CS 188 Fall 2010 — Introduction to Artificial Intelligence — Midterm Exam

INSTRUCTIONS

- You have 3 hours.
- The exam is closed book, closed notes except a one-page crib sheet.
- Please use non-programmable calculators only.
- 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.

Last Name	
First Name	
SID	
Login	
GSI	
Section Time	
Name of the person to your left	
Name of the person to your right	
All the work on this exam is my own. **(please sign)**	

For staff use only

Q. 1	Q. 2	Q. 3	Q. 4	Q. 5	Total
/19	/12	/17	/20	/22	/90

THIS PAGE INTENTIONALLY LEFT BLANK

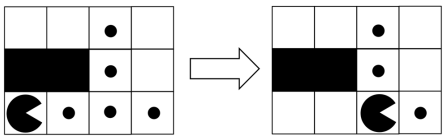
1. (19 points) Search: X-PacMen

We will discuss three variants of Pacman in which the goal is always to eat every pellet and there are no ghosts. However, in each question, Pacman has a different superpower. When not otherwise specified, the cost of every action is 1.

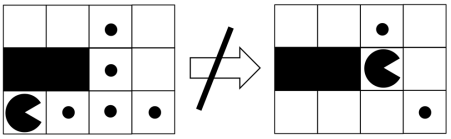
Notation: Let N and M be the width and height of a board, let F the initial number of food pellets in a start state, and let f be the number of uneaten food pellets remaining in a given state.

Quicksilver Pacman: parts (a), (b), and (c)
Pacman can run very fast in any direction, and he **must** pick up **all** the pellets he encounters along the way. He's so fast that he can move any number of squares in a single move. He cannot run through walls, he can only run in one of the four directions, and he cannot turn while running.

For example, this is a valid move:



But this is not:



For this problem, we use the following state-space:

State: $\begin{cases} \text{Pacman's location:} & (x, y), \text{ where } x \in [1, \ldots, N], y \in [1, \ldots, M] \\ \text{Flags indicating which dots remain:} & \{f_i\}, \text{ where } f_i \in \{true, false\}, i \in [1, \ldots, F] \end{cases}$

(a) (2 pt) Circle the tightest upper bound on the size of this state space:

$$N \times M, \quad F \times M \times N, \quad 2^F \times N \times M, \quad 2^{N \times M}, \quad 2^{F \times N \times M}$$

(b) (2 pt) Circle the tightest upper bound on the branching factor of this state space (the number of successor states of a single state):

$$4, \quad F, \quad N + M, \quad N \times M, \quad F \times (M + N), \quad F \times N \times M, \quad 2^F \times (M + N)$$

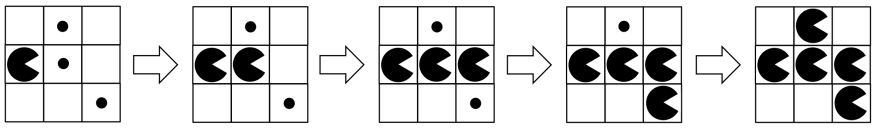
(c) (3 pt) For each of the following heuristics, indicate (yes/no) whether or not it is admissible.

Heuristic	Admissible?
f (the number of food pellets remaining)	No
$\min_{food}(\text{manhattanDistance}(pacman, food))$ (smallest Manhattan dist. from Pacman to a remaining food)	No
$f / \max(M, N)$	Yes
$f/(M + N)$	Yes

Multiple (Pac)man: parts (d) and (e)

Instead of moving, this Pacman will instead make a duplicate of itself in a free (non-wall and non-Pacman) cell that is adjacent to any current Pacman duplicate. Duplicates cannot move, and cannot be deleted; only new duplicates can be made. Only one single duplicate can be created at each turn.

