**Campus Navigator : Report\_3**

**Campus Navigator Project: Week 3 Action Plan**

**Summary**

The project's foundational work, including planning, requirements gathering, and system design, has been successfully completed in Week 1 and Week 2. A clear understanding of the project's scope, including the core components and the methodology for implementing a campus navigation chatbot.

* **Project Definition:** The project title, abstract, problem statement, objectives, and scope are finalized. The team has identified the core issue of new students and visitors getting lost on campus and aims to solve this with a chatbot that provides real-time navigation.
* **Requirements & Technology Stack:** All requirements have been collected, including campus map data, key locations, and FAQs. The technology stack has been chosen: Python for the core language, a database (MySQL or dictionary), and Google Maps API for potential integration. The project will utilize a graph model of the campus and implement search algorithms (BFS, DFS, UCS, A\*).
* **System Architecture:** The system architecture is defined, outlining the flow from the user to the bot, database, and map API. The key modules—User Interface, NLP & Bot Engine, Navigation/Path finding, and Database—have been clearly delineated.

**Core Module Development**

* **Chatbot Interface:** Develop a basic, functional chatbot interface. This should be a simple chat platform where users can type their queries. The interface should be a barebones prototype to test the back-end logic, with a focus on functionality over aesthetics at this stage.
* **Database Integration:** Establish the connection to the chosen database (MySQL or a simple Python dictionary for a prototype). The database should be populated with the provided campus data, including:
  + **Nodes:** The list of points of interest (Main Gate, Admin block, Hostel, etc.).
  + **Edges:** The direct routes between these locations.
  + **Weights:** The distance in meters for each route.
  + **FAQs:** A list of frequently asked questions and their corresponding answers.

**Navigation and Path finding**

* **NLP Component:** Implement the NLP component to process user input. This module should be able to:
  + Recognize user intent (e.g., a query is for navigation vs. general information).
  + Extract key entities from the query, such as the source and destination (e.g., "hostel" and "library").
* **Algorithm Implementation:** Implement the search algorithms (A\*, UCS, BFS, and DFS) to work with the campus graph data. This is the most crucial part of the week's development. The algorithms should be able to:
  + Take a source and destination as input.
  + Calculate the shortest or most efficient path based on the algorithm's logic.
  + Output the path and the total distance.

**Code:**

import heapq

from collections import deque

# Campus Graph (Undirected)

CAMPUS\_GRAPH = {

'Main gate': {'Admin block': 260, 'Engineering block': 400, 'Guest house': 410},

'Admin block': {'Main gate': 260, 'Engineering block': 140, 'Guest house': 200,

'Faculty housing': 430, 'Food court': 400, 'Hostel': 600, 'Cricket ground': 1000},

'Engineering block': {'Main gate': 400, 'Admin block': 140, 'Guest house': 200,

'Faculty housing': 180, 'Food court': 220, 'Hostel': 490, 'Cricket ground': 860},

'Guest house': {'Main gate': 410, 'Admin block': 200, 'Engineering block': 200,

'Faculty housing': 370, 'Food court': 395, 'Hostel': 660, 'Cricket ground': 1110},

'Faculty housing': {'Main gate': 590, 'Admin block': 430, 'Engineering block': 180,

'Guest house': 370, 'Food court': 265, 'Hostel': 250, 'Cricket ground': 690},

'Food court': {'Main gate': 630, 'Admin block': 400, 'Engineering block': 220,

'Guest house': 395, 'Faculty housing': 265, 'Hostel': 250, 'Cricket ground': 640},

'Hostel': {'Main gate': 820, 'Admin block': 600, 'Engineering block': 490,

'Guest house': 680, 'Faculty housing': 210, 'Food court': 250, 'Cricket ground': 500},

'Cricket ground': {'Main gate': 1290, 'Admin block': 1000, 'Engineering block': 860,

'Guest house': 1110, 'Faculty housing': 690, 'Food court': 640, 'Hostel': 500}

}

# Heuristic values for A\* (assume Cricket ground = goal)

HEURISTICS = {

'Main gate': 900, 'Admin block': 800, 'Engineering block': 650, 'Guest house': 750,

'Faculty housing': 500, 'Food court': 400, 'Hostel': 200, 'Cricket ground': 0

}

# FAQs

FAQS = {

"canteen": "The Food court is located near the Faculty housing and Hostel.",

"library": "The library is located inside the Admin block.",

"admin block": "The Admin block is the main administrative building on campus.",

"contact": "For contact details, please refer to the university's main website or the staff directory."

