

# Graph Traversal

[No Submission]

## Instructions for students:

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

## NOTE:

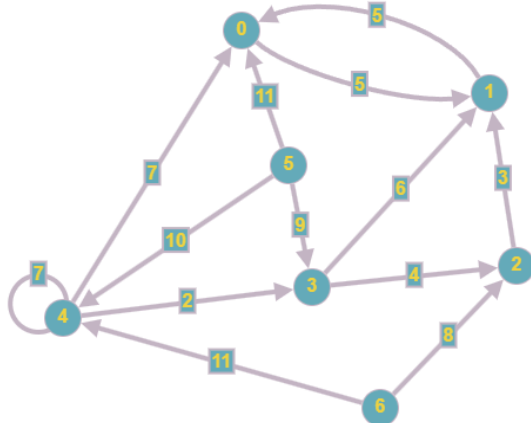
- **YOU CANNOT USE ANY BUILT-IN FUNCTION EXCEPT len IN PYTHON. [negative indexing, append is prohibited]**
- **YOU HAVE TO MENTION SIZE OF ARRAY WHILE INITIALIZATION**

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]

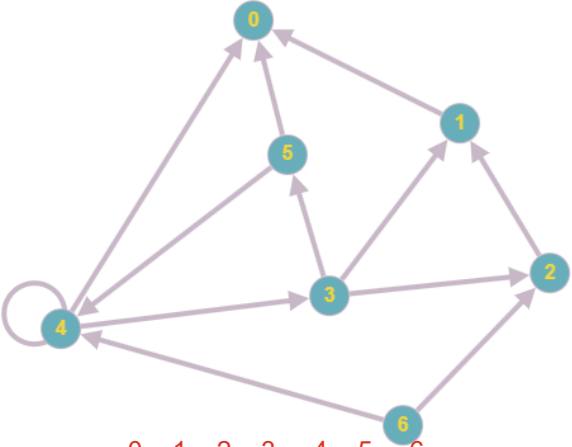
## Task 1:

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

SAMPLE GRAPH & Starting Vertex	SAMPLE OUTPUT																																																																
<div></div> <div><table><tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr><tr><th>0</th><td>0</td><td>5</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>1</th><td>5</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>2</th><td>0</td><td>3</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>3</th><td>0</td><td>6</td><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><th>4</th><td>7</td><td>0</td><td>0</td><td>2</td><td>7</td><td>0</td><td>0</td></tr><tr><th>5</th><td>11</td><td>0</td><td>0</td><td>9</td><td>10</td><td>0</td><td>0</td></tr><tr><th>6</th><td>0</td><td>0</td><td>8</td><td>0</td><td>11</td><td>0</td><td>0</td></tr></table></div> <div>Starting Vertex: 6</div>		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	<p>BFS Traversal:</p> <p>At (6) Vertex -&gt;</p> <p>-----&gt; (2) is a new neighbor</p> <p>-----&gt; (4) is a new neighbor</p> <p>At (2) Vertex -&gt;</p> <p>-----&gt; (1) is a new neighbor</p> <p>At (4) Vertex -&gt;</p> <p>-----&gt; (0) is a new neighbor</p> <p>-----&gt; (3) is a new neighbor</p> <p>-----&gt; (4) already known</p> <p>At (1) Vertex -&gt;</p> <p>-----&gt; (0) already known</p> <p>At (0) Vertex -&gt;</p> <p>-----&gt; (1) already known</p> <p>At (3) Vertex -&gt;</p> <p>-----&gt; (1) already known</p> <p>-----&gt; (2) already known</p> <p>BFS Traversal Order:</p> <p>6 2 4 1 0 3</p>
	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																																																										
<p><b>PseudoCode:</b></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 &lt;- dequeue   visited[ current ] &lt;- true   Loop through all neighbors of current     if neighbor is unvisited       enqueue each neighbor vertex       visited[ neighbor ] &lt;- true</pre>																																																																	
<p><b>Note:</b> The output will change depending on the graph &amp; 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 0 1 2 3 4 5 6 0 0, 0, 0, 0, 0, 0, 0, 0, 1 1, 0, 0, 0, 0, 0, 0, 0, 2 0, 1, 0, 0, 0, 0, 0, 0, 3 0, 1, 1, 0, 0, 0, 1, 0, 4 1, 0, 0, 1, 1, 0, 0, 0, 5 1, 0, 0, 0, 1, 0, 0, 0, 6 0, 0, 1, 0, 1, 0, 0, 0,</pre> <p>Starting Vertex: 6</p>	<p>DFS Traversal:</p> <p>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><b>PseudoCode:</b> 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</p> <p>Loop till stack is empty     current &lt;- pop     if current is unvisited         visited[ current ] &lt;- true         Loop through all neighbors of current             if neighbor is unvisited                 then push into stack</p>	
<p><b>Note:</b> The output will change depending on the graph &amp; the starting point.</p>	
<p><b>Did you know?</b> 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? <b>Fun fact:</b> even that concept utilizes something called Call Stack.</p>	