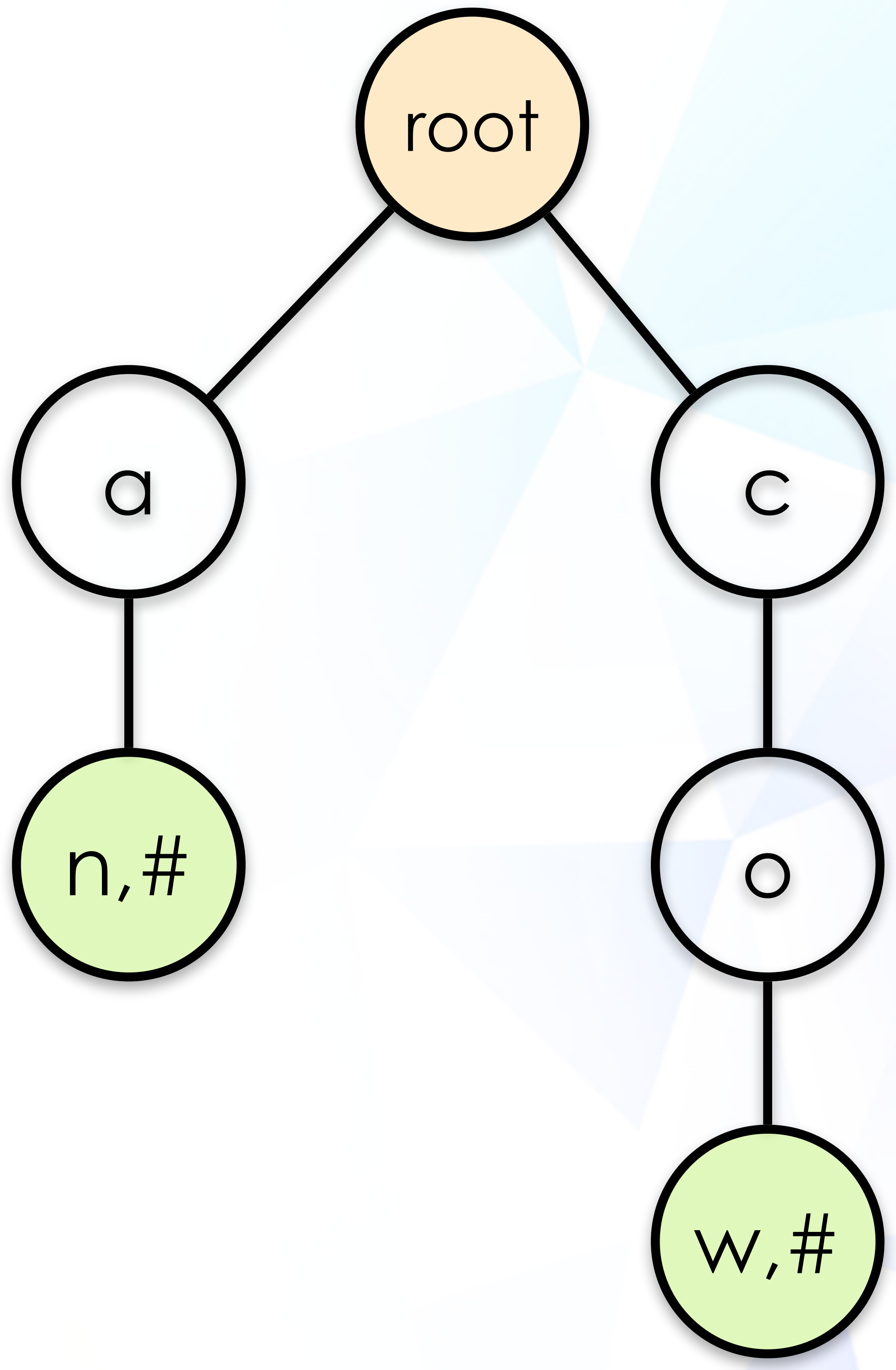
Trie

Tan Cher Wah (isstcw@nus.edu.sg)

