https://algs4.cs.princeton.edu

5.2 TRIES

- *string symbol tables*
- *R-way tries*
- *ternary search tries*
- *character-based operations*

https://algs4.cs.princeton.edu

5.2 Tries

- *string symbol tables*
- *R-way tries*
- *ternary search tries*
- *character-based operations*

Symbol tables: performance summary

Review. Two classic symbol tables: red–black BSTs and hash tables.

implementation	frequency of core operations			ordered operations	core operations on keys
	search	insert	delete		
red–black BST	$\log n$	$\log n$	$\log n$	✔	`compareTo()`
hash table	1 †	1 †	1 †		`equals()` `hashCode()`

† under uniform hashing assumption

Q. Can we do better?

A. Yes, if we can avoid examining the entire key, as with string sorting.

String symbol tables: performance summary

Goal (for string keys). Faster than hashing, more flexible than BSTs.

Benchmark. Count distinct words in a text file.

exchange rate:
L character accesses per hash
around $\Theta(\log n)$ character accesses per string compare

	character accesses (typical case)				count distinct	
implementation	search hit	search miss	insert	space (references)	moby.txt	actors.txt
red–black BST	$L + \log^2 n$	$\log^2 n$	$\log^2 n$	$4n$	1.4	97.4
hashing (linear probing)	L	L	L	$4n$ to $16n$	0.76	40.6

n = number of key–value pairs

L = length of key

R = radix

file	size	words	distinct
moby.txt	1.2 MB	210 K	32 K
actors.txt	82 MB	11.4 M	900 K

https://algs4.cs.princeton.edu

5.2 Tries

- string symbol tables
- *R-way tries*
- ternary search tries
- character-based operations

Tries

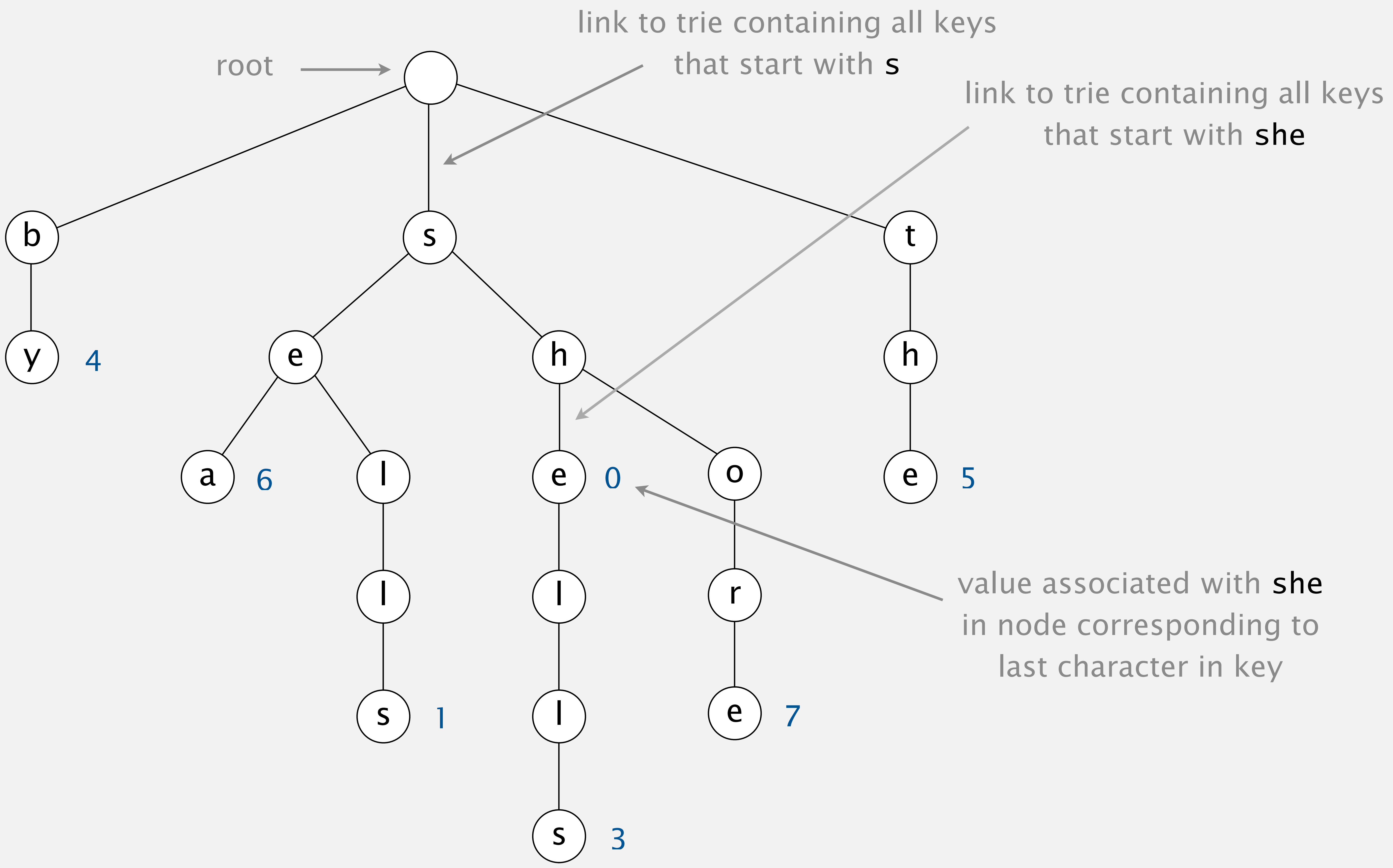
Etymology. [from retrieval, but pronounced "try"]

Tries

Abstract trie.

- Store characters in nodes (not keys).
- Each node has up to R children, one for each possible character in alphabet.

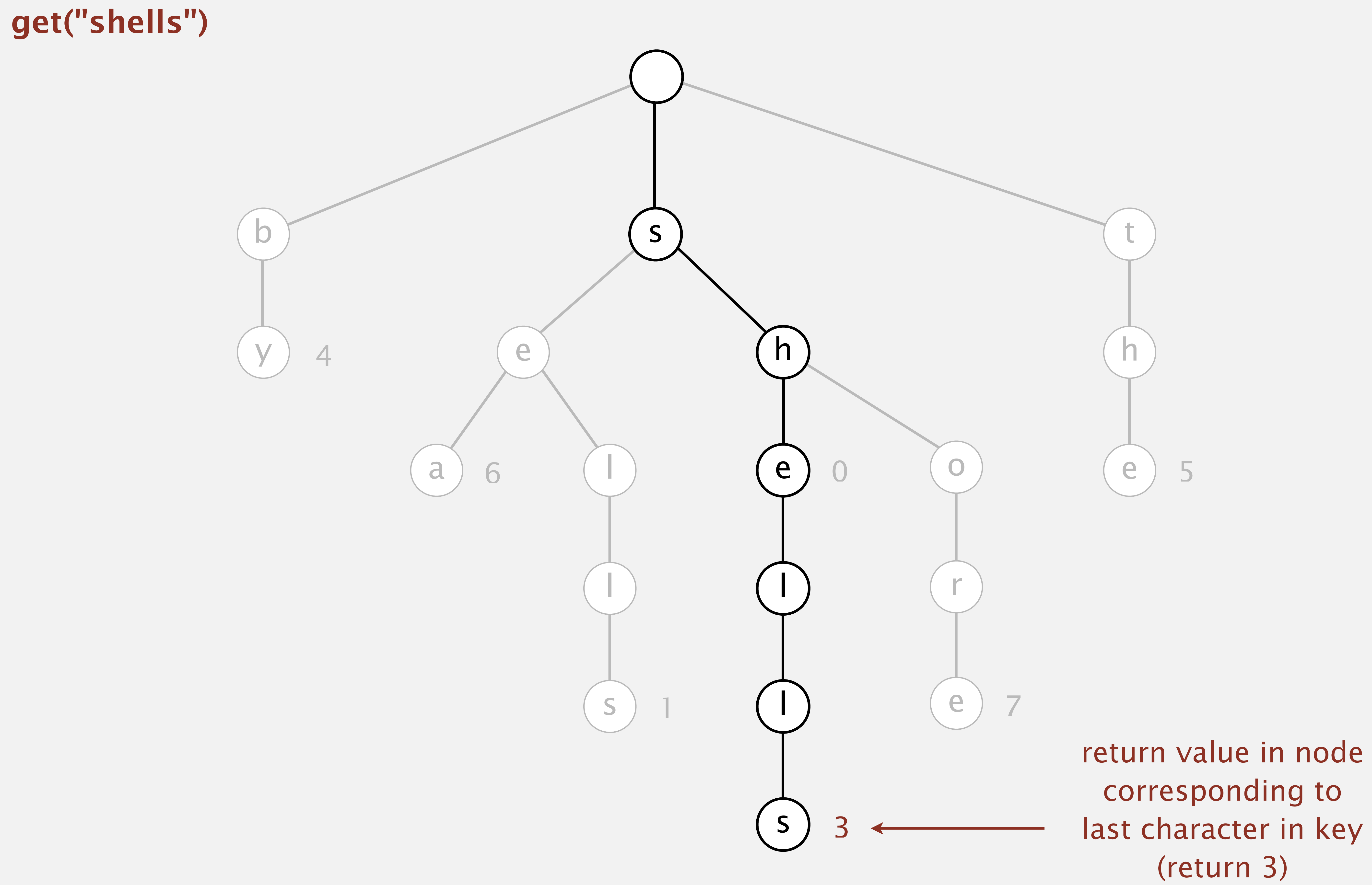
key	value
by	4
sea	6
sells	1
she	0
shells	3
shore	7
the	5



Tries: search hit

Follow links corresponding to each character in the key.

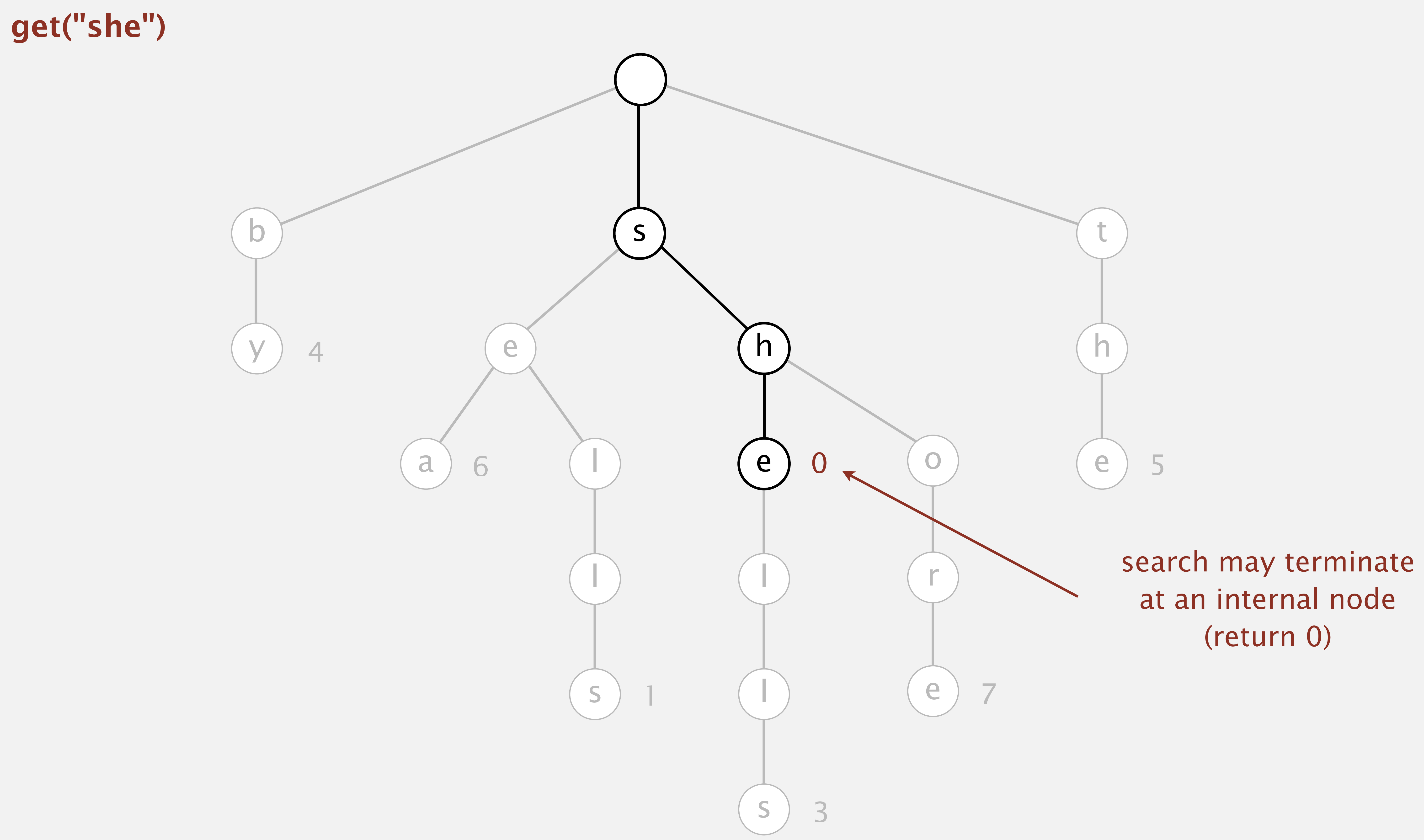
- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.



Tries: search hit

Follow links corresponding to each character in the key.

