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

MONASH University

Ready?

Agenda

- String retrieval

Agenda

- String retrieval
- Tries and suffix tries

MONASH University

Let us begin…

MONASH University

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

MONASH University

- String retrieval is one of the oldest retrieval task in the world…
- Anything can be represented as a string

MONASH University

- 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!

MONASH University

- So how can we search for string very fast?

MONASH University

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

MONASH University

- 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)

MONASH University

- 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)

MONASH University

- 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)

MONASH University

Questions?

Tries

Efficient string retrieval

MONASH University

- 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…

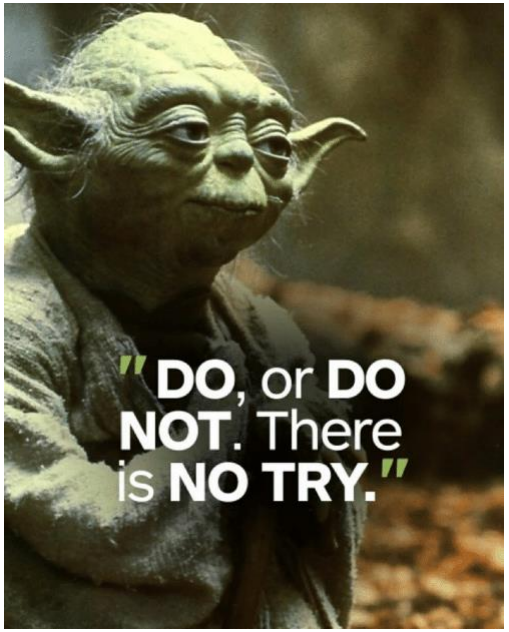
Tries

Efficient string retrieval

MONASH University

- 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



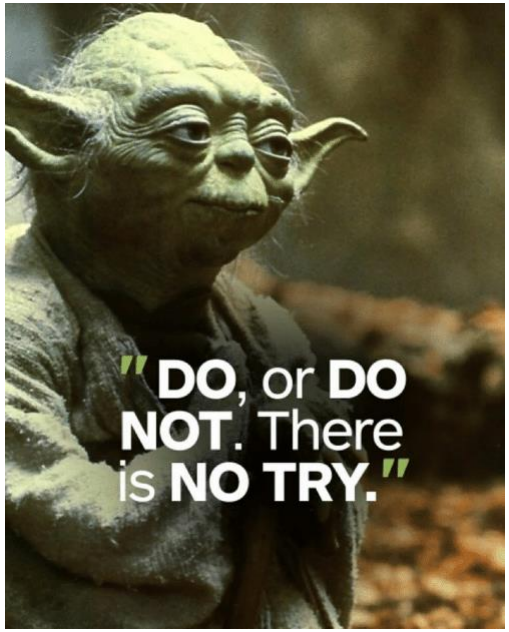
Tries

Efficient string retrieval

MONASH University

- 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



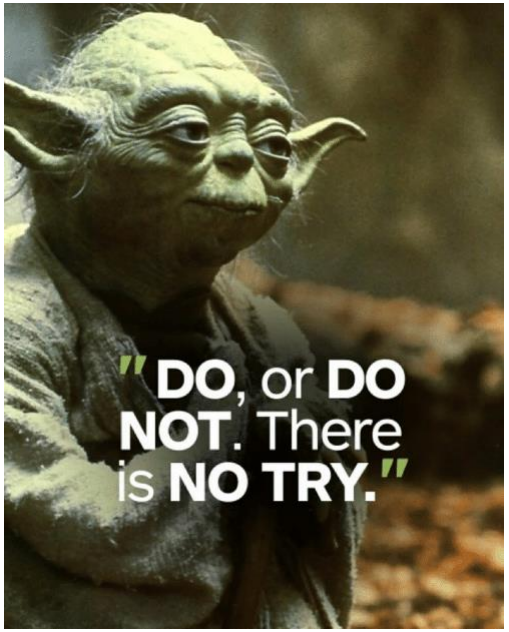
Tries

Efficient string retrieval

MONASH University

- 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
    M = number of unique character



Tries

Efficient string retrieval

MONASH University

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

Tries

Efficient string retrieval

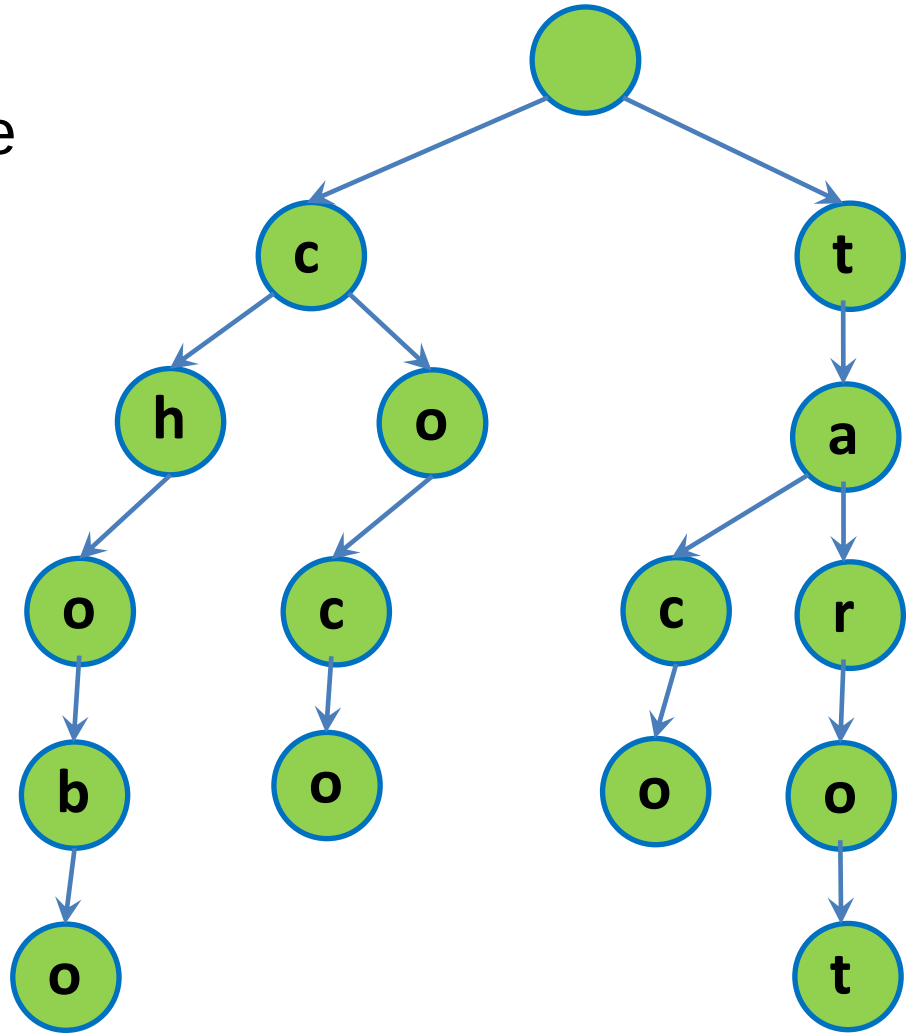
MONASH University

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

Tries

Efficient string retrieval

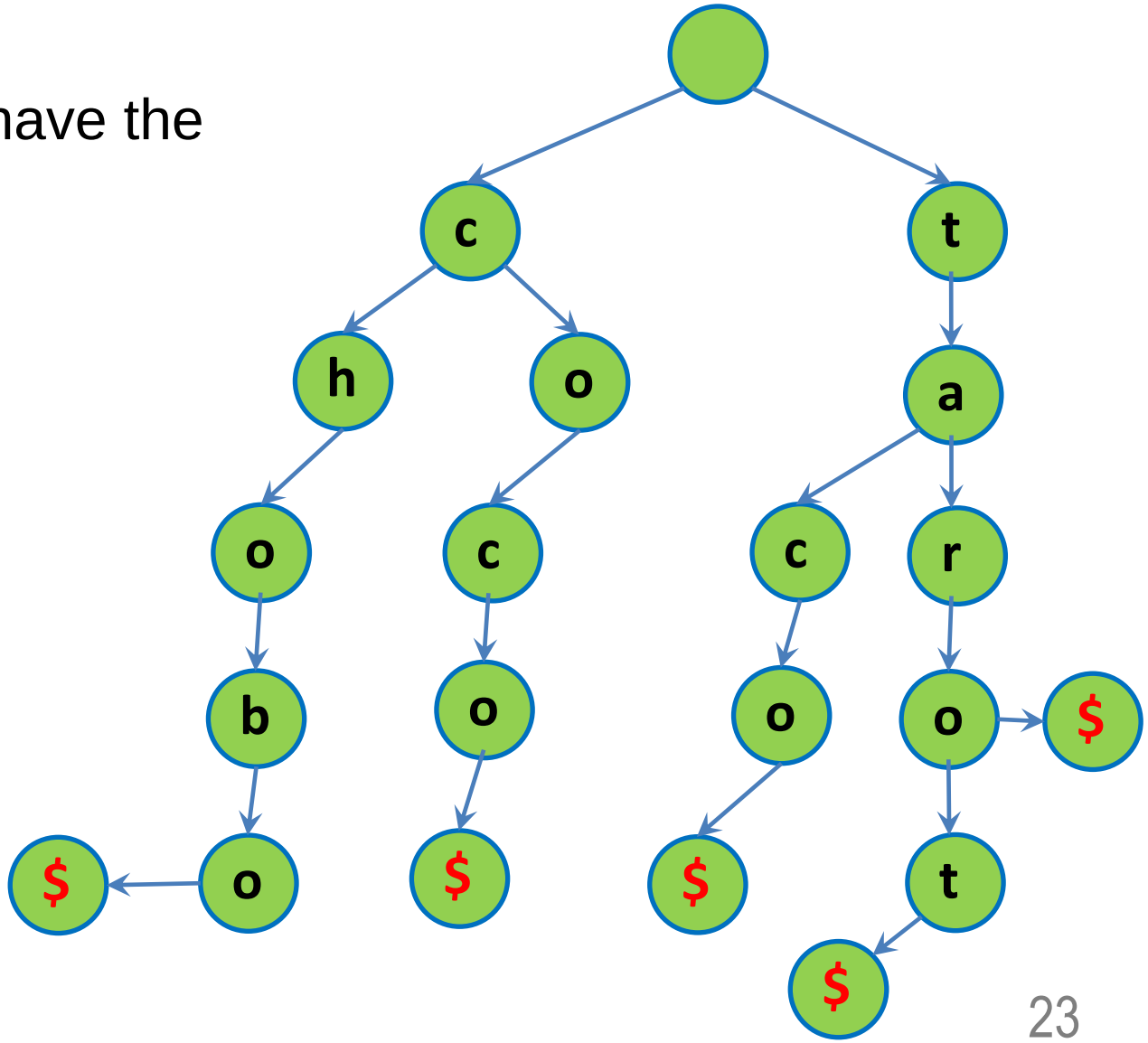
- Let assume we have the following words:
  - Taco
  - Taro
  - Tarot
  - Coco
  - Chobo
  - What is the trie?



Tries

Efficient string retrieval

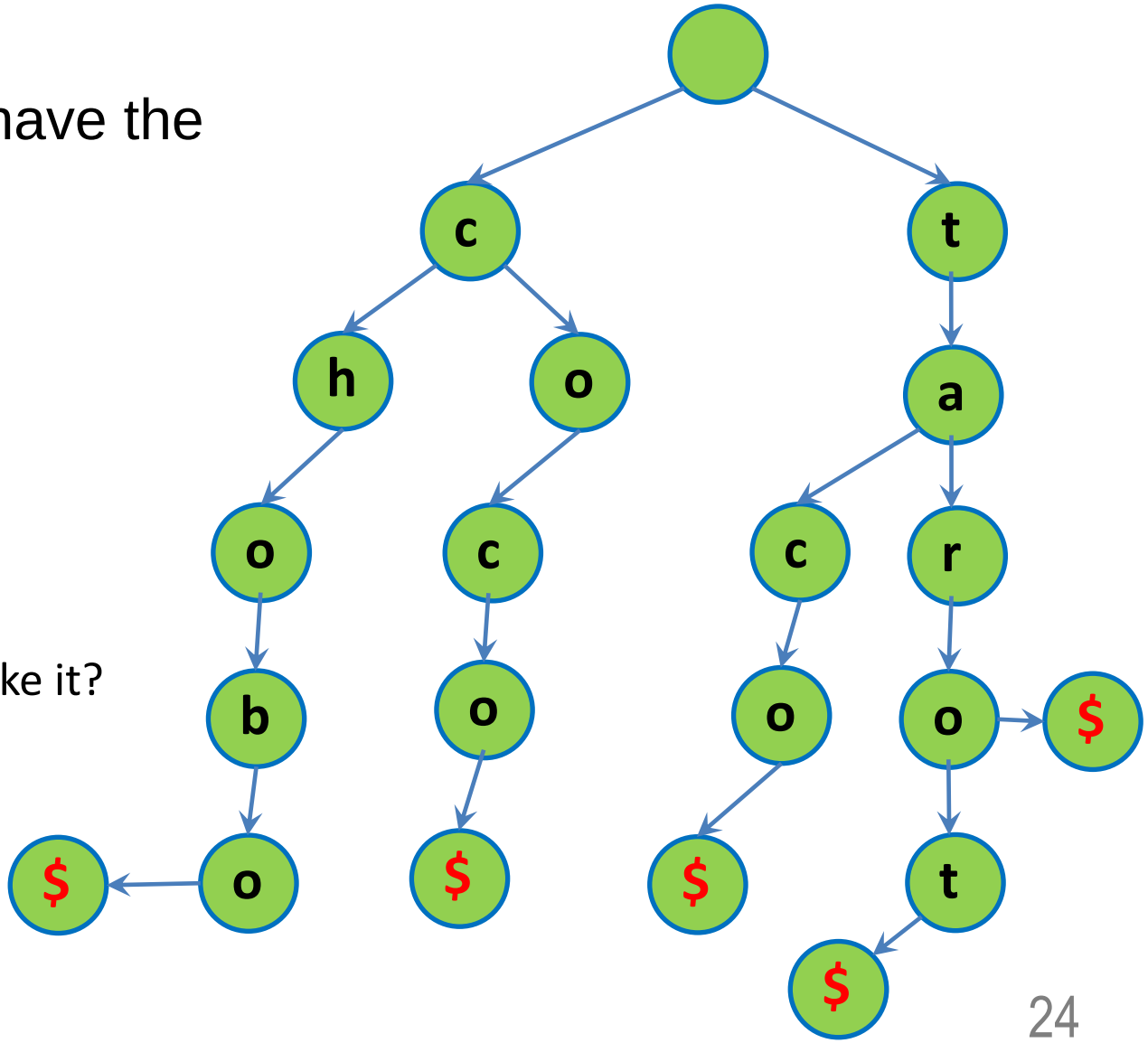
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?



Tries

Efficient string retrieval

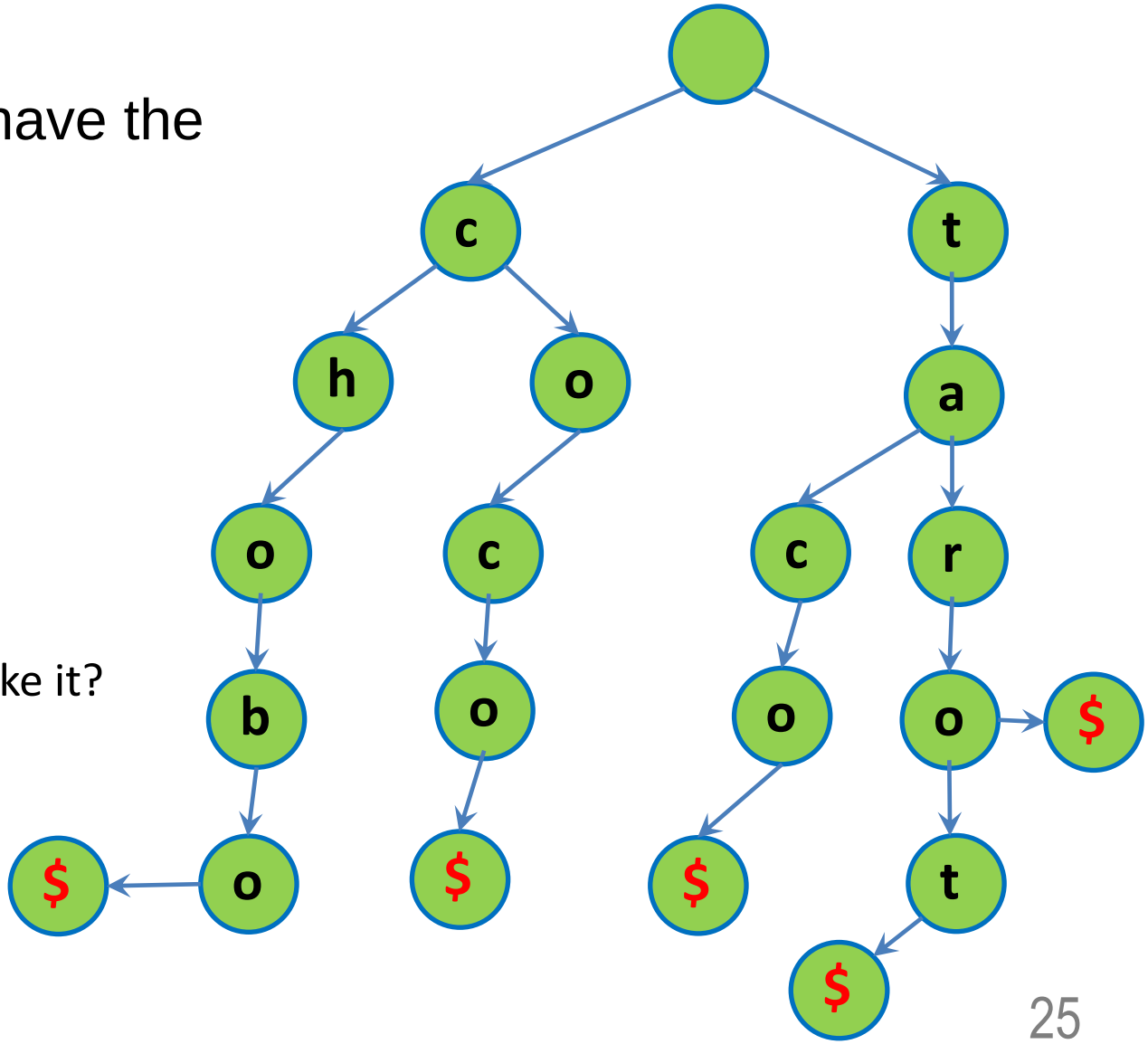
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?



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…

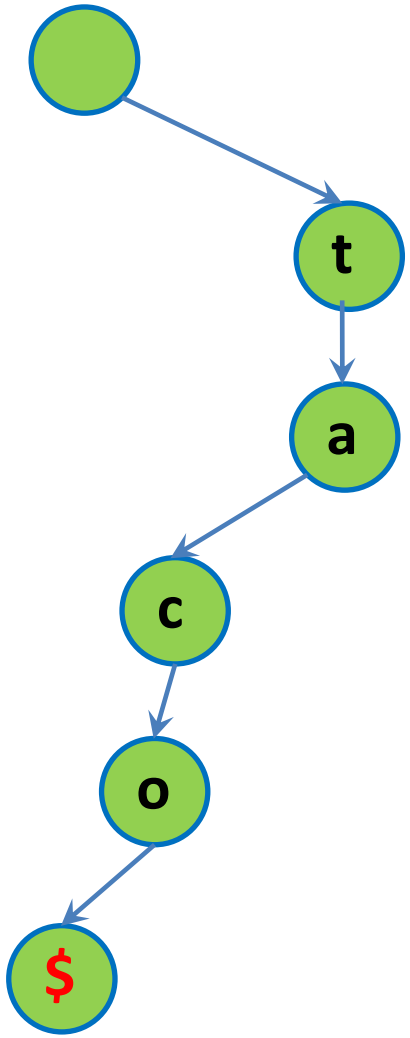


Tries

Efficient string retrieval

MONASH University

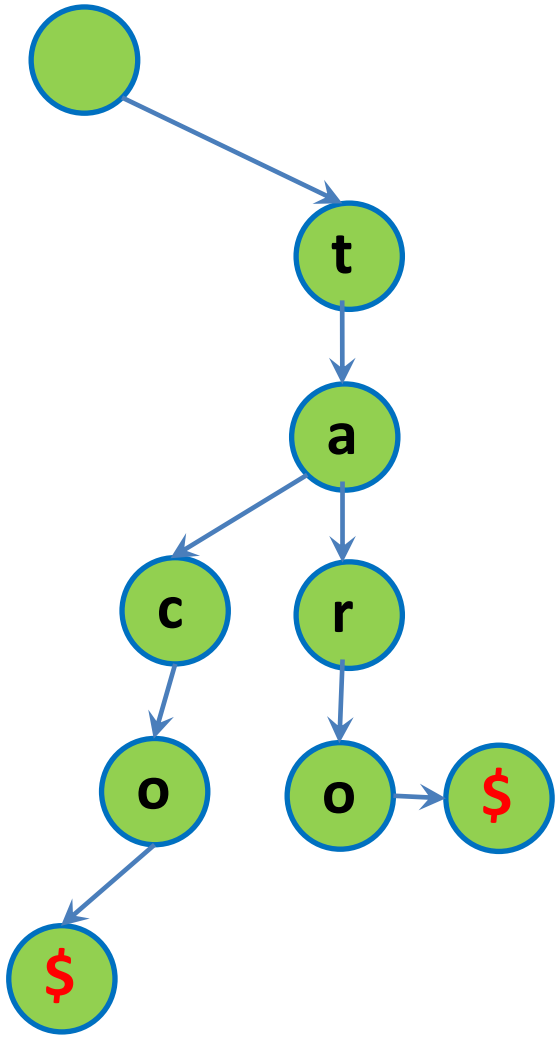
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…



Tries

Efficient string retrieval

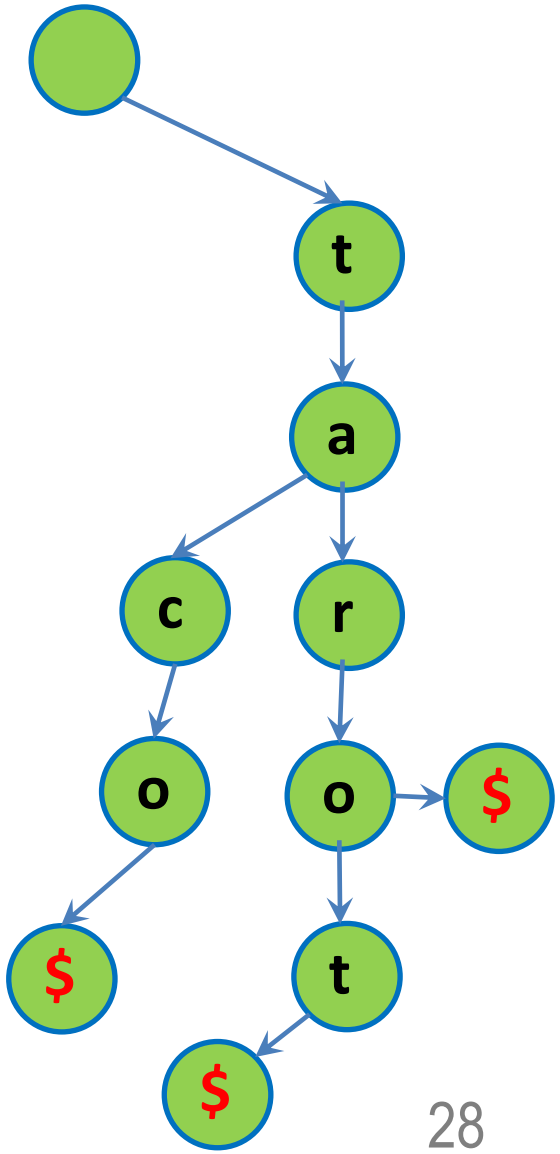
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…



Tries

Efficient string retrieval

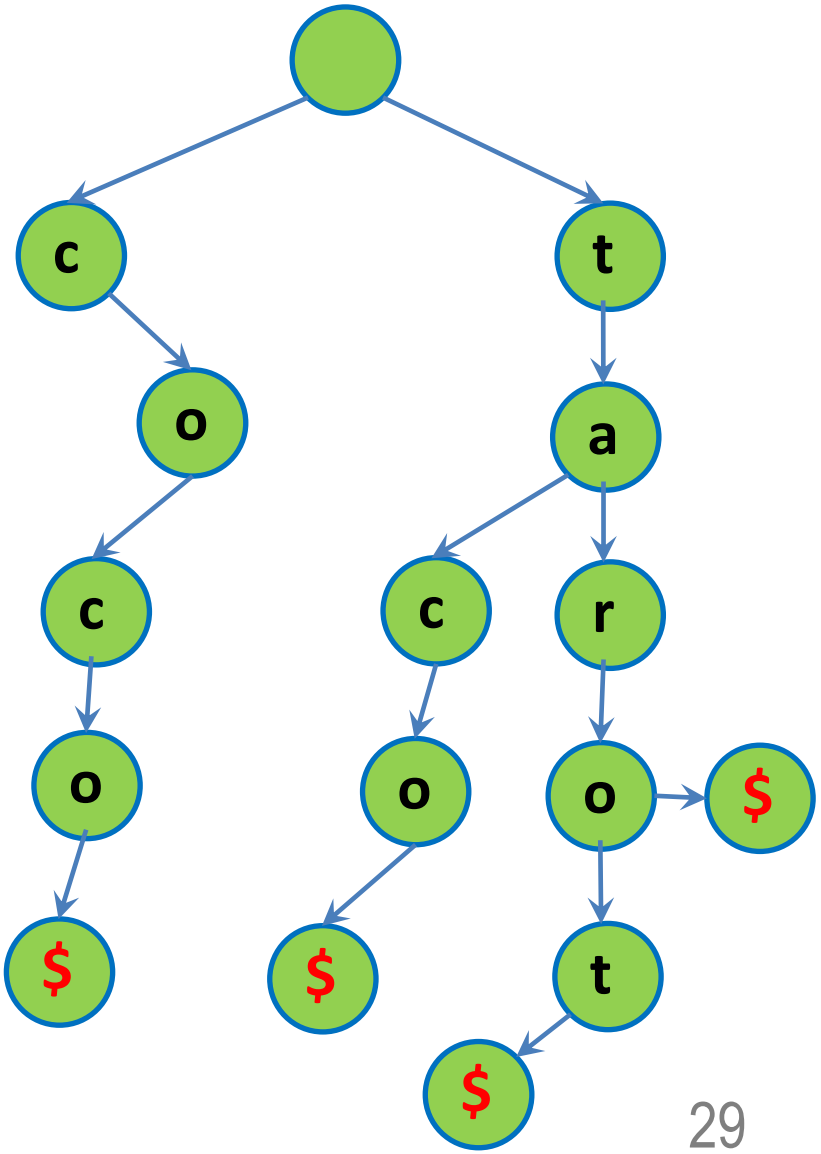
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…



Tries

Efficient string retrieval

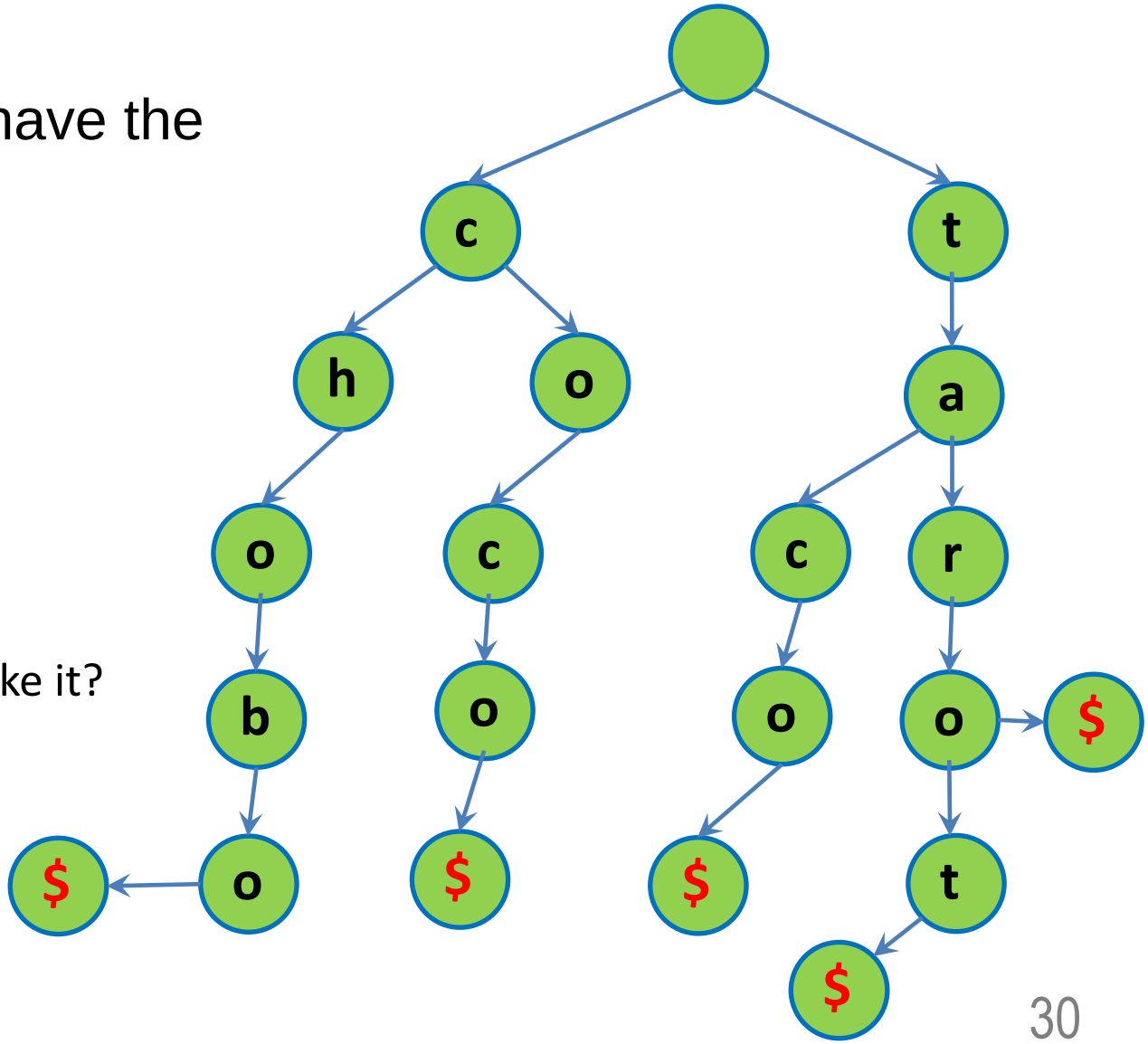
- Let assume we have the following words:
  - Taco$
  - Taro$
  - Tarot$
  - Coco$
  - Chobo$
  - What is the trie?
  - So how do we make it?
    - Step by step…



