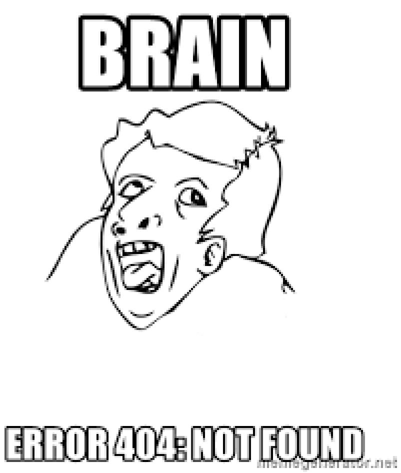
- So how do we search for retrieval?
  - Search for “tar$”

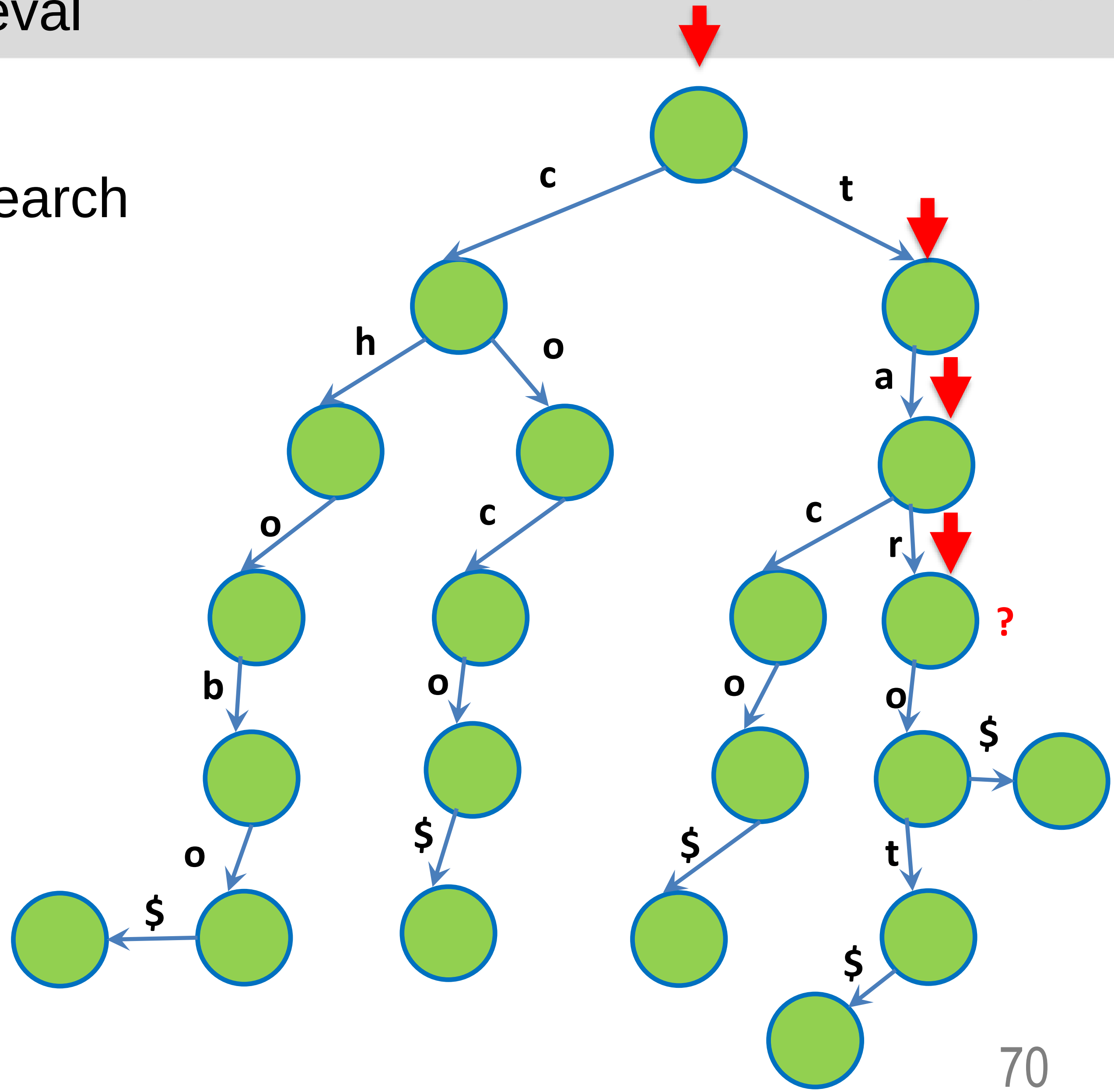


Tries

Efficient string retrieval

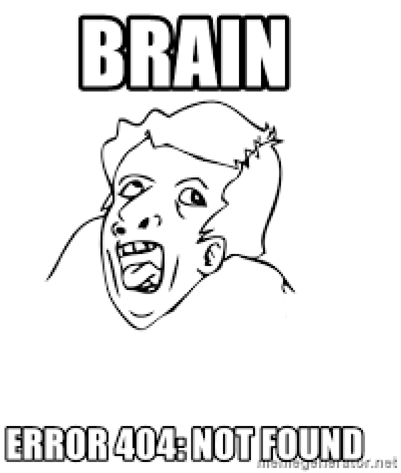
- So how do we search for retrieval?
  - Search for "tar$"
    Not found =(

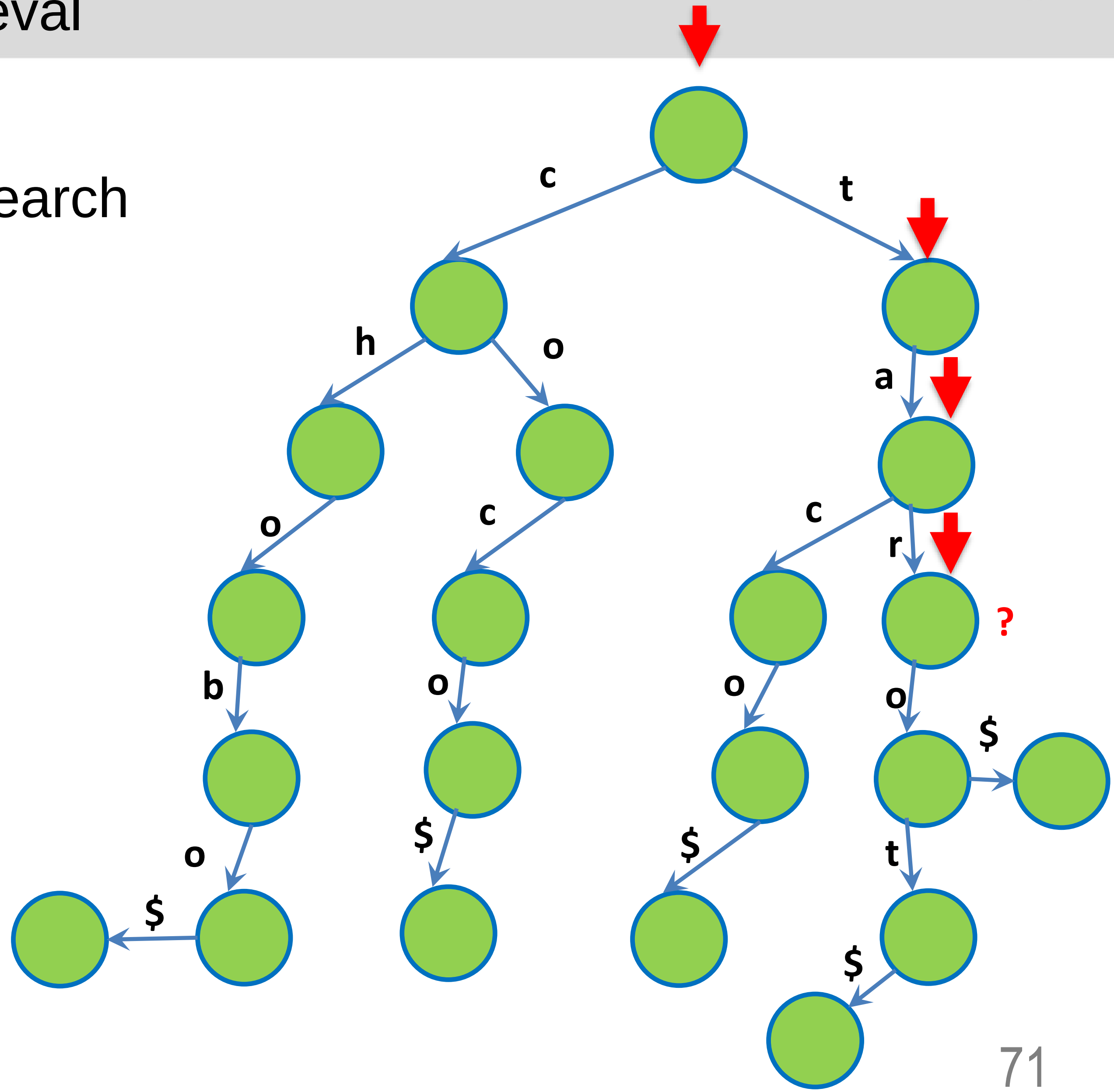




Tries
Efficient string retrieval

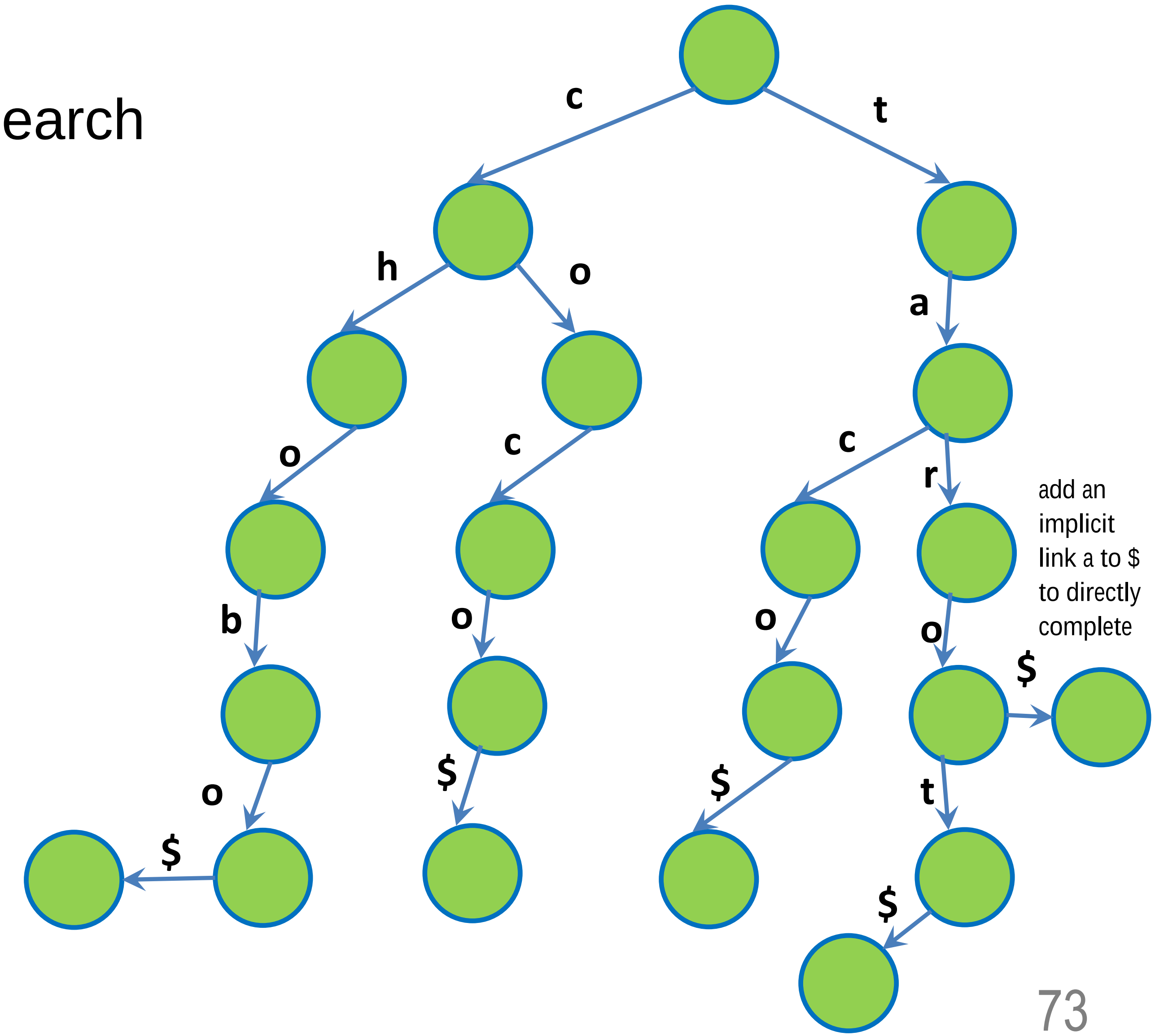
- So how do we search for retrieval?
  – Search for “tar$”
    Not found =(
    Need those $$$



