DSC 190
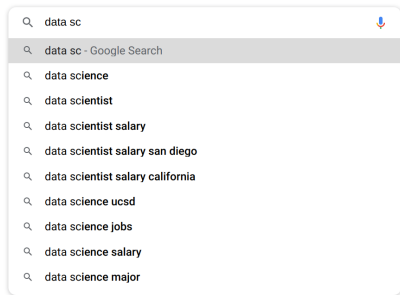
DATA STRUCTURES & ALGORITHMS

Today's Lecture

String Data Structures

▶ One of the themes of this quarter:

▶ If you're doing something once, use an algorithm.

▶ If you're doing it over and over, use an appropriate data structure.

String Data Structures

▶ Over the next two lectures, we'll look at data structures for strings.

▶ Today: **tries** for efficient repeated **prefix queries**.

Autocompletion

DSC 190

DATA STRUCTURES & ALGORITHMS

Tries

Trie

- ▶ A data structure for storing strings.

- ▶ Pronounced "try", short for "re**trie**val".

- ▶ Supports fast **prefix query** and **membership query**.

Prefixes

▶ A **prefix** p of a string s is a contiguous slice of the form s[0:t], for some *t*.

▶ Examples:
  ▶ "test" is a prefix of "testing"
  ▶ "te" is a prefix of "testing"
  ▶ "sa" is a prefix of "san diego"
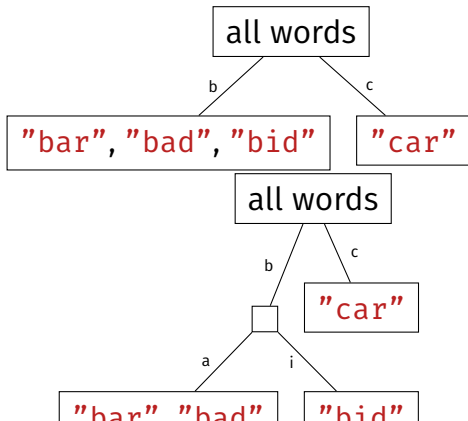 ▶ "di" is **not** a prefix of "san diego"

Prefix Query

▶ **Given**: a collection of *n* strings and a prefix, p.

▶ **Find**: all strings in the collection for which p is a prefix.

▶ Example:
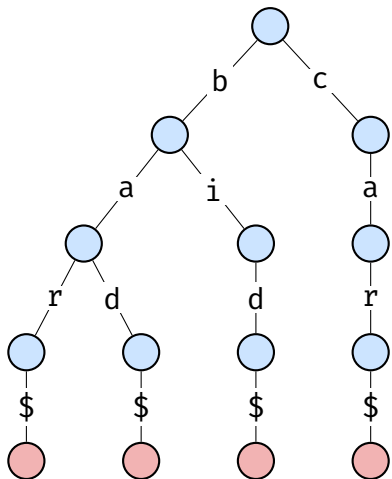 ▶ "bar", "bad", "bid", "car"
  ▶ p = "ba"

Brute Force

▶ Loop over each of *n* strings, compare against prefix p.

▶ Worst-case time: $\Theta(n \cdot |p|)$
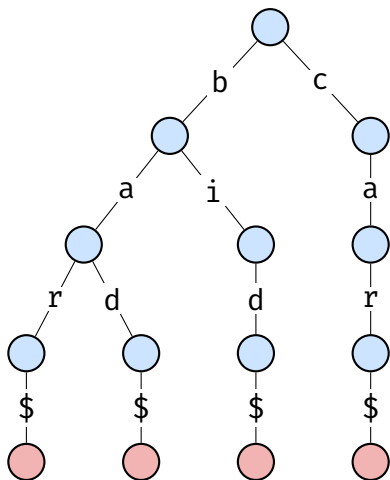
Trie: Motivation

"bar", "bad", "bid", "car"

```
              ┌───────────┐
              │ all words │
              └───────────┘
            b /           \ c
  ┌──────────────────────┐  ┌───────┐
  │ "bar", "bad", "bid"  │  │ "car" │
  └──────────────────────┘  └───────┘
              ┌───────────┐
              │ all words │
              └───────────┘
            b /           \ c
            ┌─┐          ┌───────┐
            └─┘          │ "car" │
          a /   \ i      └───────┘
  ┌──────────┐  ┌───────┐
  │"bar","bad"│  │ "bid" │
  └──────────┘  └───────┘
```

Tries

▶ Internal nodes represent prefixes.

