CSCI 136
Data Structures & Advanced Programming

Lexicon Lab

Lexicon Lab

Lexicon

> *The vocabulary of a person, language, or branch of knowledge* - Oxford Languages

Implement a program that creates, updates, and searches a Lexicon (set of words)

- Supports
  - adding/removing words
  - searching for words and prefixes
  - reading words from files
  - Iterating over all words

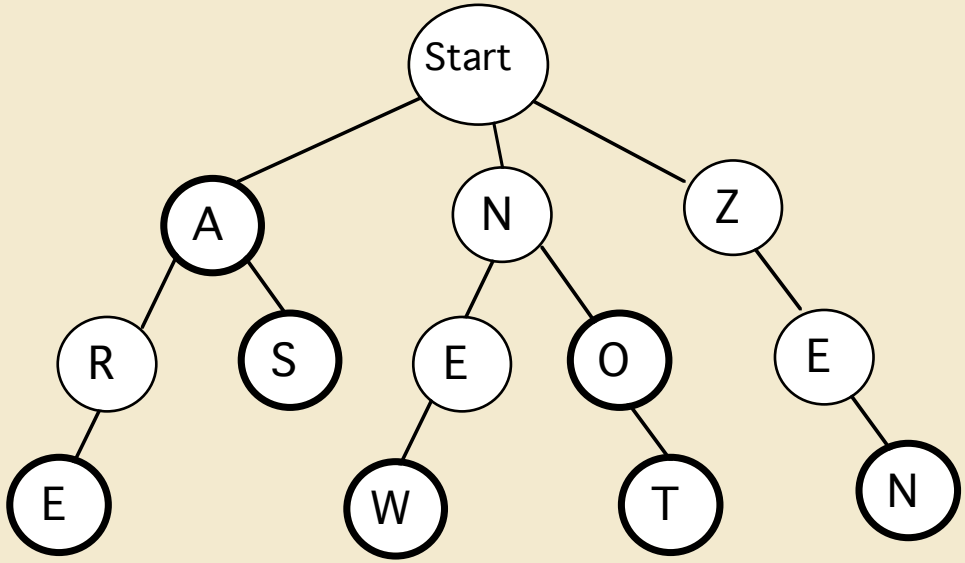
Lexicon Lab

Good lexicons also support efficient algorithms for

- Finding all words with a given prefix
- Finding all words in a given range
- Suggesting corrections for misspelled words
- *Wildcard searches*
  - A * wildcard refers to any set of 0 or more letters
  - A ? wildcard refers to any set of 0 or 1 letters
    - Example: bi*c?n* would match
      - bicentennial, bicentennials, biconcave, biconvex, billycan, billycans biomedicine

