

Assignment 3: Sorting, Searching & Graphs in Java

Summary

For the first assignment a simple program was produced by the group based on record retrieval to illustrate how sequential search and bubble sort works. The records used to search and sort on were implemented using linked list. The sorting was performed alphabetically.

To compare the speed of our bubble sort algorithm to other sorting algorithms, a program was created to calculate the difference in the sort time of the bubble sort compared to an algorithm taken from literature. The number of comparisons performed during the search was also calculated and compared.

A graph abstract data type was implemented for the third section of the assignment. Firstly we implemented it using a linked list to represent the nodes in the graph and a linked list to represent the links for each node. Then it was implemented using an array to represent the nodes and again linked list to represent the links for each node.

Then we used the Graph abstract data type to help us find the shortest path from city A to city B, using the methods requested in the assignment. Using depth first algorithm, we were able to return all possible (non-looped) paths from A to B. And if all links had been given weights the shortest path from A to B may also be calculated.

To help test the Graph abstract data type, a test program was provided, based on a menu system, that allowed the user to create a graph, specify the links for each node (if required), specify names for the nodes, specify weights for the links, and calculate the shortest path from node A to B.

Report Research

A number of sources were used to referred to when researching how to write a report like this one. Sources that we used were typically;

The Internet,

Assignment 1: Java Refreshed, Assignment 2: Stacks, Queues and Linked Lists in Java, and the guide to report writing in the back of engineering lab scripts.

Internet sites that were browsed included:

<http://fbox.vt.edu:10021/eng/mech/writing/>,
<http://www.msue.msu.edu/aee/dissthes/guide.htm>,
<http://www.neltec.com/scifair/report.htm>

Table of Contents

Page	Description
2	<i>Summary</i> (Ioan), <i>Report Research</i> (Ian)
3	<i>Table of Contents</i>
4	<i>Introduction</i> (Ioan)
5-24	<i>Program Design</i> (see pages)
25-47	<i>Testing</i> (see pages)
48	<i>Conclusions</i> (Gareth), <i>Bibliography</i> (Ioan)
49-100	<i>Appendix (Source Code, MS-DOS Output)</i> (see pages)

(The names in subscript text denote the *author* of the relevant section.)

Introduction

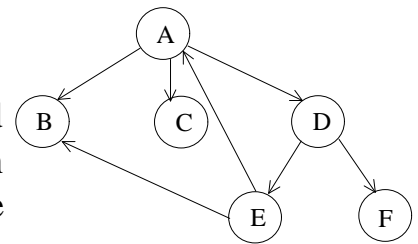
Searching and sorting is a common part of our everyday life, most of the time we do it without even knowing it. From deciding what to wear in the morning, flicking through TV channels deciding what to watch.

Most of the time, the time we take to sort and search for something is not all that important, but the time the computer takes to sort and search may be of crucial importance. The amount of data computers deal with are sometimes monumental, and the time to sort or search data may vary enormously when using different algorithms. This is why a great deal of time is spent developing more and more complicated and efficient algorithms to sort or search in the quickest time possible.

During this assignment we see how different sort and search algorithms operate, and compare the difference in speed and efficiency between different kinds of search algorithms. In the last exercise, we will implement a search algorithm (depth-first type search) that will traverse through a Graph abstract data type.

Graphs

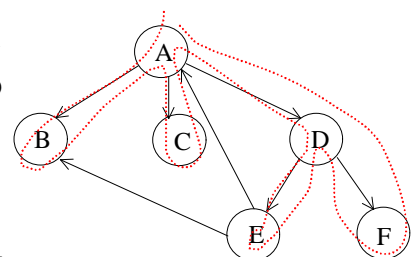
A graph is basically a collection of nodes which are connected by some links. They are seen in such applications as fuzzy set theory in mathematics, map searching and constructing family trees. An example of a directed graph (digraph) is shown on the right. For this assignment we will be only considering directed graphs.



In a computer there are various ways of representing graphs as data structures. In this assignment we will hold the collection of links for each node in a linked list, and hold the nodes themselves either in a linked list (exercise 3) or in an array (exercise 4).

Once exercise 3 & 4 are complete, we then will go on to use the graphs in a practical manner- using them to find the shortest route between two cities A and B, as stated in the question. To do this we will give every link a weight or a value, which will represent the distance between two nodes in the graph. We will then transverse through the graph, accumulating the distance traveled so far, while looking for our destination node. There may be more than one possible routes to the destination node. Therefore all possible paths will be considered to the destination node, and at the end the shortest path between the start and end node will be calculated.

The transversal will be based on a depth first algorithm, that is we follow the first link on each node we encounter until we find a node with no links or only links to nodes we have already visited. We then backtrack and follow the second link on the previous node, and so on until all node have been visited. The diagram on the right shows a possible route through a graph when performing a depth first search.



Design

Exercise 1: Searching and Sorting Family Records

Code: Gareth | Report: Gareth | Flow Chart: Ian

Initial Design Ideas:

We Need a Student class which will contain the following fields:

First Name,
Family Name,
Student Record Number.

We Need methods in the above class file which will:

Alter Information (set methods)
Read Information (get methods)

We Need a main class which will

- allow us to create a new record
- allow us to display all the records in the system
- allow us to sort the records in the system
- allow us to search for information regarding a student

The above will be based on a menu system

The system will be implemented using linked lists

We need the following methods in the linked list implementation:

- Method for adding an element to the linked list
- Method for swapping two elements of the linked list

There is no need to have a method for deleting an element (Not specified in the question).

Let us start by analysing the Student class, which provides a Student object to other classes with the following attributes: *two Strings to hold information (surname and forename); and an ID integer as a kind of index*. All these attributes were declared private, meaning that methods were needed to access the data, such as the two methods shown below to “*get*” and “*set*” information about a Student’s surname.

```
public String getSurname()
{
    return surname;
}

public void setSurname(String input)
{
    surname = input;
}
```

The second class needed was the Node class, and this was based on the code created in the previous assignment. We needed a Node class because we decided to implement the record retrieval system using a linked list, because of the variable size of the database and initial small size.

As in the previous assignment, each Node object consists of a pointer to the next node, and a data element. Here, the data element used is a Student object. Most of the code in the class is taken directly from the previous assignment, such as the constructor, the insert method and the nextNode method. As these methods have been covered in the previous assignment, I will not explain them here.

The only new method in the class is the `Swap` method, used to swap two elements in the linked list. Because we use a bubble sort, the only swapping we will be doing is to swap an element in the linked list with the **next** one in the list.

The swap method takes *three* input parameters: (a) A node in a particular linked list, (b) an integer used to represent the location of the node we want to swap in the linked list (`start`), and (c) a check integer used to traverse through the linked list. When a swap call is made, we always pass the **first** element of the linked list to the method, and the number **0** as the check integer.

The method first of all has to locate the correct element in the linked list for swapping. It does this by checking to see if the check digit (initially zero) is the same as “start” i.e. the element we want to swap. If it is, then we may swap the elements. If it is not, then we must increment the check digit by one and call the method again, this time passing in the next element of the linked list. This way, we only swap elements when we have reached the correct element of the linked list.

To do the swapping, we use a temporary node, parameter, declared locally at the start of the method. We copy the data from the first Node to the parameter node. We then copy the data from the second node to the first node. Finally, we copy the data from the parameter node to the second node.

```
parameter.data = whatList.data;
whatList.data = whatList.nextElement.data;
whatList.nextElement.data = parameter.data;
```

Finally, to bring all of the above together, we a third class file to provide a menu system and all of its associated methods. I will not concentrate on how the menu system is implemented, just to say that it uses error trapping and calls the appropriate method when the user selects an option from the menu.

There are five options that the user can select from the menu, outlined below.

(1) Create a new record.

The program creates a new node and asks the user to enter information about the Student’s forename and surname. The ID number is set automatically and increments each time after this method is called. We then insert the student record into the linked list, which has already been declared at the start of the class file.

(2) Display all records

The program displays all the records in the linked list vertically on screen. Note that this method is automatically called when the user performs other tasks e.g. inserting an element into the linked list, sorting the records. We display all the records by using a for loop to go through all the records in the linked list and a separate `getRecord` method to return information about a particular node.

(3) Sort all records

The program executes a bubble sort on all the elements in the linked list. As is usual in a bubble sort, we use a double for loop, one to count through all the elements in the list, and the second nested loop to count through all the elements except the last one.

```
for (int i=0; i < numberOfRecords; i++)
{
    for (int j=0; j < (numberOfRecords-1); j++)
    {
```

The purpose of the double for loop is to go through a list of n elements n times, each time moving the greatest element in the list to the back of the list. To move elements through the list, we compare two consecutive elements in the list. If the first element is “greater than” then second element, then we must swap the first element and the second element. In this context we are comparing strings (surnames). We use the `compareTo` method for strings to see if one string is “greater than” another string. If this is the case, then we invoke the `swap` method in the `Student` class, described previously.

```
if ((getRecord(j).getSurname()).compareTo(getRecord(j+1).getSurname()) > 0)
{
    database.swap(database, j, 0);
}
```

Note that the `compareTo` method returns a positive integer if one string is greater than another, zero if the two strings are the same, and a negative integer if one string is less than the other string.

(4) Search for a record

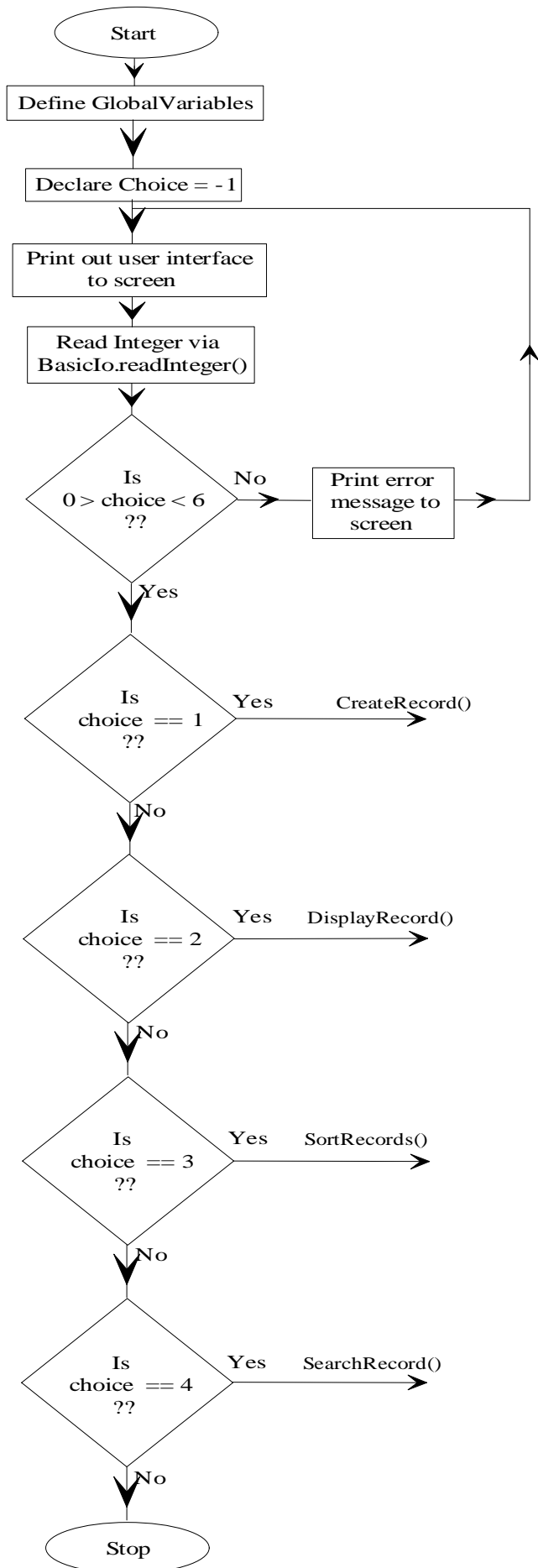
The method starts by asking the user for an ID number to search for. After making sure the number given is in the correct range, and not a string etc., the method goes on to search the Student “database” for information on the ID number in question. It does this by using a for loop to go through all the records in the linked list, and for each record, checks to see if the ID number of the record matches what we are searching for. If this is the case, we print information out about the record that we have just analyzed.

```
for (int i = 0; i < numberOfRecords; i++)
{
    if((getRecord(i).getID()+1) == searchInt)
    {
        System.out.print(getRecord(i).getID()+1);
        System.out.print(": ");
        System.out.print(getRecord(i).getForename());
        System.out.print(" ");
        System.out.println(getRecord(i).getSurname());
    }
}
```

(5) Exit Program

There is no code for this option, when selected the program just terminates.

Main Program for Exercise 1



Exercise 2: Sorting Methods

Code: Ian | Report: Ian | Flow Charts: Ian

Description of Program

This program involves testing out two sorting methods, the first method being one implemented by the user and the second method obtained from a book or some other source. The important factor in testing the program is the number of swaps each method has to carry out before the sorting is completed, therefore we need some kind of counter to keep record of the number of swaps that have taken place.

Program Analysis

This program has a `main()` method controlling the proceedings and other functions will do all the work. This kind of programming technique is called modularisation where the method can be used again by simply pasting them into the appropriate place in you programs.

At the start of the program we have to declare the arrays that the program and it's functions will use and of course declare the global variables that are to be used. We now come to the main method where all the data flow is controlled. First of all we get a prompt to enter the number of elements the arrays are going to posses. The `BasicIo` class is again very busy as usual providing the program with means of input from the keyboard.

After obtaining input the arrays can be created via the `createArray()` method with the aid of the 'arraySize' variable as a parameter. Now we need to enter numbers in the array. The way this is accomplished here is that we use the `arraySize` variable as a loop counter so that we now know how many elements need to be entered into the array. To get an unsorted array for the sort algorithms to use, the numbers were entered into the array highest first,

```
for (int i=0; i < arraySize; i++)
{
    array1[i] = j;
    System.out.println(array1[i]);
    array2[i] = j;
    j--;
}
```

As we can see both arrays will be identical after having received the numbers, and to show the arrays before sorting one of the arrays contents is displayed.

The time has arrived for the arrays to be sorted, the first array being sorted using the user created implementation of the 'BubbleSort' and the second array being sorted by the referenced source code of the 'Quicksort'.

The 'Bubblesort' method is very short and effective, again we observe that the `arraySize` variable is passed to the method so that the user input can be used by the calling method. There are two **for** loops that do all the work in this method, one of them moves the least element to the front and the other actually does the swapping. This is the piece of code that carries out the swapping of array elements,

```
if (array1[j] > array1[j+1])
{
    int temp = array1[j];
    array1[j] = array1[j+1];
    array1[j+1] = temp;
    noOfSwaps++;
}
```

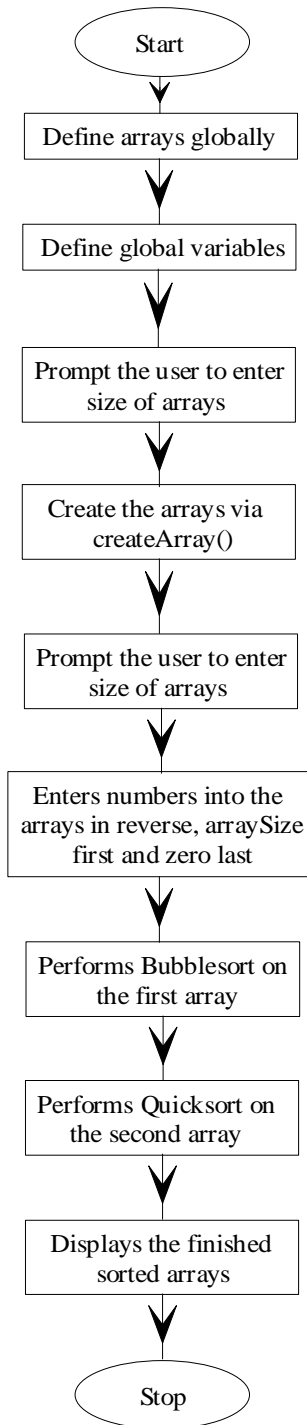
If statement is true, the temp variable will temporarily hold the value of the current element in the array, while the current element is replaced by the next element which it's compared to and then the next element in the array is set to be equal to temp (which of course is current element) and they have been swapped.

The next call in the main method is to perform the sorting for the second array, which is done using the 'Quicksort' algorithm. After this algorithm has finished its task, the results need to be displayed on the screen. Two counts were used to carry this out, each time a swap occurred in the sort methods a count would be incremented (**noOfSwaps** and **noOfSwaps2**). After the displaying of the sorted arrays the number of swaps need to be displayed as well.

There are two print statements that displays the results of how many swaps each sorting algorithm performed.

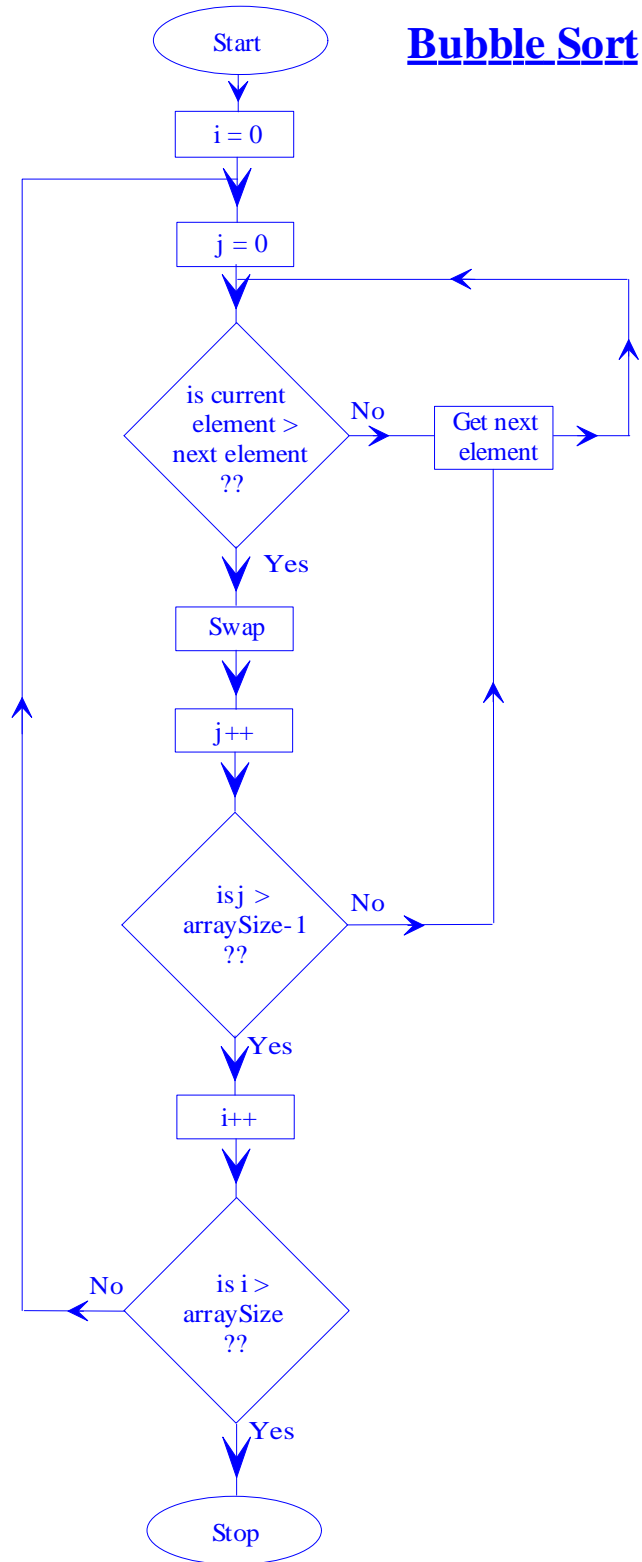
Main Program

Main()



Bubblesort()

Bubble Sort



Exercise 3: Graph Implementation (Linked Lists)

Code: Gareth | Report: Gareth | Flow Charts: Ian

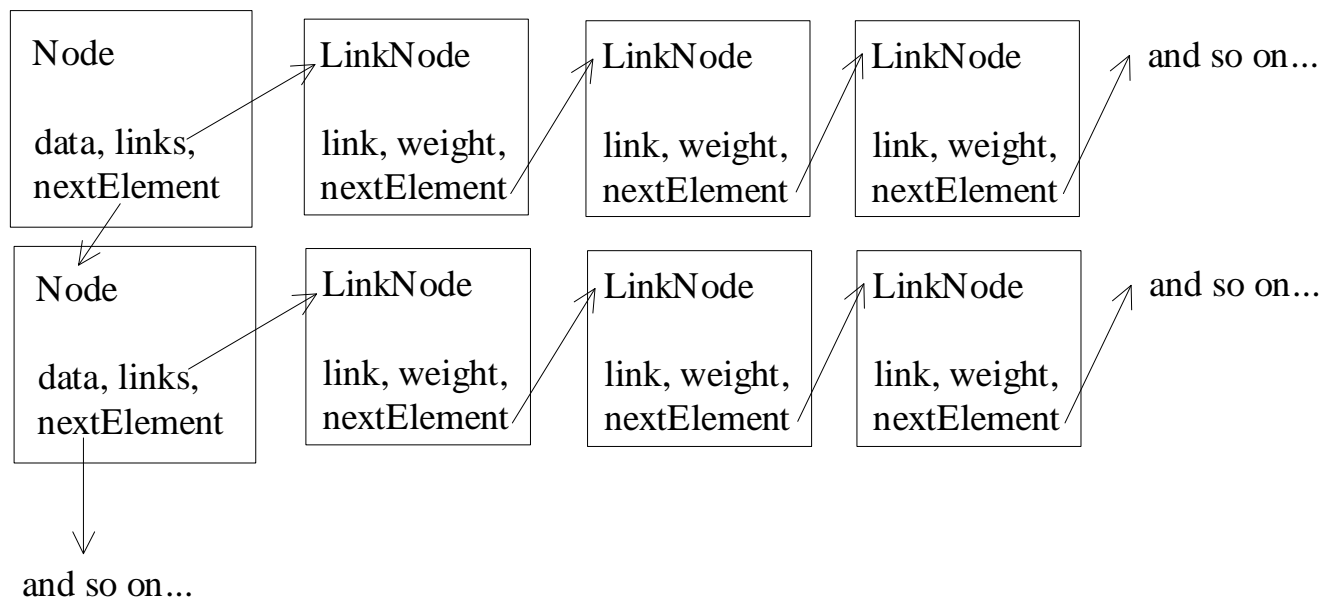
In exercise 3, we are required to implement a Graph abstract data type using a linked node list with linked adjacency lists to represent the arcs. Therefore, we need to create node data types to implement the linked lists. I decided to create two different nodes classes: one to represent the nodes in a graph, and the other to represent the links in a graph. The code was modified from assignment 2, where we first encountered linked lists.

