CS 188 Summer 2016 Introduction to Artificial Intelligence — Midterm 1

- You have approximately 2 hours and 50 minutes.
- The exam is closed book, closed calculator, and closed notes except your one-page crib sheet.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation. All short answer sections can be successfully answered in a few sentences AT MOST.
- For multiple choice questions with *circular bubbles*, you should only mark ONE option; for those with *checkboxes*, you should mark ALL that apply (which can range from zero to all options)

First name	
Last name	
edX username	

For staff use only:

Q1.	Straightforward Questions	/27
Q2.	More Advanced Problems	/19
Q3.	Wandering Poet	/12
Q4.	Search	/16
Q5.	Trees	/16
Q6.	Reward Shaping	/21
Q7.	Spinaroo	/17
	Total	/128

THIS PAGE IS INTENTIONALLY LEFT BLANK

Q1. [27 pts] Straightforward Questions

(a) Agents

(i) [1 pt] A reflex agent can be rational.

● True ○ False

(ii) [1 pt] It is impossible to be rational in a partially observable environment.

○ True ● False

(b) Search Algorithms

(i) [1 pt] Iterative deepening involves re-running breadth-first search repeatedly.

○ True ● False

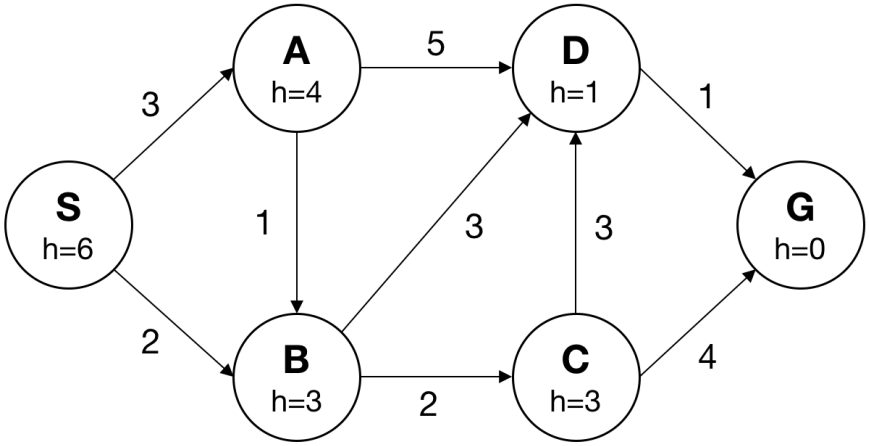
(ii) [1 pt] Greedy search can return optimal solutions.

● True ○ False

(iii) [2 pts] Which of the following search algorithms returns the optimal path if the costs are all are a fixed cost $C > 0$? Mark all that apply.

■ UCS ☐ DFS
■ BFS ■ Iterative Deepening

Consider the following graph. For the following subquestions, ties are broken in alphabetical order.



(iv) [1 pt] What path would DFS return?
S-A-B-C-D-G

(v) [1 pt] What path would BFS return?
S-A-D-G

(vi) [1 pt] What path would UCS return?
S-B-D-G

(vii) [1 pt] What path would Greedy search return?

S-B-D-G

(c) Heuristics

(i) [1 pt] The Euclidean distance is an admissible heuristic for Pacman path-planning problems.

● True ○ False

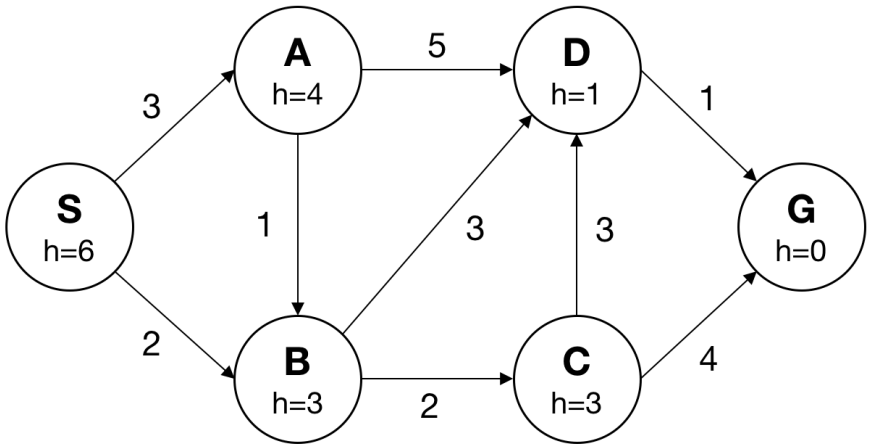
(ii) [1 pt] The sum of several admissible heuristics is still an admissible heuristic.

○ True ● False

(iii) [1 pt] Write down a heuristic that is **always** admissible, for any search problem:

$h(x) =$ 0

Consider the following graph again.



(iv) [2 pts] Is the heuristic in the above graph admissible? If not, provide a minimal set of edges whose costs must be changed along with their new costs in order to make the heuristic admissible.

● Yes ○ No

(v) [2 pts] Is the heuristic in the above graph consistent? If not, provide a minimal set of edges whose costs must be changed along with their new costs in order to make the heuristic consistent.

○ Yes ● No

Change the cost of S-B to something equal to or above 3.

(d) CSPs

Alice, Bob, Carol, and Donald are picking their entrees at a restaurant. The entrees are pasta, quesadillas, risotto, and sushi. They have some strict dietary preferences:

1. Carol will not order sushi.
2. Alice and Bob want to steal each other's food, which is pointless if they both order the same dish. So Alice and Bob will order different dishes.
3. Bob likes carbs, so he will only order pasta or risotto.
4. Carol likes to be unique in her food orders and will not order the same dish as anybody else, with one exception: Alice and Carol are actually twins, and always order the same dish as each other.
5. Donald really dislikes quesadillas and will not order them.

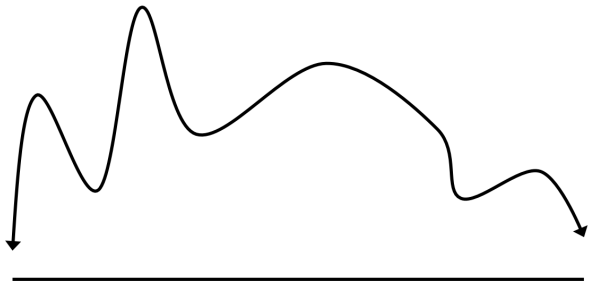
(i) [2 pts] Draw the corresponding constraint graph for this CSP.
Edges: A-B, C-A, C-B, C-D

(ii) [2 pts] We will run basic backtracking search to solve this CSP and make sure that every person (variable) is matched with their dream dish (value). We will select unassigned variables in alphabetical order, and we will also iterate over values in alphabetical order. What assignment will backtracking search return?
Alice = pasta, Bob = risotto, Carol = pasta, Donald = risotto

(iii) [3 pts] Now we will run one iteration of forward checking. Assume that no values have been assigned to the variables. Write down the value(s) that will be eliminated if we assign "pasta" to Alice. Write down "None" if no values will be eliminated.