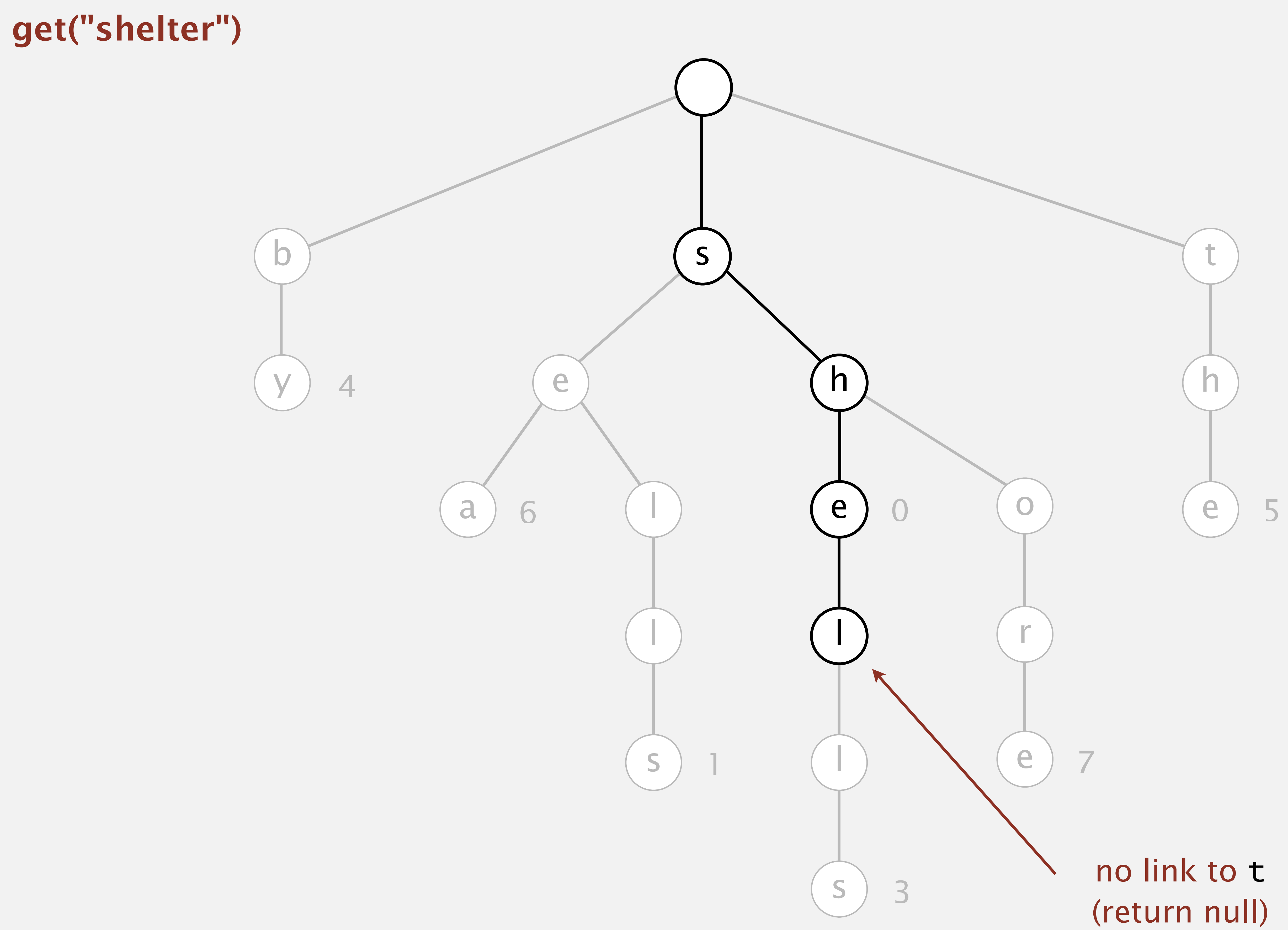
- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.



Tries: search miss

Follow links corresponding to each character in the key.

- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.

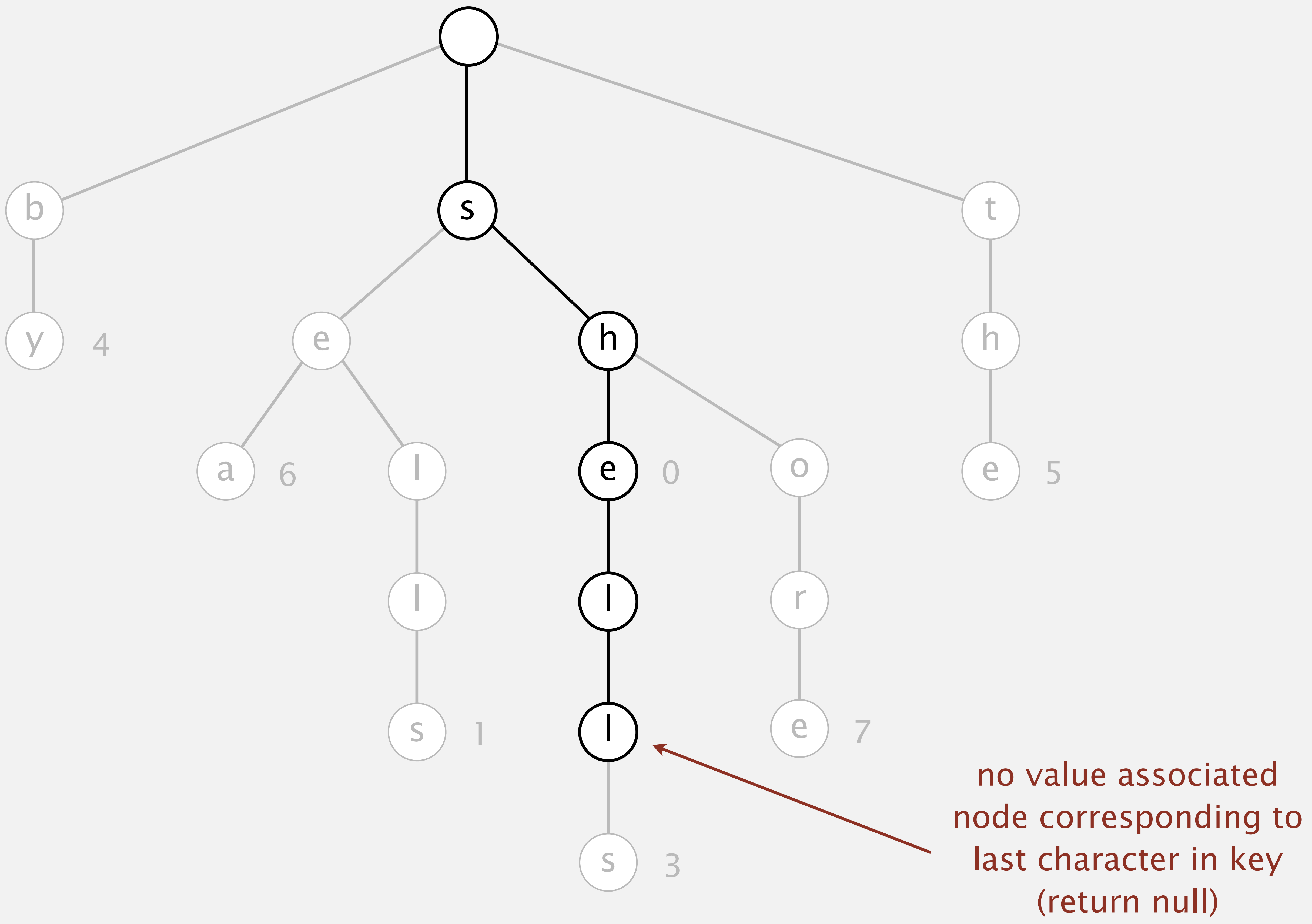


Tries: search miss

Follow links corresponding to each character in the key.

- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.

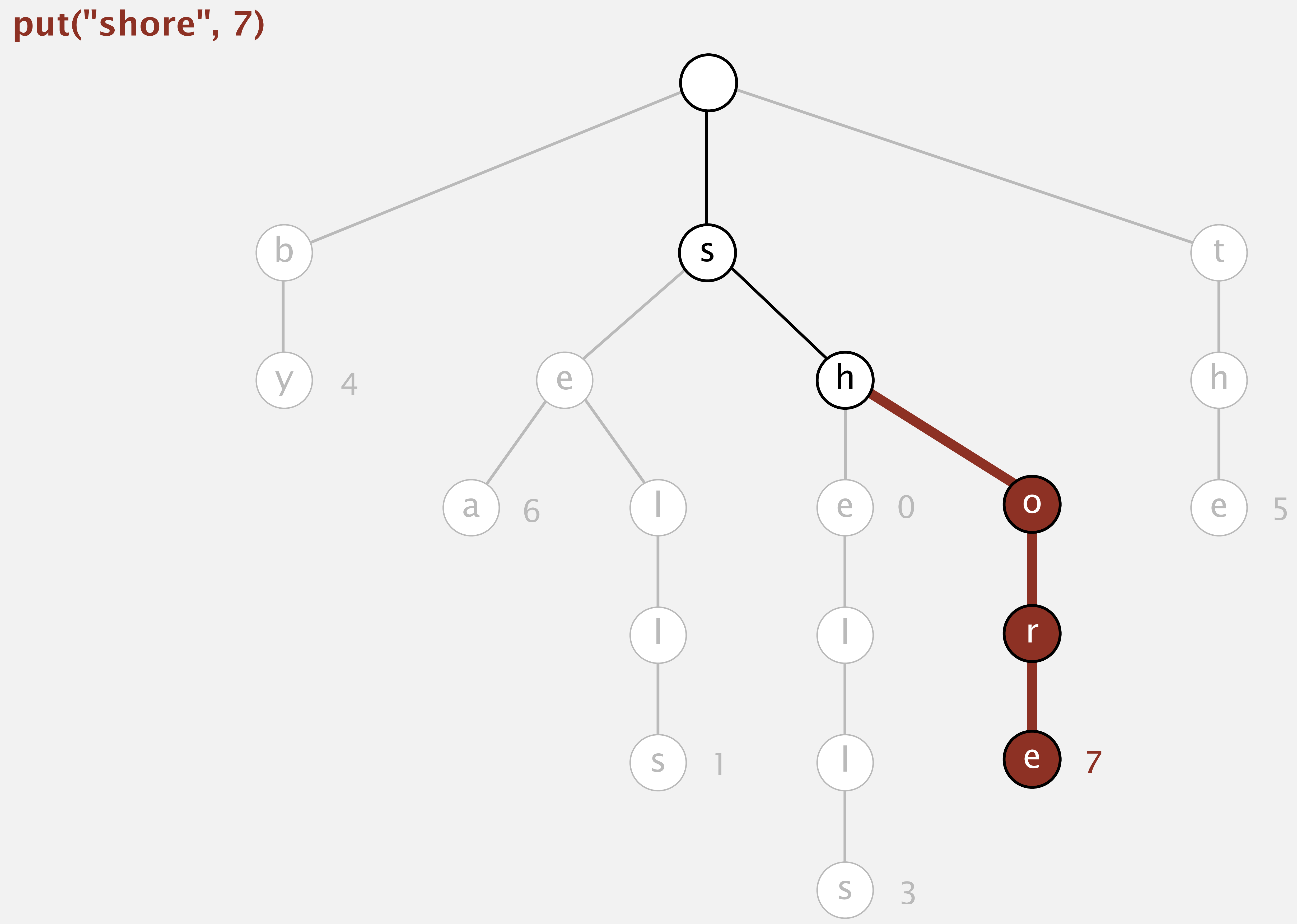
get("shell")



Tries: insertion

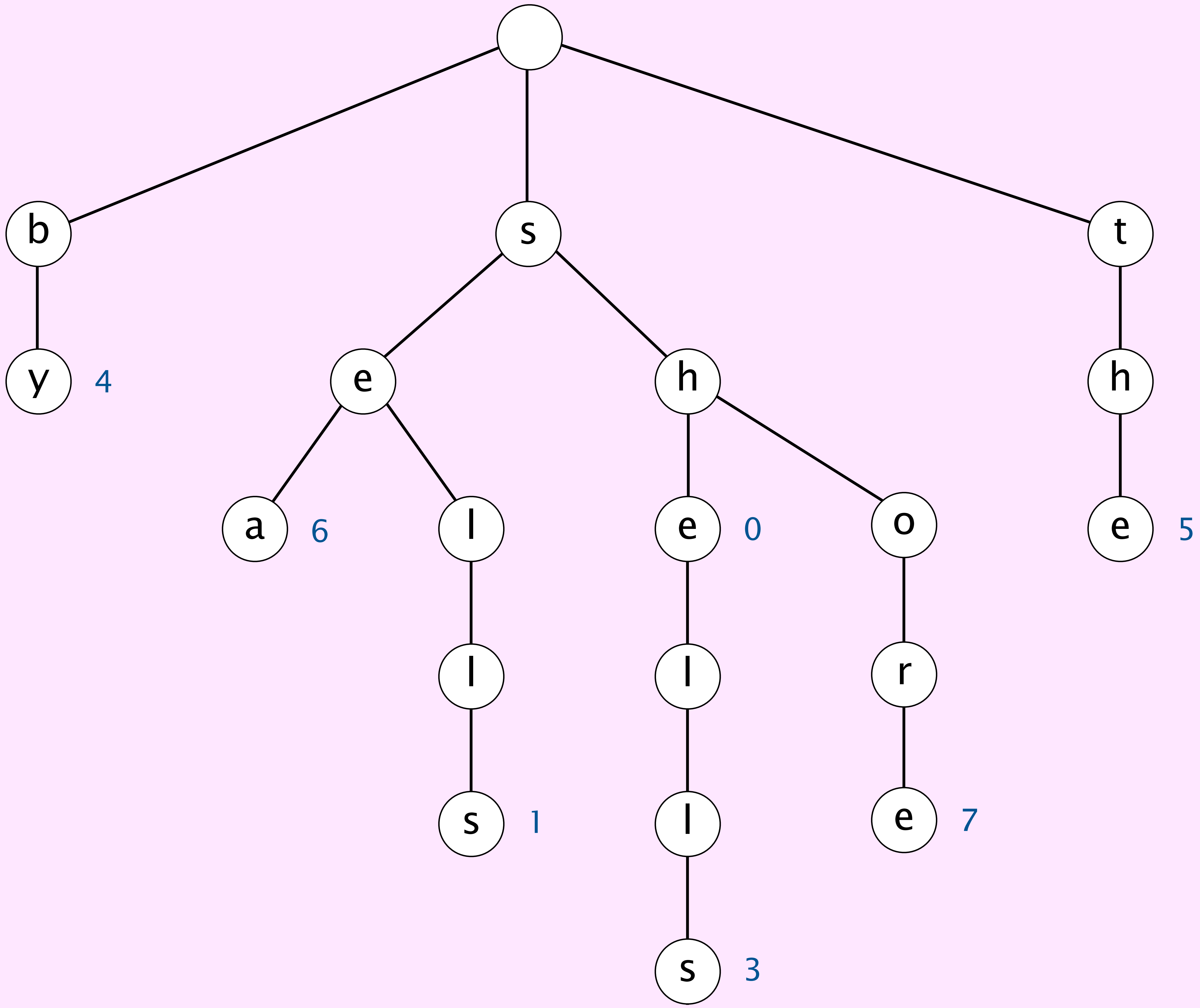
Follow links corresponding to each character in the key.

- Encounter a null link: create new node.
- Encounter the last character of the key: set value in that node.