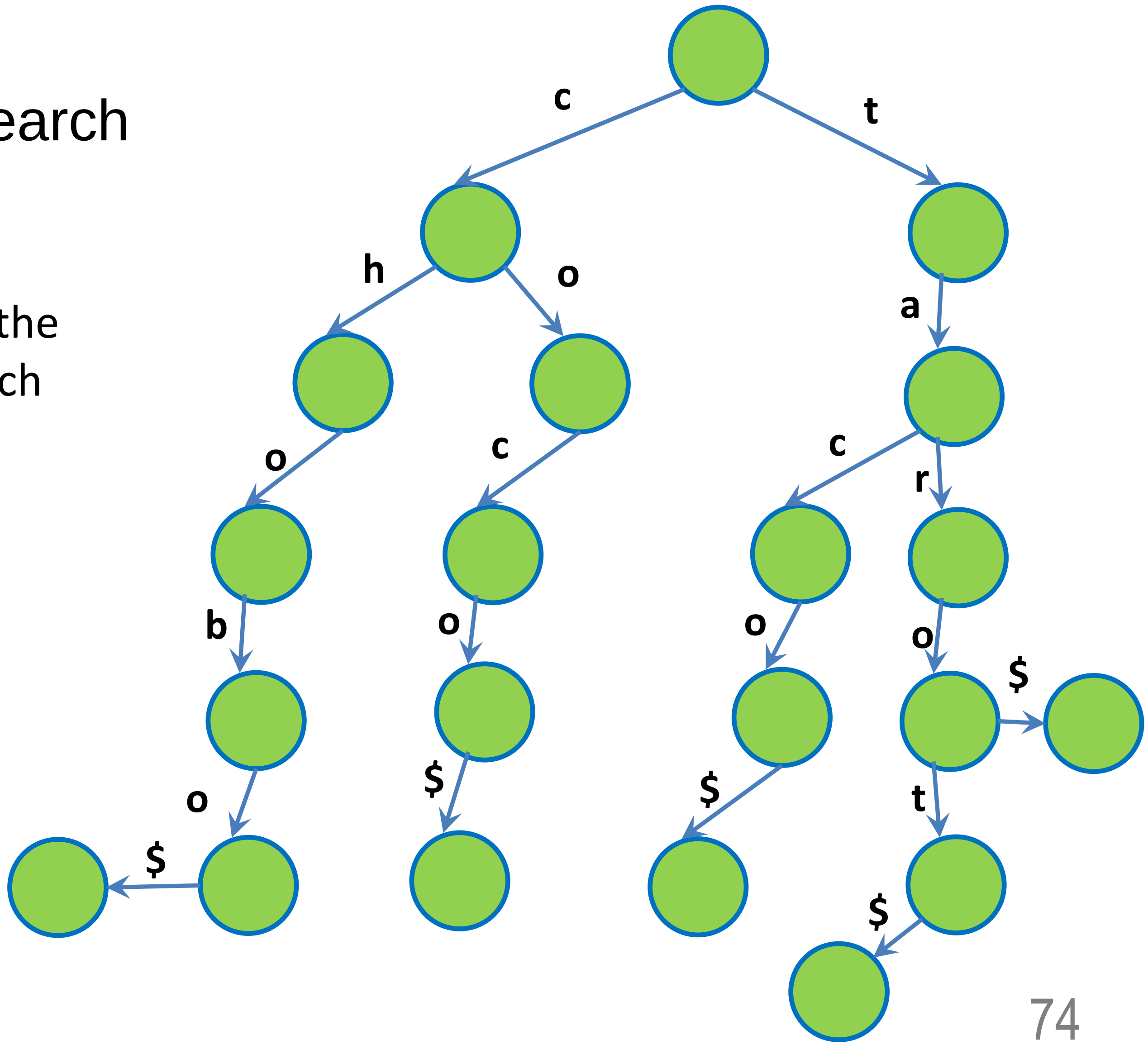


Questions?

- So how do we search for retrieval?
  - Complexity?



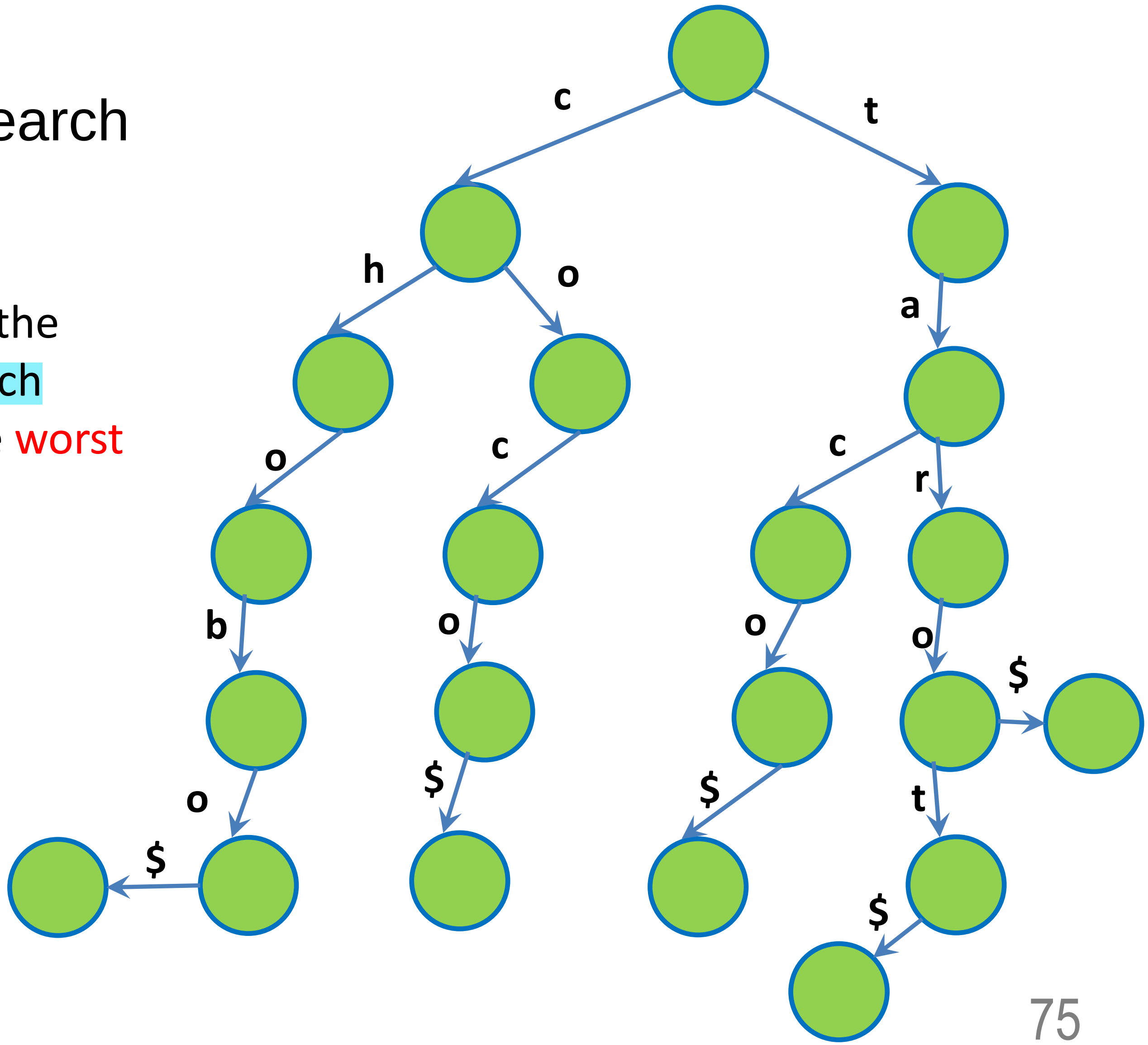
- So how do we search for retrieval?
  - Complexity?
    O(M) where M is the length of the search string…



Tries

Efficient string retrieval

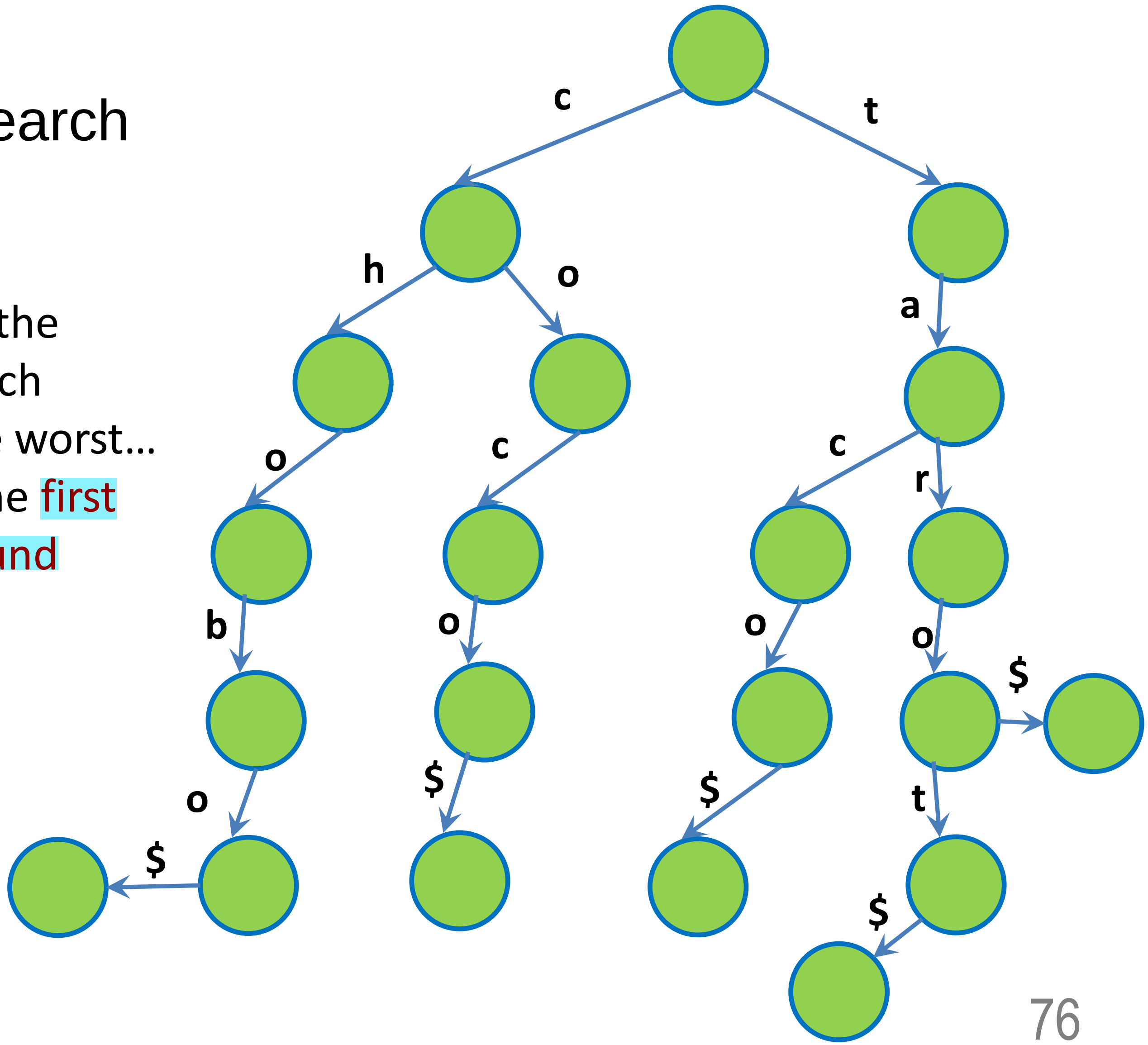
- So how do we search for retrieval?
  - Complexity?
    O(M) where M is the length of the search string… This is the worst



Tries
Efficient string retrieval

- So how do we search for retrieval?
  - Complexity?
    O(M) where M is the length of the search string… This is the worst…
    O(1) best when the first character isn't found

add implicit
link from one node
to the terminal node
by comparing
frequency
selected and
move alone the
correspond path



Questions?

