CS 188 Spring 2012 Introduction to Artificial Intelligence Midterm Exam

INSTRUCTIONS

- You have 1.5 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.
- Questions are not sequenced in order of difficulty. Make sure to look ahead if stuck on a particular question.

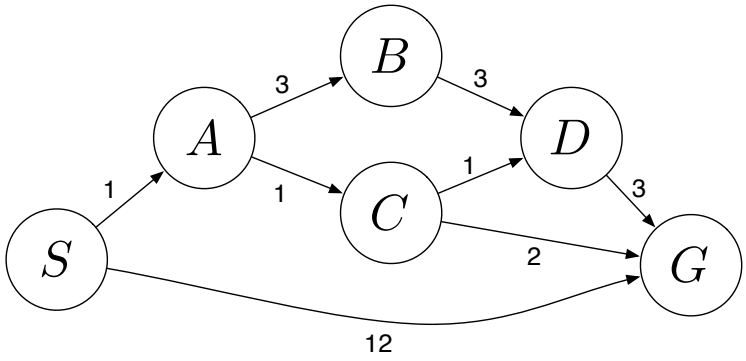
Last Name	
First Name	
SID	
Login	
All the work on this exam is my own. **(please sign)**	

For staff use only

Q. 1	Q. 2	Q. 3	Q. 4	Q. 5	Q. 6	Total
/12	/12	/12	/ 21	/ 24	/19	/100

THIS PAGE INTENTIONALLY LEFT BLANK

1. (12 points) Search



Answer the following questions about the search problem shown above. Break any ties alphabetically. For the questions that ask for a path, please give your answers in the form '$S - A - D - G$.'

(a) (2 pt) What path would breadth-first graph search return for this search problem?

$S - G$

(b) (2 pt) What path would uniform cost graph search return for this search problem?

$S - A - C - G$

(c) (2 pt) What path would depth-first graph search return for this search problem?

$S - A - B - D - G$

(d) (2 pt) What path would A* graph search, using a consistent heuristic, return for this search problem?

$S - A - C - G$

(e) (4 pt) Consider the heuristics for this problem shown in the table below.

State	h_1	h_2
S	5	4
A	3	2
B	6	6
C	2	1
D	3	3
G	0	0

i. (1 pt) Is h_1 admissible? **Yes** **No**

ii. (1 pt) Is h_1 consistent? **Yes** **No**

iii. (1 pt) Is h_2 admissible? **Yes** **No**

iv. (1 pt) Is h_2 consistent? **Yes** **No**

An admissible heuristic must underestimate or be equal to the true cost.
A consistent heuristic must satisfy $h(N) - h(L) \leq \text{path}(N \to L)$ for all paths and nodes N and L.

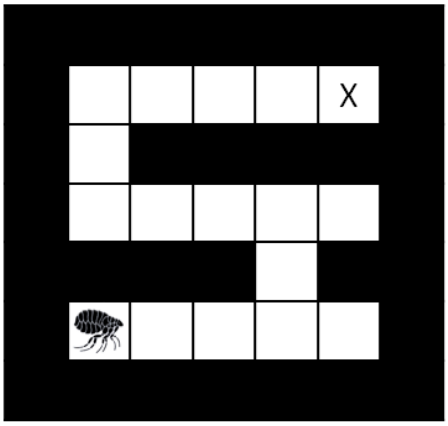
$h1$ overestimates the cost $S \to G$ as 5 when it is 4, so it is inadmissible.
$h1$ is not consistent because $h(S) - h(A) \leq \text{path}(S \to A)$ is violated as $5 - 3 \leq 1$.
$h2$ does not overestimate costs and is admissible.
$h2$ is not consistent because $h(S) - h(A) \leq \text{path}(S \to A)$ is violated as $4 - 2 \leq 1$.

2. (12 points) Hive Minds: Redux

Let's revisit our bug friends from assignment 2. To recap, you control one or more insects in a rectangular maze-like environment with dimensions $M \times N$, as shown in the figures below. At each time step, an insect can move North, East, South, or West (but not diagonally) into an adjacent square if that square is currently free, or the insect may stay in its current location. Squares may be blocked by walls (as denoted by the black squares), but the map is known.

For the following questions, you should answer for a general instance of the problem, not simply for the example maps shown.

(a) (6 pt) The Flea



You now control a single flea as shown in the maze above, which must reach a designated target location X. However, in addition to moving along the maze as usual, your flea can jump on top of the walls. When on a wall, the flea can walk along the top of the wall as it would when in the maze. It can also jump off of the wall, back into the maze. Jumping onto the wall has a cost of 2, while all other actions (including jumping back into the maze) have a cost of 1. Note that the flea can only jump onto walls that are in adjacent squares (either north, south, west, or east of the flea).

i. (2 pt) Give a *minimal* state representation for the above search problem.

