Name: ______________________________ **Date:** ______________ **Class:** _____________

Graph Theory:
Depth First Search (DFS) and Breadth First Search (BFS) Algorithms Instructions

DFS and BFS are common methods of graph traversal, which is the process of visiting every vertex of a graph. Stacks and queues are two additional concepts used in the DFS and BFS algorithms.

A stack is a type of data storage in which only the last element added to the stack can be retrieved. It is like a stack of plates where only the top plate can be taken from the stack. The three stacks operations are:

- ***Push*** – put an element on the stack
- ***Peek*** – look at the top element on the stack, but do not remove it
- ***Pop*** – take the top element off the stack

A queue is a type of data storage in which the elements are accessed in the order they were added. It is like a cafeteria line where the person at the front of the line is next. The two queues operations are:

- ***Enqueue*** – add an element to the end of the queue
- ***Dequeue*** – remove an element from the start of the queue

Considering a given node as the parent and connected nodes as children, DFS will visit the child vertices before visiting siblings using this algorithm:

Mark the starting node of the graph as visited and push it onto the stack
While *the stack is not empty*
    Peek *at top node on the stack*
    If *there is an unvisited child of that node*
        Mark the child as visited and push the child node onto the stack
    Else
        Pop *the top node off the stack*

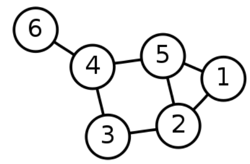
BFS will visit the sibling vertices before the child vertices using this algorithm:

Mark the starting node of the graph as visited and enqueue it into the queue
While *the queue is not empty*
    Dequeue *the next node from the queue to become the current node*
    While *there is an unvisited child of the current node*
        Mark the child as visited and enqueue the child node into the queue

Examples of the DFS and BFS algorithms are given next.

Example of the Depth First Search (DFS) Algorithm

Mark the starting node of the graph as visited and push it onto the stack
While *the stack is not empty*
    Peek *at top node on the stack*
    If *there is an unvisited child of that node*
        Mark the child as visited and push the child node onto the stack
    Else
        Pop the top node off the stack



Example using the graph to the right.

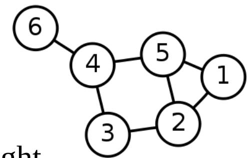
The stack push, peek and pop accesses the element on the right.

Action	Stack	Unvisited Nodes	Visited Nodes
Start with node 1	1	2, 3, 4, 5, 6	1
Peek at the stack Node 1 has unvisited child nodes 2 and 5	1	2, 3, 4, 5, 6	1
Mark node 2 visited	1, 2	3, 4, 5, 6	1, 2
Peek at the stack Node 2 has unvisited child nodes 3 and 5	1, 2	3, 4, 5, 6	1, 2
Mark node 3 visited	1, 2, 3	4, 5, 6	1, 2, 3
Peek at the stack Node 3 has unvisited child node 4	1, 2, 3	4, 5, 6	1, 2, 3
Mark node 4 visited	1, 2, 3, 4	5, 6	1, 2, 3, 4
Peek at the stack Node 4 has unvisited child node 5	1, 2, 3, 4	5, 6	1, 2, 3, 4
Mark node 5 visited	1, 2, 3, 4, 5	6	1, 2, 3, 4, 5
Peek at the stack Node 5 has no unvisited children	1, 2, 3, 4, 5	6	1, 2, 3, 4, 5
Pop node 5 off stack	1, 2, 3, 4	6	1, 2, 3, 4, 5
Peek at the stack Node 4 has unvisited child node 6	1, 2, 3, 4	6	1, 2, 3, 4, 5
Mark node 6 visited	1, 2, 3, 4, 6		1, 2, 3, 4, 5, 6

There are no more unvisited nodes so the nodes will be popped from the stack and the algorithm will terminate.

Example of the Breadth First Search (BFS) Algorithm

Mark the starting node of the graph as visited and enqueue it into the queue
***While** the queue is not empty*
    ***Dequeue** the next node from the queue to become the current node*
    ***While** there is an unvisited child of the current node*
        Mark the child as visited and enqueue the child node into the queue



Example using the graph to the right.

The queue operation enqueue adds to the left and dequeue removes from the right.

Action	Current Node	Queue	Unvisited Nodes	Visited Nodes
Start with node 1		1	2, 3, 4, 5, 6	1
Dequeue node 1	1		2, 3, 4, 5, 6	1
Node 1 has unvisited children nodes 2 and 5	1		2, 3, 4, 5, 6	1
Mark 2 as visited and enqueue into queue	1	2	3, 4, 5, 6	1, 2
Mark 5 as visited and enqueue into queue	1	5, 2	3, 4, 6	1, 2, 5
Node 1 has no more unvisited children, dequeue a new current node 2	2	5	3, 4, 6	1, 2, 5
Mark 3 as visited and enqueue into queue	2	3, 5	4, 6	1, 2, 5, 3
Node 2 has no more unvisited children, dequeue a new current node 5	5	3	4, 6	1, 2, 5, 3
Mark 4 as visited and enqueue into queue	5	4, 3	6	1, 2, 5, 3, 4
Node 5 has no more unvisited children, dequeue a new current node 3	3	4	6	1, 2, 5, 3, 4
Node 3 has no more unvisited children, dequeue a new current node 4	4		6	1, 2, 5, 3, 4
Mark 6 as visited and enqueue into queue	4	6		1, 2, 5, 3, 4, 6

There are no more unvisited nodes so the nodes will be dequeued from the queue and the algorithm will terminate.