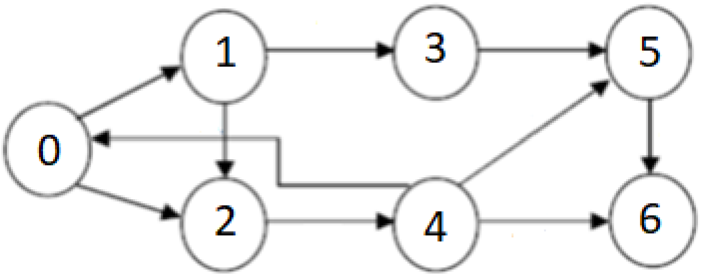
PRACTICAL TUTORIAL
WORKSHEET N°5: GRAPHS

Exercise 1:

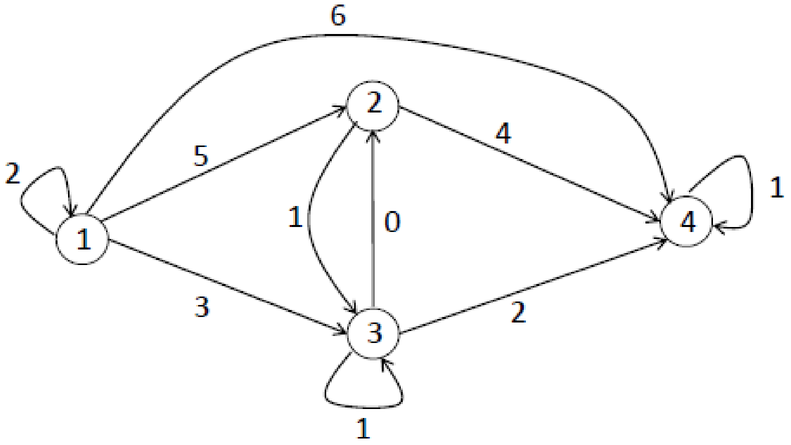
Consider the following direct graph:



1. Calculate its density.
2. Give graph representation in the form of adjacency matrix then in the form of adjacency lists (linked and contiguous) and analyze the spatial complexity of each representation.
3. Write for each representation the algorithm that test if two given nodes u and v are neighbors then analyze the time complexity of these algorithms.

Exercise 2:

Consider the following graph:



1. Unroll the BFS traversal algorithm from node 1.
2. Unroll the DFS traversal algorithm from node 1.
3. Give the complexity of these algorithms depending on number of nodes n and number of arcs m.

Exercise 3:

Consider a non-weighted directed graph G represented by adjacency matrix with the following C declaration:

```
#define N 999
typedef struct g{
      int nbNodes,nbArcs;
      int Mat[N][N];
} Graph;
```

Give the functions and their complexities which allow to:

a. Return the complementary graph of a given graph.
b. Modify the input graph by deleting all loops (must be recursive).

Exercise 4:

A source node is a node that has no predecessors. Given a directed graph of n nodes and m arcs represented in adjacency linked lists, write a Boolean function to test whether a given node is a source node. Give declaration instructions and analyze the complexity of your function.

Exercise 5:

Apply Dijkstra's algorithm on the graph of exercise 2 from node 1.

Exercise 6:

A variant of the adjacency list data structure allow to represent the set of nodes by a linked list, where each node is associated with a pointer to its successor which belongs to the linked list of nodes, as in the following example. Declare this structure then write the **addArc(u,v)** function.