Smart Traffic Management System

1.2 Team Details

Si. No.	Roll No.	Name
1.	62	SONALI JADHAV
2.	67	STUTI HUNACHAGI
3.	42	SARVESH NIRMALKAR
4.	03	AKHILESH JOSHI

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <limits.h>


#define MAX_NODES 51

#define MAX_ROADS 45

#define MAX_LOGIN_ATTEMPTS 3

#define MAX_PARKING_SLOTS 10


// Adjacency matrix to represent the connectivity between cities

int adj[MAX_NODES][MAX_NODES];

int acount = 0;// Counter for the number of cities

int ccount = 0;// Counter for the number of roads

int visited[MAX_NODES];

int i, j, k, l, v, count;

int road_info[MAX_ROADS][4];


// Structure to represent a city
```

```
typedef struct places
{
   int id;
   char name[50];
} CT;

CT S[1000];
void displayCities()
{
   printf("\nCities:\n");
   for (int i = 0; i < ccount; i++)
   {
      printf("%s\n", S[i].name);
   }
}

// Structure to represent a parking slot
typedef struct ParkingSlot
{
   int slotNumber;
   char reservedBy[30];
   int hoursReserved;
   struct ParkingSlot *next;
} ParkingSlot;

ParkingSlot *parkingSlots[MAX_PARKING_SLOTS];
//Initialize parking slot
void initializeParkingSlots()
{
   for (int i = 0; i < MAX_PARKING_SLOTS; i++)
   {
```

```
        parkingSlots[i] = NULL;
    }
}

void displayParkingSlots()
{
    printf("\nParking Slots:\n");

    for (int i = 0; i < MAX_PARKING_SLOTS; i++)
    {
        printf("Slot %d: ", i + 1);

        if (parkingSlots[i] == NULL)
        {
            printf("Available\n");
        }
        else
        {
            printf("Reserved by: %s, Hours Reserved: %d\n", parkingSlots[i]->reservedBy, parkingSlots[i]->hoursReserved);
        }
    }

    printf("\n");
}

void reserveParkingSlot(char username[])
{
    int slotNumber;
    printf("Enter the slot number to reserve: ");
    scanf("%d", &slotNumber);
```

```
    if (slotNumber < 1 || slotNumber > MAX_PARKING_SLOTS)
    {
        printf("Invalid slot number.\n");
        return;
    }

    if (parkingSlots[slotNumber - 1] != NULL)
    {
        printf("Slot %d is already reserved.\n", slotNumber);
        return;
    }

    int hoursReserved;
    printf("Enter the number of hours to reserve: ");
    scanf("%d", &hoursReserved);

    ParkingSlot *newReservation = (ParkingSlot *)malloc(sizeof(ParkingSlot));
    newReservation->slotNumber = slotNumber;
    strcpy(newReservation->reservedBy, username);
    newReservation->hoursReserved = hoursReserved;
    newReservation->next = NULL;

    parkingSlots[slotNumber - 1] = newReservation;
    printf("Parking slot %d reserved by %s for %d hours.\n", slotNumber, username, hoursReserved);
}
void searchReservedUser(char username[])
{
    for (int i = 0; i < MAX_PARKING_SLOTS; i++)
    {
        if (parkingSlots[i] != NULL)
```

```
    {
        char *reservedUsername = parkingSlots[i]->reservedBy;
        int lenReserved = strlen(reservedUsername);
        int lenSearch = strlen(username);

        for (int j = 0; j <= lenReserved - lenSearch; j++)
        {
            int k;
            for (k = 0; k < lenSearch; k++)
            {
                if (reservedUsername[j + k] != username[k])
                    break;
            }

            if (k == lenSearch)
            {
                printf("User %s has reserved Slot %d for %d hours.\n", username, parkingSlots[i]->slotNumber, parkingSlots[i]->hoursReserved);
                break;  // Assuming you want to stop after finding the first match
            }

        if(k != lenSearch)
            {
                printf("User not reserved\n");
            }
        }

    }

}
```

```
    printf("\n");
}

typedef struct User {
    char username[30];
    char password[30];
} User;

