Senjuti Maiti 19S-221

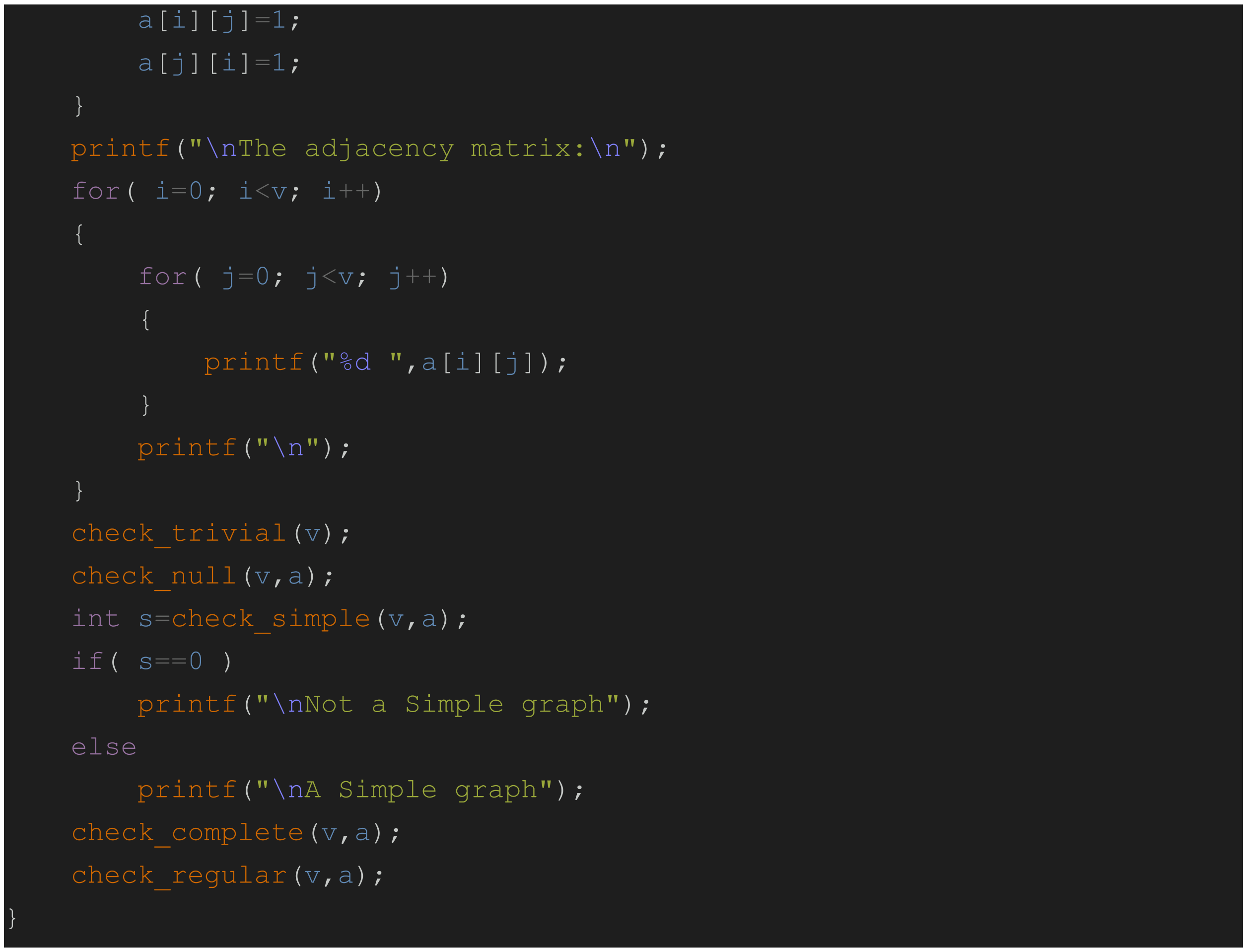
Program statement: Input an undirected, non-weighted graph and write functions to check if it is:
-Null graph
-Trivial graph
-Simple graph
-Regular graph
-Complete graph

