LAB PROJECT

Name: _____

Roll No: _____

Question 1:

Design and implement a social network-based friend recommendation system using **Breadth-First Search (BFS)**.

Allow the user to input the number of users and friendships. Then suggest mutual friends who are two levels away from the selected user (i.e., friends of friends).

Objectives:

- Represent the graph using adjacency list.
- Implement BFS traversal.
- Recommend mutual friends (level 2 nodes).

C Program:

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 100


struct Node {

   int data;

   struct Node* next;

};


struct Node* adjList[MAX];

int visited[MAX];
```

```
int level[MAX];

void addEdge(int src, int dest) {
    struct Node* newNode = malloc(sizeof(struct Node));
    newNode->data = dest;
    newNode->next = adjList[src];
    adjList[src] = newNode;

    newNode = malloc(sizeof(struct Node));
    newNode->data = src;
    newNode->next = adjList[dest];
    adjList[dest] = newNode;
}

void resetVisited(int n) {
    for (int i = 0; i <= n; i++) {
        visited[i] = 0;
        level[i] = 0;
    }
}

void recommendFriends(int start, int n) {
    int queue[MAX], front = 0, rear = -1;
    resetVisited(n);
    visited[start] = 1;
    queue[++rear] = start;
```

```
    while (front <= rear) {
        int curr = queue[front++];
        struct Node* temp = adjList[curr];
        while (temp != NULL) {
            if (!visited[temp->data]) {
                visited[temp->data] = 1;
                level[temp->data] = level[curr] + 1;
                queue[++rear] = temp->data;
            }
            temp = temp->next;
        }
    }

    printf("Suggested friends for user %d:\n", start);
    for (int i = 0; i <= n; i++) {
        if (level[i] == 2)
            printf("User %d (mutual friend)\n", i);
    }
}

int main() {
    int n, e;
    printf("Enter number of users: ");
    scanf("%d", &n);
```

```
    for (int i = 0; i <= n; i++)
        adjList[i] = NULL;

    printf("Enter number of friendships: ");
    scanf("%d", &e);
    printf("Enter each friendship (user1 user2):\n");
    for (int i = 0; i < e; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        addEdge(u, v);
    }

    int target;
    printf("Enter user to suggest friends for: ");
    scanf("%d", &target);

    recommendFriends(target, n);
    return 0;
}
```

Input:

Enter number of users: 5

Enter number of friendships: 5

0 1

0 2

1 3

2 4

4 5

Enter user to suggest friends for: 0

Output:

Suggested friends for user 0:

User 3 (mutual friend)

User 4 (mutual friend)

Question 2:

Design and implement a simple hash table in C to store student records consisting of roll number (key) and name (value).Use linear probing to resolve any collisions that occur during insertion.

Objective :

- Implement hashing to store key-value pairs.
- Handle collisions using linear probing.
- Display the final state of the hash table after all insertions.

```
#include <stdio.h>

#include <string.h>



#define SIZE 10



int keys[SIZE];

char names[SIZE][50];



int hash(int key) {

   return key % SIZE;

}
```

```
void insert(int key, char name[]) {
    int index = hash(key);
    while (keys[index] != 0) {  // Collision handling
        index = (index + 1) % SIZE;
    }
    keys[index] = key;
    strcpy(names[index], name);
}

void display() {
    printf("\nStored Student Records:\n");
    for (int i = 0; i < SIZE; i++) {
        if (keys[i] != 0)
            printf("Index %d: %d -> %s\n", i, keys[i], names[i]);
    }
}

int main() {
    int n;
    printf("Enter number of students: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        int roll;
        char name[50];
        printf("Enter roll number and name: ");
```

```
        scanf("%d %s", &roll, name);
        insert(roll, name);
    }

    display();
    return 0;
}
```

Input :

Enter number of students: 4

Enter roll number and name: 101 Ravi

Enter roll number and name: 111 Sita

Enter roll number and name: 121 Gaurav

Enter roll number and name: 131 Latha

Output :

Stored Student Records:

Index 0: 101 -> Ravi

Index 1: 111 -> Sita

Index 2: 121 -> Gaurav

Index 3: 131 -> Latha

Question 3:

Design and implement a password storage system in C where each user's ID is hashed and their password is stored securely.

Use **double hashing** to resolve key collisions in the hash table.

Objective :

- Implement a hash table with double hashing.
- Store user ID and password pairs.
- Prevent overwriting on collision using a second hash function.

C Program :

```
#include <stdio.h>

#include <string.h>


#define SIZE 10


int keys[SIZE];

char passwords[SIZE][50];


int hash1(int key) {

   return key % SIZE;

}


int hash2(int key) {

   return 7 - (key % 7); // Second hash must not be 0

}
```

```
void insert(int key, char pass[]) {
    int index = hash1(key);
    int step = hash2(key);
    int i = 0;

    while (keys[index] != 0) {
        i++;
        index = (index + i * step) % SIZE;
    }

    keys[index] = key;
    strcpy(passwords[index], pass);
}

void display() {
    printf("\nStored User Passwords:\n");
    for (int i = 0; i < SIZE; i++) {
        if (keys[i] != 0)
            printf("Index %d: %d -> %s\n", i, keys[i], passwords[i]);
    }
}

int main() {
    int n;
    printf("Enter number of users: ");
    scanf("%d", &n);
```

