Assignment 2 – deadline: lab 3 (week 5-6)

Modify the graph structure implemented at Assignment 1 in the following way:

a) Make the possibility for the graph to be undirected:
   - add a parameter in __init__ to control if the graph is directed or not.
   - Change the behaviour of all functions that care about the direction of an edge so that the graph will behave as expected for an undirected graph.
   - make sure to change ***create_random*** accordingly.

b) Make the possibility for the graph to be weighted:
   - add a parameter in __init__ to control if the graph is weighted or not.
   - create function ***random_weights*** to assign random weights to all edges. This function should take a parameter that controls the range of the random weights (i.e. a pair like [0,10], [-5, 5] etc.). ***random_weights*** should change the graph to be weighted.
   - create function ***set_weight*** to set the weight for an edge. This function should raise an error if the graph is not weighted.
   - change ***create_random*** to take an optional *weights_range* parameter to control the range of random weights if the graph is directed. If the graph is unweighted and the parameter is provided with a value, just ignore the parameter. If the graph is weighted but no value is provided you can either give a default range (e.g. [0, 10]) or raise an error, however you prefer.

c) Create function ***iter_vertex*** that, given a vertex as a parameter, creates an iterator that iterates through all the vertices reachable starting from the given vertex:
   - The iterator is a separate class, that is initialized with the graph and the starting vertex, and which has the following functions:
     - ***first*** **–** set the iterator on the first vertex (the given vertex).
     - ***get_current*** – returns the current vertex. Raises an error if the iterator is invalid.
     - ***next*** – goes to the next vertex. Raises an error if the iterator is invalid.
     - ***valid*** – returns True if the iterator is valid or False otherwise.
   - The order of the traversal is one of the following:
     1) **Breadth First Search (BFS)**
     2) **Depth First Search (DFS)**

     Please see the attendance table to find your assigned traversal algorithm for the iterator.
   - Bonus (1p): add function ***get_path_legth*** that returns the path length from the initial vetex to the current vertex. The function should run in Theta(1) and implementing this should not significantly impact the time complexity of other functions of the iterator. For BFS this path length should be the distance.

d) Add function ***create_from_file*** that creates a graph with the data read from a file. The data from the file has the following format:
    - On the first line are the number ***n*** of vertices, the number ***m*** of edges, if it is directed (T for True and F for False) and if it is weighted (T for True and F for False)
    - On each of the following ***m*** lines there are two or three numbers: x, y, and w if it is weighted, describing an edge: the initial vertex, the terminal vertex, and the weight of that edge (only if it is weighted)