Topic : Page No. : Date. : / /

Assignment - 01 (AIES)

* Tittle :
Implementation of A* algorithm for 8 puzzle problem

* Aim :
Solve 8 puzzle problem using A* algorithm

* Objective :
To study and implement A* algorithm for 8 puzzle problem

* Theory :
1.) Uniformed Seach
Uniformed seach enplore the seach space without any domain specific knowledge. They are also called blind seach algorithms and they rely purely on the structure of the problem to find a solution.
Examples : BFS and DFS

2.) BFS and DFS
BFS : Explores the seach tree level by level.
Uses a queue to keep track of nodes.
Time Complexity : $O(b^d)$
Space Complexity : $O(b^d)$

Topic : _____ Page No. : ____ Date. : / /

DFS : Explores as deep as possible down one branch before backtracking.
Uses a stack to manage nodes.
Time Complexity : $O(b^m)$
Space Complexity : $O(bm)$

3) Informed search :
Informed seach algorithms use domain specific knowledge to find solutions more efficiently. They employ heuristic to estimate the cost to reach the goal from a given node.
Examples : Greedy Best first search
A* seach.

4) A* Algorithm :
The A* algorithm is an informed seach algorithm that aims to find the shortest path to a goal. It prioritizes the nodes by using a priority queue based on the function $f(n) = g(n) + h(n)$

5) Data structure :
A* uses priority queue to manage the nodes, ensuring
The heuristic function should be admisible.

Topic : Page No. : Date. : / /

* Input :
Initial state

* Output :
Solution / goal state with optimal path

* Platform :
Windows / linux

* FAQ's :

Q.1. What is a heuristic function? What is the advantage of using a heuristic function?

→ A heuristic function is a method used in algorithms to estimate the cost or value of reaching a goal from a given state. It provides a way to evaluate which states are more promising, guiding the search process more effectively.
Heuristic functions are more commonly used in algorithms like A* and greedy best first search.
Advantages :
- In A* algorithms, a well designed heuristic function can make the search optimal and complete.
- Heuristic function can drastically reduce the time it takes to find a solution.
- Help in reducing the complexity of problems.

Topic : Page No. : Date. : / /

Q.2. Explain A* algorithm with an example.

→ The A* algorithm combines the g(n) (cost from start to current node) and h(n) to find the path with the lowest total cost.
Example : In the 8 puzzle problem, A* uses the Manhattan distance as a heuristic function to estimate the number of moves needed to reach the goal state.
It explores nodes with the lowest f(n) = g(n) + h(n) value, ensuring an optimal solution.

Q.3. Explain different heuristic functions that can be used for the eight puzzle problem.

→ (i) Misplaced tiles heuristic (h_1)
This heuristic counts the number of tiles that are not in their goal position.
It is simple and provides a basic measure on how far the current state is from the goal state.
(ii) Manhattan distance Heuristic (h_2)
This heuristic ~~abc~~ calculates the sum of the Manhattan distance between 2 points (x_1, y_1) and (x_2, y_2).

