

Graph Traversal

[No Submission]

Instructions for students:

- You may use any language to complete the tasks (Java / Python).

NOTE:

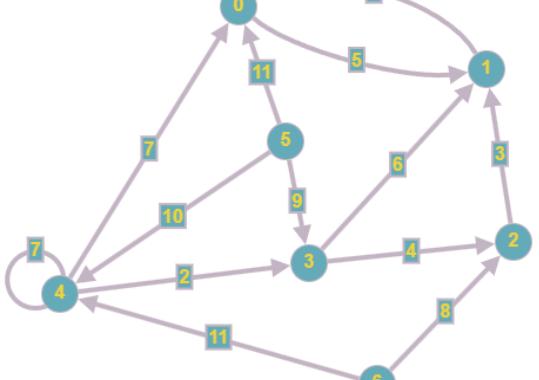
- **YOU CANNOT USE ANY BUILT-IN FUNCTION EXCEPT len IN PYTHON.** [negative indexing, append is prohibited]
- **YOU HAVE TO MENTION SIZE OF ARRAY WHILE INITIALIZATION**

Dear Students, you have been given instructions and driver code for the majority of the labs. For the last three labs of this semester, no driver code will be given. You will develop everything (necessary functions, class, driver code, etc.) in your preferred language (Java or Python).

[Design graphs according to your preference to solve the tasks]

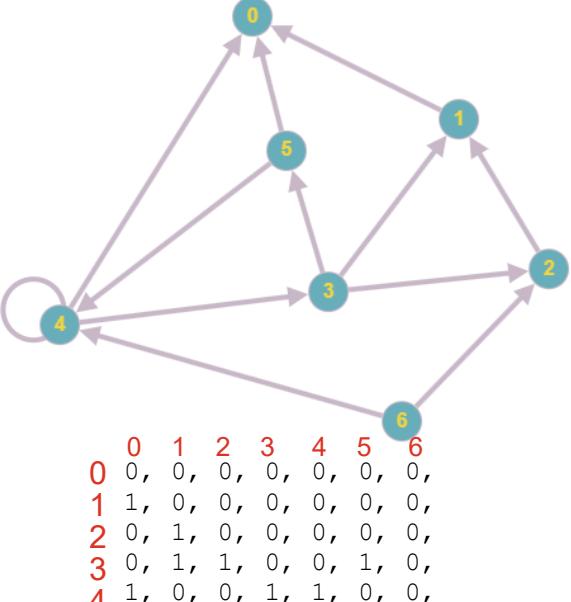
Task 1:

Implement the BFS algorithm to traverse a Directed Weighted graph represented by Adjacency Matrix.

SAMPLE GRAPH & Starting Vertex	SAMPLE OUTPUT
 <pre> 0 1 2 3 4 5 6 0 0, 5, 0, 0, 0, 0, 0 1 5, 0, 0, 0, 0, 0, 0 2 0, 3, 0, 0, 0, 0, 0 3 0, 6, 4, 0, 0, 0, 0 4 7, 0, 0, 2, 7, 0, 0 5 11, 0, 0, 9, 10, 0, 0 6 0, 0, 8, 0, 11, 0, 0 </pre> <p>Starting Vertex: 6</p>	<p>BFS Traversal: At (6) Vertex -> -----> (2) is a new neighbor -----> (4) is a new neighbor At (2) Vertex -> -----> (1) is a new neighbor At (4) Vertex -> -----> (0) is a new neighbor -----> (3) is a new neighbor -----> (4) already known At (1) Vertex -> -----> (0) already known At (0) Vertex -> -----> (1) already known At (3) Vertex -> -----> (1) already known -----> (2) already known</p> <p>BFS Traversal Order: 6 2 4 1 0 3</p>
<p>PseudoCode:</p> <pre> Initiate a boolean array of Length V //the array will keep track of the visited vertices //initially all values of the array will be False. Initiate a Queue enQueue the starting Vertex Loop until queue is empty current <- deQueue visited[current] <- true Loop through all neighbors of current if neighbor is unvisited enQueue each neighbor vertex visited[neighbor] <- true </pre>	
<p>Note: The output will change depending on the graph & the starting point.</p>	

Task 2:

Implement the DFS algorithm to traverse a Directed Unweighted graph represented by Adjacency Matrix.

SAMPLE GRAPH & Starting Vertex	SAMPLE OUTPUT
 <pre> 0 1 2 3 4 5 6 0 0, 0, 0, 0, 0, 0, 1 1, 0, 0, 0, 0, 0, 2 0, 1, 0, 0, 0, 0, 3 0, 1, 1, 0, 0, 1, 0, 4 1, 0, 0, 1, 1, 0, 0, 5 1, 0, 0, 0, 1, 0, 0, 6 0, 0, 1, 0, 1, 0, 0, </pre> <p>Starting Vertex: 6</p>	<p>DFS Traversal: Visiting (6) Visiting (4) Visiting (3) Visiting (5) Visiting (0) Visiting (2) Visiting (1)</p> <p>DFS Traversal Order: 6 4 3 5 0 2 1</p>
<p><u>PseudoCode:</u></p> <pre> Initiate a boolean array of Length V //the array will keep track of the visited vertices //initially all values of the array will be False. Initiate a Stack push the starting Vertex Loop till stack is empty current <- pop if current is unvisited visited[current] <- true Loop through all neighbors of current if neighbor is unvisited then push into stack </pre>	
<p><u>Note:</u> The output will change depending on the graph & the starting point.</p> <p><u>Did you know?</u> You can do DFS without using a Stack. In that case, you'd have to use a programming concept where backtracking exists. Can you guess what that is? Fun fact: even that concept utilizes something called Call Stack.</p>	