# Easy as Sum

Relatório Final

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica


**Grupo Easy as Sum_5:**

Pedro Dias Faria – ei11167

Rui Filipe de Oliveira Donas-Botto Figueira – ei11021

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal


13 de Dezembro de 2014

## Resumo

Este relatório tem como objectivo mostrar como foi desenvolvido o programa de pesquisa de solução, em Prolog, para o puzzle Easy as Sum. Este projecto foi realizado no âmbito da cadeira de Programação em Lógica do primeiro semestre do 3º ano do Mestrado Integrado em Engenharia Informática e de Computação.

## Índice

# 1 - Introdução

Este projecto teve como objectivo o desenvolvimento de um programa para encontrar a solução do puzzle Easy as Sum, jogo este, que devido á sua natureza pode ser descrito e transposto para um conjunto de regras facilmente descrito e resolvido pela programação em lógica com restrições, em prolog. O nosso papel enquanto programadores foi o de fazer a tranposição dessas regras de funcionamento para um conjunto de restrições a serem aplicadas na procura de uma solução para as mesmas.

## 2 - Descrição do Problema

O puzzle Easy as Sum, é um pouco semelhante com o jogo Sudoku no sentido em que tem como objectivo o preenchimento de uma matriz quadrada com dígitos de modo a que cada coluna e fila contenham cada dígito exactamente uma vez. Os dígitos usados para preencher a matriz, dependem do tamanho da mesma, indo de 1 até x-1, sendo x o número de células de cada fila/coluna. A principal diferença relativamente ao Sudoku é que nesta matriz, cada um dos lado da mesma possui uma sequência de números que representam restrições no modo como os dígitos são colocados pelo jogador.

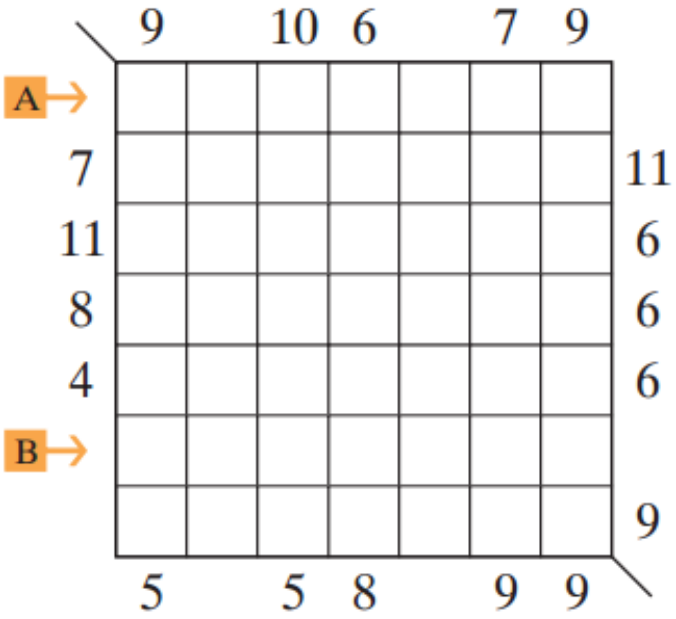



**Figura 1 - Matriz de Restrições de um puzzle 7x7**

Os números no topo e na direita representam a soma dos primeiros e últimos dígitos na coluna ou fila correspondente e os números na esquerda e no fundo são a soma dos segundos e penúltimos dígitos. Nestas somas, apenas são considerados dígitos de células preenchidas, ou seja, no caso de a primeira célula de uma coluna estar vazia, é usada a célula seguinte para o cálculo. Dentro destas sequências de restrições pode haver valores em branco, o que representa a ausência de uma regra de colocação dos dígitos para essa linha ou coluna, como é o caso da segunda coluna, ou da primeira linha na imagem. A solução é atingida quando toda a matriz for preenchida de acordo com as regras em cima descritas.

