Professor Dr. Jens Vygen
Rudolf Scheifele

Programming Exercise 2

Exercise P.2. *Task:* Implement the Undirected Minimum Mean-Weight Cycle Algorithm from Exercise 7.3.

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`.

Your submission: Your program must be written in C or C++, although the use of C++ is strongly encouraged. Program compilation and evaluation will be

performed on Linux. Your program will be compiled with gcc-4.8.5 or gcc-7.3.0 using C++11 with the options `-pedantic -Wall -Wextra -Werror` enabled, i.e. all warnings are enabled and each remaining warning will lead to compilation failure. The default gcc version to be used is gcc-7.3.0, but you may also explicitly request gcc-4.8.5 to be used for compilation. The standard library can be used as well as a provided library for solving the Minimum Weight Perfect Matching Problem (see *Help* below). Other libraries are not allowed. Your submission must contain a bash script `compile.sh` which compiles your program (which for example may just call gcc or make directly using the mandatory compile options listed above). Since you will be linking against the 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. To achieve this, your submission should consist of a single archive file in the .zip, .tar.gz or .tar.bz2 format with the correct directory structure.

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. Running time in practice will also be evaluated, as well as the elegance, cleanness and organization of your code. 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 should not trigger undefined behavior. In particular, your program should be `valgrind`-clean, i.e. it should not leak memory and should not perform invalid operations on memory.

Help: The website for the exercise class contains a set of test instances for testing your code. Moreover, a solver for the Minimum Weight Perfect Matching Problem with example code for calling it is provided. Included is also a file `README` that contains further important information. If you wish, you can also use the graph class from Programming Exercise 1 (but you have to extend it to store edge weights).

(20 points)

Deadline: Friday, December 21, 23:59, via email to scheifele@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/ws18/coex.html`