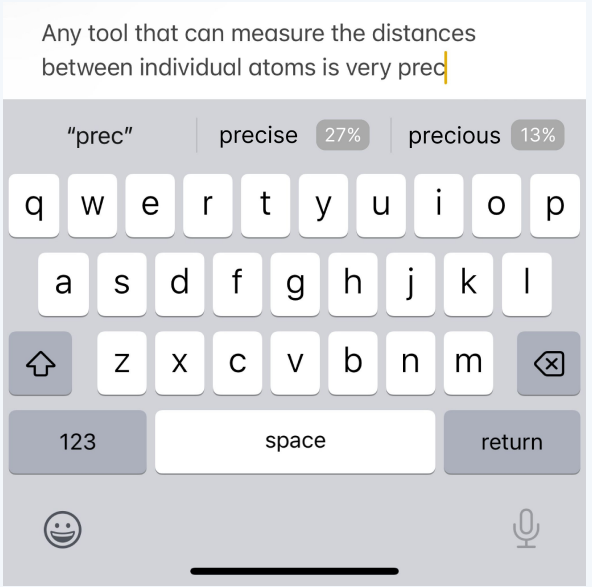
Tries Microteach

Motivation

Imagine you're writing a message on your phone. As you start typing a word, your phone suggests possible completions. How does this work?

- How do we store common words?
- How do we retrieve them based on a prefix?



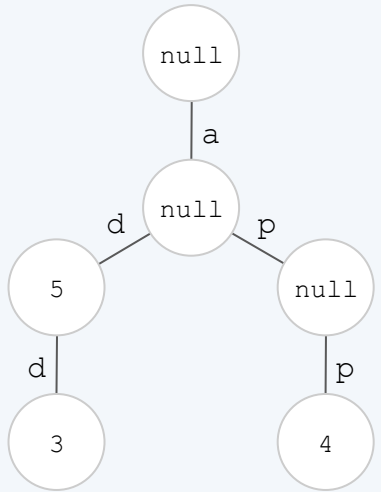
Tries

A trie is a tree-based data structure used primarily for efficient storage and retrieval of string-keyed data.

- Has regular tree components.
- Each branch stores part of a key.
- The key for a node is represented by the path from the root to that node.
- Nodes store the value corresponding to the key.

```
TrieNode {
  int value;
  Map<Character, Node> children;
}
```

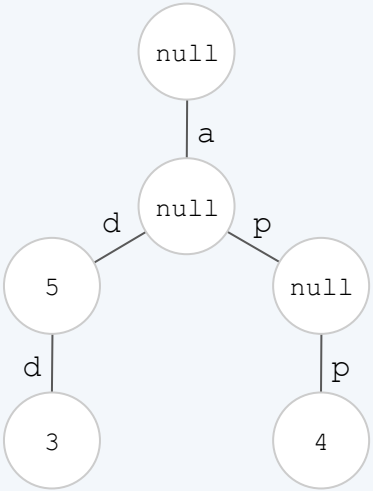
simple implementation of a Trie node

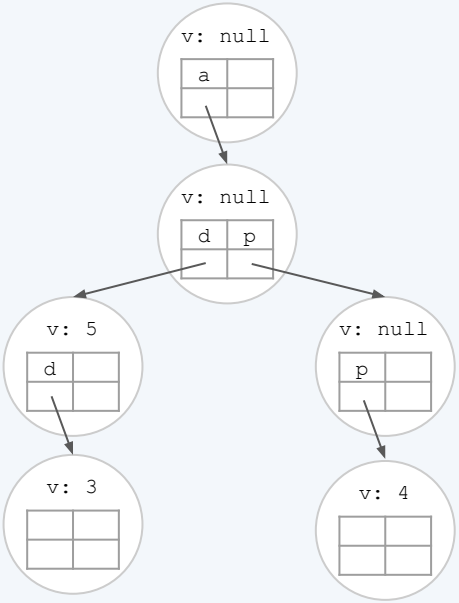


a Trie storing the key-value pairs ("ad", 5), ("add", 3), ("app", 4)

Drawing Tries

Generally, tries are drawn as a tree, where nodes have a value, and branches have a character.



