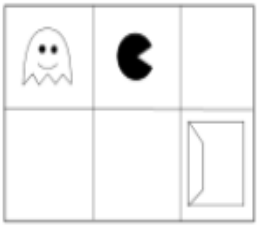
(f) (2 pt) You are running minimax in a large game tree. However, you know that the minimax value is between $x - \epsilon$ and $x + \epsilon$, where x is a real number and ϵ is a small real number. You know nothing about the minimax values at the other nodes. Describe briefly but precisely how to modify the alpha-beta pruning algorithm to take advantage of this information.

(f) (2 pt) You are running minimax in a large game tree. However, you know that the minimax value is between $x - \epsilon$ and $x + \epsilon$, where x is a real number and ϵ is a small real number. You know nothing about the minimax values at the other nodes. Describe briefly but precisely how to modify the alpha-beta pruning algorithm to take advantage of this information. Initialize the alpha and beta values at the root to $\alpha = x - \epsilon$ and $\beta = x + \epsilon$ then run alpha-beta as normal. This will prune all of the game tree out of our known minimax bounds.

שאלה 2

3. (12 points) Pac-Infiltration

A special operations Pacman must reach the exit (located on the bottom right square) in a ghost-filled maze, as shown to the right. Every time step, each agent (i.e., Pacman and each ghost) can move to any adjacent non-wall square or stay in place. Pacman moves first and the ghosts move collectively and simultaneously in response (so, consider the ghosts to be one agent). Pacman's goal is to exit as soon as possible. If Pacman and a ghost ever occupy the same square, either through Pacman or ghost motion, the game ends, and Pacman receives utility −1. If Pacman moves into the exit square, he receives utility +1 and the game ends. The ghosts are adversarial and seek to minimize Pacman's utility.



For parts (a) and (b), consider the 2x3 board with one ghost shown above. In this example, it is Pacman's move.

(a) (2 pt) What is the minimax value of this state?

(b) (2 pt) If we perform a depth-limited minimax search from this state, searching to depth $d = 1, 2, 3, 4$ and using zero as the evaluation function for non-terminal states, what minimax values will be computed? Reminder: each layer of depth includes one move from each agent.

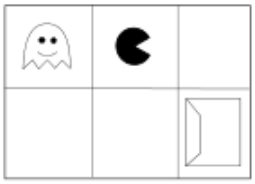
d	Minimax value
1	
2	
3	
4	

For parts (c) and (d), consider a grid of size $N \times M$ containing K ghosts. Assume we perform a depth-limited minimax search, searching to depth d.

(c) (2 pt) How many states will be expanded (asymptotically) in terms of d using standard minimax? Your answer should use all the relevant variables needed and should not be specific to any particular map or configuration.

3. (12 points) Pac-Infiltration

A special operations Pacman must reach the exit (located on the bottom right square) in a ghost-filled maze, as shown to the right. Every time step, each agent (i.e., Pacman and each ghost) can move to any adjacent non-wall square or stay in place. Pacman moves first and the ghosts move collectively and simultaneously in response (so, consider the ghosts to be one agent). Pacman's goal is to exit as soon as possible. If Pacman and a ghost ever occupy the same square, either through Pacman or ghost motion, the game ends, and Pacman receives utility -1. If Pacman moves into the exit square, he receives utility $+1$ and the game ends. The ghosts are adversarial and seek to minimize Pacman's utility.



For parts (a) and (b), consider the 2x3 board with one ghost shown above. In this example, it is Pacman's move.

(a) (2 pt) What is the minimax value of this state?
+1 because Pacman is guaranteed a victory with his first-move advantage

(b) (2 pt) If we perform a depth-limited minimax search from this state, searching to depth $d = 1, 2, 3, 4$ and using zero as the evaluation function for non-terminal states, what minimax values will be computed? Reminder: each layer of depth includes one move from each agent.

d	Minimax value
1	0
2	1
3	1
4	1

For parts (c) and (d), consider a grid of size $N \times M$ containing K ghosts. Assume we perform a depth-limited minimax search, searching to depth d.

(c) (2 pt) How many states will be expanded (asymptotically) in terms of d using standard minimax? Your answer should use all the relevant variables needed and should not be specific to any particular map or configuration. $(5^{K+1})^d$. The branching factor for Pacman is 5 since he can move to any adjacent square (4 choices) or stay put (+1 choice). The branching factor for ghosts is 5^K since each of the K ghosts has the same 5 movement options. The number of nodes expanded is the branching factor raised to the depth d.

(d) (1 pt) For small boards, why is standard recursive minimax a poor choice for this problem? Your answer should be no more than one sentence!
For small boards the depth of the goal (exit with +1) will be small, and since the goal ends the game the search need not be continued and the further expansion done by standard minimax is a waste.

שאלה 3

4. (7 points) Games: Multiple Choice

In the following problems please choose **all** the answers that apply. You may circle more than one answer. You may also circle no answers, if none apply.

(a) (2 pt) In the context of adversarial search, α-β pruning

(i) can reduce computation time by pruning portions of the game tree

(ii) is generally faster than minimax, but loses the guarantee of optimality

(iii) always returns the same value as minimax for the root of the tree

(iv) always returns the same value as minimax for all nodes on the leftmost edge of the tree, assuming successor game states are expanded from left to right

(v) always returns the same value as minimax for all nodes of the tree

(b) (2 pt) Consider an adversarial game in which each state s has minimax value $v(s)$. Assume that the maximizer plays according to the optimal minimax policy π, but the opponent (the minimizer) plays according to an unknown, possibly suboptimal policy π'. Which of the following statements are true?

(i) The score for the maximizer from a state s under the maximizer's control could be greater than $v(s)$.

(ii) The score for the maximizer from a state s under the maximizer's control could be less than $v(s)$.

(iii) Even if the opponent's strategy π' were known, the maximizer should play according to π.

(iv) If π' is optimal and known, the outcome from any s under the maximizer's control will be $v(s)$.

4. (7 points) Games: Multiple Choice

In the following problems please choose **all** the answers that apply. You may circle more than one answer. You may also circle no answers, if none apply.

(a) (2 pt) In the context of adversarial search, α-β pruning

(i) can reduce computation time by pruning portions of the game tree

(ii) is generally faster than minimax, but loses the guarantee of optimality

(iii) always returns the same value as minimax for the root of the tree

(iv) always returns the same value as minimax for all nodes on the leftmost edge of the tree, assuming successor game states are expanded from left to right

(v) always returns the same value as minimax for all nodes of the tree

(b) (2 pt) Consider an adversarial game in which each state s has minimax value $v(s)$. Assume that the maximizer plays according to the optimal minimax policy π, but the opponent (the minimizer) plays according to an unknown, possibly suboptimal policy π'. Which of the following statements are true?

(i) The score for the maximizer from a state s under the maximizer's control could be greater than $v(s)$.

(ii) The score for the maximizer from a state s under the maximizer's control could be less than $v(s)$.

(iii) Even if the opponent's strategy π' were known, the maximizer should play according to π.

(iv) If π' is optimal and known, the outcome from any s under the maximizer's control will be $v(s)$.

(c) (3 pt) Consider a very deep game tree where the root node is a maximizer, and the complete-depth minimax value of the game is known to be v_∞. Similarly, let π_∞ be the minimax-optimal policy. Also consider a depth-limited version of the game tree where an evaluation function replaces any tree regions deeper than depth 10. Let the minimax value of the depth-limited game tree be v_{10} for the current root node, and let π_{10} be the policy which results from acting according to a depth 10 minimax search at every move. Which of the following statements are true?

(i) v_∞ may be greater than or equal to v_{10}.

