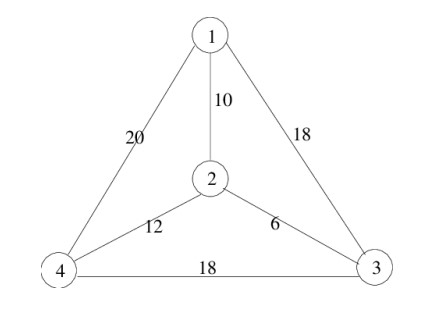
***NAME: UMAMA SAIFULLAH***

***BS(SE)2019***

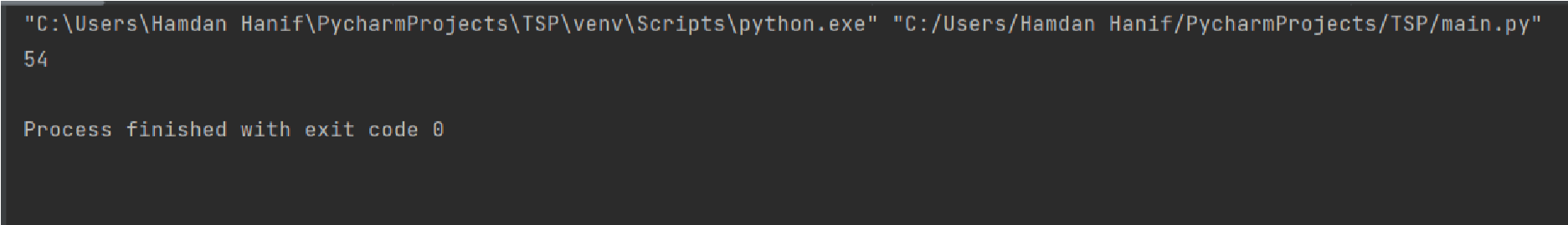
***AI LAB FILE TASK***

**Task 1**

****

|  |
| --- |
| **from sys import maxsize**  **from itertools import permutations**    **V = 4**  **def travellingSalesmanProblem(graph, s):**  **vertex = [] for i in range(V): if i != s:**  **vertex.append(i)**    **min\_path = maxsize**  **next\_permutation = permutations(vertex) for i in next\_permutation: current\_pathweight = 0 k = s for j in i: current\_pathweight += graph[k][j] k = j**  **current\_pathweight += graph[k][s]**  **min\_path = min(min\_path, current\_pathweight)**    **return min\_path**  **if \_\_name\_\_ == "\_\_main\_\_":**  **graph = [[0, 10, 18, 20], [10, 0, 6, 12], [18, 6, 0, 18], [20, 12, 18, 0]] s = 0**  **print(travellingSalesmanProblem(graph, s))** |

**Output:**

**Shortest path is: **

**Task 2**

**Q: Create facts from the following information:**

Joe is male.

Pat is male.

Sharon is female.

Mary is female.

Joe is the father of Mary.

Sharon is the mother of Mary.

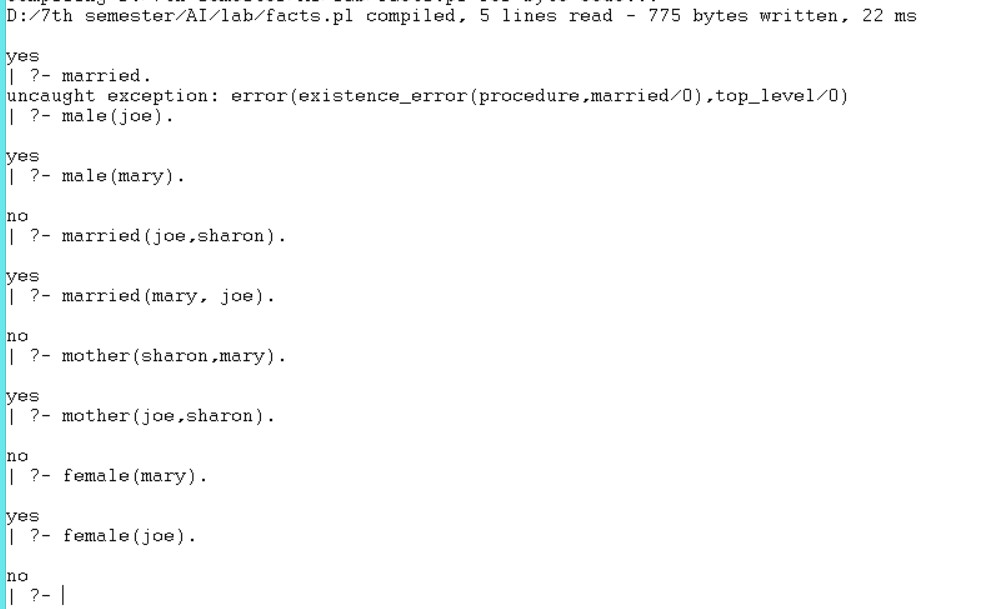
Joe is marriedto Sharon.

