AI LAB PROGRAMS

1. Write the python program to solve 8-Puzzle problem
2. from heapq import heappop, heappush

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

def manhattan(p): # Manhattan distance

dist = 0

for i in range(3):

for j in range(3):

val = p[i][j]

if val and val != 0:

x, y = divmod(val-1, 3)

dist += abs(i - x) + abs(j - y)

return dist

def neighbors(p):

i, j = [(ix, iy) for ix, row in enumerate(p) for iy, val in enumerate(row) if val == 0][0]

dirs = [(-1,0),(1,0),(0,-1),(0,1)]

for dx, dy in dirs:

x, y = i+dx, j+dy

if 0 <= x < 3 and 0 <= y < 3:

new\_p = [row[:] for row in p]

new\_p[i][j], new\_p[x][y] = new\_p[x][y], new\_p[i][j]

yield new\_p

def astar(start):

seen = set()

heap = [(manhattan(start), 0, start, [])]

while heap:

est, cost, state, path = heappop(heap)

fs = tuple(map(tuple, state))

if fs in seen:

continue

seen.add(fs)

if state == goal:

return path + [state]

for n in neighbors(state):

heappush(heap, (cost + 1 + manhattan(n), cost + 1, n, path + [state]))

# Example usage

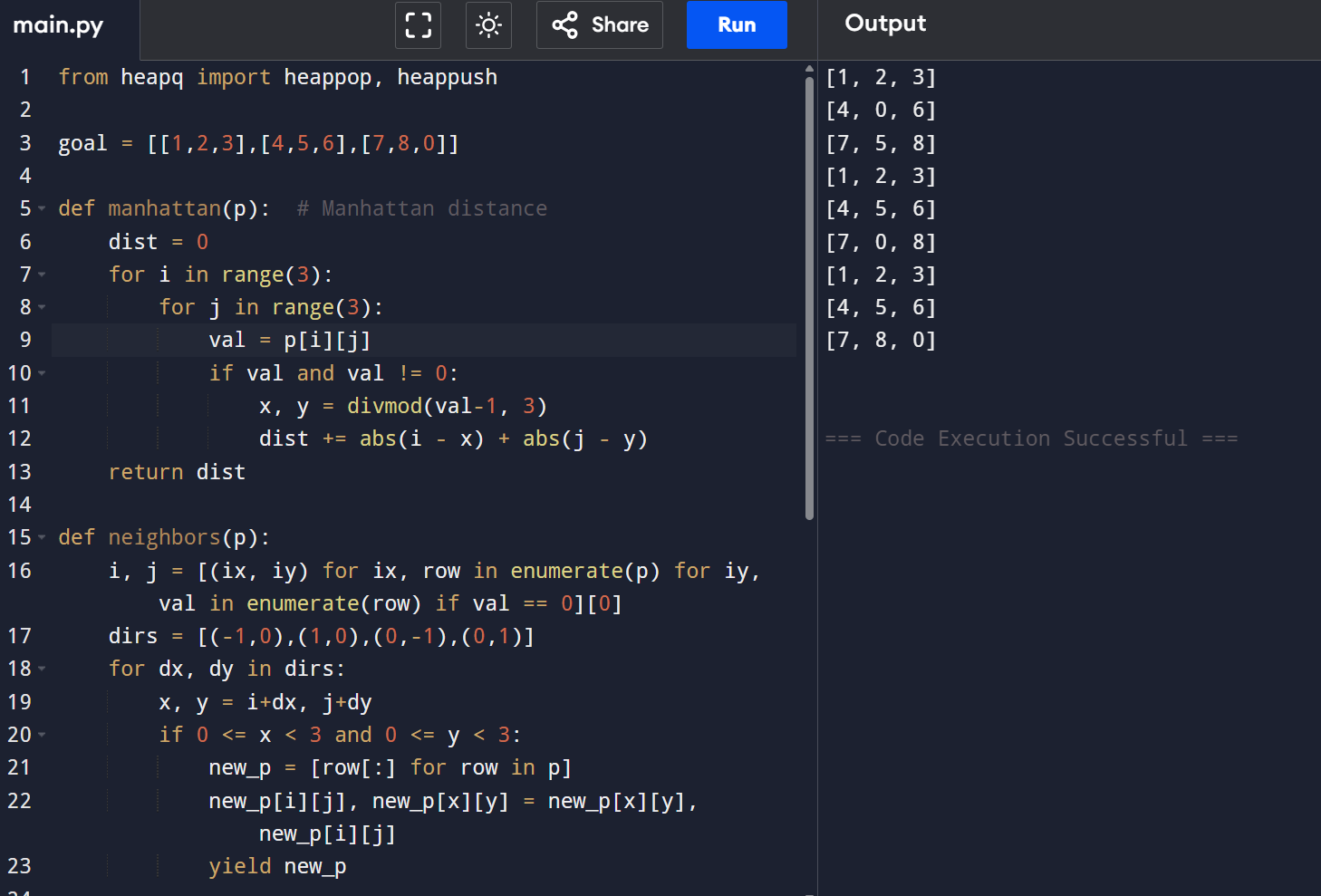
start = [[1,2,3],[4,0,6],[7,5,8]]

solution = astar(start)

for step in solution:

for row in step: print(row)

print()



