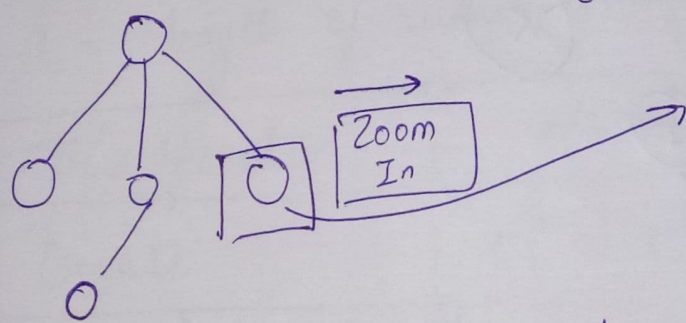


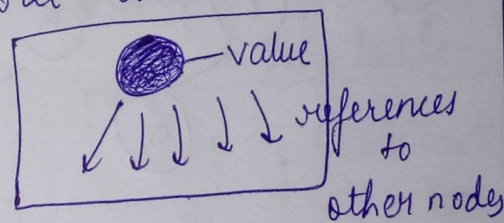
# Trie

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.



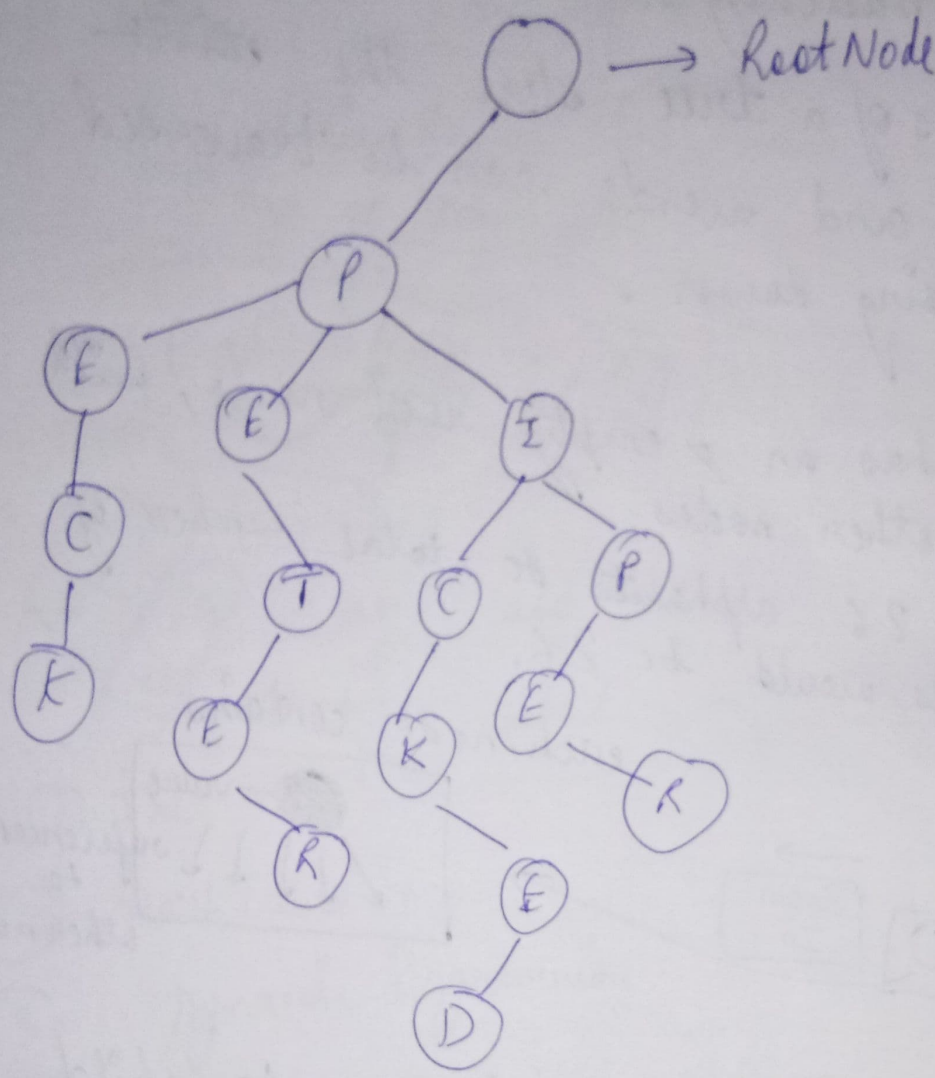
each node contains



- value can be null and references to child nodes also might be null.
- Each node ~~is 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 "".

