CSE 221	**Assignment 2**
Summer 2021	**Deadline:** Mon, August 16 at 11:59 pm

Guidelines for Student Tutors: Read the answers first, understand the demands of the questions. Penalize as you feel necessary according to the **Solution** block and **We are expecting** block! If you face any issues, dont hesitate to shoot a DM to [RRD] Rayhan Rashed!

Also, in the submissions folder, dont just look for the copies of your section based on section name values. Some students put "sec", some put "0" infront. Please check for those dopiness.

Finally, check for copies!

Guidelines and Submission: The submission link for this assignment is here: **Google Form**.

Please make sure you are accessing the Submission Link with your BracU email/account. You should cite all sources you used outside of the course material. Make sure to look at the "**We are expecting**" blocks below each problem to see what we will be grading for in each problem! Dont copy from our peers, or anywhere else. Slightest of dishonesty will bring you 00 marks in the whole!

Problems

---

1. **(10 pt.) [BFS and DFS A]**
   Give one example of a *connected undirected* graph on four vertices, A, B, C, and D, so that both DFS and BFS search *discover* the vertices in the **same** order when starting at vertex A. Then give one example of a *connected undirected* graph on four vertices, A, B, C, and D where BFS and DFS discover the vertices in a **different** order when started at A. Assume that both DFS and BFS iterate over neighbors in *alphabetical order*.

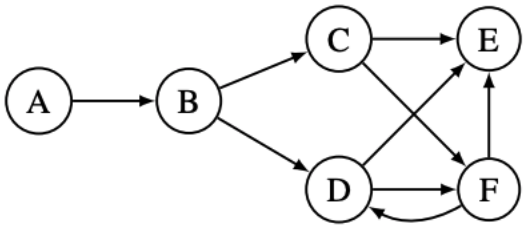
   Above, *discover* means the time that the algorithm first reaches the vertex.

   [**We are expecting:** A drawing of your two graphs **and** an ordered list of vertices discovered by BFS and DFS for each of them.]

   Solution: They will draw two graphs. For graph 1, check whether running both BFS and DFS return the same order of vertices. If yes, give 5 marks. If no, give 0! For graph 2, just do the opposite. Check whether alphabetically running the DFS & BFS yield different results, if yes, give 5, else 0

2. **(5 pt.) [BFS & DFS Tree]** Run both BFS and DFS on the graph below, starting from node A. While performing each search, visit the outgoing neighbors of a vertex

in alphabetical order. For each search, draw the resulting tree and list vertices in the order in which they were first visited.

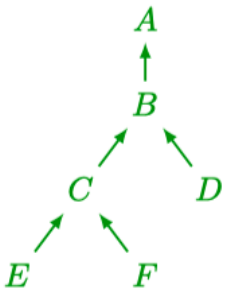


[**We are expecting:** For each of BFS and DFS: a graph on which the algorithm is run, a tree, and a list containing the order of vertices]

Solution: First check, if they drew the graph, and appropriately run bfs/dfs or not. And then, for the tree and the order check below for the correct answers:

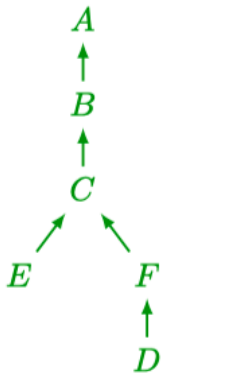
BFS:

[A, B, C, D, E, F]



DFS:

[A, B, C, E, F, D]



3. **(5 pt.) [Robust Fuel Distribution]** You are given a fuel system to analyze its robustness. Here we will consider the robustness of the fuel distribution system under a few simplifications:

   - Each edge in the graph is a pipe.
   - Each edge is directed (fuel flow is not bidirectional).
   - Each node in the graph is a combination refinery/distribution terminal (nodes both produce and consume fuel products, so they can have both incoming and outgoing edges).

   Assume the system has n nodes and m edges. Design an algorithm that verifies whether there is a way to transport fuel from any node in the graph to any other node in the graph. Your algorithm should run in $O(n + m)$.

[**We are expecting:** A description of the algorithm, a brief explanation of why the algorithm works and an informal justification of the algorithm's running time.]

