**CHANAKYA UNIVERSITY**

SCHOOL OF ENGINEERING



**ASSIGNMENT TITLE: CHANAKYA UNIVERSITY DIGITAL PATH FINDER**

**ASSIGNMENT -1**

**SUBMITTED BY:**

**VARSHITHA.T**

**REGISTER NO:24UG00549**

**SEMESTER III (ODD)**

**SUBJECT: INTRODUCTION TO AIML**

**SECTION B**

**ABSTRACT**

The Bot Brain project aims to develop a smart digital navigation assistant for Chanakya University. The system models campus navigation as a weighted graph, where the buildings are modelled as nodes and the paths that connect the buildings are modelled as edges. Bot Brain determines the best routes for users in as little time as possible using classical AI search algorithms such as Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS), and A\*, with heuristics based on search methodology. In addition to providing the best route, Bot Brain also provides valuable information about the buildings such as facilities, timings, and available services in relation to their use and purpose, thereby improving the overall campus experience. A primary intent of the project is to compare the various search algorithms and allow students to discover for themselves the effect of algorithm choices on performance and efficiency in real world applications. By combining navigation, access to information and analysis of algorithm, Bot Brain provides a tangible AI-based solution that enhances the campus experience, whether it is improving the accessibility, efficiency, or user friendliness of the campus for students and visitors alike.

**INTRODUCTION**

With its academic, residential, and recreational amenities, a university campus is like a miniature city. Long walking paths, numerous connected buildings, and a dearth of interactive navigation systems make it difficult for new students to navigate such a campus. Conventional approaches, such as static signboards or printed maps, are frequently inadequate.  
  
By using search algorithms to find the best routes and representing the campus as a graph, artificial intelligence offers a solution. Through the use of AI agent design, the Bot Brain project provides contextual information about buildings and models intelligent navigational decision-making. The system makes use of A\* for heuristic-driven efficiency, UCS for cost-aware routing, and BFS and DFS for basic search. Beyond the classroom, the project gives students practical knowledge of intelligent agents, search issues, and algorithm evaluation.

**PROBLEM STATEMENT**

It can be difficult for visitors and new students to find their way around Chanakya University's expansive and strange campus. They struggle with things like not knowing where buildings are, wasting time looking for offices or classrooms, needing help from others to get directions, and not having easy access to building information like services or timings. This results in ineffective orientation, stress, and delays.  
  
An intelligent campus navigation assistant is required to address this issue, guiding users with the shortest routes, walking times, and building details to make campus exploration more efficient and easier for newcomers.

**OBJECTIVES**

Objectives

1. Graph Model Development: Design a comprehensive graph model representing Chanakya University's campus, encompassing at least 12 key buildings.

2. Algorithm Implementation: Implement and integrate Breadth-First Search (BFS), Depth-First Search (DFS), Uniform Cost Search (UCS), and A\* algorithms to facilitate efficient navigation.

3. User Query Support: Enable users to pose queries such as "Find path from Hostel to Library" or "Navigate from Main Gate to Admin Block," and provide accurate responses.

4. Route Computation and Display: Calculate and display the shortest routes between destinations, including distance and estimated walking time.

5. Building and Service Information: Provide supplementary details about campus buildings and services, enhancing user experience.

6. Algorithmic Performance Comparison: Conduct a comparative analysis of the implemented algorithms in terms of:

- Path optimality

- Number of nodes explored

- Computational efficiency

By achieving these objectives, the project aims to develop an intelligent campus navigation system that offers efficient routing, informative building details, and insightful algorithmic comparisons.

**SCOPE**

Key Features

1. Graph-Based Campus Model: A comprehensive graph representation of the campus, enabling efficient navigation.

2. Implementation of 4 Search Algorithms: BFS, DFS, UCS, and A\* algorithms for optimal route-finding.

3. Text-Based Interface: User-friendly interface for navigation queries, such as "Find path from Hostel to Library."

4. Building Information Retrieval: Access to detailed information about campus buildings and services.

5. Algorithm Comparison and Analysis: Comparative evaluation of algorithm performance in terms of path optimality, nodes explored, and efficiency.

These features will be integrated to develop an intelligent campus navigation system, providing users with efficient routing and informative building details.

**CAMPUS LAYOUT**

* Main Gate
* Exit Gate
* Admin Block
* Academic Block
* Library
* Canteen
* Hostel
* Mart
* Food Court
* Sports Ground
* Auditorium
* Medical Facility