The state is the location of the flea as an (x, y) coordinate. The map is known, including walls and the goal, and the actions of the flea depend only on its location.

ii. (2 pt) Give the size of the state space for this search problem.

The state space is $M \times N$. The flea can occupy any free location in a given maze, and any square might be free or a wall in a maze, so any of the $M \times N$ locations are possible.

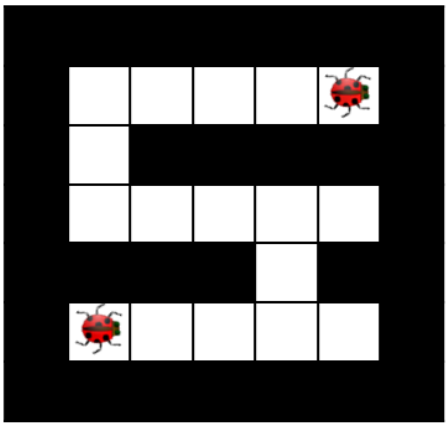
iii. (2 pt) Is the following heuristic admissible? **Yes** **No**

$$h_{\text{flea}} = \text{the Manhattan distance from the flea to the goal.}$$

It is yielded by the relaxed problem where the flea passes through walls. It never overestimates because 1. a wall can never decrease the length of a path to the goal and 2. the cost of the flea jumping up a wall (2) is higher than the cost of moving.

If it is *not* admissible, provide a nontrivial admissible heuristic in the space below.

(b) (6 pt) Long Lost Bug Friends



You now control a pair of long lost bug friends. You know the maze, but you do not have any information about which square each bug starts in. You want to help the bugs reunite. You must pose a search problem whose solution is an all-purpose sequence of actions such that, after executing those actions, both bugs will be on the same square, regardless of their initial positions. Any square will do, as the bugs have no goal in mind other than to see each other once again. Both bugs execute the actions mindlessly and do not know whether their moves succeed; if they use an action which would move them in a blocked direction, they will stay where they are. Unlike the flea in the previous question, bugs *cannot* jump onto walls. Both bugs can move in each time step. Every time step that passes has a cost of one.

i. (2 pt) Give a *minimal* state representation for the above search problem.

The state is a list of boolean variables, one for each position in the maze, which marks whether the position could contain a bug. There is no need to separately keep track of the bugs since their starting positions are not known; to ensure they meet only a single square must be possible for both.

ii. (2 pt) Give the size of the state space for this search problem.

The size is 2^{MN} since every of the $M \times N$ possible maze positions must be considered and every position has a boolean variable. A full state is the product of the individual position states, which are binary valued for the base of 2.

iii. (2 pt) Give a nontrivial admissible heuristic for this search problem.

h_{friends} = the maximum Manhattan distance of all possible pairs of points the bugs can be in.
This is never an overestimate because the number of steps to join the insects with certainty is at least the shortest path (with no obstacles) between their farthest possible locations from one another. Remember that the starting locations are unknown so the bugs cannot simply be controlled to move toward each other.

3. (12 points) A* Graph Search

```
function A* GRAPH SEARCH(problem)
    fringe ← an empty priority queue
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    closed ← an empty set
    ADD INITIAL-STATE[problem] to closed
    loop
        if fringe is empty then
            return failure
        end if
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then
            return node
        end if
        for successor in GETSUCCESSORS(problem, STATE[node]) do
            if successor not in closed then
                ADD successor to closed
                fringe ← INSERT(MAKE-SUCCESSOR-NODE(successor, node), fringe)
            end if
        end for
    end loop
end function
```

The above implementation of A graph search may be incorrect!* In the list below circle all of the problems that the bugs may cause when executing graph search and justify your answer. Note that the fringe is a priority queue. Nodes are inserted into the fringe using the standard key for A*, namely $f = g + h$. h is a consistent heuristic.

(a) The GetSuccessors function could be called multiple times on the same state.

(b) The algorithm is no longer complete.

(c) The algorithm could return a suboptimal solution.

(d) The implementation is incorrect, but none of the above problems will be caused.

(e) The implementation is correct.

To receive any credit you must briefly justify your answer in space below.

The bug is the insertion of *successor* into *closed* at time of insertion of a node into the fringe, rather than at the time that node gets popped from the fringe. As a consequence of this bug, the first path encountered to a state will put that state in the closed list. This can cause suboptimality as we only have the guarantee that a state has been reached optimally once a node reaching it gets popped off the fringe.

(a) is False as when a node that reaches a state s is placed in the fringe, that state s is also put on the closed list. This means never in the future can a node be placed on the fringe that ends in that same state s, and hence the same state s can be the argument to GetSuccessors at most once.

