

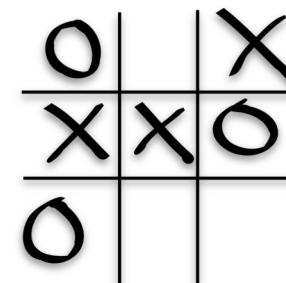


5. Adversarial Search

5.1 Adversarial search

- Search in competitive environments
 - where agents' goals are in conflict
- Games are good examples of adversarial search
 - States are easy to represent (unlike many other real world problems)
 - Agents are restricted to a small number of actions
 - Outcome of agent actions defined by precise rules
 - Usually too hard to solve

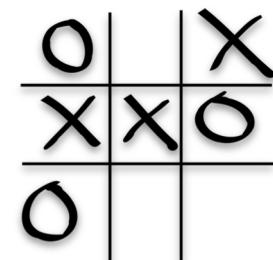
7	2	4
5		6
8	3	1



Which one/s are adversarial search?

5.1 How to formally define a game?

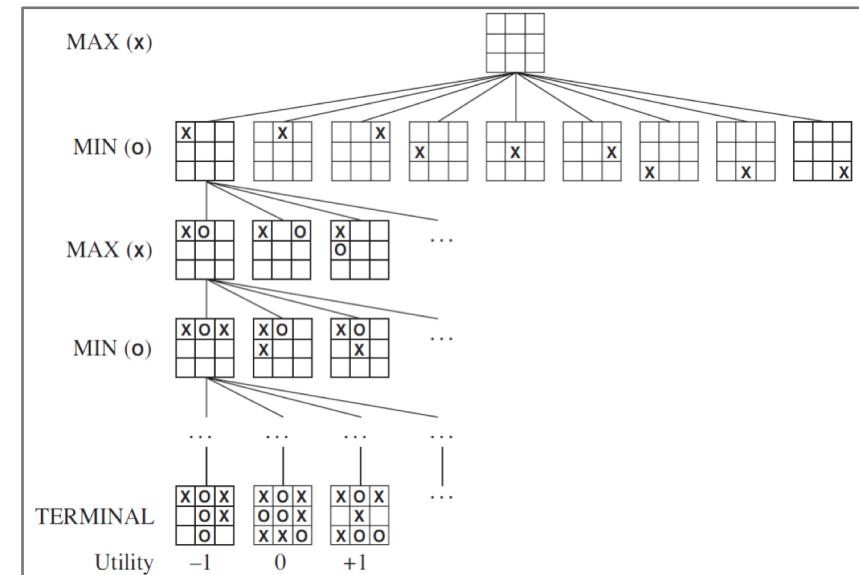
- Consider a game with two players MAX and MIN
 - MAX moves first (places X) followed by MIN
- A game can be formally defined as a search problem with the following elements:
 - S_0 - The initial state
 - Player(s) - Defines which player has the move in a state (s)
 - Actions(s) - Returns the set of legal moves in a state
 - Results (s, a) - The transition model which defines the result of a move
 - Terminal-Test (s) - True when game is over, false otherwise
 - Utility (s, p)
 - A utility function (also objective function or payoff function)
 - Defines the final numeric value **for a game that ends in a terminal state s** for player p
 - For example, in tic-tac-toe, the outcome is win, loss, or draw with values -1, 0, or 1
- The Initial State, Actions function, and Result function define game tree for a game
 - Game tree - a tree where nodes are game states and edges are moves



5.1 Game tree for tic-tac-toe

- MAX has 9 possible moves from the initial state
- Play alternates between MAX's placing an X and MIN's placing an O
- The number **in each leaf node** indicates the utility value of the terminal state **from the point of view of MAX**
 - High values are assumed to be good for MAX and bad for MIN

of nodes are in this game tree?