Here is a valid sequence of moves:



(d) (2 pt) Formally specify an efficient state space for this search problem:

An **unordered set**, not an ordered list, of the positions occupied by a pacman. Equivalently, a bit vector for each square, indicating if it has a pacman or not. A list will be very inefficient, as you can represent the same board as combinatorially many states. Representing food is not necessary, as whether or not pacman is in a square indicates whether or not the food in that square was eaten.

(e) (2 pt) For each of the following search algorithms, indicate (yes/no) whether or not it is optimal (guaranteed to produce an optimal solution) **for any instance of this search problem**, and whether or not is is complete (guaranteed to find a solution if one exists) **for any instance of this search problem**.

Algorithm	Optimal?	Complete?
Tree Search - BFS	Yes	Yes
Tree Search - DFS	No	Yes
Graph Search - BFS	Yes	Yes
Graph Search - DFS	No	Yes

Nightcrawler Pacman: parts (f), (g), and (h)

Pacman can now teleport to any non-wall square on the map, in addition to being able to move to any adjacent non-wall square. Moving to an adjacent square is easy, and costs Pacman 1, but teleporting is difficult, and costs Pacman 3.

We use the same state space as for Quicksilver Pacman:

State: $\begin{cases} \text{Pacman's location:} & (x, y), \text{ where } x \in [1, \ldots, N], y \in [1, \ldots, M] \\ \text{Flags indicating which dots remain:} & \{f_i\}, \text{ where } f_i \in \{true, false\}, i \in [1, \ldots, F] \end{cases}$

(f) (2 pt) For each of the following search algorithms, indicate (yes/no) whether or not it is optimal (guaranteed to produce an optimal solution) **for any instance of this search problem**, and whether or not is is complete (guaranteed to find a solution if one exists) **for any instance of this search problem**.

Algorithm	Optimal?	Complete?
Graph Search - BFS	No	Yes
Graph Search - DFS	No	Yes
Graph Search - UCS	Yes	Yes

(g) (3 pt) Consider the following heuristics. For each, indicate (yes/no) whether or not it is admissible.

Heuristic	Admissible?
f (the number of food pellets remaining)	Yes
$\max_{food}(\text{manhattanDistance}(pacman, food))$ (the largest Manhattan distance between Pacman and any food)	No
$\max_{food}(\text{manhattanDistance}(pacman, food))/3$ (1/3 the largest Manhattan distance between Pacman and any food)	No
$\min(\text{optimal non-teleporting path}, 3 \times f)$ (the minimum of: the optimal cost of solving the board **without** teleports, and the optimal cost of solving the board using **only** teleports)	No

(h) (3 pt) Uh oh, Nightcrawler Pacman's superpowers are becoming unreliable! Now, when Pacman tries to teleport, there's a 10% chance that he will fail, in which case he will not move at all, but will still incur a cost of 3. Modeling this problem as a "search" problem probably isn't a good idea anymore.

What is the best way to model this new Pacman? **Briefly** (no more than three sentences) describe how you would model this problem. Make sure you precisely specify what the key change is (e.g. defining a critical new quantity).

Model it as an MDP, where T encodes the probability of staying put when Pacman tries to teleport. R is -1 if Pacman moves, and -3 if he tries to teleport. Solve by doing value iteration, and then picking the actions that maximize expected utility (have the bigger V^*)

This can be modeled as an MDP, using T to encode the probability that teleporting may return you to the same state: T(s, teleport, s) = 0.1, etc. R is -1 if Pacman moves, and -3 if he tries to teleport.
OR:
This can be modeled as an expectimax problem, where the chance nodes encode the probability that teleporting fails, etc.
OR:
Modify the problem such that teleporting has a cost of 10/3, and search that space.

