SID: _____

Q1. [15 pts] CSP: Air Traffic Control

We have five planes: A, B, C, D, and E and two runways: international and domestic. We would like to schedule a time slot and runway for each aircraft to **either** land or take off. We have four time slots: $\{1, 2, 3, 4\}$ for each runway, during which we can schedule a landing or take off of a plane. We must find an assignment that meets the following constraints:

- Plane B has lost an engine and must land in time slot 1.
- Plane D can only arrive at the airport to land during or after time slot 3.
- Plane A is running low on fuel but can last until at most time slot 2.
- Plane D must land before plane C takes off, because some passengers must transfer from D to C.
- No two aircrafts can reserve the same time slot for the same runway.

(a) [3 pts] Complete the formulation of this problem as a CSP in terms of variables, domains, and constraints (both unary and binary). Constraints should be expressed implicitly using mathematical or logical notation rather than with words.

Variables: A, B, C, D, E for each plane.

Domains: a tuple *(runway type, time slot)* for runway type $\in \{international, domestic\}$ and time slot $\in \{1, 2, 3, 4\}$.

Constraints:

$$B[1] = 1$$
$$D[1] \geq 3$$
$$A[1] \leq 2$$
$$D[1] < C[1]$$
$$A \neq B \neq C \neq D \neq E$$

(b) For the following subparts, we add the following two constraints:

- Planes A, B, and C cater to international flights and can only use the international runway.
- Planes D and E cater to domestic flights and can only use the domestic runway.

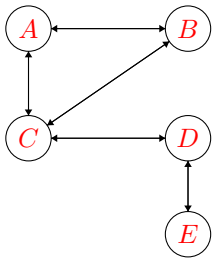
(i) [2 pts] With the addition of the two constraints above, we completely reformulate the CSP. You are given the variables and domains of the new formulation. Complete the constraint graph for this problem given the original constraints and the two added ones.

Variables: A, B, C, D, E for each plane.

Domains: $\{1, 2, 3, 4\}$

Explanation of Constraint Graph:
We can now encode the runway information into the identity of the variable, since each runway has more than enough time slots for the planes it serves. We represent the non-colliding time slot constraint as a binary constraint between the planes that use the same runways.

Constraint Graph:



(ii) [4 pts] What are the domains of the variables after enforcing arc-consistency? Begin by enforcing unary constraints. (Cross out values that are no longer in the domain.)

A	~~1~~	2	~~3~~	~~4~~
B	1	~~2~~	~~3~~	~~4~~
C	~~1~~	~~2~~	~~3~~	4
D	~~1~~	~~2~~	3	~~4~~
E	1	2	~~3~~	4

(iii) [4 pts] Arc-consistency can be rather expensive to enforce, and we believe that we can obtain faster solutions using only **forward-checking** on our variable assignments. Using the Minimum Remaining Values heuristic, perform backtracking search on the graph, breaking ties by picking lower values and characters first. List the $(variable, assignment)$ pairs in the order they occur (including the assignments that are reverted upon reaching a dead end). Enforce unary constraints before starting the search.

(You don't have to use this table, it won't be graded.)

A	1	2	3	4
B	1	2	3	4
C	1	2	3	4
D	1	2	3	4
E	1	2	3	4

Answer: (B, 1), (A, 2), (C, 3), (C, 4), (D, 3), (E, 1)

(c) [2 pts] Suppose we have just one runway and n planes, where no two planes can use the runway at once. We are assured that the constraint graph will always be tree-structured and that a solution exists. What is the runtime complexity in terms of the number of planes, n, of a CSP solver that runs arc-consistency and then assigns variables in a topological ordering?

- ○ $O(1)$
- ○ $O(n)$
- ○ $O(n^2)$
- ● $O(n^3)$
- ○ $O(n^n)$
- ○ None of the Above

Modified AC-3 for tree-structured CSPs runs arc-consistency backwards and then assigns variables in forward topological (linearized) ordering so we that we don't have to backtrack. The runtime complexity of modified AC-3 for tree-structured CSPs is $O(nd^2)$, but note that the domain of each variable must have a domain of size at least n since a solution exists

Q3. [14 pts] State Representations and State Spaces

For each part, state the size of a minimal state space for the problem. Give your answer as an expression that references problem variables. Below each term, state what information it encodes. For example, you could write $2 \times MN$ and write "whether a power pellet is in effect" under the 2 and "Pacman's position" under the MN. State spaces which are complete but not minimal will receive partial credit.

