**Problem Statement:**

WordNet is a semantic lexicon for the English language which computational linguists and cognitive scientists use extensively for example, WordNet was a key component in IBM’s Jeopardy-playing Watson computer system. WordNet divides the group of words into two types

1. Synsets: set of synonyms. For example {AND circuit, AND gate} is a synset that represent a logical gate that fires only when all of its inputs fire. WordNet also describes semantic relationships between synsets.

2. Hypernym: relationship is the is-a relationship, which connects a hyponym (more specific synset) to a hypernym (more general synset). For example, the synset {gate, logic gate} is a hypernym of {AND circuit, AND gate} because an AND gate is a kind of logic gate.

**Related Concepts:**

**Digraph**

Digraph is a directed graph to store synsets and hypernyms as vertices. In hypernyms we have synset ids only. We have relations between synset ids in hypernyms. We represent these relations as edges using Digraph.

**HashMap**

HashMap stores the data in (Key, Value) pairs. To access a value, one must know its key. HashMap is known as HashMap because it uses a technique called Hashing. Hashing is a technique of converting a large String to small String that represents the same String. A shorter value helps in indexing and faster searches.

**Breadth First Search**

Breadth First Search is an algorithm to find the path and for traversing the vertices. Here we traverse the vertices i.e., id.

**Queue**

We use queue data structure in Breadth First Search which is used to store the vertices and perform operations like enqueueing and dequeuing the elements i.e., vertices.

**Code:**

In this project, we have mainly had three java files namely WordNet.java, SAP.java, Outcast.java.

In **WordNet.java**, we have a parameterised constructor to initialise the variables we use in the file. For example, id is initialised to store the synset ids and words in it using HashMap and with String as a parameter. For hypernyms we took.

We have three private methods named **parseSynsets**() for reading synsets, **parseHypernyms**() for reading hypernyms and making relations, **hasCycle**() to check the graph has cycle or not.

We have four public methods named **nouns**() for getting nouns, **isNoun**() to know whether it is a noun or not which is hypernyms, **distance**() to get distance between two nouns using sap.java, **sap**() to get ancestor of two nouns using sap.java.

In **SAP.java**, we have four public methods. Two of those are to find length between the vertices and the other two are to find ancestor of the two vertices. We have one private method named **sap helper**() in which we have common code of all above four methods.

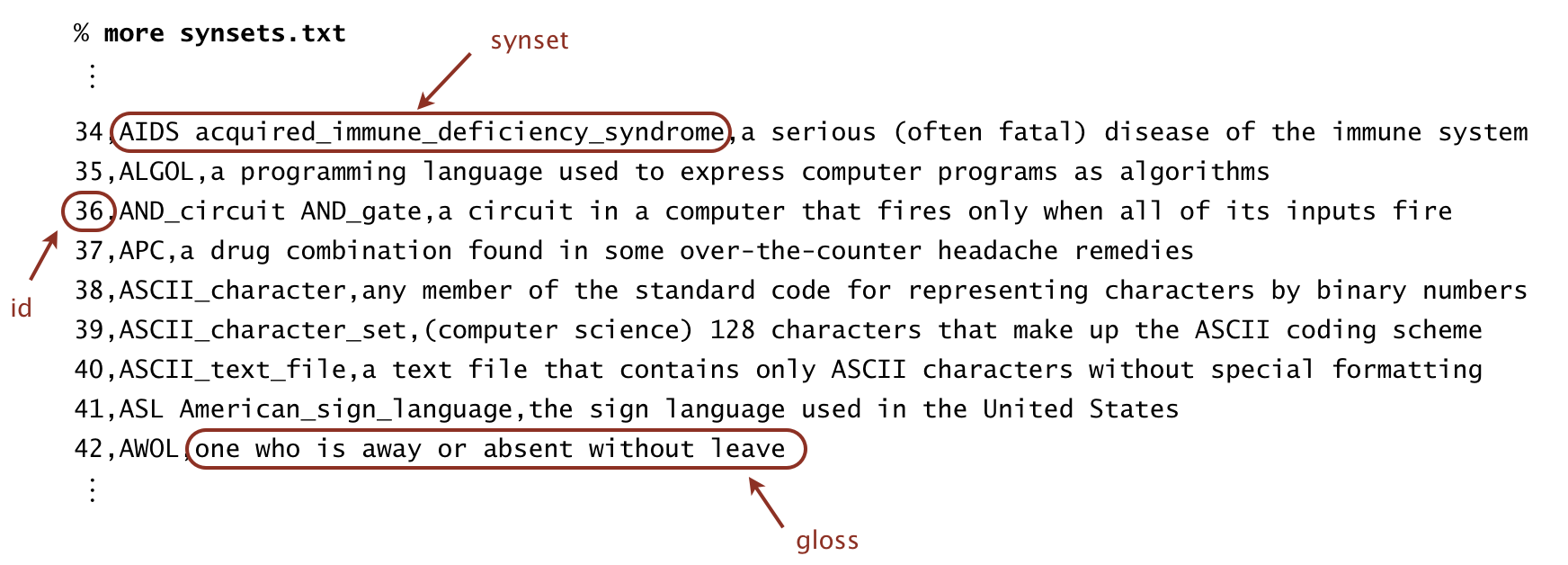
In **Outcast.java**, we find the unrelated word in the outcast files by calculating the maximum distance of the words. For example, a list of WordNet nouns *x*1, *x*2, ..., *xn*, which noun is the least related to the others? To identify *an outcast*, compute the sum of the distances between each noun and every other one:

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

**Input:**

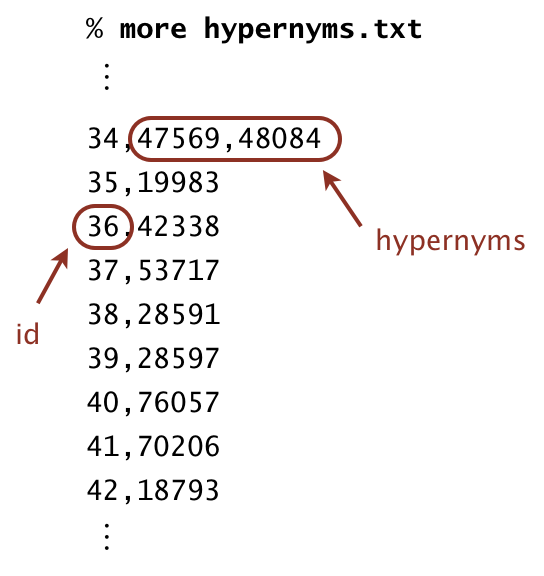
We now describe the two data files that you will use to create the WordNet digraph. The files are in comma-separated values (CSV) format: each line contains a sequence of fields, separated by commas.

List of synsets. The file 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.



For example, line 36 means that the synset { AND\_circuit, AND\_gate } has an id number of 36 and its gloss is a circuit in a computer that fires only when all of its inputs fire. The individual nouns that constitute a synset are separated by spaces. If a noun contains more than one word, the underscore character connects the words (and not the space character).

List of hypernyms. The file 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.



For example, line 36 means that synset 36 (AND\_circuit AND\_Gate) has 42338 (gate logic\_gate) as its only hypernym. Line 34 means that synset 34 (AIDS acquired\_immune\_deficiency\_syndrome) has two hypernyms: 47569 (immunodeficiency) and 48084 (infectious\_disease).

**Test Cases:**

According to course era, compilation is passed, API is passed, spot bugs are passed, pmd is passed and check style is passed but with one warning.

Based on correctness, 28 test cases are passed out of 36.

Based on memory, 4 test cases are passed out of 4.

Based on timing, 27 test cases are passed out of 27.

According to course era, the aggregate score is 87.41%.

**Complexity:**

**Conclusion:**

I did the project and in 40 plus times, I got the score of 84.

Firstly, I submitted the project but I got 0 because I didn’t import some of wanted imports and I also included exceptions.

Next, I got 5 because some of the variables are not declared as private.

Next, I got 10 because there are errors in spot bugs and pmd.

Next, I got 55 and some of the test cases are failed because of length method and some of the null pointer exceptions then again it got increased by 71 due to API.

I got cleared and got 84 score but I couldn’t get to know about the errors in distance and sap methods which were the reasons not to get 100 score.