Lexicon Lab : Tries

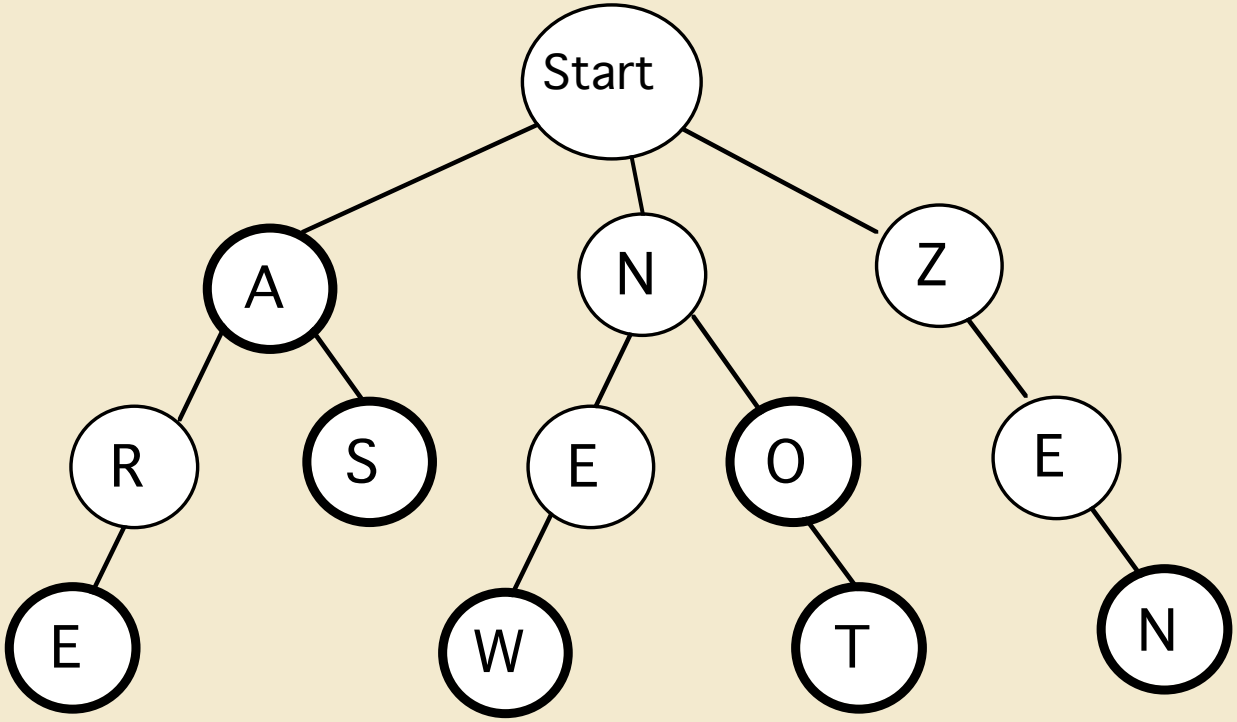
- Lab Goal: Build a data structure that can efficiently store and search a lexicon
- Using a special kind of tree called a *trie*