(ii) v_∞ may be less than or equal to v_{10}.

(iii) Against a perfect opponent, the actual outcome from following π_{10} may be greater than v_∞.

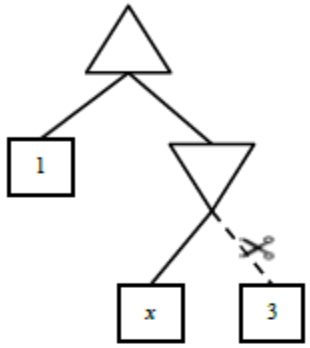
(iv) Against a perfect opponent, the actual outcome from following π_{10} may be less than v_∞.

This assumes that the perfect opponent is playing with infinite depth lookahead.

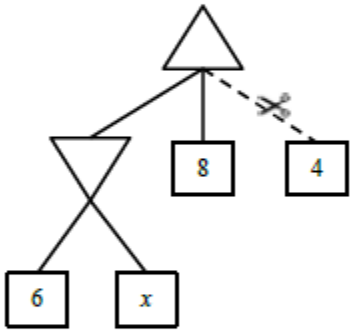
שאלה 4

Q7. [8 pts] Games: Alpha-Beta Pruning

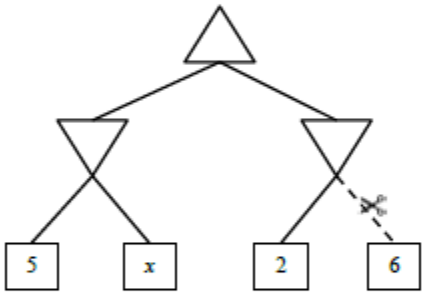
For each of the game-trees shown below, state for which values of x the dashed branch with the scissors will be pruned. If the pruning will not happen for any value of x write "none". If pruning will happen for all values of x write "all".



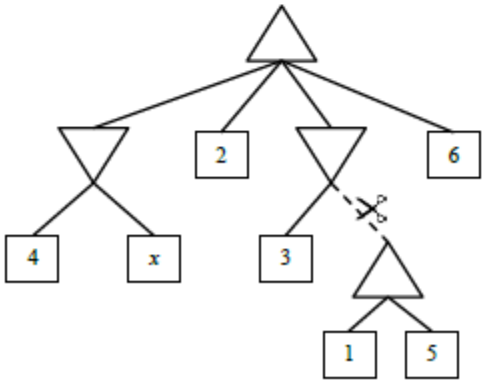
(a) Example Tree. Answer: $x \leq 1$.



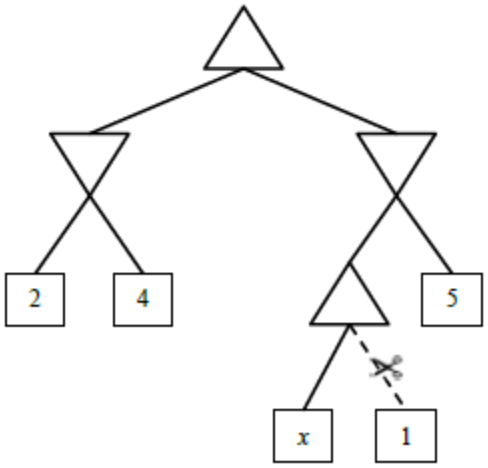
(b) Tree 1. Answer: _____



(c) Tree 2. Answer: _____



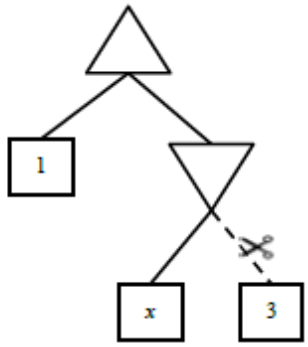
(d) Tree 3. Answer: _____



(e) Tree 4. Answer: _____

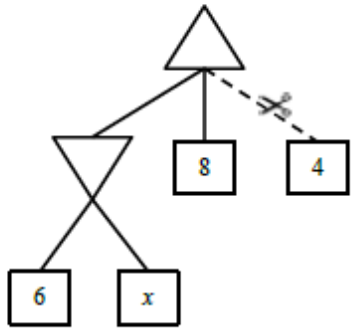
Q7. [8 pts] Games: Alpha-Beta Pruning

For each of the game-trees shown below, state for which values of x the dashed branch with the scissors will be pruned. If the pruning will not happen for any value of x write "none". If pruning will happen for all values of x write "all".

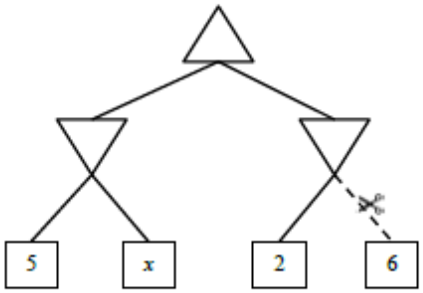


(a) Example Tree. Answer: $x \leq 1$.

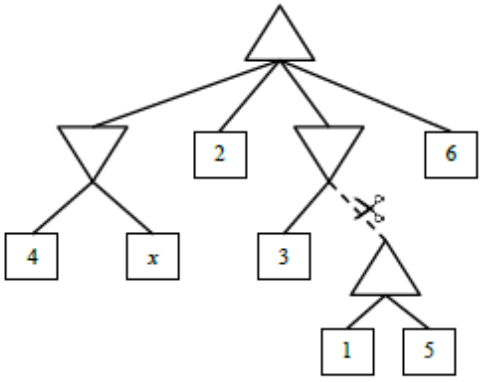
We are assuming that nodes are evaluated left to right and ties are broken in favor of the latter nodes. A different evaluation order would lead to different interval bounds, while a different tie breaking strategies could lead to strict inequalities ($>$ instead of $\geq$).
Successor enumeration order and tie breaking rules typically impact the efficiency of alpha-beta pruning.



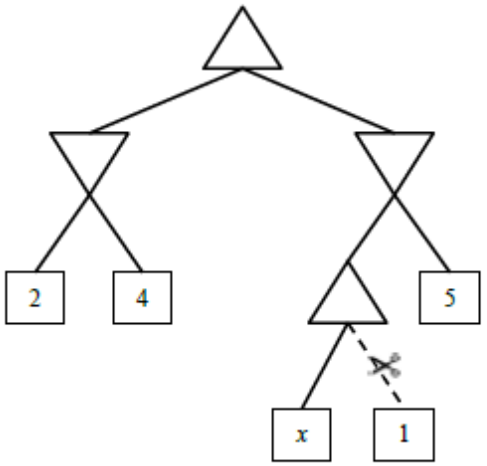
(b) Tree 1. Answer: None



(c) Tree 2. Answer: $x \geq 2$



(d) Tree 3. Answer: $x \geq 3$.