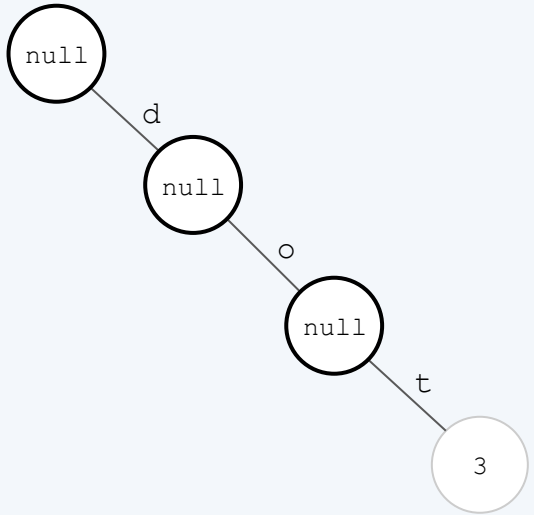


We may also draw tries such that nodes have a value, and also a table that maps characters (keys) to children nodes (values).

Tries: `insert("dot", 3)`

We want to insert the key-value ("dot", 3)

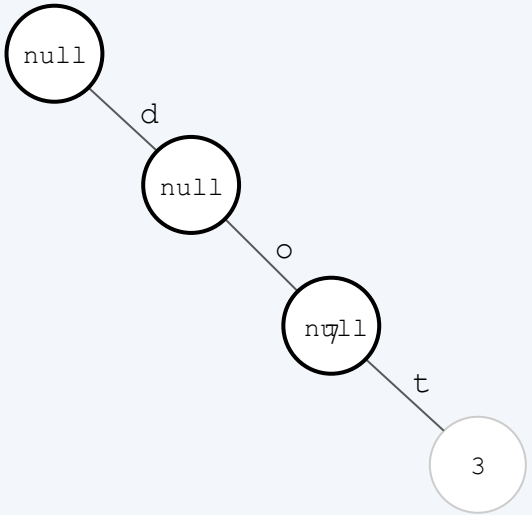
1. Start at the root
2. Create branch "d" to node with value null
3. Traverse down the "d" branch
4. Create branch "o" to node with value null
5. Traverse down the "o" branch
6. Create branch "t" to node with value 3



Tries: `insert("do", 7)`

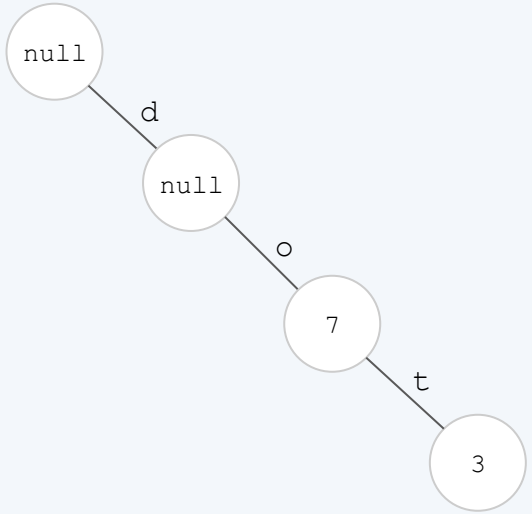
We want to insert the key-value (“do”, 7)

1. Start at the root
2. The “d” branch exists, so traverse down
3. The “o” branch exists, so traverse down
4. Update the current node’s value to 7



Tries: `find()`

```
find("")     // miss
find("d")    // miss
find("do")   // returns 7
find("dot") // returns 3
```

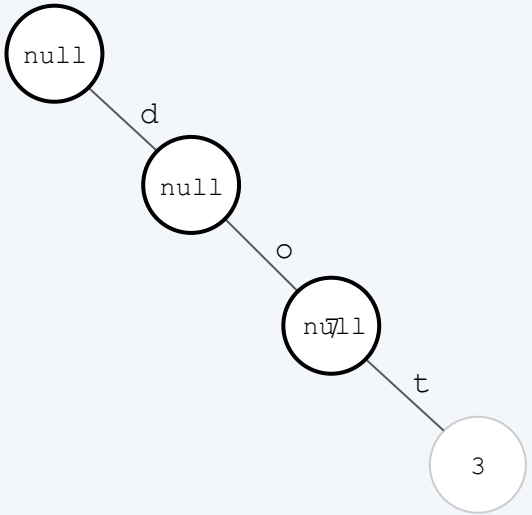


Questions?