**Solution:** male(joe). male(pat).

female(sharon). female(mary).

mother(sharon,mary).married(joe,sharon).

**Output:**

****

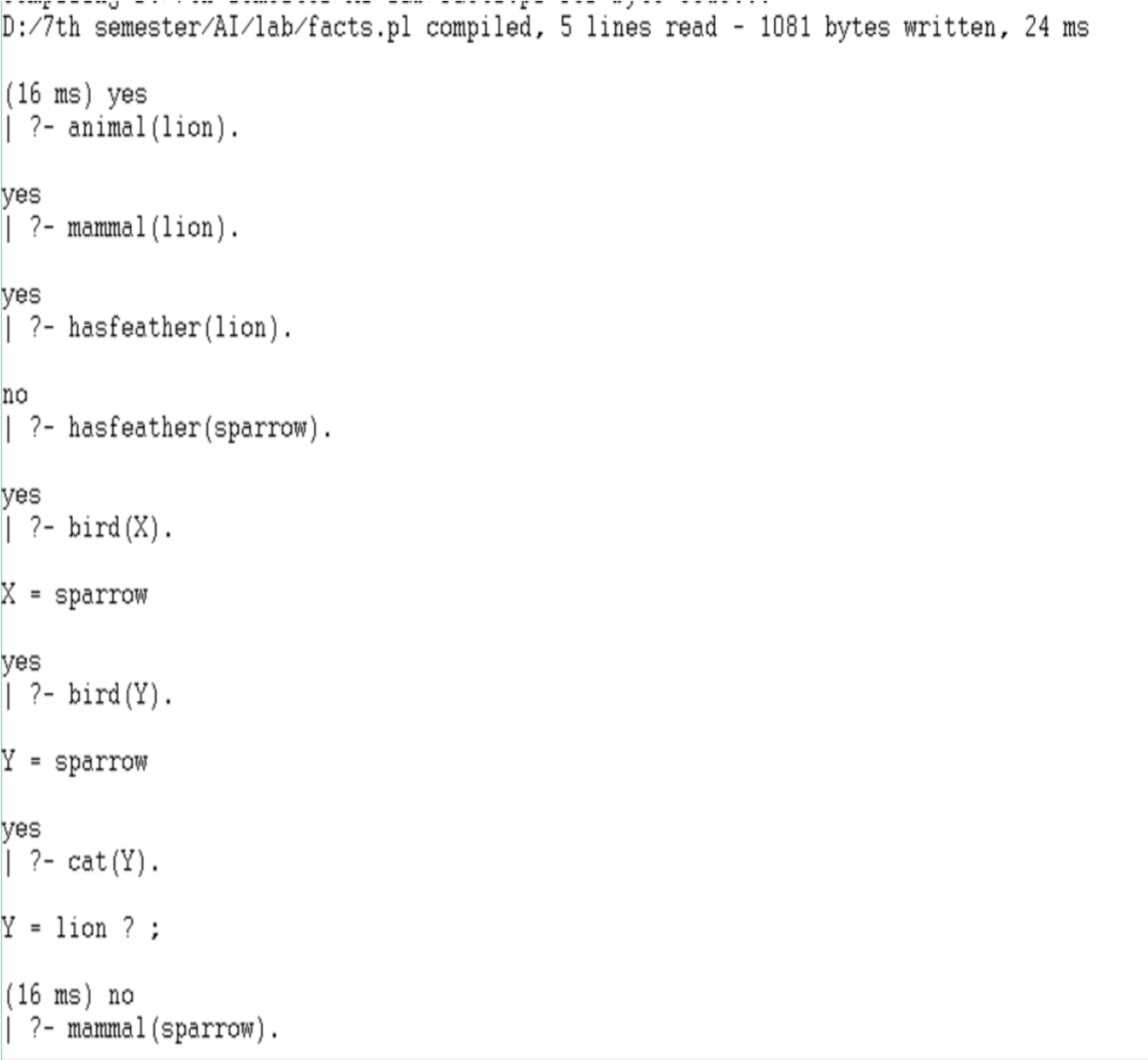
**Task 3**

**Problem 1:** animal(lion).

animal(sparrow). hasfeather(sparrow). mammal(lion).