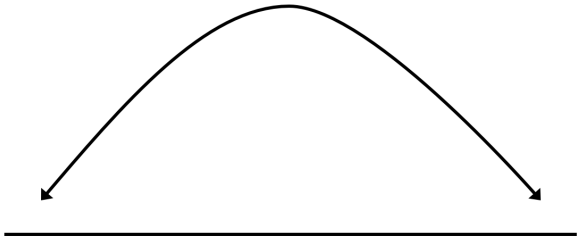
- Values that will be eliminated for Bob: pasta
- Values that will be eliminated for Carol: quesadillas, sushi, risotto (or just quesadillas, risotto)
- Values that will be eliminated for Donald: none

(e) Local Search



(i) [1/2 pts] In the above plot, identify a starting point which would lead to a suboptimal solution if we used hill climbing. Draw a vertical line from your point to the x-axis. If you cannot identify such a point, write 'not possible'. You will receive **zero credit** if you identify more than one point.

Any point that would not lead to the highest peak would suffice.



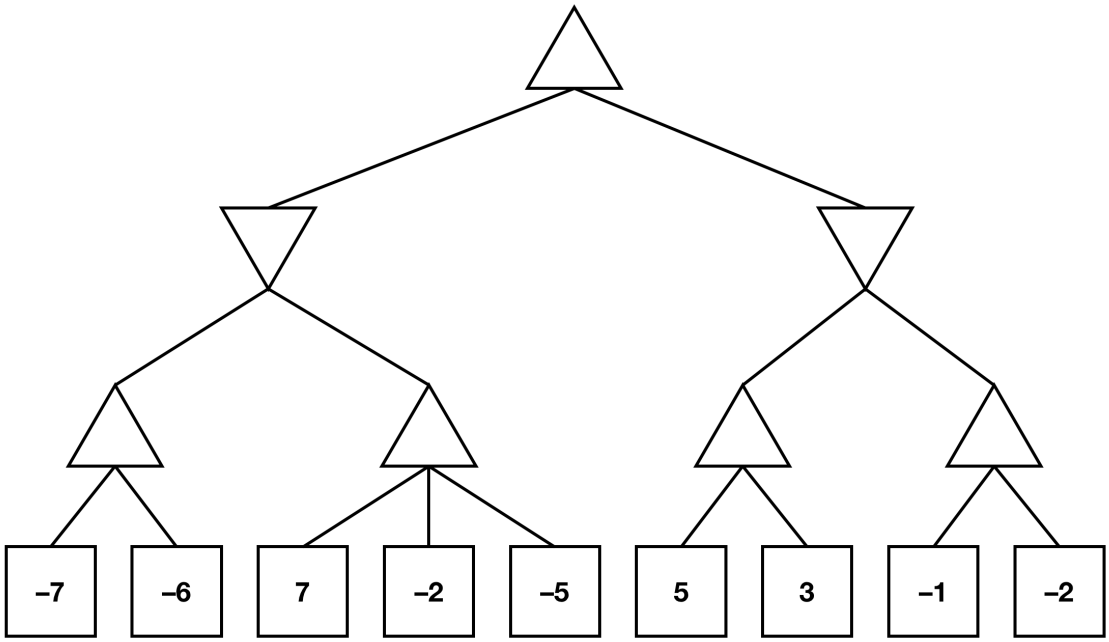
(ii) [1/2 pts] In the above plot, identify a starting point which would lead to a suboptimal solution if we used hill climbing. Draw a vertical line from your point to the x-axis. If you cannot identify such a point, write 'not possible'. You will receive **zero credit** if you identify more than one point.

Every starting point would lead to an optimal solution, because this is a convex function.

(iii) [1 pt] Simulated annealing ensures that you will always reach a global optimum.

○ True ● False

(f) Game Trees



(i) [1 pt] In the minimax tree above, fill in the values for all min and max nodes. Upward-pointing triangles are max nodes, while downward-pointing triangles are min nodes.

Top to bottom, left to right: $-1, -6, -1, -6, 7, 5, -1$

(ii) [1 pt] In the minimax tree above, cross out the branches that would be pruned by alpha-beta search.

The 4th and 5th leaf nodes will be pruned.

Q2. [19 pts] More Advanced Problems

(a) True or False.

(i) [1 pt] If $h_1(s)$ is a consistent heuristic, and $h_2(s)$ is an admissible heuristic, then the minimum of the two must be consistent.

○ True ● False

(ii) [1 pt] Depth first search will always expand at-least as many nodes as A^* search with a consistent heuristic.

○ True ● False

(iii) [1 pt] Admissibility of a heuristic for A^* search implies consistency as well.

○ True ● False

(b) State Spaces. Pacman keeps finding himself in new game scenarios that he doesn't know how to solve! For each game scenario, Pacman knows that he is on an $M \times N$ playing board and he wants to solve the games optimally.

For each of the following game scenarios, answer the questions specific to that game scenario.

(i) [4 pts] Pacman must collect 8 objects scattered across the board. He knows the position of each object. He must collect these 8 objects in a specific order (which he knows). Pacman may visit an object he does not need yet, but he is not allowed to collect it. Once he collects all 8 objects in order, he wins! Provide a minimal state representation for the game, as well as the size of the state space.

minimal state representation:
The minimal state representation includes a tuple to denote pacman's position as well as an integer ranging from 0 to 8, representing how many objects Pacman has found (in the correct order)

size of state space:
$9MN$. There are 9 possibilities for how many objects Pacman has found, and MN choices for his position.

(ii) [6 pts] Pacman realizes that there are P of him, with $P \geq 1$. More than one Pacman can be in any position. Moreover, there are K food pellets throughout the grid, but each one needs to be hit twice (by any Pacman; it can be the same one twice) in order to fully disappear. The goal of the game is to make all food pellets disappear. Provide a minimal state representation to model this game. What is the size of the state space? Lastly, provide a goal test.

minimal state representation:
The minimal state representation includes P position tuples for each pacman as well as a ternary variable for each food pellet which represents whether a food piece has been bitten 0 times, 1 time, or 2+ times.

size of the state space:
$(MN)^P * 3^K$. First, each Pacman can have a choice of MN positions, and each food pellet's ternary

variable can have one of three values.

goal test:
For all ternary variables, the value represents that it has been bitten 2+ times

(c) Heuristics For the following scenario, mark all non-trivial admissible heuristics that can be used for the search problem posed.

(i) [2 pts] K Pac-men are attempting to eat N dots throughout a board. Each Pac-man moves to an adjacent spot simultaneously during each time step. Moreover, each Pac-man starts in the same position.

- [x] Number of food left divided by K
- [ ] The average of all Manhattan distances between every (Pacmann, Food pellet) pair possible.
- [x] Minimum Manhattan distance between any pair of Pacman and food
- [ ] Maximum Manhattan distance between any pair of Pacman and food

(d) Utilities. Pacman decides to go to the market. He is specifically on the hunt for deals on Pac-berries.

Let $U_P(b)$ denote Pacman's utility function for Pacman's acquisition of Pac-berries. Assume that Pacman acts rationally.

(i) [2 pts] Pacman walks into the first vendor on the street and asks for any deals on Pac-berries. Vendor 1 gives Pacman two offers. For the same price, either he can get 10 Pac-berries, or he can pick a number out of a jar at random and get that number of Pac-berries. The jar contains the numbers 0, 16, and 20. Define an increasing function of a number of berries, $U_P(b)$, that is not constant or piecewise such that...

