Missouri University of Science & Technology | Department of Computer Science
Spring 2024 | **CS 2500: Algorithms (Sec: 102)**

Solutions to Homework 3: Graph Search

Instructor: *Sid Nadendla* | **Due:** *March 24, 2024*

Problem 1 Graph Data Structure *2 point*

Implement your own class called `Graph` which represents any unweighted, directed graph $G = (V, E)$ as an adjacency list. This `Graph` class should consist of the following subroutines:

(a) `AddVertex`$(self, v)$: Inserts a new vertex v into the `Graph` object. If v is already present in `Graph` object, raise an error.

(b) `AddEdge`$(self, u, v)$: Inserts a new edge (u, v) into the `Graph` object. If the edge (u, v) is already present in `Graph` object, raise an error.

(c) `DeleteVertex`$(self, v)$: Delete the vertex v and all its incident edges in the `Graph` object. If v is not present in `Graph` object, raise an error.

(d) `DeleteEdge`$(self, u, v)$: Delete the edge (u,v) from the `Graph` object. If the edge (u,v) is not present in `Graph` object, raise an error

(e) `AdjMatrix`$(self)$: Convert the adjacency list representation of the `Graph` object into a adjacency matrix form and return the matrix.

Test your result by first creating objects for K_5 and $K_{3,3}$ graphs. Then, convert them into K_4 and a cycle with 6 nodes (C_6) respectively, i.e. $K_5 \longrightarrow K_4$ using `DeleteVertex` subroutine and $K_{3,3} \longrightarrow C_6$ using `DeleteEdge` subroutine.

Solution: Since this is a programming exercise, the solution is not included. Students will receive feedback for their respective code submissions.

Problem 2 Breadth-First Search *4 points*

(a) Demonstrate breadth-first search (BFS) algorithm (with v_1 as the start node) on the unweighted, undirected graph shown in Figure 1. Clearly show how each node-attribute (including frontier) changes in each iteration in both the algorithms. (*1.5 points*)

(b) Implement `BFS`$(self, start\_vertex)$ subroutine in the `Graph` class you developed in Problem 1, and validate your implementation on the example graph shown in Figure 1 by comparing its output against your answer in Problem 2(b). (*2.5 points*)