User users[] = {{"admin", "admin123"}, {"user1", "password1"}, {"user2", "password2"}};
int numUsers = sizeof(users) / sizeof(users[0]);




int authenticateUser(char username[], char password[]) {
    for (int i = 0; i < numUsers; i++) {
        if (strcmp(username, users[i].username) == 0 && strcmp(password, users[i].password) == 0) {
            return 1; // Authentication successful
        }
    }
    return 0; // Authentication failed
}

void login() {
    char username[30], password[30];
    int loginAttempts = 0;

    while (loginAttempts < MAX_LOGIN_ATTEMPTS) {
        printf("\nEnter username: ");
        scanf("%s", username);
        printf("Enter password: ");
        scanf("%s", password);
```

```
        if (authenticateUser(username, password)) {
            printf("\nLogin successful!\n");
            return;
        } else {
            printf("\nIncorrect username or password. Please try again.\n");
            loginAttempts++;
        }
    }

    printf("\nToo many incorrect login attempts. Exiting program.\n");
    exit(1);
}

int calorder()
{
    FILE *fp = fopen("adjacency_matrix.txt", "r");

    char ch;
    int row = 0;

    if (fp == NULL)
    {
        printf("\nCannot open the file");
        return 0;
    }

    while ((ch = fgetc(fp)) != '\n')
    {
        if (ch == ',')
        {
```

```
            row++;
        }
    }

    fclose(fp);

    return row + 1;
}

void load_adjacency()
{
    FILE *fp;
    fp = fopen("adjacency_matrix.txt", "r");
    int n = calorder();
    int temp;
    char s;

    acount = n;

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            fscanf(fp, "%d%c", &temp, &s);
            adj[i][j] = temp;
        }
    }

    fclose(fp);
}
```

```
void load_cities()
{
   FILE *city;
   city = fopen("cities_list.txt", "r");
   int idd;
   char nam[30];

   if (city == NULL)
   {
      printf("\nCannot open the file");
      return;
   }

   while (fscanf(city, "%d%s", &idd, nam) != EOF)
   {
      S[ccount].id = idd;
      strcpy(S[ccount].name, nam);
      ccount++;
   }
   fclose(city);
}

int num = 1;

int minDistance(int dist[], int sptSet[])
{
   int min = INT_MAX, min_index;
   int v;
   for (v = 0; v < MAX_NODES; v++)
   {
      if (sptSet[v] == 0 && dist[v] <= min)
```

```
        {
            min = dist[v];
            min_index = v;
        }
    }

    return min_index;
}
//printing path
void printPath(int parent[], int j)
{
    if (parent[j] == -1)
    {
        return;
    }

    printPath(parent, parent[j]);
    num++;
    printf("-->(%d)-->%s", adj[parent[j]][j], S[j].name);
}

//printing solution
void printSolution(int dist[], int parent[], int src, int dest)
{
    printf("\n\nThe shortest Route is  \n\n\t%s", S[src].name);
    printPath(parent, dest);

    printf("\n\n Average Traffic-index : %d \n", dist[dest] / num);

    printf("\n");
}
```

```
//Using Dijkstra algorithm for finding shortest path
//based on traffic index

void dijkstra(int src, int dest)
{
   int dist[MAX_NODES];
   int sptSet[MAX_NODES];
   int parent[MAX_NODES];
   int i;

   for (i = 0; i < MAX_NODES; i++)
   {
      dist[i] = INT_MAX;
      sptSet[i] = 0;
      parent[i] = -1;
   }

   dist[src] = 0;
   int count;
   for (count = 0; count < MAX_NODES - 1; count++)
   {
      int u = minDistance(dist, sptSet);

      sptSet[u] = 1;
      int v;
      for (v = 0; v < MAX_NODES; v++)
      {
         if (!sptSet[v] && adj[u][v] && dist[u] != INT_MAX && dist[u] + adj[u][v] < dist[v])
         {
            dist[v] = dist[u] + adj[u][v];
```