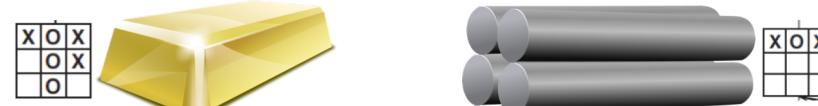
5.2 Optimal decisions in games

A normal search problem:

- The optimal solution is a sequence of actions leading to a goal state

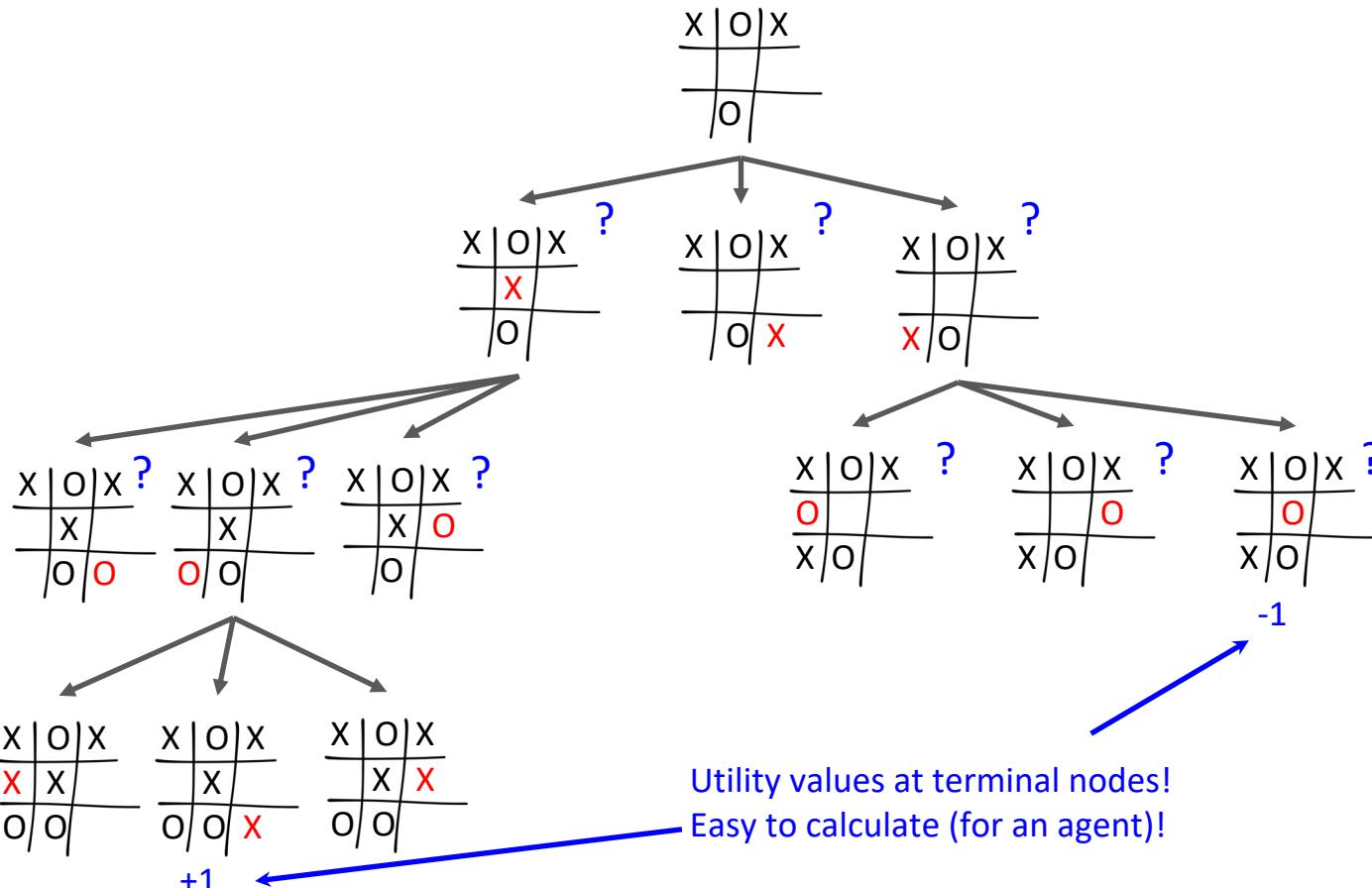
An adversarial search problem:

- ‘MIN’ interferes the sequence of actions
- Strategy for ‘MAX’:
 - Specify moves in the initial state
 - Observe every possible response by MIN
 - Specify moves in response
 - and so on..
- This optimal strategy can be determined from the **minimax value** of each node
 - This number tells you if a state is ‘gold’ or ‘brass’



How to calculate utility of non-terminal nodes?