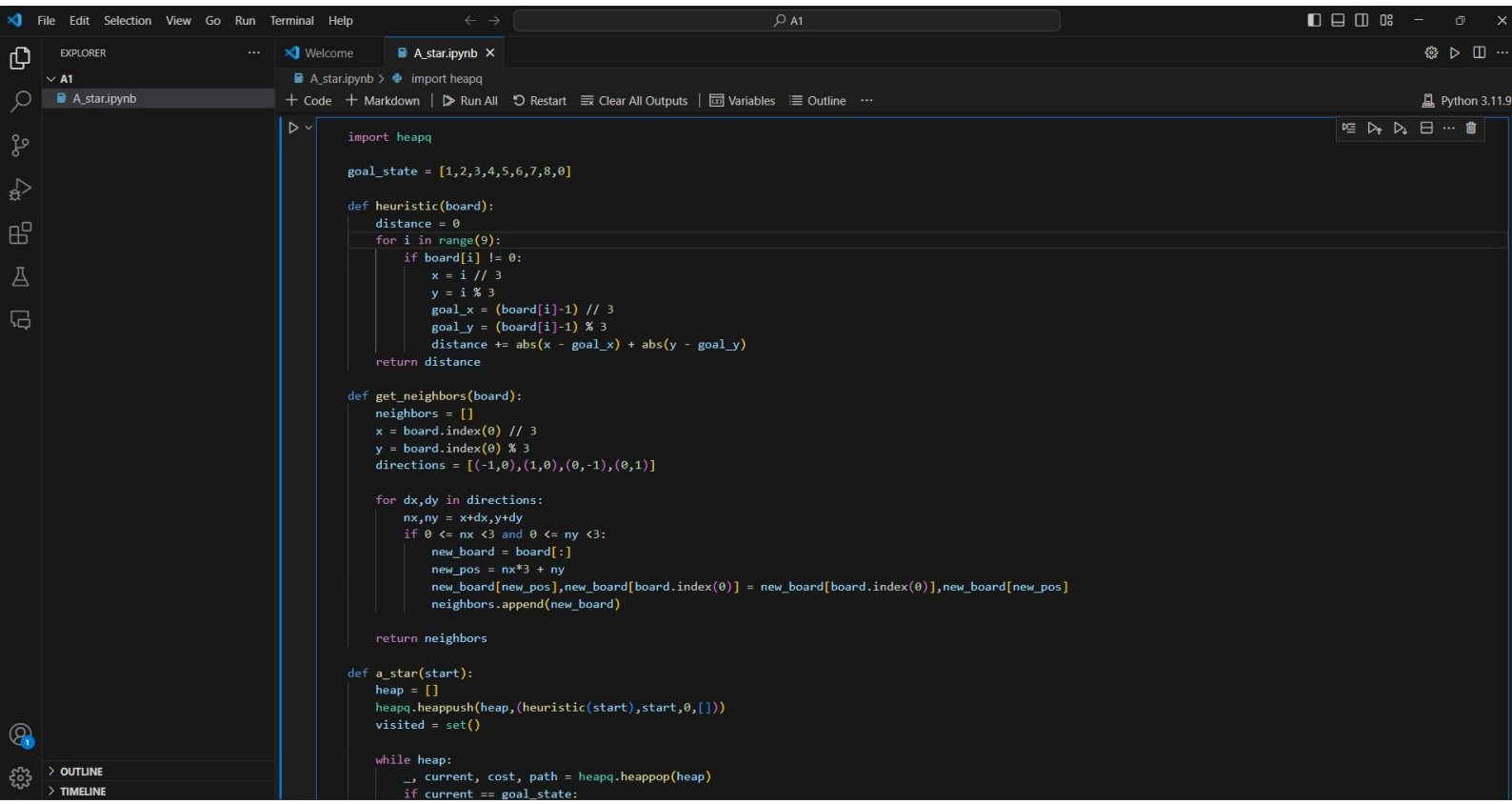
$$h_2(n) = \sum_{i=1}^{8} (|x_i - x_{goal}| + |y_i - y_{goal}|)$$

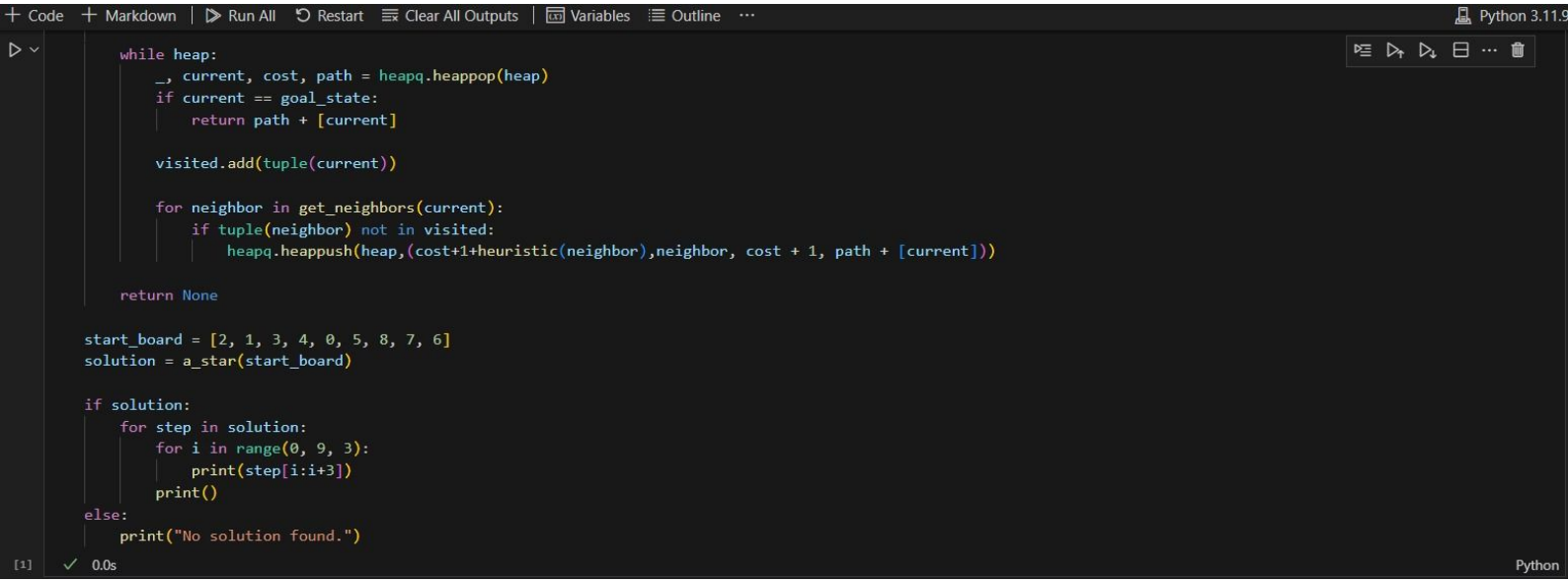
(iii) Linear conflict Heuristic
This heuristic is an extension of Manhattan heuristic. $h_3(n)$: Manhattan distance + 2 × (No. of linear conflict)