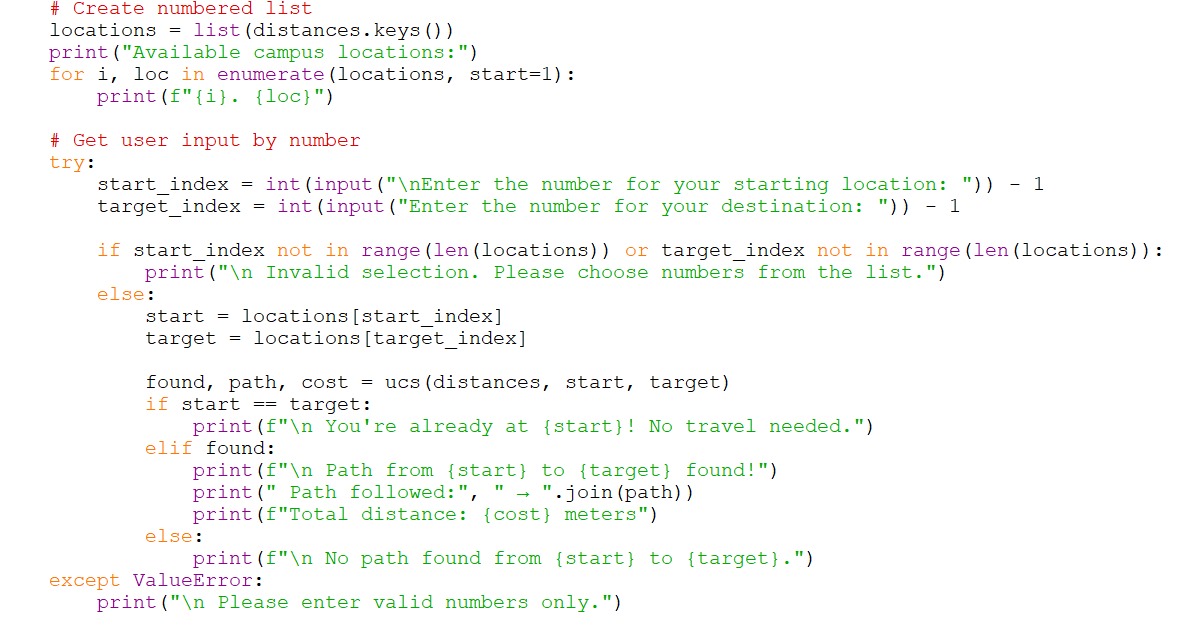
A drawing of a machine

AI-generated content may be incorrect.

**TOOLS & TECHNOLOGIES**

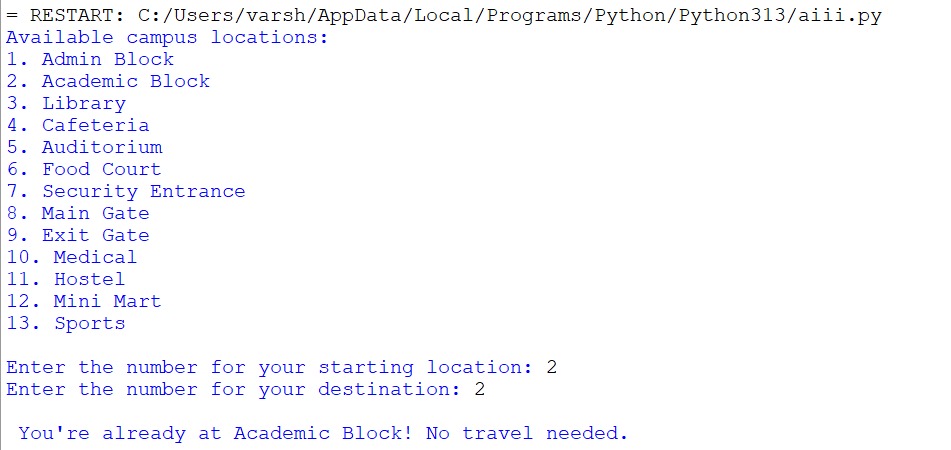
* **Python**: Selected as the core programming language for the development of the intelligent agent, implementation of search algorithms, and management of back-end logic. Python is preferred owing to its robustness, simplicity of use, and extensive ecosystem supporting artificial intelligence and computational tasks.
* **Tkinter / Py Simple GUI / Flask**: Frameworks designated for the development of the user interface. Tkinter and PySimpleGUI facilitate the rapid construction of desktop-based graphical interfaces, whereas Flask provides a minimal yet effective platform for web-based deployment.
* **GitHub**: Employed as the version control system to maintain source code integrity, enable collaborative development, streamline code organization, and facilitate systematic peer review processes.
* **Google Maps API**: Integrated to provide satellite-based campus imagery and precise geolocation services. This enables accurate digital mapping of real-world building positions, with optional enhancement for advanced overlay and visualization features.
* **Microsoft PowerPoint / Word**: Utilized for the design and preparation of technical documentation, visual diagrams, system design records, and professional project presentations.

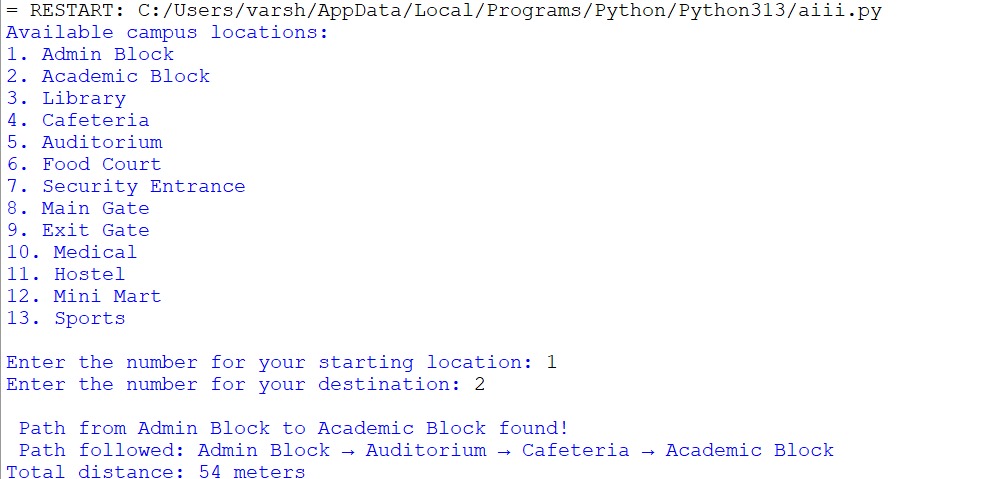
**USC (INPUT)**



OUTPUT







CODE

from queue import PriorityQueue

def ucs(graph, start, target):

if start == target:

return True, [start], 0

visited = set()

priority\_queue = PriorityQueue()

priority\_queue.put((0, start, [start]))

while not priority\_queue.empty():

cost, node, path = priority\_queue.get()

if node == target:

return True, path, cost

if node not in visited:

visited.add(node)

for neighbor, edge\_cost in graph.get(node, {}).items():

if neighbor not in visited:

priority\_queue.put((cost + edge\_cost, neighbor, path + [neighbor]))

return False, [], 0

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

# Campus distances dictionary

distances = {

"Admin Block": {"Security Entrance": 80, "Library": 16, "Auditorium": 17, "Academic Block": 126, "Cafeteria": 35, "Main Gate": 160, "Exit Gate": 220, "Medical": 80, "Hostel": 506, "Mini Mart": 434, "Food Court": 355, "Sports": 850},