(e) Tree 4. Answer: None

שאלה 5

Q5. [9 pts] Games: Three-Player Cookie Pruning

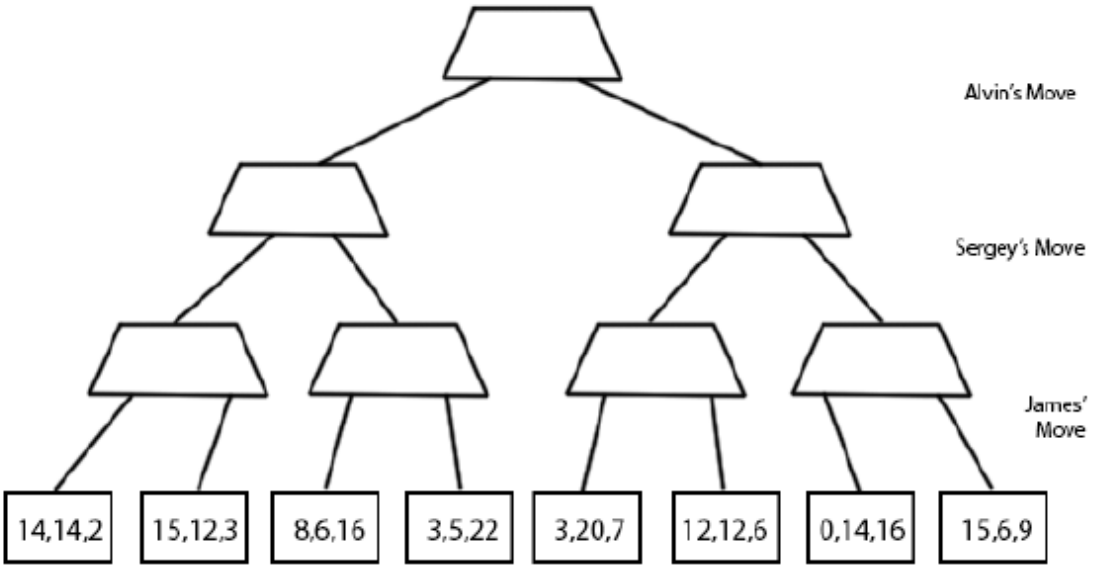
Three of your TAs, Alvin, Sergey, and James rent a cookie shuffler, which takes in a set number of cookies and groups them into 3 batches, one for each player. The cookie shuffler has three levers (with positions either UP or DOWN), which act to control how the cookies are distributed among the three players. Assume that 30 cookies are initially put into the shuffler.

Each player controls one lever, and they act in turn. Alvin goes first, followed by Sergey, and finally James. Assume that all players are able to calculate the payoffs for every player at the terminal nodes. Assume the payoffs at the leaves correspond to the number of cookies for each player in their corresponding turn order. Hence, an utility of (7,10,13) corresponds to Alvin getting 7 cookies, Sergey getting 10 cookies, and James getting 13 cookies. No cookies are lost in the process, so the sum of cookies of all three players must equal the number of cookies put into the shuffler. Players want to maximize their own number of cookies.

(a) [3 pts] **What is the utility triple propagated up to the root?**

(b) [6 pts] **Is pruning possible in this game? Fill in "Yes" or "No". If yes, cross out all nodes (both leaves and intermediate nodes) that get pruned. If no, explain in one sentence why pruning is not possible.** Assume the tree traversal goes from left to right.

- ○ Yes.
- ○ No, Reasoning:



Q5. [9 pts] Games: Three-Player Cookie Pruning

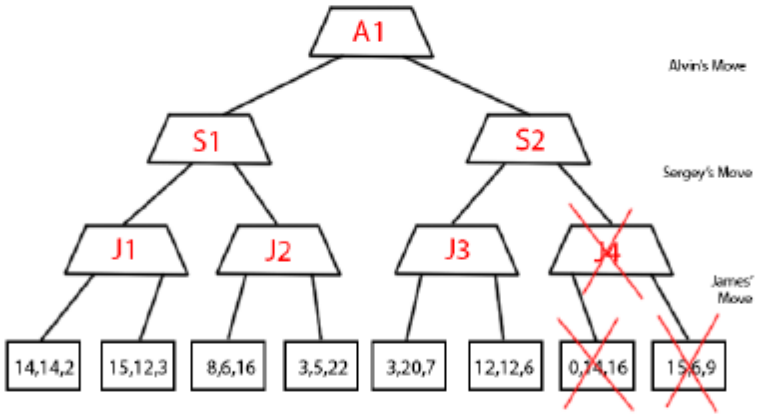
Three of your TAs, Alvin, Sergey, and James rent a cookie shuffler, which takes in a set number of cookies and groups them into 3 batches, one for each player. The cookie shuffler has three levers (with positions either UP or DOWN), which act to control how the cookies are distributed among the three players. Assume that 30 cookies are initially put into the shuffler.

Each player controls one lever, and they act in turn. Alvin goes first, followed by Sergey, and finally James. Assume that all players are able to calculate the payoffs for every player at the terminal nodes. Assume the payoffs at the leaves correspond to the number of cookies for each player in their corresponding turn order. Hence, an utility of (7,10,13) corresponds to Alvin getting 7 cookies, Sergey getting 10 cookies, and James getting 13 cookies. No cookies are lost in the process, so the sum of cookies of all three players must equal the number of cookies put into the shuffler. Players want to maximize their own number of cookies.

(a) [3 pts] **What is the utility triple propagated up to the root?** 15,12,3

(b) [6 pts] **Is pruning possible in this game? Fill in "Yes" or "No". If yes, cross out all nodes (both leaves and intermediate nodes) that get pruned. If no, explain in one sentence why pruning is not possible.** Assume the tree traversal goes from left to right.

- ● Yes.
- ○ No, Reasoning:



Left lowest subtree: James chooses the node (15,12,3) at node J1. This gets propagated up to S1. Sergey doesn't know what value he can get on his right child, so we explore that. Upon propagating (8,6,16) to J2, we must explore J2's right child since there could be a triple better for James' best option (greater than 16) and better than Sergey's best option. (greater than 12). (15,12,3) gets propagated up to A1.

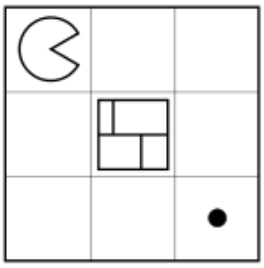
Alvin might have a good option (greater than 15) in the right subtree, so we explore down. On the (3,20,7) node, we propagate this up to J3. We need to continue going down this path because Sergey doesn't know if he can get more than 20, and Alvin doesn't know if he can get more than 15 (A1's value). Hence, (12,12,6) is explored. (3,20,7) is propagated to S2. Now, we can guarantee that Sergey will prefer any cookie count over 20. But, because the sum of cookies must be 30, this means that Alvin can get no more than 10 cookies in the right subtree. Hence, we can immediately prune any children of S2.

שאלה 6

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.



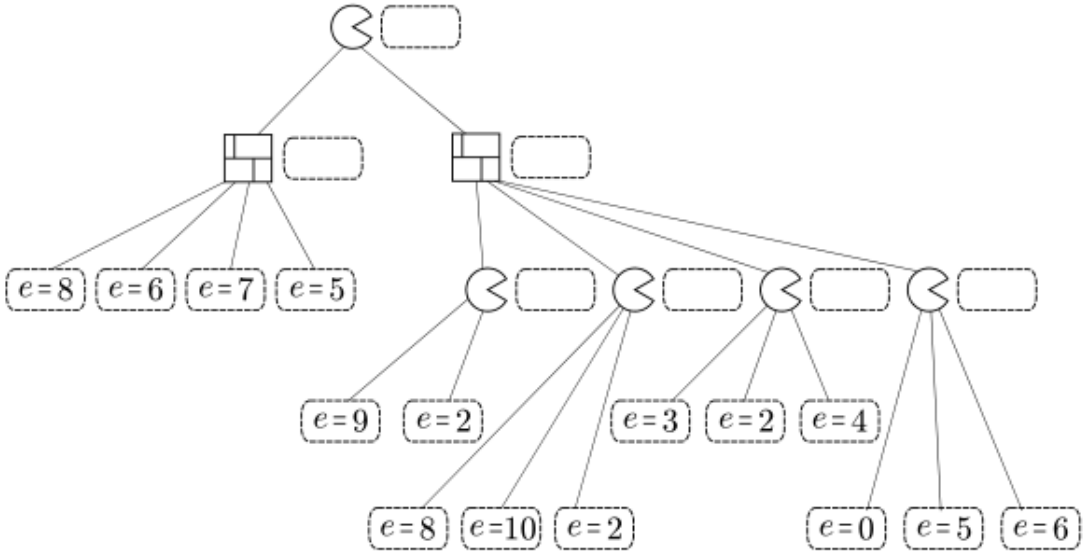
(a) (2 pt) Draw a game tree with one move for each player. Draw only the legal moves.

