**Word Net**

**Word Net -** 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](http://en.wikipedia.org/wiki/Watson_(computer)) computer system.

WordNet works on a set of synonyms called as **Synsets** . WordNet also tries to establish relationship between hyponyms and hypernyms.

**Hyponyms** – It is a specific synset not a generalized one . It usually belongs to a broader category . E.g – red is a hyponym of colour category .

**Hypernyms**  - It is the broader category . In the above example colour is the Hypernym. A synset can have multiple hypernyms .

**Purpose of the project :**

The purpose of the project is to create a rooted DAG which has synsets as vertices and the connection between vertices ‘v’ and ‘w’ suggests that w is a hypernym of ‘v’. All the vertices have one common ancestor which is the root . A user should be able to retrieve the immediate ancestor of a particular vertex from the rooted DAG .Using the built graph , we should also be able to figure the distance between two given vertices . It is also expected out of us to figure out the odd-one-out or the **‘outcast’** is. This can be figured out by measuring the lengths between the given vertices . The vertex which has the longest distance with the rest of the vertices is the outcast .

We are expected to write 3 classes – WordNet. Java , SAP.java and Outcast.java

**SAP.java**

SAP stands for shortest ancestral path , which means , the 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. This class contains methods to find the shortest ancestral path between 2 vertices v and w and to find the common ancestor between 2 vertices . The length () and ancestor() methods are written twice , once to handle vertices that are in the form of integers and another to accept Iterables as parameters .

**WordNet.java**

This class contains methods to analyse whether a given word is a noun or not based on the hash tables we have built , to find the distance between two vertices by creating an object SAP here and to find the shortest ancestral path between two vertices .

**Outcast.java**

This class has one method , outcast() which takes in a noun array//vertices array as a parameter and calculates the distance between each of those vertices .The vertex that is the farthest from the rest of the vertices , is the outcast word . This helps us determine what kind of words belong to the same category and what don’t .

**API –**

**WordNet.java**

**public class WordNet {**

// constructor takes the name of the two input files

**public WordNet(String synsets, String hypernyms)**

// returns all WordNet nouns

**public Iterable<String> nouns()**

// is the word a WordNet noun?

**public boolean isNoun(String word)**

// distance between nounA and nounB (defined below)

**public int distance(String nounA, String nounB)**

// a synset (second field of synsets.txt) that is the common ancestor of nounA and nounB

// in a shortest ancestral path (defined below)

**public String sap(String nounA, String nounB)**

// do unit testing of this class

**public static void main(String[] args)**

**}**

**SAP.java**

**public class SAP {**

// constructor takes a digraph (not necessarily a DAG)

**public SAP(Digraph G)**

// length of shortest ancestral path between v and w; -1 if no such path

**public int length(int v, int w)**

// 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)**

// length of shortest ancestral path between any vertex in v and any vertex in w; -1 if no such path

**public int length(Iterable<Integer> v, Iterable<Integer> w)**

// a common ancestor that participates in shortest ancestral path; -1 if no such path

**public int ancestor(Iterable<Integer> v, Iterable<Integer> w)**

// do unit testing of this class

**public static void main(String[] args)**

**}**

**Outcast.java**

**public class Outcast {**

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

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

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

**}**

**Time Complexities of the afore mentioned classes :**

Outcast.java – used 2 loops (loop inside loop)– O(N2)

Wordnet.java – O(N)

SAP.java – O(N)

**Failed Test Cases:**

**Correctness:**

Test 15: check whether SAP is immutable

\* digraph1.txt

- after adding edges (v, 0) to G

- v = 0, w = 3

- student length before = 2

- student length after = 1

\* digraph2.txt

- after adding edges (v, 0) to G

- v = 0, w = 2

- student length before = 4

- student length after = 1

\* digraph3.txt

- after adding edges (v, 0) to G

- v = 0, w = 1

- student length before = -1

- student length after = 1

\* digraph4.txt

- after adding edges (v, 0) to G

- v = 0, w = 1

- student length before = 3

- student length after = 1

\* digraph5.txt

- after adding edges (v, 0) to G

- v = 0, w = 1

- student length before = -1

- student length after = 1

\* digraph6.txt

- after adding edges (v, 0) to G

- v = 0, w = 2

- student length before = 2

- student length after = 1

\* digraph-ambiguous-ancestor.txt

- after adding edges (v, 0) to G

- v = 0, w = 2

- student length before = 2

- student length after = 1

==> FAILED

Test 1: check distance() with random noun pairs

\* 1000 pairs using synsets = synsets.txt; hypernyms = hypernyms.txt

- failed on pair 33 of 1000

- nounA = Lycopodiales

- nounB = warrantee

- student distance() = 14

- reference distance() = 11

==> FAILED

Test 4: check sap() with random noun pairs

\* 1000 pairs using synsets = synsets.txt; hypernyms = hypernyms.txt

- student sap() is too far to be a valid common ancestor when testing pair 1 of 1000

- nounA = Solidago\_multiradiata

- nounB = Pentecostal\_religion

- student sap() = 'agammaglobulinemia'

- reference sap() = 'entity'

- student distance() = 19

- reference distance() to 'entity' = 19

- 'agammaglobulinemia' is not a common ancestor of 'Solidago\_multiradiata' and 'Pentecostal\_religion'

==> FAILED

Test 8: check constructor when input is not a rooted DAG