

Relatório Engenharia Do Conhecimento

Fase 1

Grupo 21:

João Marques nº49038

João David nº49448

Luís Moreira nº49531

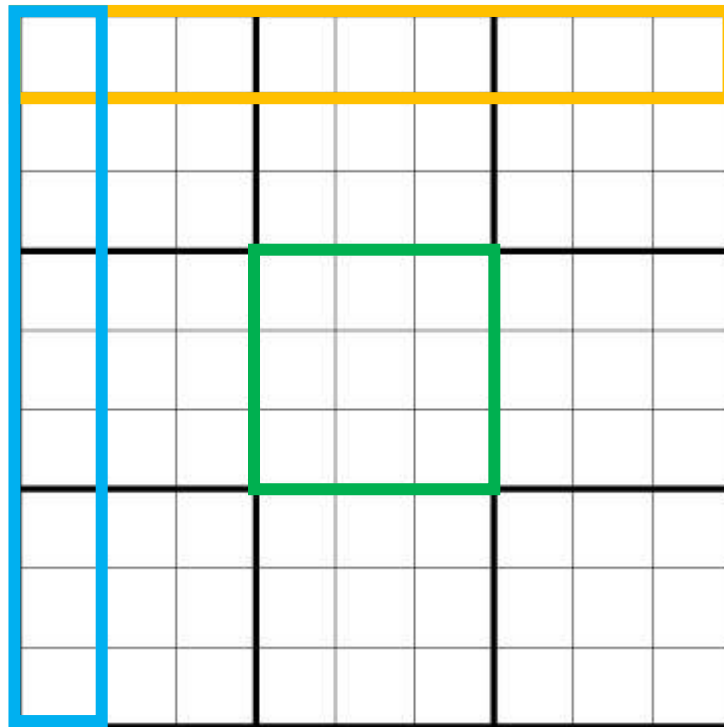
CONTEÚDO

Puzzle Escolhido: Sudoku	3
Representação CSP do sudoku	4
Programa que cria o Ficheiro com a representação F2CSP do Sudoku	5
Representação do input.....	6
Exemplo de Ficheiro de input.....	6
Algoritmo Para as restrições	7
Estatísticas do ficheiro F2CSP	9
Programa que devolve o ficheiro com as estatísticas do ficheiro F2CSP.....	9

PUZZLE ESCOLHIDO: SUDOKU

Como sabemos, o puzzle Sudoku tem três principais restrições, em que um número não pode ser repetido em:

1. Linha (Amarelo)
2. Coluna (Azul)
3. Quadrado (Verde)



De forma a ser resolvido, todas as casas têm de estar preenchidas de 1 até ao valor N que representa o Sudoku. O tabuleiro representado em cima é um Sudoku 9x9, sendo os valores dos quadrados de 1 a 9, e pela mesma lógica se fosse um Sudoku 4x4 o valor de n era igual a 4 ($n=4$), sendo os valores representados de 1 a 4, e por aí em diante para 16x16, 25x25, etc. Existe ainda mais uma regra, um Sudoku só pode ser uma matriz quadrada, logo será sempre da forma $N \times N$ e nunca $N \times M$.

REPRESENTAÇÃO CSP DO SUDOKU

A representação tomada foi a F2CSP, sendo esta do género:

1. **Título:** que indica qual é o tipo de Sudoku (se é 4x4, 9x9, etc)
2. **Domínios das variáveis:** quais os valores que elas podem tomar
3. **Variáveis:** todas as variáveis do problema
4. **Restrições:** o número de restrições do problema
 - a. Restrições em detalhe: representação de todas as restrições entre as respetivas variáveis
 - b. Existe no fim as Restrições de aceitação, quando uma variável já tem à partida um valor fixo, estes valores são as pistas para a resolução do Sudoku
5. Acaba com o objectivo, que é satisfazer as restrições todas

As variáveis existem em relação às casas do Sudoku, sendo um par de Linha-Coluna, por exemplo a primeira posição será V1-1, a segunda posição da primeira linha na segunda coluna será V1-2, etc.

Exemplo:

V1-1	V1-2	V1-3	V1-4	V1-5	V1-6	V1-7	V1-8	V1-9
V2-1								
V3-1								
V4-1								
V5-1								
V6-1								
V7-1								
V8-1								
V9-1								V9-9

As restrições em detalhe são dadas na forma:

1. Número da Restrição
2. Variáveis envolvidas nesta restrição
3. Tipo de restrição
 - a. Reject: Valores impossíveis para cada par de variável que esteja na mesma linha, coluna e quadrado
 - b. Accept: No caso de ter sido dado uma pista, é atribuída a uma variável um valor

Existe mais uma particularidade no formato F2CSP que é o facto de a seguir a cada campo, Domínio, Variáveis, Restrições, existe o número das mesmas.

Domains:

1 (número de domínios)

...

