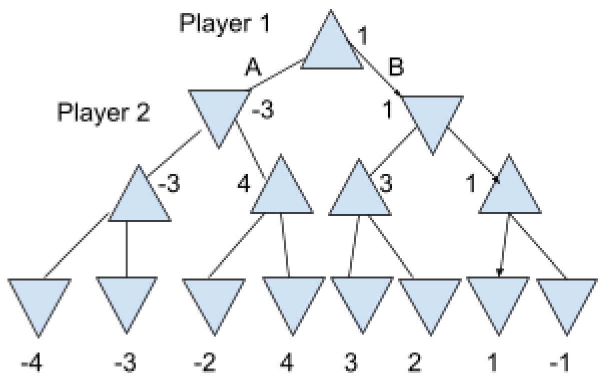
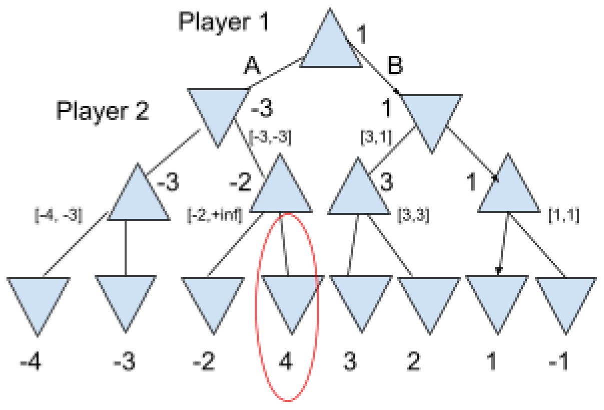
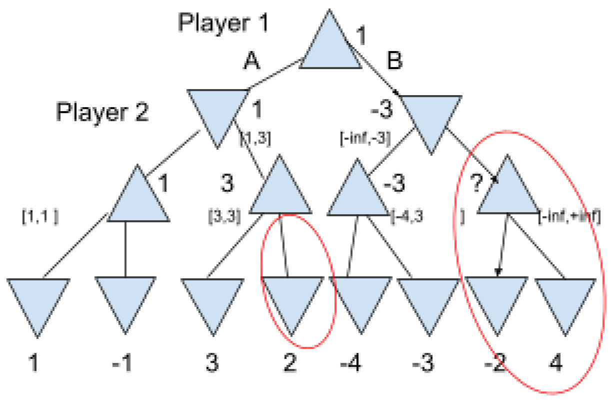


1.
   - Minimax values shown in graph.
   - Player 1 (the root node) should take path B, as it is a MAX node, and thus will choose the maximum value between of both options -3 and 1.
   - Expected outcome/payoff: +1
   - Alpha-beta pruning



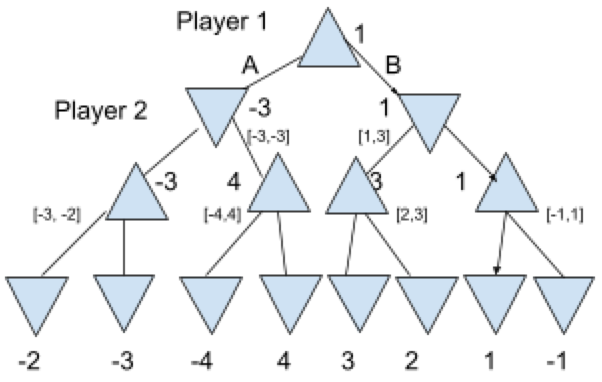
In this step, I only pruned one branch, as it was not a given that any of the others would not be necessary for the final calculation.

   - Maximize pruning:



This method maximizes pruning and prunes 4 total routes; when alpha-beta pruning travels down the B route and finds that the that one of the children, chosen by MAX when evaluating the leaf nodes, is -3 and returns to the parent node of route B, it immediately knows that the algorithm will *never* travel this route. This is because -3 is already < 1, the value of route A, and since the root (Player 1) is a MAX node, it will pick the route with the highest value. And, since Player 2 is a MIN and is trying to minimize the score, route B must be AT MOST -3–so there is no world where Player 2 will find a value higher than -3 and pick it. Therefore, we can prune the entire right side of route B.

