

String Dictionaries: Tries

Inge Li Gørtz

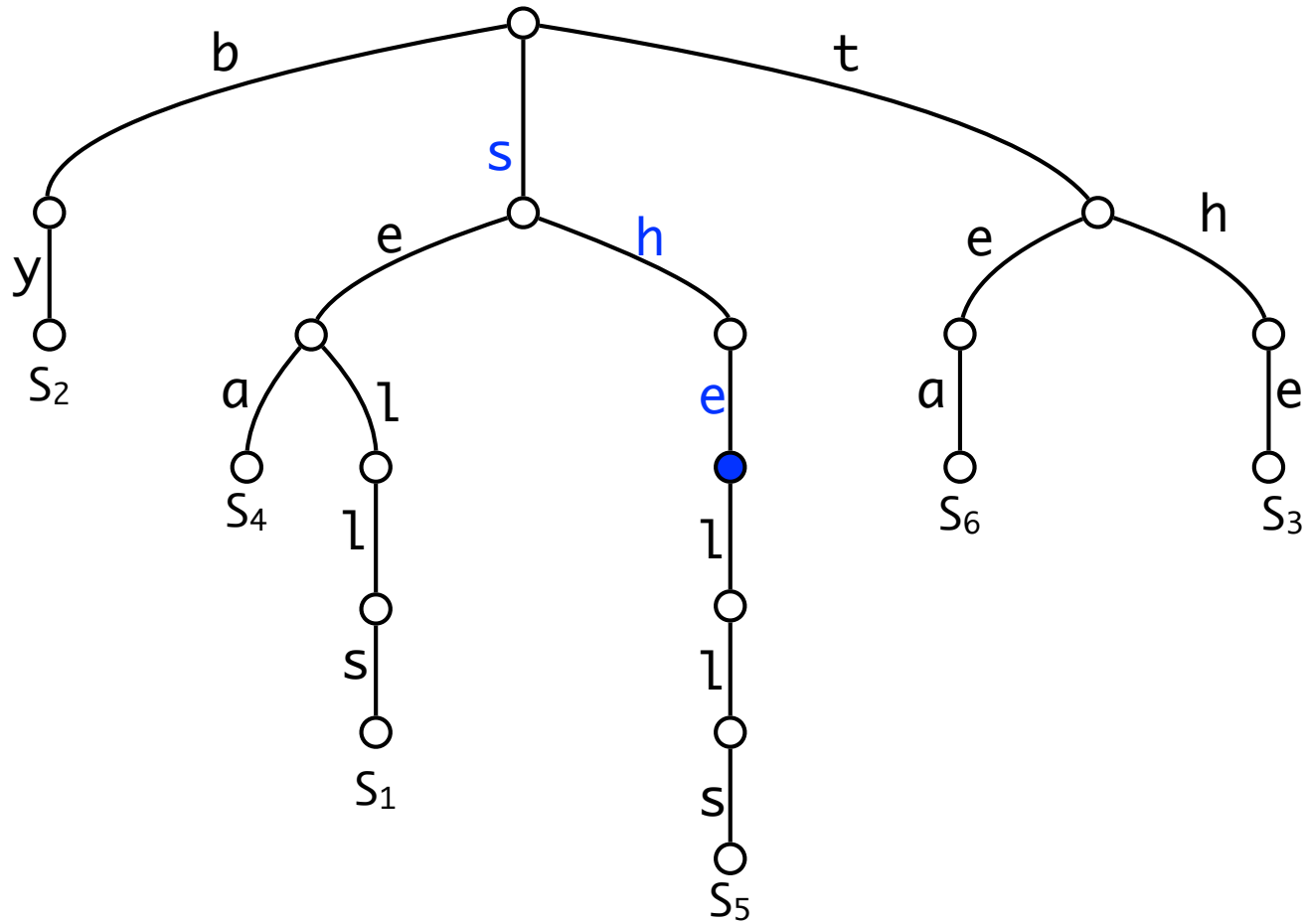
String Dictionaries

- **String dictionary problem.** Let $S = \{S_1, S_2, \dots, S_k\}$ be a set of strings of characters from alphabet Σ . Preprocess S into data structure to support:
 - **search(P):** Return yes if $P = S_i$ for some S_i in S .
 - **prefix-search(P):** Return all strings in S for which P is a prefix.
- **Example.** $S = \{\text{sells, by, the, sea, shells, tea}\}$.
 - **prefix-search('se')** returns 'sea' and 'sells'.

Tries

Tries

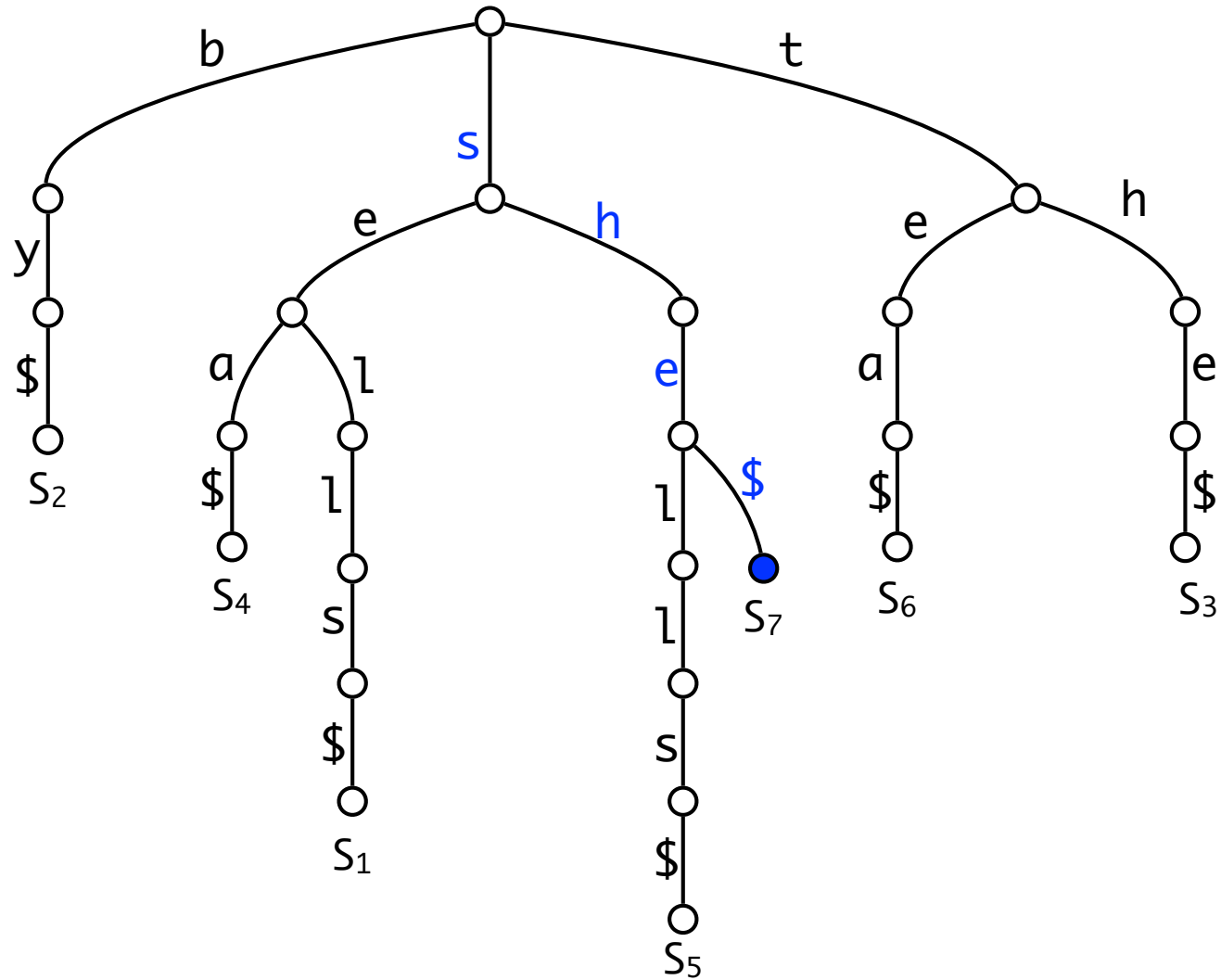
- Text retrieval
- Prefix-free?



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

Tries

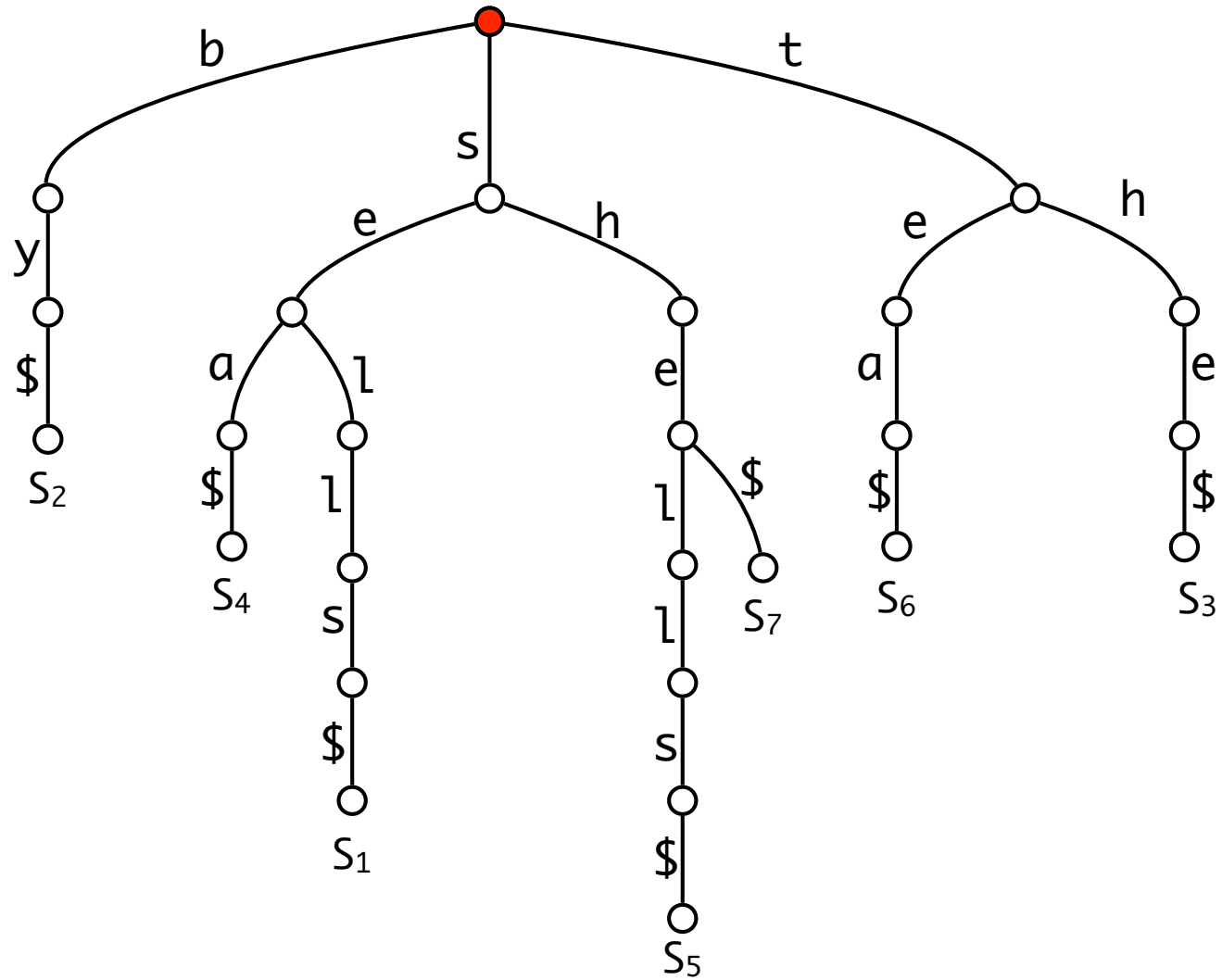
- Text retrieval
- Prefix-free?