(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.

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).



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
```

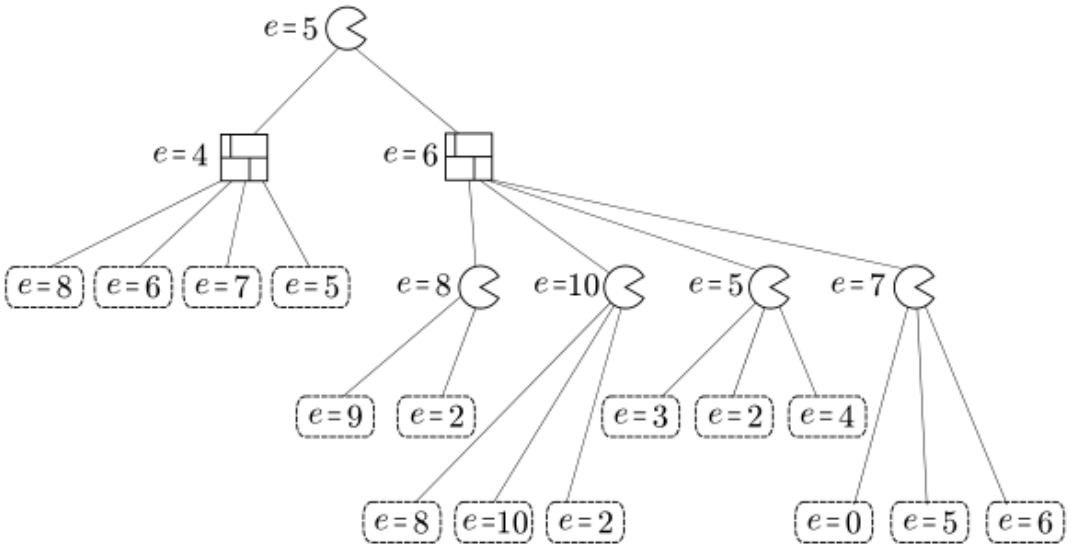
Fill in these boxes:

(1)

(2)

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

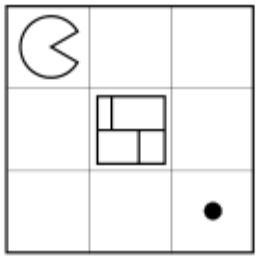
(g) **(4 pt)** In the game tree below cross off any nodes that can be pruned assuming the special property holds true. If you are not sure you correctly formalized into pseudo-code your intuition on how to exploit the special property for improved pruning, make sure to annotate your pruned nodes with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



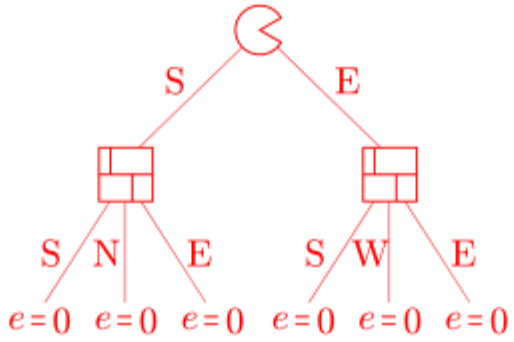
5. (24 points) Surrealist Pacman

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.

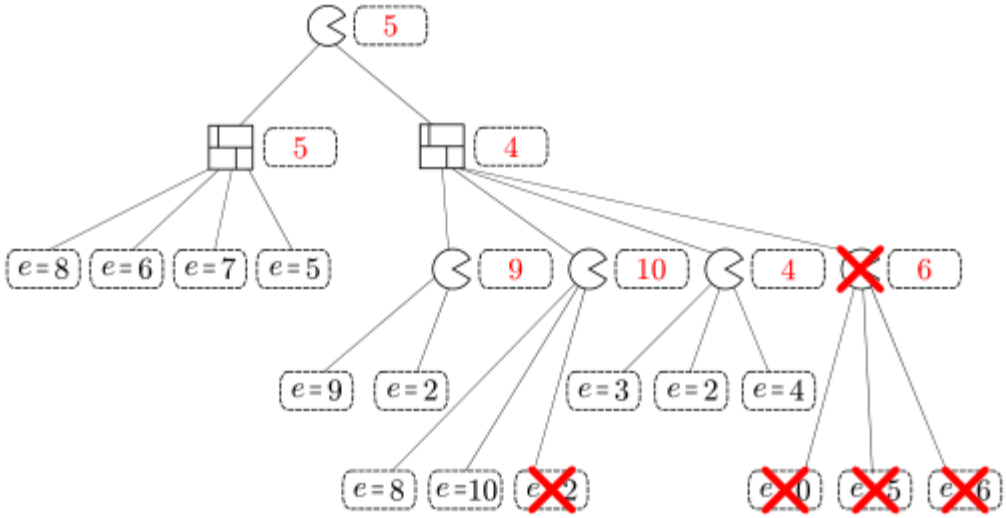
(c) (1 pt) If we were to consider a game tree with ten moves for each player (rather than just one), what would be the value of the game as computed by minimax?

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)))
```

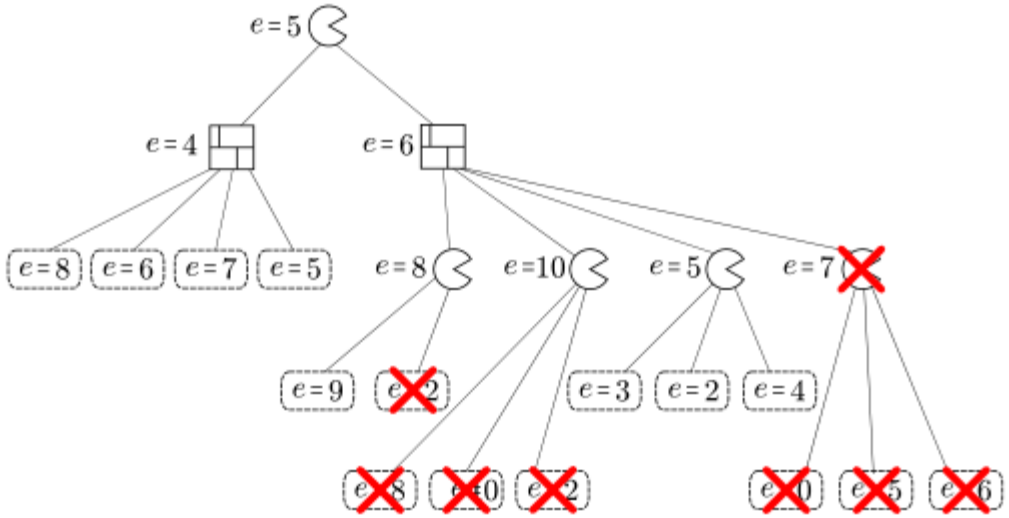
For (1), the special property guarantees $e - 2 \geq v$ and $e - 2$ is the smallest such value, so v is substituted out without violating the β inequality.

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 $\pm$ 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.*

שאלה 7

(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 **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 **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.