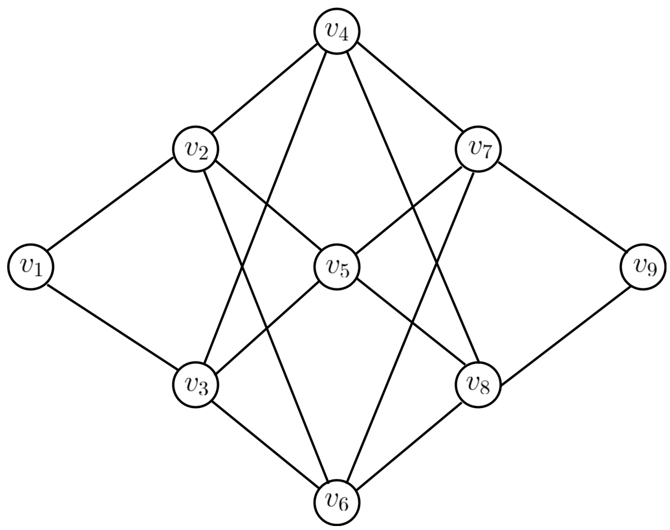
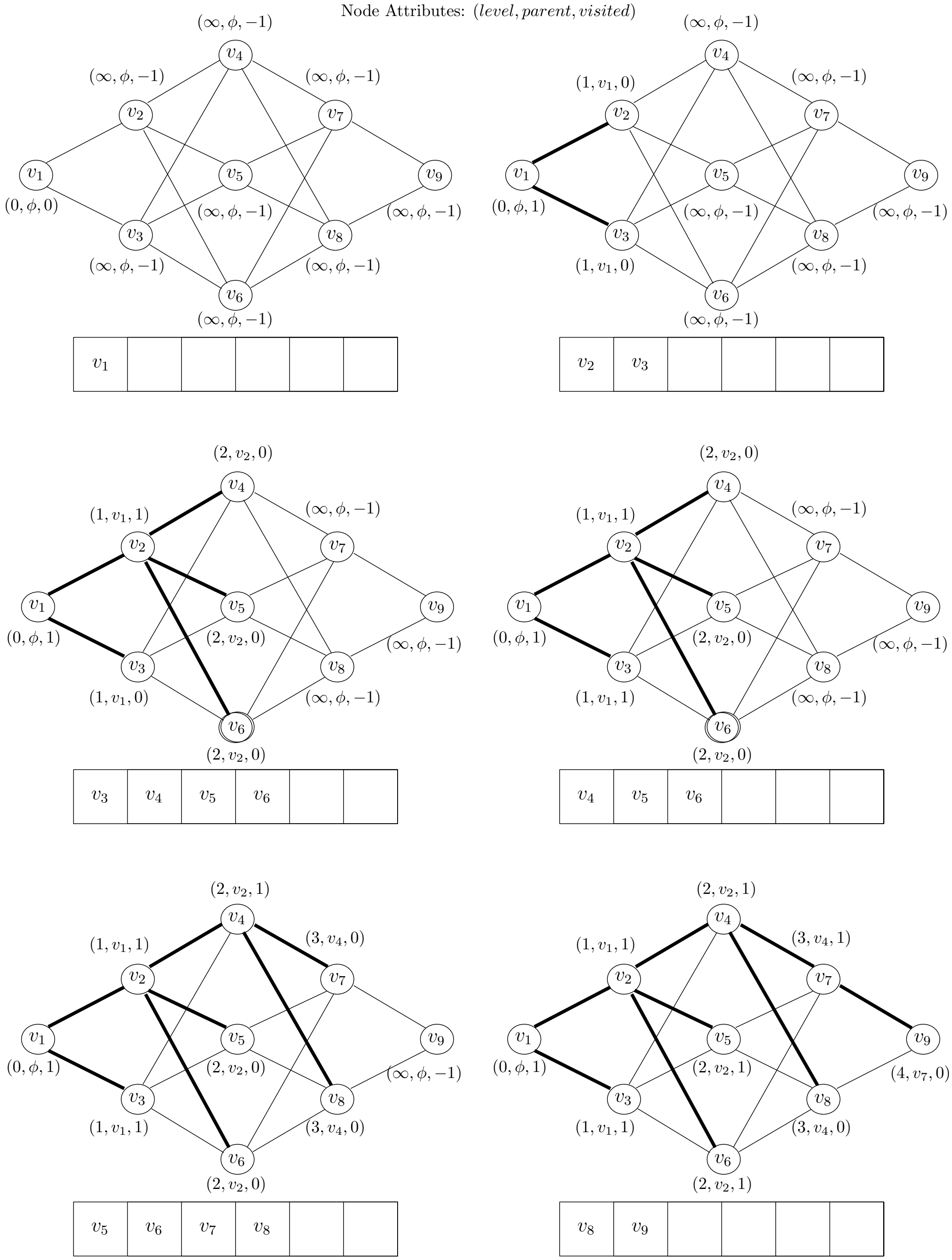


Figure 1: Example Graph for Graph Search

Solution 2a: Following is the step-by-step workflow of BFS algorithm (implemented using FIFO queue) in the context of the given example:

Frontier implemented as a first-in first-out queue.

Node Attributes: $(level, parent, visited)$



After three iterations, upon dequeuing v_7, we have another update.
Afterwards, there is no update until the queue is empty.

Solution 2b: Since this is a programming exercise, the solution is not included. Students will receive feedback for their respective code submissions.