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

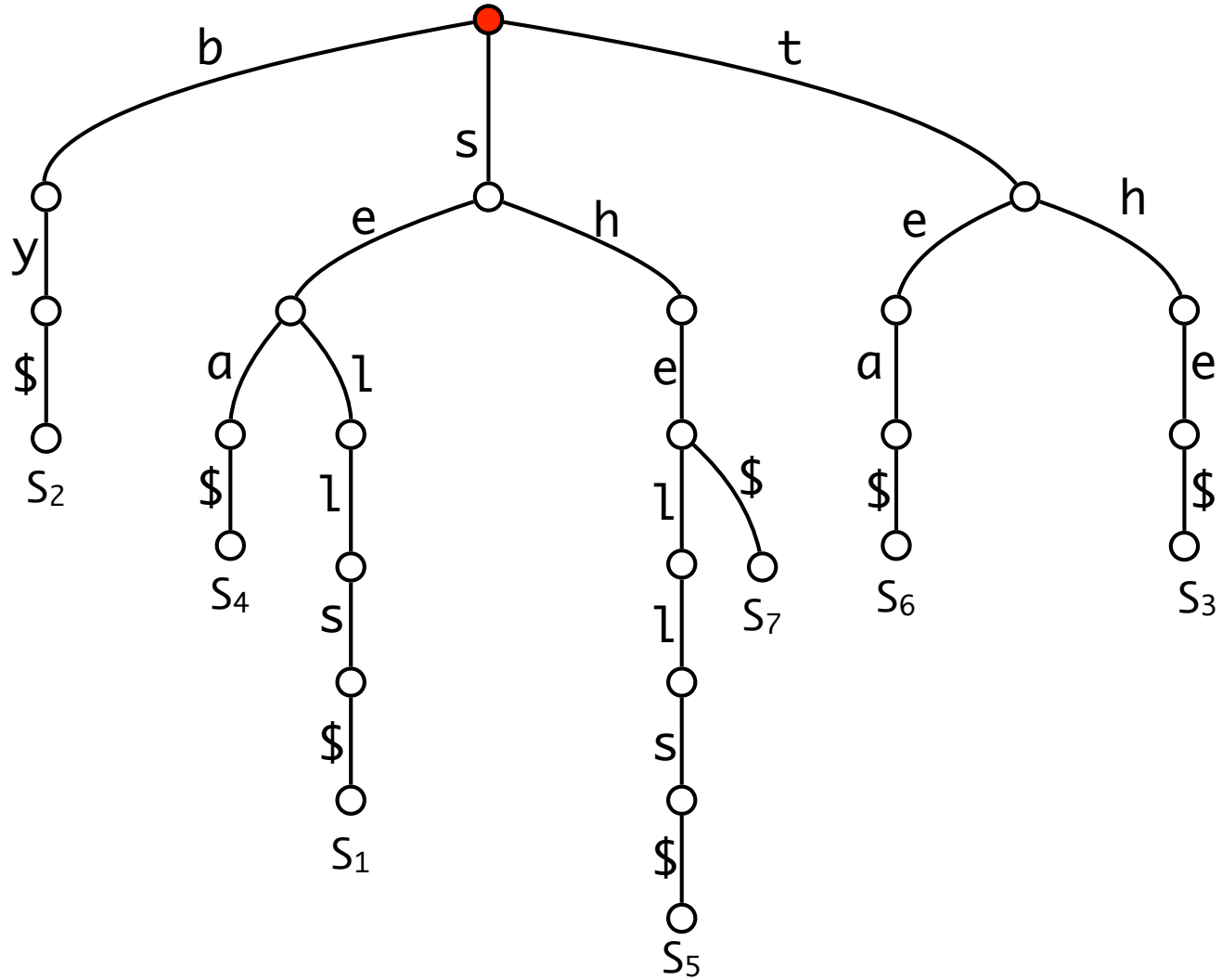
Tries: Searching

- Search for 'sea\$':



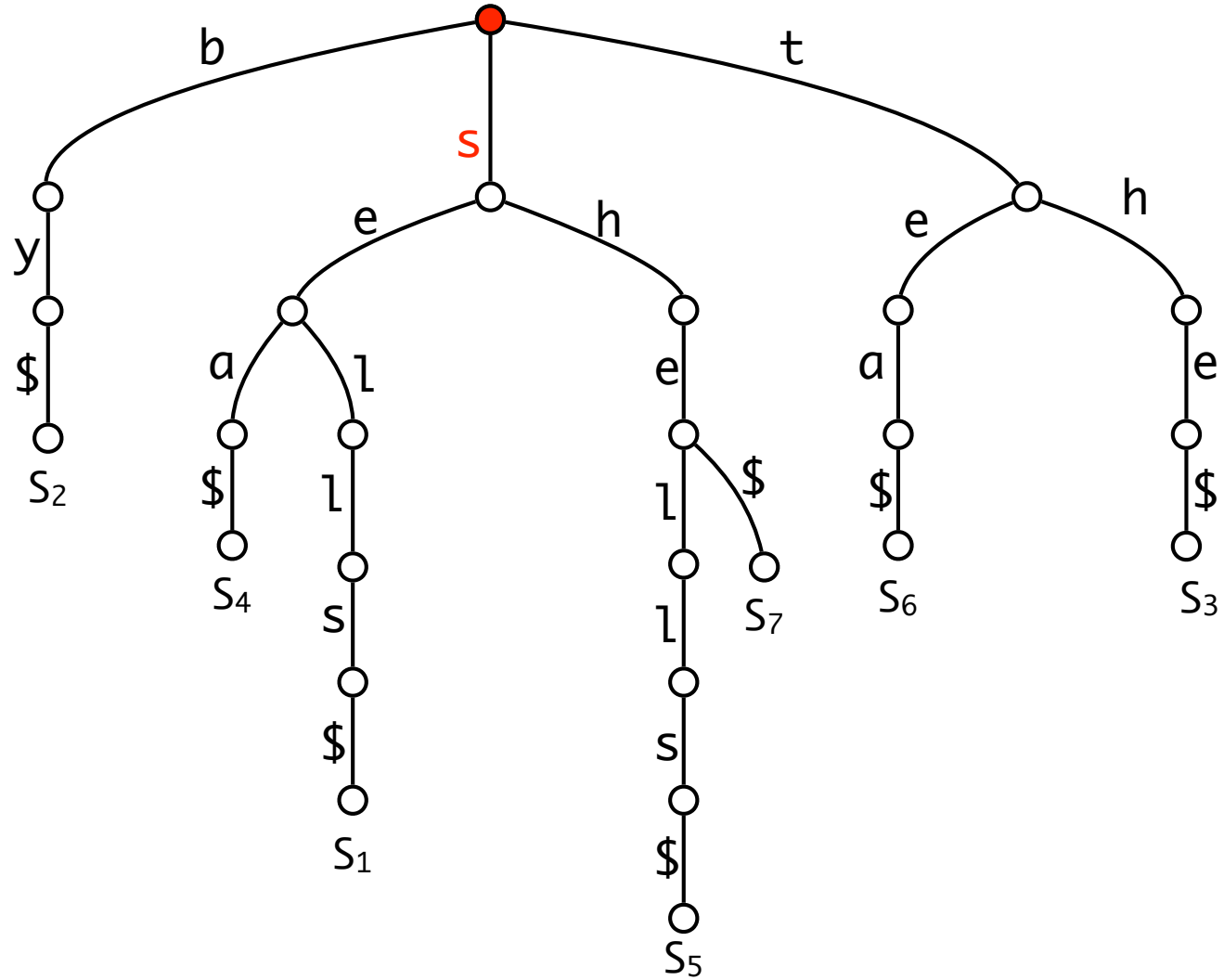
Tries: Searching

- Search for 'sea\$':



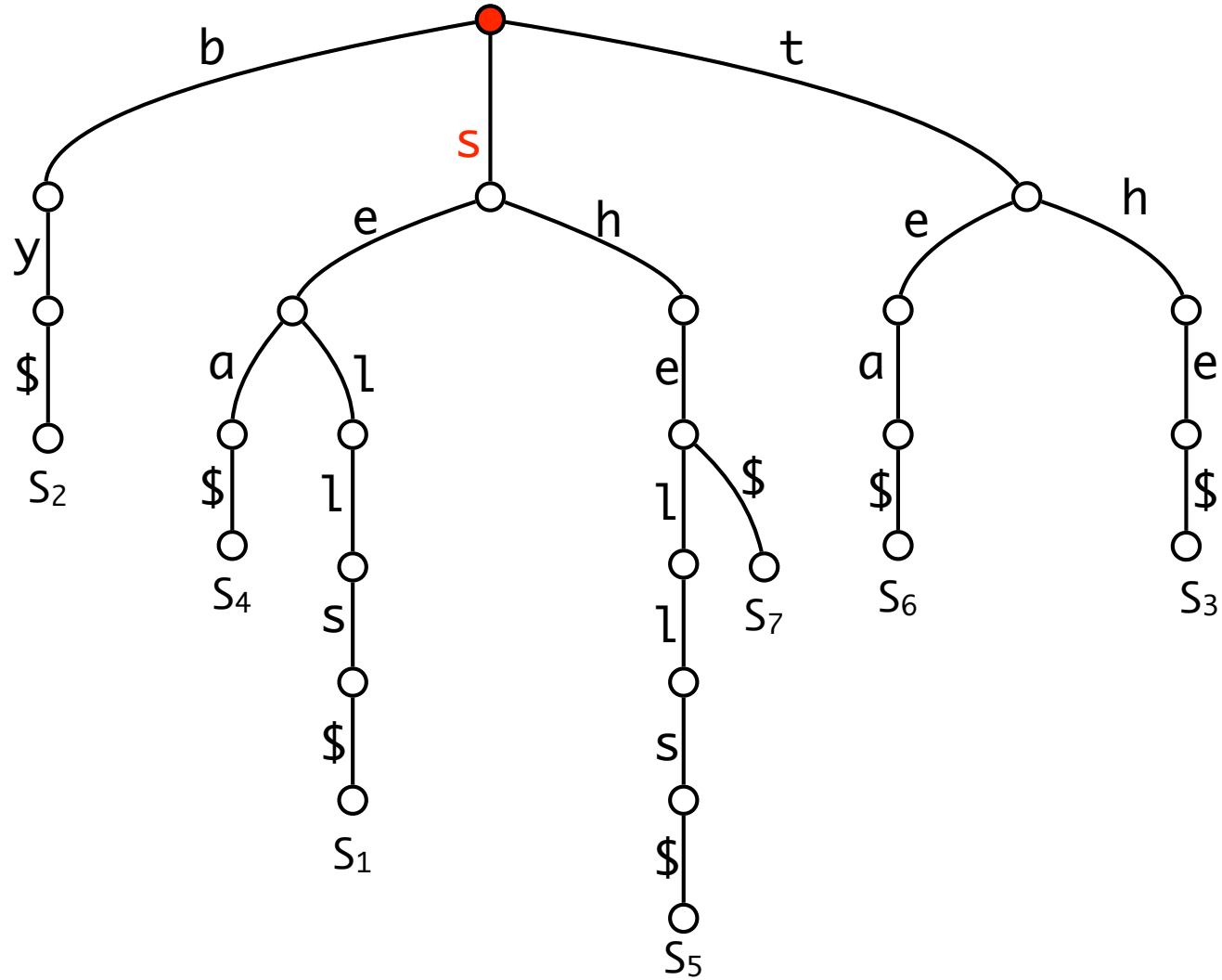
Tries: Searching

- Search for 'sea\$':



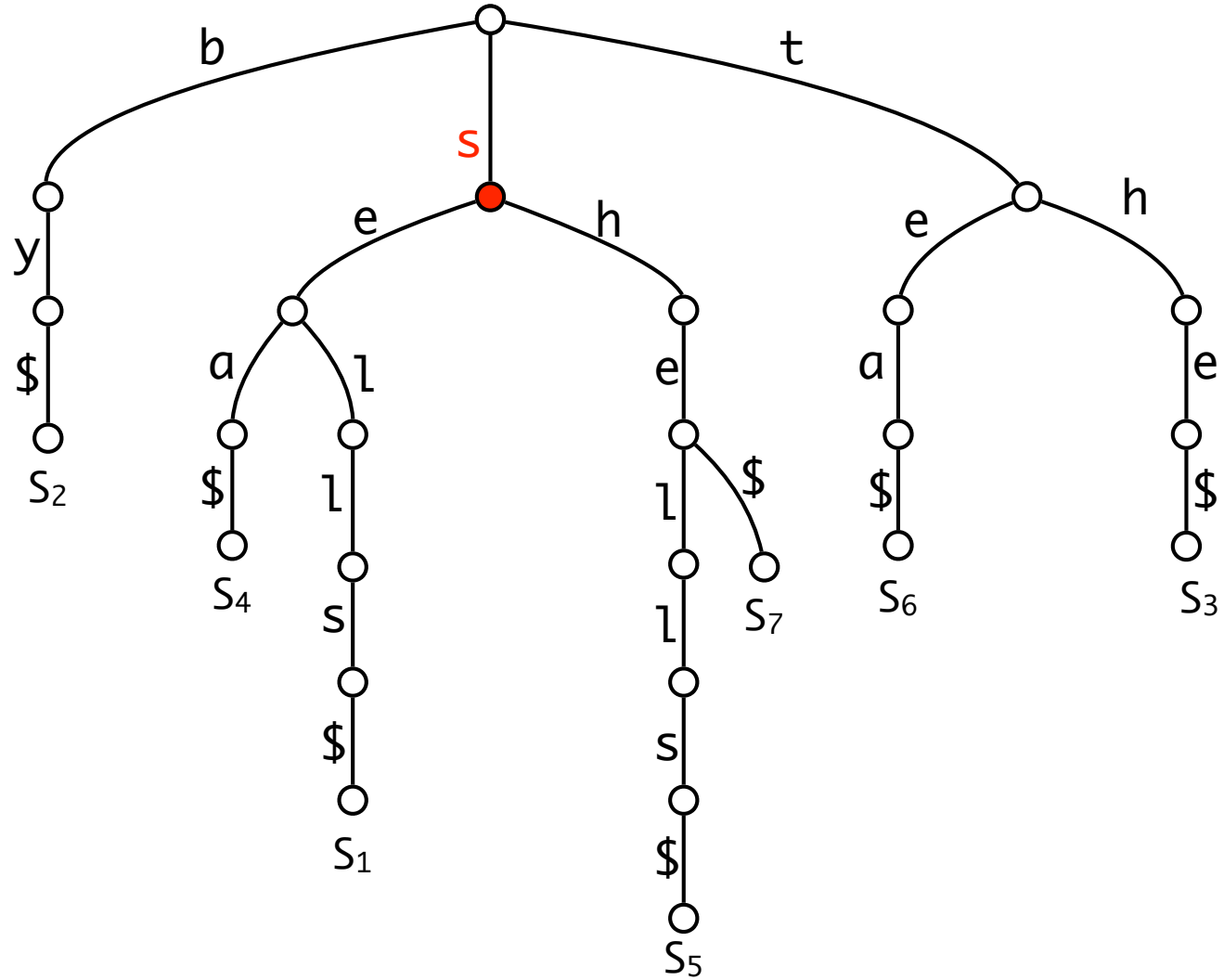
Tries: Searching

- Search for 'sea\$':