(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.*

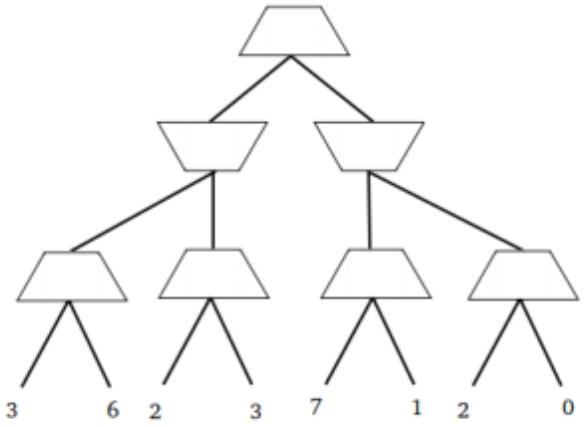
שאלה 8

Q5. [23 pts] Games with Magic

(a) Standard Minimax

(i) [2 pts] Fill in the values of each of the nodes in the following Minimax tree. The upward pointing trapezoids correspond to maximizer nodes (layer 1 and 3), and the downward pointing trapezoids correspond to minimizer nodes (layer 2). Each node has two actions available, Left and Right.

(ii) [1 pt] Mark the sequence of actions that correspond to Minimax play.



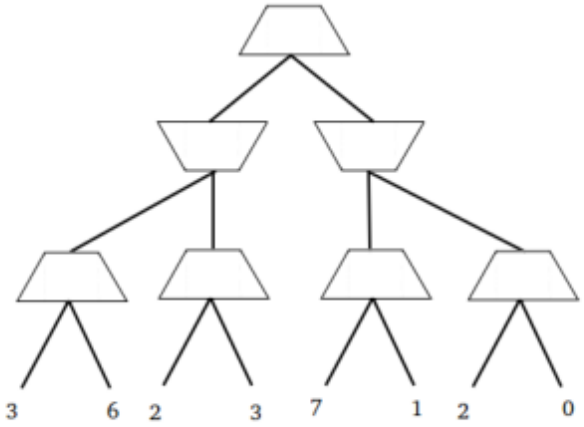
(b) Dark Magic

Pacman (= maximizer) has mastered some dark magic. With his dark magic skills Pacman can take control over his opponent's muscles while they execute their move — and in doing so be fully in charge of the opponent's move. But the magic comes at a price: every time Pacman uses his magic, he pays a price of c—which is measured in the same units as the values at the bottom of the tree.

Note: For each of his opponent's actions, Pacman has the *choice* to either let his opponent act (optimally according to minimax), or to take control over his opponent's move at a cost of c.

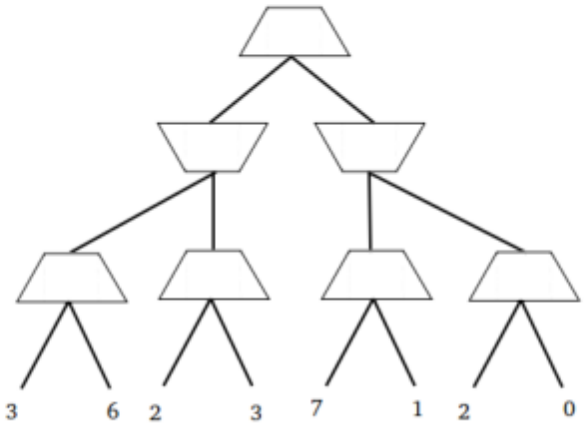
(i) [3 pts] **Dark Magic at Cost** $c = 2$

Consider the same game as before but now Pacman has access to his magic at cost $c = 2$. Is it optimal for Pacman to use his dark magic? If so, mark in the tree below where he will use it. Either way, mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



(ii) [3 pts] **Dark Magic at Cost $c = 5$**

Consider the same game as before but now Pacman has access to his magic at cost $c = 5$. Is it optimal for Pacman to use his dark magic? If so, mark in the tree below where he will use it. Either way, mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



(iii) [7 pts] **Dark Magic Minimax Algorithm**

Now let's study the general case. *Assume that the minimizer player has no idea that Pacman has the ability to use dark magic at a cost of c. I.e., the minimizer chooses their actions according to standard minimax.* You get to write the pseudo-code that Pacman uses to compute their strategy. As a starting point / reminder we give you below the pseudo-code for a standard minimax agent. Modify the pseudo-code such that it returns the optimal value for Pacman. Your pseudo-code should be sufficiently general that it works for arbitrary depth games.

```
function MAX-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← −∞
    for successor in SUCCESSORS(state) do
        v ← max(v, MIN-VALUE(successor))
    end for
    return v
end function

function MIN-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← ∞
    for successor in SUCCESSORS(state) do
        v ← min(v, MAX-VALUE(successor))
    end for
    return v
end function
```

(iv) [7 pts] **Dark Magic Becomes Predictable**

The minimizer has come to the realization that Pacman has the ability to apply magic at cost c. Hence the minimizer now doesn't play according the regular minimax strategy anymore, but accounts for Pacman's magic capabilities when making decisions. Pacman in turn, is also aware of the minimizer's new way of making decisions.

You again get to write the pseudo-code that Pacman uses to compute his strategy. As a starting point / reminder we give you below the pseudo-code for a standard minimax agent. Modify the pseudocode such that it returns the optimal value for Pacman.

```
function MAX-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← −∞
    for successor in SUCCESSORS(state) do
        v ← max(v, MIN-VALUE(successor))
    end for
    return v
end function
```

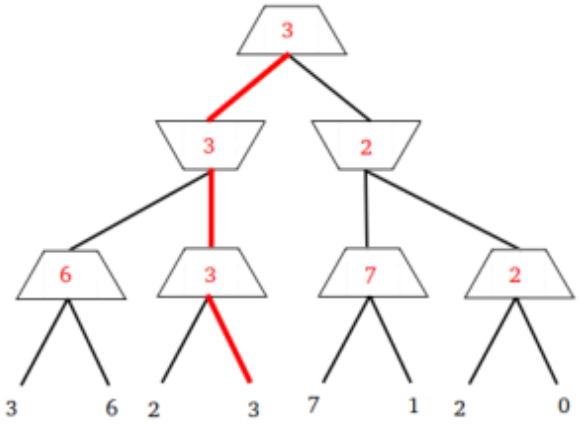
```
function MIN-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← ∞
    for successor in SUCCESSORS(state) do
        v ← min(v, MAX-VALUE(successor))
    end for
    return v
end function
```

Q5. [23 pts] Games with Magic

(a) Standard Minimax

(i) [2 pts] Fill in the values of each of the nodes in the following Minimax tree. The upward pointing trapezoids correspond to maximizer nodes (layer 1 and 3), and the downward pointing trapezoids correspond to minimizer nodes (layer 2). Each node has two actions available, Left and Right.

(ii) [1 pt] Mark the sequence of actions that correspond to Minimax play.



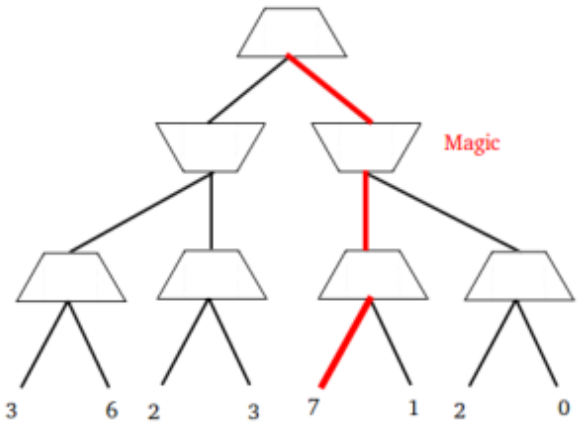
(b) Dark Magic

Pacman (= maximizer) has mastered some dark magic. With his dark magic skills Pacman can take control over his opponent's muscles while they execute their move — and in doing so be fully in charge of the opponent's move. But the magic comes at a price: every time Pacman uses his magic, he pays a price of c—which is measured in the same units as the values at the bottom of the tree.

