-------------------------------------------------

A\* Algorithm for 8-Queens Problem: Line-by-Line Explanation

-------------------------------------------------

-------------------------------------------------

[1] import heapq

- Imports the heap queue (priority queue) module from Python’s standard library.

- It allows us to maintain the open list as a min-heap efficiently.

-------------------------------------------------

[3] def calculate\_heuristic(board):

- Defines a function to calculate the heuristic value of the current board.

- Heuristic: number of attacking queen pairs.

-------------------------------------------------

[4] attacking\_pairs = 0

- Initializes a counter to store the number of conflicting (attacking) queen pairs.

-------------------------------------------------

[5-9] for i in range(n):

for j in range(i + 1, n):

if board[i] == board[j] or abs(board[i] - board[j]) == abs(i - j):

attacking\_pairs += 1

- Nested loop checks all pairs of queens.

- board[i] == board[j]: same column conflict.

- abs(board[i] - board[j]) == abs(i - j): diagonal conflict.

-------------------------------------------------

[10] return attacking\_pairs

- Returns the total heuristic (number of attacking pairs).

-------------------------------------------------

[12] def print\_board(board):

- Defines a function to visually print the current board configuration.

-------------------------------------------------

[14-16] for row in range(n):

board\_row = ['Q' if col == board[row] else '.' for col in range(n)]

print(' '.join(board\_row))

- For each row, build a list where:

- 'Q' at the queen’s position,

- '.' at empty positions.

- Prints the visual row.

-------------------------------------------------

[17] print()

- Prints a blank line for clarity.

-------------------------------------------------

[19] def a\_star(start\_board):

- Defines the main A\* algorithm function.

-------------------------------------------------

[21-22] open\_list = []

heapq.heappush(open\_list, (0 + calculate\_heuristic(start\_board), 0, start\_board))

- Initializes the priority queue with the starting board.

- f = g + h (cost so far + heuristic).

- g = 0 (initial), h = calculated heuristic.

-------------------------------------------------

[24] closed\_list = set()

- Initializes a set to keep track of visited (explored) board configurations.

-------------------------------------------------

[26] while open\_list:

- While the priority queue is not empty, continue searching.

-------------------------------------------------

[27] f, g, board = heapq.heappop(open\_list)

- Pops the board with the lowest f-value (best estimated cost) from the queue.

-------------------------------------------------

[29-32] if calculate\_heuristic(board) == 0:

print("Solution Found!")

print\_board(board)

return board

- If heuristic is zero (no conflicts), solution is found.

- Prints the board and exits.

-------------------------------------------------

[34-39] for i in range(n):

for j in range(n):

if board[i] != j:

new\_board = board[:]

new\_board[i] = j

- Generates neighbor boards by moving each queen in its row to a different column.

- board[:] creates a copy of the current board.

-------------------------------------------------

[41-48] if tuple(new\_board) not in closed\_list:

closed\_list.add(tuple(new\_board))

print(f"Placing queen at row {i}, column {j} (step: {g + 1}):")

print\_board(new\_board)

heapq.heappush(open\_list, (g + 1 + calculate\_heuristic(new\_board), g + 1, new\_board))

- If the new board hasn’t been visited:

- Add it to the closed list.

- Print its state.

- Push it to the priority queue with updated g, h, f values.

-------------------------------------------------

[51] return None

- If no solution is found (open\_list exhausted), return None.

-------------------------------------------------

[53] def get\_user\_input():

- Defines a function to get the initial board configuration from the user.

-------------------------------------------------

[56-57] print("Ankur Sonavane\n" "22129\n DIV:A")

- Displays student info.

-------------------------------------------------

[60-66] while True:

try:

user\_input = input("Enter the initial state (a list of 8 integers from 0 to 7): ")

initial\_state = list(map(int, user\_input.strip('[]').split(',')))

if len(initial\_state) == 8 and all(0 <= x < 8 for x in initial\_state):

print(f"Initial state: {initial\_state}")

return initial\_state

else:

print("Invalid input. Please enter a list of 8 integers, each between 0 and 7.")

except ValueError:

print("Invalid input. Please enter a valid list of integers.")

- Takes user input as a list of 8 integers.

- Validates the input format and values.

-------------------------------------------------

[69] initial\_state = get\_user\_input()

- Calls the input function to get the starting board.

-------------------------------------------------

[72] a\_star(initial\_state)

- Runs the A\* search algorithm on the provided initial board.

-------------------------------------------------

Possible External Viva Questions and Answers

-------------------------------------------------

1️⃣ What is the 8-Queens problem?

- Placing 8 queens on an 8x8 chessboard so that no two queens attack each other.

2️⃣ What search method is used?

- A\* search, an informed search algorithm.

3️⃣ What is the heuristic function?

- Number of pairs of queens attacking each other.

4️⃣ Why use a priority queue?

- To always explore the board with the lowest estimated total cost.

5️⃣ What are g and h in A\*?

- g: cost so far (steps taken), h: heuristic estimate to goal.

6️⃣ How is the board represented?

- As a list where index = row and value = column position of the queen.

7️⃣ Why store board states in a tuple for the closed list?

- Because lists are mutable (unhashable) and can’t be added to sets.

8️⃣ What happens if no solution is found?

- The function returns None after exhausting all possibilities.

9️⃣ What are the advantages of A\* over other search methods?

- Completeness and optimality if the heuristic is admissible.

🔟 What are alternative solutions to the 8-Queens problem?

- Backtracking, hill climbing, simulated annealing, genetic algorithms.

-------------------------------------------------

Syllabus-Based (TE Computer Engineering, SPPU LP2) Questions

-------------------------------------------------

✅ Define heuristic search.

- Search using domain-specific knowledge to guide the search process.

✅ What is the difference between blind and informed search?

- Blind search: no extra knowledge; informed search: uses heuristics.

✅ Explain the time complexity of A\*.

- Depends on the heuristic; with a good heuristic, fewer nodes are expanded.

✅ What are admissible heuristics?

- Heuristics that never overestimate the cost to reach the goal.

✅ Applications of A\* algorithm?

- Pathfinding (games, robotics), puzzle solving, route planning.

-------------------------------------------------

Prepared for practical understanding and viva preparation.