# 3 - Abordagem

## 3.1 - Váriaveis de Decisão

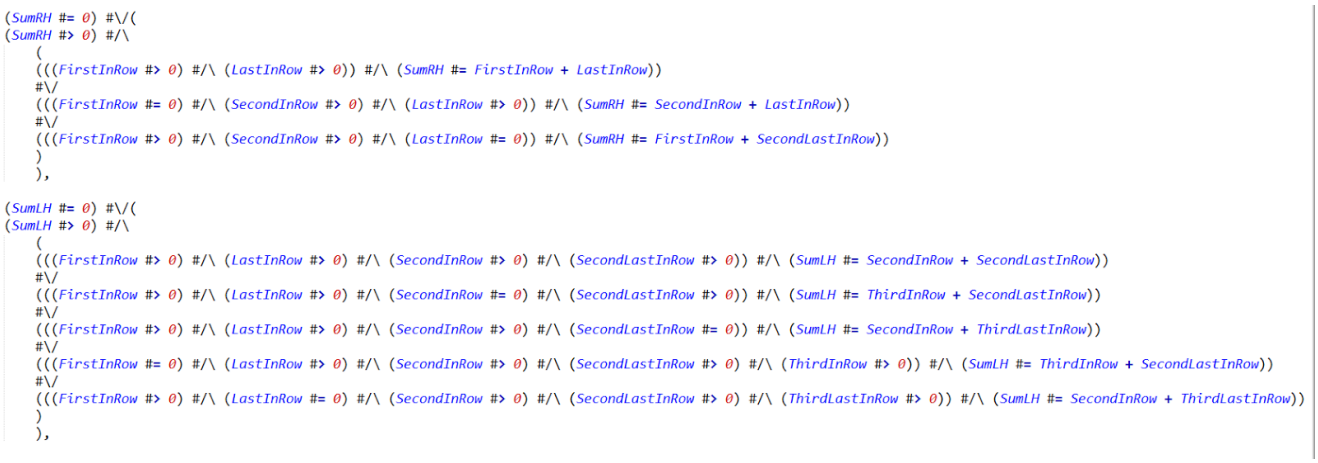
As variáveis de decisão utilizadas no nosso programa consistem nas listas de restrições, como exemplificadas na **Figura 1**. Tal como foi referido em cima, estas listas podem conter qualquer valor a começar em 0, sendo que este, quando presente numa lista de restrição, representa a ausência da mesma nessa linha/coluna. Foi também necessário restringir o domínio dos elementos da matriz desde 0, representando este uma célula vazia, até Tamanho da Matriz -1.

## 3.2 - Restrições

As restrições utilizadas para a pesquisa de solução do nosso problema foram as seguintes:

- Garantir que cada linha/coluna possui cada dígito exatamente uma vez, obrigando todos os elementos das mesmas a serem distintos entre si de modo a não existirem repetições. Esta restrição é colocada através do predicado **put_distinct(Sol, Gridsize)**, que itera sobre cada linha da matriz através de **put_distinct_line(Sol, Gridsize, 1)**, sendo depois transposta a matriz para colocar todos os elementos da coluna, também, com valores distintos.

- Restrições a nível dos somatórios dos dígitos de cada linha/coluna de acordo com as listas de restrições impostas, ou seja, obrigando a soma dos primeiros e últimos dígitos na coluna ou fila a serem igual aos valores no topo e na direita, assim como a soma dos segundos e penúltimos dígitos ser igual aos valores dos números na esquerda e no fundo. Isto é conseguido criando as restrições para cada linha da matriz, através do predicado **put_restrictions_line(Sol, Gridsize, SumR, SumL, 1)**. Este predicado, recebe como argumentos a matriz a solucionar, **Sol**, o tamanho desta, **Gridsize**, a lista de restrições à direita, **SumR** e a lista de restrições à esquerda **SumL**. Do mesmo modo, para as colunas, é chamado o mesmo predicado, mas para a matriz transposta, através de **transpose(Sol, Sol2), put_restrictions_line(Sol2, Gridsize, SumT, SumB, 1).** seguindo a mesma parametrização que a anterior.



```
(SumRH #= 0) #\/(
(SumRH #> 0) #/\
    (
    (((FirstInRow #> 0) #/\ (LastInRow #> 0)) #/\ (SumRH #= FirstInRow + LastInRow))
    #\/
    (((FirstInRow #= 0) #/\ (SecondInRow #> 0) #/\ (LastInRow #> 0)) #/\ (SumRH #= SecondInRow + LastInRow))
    #\/
    (((FirstInRow #> 0) #/\ (SecondInRow #> 0) #/\ (LastInRow #= 0)) #/\ (SumRH #= FirstInRow + SecondLastInRow))
    )
    ),

(SumLH #= 0) #\/(
(SumLH #> 0) #/\
    (
    (((FirstInRow #> 0) #/\ (LastInRow #> 0) #/\ (SecondInRow #> 0) #/\ (SecondLastInRow #> 0)) #/\ (SumLH #= SecondInRow + SecondLastInRow))
    #\/
    (((FirstInRow #> 0) #/\ (LastInRow #> 0) #/\ (SecondInRow #= 0) #/\ (SecondLastInRow #> 0)) #/\ (SumLH #= ThirdInRow + SecondLastInRow))
    #\/
    (((FirstInRow #> 0) #/\ (LastInRow #> 0) #/\ (SecondInRow #> 0) #/\ (SecondLastInRow #= 0)) #/\ (SumLH #= SecondInRow + ThirdLastInRow))
    #\/
    (((FirstInRow #= 0) #/\ (LastInRow #> 0) #/\ (SecondInRow #> 0) #/\ (SecondLastInRow #> 0) #/\ (ThirdInRow #> 0)) #/\ (SumLH #= ThirdInRow + SecondLastInRow))
    #\/
    (((FirstInRow #> 0) #/\ (LastInRow #= 0) #/\ (SecondInRow #> 0) #/\ (SecondLastInRow #> 0) #/\ (ThirdLastInRow #> 0)) #/\ (SumLH #= SecondInRow + ThirdLastInRow))
    )
    ),
```

