CS 188 Fall 2020 — Introduction to Artificial Intelligence — Midterm

- You have approximately 110 minutes.
- The exam is open book, open calculator, and open notes.
- In the interest of fairness, we want everyone to have access to the same information. To that end, we will not be answering questions about the content or making clarifications.
- 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	/15
Q2.	Demolition Pacman	/17
Q3.	Pacman Marching Band	/10
Q4.	CSP: Cramming for the Exam	/15
Q5.	Learning without Actions	/15
Q6.	BYOR - Bring Your Own Reward	/12
Q7.	Tom and Jerry	/16
	Total	/100

THIS PAGE IS INTENTIONALLY LEFT BLANK

SID: _____

Q1. [15 pts] Potpourri

(a) [2 pts] Suppose we define a modified A* graph search (M-A*GS) that takes a constant $c \geq 1$ and multiplies it by our heuristic before calculating $f(n)$. Essentially in M-A* GS, $f(n) = g(n) + c \cdot h(n)$.

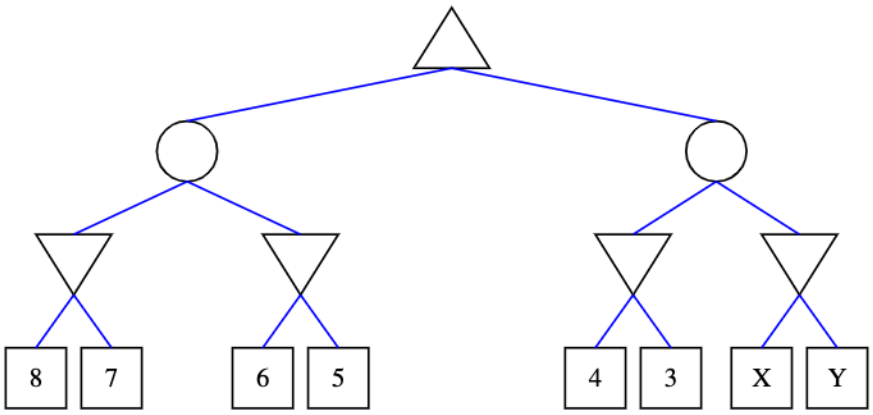
Which of the following statements are true? Select all that apply. Assume that ties of the same priority are broken alphabetically and that $h(n) \geq 0$.

- ☐ M-A*GS will return the same path as unmodified A* graph search for the same graph and heuristic.
- ☐ Given an admissible heuristic, M-A*GS is guaranteed to return the optimal path to the goal.
- ☐ Given a consistent heuristic, M-A*GS is guaranteed to return the optimal path to the goal.
- ☐ The number of nodes expanded in M-A*GS is less than or equal to the number of nodes expanded in unmodified A* graph search for the same graph and heuristic.
- ● None of the Above

M-A*GS, is a greedier version of traditional A*, which sacrifices A*'s guarantees of optimality for speed by multiplying $h(n)$ by some positive weight greater than 1.

1. False, because even using the same graph and heuristic, multiplying $h(n)$ by c means that we might end up exploring nodes in very different orders since $f(n)$ will end up varying.
2. False, because although heuristic $h(n)$ underestimates the cost to the goal by itself, multiplying by c makes it lose this guarantee. Additionally, admissibility by itself is not sufficient to guarantee optimality for traditional A* Graph Search, so it certainly is not sufficient for a greedier approach.
3. False, because similarly, the new heuristic value $c \cdot h(n)$ is not guaranteed to be consistent with the current graph, even though $h(n)$ by itself is consistent.
4. False, since even though M-A* GS is greedier in general, we can consider some cases where it is not. For example, if we consider a case where nodes with small heuristic values can be found very far away from the goal, we end up exploring these nodes first over nodes that may be among the actual optimal path to the optimal goal.

(b) Consider the following game tree, where the upward-facing triangle (the root node) is a maximizer, the upside down triangles are minimizers, the circles are chance nodes which select values uniformly at random, and the squares are leaf nodes.

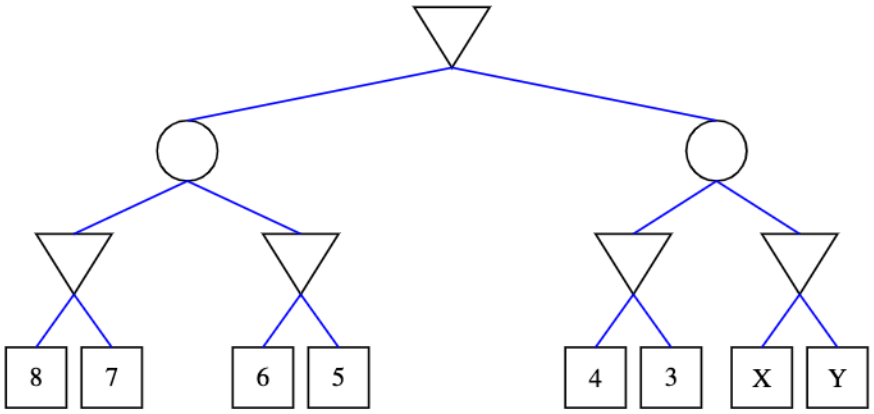


(i) [2 pts] **Assume that $X \geq 0$, $Y \geq 0$, and that we prune based on equality.** For example, for a maximizer node, if a subtree is guaranteed to propagate a value to its parent that is **less than or equal** to the parent's current value, we prune the remaining nodes in the subtree. What is the maximum possible integer value of X such that we are guaranteed to prune Y? If no such value exists, write 'Not Possible'.

9

The value propagated to the root node from the left subtree is 6. In order to prune Y, we have to ensure that the chance node of the right subtree must be guaranteed to have an expected value less than or equal to 6. Therefore, the maximum possible integer value of X that this works for must satisfy the equation $\frac{X+3}{2} = 6$, so $X = 9$.

(ii) [2 pts] Suppose we change our root node to a minimizer so the tree looks like the following:



Assume that $X \geq 0$, $Y \geq 0$, and that we prune based on equality. For example, for a minimizer node, if a subtree is guaranteed to propagate a value to its parent that is **greater than or equal** to the parent's current value, we prune the remaining nodes in the subtree. What is the maximum possible value of X such that we are guaranteed to prune Y? If no such value exists, write 'Not Possible'.

0

If $X = 0$, since we know that $Y \geq 0$, there is no value of Y that would be smaller than X, and thus Y could always be pruned.

(c) Sick of fighting with ghosts all the time, Pacman decides to start fresh, finding his new passion as a factory worker delivering packages.

We can model his job as an MDP: assume that the factory is a 10 by 10 chessboard, Pacman starts in the lower left corner $(1, 1)$ holding a package, and the goal state G is the upper right corner of the grid $(10, 10)$. Pacman can move up, down, left, or right one square at a time, unless doing so would cause him to leave the board.

Additionally, all of Pacman's actions are deterministic, and when he is at the goal state G, he can only take the *exit* action, which will move Pacman to a terminal state X and end the game.

(i) [1 pt] Denote Pacman's starting location $S = (1, 1)$. Which of the following values will be equal to $V^*(S)$?

Assume that $R(G, exit, X) = 100$ and all other state-action pairs have a reward of 0. Also assume that $\gamma = 1$ and that the values of all states are initialized to value 0 at iteration 0. Select all that apply.