Source code:

```c
// A C Program to demonstrate adjacency matrix
// representation of graphs
#include <stdio.h>
#include <stdlib.h>

void check_trivial(int v)
{
    if( v==1 )
        printf("\nA Trivial graph");
    else
        printf("\nNot a Trivial graph");
}

void check_null(int v, int a[v][v])
{
    int i,j,f=1;
    for(i=0; i<v; i++)
    {
        for(j=0; j<v; j++)
        {
            if( a[i][j]==1 )
                f=0;
        }
    }
    if( f==0 )
        printf("\nNot a NuLL graph");
    else
        printf("\nA NuLL graph");
}

int check_simple(int v, int a[v][v])
```

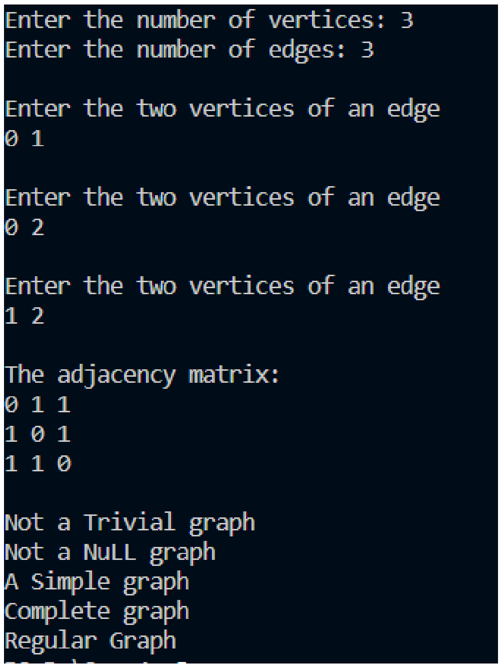
```
{
    int i,f=1;
    for(i=0; i<v; i++)
    {
        if( a[i][i]==1 )
            f=0;
    }
    return f;
}

void check_complete(int v, int a[v][v])
{
    int i,j,f=1;
    for(i=0; i<v; i++)
    {
        for(j=0; j<v; j++)
        {
            if( a[i][j]==0 && i!=j )
                f=0;
        }
    }
    // first need to check if the graph is simple or not
    int s=check_simple(v,a);
    if( f==1 && s==1 )
        printf("\nComplete graph");
    else
        printf("\nNot a Complete graph");
}

void check_regular(int v, int a[v][v])
{
    int i,j,sum1=0,sum2=0,f=1;
    if( a[0][0]==1 )
            sum1=sum1+1;
    for( i=0; i<v; i++ )
    {
        sum1=sum1+a[0][i];
    }
    for( i=1; i<v; i++ )
    {
```

```c
        for( j=0; j<v; j++ )
        {
            // for a self loop degree becomes 2 for an isolated vertex
            if ( i==j && a[i][j]==1 )
                sum2+=1;
            sum2=sum2+a[i][j];
        }
        if(sum1!=sum2)
        {
            f=0;
            break;
        }
        sum2=0;
    }
    if( f==0 )
        printf("\nNot a Regular Graph\n");
    else
        printf("\nRegular Graph\n");
}

int main()
{
    int v,e;
    printf("Enter the number of vertices: ");
    scanf("%d",&v);
    printf("Enter the number of edges: ");
    scanf("%d",&e);
    int a[v][v];
    int i,j,p;
    for( i=0; i<v; i++ )
    {
        for( j=0; j<v; j++ )
        {
            a[i][j]=0;
        }
    }
    for( p=0; p<e; p++ )
    {
        printf("\nEnter the two vertices of an edge\n");
        scanf("%d%d", &i, &j);
```