Pacman prefers to take the 10 Pac-berries immediately.

$U_P(b) =$
$\sqrt{b}$ works. Pacman must be risk avoiding in order to do this.

Pacman prefers the three-way lottery.

$U_P(b) =$
b is the easiest one to think of.

(ii) [2 pts] Assume now that the utility function $U_P(b)$ represents a monetary value, as in, $U_P(b)$ represents how much Pacman believes b pac-berries to be worth.

Pacman decides he doesn't appreciate these deals and moves on to the next vendor. Vendor 2 seems more promising, and offers Pacman a chance at either 4 Pac-berries, 15 Pac-berries, or 25 Pac-berries. Vendor 2 presents an interesting twist, however. For \$6, the vendor can assure Pacman that he will never receive the option of 4 Pac-berries. For the following utility functions, choose whether Pacman should play the original lottery or pay the money upfront for the altered lottery.

$U_P(b) = b$

- (●) Original Lottery
- ( ) Altered Lottery

Pacman should play the original lottery. His options are a value of $4/3 + 15/3 + 25/3 = 44/3$ for the original lottery, or $(15)/2 + (25)/2 - 6 = 14$, so he prefers the original lottery.

$U_P(b) = 3b + 3$

○ Original Lottery ● Altered Lottery

Pacman should play the altered lottery. $(3(4) + 3)/3 + (3(15) + 3)/3 + (3(25) + 3)/3$ is smaller than $(3(15) + 3)/2 + (3(25) + 3)/2 - 6$

Q3. [12 pts] Wandering Poet

In country B there are N cities. They are all connected by roads in a circular fashion. City 1 is connected with city N and city 2. For $2 \leq i \leq N-1$, city i is conected with cities $i-1$ and $i+1$.

A wandering poet is travelling around the country and staging shows in its different cities.

He can choose to move from a city to a neighboring one by moving East or moving West, or stay in his current location and recite poems to the masses, providing him with a reward of r_i. If he chooses to travel from city i, there is a probability $1-p_i$ that the roads are closed because of B's dragon infestation problem and he has to stay in his current location. The reward he is to reap is 0 during any successful travel day, and $r_i/2$ when he fails to travel, because he loses only half of the day.

(a) [2 pts] Let $r_i = 1$ and $p_i = 0.5$ for all i and let $\gamma = 0.5$. For $1 \leq i \leq N$ answer the following questions *with real numbers*:

Hint: Recall that $\sum_{j=0}^{\infty} u^j = \frac{1}{1-u}$ for $u \in (0,1)$.

(i) [1 pt] What is the value $V^{stay}(i)$ under the policy that the wandering poet always chooses to stay?

We have that for all i, the Bellman equations for policy evaluation are $V^{stay}(i) = r_i + \gamma V^{stay}(i)$. When $r_i = 1$ and $p_i = 1$ this reduces to $V^{stay}(i) = 1 + 0.5V^{stay}(i)$ which yields $V^{stay}(i) = 2$.

(ii) [1 pt] What is the value $V^{west}(i)$ of the policy where the wandering poet always chooses west?

$$V^{west}(1) = 0.5(\frac{1}{2} + 0.5V^{west}(1)) + 0.5(0.5V^{west}(2)) \tag{1}$$

Since all starting states are equivalent, $V^{west}(1) = V^{west}(2)$. Therefore $V^{west}(1) = V^{west}(2) = \cdots = \frac{1}{2}$.

(b) [5 pts] Let N be even, let $p_i = 1$ for all i, and, for all i, let the reward for cities be given as

$$r_i = \begin{cases} a & i \text{ is even} \\ b & i \text{ is odd,} \end{cases}$$

where a and b are constants and $a > b > 0$.

(i) [2 pts] Suppose we start at an even-numbered city. What is the range of values of the discount factor γ such that the optimal policy is to stay at the current city forever? Your answer may depend on a and b.

For all possible values of γ, staying at an even city will be optimal.

(ii) [2 pts] Suppose we start at an odd-numbered city. What is the range of values of the discount factor γ such that the optimal policy is to stay at the current city forever? Your answer may depend on a and b.

The poet should only move if losing that one extra day fo reward is worth it. So, either he can get the reward of staying for an infinite amount of time at an odd city, which is $b * \frac{1}{1-\gamma}$ or he can move to city a and lose a

whole day of as reward, which is $a * \frac{1}{1-\gamma} - a$. He will only stay if the former is greater than the latter, which is only when $\gamma < \frac{b}{a}$

(iii) [1 pt] Suppose we start at an odd-numbered city and γ does not lie in the range you computed. Describe the optimal policy.

The poet should move to an even city and stay there forever.

(c) [2 pts] Let N be even, $r_i \geq 0$, and the optimal value of being in city 1 be positive, i.e., $V^*(1) > 0$. Define $V_k(i)$ to be the value of city i after the kth time-step. Letting $V_0(i) = 0$ for all i, what is the largest k for which $V_k(1)$ could still be 0? Be wary of off-by-one errors.

Because $V^*(1) > 0$, there must be one $r_i > 0$ for some i. It then follows that $V_1(i) > 0$, $V_2(i-1), V_2(i+1) > 0$ and so on. The worst case is when the diametrically opposite to 1 is the only one having a nonzero r_i. This implies that after $k = M$ steps, $V_{k+1}(1) > 0$ is guaranteed.

(d) [3 pts] Let $N = 3$, and $[r_1, r_2, r_3] = [0, 2, 3]$ and $p_1 = p_2 = p_3 = 0.5$, and $\gamma = 0.5$. Compute:

(i) [1 pt] $V^*(3)$

(ii) [1 pt] $V^*(1)$

(iii) [1 pt] $Q^*(1, stay)$

Notice that $Q^*(1, stay) = \gamma V^*(1)$. Clearly $\pi^*(1) =$ go to 3. $V^*(1) = Q^*(1$,go to $3) = 0.5\gamma V^*(1) + 0.5\gamma V^*(3)$. $V^*(3) Q^*(3, stay) = 3 + \gamma V^*(3)$ Since $\gamma = 0.5$, we have that $V^*(3) = 6$. Therefore $V^*(1) = \frac{4}{3}\frac{1}{4} V^*(3) = 2$. And therefore $Q^*(1, stay) = 1$

Q4. [16 pts] Search

(a) Rubik's Search

Note: You do not need to know what a Rubik's cube is in order to solve this problem.

A Rubik's cube has about 4.3×10^{19} possible configurations, but any configuration can be solved in 20 moves or less. We pose the problem of solving a Rubik's cube as a search problem, where the states are the possible configurations, and there is an edge between two states if we can get from one state to another in a single move. Thus, we have 4.3×10^{19} states. Each edge has cost 1. Note that the state space graph does contain cycles. Since we can make 27 moves from each state, the branching factor is 27. Since any configuration can be solved in 20 moves or less, we have $h^*(n) \leq 20$.