```
            parent[v] = u;
          }
        }
    }

    printSolution(dist, parent, src, dest);
}

// Function to calculate traffic index (Replace this with your actual logic)
int calculateTrafficIndex(int cityId)
{
    // Placeholder logic, replace this with your actual calculation
    return cityId;
}




void heapSortByTrafficIndex(CT arr[], int n)
{
    int i;
    for (i = n / 2 - 1; i >= 0; i--)
    {
        heapifyByTrafficIndex(arr, n, i);
    }

    for (i = n - 1; i > 0; i--)
    {
        CT temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
```

```
        heapifyByTrafficIndex(arr, i, 0);
    }
}

void heapifyByTrafficIndex(CT arr[], int n, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    int indexLeft = arr[left].id;
    int indexRight = arr[right].id;

    // Corrected comparison logic for ascending order
    if (left < n && calculateTrafficIndex(indexLeft) > calculateTrafficIndex(arr[largest].id))
    {
        largest = left;
    }

    if (right < n && calculateTrafficIndex(indexRight) > calculateTrafficIndex(arr[largest].id))
    {
        largest = right;
    }

    if (largest != i)
    {
        CT temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        heapifyByTrafficIndex(arr, n, largest);
```

```
    }
}



// Function to print cities along with their traffic indices
void printCitiesWithTrafficIndex(CT arr[], int n)
{
    for (i = 0; i < n; i++)
    {
        printf("%s - Traffic Index: %d\n", arr[i].name, calculateTrafficIndex(arr[i].id));
    }
}


 int alphabet()
 {
     CT sortedCities[ccount];
           for (i = 0; i < ccount; i++)
            {
            sortedCities[i] = S[i];
            }


            printf("\nSorting Cities by Alphabetical Order:\n");
 }
// Function to search for city with minimum traffic index
void searchMinTrafficIndex()
{
    CT sortedCities[ccount];
    for (i = 0; i < ccount; i++)
    {
        sortedCities[i] = S[i];
    }
```

```
    printf("\n1. Sort by Alphabetical Order of Cities");
    printf("\n2. Sort by Traffic Index");
    printf("\nEnter your choice: ");
    int sortChoice;
    scanf("%d", &sortChoice);

    switch (sortChoice)
    {
    case 1:
        // Sorting by alphabetical order


            alphabet();
            bubbleSortByAlphabeticalOrder(sortedCities, ccount);

        break;

    case 2:
        // Sorting by traffic index
        heapSortByTrafficIndex(sortedCities, ccount);
        break;

    default:

        printf("\nInvalid choice for sorting.");
        return;
    }

    printCitiesWithTrafficIndex(sortedCities, ccount);
}
```

```
void bubbleSortByAlphabeticalOrder(CT arr[], int n)
{
   int i, j;
   for (i = 0; i < n - 1; i++)
   {
      for (j = 0; j < n - i - 1; j++)
      {
         if (strcmp(arr[j].name, arr[j + 1].name) > 0)
         {
            // Swap arr[j] and arr[j + 1]
            CT temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}



int checklist(char key[])
{
   int flag = 0;

   for (i = 0; i < ccount; i++)
   {
      if (strcmp(key, S[i].name) == 0)
         return S[i].id;
   }

   return 1000;
}
```

```
void initializeRoadInfo()
{
   // Placeholder data for road information
   // Format: {source_city_id, destination_city_id, two_lane_road, one_way_road}
   int roads[MAX_ROADS][4] = {

{0,1,0,1,0,1},
{0,2,1,0,0,0},
{0,3,0,1,1,0},
{0,4,1,0,0,1},
{0,5,0,1,0,1},
{0,6,1,0,0,0},
{0,7,0,1,0,1},
{0,9,0,1,0,0},

{1,2,0,1,1,1},
{1,8,0,1,1,0},
{1,4,1,0,0,1},
{1,5,0,1,0,1},
{1,6,0,1,1,0},

{2,7,0,1,1,1},
{2,8,0,1,1,0},
{2,1,0,1,0,1},
{2,8,0,1,1,0},
{2,4,1,0,0,1},
{2,5,0,1,1,1},
{2,6,0,1,0,1},