- Minimize pruning



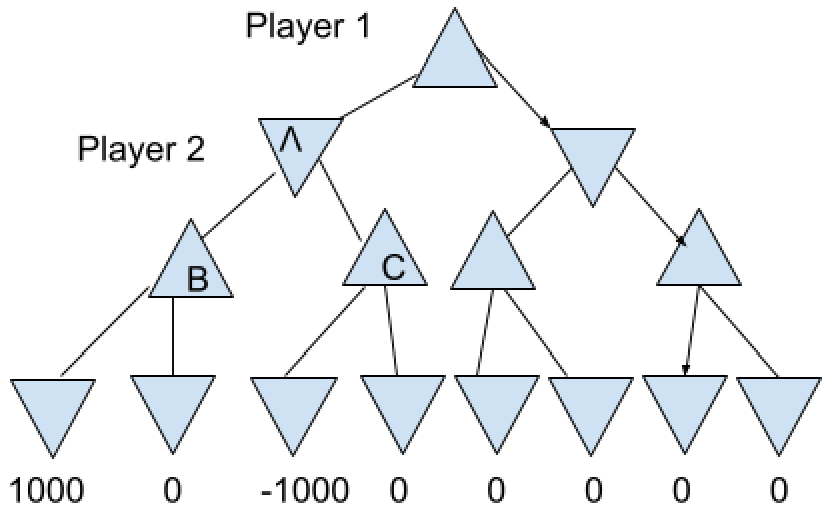
B

By switching -2 with -4 at the child nodes of route A, we ensure that the A-B pruning algorithm cannot skip the route to the child node with the value 4, and therefore eliminate all possible pruning.

2. Forcing a win:

- In a simple binary game tree of depth 4 where each player gets 2 moves and all the leaves have a utility of 0 except for one winning state (+1000) and one losing state (-1000), the player at the root *cannot force a win*. This is because the game tree is small enough that both players can see all possible moves and outcomes; the optimal strategy for the second player is to always choose the path that leads to the losing state for the first player (MIN nodes and MAX nodes).

- The specific location of the two non-zero states in the tree does not matter in this case. Either way, if states are far apart or close, the second player can always make adversarial decisions to ensure that the first player ends up in the losing state or an insignificant state (stalemate). It may seem that if the root player forces their opponent to the node before the winning state, they could force their hand, locking them into a win. However, after the root player's every turn, the opponent can *always* lead the root player towards a losing or insignificant state. Take this tree, for example:



It is in the best interest of both players to head to the left to node ^. At node ^, the opponent *should* lead the root player to node C in an optimal case for the opponent, which would ensure that the root player could not win–the tree's complexity is low enough that the opponent will be able to predict all 4 outcomes at this point in the game. At node C, there is nothing the root player can do to win, and it would be in his best interest to go to the 0 state and call it a draw. Yes, if the opponent picked node B, the root player could pick 1000, the winning state, and have won the game. But there is no way for the root to create a "checkmate" situation with both players playing optimally.

Note that, if the -1000 were placed directly adjacent to 1000, though the root player could force a win for him and a loss for the opponent if node B is traveled to, the opponent could still choose to force a draw by selecting node C a.

- If the depth of the game tree were changed to a different value, the answers to the questions could indeed change: if the depth of the game tree is increased significantly, so too would the complexity, and the first player may have a chance to force a win if they play optimally and the two non-zero states are placed in a favorable position within the tree. Basically, with deeper trees, there may be more room for strategic play and opportunities to create winning positions. The location of the non-zero states in the tree can become more critical as the depth of the tree increases, too; in deeper trees, the second player may not be able to see all possible outcomes, and the position of the non-zero states may affect the strategy and chances of winning for the first player. However, this would also complicate things for the root player, so, while theoretically possible, it would still be a difficult task.

