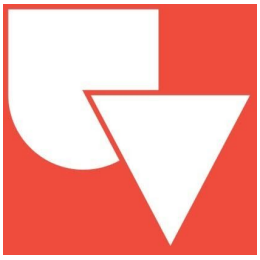


UNIVERSIDAD DEL VALLE

TALLER 1

PROGRAMACIÓN CON RESTRICCIONES

CARLOS ESTEBAN MURILLO S.
1526857-3743

PRESENTADO A:

ROBINSON DUQUE

Santiago de Cali, Febrero de 2020
Escuela de Ingeniería en Sistemas
Facultad de Ingeniería

PARTE 1 CSPs

1. SUDOKU

Análisis

Para resolver el problema del Sudoku clásico se establece como parámetro de entrada una matriz 9 x 9 de enteros que representa el estado inicial del tablero. los valores que se tomen entre 1 y 9 son valores iniciales del problema, y los valores en 0 representan los espacios en blanco.

$$initial = [\,|\,x_{1,1},...,x_{1,9}\,|\,...\,|\,x_{9,1},...,x_{9,9}\,|\,],\ donde\ x_{i,j} \in \{0,...,9\}$$

Luego se declara la matriz resultado de dimensión 9 x 9, compuesta por variables de decisión enteras que toman valores del conjunto {1,...,9}, necesarios para resolver el juego.

$$matrix = [\,|\,x_{1,1},...,x_{1,9}\,|\,...\,|\,x_{9,1},...,x_{9,9}\,|\,],\ donde\ x_{i,j} \in \{1,...,9\}$$

Una vez se tienen ambas matrices, la de entrada y en la que se almacenará la solución, se procede a establecer las restricciones. La primer restricción busca garantizar que se respeten los valores iniciales de la matriz de entrada en la matriz solución.

$$\forall_{i,j \in \{1,...,9\}}(\ matrix_{i,j} = initial_{i,j}\ )\ donde\ initial_{i,j} > 0$$

Luego se busca que para cada fila y columna de la matriz solución se encuentren todos los números del 1 hasta el 9 sin repetición. Para esto se hace uso de la restricción alldifferent, que garantiza al recibir un arreglo de elementos, que cada uno sea distinto.

$$\forall_{i \in \{1,...,9\}}(\ alldifferent(\ [\ \forall_{j \in \{1,...,9\}} matrix_{i,j}\ ]\ ))$$

$$\forall_{j \in \{1,...,9\}}(\ alldifferent(\ [\ \forall_{i \in \{1,...,9\}} matrix_{i,j}\ ]))$$

Finalmente se busca que por cada sub-cuadrado 3 x 3 de la matriz contenga los números del 1 hasta el 9 sin repetición.

$$\forall_{i,j \in \{1,...,3\}}(\ alldifferent(\ [\ \forall_{p,q \in \{1,...,3\}}\ matrix_{(i-1)*3+p,\,(j-1)*3+q}\ ]\ )))$$

Implementación

Para la implementación se utilizó el lenguaje de modelado MiniZinc de la siguiente manera:

Se declaran las constantes iniciales de las dimensiones del tablero general y de los subtableros, después se declaran los conjuntos correspondientes al valor de sus variables.
Se declara la matriz con el problema inicial y la matriz solución con sus respectivos tamaños y dominios.

```
%Constants
int: sudoDim = 9;
int: subDim = 3;

%sets
set of int: size = 1..sudoDim;
set of int: subSize = 1..subDim;

%Matrix of decision variables
array[size, size] of var 1..sudoDim: matrix;

%Matrix problem
array[size, size] of 0..sudoDim: initialMatrix;
```

Luego se expresan las restricciones anteriormente propuestas para restringir el dominio de las variables de la matriz, con el fin de encontrar una solución al juego. Finalmente se propone que resuelva a satisfacción y que represente la salida como una matriz con la solución del juego.

```
%Constraints
constraint forall(i in size, j in size )(
 if initialMatrix[i, j] > 0 then matrix[i, j] = initialMatrix[i, j]
endif
);

constraint forall(i in size)(
 alldifferent( [ matrix[i, j] | j in size ])
);

constraint forall(j in size)(
 alldifferent( [ matrix[i, j] | i in size ])
);

constraint forall(i, j in subSize)(
 alldifferent([ matrix[(i-1)*subDim + i1, (j-1)*subDim + j1] | i1, j1
in subSize ])
);
```

```
solve satisfy;

%Output
output[if j != sudoDim then "\(matrix[i,j]) " else "\(matrix[i,j])\n"
endif | i,j in size];
```

