CS-472: Design Technologies for Integrated Systems Homework 0

Assigned: 22/09/2022, Due: 29/09/2022

1 Introduction

In this programming assignment, you are asked to implement a data structure for undirected graphs in `C++`. The input will be a linked list specifying which nodes are connected to which nodes. Your task is to store the graph properly and count the number of nodes of degree k for any given k. Recall that the *degree* of a node is the number of edges it is connected to.

2 Input Format

The first line of the input file contains a number n specifying the number of nodes in the graph. A node is represented with an integer i, where $0 \leq i \leq n-1$. Then, each of the remaining lines contains several integers (nodes). The second to the last nodes are all connected to the first node.

There may be some repeated edges, but you should only store it once. For example, there may be a line `1 2 3 2` and another line `2 1`, but there is only one edge between node `1` and node `2`.

Also note that there may be self-loops appearing in the input file, but they should not be added as edges. For example, an edge from node `0` to node `0` does not count towards the degree of it.

An example is given below:

```
Input:
4
1 2 2 1 3
2 1 0
```

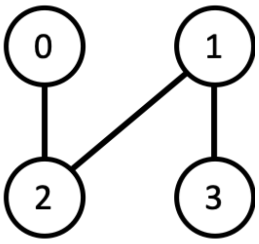

Figure 1: Example input file and the corresponding graph. The degrees of each node are: degree$(0) = 1$, degree$(1) = 2$, degree$(2) = 2$, degree$(3) = 1$.

3 Coding Interface

A template code is provided. You should implement your data structure in `graph.hpp`, where you should have a class `Graph` with the following interfaces:

- `Graph( uint32_t n )`
  The constructor. n is the number of nodes.

- `void add_edge( uint32_t from, uint32_t to )`
  This function will be called by `main.cpp` when an edge from `from` to `to` is read in. Note that the graph is undirected so the order of the two nodes is not important.

- `uint32_t num_degree_k( uint32_t k ) const`
  This function should return the number of nodes of degree k.

4 Compiling and Testing

With a command line interface, type `make` to compile the code. An executable named `hw0` will be produced if compilation is successful.

Type `./hw0 <path_to_test_file> <k>` to test the code. For example, type `./hw0 testcases/test01.txt 3` to test on the first test case with $k = 3$.

You can generate more testcases using the script in `testcases/` by typing `python gen_tests.py <testcase index> <n>` under this directory. For example, `python gen_tests.py 05 20` generates a file `test05.txt` with $n = 20$.

5 Advanced Exercise

Copy your `graph.hpp` and name it `directed_graph.hpp`. Adjust your code to store directed graphs. The interface of adding edges remains the same, but there are two additional functions required:

- `uint32_t num_in_degree_k( uint32_t k ) const`
  Returns the number of nodes with an in-degree of k. The in-degree of a node is the number of incoming edges it has.

- `uint32_t num_out_degree_k( uint32_t k ) const`
  Returns the number of nodes with an out-degree of k. The out-degree of a node is the number of outgoing edges it has.

Note that in directed graphs, an edge from node u to node v and an edge from v to u are two different edges.

To test it, type `make adv` to compile and execute by `./hw0_adv <path_to_test_file> <k>`.

6 Submission

Please submit only `graph.hpp` (and optionally `directed_graph.hpp`), compressed in one ZIP, **to Moodle**. Please do not change the names of the `hpp` files.

This homework is **NOT** graded, but you **MUST** submit a solution. We will test your code on the server where all the future (graded) homeworks will be tested and give you feedback if you did something wrong. There will be no mercy given in any case for the graded homeworks in the future. If you find this programming assignment already difficult, please consider de-registering from the course.