{3,1,0,1,0,1},
```

{3,2,1,0,0,0},

{3,4,1,0,0,1},

{3,5,0,1,0,1},

{3,9,0,1,0,0},

{4,1,1,0,0,1},

{4,2,0,1,1,1},

{4,8,0,1,1,0},

{4,5,0,1,0,1},

{4,6,0,1,1,0},

{5,7,0,1,1,1},

{5,1,0,1,0,1},

{5,2,0,1,0,0},

{5,8,0,1,1,0},

{5,4,1,0,0,1},

{5,6,0,1,0,1},

{6,1,0,1,0,1},

{6,2,1,0,0,0},

{6,4,1,0,0,1},

{6,5,0,1,0,1},

{6,9,0,1,0,0},

{7,1,0,1,1,1},

{7,2,1,0,0,0},

{7,3,0,1,0,1},

{7,5,0,1,0,1},

{7,9,0,1,0,1},

{7,4,0,1,0,1},

{7,6,0,1,0,1},

```
{7,8,0,1,0,1},

{8,7,0,1,1,1},
{8,3,0,1,1,0},
{8,1,0,1,0,1},
{8,2,0,1,0,0},
{8,5,0,1,1,0},
{8,4,1,0,0,1},
{8,6,0,1,0,1},

{9,1,0,1,0,1},
{9,2,1,0,0,0},
{9,4,1,0,0,1},
{9,5,0,1,0,1},
{9,6,0,1,0,0},
{9,7,0,1,1,1},
{9,8,1,0,1,0}

};

   // Copy the placeholder data to the road_info array
   for (int i = 0; i < MAX_ROADS; i++)
   {
      for (int j = 0; j < 5; j++)
      {
         road_info[i][j] = roads[i][j];
      }
   }
}

// Function to get road information between two cities
```

```
void roadinfo(int src, int dest)
{
   int found = 0;

   for (i = 0; i < MAX_ROADS; i++)
   {
      if (road_info[i][0] == src && road_info[i][1] == dest)
      {
         printf("\nRoad Information from City%d to City%d:\n", src, dest);
         printf("- Two Lane Road: %s\n", road_info[i][2] ? "Yes" : "No");
         printf("- One Way Road: %s\n", road_info[i][3] ? "Yes" : "No");
         printf("- Under Construction: %s\n", road_info[i][4] ? "Yes" : "No");
         printf("- Accident Prone Area: %s\n", road_info[i][5] ? "Yes (Drive Slowly)" : "No");
         found = 1;
         break;
      }
   }

   if (!found)
   {
      printf("\nRoad information not available between City%d and City%d.\n", src, dest);
   }
}



// Function to perform BFS from a given source city within a given limit of roads
void bfsReachableCities(int src, int limit)
{
   int visited[MAX_NODES] = {0};
   int queue[MAX_NODES];
```

```
    int front = 0, rear = 0;

    // Start from the source node
    visited[src] = 1;
    queue[rear++] = src;

    printf("\nCities reachable from %s within %d roads:\n", S[src].name, limit);

    while (front < rear)
    {
        int u = queue[front++];
        printf("%s, ", S[u].name);

        for (int v = 0; v < MAX_NODES; v++)
        {
            if (adj[u][v] && !visited[v])
            {
                visited[v] = 1;
                queue[rear++] = v;

                if (limit > 1)
                    limit--;

                if (limit == 0)
                    break;
            }
        }
    }

    printf("\n");
}
```

```
void updateRoadStatus()
{
   int src, dest;

   // Get source and destination cities
   while (1)
   {
      printf("\nEnter source city: ");
      char srcCity[30];
      scanf("%s", srcCity);
      src = checklist(srcCity);
      if (src != 1000)
         break;
      printf("\nThe city you have entered is not valid, please re-enter.\n");
   }

   while (1)
   {
      printf("Enter destination city: ");
      char destCity[30];
      scanf("%s", destCity);
      dest = checklist(destCity);
      if (dest != 1000)
         break;
      printf("\nThe city you have entered is not valid, please re-enter.\n");
   }

