

- You have 110 minutes.
- The exam is closed book, no calculator, and closed notes, other than two double-sided cheat sheets that you may reference.
- Anything you write outside the answer boxes or you ~~cross-out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation.

For questions with **circular bubbles**, you may select only one choice.

- Unselected option (completely unfilled)
- Only one selected option (completely filled)
- Don't do this (it will be graded as incorrect)

For questions with **square checkboxes**, you may select one or more choices.

- You can select
- multiple squares (completely filled)

| | |
|-----------------------------|--|
| First name | |
| Last name | |
| SID | |
| Name of person to the right | |
| Name of person to the left | |
| Discussion TAs (or None) | |

Honor code: “As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others.”

By signing below, I affirm that all work on this exam is my own work, and honestly reflects my own understanding of the course material. I have not referenced any outside materials (other than my cheat sheets), nor collaborated with any other human being on this exam. I understand that if the exam proctor catches me cheating on the exam, that I may face the penalty of an automatic "F" grade in this class and a referral to the Center for Student Conduct.

Signature: _____

Point Distribution

| | |
|---|-----|
| Q1. Potpourri | 14 |
| Q2. Search: Dwinelle Hall Maze | 17 |
| Q3. CSPs: Splitting CSPs on a Constraint | 22 |
| Q4. Multi-Agent Search: Faulty Functions | 16 |
| Q5. MDPs: One Piece, Zero Reward | 15 |
| Q6. Reinforcement Learning: Clumsy Test Taker | 16 |
| Total | 100 |

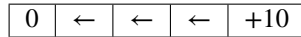
Q1. [14 pts] Potpourri

- (a) [1 pt] True or false: Local search has no optimality guarantees, so we always prefer to use A* search over local search.
- True False
- (b) [1 pt] True or false: In Monte Carlo tree search, all else equal, performing more rollouts from a state will generally result in a better estimate for the value of that state.
- True False
- (c) [1 pt] True or false: When performing a rollout in Monte Carlo tree search, we must play moves according to the optimal policy.
- True False

In the next two subparts, consider the grid world below. Reminder: in a grid world, there is only one action, “Exit”, available from the squares labeled +10 and 0, that gives the rewards of +10 and 0, respectively.

Assumptions: The living reward for every action is 0. All actions succeed with probability 1.0. The discount factor $\gamma = 1$.

We want to run policy iteration, starting at the policy shown below. In case of ties, we will choose left instead of right.



- (d) [2 pts] What is the resulting policy after one iteration of policy iteration?
- | | | | | |
|---|---|---|---|-----|
| 0 | ← | ← | ← | +10 |
|---|---|---|---|-----|

| | | | | |
|---|---|---|---|-----|
| 0 | → | ← | ← | +10 |
|---|---|---|---|-----|
- | | | | | |
|---|---|---|---|-----|
| 0 | ← | ← | → | +10 |
|---|---|---|---|-----|

| | | | | |
|---|---|---|---|-----|
| 0 | → | → | ← | +10 |
|---|---|---|---|-----|
- (e) [2 pts] Starting at the initial policy shown above (always go left), how many iterations of policy iteration are needed to compute the optimal policy?
- In other words, if π_0 is the initial policy, select the minimum k such that $\pi_k = \pi^*$.
- 1 3 More than 4
 2 4

In the next two subparts, consider an agent performing Q-learning using one of these exploration strategies:

- Strategy A: Epsilon-greedy, where ϵ is set to a fixed value. You can assume ϵ , the probability of choosing a random action, is set such that $0.5 < \epsilon < 1$.
- Strategy B: Epsilon-greedy, where ϵ starts at the same fixed value, but decreases over time.


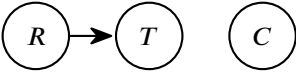
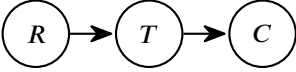
- (f) [1 pt] What happens if we run Strategy A forever?
- We prioritize exploration forever, and never exploit what we’ve learned.
 We prioritize exploiting what we know forever, and never explore new states.
 We make random moves every time.
- (g) [1 pt] Which strategy will result in higher regret if we act according to that strategy forever?

