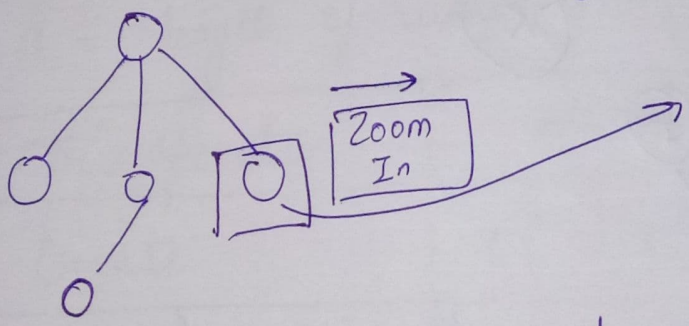
# Trie

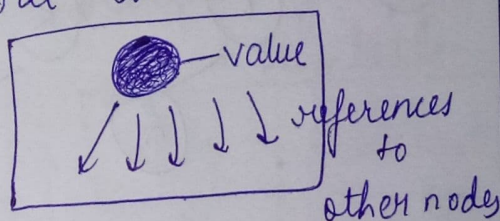
a trie is basically a tree-like data structure where nodes of a tree store the ~~entire~~ alphabet and words can be traversed by traversing down.

→ Each trie has an empty root node, with links to other nodes

→ There are 26 alphabets so total number of child nodes would be 26.



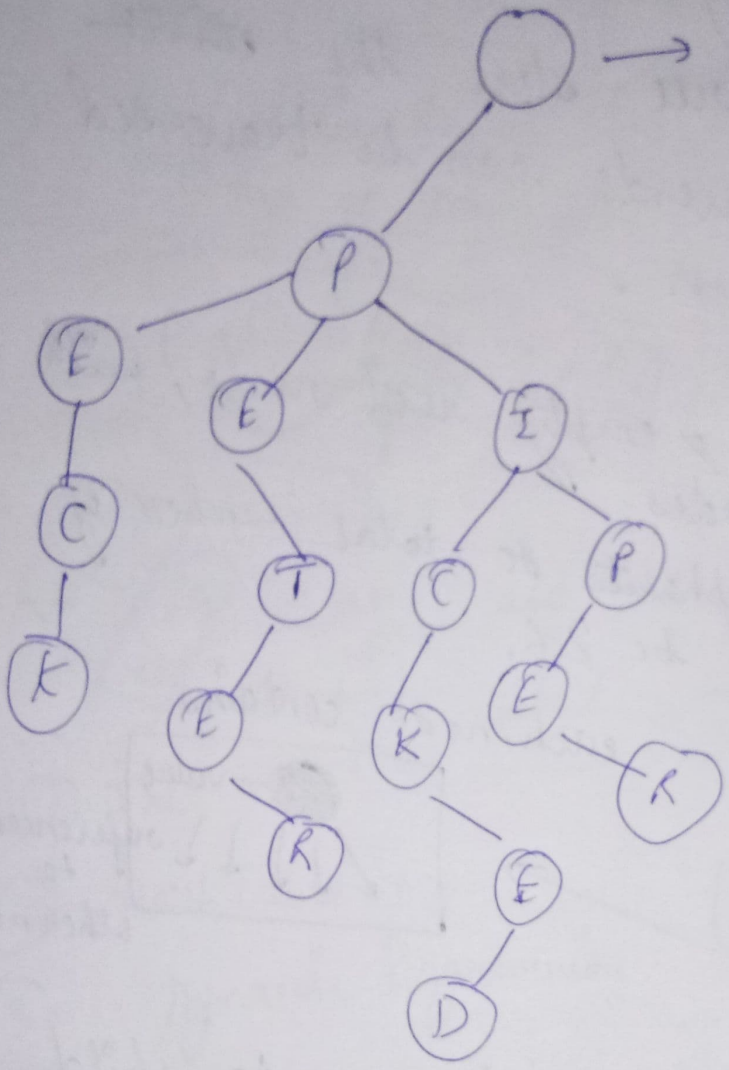

each node contains




→ Value can be null and references to child nodes also might be null.

