createGraph() Write-Up

The method createGraph() accepts a streamreader parameter named rdr. This streamreader is used later within the method to read the text datafile. My createGraph() method begins with two variable declarations, continues with a try catch block, and finishes with a final initialization of the adjacency array. This method starts off by declaring a variable of type string named “line.” This string will be used to hold all of the contents on the current line within the datafile. The next variable declaration of type array named “s,” is used to hold each item within the string variable, line. Next, createGraph() makes use of a try catch block. Within the try, we start off by declaring an integer variable named k; whose scope is limited only within the try block, and is initialized to 0. Next, it is stated that the following while loop will parse through the entire datafile using the streamreader, rdr. Following this, string line is declared to the current line’s contents within the text datafile. Each item within string line is then “split” into an array, each split declared by a “ “; array s is then declared to contain the contents of the “split” string line. The system proceeds to ignore the contents of the first line within the datafile, increments integer k by one, and uses the “continue” keyword to move onto the next line. Next, the system checks to see if k is equal to one. If this returns true, then that signifies that the system is on the second line of the datafile. The system will then take the first item on the second line, convert it from type string to integer, and sets the global variable numVertex to this value. The global adjacency array is then initialized to be a square matrix of size adjacency. Every value within adjacency is then declared to be zero. The system will then increment integer k by one to signify that it is on the next line. The system will then make use of the “continue” keyword to actually move onto the next line. Next, the system checks, and proceeds to ignore the contents of the third line within the datafile, increments integer k by one, and uses the “continue” keyword to move onto the next line. Finally, the system will then parse through the rest of the data file, each parse will use the peek() function to make sure the system is not on the last line of the data file. For every parse that is not on the last line, integers x, y, and result are declared and initialized. Note, the scopes of these items are contained only to this parse. Integer x is declared to be the first number within the data file. Integer y is declared to be the second number within the data file. Integer result is declared to be the third number within the data file. Note, integers x,y, and result, convert the string value into type integer. Finally, array adjacency at x,y is then set to be an integer result. The system will then “continue” onto the next line, until the peek() function returns the last line. Once every item within the text datafile is parsed through, the system parses through adjacency, and replaces every zero that is not on the diagonal with the INFINITY constant. The stream reader closes, and the try block exits successfully. However, if at any point within the try block there was an error, the try block exits. The catch block then takes the error, and uses the console to notify the user of this specific error. There is no more to the createGraph() method after this.

Floyd() Write-Up

The Floyd() method begins by initializing global variables of type array named D and P. Both D and P are initialized to be a square matrix of size global integer numVertex. Every value of P is then initially declared to be -1, while every value of D is initially declared to the corresponding value within the adjacency array. Next, the system parses through a nested triple for-loop to simulate the Floyd algorithm. The first for-loop uses integer k, the second for-loop uses integer i, and the third for-loop uses integer j. Integers k, i, and j parse through numbers 0 to one less of numVertex, has an exiting condition of when the associated integer equals or is greater than numVertex, and each integer increments up by one. The system then checks to see if the sum of array D at [j k] and array D at [k j] is less than the value of array D at [i j]. If this returns true, then array P at [i j] is set to the current value of integer k, and array D at [i j] is set equal to the sum of array D at [i k] and array D at [k j]. Once this nested triple for-loop exits, the system outputs the final D matrix. In the output, the system replaces any item at the constant INFINITY with string “oo”. Next the system will output all values within the P matrix. Finally, every value within the P matrix has the path() function called upon it, and the proper output is printed.