**Figura 2 - Código da implementação das restrições descritas**

## 3.3 - Função de Avaliação

A solução é avaliada conforme a colocação das restrições. Cada elemento da matriz obedece às restrições descritas no ponto 3.2.

## 3.4 - Estratégia de Pesquisa

Para a resolução do tipo de puzzle em causa, em que apenas existe uma única solução para cada conjunto de restrições imposta, não foi utilizada nenhuma estratégia de pesquisa em específico.

# 4 - Visualização da Solução

## 4.1 - Restrições fornecidas e geração de solução

Para a procura da solução do enunciado, é necessário correr o predicado **easyAsSum(Sol, 1).** O valor passado como segundo argumento, serve para escolher as restrições a usar, que estarão definidas no código antes da sua execução. Com a solução encontrada e guardada na matriz de listas **Sol** , a mesma é enviada para o predicado **draw_grid(Sol, Gridsize)**, que percorre cada lista da lista de listas que forma a matriz e imprime a mesma em formato de texto. Juntamente com a matriz solução, imprimos também o tempo de pesquisa da mesma , após impostas as restrições. Um exemplo da visualização é o desta matriz, que corresponde à solução das restrições impostas pela **Figura 1** :

```
------E A S Y   A S   S U M------

Grid: 7x7

Searching duration 0.952 sec.

---------------
| |4|6|5|2|1|3|
---------------
|6| |3|2|1|4|5|
---------------
|4|5|1|3|6| |2|
---------------
|5|6| |4|3|2|1|
---------------
|2|1|5|6| |3|4|
---------------
|1|3|2| |4|5|6|
---------------
|3|2|4|1|5|6| |
---------------
```

**Figura 3 - Solução das restrições da Figura 1**

Em termos de representação relativamente ao código, a matriz de restrições da **Figura 1** é representada por 4 listas, sendo que os seus nomes indicam qual a localização das mesmas :

```
%Restricoes a efetuar na grelha
sumTop(     [9,  0, 10, 6, 0, 7, 9], 1).
sumRight(   [0, 11,  6, 6, 6, 0, 9], 1).
sumBottom(  [5,  0,  5, 8, 0, 9, 9], 1).
sumLeft(    [0,  7, 11, 8, 4, 0, 0], 1).
```

**Figura 4 - Matriz de restrições da Figura 1**

Considerando agora uma nova matriz de restrições, desta vez com o tamanho 5x5 :

```
%Restricoes 5x5
sumTop(     [7, 6, 4, 7, 3], 2).
sumRight(   [5, 5, 5, 5, 5], 2).
sumBottom(  [3, 4, 6, 3, 7], 2).
sumLeft(    [5, 5, 5, 5, 5], 2).
```

