"""

Assignment-A2 (A\* for 8 Puzzle Problem)

Problem Statement: Problem Statement: Implement A star Algorithm for 8 puzzle problem.

"""

# This code implements A\* algorithm for 8 puzzle problem since the problem

# statement in our syllabus and handout demands implementing A\* algorithm

# for any game search problem.

# A\* implementation for maze problem and direct implementation can be found

# in the "Alternatives" folder.

# Enter the initial state as 9 numbers (0 represents the blank tile):

# Enter row 1 (3 numbers separated by spaces): 1 2 3

# Enter row 2 (3 numbers separated by spaces): 0 4 6

# Enter row 3 (3 numbers separated by spaces): 7 5 8

import heapq

goal\_state = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

# Directions for moving the blank tile

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)] # up, down, left, right

def h\_misplaced\_tiles(state):

misplaced = 0

for i in range(3):

for j in range(3):

if state[i][j] != 0 and state[i][j] != goal\_state[i][j]:

misplaced += 1

return misplaced

def find\_blank(state):

for i in range(3):

for j in range(3):

if state[i][j] == 0:

return i, j

def is\_valid(x, y):

return 0 <= x < 3 and 0 <= y < 3

def state\_to\_tuple(state):

return tuple(tuple(row) for row in state)

def a\_star(start\_state):

start = state\_to\_tuple(start\_state)

g = 0

h = h\_misplaced\_tiles(start\_state)

f = g + h

# Priority queue with elements: (f, g, state, path)

pq = [(f, g, start\_state, [])]

visited = set()

while pq:

f, g, current, path = heapq.heappop(pq)

current\_tuple = state\_to\_tuple(current)

if current\_tuple in visited:

continue

visited.add(current\_tuple)

if current == goal\_state:

return path + [current]

x, y = find\_blank(current)

for dx, dy in directions:

nx, ny = x + dx, y + dy

if is\_valid(nx, ny):

# Create new state by swapping blank

new\_state = [row[:] for row in current]

new\_state[x][y], new\_state[nx][ny] = new\_state[nx][ny], new\_state[x][y]

if state\_to\_tuple(new\_state) not in visited:

new\_g = g + 1

new\_h = h\_misplaced\_tiles(new\_state)

new\_f = new\_g + new\_h

heapq.heappush(pq, (new\_f, new\_g, new\_state, path + [current]))

return None

def print\_path(path):

for step, state in enumerate(path):

print(f"Step {step}:")

for row in state:

print(row)

print()

def get\_input\_state():

print("Enter the initial state as 9 numbers (0 represents the blank tile):")

state = []

for i in range(3):

row = input(f"Enter row {i+1} (3 numbers separated by spaces): ").split()

state.append([int(num) for num in row])

return state

initial\_state = get\_input\_state()

solution\_path = a\_star(initial\_state)

if solution\_path:

print\_path(solution\_path)

else:

print("No solution found.")

"""

SAMPLE OUTPUT

Enter the initial state as 9 numbers (0 represents the blank tile):

Enter row 1 (3 numbers separated by spaces): 1 2 3

Enter row 2 (3 numbers separated by spaces): 0 4 6

Enter row 3 (3 numbers separated by spaces): 7 5 8

Step 0:

[1, 2, 3]

[0, 4, 6]

[7, 5, 8]

Step 1:

[1, 2, 3]

[4, 0, 6]

[7, 5, 8]

Step 2:

[1, 2, 3]

[4, 5, 6]

[7, 0, 8]

Step 3:

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]

"""

Sure! Here's a **simple explanation** of the given A\* algorithm for the **8 Puzzle Problem**, followed by **viva questions and their answers**.

**✅ Easy Explanation of the Code**

**🔷 Problem Summary**

You're solving the **8 Puzzle Problem** — a 3x3 board where:

* Tiles are numbered 1–8.
* One tile is blank (represented by 0).
* The goal is to **reach the goal state** by sliding tiles.

**🔷 Goal State**

goal\_state = [[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

This is the final desired arrangement.

**🔶 What is A\* Algorithm?**

A\* (A-Star) is a **best-first search algorithm** used in pathfinding.  
It uses the formula:

f(n) = g(n) + h(n)

Where:

* g(n) = cost to reach the current node from the start.
* h(n) = estimated cost from current node to goal (called heuristic).
* f(n) = total estimated cost of the path through the current node.

In this code, h(n) is the **number of misplaced tiles**.

**🔷 Code Breakdown**

**1. Misplaced Tile Heuristic**

def h\_misplaced\_tiles(state):

...

Counts how many tiles are **not in their correct position** (excluding the blank).

**2. Find Blank Position**

def find\_blank(state):

...

Finds the location (i, j) of the 0 (blank tile) in the board.

**3. State Conversion**

def state\_to\_tuple(state):

return tuple(tuple(row) for row in state)

Converts a list-of-lists into a tuple-of-tuples, so it can be stored in a set (for tracking visited states).

**4. *A Search*\***

def a\_star(start\_state):

...

* Starts from the initial state.
* Uses a **priority queue** (heapq) to always expand the path with the lowest f(n) first.
* At each step:
  + It checks if the current state is the goal.
  + If not, it explores **all 4 directions** (up, down, left, right) by swapping the blank tile.
  + For each new state, calculates:
    - g: the path cost so far
    - h: number of misplaced tiles
    - f = g + h
  + Adds it to the priority queue.
* Stops when the goal is reached or no more paths exist.

**5. Printing the Path**

def print\_path(path):

...

