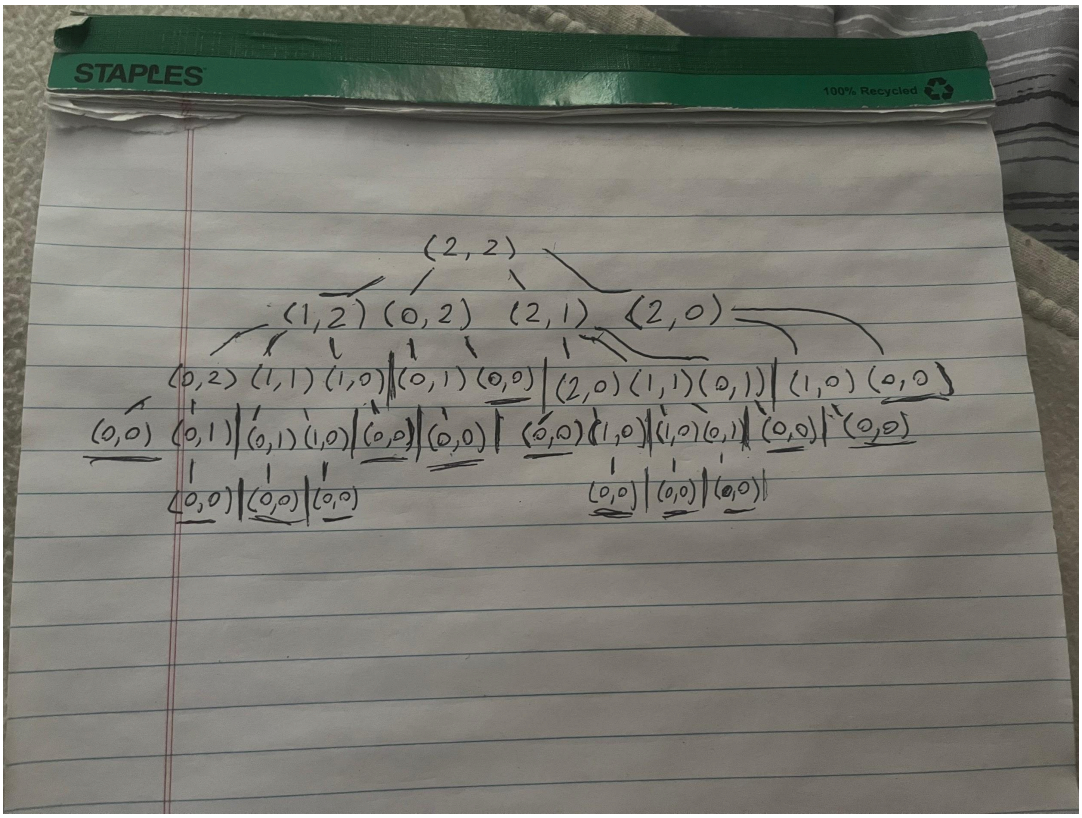
Problem 1

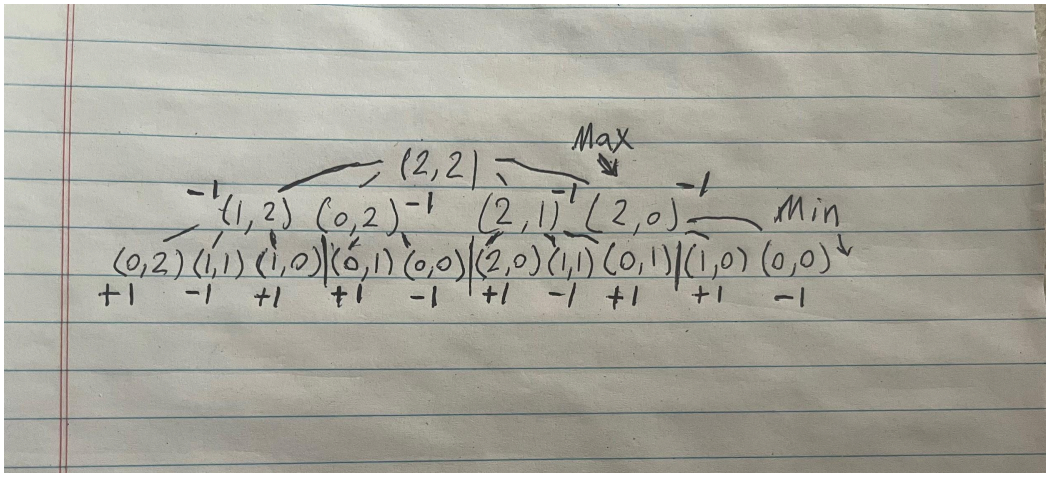
1. **Game Description:** Explain the rules of the game of Nim, including how many heaps there are, how many objects are in each heap at the start of the game, and how players take turns removing objects from the heaps.