**Figura 5 - Matriz de restrições 5x5**

E a respectiva apresentação da solução :

```
------E A S Y   A S   S U M------

Grid: 5x5

Searching duration 0.015 sec.

-----------
|3|4|1| |2|
-----------
|1|3|2|4| |
-----------
|2| |4|1|3|
-----------
| |1|3|2|4|
-----------
|4|2| |3|1|
-----------
```

**Figura 6 - Solução das restrições da figura 5**

## 4.2 - Geração de solução aleatória e restrições

Como complemento extra ao nosso trabalho, implementámos uma função para a geração aleatória de um conjunto de restrições a ser aplicado a uma matriz de solução, passando apenas qual o tamanho da matriz desejada. Esta geração é criada através do predicado **easyAsSum(Gridsize)**. Este, começa por criar uma matriz do tamanho desejado, sendo 4 o tamanho mínimo desta, assim como as quatro listas de restrições com o mesmo comprimento da matriz. Segue-se depois a aplicação das restrições geradas aleatoriamente a que a matriz de solução deve obedecer, sendo por fim gerada a solução de acordo com as mesmas. A seguir seguem-se alguns exemplos deste processo de geração aleatório.

```
------E A S Y   A S   S U M------

Grid: 5x5


-----------
|2|1|4| |3|
-----------
|3|2| |1|4|
-----------
|4| |1|3|2|
-----------
| |3|2|4|1|
-----------
|1|4|3|2| |
-----------


Top Constraints[3,5,7,3,4]
Right Constraints[5,7,6,4,3]
Bottom Constraints[7,5,3,7,6]
Left Constraints[5,3,4,6,7]
Searching duration 0.016 sec.
```

```
------E A S Y   A S   S U M------

Grid: 7x7


---------------
|3| |4|5|6|2|1|
---------------
|6|1|3| |2|5|4|
---------------
|2|6| |3|1|4|5|
---------------
| |3|6|4|5|1|2|
---------------
|5|4|2|1|3|6| |
---------------
|4|5|1|2| |3|6|
---------------
|1|2|5|6|4| |3|
---------------


Top Constraints[4,3,9,11,10,5,4]
Right Constraints[4,10,7,5,11,10,4]
Bottom Constraints[10,11,4,5,5,11,10]
Left Constraints[6,6,10,7,7,8,6]
Searching duration 0.015 sec.
```

**Figura 7 e 8 - Matrizes de tamanho 5 e 7 geradas aleatoriamente**

## 5 - Resultados

Como experiência final, foi utilizada uma série de conjuntos de restrições para a mesma solução, sendo que a única diferença entre as mesmas, era a existência de algumas restrições nulas. Através desta experiência chegamos á conclusão que quanto menor for o número de restrições nulas, mais rápida é encontrada a solução.

```
%Todas as restricoes preenchidas
sumTop(     [3, 11, 11,  4, 11, 11,  8,  3], 3).
sumRight(   [3,  5, 11, 11,  7, 11, 11,  3], 3).
sumBottom(  [7,  8,  7, 10,  7,  7,  6, 12], 3).
sumLeft(    [9, 13,  9,  9,  7,  5,  7,  9], 3).

%1 restricao nula
sumTop(     [3, 11, 11,  4, 11, 11,  8,  3], 4).
sumRight(   [3,  5, 11, 11,  7, 11, 11,  3], 4).
sumBottom(  [7,  8,  7, 10,  7,  7,  0, 12], 4).
sumLeft(    [9, 13,  9,  9,  7,  5,  7,  9], 4).

%2 restricoes nulas
sumTop(     [3,  0, 11,  4, 11, 11,  8,  3], 5).
sumRight(   [3,  5, 11, 11,  7, 11, 11,  3], 5).
sumBottom(  [7,  8,  7, 10,  7,  7,  0, 12], 5).
sumLeft(    [9, 13,  9,  9,  7,  5,  7,  9], 5).

%6 restricoes nulas
sumTop(     [3,  0, 11,  4, 11, 11,  8,  3], 6).
sumRight(   [3,  5, 11, 11,  7,  0, 11,  0], 6).
sumBottom(  [7,  0,  7, 10,  7,  7,  0, 12], 6).
sumLeft(    [9, 13,  0,  9,  7,  5,  7,  9], 6).
```

