CS 188 Summer 2021 Introduction to Artificial Intelligence Midterm

- You have approximately 110 minutes.
- The exam is open book, open calculator, and open notes. Outside internet use is not allowed.
- For multiple choice questions,
  - ☐ means mark **all options** that apply
  - ○ means mark a **single choice**

First name	
Last name	
SID	

For staff use only:

Q1.	Potpourri	/12
Q2.	Maximizing Different Reward Metrics	/20
Q3.	Q-Learning: Weights Amnesia	/15
Q4.	Search: Algorithms	/10
Q5.	Search: Bowser's Kingdom	/8
Q6.	Caramel Crush	/15
Q7.	Game Trees	/20
	Total	/100

THIS PAGE IS INTENTIONALLY LEFT BLANK

SID: ______________

Q1. [12 pts] Potpourri

(a) [2 pts] Your friend Julia proposes a new algorithm, a modified version of Inference by Enumeration called **mod-IBE**. She first marginalizes out hidden variables, then select entries in the joint table that are consistent with evidence. And then, she normalizes. Is mod-IBE a valid approach for probabilistic inference?

- ◯ Yes, since marginalizing hidden variables doesn't ever affect the rows consistent with evidence.
- [✓ Yes, but not for the reason above.]
- ◯ No, since marginalizing combines entries consistent with evidence with those inconsistent with evidence.
- ◯ No, but not for the reason above.

When marginalizing hidden variables, all rows are affected, so option 1 is incorrect. However, though rows consistent with evidence are affected by the marginalization, it is the same operation whether rows inconsistent with evidence are in the table or not. Dropping entries inconsistent with evidence before or after marginalization doesn't matter.

(b) Consider a MDP with **deterministic transitions**, representing a Golden Bear walking around. S is the set of possible states and A is the set of possible actions. We have all the optimal Q-values $Q(s, a)$ and values $V(s)$, and so can reconstruct an optimal policy.

But unfortunately, Bears behave slightly irrationally. We'll model our Bear as choosing an action a to take from state s with probability $P_Q(a|s)$.

(i) [3 pts] Which of the following options for $P_Q(a|s)$ would result in a valid probability distribution that models Bear's slightly irrational behavior?

- ☐ $\frac{V(s)}{\sum_{a' \in A} Q(s,a')}$
- ☐ $\frac{Q(s,a)}{\sum_{a' \in A} Q(s,a')}$
- ☐ $\frac{\exp V(s)}{\sum_{a' \in A} \exp Q(s,a')}$
- [✓ $\frac{\exp Q(s,a)}{\sum_{a' \in A} \exp Q(s,a')}$]
- ☐ $\frac{|V(s)|}{\sum_{a' \in A} |Q(s,a')|}$
- ☐ $\frac{|Q(s,a)|}{\sum_{a' \in A} |Q(s,a')|}$
- ☐ $\frac{V(s)^2}{\sum_{a' \in A} Q(s,a')^2}$
- ☐ $\frac{Q(s,a)^2}{\sum_{a' \in A} Q(s,a')^2}$
- ☐ None of the above

The first two options aren't valid because values and Q-values can be negative.

$\frac{\exp V(s)}{\sum_{a' \in A} \exp Q(s,a')}$, $\frac{|V(s)|}{\sum_{a' \in A} |Q(s,a')|}$, and $\frac{V(s)^2}{\sum_{a' \in A} Q(s,a')^2}$ aren't options because this is not a valid probability distribution ($P_Q(a|s)$ for all a are equal but don't necessarily sum to 1).

The solution, $\frac{\exp Q(s,a)}{\sum_{a' \in A} \exp Q(s,a)}$, means the Bear has a higher probability of selecting actions with higher Q-values.

(ii) [3 pts] True or False: The modified MDP posed by this problem can be modeled as a vanilla expectimax tree. "Vanilla" is defined as an expectimax tree as seen in lecture: alternating layers of max and expectation nodes, with multiple children for every node.

- ◯ True, since there are alternating turns between deterministic and nondeterministic behavior.
- ◯ True, but not for the reason above.
- ◯ False, since introducing $P_Q(a|s)$ causes the expectimax tree to have two layers of expectation nodes for every max node.
- [✓ False, but not for the reason above.]

The usual expectimax model for MDPs models states as MAX nodes and q-states as expectation nodes. In this setup, transitions are deterministic, so q-states are no longer nodes in a tree (there is only one outcome). The addition of the sub-optimality of the Bear causes the states to be not MAX but expectation nodes. So this is no longer an expectimax tree, but a sequence of coin flips.

(iii) [2 pts] We're interested in our Bear's trajectories. A **trajectory** is a sequence of states and actions $\{s_1, a_1, s_2, a_2..., s_T, a_T, s_{T+1}\}$ which detail an agent's movements within the MDP.

