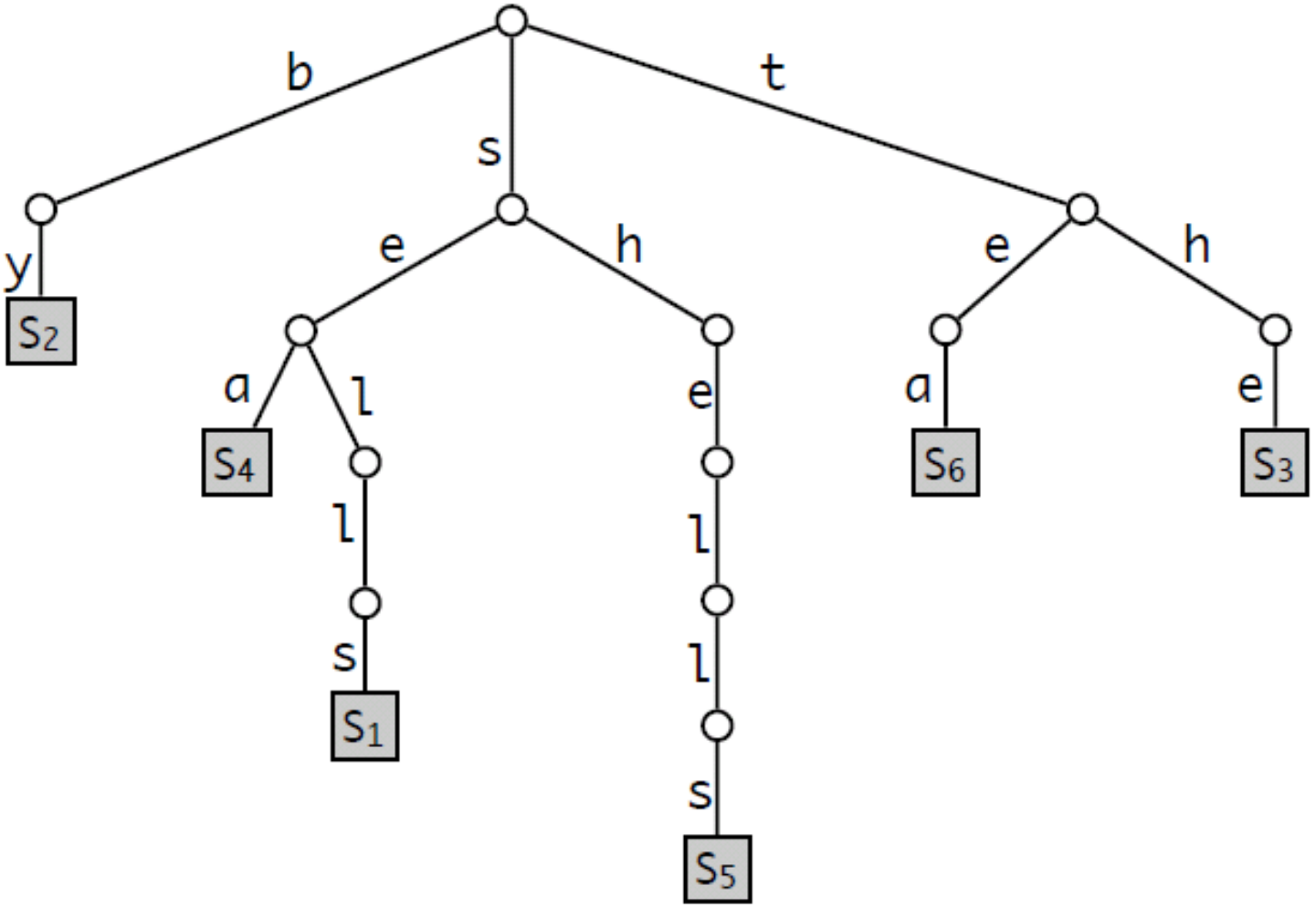
Data Structures for Strings

String indexing problem

- String matching problem. Given strings T (text) and P (pattern) over an alphabet U, report starting positions of all occurrences of P in T.
  - Finite automaton: $O(m\Sigma + n)$ time and space
  - KMP: $O(m + n)$ time and space
- String indexing problem. Given a string S of characters from an alphabet Σ. Preprocess S into a data structure to support
  - Search(P): Return starting position of all occurrences of P in S.
- Today: Data structure using $O(n)$ space and supporting Search(P) in $O(m)$ time.
- Applications:
  - Search engines, e.g. prefix searches.
  - Finding common substrings of many biological strings
  - Finding repeating substructures in biological strings
  - Detecting DNA contamination