▶ Leaf nodes represent full words.

▶ Edges are characters.

▶ Words are encoded as paths.

Sentinels

- ▶ $ is a **sentinel**.

- ▶ It is different from the dollar sign character.

- ▶ It marks the end of a word.

- ▶ Used to show that "bar" in trie, but "ba" not.

DSC 190
DATA STRUCTURES & ALGORITHMS

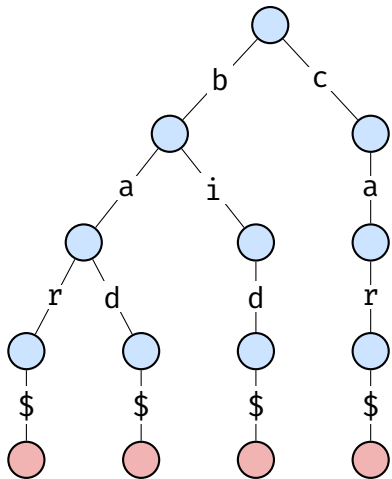**Implementing Tries**

Representation

▶ Each node has a hash table / array mapping characters to a child nodes.

▶ Sentinel represented with a singleton object?

```python
END_OF_STRING = object()

class TrieNode:

    def __init__(self):
        self.children = {}
```
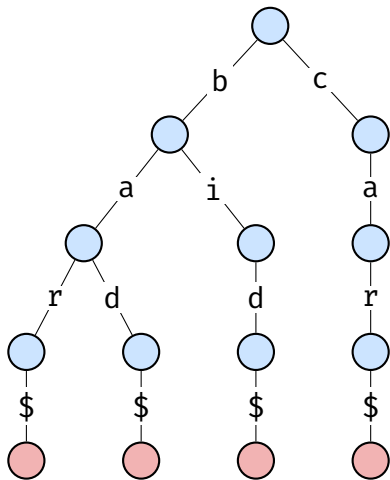
Insertion

- ▶ "Walk" down tree, creating edges and nodes as necessary.

- ▶ When no more letters left, add sentinel.

- ▶ Example: insert "cab", "card", "zoo"

Insertion (Recursive)

- ▶ Suppose we .insert(s) on root node.

- ▶ If s[0] not in self.children, create a new node.

- ▶ Otherwise, let child be self.children[s[0]].

- ▶ Recursively insert s[1:] into child.

```python
def insert(self, s, start=0, stop=None):
    """Insert s[start:stop] into the trie."""
    if stop is None:
        stop = len(s)

    if start >= stop:
        self.children[END_OF_STRING] = TrieNode()
        return

    if s[start] not in self.children:
        self.children[s[start]] = TrieNode()

    child = self.children[s[start]]
    child.insert(s, start + 1, stop)
```

Insertion Time Complexity

▶ Θ(|*w*|) time, where *w* is the string inserted.

▶ No matter how many elements in trie!

Walk
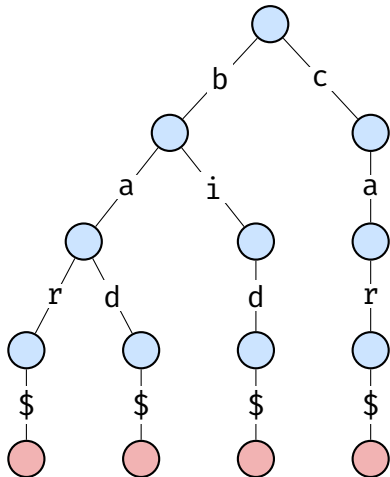
- ▶ Useful operation.

- ▶ Given a prefix, "walk" down tree.

- ▶ If we "fall off", raise error.

- ▶ Otherwise, return last node seen.

