Informatics Institute of Technology

Department of Computing

(BEng) in Software Engineering

5SENG003C: Algorithms: Theory, Design and Implementation

Module Leader: Mr. Ragu Sivaraman

Date of Submission: 07th May 2023

CW – Report

Name	: Didula Thaveesha
UoW ID	: w1870577
IIT ID	: 20210174
Tutorial group	: Group I

Justifying the choice of data structure and algorithm

Data structure – Adjacency List

Identifying whether or not the provided graph is acyclic is the current challenge. In this case, the directed graph was represented using an adjacency list data structure. An adjacency list connects each node in a network to a set of node neighbors. This enables efficient representation of sparse graphs since it only preserves the vertices and edges that really exist in the graph. The graph is represented as an adjacency list, which is a set of linked lists where each vertex is linked to the set of vertex neighbors. This data structure is effective in terms of both space and temporal complexity since it only retains the information about the edges that are strictly essential.

Additionally, the Depth First Search method finds cycles more quickly using this data structure than it does when it iterating through each node's list of neighbors, which requires an additional O (Nodes + Edges) of time. An adjacency matrix model would require more memory for sparse graphs and is less efficient than an adjacency list representation since it requires O (Nodes2) time to detect whether an edge exists between two Nodes. Algorithm – Depth First Search (DFS)

Each graph node is visited and marked as having been visited by the procedure. Additionally, it keeps track of the nodes that are currently being traveled along the path. If it encounters a node that has already been visited and is on the current route, a cycle has been found in the graph. The DFS algorithm is ideal for this purpose because it can swiftly search all linked nodes in the network and keep track of visited and unvisited nodes. Additionally, it makes use of a stack data structure to keep track of the path that is currently being looked at, making it possible to identify cycles in the graph effectively.

Why DFS over Sink Elimination Algorithm?

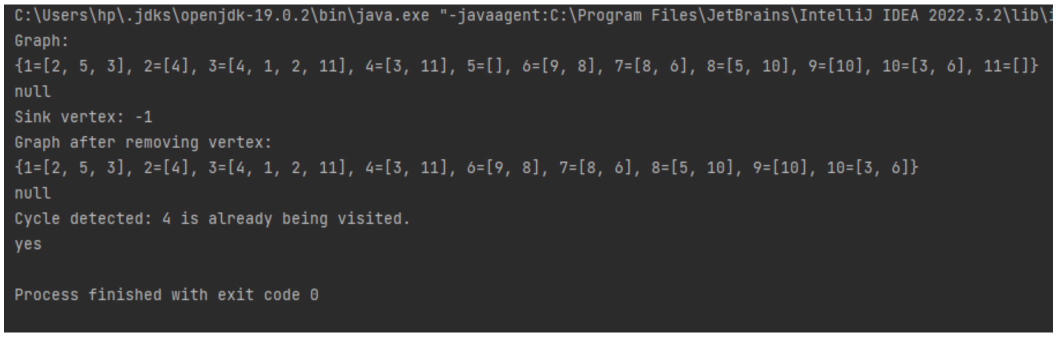
Sink elimination may not be the most efficient approach in terms of time complexity. When N is the entire number of network Nodes, the worst-case temporal complexity of it is O (N2). This is because constantly removing sinks until there are no vertices left may not be efficient, especially if the network has many sinks. While N is the number of vertices and E is the number of edges in the graph, DFS has an O(N+E) time complexity. The easiest way to determine if a given graph is acyclic is to use this technique, which works for both directed and undirected graphs..