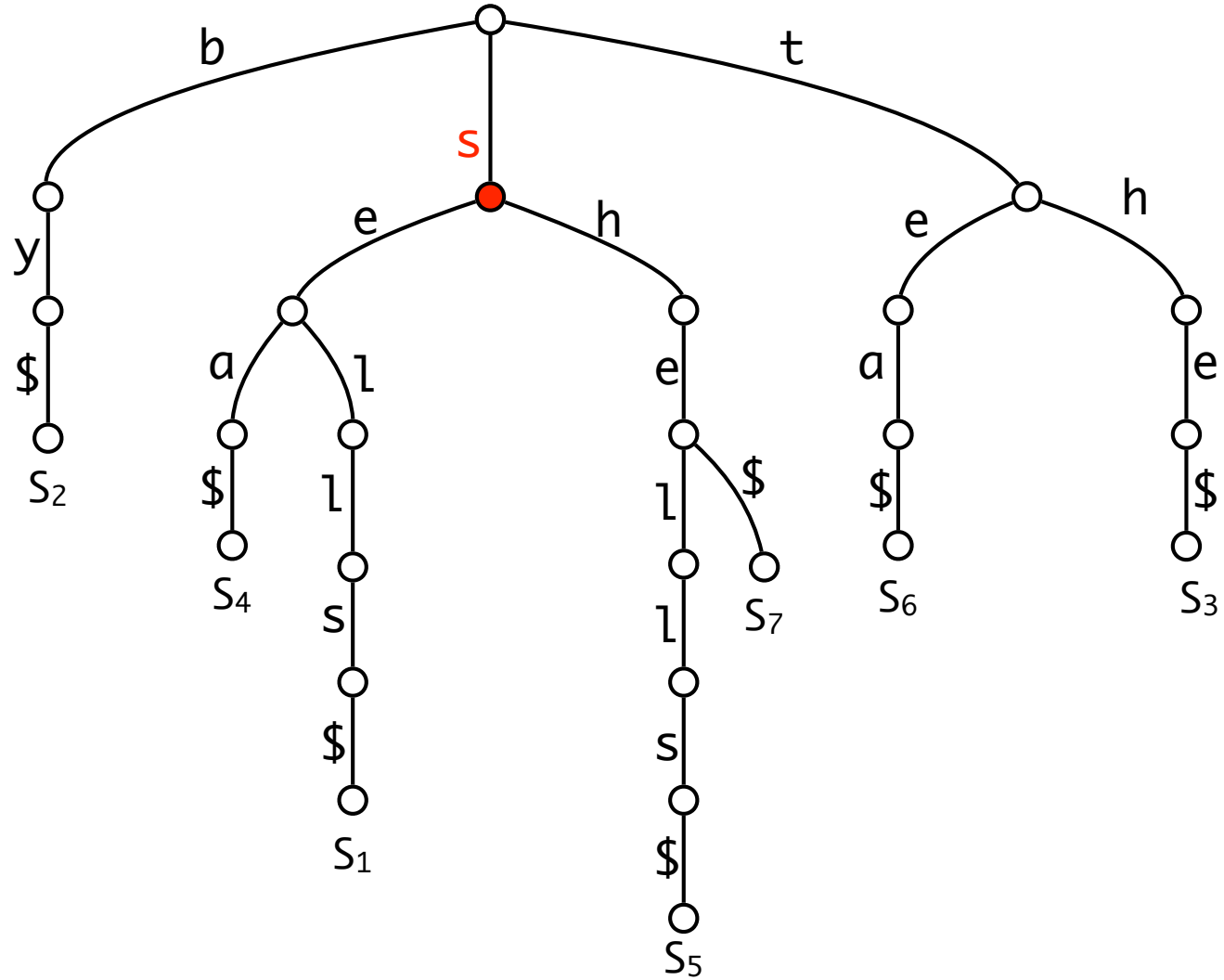
Tries: Searching

- Search for 'sea\$':



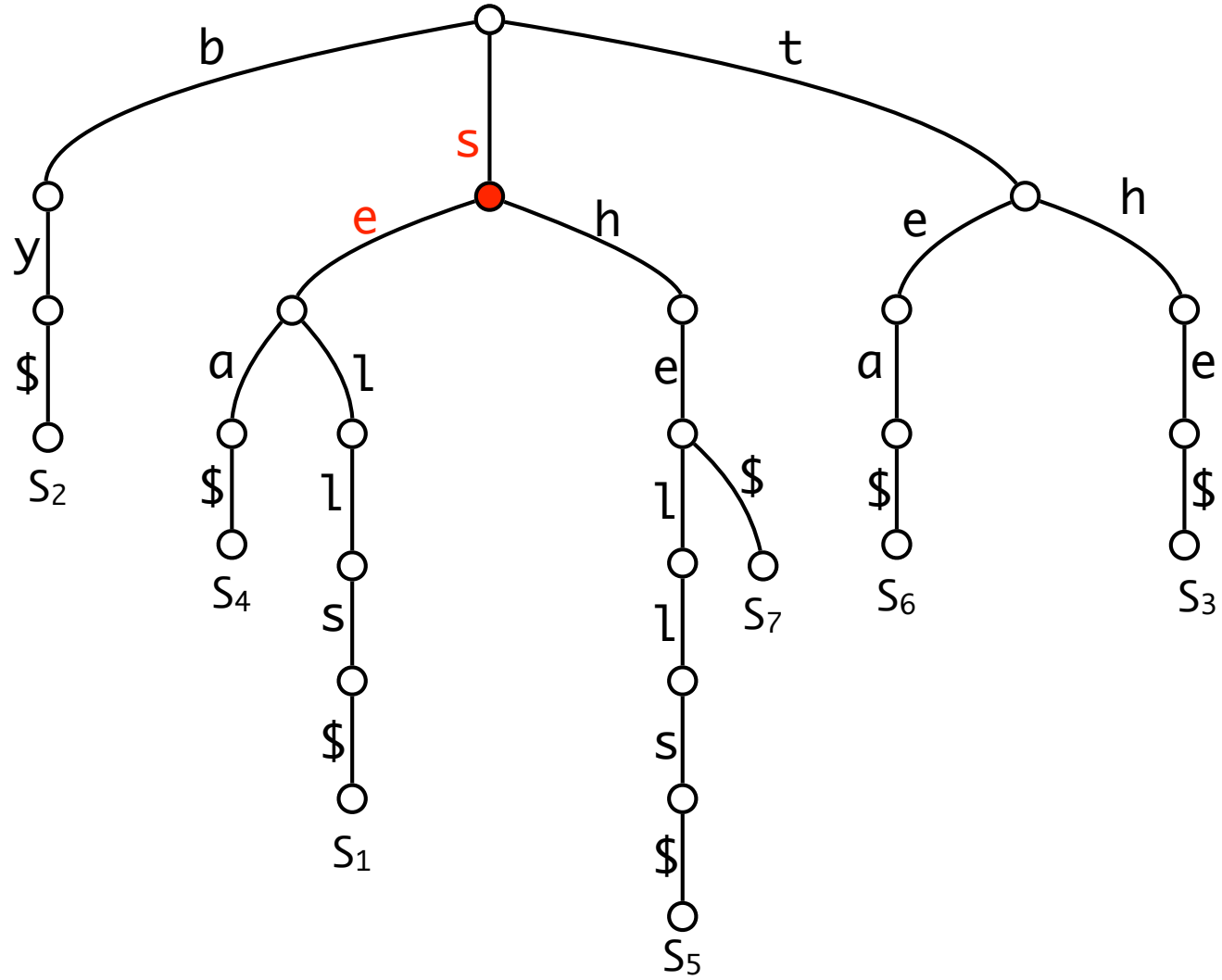
Tries: Searching

- Search for 'sea\$':



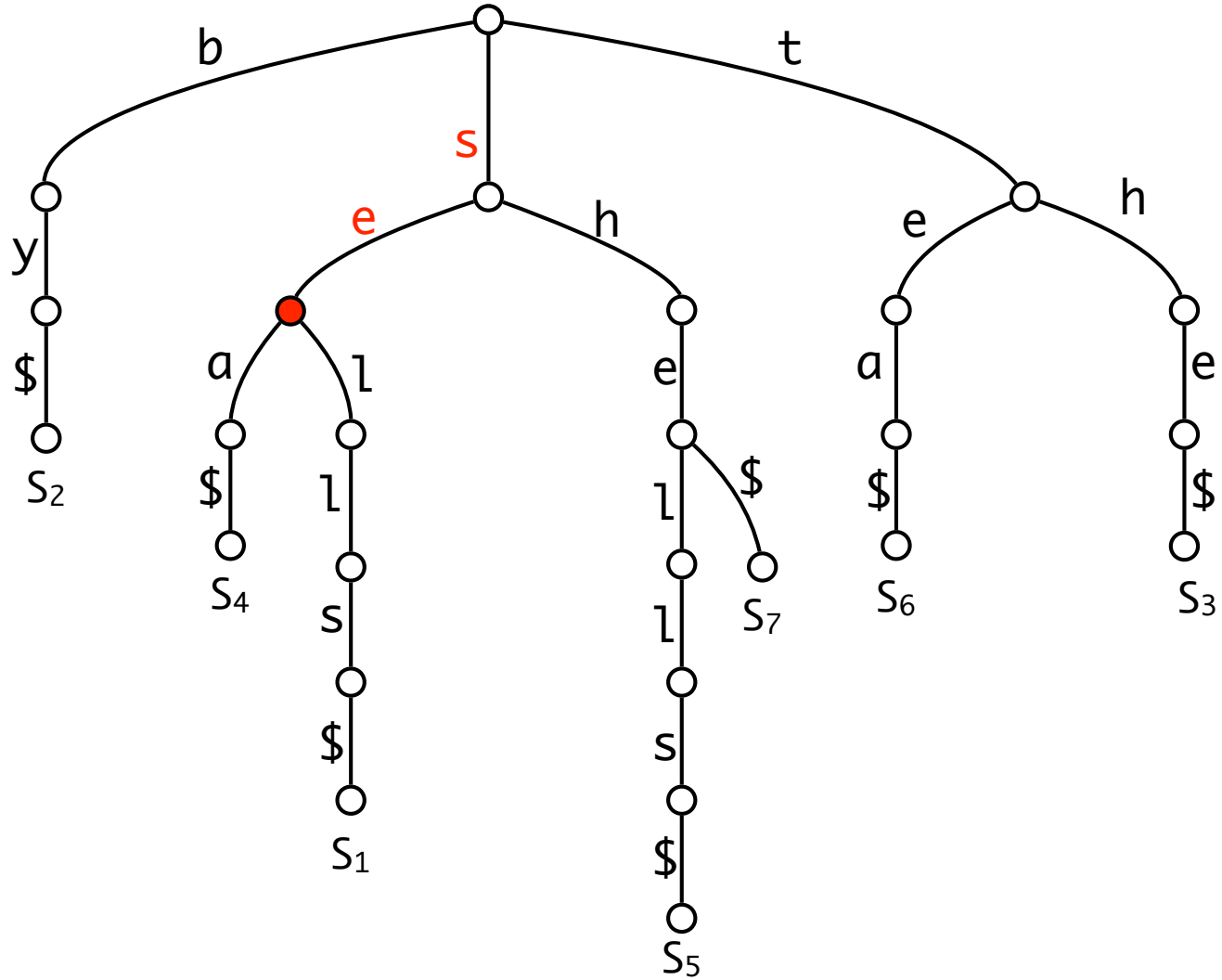
Tries: Searching

- Search for 'sea\$':



