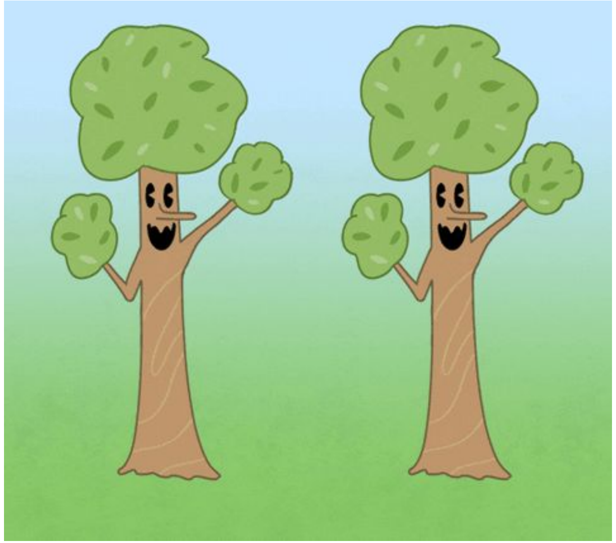
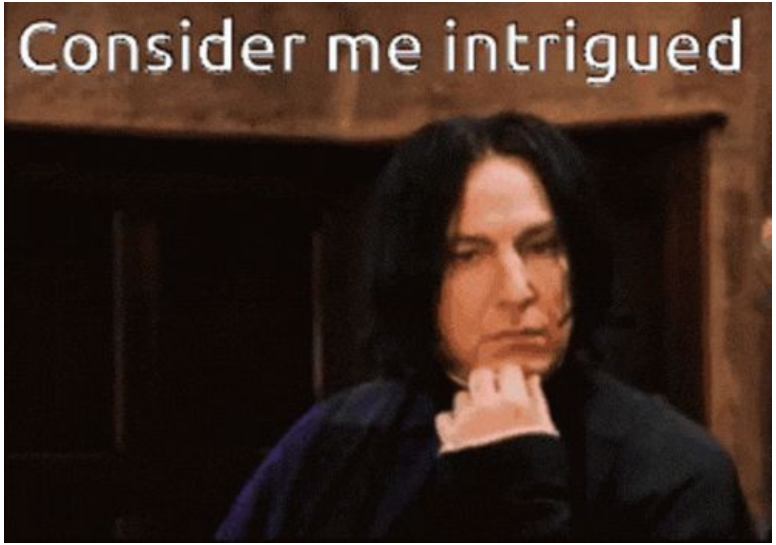
Tries

(prefix trees)

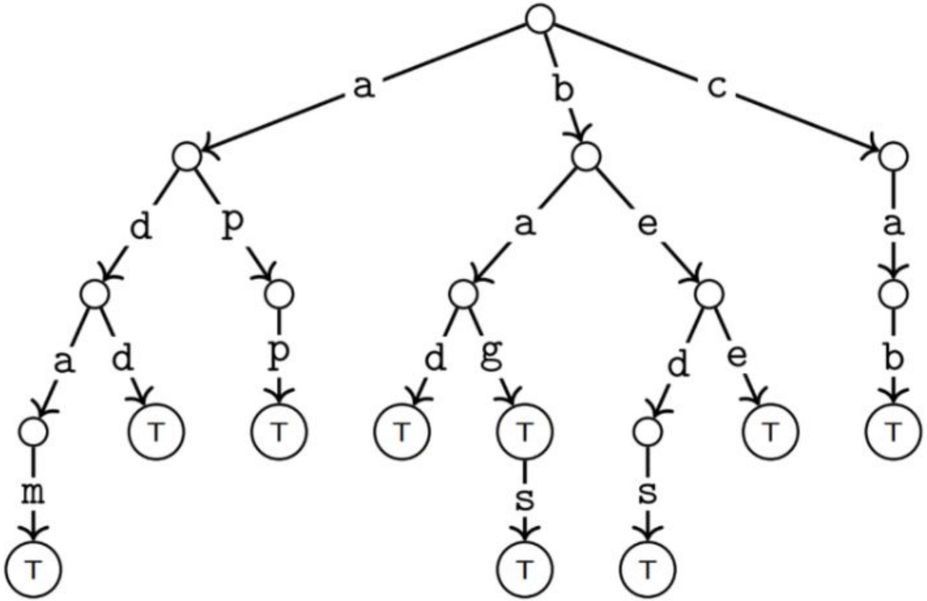


What are Tries?

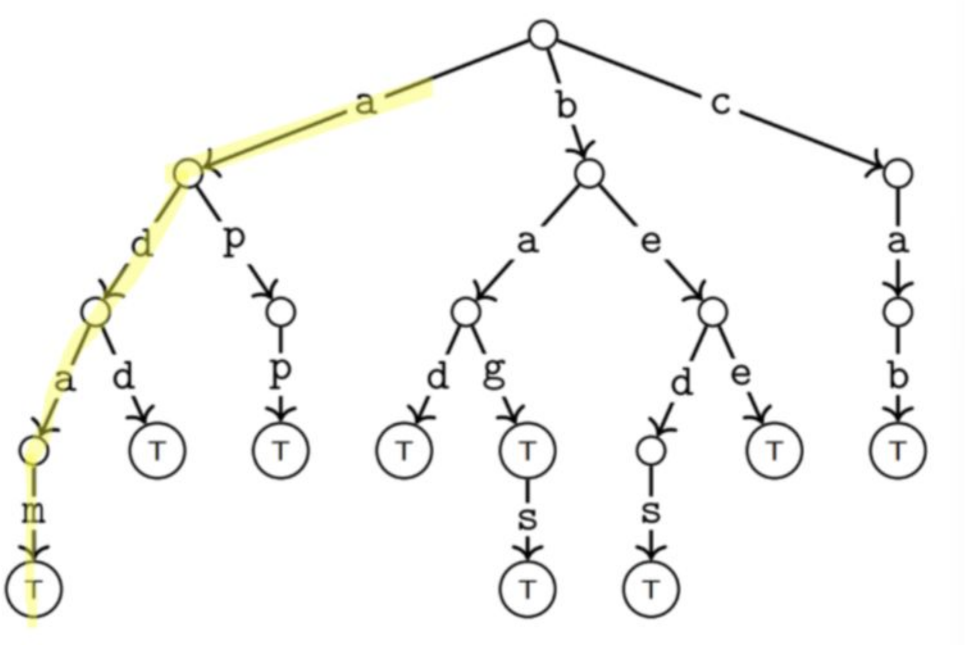


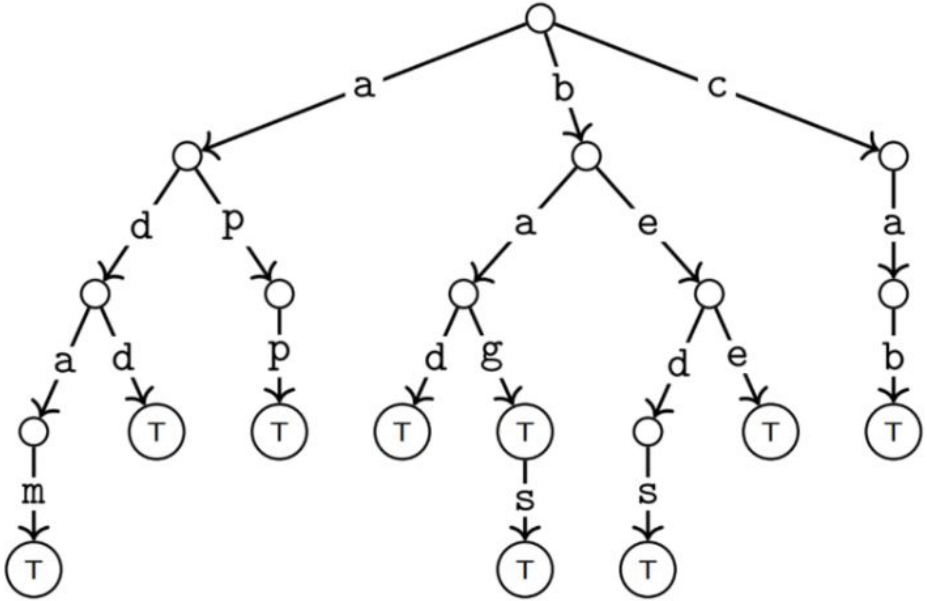
What are Tries?

- Dictionary made for storing “words”



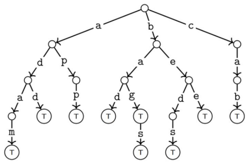
Ta Da!





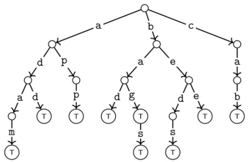
This trie represents the dictionary: `{adam, add, app, bad, bag, bags, beds, bee, cab},`

What are Tries?



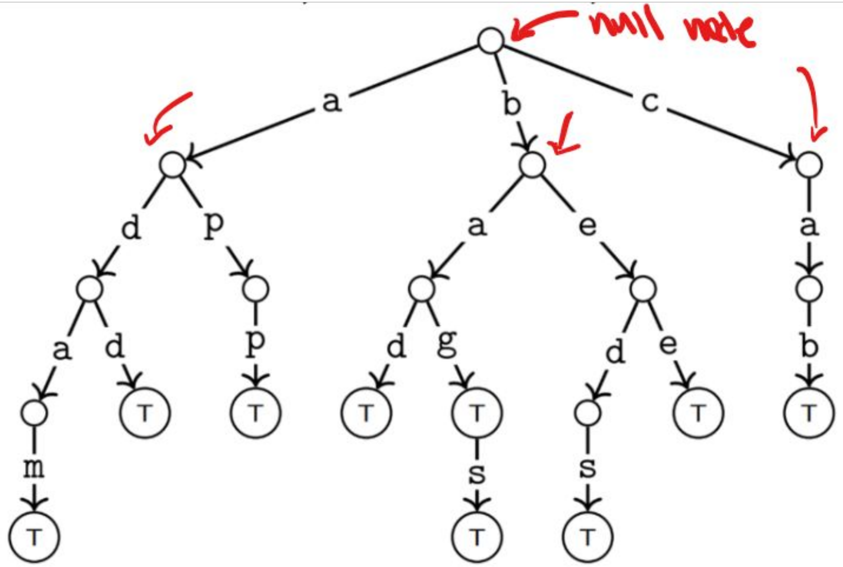
- Tree-based data structure
- Also called prefix trees or digital trees

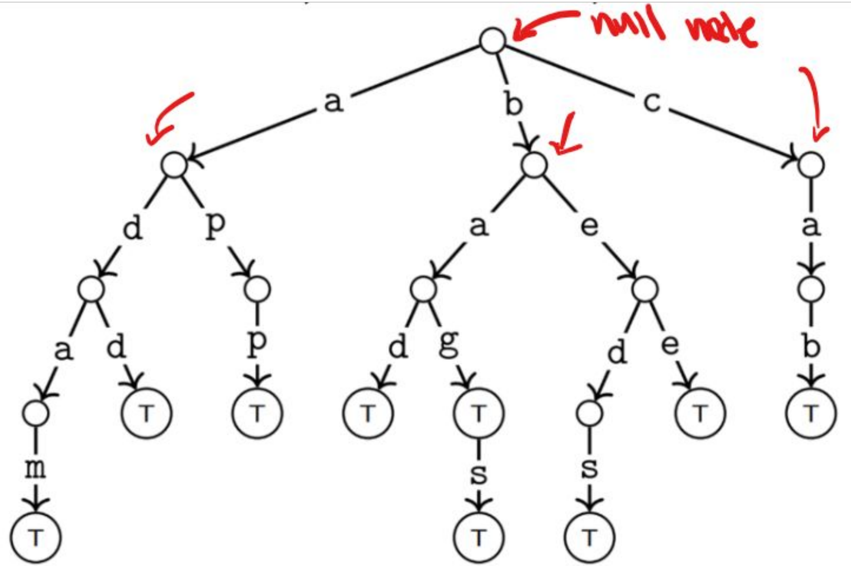
What are Tries?



