

# ECE2810J

## Data Structures and Algorithms

### Tries

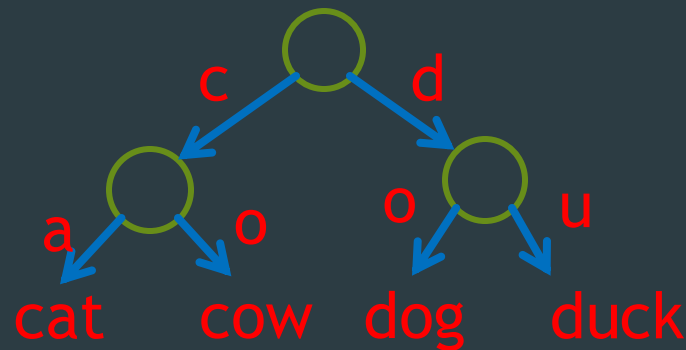
#### ► Learning Objectives:

- Know what a trie is and understand its difference between binary search trees
- Know how to implement search, insertion, and removal for a trie

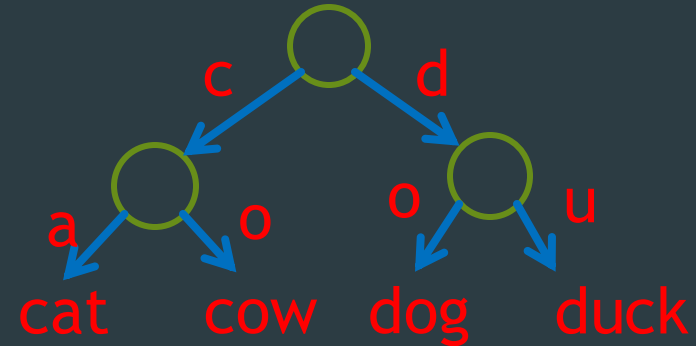


# Trie

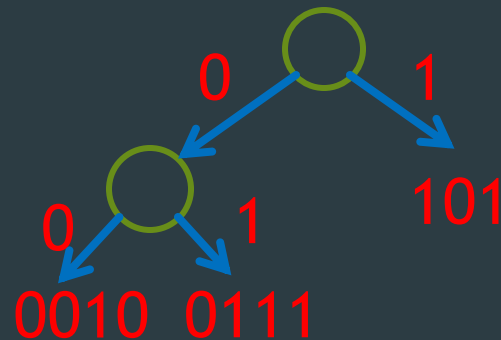
- ▶ The word “trie” comes from re**trie**val.
  - ▶ To distinguish with “tree”, it is pronounced as “try”.
- ▶ A trie is a tree that uses parts of the key, as opposed to the whole key, to perform search.
- ▶ Data records are only stored in **leaf** nodes. Internal nodes do not store records; they are “**branch**” points to direct the search process.



# Trie

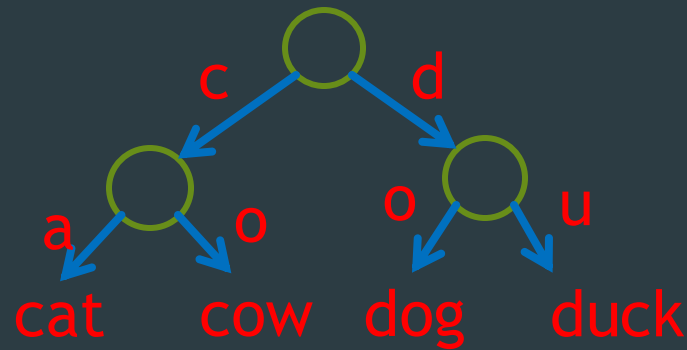


- ▶ Trie usually is used to store a set of strings from an **alphabet**.
  - ▶ The alphabet is in the general sense, not necessarily the English alphabet.
- ▶ For example,  $\{0, 1\}$  is an alphabet for binary codes  $\{0010, 0111, 101\}$ . We can store these three codes using a trie.