"Academic Block": {"Security Entrance": 240, "Library": 46, "Auditorium": 150, "Cafeteria": 27, "Main Gate": 210, "Exit Gate": 250, "Medical": 85, "Hostel": 380, "Mini Mart": 410, "Food Court": 192, "Admin Block": 126, "Sports": 561},

"Library": {"Security Entrance": 100, "Auditorium": 3, "Academic Block": 46, "Cafeteria": 63, "Main Gate": 275, "Exit Gate": 240, "Medical": 75, "Hostel": 450, "Mini Mart": 465, "Food Court": 470, "Admin Block": 16, "Sports": 857},

"Cafeteria": {"Security Entrance": 130, "Library": 63, "Auditorium": 10, "Academic Block": 27, "Main Gate": 276, "Exit Gate": 270, "Medical": 90, "Hostel": 520, "Mini Mart": 585, "Food Court": 355, "Admin Block": 35, "Sports": 920},

"Auditorium": {"Security Entrance": 120, "Library": 3, "Academic Block": 150, "Cafeteria": 10, "Main Gate": 286, "Exit Gate": 260, "Medical": 80, "Hostel": 510, "Mini Mart": 574, "Food Court": 344, "Admin Block": 17, "Sports": 930},

"Food Court": {"Security Entrance": 471, "Library": 470, "Auditorium": 344, "Academic Block": 192, "Cafeteria": 355, "Main Gate": 699, "Exit Gate": 609, "Medical": 405, "Hostel": 816, "Mini Mart": 836, "Admin Block": 355, "Sports": 427},

"Security Entrance": {"Admin Block": 80, "Academic Block": 240, "Library": 100, "Cafeteria": 130, "Auditorium": 120, "Food Court": 471, "Main Gate": 150, "Exit Gate": 180, "Medical": 90, "Hostel": 500, "Mini Mart": 520, "Sports": 880},

"Main Gate": {"Admin Block": 160, "Academic Block": 210, "Library": 275, "Cafeteria": 276, "Auditorium": 286, "Food Court": 699, "Security Entrance": 150, "Exit Gate": 100, "Medical": 180, "Hostel": 600, "Mini Mart": 620, "Sports": 900},

"Exit Gate": {"Admin Block": 220, "Academic Block": 250, "Library": 240, "Cafeteria": 270, "Auditorium": 260, "Food Court": 609, "Security Entrance": 180, "Main Gate": 100, "Medical": 160, "Hostel": 580, "Mini Mart": 600, "Sports": 870},

"Medical": {"Admin Block": 80, "Academic Block": 85, "Library": 75, "Cafeteria": 90, "Auditorium": 80, "Food Court": 405, "Security Entrance": 90, "Main Gate": 180, "Exit Gate": 160, "Hostel": 490, "Mini Mart": 510, "Sports": 840},

"Hostel": {"Admin Block": 506, "Academic Block": 380, "Library": 450, "Cafeteria": 520, "Auditorium": 510, "Food Court": 816, "Security Entrance": 500, "Main Gate": 600, "Exit Gate": 580, "Medical": 490, "Mini Mart": 200, "Sports": 300},

"Mini Mart": {"Admin Block": 434, "Academic Block": 410, "Library": 465, "Cafeteria": 585, "Auditorium": 574, "Food Court": 836, "Security Entrance": 520, "Main Gate": 620, "Exit Gate": 600, "Medical": 510, "Hostel": 200, "Sports": 320},

"Sports": {"Admin Block": 850, "Academic Block": 561, "Library": 857, "Cafeteria": 920, "Auditorium": 930, "Food Court": 427, "Security Entrance": 880, "Main Gate": 900, "Exit Gate": 870, "Medical": 840, "Hostel": 300, "Mini Mart": 320}

}

# Create numbered list

locations = list(distances.keys())

print("Available campus locations:")

for i, loc in enumerate(locations, start=1):

print(f"{i}. {loc}")

# Get user input by number

try:

start\_index = int(input("\nEnter the number for your starting location: ")) - 1

target\_index = int(input("Enter the number for your destination: ")) - 1

if start\_index not in range(len(locations)) or target\_index not in range(len(locations)):

print("\n Invalid selection. Please choose numbers from the list.")

else:

start = locations[start\_index]

target = locations[target\_index]

found, path, cost = ucs(distances, start, target)

if start == target:

print(f"\n You're already at {start}! No travel needed.")

elif found:

print(f"\n Path from {start} to {target} found!")

print(" Path followed:", " → ".join(path))

print(f"Total distance: {cost} meters")

