Lógica para Programação

Projecto

23 de Abril de 2021

Solucionador de Puzzles Kakuro

Conteúdo

O objetivo deste projecto é escrever um programa em PROLOG para resolver puzzles kakuro, de agora em diante designados apenas por "puzzles".

Um puzzle kakuro é constituído por uma grelha $n \times m$ (n linhas, m colunas). Cada posição da grelha pode estar vazia, ou conter uma barra "\", ladeada de zero, um ou dois inteiros. Na Figura 1 mostra-se um exemplo de um puzzle 5×5.

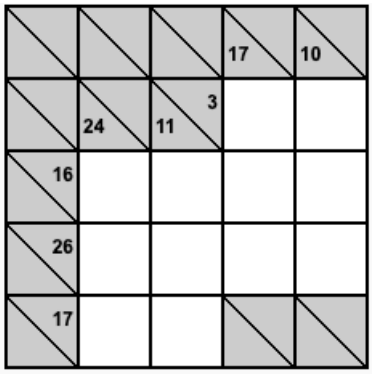

Figura 1: Puzzle de dimensão 5×5.

Designa-se por *espaço* uma sequência de posições vazias, na horizontal ou na vertical, de tamanho máximo. Por exemplo, no puzzle da Figura 1, a linha 3 tem um único espaço com 4 posições, e a coluna 2 tem um único espaço com 3 posições.

Para resolver um puzzle é necessário preencher as suas posições livres com inteiros entre um e nove, sem repetições, de forma a respeitar as restrições indicadas pelas posições que contenham uma barra. Uma posição que contenha "$s_v \backslash s_h$" implica que a soma dos inteiros no espaço vertical abaixo deve ser s_v, e que a soma dos inteiros no espaço horizontal à sua direita deve ser s_h. Assim, o puzzle da Figura 1 tem uma única solução, que é apresentada na Figura 2.

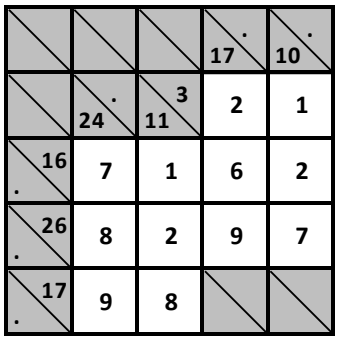

Figura 2: Solução do puzzle da Figura 1.

1 Representação de puzzles

Um puzzle é representado por uma lista de listas, correspondendo cada lista interior a uma linha do puzzle. Uma posição contendo "$s_v \backslash s_h$" é representada pela lista $[s_v, s_h]$. Se qualquer um dos inteiros s_v, s_h não existir, é representado por zero. Uma posição vazia é representada por uma variável. Por exemplo, o puzzle da Fig. 1 é representado por

```
[[[0, 0], [0, 0],  [0, 0], [17, 0], [10, 0]],
 [[0, 0], [24, 0], [11, 3],  P24,     P25],
 [[0,16],   P32,     P33,    P34,     P35],
 [[0,26],   P42,     P43,    P44,     P45],
 [[0,17],   P52,     P53,   [0,0],   [0,0]]]
```

2 Abordagem

Nesta secção apresentamos o algoritmo que o seu programa deve usar na resolução de puzzles.

2.1 Inicialização

O primeiro passo na resolução de um puzzle consiste na sua inicialização. Em alguns casos, este passo é suficiente para resolver o puzzle.

Para inicializar um puzzle, devem ser seguidos os seguintes passos:

1. Obter uma lista com os *espaços* existentes no puzzle. Cada espaço é representado por uma estrutura `espaço(soma, variáveis)`, em que `soma` é a soma das posições do espaço, e `variáveis` é a lista de variáveis do espaço. A lista deve estar ordenada de forma a apresentar os espaços da primeira para a última linha, e da esquerda para direita, seguidos dos espaços da primeira para a última coluna, e de cima para baixo. Por exemplo, para o puzzle da Fig. 1, esta lista seria

   ```
   [espaco(3,[P24,P25]),
    espaco(16,[P32,P33,P34,P35]),
    espaco(26,[P42,P43,P44,P45]),
    espaco(17,[P52,P53]),
    espaco(24,[P32,P42,P52]),
    espaco(11,[P33,P43,P53]),
    espaco(17,[P24,P34,P44]),
    espaco(10,[P25,P35,P45])]
   ```

2. Obter a lista de permutações de inteiros entre 1 e 9, possíveis para cada espaço. Esta lista será daqui em diante designada por *lista de permutações possíveis*. Uma permutação `Perm` é possível para um espaço `espaço(soma, variáveis)` se:

   - `Perm` e `variáveis` tiverem o mesmo comprimento.

