BBM 201 – DATA STRUCTURES

DEPT. OF COMPUTER ENGINEERING

TRIES

Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.

Today

- **Tries**
- **R-way tries**

Tries

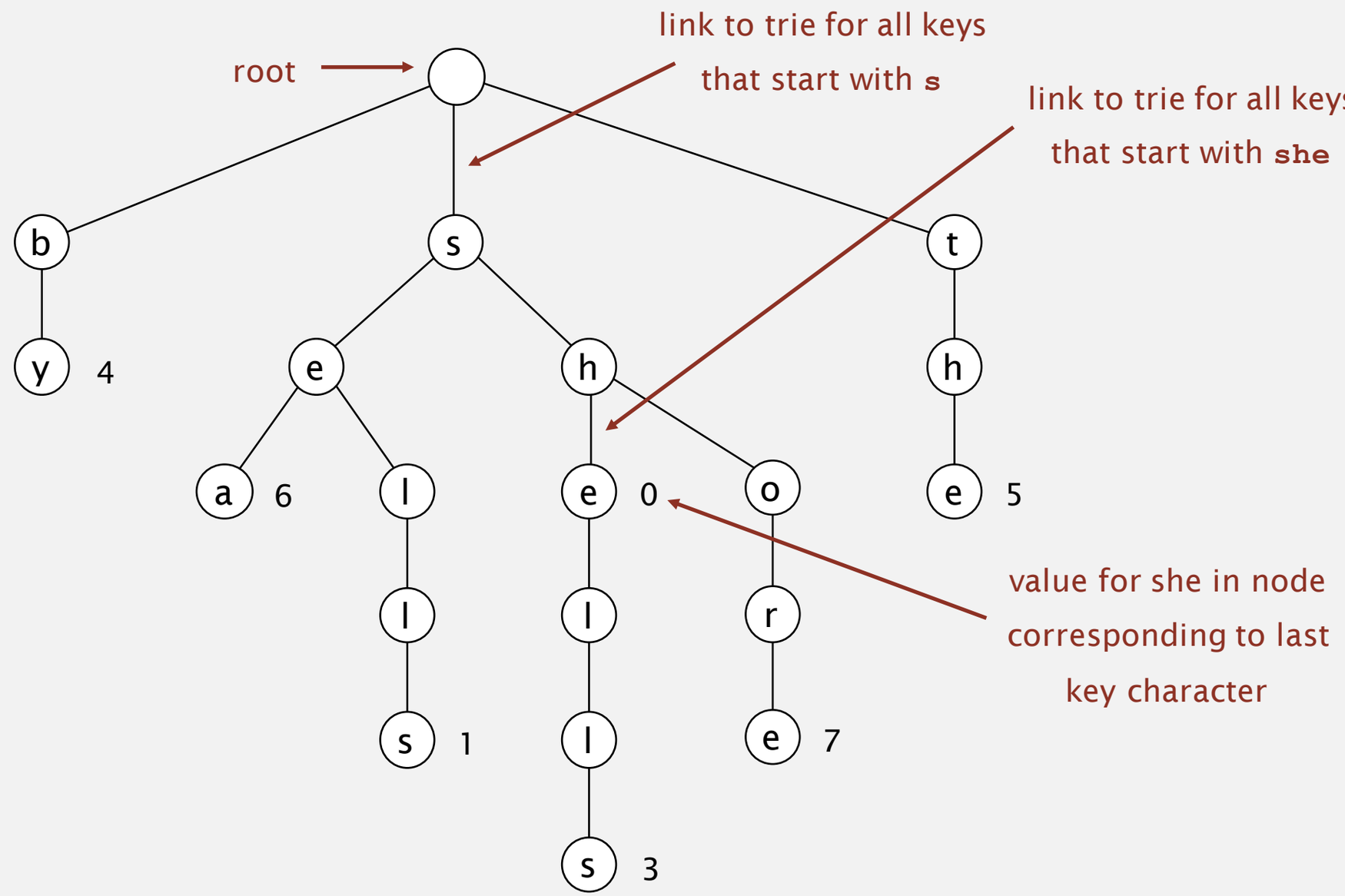
- **R-way tries**

Tries

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

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

for now, we do not draw null links

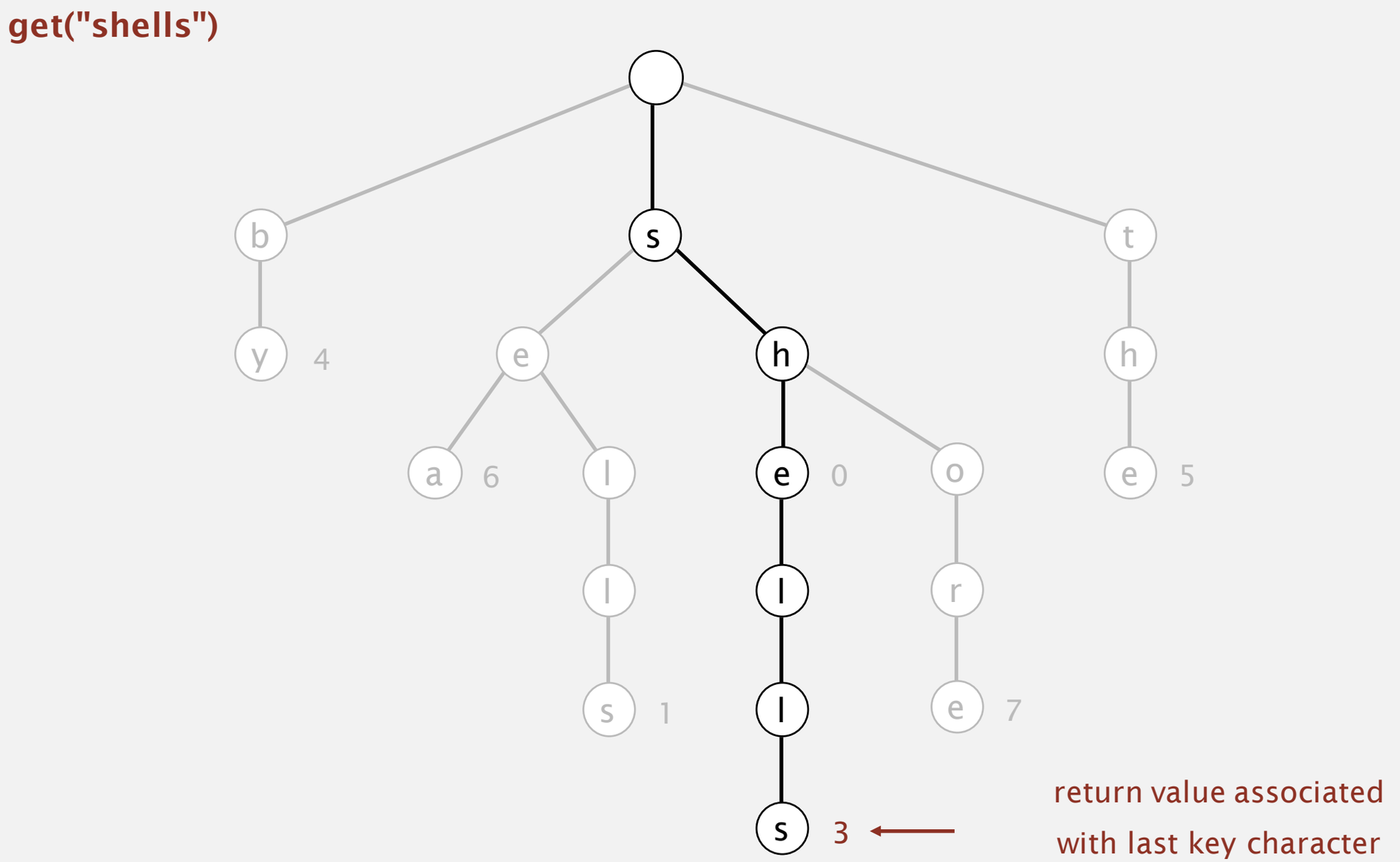


key	value
by	4
sea	6
sells	1
she	0
shells	3
shore	7
the	5

Search in a trie

Follow links corresponding to each character in the key.

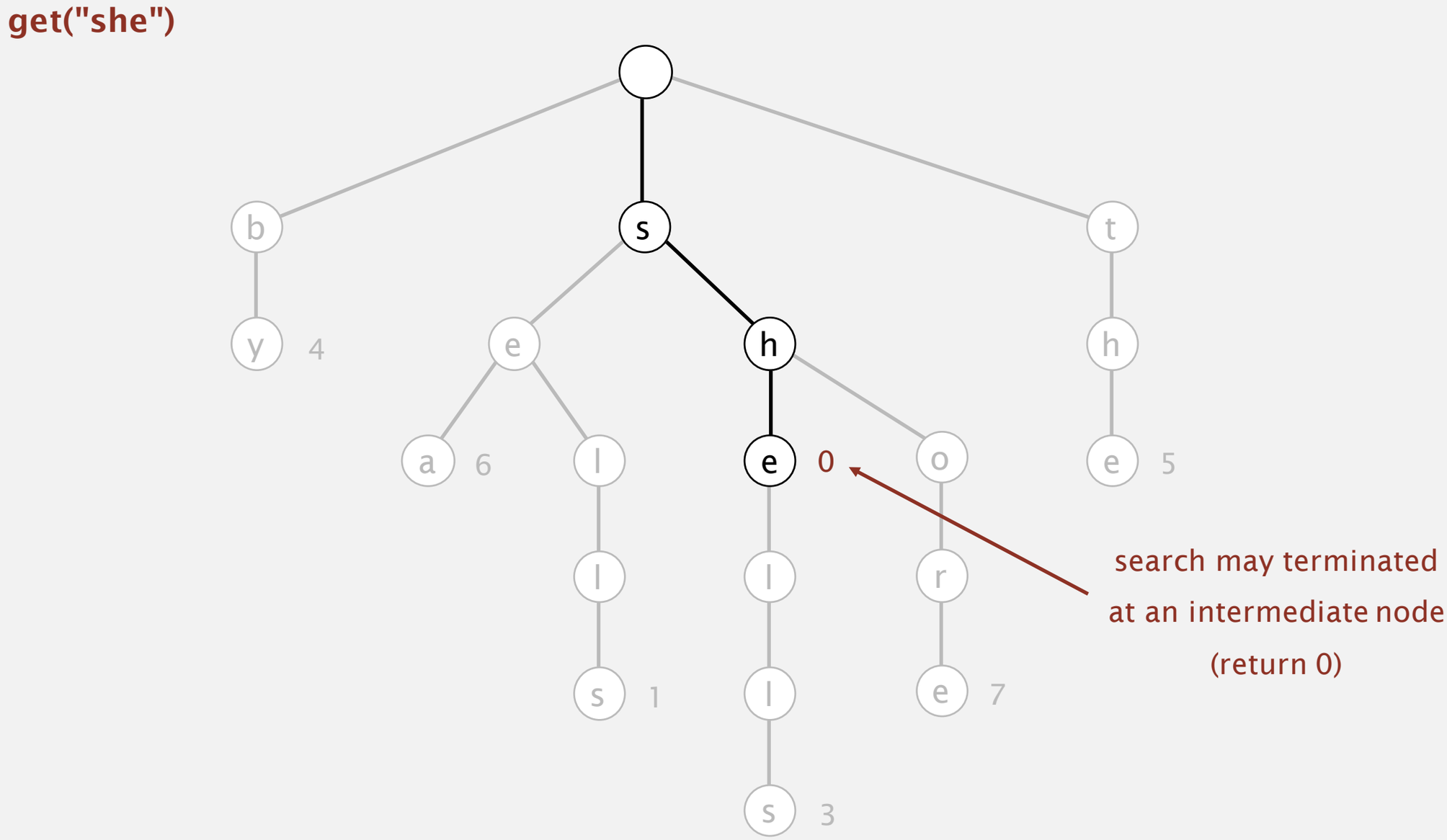
- Search hit: node where search ends has a non-null value.
- Search miss: reach a null link or node where search ends has null value.