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…

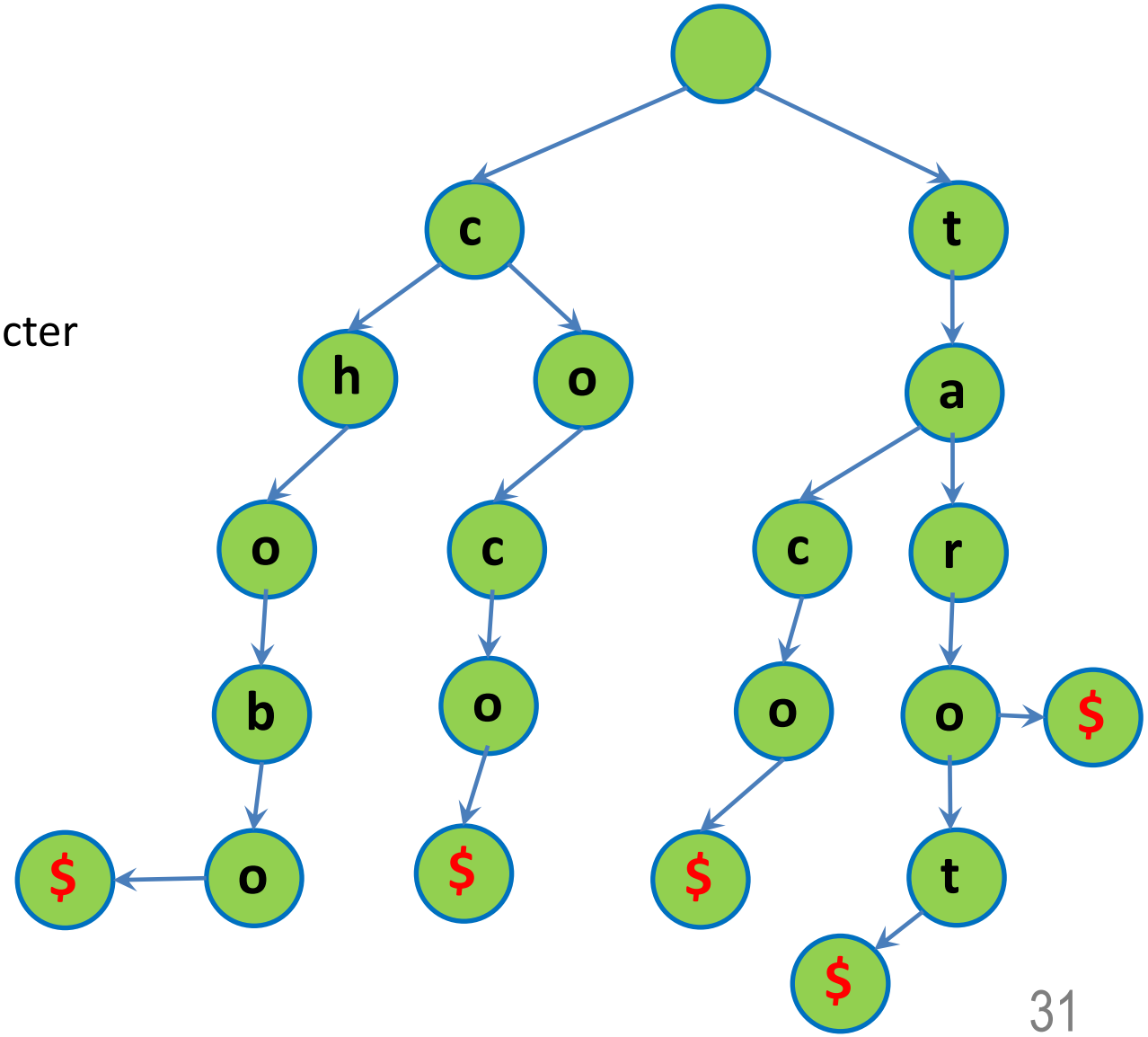


Tries

Efficient string retrieval

MONASH University

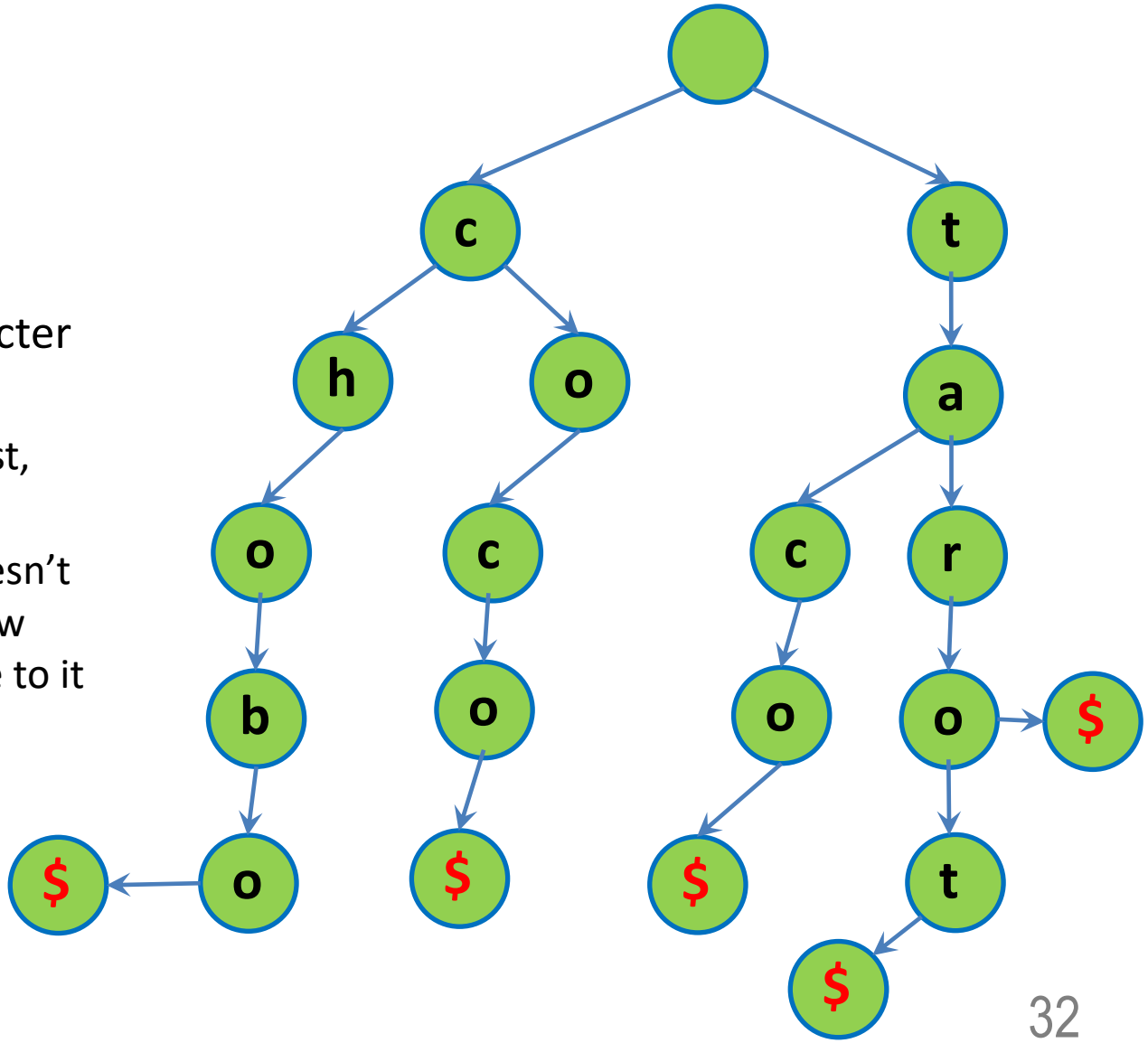
- So steps?
  - For each word
  - Start from root
  - Go through character by character



Tries

Efficient string retrieval

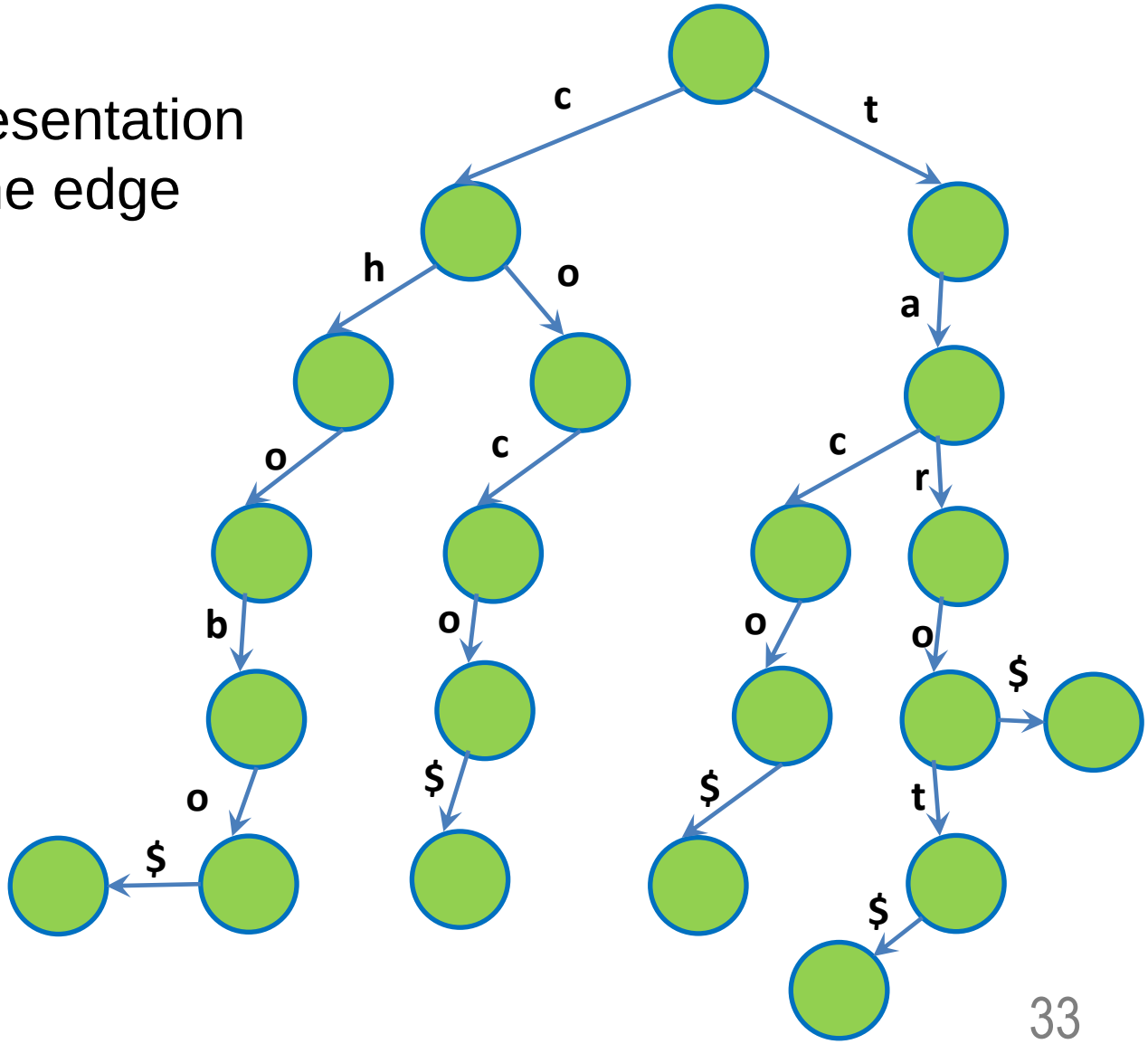
- So steps?
  - For each word
  - Start from root
  - Go through character by character
    - If character exist, follow through
    - If character doesn't exist, create new node and move to it



Tries

Efficient string retrieval

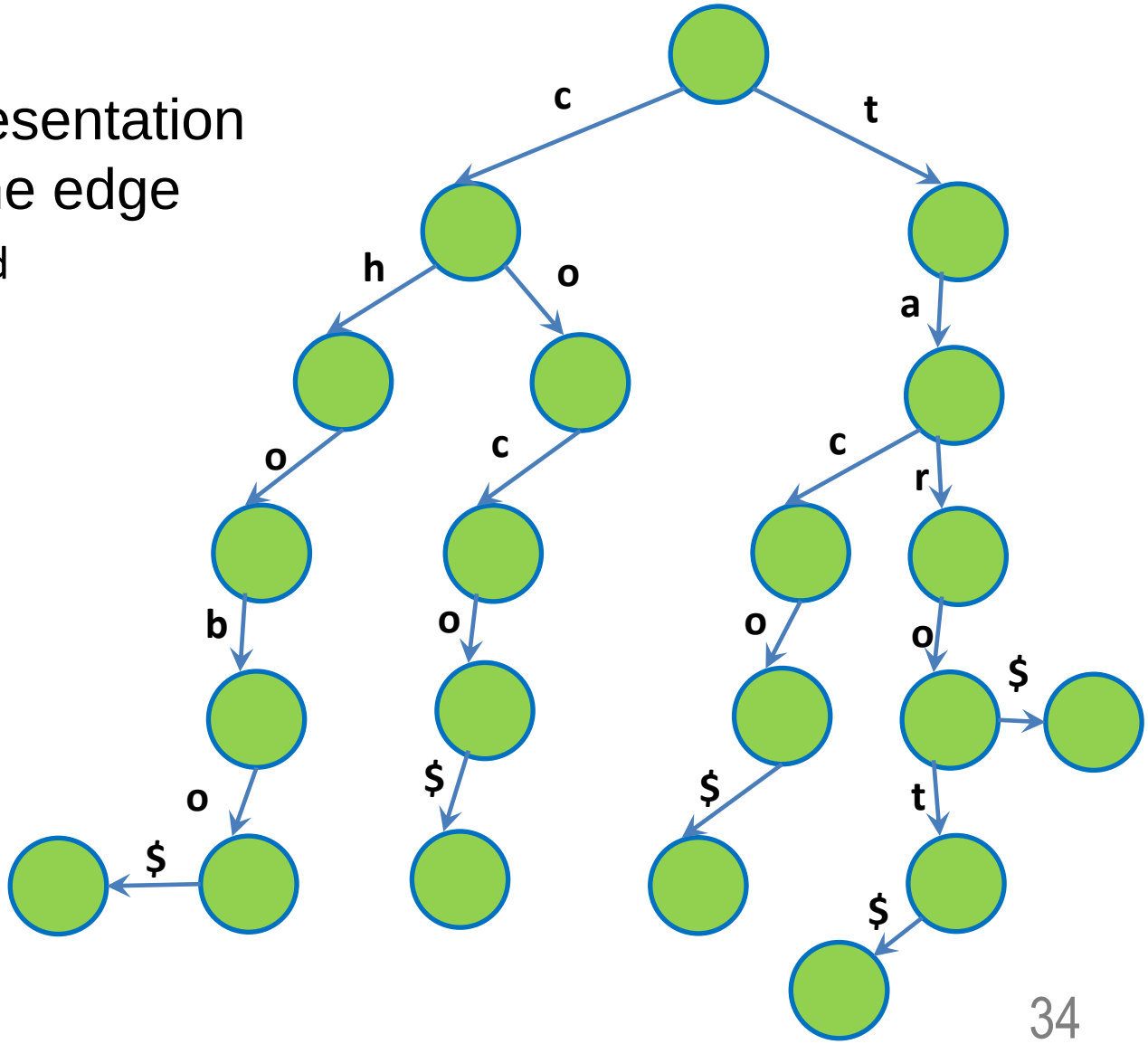
- The proper representation is character at the edge



Tries

Efficient string retrieval

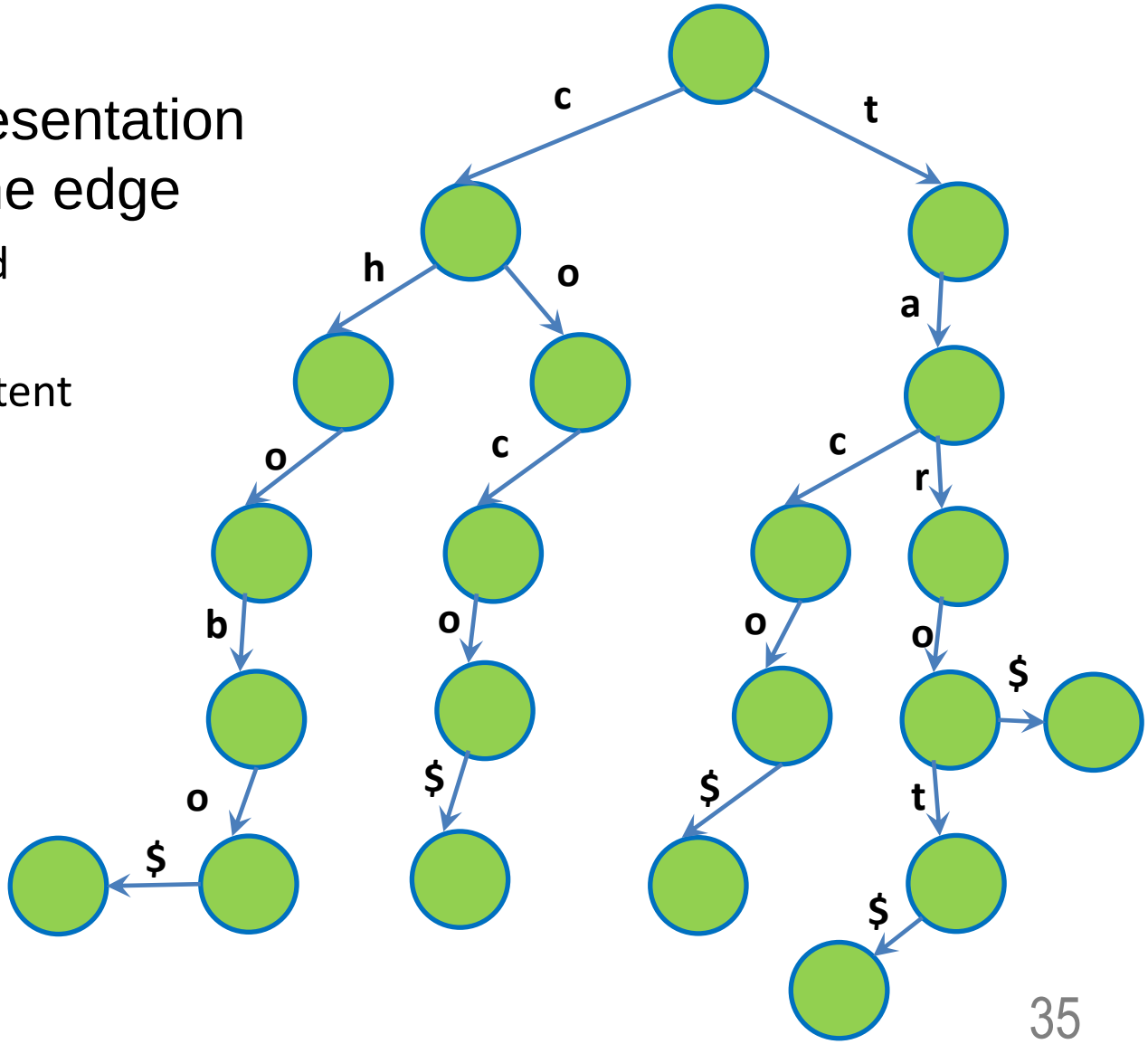
- The proper representation is character at the edge
  - Both are accepted for your exam!



Tries

Efficient string retrieval

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



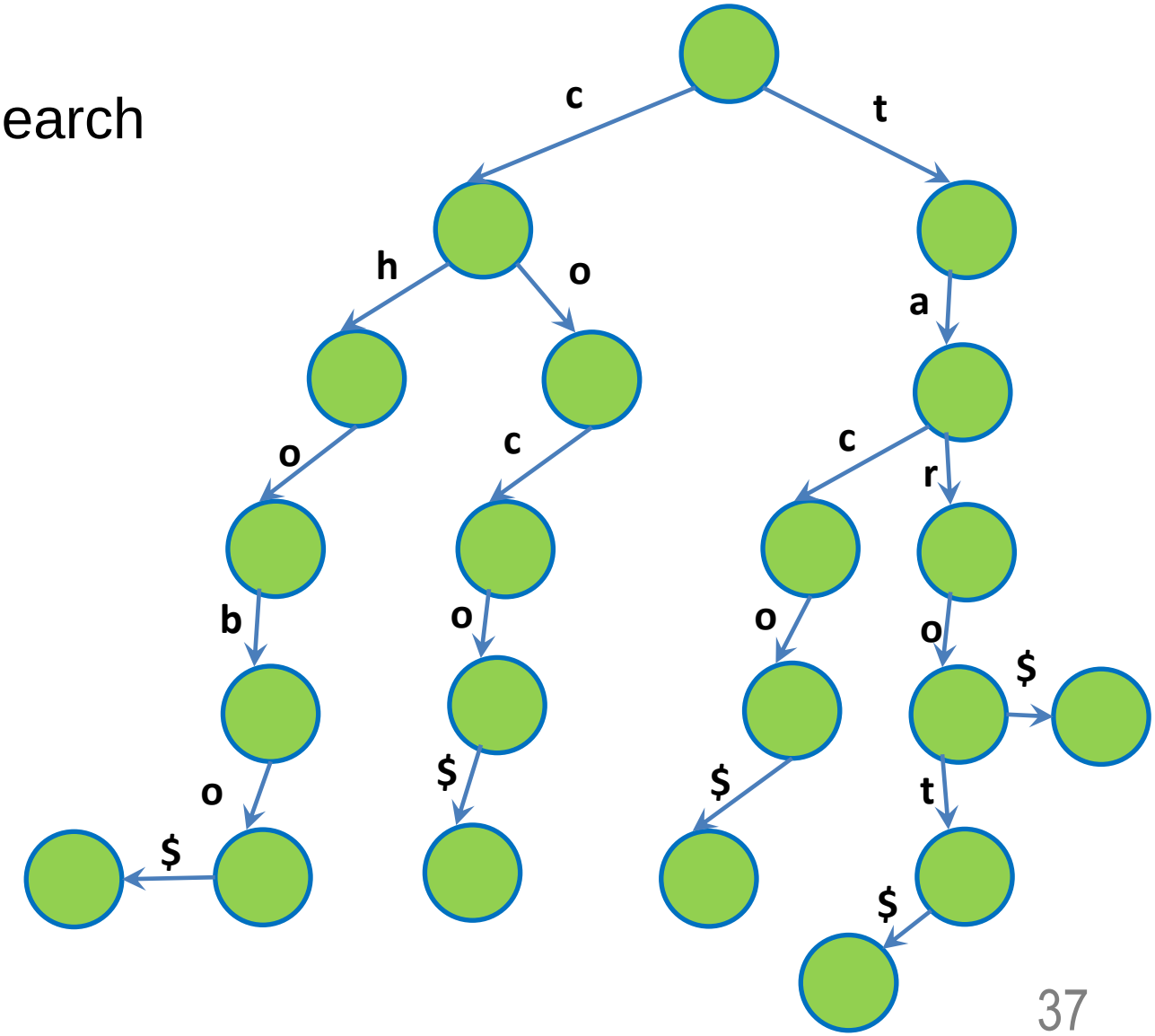
MONASH University

Questions?

Tries

Efficient string retrieval

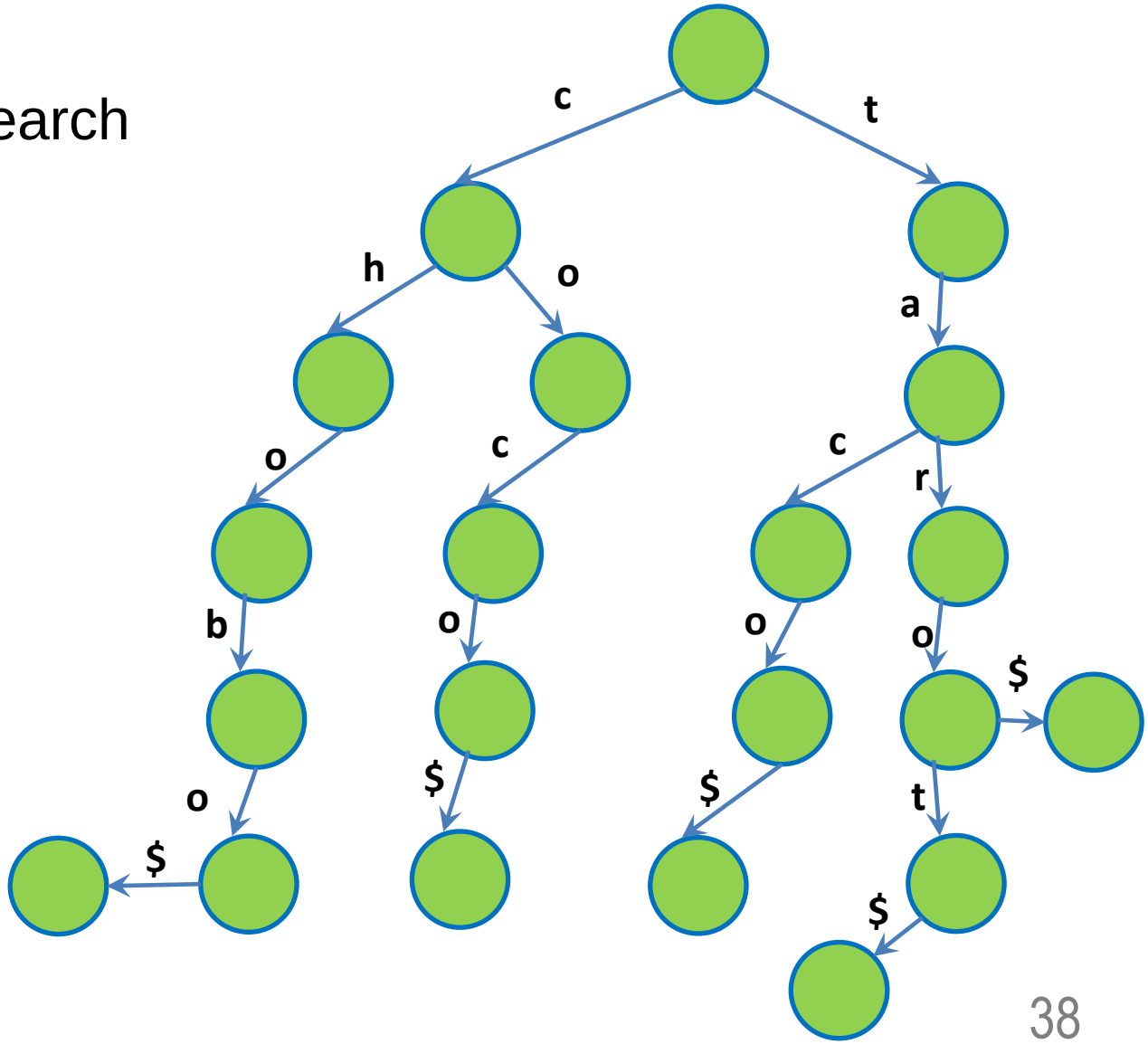
- So how do we search for retrieval?



Tries

Efficient string retrieval

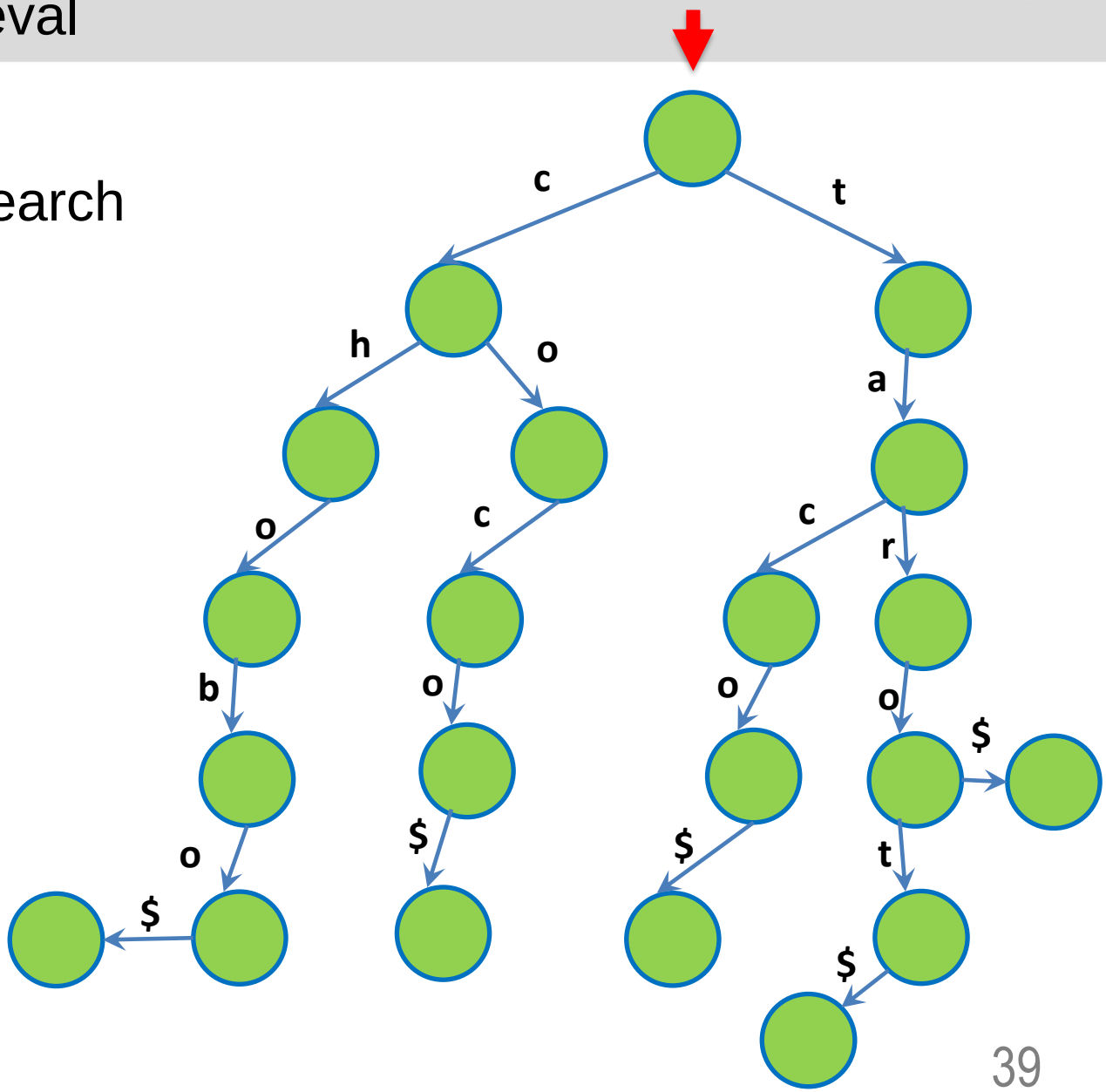
- So how do we search for retrieval?
  - Search for “coco”



