DATA STRUCTURES AND ALGORITHMS
SEMESTER PROJECT
SEMESTER III

Prepared by:

1. Ayesha Afifa
2. Ayesha Maqsood
3. Areeba Habib

Submitted to:

• Dr.Sobia Khalid 2

Metro Bus Route Planner – Project Objectives & Solutions:

Project Overview:

This project implements a Metro Bus Route Planner using graph algorithms to find the shortest path between stations. It models metro stations as nodes and connections as weighted edges, allowing users to dynamically manage the network and compute optimal routes.

1. Graph Representation:

a) Adjacency List using Dynamic Edge Linked Lists :

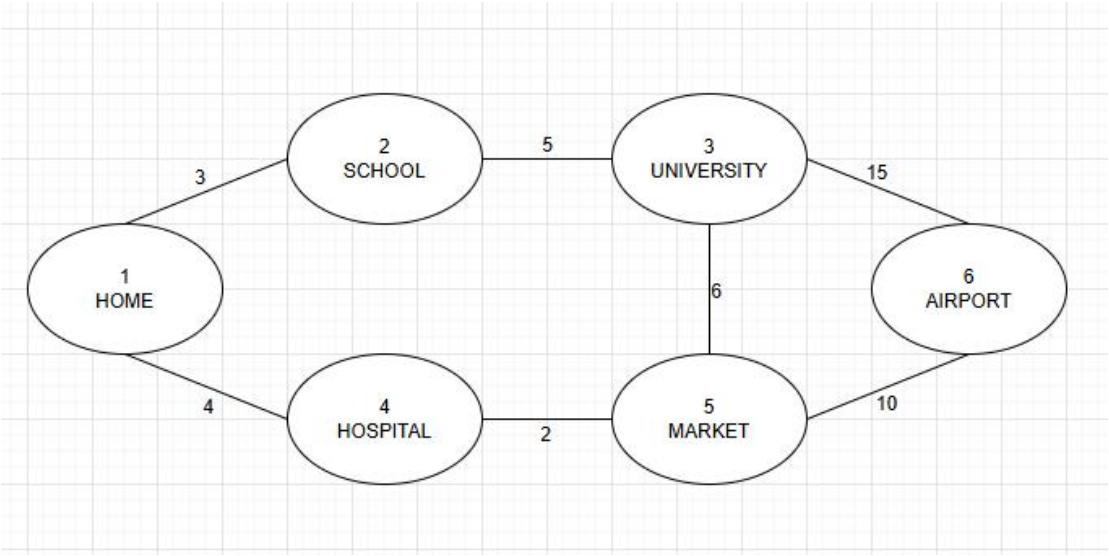
- The metro network is represented using an adjacency list.
- Each station stores a pointer to a linked list of edges.
- Every edge node contains:
    - **o destination:** Pointer to the connected station
    - **o weight:** Distance between stations (in kilometers)
    - **o next:** Pointer to the next edge in the list
- This structure:
    - **o** Provides fast access to neighboring stations
    - **o** Uses memory efficiently, especially for sparse networks

b) Circular Doubly Linked List (DLL) for Stations :

- All stations are stored in a circular doubly linked list.
- **Features:**
    - **o** Circular structure eliminates NULL pointer checks during traversal
    - **o** Doubly linked nodes allow forward and backward navigation
- This makes station management and traversal smooth and efficient.

c) Unique Station Identification :

- Each station has:
    - **o** A unique integer ID
    - **o** A descriptive station name
- This ensures easy identification, searching, and mapping within the system.



2. Shortest Path Algorithm:

Dijkstra's Algorithm :

- Dijkstra's Algorithm is used because:
    - **o** All edge weights (distances) are positive
    - **o** It guarantees the optimal shortest path
- A custom priority queue is used to:
    - **o** Always select the station with the smallest tentative distance
- Parent pointers are maintained to:
    - **o** Reconstruct and display the exact path from source to destination

3. Dynamic Network Management :

- Dynamic resizing of the adjacency list when new stations are added
- Fast station insertion using a circular doubly linked list
- Bidirectional edge insertion:
    - **o** Each connection is added in both directions
    - **o** This models two-way roads, forming an undirected graph

➢ DSA Concepts Implemented :

1. Priority Queue (Min-Priority Queue Simulation) :

Efficient selection of the station with the minimum distance during Dijkstra's algorithm.

Key Operations:

- Insert station with priority (push)
- Remove station with minimum distance (pop)
- Update distance priority (decrease key)

2. Circular Doubly Linked List (Stations) :

Efficient storage and traversal of metro stations.