Search in a trie

Follow links corresponding to each character in the key.

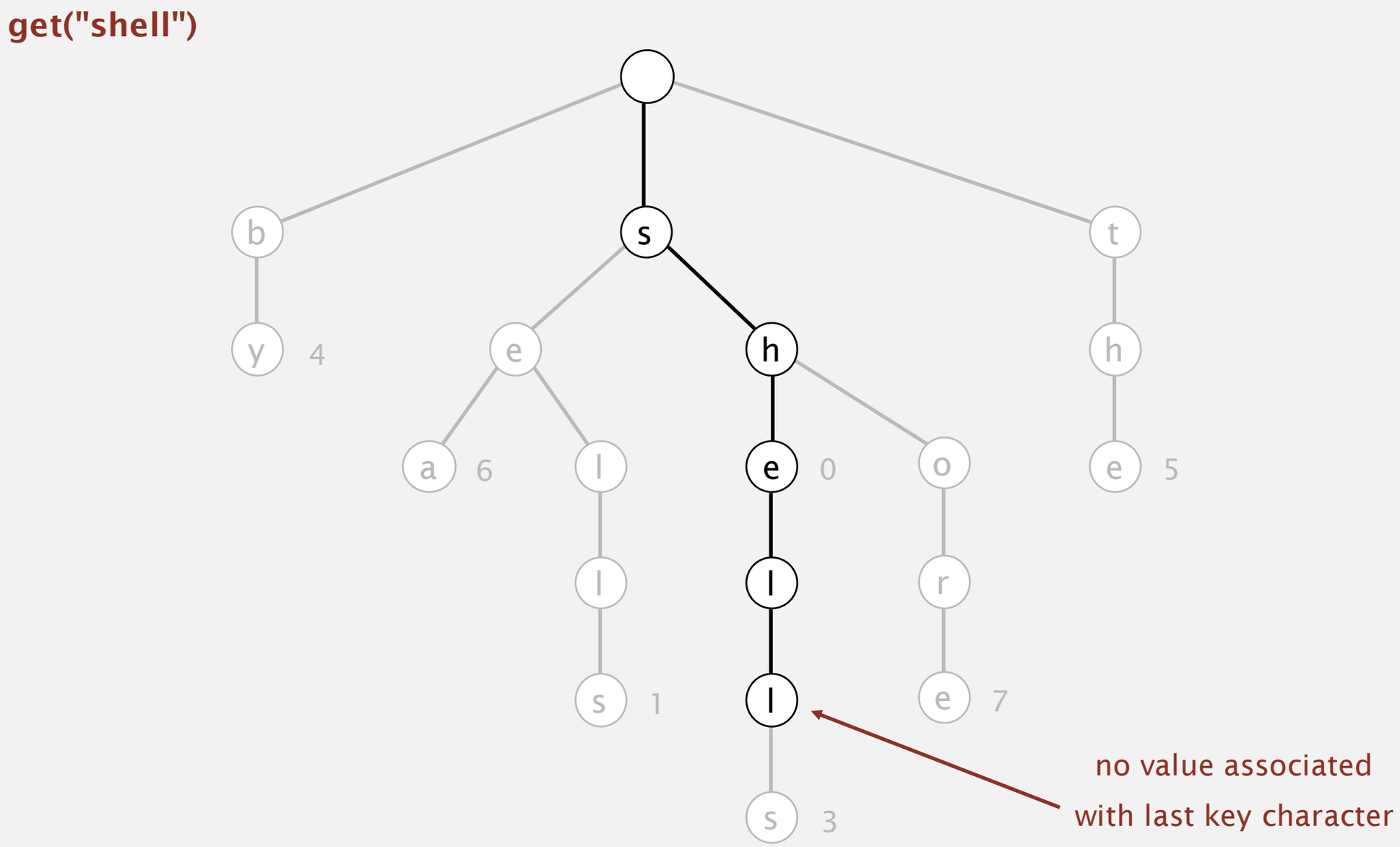
- Search hit: node where search ends has a non-null value.
- Search miss: reach a null link or node where search ends has null value.



Search in a trie

Follow links corresponding to each character in the key.

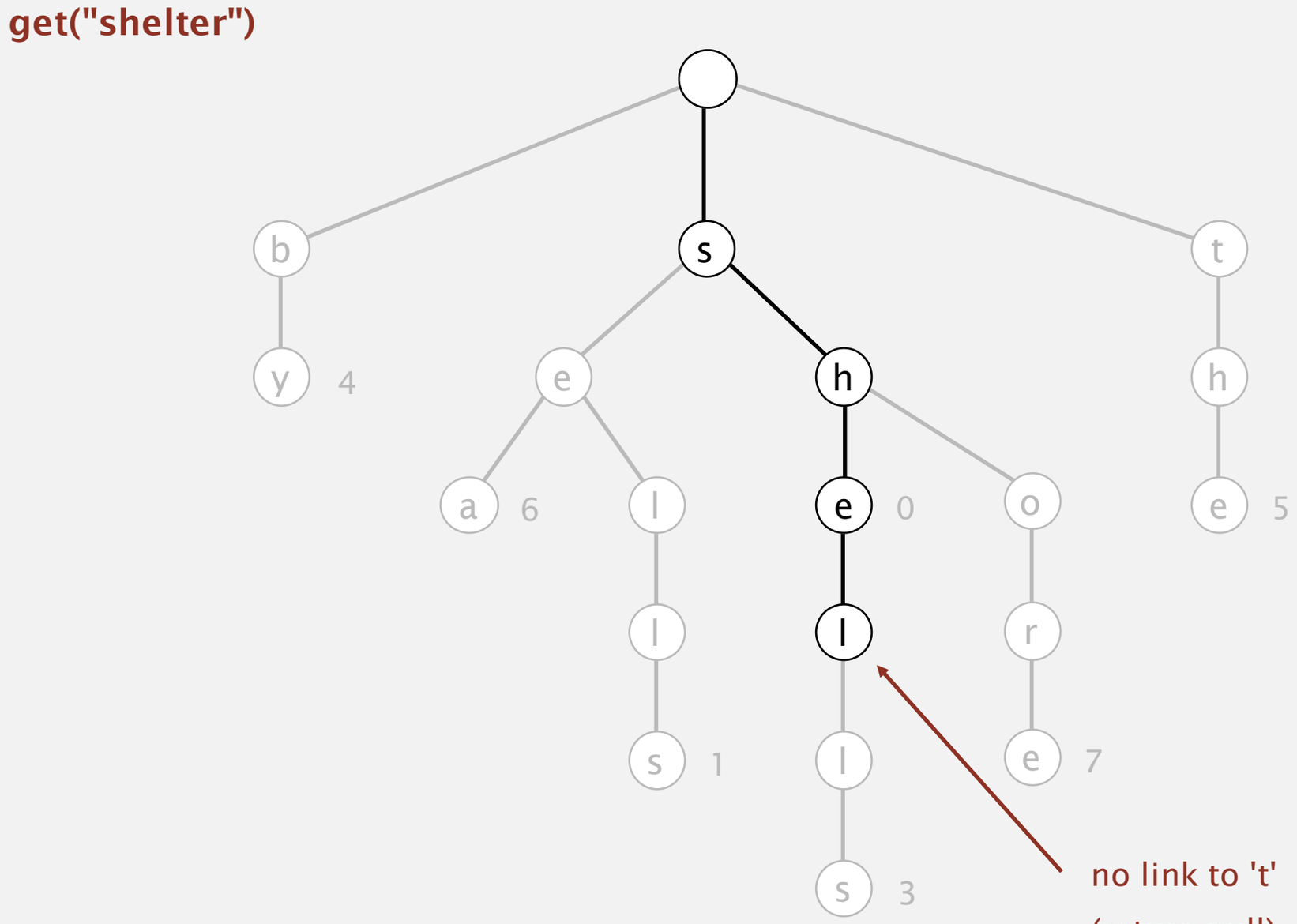
- Search hit: node where search ends has a non-null value.
- Search miss: reach a null link or node where search ends has null value.



Search in a trie

Follow links corresponding to each character in the key.