2. (12 points) Warcraft Building Order and Layout

An important element of the game of Warcraft is the construction of "buildings" which perform various functions. There are physical limitations on where these buildings can be built relative to each other. *Note: The setup in this question does not accurately reflect the game Warcraft, nor should knowledge of Warcraft be of any help.*

Imagine that there are 5 buildings: a (C)astle, a (G)oldmine, a (F)arm, a (B)arracks, and an (A)rmory, which must be laid out on the following grid:

1	2	3
4	5	6
7	8	9

Your goal is to assign each building to a grid position. Suppose we formulate this problem as a CSP where there is one variable per building, whose domain is a location 1 through 9. There are the following constraints on layout:

- Buildings cannot be on the same square.
- No building can share an edge with the (G)oldmine.
- The (G)oldmine and the (C)astle must share a corner, but not an edge.
- The (C)astle must share an edge with the (B)arracks.
- The (C)astle must share an edge with the (A)rmory.
- The (F)arm must not share an edge with the (A)rmory.
- The (F)arm must be built in the top row.

(a) (3 pt) Starting from a blank grid, cross out all assignments that are removed by enforcing arc consistency and unary constraints.

1	2	3
4	5	6
7	8	9

Variable	Domain								
A	1	~~2~~	3	4	5	6	7	8	9
B	1	2	3	4	5	6	7	8	9
C	1	2	3	4	5	6	7	8	9
F	1	2	3	~~4~~	~~5~~	~~6~~	~~7~~	~~8~~	~~9~~
G	1	~~2~~	3	4	5	6	7	8	9

(b) (3 pt) Suppose we fix the (C)astle in the center tile and run **forward checking**. Cross out all variables eliminated by forward checking. (Don't use the arc consistent results from part (a). Start from a clean slate of full domains.)

1	2	3
4	C	6
7	8	9

Variable	Domain								
A	~~1~~	2	~~3~~	4	~~5~~	6	~~7~~	8	~~9~~
B	~~1~~	2	~~3~~	4	~~5~~	6	~~7~~	8	~~9~~
C					5				
F	1	2	3	~~4~~	~~5~~	~~6~~	~~7~~	~~8~~	~~9~~
G	1	~~2~~	3	4	~~5~~	~~6~~	7	~~8~~	9

Full credit was also given if the unary constraints on F were not enforced.

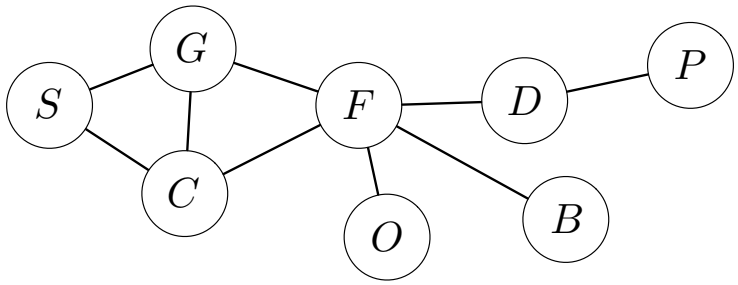
(c) (2 pt) In Warcraft, certain buildings have to be built before certain other buildings. To capture these constraints, we add **additional** variables T_{C}, T_{F}, T_{A}, T_{G} and T_{B} indicating the time step at which each building is built. The domain of these variables is an integer in $1 \ldots 5$ indicating at which of 5 possible time steps the building is built. There are also the following constraints on the time variables:

- Buildings cannot be built during the same time step.
- The (C)astle must be built before all other buildings. (i.e. $T_{\mathrm{C}} < T_{\mathrm{X}}$, for all other buildings X)
- The (F)arm must be built before the (B)arracks. ($T_{\mathrm{F}} < T_{\mathrm{B}}$)
- The (A)rmory must be built after the (B)arracks. ($T_{\mathrm{A}} > T_{\mathrm{B}}$)

Suppose you want to solve the CSP over the variables $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{F}, \mathrm{G}, T_{\mathrm{A}}, T_{\mathrm{B}}, T_{\mathrm{C}}, T_{\mathrm{F}}, T_{\mathrm{G}}\}$ in order to find a satisfactory order and layout for building construction (the constraints on layout variables are still the same as in parts (a) and (b)). You run an off-the-shelf solver and you notice that it is running too slowly. How can you speed up the solution substantially over the naive solver, using a property of this **particular** CSP?

