File I/O, Collections, & Tries

(on our way to Autocomplete)

File I/O Refresher

Distinguishing Between Two Models of Reading

- Readers (stream-based)
- Random Access Files

Distinguishing Between Two Models of Reading

- Readers (stream-based)
  - Data is *primarily* read from front to back, not much skipping around
  - Key methods:
    - close, mark, read, reset, skip
  - e.g. BufferedReader in Java

- Random Access Files
  - File cursor can be easily moved to arbitrary position in the file
  - Key methods:
    - close, read, seek, skip
  - e.g. RandomAccessFile in Java

Distinguishing Between Two Models of Reading

- Readers (stream-based)
  - Data is *primarily* read from front to back, not much skipping around
  - Key methods:
    - close, mark, read, reset, skip
  - e.g. BufferedReader in Java

- Random Access Files
  - File cursor can be easily moved to arbitrary position in the file
  - Key methods:
    - close, read, seek, skip
  - e.g. RandomAccessFile in Java

Supports moving the read position backwards?

Supports moving the read position forwards?

Supports buffered reading without providing your own array?

Distinguishing Between Two Models of Reading

- Readers (stream-based)
  - Data is *primarily* read from front to back, not much skipping around
  - Key methods:
    - close, mark, read, reset, skip
  - e.g. BufferedReader in Java

Supports moving the read position backwards (to a marked position)

Supports moving the read position forwards

Supports buffered reading without providing your own array

- Random Access Files
  - File cursor can be easily moved to arbitrary position in the file
  - Key methods:
    - close, read, seek, skip
  - e.g. RandomAccessFile in Java

Supports moving the read position backwards (to any position)

Supports moving the read position forwards

Problem Solving

- I have: a collection of people ordered by Penn ID, about 12GB large.
    - the entry on line 239281 is the person with Penn ID 239281
    - (like an array too big to fit in RAM)
- I need: to write a program that displays that person's role in the university when they sign in to PennInTouch

- Which class is more useful here? BufferedReader or RandomAccessFile?

Problem Solving

- I have: a collection of people ordered by Penn ID, about 12GB large.
  - the entry on line 239281 is the person with Penn ID 239281
  - (like an array too big to fit in RAM)
- I need: to write a program that displays that person's role in the university when they sign in to PennInTouch

- Should use a **RandomAccessFile**
  - You don't know the order that people will sign in to PiT, so you'll need to do a lot of skipping around to different lines
  - You may need to come back to different positions a bunch of different times

Problem Solving

- I have: a ~1GB file containing on each line...
  - A word
  - The frequency with which that word appears in some dataset
- I need: to read each of the entries in the file and add them to a data structure that I will keep in working memory

- Which class is more useful here? BufferedReader or RandomAccessFile?

Problem Solving

- I have: a ~1GB file containing on each line...
  - A word
  - The frequency with which that word appears in some dataset
- I need: to read each of the entries in the file and add them to a data structure that I will keep in working memory

- Should use a **BufferedReader**
  - The data fits in memory, so we just need to read everything once and come back to it
  - We're not strictly required to preserve the original order of the data, so we can just read the file start to finish, processing line by line

Speaking of line by line...

- From **BufferedReader.java**:

readLine

```
public String readLine()
                throws IOException
```

Reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.

Returns:

A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

(a favorite method from the times when we just used **Scanners** to read through files.)

What’s in a line?

```
5627187200   the
3395006400   of
2994418400   and
2595609600   to
1742063600   in
1176479700   i
1107331800   that
1007824500   was
```

- **r.readLine()** would return “5627187200 the”, e.g.
- How do you parse out the data contained in each line?

What’s in a line?

split

```
public String[] split(String regex)
```

Splits this string around matches of the given regular expression.

This method works as if by invoking the two-argument `split` method with the given expression and a limit argument of zero. Trailing empty strings are therefore not included in the resulting array.

The string `"boo:and:foo"`, for example, yields the following results with these expressions:

Regex	Result
:	`{ "boo", "and", "foo" }`
o	`{ "b", "", ":and:f" }`

“5627187200 the”.split(” “) → {“5627187200”, “the”}

(still have to take the Strings in the array to the right data type—parsing!)

