Combinatorial Problem Solving (CPS)

Laboratory. Constraint Programming. Graph Coloring.

In this session we will consider the *Graph Coloring Problem*: given an (undirected) graph $G = (V, E)$, to find the minimum number k of colors for painting the vertices so that adjacent vertices have different colors.

1. Write a program `p` with Gecode that, given a graph coloring instance, outputs the minimum number of colors and an optimal assignment of colors to vertices.

   Use "first unassigned" for variable selection strategy, and "smallest value" for value selection.

   The input file with the graph coloring instance consists of a first line with n, the number of vertices, and m, the number of edges, separated by a blank space. Then m lines follow, each of which with two numbers u and v separated by a blank space, which represent the edge $u \leftrightarrow v$.

   Assume that all numbers are integers, and in particular, that $1 \leq n \leq 200$. Assume also that vertices are numbered from 1 to n. Moreover, for every pair of vertices u and v there is at most one arc of the kind $u \leftrightarrow v$, and there are no self-edges.

   The output should contain a first line with the minimum number of colors. Then n lines should follow, each of which with two numbers u ($1 \leq u \leq n$) and c ($1 \leq c \leq k$) separated by a blank space, which represent that the color of vertex u is the c-th color. The order of the vertices is irrelevant.

   For example:

```
$ cat sample.col.pb
5 5
1 2
1 4
2 3
3 5
4 5
$ p sample.col.pb
3
1 1
2 2
3 1
4 2
5 3
```

   Run your program on the set of benchmarks that can be found at the website of CPS (`http://www.cs.upc.edu/~erodri/cps.html`) with a time limit of 15 seconds. Which ratio of the benchmarks can you solve with your program within the given time limit? How much time do the solved instances take? Use the script below (which is also available at the website of CPS).

```
#!/bin/bash

TIME_LIMIT=15 # In secs.
BENCH_DIR=instances
EXE=p

######## DO NOT CHANGE BELOW THIS LINE ########

TIMED_OUT=124
```

```
n_solved=0
n_all_pb=0
total_time=0
for ifile in $BENCH_DIR/*.pb; do

    /usr/bin/time --format "%e\n%x" --output=out.txt  \
                  timeout $TIME_LIMIT                  \
                  $EXE $ifile &> /dev/null
    time=$(head -n 1 out.txt)
    code=$(tail -n 1 out.txt)
    if [ "$code" = $TIMED_OUT ]; then
        echo "Timed␣out:␣$ifile"
    else
        echo "OK␣␣␣␣␣␣␣␣:␣$ifile"
        n_solved=$(( n_solved+1 ))
        total_time=$(echo "$total_time␣+␣$time" | bc)
    fi
    n_all_pb=$(( n_all_pb+1 ))
done
echo "Time␣solved␣problems:␣$total_time"
echo "Num.␣solved␣problems:␣$n_solved"
echo "Num.␣␣all␣␣␣problems:␣$n_all_pb"
perc=$(echo "scale=2;␣(100*$n_solved)/$n_all_pb" | bc )
echo "Percentage:␣$perc␣%"
rm --force out.txt
```

2. (*Breaking symmetries*) Note that there is the following symmetry: given a solution to a graph coloring problem, another solution can be obtained by choosing two colors a and b and swapping them, so that all vertices that were painted with a are now painted with b, and vice versa. Can you exploit this observation so as to make your program more efficient? Repeat the execution over the benchmark suite. Which ratio of the benchmarks can you solve now? How much time do the solved instances take?

   Use the same variable and value selection strategies as in the previous exercise.

3. (*Variable selection*) Use *Brelaz's heuristic* for graph-coloring: select the variable with the smallest domain, and in case of tie choose the one with the largest degree first.

   Repeat the execution over the benchmark suite. Which ratio of the benchmarks can you solve now? How much time do the solved instances take?