Extra Credit Assignment: Graph Coloring

- Input: An undirected graph $G = (N, E)$.
- Problem: Assign a color c_i to each node $i \in N$ such that
  1. $c_i \neq c_j$ for all $(i, j) \in E$
  2. The number of colors used is minimized.

Integer Programming Formulation:

$$
\begin{aligned}
\min \quad & \sum_{j \in C} y_j \\
\text{subject to} \quad & \sum_{j \in C} x_{ij} = 1 \qquad \forall i \in N, \\
& \sum_{i \in N} x_{ij} \le |C| y_j \qquad \forall j \in C, \\
& x_{ik} + x_{jk} \le 1 \qquad (i,j) \in E, k \in C, \\
& y_{i+1} \le y_i \qquad \forall i \in \{1, 2, \ldots, |C|-1\}, \\
& x_{ij} \in \{0,1\} \qquad (i,j) \in E, \\
& y_j \in \{0,1\} \qquad j \in C,
\end{aligned}
$$

where C is the set of colors.

```
# color.txt
# Solve an instance of the graph coloring problem
# with integer programming.

model graph_coloring_model.txt;
data  g1.txt;
#option solver cplex;
option cplex_options 'timing=1';
solve;

# list the color for each node
printf "node\tcolor\n";
for {u in NODES} {
  for {i in COLORS: x[u,i] == 1}
    printf "%d\t%d\n",u,i;
}
```

```
ampl < color.txt
Times (seconds):
Input =  0.026352
Solve =  0.423584
Output = 0.00488
CPLEX 8.0.0: optimal integer solution; objective 4
node    color
1       3
2       4
3       2
4       3
5       1
6       4
7       2
8       3
9       4
10      1
```

```
# This is a simple graph-coloring heuristic that colors
# the nodes in the order that AMPL stores them
# in the data file.

set NODES;
set EDGES within {NODES, NODES};

data g1.txt;

# In the worst case we need
# to use a different color for each node.
set COLORS := {1 .. card(NODES)};

# Data structures to keep track of the
# nodes that have been colored.
set COLORED within NODES ordered default {};
set UNCOLORED within NODES ordered default NODES;
```

```
# The next node to color (i.e., the first node in the
# UNCOLORED set.
param next_node;

# The set of colors allowable for next_node.
set POSSIBLE_COLORS ordered;

# node_color[i] indicates the color assigned to node i.
param node_color {NODES} default 0;

#parameters to store the time when the
#algorithm starts and stops
param start_time;
param stop_time;
```

```
# Start the clock.
let start_time := _ampl_elapsed_time;
repeat {
  # Select the first node in the uncolored list.
  let next_node := first(UNCOLORED);
  # Check the color assignments of next_node's neighbors.
  let POSSIBLE_COLORS := COLORS;
  for {u in NODES: (u, next_node) in EDGES or
   (next_node, u) in EDGES} {
     let POSSIBLE_COLORS :=
     POSSIBLE_COLORS diff {node_color[u]};
  }
  let node_color[next_node] := first(POSSIBLE_COLORS);
  let COLORED := COLORED union {next_node};
  let UNCOLORED := UNCOLORED diff {next_node};
} until card(UNCOLORED) == 0;
let stop_time := _ampl_elapsed_time;
```

```
# Print out the solution.
printf "%d colors were used:\n",
card( union {i in NODES}{node_color[i]});
display union {i in NODES} {node_color[i]};
printf "cpu seconds = %f\n",stop_time - start_time;
display node_color;
printf "\n\nVerification:\n";
for {(i,j) in EDGES} {
  printf "Edge (%d,%d):\tnode %d gets color
  %d\tnode %d gets color %d\n"
  ,i,j,i,node_color[i],j,node_color[j];
}
```

```
ampl < colorbynumbers.txt

5 colors were used:
set  union {i in NODES}  {node_color[i]} := 1 2 3 4 5;
cpu seconds = 0.000000
node_color [*] :=
 1  1
 2  2
 3  3
 4  1
 5  2
 6  4
 7  3
 8  1
 9  3
10  5
;
```