Week 2 Report

NaviBot Campus Navigation Algorithms

Overview

Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms were implemented for the NaviBot campus navigation system at Chanakya University. The workflow included precise GPS data acquisition, graph construction, route-finding algorithm development, and in-depth comparative analysis.

GPS Data Acquisition

GPS coordinates for eleven primary locations within the Chanakya University campus were obtained in Degrees, Minutes, Seconds (DMS) format and subsequently converted to decimal degrees for computational processing. These coordinates are summarized below:

Location	**Decimal Coordinates**
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)

Location	Decimal Coordinates
Mini Mart	(13.224556, 77.759139)

Campus Graph Construction

A campus graph was constructed wherein every location acts as a node, and real walking paths function as edges. Edge weights represent physical distances between locations, computed using the haversine formula. The graph's adjacency list includes the following connections and distances (in meters):

- Main Gate ↔ Admin Block (101.06)
- Admin Block ↔ Central Junction (75.73)
- Academic Block 1 ↔ Central Junction (97.3)
- Academic Block 2 ↔ Central Junction (76.94)
- Academic Block 2 ↔ Food Court (214.31)
- Academic Block 3 ↔ Central Junction (84.13)
- Hostel ↔ Food Court (201.88)
- Hostel ↔ Mini Mart (9.53)
- Hostel ↔ Laundry (216.49)
- Food Court ↔ Laundry (41.22)
- Food Court ↔ Mini Mart (210.24)
- Food Court ↔ Sports Complex (411.54)
- Laundry ↔ Hostel (216.49)

Algorithm Implementation

Breadth-First Search (BFS)

The BFS algorithm was implemented to identify shortest paths between locations. BFS explores nodes level-wise while maintaining a queue and visited set. Output includes step-by-step path discovery, node exploration count, total traversed distance, and execution time.

Depth-First Search (DFS)

The DFS algorithm was developed for alternative route identification. DFS delves deeper into possible paths before backtracking, utilizing a stack and visited set for robust exploration. The implementation records each traversed node, path, cumulative distance, and execution time.

Results and Performance Analysis

Multiple test cases were executed to evaluate both algorithms. Comparative results for each approach are provided below:

Case	BFS Steps	BFS Distance (m)	BFS Nodes	BFS Time (ms)	DFS Steps	DFS Distance (m)	DFS Nodes	DFS Time (ms)
Main Gate → Food Court	5	468.04	7	0.38	5	468.04	6	0.28
Academic Block 1 → Hostel	5	590.43	9	0.42	6	646.26	8	0.34
Admin Block → Sports Complex	5	778.52	11	0.42	5	778.52	12	0.57

Observations

- BFS produced shorter or equal paths in approximately one-third of tested cases.
- DFS demonstrated marginally faster execution times (average: 0.40 ms) compared to BFS (average: 0.41 ms).
- Both algorithms reliably found valid paths for all test scenarios, confirming campus-wide connectivity.

Achievements

- Comprehensive GPS data collection and conversion to decimal degrees
- Construction of campus graph using genuine distance metrics
- Implementation and successful testing of BFS and DFS algorithms
- Performance comparison utilizing real campus scenarios
- Verification of universal navigational reachability within the campus

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"),
```

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()
```