There are 2 players in a game of Nim, and they alternate turns. The game starts with a certain number of heaps (2 or more to actually be a game), and within these heaps are a non-negative number of objects. To make a turn, a player must select a heap and take out 1 or more objects from that heap; once they do, their turn is over. To win the game, you must be the player who removes the last object so that there are no objects left in any of the heaps.

2. **Game Tree:** Draw the game tree for a simple case of Nim, such as a game with two heaps containing 3 and 4 objects, respectively. Show all possible moves and their outcomes at each level of the tree.
(Professor said it was okay to use 2 objects in the heap to make it more doable)

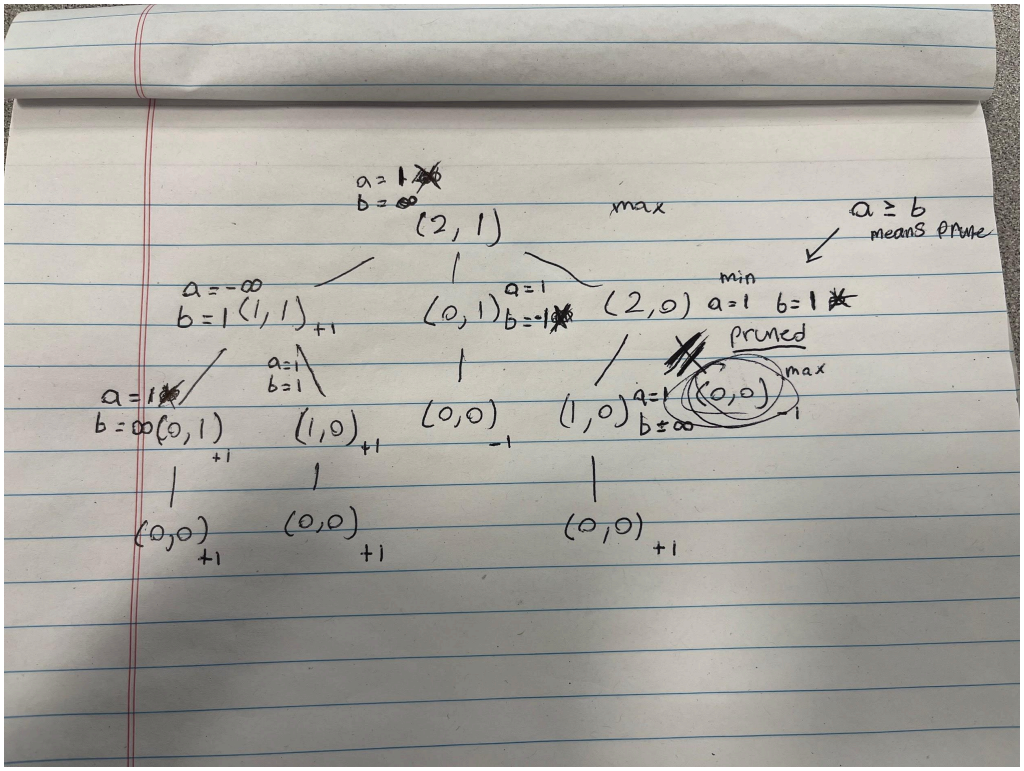


3. **Minimax Algorithm:** Apply the minimax algorithm to the game tree to determine the optimal move for each player at each level. Show the values assigned to each node in the tree by the minimax algorithm in a 2-ply game.



4. **Alpha-Beta Pruning:** Apply the alpha-beta pruning algorithm to the game tree to reduce the number of nodes that need to be evaluated. Show the alpha and beta values at each level of the tree and indicate which nodes were pruned.

