# WordNet Report

**Problem Statement :**

Wordnet is a large database of English. Nouns, verbs, Adjectives and Adverbs are grouped into small sets of cognitive synonyms(Synsets) each with a different set of meaning. Synsets are interlinked by means of conceptual semantic and lexical relations. One such relationship is, which connects a hyponym (more specific synset) to a hypernym (more general synset).

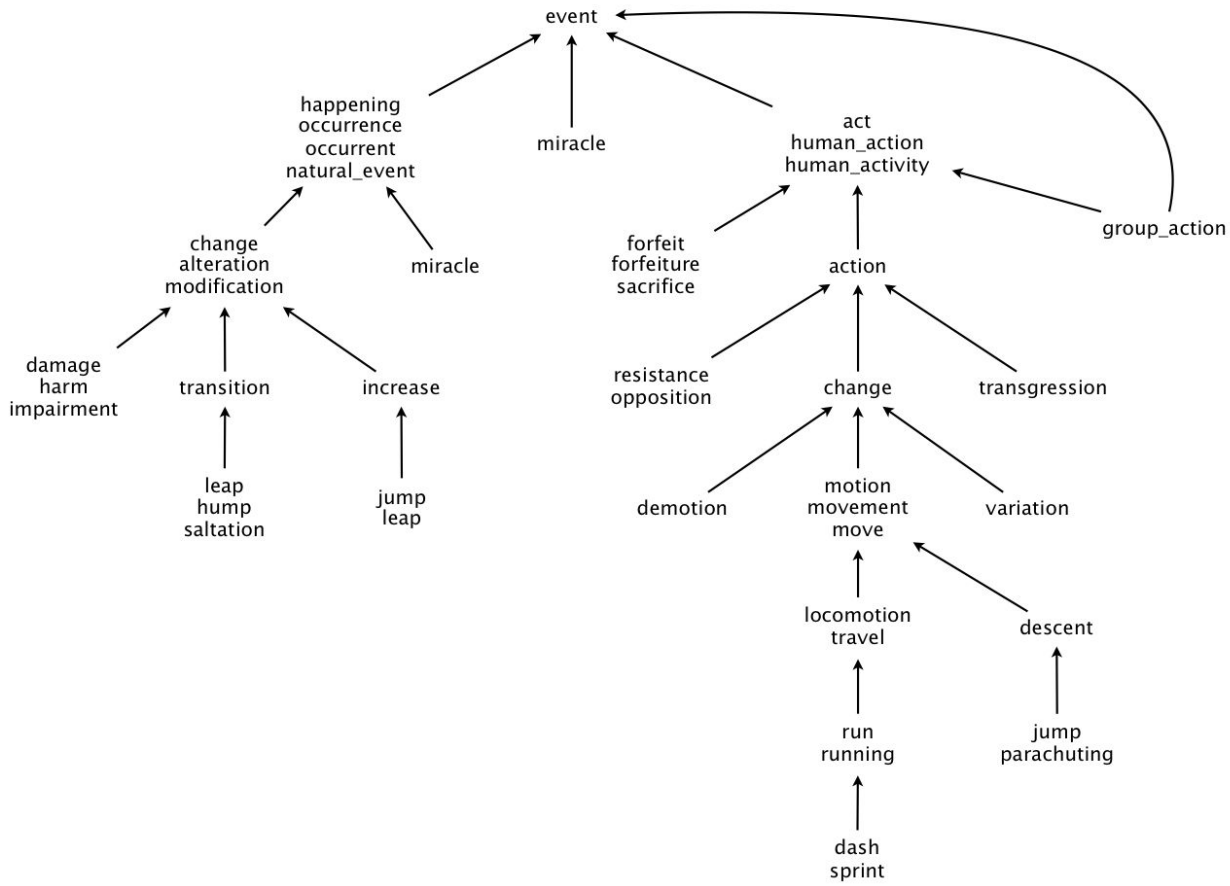
The main relation among words in WordNet is synonymy, as between the words shut and close or car and automobile. Synonyms--words that denote the same concept and are interchangeable in many contexts--are grouped into unordered sets (synsets).

In this project we only work on Synsets which are Nouns. 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.

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.

**WordNet Digraph :**

Each vertex v is an integer that represents a Synset and each directed edge v → w represents that w is a hypernym of v. WordNet digraph is a rooted DAG (Directed Acyclic graph) : it is acyclic and has one vertex the root, the ancestor of each 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.




We implement the WordNet class with the following API.

**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)
}
```

In the following corner cases we throw a new Illegal Argument Exception :

- Any argument to the constructor or an instance method is null
- The input to the constructor does not correspond to a rooted DAG.
- Any of the noun arguments in distance() or sap() is not a WordNet noun.

**Shortest Ancestral Path (SAP) :**

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. We refer to the common ancestor in a shortest ancestral path as a shortest common ancestor. Note also that an ancestral path is a path, but not a directed path.

We implement the SAP class with the following API,

```
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)
}
```

In the following Corner cases we throw a new Illegal Argument Exception :

- Any argument is null
- Any vertex argument is outside its prescribed range
- Any iterable argument contains a null item

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
- distance(x, y) = length of shortest ancestral path of subsets A and B
- sca(x, y) = a shortest common ancestor of subsets A and B

**Outcast Detection :**

Given a list of WordNet nouns x1, x2, ..., xn, 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:

di = distance(xi, x1) + distance(xi, x2) + ... + distance(xi, xn)

and return a noun xt for which dt is maximum. Note that distance(xi, xi) = 0, so it will not contribute to the sum.

We use the following API to implement the Outcast Class :

```
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
}
```

**Results :**

```
Compilation:  PASSED
API:          PASSED

Spotbugs:     PASSED
PMD:          PASSED
Checkstyle:   FAILED (0 errors, 2 warnings)

Correctness:  34/36 tests passed
Memory:       4/4 tests passed
Timing:       27/27 tests passed

Aggregate score: 96.67%
[Compilation: 5%, API: 5%, Spotbugs: 0%, PMD: 0%, Checkstyle: 0%, Correctness: 60%, Memory: 10%, Timing: 20%]

ASSESSMENT DETAILS

The following files were submitted:
----------------------------------
1.4K Jan 19 15:05 Outcast.java
5.4K Jan 19 15:05 SAP.java
7.2K Jan 19 15:05 WordNet.java
```

**Code Link :**
**https://github.com/saikotesh0102/ADS-2_2019501007/tree/master/Day%20-%205**

**Failed Test Cases :**

The test cases that are failed are 2 test cases in SAP.java