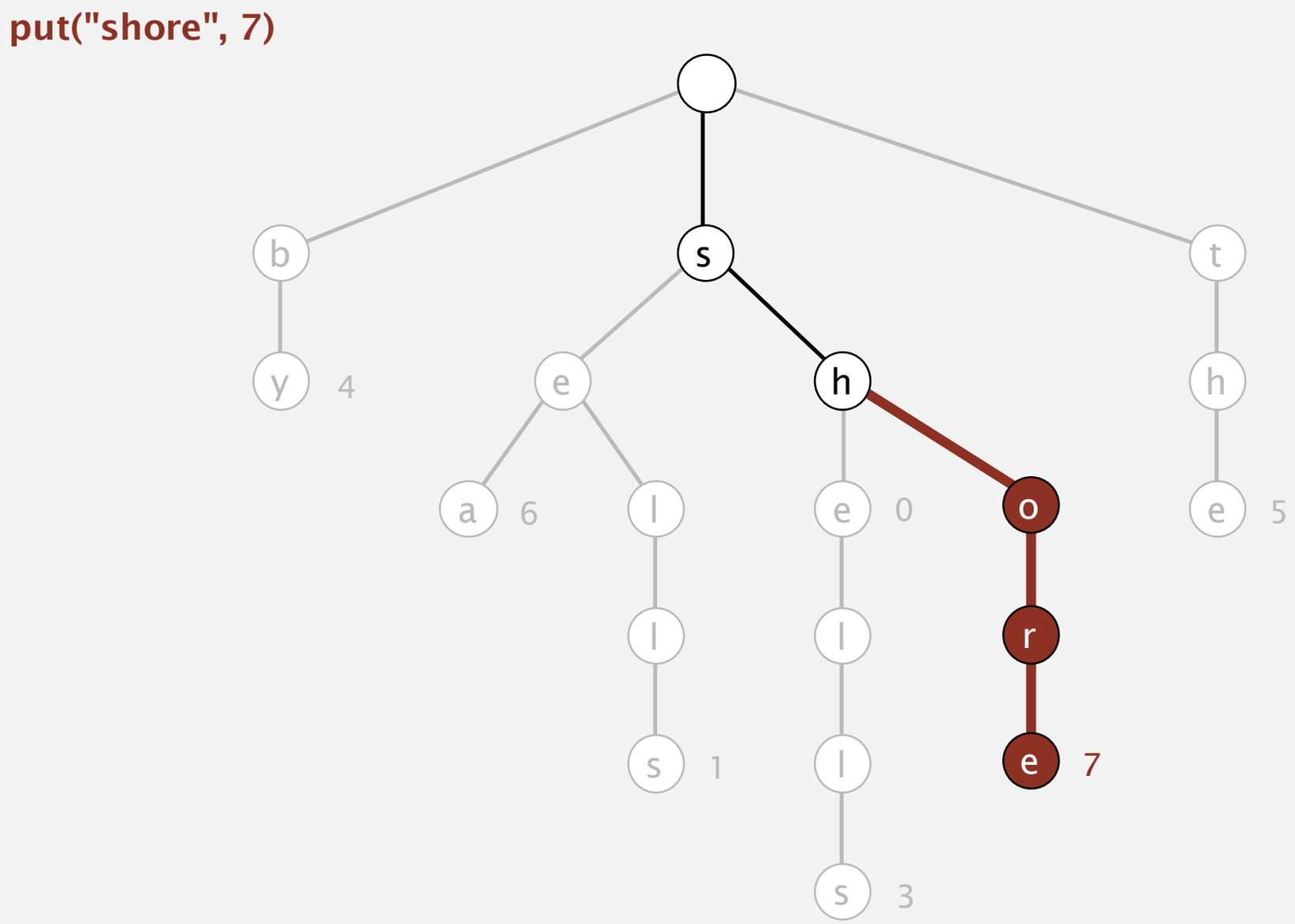
- Search hit: node where search ends has a non-null value.
- Search miss: reach a null link or node where search ends has null value.



Insertion into a trie

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

Trie construction demo

put("she", 0)

key is sequence of characters from root to value

value is in node corresponding to last character

Trie construction demo

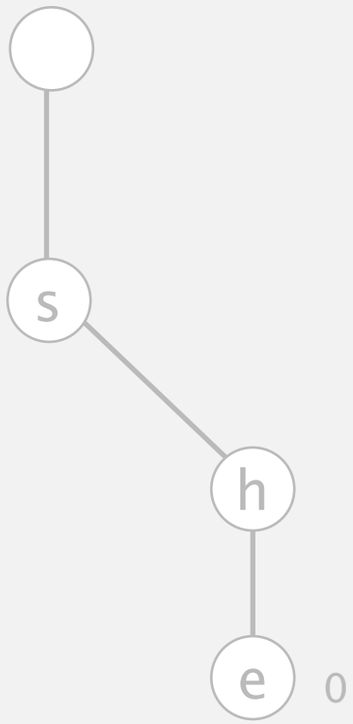
she

trie

Trie construction demo

she

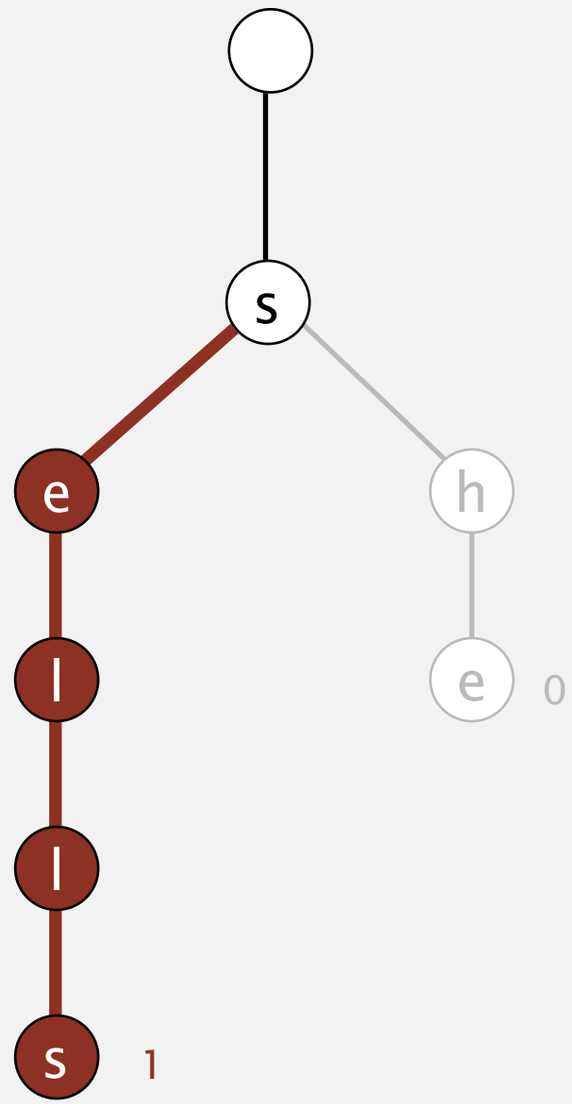
trie



Trie construction demo

she

put("sells", 1)

