Instructions: Please note that handwritten assignments **will not be graded**. Use the provided LaTeX template to complete your homework. Please do not alter the order or spacing of questions (keep each question on its own page). When you submit to Gradescope, you must mark which page(s) correspond to each question. **You may not receive credit for unmarked questions**.
When including graphical figures, we encourage the use of tools such as graphviz or packages like tikz for simple and complex figures. However, these may be handwritten only if they are neat and legible (as defined by the grader).

List any collaborators (besides TAs or professors) here:

1. (35 points) [Maze, ★★★★★] Describe the graph model for Spacewreck. How will you encode the complexities and rules of the game into a graph? You must be able to run an **unmodified** BFS to find the shortest sequence of moves for either Captain Rocket or Lieutenant Lucky to reach the Goal. In order for BFS to work, your model must have exactly one start vertex and exactly one finish vertex.

   Graph Model

   Vertex - possible game states (room location of each person)
   Edge - possible moves to new game states

   Each iteration of BFS will add all possible moves from the current game state to the queue.
   BFS will terminate when either Captain Rocket or Lieutenant Lucky reaches the goal.
   BFS will return the path from the start vertex to the goal vertex.

   This handles things like loops, because BFS will not add a game state to the queue if it has already been visited allowing it to exit the failed path and continue searching

2. (5 points) [W3, ⋆] For the following questions, select whether the statement is true or false, and write a *brief* explanation of your reasoning.

   (a) Topological Sort results in a unique ordering

   □ True ■ False

   While a topological sort guarantees precedence is followed, it is not guaranteed to be unique. For example, the graph below has two possible topological sorts: $1, 2, 3, 4$ and $1, 3, 2, 4$.

$$\begin{tikzcd}
1 \arrow[r] \arrow[d] & 2 \arrow[d] \\
3 \arrow[r] & 4
\end{tikzcd}$$

   (b) It is impossible to find a topological ordering on a cyclic graph.

   ■ True □ False

   A topological ordering is only possible on a DAG. A cyclic graph is not a DAG, so it is impossible to find a topological ordering on a cyclic graph.

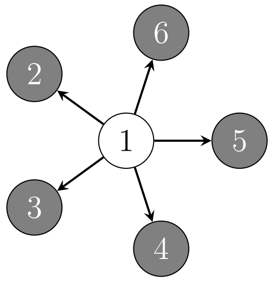
3. (20 points) [W2/3, ⋆⋆] In BFS and DFS, an undiscovered node is marked *discovered* when it is first encountered, and marked processed when it has been completely searched. At any given moment, several nodes might be simultaneously in the *discovered* state.

   (a) (7 points) Describe a graph with n vertices such that $n - 1$ vertices are simultaneously in the discovered state at some point during a **BFS** from a starting vertex v of your choosing.

   A star graph where v is the center vertex and each branch had the same depth. For example, the graph below is a star of 6 vertices, where 5 vertices are simultaneously in the discovered state at some point during a **BFS** from a starting vertex 1.

   (b) (3 points) Draw a graph with $n = 6$ vertices that matches the description in part (a). Clearly indicate which vertex is the starting vertex.

   Gray = discovered, white = undiscovered, black = processed.

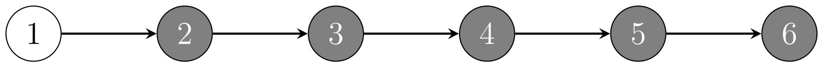


   (c) (7 points) Describe a graph with n vertices such that $n - 1$ vertices are simultaneously in the discovered state at some point during a **DFS** from a starting vertex v of your choosing.

   A linear graph in which n vertices form a list, where $n \geq 2$. For example, the graph below is a line of 6 vertices, where 5 vertices are simultaneously in the discovered state at some point during a **BFS** from a starting vertex 1.

   (d) (3 points) Draw a graph with $n = 6$ vertices that matches the description in part (b). Clearly indicate which vertex is the starting vertex.

   Gray = discovered, white = undiscovered, black = processed.



4. (10 points) [W2/3, ★★] Your friend Alice has recently returned from a trip to a wonderful land. While describing her adventures, she tells you about a curious hedge maze she encountered. "The maze went on and on forever," she tells you. "I don't think it had an end!" Fortunately, a friendly cat directed her to the exit.

   This strange tale has you thinking about infinite mazes. **Which of the following statements are true about searching infinite mazes (graphs) where all nodes have *finite* degree?**

   Assume the following about the infinite maze:

   - the maze has a single connected component and that we are starting from a start location and searching for an end location that exists somewhere **in** the maze,
   - there is a finite distance between the start and end locations, and
   - we stop the traversal once the end location is reached.

   ■ A DFS may never terminate.
   ■ A BFS may never terminate.

   DFS and BFS will never terminate on an infinite cyclic graph.

   ■ A DFS might terminate, but the solution may not be the shortest path.

   DFS only guarantees the shortest path on a tree.

   ■ A BFS might terminate, but the solution may not be the shortest path.

   BFS only guarantees the shortest path on unweighted, finite graphs.

   □ A DFS will terminate, and the solution will be the shortest path.
   □ A BFS will terminate, and the solution will be the shortest path.

   DFS and BFS are not guaranteed to terminate on infinite graphs.

5. (30 points) [W2/3, ★★★★] A primary application of BFS is finding shortest paths in an unweighted graph. Consider a weighted graph with edges of cost 0 and 1. Describe how to modify the BFS algorithm to find the shortest path between two vertices in this type of graph. The worst case complexity must remain the same ($\mathcal{O}(V + E)$).

   Modification: Instead of using a queue, use a double-ended queue. When adding a vertex to the queue, add it to the back if the edge has weight 1, and add it to the front if the edge has weight 0. This ensures that the shortest path is found first.

   0-1BFS

```
Initialize a double queue $Q$ and a parent array $P$.
Enqueue the start vertex $s$ into $Q$ and set $P[s] = NULL$.
While $Q$ is not empty:
    Dequeue the front vertex $u$ from $Q$.
    For each neighbor $v$ of $u$:
        If $P[v]$ is undefined:
            Set $P[v] = u$.
            If the edge $(u, v)$ has weight 0:
                Enqueue $v$ at the front of $Q$.
            Else: (weight of 1)
                Enqueue $v$ at the back of $Q$.
Return $P$.
```