   // Check if road information exists
   int found = 0;
   for (int i = 0; i < MAX_ROADS; i++)
   {
```

```
        if (road_info[i][0] == src && road_info[i][1] == dest)
        {
            found = 1;

            printf("\nCurrent Road Status from City%d to City%d:\n", src, dest);
            printf("- Two Lane Road: %s\n", road_info[i][2] ? "Yes" : "No");
            printf("- One Way Road: %s\n", road_info[i][3] ? "Yes" : "No");
            printf("- Under Construction: %s\n", road_info[i][4] ? "Yes" : "No");

            // Update road status
            printf("\nEnter updated road status:\n");
            printf("Two Lane Road (1 for Yes, 0 for No): ");
            scanf("%d", &road_info[i][2]);
            printf("One Way Road (1 for Yes, 0 for No): ");
            scanf("%d", &road_info[i][3]);
            printf("Under Construction (1 for Yes, 0 for No): ");
            scanf("%d", &road_info[i][4]);

            printf("\nRoad status updated successfully!\n");
            break;
        }
    }

    if (!found)
    {
        printf("\nRoad information not available between City%d and City%d.\n", src, dest);
    }
}

void menu()
{
```

```
    printf("\n\n\n");
    printf("          ********              \n");
    printf("          * TRAFFIC MANAGEMENT *              \n");
    printf("          *       SYSTEM      *              \n");
    printf("          ********              \n");
    printf("\n\n\n");
    printf("\tWHAT WOULD YOU LIKE TO EXPLORE ? \n");
    printf("\n=======================================================\n");
    printf("1. Display cities\n");
    printf("2. City Information\n");
    printf("3. Display path from source to destination\n");
    printf("4. Reserve Parking Slot\n");
    printf("5. Display Parking Slots\n");
    printf("6. Search Reserved User\n");
    printf("7. Roadinfo\n");
    printf("8. Reachable Cities\n");
    printf("9. Update Road Status\n");
    printf("10. Clear screen\n");
    printf("11. Exit\n");

    printf("=======================================================\n");
}


int main()
{
    int choice;
    login();
    load_adjacency();
    load_cities();
```

```
 char src[30], desti[30];
 int c1, c2;

printf("\n\n\n");
 printf("          ********              \n");
 printf("          * TRAFFIC MANAGEMENT *              \n");
 printf("          *       SYSTEM      *              \n");
 printf("          ********              \n");
 printf("\n\n\n");
 printf("\tWHAT WOULD YOU LIKE TO EXPLORE ? \n");
 printf("\n=======================================================\n");
 printf("1. Display cities\n");
 printf("2. City Information\n");
 printf("3. Display path from source to destination\n");
 printf("4. Reserve Parking Slot\n");
 printf("5. Display Parking Slots\n");
 printf("6. Search Reserved User\n");
 printf("7. Roadinfo\n");
 printf("8. Reachable Cities\n");
  printf("9. Update Road Status\n");
 printf("10. Clear screen\n");
 printf("11. Exit\n");

 printf("=======================================================\n");
char username[30];
initializeRoadInfo();
 while (1)
 {
    printf("\n|| Enter your choice :");

