AI LAB - Group-5

Md.Rafat Hossain Reyal-----Roll: 1910576122

Md.Alamgir Hosen ----------Roll: 1911176106

Mst.Sharmeen Akter---------Roll: 2012076155

Majid Bhuiyan----------Roll: 2010776123

Md.Tarik Bosunia----------Roll:2011176148

A* Search Algorithm Problem

```
import heapq

def a_star_search(graph, heuristic, start, goal):
    open_list = []  # Priority queue (min-heap)
    heapq.heappush(open_list, (heuristic[start], 0, start, []))  # (f, g, node, path)
    closed_list = set()

    print("Step-1:")

    while open_list:
        f, g, current, path = heapq.heappop(open_list)

        if current in closed_list:
            continue

        path = path + [current]
        closed_list.add(current)

        if current == goal:
            print(f"\nFinal Path: {' → '.join(path)} ; F(n) = {g}")
            return path

        for neighbor, cost in graph[current].items():
            if neighbor not in closed_list:
                g_new = g + cost
                f_new = g_new + heuristic[neighbor]
                heapq.heappush(open_list, (f_new, g_new, neighbor, path))

                print(f"S → {neighbor} ; F(n) = g(n) + h(n)")
                print(f"    = {g_new} + {heuristic[neighbor]} = {f_new}")
                print("(Hold)" if neighbor != goal else "")
                print()

    return None  # No path found

# Define the graph as an adjacency dictionary
graph = {
    'S': {'A': 1, 'G': 10},
    'A': {'C': 1, 'B': 2},
    'B': {'D': 5},
    'C': {'G': 4, 'D': 3},
    'D': {'G': 2},
    'G': {}
}

# Define heuristic values
heuristic = {
    'S': 5, 'A': 3, 'B': 4, 'C': 2, 'D': 6, 'G': 0
}

# Run A* Search
start = 'S'
goal = 'G'
a_star_search(graph, heuristic, start, goal)
```

```
Step-1:
S → A ; F(n) = g(n) + h(n)
    = 1 + 3 = 4
```

```
(Hold)

S → G ; F(n) = g(n) + h(n)
    = 10 + 0 = 10


S → C ; F(n) = g(n) + h(n)
    = 2 + 2 = 4
(Hold)

S → B ; F(n) = g(n) + h(n)
    = 3 + 4 = 7
(Hold)

S → G ; F(n) = g(n) + h(n)
    = 6 + 0 = 6


S → D ; F(n) = g(n) + h(n)
    = 5 + 6 = 11
(Hold)


Final Path: S → A → C → G ; F(n) = 6
['S', 'A', 'C', 'G']
```

Best-First Search (BFS) Problem

```
import heapq
import time

def best_first_search(graph, heuristic, start, goal):
    open_list = []
    heapq.heappush(open_list, (heuristic[start], start))
    closed_list = []
    parent = {start: None}

    print("\nStarting Best-First Search:\n")
    print_lists(open_list, closed_list)

    while open_list:
        _, current = heapq.heappop(open_list)

        if current in closed_list:
            continue

        closed_list.append(current)
        print_lists(open_list, closed_list)
        time.sleep(1)  # Adding delay for better visualization

        if current == goal:
            path = []
            while current:
                path.append(current)
                current = parent[current]
            print("\nGoal found! Final Path:")
            print(" → ".join(path[::-1]))
            return path[::-1]

        for neighbor in graph[current]:
            if neighbor not in closed_list:
                heapq.heappush(open_list, (heuristic[neighbor], neighbor))
                parent[neighbor] = current

    print("\nNo path found.")
    return None

def print_lists(open_list, closed_list):
    open_nodes = [node for _, node in sorted(open_list)]
    print(f"OPEN: {open_nodes}, CLOSED: {closed_list}")

# Define the graph
graph = {
    'S': ['A', 'B'],
    'A': ['C', 'D'],
    'B': ['E', 'F'],
    'C': [],
```

```
    'D': [],
    'E': [],
    'F': ['H', 'I', 'G'],
    'H': [],
    'I': [],
    'G': []
}

# Define heuristic values
heuristic = {
    'S': 14, 'A': 12, 'B': 5, 'C': 7, 'D': 3, 'E': 8, 'F': 2,
    'H': 4, 'I': 9, 'G': 0
}

# Run the algorithm
start = 'S'
goal = 'G'
best_first_search(graph, heuristic, start, goal)
```

```
Starting Best-First Search:

OPEN: ['S'], CLOSED: []
OPEN: [], CLOSED: ['S']
OPEN: ['A'], CLOSED: ['S', 'B']
OPEN: ['E', 'A'], CLOSED: ['S', 'B', 'F']
OPEN: ['H', 'E', 'I', 'A'], CLOSED: ['S', 'B', 'F', 'G']

Goal found! Final Path:
S → B → F → G
['S', 'B', 'F', 'G']
```