Hint: Regret is the absolute difference between the expected reward for following the optimal policy, and the expected reward for acting according to the given strategy.

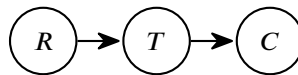
- Strategy A The regret is the same
 Strategy B

In the next two subparts, consider the following scenario: Rain (R) causes traffic (T). Rain and traffic have no effect on cavity (C).

(h) [3 pts] Select all of the Bayes' nets that can represent a joint distribution with the assertions given above.

- 
- 
- 
- None of the above

(i) [2 pts] In this subpart, consider the following Bayes' net:



Suppose R and C are binary random variables (each variable has two possible values), and T is a random variable with three possible values.

Select all true statements about the sizes of the conditional probability tables (CPTs) associated with nodes in the Bayes' nets shown above.

- Every CPT has the same number of entries.
- The CPT associated with C has 6 entries.
- None of the above

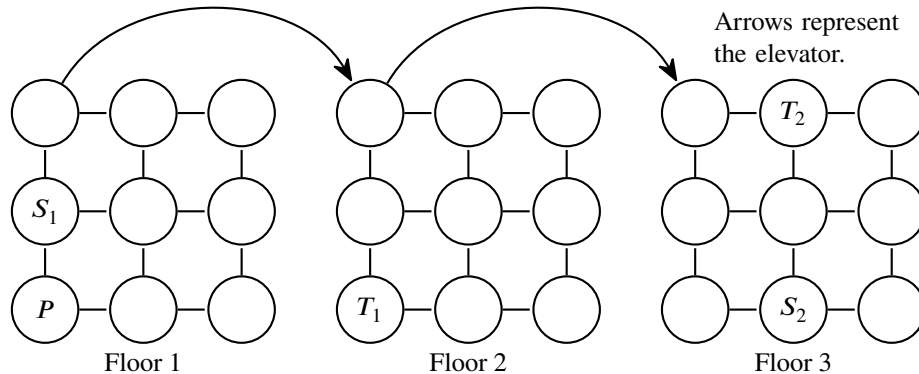
Q2. [17 pts] Search: Dwinelle Hall Maze

Some of Pacman's friends are lost in Dwinelle Hall and need help to make it to their classes!

Consider a 3-dimensional $M \times N \times F$ grid. You can think of this grid as F 2-dimensional $M \times N$ grids stacked on top of each other (where each 2-dimensional grid represents one floor of Dwinelle Hall).

Pacman (labeled P in the diagram) always starts at the bottom-left square of Floor 1. Pacman can take these actions in the grid (all actions have cost 1):

- Move to an adjacent square on the same floor (north, south, east, or west).
- From the top-left corner in any floor, take an elevator to the top-left corner of the floor directly above. The elevator cannot go down.



An example search graph is shown above, with $F = 3$ floors, and a $M \times N = 3 \times 3$ grid on each floor. **Your answers in this question should work for any arbitrary $M \times N \times F$ grid**, not just the example shown.

There are k Pacfriends, and each Pacfriend has a starting and target location (not necessarily unique). In the example above, we've shown two Pacfriends, with starting locations labeled S_1, S_2 , and target locations labeled T_1, T_2 .

Pacman's goal is to escort each Pacfriend from their starting location to their target location.

- When Pacman enters a square with a Pacfriend, Pacman automatically "picks up" the Pacfriend, who will now start to share Pacman's location as Pacman takes actions.
- When Pacman enters a Pacfriend's target location, Pacman automatically "drops off" the Pacfriend, who will stay at the target location.
- Pacman can be escorting (have picked up, but not dropped off) multiple Pacfriends at the same time.

(a) [2 pts] What is the maximum branching factor in this search problem?

(b) [2 pts] Select all true statements about this search problem.

