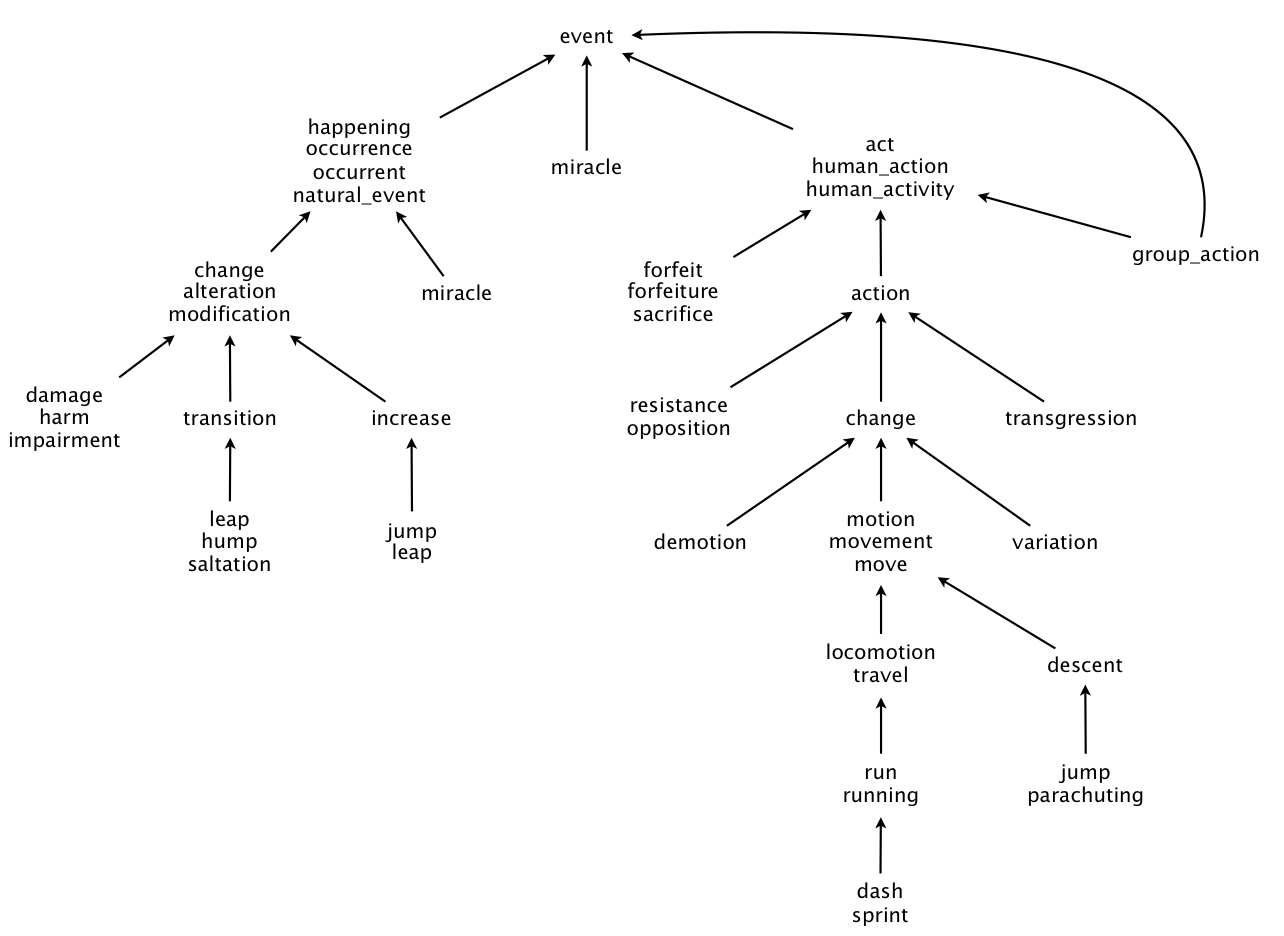
**WordNet Project Report**

1. Problem statement
2. Related concepts
3. Code
4. Test-cases
5. Complexities

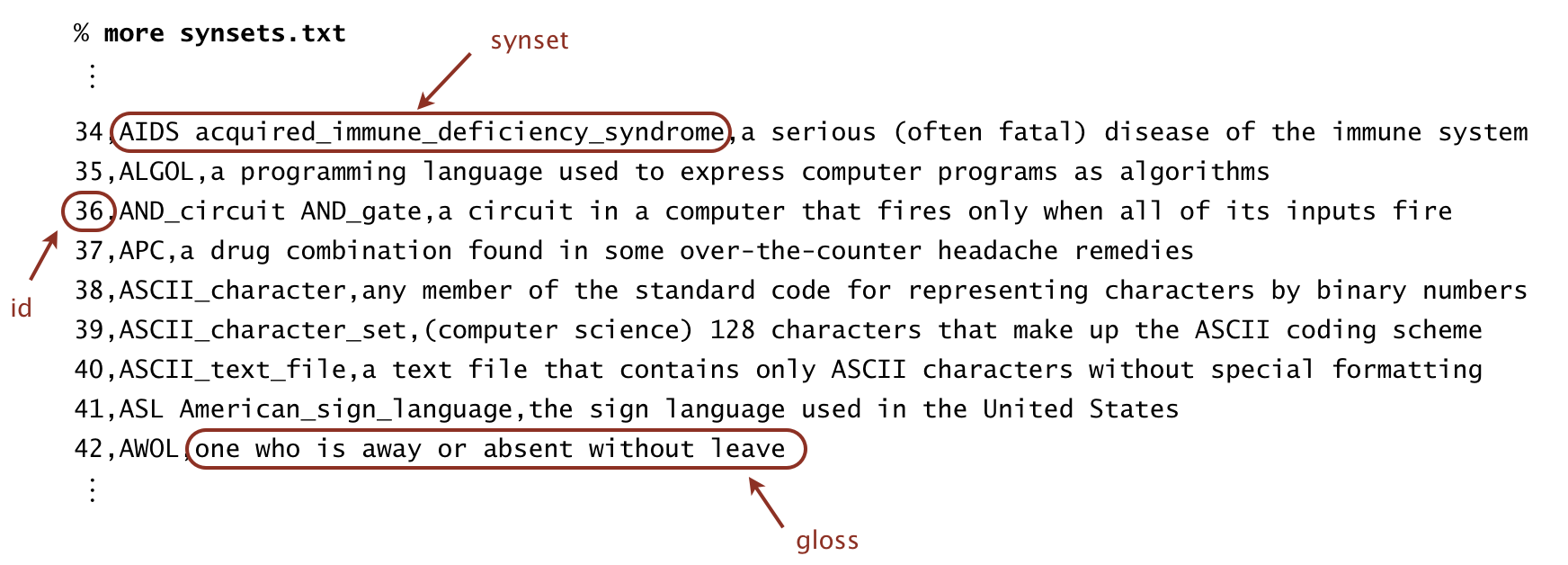
Problem statement:

We have three class files, they are WordNet.java, SAP.java, Outcast.java. We need to implement these three class files methods. We also need to pass the code in the coursera. Mainly, we have to develop a digraph for Hypernyms.txt file to add edges between the two vertices i.e., id’s contained in the synsets.txt file as follows.

