Jenna Ellis

<u>Report and Documentation:</u>

To store the information given in synset.txt, I used two hash-tables. The first being a hash table that took a string (one of the given noun entries) as the key, and stored the corresponding integers that correspond to their synsets as the value. The second hash-table took the integer value as a key and mapped to the corresponding strings as the values. I used these data structures in order to most efficiently save and access all important corresponding values, of which were necessary to access multiple times during the printing portion of the algorithm (printSap).

The information given in hypernyms.txt was stored in a Digraph structure. Because the integer points given in as pairs/triples/etc. in terms of directional edges between them, it made sense to read in a line of integers at a time, then create an edge between the two points on the digraph. This appropriately shows the directional relationship between the given hypernyms tree, and also gives opportunity for the SAP implementation to come into use.

In order to compute the SAP, I implemented a fairly straightforward algorithm involving the methods provided in Digraph and the properties of the Breadth-First (Directed) Search algorithm. The main points of my algorithm involve first initializing a Digraph (with values sourced from the given text file), and then initializing an SAP object. From this, another text file is read and two points are used to determine the shortest ancestral path between the two. The shortest path's length and the ancestor in question are printed, and the next two points are read in from the input scanner. The ancestor method which determines the common ancestor between the two values is implemented by using the BreadthFirstDirectedPaths (BFDP). Two BFDP objects were created with each node value of the graph in question as parameters. After this, the values of paths from 0 to the maximum number of nodes in the digraph are compared in each BFDP; that is, essentially, the nodes that each parameter has a path to is placed in an intersectional set. After this set is created, the ancestor with the shortest length is chosen and returned. The length calculated in the ancestor method was determined by using the sum of the distTo method included in the BFDP class from the ancestor and the two parameters. The length method in SAP uses the BFDP class as well; the BFDP objects are initialized for each parameter, the ancestor method called, and the sum of the distTo the ancestor is returned.

In order to calculate the shortest ancestral path, the Breadth First Search algorithm was used, which is the most time expensive operation used with the graph structure in my project. The complexity of the Breadth First Search (for directed graphs) is O(V + E), where V is the number of vertices and E is the number of edges in the given digraph. The breadth first search is used twice for each time the ancestor method is called; the ancestor method is called twice for each set of vertices we are interested in as specified by the input file, once to just determine the ancestor, and once to determine the length of the path between the two vertices including the common ancestor. Two additional breadth first searches were also used in the length method. Therefore, in the worst case, the run time is ~6* (V + E) → O(V+E). The best case running time for this algorithm would be constant time, O(1), which would occur when the given vertex has no edges to other vertices.