JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA
B.TECH SEMESTER-3

JAYPEE MAPS

Supervision of:
Dr. Ankita Wadhwa

Dr. Bhawna Saxena

Submitted by:
1. Name- Shantanu Rai
   Enrollment No.-22103090
2. Name- Isha Singh
   Enrollment No.-22103111
3. Name- Pratham
   Enrollment No.-22103120

INDEX

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our teacher Dr Bhawna Saxena and Dr Ankita Wadhwa who gave us the golden opportunity to do this wonderful project on Data Structures which also helped us in doing a lot of research and we came to know about so many new things .

Secondly we would also like to thank our seniors who helped us a lot in finishing this project within the limited time.

PROBLEM STATEMENT

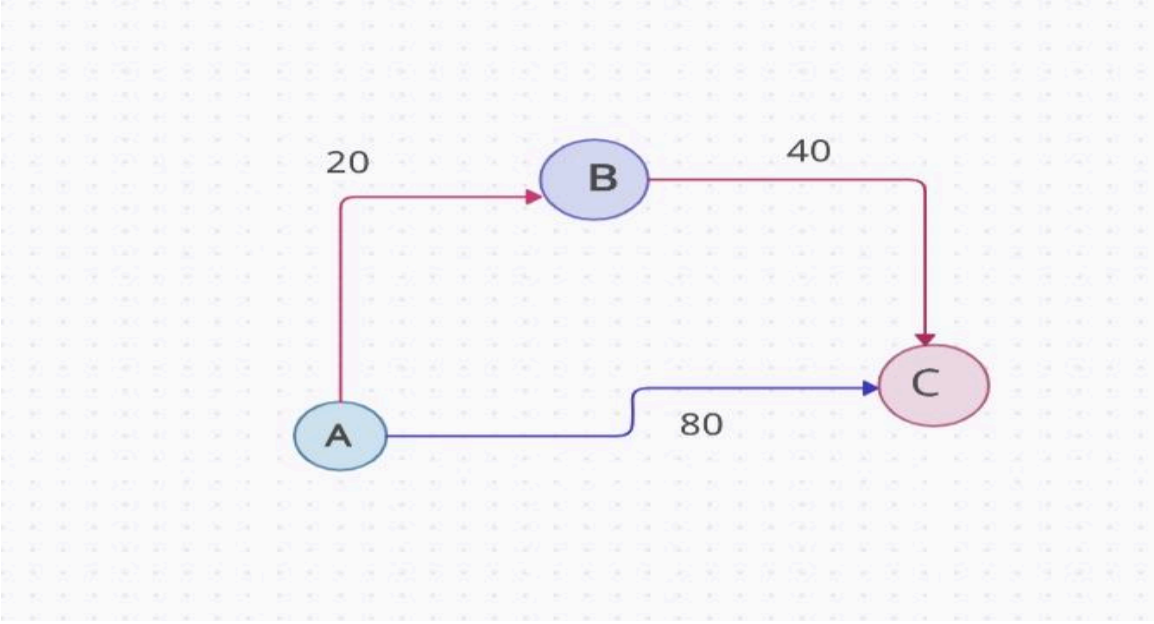
The goal of the project is to create "Jaypee Maps," a comprehensive navigation system that will improve accessibility and convenience of use on college campus. The goal of this system is to give users an efficient way to figure out the quickest route between any two points on the campus**.**

Features Expected:

•**User-Friendly Interface:** A user-friendly interface allowing users to input their desired locations and obtain the shortest path between them.

•**Graph Representation:** Modeling the campus layout using adjacency matrices and lists to capture connections and distances between locations.

•**Dijkstra's Algorithm:** Implementation of Dijkstra's algorithm for determining the shortest path between locations.



•**Optimized Data Structures:** Utilization of various data structures such as adjacency lists, matrices, queues, priority queues (implemented via Binary Heap and Fibonacci Heap), AVL Trees (for creating sets), and Linked Lists (for implementing queues) to enhance efficiency and optimize pathfinding algorithms.

•**Efficient Search and Pathfinding:** Ensuring efficient pathfinding and navigation through the college campus.

