S VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA,BELAGAVI – 590018

KARNATAKA

Mini Project Report
On
"Campus Navigation System Using Dijkstra's Algorithm"

SUBMITTED IN PARTIAL FULFILLMENT OF THE ASSIGNMENT

FOR THE Analysis & Design of Algorithms (BCS401)

COURSE OF IV SEMESTER

Submitted by

Name: Meghana N

USN: 1CG22CS064

Guide:

Mr.Asif Ulla Khan M. Tech.

Asst. Prof., Dept. of CSE

CIT, Gubbi.

HOD:

Dr. Shantala C P PhD.,

Head, Dept. of CSE

CIT, Gubbi.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Channabasaveshwara Institute of Technology

(Affiliated to VTU, Belgaum & Approved by AICTE, New Delhi)

(NAAC Accredited & ISO 9001:2015 Certified Institution)

NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka

2023-24

Rubric – B.E. Mini-Project [BCS401]

Course outcome	Rubric/Level	Excellent (91-100%)	Good (81-90%)	Average (61-80%)	Moderate (40-60%)	Score
CO1	Identification of project proposal (05 Marks)					
CO2	Design and Implementation (05 Marks)					
CO3	Presentation skill (05 Marks)					
CO 4	Individual or in a team development					
CO5	Report (05 Marks)					
Total						

Course outcome:

CO 1: Identification of project proposal which is relevant to subject of engineering.

CO 2: Design and implement proposed project methodology.

CO 3: Effective communication skill to assimilate their project work.

CO 4: Work as an individual or in a team in development of technical projects.

CO 5: Understanding overall project progress and performance.

Student Signature **Faculty signature**

ABSTRACT

Dijkstra Algorithm is one of the most famous algorithms in computer science. There might be several possible routes to reach a destination point. If someone doesn't travel through optimal path, it will consume more time and energy. This project aims to determine locations of the node that reflect all the nodes in the list, build the route by connecting nodes and evaluate the optimal path by using Dijkstra algorithm. Dijkstra's Algorithm is also known as a single source shortest path algorithm which is used to find the shortest distance/path from one node to another node in a graph. This algorithm can be used only for positive distances from one location to another.

CHAPTER 1:

INTRODUCTION

In our daily life, people commonly face many problems in finding which path leads. People usually explore every possible solution in finding their way within the campus, but not every solution can produce the optimal path. Shortest path problem is a problem in finding the fastest route or path from a directed graph. The “Campus Navigation” is used for navigating the user from one place to another within the campus. All navigation system takes user location as a starting point and gives the best optimal path to be followed. Every direction in a graph (map) have a cost to be calculated. This shortest path problem is a way to find a new route or path in a graph with a minimum sum of weight travelled through the direction. This shortest path problem is solved by using Dijkstra’s algorithm of finding the best edge path between vertices in a graph. There are several variations of algorithm that can be used to determine the node that was pursued based on the direction in given graph.

Variations of the shortest path can be distinguished from single-source objective, pair path and generalization. A pair of shortest path is finding the shortest path for two points of nodes. All pair of shortest path is a technique to find the shortest path among all directed nodes. Single-source shortest path is finding the shortest form travelled, starting from a certain node to all other nodes in the graph.

Single-objective shortest path problem is to find the shortest path from any node on the graph are directed to a single destination node. Intermediate shortest path is finding the shortest path between two nodes selected through other nodes. Generalization is significantly more efficient than the simple approach to run one-pair of shortest path algorithm on all pairs of vertices that are relevant.

CHAPTER 2:

PROBLEM STATEMENT

There exist many advanced navigation systems but most of them are unable to provide routes precisely as well as information of building within a region such as campus, shopping mall, hospital and etc.

Nowadays, as people are getting more and more connected to technology, they lost their human touch. Also, people feel more convenient to search for the problem themselves rather than asking someone for help.

An informative, reliable and precise guidance system is very important in this technological era. It should be able to navigate the user no matter the user is under the indoor or outdoor environment. The guidance system must be user-friendly and able to process data efficiently.

Since the size of any college/university campus can vary from 30 acres to anywhere around 200 acres, students spend majority of their time in travelling between different buildings. New students feel inconvenient to search their way inside the campus.