MAX's turn



5.2 How to calculate minimax value?

Consider a ‘reduced’ tic-tac-toe game (because game tree for full tic-tac-toe is too big)

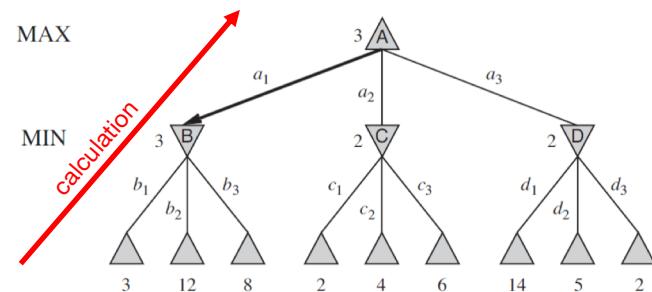
- The possible moves for MAX at the root are a_1 , a_2 , and a_3
- Possible replies to a_1 for MIN are b_1 , b_2 , b_3 , and so on

Optimal strategy can be determined using minimax value of each node $\text{MINIMAX}(n)$

- $\text{MINIMAX}(n)$ for the user MAX is the utility of being in the corresponding state
- So, MAX will always prefer to move to a state of maximum value

$\text{MINIMAX}(s) =$

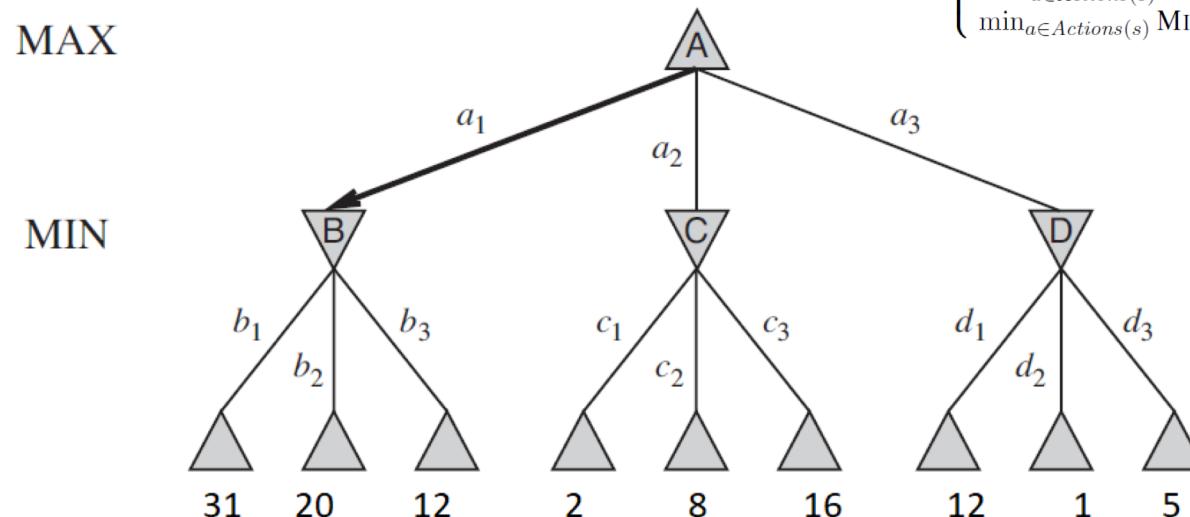
$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$



5.2 How to calculate minimax value?

Practice Problem: Calculate the minimax values at nodes A, B, C, and D for the player MAX given the game tree below. The numbers at the leaf nodes represent the values of Utility (leafnode, MAX).