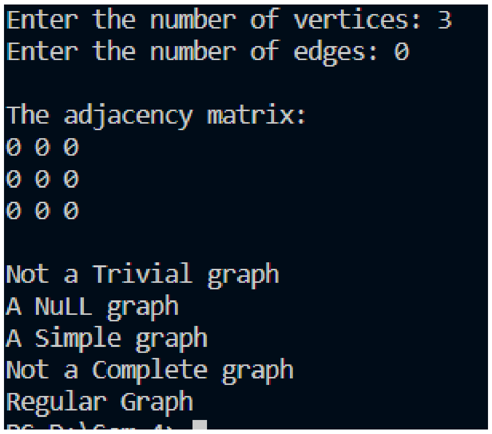


```
            a[i][j]=1;
            a[j][i]=1;
        }
        printf("\nThe adjacency matrix:\n");
        for( i=0; i<v; i++)
        {
            for( j=0; j<v; j++)
            {
                printf("%d ",a[i][j]);
            }
            printf("\n");
        }
        check_trivial(v);
        check_null(v,a);
        int s=check_simple(v,a);
        if( s==0 )
            printf("\nNot a Simple graph");
        else
            printf("\nA Simple graph");
        check_complete(v,a);
        check_regular(v,a);
}
```

Output:

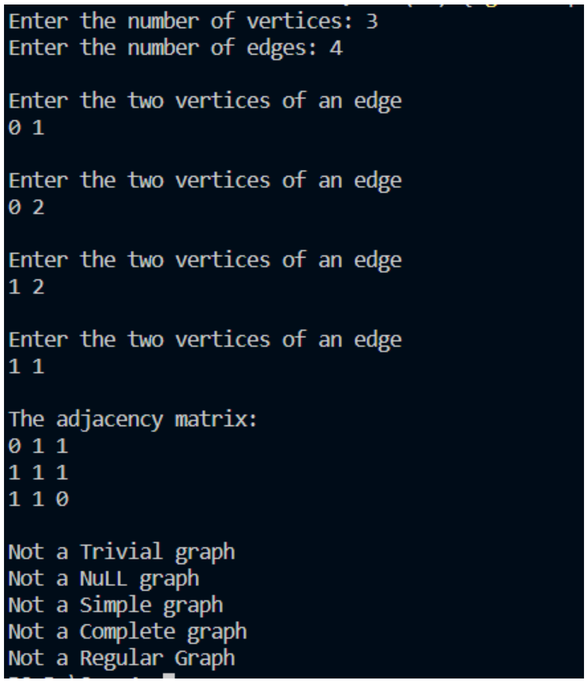


```
Enter the number of vertices: 3
Enter the number of edges: 3

Enter the two vertices of an edge
0 1

Enter the two vertices of an edge
0 2

Enter the two vertices of an edge
1 2

The adjacency matrix:
0 1 1
1 0 1
1 1 0

Not a Trivial graph
Not a NuLL graph
A Simple graph
Complete graph
Regular Graph
```

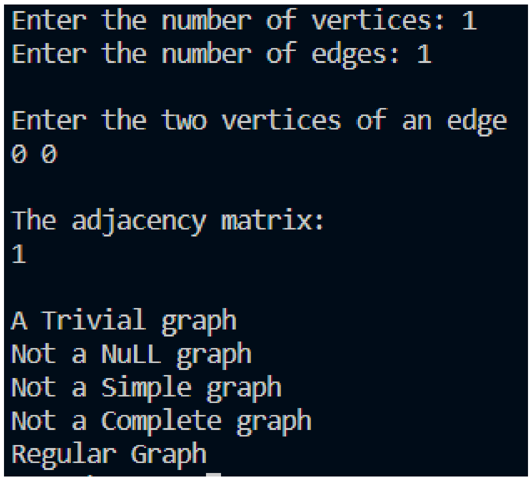


```
Enter the number of vertices: 3
Enter the number of edges: 0

The adjacency matrix:
0 0 0
0 0 0
0 0 0

Not a Trivial graph
A NuLL graph
A Simple graph
Not a Complete graph
Regular Graph
```