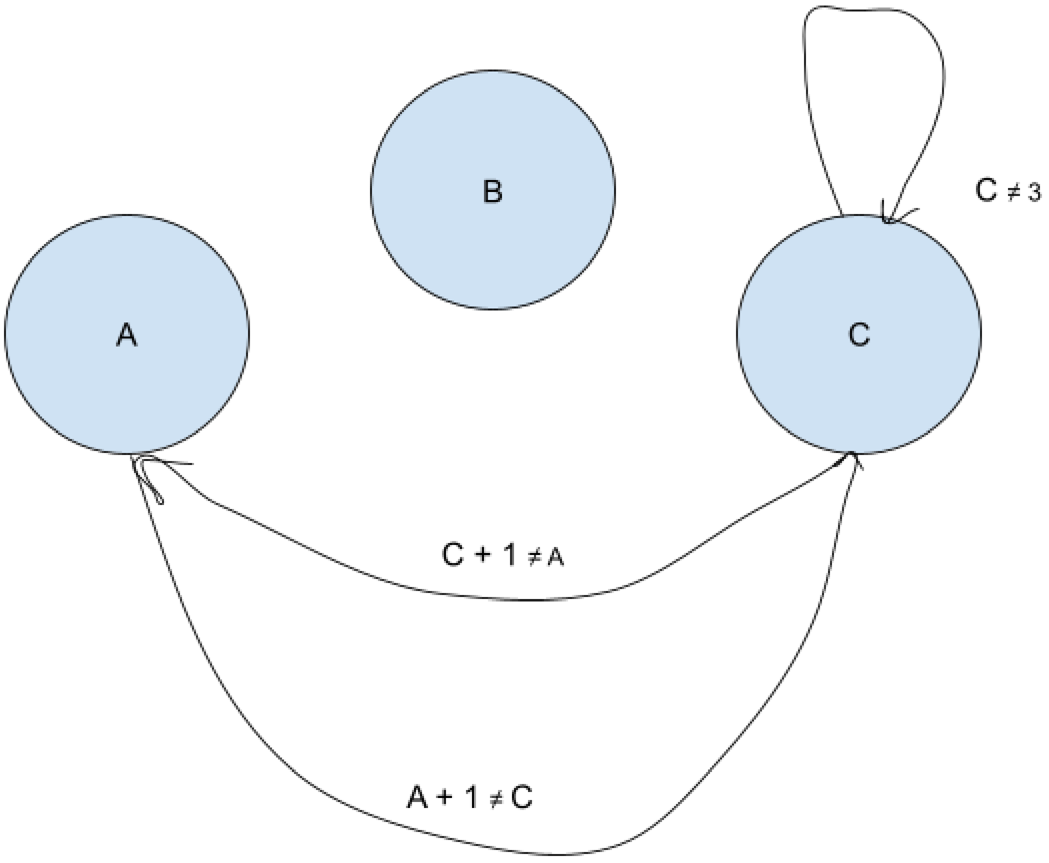
3. Constraint Problem
    - To set this up as a Constraint Satisfaction Problem, we must define:

i. The variables: these will represent the order in which the philosophers will hike. In this case, there are 3 variables, A, B, and C, representing Alex, Bob, and Charlie's positions, respectively. X = {A, B, C}.

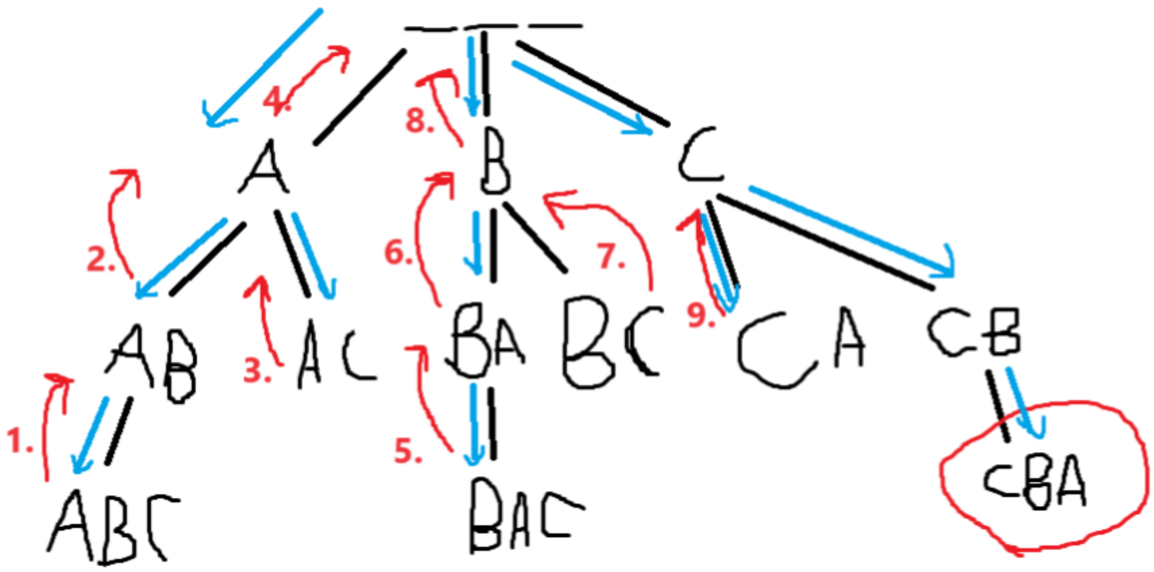
ii. The domains: these represent the possible degrees for any given philosopher. The domains are D= {1, 2, 3}, representing the order of the hikers, as it is a given that Alex and Charlie have PhD degrees and Bob has an MS degree.