Advantages:

- No NULL pointer checks due to circular nature
- Easy forward and backward traversal
- Efficient insertion and deletion

4. Adjacency List Graph Representation

- Edge** adjlist — an array of linked lists

Benefits:

- Space-efficient compared to adjacency matrix
- Fast neighbor access
- Supports dynamic edge insertion without full resizing

4. Dijkstra's Algorithm – Pseudocode:

```
Initialize all distances to ∞
Set source distance = 0
Push source into priority queue
While priority queue is not empty:
u = extract_min()
For each neighbor v of u:
If dist[v] > dist[u] + weight(u, v):
  dist[v] = dist[u] + weight(u, v)
    parent[v] = u
    update_priority(v, dist[v])
```

• Ensures efficient shortest path computation
• Parent pointers help in path visualization

➢ Strengths of the Implementation :

• Fully functional Dijkstra's Algorithm with a custom priority queue
• Dynamic graph operations (stations and connections added at runtime)
• Memory-safe design with proper cleanup of dynamically allocated memory
• Interactive menu-driven system for user-friendly interaction
• Supports bidirectional roads using an undirected graph model
• Provides clear path visualization, showing the sequence of stations on the shortest route.

➢ Code:

```
// DSA THEORY.cpp : Metro Bus Route Planner
#include <iostream>
#include <string>
using namespace std;

const int INF = 999;
class Stations;
class PriorityQueue
{
public:
	class Node
	{
	public:
		Stations* station;
		int distance;
		Node* next;
		Node(Stations* s, int d)
		{
			station = s;
			distance = d;
			next = nullptr;
		}
	};
	Node* front;
	PriorityQueue()
	{
		front = nullptr;
	}
	void push(Stations* station, int distance)
	{
		Node* newnode = new Node(station, distance);
		if (!front || distance < front->distance)
		{
			newnode->next = front;
			front = newnode;
		}
		else
		{
			Node* temp = front;
			while (temp->next && temp->next->distance <= distance)
				temp = temp->next;
			newnode->next = temp->next;
			temp->next = newnode;
		}
	}
	Stations* pop()
	{
```

```
                if (!front)
                        return nullptr;
                Node* temp = front;
                Stations* station = front->station;
                front = front->next;
                delete temp;
                return station;
        }
        bool is_empty()
        {
                return front == nullptr;
        }
        void update_priority(Stations* station, int newdistance)
        {
                Node* temp = front;
                Node* prev = nullptr;
                while (temp && temp->station != station)
                {
                        prev = temp;
                        temp = temp->next;
                }
                if (temp && newdistance < temp->distance)
                {
                        if (prev)
                                prev->next = temp->next;
                        else
                                front = temp->next;
                        delete temp;
                        push(station, newdistance);
                }
                else if (!temp)
                {
                        push(station, newdistance);
                }
        }
};

class Stations
{
public:
        string NAME;
        int ID;
        bool visited;
        int distance_from_source;
        Stations* next;
        Stations* pre;
        Stations* parent;
        Stations() : visited(false), distance_from_source(INF), next(nullptr),
pre(nullptr), parent(nullptr)
        {}
};
class Metro_Station
{
private:
        Stations* head;
        int count_stations;
        class Edge
        {
        public:
                Stations* destination;
                int weight;
                Edge* next;
```

```
		Edge(Stations* d, int w, Edge* n)
		{
			destination = d;
			weight = w;
			next = n;
		}
	};
	Edge** adjlist;
public:
	Metro_Station()
	{
		head = nullptr;
		count_stations = 0;
		adjlist = nullptr;
	}
	~Metro_Station()
	{
		for (int i = 0; i < count_stations; i++)
		{
			Edge* e = adjlist[i];
			while (e)
			{
				Edge* temp = e;
				e = e->next;
				delete temp;
			}
		}
		delete[] adjlist;
		if (head)
		{
			Stations* temp = head->next;
			while (temp != head)
			{
				Stations* t = temp;
				temp = temp->next;
				delete t;
			}
			delete head;
		}
	}
	void insert_station(string name, int id)
	{
		Stations* newstation = new Stations;
		newstation->NAME = name;
		newstation->ID = id;
		if (!head)
		{
			newstation->next = newstation;
			newstation->pre = newstation;
			head = newstation;
		}
		else
		{
			Stations* temp = head->pre;
			temp->next = newstation;
			newstation->pre = temp;
			newstation->next = head;
			head->pre = newstation;
		}
		count_stations++;
		Edge** newadj = new Edge * [count_stations];
		for (int i = 0; i < count_stations - 1; i++)
```

```
                newadj[i] = adjlist[i];
        newadj[count_stations - 1] = nullptr;
        delete[] adjlist;
        adjlist = newadj;
}
Stations* find_id(int id)
{
        if (!head)
                return nullptr;
        Stations* temp = head;
        do
        {
                if (temp->ID == id)
                        return temp;
                temp = temp->next;
        } while (temp != head);
        return nullptr;
}
int get_index(int id)
{
        if (!head)
                return -1;
        Stations* temp = head;
        int index = 0;
        do
        {
                if (temp->ID == id)
                        return index;
                temp = temp->next;
                index++;
        } while (temp != head);
        return -1;
}
void add_connections(int from, int to, int distance)
{
        Stations* A = find_id(from);
        Stations* B = find_id(to);
        if (!A || !B)
        {
                cout << "Station not found!" << endl;
                return;
        }
        int i = get_index(from);
        int j = get_index(to);
        adjlist[i] = new Edge(B, distance, adjlist[i]);