Each node in a graph can have a link to any other node in the graph. We can store the information about the links as a list of numbers, each number representing the destination of a link from a particular node. The list will have a maximum of (N-1) elements, where N is the number of nodes in the graph. The exercise requires that this list of numbers be held in a *linked* list, so the `LinkNode` class was created, each `LinkNode` having the following variables:

```
int link;           (destination node of the link)
int weight;        (the weight of the arc between the nodes)
LinkNode nextElement; (the next link in the list of links)
```

The exercise also requires us to hold the information about the nodes in a graph in a *linked list*. To do this, the `Node` class was created, each `Node` having the following variables:

```
int data;          (holds the identity number of the Node)
LinkNode links;   (holds information about each Node's links)
Node nextElement; (points to the next Node in the graph)
```



Both the `Node` and the `LinkNode` class have methods for *inserting* a new element into the linked list and for *returning* the next element in the linked list. Both also have *constructors* for creating a new element. In addition to this, the `Node` class has a method for assigning a particular list of links (a `LinkNode` object) to the i^{th} node in the linked list.

Now let us look at the `Graph` class, which has to implement four different tasks. Each `Graph` class has two variables: a `Node` data type that stores all the nodes in the graph and an integer variable 'numberOfNodes' to keep track of how many nodes exist in the `Graph` at a particular time. To help us implement the four tasks needed, several other methods were created to simplify the methods and to cut down on the amount of code required.

(1) **The GetNode method**

Given a particular identification number 'i', the `getNode` method returns the links associated with node 'i' to the method that called this method. It does this by using a `for` loop to call the `nextElement` method in the `Node` class 'i' times so that we arrive at the i^{th} element in the linked list. We can then return the links of that particular node.

(2) **The getLink method**

Given a particular list of links (a `LinkNode` object) and an integer 'i', the `getLink` method returns the destination of the i^{th} link in the list of links. If there isn't an i^{th} link, then we return "-1" to indicate that this is so. As above, we use a `for` loop to implement this, this time being careful to avoid the *Null Pointer Exceptions* that occur if there aren't 'i' links in the list .

(3) **The getStrength method**

This is exactly the same method as above, but instead of returning the *destination* of the i^{th} link in a list of links, we return the *strength* or *weight* of the i^{th} link in a list of links.

(4) **The Ask Nodes method**

Whether the user wants to specify the links in the graph manually or have the program create all possible links automatically, the user still has to enter the amount of nodes that are created in the `Graph`. This method allows the user to do this, only allowing the user to enter a positive integer, rejecting any other input and calling itself to ask for correct input. When the user has entered some correct input, the method sets the `numberOfNodes` variable to contain what the user has entered.

(5) **The EnterLinks method**

Used throughout the `Graph` class, this method asks the user to enter an integer between 1 and a preset value passed *into* the `EnterLinks` method. We catch errors by using two methods: (a) We use an `if` statement to test that the input is in the correct numerical range; (b) We use the `try` and `catch` commands to spot any non-numerical input. If any input is invalid, the method calls itself, allowing the user another go at providing some correct input.

(6) **The InsertLink method**

If the user wants to enter the links in the graph manually, then we have to ask the user to specify the *start* node and the *end* node for each link. This method allows the user to do this. It starts off by asking the user to enter two numbers representing two nodes in the Graph. We use the `EnterLinks` method described on the previous page to get these two values. Note that if the user enters the same number twice, then we can't insert a link, as we cannot have a link from one node to itself. In this case we print out an error message at the end of the method

Once we have two valid integers, we may create a link from node 'A' to node 'B'. To do this, we get the list of links for node 'A' by using the `getNode` method. Once we have the list of links, we must check that the link doesn't already exist, as in this case we don't have to create a new link. Therefore, we have to check that the number "B" doesn't exist in any of the `LinkNode` objects in the list of links for node 'A'. To do this, we use the `getLink` method for each of the links in the list, looking for the number B. If we don't find it, then we can insert the link. This is a simple matter of inserting the number 'B' into the list of Links. If we *do* find it, then we print out an explanatory message at the end of the method.

(7) **The ChooseLinks method**

Working with a `while` loop, this method asks the user whether he or she wants to add another *link* into the Graph. If the answer is "yes", then we call the `InsertLink` method above. If the answer is "no", then the while loop terminates and the method terminates. If the answer is something else i.e. some incorrect input, then we ask the user the same question again.

(8) **The CheckExistence method**

Given an integer 'A' and a list of links for a particular node, this method cycles through the list of links, looking to see if a link exists to node 'A'. If a link does exist, then we return the number "1". If a link doesn't exist, then we return the number "0".

(9) **The EnterWeight method**

Suppose that the user wants to change the weight of a particular link from one node to another node. Given the ID numbers of these nodes, this method allows the user to do just that. First of all, we ask the user to enter a valid link weight, using all the usual error trapping techniques described previously.

Once we have a valid integer, we search through the links of the first node, looking for the link to the second node. Once we have found the correct link, we place the user's input in the weight variable of that particular link.

The 9 methods described on the previous couple of pages will allow us to implement quite easily the four methods requested in the assignment script for exercise 3.

(A) **TestGraph**

When this method is called, we want to create a fully connected graph for a user defined number of nodes supplied from the keyboard. To get the said number of nodes, we call the `Ask_Nodes` method, getting back an integer 'input'. Now we need to create 'input' amount of Nodes in our graph, all fully connected to all the other nodes in the graph.

To do this, for each node (from $i=0$ to $input$), we create a new `LinkNode`, denoted `insertLinks`, adding a link to every node except itself.

```
for(int k=0; k < input; k++)
{
    if (i != k) { insertLinks.insert(insertLinks, k); }
}
```

After we have created these links, we insert a new node into the linked list of nodes, which was defined as `TestGraphNode` at the beginning of the `Graph` method, using the code `TestGraphNode.insert(TestGraphNode, i, insertLinks);`

(B) **ReadGraph**

When this method is called, we want to allow the user to enter the number of nodes created and also the links between nodes. As above, we use `Ask_Nodes` to ascertain how many nodes the user wants to create. And again we use a `for` loop to specify the links for each node. However, this time instead of creating the links automatically, we create for each node an empty list of links. After we have done this, we call the `ChooseLinks` method, which allows the user to specify the links in the graph.

```
for (int i=0; i < input; i++)
{
    LinkNode insertLinks = new LinkNode();
    TestGraphNode.insert(TestGraphNode, i, insertLinks);
}
ChooseLinks();
```

(C) **Display** (*WriteGraph* in the assignment script)

In this method we are required to print the nodes and their adjacency lists on screen. For each node in the graph, we do the following: (after printing out the ID number of that particular node)

- Get the list of links associated with the node (using `getNode`)
- For each link, get its destination node and the strength/weight of the link (using `getLink` and `getStorage`)
- If the link exists, print out the destination node
- If the link has been assigned a weight or strength, print out the weight in square brackets, [...].

(D) **AddWeight** (*setArcWeight* in the assignment script)

For this method we are required to allow the user to add a weight to a particular arc or link from one node to another node. To start with, we ask the user to enter the start node and the end node of the particular link the user wants to change the weight of. We do this by using the `EnterLinks` method twice. Note that we must check to see if the two values the user supplies are the same. If they are, then the link will never exist because we cannot have a link from a node to itself

We then get the list of links for the start node by using the `getNode` method. We check to see if a link exists to the end node in the list of links by using the `CheckExistence` method. If the link the user has requested does exist, then we may change the weight of the link by using the `EnterWeight` method described earlier.

```
LinkNode storage = getNode(departure-1, TestGraphNode);
int match = CheckExistence(destination-1, storage);

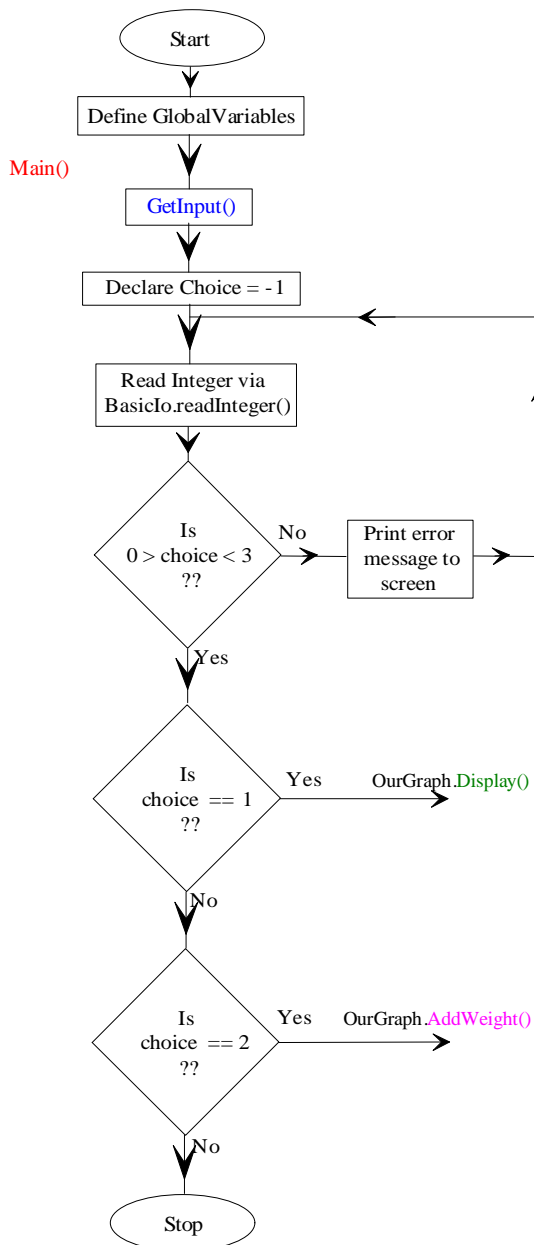
if ((destination != departure) & (match == 1))
{
    EnterWeight(departure, destination, storage);
}
else
{
    System.out.println("That link does not exist on the Graph!");
}
```

To help in debugging and to see if the Graph class was working properly, an example program was created to test the methods of the Graph class. This example program was called `Exercise3`, and executes as follows:

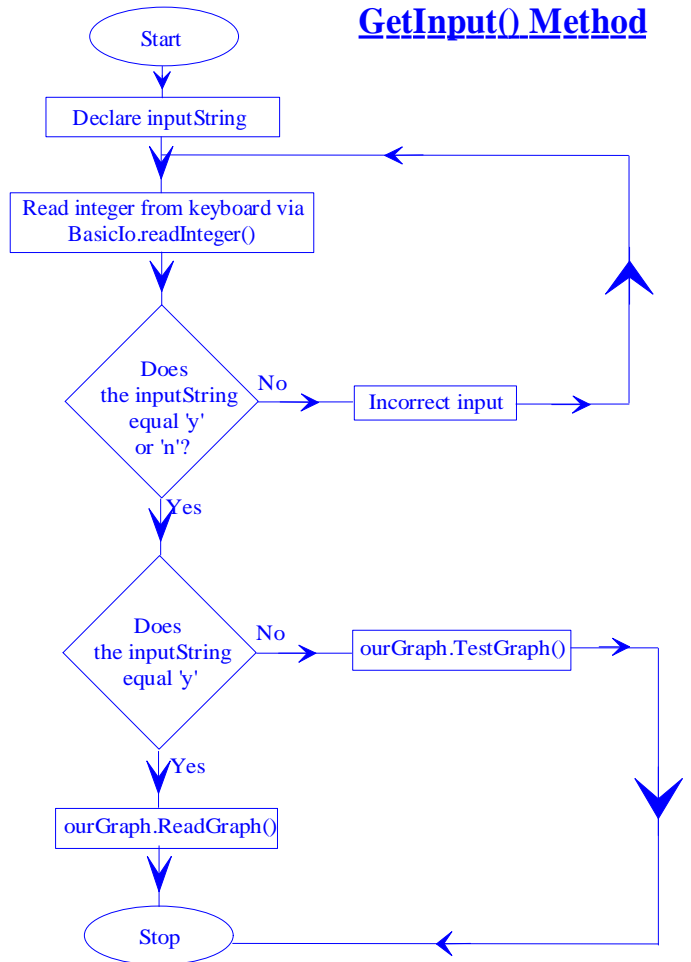
- The program defines a new Graph variable
- The program asks the user whether he or she would like to specify the links in the Graph.
- If the user answers “yes”, then we call the `ReadGraph` method in the Graph class
- If the user answers “no”, then we call the `TestGraph` method in the Graph class
- We then enter a menu system with three options:
 1. Display the Graph
 2. Add (or change) An Arc Weight
 3. Exit Program
- If the user chooses option **1**, we call the `Display` method in the Graph class
- If the user chooses option **2**, we call the `AddWeight` method in the Graph class
- If the user chooses option **3**, the program *terminates*.

Here are some flow charts to see how data flows in Exercise 3.

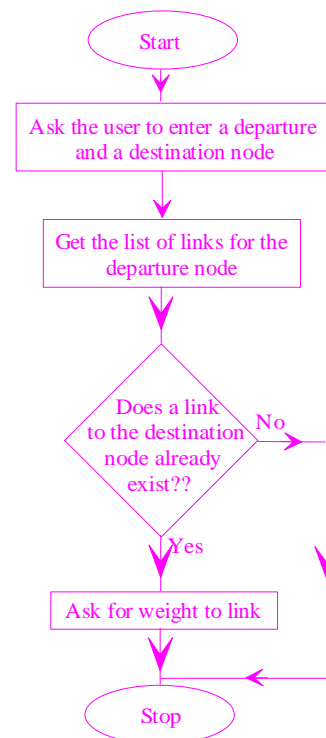
Main Program



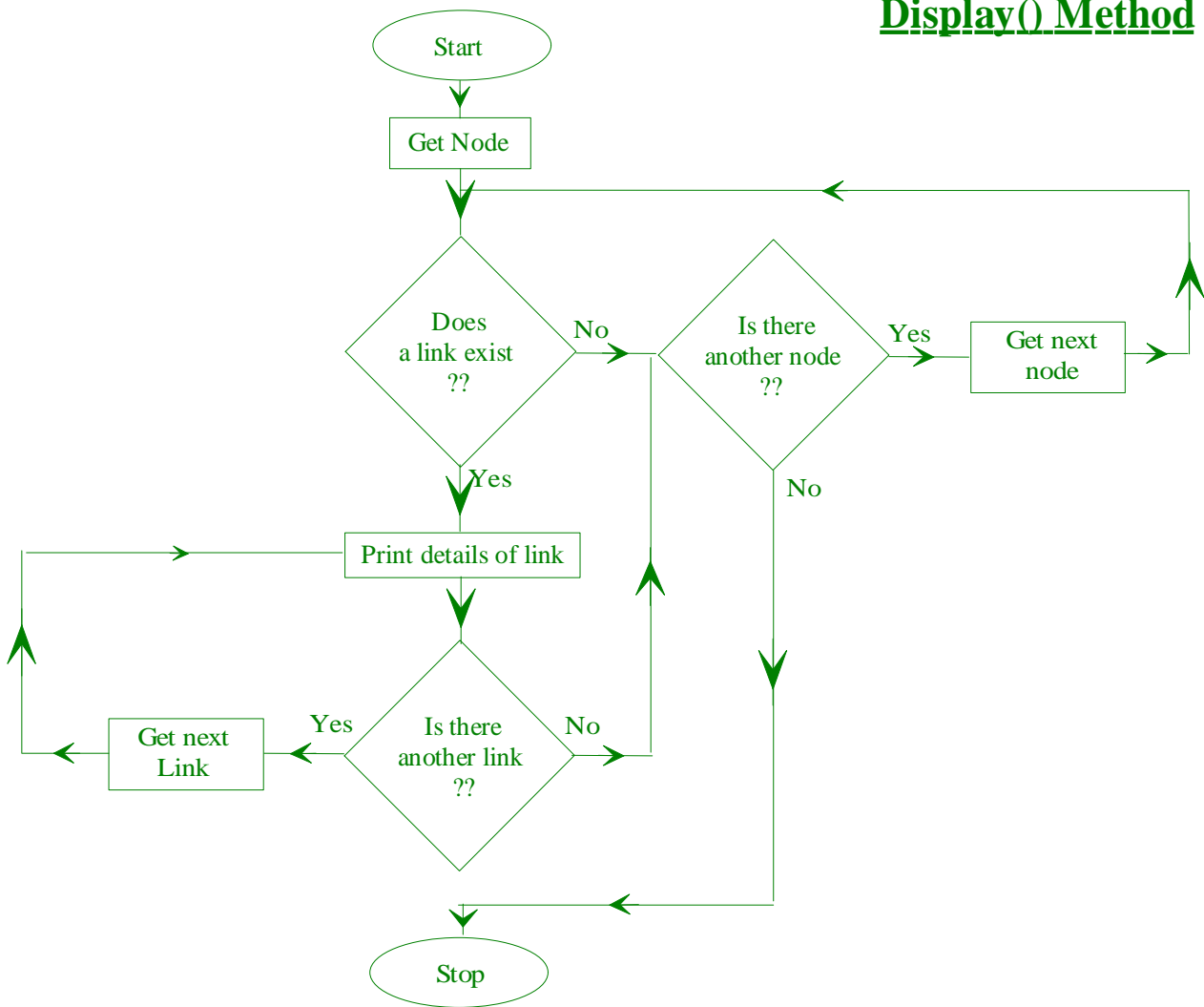
GetInput() Method



AddWeight()



Display() Method

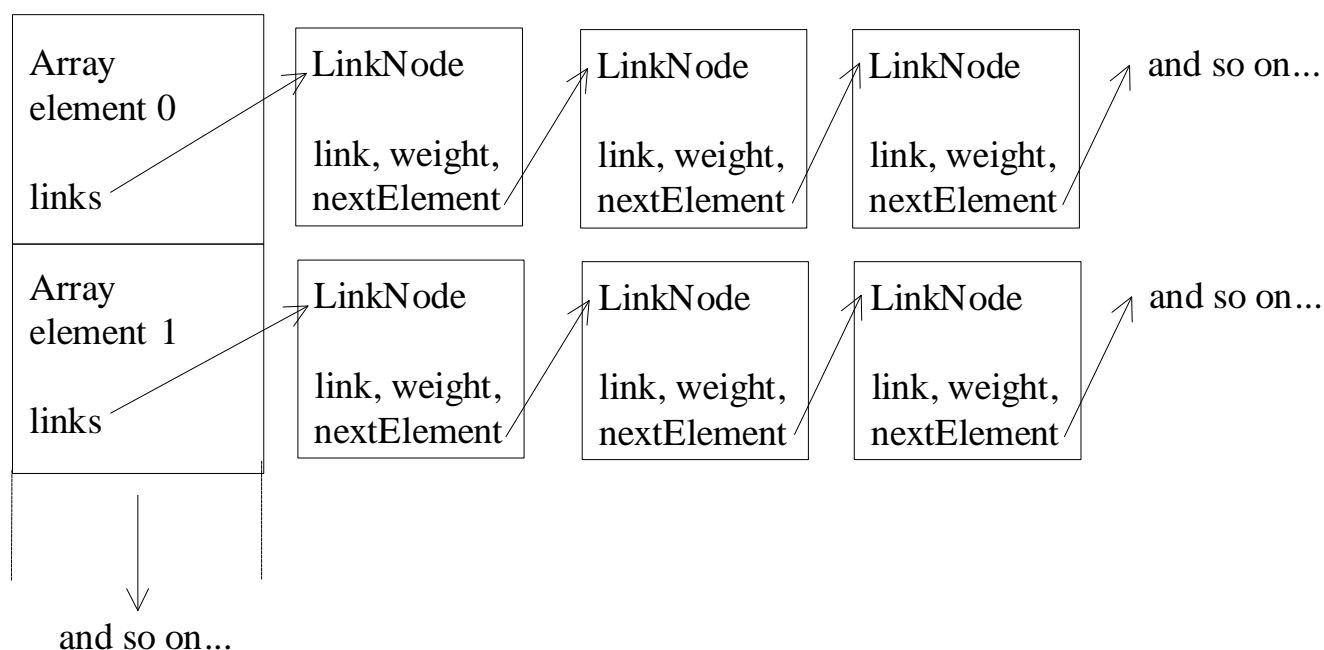


Exercise 4: Graph Implementation (Array, Linked Lists)

Code: Gareth | Report: Gareth

In exercise 4, we are required to implement a Graph abstract data type using a contiguous node list with linked adjacency lists to represent the arcs. The only difference between the code for this exercise and the code for the previous exercise is that we use an **array** to represent the graph's nodes in this exercise and a **linked list** to represent them in the previous exercise. We use the *example* program and the `LinkNode` class without modification.

