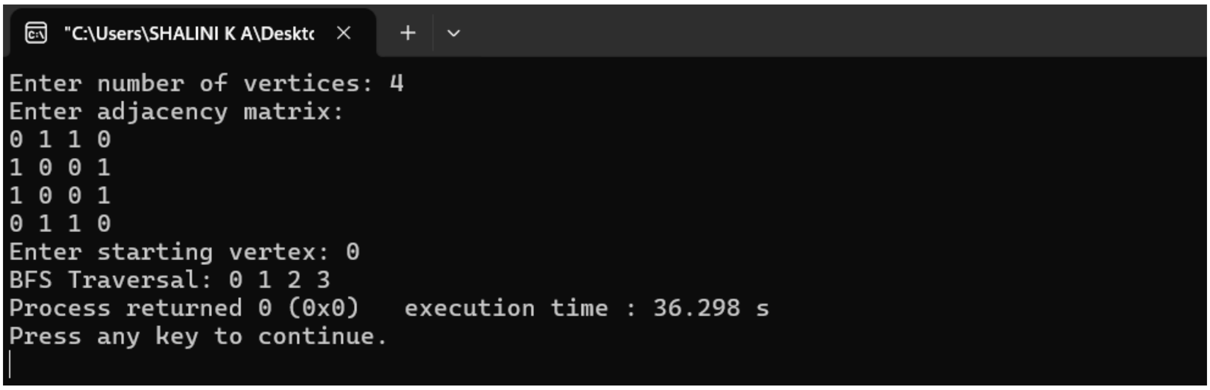
WEEK-10

Program 9:

a) Write a program to traverse a graph using the BFS method.

```
#include <stdio.h>
int queue[20], front = -1, rear = -1;
int visited[20], n;
int graph[20][20];
void enqueue(int v) {
    if (rear == 19)
        return;
    if (front == -1)
        front = 0;
    queue[++rear] = v;
}
int dequeue() {
    return queue[front++];
}
void bfs(int start) {
    int i, v;
    enqueue(start);
    visited[start] = 1;
    printf("BFS Traversal: ");
    while (front <= rear) {
        v = dequeue();
        printf("%d ", v);
        for (i = 0; i < n; i++) {
            if (graph[v][i] == 1 && visited[i] == 0) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
}
int main() {
    int i, j, start;
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        visited[i] = 0;
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter starting vertex: ");
    scanf("%d", &start);
    bfs(start);
    return 0;
}
```

OUTPUT:



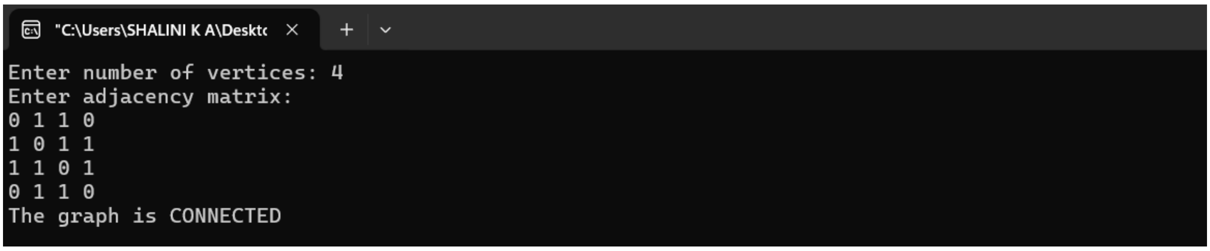
```
Enter number of vertices: 4
Enter adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter starting vertex: 0
BFS Traversal: 0 1 2 3
Process returned 0 (0x0)   execution time : 36.298 s
Press any key to continue.
```

b) Write a program to check whether given graph is connected or not using the DFS method.

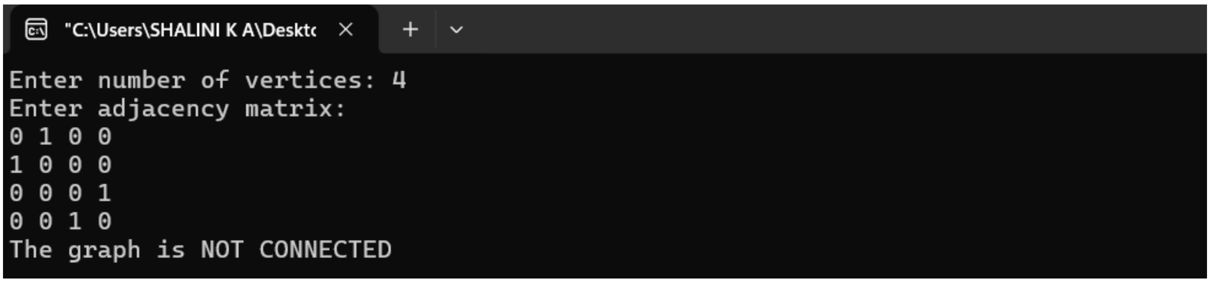
```
#include <stdio.h>
int graph[20][20], visited[20], n;
void dfs(int v) {
    int i;
    visited[v] = 1;
    for (i = 0; i < n; i++) {
        if (graph[v][i] == 1 && visited[i] == 0) {
            dfs(i);
        }
    }
}
int main() {
    int i, j, isConnected = 1;
    printf("Enter number of vertices: ");
    scanf("%d", &n);

    printf("Enter adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        visited[i] = 0;
        for (j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    dfs(0);
    for (i = 0; i < n; i++) {
        if (visited[i] == 0) {
            isConnected = 0;
            break;
}
    }
    if (isConnected)
        printf("The graph is CONNECTED\n");
    else
        printf("The graph is NOT CONNECTED\n");
    return 0;
}
```