Therefore, a navigation system is required to find the optimal path within the campus and for the aforementioned problem.

CHAPTER 3:

IMPLEMENTATION

ALGORITHM:

Step 1: **Initialize the Graph**

- Create an empty graph G.
- Define a dictionary nodes with node names as keys and their positions as values.
- Add nodes to G using their positions from the nodes dictionary.
- Define a list edges with tuples containing two nodes and the weight (distance) between them.
- Add edges to G using the edges list.

Step 2: **Draw Campus Layout Function (draw_campus)**

- Retrieve the positions of nodes from the graph.
- Initialize a plot using matplotlib.
- For each node, draw a rectangle at its position and label it with the node name.
- For each edge, draw a line between the nodes it connects.
- Annotate each edge with its weight.
- If a path is provided, highlight the path by drawing thicker lines between the nodes in the path.
- Display the plot.

Step 3: **Find Shortest Path Function (find_shortest_path)**

- Use Dijkstra's algorithm to find the shortest path between the start and end nodes.
- Return the shortest path and the total distance.

Step 4: **Handle User Input Function (handle_user_input)**

- If the user selects option '1':
- Call draw_campus to visualize the graph.
- If the user selects option '2':
- Prompt the user to input the start and end nodes.
- If the nodes are valid, find the shortest path using find_shortest_path.
- Print the shortest path and total distance.
- Call draw_campus with the shortest path highlighted.
- If the user inputs an invalid option, print an error message.

Step 5: **Main Function**

- Continuously display a menu with options:
- "1. Visualize graph"
- "2. Find shortest path"
- "3. Exit"
- Get the user's choice.
- If the user chooses option '3', exit the program.
- Otherwise, call handle_user_input with the user's choice.

Step 6: **Run the Main Function**

- If the script is executed as the main program, call the main function.

PROGRAM:

```
import matplotlib.pyplot as plt
import networkx as nx
from matplotlib.patches import Rectangle

# Define the campus layout as a graph
G = nx.Graph()

# Add nodes with their positions
nodes = {
    'College Gate': (0, 0),
    'Canteen Block': (2, 4),
    'Admin Block': (6, 6),
    'CRC Block': (12,-6),
    'MC Block': (16, 2),
    'Library':(14,-8),
    'Auditorium':(10,-8),
    'Workshop & Mech Lab': (12, 6),
    'Hostel Block': (4, -12),
    'Cricket Ground': (2,-4),
    'Sports Room':(4,-10)
}

for node, position in nodes.items():
    G.add_node(node, pos=position)

# Add edges with weights (distances)
edges = [
    ('College Gate', 'Canteen Block', 1),
    ('Canteen Block', 'Admin Block', 1),
    ('Admin Block', 'Workshop & Mech Lab', 1),
    ('Workshop & Mech Lab', 'MC Block', 1),
    ('MC Block', 'CRC Block', 1),
    ('CRC Block', 'Library', 1),
    ('CRC Block', 'Auditorium', 1),
```