Una entrada válida para este problema es:

```
initialMatrix = [|0, 9, 0, 0, 0, 0, 0, 1, 0
                 |8, 0, 4, 0, 2, 0, 3, 0, 7
                 |0, 6, 0, 9, 0, 7, 0, 2, 0
                 |0, 0, 5, 0, 3, 0, 1, 0, 0
                 |0, 7, 0, 5, 0, 1, 0, 3, 0
                 |0, 0, 3, 0, 9, 0, 8, 0, 0
                 |0, 2, 0, 8, 0, 5, 0, 6, 0
                 |1, 0, 7, 0, 6, 0, 4, 0, 9
                 |0, 3, 0, 0, 0, 0, 0, 8, 0|]
```

2. KAKURO

Análisis

Para modelar este problema es necesario tener claro las entradas o parámetros con los que se va a trabajar para que este se pueda resolver correctamente. Considere las siguientes imágenes:

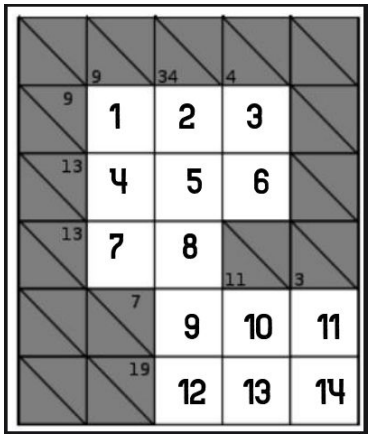


Imagen 2.1

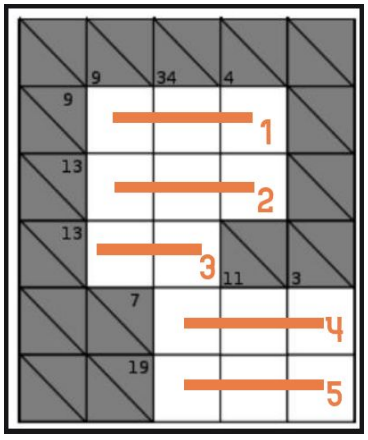


Imagen 2.2

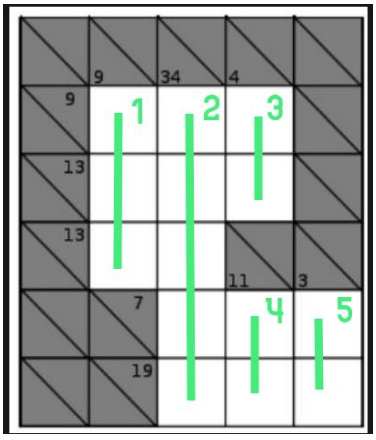


Imagen 2.3

En la primer imagen se establece un orden y la cantidad de casillas blancas del tablero, las que finalmente serán calculadas para dar una solución al juego, es este entonces el primer parámetro que se debe ingresar al modelo, el número de casillas disponibles (*whites*) que en este caso es 14.

Después como se puede ver en las imagenes 2.2 y 2.3 se establece un orden y numeración de las filas horizontales y verticales. En ambos casos se debe considerar el orden de izquierda a derecha y de arriba hacia abajo. El segundo parámetro entonces sería la suma de filas y columnas disponibles en nuestro mapa ($size$), en nuestro caso 10.

El tercer parámetro se reconoce como la longitud de la fila o columna más larga del tablero ($large$), en este caso sería la segunda columna de la imagen 2.3, con un valor de 5.