    scanf("%d", &choice);
```

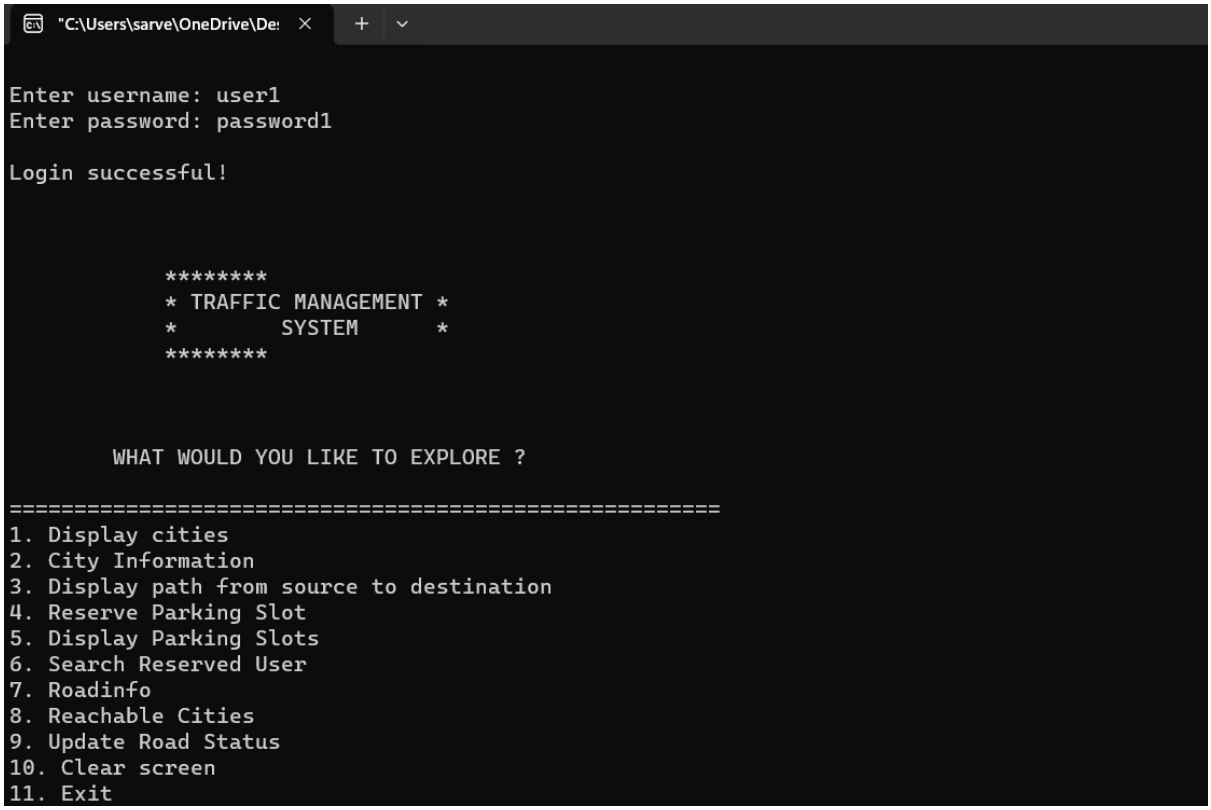
```
printf("\n-----------------------------------------------------");

switch (choice)
{

case 1 : displayCities();
            break;

case 2:
   // Search for city with minimum traffic index option
   searchMinTrafficIndex();
   break;




case 3:
   while (1)
   {
      printf("\n\n|| Enter Start Location : ");
      scanf("%s", src);
      c1 = checklist(src);
      if (c1 != 1000)
         break;
      printf("\nThe city you have entered is not valid, please re-enter.\n");
      printf("\n        ||           ");
      printf("\n        ||           ");
      printf("\n        ||           ");
      printf("\n       ----          ");
      printf("\n       \\  /          ");
      printf("\n        \\/           ");
```

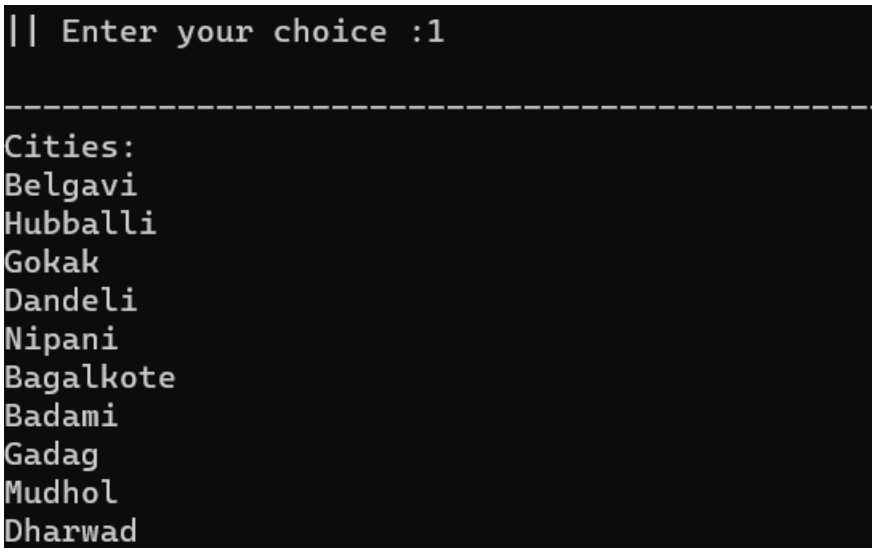
```
 }

