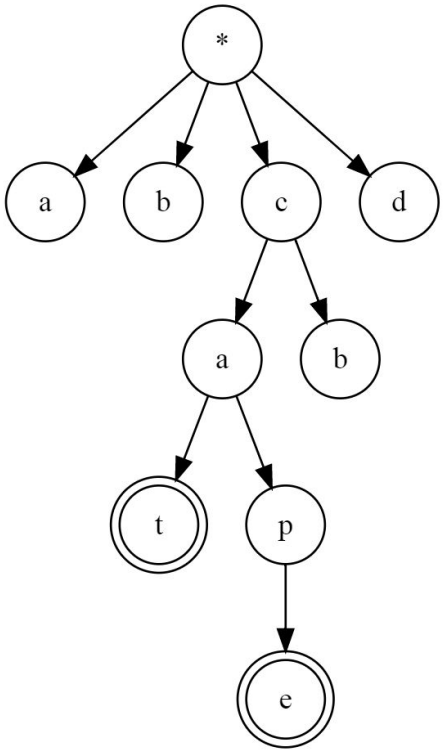
Tries/Prefix Trees

What are they?



- Data structure used for storing words or other similar information
  - Similar to a set, but makes traversal easier
- Nodes represent letters
- Paths represent sequences of letters
- “Final” nodes represent full words stored in the trie

A node has...

- A value
- A bool to indicate if it is terminal
- Array of 26 children (for each letter in alphabet)
  - Entry is null if there are no “child prefixes”
  - Entry points to another node if there is a “child prefix”