- `Perm` respeitar os números que já estejam atribuídos a elementos de `variáveis`. Por exemplo, a permutação `[2, 1]` não é possível para o espaço `espaço(3, [Var1, 2])`.
- A colocação de `Perm` em `variáveis` não impossibilitar o preenchimento de outros espaços com posições em comum com `variáveis`. Por exemplo, considere-se a parte de um puzzle ilustrada na Figura 3

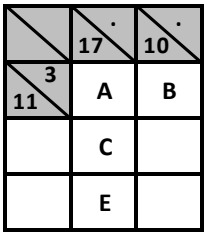

Figura 3: Puzzle parcial.

a permutação `[9, 6, 2]` não é possível para o espaço `espaço(17, [A, C, E])`, porque a posição `A` ficaria preenchida com `9`, e não existe nenhuma permutação para o espaço `espaço(3, [A, B])` que contenha um `9`.

O resultado deste passo consiste numa lista em que cada elemento é uma lista de 2 elementos: a lista de variáveis de um espaço e a lista (ordenada) de permutações possíveis para esse espaço. Por exemplo, para o puzzle da Fig. 4 a lista de permutações possíveis seria

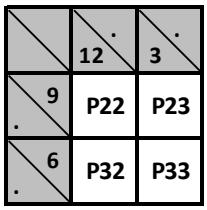

Figura 4: Puzzle de dimensão 3×3.

```
[[[P22,P23],[[7,2],[8,1]]],
 [[P32,P33],[[4,2],[5,1]]],
 [[P22,P32],[[7,5],[8,4]]],
 [[P23,P33],[[1,2],[2,1]]]]
```

3. Simplificar a lista de permutações possíveis. Para tal devem ser seguidos os seguintes passos:

   (a) Atribuir números comuns a todas as permutações possíveis. Se todas as permutações possíveis para um espaço tiverem um número em comum numa dada posição, essa posição do espaço deverá ser preenchida com esse número. Por exemplo, suponhamos que as permutações possíveis para o espaço `espaço(13, [X, Y, Z, W])` são `[[1, 2, 3, 7], [3, 2, 1, 7]]`; então o espaço deve ser actualizado para `[X, 2, Z, 7]`.

(b) Retirar permutações impossíveis: depois do passo anterior, pode acontecer que algumas listas de permutações possíveis para um espaço contenham permutações que deixaram de ser possíveis para esse espaço. Estas permutações devem ser retiradas da lista. Usando o exemplo da alínea anterior, suponhamos que a lista de permutações possíveis tem o seguinte elemento `[[T, Y, V], [[1, 2, 3], [1, 3, 2], [3, 2, 1]]]`. Ao ser atribuído o número 2 à variável `Y` no passo anterior, a permutação `[1, 3, 2]` deixou de ser possível, pelo que o elemento da lista de permutações deveria ser actualizado para `[[T, 2, V], [[1, 2, 3], [3, 2, 1]]]`.

Estes 2 passos devem ser repetidos até não haver nenhuma modificação na lista de permutações possíveis.

2.2 Resolução de listas de permutações possíveis

Se após a inicialização de um puzzle restarem posições por preencher, isto é, se ainda existirem espaços com variáveis, devem seguir-se os seguintes passos:

1. Identificar os espaços que tiverem listas de permutações possíveis com mais de uma permutação. De entre estes espaços escolher o primeiro que tenha associado um número mínimo de permutações possíveis. Por exemplo, suponhamos que tínhamos a seguinte lista de permutações possíveis:

   ```
   [[espaço1, [permutação11, permutação12, permutação13]],
    [espaço2, [permutação21, permutação22]],
    [espaço3, [permutação31]],
    [espaço4, [permutação41, permutação42]]]
   ```

   Os espaços com listas de permutações possíveis com mais de uma permutação são `espaço1, espaço2, espaço4`. Destes, `espaço2` e `espaço4` têm o número mínimo de permutações possíveis: dois. Logo, será escolhido o espaço `espaço2`.

2. Experimentar atribuir uma permutação ao espaço escolhido. Suponhamos que um dos elementos da lista de permutações possíveis era `[[1, 2, 5, X, Y],[[1, 2, 5, 7, 6], [1, 2, 5, 6, 7]]]`. Após a aplicação deste passo, este elemento seria substituído por `[[1, 2, 5, 7, 6],[[1, 2, 5, 7, 6]]]`.