- How to implement it?

- How to implement it? With OOP!

- How to implement it? With OOP!
  - Node class

```python
class Trie:
    def __init__(self):
        self.root = Node()

class Node:
    def __init__(self, data=None):
        self.data = data
        self.links = [None] * 27
```

Tries

Efficient string retrieval

- How to implement it? With OOP!
  - Node class

```python
class Trie:
    def __init__(self):
        self.root = Node()

class Node:
    def __init__(self, data=None):
        self.data = data
        self.links = [None] * 27
```

root has from index 0 to 26
in total 27 characters
each element point to another node
that has a link with multiple element in it
repeat

$ in the link has index 0

26 characters + $

  - Then we need to code the traversal from the root
    - If a link exist, travel through it
      This is O(1) due to the array data structure

Questions?

- Benefits?

Tries

Efficient string retrieval

- Benefits?
  - Better for string search than BST/ AVL
  - More versatile than hash table

- Benefits?
  - Better for string search than BST/ AVL
  - More versatile than hash table
  - Search is O(M), where M is length of string

- Benefits?
  - Better for string search than BST/ AVL
  - More versatile than hash table
  - Search is O(M), where M is length of string
  - Can sort very quickly by traversing the string
    - The edges/ links are in-order (from a to z)
    - This is O(MN)

- Benefits?
  - Better for string search than BST/ AVL
  - More versatile than hash table
  - Search is O(M), where M is length of string
  - Can sort very quickly by traversing the string
    - The edges/ links are in-order (from a to z)
    - This is O(MN)
- Disadvantage?

Tries
Efficient string retrieval

- Benefits?
  - Better for string search than BST/ AVL
  - More versatile than hash table
  - Search is O(M), where M is length of string
  - Can sort very quickly by traversing the string
    - The edges/ links are in-order (from a to z)
    - This is O(MN)
- Disadvantage?
  - At times can be slower than hash table
  - Wasted space if the self.link array is left empty most of the time

waste about 26 memory in link since storing 26 characters in order

Questions?

Usage?

- Height of the trie = length of the longest string

Tries

Usage?

- Height of the trie = length of the longest string
- Complexity is based on the length of the string we are inserting/ deleting/ searching

Usage?

- Height of the trie = length of the longest string
- Complexity is based on the length of the string we are inserting/ deleting/ searching
- We can search for the prefix of strings!
  - Useful for auto correct/ auto complete

Usage?

- Height of the trie = length of the longest string
- Complexity is based on the length of the string we are inserting/ deleting/ searching
- We can search for the prefix of strings!
  - Useful for auto correct/ auto complete
  - And many other applications!

Questions?

Suffix Trie
For suffixes

- Same as a trie
- But for suffixes

Suffix Trie
For suffixes

- Can you make a suffix trie for apple?

Suffix Trie

For suffixes

- Can you make a suffix trie for apple?
- List all the suffixes
  - Apple$
  - Pple$
  - Ple$
  - Le$
  - E$

Suffix Trie

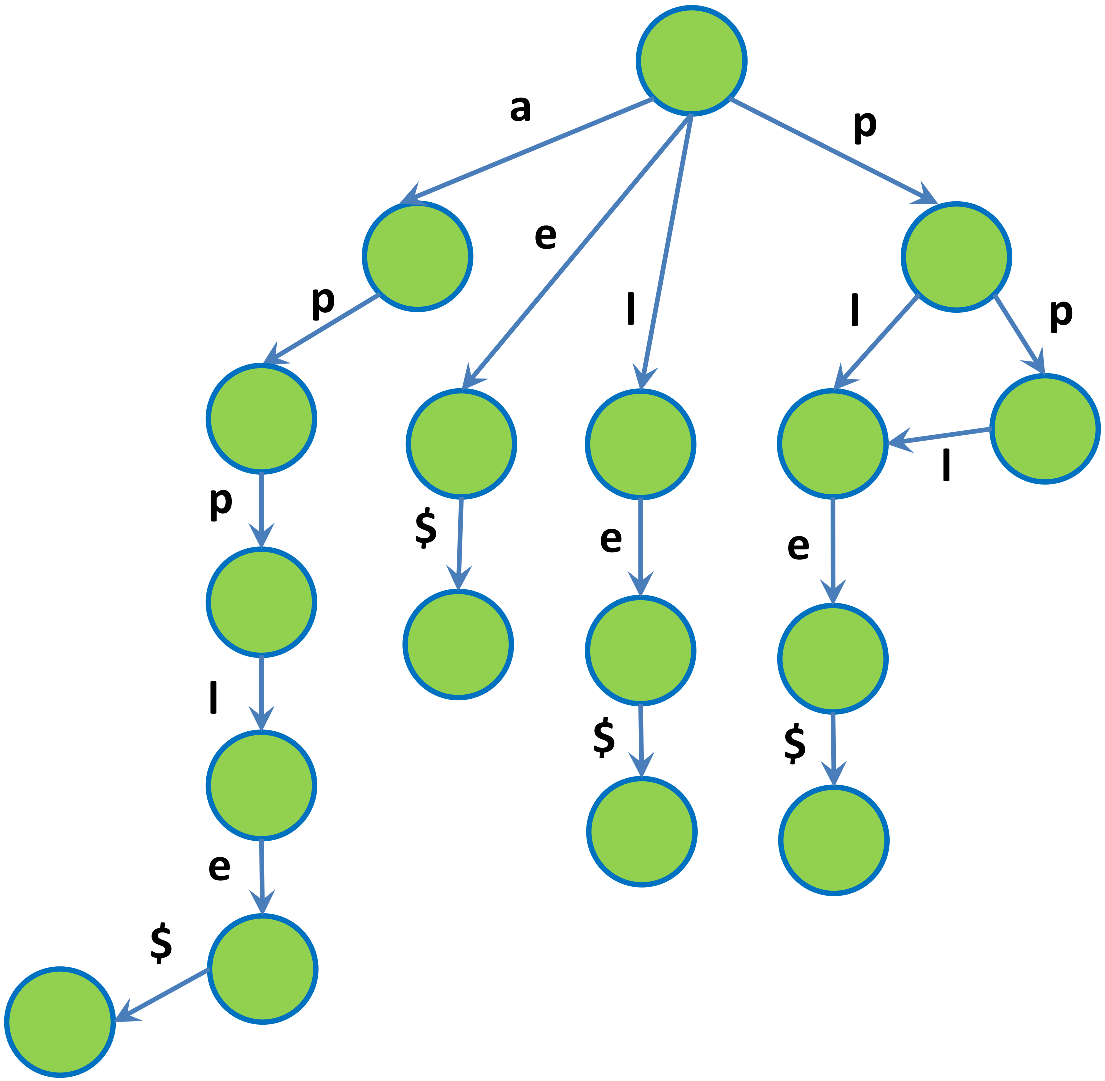
For suffixes

- Can you make a suffix trie for apple?
- List all the suffixes
  - Apple$
  - Pple$
  - Ple$
  - Le$
  - E$
- Then we just make the trie like earlier

Suffix Trie

For suffixes

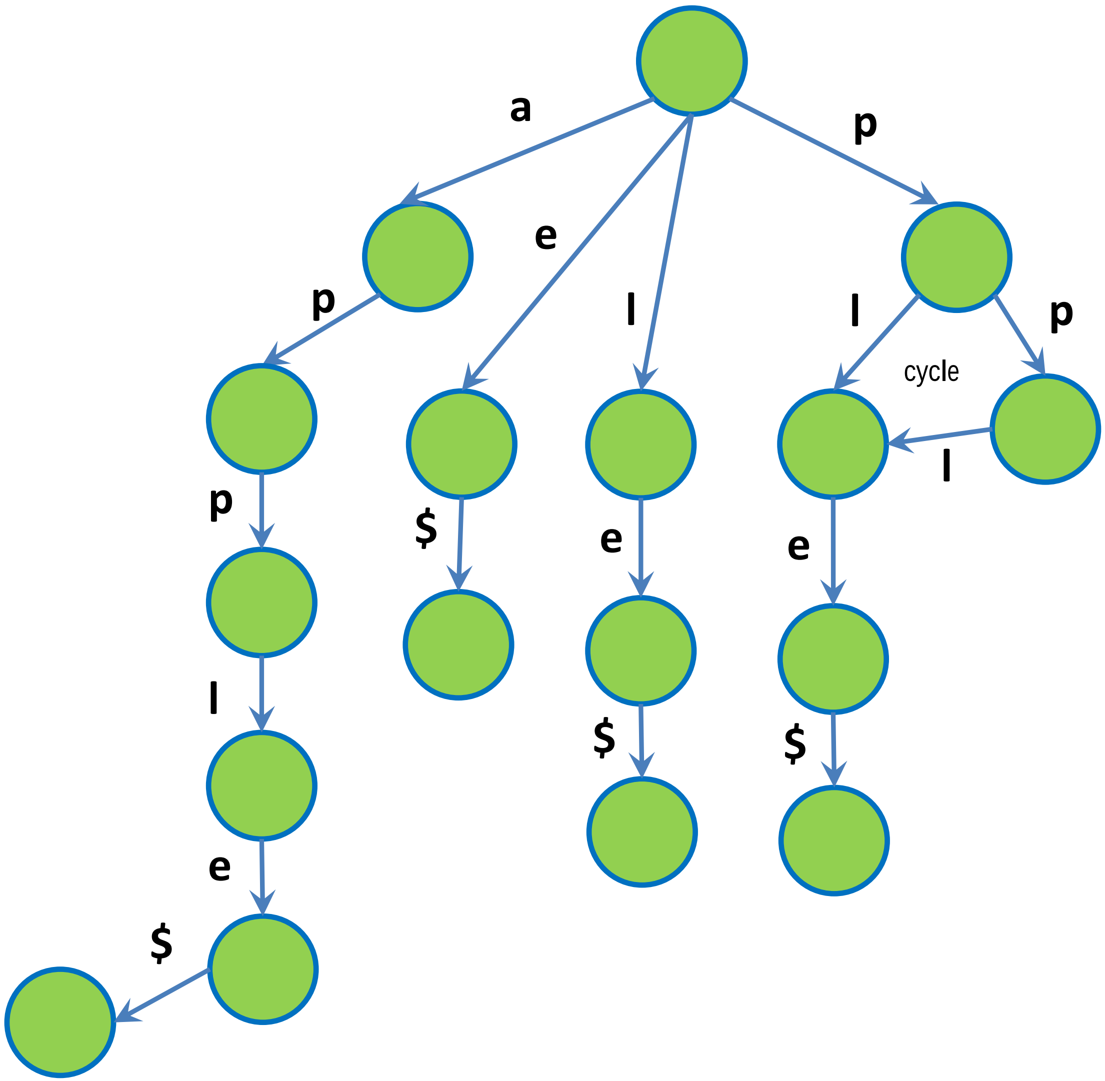
- Can you make a suffix trie for apple?
- List all the suffixes
  - Apple$
  - Pple$
  - Ple$
  - Le$
  - E$
- Then we just make the trie like earlier
- Is this right?