Because we are not required to *add* any more nodes to the graph once the user has chosen an initial value, this means that we can happily interchange the two implementations with the minimum amount of fuss.



So how do we go about changing the code from exercise 3 to answer the problem posed in this exercise? To start with, we can forget about the `Node` class used in the previous exercise, as we are not representing the nodes of the graph by a linked list any more. This leads to the deletion of the `getNode` method, which returned the links for a particular node in the previous exercise

Analysing the `Node` class used previously, we see that it had three variables:

```
int data;           (holds the identity number of the Node)
LinkNode links;    (holds information about each Node's links)
Node nextElement;  (points to the next Node in the graph)
```

Using an array, we do not need a "nextElement" variable or a "data" variable because we can use array indexing as a way of keeping track of the ID numbers of nodes. The first node in the graph will be array element 0, the second node will be array element 1, and so on. If we declare our array as an array of `LinkNodes`, then we have found a way of storing the links for each node, and therefore we can specify each node completely i.e. we can specify its identification and its links.

So we can replace the red code that was at the start of exercise 3's graph class with the blue code on the right.

```
static Node TestGraphNode = new Node();    static LinkNode[] TestGraphArray;
```

In exercise 3, when we wanted to get the linked list of links for a particular node 'i', we used code such as `LinkNode links = getNode(TestGraphNode, i);` We can now replace this code by something such as `LinkNode links = TestGraphArray[i];` Instead of using the `getNode` method to return the links for a node, we can now get them directly. We shall use this technique throughout the graph class for exercise 4.

Finally, the only other bit of code we have to change is when we actually create the Nodes in the `ReadGraph` and `WriteGraph` methods. The best way of showing how the code changes is to actually view the code side by side.

Exercise 3's TestGraph method

```
public static void TestGraph() throws Exception
{
    int input = Ask_Nodes();

    for (int i = 0; i < input; i++)
    {

        LinkNode insertLinks = new LinkNode();
        for(int k = 0; k < input; k++)
        {
            if (i != k)
            {
                insertLinks.insert(insertLinks, k);
            }
        }
        TestGraphNode.insert(TestGraphNode, i,
                             insertLinks);
    }
}
```

Exercise 3's ReadGraph method

```
public static void ReadGraph() throws Exception
{
    int input = Ask_Nodes();

    for (int i = 0; i < input; i++)
    {
        LinkNode insertLinks = new LinkNode();
        TestGraphNode.insert(TestGraphNode, i,
                             insertLinks);
    }
    ChooseLinks();
}
```

Exercise 4's TestGraph method

```
public static void TestGraph() throws Exception
{
    int input = Ask_Nodes();
    TestGraphArray = new LinkNode[input];
    for (int i = 0; i < input; i++)
    {
        TestGraphArray[i] = new LinkNode();
        LinkNode insertLinks = new LinkNode();
        for(int k = 0; k < input; k++)
        {
            if (i != k)
            {
                insertLinks.insert(insertLinks, k);
            }
        }
        TestGraphArray[i] = insertLinks;
    }
}
```

Exercise 4's ReadGraph method

```
public static void ReadGraph() throws Exception
{
    int input = Ask_Nodes();
    TestGraphArray = new LinkNode[input];

    for (int i=0; i < input; i++)
    {
        TestGraphArray[i] = new LinkNode();
    }
    ChooseLinks();
}
```

As you can see, for each node in exercise 3 we create a new `LinkNode` and insert it into the linked list of Nodes. In exercise 4 we insert the new `LinkNode` directly into the array. Note that in exercise 4 we declare the size of the array immediately after the user specifies it. We do not have to declare the size of the list of nodes in exercise 3 because a linked list has dynamic length.

(To see exactly where the code changes, view the appendix where I have denoted the code that changes in the Graph class from exercise 3 to 4 in blue)

Exercise 5: Finding the Shortest Path

Code: Gareth | Report: Gareth

The purpose of exercise 5 is to modify the code from the previous two exercises to allow us to search through a graph to find the shortest path from A to B. To do this, we need to add another method to our Graph abstract data type. Let us call this method “FindPaths”.

But first, to make the process of entering links and searching much more intuitive for the user, I added another method called “AssignInfo” to the graph class which allows each node in the Graph to be assigned a name. This way the user can see what the links actually mean, and does not have to decipher a list of numbers to get their meaning.

When modifying the code from exercise 3, I added another variable to the Node class called “*information*” which holds the name of each Node in the Graph. To allow the user to see the names entered, I also had to alter the Display method to incorporate the viewing of Node names.

To allow the user to actually assign and change Node names, another option was added to the menu system of the test program which, when chosen, chooses the FindPaths method in the Graph class. Basically, this method asks the user to enter a Node number and then asks the user to assign a name for the node. We then traverse through the node list to find the correct node and add the name to the Node.

```
String name = BasicIo.readString();
Node tempNode = TestGraphNode;
for (int n=0; n < change-1; n++)
{
    tempNode = tempNode.nextElement;
}
tempNode.information = name;
```

When modifying the code from exercise 4, I declared another array at the start of the Graph class, an array of Strings which holds the node information. The array’s size is declared when the user decides on how many nodes are in the graph. Again the display method was modified, and the FindPaths method above was slightly modified in that we do not traverse through the node list as above but simply assign the string to the appropriate position in the array of Strings described above.

Now that we can modify and display node names using the code from either exercises 3 or 4, let us get down to analysing how we may search through the graph for the shortest path from a to b. The technique is exactly the same whether we are using the code from exercise 3 or 4, except for one line where we need to get the links for a particular node. In this case we use the appropriate code for either getting the data from an array or a linked list.

Two variables are declared at the start of the `Graph` class to assist us in finding the shortest path from A to B. One is an integer called `shortestPath` and the other is a `String` called `shortestPathString`. The first variable stores the length of the path at a particular time while the second variable stores the actual path as a `String`.

When the user selects to find the shortest path from one node to another, which is an added option in the menu system of the example program, the `FindPaths` method in the `Graph` class is called. To start with, the method defines an integer array with the same number of elements as there are nodes in the graph.

This integer array will be used to store which nodes we have visited as we search through paths in the graph. The purpose of the array is to prevent looping from happening, and it uses a convention that “-1” indicates that the node has not been visited, while “1” indicates that the node has been visited. Of course, at the start of the method no nodes have been visited so we set all the elements of the array to contain “-1”.

Next, we ask the user which two nodes he or she wishes to conduct a search for the shortest path between the two nodes on. We use the `EnterLinks` method to obtain two integers to represent the start node of the path and the end node of the path. We then initialise a `String` called ‘s’ and place the start node i.e. where we start looking from, into ‘s’.

This `String` variable will be used to hold a description of the partial path we are on. It will also be used to determine whether we have found a complete path, as we can test the last character of the `String` to see if it is the same as the end node we are trying to find. More analysis of this will be conducted later.

We now find the shortest path from the start node and the end node (and print out all valid paths between these two nodes in the process) by calling the `GetPaths` method. This method, described later, takes five input parameters: the node we are going from; the node we are trying to get to; the integer array that holds which nodes we have visited; the `String` “s” described above, and an integer denoting the length of the path so far. At the start, the path has no length so we pass in zero as the last parameter.

After the `GetPaths` method has finished, we print out the shortest path, which has been meanwhile placed in the `shortestPath` and `shortestPathString` variables. Note that we do not do this if the length of the shortest path is *zero*, as this means that no link weights have been assigned.

Finally, just before the method terminates, we reset the variables `shortestPath` and `shortestPathString` to their default values (-1 and an empty `String`) so that we can use them again next time the `FindPaths` method is called.

The GetPaths method

The purpose of the `GetPaths` method is to attempt to find all the valid paths between two nodes A and B, calculating as it finds the paths the shortest path between A and B. Using a depth first search, the method has two distinct sections: In the first section we test if we have reached where we are trying to get to i.e. 'B', and in the second section we carry on with the search, following the **first** path on every node until we find a match or no links. We then backtrack up and follow the **second** path, etc., until all valid paths have been followed ("*valid paths*" means no looped paths)

The method takes five parameters in all. The first two parameters indicate the two nodes involved in the search i.e. the start node and the end node. The third parameter is the integer array used to denote which nodes have been visited previously. The fourth parameter is the String used to describe the path, and the fifth parameter indicates the length of the path so far.

At the start of the method we declare some local variables to use in the method. We set `checkEnd` to hold the length of the descriptor String, and initialise a String (`checkString`) to hold the value of the end node.

The format of the descriptor String is as follows: for each node we visit on our search, we add it to the end of the descriptor String. For example, if we were searching for a path from node 3 to node 2, and have started to search from node 3, then gone to node 1, and then gone to node 2, and found a solution, the descriptor String will now be "1-3-2".

We know we have found a solution path because the last character of the string is the same as the ID of the node we are trying to find. We can use this as a test to see if we have found a solution. So if the last number of the descriptor string is the same as `checkString`, we have found a match.

```
if (description.substring((checkEnd-(checkString.length())),(checkEnd)).equals(checkString))
{
    System.out.println("Match: " + description);
    ....
}
```

When we find a match, we print out the match and the "winning" path as shown above. If the length of the path found is non-zero, we also print out the length of the path. Now we must check to see if the length of the path is shorter than any path we have found before.

So if the variable 'length' that was passed in is less than the global variable `shortestPath`, we change `shortestPath` to store the value of `length`. Note that we must also check to see if `shortestPath` holds the value "-1" as in this case the match we have found must be the first match and we assign the value of `length` to the variable `shortestPath` in any case.

We now go onto the second section of the method, where we continue on our search to find the paths from A to B. To start with, we change the value of the Ath element of the check array to 1 as by executing the code we are analysing, we are visiting node A. We then get the list of links to other nodes associated with node A.

For each link in the list, of which there are a maximum of (numberOfNodes-1) links, we check that a link exists in the ith position of the list. If a link does exist, then we must check to see whether we have visited the destination of that link already. If we have, then we will have a 1 in the check array. If we haven't, the value returned will be -1. Alternatively, if a link doesn't exist in the ith position, we pretend that we have already visited it, to save us searching on the destination of the (nonexistent) link.

```
for (int i=0; i < numberOfNodes; i++) // for every link,
{
    int tempInt = getLink(i, tempList);
    int check = 0;
    if (tempInt == -1) // i.e. link doesn't exist,...
    {
        check = 1; // ... so we pretend that the node has already been visited
    }
    else
    {
        check = remember[tempInt]; // check if the node has already been visited
    }
}
```

So, if the ith link has a valid destination node and we haven't visited it already, then we may call the `GetPaths` method again, this time wanting to search for the paths between the destination node in question and the end node. But, before we do this, we must set up some variables.

To begin, because we are now going to the end node “*via*” this destination node, we must (temporarily) add the ID number of the destination node to our descriptor String. We must do this temporarily because after we return to this piece of code after all the subsequent recursion has finished, we want our descriptor String to be the same as it was before all the recursion went ahead, and not “polluted” by other links.

```
String newDescription = new String();
newDescription = "-" + (tempInt+1);
```

We must also temporarily add to the length of the path in much the same fashion. Note that we can only do this if the strength of the path from the node we are currently on to our destination node has been assigned or set. If it hasn't, then the length of the path remains unchanged. So, after all the variables have been set, we can call the `GetPaths` method again, using the following code.

```
GetPaths(tempInt, b, remember, description+newDescription, parameter);
```

Finally, after all the links for ‘A’ have been analysed, we must reset A’s value in our array denoting which nodes have been visited, back to -1, as we can now validly visit ‘A’ in subsequent levels of recursion without fear of being stuck in a loop.

After `GetPaths` has finished, we will have output to the screen all possible paths from a to b using any intermediate node(s), and also if applicable have calculated the shortest path from ‘A’ to ‘B’.

Testing

All Reports: Ioan

The testing strategy for all programs was as follows: First, after typing in the draft code, we ironed out any errors generated when the program was compiled. These included typing errors such as in `System.out.println("Hello");`, syntax errors such as using the incorrect amount of brackets or not including a semicolon where it should be, and logical errors such as using incorrect variable types.

Next, after the program compiles correctly, we find and correct any logical errors in our code. These would be errors where the program doesn't do what it should, and produces nonsensical or incorrect output. We should also look out for crashes at run time, such as null pointer exceptions or input errors.

Exercise 1: Searching and Sorting Family Records

Method of testing: Black Box Testing

To start with, let us test the menu system used in the program. It should only let us enter the integers from 1 to 5, rejecting any other input

1. Enter an integer out of the range 1-5

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.  
-----
```

```
1. Create New Record  
2. Display All Records  
3: Sort Records  
4: Search for Record  
5: Exit Program
```

```
6  
Sorry, Input is incorrect. Please Try Again.
```

2. Enter some non-integer input, such as a String

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.  
-----
```

```
1. Create New Record  
2. Display All Records  
3: Sort Records  
4: Search for Record  
5: Exit Program
```

```
Ioan  
Sorry, Input is incorrect. Please Try Again.  
-----
```

3. What happens if we just press Enter i.e. provide no input at all

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

- 1. Create New Record
- 2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

Sorry, Input is incorrect. Please Try Again.

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

- 1. Create New Record
- 2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

Now we shall test command 1 in the menu: the Create New Record Section

1. Does the create new record section accept some correct input?

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

- 1. Create New Record
- 2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

1

```
Please enter the Student's forename....Ian
Please enter the Student's surname.....Roberts
```

- 1: Ioan Williams
- 2: Gareth Evans
- 3: Ian Roberts

2. What happens when we try to create a blank record?

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

- 1. Create New Record
- 2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

1

```
Please enter the Student's forename....
Please enter the Student's surname.....
```

- 1: Ioan Williams
- 2: Gareth Evans
- 3: Ian Roberts
- 4:

Answer: We are allowed to create a blank record.

In this part of the testing, we shall see if the program's "Display" method functions properly. It is supposed to return a list of all records currently in the list.

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.  
-----
```

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

2

- 1: Ioan Williams
- 2: Gareth Evans
- 3: Ian Roberts
- 4:

An important part of the program is the search algorithm, a bubble sort. Let us now try to sort the list of names shown above by using command 3 in the menu. As you can see from the output, the list after sorting is properly ordered by surname.

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.  
-----
```

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

3

- 4:
- 2: Gareth Evans
- 3: Ian Roberts
- 1: Ioan Williams

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.  
-----
```

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

Let us now test the Search Record section. There are two searches we must test:

1. A search on an unsorted list.

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

4

Please enter an ID Number to search for more information on...1

Searching for element 1....

1: Ioan Williams

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

2. A search on a sorted list.

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

4

Please enter an ID Number to search for more information on...1

Searching for element 1....

1: Ioan Williams

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

Finally, let us test to see if the program will let us exit

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

5

The Program will now close...

Exercise 2: Sorting Methods

Method of testing: Black Box Testing

To start with, let us test various size of arrays to compare the number of swaps with the BubbleSort and QuickSort.

Please enter the sizes of the arrays: 1

1
The contents of the first Array

Location 0: 1

The contents of the second Array

Location 0: 1
The number of swaps with the BUBBLESORT: 0
The number of swaps with the QUICKSORT: 0

Please enter the sizes of the arrays: 2

2
1
The contents of the first Array

Location 0: 1
Location 1: 2

The contents of the second Array

Location 0: 1
Location 1: 2
The number of swaps with the BUBBLESORT: 1
The number of swaps with the QUICKSORT: 1

Please enter the sizes of the arrays: 5

5
4
3
2
1

The contents of the first Array

Location 0: 1
Location 1: 2
Location 2: 3
Location 3: 4
Location 4: 5

The contents of the second Array

Location 0: 1
Location 1: 2
Location 2: 3
Location 3: 4
Location 4: 5

The number of swaps with the BUBBLESORT: 10

The number of swaps with the QUICKSORT: 2

Please enter the sizes of the arrays: 10

10
9
8
7
6
5
4
3
2
1

The contents of the first Array

Location 0: 1
Location 1: 2
Location 2: 3
Location 3: 4
Location 4: 5
Location 5: 6
Location 6: 7
Location 7: 8
Location 8: 9
Location 9: 10

The contents of the second Array

Location 0: 1
Location 1: 2
Location 2: 3
Location 3: 4
Location 4: 5
Location 5: 6
Location 6: 7
Location 7: 8
Location 8: 9
Location 9: 10

The number of swaps with the BUBBLESORT: 45

The number of swaps with the QUICKSORT: 5

Now we will create two 100 element arrays, this is the output obtained:

The number of swaps with the BUBBLESORT: 4950
The number of swaps with the QUICKSORT: 50

Now we will create two 500 element arrays, this is the output obtained:

The number of swaps with the BUBBLESORT: 124750
The number of swaps with the QUICKSORT: 250

Now we will create two 1000 element arrays, this is the output obtained:

The number of swaps with the BUBBLESORT: 499500
The number of swaps with the QUICKSORT: 500

To summarise the results given from the testing, a table is given below:

Size of array	5	100	500	1,000
Swaps with BubbleSort	10	4,950	124,750	499,500
Swaps with QuickSort	2	50	250	500
Percentage of swaps performed (QuickSort Vs BubbleSort)	20%	1%	0.2%	0.1%

QuickSort was found to be much more efficient when dealing with large arrays, taking much less swaps and therefore much less computing time. Also seen in the table is the percentage of swaps performed, this shows as the array increases the percentage of swaps of QuickSort compared with BubbleSort decreases steeply. To conclude, it is worth the time and effort to produce a more complex, efficient algorithm. When dealing with a very large database search time is of paramount importance.

Exercise 3 & 4: Graph implementation

Method of testing: Black Box Testing

The testing for exercises 3 and 4 is identical because both exercises strive to produce the same program, using different implementations. For this reason, I will amalgamate the testing for exercises 3 & 4 into this one section.

When the program starts, it asks the user to enter the number of nodes in the graph. Let us test this section of the program by entering invalid values.

1. Enter Numerical Input

```
Would you like to specify the links in the Graph?  
Answer 'y' or 'n'...3
```

```
Incorrect Input. Try Again  
Would you like to specify the links in the Graph?  
Answer 'y' or 'n'...
```

2. Enter an incorrect String

```
Would you like to specify the links in the Graph?  
Answer 'y' or 'n'...Test
```

```
Incorrect Input. Try Again  
Would you like to specify the links in the Graph?  
Answer 'y' or 'n'...
```

The program now branches off to two sections, both meeting when we arrive at the menu loop. The first branch is executed when we answer “y” to the question “Would you like to specify the links in the Graph. Let us test this section now.

