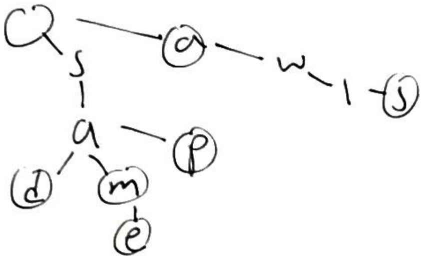
CS Lecture: Tries #37.

We have:

- Use Comparison based sorting ← uses (Compare/compare to).
  - Small-Alphabet (Integer) Sorting
  - Digit-by-Digit sorting

Tries:

Suppose we insert sam, sad, sap, same, a, and awls.



○ = end of a word (marked)

- A different type of data set.
  - we have BST, Hashsets, ... right now.
- Tries potentially save space because not each word needs to be stored.
- Short for re$\boxed{\text{tri}}$eval tree

Given a Trie with N keys & a key w/ L digits: → entire word.

What is the

- Worst case RT for insert: Θ(L)
- Worst case search RT: Θ(L)
- Best case search RT: Θ(1)
  - ↳ first digit is a miss.

*Assume child can be found in constant time.

Trie worst case: Θ(L)

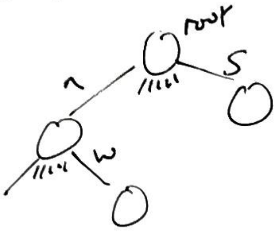
Best case: Θ(1)

- Comparison is done digit-by digit, so we can skip some characters.

Uses of Tries!

- support rapid prefix matching.
  - Finding keys with a particular prefix.
  - Finding longest prefix of: longest prefix of ("sample")
    - → "sam" is longest prefix of sample.

Implementation:



"Broken" edges are just null.

Longest Prefix of:

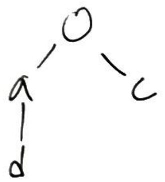
- Check each digit in turn, walk down the tree, keep track of most recent blue thing.
- If next isn't there, return current word.

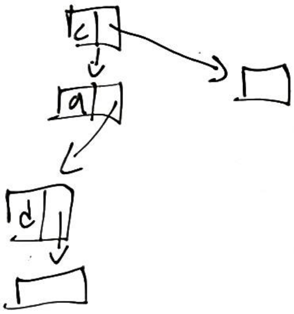
Tries:

- great performance character-based operations.
- Very memory hungry!

Child link optimizations:

- previously, all children were arrays.
- How about using a map?





Ternary Search Trees

Insert "sam", Insert "sad", Insert "sap", Insert "sae", Insert "a"

every node has exactly 3 links