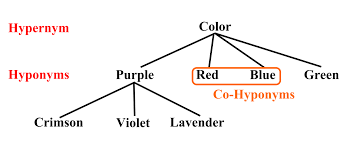
**WordNet**

**Problem Statement**

WordNet is a database of English words which are linked by their semantic relationships. It groups English words into sets of synonyms called synsets, provides short definitions and usage examples in the way provided by hypernyms. Hence WordNet is Rooted Directed Acyclic Graph.

**Ex :** color is a hypernym of green



This program expects us to

1. **Calculate SAP and distance between the two nouns A and B**

Typically the ancestral path is the directional path from a vertex A to a common ancestor plus the directional path from the vertex B to the same common ancestor.

There might be several ancestors for given set of nouns and this program expects us to find the shortest common ancestor (SAP) as well as distance from ancestor to the given nouns.

1. **Outcast**

Outcast is a strange word from the given set of words and this program expects us to find the outcast when given an array of nouns.

**Related Concepts**

1. Used a Directed Graph to create a graphic structure between the vertices provided with the hypernyms.
2. There are several algorithms like DFS, BFS etc that helps us to traverse through the graph. I used Breadth First Search algorithm because it starts traversing the graph from root node and explores all the neighbouring nodes which guarantees the shortest path.

**Test Cases**

1. Two test cases are failed and scored 86/100.
2. The two test cases failed are check immutability of SAP and random calls to distance and sap methods in word net

**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)**

**}**

**SAP**

**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**

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

**In in = new In(args[0]);**

**Digraph G = new Digraph(in);**

**SAP sap = new SAP(G);**

**while (!StdIn.isEmpty()) {**

**int v = StdIn.readInt();**

**int w = StdIn.readInt();**

**int length = sap.length(v, w);**

**int ancestor = sap.ancestor(v, w);**

**StdOut.printf("length = %d, ancestor = %d\n", length, ancestor);**

**}**

**}**

**Complexity**

|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Methods** | **Time Complexity**  **(Worst Case)** | **Space Complexity** |
| **WordNet** | Constructor | Proportional to input files size | 2 \* proportional to number of nouns in the input file  O (n) |
|  | isNoun(String word) | Proportional to number of nouns | 1 |
|  | distance  (String A, String B) | E + V | 1 |
|  | sap(String A, String B) | E + V | 1 |
|  | Iterable nouns() | Proportional to number of nouns | 1 |
| **SAP** | length(int v, int w) | E + V | E + V |
|  | ancestor(int v, int w) | E + V | E + V |
|  | length  (Iterable v, Iterable w) | (E + V) \* (v.length + w.length) | E + V |
|  | ancestor  (Iterable v, Iterable w) | (E + V) \* (v.length + w.length) | E + V |
| **Outcast** | outcast(String[] nouns) | V^2 | 1 |