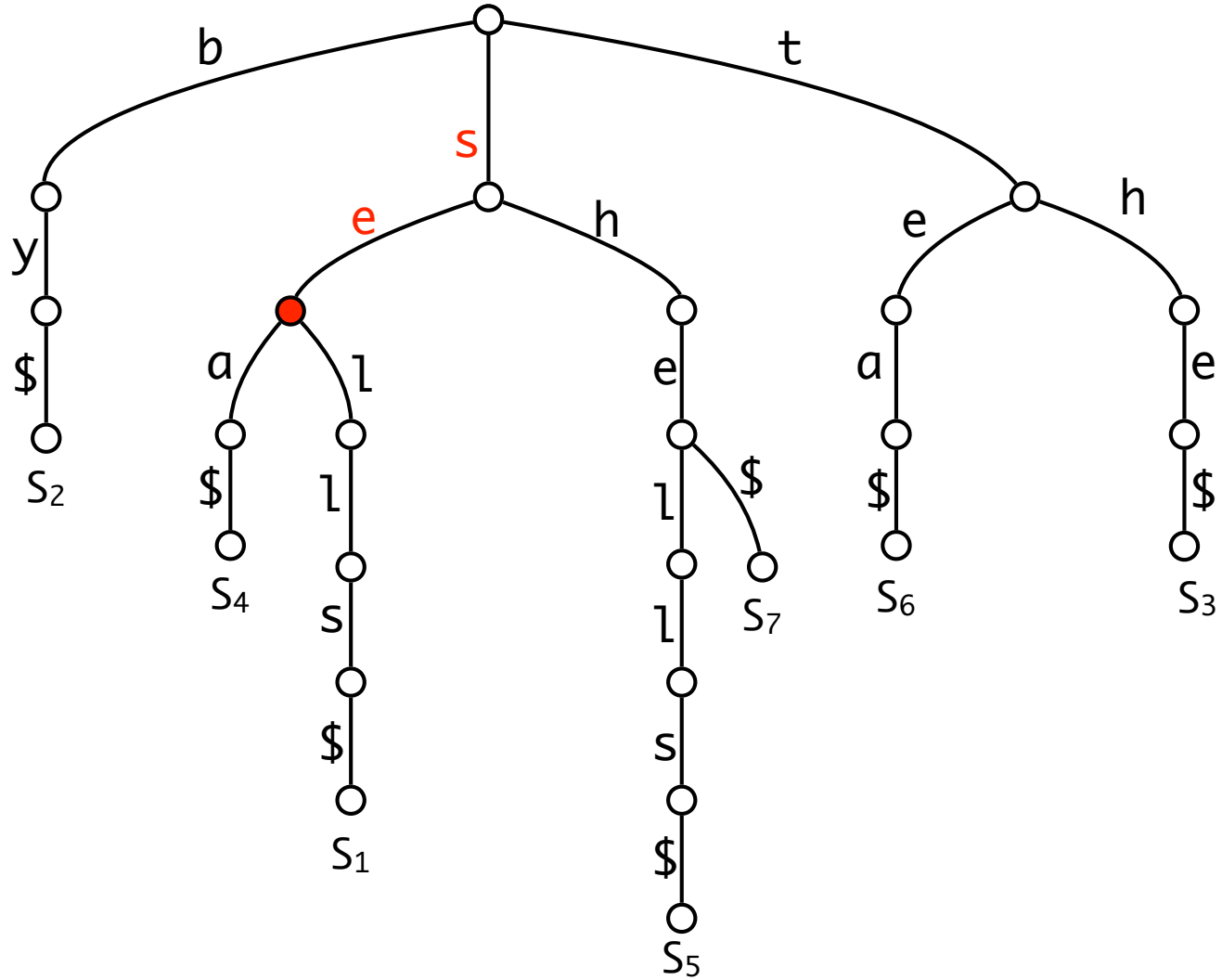
Tries: Searching

- Search for 'sea\$':



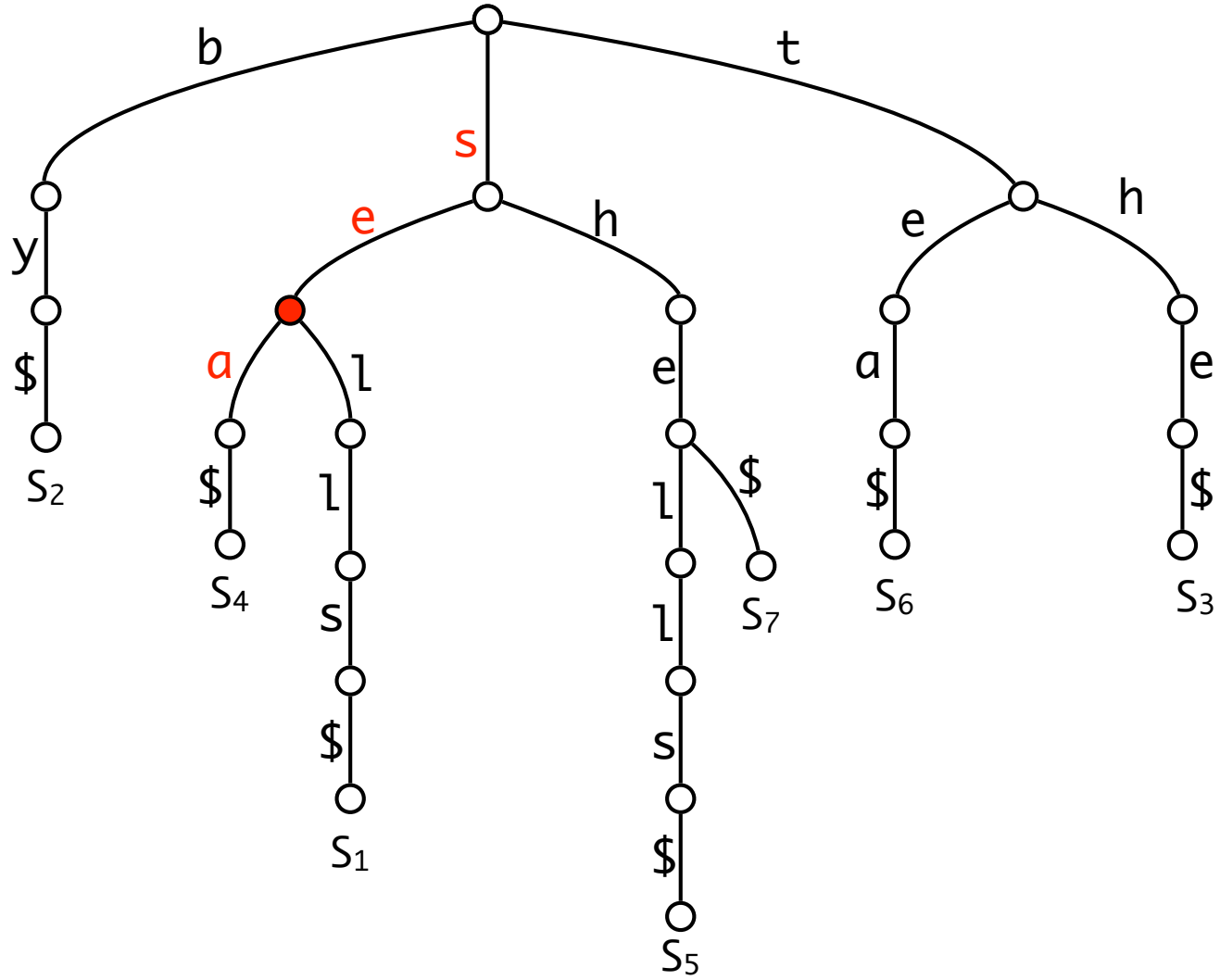
Tries: Searching

- Search for 'sea\$':



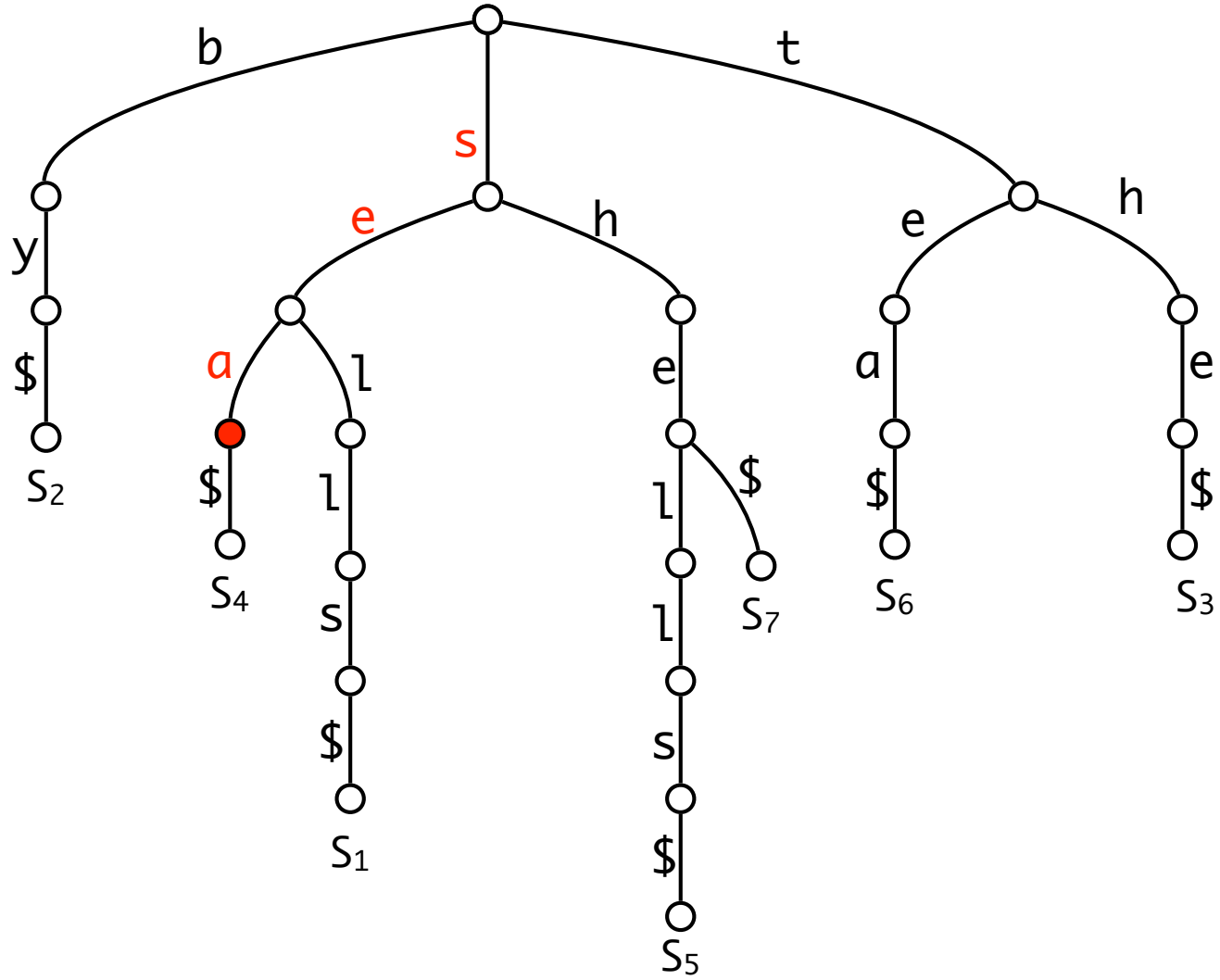
Tries: Searching

- Search for 'sea\$':