1. Write the python program to solve 8-Queen problem

def is\_safe(board, row, col):

for i in range(col):

if board[row][i] == 1:

return False

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

for i, j in zip(range(row, len(board)), range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solve\_n\_queens\_util(board, col):

if col >= len(board):

return True

for i in range(len(board)):

if is\_safe(board, i, col):

board[i][col] = 1

if solve\_n\_queens\_util(board, col + 1):

return True

board[i][col] = 0 # Backtrack

return False

def solve\_n\_queens(n):

board = [[0] \* n for \_ in range(n)]

if not solve\_n\_queens\_util(board, 0):

return "No solution exists"

return board

# Example usage

n = 8

solution = solve\_n\_queens(n)

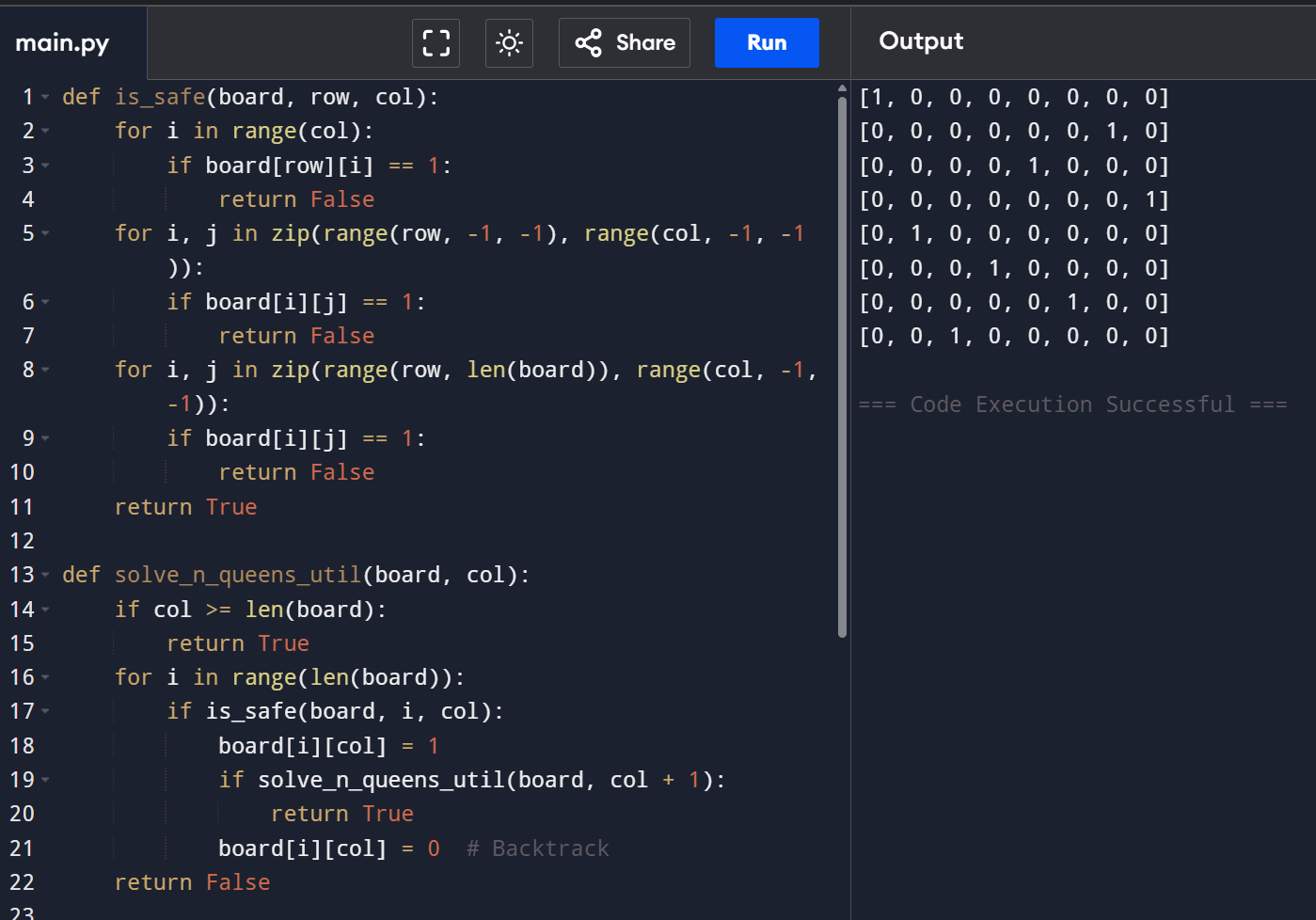
if isinstance(solution, str):

print(solution)

else:

for row in solution:

print(row)



3. Write the python program for Water Jug Problem

from collections import deque

def water\_jug\_bfs(jug1, jug2, target):

visited = set()

queue = deque([(0, 0)])

while queue:

a, b = queue.popleft()

if (a, b) in visited:

continue

visited.add((a, b))

if a == target or b == target:

print(f"Solution found: Jug1 = {a}, Jug2 = {b}")

return True

queue.extend([

(jug1, b), (a, jug2), (0, b), (a, 0),

(min(jug1, a + b), max(0, a + b - jug1)),

(max(0, a + b - jug2), min(jug2, a + b))