bird(X):-animal(X),hasfeather(X). cat(Y):- animal(Y), mammal(Y).

**Output:**

****

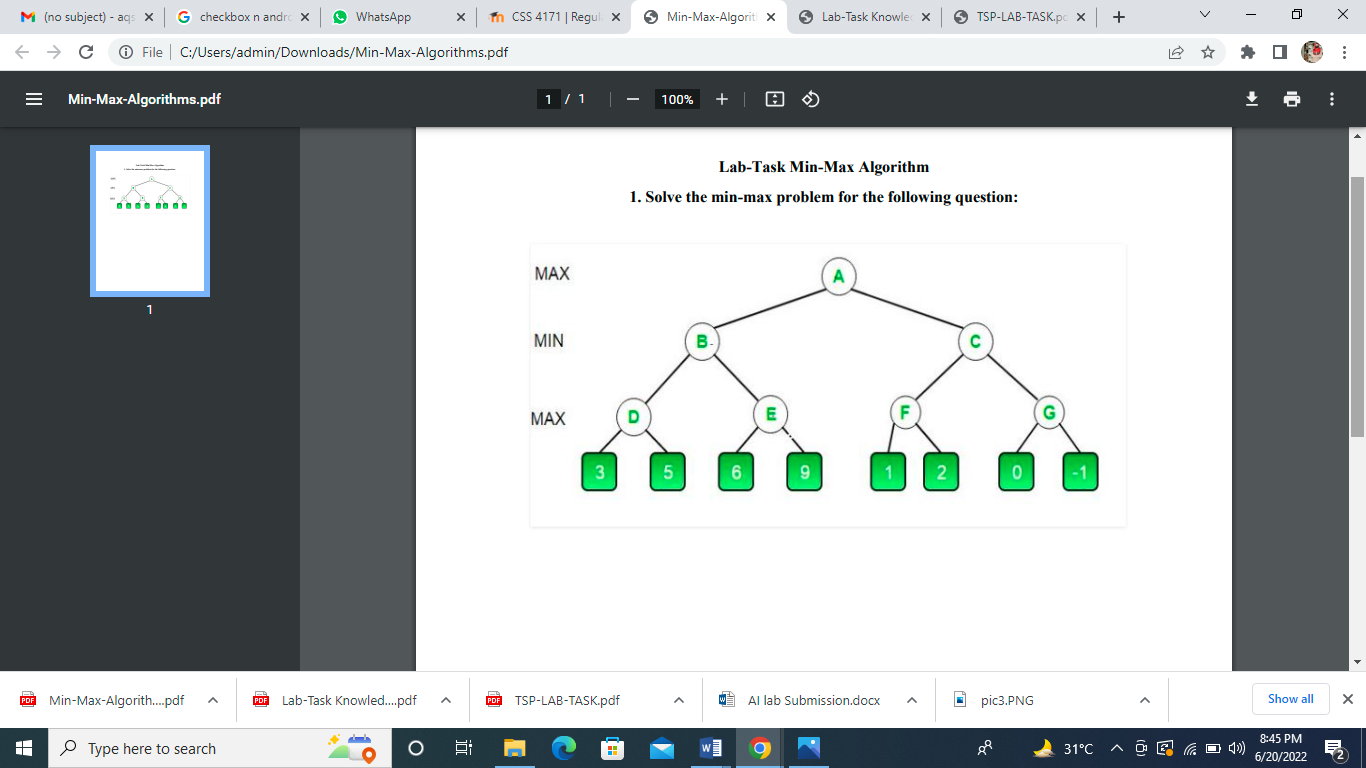
**Problem 2:**

likes(mary,food). likes(mary,apple). likes(tom,apple). likes(tom,mary)