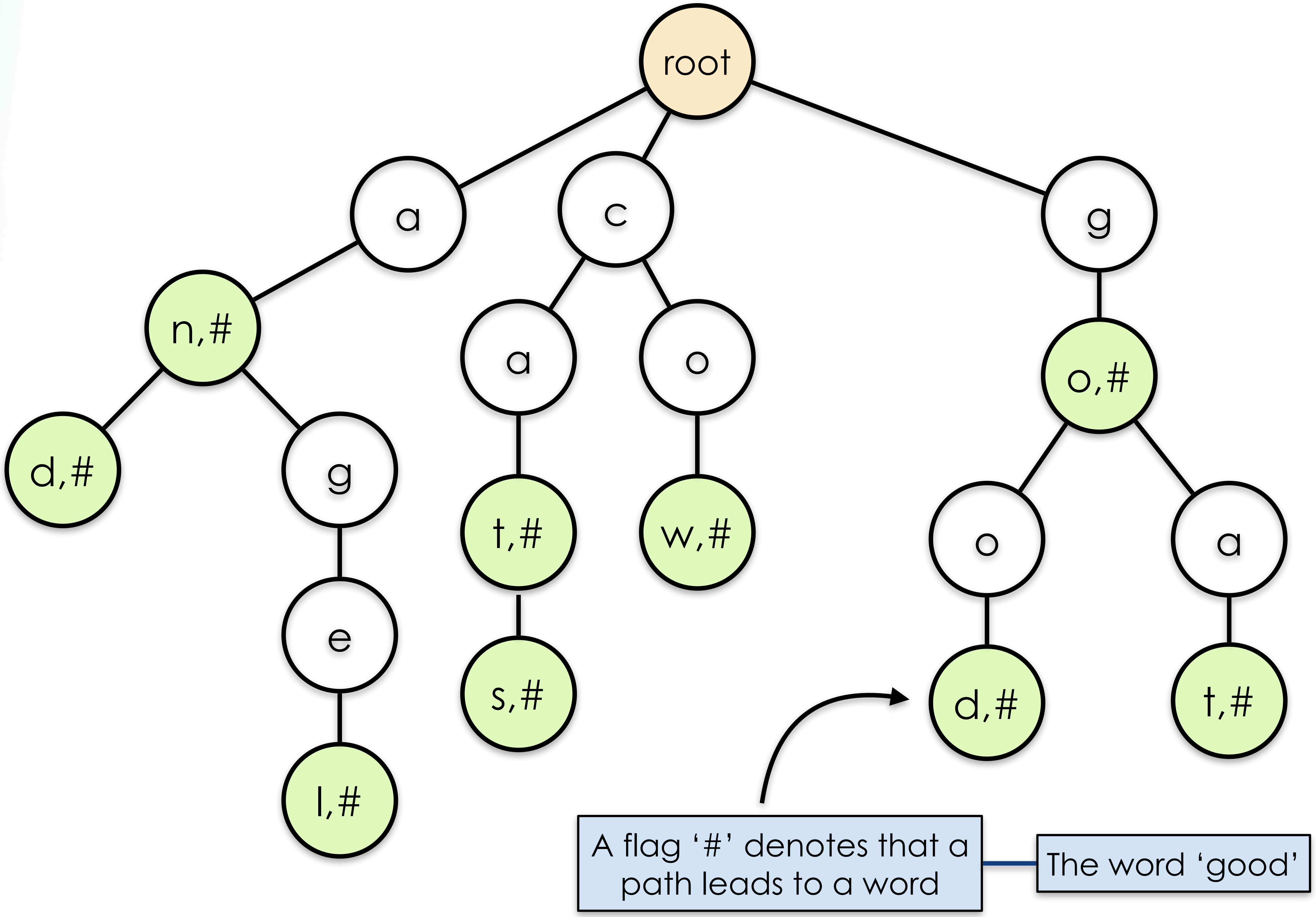
Trie

- A Trie is a tree data structure where each node contains part of a **prefix** to a string
- The **Root** node represents an **empty string**
- Each node stores a **character**, along with a **flag** (to denote if the path from Root to itself yields an actual word)
- Each node contains a **dictionary**, with **entries** pointing to **other Trie nodes**



Trie

This Trie data structure stores every word defined in the green box



Words in Trie
- an
- and
- angel
- cat
- cats
- cow
- go
- good
- goat

Common Operations

- Insert a string
- Search for a string
- Delete a string

Applications

- Spell Checker - Allow text-editing software to quickly look up if a given word is in a dictionary
- Word Suggestion - Suggest similar words that have the same prefix in response to a misspelled word
- Auto Completion - Provide a list of possible words or phrases to finish a given query

A Trie Node

A Node implementation for a Trie data structure

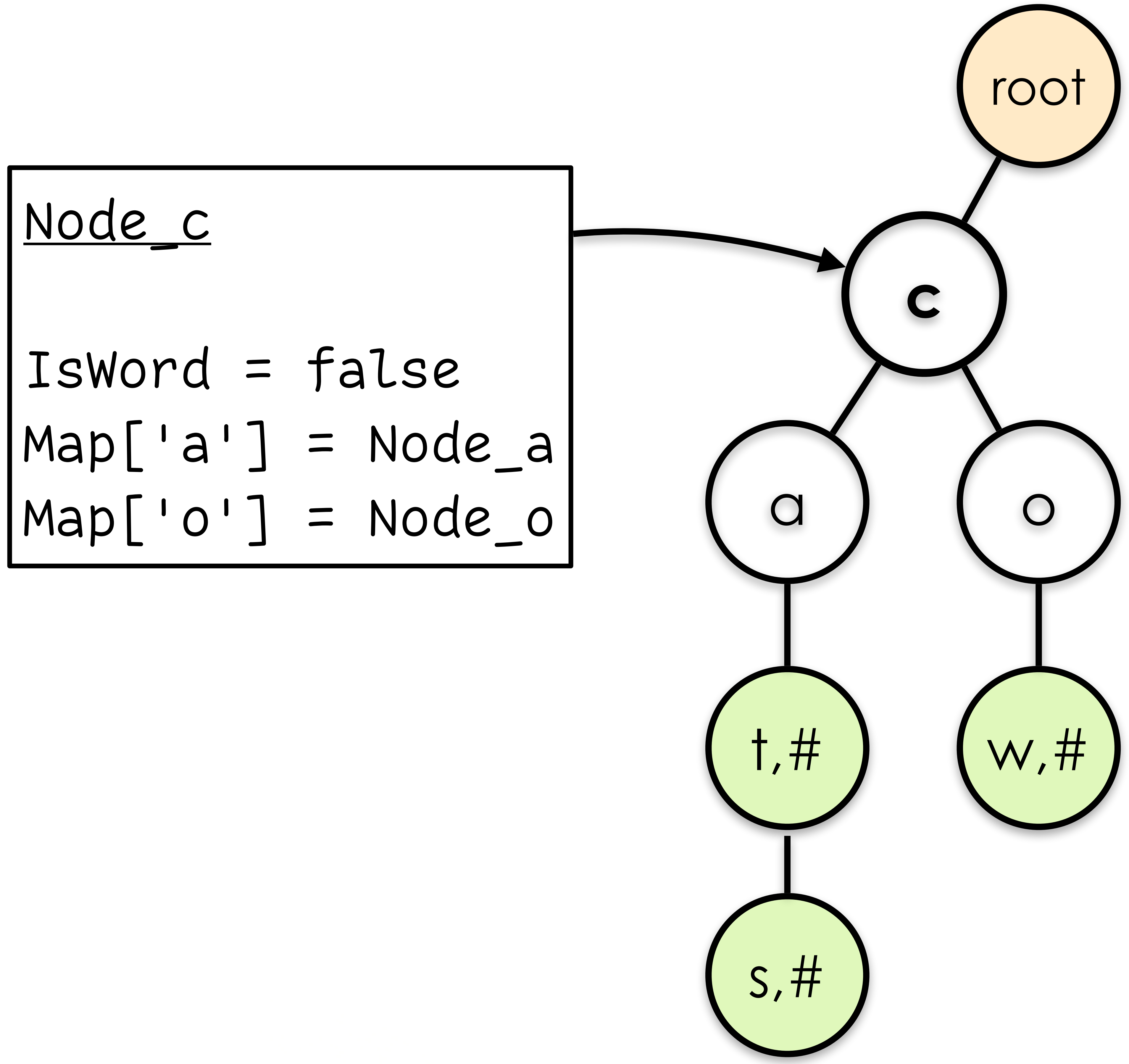
Using a boolean flag to denote if a path leads to a word

```
public class Node
{
    public bool IsWord { get; set; }
    public Dictionary<char, Node> CharMap { get; set; }

    public Node()
    {
        IsWord = false;
        CharMap = new Dictionary<char, Node>();
    }
}
```