What is the probability of observing a full trajectory $\{s_1, a_1, s_2, a_2..., s_T, a_T, s_{T+1}\}$ given that we know the Bear has already gone through a sub-trajectory $\{s_1, a_1, s_2, a_2..., s_t, a_t, s_{t+1}\}$, $1 <= t < T$?

$$P(\{s_1, a_1, s_2, a_2...s_{T+1}\}|\{s_1, a_1, s_2, a_2...s_{t+1}\}) = ...$$

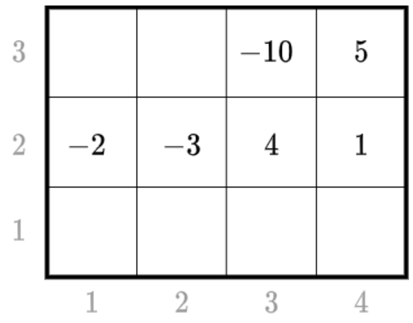
- ☐ $\sum_1^t P_Q(s, a)$
- ☐ $\prod_1^t P_Q(s, a)$
- ☐ $\sum_t^T P_Q(s, a)$
- ☐ $\prod_t^T P_Q(s, a)$
- ☐ $\frac{\sum_1^T P_Q(s,a)}{\sum_1^t P_Q(s,a)}$
- ☐ $\sum_{t+1}^T P_Q(s, a)$
- [✓] $\prod_{t+1}^T P_Q(s, a)$
- ☐ $\frac{\sum_1^T P_Q(s,a)}{\sum_{t+1}^T P_Q(s,a)}$
- ☐ None of the above

(c) [2 pts] Select all of the following algorithms that would require a reinforcement learning agent to construct estimates of transition probabilities and rewards in the course of updating its policy:

- [✓] Model-Based Learning
- [✓] Direct Evaluation
- [✓] Temporal Difference Learning
- ☐ Q-Learning

By definition, Model Based Learning learns T and R. Direct evaluation and TD learning will eventually learn the true value of all states under the policy they follow, but an agent needs the Q-values of states to update the policy. Getting Q-values from values requires some model of T and R.

Q2. [20 pts] Maximizing Different Reward Metrics



Consider a robot navigating in a 3×4 2D grid with walls around the edges as shown above. The robot wants to move around and get high reward, but it needs your help to plan ahead to achieve this!

The robot's state is represented by $(row, column)$, with the starting position at $(1, 1)$. He only has two actions $a \in \{\text{right}, \text{up}\}$. The robot cannot leave the grid. The robot can choose to move into the wall, but its state will stay the same after the action.

Note that in this grid, the transitions described above are *deterministic*. We characterize this MDP's dynamics with a function $f(s, a) = s'$. When the robot transitions into state s, it receives a reward $R(s)$ which depends only on the state it transitions to. $R(s)$ for every cell is shown above. $R(s)$ is defined to be 0 at all grid cells without a numerical label.

For example, the robot can incur a -10 reward by using the up action from $(3, 3)$ without changing its position.

(a) **(i)** [2 pts] What is the equivalent equation for computing optimal values in the MDP defined above in terms of transition function $f(s, a)$ and reward function $R(s)$? Assume no discounting ($\gamma = 1$). Also, define $x@y = max(x, y)$. For each letter **(A)**, **(B)**, **(C)**, **(D)**, **(E)**, fill in a single entry for the term corresponding to the correct equation to implement value iteration for this MDP. Select one bubble per row to form the whole equation.

$$V^*(s) = (\_\mathbf{A}\_)(\_\mathbf{B}\_)\left[R((\_\mathbf{C}\_))(\_\mathbf{D}\_)V^*((\_\mathbf{E}\_))\right] \qquad (1)$$

- **(A):** ○ 1 [✓ $\max_a$] ○ $\mathbb{E}_a$
- **(B):** ○ $\sum_{s'}$ ○ $\sum_a$ [✓ 1]
- **(C):** [✓ $f(s, a)$] ○ $f(s', a)$ ○ $V^*(s)$
- **(D):** ○ $\times$ [✓ $+$] ○ $@$
- **(E):** [✓ $f(s, a)$] ○ $R(s')$ ○ $R(s)$

(ii) [2 pts] What is the optimal value of the starting state when you can at most take N steps in the environment?

$N = 3$? ____4____
$N = 5$? ____10____
$N = 10$? ____35____

For $N = 3$, the optimal value is 4 since the robot can accumulate a total reward of 4 by reaching state $(2, 3)$, at which point its time runs out. For $N = 5$, the optimal value is 10 since the robot can accumulate a total reward of 10 by reaching state $(3, 4)$ through state $(2, 3)$ and state $(2, 4)$, at which point its time runs out. For $N = 10$, the optimal value is $10 + 5 \cdot 5 = 35$ since the robot can accumulate a reward of 10 by reaching state $(3, 4)$ through state $(2, 3)$ and state $(2, 4)$ in the first 5 timesteps, and then accumulate a further reward of $5 \cdot 5 = 25$ by selecting any action in state $(3, 4)$ and repeatedly transitioning back into the same state.

(b) Normally, the optimal value function V^* represents the maximum possible **sum** of future rewards since we typically want to find a policy which maximizes the sum of future rewards in the MDP. However, suppose that we instead want to find a policy which maximizes the future **product** of rewards. Thus, we want to compute V^* such that it represents the maximum possible **product** of future rewards.

