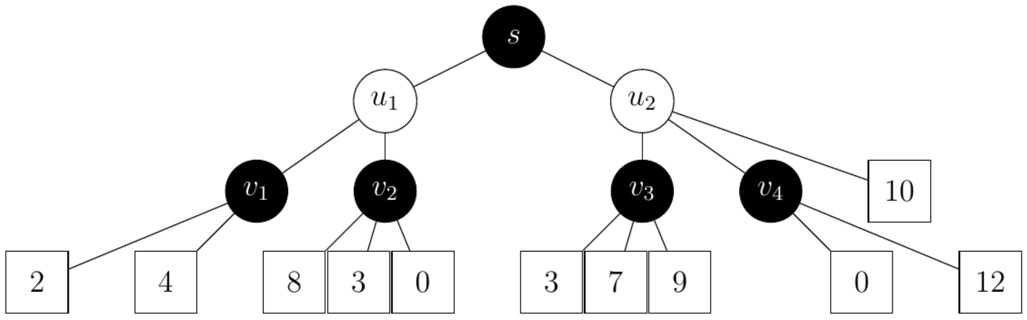
National University of Singapore
School of Computing
CS3243 Introduction to AI

Tutorial 6: Adversarial Search

SOLUTIONS

1. Consider the **Minimax** tree given in the figure below. In the figure, black nodes are controlled by the MAX player, and white nodes are controlled by the MIN player. Payments (terminal nodes) are squares, where the number within denotes the amount that the MIN player pays to the MAX player (an amount of 0 means that MIN pays nothing to MAX). Naturally, MAX wants to maximise the amount they receive, and MIN wants to minimise the amount they pay.



Suppose that we use the **α-β Pruning** algorithm, given in Figure 5.7 of AIMA 4th edition (reproduced in the **Appendix**).

(a) ***Assume that we iterate over nodes from right to left***; mark with an "**X**" all edges that are pruned by α-β pruning, if any.

(b) ***Assume that we iterate over nodes from left to right***; mark with an "**X**" all edges that are pruned by α-β pruning, if any.

(c) Determine if the effectiveness of pruning depends on iteration order (i.e., provide a **yes or no** answer. (There is no need to elaborate on this answer for your assignment submission However, you should consider why iteration order is important for discussion during the tutorial.)

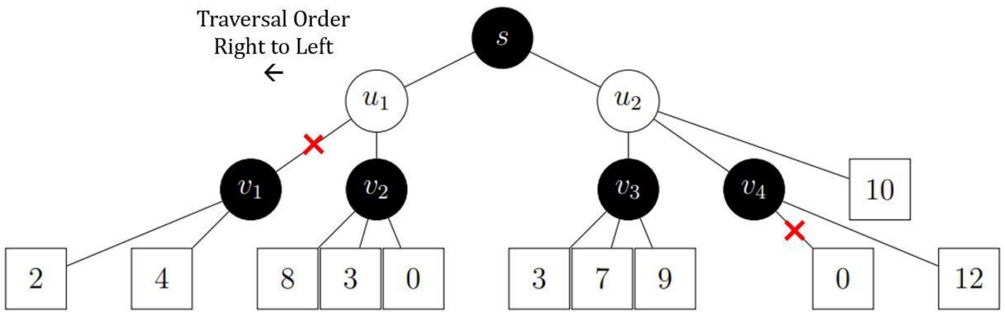
Solution: We refer to the players' actions as R (right), M (middle), and L (left). We let $\alpha(x)$ be the least payoff that MAX can guarantee at node x, and $\beta(x)$ be the maximal payoff MIN must pay at node x.

(a) Let us consider a run of the **α-β Pruning** algorithm here. First, the MAX player will explore R, followed by the MIN player exploring R. At this point, $\beta(u_2) = 10$.

Next, MIN plays M and explores v_4; MAX explores R and observes a value of 12 (so $\alpha(v_4) = 12$); since this value is greater than $\beta(u_2)$ we know that choosing M at u_2 yields a lower value than playing R, the edge between v_4 and 0 is pruned. Next, MIN explores L (the node v_3), seeing the values 9, 7 and 3 in this order; this yields $\alpha(v_3) = 9$, and we update $\beta(u_2) = 9$, and subsequently $\alpha(s) = 9$ (in other words, if MAX chooses R at s, they can get a payoff of at least 9).