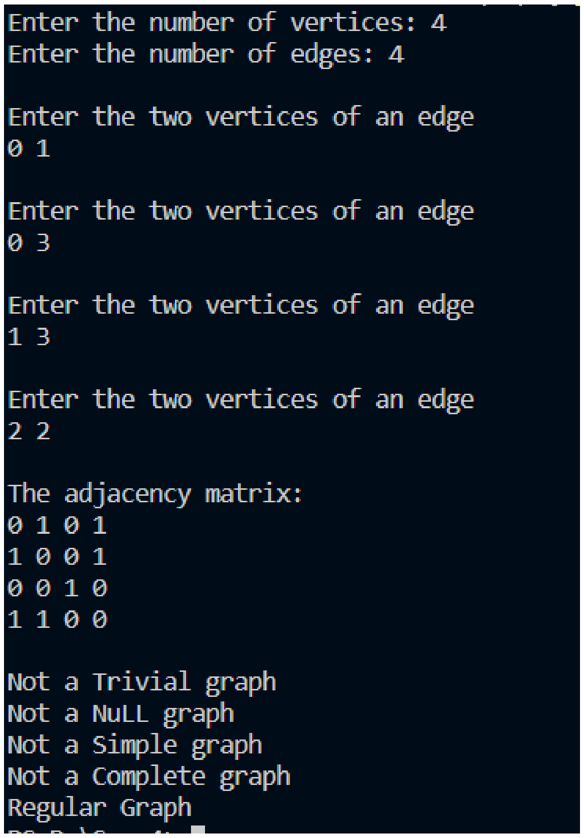
Senjuti Maiti 19S-221



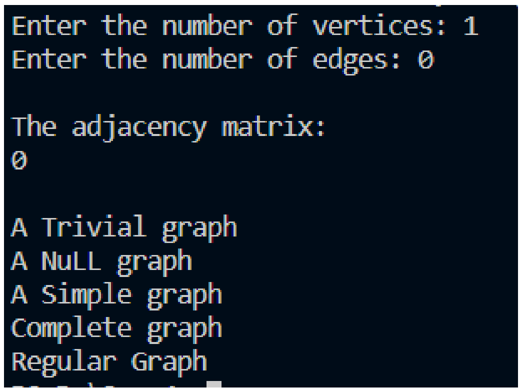
```
Enter the number of vertices: 3
Enter the number of edges: 4

Enter the two vertices of an edge
0 1

Enter the two vertices of an edge
0 2

Enter the two vertices of an edge
1 2

Enter the two vertices of an edge
1 1

The adjacency matrix:
0 1 1
1 1 1
1 1 0

Not a Trivial graph
Not a NuLL graph
Not a Simple graph
Not a Complete graph
Not a Regular Graph
```



```
Enter the number of vertices: 1
Enter the number of edges: 1

Enter the two vertices of an edge
0 0

The adjacency matrix:
1

A Trivial graph
Not a NuLL graph
Not a Simple graph
Not a Complete graph
Regular Graph
```



```
Enter the number of vertices: 4
Enter the number of edges: 4

Enter the two vertices of an edge
0 1

Enter the two vertices of an edge
0 3

Enter the two vertices of an edge
1 3

Enter the two vertices of an edge
2 2

The adjacency matrix:
0 1 0 1
1 0 0 1
0 0 1 0
1 1 0 0

Not a Trivial graph
Not a NuLL graph
Not a Simple graph
Not a Complete graph
Regular Graph
```



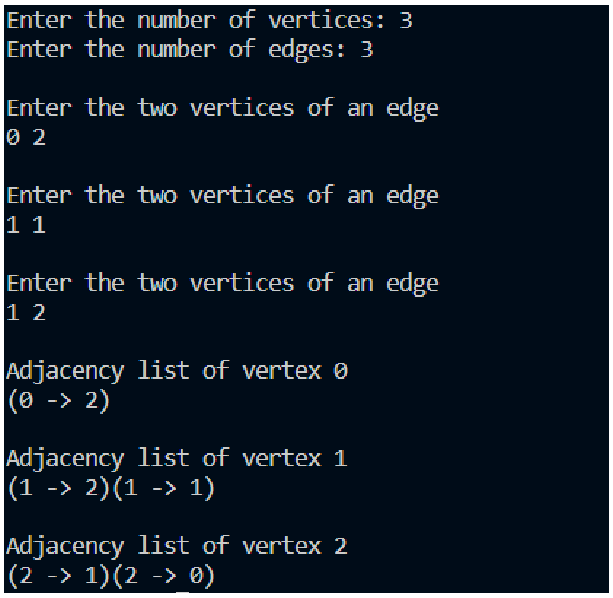
```
Enter the number of vertices: 1
Enter the number of edges: 0

The adjacency matrix:
0

A Trivial graph
A NuLL graph
A Simple graph
Complete graph
Regular Graph
```

Program statement: Adjacency list representation of a non-weighted, directed graph using Linked List.