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$



Utility function vs Minimax function

Utility (s, p)

- defines the final numeric value for a game that ends in a terminal state s for player p

Minimax (s, p)

- defines the numeric values at all other nodes

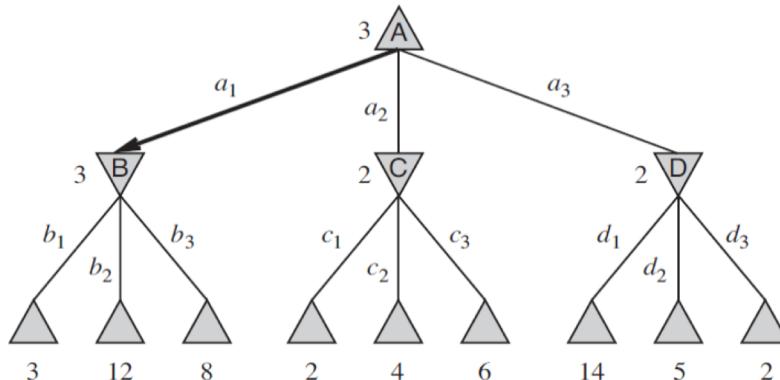
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

5.2.1 The minimax algorithm

```
MINIMAX( $s$ ) =  
   $\begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$ 
```

MAX

MIN



The recursion proceeds all the way down to the leaves of the tree, and then the **minimax values are backed up through the tree** as the recursion unwinds.

The minimax algorithm performs a **complete depth-first exploration of the game tree**.

If the maximum depth of the tree is ' m ' and there are ' b ' legal moves at each point, time Complexity = $O(b^m)$.

```
function MINIMAX-DECISION( $state$ ) returns an action  
  return  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(state, a))$ 
```

```
function MAX-VALUE( $state$ ) returns a utility value  
  if TERMINAL-TEST( $state$ ) then return UTILITY( $state$ )  
   $v \leftarrow -\infty$   
  for each  $a$  in ACTIONS( $state$ ) do  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$   
  return  $v$ 
```

```
function MIN-VALUE( $state$ ) returns a utility value  
  if TERMINAL-TEST( $state$ ) then return UTILITY( $state$ )  
   $v \leftarrow \infty$   
  for each  $a$  in ACTIONS( $state$ ) do  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$   
  return  $v$ 
```

5.3 Do we need to compute all ‘MINIMAX’ values?

Let the two unevaluated successors of C have values x and y.

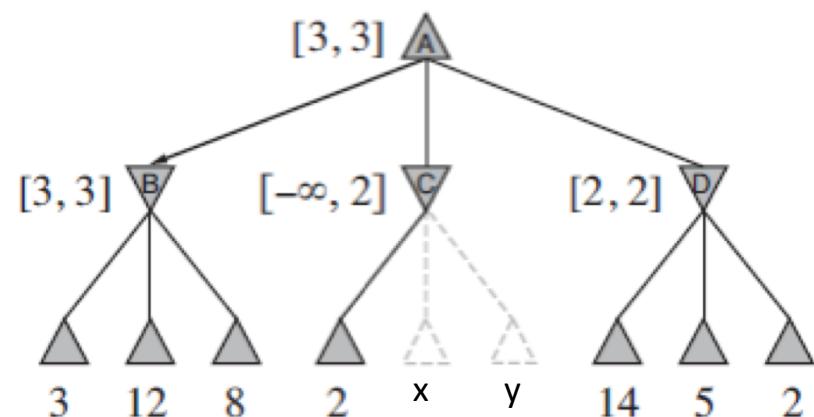
Calculate the MINIMAX(root) or node A (in terms of x and y):