(Used 2 heaps with 2 and 1 items just to make the alpha-beta pruning more manageable with the full tree)



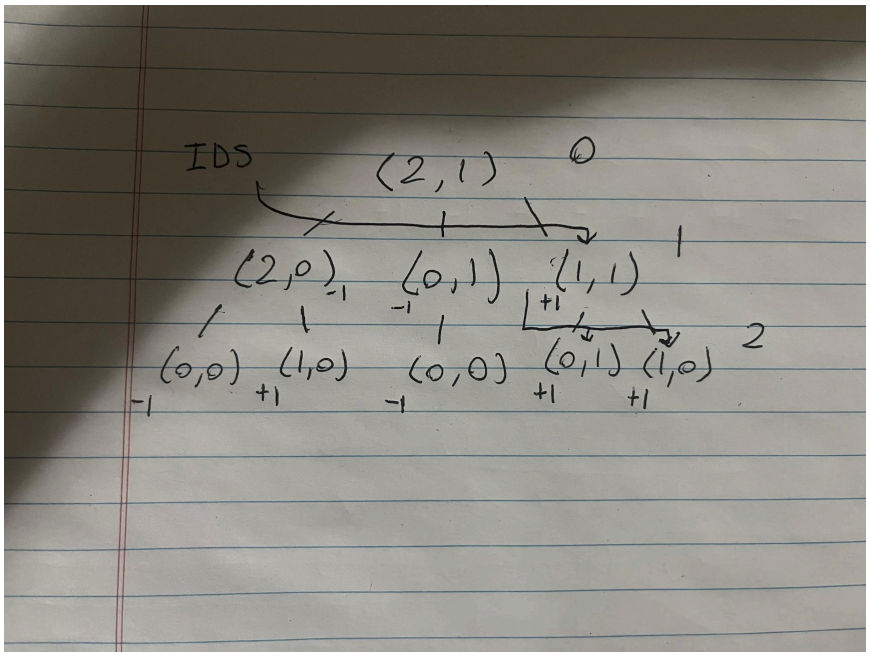
5. **Analysis:** Discuss the effectiveness of the minimax algorithm and alpha-beta pruning in solving the game of Nim. Consider factors such as the size of the game tree, number of heaps, number of objects in each heap, the number of nodes pruned by alpha-beta pruning, and the complexity of the game.

The minimax algorithm is effective in small games of Nim because it always finds the best possible move by exploring every future outcome. However, when the number of heaps or objects in them increases, the game tree becomes huge. The number of possible states does as well. A large game tree for a complex game of Nim with larger amounts of heaps or items in the heaps makes minimax too slow and complex to use alone on games.

Alpha-beta pruning also always finds the best possible move, but it improves minimax by cutting off parts of the tree that don't affect the final decision. This means that the algorithm will give the same solution but end up doing less work and checking fewer nodes. So alpha-beta pruning is more efficient than minimax in almost every situation where pruning is actually possible, but its effectiveness and efficiency still depend on the number of heaps and the number of items in them. A large game tree for a complex game of Nim with larger amounts of heaps or items will still greatly impact its speed and efficiency. The effectiveness and number of nodes pruned also depend on the ordering of the moves in the tree; the better they are ordered, the more nodes will get pruned.