- Depth-first tree search is guaranteed to find a solution, or report that no solutions exist.
- Breadth-first tree search will expand at least as many nodes as breadth-first graph search.
- None of the above

(c) [2 pts] In this subpart, suppose that all Pacfriends' starting and target locations are on Floor 1. Select all true statements about this specific problem.

- Depth-first graph search will not explore any nodes on higher floors.
- Breadth-first graph search will not explore any nodes on higher floors.
- None of the above

(d) [4 pts] Select each state space representation (not necessarily the most efficient representation) that could be used to model this search problem.

Each answer choice is a different state space representation.

- Locations of Pacman and every Pacfriend
- Pacman location, and one Boolean variable for each Pacfriend
- Pacman location, and two Boolean variables for each Pacfriend
- Location of every Pacfriend
- None of the above

(e) [3 pts] Consider the state space representation: Pacman location, and 5 Boolean variables for each Pacfriend.

(Note: this may or may not be a correct or minimal state space representation.)

How many possible states exist in this representation? Your answer could be in terms of M , N , F , k .

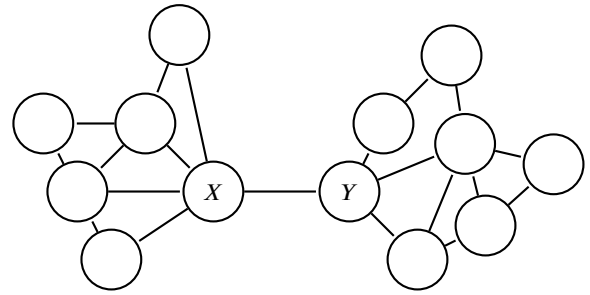
(f) [4 pts] Select all admissible heuristics for this problem.

- Minimum distance between any Pacfriend's current location and their target location
- Maximum distance between any Pacfriend's current location and their target location
- Number of the highest floor, minus number of the floor Pacman is currently on
- Maximum difference between any Pacfriend's current floor and their target location's floor
- None of the above

Q3. [22 pts] CSPs: Splitting CSPs on a Constraint

Consider a CSP with $L + R$ variables. There are d possible assignments to each variable. The CSP only has binary constraints.

We notice that if we remove one of the binary constraints, then the CSP can be decomposed into two independent subproblems, with no constraints between the subproblems: a “left” CSP with L variables and a “right” CSP with R variables. Let X and Y be the variables in the left and right subproblems, respectively, connected by the constraint we removed.



An example is shown to the right (but your answers should not assume we’re using this example). If we remove the binary constraint relating X and Y , this CSP can be decomposed into two independent subproblems. The “left” CSP includes X and has 6 variables. The “right” CSP includes Y and has 7 variables.

- (a) [2 pts] Without removing any constraints, we assign a value to one of the variables on the left side of the CSP. Then, we run forward checking.

What is the maximum number of variables whose domains could change as a result of running forward checking? (Do not count the variable whose value was just assigned.)

- 0
 1
 L
 R
 $L + R - 1$

For the next two subparts: Without removing any constraints, we assign a value to one of the variables on the left side of the CSP. Then, we enforce arc consistency on the entire CSP.

- (b) [3 pts] Select all true statements.

- If the variable we select is not X , the domains of all variables on the right side of the CSP will be unchanged.
 If the variable we select is X , the domains of all variables on the right side of the CSP will be unchanged.
 After enforcing arc consistency, it is guaranteed that we can find a solution to the CSP without performing any more backtracking search.
 None of the above

- (c) [2 pts] If we use the AC3 algorithm to enforce arc consistency, what is the maximum number of arcs we would need to place on the queue at the beginning of the algorithm?

- L
 $2(L + R)^2 + 2$
 R
 $L(L - 1) + R(R - 1) + 2$
 $2(L + R) + 2$
 $L^2 + R^2 + 1$

- (d) [2 pts] Suppose we solve this CSP using backtracking search with no improvements (no filtering, using arbitrary ordering, and no consideration of the CSP structure).

In the worst case, how many assignments do we need to check in order to solve this CSP?

Your expression could be in terms of L , R , and d .