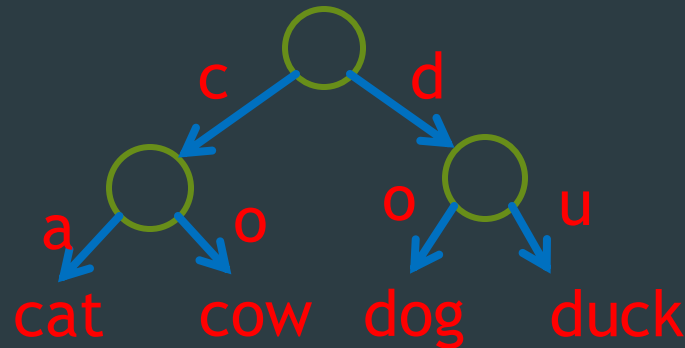
# Trie

- ▶ Each edge of the trie is labeled with symbols from the alphabet.
- ▶ Labels of edges on the path from the root to any leaf in the trie forms a **prefix** of a string in that leaf.
  - ▶ Trie is also called **prefix-tree**.



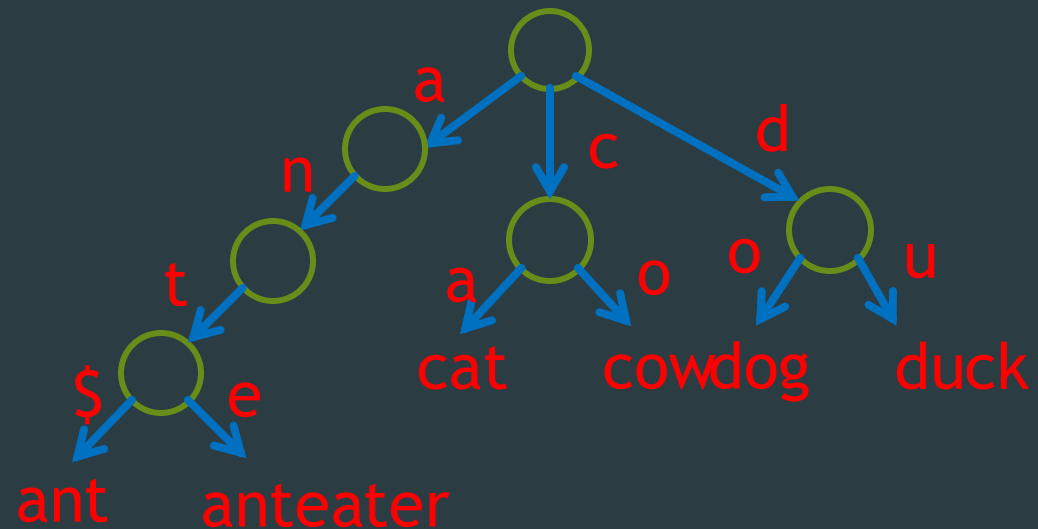
# Trie

- ▶ The most significant symbol in a string determines the branch direction at the root.
- ▶ Each internal node is a “**branch**” point.
- ▶ As long as there is only one key in a branch, we do not need any further internal node below that branch; we can put the word directly as the leaf of that branch.



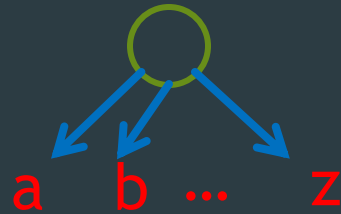
# Trie Implementation Issue

- ▶ Sometimes, a string in the set is exactly a **prefix** of another string.
  - ▶ For example, “ant” is a prefix of “anteater”.
  - ▶ How can we make “ant” as a leaf in the trie?
- ▶ We add a symbol to the alphabet to indicate the end of a string. For example, use “\$” to indicate the end.

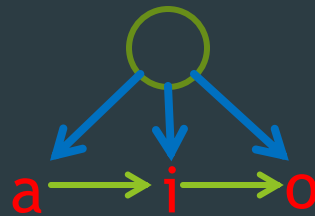


# Trie Implementation Issue

- ▶ We can keep an array of pointers in a node, which corresponds to **all** possible symbols in the alphabet.