3. Simplificar a lista de permutações possíveis, como indicado no passo 3, da Secção 2.1.

Estes 3 passos devem ser repetidos enquanto existirem espaços com um número de permutações possíveis superior a um. Se a atribuição de uma permutação a um espaço levar a uma situação impossível, retrocede-se até à última escolha. Note que o mecanismo de retrocesso do PROLOG faz precisamente isto.

2.3 Resolução de puzzles

Finalmente, e usando os algoritmos anteriormente descritos, para resolver um puzzle basta em primeiro lugar proceder à inicialização, obtendo a lista de permutações possíveis simplificada (ver Secção 2.1), e em seguida resolver esta lista (ver Secção 2.2).

3 Trabalho a desenvolver

Nesta secção são descritos os predicados que deve implementar no seu projecto. Estes serão os predicados avaliados e, consequentemente, devem respeitar escrupulosamente as especificações apresentadas. Para além dos predicados descritos, poderá implementar todos os predicados que julgar necessários.

Na implementação dos seus predicados, poderá usar os predicados fornecidos no ficheiro `codigo_comum.pl`. No ficheiro `puzzles_publicos.pl` encontram-se definidos alguns puzzles, alguns dos quais são usados nos exemplos desta Secção. Para usar estes ficheiros, deve colocar o comando `:- [codigo_comum, puzzles_publicos].` no início do ficheiro que contém o seu projecto.

3.1 Predicados para a inicialização de puzzles

Nesta secção são definidos os predicados necessários para a inicialização de um puzzle, tal como foi descrito na Secção 2.1.

3.1.1 Predicado `combinacoes_soma/4`

Implemente o predicado `combinacoes_soma/4` , tal que:

`combinacoes_soma(N, Els, Soma, Combs)`, em que `N` é um inteiro, `Els` é uma lista de inteiros, e `Soma` é um inteiro, significa que `Combs` é a lista ordenada cujos elementos são as combinações `N` a `N`, dos elementos de `Els` cuja soma é `Soma` .

Por exemplo,

```
?- combinacoes_soma(2, [1,2,3,4,5,6,7,8,9], 13, Combs).
Combs = [[4, 9], [5, 8], [6, 7]].
```

Sugestão: utilize o predicado `combinacao/2` definido no ficheiro `codigo_comum.pl`, tal que `combinacao(N, Els, Comb)` significa que `Comb` é uma combinação `N` a `N`, dos elementos de `Els` . Por exemplo,

```
?- combinacao(2, [1,2,3,4,5,6,7,8,9], Comb).
Comb = [1, 2] ;
Comb = [1, 3] ;
Comb = [1, 4]  .
```

3.1.2 Predicado `permutacoes_soma/4`

Implemente o predicado `permutacoes_soma/4`, tal que:

`permutacoes_soma(N, Els, Soma, Perms)`, em que `N` é um inteiro, `Els` é uma lista de inteiros, e `Soma` é um inteiro, significa que `Perms` é a lista ordenada cujos elementos são as permutações das combinações `N` a `N`, dos elementos de `Els` cuja soma é `Soma`.

Por exemplo,

```
?- permutacoes_soma(2, [1,2,3,4,5,6,7,8,9], 13, Perms).
Perms = [[4, 9], [5, 8], [6, 7], [7, 6], [8, 5], [9, 4]].
```

Sugestão: utilize o predicado pré-definido `permutation/2`, tal que `permutation(L1, L2)` significa que `L2` é uma permutação de `L1`. Por exemplo,

```
?- permutation([1,2,3], L).
L = [1, 2, 3] ;
L = [1, 3, 2] ;
L = [2, 1, 3] ;
L = [2, 3, 1] .
```

3.1.3 Predicado `espaco_fila/2`

Implemente o predicado `espaco_fila/2`, tal que:

`espaco_fila(Fila, Esp, H_V)`, em que `Fila` é uma fila (linha ou coluna) de um puzzle e `H_V` é um dos átomos `h` ou `v`, conforme se trate de uma fila horizontal ou vertical, respectivamente, significa que `Esp` é um espaço de `Fila`, tal como descrito na Secção 2.1, no passo 1.