Hasta ahora se tiene entonces que: $(whites,\ size,\ large)\ \in\ N$

La siguiente entrada corresponde a una matriz $size\,x\,large$ que representa las casillas blancas involucradas en cada condición del juego, bien sea fila o columna. Si existe una fila o columna que no sea de longitud $large$, se rellena el excedente con ceros.

$$pro\ = [\ |\ x_{1,\,1}, ..., x_{1,\,large}\ |\ ...\ |\ x_{size,1}, ..., x_{size,large}\ |\ ],\ donde\, x_{i,j}\ \in\ \{0, ..., n\},\ n\ \in\ N$$

Para el ejemplo anterior la entrada sería considerando primero las filas y luego las columnas:

1	2	3	0	0
4	5	6	0	0
7	8	0	0	0
9	10	11	0	0
12	13	14	0	0
1	4	7	0	0
2	5	8	9	12
3	6	0	0	0
10	13	0	0	0
11	14	0	0	0

La última entrada seria una matriz de tamaño $size$, la cual representará el resultado i que se debe obtener al sumar los elementos de la fila i de la matriz pro.

$$res\ = [\ x_1, ..., x_{size}\ ],\ donde\, x_i\ \in\ \{1, ..., n\},\ n\ \in\ N$$

Para el ejemplo que se está trabajando el arreglo se veria asi:

9	13	13	7	19	9	34	4	11	3

Ahora se procede a declarar el arreglo que guardara la solución para cada una de las $whites$ variables.

$$W = [x_i, ..., x_{whites}]\, x_i \in \{1, ..., 9\}$$

Las restricciones que se consideran son:

1. $\forall_{i \in \{1,...,size\}}(alldifferent([\ \forall_{j \in \{1,...,size\} \wedge pro_{i,j} \neq 0} W_{pro_{i,j}}]\ ))$
2. $\forall_{i \in \{1,...,size\}}(\sum_{\forall_{j \in \{1,...,large\} \wedge pro_{i,j} \neq 0}} W_{pro_{i,j}} = res_i)$

De la primer restricción se garantiza que cada fila y columna tenga valores distintos sin repetición. La Segunda garantiza que al sumar los elementos de cada fila o columna del tablero original de como resultado el valor esperado correspondiente.

Implementación

Se declaran los parámetros antes descritos con sus respectivos dominios.

```
%Parameters
int: whites;
int: size;
int: large;

%Array Parameters
array[1..size, 1..large] of int: pro;

array[1..size] of int: res;

%Array of decision variables
array[1..whites] of var 1..9: w;
```

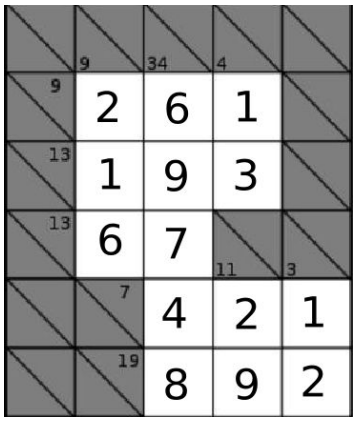
Se declaran las restricciones correspondientes con ligeras variaciones en la sintaxis para mayor legibilidad. Finalmente se imprime el resultado del arreglo.

```
%Constraints
constraint forall(i in 1..size)(
```

```
 alldifferent( [ w[pro[i, j]] | j in 1..large where pro[i, j] != 0] )
 /\
 sum( [ w[pro[i, j]] | j in 1..large where pro[i, j] != 0] ) == res[i]
 );

solve satisfy;

%Output
output[show(w)];
```

Salida para el problema planteado:

```
[2, 6, 1, 1, 9, 3, 6, 7, 4, 2, 1, 8, 9, 2]
```



3. SECUENCIA MÁGICA

Análisis

Como parámetro inicial $size$ indicará el tamaño correspondiente de la secuencia. seq será el arreglo de variables de decisión donde se almacenará la solución correspondiente:

$$seq = [\, x_0, ..., x_{size-1} \,] \; donde \; x_i \in \{0, .., size - 1\}$$

La solución general del problema hace uso de una restricción global conocida como $among(x, y, z)$ la cual permite que exactamente x variables del arreglo y obtenga algún valor del conjunto z.