Tries

Efficient string retrieval

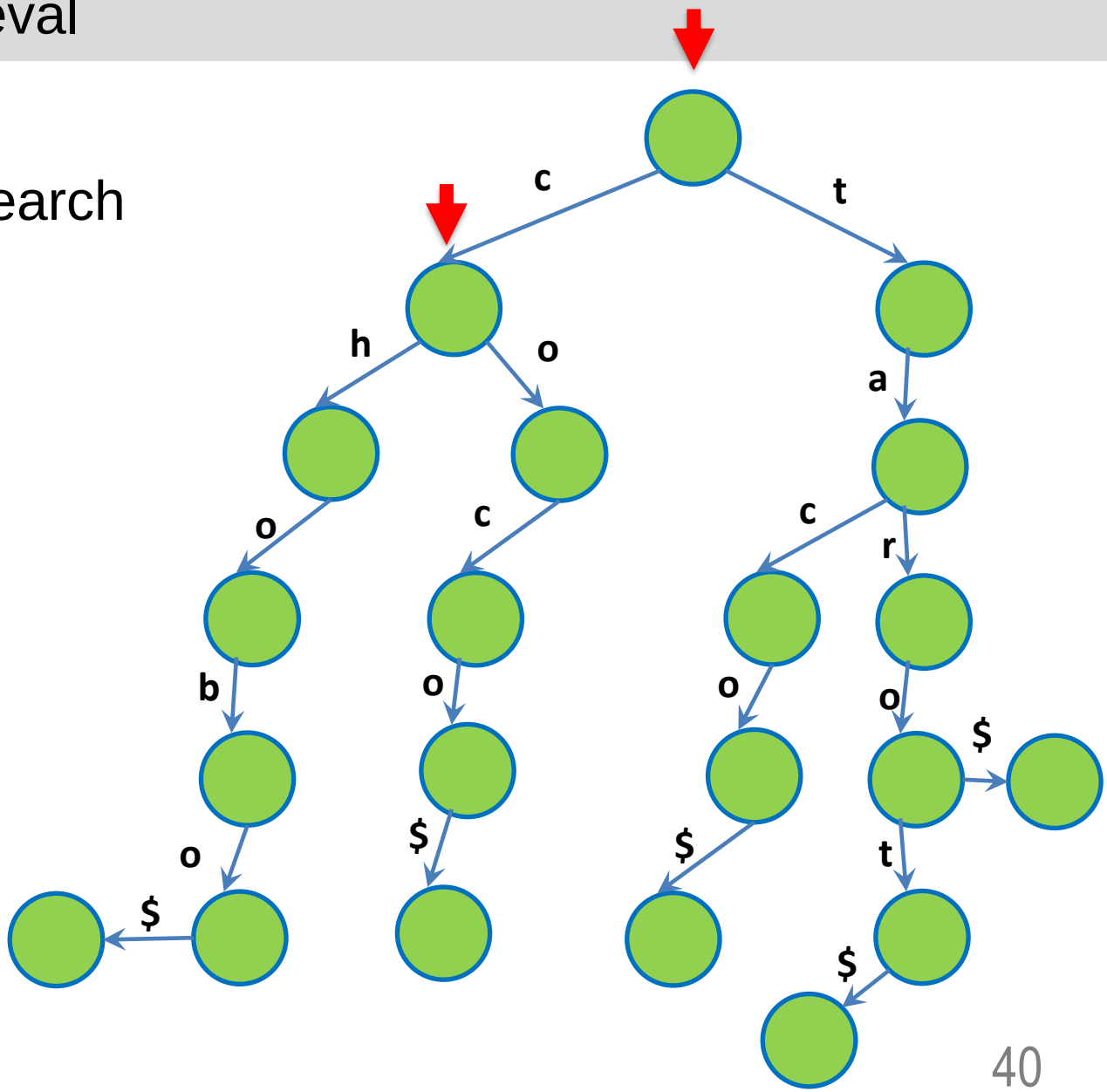
- So how do we search for retrieval?
  - Search for “coco”



Tries

Efficient string retrieval

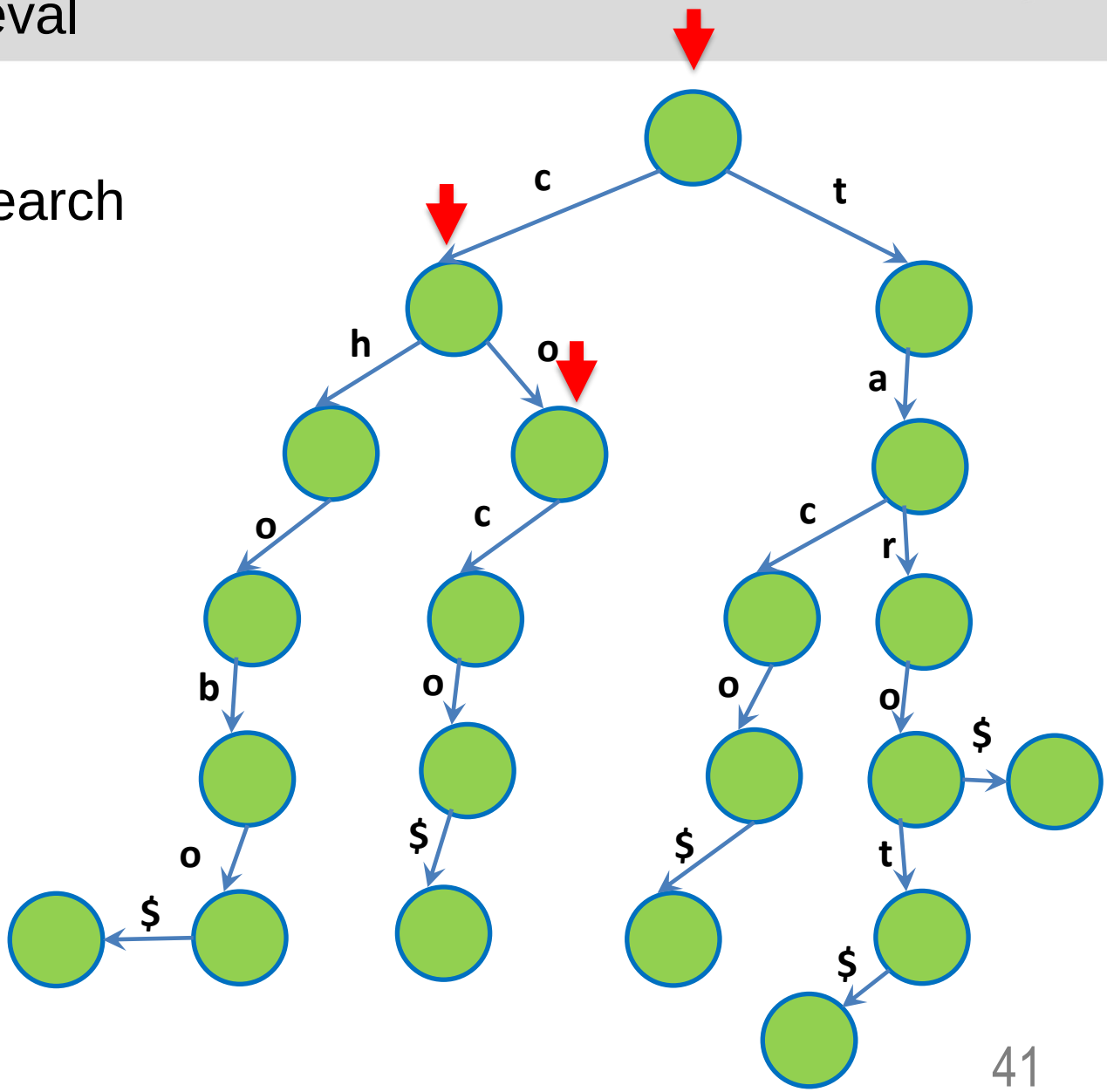
- So how do we search for retrieval?
  - Search for “coco”



Tries

Efficient string retrieval

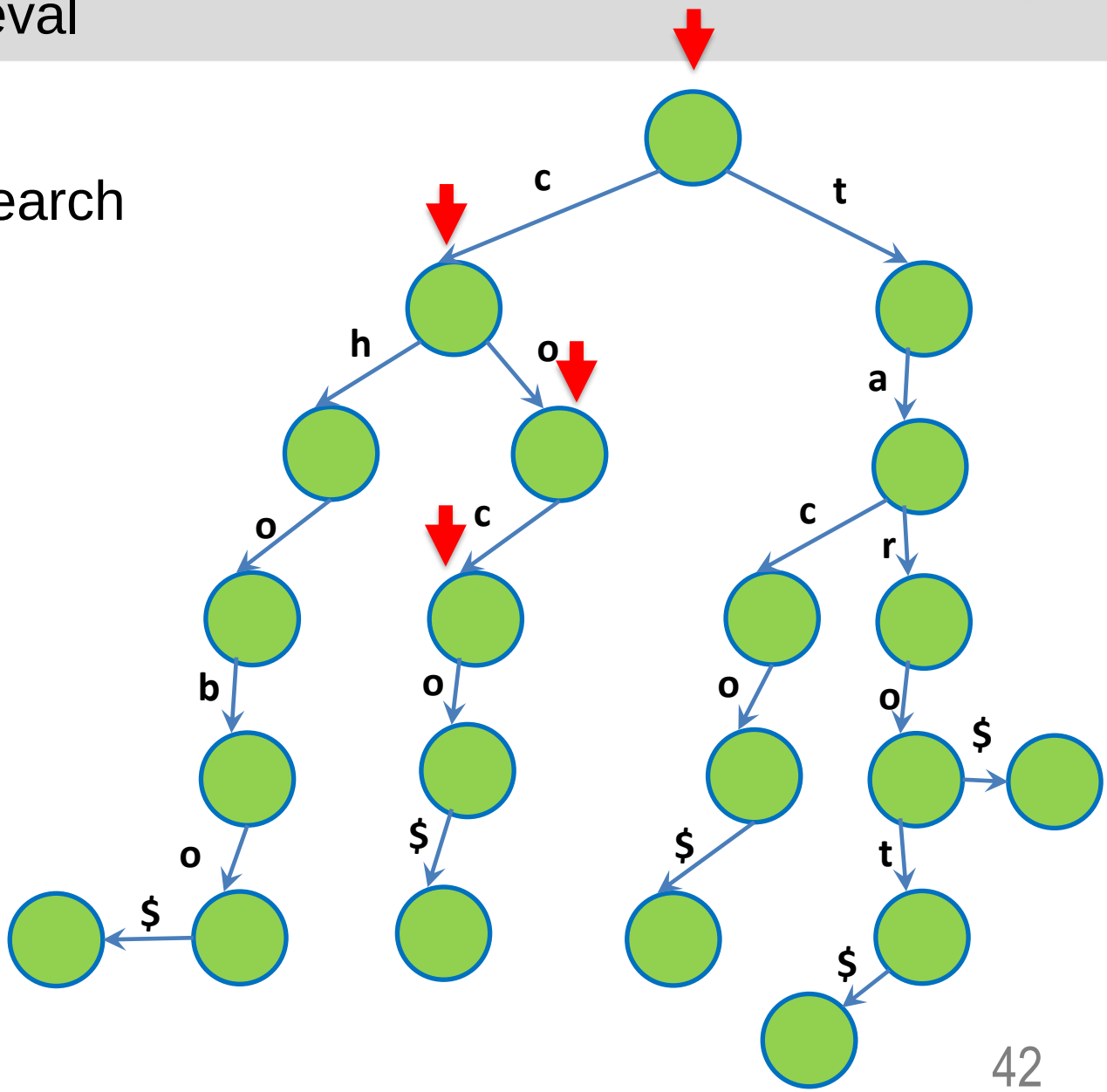
- So how do we search for retrieval?
  - Search for “coco”



Tries

Efficient string retrieval

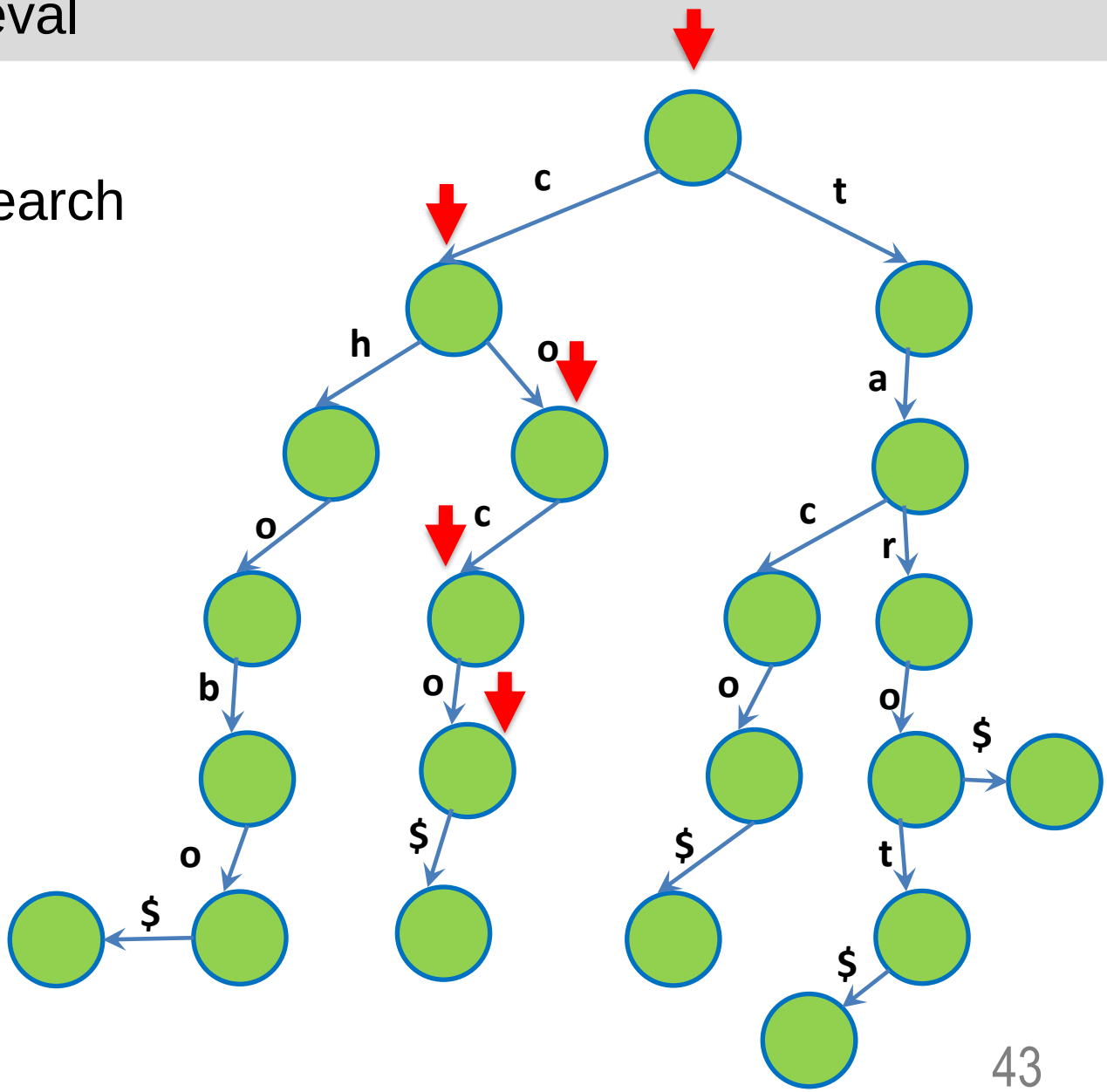
- So how do we search for retrieval?
  - Search for “coco”



Tries

Efficient string retrieval

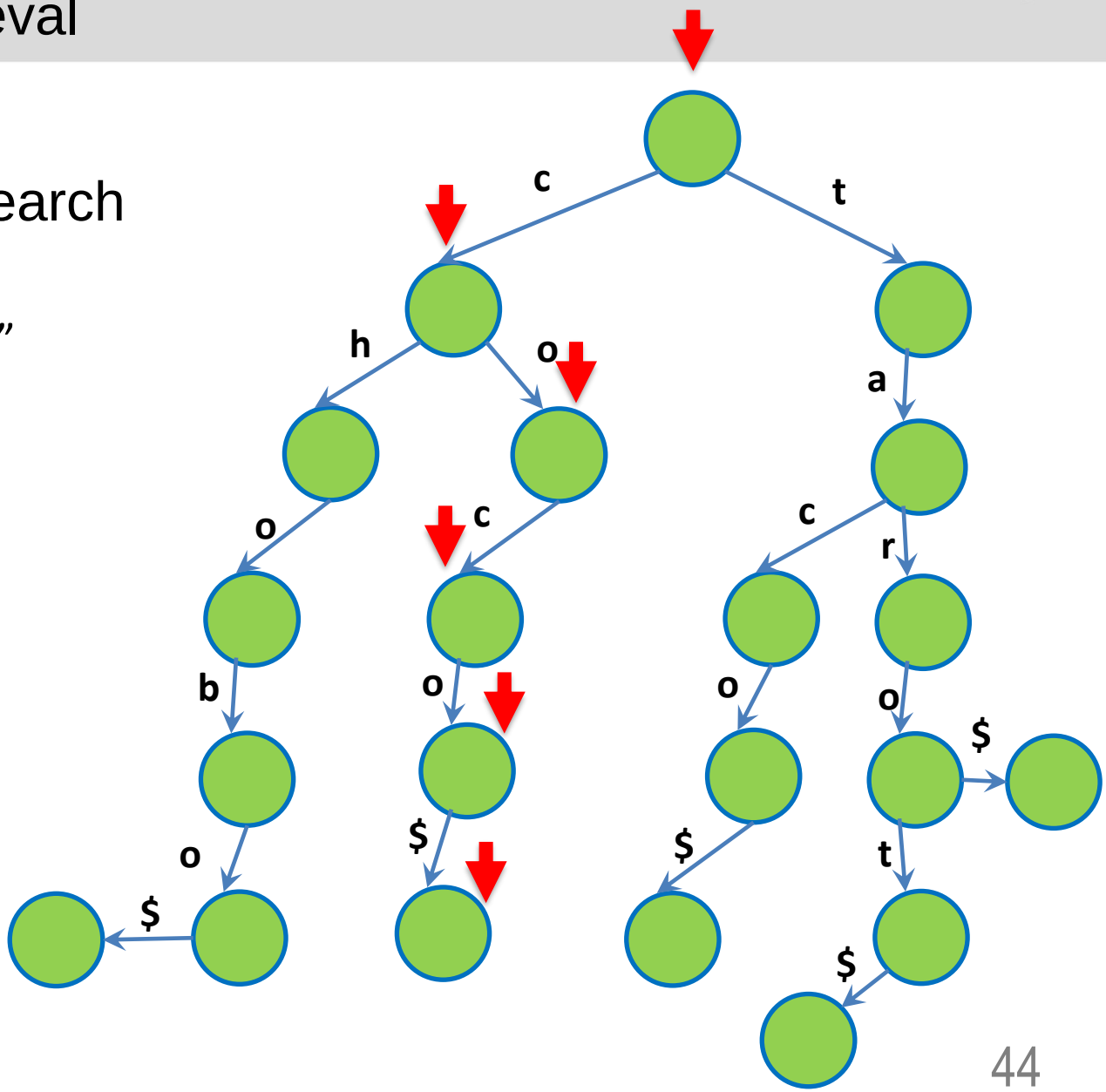
- So how do we search for retrieval?
  - Search for “coco”



Tries

Efficient string retrieval

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

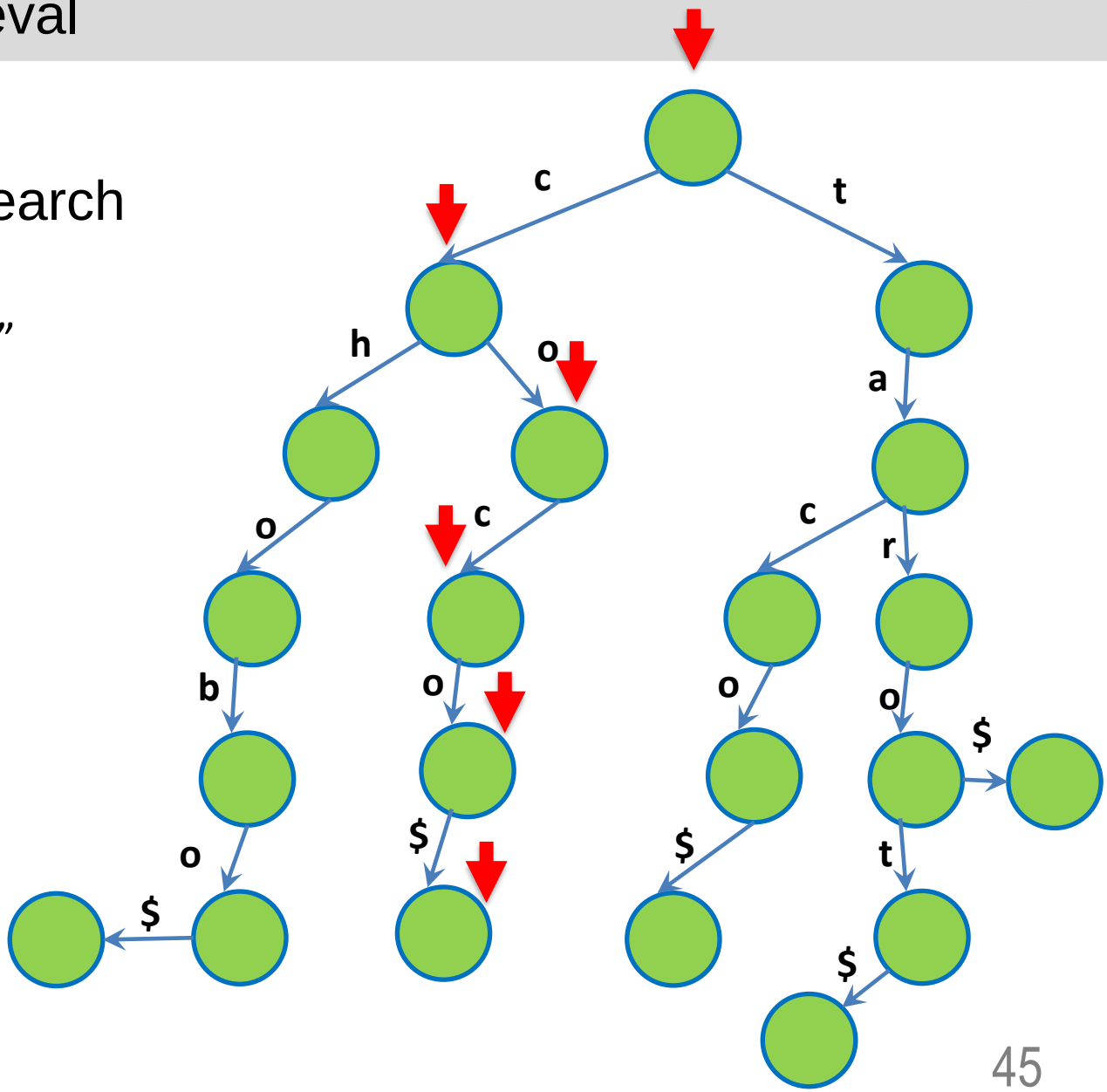


Tries

Efficient string retrieval

MONASH University

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



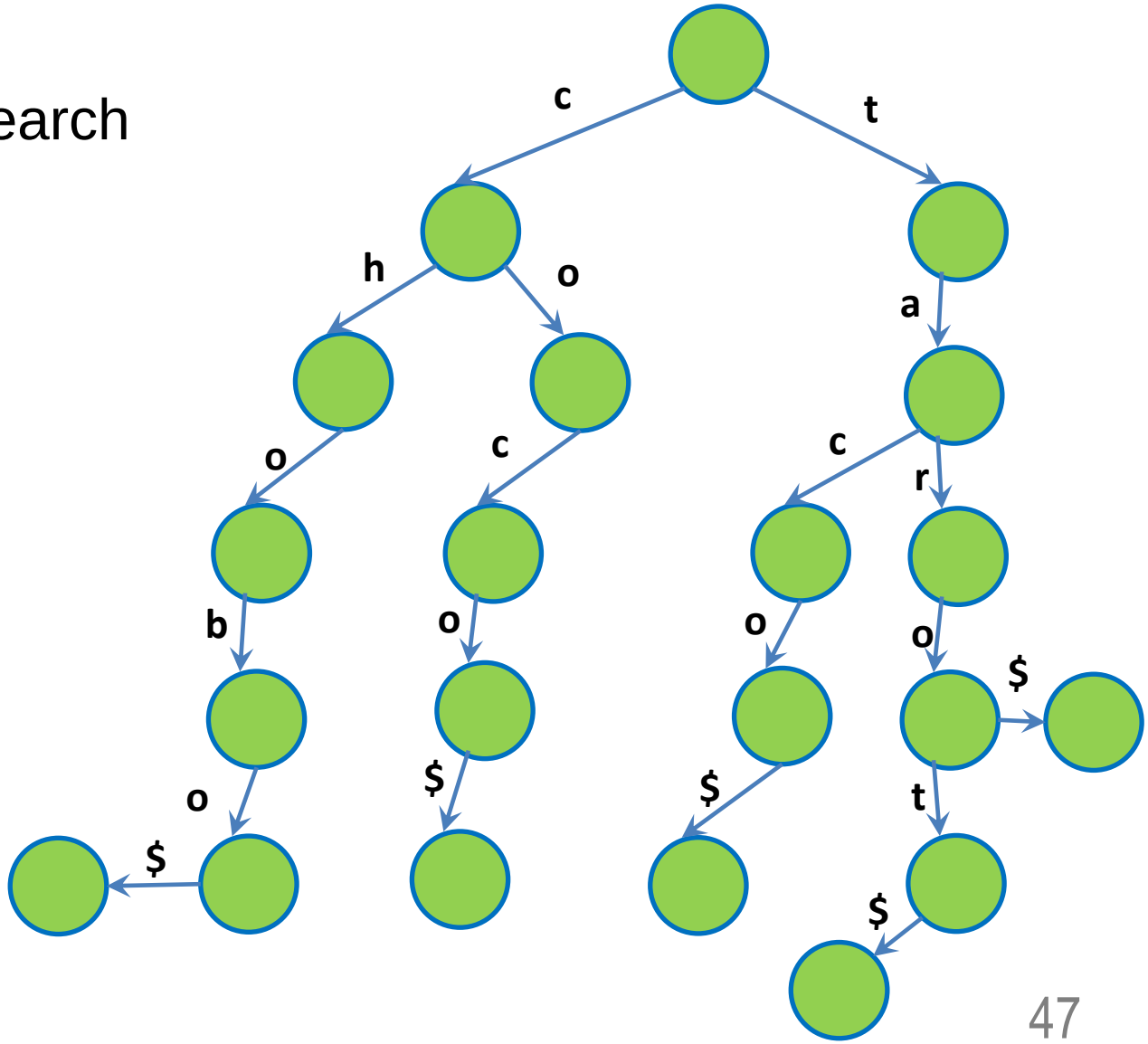
MONASH University

Questions?

Tries

Efficient string retrieval

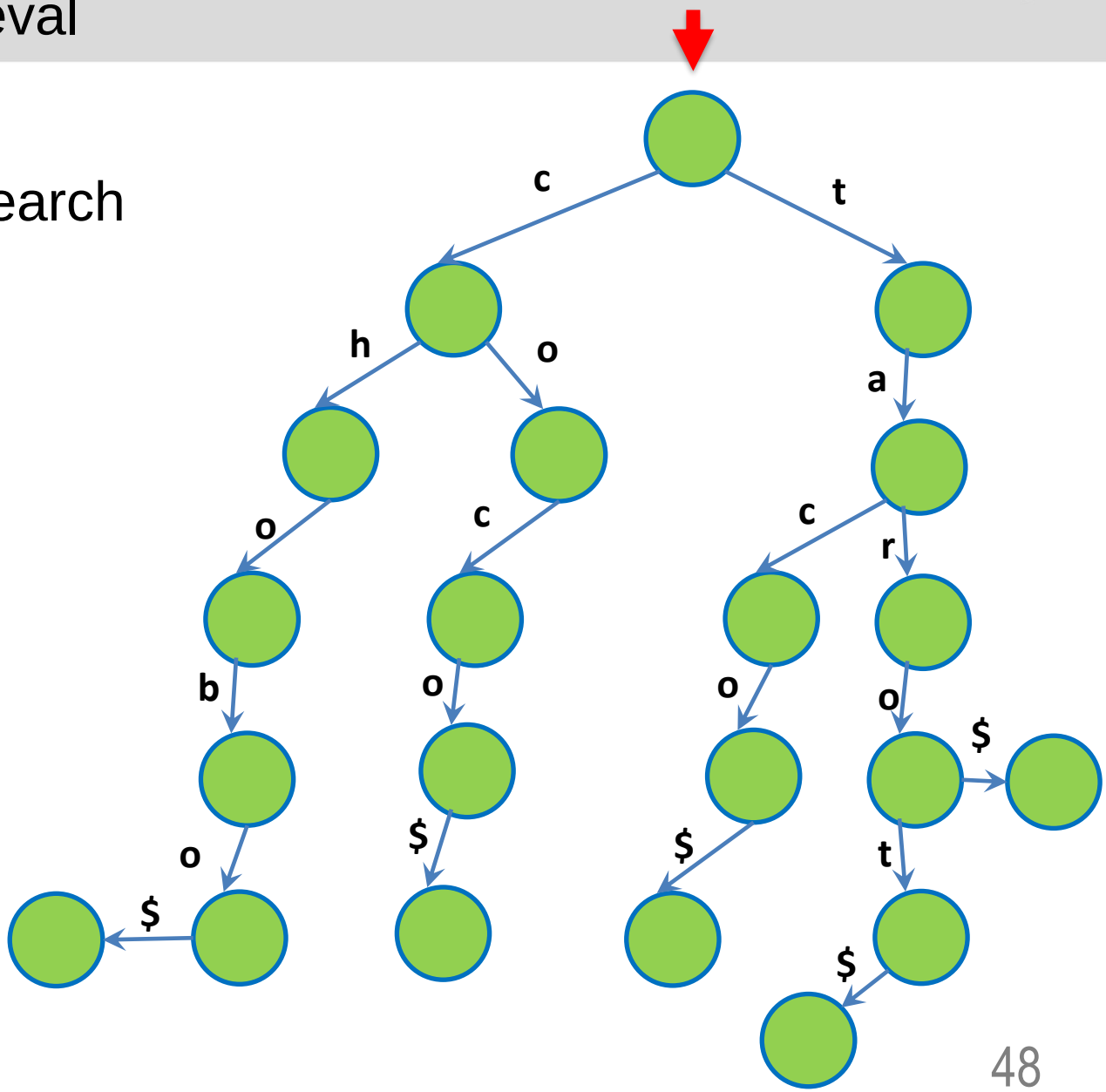
- So how do we search for retrieval?
  - Search for “tarot”



Tries

Efficient string retrieval

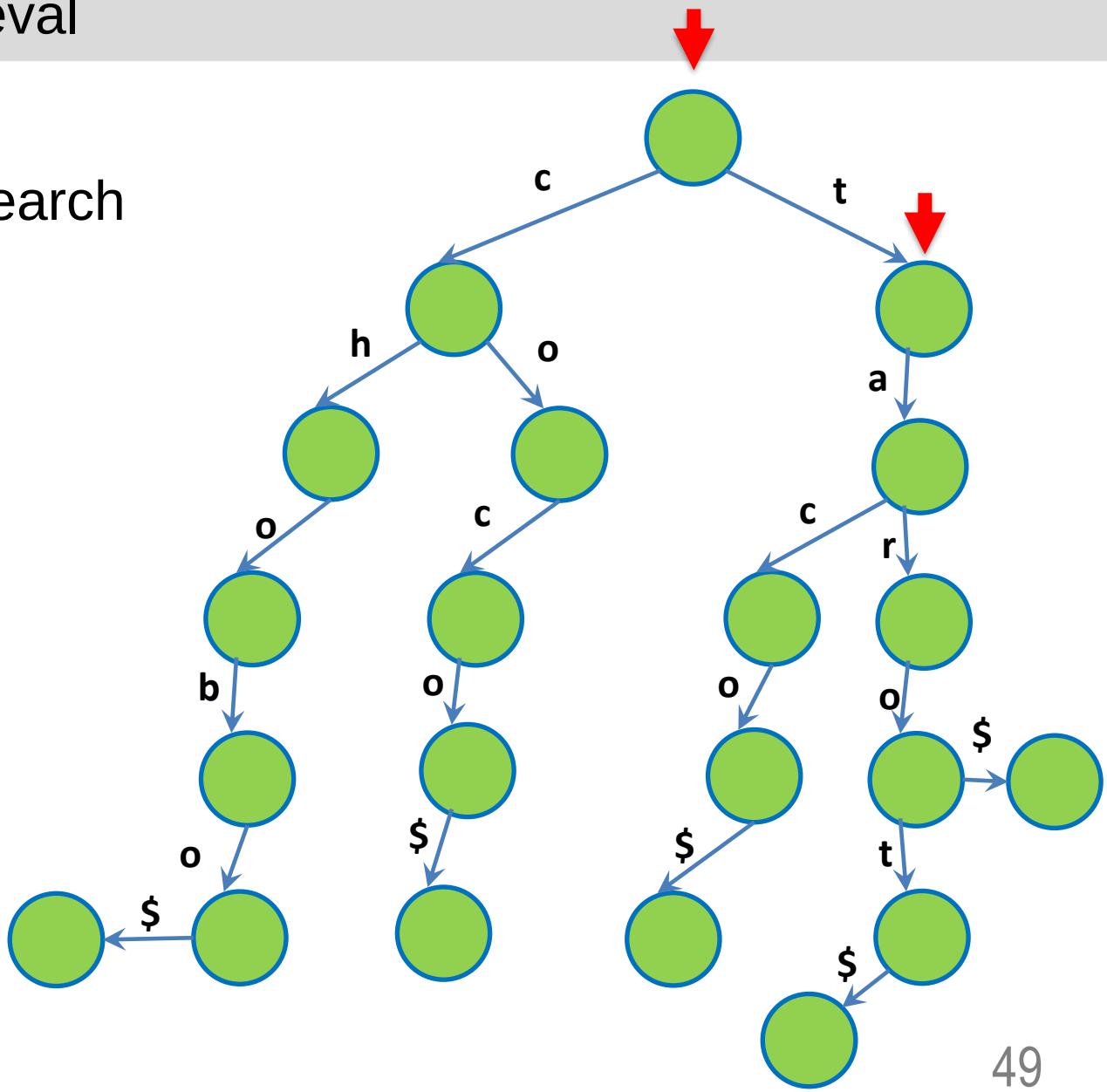
- So how do we search for retrieval?
  - Search for “tarot”