Solution: In this question, we are basically asking to check, "**whether a given graph is strongly connected or not.**" The students are asked to present the algorithm/idea, then clearly justify why their algorithm/idea work. Pay attention to their reasoning. For instance, if they say, checking for Strongly Connectedness is enough, look for the justifications they wrote. Penalize as you feel necessary, if you think they could not explain enough. Also, if they do not analyze their running time, penalize.

4. **(10 pt.) [DPDC]** The Dhaka Power Distribution Company [DPDC] network contains n power plants and n^3 buildings, pairs of which may be directly connected to each other via bidirectional wires. Every building is powered by either: having a wire directly to a power plant, or having a wire to another building that is recursively powered. Note that the network has the property that no building could be powered by more than one power plant. DPDC has secured funds to install an emergency generator in one of the power plants, which would provide backup power to all buildings powered by it, were the power plant to fail. Given a list L of all the wires in the network, describe an $O(n^6)$-time algorithm to determine the power plant where DPDC should install the generator that would provide backup power to the most buildings upon plant failure.

   [**We are expecting:** A description of the algorithm, a brief explanation of why the algorithm works and an informal justification of the algorithm's running time.]

   Solution: Construct undirected graph G whose vertices are the buildings and power plants, and whose edges are wires between them. There are $n^3 + n$ vertices and $|L| = O(n^6)$, under the assumption that at most one wire connects any pair of vertices.

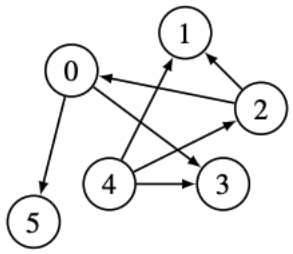
   Since every building is powered and no building is powered by the same power plant, there is exactly one power plant in each connected component of G. So the number of buildings that would benefit by having a generator installed at plant p would be the size of the connected component containing p minus one (since the plant is not a building). So we want to install the generator at the plant that powers the largest connected component.

   So run either **BFS** (Yes, you can also check for connectedness using BFS) or **DFS** to count the size of each connected component in the graph, and return the plant contained in the largest one. Constructing G takes $O(n^6)$ time, running BFS or DFS also takes $O(n^6)$ time, and finding the plant in a connected component can be done in $O(n)$ time, so this algorithm runs in $O(n^6)$ time.

5. **(10 pt.) [Topological Exercise]** Please answer the following questions about the unweighted directed graph G below:

   (a) **(5 pt.)** State a topological ordering of G. Then state and justify the number of distinct topological orderings of G.

   Solution: (a) Vertices 4 and 2 must be the first and second vertex in any topo-



logical order, since there is a directed path from 4 through 2 to every other vertex in the graph. For the remaining four vertices, the ordering of 0, 3, and 5 are independent from 1. Vertices 0, 3, and 5 have two possible orderings: (0, 3, 5) and (0, 5, 3), while 1 can be placed in one of four positions relative to either ordering. Thus there are 8 distinct topological orders of G. Specifically:

$$(4,2,1,0,3,5) \quad (4,2,0,1,3,5) \quad (4,2,0,3,1,5) \quad (4,2,0,3,5,1)$$
$$(4,2,1,0,5,3) \quad (4,2,0,1,5,3) \quad (4,2,0,5,1,3) \quad (4,2,0,5,3,1)$$

(b) **(5 pt.)** State a single directed edge that could be added to *G* to construct a *simple graph* with no topological ordering. Then state and justify the number of distinct single edges that could be added to *G* to construct a *simple graph* with no topological ordering

Solution: (b) A topological ordering will not exist, if and only if the resulting graph contains a cycle. All vertices are reachable from 4, so we can add any edge from vertices 0, 1, 2, 3, or 5 to 4 to create a cycle. All vertices except 4 are reachable from 2, so we can add any edge from vertices 0, 1, 3, or 5 to create a cycle. Vertices 3 and 5 are reachable from 0, so we can add either edge to 0. No vertices are reachable from 1, 3, or 5, so no edge can be added to them to construct a cycle. **Thus there are eleven such directed edges**, specifically:

$$\{(0,4),(1,4),(2,4),(3,4),(5,4)(0,2),(1,2),(3,2),(5,2),(3,0),(5,0)\}.$$