CS 188
Fall 2019

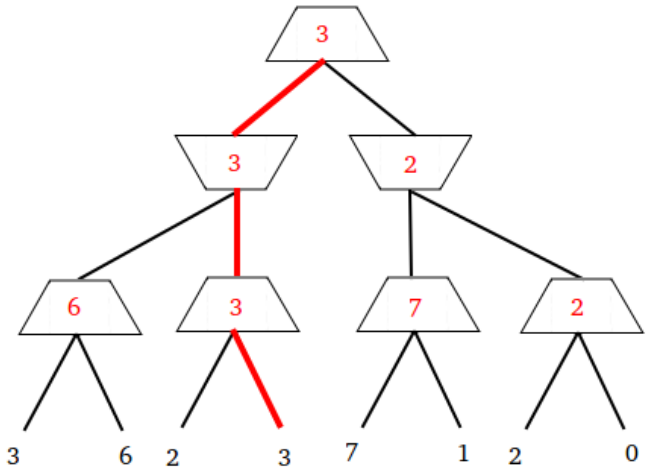
Exam Prep 3 Solutions

Q1. Games with Magic

(a) Standard Minimax

(i) 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) Mark the sequence of actions that correspond to Minimax play.



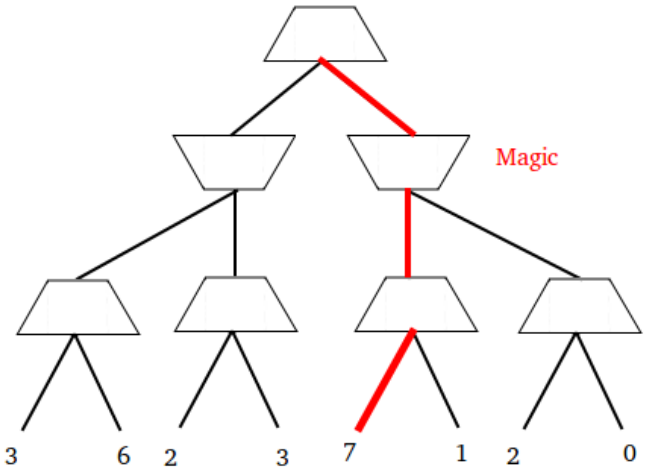
(b) Dark Magic

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) Dark Magic at Cost $c = 2$

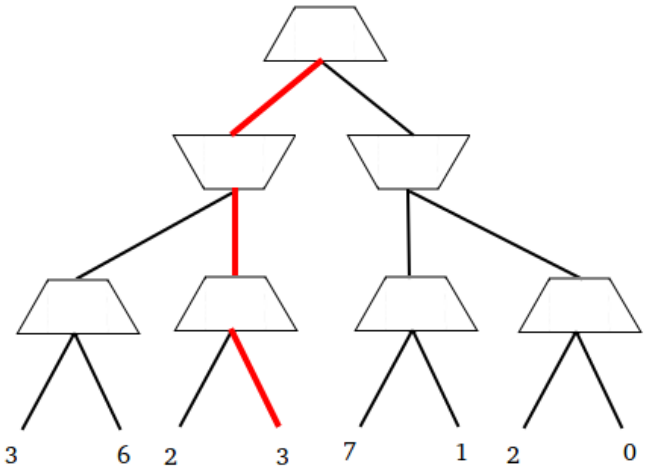
Consider the same game as before but now Pacman has access to his magic at cost $c = 2$. Is it optimal for Pacman to use his dark magic? If so, mark in the tree below where he will use it. Either way, mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



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) 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) 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), 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) 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.

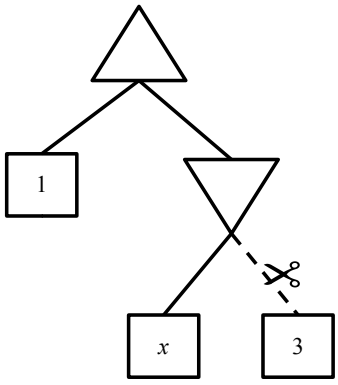
```
function MAX-VALUE(state)
    if state is leaf then
        return UTILITY(state)
    end if
    v ← −∞
    for successor in SUCCESSORS(state) do
        v ← max(v, MIN-VALUE(successor))
    end for
    return v
end function
```

```
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
```

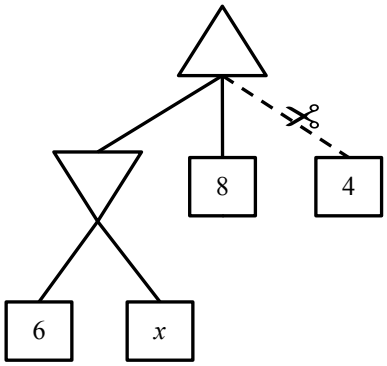
Q2. Games: Alpha-Beta Pruning

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".

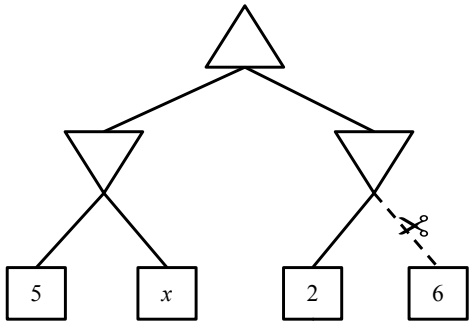


(a) Example Tree. Answer: $\mathbf{x \leq 1}$.

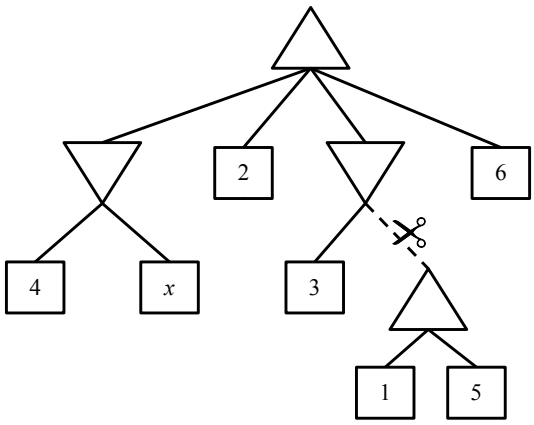
We are assuming that nodes are evaluated left to right and ties are broken in favor of the latter nodes. A different evaluation order would lead to different interval bounds, while a different tie breaking strategies could lead to strict inequalities ($>$ instead of $\geq$).
Successor enumeration order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



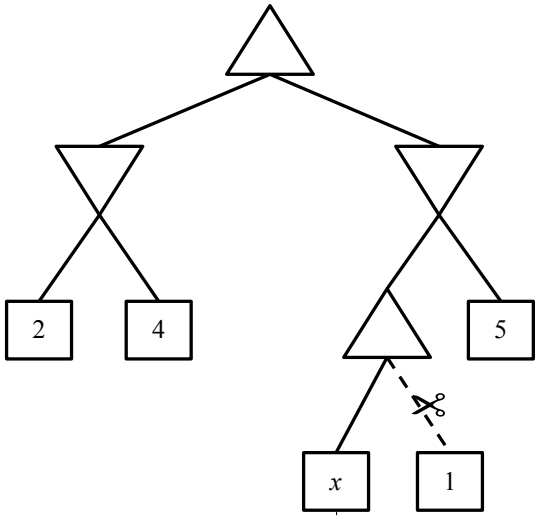
(b) Tree 1. Answer: None



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



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



(e) Tree 4. Answer: None