**Figura 9 - Código das listas de restrições usadas para testar a mesma solução**

```
| ?- easyAsSum(S, 3).

------E A S Y   A S   S U M------

Grid: 8x8

Searching duration 0.406 sec.

-----------------
|1|6|4| |5|7|3|2|
-----------------
|3|7|5|1|4|6|2| |
-----------------
|6|2| |4|1|3|7|5|
-----------------
|7|3|1|2| |5|6|4|
-----------------
| |4|6|5|7|2|1|3|
-----------------
|5|1|3|7|2| |4|6|
-----------------
|4| |2|6|3|1|5|7|
-----------------
|2|5|7|3|6|4| |1|
-----------------
```

**Figura 10 - Matriz de solução para a lista de restrições sem nenhuma nula**

```
| ?- easyAsSum(S, 4).

------E A S Y   A S   S U M------

Grid: 8x8

Searching duration 0.515 sec.

-----------------
|1|6|4| |5|7|3|2|
-----------------
|3|7|5|1|4|6|2| |
-----------------
|6|2| |4|1|3|7|5|
-----------------
|7|3|1|2| |5|6|4|
-----------------
| |4|6|5|7|2|1|3|
-----------------
|5|1|3|7|2| |4|6|
-----------------
|4| |2|6|3|1|5|7|
-----------------
|2|5|7|3|6|4| |1|
-----------------
```

**Figura 11 - Matriz de solução para a lista de restrições com uma nula**

```
| ?- easyAsSum(S, 5).

------E A S Y   A S   S U M------

Grid: 8x8

Searching duration 1.326 sec.

-----------------
|1|6|4| |5|7|3|2|
-----------------
|3|7|5|1|4|6|2| |
-----------------
|6|2| |4|1|3|7|5|
-----------------
|7|3|1|2| |5|6|4|
-----------------
| |4|6|5|7|2|1|3|
-----------------
|5|1|3|7|2| |4|6|
-----------------
|4| |2|6|3|1|5|7|
-----------------
|2|5|7|3|6|4| |1|
-----------------
```

**Figura 12 - Matriz de solução para a lista de restrições com duas nulas**

```
| ?- easyAsSum(S, 6).

------E A S Y   A S   S U M------

Grid: 8x8

Searching duration 12.293 sec.

-----------------
|1|5|7|3|6|4|2| |
-----------------
|3|6|5|4| |1|7|2|
-----------------
|6|7|1|2|4|3| |5|
-----------------
|7| |6|5|1|2|3|4|
-----------------
| |1|3|7|2|5|4|6|
-----------------
|5|4|2|6|7| |1|3|
-----------------
|4|2| |1|3|6|5|7|
-----------------
|2|3|4| |5|7|6|1|
-----------------
```

**Figura 13 - Matriz de solução para a lista de restrições com seis nulas**

## 6 - Conclusões e Trabalho Futuro

Com este trabalho, acreditamos que aprofundamos o nosso conhecimento e uso de restrições em prolog para a resolução de problemas complexos. Permitiu-nos ainda melhorar o nosso conhecimento sobre prolog relativamente ao primeiro projecto, devido á diferente abordagem a que recorremos para desenvolver o mesmo. A nível de objectivos, cumprimos com todos os que tinhamos planeados no ínicio, inclusivé com a geração aleatória de matrizes de solução, com as respectivas restrições. O balanço final face ás nossas expectativas e objectivos é positivo.