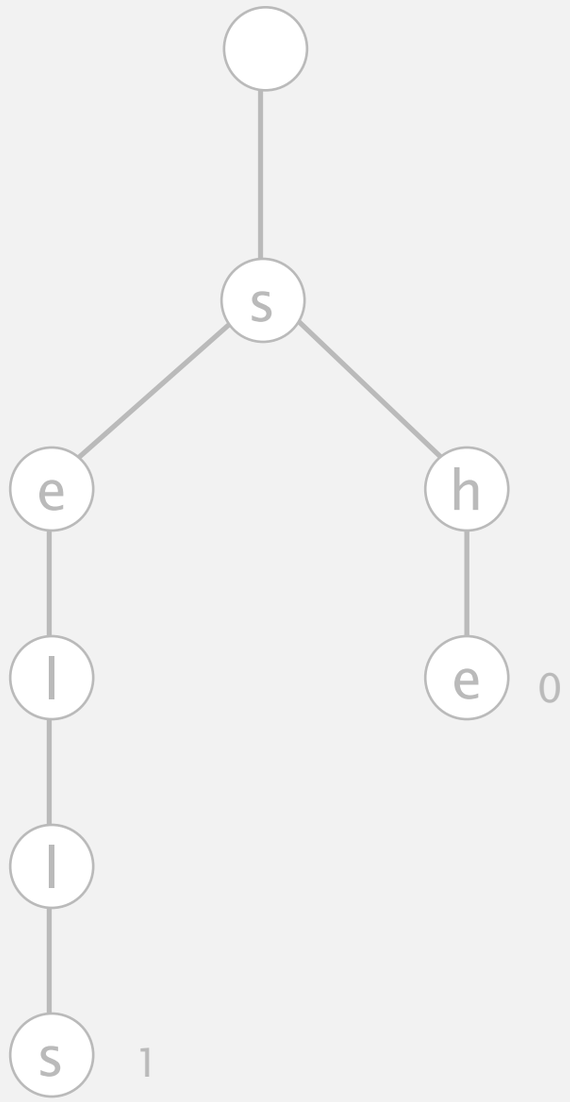


Trie construction demo

she

sells

trie

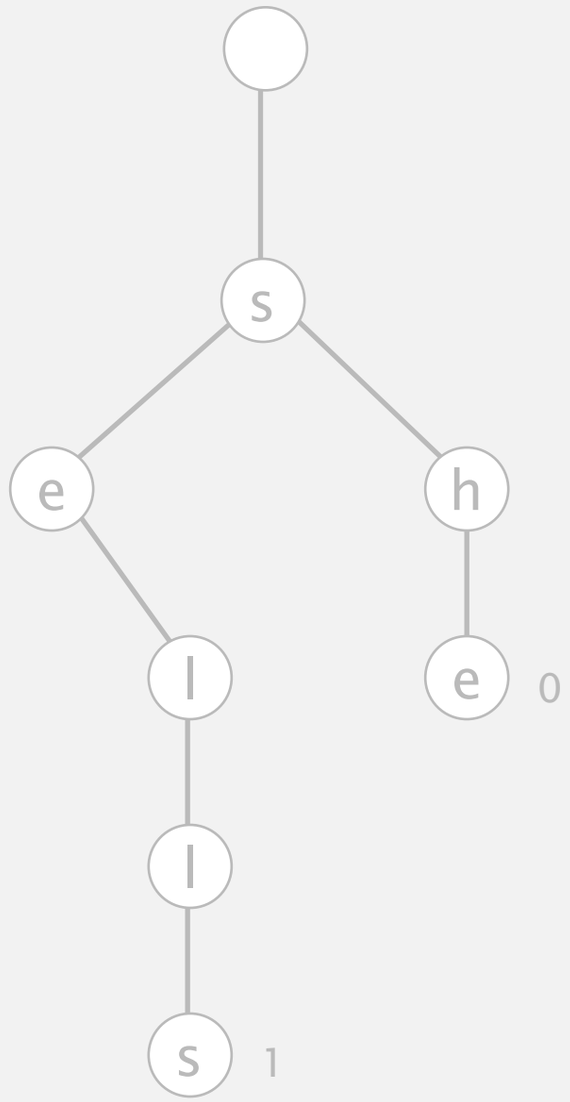


Trie construction demo

she

sells

trie

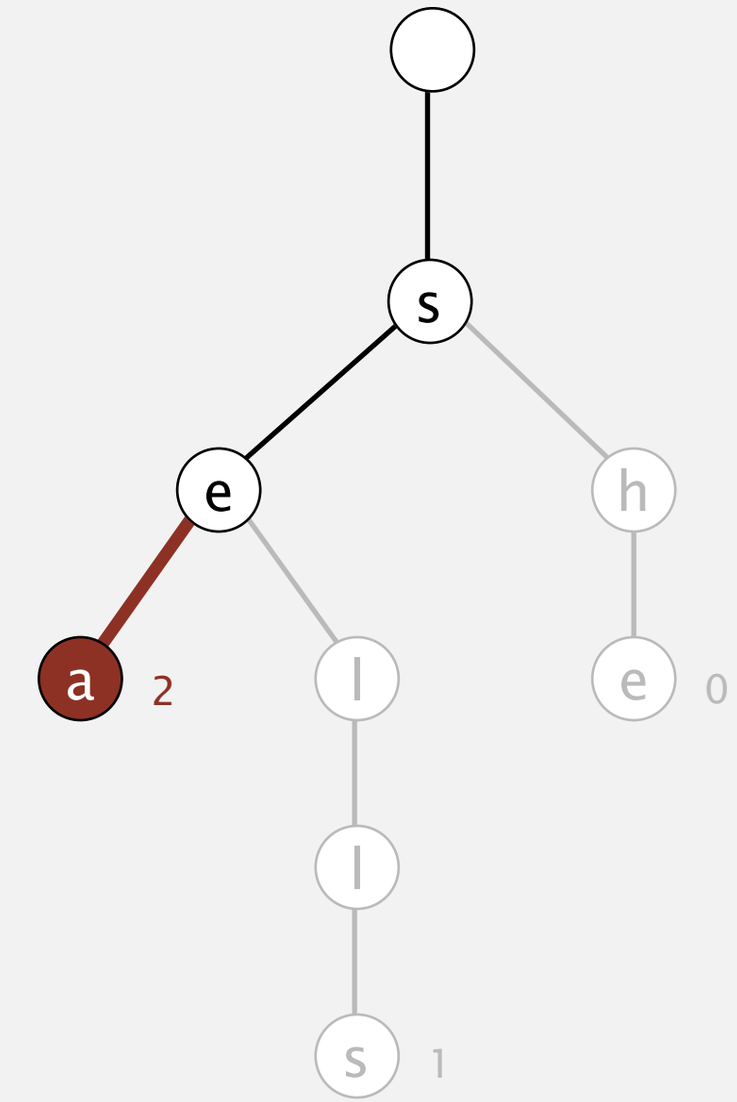


Trie construction demo

she

sells

put("sea", 2)



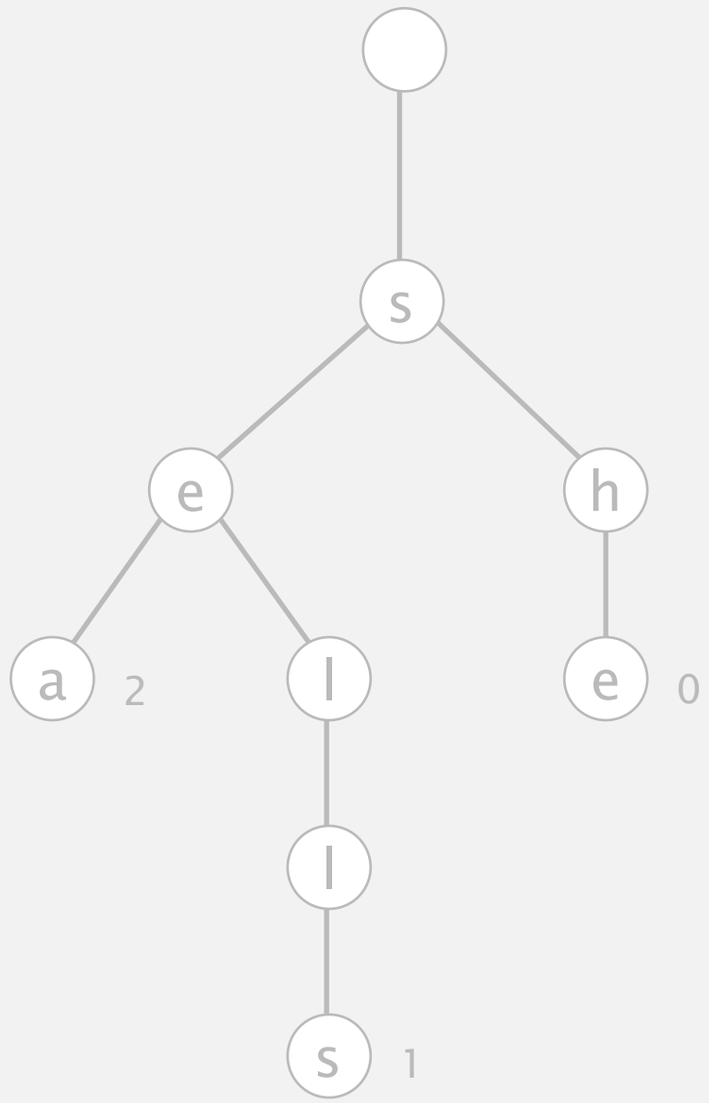
Trie construction demo

she

sells

sea

trie



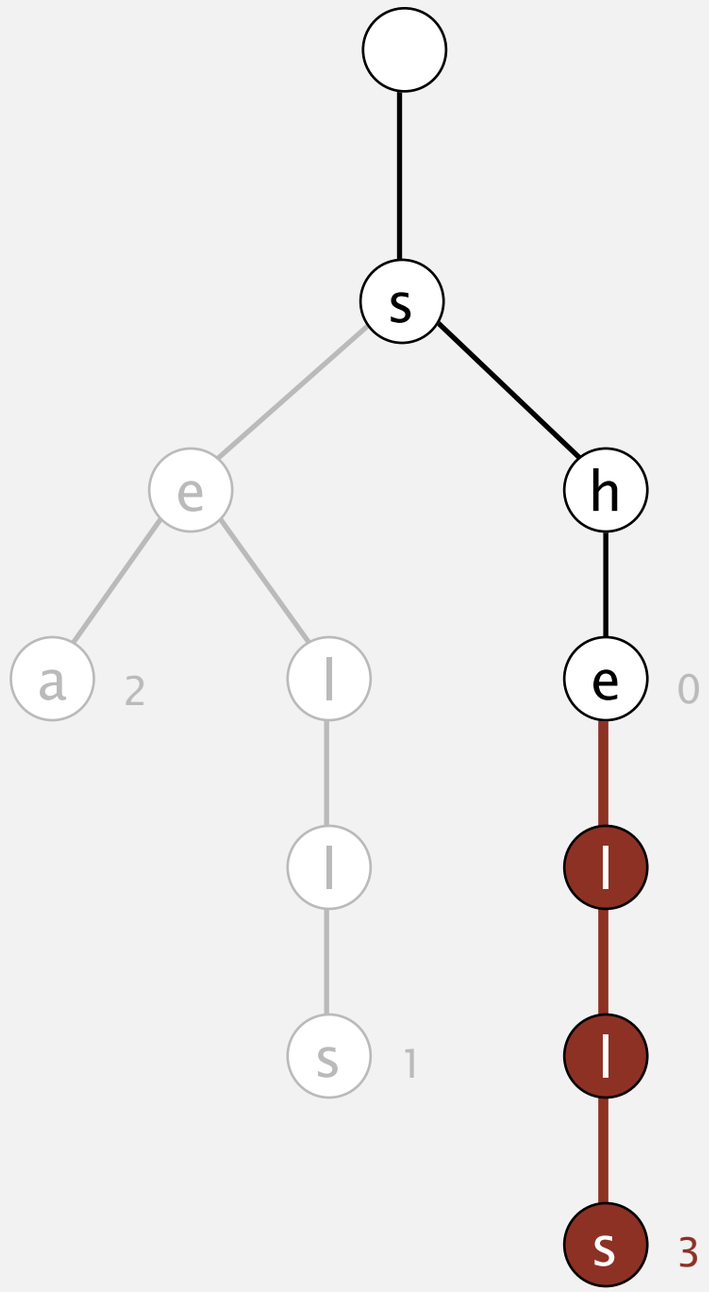
Trie construction demo

she

sells

sea

put("shells", 3)