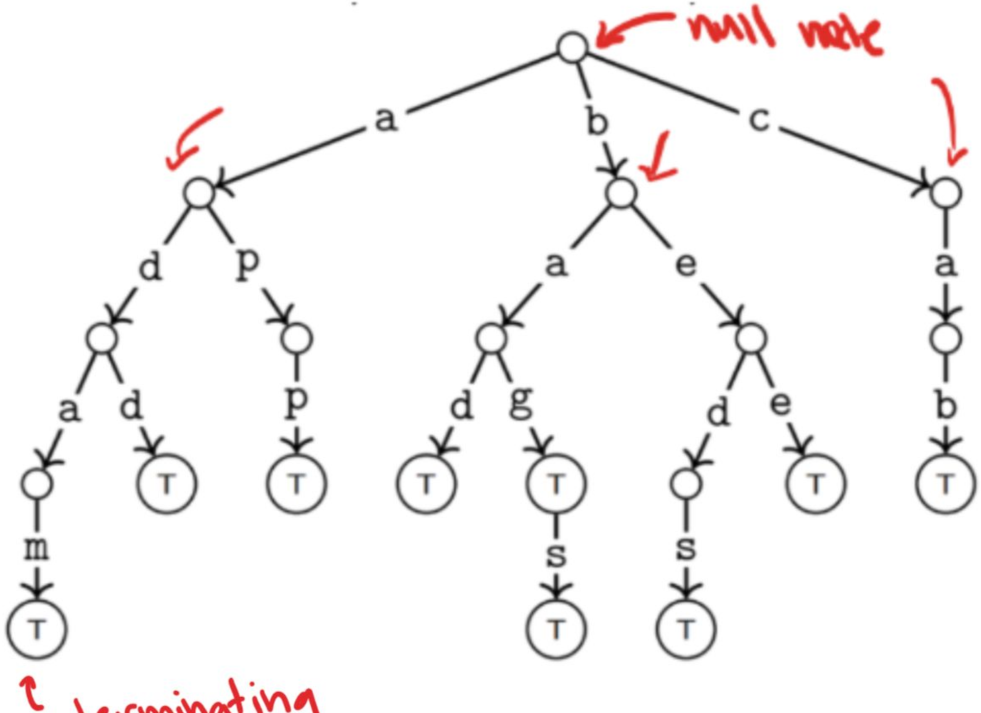
- Tree-based data structure
- Also called prefix trees or digital trees
- Re**trie**val
  - Retrieving things, retrieving strings of symbols
  - Words out of characters: {a, d, a, m}, Genome Sequences {T, C, T, T, A, G, A}
- Can store anything that can be turned into a sequence over a finite set/alphabet
  - Characters | bits | digits | path segments (C:/users/hana/downloads) | tuples | etc

How do they work?


null node
a
b
c
d
p
a
e
a
a
d
p
d
g
d
e
b
T
T
T
T
T
T
m
s
s
T
T
T



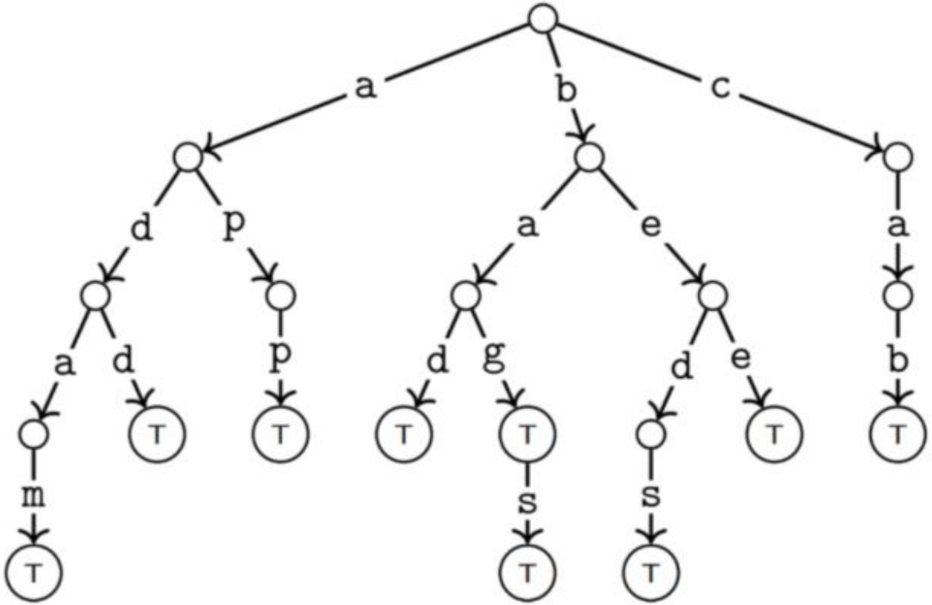
- Root node
- Empty String Node
- Empty Prefix Node



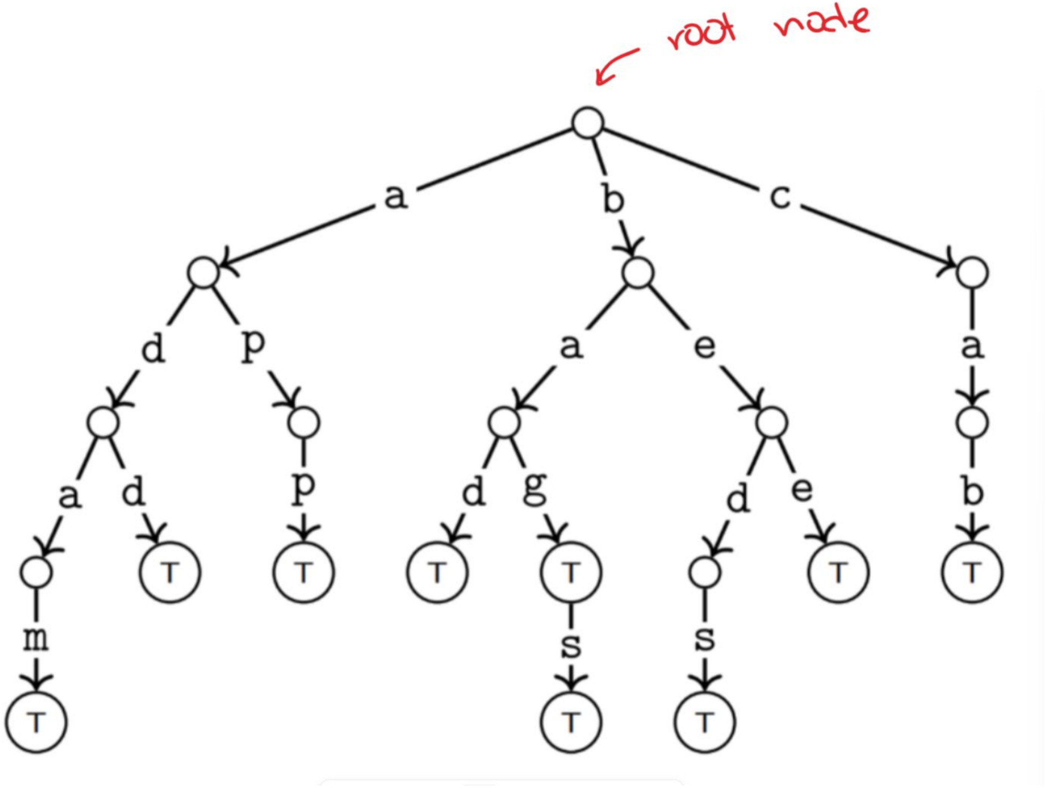
null node

terminating nodes

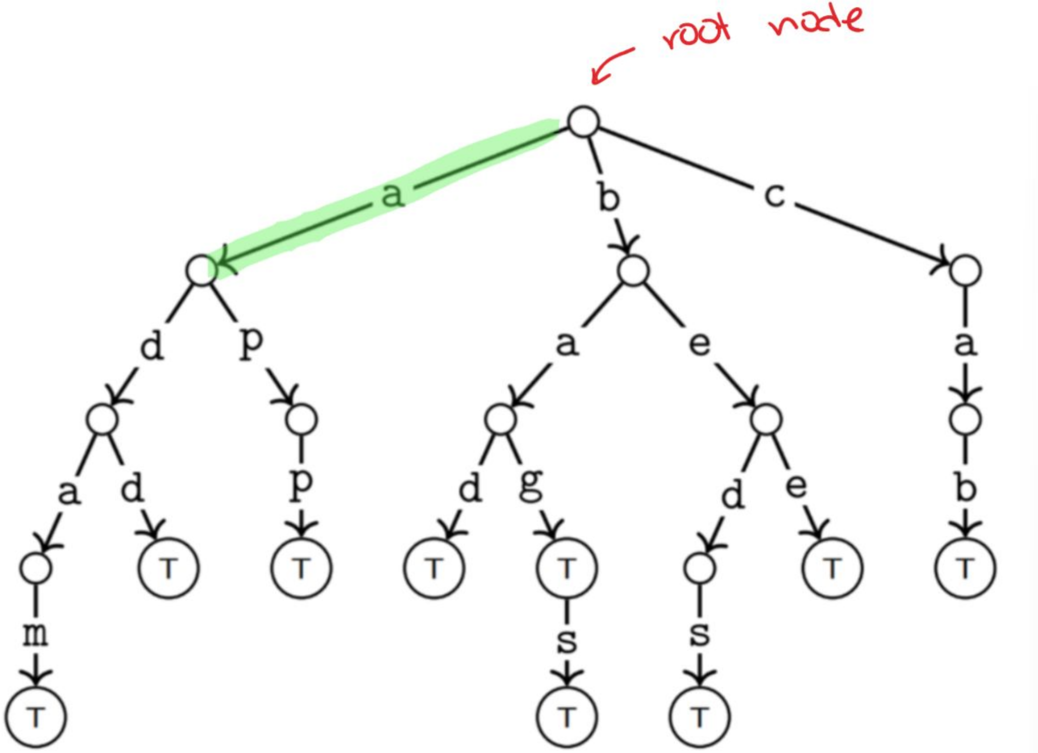
Find (“add”)



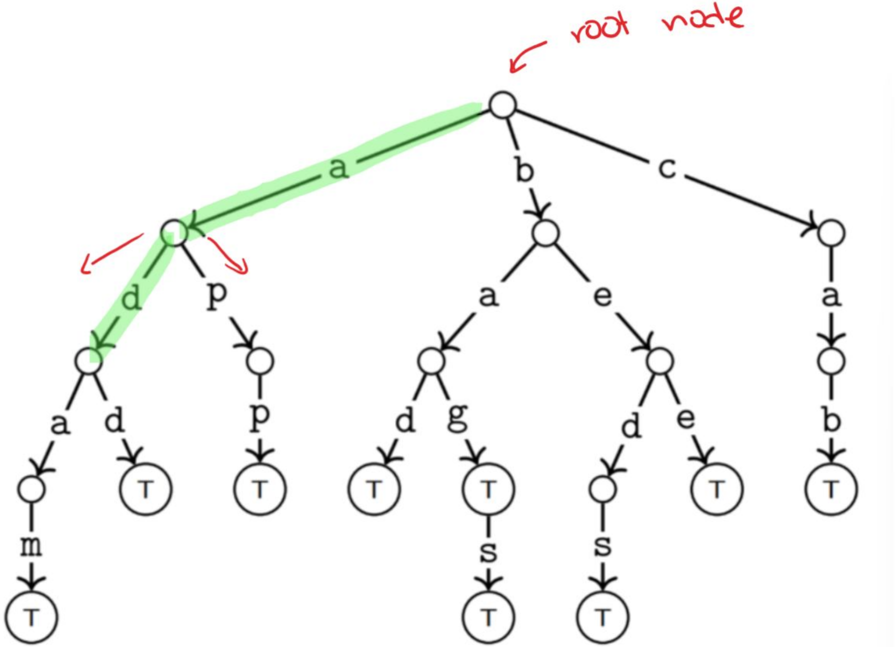
Find (“add”)



