**WORDNET PROJECT REPORT**

**INDEX**:

1. Problem Statement

2. Related Concepts

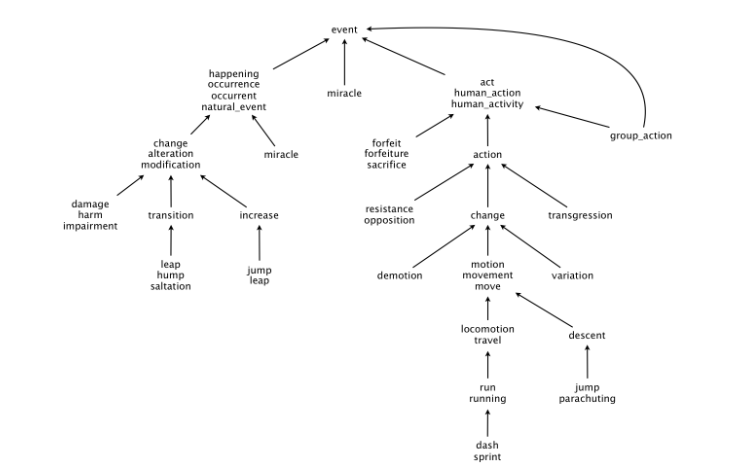
3. Code

4. Test Cases

5. Complexity

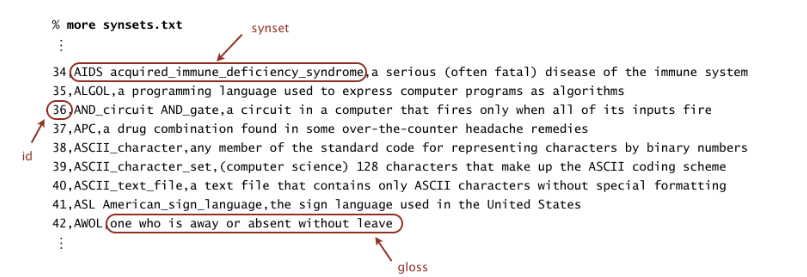
**Problem Statement:**

We need to build a WordNet Digraph which is one of the key components used in IBM’s Watson. We have 2 word inputs called as vertices v , w which will be present in synsets. The link between v and w is called as edge that is in hypernyms.

A wordnet digraph is a directed graph which traverse from child to ancestors unlike trees. So, this is used in some of the applications like chatbots, word suggestions in mail and messaging applications in mobile based in the scenario that we are typing. This is built with the help of synsets and hypernyms. 

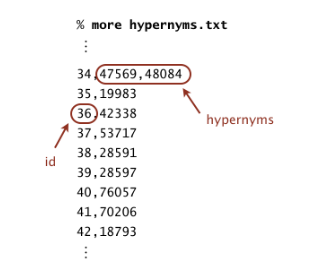
**The WordNet input file formats.** We now describe the two data files that you will use to create the WordNet digraph. The files are in comma-separated values (CSV) format: each line contains a sequence of fields, separated by commas.

**Synsets**:



This file contains all noun synsets in WordNet. It is the set of data that contains ID, noun and description. we need to create the links between these words.

**Hypernyms**:



The file contains the hypernym relationships. They provide the links that maintain the data which is the link between certain words.

**Related Concepts:**

Some of the concepts and data structures that we have used in building the WordNet are:

● Digraph

● Bag

● HashMap

● Breadth First Search

● Arrays

● SAP Data Type (used to find the shortest distance between 2 nouns and common ancestors)

● WordNet Data Type (used to find distance between 2 nouns, shortest ancestral path)

● Outcast Data Type (Used to find the odd word in the given words)

**Code**

Classes:

1. Wordnet
2. SAP
3. Outcast

**WordNet**:

● WordNet(String synsets, String hypernyms) – A constructor that takes filenames of synsets and hypernyms as input and initializes the private attributes SAP, Digraph, HashMaps.

● isNoun(String word) - checks whether the given word is noun or not.

● distance(String nounA, String nounB) - finds distance between the two nouns.

● sap(String nounA, String NounB) - finds the common ancestor and the shortest ancestral path.

● parseSynsets(String synsets) - reads the file and creates hash tables with id and nouns.

● parseHypernym(String hypernym) - read the file and creates the edges in digraph.

**SAP**:

● SAP(Digraph G) – A constructor that initializes digraph G.

● length(int v, int w) - it is used to find the shortest ancestral path between two vertices.

● ancestor(int v, int w) - it is used to find 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 ancestral path between two vertices given v and w are set of vertices.

● ancestor(Iterable<Integer> v, Iterable<Integer> w) - it is used to find the common ancestor for v and w vertices and the shortest ancestral path given v and w are set of vertices.

**OutCast**:

● OutCast(WordNet wordnet) – A constructor that initializes wordnet.

● outcast(String [ ] nouns) - it is used to find the outcast noun among the given set of nouns.

**Complexities**

**WordNet**:

● WordNet(String synsets,String hypernyms) - O(N^2)

● isNoun(String word) - O(1)

● distance(String nounA,String nounB) - O(N)

● sap(String nounA, String NounB) - O(N)

● parseSynsets(String synsets) - O(N)

● parseHypernym(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 v, Iterable w) -O(N)

● ancestor((Iterable v, Iterable w) - O(N)

**OutCast**:

● OutCast(WordNet wordnet) - O(1)

● outcast(String [ ] nouns) - O(N^2)