Each part is independent. A maze has **height M and width N**. A Pacman can move *NORTH, SOUTH, EAST, or WEST*. There is initially a **pellet in every position** of the maze. The **goal is to eat all of the pellets.**

(a) [4 pts] Personal Space

In this part, there are P Pacmen, numbered $1, \ldots, P$. Their turns cycle so Pacman 1 moves, then Pacman 2 moves, and so on. Pacman 1 moves again after Pacman P. Any time two Pacmen enter adjacent positions, the one with the lower number dies and is removed from the maze.

$$(MN)^P \times 2^{MN} \times P$$

$(MN)^P$: for each Pacman (the dead Pacmen are "eaten" by the alive Pacman, we encapsulate this in the transition function so that Pacmen in the same position must move together and can only move during the turn of the higher numbered Pacman)

2^{MN}: for each position, whether the pellet there has been eaten

P: the Pacman whose turn it is

$(MN+1)^P \times 2^{MN} \times P$ is also accepted for most of the points. Where $(MN+1)$ also includes DEAD as a position

(b) [4 pts] Road Not Taken

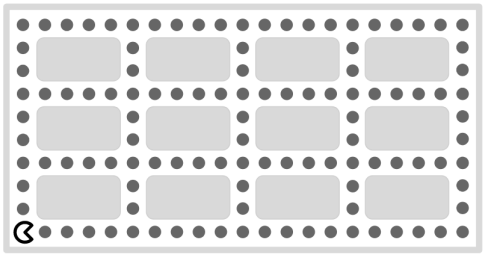
In this part, there is one Pacman. Whenever Pacman enters a position which he has visited previously, the maze is reset – each position gets refilled with food and the "visited status" of each position is reset as well.

$$MN \times 2^{MN}$$

MN: Pacman's position

2^{MN}: for each position, whether the pellet there has been eaten (and equivalently, whether it has been visited)

(c) [6 pts] Hallways



In this part, there is one Pacman. The walls are arranged such that they create a grid of H hallways **total**, which connect at I intersections. (In the example above, $H = 9$ and $I = 20$). In a single action, Pacman can move from one intersection into an adjacent intersection, eating all the dots along the way. Your answer should only depend on I and H.

(note: H = number of vertical hallways + number of horizontal hallways)

SID: _____

$$I \times 2^{2I-H}$$

I: Pacman's position in any one of the intersections

2^{2I-H}: for each path between intersections, whether the pellets there have been eaten. The exponent was calculated via the following logic:

Approach 1:
Let v be the number of vertical hallways and h be the number of horizontal hallways. Notice that $H = v + h$ and $I = v * h$

Each vertical hallway has $h - 1$ segments, and each horizontal hallway has $v - 1$ segments.
Together, these sum to a total of $v(h-1)+h(v-1) = 2vh - v - h = 2I - H$ segments, each of which is covered by a single action.

Approach 2:
Let $H = v + h$ where v is the number of vertical hallways and h is the number of horizontal hallways

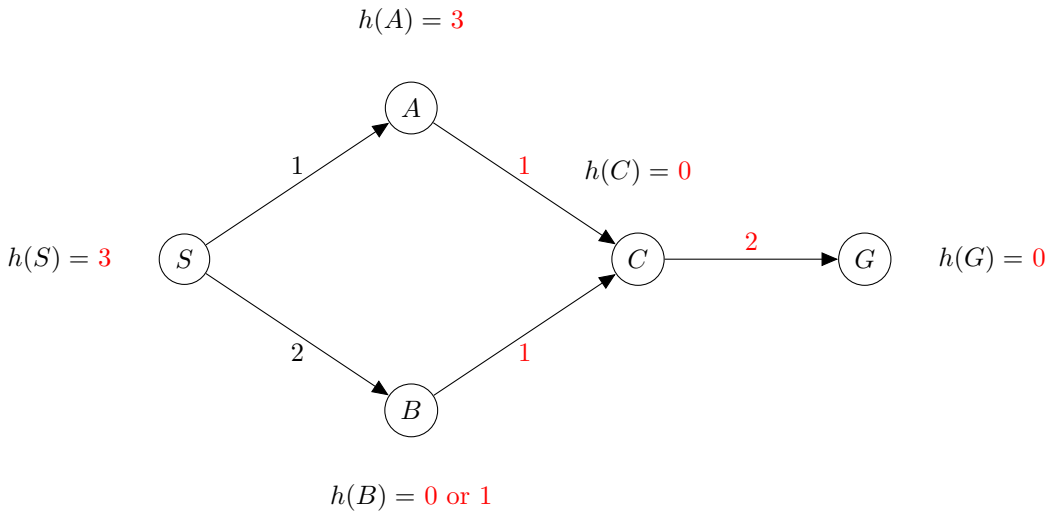
$4I$ = Every intersection has 4 paths adjacent to it
$-2v$ = The top and bottom intersection of each vertical hallway has one less vertical path
$-2h$ = The right-most and left-most intersection of each horizontal hallway has one less horizontal path
$4I - 2v - 2h$ must be divided by 2 since you count every path twice in each direction from the intersections which gives you