OUTPUT:



```
Enter number of vertices: 4
Enter adjacency matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
The graph is CONNECTED
```



```
Enter number of vertices: 4
Enter adjacency matrix:
0 1 0 0
1 0 0 0
0 0 0 1
0 0 1 0
The graph is NOT CONNECTED
```

Program 10:

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.

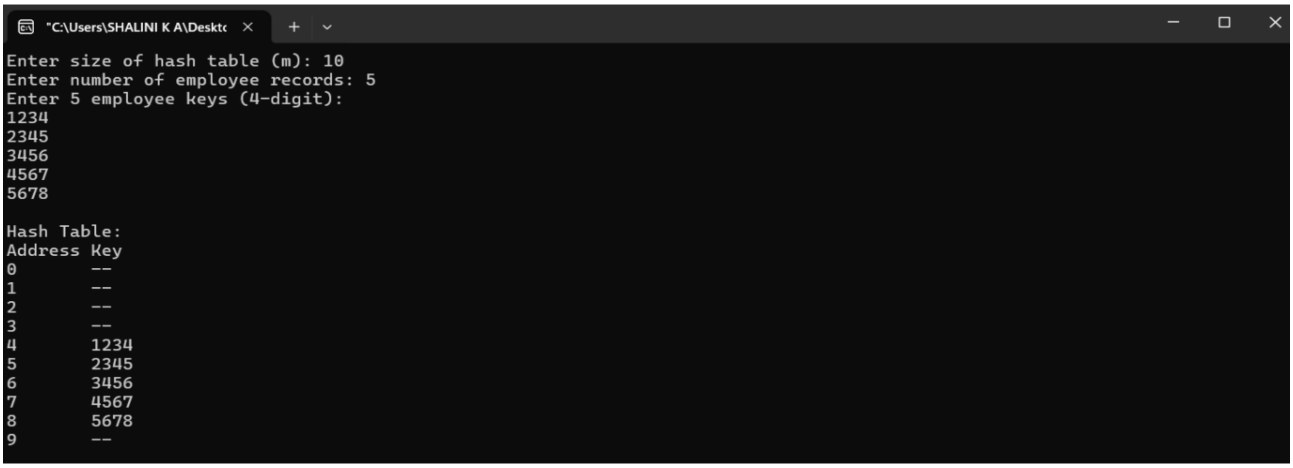
Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K -> L as H(K)=K mod m (remainder method), 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>
#define MAX 50

int hashTable[MAX];
int m;
void insert(int key) {
    int index = key % m;
    int startIndex = index;

    while (hashTable[index] != -1) {
        index = (index + 1) % m;
        if (index == startIndex) {
            printf("Hash table is full. Cannot insert %d\n", key);
            return;
        }
    }
    hashTable[index] = key;
}
void display() {
    int i;
    printf("\nHash Table:\n");
    printf("Address\tKey\n");
    for (i = 0; i < m; i++) {
        if (hashTable[i] != -1)
            printf("%d\t%d\n", i, hashTable[i]);
        else
            printf("%d\t--\n", i);
    }
}
int main() {
    int n, key, i;
    printf("Enter size of hash table (m): ");
    scanf("%d", &m);
    for (i = 0; i < m; i++)
        hashTable[i] = -1;
    printf("Enter number of employee records: ");
    scanf("%d", &n);
    printf("Enter %d employee keys (4-digit):\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &key);
        insert(key);
    }
    display();
    return 0;}
```

OUTPUT:



```
Enter size of hash table (m): 10
Enter number of employee records: 5
Enter 5 employee keys (4-digit):
1234
2345
3456
4567
5678

Hash Table:
Address Key
0       --
1       --
2       --
3       --
4       1234
5       2345
6       3456
7       4567
8       5678
9       --
```