Trie construction demo

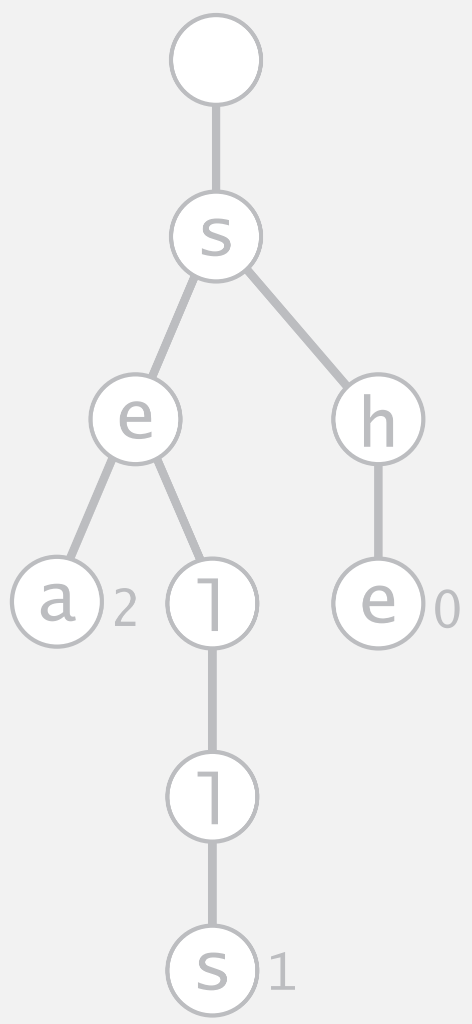
trie

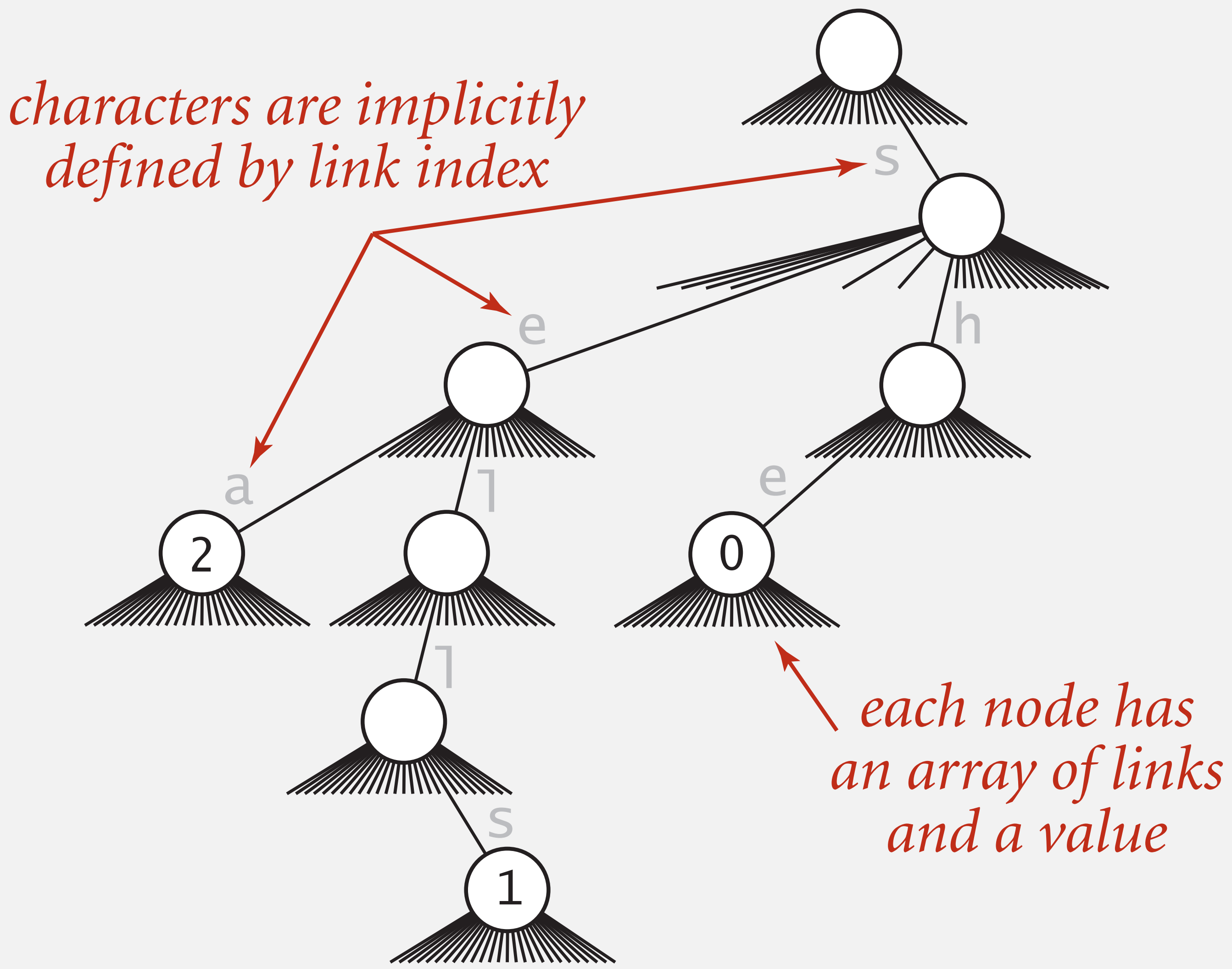


R-way tries: Java representation

Node. A value, plus references to R nodes.

```
private static class Node
{
   private Object val;   ← no generic array creation
   private Node[] next = new Node[R];
}
```





Remark. An R-way trie stores neither keys nor characters explicitly.

R-way tries: Java implementation

```
public class TrieST<Value>
{
   private static final int R = 256;   ←—— extended ASCII
   private Node root = new Node();

   private static class Node
   {  /* see previous slide */  }

   public void put(String key, Value val)
   {  root = put(root, key, val, 0);  }

   private Node put(Node x, String key, Value val, int d)
   {
      if (x == null) x = new Node();
      if (d == key.length()) { x.val = val; return x; }
      char c = key.charAt(d);
      x.next[c] = put(x.next[c], key, val, d+1);
      return x;
   }

   private Value get(String key)
   {  /* similar, see book or booksite */  }
}
```

R-way trie: performance

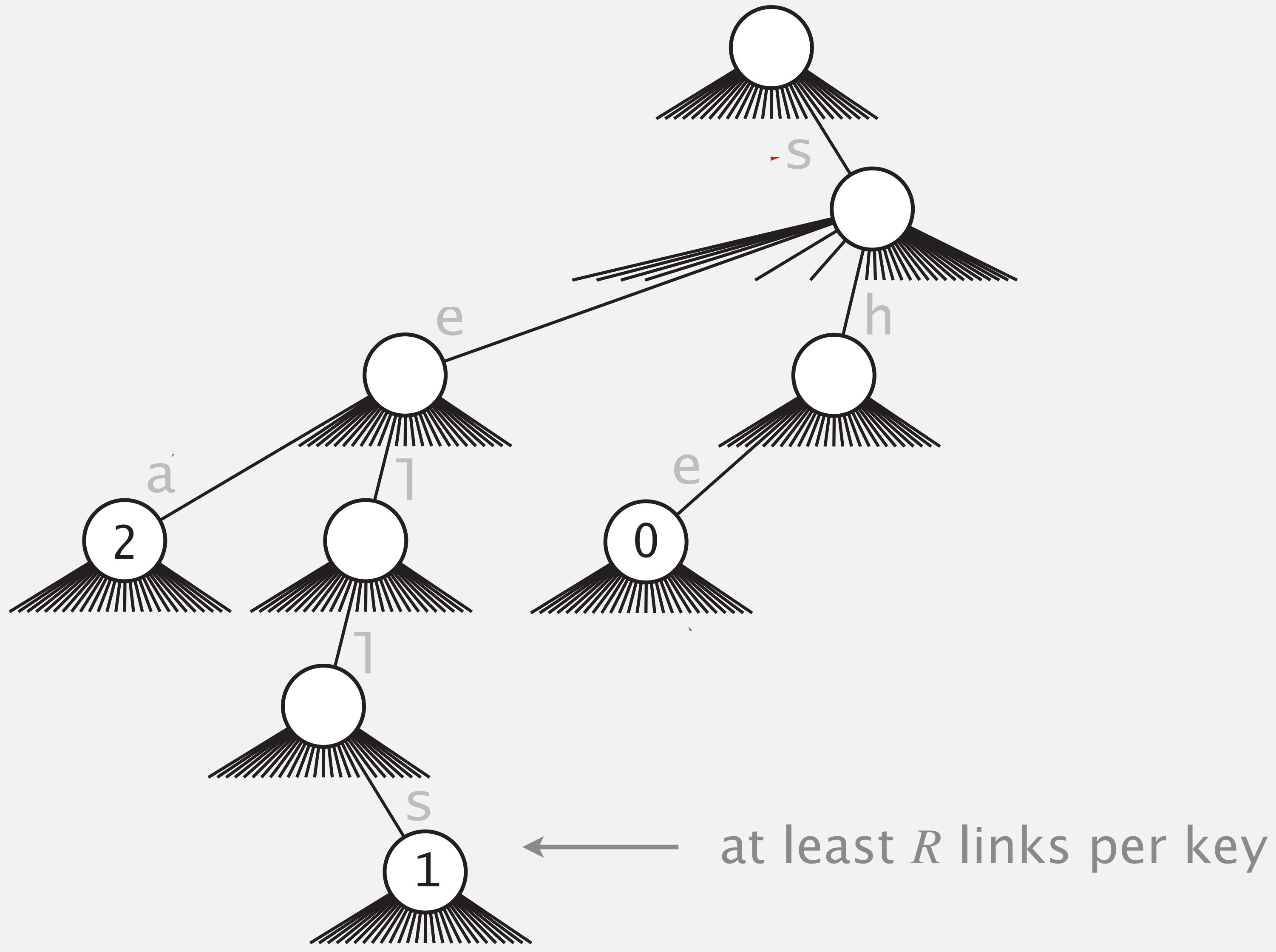
Parameters. n = number of key–value pairs; L = length of key; R = alphabet size.

Search hit. $\Theta(L)$.

Search miss (worst case). $\Theta(L)$.

Search miss (typical case). $\Theta(\log_R n)$. ← sublinear in L

Space. At least $\Theta(nR)$ space.



Bottom line. Fast search hit; even faster search miss; but wastes space.

Trie quiz 1

What is worst-case running time to insert a key of length L into an R-way trie that contains n key-value pairs?

n = number of key–value pairs

L = length of key

R = alphabet size

A. $\Theta(L)$

B. $\Theta(R + L)$

C. $\Theta(n + L)$

D. $\Theta(R\ L)$

String symbol table implementations cost summary

	character accesses (typical case)				count distinct	
implementation	search hit	search miss	insert	space (references)	moby.txt	actors.txt
red-black BST	$L + \log^2 n$	$\log^2 n$	$\log^2 n$	$4n$	1.4	97.4
hashing (linear probing)	L	L	L	$4n$ to $16n$	0.76	40.6
R-way trie	L	$\log_R n$	$R + L$	$(R+1)\ n$	1.12	*out of memory*

R-way trie.

- Method of choice for small R.
- Effective for medium R.
- Too much memory for large R.

Challenge. Use less memory, e.g., a 65,536-way trie for Unicode!

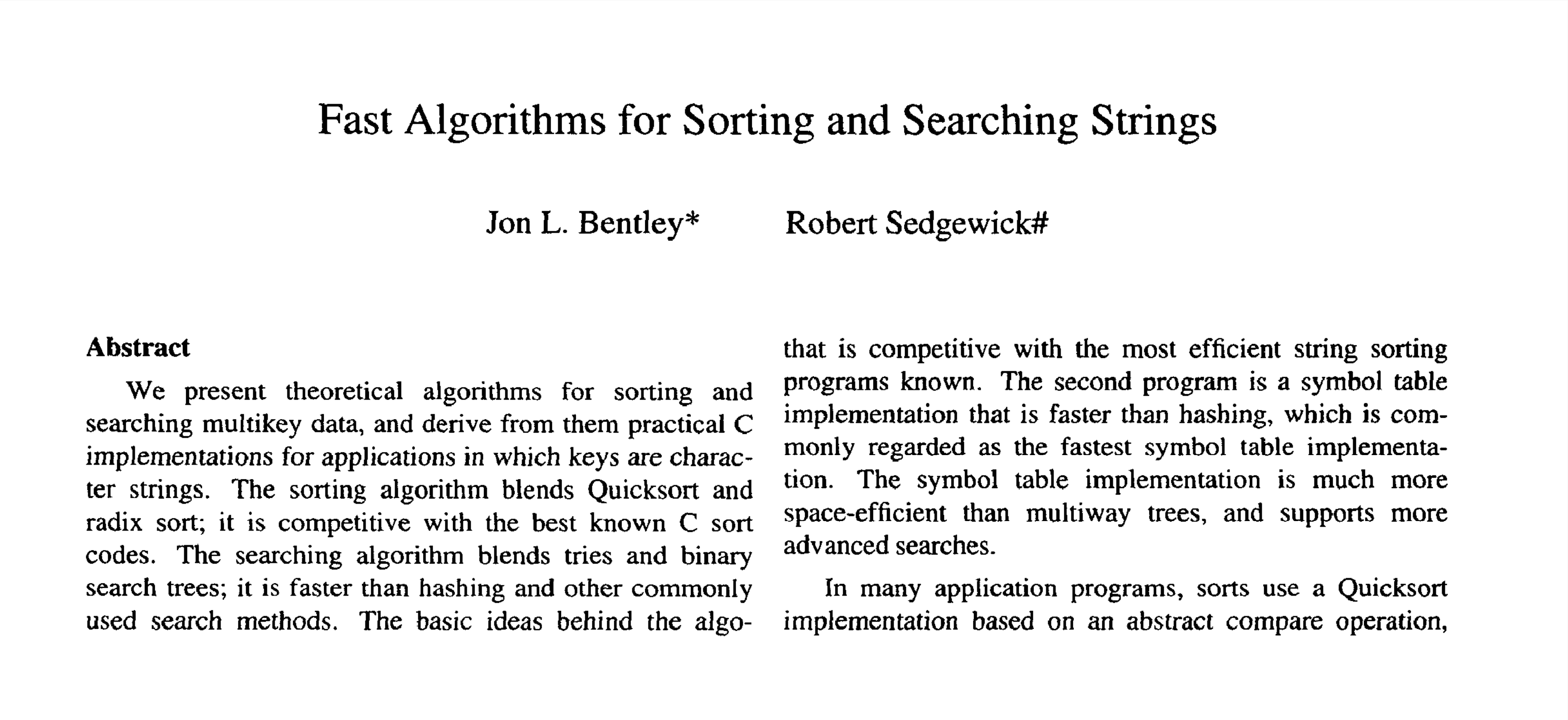
https://algs4.cs.princeton.edu

5.2 Tries

- *string symbol tables*
- *R-way tries*
- *ternary search tries*
- *character-based operations*

Ternary search tries

- Store characters and values in nodes (not keys).
- Each node has three children: smaller (left), equal (middle), larger (right).