Using a Dictionary to track entries that extends current prefix

A Trie Node

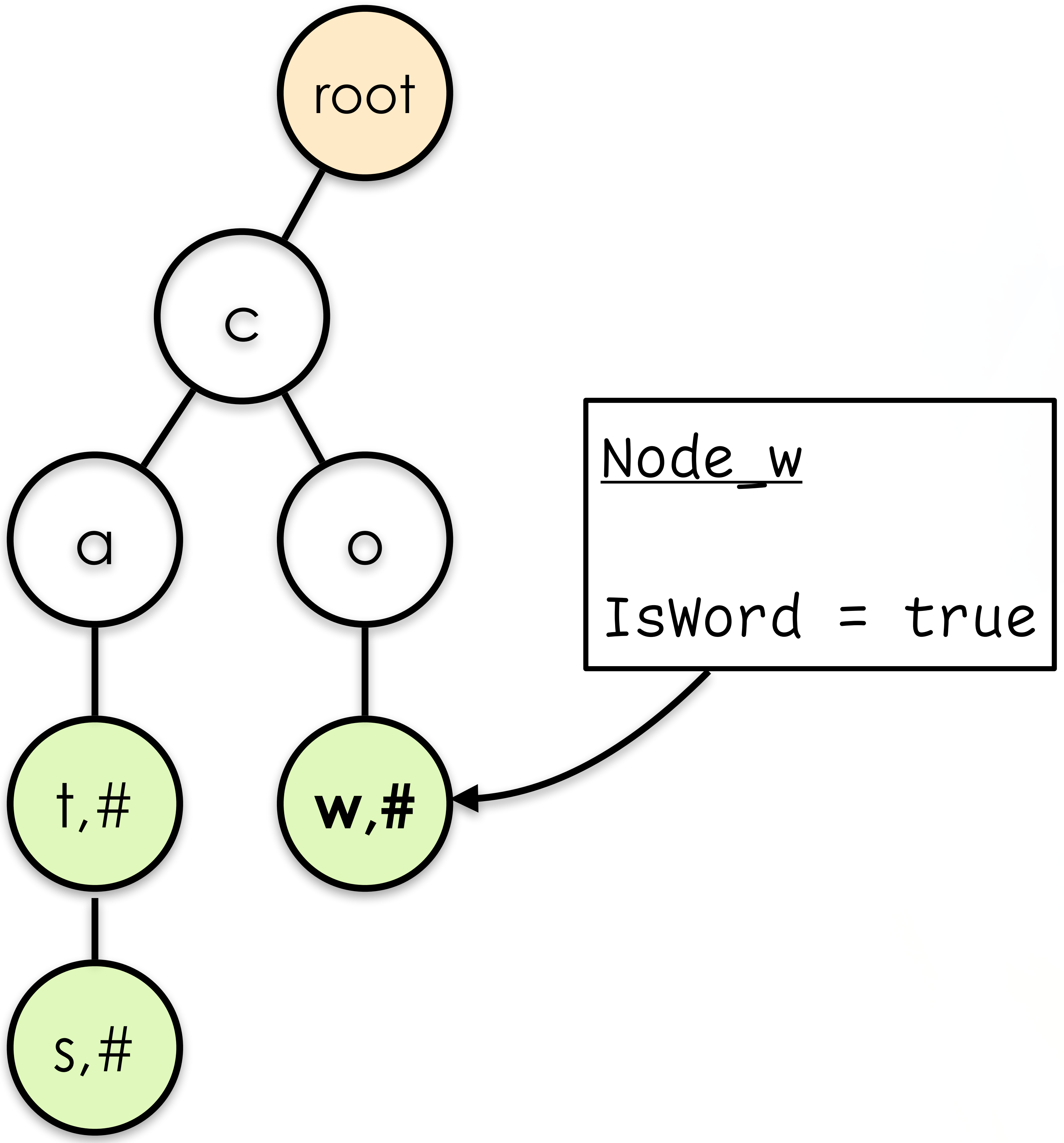
This is how Node_c would look like when the program runs



```
Node_c

IsWord = false
Map['a'] = Node_a
Map['o'] = Node_o
```