Tries

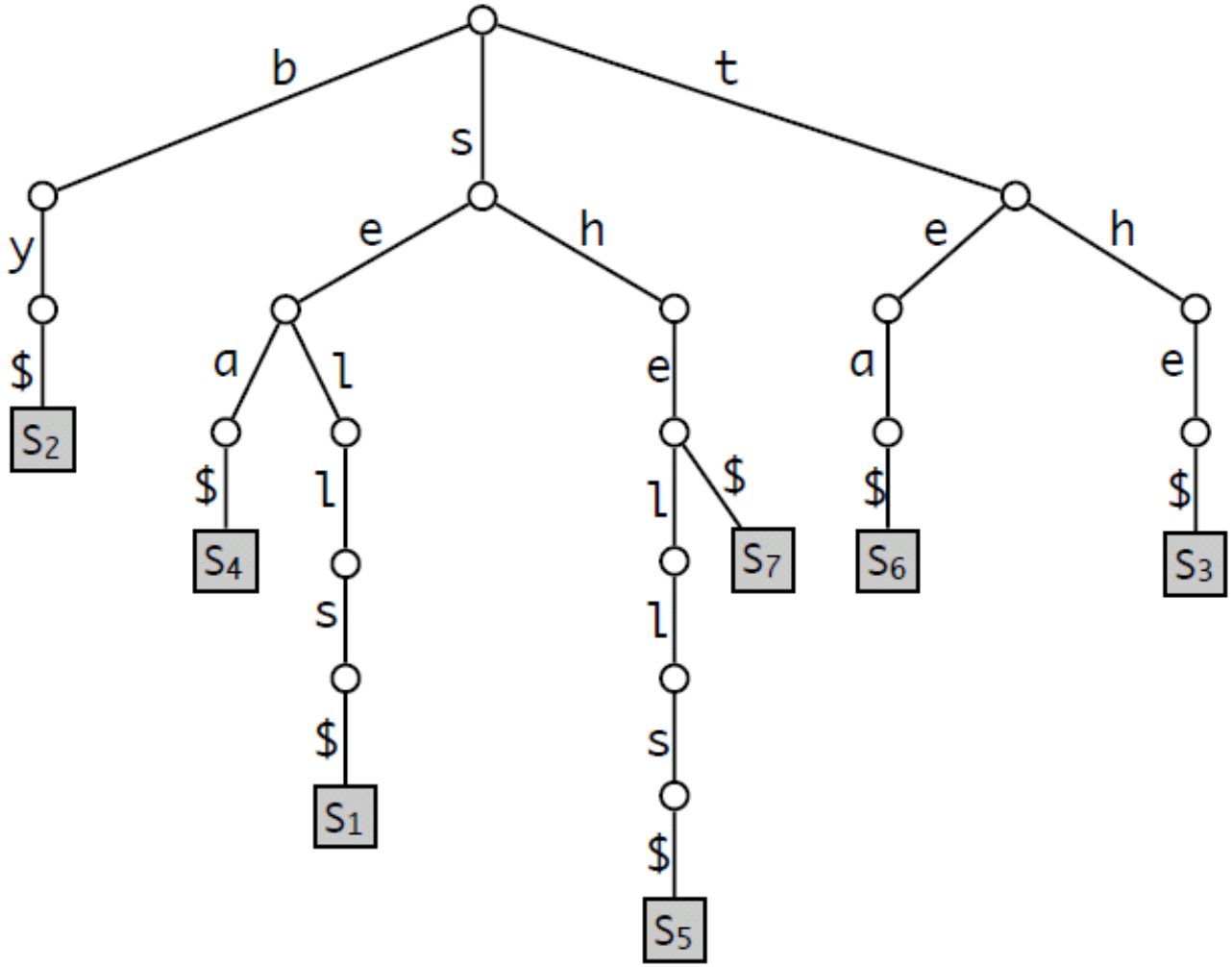
Text retrieval



Trie over the strings: sells, by, the, sea, shells, tea.

Tries

Prefix-free?



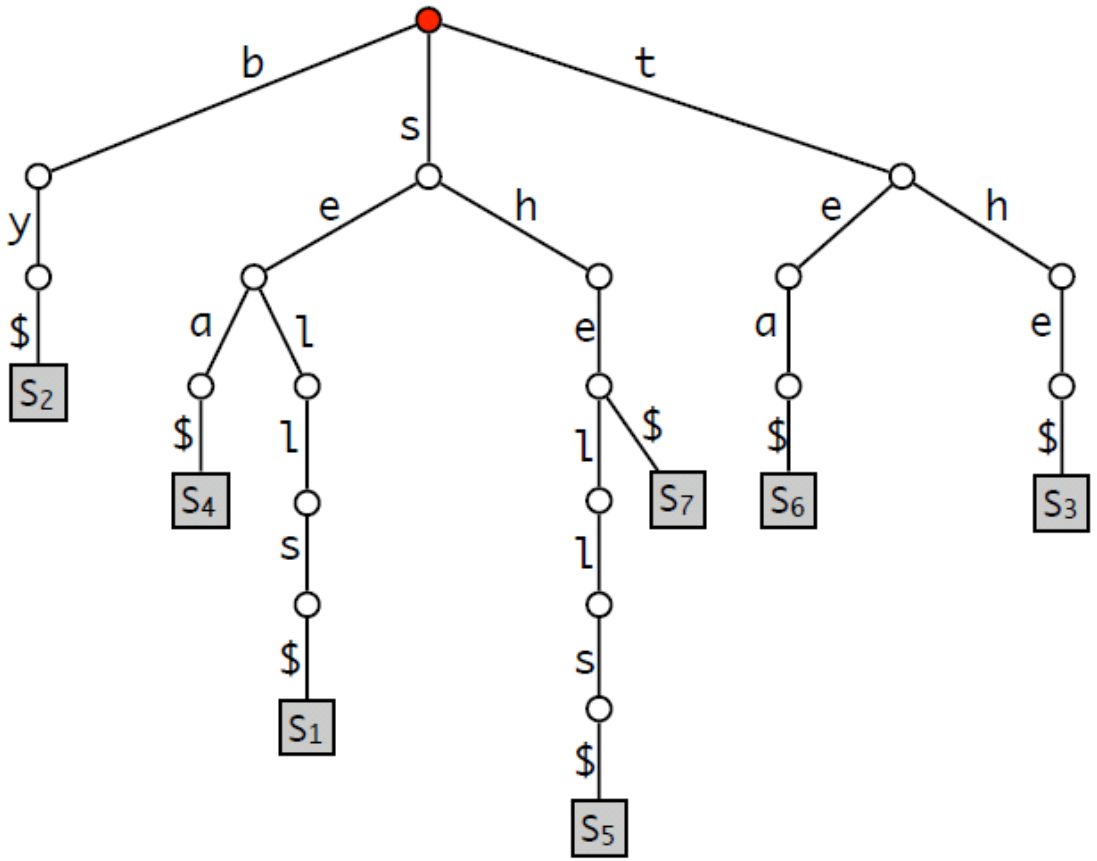
Trie over the strings: sells, by, the, sea, shells, tea, she.

Tries

- Properties of the trie. A trie T storing a collection S of s strings of total length n from an alphabet of size d has the following properties:
  - How many children can a node have?
  - How many leaves does T have?
  - What is the height of T?
  - What is the number of nodes in T?
- Search time: $O(d)$ in each node $\Rightarrow O(dm)$.
- d not constant: use dictionary
  - Hashing O(1)
  - Balanced BST: $O(logd)$
- Time and space for a trie (for small/constant d):
  - $O(m)$ for searching for a string of length m.
  - $O(n)$ space
  - Preprocessing: $O(n)$

Tries

Prefix search: return all words in the trie starting with se.