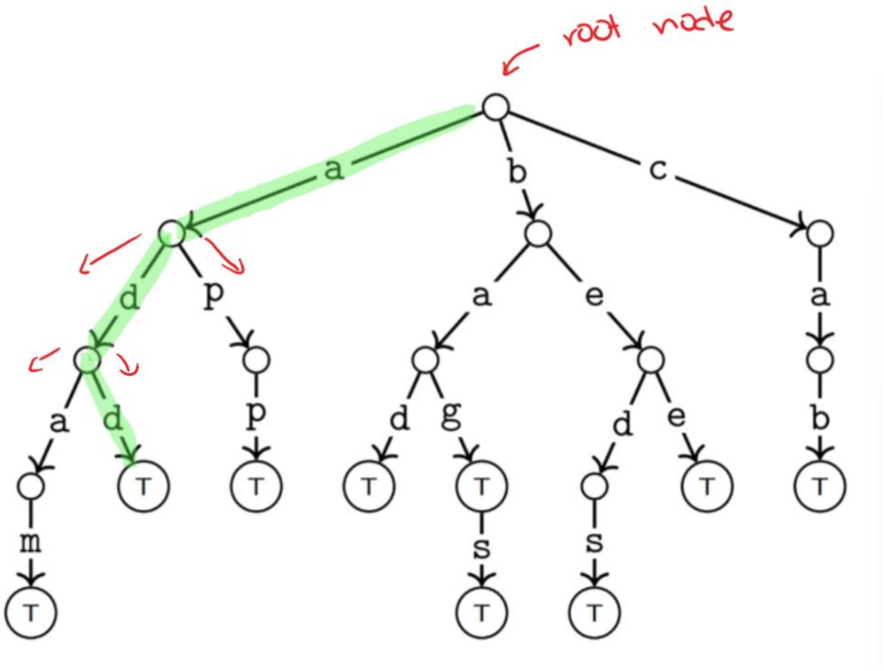
Find ("add")



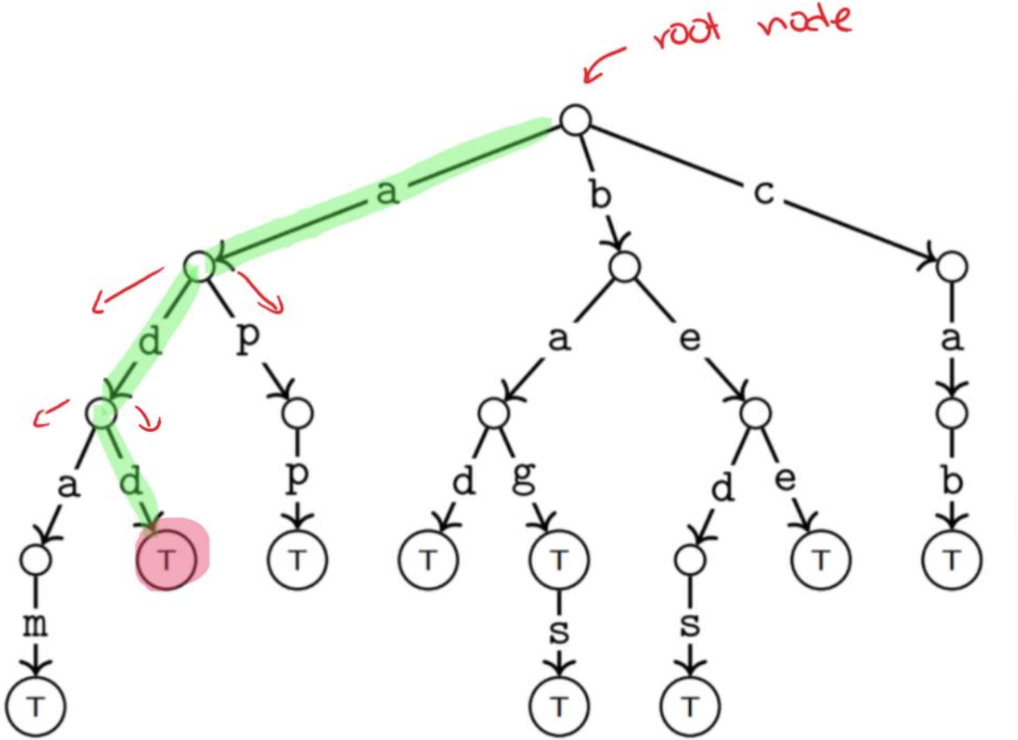
Find (“add”)



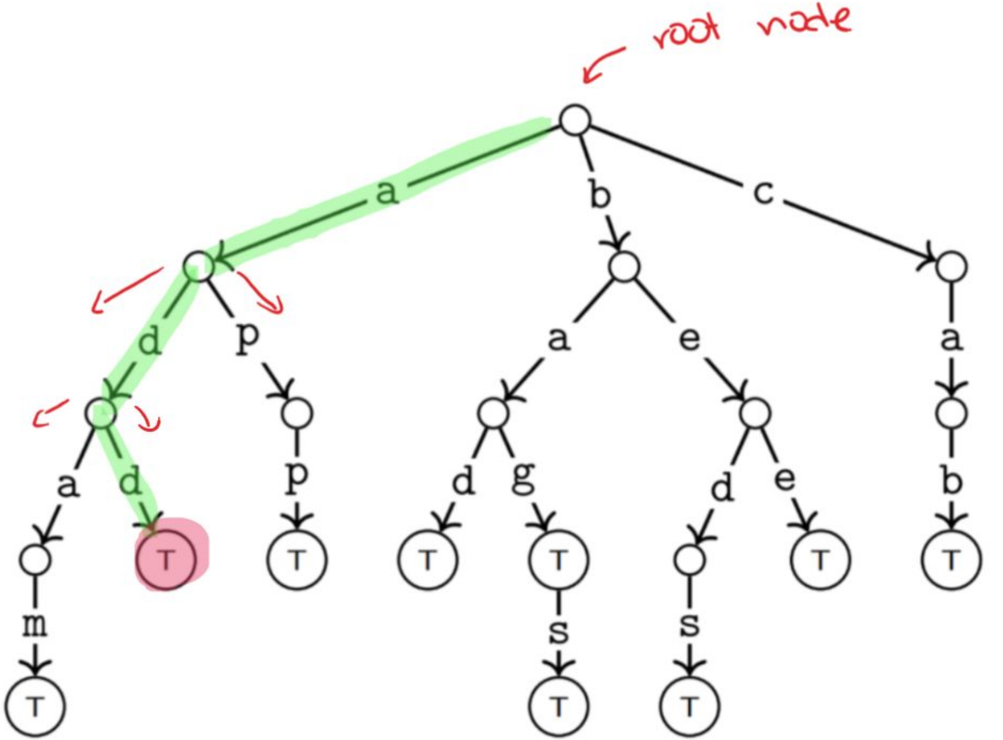
Find ("add")



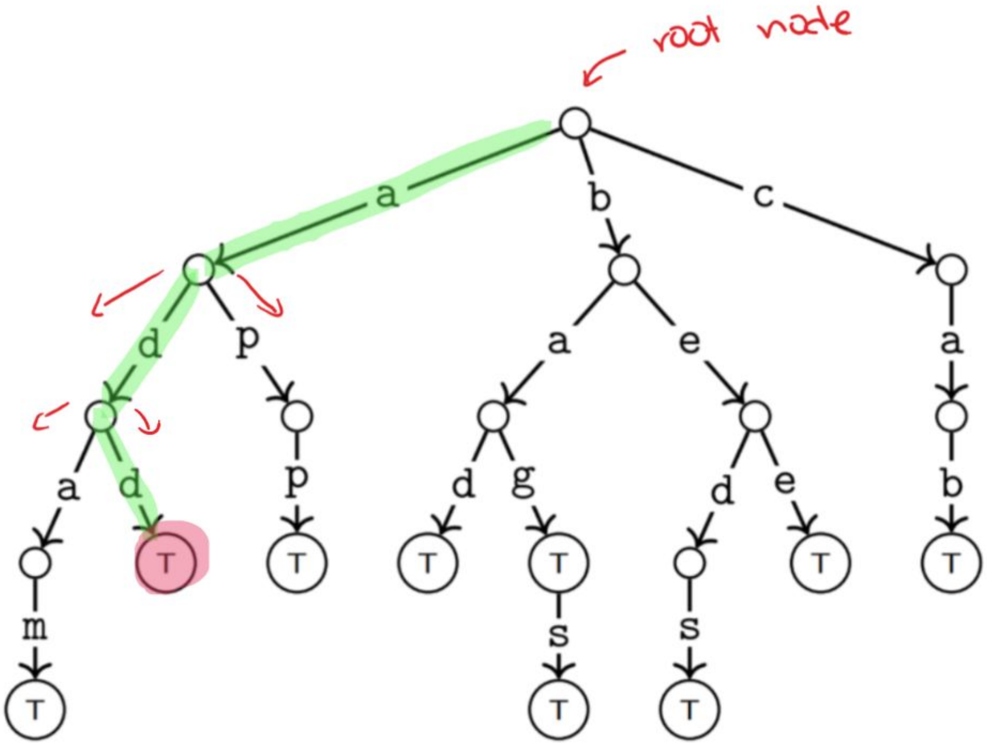
Find ("add")



Find (“add”)



Find (“add”)

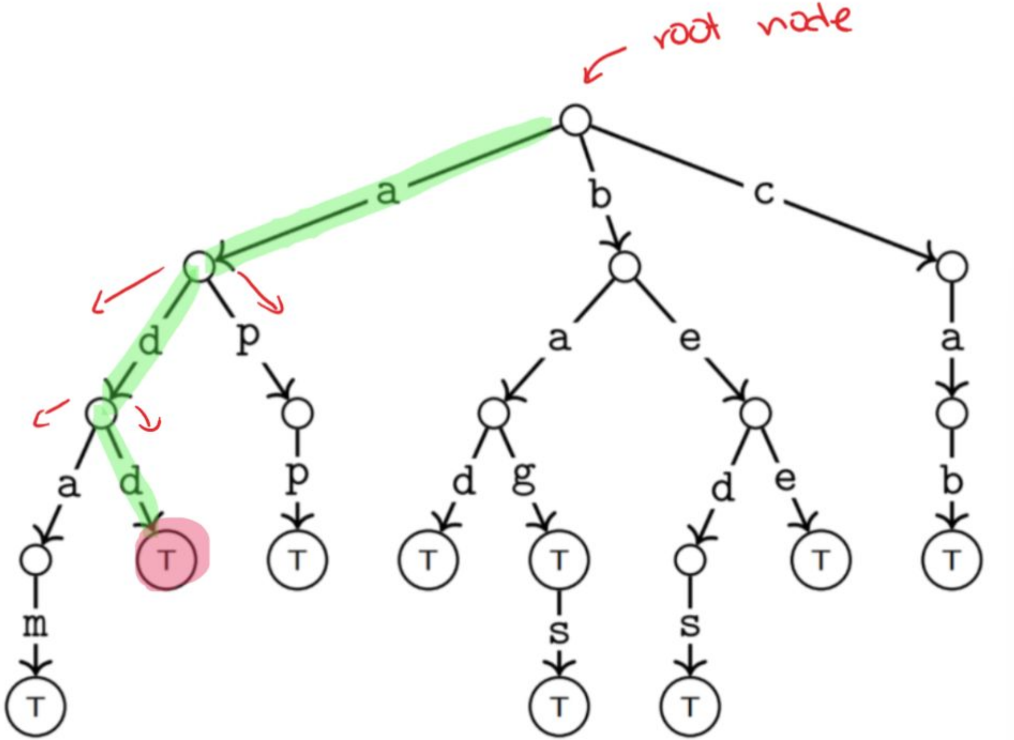


Runtime of Find:

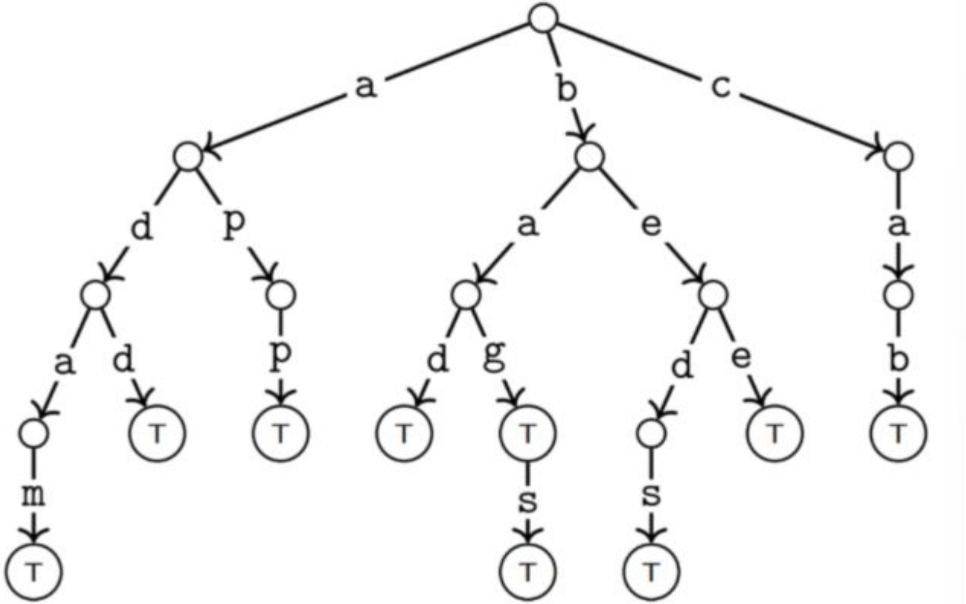
O(L)

Where L is the length of the key you’re searching for

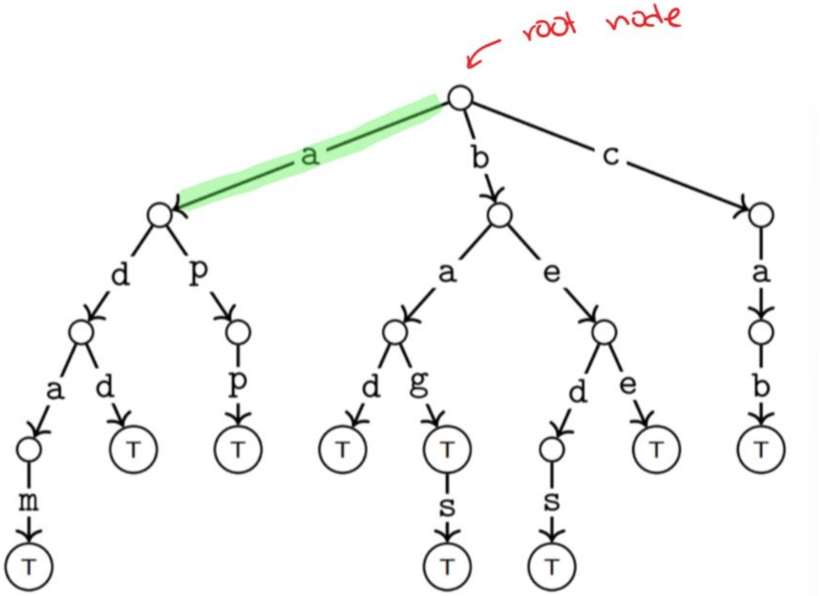
Find ("a")



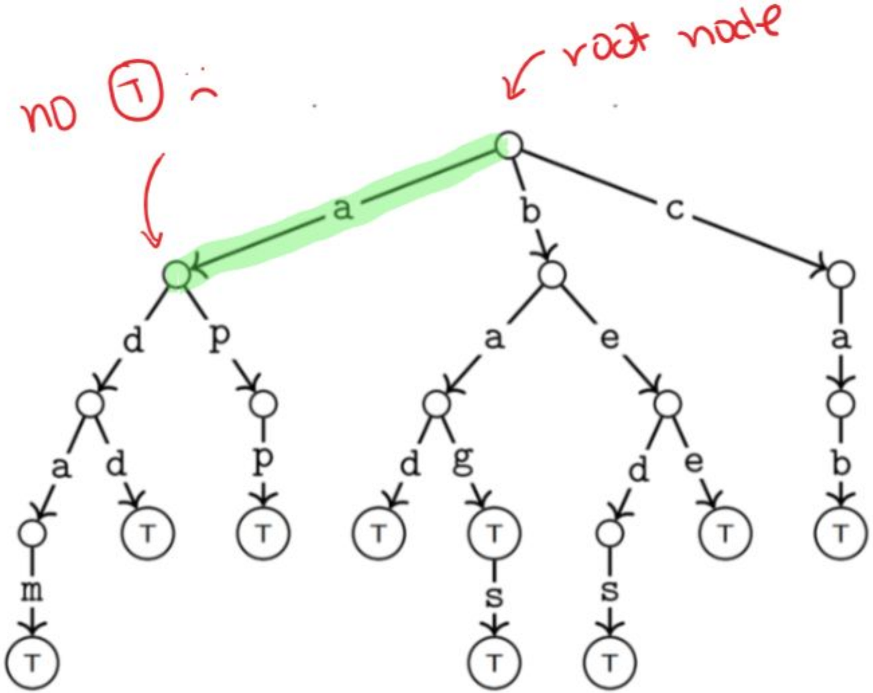
Find (“a”)

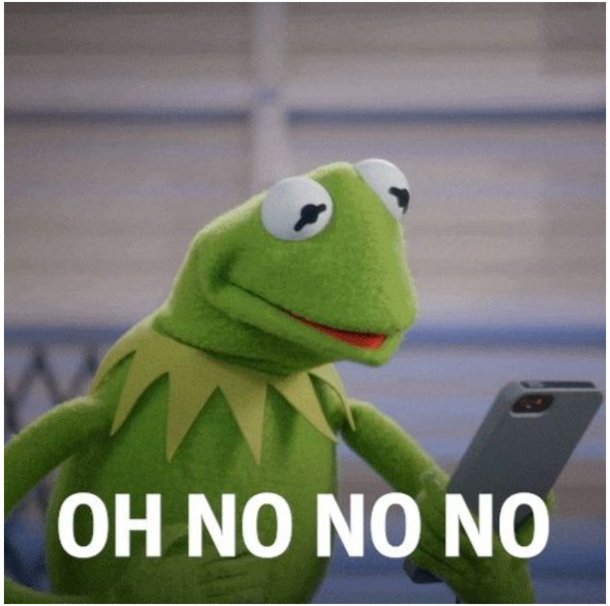


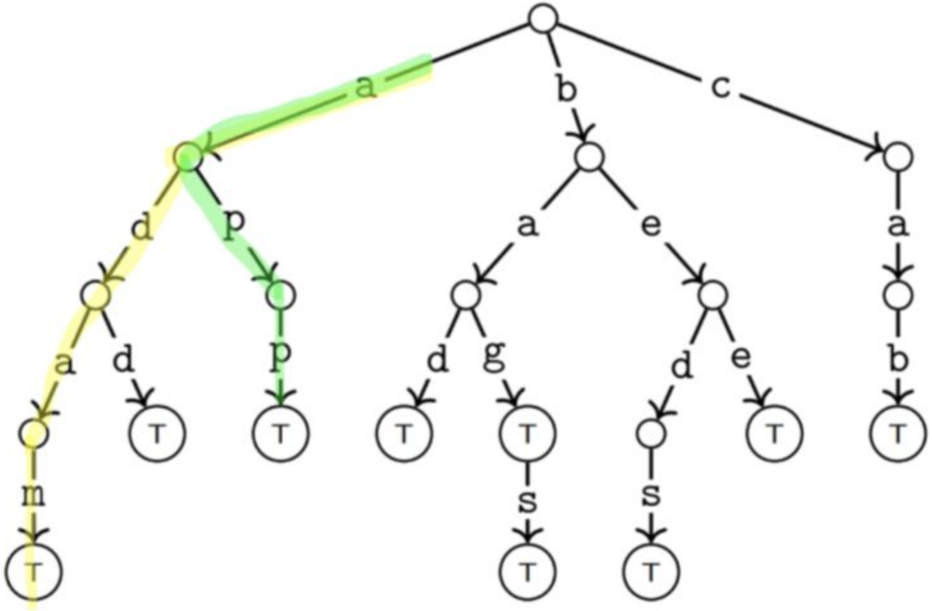
Find ("a")



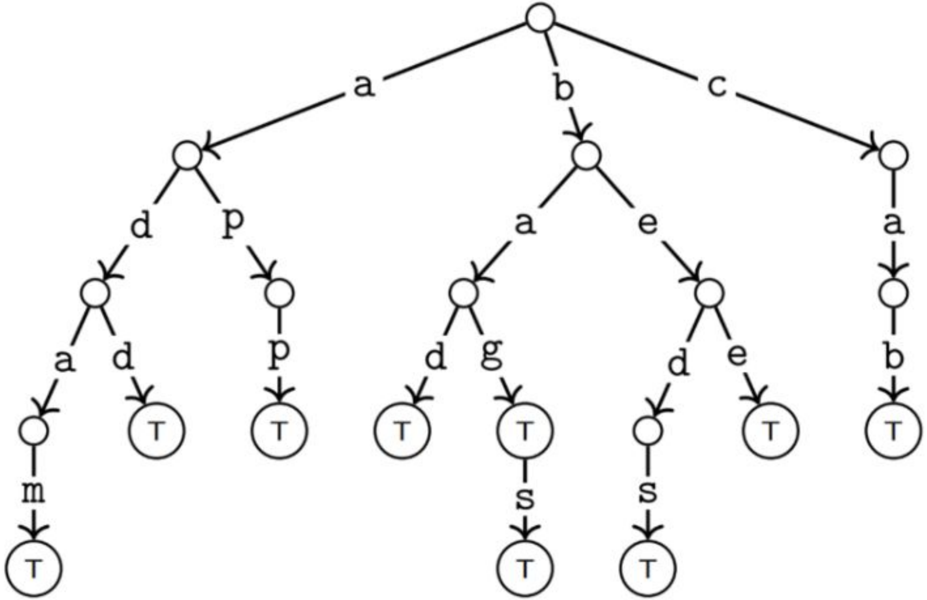
Find ("a")



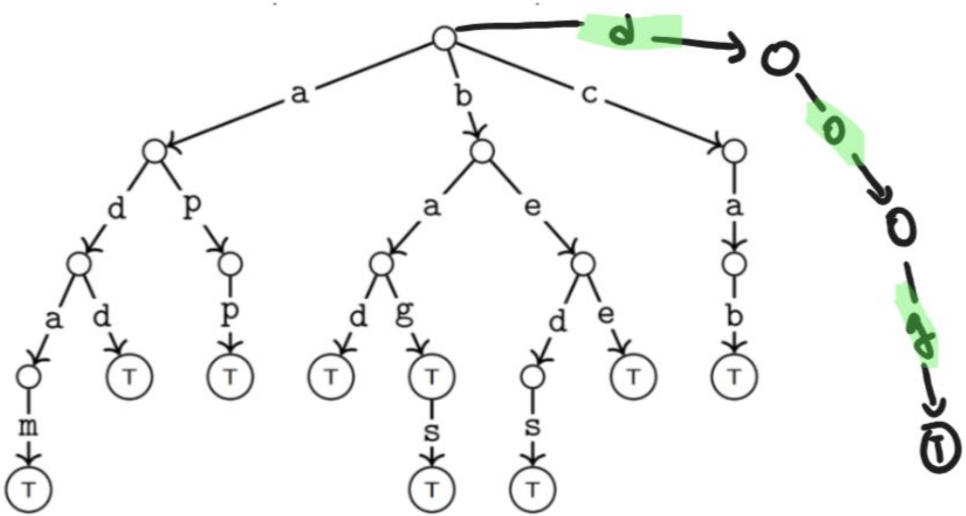



a
b
c
d
p
a
e
a
a
d
p
d
g
d
e
b
m
T
T
T
T
T
T
T
s
s
T
T

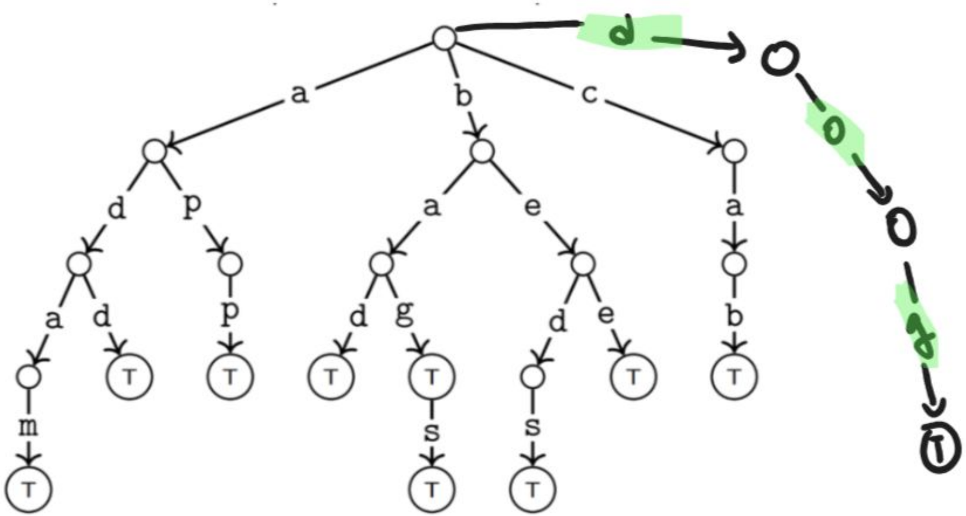
Insert(“dog”)



