CS 188 Summer 2019 — Introduction to Artificial Intelligence — Midterm 1

- You have approximately 80 minutes.
- The exam is closed book, closed calculator, and closed notes except your one-page crib sheet.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation. All short answer sections can be successfully answered in a few sentences AT MOST.
- For multiple choice questions,
  - ☐ means mark **all options** that apply
  - ◯ means mark a **single choice**
  - When selecting an answer, please fill in the bubble or square **completely** (● and ■)

First name	
Last name	
SID	
Student to your right	
Student to your left	

Your Discussion TA (fill all that apply):

☐ Caryn (TTh 10:00am Wheeler)

☐ Bobby (MW 3:00 pm)

☐ Benson (MW 4:00 pm)

☐ Arin (TTh 10:00am Dwinelle)

☐ Mike(TTh 11:00am)

☐ Do not attend any

For staff use only:

Q1.	They See Me Rolling (Search Problem)	/21
Q2.	Search Algorithms Potpourri	/19
Q3.	MDP Solvers	/28
Q4.	Model-Free RL	/14
Q5.	Game Trees	/18
	Total	/100

THIS PAGE IS INTENTIONALLY LEFT BLANK

Q1. [21 pts] They See Me Rolling (Search Problem)

Pacman buys a car to start Rolling in the Pac-City! But driving a car requires a new set of controls because he can now travel faster than 1 grid per turn (second). Instead of solely moving [North, South, East, West, Stop], Pacman's car has two distinct integers to control: throttle, and steering.

Throttle: $t_i \in \{1, 0, -1\}$, corresponding to {Gas, Coast, Brake}. This controls the **speed** of the car by determining its acceleration. The integer chosen here will be added to his velocity for the next state. For example, if Pacman is currently driving at 5 grid/s and chooses Gas he will be traveling at 6 grid/s in the next turn.
Steering: $s_i \in \{1, 0, -1\}$, corresponding to {Turn Left, Neutral, Turn Right}. This controls the **direction** of the car. For example, if he is facing North and chooses Turn Left he will be facing West in the next turn.

(a) [11 pts] Suppose Pac-city has dimension m by n, but only $k < mn$ squares are legal roads. The speed limit of Pac-city is 3 grid/s. For this sub-part only, suppose Pacman is a law-abiding citizen, so $0 \leq v \leq 3$ at all time, and he only drives on legal roads.

(i) [3 pts] Without any additional information, what is the **tightest upper bound** on the size of state space, if he wants to search a route (not necessarily the shortest) from his current location to anywhere in the city. Please note that your state space representation must be able to represent **all** states in the search space.

○ $4mn$ ○ $4k$ ○ $12mn$ ○ $12k$ ○ $16mn$ ○ $16k$ ○ $48mn$ ○ $48k$

(ii) [3 pts] What is the maximum branching factor? The answer should be an integer.

[ ]

(iii) [3 pts] Which algorithm(s) is/are guaranteed to return a path between two points, if one exists?

☐ Depth First Tree Search ☐ Breadth First Tree Search
☐ Depth First Graph Search ☐ Breadth First Graph Search

(iv) [2 pts] Is Breadth First Graph Search guaranteed to return the path with the shortest grid distance?

○ Yes ○ No

(b) [5 pts] Now let's remove the constraint that Pacman follows the speed limit. Now Pacman's speed is limited by the mechanical constraints of the car, which is 6 grid/s, double the speed limit.

Pacman is now able to drive twice as fast on the route to his destination. How do the following properties of the search problem change as a result of being able to drive twice as fast?

(i) [1 pt] Size of State Space:

○ Increases ○ Stays the same ○ Decreases

(ii) [1 pt] Maximum Branching Factor:

○ Increases ○ Stays the same ○ Decreases

For the following part, **mark all choices that could happen on any graph**

(iii) [3 pts] The number of nodes expanded with **Depth** First Graph Search:

☐ Increases ☐ Stays the same ☐ Decreases

(c) [5 pts] Now we need to consider that there are $p > 0$ police cars waiting at $p > 0$ distinct locations trying to catch Pacman riding dirty!! All police cars are stationary, but once Pacman takes an action which lands him in the same grid as one police car, Pacman will be arrested and the game ends.