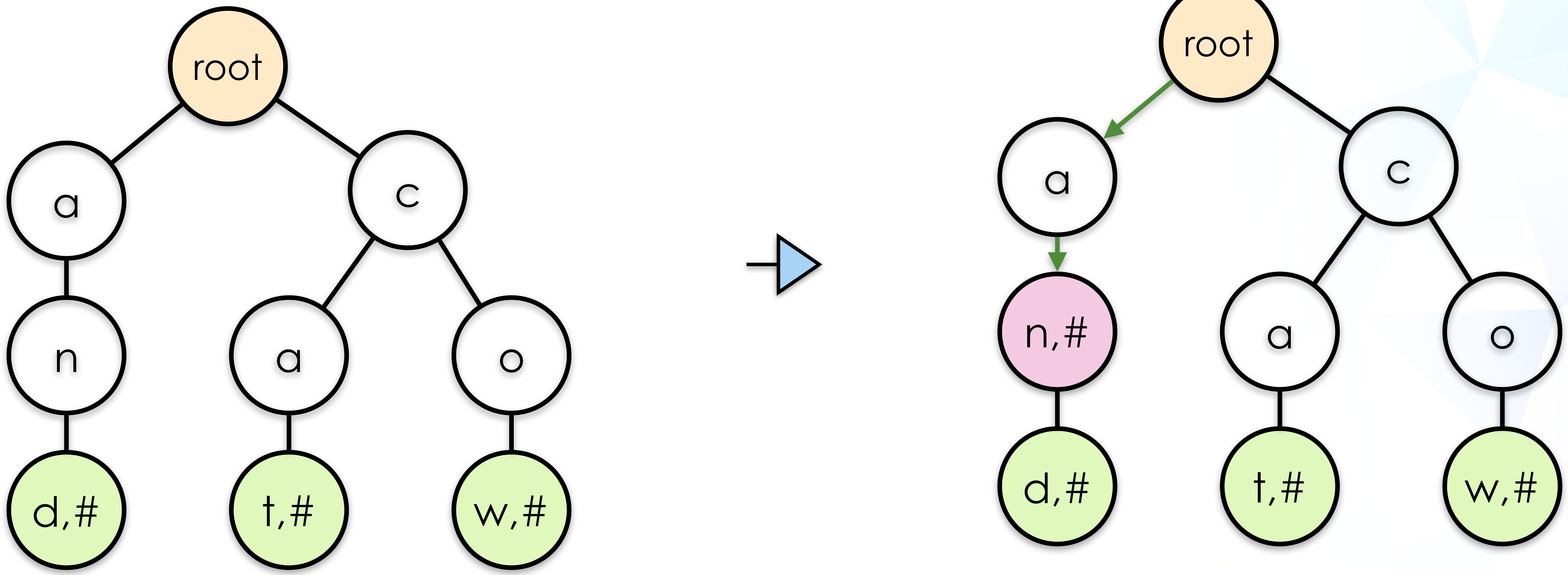
A Trie Node

This is how Node_w would look like when the program runs



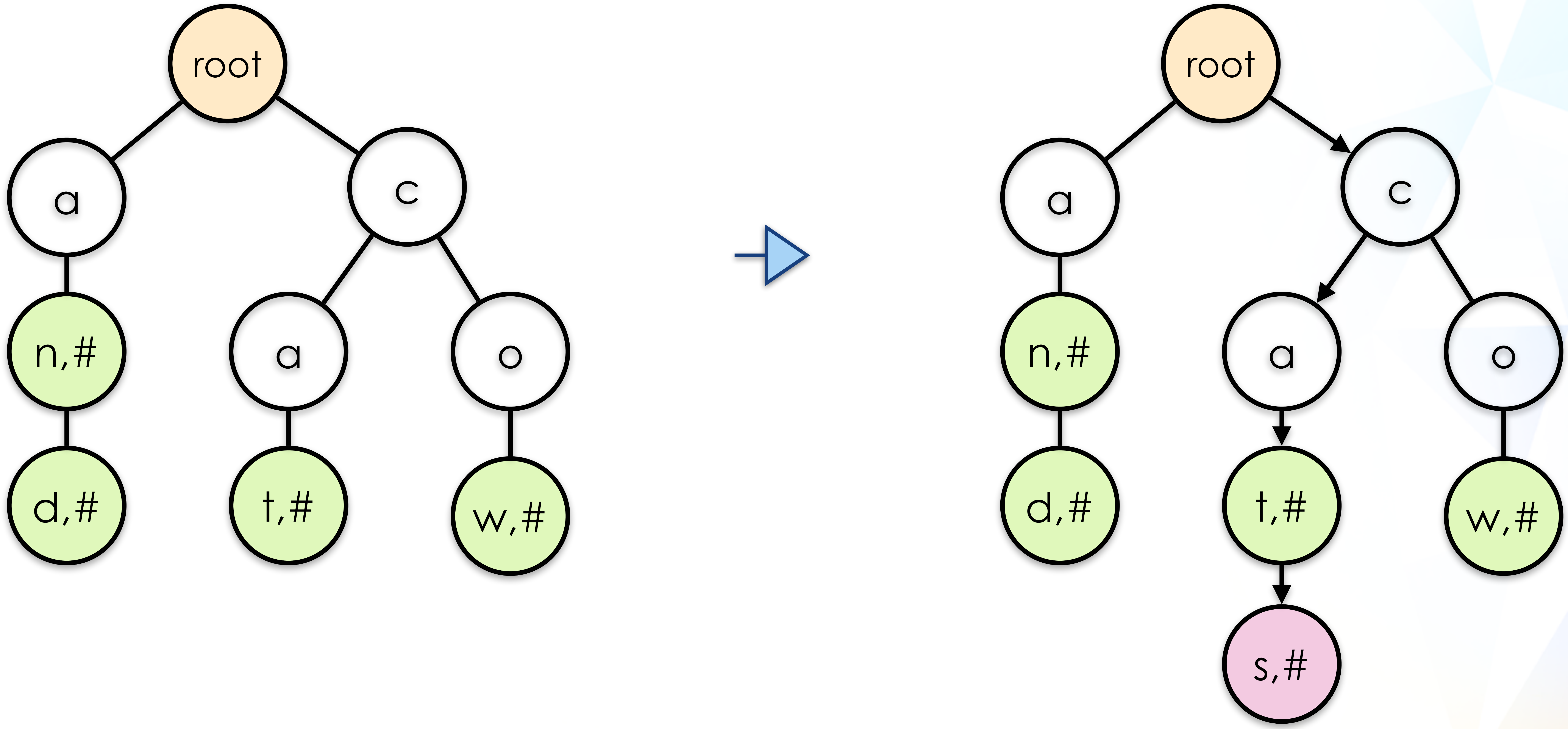
Insert

Given the left Trie, insert the word 'an'



Insert

Given the Trie on the left, insert the word 'cats'



Insert

A C# implementation for inserting a new word in a Trie data structure