Por exemplo,

```
?- Fila = [[5, 13], _, _, [0, 0], [6, 9], _, _, _],
   espaco_fila(Fila, Esp, h) .
Fila = [[5, 13], _322, _328, [0, 0], [6, 9], _370, _376, _382],
Esp = espaco(13, [_322, _328]) ;
Fila = [[5, 13], _322, _328, [0, 0], [6, 9], _370, _376, _382],
Esp = espaco(9, [_370, _376, _382])
false.

?- Fila = [[5, 13], _, _, [0, 0], [6, 9], _, _, _],
   espaco_fila(Fila, Esp, v) .
Fila = [[5, 13], _322, _328, [0, 0], [6, 9], _370, _376, _382],
Esp = espaco(5, [_322, _328]) ;
Fila = [[5, 13], _322, _328, [0, 0], [6, 9], _370, _376, _382],
```

```
Esp = espaco(6, [_370, _376, _382])
false.
```

3.1.4 Predicado `espacos_fila/2`

Implemente o predicado `espacos_fila/2` , tal que:

`espacos_fila(H_V, Fila, Espacos)`, em que `Fila` é uma fila (linha ou coluna) de uma grelha e e `H_V` é um dos átomos `h` ou `v`, significa que `Espacos` é a lista de todos os espaços de `Fila`, da esquerda para a direita.

Por exemplo,

```
?- Fila = [[5, 13], _, _, [0, 0], [6, 9], _, _, _],
   espacos_fila(h, Fila, Esps) .
Fila = [[5, 13], _328, _334, [0, 0], [6, 9], _376, _382, _388],
Esps = [espaco(13, [_328, _334]), espaco(9, [_376, _382, _388])].

?- Fila = [[5, 13], _, _, [0, 0], [6, 9], _, _, _],
   espacos_fila(v, Fila, Esps) .
Fila = [[5, 13], _328, _334, [0, 0], [6, 9], _376, _382, _388],
Esps = [espaco(5, [_328, _334]), espaco(6, [_376, _382, _388])].

?- Fila = [[5, 13],[0, 0], [6, 9]], espacos_fila(v, Fila, Esps) .
Fila = [[5, 13], [0, 0], [6, 9]],
Esps = [].
```

3.1.5 Predicado `espacos_puzzle/2`

Implemente o predicado `espacos_puzzle/2` , tal que:

`espacos_puzzle(Puzzle, Espacos)`, em que `Puzzle` é um puzzle, significa que `Espacos` é a lista de espaços de `Puzzle`, tal como descrito na Secção 2.1, no passo 1. Sugestão: use o predicado `mat_transposta`, definido no ficheiro `codigo_comum.pl`.

Por exemplo,

```
?- Puzzle = [[[0, 0], [0, 0],  [0, 0], [17, 0], [10, 0]],
             [[0, 0], [24, 0], [11, 3],  P24,     P25],
              [[0,16],   P32,    P33,    P34,     P35],
              [[0,26],   P42,    P43,    P44,     P45],
              [[0,17],  P52,     P53,   [0,0],   [0,0]]],
   espacos_puzzle(Puzzle, Espacos),
   Espacos = [E1, E2, E3, E4, E5, E6, E7, E8].
Puzzle = [[[0, 0], [0, 0], [0, 0],  ...,
Espacos = [espaco(3, [P24, P25]),  ...],
E1 = espaco(3, [P24, P25]),
```

```
E2 = espaco(16, [P32, P33, P34, P35]),
E3 = espaco(26, [P42, P43, P44, P45]),
E4 = espaco(17, [P52, P53]),
E5 = espaco(24, [P32, P42, P52]),
E6 = espaco(11, [P33, P43, P53]),
E7 = espaco(17, [P24, P34, P44]),
E8 = espaco(10, [P25, P35, P45]).
```

3.1.6 Predicado `espacos_com_posicoes_comuns/3`

Implemente o predicado `espacos_com_posicoes_comuns/3`, tal que:

`espacos_com_posicoes_comuns(Espacos, Esp, Esps_com)`, em que `Espacos` é uma lista de espaços e `Esp` é um espaço, significa que `Esps_com` é a lista de espaços com variáveis em comum com `Esp`, exceptuando `Esp`. Os espaços em `Esps_com` devem aparecer pela mesma ordem que aparecem em `Espacos`.

Por exemplo,

```
?- Puzzle = [[[0, 0], [0, 0],  [0, 0], [17, 0], [10, 0]],
             [[0, 0], [24, 0], [11, 3], P24,     P25],
             [[0,16],   P32,    P33,    P34,     P35],
             [[0,26],   P42,    P43,    P44,     P45],
             [[0,17],  P52,     P53,   [0,0],   [0,0]]],
   espacos_puzzle(Puzzle, Espacos),
   nth1(1, Espacos, Esp1),
   espacos_com_posicoes_comuns(Espacos, Esp1, Esps_com) .

                                  ...
Esp1 = espaco(3, [P24, P25]),
Esps_com = [espaco(17, [P24, P34, P44]), espaco(10, [P25, P35, P45])].
```

3.1.7 Predicado `permutacoes_soma_espacos/2`

Implemente o predicado `permutacoes_soma_espacos/2`, tal que:

`permutacoes_soma_espacos(Espacos, Perms_soma)`, em que `Espacos` é uma lista de espaços, significa que `Perms_soma` é a lista de listas de 2 elementos, em que o 1º elemento é um espaço de `Espacos` e o 2º é a lista ordenada de permutações cuja soma é igual à soma do espaço.