Lexicon Lab : Tries

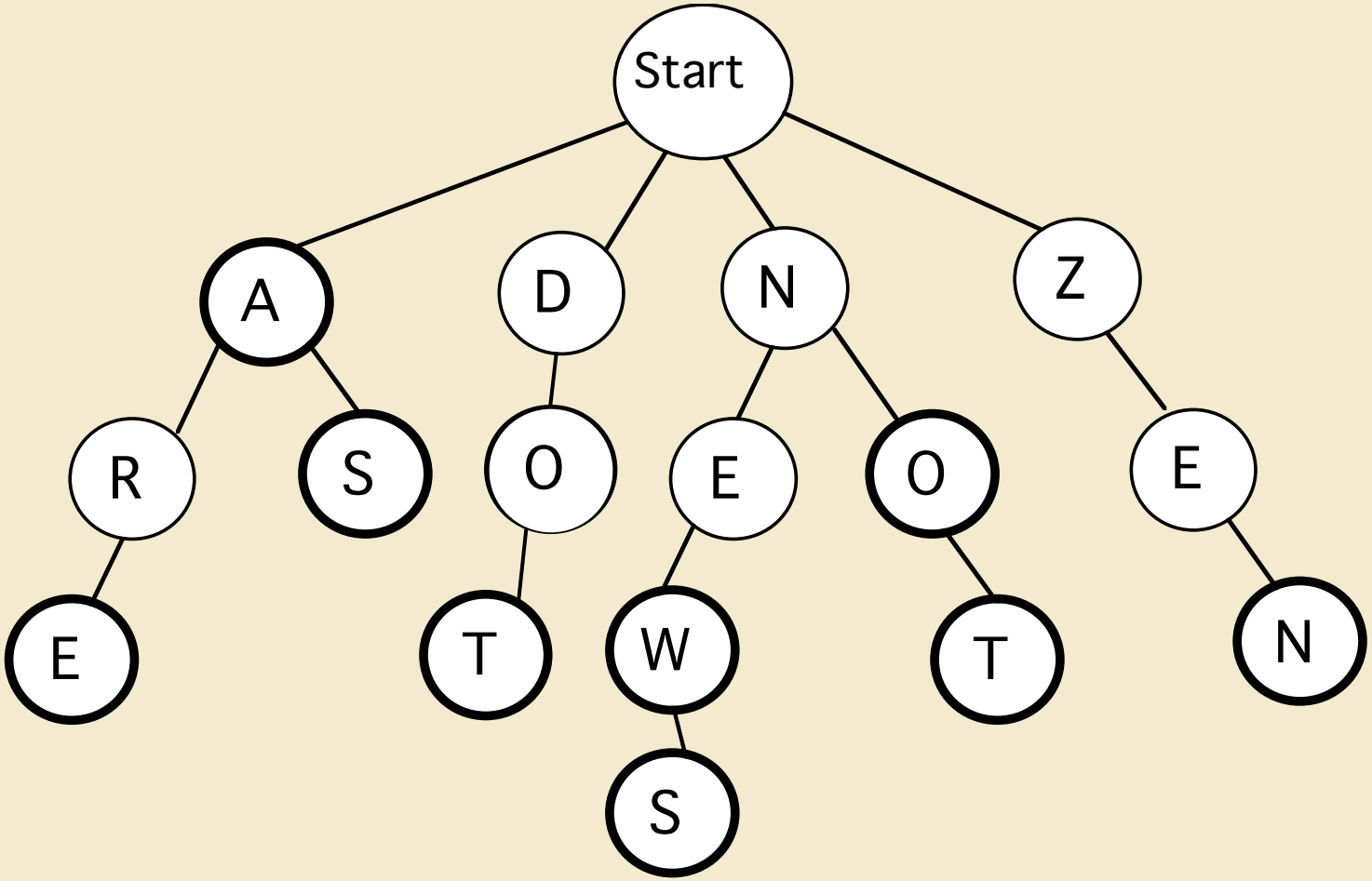
- A trie is a tree that stores words where
  - Each node holds a letter
  - Some nodes are "word" nodes (dark circles)
  - Any path from the root to a word node describes one of the stored words
  - All paths from the root form prefixes of stored words (a word is considered a prefix of itself)

