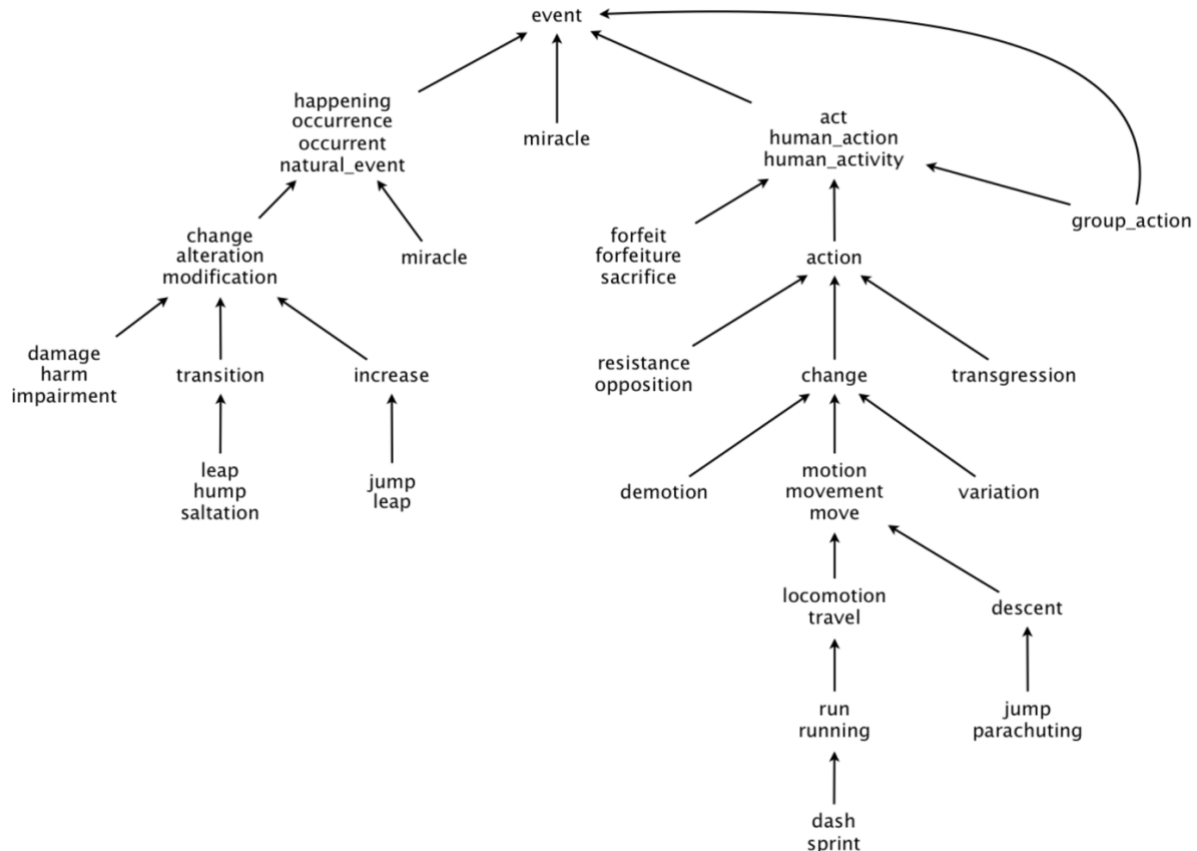


Programming Assignment 4

Make sure you use proper style when you name variables/constants and align statements. Do not forget to put your name in the top comments of each of your programs. The format of your programs is very important when learning a programming language and bad style may be penalized with up to 30%. **Please use object-oriented programming style and DO NOT use the standard libraries from the book site.**

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 $v \rightarrow w$ 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. Here is a small subgraph of the WordNet digraph:



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.

```
% more synsets.txt
:
34,AIDS acquired_immune_deficiency_syndrome,a serious (often fatal) disease of the immune system
35,ALGOL,a programming language used to express computer programs as algorithms
36,AND_circuit AND_gate,a circuit in a computer that fires only when all of its inputs fire
37,APC,a drug combination found in some over-the-counter headache remedies
38,ASCII_character,any member of the standard code for representing characters by binary numbers
39,ASCII_character_set,(computer science) 128 characters that make up the ASCII coding scheme
40,ASCII_text_file,a text file that contains only ASCII characters without special formatting
41,ASL American_sign_language,the sign language used in the United States
42,AWOL,one who is away or absent without leave
:
```

Diagram annotations: An arrow labeled "id" points to the number 36 in line 36. An arrow labeled "synset" points to the text "AND_circuit AND_gate" in line 36. An arrow labeled "gloss" points to the text "a circuit in a computer that fires only when all of its inputs fire" in line 36.