For each of the following searches, estimate the approximate number of states expanded. Mark the option that is closest to the number of states expanded by the search. Assume that the shortest solution for our start state takes exactly 20 moves. Note that 27^{20} is much larger than 4.3×10^{19}.

(i) [2 pts] DFS Tree Search

	20	4.3×10^{19}	27^{20}	∞ (never finishes)
Best Case:	●	○	○	○
Worst Case:	○	○	○	●

(ii) [2 pts] DFS graph search

	20	4.3×10^{19}	27^{20}	∞ (never finishes)
Best Case:	●	○	○	○
Worst Case:	○	●	○	○

(iii) [2 pts] BFS tree search

	20	4.3×10^{19}	27^{20}	∞ (never finishes)
Best Case:	○	○	●	○
Worst Case:	○	○	●	○

(iv) [2 pts] BFS graph search

	20	4.3×10^{19}	27^{20}	∞ (never finishes)
Best Case:	○	●	○	○
Worst Case:	○	●	○	○

(v) [1 pt] A* tree search with a perfect heuristic, $h^*(n)$, Best Case

● 20 ○ 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

(vi) [1 pt] A* tree search with a bad heuristic, $h(n) = 20 - h^*(n)$, Worst Case

○ 20 ○ 4.3×10^{19} ● 27^{20} ○ ∞ (never finishes)

(vii) [1 pt] A* graph search with a perfect heuristic, $h^*(n)$, Best Case

● 20 ○ 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

(viii) [1 pt] A* graph search with a bad heuristic, $h(n) = 20 - h^*(n)$, Worst Case

○ 20 ● 4.3×10^{19} ○ 27^{20} ○ ∞ (never finishes)

(b) Limited A^* Graph Search

Consider a variant of A^* graph search called Limited A^* graph search. It is exactly like the normal algorithm, but instead of keeping all of the fringe, at the end of each iteration of the outer loop, the fringe is reduced to just a certain amount of the best paths. I.e. after all children have been inserted, the fringe is cut down to the a certain length. The pseudo-code for normal A^* graph search is reproduced below, the only modification being an argument W for the limit.

```
1:  function A* GRAPH SEARCH(problem, W)
2:      fringe ← an empty priority queue
3:      fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
4:      closed ← an empty set
5:      ADD INITIAL-STATE[problem] to closed
6:      loop
7:          if fringe is empty then
8:              return failure
9:          node ← REMOVE-FRONT(fringe)
10:         if GOAL-TEST(problem, STATE[node]) then
11:             return node
12:         if STATE[node] not in closed then
13:             ADD STATE[node] to closed
14:             for successor in GETSUCCESSORS(problem, STATE[node]) do
15:                 fringe ← INSERT(MAKE-SUCCESSOR-NODE(successor, node), fringe)
16:         fringe = fringe[0:W]
```

(i) [1 pt] For a positive W, limited A^* graph search is complete.

○ True ● False

(ii) [1 pt] For a positive W, limited A^* graph search is optimal.

○ True ● False

(iii) [2 pts] Provide the smallest value of W such that this algorithm is equivalent to normal A^* graph search (i.e. the addition of line 16 makes no difference to the execution of the algorithm).

W = Size of the State Space

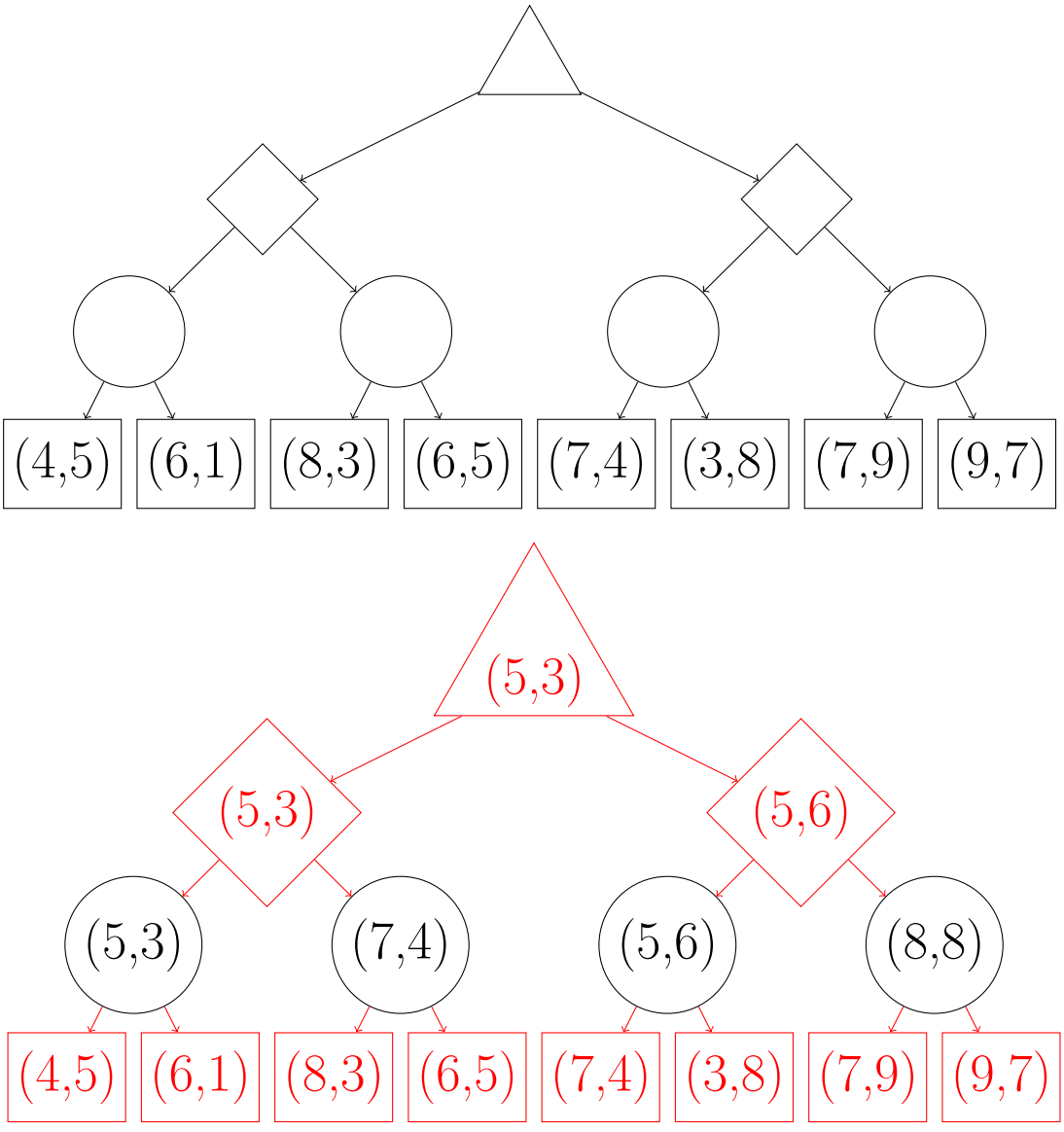
Q5. [16 pts] Trees

(a) [4 pts]
Consider the following game tree for a two-player game. Each leaf node assigns a score to each player, in order (e.g. the leaf node (3, 1) corresponds to player 1 getting 3 points and player 2 getting 1 point).