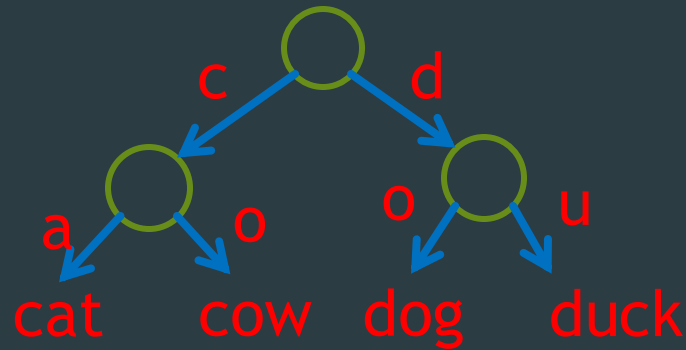


- ▶ However, most internal nodes have branches to only a small fraction of the possible symbols in the alphabet.
  - ▶ An alternate implementation is to store a linked list of pointers to the child nodes.



# Trie Search

- ▶ Follow the search path, starting from the root.
- ▶ When there is no branch, return false.
- ▶ When the search leads to a leaf, further compare with the key at the leaf.



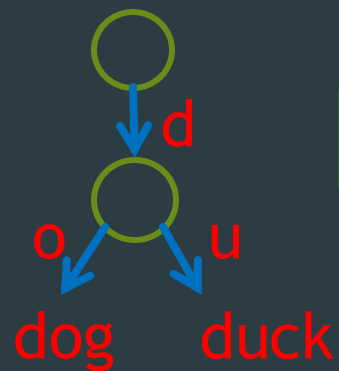
search “monkey”

search “cat”

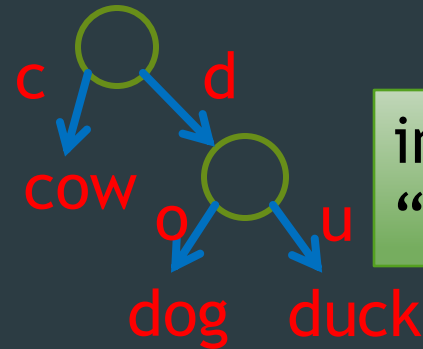


# Trie Insertion

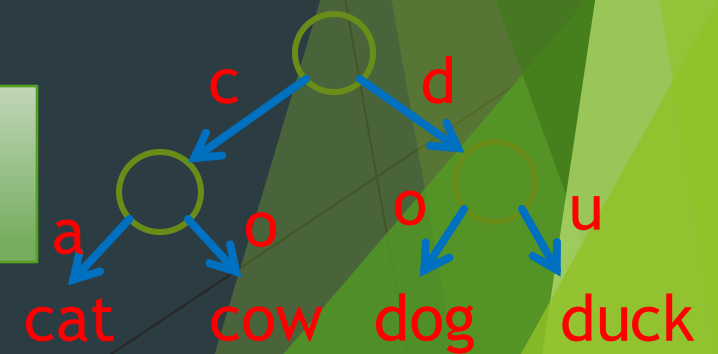
- ▶ Follow the search path, starting from the root.
- ▶ If a new branch is needed, add it.
- ▶ When the search leads to a leaf, a conflict occurs. We need to branch.
  - ▶ Use the next symbol in the key
  - ▶ The originally-unique word must be moved to lower level



insert "cow"