Heuristics search Problem

```
import heapq
import numpy as np

class Puzzle:
    def __init__(self, state, parent=None, move=None, depth=0, cost=0):
        self.state = state
        self.parent = parent
        self.move = move
        self.depth = depth
        self.cost = cost  # f(n) = g(n) + h(n)

    def __lt__(self, other):
        return self.cost < other.cost

def manhattan_distance(state, goal):
    distance = 0
    for num in range(1, 9):  # Exclude zero
        x1, y1 = np.where(state == num)
        x2, y2 = np.where(goal == num)
        distance += abs(x1 - x2) + abs(y1 - y2)
    return distance[0]

def get_neighbors(state):
    x, y = np.where(state == 0)
    x, y = int(x), int(y)
    moves = []
    directions = {'Up': (x - 1, y), 'Down': (x + 1, y), 'Left': (x, y - 1), 'Right': (x, y + 1)}
    for move, (nx, ny) in directions.items():
        if 0 <= nx < 3 and 0 <= ny < 3:
            new_state = state.copy()
            new_state[x, y], new_state[nx, ny] = new_state[nx, ny], new_state[x, y]
            moves.append((new_state, move))
    return moves

def reconstruct_path(node):
    path = []
    while node:
        path.append((node.move, node.state))
        node = node.parent
    return path[::-1]
```

```
def a_star(start, goal):
    start_state = np.array(start)
    goal_state = np.array(goal)
    open_list = []
    heapq.heappush(open_list, Puzzle(start_state, cost=manhattan_distance(start_state, goal_state)))
    closed_set = set()
    step = 0

    print(f"Start State:\n{start_state}\n")

    while open_list:
        node = heapq.heappop(open_list)
        if np.array_equal(node.state, goal_state):
            return reconstruct_path(node)

        closed_set.add(tuple(map(tuple, node.state)))

        neighbors = get_neighbors(node.state)
        step += 1
        print(f"Step {step} :")
        print(f"In Step {step - 1} least f(n) is:\n{node.state}\nNow expanding this below:")

        children = []
        for new_state, move in neighbors:
            if tuple(map(tuple, new_state)) in closed_set:
                continue
            g = node.depth + 1
            h = manhattan_distance(new_state, goal_state)
            f = g + h
            children.append((f, g, h, move, new_state))

        children.sort()
        for f, g, h, move, new_state in children:
            print(f"Move: {move}, f(n)={f} (g(n)={g}, h(n)={h})\n{new_state}\n")
            heapq.heappush(open_list, Puzzle(new_state, parent=node, move=move, depth=g, cost=f))
        print("-" * 100)
    return None

start = [[2, 8, 3], [1, 6, 4], [7, 0, 5]]
goal = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]
path = a_star(start, goal)

print("Solution Path:")
for step, (move, state) in enumerate(path):
    print(f"Step {step}: Move: {move}\n{state}\n")
```

```
 [1 0 4]
 [7 6 5]]

Step 2: Move: Up
[[2 0 3]
 [1 8 4]
 [7 6 5]]

Step 3: Move: Left
[[0 2 3]
 [1 8 4]
 [7 6 5]]

Step 4: Move: Down
[[1 2 3]
 [0 8 4]
 [7 6 5]]

Step 5: Move: Right
[[1 2 3]
 [8 0 4]
 [7 6 5]]

<ipython-input-2-1a316262d512>:25: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error
  x, y = int(x), int(y)
```