Insert("dog")



Insert(“dog”)

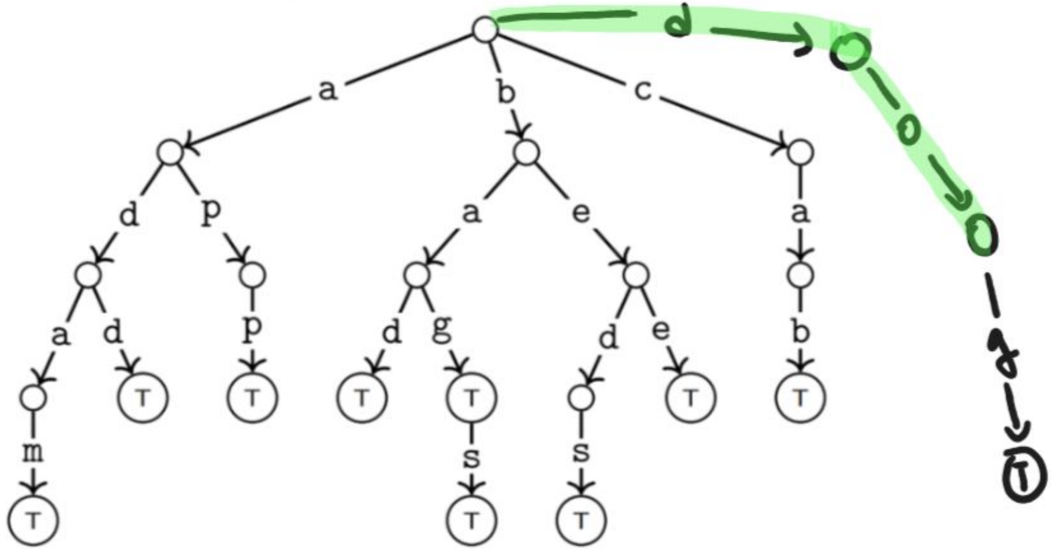


Runtime of Insert:

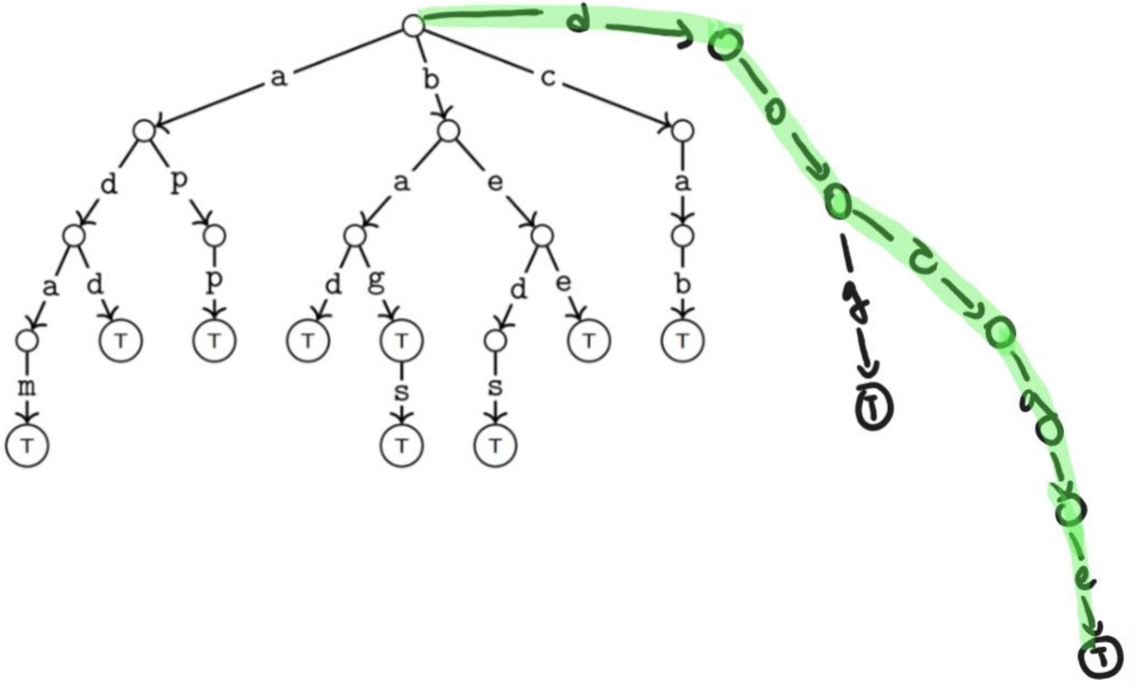
O(L)

Where L is the length of the key you’re searching for

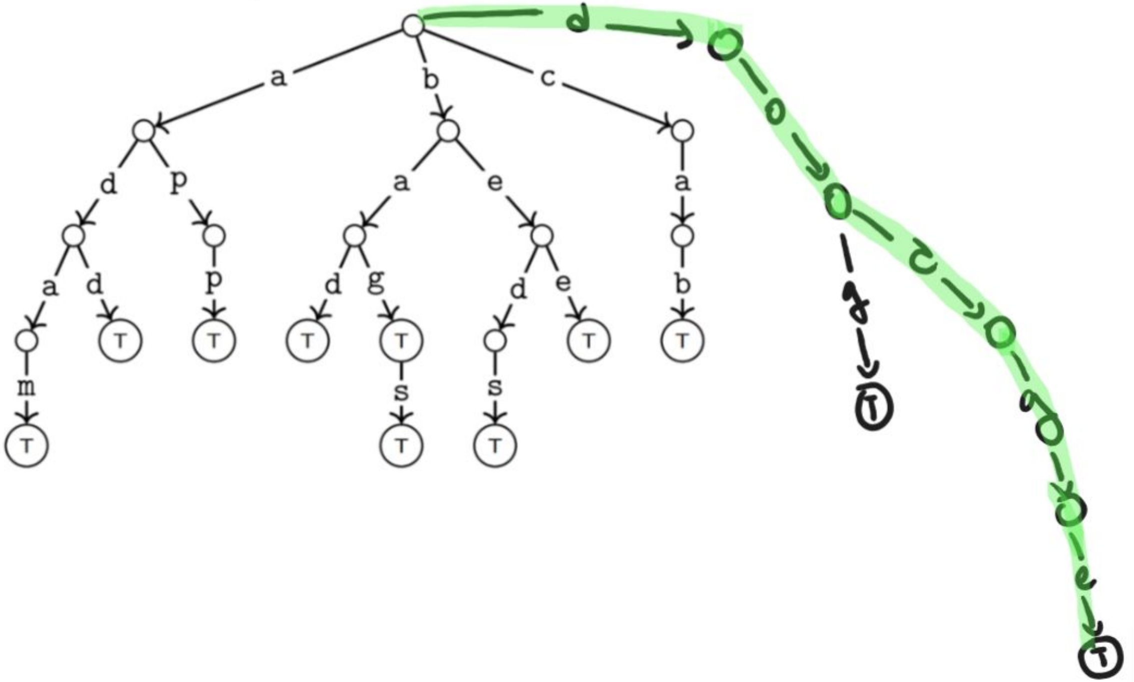
Insert("dodge")

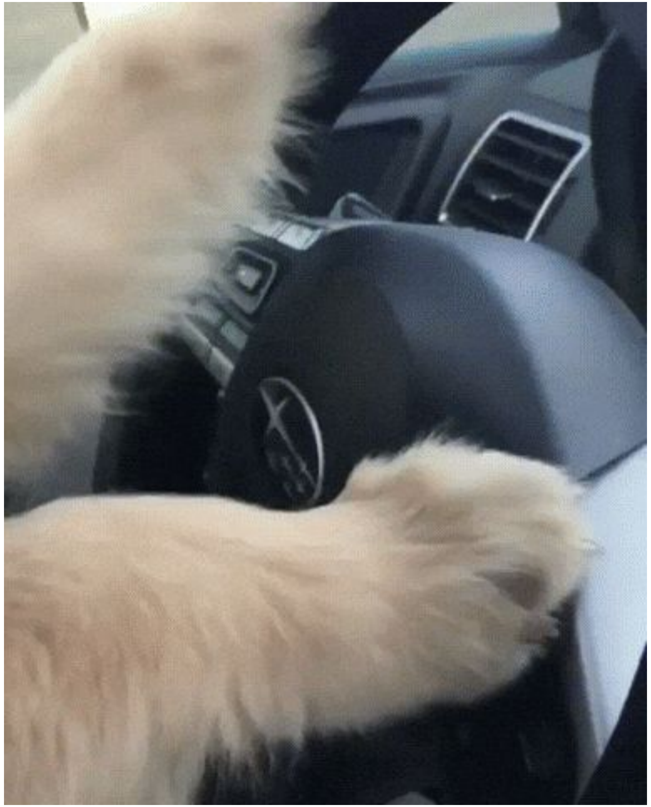


Insert(“dodge”)

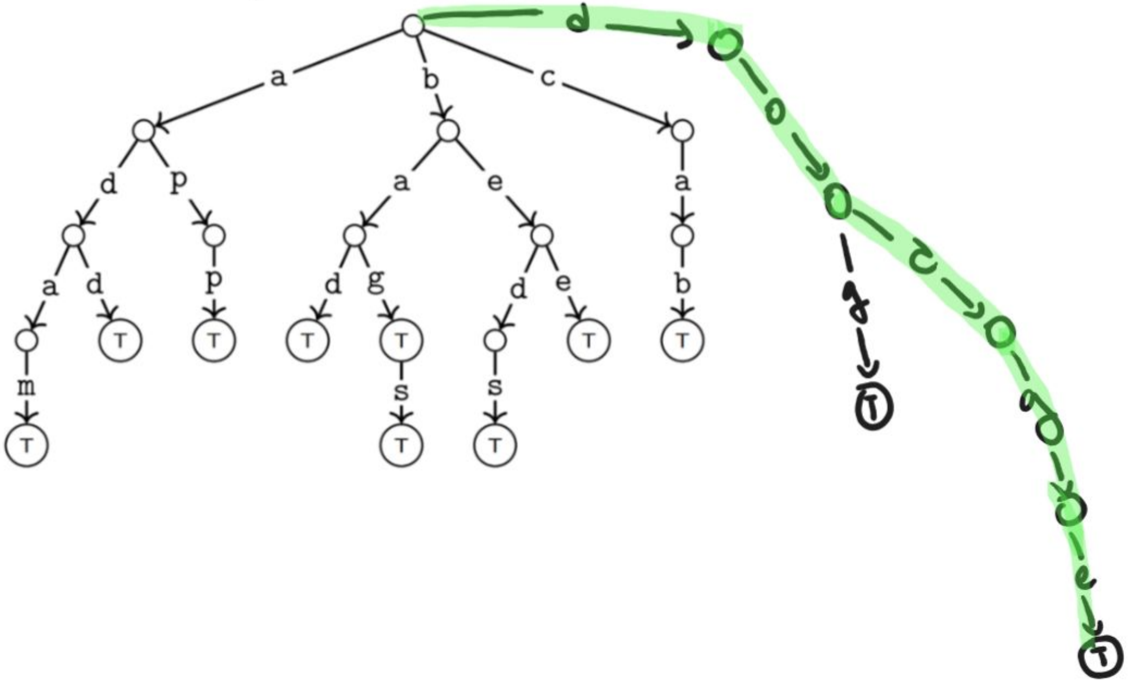


Insert("dodge")