Implementación

Primero se presenta una versión sin considerar las restricciones redundantes, en la cual se pide como entrada el tamaño de la secuencia que se quiere generar ($size$), se genera un conjunto con los valores entre 0 y el parámetro ($range$), luego se crea un arreglo de variables de decisión indexado y con dominios según $range$.
Posteriormente se establece una restricción para cada elemento del arreglo en la cual se busca que cada uno aparezca en la secuencia i veces segun su ubicacion.

```
%Parameters
int: size;

% sets
set of int: range = 0..(size - 1);

%Array of decision variables
array[range] of var range: seq;

%Constraints
constraint forall(i in range)( among(seq[i], seq, i..i) );

% Output
output[show(seq)];
```

La segunda versión es igual solo que se adiciona un par de restricciones que resultan más eficientes aunque sean redundantes que la versión anterior, ya que restringen de forma más directa el dominio de las variables, basándose en propiedades de la indexación presente en la secuencia.

```
%Parameters
int: size;

% sets
set of int: range = 0..(size - 1);

%Array of decision variables
array[range] of var range: seq;

%Constraints
constraint sum(i in range)( seq[i] * (i - 1) ) == 0;
constraint sum(seq) == size;
constraint forall(i in range)( among(seq[i], seq, i..i) );

solve satisfy;

% Output
```

```
output[show(seq)];
```

4. ACERTIJO LÓGICO

Análisis

Para este punto se decidió asignar un número entre 1 a 3 sobre cada característica de las personas, con el fin de mostrar el resultado asociando las características que tienen valor en común. Las variables son:

$\{Juan, Oscar, Dario\} \in \{1..3\}$
$\{Gonzalez, Garcia, Lopez\} \in \{1..3\}$
$\{clásica, pop, jazz\} \in \{1..3\}$
$\{vCuatro, vCinco, vSesis\} \in \{1..3\}$

Primero se debe considerar que por cada característica las variables deben tomar valores diferentes, para luego presentar las siguientes restricciones:

1. $Juan \neq Gonzalez \wedge Juan \neq vCuatro \wedge Gonzalez \neq vSeis \wedge Gonzalez = clasica$
2. $pop \neq Garcia \wedge pop \neq vCuatro$
3. $Oscar \neq Lopez \wedge Oscar \neq vCinco$
4. $Dario \neq jazz$

Implementación

```
%Decision Variables
var 1..3: Juan;
var 1..3: Oscar;
var 1..3: Dario;

var 1..3: Gonzalez;
var 1..3: Garcia;
var 1..3: Lopez;

var 1..3: clasica;
var 1..3: pop;
var 1..3: jazz;

var 1..3: vCuatro;
var 1..3: vCinco;
var 1..3: vSeis;
```

```
% Constraints
constraint all_different([Juan, Oscar, Dario]);
constraint all_different([Gonzalez, Garcia, Lopez]);
constraint all_different([clasica, pop, jazz]);
constraint all_different([vCuatro, vCinco, vSeis]);

constraint Juan != Gonzalez /\ Juan != vCuatro /\ Gonzalez != vSeis /\
Gonzalez == clasica;
constraint pop != Garcia /\ pop != vCuatro;
constraint Oscar != Lopez /\ Oscar == vCinco;
constraint Dario != jazz;

%Solving symmetries
constraint increasing([Juan, Oscar, Dario]);

solve satisfy;

%Output
output[
"Names     : ", show([Juan, Oscar, Dario]), "\n",
"Lastnames : ", show([Gonzalez, Garcia, Lopez]), "\n",
"Music     : ", show([clasica, pop, jazz]), "\n",
"Age       : ", show([vCuatro, vCinco, vSeis]), "\n"];
```

Solución:

```
Names :     [1, 2, 3]
Lastnames : [3, 2, 1]
Music :     [3, 1, 2]
Age :       [3, 2, 1]
```

La interpretación de la solución es:

Juan Lopez escucha música pop y tiene 26 años.
Oscar Garcia escucha música Jazz y tiene 25 años.
Dario Gonzalez escucha música clásica y tiene 24 años.

5. UBICACIÓN DE PERSONAS EN UNA REUNIÓN

Análisis

Los parámetros correspondientes para este punto son, el arreglo $persons$ y las matrices $next,\ separate\ y\ distance$.