Prints each step from the initial state to the goal state.

**6. Getting Input**

def get\_input\_state():

...

Takes user input as a 3x3 board.

**🧠 Sample Flow**

Input:

1 2 3

0 4 6

7 5 8

Goal:

1 2 3

4 5 6

7 8 0

The algorithm slides tiles step-by-step until the board matches the goal.

**🎤 Viva Questions & Answers**

***1. What is the A algorithm?*\***

A\* is a pathfinding algorithm that uses both the cost from the start (g) and an estimated cost to the goal (h). It picks the path with the lowest total cost f = g + h.

**2. What heuristic is used in this code?**

The number of **misplaced tiles** (how many tiles are not in the correct place).

**3. Why use a priority queue (heap)?**

To always expand the path with the **lowest f-value first**, ensuring we move toward the optimal solution efficiently.

**4. What is the role of the visited set?**

It prevents revisiting already explored states, avoiding infinite loops and improving efficiency.

***5. What is g, h, and f in A?*\***

* g: Cost from the start to the current state (number of moves).
* h: Heuristic (estimated cost to goal).
* f: Total cost = g + h.

***6. Can A guarantee the shortest path?*\***

Yes, if the heuristic is **admissible** (never overestimates the cost), A\* always finds the optimal path.

**7. Why is 0 used to represent the blank tile?**

0 is a neutral number that can be easily recognized and ignored during heuristic checks.

**8. What are valid moves in 8-puzzle?**

The blank tile (0) can move **up, down, left, or right**, swapping with an adjacent number tile.

**9. What is the purpose of state\_to\_tuple()?**

Python set can only store **immutable data**. So we convert the state (list of lists) into a tuple of tuples for checking visited states.

**10. How does the program avoid going back and forth between the same two states?**

By checking if a state is in the visited set before exploring it again.

**11. What happens if no solution exists?**

The function returns None, and the program prints "No solution found."

***12. What is the time complexity of A?*\***

In the worst case, it's **exponential**, but A\* is more efficient than BFS or DFS when a good heuristic is used.

Here are the most relevant **viva questions with answers** for the A\* Algorithm for the 8 Puzzle Problem, specifically tailored for the **SPPU 2019 Pattern (Assignment A2)**:

**🔹 Conceptual Questions**

1. **Q: What is the 8 puzzle problem?**  
   **A:** The 8 puzzle is a sliding puzzle consisting of a 3x3 grid with 8 numbered tiles and one blank space. The goal is to move the tiles to reach a specific goal state by sliding one tile at a time into the blank space.
2. *Q: What is the A algorithm?*\*  
   **A:** A\* is a heuristic-based search algorithm used to find the shortest path from a start state to a goal state. It uses the cost function:  
   f(n)=g(n)+h(n)f(n) = g(n) + h(n)  
   where
   * **g(n)** is the cost from start to node n
   * **h(n)** is the heuristic estimate from n to goal
3. *Q: Why is A considered an informed search algorithm?*\*  
   **A:** Because it uses heuristics (h(n)) to estimate the cost to reach the goal, thereby guiding the search in a more efficient way.

**🔹 Heuristic Questions**

1. **Q: Which heuristic is used in your implementation?**  
   **A:** The heuristic used is the **misplaced tile heuristic**, which counts the number of tiles that are not in their goal position.
2. **Q: Is the misplaced tile heuristic admissible? Why?**  
   **A:** Yes, it is admissible because it never overestimates the cost to reach the goal — it's a lower bound on the actual moves required.
3. **Q: Can we use Manhattan Distance as a heuristic?**  
   **A:** Yes, Manhattan Distance (sum of horizontal and vertical distances from the goal position) is another admissible and often better heuristic than misplaced tiles.

**🔹 Code & Implementation Questions**

1. **Q: What data structure is used to implement the priority queue?**  
   **A:** Python’s heapq module is used to implement a min-heap priority queue based on the f(n) value.
2. **Q: Why do we use a set called visited?**  
   **A:** To avoid revisiting the same states and entering infinite loops or redundant processing.
3. **Q: How are states compared in your code?**  
   **A:** States are converted into tuples of tuples using state\_to\_tuple() so they can be hashed and compared efficiently in sets.
4. **Q: What is the role of the path list in the code?**  
   **A:** It keeps track of the sequence of states from the initial to the current state, to reconstruct and display the full solution path.

**🔹 Edge Cases / Theoretical Questions**

1. **Q: Is the 8 puzzle problem always solvable?**  
   **A:** No. Only half of all possible initial configurations are solvable. Solvability depends on the number of **inversions** in the puzzle.
2. **Q: What is an inversion in the 8 puzzle?**  
   **A:** An inversion is a pair of tiles (a, b) such that a appears before b but a > b. The puzzle is solvable only if the total number of inversions is even.
3. *Q: Can A get stuck in infinite loop?*\*  
   **A:** No, not if implemented correctly with a visited set. It always terminates if a solution exists or declares failure otherwise.

**🔹 Time and Space Complexity**

1. *Q: What is the time complexity of A for 8 puzzle?*\*  
   **A:**

* **Worst case time complexity:** O(b^d), where b is branching factor and d is the depth of the solution.
* For 8-puzzle, branching factor ≈ 2.13 on average.

1. **Q: What is the space complexity?**  
   **A:**

* **Space complexity:** O(b^d) as well, due to the storage of paths and visited states.

Would you like these questions compiled into a **PDF viva guide** or with **additional MCQs** for practice?