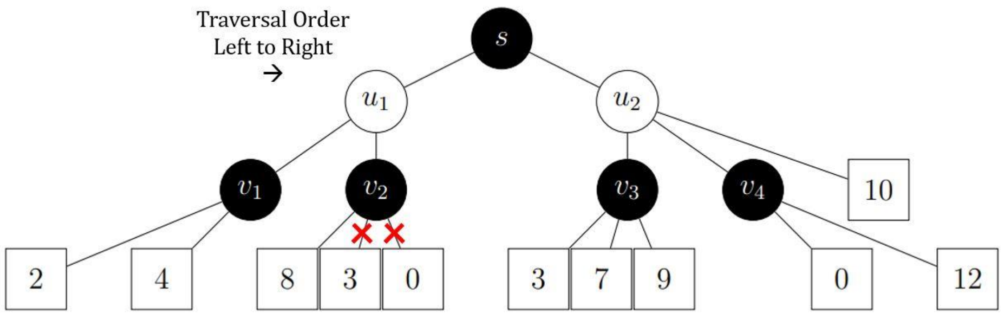
Next, MAX explores L at s, MIN explores R at u_1, and MAX explores R, followed by M and L, at v_2. This yields $\alpha(v_2) = 8$, which means that $\beta(u_1) = 8$, which is strictly smaller than $\alpha(s)$. What this immediately implies is that the action L is strictly worse for MAX than the action R, and there is no point in continuing to explore its subtree: if MAX chooses L, then MIN can choose R to obtain a better outcome (for MIN). At this point, there is no longer any need to explore the action *L* at u_1 (and its subtree) and the algorithm stops: the optimal action sequence is *R* at *s*, *L* at u_2 and *R* at v_3, with MIN paying 9 to MAX.

This is summarised by the following search tree.



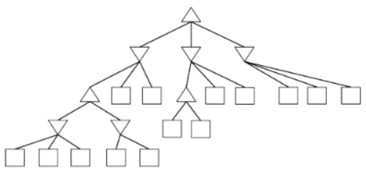
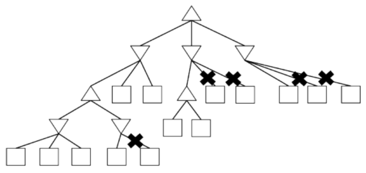
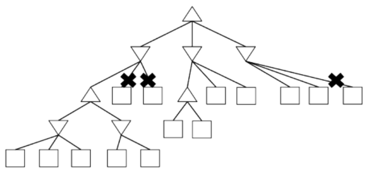
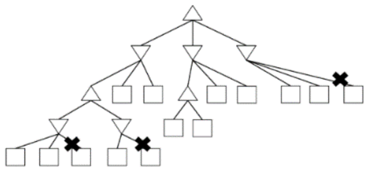
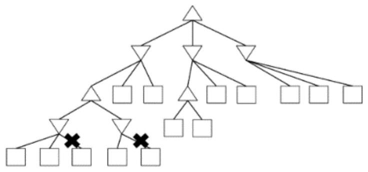
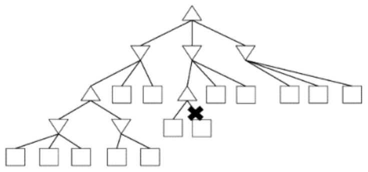
(b) Next, we assume that agents explore actions from left to right. The MAX player first explores (s, u_1), followed by the MIN player exploring (u_1, v_1), and the MAX player checking $(v_1, 2)$; then we check $(v_1, 4)$, updating $\alpha(v_1) = 4$ and in turn, $\beta(u_1) = 4$ (i.e., in the worst case (for MAX) the MIN player loses 2 here by playing *L*, assuming MAX does not play optimally). Then we check (u_1, v_2) and $(v_2, 8)$. At this point we stop since $\alpha(v_2) = 8$ but $\beta(u_1) = 4$. Consequently, we prune both $(v_2, 3)$ and $(v_2, 0)$. Onto checking (s, u_2): here, we next check (u_2, v_3) and $(v_3, 3)$, $(v_3, 7)$ and $(v_3, 9)$, yielding $\alpha(v_3) = \beta(u_2) = 9$. Choosing (u_2, v_4) next, we check both $(v_4, 0)$ and $(v_4, 12)$ to get $\alpha(v_4) = 12$, while $\beta(u_2)$ remains at 9. Finally, we check $(u_2, 10)$ and nothing changes.

This is summarised by the following search tree.



(c) As we can clearly see from the solutions above, order of traversal makes a difference.

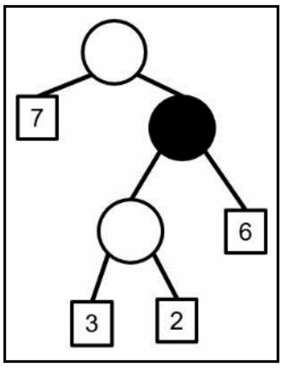
2. Assume that α-β (alpha-beta) pruning is applied to each of the following game trees in a left-to-right exploration. Further, assume zero-sum games. Select the options where the given pruning **is possible** under some choice of pay-off values. Note that the crossed-off edges indicate where pruning has occurred.