```
    for (int i = 0; i < n; i++) {
        int id;
        char pass[50];
        printf("Enter user ID and password: ");
        scanf("%d %s", &id, pass);
        insert(id, pass);
    }

    display();
    return 0;
}
```

Input :

Enter number of users: 4

Enter user ID and password: 101 pass123

Enter user ID and password: 111 hello123

Enter user ID and password: 121 welcome

Enter user ID and password: 131 admin@321

Output :

Stored User Passwords:

Index 1: 101 -> pass123

Index 2: 111 -> hello123

Index 3: 121 -> welcome

Index 4: 131 -> admin@321

Question 4:

Develop a C program to simulate a **maze solver** using the **Depth-First Search (DFS)** algorithm. The maze is represented as a 2D matrix where:

- 1 indicates a valid path
- 0 indicates a wall
- The user provides a **start point** and an **end point**, and the program should determine if a path exists.

Objective

:

- Represent a maze as a 2D array
- Traverse the maze using DFS recursively
- Indicate whether a path from the start to end point exists
- Use backtracking to explore all possible paths

C Program :

```
#include <stdio.h>


#define MAX 10


int maze[MAX][MAX], visited[MAX][MAX];

int rows, cols;

int dx[] = {-1, 1, 0, 0}; // Directions: Up, Down, Left, Right

int dy[] = {0, 0, -1, 1};

int found = 0;


void dfs(int x, int y, int ex, int ey) {

   if (x < 0 || y < 0 || x >= rows || y >= cols) return;

   if (maze[x][y] == 0 || visited[x][y]) return;

   if (x == ex && y == ey) {

      found = 1;
```

```
        return;
    }

    visited[x][y] = 1;

    for (int i = 0; i < 4; i++) {
        dfs(x + dx[i], y + dy[i], ex, ey);
        if (found) return;
    }

    visited[x][y] = 0; // Backtrack
}

int main() {
    int sx, sy, ex, ey;

    printf("Enter maze size (rows cols): ");
    scanf("%d %d", &rows, &cols);

    printf("Enter the maze (0=wall, 1=path):\n");
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            scanf("%d", &maze[i][j]);

    printf("Enter start (x y): ");
    scanf("%d %d", &sx, &sy);
```

```
    printf("Enter end (x y): ");
    scanf("%d %d", &ex, &ey);

    dfs(sx, sy, ex, ey);

    if (found)
        printf("Path found from (%d, %d) to (%d, %d)\n", sx, sy, ex, ey);
    else
        printf("No path found.\n");

    return 0;
}
```

Input :

```
Enter maze size (rows cols): 4 4
Enter the maze (0=wall, 1=path):
1 0 1 1
1 1 1 0
0 0 1 1
1 1 1 1
Enter start (x y): 0 0
Enter end (x y): 3 3
```

Output :

```
Path found from (0, 0) to (3, 3)
```

Question 5:

Design and implement a student record management system using **hashing with separate chaining** in C.

Each student has a roll number and name. Use a **linked list at each index** to handle collisions in the hash table.

Objective :

- Create a hash table using an array of linked lists
- Handle collisions using separate chaining
- Allow insertion and display of records

C Program :

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


#define SIZE 10


struct Node {

   int roll;

   char name[50];

   struct Node* next;

};


struct Node* hashTable[SIZE];


int hash(int key) {

   return key % SIZE;
```

```
}

void insert(int roll, char name[]) {
    int index = hash(roll);
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->roll = roll;
    strcpy(newNode->name, name);
    newNode->next = NULL;

    if (hashTable[index] == NULL) {
        hashTable[index] = newNode;
    } else {
        struct Node* temp = hashTable[index];
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
}

void display() {
    printf("\nStudent Records in Hash Table:\n");
    for (int i = 0; i < SIZE; i++) {
        struct Node* temp = hashTable[i];
        if (temp != NULL) {
            printf("Index %d: ", i);
            while (temp != NULL) {
```

```
            printf("%d -> %s", temp->roll, temp->name);
            temp = temp->next;
            if (temp != NULL)
                printf(" -> ");
        }
        printf("\n");
    }
  }
}

int main() {
  int n;
  printf("Enter number of students: ");
  scanf("%d", &n);

  for (int i = 0; i < SIZE; i++)
    hashTable[i] = NULL;

  for (int i = 0; i < n; i++) {
    int roll;
    char name[50];
    printf("Enter roll number and name: ");
    scanf("%d %s", &roll, name);
    insert(roll, name);
  }
```

```
    display();
    return 0;
}
```

Input :

```
Enter number of students: 4
Enter roll number and name: 105 Ravi
Enter roll number and name: 115 Sita
Enter roll number and name: 125 Gaurav
Enter roll number and name: 135 Latha
```

Output :

```
Student Records in Hash Table:
Index 5: 105 -> Ravi -> 115 -> Sita -> 125 -> Gaurav -> 135 -> Latha
```