(b) is False. A* tree search is complete. The difference is that the above algorithm will cut off parts of the tree search whenever it has placed a node on the fringe in the past that ends in the same state. So compared to tree search we only lose copies of subtrees that we are covering. Hence the above algorithm is complete.

(c) is True. See explanation at beginning of solution.

(d) is False.

(e) is False.

4. (21 points) Time Management

Two of our GSIs, Arjun and Woody, are making their schedules for a busy morning. There are five tasks to be carried out:

(F) Pick up food for the group's research seminar, which, sadly, takes one precious hour.
(H) Prepare homework questions, which takes 2 consecutive hours.
(P) Prepare the PR2 robot for a group of preschoolers' visit, which takes one hour.
(S) Lead the research seminar, which takes one hour.
(T) Teach the preschoolers about the PR2 robot, which takes 2 consecutive hours.

The schedule consists of one-hour slots: 8am-9am, 9am-10am, 10am-11am, 11am-12pm. The requirements for the schedule are as follows:

(a) In any given time slot each GSI can do at most one task (F, H, P, S, T).

(b) The PR2 preparation (P) should happen before teaching the preschoolers (T).

(c) The food should be picked up (F) before the seminar (S).

(d) The seminar (S) should be finished by 10am.

(e) Arjun is going to deal with food pick up (F) since he has a car.

(f) The GSI not leading the seminar (S) should still attend, and hence cannot perform another task (F, T, P, H) during the seminar.

(g) The seminar (S) leader does not teach the preschoolers (T).

(h) The GSI who teaches the preschoolers (T) must also prepare the PR2 robot (P).

(i) Preparing homework questions (H) takes 2 consecutive hours, and hence should start at or before 10am.

(j) Teaching the preschoolers (T) takes 2 consecutive hours, and hence should start at or before 10am.

To formalize this problem as a CSP, use the variables F, H, P, S and T. The values they take on indicate the GSI responsible for it, and the starting time slot during which the task is carried out (for a task that spans 2 hours, the variable represents the starting time, but keep in mind that the GSI will be occupied for the next hour also - make sure you enforce constraint (a)!). Hence there are eight possible values for each variable, which we will denote by A8, A9, A10, A11, W8, W9, W10, W11, where the letter corresponds to the GSI and the number corresponds to the time slot. For example, assigning the value of A8 to a variables means that this task is carried about by Arjun from 8am to 9am.

(a) (2 pt) What is the size of the state space for this CSP?

8^5 since every task variable has 8 values, 4 time slots $\times$ 2 GSIs to carry them out, and there are 5 such tasks.

(b) (2 pt) Which of the statements above include unary constraints?

A unary constraint constrains the domain of a single variable. (d), (e), (i), (j) are unary constraints. (i) and (j) express both unary constraints (on the time of the tasks) and binary constraints (the length of tasks excludes other assignments during their time).

(c) (4 pt) In the table below, enforce all unary constraints by crossing out values in the table on the left below. If you made a mistake, cross out the whole table and use the right one.

F	A8	A9	A10	A11	~~W8~~	~~W9~~	~~W10~~	~~W11~~
H	A8	A9	A10	~~A11~~	W8	W9	W10	~~W11~~
P	A8	A9	A10	A11	W8	W9	W10	W11
S	A8	A9	~~A10~~	~~A11~~	W8	W9	~~W10~~	~~W11~~
T	A8	A9	A10	~~A11~~	W8	W9	W10	~~W11~~

(d) (3 pt) Start from the table above, select the variable S and assign the value A9 to it. Perform forward checking by crossing out values in the table below. Again the table on the right is for you to use in case you believe you made a mistake.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~
P	A8	~~A9~~	A10	A11	W8	~~W9~~	W10	W11
S	~~A8~~	A9	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~

Forward checking prunes the variables' domains of values inconsistent with S = A9, including: other choices for S, conflicting time slots for A9 (a), the choices of F that do not precede 9 (c), W from working during 9 (f), and A teaching the preschoolers (g).

(e) **(3 pt)** Based on the result of (d), what variable will we choose to assign next based on the MRV heuristic (breaking ties alphabetically)? Assign the first possible value to this variable, and perform forward checking by crossing out values in the table below. Again the table on the right is for you to use in case you believe you made a mistake.

Variable F is selected and gets assigned value A8.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~
P	~~A8~~	~~A9~~	A10	A11	W8	~~W9~~	W10	W11
S	~~A8~~	A9	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~

Have we arrived at a dead end (i.e., has any of the domains become empty)?

No.

(f) **(4 pt)** We return to the result from enforcing just the unary constraints, which we did in (c). Select the variable S and assign the value A9. Enforce arc consistency by crossing out values in the table below.