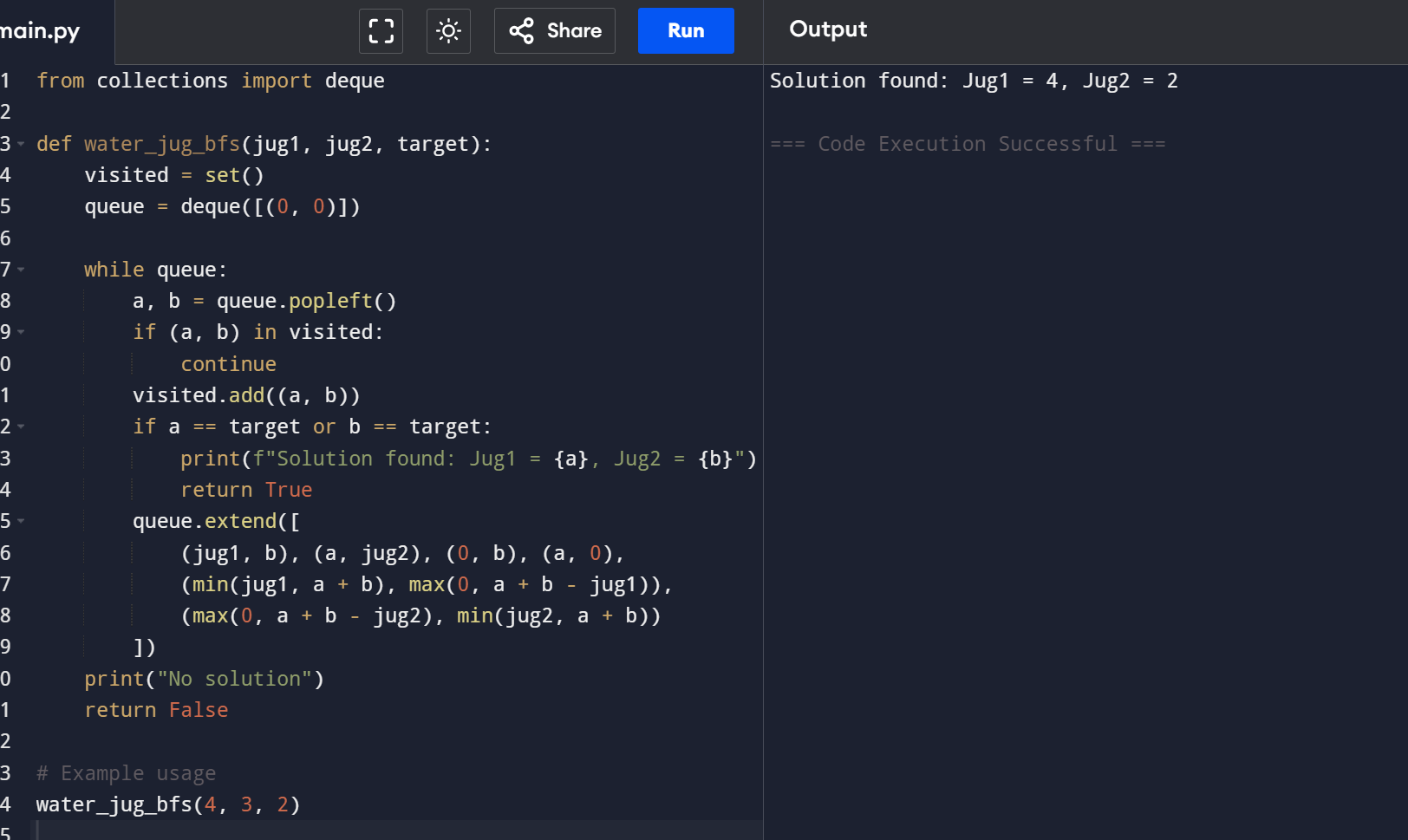
])

print("No solution")

return False

# Example usage

water\_jug\_bfs(4, 3, 2)



1. Write the python program for Cript-Arithmetic problem

import itertools

def solve\_cryptarithmetic():

letters = 'SENDMORY'

digits = range(10)

for perm in itertools.permutations(digits, len(letters)):

s, e, n, d, m, o, r, y = perm

if s == 0 or m == 0:

continue # No leading zero allowed

send = 1000 \* s + 100 \* e + 10 \* n + d

more = 1000 \* m + 100 \* o + 10 \* r + e

money = 10000 \* m + 1000 \* o + 100 \* n + 10 \* e + y

if send + more == money:

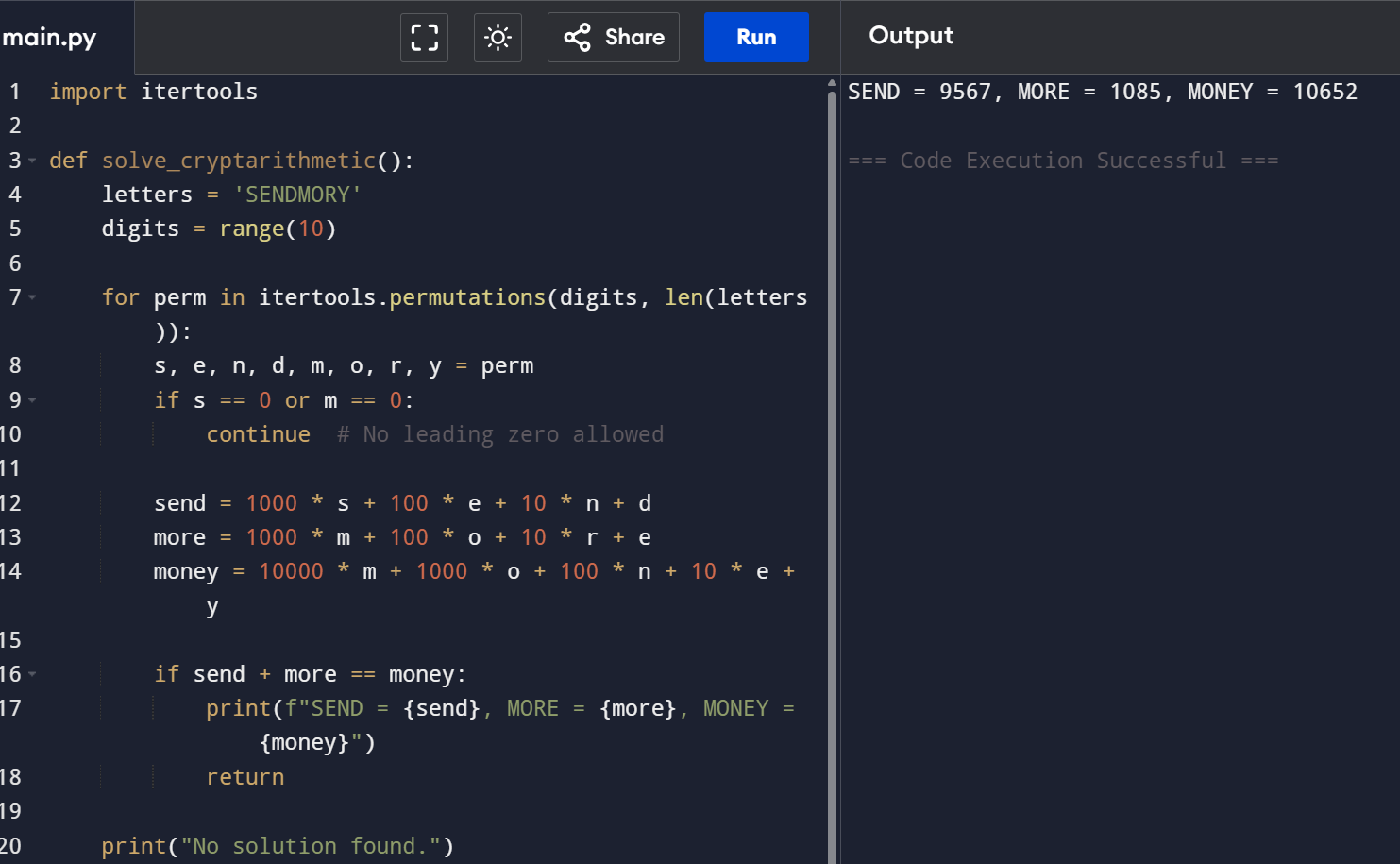
print(f"SEND = {send}, MORE = {more}, MONEY = {money}")

return

print("No solution found.")

# Run the solver

solve\_cryptarithmetic()



5 Write the python program for Missionaries Cannibal problem

from collections import deque

def is\_valid(m, c):

return (m == 0 or m >= c) and (3 - m == 0 or 3 - m >= 3 - c)

def missionaries\_and\_cannibals():

visited = set()

queue = deque([((3, 3, 1), [])]) # (M, C, Boat\_side), Path

moves = [(1,0), (2,0), (0,1), (0,2), (1,1)]

while queue:

(m, c, b), path = queue.popleft()

if (m, c, b) in visited: continue

visited.add((m, c, b))

path = path + [((m, c, b))]

if m == 0 and c == 0 and b == 0:

for step in path: print(step)

return True

for dm, dc in moves:

nm, nc, nb = (m - dm, c - dc, 0) if b else (m + dm, c + dc, 1)

if 0 <= nm <= 3 and 0 <= nc <= 3 and is\_valid(nm, nc):

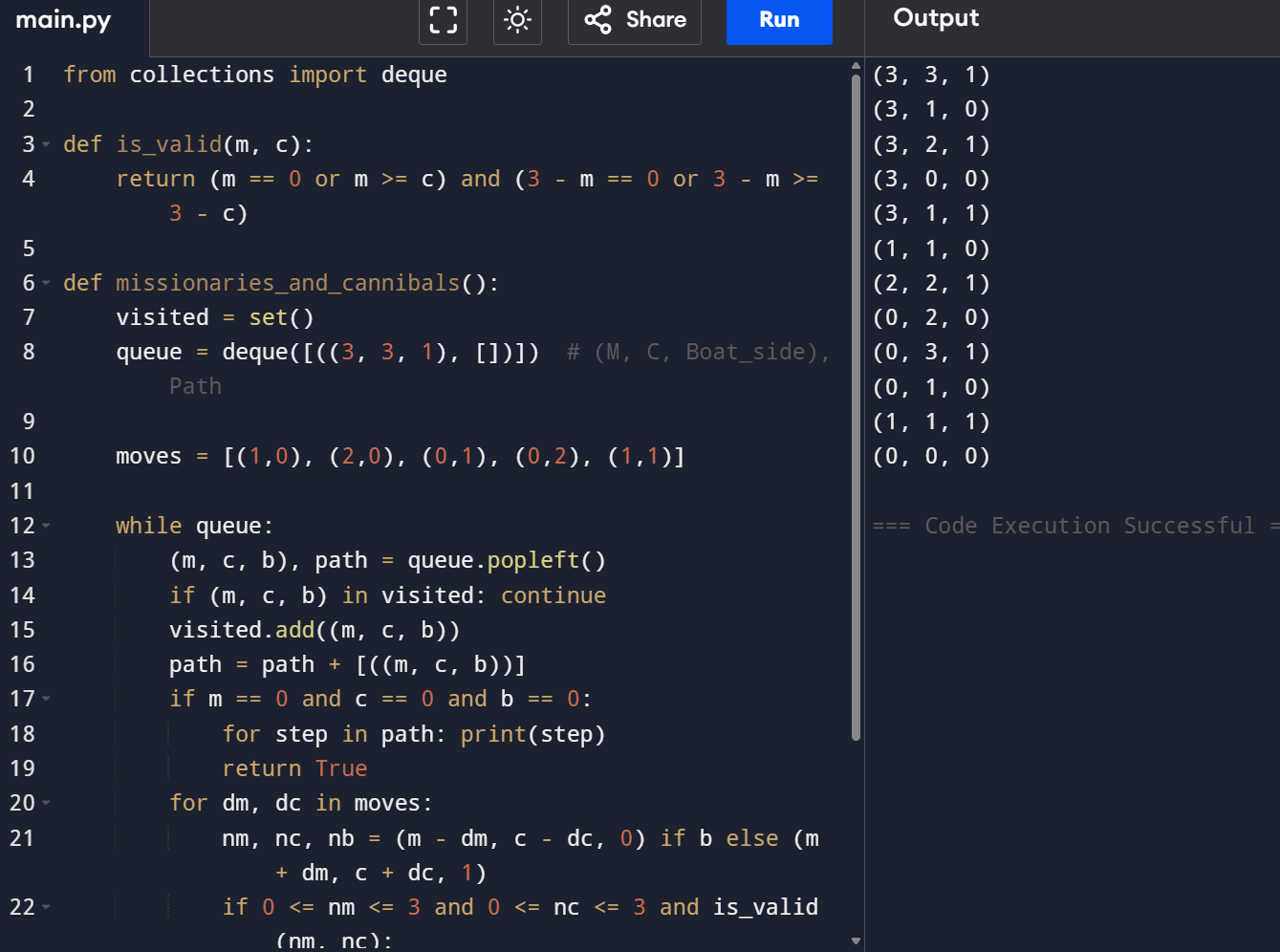
queue.append(((nm, nc, nb), path))