Fast Algorithms for Sorting and Searching Strings

Jon L. Bentley* Robert Sedgewick#

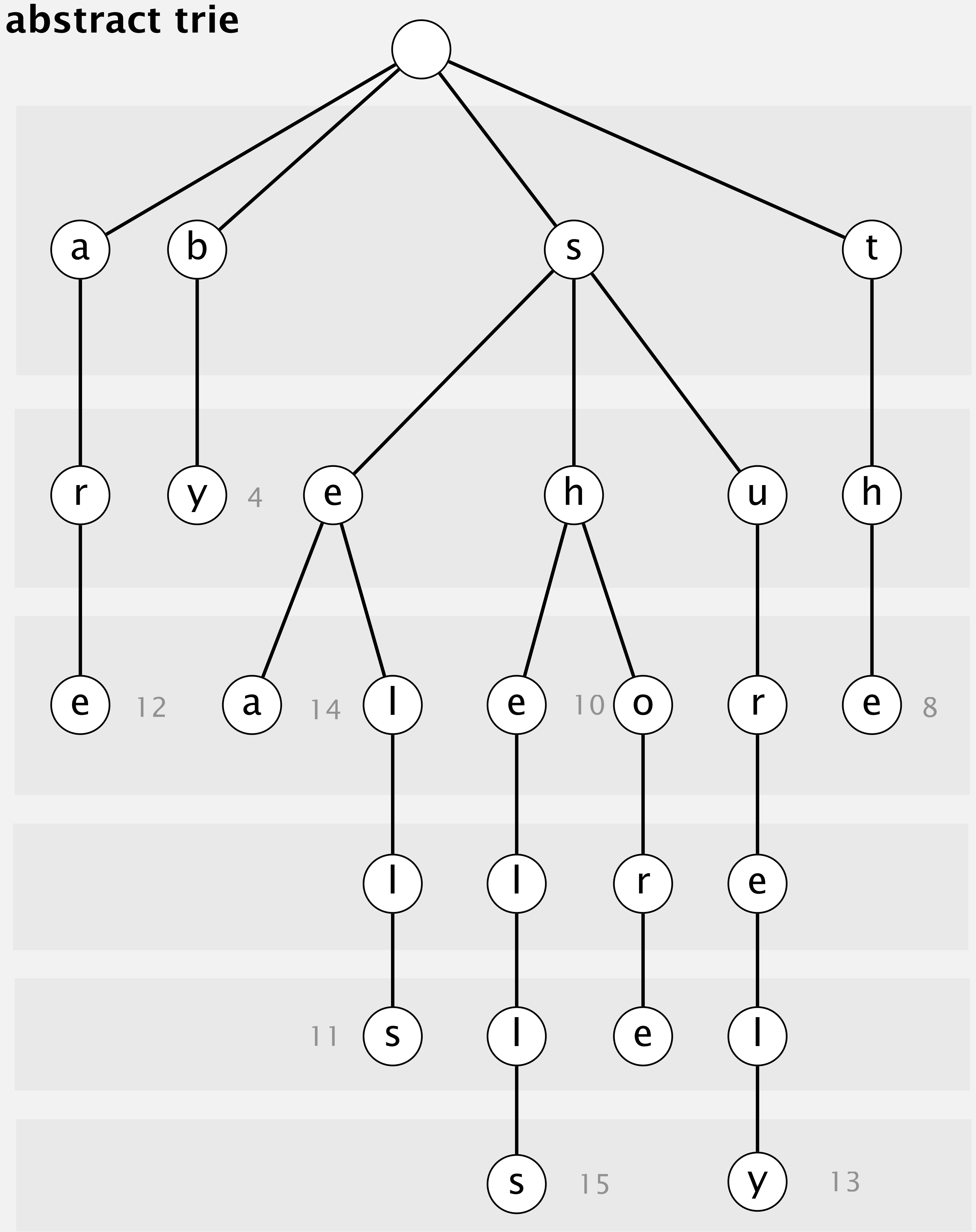
Abstract

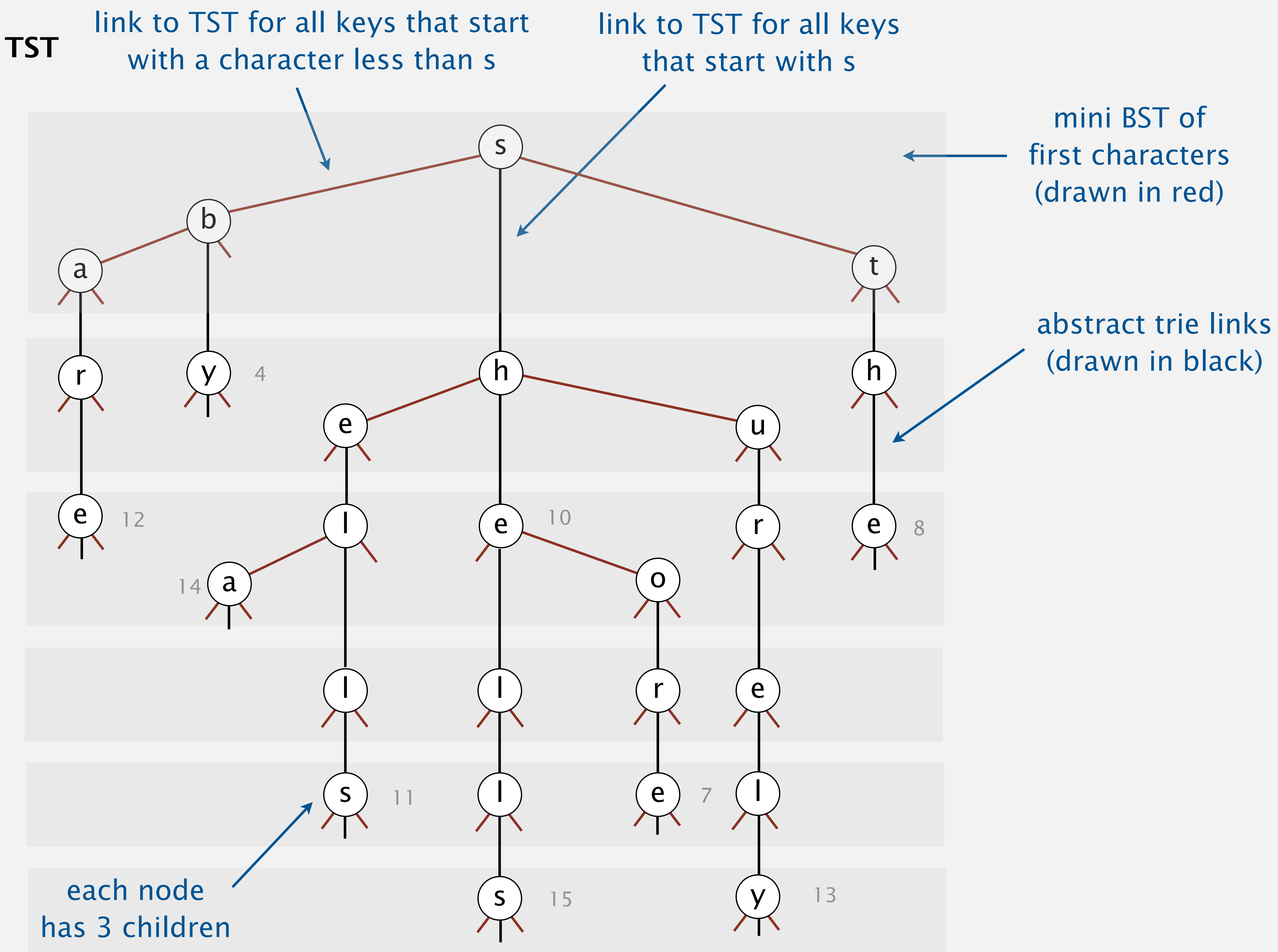
We present theoretical algorithms for sorting and searching multikey data, and derive from them practical C implementations for applications in which keys are character strings. The sorting algorithm blends Quicksort and radix sort; it is competitive with the best known C sort codes. The searching algorithm blends tries and binary search trees; it is faster than hashing and other commonly used search methods. The basic ideas behind the algo- that is competitive with the most efficient string sorting programs known. The second program is a symbol table implementation that is faster than hashing, which is commonly regarded as the fastest symbol table implementation. The symbol table implementation is much more space-efficient than multiway trees, and supports more advanced searches.

In many application programs, sorts use a Quicksort implementation based on an abstract compare operation,

Ternary search tries

- Store characters and values in nodes (not keys).
- Each node has three children: smaller (left), equal (middle), larger (right).





Trie quiz 2

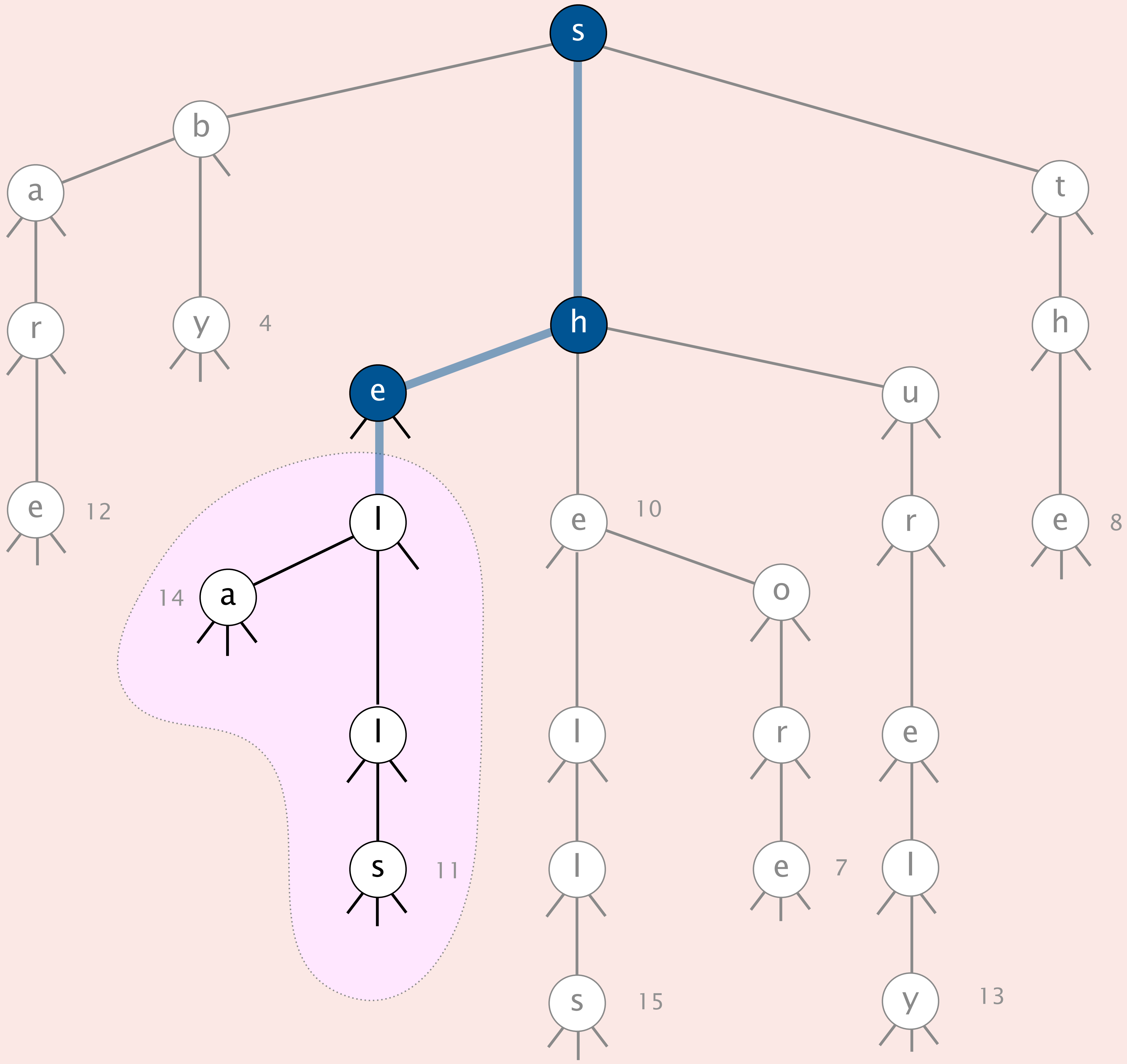
Which keys are stored in the designated subtrie of the TST?