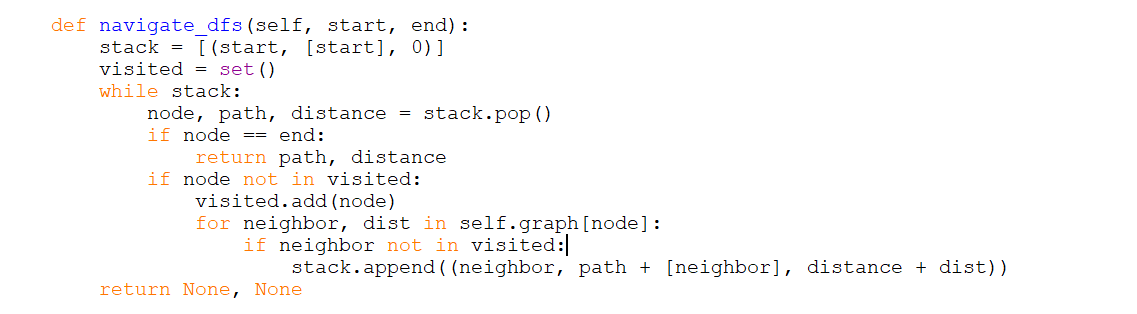
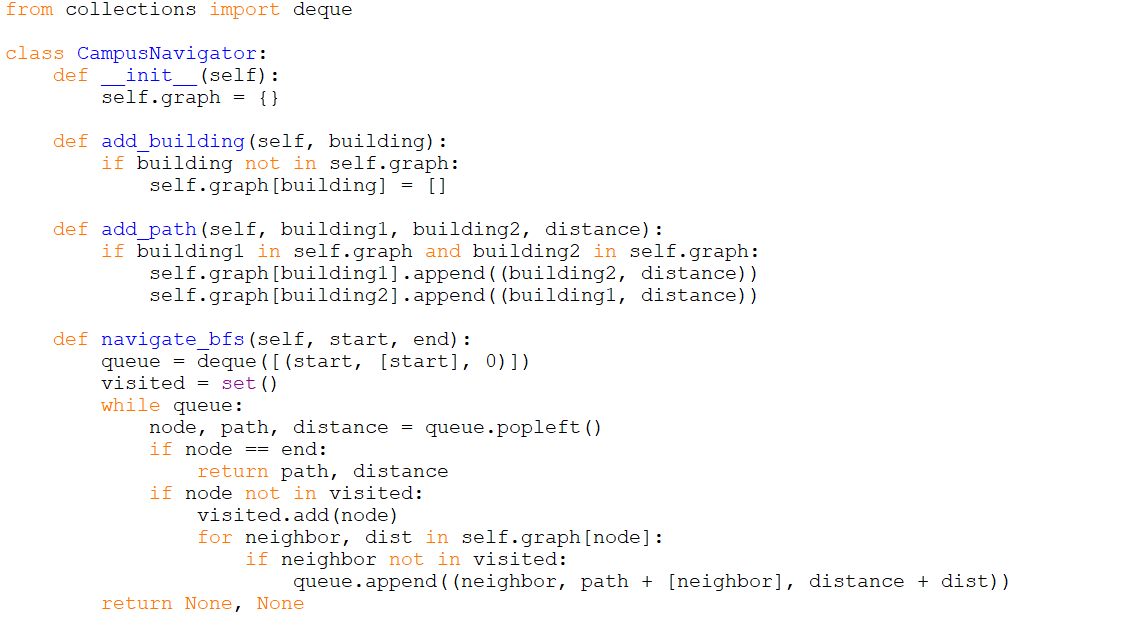
else:

print(f"\n No path found from {start} to {target}.")

except ValueError:

print("\n Please enter valid numbers only.")

DFS & BFS(INPUT)

A screenshot of a computer

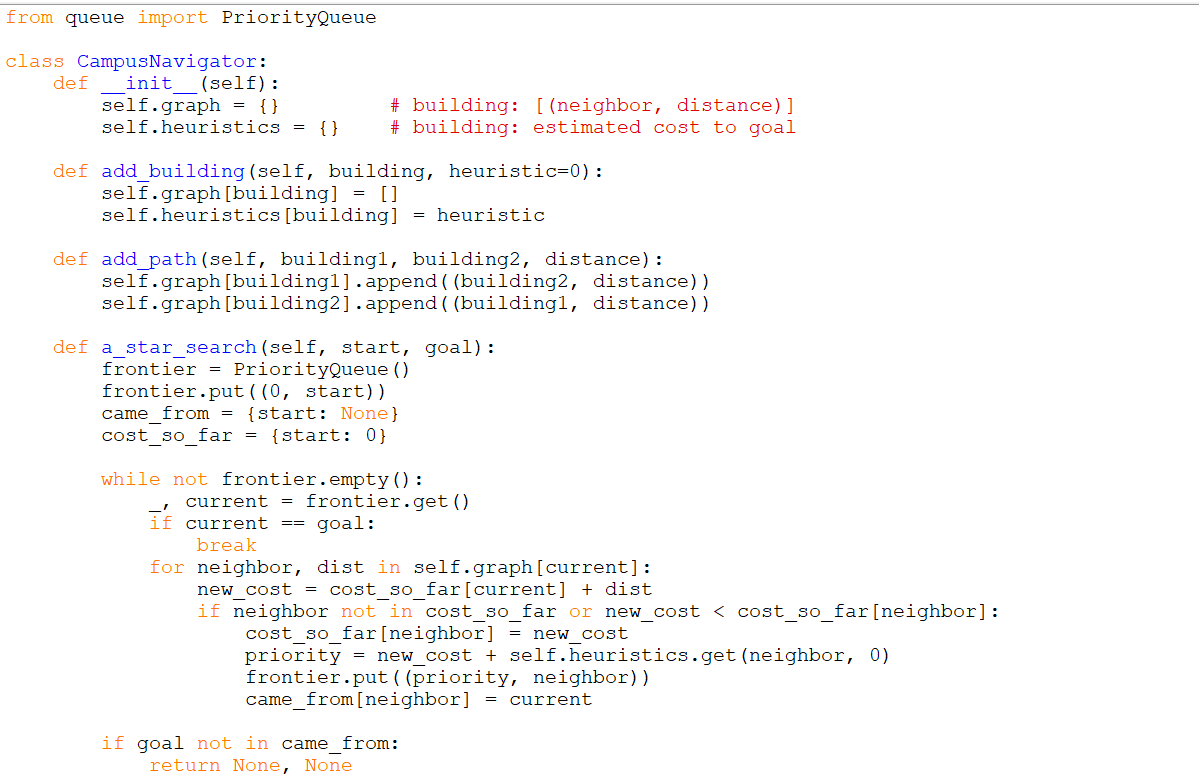
AI-generated content may be incorrect.