print("No solution")

return False

# Run the function

missionaries\_and\_cannibals()



6.Write the python program for Vacuum Cleaner problem

class VacuumCleaner:

def \_\_init\_\_(self, grid\_size):

self.grid\_size = grid\_size

self.grid = [[0 for \_ in range(grid\_size)] for \_ in range(grid\_size)]

self.position = [0, 0] # Starting position (top-left corner)

def place\_dirt(self, x, y):

"""Place dirt at the specified position"""

if 0 <= x < self.grid\_size and 0 <= y < self.grid\_size:

self.grid[x][y] = 1

def is\_dirty(self, x, y):

"""Check if current position is dirty"""

return self.grid[x][y] == 1

def clean\_current\_position(self):

"""Clean the current position"""

self.grid[self.position[0]][self.position[1]] = 0

def move(self, direction):

"""Move the vacuum cleaner in the specified direction"""

x, y = self.position

if direction == "up" and x > 0:

self.position[0] -= 1

elif direction == "down" and x < self.grid\_size - 1:

self.position[0] += 1

elif direction == "left" and y > 0:

self.position[1] -= 1

elif direction == "right" and y < self.grid\_size - 1:

self.position[1] += 1

def clean\_room(self):

"""Main cleaning algorithm"""

moves = 0

# Continue until all dirt is cleaned

while any(1 in row for row in self.grid):

x, y = self.position

# Clean current position if dirty

if self.is\_dirty(x, y):

self.clean\_current\_position()

print(f"Cleaned position ({x}, {y})")

# Move to next position

if y < self.grid\_size - 1:

self.move("right")

elif x < self.grid\_size - 1:

self.move("down")

self.position[1] = 0 # Reset to leftmost position

moves += 1

return moves

# Example usage

def main():

# Create a 4x4 grid

vacuum = VacuumCleaner(4)

# Place some dirt

dirt\_positions = [(0, 1), (1, 3), (2, 2), (3, 0)]

for x, y in dirt\_positions:

vacuum.place\_dirt(x, y)

# Print initial state

print("Initial grid state:")

for row in vacuum.grid:

print(row)

# Clean the room

total\_moves = vacuum.clean\_room()

# Print final state

print("\nFinal grid state:")

