Midterm 2

Instructions:

- Do not begin the exam until directed to do so. Read all the instructions on this page.
- Write your name below. In addition, write your first or last name at the top of **every** subsequent page.
- Stay inside the frame, and do not separate or detach pages. All pages must be available for scanning.
- **You are allowed two double-sided, letter-sized sheets with your own notes** (in addition to the handout page with the table). Calculators, cell phones, programmable devices, and communication devices are prohibited.
- Do not discuss the exam until after grades have been released.
- You should be able to fit your solutions in the spaces provided. If you need more space, continue on the extra pages at the end of the exam booklet. Make sure to leave pointers in both directions (e.g. Continued on Extra Page 2 / Problem 4 continued) so that we can find your solutions.
- Solutions must be **clear and concise**. Algorithms should be described in English, not written in Python.
- Proof is always required unless otherwise specified.
- Runtime analysis of an algorithm requires both upper and lower bounds unless otherwise specified.

Advice:

- You have 120 minutes to earn a maximum of 100 points. **Do not spend too much time on any single problem.** Read them all first, and attack them in the order that allows you to make the most progress.
- Do not waste time re-deriving results from class. Simply state and cite them.

Question	Points
Multiple Choice	30
Unreachability	20
Rainbow Graph	18
Hungry Psyducks	32
Total:	100

Name: _____ MIT ID (9 digit #): _____

Problem 1. [30 points] **Multiple Choice** (8 parts)

FILL IN THE APPROPRIATE SQUARES COMPLETELY to mark your answers. This problem will be graded automatically, so you may lose credit if you mark your answers differently!

Proof is not required for this problem; points will be awarded solely based on your final answer. Some parts require you to select exactly one answer; others require you to select all correct answers.

(a) [3 points] Which of the following solves SSSP most efficiently on weighted DAGs?
Select the one best answer.

- ☐ BFS
- ■ DAG SP
- ☐ Dijkstra
- ☐ Bellman-Ford
- ☐ Johnson

(b) [3 points] Which of the following solves SSSP most efficiently on weighted directed graphs?
Select the one best answer.

- ☐ BFS
- ☐ DAG SP
- ☐ Dijkstra
- ■ Bellman-Ford
- ☐ Johnson

(c) [3 points] Which of the following solves SSSP most efficiently on unweighted directed graphs?
Select the one best answer.

- ■ BFS
- ☐ DAG SP
- ☐ Dijkstra
- ☐ Bellman-Ford
- ☐ Johnson

(d) [3 points] Which of the following solves SSSP most efficiently on positive-weighted undirected graphs?
Select the one best answer.

- ☐ BFS
- ☐ DAG SP
- ■ Dijkstra
- ☐ Bellman-Ford
- ☐ Johnson

(e) [3 points] Which of the following solves APSP most efficiently on directed graphs?
Select the one best answer.

- ☐ BFS
- ☐ DAG SP
- ☐ Dijkstra
- ☐ Bellman-Ford
- ■ Johnson

(f) [3 points] Which of the following can be used to compute topological orders most efficiently in DAGs?
Select the one best answer.

- ☐ DFS
- ■ Full DFS
- ☐ BFS
- ☐ Kosaraju-Sharir

Solution: While K-S will work, Full DFS is the better answer.

(g) [6 points] Which of the following are always true of the condensation graph H of a DAG G?
Select all that apply.

- ■ H is acyclic
- ☐ H is strongly connected
- ■ H is unique (up to relabeling of vertices)
- ■ H is the same as G (up to relabeling of vertices)

Solution: Condensation graphs are always acyclic and unique. If G is a DAG, each vertex is its own SCC, so $H \cong G$. Acyclic graphs with multiple vertices are never strongly connected.

Rubric:

- -3pts per error (down to min of 0)

(h) [6 points] Which of the following can be used to solve s-t Reachability in linear time?
Select all that apply.

- ■ DFS
- ☐ Full DFS
- ■ BFS
- ☐ Kosaraju-Sharir

Solution: DFS and BFS both work. Depending on implementation of Full DFS and K-S, they could be interpreted to work or not. Staff were not consistent in answering questions, so we treat all answers as correct.

Rubric:

- 3pts DFS
- 3pts BFS

Problem 2. [20 points] **Unreachability** (2 parts)
Let $G = (V, E)$ be a directed graph. Say that vertices $u, v \in V$ are *semi-connected* iff G contains either a u-v path or a v-u path (or both).