F	A8	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
H	~~A8~~	~~A9~~	A10	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
P	~~A8~~	~~A9~~	~~A10~~	~~A11~~	W8	~~W9~~	~~W10~~	~~W11~~
S	~~A8~~	A9	~~A10~~	~~A11~~	~~W8~~	~~W9~~	~~W10~~	~~W11~~
T	~~A8~~	~~A9~~	~~A10~~	~~A11~~	~~W8~~	~~W9~~	W10	~~W11~~

(g) **(2 pt)** Compare your answers to (d) and to (f). Does arc consistency remove more values or less values than forward checking does? Explain why.

Arc consistency removes more values by checking more relationships between variables: AC checks consistency between every pair of variables, and re-checks after domain pruning, while FC only checks between assigned and unassigned variables.

(h) **(1 pt)** Check your answer to (f). Without backtracking, does any solution exist along this path? Provide the solution(s) or state that there is none.

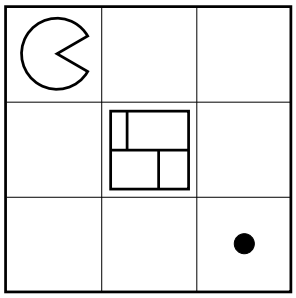
AC along this path gives 1 solution: F: A8 H: A10 P: W8 S: A9 T: W10

Backtracking is unnecessary since the constraints have been enforced by arc consistency and only single values remained in each domain.

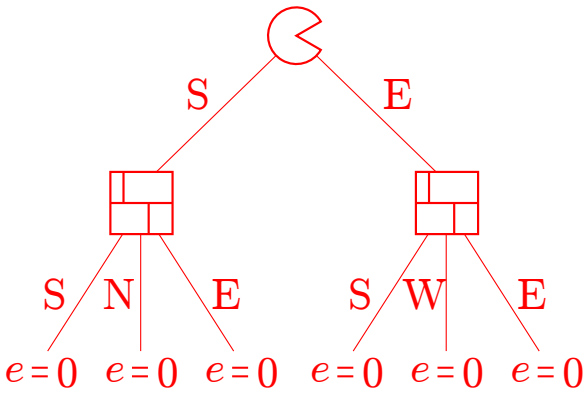
5. (24 points) Surrealist Pacman

In the game of Surrealist Pacman, Pacman plays against a moving wall. On Pacman's turn, Pacman must move in one of the four cardinal directions, and must move into an unoccupied square. On the wall's turn, the wall must move in one of the four cardinal directions, and must move into an unoccupied square. The wall cannot move into a dot-containing square. Staying still is not allowed by either player. Pacman's score is always equal to the number of dots he has eaten.

The first game begins in the configuration shown below. Pacman moves first.



(a) (2 pt) Draw a game tree with one move for each player. Draw only the legal moves.



(b) (1 pt) According to the depth-limited game tree you drew above what is the value of the game? Use Pacman's score as your evaluation function.

0, since all leaves have value 0 as Pacman cannot eat a dot in one move.

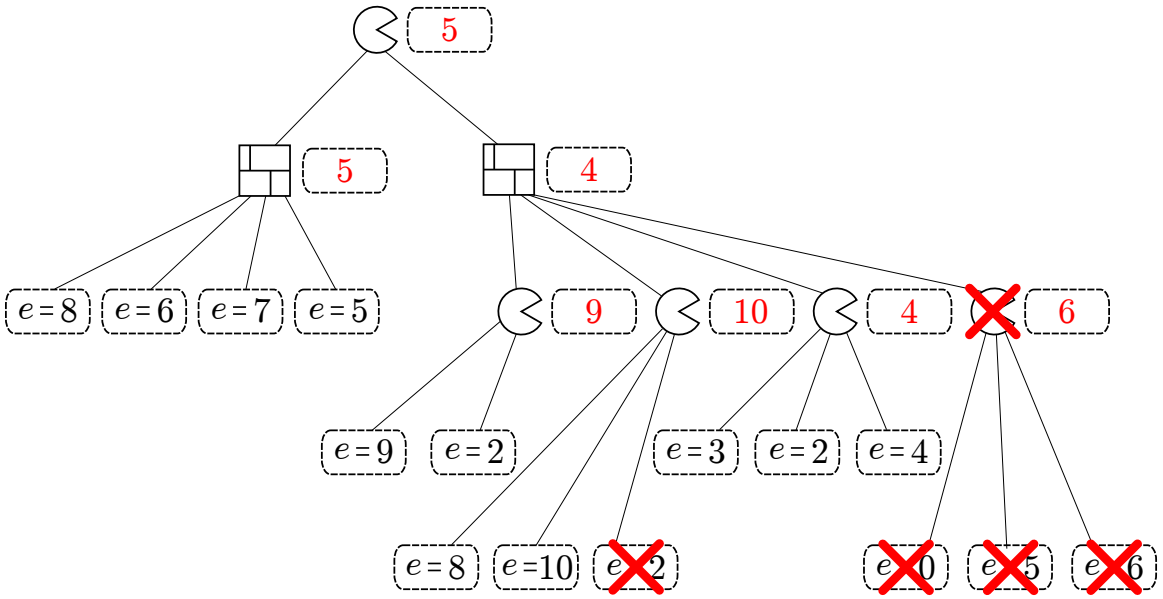
(c) (1 pt) If we were to consider a game tree with ten moves for each player (rather than just one), what would be the value of the game as computed by minimax?