**Output:**

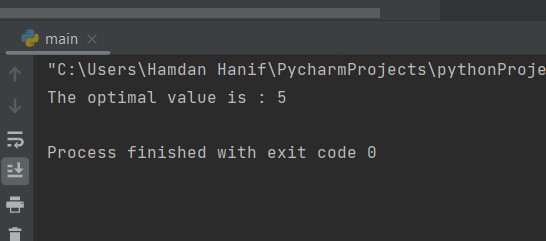
****

**Task 4**

****

|  |
| --- |
| **import math**  **def minimax (curDepth, nodeIndex, maxTurn, scores, targetDepth):**    **# base case : targetDepth reached if (curDepth == targetDepth): return scores[nodeIndex]**  **if (maxTurn):**  **return max(minimax(curDepth + 1, nodeIndex \* 2, False, scores, targetDepth),**  **minimax(curDepth + 1, nodeIndex \* 2 + 1, False, scores, targetDepth))**  **else:**  **return min(minimax(curDepth + 1, nodeIndex \* 2, True, scores, targetDepth),**  **minimax(curDepth + 1, nodeIndex \* 2 + 1, True, scores, targetDepth))**    **# Driver code**  **scores = [3, 5, 6, 9, 1, 2, 0, -1]**    **treeDepth = math.log(len(scores), 2)**    **print("The optimal value is : ", end = "") print(minimax(0, 0, True, scores, treeDepth))** |