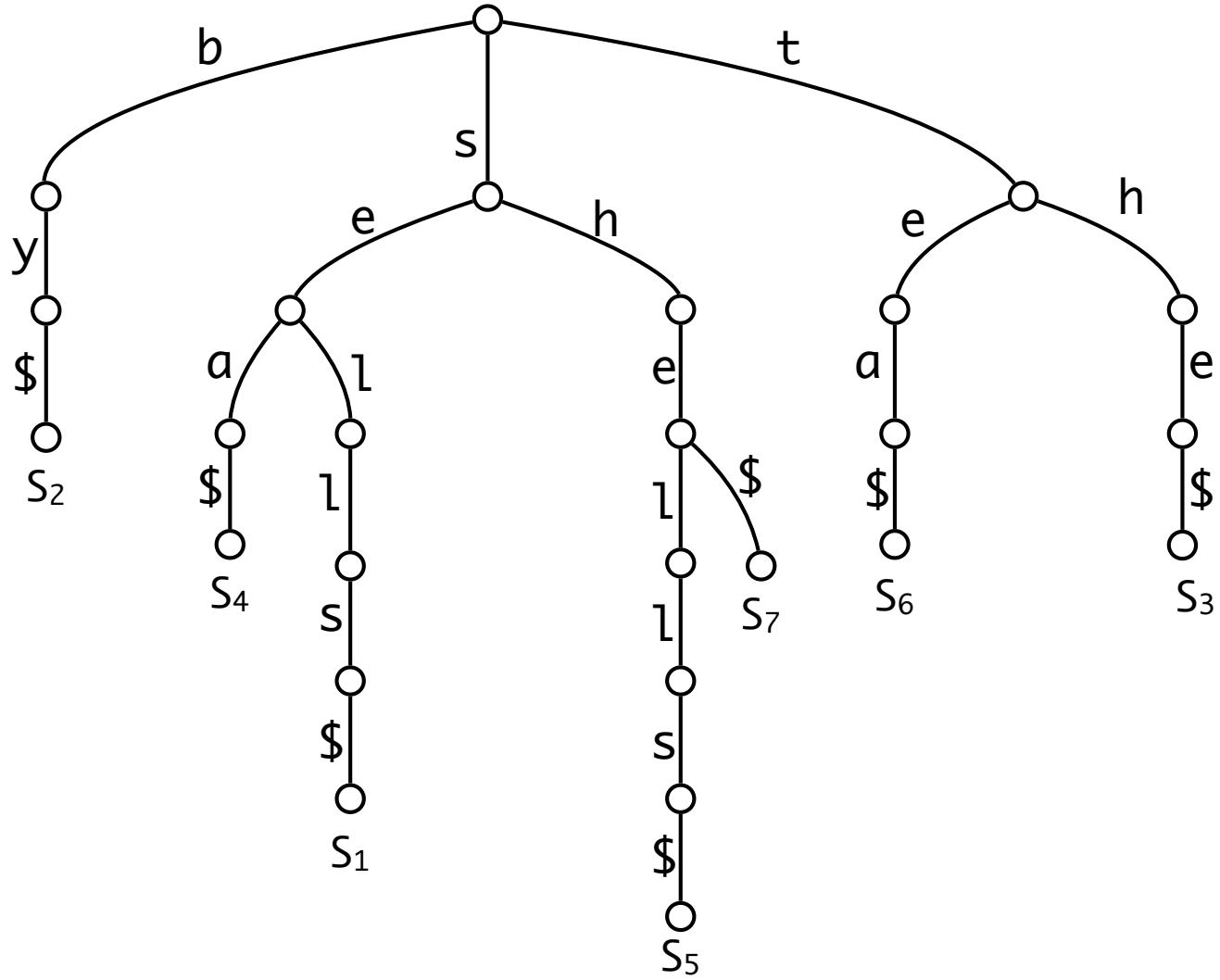


Tries: Searching

- Search for 'sea\$':



Tries: Searching



Tries

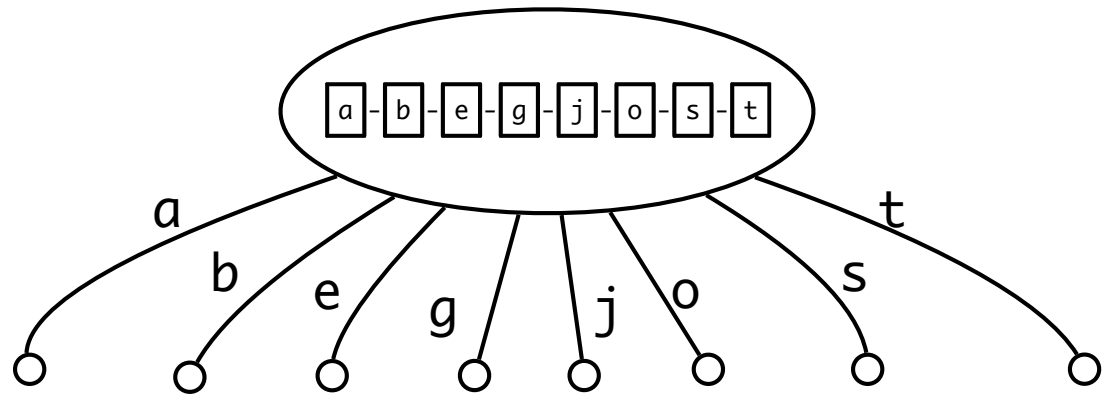
- [Fredkin 1960](#)]. Retrieval. Store a set of strings in a rooted tree such that:
 - Each edge is labeled by a character. Edges to children of a node are sorted from left-to-right alphabetically.
 - Each root-to-leaf path represents a string in the set. (obtained by concatenating the labels of edges on the path).
 - Common prefixes share same path maximally.

Trie

- **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? at most d
 - How many leaves does T have? s
 - What is the height of T ? length of longest string
 - What is the number of nodes in T ? $O(n)$

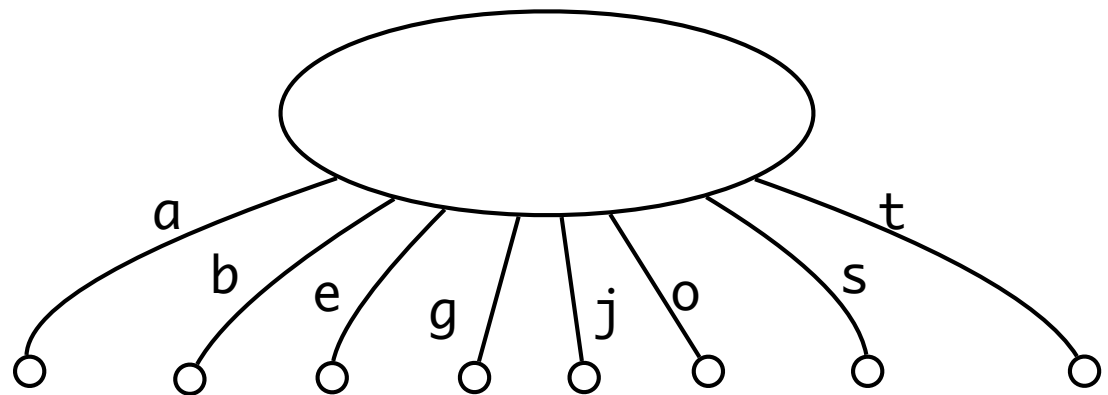
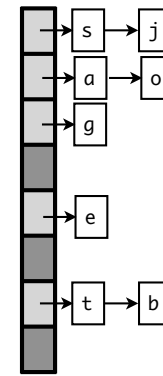
Trie

- Search time:
 - List: $O(d)$ in each node $\Rightarrow O(dm)$.
 - $O(m)$ if d constant.



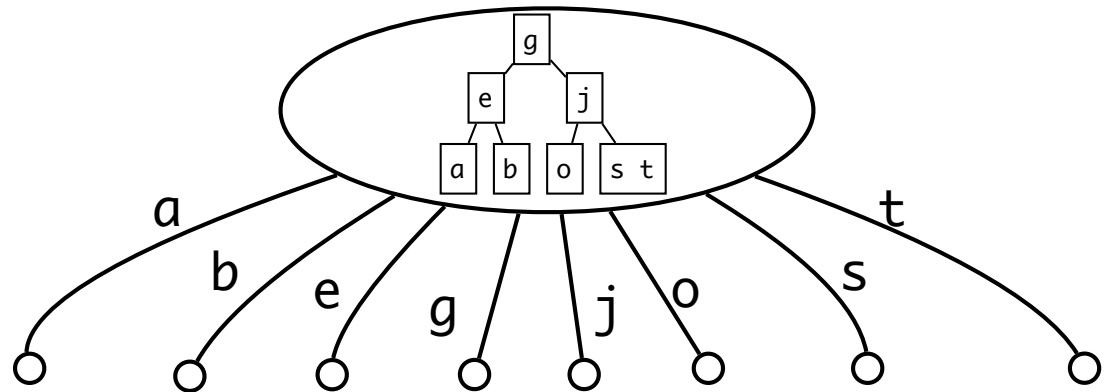
Trie

- Search time:
 - List: $O(d)$ in each node $\Rightarrow O(dm)$.
 - $O(m)$ if d constant.
 - d not constant: use dictionary
 - Hashtable with universal hashing: expected $O(1)$



Trie

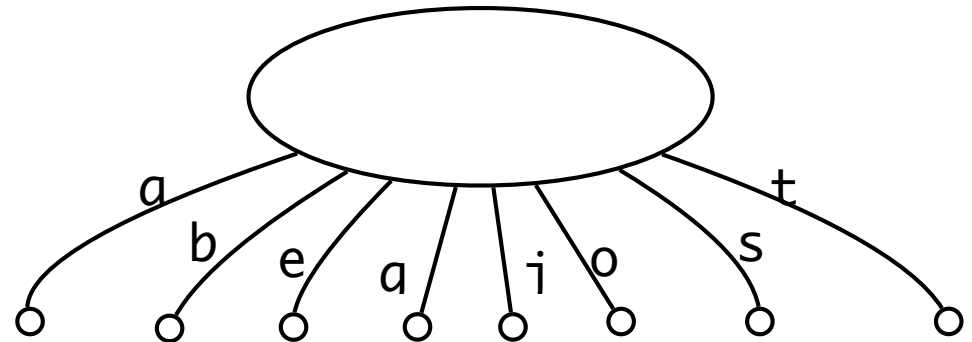
- Search time:
 - List: $O(d)$ in each node $\Rightarrow O(dm)$.
 - $O(m)$ if d constant.
 - d not constant: use dictionary
 - Hashtable with universal hashing: expected $O(1)$
 - Balanced Search Tree: $O(\log d)$



Trie

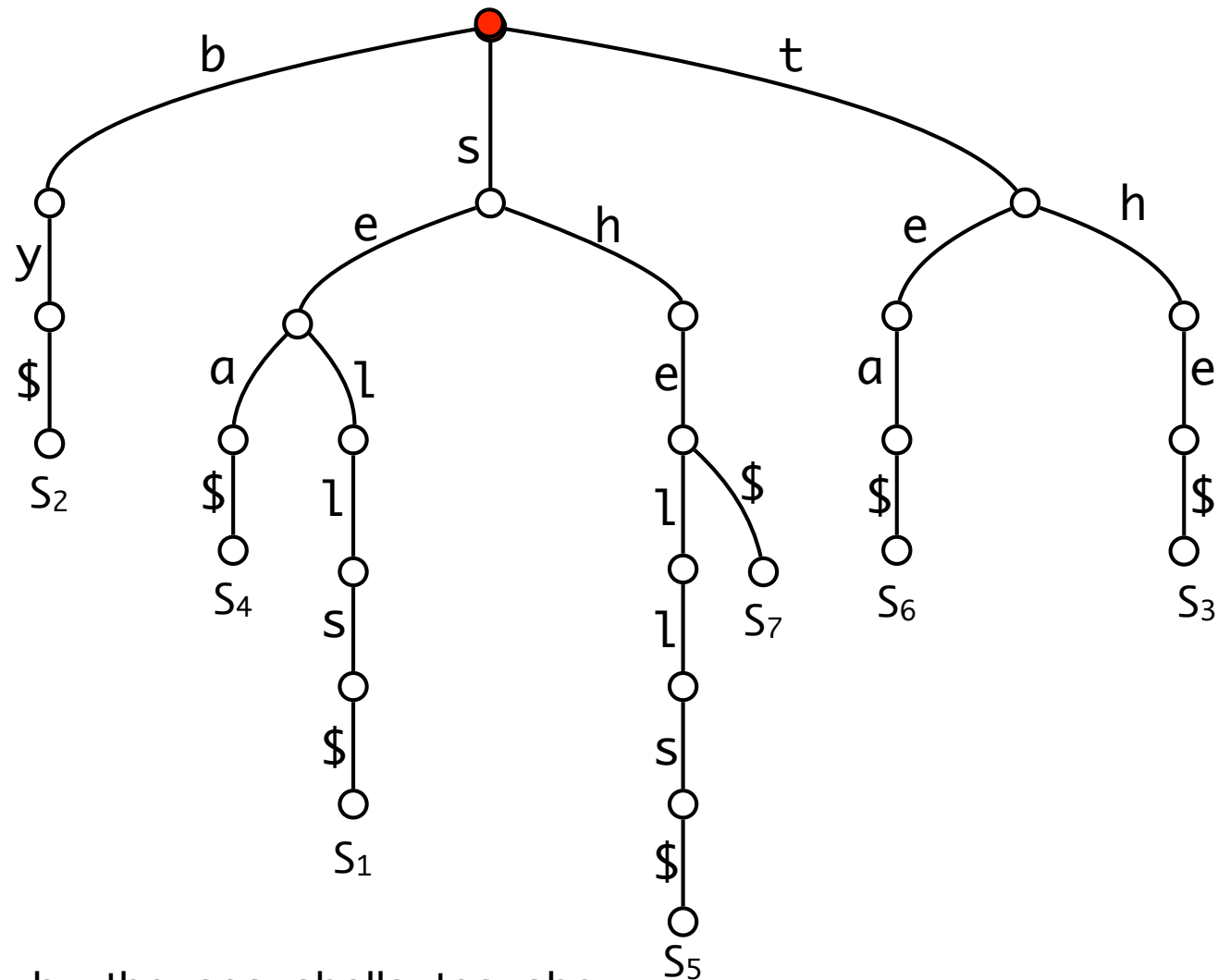
- Search time:
 - List: $O(d)$ in each node $\Rightarrow O(dm)$.
 - $O(m)$ if d constant.
 - d not constant: use dictionary
 - Hashtable with universal hashing $O(1)$
 - Balanced BST: $O(\log d)$
- Space for a trie:
 - $O(n)$ nodes and edges.
 - Space for data structure in each node:
 - If each node uses space proportional to the number of edges out of it, then

$$\text{space} = \sum_v \text{size of data structure} = \# \text{edges} = O(n).$$



