

## Programming Exercise 1

**Exercise P.1.** *Task:* Implement the CARDINALITY MATCHING ALGORITHM as described in the lecture.

*Usage:* Your program should be called as follows:

```
program_name --graph file1.dmx [--hint file2.dmx]
```

*Input:* The first argument, `file1.dmx`, is mandatory (i.e., your program should exit with an error message if it is not present), and it encodes the graph  $G$  for which your program should find a maximum cardinality matching.

The second argument, `file2.dmx`, is optional, and when present it is expected to encode a matching  $M_0$  in  $G$ . To be precise (since DIMACS files encode graphs and not sets of edges), it encodes the subgraph  $(V(G), M_0)$  of  $G$  that corresponds to this matching. If this argument is not present,  $M_0$  is implicitly defined as the empty set. More on how  $M_0$  should be used below. Note that the square brackets [ and ] are not part of the actual program call, but just indicate that everything within is optional. For example, if `file1.dmx` and `file2.dmx` are DIMACS files, possible program calls include:

```
program_name --graph file1.dmx  
program_name --graph file1.dmx --hint file2.dmx
```

Files `file1.dmx` (and `file2.dmx` if present) are expected to be 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
```

representing that  $\{i, j\} \in E(G)$ .

*Output:* Your program should return a maximum cardinality matching  $M$  in  $G$  by writing the complete DIMACS encoding of the subgraph  $(V(G), M)$  to the standard output.

*Use of the hint matching:* The hint matching  $M_0$  should be used by your algorithm as an initial matching, so that you only need to perform  $\nu(G) - |M_0|$  augmentations to find a maximum matching. With this, you should achieve a runtime of  $O((\nu(G) - |M_0| + 1) \cdot n^2)$ .

*Use of heuristics:* As an extra speed-up, if you wish, you may use some heuristics. For example, you can start with a greedy routine to add edges to  $M_0$  until it is maximal.

*Programming language:* 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-9.0.0 using 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 can be used as you wish, but no other libraries.

*Algorithm evaluation:* You are expected to implement the algorithm as described in the lecture or in the book by Korte and Vygen. The runtime requirement will not be handled strictly, so complicated implementation details that make your program *slightly* slower may be overlooked, but the core of the algorithm must be as efficient as described in class.

*Code evaluation:* 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:* On the website for the exercise classes, which you should visit regularly, you will find a sample c++ graph class with code for parsing an input file. You may use it in your solution if you wish. You will also find some help for compiling and testing your code, including some instances to test it on.

**Please submit your programs in groups of up to 2 students.**

(64 points)

**Deadline:** November 18<sup>th</sup>, 8:00, via email to [rabenstein@or.uni-bonn.de](mailto: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](http://www.or.uni-bonn.de/lectures/ws19/co_exercises/exercises.html)