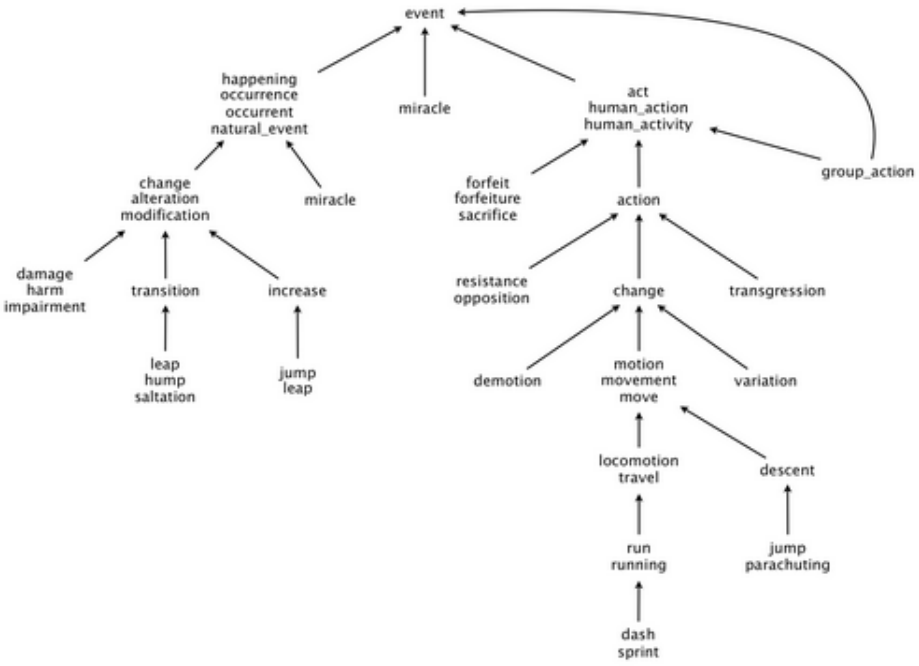
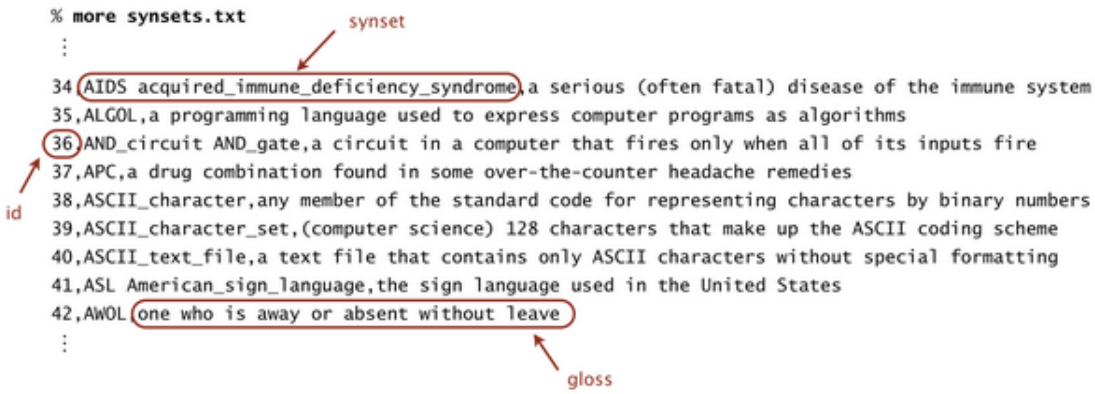
WordNet® is a semantic lexicon for the English language that computational linguists and cognitive scientists use extensively. For example, WordNet was a key component in IBM's Jeopardy-playing Watson® computer system. WordNet groups words into sets of synonyms called synsets. For example, {*AND circuit, AND gate*} is a synset that represent a logical gate that fires only when all of its inputs fire. WordNet also describes semantic relationships between synsets. One such relationship is the *is-a* relationship, which connects a *hyponym* (more specific synset) to a *hypernym* (more general synset). For example, the synset {*gate, logic gate*} is a hypernym of {*AND circuit, AND gate*} because an AND gate is a kind of logic gate.