Run of the algorithm

Cyclic

Input File:

```
1 2
1 5
1 3
2 4
3 4
 3 1
4 3
```



```
C:\Users\hp\.jdks\openjdk-19.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.3.2\lib\i
Graph:
{1=[2, 5, 3], 2=[4], 3=[4, 1, 2, 11], 4=[3, 11], 5=[], 6=[9, 8], 7=[8, 6], 8=[5, 10], 9=[10], 10=[3, 6], 11=[]}
null
Sink vertex: -1
Graph after removing vertex:
{1=[2, 5, 3], 2=[4], 3=[4, 1, 2, 11], 4=[3, 11], 6=[9, 8], 7=[8, 6], 8=[5, 10], 9=[10], 10=[3, 6]}
null
Cycle detected: 4 is already being visited.
yes

Process finished with exit code 0
```

6 9

7 8

7 6

8 5

10 3

10 6

8 10

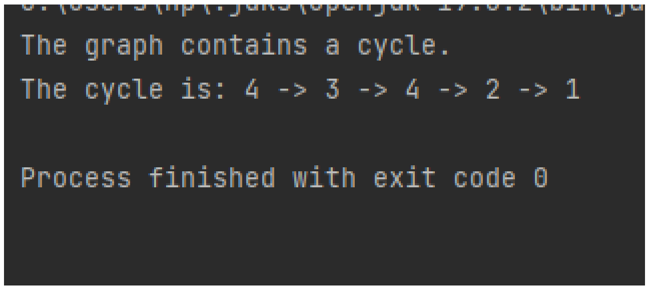
6 8

9 10

3 2

3 11

4 11



```
The graph contains a cycle.
The cycle is: 4 -> 3 -> 4 -> 2 -> 1

Process finished with exit code 0
```

Acyclic

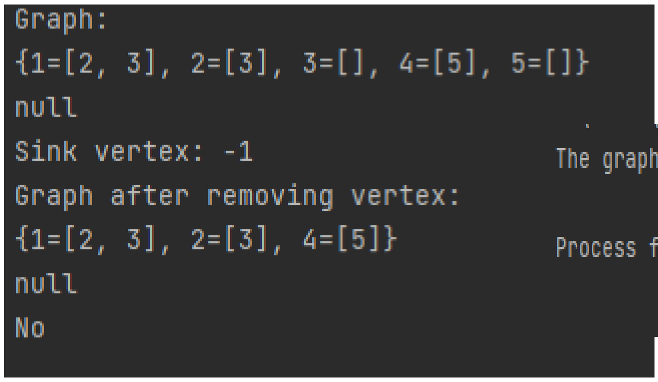
Input File: Output:

1 2

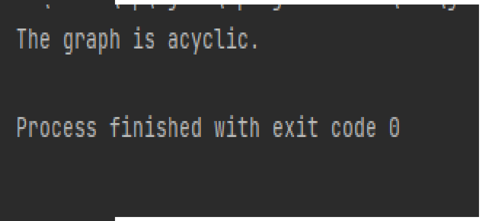
2 3

1 3

4 5



```
Graph:
{1=[2, 3], 2=[3], 3=[], 4=[5], 5=[]}
null
Sink vertex: -1
Graph after removing vertex:
{1=[2, 3], 2=[3], 4=[5]}
null
No
```