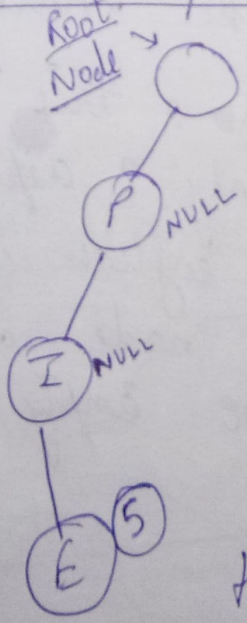
→ Each node ~~in~~ ~~a~~ in a trie including the root node has only 2 aspects. When a trie representing the english language, it consist of single Root node and Root node Value is set to be empty string "".

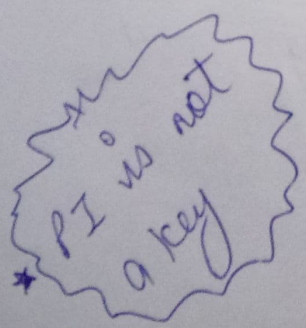
eg insert Peter, Piper, Picked, peck, pickled




→ each branch represents a word.

## Searching Through a Trie




* If we search for a key "PIE" we can look at its value and return
* If we are searching for "PI" and find the node value to be Null at last character -

# Difference Between Tries and Hash tables

→ Hash table should be used for lookups only. as it is O(1) as compared to ~~O(t~~ O(k) in tries where k is length of the word.

→ If your application performs operations like partial search, all strings with given prefix, all words in sorted order. (Go with trie).

---

m = length of longest word.
n = number of words in a trie
a = length of word you are searching for

| Trie operation | Worst |
|---|---|
| Create | O(m.n) |
| Lookup | O(a.n) |
| Insert | O(a.n) |
| Delete | O(a.n) |

Hashtable

| Avg | ~~Av~~ Worst |
|---|---|
| Ø | — — |
| O(1) | O(n) |
| O(1) | O(n) |
| O(1) | O(n) |

Looking up entire word is easy in hashtable However tries allows you to look up words by their prefixes; something that hashtable cannot do because keys can't split.

```cpp
#include<bits/stdc++.h>
using namespace std;
class TireNode{
public:
    unordered_map<char, TireNode*> children;
    char val;
    bool isEnd = false;

    TireNode(){}

    TireNode(char v){
        this->val = v;
    }
};

class Trie {
public:
    /** Initialize your data structure here. */
    Trie() {
        root = new TireNode();
    }

    /** Inserts a word into the trie. */
    void insert(string word) {
        TireNode* node = root;
        for(int i = 0; i < word.size(); i++){
            char c = word[i];
            if(node->children.find(c) == node->children.end()){
                node->children[c] = new TireNode(c);
            }

            node = node->children[c];
        }

        node->isEnd = true;
    }

    /** Returns if the word is in the trie. */
    bool search(string word) {
        TireNode* node = root;
        for(int i = 0; i < word.size(); i++){
            char c = word[i];
            if(node->children.find(c) == node->children.end()) return false;
            node = node->children[c];
        }

        return node->isEnd;
    }                                    // check is it at the
```

```cpp
    void insert(string word) {
        TireNode* node = root;
        for(int i = 0; i < word.size(); i++){
            char c = word[i];
            if(node->children.find(c) == node->children.end()){
                node->children[c] = new TireNode(c);                    // if
            }

            node = node->children[c];
        }

        node->isEnd = true;
    }

    /** Returns if the word is in the trie. */
    bool search(string word) {
        TireNode* node = root;
        for(int i = 0; i < word.size(); i++){
            char c = word[i];
            if(node->children.find(c) == node->children.end()) return false;
            node = node->children[c];
        }

        return node->isEnd;                                 // check is it at the end of t
    }

    /** Returns if there is any word in the trie that starts with the given prefix. */
    bool startsWith(string prefix) {
        TireNode* node = root;
        for(int i = 0; i < prefix.size(); i++){
            char c = prefix[i];
            if(node->children.find(c) == node->children.end()) return false;
            node = node->children[c];
        }

        return true;
    }
private:
    TireNode* root;
};
int main()
{
    Trie trie;

    trie.insert("Apoorva");
    // if(trie.search("Apoo")==true)cout<<"yes";
    // else cout<<"false";
    if(trie.startsWith("Apoo")==true)cout<<"yes";
```