Por exemplo,

```
?- Puzzle = [[[0, 0], [0, 0],  [0, 0], [17, 0], [10, 0]],
             [[0, 0], [24, 0], [11, 3], P24,     P25],
             [[0,16],   P32,    P33,    P34,     P35],
             [[0,26],   P42,    P43,    P44,     P45],
```

```
            [[0,17],  P52,     P53,   [0,0],   [0,0]]],
   espacos_puzzle(Puzzle, Espacos),
   permutacoes_soma_espacos(Espacos, Perms_soma),
   Perms_soma = [PS1, PS2, PS3, PS4, PS5, PS6, PS7, PS8].
                                ...
PS1 = [espaco(3, [P24, P25]), [[1, 2], [2, 1]]],
PS2 = [espaco(16, [P32, P33, P34, P35]), [[1, 2, 4, 9], [1, 2, 5, 8], ...],
PS3 = [espaco(26, [P42, P43, P44, P45]), [[2, 7, 8, 9], [2, 7, 9, 8],...],
PS4 = [espaco(17, [P52, P53]), [[8, 9], [9, 8]]],
PS5 = [espaco(24, [P32, P42, P52]), [[7, 8, 9], [7, 9, 8], [8, 7, 9], ...],
PS6 = [espaco(11, [P33, P43, P53]), [[1, 2, 8], [1, 3, 7], [1, 4, 6], [...],
PS7 = [espaco(17, [P24, P34, P44]), [[1, 7, 9], [1, 9, 7], [2, 6, 9], ...],
PS8 = [espaco(10, [P25, P35, P45]), [[1, 2, 7], [1, 3, 6], [1, 4, 5], ...].
```

3.1.8 Predicado `permutacao_possivel_espaco/4`

Implemente o predicado `permutacao_possivel_espaco/4` , tal que:

`permutacao_possivel_espaco(Perm, Esp, Espacos, Perms_soma)`, em que `Perm` é uma permutação, `Esp` é um espaço, `Espacos` é uma lista de espaços, e `Perms_soma` é uma lista de listas tal como obtida pelo predicado anterior, significa que `Perm` é uma permutação possível para o espaço `Esp`, tal como descrito na Secção 2.1, no passo 2.

Por exemplo,

```
?- Puzzle = [[[0, 0], [0, 0],  [0, 0], [17, 0], [10, 0]],
             [[0, 0], [24, 0], [11, 3], P24,     P25],
             [[0,16],   P32,    P33,    P34,     P35],
             [[0,26],   P42,    P43,    P44,     P45],
             [[0,17],  P52,     P53,   [0,0],   [0,0]]],
   espacos_puzzle(Puzzle, Espacos),
   permutacoes_soma_espacos(Espacos, Perms_soma),
   nth1(4, Espacos, Esp),
   permutacao_possivel_espaco(Perm, Esp, Espacos, Perms_soma) .
                              ...
Perm = [9, 8].
```

3.1.9 Predicado `permutacoes_possiveis_espaco/4`

Implemente o predicado `permutacoes_possiveis_espaco(Espacos, Perms_soma, Esp, Perms_poss)/4`, tal que:

`permutacoes_possiveis_espaco(Espacos, Perms_soma, Esp, Perms_poss)`, em que `Espacos` é uma lista de espaços, `Perms_soma` é uma lista de listas tal como obtida pelo predicado `permutacoes_soma_espacos`, e `Esp` é um espaço, significa que `Perms_poss` é uma lista de 2 elementos em que o primeiro é a lista de variáveis de `Esp` e o segundo é a lista ordenada de permutações possíveis para o espaço `Esp`, tal como descrito na Secção 2.1, no passo 2.