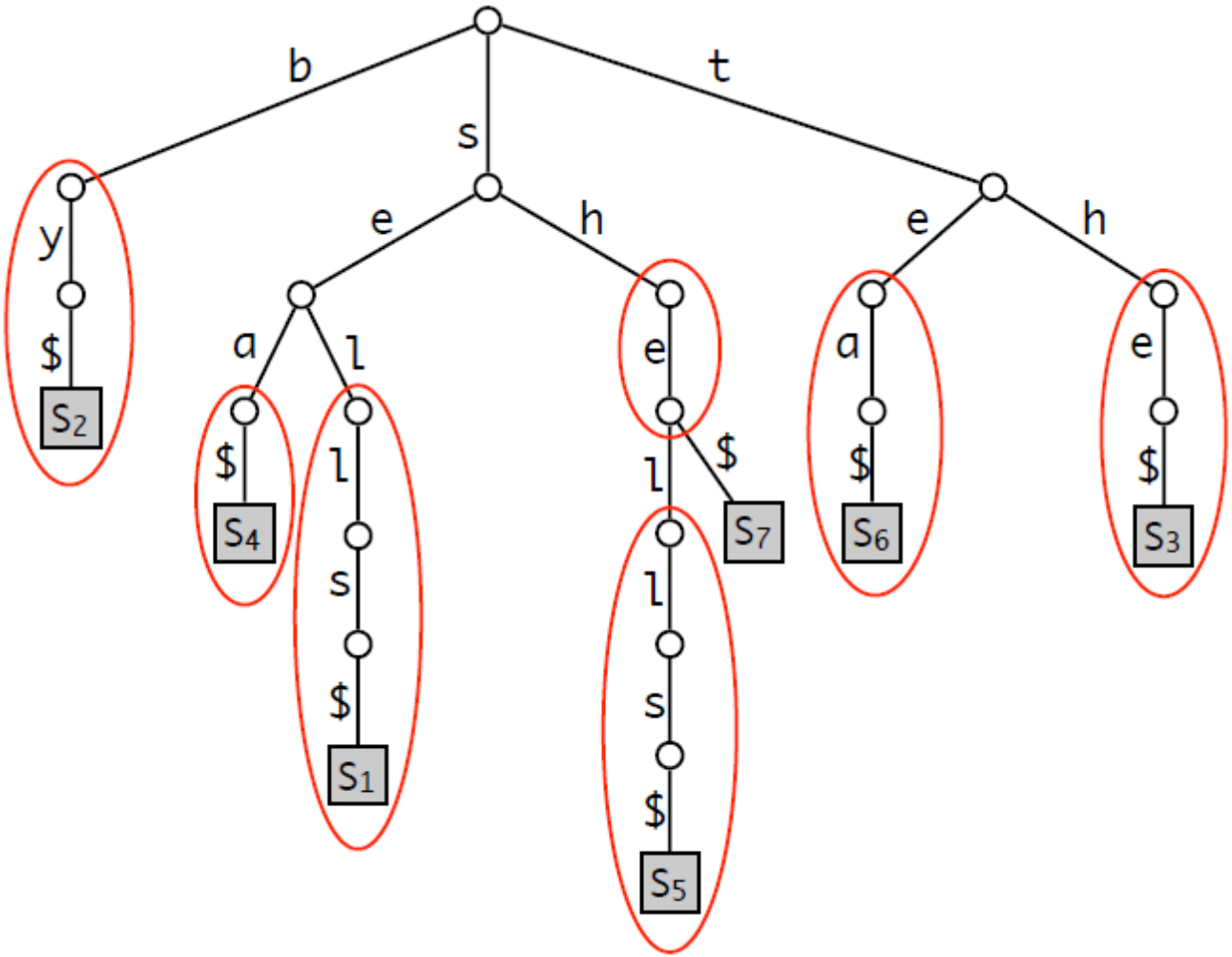


Time for prefix search: $O(m)$ + time to report all occurrences. $\Leftarrow$ Could be large!

Solution: compact tries.

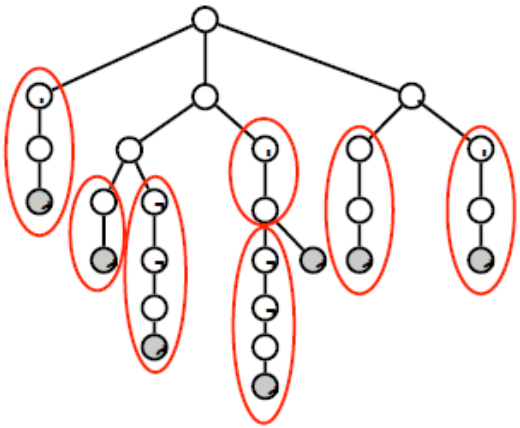
Compact Tries

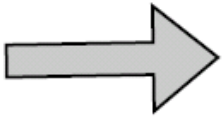
Compact trie: Chains of nodes with a single child is merged into a single node.

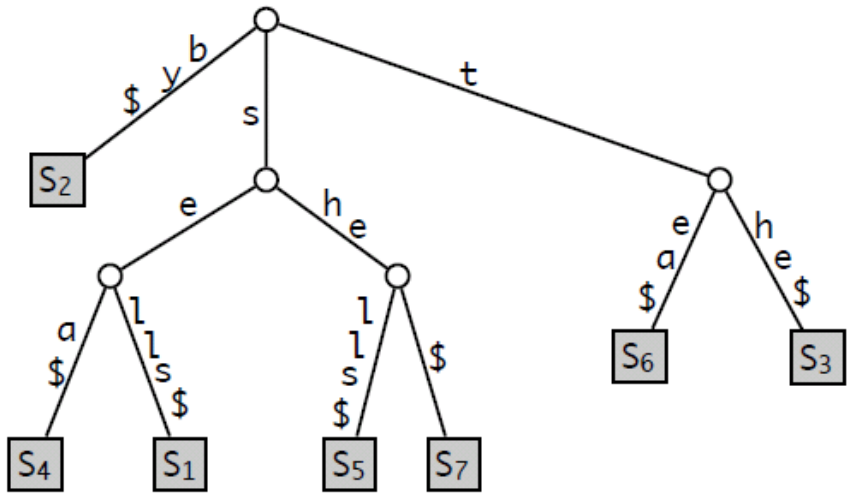


Compact Tries

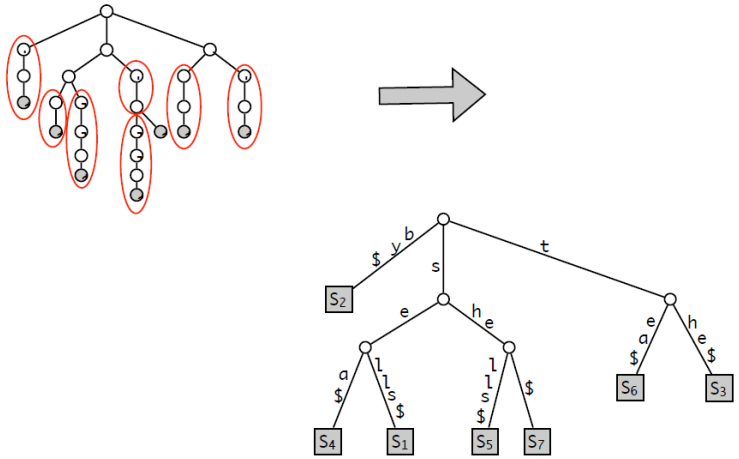
Compact trie: Chains of nodes with a single child is merged into a single node.