OUTPUT

A screen shot of a computer

AI-generated content may be incorrect.

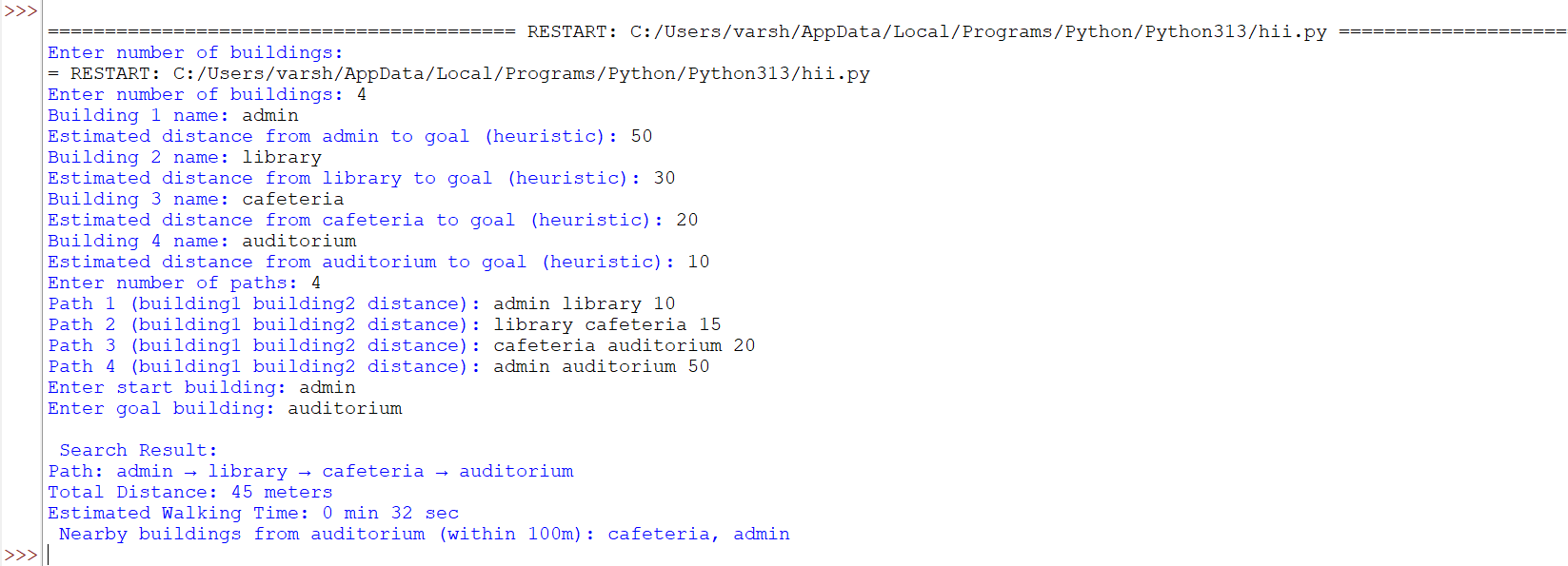
A\*

A computer screen shot of a computer code

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

OUTPUT



CODE

from queue import PriorityQueue

class CampusNavigator:

def \_\_init\_\_(self):

self.graph = {} # building: [(neighbor, distance)]

self.heuristics = {} # building: estimated cost to goal

def add\_building(self, building, heuristic=0):

self.graph[building] = []

self.heuristics[building] = heuristic

def add\_path(self, building1, building2, distance):

self.graph[building1].append((building2, distance))

self.graph[building2].append((building1, distance))

def a\_star\_search(self, start, goal):

frontier = PriorityQueue()

frontier.put((0, start))

came\_from = {start: None}

cost\_so\_far = {start: 0}

while not frontier.empty():

\_, current = frontier.get()

if current == goal:

break

for neighbor, dist in self.graph[current]:

new\_cost = cost\_so\_far[current] + dist

if neighbor not in cost\_so\_far or new\_cost < cost\_so\_far[neighbor]:

cost\_so\_far[neighbor] = new\_cost

priority = new\_cost + self.heuristics.get(neighbor, 0)

frontier.put((priority, neighbor))

came\_from[neighbor] = current

if goal not in came\_from:

return None, None

path = []

node = goal

while node:

path.append(node)

node = came\_from[node]

path.reverse()

return path, cost\_so\_far[goal]

def nearby\_buildings(self, building, radius=100):

return [b for b, dist in self.graph[building] if dist <= radius]

def main():

navigator = CampusNavigator()

# Input buildings and heuristics

n = int(input("Enter number of buildings: "))

for i in range(n):

name = input(f"Building {i+1} name: ").strip()

heuristic = int(input(f"Estimated distance from {name} to goal (heuristic): "))

navigator.add\_building(name, heuristic)

# Input paths

m = int(input("Enter number of paths: "))

for i in range(m):

b1, b2, dist = input(f"Path {i+1} (building1 building2 distance): ").split()

navigator.add\_path(b1, b2, int(dist))

# Input start and goal

start = input("Enter start building: ").strip()

goal = input("Enter goal building: ").strip()

path, cost = navigator.a\_star\_search(start, goal)

print("\n Search Result:")

if path:

print("Path:", " → ".join(path))

print(f"Total Distance: {cost} meters")

speed = 1.4 # meters per second

time\_sec = cost / speed

print(f"Estimated Walking Time: {int(time\_sec // 60)} min {int(time\_sec % 60)} sec")

nearby = navigator.nearby\_buildings(goal)

if nearby:

print(f" Nearby buildings from {goal} (within 100m): {', '.join(nearby)}")