Note that while player 1's goal is to maximize her own score, player 2's goal is to maximize his relative score (i.e. maximize the expression "player 2's score minus player 1's score").

An upward-facing triangle represents a node for player 1, a diamond represents a node for player 2, and a circle represents a chance node (where either branch could be chosen with equal probability).

Fill in the values of all of the nodes; in addition, put an X on the line of all branches that can be pruned, or write "No pruning possible" below the tree if this is the case. Assume that branches are explored left-to-right.



No pruning.

(b) [4 pts]

Based on the above strategies for each player, fill in the two sides of the inequality below with expressions using the below variables so that if the following inequality holds true, we can prune for player 2. If no pruning is ever possible in a search tree for this game, write 'No pruning possible.'

You may make reference to the following variables:

- α_1 and α_2 are the scores for the best option that any node for player 1 on the path from the root to the current node has seen, including the current node.
- β_1 and β_2 are the scores for the best option that any node for player 2 on the path from the root to the current node has seen, including the current node.

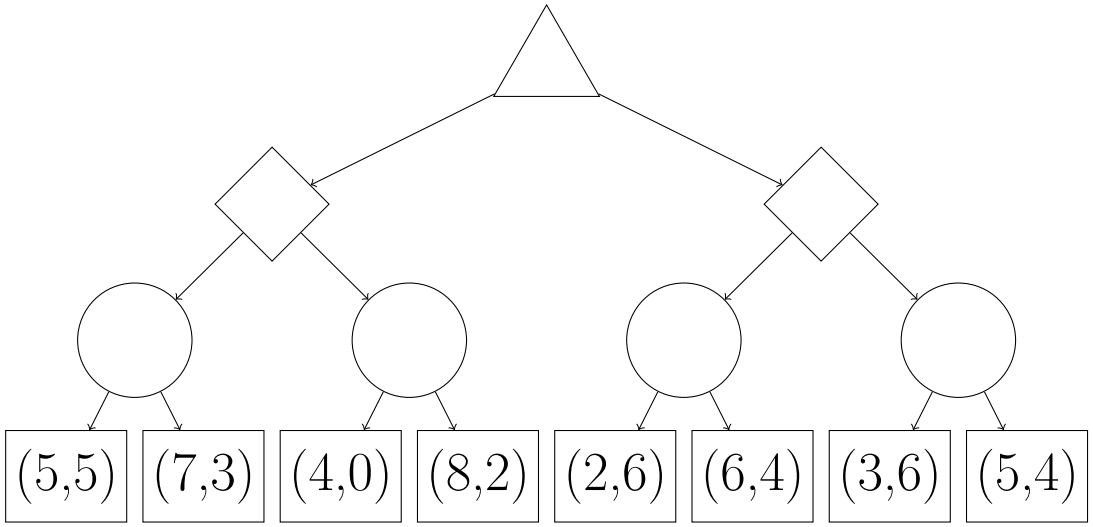
_____ > _____

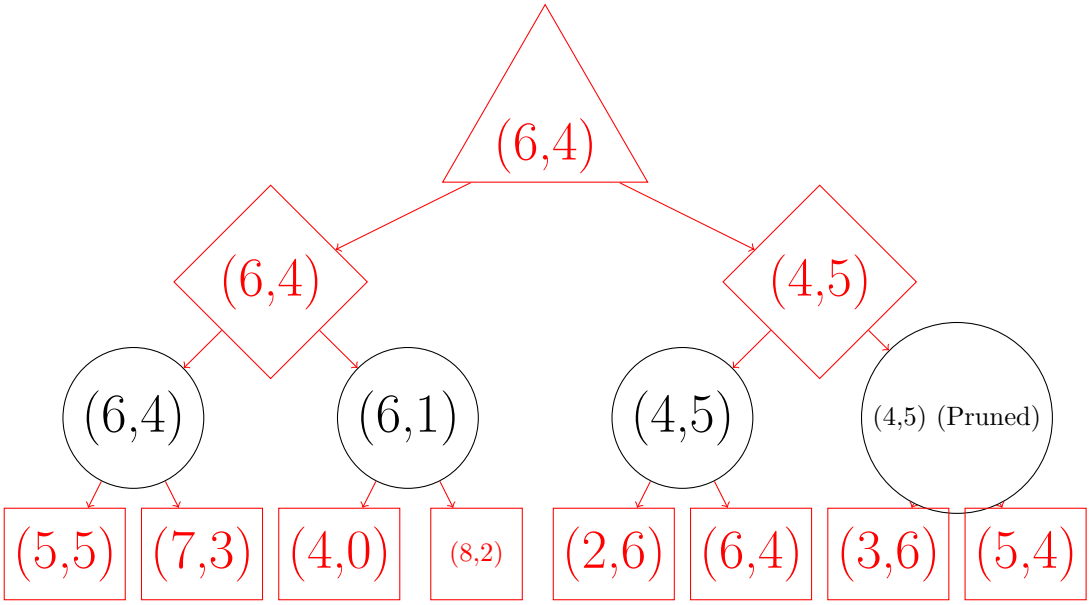
No pruning is possible. For each of the leaf states, we can replace player 2's score with the utility player 2 associates with that state (i.e. subtract player 1's score), and the resulting tree is a regular two-player non-zero-sum game, which cannot be pruned

(c) [4 pts]

Player 1's and player 2's goals remain the same from the two previous parts. However, in this game, no pair of scores at a leaf node can have a sum greater than 10, and both players know this.

Fill in the values of all of the nodes, and put an X on the line of all branches that can be pruned, or write 'No pruning possible' below the tree if this is the case. Assume that branches are explored left-to-right.





(d) [4 pts]

Taking into account this new information, fill in the two sides of the inequality below with expressions using the below variables so that if the following inequality holds true, we can prune for player 2. If no pruning is ever possible in a search tree for this game, write 'No pruning possible.'

You may make reference to the following variables:

- α_1 and α_2 are the scores for the best option that any node for player 1 on the path from the root to the current node has seen, including the current node.
- β_1 and β_2 are the scores for the best option that any node for player 2 on the path from the root to the current node has seen, including the current node.

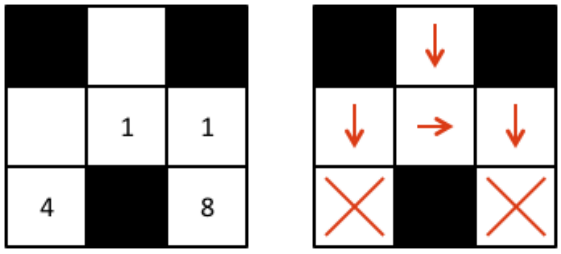
_____ > _____

Any option that player 2 would prefer to (β_1, β_2) must have player 2's score increase by more than player 1's score. In total, both players' scores can increase by at most $10 - \beta_1 - \beta_2$ points, so the highest score player 1 can get in any option player 2 chooses would be $\beta_1 + (10 - \beta_1 - \beta_2)/2$. Pruning can occur if this number is still less than the best option that player 1 has seen for herself, α_1. Thus, the pruning condition is $\alpha_1 > (10 + \beta_1 - \beta_2)/2$.