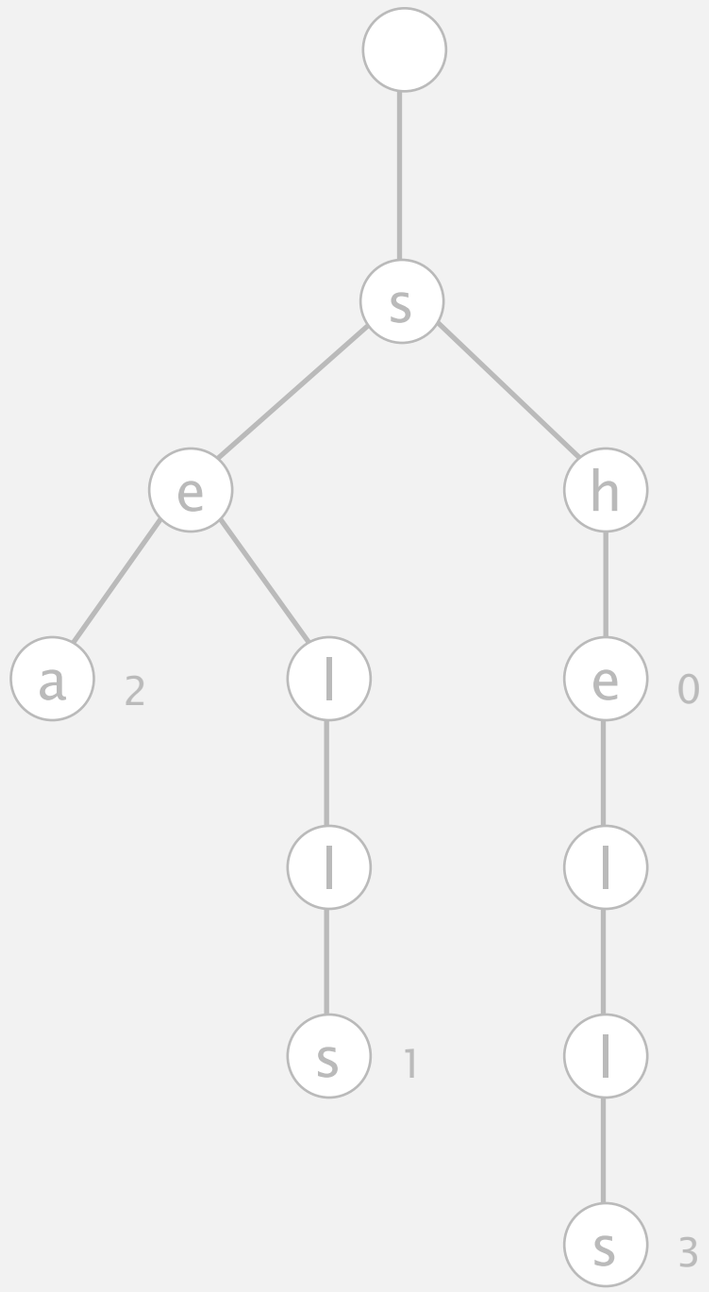
Trie construction demo

she

sells

sea

trie



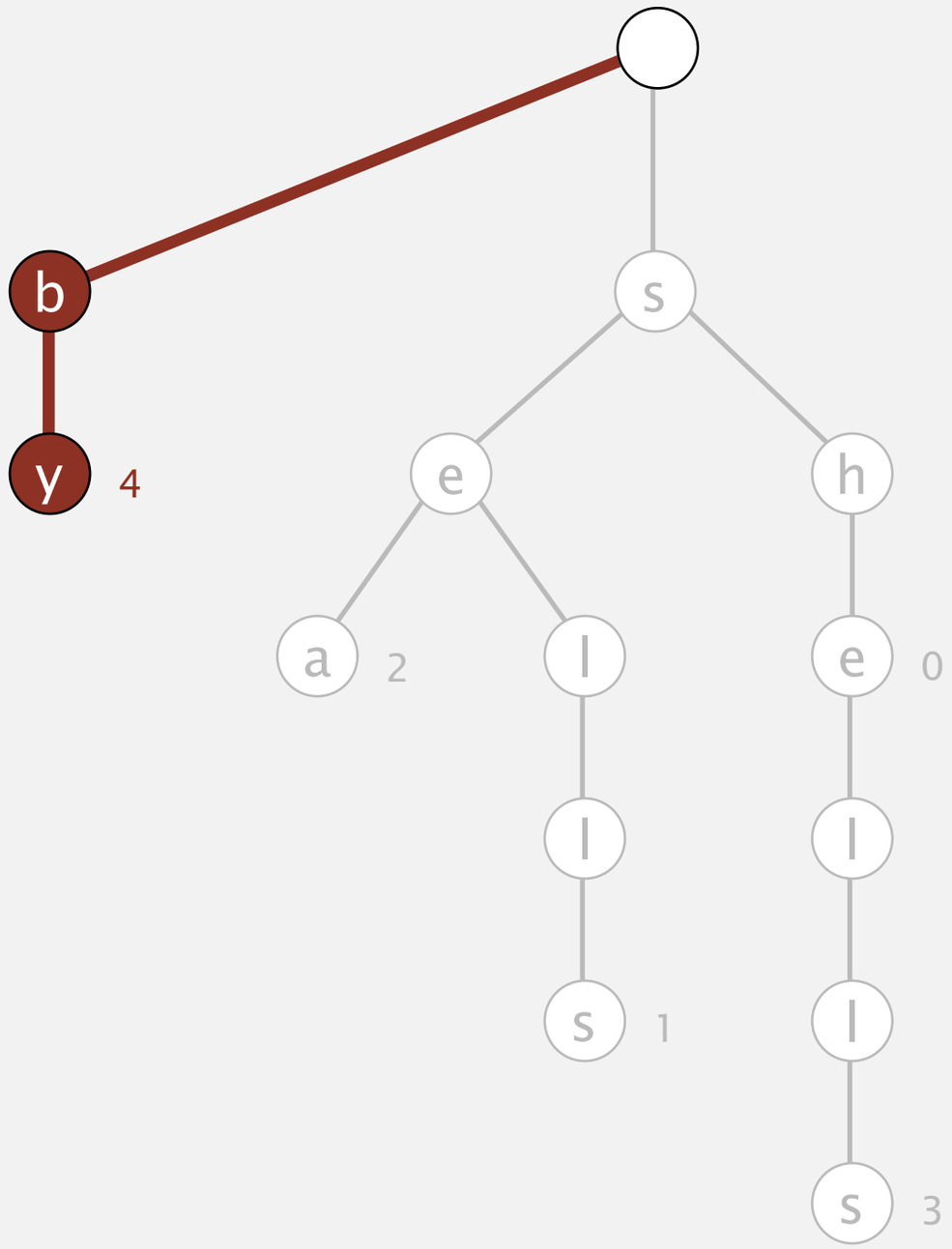
Trie construction demo

she

sells

sea

put("by", 4)