Tries

Efficient string retrieval

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

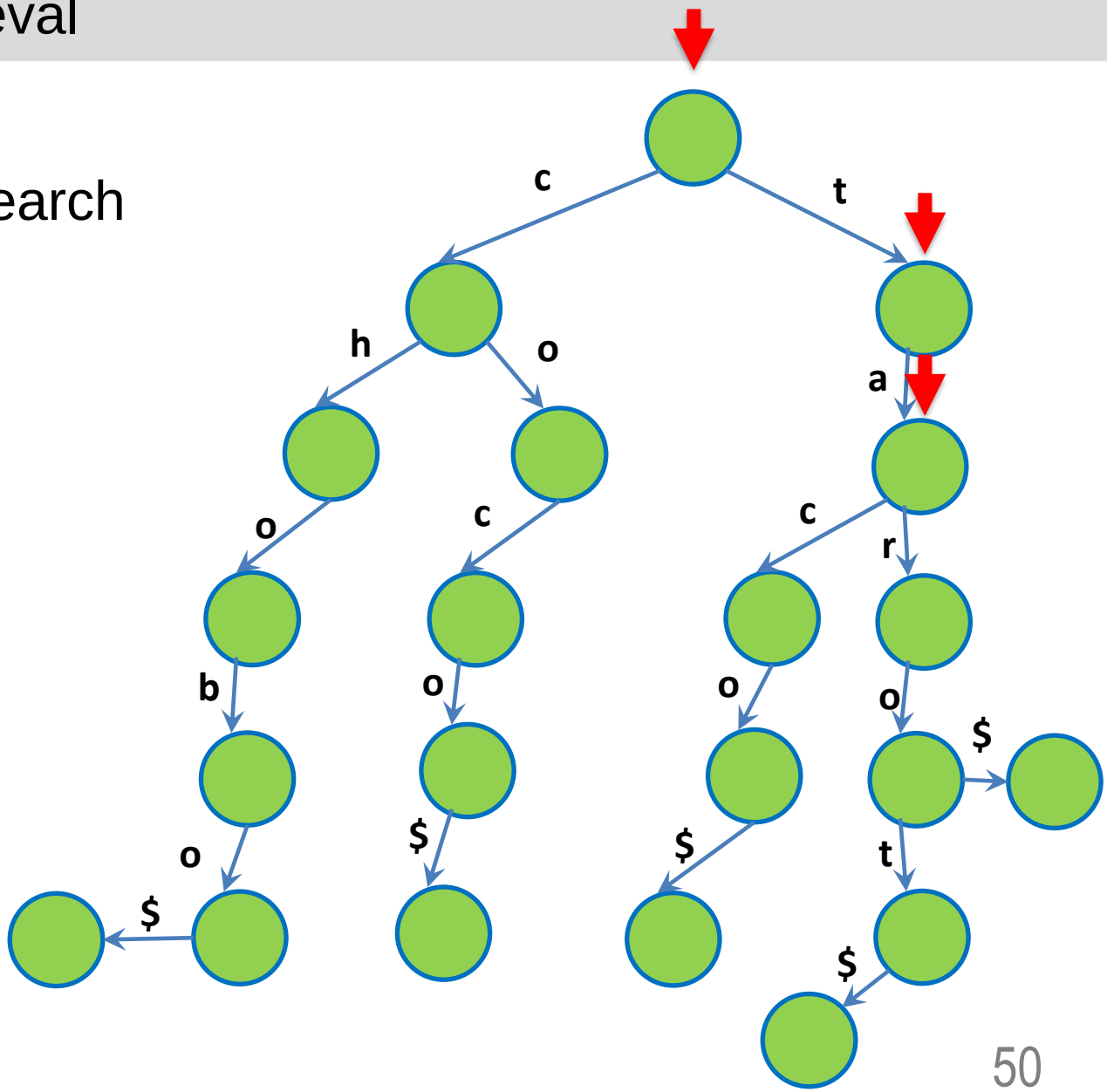


Tries

Efficient string retrieval

MONASH University

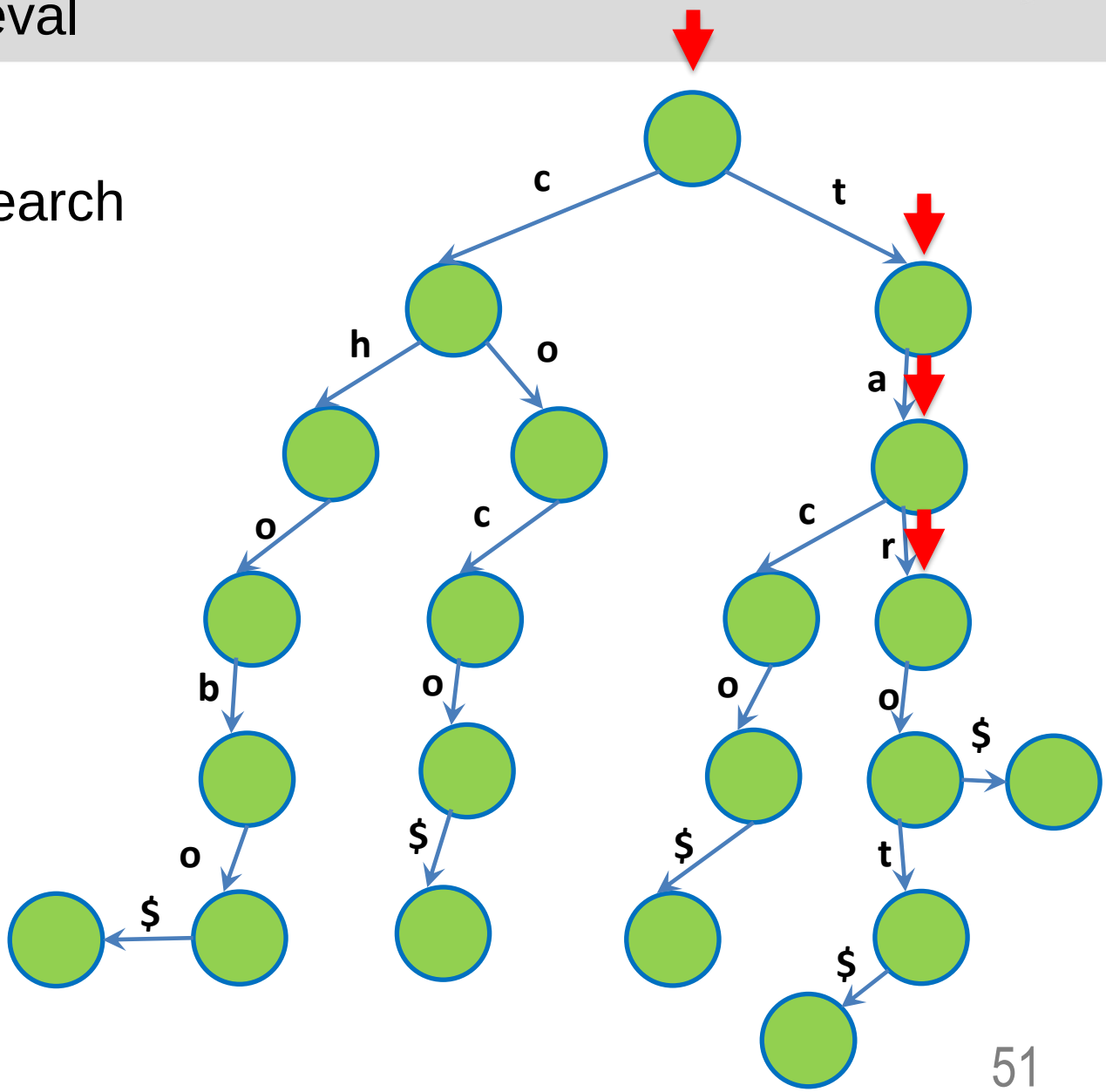
- So how do we search for retrieval?
  - Search for “tarot”



Tries

Efficient string retrieval

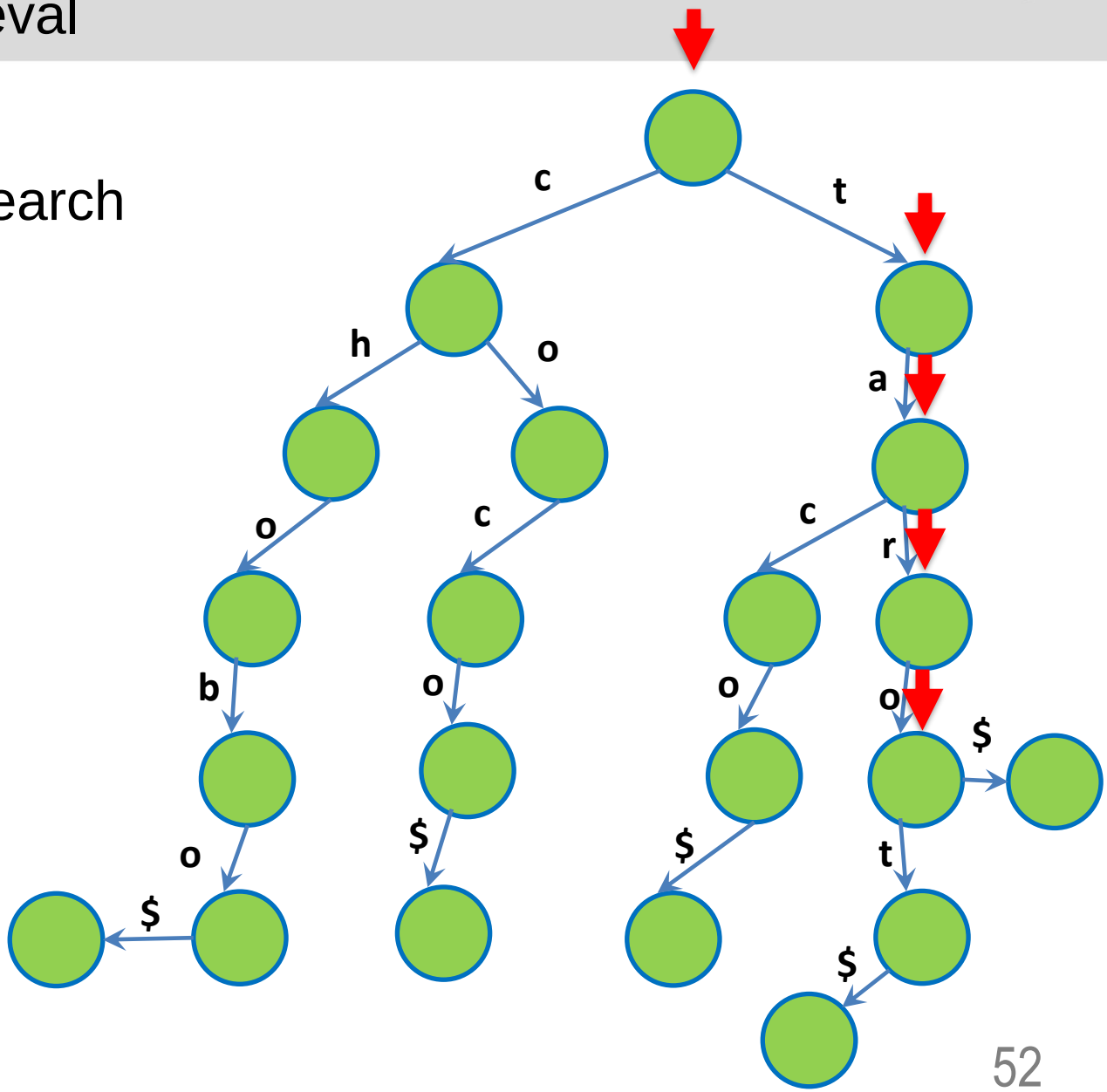
- So how do we search for retrieval?
  - Search for “tarot”



Tries

Efficient string retrieval

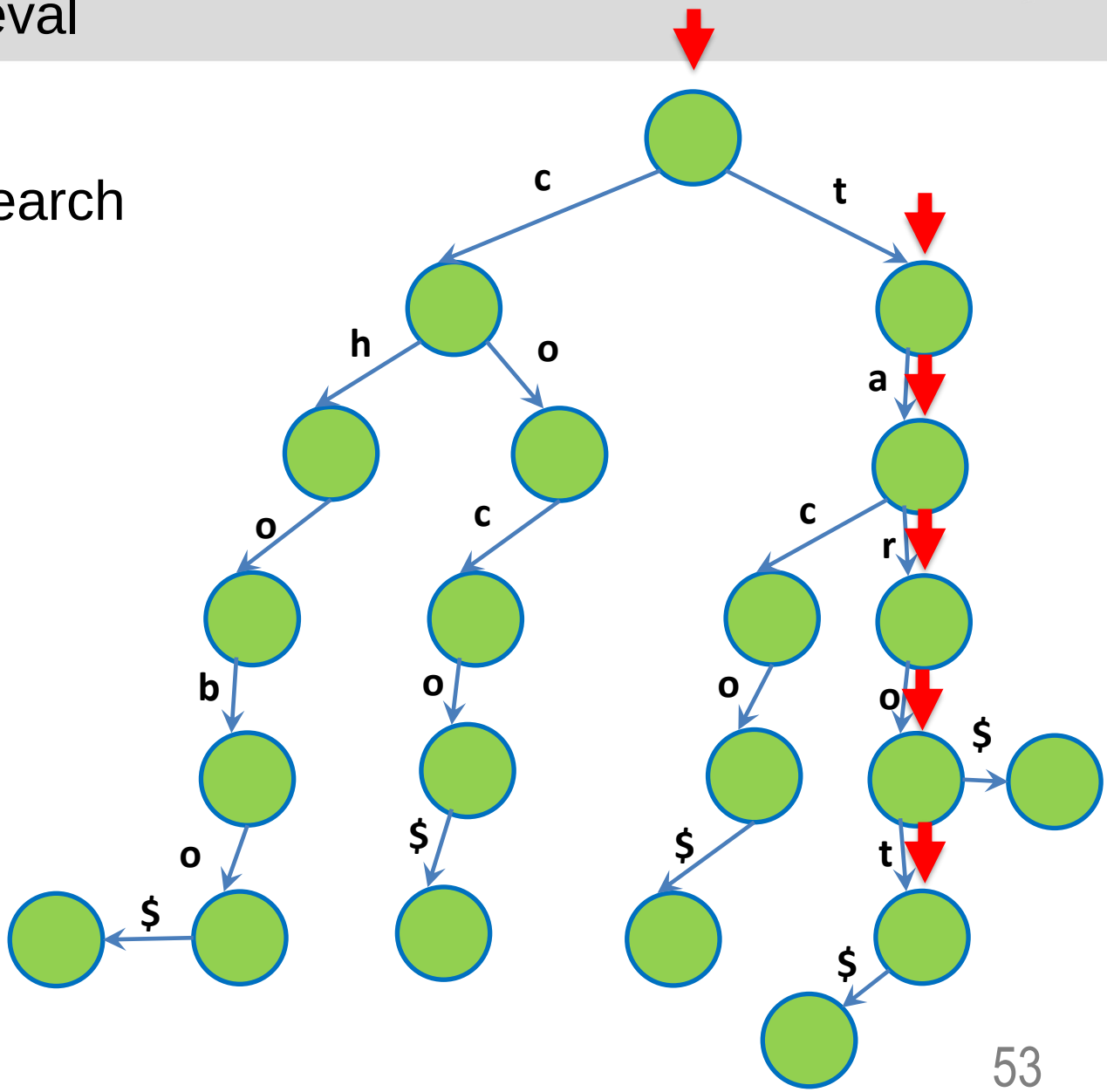
- So how do we search for retrieval?
  - Search for “tarot”



Tries

Efficient string retrieval

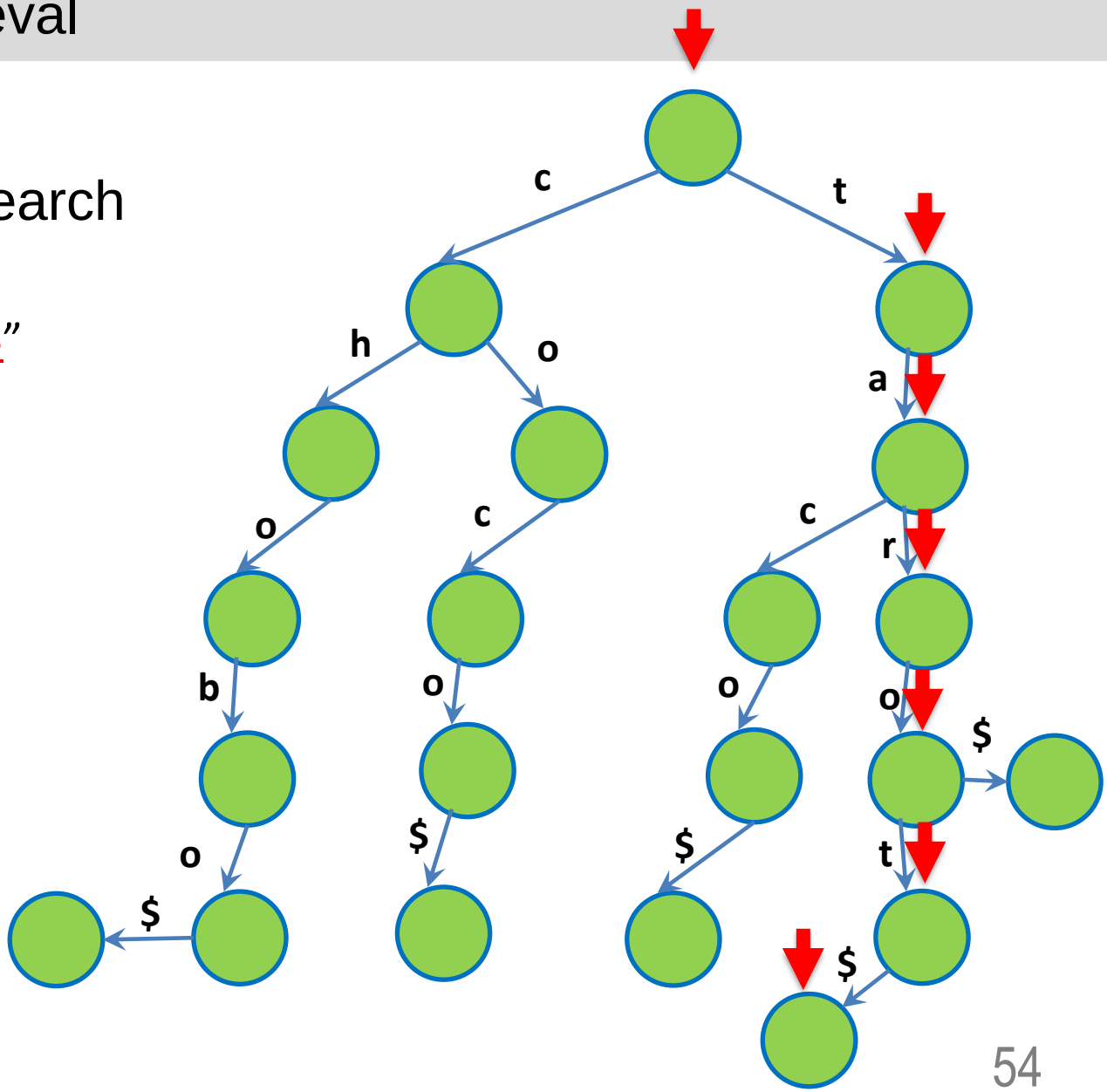
- So how do we search for retrieval?
  - Search for “tarot”



Tries

Efficient string retrieval

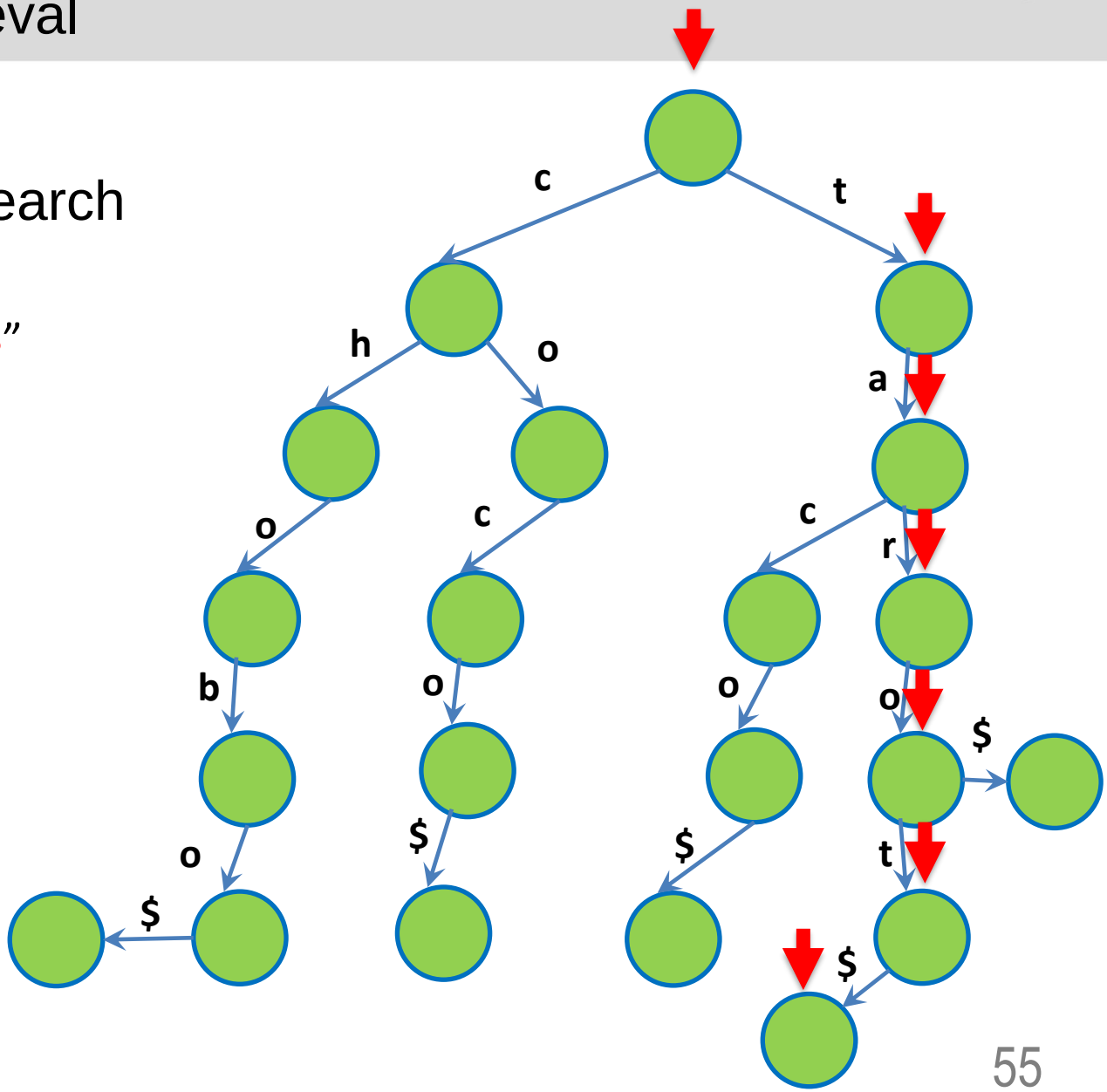
- So how do we search for retrieval?
  - Search for “tarot$”



Tries

Efficient string retrieval

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



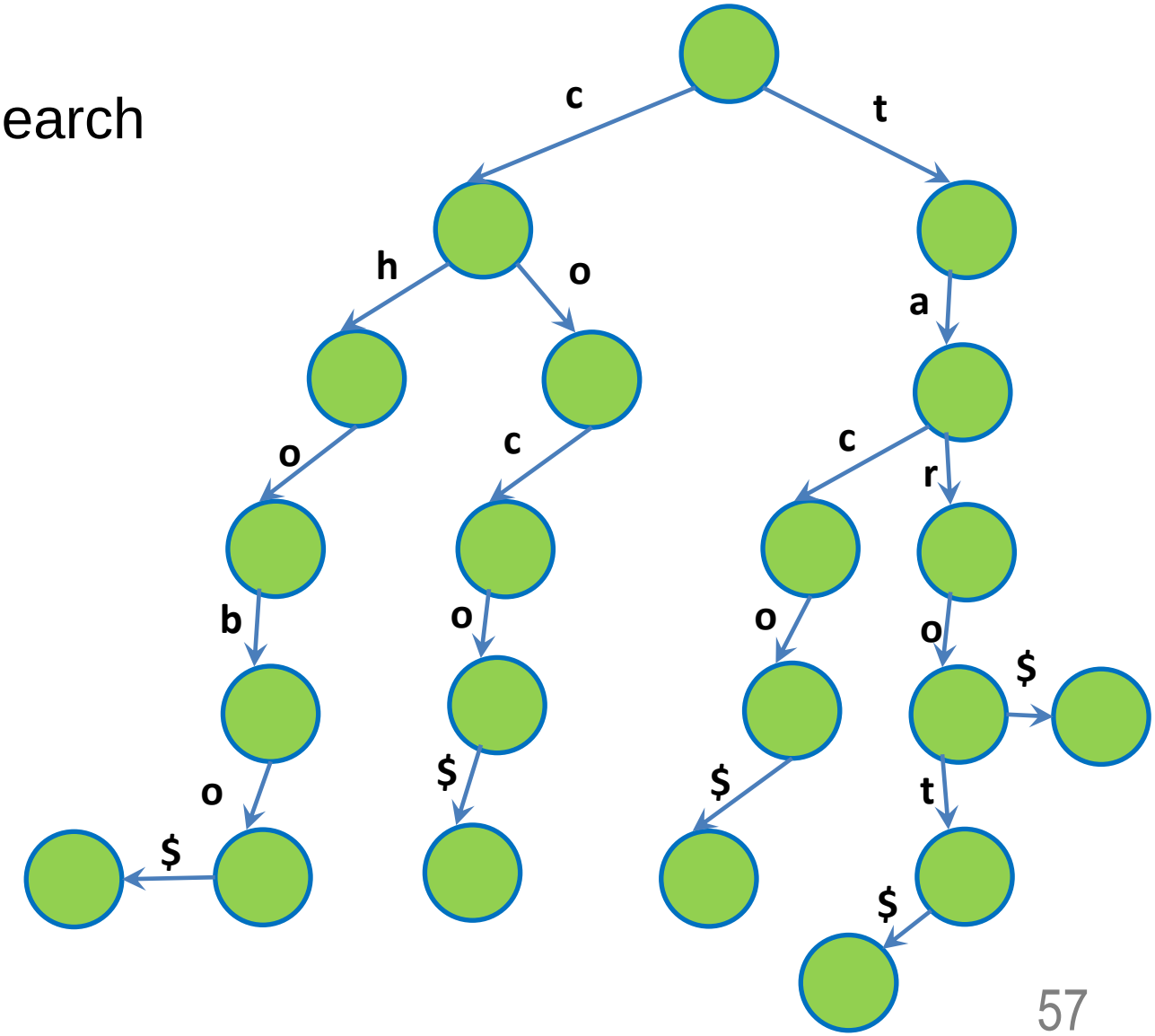
MONASH University

Questions?