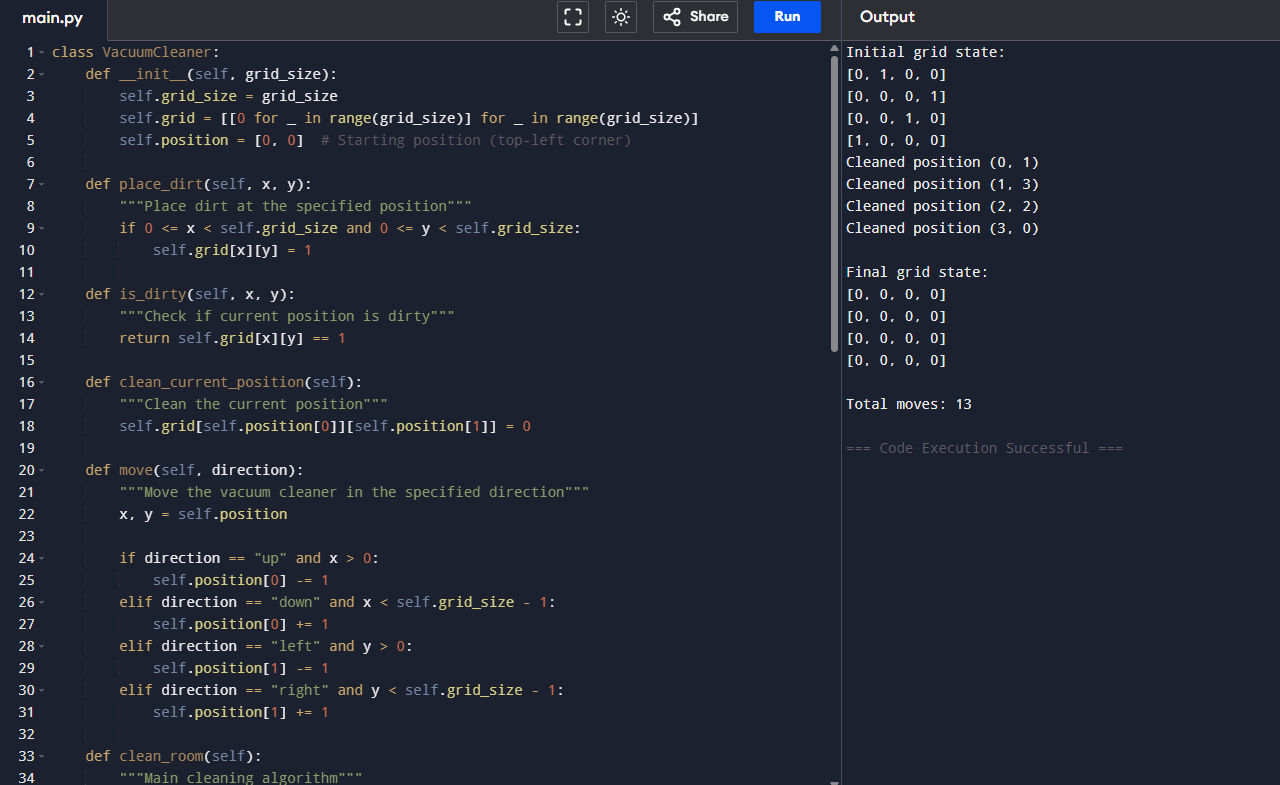
for row in vacuum.grid:

print(row)

print(f"\nTotal moves: {total\_moves}")

if \_\_name\_\_ == "\_\_main\_\_":

main()



7. Write the python program to implement BFS.

from collections import deque

# BFS function

def bfs(graph, start):

visited = set()

queue = deque([start])

while queue:

node = queue.popleft()

if node not in visited:

print(node, end=" ")

visited.add(node)

for neighbor in graph[node]:

if neighbor not in visited:

queue.append(neighbor)