1. Pacman can force a win and eat the dot in ten moves, so the score increases to 1 once the dot is eaten and remains there. For an example of such a win, consider the sequences E-S-S-E or E-S-E-S by Pacman and the movement rules of the players.

A second game is played on a more complicated board. A partial game tree is drawn, and leaf nodes have been scored using an (unknown) evaluation function e.

(d) **(3 pt)** In the dashed boxes, fill in the values of all internal nodes using the minimax algorithm.

(e) **(3 pt)** Cross off any nodes that are not evaluated when using alpha-beta pruning (assuming the standard left-to-right traversal of the tree).



Running alpha-beta pruning on the game tree.

Root: $\alpha = -\infty, \beta = \infty$

- - Left wall: $\alpha = -\infty, \beta = \infty$

- - - - Leaf node: e = 8. Propagate e = 8 back to parent.

- - Left wall: Current value is 8. $\alpha = -\infty, \beta = 8$. Max doesn't have a best value, so continue exploring.

- - - - Leaf node: e = 6. Propagate e = 6 back to parent.

- - Left wall: Current value is 6. $\alpha = -\infty, \beta = 6$. Max doesn't have a best value, so continue exploring.

- - - - Leaf node: e = 7. Propagate e = 7 back to parent.

- - Left wall: No update. Current value is 6. $\alpha = -\infty, \beta = 6$. Max doesn't have a best value, so continue exploring.

- - - - Leaf node: e = 5. Propagate e = 5 back to parent.

- - Left wall: Current value is 5. We're done here, so propagate 5 to root.

Root: Current value is 5. $\alpha = 5, \beta = \infty$. Explore right.

- - Right wall: $\alpha = 5, \beta = \infty$.

- - - - 1st Pac: $\alpha = 5, \beta = \infty$

- - - - - - Leaf node: e = 9. Propagate e = 9 back to parent.

- - - - 1st Pac: Current value is 9. $\alpha = 9, \beta = \infty$ MIN doesn't have a best value, so continue exploring.

- - - - - - Leaf node: e = 2. Propagate e = 2 back to parent.

- - - - 1st Pac: No change. Current value is 9. Propagate 9 to parent.

- - Right wall: Current value is now 9. $\alpha = 5, \beta = 9$. MIN wants anything less than 9 at this point, but it's still possible for MAX to get more than 5. Continue exploring.

- - - - 2nd Pac: $\alpha = 5, \beta = 9$

- - - - - - Leaf node: e = 8. Propagate e = 8 back to parent.

- - - - 2nd Pac: Current value is now 8. $\alpha = 8, \beta = 9$. Again, still possible for both players to benefit (Imagine value = 8.5). Continue exploring.

- - - - - - Leaf node: e = 10. Propagate e = 10 back to parent.

- - - - 2nd Pac: Current value is now 10. So now, we know that $v > \beta$, which means that one of the players is going to be unhappy. MAX wants something more than 10, but MIN is only satisfied with something less than 9, so we don't have to keep exploring.

- - - - *PRUNE e = 2.*

- - - - 2nd Pac: returns value of 10 to parent.

- - Left Wall: No change in value, current value is still 9. $\alpha = 5, \beta = 9$. Again, still possible for both players to benefit, so continue exploring.

- - - - 3rd Pac: $\alpha = 5, \beta = 9$

- - - - - - Leaf node: e = 3. Propagate e = 3 back to parent.

- - - - 3rd Pac: Current value is 3. $\alpha = 5, \beta = 9$. Continue exploring.

- - - - - - Leaf node: e = 2. Propagate e = 2 back to parent.

- - - - 3rd Pac: No change in value. Current value is 3. $\alpha = 5, \beta = 9$. Continue exploring.

- - - - - - Leaf node: e = 4. Propagate e = 4 back to parent.

- - - - 3rd Pac: Current value is 4. We're done, so return value of 4 to parent.

