CSE250, Spring 2022 Assignment 4 Due Sat. Apr. 9, 11:59pm

Lectures and Reading:

The textbook in Chapter 13 presents *priority queues* in the context of arrays or lists that are *sorted*. That is to say, the text is still in the context of "linear" data structures. This is fine as the main theme of the next three weeks is the motive for transiting to "nonlinear" data structures, in particular trees—which furnish the theoretically ultimate implementation of priority queues. Beyond priority queues, we will see how trees and (later) hash tables (last chapter in the text!) provide the main implementations of *associative lookup* for information storage and retrieval (and removal, one might add). Before reaching trees in chapter 16, we will cover graphs in Chapter 15—but not the array representation the text uses back in section 7.9, instead the standard inner-Node class representation it first used on page 421.

Thus the reading, after section 13.1 was this week: Chapter 15, picking up in section 15.3 where the Graph data structure is properly introduced. Note that GNode and GEdge are not defined until the very end of a lot of GUI code, on page 479. We will then go into sorting algorithms in the rest of chapter 15. The following week will get to Chapter 16.

——— *Assignment 4, due Apr. 9 "midnight stretchy" on CSE Autograder* ———

Task Statement:

We use a book of synonyms and antonyms by Samuel Fallows that (unlike *Roget's Thesaurus*) is in the public domain and provided as a plain text file `Fallows1898.txt`. Here is a section of its entries with some relevant features:

```
KEY: Rebuff \v.\.
SYN: Rebuke, repel, repulse, check, snub, oppose, [See REPEL].
=
KEY: Rebuff \n.\.
SYN: Rebuke, discouragement, repulsion, check.
ANT: Welcome, acceptance, encouragement.
=
KEY: Rebuke.
SYN: Reprove, chide, rebuff, reprimand, censure.
ANT: Approve, encourage, eulogize, applaud, incite.
=
KEY: Rebut.
SYN: Meet, retort, annul, confute.
ANT: Accept, sanction, recognize, confirm.
=
KEY: Recal.
SYN: Restore, reassemble, revoke, supersede, recollect, callback,
remember, cancel.
ANT: Relegate, dismiss, dispense, delegate, appoint, commission,
forget.
```

```
...

KEY: Repel.
SYN: Repulse, reject, refuse, deter, resist, check.
ANT: Promote, propel, welcome, advance, accept, encourage, further.
```

Yes, he spelled "recall" with only one 'l' and there are other archaisms such as "revolter" and "confute." Our attitude, as with some typos in the file as it appears at `https://www.gutenberg.org/files/51155/51155-h/51155-h.htm`, is to take it as-is. The only glitch I have fixed is 4 cases of "SYN:" not starting a line but being prefixed by an underscore meant to stand for a space. The copy of `Fallows1898` with this fix has a note at the top.] We will use only the synonym entries—note that some don't have antonyms.

The idea of the task is that logically the relation "Y is a synonym of X" ought to be symmetric, so that X would always be listed as a synonym of Y. But that is often not true in Fallows. For instance, while `rebuff` and `rebuke` list each other as synonyms, `repel` does not list `rebuff` back. Sometimes the lack of symmetry is because the listed word does not appear at all. Strange to say, there is no `KEY: Repulse` in the dictionary, only `Repulsive` which is a much different word. Sometimes the words appear but are cross-referenced in ways we will simply ignore:

```
KEY: Confute, [See REFUTE].
...
KEY: Discouragement, [See ENCOURAGE].
```

The task is: to find all cases where:

(i) Y is listed as a synonym of X and X is listed as a synonym of Y, so that symmetry holds; and

(ii) Y is listed as a synonym of X, and the dictionary does have a key entry for Y with a non-empty list of words (not just a cross-reference), but X does not appear in that list.

Output to a file `output.txt` in your project root folder the results on all words beginning with `Q`, in lines of the form:

```
quaff and drink are reciprocal synonyms
quaff lists swallow but swallow has a list of words without quaff
```

For this stage of the assignments, you are expected to use standard Scala classes rather than any of the "ISR" repo code. Make a simple class with fields `key` and `synList` for each entry (OK to take just those that have a nonempty synonym list). Read them into a Scala `Map[String, Set[String]]` or a `Map[String, List[String]]` object. Note that some words appear as different parts of speech. In order for `Map` to keep uniqueness, you may tag them in the manner of `rebuff_n` for noun, `rebuff_v` for verb, and `_a` for adjectives or adverbs. We will later discuss other ways to handle items with duplicate keys. It is fine for the tag to appear in your output. Use of any prior code for tokenizing etc. is optional—the file comes fairly well parsed already. This is again a single-file assignment, with required name `Synonyms.scala` and main being `object Synonyms extends App`. The code is worth 60 pts., breakdown to be specified—no essays or etc.