```
Would you like to specify the links in the Graph?  
Answer 'y' or 'n'...y
```

```
Please enter the number of nodes in the Test Graph...n           (Entering a String)  
Incorrect Input. Please enter a positive integer
```

```
Please enter the number of nodes in the Test Graph...-1        (Entering a -ve integer)  
Incorrect Input. Please enter a positive integer
```

```
Please enter the number of nodes in the Test Graph...0         (Entering zero)  
Incorrect Input. Please enter a positive integer
```

```
Please enter the number of nodes in the Test Graph...5         (Entering correct input)
```

```
Node 1 has links to  
Node 2 has links to  
Node 3 has links to  
Node 4 has links to  
Node 5 has links to
```

```
Would you like to insert a link?  
Answer 'y' or 'n'
```

We are now given the option to insert some links into the Graph. At each stage, the program displays the current status of the Graph. Let us now test this section.

Would you like to insert a link?

Answer 'y' or 'n'

j

(Enter String input)

Incorrect Input. Try Again

Node 1 has links to

Node 2 has links to

Node 3 has links to

Node 4 has links to

Node 5 has links to

Would you like to insert a link?

Answer 'y' or 'n'

0

(Enter numerical input)

Incorrect Input. Try Again

Node 1 has links to

Node 2 has links to

Node 3 has links to

Node 4 has links to

Node 5 has links to

Would you like to insert a link?

Answer 'y' or 'n'

y

(Enter correct input)

From which node does the link go from?

Enter a number between 1 and 5

6

(Enter value out of range: +ve)

Incorrect Input. Please enter a positive integer between 1 and 5

-1

(Enter value out of range: -ve)

Incorrect Input. Please enter a positive integer between 1 and 5

1

(Enter correct input)

From which node does the link go to?

Enter a number between 1 and 5, excluding 1

2

Node 1 has links to 2

Node 2 has links to

Node 3 has links to

Node 4 has links to

Node 5 has links to

Would you like to insert a link?

Answer 'y' or 'n'

Y

From which node does the link go from?

Enter a number between 1 and 5

4

From which node does the link go to?

Enter a number between 1 and 5, excluding 4

4

(Enter the same node twice)

Cannot have a link from a node to the same node.

Node 1 has links to 2
Node 2 has links to
Node 3 has links to
Node 4 has links to
Node 5 has links to

Would you like to insert a link?
Answer 'y' or 'n'
y

From which node does the link go from?
Enter a number between 1 and 5
5

From which node does the link go to?
Enter a number between 1 and 5, excluding 5
8 (Enter incorrect input)
Incorrect Input. Please enter a positive integer between 1 and 5
2

Node 1 has links to 2
Node 2 has links to
Node 3 has links to
Node 4 has links to
Node 5 has links to 2

Would you like to insert a link?
Answer 'y' or 'n'
y

From which node does the link go from?
Enter a number between 1 and 5
3

From which node does the link go to?
Enter a number between 1 and 5, excluding 3
4

Node 1 has links to 2
Node 2 has links to
Node 3 has links to 4
Node 4 has links to
Node 5 has links to 2

Would you like to insert a link?
Answer 'y' or 'n'
n

MAIN MENU - PLEASE ENTER YOUR CHOICE.

-
1. Display the Graph
 2. Add (or change) An Arc Weight
 - 3: Exit Program

Having reached the menu system for the first branch, let us now go back and test the code for when we answer “n” to the question “Would you like to specify the links in the Graph?”

Would you like to specify the links in the Graph?
Answer 'y' or 'n'...n

Please enter the number of nodes in the Test Graph....-3 (Enter a -ve integer)
Incorrect Input. Please enter a positive integer

Please enter the number of nodes in the Test Graph....Test (Enter String input)
Incorrect Input. Please enter a positive integer

Please enter the number of nodes in the Test Graph....6 (Enter correct input)

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

Let us now test the menu system of the program.

Branch 1

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

4
Sorry, Input is incorrect. Please Try Again.

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

-1
Sorry, Input is incorrect. Please Try Again.

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

Test
Sorry, Input is incorrect. Please Try Again.

1
Node 1 has links to 2
Node 2 has links to
Node 3 has links to 4
Node 4 has links to
Node 5 has links to 2

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 5

6

Incorrect Input. Please enter a positive integer between 1 and 5

(Enter value out of range)

0

Incorrect Input. Please enter a positive integer between 1 and 5

(Enter value out of range)

Test

Incorrect Input. Please enter a positive integer between 1 and 5

(Enter String input)

4

(Enter correct input)

From which node does the link go to?

Enter a number between 1 and 5, excluding 4

5

(Enter a non existent link)

That link does not exist on the Graph!

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 5

2

From which node does the link go to?

Enter a number between 1 and 5, excluding 2

6

Incorrect Input. Please enter a positive integer between 1 and 5

(Enter value out of range)

-1

Incorrect Input. Please enter a positive integer between 1 and 5

(Enter value of range)

Test

Incorrect Input. Please enter a positive integer between 1 and 5

(Enter String input)

4

(Enter correct input)

That link does not exist on the Graph!

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 5

1

From which node does the link go to?

Enter a number between 1 and 5, excluding 1

2

Enter the strength of the link (must be positive)

-5

Incorrect Input. Please try again

(Enter invalid link strength)

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 5

1

From which node does the link go to?

Enter a number between 1 and 5, excluding 1

2

Enter the strength of the link (must be positive)

Test

(Enter invalid link strength)

Incorrect Input. Please try again

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 5

1

From which node does the link go to?

Enter a number between 1 and 5, excluding 1

2

Enter the strength of the link (must be positive)

100

(Enter correct input)

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

1

Node 1 has links to 2 [100],

Node 2 has links to

Node 3 has links to 4

Node 4 has links to

Node 5 has links to 2

(Check to see if strength has been assigned)

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 5

5

From which node does the link go to?

Enter a number between 1 and 5, excluding 5

2

Enter the strength of the link (must be positive)

56

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 5

1

From which node does the link go to?

Enter a number between 1 and 5, excluding 1

2

Enter the strength of the link (must be positive)

67

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

1

Node 1 has links to 2 [67],

Node 2 has links to

Node 3 has links to 4

Node 4 has links to

Node 5 has links to 2 [56],

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

3

The Program will now close...

D:\Gareth's Documents\COLEG\E2027\Assignment 3\ex3>

Branch 2

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

1

Node 1 has links to 2 3 4 5 6

Node 2 has links to 1 3 4 5 6

Node 3 has links to 1 2 4 5 6

Node 4 has links to 1 2 3 5 6

Node 5 has links to 1 2 3 4 6

Node 6 has links to 1 2 3 4 5

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

4

From which node does the link go to?

Enter a number between 1 and 6, excluding 4

4

That link does not exist on the Graph!

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

1

From which node does the link go to?

Enter a number between 1 and 6, excluding 1

4

Enter the strength of the link (must be positive)

100

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

5

From which node does the link go to?

Enter a number between 1 and 6, excluding 5

3

Enter the strength of the link (must be positive)

12

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

1

Node 1 has links to 2 3 4 [100], 5 6

Node 2 has links to 1 3 4 5 6

Node 3 has links to 1 2 4 5 6

Node 4 has links to 1 2 3 5 6

Node 5 has links to 1 2 3 [12], 4 6

Node 6 has links to 1 2 3 4 5

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

3

The Program will now close...

D:\Gareth's Documents\COLEG\E2027\Assignment 3\ex3>

Exercise 5: Finding the Shortest Path

Method of testing: Black Box Testing

As this program is based on exercises 3 & 4, the code that is reused from those exercises will not be re-tested here. However, the new code that was produced will be tested. This consists of the code for the two new items added to the main menu: “Assign a name to a node” and “Find the paths from one node to another”.

1. “Assign a name to a node”

Consider that we have created a 5 node, fully connected Graph.

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

```
1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
5: Exit Program
```

```
1
Node 1 has links to 2 3 4 5
Node 2 has links to 1 3 4 5
Node 3 has links to 1 2 4 5
Node 4 has links to 1 2 3 5
Node 5 has links to 1 2 3 4
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

```
1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
5: Exit Program
```

```
3
```

```
Which Node do you want to change the name of?
```

```
Enter a number between 1 and 5
```

```
6 (Enter value out of range: +ve)
```

```
Incorrect Input. Please enter a positive integer between 1 and 5
```

```
-1 (Enter value out of range: -ve)
```

```
Incorrect Input. Please enter a positive integer between 1 and 5
```

```
Test (Enter String input)
```

```
Incorrect Input. Please enter a positive integer between 1 and 5
```

```
2 (Enter correct input)
```

```
What name do you want to give Node 2?
```

```
Test
```

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

3

Which Node do you want to change the name of?
Enter a number between 1 and 5

4

What name do you want to give Node 4?

(Empty input)

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

3

Which Node do you want to change the name of?
Enter a number between 1 and 5

5

What name do you want to give Node 5?
Node Five

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

1

Node 1 has links to 2 3 4 5
Node 2 Test has links to 1 3 4 5
Node 3 has links to 1 2 4 5
Node 4 has links to 1 2 3 5
Node 5 Node Five has links to 1 2 3 4

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

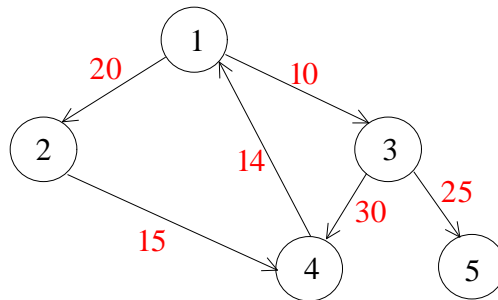
5

The Program will now close...

D:\Gareth's Documents\COLEG\E2027\Assignment 3\ex5>List>

2. "Find the paths from one node to another"

Let us test this command with some *different* Graph configurations. Note that the fully connected graph in the following examples has **5** nodes while the partially connected graph has the following structure:



(a) Fully connected Graph; No arc weights

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.  
-----
```

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

```
1  
Node 1 has links to 2 3 4 5  
Node 2 has links to 1 3 4 5  
Node 3 has links to 1 2 4 5  
Node 4 has links to 1 2 3 5  
Node 5 has links to 1 2 3 4
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.  
-----
```

1. Display the Graph
 2. Add (or change) an Arc Weight
 3. Assign a Name to a Node
 4. Find the Paths from one Node to another
 - 5: Exit Program
- 4

From which node does the path go from?

Enter a number between 1 and 5

1

From which node does the path go to?

Enter a number between 1 and 5, excluding 1

4

```
MATCH: 1-2-3-4  
MATCH: 1-2-3-5-4  
MATCH: 1-2-4  
MATCH: 1-2-5-3-4  
MATCH: 1-2-5-4  
MATCH: 1-3-2-4  
MATCH: 1-3-2-5-4  
MATCH: 1-3-4  
MATCH: 1-3-5-2-4  
MATCH: 1-3-5-4  
MATCH: 1-4  
MATCH: 1-5-2-3-4  
MATCH: 1-5-2-4  
MATCH: 1-5-3-2-4  
MATCH: 1-5-3-4  
MATCH: 1-5-4
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.  
-----
```

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

(b) Fully connected Graph, some arc weights

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

1
Node 1 has links to 2 3 [45], 4 5
Node 2 has links to 1 3 4 5
Node 3 has links to 1 [34], 2 4 5
Node 4 has links to 1 2 3 [24], 5 [34],
Node 5 has links to 1 2 [13], 3 4

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
 2. Add (or change) an Arc Weight
 3. Assign a Name to a Node
 4. Find the Paths from one Node to another
 - 5: Exit Program
- 4

From which node does the path go from?

Enter a number between 1 and 5

1

From which node does the path go to?

Enter a number between 1 and 5, excluding 1

4

MATCH: 1-2-3-4
MATCH: 1-2-3-5-4
MATCH: 1-2-4
MATCH: 1-2-5-3-4
MATCH: 1-2-5-4
MATCH: 1-3-2-4
MATCH: 1-3-2-5-4
MATCH: 1-3-4
MATCH: 1-3-5-2-4
MATCH: 1-3-5-4
MATCH: 1-4
MATCH: 1-5-2-3-4
MATCH: 1-5-2-4
MATCH: 1-5-3-2-4
MATCH: 1-5-3-4
MATCH: 1-5-4

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

(c) Fully connected Graph, all links are weighted

Node 1 has links to 2 [13], 3 [24], 4 [60], 5 [34],
Node 2 has links to 1 [29], 3 [38], 4 [10], 5 [34],
Node 3 has links to 1 [34], 2 [34], 4 [65], 5 [23],
Node 4 has links to 1 [30], 2 [12], 3 [24], 5 [34],
Node 5 has links to 1 [26], 2 [13], 3 [21], 4 [29],

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
 2. Add (or change) an Arc Weight
 3. Assign a Name to a Node
 4. Find the Paths from one Node to another
 - 5: Exit Program
- 4

From which node does the path go from?

Enter a number between 1 and 5

1

From which node does the path go to?

Enter a number between 1 and 5, excluding 1

5

MATCH: 1-2-3-4-5, Length: 150
MATCH: 1-2-3-5, Length: 74
MATCH: 1-2-4-3-5, Length: 70
MATCH: 1-2-4-5, Length: 57
MATCH: 1-2-5, Length: 47
MATCH: 1-3-2-4-5, Length: 102
MATCH: 1-3-2-5, Length: 92
MATCH: 1-3-4-2-5, Length: 135
MATCH: 1-3-4-5, Length: 123
MATCH: 1-3-5, Length: 47
MATCH: 1-4-2-3-5, Length: 133
MATCH: 1-4-2-5, Length: 106
MATCH: 1-4-3-2-5, Length: 152
MATCH: 1-4-3-5, Length: 107
MATCH: 1-4-5, Length: 94
MATCH: 1-5, Length: 34

The shortest path from 1 to 5 was

1-5 with length 34

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
 2. Add (or change) an Arc Weight
 3. Assign a Name to a Node
 4. Find the Paths from one Node to another
 - 5: Exit Program
- 4

From which node does the path go from?

Enter a number between 1 and 5

1

From which node does the path go to?

Enter a number between 1 and 5, excluding 1

4

MATCH: 1-2-3-4, Length: 116
MATCH: 1-2-3-5-4, Length: 103
MATCH: 1-2-4, Length: 23
MATCH: 1-2-5-3-4, Length: 133
MATCH: 1-2-5-4, Length: 76
MATCH: 1-3-2-4, Length: 68
MATCH: 1-3-2-5-4, Length: 121
MATCH: 1-3-4, Length: 89
MATCH: 1-3-5-2-4, Length: 70
MATCH: 1-3-5-4, Length: 76
MATCH: 1-4, Length: 60
MATCH: 1-5-2-3-4, Length: 150
MATCH: 1-5-2-4, Length: 57
MATCH: 1-5-3-2-4, Length: 99
MATCH: 1-5-3-4, Length: 120
MATCH: 1-5-4, Length: 63

The shortest path from 1 to 4 was

1-2-4 with length 23

(d) Partially connected Graph, No arc weights

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

1
Node 1 has links to 2 3
Node 2 has links to 4
Node 3 has links to 4 5
Node 4 has links to 1
Node 5 has links to

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

4

From which node does the path go from?

Enter a number between 1 and 5

1

From which node does the path go to?

Enter a number between 1 and 5, excluding 1

4

MATCH: 1-2-4

MATCH: 1-3-4

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

(e) Partially connected Graph, some arc weights

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

1
Node 1 has links to 2 3 [25],
Node 2 has links to 4
Node 3 has links to 4 5
Node 4 has links to 1 [20],
Node 5 has links to

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

4

```
From which node does the path go from?
Enter a number between 1 and 5
1

From which node does the path go to?
Enter a number between 1 and 5, excluding 1
3

MATCH: 1-3, Length: 25

The shortest path from 1 to 3 was
1-3 with length 25
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
-----
```

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

4

```
From which node does the path go from?
Enter a number between 1 and 5
1
```

```
From which node does the path go to?
Enter a number between 1 and 5, excluding 1
5
```

```
MATCH: 1-3-5
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
-----
```

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

(f) Partially connected Graph, all links are weighted.

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
-----
```

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

```
1
Node 1 has links to 2 [10], 3 [25],
Node 2 has links to 4 [35],
Node 3 has links to 4 [45], 5 [10],
Node 4 has links to 1 [20],
Node 5 has links to
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
-----
```

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

4

```
From which node does the path go from?
Enter a number between 1 and 5
1
```

```
From which node does the path go to?
Enter a number between 1 and 5, excluding 1
3
```

MATCH: 1-3, Length: 25

The shortest path from 1 to 3 was
1-3 with length 25

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

4

From which node does the path go from?
Enter a number between 1 and 5

1

From which node does the path go to?
Enter a number between 1 and 5, excluding 1

4

MATCH: 1-2-4, Length: 45
MATCH: 1-3-4, Length: 70

The shortest path from 1 to 4 was
1-2-4 with length 45

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

5

The Program will now close...

(g) Input tests.

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

4

From which node does the path go from?
Enter a number between 1 and 4

-1

Incorrect Input. Please enter a positive integer between 1 and 4

5

Incorrect Input. Please enter a positive integer between 1 and 4

Test

Incorrect Input. Please enter a positive integer between 1 and 4

3

From which node does the path go to?
Enter a number between 1 and 4, excluding 3

5

Incorrect Input. Please enter a positive integer between 1 and 4

-1

Incorrect Input. Please enter a positive integer between 1 and 4

Test

Incorrect Input. Please enter a positive integer between 1 and 4

2

MATCH: 3-1-2
MATCH: 3-1-4-2
MATCH: 3-2
MATCH: 3-4-1-2
MATCH: 3-4-2

Conclusions

In this assignment, we completed the following objectives:

- We implemented a simple sequential search and a bubble sort on a record retrieval system involving names and ID numbers. Sorting was performed alphabetically.
- We tested the speed of a bubble sort when compared to a quick sort obtained from literature. It was found that the more complicated algorithm was far more efficient, taking much less time to sort our arrays, the efficiency increasing as the size of the arrays increased.
- We implemented a Graph abstract data type using two different techniques. In our first implementation,
 - We used a linked list to represent the nodes and a linked list to represent the links for a particular node
 - We provided four methods that allowed any other class to create a graph with a specified number of nodes with links provided automatically or manually from the keyboard, a method to display all the nodes and their associated links, and a method to allow the user to specify a weight for a link from one node to another.
- In our second implementation,
 - We used an array to represent the nodes and a linked list to represent the links for a particular node
 - We provided the same four methods as above.
- We used the above graph abstract data types to find the shortest path between two cities A and B. We searched through the graph using a depth first search, finding all possible paths between A and B, and calculating the shortest distance (based on arc weights) as we went along. The method then returned the shortest path (if arc weights had all been assigned) together with a list of all “winning” paths.

Bibliography

Roger, G. Mariani, J. (1998) Java: First Contact, 1st ed. Course Technology.

Shaffer, C. (1998) A Practical Introduction To Data Structures And Algorithm Analysis, 1st ed. New Jersey: Prentice-Hall, Inc.

References

Roberts, H. (1996) Pascal quickSort program, 1st ed. Dyffryn Conwy

Appendix: Source Code and MS-DOS Output

Exercise 1: Searching and Sorting Family Records.

