9th Recitation

Graphs Representation and Manipulation

Instructions:

- Download the file `aed2324_p09.zip` from the course page and unzip it (it contains the `lib` folder, the `Tests` folder with the files `Graph.h`, `funWithGraphs.h`, `funWithGraphs.cpp`, `Person.h`, `Person.cpp`, and `tests.cpp`, and the files `CmakeLists.txt` and `main.cpp`);
- In CLion, open a project by selecting the folder containing the files from the previous point;
- If you can't compile, perform "Reload CMake Project" on the `CMakeLists.txt` file;
- Implement it in the file `Graph.h` and `funWithGraphs.cpp;`
- Note that all the tests are uncommented. They should fail when run for the first time (before implementation). As you solve the exercises, the respective tests should pass.

1. Graphs: representation and CRUD

Consider the Graph class below, as defined in the `Graph.h` file:

```
template <class T> class Vertex {
    T info;
    vector<Edge<T> > adj;
public:
    //...
    friend class Graph<T>;
};

template <class T> class Edge {
    Vertex<T> * dest;
    double weight;
public:
    //...
    friend class Graph<T>;
    friend class Vertex<T>;
};
```

```
template <class T> class Graph {
    vector<Vertex<T> *> vertexSet;
    //...
public:
    //...
};
```

1.1) In the `Graph` class, implement the member function below:

```
bool addVertex(const T &in)
```

This function adds the vertex with content `in` (*info*) to the graph. It returns `true` if the vertex was successfully added to the graph and `false` if the graph already includes a node with the same content.

1.2) In the `Graph` class, implement the member function below:

```
bool addEdge(const T &sourc, const T &dest, double w)
```

This function adds to the graph an edge originating at vertex `sourc`, ending at vertex `dest`, and weight `w`. It returns `true` if the edge was successfully added and `false` if it is not possible to insert that edge (because one or both of the *source* and *destination* vertices do not exist).

1.3) In the `Graph` class, implement the member function below:

```
bool removeEdge(const T &sourc, const T &dest)
```

This function removes an edge originating at vertex `sourc` and ending at vertex `dest`. It returns `true` if the edge was successfully removed and `false` otherwise (edge does not exist).

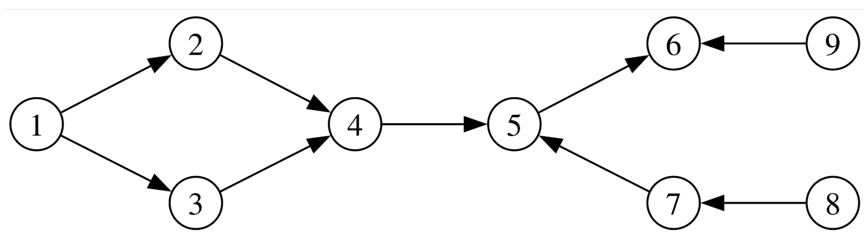
1.4) In the `Graph` class, implement the member function below:

```
bool removeVertex(const T &in)
```

This function removes the vertex with content `in`. It returns `true` if the vertex was successfully removed and `false` otherwise (the vertex does not exist). Removing a vertex implies removing all edges with origin and / or destination at that vertex.

2. Finding in and out degrees of all vertices in a graph

Consider the following directed graph:



2.1) Returning a vertex's *out-degree*. Implement the following function in the `funWithGraphs.cpp` file:

```
int FunWithGraphs::outDegree(const Graph<int> g, const int &v)
```

This function returns the *out-degree* of node `v` of graph `g`, or `-1` if the given node is not valid. Remember that the *out-degree* of a directed graph vertex reflects the total number of edges emanating from that node. It is always positive and never negative. If a directed graph's vertex does not have any edges leading to itself or other vertices, then its *out-degree* will be `0`.

<u>Suggestion</u>: look at the size of the node's adjacency list (and be careful to check that the node exists).

Execution example, for the graph depicted above:

input: v = `1`
output: *result* = `2`
The out-degree of node 1 is 2.

input: v = `4`
output: *result* = `1`
The out-degree of node 4 is 1.

input: v = `6`
output: *result* = `0`
The out-degree of node 6 is 0.

input: v = `10`
output: *result* = `-1`
Node 10 does not exist.