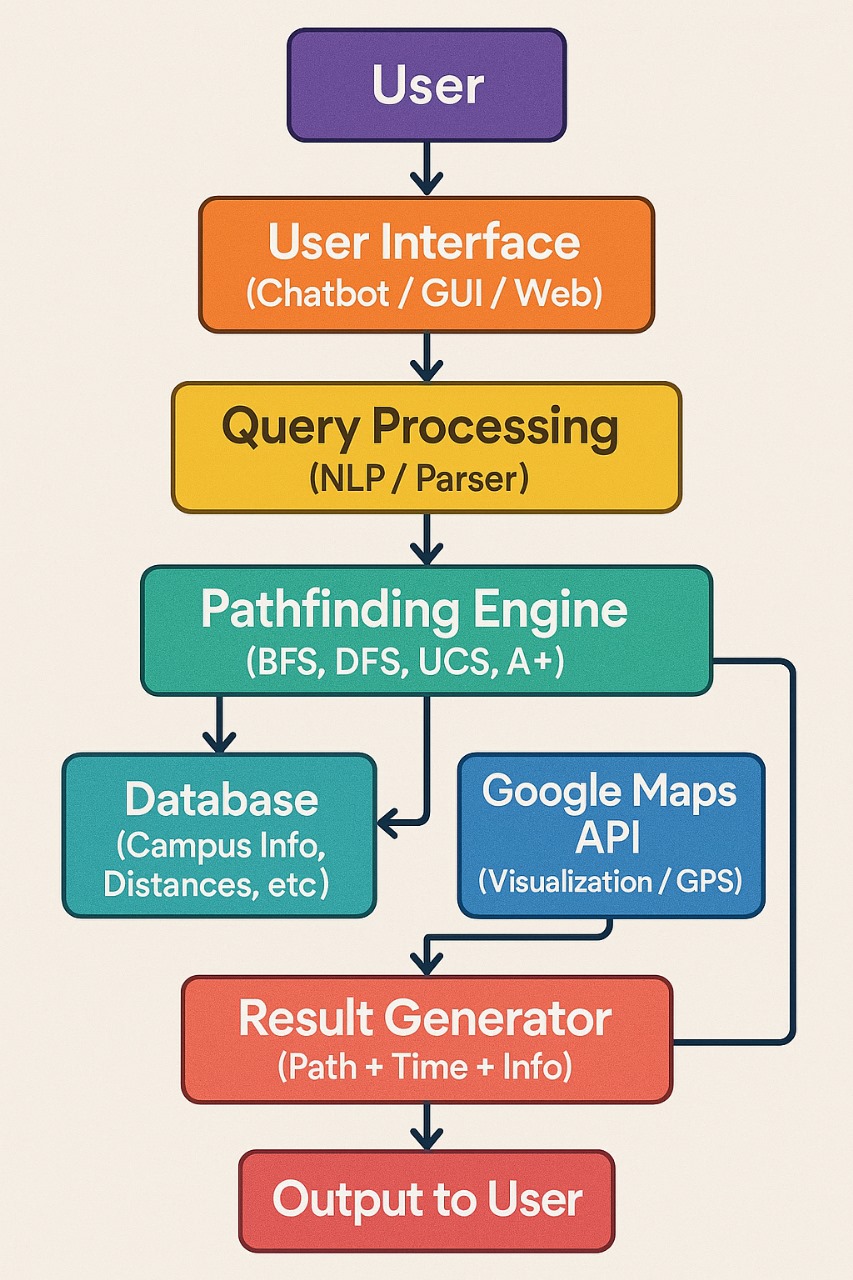
else:

print("No path found.")

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

main()

**SYSTEM DESIGN AND ARCHITECTURE**



|  |  |  |
| --- | --- | --- |
| Module | Purpose | Feature |
| User Interface | Engage users via chatbot,GUI,or web | Adaptive UI based on user type |
| Query processing | NLP based parsing of user queries | Context aware logic |
| Pathfinding Engine | Implements BFS, DFS, UCS, A\* for optimal pathfinding | Custom heuristics |
| Database | Stores campus data, distances, building info | Dynamic updates from user feedback |
| Google Maps API | Visualizes routes and syncs with GPS | Campus overlays |
| Result Generator | Synthesizes path, time, and contextual info | “Why this route?” explanations |
| Output to User | Presents final results in interactive format | Shareable route cards, voice-guided directions |

USC WITH STAFF ADRESS

from queue import PriorityQueue

# ---------- UCS Algorithm ----------

def ucs(graph, start, target):

if start == target:

return True, [start], 0

visited = set()

priority\_queue = PriorityQueue()

priority\_queue.put((0, start, [start]))

while not priority\_queue.empty():

cost, node, path = priority\_queue.get()

if node == target:

return True, path, cost

if node not in visited:

visited.add(node)

for neighbor, edge\_cost in graph.get(node, {}).items():

if neighbor not in visited:

priority\_queue.put((cost + edge\_cost, neighbor, path + [neighbor]))

return False, [], 0

# ---------- Location Mapper ----------

def find\_contact\_location(department, subunit=None):

key = subunit if subunit else department

return contact\_locations.get(key, "Unknown")

# ---------- Campus Distances ----------

distances = {

"Admin Block": {"Security Entrance": 80, "Library": 16, "Auditorium": 17, "Academic Block": 126, "Cafeteria": 35, "Main Gate": 160, "Exit Gate": 220, "Medical": 80, "Hostel": 506, "Mini Mart": 434, "Food Court": 355, "Sports": 850},

"Academic Block": {"Security Entrance": 240, "Library": 46, "Auditorium": 150, "Cafeteria": 27, "Main Gate": 210, "Exit Gate": 250, "Medical": 85, "Hostel": 380, "Mini Mart": 410, "Food Court": 192, "Admin Block": 126, "Sports": 561},

"Library": {"Security Entrance": 100, "Auditorium": 3, "Academic Block": 46, "Cafeteria": 63, "Main Gate": 275, "Exit Gate": 240, "Medical": 75, "Hostel": 450, "Mini Mart": 465, "Food Court": 470, "Admin Block": 16, "Sports": 857},