A. Strings that start with `s`.

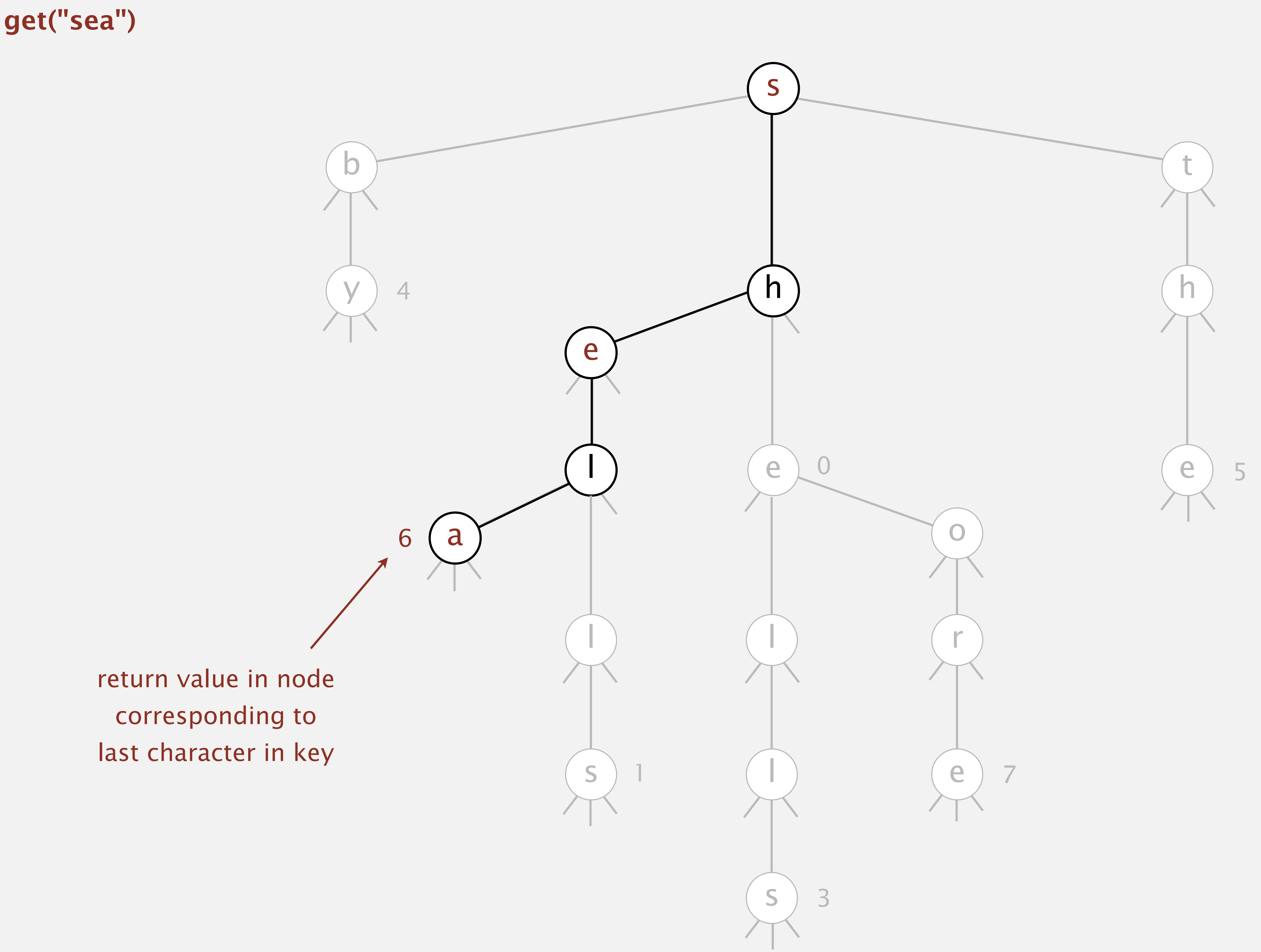
B. Strings that start with `se`.

C. Strings that start with `sh`.

D. Strings that start with `she`.

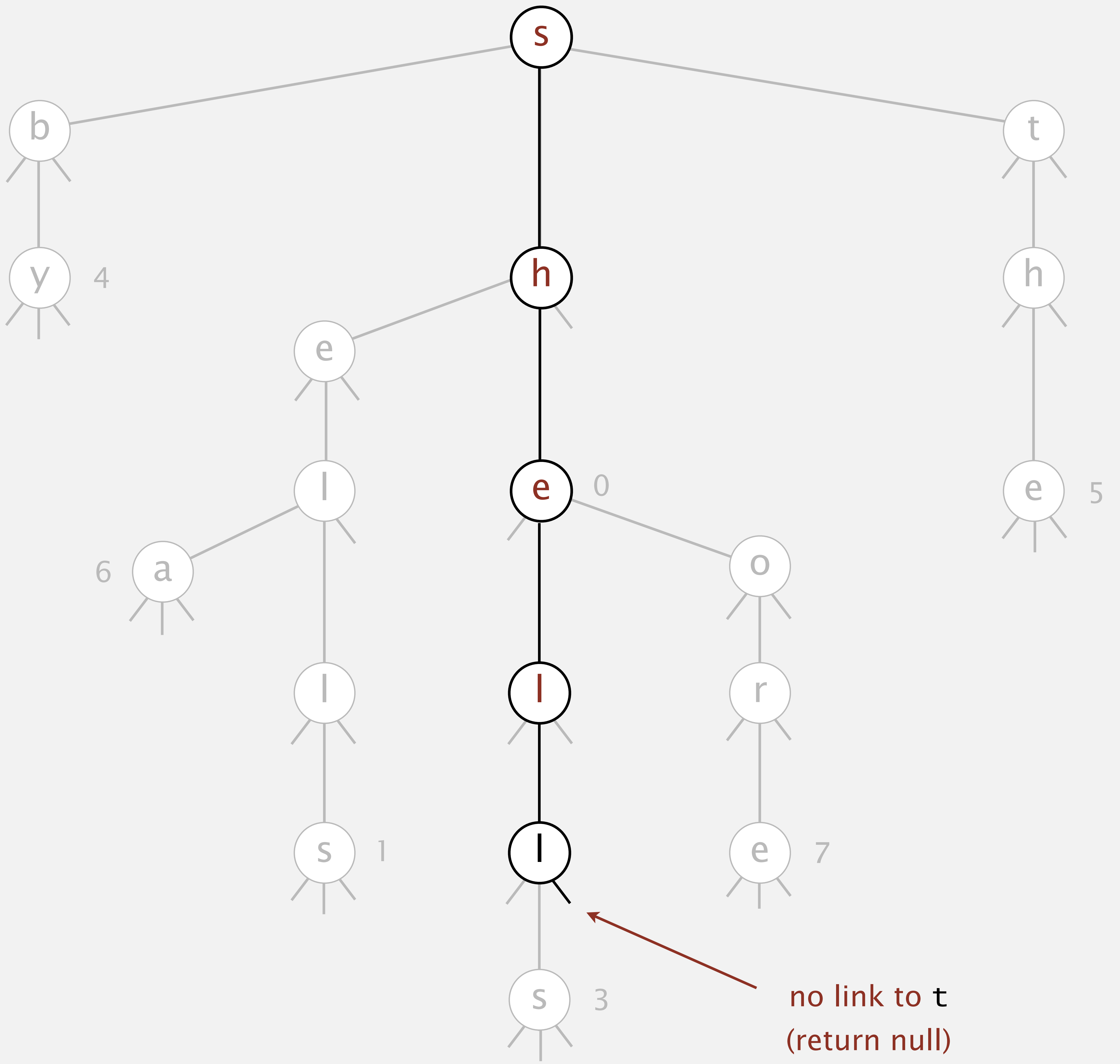


Search hit in a TST



Search miss in a TST

get("shelter")



Search in a TST

Compare key character to key in node and follow links accordingly:

- If less, go left.
- If greater, go right.
- If equal, go middle and advance to the next key character.

Search hit. Node where search ends has a non-null value.

Search miss. Either (1) reach a null link or (2) node where search ends has null value.

Trie quiz 3

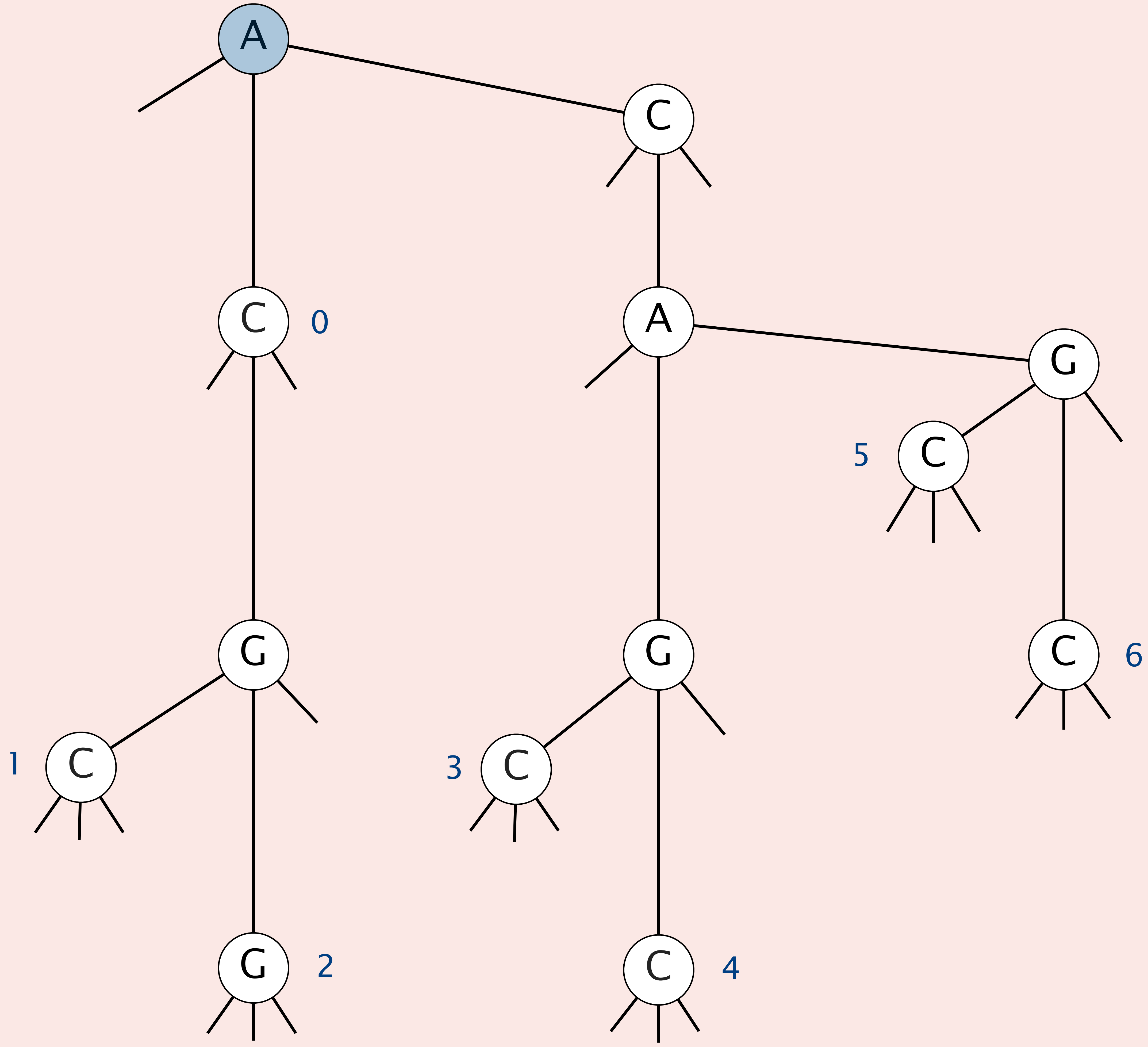
Which value is associated with the key CAC ?

A. 3

B. 4

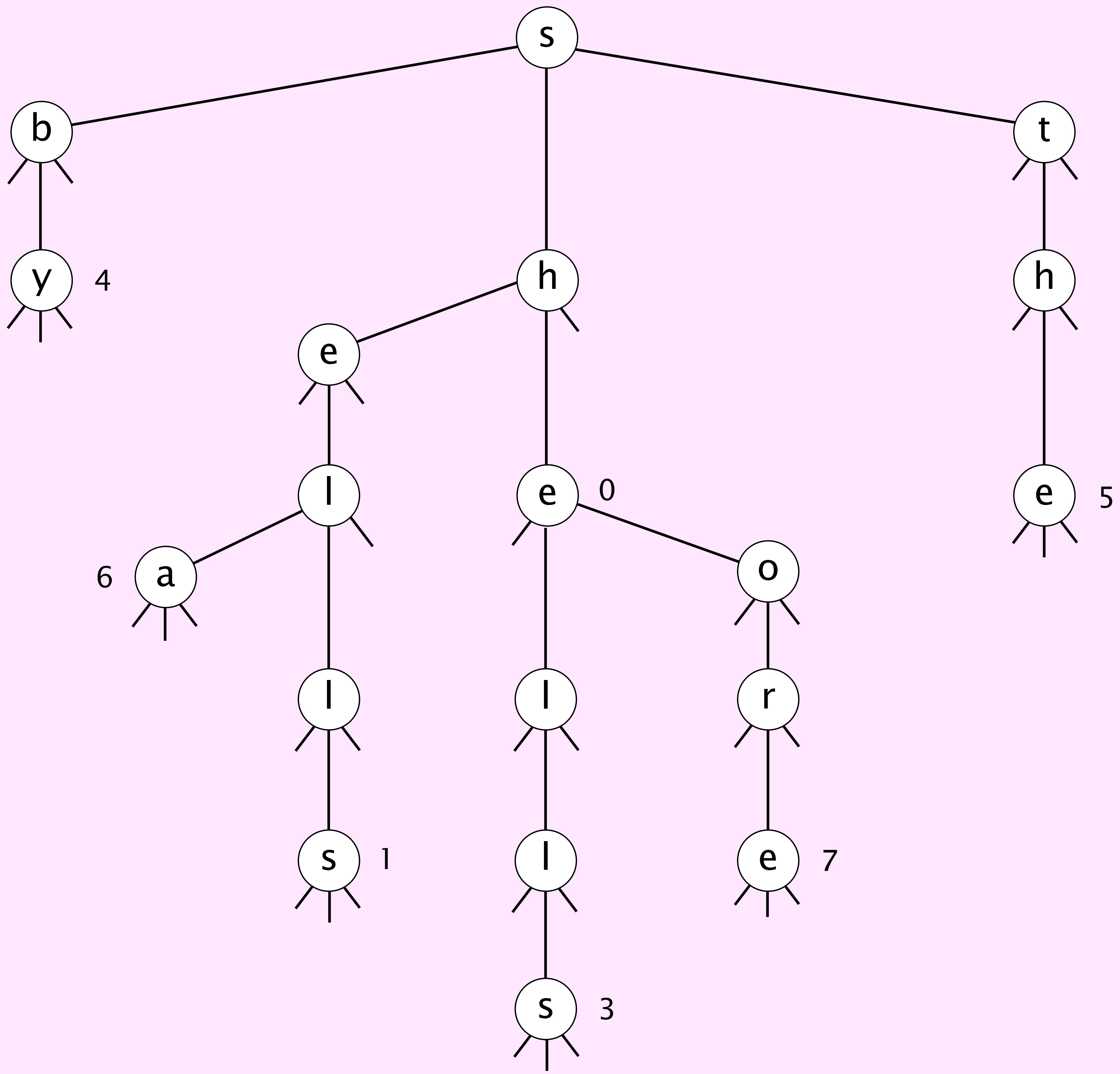
C. 5

D. *null*



Ternary search trie construction demo

ternary search trie



Trie quiz 4

In which subtrie would the key CCC be inserted?