Suffix Trie

For suffixes

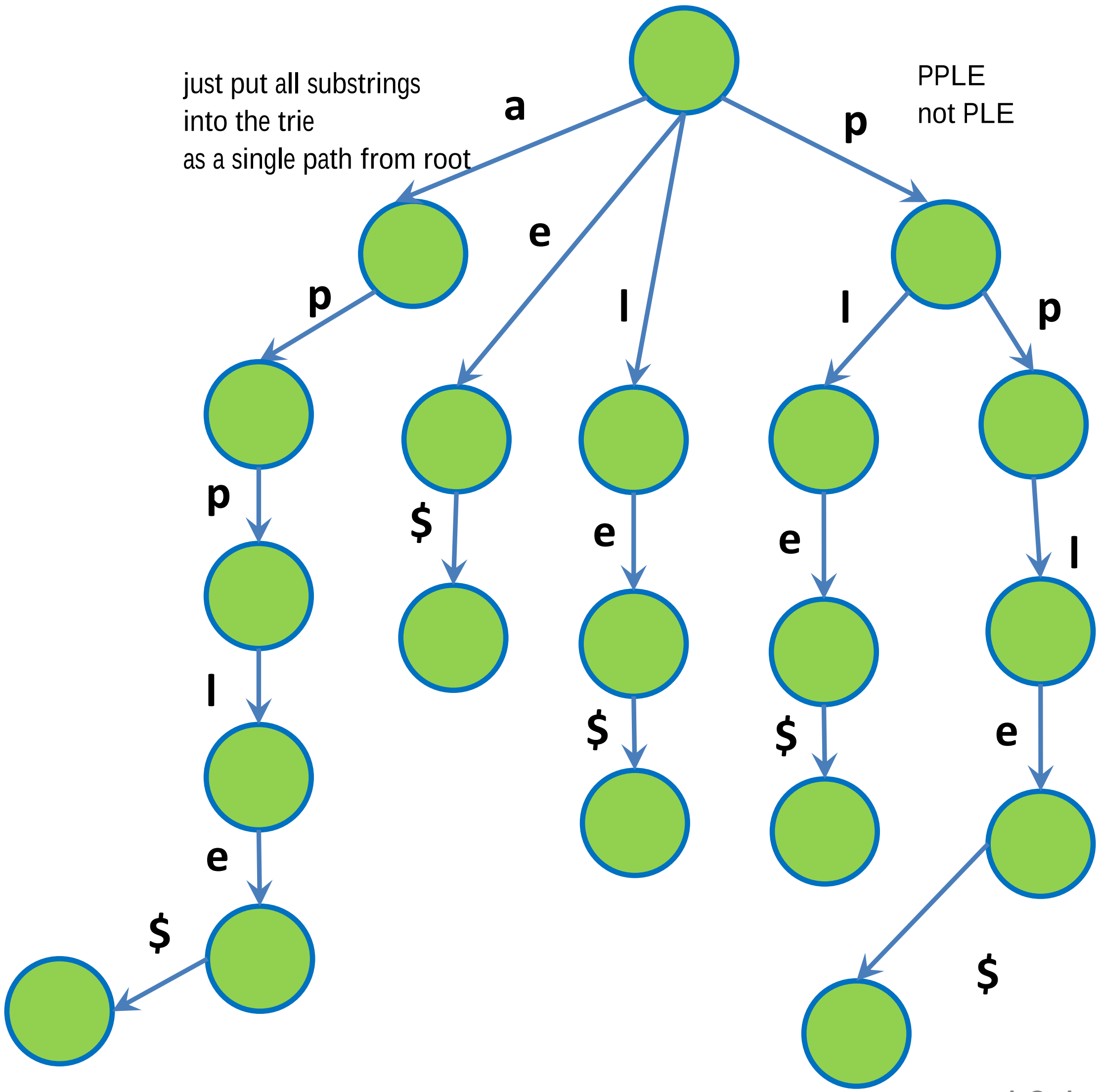
- Can you make a suffix trie for apple?
- List all the suffixes
  - Apple$
  - Pple$
  - Ple$
  - Le$
  - E$
- Then we just make the trie like earlier
- Is this right?
  NO! CYCLE!
  so this is wrong…



Suffix Trie

For suffixes

- Can you make a suffix trie for apple?
- List all the suffixes
  - Apple$
  - Pple$
  - Ple$
  - Le$
  - E$
- Then we just make the trie like earlier



Questions?

Applications?

- Same as earlier
- But more goodies now!

Suffix Trie

Applications?

- Same as earlier
- But more goodies now!
  - We can now find substring
    substring = prefix of a suffix

Suffix Trie
Applications?

- Same as earlier
- But more goodies now!
  - We can now find substring
    substring = prefix of a suffix
  - We can find the number of occurrences as well

- Same as earlier
- But more goodies now!
  - We can now find substring
    substring = prefix of a suffix
  - We can find the number of occurrences as well
    Number of leave nodes!
    Same for substrings!

- Same as earlier
- But more goodies now!
  - We can now find substring
    substring = prefix of a suffix
  - We can find the number of occurrences as well
    Number of leave nodes!
    Same for substrings!
  - Finding longest repeated substring

Suffix Trie
Applications?

- Same as earlier
- But more goodies now!
  - We can now find substring
    substring = prefix of a suffix
  - We can find the number of occurrences as well
    Number of leave nodes!
    Same for substrings!
  - Finding longest repeated substring
    Deepest node with at least 2 children

Suffix Trie
Applications?

- Same as earlier
- But more goodies now!
  - We can now find substring
    substring = prefix of a suffix
  - We can find the number of occurrences as well
    Number of leave nodes!
    Same for substrings!
  - Finding longest repeated substring
    Deepest node with at least 2 children

Suffix Trie
Applications?

- Same as earlier
- But more goodies now!
  - We can now find substring
    <u>substring = prefix of a suffix</u>
  - We can find the number of occurrences as well
    Number of leave nodes!
    Same for substrings!
  - Finding longest repeated substring
    Deepest node with at least 2 children

MONASH University

- Same as earlier
- But more goodies now!
  - We can now find substring
    substring = prefix of a suffix
  - We can find the number of occurrences as well
    Number of leave nodes!
    Same for substrings!
  - Finding longest repeated substring
    Deepest node with at least 2 children

- And many more…

Questions?

Suffix Trie
Applications?

- Space complexity?

Suffix Trie
Applications?

- Space complexity?
  – $O(N^2)$

Suffix Trie

Applications?

- Space complexity?
  - O(N^2)
  - N suffixes, longest suffix is N character

Suffix Trie
Applications?

- Space complexity?
  - O(N^2)
  - N suffixes, longest suffix is N character
  - Have N number of leaves!

Questions?

- What is a suffix tree?

- What is a suffix tree?

- What is a suffix tree?

Suffix Trie Suffix Tree

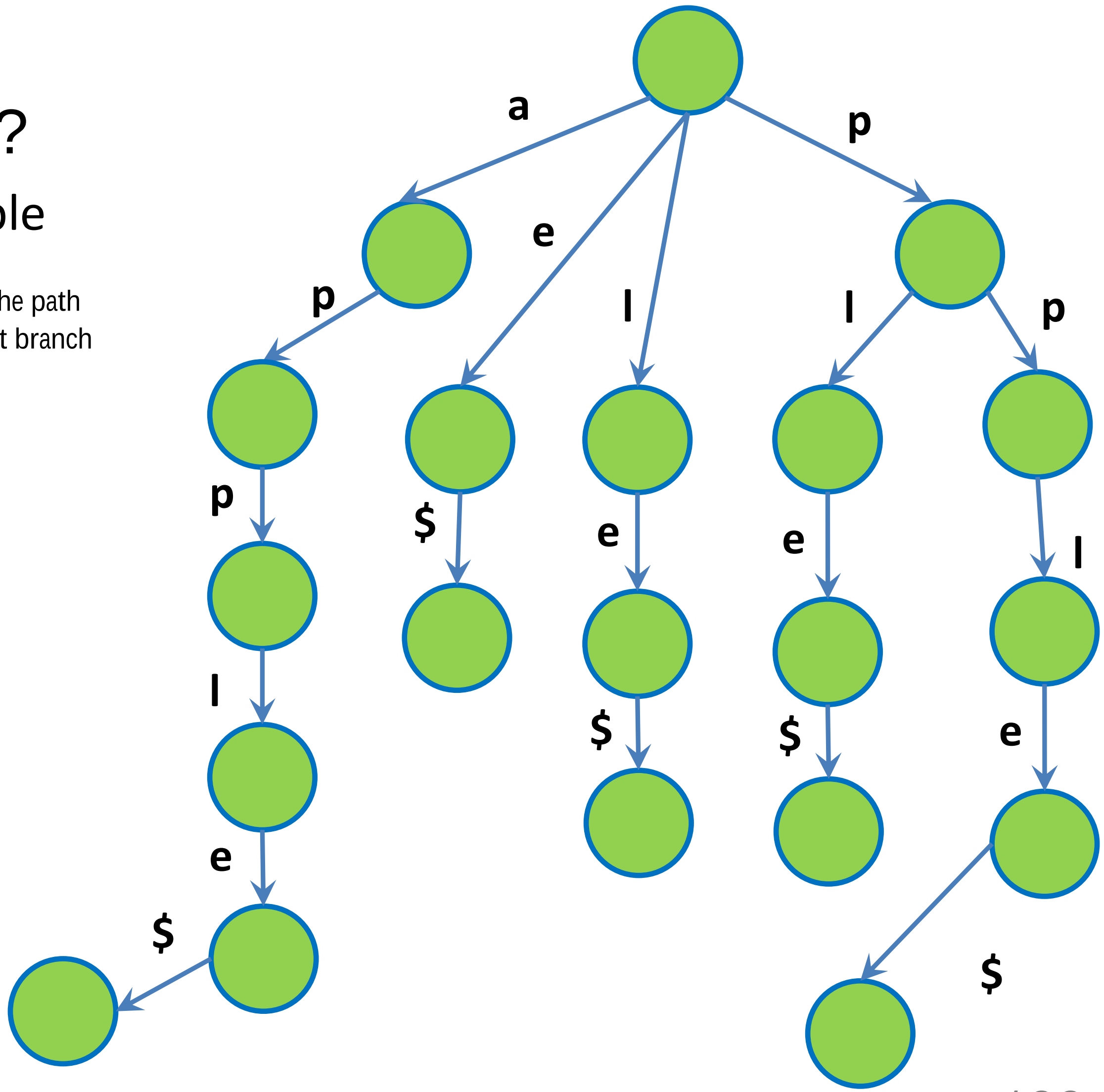
- What is a suffix tree?
  - Using our same example