Program Code

```
/*
 *
 * Gareth Evans
 *
 * Started: 15th November 1999
 * Finished: 15th November 1999 (Revision 1.1)
 *
 * Assignment 3, Exercise 1
 *
 * Student Class
 * Variables: forename, surname, ID
 *
 */

public class Student
{
    private String forename;
    private String surname;
    private int    ID;

    // METHODS FOR RETURNING DATA

    // Returns the forename attribute of the Student
    public String getForename()
    {
        return forename;
    }

    // Returns the surname attribute of the Student
    public String getSurname()
    {
        return surname;
    }

    // Returns the ID number of the Student
    public int getID()
    {
        return ID;
    }

    // METHODS FOR SETTING DATA

    // Set the forename of the Student
    public void setForename(String input)
    {
        forename = input;
    }

    // Set the surname of the Student
    public void setSurname(String input)
    {
        surname = input;
    }

    // Set the ID Number of the Student
    public void setID(int input)
    {
        ID = input;
    }
} // end of class Student
```

```

/*
 *
 * Gareth Evans
 *
 * Started: 15th November 1999
 * Finished: 17th November 1999 (Revision 1.2)
 *
 * Assignment 3, Exercise 1
 *
 * Node Class
 * Linked List Implementation
 *
 */

public class Node
{
    // Define Variables for each Node

    Student data;
    Node nextElement;

    // Constructor for creating a new node

    Node()
    {
        data = null;
        nextElement = null;
    }

    // Insert method

    public void insert(Node whatList, Student obj)
    {
        if (whatList.nextElement == null) // Checks next element in list
        {
            whatList.data = obj; // Add element to list
            whatList.nextElement = new Node(); // Create a fresh new node
        }
        else
        {
            insert(whatList.nextElement, obj); // Calls the method again, within itself
        }
    }

    // Swap Method

    public void swap(Node whatList, int start, int check)
    {
        Node parameter = new Node();

        if (check == start)
        {
            parameter.data = whatList.data;
            whatList.data = whatList.nextElement.data;
            whatList.nextElement.data = parameter.data;
        }
        else
        {
            check++;
            swap(whatList.nextElement, start, check);
        }
    }

    // NEXT METHOD

    public Node nextNode(Node whatList) // pass in a node, want to return the next node
    {
        return whatList.nextElement;
    }
}

```

```

/*
 *
 * Gareth Evans
 *
 * Started: 15th November 1999
 * Finished: 22nd November 1999 (Revision 1.3)
 *
 * Assignment 3, Exercise 1
 *
 * System Class
 * Main Class for Exercise 1
 */

import java.bangor.*;

public class Exercisel
{
    // Define Global Variables

    static Node database = new Node();
    static Node storageNode = new Node();
    static int numberOfRecords = 0;

    // CREATE RECORD METHOD

    public static void CreateRecord() throws Exception
    {
        // declare local variables
        Student studentData = new Student();
        String inputString = new String();

        // prepare the student record
        studentData.setID(numberOfRecords);
        numberOfRecords++;

        // ask the user to enter the Student's forename
        System.out.println();
        System.out.print("Please enter the Student's forename...");
        inputString = BasicIo.readString();
        studentData.setForename(inputString);

        // ask the user to enter the Student's surname
        System.out.print("Please enter the Student's surname....");
        inputString = BasicIo.readString();
        studentData.setSurname(inputString);

        // Call the insert method in Node.java, passing in the root node of the linked
        // list and the object to be inserted in the list.

        database.insert(database, studentData);
        DisplayRecords();
    }

    // GET RECORD METHOD

    public static Student getRecord(int i)
    {
        Node storageNode = database;
        for (int j = 0; j < i; j++)
        {
            storageNode = storageNode.nextNode(storageNode);
        }
        return storageNode.data;
    }

    // DISPLAY RECORDS METHOD

    public static void DisplayRecords()
    {
        System.out.println();

        for( int i=0; i < numberOfRecords; i++ )
        {
            System.out.print(getRecord(i).getID()+1);
            System.out.print(" : ");
            System.out.print(getRecord(i).getForename());
            System.out.print(" ");
            System.out.println(getRecord(i).getSurname());
        }
        System.out.println();
    }
}

```

```

// SORT RECORDS METHOD

public static void SortRecords()
{
    // BUBBLE SORT

    for (int i=0; i < numberOfRecords; i++)
    {
        for (int j=0; j < (numberOfRecords-1); j++)
        {
            if ((getRecord(j).getSurname()).compareTo(getRecord(j+1).getSurname()) > 0)

                // Explanation: we compare two strings. The compare command returns an
                // integer >0 if the first string is "greater than" the second string.

                {
                    // SWAP - swaps j and j+1
                    database.swap(database, j, 0);
                }
        }
    }
    DisplayRecords();
}

// SEARCH RECORD METHOD

public static void SearchRecord()
{
    // local variables
    int searchInt = -1;

    // ASK USER FOR ID NUMBER
    System.out.println();
    System.out.print("Please enter an ID Number to search for more information on...");

    try
    {
        searchInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    { }

    if ((searchInt < 1) | (searchInt > numberOfRecords))
    {
        System.out.println("Incorrect Input. Please enter a valid positive ID number");
    }
    else
    {
        System.out.println();
        System.out.println("Searching for element " + searchInt + "....");

        // SEARCH CODE

        for (int i = 0; i < numberOfRecords; i++)
        {
            if((getRecord(i).getID()+1) == searchInt)
            {
                System.out.print(getRecord(i).getID()+1);
                System.out.print(": ");
                System.out.print(getRecord(i).getForename());
                System.out.print(" ");
                System.out.println(getRecord(i).getSurname());
            }
        }
    }
}

```

```

public static void main(String args[]) throws Exception
{
    int Choice = -1;

    // MENU

    do // repeat this until option 5 is chosen
    {
        Choice = -1;

        System.out.println();
        System.out.println("MAIN MENU - PLEASE ENTER YOUR CHOICE.");
        System.out.println("-----");
        System.out.println();
        System.out.println("1. Create New Record");
        System.out.println("2. Display All Records");
        System.out.println("3: Sort Records");
        System.out.println("4: Search for Record");
        System.out.println("5: Exit Program");
        System.out.println();

        try
        {
            Choice = BasicIo.readInteger();
        }

        catch (Exception exp) { }

        switch(Choice)
        {
            case 1: CreateRecord();
                    break;

            case 2: DisplayRecords();
                    break;

            case 3: SortRecords();
                    break;

            case 4: SearchRecord();
                    break;

            case 5: System.out.println("The Program will now close...");
                    break;

            default: System.out.println("Sorry, Input is incorrect. Please Try Again. ");
        }
    }
    while (Choice != 5);
}

```

*

MS-DOS Output

```
M:\e2027\Assignment 3\ex1>java Exercisel
Symantec Java! JustInTime Compiler Version 210.050 for JDK 1.1
Copyright (C) 1996-97 Symantec Corporation
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

1

```
Please enter the Student's forename....Ian
Please enter the Student's surname.....Roberts
```

1: Ian Roberts

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

1

```
Please enter the Student's forename....Ioan
Please enter the Student's surname.....Williams
```

1: Ian Roberts
2: Ioan Williams

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

(Carry on and enter 6 more records....)

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

1

```
Please enter the Student's forename....Bill
Please enter the Student's surname.....Clinton
```

1: Ian Roberts
2: Ioan Williams
3: Gareth Evans
4: Arwel Stevens
5: Sion Evans
6: Arwel Lloyd
7: Wil Ddrwg
8: Bill Clinton

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

2

- 1: Ian Roberts
- 2: Ioan Williams
- 3: Gareth Evans
- 4: Arwel Stevens
- 5: Sion Evans
- 6: Arwel Lloyd
- 7: Wil Ddrwg
- 8: Bill Clinton

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

4

Please enter an ID Number to search for more information on...5

Searching for element 5....

- 5: Sion Evans

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

3

- 8: Bill Clinton
- 7: Wil Ddrwg
- 3: Gareth Evans
- 5: Sion Evans
- 6: Arwel Lloyd
- 1: Ian Roberts
- 4: Arwel Stevens
- 2: Ioan Williams

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

4

Please enter an ID Number to search for more information on...1

Searching for element 1....

- 1: Ian Roberts

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

1

Please enter the Student's forename....Monica

Please enter the Student's surname.....Lewinsky

- 8: Bill Clinton
- 7: Wil Ddrwg
- 3: Gareth Evans
- 5: Sion Evans
- 6: Arwel Lloyd
- 1: Ian Roberts
- 4: Arwel Stevens
- 2: Ioan Williams
- 9: Monica Lewinsky

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

3

- 8: Bill Clinton
- 7: Wil Ddrwg
- 3: Gareth Evans
- 5: Sion Evans
- 9: Monica Lewinsky
- 6: Arwel Lloyd
- 1: Ian Roberts
- 4: Arwel Stevens
- 2: Ioan Williams

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Create New Record
2. Display All Records
- 3: Sort Records
- 4: Search for Record
- 5: Exit Program

5

The Program will now close...

M:\e2027\Assignment 3\ex1>

Exercise 2: Sorting Methods

Program Code

```
/*
 *
 *
 * Written by      Ian C Roberts
 * Date:          22-11-99
 * Description:    This program compare the use of our
 *                bubble sorting method and the shaffer
 *                book's quick sort.
 *
 */
import java.bangor.*;

public class Exercise2
{
    //Declare Arrays
    static int[] array1;
    static int[] array2;

    //declare global Variables
    static int noOfSwaps =0;
    static int noOfSwaps2 = 0;

    //Method to create a new array with user defined parameters
    public static void createArray(int arraySize)
    {
        array1 = new int[arraySize];
        array2 = new int[arraySize];
    }

    //Method for displaying the contents of the arrays
    public static void displayArrays(int arraySize)
    {
        int j=0;
        int x=0;
        System.out.println("The contents of the first Array");
        System.out.println("-----");
        System.out.println();

        for(int i=0; i < arraySize; i++)
        {
            System.out.print("Location "+j);           //Printing out the number of elements
            System.out.println(": "+array1[i]);        //in array1 in reverse order to be
            j++;                                       //sorted by the
        }
        System.out.println();

        System.out.println("The contents of the second Array");
        System.out.println("-----");
        System.out.println();

        for(int k=0; k < arraySize; k++)
        {
            System.out.print("Location "+x);           // as above
            System.out.println(": "+array2[k]);
            x++;
        }
    }
}
```

```

//Method to sort first array with the Bubble sort
public static void BubbleSort(int arraySize)
{
    for (int i=0; i < arraySize; i++)        // first loop
    {
        for (int j=0; j < arraySize-1; j++) // second loop
        {
            if (array1[j] > array1[j+1])
                // if successive elements not in order swap them
                {
                    int temp = array1[j];
                    array1[j] = array1[j+1];
                    array1[j+1] = temp;
                    noOfSwaps++;
                }
        }
    }
}

```

//method obtained from Pascal A level material

```

public static void swapit(int A, int B)
{
    int temp;

    temp = array2[A];
    array2[A] = array2[B];
    array2[B] = temp;

    noOfSwaps2++;
}

public static void quicksortit(int first, int last) throws Exception
{
    int one, two, pivot, adj1, adj2;

    one = first;
    two = last;
    pivot = 0;

    while(one!=two)
    {
        while( (array2[one] < array2[two]) & (one < two) )
        {
            two--;
        }

        if (one < two)
        {
            if (array2[one] != array2[two] )
            {
            }
            else
            {
                one++;
            }
        }

        while( (array2[one] < array2[two]) & (one < two) )
        {
            one++;
        }
    }
}

```

```

    if(one < two)
    {
        if(array2[one] != array2[two])
        {
            swapit(one,two);
        }
        else
        {
            two--;
        }
    }

    pivot = one;

    if(((pivot-1)-first) == 1)
    {
        adj1 = pivot-1;
        quicksortit(first, adj1);
    }

    if((last-(pivot+1))>=1)
    {
        adj2 = pivot+1;
        quicksortit(adj2, last);
    }
}
}

```

//MAIN METHOD

```

public static void main(String args[]) throws Exception
{
    //ask the user to enter the size of the arrays

    System.out.print("Please enter the sizes of the arrays: ");
    int arraySize = BasicIo.readInteger();
    System.out.println();

    //create arrays
    createArray(arraySize);

    // initialise variables
    int k = 0;
    int j=arraySize;

    // enter data into arrays (reverse order)
    for (int i=0; i < arraySize; i++)
    {
        array1[i] = j;
        System.out.println(array1[i]);
        array2[i] = j;
        j--;
    }

    // sort using both algorithms and then print out the arrays
    BubbleSort(arraySize);
    quicksortit(0, arraySize-1);
    displayArrays(arraySize);

    //report on the number of swaps done
    System.out.print("The number of swaps with the BUBBLESORT: "+noOfSwaps);
    System.out.println();
    System.out.print("The number of swaps with the QUICKSORT: "+noOfSwaps2);
}
}

```

MS-DOS Output

C:\WINDOWS\Desktop\ex2 Fri 10/12/1999 0:44:00.26 >java Exercise2

Please enter the sizes of the arrays: 8

8
7
6
5
4
3
2
1

The contents of the first Array

Location 0: 1
Location 1: 2
Location 2: 3
Location 3: 4
Location 4: 5
Location 5: 6
Location 6: 7
Location 7: 8

The contents of the second Array

Location 0: 1
Location 1: 2
Location 2: 3
Location 3: 4
Location 4: 5
Location 5: 6
Location 6: 7
Location 7: 8

The number of swaps with the BUBBLESORT: 28

The number of swaps with the QUICKSORT: 4

C:\WINDOWS\Desktop\ex2 Fri 10/12/1999 0:44:19.87 >

Exercise 3: Graph Implementation (Linked Lists)

Program Code

```
/*
 *
 * Gareth Evans
 *
 * Started: 29th November 1999
 * Finished: 29th November 1999 (Revision 1.1)
 *
 * Assignment 3, Exercise 3
 *
 * Node Class
 * Linked List Implementation for Graph Nodes
 *
 */

public class Node
{
    // Define Variables for each Node

    int data;
    LinkNode links;
    Node nextElement;

    // Constructor for creating a new node

    Node()
    {
        data = -1;
        nextElement = null;
        links = new LinkNode();
    }

    // INSERT METHOD

    public void insert(Node whatList, int obj, LinkNode linkobj)
    {
        if (whatList.nextElement == null) // Checks next element in list
        {
            whatList.data = obj; // Add element to list
            whatList.nextElement = new Node(); // Create a fresh new node
            whatList.links = linkobj; // Add the node Links
        }
        else
        {
            insert(whatList.nextElement, obj, linkobj); // Calls the method again
        }
    }

    // INSERT LINKS

    public void insertList(Node whatList, int start, int check, LinkNode obj)
    {
        if (check == start)
        {
            whatList.links = obj;
        }
        else
        {
            check++;
            insertList(whatList.nextElement, start, check, obj);
        }
    }

    // NEXT METHOD

    public Node nextNode(Node whatList) // pass in a node, want to return the next node
    {
        return whatList.nextElement;
    }
}
```

```

/*
 *
 * Gareth Evans
 *
 * Started: 29th November 1999
 * Finished: 1st December 1999 (Revision 1.2)
 *
 * Assignment 3, Exercise 3
 *
 * LinkNode Class
 * Linked List Implementation for Node Links
 *
 */

public class LinkNode
{
    // Define Variables for each Node

    int link;
    int weight;
    LinkNode nextElement;

    // Constructor for creating a new node

    LinkNode()
    {
        link = -1;
        weight = -1;
        nextElement = null;
    }

    // INSERT METHOD

    public void insert(LinkNode whatList, int obj)
    {
        if (whatList.nextElement == null)           // Checks next element in list
        {
            whatList.link = obj;                    // Add element to list
            whatList.nextElement = new LinkNode(); // Create a fresh new node
        }
        else
        {
            insert(whatList.nextElement, obj);     // Calls the method again
        }
    }

    // NEXT METHOD

    public LinkNode nextNode(LinkNode whatList)
    // pass in a node, want to return the next node
    {
        return whatList.nextElement;
    }
}

```

```

/*
 *
 * Gareth Evans
 *
 * Started: 29th November 1999
 * Finished: 2nd December 1999 (Revision 2.1)
 *
 * Assignment 3, Exercise 3
 *
 * Graph Class
 *
 */

import java.bangor.*;

public class Graph
{
    // Declare Graph variables

    static Node TestGraphNode = new Node();
    static int numberOfNodes = 0;

    // GET NODE METHOD

    public static LinkNode getNode(int i, Node list)

    // This method returns the links for a particular node in the graph.
    {
        Node storageNode = list;

        // Go to the node requested in the graph by using the following loop:
        for (int j = 0; j < i; j++)
        {
            storageNode = storageNode.nextNode(storageNode);
        }
        return storageNode.links;
    }

    // GET LINK METHOD

    public static int getLink(int i, LinkNode list)

    // This method returns the destination of the ith link in a list of links
    {
        int returnInt = list.link; // need this assignment in the case i=0
        LinkNode storageLink = list;

        for (int j = 0; j < i; j++)
        {
            // If the link we are looking at is empty, then there is no ith link
            // (There are no more links in the list), and we return -1, indicating
            // that the ith link does not exist.

            if (storageLink.link == -1)
            {
                returnInt = -1;
            }
            else // i.e. there are more links in the list
            {
                storageLink = storageLink.nextNode(storageLink);
                returnInt = storageLink.link;
            }
        }
        return returnInt;
    }

    // GET LINK STRENGTH METHOD

    public static int getStrength(int i, LinkNode list)

    // This method returns the strength of the ith link in a particular list of links
    {
        int returnInt = list.weight; // need this assignment in the case i=0
        LinkNode storageLink = list;

        for (int j = 0; j < i; j++)
        {
            if (storageLink.link == -1) // i.e. no more links in the list
            {
                returnInt = -1;
            }
            else
            {
                storageLink = storageLink.nextNode(storageLink);
                returnInt = storageLink.weight;
            }
        }
        return returnInt;
    }
}

```

```
// METHOD TO ASK THE USER TO ENTER THE NUMBER OF NODES
```

```
public static int Ask_Nodes() throws Exception
{
    int inputInt;

    // Ask the user to enter the Number of nodes in the list
    System.out.println();
    System.out.print("Please enter the number of nodes in the Test Graph...");

    try
    {
        inputInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
        inputInt = -1;
    }

    if (inputInt < 1)
    {
        System.out.println("Incorrect Input. Please enter a positive integer");
        inputInt = Ask_Nodes(); // call the method again
        return inputInt;
    }
    else // i.e. the user has entered correct input
    {
        numberOfNodes = inputInt;
        System.out.println();
        return inputInt;
    }
}
```

```
// ENTER LINKS METHOD
```

```
public static int EnterLinks(int limit) throws Exception

// This method allows the user to enter a departure node for a link
{
    int enterInt = 0;

    try
    {
        enterInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    { }

    if ((enterInt < 1) || (enterInt > limit))
    {
        System.out.println("Incorrect Input. Please enter a positive integer between 1 and " + limit);
        enterInt = EnterLinks(limit); // call the method again
    }
    return enterInt;
}
```

```
// INSERT LINK METHOD
```

```
public static void InsertLink() throws Exception
{
    // Ask the user for the departure node of the link
    System.out.println("From which node does the link go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int departure = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user for the destination node of the link
    System.out.println("From which node does the link go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + departure);
    int destination = EnterLinks(numberOfNodes);
    System.out.println();

    // Let check denote whether the link from departure to destination already exists.
    LinkNode checkLinks = getNode(departure-1, TestGraphNode);
    int check = 0;

    for (int z=0; z < numberOfNodes; z++)
    {
        int storageInt = getLink(z, checkLinks);
        if (storageInt == destination-1)
        {
            check = 1;
        }
    }
}
```

```

if ((destination != departure) & (check == 0))
{ // i.e. we can insert the link

    LinkNode tempLinks = getNode(departure-1, TestGraphNode); // get the list to insert link into
    tempLinks.insert(tempLinks, destination-1); // insert the link

}
else
{
    if (destination == departure)
    {
        System.out.println("Cannot have a link from a node to the same node.");
    }
    else
    {
        System.out.println("The specified link already exists!");
    }
    System.out.println();
}
}

// CHOOSE LINKS METHOD

public static void ChooseLinks() throws Exception
{
    boolean carryOn = true;
    String inputString = new String();

    while (carryOn == true)
    {
        Display(); // Shows the Graph at this particular time
        System.out.println();
        System.out.println("Would you like to insert a link?");
        System.out.println("Answer 'y' or 'n'");
        inputString = BasicIo.readString();
        System.out.println();

        if (inputString.equals("y"))
        {
            InsertLink(); // Calls the above procedure
        }
        else
        {
            if (inputString.equals("n"))
            {
                carryOn = false;
            }
            else
            {
                System.out.println("Incorrect Input. Try Again");
                System.out.println();
                carryOn = true;
            }
        }
    }
}