Collections Refresher

Plenty of Useful Methods in java.util.Collections!

Name	Use
frequency(Collection c, Object o)	Count how often **o** appears in **c**
max(Collection c)/min(Collection c)	Find the largest/smallest element in **c** based on the natural ordering of elements
shuffle(List l)	Permute the elements in **l** randomly
sort(List l)	Sorts the list **l** into ascending order based on the natural ordering of the elements.

Sometimes you don't care about the "Natural Order"

Natural order (philosophy)

From Wikipedia, the free encyclopedia

Not to be confused with Ordo naturalis, a grouping of organisms formerly used in biology

This article **needs additional citations for verification**. Please help im
material may be challenged and removed.

Find sources: "Natural order" philosophy – news · newspapers · books · scholar · JST

In philosophy, the **natural order** is the moral source from which natural law seeks to derive its authority. I

Sometimes you don’t care about the “Natural Order”

Natural order (philosophy)

From Wikipedia, the free encyclopedia

Not to be confused with Ordo naturalis, a grouping of organisms formerly us… biology

This article **needs additional c**… imp…
material may be challenged and rem…
Find sources: "Natural order" philosophy – news … ks · scholar · JST…

In philosophy, the **natural order** is the moral source from which natural la… its authority. I…

This is a joke–
ignore this
definition

Comparator Versions

Name	Use
max(Collection c, Comparator comp)	Find the largest/smallest element in **c** based on the result of **comp**.compare()
sort(List l, Comparator comp)	Sorts the list **l** into ascending order based on the result of **comp**.compare()

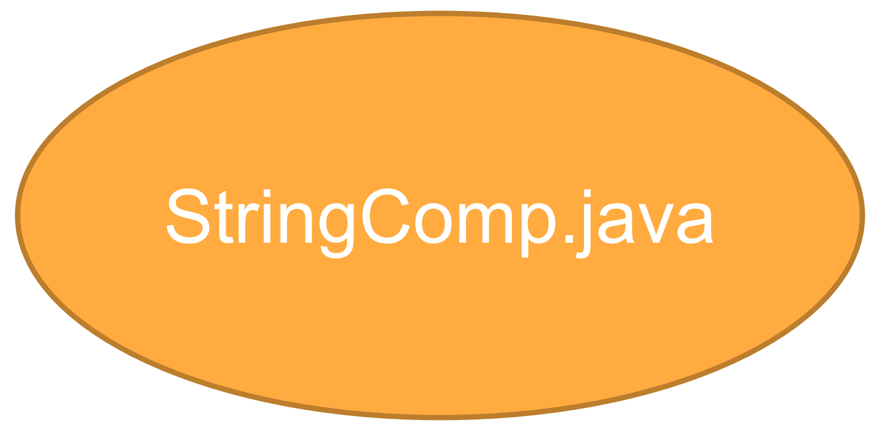
Reinventing the Natural Order: An Example

- Imagine we want to sort some Strings **first** by their lengths, and **then** alphabetically
  - (e.g. *book* comes before *antagonist* but after *been*)

- Strings already have their own **compareTo** and **equals** methods
  - We can’t and don’t want to override those
  - We might want to keep a bunch of options around: **STRATEGY pattern**