Tries

Efficient string retrieval

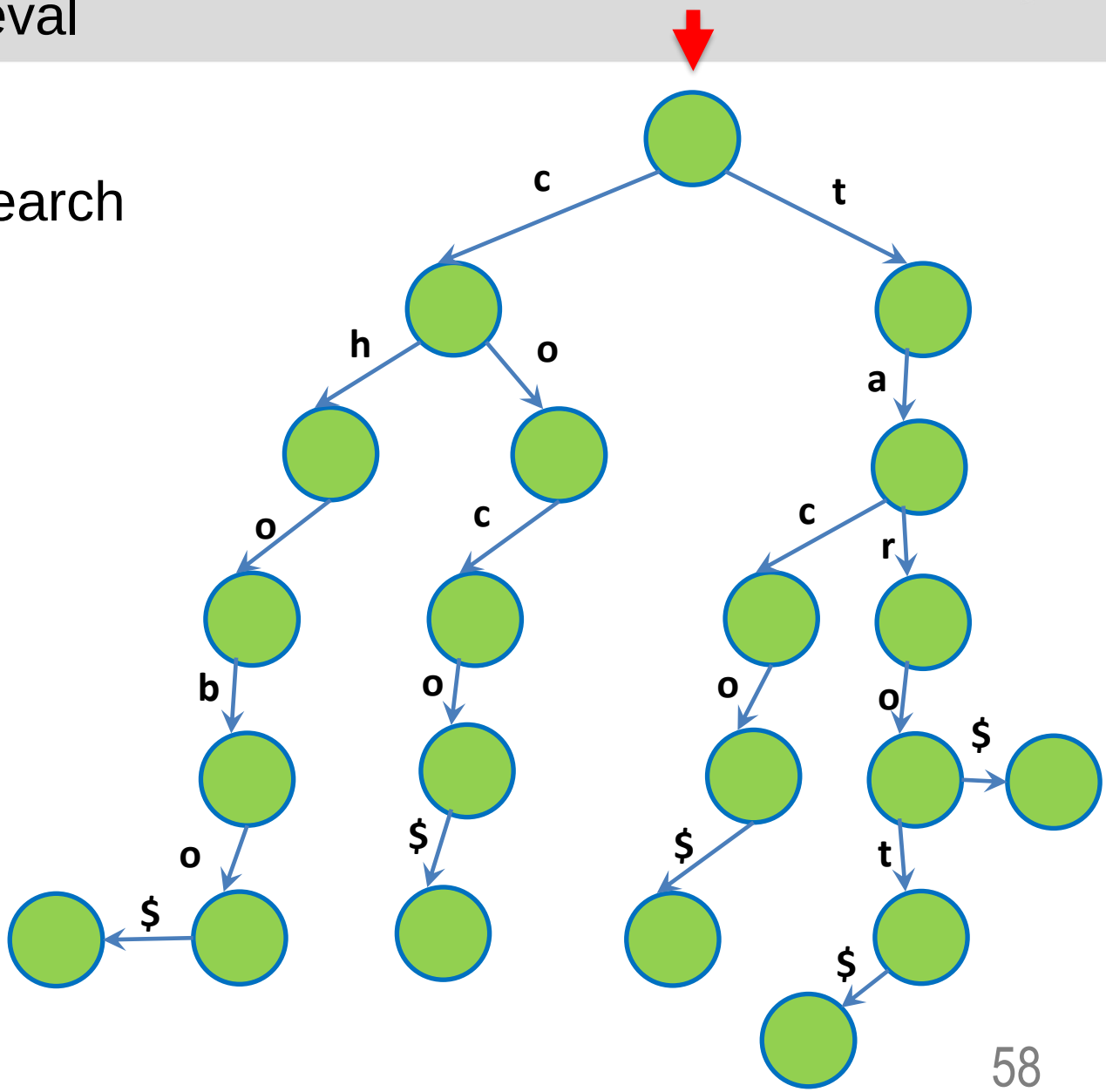
- So how do we search for retrieval?
  - Search for “cow”



Tries

Efficient string retrieval

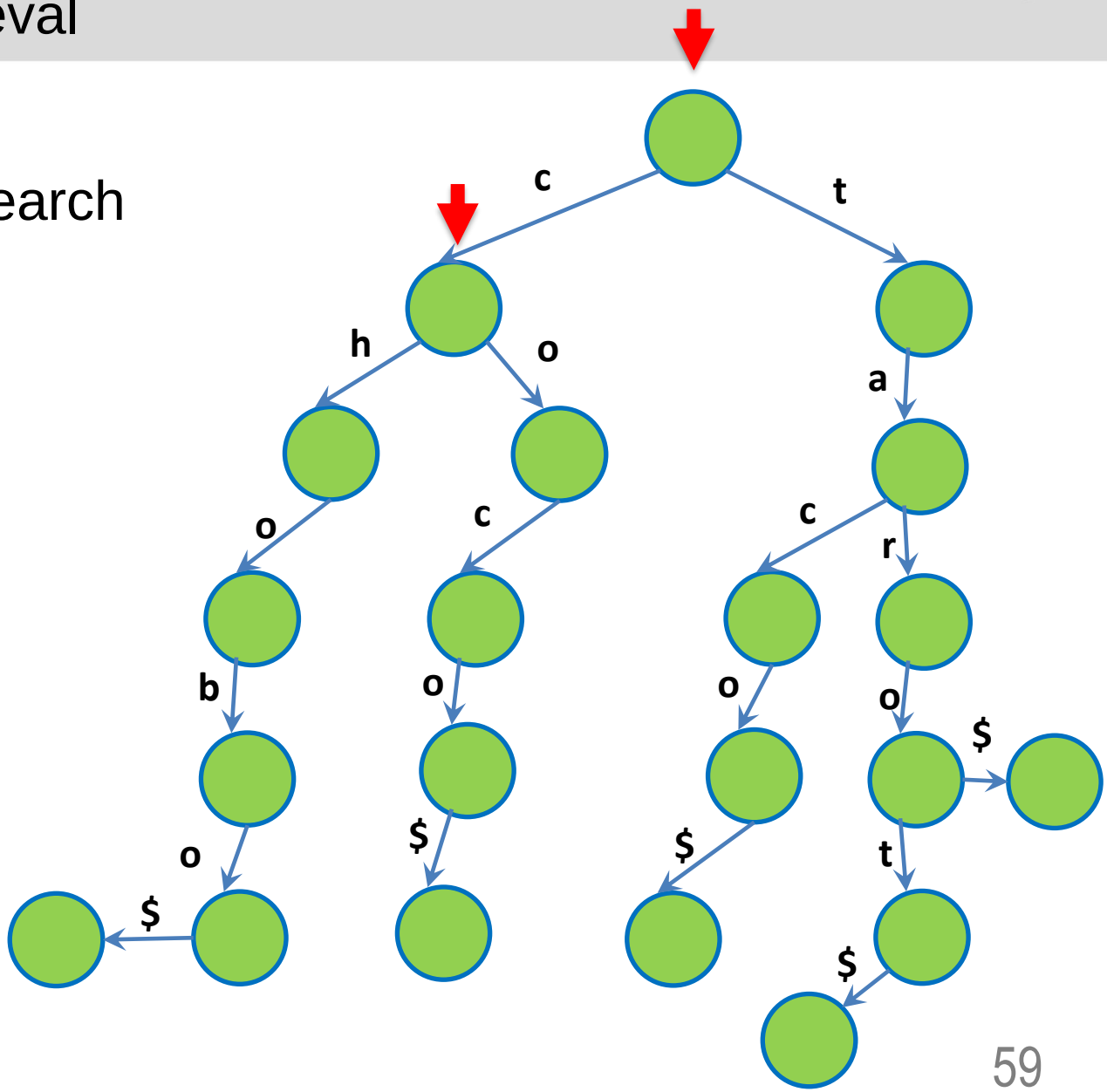
- So how do we search for retrieval?
  - Search for “cow”



Tries

Efficient string retrieval

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

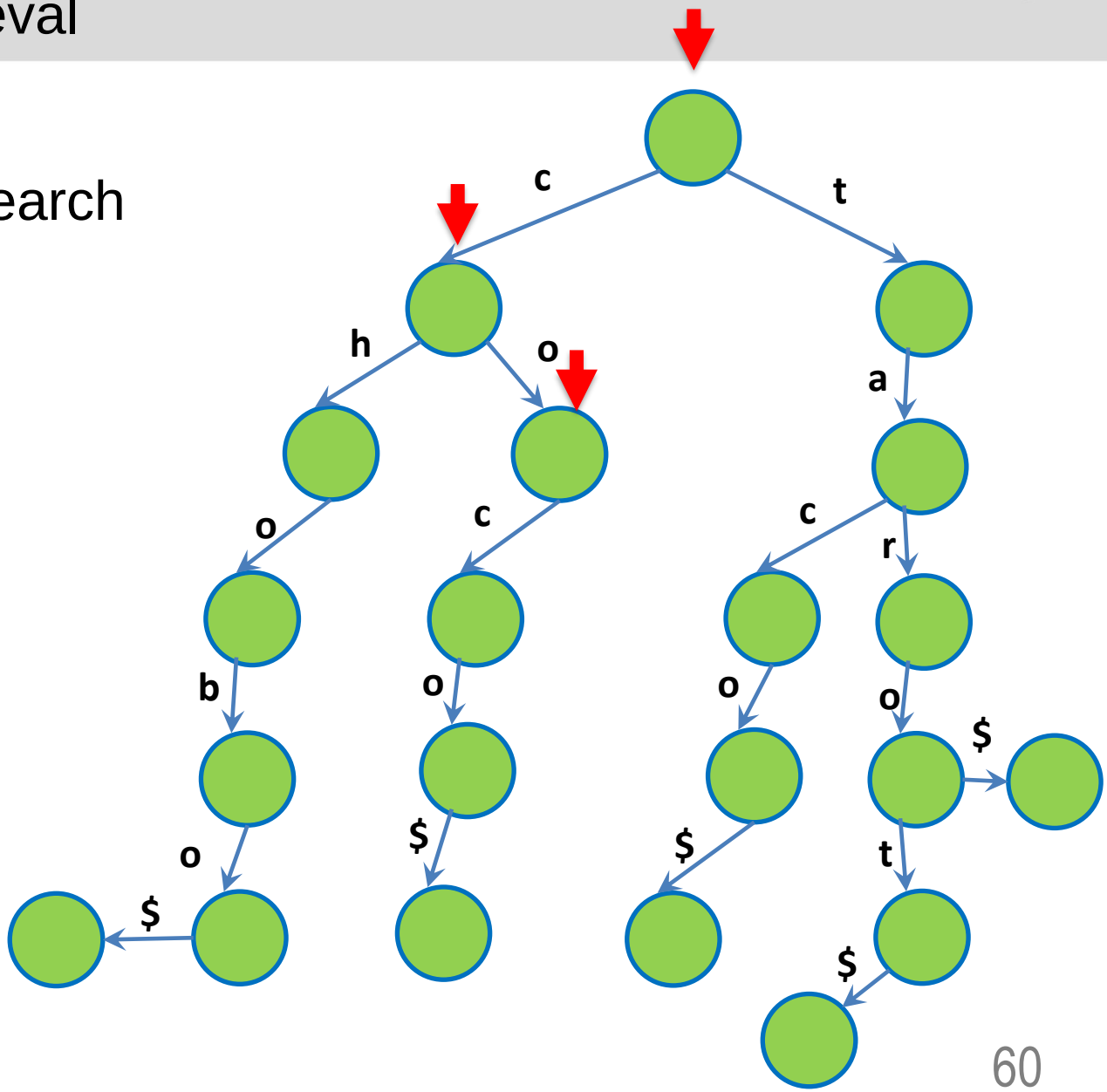


Tries

Efficient string retrieval

MONASH University

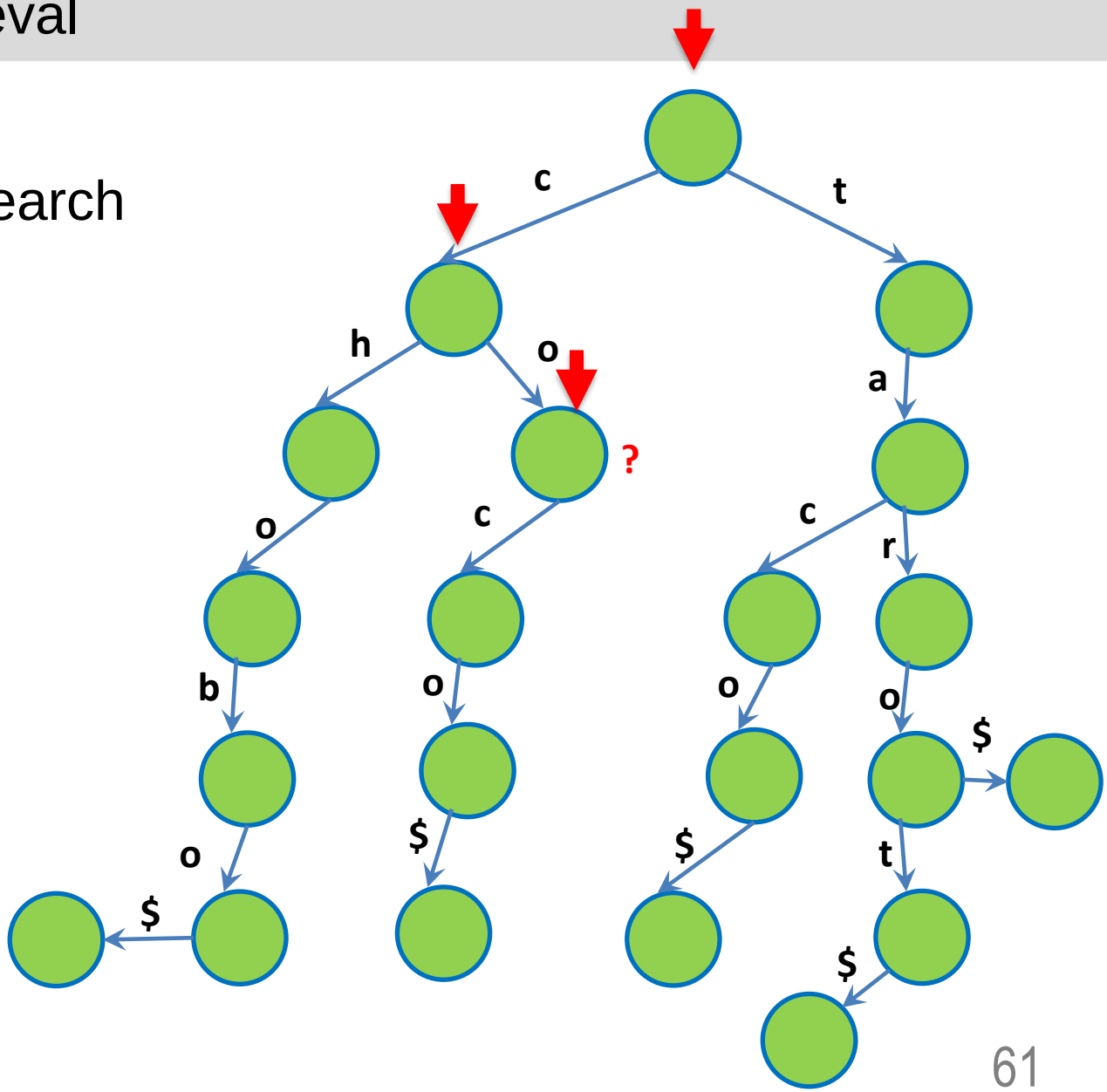
- So how do we search for retrieval?
  - Search for “cow”



Tries

Efficient string retrieval

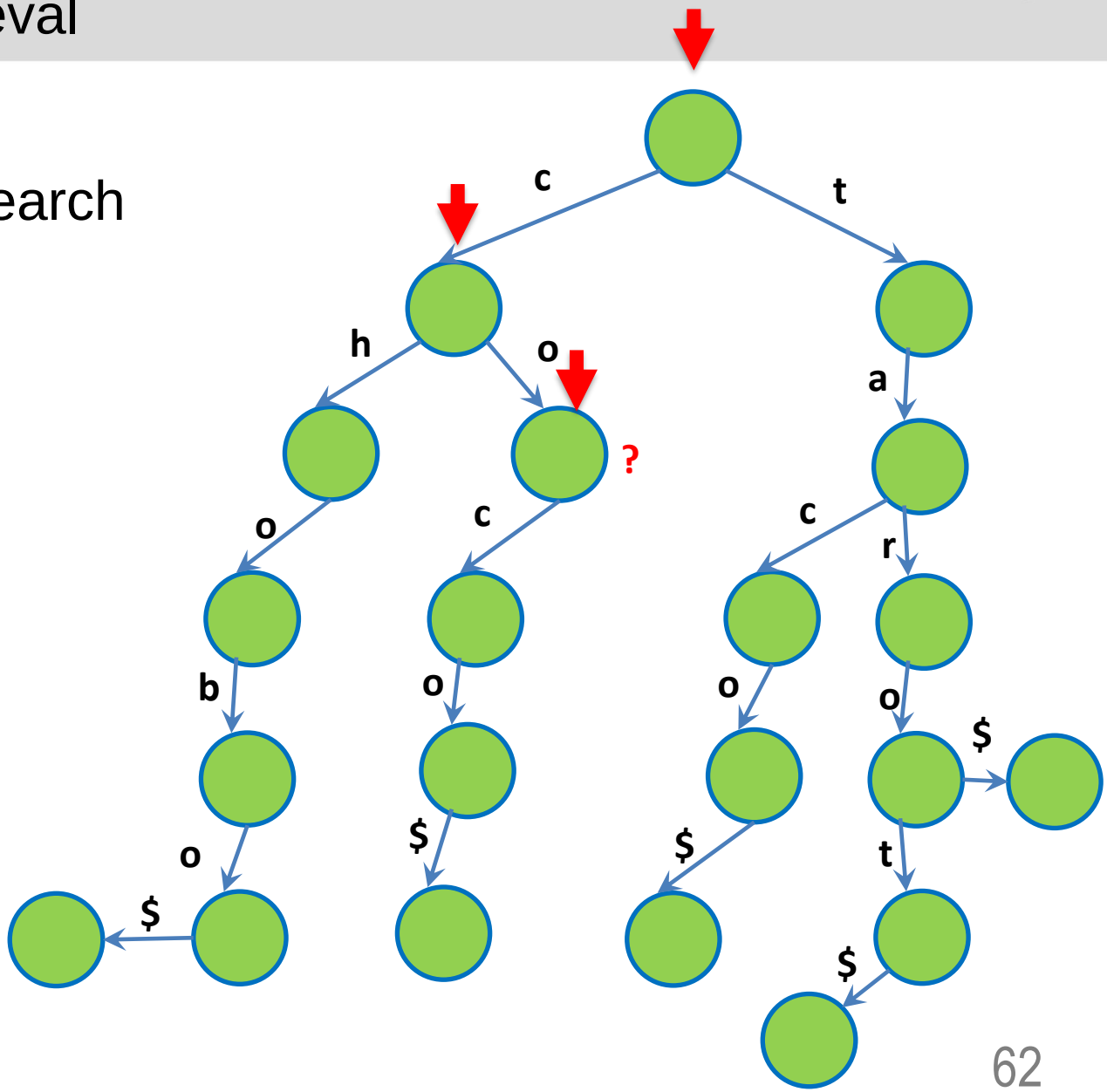
- So how do we search for retrieval?
  - Search for “cow”



Tries

Efficient string retrieval

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



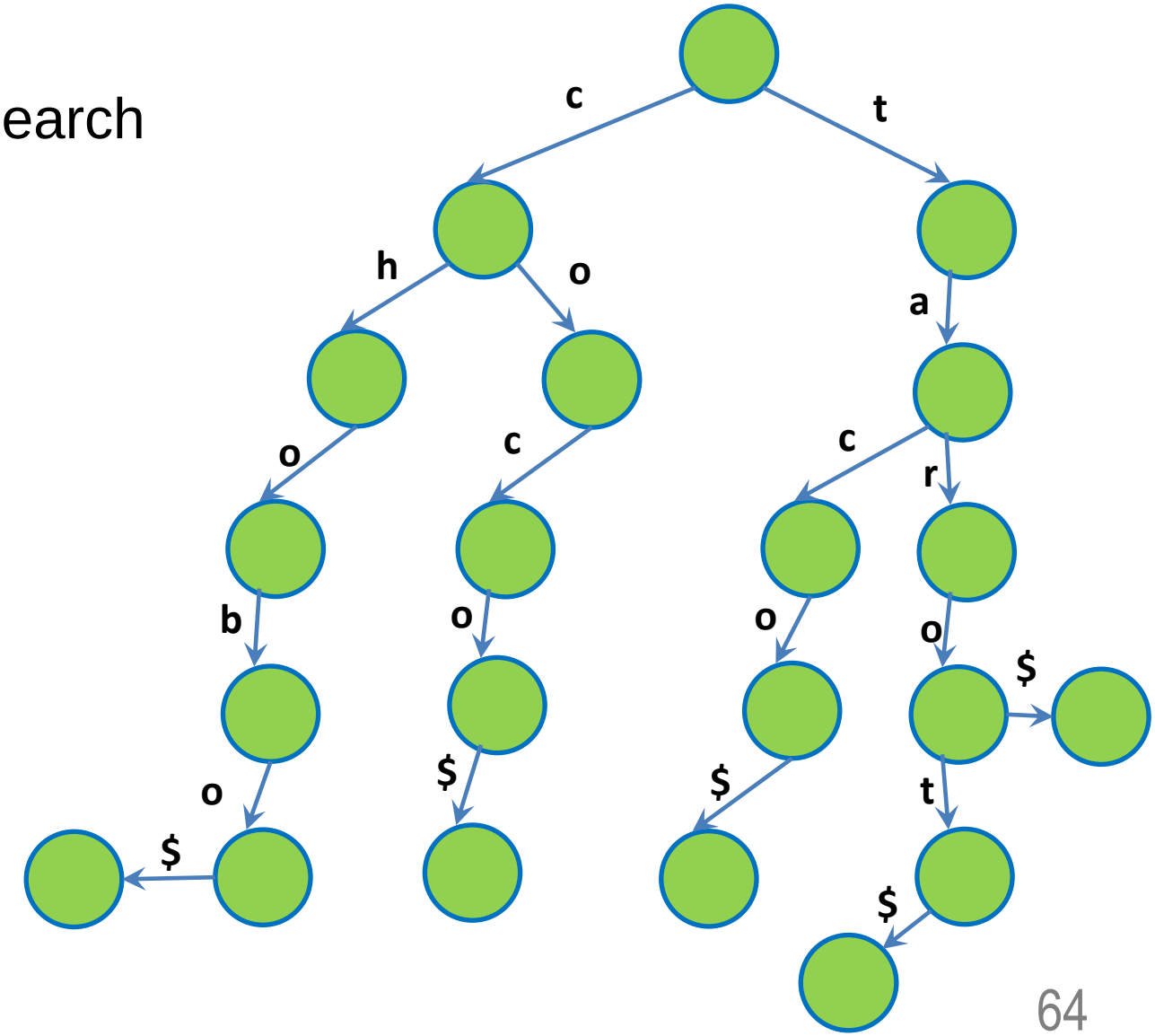
MONASH University

Questions?

Tries

Efficient string retrieval

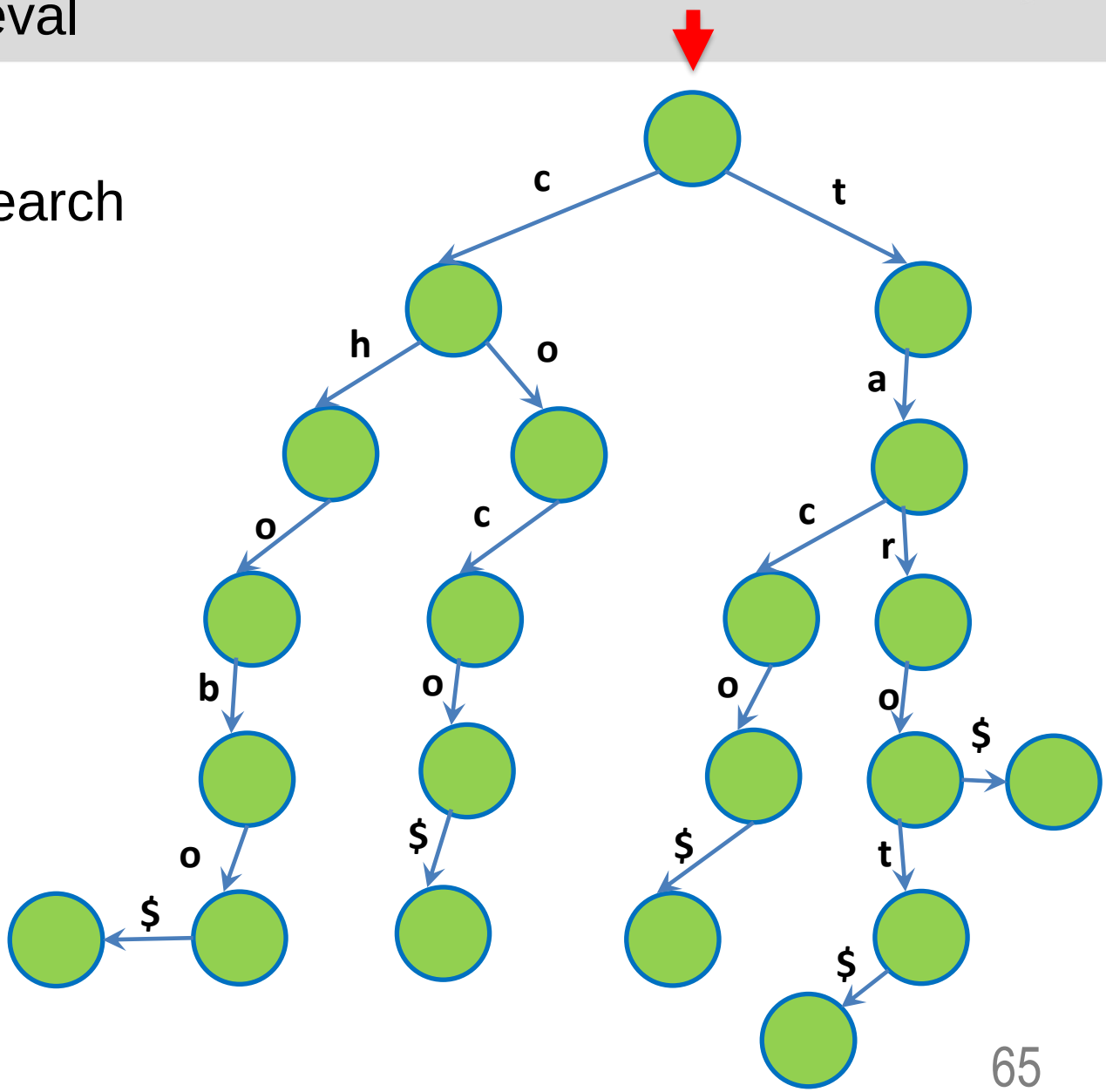
- So how do we search for retrieval?
  - Search for “tar”



Tries

Efficient string retrieval

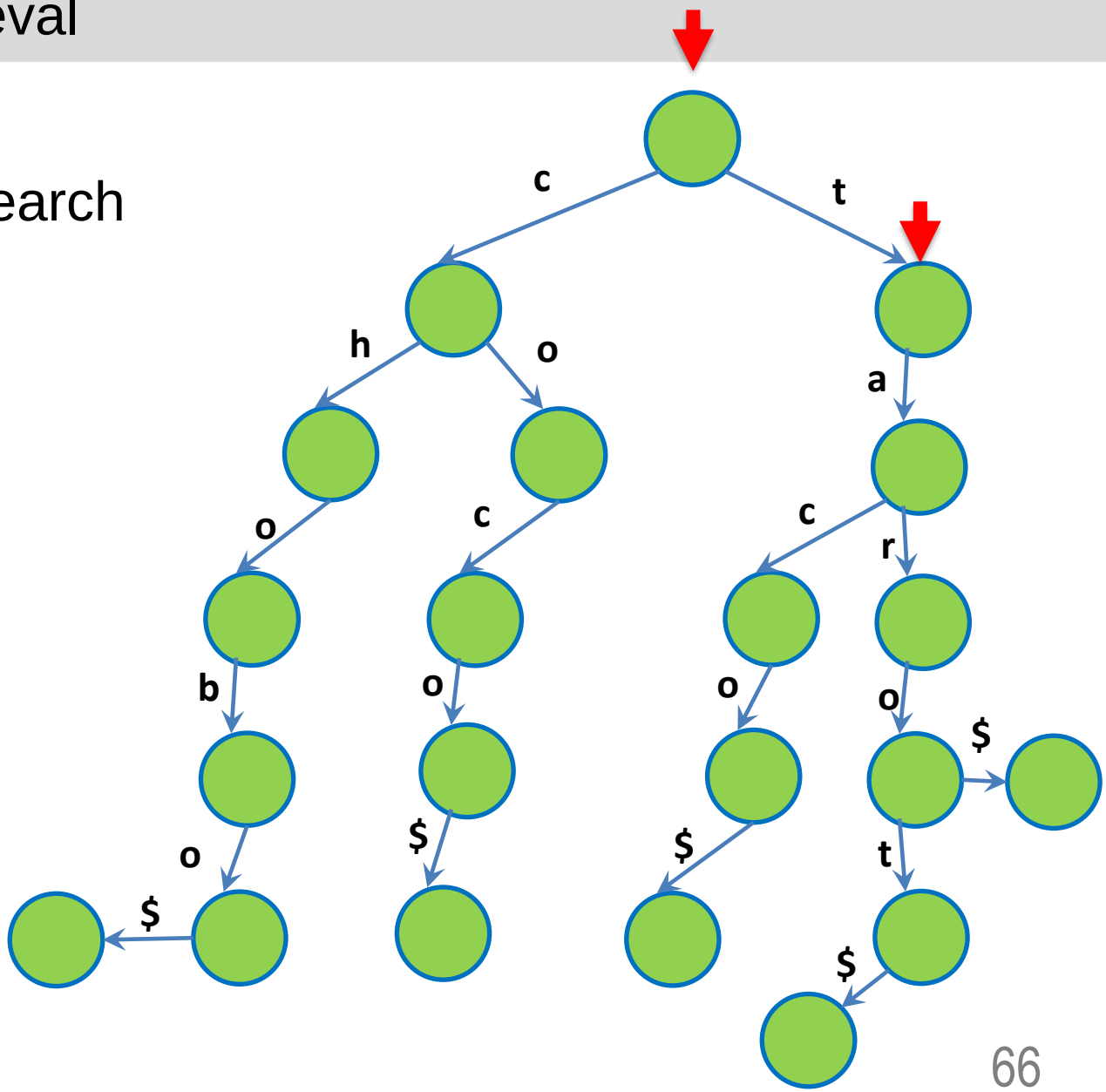
- So how do we search for retrieval?
  - Search for “tar”



Tries

Efficient string retrieval

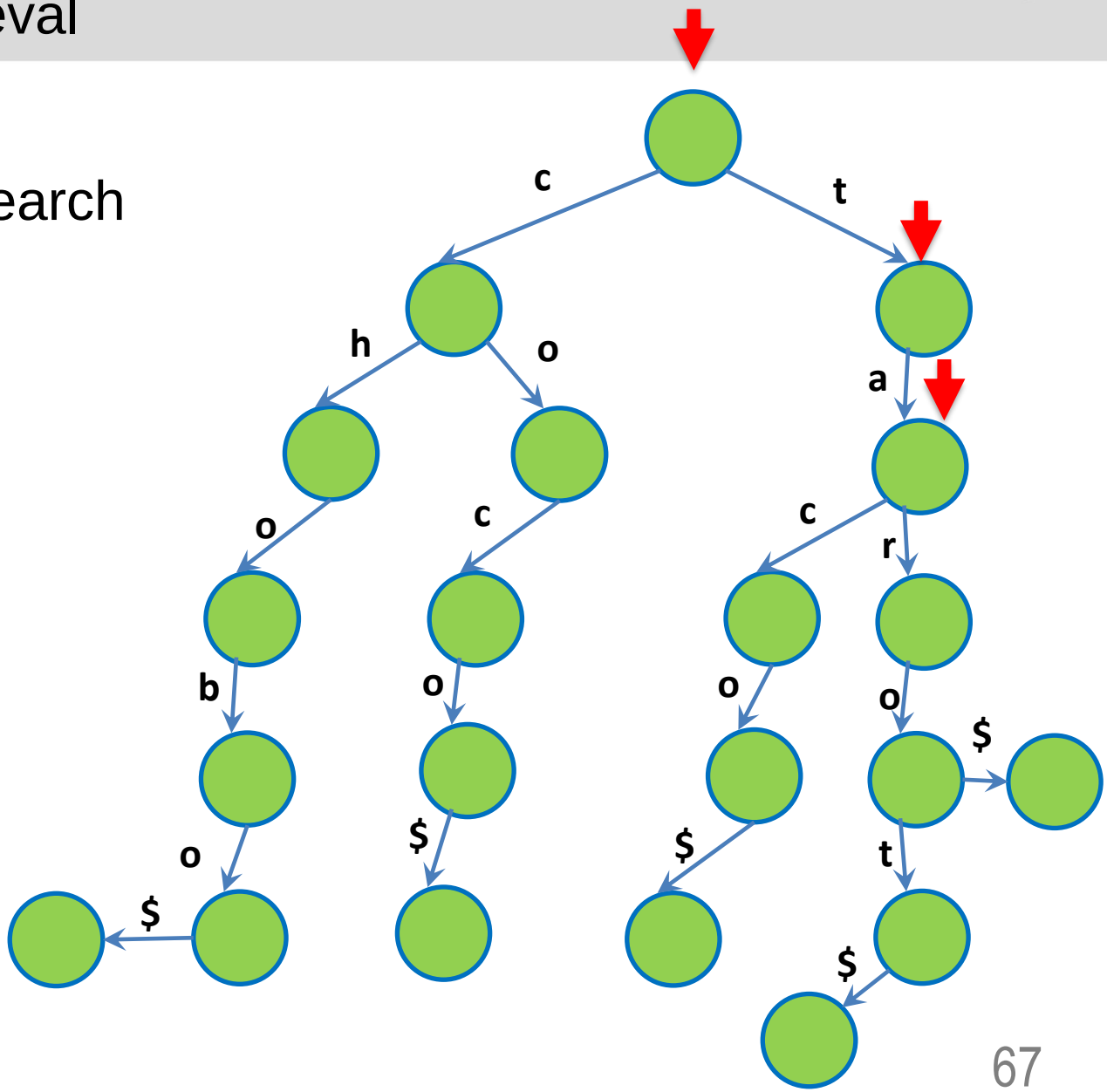
- So how do we search for retrieval?
  - Search for “tar”



