Describe the data structure(s) you used to store the information in synsets.txt. Why did you make this choice?
Answer: HashTable.

Using the identity of hashtable's "map keys to values", I was able to use two hashtables to store nouns and synset by their noun ID number, which can be **quickly access** with their ID, for example, get synset using ancestor ID. Moreover, I parse the value(nouns) into strings that separated by comma, which can be easily separated by String.split() later in the printSap function. There are other data structure can be implemented (like symbol table), because I did not fully utilize hash function from the hashtable due to the fact that I do not really need to.

Describe the data structure(s) you used to store the information in hypernyms.txt. Why did you make this choice?
Answer: I did not use any special data structure to store hypernyms. I simply used Digraph(int) to store all the hypernyms. Which uses the Bag<>[] to store adjacency list of vertex. By using Bag, for each vertex, it can store all the edge to it using the id number as reference to vertex, then use the value of hypernyms to add edge to it.

Describe your algorithm to compute the SAP. What is the worst-case running time as a function of the structure of the graph (height, number of vertices, number of edges, etc)? Best case running time?
My algorithm assumes there is a root vertex that has no links leaving it (i.e. no ancestors), finds it, and finds the paths from v and w to it. As it finds the vertices on the way to the root vertex (using BFS), the algorithm checks to see if the paths from v to root and w to root have any shared vertices, Once all the possible scenarios are considered (i.e. v or w being the common ancestor), the Ancestor with the smallest distance sum is returned.

Worst case: O(E + V + ln(E))

Best case: O(E + V)