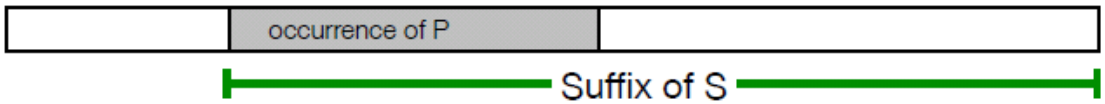
Properties of Compact Tries



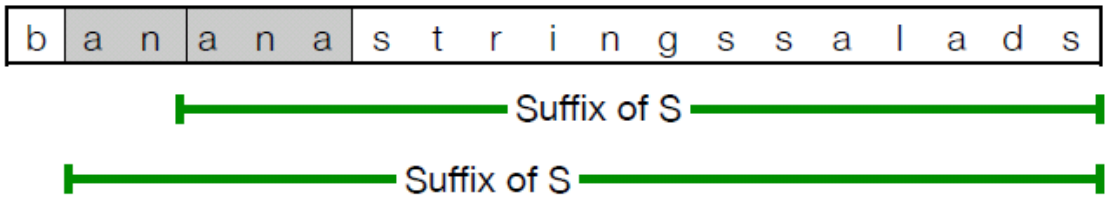
- Properties of the compact trie. A compact trie T storing a collection S of s strings of total length n from an alphabet of size d has the following properties:
  - Every internal node of T has at least 2 and at most d children.
  - T has s leaves
  - The number of nodes in T is $< 2s$.
- Time and space for a compact trie (constant d):
  - $O(m)$ for searching for a string of length m.
  - $O(m + occ)$ for prefix search, where occ = #occurrences
  - $O(s)$ space.
  - Preprocessing: $O(n)$

Suffix trees

- String indexing problem. Given a string S of characters from an alphabet U. Preprocess S into a data structure to support. Search(P): Return starting position of all occurrences of P in S.
- Build a compressed trie over all suffixes of S (suffix tree). Label leaves with index of suffix.
- Observation: An occurrence of P is a prefix of a suffix of S.

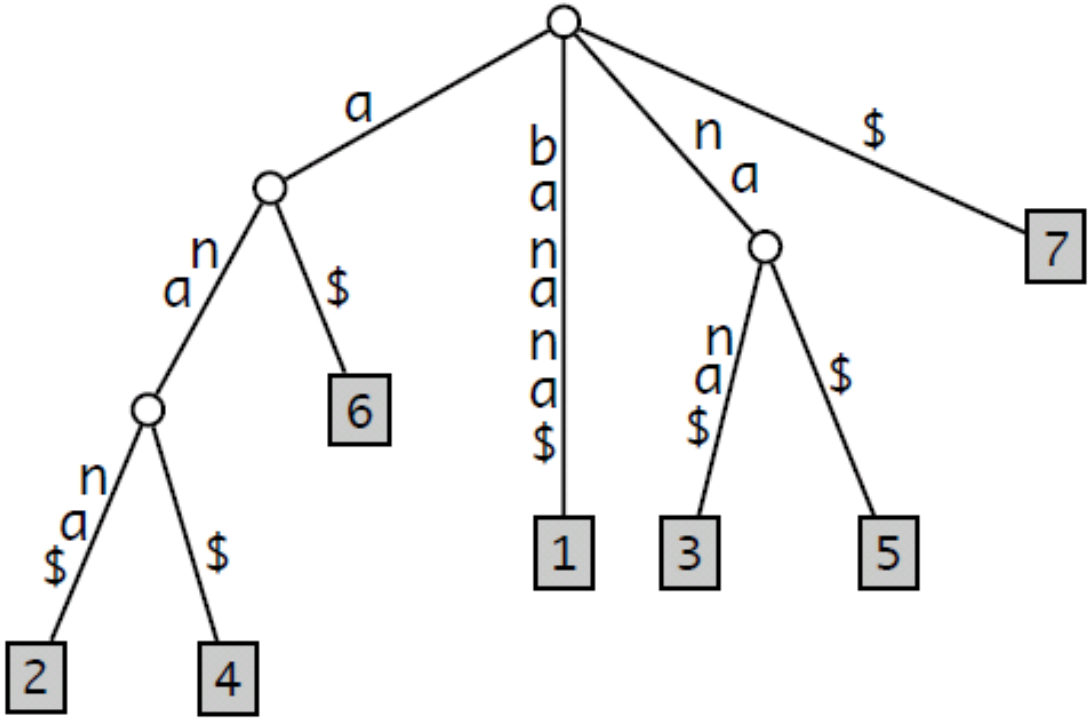


- Example: P = ana.