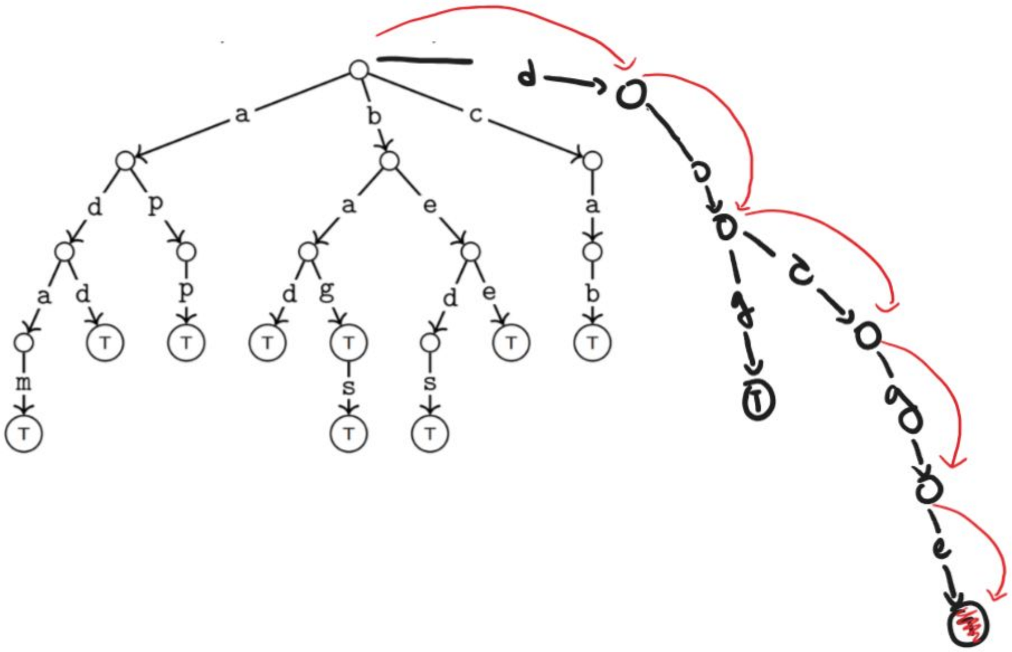




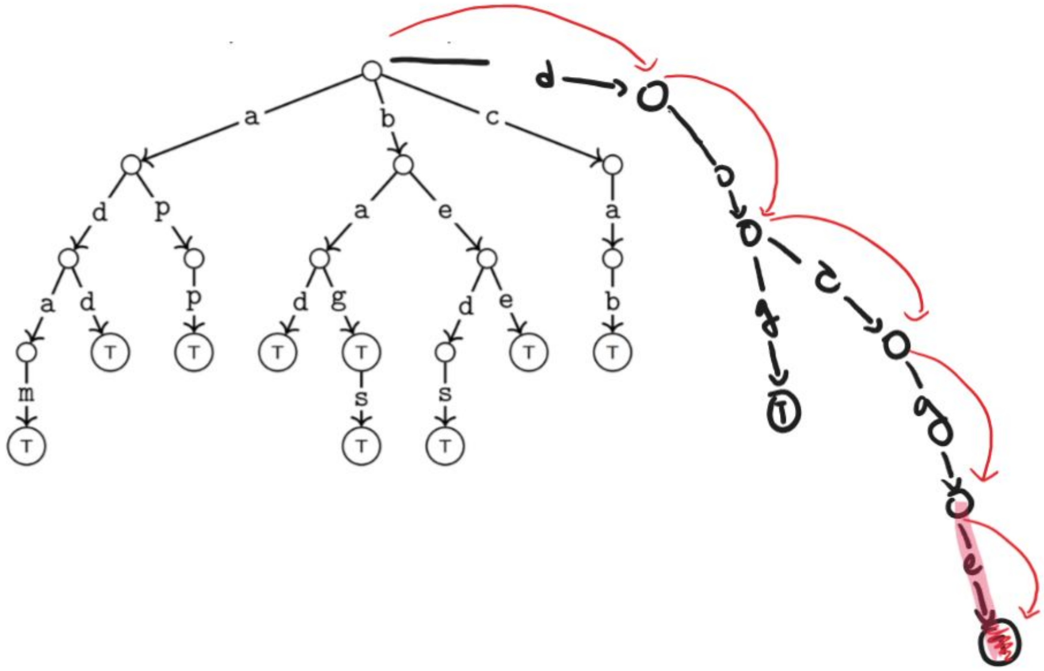
Remove("dodge")



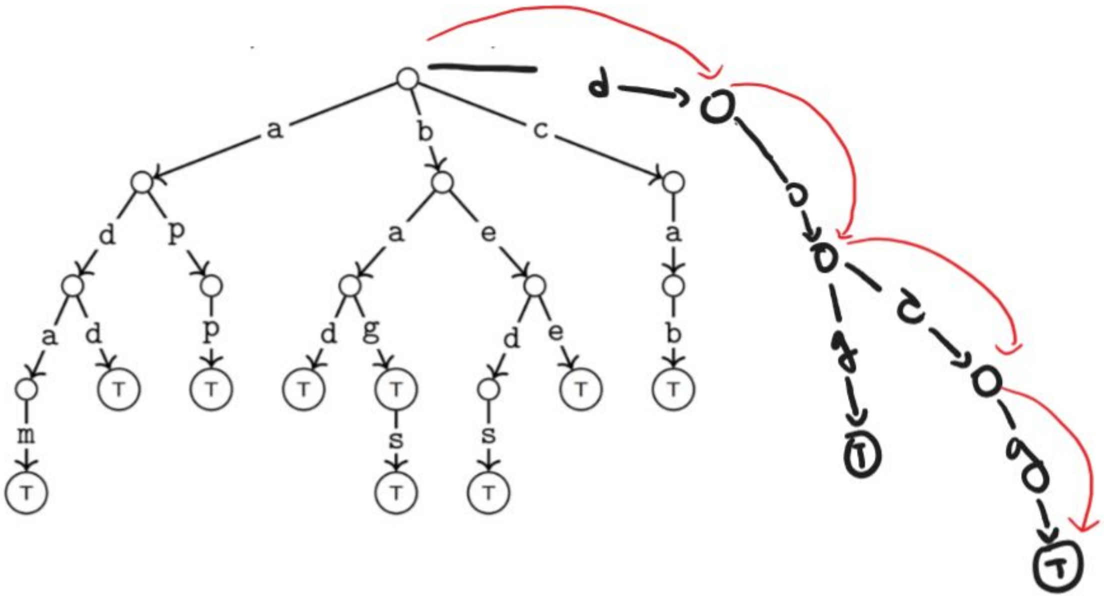
Remove("dodge")



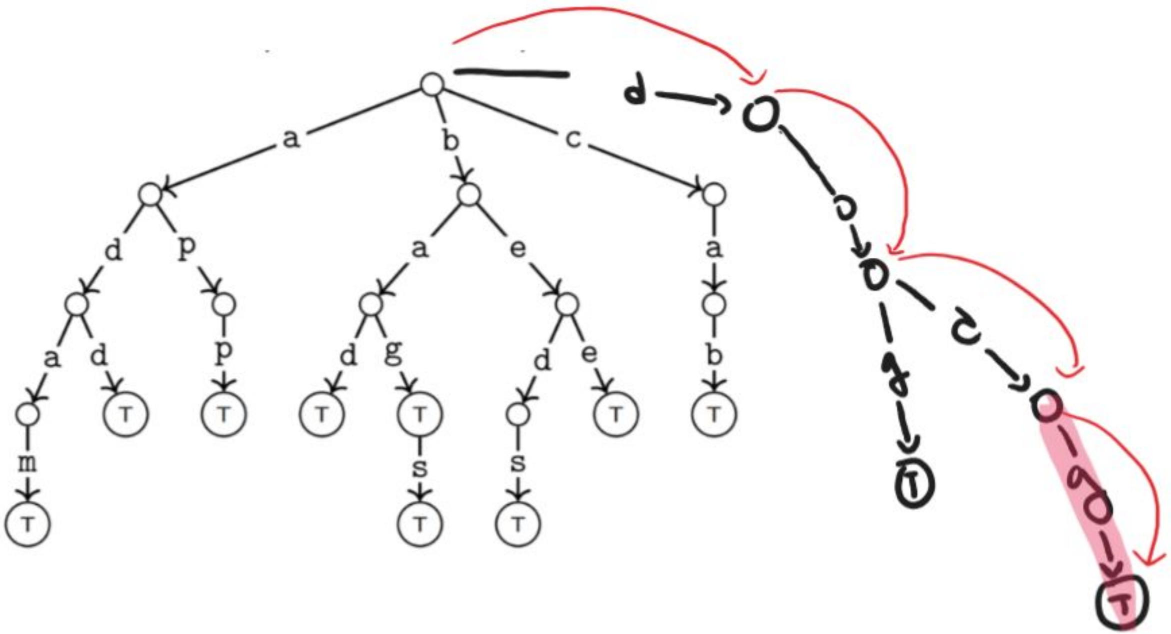
Remove("dodge")



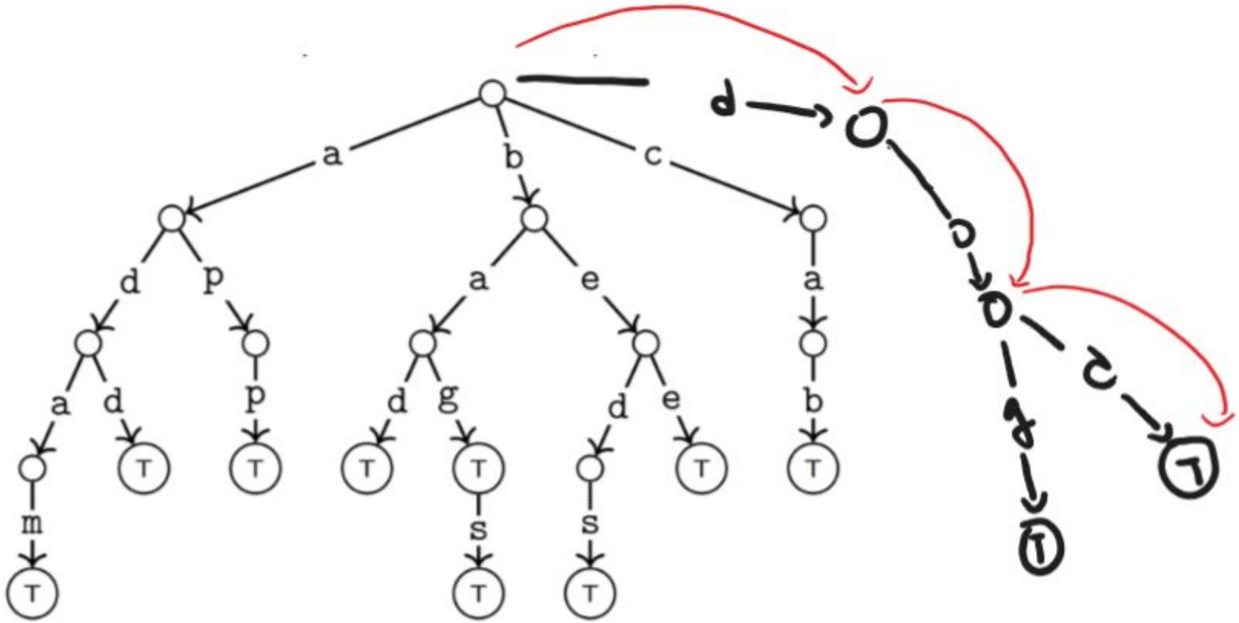
Remove("dodge")



