Week-2

This week I have tried to implement BFS and DFS algorithms for the BotBrain campus navigation system. The implementation involved getting GPS coordinates of campus locations, creating a campus graph, and implementing both search algorithms.

First Get the Coordinates

I collected real GPS coordinates for 11 key locations of Chanakya University campus in DMS (Degrees, Minutes, Seconds) format and converted them to decimal degrees for computational processing.

GPS Coordinates Collected:

- Main Gate: 13°13'16.7"N 77°45'18.2"E
- Admin Block: 13°13'19.9"N 77°45'18.9"E
- Academic Block 1: 13°13'24.0"N 77°45'17.7"E
- Academic Block 2: 13°13'24.2"N 77°45'21.4"E
- Academic Block 3: 13°13'20.6"N 77°45'22.6"E
- Hostel: 13°13'28.3"N 77°45'32.6"E
- Food Court: 13°13'29.5"N 77°45'26.0"E
- Sports Complex: 13°13'42.3"N 77°45'29.8"E
- Central Junction: 13°13'22.0"N 77°45'20.2"E
- Laundry: 13°13'28.3"N 77°45'25.4"E
- Mini Mart: 13°13'28.4"N 77°45'32.9"E

Converted to Decimal Degrees:

- Main Gate: (13.221306, 77.755056)
- Admin Block: (13.222194, 77.755250)
- Academic Block 1: (13.223333, 77.754917)
- Academic Block 2: (13.223389, 77.755944)
- Academic Block 3: (13.222389, 77.756278)
- Hostel: (13.224528, 77.759056)
- Food Court: (13.224861, 77.757222)
- Sports Complex: (13.228417, 77.758278)
- Central Junction: (13.222778, 77.755611)
- Laundry: (13.224528, 77.757056)
- Mini Mart: (13.224556, 77.759139)

➢

Campus Graph

Code:

```python
import math

def dms_to_decimal(dms_string):
    """Convert DMS coordinate string to decimal degrees"""
    dms_string = dms_string.replace('°', ' ').replace("'", ' ').replace('"', ' ')

    direction = 1
    if 'S' in dms_string or 'W' in dms_string:
        direction = -1

    dms_string = dms_string.replace('N', '').replace('S', '').replace('E', '').replace('W', '')
    parts = [float(x) for x in dms_string.split() if x]

    decimal = parts + parts/60 + parts/3600
    return decimal * direction

def haversine_distance(lat1, lon1, lat2, lon2):
    """Calculate the great circle distance between two points on Earth in meters"""
    lat1, lon1, lat2, lon2 = map(math.radians, [lat1, lon1, lat2, lon2])

    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = math.sin(dlat/2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon/2)**2
    c = 2 * math.asin(math.sqrt(a))

    r = 6371000  # Earth's radius in meters
    return c * r

def create_campus_graph():
    """Create graph representation based on the campus map"""
    graph = {location: [] for location in decimal_coordinates.keys()}

    connections = [
        ("Main Gate", "Admin Block"),
        ("Admin Block", "Central Junction"),
        ("Central Junction", "Academic Block 1"),
        ("Central Junction", "Academic Block 2"),
        ("Central Junction", "Academic Block 3"),
        ("Academic Block 2", "Food Court"),
        ("Food Court", "Laundry"),
        ("Food Court", "Hostel"),
        ("Food Court", "Mini Mart"),
        ("Hostel", "Mini Mart"),
        ("Food Court", "Sports Complex"),
        ("Laundry", "Hostel")
    ]

    for loc1, loc2 in connections:
        lat1, lon1 = decimal_coordinates[loc1]
        lat2, lon2 = decimal_coordinates[loc2]
        distance = haversine_distance(lat1, lon1, lat2, lon2)

        graph[loc1].append({'node': loc2, 'distance': round(distance, 2)})
        graph[loc2].append({'node': loc1, 'distance': round(distance, 2)})

    return graph
```

Output:

Campus Graph Adjacency List:

Main Gate:

-> Admin Block (101.06 meters)

Admin Block:

-> Main Gate (101.06 meters)

-> Central Junction (75.73 meters)

Academic Block 1:

-> Central Junction (97.3 meters)

Academic Block 2:

-> Central Junction (76.94 meters)

-> Food Court (214.31 meters)

Academic Block 3:

-> Central Junction (84.13 meters)

Hostel:

-> Food Court (201.88 meters)

-> Mini Mart (9.53 meters)

-> Laundry (216.49 meters)

Food Court:

-> Academic Block 2 (214.31 meters)

-> Laundry (41.22 meters)

-> Hostel (201.88 meters)

-> Mini Mart (210.24 meters)

-> Sports Complex (411.54 meters)

Sports Complex:

-> Food Court (411.54 meters)

Central Junction:

-> Admin Block (75.73 meters)

-> Academic Block 1 (97.3 meters)

-> Academic Block 2 (76.94 meters)

-> Academic Block 3 (84.13 meters)

Laundry:

-> Food Court (41.22 meters)

-> Hostel (216.49 meters)

Mini Mart:

-> Food Court (210.24 meters)

-> Hostel (9.53 meters)

- **BFS:**

  code:

```python
from collections import deque
import time

def breadth_first_search(graph, start, goal):
    """
    Breadth-First Search implementation for campus navigation
    Returns: path, nodes_explored, distance_traveled, execution_time
    """
    start_time = time.time()

    if start not in graph or goal not in graph:
        return None, 0, 0, 0

    if start == goal:
        return [start], 1, 0, time.time() - start_time

    # Initialize BFS data structures
    queue = deque([(start, [start], 0)])  # (current_node, path, total_distance)
    visited = set()
    nodes_explored = 0

    print(f"\\n🔍 BFS Search from {start} to {goal}")
    print("=" * 60)
    print("Step | Current Node    | Queue Size | Visited | Path")
    print("-" * 60)

    step = 0
    while queue:
        current_node, path, total_distance = queue.popleft()
        nodes_explored += 1
        step += 1

        print(f"{step:4d} | {current_node:15s} | {len(queue):10d} | {len(visited):7d} | {' -> '.join(path)}")

        if current_node == goal:
            end_time = time.time()
            print(f"\\n✅ Goal reached! Path found: {' -> '.join(path)}")
            print(f"Total distance: {total_distance:.2f} meters")
            return path, nodes_explored, total_distance, end_time - start_time

        if current_node not in visited:
            visited.add(current_node)

            # Add neighbors to queue
            for neighbor_info in graph[current_node]:
                neighbor = neighbor_info['node']
                distance = neighbor_info['distance']

                if neighbor not in visited:
                    new_path = path + [neighbor]
                    new_distance = total_distance + distance
                    queue.append((neighbor, new_path, new_distance))

    end_time = time.time()
    print(f"\\n❌ No path found from {start} to {goal}")
    return None, nodes_explored, 0, end_time - start_time
```

Output:

Step	Current Node	Queue Size	Visited	Path
1	Main Gate	0	0	Main Gate
2	Admin Block	0	1	Main Gate -> Admin Block
3	Central Junction	0	2	Main Gate -> Admin Block -> Central Junction
4	Academic Block 1	2	3	Main Gate -> Admin Block -> Central Junction -> Academic Block 1
5	Academic Block 2	1	4	Main Gate -> Admin Block -> Central Junction -> Academic Block 2
6	Academic Block 3	1	5	Main Gate -> Admin Block -> Central Junction -> Academic Block 3
7	Food Court	0	6	Main Gate -> Admin Block -> Central Junction -> Academic Block 2 -> Food Court

✅ Goal reached! Path found: Main Gate -> Admin Block -> Central Junction -> Academic Block 2 -> Food Court

Total distance: 468.04 meters

BFS Performance:

- Path Length: 5 steps
- Nodes Explored: 7
- Total Distance: 468.04 meters
- Execution Time: 0.38 ms

- **DSF**:

CODE:

```
def depth_first_search(graph, start, goal):
    """Depth-First Search implementation for campus navigation
    Returns: path, nodes_explored, distance_traveled, execution_time"""
    start_time = time.time()

    if start not in graph or goal not in graph:
        return None, 0, 0, 0

    if start == goal:
        return [start], 1, 0, time.time() - start_time

    # Initialize DFS data structures
    stack = [(start, [start], 0)]  # (current_node, path, total_distance)
    visited = set()
    nodes_explored = 0

    print(f"\\n🔍 DFS Search from {start} to {goal}")
    print("=" * 60)
    print("Step | Current Node     | Stack Size | Visited | Path")
    print("-" * 60)

    step = 0
    while stack:
        current_node, path, total_distance = stack.pop()
        nodes_explored += 1
        step += 1

        print(f"{step:4d} | {current_node:15s} | {len(stack):10d} | {len(visited):7d} | {' -> '.join(path)}")

        if current_node == goal:
            end_time = time.time()
            print(f"\\n✅ Goal reached! Path found: {' -> '.join(path)}")
            print(f"Total distance: {total_distance:.2f} meters")
            return path, nodes_explored, total_distance, end_time - start_time

        if current_node not in visited:
            visited.add(current_node)

            # Add neighbors to stack (in reverse order for consistent exploration)
            neighbors = []
            for neighbor_info in graph[current_node]:
                neighbor = neighbor_info['node']
                distance = neighbor_info['distance']

                if neighbor not in visited:
                    new_path = path + [neighbor]
                    new_distance = total_distance + distance
                    neighbors.append((neighbor, new_path, new_distance))

            # Add in reverse order to maintain left-to-right exploration
            for neighbor_data in reversed(neighbors):
                stack.append(neighbor_data)

    end_time = time.time()
    print(f"\\n❌ No path found from {start} to {goal}")
    return None, nodes_explored, 0, end_time - start_time
```

OUTPUT

Step	Current Node	Stack Size	Visited	Path
1	Main Gate	0	0	Main Gate
2	Admin Block	0	1	Main Gate -> Admin Block
3	Central Junction	0	2	Main Gate -> Admin Block -> Central Junction
4	Academic Block 1	2	3	Main Gate -> Admin Block -> Central Junction -> Academic Block 1
5	Academic Block 2	1	4	Main Gate -> Admin Block -> Central Junction -> Academic Block 2
6	Food Court	1	5	Main Gate -> Admin Block -> Central Junction -> Academic Block 2 -> Food Court

Goal reached! Path found:

Main Gate -> Admin Block -> Central Junction -> Academic Block 2 -> Food Court

Total distance: 468.04 meters

DFS Performance:

- Path Length: 5 steps

- Nodes Explored: 6

- Total Distance: 468.04 meters

- Execution Time: 0.28 ms

➢ **Algorithm Comparison Results**

Test Cases:

1. **Main Gate → Food Court**
    - BFS: 5 steps, 468.04m, 7 nodes explored, 0.38ms
    - DFS: 5 steps, 468.04m, 6 nodes explored, 0.28ms
2. **Academic Block 1 → Hostel**
    - BFS: 5 steps, 590.43m, 9 nodes explored, 0.42ms
    - DFS: 6 steps, 646.26m, 8 nodes explored, 0.34ms
3. **Admin Block → Sports Complex**
    - BFS: 5 steps, 778.52m, 11 nodes explored, 0.42ms
    - DFS: 5 steps, 778.52m, 12 nodes explored, 0.57ms

Summary:

- BFS found shorter paths in 1/3 cases
- DFS was faster on average (0.40ms vs 0.41ms)
- Both algorithms successfully found paths in all test cases
- Campus graph connectivity verified - all locations reachable

Week 2 Objectives Completed:

- ☑ GPS coordinate collection and conversion
- ☑ Campus graph construction with real distances
- ☑ BFS algorithm implementation and testing
- ☑ DFS algorithm implementation and testing
- ☑ Performance comparison and analysis