Trie construction demo

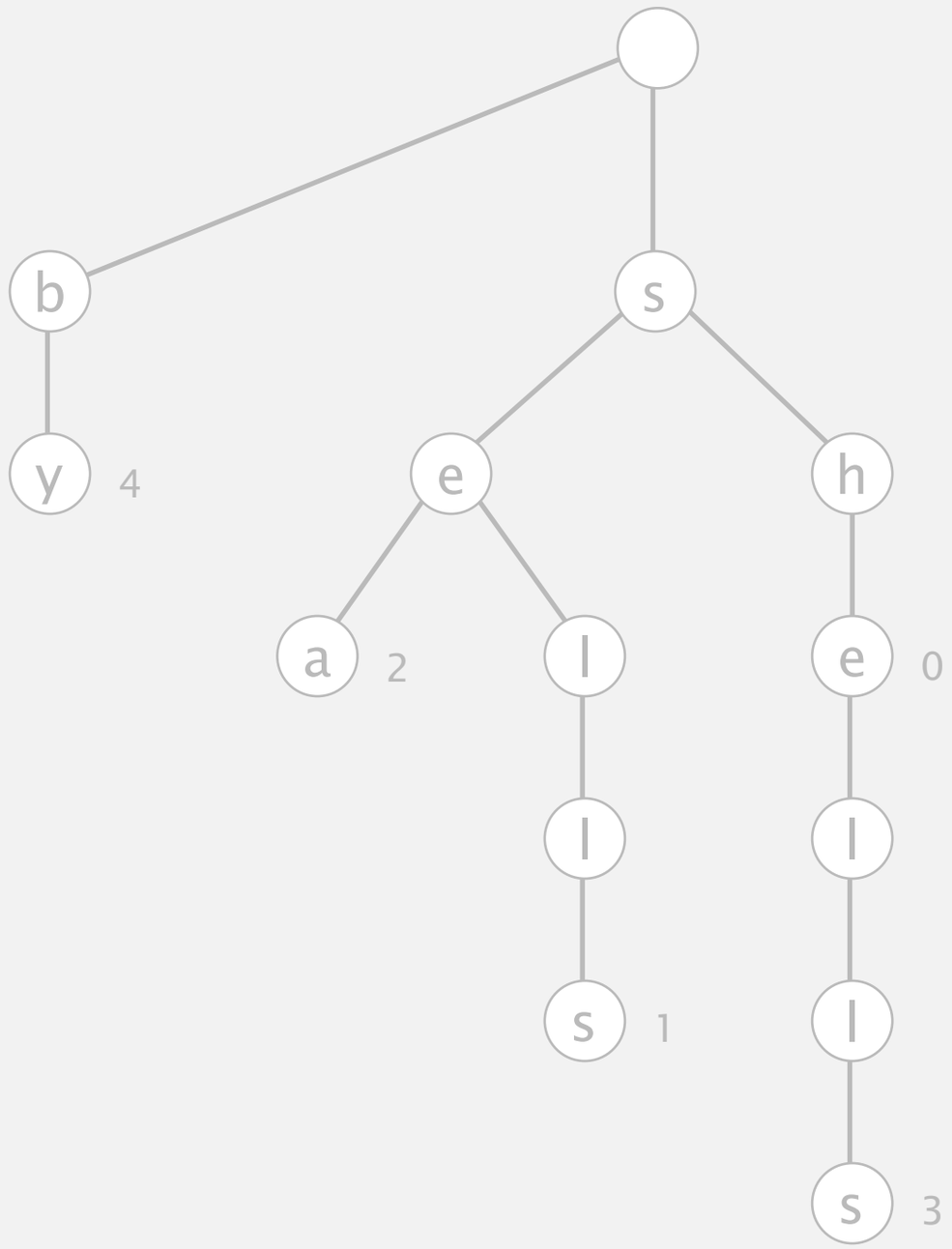
she

sells

sea

by

trie



Trie construction demo

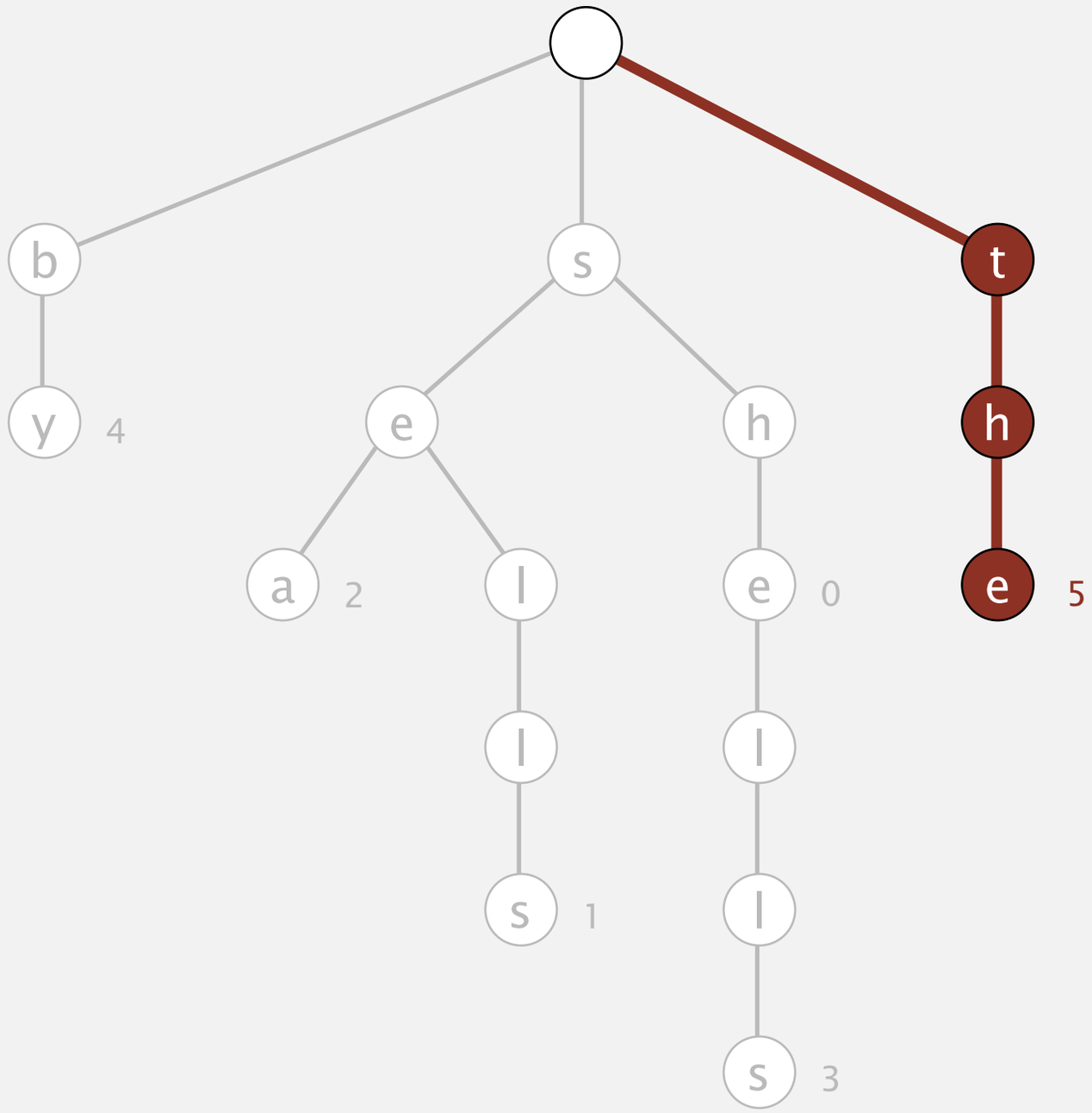
she

sells

sea

by

put("the", 5)