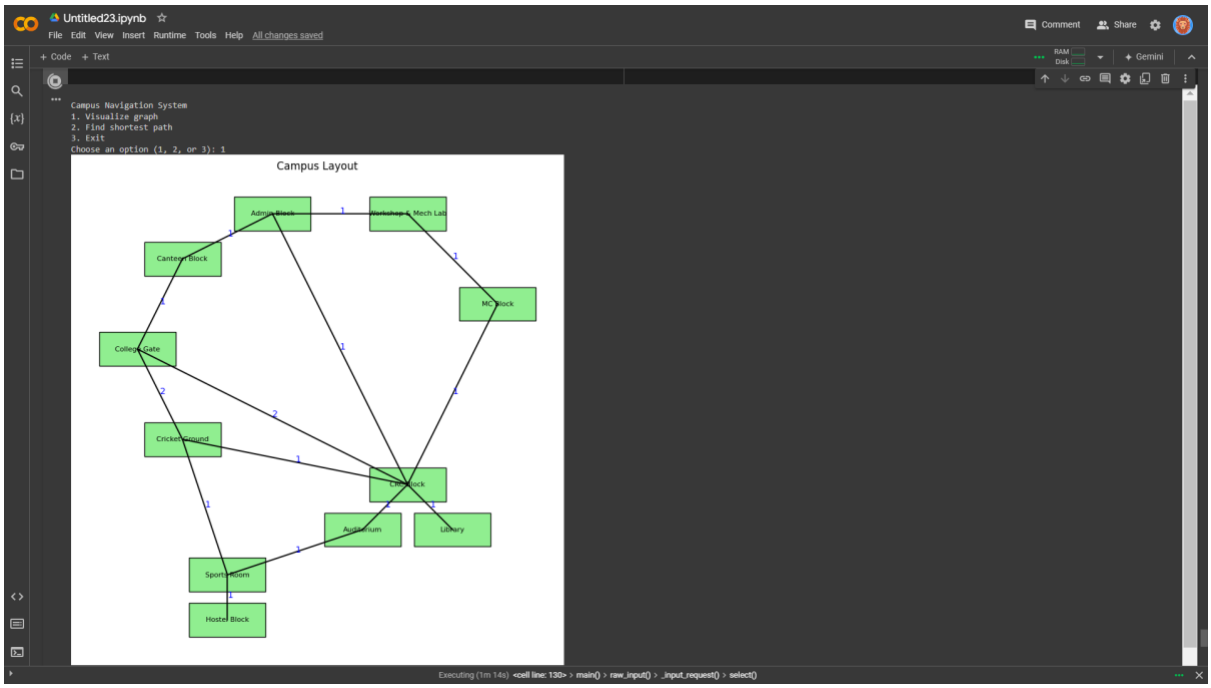
```
    ('College Gate','Cricket Ground',2),
    ('Cricket Ground','Sports Room',1),
    ('Sports Room', 'Hostel Block', 1),
    ('College Gate','CRC Block',2),
    ('Cricket Ground','CRC Block',1),
    ('Sports Room','Auditorium',1),
    ('Admin Block','CRC Block',1)

]

for edge in edges:
    G.add_edge(edge[0], edge[1], weight=edge[2])

# Function to draw the campus layout
def draw_campus(G, path=None):
    pos = nx.get_node_attributes(G, 'pos')
    fig, ax = plt.subplots(figsize=(14, 10))

    # Draw nodes as rectangles
    for node, (x, y) in pos.items():
        width = 3.4
        height = 1.5
        ax.add_patch(Rectangle((x - width/2, y - height/2), width, height, edgecolor='black',
facecolor='lightGreen'))
        ax.text(x, y, node, ha='center', va='center', fontsize=8)

    # Draw edges
    for edge in G.edges():
        x1, y1 = pos[edge[0]]
        x2, y2 = pos[edge[1]]
        ax.plot([x1, x2], [y1, y2], 'k-')

    # Draw edge labels (weights)
    edge_labels = nx.get_edge_attributes(G, 'weight')
    for (n1, n2), weight in edge_labels.items():
```

```
        x1, y1 = pos[n1]
        x2, y2 = pos[n2]
        ax.text((x1 + x2) / 2, (y1 + y2) / 2, weight, color='blue', fontsize=10, ha='left')

    # Highlight the path if given
    if path:
        path_edges = list(zip(path, path[1:]))
        for edge in path_edges:
            x1, y1 = pos[edge[0]]
            x2, y2 = pos[edge[1]]
            ax.plot([x1, x2], [y1, y2], 'r-', linewidth=2)

    plt.title("Campus Layout")
    ax.set_aspect('equal')
    plt.axis('off')
    plt.show()

# Function to find the shortest path and total distance using Dijkstra's algorithm
def find_shortest_path(G, start, end):
    path = nx.dijkstra_path(G, start, end)
    total_distance = nx.dijkstra_path_length(G, start, end)
    return path, total_distance


# Function to handle user input and call the appropriate functions
def handle_user_input(option):
    if option == '1':
        draw_campus(G)
    elif option == '2':
        start_node = input("Enter the start node: ")
        end_node = input("Enter the end node: ")

        if start_node not in nodes or end_node not in nodes:
            print("Invalid start or end node. Please make sure the nodes exist in the campus layout.")
```