In the next three subparts, we will use an idea similar to cutset conditioning to solve this CSP more efficiently.

(e) [1 pt] First, we assign these variable(s) in all possible ways:

- X
- All variables directly connected to X and Y by a constraint
- All variables in the smaller side of the CSP
- All variables in the larger side of the CSP

(f) [2 pts] Next, for each assignment, solve the residual CSP (with the assigned variable(s) removed).

What is the runtime to solve each residual CSP, and why?

- $O((L + R - 1)d^2)$, because the residual CSP is tree-structured.
- $O(d^{L-1}d^R)$, because the residual CSP is tree-structured.
- $O(d^{L-1} + d^R)$, because the residual CSP has independent subproblems.
- $O(d^{L-1}d^R)$, because the residual CSP has independent subproblems.

(g) [1 pt] Finally, how do we output a solution to the original CSP?

- Find one solution to one residual CSP.
- Find all solutions to one residual CSP.
- Find one solution to all residual CSPs.
- Find all solutions to all residual CSPs.

In the rest of the question, we will consider a different algorithm that uses the structure of the CSP to solve it more efficiently.

First, we'll remove the binary constraint to split the CSP into two independent subproblems.

(h) [2 pts] How should we solve the remaining independent subproblems?

- Find one solution to each CSP.
- Find one solution to the smaller CSP, and all solutions to the larger CSP.
- Find all solutions to the smaller CSP, and one solution to the larger CSP.
- Find all solutions to each CSP.

(i) [1 pt] Suppose we have a solution to the left subproblem, and a solution to the right subproblem. Which variables do we need to consider to check whether the solutions can be combined into a solution of the original CSP?

- All $L + R$ variables
- Only X and Y
- All variables in the smaller subproblem
- All variables connected to X and Y (including X and Y)

Finally, how do we solve the original CSP after solving the independent subproblems? Fill in the details of the algorithm described below.

- Create a new hash map (e.g. Python dictionary) for the left CSP.
- For each solution we found to the left CSP (possibly only one): add a key-value pair to the left subproblem's hash map.
- Create a second new hash map for the right CSP.
- For each solution we found to the right CSP (possibly only one): add a key-value pair to the right subproblem's hash map.
- Perform a nested iteration: for each key of the left hash map, iterate through all the keys of the right hash map.

(j) [2 pts] What are the key-value pairs we add to the left hash map?

- | | | |
|-----------------------|---|---|
| <input type="radio"/> | Key: The solution we found. | Value: The assignment to X in the solution. |
| <input type="radio"/> | Key: The assignment to X in the solution. | Value: The solution we found. |
| <input type="radio"/> | Key: The solution we found. | Value: 1. |
| <input type="radio"/> | Key: 1. | Value: The solution we found. |

(k) [2 pts] After creating both hash maps, we run a nested iteration through the keys of the left and right hash maps.

For left hash map key x and right hash map key y : which of these conditions tells us that we've found a solution to the original CSP?

- x and y are equal.
- x and y are not equal.
- x and y violate a constraint in the original CSP.
- x and y don't violate a constraint in the original CSP.

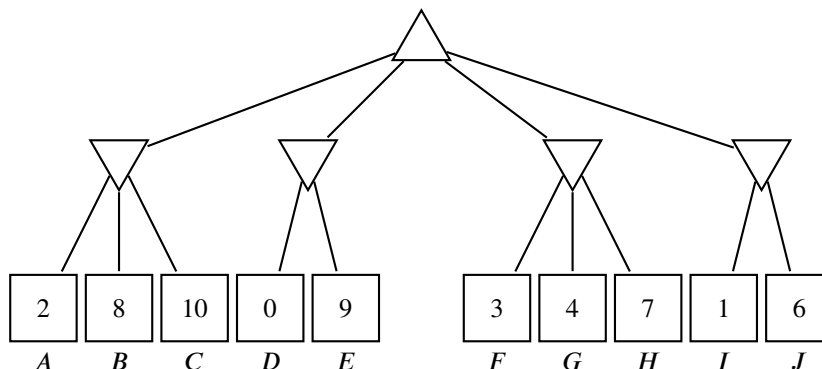
(l) [2 pts] What is a tight big-O bound on the runtime of the nested iteration through the keys of the left and right hash maps?