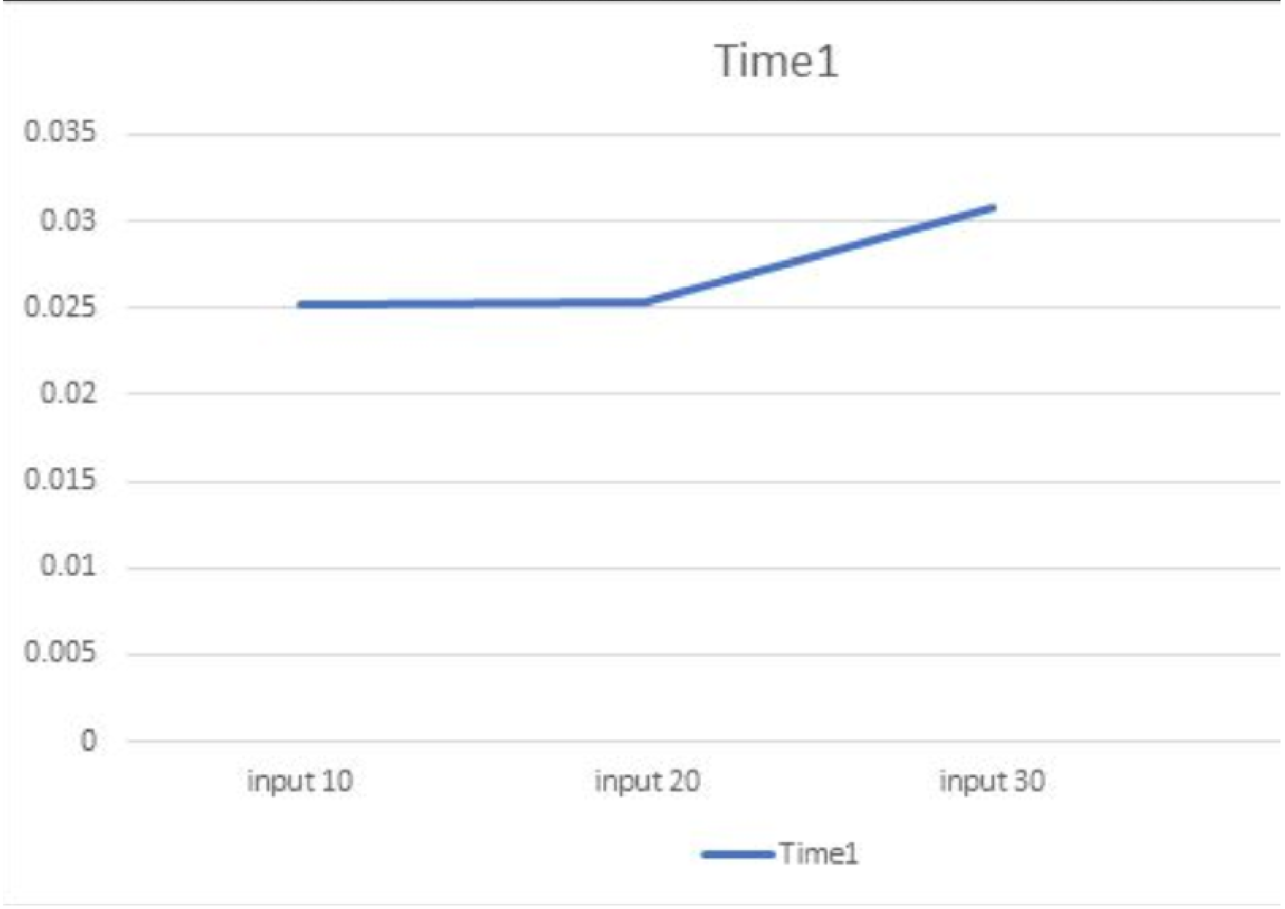
```
The graph is acyclic.

Process finished with exit code 0
```

Performance analysis

The effectiveness of the DFS approach for evaluating whether or not a graph is cyclic may be examined using theoretical considerations. Where V is the number of graph vertices and E is the number of graph edges, the DFS method has an O (V+E) time complexity. In the worst scenario, the DFS algorithm will only visit each vertex and edge once, giving it an O(V+E) temporal complexity. The algorithm's temporal complexity therefore varies with the amount of the input..

The DFS algorithm's spatial complexity, however, can raise some questions. The maximum size of the stack that the method may use to keep track of the visited vertices is determined by the length of the longest path in the graph. In the worst case, the DFS algorithm must travel over each vertex of a linear chain graph before turning around. As a result, the stack will be O (V) in size, which might provide issues for very large graphs. However, the approach's space complexity is regarded acceptable because the average stack size is substantially smaller than the total number of vertices. The DFS method is flawless, with a time complexity of O(V+E) and a space complexity of O(V). The approach may also be put through a worst-case analysis, which considers the graph's worst-case number of edges and vertices. In the worst case, the graph is complete and has V(V-1) edges. The algorithm has a quadratic temporal complexity of O(V2) in this case. However, this is unlikely to occur in practice because most graphs are sparse and contain a lot fewer edges than a full graph.


Time1
input 10
input 20
input 30
Time1