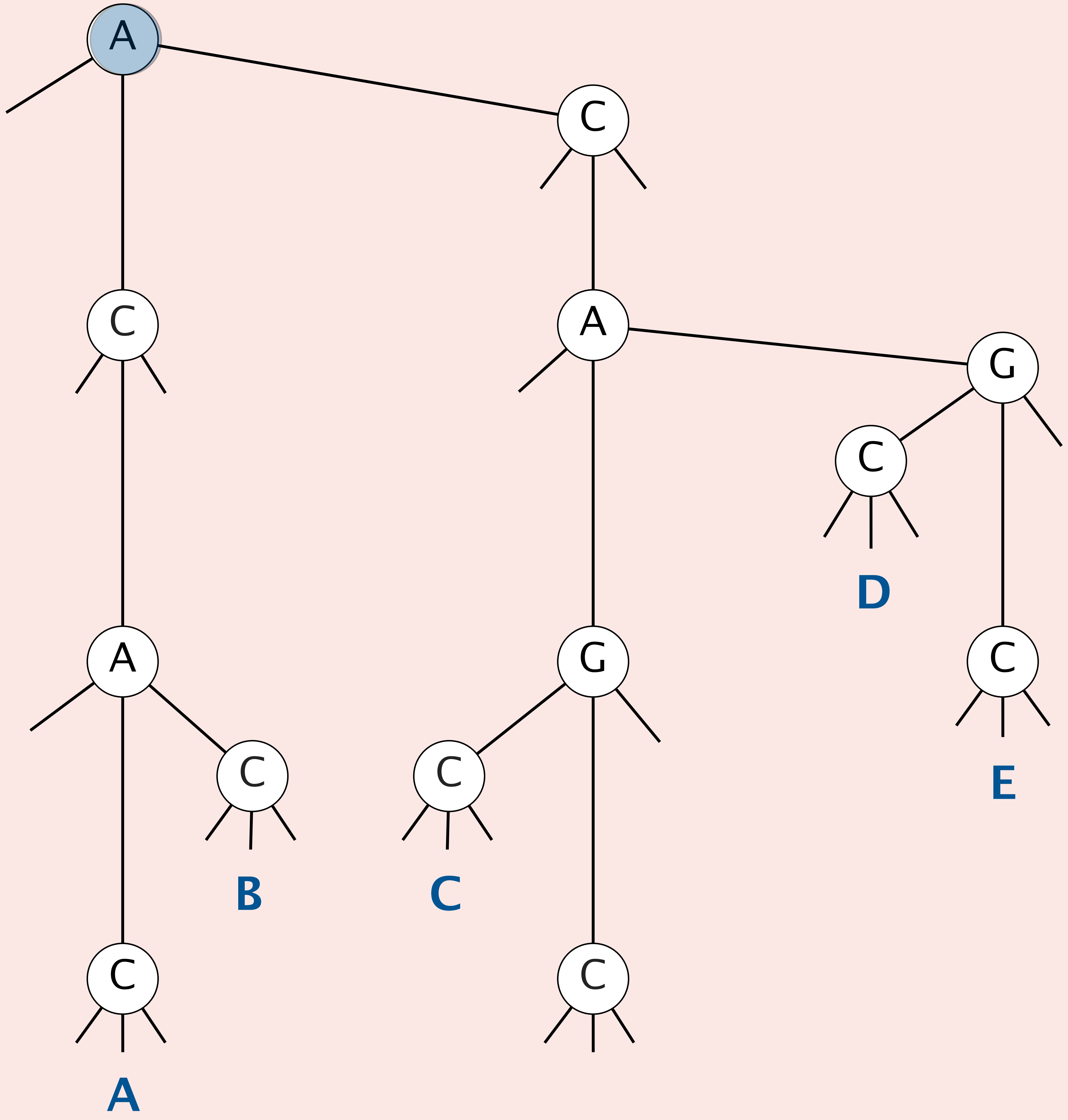
A.

B.

C.

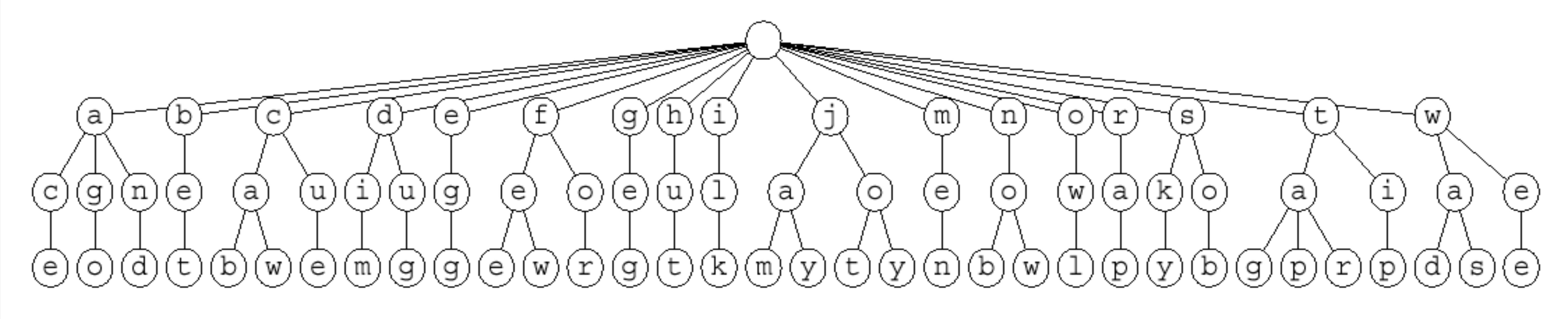
D.

E.



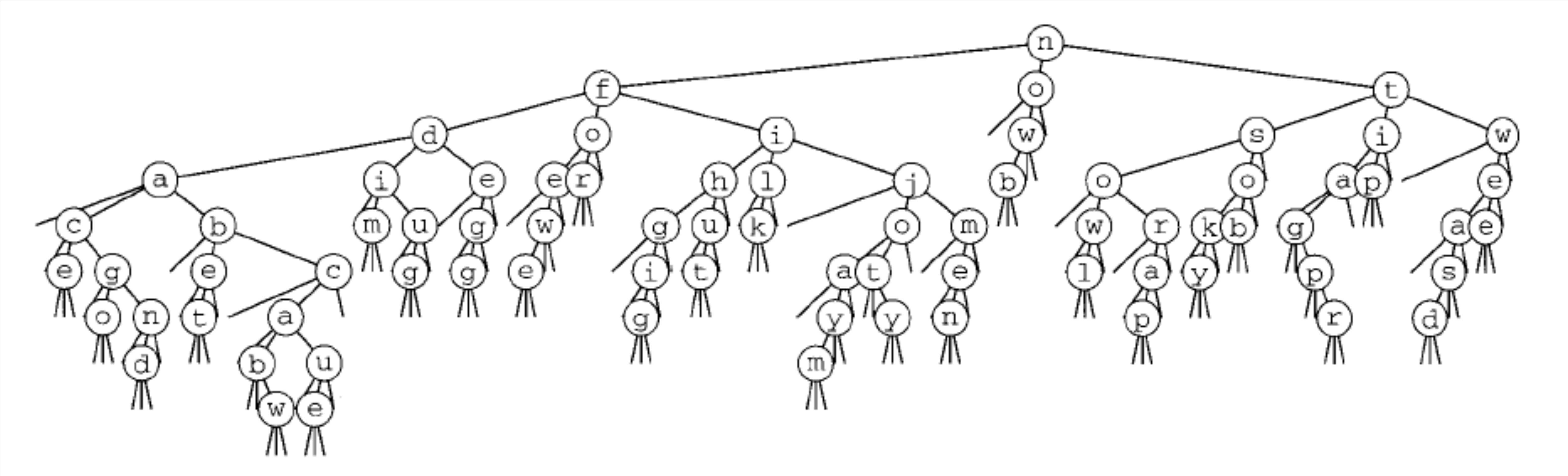
26-way trie vs. TST

26-way trie. 26 null links in each leaf.



26-way trie (1035 null links, not shown)

TST. 3 null links in each leaf.



TST (155 null links)

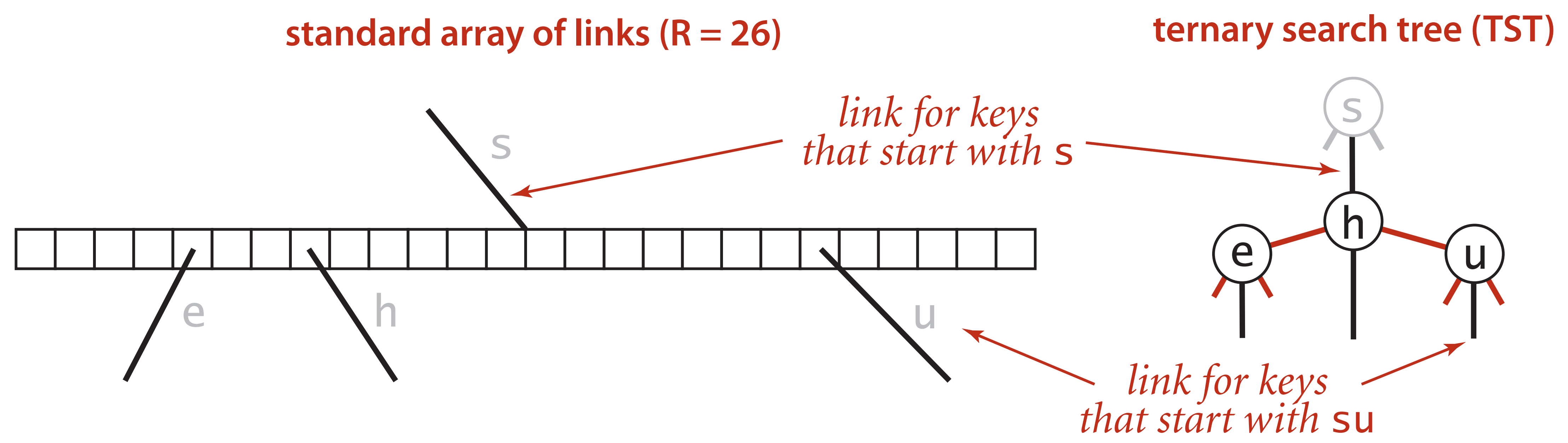
now
for
tip
ilk
dim
tag
jot
sob
nob
sky
hut
ace
bet
men
egg
few
jay
owl
joy
rap
gig
wee
was
cab
wad
caw
cue
fee
tap
ago
tar
jam
dug
and

TST representation in Java

A TST node is five fields:

- A value.
- A character.
- A reference to a left TST.
- A reference to a middle TST.
- A reference to a right TST.

```
private class Node
{
   private Value val;
   private char c;
   private Node left, mid, right;
}
```



Trie node representations

TST: Java implementation

```java
public class TST<Value>
{
   private Node root;
   private class Node
   {  /* see previous slide */  }

   public Value get(String key)
   {  return get(root, key, 0);  }

   private Value get(Node x, String key, int d)
   {
      if (x == null) return null;
      char c = key.charAt(d);
      if      (c < x.c)               return get(x.left,  key, d);
      else if (c > x.c)               return get(x.right, key, d);
      else if (d < key.length() - 1)  return get(x.mid,   key, d+1);
      else                            return x.val;
   }

   public void put(String Key, Value val)
   {  /* similar, see book or booksite */  }
}
```

String symbol table implementation cost summary

	character accesses (typical case)				count distinct	
implementation	**search hit**	**search miss**	**insert**	**space (references)**	**moby.txt**	**actors.txt**
red-black BST	$L + \log^2 n$	$\log^2 n$	$\log^2 n$	$4n$	1.4	97.4
hashing (linear probing)	L	L	L	$4n$ to $16n$	0.76	40.6
R-way trie	L	$\log_R n$	$R + L$	$(R+1)\,n$	1.12	*out of memory*
TST	$L + \log n$	$\log n$	$L + \log n$	$4n$	0.72	38.7

Bottom line. TST is as fast as hashing (for string keys) and space efficient.

5.2 TRIES

- *string symbol tables*
- *R-way tries*
- *ternary search tries*
- *character-based operations*