Tries: `delete("do")`

We want to delete the key “do”

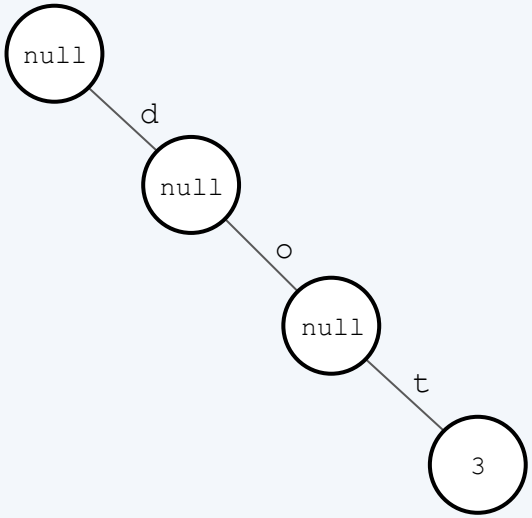
1. Start at the root.
2. The “d” branch exists, so traverse down
3. The “o” branch exists, so traverse down
4. The node we want to remove has a child, so only remove the node’s value (i.e. lazy deletion)



Tries: `delete("dot")`

We want to delete the key "dot"

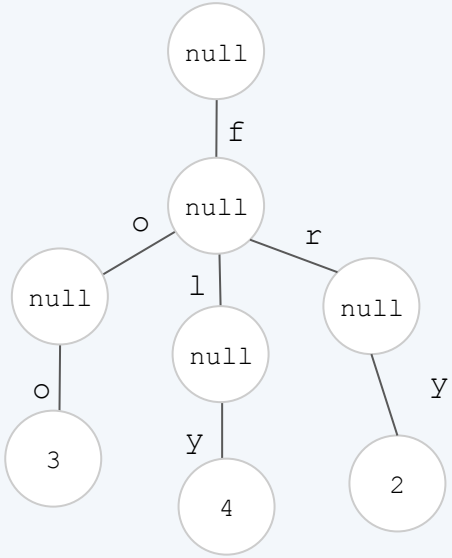
1. Start at the root
2. The "d" branch exists, so traverse down
3. The "o" branch exists, so traverse down
4. The "t" branch exists, so traverse down
5. The node we want to remove has no children, so remove the entire node
6. Backtrack through previously traversed nodes and remove them until we either reach a node with value, a node with children, or the root



Problem 1

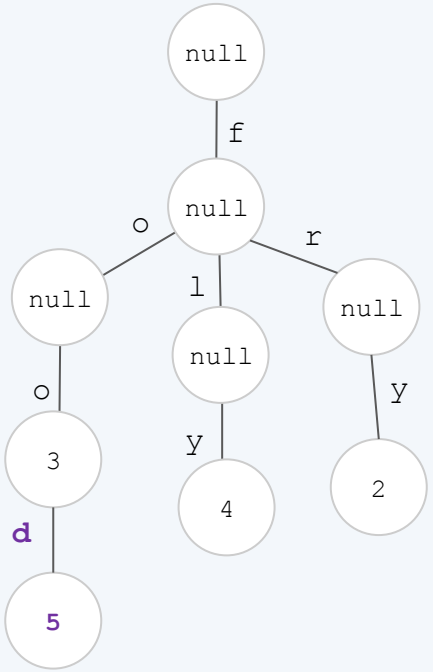
Problem 1

Try insert, find, and delete operations on the following Trie.



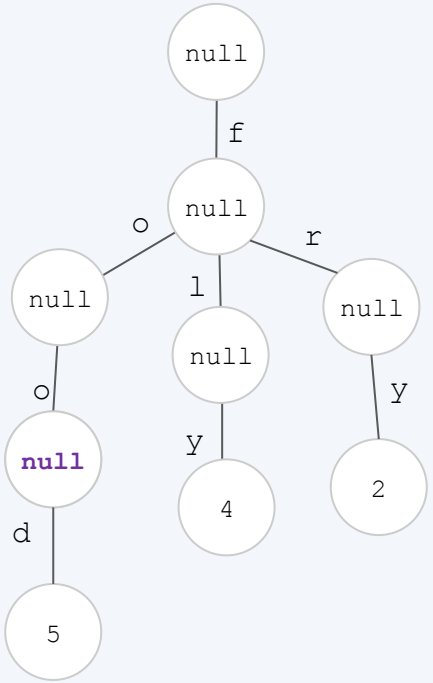
Problem 1a: insert("food", 5)