- [x] $V_{100}(S)$
- [x] $V_{22}(S)$
- [x] $V_{20}(S)$
- [ ] $V_{18}(S)$
- [ ] $V_{10}(S)$
- [ ] $V_0(S)$
- ○ None of the Above

Since the discount factor is 1 and the only reward to be gained is from the terminal state, the value of the starting location will reach its optimal value in the minimum number of steps it is away from the goal state, plus one for performing the *exit* action. Therefore, since the shortest direct path from S to G is 18 spaces, the first iteration of $V_i(S)$ where $V_i(S) = V^*(S)$ is $i = 19$. All values for iterations after this will already have converged, and also be equivalent.

(ii) [2 pts] Pacman wants his optimal action at every state to be to head in the direction of the goal state G. For which of the following reward functions will running value iteration **guarantee** that the optimal action will be to head toward G for every state (except at G, where the optimal action should be to *exit*)?

Recall that $G = (10, 10)$. Assume that $\gamma = 1$ and that the values of all states are initialized to value 0 at iteration 0. Select all that apply.

- [ ] $R(s, a, s') = 1$ for non-goal states s and $R(G, exit, X) = 100$
- [ ] $R(s, a, s') = 0$ for non-goal states s and $R(G, exit, X) = 100$

- [x] $R(s, a, s') = -1$ for non-goal states s and $R(G, exit, X) = 100$
- [ ] None of the Above

Only a living reward of -1 works, since we need to incentivize Pacman to reach and leave from the terminal state as soon as possible. The reason 0 doesn't work is because all states will end up converging to the same value, so the policy extracted from this is not guaranteed to be going towards G. 1 doesn't work because Pacman is incentivized to never leave and claim infinite reward.

(iii) [2 pts] Now suppose that Pacman decided to use samples to figure out his Q-values instead, and after several episodes, he arrives at the following Q-values for his starting state $S = (1, 1)$:

- $Q(S, up) = 4.5$
- $Q(S, right) = 10$
- $Q(S, left) = 0$
- $Q(S, down) = 0$

Suppose that Pacman is now back at state S and wants to decide what action he should take using an $\epsilon-$greedy exploration method. What is the probability that the action he chooses is *right*? Your answer may contain ϵ. Assume that any random movements are chosen uniformly from all possible actions.

- [ ] 0
- [ ] $\frac{1}{4}$
- [ ] $\frac{\epsilon}{4}$
- [ ] $\frac{1-\epsilon}{4}$
- [ ] $\frac{1+3\epsilon}{4}$
- [x] $1 - \frac{3\epsilon}{4}$
- [ ] $\frac{1}{2}$
- [ ] $1 - \frac{\epsilon}{2}$
- [ ] $1 - \frac{\epsilon}{4}$
- [ ] $\frac{3}{4}$
- [ ] 1
- [ ] None of these

In $\epsilon-$greedy exploration, we exploit with probability $(1 - \epsilon)$ and explore with probability ϵ. Since we know that action *right* corresponds to the maximum Q-value at state S, both exploiting and exploring can result in taking action *right*. Therefore, we have $(1 - \epsilon) + \frac{\epsilon}{4} = 1 - \frac{3\epsilon}{4}$

(d) You decide to use approximate Q-learning to figure out the optimal policy in a generic MDP. You have n feature functions, $f_1(s, a), f_2(s, a), \ldots, f_n(s, a)$ and n corresponding weights to learn, $w_1, w_2, \ldots, w_n$ that are all initialized to zero. Assume that discount factor $\gamma = 1$ and learning rate $\alpha = 1$.

(i) [1 pt] You observe your first transition from state, s_1 to state s_2, having taken action a_1, and earning a reward of 10. After updating the weights according to the weight update from lecture you noticed w_1 increased. Select the most accurate choice below.

- [x] $f_1(s_1, a_1)$ must be positive
- [ ] $f_1(s_1, a_1)$ must be negative
- [ ] $f_1(s_1, a_1)$ must be equal to 0
- [ ] There is not enough information to select any of the above choices

Sample = Reward + Highest Q value out of s_2. We know all the Q values are 0 after initialization so Sample = Reward = 10. Thus difference between sample and the old Q value is 10 so $f_1(s_1, a_1)$ has to be positive for $f_1(s_1, a_1)$ * difference to be positive.

(ii) [1 pt] You let the agent run for a while, updating your weights as you go. Then, you observe a transition from state s_3 to state s_4, having taken action a_3, and earning a reward of 10. After updating the weights according to the weight update from lecture you noticed w_1 increased. Select the most accurate choice below.

- [ ] $f_1(s_3, a_3)$ must be positive
- [ ] $f_1(s_3, a_3)$ must be negative
- [ ] $f_1(s_3, a_3)$ must be equal to 0
- [x] There is not enough information to select any of the above choices

We don't know what the weights were set to while the algorithm was running so we don't know what the max Q-value out of s_4 is so we can't know what the sample is.

(iii) [1 pt] You finish training your weights and you are about to use them to calculate the optimal policy for each state. However, you accidentally increment one of your weights, w_2, by an unknown amount. You notice the corresponding feature function, $f_2(s, a)$, is actually only dependent on s (i.e.: $f_2(s, a) = g(s)$ for some function g). Could your mistake change the optimal policy you calculate?

- [ ] Your change to w_2 can affect your calculated policy
- [x] Your change to w_2 cannot affect your calculated policy

f_2 is not dependent on the action, so it cannot help you distinguish between actions when calculating all the Q-values for a single state since they will all be incremented by the same amount.

(iv) [1 pt] Your friend argues that the true value of w_2 is not very important. They claim that if you continued to train your weights infinitely, on data with sufficient random exploration, w_2 will approach zero. Is your friend correct?

○ Yes, w_2 will approach zero as you continue to train

● No, w_2 may not approach zero as you continue to train

$f_2(s, a)$ and w_2 can still contribute to accurately estimating the Q-value since it can show that certain states are "good" or "bad" which is relevant for other states that transition to those certain states.

SID: _____

Q2. [17 pts] Demolition Pacman

Pacman just bought a new square (S) in an M by N grid world that contains some rough terrain. He wants to clear a path from his new home to the closest grocery store, which is another square (G) on the grid world. However, there are R squares on the grid that are impassable because there are rocks in the way. Fortunately, Pacman is carrying $D-1$ sticks of dynamite with him. On any turn, Pacman can throw a stick of dynamite at one of eight neighboring squares (diagonal throws are allowed) if it contains rocks, consuming a stick and removing the rocks in that location. If he chooses not to throw dynamite, Pacman can also drive his truck onto one of eight neighboring squares(diagonal moves are allowed) if there is nothing blocking him. However, his truck only has $F-1$ units of fuel and whenever Pacman moves to a square, one unit of fuel is consumed. Once Pacman runs out of fuel and dynamite he has no more actions available. Assume R, D, and F are all greater than 3 and $2 * D < R$.

(a) [4 pts]

(i) [3 pts] Complete the expression below so that X evaluates to the size of the minimal state space. You can use integers as well as any of the variables mentioned in the problem.

$$X = M^a * N^b * R^c * D^d * F^e * 2^f$$

$a =$ 1 $b =$ 1 $c =$ 0

$d =$ 0 $e =$ 1 $f =$ R

We have $M * N$ possible values for the location. F possible values for the amount of fuel left. We need to store a boolean for each of R rocks for whether it was destroyed or not. Thus we raise 2^R and we don't need R as a base. We don't need to keep track of how much dynamite is left because every time we destroy a rock we have used a dynamite and we can figure out how many rocks we destroyed with our booleans for each rock.