Algorithms

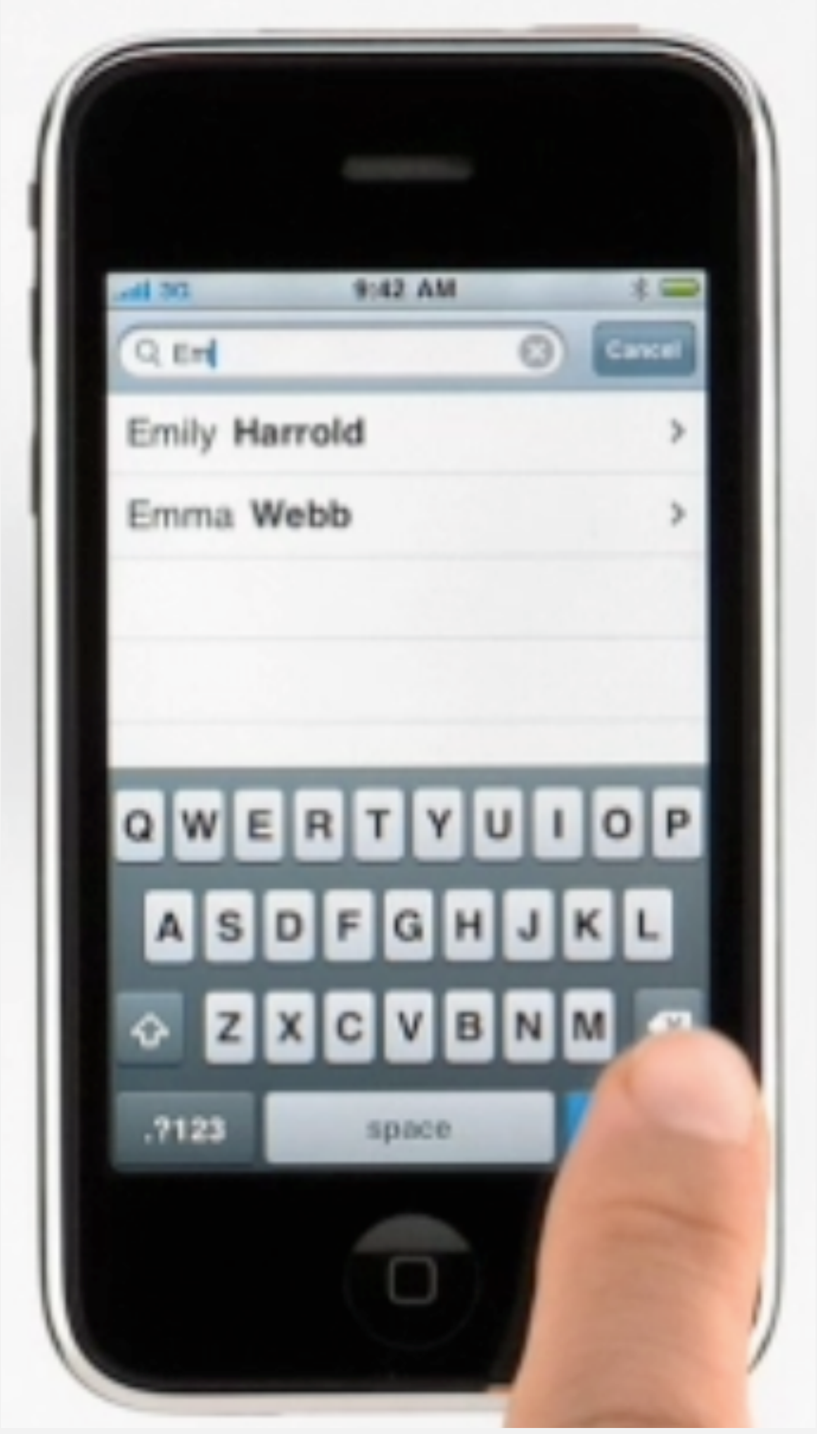
ROBERT SEDGEWICK | KEVIN WAYNE

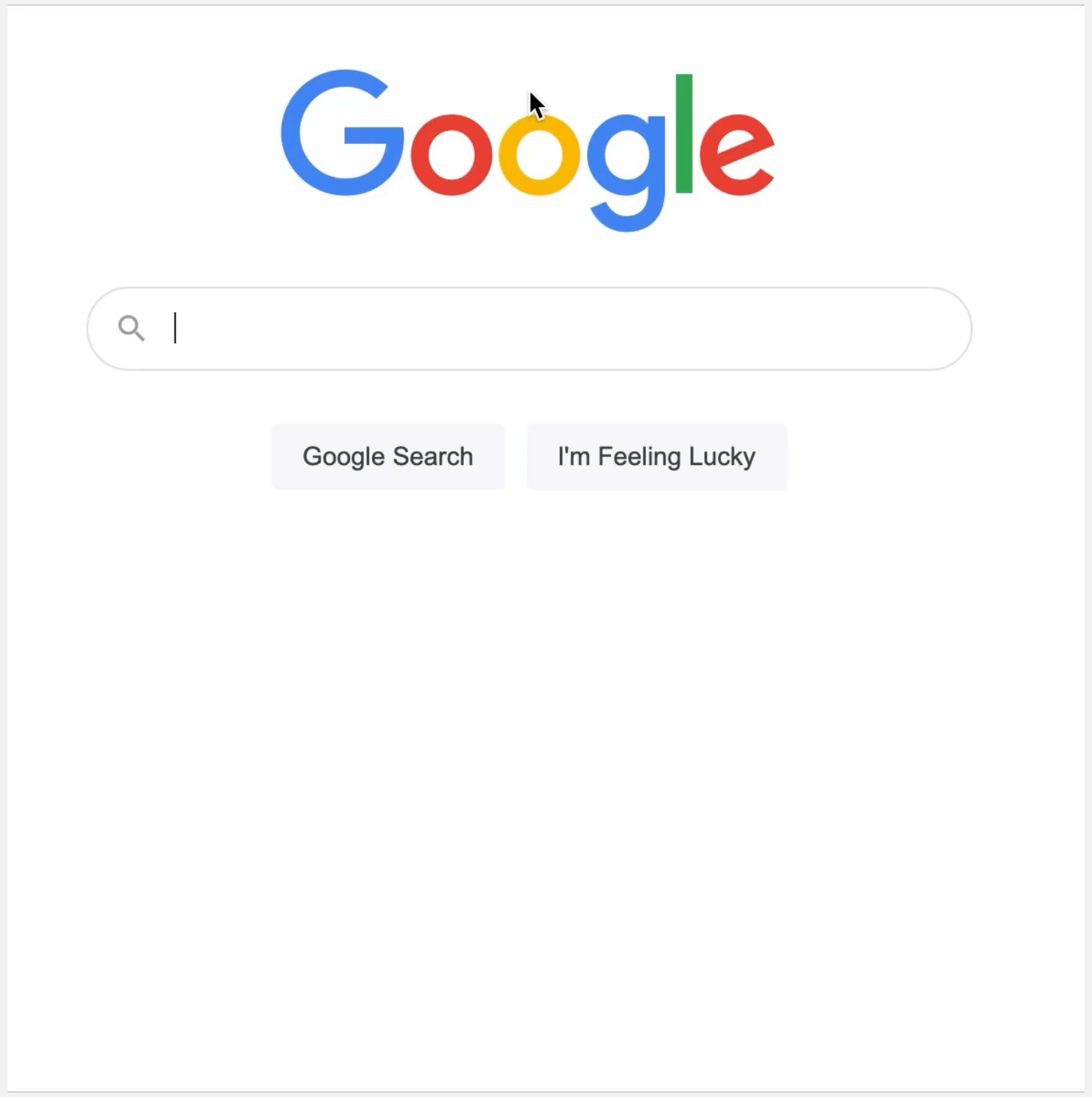
https://algs4.cs.princeton.edu

Autocompletion

Autocompletion.

- User types characters one at a time. ← in a cell phone, search bar, text editor, shell, ...
- System reports all matching strings.





Prefix matches

Prefix matches. Find all keys in symbol table that start with a given prefix.

Ex 1. Prefix = "sh" $\Longrightarrow$ matches = "she", "shells", and "shore".

Ex 2. Prefix = "se" $\Longrightarrow$ matches = "sea" and "sells".

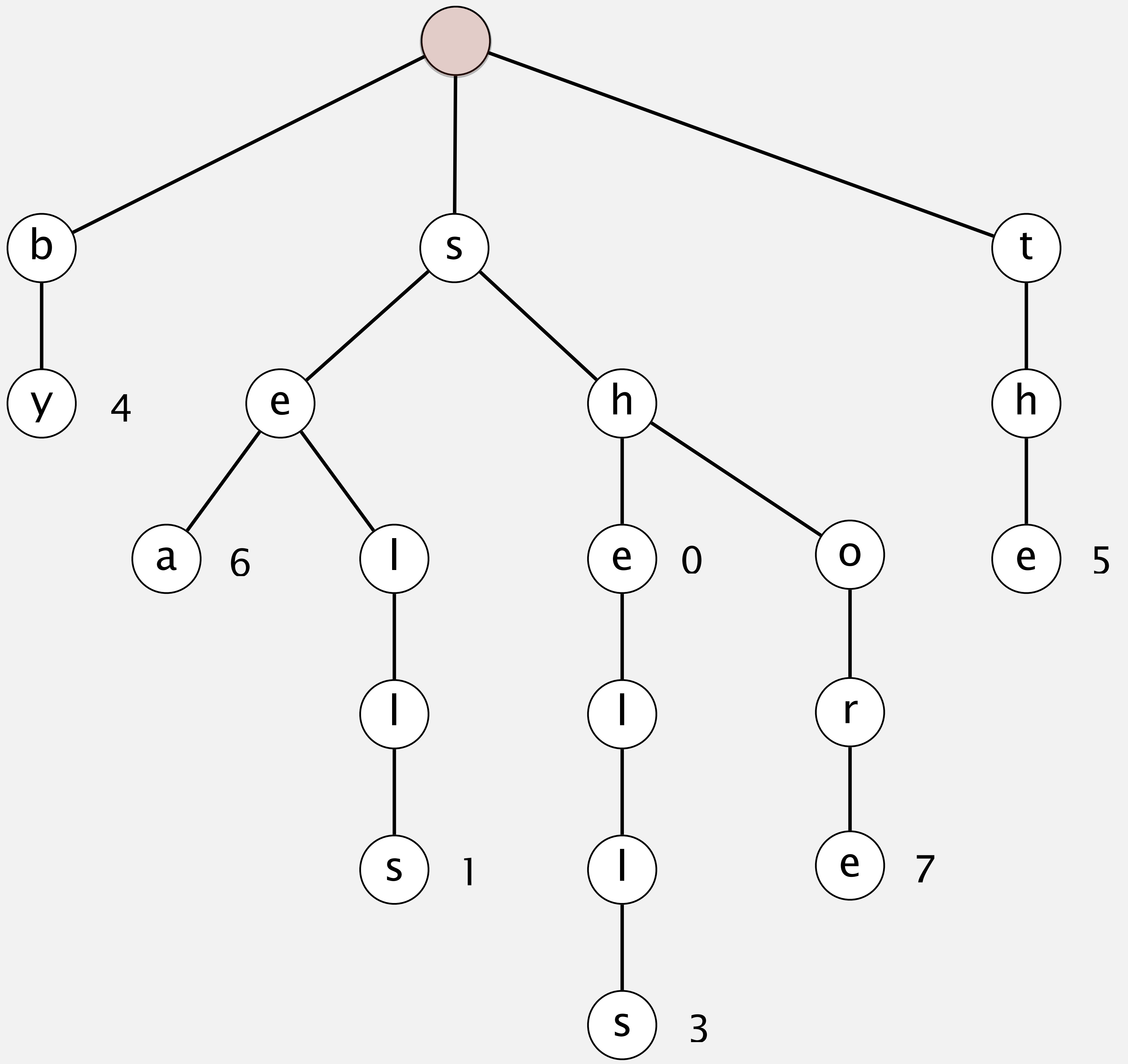
key	value
by	4
sea	6
sells	1
she	0
shells	3
shore	7
the	5

Warmup: ordered iteration

To iterate over all keys in sorted order:

- Do inorder traversal of trie; add keys encountered to a queue.
- Maintain sequence of characters on path from root to node.

prefix	queue
b	
by	by
s	
se	
sea	by sea
sel	
sell	
sells	by sea sells
sh	
she	by sea sells she
shel	
shell	
shells	by sea sells she shells
sho	
shor	
shore	by sea sells she shells shore
t	
th	
the	by sea sells she shells shore the



Ordered iteration: Java implementation

To iterate over all keys in sorted order:

- Do inorder traversal of trie; add keys encountered to a queue.
- Maintain sequence of characters on path from root to node.

```
public Iterable<String> keys()
{
   Queue<String> queue = new Queue<String>();
   collect(root, "", queue);
   return queue;
}

private void collect(Node x, String prefix, Queue<String> queue)
{
   if (x == null) return;
   if (x.val != null) queue.enqueue(prefix);
   for (char c = 0; c < R; c++)
      collect(x.next[c], prefix + c, queue);
}
```

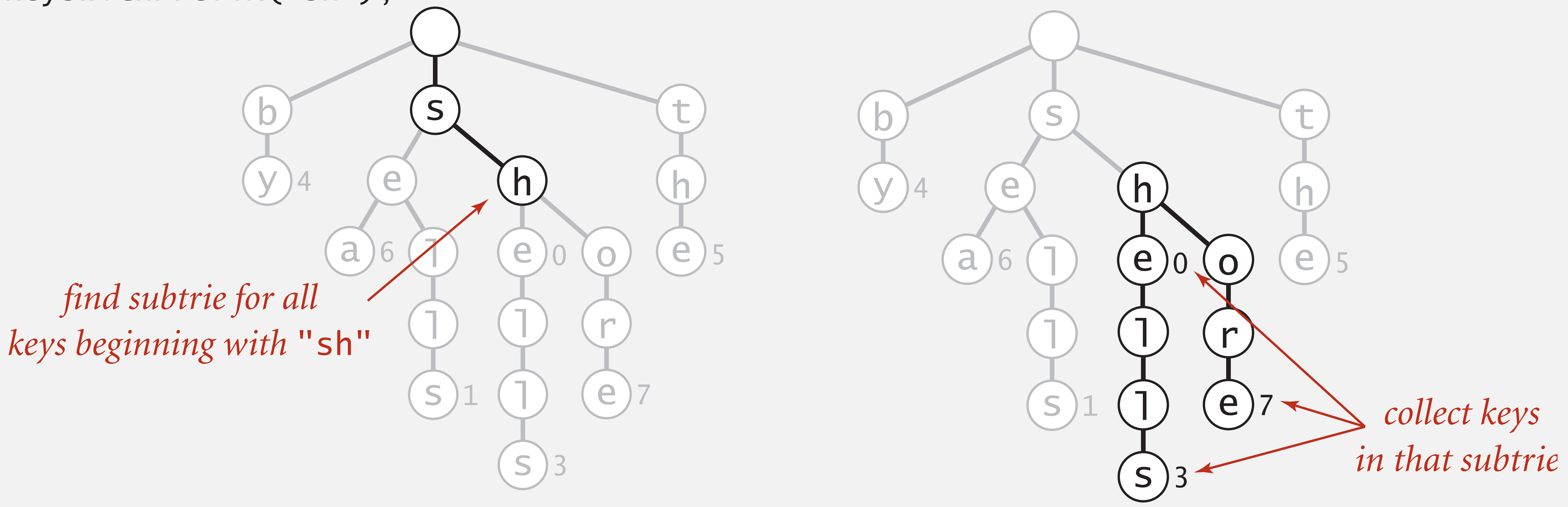
sequence of characters on path from root to x (annotation pointing to `prefix` in the `collect` signature)

or use StringBuilder (annotation pointing to `prefix + c`)

Prefix matches in an R-way trie

Prefix matches. Find all keys in symbol table that start with a given prefix.

```
keysWithPrefix("sh");
```



T9 texting (predictive texting)

Goal. Type text messages on a phone keypad.

Multi-tap input. Enter a letter by repeatedly pressing a key.