        adjlist[j] = new Edge(A, distance, adjlist[j]);
}
void reset_stations()
{
        if (!head)
                return;
        Stations* temp = head;
        do
        {
                temp->distance_from_source = INF;
                temp->visited = false;
                temp->parent = nullptr;
                temp = temp->next;
        } while (temp != head);
}
void find_shortest_path(int src, int dest)
{
```

```
		reset_stations();
		Stations* source = find_id(src);
		Stations* destination = find_id(dest);
		if (!source || !destination)
		{
			cout << "Invalid source or destination!" << endl;
			return;
		}
		source->distance_from_source = 0;
		PriorityQueue pq;
		pq.push(source, 0);
		while (!pq.is_empty())
		{
			Stations* u = pq.pop();
			if (!u)
				break;
			u->visited = true;
			int idx = get_index(u->ID);
			Edge* edge = adjlist[idx];
			while (edge)
			{
				Stations* v = edge->destination;
				int newdist = u->distance_from_source + edge-
>weight;
				if (newdist < v->distance_from_source)
				{
					v->distance_from_source = newdist;
					v->parent = u;
					pq.update_priority(v, newdist);
				}
				edge = edge->next;
			}
		}
		if (destination->distance_from_source == INF)
		{
			cout << "No path found!" << endl;
			return;
		}
		cout << "Shortest distance: " << destination-
>distance_from_source << " km" << endl;
		cout << "Path: ";
		Stations* temp = destination;
		while (temp)
		{
			cout << temp->NAME;
			if (temp->parent)
				cout << " -> ";
			temp = temp->parent;
		}
		cout << endl;
	}
	void display_stations_list()
	{
		if (!head)
			return;
		Stations* temp = head;
		cout << "Stations List:" << endl;
		do
		{
			cout << "ID: " << temp->ID << ", Name: " << temp->NAME <<
endl;
			temp = temp->next;
```

```
            } while (temp != head);
        }
        void display_connections()
        {
                if (!head) return;
                Stations* temp = head;
                int index = 0;
                cout << "Connections:" << endl;
                do
                {
                        cout << temp->NAME << " -> ";
                        Edge* e = adjlist[index];
                        while (e)
                        {
                                cout << e->destination->NAME << "(" << e->weight <<
"km) ";
                                e = e->next;
                        }
                        cout << endl;
                        temp = temp->next;
                        index++;
                } while (temp != head);
        }
};
int main()
{
        Metro_Station metro;
        cout << "----------- Metro Bus Route Planner -----------" << endl;
        metro.insert_station("Home", 1);
        metro.insert_station("School", 2);
        metro.insert_station("University", 3);
        metro.insert_station("Hospital", 4);
        metro.insert_station("Market", 5);
        metro.insert_station("Airport", 6);
        metro.add_connections(1, 2, 3);
        metro.add_connections(2, 3, 5);
        metro.add_connections(1, 4, 4);
        metro.add_connections(4, 5, 2);
        metro.add_connections(3, 5, 6);
        metro.add_connections(5, 6, 10);
        metro.add_connections(3, 6, 15);
        int choice;
        do
        {
                cout << "\n========= MAIN MENU =========" << endl;
                cout << "1. Display all stations" << endl;
                cout << "2. Display all connections" << endl;
                cout << "3. Find shortest path" << endl;
                cout << "4. Add new station" << endl;
                cout << "5. Add new connection" << endl;
                cout << "6. Exit" << endl;
                cout << "Enter your choice: ";
                cin >> choice;
                switch (choice)
                {
                case 1:
                        metro.display_stations_list();
                        break;
                case 2:
                        metro.display_connections();
                        break;
                case 3:
```

```
                        {
                                int src, dest;
                                cout << "Enter source station ID: ";
                                cin >> src;
                                cout << "Enter destination station ID: ";
                                cin >> dest;
                                metro.find_shortest_path(src, dest);
                                break;
                        }
                case 4:
                        {
                                string name;
                                int id;
                                cin.ignore();
                                cout << "Enter station name: ";
                                getline(cin, name);
                                cout << "Enter station ID: ";
                                cin >> id;
                                metro.insert_station(name, id);
                                cout << "Station added successfully!" << endl;
                                break;
                        }
                case 5:
                        {
                                int from, to, dist;
                                cout << "Enter from station ID: ";
                                cin >> from;
                                cout << "Enter to station ID: ";
                                cin >> to;
                                cout << "Enter distance (km): ";
                                cin >> dist;
                                metro.add_connections(from, to, dist);
                                cout << "Connection added successfully!" << endl;
                                break;
                        }
                case 6:
                        cout << "Exiting program. Goodbye!" << endl;
                        break;
                default:
                        cout << "Invalid choice! Try again." << endl;
                }
        } while (choice != 6);
        return 0;
}
```