(i) [2 pts] Give an expression for a corresponding equation for V^* which will compute this quantity. Define $x@y = max(x, y)$. For each letter **(A)**, **(B)**, **(C)**, **(D)**, **(E)**, fill in a single entry for the term corresponding to the correct equation to implement value iteration to maximize the product of future rewards. Select one bubble per row to form the whole equation.

$$V^*(s) = \textbf{(A)}\textbf{(B)}\left[R((\textbf{C}))\textbf{(D)}V^*((\textbf{E}))\right] \tag{2}$$

- **(A):** ○ 1 [✓ $\max_a$] ○ $\mathbb{E}_a$
- **(B):** ○ $\sum_{s'}$ ○ $\sum_a$ [✓ 1]
- **(C):** [✓ $f(s,a)$] ○ $f(s',a)$ ○ $V^*(s)$
- **(D):** [✓ $\times$] ○ $+$ ○ @
- **(E):** [✓ $f(s,a)$] ○ $R(s')$ ○ $R(s)$

(ii) [2 pts] Given this new objective, what is the optimal value of the starting state when you can at most take N steps in the environment?

$N = 3$? 24

$N = 5$? 2400

$N = 10$? $1.2 \cdot 10^8$

For $N = 3$, the optimal value is 24 since the robot can accumulate a total reward of 24 by reaching state $(2,3)$ at which point its time runs out. For $N = 5$, the optimal value is 2400 since the robot can accumulate a total reward of 2400 by reaching state $(3,3)$ through the middle row, staying there for one extra step, and then time runs out. For $N = 10$, the optimal value is $1.2 \cdot 10^8$. This is achieved by transitioning into state $(3,3)$ via the middle row (resulting in a total product of -240, spending 5 timesteps selecting the up action, resulting in a total product of $2.4 \cdot 10^7$, and finally selecting the right action to transition into state $(3,4)$, resulting in a total product of $1.2 \cdot 10^8$.

(c) Now, suppose that we instead want to find a policy which maximizes the **highest** reward achieved at any timestep. Thus, we want to compute V^* such that it represents the maximum possible value of the **highest** reward the robot ever achieves.

(i) [2 pts] Give an expression for a corresponding equation for V^* which will compute this quantity. Define $x@y = max(x, y)$. For each letter **(A)**, **(B)**, **(C)**, **(D)**, **(E)**, fill in a single entry for the term corresponding to the correct equation to implement value iteration to maximize the **highest** reward achieved by the robot. Select one bubble per row to form the whole equation.

$$V^*(s) = \textbf{(A)}\textbf{(B)}\left[R((\textbf{C}))\textbf{(D)}V^*((\textbf{E}))\right] \tag{3}$$

- **(A):** ○ 1 [✓ $\max_a$] ○ $\mathbb{E}_a$
- **(B):** ○ $\sum_{s'}$ ○ $\sum_a$ [✓ 1]
- **(C):** [✓ $f(s,a)$] ○ $f(s',a)$ ○ $V^*(s)$
- **(D):** ○ $\times$ ○ $+$ [✓ @]
- **(E):** [✓ $f(s,a)$] ○ $R(s')$ ○ $R(s)$

(ii) [2 pts] Given this new objective, what is the optimal value of the starting state when you can at most take N steps in the environment?

$N = 3$? 4

$N = 5$? 5

$N = 10$? 5

For $N = 3$, the highest achievable reward is simply 4 at state $(2,3)$. For $N = 5$, the highest achievable reward is 5 at state $(3,4)$. For $N = 10$, the highest achievable reward is still 5.

(d) The standard Bellman equation provides a condition the optimal policy must satisfy by performing a one-step look-ahead by defining a recursive relationship between $V^*(s)$ and $V^*(s')$, where s' is a possible future next state. However, in general, one could define an analagous update with a two-step lookahead.

(i) [3 pts] Which of the following would be the correct way to formulate the Bellman equation with a two-step lookahead?

$$V^*(s) = (\_\textbf{A}\_)\left[R((\_\textbf{B}\_)) + (\_\textbf{C}\_)\left[R((\_\textbf{D}\_)) + V^*((\_\textbf{E}\_))\right]\right] \tag{4}$$

- **(A)**: ○ $\sum_a$ [✓ $\max_a$] ○ $\mathbb{E}_a$
- **(B)**: ○ s [✓ $f(s,a)$] ○ s'
- **(C)**: ○ $\sum_{a'}$ [✓ $\max_{a'}$] ○ $\mathbb{E}_{a'}$
- **(D)**: ○ $f(s,a')$ [✓ $f(f(s,a),a')$] ○ $f(f(s,a'),a)$
- **(E)**: ○ $f(s,a')$ [✓ $f(f(s,a),a')$] ○ $f(f(s,a'),a)$

(ii) [2 pts] If we perform value iteration using the above two-step Bellman equation, what is the first iteration at which state $(1,1)$ achieves a positive value? Assume as usual that $V_0(s) = 0$ for all s. 2 because we can now propagate values by two-steps directly.

(iii) [1 pt] How many iterations of two-step Bellman equation will it take for the value function to converge if N=6? Assume as usual that $V_0(s) = 0$ for all s. 3. By doing three iterations of value iteration with each considering two steps we will find optimal value for path lengths of 6.

