Professor Dr. Stephan Held
Stefan Rabenstein

Programming Exercise 2

Exercise P.2. *Task:* Implement the UNDIRECTED MINIMUM MEAN-WEIGHT CYCLE ALGORITHM from Exercise 6.2.

Usage: Your program should be named `min_mean_cycle`, and it should be called as follows:

```
min_mean_cycle input_graph output_graph
```

Input: The arguments `input_graph` and `output_graph` are mandatory, i.e. your program should exit with an error message if they are not present. Here, the file `input_graph` encodes the input graph for which your program should find a minimum mean-weight cycle, and you should write the cycle you found (and nothing else) to `output_graph` (see *Output* below).

The file `input_graph` is given in DIMACS format, which is used to encode undirected graphs as follows: All lines beginning with a `c` are comments. Now, ignoring any comment lines, to encode a graph G, the first line has the format

`p edge` n m

where $n = |V(G)|$ and $m = |E(G)|$. From this, $V(G)$ is implicitly identified with $\{1, \dots, n\}$. Note that vertex indices start with 1 in the DIMACS format. The following m lines have the format

`e` i j c

representing that $\{i, j\} \in E(G)$ with weight $c \in \mathbb{Z}$. You can assume that n, m and every edge weight c fit into an `integer` on the machine to be used for evaluation.

Output: Your program should return a minimum mean-weight cycle C in G by writing the complete DIMACS encoding (including edge weights) of the subgraph $(V(G), E(C))$ to the file `output_graph` (and nothing else). Note that the vertex set of your output graph should be $V(G)$ (not $V(C)$). In particular, your program should be able to read in `output_graph` as an input graph file again. If G is acyclic, then, as a convention, write the DIMACS encoding of the graph $(V(G), \emptyset)$ to `output_graph`.

Programming conditions: Your program should be written in C or C++, although the use of C++ is strongly encouraged. By default, your program will be compiled using clang-9.0.0using C++17. Different compilers or compiler versions are available upon request. Your program will be compiled using `-pedantic -Wall -Wextra -Werror`, i.e., all warnings are enabled and each remaining warning will lead to compilation failure. Program evaluation will be performed on Linux. The standard library as well as a one of the provided libraries for solving the MINIMUM WEIGHT PERFECT

Matching Problem (see *Help* below) can be used as you wish. No other libraries are allowed.

Submission Format: Your submission should consist of a single archive file in the .zip, .tar.gz or .tar.bz2 format, which contains all contents of your top level directory (but not the directory itself). For easier testing, your submission must contain a bash script `compile.sh` in its top level directory, which builds the executable (e.g. by directly calling the compiler or by executing some make command) when called without any arguments. Your executable must be called `min_mean_cycle` (as implied above) and be created in a subfolder called `bin` of the top level directory. Since you will (most likely) be linking against a library for solving the Minimum Weight Perfect Matching Problem, you should also provide a copy of the library located at the correct relative path with your submission.

Algorithm evaluation: The algorithm is to be implemented as described in the exercise description.

Code evaluation: Your code must implement the Undirected Minimum Mean-Weight Cycle Algorithm correctly. The correctness, readability and organization of your code will be evaluated. Make sure to add good documentation and give the variables, functions and types meaningful names that make their role clear. Break your complicated functions into small simple ones, break your program into a few units etc. Of course, your program may not trigger undefined behavior. In particular, your program must be `valgrind`-clean, i.e., must not leak memory and must not perform invalid operations on memory.

Help: The website for the exercise class contains a set of test instances for testing your code. Moreover, an updated class to store an undirected graph with edge weights, a public solver for the Minimum Weight Perfect Matching Problem (blossomV) and example code for reading in a graph and calling the solver is provided, so you can start implementing the algorithm right away. Included is also a file `README` that contains further important information.

(64 points)

Deadline: December 16th, 8:00, via email to rabenstein@or.uni-bonn.de. The websites for the lecture with all exercises and test instances can be found at:

`http://www.or.uni-bonn.de/lectures/ws19/co_exercises/exercises.html`