```c
/*
http://didawiki.cli.di.unipi.it/lib/exe/fetch.php/informatica/all-b/esercizigrafi19.pdf

Sia G = (V, E) un grafo connesso e non orientato. Progettare un algoritmo che ricevuto in ingresso G e un suo vertice r, restituisca il numero di vertici che si trovano a distanza massima da r.

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

*/

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

// ------------------------------------------------
// Definizione della coda FIFO utilizzata nella BFS

typedef struct _queue
{
    // array dove verrà mantenuto il contenuto della coda
    int *values;
    // capacità massima della coda
    int capacity;
    // indice della posizione dell'array disponibile per l'operazione pop
    int head;
    // indice della posizione dell'array disponibile per l'operazione push

    int tail;
    // numero di elementi nella coda (non utilizzata qui)
    int size;
} queue;

void queue_init(queue *q, int capacity);
void queue_deinit(queue *q);
void queue_pushBack(queue *q, int value);
int queue_popFront(queue *q);
int queue_isEmpty(queue *q);
int queue_size(queue *q);

void queue_init(queue *q, int capacity)
{
    q->capacity = capacity;
    q->values = (int *) malloc(capacity * sizeof(int));
    q->tail = 0;
    q->head = 0;
    q->size = 0;
    return;
}
```

```c
void queue_deinit(queue *q)
{
    free(q->values);
    return;
}
void queue_pushBack(queue *q, int value)
{
    q->values[q->tail] = value;
    q->tail++;
    q->size++;
    return;
}

int queue_popFront(queue *q)
{
    int to_return = q->values[q->head];
    q->head++;
    q->size--;
    return to_return;
}

int queue_isEmpty(queue *q)
{
    if(q->tail == q->head)
        return 1;
    else
        return 0;
}

int queue_size(queue *q){
    return q->size;
}

// _____________________________________________
// Fine della definizione della coda FIFO utilizzata nella BFS


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

// bfs che restituisce il numero di nodi alla distanza massima dal nodo from
// E : grafo
// n : numero di nodi
// from: nodo dal quale partire
int bfs(edges *E, int n, int from)
{
   // salvo nell'array distanze la distanza dall'origine (from)
   int *distanze = (int *)malloc(sizeof(int) * n);
   queue q;
   int src, dest, i;
   // inizializzo le distanze
   for (i = 0; i < n; ++i)
      distanze[i] = -1; // non ancora visitati
   distanze[from] = 0; // distanza tra l'origine e l'origine è 0
   // inizializzo la coda
   queue_init(&q, n);
   queue_pushBack(&q, from);
   // numero di nodi alla distanza massima in questo momento
   int num_nodes_max_dist = 0;
   // distanza massima in questo momento
   int curr_max_dist = 0;
   // loop fino a terminazione della coda
   while (!queue_isEmpty(&q))
```

```
    {
        // src -> nodo corrente
        src = queue_popFront(&q);
        for (i = 0; i < E[src].num_edges; ++i)
        {
            // dest -> nodo vicino al nodo corrente
            dest = E[src].edges[i];
            if (distanze[dest] == -1) // dest non è già stato visitato
            {
                // un nodo (dest) ha la distanza dall'origine=alla distanza del
                // nodo dal quale è arrivato (src) + 1
                distanze[dest] = distanze[src] + 1;
                queue_pushBack(&q, dest);

                // se la distanza dall'origine del nodo src è maggiore del
                // distanza massima incontrata fino ad ora -> aggiorno!
                if(distanze[src] > curr_max_dist){
                    curr_max_dist = distanze[src];
                    num_nodes_max_dist = 0;
                }
                // in ogni caso ho incontrato un nuovo nodo a distanza massima
                num_nodes_max_dist++;
            }
        }
    }
    // stampo le distanze
    for (i = 0; i < n; ++i)
    {
        printf("Distanza del nodo %d dal nodo %d = %d \n", i, from, distanze[i]);
    }
    // libero la memoria
    queue_deinit(&q);
    free(distanze);
    return num_nodes_max_dist;
}

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);
    int starting_node;
    // leggo il nodo da cui partire
    scanf("%d", &(starting_node));
    // faccio calcolare il risultato dell'esercizio dalla funzione bfs
    int num_nodes_max_distance = bfs(edges, num_nodes, starting_node);
```

```
    printf("\nNumero di nodi alla distanza massima = %d\n", num_nodes_max_distanc
e);
    free_graph(edges, num_nodes);
    return 0;
}
```