Your expression could be in terms of L , R , and d (you may not need all variables).

Q4. [16 pts] Multi-Agent Search: Faulty Functions

Pacman is in a two-player, zero-sum game, where the players can take turns choosing actions many times before the game ends.

To choose his next action, Pacman runs a depth-limited minimax search, resulting in the game tree below. Upwards-pointing triangles represent maximizer nodes, and downward-pointing triangles represent minimizer nodes.



(a) [1 pt] What is the value at the root node of the tree?

(b) [3 pts] Select all branches that would **not** be explored due to alpha-beta pruning, assuming we explore from left to right. (Hint: A **does not** get selected, because when we run alpha-beta pruning, node A does get explored.)

- A
 B
 C
 D
 E
 F
 G
 H
 I
 J

Pacman learns that he might have used a faulty evaluation function that sometimes outputs a value 5 greater than the intended value. Formally, if the intended evaluation function is $f(x)$, then the faulty evaluation function $f'(x)$ is defined as:

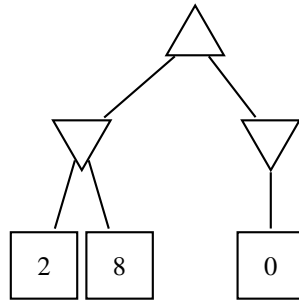
$$f'(x) = \begin{cases} f(x) & \text{with probability 0.5} \\ f(x) + 5 & \text{with probability 0.5} \end{cases}$$

(c) [4 pts] Select all true statements. Do not make any assumptions about the original evaluation function $f(x)$ (i.e. it can be any arbitrary evaluation function).

Note: "True minimax value" refers to the minimax value that would be computed with no depth limit.

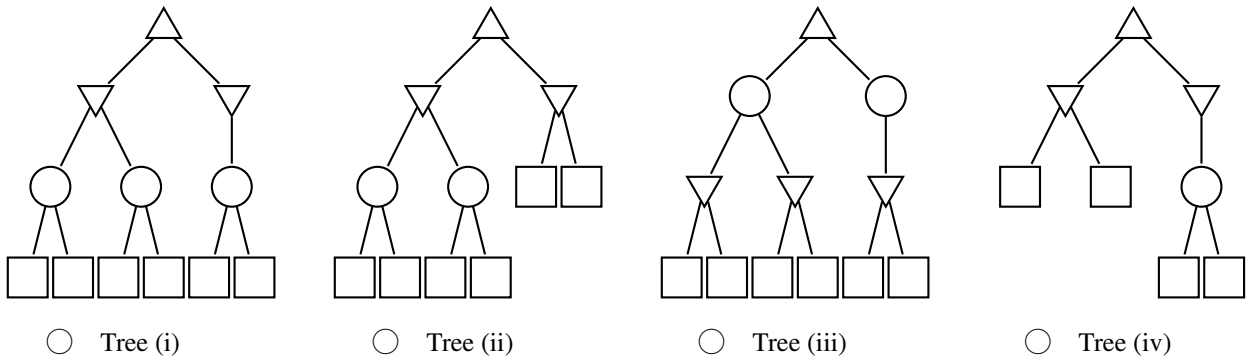
- If the evaluation function is not faulty, Pacman is guaranteed to compute the true minimax value at the root node.
- If the evaluation function is not faulty, Pacman is guaranteed to select the optimal action at the root node.
- If the evaluation function is faulty, the value computed at the root node is guaranteed to be at most 5 away from the true minimax value.
- If the evaluation function is faulty, Pacman is guaranteed to select the optimal action at the root node.
- None of the above

For the rest of the question, assume that Pacman's depth-limited minimax search instead results in the game tree below, and that Pacman is certain that the values in the tree were outputted by the faulty evaluation function.