For example, line 36 means that the synset { AND_circuit, AND_gate } has an id number of 36 and its 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, the underscore character connects the words (and not the space character).

- List of hypernyms. The file `hypernyms.txt` contains the hypernym relationships. Line i of the file (counting from 0) 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.

```
% more hypernyms.txt
:
34,48504,49019
35,20918
36,43273
37,54652
38,29526
39,29532
40,76992
41,71141
42,19728
:
```

Diagram annotations: An arrow labeled "id" points to the number 36 in line 36. An arrow labeled "hypernyms" points to the text "43273" in line 36.

For example, line 36 means that synset 36 (AND_circuit AND_gate) has 43273 (gate logic_gate) as its only hypernym. Line 34 means that synset 34 (AIDS acquired_immune_deficiency_syndrome) has two hypernyms: 48504 (immunodeficiency) and 49019 (infectious_disease).

WordNet data type. Implement an immutable data type WordNet with the following API:

```
public class WordNet {
```

```

// constructor takes the name of the two input files
public WordNet(String synsets, String hypernyms)

// the set of all WordNet nouns
public Iterable<String> nouns()

// is the word a WordNet noun?
public boolean isNoun(String word)

// a synset (second field of synsets.txt) that is a shortest common ancestor
// of noun1 and noun2 (defined below)
public String sca(String noun1, String noun2)

// distance between noun1 and noun2 (defined below)
public int distance(String noun1, String noun2)

// unit testing (required)
public static void main(String[] args)
}

```

Corner cases. Throw an `IllegalArgumentException` in the following situations:

- Any argument to the constructor or an instance method is null
- Any of the noun arguments in `distance()` or `sca()` is not a WordNet noun.

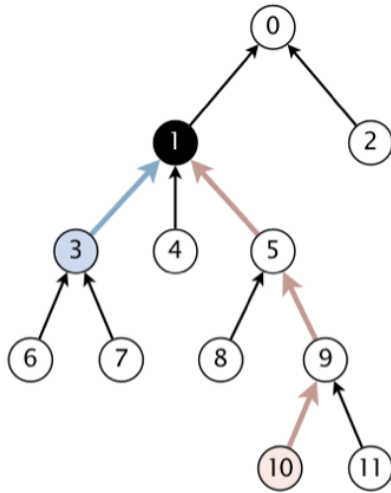
You may assume that the input files are in the specified format and that the underlying digraph is a rooted DAG.

Unit testing. Your `main()` method must call each public constructor and method directly and help verify that they work as prescribed (e.g., by printing results to standard output).

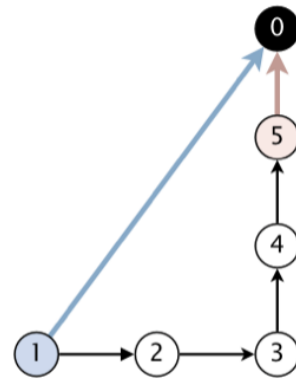
Performance requirements. Your implementation must achieve the following performance requirements. In the requirements below, assume that the number of characters in a noun or synset is bounded by a constant.

- Your data type must use space linear in the input size (size of `synsets` and `hypernyms` files).
- The constructor must take time linearithmic (or better) in the input size.
- The method `isNoun()` must run in time logarithmic (or better) in the number of nouns.
- The methods `distance()` and `sca()` must make exactly one call to the `lengthSubset()` and `ancestorSubset()` methods in `ShortestCommonAncestor`, respectively.

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.