Answer:



Problem 1b: delete(“foo”)

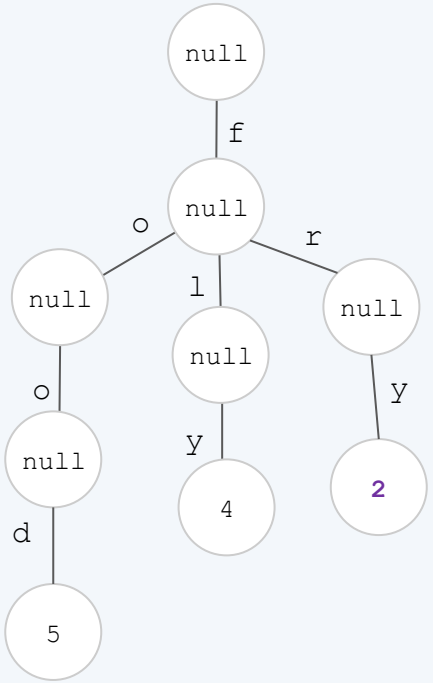
Answer:



Problem 1c: find(“fry”)

Answer:

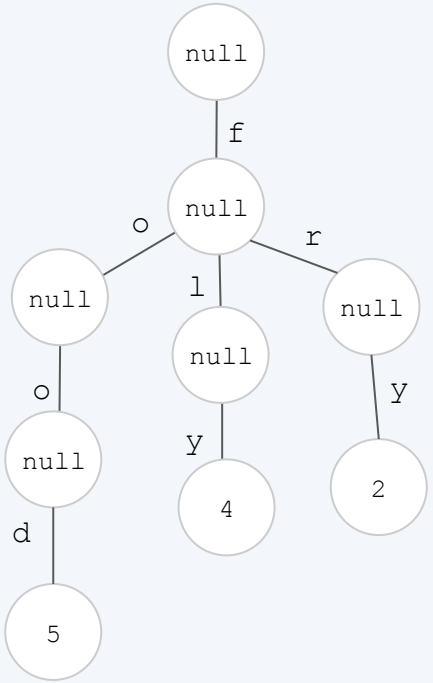
find(“fry”) //returns 2



Problem 1d:

Answer:

("food", 5)
("fly", 4)
("fry", 2)



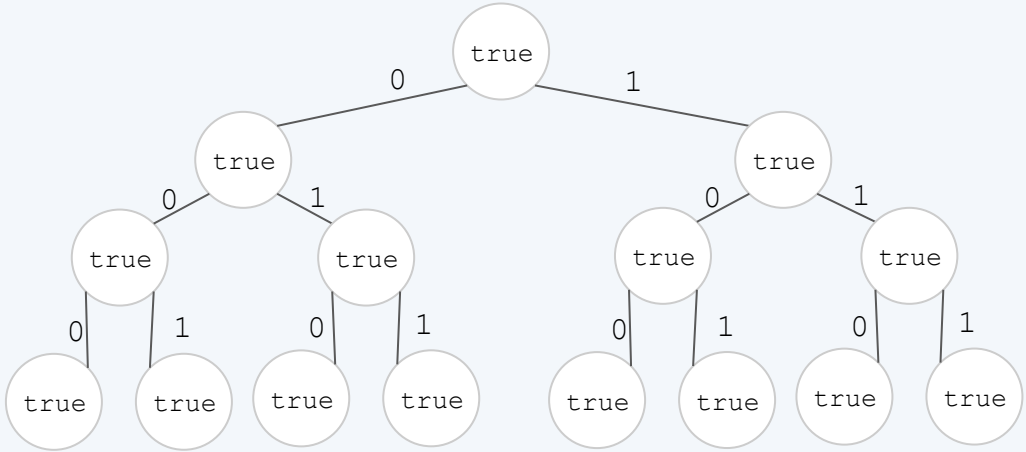
Questions?

Problem 2

Problem 2a

Insert all possible binary strings of lengths 0-3 (i.e. ‘’, ‘0’, ‘1’, ..., ‘110’, ‘111’) into a trie.