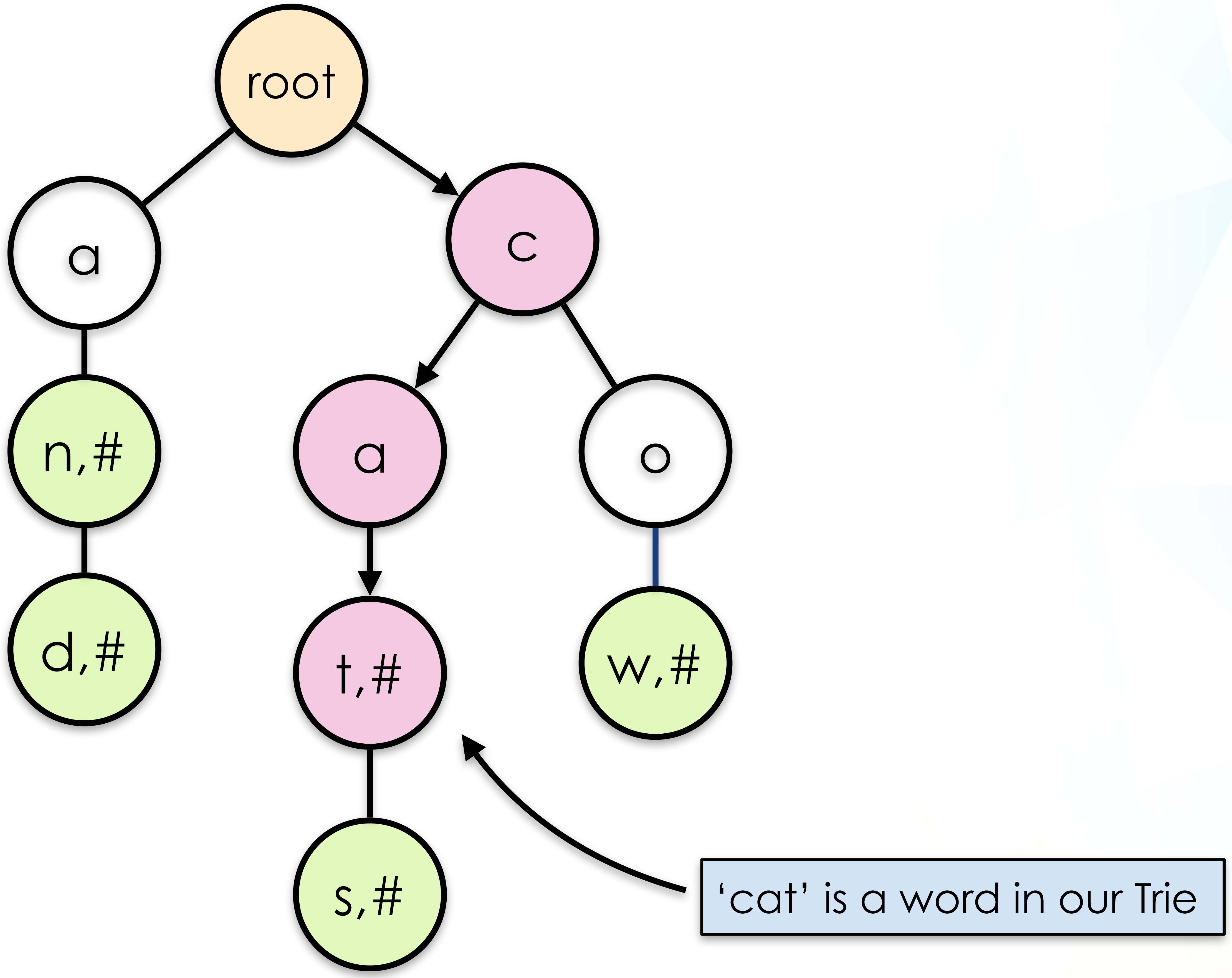
```
bool Insert(string word, Node root)
{
    Node curr = root;

    foreach (char ch in word) {
        if (! curr.CharMap.ContainsKey(ch)) {
            curr.CharMap[ch] = new Node();
        }
        curr = curr.CharMap[ch];
    }

    curr.IsWord = true;
    return true;
}
```