Tries

Efficient string retrieval

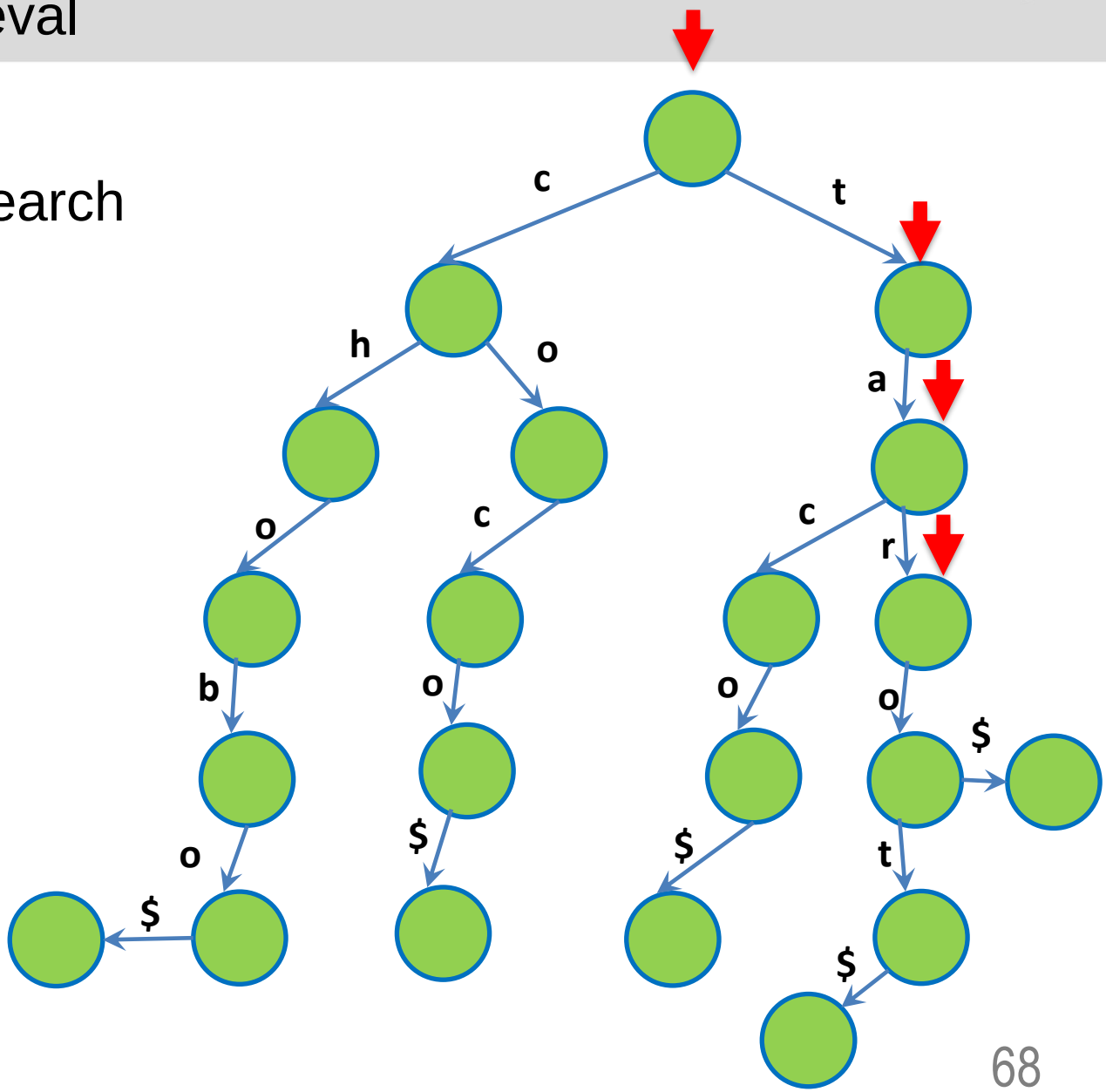
- So how do we search for retrieval?
  - Search for “tar”



Tries

Efficient string retrieval

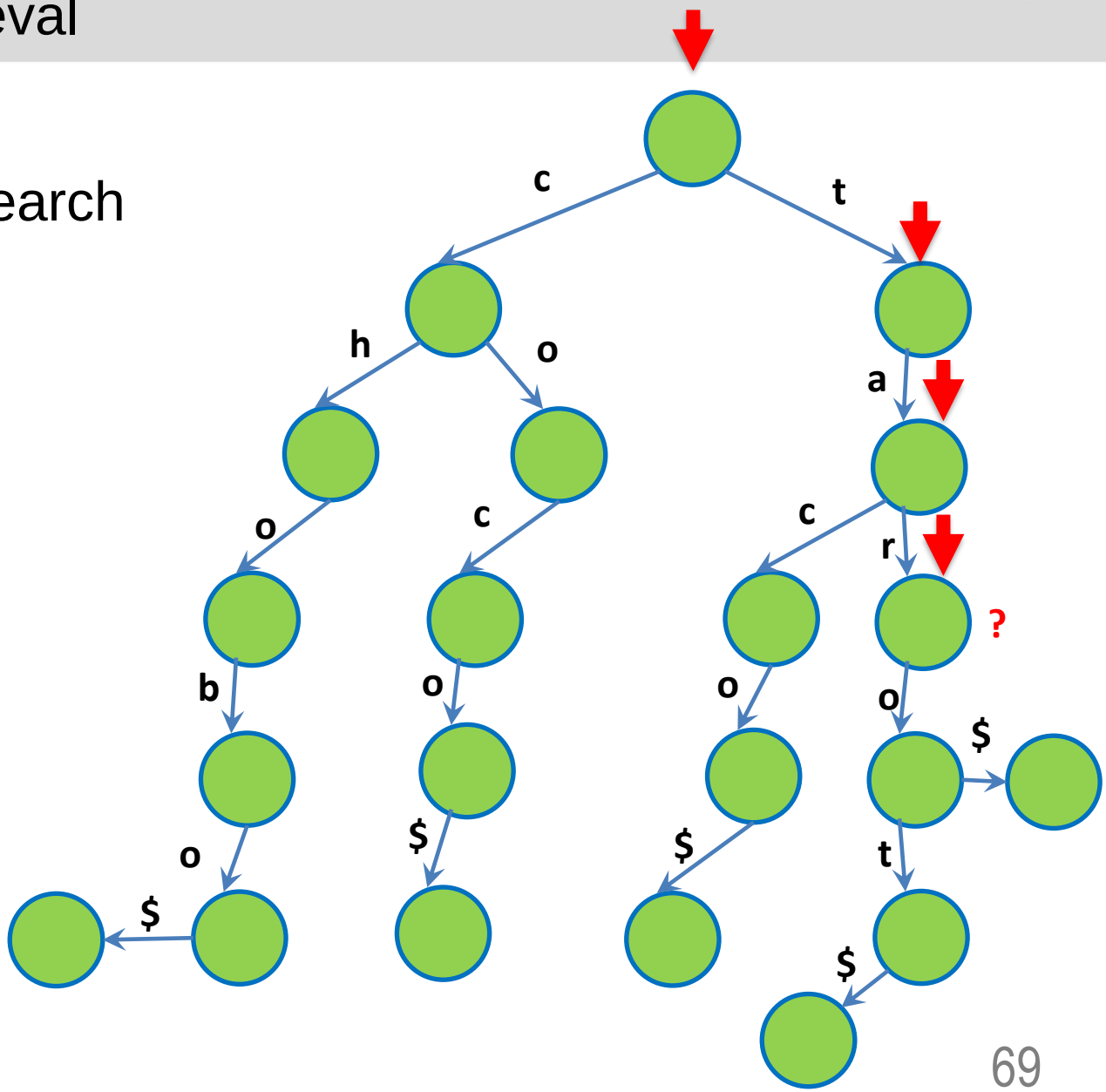
- So how do we search for retrieval?
  - Search for “tar”



Tries

Efficient string retrieval

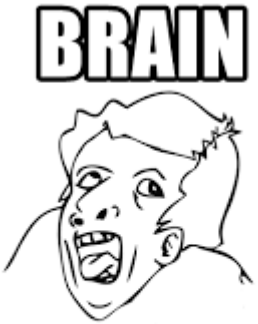
- So how do we search for retrieval?
  - Search for “tar$”

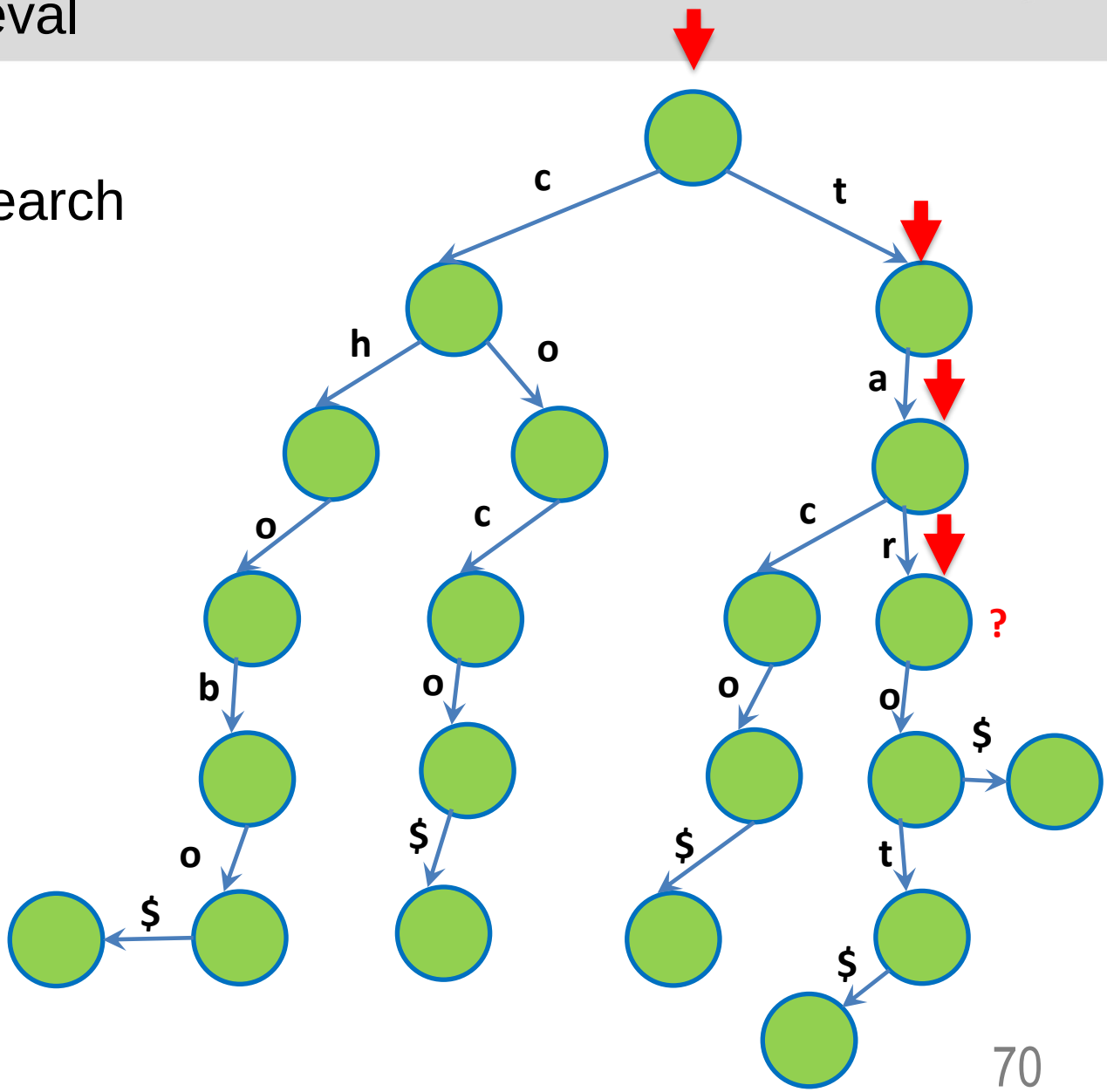


Tries

Efficient string retrieval

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

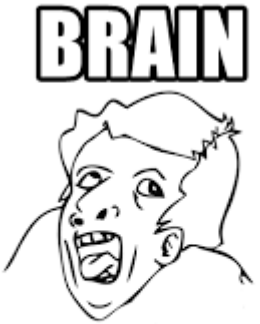


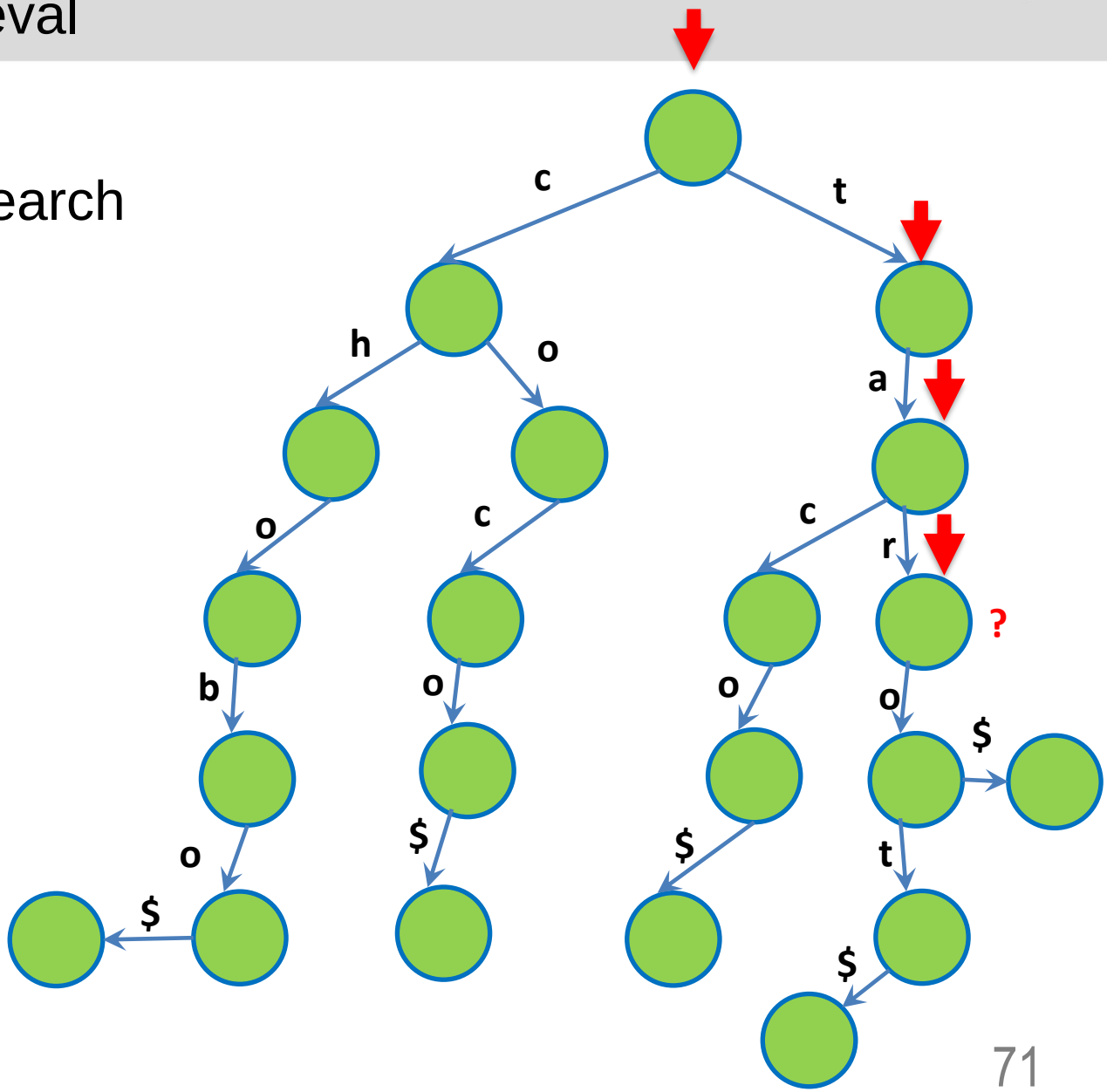


Tries

Efficient string retrieval

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





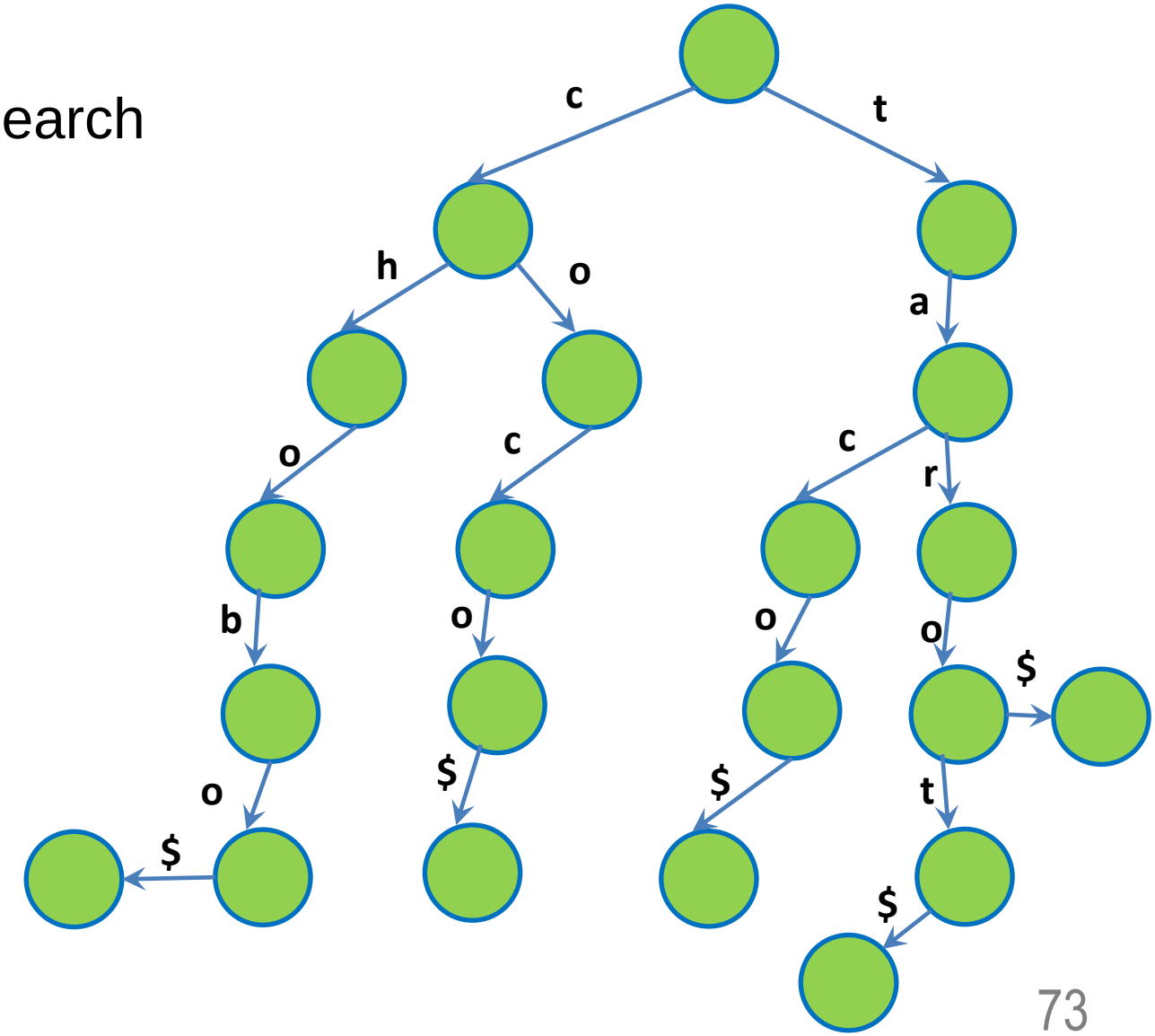
MONASH University

Questions?

Tries

Efficient string retrieval

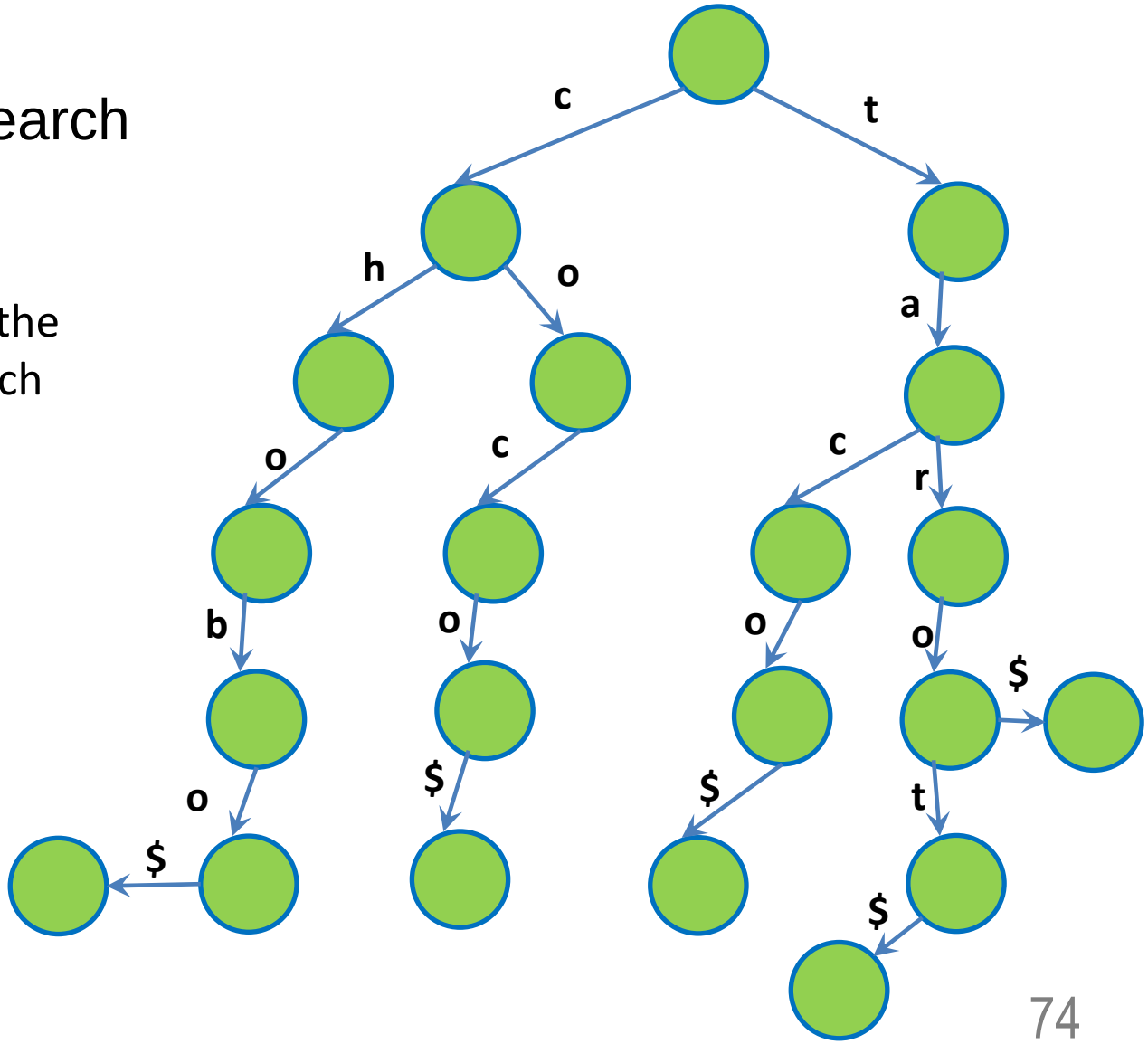
- So how do we search for retrieval?
  - Complexity?



Tries

Efficient string retrieval

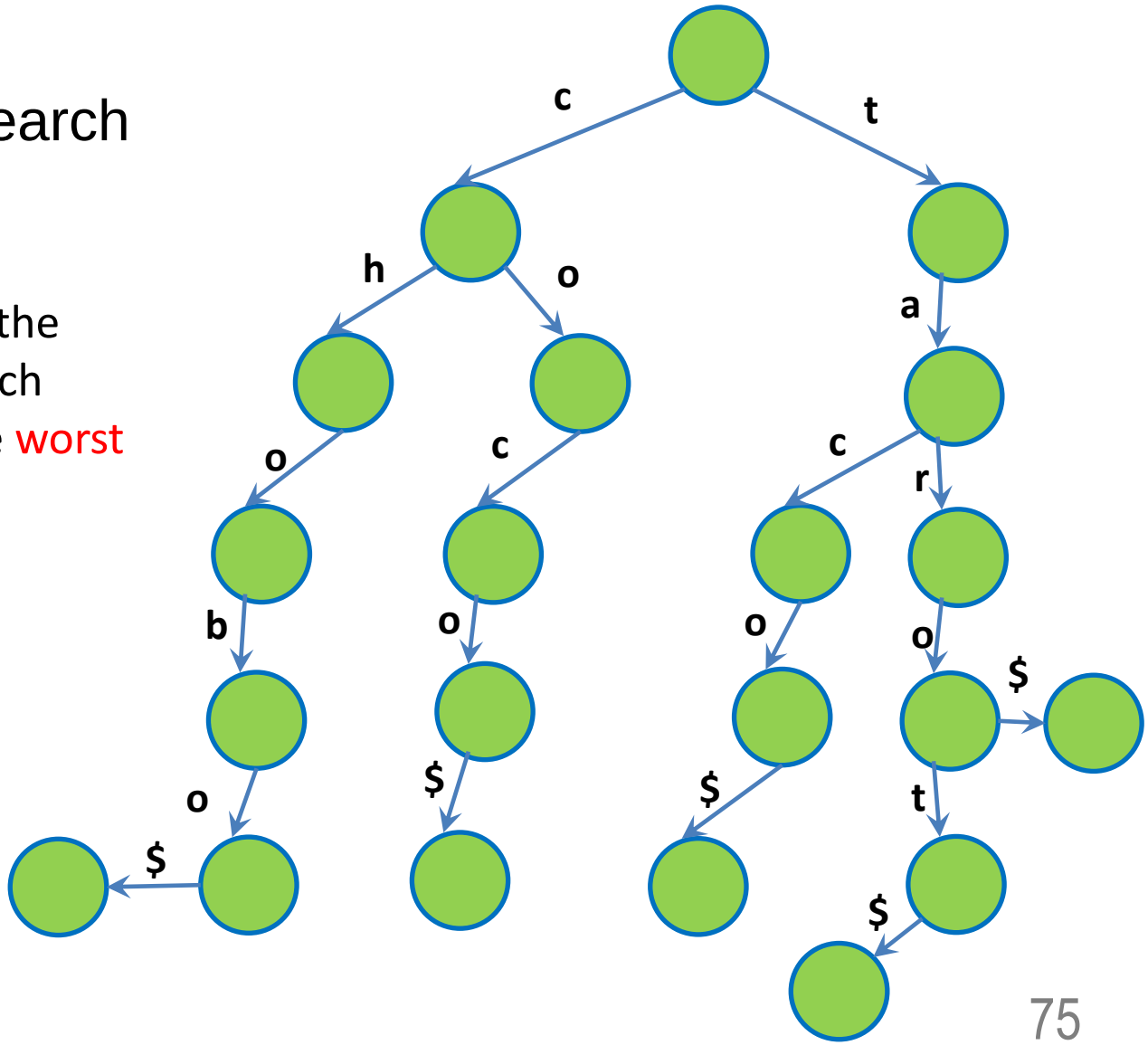
- So how do we search for retrieval?
  - Complexity?
    O(M) where M is the length of the search string…



Tries

Efficient string retrieval

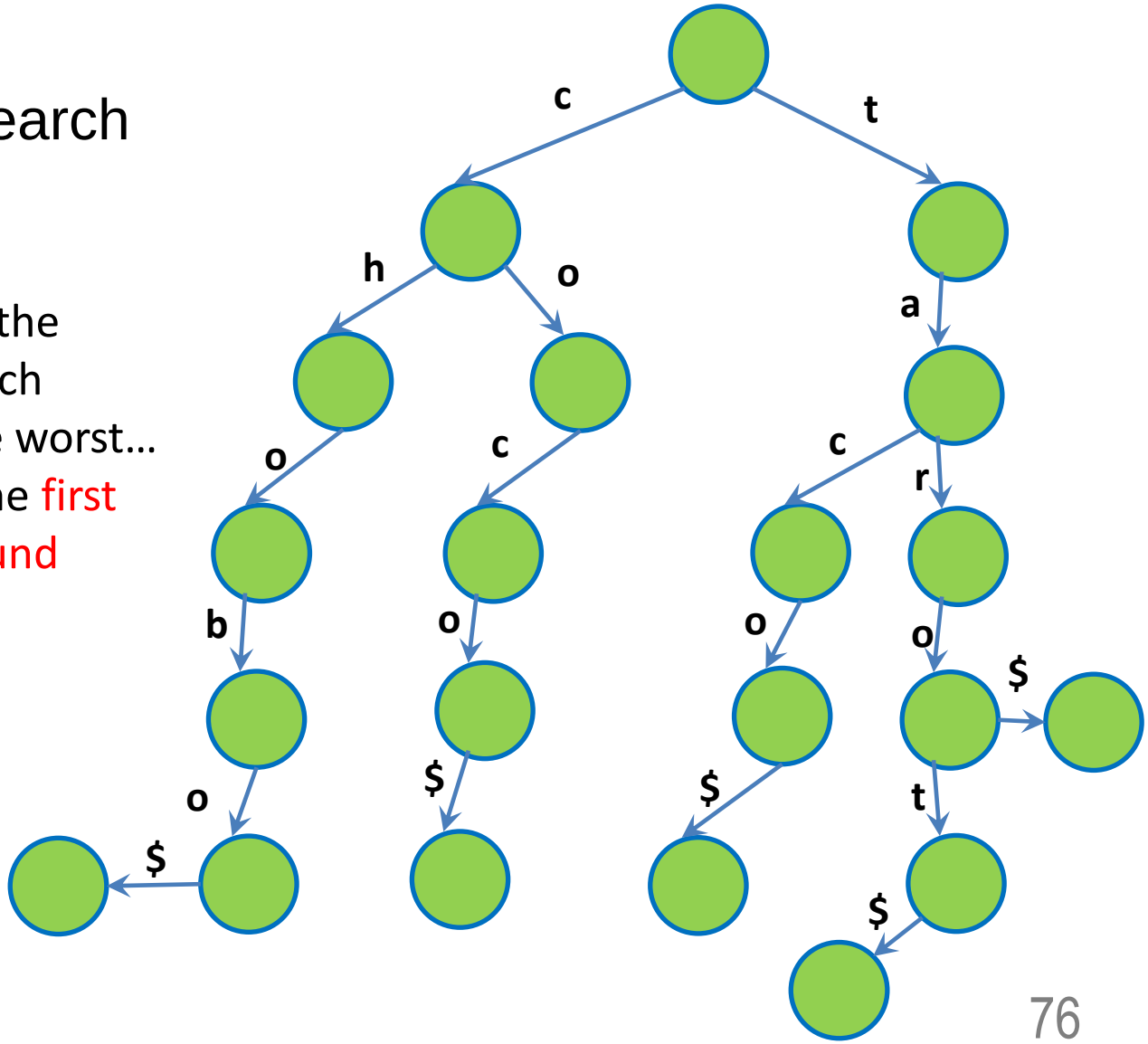
- So how do we search for retrieval?
  - Complexity?
    O(M) where M is the length of the search string… This is the worst



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



MONASH University

Questions?

