INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

5SENG003C.2 Algorithms: Theory, Design, and Implementation

Coursework – Individual Report

Student Name: B. K. Praveen Navanjana

IIT Index No: 20200877

UOW Index No: w1870590

Module Leader: Mr. Klaus Draeger

Task 6

a) A short explanation of the data structure and algorithm

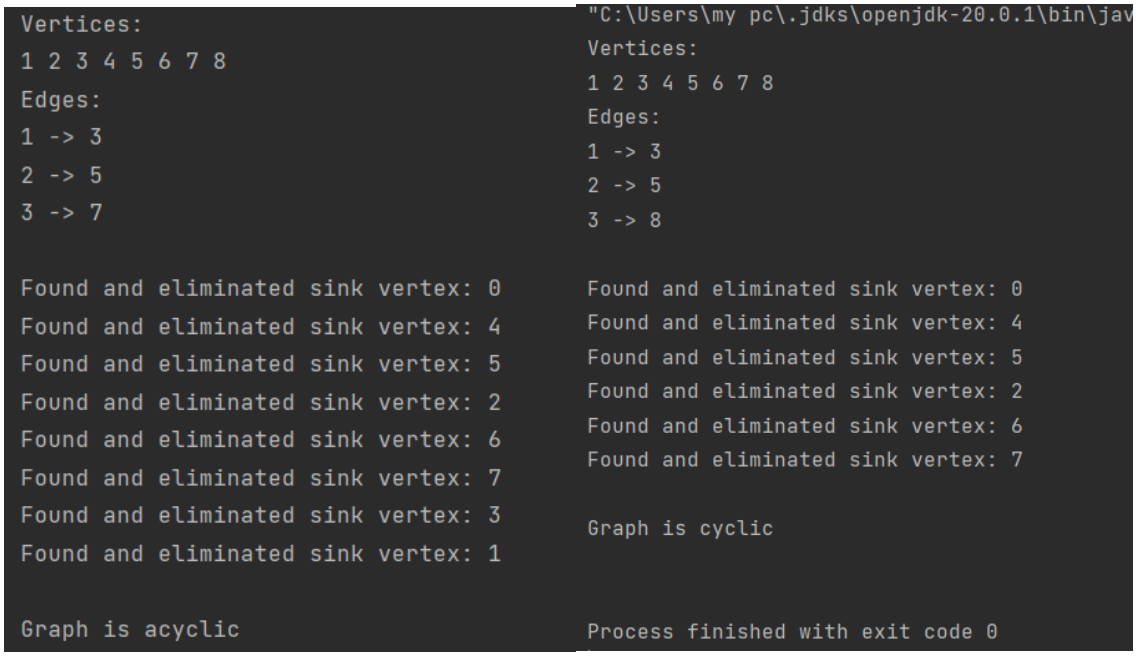
adjacency sets data structure.

The adjacency array set data structure depicts the relationships between graph vertices. It employs an array of sets, with each element representing a vertex and its associated set containing the vertex's neighbors. This structure enables efficient neighbor information storage and retrieval, as well as constant-time access to vertices and rapid set membership tests. It is well suited to sparse graphs because it optimizes memory usage by maintaining just information about existing links. This data format is commonly utilized in graph algorithms and applications that need efficient graph traversal or analysis. It strikes a compromise between memory economy and access speed, making it an adaptable option for a variety of graph-related activities.

Depth-First Search (DFS) algorithm

The Depth-First Search (DFS) algorithm is a graph traversal strategy that examines vertices and their edges as deeply as possible before returning to the starting point. DFS recursively examines undiscovered neighbors, marking them as visited along the way, beginning with a specified source vertex. This algorithm keeps track of visited vertices and the sequence of exploration using a stack or recursion. DFS is frequently used to tackle graph-related issues such as connecting components, identifying cycles, and exploring pathways in a graph. While DFS does not guarantee the shortest path or the best answer, it is a versatile algorithm that offers a versatile approach to graph traversal and analysis.