- - Left Wall: Current value becomes 4. At this point, we know that MIN wants anything that is less than or equal to 4. However, MAX is only satisfied with something that is 5 or greater. Hence, we don't need to explore the rest of the children of this node since MAX will never let the game get down to this branch.- -

Prune rest

(Filling values returned by alpha-beta or not crossing off children of a crossed off node were not penalized.)

Suppose that this evaluation function has a special property: it is known to give the correct minimax value of any internal node to within 2, and the correct minimax values of the leaf nodes exactly. That is, if v is the true minimax value of a particular node, and e is the value of the evaluation function applied to that node, $e-2 \leq v \leq e+2$, and $v = e$ if the node is a dashed box in the tree below.

Using this special property, you can modify the alpha-beta pruning algorithm to prune more nodes.

(f) (10 pt) Standard alpha-beta pseudocode is given below (only the max-value recursion). Fill in the boxes on the right to replace the corresponding boxes on the left so that the pseudocode prunes as many nodes as possible, taking account of this special property of the evaluation function.

```
function MAX-VALUE(node, α, β)
    e ← EVALUATIONFUNCTION(node)
    if node is leaf then
        return e
    end if
    [                                ] (1)
    v ← −∞
    for child ← CHILDREN(node) do
        [ v ← MAX(v, MIN-VALUE(child, α, β)) ] (2)
        if v ≥ β then
            return v
        end if
        α ← MAX(α, v)
    end for
    return v
end function
```

Fill in these boxes:

(1)

```
if e − 2 ≥ β then
    return e − 2
end if
```

(2)

```
v ← MAX(v, MIN-VALUE(child,
MAX(α, e − 2), MIN(β, e + 2)))
```

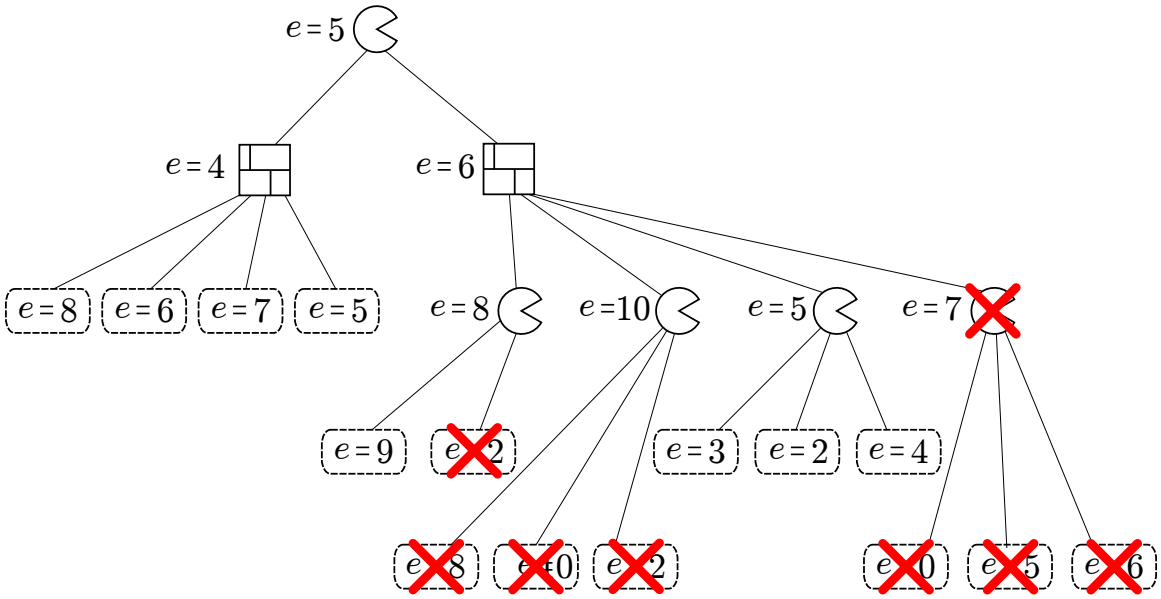
For (1), the special property guarantees $e-2 \geq v$ and $e-2$ is the smallest such value, so v is substituted out without violating the β inequality.

For (2), both sides of the inequality for the special property are used to bound the best action value for MAX α and the best action value for MIN β. α and β are sharpened whenever the ± 2 guarantee on e gives a higher lower-bound (α) or lower upper-bound (β).

(Variations are possible.)

The same game tree is shown below, with the evaluation function applied to *internal* as well as leaf nodes.

(g) **(4 pt)** In the game tree below cross off any nodes that can be pruned assuming the special property holds true. If you are not sure you correctly formalized into pseudo-code your intuition on how to exploit the special property for improved pruning, make sure to annotate your pruned nodes with a brief explanation of why each of them was pruned.