(iv) [2 pts] If we were to define the two-step look-ahead Bellman equation for **highest reward** and **product of rewards**, which of the three formulations will converge the fastest for the above MDP and N = 3?

○ Sum of rewards
○ Highest reward
○ Product of rewards
○ Two of them converge the fastest, in the same number of iterations
[✓ All of them converge in the same number of iterations]

Q3. [15 pts] Q-Learning: Weights Amnesia

Pacman travels to a simple gridworld with a dummy ghost who doesn't know how to move! Unfortunately, he caught a fever on its way which messed up the weights $w \in \mathbb{R}^2$ it learned for approximate Q-Learning. The only thing he remembered is the feature extraction function $f(s,a) = \begin{bmatrix} d_{ghost}+1 \\ d_{food}+1 \end{bmatrix}$, where d_{ghost} and d_{food} are the Manhattan distances to the ghost and the food, respectively, after taking action a from state s.

Pacman tells you that its learning rate $\alpha = 0.2$, its discount factor $\gamma = 0.9$, and its current weights $w = \begin{bmatrix} -3 \\ 2 \end{bmatrix}$. Work out the following s to help it learn a better weight!

You also notice some properties of this gridworld:

- All actions will succeeed with probability 1. If an action makes Pacman move into a wall, then Pacman's position will remain unchanged for this action.
- If Pacman runs into a ghost, the game ends, and Pacman will receive a -10 penalty. (E is a terminal state)
- If Pacman runs into a food, the game ends, and Pacman will receive a +10 reward. (A is a terminal state)
- Otherwise, Pacman will receive a -1 penalty.

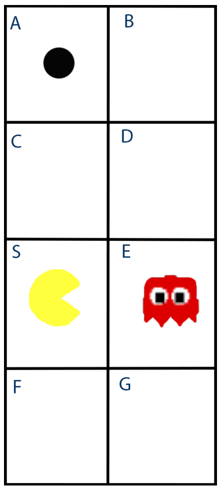


Figure 1

(a) **(i)** [2 pts] Compute the following features and Q-values.

$f(S, \rightarrow) =$ $[1, 4]$ $\qquad f(S, \uparrow) =$ $[3, 2]$

$Q(S, \rightarrow) =$ 5 $\qquad Q(S, \uparrow) =$ -5

(b) **(i)** [1 pt] Now, in order to learn a good policy, you tell Pacman to use ϵ-greedy policy. Assume that $\epsilon = 0.6$, what's the probability assigned to the action with the highest Q value?

$\mathbb{P}(a^*|S) =$ $0.15 + 0.4 = 0.55$

(ii) [2 pts] Pacman then randomly sampled a RIGHT($\rightarrow$) action and gathers a transition experience, what will be Pacman's updated weights?

$w_1 =$ -6

$w_2 =$ -10

Since moving to the right triggers a terminal state, we don't need to consider the discounted future rewards. Transition experience is (S, →, terminal, -10), so weights should be updated as follows:

$$\text{difference} = R(S, \rightarrow) - Q(S, \rightarrow) = -10 - 5 = -15$$

$$w := w + \alpha \cdot \text{difference} \cdot f(S, \rightarrow) = \begin{bmatrix} -3 \\ 2 \end{bmatrix} - 3 \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} -6 \\ -10 \end{bmatrix}$$

(iii) [4 pts] After updating the weights, rank the 4 new Q-values at state S in a decreasing order.

	$Q(S, \rightarrow)$	$Q(S, \downarrow)$	$Q(S, \leftarrow)$	$Q(S, \uparrow)$
Largest	○	○	○	●
	○	○	●	○
	●	○	○	○
Smallest	○	●	○	○

$Q(S, \rightarrow) = -46, Q(S, \downarrow) = -58, Q(S, \leftarrow) = -42, Q(S, \uparrow) = -38$

(c) Assume that you train Pacman with an infinite number of episodes, a proper learning rate α, and everything else kept unchanged.

(i) [2 pts] Is it able to learn the true optimal Q-values?

- Yes, since we visited every state infinitely many times, Q-learning will converge to optimal.
- Yes, but not for the reason above.
- No, since the ϵ-greedy policy we use for learning is not optimal.
- [✓ No, but not for the reason above.]

The reason is that the feature extractor function is not powerful enough. The optimal Q-values $Q(C, \uparrow) = R(C, \uparrow) = 10 = w^T f(C, \uparrow)$ and $Q(S, \rightarrow) = R(S, \rightarrow) = -10 = w^T f(S, \rightarrow)$. Solving these two equations gives $w = \begin{bmatrix} \frac{10}{3} \\ -\frac{10}{3} \end{bmatrix}$. Pick another state-action pair's Q-value, for example, $Q(S, \uparrow) = R(S, \uparrow) + \gamma \max_a Q(C, a) = R(S, \uparrow) + \gamma Q(C, \uparrow) = -1 + 0.9 \cdot 10 = 8$. Verify that $w^T f(S, \uparrow) = \frac{10}{3} \neq 8$. Hence, it's impossible to learn the optimal Q-values.

(ii) [2 pts] Is it able to learn the true optimal state values?