The constraints on building order and building layout are independent and can be solved as two separate (smaller) CSPs. Partial credit was given for noticing that some particular constraints were easy to enforce by inspection, e.g. $T_{\mathrm{C}} = 1$.

(d) (4 pt) In real Warcraft, there are far more buildings to deal with. Suppose the constraint graph for an enhanced version of Warcraft looks as follows.



Typically, CSPs are exponential in the number of variables. However, based on your intuition that this graph is similar to a tree, you would like to use an efficient, special purpose CSP solver that can only solve tree-structured CSPs. For this particular graph, there are **two** methods of exploiting your tree-structured solver. Briefly but precisely describe two methods of efficiently solving this CSP that make one or more calls to the tree-structured solver.

Method 1:

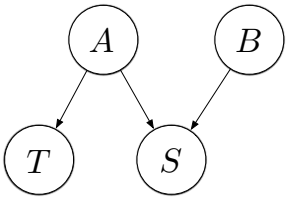
Cutset conditioning. We pick a set of variables V which, when removed from the graph, leave a tree-structured CSP behind. We assign a value to each variable in V and prune the domains of any other variables. We then solve the tree-structured CSP and merge with our assignment to V. If all constraints are satisfied, we have a solution, otherwise we must pick another assignment to V and try the whole process again. In order to receive full credit, you had to specify which specific variables you would put in V. V=$\{G\}$ and V=$\{C\}$ were common choices.

Method 2:

Tree decomposition. We merge several variables together to form "megavariables" whose domain is the cross product of its component variables, and constraints are the union of its component constraints. If after replacing each set of variables with its "megavariable", we are left with a tree structured CSP, then we can solve the tree-strucutred CSP with an off-the-shelf solver. To receive full credit, you had to mention which megavariables you would create. $G - C$ was a common choice, as was $S - G - C$. Full credit was given to any set of megavariables which left behind a tree-structured CSP.

3. (17 points) Bayes' Nets

Suppose that a patient can have a symptom (S) that can be caused by two different diseases (A and B). Disease A is much rarer, but there is a test T that tests for the presence of A. The Bayes' Net and corresponding conditional probability tables for this situation are shown below. For each part, you may leave your answer as a fraction.



$\mathbf{P}(A)$	
$+a$	0.1
$-a$	0.9

$\mathbf{P}(B)$	
$+b$	0.5
$-b$	0.5

$\mathbf{P}(T\|A)$		
$+a$	$+t$	1
$+a$	$-t$	0
$-a$	$+t$	0.2
$-a$	$-t$	0.8

$\mathbf{P}(S\|A,B)$			
$+a$	$+b$	$+s$	1
$+a$	$+b$	$-s$	0
$+a$	$-b$	$+s$	0.80
$+a$	$-b$	$-s$	0.20
$-a$	$+b$	$+s$	1
$-a$	$+b$	$-s$	0
$-a$	$-b$	$+s$	0
$-a$	$-b$	$-s$	1

(a) (1 pt) Compute the following entry from the joint distribution:

$\mathbf{P}(-a, -t, +b, +s) =$

$\mathbf{P}(-t|-a)\mathbf{P}(-a)\mathbf{P}(+s|+b,-a)\mathbf{P}(+b) = (0.8)(0.9)(1)(0.5) = 0.36$

(b) (2 pt) What is the probability that a patient has disease A given that they have disease B?