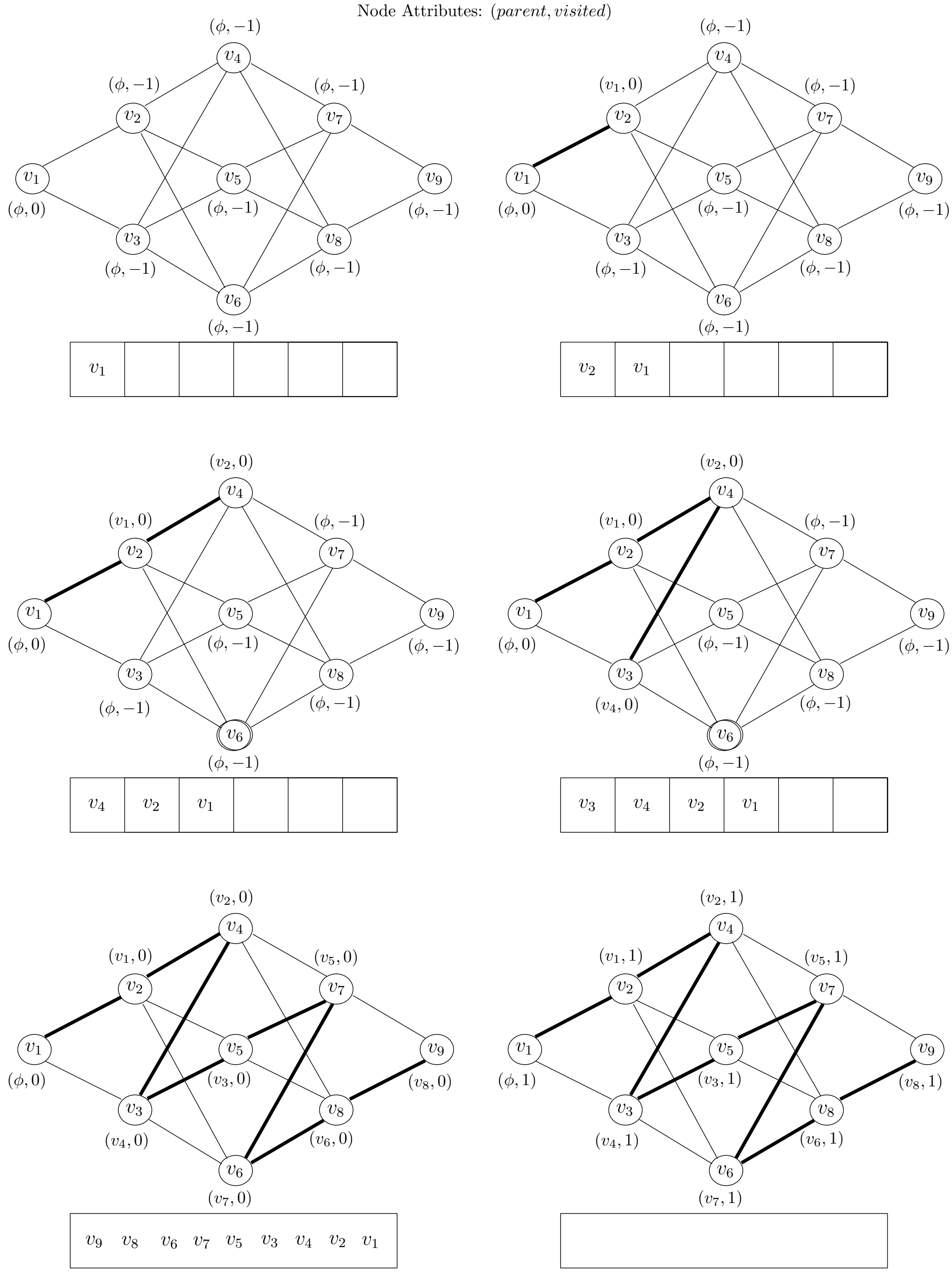
Problem 3 Depth-First Search ***4 points***

(a) Demonstrate depth-first search (DFS) algorithm (with v_1 as the start node) on the unweighted, undirected graph shown in Figure 1. Clearly show how each node-attribute (including frontier) changes in each iteration in both the algorithms. (*1.5 points*)

(b) Implement `DFS`$(self, start\_vertex)$ subroutine in the `Graph` class you developed in Problem 1, and validate your implementation on the same example graph shown in Figure 1 by comparing its output against your answer in Problem 3(b). (*2.5 points*)

Solution 3a: Following is the step-by-step workflow of DFS (implemented using stacks) in the context of the given example:

Frontier implemented as a stack.
Node Attributes: $(parent, visited)$



In the next five iterations, push the nodes v_5, v_7, v_6, v_8 and v_9 into the *frontier* stack.

As the algorithm detects a potential cycle at each node, the stack is popped until it is empty.

Solution 3b: Since this is a programming exercise, the solution is not included. Students will receive feedback for their respective code submissions.