```c
/*

Esercizio 2 Grafi

Scrivere un programma che legga da tastiera un grafo indiretto e stampi 1
se il grafo è connesso, 0 altrimenti. Il grafo è rappresentato nel seguente
formato: la prima riga contiene il numero n di nodi, le successive n righe
contengono, per ciascun nodo i, con 0 □ i < n, il numero n i di archi uscenti
da i seguito da una lista di n i nodi destinazione, rappresentati con i numeri
[0, n). Si assuma che l'input contenga un grafo indiretto, e quindi che per
ciascun arco da i a j esiste anche l'arco da j ad i.
Un grafo è connesso quando esiste un percorso tra due vertici qualunque
del grafo. Il programma deve eseguire una visita DFS (a partire da un nodo
qualunque, perché?) del grafo per stabilire se questo è connesso.

Antonio Boffa (a.boffa@studenti.unipi.it)

*/

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

typedef struct _edges
{
    int num_edges; // numero di nodi adiacenti → out-degree
    int *edges;    // array dei nodi adiacenti
} edges;

// funzione che dealloca il grafo.
// E : grafo
// n : numero di nodi
void free_graph(edges *E, int n)
{
    int i;
    for (i = 0; i < n; ++i)
    {
        free(E[i].edges);
    }
    free(E);
}

// funzione che legge il grafo da input.
// n : numero di nodi
edges *read_graph(int n)
{
    edges *E;
    int ne, i, j;
    E = (edges *)malloc(sizeof(edges) * n);
    for (i = 0; i < n; ++i)
    {
        scanf("%d", &(ne));
        E[i].num_edges = ne;
        E[i].edges = (int *)malloc(sizeof(int) * ne);
        for (j = 0; j < ne; ++j)
```

```c
        {
            scanf("%d", E[i].edges + j);
        }
    }
    return E;
}

// dfs che restituisce il numero di nodi visitati
// E : grafo da visitare
// n = numero di nodi
// from : nodo da cui iniziare la ricerca
int dfs_num_visited_nodes(edges *E, int n, int from)
{
    int *colors = (int *)malloc(sizeof(int) * n);
    int *stack = (int *)malloc(sizeof(int) * n);
    int stack_size, src, dest, i;
    for (i = 0; i < n; ++i)
        colors[i] = 0;
    colors[from] = 1;
    stack[0] = from;

    int num_visited_nodes = 1;

    stack_size = 1;

    while (stack_size) // finchè ≠ 0
    {
        src = stack[--stack_size];
        for (i = 0; i < E[src].num_edges; ++i)
        {
            dest = E[src].edges[i];
            if (!colors[dest]) // se == 0 → non visitato
            {
                num_visited_nodes++;
                colors[dest] = 1;
                stack[stack_size++] = dest;
            }
        }
    }

    free(stack);
    free(colors);

    return num_visited_nodes;
}

int main()
{
    int num_nodes;
    // leggo il numero di nodi da input
    scanf("%d", &(num_nodes));
    // leggo il grafo da input
    edges *edges = read_graph(num_nodes);
    // nodo casuale da cui partire
    int starting_node = 0;
```

```c
    // far partire una dfs per ottenere il numero di nodi "colorati"
    int num_visited_nodes = dfs_num_visited_nodes(edges, num_nodes, starting_node);
    // numero di nodi "colorati" < numero di nodi effettivi del grafo
    if (num_visited_nodes < num_nodes)
    {
        // non ho "colorato" tutti i nodi
        printf("NON CONNESSO\n");
    }
    else
    {
        // ho "colorato" tutti i nodi → num_visited_nodes == num_nodes
        printf("CONNESSO\n");
    }
    free_graph(edges, num_nodes);
    return 0;
}

```