$\mathbf{P}(+a|+b) = \mathbf{P}(+a) = 0.1$

(c) (2 pt) What is the probability that a patient has disease A given that they have symptom S, disease B, and test T returns positive?

$$\mathbf{P}(+a|+t,+s,+b) = \frac{\mathbf{P}(+a,+t,+s,+b)}{\mathbf{P}(+t,+s,+b)} = \frac{\mathbf{P}(+a)\mathbf{P}(+t|+a)\mathbf{P}(+s|+a,+b)\mathbf{P}(+b)}{\sum_{a\in\{+a,-a\}}\mathbf{P}(a,+t,+s,+b)}$$
$$= \frac{(0.1)(1)(1)(0.5)}{(0.1)(1)(1)(0.5)+(0.9)(0.2)(1)(0.5)} = \frac{.05}{.05+.09} = \frac{5}{14} \approx 0.357$$

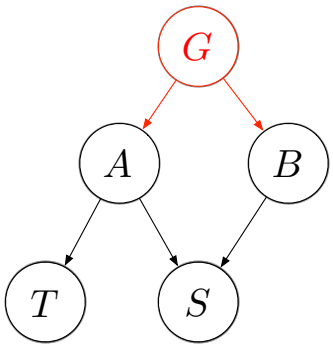
(d) (3 pt) What is the probability that a patient has disease A given that they have symptom S and test T returns positive?

$$\mathbf{P}(+a|+t,+s) = \frac{\mathbf{P}(+a,+t,+s)}{\mathbf{P}(+t,+s)} \frac{\sum_{b\in\{+b,-b\}}\mathbf{P}(+a)\mathbf{P}(+t|+a)\mathbf{P}(+s|+a,b)\mathbf{P}(b)}{\sum_{a\in\{+a,-a\}}\sum_{b\in\{+b,-b\}}\mathbf{P}(a)\mathbf{P}(+t|a)\mathbf{P}(+s|a,b)\mathbf{P}(b)} = 0.5$$

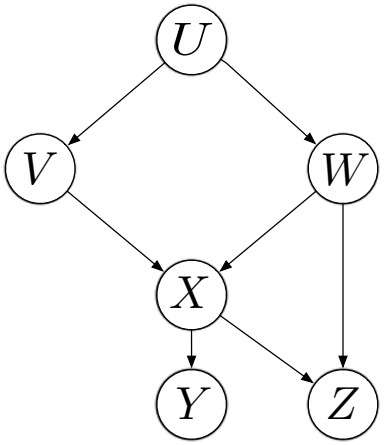
(e) (3 pt) Suppose that both diseases A and B become more likely as a person ages. Add any necessary variables and/or arcs to the Bayes' net to represent this change. For any variables you add, *briefly* (one sentence or less) state what they represent. Also, state one independence or conditional independence assertion that is **removed** due to your changes.

New variable(s) (and brief definition):
G: Age in years, or as a boolean for old or not, etc.

Removed conditional independence assertion:
There are a few. The simplest is $A \perp\!\!\!\perp B$ is no longer guaranteed. Another is $B \perp\!\!\!\perp T$.



(f) (6 pt) Based only on the structure of the (new) Bayes' Net given below, circle whether the following conditional independence assertions are guaranteed to be true, guaranteed to be false, or cannot be determined by the structure alone.



i. $V \perp\!\!\!\perp W$ — Guaranteed true — Cannot be determined — Guaranteed false

ii. $V \perp\!\!\!\perp W \mid U$ — Guaranteed true — Cannot be determined — Guaranteed false

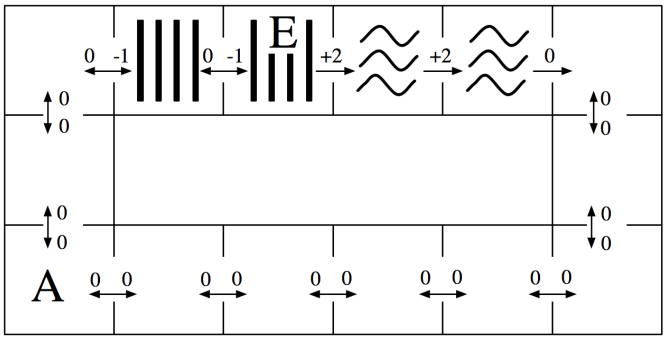
iii. $V \perp\!\!\!\perp W \mid U, Y$ — Guaranteed true — Cannot be determined — Guaranteed false

iv. $V \perp\!\!\!\perp Z \mid U, X$ — Guaranteed true — Cannot be determined — Guaranteed false

v. $X \perp\!\!\!\perp Z \mid W$ — Guaranteed true — Cannot be determined — Guaranteed false