# Example graph (Adjacency List)

graph = {

'A': ['B', 'C'],

'B': ['A', 'D', 'E'],

'C': ['A', 'F'],

'D': ['B'],

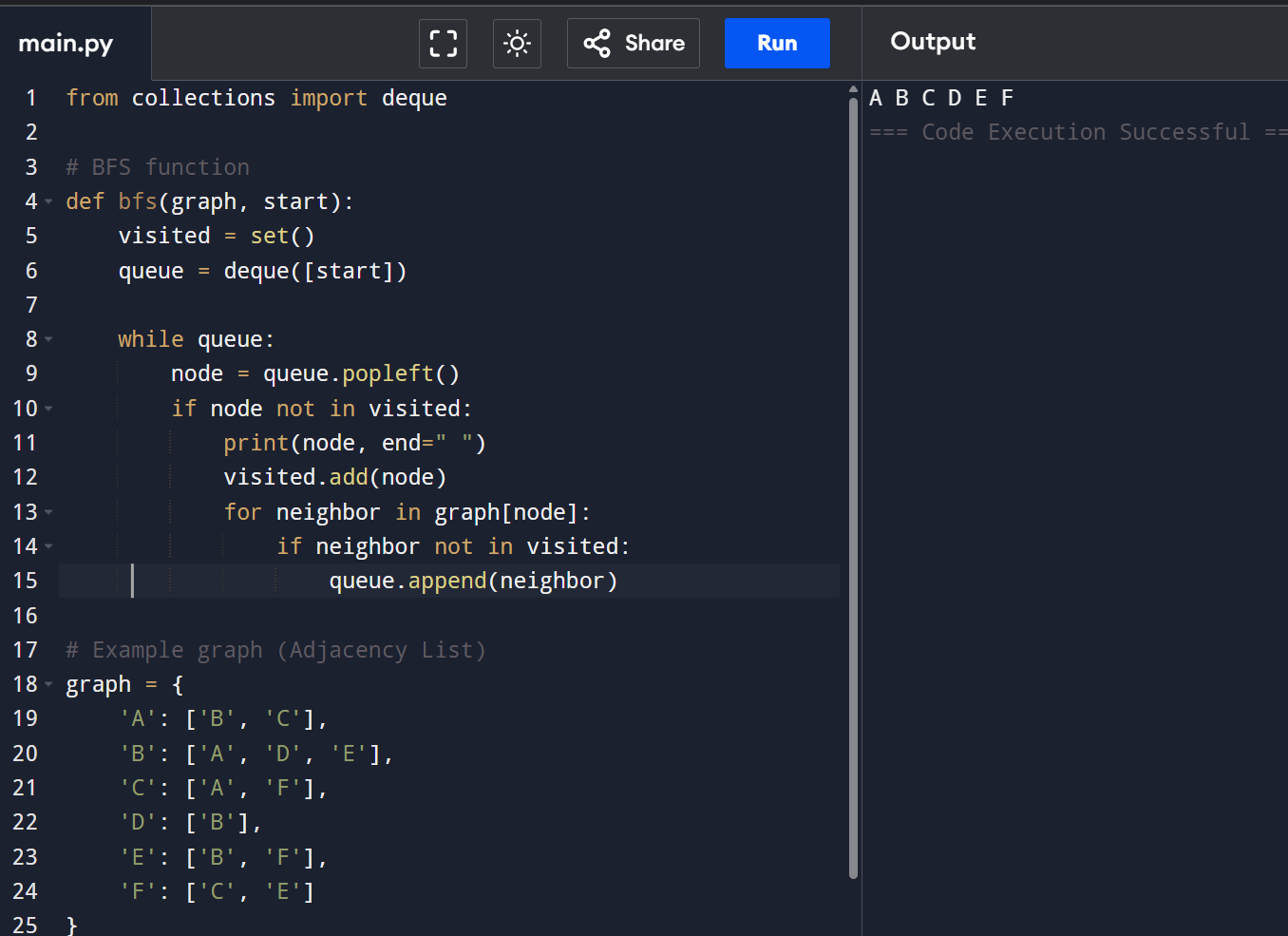
'E': ['B', 'F'],

'F': ['C', 'E']

}

# Run BFS

bfs(graph, 'A')



8 Write the python program to implement DFS.

# DFS function (recursive)

def dfs(graph, node, visited=None):

if visited is None:

visited = set()

if node not in visited:

print(node, end=" ")

visited.add(node)

for neighbor in graph[node]:

dfs(graph, neighbor, visited)

# Example graph (Adjacency List)

graph = {

'A': ['B', 'C'],

'B': ['D', 'E'],

'C': ['F'],

'D': [],

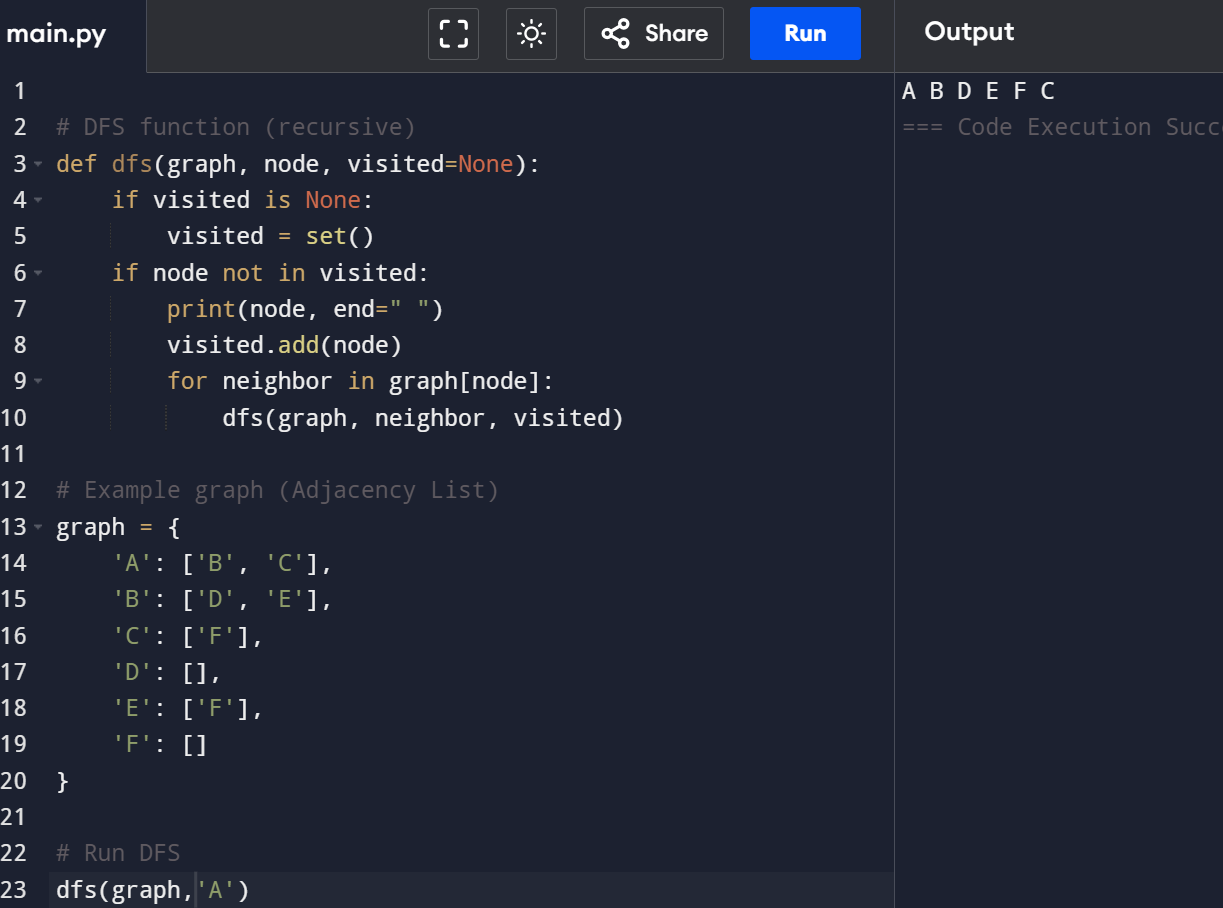
'E': ['F'],

'F': []

}

# Run DFS

dfs(graph,'A')



