Namal University, Mianwali
Department of Electrical Engineering

Course Title: EE-253 Data Structures and Algorithms

LAB MANUAL

Lab 11
Depth First Search and Breadth First Search in Graphs

In previous lab, we have studied the concept of graphs in python. We have also studied how to implement a graph in python. Further, students learn how to add and remove the vertices and edges. Further students developed method to search for a node with particular key in the graph. Then students, developed adjacency matrix structure for graph.

1. Lab Objectives

The objective of this lab is to introduce graph traversal techniques, Depth First Search and Breadth First Search for a graph. Student will also do a comparison of both techniques. Students will also examine the running time of each technique. Students will be provided with examples, followed by performing lab tasks.

2. Lab Outcomes

- CLO 1: Recognize the usage of fundamental Data structure using Python Programming Language.
- CLO 2:
- CLO 3: Demonstrate solution to real life problems using appropriate data structures.

3. Equipment

- Software
    - IDLE (Python 3.11)

4. Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as 'Example') are for practice purposes only while others (marked as 'Task') have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
    a. Code (copy and pasted, NOT a screenshot)
    b. Output figure (as instructed in 3)
    c. Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

5. Background

Graph Traversal

Formally, a ***traversal*** is a systematic procedure for exploring a graph by examining all of its vertices and edges. A traversal is efficient if it visits all the vertices and edges in time proportional to their number, that is, in linear time. Graph traversal algorithms are key to answering many fundamental questions about graphs involving the notion of ***reachability***, that is, in determining how to travel from one vertex to another while following paths of a graph.

1. Depth First Search

Depth-first search is useful for testing a number of properties of graphs, including whether there is a path from one vertex to another and whether or not a graph is connected.

Algorithm of DFS is given as,

```
Algorithm DFS(G,u):          {We assume u has already been marked as visited}
  Input: A graph G and a vertex u of G
  Output: A collection of vertices reachable from u, with their discovery edges
  for each outgoing edge e = (u,v) of u do
    if vertex v has not been visited then
      Mark vertex v as visited (via edge e).
      Recursively call DFS(G,v).
```

2. Breadth First Search

Another algorithm for traversing a connected component of a graph, known as a ***breadth-first search*** (BFS). The BFS algorithm is more akin to sending out, in all directions, many explorers who collectively traverse a graph in coordinated fashion.

Algorithm of BFS is given as,

```
def BFS(g, s, discovered):
  """Perform BFS of the undiscovered portion of Graph g starting at Vertex s.

  discovered is a dictionary mapping each vertex to the edge that was used to
  discover it during the BFS (s should be mapped to None prior to the call).
  Newly discovered vertices will be added to the dictionary as a result.
  """
  level = [s]                          # first level includes only s
  while len(level) > 0:
    next_level = [ ]                   # prepare to gather newly found vertices
    for u in level:
      for e in g.incident_edges(u):    # for every outgoing edge from u
        v = e.opposite(u)
        if v not in discovered:        # v is an unvisited vertex
          discovered[v] = e            # e is the tree edge that discovered v
          next_level.append(v)         # v will be further considered in next pass
    level = next_level                 # relabel 'next' level to become current
```

6. Lab Tasks:

Task 1: Using the code developed in previous lab, add another member function of DFS based traversal in the existing class of graph. Apply the DFS on graph provided in Fig. 1.

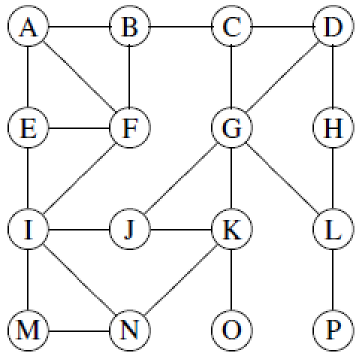

Figure 1 An undirected graph starting from vertex A

Task 2: Develop a method for BFS based traversal. Apply the BFS on graph provided in Fig. 1.

Task 3: Perform a comparison for edge count in BFS and DFS, and explain which one is better. Computing the running time of both traversal schemes in the form of O(n).

Task 4: Imagine you have a maze represented as a grid, where each cell can be either a wall (W) or a passage (1). You want to find a path from a start cell to an end cell in the maze. Find the path either using DFS or BFS traversal method.

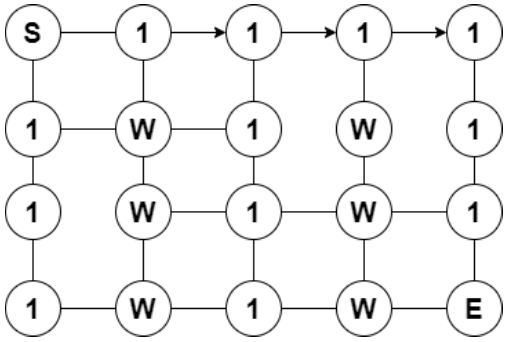

Figure 2 A maze based graph

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-2**	**Marginal 3-5**	**Satisfactory 6-8**	**Exemplary 9-10**	**Allocated Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested by others. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend for the instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around. More than two incidents of talking non-lab related stuff in laband/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non-lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff in lab Voice level exceeding theappropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceedingthe appropriate level, use of cell phones and involvementin any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Tookcare of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	