Pacman wants to find a route to his destination, without being arrested. How do the following properties of the search problem change as a result of avoiding the police?

(i) [1 pt] Size of State Space:
◯ Increases ◯ Stays the same ◯ Decreases

(ii) [1 pt] Maximum Branching Factor:
◯ Increases ◯ Stays the same ◯ Decreases

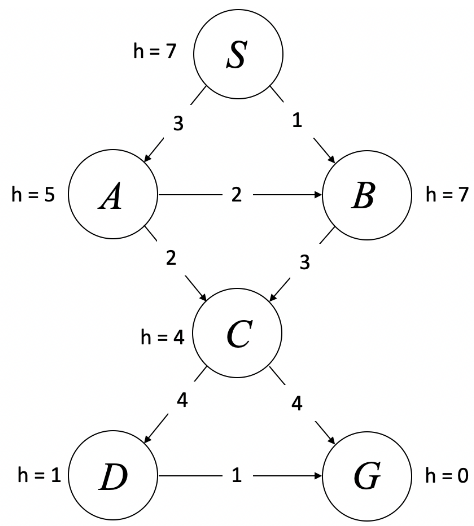
For the following part, **mark all choices that could happen on any graph**

(iii) [3 pts] Number of nodes expanded with **Breadth** First Graph Search:
☐ Increases ☐ Stays the same ☐ Decreases

SID: ____________________

Q2. [19 pts] Search Algorithms Potpourri

(a) [8 pts] We will investigate various search algorithms for the following graph. Edges are labeled with their costs, and heuristic values h for states are labeled next to the states. S is the start state, and G is the goal state. In all search algorithms, assume ties are broken in alphabetical order.



(i) [2 pts] Select all boxes that describe the given heuristic values.

☐ admissible ☐ consistent ☐ Neither

(ii) [2 pts] Given the above heuristics, what is the order that the states are going to be expanded in, assuming we run A* graph search with the heuristic values provided.

Index	1	2	3	4	5	Not Expanded
S	○	○	○	○	○	○
A	○	○	○	○	○	○
B	○	○	○	○	○	○
C	○	○	○	○	○	○
D	○	○	○	○	○	○
G	○	○	○	○	○	○

(iii) [1 pt] Assuming we run A* graph search with the heuristic values provided, what path is returned?

○ $S \to A \to B \to C \to D \to G$ ○ $S \to A \to C \to G$ ○ $S \to A \to C \to D \to G$
○ $S \to B \to C \to G$ ○ $S \to A \to C \to D \to G$ ○ $S \to A \to C \to D \to G$
○ $S \to A \to B \to C \to G$ ○ None of the above

(iv) [2 pts] Given the above heuristics, what is the order that the states are going to be expanded in, assuming we run greedy graph search with the heuristic values provided.

Index	1	2	3	4	5	Not Expanded
S	○	○	○	○	○	○
A	○	○	○	○	○	○
B	○	○	○	○	○	○
C	○	○	○	○	○	○
D	○	○	○	○	○	○
G	○	○	○	○	○	○

(v) [1 pt] What path is returned by greedy graph search?

○ $S \to A \to B \to C \to D \to G$ ○ $S \to A \to C \to G$ ○ $S \to A \to C \to D \to G$
○ $S \to A \to C \to D \to G$ ○ $S \to A \to C \to D \to G$ ○ None of the above

(b) [4 pts] Consider a complete graph, K_n, the undirected graph with n vertices where all n vertices are connected (there is an edge between every pair of vertices), resulting in $\binom{n}{2}$ edges. Please select the maximum possible depth of the resulting tree when the following **graph** search algorithms are run(assume any possible start and goal vertices).

	1	$\lceil \frac{n}{2} \rceil$	$n-1$	$\binom{n}{2}$	None of the above
BFS	○	○	○	○	○
DFS	○	○	○	○	○

(c) [3 pts] Given two admissible heuristics h_A and h_B.

(i) [2 pts] Which of the following are guaranteed to also be admissible heuristics?

☐ $h_A + h_B$ ☐ $\frac{1}{2}(h_A)$ ☐ $\frac{1}{2}(h_B)$ ☐ $\frac{1}{2}(h_A + h_B)$ ☐ $h_A * h_B$ ☐ $max(h_A, h_B)$
☐ $min(h_A, h_B)$

(ii) [1 pt] Consider performing A* **tree** search. Which is generally best to use if we want to expand the fewest number of nodes?

○ $h_A + h_B$ ○ $\frac{1}{2}(h_A)$ ○ $\frac{1}{2}(h_B)$ ○ $\frac{1}{2}(h_A + h_B)$ ○ $h_A * h_B$ ○ $max(h_A, h_B)$
○ $min(h_A, h_B)$

(d) [4 pts] Consider performing tree search for some search graph. Let $depth(n)$ be the depth of search node n and $cost(n)$ be the total cost from the start state to node n. Let G_d be a goal node with minimum depth, and G_c be a goal node with minimum total cost.

(i) [2 pts] For iterative deepening (where we repeatedly run DFS and increase the maximum depth allowed by 1), mark all conditions that are guaranteed to be true for every node n that could be expanded during the search, or mark "None of the above" if none of the conditions are guaranteed.

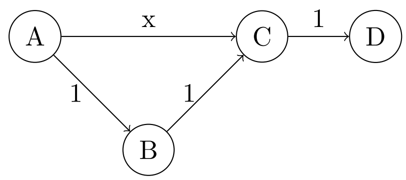
☐ $cost(n) \leq cost(G_c)$
☐ $cost(n) \leq cost(G_d)$
☐ $depth(n) \leq depth(G_c)$
☐ $depth(n) \leq depth(G_d)$
○ None of the above

(ii) [2 pts] What is necessarily true regarding iterative deepening on any search tree?

☐ Complete as opposed to DFS tree search
☐ Strictly faster than DFS tree search
☐ Strictly faster than BFS tree search
☐ More memory efficient than BFS tree search
☐ A type of stochastic local search
○ None of the above

Q3. [28 pts] MDP Solvers

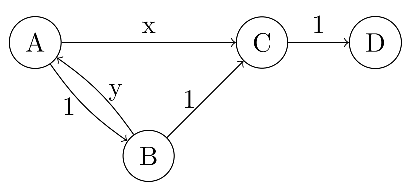
(a) [5 pts] Consider the following finite-state MDP, with states A, B, C, and D. Each edge indicates an action between states, and all transitions are deterministic. The edge weights denote the transition rewards. Assume $\gamma = 1$ for all parts.



For each of the states, let k be the **first** iteration in Value Iteration where the **values** could have converged for some value of x (in other words, the smallest k such that $V_k(s) = V^*(s)$). List **all** possible k for each state. Mark $k = \infty$ if the value for the state never converges.

State	k
A	
B	
C	
D	

(b) [11 pts] Consider the following finite-state MDP, with states A, B, C, and D. Each edge indicates an action between states, and all transitions are deterministic. Let $x, y \geq 0$. The edge weights denote the transition rewards. Assume $\gamma = 1$ for all parts.

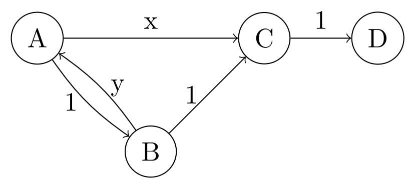


(i) [5 pts]

For each of the states, let k be the **first** iteration in Value Iteration where the values could have converged for some nonnegative **value** of x or y (in other words, the smallest k such that $V_k(s) = V^*(s)$). Find all possible k for each state. Mark $k = \infty$ if the value for the state never converges.

State	k
A	
B	
C	
D	

(ii) [6 pts] Additionally, assume $x+y<1$. For each of the states, let k be the **first** iteration in Policy Iteration where the **policy** is optimal for a particular state (in other words, the smallest k such that $\pi_k(s) = \pi^*(s)$). Find k fora each state. Write $k = \infty$ if the policy for the state never converges. Assume tie-breaking during policy improvement is alphabetical. (The graph is redrawn below for convenience)



Let π_0 be the following:

State (s)	Action ($\pi_0(s)$)
A	C
B	C
C	D
D	D

State	k
A	
B	
C	
D	0

(c) **(i)** [2 pts] Mark **all** of the statements that must be true for any MDP.

☐ For no state s and for all policies π, $V^*(s) \geq V^\pi(s)$

☐ For some state s and some policy π, $V^*(s) \geq V^\pi(s)$

☐ For all states s and all policies π, $V^*(s) \geq V^\pi(s)$

○ None of the above

(ii) [2 pts] Mark **all** of the statements that are true for value iteration.

☐ Each iteration of value iteration produces a value function that has higher value than the prior value functions **for all states**.

☐ Each iteration of value iteration produces a value function that has value at least as good as the prior value functions **for all states**.

☐ Value iteration can produce a value function that has lower value than the earlier value functions **for some state**.

☐ At convergence, the value iteration does not change the value function **for any state**.

○ None of the above

(d) [8 pts] Consider an MDP alternating game, where Pacman takes turns with Ghost to stochastically take actions.

Pacman takes actions to maximize the expected score, whereas Ghost takes actions to minimize the expected score. Both know the others strategy.

Let $V^*(s)$ be the value function for Pacman. Pacman and Ghost have identical transition, reward functions, and action space. (Discounting is applied for Pacmans actions)

Derive the new Bellman equation for $V^*(s)$ in terms of γ, R, T, and V^*:

$V^*(s) =$

(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)

For each blank (i) through (iv), mark the appropriate subexpression. If it is possible to write the expression for $V^*(s)$ without a particular sub-expression, mark "None".

(i) [1 pt] ○ $\frac{1}{2}$ ○ $\frac{1}{3}$ ○ 2 ○ -1 ○ 3 ○ γ ○ None

(ii) [1 pt] ○ $\max_a$ ○ $\min_a$ ○ None

(iii) [1 pt] ○ $\sum_{s'}$ ○ $\sum_{s'} T(s, a, s')$ ○ None

(iv) [1 pt] ○ $[$ ○ $[R(s, a, s')+$ ○ $[R(s, a, s')-$ ○ None

(v) [1 pt] ○ $\max_a$ ○ $\min_a$ ○ $\max_{a'}$ ○ $\min_{a'}$ ○ None

(vi) [1 pt] ○ $\sum_{s'}$ ○ $\sum_{s'} T(s, a, s')$ ○ $\sum_{s''}$ ○ $\sum_{s''} T(s', a', s'')$ ○ None

(vii) [1 pt] ○ $[$ ○ $[R(s, a, s')+$ ○ $[R(s, a, s')-$
○ $[R(s', a', s'')+$ ○ $[R(s', a', s'')-$ ○ None

(viii) [1 pt] ○ $\frac{1}{2}V^*(s')]]$ ○ $\gamma V^*(s')]]$ ○ $V^*(s')]]$ ○ $\frac{1}{2}V^*(s'')]]$
○ $\gamma V^*(s'')]]$ ○ $V^*(s'')]]$ ○ None

Q4. [14 pts] Model-Free RL

(a) [8 pts]
Consider the following MDP with state space $\mathcal{S} = \{A, B, C, D, E, F\}$ and action space $\mathcal{A} = \{left, right, up, down, stay\}$. However, we do not know the transition dynamics or reward function (we do not know what the resulting next state and reward are after applying an action in a state).

A	B	C
D	E	F

(i) [4 pts] We will run Q-learning to try to learn a policy that maximizes the expected reward in the MDP, with discount factor $\gamma = 1$ and learning rate $\alpha = 0.5$. Suppose we observe the following transitions in the environment. After observing each transition, we update the Q function, which is initially 0. Fill in the blanks with the corresponding values of the Q function after these updates.

Episode Number	State	Action	Reward	Next State
1	A	right	2	B
2	B	right	10	C
3	B	right	-3	E
4	C	stay	-5	A

State	Action	Q(state, action)
A	stay	
A	right	
B	right	
C	stay	

(ii) [4 pts] We now observe the following samples:

Episode Number	State	Action	Reward	Next State
5	B	down	-5	E
6	B	down	-5	E
7	B	up	-2	B
8	B	left	-8	B
9	B	left	-8	A
10	B	stay	-6	B
11	A	right	1	B

Continuing with the Q function from the last subproblem, we train the Q function on these samples. What is Q(A, right)?

(b) [6 pts] We are now given a policy π and would like to determine how good it is using Temporal Difference Learning with $\alpha = 0.5$ and $\gamma = 1$. We run it in the environment and observe the following transitions. After observing each transition, we update the value function, which is initially 0. Fill in the blanks with the corresponding values of the Value function after these updates.

Episode Number	State	Action	Reward	Next State
1	A	right	2	B
2	B	right	12	C
3	B	right	-8	E
4	C	down	-6	F
5	F	stay	12	F
6	C	down	-6	F

State	$V^{\pi}(state)$
A	
B	
C	
D	
E	
F	

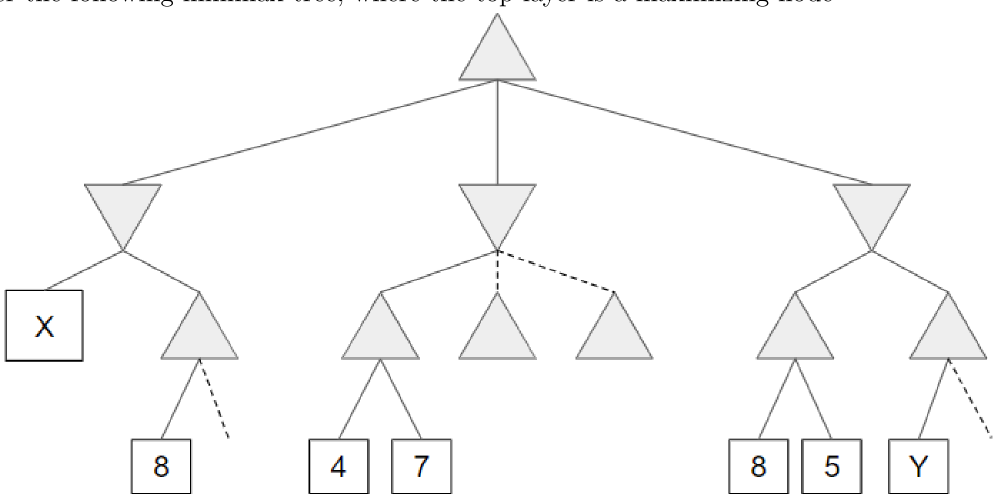
Q5. [18 pts] Game Trees

(a) [4 pts] Please mark if the following statements are true or false, and provide a brief justification or counterexample. Note a function, $f(x)$, is monotonically increasing if $f(a) > f(b)$ for all $a > b$.

(i) [2 pts] ◯ True ◯ False Alpha-beta pruning will prune the same branches after a monotonically increasing function is applied to the leaves.

(ii) [2 pts] ◯ True ◯ False Expectimax ordering will choose the same actions after a monotonically increasing function is applied to the leaves.

(b) [6 pts] Consider the following minimax tree, where the top layer is a maximizing node



Please determine the range of X and Y such that the dashed lines are pruned in Left-to-Right alpha beta pruning. The solid lines should not be pruned for the values in your range. You might find $-\infty$ and ∞ helpful. Note, pruning occurs from left to right and latter nodes are pruned in the case of ties.

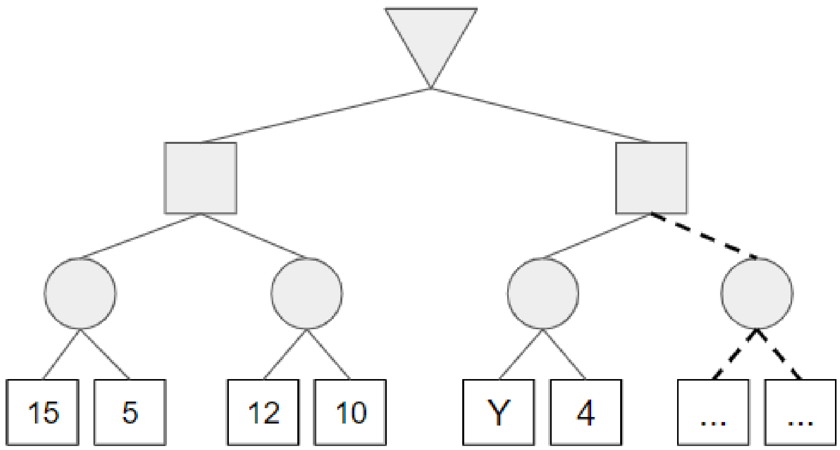
(i) [3 pts] The range of X will be

$\boxed{} \leq X < \boxed{}$

(ii) [3 pts] The range of Y will be

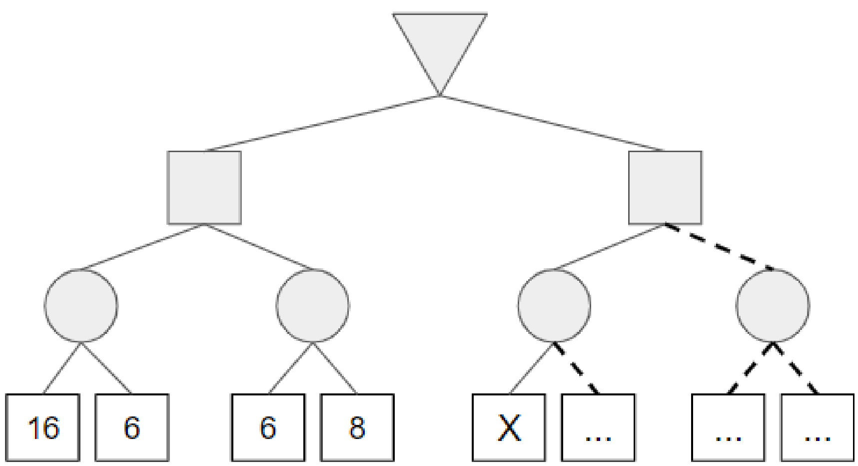
$\boxed{} \leq Y < \boxed{}$

(c) [8 pts] You are a casino owner (triangle) who is frequented by risky gamblers (squares). Your goal is to minimize the expected reward of the gamblers. Given the choice between lotteries, risky gamblers prefer the lottery (circle) that has the highest possible payout. Each reward under a lottery is equally likely. In the case of a tie between lotteries in maximum reward, the risky gambler will pick the lottery with the greater expected value. All rewards are **nonnegative integers**. Note, pruning occurs from left to right and latter nodes are pruned in the case of ties.



(i) [4 pts] For the scenario depicted in the above game tree, write the smallest possible Y such that the dotted branches are pruned (and no solid branches are pruned). If you believe no such value of Y exists, write "N/A".

Y =



(ii) [4 pts] For the scenario depicted in the above game tree, write the smallest possible X such that the all dotted branches are pruned (and no solid branches are pruned). If you believe no such value of X exists, write "N/A".

X =