$persons\ =\ ["p_1",\ "p_2",\ ...,\ "p_n"]$

$next\ =\ [|p_{i,1},\ p_{j,1}|...|p_{i,k},\ p_{j,k}|]\ donde\ p_{i,1}...\ p_{j,1}, p_{i,k}...\ p_{j,k}\ \in \{1, ..., n\}$

$separate\ =\ [|p_{x,1},\ p_{y,1}|...|p_{x,s},\ p_{y,s}|]\ donde\ p_{x,1}...\ p_{x,s}, p_{y,1}...\ p_{y,s}\ \in \{1, ..., n\}$

$distance\ =\ [|p_{a,1},\ p_{b,1}, p_1\ |...|p_{a,d},\ p_{b,d}\ , p_c|]\ donde\ p_{a,1}...\ p_{a,d}, p_{b,1}...\ p_{b,d}\ \in \{1, ..., n\}$
$y\ c_1...\ c_d\ \in \{1, ..., n-2\}$

Sea $nnext,\ nsep,\ ndis$ el número de filas correspondiente a cada tipo de matriz y l la longitud del arreglo $persons$. Se procede a declarar el arreglo de variables per donde se almacenará la solución.

$$per\ =\ [x_1,\ x_2,\ ...,\ x_n]\ donde\ x_i \in N$$

A continuación se declaran las restricciones correspondientes al efecto que produce cada una de las matrices de entrada:

1. $alldifferent(\ per\ )$

2. $\forall_{\ i\ \in\ \{1,..,\ ndis\}}\ distance_{i,\ 3} \geq 1 \wedge distance_{i,\ 3} \leq l - 2$

3. $\forall_{p\ \in\ \{1,..,\ nnext\}}(per_1 = next_{p,1} \wedge per_2 = next_{p,2}\ \vee$
   $per_l = next_{p,1} \wedge per_{l-1} = next_{p,2} \vee$
   $\exists_{\ i\ \in\ \{2,..,l-1\}} per_i = next_{i,1} \wedge (per_{i+1} = next_{p,2} \vee per_{i-1} = next_{p,2}))$

4. $\forall_{p\ \in\ \{1,..,\ nnext\}}(per_1 = separate_{p,1} \wedge per_2 \neq separate_{p,2}\ \vee$
   $per_l = separate_{p,1} \wedge per_{l-1} \neq separate_{p,2} \vee$
   $\exists_{\ i\ \in\ \{2,..,l-1\}} per_i = separate_{i,1} \wedge (per_{i+1} \neq separate_{p,2} \vee per_{i-1} \neq separate_{p,2}))$

5. $\forall_{p\ \in\ \{1,..,\ ndis\}}\ \exists_{\ i,j\ \in\ \{1,..,l\}} per_i = distance_{p,1} \wedge per_j = distance_{j,2} \wedge$
   $abs(i-j) - 1 \leq distance_{p,3} \wedge abs(i-j) - 1 \geq 1$

De la primer restricción se debe satisfacer que todos son diferentes. De la segunda, que la tercer columna de cada fila de la matriz *distance* este en el rango que se necesita. La tercer restricción busca que cada fila de *next* se cumpla, para eso inicialmente y con el fin de evitar desbordamientos, se validan las esquinas del arreglo para luego generalizar la comparación de los elementos que deben estar unos con otros dentro del arreglo.
El mismo procedimiento se repite para la matriz *separate* pero esta vez validando que los elementos no estén juntos.
Finalmente se busca que cada fila de la matriz *distance* se cumpla verificando que la distancia entre los elementos esté en el rango deseado.