Reinventing the Natural Order: An Example

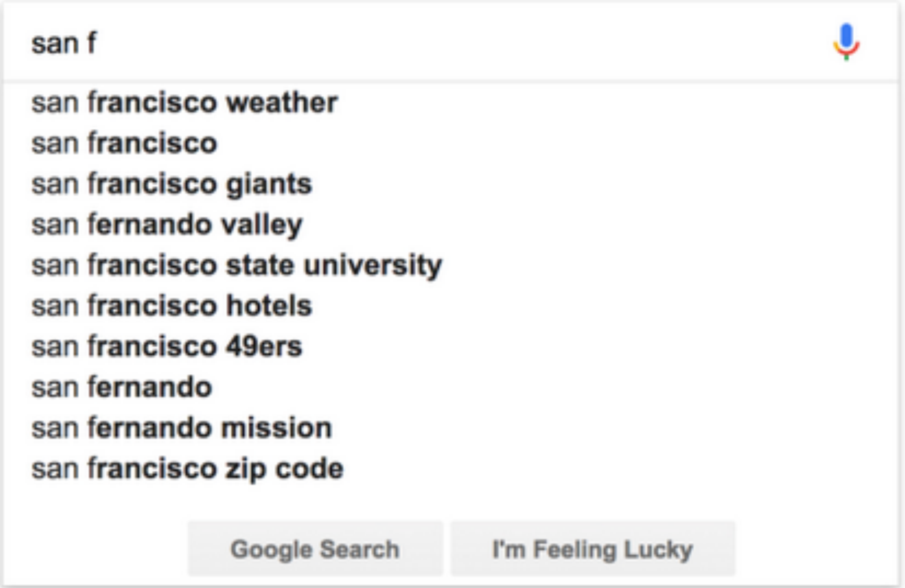
- Imagine we want to sort some Strings **first** by their lengths, and **then** alphabetically
  - (e.g. *book* comes before *antagonist* but after *been*)



Tries

Working towards Autocomplete

- Main idea: given a prefix of a String, suggest the most likely word that has that prefix



Working towards Autocomplete

- Main idea: given a prefix of a String, suggest the most likely word that has that prefix
- What we need:
  - A way of storing all possible words
  - A way of considering which words have a certain prefix
  - A way of associating a possible word with its likelihood of being the correct word

Using our Previous Data Structures

- Lists
  - Can store all candidate words
  - Tricky—but not impossible—to find words that have a certain prefix
  - No clean way of mapping words to likelihoods

Using our Previous Data Structures

- Lists
  - Can store all candidate words
  - Tricky—but not impossible—to find words that have a certain prefix
  - No clean way of mapping words to likelihoods
- Sets
  - Can store all candidate words
  - Tricky—but not impossible—to find words that have a certain prefix
  - No clean way of mapping words to likelihoods

Using our Previous Data Structures

- Lists
  - Can store all candidate words
  - Tricky—but not impossible—to find words that have a certain prefix
  - No clean way of mapping words to likelihoods
- Sets
  - Can store all candidate words
  - Tricky—but not impossible—to find words that have a certain prefix
  - No clean way of mapping words to likelihoods
- Hash Maps
  - Can store all candidate words and their likelihoods
  - Possibly even trickier to find words with a certain prefix

Using our Previous Data Structures

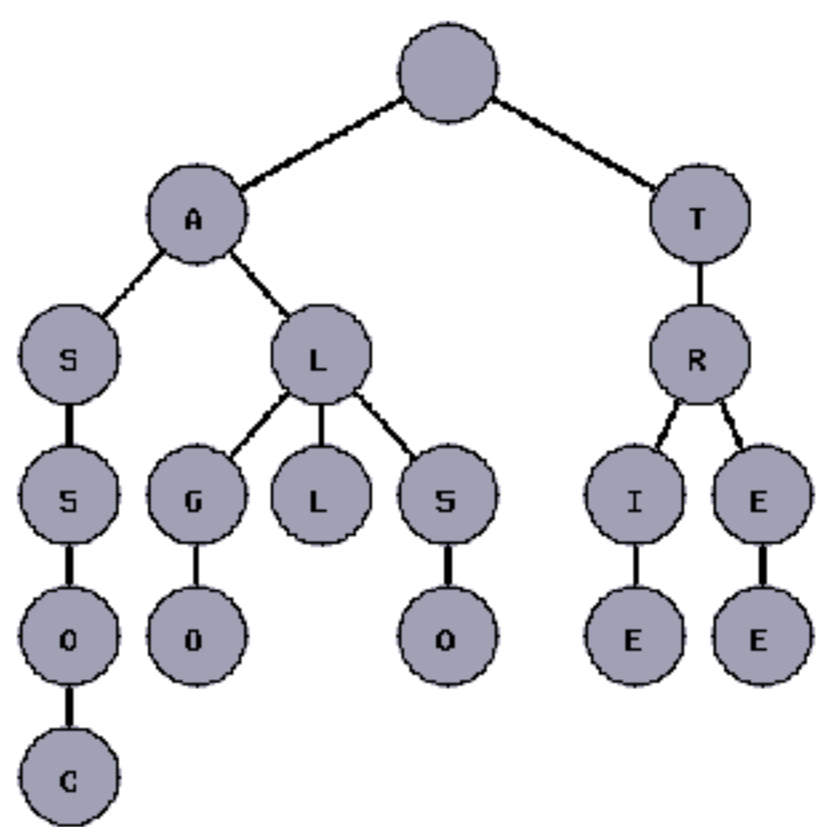
- Trees
    - Can store all candidate words
    - BSTs could be useful for finding words with a prefix…
    - Could store the likelihood of a word in the node containing that word

