Develop a Program in C for the following operations on Graph(G) of Cities
a. Create a Graph of N cities using Adjacency Matrix.
b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

```
#include <stdio.h>
#include <stdio.h>

const int MAX = 100;
const int SIZE = 10;
void fnBreadthFirstSearchReach(int vertex, int g[MAX][MAX], int v[MAX], int n);

typedef struct
{
        int iaItems[10];
        int iFront;
        int iRear;
}QUEUE;

void fnQInsert(QUEUE *stQueue, int elem);
int fnQDelete(QUEUE *stQueue);
int fnQFull(QUEUE *stQueue);
int fnQEmpty(QUEUE *stQueue);


int main(void)
{
        int graph[MAX][MAX];
        int visited[MAX];
        int numVert, startVert, i,j;

        printf("Enter the number of vertices : ");
        scanf("%d", &numVert);
        printf("Enter the adjacency matrix :\n");
        for (i=0; i<numVert; i++)
                visited[i] = 0;
        for (i=0; i<numVert; i++)
                for (j=0; j<numVert; j++)
                        scanf("%d", &graph[i][j]);
        printf("Enter the starting vertex : ");
        scanf("%d", &startVert);
        fnBreadthFirstSearchReach(startVert-1,graph,visited,numVert);
        printf("Vertices which can be reached from vertex %d are :-\n",startVert);
        for (i=0; i<numVert; i++)
                if (visited[i])
                        printf("%d ",i+1);
        printf("\n");
        return 0;
}

void fnBreadthFirstSearchReach(int vertex, int g[MAX][MAX], int v[MAX], int n)
{
        QUEUE stQueue;
        stQueue.iFront = 0;
        stQueue.iRear = -1;
        int frontVertex, i;
        v[vertex] = 1;
        fnQInsert(&stQueue, vertex);
        while (!fnQEmpty(&stQueue))
        {
                frontVertex = fnQDelete(&stQueue);
                for (i=0; i<n; i++)
                {
                        if (g[frontVertex][i] && !v[i])
                        {
                                v[i] = 1;
                                fnQInsert(&stQueue, i);
                        }
                }
        }
}



void fnQInsert(QUEUE *stQueue, int iItem)
{
        if(fnQFull(stQueue))
                printf("\nQueue Overflow\n");
        else
        {
                stQueue->iRear++;
                stQueue->iaItems[stQueue->iRear] = iItem;
        }
}


int fnQDelete(QUEUE *stQueue)
{
        int item;
        if(fnQEmpty(stQueue))
        printf("\nQueue Underflow\n");
        else
        if(stQueue->iRear == stQueue->iFront)
        {
                item = stQueue->iaItems[stQueue->iFront];
                stQueue->iRear=-1;
                stQueue->iFront=0;
        }
        else
        {
                item = stQueue->iaItems[stQueue->iFront++];
        }
        return item;
}


int fnQFull(QUEUE *stQueue)
{
        if(stQueue->iRear == SIZE-1)
                return 1;
        else
                return 0;
}


int fnQEmpty(QUEUE *stQueue)
{
        if(stQueue->iRear == stQueue->iFront-1)
                return 1;
        else
                return 0;
}
```

---

b) Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

```
#include <stdio.h>
const int MAX = 100;
void fnDepthFirstSearch(int currentVertex, int v[MAX], int g[MAX][MAX], int n);

int main(void)
{
        int i,j,k;
        int visited[MAX];
        int graph[MAX][MAX];
        int numVert, Vert;
        printf("Enter the number of vertices : ");
```

```
	scanf("%d", &numVert);
	for (i=0; i<numVert; i++)
		visited[i] = 0;
	printf("Enter the adjacency matrix :\n");
	for (i=0; i<numVert; i++)
		for (j=0; j<numVert; j++)
			scanf("%d", &graph[i][j]);
	printf("Enter the source vertex : ");
	scanf("%d", &Vert);
	fnDepthFirstSearch(Vert,visited,graph,numVert);
	for (k=0; k<numVert; k++)
	{
		if(visited[k])
		{
			printf("\nVertex %d is reachable\n", k+1);
		}
		else
		{
			printf("\nVertex %d is not reachable\n", k+1);
		}
	}

	return 0;
}

void fnDepthFirstSearch(int currentVertex, int v[MAX], int g[MAX][MAX], int n)
{
	int i;
	v[currentVertex] = 1;
	for (i=0; i<n; i++)
	{
		if (g[currentVertex][i] && !v[i])
			fnDepthFirstSearch(i,v,g,n);
	}
}
```

12. Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Develop a Program in C that uses Hash function H: K →L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NUM_EMPLOYEES 100  // Maximum number of employees
#define MAX_HASH_TABLE_SIZE 50  // Maximum size of the hash table

// Define the structure for an employee record
typedef struct
{
   int iKey;  // 4-digit key
   char cName[50];
}EMPLOYEE;

// Define the hash table as an array of employee pointers
EMPLOYEE* stHashTable[MAX_HASH_TABLE_SIZE];

int fnCompHash(int, int);
void fnInsRecord(EMPLOYEE*, int);
EMPLOYEE* fnSrchRecord(int, int);

int main()
{
   int m;  // Size of the hash table
   printf("Enter the size of the hash table (m): ");
   scanf("%d", &m);

   // Initialize the hash table with NULL pointers
   for (int i = 0; i < m; i++)
         {
      stHashTable[i] = NULL;
   }

   FILE* file = fopen("employee.txt", "r");
   if(file == NULL)
         {
      printf("Error opening file.\n");
      return 1;
   }
```

```
    int n = 0;
    EMPLOYEE emp;
    while(fscanf(file, "%d %s", &emp.iKey, emp.cName) != EOF)
        {
        EMPLOYEE* newEmp = (EMPLOYEE*)malloc(sizeof(EMPLOYEE));
        newEmp->iKey = emp.iKey;
        strcpy(newEmp->cName, emp.cName);
        fnInsRecord(newEmp, m);
        n++;
    }

    fclose(file);

    int iSrchKey;
    printf("Enter a key to search for an employee record: ");
    scanf("%d", &iSrchKey);

    EMPLOYEE* found = fnSrchRecord(iSrchKey, m);
    if(found != NULL)
        {
        printf("Employee found with key %d:\n", found->iKey);
        printf("Name: %s\n", found->cName);
    }
    else
        {
        printf("Employee with key %d not found.\n", iSrchKey);
    }

    return 0;
}

void fnInsRecord(EMPLOYEE* emp, int m)
{
    int index = fnCompHash(emp->iKey, m);

    // Linear probing if collisions happen
    while(stHashTable[index] != NULL)
        {
        index = (index + 1) % m;
    }

    stHashTable[index] = emp;
}

int fnCompHash(int iKey, int m)
```

```
{
   return iKey % m;
}

EMPLOYEE* fnSrchRecord(int iKey, int m)
{
   int index = fnCompHash(iKey, m);

   // Linear probing
   while(stHashTable[index] != NULL)
         {
      if(stHashTable[index]->iKey == iKey)
                  {
         return stHashTable[index];
      }
      index = (index + 1) % m;
   }
   return NULL; // Employee record not found
}
```