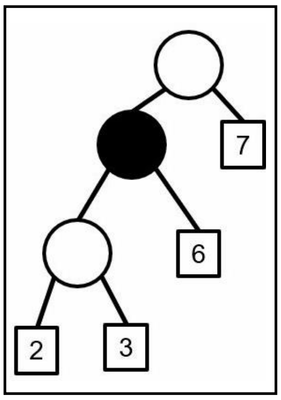
Option **A**: 	Option **B**: 	Option **C**: 
Option **D**: 	Option **E**: 	Option **F**: 

Option **G**: None of the above.

Solution: Options **A** and **B**.

3. A game tree is ***best-score-first*** ordered if for all non-terminal nodes, its first successor represents the best possible move for that position. The tree on the right is an example of a game tree that is ***best-score-first*** ordered. Notice that at the first node (which is a MAX node), the first action is best as it leads to a utility score of 7 while the second action leads to a utility score of 3. In the second node (which is a MIN node), the first action is best as it leads to a utility score of 3 while the second action leads to a utility score of 6. In the third node (which is a MAX node), the first action is once again best. Hence, the above tree is ***best-score-first*** ordered.



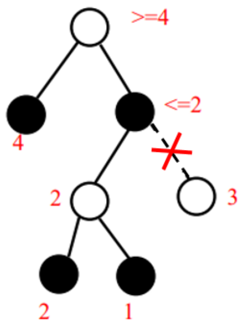


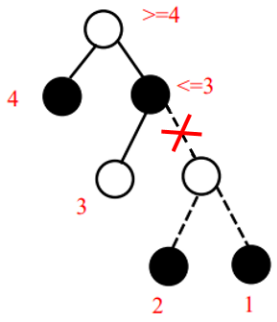
Note that a different agent can use a different move-ordering heuristic to generate a different game tree for the same game. For example, the game tree on the left might be generated instead. This is a permutation of the first tree. Note that there are many possible permutations for a given game tree, all of which corresponds to a slightly different move-ordering heuristic being used. Notice how in the first tree (which is ***best-score-first*** ordered), one branch is pruned by the α-β Pruning algorithm. However, in the second tree (which is a permutation, but not ***best-score-first*** ordered), no branch is pruned by the α-β Pruning algorithm. Hence, fewer nodes are visited with the first tree than the second tree.

Prove or disprove the following statement.

Let ***T*** be a ***best-score-first*** ordered game tree. Suppose ***T'*** is a permutation of ***T*** such that it is <u>not</u> ***best-score-first*** ordered. If the α-β Pruning algorithm visits N nodes given ***T***, then it will visit $M \geq N$ nodes given ***T'***.

Solution: False. The following is a counterexample. The tree on the left is ***best-score-first*** ordered, but the tree on the right, which is not ***best-score-first*** ordered, has less nodes explored.

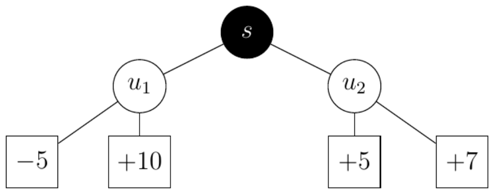




4. With the **Minimax** algorithm, we know that the value *v*, computed at the root (i.e., the utility for the MAX player), is a worst-case value. This means that if the opponent MIN does not act optimally, the actual outcome *v*' for MAX can only be better, and never worse than *v*. That said, the **Minimax** algorithm may not select the optimal move given sub-optimal play from the MIN player.

   Construct an example where, should the MIN player play sub-optimally, the **Minimax** algorithm makes a sub-optimal move.

   Solution: An example is depicted below. Based on the **Minimax** algorithm, MAX plays $s \rightarrow u_2$. However, assuming MIN plays sub-optimally, $s \rightarrow u_1$ should instead be picked for the +10 payout.



   The above is an example game tree when MAX may exploit sub-optimal play.

Appendix

The following is a description of the **α-β Pruning** algorithm. It is referenced from the AIMA 4th edition textbook (Figure 5.7).

```
function ALPHA-BETA-SEARCH(game, state) returns an action
  player ← game.TO-MOVE(state)
  value, move ← MAX-VALUE(game, state, $-\infty$, $+\infty$)
  return move

function MAX-VALUE(game, state, $\alpha$, $\beta$) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  $v \leftarrow -\infty$
  for each a in game.ACTIONS(state) do
    v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), $\alpha$, $\beta$)
    if $v2 > v$ then
      v, move ← v2, a
      $\alpha \leftarrow$ MAX($\alpha$, v)
    if $v \geq \beta$ then return v, move
  return v, move

function MIN-VALUE(game, state, $\alpha$, $\beta$) returns a (utility, move) pair
  if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
  $v \leftarrow +\infty$
  for each a in game.ACTIONS(state) do
    v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), $\alpha$, $\beta$)
    if $v2 < v$ then
      v, move ← v2, a
      $\beta \leftarrow$ MIN($\beta$, v)
    if $v \leq \alpha$ then return v, move
  return v, move
```