4. (20 points) MDPs: Grid-World Water Park

Consider the MDP drawn below. The state space consists of all squares in a grid-world water park. There is a single waterslide that is composed of two ladder squares and two slide squares (marked with vertical bars and squiggly lines respectively). An agent in this water park can move from any square to any neighboring square, unless the current square is a slide in which case it must move forward one square along the slide. The actions are denoted by arrows between squares on the map and all deterministically move the agent in the given direction. The agent cannot stand still: it must move on each time step. Rewards are also shown below: the agent feels great pleasure as it slides down the water slide (+2), a certain amount of discomfort as it climbs the rungs of the ladder (-1), and receives rewards of 0 otherwise. The time horizon is infinite; this MDP goes on forever.



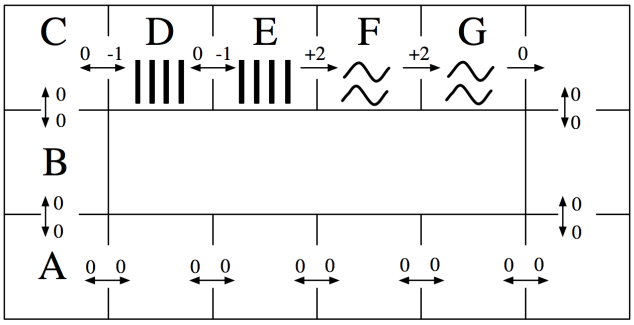
(a) (2 pt) How many (deterministic) policies π are possible for this MDP?

2^{11}

(b) (9 pt) Fill in the blank cells of this table with values that are correct for the corresponding function, discount, and state. *Hint: You should not need to do substantial calculation here.*

	γ	$s = A$	$s = E$
$V_3^*(s)$	1.0	0	4
$V_{10}^*(s)$	1.0	2	4
$V_{10}^*(s)$	0.1	0	2.2
$Q_1^*(s, \text{left})$	1.0	——	0
$Q_{10}^*(s, \text{left})$	1.0	——	3
$V^*(s)$	1.0	∞	∞
$V^*(s)$	0.1	0	2.2

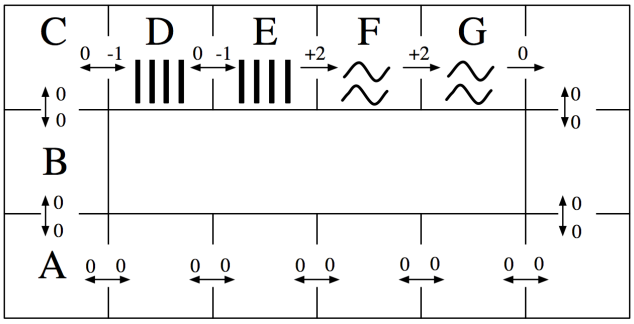
Use this labeling of the state space to complete the remaining subproblems:



(c) (5 pt) Fill in the blank cells of this table with the Q-values that result from applying the Q-update for the transition specified on each row. You may leave Q-values that are unaffected by the current update blank. Use discount $\gamma = 1.0$ and learning rate $\alpha = 0.5$. Assume all Q-values are initialized to 0. (Note: the specified transitions would not arise from a single episode.)

	$Q(D, \text{left})$	$Q(D, \text{right})$	$Q(E, \text{left})$	$Q(E, \text{right})$
Initial:	0	0	0	0
Transition 1: $(s = D, a = \text{right}, r = -1, s' = E)$		-0.5		
Transition 2: $(s = E, a = \text{right}, r = +2, s' = F)$				1.0
Transition 3: $(s = E, a = \text{left}, r = 0, s' = D)$				
Transition 4: $(s = D, a = \text{right}, r = -1, s' = E)$		-0.25		

The agent is still at the water park MDP, but now we're going to use function approximation to represent Q-values. Recall that a policy π is *greedy* with respect to a set of Q-values as long as $\forall a, s\ Q(s, \pi(s)) \geq Q(s, a)$ (so ties may be broken in any way).