b) A run of algorithm on two small benchmark examples

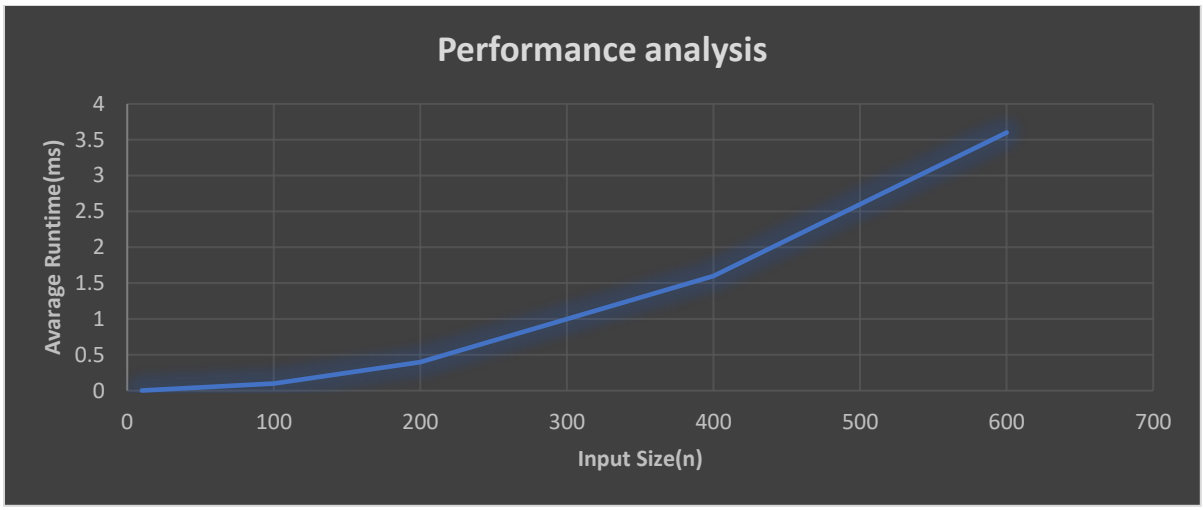


```
Vertices:
1 2 3 4 5 6 7 8
Edges:
1 -> 3
2 -> 5
3 -> 7

Found and eliminated sink vertex: 0
Found and eliminated sink vertex: 4
Found and eliminated sink vertex: 5
Found and eliminated sink vertex: 2
Found and eliminated sink vertex: 6
Found and eliminated sink vertex: 7
Found and eliminated sink vertex: 3
Found and eliminated sink vertex: 1

Graph is acyclic
```

```
"C:\Users\my pc\.jdks\openjdk-20.0.1\bin\jav
Vertices:
1 2 3 4 5 6 7 8
Edges:
1 -> 3
2 -> 5
3 -> 8

Found and eliminated sink vertex: 0
Found and eliminated sink vertex: 4
Found and eliminated sink vertex: 5
Found and eliminated sink vertex: 2
Found and eliminated sink vertex: 6
Found and eliminated sink vertex: 7

Graph is cyclic


Process finished with exit code 0
```

c) A performance analysis of algorithmic design and implementation

Input Size(n)	Average Runtime(ms)
10	0.002
100	0.100
200	0.400
400	1.600
600	3.600



Big O Notation

Big O notation is a mathematical notation that is used to express an algorithm's or function's efficiency and scalability. It allows you to see how an algorithm's runtime or space requirements change as the input size rises. Big O notation denotes the worst-case scenario or an upper constraint on the time or space complexity of the procedure. It defines the algorithm's growth rate in relation to the input size. The notation is O(f(n)), where f(n) is a function that characterizes the performance of the method.

Function	Name	Function	Name
O (1)	Constant	O(n2)	Quadratic
O (logn)	Logarithmic	O(n^3)	Cubic
O(log2n)	Log-square	O(n^4)	Quartic
O(√ n)	Root-n	O(2^n)	Exponential
O(n)	Linear	O(e^n)	Exponential
O(nlogn)	Linearithmic	O(n!)	n-Factorial