 while (1)
 {
    printf("\n|| Enter The Destination  : ");
    scanf("%s", desti);
    printf("\n-----------------------------------------------------");
    c2 = checklist(desti);
    if (c2 != 1000)
       break;
    printf("\nThe city you have entered is not valid , please re-enter.\n");
    printf("\n        ||           ");
    printf("\n        ||           ");
    printf("\n        ||           ");
    printf("\n       ----          ");
    printf("\n       \\  /          ");
    printf("\n        \\/           ");

 }

     dijkstra(c1, c2);
       break;

case 4:
      // Reserve Parking Slot
       printf("Enter your username: ");
      scanf("%s", username);
      reserveParkingSlot(username);
      break;

case 5:
```

```
        // Display Parking Slots
        displayParkingSlots();
        break;

case 6:
        // Search Reserved User
        printf("Enter the username to search: ");
        scanf("%s", username);
        searchReservedUser(username);
        break;

case 7:

       while (1)
       {
        printf("\n\n|| Enter Start Location : ");
        scanf("%s", src);
        c1 = checklist(src);
        if (c1 != 1000)
        break;
        printf("\nThe city you have entered is not valid, please re-enter.\n");
        printf("\n        ||           ");
        printf("\n        ||           ");
        printf("\n        ||           ");
        printf("\n       ----          ");
        printf("\n       \\  /          ");
        printf("\n        \\/           ");
        }

       while (1)
       {
```

```
      printf("\n|| Enter The Destination  : ");
      scanf("%s", desti);
      printf("\n-------------------------------------------------------");
      c2 = checklist(desti);
      if (c2 != 1000)
       break;
       printf("\nThe city you have entered is not valid , please re-enter.\n");
       printf("\n        ||           ");
       printf("\n        ||           ");
       printf("\n        ||           ");
       printf("\n       ----          ");
       printf("\n       \\  /          ");
       printf("\n        \\/           ");
      }

      // Call dijkstra to find the shortest path
       dijkstra(c1, c2);

      // Call roadinfo to get road information
       roadinfo(c1, c2);
       break;




case 8:
      // BFS to find cities reachable within a specific number of roads
      while (1)
      {
       printf("\n\n|| Enter Start Location : ");
       scanf("%s", src);
       c1 = checklist(src);
```

```
                if (c1 != 1000)
                 break;

                 printf("\nThe city you have entered is not valid, please re-enter.\n");
                }

                printf("\nEnter the limit of roads: ");
                int roadLimit;
                scanf("%d", &roadLimit);

                bfsReachableCities(c1, roadLimit);
                break;

        case 9:
           // Update Road Status
           updateRoadStatus();
           break;

        case 10: system("cls");
                menu();
                break;
        case 11: exit(0);
                break;

        default:
           printf("\nInvalid choice, please enter a valid choice");
      }
   }

   return 0;
}
```


"C:\Users\sarve\OneDrive\De:
Enter username: user1
Enter password: password1
Login successful!
****
* TRAFFIC MANAGEMENT *
* SYSTEM *
****
WHAT WOULD YOU LIKE TO EXPLORE ?
1. Display cities
2. City Information
3. Display path from source to destination
4. Reserve Parking Slot
5. Display Parking Slots
6. Search Reserved User
7. Roadinfo
8. Reachable Cities
9. Update Road Status
10. Clear screen
11. Exit


|| Enter your choice :1
Cities:
Belgavi
Hubballi
Gokak
Dandeli
Nipani
Bagalkote
Badami
Gadag
Mudhol
Dharwad

