```java
package Assign_4;

/** This class is an implementation of the WordNet class for this word-embedding program.
 * Micah Rose-Mighty
 * 6498935
 * 2020-12-04
 * Created using IntelliJ
 * Note: when running this you must be patient for the module data to be produced. Sorry
 * The program runs alot faster once the find_module line is commented out of the main method but i left it in for marks
 */

import java.io.*;
import java.util.*;
import java.util.stream.Collectors;

public class WordNet {

    private int size = 3000;
    private boolean A[][];
    private double vectors[][];
    private double W[][];
    private String words[];
    private int N;
    private Map<String, Integer> vertexIndex;
    private int M = 50;
    private double threshold = 3;




    WordNet(String fileName){ //This clause of code creates a new graph and initializes all instance variables
        A = new boolean[size][size];
        W = new double[size][size];
        words = new String[size];
        vectors = new double[size][M];
        N=size;
        vertexIndex = new HashMap<>();
        read_data(fileName);
        calculateEuclideanDistance();
    }

    public void addEdge(Integer i, Integer j){ //adds a new edge to the graph manually if needed
        A[i][j] = true;
        A[j][i] = true;
    }

    public void removeEdge(Integer i, Integer j){ // removes an edge from the graph manually if needed
        A[i][j] = false;
        A[j][i] = false;
    }

    public Integer size(){ //returns the graph size
        return this.size;
    }

    public boolean existEdge(Integer i, Integer j){ //checks if there exists an edge
        return A[i][j];
    }

    public String toString(){ //Displays adjacency matrix of the graph although the method for doing that in the main method has been commented out due to the amount of time and space it compromises.
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < size; i++){
            s.append(i + ": ");
            for (boolean j : A[i]){
                s.append((j?1:0) + " ");
            }
            s.append("\n");
        }
        return s.toString();
    }

    int minDistance(double path_array[], Boolean sptSet[])  {
        double min = Double.MAX_VALUE;
        int min_index = -1;
        for (int v = 0; v < size; v++)
```

```java
            if (sptSet[v] == false && path_array[v] <= min) {
                min = path_array[v];
                min_index = v;
            }

        return min_index;
    }

    void printMinpath(double path_array[])  { // prints array of distances if needed
        System.out.println("Vertex # \t Minimum Distance from Source");
        for (int i = 0; i < size; i++)
            System.out.println(i + " \t\t\t " + path_array[i]);
    }

    void printMinPathTo(double path_array[], int to){
        System.out.println(path_array[to]);
    }

    void algo_dijkstra(int src_node, int to_node) { //Dijkstra's Algorithm for the graph (adjacency matrix)
        double path_array[] = new double [size]; // output array with dist[i] holding the shortest distance from src to i

        Boolean sptSet[] = new Boolean[size]; //spt (shortest path set) which contains the vertices that have the shortest path.
        //Before the process is run all the distances are set to infinity and sptSet[] is false
        for (int i = 0; i < size; i++) {
            path_array[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        path_array[src_node] = 0; //Path between any vertex and itself is always 0

        for (int count = 0; count < size - 1; count
++) { //for loop to find shortest path for vertices
            int u = minDistance(path_array, sptSet); //must use minDistance helper method to find the vertex with smallest distance
            sptSet[u] = true; //shows that current vertex u has already been processed

            for (int v = 0; v < size; v++) //for loop to process the nodes adjacent to the current vertex
                if (!sptSet[v] && A[u][v] != false && path_array[u] != Integer.MAX_VALUE && path_array[u] + 1 < path_array[v])// if vertex v isn't in the sptSet already then we must update it
                    path_array[v] = path_array[u] + 1;
        }

        printMinPathTo(path_array, to_node); //print the path array
    }

    public void printShortestDistanceBFS(int s, int dest){
        int pred[] = new int[size]; //stores predecessor of i
        int dist[] = new int[size]; //stores distance of i from s

        if (BFS(s, dest, pred, dist) == false) {
            System.out.println("Given source and destination" + "are not connected");
            return;
        }

        LinkedList<Integer> path = new LinkedList<Integer>(); //Linked List to store the path
        int crawl = dest;
        path.add(crawl);
        while (pred[crawl] != -1) {
            path.add(pred[crawl]);
            crawl = pred[crawl];
        }

        System.out.println("Shortest path length is : " + dist[dest]); //print distance

        System.out.println("Path is:"); //print the path
        for (int i = path.size() - 1; i >= 0; i--) {
            System.out.print(words[path.get(i)] + " ");
        }
    }

    public boolean BFS(int src, int dest, int pred[], int dist[]) { //BFS algorithm using LinkedList of Integer type
        LinkedList<Integer> queue = new LinkedList<Integer>(); //queue to maintain order of vertices whose adjacency list is to be scanned
        boolean visited[] = new boolean[size]; //array that stored information whether a vertex has been visited in BFS

