CS25100: Data Structures and Algorithms, Spring 2017

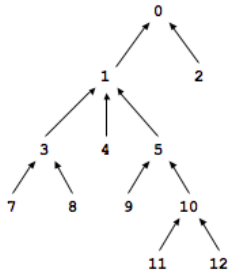
Project 4: WordNet

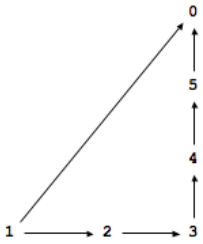
Description

The project will consist of two main tasks revolving around the use of directed graphs: finding shortest ancestral paths and implementing a WordNet application.

Task 1: Shortest Ancestral Paths

An *ancestral path* between two vertices *v* and *w* in a digraph 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. For example, in the digraph at left, the shortest ancestral path between 3 and 11 has length 4 (with common ancestor 1). In the digraph at right, one ancestral path between 1 and 5 has length 4 (with common ancestor 5), but the shortest ancestral path has length 2 (with common ancestor 0).





Implement a immutable data type `SAP` with the following API:

`SAP.java:`

```
public class SAP {
    // constructor
    public SAP(Digraph G) {} // Use the Digraph implementation provided by the book

    // return length of shortest ancestral path of v and w; -1 if no such path
    public int length(int v, int w) {}

    // return a common ancestor of v and w that participates in a shortest
        // ancestral path; -1 if no such path
    public int ancestor(int v, int w) {}
}
```

`SAP.hpp:`

```
class SAP {
public:
  // constructor
  SAP(Digraph G); // Use the C++ implementation of Digraph provided below

  // return length of shortest ancestral path of v and w; -1 if no such path
  int length(int v, int w) const;

  // return a common ancestor of v and w that participates in a shortest
  // ancestral path; -1 if no such path
  int ancestor(int v, int w) const;
};
```

Note that Digraph is an existing class that you DO NOT NEED to implement. You can use algs4.jar (Documentation here), the implementation of classes provided with the textbook, adding it to the classpath for compilation and execution, the same way stdlib.jar (Documentation here) has been used for previous projects.

Also, include a `main()` function (to be included in SAP.cpp if implemented in C++) that takes the names of two files as command-line arguments: a digraph input file (standard format used by `Digraph`) and a digraph test input file. Your main will construct the digraph and read in vertex pairs from the test input file, and prints out the length of a shortest ancestral path between the two vertices and a common ancestor that participates in that path. Note that there can be more than one path with the same length; output the information corresponding to any of them.

Your implementation can be tested using digraphX.txt and digraph1.input where X is 1 or 2. For example, digraph1.txt represents the graph on the left side of the previous figure. In this case the file specifies that the graph contains 13 vertices (first line), 11 edges (second line) and the rest of the lines enumerate the edges. The file digraph1.input contains 4 pairs of vertices and we want to test their shortest ancestral paths (length and ancestor). By running the command on the rightmost section we determine that the first three do have a SAP, while the 4th one doesn't (returning -1 to indicate this).

If there's more than one ancestor **Do not include extra output, other than what is indicated here (like debug prints). The format shown here must be preserved**

```
% more digraph1.txt
13
11
 7  3
 8  3
 3  1
 4  1
 5  1
 9  5
10  5
11 10
12 10
 1  0
 2  0
```

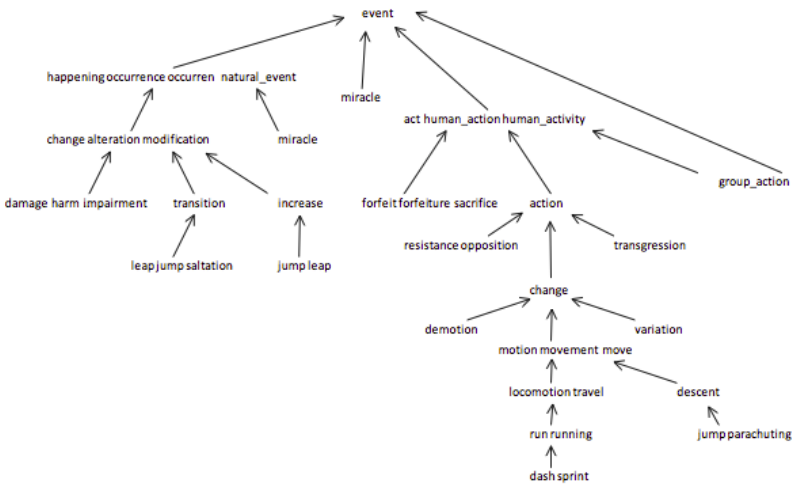
```
% more digraph1.input
3 11
9 12
7 2
1 6
```

```
% java SAP digraph1.txt digraph1.input
sap = 4, ancestor = 1
sap = 3, ancestor = 5
sap = 4, ancestor = 0
sap = -1, ancestor = -1
```