// ***
// TEST GRAPH METHOD
// ***

public static void TestGraph() throws Exception
{
    // Ask the user how many nodes he or she wants
    int input = Ask_Nodes();

    // We now need to create "input" amount of new nodes
    for (int i = 0; i < input; i++)
    {
        LinkNode insertLinks = new LinkNode();

        // For each created node, we create links to every other node
        for(int k = 0; k < input; k++)
        {
            if (i != k) // i.e. do not create a link from a node to the same node
            {
                // Insert a link into the adjacency list
                insertLinks.insert(insertLinks, k);
            }
        }
        // Insert a node into the graph
        TestGraphNode.insert(TestGraphNode, i, insertLinks);
    }
}

```

```

// ***
// READ GRAPH METHOD
// ***

public static void ReadGraph() throws Exception
{
    // Ask the user how many nodes he or she wants
    int input = Ask_Nodes();

    // We now need to create "input" amount of new nodes
    for (int i = 0; i < input; i++)
    {
        // For each node, we create an empty list of lists

        LinkNode insertLinks = new LinkNode();
        TestGraphNode.insert(TestGraphNode, i, insertLinks);
    }
    // Now let the user decide on the links in the Graph
    ChooseLinks();
}

// ***
// DISPLAY METHOD
// ***

public static void Display()
{
    // For each Node we shall display its links sequentially
    for (int k=0; k < numberOfNodes; k++)
    {
        System.out.print("Node " + (k+1) + " has links to ");

        // Now get the links for a particular node:
        LinkNode storage = getNode(k, TestGraphNode);

        // We check to see which links exist and print them out
        for (int x=0; x < numberOfNodes; x++)
        {
            // For a particular link, we now get the destination and the strength of the link
            int storageInt = getLink(x, storage);
            int storageInt2 = getStrength(x, storage);

            // If a link exists to the particular node we are looking at....
            if (storageInt != -1)
            {
                // ...Print the destination of the link, then.....
                System.out.print((storageInt+1) + " ");

                // ....If a strength has been assigned to the link,.....
                if (storageInt2 != -1)
                {
                    // ...Print the strength of the link.
                    System.out.print("[ " + storageInt2 + " ], ");
                }
            }
        }
        System.out.println();
    }
}

// CHECK EXISTENCE OF LINK METHOD

public static int CheckExistence(int a, LinkNode inputLink)
{
    int returnInt = 0;
    LinkNode storage = inputLink;

    // Cycle through the list of links, looking to see if a particular link exists
    for (int y = 0; y < numberOfNodes; y++)
    {
        if(storage.link == a) // i.e. we have found a match!
        {
            returnInt = 1;
        }
        else
        {
            if(storage.link != -1) // i.e. if more links exists
            {
                storage = storage.nextElement; // look at the next link in the list.
            }
        }
    }
    return returnInt;
}

```

```

// ENTER LINK WEIGHT METHOD

public static void EnterWeight(int departure, int destination, LinkNode storage)
{
    // Ask the user for the strength of the link in question
    System.out.println("Enter the strength of the link (must be positive)");
    int strength = -1;

    try
    {
        strength = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
        strength = -1;
    }

    if (strength < 0)
    {
        System.out.println("Incorrect Input. Please try again");
    }
    else // i.e. the user has entered some correct input
    {
        // Get the list of links for the Node in question
        LinkNode storage2 = getNode(departure-1, TestGraphNode);

        // We must now cycle through the list, looking for the correct link
        for (int z = 1; z < numberOfNodes; z++)
        {
            if(storage.link == destination-1) // i.e. we have reached the correct link
            {
                // Place the link weight in the list of links
                storage.weight = strength;
            }
            else // i.e. have not yet reached the correct link
            {
                if(storage.link != -1)
                {
                    storage = storage.nextElement;
                }
            }
        }
    }
}

// ***
// ADD WEIGHT TO ARC METHOD
// ***

public static void AddWeight() throws Exception
{
    // Ask the user to enter a start point for a link
    System.out.println("From which node does the link go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int departure = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user to enter the end point for a link
    System.out.println("From which node does the link go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + departure);
    int destination = EnterLinks(numberOfNodes);
    System.out.println();

    // Get the list of links for the departure node
    LinkNode storage = getNode(departure-1, TestGraphNode);

    // See if a link to the destination node actually exists. We get a "1" back if it does
    int match = CheckExistence(destination-1, storage);

    if ((destination != departure) & (match == 1))
    {
        // Ask the user to enter a weight for the link in question
        EnterWeight(departure, destination, storage);
    }
    else
    {
        System.out.println("That link does not exist on the Graph!");
    }
}
}

```

```

/*
 *
 * Ioan Williams
 *
 * Started: 30th November 1999
 * Finished: 1st December 1999
 * Assignment 3, Exercise 3
 *
 * Main Class for Exercise 3
 *
 */

import java.bangor.*;

public class Exercise3
{
    // Define Global Variables
    static Graph ourGraph = new Graph();

    // GET INPUT METHOD

    public static void GetInput() throws Exception
    {
        String inputString = BasicIo.readString();

        if (inputString.equals("y"))
        {
            ourGraph.ReadGraph();
        }
        else
        {
            if (inputString.equals("n"))
            {
                ourGraph.TestGraph();
            }
            else
            {
                System.out.println();
                System.out.println("Incorrect Input. Try Again");
                System.out.println("Would you like to specify the links in the Graph?");
                System.out.print("Answer 'y' or 'n'...");
                GetInput();
            }
        }
    }

    public static void main(String args[]) throws Exception
    {
        System.out.println();
        System.out.println("Would you like to specify the links in the Graph?");
        System.out.print("Answer 'y' or 'n'...");
        GetInput();
        System.out.println();

        int Choice = -1;

        // MENU

        do // repeat this until option 3 is chosen
        {
            Choice = -1;

            System.out.println();
            System.out.println("MAIN MENU - PLEASE ENTER YOUR CHOICE.");
            System.out.println("-----");
            System.out.println();
            System.out.println("1. Display the Graph");
            System.out.println("2. Add (or change) An Arc Weight");
            System.out.println("3: Exit Program");
            System.out.println();

            try
            {
                Choice = BasicIo.readInteger();
            }

            catch (Exception exp) { }
        }
    }
}

```

```
switch(Choice)
{
    case 1:  ourGraph.Display();
            break;

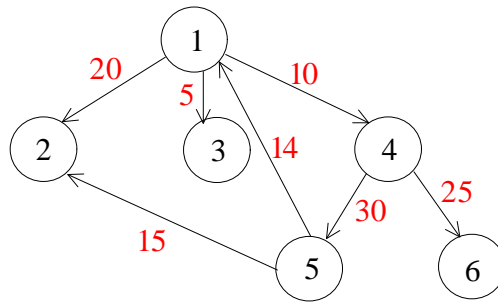
    case 2:  ourGraph.AddWeight();
            break;

    case 3:  System.out.println("The Program will now close...");
            break;

    default: System.out.println("Sorry, Input is incorrect. Please Try Again. ");
}
}
while (Choice != 3);
}
}
```

MS-DOS Output

As example output, let us first create a 5 Node tree with no arc weights. Then, let us create the Graph shown below.



```
D:\Gareth's Documents\COLEG\E2027\Assignment 3\ex3>java Exercise3
```

```
Would you like to specify the links in the Graph?
```

```
Answer 'y' or 'n'...n
```

```
Please enter the number of nodes in the Test Graph...5
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

```
1
```

```
Node 1 has links to 2 3 4 5
```

```
Node 2 has links to 1 3 4 5
```

```
Node 3 has links to 1 2 4 5
```

```
Node 4 has links to 1 2 3 5
```

```
Node 5 has links to 1 2 3 4
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

```
-----
```

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

```
3
```

```
The Program will now close...
```

```
D:\Gareth's Documents\COLEG\E2027\Assignment 3\ex3>
```

D:\Gareth's Documents\COLEG\E2027\Assignment
3\ex3>java Exercise3

Would you like to specify the links in the Graph?
Answer 'y' or 'n'...y

Please enter the number of nodes in the Test
Graph...6

Node 1 has links to
Node 2 has links to
Node 3 has links to
Node 4 has links to
Node 5 has links to
Node 6 has links to

Would you like to insert a link?
Answer 'y' or 'n'
Y

From which node does the link go from?
Enter a number between 1 and 6
1

From which node does the link go to?
Enter a number between 1 and 6, excluding 1
2

Node 1 has links to 2
Node 2 has links to
Node 3 has links to
Node 4 has links to
Node 5 has links to
Node 6 has links to

Would you like to insert a link?
Answer 'y' or 'n'
Y

From which node does the link go from?
Enter a number between 1 and 6
1

From which node does the link go to?
Enter a number between 1 and 6, excluding 1
3

Node 1 has links to 2 3
Node 2 has links to
Node 3 has links to
Node 4 has links to
Node 5 has links to
Node 6 has links to

Would you like to insert a link?
Answer 'y' or 'n'
Y

From which node does the link go from?
Enter a number between 1 and 6
1

From which node does the link go to?
Enter a number between 1 and 6, excluding 1
4

Node 1 has links to 2 3 4
Node 2 has links to
Node 3 has links to
Node 4 has links to
Node 5 has links to
Node 6 has links to

Would you like to insert a link?
Answer 'y' or 'n'
Y

From which node does the link go from?
Enter a number between 1 and 6
4

From which node does the link go to?
Enter a number between 1 and 6, excluding 4
5

Node 1 has links to 2 3 4
Node 2 has links to
Node 3 has links to
Node 4 has links to 5
Node 5 has links to
Node 6 has links to

Would you like to insert a link?
Answer 'y' or 'n'
Y

From which node does the link go from?
Enter a number between 1 and 6
4

From which node does the link go to?
Enter a number between 1 and 6, excluding 4
6

Node 1 has links to 2 3 4
Node 2 has links to
Node 3 has links to
Node 4 has links to 5 6
Node 5 has links to
Node 6 has links to

Would you like to insert a link?
Answer 'y' or 'n'
Y

From which node does the link go from?
Enter a number between 1 and 6
5

From which node does the link go to?
Enter a number between 1 and 6, excluding 5
2

Node 1 has links to 2 3 4
Node 2 has links to
Node 3 has links to
Node 4 has links to 5 6
Node 5 has links to 2
Node 6 has links to

Would you like to insert a link?
Answer 'y' or 'n'
Y

From which node does the link go from?
Enter a number between 1 and 6
5

From which node does the link go to?
Enter a number between 1 and 6, excluding 5
1

Node 1 has links to 2 3 4
Node 2 has links to
Node 3 has links to
Node 4 has links to 5 6
Node 5 has links to 2 1
Node 6 has links to

Would you like to insert a link?
Answer 'y' or 'n'
n

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
3: Exit Program

2
From which node does the link go from?
Enter a number between 1 and 6
1

From which node does the link go to?
Enter a number between 1 and 6, excluding 1
2

Enter the strength of the link (must be positive)
20

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

1

From which node does the link go to?

Enter a number between 1 and 6, excluding 1

3

Enter the strength of the link (must be positive)

5

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

1

From which node does the link go to?

Enter a number between 1 and 6, excluding 1

4

Enter the strength of the link (must be positive)

10

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

4

From which node does the link go to?

Enter a number between 1 and 6, excluding 4

5

Enter the strength of the link (must be positive)

30

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

4

From which node does the link go to?

Enter a number between 1 and 6, excluding 4

6

Enter the strength of the link (must be positive)

25

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

5

From which node does the link go to?

Enter a number between 1 and 6, excluding 5

2

Enter the strength of the link (must be positive)

15

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 6

5

From which node does the link go to?

Enter a number between 1 and 6, excluding 5

1

Enter the strength of the link (must be positive)

14

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

1

Node 1 has links to 2 [20], 3 [5], 4 [10],

Node 2 has links to

Node 3 has links to

Node 4 has links to 5 [30], 6 [25],

Node 5 has links to 2 [15], 1 [14],

Node 6 has links to

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) An Arc Weight
- 3: Exit Program

3

The Program will now close...

D:\Gareth's Documents\COLEG\E2027\Assignment 3\ex3>

Exercise 4: Graph Implementation (Array, Linked Lists)

Program Code

The program code for this exercise was modified from the code for exercise 3. We use exactly the same **LinkNode** class as in exercise 3, do **NOT** use the **Node** class, use exactly the same “*main*” program and slightly modify the **Graph** class. I will give the Graph class below, and where code has been modified, I will highlight it in **blue**.

```
/*
 *
 * Gareth Evans
 *
 * Started: 29th November 1999
 * Finished: 2nd December 1999 (Revision 2.1)
 *
 * Assignment 3, Exercise 4
 *
 * Graph Class
 *
 */

import java.bangor.*;

public class Graph
{
    // Declare Graph variables

    static LinkNode[] TestGraphArray;
    static int numberOfNodes = 0;

    // GET LINK METHOD

    public static int getLink(int i, LinkNode list)

    // This method returns the destination of the ith link in a list of links
    {
        int returnInt = list.link; // need this assignment in the case i=0
        LinkNode storageLink = list;

        for (int j = 0; j < i; j++)
        {
            // If the link we are looking at is empty, then there is no ith link
            // (There are no more links in the list), and we return -1, indicating
            // that the ith link does not exist.

            if (storageLink.link == -1)
            {
                returnInt = -1;
            }
            else // i.e. there are more links in the list
            {
                storageLink = storageLink.nextNode(storageLink);
                returnInt = storageLink.link;
            }
        }
        return returnInt;
    }
}
```

```

// GET LINK STRENGTH METHOD
public static int getStrength(int i, LinkNode list)

// This method returns the strength of the ith link in a particular list of links
{
    int returnInt = list.weight; // need this assignment in the case i=0
    LinkNode storageLink = list;

    for (int j = 0; j < i; j++)
    {
        if (storageLink.link == -1) // i.e. no more links in the list
        {
            returnInt = -1;
        }
        else
        {
            storageLink = storageLink.nextNode(storageLink);
            returnInt = storageLink.weight;
        }
    }
    return returnInt;
}

// METHOD TO ASK THE USER TO ENTER THE NUMBER OF NODES
public static int Ask_Nodes() throws Exception
{
    int inputInt;

    // Ask the user to enter the Number of nodes in the list
    System.out.println();
    System.out.print("Please enter the number of nodes in the Test Graph...");

    try
    {
        inputInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
        inputInt = -1;
    }

    if (inputInt < 1)
    {
        System.out.println("Incorrect Input. Please enter a positive integer");
        inputInt = Ask_Nodes(); // call the method again
        return inputInt;
    }
    else // i.e. the user has entered correct input
    {
        numberOfNodes = inputInt;
        System.out.println();
        return inputInt;
    }
}

// ENTER LINKS METHOD
public static int EnterLinks(int limit) throws Exception

// This method allows the user to enter a departure node for a link
{
    int enterInt = 0;

    try
    {
        enterInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
    }

    if ((enterInt < 1) || (enterInt > limit))
    {
        System.out.println("Incorrect Input. Please enter a positive integer between 1 and " + limit);
        enterInt = EnterLinks(limit); // call the method again
    }
    return enterInt;
}

```

```
// INSERT LINK METHOD
```

```
public static void InsertLink() throws Exception
```

```
{  
    // Ask the user for the departure node of the link  
    System.out.println("From which node does the link go from?");  
    System.out.println("Enter a number between 1 and " + numberOfNodes);  
    int departure = EnterLinks(numberOfNodes);  
    System.out.println();  
  
    // Ask the user for the destination node of the link  
    System.out.println("From which node does the link go to?");  
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + departure);  
    int destination = EnterLinks(numberOfNodes);  
    System.out.println();  
  
    // Let check denote whether the link from departure to destination already exists.  
    LinkNode checkLinks = TestGraphArray[departure-1];  
    int check = 0;  
  
    for (int z=0; z < numberOfNodes; z++)  
    {  
        int storageInt = getLink(z, checkLinks);  
        if (storageInt == destination-1)  
        {  
            check = 1;  
        }  
    }  
  
    if ((destination != departure) & (check == 0))  
    { // i.e. we can insert the link  
  
        LinkNode tempLinks = TestGraphArray[departure-1]; // get the list to insert the link into  
        tempLinks.insert(tempLinks, destination-1); // insert the link  
    }  
    else  
    {  
        if (destination == departure)  
        {  
            System.out.println("Cannot have a link from a node to the same node.");  
        }  
        else  
        {  
            System.out.println("The specified link already exists!");  
        }  
        System.out.println();  
    }  
}
```

```
// CHOOSE LINKS METHOD
```

```
public static void ChooseLinks() throws Exception
```

```
{  
    boolean carryOn = true;  
    String inputString = new String();  
  
    while (carryOn == true)  
    {  
        Display(); // Shows the Graph at this particular time  
        System.out.println();  
        System.out.println("Would you like to insert a link?");  
        System.out.println("Answer 'y' or 'n'");  
        inputString = BasicIo.readString();  
        System.out.println();  
  
        if (inputString.equals("y"))  
        {  
            InsertLink(); // Calls the above procedure  
        }  
        else  
        {  
            if (inputString.equals("n"))  
            {  
                carryOn = false;  
            }  
            else  
            {  
                System.out.println("Incorrect Input. Try Again");  
                System.out.println();  
                carryOn = true;  
            }  
        }  
    }  
}
```

```

// ***
// TEST GRAPH METHOD
// ***
public static void TestGraph() throws Exception
{
    // Ask the user how many nodes he or she wants
    int input = Ask_Nodes();

    // Create an array with "input" amount of rows
    TestGraphArray = new LinkNode[input];

    // For each row we must initisliase and create links
    for (int i = 0; i < input; i++)
    {
        TestGraphArray[i] = new LinkNode();
        LinkNode insertLinks = new LinkNode();

        // For each created node, we create links to every other node
        for(int k = 0; k < input; k++)
        {
            if (i != k) // i.e. do not create a link from a node to the same node
            {
                // Insert a link into the adjacency list
                insertLinks.insert(insertLinks, k);
            }
        }
        // Insert the links for a particular node
        TestGraphArray[i] = insertLinks;
    }
}

// ***
// READ GRAPH METHOD
// ***
public static void ReadGraph() throws Exception
{
    // Ask the user how many nodes he or she wants
    int input = Ask_Nodes();

    // Create an array with "input" amount of rows
    TestGraphArray = new LinkNode[input];

    for (int i=0; i < input; i++)
    {
        TestGraphArray[i] = new LinkNode();
    }

    // Now let the user decide on the links in the Graph
    ChooseLinks();
}

// ***
// DISPLAY METHOD
// ***
public static void Display()
{
    // For each Node we shall display its links sequentially
    for (int k=0; k < numberOfNodes; k++)
    {
        System.out.print("Node " + (k+1) + " has links to ");

        // Now get the links for a particular node:
        LinkNode storage = TestGraphArray[k];

        // We check to see which links exist and print them out
        for (int x=0; x < numberOfNodes; x++)
        {
            // For a particular link, we now get the destination and the strength of the link
            int storageInt = getLink(x, storage);
            int storageInt2 = getStrength(x, storage);

            // If a link exists to the particular node we are looking at....
            if (storageInt != -1)
            {
                // ...Print the destination of the link, then.....
                System.out.print("(" + storageInt + " ");

                // ....If a strength has been assigned to the link,.....
                if (storageInt2 != -1)
                {
                    // ...Print the strength of the link.
                    System.out.print("[" + storageInt2 + "], ");
                }
            }
        }
        System.out.println();
    }
}

```

```

// CHECK EXISTENCE OF LINK METHOD
public static int CheckExistence(int a, LinkNode inputLink)
{
    int returnInt = 0;
    LinkNode storage = inputLink;

    // Cycle through the list of links, looking to see if a particular link exists
    for (int y = 0; y < numberOfNodes; y++)
    {
        if(storage.link == a) // i.e. we have found a match!
        {
            returnInt = 1;
        }
        else
        {
            if(storage.link != -1) // i.e. if more links exists
            {
                storage = storage.nextElement; // look at the next link in the list.
            }
        }
    }
    return returnInt;
}

// ENTER LINK WEIGHT METHOD
public static void EnterWeight(int departure, int destination, LinkNode storage)
{
    // Ask the user for the strength of the link in question
    System.out.println("Enter the strength of the link (must be positive)");
    int strength = -1;

    try
    {
        strength = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
        strength = -1;
    }

    if (strength < 0)
    {
        System.out.println("Incorrect Input. Please try again");
    }
    else // i.e. the user has entered some correct input
    {
        // Get the list of links for the Node in question
        LinkNode storage2 = TestGraphArray[departure-1];