... (representação de todos os domínios)

Variables:

16 (número de variáveis)

... (representação de todas as variáveis e respectivos domínios)

...

Constraints:


59 (número de restrições do puzzle)

Vars:

2 (número de variáveis)

Reject:

4 (número de tuplos rejeitados)



Informação das variáveis
afectas e tuplos rejeitados
em cada restrição

PROGRAMA QUE CRIA O FICHEIRO COM A REPRESENTAÇÃO F2CSP
DO SUDOKU

SudokuToF2CSP.py

Ao correr, deve-se escrever na consola o ficheiro de input com a representação correta (próxima página), premir **enter**, e escrever o nome do ficheiro de output que se quer.

REPRESENTAÇÃO DO INPUT

A representação do input é bastante simples, e é dada por linhas, e esta é a seguinte:

1º Linha: Número que representa o tamanho da matriz do Sudoku, ou seja o N Sudoku (N×N)

2º Linha: O número (**x**) de pistas que vão estar no tabuleiro do Sudoku

Restantes x linhas: As pistas com a posição e valor que vai estar nessa posição, e que vai ser associada a uma variável. Exemplo: 1 1 5 → 1 1 é posição, logo é a primeira casa do Sudoku e o 5 é o valor que vai estar representado nessa casa

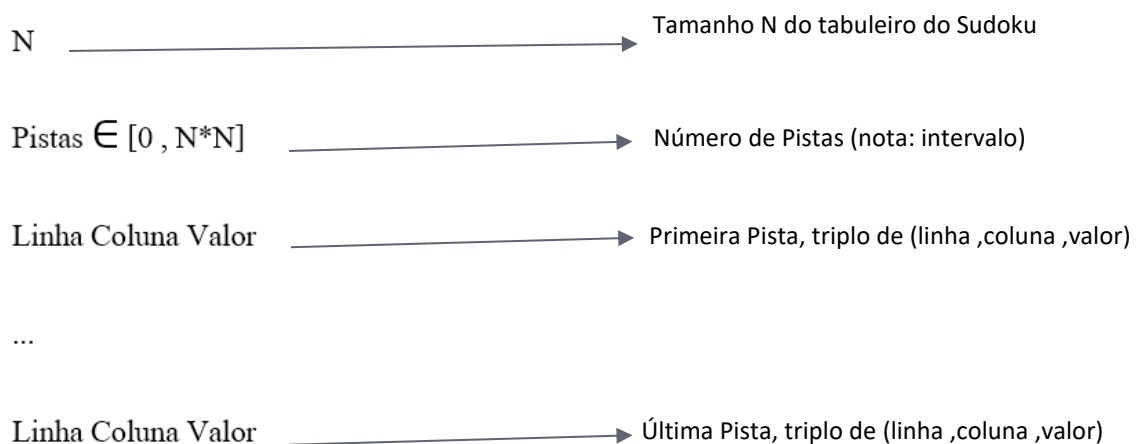
Notas:

O número de pistas não deve ser maior que (N×N). Caso não haja pistas é **obrigatório** referir zero pistas.

O valor da pista não pode ser maior que N, nem menor que 1. ($1 \leq \text{VALOR} \leq N$)

A posição da pista, X Y, não pode superar N, nem ser menor que 1. ($1 \leq X \leq N$) e ($1 \leq Y \leq N$)

EXEMPLO DE FICHEIRO DE INPUT



ALGORITMO PARA AS RESTRIÇÕES

Como foi dito no capítulo PUZZLE ESCOLHIDO: SUDOKU, as restrições são de linha, coluna e quadrado. Desta forma fizemos a leitura das restrições não ser mais excessiva do que ela já é, limitando o número de comparações que temos de fazer ao mínimo possível.

Vocabulário:

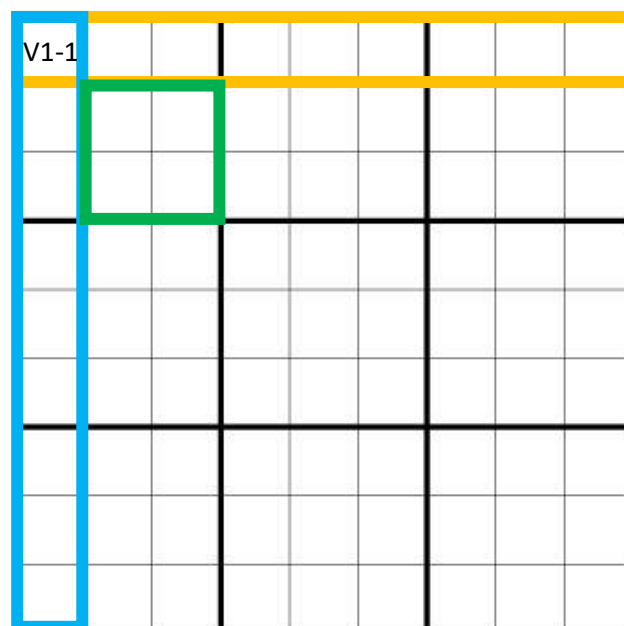
\neq : diferente

Sabemos da particularidade da lógica que se $Z \neq Y$, então $Y \neq Z$, desta forma sempre que fazemos uma restrição que implica $V1-1$ e $V1-2$, por exemplo, temos que

$V1-1 \neq V1-2$ logo não é necessário fazer $V1-2 \neq V1-1$. Isto leva-nos a nunca andar para trás no puzzle, limitando o número de comparações.