Note: For each of his opponent's actions, Pacman has the *choice* to either let his opponent act (optimally according to minimax), or to take control over his opponent's move at a cost of c.

(i) [3 pts] **Dark Magic at Cost** $c = 2$

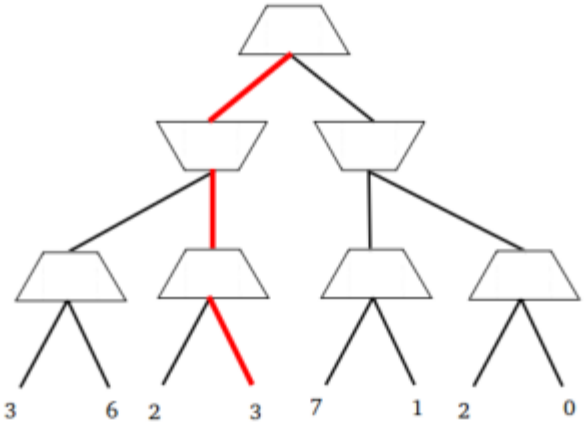
Consider the same game as before but now Pacman has access to his magic at cost $c = 2$. Is it optimal for Pacman to use his dark magic? If so, mark in the tree below where he will use it. Either way, mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



Pacman goes right and uses dark magic to get 7-2=5. Not using dark magic would result in the normal minimax value of 3. Going left and using dark magic would have resulted in 6-2=4. So, in either case using magic benefits Pacman, but using it when going right is best.

(ii) [3 pts] **Dark Magic at Cost** $c = 5$

Consider the same game as before but now Pacman has access to his magic at cost $c = 5$. Is it optimal for Pacman to use his dark magic? If so, mark in the tree below where he will use it. Either way, mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



Pacman doesn't use dark magic. Going left and using dark magic would result in 6-5=1, and going right and using dark magic would result in 7-5=2, while not using dark magic results in 3.

(iii) [7 pts] **Dark Magic Minimax Algorithm**

Now let's study the general case. *Assume that the minimizer player has no idea that Pacman has the ability to use dark magic at a cost of c. I.e., the minimizer chooses their actions according to standard minimax.* You get to write the pseudo-code that Pacman uses to compute their strategy. As a starting point / reminder we give you below the pseudo-code for a standard minimax agent. Modify the pseudo-code such that it returns the optimal value for Pacman. Your pseudo-code should be sufficiently general that it works for arbitrary depth games.

```
function MAX-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← −∞
    for successor in SUCCESSORS(state) do
        v ← max(v, MIN-VALUE(successor))
    end for
    return v
end function
```

```
function MIN-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← ∞
    for successor in SUCCESSORS(state) do
        v ← min(v, MAX-VALUE(successor))
    end for
    return v
end function
```

```
function MAX-VALUE(state)
    if state is leaf then
        return (UTILITY(state), UTILITY(state))
    end if
    v_min ← −∞
    v_max ← −∞
    for successor in SUCCESSORS(state) do
        vNext_min, vNext_max ← MIN-VALUE(successor)
        v_min ← max(v_min, vNext_min)
        v_max ← max(v_max, vNext_max)
    end for
    return (v_min, v_max)
end function
```

```
function MIN-VALUE(state)
    if state is leaf then
        return (UTILITY(state), UTILITY(state))
    end if
    v_min ← ∞
    min_move_v_max ← −∞
    v_magic_max ← −∞
    for state in SUCCESSORS(state) do
        vNext_min, vNext_max ← MAX-VALUE(successor)
        if v_min > vNext_min then
            v_min ← vNext_min
            min_move_v_max ← vNext_max
        end if
        v_magic_max ← max(vNext_max, v_magic_max)
    end for
    v_max ← max(min_move_v_max, v_magic_max − c)
    return (v_min, v_max)
end function
```

The first observation is that the maximizer and minimizer are getting different values from the game. The maximizer gets the value at the leaf minus c*(number of applications of dark magic), which we denote by v_{max}. The minimizer, as always, tries to minimize the value at the leaf, which we denote by v_{min}.

In $Max - Value$, we now compute two things.

(1) We compute the max of the children's v_{max} values, which tells us what the optimal value obtained by the maximizer would be for this node.

(2) We compute the max of the children's v_{min} values, which tells us what the minimizer thinks would happen in that node.

In $Min - Value$, we also compute two things.

(1) We compute the min of the children's v_{min} values, which tells us what the minimizer's choice would be in this node, and is being tracked by the variable v_{min}. We also keep track of the value the maximizer would get if the minimizer got to make their move, which we denote by $min\_move\_v_{max}$.

(2) We keep track of a variable v_{magic_max} which computes the maximum of the children's v_{max}.

If the maximizer applies dark magic he can guarantee himself $v_{magic\_max} - c$. We compare this with the $min\_move\_v_{max}$ from (1) and set v_{max} to the maximum of the two.

(iv) [7 pts] **Dark Magic Becomes Predictable**

The minimizer has come to the realization that Pacman has the ability to apply magic at cost c. Hence the minimizer now doesn't play according the regular minimax strategy anymore, but accounts for Pacman's magic capabilities when making decisions. Pacman in turn, is also aware of the minimizer's new way of making decisions.

You again get to write the pseudo-code that Pacman uses to compute his strategy. As a starting point / reminder we give you below the pseudo-code for a standard minimax agent. Modify the pseudocode such that it returns the optimal value for Pacman.

```
function MAX-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← −∞
    for successor in SUCCESSORS(state) do
        v ← max(v, MIN-VALUE(successor))
    end for
    return v
end function
```

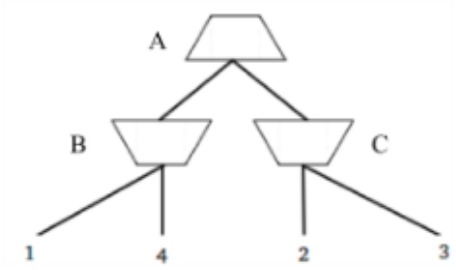
```
function MIN-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← ∞
    for successor in SUCCESSORS(state) do
        v ← min(v, MAX-VALUE(successor))
    end for
    return v
end function
```

```
function MIN-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← ∞
    v_m ← −∞
    for state in SUCCESSORS(state) do
        temp ← MAX-VALUE(successor)
        v ← min(v, temp)
        v_m ← max(v_m, temp)
    end for
    return max(v, v_m − c)
end function
```

שאלה 9

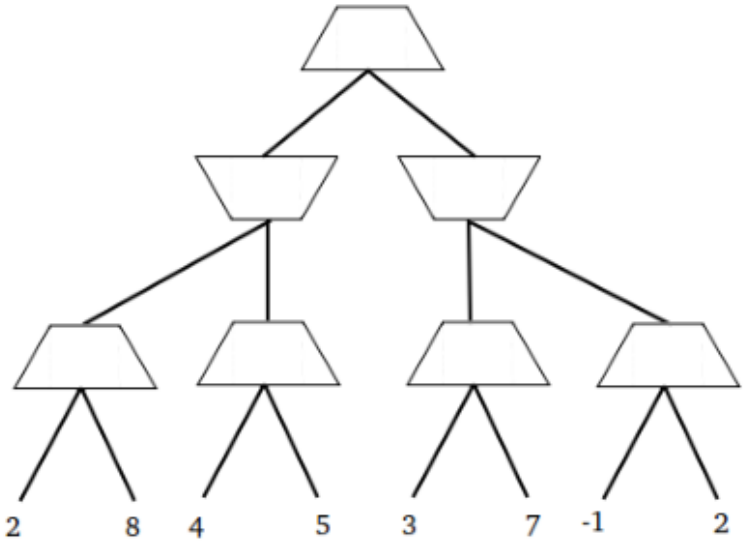
Q6. [10 pts] Pruning and Child Expansion Ordering

The number of nodes pruned using alpha-beta pruning depends on the order in which the nodes are expanded. For example, consider the following minimax tree.