Construction of Tries

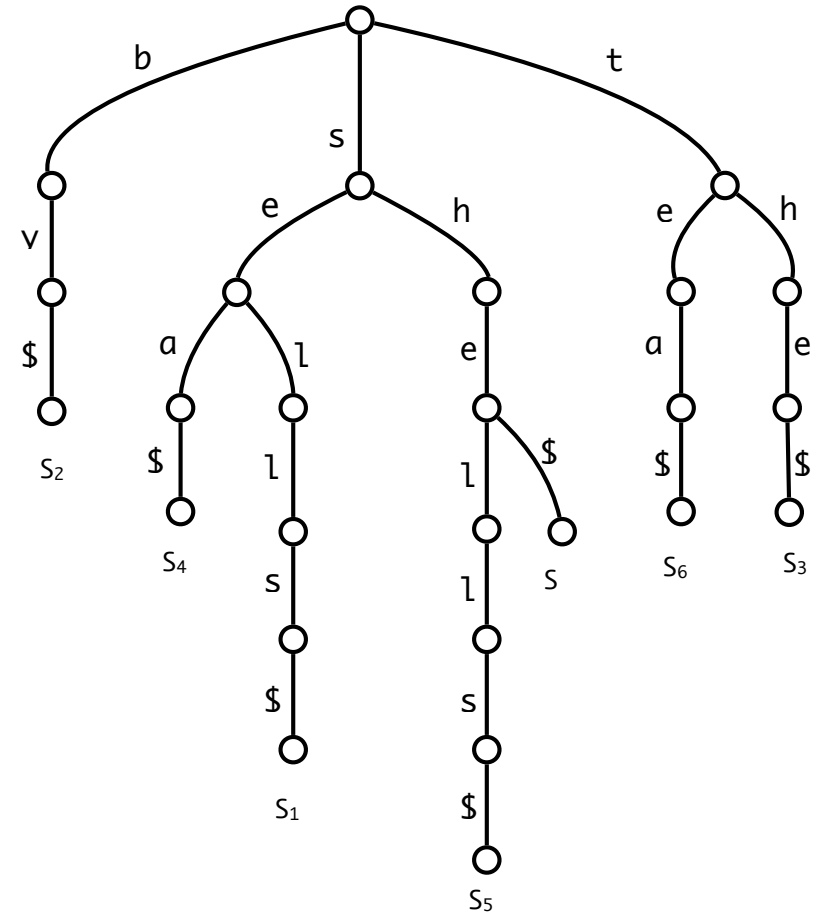
- Insert the strings one-by-one.



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

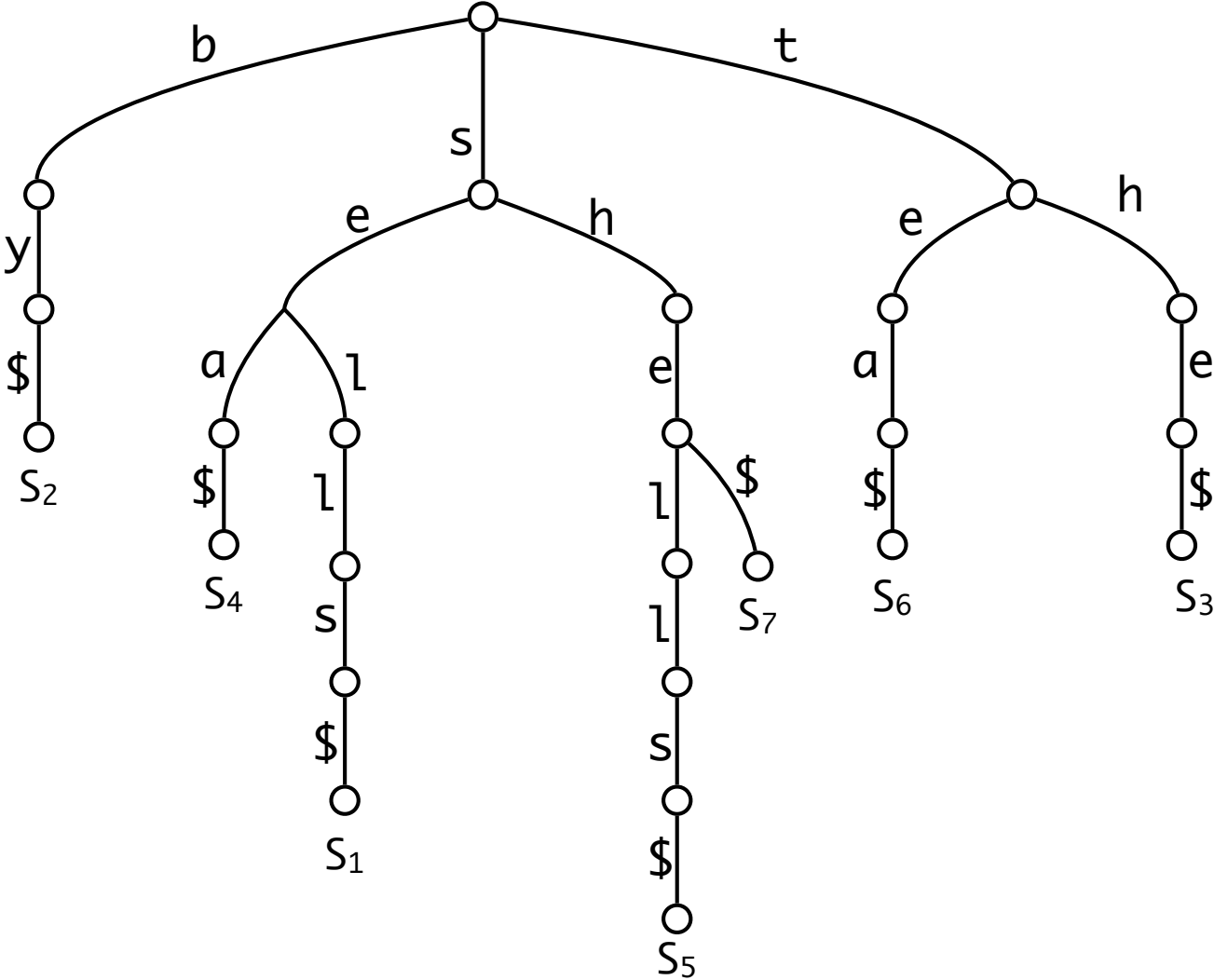
Trie

- Time and space for a trie (for constant d):
 - $O(m)$ for searching for a string of length m .
 - $O(n)$ space.
 - Preprocessing: $O(n)$



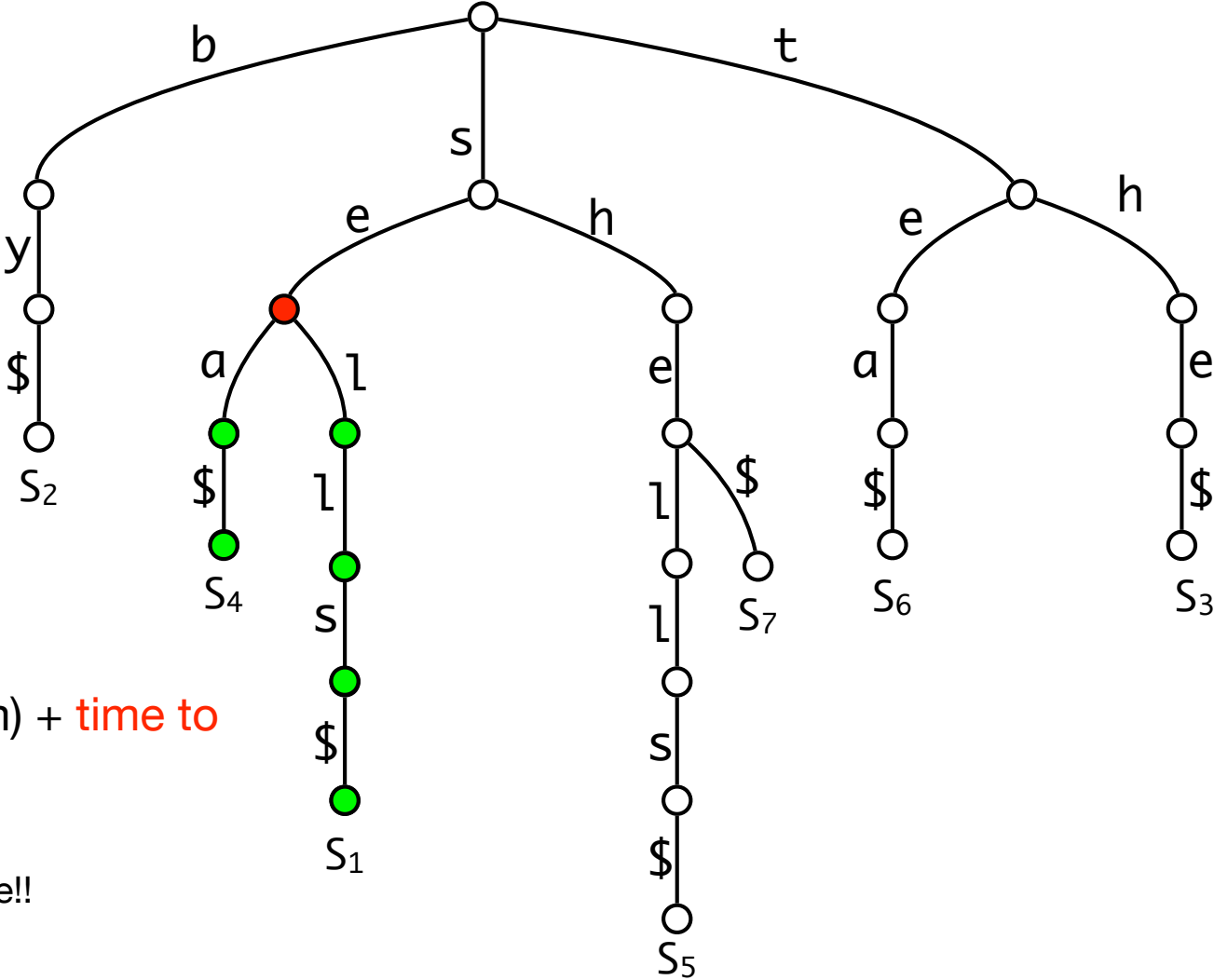
Prefix search

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



Prefix search

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



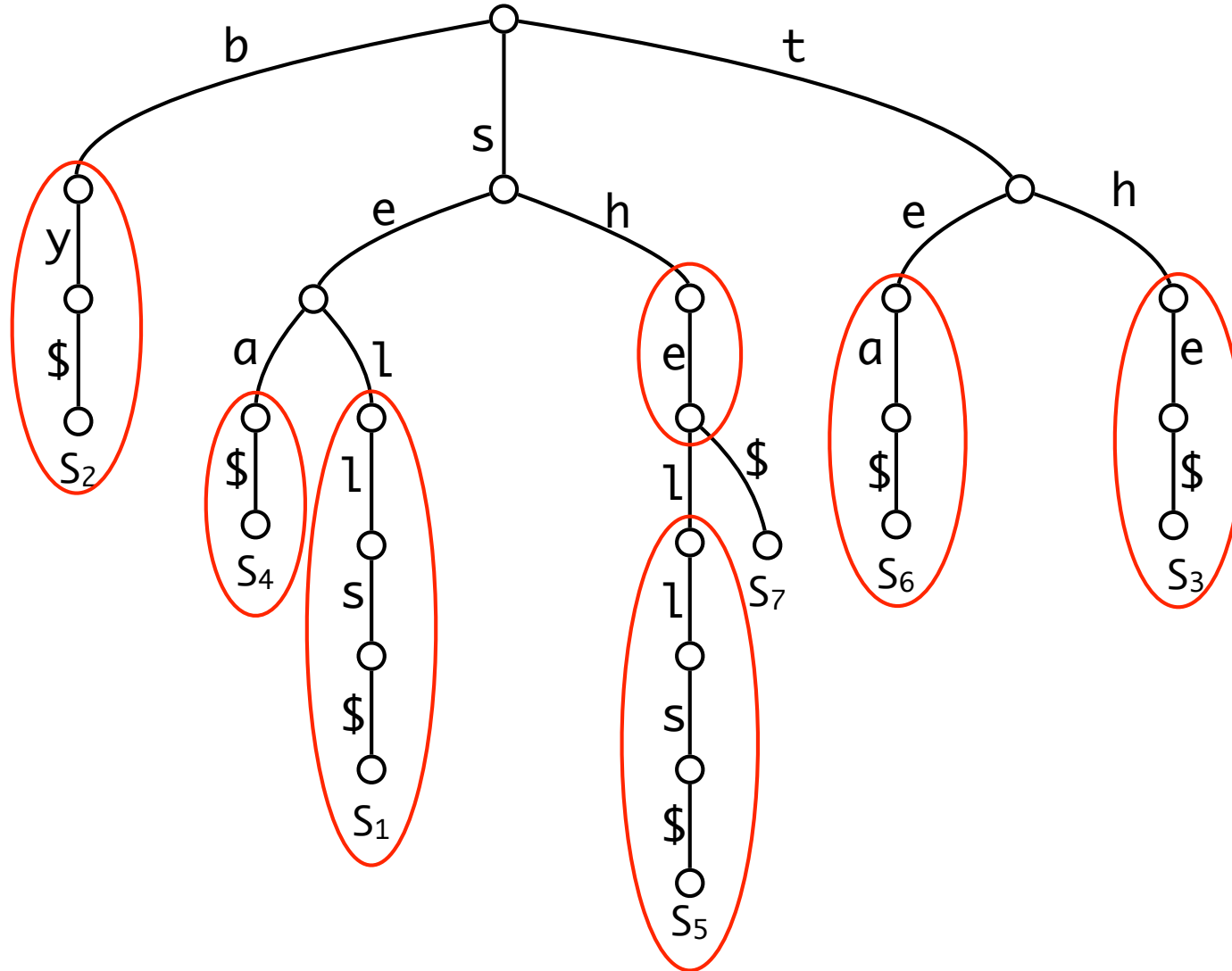
- Time for prefix search: $O(m)$ + time to report all occurrences.