Purpose:

The primary aim of this project is to offer an efficient and reliable navigational tool within the college premises. It intends to minimize travel time between different locations, benefiting students, faculty, and visitors by facilitating convenient and optimized navigation. Additionally, the project aims to leverage various data structures and algorithms, emphasizing their roles in creating an effective navigational system.

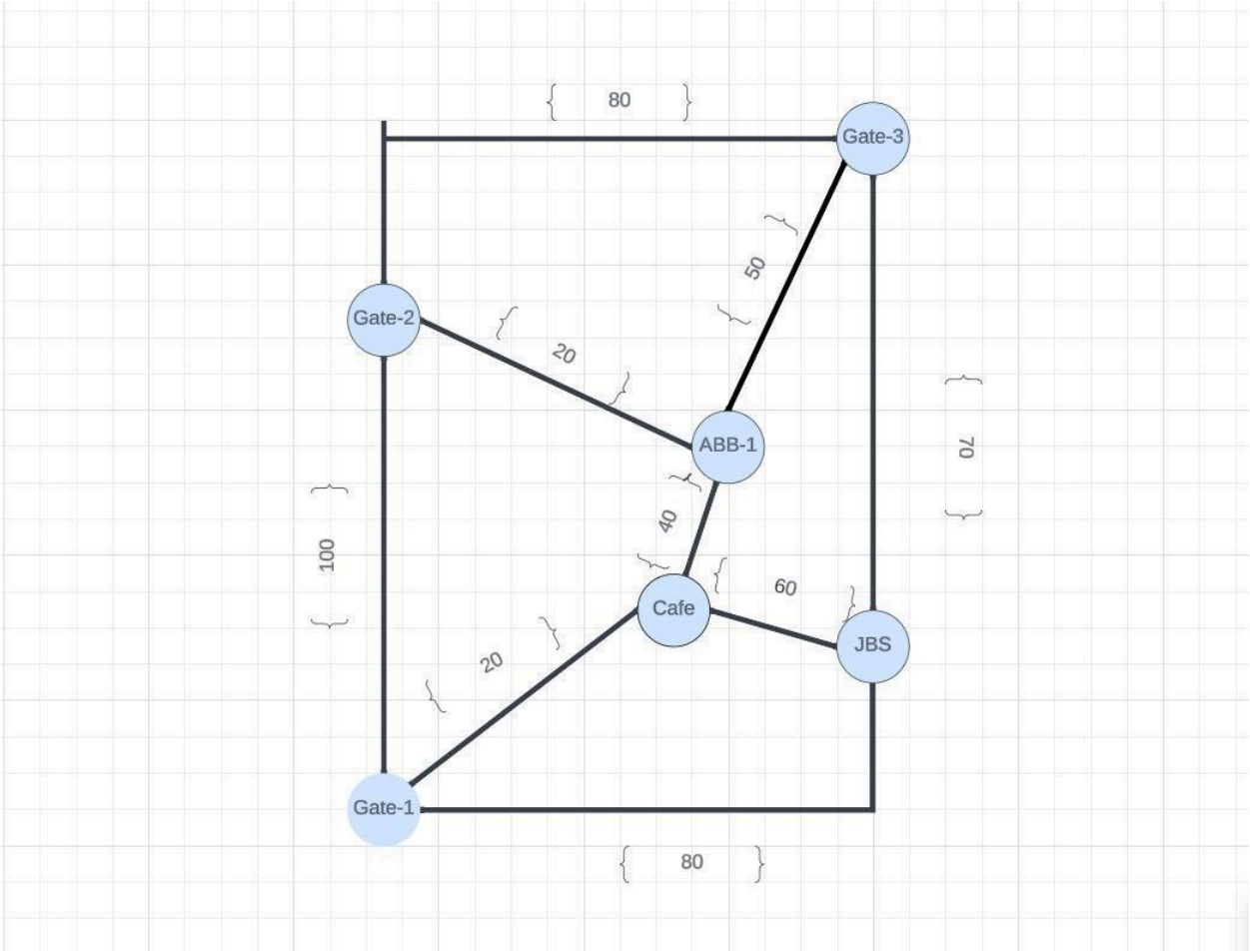
Deliverables:

•A fully functional navigational system, "Jaypee Maps," allowing users to input their desired locations and obtain the shortest path between them.