We would like to draw a new tree that models the situation: Pacman runs depth-limited minimax search, but Pacman does not know if the values computed by the faulty evaluation function are accurate, or 5 higher than intended.

(d) [2 pts] Which of the following modified trees best models the new situation? Circles represent chance nodes.



(e) [2 pts] Which values should exist in the terminal nodes of the modified game tree?

- | | |
|---|--|
| <input type="radio"/> 0, 2, 5, 8 | <input type="radio"/> -5, -3, 0, 2, 3, 8 |
| <input type="radio"/> -5, 0, 2, 8 | <input type="radio"/> 0, 2, 2.5, 3.5, 6.5, 8 |
| <input type="radio"/> 0, 2, 7, 5, 8, 13 | <input type="radio"/> -2.5, -1.5, 0, 1.5, 2, 8 |

(f) [3 pts] Is it possible to model the same situation by only changing the values at terminal nodes, without modifying the structure of the game tree?

- Yes
 No

If you answered yes, write the values that you would use to replace the terminal node values 0, 2 and 8, in that order. (Your answer should be a list of 3 numbers.)

If you answered no, write a brief justification (one sentence is sufficient).

(g) [1 pt] Now, suppose that the faulty function outputs $f(x) + 5$ with probability 0.2, and $f(x)$ with probability 0.8.

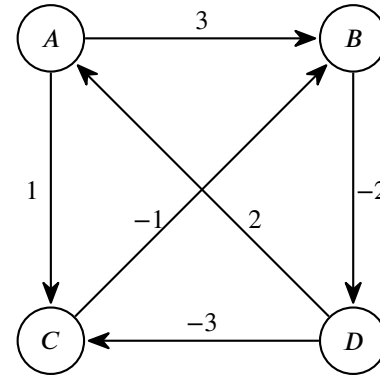
Is it still possible to draw a new game tree that models the situation?

- Yes
 No

Q5. [15 pts] MDPs: One Piece, Zero Reward

Luffy is trying to find the One Piece by taking actions in the MDP shown to the right.

- From A and D , there are two possible actions (going to two possible successor states). With probability 0.8, the intended successor state is reached. With probability 0.2, the wrong successor state is reached.
- The reward only depends on the state and successor state, not the action. For example, transitioning from A to B receives reward 3, regardless of whether Luffy successfully took action $A \rightarrow B$, or unsuccessfully took action $A \rightarrow C$.
- From B and C , only one action is available, and it always succeeds.



For this entire question, assume that $\gamma = 1$.

(a) [4 pts] Select all true statements.

- $Q(A, A \rightarrow B) > Q(A, A \rightarrow C)$ because the reward for action $A \rightarrow B$ is greater than the reward for action $A \rightarrow C$.
- $V_1(C) = -1$
- $V_1(A) = (0.8 \cdot 3) + (0.2 \cdot 1)$
- $V_1(B) = 0$
- None of the above

In the rest of the question, consider a modification to this scenario. At every time step, Luffy keeps track of the total cumulative reward accumulated from all actions so far.

If Luffy ever takes an action that causes his total reward to become negative, then Luffy transitions to a terminal state called *End*. There are no further actions or rewards available from this *End* state.

(b) [2 pts] Consider modifying any MDP (not necessarily the one shown above) such that the game ends if the total reward becomes negative.

Luffy suggests that π^* , the optimal policy of the original, unmodified MDP, will also be the optimal policy for this modified problem. Is Luffy correct?

- No, because π^* does not account for total reward and avoiding negative rewards.
- No, because π^* only tells us the values of states, not the optimal actions.
- Yes, because π^* already accounts for total reward and avoiding negative rewards.
- Yes, because even though π^* doesn't account for total reward, π^* maximizes expected rewards, and maximizing reward is always the same as avoiding a negative total reward.