```
|| Enter your choice :3

-----------------------------------------------------

|| Enter Start Location : Belgavi

|| Enter The Destination  : Gokak

-----------------------------------------------------

The shortest Route is

        Belgavi-->(3)-->Gokak

 Average Traffic-index : 1
```

```
|| Enter your choice :4

-----------------------------------------------------Enter your username: Raj
Enter the slot number to reserve: 3
Enter the number of hours to reserve: 2
Parking slot 3 reserved by Raj for 2 hours.

|| Enter your choice :5

-----------------------------------------------------
Parking Slots:
Slot 1: Available
Slot 2: Available
Slot 3: Reserved by: Raj, Hours Reserved: 2
Slot 4: Available
Slot 5: Available
Slot 6: Available
Slot 7: Available
Slot 8: Available
Slot 9: Available
Slot 10: Available
```

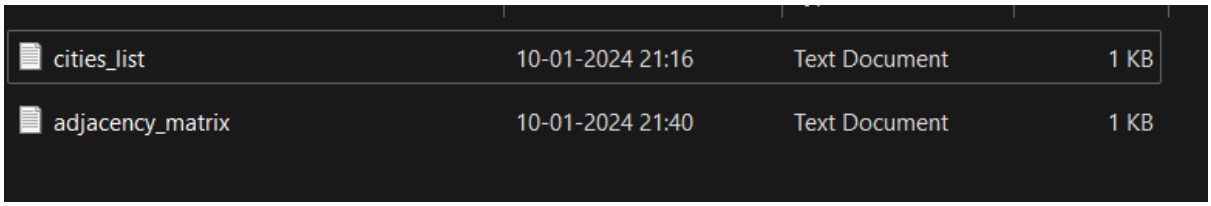
```
|| Enter your choice :6

-----------------------------------------------------Enter the username to search: Raj
User Raj has reserved Slot 3 for 2 hours.
```

```
|| Enter your choice :7

-----------------------------------------------------

|| Enter Start Location : Belgavi

|| Enter The Destination  : Gokak

-----------------------------------------------------

The shortest Route is

        Belgavi-->(3)-->Gokak

 Average Traffic-index : 1


Road Information from City0 to City2:
- Two Lane Road: Yes
- One Way Road: No
- Under Construction: No
- Accident Prone Area: Yes (Drive Slowly)

|| Enter your choice :
```

```
|| Enter your choice :9

-----------------------------------------------------
Enter source city: Belgavi
Enter destination city: Gokak

Current Road Status from City0 to City2:
- Two Lane Road: Yes
- One Way Road: No
- Under Construction: No

Enter updated road status:
Two Lane Road (1 for Yes, 0 for No): 0
One Way Road (1 for Yes, 0 for No): 1
Under Construction (1 for Yes, 0 for No): 0

Road status updated successfully!
```

```
        WHAT WOULD YOU LIKE TO EXPLORE ?

=========================================================
1. Display cities
2. City Information
3. Display path from source to destination
4. Reserve Parking Slot
5. Display Parking Slots
6. Search Reserved User
7. Roadinfo
8. Reachable Cities
9. Update Road Status
10. Clear screen
11. Exit
=========================================================

|| Enter your choice :
```

FILES USED:



Cities_list:

```
0 Belgavi
1 Hubballi
2 Gokak
3 Dandeli
4 Nipani
5 Bagalkote
6 Badami
7 Gadag
8 Mudhol
9 Dharwad
```

Adjacency Matrix:

```
0, 6, 3, 5, 0, 0, 0, 0, 0, 0,
6, 0, 0, 9, 0, 0, 0, 6, 3, 4,
3, 0, 0, 0, 9, 0, 9, 0, 4, 0,
5, 9, 0, 0, 0, 0, 0, 0, 0, 6,
0, 0, 9, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 2, 0,
0, 0, 9, 0, 0, 1, 0, 5, 0, 0,
0, 6, 0, 0, 0, 0, 5, 0, 0, 6,
0, 3, 4, 0, 0, 2, 0, 0, 0, 0,
0, 4, 0, 6, 0, 0, 0, 6, 0, 0,
```