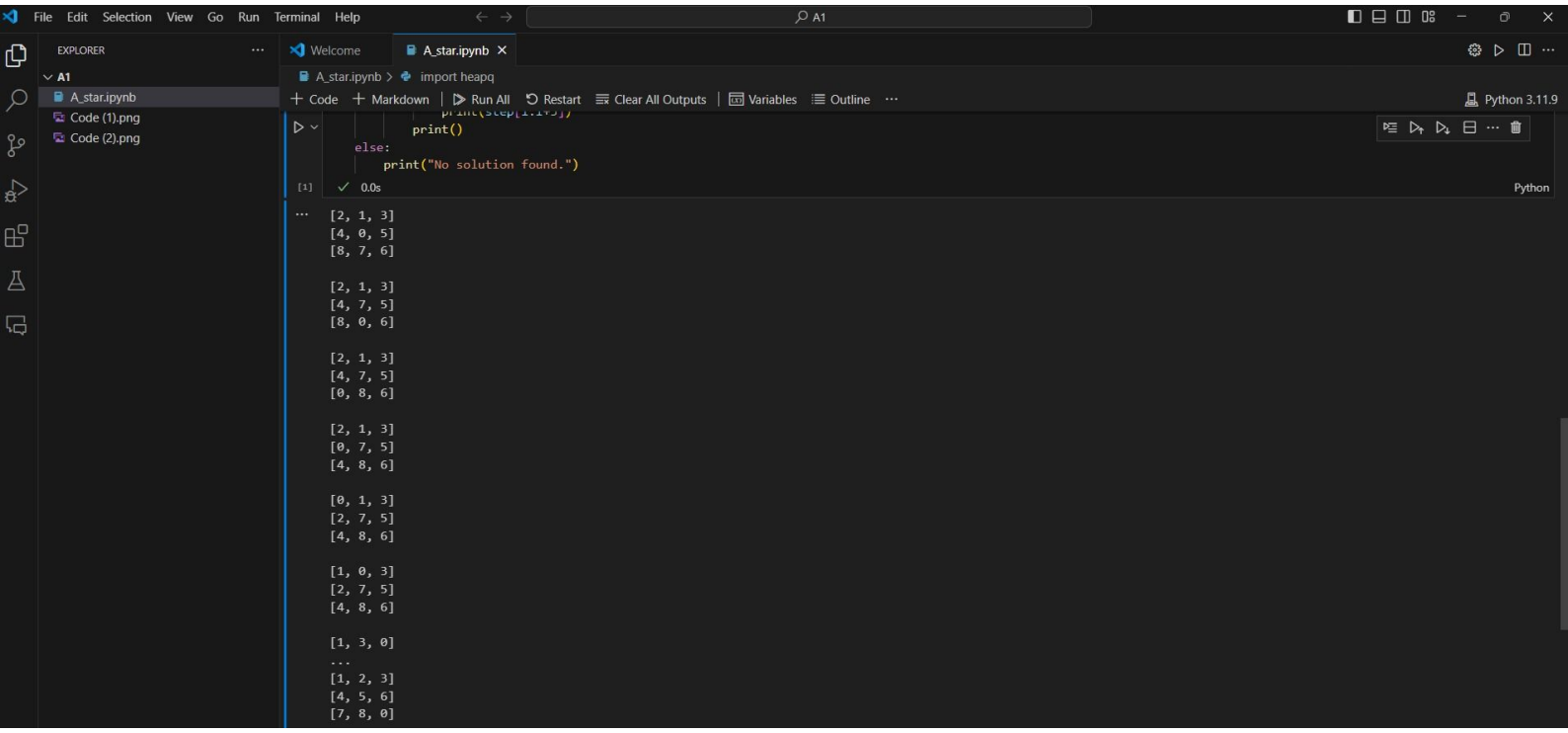
-> A* Algorithm

```
function AStar (start, goal)
    openset = {start}
    camefrom = { }

    while openset is not empty
        current = node in openset with
                  lowest fscore [current]
    if current == goal
        return reconstruct_path (camefrom,
                                 current)

    openset.remove (current)

    for each neighbour in current
        tentative_score : gscore [current] +
                          dist_between (current,
                                        neighbour)

    function reconstruct_path (camefrom,
                               current)

        total_path = [current]
        while (current in camefrom):
            current = camefrom (current)
        return total_path
```