- Yes, because the learned Q-values could be optimal.
- Yes, but not for the reason above.
- [✓ No, because the learned Q-values are never optimal.]
- No, but not for the reason above.

Since it cannot learn the true Q-values for certain state-action pairs (see the solution for the previous question), it cannot learn the true state values as well.

(iii) [2 pts] Is it able to learn the true optimal policy?

- Yes, because it could learn the optimal Q-values.
- [✓ Yes, but not for the reason above.]
- No, because it never learns the optimal Q-values.
- No, but not for the reason above.

For the policy to be optimal, we want $\pi^*(s) = \arg\max_a Q(s, a)$, and the optimal policy is to move closer to the food. Suppose the weight is $w = [a, b]$, feature is $f(s, a) = [x, y]$, consider the following claims:

- We only move closer to the ghost if it moves us closer to the food as well (e.g. $\pi(F) = \uparrow$). If it doesn't, we would prefer other actions (e.g. stay still by moving into the wall). Thus, we want $a(x-1) + b(y-1) > ax + by > a(x-1) + b(y+1) \implies b < a < -b$.
- We always move closer to the food and move farther away from the ghost if such action exists. This translates to $a(x+1) + b(y-1) > ax + by \implies a > b$.

Combining the two claims gives us the conditions $b < a < -b$ of the weights for the optimal policy. Clearly, this condition is attainable.

Q4. [10 pts] Search: Algorithms

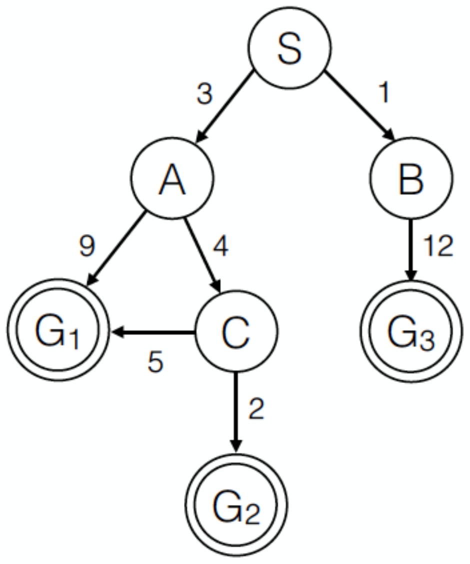


Figure 2: Search graph

	A	B	C	S
H-1	0	0	0	0
H-2	6	7	1	7
H-3	7	7	1	7
H-4	4	7	1	7

Figure 3: Heuristics table

(a) Consider the search graph and heuristics shown above. Select all of the goals that could be returned by each of the search algorithms below.

For this question, **if there is a tie on the fringe, assume the tie is broken randomly.**

(i) [1 pt] DFS

[✓ G_1]
[✓ G_2]
[✓ G_3]

DFS can arbitrarily break ties on the fringe so the search can reach any goal.

(ii) [1 pt] BFS

[✓ G_1]
☐ G_2
[✓ G_3]

BFS will only find the goals two edges away from the start node since it will always consider them before G2.

(iii) [1 pt] UCS

☐ G_1
[✓ G_2]
☐ G_3

UCS will find the closest goal by distance, which is G2.

(iv) [1 pt] Greedy with H-1

[✓ G_1]
[✓ G_2]

[✓ G_3]

Greedy search is equivalent to DFS if the heuristic is always 0.

(v) [1 pt] Greedy with H-3

[✓ G_1]

G_2

[✓ G_3]

Here greedy search will only find G1 and G3 since it will explore either node A or B and find the goal immediately.

(vi) [1 pt] A^* with H-1

G_1

[✓ G_2]

G_3

A* with H-1 is equivalent to UCS.

(vii) [1 pt] A^* with H-2

G_1

[✓ G_2]

G_3

Here A* will find the optimal solution since node C has a really low heuristic value and cost to get to.

(b) For each heuristic, indicate whether it is consistent, admissible, or neither (select more than one option if appropriate):

(i) [1 pt] H-1

[✓ Consistent]

[✓ Admissible]

Neither

All 0 heuristics are always both.

(ii) [1 pt] H-2

Consistent

[✓ Admissible]

Neither

Not consistent due to the sharp drop A-C.

(iii) [1 pt] H-3

Consistent

Admissible

[✓ Neither]

Neither since it overestimates the total path to the goal A-C-G2.

Q5. [8 pts] Search: Bowser's Kingdom

(a) Mario is looking for Princess Peach in Bowser's N x M kingdom. Along the way, he has to collect all K > 0 stationary stars by landing in the same spot as the star. Every time he collects a star, he gets a power up that allows him to move one extra space per timestep. For example, if Mario collects 1 star, he can now move up to 2 spaces per time step, when he collects a 2nd star he can move 3 spaces per time step and so on. The power up comes into effect in the timestep after Mario collects the star and lasts for the rest of the search.

Princess Peach is a capable independent woman who has already escaped and is now **also moving around one space per timestep** in the kingdom. Princess Peach cannot collect stars. Both Mario and Princess Peach can only move North, South, East, and West. The search ends when Mario and Peach land in the same space, and Mario has collected every star in the kingdom.

(i) [3 pts] What is the size of the minimum state space for this problem?

