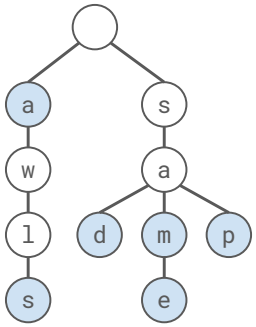
Searching in Tries

contains("sam"): true, blue. **hit**.
contains("sa"): false, white. **miss**.
contains("a"): true, blue. **hit**.
contains("saq"): false, fell off. **miss**.

Two ways to have a **search miss**.

1. If the final node is white.
2. If we fall off the tree.



Given a trie with N keys, what is the runtime for contains given a key of length L?

$\Theta(\log L)$

$\Theta(L)$

$\Theta(\log N)$

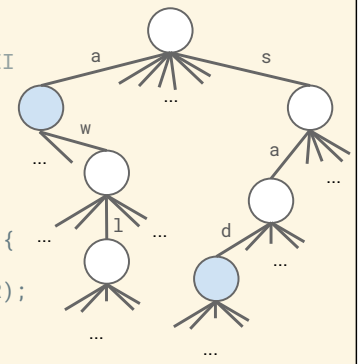
$\Theta(N)$

$\Theta(N + L)$

Trie: Design 1.5

```
public class TrieSet {
  private static final int R = 128; // ASCII
  private Node root;

  private static class Node {
    private char ch;
    private boolean isKey;
    private DataIndexedCharMap<Node> next;
    private Node(char c, boolean b, int R) {
      ch = c; isKey = b;
      next = new DataIndexedCharMap<Node>(R);
    }
  }
}
```



Does the structure of a trie depend on the order in which strings are inserted?

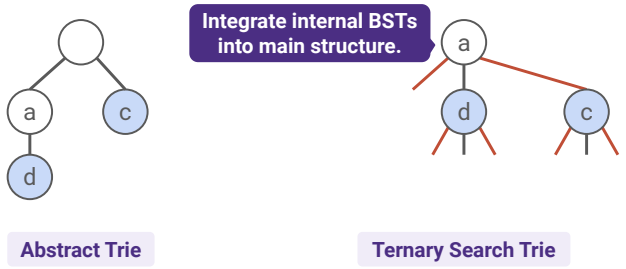
Yes

No

Not sure

We can remove the character ch from the node because we'll know which character we're on when we index into the DataIndexedCharMap next.

v4.0: Ternary Search Trie (TST)

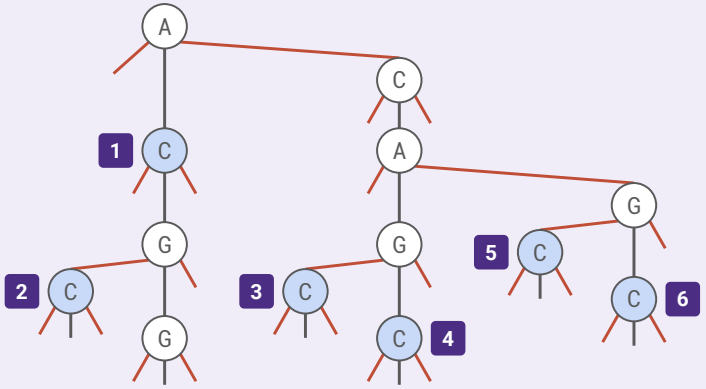


17

?: How do you look up the string “ad” in the ternary search trie? The string “c”?

?: How is the ternary search trie different from the abstract trie? From the BST-based trie?

Q Which value is associated with the key “CAC”?



18

Tries in COS 226 (Sedgewick, Wayne/Princeton)

If you’re not sure where to start, look back at the previous example.

Q1: Which value is associated with the key “CAC”?

Search in a TST

Follow links corresponding to each character in the key.

- If less, take left link; if greater, take right link.
- If equal, take the middle link and move to the next key character.

Search hit. Final node is blue (`isKey == true`).

Search miss. Reach a null link or final node is white (`isKey == false`).

Does the structure of a TST depend on the order in which strings are inserted?

Yes

No

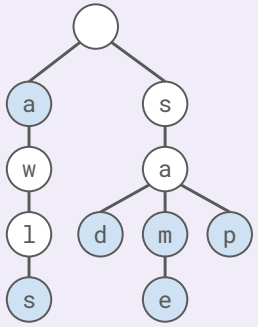
Not sure

Q Collecting Trie Keys

Describe in English an algorithm to collect all the keys in a trie.

`collect(): ["a","awls","sad","sam","same","sap"]`

1. Create an empty list of results x.
2. For character c in `root.next.keys()`:
   a. Call `colHelp("c", x, root.next.get(c))`.
3. Return x.



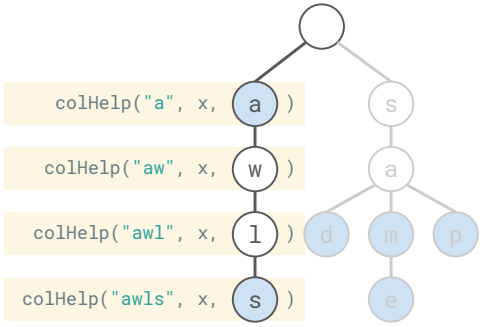
Abstract Trie

`colHelp(String s, List<String> x, Node n)`

1. ???

Collecting Trie Keys

```
collect(): [
   "a",
   "awls",
]
```



`colHelp(String s, List<String> x, Node n)`

Abstract Trie

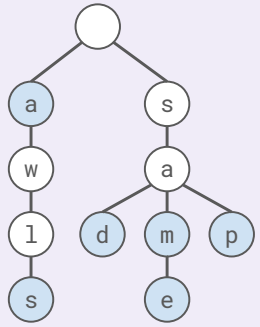
1. If `n.isKey`, then `x.add(s)`.
2. For character c in `n.next.keys()`:
   a. Call `colHelp(s + c, x, n.next.get(c))`.

Q1: Describe in English an algorithm to collect all the keys in a trie.

Q Prefix Operations with Tries

Describe in English an algorithm for `keysWithPrefix`.

`keysWithPrefix("sa"): ["sad","sam","same","sap"]`



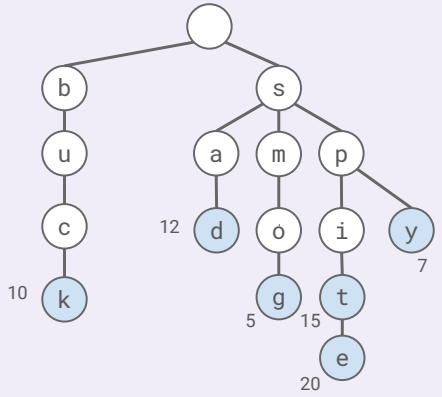
Abstract Trie

Q1: Describe in English an algorithm for `keysWithPrefix`.

Q Top-3 Matches for "s"

1. Call `keysWithPrefix("s")`.
   a. sad, smog, spit, spite, spy
2. Return the 3 keys with highest value.
   a. spit, spite, sad

This algorithm is slow. Why?



Abstract Trie

Q1: This algorithm is slow. Why?