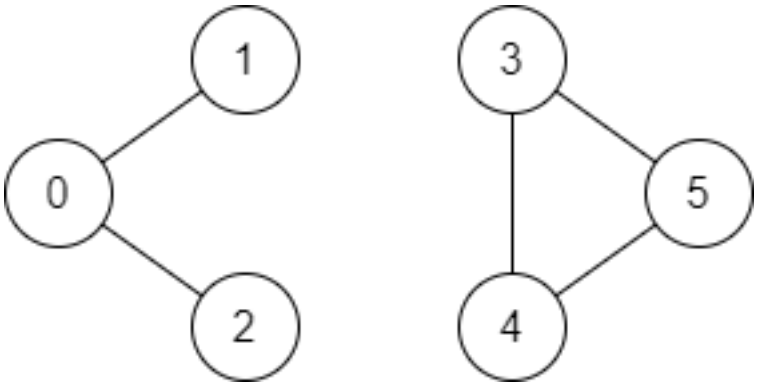
Project 5: Find if Path Exists in Graph
CS 3343: Analysis of Algorithms Summer 2024
Instructor: Dr. Mohammad Imran Chowdhury
Total: 100 Points
Due: 08/06/2024 11:59 PM

In this project, I invite you to solve the ``Find if Path Exists in Graph" using the various aspects of algorithmic thinking such as the Breadth First Search Approach, and Python programming discussed in class.

Problem Description:

Given an unweighted bi-directional graph represented by a **dict_graph = {node: [neighbors]}** and two vertices source and destination. Your task is to check if there exists a path from source to destination.



For instance, for the given example above the **dict_graph** would be as follows:

```
dict_graph = {
        0: [1, 2],
        1: [0],
        2: [0],
        3: [4, 5],
        4: [3, 5],
        5: [3, 4]
       }
```

Now, your task is to check if there exists a path from the source to the destination. If the path exists then print the path otherwise print the Error message as per the project's writeup sample input/output format.

Breadth First Search Approach

The given problem can be solved using Depth-First-Search (DFS) or Breadth-First-Search (BFS) algorithms. In this solution, we will use BFS to solve the problem.

We need to traverse the graph starting from the source node and mark all the nodes that can be reached from the source node. We can traverse the graph using a queue as it will follow a level-by-level traversal.

Algorithm:

1. Create a queue and push the source node into it.
   queue = [[start]]
2. Create a visited array in order to mark the visited nodes.
   visited = set()
3. While the queue is not empty:
   while queue:
   - Retrieve the head of the queue and mark it as visited.
     path = queue.pop(0)
   - Check if the retrieved node is the **destination_node**. If true, return true as there exists a path from source to destination. And print the path.
     vertex = path[-1]
     if vertex == destination_node:
     return True, "Path found!", path
   - Otherwise, traverse all the adjacent nodes of the retrieved node that are not visited and push them into the queue.
     elif vertex **not** in visited:
     for neighbour **in** dict_graph.get(vertex, []):
     new_path = list(path)
     new_path.append(neighbour)
     queue.append(new_path)
     visited.add(vertex)
4. If we can't find the destination node after traversing all nodes, return false. Print the Error message.
   return False, 'Path Does Not Exist!!!'

Source code:

You need to start work with the following source code also available through the project folder found on the CANVAS course website i.e., the starter file find_if_path_exists_in_graph.py, and complete the TO-DO portion to solve the ``Find if Path Exists in Graph" using the Breadth First Search Approach, and Python programming discussed in class.

```
n = int(input("Enter total no. of graph nodes: "))
dict_graph = {}
for i in range(n):
    key = int(input("Enter Node: "))
```

```
    values = [int(neighbor) for neighbor in input(f"Enter Node
{key}'s Neighbor: ").split()]
    dict_graph[key] = values

start = int(input("Enter start node: "))
end = int(input("Enter end node: "))

print("The Graph: ", dict_graph)
print("Start Node: ", start)
print("End Node: ", end)




def find_if_path_exists_in_graph(dict_graph, start, end):
    pass
  # TO-DO #
  # Your code goes here

print(find_if_path_exists_in_graph(dict_graph, start, end))
```

Sample I/O:

Run 1:

Enter total no. of graph nodes: 6
Enter Node: 0
Enter Node 0's Neighbor: 1
Enter Node: 1
Enter Node 1's Neighbor: 0 2 3
Enter Node: 2
Enter Node 2's Neighbor: 1 3 5
Enter Node: 3
Enter Node 3's Neighbor: 1 2 4
Enter Node: 4
Enter Node 4's Neighbor: 3 5
Enter Node: 5
Enter Node 5's Neighbor: 2 4
Enter start node: 0
Enter end node: 5
The Graph: {0: [1], 1: [0, 2, 3], 2: [1, 3, 5], 3: [1, 2, 4], 4: [3, 5], 5: [2, 4]}
Start Node: 0
End Node: 5
(True, 'Path found!', [0, 1, 2, 5])

Run 2:

```
Enter total no. of graph nodes: 6
Enter Node: 0
Enter Node 0's Neighbor: 1 2
Enter Node: 1
Enter Node 1's Neighbor: 0
Enter Node: 2
Enter Node 2's Neighbor: 0
Enter Node: 3
Enter Node 3's Neighbor: 4 5
Enter Node: 4
Enter Node 4's Neighbor: 3 5
Enter Node: 5
Enter Node 5's Neighbor: 3 4
Enter start node: 0
Enter end node: 4
The Graph:  {0: [1, 2], 1: [0], 2: [0], 3: [4, 5], 4: [3, 5], 5: [3, 4]}
Start Node:  0
End Node:  4
(False, 'Path Does Not Exist!!!')
```

Run 3:

```
Enter total no. of graph nodes: 6
Enter Node: 0
Enter Node 0's Neighbor: 1 2
Enter Node: 1
Enter Node 1's Neighbor: 0 2 4
Enter Node: 2
Enter Node 2's Neighbor: 0 1 3 4
Enter Node: 3
Enter Node 3's Neighbor: 2 4
Enter Node: 4
Enter Node 4's Neighbor: 1 2 3 5
Enter Node: 5
Enter Node 5's Neighbor: 4
Enter start node: 0
Enter end node: 5
The Graph:  {0: [1, 2], 1: [0, 2, 4], 2: [0, 1, 3, 4], 3: [2, 4], 4: [1, 2, 3, 5], 5: [4]}
Start Node:  0
End Node:  5
(True, 'Path found!', [0, 1, 4, 5])
```

The submission grading rubric is as follows (points out of 100 total):

Project element	**Points**
Code readability such as the usage of comments in code	10
Proper coding implementation	60
Screenshots of the program output	10
Program's sample Input/Output format matching as per project writeup	20

Submission Instructions: Create a compressed file (.zip or .tar.gz files are accepted) with all your source files such as .py files. Within this compressed .zip folder, you should provide some screenshots of your running program's output as proof. Generally speaking, to complete the `` Find if Path Exists in Graph", you just need one .py file. But it's better to submit everything as a compressed file. Submit the compressed file to CANVAS.

Late submission policy: As described in the syllabus, any late submission will the penalized with 10% off after each 24 hours late. For example, an assignment worth 100 points turned in 2 days late will receive a 20-point penalty. Assignments turned in 5 or more days after the due date will receive a grade of 0.