Frequently Asked Questions (WordNet)

Can I read the synset or hypernym file twice?

No. File I/O is very expensive; read each file only once and store it in an appropriate data structure.

Any advice on how to read and parse the synset and hypernym data files?

Use the `nextLine()` method in the Scanner library to read the data one line at a time. Use the `split()` method in Java's String library to divide a line into fields. Use either `Integer.parseInt()` to convert String id numbers into int values or `Integer.valueOf()` to convert into Integer objects.

Which data structure(s) should I use to store the synsets, synset ids, and hypernyms?

This part of the assignment is up to you. You must carefully select data structures to achieve the specified performance requirements.

What should `sca()` return if is there is a tie for the shortest common ancestor?

The API does not specify, so you are free to return any shortest common ancestor.

Should `nouns()` return each distinct noun once? Or multiple times if the noun appears in multiple synsets?

The API says to return the set of nouns, so no duplicates.

Should `nouns()` return the nouns in alphabetical order?

The API does not specify, so you are free to return them in any order.

Do I need to store the glosses? No, you won't use them on this assignment.

What is the root synset for the WordNet DAG?

```
38938,entity,that which is perceived or known or inferred to have its own
distinct existence (living or nonliving)
```

Can a noun appear in more than one synset?

Absolutely. It will appear once for each of the noun's distinct meanings. For example, the noun `word` appears in these 8 synsets:

```
36467,discussion give-and-take word,an exchange of views on some topic; "we
had a good discussion"; "we had a word or two about it"
57522,news intelligence tidings word,information about recent and important
events; "they awaited news of the outcome"
60202,parole word word_of_honor,a promise; "he gave his word"
60400,password watchword word parole countersign,a secret word or phrase
known only to a restricted group; "he forgot the password"
82510,word,a brief statement; "he didn't say a word about it"
82511,word,a string of bits stored in computer memory; "large computers use
words up to 64 bits long"
82512,word,a unit of language that native speakers can identify; "words are
the blocks from which sentences are made"; "he hardly said ten words all
morning"
82513,word,a verbal command for action; "when I give the word, charge!"
```

Do not assume that the number of synsets in which a noun participates is bounded by a constant. The noun `head` appears in 33 synsets.

Can a synset consist of exactly one noun?

Yes. Moreover, there can be several different synsets that consist of the same noun.

```
66,Aberdeen,a city in northeastern Scotland on the North Sea
67,Aberdeen,a town in northeastern Maryland
68,Aberdeen,a town in northeastern South Dakota
69,Aberdeen,a town in western Washington
```

I'm an ontologist and I noticed that your `hypernyms.txt` file contains both is-a and is-instance-of relationships.

Yes, you caught us. This ensures that every noun (except `entity`) has a hypernym.

Frequently Asked Questions (ShortestCommonAncestor)

Can I use my own Digraph class?

No. You must use `Digraph.java`. That is the type of the argument to the constructor.

How can I make the data type `ShortestCommonAncestor` immutable?

You can (and should) save the associated digraph in an instance variable. However, because Digraph is mutable, you must first make a defensive copy by calling the copy constructor in Digraph.

Is a vertex considered an ancestor of itself?

Yes.

What should `ancestor()` return if there is a tie for the shortest common ancestor?

The API does not specify, so you are free to return any shortest common ancestor.

I understand how to compute the `length()` method in $\Theta(E+V)$ time in the worst case but my `lengthSubset()` method takes $\Theta(a \times b \times (E+V))$ time, where a and b are the sizes of the two iterables. How can I improve it to be $\Theta(E+V)$ time?

The key is to use a multi-source version of breadth-first search, as in the constructor of `BreadthFirstDirectedPaths` that accepts an iterable of sources as an argument (instead of a single source).

Should I construct a new `ShortestCommonAncestor` object for each call to `sca()` and `distance()`?

No. You need only one `ShortestCommonAncestor` object per `WordNet` object. The methods `sca()` and `distance()` should make one call to either `lengthSubset()` or `ancestorSubset()`.