Q6. [21 pts] Reward Shaping

Consider the following Gridworld-like environment. The robot can move deterministically Up, Down, Right, or Left, or at any time it can exit to a terminal state (where it remains). The reward for any non-exit action is always 0. If the robot is on a square with a number written on it, it receives a reward of that size **on Exiting**. If the robot exits from any square without a number written on it, it receives a reward of 0 (and still exits to a terminal state). Note that when it is on any of the squares (including numbered squares), it can either move Up, Down, Right, Left or Exit. However, it only receives a non-zero reward when it Exits on a numbered square. **The robot is not required to exit on a numbered square; it can also move off of that square. However, if it does not exit, it does not get the reward.**

(a) [3 pts] Draw an arrow in **each** square (including numbered squares) in the following board on the right to indicate the optimal policy PacBot will calculate with the discount factor $\gamma = 0.5$ in the board on the left. (For example, if PacBot would move Down from the square in the middle on the left board, draw a down arrow in that square on the right board.) If PacBot's policy would be to exit from a particular square, draw an X instead of an arrow in that square.



From the middle-right square, we discount once before getting to the 8, making the value of moving that way 4, higher than anything else reachable. Similarly, from the middle square, the value of moving right is 2 (as opposed to 1 for exiting or 1 for moving left), and the value of moving down from the top state is 1. From middle-left, the value of moving down is 2 as opposed to 1 for moving right. From bottom-left, the value for exiting is 4 as opposed to 0.5 for moving up.

The key thing to notice for this problem is that the value of a policy that moves toward an exit decays by a factor of 1/2 for every step. With this in mind, you can compare actions from each state by considering how far you are from each goal. Alternatively, any algorithm we've seen for exactly solving MDPs (value iteration, policy iteration, Q-value iteration) would have worked.

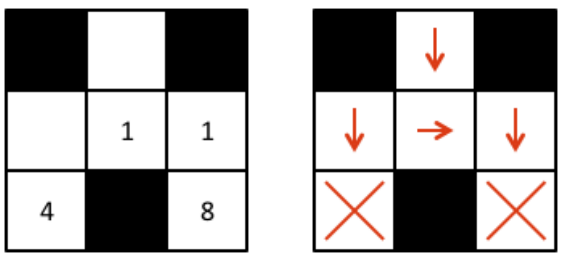
The agent now operates in a new environment with an additional reward function $F(s, a, s')$, which is added to the original reward function $R(s, a, s')$ for every (s, a, s') triplet, so that the new reward function $R'(s, a, s') = R(s, a, s') + F(s, a, s')$.

(b) [3 pts] Consider an additional reward F_1 that favors moving toward numbered squares. Let $d(s)$ be defined as the Manhattan distance from s to the nearest numbered square. If s is numbered, $d(s) = 0$.

$$F_1(s, a, s') = 6\left(d(s) - \frac{1}{2}d(s')\right).$$

F_1 is always 0 when s' is a terminal state (equivalently, when a is the Exit action).

Fill in the diagram as in (a) in the following board to indicate the optimal policy PacBot will calculate with the discount factor $\gamma = 0.5$ and the modified reward function $R'_1(s, a, s') = R(s, a, s') + F_1(s, a, s')$.



This added reward makes no difference to the optimal policy. For any state s in this MDP, $d(s)$ can only be 0 (on a numbered square) or 1 (on a blank square). Any sequence of steps which takes us from a numbered square to another numbered square either stays on numbered squares the whole time, in which case $d(s) = 0$ everywhere and thus $F_1 = 0$ on every transition, or we take a step from a numbered to a non-numbered square and back, in which case we accrue F_1 rewards as follows: if our sequence of states and actions is s_1, a_1, s_2, a_2, s_3 with $d(s_1) = d(s_3) = 0$ and $d(s_2) = 1$, then we get

$$\begin{aligned} F(s_1, a_2, s_2) + \gamma F(s_2, a_2, s_3) &= 6\left(d(s_1) - \frac{1}{2}d(s_2)\right) + \gamma 6\left(d(s_2) - \frac{1}{2}d(s_3)\right) \\ &= 6\left(0 - \frac{1}{2}\cdot 1\right) + \frac{1}{2}\cdot 6\left(1 - \frac{1}{2}\cdot 0\right) \\ &= -3 + 3 \qquad\qquad = 0. \end{aligned}$$

What this means is that total effect of switching which square we exit from by F_1 is 0, so F_1 actually makes no difference. Intuitively, because of our discount factor $\gamma = \frac{1}{2}$, every negative reward will cancel out with the positive rewards, and all but the first and last terms in all the F_1 rewards added together will cancel out. When we are determining the optimal policy, we must compare the actions we can take at state s. $d(s)$ will be the first term in the sum of F_1 rewards, and the last term will always be 0 since we exit from a numbered square, so F_1 makes no difference as to the optimal action.

There is a more rigorous proof that F_1 will not change optimal policies that is beyond the scope of the course. You can also confirm that this policy is optimal in this particular maze by evaluating this policy in the modified environment and confirming that those values satisfy the Bellman equations.

(c) [1 pt] If the robot now executes this policy π in the **original** environment, without the extra added rewards F, what is $V^\pi(s)$ where s is the top-most state?

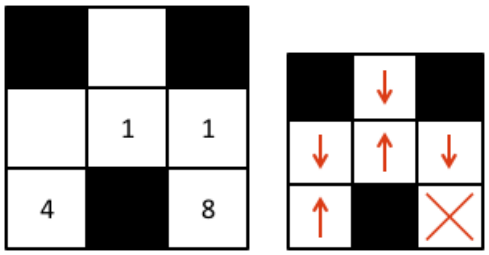
The robot takes three steps before it can exit and gather the reward of 8, so the value of the top-most state is $\gamma^3 \cdot 8 = \boxed{1}$.

(d) [3 pts] Consider a different artificial reward that also favors moving toward numbered squares in a slightly different way:

$$F_2(s, a, s') = \begin{cases} 6 & d(s') < d(s) \text{ i.e. } s' \text{ is closer to a numbered square than } s \text{ is,} \\ 0 & d(s') \geq d(s). \end{cases}$$

F_2 is always 0 when s' is a terminal state (equivalently, when a is the Exit action).

Fill in the diagram on the right as in (a) to indicate the optimal policy PacBot will calculate with the discount factor $\gamma = 0.5$ and the modified reward function $R_2'(s, a, s') = R(s, a, s') + F_2(s, a, s')$ in the board on the left. Break ties in the following order: Up, Down, Right, Left, Exit.



Unlike F_1, F_2 actually encourages cycles. Since there are no penalties for moving away from numbered squares here as there were previously, the agent can accrue large positive rewards by stepping on and off a numbered square repeatedly. Specifically, for stepping on a numbered square, it gets reward 6, then if it steps off and on again, it gets reward $\gamma^2 \cdot 6 = 3/2$. This cycle gives a geometric sum worth $\frac{6}{1-\gamma^2} = \frac{6}{3/4} = 8$. We consider then the value of being on a non-numbered square to be 8, and we can apply the same reasoning as in (a) to determine the optimal policy. If the agent is closer to where it can exit for 8 points, it moves toward that. If it is closer to the cycle where it can accrue F_2 rewards worth 8, it moves toward that cycle. (Note that $F_2 = 0$ when moving between two different numbered squares.)