Running pruning on game tree in detail. W1 and W2 refer to the left and right wall nodes respectively. P1, P2, P3, P4 refer to Pacman nodes on the third level, left to right.

Root: Evaluation function returns 5. Range of value: [3, 7]. Set $\alpha : 3, \beta : 7$ and explore(W1, α, β).

- W1: Evaluation function returns 4. Range of value: [2,6]. Set $\alpha : 3, \beta : 6$.

- - Leaf node: $e = 8$. Send 8 back to W1.

- W1: $v = 8$. $v < \alpha$? No. This means that MAX might still prefer this path.

- - Leaf node: $e = 6$. Send 6 back to W1.

- W1: $6 < 8$ so $v = 6$. $v < \alpha$? No. Continue exploring.

- - Leaf node: $e = 7$. Send 7 back to W1.

- W1: $7 > 6$, so no change, $v = 6$. $v < \alpha$? No. Continue exploring.

- - Leaf node: $e = 5$. Send 5 back to W1.

- W1: $5 < 6$, so $v = 5$. Done. Send 5 back to root.

Root: Gets value 5, so $v = 5$. $\alpha : \max(3, 5) = 5, \beta : 7$. Still exploring right branch since MAX could get a value $5 \leq v \leq 7$. Explore(W2, α, β).

- W2: Evaluation function returns 6. Range of values $[4, 8]$. $\alpha : 5, \beta : 7$, explore(P1,α, β).

- - P1: Evaluation function returns 8. Range: $[6, 10]$, $\alpha : 6, \beta : 7$.

- - - Leaf node: $e = 9$. Send $e = 9$ back to P1.

- - P1: $v = 9$. $v > \beta$? *Yes!*. We can prune here since P1 wants any value > 9. However, at the root we know that the maximum value that MAX can get is 7. Hence, there is no way the game can get down to P1. (Meaning that the value at the root can not be 9).

- - - Leaf node: Prune $e = 2$. Return 9 to W2.

- W2: $v = 9$. α, β don't change: $\alpha : 5, \beta : 7$. Explore(P2, α, β).

- - P2: Evaluation function returns 10. Range $[8, 12]$, $\alpha : 8, \beta : 7$. Notice that the best value for MIN that can be achieved at P2 is 8. However, the best value for MIN at the root is 7. Hence, there's no way the game can get down to P2. (Meaning the value of the root can not by 8). *Prune all of P2's children!*. We can return 8 to W2 since we know that there is some other path through W2 that yields a reward ≤ 7.

- W2: $v : \min(8, 9) = 8, \alpha : 5, \beta : 7$. $v < \alpha$? No! Explore(P3,α, β).
- - P3: Evaluation function returns 5. Range: $[3, 7], \alpha : 5, \beta : 7$.
- - - Leaf node: $e = 5$. Send 3 back to P3.
- - P3: $v = 3. v > \beta$? No! Meaning MIN might still prefer this branch. $\alpha : 5, \beta : 7$.
- - - Leaf node: $e = 2$. Send 2 back to P3.
- - P3: $v = 3. v > \beta$? No! $\alpha : 5, \beta : 7$.
- - - Leaf node: $e = 4$. Send 4 back to P3.
- - P3: $v = 4$. Done. Return 4 to W2.
- W2: $v : \min(8, 4) = 4, \alpha : 5, \beta : 7$. $v < \alpha$? *Yes!* Since MAX can guarantee a value of 5 and MIN will only accept something < 4, don't need to explore any further. *Prune P4 and all its children.* Return 4 to root.

Root: *Done.*

6. (19 points) Multiple Choice and Short Answer Questions

Each true/false question is worth 1 point. Leaving a question blank is worth 0 points. **Answering incorrectly is worth −1 point.**

For the questions that are not True/False, answer as concisely as possible (and no points are subtracted for a wrong answer to these).

(a) (7 pt) Consider a graph search problem where for every action, the cost is at least ϵ, with $\epsilon > 0$. Assume the used heuristic is consistent.

True **False** Depth-first graph search is guaranteed to return an optimal solution.
False. Depth first search has no guarantees of optimality. Further, it measures paths in length and not cost.

True **False** Breadth-first graph search is guaranteed to return an optimal solution.
False. Breadth first search has no guarantees of optimality unless the actions all have the same cost, which is not the case here.

True **False** Uniform-cost graph search is guaranteed to return an optimal solution.
True. UCS expands paths in order of least total cost so that the optimal solution is found.

True **False** Greedy graph search is guaranteed to return an optimal solution.
False. Greedy search makes no guarantees of optimality. It relies solely on the heuristic and not the true cost.

True **False** A* graph search is guaranteed to return an optimal solution.
True, since the heuristic is consistent in this case.

