Trie: Introduction

Let's go over the Trie pattern, its real-world applications, and some problems we can solve with it.

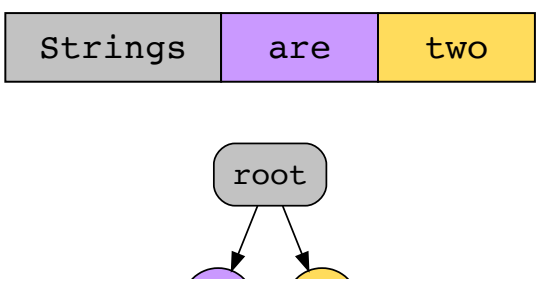
We'll cover the following

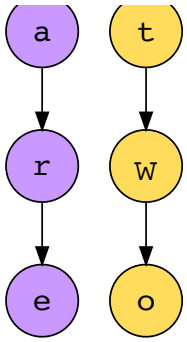
Overview

Trie is a tree data structure used for storing and locating keys from a set. The keys are usually strings that are stored character by character—each node of a trie corresponds to a single character rather than the entire key.

The order of characters in a string is represented by edges between the adjacent nodes. For example, in the string "are", there will be an edge from node a to node r to node e. That is, node a will be the parent of node r, and node r will be the parent of node e.

The following illustration can help you understand how the strings are stored:

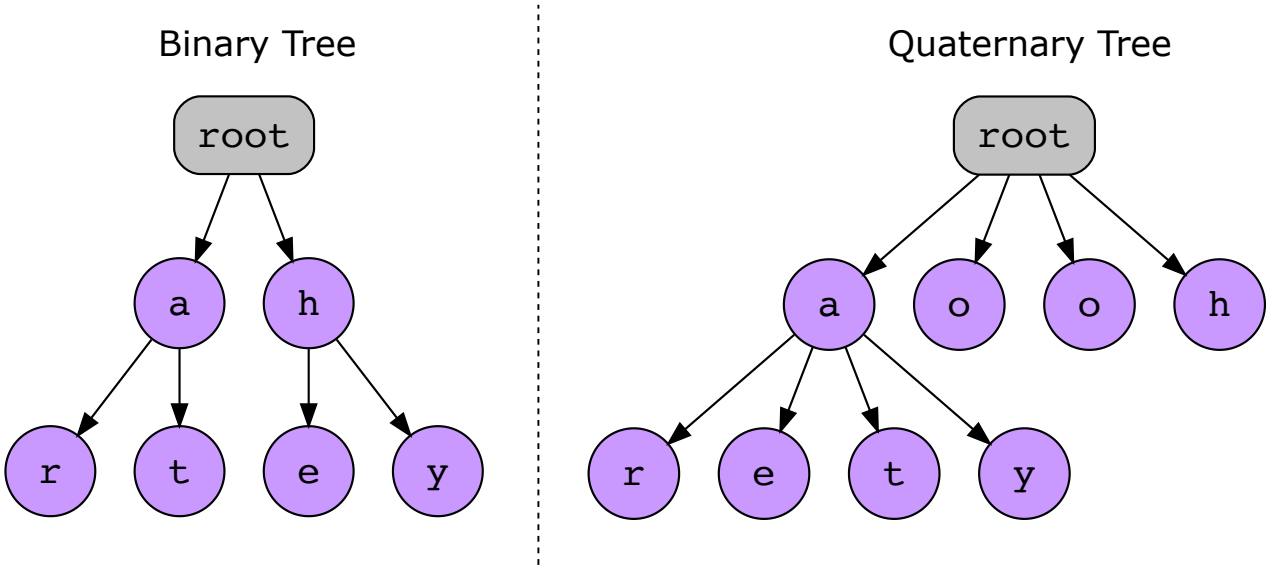




Storing words in a trie

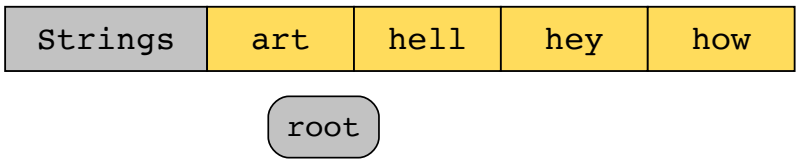
This way, additional space is not required for storing strings with common prefixes. We can keep moving down the tree until a new character, that's not present in the node's children, is encountered and add it as a new node. Similarly, searches can also be performed using depth first search by following the edges between the nodes. Essentially, in a trie, words with the same prefix or stem share the memory area that corresponds to the prefix.