Ex. good: 4 6 6 6 6 6 6 3

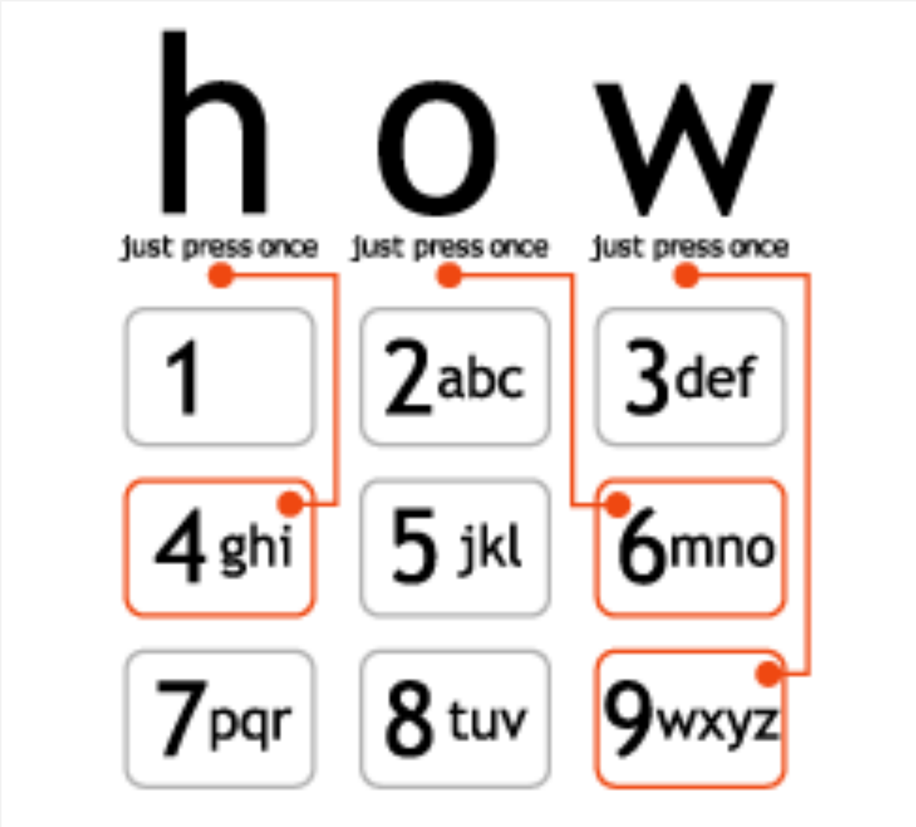
"a much faster and more fun way to enter text"

T9 text input (on 4 billion handsets).

- Find all words that correspond to given sequence of numbers.

  4663: good, home, gone, hoof. ← textonyms

- Press * to select next option.
- Press 0 to see all completion options.
- System adapts to user's tendencies.



http://www.t9.com

T9 TEXTING

Q. How to implement T9 texting on a mobile phone?

SONY verizon SIEMENS

NEC SANYO SAMSUNG ELECTRONICS

Network router IP address lookup

IP address lookup. To send packet toward destination IP address x, network router finds longest IP address in its routing table that is a prefix of x.

backbone router might have 1M entries and process millions of queries per second

routing table

destination (key)	*gateway (value)*
128	
128.112	
128.112.055	
128.112.055.15	
128.112.136	
128.112.155.11	
128.112.155.13	
128.222	
128.222.136	

represented as 32-bit binary number for IPv4 (instead of string)

```
longestPrefixOf(128.112.100.16) = 128.112
longestPrefixOf(128.166.123.45) = 128
longestPrefixOf(128.112.136.11) = 128.112.136
```

Note. Not the same as floor: `floor(128.112.100.16) = 128.112.055.15`

Longest prefix match

Longest prefix match. Find longest key in symbol table that is a prefix of query string.

Ex 1. Query = "shellsort" $\Longrightarrow$ match = "shells".

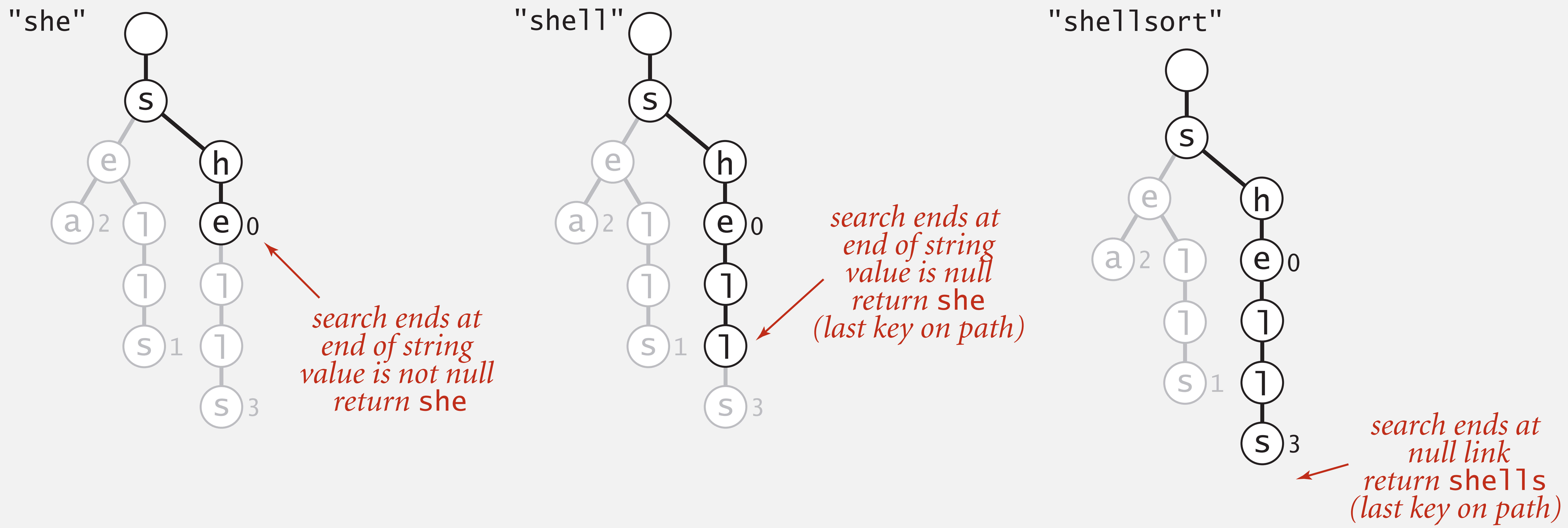
Ex 2. Query = "sheep" $\Longrightarrow$ match = "she".

key	value
by	4
sea	6
sells	1
she	0
shells	3
shore	7
the	5

Longest prefix match in an R-way trie

Longest prefix match. Find longest key in symbol table that is a prefix of query string.

- Search for query string.
- Keep track of longest key encountered.

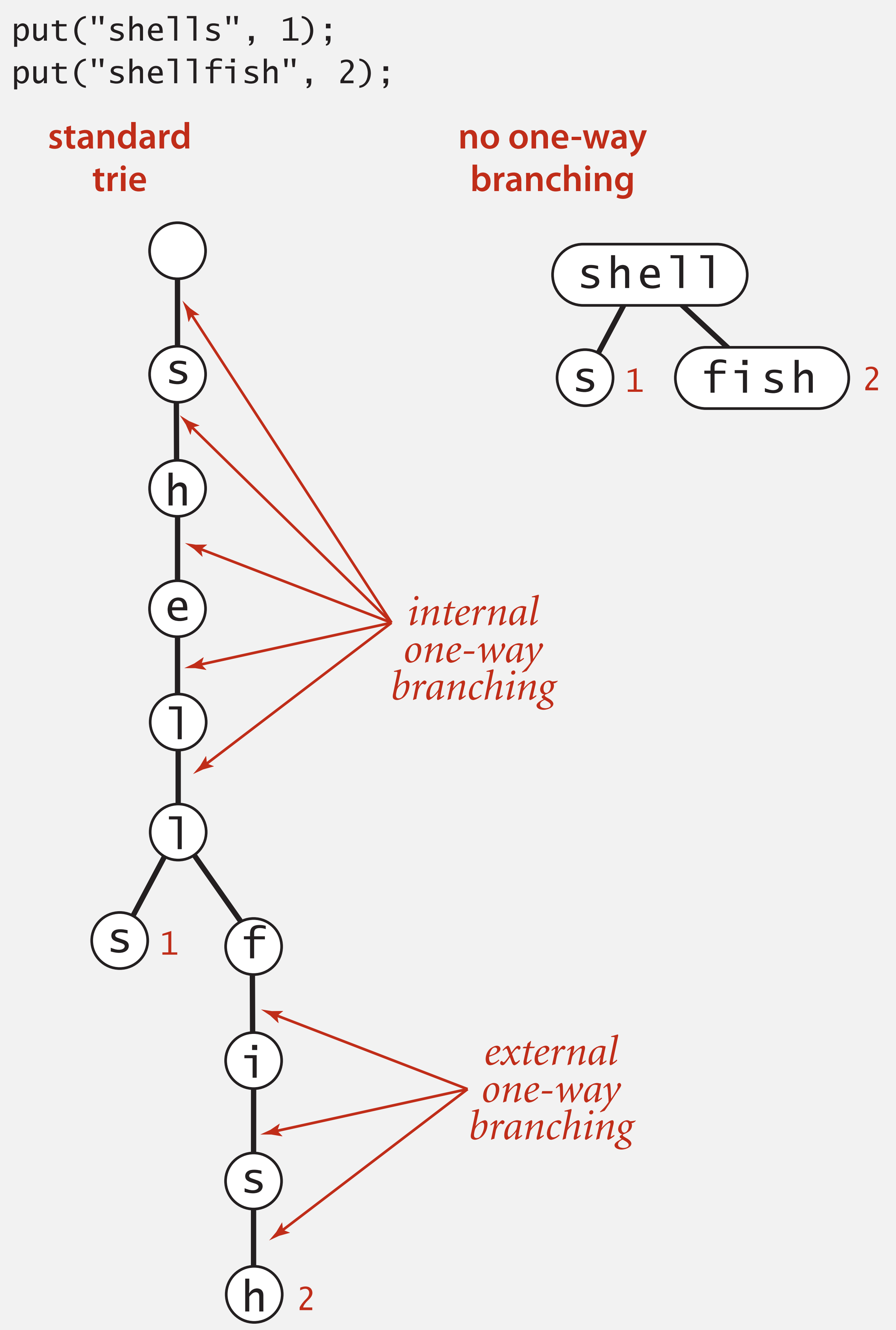


Possibilities for `longestPrefixOf()`

Patricia tries

Patricia trie. [Practical Algorithm to Retrieve Information Coded in Alphanumeric]

- Remove one-way branching.
- Each node represents a sequence of characters.
- Implementation: one step beyond this course.



Applications.

- Database search.
- P2P network search.
- IP routing tables: find longest prefix match.
- Compressed quad-tree for n-body simulation.
- Efficiently storing and querying XML documents.

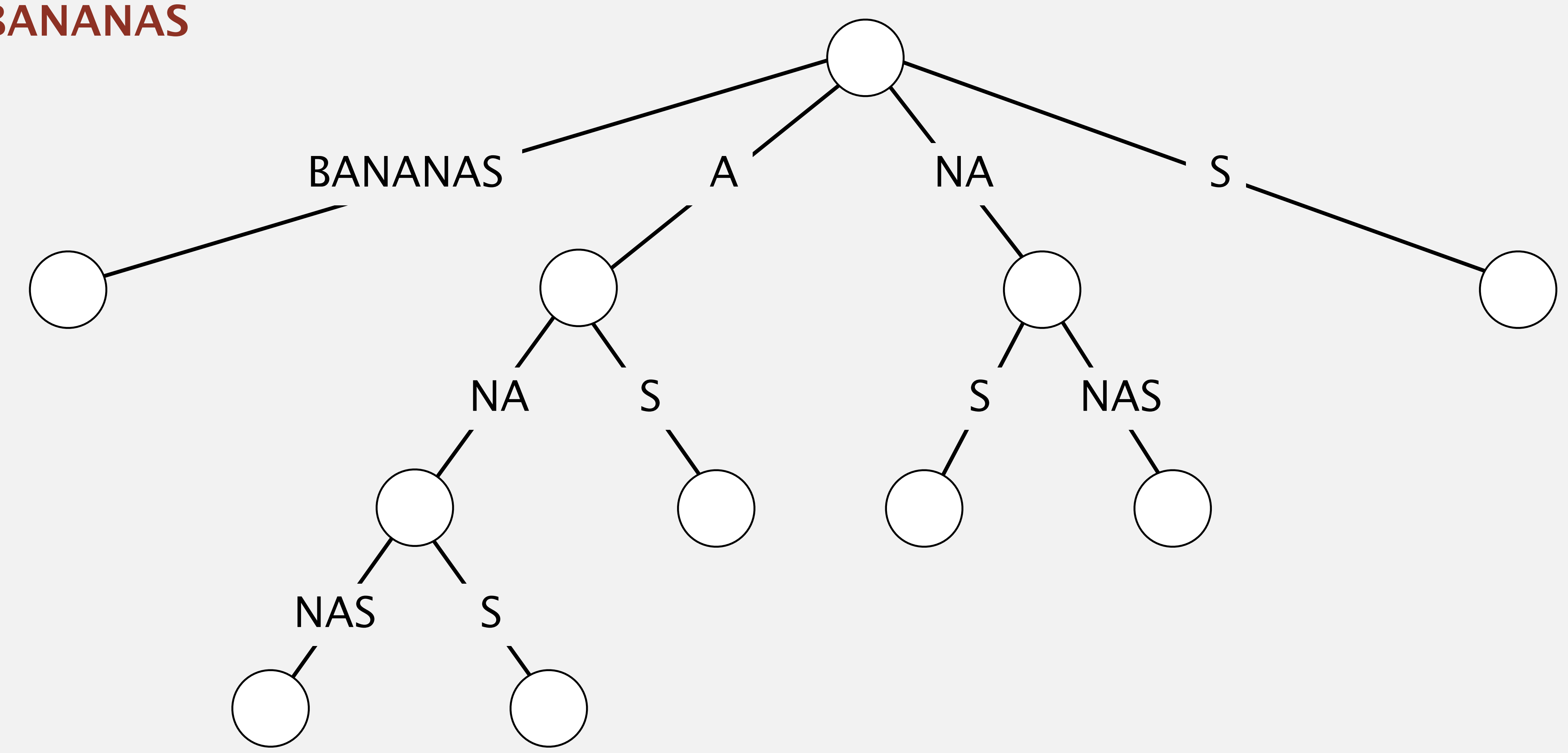
Also known as: crit-bit tree, radix tree.

Suffix trees

Suffix tree.

- Patricia trie of suffixes of a string.
- Linear-time construction: well beyond scope of this course.

suffix tree for BANANAS



Applications.

- Linear-time: longest repeated substring, longest common substring, longest palindromic substring, substring search, tandem repeats, ...
- Computational biology databases (BLAST, FASTA).

String symbol tables summary

A success story in algorithm design and analysis.

Balanced BSTs. [red–black BSTs]

- $\Theta(\log n)$ key compares per search/insert. ← worst case
- Supports ordered operations (e.g., rank, select, floor).

Hash tables. [separate chaining, linear probing]

- $\Theta(1)$ probes per search/insert. ← uniform hashing assumption

Tries. [R-way tries, ternary search tries]

- $\Theta(L + \log n)$ character accesses per search hit/insert.
- $\Theta(\log n)$ character accesses per search miss.

← typical applications

- Supports character-based operations (e.g., prefix match).
- Works only for string (or digital) keys.