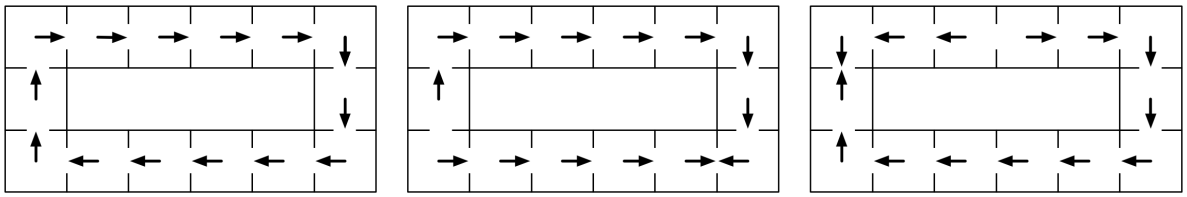
For the next subproblem, consider the following feature functions:

$$f(s,a) = \begin{cases} 1 & \text{if } a = \text{right}, \\ 0 & \text{otherwise.} \end{cases}$$

$$f'(s,a) = \begin{cases} 1 & \text{if } (a = \text{right}) \wedge \text{isSlide}(s), \\ 0 & \text{otherwise.} \end{cases}$$

(Note: isSlide(s) is true iff the state s is a slide square, i.e. either F or G.)

Also consider the following policies:



(d) (4 pt) Which are greedy policies with respect to the Q-value approximation function obtained by running the single Q-update for the transition $(s = F, a = \text{right}, r = +2, s' = G)$ while using the specified feature function? You may assume that all feature weights are zero before the update. Use discount $\gamma = 1.0$ and learning rate $\alpha = 1.0$. Circle all that apply.

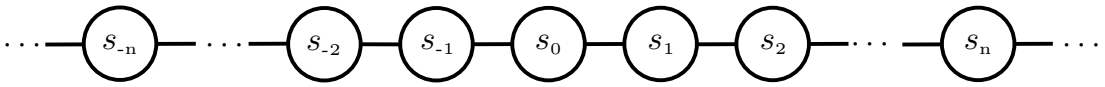
f	π_1	π_2	π_3
f'	π_1	π_2	π_3

5. (22 points) Multiple Choice

(a) (3 pt) Consider using A* to solve a generic search problem that has a finite number of states, positive edge weights, and multiple goals. When running A*, we might be using the tree search or graph search variant, and our heuristic may be inadmissible, admissible but inconsistent, or admissible and consistent. For each A* variant in the following table, indicate whether it is complete and/or optimal by circling the appropriate choices. (Note: you may circle one, both, or neither.)

	Type of Search	
Heuristic	Tree	Graph
Inadmissible	Complete / Optimal	Complete / Optimal
Admissible, Inconsistent	Complete / Optimal	Complete / Optimal
Admissible, Consistent	Complete / Optimal	Complete / Optimal

(b) (4 pt) Consider the following 1-d search problem: there is a state s_i for every integer $i = \cdots, -2, -1, 0, 1, 2, \cdots$. The search starts at s_0, and each node s_i has two successors, s_{i-1} and s_{i+1}. The only goal nodes in the search are s_n and s_{-n}, for some fixed $n > 0$.



i. About how many nodes are expanded in a breadth-first **tree** search before a goal node is popped off the queue?

n $\quad 2n$ $\quad 2^n$ $\quad 2^{2n}$ $\quad 2^{2^n}$ $\quad \infty$

ii. About how many nodes are expanded in a breadth-first **graph** search before a goal node is popped off the queue?

n $\quad 2n$ $\quad 2^n$ $\quad 2^{2n}$ $\quad 2^{2^n}$ $\quad \infty$

(c) (2 pt) When solving a constraint satisfaction problem, we can use forward checking and/or enforce arc consistency to filter the domains of unassigned variables. Which of the following statements are true, if any? Circle all that apply.

i. If forward checking eliminates an inconsistent value, enforcing arc consistency would eliminate it as well.

ii. If enforcing arc consistency eliminates an inconsistent value, forward checking would eliminate it as well.