The WordNet Digraph Your first task is to build the WordNet digraph: each vertex v is an integer that represents a synset, and each directed edge $v \to w$ represents that w is a hypernym of v. The WordNet digraph is a *rooted DAG*: it is acyclic and has one vertex — the root — that is an ancestor of every other vertex. However, it is not necessarily a tree because a synset can have more than one hypernym. A small subgraph of the WordNet digraph appears below.



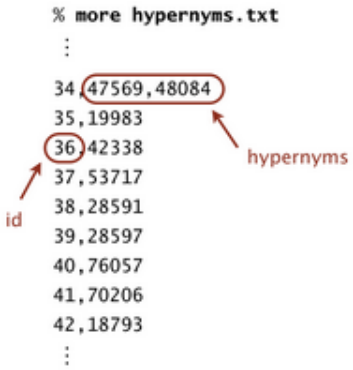
The WordNet Input File Formats We now describe the two data files that you will use to create the WordNet digraph. The files are in *comma-separated values* (CSV) format: each line contains a sequence of fields, separated by commas.

- *List of synsets.* The file `synsets.txt` contains all noun synsets in WordNet, one per line. Line i of the file (counting from 0) contains the information for synset i. The first field is the *synset id*, which is always the integer i; the second field is the synonym set (or synset); and the third field is its dictionary definition (or *gloss*), which is not relevant to this assignment.



For example, line 36 means that the synset `AND_circuit AND_gate` has an id number of 36 and it's gloss is a circuit in a computer that fires only when all of its inputs fire. The individual nouns that constitute a synset are separated by spaces. If a noun contains more than one word, it uses the underscore character to connect the words (and not the space character).

- *List of hypernyms.* The file `hypernyms.txt` contains the hypernym relationships. Line i of the file contains the hypernyms of synset i. The first field is the synset id, which is always the integer i; subsequent fields are the id numbers of the synset's hypernyms.



For example, line 36 means that synset 36 (`AND_circuit AND_Gate`) has 42338 (`gate logic_gate`) as it only hypernym. Line 34 means that synset 34 (`AIDS acquired_immune_deficiency_syndrome`) has two hypernyms: 47569 (`immunodeficiency`) and 56099 (`infectious_disease`).

Problem 1. (*`WordNet` Data Type*) Implement an immutable data type `WordNet` with the following API:

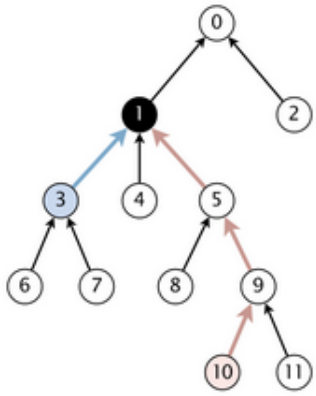
method	description
`public WordNet(String synsets, String hypernyms)`	construct `WordNet` object given the names of the input (synset and hypernym) files
`public Iterable<String> nouns()`	all WordNet nouns
`public boolean isNoun(String word)`	is the word a WordNet noun?
`public String sca(String noun1, String noun2)`	a synset (second field of `synsets.txt`) that is a shortest common ancestor of noun1 and noun2 (defined below)
`public int distance(String noun1, String noun2)`	distance between noun1 and noun2 (defined below)

Corner cases. All methods and the constructor should throw a `java.lang.NullPointerException` if any argument is `null`. The `distance()` and `sca()` methods should throw a `java.lang.IllegalArgumentException` unless both of the noun arguments are WordNet nouns. You may assume that the input files are in the specified format (and that the underlying digraph is a rooted DAG).

