PWr Week 12
Data Structures and Algorithms
Laboratory – **List 11**

To the applications from previous list (List 10) add the following functions:

1. **`void`** `BFS(Graph &g, int s)` - implementation of breadth-first search starting from vertex `s`. Assumptions: the procedure analyze adjacency vertices in ordered sequence sorted by its numbers. The function have to write on console the order of visited (first time) vertices. After every number of vertex, it has to be a comma.
2. **`void`** `DFS(Graph &g, int s)` - implementation of depth-first search starting from vertex `s`. Assumptions: the procedure analyze adjacency vertices in ordered sequence sorted by its numbers. The function have to write on console the order of visited (first time) vertices. After every number of vertex, it has to be a comma.

   Assumptions: There is a path from s to every other vertex.
   Note: in these tasks we do not use weights.

For **10 points** present solutions for this list till **Week 13**.
For 8 **points** present solutions for this list till **Week 14**.
For 5 **points** present solutions for this list till **Week 15**.
After Week 15 the list is closed.

Appendix 1

The solution will be automated tested with tests from console of presented below format. The test assumes, that there are up to X different graphs, which there are created as the first operation in the test. Each graph will be loaded from input stream.

If a line starts from '`#`' sign, the line have to be ignored.
In any other case, your program should print an exclamation mark and write (copy) introduced a line and then, depending on the command follow the correct procedure / function.

If a line has a format:
`GO` *X*
your program has to create *n* graphs (without initialization). The graphs are numbered from 0 like an array of lists. Default current graph is a graph with number 0. This operation will be called once as the first command.

If a line has a format:
`CH` *n*
your program has to choose a graph of a number `n`, and all next functions will operate on this graph. There is $n>=0$ and $n<X$.

If a line has a format:
`LG n m`
your program has to call `loadGraph(g, n, m)` for current graph g. For any graph this operation will be called once, before using the graph. **The next `m` lines will present the information about edges.**

If a line has a format:
`IE` *u v w*
your program has to call `insertEdge(g,u,v,w )` for current graph `g`.

If a line has a format:
`FE u v`
your program has to call `findEdge(g,u,v,w)` for current graph `g`, and if the function return **`true`**, write on the output returned value `w`. Otherwise write "false" with new line character.

If a line has a format:
`SM`
your program has to call `showAsMatrix(g)` for current graph `g`.

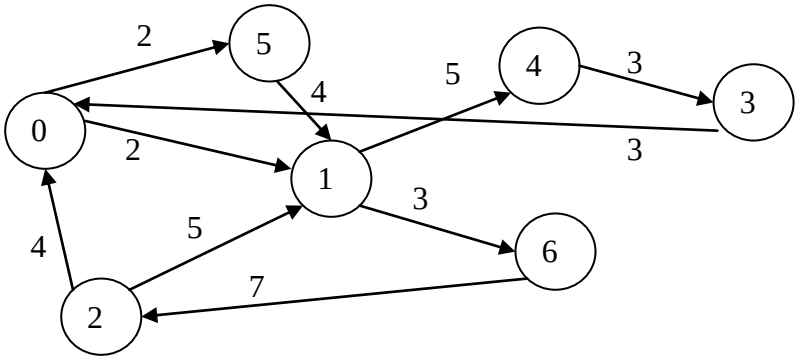
If a line has a format:
`SA`
your program has to call `showAsArrayOfLists(g)` for current graph `g`.

If a line has a format:
`HA`
your program has to end the execution, writing as the last line "END OF EXECUTION". Every test ends with this line.

If a line has a format:
`BF u`
your program has to call `BFS(g,u)` for current graph `g`.

If a line has a format:
`DF u`
your program has to call `DFS(g,u)` for current graph `g`.

A graph from example test:



For example for input test:

```
GO 2
LG 7 10
0 5 2
5 1 4
1 4 5
4 3 3
3 0 3
0 1 2
1 6 3
2 0 4
2 1 5
6 2 7
BF 2
DF 2
HA
```

The output have to be:

```
START
!GO 2
!LG 7 10
!BF 2
2,0,1,5,4,6,3,
!DF 2
```

```
2,0,1,4,3,6,5,
!HA
END OF EXECUTION
```