○ $(MN)^2$ [✓ $(MN)^2(2^K)$] ○ $4*(MNK)^2$

○ 2^{MN} ○ $(MNK)^2$ ○ $4*2^{K(MN)^2}$

○ $MN(2^K)$ ○ $4*MN(2^K)$ ○ $4*(MN)^2(2^K)$

Now Bowser has been alerted and has begun scanning the kingdom by quadrants. At each time step Bowser will scan a quadrant in clockwise order starting from the top right quadrant. If Mario is caught, i.e. he is in the quadrant that is currently being scanned, then it will be game over.

(ii) [1 pt] If state space in the previous part was X, state the size of the minimum state space with this updated game rule?

[✓ $4X$] ○ 2^4X

○ X ○ 4^X

○ $\frac{X}{4}$ ○ None of the above

(iii) [4 pts] Assume that the scanning is still in place. Check all of the following that are admissible heuristics (Note that any distances from Mario to stars or from stars to Princess Peach are 0 if there are no remaining stars):

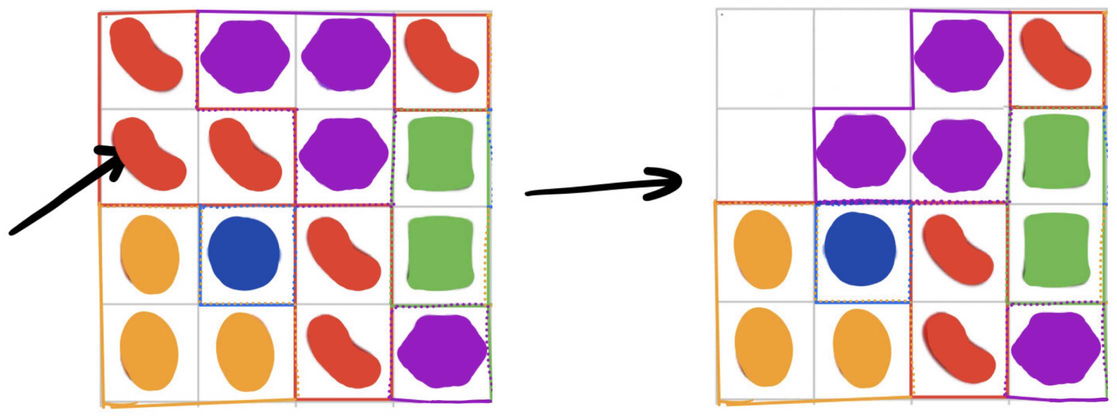
☐ 1 for every state.
☐ Manhattan distance from Mario to Princess Peach divided by 2.
[✓ Manhattan distance from Mario to Princess Peach divided by (K + 2).]
☐ Sum of Manhattan distances from Mario to all stars and to Princess Peach.
☐ (Sum of Manhattan distances from Mario to each star + Manhattan distance from Mario to Princess Peach)/(3K).
[✓ (Manhattan distance from Mario to furthest star + Manhattan distance from that furthest star to Princess Peach)/(K + 2).]
☐ (Manhattan distance from Mario to closest star + Manhattan distance from that closest star to Princess Peach)/ K.

SID: ______________

Q6. [15 pts] Caramel Crush

Mesut is on his phone during a staff meeting and decides to play his favorite mobile game - Caramel Crush. He decides to use informed search to solve the game optimally.

Caramel Crush has an N by N grid of squares that be either be empty, or one of $C - 1$ differently colored gems. At each turn, Mesut can destroy a colored gem from the grid, also recursively destroying any touching gems of the same color. After each destruction, the gem will fall down (such that there is never an empty space under any gem). The goal of the game is to destroy all gems with as few moves as possible.



(a) First, let's help Mesut understand the game.

(i) [2 pts] Does every state have a unique sequence of moves to the goal state?

○ Yes [✓ No]

The sequence of moves to the goal state from every state is not unique since often times the order in which you destroy different colored gems does not matter.

(ii) [2 pts] What is the size of the state space?

○ N^2 [✓ C^{N^2}]

○ CN^2 ○ N^{2^C}

Each square within the grid can be either empty or one of $C - 1$ colors, so we have C choices for each of the squares, giving us C^{N^2}.

(b) Now, Mesut asks us to come up with a heuristic and search algorithm for him. Given a heuristic $h(n)$, state whether it is optimal for A* tree search, A* graph search, both, or neither. Note: each action has a cost of 1

Admissibility of the heuristic guarantees optimality for A* tree search.

Consistency of the heuristic guarantees optimality for both A* tree and graph search.

(i) [2 pts] h_1 = number of gems left

○ optimality for A* tree search only [✓ optimality for neither]

○ optimality for A* graph search only ○ optimality for both

Due to the recursive destruction rule, the heuristic will severely overestimate the number of moves left to get to the goal state, which means it is inadmissible and not optimal for either.

(ii) [2 pts] h_2 = unique number of gem colors left

○ optimality for A* tree search only
○ optimality for A* graph search only
○ optimality for neither
[✓ optimality for both]

Since we can at most get rid of one color with every action, the heuristic is admissible and consistent since $0 \leq h_2 \leq h^\star$ and heuristic decreases by at most 1 with every action.