Suffix Tree

A tree, not a trie

- What is a suffix tree?
  - Using our same example

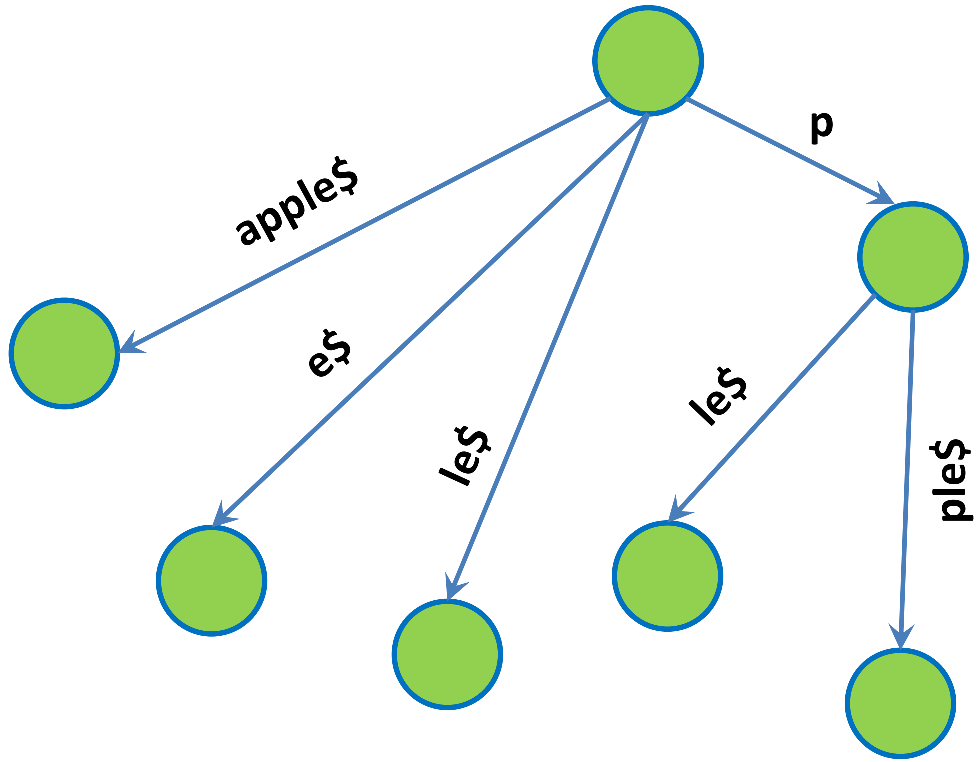
compress the path
that has not branch



Suffix Tree

A tree, not a trie

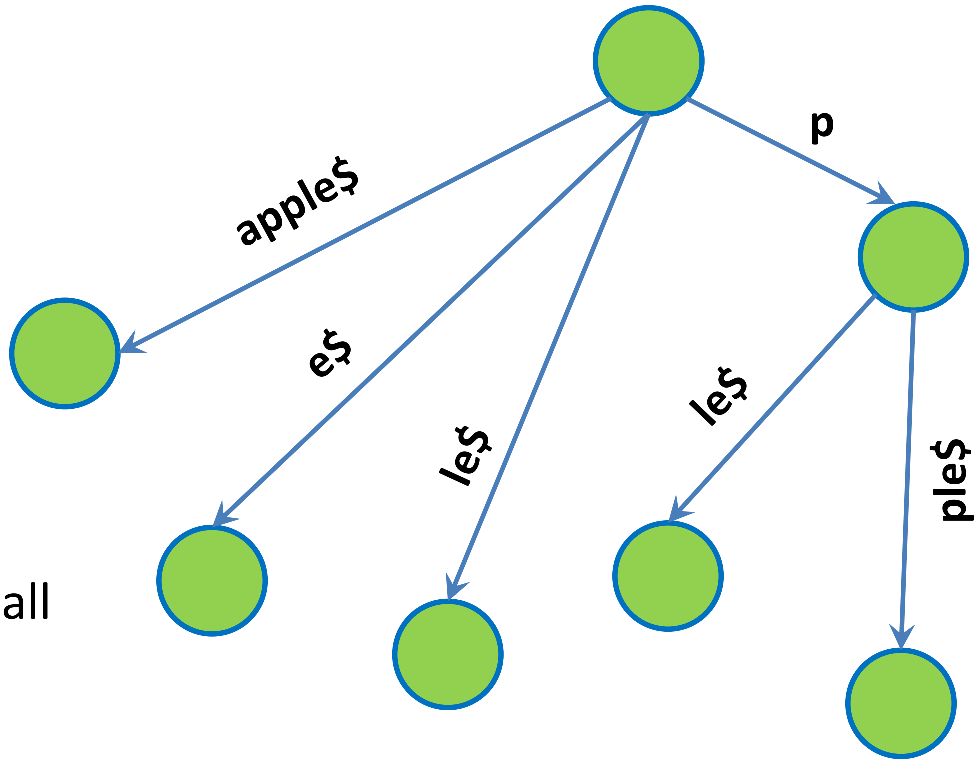
- What is a suffix tree?
  - Using our same example
- What is our space complexity?



Suffix Tree

A tree, not a trie

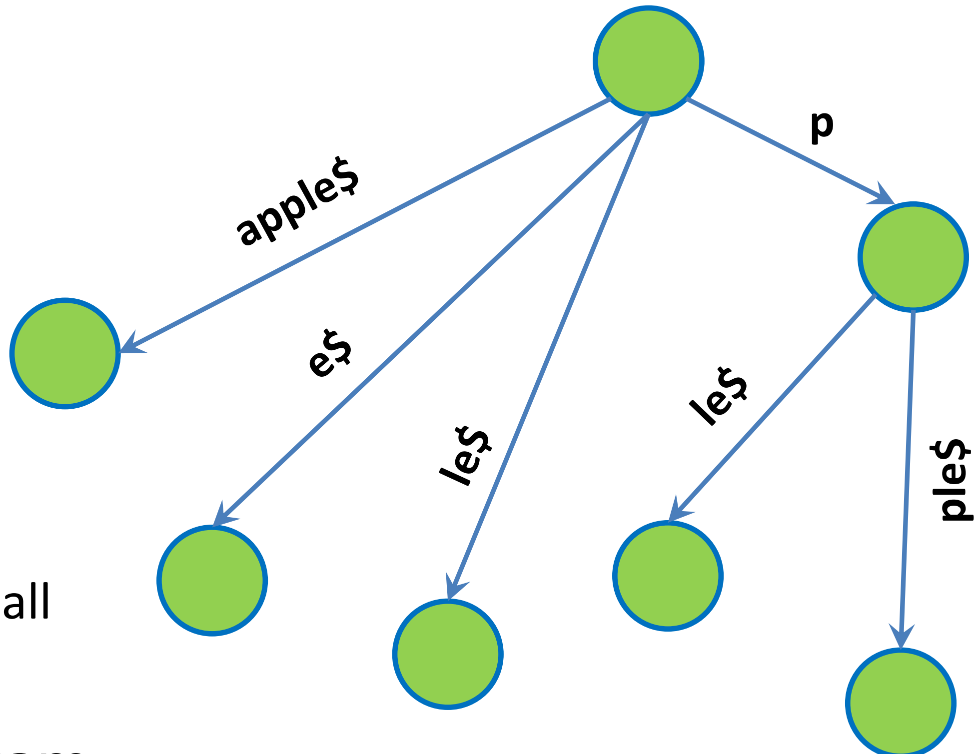
- What is a suffix tree?
  - Using our same example
- What is our space complexity?
  - O(N^2) still because we still store the characters all



Suffix Tree

A tree, not a trie

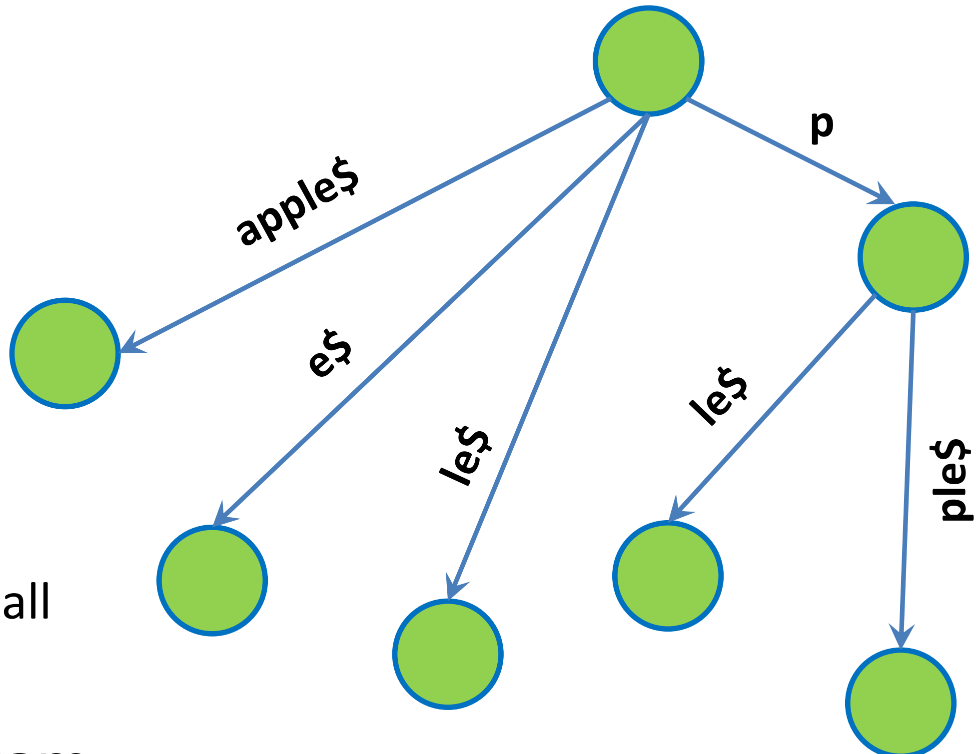
- What is a suffix tree?
  - Using our same example
- What is our space complexity?
  - O(N^2) still because we still store the characters all
- When asked in the exam…
  - Draw a suffix trie
  - Then compress to suffix tree



Suffix Tree

A tree, not a trie

- What is a suffix tree?
  - Using our same example
- What is our space complexity?
  - O(N^2) still because we still store the characters all
- When asked in the exam…
  - Draw a suffix trie
  - Then compress to suffix tree
- Note: Some like to separate out the $ node



Questions?

