# Lab. 08 - Graph Problems (1)

## 1. Graph Reading

Consider the undirected weighted graphs specified in the zip file **graphs.zip**, with the following DIMACS format

```
nn na
ni nj wij
```

where the first line indicates the number **nn** of nodes and the number **na** of arcs, and the subsequent lines specify all the **na** arcs, each by a triple `<ni, nj, wij>` where **ni** and **nj** are the node identifiers and **wij** the weight of the connecting arc. Note that the graphs are undirected, and so for any arc between nodes **i** and **j** there is an arc between nodes **j** and **i** with the same weight.

Specify a function with signature

```
def dimacs_read(filename)
```

that reads a graph with the above DIMACS format from a file with name filename and returns the adjacency matrix A of the represented graph. In the adjacency matrix, unconnected nodes (including self-connections) are denoted with -1.

**Note:** The files format assume that the nodes are numbered from **1** no **nn**.

## 2. Graph Writing

Consider an undirected graph specified by its adjacency matrix A. Specify a function with signature

```
def dimacs_write(A, fname)
```

that prints the graph **M** in a file with the given **filename**, with the DIMACS format (see above). Test this function with the graphs obtained in the previous question.

## 3. Graph Encodings

a) Implement function with signature

```
def matrix_to_dicts_convert(A)
```

that, for a given **A**, the adjacency matrix encoding of a graph, returns its dictionary list encoding.

b) Implement function with signature

```
def dicts_to_matrix_convert(D)
```

that, for a given **D**, the dictionary list encoding of a graph, returns its adjacency matrix encoding.

Test your codes with the matrix shown in the slides of class 9.

## 4. Subgraph Projection

Consider an undirected graph specified by its adjacency matrix **A**. Implement a function with signature

```
def subgraph_projection(A, Nodes)
```

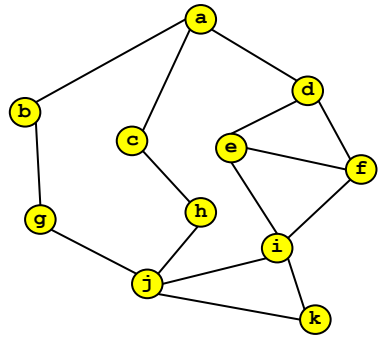
that returns the adjacency matrix **S** of the subgraph of **A** obtained by its projection to the list **Nodes.**

**Note 1:** Notice that matrix **S** of the subgraph should have the nodes numbered from **0**. If **A** encodes a graph with nodes **0..9**, and **Nodes** is **[2,5,7]** than **S** has **3** rows and columns and **Nodes** is a mapping from the new nodes to the original ones (i.e. nodes **0**/**1**/**2** of **S** are the original nodes **2**/**5**/**7** of **A**).

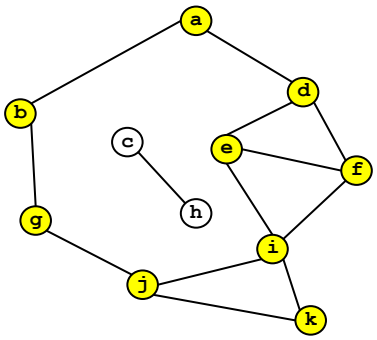
**Note 2**: When the **n** nodes of a graph have labels different from the natural numbers **0..n-1**, than the graph should be represented by a pair **G = (A,M)** where **A** is an adjacency matrix and **M** the mapping of the nodes to their intended labels.

For graphs G1 and G2 below, solve the following problems:

**G1:**




**G2:**




## 5. Spanning trees

1. Use Prim's algorithm studied in the lecture to obtain the minimum spanning tree (MST) of a graph, e.g. G7, in the form of an adjacency matrix.

```
prim(G7)
```

2. What is the length of the spanning tree obtained? Use a function with signature

```
def mst_length(G):
```

3. Assume that a graph G corresponds to a distribution network, for which some redundancy is intended. For this purpose, specify a program that obtains, in the form of an adjacency matrix, the two minimal spanning trees that share no arc between them (if they do not exist, return an empty list for each non existing MST). Use function with signature:

```
def prim_2(G):
```

**Hint**: Obtain an MST, remove the arcs, and obtain a second MST

**Note:** Test your program in graph G8, obtained by adding the following arcs to G7:

<a,e> = 12;  <c,d> = 7;  <c,g> = 16;

## 6. Graph Distances

Use the Floyd-Warshall's algorithm to

a) Identify the most central node of the graph;

```
def most_central(G):
```

b) Identify the pair of nodes of the graph more far apart; and

```
def longest_distance(G):
```

c) Obtain the path between these nodes.

```
def longest_path(G):
```

## Other Graphs

Repeat the above problems for other graphs in graphs.zip file, namely

- graph_10_10.txt, graph_10_20.txt graph_10_50.txt graph_10_60.txt, graph_10_90.txt.
- graph_15_10.txt, graph_15_20.txt graph_15_50.txt graph_15_60.txt, graph_15_90.txt.
- graph_20_10.txt, graph_20_20.txt graph_20_50.txt graph_20_60.txt graph_20_90.txt.
- graph_50_10.txt, graph_50_20.txt graph_20_50.txt graph_50_60.txt graph_50_90.txt.
- graph_100_10.txt, graph_100_20.txt graph_20_50.txt graph100_60.txt graph_100_90.txt.