Tries

Efficient string retrieval

- How to implement it?

Tries

Efficient string retrieval

MONASH University

- How to implement it? With OOP!

Tries

Efficient string retrieval

MONASH University

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

```
1  class Trie:
2      def __init__(self):
3          self.root = Node()
4
5  class Node:
6      def __init__(self, data=None):
7          self.data = data
8          self.links = [None] * 27
```

Tries

Efficient string retrieval

MONASH University

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

```
1  class Trie:
2      def __init__(self):
3          self.root = Node()
4
5  class Node:
6      def __init__(self, data=None):
7          self.data = data
8          self.links = [None] * 27
```

  - 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

MONASH University

Questions?

Tries

Efficient string retrieval

MONASH University

- Benefits?

Tries

Efficient string retrieval

MONASH University

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

Tries

Efficient string retrieval

MONASH University

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

Tries

Efficient string retrieval

MONASH University

- 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)

Tries

Efficient string retrieval

MONASH University

- 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

MONASH University

- 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

MONASH University

Questions?

Tries

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

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
- We can search for the prefix of strings!
  - Useful for auto correct/ auto complete

Tries

Usage?

MONASH University

- 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!

MONASH University

Questions?

Suffix Trie

For suffixes

MONASH University

- Same as a trie
- But for suffixes

Suffix Trie

For suffixes

- Can you make a suffix trie for apple?

Suffix Trie

For suffixes

MONASH University

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

Suffix Trie

For suffixes

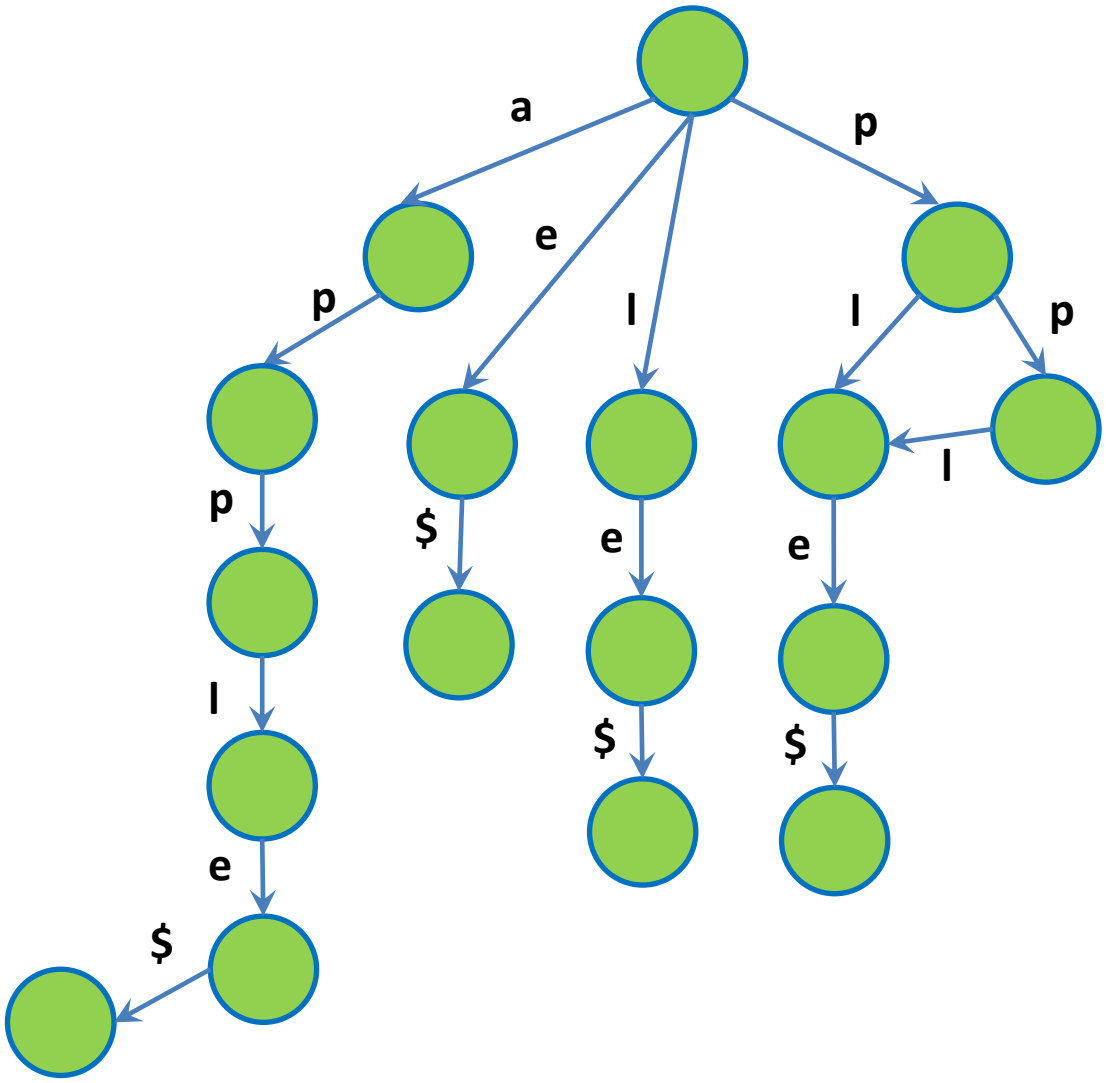
MONASH University

- 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

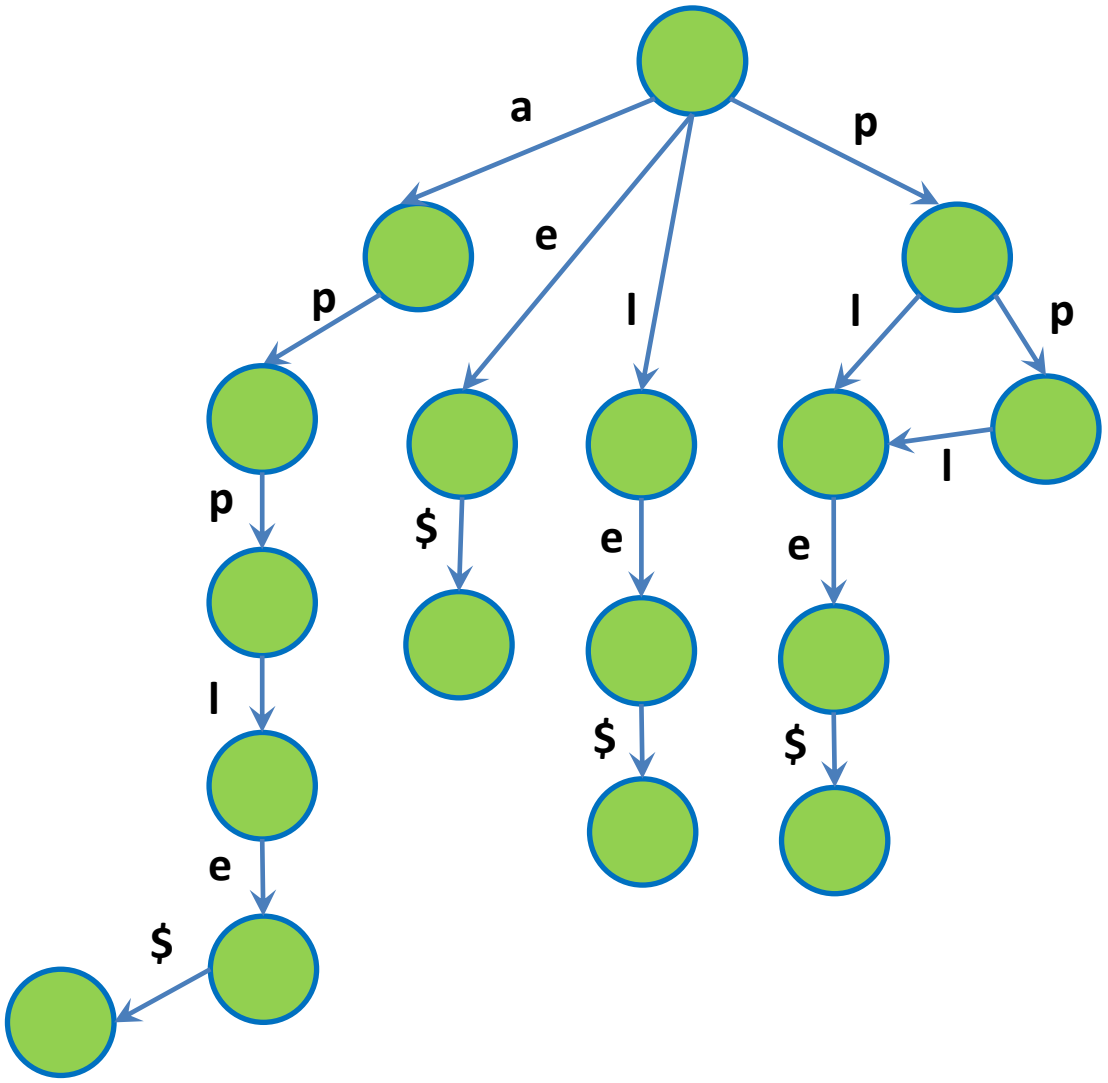
- Can you make a suffix trie for apple?
- List all the suffixes
  - Apple$
  - Pple$
  - Ple$
  - Le$
  - E$
- Then we just make the trie like earlier
- Is this right?



Suffix Trie

For suffixes

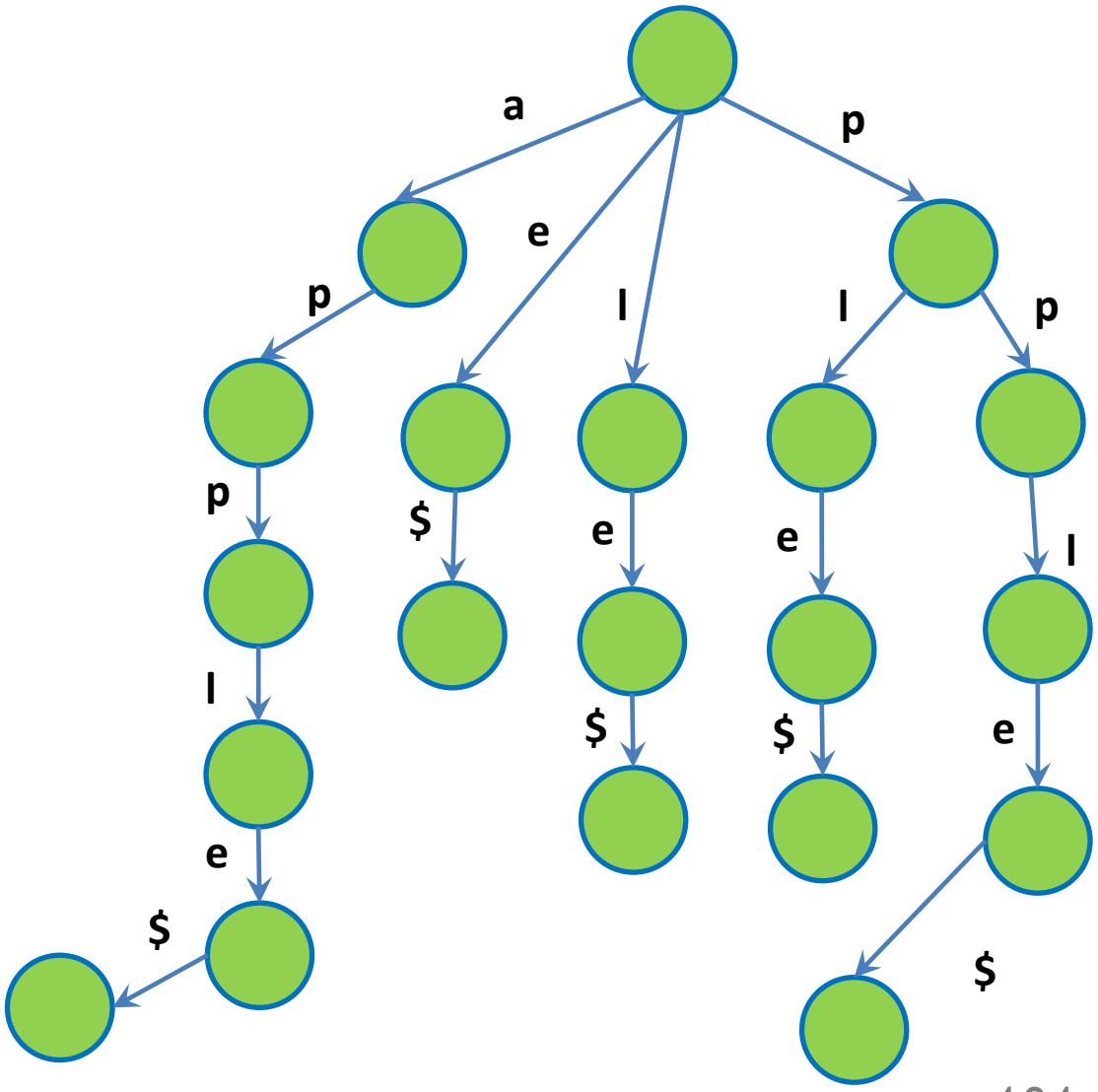
- Can you make a suffix trie for apple?
- List all the suffixes
  - Apple$
  - Pple$
  - Ple$
  - Le$
  - E$
- Then we just make the trie like earlier
- Is this right?
  NO! CYCLE!
  so this is wrong…



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



MONASH University

Questions?

Suffix Trie

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

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!

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

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?

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

Suffix Trie

Applications?

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

Suffix Trie

Applications?

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…

MONASH University

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!

MONASH University

Questions?

Suffix Tree

A tree, not a trie

MONASH University

- What is a suffix tree?

Suffix Tree

A tree, not a trie

MONASH University

- What is a suffix tree?

Suffix Tree

A tree, not a trie

MONASH University

- What is a suffix tree?

Suffix Trie

Suffix Tree

Suffix Tree

A tree, not a trie

MONASH University

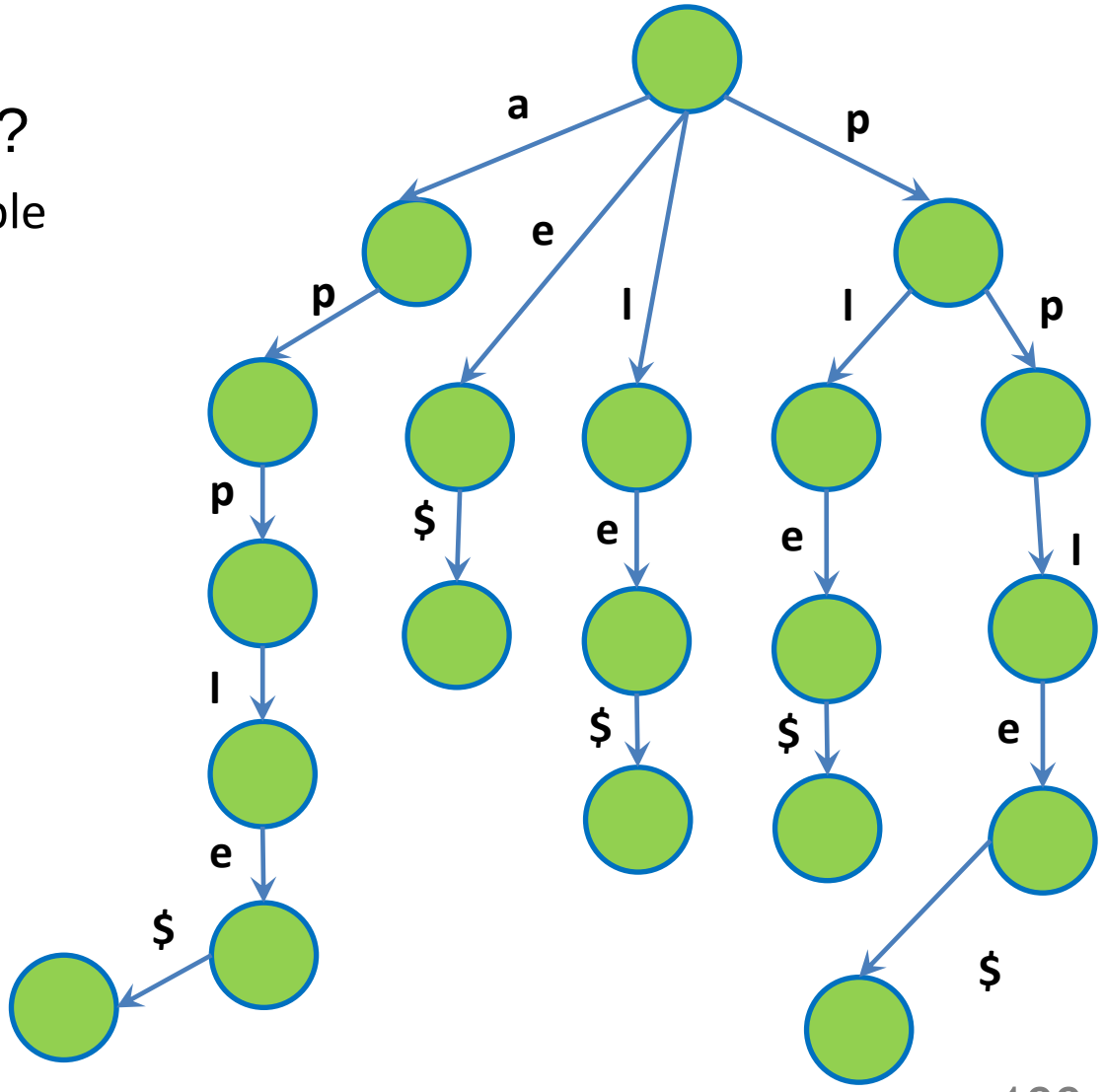
- What is a suffix tree?
  - Using our same example

Suffix Tree

A tree, not a trie

MONASH University

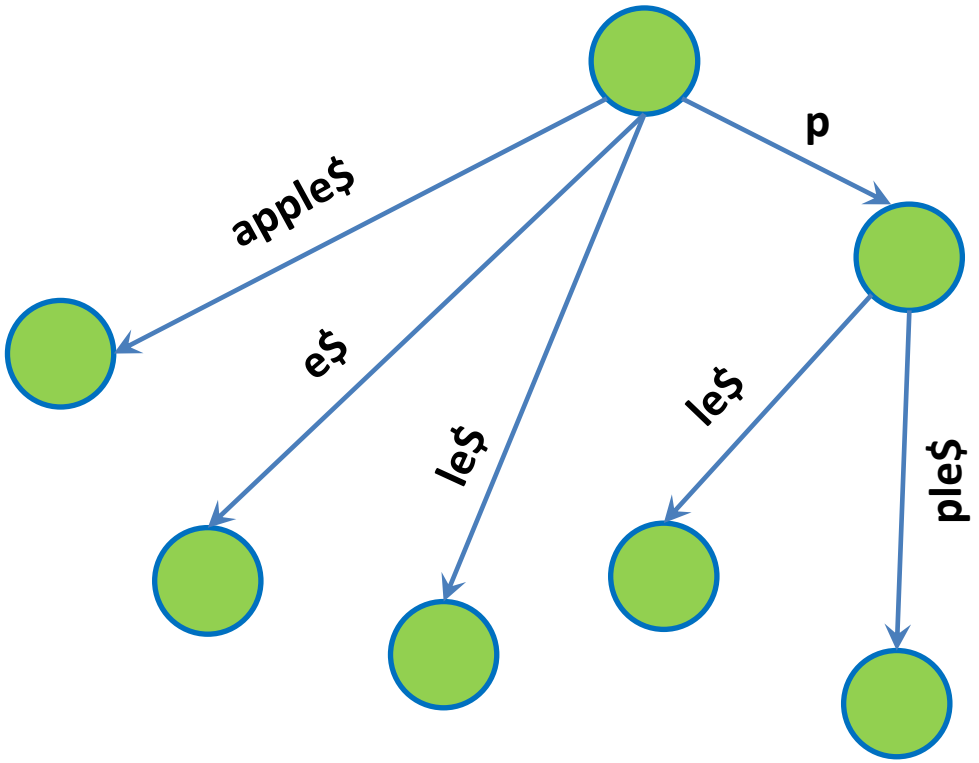
- What is a suffix tree?
  - Using our same example



Suffix Tree

A tree, not a trie

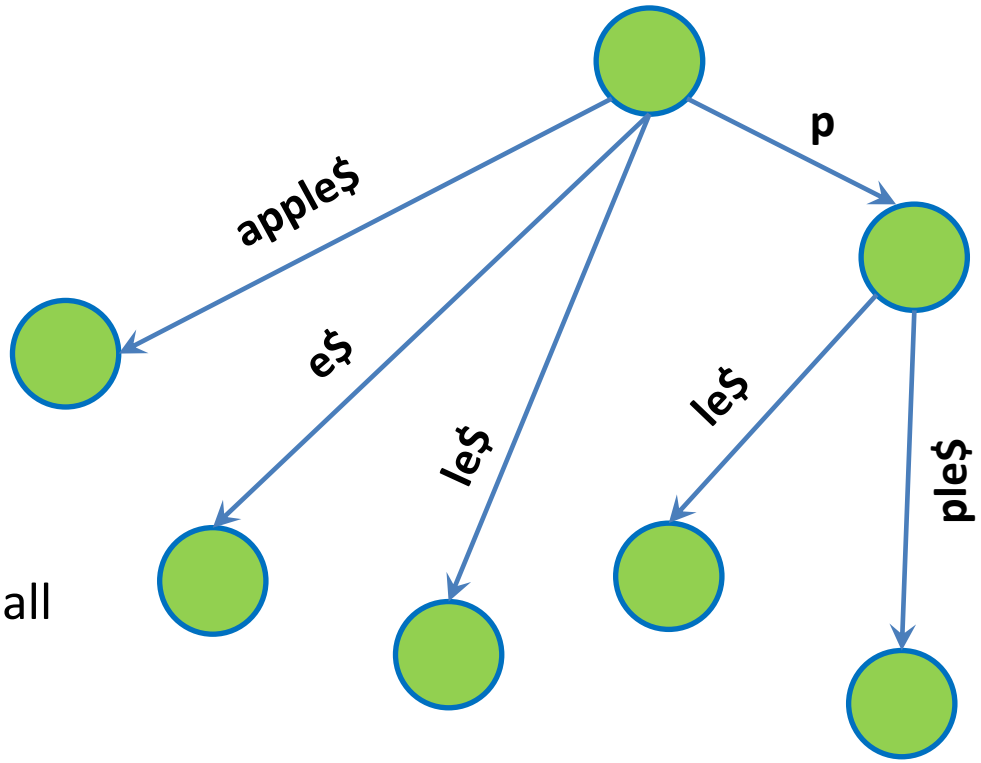
- What is a suffix tree?
  - Using our same example
- What is our space complexity?



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

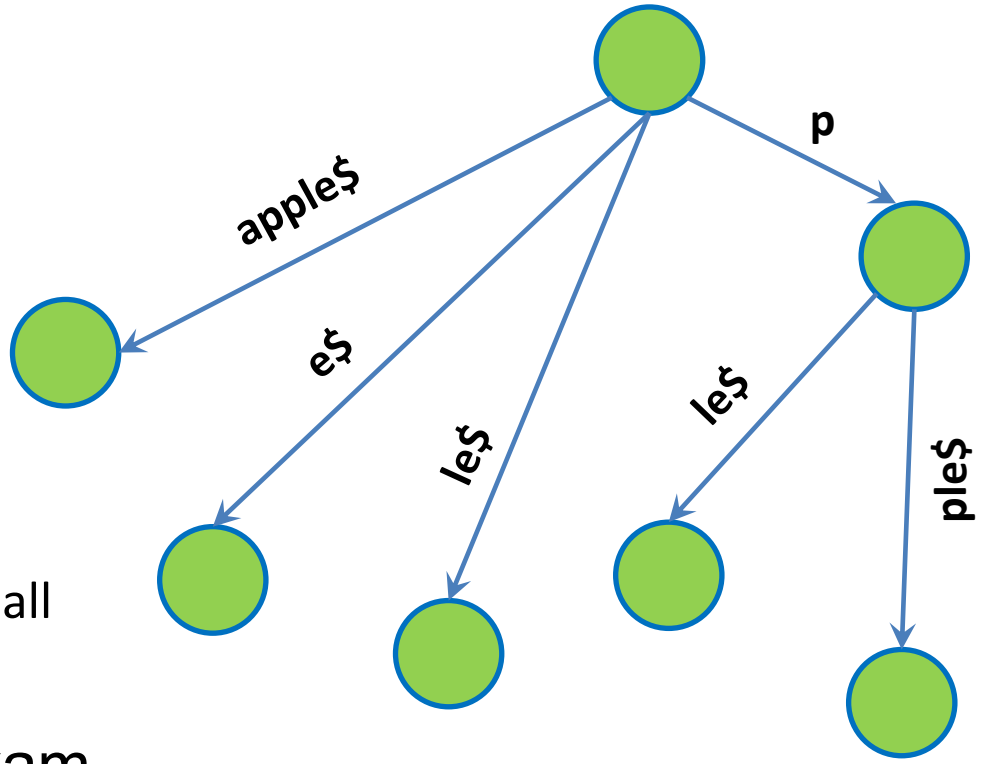


Suffix Tree

A tree, not a trie

MONASH University

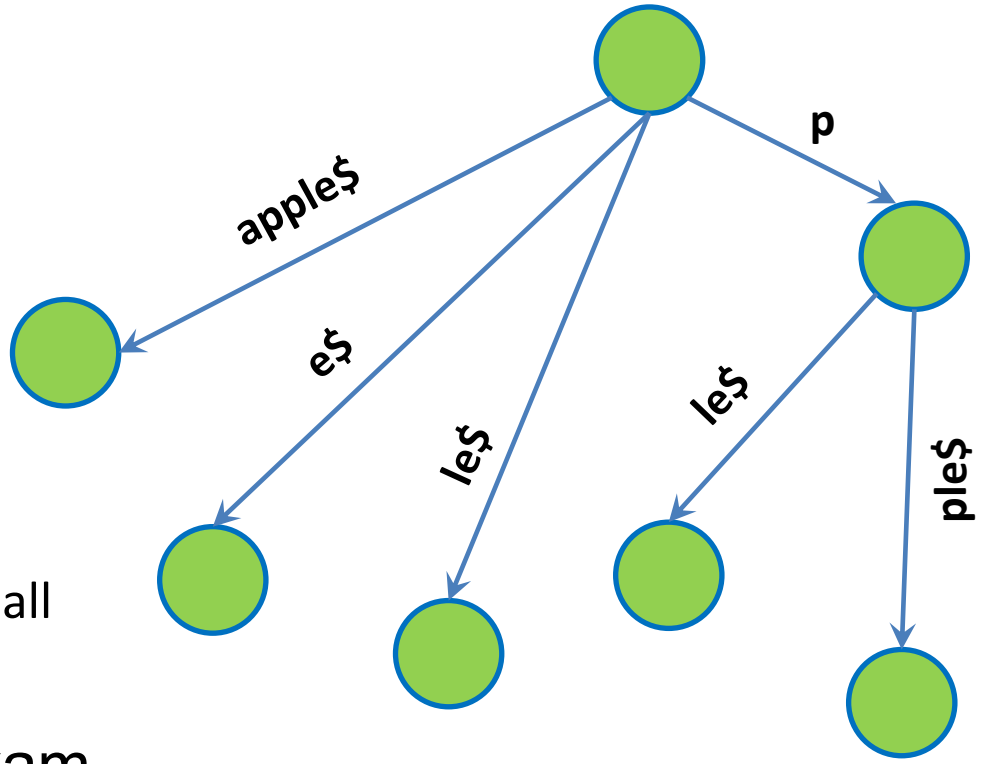
- What is a suffix tree?
  - Using our same example
- What is our space complexity?
  - O(N^2) still because we still store the characters all
- When asked in the exam…
  - Draw a suffix trie
  - Then compress to suffix tree



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



MONASH University

Questions?

Suffix Tree

A tree, not a trie

MONASH University

- Space complexity O(N^2)

Suffix Tree

A tree, not a trie

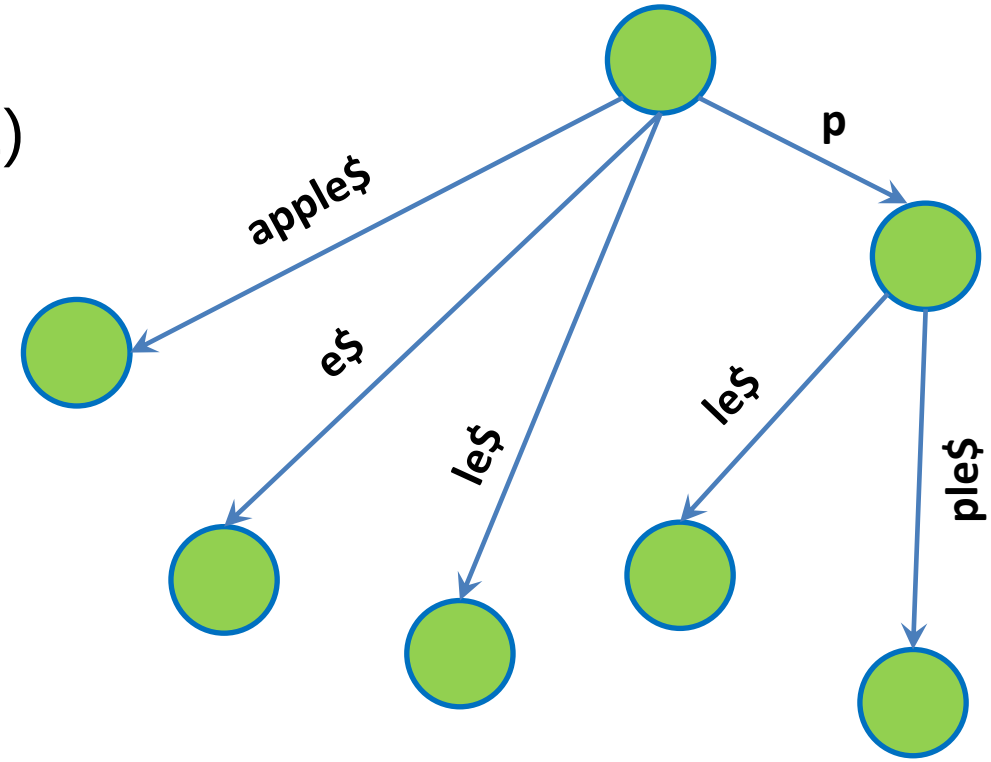
MONASH University

- 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?