(d) **(2 pt)** In a general CSP with n variables, domains of size d, and a arcs, suppose we wish to enforce arc consistency using AC3 (the arc consistency algorithm from class). Below, circle the tightest upper bound (in terms of n, d, and a) on the number of times we might need to check a **single** arc before AC3 terminates.

n $\quad$ a $\quad$ d $\quad$ nd $\quad$ nda $\quad$ n^d $\quad$ an^d $\quad$ ∞

(e) **(2 pt)** When applying alpha-beta pruning to minimax game trees, which of the following statements are true, if any? Assume the max player acts first. Circle all that apply.

i. Pruning nodes does not change the value of the root to the max player.

ii. When using alpha-beta pruning, the children of the root node will return the same value as they return in a minimax search.

iii. Alpha-beta pruning can prune different numbers of nodes if the children of the root node are reordered.

iv. When using alpha-beta pruning, the children of the root node are never pruned.

(f) **(4 pt)** Consider the following lotteries:

- L_1 always pays 1.
- L_2 pays 0 with probability 0.5 and 2 with probability 0.5.
- L_3 always pays 2.

The CS188 GSIs have expressed their preferences over these lotteries as follows:

- Adam is indifferent between lottery L_2 and lottery L_1.
- Arjun prefers lottery L_1 to lottery L_2.
- Jie is indifferent between lottery L_3 and lottery L_2.
- Taylor prefers lottery L_2 to lottery L_1.

Draw lines to match each GSI with a utility function that is consistent with their stated preferences. Each GSI has a different utility function.

1. Adam b.
2. Arjun c.
3. Jie d.
4. Taylor a.

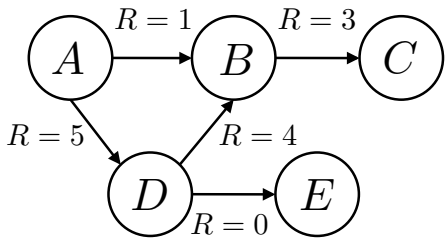
a. $U(x) = x^2$

b. $U(x) = x$

c. $U(x) = \sqrt{x}$

d. None of the above

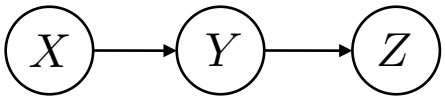
(g) **(2 pt)** The following figure shows a simple MDP with five states A, B, C, D, E and a single action *go*. An arrow from a state x to a state y indicates that it is possible to transition from state x to next state y when *go* is taken. If there are multiple arrows leaving a state x, transitioning to each of the next states is equally likely. The rewards for making each state transition are given on each edge in the figure. Assume a discount factor $\gamma = 1.0$. (Note: you should not need to do value iteration to solve this problem.)



How many iterations of value iteration will it take to converge to the true optimal values?

1 3 5 2^5 ∞

(h) **(3 pt)** Which of the following statements are guaranteed to be true of the distribution encoded by the Bayes net given below, if any? Circle all that apply.



i. $P(X, Y, Z) = P(X)P(Y|X)P(Z|X, Y)$

ii. $P(X, Y, Z) = P(X)P(Y|X)P(Z|Y)$

iii. $P(X, Y, Z) = P(X)P(Y|Z)P(Z|X)$

iv. $P(X, Y, Z) = P(Z)P(Y|Z)P(X|Y, Z)$

v. $P(X, Y, Z) = P(Z)P(Y|Z)P(X|Y)$