# **Trie Introduction**

* A ***Trie*** is a general tree, meaning that each node can have ***any number of children***.
* A ***Trie*** stores a set of words in the following way:
  + Each child of a node is labeled with a character.
  + Each node contains a boolean indicating whether the labels in the path from the root to that node form a word in the set.

A ***Trie*** is a type of dictionary made for storing "words'' (types made up of letters).

Diagram

Description automatically generated

This trie represents the dictionary: {adam, add, app, bad, bag, bags, beds, bee, cab}, because if we go from the root of the trie reading in letters until we hit a "true'' node, we get a word.

Recall that in huffman, we had two possibilities (0 and 1) and we read from the root to a leaf.

It is used to store a dictionary (list) of words that can be searched on, in a manner that allows for efficient generation of completion lists.

The word list is originally stored in an array, and the trie is built off of this array. Here are some examples of word lists and the tries built to store the words, followed by an explanation of the trie structure and its relationship to its source word list.

A Trie (pronounced “try”) is a tree that exploits some structure in the keys

e.g. if the keys are strings, a binary search tree would compare the entire strings, but a trie would look at their individual characters

Suffix trie are a space-efficient data structure to store a string that allows many kinds of queries to be answered quickly.

Chart, scatter chart, box and whisker chart

Description automatically generated