iii. The constraints: these are the defined rules or restrictions for the problem. They are: adjacent hikers must have different degrees, and charlie does not want to be last. So, C = {A+1≠C, C+1≠A ,C≠3}

- Constraint Graph



i.



- Assignment = {}
- * Numbers below do not match with the numbers of the backtracking trace, but they occur in the same order

   i. Start with **A = 1**
      1. B = 2 ({A,B})
         a. C = 3 ({A,B,C})
            i. A + 1 ≠ C
            ii. C + 1 ≠ A
            iii. C ≠ 3 (not satisfied)
      2. C ≠ 3 not satisfied, **backtrack**. C = 2 ({A,C})
         a. A + 1 ≠ C (not satisfied)
      3. R*eturn to start **backtrack**.*

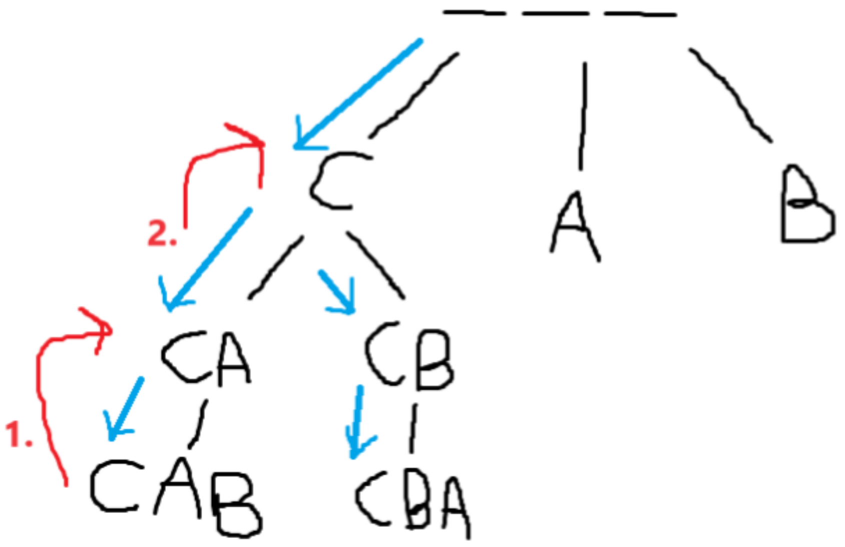
---

   ii. B = 1
      1. A = 2 ({B,A})
         a. C = 3 ({B,A,C})
            i. A + 1 ≠ C (not satisfied)
            ii. C + 1 ≠ A
            iii. C ≠ 3
      2. C ≠ 3, A + 1 ≠ C not satisfied, **backtrack.** C = 2 ({B,C})
         a. C + 1 ≠ A (not satisfied)
      3. *Return to start **backtrack.***

iii. **C = 1**
   1. A = 2 ({C,A})
      a. C + 1 ≠ A (not satisfied)
   2. C + 1 ≠ A not satisfied, **backtrack** B = 2 ({C,B})
      a. A = 3 ({C,B,A})
         i. A + 1 ≠ C
         ii. C + 1 ≠ A
         iii. C ≠ 3
   3. *All constraints satisfied.*

Final assignment: {C, B, A}

- MRV Heuristic – C has the fewest legal values, as it has the highest amount of constraints on it–therefore, we would want to start with C.



i.

ii. We start with C, then the next path would be A, then B (however, we will never need to travel those).

iii. Start with **C = 1**
   1. A = 2 ({C,A})
      a. A + 1 ≠ C
      b. C + 1 ≠ A (not satisfied)
   2. C + 1 ≠ A not satisfied. B = 2 ({C,B})
      a. A = 3
         i. A + 1 ≠ C
         ii. C + 1 ≠ A
         iii. C ≠ 3

3. *All constraints satisfied.*