In Additional Performance Requirements, do `length()`, `lengthSubset()`, `ancestor()`, and `ancestorSubset()` need to take time proportional to the number of vertices and edges reachable from the argument vertices in the worst case? Or, may I use hashing?

You can make standard technical assumptions (such as the uniform hashing assumption). If you do so, state any assumptions that you make in your `readme.txt` file.

Should I re-implement breadth-first search to compute shortest common ancestors?

Unless you are attempting the extra credit, you should use `BreadthFirstDirectedPaths.java`. To earn the extra credit, however, you'll need to re-implement breadth-first search.

Should I re-implement depth-first search to find directed cycles?

No. Instead, use `DirectedCycle.java`.

Do I need to throw exceptions explicitly with a throw statement?

No, it's fine if they are thrown implicitly. For example, you can rely on any method in `Digraph.java` to throw an `IllegalArgumentException` if passed a vertex argument outside of the prescribed range. A good API documents the requisite behavior for all possible arguments but, hopefully, you should not need much extra code to deal with these corner cases.

Frequently Asked Questions (Outcast)

What should `outcast()` return if there is a tie for the outcast?

The API does not specify, so you are free to return any outcast.

My algorithm computes the distance between every pair of nouns. Is that okay?

Yes, that's fine.

Input, Output, and Testing

Some examples. Here are some interesting examples that you can use to test your code.

- The noun `President` (in capitalized form) appears in two different synsets:

```
14479,President_of_the_United_States President Chief_Executive,the
office of the United States ....
14480,President_of_the_United_States United_States_President President
Chief_Executive,the person who holds the office .....
```

- The synset `municipality` has two paths to `region`:

```
municipality -> administrative_district -> district -> region
municipality -> populated_area -> geographic_area -> region
```

- The synsets `individual` and `edible_fruit` have several different paths to their common ancestor `physical_entity`:

```
individual -> organism being -> living_thing animate_thing -> whole
unit -> object physical_object -> physical_entity
person individual someone somebody mortal soul -> causal_agent cause
causal_agency -> physical_entity
edible_fruit -> garden_truck -> food solid_food -> solid -> matter ->
physical_entity
```

```
edible_fruit -> fruit -> reproductive_structure -> plant_organ ->
plant_part -> natural_object -> unit -> object -> physical_entity
```

- The following pairs of nouns are very far apart:

```
(distance = 23) white_marlin, mileage
(distance = 33) Black_Plague, black_marlin
(distance = 27) American_water_spaniel, histology
(distance = 29) Brown_Swiss, barrel_roll
```

- The following synset has many paths to `entity`:

```
Ambrose Saint_Ambrose St._Ambrose
```

- Also, we encourage you to use the small collection of sample files provided.

Possible Progress Steps

- Create the data type `ShortestCommonAncestor`. First, think carefully about designing a correct and efficient algorithm for computing the shortest common ancestor. In addition to the digraph*.txt files, design small rooted DAGs to test and debug your code. Modularize by sharing common code.
- Add code to `ShortestCommonAncestor` to detect whether a digraph is a rooted DAG. As defined in the assignment, a digraph is a rooted DAG if it is acyclic and has one vertex—the root—that is an ancestor of every other vertex.
- Read and parse the files described in the assignment, `synsets.txt` and `hypernyms.txt`. Do not worry about storing the data in any data structures yet. Test that you are parsing the input correctly before proceeding.
- Create a data type `WordNet`. Divide the constructor into two (or more) subtasks (private methods).
    - Read the synsets file and build appropriate data structures. The file `synsets.txt` contains 83,127 synsets, composed from 120,119 nouns. Do not hardwire either of these numbers; your program must work for any valid synset file. Record the number of synsets for use when constructing the underlying digraph from the hypernyms file.
    - Read the hypernyms file and build a Digraph. The file `hypernyms.txt` corresponds to a rooted DAG with 83,127 vertices and 85,441 edges. Do not hardwire either of these numbers; your program must work for any valid hypernym file.
- Implement the remaining `WordNet` methods.

- Implement `Outcast`. This should be relatively straightforward by calling the appropriate methods from the `WordNet` data type.