"Cafeteria": {"Security Entrance": 130, "Library": 63, "Auditorium": 10, "Academic Block": 27, "Main Gate": 276, "Exit Gate": 270, "Medical": 90, "Hostel": 520, "Mini Mart": 585, "Food Court": 355, "Admin Block": 35, "Sports": 920},

"Auditorium": {"Security Entrance": 120, "Library": 3, "Academic Block": 150, "Cafeteria": 10, "Main Gate": 286, "Exit Gate": 260, "Medical": 80, "Hostel": 510, "Mini Mart": 574, "Food Court": 344, "Admin Block": 17, "Sports": 930},

"Food Court": {"Security Entrance": 471, "Library": 470, "Auditorium": 344, "Academic Block": 192, "Cafeteria": 355, "Main Gate": 699, "Exit Gate": 609, "Medical": 405, "Hostel": 816, "Mini Mart": 836, "Admin Block": 355, "Sports": 427},

"Security Entrance": {"Admin Block": 80, "Academic Block": 240, "Library": 100, "Cafeteria": 130, "Auditorium": 120, "Food Court": 471, "Main Gate": 150, "Exit Gate": 180, "Medical": 90, "Hostel": 500, "Mini Mart": 520, "Sports": 880},

"Main Gate": {"Admin Block": 160, "Academic Block": 210, "Library": 275, "Cafeteria": 276, "Auditorium": 286, "Food Court": 699, "Security Entrance": 150, "Exit Gate": 100, "Medical": 180, "Hostel": 600, "Mini Mart": 620, "Sports": 900},

"Exit Gate": {"Admin Block": 220, "Academic Block": 250, "Library": 240, "Cafeteria": 270, "Auditorium": 260, "Food Court": 609, "Security Entrance": 180, "Main Gate": 100, "Medical": 160, "Hostel": 580, "Mini Mart": 600, "Sports": 870},

"Medical": {"Admin Block": 80, "Academic Block": 85, "Library": 75, "Cafeteria": 90, "Auditorium": 80, "Food Court": 405, "Security Entrance": 90, "Main Gate": 180, "Exit Gate": 160, "Hostel": 490, "Mini Mart": 510, "Sports": 840},

"Hostel": {"Admin Block": 506, "Academic Block": 380, "Library": 450, "Cafeteria": 520, "Auditorium": 510, "Food Court": 816, "Security Entrance": 500, "Main Gate": 600, "Exit Gate": 580, "Medical": 490, "Mini Mart": 200, "Sports": 300},

"Mini Mart": {"Admin Block": 434, "Academic Block": 410, "Library": 465, "Cafeteria": 585, "Auditorium": 574, "Food Court": 836, "Security Entrance": 520, "Main Gate": 620, "Exit Gate": 600, "Medical": 510, "Hostel": 200, "Sports": 320},

"Sports": {"Admin Block": 850, "Academic Block": 561, "Library": 857, "Cafeteria": 920, "Auditorium": 930, "Food Court": 427, "Security Entrance": 880, "Main Gate": 900, "Exit Gate": 870, "Medical": 840, "Hostel": 300, "Mini Mart": 320}

}

# ---------- Department Contacts ----------

contacts = {

"Admin": {

"Admission": [{"name": "Thanushree", "phone": "08031233133"}, {"name": "Sri Vijay Kumar", "phone": "08031233103"}],

"Registrar": [{"name": "Sri Gautam", "phone": "08031233104"}, {"name": "Sri Dhanashri", "phone": "08031233101"}],

"Communication": {"name": "Sri Chandrashekar", "phone": "08031233107"},

"Finance": {"name": "Sri Sudheerda K.M", "phone": "08031233102"},

"IT Support": {"name": "Sri Sachin Goni", "phone": "08031233109"}

},

"Academic": {

"VC Office": [

{"name": "Shilparanganathra", "phone": "08031233122"},

{"name": "Ashwin Kumar", "phone": "Contact VC Office"},

{"name": "Subhant T. Joshi", "phone": "08031233104"},

{"name": "Dr. Padmavathi B.S", "phone": "Contact VC Office"},

{"name": "Dr. Vineeth Paleni", "phone": "Contact VC Office"}

]

},

"Library": {"name": "Bharathkumar V", "phone": "8861775721"},

"Medical Center": {"name": "Dr. Anjana", "phone": "08031233103"},

"Sports & P.E": {"name": "Sri Hemanth", "phone": "9449141869"},

"Hostel": {

"Girls": {"name": "Jayshree", "phone": "9513228510"},

"Boys": {"name": "Ullas Kallur", "phone": "08031233100"}

}

}

# ---------- Location Mapping ----------

contact\_locations = {

"Admission": "Admin Block",

"Registrar": "Admin Block",

"Communication": "Admin Block",

"Finance": "Admin Block",

"IT Support": "Admin Block",

"VC Office": "Academic Block",

"Library": "Library",

"Medical Center": "Medical",

"Sports & P.E": "Sports",

"Girls": "Hostel",

"Boys": "Hostel"

}

# ---------- Main Program ----------

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

locations = list(distances.keys())

while True:

print("\nAvailable campus locations:")

for i, loc in enumerate(locations, start=1):

print(f"{i}. {loc}")

print("\nChoose an option:")

print("1. View contact details")

print("2. Navigate to a location")

print("3. Exit")

choice = input("Enter 1, 2, or 3: ").strip()

if choice == "1":

print("\nAvailable departments:")

for d in contacts.keys():

print(f"- {d}")

dept = input("\nEnter department: ").strip().title()

if dept in contacts:

subunits = contacts[dept]

if isinstance(subunits, dict):

print(f"\nAvailable units in {dept}:")

for i, unit in enumerate(subunits.keys(), start=1):

print(f"{i}. {unit}")

unit\_input = input("Enter unit name or number: ").strip().lower()