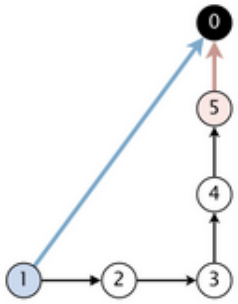
Performance requirements. Your data type should use space linear in the input size (size of synsets and hypernyms files). The constructor should take time linearithmic (or better) in the input size. The method `isNoun()` should run in time logarithmic (or better) in the number of nouns. The methods `distance()` and `sca()` should make exactly one call to the `length()` and `ancestor()` methods in `ShortestCommonAncestor`, respectively. For the analysis, assume that the number of characters in a noun or synset is bounded by a constant.

```
$ java WordNet data/synsets.txt data/hypernyms.txt worm bird
# of nouns = 119188
isNoun(worm) = true
isNoun(bird) = true
isNoun(worm bird) = false
sca(worm, bird) = animal animate_being beast brute creature fauna
distance(worm, bird) = 5
```

Shortest Common Ancestor An *ancestral path* between two vertices v and w in a rooted DAG is a directed path from v to a common ancestor x, together with a directed path from w to the same ancestor x. A shortest ancestral path is an ancestral path of minimum total length. We refer to the common ancestor in a shortest ancestral path as a *shortest common ancestor*. Note that a shortest common ancestor always exists because the root is an ancestor of every vertex. Note also that an ancestral path is a path, but not a directed path.

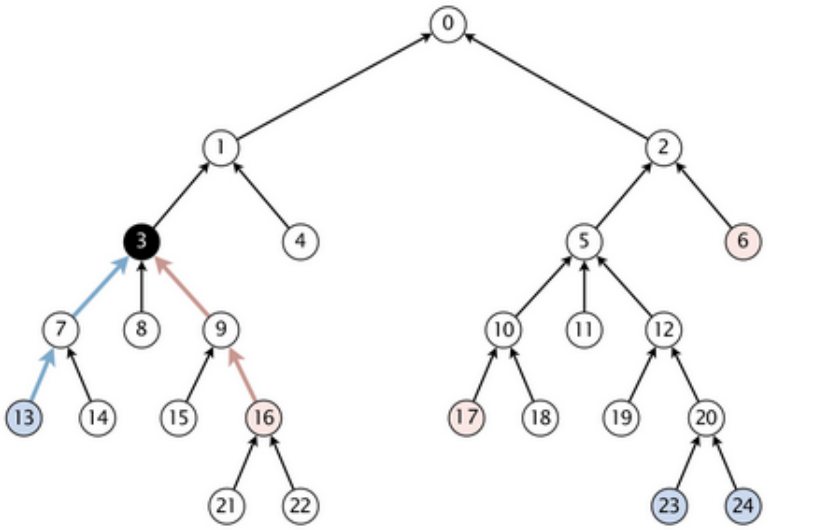


v = 3, w = 10
shortest ancestral path: 3–1–5–9–10
associated length: 4
shortest common ancestor: 1



v = 1, w = 5
ancestral path: 1–2–3–4–5
shortest ancestral path: 1–0–5
associated length: 2
shortest common ancestor: 0

We generalize the notion of shortest common ancestor to subsets of vertices. A shortest ancestral path of two subsets of vertices A and B is a shortest ancestral path over all pairs of vertices v and w, with v in A and w in B.