Any missing characters in between will be created

Mark the final node as ending as a word

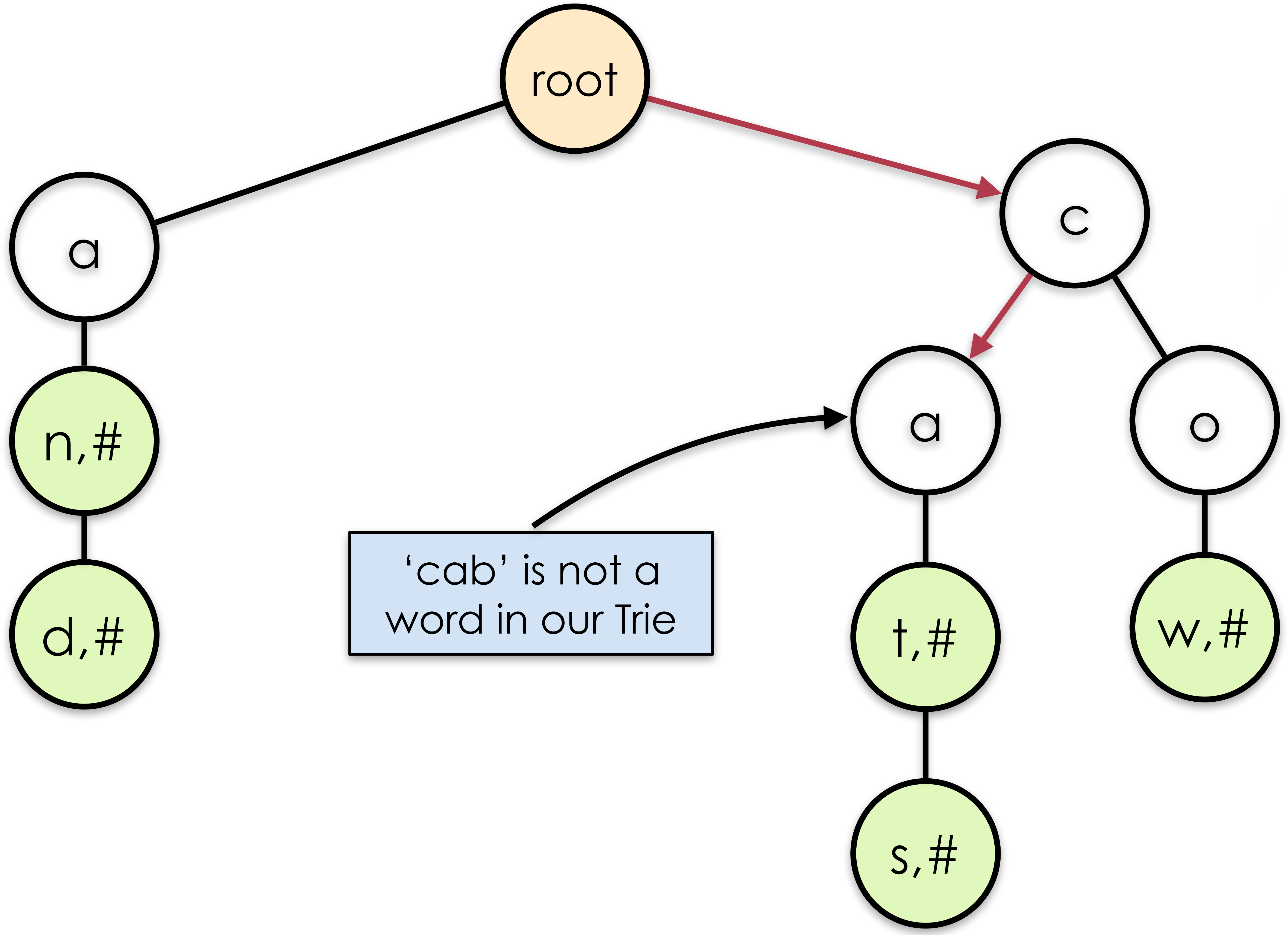
Search

Search for the word ‘cat’ in the given Trie



Search

Search for the word 'cab' in the given Trie



'cab' is not a word in our Trie

Search

A C# implementation for searching a word in a Trie data structure

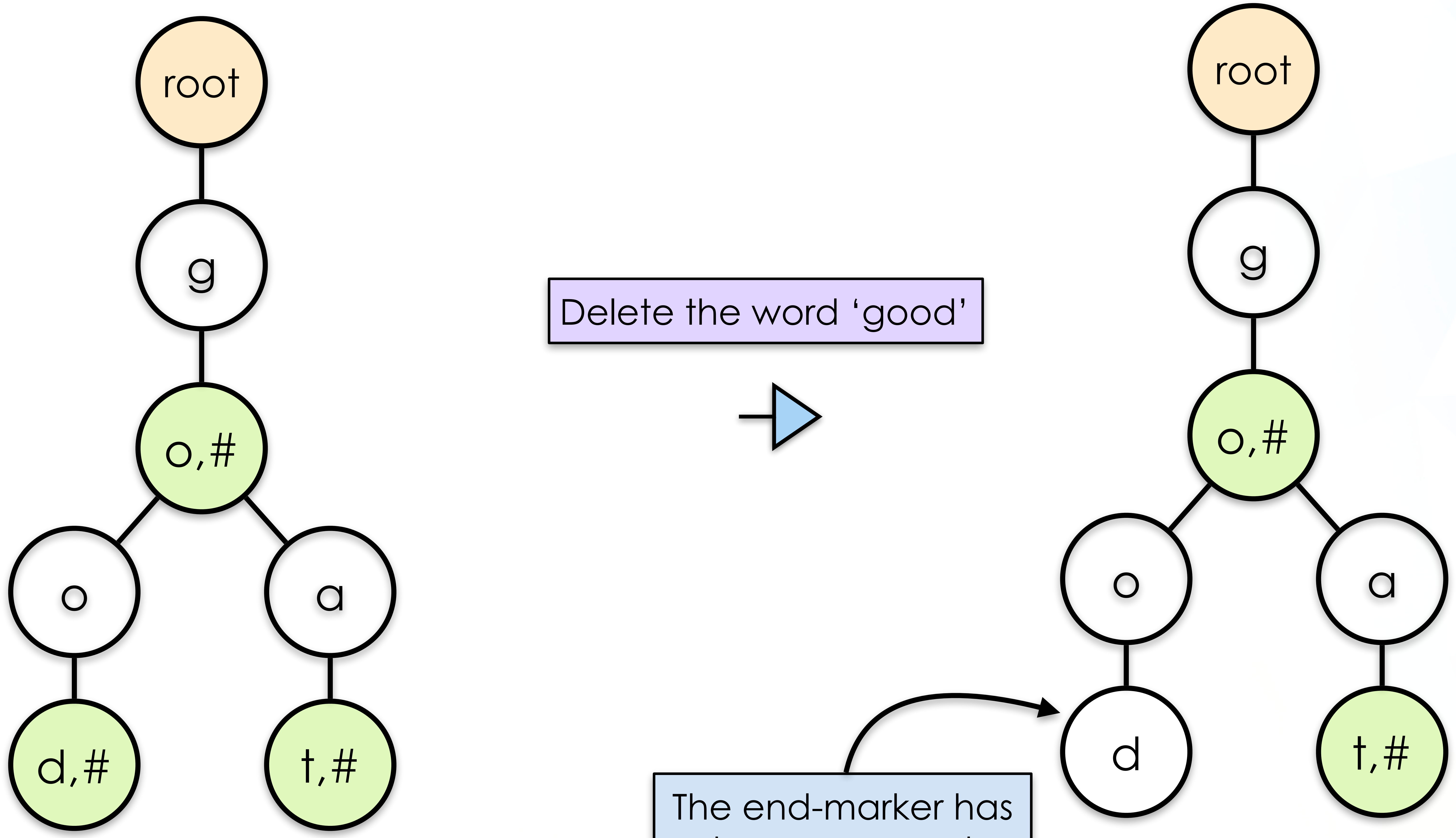
```
bool Search(string word, Node root)
{
    Node curr = root;

    foreach (char ch in word) {
        if (! curr.CharMap.ContainsKey(ch)) {
            return false;
        }

        curr = curr.CharMap[ch];
    }