- Key sticking point: Still hard to find all words that have a certain prefix!!

We would like to do retrieval based on prefixes

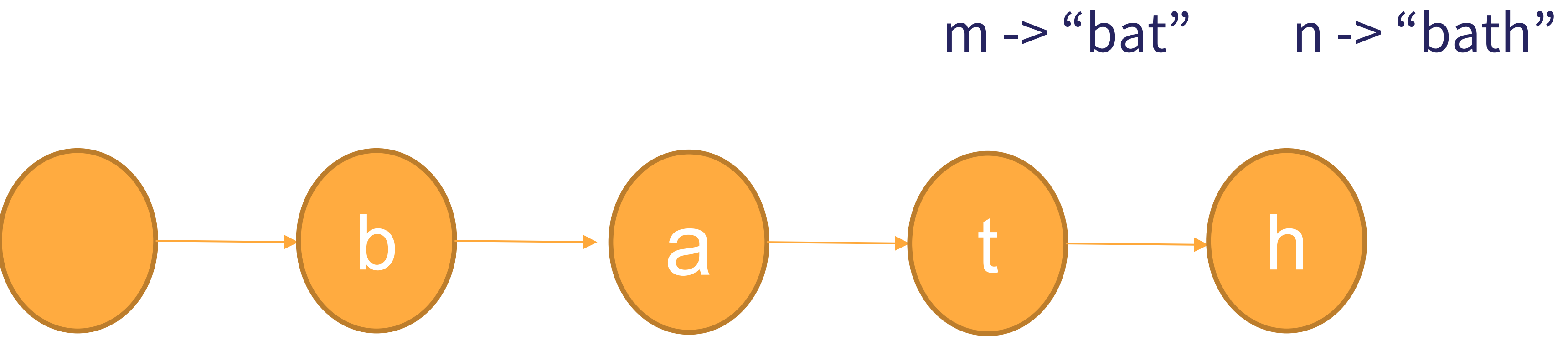
- Trie (pronounced try) is a tree-based data structure used to find words in a dictionary based on the prefixes of that word
    - Re**trie**val → Trie

- At right: the trie containing the words "assoc", "algo", "all", "also", "trie", and "tree"



The Trie Property

- Each node represents a prefix
- If node **n** is a descendent of node **m**, then the String that **m** represents is a prefix of the String that **n** represents.



Details about Tries

- Tries are 26-ary trees for English
    - 26 letters from A-Z
    - Not necessarily complete or full!
- Trie Nodes do not necessarily need to contain a char/String
    - This is an abstraction for the purposes of demonstration
    - Instead, the character contained in a Trie Node is determined implicitly from which child it is
        - Child 0 of node representing “gal” would itself represent “gala”
        - Child 25 of node representing “jaz” would itself represent “jazz”
    - The HW implementation will have you store the words in the nodes

Fields of a Trie Node (for Autocomplete)

- Term t
  - The word represented by this Node in the Trie, along with its corresponding weight
- int words
  - The number of words in this Tree represented by this current prefix **(always 0 or 1 for this assignment)**
- int prefixes
  - The number of words in this Tree that contain this current prefix
- Node[] references
  - An array of length 26, where position *i* contains a reference to the node representing this prefix with the *ith* character of the alphabet appended to the end

A Small Example:

- Class 10
- Clap 3
- Claps 2

HW 5: Autocomplete Me