(e) [1 pt] If the robot now executes this policy π in the **original** environment, without the extra added rewards F, what is $V^{\pi}(s)$ where s is the top-most state?

In any policy that is optimal in the altered environment, the agent will get stuck in a cycle if it starts from the top state, meaning it will never exit. In the original environment, exiting is the only way to get any non-zero reward, so it will get no reward and the value is $\boxed{0}$.

(f) [4 pts] For each of the following conditions on $F(s, a, s')$, state whether the condition is necessary and/or sufficient for the set of optimal policies to be unchanged in a general Gridworld-like MDP (i.e. an MDP following the rules laid out at the beginning of this question, but with any arbitrary board configuration) by adding F to the reward function. Assume $\gamma = 1$, all states are reachable from all non-terminal states, and there is at least one positive number on the board. Note that the set of optimal policies is unchanged between a pair of MDPs when a policy is optimal in one MDP if and only if it is also optimal in the other.

(i) [1 pt] Condition 1: If M is the maximum number on the board, then in the modified MDP, the set of all optimal policies is all policies such that no matter where you start, you will exit from a square showing M.

■ necessary ■ sufficient ☐ neither

This is equivalent to optimal policies being unchanged because this is a full description of optimal policies in the *unmodified* MDP. Any square is reachable from any other, and since we are dealing with the undiscounted case, if we are at a square showing less than M, it is always better to move to a square showing M and exit than to exit from the current square.

(ii) [1 pt] Condition 2: If M is the maximum number on the board, $|F(s, a, s')| \leq M$ for all s, a, s' with s' not a terminal state.

☐ necessary ☐ sufficient ■ neither

For a counterexample for necessity, consider $F(s, a, s') = k(d(s) - d(s'))$, a function similar to F_1. For any k, the arguments in the solution to (b) still hold, and so for arbitrarily large additive rewards, we can maintain optimal policies. For a counterexample for sufficiency, use the board from part (a), use $\gamma = 1$, and let $F(s, a, s')$ be 1 everywhere. Condition (iv) is fulfilled, since $|1| < 8$, but the policy in the modified environment will never exit since it can move back and forth between squares to get infinite reward.

(iii) [1 pt] Condition 3: The value function is unchanged; that is, $V'(s) = V(s)$ for all s, where V' is the value function in the modified environment.

☐ necessary ☐ sufficient ■ neither

This is not necessary, because we can design some additive reward that increases all of the values by 1 (for example, by making $F = 1$ for exiting from the highest numbered square), and we will still extract the same policy. It is harder to see that it is not sufficient. Suppose this condition is fulfilled and all the values are unchanged. To extract a policy for V values, we still have to do a one-step lookahead, which allows the additive rewards to add some influence. Suppose for a concrete counterexample that we use the board from (a) and have a reward of 1 added for exiting from the top-most state. The value of that state is still 1 as it was in the unmodified environment (see part (c)), but we have added a new optimal policy: exiting from that square now also achieves optimal value.

(iv) [1 pt] Condition 4: The Q-value function is unchanged; that is, $Q'(s, a) = Q(s, a)$ for all s and a, where Q' is the Q-value function in the modified environment.

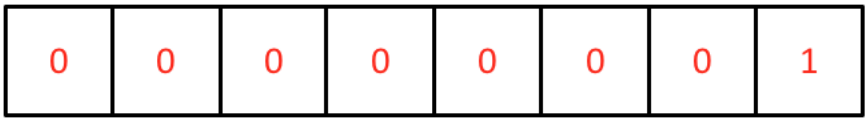
☐ necessary ■ sufficient ☐ neither

This is not necessary for the same reason that the previous condition is not necessary. However, it is sufficient, because extracting a policy from Q-values only requires taking an argmax, rather than doing the one-step lookahead. The Q-values explicitly designate a policy, making this a stronger condition than unchanged V-values that is actually sufficient.

Consider the following new Gridworld-like environment consisting of 8 states all in a row with all squares blank except the rightmost, which shows a 1. We restrict actions to Right, Left, and Exit. (The rules are exactly the same as in (a), except that the Up and Down actions are no longer available).

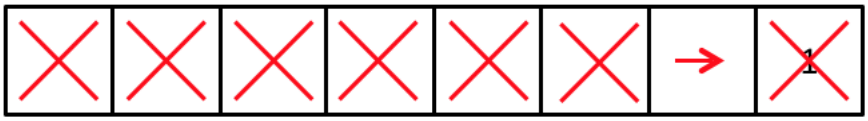
(g) In this environment, initialize a policy π_0 in which we exit at every state.

(i) [1 pt] Fill in the following diagram with values for V^{π_0}, the result of running policy evaluation with $\gamma = 1$. (For example, if the leftmost square has value 4 when following this policy, write a 4 in that square on the diagram. Note that this is the same MDP as above, but the 1 has been removed for your convenience. The robot still receives a reward of 1 for exiting from the rightmost square.)



The value of exiting from each state is 0 for a non-numbered square and 1 on the square showing a 1.

(ii) [1 pt] Let π_1 be the new policy after one step of policy improvement. Break ties in the following order: Exit, Left, Right. As in (a), draw an arrow or an X in each square to represent this policy.



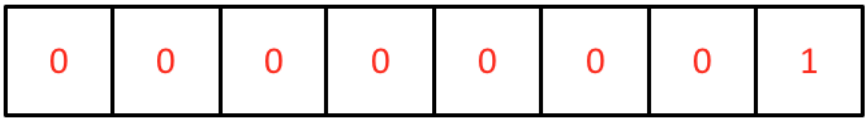
For the rightmost square, the optimal action is to exit, giving reward 1. From the square immediately to the left of that, the optimal action is to move to the right, since the rightmost square has value 1. As seen in the previous answer, with the current values, from all other squares, all actions give reward zero and sum of future rewards equal to 0, so all actions are equal and we go with the action that takes precedence in the tiebreaking order: Exit.

(iii) [1 pt] How many iterations of policy iteration are necessary to converge on the optimal policy? 7 iterations; after each one, one more state will switch to moving right instead of exiting. Some students considered the iteration in which we know we've converged, after we've seen the same policy twice, so credit was also awarded for the answer 8.

(h) We now reintroduce the extra reward $F_1(s, a, s') = 6\left(d(s) - \frac{1}{2}d(s')\right)$ from part (b). For the rest of this question, we are working in a modified version of the long environment above where the reward function is $R'(s, a, s') = R(s, a, s') + F_1(s, a, s')$.

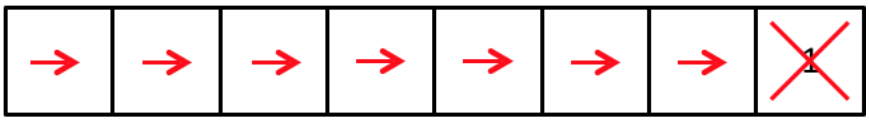
Once again, initialize a policy π_0 in which we exit at every state.

(i) [1 pt] As in (g), fill in the following diagram with values for V^{π_0}, the result of running policy evaluation with $\gamma = \frac{1}{2}$ in the **modified** environment.