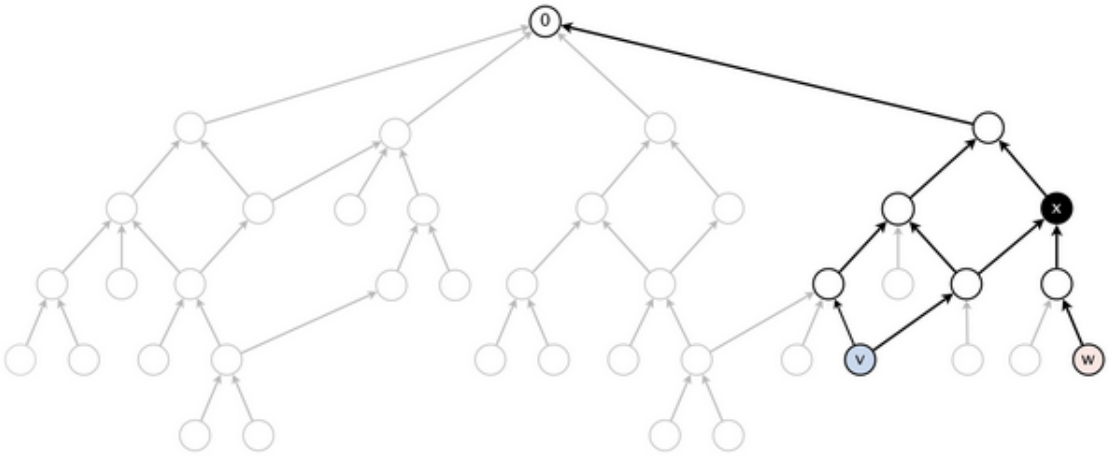


Problem 2. (*ShortestCommonAncestor Data Type*) Implement an immutable data type `ShortestCommonAncestor` with the following API:

method	description
`public ShortestCommonAncestor(Digraph G)`	construct a `ShortestCommonAncestor` object given a rooted DAG
`public int length(int v, int w)`	length of shortest ancestral path between v and w
`public int ancestor(int v, int w)`	a shortest common ancestor of vertices v and w
`public int length(Iterable<Integer> subsetA, Iterable<Integer> subsetB)`	length of shortest ancestral path of vertex subsets A and B
`public int ancestor(Iterable<Integer> subsetA, Iterable<Integer> subsetB)`	shortest common ancestor of vertex subsets A and B

Corner cases. All methods and the constructor should throw a `java.lang.NullPointerException` if any argument is `null`. The constructor should throw a `java.lang.IllegalArgumentException` if the digraph is not a rooted DAG. The methods `length()` and `ancestor()` should throw a `java.lang.IndexOutOfBoundsException` if any argument vertex is invalid and an `java.lang.IllegalArgumentException` if any iterable argument contains zero vertices.

Performance requirements. Your data type should use space proportional to $E+V$, where E and V are the number of edges and vertices in the digraph, respectively. All methods and the constructor should take time proportional to $E+V$ (or better). You will receive most of the credit for meeting these basic requirements. In addition, for full credit, the methods `length()` and `ancestor()` should take time proportional to the number of vertices and edges reachable from the argument vertices (or better), For example, to compute the shortest common ancestor of v and w in the digraph below, your algorithm can examine only the highlighted vertices and edges and it cannot initialize any vertex-indexed arrays.