(a) [12 points] Suppose G is acyclic. Describe an algorithm that finds and outputs a pair of vertices that are **not** semi-connected, or $\perp$ if no such pair exists. Briefly justify correctness. Prove **one** of the runtime upper bounds below. Mark your selected option by filling in the square completely.

- ■ For up to 12 points, prove that your algorithm runs in $O(|V| + |E|)$ time.
- ☐ For up to 10 points, prove that your algorithm runs in $O(|V| \log |V| + |E|)$ time.
- ☐ For up to 6 points, prove that your algorithm runs in $O(|V||E|)$ time.

Solution: 12pts

1. Add a supersource s with edges to every $v \in V$, and set all edge weights to -1.
2. DAG SP from s to compute distances δ.
3. COUNTING SORT vertices by $-\delta$; output any collision, or $\perp$ if none.

Runtime analysis:
Step 1 takes $O(|V| + |E|)$ time. Step 2 takes $\Theta(|V| + |E|)$ time. Step 3 takes $O(|V|)$ time. Total is $\Theta(|V| + |E|)$.
Correctness:
Suppose u and v are output. Then $u \neq v$ by construction, and $\delta(u) = \delta(v)$. Every non-empty path has negative weight, so there is no path between them.
Suppose instead $\perp$ is output. Then every vertex has a different distance from s, so there exists v at distance (at most) $-n$. Every edge has weight -1, so the path π from s to v contains every vertex in G, i.e. all vertices are semi-connected.
Variants:

1. Use BFS instead of DAG SP.
2. Start by initializing all distances to 0 instead of adding a supersource.

Solution: 12pts

1. Sort vertices topologically.
2. Output any two consecutive non-adjacent vertices, or $\perp$ if no such pair exists.

Runtime Analysis:
Step 1 takes $\Theta(|V| + |E|)$ time with Full DFS. Step 2 takes $\Theta(|V|)$ time. Total is $\Theta(|V| + |E|)$.
Correctness:
Suppose u and v are output. Then there is no path of length ≤ 1 between u and v by construction. Any longer path would have to pass through a vertex that is not between u and v in the topological order, so would have an edge that violates the order.
Suppose instead that $\perp$ is output. Then the topological order is a path of length $n - 1$, i.e. all vertices are semi-connected.

Solution: 10pts
Use Dijkstra or comparison sort.

Solution: 6pts
Use Bellman-Ford or AP reachability.

Common Mistakes:

- Solution doesn't work on graphs that are weakly connected but not semi-connected
  - Examples (algorithm returns $\perp$ on any of the following, even though u and w are not semi-connected):
    - $u \leftarrow v \rightarrow w$
    - $u \rightarrow v \leftarrow w$
    - $v \rightarrow w \rightarrow x$ and $v \rightarrow u \rightarrow x$ in one graph
  - Claims that weakly connected is the same as semi-connected
  - Attempts to solve like Pset 4 origin/meeting point problem
  - Attempts to make graph undirected and then find non-connected vertices
- Attempts to use condensation graph (this does nothing; DAGs are their own condensation graphs)
- Attempts to DAG relax from first vertex in topological order and simply return any pair of vertices where one vertex is reached and other is not
- Attempts to e.g. Full DFS or B-F and detect what is not reached, but this depends on starting from a good vertex
- **BEFORE YOU SUBMIT A REGRADE REQUEST**, check that your algorithm returns u and v (and nothing else) in the above three examples, no matter where you begin your graph search.

(b) [8 points] For this part, you may assume an $O(|V|+|E|)$ time solution to (a), even if you did not solve it yourself. You may no longer assume that G is acyclic. Describe an algorithm that finds and outputs a pair of vertices that are **not** semi-connected, or $\perp$ if no such pair exists. Prove that your algorithm runs in $O(|V|+|E|)$ time. You need not prove correctness.

Solution:

1. Compute the condensation graph G' of G, and replace each SCC in G' with one of its vertices.
2. Run (a) on G' and output the result.

Runtime analysis:
Step 1 takes $\Theta(|V|+|E|)$ time with Kosaraju. Step 2 takes $O(|V|+|E|)$ time by assumption. Total runtime is $\Theta(|V|+|E|)$.

