<u>IT251 Assignment – 4</u>

NAME: SUYASH SATISH CHINTAWAR

ROLL NO.: 191IT109

TOPIC: APPLICATIONS OF DFS

VARIABLES:

- **n** – number of vertices in the undirected graph
- **edges** – number of edges in the undirected graph
- **edge[edges][2]** – array to store each pair of edges
- **adj_list[n]** – vector of vector consisting of adjacency list of each vertex
- **visited_dfs** – boolean vector to keep track of visited vertices when DFS is carried out.
- **v** – starting vertex for any DFS call
- **n_comp** – total number of components in the graph
- **comp_no** – integer to keep track of the current component number
- **visited_2ec** – boolean vector to keep track of visited vertices when DFS is carried out for 2-edge connectedness.
- **arrival** – integer vector to keep track of arrival times of each vertex during DFS.
- **time** – integer which tracks time
- **flag** – integer to know whether graph is 2-edge connected or not. Value is either 0/1.
- **bridge** – vector of pair of integers which stores all the bridge edges in the graph if any.
- **parent** – integer which stores parent vertex of current vertex
- **deepest_be** – stores arrival time of deepest back edge.
- **i,j,x** – simple iterators.

FUNCTIONS:

- **adjacency_list()** – Compute adjacency lists of all vertices
- **two_edge_conn()** – Perform DFS to know whether graph is 2-edge connected or not and to find all the bridge edges if not 2-edge connected
- **dfs()** – Perform usual DFS which spans one component at one call
- **dft()** – Perform DFS(by calling above two functions) to check connectedness and 2-edge connectedness of the graph. This function controls above two functions.

README and assumptions

The program checks 2-edge connectedness of undirected graphs using the following steps:

- Adjacency list is generated using the edge set given by the user by calling **adjacency_list().**
- **dft()** is then called and assuming starting vertex to be 1(vertices are 1,2…,n), **dfs()** is called on the whole graph to check the number of components in it.
- If the number of components is greater than one, then the graph is disconnected and it is shown in the output along with the number of components. In this case, **two_edge_conn()** is called on each of the components and are checked individually for 2-edge connectedness, and the

bridges are outputted if they exist which are stored in **bridge.**

- In the above case, for each component, **time** and **flag** gets initialized to zero, and **bridge** gets cleared to fill new entries of the current component.
- The similar process is followed even if the number of components in the graph is one, only the difference is, **two_edge_conn()** is called only once which spans the whole graph and process is similar for the output too.
- Whenever **two_edge_conn()** is called, the flag is later checked and if it is one, bridge edge is present and hence the graph/component is not 2-edge connected. And if **flag**=0, then the graph/component is 2-edge connected.

The overall time complexity is O(V+E), i.e. linear, where V is number of vertices and E is number of edges.

Output screenshots for the given test case in the problem are attached in the folder.