        // We must now cycle through the list, looking for the correct link
        for (int z = 1; z < numberOfNodes; z++)
        {
            if(storage.link == destination-1) // i.e. we have reached the correct link
            {
                // Place the link weight in the list of links
                storage.weight = strength;
            }
            else // i.e. have not yet reached the correct link
            {
                if(storage.link != -1)
                {
                    storage = storage.nextElement;
                }
            }
        }
    }
}

```

```

// ***
// ADD WEIGHT TO ARC METHOD
// ***

public static void AddWeight() throws Exception
{
    // Ask the user to enter a start point for a link
    System.out.println("From which node does the link go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int departure = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user to enter the end point for a link
    System.out.println("From which node does the link go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + departure);
    int destination = EnterLinks(numberOfNodes);
    System.out.println();

    // Get the list of links for the departure node
    LinkNode storage = TestGraphArray[departure-1];

    // See if a link to the destination node actually exists. We get a "1" back if it does
    int match = CheckExistence(destination-1, storage);

    if ((destination != departure) & (match == 1))
    {
        // Ask the user to enter a weight for the link in question
        EnterWeight(departure, destination, storage);
    }
    else
    {
        System.out.println("That link does not exist on the Graph!");
    }
}
}

```

MS-DOS Output

See exercise 3

Exercise 5: Finding the Shortest path

Program Code

For exercise 5, we have two slightly different implementations of the problem. Both implementations use the same **LinkNode** class, which is modified from exercise 3 to allow for the deletion of links. Both also use the same “*main*” class, which adds two new options to the menu system, relevant to this exercise. Where the implementations differ is in the Graph class and the optional use of the Node class

Files common to both implementations

```
/*
 *
 * Gareth Evans
 *
 * Started: 29th November 1999
 * Finished: 1st December 1999 (Revision 1.2)
 *
 * Assignment 3, Exercise 5
 *
 * LinkNode Class
 * Linked List Implementation for Node Links
 *
 */

public class LinkNode
{
    // Define Variables for each Node

    int link;
    int weight;
    LinkNode nextElement;

    // Constructor for creating a new node

    LinkNode()
    {
        link = -1;
        weight = -1;
        nextElement = null;
    }

    // INSERT METHOD

    public void insert(LinkNode whatList, int obj)
    {
        if (whatList.nextElement == null)           // Checks next element in list
        {
            whatList.link = obj;                     // Add element to list
            whatList.nextElement = new LinkNode(); // Create a fresh new node
        }
        else
        {
            insert(whatList.nextElement, obj);      // Calls the method again
        }
    }
}
```

```

// NEXT METHOD

public LinkNode nextNode(LinkNode whatList)
// pass in a node, want to return the next node
{
    return whatList.nextElement;
}

// DELETE METHOD (NEW!)

public void delete(LinkNode whatList, int obj)
// Deletes the obj'th element of the linked list
{
    LinkNode storage = new LinkNode();
    storage = whatList;
    for (int i=0; i < obj; i++) // traverse to the obj'th-1 element
    {
        storage = storage.nextElement;
    }
    storage.nextElement = null; // delete the obj'th element
}
}



---



/*
 *
 * Ioan Williams
 *
 * Started: 3rd December 1999
 * Finished: 3rd December 1999
 *
 * Assignment 3, Exercise 5
 *
 * Main Class for Exercise 5
 */

import java.bangor.*;

public class Exercise5
{
    // Define Global Variables
    static Graph ourGraph = new Graph();

    // GET INPUT METHOD

    public static void GetInput() throws Exception
    {
        String inputString = BasicIo.readString();

        if (inputString.equals("y"))
        {
            ourGraph.ReadGraph();
        }
        else
        {
            if (inputString.equals("n"))
            {
                ourGraph.TestGraph();
            }
            else
            {
                System.out.println();
                System.out.println("Incorrect Input. Try Again");
                System.out.println("Would you like to specify the links in the
                                     Graph?");

                System.out.print("Answer 'y' or 'n'...");
                GetInput();
            }
        }
    }
}

```

```

public static void main(String args[]) throws Exception
{
    System.out.println();
    System.out.println("Would you like to specify the links in the Graph?");
    System.out.print("Answer 'y' or 'n'...");
    GetInput();
    System.out.println();

    int Choice = -1;

    // MENU

    do // repeat this until option 5 is chosen
    {
        Choice = -1;

        System.out.println();
        System.out.println("MAIN MENU - PLEASE ENTER YOUR CHOICE.");
        System.out.println("-----");
        System.out.println();
        System.out.println("1. Display the Graph");
        System.out.println("2. Add (or change) an Arc Weight");
        System.out.println("3. Assign a Name to a Node");
        System.out.println("4. Find the Paths from one Node to another");
        System.out.println("5: Exit Program");
        System.out.println();

        try
        {
            Choice = BasicIo.readInteger();
        }

        catch (Exception exp) { }

        switch(Choice)
        {
            case 1:  ourGraph.Display();
                    break;

            case 2:  ourGraph.AddWeight();
                    break;

            case 3:  ourGraph.AssignInfo();
                    break;

            case 4:  ourGraph.FindPaths();
                    break;

            case 5:  System.out.println("The Program will now close...");
                    break;

            default: System.out.println("Sorry, Input is incorrect. Please
                                         Try Again. ");
        }
    }
    while (Choice != 5);
}
}

```

Implementation 1 files

In the *first* implementation, we modify the **Node** class from exercise 3, allowing names to be assigned to each node. We also modified the **Graph** class from exercise 3, adding the code needed to allow the two new options in the menu to work. The code for the *first* implementation follows on the next couple of pages.

```

/*
 *
 * Gareth Evans
 *
 * Started: 29th November 1999
 * Finished: 29th November 1999 (Revision 1.1)
 *
 * Assignment 3, Exercise 5
 *
 * Node Class
 * Linked List Implementation for Graph Nodes
 *
 */

public class Node
{
    // Define Variables for each Node

    int data;
    String information;
    LinkNode links;
    Node nextElement;

    // Constructor for creating a new node

    Node()
    {
        data = -1;
        nextElement = null;
        links = new LinkNode();
        information = new String();
    }

    // INSERT METHOD

    public void insert(Node whatList, int obj, LinkNode linkobj)
    {
        if (whatList.nextElement == null) // Checks next element in list
        {
            whatList.data = obj; // Add element to list
            whatList.nextElement = new Node(); // Create a fresh new node
            whatList.links = linkobj; // Add the node Links
        }
        else
        {
            insert(whatList.nextElement, obj, linkobj); // Calls the method again
        }
    }

    // INSERT LINKS

    public void insertList(Node whatList, int start, int check, LinkNode obj)
    {
        if (check == start)
        {
            whatList.links = obj;
        }
        else
        {
            check++;
            insertList(whatList.nextElement, start, check, obj);
        }
    }

    // NEXT METHOD

    public Node nextNode(Node whatList)
    // pass in a node, want to return the next node
    {
        return whatList.nextElement;
    }
}

```

```

/*
 *
 * Gareth Evans
 *
 * Started: 29th November 1999
 * Finished: 3rd December 1999 (Revision 2.4)
 *
 * Assignment 3, Exercise 5
 *
 * Graph Class
 * Linked List Implementation
 */

import java.bangor.*;

public class Graph
{
    // Declare Graph variables

    static Node TestGraphNode = new Node();
    static int numberOfNodes = 0;

    // Define two variables to use when finding the shortest path
    static int shortestPath = -1;
    static String shortestPathString = new String();

    // GET NODE METHOD

    public static LinkNode getNode(int i, Node list)

    // This method returns the links for a particular node in the graph.
    {
        Node storageNode = list;

        // Go to the node requested in the graph by using the following loop:
        for (int j = 0; j < i; j++)
        {
            storageNode = storageNode.nextNode(storageNode);
        }
        return storageNode.links;
    }

    // GET LINK METHOD

    public static int getLink(int i, LinkNode list)

    // This method returns the destination of the ith link in a list of links
    {
        int returnInt = list.link; // need this assignment in the case i=0
        LinkNode storageLink = list;

        for (int j = 0; j < i; j++)
        {
            // If the link we are looking at is empty, then there is no ith link
            // (There are no more links in the list), and we return -1, indicating
            // that the ith link does not exist.

            if (storageLink.link == -1)
            {
                returnInt = -1;
            }
            else // i.e. there are more links in the list
            {
                storageLink = storageLink.nextNode(storageLink);
                returnInt = storageLink.link;
            }
        }
        return returnInt;
    }
}

```

```

// GET LINK STRENGTH METHOD
public static int getStrength(int i, LinkNode list)

// This method returns the strength of the ith link in a particular list of links
{
    int returnInt = list.weight; // need this assignment in the case i=0
    LinkNode storageLink = list;

    for (int j = 0; j < i; j++)
    {
        if (storageLink.link == -1) // i.e. no more links in the list
        {
            returnInt = -1;
        }
        else
        {
            storageLink = storageLink.nextNode(storageLink);
            returnInt = storageLink.weight;
        }
    }
    return returnInt;
}

// METHOD TO ASK THE USER TO ENTER THE NUMBER OF NODES
public static int Ask_Nodes() throws Exception
{
    int inputInt;

    // Ask the user to enter the Number of nodes in the list
    System.out.println();
    System.out.print("Please enter the number of nodes in the Test Graph...");

    try
    {
        inputInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
        inputInt = -1;
    }

    if (inputInt < 1)
    {
        System.out.println("Incorrect Input. Please enter a positive integer");
        inputInt = Ask_Nodes(); // call the method again
        return inputInt;
    }
    else // i.e. the user has entered correct input
    {
        numberOfNodes = inputInt;
        System.out.println();
        return inputInt;
    }
}

// ENTER LINKS METHOD
public static int EnterLinks(int limit) throws Exception

// This method allows the user to enter a departure node for a link
{
    int enterInt = 0;

    try
    {
        enterInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
    }

    if ((enterInt < 1) || (enterInt > limit))
    {
        System.out.println("Incorrect Input. Please enter a positive integer between 1 and " + limit);
        enterInt = EnterLinks(limit); // call the method again
    }
    return enterInt;
}

```

```
// INSERT LINK METHOD
```

```
public static void InsertLink() throws Exception
{
    // Ask the user for the departure node of the link
    System.out.println("From which node does the link go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int departure = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user for the destination node of the link
    System.out.println("From which node does the link go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + departure);
    int destination = EnterLinks(numberOfNodes);
    System.out.println();

    // Let check denote whether the link from departure to destination already exists.
    LinkNode checkLinks = getNode(departure-1, TestGraphNode);
    int check = 0;

    for (int z=0; z < numberOfNodes; z++)
    {
        int storageInt = getLink(z, checkLinks);
        if (storageInt == destination-1)
        {
            check = 1;
        }
    }

    if ((destination != departure) & (check == 0))
    { // i.e. we can insert the link

        LinkNode tempLinks = getNode(departure-1, TestGraphNode); // get list to insert the link into
        tempLinks.insert(tempLinks, destination-1); // insert the link

    }
    else
    {
        if (destination == departure)
        {
            System.out.println("Cannot have a link from a node to the same node.");
        }
        else
        {
            System.out.println("The specified link already exists!");
        }
        System.out.println();
    }
}
}
```

```
// CHOOSE LINKS METHOD
```

```
public static void ChooseLinks() throws Exception
{
    boolean carryOn = true;
    String inputString = new String();

    while (carryOn == true)
    {
        Display(); // Shows the Graph at this particular time
        System.out.println();
        System.out.println("Would you like to insert a link?");
        System.out.println("Answer 'y' or 'n'");
        inputString = BasicIo.readString();
        System.out.println();

        if (inputString.equals("y"))
        {
            InsertLink(); // Calls the above procedure
        }
        else
        {
            if (inputString.equals("n"))
            {
                carryOn = false;
            }
            else
            {
                System.out.println("Incorrect Input. Try Again");
                System.out.println();
                carryOn = true;
            }
        }
    }
}
}
```

```

// ***
// TEST GRAPH METHOD
// ***
public static void TestGraph() throws Exception
{
    // Ask the user how many nodes he or she wants
    int input = Ask_Nodes();

    // We now need to create "input" amount of new nodes
    for (int i = 0; i < input; i++)
    {
        LinkNode insertLinks = new LinkNode();

        // For each created node, we create links to every other node
        for(int k = 0; k < input; k++)
        {
            if (i != k) // i.e. do not create a link from a node to the same node
            {
                // Insert a link into the adjacency list
                insertLinks.insert(insertLinks, k);
            }
        }
        // Insert a node into the graph
        TestGraphNode.insert(TestGraphNode, i, insertLinks);
    }
}

// ***
// READ GRAPH METHOD
// ***
public static void ReadGraph() throws Exception
{
    // Ask the user how many nodes he or she wants
    int input = Ask_Nodes();

    // We now need to create "input" amount of new nodes
    for (int i = 0; i < input; i++)
    {
        // For each node, we create an empty list of lists

        LinkNode insertLinks = new LinkNode();
        TestGraphNode.insert(TestGraphNode, i, insertLinks);
    }
    // Now let the user decide on the links in the Graph
    ChooseLinks();
}

// ***
// DISPLAY METHOD
// ***
public static void Display()
{
    // For each Node we shall display its links sequentially
    for (int k=0; k < numberOfNodes; k++)
    {
        Node tempNode = TestGraphNode;
        for (int m=0; m < k; m++)
        {
            tempNode = tempNode.nextElement;
        }

        System.out.print("Node " + (k+1) + " " + tempNode.information + " has links to ");

        // Now get the links for a particular node:
        LinkNode storage = getNode(k, TestGraphNode);

        // We check to see which links exist and print them out
        for (int x=0; x < numberOfNodes; x++)
        {
            // For a particular link, we now get the destination and the strength of the link
            int storageInt = getLink(x, storage);
            int storageInt2 = getStrength(x, storage);

            // If a link exists to the particular node we are looking at....
            if (storageInt != -1)
            {
                // ...Print the destination of the link, then.....
                System.out.print((storageInt+1) + " ");

                // ....If a strength has been assigned to the link,.....
                if (storageInt2 != -1)
                {
                    // ...Print the strength of the link.
                    System.out.print("[ " + storageInt2 + " ], ");
                }
            }
        }
        System.out.println();
    }
}
}

```

```

// CHECK EXISTENCE OF LINK METHOD
public static int CheckExistence(int a, LinkNode inputLink)
{
    int returnInt = 0;
    LinkNode storage = inputLink;

    // Cycle through the list of links, looking to see if a particular link exists
    for (int y = 0; y < numberOfNodes; y++)
    {
        if(storage.link == a) // i.e. we have found a match!
        {
            returnInt = 1;
        }
        else
        {
            if(storage.link != -1) // i.e. if more links exists
            {
                storage = storage.nextElement; // look at the next link in the list.
            }
        }
    }
    return returnInt;
}

// ENTER LINK WEIGHT METHOD
public static void EnterWeight(int departure, int destination, LinkNode storage)
{
    // Ask the user for the strength of the link in question
    System.out.println("Enter the strength of the link (must be positive)");
    int strength = -1;

    try
    {
        strength = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
        strength = -1;
    }

    if (strength < 0)
    {
        System.out.println("Incorrect Input. Please try again");
    }
    else // i.e. the user has entered some correct input
    {
        // Get the list of links for the Node in question
        LinkNode storage2 = getNode(departure-1, TestGraphNode);

        // We must now cycle through the list, looking for the correct link
        for (int z = 1; z < numberOfNodes; z++)
        {
            if(storage.link == destination-1) // i.e. we have reached the correct link
            {
                // Place the link weight in the list of links
                storage.weight = strength;
            }
            else // i.e. have not yet reached the correct link
            {
                if(storage.link != -1)
                {
                    storage = storage.nextElement;
                }
            }
        }
    }
}

// ***
// ADD WEIGHT TO ARC METHOD
// ***
public static void AddWeight() throws Exception
{
    // Ask the user to enter a start point for a link
    System.out.println("From which node does the link go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int departure = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user to enter the end point for a link
    System.out.println("From which node does the link go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + departure);
    int destination = EnterLinks(numberOfNodes);
    System.out.println();
}

```

```

// Get the list of links for the departure node
LinkNode storage = getNode(departure-1, TestGraphNode);

// See if a link to the destination node actually exists. We get a "1" back if it does
int match = CheckExistence(destination-1, storage);

if ((destination != departure) & (match == 1))
{
    // Ask the user to enter a weight for the link in question
    EnterWeight(departure, destination, storage);
}
else
{
    System.out.println("That link does not exist on the Graph!");
}
}

// ***
// ADD NODE INFORMATION METHOD
// ***

public static void AssignInfo() throws Exception
{
    // Ask the user to enter a node to change it's information
    System.out.println();
    System.out.println("Which Node do you want to change the name of?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int change = EnterLinks(numberOfNodes);
    System.out.println();

    System.out.println("What name do you want to give Node " + change + "?");
    String name = BasicIO.readString();
    System.out.println();

    Node tempNode = TestGraphNode;
    for (int n = 0; n < change-1; n++)
    {
        tempNode = tempNode.nextElement;
    }
    tempNode.information = name;
}

// GET PATHS METHOD

public static void GetPaths(int a, int b, int[] remember, String description, int length)
{
    // Declare local variables
    int checkEnd = description.length(); // checkEnd holds the length of the traversal String
    int checkLength;
    String checkString = new String();
    checkString = checkString + (b+1);

    // ** PART 1 ** CHECK FOR SOLUTION
    // If the last letter in the traversal String is the same as where we are trying to
    // go to, then we may print out some results.

    if (description.substring((checkEnd-(checkString.length())),(checkEnd)).equals(checkString))
    {
        System.out.print("MATCH: " + description);
        if (length != 0) // print out length if non zero i.e. has been assigned
        {
            System.out.print(", Length: " + length);
        }
        System.out.println();

        checkLength = length;
        if (shortestPath == -1) // need this for the first match
        {
            shortestPath = checkLength;
            shortestPathString = description;
        }
        if (checkLength < shortestPath) // need this for subsequent matches
        {
            shortestPath = checkLength;
            shortestPathString = description;
        }
    }
}

```

```

// ** PART 2 **    FIND MORE SOLUTIONS
// If we didn't find any matches above, then we have to go through all the
// links for node 'a', calling this method for every link 'a' has.
remember[a] = 1; // indicate that a has now been visited
LinkNode tempList = getNode(a, TestGraphNode); // get a's links
for (int i=0; i < numberOfNodes; i++) // for every link,
{
    int tempInt = getLink(i, tempList);
    int check = 0;
    if (tempInt == -1) // i.e. link doesn't exist,...
    {
        check = 1; // ... so we pretend that the node has already been visited
    }
    else
    {
        check = remember[tempInt]; // check if the node has already been visited
    }
    if ((check == -1) & (tempInt != -1))
    // if node hasn't been visited and the node exists
    {
        // temporarily concatenate the traversal string
        String newDescription = new String();
        newDescription = "-" + (tempInt+1);

        // temporarily add to the length of the path
        int parameter = 0;
        if (tempInt == -1) // i.e. link doesn't exist
        {
            parameter = 0;
        }
        else
        {
            if ((getStrength(i, tempList) != -1) // i.e. has a link strength been assigned?
            {
                parameter = length + getStrength(i, tempList);
            }
        }
        // call the procedure again, passing in the next Node and parameters
        GetPaths(tempInt, b, remember, description+newDescription, parameter);
    }
}
// IMPORTANT - reset the array so now we can revisit 'a'
remember[a] = -1;
}
// ***
// FIND PATHS METHOD
// ***
public static void FindPaths() throws Exception
{
    // Create a list to store whether we have visited a node already or not.
    // The node will contain "numberOfNodes" elements and will contain 1 if
    // the node has already been visited; =1 otherwise.

    int[] rememberVisits = new int[numberOfNodes];
    for (int i=0; i < numberOfNodes; i++)
    {
        rememberVisits[i] = -1;
    }

    // Ask the user to enter a start point for the path
    System.out.println();
    System.out.println("From which node does the path go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int pathStart = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user to enter the end point for the path
    System.out.println("From which node does the path go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + pathStart);
    int pathEnd = EnterLinks(numberOfNodes);
    System.out.println();

    // Create a string which will store the route of traversal through the graph
    String s = new String();
    s = s + pathStart;

    GetPaths(pathStart-1, pathEnd-1, rememberVisits, s, 0);

    System.out.println();
    if (shortestPath != 0) // i.e. have weights been assigned?
    {
        System.out.println("The shortest path from " + pathStart + " to " + pathEnd + " was");
        System.out.println(shortestPathString + " with length " + shortestPath);
    }
    // reset variables for next time
    shortestPath = -1;
    shortestPathString = "";
}
}

```

Implementation 2 files

In the *second* implementation, we modify the **Graph** class used in the first implementation in exactly the same way as we modified the Graph class when going from exercise 3 to exercise 4. The modified bits are shown in [blue](#).