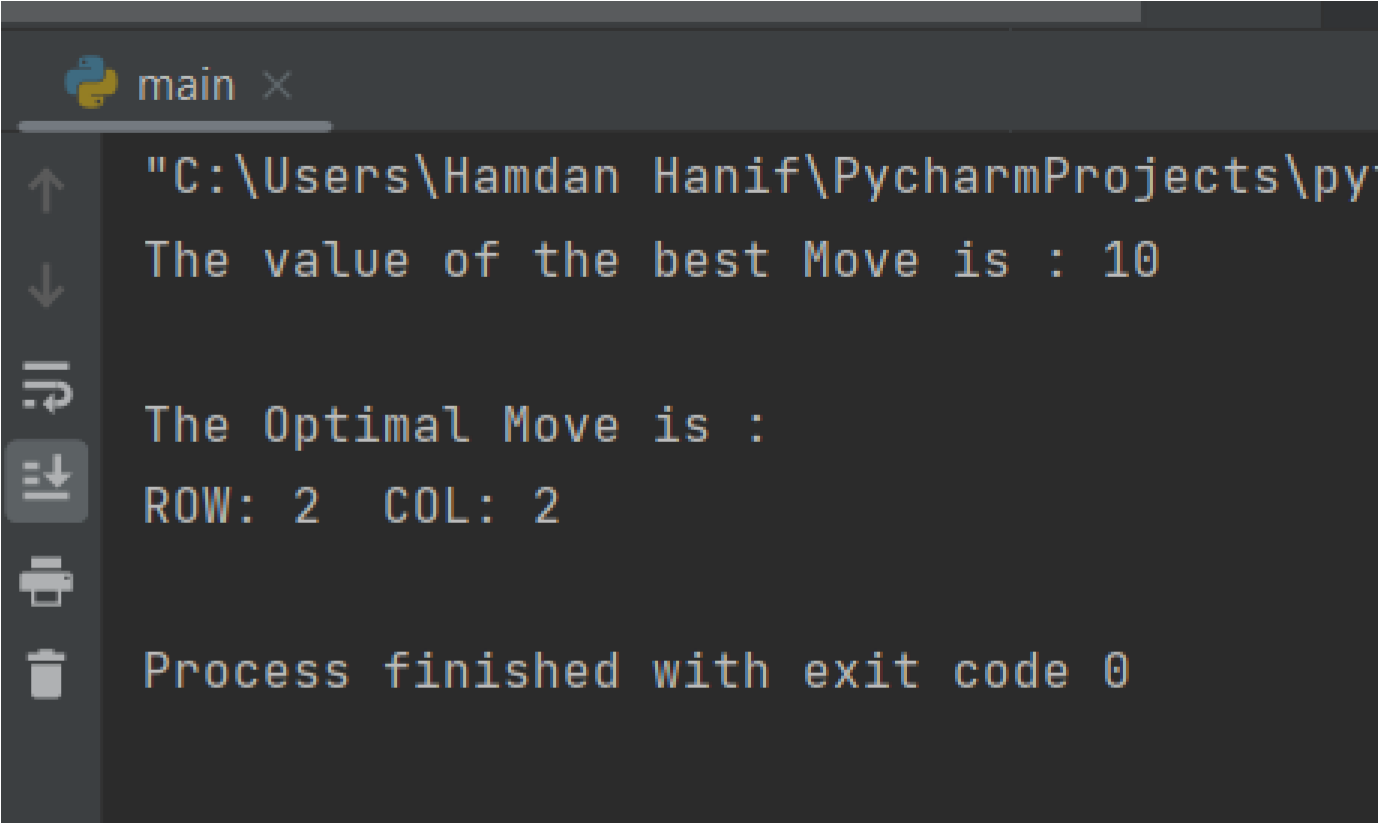
**Output:**

****

**Task 5**

|  |
| --- |
| **player, opponent = 'x', 'o'**  **def isMovesLeft(board): for i in range(3): for j in range(3): if (board[i][j] == '\_'):**  **return True return False**  **def evaluate(b):**  **for row in range(3): if (b[row][0] == b[row][1] and b[row][1] == b[row][2]): if (b[row][0] == player):**  **return 10 elif (b[row][0] == opponent):**  **return -10**  **for col in range(3):**  **if (b[0][col] == b[1][col] and b[1][col] == b[2][col]):**  **if (b[0][col] == player):**  **return 10 elif (b[0][col] == opponent):**  **return -10**  **if (b[0][0] == b[1][1] and b[1][1] == b[2][2]):**  **if (b[0][0] == player):**  **return 10 elif (b[0][0] == opponent):**  **return -10** |
| **if (b[0][2] == b[1][1] and b[1][1] == b[2][0]):**  **if (b[0][2] == player):**  **return 10 elif (b[0][2] == opponent):**  **return -10**    **return 0**  **def minimax(board, depth, isMax):**  **score = evaluate(board)**  **if (score == 10): return score**  **if (score == -10): return score**  **if (isMovesLeft(board) == False):**  **return 0**  **if (isMax):**  **best = -1000**  **for i in range(3): for j in range(3):**  **if (board[i][j] == '\_'):**    **board[i][j] = player**  **best = max(best, minimax(board, depth + 1, not isMax))**    **board[i][j] = '\_' return best**  **else:**  **best = 1000**  **for i in range(3):**  **for j in range(3):**  **if (board[i][j] == '\_'):**    **board[i][j] = opponent**    **best = min(best, minimax(board, depth + 1, not isMax))**  **board[i][j] = '\_' return best**  **def findBestMove(board): bestVal = -1000 bestMove = (-1, -1)** |
| **for i in range(3): for j in range(3):**  **if (board[i][j] == '\_'):**    **board[i][j] = player**  **moveVal = minimax(board, 0, False)**  **board[i][j] = '\_' if (moveVal > bestVal): bestMove = (i, j) bestVal = moveVal**    **print("The value of the best Move is :", bestVal) print()**  **return bestMove**    **board = [**  **['x', 'o', 'x'],**  **['o', 'o', 'x'],**  **['\_', '\_', '\_']**  **]**  **bestMove = findBestMove(board)**    **print("The Optimal Move is :")**  **print("ROW:", bestMove[0], " COL:", bestMove[1])** |

**Output:**

****