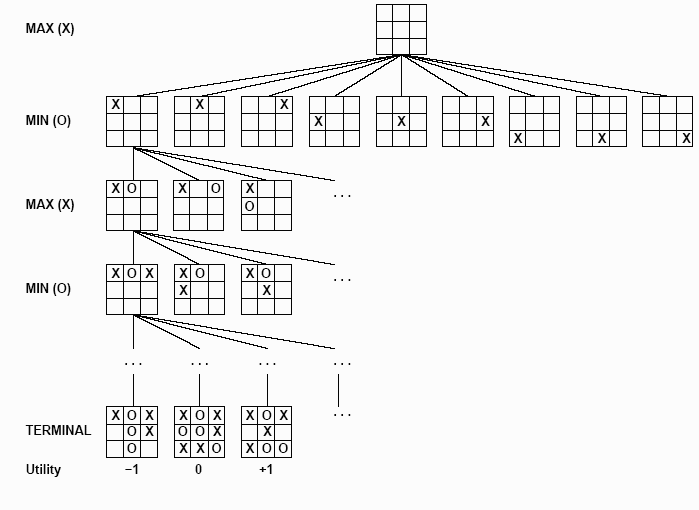
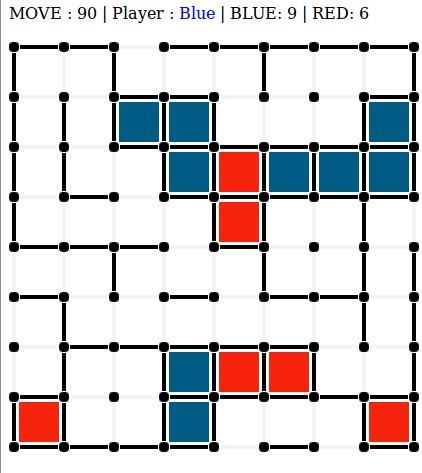
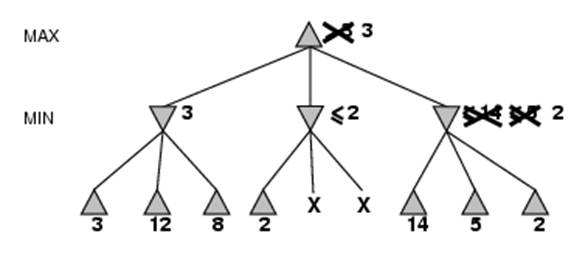
**Week 5 Topic Outline – Adversarial Search**

1. Overview of adversarial search
   1. Deterministic
   2. Fully Observable
   3. Two agents act alternately
   4. Utility values at the end of the game are always equal and opposite (win/lose)
   5. Focus is on games that meet these criteria
      1. Chess
      2. Checkers
      3. Tic-Tac-Toe
      4. Connect Four
      5. That line-drawing, connect-the-dots game…whatever it’s called
   6. What about these games?
      1. Chutes and Ladders
      2. Monopoly
2. Some terms:
   1. **Transition model** – list of states that would result from an action
   2. **Terminal Test/State/Node** – A terminal test checks to see if the game has ended. A terminal state is any state that causes the game to end. A terminal node is a node that is a part of a terminal state.
   3. **Utility Function** – defines the final numeric value for a game that ends in a terminal state. Sometimes this is zero-sum (-1, 1, ½), and sometimes it has a greater range of values.
   4. **Ply** – If you were to break game states down into a tree from separate actions, each level of the tree would be called a ply.
3. **Strategerie** – don’t misunderestimate your moves!
   1. Given a game tree, the optimal strategy can be determined from the minimax value of each node.
   2. **Minimax value** – the utility of being in a given state (numeric value of how good a state is)
      1. Moving to node ABC may give player A 10 points and player B 15 points, but moving to node XYZ may only give player A 8 points, but limits player B to 3 points.
      2. Minimax(ABC) = -5 (10 + -15)
      3. Minimax(XYZ) = 5 (8 + -3)
   3. Look at some examples:



1. **The minimax algorithm**



**3**

**2**

**2**

v

v

**6**

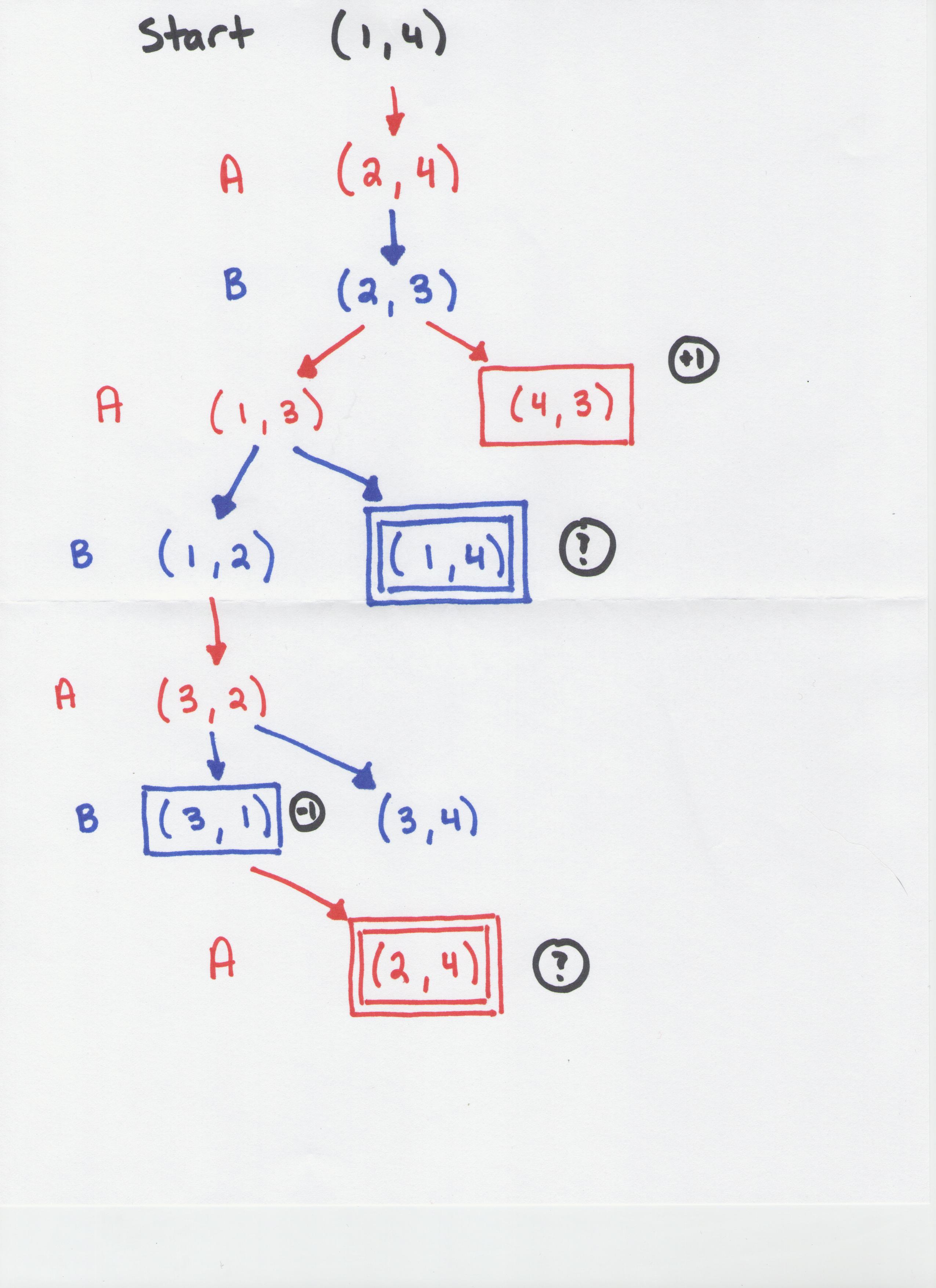
**4**

* 1. MAX needs to make a move that will maximize his score and minimize his opponent’s (MIN), but once he makes a move, MIN will make a move that will attempt to minimize his opponent’s (MAX).
  2. Computes the minimax decision from the current states
  3. It uses a simple recursive computation of the minimax values of each successor state.
  4. 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.
  5. Essentially, the minimax algorithm performs a complete depth-first exploration of the game tree.