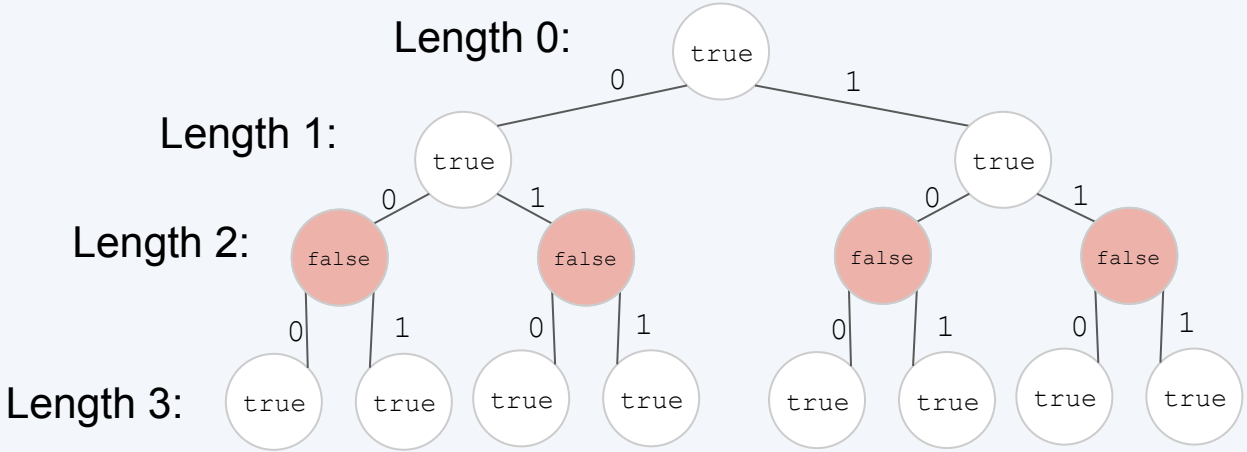
Answer:



Problem 2b

From here, remove all binary strings of length 2. How many nodes would disappear? Why?

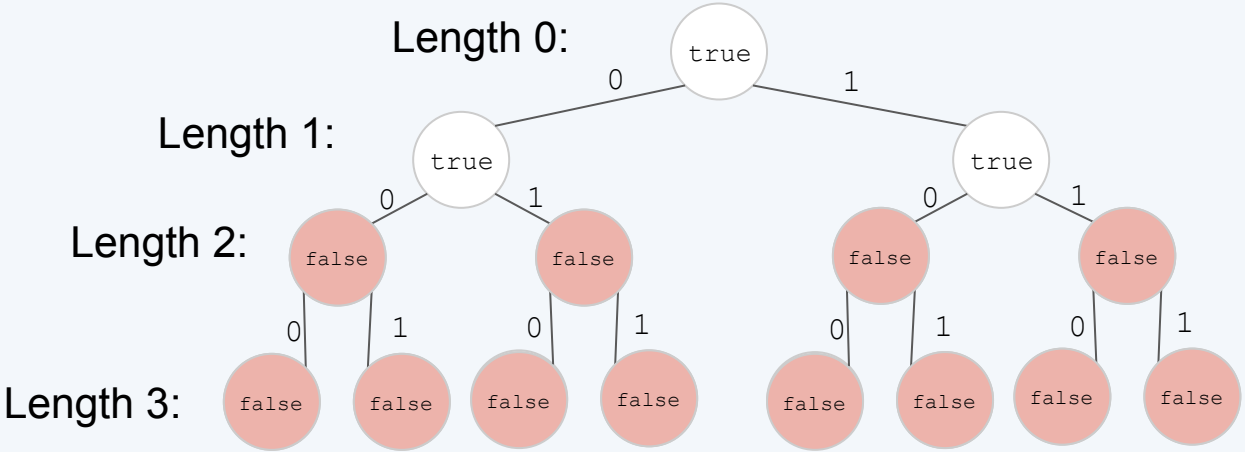
Answer: 0 nodes. We still need to maintain pointers to nodes representing binary strings of length 3. Therefore, we only update the values in the nodes to null.



Problem 2c

From here, remove all binary strings of length 3. How many nodes would disappear? Why?

Answer: 12 nodes. We can delete all nodes representing binary strings of length 2-3 since these do not point to any relevant nodes.



Thank You!