Suffix trees

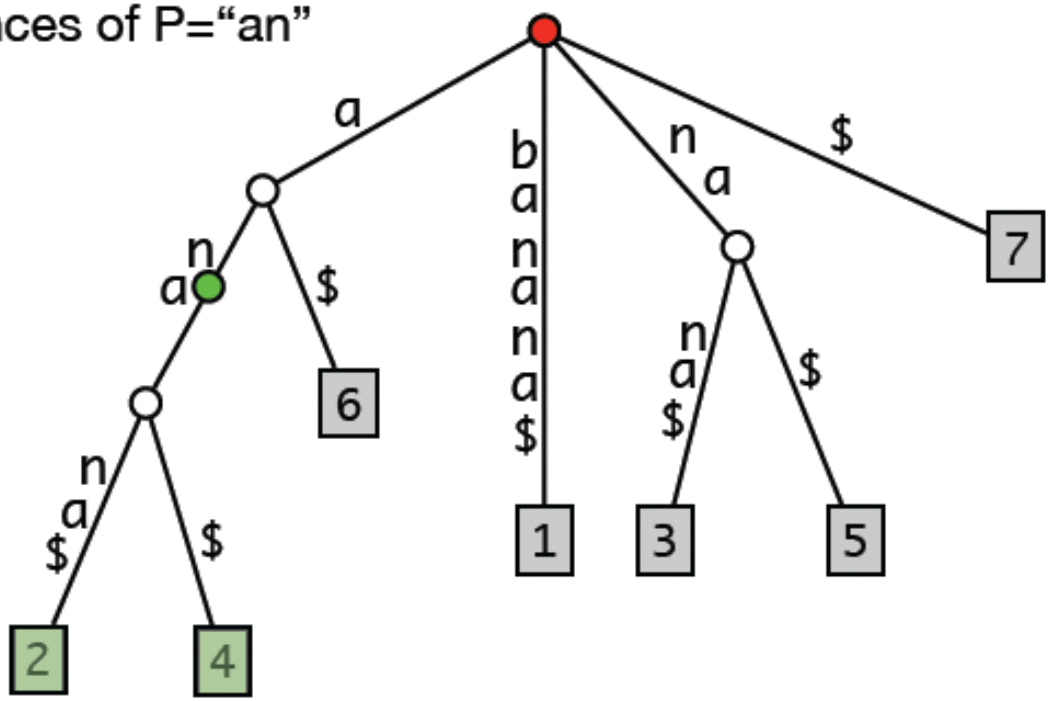
Suffix tree: over the string banana$



Find all occurrences of P=an

Suffix trees

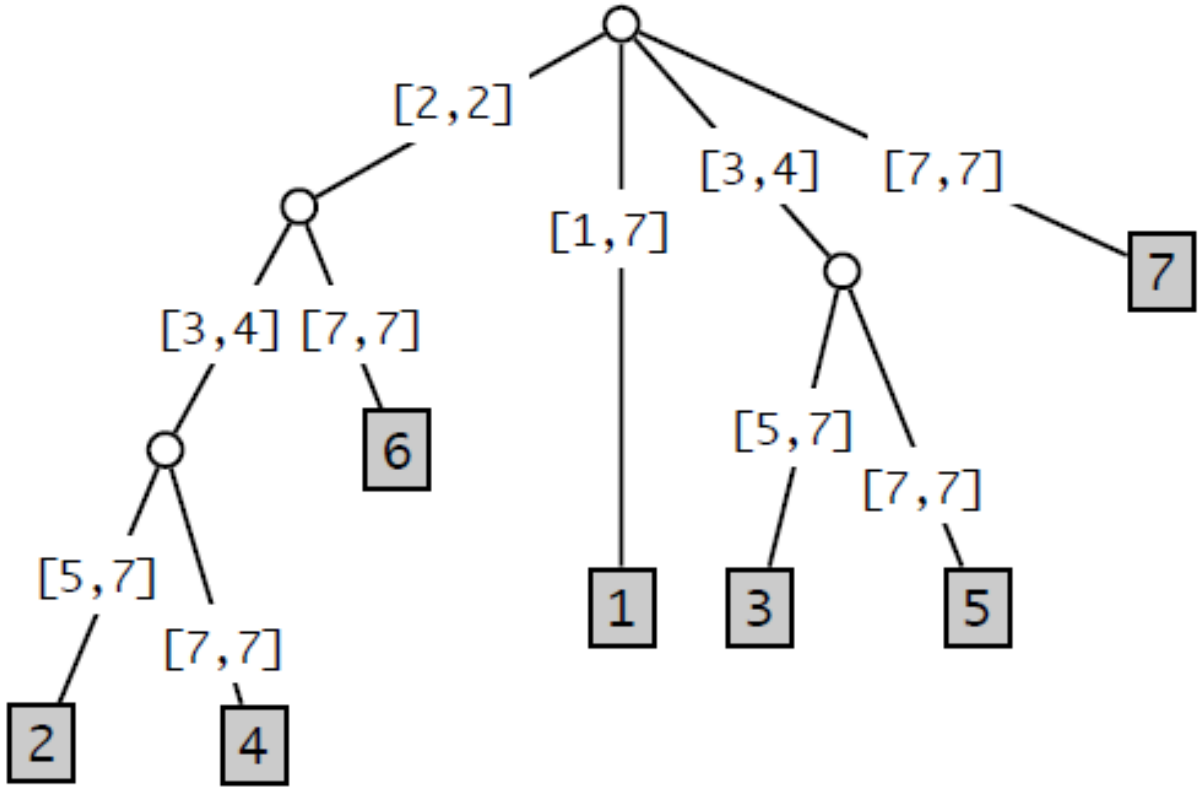
- Find all occurrences of P="an"