Could be large!!

- Want: $O(m)$ + $O(\text{number of occurrences})$

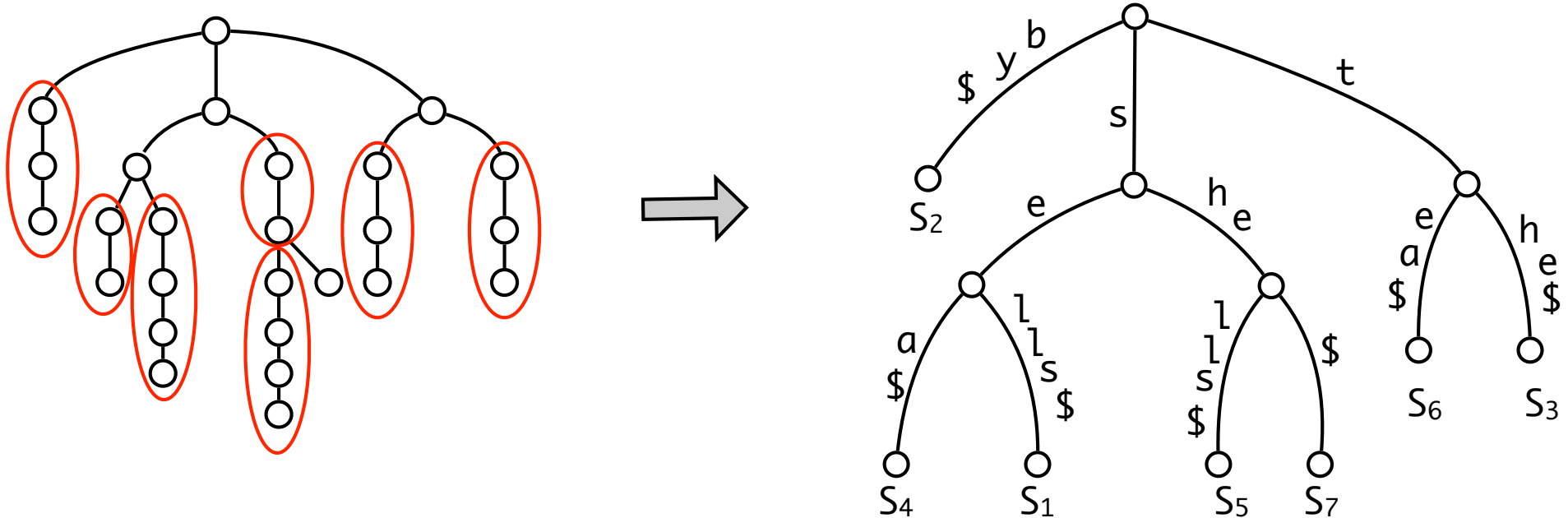
Tries

- **Compact trie.** Chains of nodes with a single child is merged into a single edge.



Tries

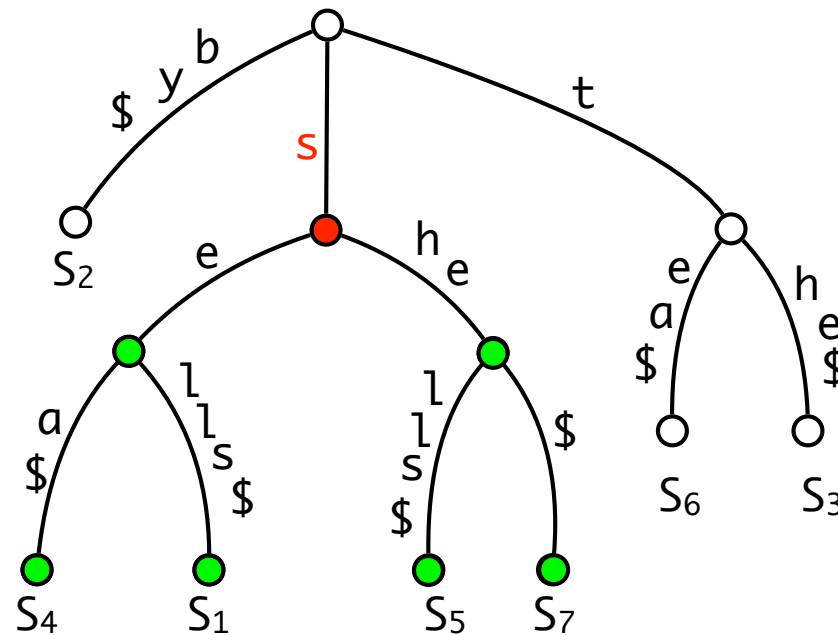
- **Compact trie.** Chains of nodes with a single child is merged into a single edge.



- **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 (except possibly the root) has at least 2 and at most d children.
 - T has s leaves
 - The number of nodes in T is $\leq 2s$.

Prefix Search in Compact Trie

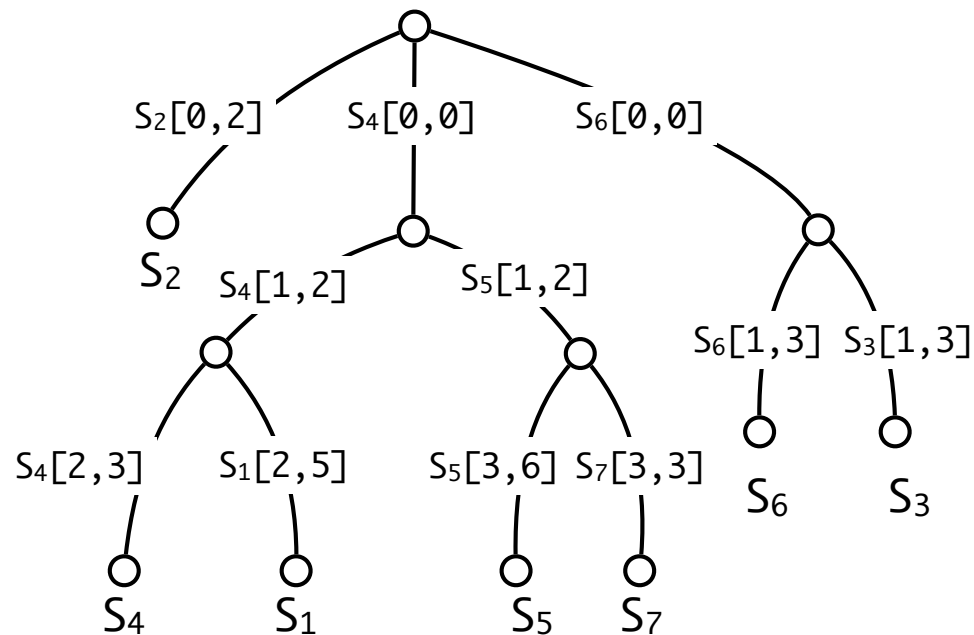
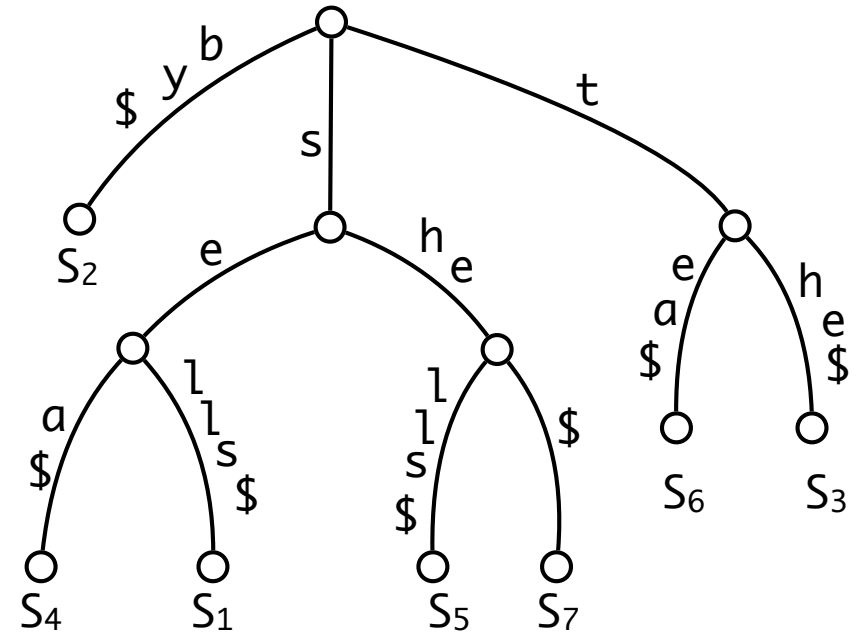
- Prefix search: return all words in the trie starting with “s”



- Time for prefix search: $O(m)$ + time to report all occurrences. = $O(m + \text{number of occurrences})$

Storing the Compact Trie

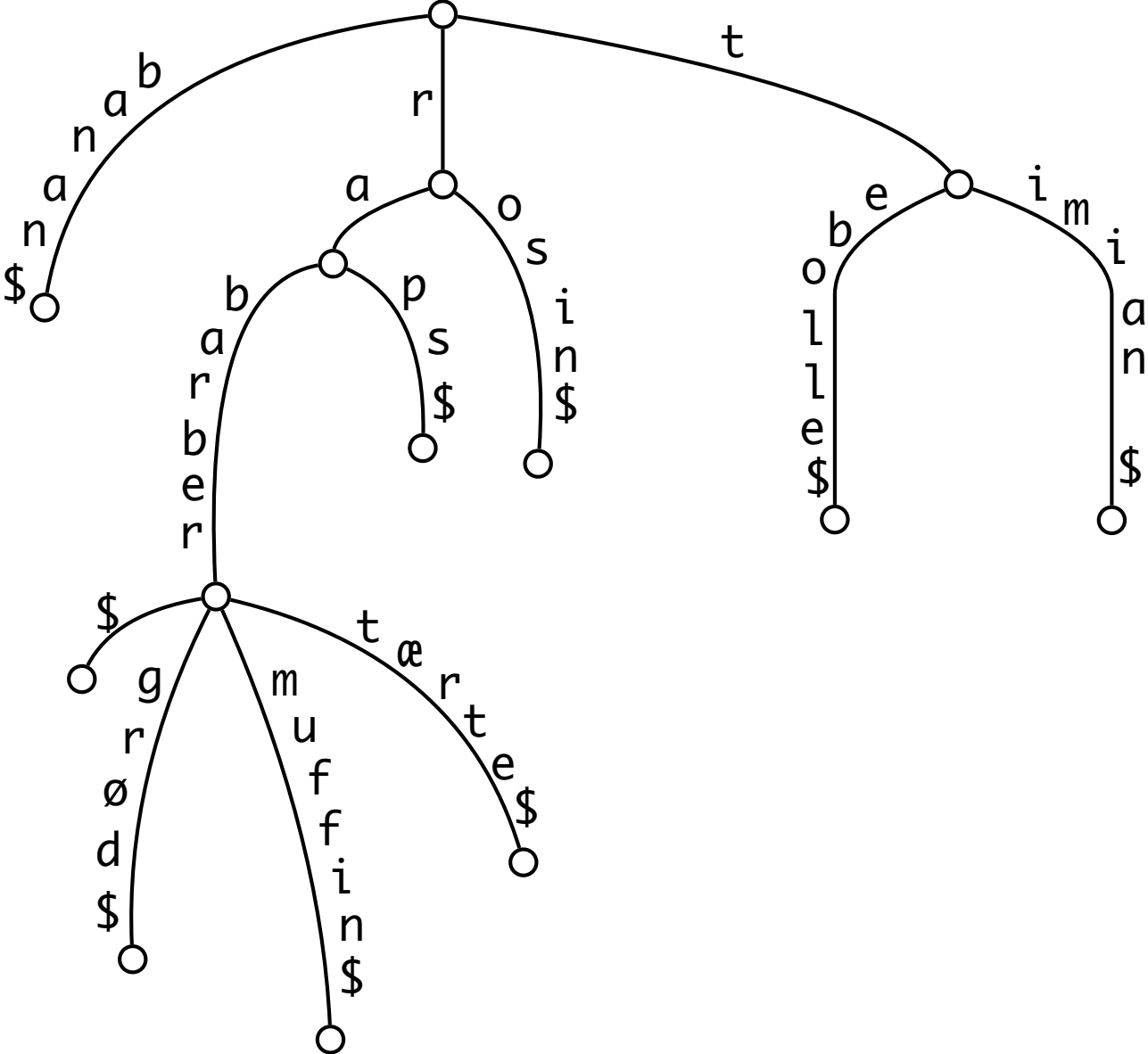
- If we store all edge labels explicitly, we still use $O(n)$ space.
- Instead store edge labels as reference into the strings (strings are stored).



