Birla Institute of Technology & Science, Pilani
Data Structures & Algo (CS F211)
Online Lab test 2() OPEN book Duration: 90 min Date: 25th April, 2015

A completely connected component K_n of a Graph G(V,E) is a connected component of the graph containing n vertices such that every pair of vertices in Kn have an edge connecting them.

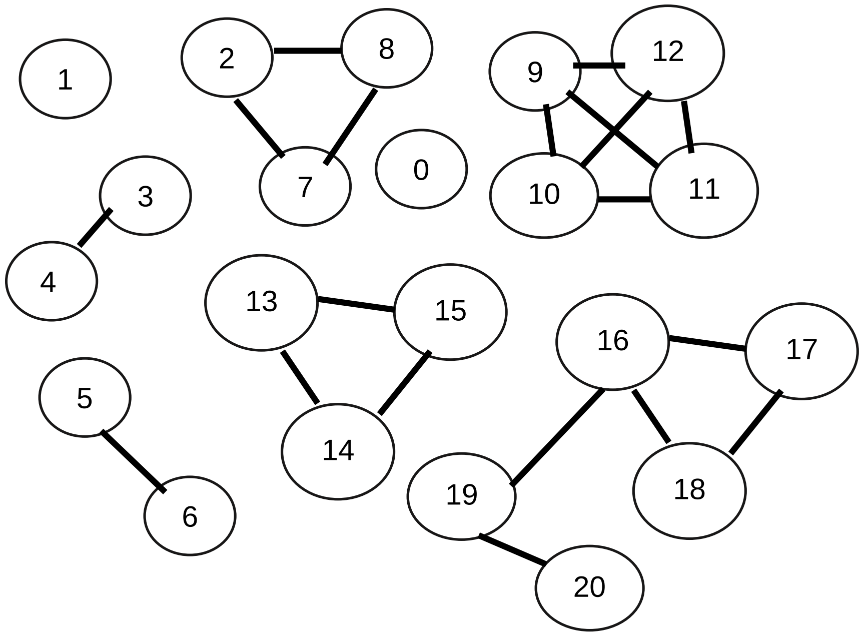
Problem statement:

Given an undirected graph, find the number of completely connected components of the graph of size 1,2,3,4, i.e, count of K_1 K_2 K_3 K_4.

File formats:

Input : input.txt
line 1:number of vertices(n) [numbered from 0 to n]
line 2: number of edges(m)
next m lines, an edge between nodes u and v
Output : output.txt
Four lines, containing the count of K_1, K_2, K_3, K_4 in the four lines.

A sample scenario



Sample Input:
```
21
19
3 4
2 8
7 8
5 6
2 7
9 11
10 11
12 11
13 15
14 15
9 10
9 12
10 12
13 14
16 17
17 18
16 19
19 20
16 18
```

Output for the above graph:
```
2
2
2
1
```

K_1={1}, {0}
K_2={5,6}, {3,4}
K_3={2,7,8}, {13,14,15}
K_4={9,10,11,12}

Overview of the solution:

The graph is represented as adjacency list. The relevant code for creating the graph is already provided to you.

The solution for this problem is achieved via depth first traversal.

In the depth first traversal (DFT), the depth first search (DFS) starting at node v is used to identify connected component in which the vertex is part of. The code for depth first traversal is already provided to you.

In the depth first search, you must keep track of the count of vertices in the current connected component. If the count is less than or equal to 4, this DFS call should return the list of vertices(node numbers) in the connected component. If count is larger than 4, then it should return -1 in all cells.

The DFT code verifies whether the connected component is K_1, K_2, K_3, K_4 and appropriately modifies the count.

Codes to write:

1) **`int isNeighbourOf(Graph G, int u, int v):`** This function determines whether the vertices identified by **`u`** and **`v`** are neighbours or not. This function returns 1 (for true) or 0 (for false).
2) **`int DFS(Graph G, int v, int result[4]):`** This function performs DFS search starting at node with node number as **`v`**. The list of the vertices visited are maintained in the array **`result`**. If only one node is in this connected component, only result[0] will be having valid node number. If two nodes are in this connected component, result[0] and result[1] will be having valid node numbers. If three nodes are in this connected component, result[0], result[1] and result[2] will be having valid node numbers.
3) **`int checkK3(Graph G, int result[4]):`** This function verifies whether the three vertices in the array **`result`** form a K_3 or not. This function returns 1 (for true) or 0 (for false).
4) **`int checkK4(Graph G, int result[4]):`** This function verifies whether the four vertices in the array **`result`** form a K_4 or not. This function returns 1 (for true) or 0 (for false).

All code is provided in a file a named **`driver.c`**

`Deliverables: driver.c ,input.txt, output.txt`