Tries



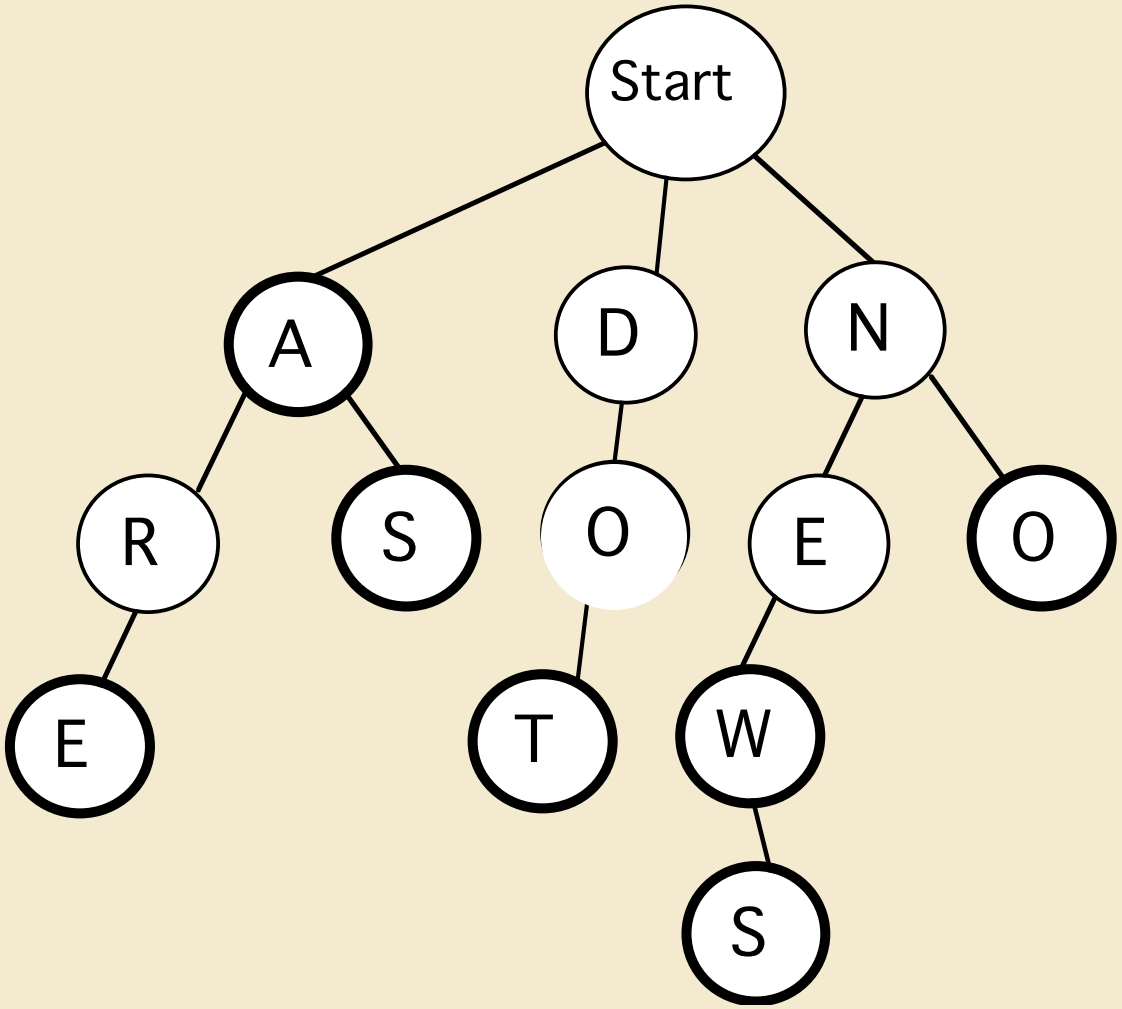
Now add “dot” and “news”

Tries



Now remove “not” and “zen”

Tries



The Lexicon Interface

An interface that provides the methods

```
public interface Lexicon {
    public boolean addWord(String word);
    public int addWordsFromFile(String filename);
    public boolean removeWord(String word);
    public int numWords();
    public boolean containsWord(String word);
    public boolean containsPrefix(String prefix);
    public Iterator<String> iterator();
    public Set<String> suggestCorrections(String
            target, int maxDistance);
    public Set<String> matchRegex(String pattern);
}
```

Lexicon Lab

- Implement the Lexicon interface using tries
  - LexiconTrie implements the Lexicon Interface
    - Each node of the Trie is a LexiconNode
    - Analogous to a SLL consisting of SLLNodes
  - LexiconTrie supports
    - adding/removing words
    - searching for words and prefixes
    - reading words from files
    - Iterating over all words
    - And, *optionally* (not required!)
      - spelling suggestions
      - wildcard searches

The Starter Repository

- Lexicon.java [Do not edit this file]
  - The lexicon interface specification
- LexiconTrie.java [Edit this file]
  - A skeleton implementation of trie structure that implements the Lexicon interface
- LexiconNode.java [Edit this file]
  - A skeleton implementation of the node structure for the LexiconTrie
- Main.java [No need to edit this file]
  - The app that will allow a user to make queries of your LexiconTrie
- inputs directory : three sample dictionaries
- The usual: PROBLEMS.md, README.md, ….

Tips

- Start by implementing LexiconNode.java
  - Add a main method to carefully test all of the LexiconNode methods
- Then implement LexiconTrie.java
  - Add a main method to carefully test all of the LexiconTrie methods
- Develop and test code *incrementally*
- Use the small dictionaries for initial debugging
  - You can easily determine what the output should be
- Several methods may be implemented recursively or iteratively. Think about what works best for you!