(iii) [3 pts] h_3 = number of contiguous chunks of the same color gem (as outlined in the figure)

○ optimality for A* tree search only
○ optimality for A* graph search only
[✓ optimality for neither]
○ optimality for both

h_3 is inadmissible because the number of contiguous blocks of the same color can overestimate the number of moves to solve the problem optimally. Due to gems falling down after each destruction, two chunks can join together, allowing Mesut to destroy the joined chunk in one move (which allows Mesut to finish in less steps than given by heuristic for some states).

(c) Mesut gave up coding the Caramel Crush solver.

Instead, let's think about an abstract general search problem with the given properties.

(i) [4 pts] Suppose we construct a heuristic h_1, such that for every edge n_i to n_j with cost c_{ij}, our heuristic follows the property $\frac{h_1(n_i)}{h_1(n_j)} = e^{c_{ij}}$. What properties would a new heuristic h_2 have if $h_2 = ln(h_1)$?

○ Only admissible
○ Only consistent
[✓ Both]
○ Neither

We can plug in h_2 into the definition of consistency. Consider an edge from n_i to n_j. $h_2(n_i) - h_2(n_j) = ln(h_1(n_i)) - ln(h_1(n_j)) = ln(\frac{h_1(n_i)}{h_1(n_j)}) = ln(e^{c_{ij}}) = c_{ij} \leq c_{ij}$. Therefore the heuristic is admissable and consistent.

SID: ______________

Q7. [20 pts] Game Trees

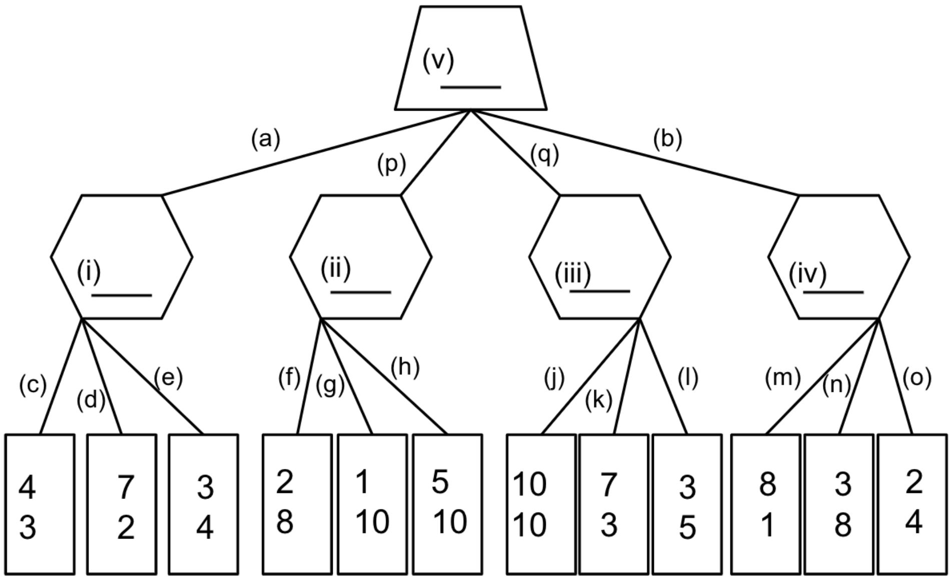
Pacman and the ghost are playing a game, where Pacman and the ghost have their own score. Both are trying to maximize their personal score. When breaking ties, they will also prefer a larger difference between their score and the other's.

A player can score anywhere between 1 and 10 points.

The figure below shows the game tree of pacman's max node (quadrilateral) followed by the ghost's nodes (hexagons). The scores shown at the leaf nodes follow $\begin{bmatrix} pacmanscore \\ ghostscore \end{bmatrix}$

For example, pacman would choose $\begin{bmatrix} 4 \\ 2 \end{bmatrix}$ over $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and the ghost would choose $\begin{bmatrix} 5 \\ 9 \end{bmatrix}$ over $\begin{bmatrix} 6 \\ 9 \end{bmatrix}$.

(a) Fill in the blanks with the pair of scores preferred by each node of the game tree.



(i) [1 pt] What branch is chosen in node i)?

○ c)

○ d)

● e)

(ii) [1 pt] What branch is chosen in node ii)?

○ f)

● g)

○ h)

(iii) [1 pt] What branch is chosen in node iii)?

● j)

○ k)

○ l)

(iv) [1 pt] What branch is chosen in node iv)?