    return curr.IsWord;
}
```

No path to the search word

Return the result if the node ends as a word

Delete

Scenario 1: Given the Trie on the left, delete a word without pruning



The end-marker has been removed

Delete

Scenario 1: A C# implementation for deleting a word without pruning

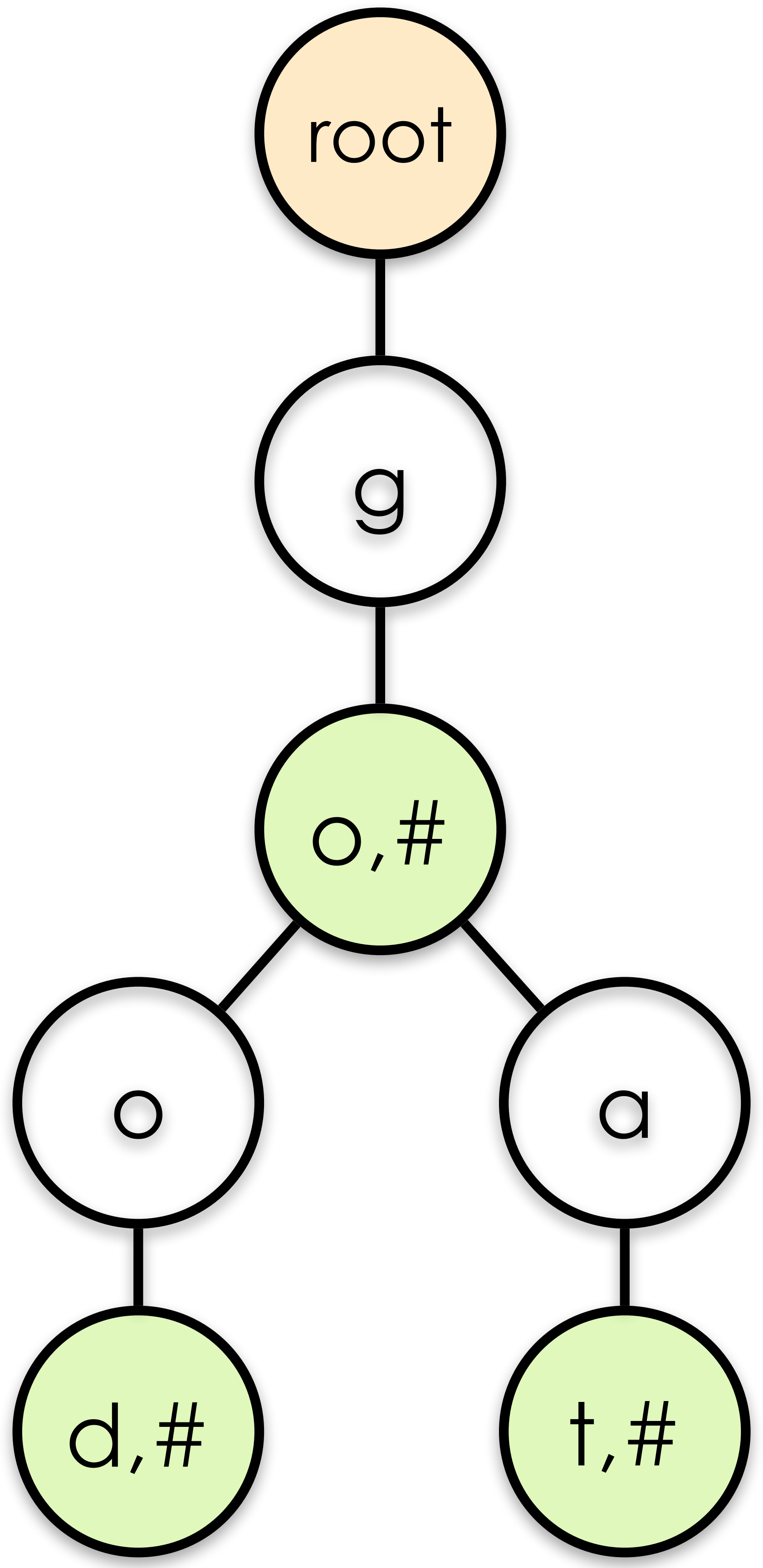
```
public bool Delete(string word, Node root)
{
    Node curr = root;

    foreach (char ch in word) {
        if (!curr.CharMap.ContainsKey(ch)) {
            return false;
        }
        curr = curr.CharMap[ch];
    }

    curr.IsWord = false;
    return true;
}
```