Trie construction demo

she

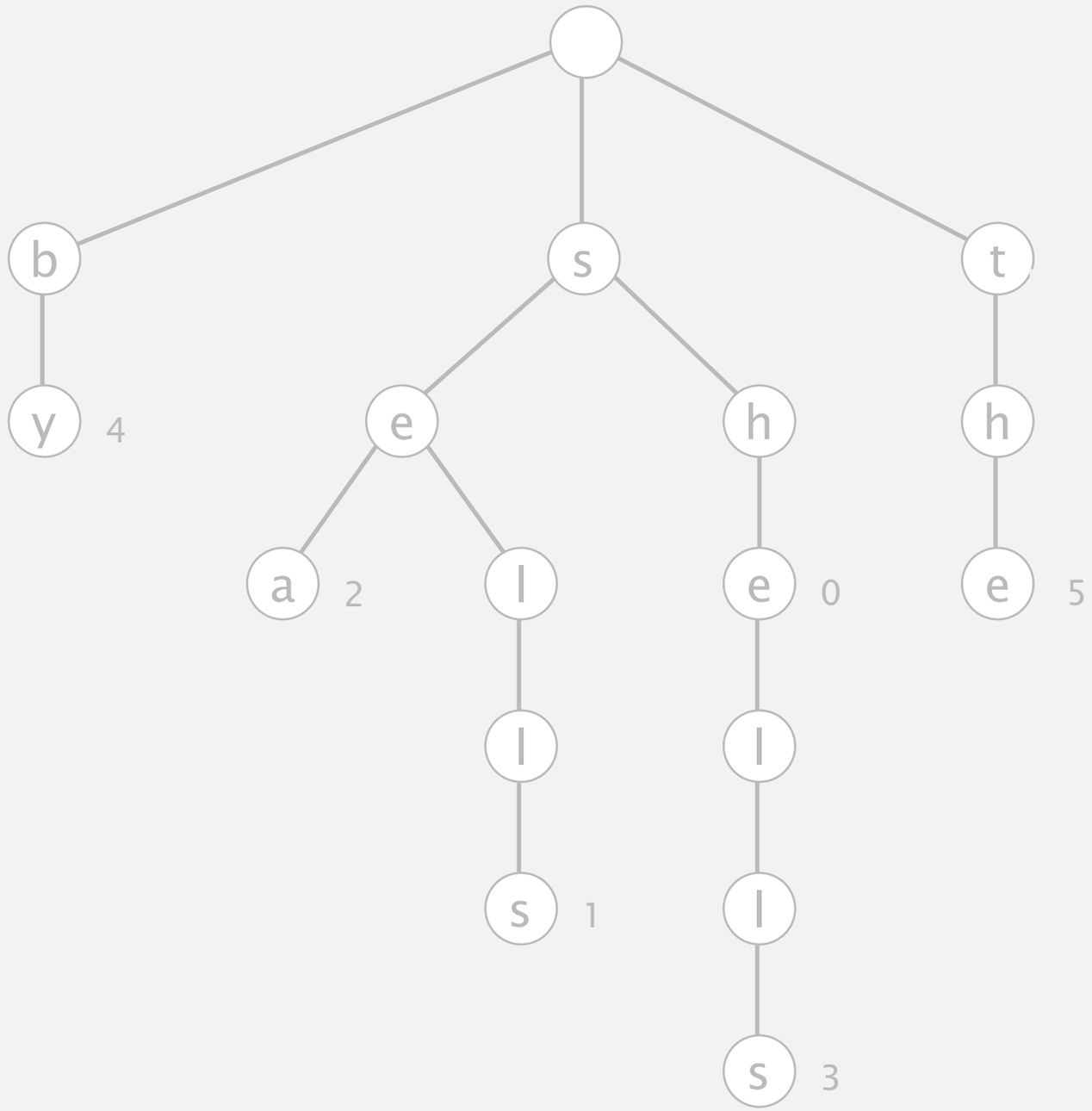
sells

sea

by

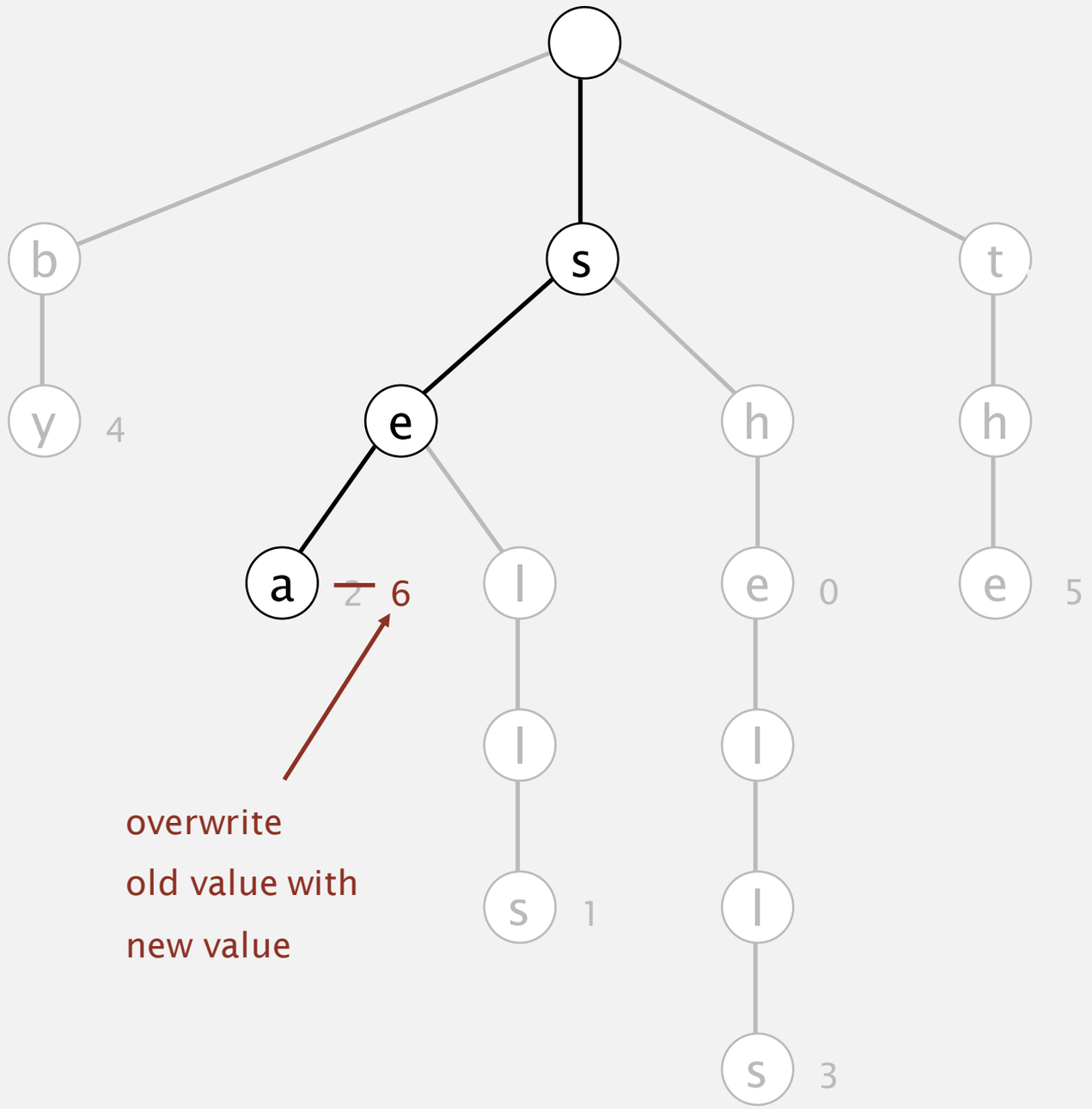
the

trie



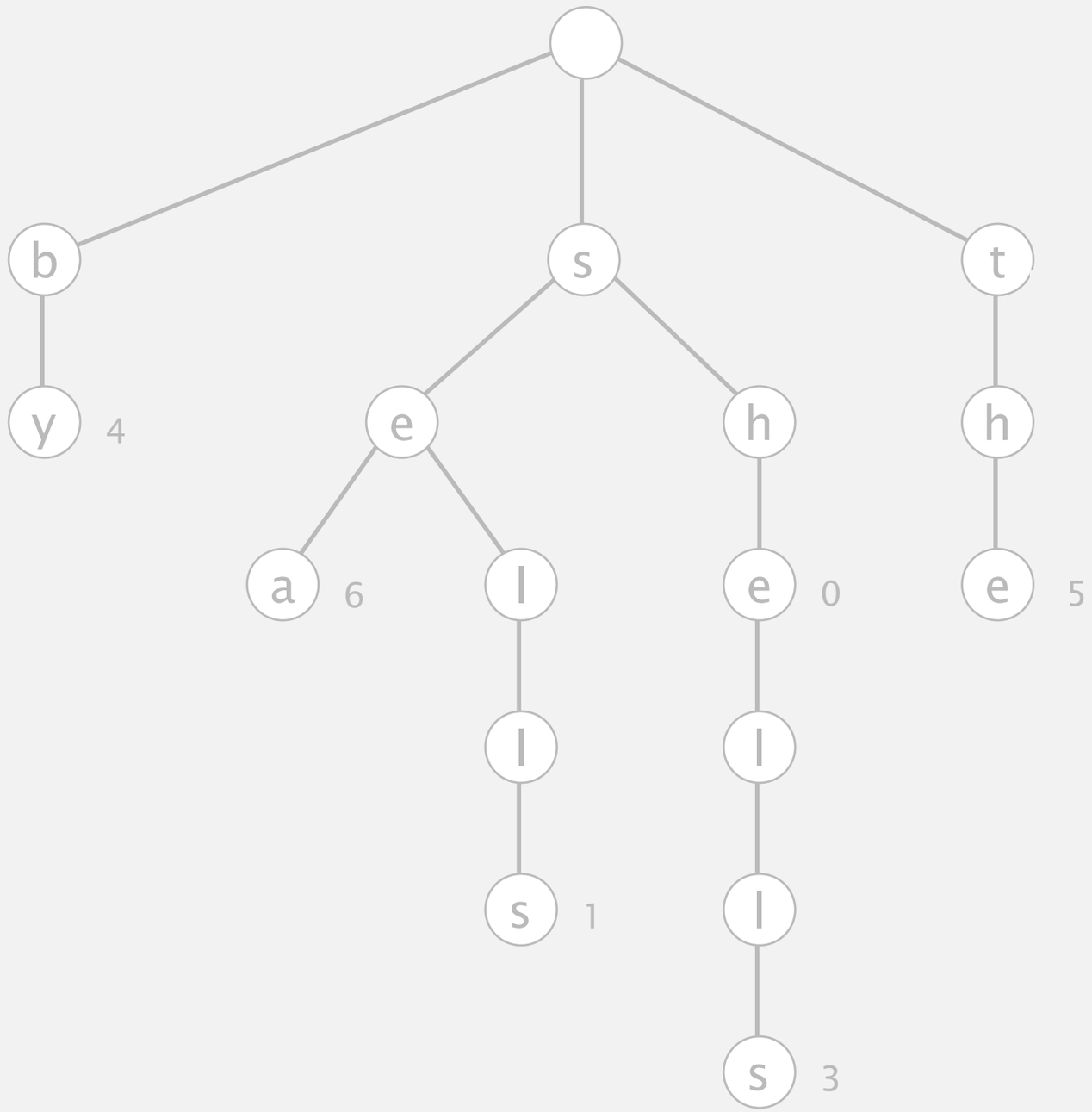
Trie construction demo

put("sea", 6)