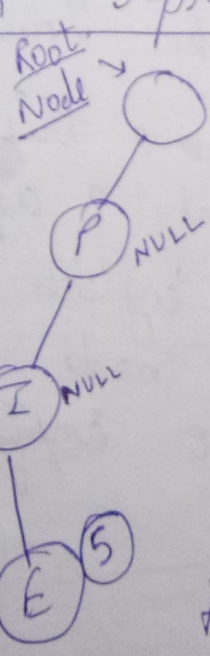


eg insert Peter, Piper, Picked, peek, pickled



→ each branch represents a word

Searching Through  
a Trie



\* PI is not a key

\* if we search for a key "PIE" we can look at its value and return

\* if we are searching of "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  $O(k)$  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	$O(m \cdot n)$
Lookup	$O(a \cdot n)$
Insert	$O(a \cdot n)$
Delete	$O(a \cdot n)$

Avg	<del>Avg</del> Worst
$\emptyset$	---
$O(1)$	$O(n)$
$O(1)$	$O(n)$
$O(1)$	$O(n)$

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.



```

1 #include <bits/stdc++.h>
2 using namespace std;
3 class TireNode{
4 public:
5     unordered_map<char, TireNode*> children;
6     char val;
7     bool isEnd = false;
8
9     TireNode(){}
10
11     TireNode(char v){
12         this->val = v;
13     }
14 };
15
16 class Trie {
17 public:
18     /** Initialize your data structure here. */
19     Trie() {
20         root = new TireNode();
21     }
22
23     /** Inserts a word into the trie. */
24     void insert(string word) {
25         TireNode* node = root;
26         for(int i = 0; i < word.size(); i++){
27             char c = word[i];
28             if(node->children.find(c) == node->children.end()){
29                 node->children[c] = new TireNode(c);
30             }
31
32             node = node->children[c];
33         }
34
35         node->isEnd = true;
36     }
37
38     /** Returns if the word is in the trie. */
39     bool search(string word) {
40         TireNode* node = root;
41         for(int i = 0; i < word.size(); i++){
42             char c = word[i];
43             if(node->children.find(c) == node->children.end()) return false;
44             node = node->children[c];
45         }
46
47         return node->isEnd;
48     }
49
50     // check is it at the

```



```

Help
a.cpp x sanketSingh.txt SanketTemplate.cpp check.cpp trie.cpp c_cpp_properties.json
pp > Trie > insert(string)

4 void insert(string word) {
5     TireNode* node = root;
6     for(int i = 0; i < word.size(); i++){
7         char c = word[i];
8         if(node->children.find(c) == node->children.end()){
9             node->children[c] = new TireNode(c); // if
10        }
11
12        node = node->children[c];
13    }
14
15    node->isEnd = true;
16 }
17
18 /** Returns if the word is in the trie. */
19 bool search(string word) {
20     TireNode* node = root;
21     for(int i = 0; i < word.size(); i++){
22         char c = word[i];
23         if(node->children.find(c) == node->children.end()) return false;
24         node = node->children[c];
25     }
26
27     return node->isEnd; // check is it at the end of t
28 }
29
30 /** Returns if there is any word in the trie that starts with the given prefix. */
31 bool startsWith(string prefix) {
32     TireNode* node = root;
33     for(int i = 0; i < prefix.size(); i++){
34         char c = prefix[i];
35         if(node->children.find(c) == node->children.end()) return false;
36         node = node->children[c];
37     }
38
39     return true;
40 }
41
42 private:
43     TireNode* root;
44 };
45
46 int main()
47 {
48     Trie trie;
49
50     trie.insert("Apoorva");
51     // if(trie.search("Apoo")==true)cout<<"yes";
52     // else cout<<"false";
53     if(trie.startsWith("Apoo")==true)cout<<"yes";
54     // else cout<<"false";
55 }

```