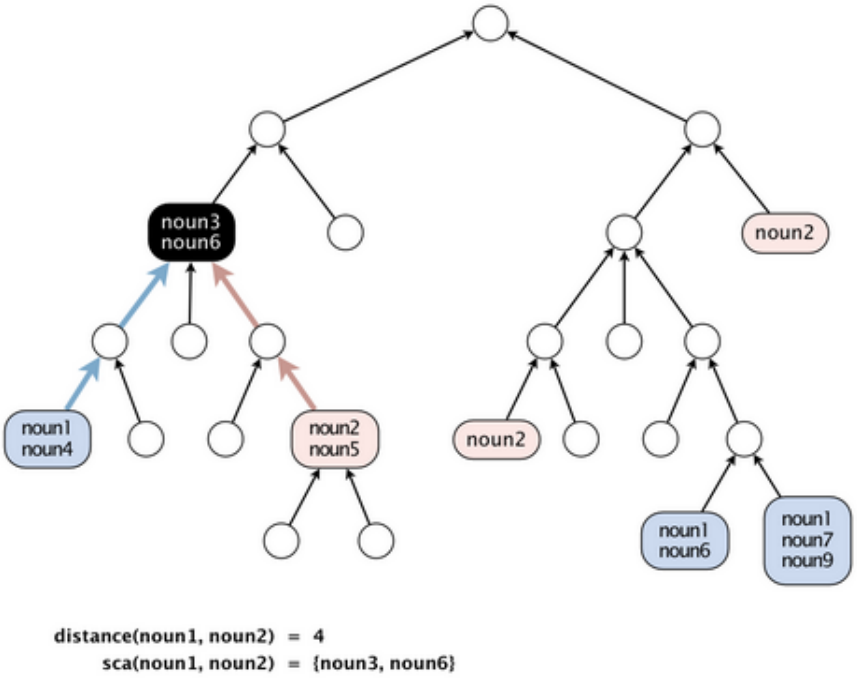

```
$ java ShortestCommonAncestor data/digraph1.txt
3 10 8 11 6 2
<ctrl-d>
length = 4, ancestor = 1
length = 3, ancestor = 5
length = 4, ancestor = 0
```

Measuring the Semantic Relatedness of Two Nouns Semantic relatedness refers to the degree to which two concepts are related. Measuring semantic relatedness is a challenging problem. For example, you consider *George W. Bush* and *John F. Kennedy* (two U.S. presidents) to be more closely related than *George W. Bush* and *chimpanzee* (two primates). It might not be clear whether *George W. Bush* and *Eric Arthur Blair* are more related than two arbitrary people. However, both *George W. Bush* and *Eric Arthur Blair* (aka George Orwell) are famous communicators and, therefore, closely related.

We define the semantic relatedness of two WordNet nouns x and y as follows:

- A is set of synsets in which x appears
- B is set of synsets in which y appears
- $sca(x, y)$ a shortest common ancestor of subsets A and B
- $distance(x, y)$ is length of shortest ancestral path of subsets A and B

This is the notion of distance that you will use to implement the `distance()` and `sca()` methods in the `WordNet` data type.



Outcast Detection Given a list of WordNet nouns $x_1, x_2, \ldots, x_n$, which noun is the least related to the others? To identify an outcast, compute the sum of the distances between each noun and every other one:

$$d_i = distance(x_i, x_1) + distance(x_i, x_2) + \cdots + distance(x_i, x_n)$$

and return a noun x_t for which d_t is maximum. Note that because $distance(x_i, x_i) = 0$, it will not contribute to the sum.

Problem 3. (*`Outcast` Data Type*) Implement an immutable data type `Outcast` with the following API:

method	description
`public Outcast(WordNet wordnet)`	construct an `Outcast` object given a `WordNet` object
`public String outcast(String[] nouns)`	the outcast noun from nouns

You may assume that argument to `outcast()` contains only valid WordNet nouns (and that it contains at least two such nouns).

```
$ java Outcast data/synsets.txt data/hypernyms.txt data/outcast5.txt data/outcast8.txt data/outcast11.txt
outcast(data/outcast5.txt) = table
outcast(data/outcast8.txt) = bed
outcast(data/outcast11.txt) = potato
```

Data Under the `data` directory, we provide several sample input files for testing.

Files to Submit:

1. `WordNet.java`
2. `ShortestCommonAncestor.java`
3. `Outcast.java`
4. `report.txt`

Before you submit:

- Make sure your programs meet the input and output specifications by running the following command on the terminal:

```
$ python run_tests.py -v [<problems>]
```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test; all the problems are tested if no argument is given.

- Make sure your programs meet the style requirements by running the following command on the terminal:

```
$ check_style <program>
```

  where `<program>` is the `.java` file whose style you want to check.

Acknowledgements This project is an adaptation of the WordNet assignment developed at Princeton University by Alina Ene and Kevin Wayne.