•Implementation details, including code snippets, algorithm flowcharts, and an in-depth project report highlighting the methodologies, data structures used, algorithmic implementations, complexities, and performance analysis.

Methodology to solve problem/Solution Approach: Explanation along with flowchart:

The project employs graph-based models to represent the campus layout. Utilizing adjacency matrices and adjacency lists, the system captures the connections and distances between various locations within the college campus.



Graph Model:

•**Node Mapping:** Assigning unique identifiers or labels to different locations within the campus, treating each location as a node in the graph.
•**Edge Representation:** Defining edges between nodes to represent pathways, walkways, or roads connecting different locations.

•**Distance Calculation:** Associating each edge with the corresponding distance between connected nodes.

Purpose of Graph-Based Models:

•**Comprehensive Campus Layout:** The graph-based models aim to provide a comprehensive and accurate representation of the spatial layout of the college campus.
•**Efficient Pathfinding:** By capturing connections and distances between locations, these models facilitate efficient algorithms for finding the shortest paths between any two points within the campus.

Advantages of Matrix and List Representation:

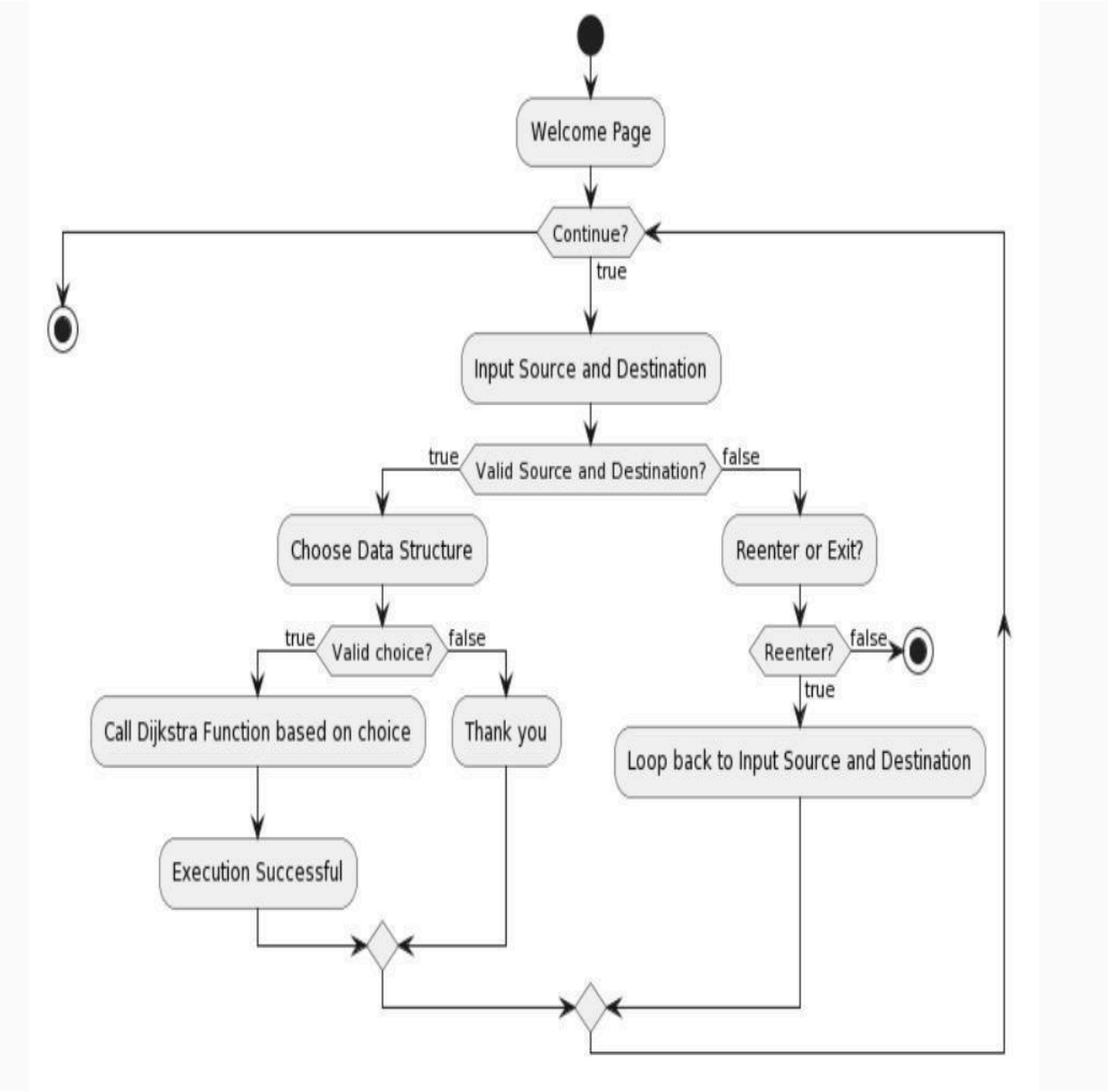
•**Matrix Advantages:** Matrices provide a clear visualization of the entire connectivity between nodes. They are particularly useful for dense graphs where most nodes are interconnected.
•**List Advantages:** Lists, on the other hand, are efficient for sparse graphs where fewer connections exist between nodes. They offer better memory utilization by only storing connections that exist.

