Project 6 (WordNet)

Clarifications and Hints

Prologue

Project goal: find the shortest common ancestor of a digraph in WordNet, a semantic lexicon for the English language that computational linguists and cognitive scientists use extensively

The zip file (`http://www.swamiiyer.net/cs210/project6.zip`) for the project contains

- project specification (`project6.pdf`)
- starter files (`WordNet.java`, `ShortestCommonAncestor.java`, `Outcast.java`)
- test script (`run_tests.py`)
- test data (`data/`)
- report template (`report.txt`)

This checklist will help only if you have read the writeup for the project and have a good understanding of the problems involved. So, please read the project writeup before you continue with this checklist.

Problems

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

method	description
`WordNet(String synsets, String hypernyms)`	construct `WordNet` object given the names of the input (synset and hypernym) files
`Iterable<String> nouns()`	all WordNet nouns
`boolean isNoun(String word)`	is the word a WordNet noun?
`String sca(String noun1, String noun2)`	a synset (second field of `synsets.txt`) that is a shortest common ancestor of $noun_1$ and $noun_2$
`int distance(String noun1, String noun2)`	distance between $noun_1$ and $noun_2$

Hints

- Instance variables
  - A symbol table that maps a synset noun to a set of synset IDs (a synset noun can belong to multiple synsets), `RedBlackBST<String, SET<Integer>> st`
  - A symbol table that maps a synset ID to the corresponding synset string, `RedBlackBST<Integer, String> rst`
  - `ShortestCommonAncestor sca`

Problems

- `WordNet(String synsets, String hypernyms)`
  - Initialize instance variables `st` and `rst` appropriately using the synset file
  - Construct a `Digraph` object `G` (representing a rooted DAG) with V vertices (equal to the number of entries in the synset file), and add edges to it, read in from the hypernyms file
  - Initialize `sca` using `G`
- `Iterable<String> nouns()`
  - Return all the nouns as an iterable object
- `boolean isNoun(String word)`
  - Return `true` if the given word is a synset noun, and `false` otherwise
- `String sca(String noun1, String noun2)`
  - Return the shortest common ancestor of the given nouns, computed using `sca`
- `int distance(String noun1, String noun2)`
  - Return the length of the shortest ancestral path between the given nouns, computed using `sca`

Problems

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

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

Hints

- Instance variable
  - A rooted DAG, `Digraph G`
- `ShortestCommonAncestor(Digraph G)`
  - Initialize instance variable appropriately

Problems

- `SeparateChainingHashST<Integer, Integer> distFrom(int v)`
  - Return a map of vertices reachable from `v` and their respective shortest distances from `v`, computed using BFS starting at `v`
- `int ancestor(int v, int w)`
  - Return the shortest common ancestor of vertices `v` and `w`; to compute this, enumerate the vertices in `distFrom(v)`, and find a vertex `x` that is also in `distFrom(w)` and yields the minimum value for `dist(v, x) + dist(x, w)`
- `int length(int v, int w)`
  - Return the length of the shortest ancestral path between `v` and `w`; use `int length(int v, int w)` and `int ancestor(int v, int w)` to implement this method
- `int[] triad(Iterable<Integer> A, Iterable<Integer> B)`
  - Return a 3-element array consisting of a shortest common ancestor `a` of vertex subsets `A` and `B`, a vertex `v` from `A`, and a vertex `w` from `B` such that the path `v-a-w` is the shortest ancestral path of `A` and `B`; use `int length(int v, int w)` and `int ancestor(int v, int w)` to implement this method
- `int length(Iterable<Integer> A, Iterable<Integer> B)`
  - Return the length of the shortest ancestral path of vertex subsets `A` and `B`; use `int[] triad((Iterable<Integer> A, Iterable<Integer> B)` and `SeparateChainingHashST<Integer, Integer> distFrom(int v)` to implement this method
- `int ancestor(Iterable<Integer> A, Iterable<Integer> B)`
  - Return a shortest common ancestor of vertex subsets `A` and `B`; use `int[] triad((Iterable<Integer> A, Iterable<Integer> B)` to implement this method

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

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

Hints

- Instance variable
  - `WordNet wordnet`
- `Outcast(WordNet wordnet)`
  - Initialize instance variable appropriately
- `String outcast(String[] nouns)`
  - Compute the sum of the distances (computed using `wordnet`) between each noun in `nouns` and every other, and return the noun with the largest such distance

Epilogue

The `data` directory has a number of sample input files for testing

- See assignment writeup for the format of the synset (`synset*.txt`) and hypernym (`hypernym*.txt`) files
- The files `digraph*.txt` representing digraphs can be used as inputs for the test client in `ShortestCommonAncestor`

```
$ more digraph1.txt
12
11
 6  3
 7  3
 3  1
 4  1
 5  1
 8  5
 9  5
10  9
11  9
 1  0
 2  0
```

- The files `outcast*.txt`, each containing a list of nouns, can be used as inputs for the test client in `Outcast`

```
$ more outcast5.txt
horse
zebra
cat
bear
table
```

Epilogue

To receive full credit for a problem, your solution must implement the "corner cases" (if any) and meet the "performance requirements" (if any)

Your project report (use the given template, `report.txt`) must include

- time (in hours) spent on the project
- short description of how you approached each problem, issues you encountered, and how you resolved those issues
- acknowledgement of any help you received
- other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Before you submit your files

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

  ```
  $ python3 run_tests.py -v [<problems>]
  ```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test, separated by spaces; 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
- make sure your report isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling/grammatical mistakes