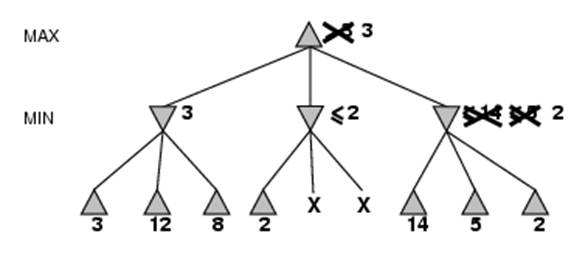
1. **Independent practice**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

1 2 3 4

****

1. **Break**
2. **Alpha-Beta Pruning**
   1. The main problem with minimax is that the number of game states that it has to evaluate is exponential in the depth of the tree
   2. We can eliminate large parts of the tree that we know will lead us down paths that won’t influence the final decision



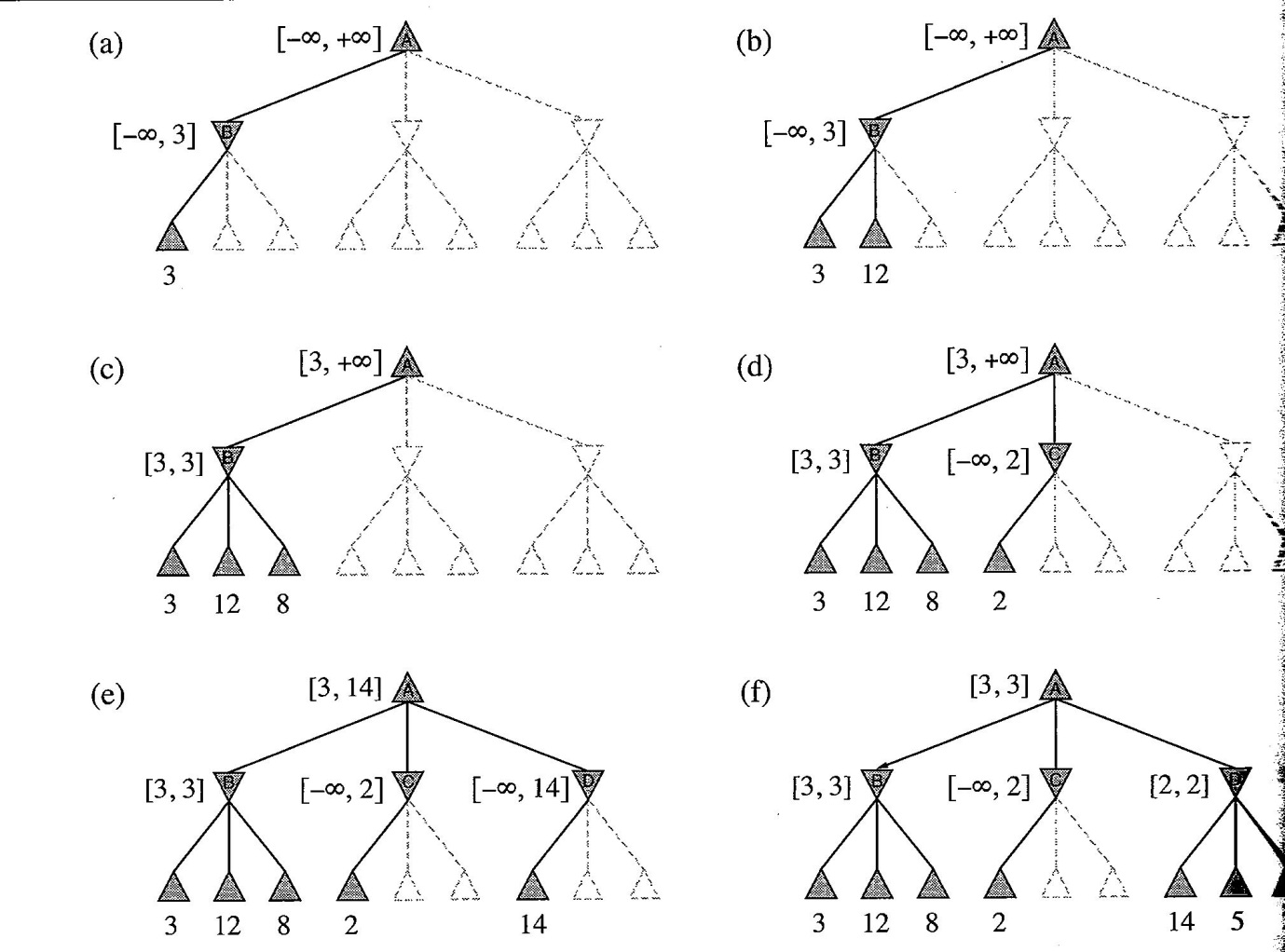
* 1. Minimax(root) = max(min(3, 12, 8), min(2, x, y), min(14, 5, 2))