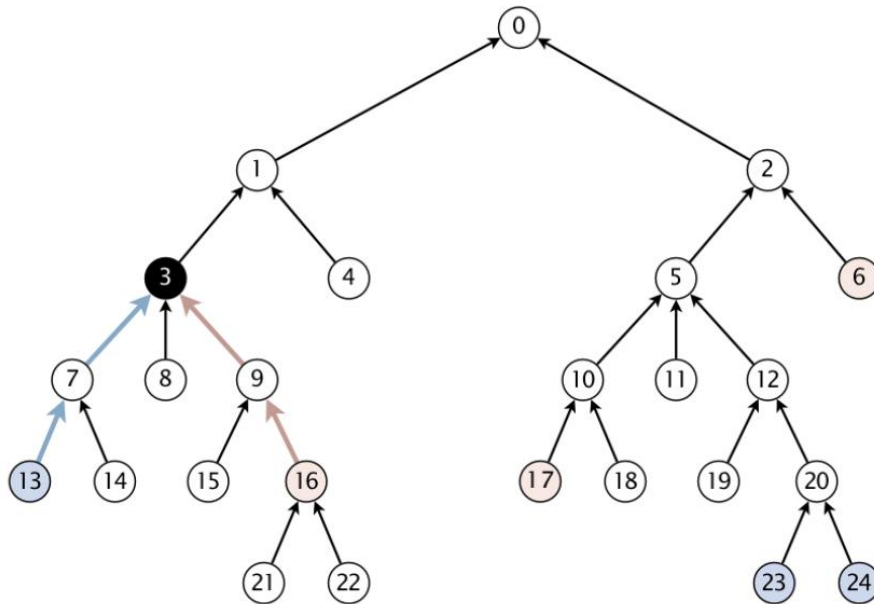


$v = 3, w = 10$
shortest ancestral path: 3-1-5-9-10
associated length: 4
shortest common ancestor: 1



$v = 1, w = 5$
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 among all pairs of vertices v and w , with v in A and w in B . As an example, the following figure (digraph25.txt) identifies several (but not all) ancestral paths between the red and blue vertices, including the shortest one.



$A = \{ 13, 23, 24 \}, B = \{ 6, 16, 17 \}$
ancestral path: 13-7-3-1-0-2-6
ancestral path: 23-20-12-5-10-17
ancestral path: 23-20-12-5-2-6

shortest ancestral path: 13-7-3-9-16
associated length: 4
shortest common ancestor: 3

Shortest common ancestor data type. Implement an immutable data type `ShortestCommonAncestor` with the following API:

```
public class ShortestCommonAncestor {  
  
    // constructor takes a rooted DAG as argument  
    public ShortestCommonAncestor(Digraph G)  
  
    // length of shortest ancestral path between v and w  
    public int length(int v, int w)  
  
    // a shortest common ancestor of vertices v and w  
    public int ancestor(int v, int w)  
  
    // length of shortest ancestral path of vertex subsets A and B  
    public int lengthSubset(Iterable<Integer> subsetA, Iterable<Integer> subsetB)  
  
    // a shortest common ancestor of vertex subsets A and B  
    public int ancestorSubset(Iterable<Integer> subsetA, Iterable<Integer> subsetB)  
  
    // unit testing (required)  
    public static void main(String[] args)  
  
}
```

Corner cases. Throw an `IllegalArgumentException` in the following situations:

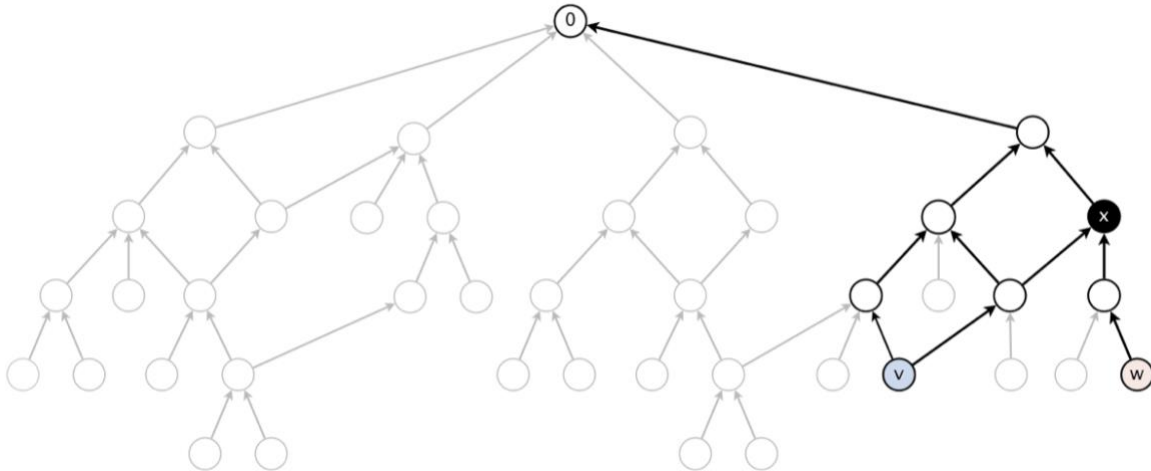
- The argument to the constructor is not a rooted DAG
- Any argument is null
- Any vertex argument is outside its prescribed range
- Any iterable argument contains zero vertices
- Any iterable argument contains a null item

Unit testing. Your `main()` method must call each public constructor and method directly and help verify that they work as prescribed (e.g., by printing results to standard output).

Basic performance requirements. Your implementation must achieve the following worst-case performance requirements, where E and V are the number of edges and vertices in the digraph, respectively.

- Your data type must use $O(E+V)$ space.
- All methods and the constructor must take $O(E+V)$ time.

Additional performance requirements. The methods `length()`, `lengthSubset()`, `ancestor()`, and `ancestorSubset()` must 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 following digraph, your algorithm can examine only the highlighted vertices and edges; it cannot initialize any vertex-indexed arrays.



Test client. The following test client takes the name of a digraph input file as an argument; creates the digraph; reads vertex pairs from standard input; and prints the length of the shortest ancestral path between the two vertices, along with a shortest common ancestor:

```
public static void main(String[] args) {
    File inFile = new File("digraph1.txt");
    Scanner input = new Scanner(inFile);
    Digraph G = new Digraph(input);

    ShortestCommonAncestor sca = new ShortestCommonAncestor(G);
    Scanner input1 = new Scanner(System.in);
    while (input1.hasNext()) {
        int v = Input1.nextInt();
        int w = Input1.nextInt();
        int length = sca.length(v, w);
        int ancestor = sca.ancestor(v, w);
        System.out.printf("length = %d, ancestor = %d\n", length, ancestor);
    }
}
```

Here is a sample execution:

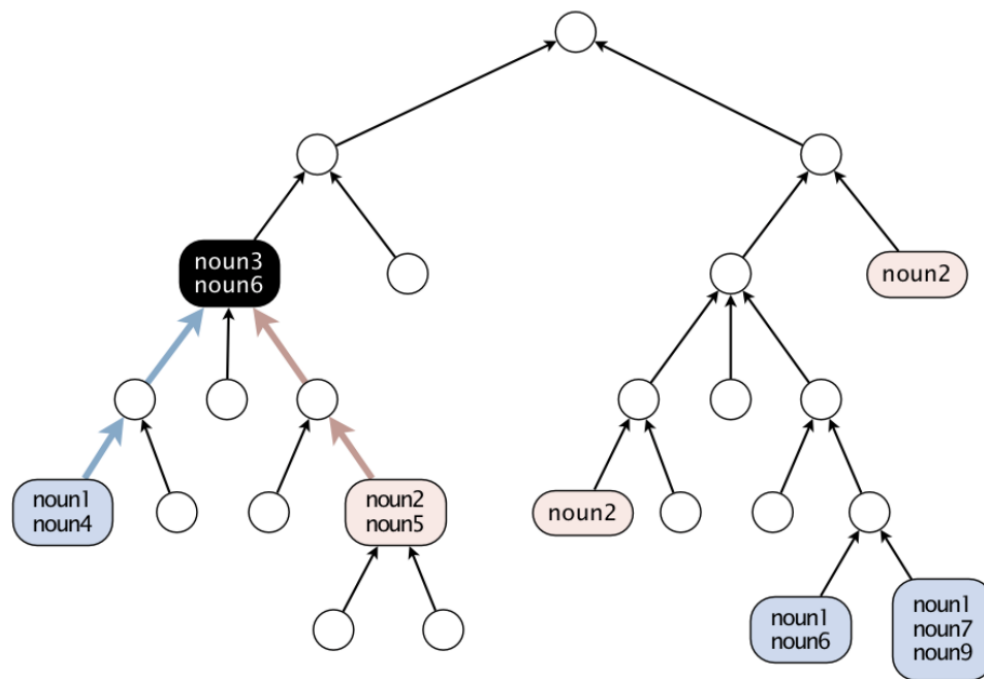
digraph1.txt	digraph1.txt:
12	3 10
11	length = 4, ancestor = 1
6 3	8 11
7 3	length = 3, ancestor = 5
3 1	6 2
4 1	length = 4, ancestor = 0
5 1	
8 5	
9 5	
10 9	
11 9	
1 0	