Output:

1. All stations of metro:

```
---------- Metro Bus Route Planner ----------

========= MAIN MENU =========
1. Display all stations
2. Display all connections
3. Find shortest path
4. Add new station
5. Add new connection
6. Exit
Enter your choice: 1
Stations List:
ID: 1, Name: Home
ID: 2, Name: School
ID: 3, Name: University
ID: 4, Name: Hospital
ID: 5, Name: Market
ID: 6, Name: Airport

========= MAIN MENU =========
1. Display all stations
2. Display all connections
3. Find shortest path
4. Add new station
5. Add new connection
6. Exit
Enter your choice:
```

2. All connections of Stations

```
Enter your choice: 2
Connections:
Home -> Hospital(4km) School(3km)
School -> University(5km) Home(3km)
University -> Airport(15km) Market(6km) School(5km)
Hospital -> Market(2km) Home(4km)
Market -> Airport(10km) University(6km) Hospital(2km)
Airport -> University(15km) Market(10km)
```

3. Find Shortest Path

```
========= MAIN MENU =========
1. Display all stations
2. Display all connections
3. Find shortest path
4. Add new station
5. Add new connection
6. Exit
Enter your choice: 3
Enter source station ID: 2
Enter destination station ID: 6
Shortest distance: 19 km
Path: Airport -> Market -> Hospital -> Home -> School
```

4. Add new station:

```
========= MAIN MENU =========
1. Display all stations
2. Display all connections
3. Find shortest path
4. Add new station
5. Add new connection
6. Exit
Enter your choice: 4
Enter station name: Bank
Enter station ID: 7
Station added successfully!
```

5. Add new connections:

```
========= MAIN MENU =========
1. Display all stations
2. Display all connections
3. Find shortest path
4. Add new station
5. Add new connection
6. Exit
Enter your choice: 5
Enter from station ID: 6
Enter to station ID: 7
Enter distance (km): 34
Connection added successfully!

========= MAIN MENU =========
1. Display all stations
2. Display all connections
3. Find shortest path
4. Add new station
5. Add new connection
6. Exit
Enter your choice:
```

After a dding new connections and new station:

```
========= MAIN MENU =========
1. Display all stations
2. Display all connections
3. Find shortest path
4. Add new station
5. Add new connection
6. Exit
Enter your choice: 2
Connections:
Home -> Hospital(4km) School(3km)
School -> University(5km) Home(3km)
University -> Airport(15km) Market(6km) School(5km)
Hospital -> Market(2km) Home(4km)
Market -> Airport(10km) University(6km) Hospital(2km)
Airport -> Bank(34km) University(15km) Market(10km)
Bank -> Airport(34km)
```

Now we have 7 stations after adding one more :

```
Enter your choice: 1
Stations List:
ID: 1, Name: Home
ID: 2, Name: School
ID: 3, Name: University
ID: 4, Name: Hospital
ID: 5, Name: Market
ID: 6, Name: Airport
ID: 7, Name: Bank
```