Since the added rewards are zero for all exit transitions, for π_0, the added rewards make no difference whatsoever.

(ii) [1 pt] Let π_1 be the new policy after one step of policy improvement in the **modified** environment. Break ties in the following order: Stop, Left, Right. As in (a), draw an arrow or an X in each square to represent this policy.



Because policy improvement takes the action that maximizes $R'(s, a, s') + \gamma V(s') = R(s, a, s') + F_1(s, a, s') + \gamma V(s')$ (since transitions are deterministic) and moving right from all squares but the rightmost gives positive F_1 value, the action chosen at all of those states is to move right.

(iii) [1 pt] How many iterations of policy iteration are necessary to converge on the optimal policy? Only one iteration is necessary. The policy described in part (ii) is optimal. Some students considered the iteration in which we know we've converged, after we've seen the same policy twice, so credit was also awarded for the answer 2.

There was a typo in the original version of the exam in which $\gamma = 1$ for part (h). Most students gave the intended answer, but credit was also awarded to students who answered the question correctly with $\gamma = 1$.

Q7. [17 pts] Spinaroo

A casino considers adding the game Spinaroo to their collection, but needs you to analyze it before releasing on their floor. The game starts by the dealer rolling a 4-sided die, which can take on values $\{1, 2, 3, 4\}$ with equal probability. You get to observe this rolled value, D (for dealer). You are then given a separate 2-sided die, which can take on values $\{1, 2\}$ with equal probability. You are initially forced to roll this die once and observe its value G (for gambler). At this point, you can choose whether to continue rolling or to stop. Each time you roll the die, the observed value gets added to the cumulative sum G. Once you stop, the game ends. If the cumulative sum $G < D$, you lose 1 dollar. If $G = D$, you neither win nor lose money. If $D < G < 5$, you win 1 dollar. If $G \geq 5$, you lose 1 dollar.

You decide to model this game via a Markov Decision Process (MDP). You model the states as tuples $(d, g) \in \{1, 2, 3, 4\} \times \{1, 2, 3, 4, \mathit{Bust}\}$, where d denotes the dealer's roll and g denotes the current cumulative sum. In particular, we set g to *Bust* when it is 5 or higher. After a players first forced roll, their available actions are *Roll* and *Stop*. The reward, the amount of money they win, is awarded once they *Stop*, transitioning them to the *End* state. The discount factor is 1.

(a) [5 pts] You first consider policy iteration in solving this problem. The initial policy π is in the table below. For example, the initial policy prescribes *Roll* in the state $(a, b) = (3, 2)$.

	$d = 1$	$d = 2$	$d = 3$	$d = 4$
$g = 1$	*Roll*	*Roll*	*Stop*	*Roll*
$g = 2$	*Stop*	*Stop*	*Stop*	*Stop*
$g = 3$	*Stop*	*Stop*	*Roll*	*Roll*
$g = 4$	*Stop*	*Stop*	*Stop*	*Stop*

Fill in the following table, denoting V^π. Some values have been filled in for you.

	$d = 1$	$d = 2$	$d = 3$	$d = 4$
$g = 1$	1	$\frac{1}{2}$	-1	$-\frac{3}{4}$
$g = 2$	1	0	-1	-1
$g = 3$	1	1	0	$-\frac{1}{2}$
$g = 4$	1	1	1	0

(b) [4 pts] At some point of time during policy iteration, you notice that V^{π} is as follows:

	$d=1$	$d=2$	$d=3$	$d=4$
$g=1$	1	1	$\frac{1}{4}$	$-\frac{3}{8}$
$g=2$	1	1	$\frac{1}{2}$	$-\frac{1}{4}$
$g=3$	1	1	0	$-\frac{1}{2}$
$g=4$	1	1	1	0

where π was:

	$d=1$	$d=2$	$d=3$	$d=4$
$g=1$	*Roll*	*Roll*	*Roll*	*Roll*
$g=2$	*Stop*	*Roll*	*Roll*	*Roll*
$g=3$	*Stop*	*Stop*	*Roll*	*Roll*
$g=4$	*Stop*	*Stop*	*Stop*	*Stop*

Perform a policy improvement step to extract policy π'. In the case where both Roll and Stop are acceptable updates, write Roll/Stop. Parts of the policy have been filled in:

	$d=1$	$d=2$	$d=3$	$d=4$
$g=1$	*Roll*	*Roll*	*Roll*	*Roll*
$g=2$	*Stop*	*Roll*	*Roll*	*Roll*
$g=3$	*Stop*	*Stop*	*Roll/Stop*	*Roll*
$g=4$	*Stop*	*Stop*	*Stop*	*Stop*

(c) [3 pts] Is the policy, π, given from the previous part optimal? Note that we're asking about the optimality of policy π and not π'. From the information we have, can we deduce whether policy iteration has converged? Briefly explain your reasoning.

Policy iteration has converged, since the set of policies we consider updating contains π. π is optimal, since policy iteration converges if and only if policy iteration yields an optimal policy.

(d) [2 pts] Spinaroo is released to the floor, and it's doing very well! The casino is concerned that players might determine the optimal policy, so they decide to use weighted dice (for both the 2 and 4-sided dice), where the weights are unknown.

The casino wants you to determine an optimal policy for this Spinaroo variant. You decide to use Q-Learning to solve this problem, where states, actions, and rewards are modeled as in the previous questions. Select all strategies that will guarantee that a certain player learns the optimal Q-values.

☐ Strategy 1: During learning, the placer acts according to the optimal policy π: namely, the policy where $\pi(s) = \text{argmax}_a Q(s, a)$. Learning continues until convergence.

☐ Strategy 2: During learning, the player acts according to the pessimal policy π: namely, the policy where $\pi(s) = \text{argmin}_a Q(s, a)$. Learning continues until convergence.

■ Strategy 3: During learning, the player chooses *Roll* and *Stop* at random. Learning continues until convergence.

☐ Strategy 4: During learning, the player chooses *Roll* and *Stop* at random. Learning continues until each state-action pair has been seen at least 20 times.

■ Strategy 5: During learning, in a state s, the player chooses the action $a \in \{Roll, Stop\}$ that the placer has chosen least often in s, breaking ties randomly. Learning continues until convergence.

(e) [3 pts] Your manager is proud of being an indecisive person. As such, you decide to impress your manager by devising indecisive learning strategies. Each choice X, Y below corresponds to the following strategy: when asked to take an action, choose the action prescribed by strategy X with probability $0 < \epsilon < 1$ and the action prescribed by strategy Y with probability $1 - \epsilon$. Refer to the previous part for the names of the strategies. Which of the following indecisive strategies will lead to learning the optimal Q-values? Note: in the case where strategy 4 is used, learning continues until each state-action pair has been seen at least 20 times, and not until convergence.

Hint: consider the number of actions that are available.

■ $1, 2$
■ $1, 3$
☐ $1, 4$
■ $1, 5$
■ $2, 3$
☐ $2, 4$
■ $2, 5$
■ $3, 5$
☐ $3, 4$
☐ $4, 5$

THIS PAGE IS INTENTIONALLY LEFT BLANK