Exercício 1 (Investimento, Formulação, fácil)
Sejam u_i as unidades produzidas do produto $i \in P = [1,4]$, $x_i \in \mathbb{B}$ variáveis booleanas que indicam se o produto i é produzido, c_i e l_i o custo inicial e os lucros, respectivamente.

$$
\begin{array}{rlll}
\textbf{max} & \sum_{i\in P} u_i l_i - \sum_{i\in P} c_i x_i & & \\
\textbf{s.a} & \sum_{i\in P} x_i \leq 2 & & \text{No máximo dois tipos de produtos} \\
& x_3 \leq x_1 + x_2 & & \text{Tipo 3 somente so tipo 1 ou 2} \\
& x_4 \leq x_1 + x_2 & & \text{Tipo 4 somente so tipo 1 ou 2} \\
& u_i \leq 2000 & \forall i \in P & \text{Limite produção (redundante)} \\
& u_i \leq 2000 x_i & \forall i \in P & \text{Vínculo das variáveis}
\end{array}
$$

Alternativas para a segunda restrição

$$
\begin{array}{ll}
x_3 + x_4 \leq 2(x_1 + x_2) & \text{Mais fraco, a soma das duas restrições acima} \\
x_3 + x_4 \leq x_1 + x_2 & \text{Mais forte, válido porque no máximo dois projetos}
\end{array}
$$

Exercício 2 (Coloração de grafos)
Para um grafo $G = (V, A)$ com n vértices e $V = [n]$ seja $x_{ij} \in \mathbb{B}$ para $1 \leq v, j \leq n$ um indicador se o vértice v possui cor $i \in C$, com $C = [n]$ o conjunto de cores permitidos. (Permitimos até n cores, para garantir uma coloração.) Seja ainda $c_i \in \mathbb{B}$ para $i \in C$ uma variável auxiliar que indice se a cor i está usada.

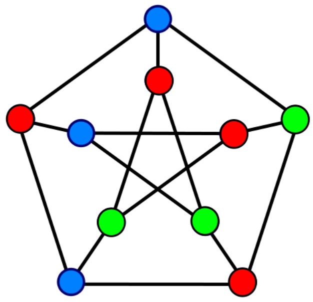
$$
\begin{array}{rlll}
\textbf{minimiza} & \sum_{i\in C} c_i & & \text{menor número de cores} \\
\textbf{sujeito a} & \sum_{1\leq j\leq n} x_{vj} = 1 & v \in V & \text{Garantir exatamente um cor por vértice} \\
& x_{ui} + x_{vi} \leq 1 & i \in C, uv \in E & \text{Coloração viável} \\
& n c_i \geq \sum_{v\in V} x_{vi} & i \in C & \text{Define variáveis aux.}
\end{array}
$$

A implementação correspondente em AMPL para o caso do grafo de Peterson é

```
set V;                                # set of vertices
set E within V cross V;               # set of edges
var c { i in V } binary;              # color i set?
var x { i in V, j in V } binary;      # vertex i has color j?

minimize Colors:
        sum { i in V } c[i];

subject to OneColor { i in V }:
        sum { j in V } x[i,j] = 1;

subject to NoAdjacent { (i,j) in E, k in V }:
        x[i,k] + x[j,k] <= 1;

subject to DefineColor { i in V }:
        10*c[i] >= sum { j in V } x[j,i];
```

```
data;

set V := 1 2 3 4 5 6 7 8 9 10;
set E := (1,2) (1,3) (1,6) (2,7) (2,8) (3,4) (3,9) (6,5) (6,10) (4,5)
    (4,8) (5,7) (7,9) (8,10) (9,10) ;
# for use with cm
set A := (1,2) (1,3) (1,6) (2,7) (2,8) (3,4) (3,9) (5,6) (6,10) (4,5)
    (4,8) (5,7) (7,9) (8,10) (9,10) ;

end;
```

Solução:



Exercício 3 (Sudoku)
O seguinte modelo em AMPL formaliza as restrições:

```
set digitos := 1 .. 9;
set linhas  := 1 .. 9;
set colunas := 1 .. 9;

var numero { linhas, colunas, digitos } binary;

maximize SomaDiagonalSuperior:
        sum { i in linhas, d in digitos } d*numero[i,i,d];

# cada quadro contém exatamente um digito
subject to QuadroMenor { i in linhas, j in colunas }:
        sum { d in digitos } numero[i,j,d] = 1;
# cada linha contém de cada dígito exatamente uma vez
subject to Linha { i in linhas, d in digitos }:
        sum { j in colunas } numero[i,j,d] = 1;
# cada coluna contém de cada dígito exatamente uma vez
subject to Coluna { j in colunas, d in digitos }:
        sum { i in linhas } numero[i,j,d] = 1;
# cada quadro maior contém cada dígito exatamente uma vez
subject to QuadroMaior { i in {1,4,7}, j in {1,4,7}, d in digitos }:
        sum { di in 0..2, dj in 0..2 } numero[i+di,j+dj,d] = 1;
```

Um exemplo de uma solucão é

8	3	1	4	2	5	8	9	6
4	8	2	6	9	1	3	5	7
5	6	9	3	7	8	2	1	4
6	2	5	7	1	3	9	4	8
1	9	3	2	8	4	6	7	5
8	4	7	5	6	9	1	2	3
9	5	4	8	3	2	7	6	1
3	1	6	9	5	7	4	8	2
2	7	8	1	4	6	4	3	9

Exercício 4 (Caixeiro viajante)
A implementação do modelo segue a notas de aula.

```
param cities;
set City := 1..cities;
param dist {City,City};
set arcs := City cross City;
var route {arcs} binary;

minimize distance:
        sum {(i,j) in arcs} route[i,j]*dist[i,j];
subject to onein {i in City}:
        sum { j in City } route[j,i] = 1;
subject to oneout {i in City}:
        sum { j in City } route[i,j] = 1;
subject to nosubtour1 { i in City }:
        route[i,i] = 0;
subject to nosubtour2 { i in City, j in City}:
        route[i,j] + route[j,i] <= 1;
subject to nosubtour3 { i in City, j in City, k in City}:
        route[i,j] + route[j,i] + route[i,k] + route[k,i]
          + route[j,k]+ route[k,j] <= 2;
end;
```

Solução: 16565 com nosubtour1 (0 s), 21073 com nosubtour2 (0 s), e 21539 com nosubtour3 (707 s). A solução exata é 25395.

Exercício 5 (Ponder this, Formulação, difícil)
Uma formulação em AMPL é

```
#
# IBM "Ponder this", 12/2012
# http://domino.research.ibm.com/Comm/wwwr_ponder.nsf/Challenges/
    December2012.html
#
# For n=6 the optimal value is 93 (found in 25s, proven in 75s on my Core
    2 Q6600 @ 2.4 GHz)
#   for an average of 2.58 per cell, and CPLEX found 7 solutions.
#
# 1 3 2 3 1 3
```

```
# 2 5 1 5 4 2
# 3 4 1 2 3 1
# 1 2 3 5 4 1
# 3 4 5 1 2 3
# 2 1 2 4 3 1
#
# For n=7 the optimal value is in [128,143] (128 found in 800s), CPLEX
    uses >= 2GB.
#
param n := 7;
set digits  := 1 .. 5;
set lines   := 1 .. n;
set columns := 1 .. n;

set Cells   := lines cross columns;
set neighbors { i in lines, j in columns } := { (i-1,j), (i+1,j), (i,j-1),
     (i,j+1) } inter Cells;

var number { lines, columns, digits } binary;

maximize totalSum:
        sum { i in lines, j in columns, d in digits } d*number[i,j,d];

# every cell contains exactly one digit
subject to cellsOccupied { i in lines, j in columns }:
        sum { d in digits } number[i,j,d] = 1;
# every cell has neighbors with the smaller numbers
subject to hasNeighbors { i in lines, j in columns, d in digits,
                          d0 in digits : d0 < d }:
        number[i,j,d] <= sum { (i0,j0) in neighbors[i,j]} number[i0,j0,d0
            ];
```