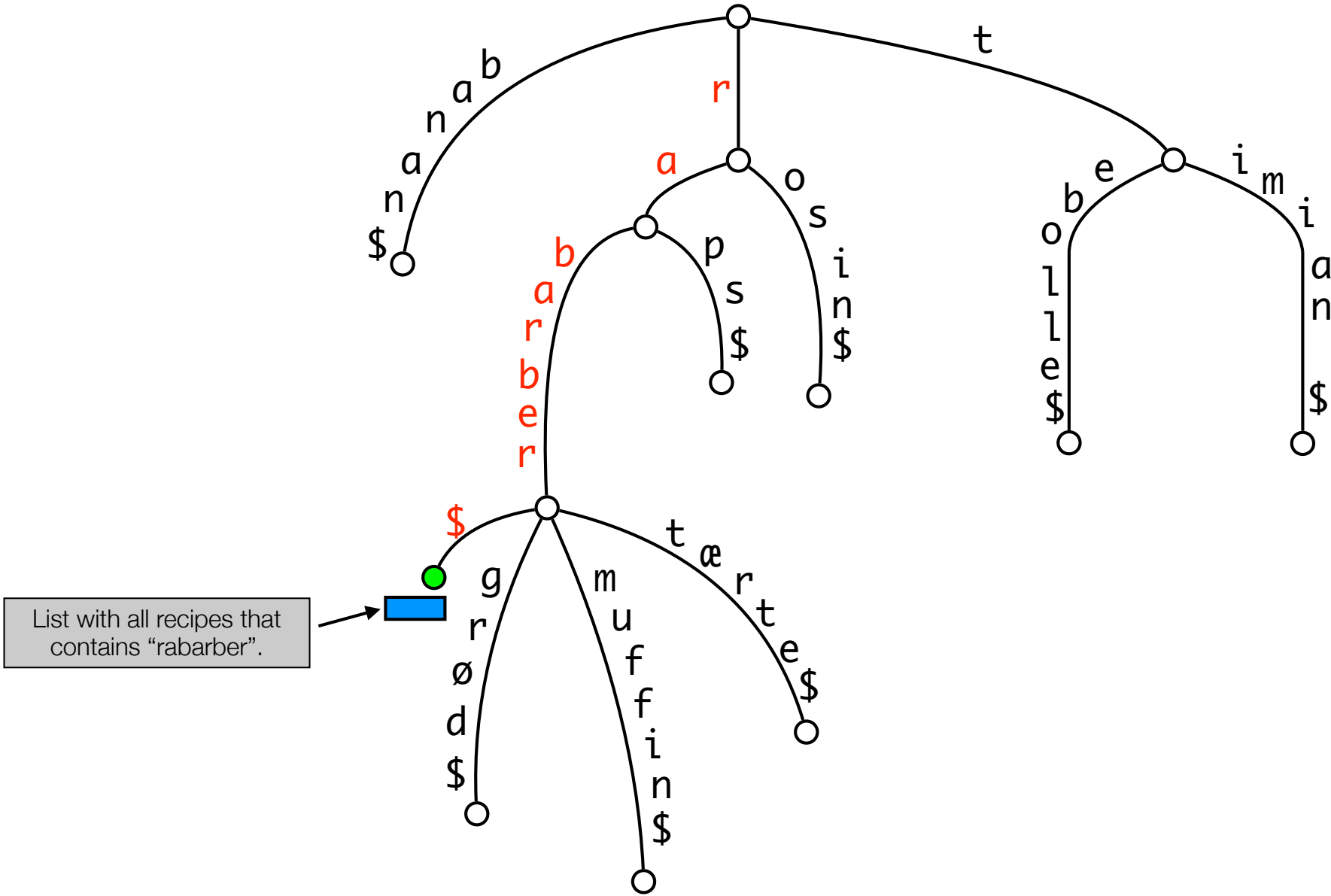
Trie

- Time and space for a compact trie of s strings of total length n over a constant size alphabet.
 - $O(m)$ for searching for a pattern of length m .
 - $O(m + \text{occ})$ for prefix search, where $\text{occ} = \#\text{occurrences}$
 - $O(s)$ space (in addition to the input strings).
 - Preprocessing time: $O(n)$

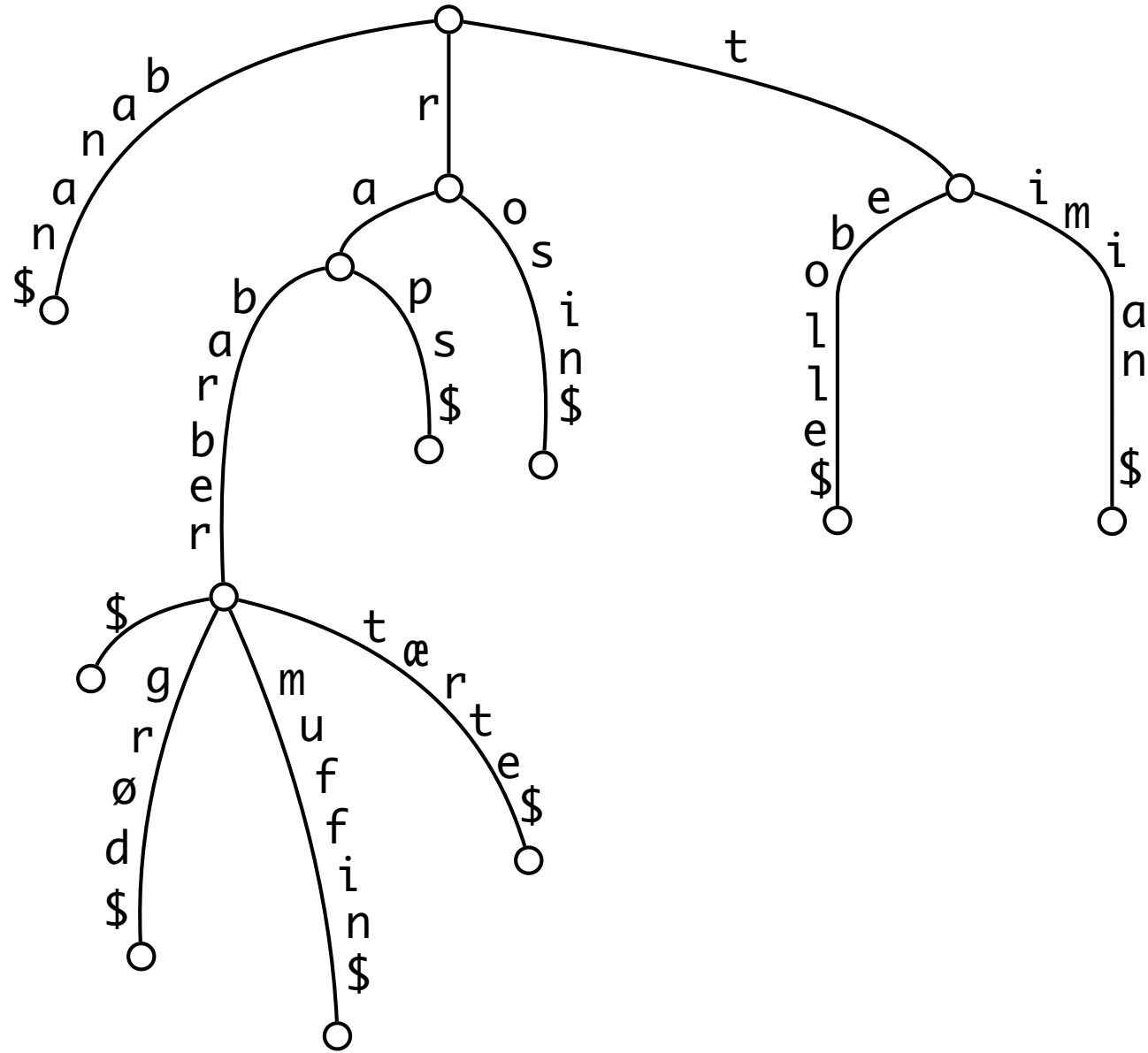
Trie



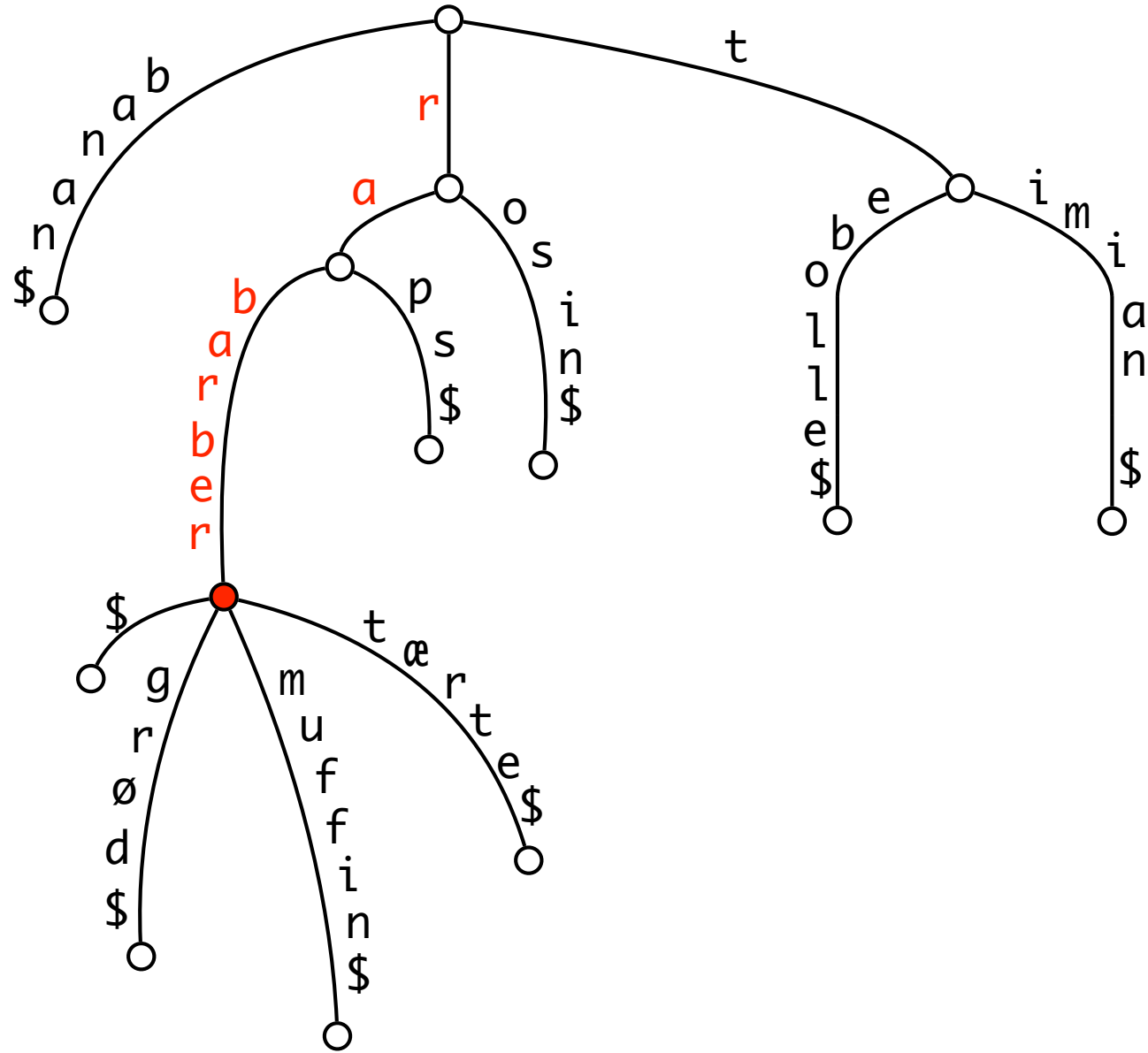
Inverted Index



Tries: Prefix search



Tries: Prefix search



Tries: Prefix search