To understand how tries are more efficient for storing and searching strings, consider a binary tree. The time complexity of a binary tree is $O(\log n)$, where we talk in terms of log base 2. Instead, think of a quaternary tree, where every node has a fan-out of four, so each node can have four children. The time complexity of this tree is still $O(\log n)$. However, now we're talking in terms of log with base 4. That's an improvement in the performance even if it's by a constant factor. As our trees become wider and shorter, the operations become more efficient. This is because we don't have to traverse as deep.



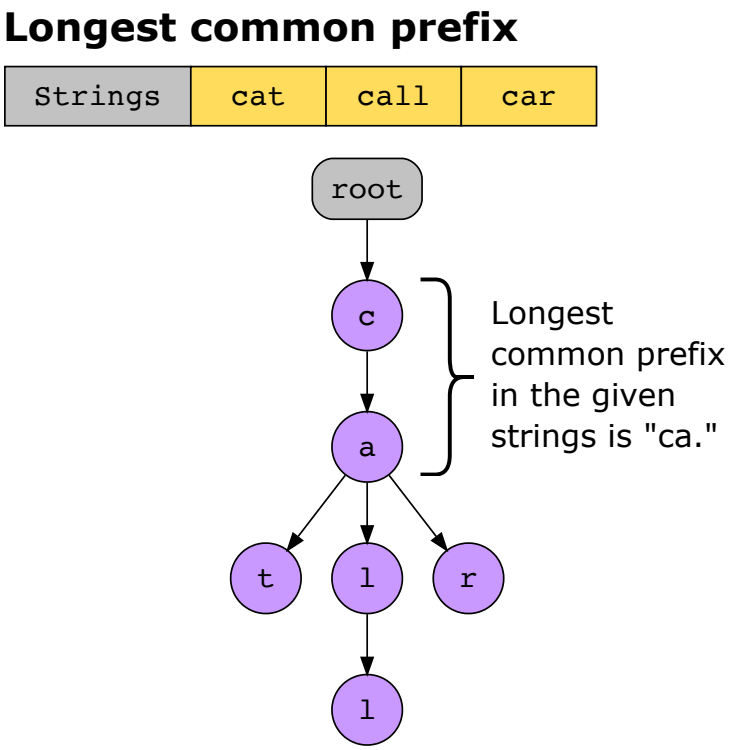
This is exactly the motivation behind a trie. What if we had an n-ary tree with the fan-out equal to the number of unique values in the given dataset? For example, if we're considering strings in English, the fan-out would be 26, corresponding to the number of letters in the English language. This makes the tree wider and shorter! The maximum depth of the trie would be the maximum length of a word or string.

The following illustration shows how strings are stored in a trie:



Examples

The following examples illustrate some problems that can be solved with this approach:



1 of 2

Does my problem match this pattern?

- Yes, if either of these conditions is fulfilled:
  - We need to compare two strings to detect partial matches, based on the initial characters of one or both strings.
  - We wish to optimize the space used to store a dictionary of words. Storing shared prefixes once allows for significant savings.
- No, if either of these conditions is fulfilled:

- The problem statement restricts us from breaking down the strings into individual characters.
- Partial matches between pairs of strings are not significant to solving the problem.

Real-world problems

examples.

- **Autocomplete system:** One of the most common applications of trie is the autocomplete system in search engines, such as Google. This is the feature that prompts the search engine to give us some suggestions to complete our query when we start typing something in the search bar. These suggestions are given based on common queries that users have searched already that match the prefix we have typed.
- **Orthographic corrector:** Ever seen pop-up suggestions or red lines under a word while you're typing a message? That's an orthographic corrector making suggestions and pointing out spelling mistakes by searching through a dictionary. It uses a trie data structure for efficient searches and retrievals from the available database.