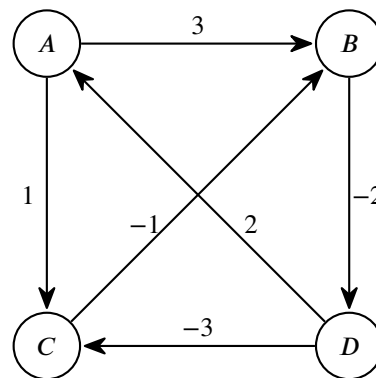
(c) [1 pt] In a normal MDP, we have a table of values $V(s)$, one value for every state.

To model this modified situation, we will create a table of values $V(s, x)$. What does x represent?

- $x = 0$ if Luffy's total reward is negative, and $x = 1$ otherwise.
- The total reward Luffy has accumulated so far.
- The previous state Luffy was in.
- Luffy's starting state.

The question is reprinted for your convenience below. There is no new information until subpart (d) begins.

- From A and D , there are two possible actions (going to two possible successor states). With probability 0.8, the intended successor state is reached. With probability 0.2, the wrong successor state is reached.
- The reward only depends on the state and successor state, not the action. For example, transitioning from A to B receives reward 3, regardless of whether Luffy successfully took action $A \rightarrow B$, or unsuccessfully took action $A \rightarrow C$.
- From B and C , only one action is available, and it always succeeds.



For this entire question, assume that $\gamma = 1$.

At every time step, Luffy keeps track of the total cumulative reward accumulated from all actions so far.

If Luffy ever takes an action that causes his total reward to become negative, then Luffy transitions to a terminal state called *End*. There are no further actions or rewards available from this *End* state.

(d) [4 pts] Write a modified Bellman equation for this modified MDP.

$$V^*(s, x) = \text{(i) (ii) (iii)} [R(s, s') + \gamma V^*(\text{(iv)}, \text{(v)})]$$

- | | | | | |
|-------|------------------------------------|--------------------------------------|--------------------------------------|--|
| (i) | <input type="radio"/> \max_a | <input type="radio"/> \sum_a | <input type="radio"/> $\max_{s'}$ | <input type="radio"/> $\sum_{s'}$ |
| (ii) | <input type="radio"/> \max_a | <input type="radio"/> \sum_a | <input type="radio"/> $\max_{s'}$ | <input type="radio"/> $\sum_{s'}$ |
| (iii) | <input type="radio"/> $T(s, x, a)$ | <input type="radio"/> $T(s, s')$ | <input type="radio"/> $T(s, a, s')$ | <input type="radio"/> $T(s, x, a, s')$ |
| (iv) | <input type="radio"/> s | <input type="radio"/> s' | <input type="radio"/> x | <input type="radio"/> a |
| (v) | <input type="radio"/> x | <input type="radio"/> $x - R(s, s')$ | <input type="radio"/> $x + R(s, s')$ | <input type="radio"/> $x \cdot R(s, s')$ |

(e) [3 pts] Select all true statements.

- | | |
|--|--|
| <input type="checkbox"/> $V_k(End, -1) = 0$ for all k . | <input type="checkbox"/> $V^*(B, i) = 0$ for all $0 < i < 2$. |
| <input type="checkbox"/> $V^*(End, i) = 0$ for all $i < 0$. | <input type="radio"/> None of the above |

(f) [1 pt] For this subpart only, consider a modification to the scenario. The game now ends if Luffy's total reward is less than -1 (instead of ending when the total reward is less than 0).

Is it still possible to model this modified scenario as an MDP and solve it?

- Yes No

Q6. [16 pts] Reinforcement Learning: Clumsy Test Taker

Pacman is working on an exam with N pages, numbered 1 through N .

From page i , Pacman can try to turn to page $i + 1$ or page $i - 1$, for $2 \leq i \leq N - 1$. On page 1 and page N , Pacman can only try to turn to page 2 and page $N - 1$, respectively.

However, each time Pacman turns a page, Pacman might drop the exam packet and end up on any random page (with equal probability). Pacman does not know the probability of dropping the exam.

(a) [1 pt] What is $|S|$, the size of the state space, in this problem?

Your answer could be in terms of N .