Por exemplo,

```
?- puzzle_publico(2, Puzzle),
   espacos_puzzle(Puzzle, Espacos),
   permutacoes_soma_espacos(Espacos, Perms_soma),
   nth1(1, Espacos, Esp),
   permutacoes_possiveis_espaco(Espacos, Perms_soma, Esp, Perms_poss).
                              ...
Esp = espaco(3, [_234, _240]),
                              ...
Perms_poss = [[_234, _240], [[1, 2], [2, 1]]].

?- puzzle_publico(2, Puzzle),
   espacos_puzzle(Grelha, Espacos),
   permutacoes_soma_espacos(Espacos, Perms_soma),
   nth1(4, Espacos, Esp),
   permutacoes_possiveis_espaco(Espacos, Perms_soma, Esp, Perms_poss).
                              ...
Esp = espaco(17, [_366, _372]),
                              ...
Perms_poss = [[_366, _372], [[9, 8]]].
```

3.1.10 Predicado `permutacoes_possiveis_espacos/2`

Implemente o predicado `permutacoes_possiveis_espacos/2`, tal que:

`permutacoes_possiveis_espacos(Espacos, Perms_poss_esps)`, em que `Espacos` é uma lista de espaços, significa que `Perms_poss_esps` é a lista de permutações possíveis, tal como descrito na Secção 2.1, no passo 2.

Por exemplo,

```
?- puzzle_publico(2, Puzzle),
   espacos_puzzle(Puzzle, Espacos),
   permutacoes_possiveis_espacos(Espacos, Perms_poss_esps),
   maplist(writeln,  Perms_poss_esps).
[[_206,_212],[[1,2],[2,1]]]
[[_242,_248,_254,_260],[[7,1,2,6],[7,1,3,5],[7,1,5,3],[7,1,6,2], ...]]
[[_290,_296,_302,_308],[[7,4,9,6],[7,5,8,6],[7,6,8,5],[7,6,9,4], ...]]
[[_338,_344],[[9,8]]]
```

```
[[_242,_290,_338],[[7,8,9],[7,9,8],[8,7,9],[9,7,8]]]
[[_248,_296,_344],[[1,2,8]]]
[[_206,_254,_302],[[1,7,9],[1,9,7],[2,6,9],[2,7,8],[2,8,7],[2,9,6]]]
[[_212,_260,_308],[[1,2,7],[1,3,6],[1,4,5],[1,5,4],[1,6,3],[1,7,2], ...]]
                    ...
```

3.1.11 Predicado `numeros_comuns/2`

Implemente o predicado `numeros_comuns/2`, tal que:

`numeros_comuns(Lst_Perms, Numeros_comuns)`, em que `Lst_Perms` é uma lista de permutações, significa que `Numeros_comuns` é uma lista de pares `(pos, numero)`, significando que todas as listas de `Lst_Perms` contêm o número `numero` na posição `pos`.

Por exemplo,

```
?- numeros_comuns([[7,1,5,3],[7,5,1,3],[7,4,2,3]], Numeros_comuns) .
Numeros_comuns = [(1, 7),  (4, 3)].
?- numeros_comuns([[7,1,5,3],[7,5,1,3],[9,4,2,1]], Numeros_comuns) .
Numeros_comuns = [].
```

3.1.12 Predicado `atribui_comuns/1`

Implemente o predicado `atribui_comuns/1`, tal que:

`atribui_comuns(Perms_Possiveis)`, em que `Perms_Possiveis` é uma lista de permutações possíveis, actualiza esta lista atribuindo a cada espaço números comuns a todas as permutações possíveis para esse espaço, tal como descrito na Secção 2.1, no passo 3a.

Por exemplo,

```
?- puzzle_publico(2, Puzzle),
   espacos_puzzle(Puzzle, Espacos),
   permutacoes_possiveis_espacos(Espacos, Perms_poss_esps),
   atribui_comuns(Perms_poss_esps),
   maplist(writeln, Perms_poss_esps).
[[_206,_212],[[1,2],[2,1]]]
[[_242,1,_254,_260],[[7,1,2,6],[7,1,3,5],[7,1,5,3],[7,1,6,2], ...]]
[[_290,2,_302,_308],[[7,4,9,6],[7,5,8,6],[7,6,8,5],[7,6,9,4], ...]]
[[9,8],[[9,8]]]
[[_242,_290,9],[[7,8,9],[7,9,8],[8,7,9],[9,7,8]]]
[[1,2,8],[[1,2,8]]]
[[_206,_254,_302],[[1,7,9],[1,9,7],[2,6,9],[2,7,8],[2,8,7],[2,9,6]]]
```

```
[[_212,_260,_308],[[1,2,7],[1,3,6],[1,4,5],[1,5,4],[1,6,3],[1,7,2],...]]
                  ...
```

3.1.13 Predicado `retira_impossiveis/2`

Implemente o predicado `retira_impossiveis/2`, tal que:

`retira_impossiveis(Perms_Possiveis, Novas_Perms_Possiveis)`, em que `Perms_Possiveis` é uma lista de permutações possíveis, significa que `Novas_Perms_Possiveis` é o resultado de tirar permutações impossíveis de `Perms_Possiveis`, tal como descrito na Secção 2.1, no passo 3b.