(ii) [1 pt] What is the maximum possible branching factor?

8

There are 8 neighboring squares. If there is a rock in one, you can throw a dynamite but you cannot move into it. If there is no rock, you can move into it but you cannot throw a dynamite.

(b) [4 pts] For each subpart below, select all algorithms that can be used to find the specified sequence of actions. The algorithm should be able to find such a sequence on any grid world that contains that sequence without failure. Assume you are allowed to define costs between states however you want.

(i) [1 pt] Any sequence of actions that results in Pacman reaching the Grocery store.

■ DFS Tree Search ■ DFS Graph Search ■ BFS ■ UCS ○ None of these

DFS Graph Search, BFS, and UCS are complete so they will always find a solution. DFS Tree Search is also complete in this problem because it cannot infinitely recurse since dynamite or fuel is used after every turn.

(ii) [1 pt] The sequence of actions that results in Pacman reaching the Grocery store while using the least amount of dynamite possible.

☐ DFS Tree Search ☐ DFS Graph Search ☐ BFS ■ UCS ○ None of these

We must use UCS with an edge cost between nodes that use dynamite and 0 edge cost between nodes that use fuel.

(iii) [1 pt] The sequence of actions that results in Pacman reaching the Grocery store while using the most amount of dynamite possible.

☐ DFS Tree Search ☐ DFS Graph Search ☐ BFS ☐ UCS ● None of these

We cannot use UCS (or any of the other algorithms) to find a path with the "most" amount of transitions. You cannot use a positive edge costs for transitions that don't use dynamite because that would find a path that avoids not using dynamite the most, which is different from using the most dynamite.

(iv) [1 pt] The sequence of actions that results in Pacman reaching the Grocery store while having the lowest sum of fuel consumed plus dynamite consumed.

☐ DFS Tree Search ☐ DFS Graph Search ■ BFS ■ UCS ○ None of these

This is equivalent to finding the path with the least amount of turns since a dynamite or a fuel is used on every turn. UCS and BFS can both find this.

(c) [9 pts] Each subpart below describes a change to the above problem. **All subparts are independent** of each other, so answer each question as if it was the only change to the above description. **Note, X is equal to the size of the state space from the original problem.** You are essentially finding a scaling factor that should be multiplied by the original state space size.

(i) [3 pts] Pacman upgraded his truck so that he can now throw dynamite towards the north for free. It still costs him dynamite to throw in any of the other seven directions.
Complete the expression below so that it evaluates to the size of the new minimal state space. You can use integers as well as any of the variables mentioned in the problem.

$$X * R^a * D^b * F^c * 2^d * 3^e$$

$a =$ 0 $b =$ 1 $c =$ 0

$d =$ 0 $e =$ 0

We now have to keep track of how much dynamite was used separately, because some of the rocks may have been destroyed for free.
Select all that could possibly be true when comparing to the original problem statement.

- [x] The new ruleset will cause BFS to expand less nodes
- [x] The new ruleset will cause BFS to expand the same amount of nodes
- [x] The new ruleset will cause BFS to expand more nodes

Since we did not specify what the grid world setup is, all of them are possible. BFS will expand less nodes if there is a short path of rocks to the goal (to the north) that Pacman didn't have enough dynamite to break before. BFS will expand the same amount if there are no rocks in the way. BFS will expand more nodes if there are a lot of rocks north of the starting location that don't go towards the goal.

(ii) [3 pts] Y of the R rocks are larger than others ($1 < Y < R$). They require two sticks of dynamite to be removed (Pacman can still only throw one stick of dynamite per turn).
Complete the expression below so that it evaluates to the size of the new minimal state space. You can use integers as well as any of the variables mentioned in the problem.

$$X * R^a * D^b * F^c * 2^d * 3^e * Y^f$$

$a =$ 0 $b =$ 0 $c =$ 0

$d =$ $-Y$ $e =$ Y $f =$ 0

We now have 3 possible states instead of 2 for Y of the rocks so the state space is scaled by a factor of $\frac{3^Y}{2^Y}$.
Select all that could possibly be true when comparing to the original problem statement.

- [x] The new ruleset will cause BFS to expand less nodes
- [x] The new ruleset will cause BFS to expand the same amount of nodes
- [x] The new ruleset will cause BFS to expand more nodes

Since we did not specify what the grid world setup is all of them are possible. BFS will expand more nodes if there is a short path of rocks to the goal that Pacman no longer has enough dynamite to break. BFS will expand the same amount if there are no rocks in the way. BFS will expand less nodes if there are a lot of rocks around the starting location that don't go towards the goal that pacman can no longer break.

(iii) [3 pts] Y of the R rocks are now too large for Pacman to destroy ($1 < Y < R$). Dynamite does nothing on them so Pacman will have to move around them.
Complete the expression below so that it evaluates to the size of the new minimal state space. You can use integers as well as any of the variables mentioned in the problem.

$$X * R^a * D^b * F^c * 2^d * 3^e * Y^f$$

$a =$	0	$b =$	0	$c =$	0
$d =$	$-Y$	$e =$	0	$f =$	0

We no longer have to keep track of Y of the rocks so the state space is scaled by $\frac{1}{2^Y}$

Select all that could possibly be true when comparing to the original problem statement.

- [x] The new ruleset will cause BFS to expand less nodes
- [x] The new ruleset will cause BFS to expand the same amount of nodes
- [x] The new ruleset will cause BFS to expand more nodes

Since we did not specify what the grid world setup is all of them are possible. BFS will expand more nodes if there is a short path of rocks to the goal that Pacman can no longer break. BFS will expand the same amount if there are no rocks in the way. BFS will expand less nodes if there are a lot of rocks around the starting location that don't go towards the goal that Pacman can no longer break.

Q3. [10 pts] Pacman Marching Band

The University of Blockley has a Pacman marching band, which consists of n members. It is the morning before the Big Game, and all band members need to get together from their homes for one last rehearsal. Suppose you can control all n Pacmen simultaneously. Multiple Pacmen can be on the same square, and every Pacman can pause, or move left, right, up, or down. Suppose there are M squares on the map that are not walls, where the Pacmen can go. Your goal is to move all n Pacmen to the same square in the minimum number of time steps. Let's formulate the problem as a search problem.

(a) [8 pts] Suppose you use A^* search with heuristics to solve the search problem. At any time step, let $p_i = (x_i, y_i)$ be the coordinate of the ith Pacman. For each heuristic below, indicate (1) whether the heuristic is admissible, and (2) whether the heuristic is consistent.

(i) [2 pts] The number of pairs of Pacmen that are not on the same square.

○ Admissible ● Not Admissible
○ Consistent ● Not Consistent

Not admissible or consistent. Suppose one pacman is at (a, b) while $n-1$ Pacmen are at $(a+1, b)$, then all Pacmen can meet in 1 step.

(ii) [2 pts] 2 if the closest pair of Pacmen is 4 squares away, 0 otherwise.

● Admissible ○ Not Admissible
○ Consistent ● Not Consistent

Admissible but not consistent. Admissible because it is only ever 2 when we need at least 2 turns to win. It is not consistent because it can jump from 2 to 0 in 1 turn.