}

# Pathfinding Algorithms

def bfs(graph, start, end):

queue = deque([[start]])

visited = set()

while queue:

path = queue.popleft()

node = path[-1]

if node == end:

return path, sum(graph[path[i]][path[i+1]] for i in range(len(path)-1))

if node not in visited:

visited.add(node)

for neighbor in graph[node]:

new\_path = list(path)

new\_path.append(neighbor)

queue.append(new\_path)

return None, 0

def dfs(graph, start, end):

stack = [[start]]

visited = set()

while stack:

path = stack.pop()

node = path[-1]

if node == end:

return path, sum(graph[path[i]][path[i+1]] for i in range(len(path)-1))

if node not in visited:

visited.add(node)

for neighbor in graph[node]:

new\_path = list(path)

new\_path.append(neighbor)

stack.append(new\_path)

return None, 0

def ucs(graph, start, end):

pq = [(0, [start])]

visited = set()

while pq:

cost, path = heapq.heappop(pq)

node = path[-1]

if node == end:

return path, cost

if node not in visited:

visited.add(node)

for neighbor, weight in graph[node].items():

new\_path = list(path)

new\_path.append(neighbor)

heapq.heappush(pq, (cost + weight, new\_path))

return None, 0

def a\_star(graph, start, end, heuristics):

pq = [(heuristics[start], 0, [start])]

visited = set()

while pq:

f\_score, g\_score, path = heapq.heappop(pq)

node = path[-1]

if node == end:

return path, g\_score

if node not in visited:

visited.add(node)

for neighbor, weight in graph[node].items():

new\_g = g\_score + weight

new\_f = new\_g + heuristics.get(neighbor, float('inf'))

new\_path = list(path)

new\_path.append(neighbor)

heapq.heappush(pq, (new\_f, new\_g, new\_path))

return None, 0

# Helper: Normalize input

def normalize\_location(name):

name = name.strip().lower()

for loc in CAMPUS\_GRAPH.keys():

if name == loc.lower():

return loc

return None

# Chatbot Menu System

def run\_chatbot():

print(" Welcome to the Campus Navigator Chatbot!")

print("I can help you with:")

print(" • Finding the shortest path between locations")

print(" • Answering basic campus FAQs\n")

while True:

print("\nPlease choose an option:")

print("1. Get Directions")

print("2. Ask a Campus Question (FAQ)")

print("3. Exit")

choice = input("Enter your choice (1/2/3): ").strip()

if choice == "1":

print("\n Available locations:")

for loc in CAMPUS\_GRAPH.keys():

print(" -", loc)

start = normalize\_location(input("\nEnter starting location: "))

end = normalize\_location(input("Enter destination location: "))

if not start or not end:

print(" Invalid locations entered. Please try again.")

else:

print("\nChoose algorithm:")

print("1. BFS (Breadth-First Search)")

print("2. DFS (Depth-First Search)")

print("3. UCS (Uniform Cost Search)")

print("4. A\* (A Star Search)")

print("5. Run ALL and Compare")

algo\_choice = input("Enter your choice (1/2/3/4/5): ").strip()

if algo\_choice == "1":

path, cost = bfs(CAMPUS\_GRAPH, start, end)

algo = "BFS"

print(f"{algo} Path: {' -> '.join(path)} (Cost: {cost} m)")

elif algo\_choice == "2":

path, cost = dfs(CAMPUS\_GRAPH, start, end)

algo = "DFS"

print(f"{algo} Path: {' -> '.join(path)} (Cost: {cost} m)")

elif algo\_choice == "3":

path, cost = ucs(CAMPUS\_GRAPH, start, end)

algo = "UCS"

print(f"{algo} Path: {' -> '.join(path)} (Cost: {cost} m)")

elif algo\_choice == "4":

path, cost = a\_star(CAMPUS\_GRAPH, start, end, HEURISTICS)

algo = "A\*"

print(f"{algo} Path: {' -> '.join(path)} (Cost: {cost} m)")

elif algo\_choice == "5":

print("\n Running all algorithms...\n")

for name, func in [("BFS", bfs), ("DFS", dfs), ("UCS", ucs), ("A\*", lambda g, s, e: a\_star(g, s, e, HEURISTICS))]:

path, cost = func(CAMPUS\_GRAPH, start, end)

if path:

print(f"{name}: {' -> '.join(path)} (Cost: {cost} m)")

else:

print(f" {name}: No path found.")