$$= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2))$$

=

=

=



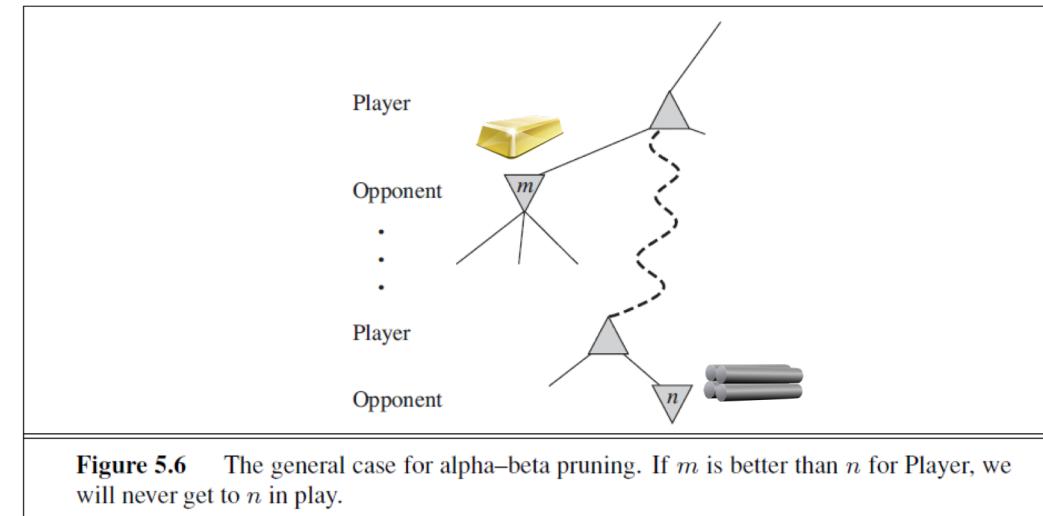
Is the minimax decision dependent on the values of x and y?

5.3 Which nodes to prune?

Principle for pruning:

- Consider a node ‘n’ somewhere in the tree, such that the “player” has a choice of moving to that node
- If the “player” has a better choice ‘m’ either at the parent of node ‘n’ or at any choice point further up, then ‘n’ will never be reached in actual play

Once we know enough about ‘n’,
to reach such a conclusion,
we can prune it.



5.3 The Alpha-beta pruning algorithm

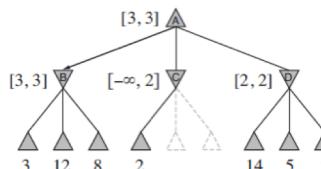
```
function MINIMAX-DECISION(state) returns an action
    return arg maxa ∈ ACTIONS(s) MIN-VALUE(RESULT(state, a))
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, a)))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, a)))
    return v
```

Minimax search is depth-first, so we consider the nodes along a single path in the search tree.
 α = the value of the **best node** (highest value) we have found so far at any choice point along the path **for MAX**
 β = the value of the **best node** (lowest value) we have found so far at any choice point along the path **for MIN**

Principle: Alpha-beta search updates the values of α and β as it goes along and prunes the remaining branches at a node (i.e. terminates the recursive call) as soon as the value of the current node is known to be worse than the current value of α and β for MAX and MIN respectively.



```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, a),  $\alpha$ ,  $\beta$ ))
        if  $v \geq \beta$  then return  $v$ 
         $\alpha$  ← MAX( $\alpha$ ,  $v$ )
    return  $v$ 
```

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, a),  $\alpha$ ,  $\beta$ ))
        if  $v \leq \alpha$  then return  $v$ 
         $\beta$  ← MIN( $\beta$ ,  $v$ )
    return  $v$ 
```

5.3 The Alpha-beta pruning algorithm

```
1 function ALPHA-BETA-SEARCH(state) returns an action
2    $v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)$ 
3   return the action in ACTIONS(state) with value v


---


1 function MAX-VALUE(state,  $\alpha, \beta$ ) returns a utility value
2   if TERMINAL-TEST(state) then return UTILITY(state)
3    $v \leftarrow -\infty$ 
4   for each a in ACTIONS(state) do
5      $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
6     if  $v \geq \beta$  then return v
7      $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
8   return v