Implementación

```
array[int] of string: persons;
array[int, 1..2] of int: next; array[int, 1..2] of int: separate;
array[int, 1..3] of int: distance;

% Constants
int: nnext = floor(length(next) / 2);
int: nsep = floor(length(separate) / 2);
int: ndis = floor(length(distance) / 3);
int: l = length(persons);

%Array of decision variables
array[1..l] of var 1..l: per;

% Constraints
constraint alldifferent(per);

constraint forall(i in 1..ndis)(
 distance[i, 3] >= 1 /\ distance[i, 3] <= l-2
);

constraint forall(p in 1..nnext)(
 (per[1] == next[p, 1] /\ per[2] == next[p, 2]) \/
 (per[l] == next[p, 1] /\ per[l-1] == next[p, 2]) \/
 exists(i in 2..l-1)(
   (per[i] == next[p, 1] /\ (per[i+1] == next[p, 2] \/ per[i-1] ==
next[p, 2]))
 )
);

constraint forall(p in 1..nsep)(
 (per[1] == separate[p, 1] /\ per[2] != separate[p, 2]) \/
 (per[l] == separate[p, 1] /\ per[l-1] != separate[p, 2]) \/
 exists(i in 2..l-1)(
```

```
   (per[i] == separate[p, 1] /\ per[i+1] != separate[p, 2] /\ per[i-1]
!= separate[p, 2])
 )
);

constraint forall(p in 1..ndis)(
 exists(i,j in 1..l)(
     per[i] == distance[p, 1] /\ per[j] == distance[p, 2] /\
     abs(i - j) - 1 <= distance[p, 3] /\ abs(i - j) - 1 >= 1
 )
);

output[show(per)];
```

6. CONSTRUCCIÓN DE UN RECTÁNGULO

Análisis

Como parámetro se recibe el ancho y el largo $(width, height)$ los cuales representan las dimensiones del rectángulo a construir. sq es el arreglo de entrada donde cada elemento representa el tamaño de un cuadrado.

$$\{width\ height\} \in N$$
$$sq = [x_1, ..., x_n]\ donde\ x_i \in N$$

Se declaran los arreglos de variables correspondientes a cada una de las coordenadas sobre el plano y el tamaño correspondiente que debe obtener para formar el rectángulo deseado.

$$x = [x_1, ...x_n]\ donde\ x_i \in \{0, .., max(\ width,\ height\ )\}$$
$$y = [y_1, ..y_n]\ donde\ y_i \in \{0, .., max(\ width,\ height\ )\}$$

El dominio de los elementos de x e y se declara de esta manera para que los cuadrados se ubiquen en una parte limitada del plano, reduciendo el espacio donde se pueden colocar.

Las restricciones buscarán entonces posicionar los cuadrados y evitar el solapamiento:

1. $(\sum_{i=1}^{n} sq_i^2) = width * height$

2. $\forall_{i \in \{1,..,n\}}\ x_i + sq_i \leq width\ \wedge\ y_i + sq_i \leq height$

3. $\forall_{i \in \{1,..,n\}} \forall_{j \in \{1,..,n\}} x_i + sq_i \leq x_j \vee x_j + sq_j \leq x_i \vee y_i + sq_i \leq y_j \vee y_j + sq_j \leq y_i$

La primer restricción garantiza que los cuadrados ingresados suman el área total que se necesita. La segunda permite que cada cuadrado teniendo en cuenta su posición y tamaño no se desborde del límite del rectángulo. La tercer restricción permite evitar el solapamiento sobre cualquier par de cuadrados presentes teniendo en cuenta su posible posición inicial y el tamaño correspondiente de un cierto cuadrado.