Detailed diagram on next slide

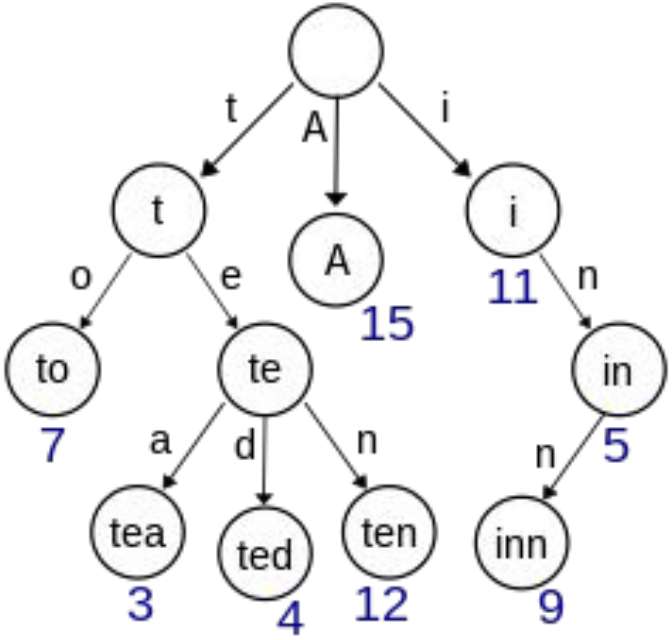
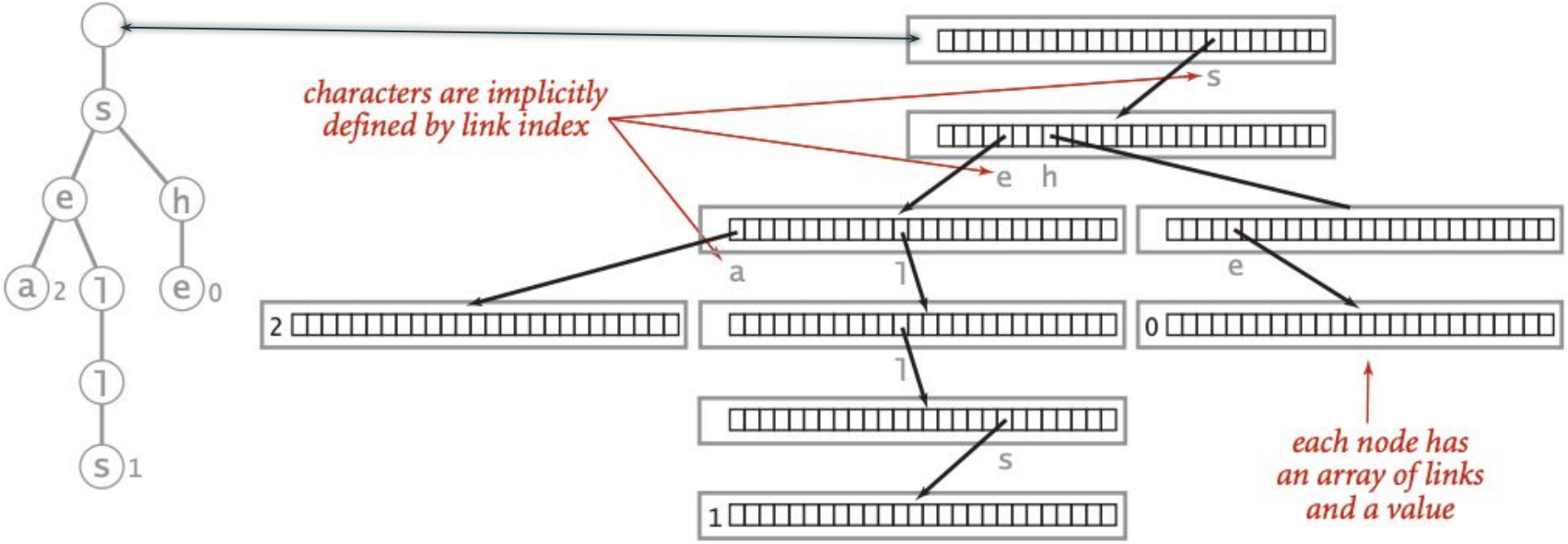
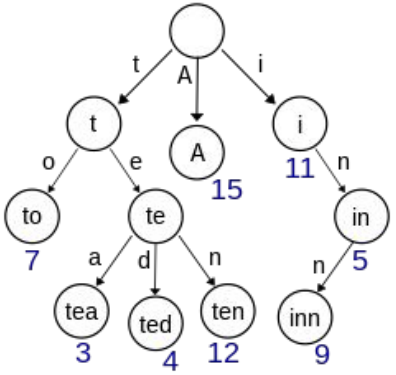


Diagram for previous slide



Important operations



Tries are similar to hashmaps or sets.

Insert

Inserts a word into the trie. *Strategy: go through each letter in the word. If there's no next node for that letter, make the node. Afterward, go to the node for that letter. Mark last node as "terminal."*

Search

Checks if a word is in the trie. *Strategy: go through each letter in the word. If there's no next node for that letter, the word is not in the trie. Otherwise, go to the node for that letter. If the search ends at a terminal node, the word is in the trie.*

Remove

Removes a word from the trie. *Strategy: similar to search, but when the word is found, mark it as non-terminal. (Gets more complicated if you want to remove unused nodes).*

Real-life use cases

Anything that involves string searching

- Predictive text
- Spell-checking

Pseudocode (ripped from Wikipedia)

Node data structure:

```
structure Node
    Children Node[Alphabet-Size]
    Is-Terminal Boolean
    Value Data-Type
end structure
```

Search algorithm:

```
Trie-Find(x, key)
    for 0 ≤ i < key.length do
        if x.Children[key[i]] = nil then
            return false
        end if
        x := x.Children[key[i]]
    repeat
    return x.Value
```

Insert algorithm

```
1   Trie-Insert(x, key, value)
2       for 0 ≤ i < key.length do
3           if x.Children[key[i]] = nil then
4               x.Children[key[i]] := Node()
5           end if
6           x := x.Children[key[i]]
7       repeat
8       x.Value := value
9       x.Is-Terminal := True
```