OPT: HW2

Optimization Methods Summer 2014/15 — Deadline: Thursday June 11 2015 8:00

General information

Your solutions should be sent to your Practical class TA, with the subject containing `OPT: HW2`. Your task consists of creating a *generator* that creates a linear/integer program and its associated *documentation*.

You can download sample inputs from the URL `http://iuuk.mff.cuni.cz/~husek/opt-ukol2.zip`.[1]

The archive contains standard input files labeled `vstup`i`-`x`.txt`, where i is the task number and x the input numbering. The running times are computed based on the standard input files. To troubleshoot your program, you can make use of the small input files in the archive, denoted with `-s`.

The rules for the first task are the same as in the first homework – your solution should be a generator of an integer or a linear program, whic you send to the TA. The generator itself should not solve the LP.

Again, you can use the `glpk` library directly in your code, in which case you have to solve the (single) LP/IP you formulate.

The second task has slightly different rules – in it, the generator may call the command `glpsol -m program.mod` (or `glpk`) and parse its output. In fact, you are allowed to create and run several linear/integer programs. Therefore, the output for the second task is created by the generator itself.

In either case, it should hold true that your solution makes use of an IP/LP in a substantial way.

Both tasks are required to contain documentation, similarly to the first homework assignment.

Contact your TA if anything is unclear.

[1] Your generator must work for other, similar inputs as well.

Task 1 – Independence in the school acquaintance graph [12 pts]

You have a large graph on input which describes acquaintances at a school. Imagine a large school with several classes, where vertices are pupils and edges correspond to who knows whom – this relation being symmetric in our case.

The school acquaiantance graph on input is not uniformly random. It always contains several large classes – groups of pupils where everyone knows each other. As in real life, there are pupils who know pupils from other classes, but those edges are relatively scarce in the school acquaintance graph.

Your task is to generate a linear/integer program that finds a *maximum independent set* – the largest group of pupils where nobody knows each other.

The important condition is that your solution should be relatively fast. For the inputs given in our archive, your IP/LP should run in the matter of *seconds*, not minutes (on a fairly recent computer).

Input format

The file with the undirected graph has the following format:

The first line starts with `GRAPH` and this keyword is followed by the number of vertices and edges, respectively. Both are separated by a whitespace. At the end of the first line there is a colon. The vertices are numbered starting with zero.

The following lines have a form i`--`j and they specify individual edges. An example (graph $K_{1,2}$):

```
GRAPH 3 2:
 0 -- 1
 0 -- 2
```

Output format

Your generator should output a linear or integer program in the language `MathProg`. The program, when run with a solver, can output any additional information, but it also must contain the following required section:

The required section starts with `#OUTPUT:` k and ends with `#OUTPUT END`, where k is the size of the detected maximum independent set. Between the two lines there should be a list of vertices in the set, each in the form `v_`i on a separate line. The number i is the index number of the right vertex.

An output example for $K_{1,2}$:

```
#OUTPUT: 2
v_1
v_2
#OUTPUT END
```

Task 2 – The cycle-free graph strikes again! [13 pts]

Similarly to the second task of HW1, your goal is to remove all directed cycles from a graph. However, this time you have no limit on their length – you have to remove every directed cycle.

Your input will be a directed graph with a weight function on the edges. Your goal is to use IP/LP to find the smallest weighted subset of edges such that after removing this subset of edges, the resulting graph contains no **directed** cycle. The input graph does not contain loops or cycles of length 2.

This task is different from the previous ones: you can run the `glpsol` command from the generator and read the output. This means that you **can call `glpsol` repeatedly**, which might be quite useful.

The running time of your generator is important here. The rough running time on the majority of the inputs should be under a minute (and many in a couple seconds, depending on the inputs). Tens of minutes is too much time.

Input format

The input file with the directed graph has the following form: The first line starts with `WEIGHTED DIGRAPH` which is followed by the number of vertices and edges, separated by spaces. At the end of the first line there is a colon. The vertices are numbered starting with 0.

The following lines are of the form i`-->`j`(` w`)` and denote the directed edges of the graph, with w being the non-negative integral weight of an edge.

An example (directed K_4):

```
WEIGHTED DIGRAPH 4 6:
 0 --> 1 (4)
 0 --> 2 (3)
 0 --> 3 (1)
 1 --> 2 (4)
 2 --> 3 (2)
 3 --> 1 (5)
```

Output format

The generator can print out additional information, but its output (either the standard output or the output file) always must contain the following mandatory part:

The mandatory part is delimited by lines `#OUTPUT:` W and `#OUTPUT END`, W is the total weight of the removed edges. Between the delimeter lines there is a list of edges i`-->`j – corresponding to those that your solution removes. An example for the above input:

```
#OUTPUT: 2
2 --> 3
#OUTPUT END
```