- Space complexity O(N^2)

Suffix Tree

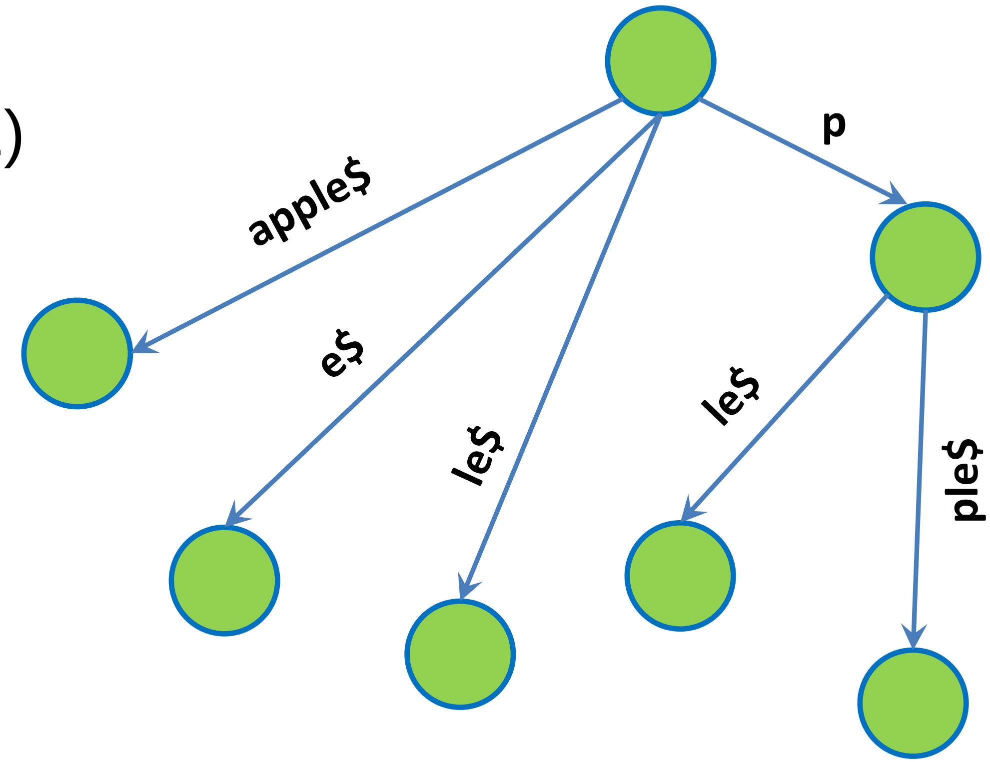
A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

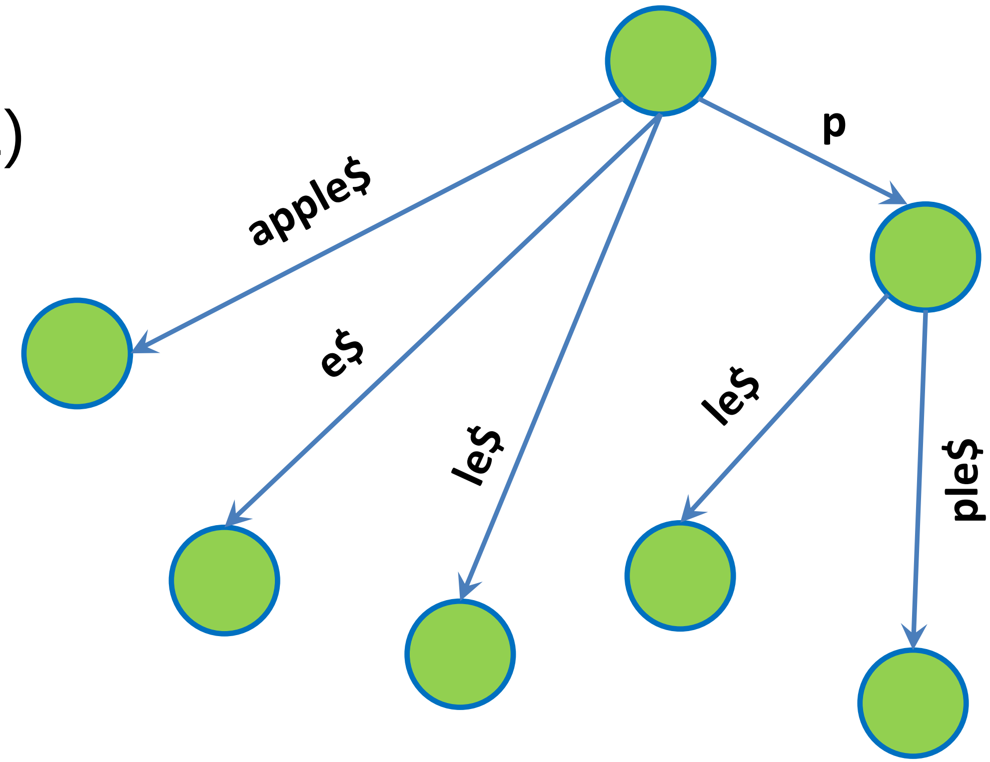


Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$

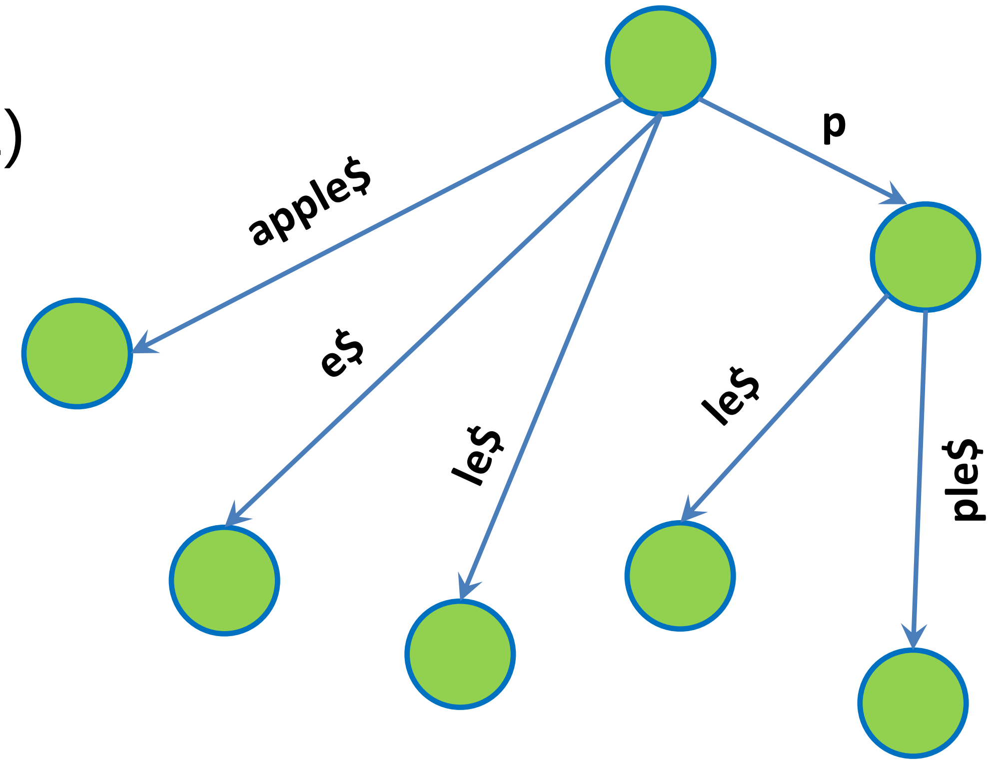


Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$



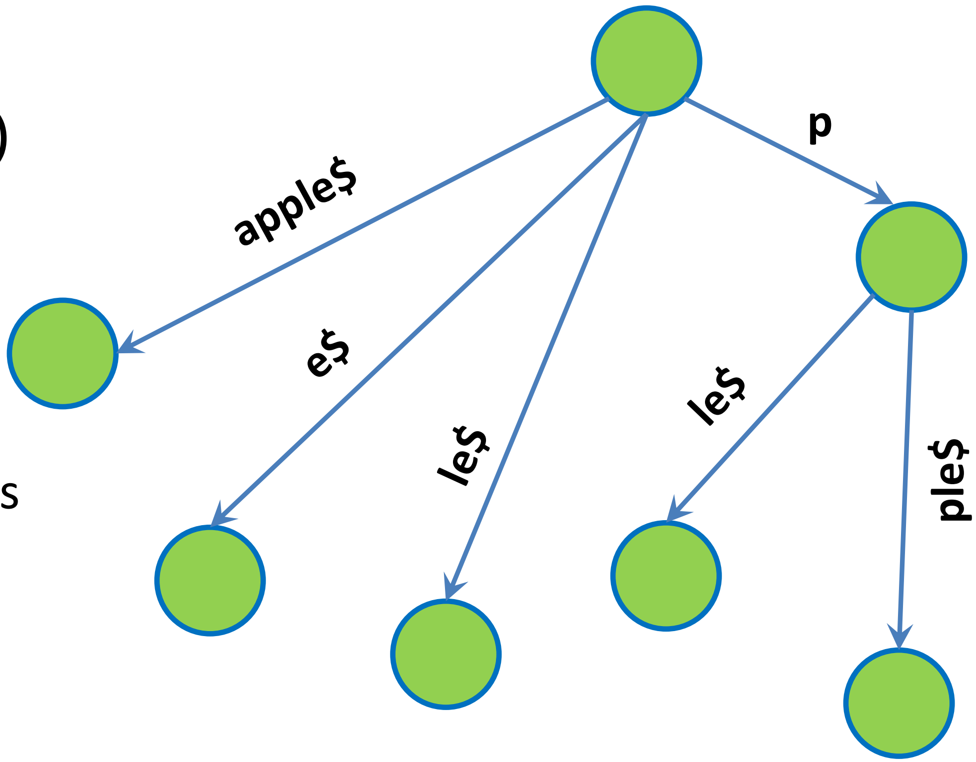
a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$
  – As our suffixes are continuous we can compress them!



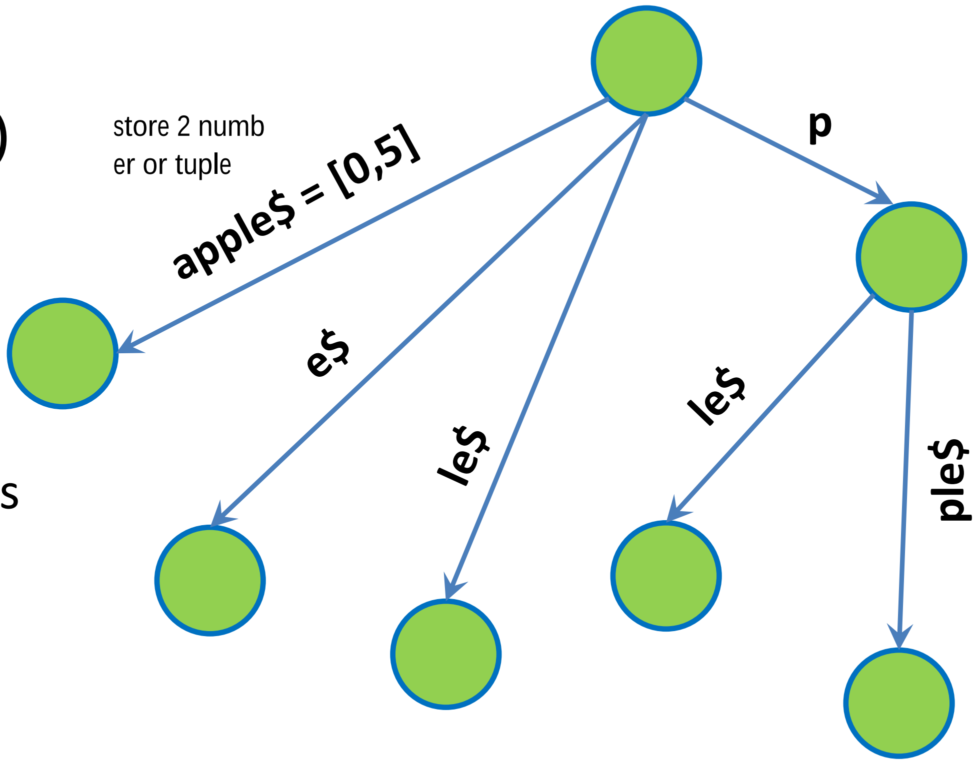
a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$
  - As our suffixes are continuous we can compress them!