## 7 - Anexos

```
:-use_module(library(clpfd)).
:-use_module(library(lists)).
:-use_module(library(random)).

%%%%%%%%%%%%%%%%%%%%%%%%%
%       EASY AS SUM       %
%%%%%%%%%%%%%%%%%%%%%%%%%

%TRABALHO REALIZADO POR:
%Pedro Faria        -       ei11167
%Rui Figueira       -       ei11021

%0 Interface
draw_empty_line(0):-write('-'),nl.
draw_empty_line(N):-write('--'), N1 is N-1, draw_empty_line(N1).

draw_line([]).
draw_line([H|T]):-    write('|'),
                                ((H =:= 0) -> write(' ');
                                                write(H)
                                ) ,
                                draw_line(T).

draw_lines(_, 0, _).
draw_lines([H|T], N, GSize):- draw_line(H), write('|'), nl, draw_empty_line(GSize), N2 is
N-1, draw_lines(T,N2, GSize).

draw_grid(Grid, GSize):- draw_empty_line(GSize), draw_lines(Grid, GSize, GSize).

%1 Decl Vars

%Restricoes a efetuar na grelha

%Restricoes do enunciado
sumTop(                 [9,  0, 10, 6, 0, 7, 9], 1).
sumRight(       [0, 11,  6, 6, 6, 0, 9], 1).
```

```
sumBottom(   [5,  0,  5, 8, 0, 9, 9], 1).
sumLeft(     [0,  7, 11, 8, 4, 0, 0], 1).

%Restricoes 5x5
sumTop(      [7, 6, 4, 7, 3], 2).
sumRight(    [5, 5, 5, 5, 5], 2).
sumBottom(   [3, 4, 6, 3, 7], 2).
sumLeft(     [5, 5, 5, 5, 5], 2).

%Solucao 2
% 3 | 4 | 1 |   | 2
% 1 | 3 | 2 | 4 |
% 2 |   | 4 | 1 | 3
%   | 1 | 3 | 2 | 4
% 4 | 2 |   | 3 | 1

%Todas as restricoes preenchidas
sumTop(      [3, 11, 11,  4, 11, 11,  8,  3], 3).
sumRight(    [3,  5, 11, 11,  7, 11, 11,  3], 3).
sumBottom(   [7,  8,  7, 10,  7,  7,  6, 12], 3).
sumLeft(     [9, 13,  9,  9,  7,  5,  7,  9], 3).

%1 restricao nula
sumTop(      [3, 11, 11,  4, 11, 11,  8,  3], 4).
sumRight(    [3,  5, 11, 11,  7, 11, 11,  3], 4).
sumBottom(   [7,  8,  7, 10,  7,  7,  0, 12], 4).
sumLeft(     [9, 13,  9,  9,  7,  5,  7,  9], 4).

%2 restricoes nulas
sumTop(      [3,  0, 11,  4, 11, 11,  8,  3], 5).
sumRight(    [3,  5, 11, 11,  7, 11, 11,  3], 5).
sumBottom(   [7,  8,  7, 10,  7,  7,  0, 12], 5).
sumLeft(     [9, 13,  9,  9,  7,  5,  7,  9], 5).

%6 restricoes nulas
sumTop(      [3,  0, 11,  4, 11, 11,  8,  3], 6).
sumRight(    [3,  5, 11, 11,  7,  0, 11,  0], 6).
sumBottom(   [7,  0,  7, 10,  7,  7,  0, 12], 6).
sumLeft(     [9, 13,  0,  9,  7,  5,  7,  9], 6).
```

```
%2 Decl Dominios
gridSize([],0).
gridSize([_|L],N) :- gridSize(L,N1), N is N1 + 1.

len_row([],_).
len_row([L|T],NumberCol):-
                                        length(L,NumberCol),      % define o numero de
celulas de cada linha da grelha final
                                        len_row(T,NumberCol).

%Dominio: [0 - Tamanho-1]
%Exempl: Grelha 7x7, tem dominio [0-6]
put_domain_row(_, _, 0).
put_domain_row([H|T], UpperLimit, N):- N > 0,
                                                domain(H, 0, UpperLimit),
                                                N1 is N-1,
                                                put_domain_row(T,
UpperLimit, N1).


put_domains(Sol, Gridsize):- UpperLimit is Gridsize-1,
                                        N is Gridsize,
                                        put_domain_row(Sol, UpperLimit, N).


%3 Decl Restricoes
%Um numero nao se pode repetir em cada linha
put_distinct_line(_, Gridsize, N):- N > Gridsize.
put_distinct_line([H|T], Gridsize, N):-
                                        all_distinct(H),
                                        N1 is N + 1,
                                        put_distinct_line(T, Gridsize, N1).

put_distinct(Sol, Gridsize):-put_distinct_line(Sol, Gridsize, 1),
                                        transpose(Sol, SolTransp),
                                        put_distinct_line(SolTransp, Gridsize, 1).
```

```
put_restrictions_line([], _, _,_,_).
put_restrictions_line([H|T], Gridsize, [SumRH|SumRT],[SumLH|SumLT], N):-
                                                element(1, H, FirstInRow),
                                                element(Gridsize, H, LastInRow),

                                                element(2, H, SecondInRow),
                                                SecondLastIndex is Gridsize -1,
                                                element(SecondLastIndex,        H,
SecondLastInRow),

                                                element(3, H, ThirdInRow),
                                                ThirdLastIndex is Gridsize -2,