- Search for P.
- Report labels of all leaves below final node

Suffix trees

Suffix tree: over the string banana$

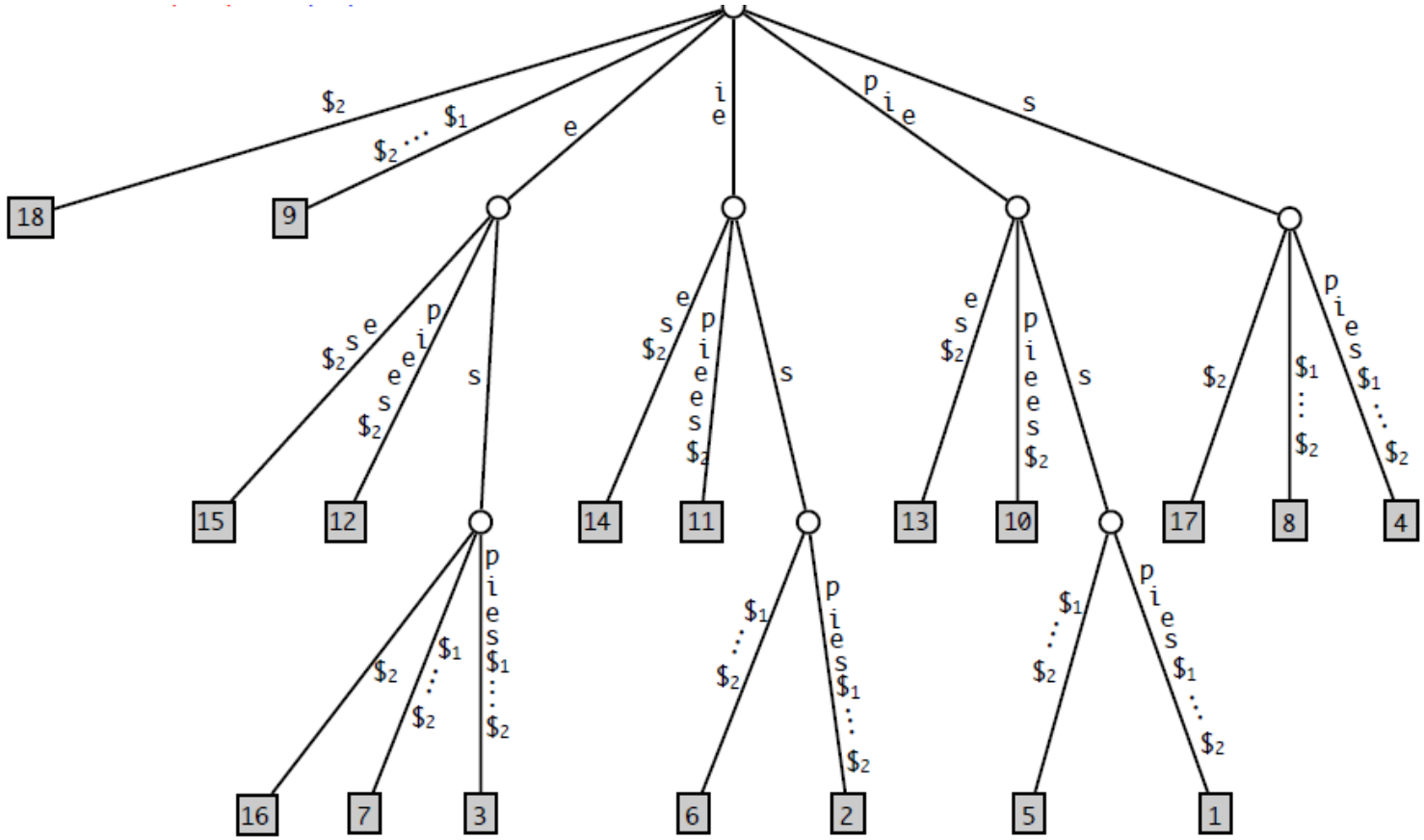


- Store S and store node labels by reference to S.

1 2 3 4 5 6 7
b a n a n a $

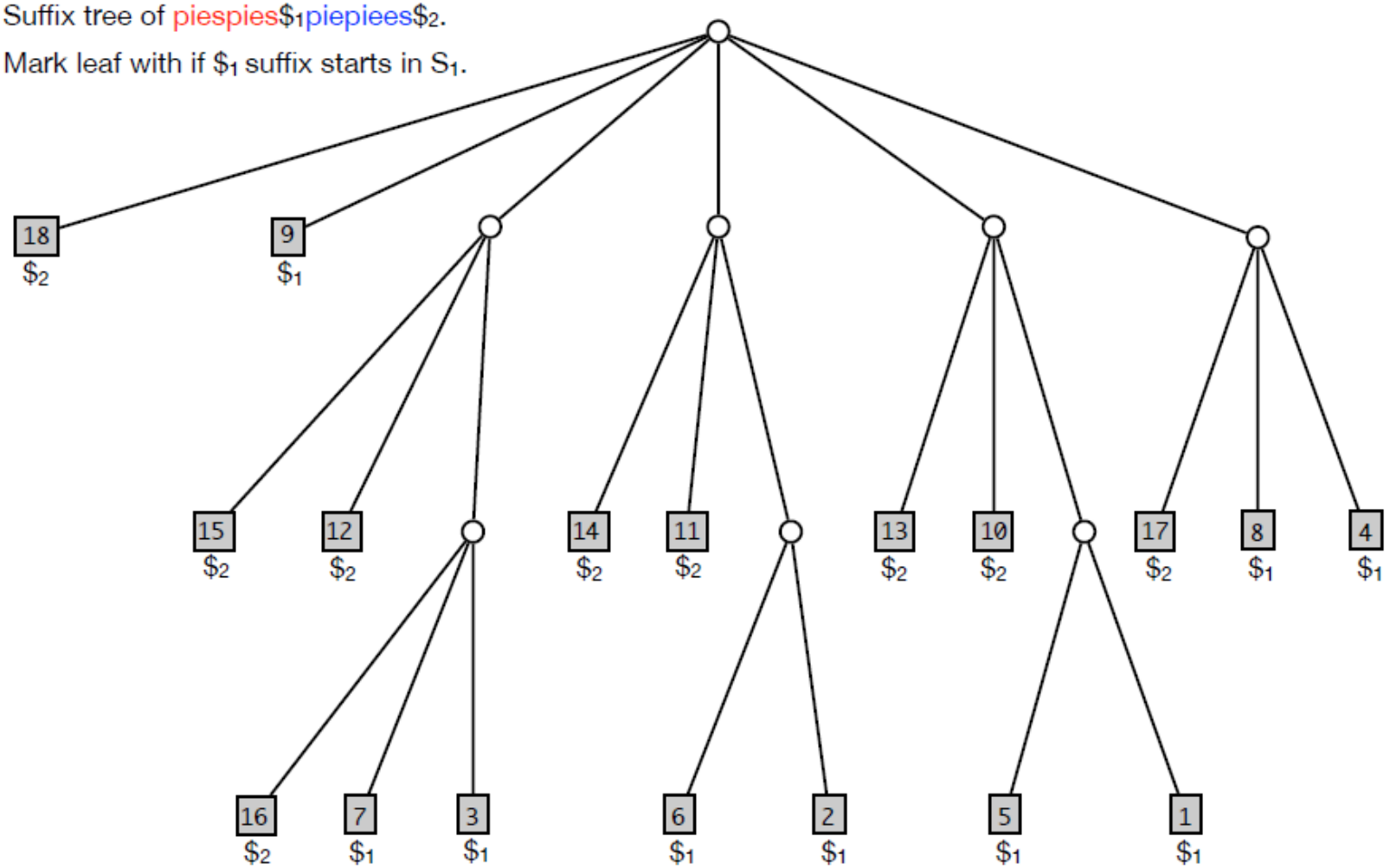
Longest common substring

- Find longest common substring of strings S_1 and S_2
- Construct the suffix tree over $S_1\$_1S_2\$_2$.
- Find longest common substring of **piespies** and piepiees: Construct suffix tree of **piespies**$\$_1$piepiees$\$_2$.