True **False** A* graph search is guaranteed to expand no more nodes than depth-first graph search.
False. Depth-first graph search could, for example, go directly to a sub-optimal solution.

True **False** A* graph search is guaranteed to expand no more nodes than uniform-cost graph search.
True. The heuristic could help to guide the search and reduce the number of nodes expanded. In the extreme case where the heuristic function returns zero for every state, A* and UCS will expand the same number of nodes. In any case, A* with a consistent heuristic will never expand more nodes than UCS.

(b) (2 pt) Iterative deepening is sometimes used as an alternative to breadth first search. Give one advantage of iterative deepening over BFS, and give one disadvantage of iterative deepening as compared with BFS. Be concise and specific! Advantage: iterative deepening requires less memory (limited to the current depth). Disadvantage: iterative deepening repeats computations and therefore can require additional run time.

(c) (2 pt) Consider two different A* heuristics, $h_1(s)$ and $h_2(s)$, that are each admissible. Now, combine the two heuristics into a single heuristic, using some (not yet specified) function g. Give the choice for g that will result in A* expanding a minimal number of nodes while still guaranteeing admissibility. Your answer should be a heuristic function of the form $g(h_1(s), h_2(s))$.
$g = \max(h_1(s), h_2(s))$. Consider the true cost $h^*(s)$. For admissibility, both $h_1(s) \leq h^*(s)$ and $h_2(s) \leq h^*(s)$, and their max can be no larger. $h^*(s)$ leads to expanding the minimal number of nodes: in particular it expands the nodes of an optimal path to a goal and no more. g is the closest heuristic to h^* of guaranteed admissibility by h_1 and h_2 and so expands a minimal number of nodes for functions over h_1, h_2.

(d) (3 pt) Let $h_1(s)$ be an admissible A* heuristic. Let $h_2(s) = 2h_1(s)$. Then:

True **False** The solution found by A* tree search with h_2 is guaranteed to be an optimal solution.
False. h_2 is not guaranteed to be admissible since only one side of the admissibility inequality is doubled.

True **False** The solution found by A* tree search with h_2 is guaranteed to have a cost at most twice as much as the optimal path.
True. In A* tree search we always have that as long as the optimal path to the goal has not been found, a prefix of this optimal path has to be on the fringe. Hence, if a non-optimal solution is found, then at time of popping the non-optimal path from the fringe, a path that is a prefix of the optimal path to the goal is sitting on the fringe. The cost $\bar{g}$ of a

non-optimal solution when popped is its f-cost. The prefix of the optimal path to the goal has an f-cost of $g + h_0 = g + 2h_1 \leq 2(g + h_1) \leq 2C^*$, with C^* the optimal cost to the goal. Hence we have that $\bar{g} \leq 2C^*$ and the found path is at most twice as long as the optimal path.

True **False** The solution found by A* graph search with h_2 is guaranteed to be an optimal solution.
False. h_2 is not guaranteed to be admissible and graph search further requires consistency for optimality.

(e) **(2 pt)** Consider a CSP with three variables: A, B, and C. Each of the three variables can take on one of two values: either 1 or 2. There are three constraints: $A \neq B$, $B \neq C$, and $A \neq C$. In the table below, cross off all values that are eliminated by enforcing *arc-consistency*. Also write one sentence interpreting your answer.

A	1	2
B	1	2
C	1	2

No variables should be crossed out. There is no solution to this CSP, but arc-consistency does not detect this. There is a consistent assignment in the head for each value of the tail and no values are eliminated.

(f) **(3 pt)** Consider an adversarial game tree where the root node is a maximizer, and the minimax value of the game is v_M. Now, also consider an otherwise identical tree where every minimizer node is replaced with a chance node (with an arbitrary but known probability distribution). The expectimax value of the modified game tree is v_E.

True **False** v_M is guaranteed to be less than or equal to v_E.
True. The maximizer is guaranteed to be able to achieve v_M if the minimizer acts optimally. He can potentially do better if the minimizer acts suboptimally (e.g. by acting randomly). The expectation at chances nodes can be no less than the minimum of the nodes' successors.

True **False** Using the optimal *minimax* policy in the game corresponding to the modified (chance) game tree is guaranteed to result in a payoff of at least v_M.
True. The minimax policy is always guaranteed to achieve payoff of at least v_M, no matter what the minimizer does.

True **False** Using the optimal *minimax* policy in the game corresponding to the modified (chance) game tree is guaranteed to result in a payoff of at least v_E.
False. In order to achieve v_E in the modified (chance) game, the maximizer may need to change his or her strategy. The minimax strategy may avoid actions where the minimum value is less than the expectation. Moreover, even if the maximizer followed the expectimax strategy, he or she is only guaranteed v_E *in expectation*.