```
/*
 *
 * Gareth Evans
 *
 * Started: 29th November 1999
 * Finished: 3rd December 1999 (Revision 2.4)
 *
 * Assignment 3, Exercise 5
 *
 * Graph Class
 * Array Implementation
 */

import java.bangor.*;
public class Graph
{
    // Declare Graph variables

    static LinkNode[] TestGraphArray;
    static String[] NamesArray;
    static int numberOfNodes = 0;

    // Define two variables to use when finding the shortest path
    static int shortestPath = -1;
    static String shortestPathString = new String();

    // GET LINK METHOD

    public static int getLink(int i, LinkNode list)

    // This method returns the destination of the ith link in a list of links
    {
        int returnInt = list.link; // need this assignment in the case i=0
        LinkNode storageLink = list;

        for (int j = 0; j < i; j++)
        {
            // If the link we are looking at is empty, then there is no ith link
            // (There are no more links in the list), and we return -1, indicating
            // that the ith link does not exist.

            if (storageLink.link == -1)
            {
                returnInt = -1;
            }
            else // i.e. there are more links in the list
            {
                storageLink = storageLink.nextNode(storageLink);
                returnInt = storageLink.link;
            }
        }
        return returnInt;
    }

    // GET LINK STRENGTH METHOD

    public static int getStrength(int i, LinkNode list)

    // This method returns the strength of the ith link in a particular list of links
    {
        int returnInt = list.weight; // need this assignment in the case i=0
        LinkNode storageLink = list;

        for (int j = 0; j < i; j++)
        {
            if (storageLink.link == -1) // i.e. no more links in the list
            {
                returnInt = -1;
            }
            else
            {
                storageLink = storageLink.nextNode(storageLink);
                returnInt = storageLink.weight;
            }
        }
        return returnInt;
    }
}
```

```
// METHOD TO ASK THE USER TO ENTER THE NUMBER OF NODES
```

```
public static int Ask_Nodes() throws Exception
{
    int inputInt;

    // Ask the user to enter the Number of nodes in the list
    System.out.println();
    System.out.print("Please enter the number of nodes in the Test Graph...");

    try
    {
        inputInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
        inputInt = -1;
    }

    if (inputInt < 1)
    {
        System.out.println("Incorrect Input. Please enter a positive integer");
        inputInt = Ask_Nodes(); // call the method again
        return inputInt;
    }
    else // i.e. the user has entered correct input
    {
        numberOfNodes = inputInt;
        System.out.println();
        return inputInt;
    }
}
```

```
// ENTER LINKS METHOD
```

```
public static int EnterLinks(int limit) throws Exception

// This method allows the user to enter a departure node for a link
{
    int enterInt = 0;

    try
    {
        enterInt = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
    }

    if ((enterInt < 1) || (enterInt > limit))
    {
        System.out.println("Incorrect Input. Please enter a positive integer between 1 and " + limit);
        enterInt = EnterLinks(limit); // call the method again
    }
    return enterInt;
}
```

```
// INSERT LINK METHOD
```

```
public static void InsertLink() throws Exception
{
    // Ask the user for the departure node of the link
    System.out.println("From which node does the link go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int departure = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user for the destination node of the link
    System.out.println("From which node does the link go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + departure);
    int destination = EnterLinks(numberOfNodes);
    System.out.println();

    // Let check denote whether the link from departure to destination already exists.
    LinkNode checkLinks = TestGraphArray[departure-1];
    int check = 0;

    for (int z=0; z < numberOfNodes; z++)
    {
        int storageInt = getLink(z, checkLinks);
        if (storageInt == destination-1)
        {
            check = 1;
        }
    }
}
```

```

if ((destination != departure) & (check == 0))
{ // i.e. we can insert the link

    LinkNode tempLinks = TestGraphArray[departure-1]; // get the list to insert the link into
    tempLinks.insert(tempLinks, destination-1); // insert the link
}
else
{
    if (destination == departure)
    {
        System.out.println("Cannot have a link from a node to the same node.");
    }
    else
    {
        System.out.println("The specified link already exists!");
    }
    System.out.println();
}
}

// CHOOSE LINKS METHOD

public static void ChooseLinks() throws Exception
{
    boolean carryOn = true;
    String inputString = new String();

    while (carryOn == true)
    {
        Display(); // Shows the Graph at this particular time
        System.out.println();
        System.out.println("Would you like to insert a link?");
        System.out.println("Answer 'y' or 'n'");
        inputString = BasicIo.readString();
        System.out.println();

        if (inputString.equals("y"))
        {
            InsertLink(); // Calls the above procedure
        }
        else
        {
            if (inputString.equals("n"))
            {
                carryOn = false;
            }
            else
            {
                System.out.println("Incorrect Input. Try Again");
                System.out.println();
                carryOn = true;
            }
        }
    }
}

// ***
// TEST GRAPH METHOD
// ***

public static void TestGraph() throws Exception
{
    // Ask the user how many nodes he or she wants
    int input = Ask_Nodes();

    // Create an array with "input" amount of rows
    TestGraphArray = new LinkNode[input];
    NamesArray = new String[input];

    // For each row we must initisliase and create links
    for (int i = 0; i < input; i++)
    {
        TestGraphArray[i] = new LinkNode();
        NamesArray[i] = new String();
        LinkNode insertLinks = new LinkNode();

        // For each created node, we create links to every other node
        for(int k = 0; k < input; k++)
        {
            if (i != k) // i.e. do not create a link from a node to the same node
            {
                // Insert a link into the adjacency list
                insertLinks.insert(insertLinks, k);
            }
        }
        // Insert the links for a particular node
        TestGraphArray[i] = insertLinks;
    }
}

```

```

// ***
// READ GRAPH METHOD
// ***

public static void ReadGraph() throws Exception
{
    // Ask the user how many nodes he or she wants
    int input = Ask_Nodes();

    // Create an array with "input" amount of rows
    TestGraphArray = new LinkNode[input];
    NamesArray = new String[input];

    for (int i=0; i < input; i++)
    {
        TestGraphArray[i] = new LinkNode();
        NamesArray[i] = new String();
    }

    // Now let the user decide on the links in the Graph
    ChooseLinks();
}

// ***
// DISPLAY METHOD
// ***

public static void Display()
{
    // For each Node we shall display its links sequentially
    for (int k=0; k < numberOfNodes; k++)
    {
        System.out.print("Node " + (k+1) + " " + NamesArray[k] + " has links to ");

        // Now get the links for a particular node:
        LinkNode storage = TestGraphArray[k];

        // We check to see which links exist and print them out
        for (int x=0; x < numberOfNodes; x++)
        {
            // For a particular link, we now get the destination and the strength of the link
            int storageInt = getLink(x, storage);
            int storageInt2 = getStrength(x, storage);

            // If a link exists to the particular node we are looking at....
            if (storageInt != -1)
            {
                // ...Print the destination of the link, then.....
                System.out.print((storageInt+1) + " ");

                // ....If a strength has been assigned to the link,.....
                if (storageInt2 != -1)
                {
                    // ...Print the strength of the link.
                    System.out.print("[ " + storageInt2 + " ], ");
                }
            }
        }
        System.out.println();
    }
}

// CHECK EXISTENCE OF LINK METHOD

public static int CheckExistence(int a, LinkNode inputLink)
{
    int returnInt = 0;
    LinkNode storage = inputLink;

    // Cycle through the list of links, looking to see if a particular link exists
    for (int y = 0; y < numberOfNodes; y++)
    {
        if(storage.link == a) // i.e. we have found a match!
        {
            returnInt = 1;
        }
        else
        {
            if(storage.link != -1) // i.e. if more links exists
            {
                storage = storage.nextElement; // look at the next link in the list.
            }
        }
    }
    return returnInt;
}

```

```
// ENTER LINK WEIGHT METHOD
```

```
public static void EnterWeight(int departure, int destination, LinkNode storage)
{
    // Ask the user for the strength of the link in question
    System.out.println("Enter the strength of the link (must be positive)");
    int strength = -1;

    try
    {
        strength = BasicIo.readInteger();
    }

    catch (Exception exp)
    {
        strength = -1;
    }

    if (strength < 0)
    {
        System.out.println("Incorrect Input. Please try again");
    }
    else // i.e. the user has entered some correct input
    {
        // Get the list of links for the Node in question
        LinkNode storage2 = TestGraphArray[departure-1];

        // We must now cycle through the list, looking for the correct link
        for (int z = 1; z < numberOfNodes; z++)
        {
            if(storage.link == destination-1) // i.e. we have reached the correct link
            {
                // Place the link weight in the list of links
                storage.weight = strength;
            }
            else // i.e. have not yet reached the correct link
            {
                if(storage.link != -1)
                {
                    storage = storage.nextElement;
                }
            }
        }
    }
}
```

```
// ***
```

```
// ADD WEIGHT TO ARC METHOD
```

```
// ***
```

```
public static void AddWeight() throws Exception
```

```
{
    // Ask the user to enter a start point for a link
    System.out.println("From which node does the link go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int departure = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user to enter the end point for a link
    System.out.println("From which node does the link go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + departure);
    int destination = EnterLinks(numberOfNodes);
    System.out.println();

    // Get the list of links for the departure node
    LinkNode storage = TestGraphArray[departure-1];

    // See if a link to the destination node actually exists. We get a "1" back if it does
    int match = CheckExistence(destination-1, storage);

    if ((destination != departure) & (match == 1))
    {
        // Ask the user to enter a weight for the link in question
        EnterWeight(departure, destination, storage);
    }
    else
    {
        System.out.println("That link does not exist on the Graph!");
    }
}
```

```

// ***
// ADD NODE INFORMATION METHOD
// ***

public static void AssignInfo() throws Exception
{
    // Ask the user to enter a node to change it's information
    System.out.println();
    System.out.println("Which Node do you want to change the name of?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int change = EnterLinks(numberOfNodes);
    System.out.println();

    System.out.println("What name do you want to give Node " + change + "?");
    String name = BasicIO.readString();
    System.out.println();

    NamesArray[change-1] = name;
}

// GET PATHS METHOD

public static void GetPaths(int a, int b, int[] remember, String description, int length)
{
    // Declare local variables
    int checkEnd = description.length(); // checkEnd holds the length of the traversal String
    int checkLength;
    String checkString = new String();
    checkString = checkString + (b+1);

    // ** PART 1 **    CHECK FOR SOLUTION
    //
    // If the last letter in the traversal String is the same as where we are trying to
    // go to, then we may print out some results.

    if (description.substring((checkEnd-(checkString.length())),(checkEnd)).equals(checkString))
    {
        System.out.print("MATCH: " + description);
        if (length != 0) // print out length if non zero i.e. has been assigned
        {
            System.out.print(", Length: " + length);
        }
        System.out.println();

        checkLength = length;
        if (shortestPath == -1) // need this for the first match
        {
            shortestPath = checkLength;
            shortestPathString = description;
        }
        if (checkLength < shortestPath) // need this for subsequent matches
        {
            shortestPath = checkLength;
            shortestPathString = description;
        }
    }

    // ** PART 2 **    FIND MORE SOLUTIONS
    //
    // If we didn't find any matches above, then we have to go through all the
    // links for node 'a', calling this method for every link 'a' has.

    remember[a] = 1; // indicate that a has now been visited
    LinkNode tempList = TestGraphArray[a]; // get a's links

    for (int i=0; i < numberOfNodes; i++) // for every link,
    {
        int tempInt = getLink(i, tempList);
        int check = 0;
        if (tempInt == -1) // i.e. link doesn't exist,...
        {
            check = 1; // ... so we pretend that the node has already been visited
        }
        else
        {
            check = remember[tempInt]; // check if the node has already been visited
        }
    }
}

```

```

    if ((check == -1) & (tempInt != -1))
    // if node hasn't been visited and the node exists
    {
        // temporarily concatenate the traversal string
        String newDescription = new String();
        newDescription = "-" + (tempInt+1);

        // temporarily add to the length of the path
        int parameter = 0;
        if (tempInt == -1) // i.e. link doesn't exist
        {
            parameter = 0;
        }
        else
        {
            if ((getStrength(i, tempList)) != -1) // i.e. has a link strength been assigned?
            {
                parameter = length + getStrength(i, tempList);
            }
        }
        // call the procedure again, passing in the next Node and parameters
        GetPaths(tempInt, b, remember, description+newDescription, parameter);
    }
}
// IMPORTANT - reset the array so now we can revisit 'a'
remember[a] = -1;
}

// ***
// FIND PATHS METHOD
// ***

public static void FindPaths() throws Exception
{
    // Create a list to store whether we have visited a node already or not.
    // The node will contain "numberOfNodes" elements and will contain 1 if
    // the node has already been visited; =1 otherwise.

    int[] rememberVisits = new int[numberOfNodes];
    for (int i=0; i < numberOfNodes; i++)
    {
        rememberVisits[i] = -1;
    }

    // Ask the user to enter a start point for the path
    System.out.println();
    System.out.println("From which node does the path go from?");
    System.out.println("Enter a number between 1 and " + numberOfNodes);
    int pathStart = EnterLinks(numberOfNodes);
    System.out.println();

    // Ask the user to enter the end point for the path
    System.out.println("From which node does the path go to?");
    System.out.println("Enter a number between 1 and " + numberOfNodes + ", excluding " + pathStart);
    int pathEnd = EnterLinks(numberOfNodes);
    System.out.println();

    // Create a string which will store the route of traversal through the graph
    String s = new String();
    s = s + pathStart;

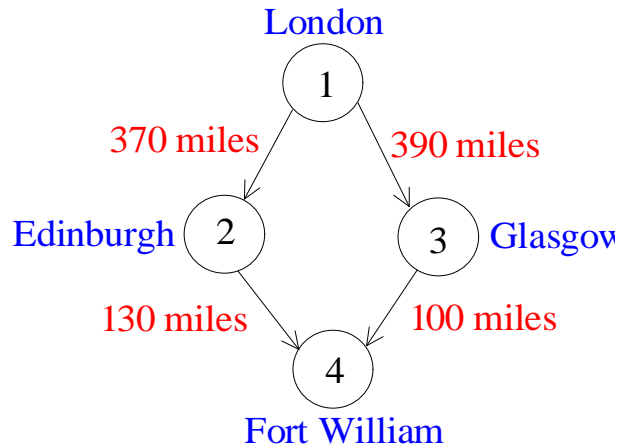
    GetPaths(pathStart-1, pathEnd-1, rememberVisits, s, 0);

    System.out.println();
    if (shortestPath != 0) // i.e. have weights been assigned?
    {
        System.out.println("The shortest path from " + pathStart + " to " + pathEnd + " was");
        System.out.println(shortestPathString + " with length " + shortestPath);
    }
    // reset variables for next time
    shortestPath = -1;
    shortestPathString = "";
}
}

```

MS-DOS Output

For output, let us solve the question asked about the shortest journey from **London** to **Fort William** via either **Glasgow** or **Edinburgh**. For this problem, we have the tree shown below.



Looking at the above diagram, it is obvious that the optimal route through the Graph is using the path **1-3-4** i.e. London → Glasgow → Fort William. Let us now see if the program gives us back this answer.

```
D:\Gareth's Documents\COLEG\E2027\Assignment
3\ex5>List>java Exercise5
```

```
Would you like to specify the links in the Graph?
Answer 'y' or 'n'...y
```

```
Please enter the number of nodes in the Test
Graph...4
```

```
Node 1 has links to
Node 2 has links to
Node 3 has links to
Node 4 has links to
```

```
Would you like to insert a link?
Answer 'y' or 'n'
y
```

```
From which node does the link go from?
Enter a number between 1 and 4
1
```

```
From which node does the link go to?
Enter a number between 1 and 4, excluding 1
2
```

```
Node 1 has links to 2
Node 2 has links to
Node 3 has links to
Node 4 has links to
```

```
Would you like to insert a link?
Answer 'y' or 'n'
y
```

```
From which node does the link go from?
Enter a number between 1 and 4
1
```

```
From which node does the link go to?
Enter a number between 1 and 4, excluding 1
3
```

```
Node 1 has links to 2 3
Node 2 has links to
Node 3 has links to
Node 4 has links to
```

```
Would you like to insert a link?
Answer 'y' or 'n'
y
```

```
From which node does the link go from?
Enter a number between 1 and 4
2
```

```
From which node does the link go to?
Enter a number between 1 and 4, excluding 2
4
```

```
Node 1 has links to 2 3
Node 2 has links to 4
Node 3 has links to
Node 4 has links to
```

```
Would you like to insert a link?
Answer 'y' or 'n'
y
```

```
From which node does the link go from?
Enter a number between 1 and 4
3
```

```
From which node does the link go to?
Enter a number between 1 and 4, excluding 3
4
```

```
Node 1 has links to 2 3
Node 2 has links to 4
Node 3 has links to 4
Node 4 has links to
```

```
Would you like to insert a link?
Answer 'y' or 'n'
n
```

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

3

Which Node do you want to change the name of?

Enter a number between 1 and 4

1

What name do you want to give Node 1?

London

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

3

Which Node do you want to change the name of?

Enter a number between 1 and 4

2

What name do you want to give Node 2?

Glasgow

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

3

Which Node do you want to change the name of?

Enter a number between 1 and 4

3

What name do you want to give Node 3?

Edinburgh

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

3

Which Node do you want to change the name of?

Enter a number between 1 and 4

4

What name do you want to give Node 4?

Fort William

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

1

Node 1 London has links to 2 3

Node 2 Glasgow has links to 4

Node 3 Edinburgh has links to 4

Node 4 Fort William has links to

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 4

1

From which node does the link go to?

Enter a number between 1 and 4, excluding 1

2

Enter the strength of the link (must be positive)

390

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 4

1

From which node does the link go to?

Enter a number between 1 and 4, excluding 1

3

Enter the strength of the link (must be positive)

370

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 4

2

From which node does the link go to?

Enter a number between 1 and 4, excluding 2

4

Enter the strength of the link (must be positive)

100

MAIN MENU - PLEASE ENTER YOUR CHOICE.

1. Display the Graph
2. Add (or change) an Arc Weight
3. Assign a Name to a Node
4. Find the Paths from one Node to another
- 5: Exit Program

2

From which node does the link go from?

Enter a number between 1 and 4

3

From which node does the link go to?

Enter a number between 1 and 4, excluding 3

4

Enter the strength of the link (must be positive)

130

MAIN MENU - PLEASE ENTER YOUR CHOICE.

- 1. Display the Graph
 - 2. Add (or change) an Arc Weight
 - 3. Assign a Name to a Node
 - 4. Find the Paths from one Node to another
 - 5: Exit Program
- 1
- Node 1 London has links to 2 [390], 3 [370]
 Node 2 Glasgow has links to 4 [100],
 Node 3 Edinburgh has links to 4 [130],
 Node 4 Fort William has links to

MAIN MENU - PLEASE ENTER YOUR CHOICE.

- 1. Display the Graph
- 2. Add (or change) an Arc Weight
- 3. Assign a Name to a Node
- 4. Find the Paths from one Node to another
- 5: Exit Program

4

From which node does the path go from?

Enter a number between 1 and 4

1

From which node does the path go to?

Enter a number between 1 and 4, excluding 1
4

MATCH: 1-2-4, Length: 490
MATCH: 1-3-4, Length: 500

The shortest path from 1 to 4 was
1-2-4 with length 490

MAIN MENU - PLEASE ENTER YOUR CHOICE.

- 1. Display the Graph
- 2. Add (or change) an Arc Weight
- 3. Assign a Name to a Node
- 4. Find the Paths from one Node to another
- 5: Exit Program

5
The Program will now close...

D:\Gareth's Documents\COLEG\E2027\Assignment
3\ex5>List>

Note: The journey distances were calculated using *Softkey's Journey Planner*, and rounded up to the nearest 10. Here is an example calculation of a journey:

