CS 188 Fall 2019 — Introduction to Artificial Intelligence — Midterm

- You have approximately 110 minutes.
- The exam is closed book, closed calculator, and closed notes except your one-page crib sheet.
- Mark your answers ON THE EXAM ITSELF. Provide a *brief* explanation if applicable.
- 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/Exam Prep* TA(s) (fill all that apply):

☐ Shizhan	☐ Cathy	☐ Lindsay	☐ Andreea	☐ Jinkyu
☐ Carl	☐ Peyrin*	☐ Gokul*	☐ Chandan	
☐ Emma	☐ Andy	☐ Rachel	☐ Sherman*	☐ Lawrence
☐ Mesut*	☐ Wilson	☐ Henry*	☐ Mike	
☐ Jesse	☐ Ryan	☐ Alan	☐ Danny*	☐ Albert

For staff use only:

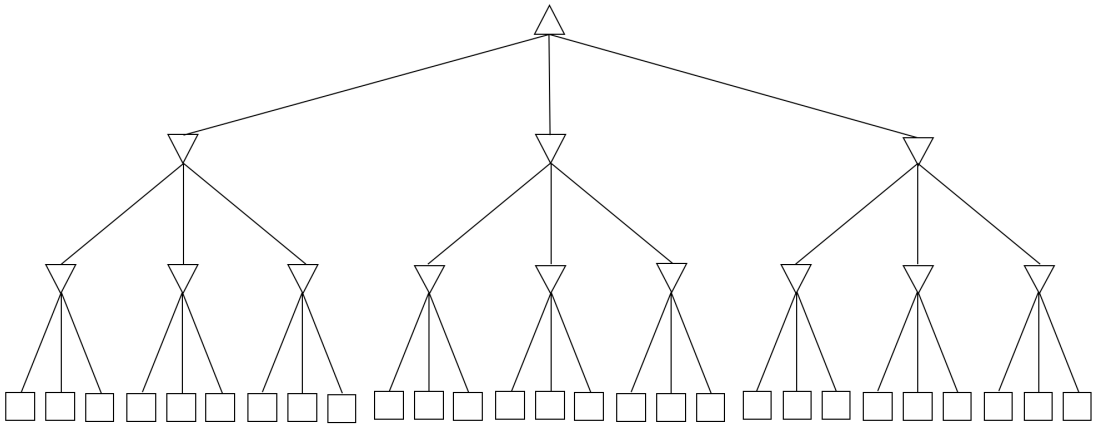
Q1.	Potpourri	/9
Q2.	CSP: The Picnic	/17
Q3.	MDP: Blackjack	/12
Q4.	RL: Blackjack, Redux	/17
Q5.	Games	/15
Q6.	Search: Snail search for love	/14
Q7.	Searching with Heuristics	/16
	Total	/100

THIS PAGE IS INTENTIONALLY LEFT BLANK

SID: _____

Q1. [9 pts] Potpourri

(a) [3 pts] We are given the following game tree.



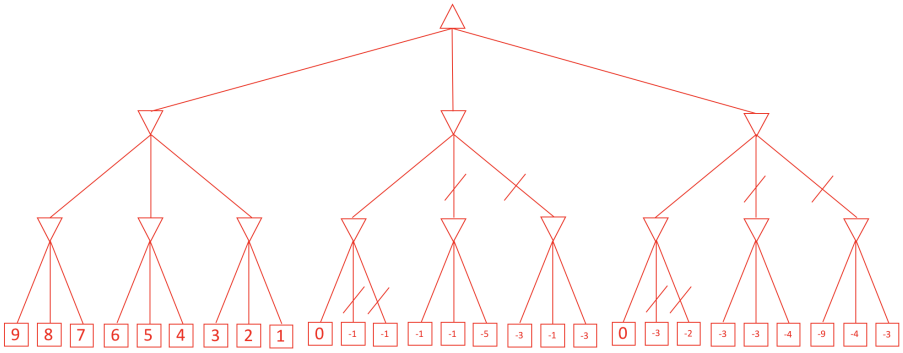
(i) [1 pt] What is the total number of **terminal nodes** (denoted by squares at bottom)? Your answer should be an integer.

Answer: 27

Count the diagram

(ii) [2 pts] Suppose you use alpha–beta pruning to prune branches from game tree. What is the **maximum** total number of terminal nodes **whose value is never explored because an immediate or upstream branch is pruned** in one single set of assignments for the terminal nodes. Your answer should be an integer.

Answer: 16



(b) [2 pts] **Arc consistency**

$X \longrightarrow Y$ is consistent if and only if _____(*i*)_____, _____(*ii*)_____

(i) [1 pt]

- ○ for some value of x in X
- ● for all values of x in X
- ○ for some value of y in Y
- ○ for all values of y in Y

(ii) [1 pt]

- ○ there exists some allowed x in X
- ○ all values of x in X are allowed
- ● there exists some allowed y in Y
- ○ all values of y in Y are allowed

(c) [4 pts] Which of the following algorithms is always guaranteed to provide the optimal solution for the corresponding problems they solve? We define "corresponding problem" as one of the following: uninformed/informed search, CSPs, MDPs, game trees, reinforcement learning.

- ☐ DFS
- ☐ Greedy search
- ☐ Hill climbing
- ☐ Simulated annealing
- ○ None of the above
- ■ Policy extracted from policy iteration
- ■ Policy extracted from value iteration
- ■ Minimax search with α–β pruning (assuming both players are playing optimally)

SID: _____

Q2. [17 pts] CSP: The Picnic

Six CS 188 TAs secretly scheduled a Picnic behind the back of other TAs! In order for the Picnic to be successful, Andy (A), Cathy (Ca), Chandan (Ch), Lindsay (L), Mesut (Me), and Mike (Mi) will each be assigned 1 of the 5 tasks. Please note that one task requires 2 TAs. In this question, the TAs are the variables, and the domains are the tasks $\{1, \ldots, 5\}$. Your mission is to use your knowledge of CSPs to find an assignment that meets the following constraints:

- Only Andy (A) is capable of task 4.
- Cathy (Ca) and Mesut (Me) must be assigned adjacent (difference in number is 1) tasks. (e.g. task 4 and task 5 are adjacent, but task 1 and task 5 are not)
- Chandan (Ch) can only do odd-numbered tasks.
- Lindsay (L) can only do tasks with number ≤ 2.
- Mesut (Me) can only do either task 2 or task 5.
- Mike (Mi) has to take a task with a bigger number than Lindsay (L)'s.
- Task 2 needs exactly 2 TAs.
- Every task other than task 2 needs exactly 1 TA.
- The 2 TAs doing task 2 needs to have the same initial letter.

(a) [8 pts] Let's start with looking into the constraints

(i) [1 pt] What type of constraint is *Mike (Mi) has to take a task with a bigger number than Lindsay (L)'s*?
○ Unary Constraint ● Binary Constraint ○ Higher Order Constraint
This constraints uniquely involves two variables.

(ii) [1 pt] What type of constraint is *Only Andy (A) is capable of task 4*?
● Unary Constraint ○ Binary Constraint ○ Higher Order Constraint
It eliminates 4 from the domain of everyone who is not Andy.

(iii) [1 pt] What type of constraint is *Task 2 needs exactly 2 TAs*?
○ Unary Constraint ○ Binary Constraint ● Higher Order Constraint
This constrain can be written as $(Me2 \wedge Mi2) \vee (Ca2 \wedge Ch2)$, which cannot be re-written in a clause that only involve 2 variables.

(iv) [3 pts] Please select the elements in the domains that will be **crossed out** after enforcing unary constraints.

	1	2	3	4	5
A	☐	☐	☐	☐	☐
Ca	☐	☐	☐	■	☐
Ch	☐	■	☐	■	☐
L	☐	☐	■	■	■
Me	■	☐	■	■	☐
Mi	☐	☐	☐	■	☐

1, 3, 4, 5 are the only Unary Constraints. Constraint 1 crosses out Ca4, Ch4, L4, Me4m Mi4. Constraint 3 crosses out Ch2, Ch4. Constraint 4 crosses oout L3, L4, L5. Constraint 5 crosses out Me1, Me3, Me4.

(v) [2 pts] Please select the elements in the domains that will be **crossed out** after enforcing unary constraints and binary constraints through **arc-consistency**. Ignore high order constraints. (Note: this part will be graded **independently**, which means no partial credit if your answer is incorrect due to errors in previous parts. So please double check the correctness of previous parts.)

A	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Ca	☐ 1	■ 2	☐ 3	■ 4	■ 5
Ch	☐ 1	■ 2	☐ 3	■ 4	☐ 5
L	☐ 1	☐ 2	■ 3	■ 4	■ 5
Me	■ 1	☐ 2	■ 3	■ 4	■ 5
Mi	■ 1	☐ 2	☐ 3	■ 4	☐ 5

Me5 , Ca2, Ca5 because of constraint 2, Mi1 because of constraint 6, in combination of 4

(b) [2 pts] Heuristics in CSPs could help you find a satisfying assignment more quickly. Which of the heuristics fits each of the given descriptions?

(i) [1 pt] Prioritize assigning a variable that has the minimum number of remaining values in its domain

● MRV ○ LCV ○ Both ○ Neither

(ii) [1 pt] Prioritize assigning a variable that is involved in the least number of constraints

○ MRV ○ LCV ○ Both ● Neither

LCV is prioritize assigning a variable to a value that will eliminate the least number of values in other variables domains

(c) [2 pts] Suppose you can't use either of those helpful heuristics because you ran out of time implementing them [sad reacts only :( ] and decide to run the backtracking search algorithm manually. Fortunately, Mike told you an extra constraint: he (Mi) has to be assigned to task 2 at the picnic.

(i) [1 pt] Using this additional information, could you find at least one assignment that satisfies all the constraints?

● Yes ○ No

(ii) [1 pt] If yes, which task(s) can Cathy possibly get assigned to?

☐ 1 ☐ 2 ■ 3 ☐ 4 ☐ 5 ○ Not applicable

(d) [5 pts] Now suppose all N CS 188 students and staff are going on the trip together. There are D tasks that need exactly 2 people, while all other tasks need exactly 1 person. In addition, there are $b > 0$ binary constraints, and $h > 0$ higher-order constraints. There exists a satisfying assignment where everyone gets one and only one task, and all tasks are filled to the required number.

Since there are D tasks that need 2 exact people, and all other need 1, there are $N - D$ tasks in total, which means the domain size is $d = N - D$. for all N variables

(i) [1 pt] Without further assumptions, which of the following is the tightest upper bound for the runtime for finding a satisfying assignment?

○ $O(D^N)$ ○ $O(N^D)$ ● $O((N-D)^N)$ ○ $O(N^{(N-D)})$ ○ $O(ND^2)$
○ $O(N(N-D)^2)$ ○ $O(N^2D)$ ○ $O(N^2(N-D))$ ○ $O(N^2D^3)$ ○ $O(N^2(N-D)^3)$

Since backtracking in the worst case has runtime $O(d^N)$

(ii) [2 pts] We **remove all the higher-order constraints**. Without further assumptions, which of the following is the tightest upper bound for the runtime for the AC-3 Algorithm in this setting?

○ $O(D^N)$ ○ $O(N^D)$ ○ $O((N-D)^N)$ ○ $O(N^{(N-D)})$ ○ $O(ND^2)$
○ $O(N(N-D)^2)$ ○ $O(N^2D)$ ○ $O(N^2(N-D))$ ○ $O(N^2D^3)$ ● $O(N^2(N-D)^3)$

Please note that even if we remove all higher-order constraints, we can run AC-3 with time $O(n^2d^3) = O(N^2(N-D)^3)$

(iii) [2 pts] Suppose the binary constraints turn out to be very sparse ($b << N^2$). After **removing all the higher-order constraints**, without further assumptions, which of the following is the tightest upper bound for the runtime for the AC-3 Algorithm in this setting now?

○ $O(D^{\sqrt{b}})$ ○ $O(\sqrt{b}^D)$ ○ $O((N-D)^{\sqrt{b}})$ ○ $O(\sqrt{b}^{(N-D)})$ ○ $O(\sqrt{b}D^2)$
○ $O(\sqrt{b}(N-D)^2)$ ○ $O(bD)$ ○ $O(b(N-D))$ ○ $O(bD^3)$ ● $O(b(N-D)^3)$

The n^2 in $O(n^2d^3)$ comes from a bound on the total number of arcs ($\frac{n(n-1)}{2} = O(n^2)$). But since we know b, we can replace n^2 with b to get a better bound on the runtime $O(bd^3) = O(b(N-D)^3)$

SID: _____

Q3. [12 pts] MDP: Blackjack

There's a new gambling game popping up in Vegas! It's similar to blackjack, but it's played with a single die. CS188 staff is interested in winning a small fortune, so we've hired you to take a look at the game!

We will treat the game as an MDP. The game has states 0,1,…,8, corresponding to dollar amounts, and a *Done* state where the game ends. The player starts with \$2, i.e. at state 2. The player has two actions: Stop and Roll, and is forced to take the Stop action at states 0,1,and 8.

When the player takes the Stop action, they transition to the *Done* state and receive reward equal to the amount of dollars of the state they transitioned from: e.g. taking the stop action at state 3 gives the player \$3. The game ends when the player transitions to *Done*.

The Roll action is available from states 2-7. The player rolls a **biased** 6-sided die that will land on 1, 2, 3, or 4 with $\frac{1}{8}$ probability each and 5 or 6 with probability $\frac{1}{4}$ each.

If the player Rolls from state s and the die lands on outcome o, the player transitions to state $s + o - 2$, as long as $s + o - 2 \leq 8$ (s is the amount of dollars of the current state, o is the amount rolled, and the negative 2 is the price to roll). If $s + o - 2 > 8$, the player busts, i.e. transitions to Done and does NOT receive reward.

(a) [4 pts] In solving this problem, you consider using policy iteration. Your initial policy π^a is in the table below. Evaluate the policy at each state, with $\gamma = 1$. Note that the action at state 0, 1, 8 is fixed into the rule, so we will not consider those states in the update. (*Hint: how does the bias in the die affect this?*)

State	2	3	4	5	6	7
$\pi^a(s)$	Roll	Roll	Stop	Stop	Stop	Stop
$V^{\pi^a(s)}$	14/3	17/3	4	5	6	7

We can write a system of equations to solve for the value of being in states 2 and 3. First, we find that the value of being in state 2 is: $V(2) = \frac{1}{8}V(1) + \frac{1}{8}V(2) + \frac{1}{8}V(3) + \frac{1}{8}V(4) + \frac{2}{8}V(5) + \frac{2}{8}V(6)$. Plugging in the values of stopping at states 4, 5, and 6, we find that $7V(2) = V(3) + 27$. Similarly, for state 3: $V(3) = \frac{1}{8}V(2) + \frac{1}{8}V(3) + \frac{1}{8}V(4) + \frac{1}{8}V(5) + \frac{2}{8}V(6) + \frac{2}{8}V(7)$. This simplifies to $7V(3) = V(2) + 35$. Solving this system of equations gives $V(2) = \frac{14}{3}$ and $V(3) = \frac{17}{3}$.

(b) [4 pts] Deciding against the previous policy, you come up with a simpler policy $\pi^{(0)}$, as shown below, to start with. Perform one iteration of Policy Iteration (i.e. policy evaluation followed by policy improvement) to find the new policy $\pi^{(1)}$. In this part as well, we have $\gamma = 1$.

In the table below, R stands for *Roll* and S stands for *Stop*. Select both R and S if both actions are equally preferred.

State	2	3	4	5	6	7
$\pi^{(0)}(s)$	Stop	Stop	Stop	Stop	Stop	Stop
$\pi^{(1)}(s)$	■ R ☐ S	■ R ☐ S	■ R ☐ S	☐ R ■ S	☐ R ■ S	☐ R ■ S

We compare the values obtained by either rolling or stopping. Stopping in a state i yields a value of i. The value of rolling will be an expectation over the value of the states we could land in. At each state, we take the action that yields the highest reward.

Note: we accept both "R only" and "R and S" for state 4. If we don't consider 8 as a state, "R and S" is the correct answer; if we consider 8 as a state, "R" is the correct answer

(c) [2 pts] Suppose you start with a initial policy π_0 that is the **opposite** of the optimal policy (which means if $\pi^*(s) = Roll$, $\pi_0(s) = Stop$, and vice versa). Your friend Alice claims that the Policy Iteration Algorithm can still find the optimal policy in this specific scenario. Is Alice right?

- (●) Alice is right, because Policy Iteration can find the optimal policy regardless of initial policy.
- ( ) Alice is right, but not for the reason above.
- ( ) Alice is wrong, because a very bad initial policy can block the algorithm from exploring the optimal actions.
- ( ) Alice is wrong, but not for the reason above.

(d) [2 pts] Suppose you want to try a different approach, and implement a **value iteration** program to find the optimal policy for this new game. Your friend Bob claims that $V_k(s)$ has to converge to $V^*(s)$ for all states before the program declares it has found the optimal policy. Is Bob right?

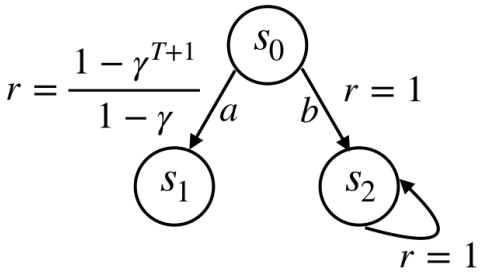
- ( ) Bob is right, because $V_k(s)$ always converge to $V^*(s)$ for all states when the optimal policy π_* is found.
- ( ) Bob is right, but not for the reason above.
- ( ) Bob is wrong, because we cannot use value iteration to find the optimal policy.
- (●) Bob is wrong, but not for the reason above

Note that the policy extracted from value iteration can converge to the optimal policy long before the values themselves converge. (We gave an example of this in class.)

However, based on the unclear wording, choice 2 was technically the most correct answer (both choice 2 and 4 will receive full credit). This is because the value iteration algorithm has no general way of detecting if the policy has converged to the optimal policy, except insofar as the values have converged.

One common misunderstanding is that the policy has converged to the optimal policy if we see $\pi_k = \pi_{k+1} = \cdots = \pi_{k+T}$ for some large T. This is actually not the case.

For example, consider the following 2-state MDP, where T is a positive integer and $0 < \gamma < 1$:



We have $Q^*(s_0, a) = \frac{1-\gamma^{t+1}}{1-\gamma}$, $Q^*(s_0, b) = \frac{1}{1-\gamma}$. So $\pi^*(s_0) = b$.

After k value iteration steps, $V_k(s_2) = 1 + \gamma + \gamma^2 + \cdots + \gamma^{k-1} = \frac{1-\gamma^k}{1-\gamma}$. So the Q value of b extracted after k value iteration steps is $Q_k(s_0, b) = 1 + \gamma V_k(s_2) = \frac{1-\gamma^{k+1}}{1-\gamma}$. For $k \leq T$, we have $Q_k(s_0, a) \geq Q_k(s_0, b)$ This means that, if we tiebreak actions alphabetically, we will have $\pi_0 = \pi_1 = \cdots = \pi_T$. However, for $k > T$, we have $Q_k(s_0, a) < Q_k(s_0, b)$, and therefore $\pi_T \neq \pi_{T+1}$. So in general, observing that the extracted policy has remained the same for many timesteps does not allow you to infer that the policy has converged to the optimal policy.

SID: _____

Q4. [17 pts] RL: Blackjack, Redux

After playing the Blackjack game in Q3 a few times with the optimal policy you found in the previous problem, you find that you're doing worse than expected! (Hint: you may want to do Q3 before attempting this problem.) In fact, you are beginning to suspect that the Casino was not honest about the probabilities of dice's outcome. Seeing no better option, you decided to do some good old fashioned reinforcement learning (RL).

(a) First, you need to decide what RL algorithm to use.

(i) [2 pts] Suppose you had a policy π and wanted to find the value V^π of each of the states under this policy. Which algorithms are appropriate for performing this calculation? Note that we **do not** know the transition probabilities, and we don't have sufficient samples to approximate them.

☐ Value Iteration ☐ Policy Iteration ☐ Q-learning ■ Direct Evaluation ■ Temporal difference learning

We cannot use Value Iteration or Policy Iteration because we don't have the transition probabilities. In addition, we cannot use Q-learning, because it estimates the value of the optimal policy.

(ii) [2 pts] Being prudent with your money, you decide to begin with observing what happens when other people randomly play the blackjack game. Which of the following algorithms can recover the optimal policy given this play data?

☐ Value Iteration ☐ Policy Iteration ■ Q-learning ☐ Direct Evaluation ☐ Temporal difference learning

Q-learning estimates the value of the optimal policy, even given data generated from a random policy.

(b) You decide to use Q-learning to play this game.

(i) [2 pts] Suppose your initial policy is π_0. Which of the following is the update performed by Q-learning with learning rate α, upon getting reward $R(s, a, s')$ and transitioning to state s' after taking action a in state s?

● $Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
○ $Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha(R(s,a,s') + \gamma Q_k(s',\pi_0(s')))$
○ $V_{k+1}(s) = (1-\alpha)V_k(s) + \alpha(R(s,a,s') + \gamma \max_{s''} V_k(s''))$
○ $V_{k+1} = (1-\alpha)V_k + \alpha(R(s,a,s') + \gamma V_k(s'))$

(ii) [2 pts] As with the previous problem, denote a policy at any time-step k as π_k (and $\pi_k(a|s)$ means the **probability** of taking action a at state), and the Q values at that timestep as Q_k. In the limit of infinite episodes, which of these policies will always do each action in each state an infinite amount of times?

■ $\pi_k(Roll|s) = \pi_k(Stop|s) = \frac{1}{2}$
■ $\pi_k(a|s) = 1 - \frac{\epsilon}{2}$ if $a == \arg\max_a Q_k(s,a)$ else $\frac{\epsilon}{2}$
☐ $\pi_k(Roll|s) = 1, \pi_k(Stop|s) = 0$
■ $\pi_k(Roll|s) = \frac{1}{3}, \pi_k(Stop|s) = \frac{2}{3}$
○ None of the above

(iii) [3 pts] Suppose you decide to use an exploration function $f(s', a')$, used in-place of $Q(s', a')$ in the Q-learning update. Which of the following choices of an exploration functions encourage you to take actions you haven't taken much before? (Recall that $N(s, a)$ is the number of times the q-state (s, a) has been visited, assuming every (s, a) has been visited at least once.)

☐ $f(s,a) = Q(s,a)$
☐ $f(s,a) = Q(s,a) + N(s,a)$
☐ $f(s,a) = \max_{a'} Q(s,a')$
■ $f(s,a) = Q(s,a) + \frac{k}{N(s,a)}$, where $k > 0$
■ $f(s,a) = Q(s,a) + \sqrt{\frac{\log(\sum_{a'} N(s,a'))}{N(s,a)}}$
■ $f(s,a) = \frac{1}{N(s,a)^2}$
○ None of the above

(iv) [3 pts] Suppose you start with the following Q-value table:

State	2	3	4	5	6	7
Q(State, Roll)	0	0	5	3	4	2
Q(State, Stop)	2	3	4	5	6	7

After you observe the trajectory

$$(s = 2, a = \text{Roll}, s' = 4, r = 0), (s = 4, a = \text{Roll}, s' = 7, r = 0), (s = 7, a = \text{Stop}, s' = \textit{Done}, r = 7)$$

What are the resulting Q-values after running one pass of Q-learning over the given trajectory? Suppose discount rate $\gamma = 1$, and learning rate $\alpha = 0.5$.

State	2	3	4	5	6	7
Q(State, Roll)	2.5	0	6	3	4	2
Q(State, Stop)	2	3	4	5	6	7

We only update the Q-values for state, action pairs we observe. So only $Q(2, R), Q(4, R), Q(7, S)$ are changed. By performing the bellman backups, we get:

$$\textit{sample}(2, \textit{Roll}, 4) = 0 + 1 * (5) = 5 \qquad Q'(2, R) = 0.5(0) + 0.5(5) = 2.5$$
$$\textit{sample}(4, \textit{Roll}, 7) = 0 + 1 * (7) = 7 \qquad Q'(4, R) = 0.5(5) + 0.5(7) = 6$$
$$\textit{sample}(7, \textit{Stop}, \textit{Done}) = 7 + 1 * (0) = 7 \qquad Q'(7, S) = 0.5(7) + 0.5(7) = 7$$

(v) [1 pt] One of the other gamblers looks over your shoulder as you perform Q-learning, and tells you that you're learning too slowly. "You should use a learning rate of $\alpha = 1$", they suggest.

If you use constant $\alpha = 1$, is Q-learning guaranteed to eventually converge to the optimal policy, assuming you observe every state, action pair an infinite amount of times? ○ Yes ● No

With $\alpha = 1$, your estimate of Q values is simply the latest sample. This means that the greedy policy flip flops depending on your latest observation. For example, if you observe $(s = 5, a = \text{Roll}, s' = 8, r = 0)$, you will have $Q(5, \text{Roll}) = 8$, causing the greedy policy with respect to the Q-values to be $\pi^{\text{sGreedy}}(5) = \text{Roll}$. However, you might then observe $(s = 5, a = \text{Roll}, s' = \textit{Done}, r = 0)$, therefore giving you $Q(5, \text{Roll}) = 0$. This causes the greedy policy with respect to the Q-values to be $\pi^{\text{Greedy}}(5) = \text{Stop}$.

So your policy is not guaranteed to converge.

(vi) [2 pts] If you continue with constant $\alpha = 0.5$, is Q-learning guaranteed to eventually converge to the optimal policy, assuming you observe every state, action pair an infinite amount of times? ○ Yes ● No

With $\alpha = 0.5$, your estimate of Q values is highly influenced by your latest samples. This means that the greedy policy might flip-flop depending on your latest observations.

For example, after observing enough transitions, we'll have $Q(s, \text{Stop}) \in [s - 0.2, s + 0.2]$ and have $Q(s, \text{Roll}) \leq 10$ for all s.

If you observe 2 transitions of $(s = 5, a = \text{Roll}, s' = 8, r = 0)$, you will have $Q(5, \text{Roll}) \geq \frac{3}{4}(7.8) + \frac{1}{4}(-0.2) = 5.8 > 5.2 \geq Q(5, \text{Stop})$, causing the greedy policy with respect to the Q-values to be $\pi^{\text{Greedy}}(5) = \text{Roll}$. However, you might then observe 2 transitions of $(s = 5, a = \text{Roll}, s' = \textit{Done}, r = 0)$, therefore giving you $Q(5, \text{Roll}) \leq \frac{3}{4}0 + \frac{1}{4}10 \leq 2.5 < 4.8 \leq Q(5, \text{Stop})$. This causes the greedy policy with respect to the Q-values to be $\pi^{\text{Greedy}}(5) = \text{Stop}$.

So your policy is not guaranteed to converge.

SID: _____

Q5. [15 pts] Games

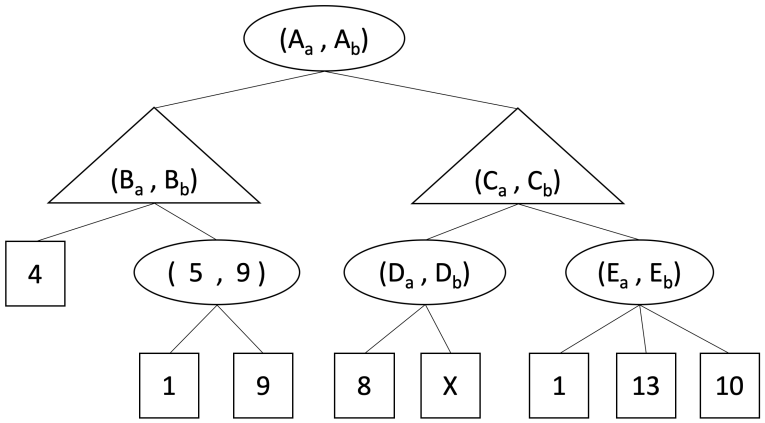
Alice is playing a two-player game with Bob, in which they move alternately. Alice is a maximizer. Although Bob is also a maximizer, Alice believes Bob is a minimizer with probability 0.5, and a maximizer with probability 0.5. Bob is aware of Alice's assumption.

In the game tree below, square nodes are the outcomes, triangular nodes are Alice's moves, and round nodes are Bob's moves. Each node for Alice/Bob contains a tuple, the left value being Alice's expectation of the outcome, and the right value being Bob's expectation of the outcome.

Tie-breaking: choose the left branch.

The left values are Alice's expectations, and are the only thing Alice can refer to when making decisions.

The right values are Bob's expectations, and they also accurately track the expected outcome of the game according to each choice of branching (regardless of it is Alice's or Bob's decision, since Bob has all the information). Hence the right values are accurate information about the game, and would be what Bob looks at when making his decisions. However, when it is Alice's turn to make decisions, Bob will think about how Alice would maximize the outcome w.r.t to what she believes, and he will update his expectations accordingly.



(a) [2 pts] In the blanks below, fill in the tuple values for tuples (B_a, B_b) and (E_a, E_b) from the above game tree.

$(B_a, B_b) = ($ 5 , 9 $)$

$(E_a, E_b) = ($ 7 , 13 $)$

For a square node, its value v means the same to Alice and Bob, i.e., we can think of it as a tuple (v,v).

The left value of Alice's nodes is the maximum of the left values of it's children nodes, since Alice believes that the values of the nodes are given by left values, and it's her turn of action, so she will choose the largest value.

The right value of Alice's nodes is the right value from the child node that attains the maximum left value since Bob's expectation is consistent with how Alice will act.

So for a triangular node, its tuple is the same as its child that has the maximum left value.

The left value of Bob's nodes is the average of the maximum and minimum of the left values of it's children nodes since Alice believes Bob is 50% possible to be adversarial and 50% possible to be friendly.

The right value of Bob's nodes is the maximum of the right values of the immediate children nodes since Bob would choose the branch that gives the maximum outcome during his turn.

So for a round node, left = 0.5(max(children.left) + min(children.left)), and right = max(children.right).

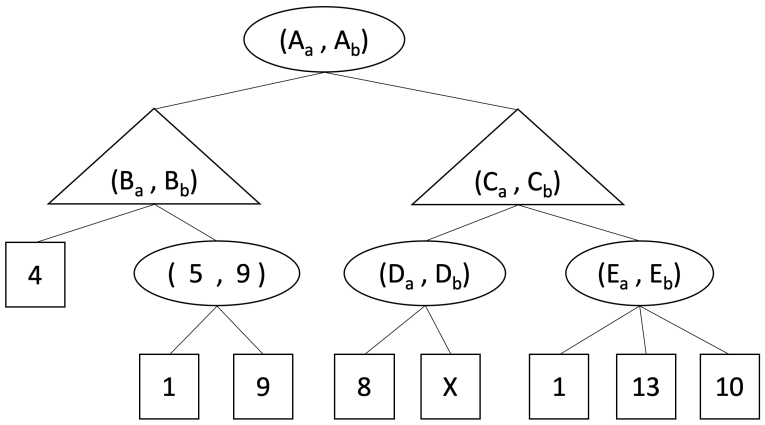
(b) [2 pts] In this part, we will determine the values for tuple (D_a, D_b).

(i) [1 pt] $D_a =$ ○ 8 ○ X ○ 8+X ● 4+0.5X ○ min(8,X) ○ max(8,X)

(ii) [1 pt] $D_b =$ ○ 8 ○ X ○ 8+X ○ 4+0.5X ○ min(8,X) ● max(8,X)

It's a round node, so left = 0.5(max(children.left) + min(children.left)), and right = max(children.right).

Its children: (8,8) and (X,X). So left = 0.5(8+X) = 4+0.5X, and right = max(8, X).



(The graph of the tree is copied for your convenience. You may do problem e on this graph.)

(c) [6 pts] Fill in the values for tuple (C_a, C_b) below. For the bounds of X, you may write scalars, ∞ or $-\infty$.
If your answer contains a fraction, please write down the corresponding **simplified decimal value** in its place. (i.e., 4 instead of $\frac{8}{2}$, and 0.5 instead of $\frac{1}{2}$).

1. If $-\infty < X <$ [6], $(C_a, C_b) =$ ([7] , [13])
2. Else, $(C_a, C_b) =$ ([4+0.5X] , max([8] , [X]))

It's a triangular node, so its tuple is the same as its child that has the maximum left value.
Its children: (4+0.5X, max(8,X)) and (7, 13).
So if 4+0.5X < 7, i.e. $-\infty < X < 6$, it's the same as child node (7, 13), and otherwise it's (4+0.5X, max(8,X)).

(d) [4 pts] Fill in the values for tuple (A_a, A_b) below. For the bounds of X, you may write scalars, ∞ or $-\infty$.
If your answer contains a fraction, please write down the corresponding **simplified decimal value** in its place. (i.e., 4 instead of $\frac{8}{2}$, and 0.5 instead of $\frac{1}{2}$).

1. If $-\infty < X <$ [6], $(A_a, A_b) =$ ([6] , [13])
2. Else, $(A_a, A_b) =$ ([4.5+0.25X] , max([9] , [X]))

It's a round node, so left = 0.5(max(children.left) + min(children.left)), and right = max(children.right).
Its children: (5,9) and node "Part (c)".
If $-\infty < X < 6$, these children are (5,9) and (7, 13).
left = 0.5(max(children.left) + min(children.left)) = 0.5(5+7) = 6
right = max(children.right) = max(9, 13) = 13.
Otherwise ($6 < X < +\infty$), these children are (5,9) and (4+0.5X, max(8,X)).
left = 0.5(max(children.left) + min(children.left)) = 0.5(5+4+0.5X) = 4.5 + 0.25X
right = max(children.right) = max(9, max(8,X)) = max(9,X).

(e) [1 pt] When Alice computes the left values in the tree, some branches can be pruned and do not need to be explored. In the game tree graph on this page, put an 'X' on these branches. If no branches can be pruned, mark the "Not possible" choice below.
Assume that the children of a node are visited in left-to-right order and that you should not prune on equality.

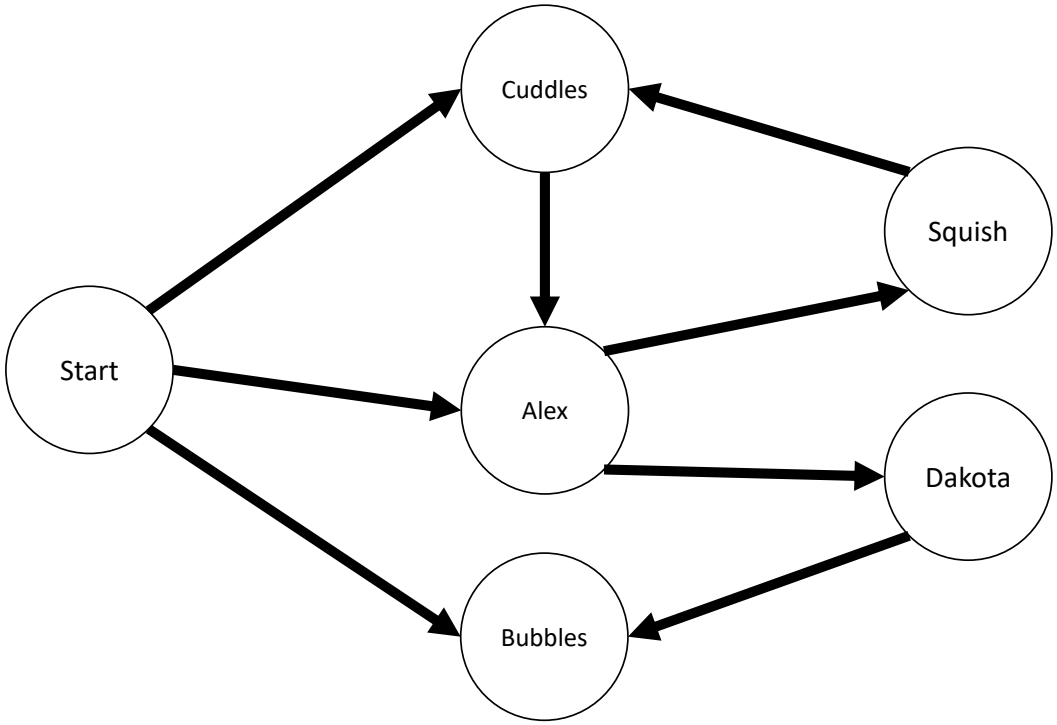
● Not possible

It's impossible to determine the average of *min* and *max* until all children nodes are seen, so no pruning can be done for Alice. Leaving "Not possible" unmarked and no 'X' found in the graph is interpreted as 'no conclusion' and will not be given credit.

SID: _____

Q6. [14 pts] Search: Snail search for love

Scorpblorg the snail is looking for a mate. It can visit different potential mates based on a trail of ooze to nearby snails, and then test them for chemistry, as represented in the below graph, where each node represents a snail. In all cases, nodes with equal priority should be visited in alphabetical order.



(a) [5 pts] **Simple search**
In this part, assume that the only match for Scorpblorg is Squish (i.e. Squish is the goal state). Which of the following are true **when searching the above graph**?

(i) [1 pt] BFS Tree Search expands more nodes than DFS Tree Search ● True ○ False

DFS Tree Search expands the path Alex, then Dakota, then Bubbles, then Squish. In contrast, BFS Tree Search expands Alex, Bubbles, Cuddles, Alex, and Dakota before opening Squish.

(ii) [1 pt] DFS Tree Search finds a path to the goal for this graph ● True ○ False

DFS Tree Search does not get stuck in any loops on this graph and does return the solution path.

(iii) [1 pt] DFS Graph Search finds the shortest path to the goal for this graph ● True ○ False

DFS Graph Search does return the shortest solution path.

(iv) [2 pts] If we remove the connection from Cuddles → Alex, can DFS Graph Search find a path to the goal for the altered graph? ● Yes ○ No

Yes, DFS Graph Search will return the correct path, regardless of the connection from Cuddles → Alex.

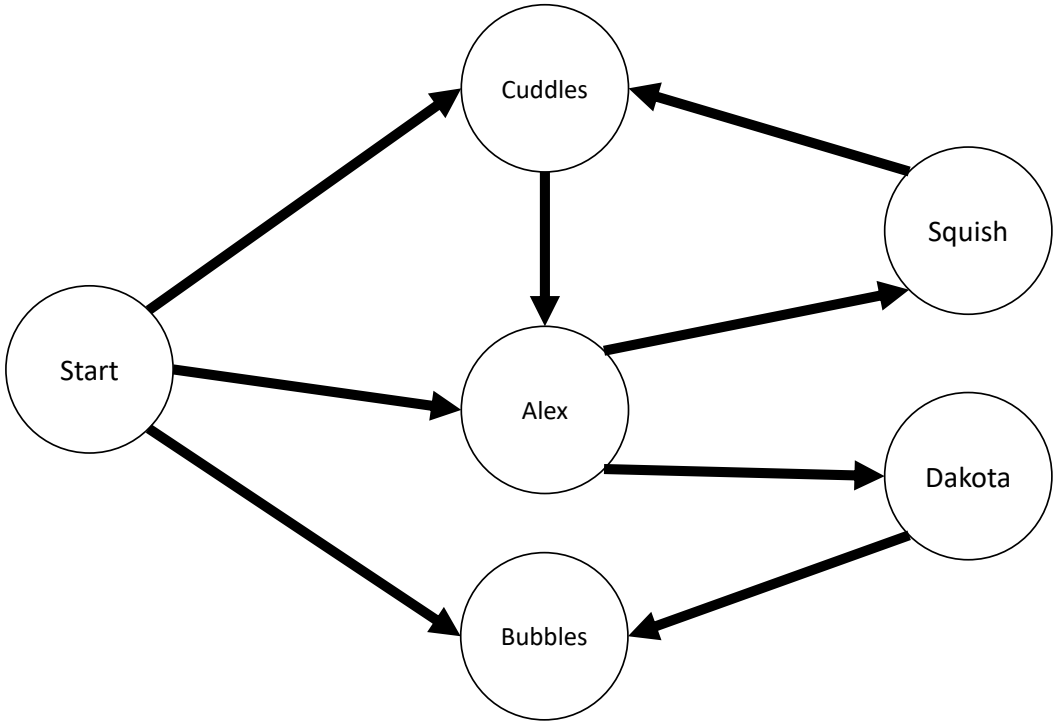
(b) [5 pts] **Third Time's A Charm**

Now we assume that Scorpblorg's mate preferences have changed. The new criteria she is looking for in a mate is that she has **visited the mate twice before** (i.e. when she visits any state for the third time, she has found a path to the goal).

(i) [3 pts] What should the most simple yet sufficient new state space representation include?

- [x] The current location of Scorpblorg
- [ ] The total number of edges travelled so far
- [ ] An array of booleans indicating whether each snail has been visited so far
- [x] An array of numbers indicating how many times each snail has been visited so far
- [ ] The number of distinct snails visited so far

The current location is needed to generate successors. The array of number indicating how many times each snail has been visited so far is needed for the goal test. A list of boolean is insufficient because we need to revisit more than once. Other information is redundant



(The graph is copied for your convenience)

(ii) [1 pt] DFS Tree Search finds a path to the goal for this graph ● True ○ False

DFS Tree Search does not get stuck in any loops on this graph and does return the solution path.

(iii) [1 pt] BFS Graph Search finds a path to the goal for this graph ● True ○ False

Revisiting a location is allowed with BFS Graph search because the "visited" set keep track of the augmented states, which means revisiting any location is right

(iv) [2 pts] If we remove the connection from Cuddles → Alex, can DFS Graph Search finds a path to the goal for the altered graph? ○ Yes ● No

Meeting three time requires the Alex, Cuddles, Squish cycle. Since it is the only cycle, removing it will prevent Scorpblorg from meeting any mate three times

We continue as in part (b) where the goal is still to find a mate who is visited for the third time.

(c) [4 pts] **Costs for visiting snails**

Assume we are using Uniform cost search and we can now add costs to the actions in the graph.

(i) [2 pts] Can one assign (non-negative) costs to the actions in the graph such that the goal state returned by UCS (Tree-search) changes? ● Yes ○ No

Yes, if the costs are all equal, UCS will return the same goal state as BFS (Tree-search): Alex. However, putting a very large cost on the path from Cuddles to Alex will change the goal state to Cuddles. Other Examples exist.

(ii) [2 pts] Can one assign (potentially negative) costs to the actions in the graph such that UCS (Tree-search) will never find a goal state? ○ Yes ● No

No, regardless of the costs on the graph, eventually a state will be re-visited, resulting in a goal state.

Q7. [16 pts] Searching with Heuristics

Consider the A* searching process on the connected undirected graph, with starting node S and the goal node G. Suppose the cost for each connection edge is **always positive**. We define $h^*(X)$ as the shortest (optimal) distance to G from a node X.

Answer Questions (a), (b) and (c). You may want to solve Questions (a) and (b) at the same time.

(a) [6 pts] Suppose h is an **admissible** heuristic, and we conduct A* **tree search** using heuristic h' and finally find a solution. Let C be the cost of the found path (directed by h', defined in part (a)) from S to G

(i) [4 pts] Choose **one best** answer for each condition below.

1. If $h'(X) = \frac{1}{2}h(X)$ for all Node X, then ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
2. If $h'(X) = \frac{h(X)+h^*(X)}{2}$ for all Node X, then ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
3. If $h'(X) = h(X) + h^*(X)$ for all Node X, then ○ $C = h^*(S)$ ○ $C > h^*(S)$ ● $C \geq h^*(S)$
4. If we define the set $K(X)$ for a node X as all its neighbor nodes Y satisfying $h^*(X) > h^*(Y)$, and the following always holds

$$h'(X) \leq \begin{cases} \min_{Y \in K(X)} h'(Y) - h(Y) + h(X) & \text{if } K(X) \neq \emptyset \\ h(X) & \text{if } K(X) = \emptyset \end{cases}$$

then, ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$

5. If K is the same as above, we have

$$h'(X) = \begin{cases} \min_{Y \in K(X)} h(Y) + cost(X, Y) & \text{if } K(X) \neq \emptyset \\ h(X) & \text{if } K(X) = \emptyset \end{cases}$$

where $cost(X, Y)$ is the cost of the edge connecting X and Y, then, ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$

6. If $h'(X) = \min_{Y \in K(X)+\{X\}} h(Y)$ (K is the same as above), ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$

(ii) [2 pts] In which of the conditions above, h' is still **admissible** and for sure to dominate h? Check all that apply. Remember we say h_1 dominates h_2 when $h_1(X) \geq h_2(X)$ holds for all X. ☐ 1 ■ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6

(b) [7 pts] Suppose h is a **consistent** heuristic, and we conduct A* **graph search** using heuristic h' and finally find a solution.

(i) [5 pts] Answer exactly the same questions for each conditions in Question (a)(i).

1. ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
2. ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
3. ○ $C = h^*(S)$ ○ $C > h^*(S)$ ● $C \geq h^*(S)$
4. ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
5. ● $C = h^*(S)$ ○ $C > h^*(S)$ ○ $C \geq h^*(S)$
6. ○ $C = h^*(S)$ ○ $C > h^*(S)$ ● $C \geq h^*(S)$

(ii) [2 pts] In which of the conditions above, h' is still **consistent** and for sure to dominate h? Check all that apply.

☐ 1 ■ 2 ☐ 3 ☐ 4 ■ 5 ☐ 6

Grading for Bubbles: 0.5 pts for a1 a2 a3 a6 b1 b2. 1 pts for a4 a5 b3 b4 b5 b6.

Explanations:

All the $C > h^*(S)$ can be ruled out by this counter example: there exists only one path from S to G.

Now for any $C = h^*(S)$ we shall provide a proof. For any $C \geq h^*(S)$ we shall provide a counter example.

a3b3 - Counter example: SAG fully connected. cost: SG=10, SA=1, AG=7. h*: S=8, A=7, G=0. h: S=8, A=7, G=0. h': S=16, A=14, G=0.

a4 - Proof: via induction. We can have an ordering of the nodes $\{X_j\}_{j=1}^n$ such that $h^*(X_i) \geq h^*(X_j)$ if $i < j$. Note any $X_k \in K(X_j)$ has $k > j$.

X_n is G, and has $h'(X_n) \leq h(X_n)$.

Now for j, suppose $h'(X_k) \leq h(X_k)$ for any $k > j$ holds, we can have $h'(X_j) \leq h'(X_k) - h(X_k) + h(X_j) \leq h(X_j)$ ($K(X_j) = \emptyset$ also get the result).

b4 - Proof: from a4 we already know that h' is admissible.

Now for each edge XY, suppose $h^*(X) \geq h^*(Y)$, we always have $h'(X) \leq h'(Y) - h(Y) + h(X)$, which means $h'(X) - h'(Y) \leq h(X) - h(Y) \leq cost(X, Y)$, which means we always underestimate the cost of each edge **from the potential optimal path direction**. Note h' is not necessarily to be consistent ($h'(Y) - h'(X)$ might be very large, e.g. you can arbitrarily modify h'(S) to be super small), but it always comes with optimality.

a5 - Proof: the empty K path: $h'(X) \leq h(X) \leq h^*(X)$. the non-empty K path: there always exists a $Y_0 \in K(X)$ such that Y_0 is on the optimal path from X to G. We know $cost(X, Y_0) = h^*(X) - h^*(Y_0)$, so we have $h'(X) \leq h(Y_0) + cost(X, Y_0) \leq h^*(Y_0) + cost(X, Y_0) = h^*(X)$.

b5 - Proof:

First we prove $h'(X) \geq h(X)$. For any edge XY, we have $h(X) - h(Y) \leq cost(X, Y)$. So we can have $h(Y) + cost(X, Y) \geq h(X)$ holds for any edge, and hence we get the dominace of h' over h. Note this holds only for consistent h.

We then have $h'(X) - h'(Y) \leq h(Y) + cost(X, Y) - h'(Y) \leq cost(X, Y)$. So we get the consistency of h'.

Extension Conclusion 1: If we change K(X) into {all neighbouring nodes of X} + {X}, h' did not change.

Extension Conclusion 2: h' dominates h, which is a better heuristics. This (looking one step ahead with h') is equivalent to looking two steps ahead in the A* search with h (while the vanilla A* search is just looking one step ahead with h).

a6 - Proof: $h'(X) \leq h(X) \leq h^*(X)$.

b6 - counter example: SAB fully connected, BG connected. cost: SA=8, AB=1, SB=10, BG=30. h*: A=31, B=30 G=0. h=h*. h': A=30, B=0, C=0.

(c) [3 pts] Suppose h is an **admissible** heuristic, and we conduct A* **tree search** using heuristic h' and finally find a solution. If $\epsilon > 0$, and X_0 is a node in the graph, and h' is a heuristic such that

$$h'(X) = \begin{cases} h(X) & \text{if } X = X_0 \\ h(X) + \epsilon & \text{otherwise} \end{cases}$$

- Alice claims h' can be inadmissible, and hence $C = h^*(S)$ does not always hold.
- Bob instead thinks the node expansion order directed by h' is the same as the heuristic h'', where

$$h''(X) = \begin{cases} h(X) - \epsilon & \text{if } X = X_0 \\ h(X) & \text{if otherwise} \end{cases}$$

  Since h'' is admissible and will lead to $C = h^*(S)$, and so does h'. Hence, $C = h^*(S)$ always holds.

The two conclusions (underlined) apparently contradict with each other, and **only exactly one of them are correct and the other is wrong**. Choose the **best** explanation from below - which student's conclusion is wrong, and why are they wrong?

◯ Alice's conclusion is wrong, because the heuristic h' is always admissible.

◯ Alice's conclusion is wrong, because an inadmissible heuristics does not necessarily always lead to the failure of the optimality when conducting A* tree search.

◯ Alice's conclusion is wrong, because of another reason that is not listed above.

◯ Bob's conclusion is wrong, because the node visiting expansion ordering of h'' during searching might not be the same as h'.

◯ Bob's conclusion is wrong, because the heuristic h'' might lead to an incomplete search, regardless of its optimally property.

● Bob's conclusion is wrong, because of another reason that is not listed above.

Choice 4 is incorrect, because the difference between h' and h'' is a constant. During searching, the choice of the expansion of the fringe will not be affected if all the nodes add the same constant to the heuristics.

Choice 5 is incorrect because there will never be an infinite loop if there are no cycle has negative COST sum (rather than HEURISTICS). If there is a cycle, such that its COST sum is positive, and all the nodes in the cycle have negative heuristics, when we do g+h, g is getting larger and larger, while h remains a not-that-large negative value. Soon, the search algorithm will be favoring other paths even if the h in there are not negative.

The true reason: h'' violate a property of admissible heuristic. Since h is admissible, we have $h(G) = 0$. If $X_0 = G$, we have a negative heuristic value at $h''(G)$, and it is no longer admissible. If $X_0 \neq G$, then it is indeed that the optimality holds - the only change is that more nodes will be likely to be expanded for h' and h'' compared to h.