Por exemplo,

```
?- puzzle_publico(2, Puzzle),
   espacos_puzzle(Puzzle, Espacos),
   permutacoes_possiveis_espacos(Espacos, Perms_Possiveis),
   atribui_comuns(Perms_Possiveis),
   retira_impossiveis(Perms_Possiveis, Novas_Perms_Possiveis),
   maplist(writeln, Novas_Perms_poss_esps).
[[_220,_226],[[1,2],[2,1]]]
[[_256,1,_268,_274],[[7,1,2,6],[7,1,3,5],[7,1,5,3],[7,1,6,2],
[8,1,2,5],[8,1,3,4],[8,1,4,3],[8,1,5,2],[9,1,2,4],[9,1,4,2]]]
[[_304,2,_316,_322],[[8,2,9,7],[9,2,8,7]]]
[[9,8],[[9,8]]]
[[_256,_304,9],[[7,8,9],[8,7,9]]]
[[1,2,8],[[1,2,8]]]
[[_220,_268,_316],[[1,7,9],[1,9,7],[2,6,9],[2,7,8],[2,8,7],[2,9,6]]]
[[_226,_274,_322],[[1,2,7],[1,3,6],[1,4,5],[1,5,4],[1,6,3],
[1,7,2],[2,1,7],[2,3,5],[2,5,3]]]
                  ...
```

3.1.14 Predicado `simplifica/2`

Implemente o predicado `simplifica/2` , tal que:

`simplifica(Perms_Possiveis, Novas_Perms_Possiveis)`, em que `Perms_Possiveis` é uma lista de permutações possíveis, significa que `Novas_Perms_Possiveis` é o resultado de simplificar `Perms_Possiveis` , tal como descrito na Secção 2.1, no passo 3. Para simplificar uma lista de permutações possíveis, deve aplicar-lhe os predicados `atribui_comuns` e `retira_impossiveis`, por esta ordem, até não haver mais alterações.

Por exemplo,

```
?- puzzle_publico(2, Puzzle),
   espacos_puzzle(Puzzle, Espacos),
   permutacoes_possiveis_espacos(Espacos, Perms_Possiveis),
   simplifica(Perms_Possiveis, Novas_Perms_Possiveis),
   maplist(writeln, Novas_Perms_Possiveis).
[[_220,_226],[[1,2],[2,1]]]
[[_256,1,_268,_274],[[7,1,2,6],[7,1,3,5],[7,1,5,3],[7,1,6,2],
    [8,1,2,5],[8,1,3,4], [8,1,4,3],[8,1,5,2],[9,1,2,4],[9,1,4,2]]]
[[_304,2,_316,7],[[8,2,9,7],[9,2,8,7]]]
[[9,8],[[9,8]]]
[[_256,_304,9],[[7,8,9],[8,7,9]]]
[[1,2,8],[[1,2,8]]]
[[_220,_268,_316],[[1,7,9],[1,9,7],[2,6,9],[2,7,8],[2,8,7],[2,9,6]]]
[[_226,_274,7],[[1,2,7],[2,1,7]]]

                    ...
```

3.1.15 Predicado `inicializa/2`

Implemente o predicado `inicializa/2` , tal que:

`inicializa(Puzzle, Perms_Possiveis)`, em que `Puzzle` é um puzzle, significa que `Perms_Possiveis` é a lista de permutações possíveis *simplificada* para `Puzzle`.

Na implementação deste predicado deverá usar os predicados indicados anteriormente, de forma a seguir os passos descritos na Secção 2.1. Por exemplo,

```
?- puzzle_publico(2, Puzzle),
   inicializa(Puzzle, Perms_Possiveis),
   maplist(writeln, Perms_Possiveis).
[[_192,_198],[[1,2],[2,1]]]
[[_228,1,_240,_246],[[7,1,2,6],[7,1,3,5],[7,1,5,3],[7,1,6,2],[8,1,2,5],
                     [8,1,3,4],[8,1,4,3],[8,1,5,2],[9,1,2,4],[9,1,4,2]]]
[[_276,2,_288,7],[[8,2,9,7],[9,2,8,7]]]
[[9,8],[[9,8]]]
[[_228,_276,9],[[7,8,9],[8,7,9]]]
[[1,2,8],[[1,2,8]]]
[[_192,_240,_288],[[1,7,9],[1,9,7],[2,6,9],[2,7,8],[2,8,7],[2,9,6]]]
[[_198,_246,7],[[1,2,7],[2,1,7]]]

                    ...
```

3.2 Predicados para a resolução de listas de permutações possíveis

Nesta secção são definidos os predicados necessários para a resolução de uma lista de permutações possíveis, tal como foi descrito na Secção 2.2.