Common Mistakes:

- Trying to use a different (incorrect) algorithm than (a)
- Trying to remove cycles from the graph using Full DFS

Problem 3. [18 points] **Rainbow Graph** (1 part)
Let $G = (V, E, w)$ be a weighted, directed graph. Suppose each vertex of G is colored with one of seven colors. Design an algorithm that outputs, for every vertex v, the shortest distance from v to any vertex whose color is different from that of v. You need not prove correctness. Prove **one** of the runtime upper bounds below. Mark your selected option by filling in the square completely.

- ■ For up to 18 points, prove that your algorithm runs in $O(|V||E|)$ time.
- ☐ For up to 8 points, prove that your algorithm runs in $O(|V|^2 \log |V| + |V||E|)$ time.
- ☐ For up to 4 points, prove that your algorithm runs in $(|V| + |E|)^{O(1)}$ time.

Solution: 18pts

1. WLOG $|E| \in \Omega(|V|)$
2. Reverse edges of G to form a graph G'.
3. For each color i, add a supersource s_i to G' with weight 0 edges to all vertices of color i.
4. Use Bellman-Ford starting from each s_i to compute distances δ_i.
5. For each vertex v, compute the minimum $\delta_i(v)$ over all i that aren't v's color.

G' has $|V|+7$ vertices and $\Theta(|V|+|E|)$ edges, so can be constructed in $\Theta(|V|+|E|)$ time. Each Bellman-Ford takes $\Theta((|V|+7)(|V|+|E|)) = \Theta(|V||E|)$ time, so step 3 takes $\Theta(|V||E|)$ time. Step 4 iterates over V and performs a constant number of comparisons per vertex, so takes $\Theta(|V|)$ time. Total runtime is then $\Theta(|V||E|)$.

Step 1 can be addressed in any of several ways:

- First remove vertices with no outgoing edges, and record their outputs as ∞.
- In step 3, add edges only to the vertices of color i with outgoing edges.
- Omit step 3, and instead initialize distance 0 to all vertices of one color when running Bellman-Ford.
- Omit step 3, and instead add the supersource to the first layer of the DAG generated by Bellman-Ford.
- If using the iterative version of Bellman-Ford, relax supersource edges only once.

Variant: Connect s_i to vertices *not* of color i, and then output $\delta_i(v)$ if v has color i.

Solution: 8pts
Use Johnson and then brute-force.

Solution: 4pts
Brute-force with $|V|$ iterations of Bellman-Ford.

Common Mistakes:

- Not addressing Step 1 above
- Failing to reverse edges or assuming G is undirected
- Forgetting to specify output

- Assuming (perhaps implicitly) that G has non-negative edge weights or similar
- SP runtime analysis w.r.t. wrong graph, e.g. claiming $\Theta(|V||E|)$ runtime for Bellman-Ford on a graph that doesn't have $|V|$ vertices or $|E|$ edges.
- Solving the wrong problem:
  - Solving the problem for only a single v instead of all v
  - Finding the shortest *edge* (instead of *path*) from v to a vertex of a different color
  - Finding the shortest path between *any* two vertices of different colors
  - Not meeting the chosen runtime bound
  - These errors and similar result in an automatic zero for any solution that selected the full credit runtime option.

Problem 4. [32 points] **Hungry Psyducks** (3 parts)
You are trying to navigate from your dorm room to your 6.1210 exam. Unfortunately, MIT has been taken over by a waddling[1] of hungry Psyducks. MIT has n rooms, m pairs of which have a door between them, where $m \geq n$. You may assume that all doors can be opened from either side. You need not prove correctness in this problem.

(a) [6 points] Suppose that doors are guarded by Psyducks. Every door is guarded by at least one Psyduck, and different doors may have different numbers of Psyducks. In order to pass through a door, you must feed a cookie to each of the guarding Psyducks. Design an algorithm to compute the minimum number of cookies you need to bring in order to make it to your exam. Prove **one** of the runtime upper bounds below. Mark your selected option by filling in the square completely.

■ For up to 6 points, prove that your algorithm runs in $O(n \log n + m)$ time.
□ For up to 3 points, prove that your algorithm runs in $O(mn)$ time.

Solution: 6pts
Let s be the dorm room, t exam room.

1. Create a weighted graph $G = (V, E, w)$, where
   - V is set of rooms
   - E is set of doors
   - $w(e)$ is the number of Psyducks guarding e
2. Run Dijkstra from s and output $\delta(s, t)$.