---


1 function MIN-VALUE(state,  $\alpha, \beta$ ) returns a utility value
2   if TERMINAL-TEST(state) then return UTILITY(state)
3    $v \leftarrow +\infty$ 
4   for each a in ACTIONS(state) do
5      $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
6     if  $v \leq \alpha$  then return v
7      $\beta \leftarrow \text{MIN}(\beta, v)$ 
8   return v
```

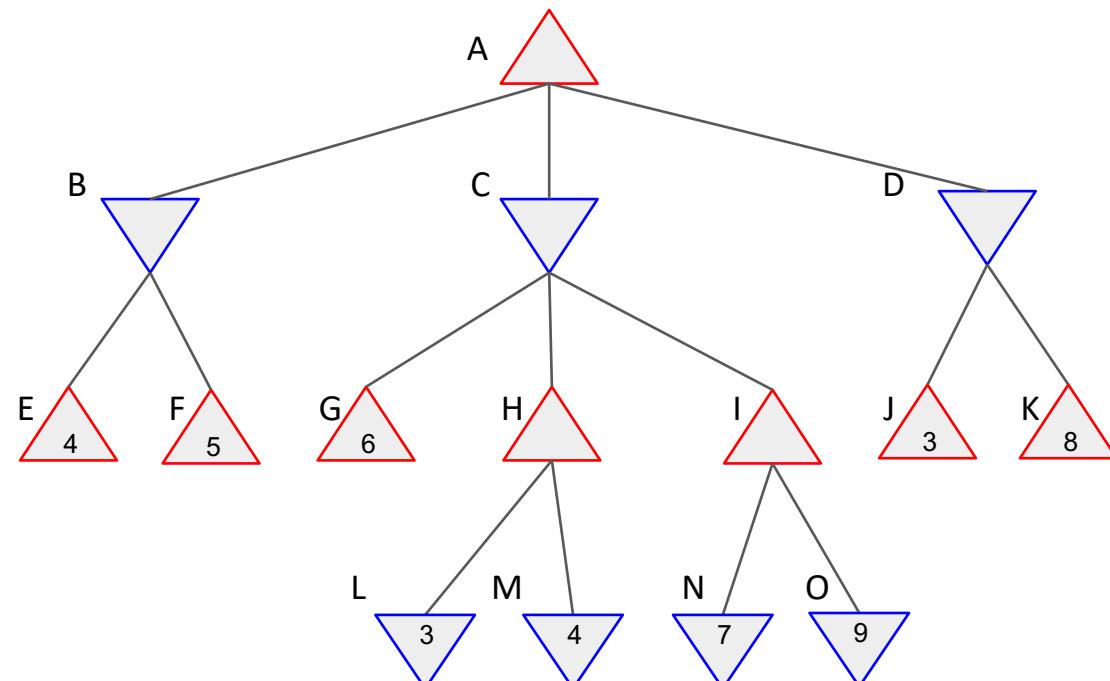
α = the value of the best node (highest value) we have found so far at any choice point along the path for MAX

β = the value of the best node (lowest value) we have found so far at any choice point along the path for MIN

Tips:

- (1) Maintain alpha, beta, and *v* at each non-leaf node
- (2) Update '*v*' step-by-step

Problem: Calculate the minimax values for the non-leaf nodes in the search tree below. Also, identify the nodes (or sub-tree) that will be pruned. Assume: 1) that MAX plays first at node A followed by MIN, and 2) actions are taken in alphabetical order.



Algorithm will be provided in tests!

Summary

- A game can be defined by the **initial state** (how the board is set up), the **legal actions in each state**, the **result of each action**, a **terminal test** (which says when the game is over), and a **utility function** that applies to terminal states.
- In two-player zero-sum games with perfect information, the minimax algorithm can select optimal moves by a depth-first enumeration of the game tree.
- The alpha–beta search algorithm **computes the same optimal move as minimax**, but achieves much greater efficiency by eliminating subtrees that are **provably irrelevant**.
- Practice more at
 - http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

Solution