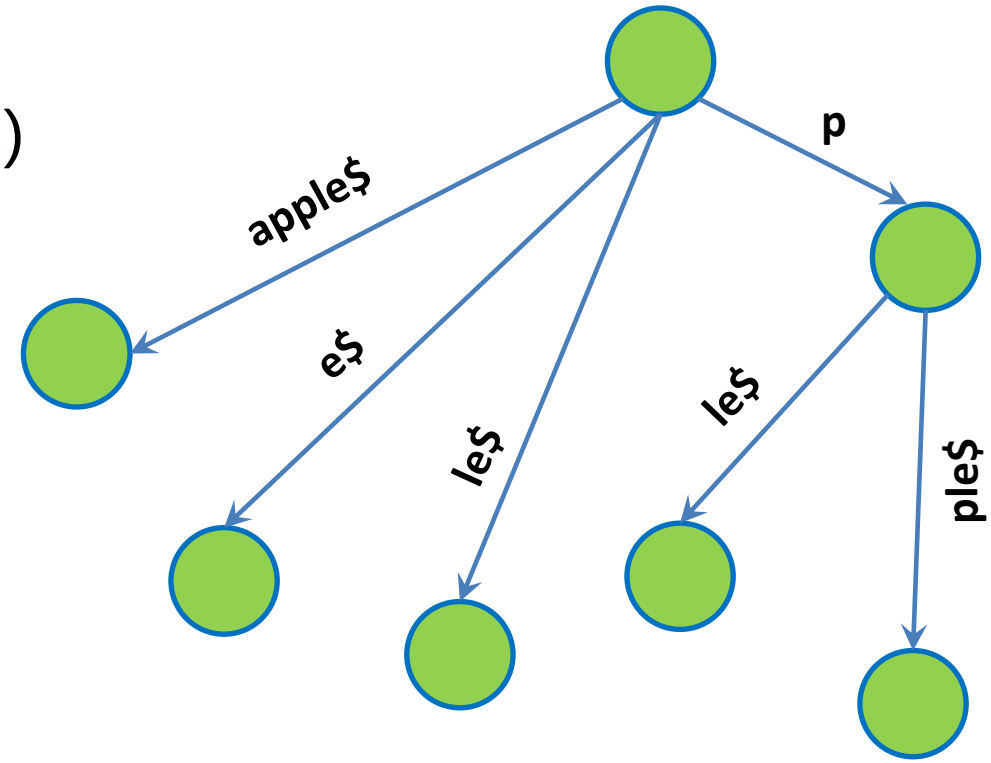


Suffix Tree

A tree, not a trie

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

- Our string is apple$

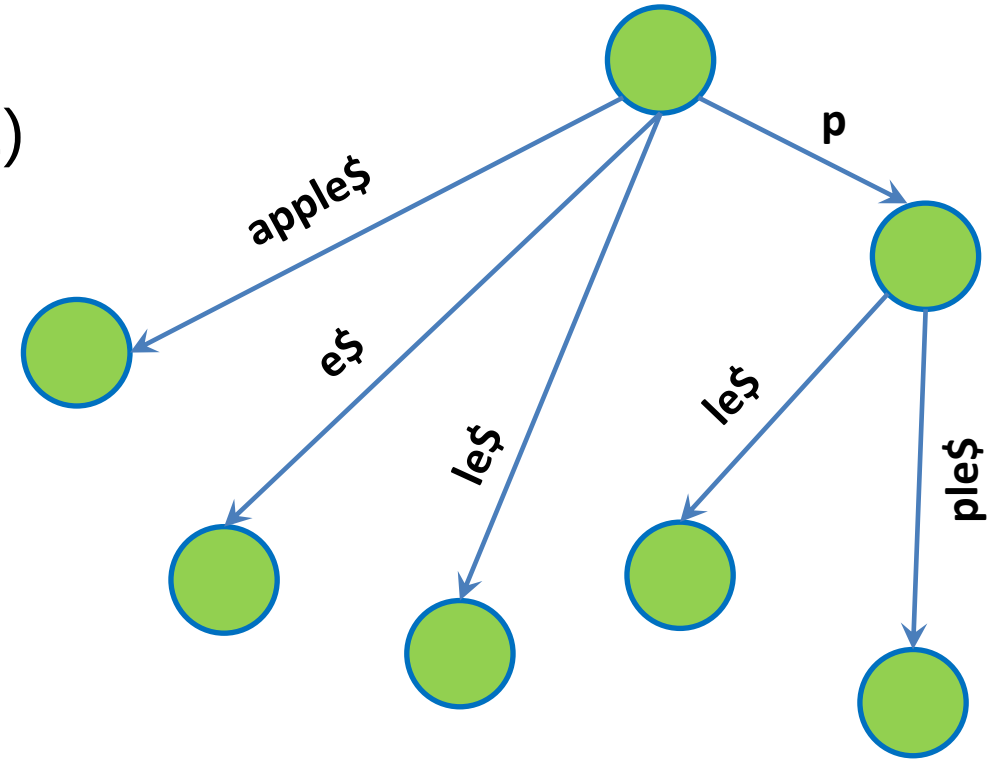


Suffix Tree

A tree, not a trie

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

- Our string is apple$



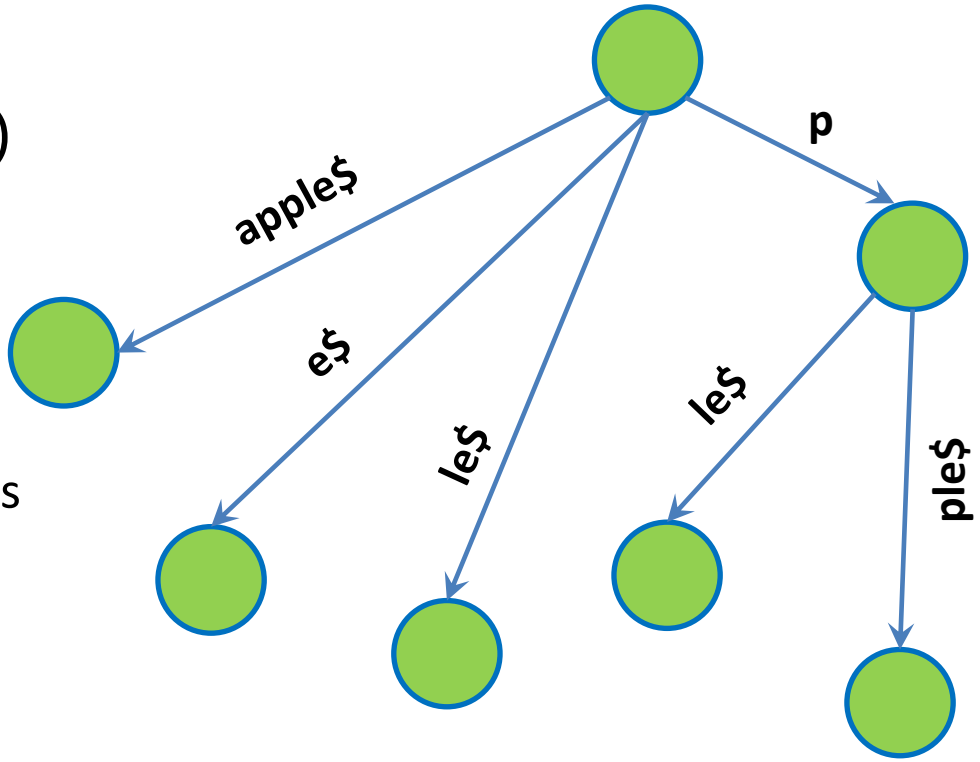
a	p	p	l	e	$
0	1	2	3	4	5

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!



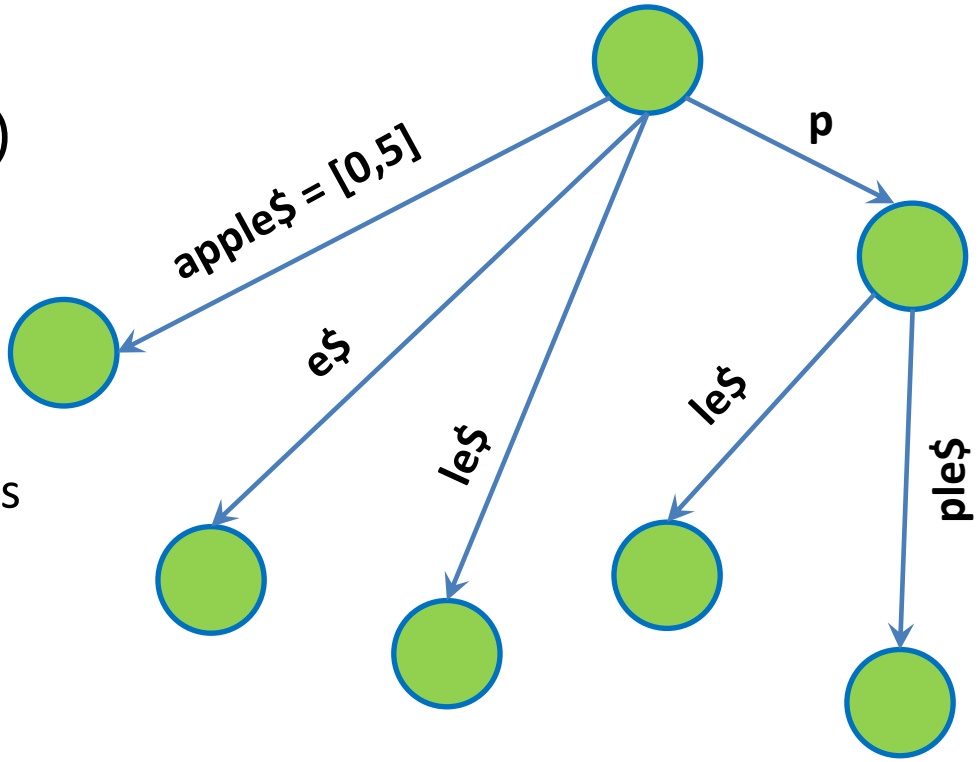
a	p	p	l	e	$
0	1	2	3	4	5

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!



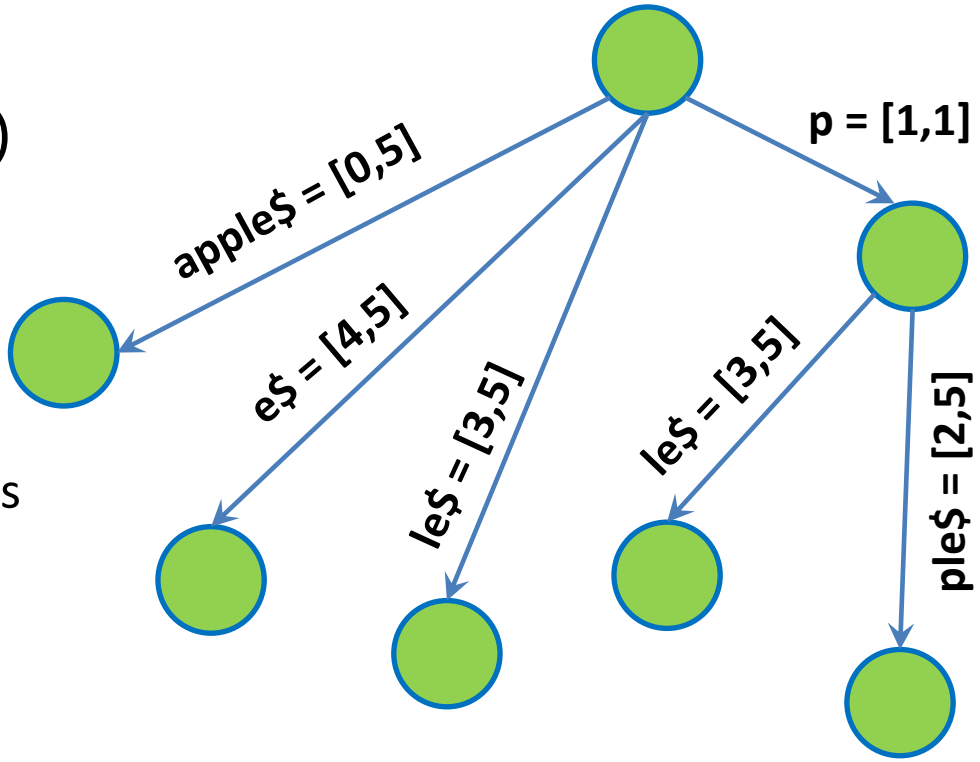
a	p	p	l	e	$
0	1	2	3	4	5

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!



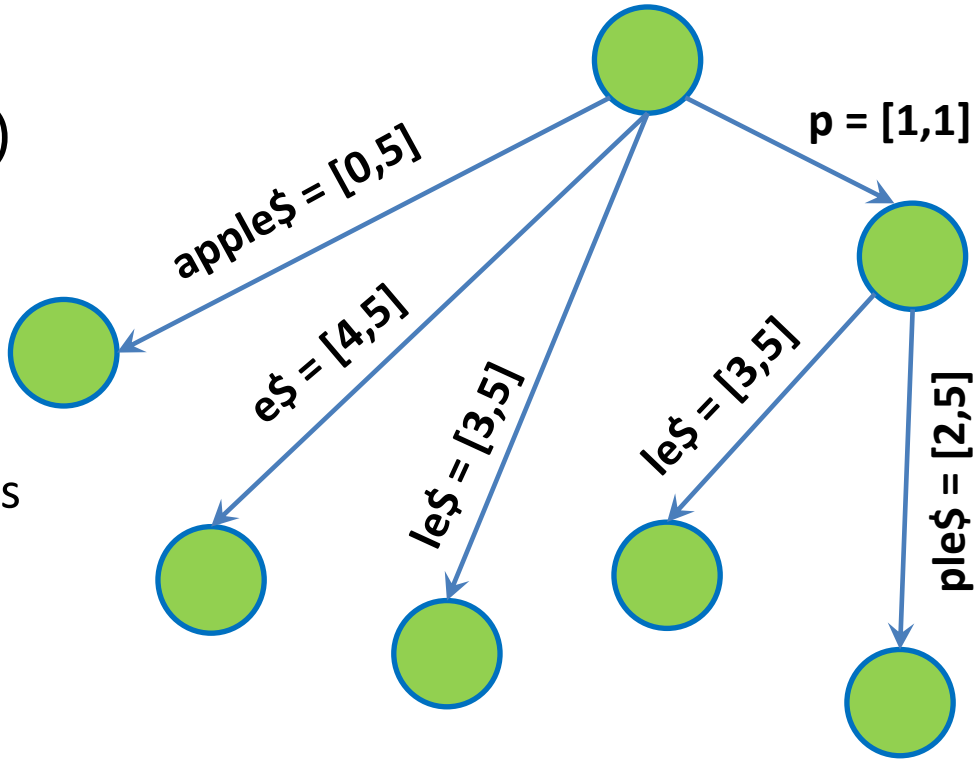
a	p	p	l	e	$
0	1	2	3	4	5

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]



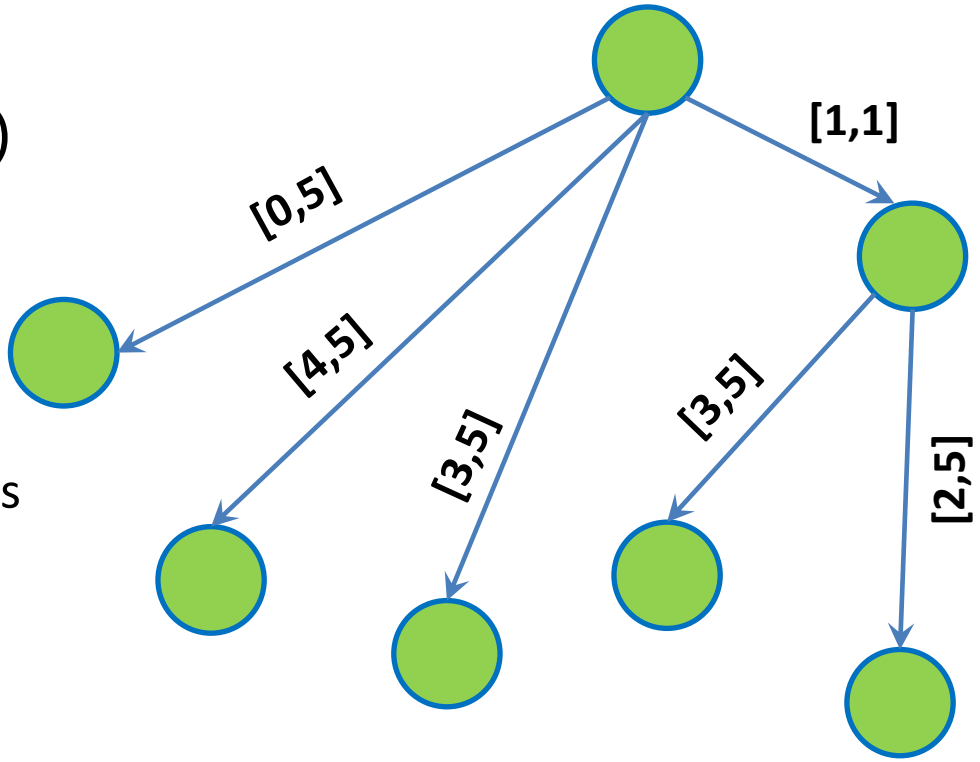
a	p	p	l	e	$
0	1	2	3	4	5

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

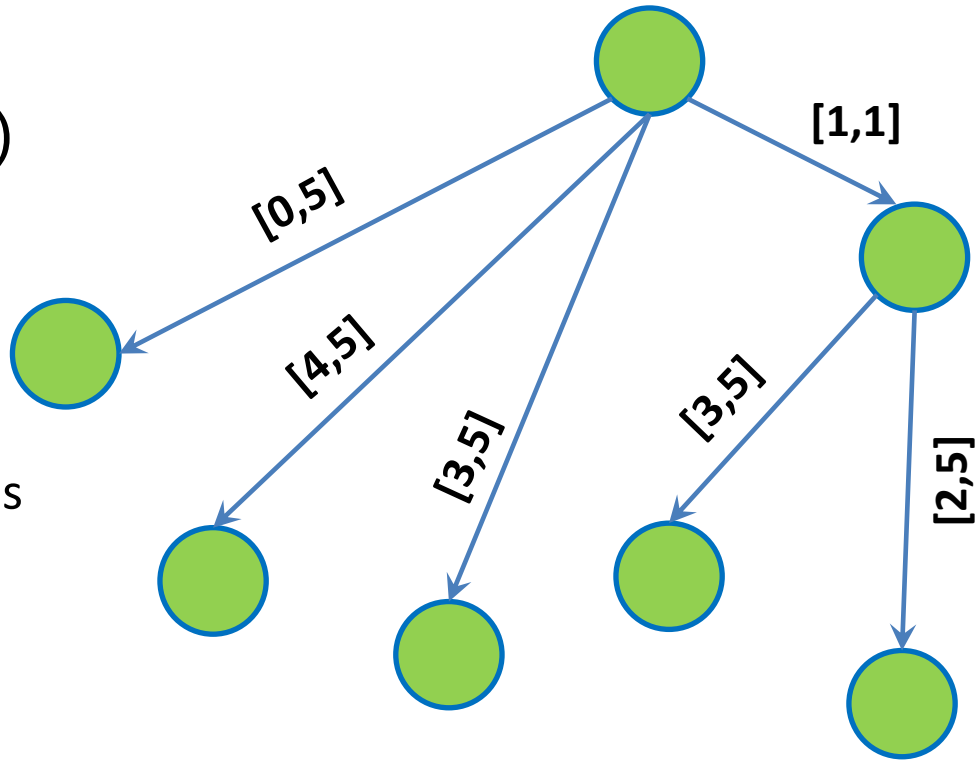
Suffix Tree

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

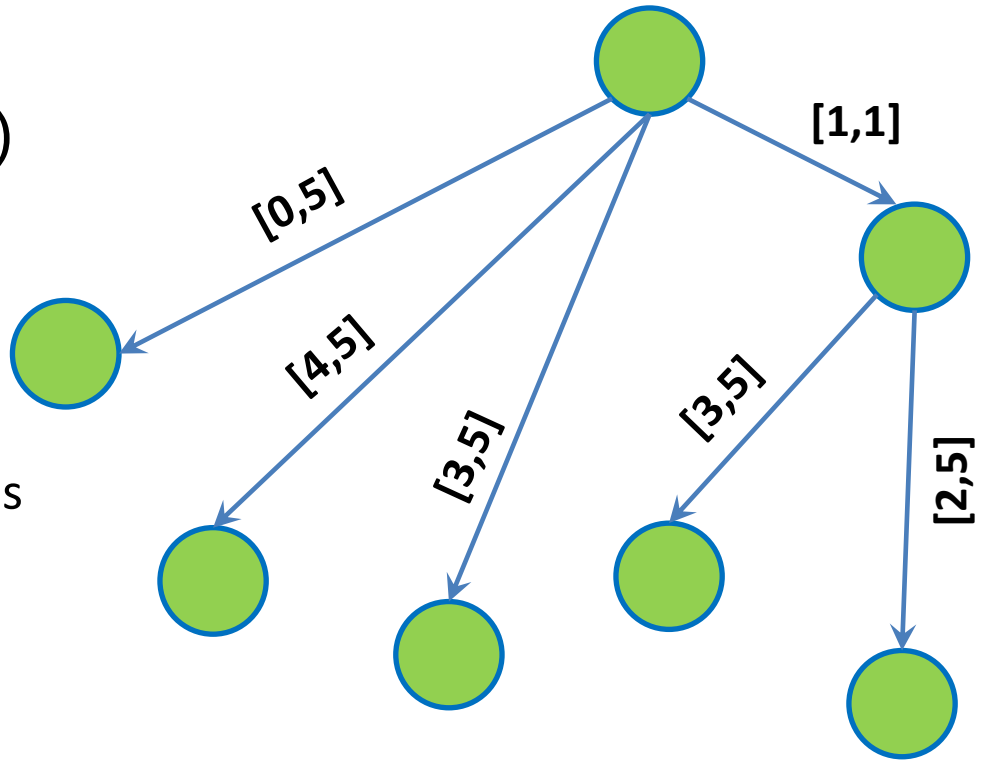
A tree, not a trie

MONASH University

- 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)



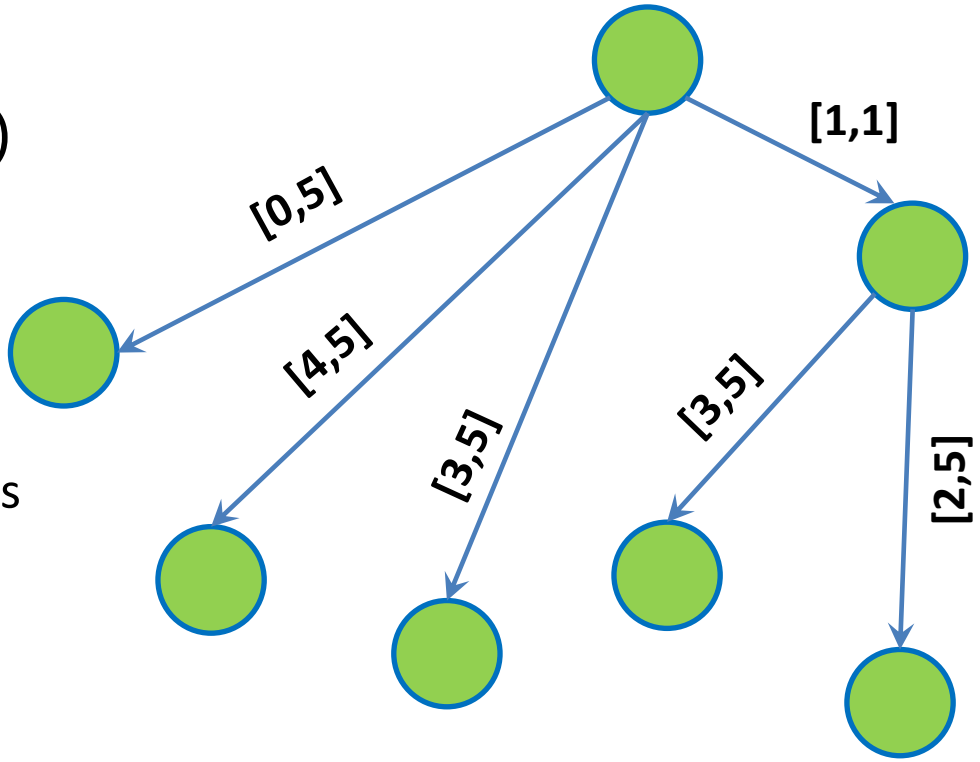
a	p	p	l	e	$
0	1	2	3	4	5

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]



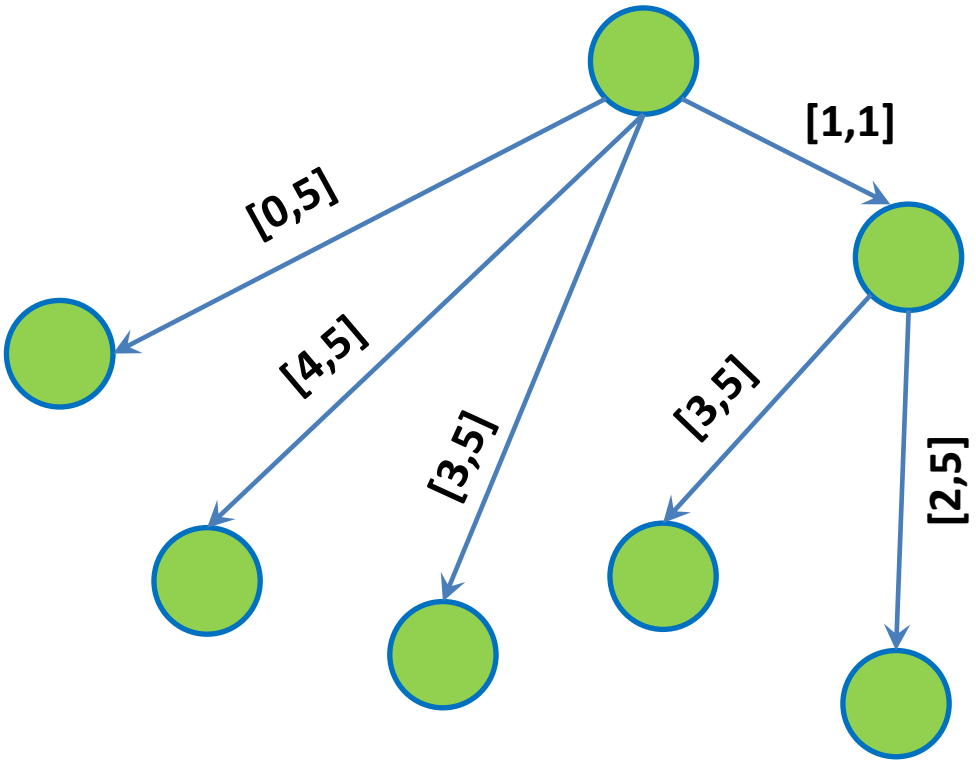
a	p	p	l	e	$
0	1	2	3	4	5

- 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)



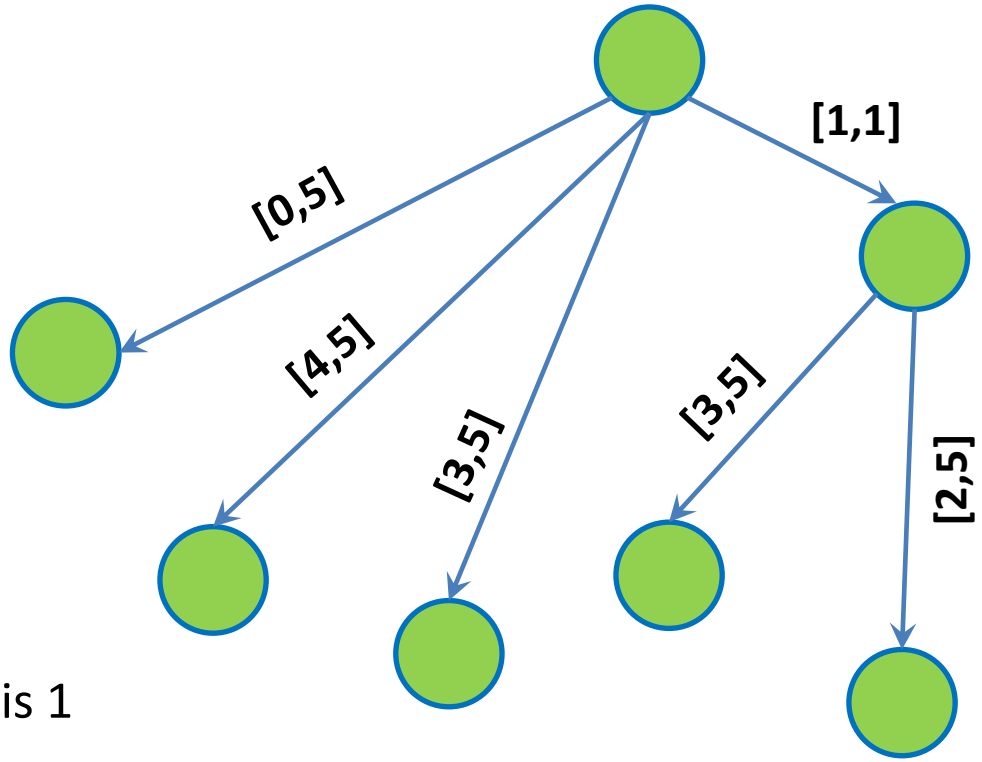
a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

MONASH University

- 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



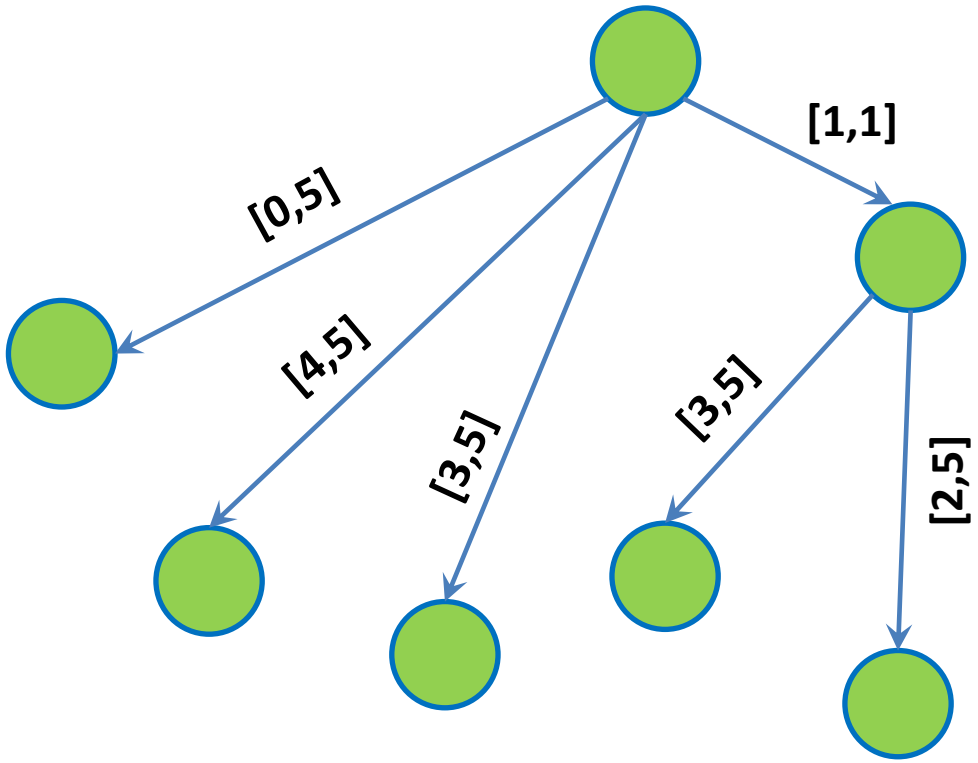
a	p	p	l	e	$
0	1	2	3	4	5

Suffix Tree

A tree, not a trie

MONASH University

- 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

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

Suffix Tree

A tree, not a trie

MONASH University

- 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)

MONASH University

Questions?

MONASH University

- Let us try to implement it!

MONASH University

- Let us try to implement it!

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

MONASH University

- 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

MONASH University

- 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)

MONASH University

Questions?

MONASH University

Thank You