Assignment 01
Graph Traversal

Please submit your solutions in PDF format. The PDF must be typed, NOT handwritten. Solution for each problem must start on a new page. The solutions should be concise; complicated and half-witted solutions might receive low marks, even when they are correct. Solutions should be submitted on the course website.

Problem 1: Collaborators [2 points]

List the name of the collaborators for this assignment. If you did not collaborate with anyone, write "None" (without the quotes).

Solution:

Tasneea Hossain

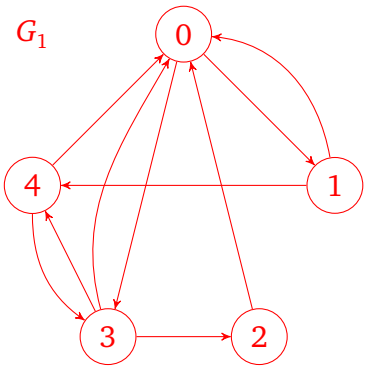
Problem 2: Welcome to the World of Graphs [17 points]

(a) [4 points] **Representation to Graph**
Using the following graph representations, draw the directed graph associated with them. Here, G_1 is represented by an adjacency matrix.

$G_1 =$

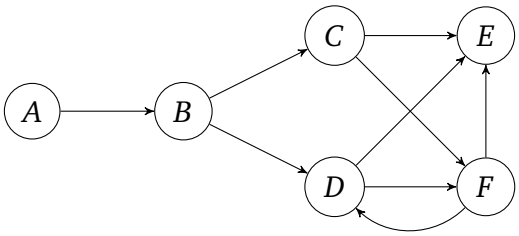
	0	1	2	3	4
0	0	1	0	1	0
1	1	0	0	0	1
2	1	0	0	0	0
3	1	0	1	0	1
4	1	0	0	1	0

Solution:



Rubric: -1 point for each incorrect edge; minimum 0 points

(b) [3 points] **Graph to Representation**
Write down the adjacency list representation of the graph shown below. The vertices should be listed in lexicographical order.



Solution:

$adjL =$

A	B
B	C D
C	E F
D	E F
E	
F	D E

(c) [6 points] **Traversal**
Run both BFS and DFS on the graph from part (b). Take A as your source node. While performing each search, each outgoing neighbors will be visited in lexicographical order. For each search, list the vertices in the order in which they were first visited.

Solution:

BFS: A, B, C, D, E, F
DFS: A, B, C, E, F, D

(d) [4 points] **DAG**

Removing one single edge from the graph in part (b) can make it a DAG. Find out all such possible edges with this property, and for each, state the topological sort order of the resulting DAG.

Solution:

There is only one cycle in the graph between vertices D and F, so we can remove either the edge (D, F) or (F, D) to make the graph acyclic. Removing edge (D, F) results in a DAG with a unique topological ordering (A, B, C, F, D, E). Removing edge (F, D) results in a DAG with two topological ordering: (A, B, C, D, F, E) and (A, B, D, C, F, E).

Problem 3: Diameter of a Tree [5 points]

The diameter $d(u)$ of a node $u \in V$ denotes the distance to the farthest node $v \in V$. The diameter $d(G)$ of an undirected graph $G = (V, E)$ is the largest diameter of any nodes. Consider that, G is a tree, that means any two vertices are connected by exactly one path. Describe a $O(V + E)$-time algorithm to compute the diameter of a given connected undirected graph.

Solution:

Start BFS from one node u, and calculate the longest distance from u to v. Call the node v, and then start another BFS from v, and calculate the longest distance from v to another node. The second distance will be the diameter of the tree.
Each BFS takes O(V+E) time, making the total runtime: O(V+E). analysis of that algorithm.

Problem 4: Properties of a Graph [16 points]

(a) [8 points] **Pickle Mick**

Mick has gotten into a pickle. His neighbors do not get along with each other. You see, Mick's grandson, Rorty wants to invite all their neighbors to a party, but that would cause chaos. That's why Mick advised Rorty to throw two separate parties: one on Saturday and another on Sunday. Rorty wants to invite all his neighbors to at most one of the parties. He is worried that if two neighbors who don't like each other are invited to the same party, there will be too much drama. To solve the problem, Mick has created a list of conflicting pairs of neighbors. Given the list, you need to come up with a linear time algorithm to determine whether Mick and Rorty can invite all their neighbors to the parties so that two conflicting neighbors do not come to the same party.

Solution:

Construct a graph on Rorty's neighbors with an edge between neighbor u and neighbor v if they are a conflicting pair. We must determine whether this graph is bipertite: every vertex can be assigned one of two colors, either red or blue, such that no edge connects two vertices of the same color. Pick an arbitrary vertex $v \in V$ and color it blue. Run a BFS from v and color red all vertices reachable from v in one step. Continue the BFS, coloring nodes in even levels blue and odd levels red. Then loop through the edges of the graph. If there exists an edge connecting two vertices a and b having the same color, the shortest paths from v to r and b, together with the edge, form a cycle with odd length. This cycle is not two-colorable, so neighbors in this cycle cannot be assigned to parties without some conflicting pair being invited to the same party. Otherwise, if no edge between same colored vertices exists, then invite red friends on Saturday and blue friends no Sunday, no drama!

BFS takes $O(V+E)$ time, then traversing through each edge takes $O(E)$ time, making the total runtime: $O(V+E)$.

(b) [8 points] **Mucced**

BaceFook is a new social network. Every pair of users in BaceFook is either friends with each other or not. Two users u_1 and u_2 are considered mutual if there exists a sequence of users starting from u_1 and ending with u_2 where each adjacent pair of users in the sequence are friends. Zark Muckerberg has recently joined BaceFook and is not friends with anyone. Given a list of all pairs of users that are friends on BaceFook, describe an efficient algorithm to determine the minimum number of users with whom Zark needs to become friends with to be mutual with every user in BaceFook.

Solution:

Let n be the number of input pairs (so the number of users are also atmost n). The BaceFook network can be represented by an undirected graph on the users, with an undirected edge between users u and v if they are friends. This graph may have one or more connected components. For Zark to be mutual with every user requires that Zark becomes friends with at least one user from every connected component, so the solution to this problem is equivalent to counting the number of connected components in a graph. To count the number of connected components, repeatedly choose an arbitrary unvisited vertex and run a graph search (either BFS or DFS) to mark all vertices reachable from that vertex. Then return the number of distinct searches that were made.

Because each user is contained in at most one search, and each search runs in linear time with respect to the connected component it explores, this full graph search takes $O(n)$ time.