= max( 3, min(2, x, y), 2)

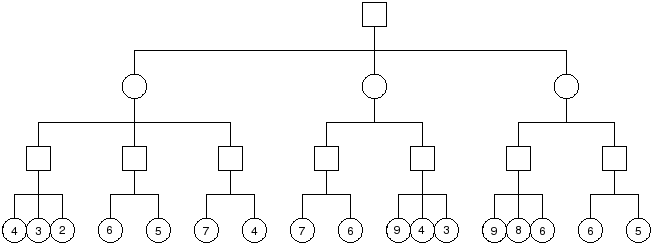
= max(3, z, 2) where z = min(2, x, y) <= 2

= 3

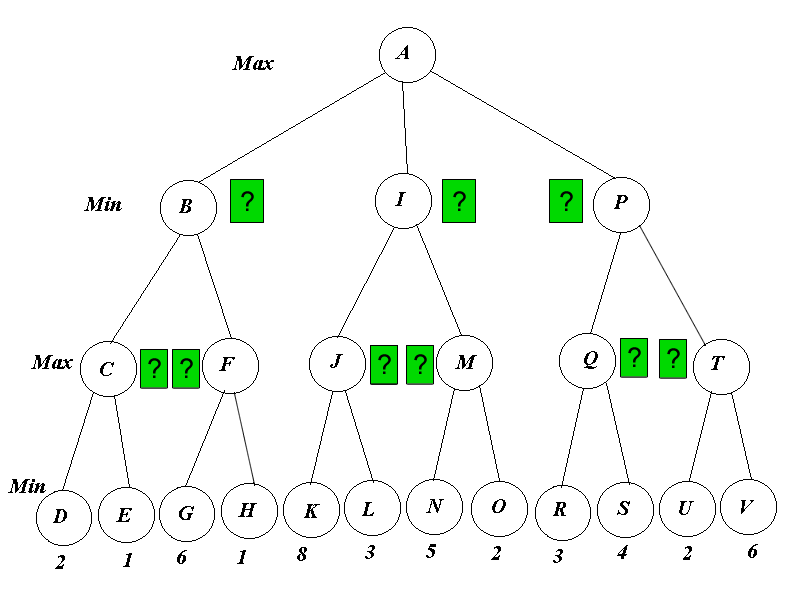
* 1. Walk through how this is evaluated



* 1. Explanation of the steps:
     1. (a) – The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of at most 3.
     2. (b) – The second leaf below B has a value of 12; MIN would avoid this move, so that value of B is still at most 3.
     3. (c) – The third leaf below B has a value of 8; we have seen all B’s successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is at least 3, because MAX has a choice worth 3 at the root.
     4. (d) – The first leaf below C has the value of 2. Hence, C, which is a MIN node, has a value of at most 2. But we know that B is worth 3, so MAX would never choose C. There, there is no point in looking at the other successor states of C.
     5. (e) – The first leaf below D has the value 14, so D is worth at most 14. This is still higher than MAX’s best alternative (3), so we need to keep exploring D’s successor states.
     6. (f) – The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX’s decision at the root is to move to B, giving a value of 3.
  2. Alpha-Beta pruning gets its name from the two parameters that describe bounds on the backed-up values that appear along the path [a, b].
     1. A = the value of the best choice we have found so far at any choice point along the path for MAX
     2. B = the value of the best choice we have found so far at any choice point along the path for MIN
  3. Let’s do one together:



* 1. Try one on your own (good test question)
     1. *Write the backed up values for each of the green rectangles (if they aren’t pruned)*
     2. *Cross out the branches that should be pruned.*
     3. *Draw the final path using arrows that should be followed*



*Solution:*