Exemplo: Para a variável $V1-1$ temos as seguintes restrições

- Todas as variáveis da linha são diferentes de $V1-1$
 - $V1-1 \neq V1-Y$, com $Y \in [1,N]$
- Todas as variáveis da coluna são diferentes de $V1-1$
 - $V1-1 \neq VX-1$, com $X \in [1,N]$
- Todas as variáveis do quadrado exceto as já verificadas nas linhas e colunas são diferentes de $V1-1$
 - $V1-1 \neq VX-Y$, com $X \in [2,\sqrt{N}]$ e $Y \in [2,\sqrt{N}]$



A verificação das linhas e colunas é tarefa fácil, um ciclo de 1 até N+1 resolve todas essas restrições, porém a verificação do subquadrado requer algumas verificações e a nossa solução passou por usar os restos de divisões que as linguagens de programação oferecem.

Para a restrição dos subquadrados temos para a verificação das linhas:

```
frontRows = row

while((frontRows % math.sqrt(self.n)) != 0 )
```

Que indica-nos quando estamos na última linha do quadrado e temos de parar as restrições, e basta apenas esta verificação porque pelo nosso método nunca é preciso voltar para trás no puzzle.

Para os lados temos as seguintes verificações:

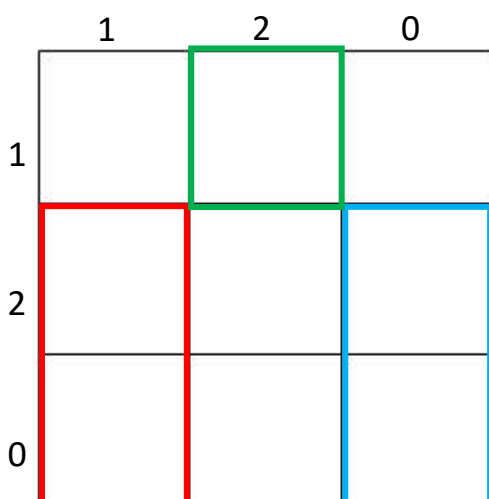
```
frontCols = col
backCols = col

while(backCols % math.sqrt(self.n) != 1)

.....
.....
    MAIS CÓDIGO
.....
.....

while(frontCols % math.sqrt(self.n) != 0)
```

Isto indica-nos o limite nos lados dos quadrados, pois o resto da divisão na última posição é 0 enquanto na primeira é 1.



Representado estão os restos das divisões por \sqrt{N} , que é o que permite o algoritmo não fazer mais comparações que as necessárias, isto é, ele nunca passa uma única vez por alguma casa já com restrições, logo não é necessário o uso de ifs no código e sim apenas os ciclos. Representado na figura a cores, a **Vermelho** as posições que faltam fazer restrições, com linha(s) à frente, e coluna(s) atrás da variável a **verde**, e a **azul** as posições que faltam fazer restrições, com linha(s) e coluna(s) à frente da variável **verde**. É um exemplo do algoritmo numa posição diferente da primeira, aplicando-se igual para as outras posições.

ESTATÍSTICAS DO FICHEIRO F2CSP

A leitura é feita pelos valores já explicados no capítulo REPRESENTAÇÃO CSP DO SUDOKU.

Vai-se lendo o ficheiro F2CSP dado pelo o outro programa, e sempre que achamos palavras chaves como “Domains”, “Variables”, “Constraints”, “Reject” e “Accept” somamos os valores na linha seguinte a um contador e ignoramos o número de linhas seguintes igual ao número lido.

Dominios: dado pelo número seguinte à palavra chave “Domains”

Variaveis: dado pelo número seguinte à palavra chave “Variables”

Restricoes: dado pelo número seguinte à palavra chave “Constraints”

Tuplos Aceites: dado pela soma de todos os números seguintes à palavra chave “Accept” em cada restrição que a contenha

Tuplos Rejeitados: dado pela soma de todos os números seguintes à palavra chave “Reject” em cada restrição que a contenha

PROGRAMA QUE DEVOLVE O FICHEIRO COM AS ESTATISTICAS DO FICHEIRO F2CSP

F2CSP_Stats.py

Ao correr, deve-se escrever na consola o ficheiro de input, este sendo um ficheiro de output do programa SudokuCSP.py, premir **enter**, e escrever o nome do ficheiro de output que se quer.