Esercizio 2

Grafo connesso

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 $0 \leq i < n$, 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.

Esercizio 2

```
int dfs_coloring(edges *E, int n) {
   int * colors = (int *) malloc(sizeof(int) * n);
   int * stack = (int *) malloc(sizeof(int) * n);
   int stack_size, src=0, dest, i;
   // inizializzo i colori
   for (i=0; i < n; ++i) colors[i] = 0;
   colors[src] = 1;
   // inizializzo lo stack
   stack[0] = src; stack_size = 1;
   // loop fino a terminazione dello stack
   while (stack_size) {
    src = stack[--stack_size];
    for (i=0; i < E[src].num; ++i) {
          dest = E[src].edges[i];
          if (!colors[dest]) {
               colors[dest] = 1;
               stack[stack_size++] = dest;
          }
    }
   }
```

Esercizio 2

```
    int result = 1;
    for (i=0; i < n; ++i) {
     if (!colors[i]) {
           result = 0;
           break;
     }
    }
    // libero la memoria
    free(stack);
    free(colors);
    return result;
}

int main() {
    int n;
    edges * E = read_graph(&n);

    printf("%d\n", dfs_coloring(E, n));
    return 0;
}
```

Esercizio 3

Percorso minimo

Scrivere un programma che legga da tastiera un grafo diretto, una sequenza di m query composte da due indici ciascuna e stampi, per ciascuna query, la lunghezza del percorso minimo che collega i rispettivi due nodi della query. 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 $0 \leq i < n$, il numero n_i di archi uscenti da i seguito da una lista di n_i nodi destinazione, rappresentati con i numeri $[0, n)$.

Il percorso minimo dal nodo i al nodo j è il percorso che porta da i a j avente il minor numero di nodi. A tale scopo si esegua una visita BFS del grafo a partire dal nodo i per stabilire il percorso minimo che porta al nodo j, qualora questo esista.

Esercizio 3

```
typedef struct _queue {
    int * elements;
    int size;
    int head;
    int tail;
} queue;

void init_queue(queue * Q, int size) {
    Q->elements = (int *) malloc(sizeof(int) * size);
    Q->size = size;
    Q->head = Q->tail = 0;
}

void deinit_queue(queue * Q) {
    free(Q->elements);
    Q->size = 0;
    Q->head = Q->tail = 0;
}
```

Esercizio 3

```
void enqueue(queue * Q, int element) {
    Q->elements[Q->tail++] = element;
}

int dequeue(queue * Q) {
    return Q->elements[Q->head++];
}

int bfs_distance(edges *E, int n, int from, int to) {
    if (from == to) {
     return 0;
    }
    int * dist = (int *) malloc(sizeof(int) * n);
    queue q;
    int src, dest, i;
    // inizializzo le distanze
    for (i=0; i < n; ++i) dist[i] = -1;
    dist[from] = 0;

     // continua …
```

Esercizio 3

```
// inizializzo la coda
init_queue(&q, n); enqueue(&q, from);
// loop fino a terminazione della coda
while (q.head != q.tail) {
  src = dequeue(&q);
  for (i=0; i < E[src].num; ++i) {
        dest = E[src].edges[i];
        if (dist[dest] == -1) {
              dist[dest] = dist[src] + 1;
              if (dest == to) {
                    int result = dist[dest];
                    deinit_queue(&q);
                    free(dist);
                    return result;
              }
              enqueue(&q, dest);
        }
  }
}
// continua …
```

Esercizio 3

```
    // libero la memoria
    deinit_queue(&q);
    free(dist);
    return -1;
}

int main() {
    int n;
    edges * E = read_graph(&n);

    int m, from, to;
    scanf("%d", &m);
    for (int i=0; i < m; ++i) {
      scanf("%d%d", &from, &to);
      printf("%d\n", bfs_distance(E, n, from, to));
    }
    return 0;
}
```