        for (int i = 0; i < size; i++) {
            visited[i] = false; //at first all vertices are unvisited
            dist[i] = Integer.MAX_VALUE; //at first all distances are also infinite
            pred[i] = -1;
        }
        visited[src] = true; //src is first vertex to be visited
        dist[src] = 0; //distance of src to it self is 0 of course
        queue.add(src);

        while(!queue.isEmpty()) { //BFS algortithm
            int u = queue.remove();
            List<Integer> neighbours = getNeighbors(u);
            for (int i : neighbours) {
                if (visited[i] == false) {
                    visited[i] = true;
                    dist[i] = dist[u] + 1;
                    pred[i] = u;
                    queue.add(i);

                    if (i == dest) // closing condition once destination is found
                        return true;
                }
            }
        }
        return false;
    }


    public List<Integer> getNeighbors(int vertexIndex){ //method to find vertices that are adjacent to current vertex
        List<Integer> neighbours = new ArrayList<>();
        for(int i = 0; i < A.length; i++){
            if(existEdge(i, vertexIndex))
                neighbours.add(i);
        }
        return neighbours;
    }

    public int printTotalEdges(){ //method to print total number of edges if needed
        int total = 0;
        for (int i = 0; i < A.length; i++){
            for (int j=i+1; j < A[i].length; j++){
                if (existEdge(i, j))
                    total++;
            }
        }
        return total;
    }

    void DFSUtil (List<Integer> path, int v, boolean visited[]){ //Helper method to assist in DFS traversal
        visited[v] = true; //mark current node as visited and print it
        path.add(v);
        List<Integer> i = getNeighbors(v);
        for (int n : i){ //recur for all adjacent vertices
```

```java
            if (!visited[n])
                DFSUtil(path, n, visited);
        }
    }

    int DFS(int v){ //Method for DFS traversal that uses DFSUtil recursively as a helper method
        boolean visited[] = new boolean[size];
        List<Integer> path = new ArrayList<>();
        DFSUtil(path, v, visited); //recursive call to DFSUtil helper method to print DFS traversal
        return path.size();
    }

    public List<Integer> find_modules(){ //method to find all different modules within the graph
        Set<Integer> set = new TreeSet<>();
        for (int j = 0; j < A.length; j++){
            int length = DFS(j);
            set.add(length);
        }
        return set.stream().sorted(Comparator.reverseOrder()).limit(20).collect(Collectors.toList());
    }
    public static void main(String args[]){ //main mehtod to actually run the processes of this program
        WordNet g = new WordNet("wordvector"); // create graph instance and specify file which data will be read from
        System.out.println("Total Edges: "+g.printTotalEdges());
        System.out.println("Number of vertices in graph: "+g.size());
        //System.out.println("Here is the graph of adjacency matrix: \n" + g.toString());
        System.out.println("Top 20 modules and their sizes: " + g.find_modules());
        String src = "money"; //can change source word to any word within the file
        String target = "future"; //can change target word to any word within the file
        System.out.println("Source: "+src+"\nTarget: "+target);
        System.out.println("\nBFS Method");
        g.printShortestDistanceBFS(g.vertexIndex.get(src), g.vertexIndex.get(target));
        System.out.println("\nDjikstra Method");
        System.out.println("Shortest Path Length");
        g.algo_dijkstra(g.vertexIndex.get(src), g.vertexIndex.get(target));
    }

    public int read_data(String fileName){ //method to read the appropriate data and correct lines within the file
        int wordCount = 0;
        try(BufferedReader reader = new BufferedReader(new FileReader(fileName))){

            for(int i=1; i <= 3100; i++){
                if(i>=101){
                    String row = reader.readLine();
                    String content[] = row.split(" ");
                    int size = content.length-1;
                    words[wordCount] = content[0];
                    vertexIndex.put(content[0], wordCount);
                    for (int vec = 0; vec < size; vec++){
                        vectors[wordCount][vec] = Double.parseDouble(content[vec + 1]);
                    }

                    wordCount++;
                }
            }
        } catch (FileNotFoundException notFoundException){
            notFoundException.printStackTrace();
        }catch (IOException ioException){
            ioException.printStackTrace();
        }
        return wordCount;
    }

    public void calculateEuclideanDistance(){ //
```

```java
Method to calculate euclidean distance between vertices
        for(int i = 0; i < words.length-1; i++){
            for (int j =i+1; j < words.length; j++){
                double distance = 0;
                for (int m=0; m < M; m++){
                    distance+=Math.pow(vectors[i][m] - vectors[j][m], 2);
                }
                distance = Math.sqrt(distance);
                W[i][j] = Math.min(distance, threshold);
                W[j][i] = Math.min(distance, threshold);
                A[j][i] = distance < threshold;
                A[i][j] = distance < threshold;

            }
        }
    }
}

```