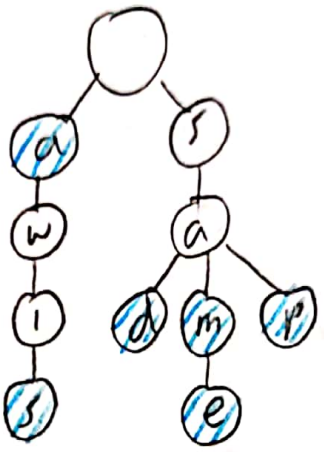
Tries

- Basics
  - bood for tracking strings
  - Same as tree bout every node represents a char
  - Root node is a sentinel node
  - A word represents sum of chars from sentinel node to a "marked node" (end of word)
  - example:

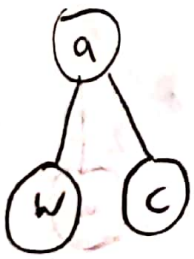
Contains("a") = true
contains("a w/s") = true
Contains("sad") = true
Contains("sam") = true
contains("same") = true
Contains("sap") = true
} "hit"

Contains("sa") = false
Contain("sax") = false
} "miss"

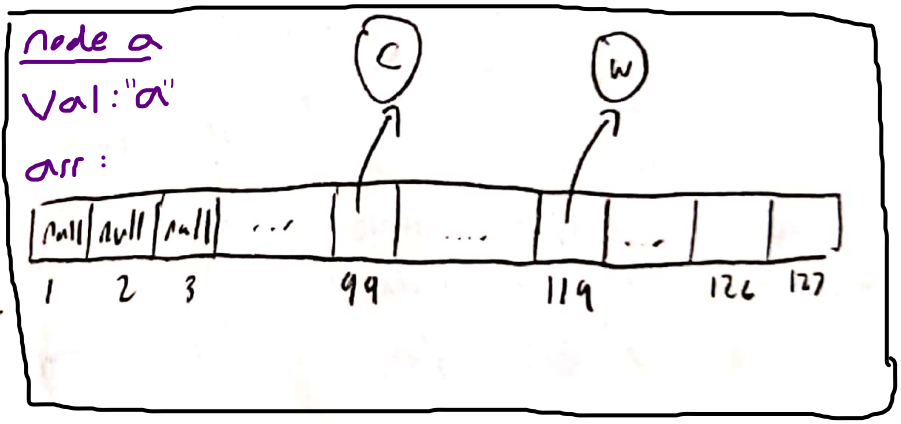


/// = marked
↑ indicates ending of word

- Representing node's children: Data Indexed Array Implementation
  - create a size 128 array w/ the ith index representing a child node with the ith ASCII char (128 ASCII chars)