                                                element(ThirdLastIndex,         H,
ThirdLastInRow),

                                                (SumRH #= 0) #\/(
                                                (SumRH #> 0) #/\
                                                        (
                                                        (((FirstInRow   #>   0)   #/\
(LastInRow #> 0)) #/\ (SumRH #= FirstInRow + LastInRow))
                                                        #\/
                                                        (((FirstInRow   #=   0)   #/\
(SecondInRow #> 0) #/\ (LastInRow #> 0)) #/\ (SumRH #= SecondInRow + LastInRow))
                                                        #\/
                                                        (((FirstInRow   #>   0)   #/\
(SecondInRow   #>   0)   #/\   (LastInRow   #=   0))   #/\   (SumRH   #=   FirstInRow   +
SecondLastInRow))
                                                        )
                                                        ),

                                                (SumLH #= 0) #\/(
                                                (SumLH #> 0) #/\
                                                        (
                                                        (((FirstInRow   #>   0)   #/\
(LastInRow #> 0) #/\ (SecondInRow #> 0) #/\ (SecondLastInRow #> 0)) #/\ (SumLH #=
SecondInRow + SecondLastInRow))
                                                        #\/
```

```
                                                    (((FirstInRow #> 0) #/\
(LastInRow #> 0) #/\ (SecondInRow #= 0) #/\ (SecondLastInRow #> 0)) #/\ (SumLH #=
ThirdInRow + SecondLastInRow))
                                                    #\/
                                                    (((FirstInRow #> 0) #/\
(LastInRow #> 0) #/\ (SecondInRow #> 0) #/\ (SecondLastInRow #= 0)) #/\ (SumLH #=
SecondInRow + ThirdLastInRow))
                                                    #\/
                                                    (((FirstInRow #= 0) #/\
(LastInRow #> 0) #/\ (SecondInRow #> 0) #/\ (SecondLastInRow #> 0) #/\ (ThirdInRow #>
0)) #/\ (SumLH #= ThirdInRow + SecondLastInRow))
                                                    #\/
                                                    (((FirstInRow #> 0) #/\
(LastInRow #= 0) #/\ (SecondInRow #> 0) #/\ (SecondLastInRow #> 0) #/\
(ThirdLastInRow #> 0)) #/\ (SumLH #= SecondInRow + ThirdLastInRow))
                                                    )
                                                    ),

                                                N1 is N+1,
                                                put_restrictions_line(T,        Gridsize,
SumRT, SumLT, N1).


put_restrictions(Sol, Gridsize, SumT, SumR, SumB, SumL):-
                                                put_restrictions_line(Sol,      Gridsize,
SumR, SumL, 1),
                                                transpose(Sol, Sol2),
                                                put_restrictions_line(Sol2,     Gridsize,
SumT, SumB, 1).

%4 Pesq Solucao

%Ex é o numero do exemplo definido nas declaracoes acima
%Para pesquisar a solucao do enunciado: easyAsSum(Sol, 1).
easyAsSum(Sol, Ex):-
                        sumTop(SumT, Ex), sumRight(SumR, Ex), sumBottom(SumB,
Ex),sumLeft(SumL, Ex),
%Inicializacao das listas de restricao
```

```
                    gridSize(SumT, Gridsize),gridSize(SumR, Gridsize),gridSize(SumB, Gridsize),gridSize(SumL, Gridsize),      %Confirmar que ha o mesmo numero de elementos nas listas de restricao

                    nl,nl,write('------E A S Y   A S   S U M------'),
                    nl,nl,write('Grid:
'),write(Gridsize),write('x'),write(Gridsize),nl,nl,

                    length(Sol, Gridsize),
                            %Lista com o tamanho da grelha final
                    len_row(Sol, Gridsize),
                            %Numero de listas de listas igual ao tamanho da grelha final
                    put_domains(Sol, Gridsize),
                        %Colocar dominios
                    put_distinct(Sol, Gridsize),
                    %Colocar todas as linhas/colunas com elementos diferentes
                    put_restrictions(Sol, Gridsize, SumT, SumR, SumB, SumL),
            %Colocar restricoes de acordo com as listas