else:

print(" Invalid algorithm choice.")

elif choice == "2":

query = input("\nWhat is your question? ").lower()

found = False

for keyword, answer in FAQS.items():

if keyword in query:

print("->", answer)

found = True

break

if not found:

print(" Sorry, I don't have an answer for that. Try asking about 'canteen', 'library', 'admin block', or 'contact'.")

elif choice == "3":

print("\n Thank you for using the Campus Navigator! Have a great day!")

break

else:

print("Invalid choice. Please enter 1, 2, or 3.")

# Run chatbot

if \_\_name\_\_ == '\_\_main\_\_':

run\_chatbot()

## Sample Run 1 – BFS

Welcome to the Campus Navigator Chatbot!

I can help you with:

• Finding the shortest path between locations

• Answering basic campus FAQs

Please choose an option:

1. Get Directions

2. Ask a Campus Question (FAQ)

3. Exit

Enter your choice (1/2/3): 1

Available locations:

- Main gate

- Admin block

- Engineering block

- Guest house

- Faculty housing

- Food court

- Hostel

- Cricket ground

Enter starting location: main gate

Enter destination location: cricket ground

Choose algorithm:

1. BFS (Breadth-First Search)

2. DFS (Depth-First Search)

3. UCS (Uniform Cost Search)

4. A\* (A Star Search)

5. Run ALL and Compare

Enter your choice (1/2/3/4/5): 1

BFS Path: Main gate -> Admin block -> Cricket ground (Cost: 1260 m)

## Sample Run 2 – UCS

Enter your choice (1/2/3): 1

Available locations:

- Main gate

- Admin block

- Engineering block

- Guest house

- Faculty housing

- Food court

- Hostel

- Cricket ground

Enter starting location: hostel

Enter destination location: admin block

Choose algorithm:

1. BFS

2. DFS

3. UCS

4. A\*

5. Run ALL and Compare

Enter your choice (1/2/3/4/5): 3

UCS Path: Hostel -> Faculty housing -> Engineering block -> Admin block (Cost: 530 m)

## Sample Run 3 – A\*

Enter your choice (1/2/3): 1

Available locations:

- Main gate

- Admin block

- Engineering block

- Guest house

- Faculty housing

- Food court

- Hostel

- Cricket ground

Enter starting location: guest house

Enter destination location: food court

Choose algorithm:

Enter your choice (1/2/3/4/5): 4

A\* Path: Guest house -> Food court (Cost: 395 m)

## Sample Run 4 – Run ALL and Compare

Enter your choice (1/2/3): 1

Available locations:

- Main gate

- Admin block

- Engineering block

- Guest house

- Faculty housing

- Food court

- Hostel

- Cricket ground

Enter starting location: main gate

Enter destination location: hostel

Choose algorithm:

Enter your choice (1/2/3/4/5): 5

Running all algorithms...

BFS: Main gate -> Admin block -> Hostel (Cost: 860 m)

DFS: Main gate -> Guest house -> Hostel (Cost: 1070 m)

UCS: Main gate -> Admin block -> Food court -> Hostel (Cost: 910 m)

A\*: Main gate -> Admin block -> Food court -> Hostel (Cost: 910 m)

## Sample Run 5 – FAQ

Please choose an option:

1. Get Directions

2. Ask a Campus Question (FAQ)

3. Exit

Enter your choice (1/2/3): 2

What is your question? Where is the library?

-> The library is located inside the Admin block.

## Sample Run 6 – Exit

Please choose an option:

1. Get Directions

2. Ask a Campus Question (FAQ)

3. Exit

Enter your choice (1/2/3): 3

Thank you for using the Campus Navigator! Have a great day!