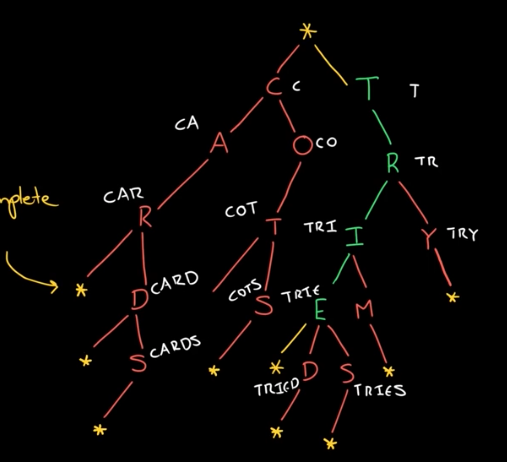
# Tries(Prefix Trees)

* Each node saves a letter.
* Each path saves a word.
* '\*' determines that word is complete.
* Rather than having left and right, you have hashmaps that says, do you have this character? Yes? Give me the node.
* It also has a boolean called completed which says if the word is complete.
* This data structure is very common in case of validation of words.
* Used in prefix based search.
* 

## Node

* **public class TrieNode {**
* **public HashMap<Character, TrieNode> map;**
* **public boolean endOfWord;**
* **public TrieNode() {**
* **map = new HashMap<Character, TrieNode>();**
* **endOfWord = false;**
* **}**
* **}**

## Insertion

* Start from the root node of the trie.
* Check in the map if the next character is there or not.
* If it is not there then add an element in the hashmap with the character and a new TrieNode.
* Then go to the specific TrieNode and do the same thing with the next character of the word.
* Once you have reached the end of the word, set the endOfWord to true.

## Search

* See if you have the next character in the map.
* If you dont then return false.
* If you do then go to the next node and the next character in string.
* Once you have reached then end, check if endOfWord is true or node.
* If it is true then you have found the word.
* If it is not then you have not found the word and return false.

## Deletion

* This will be solved using recursion.
* First make the base case if the length of string is zero.
* If length of string is zero and then endOfWord boolean is true then turn it to false.
* Now if it has children then return the node, otherwise return the null.
* Now check if the next character is there in map.
* If its not then return node because the word is not there and nothing has to be removed.
* If it is there then keep going in recursively with the node of the character and substring from character 1.
* Once you reach the end, if you are getting back null then remove that specific mapping in the hashmap.
* After this check if the map has any elements left.
* If it doesn’t then return null because even this has to be removed.
* If the map has any elements then just return the node.

## Search if a character is there in the HashMap

* **for (int i = 'A'; i <= 'z'; i++) {**
* **if (node.map.get((char)i) != null) System.out.println((char)i);**
* **}**

## Preorder Traversal

* **for (int i = 'A'; i <= 'z'; i++) {**
* **if (node.map.get((char)i) != null) System.out.println((char)i);**
* **}**

## How to handle duplicate words

* Change the TrieNode. Rather than the endOfWord as a boolean, change it to int and specify how many words are there.
* **class TrieNodeSpecial {**
* **public HashMap<Character, TrieNodeSpecial> map;**
* **public int words;**
* **public TrieNodeSpecial() {**
* **map = new HashMap<Character, TrieNodeSpecial>();**
* **words = 0;**
* **}**
* **}**