Runtime analysis:
G has n vertices and m edges, so step 1 takes $\Theta(m + n)$ time, and step 2 takes $\Theta(n \log n + m)$ time. The latter dominates.

Solution: 3pts
Use Bellman-Ford.

Common Mistakes:

- Using SP algorithm without defining a graph

[1] "Waddling" is the collective noun for ducks when on land.

(b) [10 points] As in the previous part, suppose that doors are guarded by Psyducks. However, one door is guarded by a single confused Psyduck. The confused Psyduck will give you a dozen cookies instead of demanding a cookie the first time you pass through its door. Design an algorithm to compute the minimum number of cookies you need to bring in order to make it to your exam. Prove **one** of the runtime upper bounds below. Mark your selected option by filling in the square completely.

■ For up to 10 points, prove that your algorithm runs in $O(n \log n + m)$ time.

□ For up to 5 points, prove that your algorithm runs in $O(mn)$ time.

Solution: 10pts
Let s be the dorm room, t the exam room, and suppose that the confused Psyduck is between rooms u and v.

1. Create G as before.
2. Create G' from G by removing edge $\{u, v\}$
3. Use Dijkstra to compute distances δ from t in G and δ' from s in G'.
4. Output the minimum of the following:
   - $\delta'(s, t)$
   - $\delta'(s, u) + \max(\delta(t, v) - 12, 0)$
   - $\delta'(s, v) + \max(\delta(t, u) - 12, 0)$

Runtime analysis:
Same analysis as previous part, increased by a factor of 2.

Solution: 5pts
Use Bellman-Ford.

Common Mistakes:

- Entering cookie debt, e.g. by finding the smallest *net* cookie cost without regard to when the cookies are paid or acquired
- Treating doors as vertices, including assuming that the confused Psyduck only guards one vertex
- Running Dijkstra on negative-weighted graph
- Attempting to edit edge weights after traversal with SP algorithms
- Reweighting edges (e.g. universal $+12$) without preserving shortest paths
- Incorrect path duplication
- Running B-F on a graph with a -12 edge without considering the resulting negative cycle

(c) [16 points] Suppose that the Psyducks are all in the interiors of rooms instead of guarding doors. Before entering a room, you must feed a cookie to each Psyduck in the room. There are m Psyducks in total, and each room is occupied by at least one Psyduck. Design an algorithm to compute the minimum number of cookies you need to bring in order to make it to your exam. Prove **one** of the runtime upper bounds below. Mark your selected option by filling in the square completely.

- ■ For up to 16 points, prove that your algorithm runs in $O(m)$ time.
- ☐ For up to 12 points, prove that your algorithm runs in $O(m \log m)$ time.
- ☐ For up to 8 points, prove that your algorithm runs in $O(mn)$ time.

Solution: 16pts

1. Create a directed graph G which contains:
   - For each room v occupied by k Psyducks, a directed path of k vertices $v_1, v_2, \ldots, v_{k-1}, v_f$
   - For each door between rooms u and v, edges (u_f, v_1) and (v_f, u_1).
2. Run BFS on G from s_f, and output $\delta(s_f, t_f)$.

(The vertex v_f represents being in room v, and v_i represents trying to enter room v and having fed i of the occupying Psyducks.)

Runtime analysis:
G contains m vertices and $3m - n$ edges. Both steps take time linear in the size of G, i.e. time $\Theta(m)$.

Solution: 12pts

1. Create a directed graph G which contains:
   - One vertex for each room
   - For each door between rooms u and v, antiparallel edges (u, v) (with weight equal to the number of Psyducks in v) and (v, u) (with weight equal to the number of Psyducks in u)
2. Run Dijkstra on G from s, and output $\delta(s, t)$.

Runtime analysis: G has $n \leq m$ vertices and $2m$ edges, so step 1 takes $\Theta(m)$ time, and step 2 takes $\Theta(m \log m)$ time.

Solution: 8pts
Use Bellman-Ford, or duplicate edges instead of vertices in first solution

Common Mistakes:

- Duplicating edges instead of vertices, which can result in a graph of size $\Omega(mn)$
- Trying to compute SCCs - this is useless
- Treating doors as vertices and rooms as edges; this gives a hypergraph, not a graph