* Conclusion :- We explored search strategies including uninformed and informed search.


File Edit Selection View Go Run Terminal Help
A1
EXPLORER
A1
A_star.ipynb
Welcome
A_star.ipynb
A_star.ipynb > import heapq
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.11.9
import heapq
goal_state = [1,2,3,4,5,6,7,8,0]
def heuristic(board):
distance = 0
for i in range(9):
if board[i] != 0:
x = i // 3
y = i % 3
goal_x = (board[i]-1) // 3
goal_y = (board[i]-1) % 3
distance += abs(x - goal_x) + abs(y - goal_y)
return distance
def get_neighbors(board):
neighbors = []
x = board.index(0) // 3
y = board.index(0) % 3
directions = [(-1,0),(1,0),(0,-1),(0,1)]
for dx,dy in directions:
nx,ny = x+dx,y+dy
if 0 <= nx <3 and 0 <= ny <3:
new_board = board[:]
new_pos = nx*3 + ny
new_board[new_pos],new_board[board.index(0)] = new_board[board.index(0)],new_board[new_pos]
neighbors.append(new_board)
return neighbors
def a_star(start):
heap = []
heapq.heappush(heap,(heuristic(start),start,0,[]))
visited = set()
while heap:
_, current, cost, path = heapq.heappop(heap)
if current == goal_state:
OUTLINE
TIMELINE


Code
Markdown
Run All
Restart
Clear All Outputs
Variables
Outline
Python 3.11.9
while heap:
_, current, cost, path = heapq.heappop(heap)
if current == goal_state:
return path + [current]
visited.add(tuple(current))
for neighbor in get_neighbors(current):
if tuple(neighbor) not in visited:
heapq.heappush(heap,(cost+1+heuristic(neighbor),neighbor, cost + 1, path + [current]))
return None
start_board = [2, 1, 3, 4, 0, 5, 8, 7, 6]
solution = a_star(start_board)
if solution:
for step in solution:
for i in range(0, 9, 3):
print(step[i:i+3])
print()
else:
print("No solution found.")
[1]
0.0s
Python


File Edit Selection View Go Run Terminal Help
A1
EXPLORER
A1
A_star.ipynb
Code (1).png
Code (2).png
Welcome
A_star.ipynb
A_star.ipynb > import heapq
Code Markdown Run All Restart Clear All Outputs Variables Outline
Python 3.11.9
print()
else:
print("No solution found.")
[1] 0.0s
Python
[2, 1, 3]
[4, 0, 5]
[8, 7, 6]
[2, 1, 3]
[4, 7, 5]
[8, 0, 6]
[2, 1, 3]
[4, 7, 5]
[0, 8, 6]
[2, 1, 3]
[0, 7, 5]
[4, 8, 6]
[0, 1, 3]
[2, 7, 5]
[4, 8, 6]
[1, 0, 3]
[2, 7, 5]
[4, 8, 6]
[1, 3, 0]
...
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]