vaccand8puzzle

November 9, 2024

[3]:

print("Name:Vismay Pawar N","USN:1BM22CS331",sep="**\n**")

**def** vacuum\_world():

*# Initializing goal\_state*

*# 0 indicates Clean and 1 indicates Dirty*

goal\_state = {'A': '0', 'B': '0'}

cost = 0

location\_input = input("Enter Location of Vacuum (A or B): ").strip().

↪upper() *# User input for vacuum location*

status\_input = input(f"Enter status of **{**location\_input**}** (0 for Clean, 1 for␣

↪Dirty): ").strip() *# Status of the current location*

other\_location = 'B' **if** location\_input == 'A' **else** 'A' status\_input\_complement = input(f"Enter status of **{**other\_location**}** (0 for␣

↪Clean, 1 for Dirty): ").strip() *# Status of the other room*

print("Initial Location Condition: " + str(goal\_state))

*# Helper function to clean a location*

**def** clean(location):

**nonlocal** cost goal\_state[location] = '0'

cost += 1 *# Cost for sucking dirt*

print(f"Location **{**location**}** has been Cleaned. Cost: **{**cost**}**")

*# Main logic*

**if** location\_input == 'A':

print("Vacuum is placed in Location A.")

**if** status\_input == '1': print("Location A is Dirty.") clean('A')

**if** status\_input\_complement == '1': print("Location B is Dirty.") print("Moving right to Location B.") cost += 1 *# Cost for moving right* print(f"COST for moving RIGHT: **{**cost**}**") clean('B')

# else:

**else**:

print("Location B is already clean.")

print("Location A is already clean.")

**if** status\_input\_complement == '1': print("Location B is Dirty.") print("Moving right to Location B.") cost += 1 *# Cost for moving right* print(f"COST for moving RIGHT: **{**cost**}**") clean('B')

# else:

print("Location B is already clean.")

**else**: *# Vacuum is placed in Location B* print("Vacuum is placed in Location B.") **if** status\_input == '1':

print("Location B is Dirty.") clean('B')

**if** status\_input\_complement == '1': print("Location A is Dirty.") print("Moving left to Location A.") cost += 1 *# Cost for moving left* print(f"COST for moving LEFT: **{**cost**}**") clean('A')

# else:

print("Location A is already clean.")

# else:

print("Location B is already clean.")

**if** status\_input\_complement == '1': print("Location A is Dirty.") print("Moving left to Location A.") cost += 1 *# Cost for moving left* print(f"COST for moving LEFT: **{**cost**}**") clean('A')

# else:

print("Location A is already clean.")

*# Done cleaning*

print("GOAL STATE: ")

print(goal\_state)

print("Performance Measurement: " + str(cost))

*# Output*

vacuum\_world()

Name:Vismay Pawar N USN:1BM22CS331

Enter Location of Vacuum (A or B): B

[4]:

Enter status of B (0 for Clean, 1 for Dirty): 1 Enter status of A (0 for Clean, 1 for Dirty): 1 Initial Location Condition: {'A': '0', 'B': '0'} Vacuum is placed in Location B.

Location B is Dirty.

Location B has been Cleaned. Cost: 1 Location A is Dirty.

Moving left to Location A. COST for moving LEFT: 2

Location A has been Cleaned. Cost: 3 GOAL STATE:

{'A': '0', 'B': '0'}

Performance Measurement: 3

*# 8 puzzle problem using BFS technique*

print("Name:Vismay Pawar N","USN:1BM22CS331",sep="**\n**")

**from collections import** deque

**def** solve\_8puzzle\_bfs(initial\_state):

*"""*

*Solves the 8-puzzle using Breadth-First Search.*

*Args:*

*initial\_state: A list of lists representing the initial state of the*␣

↪*puzzle.*

*Returns:*

*A list of lists representing the solution path, or None if no solution*␣

↪*is found.*

*"""*

**def** find\_blank(state):

*"""Finds the row and column of the blank tile."""*

**for** row **in** range(3):

**for** col **in** range(3):

**if** state[row][col] == 0:

**return** row, col

**def** get\_neighbors(state):

*"""Generates possible neighbor states by moving the blank tile."""*

row, col = find\_blank(state) neighbors = []