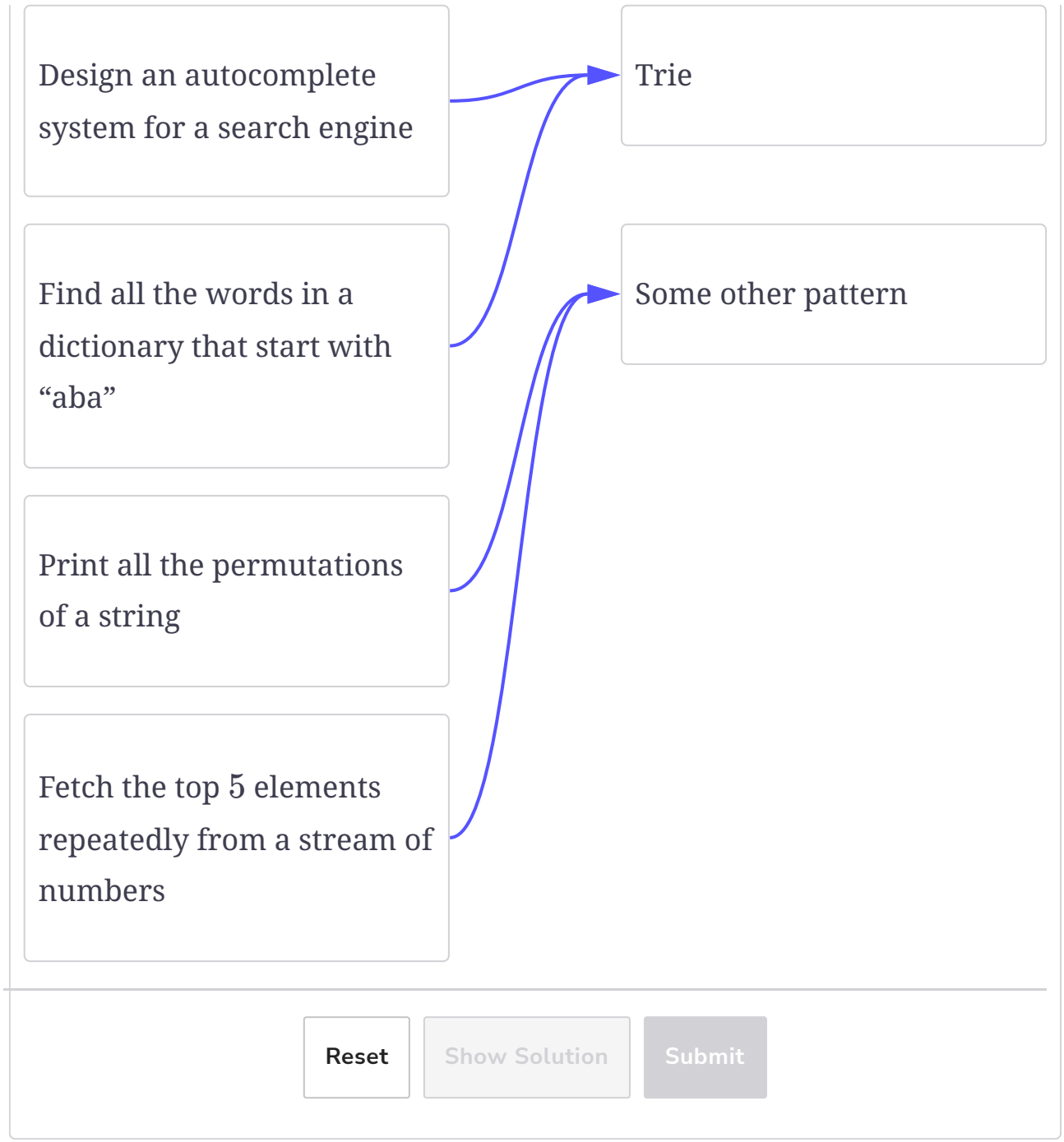
Strategy time!

Match the problems that can be solved using the trie pattern.

> **Note:** Select a problem in the left-hand column by clicking it, and then click one of the two options in the right-hand column.

Match The Answer

Select an option from the left-hand side



Reset | Show Solution | Submit