Task 2: WordNet

WordNet is a semantic lexicon for the English language that is used extensively by computational linguists and cognitive scientists. WordNet groups words into sets of synonyms called *synsets* and describes semantic relationships between them. One such relationship is the *is-a* relationship, which connects a *hyponym* (more specific synset) to a *hypernym* (more general synset). For example, a *plant organ* is a hypernym to *plant root* and *plant root* is a hypernym to *carrot*.

For this task, you will build the WordNet graph: 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 graph is directed and acyclic (a DAG), though not necessarily a tree since a synset can have several hypernyms. A small subgraph of the WordNet graph is illustrated below.



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

- *List of noun synsets.* The file `synsets.txt` lists all the (noun) synsets in WordNet. The first field is the *synset id* (an integer), the second field is the *synset*, and the third field is its dictionary definition (or *gloss*). For example, the following line

```
45,AND_circuit AND_gate,a circuit in a computer that fires only when all of its inputs fire
```

means that the synonym set whose elements are `AND_circuit` and `AND_gate` has an id number of 45, and it's gloss is `a circuit in a computer that fires only when all of its inputs fire`. The individual nouns that comprise a synset are separated by spaces (and a synset element is not permitted to contain a space).

Note that a word can appear in more than one synset.

- *List of hypernyms.* The file `hypernyms.txt` contains the hypernym relationships: The first field is a synset id; subsequent fields are the id numbers of the synset's hypernyms. For example, the following line

  ```
  171,22798,57458
  ```

  means that the the synset `171` (`"Actifed"`) has 2 hypernyms: `22798` (`"antihistamine"`) and `57458` (`"nasal_decongestant"`), representing that Actifed is both an antihistamine and a nasal decongestant. The synsets are obtained from the corresponding lines in the file `synsets.txt`.

  ```
  171,Actifed,trade name for a drug containing an antihistamine and a decongestant...
  22798,antihistamine,a medicine used to treat allergies...
  57458,nasal_decongestant,a decongestant that provides temporary relief of nasal...
  ```

Implement a data type `WordNet` that uses the previously defined SAP data type and has the following API:

WordNet.java:

```
public final class WordNet {
    // constructor takes the name of the two input files
    public WordNet(String synsets, String hypernyms) {}

    // is the word a WordNet noun? This can be used to search for existing
    // nouns at the beginning of the printSap method
    public boolean isNoun(String word) {}

    // print the synset (second field of synsets.txt) that is the common ancestor
    // of nounA and nounB in a shortest ancestral path as well as the length of the path,
    // following this format: "sap<space>=<space><number>,<space>ancestor<space>=<space><synsettext>"
    // If no such path exists the sap should contain -1 and ancestor should say "null"
    // This method should use the previously defined SAP datatype
    public void printSap(String nounA, String nounB) {}
}
```

WordNet.hpp:

```
class WordNet {
public:
    // constructor takes the name of the two input files
    WordNet(const std::string& synsets, const std::string& hypernyms) {}

    // is the word a WordNet noun? This can be used to search for existing
    // nouns at the beginning of the printSap method
    bool isNoun(const std::string& word) const {}

    // print the synset (second field of synsets.txt) that is the common ancestor
    // of nounA and nounB in a shortest ancestral path as well as the length of the path,
    // following this format: "sap<space>=<space><number>,<space>ancestor<space>=<space><synsettext>"
    // If no such path exists the sap should contain -1 and ancestor should say "null"
    // This method should use the previously defined SAP datatype
    void printSap(const std::string& nounA, const std::string& nounB) const {}
};
```