# Try matching by number

try:

unit\_index = int(unit\_input) - 1

selected\_unit = list(subunits.keys())[unit\_index]

except:

# Fallback to fuzzy name match

selected\_unit = None

for unit in subunits.keys():

if unit\_input in unit.lower():

selected\_unit = unit

break

if selected\_unit:

contact\_info = subunits[selected\_unit]

print(f"\nContact details for {selected\_unit}:")

if isinstance(contact\_info, list):

for contact in contact\_info:

print(f"- {contact['name']}, Phone: {contact['phone']}")

else:

print(f"- {contact\_info['name']}, Phone: {contact\_info['phone']}")

else:

print("Unit not found.")

else:

print(f"\nContact: {subunits['name']}, Phone: {subunits['phone']}")

else:

print("Department not found.")

elif choice == "2":

try:

start\_index = int(input("\nEnter the number for your starting location: ")) - 1

target\_index = int(input("Enter the number for your destination: ")) - 1

if start\_index not in range(len(locations)) or target\_index not in range(len(locations)):

print("\nInvalid selection. Please choose numbers from the list.")

else:

start = locations[start\_index]

target = locations[target\_index]

if start == target:

print(f"\nYou're already at {start}! No travel needed.")

else:

found, path, cost = ucs(distances, start, target)

if found:

print(f"\nPath from {start} to {target}:")

print(" → ".join(path))

print(f"Total distance: {cost} meters")

else:

print("\nNo path found.")

except ValueError:

print("\nPlease enter valid numbers only.")

elif choice == "3":

print("\nThank you for using the campus assistant. Goodbye!")

break

else:

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

**SAMPLE OUTPUT**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**A\*(with staff address)**

from queue import PriorityQueue

class CampusNavigator:

def \_\_init\_\_(self):

self.graph = {}

self.heuristics = {}

def add\_building(self, building, heuristic=0):

self.graph[building] = []

self.heuristics[building] = heuristic

def add\_path(self, b1, b2, distance):

self.graph[b1].append((b2, distance))

self.graph[b2].append((b1, distance))

def a\_star\_search(self, start, goal):

frontier = PriorityQueue()

frontier.put((0, start))

came\_from = {start: None}

cost\_so\_far = {start: 0}

while not frontier.empty():

\_, current = frontier.get()

if current == goal:

break

for neighbor, dist in self.graph[current]:

new\_cost = cost\_so\_far[current] + dist

if neighbor not in cost\_so\_far or new\_cost < cost\_so\_far[neighbor]:

cost\_so\_far[neighbor] = new\_cost

priority = new\_cost + self.heuristics.get(neighbor, 0)

frontier.put((priority, neighbor))

came\_from[neighbor] = current

if goal not in came\_from:

return None, None

path = []

node = goal

while node:

path.append(node)

node = came\_from[node]

path.reverse()

return path, cost\_so\_far[goal]

contacts = {

"Admin": {

"Registrar": [

{"name": "Sri Gautam", "phone": "08031233104"},

{"name": "Sri Dhanashri", "phone": "08031233101"}

],

"Finance": {"name": "Sri Sudheerda K.M", "phone": "08031233102"}

},

"Library": {"name": "Bharathkumar V", "phone": "8861775721"},

"Medical": {"name": "Dr. Anjana", "phone": "08031233103"}

}

def main():

navigator = CampusNavigator()

print("Setup Campus Map")

n = int(input("Enter number of buildings: "))

for i in range(n):

name = input(f"Building {i+1} name: ").strip()

heuristic = int(input(f"Heuristic for {name} (estimated distance to goal): "))

navigator.add\_building(name, heuristic)

m = int(input("Enter number of paths: "))

for i in range(m):

print(f"Path {i+1}:")

b1 = input(" Building 1 name: ").strip()

b2 = input(" Building 2 name: ").strip()

dist = int(input(" Distance between them: "))

navigator.add\_path(b1, b2, dist)

while True:

print("\nChoose an option:")

print("1. View staff contact details")

print("2. Find shortest path using A\*")

print("3. Exit")

choice = input("Enter 1, 2, or 3: ").strip()

if choice == "1":

print("Available departments:")

for dept in contacts:

print(f"- {dept}")

dept\_input = input("Enter department: ").strip().title()

if dept\_input in contacts:

unit = contacts[dept\_input]

if isinstance(unit, dict):

print(f"Units in {dept\_input}:")

for i, sub in enumerate(unit.keys(), start=1):

print(f"{i}. {sub}")

sub\_input = input("Enter unit name or number: ").strip().lower()

try:

sub\_index = int(sub\_input) - 1

selected = list(unit.keys())[sub\_index]

except:

selected = None

for key in unit:

if sub\_input in key.lower():

selected = key

break

if selected:

info = unit[selected]

print(f"Contact details for {selected}:")

if isinstance(info, list):

for person in info:

print(f"- {person['name']}, Phone: {person['phone']}")

else:

print(f"- {info['name']}, Phone: {info['phone']}")

else:

print("Unit not found.")

else:

print(f"- {unit['name']}, Phone: {unit['phone']}")

else:

print("Department not found.")

elif choice == "2":

start = input("Enter start building: ").strip()

goal = input("Enter goal building: ").strip()

path, cost = navigator.a\_star\_search(start, goal)

if path:

print("Path found:")

print(" -> ".join(path))

print(f"Total Distance: {cost} meters")

speed = 1.4

time\_sec = cost / speed

print(f"Estimated Walking Time: {int(time\_sec // 60)} min {int(time\_sec % 60)} sec")

else:

print("No path found.")

elif choice == "3":

print("Exiting campus assistant.")

break

else:

print("Invalid option. Try again.")

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

main()

**SAMPLE OUTPUT**

**A screenshot of a computer program

AI-generated content may be incorrect.**

A screenshot of a computer program

AI-generated content may be incorrect.