- ▶ Examples: "ba", "bo"

```python
def walk(self, s, start=0, stop=None):
    """Walk the trie following s[start:stop].
    Raises ValueError if falls off tree.
    Returns last node encountered otherwise."""
    if stop is None:
        stop = len(s)

    if start >= stop:
        return self

    if s[start] not in self.children:
        raise ValueError('Fell off tree.')
    else:
        child = self.children[s[start]]
        return child.walk(s, start + 1, stop)
```
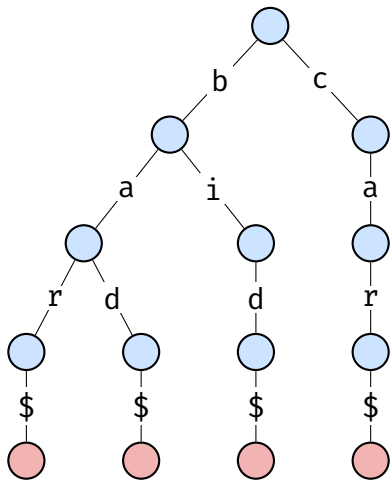
Walk Time Complexity

▶ Worst-case $\Theta(|p|)$ time, where p is the prefix searched.

▶ No matter how many elements in trie!

Membership Query

- Given p, return True/False if p in collection.

- "Walk" down tree.

- If we "fall off", return False.

- If not, check that sentinel in children.

- Examples: "ba", "bad"

```python
def membership_query(self, s, start=0, stop=None):
    """Determine if s[start:stop] is in trie."""
    try:
        node = self.walk(s, start, stop)
    except ValueError:
        return False

    return END_OF_STRING in node.children
```

Membership Query Time Complexity

▶ Worst-case $\Theta(|w|)$ time, where w is the prefix searched.

▶ No matter how many elements in trie!

Produce
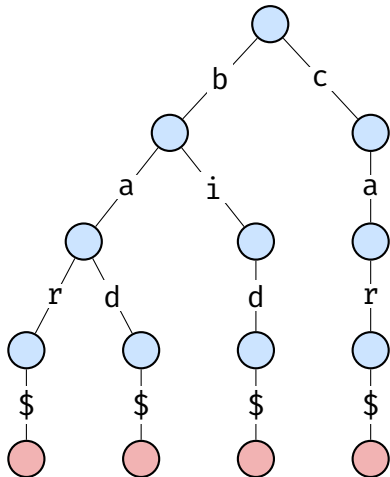
- ▶ Goal: generate all words in subtrie.

- ▶ Perform a DFS, keeping track of letters along path.

- ▶ If we find a sentinel, print path.

```python
def produce(self, pathchars=''):
    """Generate the words in the trie."""
    for letter, child in self.children.items():
        if letter is END_OF_STRING:
            yield pathchars
        else:
            yield from child.produce(pathchars + letter)
```

Produce Time Complexity

▶ Worst-case $\Theta(\ell)$ time, where ℓ is total length of all strings stored in the `trie`.

▶ If length strings is considered a constant, this is $\Theta(n)$.

Prefix Query (Complete)
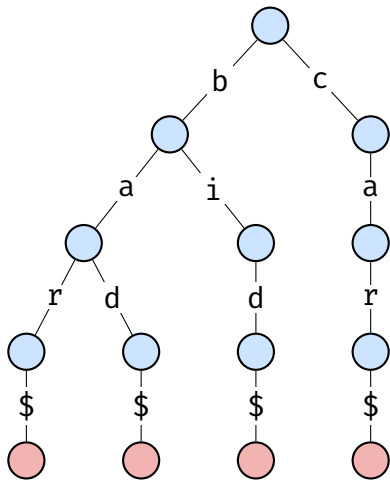
- ▶ Given p, return all completions.

- ▶ "Walk" down tree.

- ▶ If we "fall off", return empty list..

- ▶ If not, produce all nodes in subtrie.

- ▶ Examples: "ba", "bad"

```python
def complete(self, prefix):
    try:
        node = self.walk(prefix)
    except ValueError:
        return []
    return list(node.produce())
```

Prefix Query Time Complexity

▶ Worst-case $\Theta(|p| + \ell_p)$ time, where p is the prefix searched and ℓ_p is the total length of all matches.

▶ If length is considered constant, this is $\Theta(|p| + z)$, where z is number of matches.

DSC 190

DATA STRUCTURES & ALGORITHMS

Demo