w = 119th ASCII char
c = 99th ASCII char

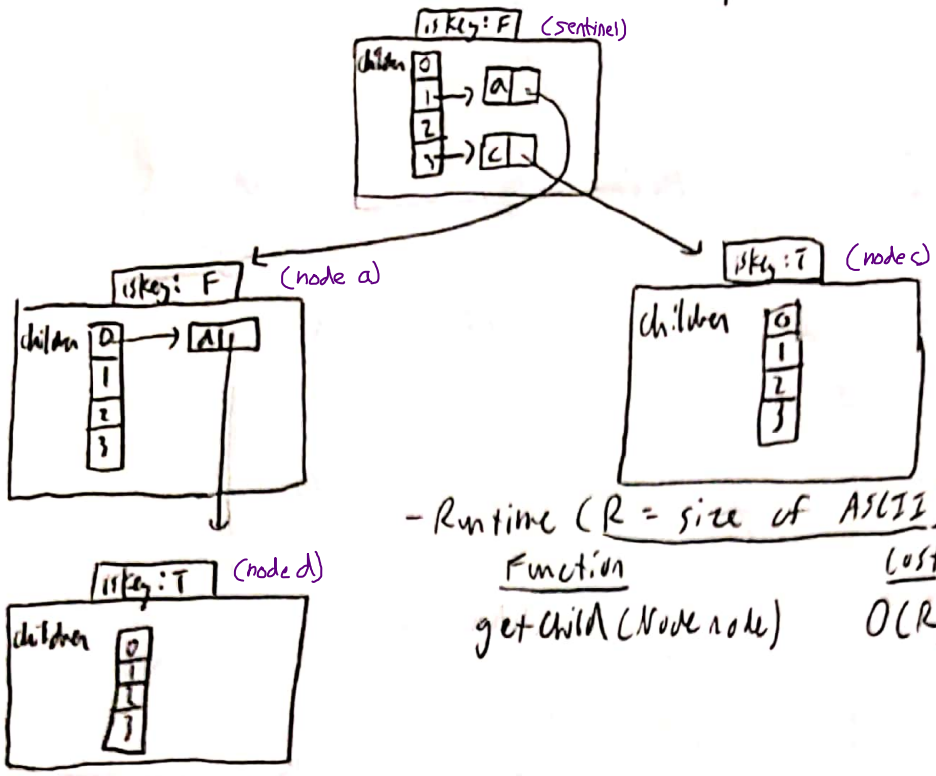


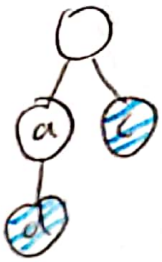
  - Runtime:

Function	Cost
add(String word)	$\Theta(word.length()) \approx \Theta(1)$
contains(String word)	$\Theta(word.length()) \approx \Theta(1)$
getChild(Node node)	$\Theta(1)$

  - Downside: many useless nulls

- Representing children 2: Hash-Table Based Trie Implementation

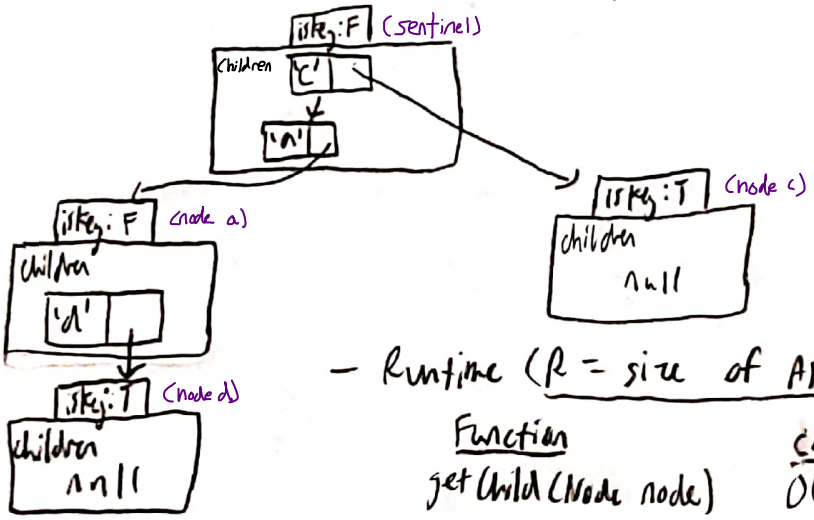


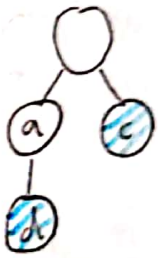


- Runtime (R = size of ASCII)

Function	Cost
getChild(Node node)	$O(R) \approx \Theta(1)$, since R fixed

- Representing children 3: BST/Treeset Based Trie Implementation





- Runtime (R = size of ASCII)

Function	Cost
getChild(Node node)	$O(\log R) \approx \Theta(1)$, since R fixed

- Data Index Array vs. Hash-table vs. BST
  - Data Index Array was more space
  - Hash-Table and BST slightly slower for finding children ($O(R)$ and $O(\log R)$)