OPT: HW1

Optimization Methods Summer 2014/15 — Deadline: Saturday April 18 2015 at 8:00

General information

Your solutions should be sent to your Practical class TA, with the subject containing `OPT: HW1`. Your task consists of creating a *generator* that creates a linear/integer program and its associated *documentation*.

You will find a sample of the input data at the URL `http://iuuk.mff.cuni.cz/~husek/opt-ukol1.zip`. [1]

Besides the sample inputs denoted `vstup`i-x`.txt`, where i is the task number and x the id of the input, the archive also contains various small inputs denoted `-s` and some (possibly) infeasible inputs denoted `-c`.

A *generator* of an LP/IP instance is a program in any reasonable programming language, which transforms the given input into an LP/IP instance described in the language GNU MathProg, which then should be ran through the solver `glpsol`.

You will find a simple introductory guide at the URL `http://iuuk.mff.cuni.cz/~bohm/texts/mathprog_intro.html`, the official documentation can be found inside the `glpk` distribution itself or alternatively at the address `https://www3.nd.edu/~jeff/mathprog/glpk-4.47/doc/gmpl.pdf`.

Your generator can read the input from the standard input or from a file (given as a parameter of the generator). The same is true for output.

After running the `glpsol` solver, the solver should print out a specific output dependent on the exercise. See below for details. If your LP/IP has no solution, you do not need to output anything. However, any such case should be described in the attached documentation.

The generator should be submitted as a source file, not the compiled program. The following programming languages are explicitly allowed: C, C++, Java, C#, Python, Perl, Bash. If you need any other language, please check it with your TA. The source code of the generator should be human readable and commented where necessary. On the other hand, the generated LP code can be as long and unreadable as you require.

An alternate approach to constructing the LP/IP is to use the C bindings of the library `glpk` and call the library yourself. You can choose this approach if you wish.

An important and mandatory part of the homework submission is the *documentation*, which should contain:

1. Instructions on how to build the generator.
2. Instructions on how to operate the generator.
3. A brief description of the resulting LP/IP.
4. How to intepret situations when the IP/LP has no solution or has an unbounded solution.

The documentation does not need to be too long; we expect it to fit on one or two pages. The documentation should be submitted as either a PDF or as plain text (Markdown is fine).

Note: Any submitted homework solution must use a linear program in a substantial way. You can use some reasonable preprocessing of the input, but you must not solve the problem combinatorially and then print out just a trivial LP. The goal is to practice creating LP models; a more challenging task will be given in Homework 2.

[1] Your generator needs to work for other inputs as well.

Task 1: Topological sorting [10 points]

You get an unweighted directed graph on input. Your task is to formulate an LP program that attempts to find a (partial) ordering of the vertices such that the following is satisfied:

For every edge uv it holds that the vertex u is ordered before vertex v.

Your ordering should be represented as non-negative integers starting with 0. The expected running time of your solver on the test data should be around 10 seconds.

Input format

The input file has the following syntax:

The first line starts with the word `DIGRAPH` which is followed by two integers representing the number of vertices and number of edges, respectively. The three items on the first line are separated by a single space each.

The vertices are numbered starting with 0. Any other line describes a single edge, containing i`-->`j which says that the edge goes from i to j. An example of the graph $K_{1,2}$:

```
DIGRAPH 3 2
 0 --> 1
 0 --> 2
```

Output format

The solver may output any debug information that you need, but it must contain at some point the following mandatory part:

The mandatory part starts with `#OUTPUT:` and ends with `#OUTPUT END`. Between the two lines there is a sequence of vertex orders of the form `v_`i`:` x, where i is an id of the vertex and x is its position in the order. The position in the order must be a non-negative integer. A sample output for the graph $K_{1,2}$ follows:

```
#OUTPUT:
v_0: 0
v_1: 1
v_2: 1
#OUTPUT END
```

Task 2: Removing short cycles [15 points]

You start with an input digraph with a weight function on the edges.

Your task is to write an integer progrm which finds the minimum weighted subset of edges R such that removing all the edges of R from the graph disconnects every **directed cycle** of length 4 or less. We do not care about larger cycles and we do not care about incorrectly-directed cycles.

The input graph contains no loops. The expected running time of the solver is between 5 and 120 seconds, depending on the input.

Input format

The input file containing the digraph has the following format: The first line starts with the word `WEIGHTED DIGRAPH`, which is followed by a number of vertices and the number of edges, both separated by a single space.

The vertices are denoted by non-negative integers, starting with 0. Further lines are of the form i`-->`j `(`w`)` and describe the edges of the digraph, including w, the non-negative integral weight of an edge.

An sample input containing 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 solver may output any debug information that you need, but it must contain at some point the following mandatory part:

The mandatory part starts with the line `#OUTPUT:` W and ends with the line `#OUTPUT END`, W is the total weight of removed edges. Between the two defining lines, there should be a list of edges that are removed, one edge per line. The format for a line is again i`-->`j. An example for the input above:

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