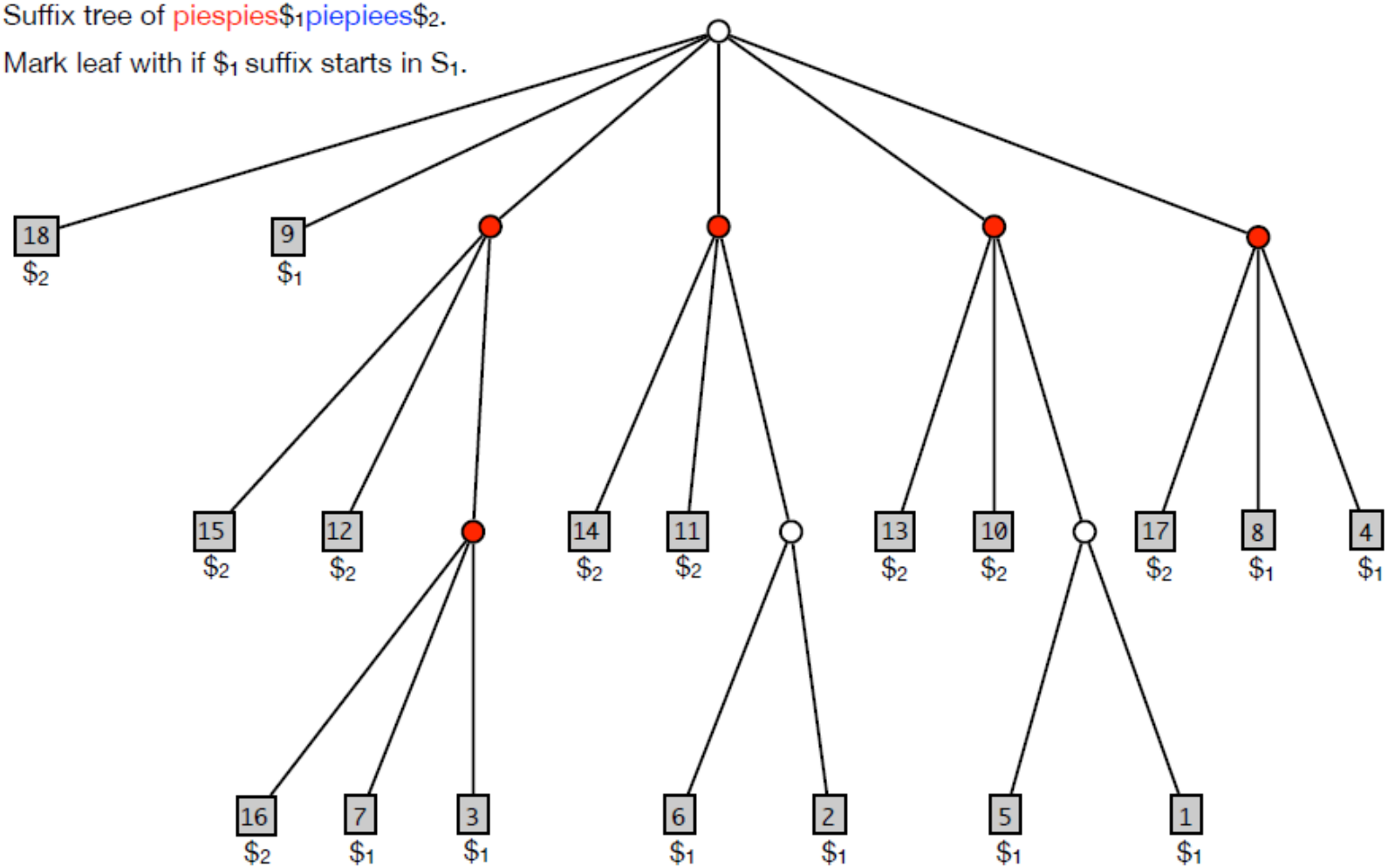
Generalized suffix tree

- Suffix tree of piespies$\$_1$piepiees$\$_2$.
- Mark leaf with if $\$_1$ suffix starts in S_1.