                    append(Sol, Sol2),!,
                    statistics(runtime, [T0| _]),
                    %Instante em que inicia o predicado labeling
                    labeling([], Sol2),
                            %Pesq da Solucao
                    statistics(runtime, [T1|_]),
                    %instante em que termina o predicado labeling
                    T is T1 - T0,
                    format('Searching duration ~3d sec.~n', [T]),
                %Print de estatisticas
                    nl,draw_grid(Sol, Gridsize),nl,nl,nl.
                %Print da Solucao


%% choose(List, Elt) - chooses a random element
%% in List and unifies it with Elt.
choose([], []).
choose(List, Elt) :-
        length(List, Length),
        random(0, Length, Index),
```

```
        nth0(Index, List, Elt).

%% shuffle(ListIn, ListOut) - randomly shuffles
%% ListIn and unifies it with ListOut
shuffle([], []).
shuffle(List, [Element|Rest]) :-
        choose(List, Element),
        delete(List, Element, NewList),
        shuffle(NewList, Rest).

%Funcao para misturar cada linha/coluna da matriz
shuffle_grid(Sol, Shuffled):- shuffle(Sol, A), transpose(A, B), shuffle(B, Shuffled).

%Gridsize e' o tamanho desejado para a matriz aleatoria
easyAsSum(Gridsize):-

                        length(SumT, Gridsize),length(SumR, Gridsize),length(SumB,
Gridsize),length(SumL, Gridsize),       %Confirmar que ha o mesmo numero de
elementos nas listas de restricao

                        nl,nl,write('------E A S Y  A S  S U M------'),
                        nl,nl,write('Grid:
'),write(Gridsize),write('x'),write(Gridsize),nl,nl,

                        ( (Gridsize >= 5);
                                write('Size must be > 4'),nl,nl,nl,fail
                                ),

                        statistics(runtime, [T0| _]),
                        %Instante em que inicia o predicado labeling
                        length(Sol, Gridsize),
                                %Lista com o tamanho da grelha final
                        len_row(Sol, Gridsize),
                        put_domains(Sol, Gridsize),
                                %Colocar dominios
                        put_distinct(Sol, Gridsize),
                        %Colocar todas as linhas/colunas com elementos diferentes
                        put_restrictions(Sol, Gridsize, SumT, SumR, SumB, SumL),
                %Colocar restricoes de acordo com as listas
```

```
                append(Sol, Sol2),!,

                labeling([], Sol2),
                                %Pesq da Solucao

                MaxSum is (Gridsize-1) + (Gridsize-2),
                %A soma maxima sera' a soma dos 2 maiores elementos do
dominio
                domain(SumT, 1, MaxSum),
                labeling([], SumT),
                domain(SumR, 1, MaxSum),
                labeling([], SumR),
                domain(SumB, 1, MaxSum),
                labeling([], SumB),
                domain(SumL, 1, MaxSum),
                labeling([], SumL),

                shuffle_grid(Sol, Shufl),
                length(ShuflST,                 Gridsize),length(ShuflSR,
Gridsize),length(ShuflSB, Gridsize),length(ShuflSL, Gridsize),  %Cria aleatoriedade entre
as linhas e colunas da matriz

                put_restrictions(Shufl, Gridsize, ShuflST, ShuflSR, ShuflSB,
ShuflSL),                                                       %Colocar
restricoes de acordo com as listas
                append(Shufl, Shufl2),!,
                labeling([], Shufl2),!,
                domain(ShuflST, 1, MaxSum),
                labeling([], ShuflST),!,
                domain(ShuflSR, 1, MaxSum),
                labeling([], ShuflSR),!,
                domain(ShuflSB, 1, MaxSum),
                labeling([], ShuflSB),!,
                domain(ShuflSL, 1, MaxSum),
                labeling([], ShuflSL),!,

                statistics(runtime, [T1|_]),
                %instante em que termina o predicado labeling
```

```
        T is T1 - T0,
        nl,draw_grid(Shufl, Gridsize),nl,nl,nl,

        write('Top Constraints'),write(ShuflST),nl,
        write('Right Constraints'),write(ShuflSR),nl,
        write('Bottom Constraints'),write(ShuflSB),nl,
        write('Left Constraints'),write(ShuflSL),nl,

        format('Searching duration ~3d sec.~n', [T]),nl,nl.
%Print de estatisticas
```