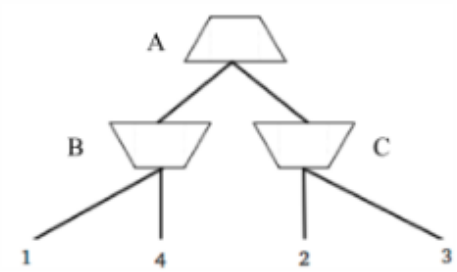
In this tree, if the children of each node are expanded from left to right for each of the three nodes then no pruning is possible. However, if the expansion ordering were to be first Right then Left for node A, first Right then Left for node C, and first Left then Right for node B, then the leaf containing the value 4 can be pruned. (Similarly for first Right then Left for node A, first Left then Right for node C, and first Left then Right for node B.)

For the following tree, give an ordering of expansion for each of the nodes that will maximize the number of leaf nodes that are never visited due the search (thanks to pruning). ***For each node, draw an arrow indicating which child will be visited first. Cross out every leaf node that never gets visited.***
Hint: Your solution should have three leaf nodes crossed out and indicate the child ordering for 6 of the 7 internal nodes.



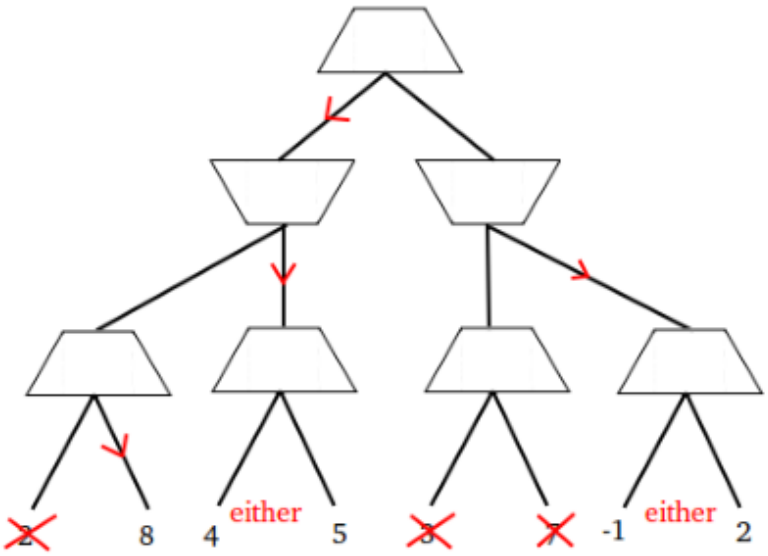
Q6. [10 pts] Pruning and Child Expansion Ordering

The number of nodes pruned using alpha-beta pruning depends on the order in which the nodes are expanded. For example, consider the following minimax tree.



In this tree, if the children of each node are expanded from left to right for each of the three nodes then no pruning is possible. However, if the expansion ordering were to be first Right then Left for node A, first Right then Left for node C, and first Left then Right for node B, then the leaf containing the value 4 can be pruned. (Similarly for first Right then Left for node A, first Left then Right for node C, and first Left then Right for node B.)

For the following tree, give an ordering of expansion for each of the nodes that will maximize the number of leaf nodes that are never visited due the search (thanks to pruning). ***For each node, draw an arrow indicating which child will be visited first. Cross out every leaf node that never gets visited.***
Hint: Your solution should have three leaf nodes crossed out and indicate the child ordering for 6 of the 7 internal nodes.



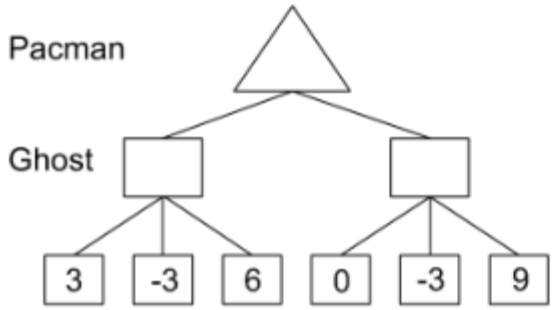
The thing to understand here is how pruning works conceptually. A node is pruned from under a max node if it "knows" that the min node above it has a better – smaller – value to pick than the value that the max node just found. Similarly, a node is pruned from under a min node if it knows that the max node above it has a better – larger – value to pick than the value that the min node just found.

שאלה 10

Q4. [15 pts] Variants of Trees

(a) Pacman is going to play against a careless ghost, which makes a move that is optimal for Pacman $\frac{1}{3}$ of the time, and makes a move that that minimizes Pacman's utility the other $\frac{2}{3}$ of the time.

(i) [2 pts] Fill in the correct utility values in the game tree below where Pacman is the maximizer:

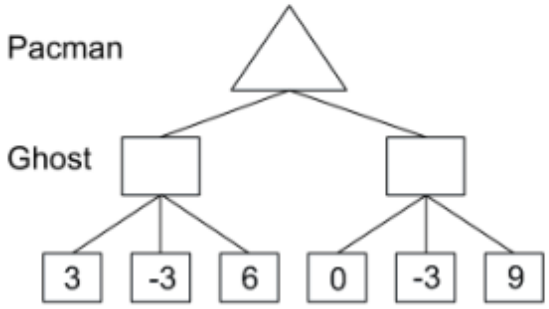


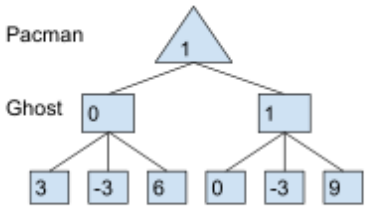
(ii) [2 pts] Draw a complete game tree for the game above that contains only max nodes, min nodes, and chance nodes.

Q4. [15 pts] Variants of Trees

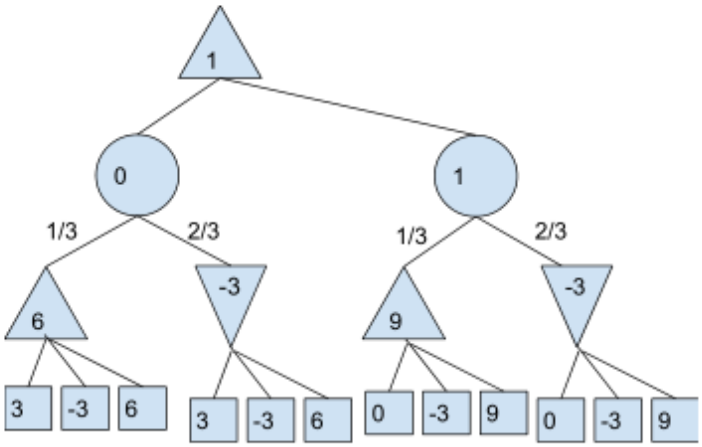
(a) Pacman is going to play against a careless ghost, which makes a move that is optimal for Pacman $\frac{1}{3}$ of the time, and makes a move that that minimizes Pacman's utility the other $\frac{2}{3}$ of the time.