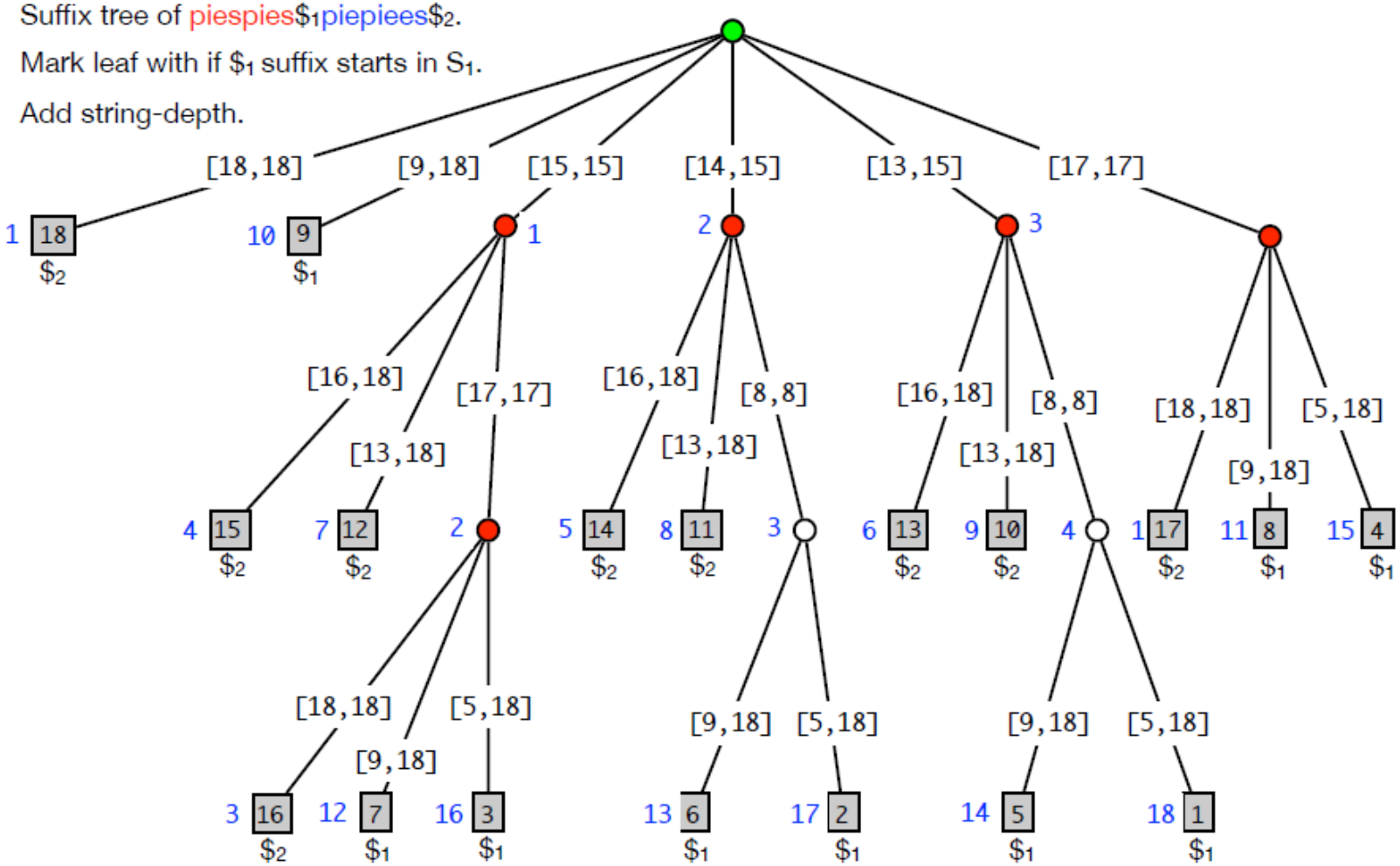
Generalized suffix tree

- Suffix tree of piespies$\$_1$piepiees$\$_2$.
- Mark leaf with if $\$_1$ suffix starts in S_1.

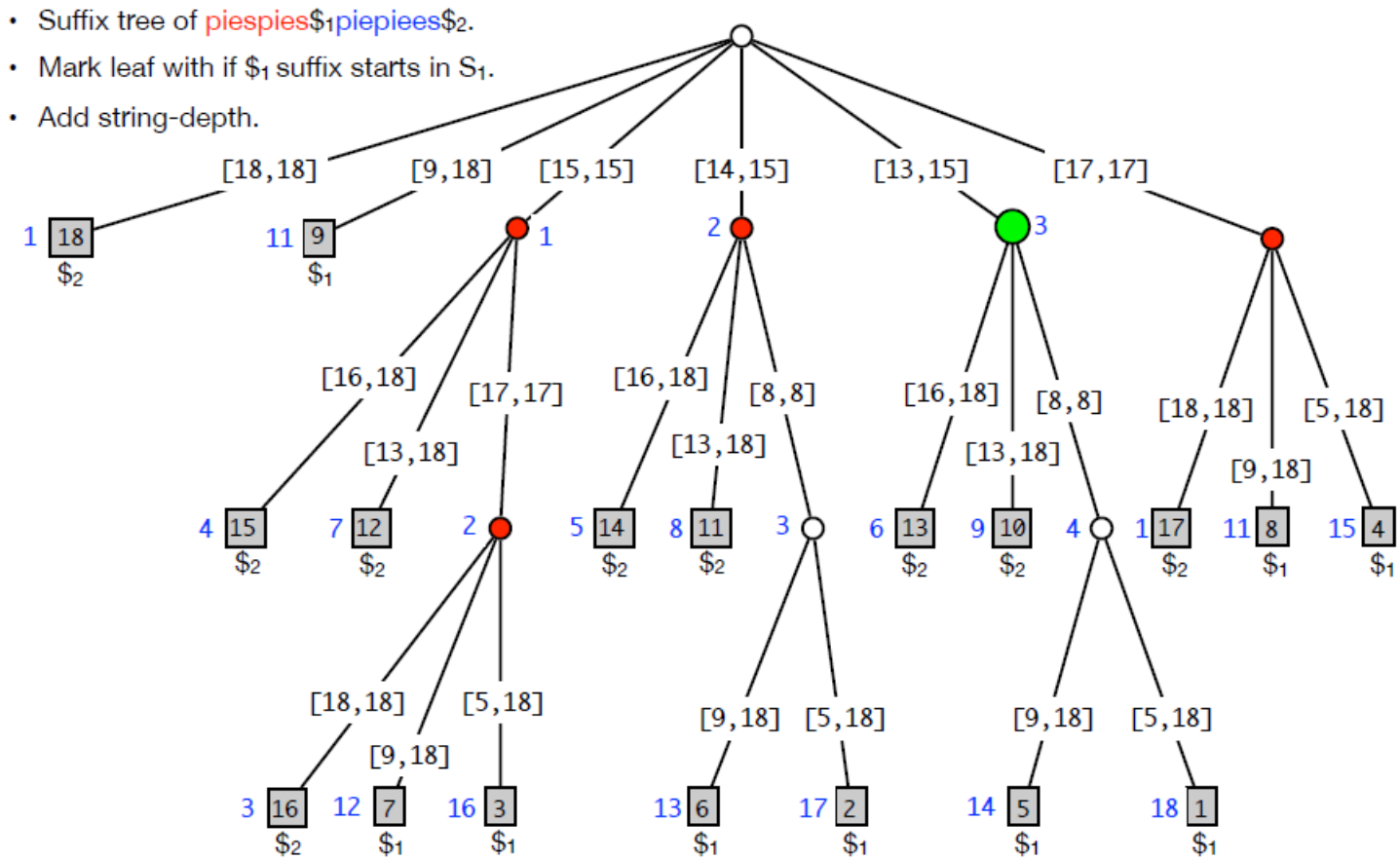


Generalized suffix tree

- Suffix tree of piespies$\$_1$piepiees$\$_2$.
- Mark leaf with if $\$_1$ suffix starts in S_1.
- Add string-depth.



Generalized suffix tree



S[13,15] = ”pie” is the longest common substring.

Using a suffix tree we can solve the longest common substring problem in linear time (for a constant size alphabet).