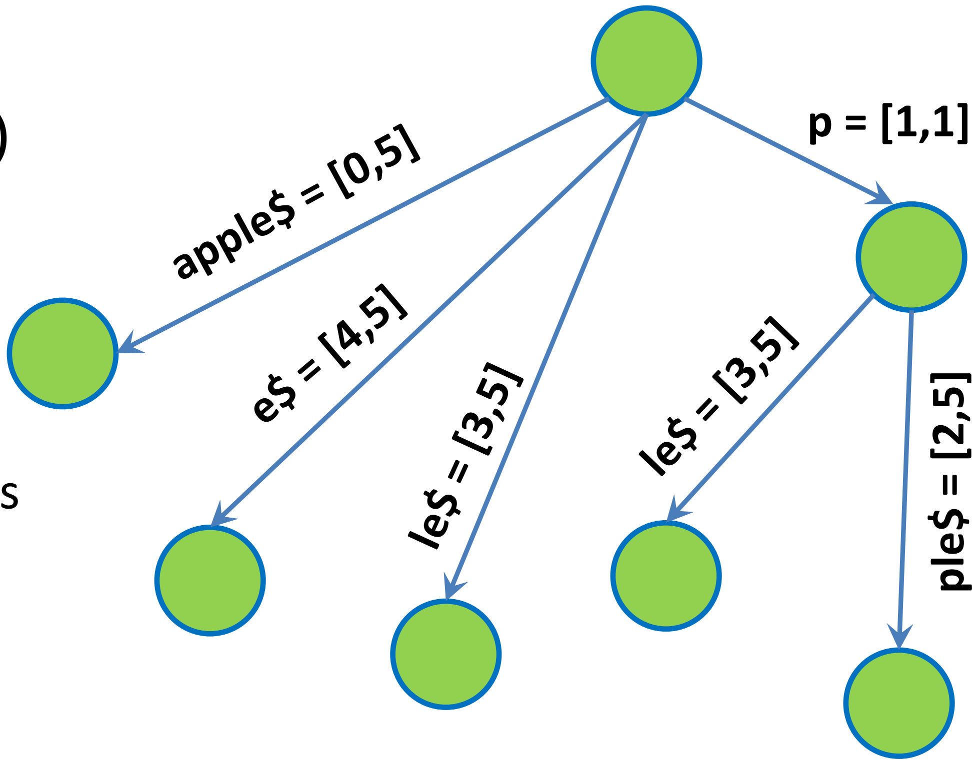


a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree
A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$
  - As our suffixes are continuous we can compress them!



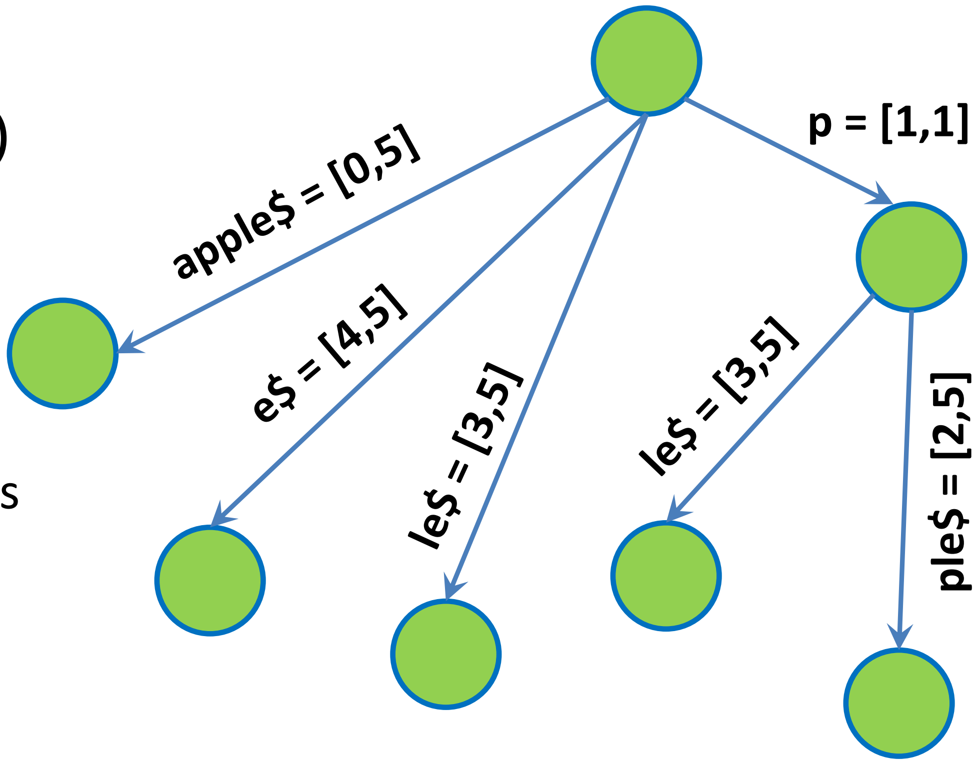
a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$
  - As our suffixes are continuous we can compress them!
  - So each we can just store [start, end]



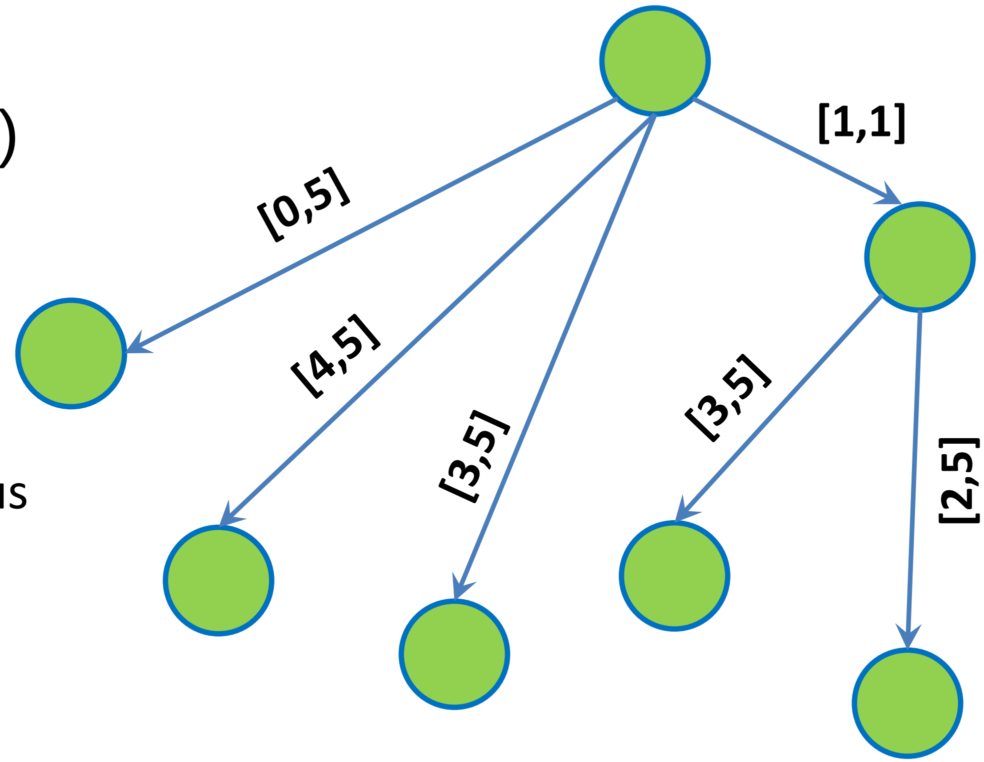
a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$
  - As our suffixes are continuous we can compress them!
  - So each we can just store [start, end]

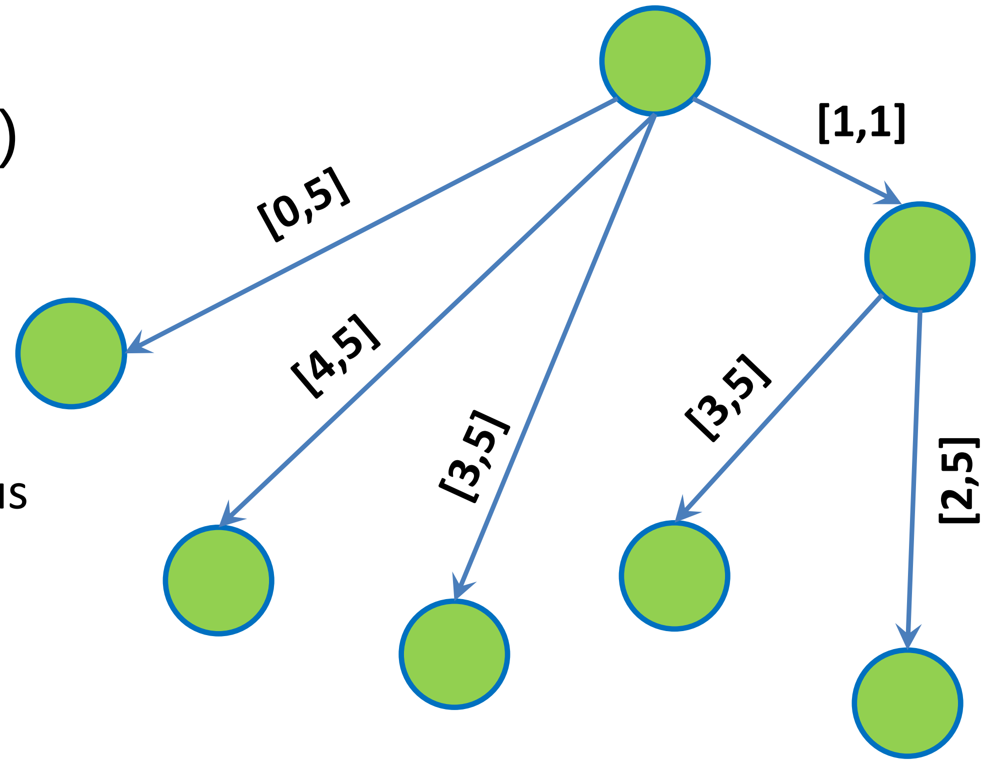


a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?
- Our string is apple$
  - As our suffixes are continuous we can compress them!
  - So each we can just store [start, end]
- Space complexity?



a	p	p	l	e	$
0	1	2	3	4	5

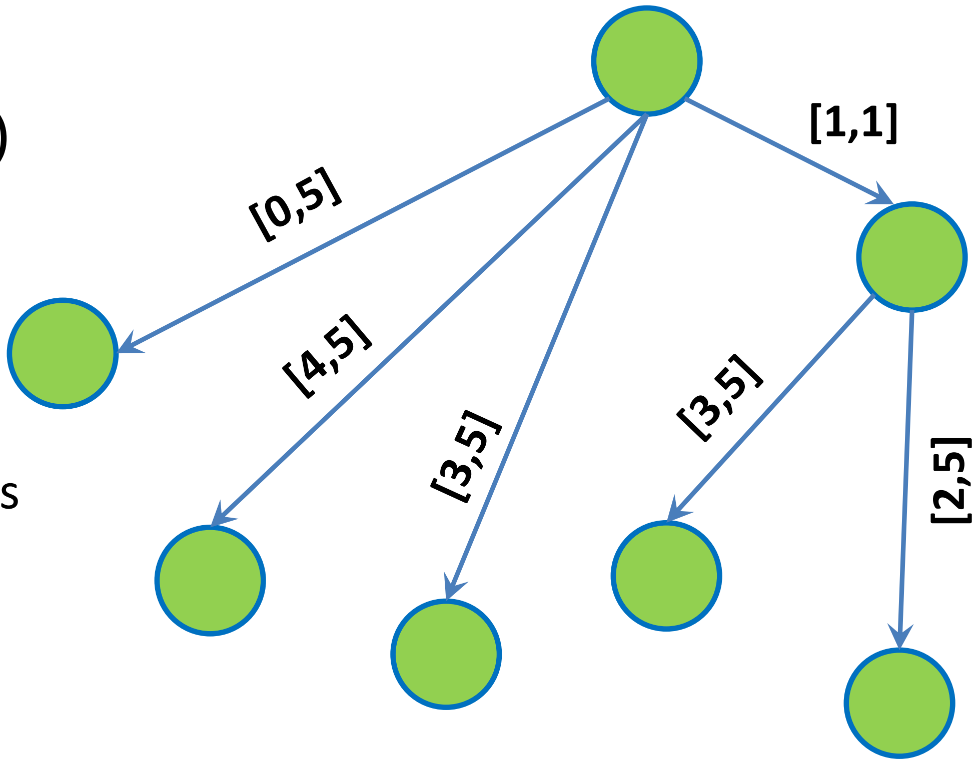
Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$
  - As our suffixes are continuous we can compress them!
  - So each we can just store [start, end]