(i) [2 pts] Fill in the correct utility values in the game tree below where Pacman is the maximizer:

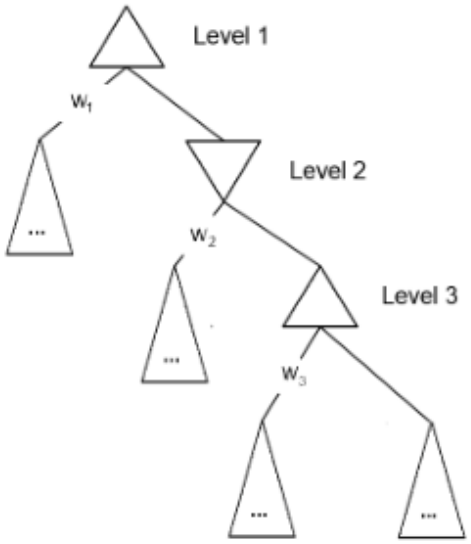




(ii) [2 pts] Draw a complete game tree for the game above that contains only max nodes, min nodes, and chance nodes.



(b) Consider a modification of alpha-beta pruning where, rather than keeping track of a single value for α and β, you instead keep a list containing the best value, w_i, for the minimizer/maximizer (depending on the level) at each level up to and including the current level. Assume that the root node is always a max node. For example, consider the following game tree in which the first 3 levels are shown. When the considering the right child of the node at level 3, you have access to w_1, w_2, and w_3.



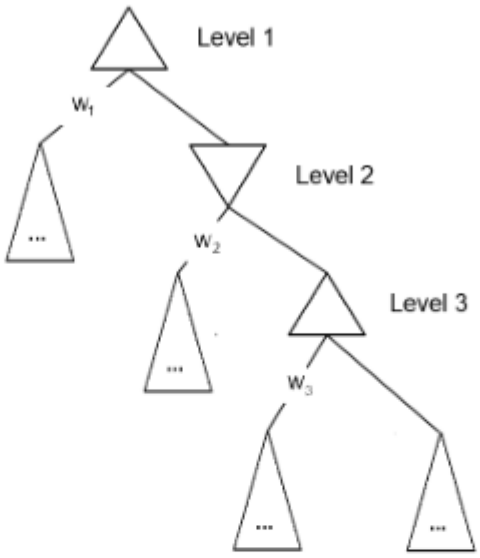
(i) [1 pt] Under this new scenario, what is the pruning condition for a max node at the n^{th} level of the tree (in terms of v and $w_1 ... w_n$)?

(ii) [1 pt] What is the pruning condition for a min node at the n^{th} level of the tree?

(iii) [2 pts] What is the relationship between α, β and the list of $w_1 ... w_n$ at a max node at the n^{th} level of the tree?

- ○ $\sum_i w_i = \alpha + \beta$
- ○ $\max_i w_i = \alpha, \quad \min_i w_i = \beta$
- ○ $\min_i w_i = \alpha, \quad \max_i w_i = \beta$
- ○ $w_n = \alpha, \quad w_{n-1} = \beta$
- ○ $w_{n-1} = \alpha, \quad w_n = \beta$
- ○ None of the above. The relationship is _____

(b) Consider a modification of alpha-beta pruning where, rather than keeping track of a single value for α and β, you instead keep a list containing the best value, w_i, for the minimizer/maximizer (depending on the level) at each level up to and including the current level. Assume that the root node is always a max node. For example, consider the following game tree in which the first 3 levels are shown. When the considering the right child of the node at level 3, you have access to w_1, w_2, and w_3.



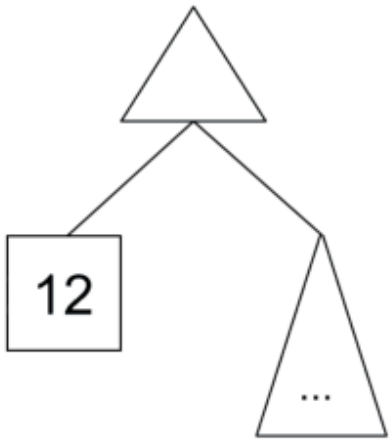
(i) [1 pt] Under this new scenario, what is the pruning condition for a max node at the n^{th} level of the tree (in terms of v and $w_1 ... w_n$)? $v > min_i(w_i)$, where i is even;

(ii) [1 pt] What is the pruning condition for a min node at the n^{th} level of the tree?
$v < max_i(w_i)$, where i is odd;

(iii) [2 pts] What is the relationship between α, β and the list of $w_1 ... w_n$ at a max node at the n^{th} level of the tree?

- ○ $\sum_i w_i = \alpha + \beta$
- ○ $\max_i w_i = \alpha, \quad \min_i w_i = \beta$
- ○ $\min_i w_i = \alpha, \quad \max_i w_i = \beta$
- ○ $w_n = \alpha, \quad w_{n-1} = \beta$
- ○ $w_{n-1} = \alpha, \quad w_n = \beta$
- ○ None of the above. The relationship is $\beta = min(w_2, w_4, w_6 ...), \quad \alpha = max(w_1, w_3, w_5 ...)$

(c) Pacman is in a dilemma. He is trying to maximize his overall utility in a game, which is modeled as the following game tree.



The left subtree contains a utility of 12. The right subtree contains an unknown utility value. An oracle has told you that the value of the right subtree is one of -3, -9, or 21. You know that each value is equally likely, but without exploring the subtree you do not know which one it is.

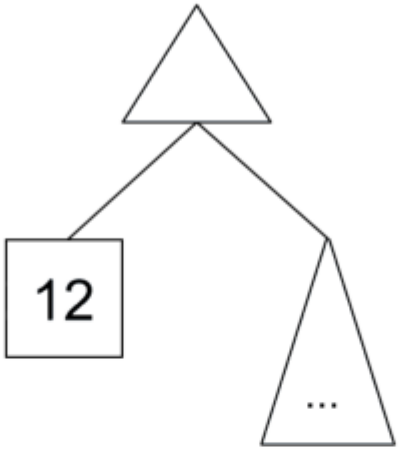
Now Pacman has 3 options:

1. Choose left;
2. Choose right;
3. Pay a cost of c = 1 to explore the right subtree, determine the exact utility it contains, and then make a decision.

(i) [3 pts] What is the expected utility for option 3?

(ii) [4 pts] For what values of c (for example, $c > 5$ or $-2 < c < 2$) should Pacman choose option 3? If option 3 is never optimal regardless of the value for c, write None.

(c) Pacman is in a dilemma. He is trying to maximize his overall utility in a game, which is modeled as the following game tree.



The left subtree contains a utility of 12. The right subtree contains an unknown utility value. An oracle has told you that the value of the right subtree is one of -3, -9, or 21. You know that each value is equally likely, but without exploring the subtree you do not know which one it is.

Now Pacman has 3 options:

1. Choose left;
2. Choose right;
3. Pay a cost of c $= 1$ to explore the right subtree, determine the exact utility it contains, and then make a decision.

(i) [3 pts] What is the expected utility for option 3? $12 * \frac{2}{3} + 21 * \frac{1}{3} - 1 = 14$

(ii) [4 pts] For what values of c (for example, $c > 5$ or $-2 < c < 2$) should Pacman choose option 3? If option 3 is never optimal regardless of the value for c, write None.
According to the previous part, we have $15 - c > 12$, then we will choose option 3.
Therefore, $c < 3$