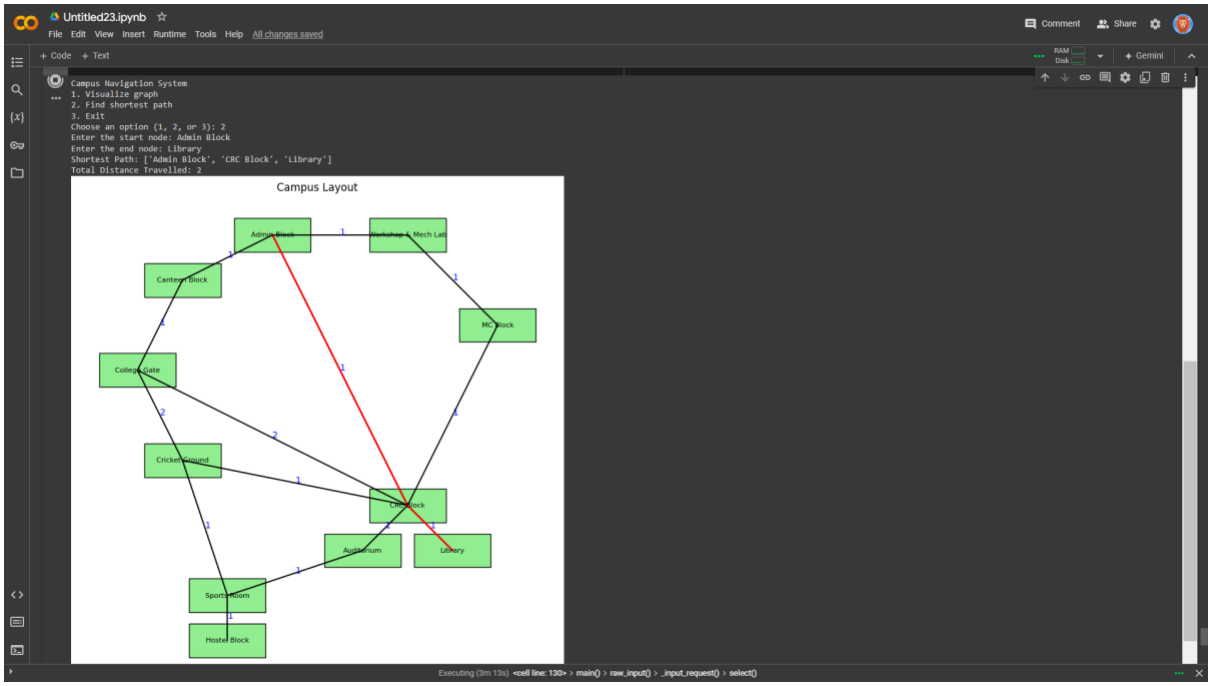
```
        else:
            shortest_path, total_distance = find_shortest_path(G, start_node, end_node)
            print("Shortest Path:", shortest_path)
            print("Total Distance Travelled:", total_distance)
            draw_campus(G, shortest_path)
    else:
        print("Invalid option.")


# Main function to present user options
def main():
    while True:
        print("\nCampus Navigation System")
        print("1. Visualize graph")
        print("2. Find shortest path")
        print("3. Exit")

        option = input("Choose an option (1, 2, or 3): ")

        if option == '3':
            print("Exiting the system.")
            break
        else:
            handle_user_input(option)

# Run the main function
if __name__ == "__main__":
    main()
```

CHAPTER 4:

RESULT



After enter choice 1 it shows the campus layout.



After entering the choice 2, it shows the shortest distance from starting node to ending node by highlighting the path.

CHAPTER 5:

CONCLUSION

The campus navigation system program demonstrates the practical application of graph theory and visualization techniques in solving spatial problems. By representing the campus as a graph with nodes and edges, the program allows for efficient management and manipulation of campus layout data. The use of Dijkstra's algorithm ensures accurate and optimal pathfinding between any two locations. The clear visualization provided by the program makes it easy for users to understand the campus layout and the shortest routes between buildings.

This tool proves valuable for students, staff, and visitors by enhancing their ability to navigate the campus efficiently. Additionally, it has potential applications in event planning and emergency response by offering quick and reliable route information. The program's modular design allows for easy updates and extensions, making it a flexible solution for various campus navigation needs.

REFERENCE

1. https://www.geeksforgeeks.org/courses
2. https://github.com//