- 0 1 2 N N^2

(b) [1 pt] What is $|A|$, an upper-bound on the number of actions per state, in this problem?

- 0 1 2 N N^2

(c) [4 pts] Select the correct space requirements for each reinforcement learning (RL) algorithm. Select one bubble per row.

Model-based learning $|S|$ $|A|$ $|S| \cdot |A|$ $|S|^2 \cdot |A|$

Direct evaluation $|S|$ $|A|$ $|S| \cdot |A|$ $|S|^2 \cdot |A|$

Temporal difference (TD learning) $|S|$ $|A|$ $|S| \cdot |A|$ $|S|^2 \cdot |A|$

Q-learning $|S|$ $|A|$ $|S| \cdot |A|$ $|S|^2 \cdot |A|$

The rest of the question is independent of the previous subparts.

- (d) [2 pts] When we run TD learning, which samples do we use to update $V^\pi(s)$, for a specific state s ?
- Only samples where the agent is at s , takes an action $\pi(s)$, and lands back in the same state s
 - Only samples where the agent is at s and takes an action $\pi(s)$
 - Only samples where the agent is at some other state s' , takes an action $\pi(s')$, and lands in successor state s
 - All samples, regardless of state and successor state

Consider an arbitrary MDP, with unknown transition and reward models. We observe an agent acting according to policy π .

In standard TD learning, we would process each sample one at a time:

$$V^\pi(s) \leftarrow (1 - \alpha)V^\pi(s) + \alpha [r_i + \gamma V^\pi(s'_i)]$$

- (e) [4 pts] Suppose we want to process k samples in one update instead of k separate updates. Select the most appropriate modified update equation.

Note: The correct update equation will converge to the true value of $V^\pi(s)$, assuming we provide enough samples and set α properly, but you don't need to show this to solve the question.

$$V^\pi(s) \leftarrow (1 - \alpha) \cdot \text{(i)} \cdot \text{(ii)} + \alpha \cdot \text{(iii)} \cdot \text{(iv)} [\text{(v)} + \text{(vi)}]$$

- | | | | | |
|-------|---|--|--|--|
| (i) | <input type="radio"/> 0 | <input type="radio"/> 1 | <input type="radio"/> k | <input type="radio"/> $1/k$ |
| (ii) | <input type="radio"/> $\sum_s V^\pi(s)$ | <input type="radio"/> $\prod_s V^\pi(s)$ | <input type="radio"/> $\max_s V^\pi(s)$ | <input type="radio"/> $V^\pi(s)$ |
| (iii) | <input type="radio"/> 0 | <input type="radio"/> 1 | <input type="radio"/> k | <input type="radio"/> $1/k$ |
| (iv) | <input type="radio"/> $\sum_{i=1}^k$ | <input type="radio"/> $\prod_{i=1}^k$ | <input type="radio"/> $\max_{1 \leq i \leq k}$ | <input type="radio"/> $\min_{1 \leq i \leq k}$ |
| (v) | <input type="radio"/> r_i | <input type="radio"/> γr_i | <input type="radio"/> αr_i | <input type="radio"/> r_i^2 |
| (vi) | <input type="radio"/> $V^\pi(s_i)$ | <input type="radio"/> $V^\pi(s'_i)$ | <input type="radio"/> $\gamma V^\pi(s_i)$ | <input type="radio"/> $\gamma V^\pi(s'_i)$ |

In the modified TD learning algorithm: we wait to receive k samples that would normally cause an update to $V^\pi(s)$, and then we update $V^\pi(s)$ with all k samples in a single update.

- (f) [4 pts] Select all true statements about this modified algorithm. Assume $0 < \alpha < 1$ and $k > 1$.
- Given the same finite set of samples: the values computed with the modified algorithm are always identical to the values computed with the original algorithm.
 - More recent sets of k samples influence the value more than older sets of k samples.
 - Within a single set of k values, more recent samples will influence the value more than older samples.
 - We can learn the optimal policy by storing only $V^\pi(s)$.
 - None of the above