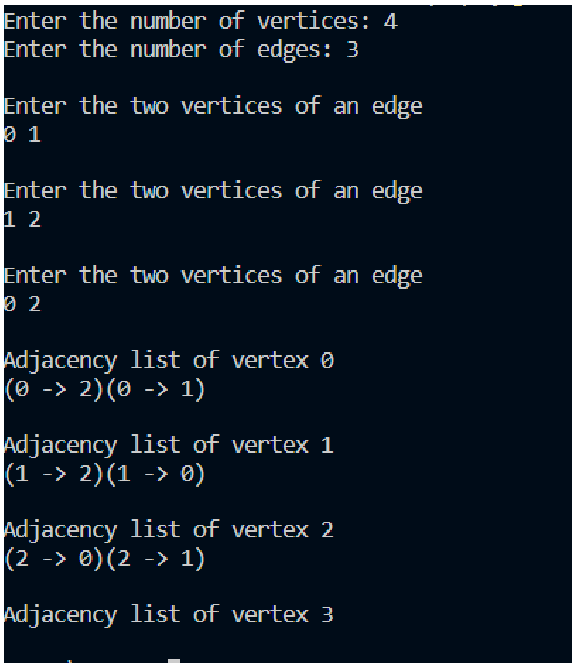
Source code:

```c
// A C Program to demonstrate adjacency list
// representation of graphs using Linked list
#include <stdio.h>
#include <stdlib.h>

// A structure to represent an adjacency list node
struct Node
{
    int dest;
    struct Node* next;
};

// A structure to represent an adjacency list
struct AList
{
    struct Node *head;
};

struct Node* new_AList_Node(int dest)
{
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node) * 1);
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

struct AList* new_Graph()
{
    struct AList* graph;
    graph = (struct AList*) malloc(sizeof(struct AList) * 1);
    graph->head = NULL;
    return graph;
}

void Edge(struct AList* src, struct AList* dest, int s, int d)
{
    struct Node* newNode = new_AList_Node(d);
    newNode->next = src->head;
    src->head = newNode;
    if ( s!=d )
    {
        newNode = new_AList_Node(s);
        newNode->next = dest->head;
        dest->head = newNode;
    }
}

void print(struct AList* graph, int src)
{
    struct Node* temp = graph->head;
    while (temp!=NULL)
    {
        printf("(%d -> %d)", src, temp->dest);
        temp = temp->next;
    }
    printf("\n");
}

int main()
{
    int v,e,i,a,b;
    printf("Enter the number of vertices: ");
    scanf("%d",&v);
    printf("Enter the number of edges: ");
    scanf("%d",&e);
    struct AList* graph[v];
    for( i=0; i<v; i++ )
    {
        graph[i]=new_Graph();
    }
    for( i=0; i<e; i++ )
    {
        printf("\nEnter the two vertices of an edge\n");
        scanf("%d%d", &a, &b);
        Edge(graph[a], graph[b], a, b);
    }
    for( i=0; i<v; i++ )
    {
        printf("\nAdjacency list of vertex %d\n", i);
        print(graph[i], i);
    }
}
```

Output:



```
Enter the number of vertices: 3
Enter the number of edges: 3

Enter the two vertices of an edge
0 2

Enter the two vertices of an edge
1 1

Enter the two vertices of an edge
1 2

Adjacency list of vertex 0
(0 -> 2)

Adjacency list of vertex 1
(1 -> 2)(1 -> 1)

Adjacency list of vertex 2
(2 -> 1)(2 -> 0)
```



```
Enter the number of vertices: 4
Enter the number of edges: 3

Enter the two vertices of an edge
0 1

Enter the two vertices of an edge
1 2

Enter the two vertices of an edge
0 2

Adjacency list of vertex 0
(0 -> 2)(0 -> 1)

Adjacency list of vertex 1
(1 -> 2)(1 -> 0)

Adjacency list of vertex 2
(2 -> 0)(2 -> 1)

Adjacency list of vertex 3
```