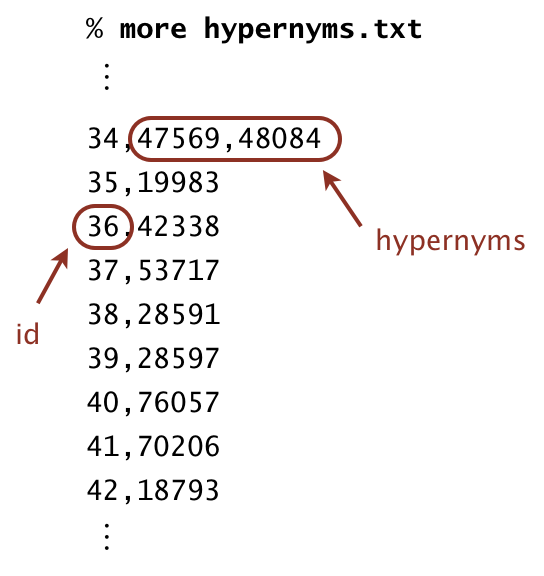


A WordNet digraph is a directed graph which traverse from child to its’ ancestors. This WordNet is used in some apps such as chatbots, word suggestions in search bars, dictionary applications, etc.

Synsets:



Hypernyms:



Related concepts:

Some of the concepts and data structures that we need to use in building the WordNet are:

* ArrayList
* Digraph
* Bag
* LinearProbingHashTable
* BreathFirstDirectedPaths
* SAP class – it is used to find the shortest distance and shortest ancestor path between the two synsets id’s of the Digraph.
* WordNet class – it is used to find length and shortest ancestor path between two input nouns.
* Outcast class – to find the odd word from the word array given.

Code:

1. WordNet:

* WordNet (String synsets, String hypernyms) –
  + Constructor that takes filenames filenames of synsets, hypernyms.
* nouns() –
  + returns an iterable which is the set of nouns.
* isNoun() –
  + checks whether the given word is noun.
* distance (String nounA, String nounB) –
  + finds the distance between the two nouns.
* sap (String nounA, String nounB) –
  + finds the common ancestor between the two nouns.
* readSynsets (String synsets) –
  + reads the synsets file and store data in the hashtable
* readHypernyms (String hypernyms) –
  + reads the hypernyms file and store data in the ArrayLists.

1. SAP:

* SAP (Digraph dg) - constructor for initializing the digraph.
* length (int v, int w) - It is used to return the shortest length between two vertices v and w.
* ancestor (int v, int w) - it is used to return the common ancestor for v and w vertices and the shortest ancestral path.
* length (Iterable<Integer> v, Iterable<Integer> w) - it is used to find the shortest length between two iterables v and w.
* ancestor (Iterable<Integer> v, Iterable<Integer> w) - it is used to find the shortest ancestor between two iterables v and w.

1. Outcast:

* Outcast (WordNet wordnet) – Constructor for initializing the wordnet.
* Outcast (String [ ] nouns) - it is used to find the outcast noun among the given set of nouns.

Test cases:

First, I got some compilation errors and later cleared them and faced API errors where, some of

the methods and attributes were public. So, I have changed them to private and API is passed.

Some of the spot bugs are there because of scanner class used. So replaced them with In class

Now, the outcast method worked perfectly but as I have used sap in wordnet both classes have

logical errors. So, the ancestor output of the sap methods lead to wrong outputs as the variable

that needs to be returned was returned incorrectly. Now I have changed them to the required

variable and finally passed the code.

Complexities:

WordNet:

* ● WordNet (String synsets, String hypernyms) - O(N^2)
* ● Nouns() - O(N)
* ● isNoun(String word) - O(1)
* ● distance(String nounA,String nounB) - O(N)
* ● sap(String nounA, String NounB) - O(N)
* ● parsesynsets(String synsets) - O(N)
* ● parehypernym(String hypernym) - O(N^2)

SAP:

* ● SAP(Digraph dg) - O(1)
* ● length(int v, int w) - O(N)
* ● ancestor(int v, int w) -O(N)
* ● length(Iterable<Integer> v, Iterable<Integer> w) -O(N)
* ● ancestor((Iterable<Integer> v, Iterable<Integer> w) - O(N)
* OutCast:
* ● OutCast(WordNet wordnet) - O(1)
* ● outcast(String [ ] nouns) - O(N^2)