Also, include a `main()` (to be included in SAP.cpp if implemented in C++) that takes the names of three files as command-line arguments: one for hypernyms (hypernyms.txt), one for synsets (synsets.txt) and a test input file (wordnet.input). The first two files will be used to call the WordNet constructor. The third one will contain a series of noun pairs (separated by a single space) that should be used to call printSap, which will output the length of the path and ancestor as described in printSap's comments.

synsets.txt and hypernyms.txt are the data sources and wordnet.input contains the noun pairs that will be used to call printSap. It contains two pairs of nouns that are contained in the test files provided and a third one that is not. When WordNet is executed it will return the length and ancestors for the first two and it will indicate that the third one was not successful (as specified in printSap comments). **Do not include extra output, other than what is indicated here (like debug prints). The format shown here must be preserved**

```
% more wordnet.input
administrative_district populated_area
individual edible_fruit
individual nonexistingword
```

```
% java WordNet synsets.txt hypernyms.txt wordnet.input
sap = 4, ancestor = region
sap = 7, ancestor = physical_entity
sap = -1, ancestor = null
```

The tests provided are a sample of the relationships and should guide the development of the assignment. Additional examples about the relationships present in the input files are noted next:

- 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 diferent paths to their common ancestor `physical_entity`.

```
individual -> object -> physical_entity
individual -> causal_agency -> physical_entity
edible_fruit -> garden_truck -> food -> solid -> matter -> physical_entity
edible_fruit -> reproductive_structure -> plant_organ -> plant_part ->
                natural_object -> unit -> object -> physical_entity
```

- The following pairs of nouns are very far apart:

```
23 white_marlin, mileage
32 Black_Plague, black_marlin
32 American_water_spaniel, histology
32 Brown_Swiss, barrel_roll
```

- The following synset has many ancestors and paths to `entity`.

```
Ambrose Saint_Ambrose St._Ambrose
```

Report

Submit a report (PDF file) answering the following questions:

- Describe the data structure(s) you used to store the information in synsets.txt. Why did you make this choice?
- Describe the data structure(s) you used to store the information in hypernyms.txt. Why did you make this choice?
- Describe your algorithm to compute the SAP. What is the worst-case running time as a function of the structure of the graph (height, number of vertices, number of edges, etc)? Best case running time?

Checklist

Before submitting your project make sure that you comply with the following (while on lore):

Task 1: Shortest Ancestral Paths

- java SAP receives two command-line arguments: the file that specifies the graph and the one with the vertices that will be tested
- The output of java SAP only contains what is requested (and nothing more). The format that each line must follow is: "sap<space>=<space><number>,<space>ancestor<space>=<space><number>"
- You are allowed to use BreadthFirstDirectedPaths.java from the author's code to implement SAP (Hint: Check exercise 4.2.25)
- A vertex is considered an ancestor of itself. Make sure you take this into account
- If there are several shortest paths with the same length, you only need to output the information from one of them
- Running "java SAP digraph1.txt digraph1.input" returns the values shown on the main page as an example (Note that this is a sample test and you can use the rest of the files for more testing).
- Your main function is contained in SAP.java. Additionally, check that you include all additional files required for SAP before submitting
- Your program returns "sap = -1, ancestor = -1" when the sap does not exist

Task 2: WordNet

- java WordNet receives three command-line arguments in the following order: "java WordNet synsetfile hypernymfile wordnettestfile"
- The output of java WordNet only contains what is requested (and nothing more). The format that each line must follow is: "sap<space>=<space><number>,<space>ancestor<space>=<space><synsettext>"

- When the nouns do not have a common ancestor your program returns "sap = -1, ancestor = null"
- A synset can have several hypernyms (i.e. you can support more than one outgoing edge per node)
- The same noun can appear in several synsets. Make sure you take this into account
- You can use String.split() to parse the comma-separated input files
- Although glosses (third column in synsets file) are part of the file, we will not use them for this project
- Your main function is contained in WordNet.java. Additionally, check that you include all additional files required for WordNet before submitting
- Your program returns "sap = -1, ancestor = null" when the common ancestor does not exist

Submission

Submit your solution before Apr 7 11:59pm. Vocareum will be used to submit assignments.