2.2) Returning a vertex's in-degree. Implement the following function in the `funWithGraphs.cpp` file:

```
int FunWithGraphs::inDegree(const Graph<int> g, const int &v)
```

This function returns the *in-degree* of node `v` of graph `g`, or `-1` if the given node is not valid. Remember that the *in-degree* of a vertex is defined as the number of incoming edges incident on that vertex in a directed graph. It is always positive and never negative. If a directed graph's vertex does not have any edges coming from itself or other vertices, then its *in-degree* will be `0`.

Suggestion: check the number of vertices in the graph for which the given vertex is a destination (and be careful to check that the node exists).

Execution example, for the graph depicted above:

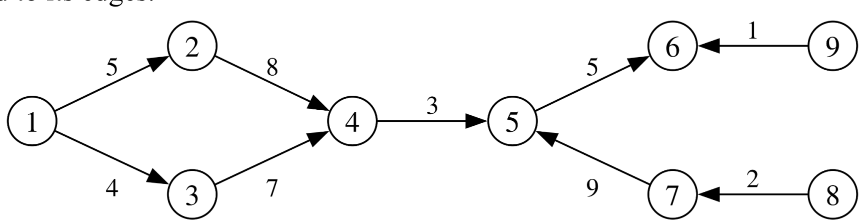
input: v = `1`
output: *result* = `0`
The in-degree of node 1 is 0.

input: v = `4`
output: *result* = `2`
The in-degree of node 4 is 2.

input: v = `7`
output: *result* = `1`
The in-degree of node 7 is 1.

input: v = `10`
output: *result* = `-1`
Node 10 does not exist.

2.3) Returning a vertex's weighted out-degree. Consider the following weighted directed graph, i.e. with weights assigned to its edges.



Implement the following function in the `funWithGraphs.cpp` file:

```
int FunWithGraphs::weightedOutDegree(const Graph<int> g, const int &v)
```

The weighted degree of a node is based on the number of edges for a node, but ponderated by the weight of each edge. This function returns the weighted out-degree of node *v*, i.e. the sum of the weights of the edges from node *v* to other nodes. If the given node is not valid, it returns `-1`.

Suggestion: go through the adjacency list of node `v` and sum the weights of the edges.

Execution example, for the graph depicted above:

input: v = `1`
output: *result* = `9`
The weighed out-degree of node 1 is 5+4 = 9.

input: v = `4`
output: *result* = `3`
The weighted out-degree of node 4 is 3.

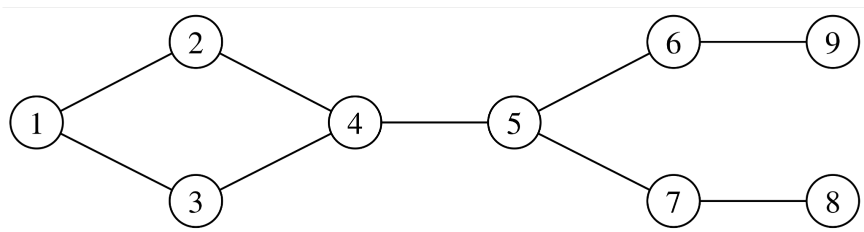
input: v = `6`
output: *result* = `0`
The weighted out-degree of node 6 is 0.

input: v = `10`
output: *result* = `-1`
Node 10 does not exist.

2.4) Vertex degree in undirected graphs.

An undirected graph is internally represented with bidirectional edges, i.e. each edge is composed of an edge from a node *v* to a node *w* and another from the node *w* to the node *v*. Hence, for undirected graphs, there is no *in-degree* and *out-degree;* there is just the *degree*: the number of edges incident to a given vertex.

As undirected graphs don't have an *in-degree* and *out-degree,* these statistics are exactly the same as *degree*. As the methods for determining those vertex degrees have already been implemented in the previous sub-exercises, check that both give the correct results for the undirected graph below (see also `tests.cpp`):



Expected results:

v = `1`; *degree* = `2`
v = `2`; *degree* = `2`
v = `3`; *degree* = `2`
v = `4`; *degree* = `3`
v = `5`; *degree* = `3`
v = `6`; *degree* = `2`
v = `7`; *degree* = `2`
v = `8`; *degree* = `1`
v = `9`; *degree* = `1`

And check that, for a graph with vertex set *V* and edge set *E,*

$$\sum_{v \in V} deg(v) = 2|E|$$