Flexibility and Scalability:

•**Scalability:** Both matrix and list representations offer scalability, accommodating the addition or removal of nodes and connections within the campus layout.
•**Adaptability:** These representations allow for updates or modifications in the campus layout over time, ensuring the system's adaptability to changes in the physical infrastructure.
Systematic Navigation:
•**Foundation for Pathfinding:** These graph-based models serve as the foundational structures for implementing pathfinding algorithms like Dijkstra's, enabling efficient navigation across the college campus by identifying the shortest routes between any two given points.

In essence, the use of adjacency matrices and adjacency lists forms the backbone of the project's solution approach, enabling an accurate and efficient representation of the college campus layout and facilitating optimized pathfinding mechanisms for navigation.

FLOWCHART


Welcome Page
Continue?
true
Input Source and Destination
Valid Source and Destination?
true
false
Choose Data Structure
Reenter or Exit?
Valid choice?
true
false
Reenter?
false
true
Call Dijkstra Function based on choice
Thank you
Loop back to Input Source and Destination
Execution Successful

Data Structures Used

1. Binary Heap & Fibonacci Heap:

•**Purpose:** These are used for implementing priority queues efficiently.
•**Explanation:** Binary heaps offer simplicity and fast operations, like extracting the minimum element.
•**Usage Rationale:** Fibonacci heaps are more complex but provide better amortized performance for decrease-key operations, crucial in Dijkstra's algorithm when updating node distances.

2. Priority Queue:

•**Purpose:** Essential for Dijkstra's algorithm to prioritize nodes based on their distance from the source.
•Explanation: A priority queue efficiently manages nodes according to their distances from the source node.
•Usage Rationale: It ensures the algorithm selects the shortest path candidates effectively.

3. Sets:

•Purpose: Used to maintain unique elements, often for tracking visited nodes in graph traversal algorithms.
•Explanation: Sets help prevent repeated processing of nodes, ensuring efficiency and correctness in algorithms.
•Usage Rationale: Avoiding duplicate processing saves computation time and prevents infinite loops in pathfinding.

4. Queue:

•Purpose: Utilized for managing nodes during breadth-first search or similar algorithms.
•Explanation: Queues follow a first-in, first-out (FIFO) approach, crucial in certain algorithms like breadth-first search.
•Usage Rationale: Helps explore neighboring nodes in a controlled manner without missing potential paths.

5.Graph:

•Purpose: Represents the college map and the connections between locations.

•Explanation: Graphs model relationships between locations (nodes) and paths (edges).
•Usage Rationale: Facilitates efficient pathfinding algorithms like Dijkstra's by storing the structure of the college map.

6. Linked List for Implementing Queue:

•Purpose: Used to implement a queue for managing nodes during algorithms like breadth-first search.
•Explanation: Linked lists offer efficient insertion and deletion at both ends (front and rear), crucial for queue operations.
•Usage Rationale: Each node in the linked list represents an element in the queue, enabling FIFO operations efficiently.

