Consider the following problem:

As per the law, it is a crime for an American to sell weapons to hostile nations. Country A an enemy of America has some missiles and all the missiles it were sold to it by Robert, who is an American citizen.

Prove that "Robert is criminal"

Proof:

- American(p) ∧ weapon(q) ∧ sells(p, q, r) ∧ hostile(r) ⇒ Criminal(p)

  p → American
  q → weapon
  r → Hostile
  p → Criminal(p)

- Country A has missiles. This can be expressed as:

  ∃x (Owns(A, x) ∧ missile(x))

  Country A owns an object x and that x is a missile

- Instantiating with a specific missile(T1):

  Owns(A, T1) ∧ missile(T1)

  We are picking a specific missile called T1 and replacing x with it

- ∀x (missile(x) ∧ Owns(A, x)) ⇒ Sells(Robert, x, A)

  Sells(Robert, T1, A) — Robert sold missile T1 to country A

  For every missile country owns Robert sold all the missiles to country A

- Missiles are weapons

  missiles(x) ⇒ weapons(x)

- Enemy of america which is country A is a hostile nation

  ∀x (Enemy(x, America) ⇒ Hostile(x))

- Robert is an American

  American(Robert)

Applying the law

American (Robert)

Weapon (T1)

Robert sold missile T1 to country A:

Sells (Robert, T1, A)

Hostile (A)

∴ American (p) ∧ Weapon (q) ∧ Sells (p, q, r) ∧ Hostile (r) ⇒ Criminal (p)

p = Robert

q = T1

r = A

American (Robert) ∧ Weapon (T1) ∧ Sells (Robert, T1, A) ∧ Hostile (A) ⇒ Criminal (Robert)

Proceed

O/P:

Robert is a criminal

Alpha-Beta pruning in 8-Queens

```
def is_valid(board, row, col):
    for r in range(row):
        if board[r] == col or board[r] - r == col - row:
            return False
    return True

def alpha_beta(board, row, alpha, beta):
    if a queen is placed in all rows we found a solution
    if row == 8:
        return True
    Loop through each column
    for col in range(8):
        if is_valid(board, row, col):
            board[row] = col
            If the current branch cannot result in a better solution
            if alpha >= beta:
                break
            Recursively iterate for each row
            if alpha_beta(board, row+1, alpha, beta):
                return True
            Undo current move
            board[row] = -1
    return False

def solve():
    board = [-1] * 8
    alpha = float('inf')
    beta = float('inf')
    if alpha_beta(board, 0, alpha, beta):
        return board
    else:
        return None
```

Proceed

O/P:
Solution found
[0, 4, 7, 5, 2, 6, 1, 3]

Mini Max Algorithm for Tic Tac Toe

```
def gameover(board):
    # check rows, columns for a win
    for row in range(3):
        if board[row][0] == board[row][1] == board[row][2]
        and board[row][0] != ' ':
            return True
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col]
        and board[0][col] != ' ':
            return True
    if board[0][0] == board[1][1] == board[2][2] and
    board[0][0] != ' ':
    # check if board is full
    for row in range(3):
        for col in range(3):
            if board[row][col] == ' ':
                return False
    return True
# Evaluate the board and return the score
def evaluate(board):
    if board[row][0] == 'X':
        return 10
    else:
        return -10
    if board[col][0] == 'X':
    if board[0][0] == 'X':
    if board[0][2] == 'X':
    return 0
```

'X' : 10

'O' : -10

'draw' : 0

```
def minimax(board, depth):
    # Explore all possible future moves, evaluate their
    outcomes and choose the best one for the current
    if is_gameOver(board):
        return evaluate(board) # Return score
    # Try every possible move along rows
    for row in range(3):
        for col in range(3):
            if board[row][col] == ' ':
                board[row][col] = 'X'
                eval = minimax(board, depth+1, False)
    # Try every possible moves along column
    else
            if board[row][col] == ' ':
                board[row][col] == 'O'
                eval = minimax(board, depth+1, True)
```

Output

current board

X O X

O X O

best move is (2,0)

Proved

3/12/24