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 (a.k.a. George Orwell) are famous communicators and, therefore, closely related.

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

- A = set of synsets in which x appears
- B = set of synsets in which y appears
- $\text{distance}(x, y)$ = length of shortest ancestral path of subsets A and B
- $\text{sca}(x, y)$ = a shortest common ancestor 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.



`distance(noun1, noun2) = 4`
`sca(noun1, noun2) = {noun3, noun6}`

Outcast detection. Given a list of WordNet nouns x_1, x_2, \dots, 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:

$$d_i = \text{distance}(x_i, x_1) + \text{distance}(x_i, x_2) + \dots + \text{distance}(x_i, x_n)$$

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

Implement an immutable data type Outcast with the following API:

```

public class Outcast {

    // constructor takes a WordNet object
    public Outcast(WordNet wordnet)

    // given an array of WordNet nouns, return an outcast
    public String outcast(String[] nouns)

    // test client (see below)
    public static void main(String[] args)

}

```

Corner cases. Assume that the argument to `outcast()` contains only valid WordNet nouns and that it contains at least two such nouns.

Test client. The following test client takes the name of a synset file, the name of a hypernym file, followed by the names of outcast files, and prints an outcast in each file:

```

public static void main(String[] args) {
    WordNet wordnet = new WordNet("synsets.txt", "hypernyms.txt");
    Outcast outcast = new Outcast(wordnet);

    for (int t = 0; t < 3; t++) {
        Scanner input = Scanner(System.in);
        System.out.print("Enter the file name: ");
        String fileName = input.nextLine();
        System.out.print("Enter the file size: ");
        int fileSize = input.nextInt();

        File inFile = new File(fileName);
        Scanner input1 = new Scanner(inFile);
        String[] nouns = new String[fileSize];

        int i=0;
        while(input1.hasNextLine()){
            nouns[i] = input1.nextLine();
            i++;
        }

        System.out.println(fileName + ": " + outcast.outcast(nouns));
    }
}

```

Here is a sample execution:

```

outcast5.txt
horse zebra cat bear table

outcast8.txt
water soda bed orange_juice milk apple_juice tea coffee

outcast11.txt
apple pear peach banana lime lemon blueberry strawberry mango watermelon potato

synsets.txt hypernyms.txt outcast5.txt outcast8.txt outcast11.txt
Enter the file name: outcast5.txt

```



```
Enter the file size: 5
outcast5.txt: table
Enter the file name: outcast8.txt
Enter the file size: 8
outcast8.txt: bed
Enter the file name: outcast11.txt
Enter the file size: 11
outcast11.txt: potato
```

Analysis of running time. Analyze the potential effectiveness of your approach to this problem by answering the following questions:

- What is the order of growth of the worst-case running time of the `length()`, `lengthAncestor()`, `ancestor()`, and `ancestorSubset()` methods in `ShortestCommonAncestor`?
- What is the order of growth of the best-case running time of the `length()`, `lengthAncestor()`, `ancestor()`, and `ancestorSubset()` methods in `ShortestCommonAncestor`?

Give your answers as a function of the number of vertices V and the number of edges E in the digraph.

Deliverables. Submit all the `.java` and data files in your solution. You may not call library functions except those in `java.lang` and `java.util`. Finally, submit a `readme.txt` file and answer the questions.