Winter term 2016/17
Prof. Dr. Stefan Hougardy
Tomás Silveira Salles

Research Institute for Discrete Mathematics
University of Bonn

Combinatorial Optimization
Exercise set 3

Exercise 3.1: Let G be an undirected, k-vertex-connected graph which has neither a perfect nor a near-perfect matching.

(i) Show that $\nu(G) \geq k$. (2 points)

(ii) Show that $\tau(G) \leq 2 \cdot \nu(G) - k$. (2 points)

Exercise 3.2: Let G be a factor-critical graph which is minimal, in the sense that for all non-empty $F \subseteq E(G)$ the subgraph $G - F$ is not factor-critical.

(i) Show that $2 \cdot |E(G)| \leq 3 \cdot (|V(G)| - 1)$. (3 points)

(ii) Show that the bound from item (i) is tight. (1 point)

Note: This is the updated version of exercise 3.2 and is slightly different from the printed version you got in class.

Exercise 3.3: Consider the Minimum Cost Edge Cover Problem: Given a graph G with weights $c : E(G) \to \mathbb{R}_{\geq 0}$, find an edge cover $F \subseteq E(G)$ that minimizes $\sum_{e \in F} c(e)$. Show that the Minimum Cost Edge Cover Problem can be linearly reduced to the Minimum Weight Perfect Matching Problem. (4 points)

Exercise 3.4: Consider the Shortest Even/Odd Path Problem: Given a graph G with weights $c : E(G) \to \mathbb{R}_{\geq 0}$ and $s, t \in V(G)$, find an s-t-path P of even/odd length in G that minimizes $\sum_{e \in E(P)} c(e)$ among all s-t-paths of even/odd length in G. Show that both the even and the odd version can be linearly reduced to the Minimum Weight Perfect Matching Problem. (4 points)

Deadline: Thursday, November 10, 2016, before the lecture.
Programming task below!

Programming task 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: Files `file1.dmx` and `file2.dmx` 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 $\{1, \dots, n\}$. The following m lines have the format

`e` i j

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

The first argument, `file1.dmx`, is mandatory (i.e., your program can expect that it will always be 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.

Output: Your program should return a maximum cardinality matching M in G by writing the 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 $O((\nu(G) - |M_0| + 1) \cdot m \log n)$.

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++ and compile with a GNU compiler on a current Linux machine. You must also provide a command with which the program can be compiled. The STL can be used as you wish, but no other libraries.

Algorithm evaluation: You are expect to implement the algorithm as described in the lecture (slightly different from the presentation in Korte-Vygen, but it would certainly be helpful to take a look at their description as well). 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 elegance, cleanness 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. Compiler warnings will cost you points.

Help: On the website for the exercise classes, which you should visit regularly, you will find a C++ unit providing a simple graph class that you can use (if you wish), as well as a precise definition of the DIMACS format and a few instances that you can use to test your program.

(64 points)

Please submit your programs in groups of 2 students.
Deadline for the programming task: Thursday, December 8, 2016, until 11:59 pm.