$$\frac{4I - 2v - 2h}{2} = 2I - v - h = 2I - H$$

Q4. [14 pts] Informed Search and Heuristics

(a) [6 pts] Consider the state space shown below, with starting state S and goal state G. Fill in a cost from the set $\{1, 2\}$ **for each blank edge** and a heuristic value from the set $\{0, 1, 2, 3\}$ **for each node** such that the following properties are satisfied:

- The heuristic is admissible but not consistent.
- The heuristic is monotonic non-increasing along paths from the start state to the goal state.
- A* graph search finds a suboptimal solution.
- You will never encounter ties (two elements in the fringe with the same priority) during execution of A*.

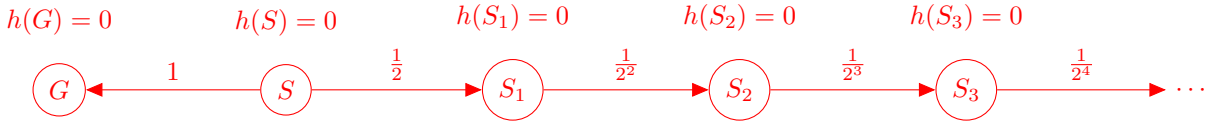


(b) [8 pts] **Don't spend all your time on this question.** As we saw in class, A* graph search with a consistent heuristic will always find an optimal solution when run on a problem with a finite state space. However, if we turn to problems with infinite state spaces, this property may no longer hold. Your task in this question is to provide a concrete example with an infinite state space where A* graph search fails to terminate or fails to find an optimal solution.

Specifically, you should describe the state space, the starting state, the goal state, the heuristic value at **each node**, and the cost of **each transition**. Your heuristic should be consistent, and all step costs should be strictly greater than zero ($cost \in \mathbb{R}_{>0}$) to avoid trivial paths with zero cost. To keep things simple, each state should have a finite number of successors, and the goal state should be reachable in a finite number of actions from the starting state.

You may want to start by drawing a diagram.

A* may not terminate if the costs are not bounded away from zero. Consider the following example with starting state S, goal state G, and the trivial heuristic $h(s) = 0$ for all s:



After expanding the starting state S, A* will expand $S_1, S_2, \ldots$ and will never expand the goal G, since the path cost of $S \to S_1 \to \cdots \to S_k$ is

$$\sum_{j=1}^{k} \frac{1}{2^j} = 1 - \frac{1}{2^k},$$

which is strictly less than 1 for all k.

There is also a shortcut which involves you noticing that the problem is highly symmetrical such that $Q^*(A, DOWN) = Q^*(A, RIGHT)$ is the same as solving the equivalence of $V^*(A)$ in the previous part and the utility of an infinite cycle with reward scaled by half (to account for staying) and discount = 0.5. That leads us to conclude $0.5 + 0.5x = \frac{0.5}{1-0.5} = 1$ so $x = 1$

(c) [4 pts] We now add one more layer of complexity. Turns out that the reward function is not guaranteed to give a particular reward when the agent takes an action. Every time an agent transitions from one state to another, once the agent reaches the new state s', a fair 6-sided dice is rolled. If the dices lands with value x, the agent receives the reward $R(s, a, s') + x$. The sides of dice have value $1, 2, 3, 4, 5$ and 6.

Write down the new bellman update equation for $V_{k+1}(s)$ in terms of $T(s, a, s')$, $R(s, a, s')$, $V_k(s')$, and γ.

$$V_{k+1}(s) = \max_a \sum_{s'} T(s, a, s')[\tfrac{1}{6}(\sum_{i=1}^{6} R(s, a, s') + i) + \gamma V_k(s)]$$
$$= \max_a \sum_{s'} T(s, a, s')(R(s, a, s') + 3.5 + \gamma V_k(s))$$