Problem 2

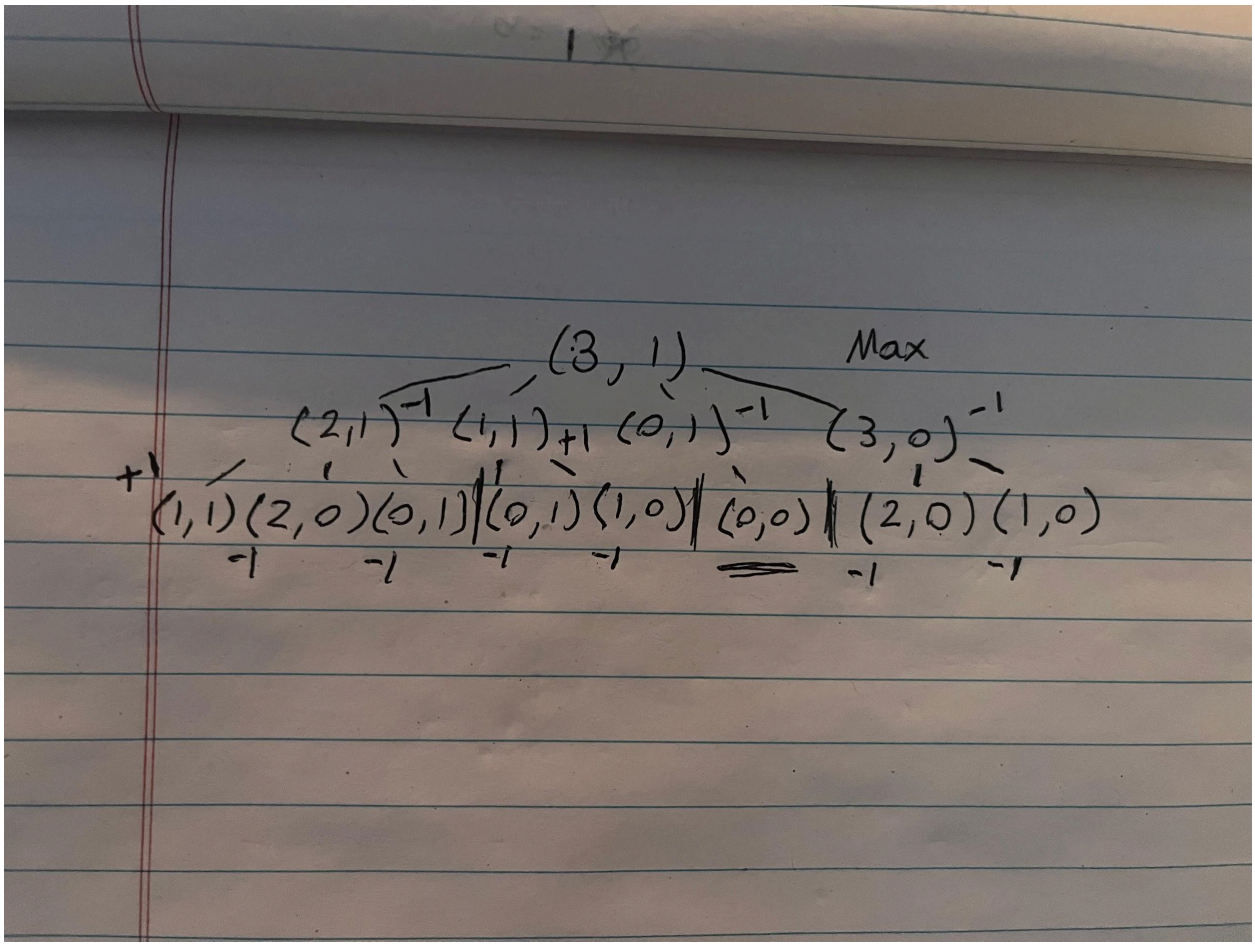
1. **Optimization Techniques:** Explore additional optimization techniques for the minimax algorithm, such as the use of iterative deepening, to improve its performance in analyzing larger game trees. Use a 2-ply game to analyze and explain your strategy.



So the way that I see iterative deepening being used to optimize the minimax algorithm is by finding the best winning moves without having to explore all of the child nodes first, which would especially help with larger game trees. The way you could use iterative deepening to do this is when you go and search through the first layer, your algorithm should be looking for moves that create the heaps to have the same number of items in a pile. I noticed that when a player makes a move that forces their opponent to make a move from a state where the two heaps are equal, it always or almost always results in a winning position for that player and a losing position for the opponent.

So in this example specifically, it would start searching the children of the head from left to right, looking for a move that makes two equal heaps. If it finds such a node, it will then continue its iterative deepening minimax search, continuing from that node's children. If it doesn't, then it will just continue on as it usually would. In this case it would find that with the (1,1) node and then continue its search from that node's children. Using this strategy, the algorithm can quickly find the best moves and skip over a lot of pointless moves that will just lead to a loss. Finding the best move of the child nodes by using this idea would greatly cut down on the algorithm's takes.

2. **Heuristic Evaluation:** Develop a heuristic evaluation function to estimate the value of game states in Nim, and compare the expected performance of heuristic-based minimax with the standard minimax algorithm. Discuss your approach using a 2-ply game.



The heuristic I came up with is based around the same idea I came up with in the last part of problem 2, basing it off the idea that in any given move you want to force your opponent to have two heaps of equal size because that will put them in a losing position. The heuristic function would be to do an XOR (exclusive or) on both heaps of the child nodes; if the XOR = 0, it means that the heaps have the same amount of items in them and if the XOR ≠ 0, it means that the heaps have different amounts of items in them. For this heuristic, if the XOR = 0, then the node will have a heuristic value of +1 and if the XOR ≠ 0, the node will have a heuristic value of -1.

As you can see in this example, when you apply the XOR to all of the child nodes of the root (possible moves for player one), it correctly identifies (1,1) as the best move by giving it a +1 heuristic because its XOR = 0. Then after this move there is no move that the opponent can make that will put them in a winning

position; there are only bad moves. This is better than the regular minimax algorithm because it explores every possible move and all of its future responses (children, grandchildren, …) before it can decide which move is best. When you use this heuristic, you can evaluate the first set of moves immediately, without needing to explore deeper levels.