



INFORMATICS
INSTITUTE OF
TECHNOLOGY

UNIVERSITY OF
WESTMINSTER[®]

Informatics Institute of Technology

Department of computing

(B.Eng.)Software Engineering

**Module : 5SENG003C.2 Algorithms, Design and
Implementation**

Module Leader : Mr. Ragu Sivaraman

INDIVIDUAL COURSEWORK

Student ID	-: 20200516
Student UoW ID	-: w1830180
Student Name	-: Dinil Hansindu Perera
Tutorial Group	-: SE Group I

Brief description of the data structure (Adjacency Matrix) utilised in implementation

Data structures are used to organise and handle data efficiently, and they are an important part of many methods. A table is used to show a line in an adjacency matrix. The table is made up of rows and columns, and each cell is made up of a two-dimensional collection. If there is a connection between two points, we put a True in the cell that goes with them. If they aren't linked, we write "False." This helps us keep track of how the points in the graph are linked.

A big benefit of using an adjacency matrix to describe a graph is that it is easy to see if there is a link between two points. This is because we only need to look at one number in the table to see if there is a link. Since this is easy to do quickly, adjacency matrices are great for graphs with a lot of links. Overall, the decision to use an adjacency matrix to describe a graph will rest on how the graph is made and what operations will be done on it.

Brief explanation of algorithm (sink elimination) used in implementation

A straightforward and effective method for determining whether or not a directed graph is acyclic is the sink elimination technique. In order to eliminate all remaining sinks from the network, the algorithm continually removes sinks (vertices with no outgoing edges) from it. There will always be at least one sink in an acyclic network, which may be eliminated with each cycle.

We can carry on doing this until there are no more vertices because removing a sink does not cause any new cycles to form in the graph. The graph is empty at this point, therefore we may infer that it is acyclic. However, if the graph has a cycle, there will always be at least one vertex that is not a sink, making it impossible for us to eliminate all vertices from the graph. If this happens, the process will end with at least one vertex of the graph still present, proving that it is cyclic.

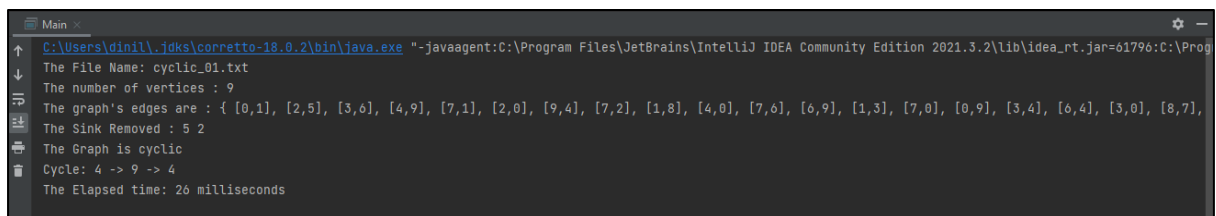
To maintain track of visited vertices and vertices that are presently on the call stack, the helper method makes use of two Boolean arrays. The current path being travelled is also tracked using a list. A cycle is recognised and the route from the cycle's start vertex to its end vertex is returned as an integer array if a visited vertex is found that is on the call stack. The algorithm returns null if no cycle is detected in the graph.

Experimenting with the Algorithm on Some Simple Test Cases

Example 1 (Cyclic Graph) :

```
0 1
2 5
3 6
4 9
7 1
2 0
9 4
7 2
1 8
4 0
7 6
6 9
1 3
7 0
0 9
3 4
6 4
3 0
8 7
6 1
```

Graph Edges



```
Main x
C:\Users\dinil\.jdk\corretto-18.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2021.3.2\lib\idea_rt.jar=61796:C:\Progr
The File Name: cyclic_01.txt
The number of vertices : 9
The graph's edges are : { [0,1], [2,5], [3,6], [4,9], [7,1], [2,0], [9,4], [7,2], [1,8], [4,0], [7,6], [6,9], [1,3], [7,0], [0,9], [3,4], [6,4], [3,0], [8,7],
The Sink Removed : 5 2
The Graph is cyclic
Cycle: 4 -> 9 -> 4
The Elapsed time: 26 milliseconds
```

Output

Example 2 (Acyclic Graph):

```
0 1
0 2
1 2
1 3
1 4
2 5
3 4
7 4
9 2
1 8
```

Graph Edges

```
The File Name: acyclic_01.txt
The number of vertices : 8
The graph's edges are : { [0,1], [0,2], [1,2], [1,3], [1,4], [2,5], [3,4], [7,4], [9,2], [1,8] }
The Sink Removed : 8 6 5 4 7 3 2 1
The Graph is acyclic
The Elapsed time: 3 milliseconds
```

Output

An evaluation of how well algorithms are designed and used

Where n is the size of the input array, the code's method has a time complexity of $O(n^2)$. This occurs because the outer loop only goes through the complete array once, whereas the inner loop goes through progressively smaller increments of the array with each iteration of the outer loop.

The doubling hypothesis may be used to do an empirical analysis of the algorithm's efficiency. This is done by timing how long the method needs to process arrays that are progressively larger, and then comparing the ratios of those durations. Theoretical research suggests that the procedure has a quadratic time complexity if the running time roughly doubles for each doubling of the input size.