(iii) [2 pts] $\frac{1}{2}\max_{i,j}|x_i - x_j| + \frac{1}{2}\max_{i,j}|y_i - y_j|$

○ Admissible ● Not Admissible
○ Consistent ● Not Consistent

Not admissible or consistent. Suppose all Pacmen are at $(a, b), (a+1, b+1), (a+2, b)$, then all Pacmen can meet in 1 step, but this heuristic would evaluate to 1.5.

(iv) [2 pts] $\frac{1}{2}\max(\max_{i,j}|x_i - x_j|, \max_{i,j}|y_i - y_j|)$

● Admissible ○ Not Admissible
● Consistent ○ Not Consistent

Admissible and consistent. Admissible because it corresponds to the relaxed solution where Pacmen can move diagonally with no walls, since it only looks at the dimension that has the farthest difference in coordinates. It is also consistent because the absolute value decreases by at most 2 which when multiplied by $\frac{1}{2}$, the heuristic can only decrease by 1 per turn.

(b) [2 pts] Suppose you come up with an admissible and consistent heuristic h_a for the previous part based on the positions p_i of all Pacmen. Now the problem changes, and you have $m \leq n$ Pacmen carrying tubas. Because the tuba is heavier than other instruments, the tuba players need to rest for 1 time step after every time step that they move. The rest of the Pacmen remain the same as before. Is h_a still admissible for the new problem? Is it consistent?

● Admissible ○ Not Admissible
● Consistent ○ Not Consistent

Admissible because the original is a relaxed problem of this one. Consistent because for every s, s' in part a where consistency hold: $h(s) + c_a(s, s') \geq h(s')$, the new cost $c_b(s, s')$ is no less than $c_a(s, s')$ because tuba players need to rest. Thus $h(s) + c_b(s, s') \geq h(s')$ still holds.

SID: _____

Q4. [15 pts] CSP: Cramming for the Exam

Pacman is taking CS 881: Intro to Ghost Intelligence at University of Blockley this semester. It is 7 days before the midterm, but Pacman is still procrastinating! Pacman still has 1 Electronic Homework (E), 1 Written Homework (W), and 1 Project (P) to finish before the exam. Each of them takes 1 day to complete, and Pacman can only work on at most one task every day. Also, Pacman needs 2 days to review the course material before the exam (R_1 and R_2). Pacman needs your help to assign the dates to complete theses tasks!

Pacman formulates the problem as a CSP, where the tasks (E, W, P, R_1, R_2) are variables, each with domain $\{1, ..., 7\}$, representing the seven days from now until the exam.

Pacman wants the assignments of tasks to meet the following constraints:

1. Each task (E, W, P, R_1, R_2) must be assigned to a different day.
2. Both the Electronic Homework (E) and Project (P) are due in 4 days, so they must be finished in days $1, 2, 3$, or 4.
3. Since we use R_1 and R_2 to represent the first and the second day of reviewing for the exam, we assume $R_1 < R_2$, and the two days for reviewing (R_1, R_2) must also **not** be consecutive.
4. Pacman must finish all the assignments (E, W, P) before starting to review for the exam (R_1).
5. The Written Homework (W) can only be completed on even-numbered days.

We recommend summarizing and noting down these constraints for your later reference.

For all the questions below, treat the higher order constraints (1) and (4) as an equivalent series of binary constraints, and treat constraint (3) as a single binary constraint. *E.g., statement (4) represents three binary constraints: $E < R_1$, $W < R_1$, and $P < R_1$. We count them as three distinct constraints.*

(a) Pacman tries to solve the problem with local search, using the min-conflicts heuristic. Pacman starts with an arbitrary assignment: $E = 1, W = 3, P = 6, R_1 = 4, R_2 = 2$. Recall that in min-conflicts local search, Pacman will choose a conflicted variable and reassign its value to the one that violates the fewest constraints in every iteration.

(i) [1 pt] Assume that Pacman decides to reassign R_1. Which value will Pacman choose to reassign the variable to? Break value selection ties from smallest (1) to largest (7).

○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ● 7

R_1 violates two constraints in the current assignment: the constraint that P must be finished before R_1, and the constraint that $R_1 < R_2$. For $R_1 = 2, 3, 4, 5, 6$, these two constraints are still violated. For $R_1 = 1$, it violates the constraints that E, W, P must be completed before R_1, and the constraint that no two tasks can be assigned to the same day. $R_1 = 7$ only violates one constraint which is constraint (3), so we will assign it to 7.

(b) **(i)** [3 pts] Pacman notices that local search is not guaranteed to find a solution, so Pacman decides to use backtracking search with arc-consistency instead.

Please select the elements in the domains that will **remain** after enforcing unary constraints and arc consistency. Note that we have not assigned any value to any variable yet.

	1	2	3	4	5	6	7
E	■	■	■	■	□	□	□
W	□	■	□	■	□	□	□
P	■	■	■	■	□	□	□
R_1	□	□	■	■	■	□	□
R_2	□	□	□	□	■	■	■

Enforcing all the unary constraints gives the values of E and P: $1, 2, 3$ and 4. Enforcing unary constraints on P gives $2, 4, 6$, but R_1 cannot be 7 (eliminated when enforcing constraint $R_1 - R_2$), so enforcing $P - R_1$ eliminates $P = 6$. Enforcing $W - R_1$ and $W - R_2$ then eliminates 1 and 2 for R_1 and R_2, and then enforcing $R_1 - R_2$ gives the desired result. $R_1 = 3$ and $R_2 = 5$ is not eliminated here because eliminating them requires 3-consistency.

(ii) [1 pt] Pacman decides to do backtracking search using the remaining domains. Recall that in backtracking search we first need to pick a variable and assign a value to it. According to the MRV Heuristic, which variable(s) should

Pacman prioritize assigning to? If there are multiple variables tying, mark **all** of them.

☐ E ■ W ☐ P ☐ R_1 ☐ R_2

The variable W has only two values left, while all other variables have at least three values left.

(iii) [2 pts] After picking a value with the MRV heuristic, Pacman is not sure whether he should use the LCV heuristic. Select **all** of the following statements about the LCV heuristic that are correct. Note that for a statement to be correct **both the conclusion and the reasoning need to be correct**.

☐ LCV prioritizes assigning a variable to a value that results in the least number of choices in the remaining variables.

☐ LCV makes backtracking search much more efficient because it improves the asymptotic runtime of naive backtracking search.

■ LCV empirically speeds up backtracking search (compared to selecting a random value to assign) because it often reduces the number of backtracks in the search process.

■ Depending on the specific setup of a CSP problem, one may not want to use the LCV heuristic because it introduces additional computation.

Running LCV introduces additional computation because it requires rerunning arc consistency/forward checking or other filtering methods for each value, and it cannot improve the asymptotic runtime of naive backtracking in that you can always design adversarial example. However, it still often empircially speeds up backtracking.

(iv) [2 pts] Pacman feels tired after formulating and trying to solve the CSP. He adds two new constraints: he wants to take an extra day off, which means no tasks can be assigned to day 1. Also, he wants the Electronic Homework (E) to be completed before the written homework (W). Solve for all the possible solutions (if any) to this CSP after enforcing all the constraints above and these two extra constraints. Which day(s) can the project (P) possibly get assigned to? If there is no possible solution, choose "None".

○ None ☐ 1 ■ 2 ■ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7

From part b(i) the possible values for P are 1, 2, 3, and 4. The first new constraint eliminates the value 1. The second new constraint enforces W to take the value 4, so P cannot take the value 4. 2 and 3 are still valid.