○ m)

● n)

○ o)

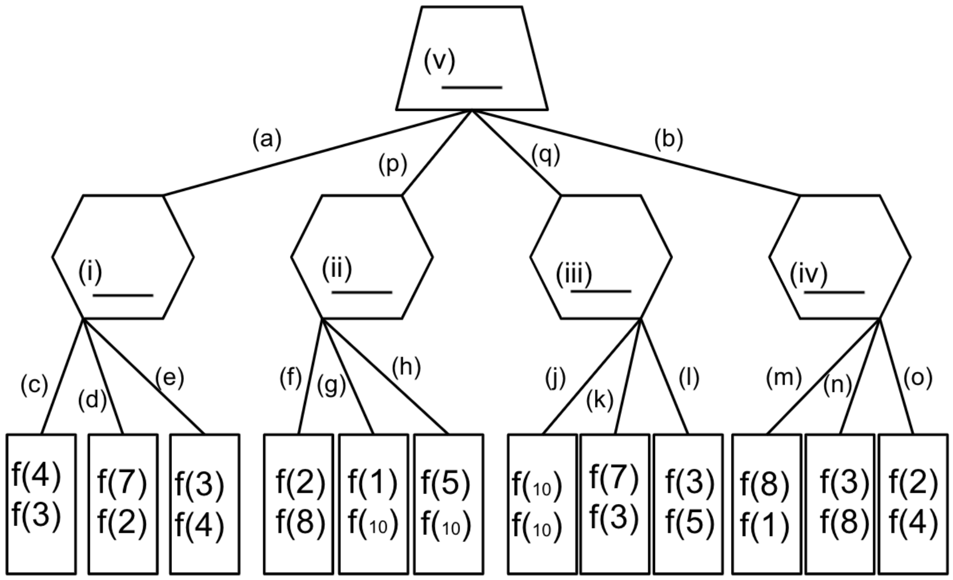
(v) [1 pt] What branch is chosen in node v)?

- ○ a)
- ○ p)
- [✓ q)]
- ○ b)

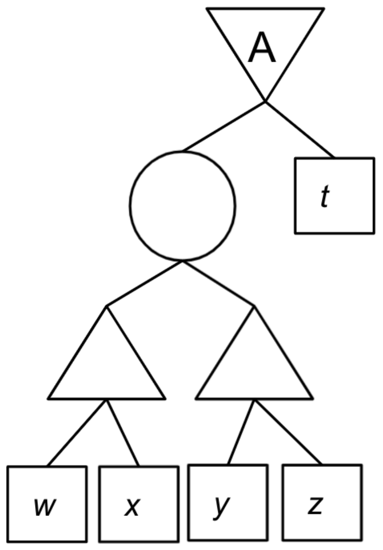
(b) [3 pts] You decide to save time by pruning branches in your game tree search. Mark the branches that do not need to be explored. Assume that branches are explored from left to right.

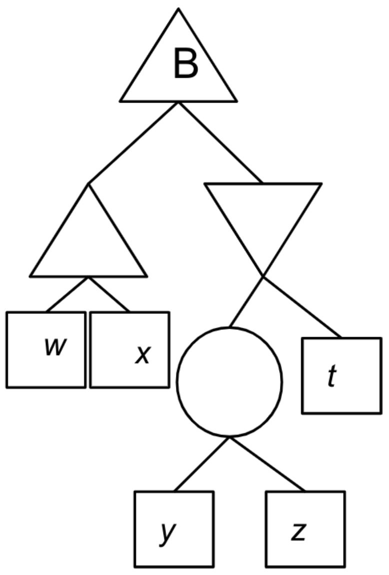
☐ (a)	☐ (f)	[✓ (l)]
[✓ (b)]	☐ (g)	☐ (n)
☐ (c)	[✓ (h)]	☐ (o)
☐ (d)	☐ (j)	☐ (p)
☐ (e)	[✓ (k)]	☐ (q)

(c) [2 pts] Suppose we have square function $f(x) = x^2$ that is applied to the leaves of the same game tree as shown below. Now determine which branches will be pruned by crossing out the line of branches that do not need to be explored. Again, assume that branches are explored from left to right.



☐ (a)	☐ (f)	[✓ (l)]
[✓ (b)]	☐ (g)	☐ (n)
☐ (c)	[✓ (h)]	☐ (o)
☐ (d)	☐ (j)	☐ (p)
☐ (e)	[✓ (k)]	☐ (q)





(d) We want to identify the relationships between the values at the root (value of A and B) and variables in the game tree.

Both game trees have the same variables (w, x, y, z, t) and $\mathbf{x} > \mathbf{t}$.

Upwards triangles represent max nodes while downwards triangles represent min nodes. Change nodes are circles and prefer the left action with probability 0.75.

For each of the following, indicate whether the relationship is always, sometimes, or never true.

(i) [1 pt] $A > t$

- ○ Always true
- ○ Sometimes true
- ● Never true

(ii) [1 pt] $A < t$

- ○ Always true
- ● Sometimes true
- ○ Never true

(iii) [1 pt] $A \geq x \, OR \, A \geq y$

- ○ Always true
- ● Sometimes true
- ○ Never true

(iv) [1 pt] $A \leq max(w, x, y, z)$

- ● Always true
- ○ Sometimes true
- ○ Never true

(v) [1 pt] $B < w$

- ○ Always true
- ○ Sometimes true
- ● Never true

(vi) [1 pt] $B \geq x$

- ● Always true
- ○ Sometimes true

○ Never true

(vii) [2 pts] $B < t$

○ Always true

○ Sometimes true

[✓ Never true]

(viii) [1 pt] $B \geq max(w, x)$

[✓ Always true]

○ Sometimes true

○ Never true

(ix) [1 pt] $A < B$

[✓ Always true]

○ Sometimes true

○ Never true