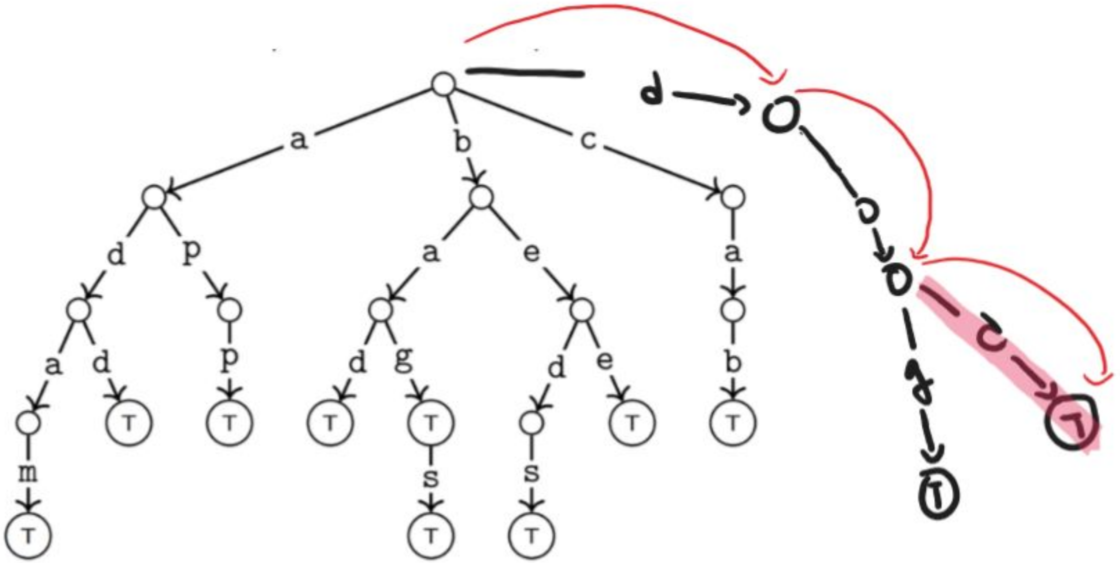
Remove("dodge")



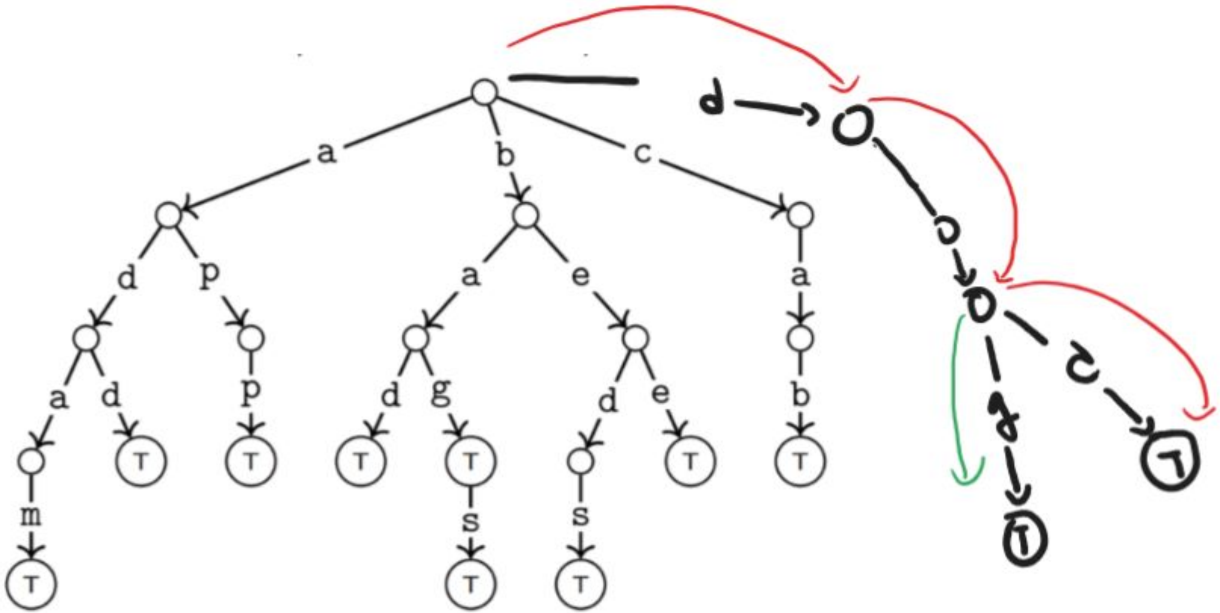
Remove("dodge")



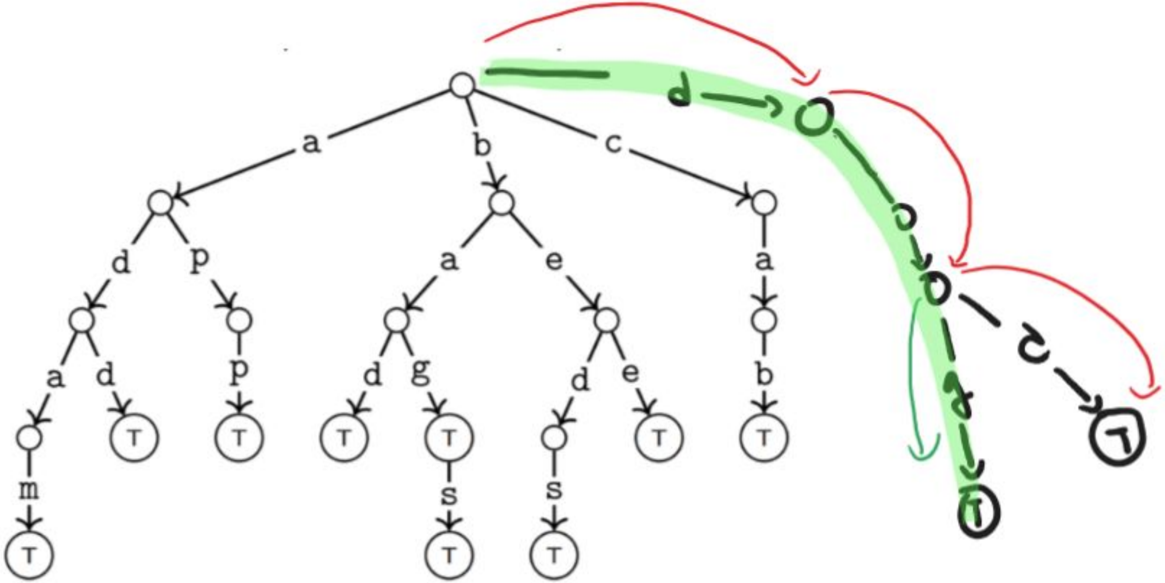
Remove("dodge")



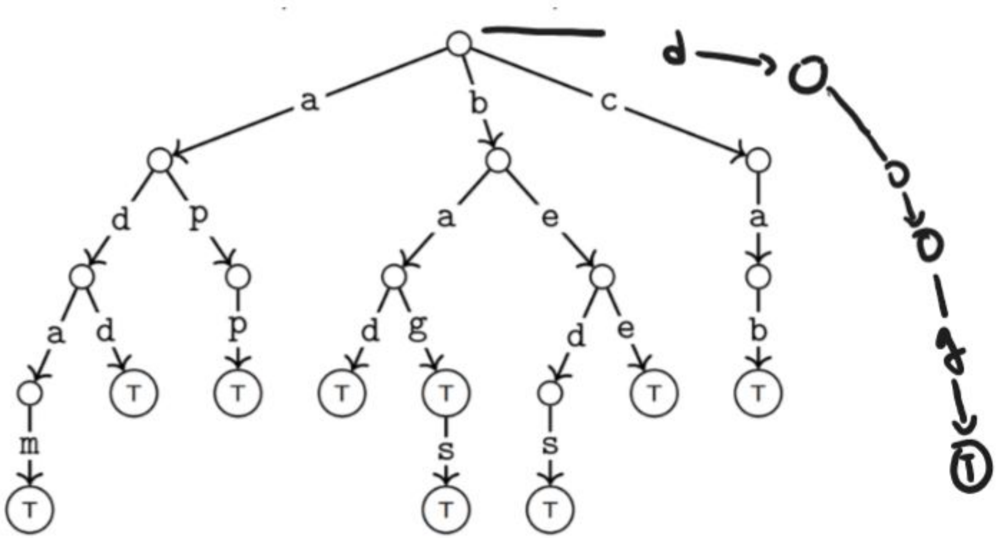
Remove(“dodge”)



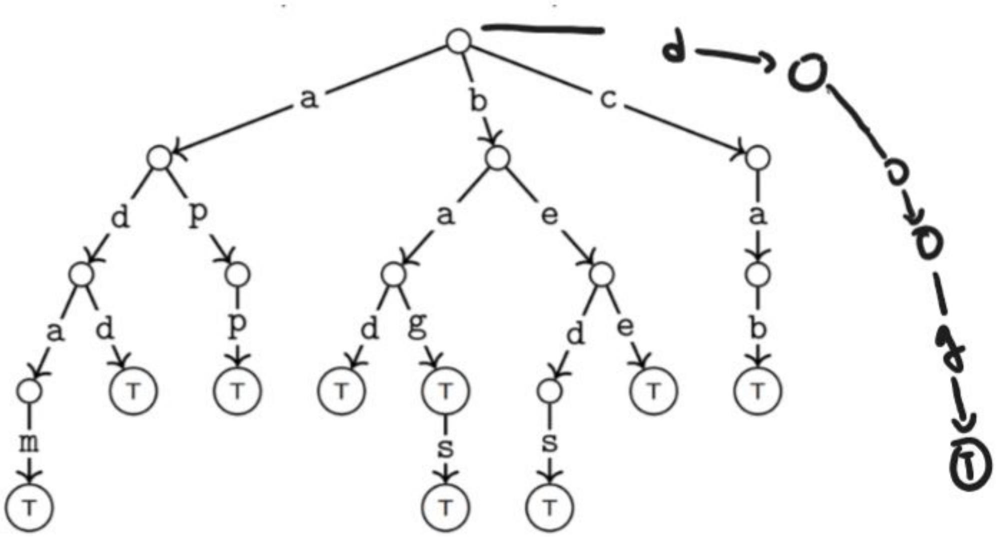
Remove("dodge")



Remove("dodge")



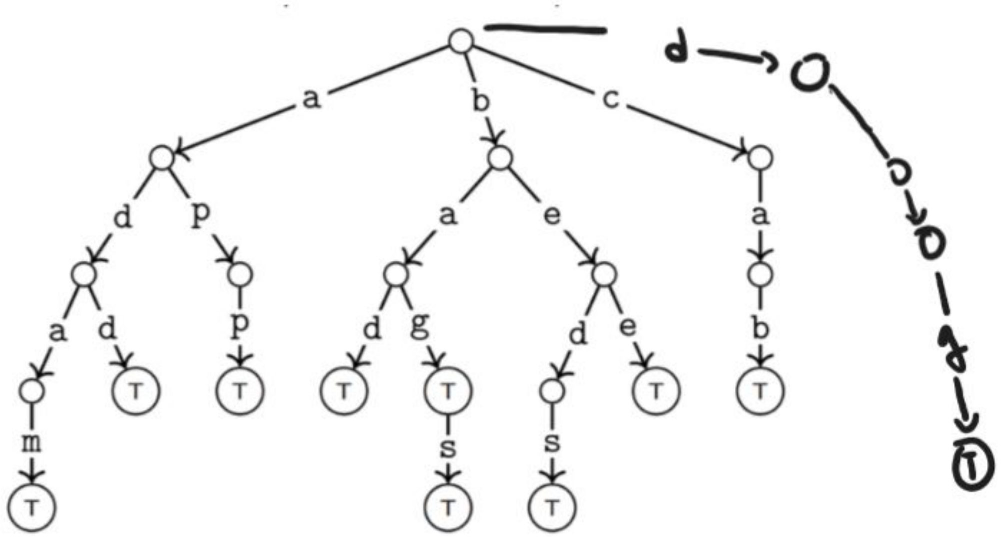
Remove("dodge")