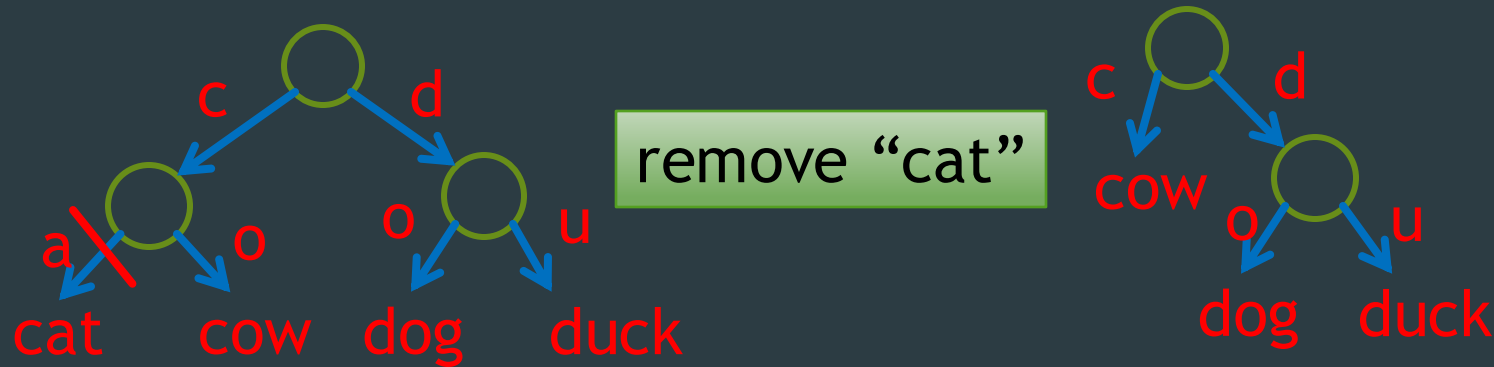


insert "cat"



# Trie Removal

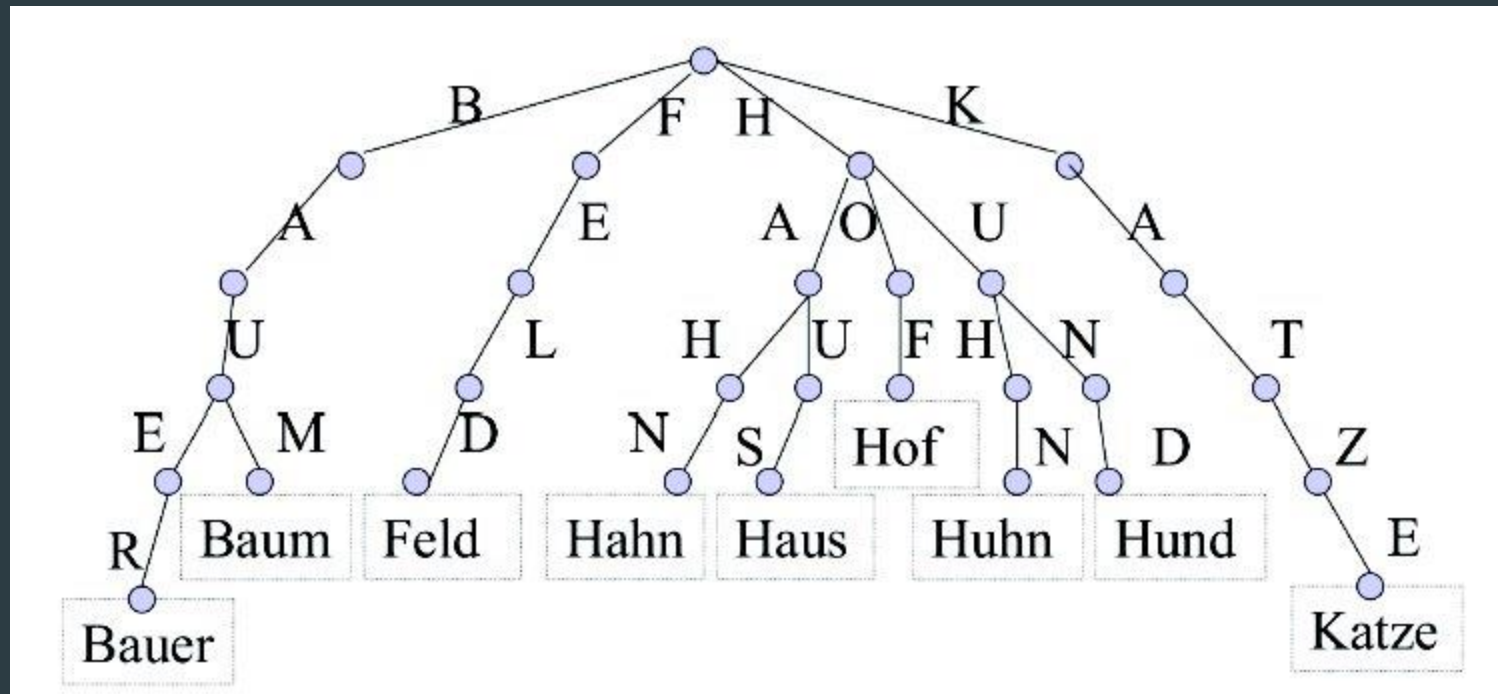
- ▶ The key to be removed is always at the leaf.
- ▶ After deleting the key, if the parent of that key now has only one child  $C$ , remove the parent node and move key  $C$  one level up.
  - ▶ If key  $C$  is the only child of its new parent, repeat the above procedure again.