(c) Pacman survived the midterm! He decides to make a plan for the second half of the semester.

The second half of the semester has N_e electronic homeworks, N_w written homeworks, and N_p projects, and we know $N_e > N_w > N_p$. Each electronic homework, written homework and project needs to be finished in a window of D_e, D_w, and D_p days, respectively, with $D_e < D_w < D_p$. For example, the domain of each variable that represents an electronic homework is $\{1, 2, ..., D_e\}$.

(i) [2 pts] With only the information given above, what is the tightest upper bound for the runtime for finding a satisfying assignment?

○ $O(D_e^{N_e} + D_w^{N_w} + D_p^{N_p})$ ● $O(D_e^{N_e} D_w^{N_w} D_p^{N_p})$ ○ $O((D_e + D_w + D_p)^{N_e+N_w+N_p})$ ○ $O(\max\{D_e^{N_e}, D_w^{N_w}, D_p^{N_p}\})$

○ $O(D_e^{N_e+N_w+N_p})$ ○ $O(D_p^{N_e+N_w+N_p})$ ○ $O((D_e + D_w + D_p)^{N_e})$ ○ $O((D_e + D_w + D_p)^{N_p})$

In the worst case backtracking search needs to visit every possible state to solve the CSP, so the worst-case runtime is $O(D_e^{N_e} D_w^{N_w} D_p^{N_p})$.

Pacman realizes that he also has homework drops! The CS 881 course policy allows at most one electronic homework drop and at most one written homework drop. An additional option, 0, is added to the domain of each variable representing an electronic homework or a written homework. For example, the domain of each variable that represents an electronic homework is now $\{0, 1, 2, ..., D_e\}$, but only one of these variables representing electronic homeworks can be assigned the value "zero".

(ii) [2 pts] The statement "There can be at most one electronic homework drop and at most one written homework drop" involves higher order constraints. Can you write a series of binary constraints that is equivalent to the higher order constraints posed in this statement? If so, what is the size of the smallest set of binary constraints that achieves this?

○ $(N_e(N_e - 1) + N_w(N_w - 1))$ ○ $(N_e^2 + N_w^2)$ ○ $\frac{N_e^2 + N_w^2}{2}$

● $\frac{N_e(N_e-1)+N_w(N_w-1)}{2}$ ○ $\frac{(N_e-1)+(N_w-1)}{2}$ ○ $(N_e - 1) + (N_w - 1)$ ○ Not Possible

For any pair of electronic homeworks and any pair of written homeworks, Pacman cannot drop both of them. This is necessary and sufficient because if Pacman drops more than one electronic homework or written homework, the pair of homeworks Pacman dropped violate the binary constraint between them.

(iii) [2 pts] Suppose that a power outage is expected to take place and the instructor decides to compensate with an additional homework drop which can be used for either written homeworks or electronic homeworks. Now Pacman has a total of three homework drops, while at most two can be used for either electronic homeworks or written homeworks. Can you write a series of binary constraints that is equivalent to the higher order constraints posed in this new statement? If so, what is the size of the smallest set of binary constraints that achieves this?

○ $(N_e(N_e-1)+N_w(N_w-1))$ ○ N_eN_w ○ $\frac{N_eN_w}{2}$

○ $\frac{N_e(N_e-1)+N_w(N_w-1)}{2}$ ○ $\frac{(N_e-1)(N_w-1)}{2}$ ○ $(N_e-1)(N_w-1)$ ● Not Possible

For a pair of variables, any possible combination of assignments of them (drop both, drop anyone of them, or drop neither) can occur, so we cannot write the higher order constraint in terms of several binary constraints.

Q5. [15 pts] Learning without Actions

We are looking for ways to change the vanilla Bellman equations presented many times in this course. To start, consider a grid world Markov Decision Process (MDP) shown in Figure 1 with **deterministic** dynamics. The agent will start in state A and the only terminal state is I, where it can take any action to exit for a reward of 10. The action space is defined as $\mathcal{A}$: {right, down, left, up, stay}. We define the following values for the MDP:

1. Discount factor $\gamma = 0.5$.
2. Living reward $r(s, a, s') = 0$ for all states $s \neq$ I.
3. Deterministic dynamics: *The next-states in this problem follow directly from the action. E.g.* if the agent selects right, it will move east, unless it is blocked by a wall, in which case the agent will stay in the current grid location, or in a terminal state, where the agent will always exit and get the reward.

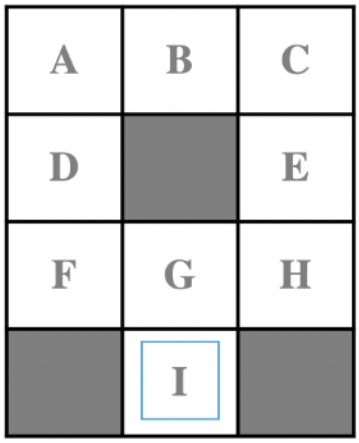


Figure 1: The MDP that we are learning.

With this information, we remember we can do value iteration to learn values of states. Let's start here.

(a) [1 pt] What is the value after one iteration of value iteration for the following states? **To start, all non-terminal states have a value of 0**, $V_0(s) = 0$ and $V_0(I) = 10$.

- $V_1(A) =$ 0
- $V_1(F) =$ 0
- $V_1(G) =$ 5

For a single iteration, the only value that will update is $V_1(G)$, since it is the only state where you can gain positive reward, For Q, we can calculate this by using the given information, and deterministic dynamics.

$V_1(A) = 0.$

$V_1(F) = 0.$

$V_1(G) = \max_a \sum_{s'} R(s, a, s') + \gamma V(s') = 0 + 0.5 * 10 = 5$

(b) [3 pts] (1 pt each) What is the optimal value function at each state?

- $V^*(A) =$ 0.625
- $V^*(F) =$ 2.5
- $V^*(G) =$ 5

Here, we will use the same reasoning, but the math is more challenging. There is a trick to seeing the solution because there is no living reward, and the value of the terminal state is set: the value is $\gamma^s \cdot V(I)$ where s is the number of steps to the exit. $V^*(A) = \gamma^4 \cdot V(I) = 0625$. $V^*(F) = \gamma^2 \cdot V(I) = 2.5$. $V^*(G) = \gamma^1 \cdot V(I) = 5$.

Now, we are working with a different formulation. Given the Markov Decision Process above, we are wondering if we can create a formulation to learn the value of states when we no longer need to consider actions: in a deterministic MDP, the final state encodes the action implicitly; how does this come into play numerically? This can give us a new value function, $Q(s, s')$.

Consider a new model, called an **inverse dynamics model**, $\mathcal{I}(s, s')$, that returns an action given the current state and the next state – this could be useful. Reminder, the action space is $\mathcal{A}$: {**right, down, left, up, stay**}.

$$\mathcal{I}(s, s') : \mathcal{S} \times \mathcal{S} \mapsto \mathcal{A}$$

(c) [3 pts] (1 pt each) As a check for understanding, return the evaluation of the inverse dynamics at the following state-pairs: (If no such action exists, the inverse dynamics model returns *None*.)

- $\mathcal{I}(A, B) =$ Right
- $\mathcal{I}(C, C) =$ Stay, Right, or Up
- $\mathcal{I}(D, E) =$ None

(d) [5 pts] Alice wishes to derive an algorithmic procedure that is analogous to Q-value iteration in this new formulation. For each letter **(A), (B), (C)** and **(D)**, fill in a single entry for the term corresponding to the correct equation to implement Q-value iteration. $N(s)$ refers to the *neighboring states* of state s (i.e.: the set of states s' that can possibly be reached by taking some action from state s). Select one bubble per row to form the whole equation.

$$\text{Regular MDP:} \quad Q(s,a) \leftarrow \sum_{s' \in N(s)} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right] \tag{1}$$

$$\text{New formulation:} \quad Q(s,s') \leftarrow \textbf{(A)} \quad \Big[\textbf{(B)} \;+\; \textbf{(C)} \;\; \textbf{(D)}\Big] \tag{2}$$

- **(A)** : ● 1 ○ $\sum_{s'' \in N(s)}$ ○ $\sum_{s'' \in N(s')}$
- **(B)** : ● $R\big(s, \mathcal{I}(s,s'), s'\big)$ ○ 0 ○ 1
- **(C)** : ● γ ○ 0 ○ 1
- **(D)** : ○ $\max_{s'' \in N(s)} Q(s,s'')$ ○ $\max_{s'' \in N(s)} V(s'')$ ○ $\max_{s'' \in N(s')} Q(s,s'')$ ○ $\max_{s'' \in N(s')} V(s'')$ ● $\max_{s'' \in N(s')} Q(s',s'')$ ○ $V(s'')$ ○ $Q(s',s'')$ ○ $Q(s,s'')$

$$Q(s,s') \leftarrow R(s, I(s,s'), s') + \gamma \max_{s'' \in N(s')} Q(s',s'')$$

(e) Conceptual questions about the new formulation:

(i) [1 pt] In a deterministic MDP like above, can Q-learning with the new value function $Q(s,s')$ learn the optimal policy?

● Yes ○ No

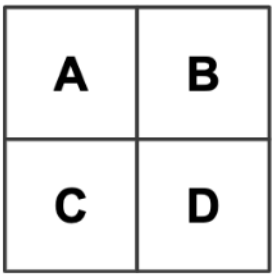
(ii) [2 pts] We can also extend this formulation of Q-learning without actions, $Q(s,s')$, to non-deterministic dynamics $\mathcal{T}(s,a,s')$ as follows. For each transition $(s,a,s',R(s,a,s'))$ we take, we update the corresponding $Q(s,s')$ value according to equation (2), where we replace all occurrences of $\mathcal{I}(s,s')$ by the action we actually take, a. After the Q-values converge, we extract the policy with $\pi(s) = \arg\max_a \sum_{s' \in N(s)} T(s,a,s')Q(s,s')$. Will this new Q-value iteration process always converge to the same policy as vanilla Q-learning in environments with non-deterministic dynamics in $\mathcal{T}(s,a,s')$?

○ Yes ● No

To see why this is false, standard Q learning has a max over actions, which in non-deterministic environments allows choosing actions in expectation over such uncertainty. In next-state Q-learning, the s' value tracks a reasonable value, but it is not guaranteed to be the optimal value, and therefore the two types of q-learning can converge to different policies.

Q6. [12 pts] BYOR - Bring Your Own Reward

Consider the following gridworld as an MDP. You assign your agent Mesut-Bot to traverse it:



From state A, the possible actions are *right* ($\rightarrow$) and *down* ($\downarrow$). From state B, the possible actions are *left* ($\leftarrow$) and *down* ($\downarrow$). For states C and D, the only valid action is to *exit*, after which Mesut-Bot will enter the *done* state and the rollout will end. We also know that in this MDP all actions are deterministic and always succeed.

Consider the following two episodes. You want to use Q-learning to learn the Mesut-Bot's optimal behavior; however, we don't know the reward value of each sample, and denote them as r_1, r_2, r_3, r_4, and r_5 accordingly (assume $r_1, r_2, r_3, r_4, r_5 \geq 0$).

s	a	s'	r
A	$\rightarrow$	B	r_1
B	$\downarrow$	D	r_3
D	exit	*done*	r_5

s	a	s'	r
A	$\downarrow$	C	r_2
C	exit	*done*	r_4

(a) [3 pts] Suppose that $\gamma = 0.5, r_1 = r_2 = r_3 = 0, r_4 = 1, r_5 = 10$. Calculate the following values after repeatedly processing the episodes one at a time using Q-Learning until convergence. Assume that all Q-values are initialized to 0 and that $\alpha = 1$.

- $Q(A, \rightarrow) =$ 2.5
- $Q(A, \downarrow) =$ 0.5
- $Q(B, \leftarrow) =$ 0

We can set up the Q-Learning equation $Q(s,a) = (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$ and repeat the two episodes to convergence as follows. We note that the first term in the equation is always equal to 0, since $\alpha = 1$, to simplify our calculations. We also denote the first episode as Episode 1 and the second episode as Episode 2:

Episode 1 (Iteration 1):
$Q(A, \rightarrow) = 0 + (0.5)(0) = 0$
$Q(B, \downarrow) = 0 + (0.5)(0) = 0$
$Q(D, exit) = 10 + (0.5)(0) = 10$

Episode 2 (Iteration 1):
$Q(A, \downarrow) = 0 + (0.5)(0) = 0$
$Q(C, exit) = 1 + (0.5)(0) = 1$

Episode 1 (Iteration 2):
$Q(A, \rightarrow) = 0 + (0.5)(0) = 0$
$Q(B, \downarrow) = 0 + (0.5)(10) = 5$
$Q(D, exit) = 10 + (0.5)(0) = 10$

Episode 2 (Iteration 2):
$Q(A, \downarrow) = 0 + (0.5)(1) = 0.5$
$Q(C, exit) = 1 + (0.5)(0) = 1$

Episode 1 (Iteration 3):
$Q(A, \rightarrow) = 0 + (0.5)(5) = 2.5$
$Q(B, \downarrow) = 0 + (0.5)(10) = 5$
$Q(D, exit) = 10 + (0.5)(0) = 10$

Episode 2 (Iteration 3):
$Q(A, \downarrow) = 0 + (0.5)(1) = 0.5$
$Q(C, exit) = 1 + (0.5)(0) = 1$

We can run the episodes for one more iteration to prove to ourselves that the Q-values have converged. Therefore, $Q(A, \rightarrow) = 2.5$ and $Q(A, \downarrow) = 0.5$. $Q(B, \leftarrow) = 0$ because it was not part of our episodes, so its Q-value never updated from 0.

(b) Now assume that $\gamma = 1$, $\alpha = 1$ and Mesut-Bot starts from state A. You don't know the rewards, but you know that you want the optimal behavior to be $(A - B - D - done)$. You have the following reward designs at your disposal:

- $S_1 : (r_1 = 0, r_2 = 0, r_3 = 0, r_4 = 0, r_5 = 0)$
- $S_2 : (r_1 = 0, r_2 = 0, r_3 = 0, r_4 = 1, r_5 = 10)$
- $S_3 : (r_1 = 5, r_2 = 5, r_3 = 5, r_4 = 5, r_5 = 5)$
- $S_4 : (r_1 = 1, r_2 = 0, r_3 = 0, r_4 = 2, r_5 = 1)$

(i) [2 pts] Using the same episodes, which of the above reward setup(s) **could result in**, but **do not guarantee**, the optimal behavior after Q-learning converges?

■ S_1 ☐ S_2 ☐ S_3 ■ S_4 ○ None

S_1 gives random actions as the optimal policy, since all Q-values will be the same, which means it could result in the desired behavior. S_2 and S_3 will guarantee the optimal behavior using the above episodes. S_4 will result in $Q(A, \rightarrow) = Q(A, \downarrow) = 2$

(ii) [2 pts] Using the same episodes, which of the above reward setup(s) will **guarantee** the optimal behavior after Q-learning converges?

☐ S_1 ■ S_2 ■ S_3 ☐ S_4 ○ None

S_2 and S_3 can guarantee the Q-learning to converge to the optimal behavior, because under S_2, $Q(A, \rightarrow) = 10, Q(A, \downarrow) = 1$, and under S_3, $Q(A, \rightarrow) = 15, Q(A, \downarrow) = 10$

(iii) [1 pt] For $0 < \gamma \leq 1$ and $\alpha = 1$, which of the following is the condition for Q-learning to **guarantee** the optimal behavior upon convergence? If none of these are the correct condition, select "None".

● $r_2 + \gamma r_4 < r_1 + \gamma r_3 + \gamma^2 r_5$ ○ $r_2 + \gamma r_4 = r_1 + \gamma r_3 + \gamma^2 r_5$ ○ $r_2 + \gamma r_4 > r_1 + \gamma r_3 + \gamma^2 r_5$ ○ None

The optimal behavior follows the policy taken in the first episode, so to guarantee the optimal behavior upon convergence, we must make sure that the overall reward from taking those actions must be greater than the alternate policy in the second episode. Starting from state A, the reward gained from following the first episode is $r_2 + \gamma r_4$, and the reward gained from following the second is $r_1 + \gamma r_3 + \gamma^2 r_5$, so the correct condition is $r_2 + \gamma r_4 < r_1 + \gamma r_3 + \gamma^2 r_5$. Equality is not enough to **guarantee** the correct optimal behavior, since tie-breaking is ambiguous here.

(c) Now we add one more transition to Mesut-Bot's dataset before $(A, \downarrow, C, r_2)$ in the second episode: $(B, \leftarrow, A, r)$, where $r = max(r_1, r_2, r_3, r_4, r_5)$. Assume that $\gamma = 1$ and $\alpha = 1$. The modified episodes and reward designs are repeated below for your convenience:

s	a	s'	r
A	$\rightarrow$	B	r_1
B	$\downarrow$	D	r_3
D	exit	$done$	r_5

s	a	s'	r
B	$\leftarrow$	A	r
A	$\downarrow$	C	r_2
C	exit	$done$	r_4

- $S_1 : (r_1 = 0, r_2 = 0, r_3 = 0, r_4 = 0, r_5 = 0)$
- $S_2 : (r_1 = 0, r_2 = 0, r_3 = 0, r_4 = 1, r_5 = 10)$
- $S_3 : (r_1 = 5, r_2 = 5, r_3 = 5, r_4 = 5, r_5 = 5)$
- $S_4 : (r_1 = 1, r_2 = 0, r_3 = 0, r_4 = 2, r_5 = 1)$

(i) [2 pts] Using the modified episodes, which of the above reward setup(s) **could result in**, but **do not guarantee**, the optimal behavior after Q-learning converges?

■ S_1 ☐ S_2 ☐ S_3 ☐ S_4 ○ None

Now, S_3, S_4 can no longer make Q-learning converge because there's a positive reward cycle: $A - B - A$. This phenomena is called "reward hacking".

(ii) [2 pts] Which of the above reward setup(s) will **guarantee** the optimal behavior after Q-learning converges?

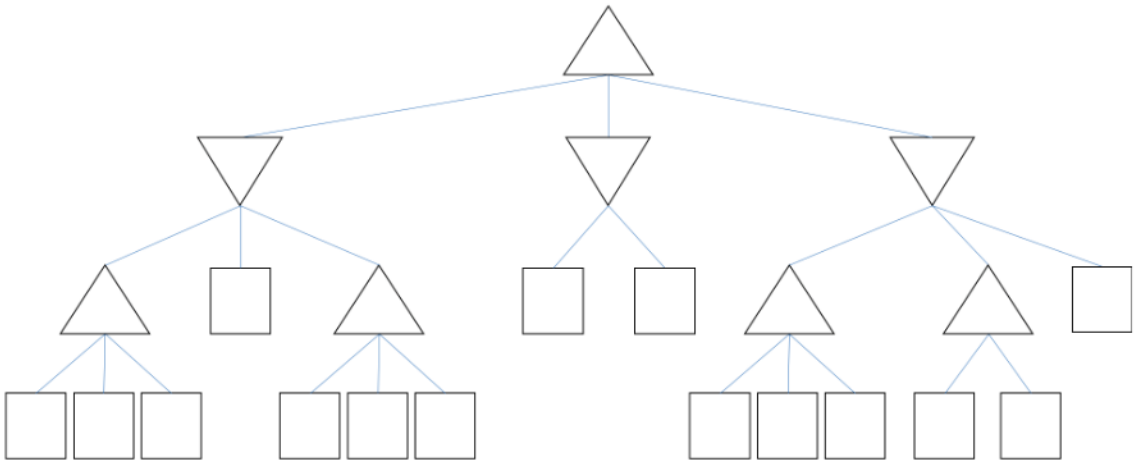
☐ S_1 ☐ S_2 ☐ S_3 ☐ S_4 ● None

Because $r = max(r_1, ..., r_5)$, and all rewards are non-negative, there's no way to eliminate the positive reward cycle unless all rewards are 0 as in S_1. However, that setup does not guarantee optimal behavior, and instead leads to random behavior instead.

Q7. [16 pts] Tom and Jerry

Bored at home, Tom and Jerry decided to play a game.
For this question, assume that branches are visited in left to right order.

(a) [3 pts] To analyze the game, Tom drew a game tree. However, Jerry quickly erased all the leaf node values in the tree, as shown below, and asked Tom to think about running alpha-beta pruning on game trees with the same structure.



(i) [2 pts]

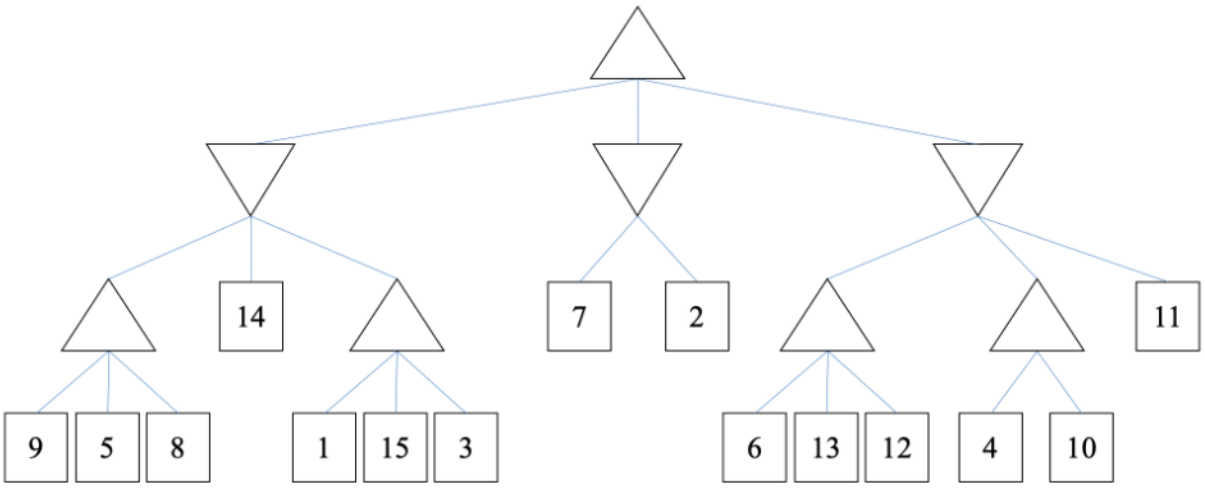
The maximum possible number of leaf nodes pruned = 6

(ii) [1 pt]

The minimum possible number of leaf nodes pruned = 0.

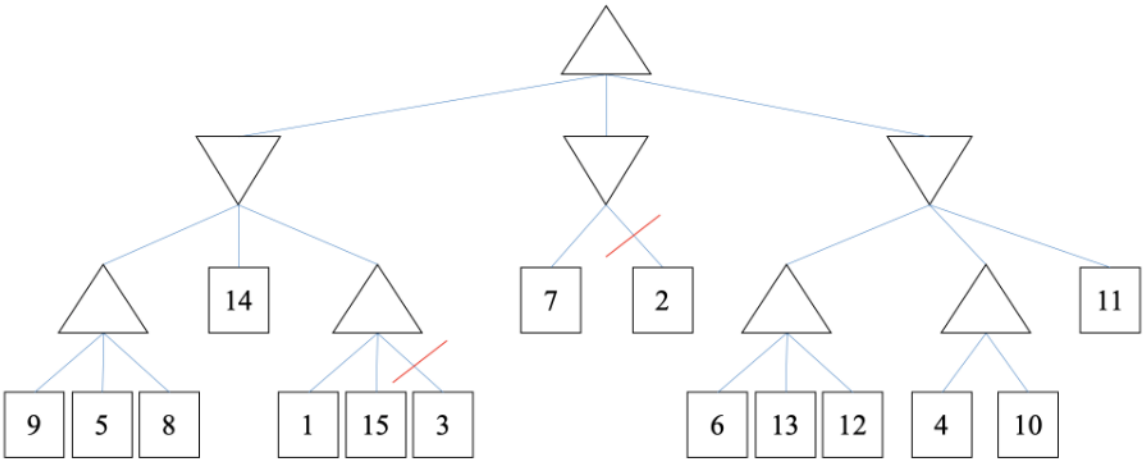
For the explanation, see part c.

Tom answered the questions correctly, so Jerry filled out the leaf node values.

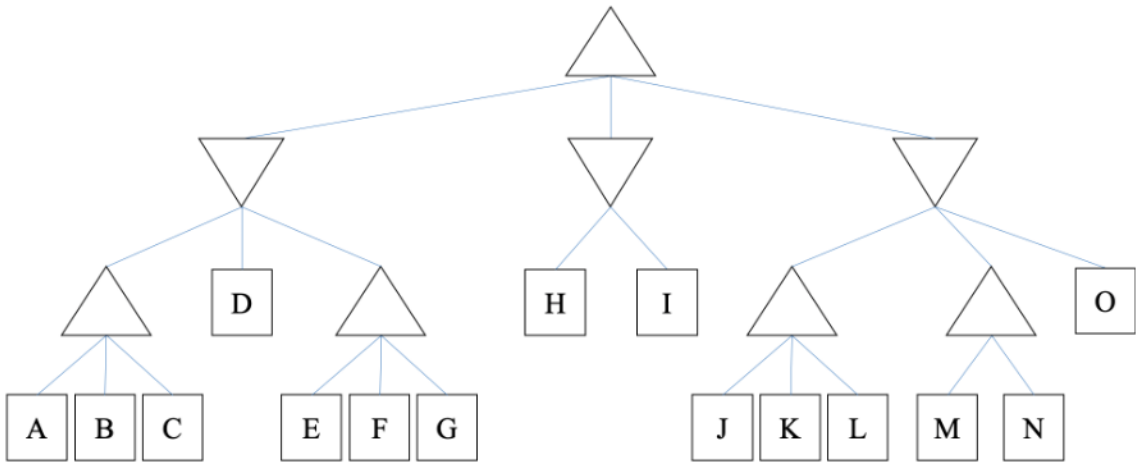


(b) [2 pts] Using alpha beta pruning, how many leaf nodes can be pruned?

2.



(c) [11 pts] Tom rearranges the **leaf node values from the tree above** such that alpha beta pruning prunes the maximum amount of leaf nodes. Each leaf node value could have been moved to any of the other leaf nodes (or not moved at all). Assume we are talking about such a tree for all subquestions in this part. **Note, there are 15 leaf nodes and 15 possible values from 1-15 so each value is used exactly once.**
Hint: Tom has a few options in how to rearrange leaves to get max pruning. Try to derive constraints on the values that need to hold for max pruning to happen.
Use this tree with place holder letters to answer the questions below.



The maximal pruning can have F, G, I, M, N and O pruned.
To get I pruned, H must be smaller than the left minimizer, which is also the root value.
To get F and G pruned, E must be larger than the root value, which is the smaller one between D or max(A, B, C).
To get M, N and O pruned, max(J, K and L) must be smaller than the root value, which means J, K and L are all smaller than the root value.

The largest result is therefore 13, where we have max(A, B, C), D, and max(E, F, G) each taking a value from 15, 14 and 13.

The smallest result is 5. We've shown that H, J, K and L are smaller than the root value. To make the root smallest, we should let D be smaller than A, B, and C. In this case, root = D, and is larger than H, J, K and L, so root = 5. If root = max(A, B, C), it is at least 7.

(i) [2 pts] If the value of the root is 8, which of the following leaf nodes are guaranteed to have value < 8?
☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ■ H ☐ I ■ J
■ K ■ L ☐ M ☐ N ☐ O ○ None of the above

(ii) [2 pts] If the value of the root is 8, which of the following leaf nodes are guaranteed to have value ≥ 8?
☐ A ☐ B ☐ C ■ D ■ E ☐ F ☐ G ☐ H ☐ I ☐ J
☐ K ☐ L ☐ M ☐ N ☐ O ○ None of the above

(iii) [2 pts] If the value of the root is 8, which of the following leaf nodes are guaranteed to have value 8?

☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J
☐ K ☐ L ☐ M ☐ N ☐ O ● None of the above

Since the root can come from max(A,B,C) or D neither can be guaranteed to be 8.

(iv) [2 pts] If the value of the root is 6, which of the following leaf nodes are guaranteed to have value 6?

☐ A ☐ B ☐ C ■ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J
☐ K ☐ L ☐ M ☐ N ☐ O ○ None of the above

There are 2 possibilities for the root: it is max(A, B, C), or it is D. If it is max(A, B, C), the root is at least 7. So if the root is 8, it could be either max(A, B, C) or D, so we cannot guarantee any of these leaf node values.
If the root is 6, it must be D.

(v) [3 pts] Which of the results (root values) are possible for Tom's rearranged game tree?

☐ 1 ☐ 2 ☐ 3 ☐ 4 ■ 5 ■ 6 ■ 7 ■ 8 ■ 9 ■ 10
■ 11 ■ 12 ■ 13 ☐ 14 ☐ 15