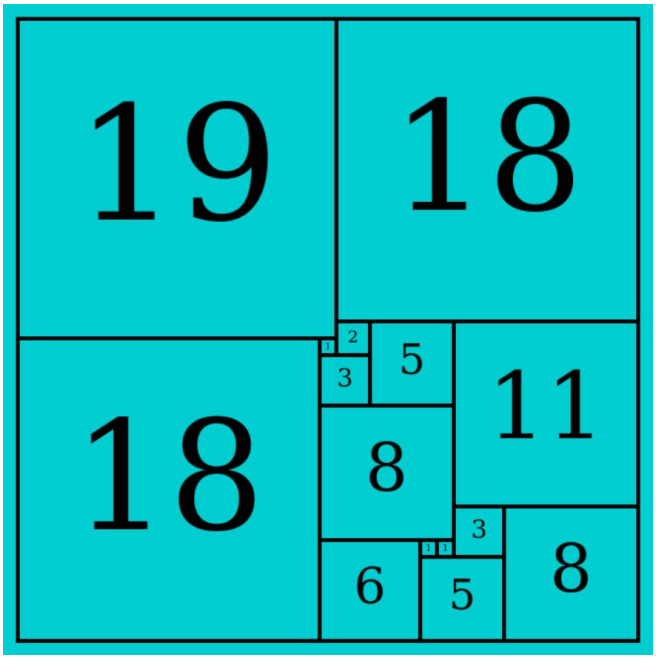
Implementación

```
% Parameters
int: width;
int: height;
array[int] of int: sq;

% Constants
int: n_sq = length(sq);
int: max = if width > height then width else height endif;

% Array of variables
array[1..n_sq] of var 0..max: x;
array[1..n_sq] of var 0..max: y;

%Constraints
constraint sum(i in 1..n_sq)( sq[i]^2 ) == width * height;

constraint forall(i in 1..n_sq)(x[i] + sq[i] <= width /\ y[i] + sq[i]
<= height);

constraint forall(i, j in 1..n_sq where i > j)(
                x[i] + sq[i] <= x[j] \/ x[j] + sq[j] <= x[i] \/
                y[i] + sq[i] <= y[j] \/ y[j] + sq[j] <= y[i]);

solve satisfy;

output ["width = ", show(width), " height = ", show(height), "\n"] ++
["x = ", show(x), "\n"] ++
["y = ", show(y), "\n"];
```

Una instancia particular del problema sería:

```
width = 37;
height = 37;
```

```
sq = [19, 18, 18, 11, 8, 8, 6, 5, 5, 3, 3, 2, 1, 1, 1];
```



PARTE 2 COPs

7. CONSTRUCCIÓN DE TRANSFORMADORES

Análisis

Sea $lFerro$ y $lHours$ el límite de material ferromagnético y las horas disponibles respectivamente. n como el número de tipo de transformadores que se pueden construir y los arreglos $ferro, hours\ y\ prof$ que representan el gasto de material, de horas y el beneficio por construir un transformador i respectivamente.
Sea $Transformers$ el arreglo de variables donde se calculara la cantidad de cada transformador i a construir.

Las restricciones a tener en cuenta serían las siguientes:

1. $\forall_{i \in \{1,..,n\}} transformers_i \geq 0$
2. $(\sum_{i=1}^{n} ferro_i * transformers_i) \leq lFerro$
3. $(\sum_{i=1}^{n} hours_i * transformers_i) \leq lHours$

Función objetivo a maximizar:

4. $\sum_{i=1}^{n} prof_i * transformers_i$

Implementación

```
%Parameters
int: n;
int: lFerro;
int: lHours;
array[1..n] of int: ferro;
array[1..n] of int: hours;
array[1..n] of int: prof;

%Array of decision variables
array[1..n] of var int: transformers;

%Constraints
constraint forall(i in 1..n)( transformers[i] >= 0 );
% Ferromagneto
constraint sum(i in 1..n)( ferro[i] * transformers[i] ) <= lFerro;
% Hours of work
constraint sum(i in 1..n)( hours[i] * transformers[i] ) <= lHours;

%Objetive function
solve maximize sum(i in 1..n)( prof[i] * transformers[i] );

%Output
output [ "Transformer\(i) = \(transformers[i])\n" | i in 1..n ];
output [ "Profit = $\(sum(i in 1..n)( prof[i]*transformers[i] ))\n"];
output [ "ferro = \(sum(i in 1..n)( ferro[i]*transformers[i] ))\n"];
output [ "hours = \(sum(i in 1..n)( hours[i]*transformers[i] ))"];
```

El problema específico que se plantea es:

Límite de material Ferromagnético = 6 toneladas.
Limite de tiempo = 28 Horas.

	Tipo 1	Tipo 2
Material F.	2 Toneladas	1 Toneladas
Tiempo	7 Horas	8 horas
Beneficio	120 000 dólares	80 000 dólares

Entrada:

```
n = 2;
lFerro = 6;
lHours = 28;
ferro = [2, 1];
hours = [7, 8];
prof = [120000, 80000];
```

Salida:

```
Transformer1 = 3
Transformer2 = 0
Profit = $360000
ferro = 6
hours = 21
```