**if** row > 0:

new\_state = [row[:] **for** row **in** state]

new\_state[row][col], new\_state[row - 1][col] = new\_state[row -␣

↪1][col], new\_state[row][col]

neighbors.append(new\_state)

**if** row < 2:

new\_state = [row[:] **for** row **in** state]

new\_state[row][col], new\_state[row + 1][col] = new\_state[row +␣

↪1][col], new\_state[row][col]

neighbors.append(new\_state)

**if** col > 0:

new\_state = [row[:] **for** row **in** state]

new\_state[row][col], new\_state[row][col - 1] = new\_state[row][col -␣

↪1], new\_state[row][col]

neighbors.append(new\_state)

**if** col < 2:

new\_state = [row[:] **for** row **in** state]

new\_state[row][col], new\_state[row][col + 1] = new\_state[row][col +␣

↪1], new\_state[row][col]

neighbors.append(new\_state)

**return** neighbors

goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

queue = deque([(initial\_state, [])]) visited = set()

**while** queue:

current\_state, path = queue.popleft()

**if** current\_state == goal\_state:

**return** path + [current\_state]

visited.add(tuple(map(tuple, current\_state)))

**for** neighbor **in** get\_neighbors(current\_state):

**if** tuple(map(tuple, neighbor)) **not in** visited: queue.append((neighbor, path + [current\_state]))

**return None** *# No solution found # Example usage:*

initial\_state = [[1, 2, 3], [4, 0, 6], [7, 5, 8]]

solution = solve\_8puzzle\_bfs(initial\_state)

**if** solution:

print("Solution found:")

**for** state **in** solution:

**for** row **in** state: print(row)

print()

# else:

print("No solution found.")

Name:Vismay Pawar N USN:1BM22CS331

Solution found: [1, 2, 3]

[4, 0, 6]

[7, 5, 8]

[1, 2, 3]

[4, 5, 6]

[7, 0, 8]

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]

[5]:

*# 8 puzzle problem using DFS technique*

print("Name:Vismay pawar N","USN:1BM22CS331",sep="**\n**")

**from collections import** deque

**def** solve\_8puzzle\_dfs(initial\_state):

*"""*

*Solves the 8-puzzle using Depth-First Search.*

*Args:*

*initial\_state: A list of lists representing the initial state of the*␣

↪*puzzle.*

*Returns:*

*A list of lists representing the solution path, or None if no solution*␣

↪*is found.*

*"""*

**def** find\_blank(state):

*"""Finds the row and column of the blank tile."""*

**for** row **in** range(3):

**for** col **in** range(3):

**if** state[row][col] == 0:

**return** row, col

**def** get\_neighbors(state):

*"""Generates possible neighbor states by moving the blank tile."""*

row, col = find\_blank(state) neighbors = []

directions = [(-1, 0), (1, 0), (0, -1), (0, 1)] *# Up, Down, Left, Right*

**for** dr, dc **in** directions:

new\_row, new\_col = row + dr, col + dc

**if** 0 <= new\_row < 3 **and** 0 <= new\_col < 3: new\_state = [r[:] **for** r **in** state]

new\_state[row][col], new\_state[new\_row][new\_col] =␣

↪new\_state[new\_row][new\_col], new\_state[row][col] neighbors.append(new\_state)

**return** neighbors

goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]

stack = [(initial\_state, [])] visited = set()

**while** stack:

current\_state, path = stack.pop()

state\_tuple = tuple(map(tuple, current\_state)) *# Convert to tuple for*␣

↪*set*

**if** state\_tuple **in** visited:

**continue**

visited.add(state\_tuple)

**if** current\_state == goal\_state:

**return** path + [current\_state]

**for** neighbor **in** get\_neighbors(current\_state): stack.append((neighbor, path + [current\_state]))

**return None** *# No solution found # Example usage:*

initial\_state = [[1, 2, 3], [4, 5, 6], [0, 7, 8]]

solution = solve\_8puzzle\_dfs(initial\_state)

**if** solution:

print("Solution found:")

**for** state **in** solution:

**for** row **in** state: print(row)

print()

**else**:

print("No solution found.")

Name:Vismay Pawar N USN:1BM22CS331

Solution found: [1, 2, 3]

[4, 5, 6]

[0, 7, 8]

[1, 2, 3]

[4, 5, 6]

[7, 0, 8]

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]