7. AVL Tree for Set:

•Purpose: Utilized for maintaining a set of visited nodes or unique elements in algorithms.
•Explanation: AVL trees ensure efficient searching, insertion, and deletion operations with self-balancing properties.
•Usage Rationale: By maintaining a balanced tree, AVL trees provide faster search times, crucial for checking node visitation status in graph traversal algorithms like Dijkstra's.

COMPARISON OF COMPLEXITY OF VARIOUS DATA STRUCTURE USED

1. Binary Heap vs. Fibonacci Heap:

•Binary Heap:
•Extract-Min Time Complexity: O(log V)
•Decrease Key Time Complexity: O(log V)
•Fibonacci Heap:
•Extract-Min Time Complexity: O(log V) amortized
•Decrease Key Time Complexity: O(1) amortized
•Conclusion:
•Binary heaps are simpler to implement and have better constant factors for small graphs.
•Fibonacci heaps are more efficient for graphs with many decrease key operations but have higher constant factors.

2. Linked List vs. Other Queue Implementations:

•Linked List:
•Enqueue/Dequeue Time Complexity: O(1)
•Other Queue Implementations (e.g., Priority Queue):

•Enqueue/Dequeue Time Complexity: O(log V) or O(1), depending on the implementation.

•Conclusion:

•Linked lists are simple and efficient for enqueuing and dequeuing but lack priority functionality.

•Priority queues, especially binary heaps, are often used for Dijkstra's algorithm due to efficient extraction of the minimum element.

3. AVL Tree vs. Alternative Sets:

•AVL Tree:

•Insert/Delete/Search Time Complexity: O(log V)

•Sets (e.g., Hash Set, Red-Black Tree):

•Insert/Delete/Search Time Complexity: O(log V) or O(1), depending on the implementation.

•Conclusion:

•AVL trees provide guaranteed logarithmic time complexity but may have higher constant factors.

•Alternative sets like hash sets or red-black trees might be more efficient in practice for certain scenarios.

4. Priority Queue vs. Queue:

•Priority Queue:

•Enqueue/Dequeue Time Complexity: O(log V) (for binary heap-based implementations)

•Queue (e.g., Linked List-Based Queue):

•Enqueue/Dequeue Time Complexity: O(1)

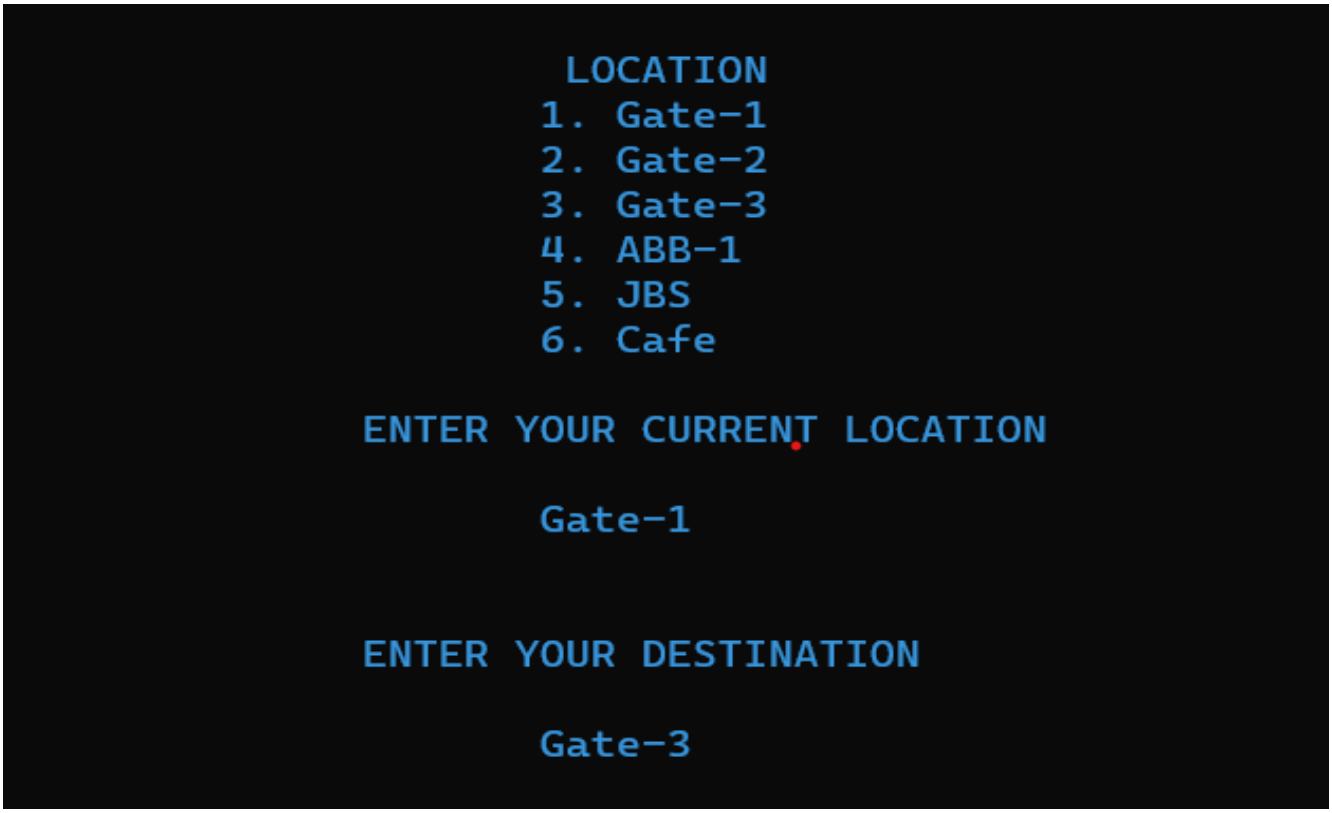
•Conclusion:

•Priority queues are generally more *efficient* for Dijkstra's algorithm due to their ability to efficiently extract the minimum element.

•Regular queues, while simpler, may be inefficient for Dijkstra's algorithm as they lack priority functionality.

In summary, the choice of data structures for Dijkstra's algorithm depends on the characteristics of the graph and the specific requirements of the application. The efficiency considerations include both theoretical time complexity and practical considerations such as constant factors and ease of implementation

Implementation and Results



```
INVALID LOCATION

Press 1 to renter the choices
Press 0 to exit
```

```
Tell Us How Do You want To Implement it

        1. QUEUE
        2. SET
        3. PRIORITY QUEUE
        4. BINARY HEAP
        5. FIBONACCI HEAP


Enter Your Choice:
```

```
Shortest path using QUEUE :

Gate-1->Cafe->ABB-1->Gate-3->Reached


Shortest Distance to reach Gate-3 from Gate-1 is: 110


Thanks for visiting
```

```
Shortest path using SET :

Gate-1->Cafe->ABB-1->Gate-3->Reached

Shortest Distance to reach Gate-3 from Gate-1 is: 110

Thanks for visiting
```

```
Shortest path using PRIORITY QUEUE :

Gate-1->Cafe->ABB-1->Gate-3->Reached

Shortest Distance to reach Gate-3 from Gate-1 is: 110

Thanks for visiting
```

```
Shortest path using BINARY HEAP:

Gate-1->Cafe->ABB-1->Gate-3->Reached

Shortest Distance to reach Gate-3 from Gate-1 is: 110

Thanks for visiting
```

```
Shortest path using FIBONACCI HEAP :

Gate-1->Cafe->ABB-1->Gate-3->Reached

Shortest Distance to reach Gate-3 from Gate-1 is: 110

Thanks for visiting
```

REFERENCES

- https://stackoverflow.com/questions/40247747/understanding-dijkstra-algorithm
- https://zriyansh.medium.com/c-stl-under-the-hood-76290ca58bcb
- https://www.freecodecamp.org/news/avl-tree-insertion-rotation-and-balance-factor/
- https://www.sanfoundry.com/cpp-program-implement-fibonacci-heap/
- https://takeuforward.org/data-structure/dijkstras-algorithm-using-priority-queue-g-32/
- https://www.geeksforgeeks.org/binary-heap/