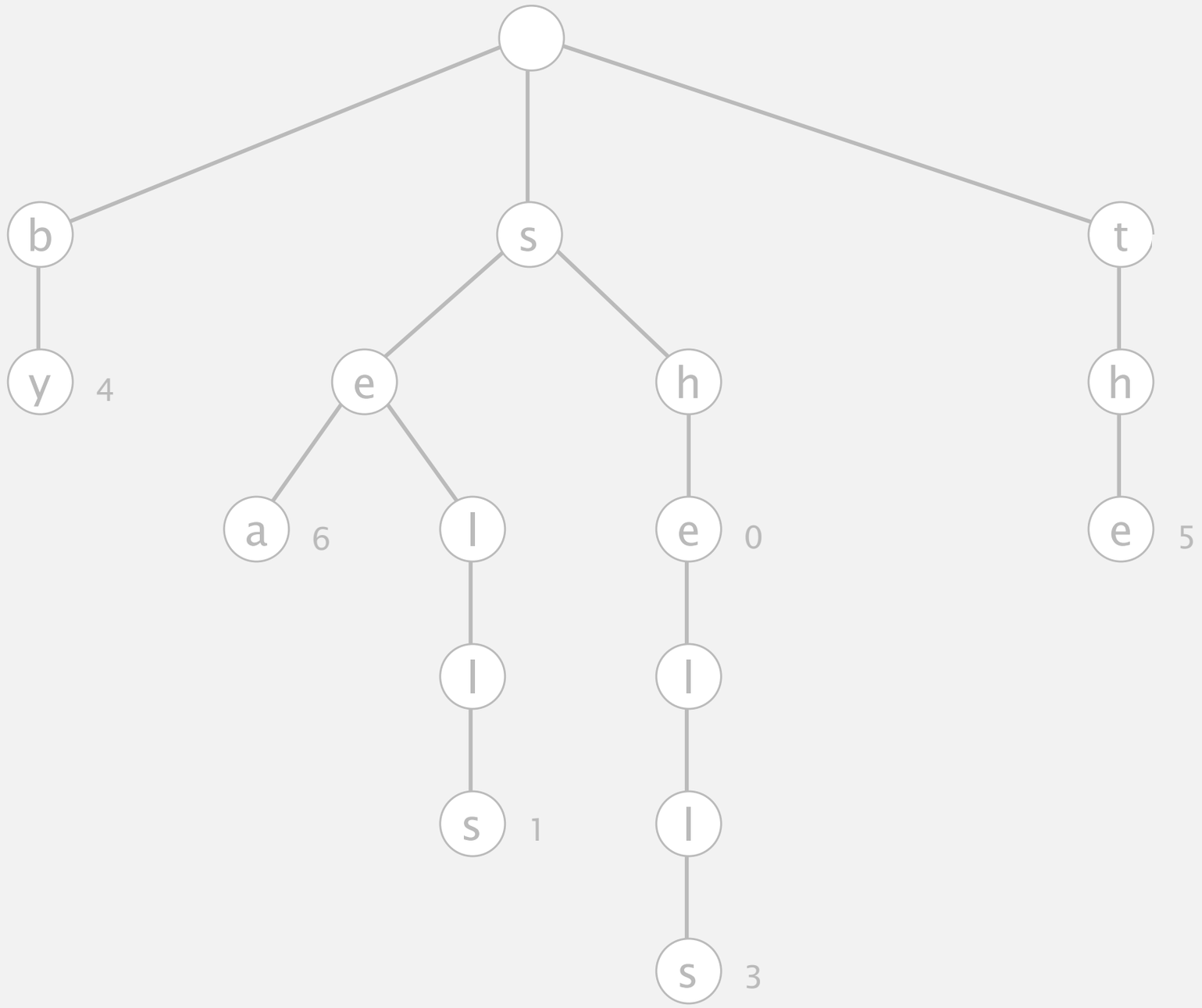
Trie construction demo

trie



Trie construction demo

trie



Trie construction demo

she

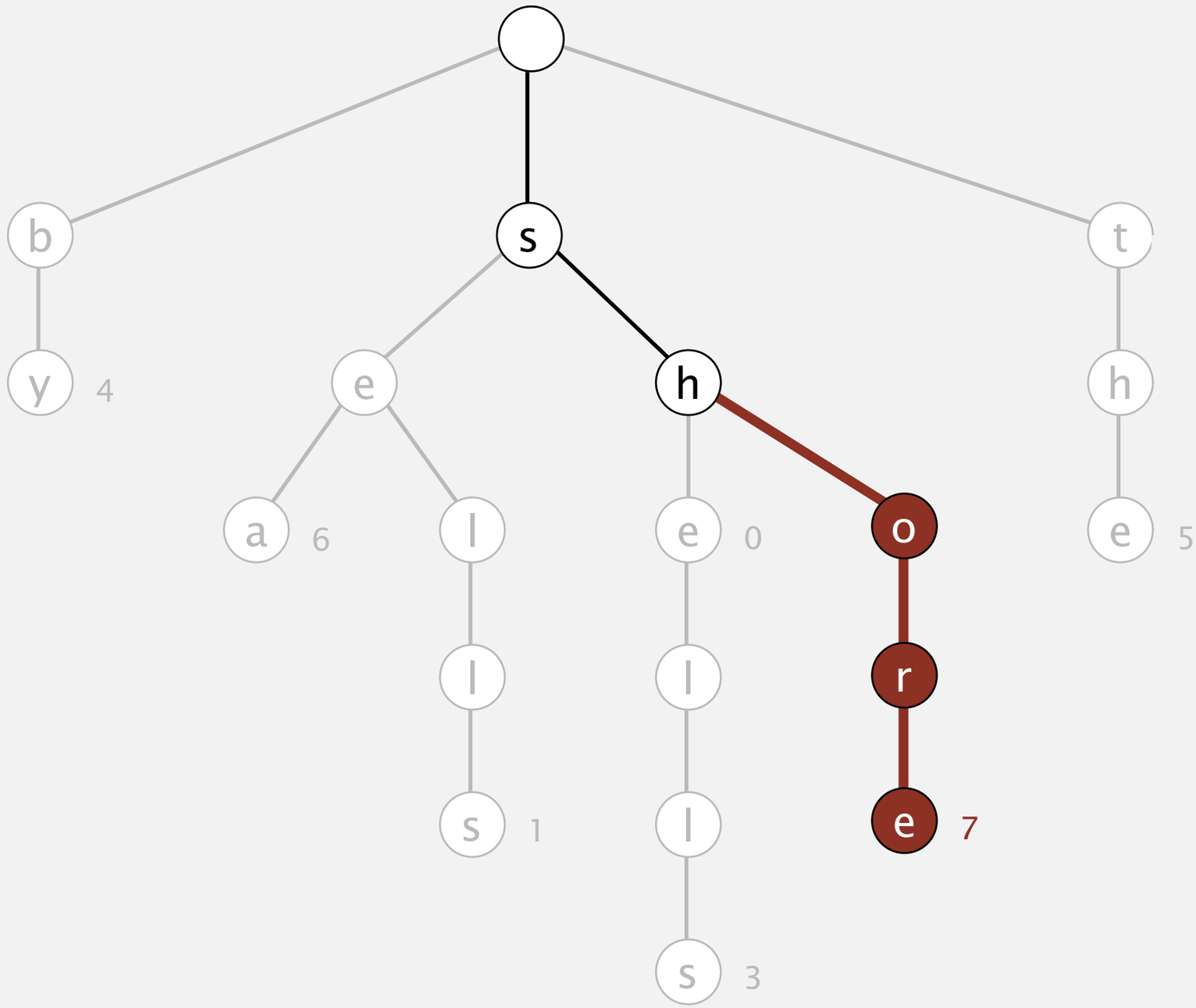
sells

sea

by

the

put("shore", 7)



Trie construction demo

she

sells

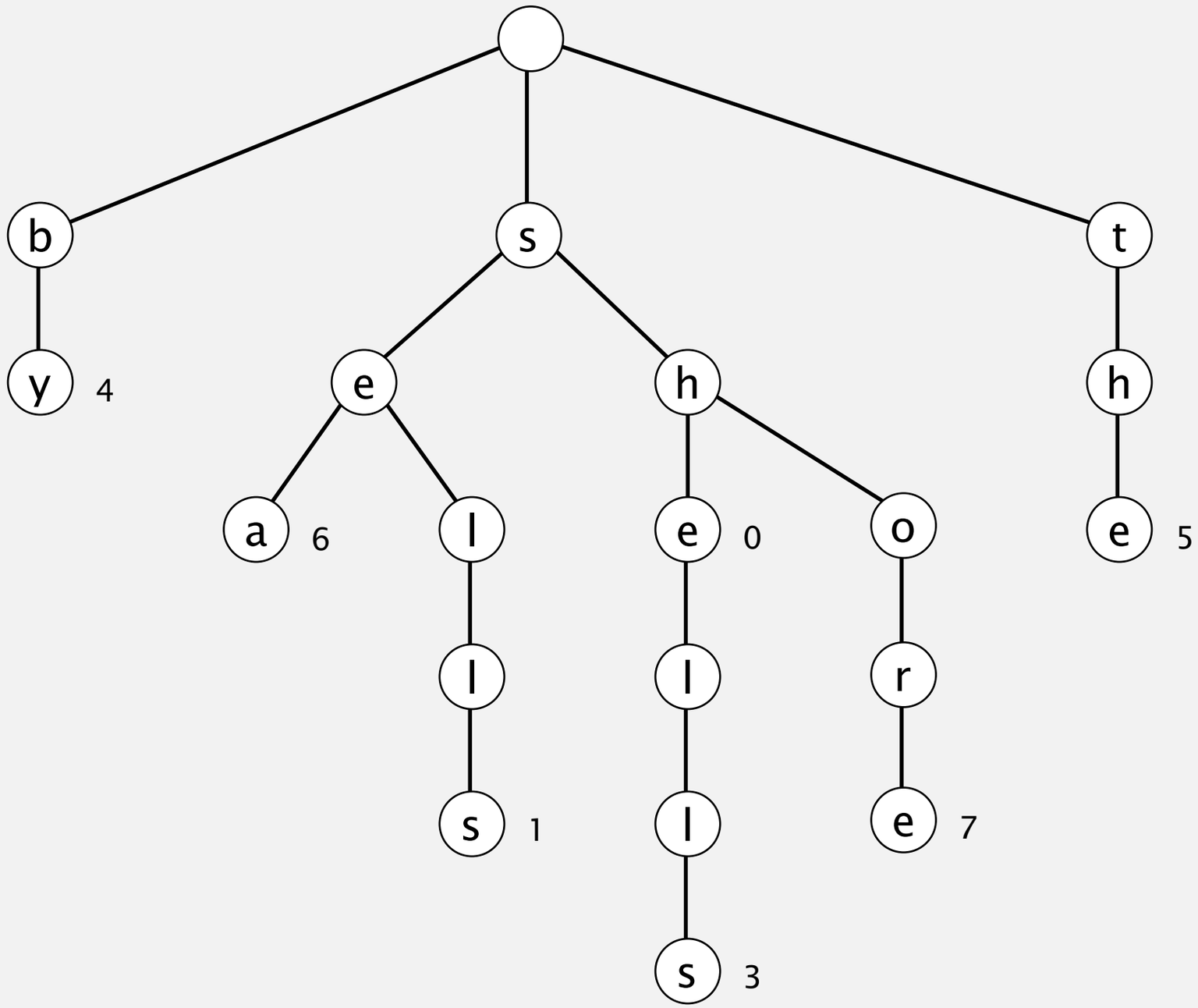
sea

by

the

shore

trie



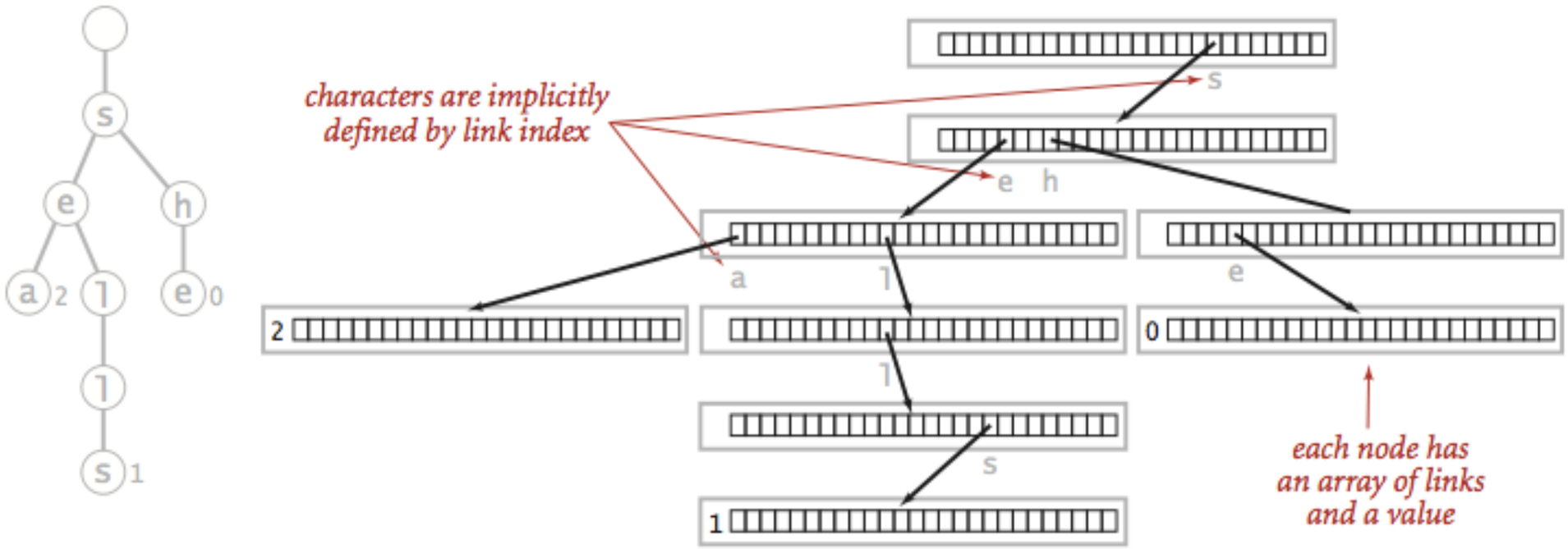
Trie representation: implementation

Node. A value, plus references to R nodes.

```
struct Node
{
   int value;
   Node * next[R];
}
```

A child node for each character in Alphabet.

No need to search for character, but a pointer reserved for each character in memory



Trie representation (R = 26)

R-way trie: implementation

```
#define R 256 ←——— extended ASCII

Node * root;

put(&root, key, val, 0);

void put(Node ** x, char *key, int val, int d)
{
   if (*x == null) *x = getNode();
   if (d ==strlen(key)) { *x->value = val; return;}
   char c = key[d];
   put(&(x->next[c]), key, val, d+1);
}

⋮
```

R-way trie: implementation (continued)

```
Node * getNode(){
    Node * pNode = NULL;
    pNode = (Node *)malloc(sizeof(Node));
    if (pNode){
        for (int i = 0; i < R; i++)
            pNode->next[i] = NULL;
    }
    return pNode;
}
```

R-way trie: implementation (continued)

```
int get(Node * x, char * key, int d)
   {
      if (x == null) return -1; //-1 refers no match
      if (d == strlen(key)) return x->value;
      char c = key[d];
      return get(x->next[c], key, d+1);
   }


}
```

Trie performance

Search hit. Need to examine all L characters for equality.

Search miss.

- Could have mismatch on first character.
- Typical case: examine only a few characters (sublinear).

Space. R null links at each leaf.

(but sublinear space possible if many short strings share common prefixes)

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

String symbol table implementations cost summary

	character accesses (typical case)			
implementation	search hit	Search miss	insert	space (references)
hashing (separate chaining)	N	N	1	N
R-way trie	L	$\log_R N$	L	RNw

N = number of entries, L= key length,

R= alphabet size, w= average key length

R-way trie.

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

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