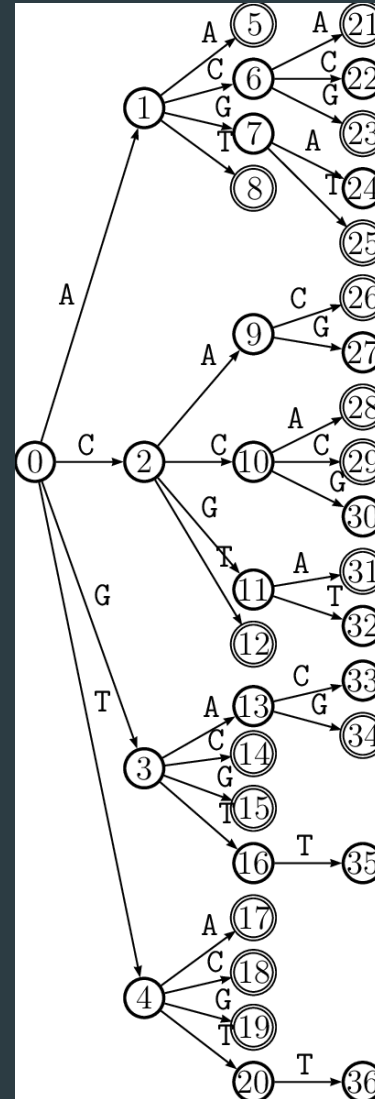
# Time Complexity of Trie

- ▶ In the worst case, inserting or finding a key that consists of  $k$  symbols is  $O(k)$ .
  - ▶ This does not depend on the number of keys  $N$ .
  - ▶ Comparison: storing 32 integers in the range  $[0, 127]$  using a trie versus using a BST. What are heights in the **worst case**? BST: 32; Trie: 7
- ▶ Sometimes we can access records even faster.
  - ▶ A key is stored at the depth which is enough to distinguish it with others.
  - ▶ For example, in the previous example, we can find the word “duck” with just “du”.

# Use Case 1 - Spell Checking



# Use Case 2 - Human Genome



# Exercise 1

## Problem 208. Implement Trie (Prefix Tree)

Search

The screenshot shows a coding platform interface with a dark theme. At the top, there are topic filters: Array (1495), String (643), Hash Table (519), Dynamic Programming (468), Math (459), Sorting (347), Greedy, and an Expand button. Below these are category buttons: All Topics, Algorithms, Database, pandas, JavaScript, and Shell. A search bar contains the text '208' and is highlighted with a red box. A red arrow points from the word 'Search' to this search bar. Below the search bar are filters for Lists, Difficulty, Status, and Tags. A 'Pick One' button and a 'Reset' button are also visible. The main content area displays a table of search results.

Status	Title	Solution	Acceptance	Difficulty	Frequency
	2785. Sort Vowels in a String		82.3%	Medium	
	208. Implement Trie (Prefix Tree)		64.1%	Medium	