9. Write the python to implement Travelling Salesman Problem

import itertools

def tsp\_brute\_force(graph):

n = len(graph)

cities = list(range(n))

min\_path = None

min\_cost = float('inf')

for perm in itertools.permutations(cities[1:]):

path = [0] + list(perm) + [0]

cost = sum(graph[path[i]][path[i + 1]] for i in range(n))

if cost < min\_cost:

min\_cost = cost

min\_path = path

return min\_path, min\_cost

# Example usage: Distance matrix (symmetric)

graph = [

[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

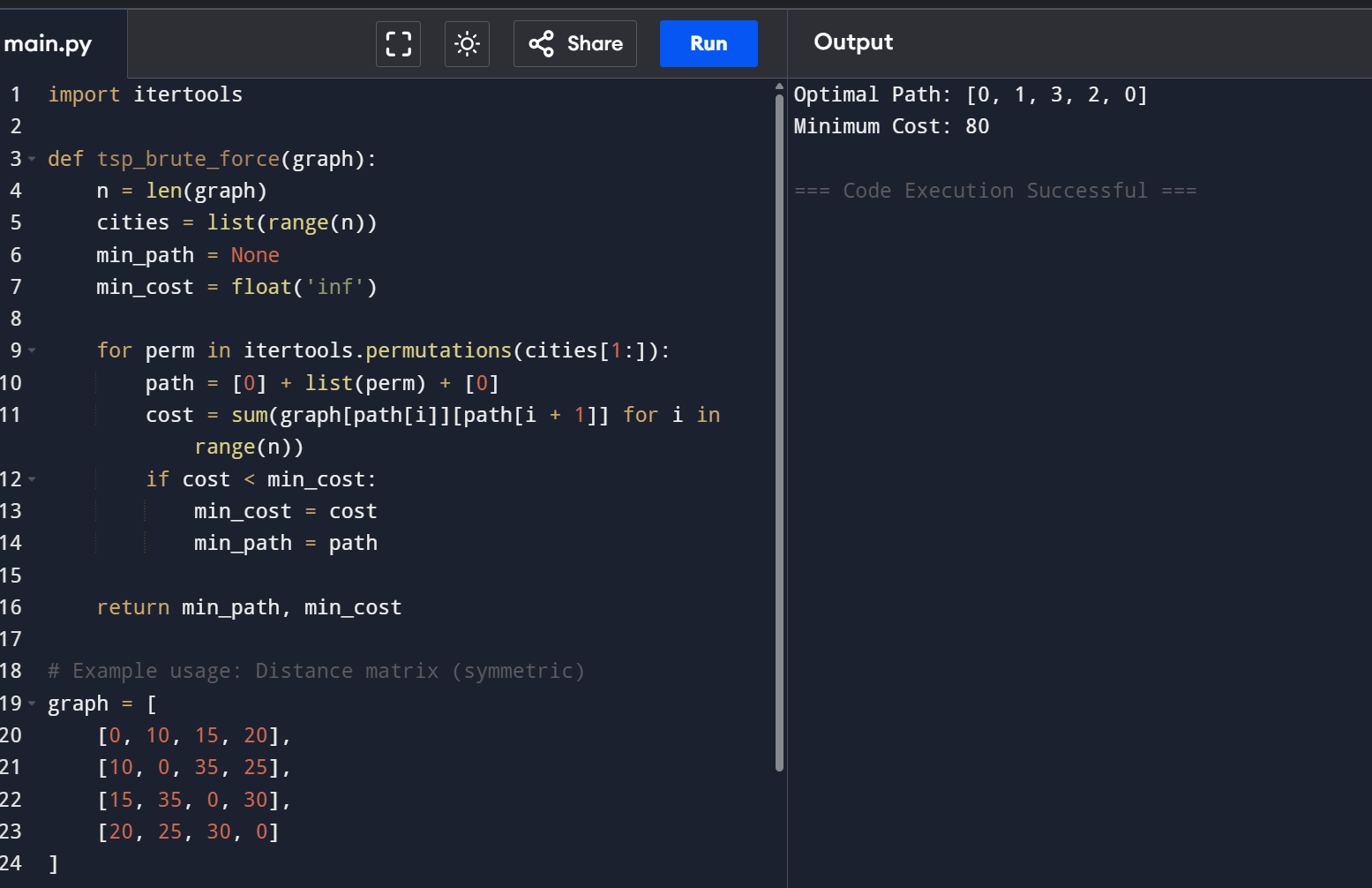
[20, 25, 30, 0]

]

path, cost = tsp\_brute\_force(graph)

print("Optimal Path:", path)

print("Minimum Cost:", cost)



10. Write the python program to implement A\* algorithm

import heapq

def a\_star(start, goal, graph, heuristic):

open\_set = []

heapq.heappush(open\_set, (0, start))

came\_from = {}

g\_score = {node: float('inf') for node in graph}

g\_score[start] = 0

while open\_set:

\_, current = heapq.heappop(open\_set)

if current == goal:

path = []

while current in came\_from:

path.append(current)

current = came\_from[current]

path.append(start)

return path[::-1]

for neighbor, cost in graph[current]:

tentative\_g = g\_score[current] + cost

if tentative\_g < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = tentative\_g

f\_score = tentative\_g + heuristic[neighbor]

heapq.heappush(open\_set, (f\_score, neighbor))

return None

# --- Example usage ---

graph = {

'A': [('B', 1), ('C', 4)],

'B': [('A', 1), ('C', 2), ('D', 5)],

'C': [('A', 4), ('B', 2), ('D', 1)],

'D': [('B', 5), ('C', 1)]

}

heuristic = {

'A': 7,

'B': 6,

'C': 2,

'D': 0 # Goal node

}

path = a\_star('A', 'D', graph, heuristic)

if path:

print("Shortest Path from A to D:", " → ".join(path))

else:

print("No path found.")