Follow the trail to find the word to delete

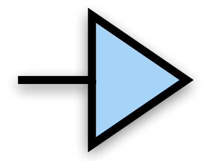
Mark the node as not ending as a word

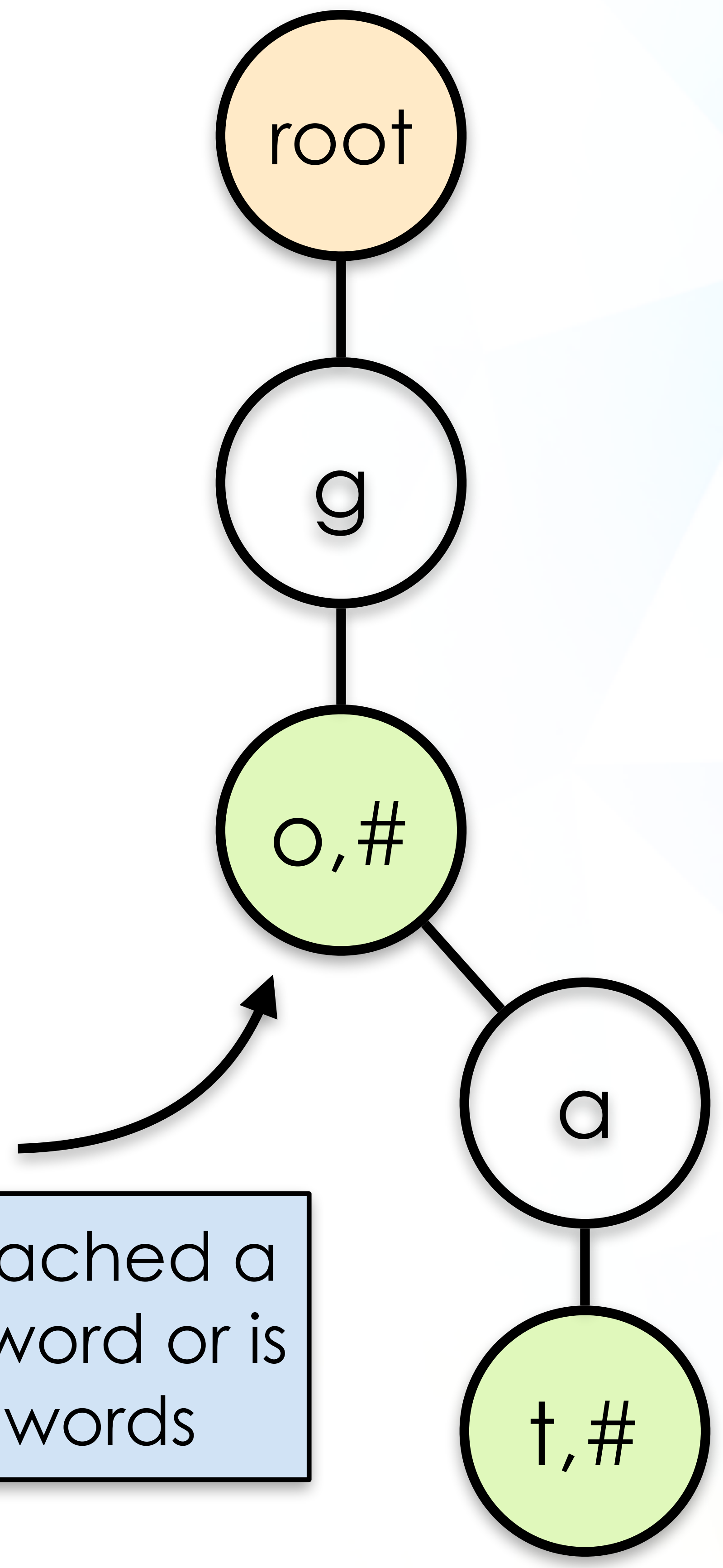
Delete

Scenario 2: Given the Trie on the left, delete a word with pruning



Delete the word 'good'





Remove until we reached a node that is a end-word or is a prefix for other words

Delete

```
public bool PurgeDelete(Node node, string word)
{
    if (word.Length == 0) {
        if (node.IsWord) {
            node.IsWord = false;
            return true;
        }
        return false;
    }

    bool status = false;

    if (node.CharMap.ContainsKey(word[0])) {
        Node child = node.CharMap[word[0]];
        if (status = PurgeDelete(child, word.Substring(1))) {
            if (!child.IsWord && child.CharMap.Count == 0) {
                node.CharMap.Remove(word[0]);
            }
        }
    }
    return status;
}
```

Mark the node as not ending as a word

Scenario 2
C# implementation for deleting with pruning

Follow the trail to find the word to delete

Performs Pruning

THE END