- Space complexity?
  - O(N)



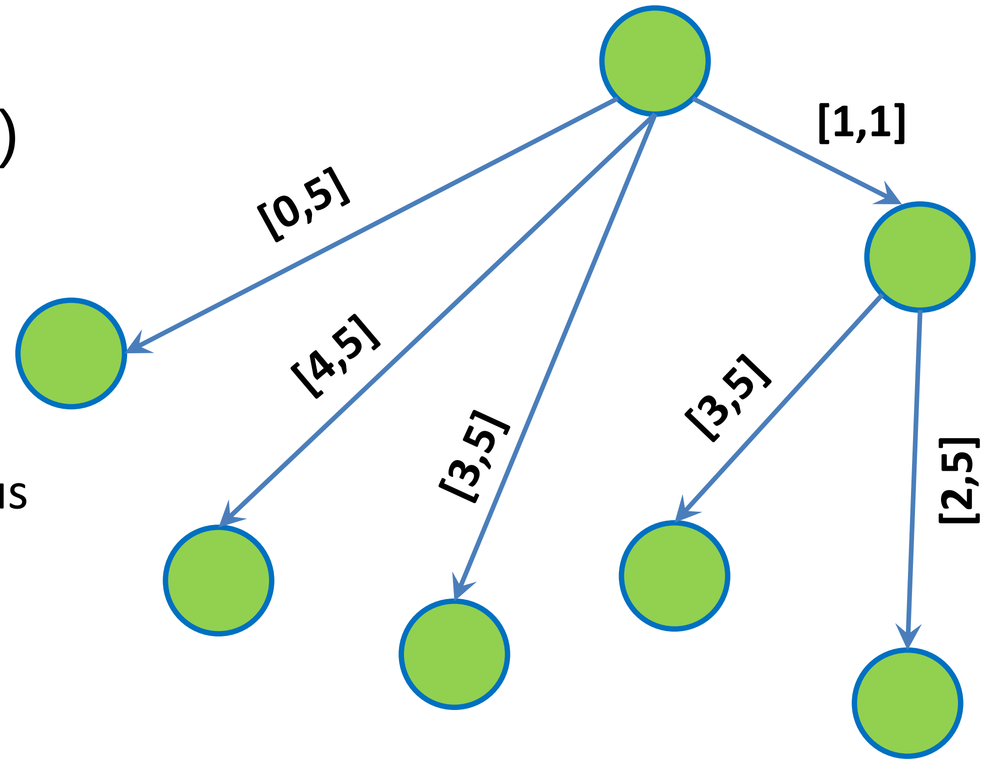
a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity O(N^2)
- Can we do better?

- Our string is apple$
  - As our suffixes are continuous we can compress them!
  - So each we can just store [start, end]



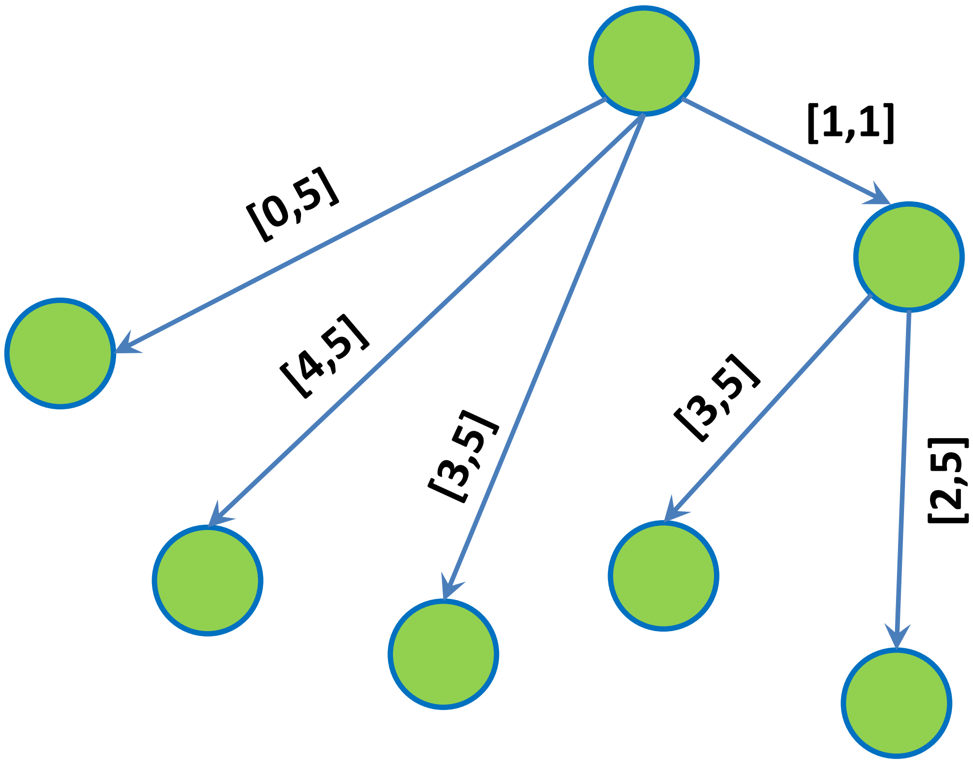
a	p	p	l	e	$
0	1	2	3	4	5

- Space complexity?
  - O(N)
  - N leaves
  - Each non-leaf node has at least 2 children

Suffix Tree

A tree, not a trie

- Space complexity?
  - O(N)
  - N leaves
  - Each non-leaf node has at least 2 children
  - Total number of node = O(N + N/2 + N/4 + …) = O(N)

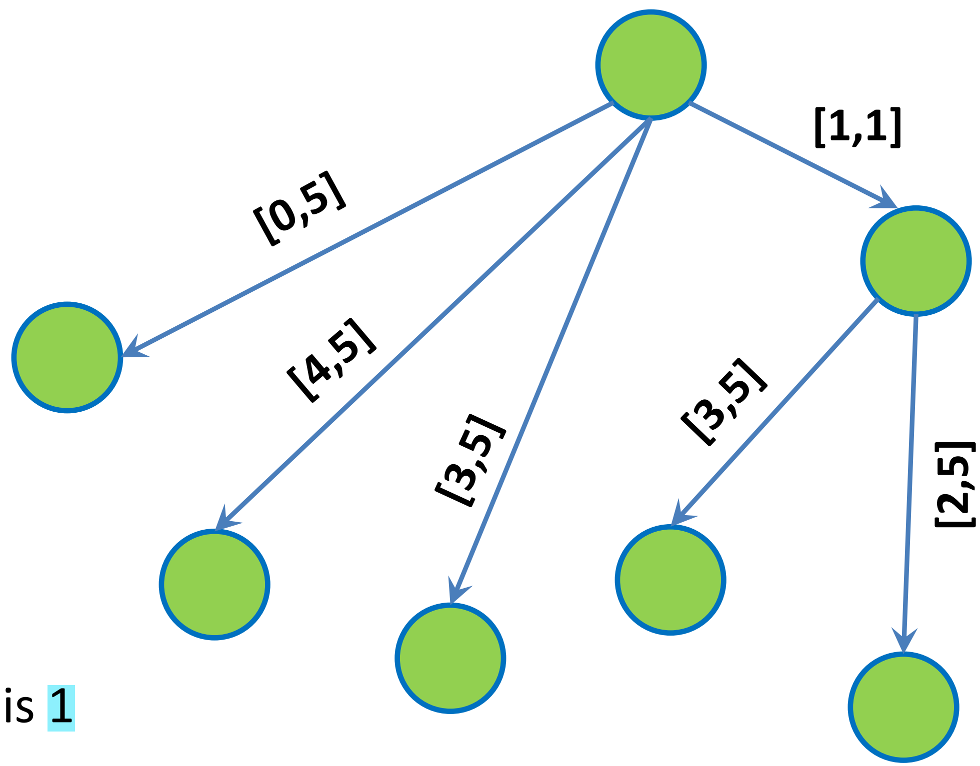


a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity?
  - O(N)
  - N leaves
  - Each non-leaf node has at least 2 children
  - Total number of node = O(N + N/2 + N/4 + …) = O(N)
    * cause we go till root which is 1 from leaves



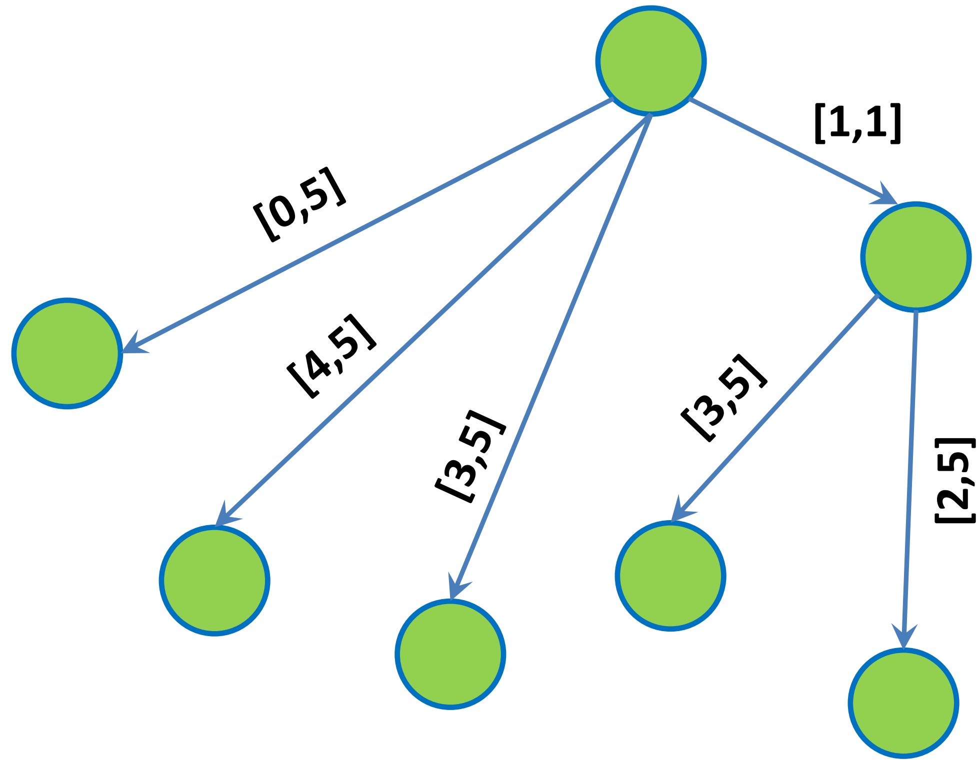
a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity?
  - O(N)
  - N leaves
  - Each non-leaf node has at least 2 children
  - Total number of node = O(N + N/2 + N/4 + …) = O(N)

- Time complexity remains O(N^2) as we still need to insert every suffix with N character max



[i,j] or (i,j)

a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

- Space complexity?
  - O(N)
  - N leaves
  - Each non-leaf node has at least 2 children
  - Total number of node
    = O(N + N/2 + N/4 + …)
    = O(N)

- Time complexity remains O(N^2) as we still need to insert every suffix with N character max

We learn the hax called Ukkonen’s algorithm (1995) in FIT3155 to do in O(N)

Questions?

- Let us try to implement it!

- Let us try to implement it!

- As a class activity
- … and some of the same functions

- Let us try to implement it!

- As a class activity
- … and some of the same functions
  - Better than you searching online and not understanding what is happening

- Let us try to implement it!

- As a class activity
- … and some of the same functions
  - Better than you searching online and not understanding what is happening
  - But 2 implementation
    - Iterative
    - Recursive (efficient)

Questions?

MONASH University

Thank You