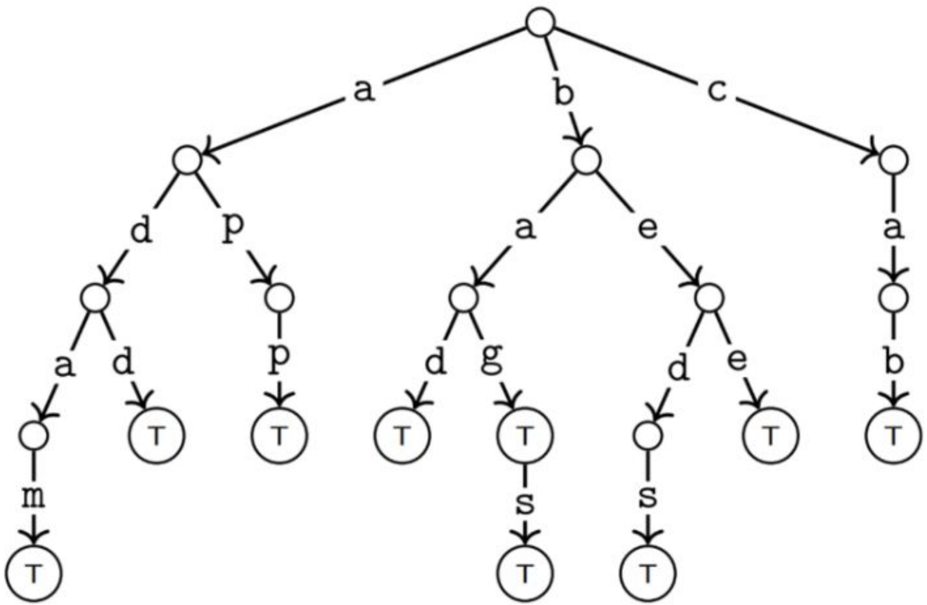
When removing:

1. Start at the root
2. Keep traversing down to bottom of the word if branch exists
3. If node we want to remove has no children remove entire node
4. Backtrack through previously traversed nodes and remove until reach a node with value, with children, or the root itself.

Remove("dodge")

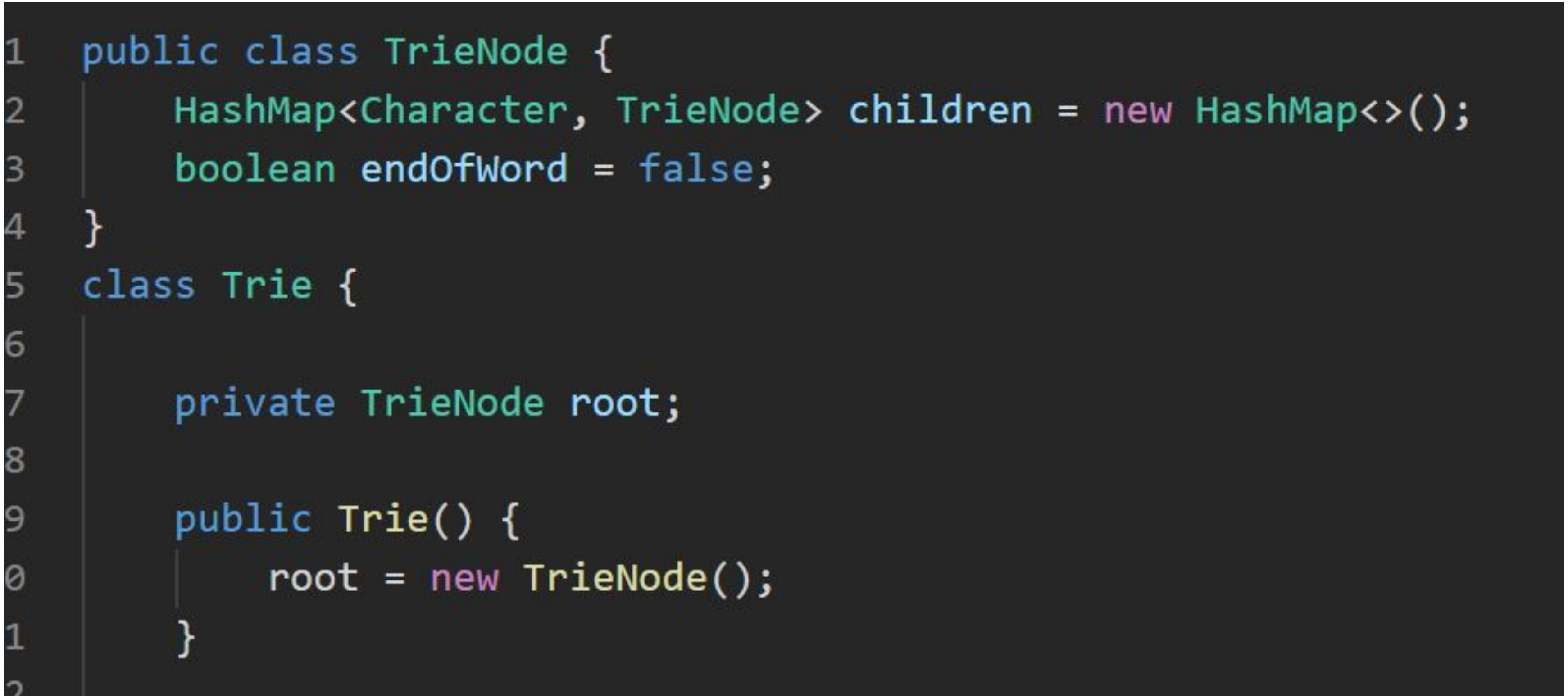


Runtime of Remove:
O(L)
Where L is the length of the key you're searching for



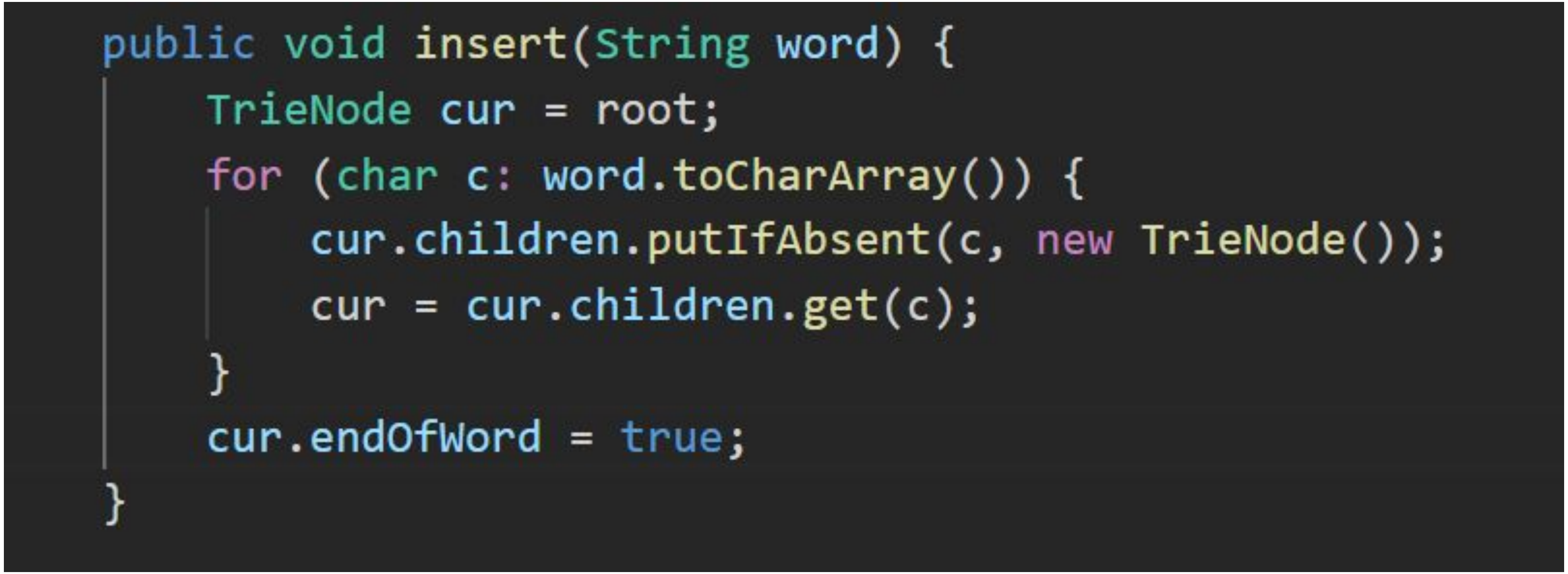
This trie represents the dictionary: `{adam, add, app, bad, bag, bags, beds, bee, cab},`

Code Overview:



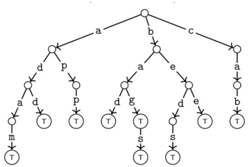
```
public class TrieNode {
    HashMap<Character, TrieNode> children = new HashMap<>();
    boolean endOfWord = false;
}
class Trie {

    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }
```

Code Overview:



```java
public void insert(String word) {
    TrieNode cur = root;
    for (char c: word.toCharArray()) {
        cur.children.putIfAbsent(c, new TrieNode());
        cur = cur.children.get(c);
    }
    cur.endOfWord = true;
}
```

What are Tries?



- The key for a node is represented by the path from the root to that node
- Nodes store the value corresponding to the key
- The nodes represent all the items that could possibly come afterwards
- Terminal Nodes marks the end of the word

Pros of Tries

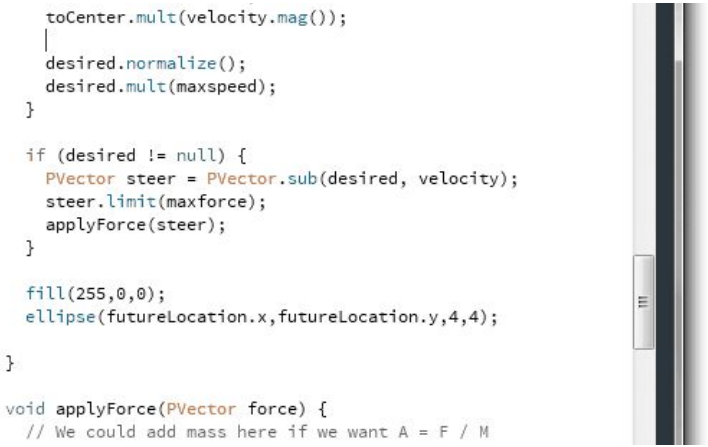
- Fast lookup, insert, delete (O(L))
  - Very predictable!
- No Hashing collisions,
- No rebalancing
- Ordering comes for free (lexicographic order)
- Flexible alphabet

Cons of Tries

- High memory overhead
    - Each node has Pointers to many children
- Inefficient for small datasets
- Inefficient for long unique keys
- Not balanced

Uses of Tries

- Autocomplete/Spell Checkers/Search Suggestions:
  - Lookup prefix “ad” → reach a subtree
  - DFS/BFS from that node
  - Hashtables/BSTs can’t do this efficiently
- File System Path indexing
  - C:/users/hana/downloads
- Word Games
  - Scrabble, or a super mean wordle



```
    toCenter.mult(velocity.mag());

    desired.normalize();
    desired.mult(maxspeed);
  }

  if (desired != null) {
    PVector steer = PVector.sub(desired, velocity);
    steer.limit(maxforce);
    applyForce(steer);
  }

  fill(255,0,0);
  ellipse(futureLocation.x,futureLocation.y,4,4);

}

void applyForce(PVector force) {
  // We could add mass here if we want A = F / M
```

LeetCode Trie Problems:

- Implement Trie (#208)
- Design Add and Search Words Data Structure (#211)
- Word Break (#139)
- Longest Common Prefix (#14)

Thanks!