3.2.1 Predicado `escolhe_menos_alternativas/2`

Implemente o predicado `escolhe_menos_alternativas/2`, tal que:

`escolhe_menos_alternativas(Perms_Possiveis, Escolha)`, em que `Perms_Possiveis` é uma lista de permutações possíveis, significa que `Escolha` é o elemento de `Perms_Possiveis` escolhido segundo o critério indicado na Secção 2.2, no passo 1. Se todos os espaços em `Perms_Possiveis` tiverem associadas listas de permutações unitárias, o predicado deve devolver "falso".

Por exemplo,

```
?-  puzzle_publico(2, Puzzle),
    inicializa(Puzzle, Perms_Possiveis),
    escolhe_menos_alternativas(Perms_Possiveis, Escolha).
Perms_Possiveis = [[[_206, _212], [[1, 2], [2, 1]]],
                   [[_242, 1, _254, _260], [[7, 1, 2, 6], [7, 1, 3|...],
                                       ...
Escolha = [[_206, _212], [[1, 2], [2, 1]]].

?- Perms_Possiveis = [[[3,6],[[3,6]]],
                      [[6,1],[[6,1]]],
                      [[3,6],[[3,6]]],
                      [[6,1],[[6,1]]]],
    escolhe_menos_alternativas(Perms_Possiveis, Escolha).
false.
```

3.2.2 Predicado `experimenta_perm/3`

Implemente o predicado `experimenta_perm/3` , tal que:

A chamada `experimenta_perm(Escolha, Perms_Possiveis, Novas_Perms_Possiveis)`, em que `Perms_Possiveis` é uma lista de permutações possíveis, e `Escolha` é um dos seus elementos (escolhido pelo predicado anterior), segue os seguintes passos:

1. Sendo `Esp` e `Lst_Perms` o espaço e a lista de permutações de `Escolha`, respectivamente, escolhe uma permutação de `Lst_Perms`, `Perm`. Utilize o predicado `member` para escolher esta permutação.
2. Unifica `Esp` com `Perm`.
3. `Novas_Perms_Possiveis` é o resultado de substituir, em `Perms_Possiveis`, o elemento `Escolha` pelo elemento `[Esp, [Perm]]`.

Por exemplo,

```
?- puzzle_publico(2, Puzzle),
    inicializa(Puzzle, Perms_Possiveis),
    escolhe_menos_alternativas(Perms_Possiveis, Escolha),
    experimenta_perm(Escolha, Perms_Possiveis, Novas_Perms_Possiveis),
    maplist(writeln, Novas_Perms_Possiveis).
[[1,2],[[1,2]]]
[[_256,1,_268,_274],[[7,1,2,6],[7,1,3,5],[7,1,5,3],[7,1,6,2],[8,1,2,5],
                     [8,1,3,4],[8,1,4,3],[8,1,5,2],[9,1,2,4],[9,1,4,2]]]
[[_304,2,_316,7],[[8,2,9,7],[9,2,8,7]]]
[[9,8],[[9,8]]]
[[_256,_304,9],[[7,8,9],[8,7,9]]]
[[1,2,8],[[1,2,8]]]
[[1,_268,_316],[[1,7,9],[1,9,7],[2,6,9],[2,7,8],[2,8,7],[2,9,6]]]
[[2,_274,7],[[1,2,7],[2,1,7]]]
```

3.2.3 Predicado `resolve_aux/2`

Implemente o predicado `resolve_aux/2`, tal que:

`resolve_aux(Perms_Possiveis, Novas_Perms_Possiveis)`, em que `Perms_Possiveis` é uma lista de permutações possíveis, significa que `Novas_Perms_Possiveis` é o resultado de aplicar o algoritmo descrito na Secção 2.2 a `Perms_Possiveis`.

Por exemplo,

```
?- puzzle_publico(2, Puzzle),
    inicializa(Puzzle, Perms_Possiveis),
    resolve_aux(Perms_Possiveis, Novas_Perms_Possiveis),
    maplist(writeln, Novas_Perms_Possiveis).
[[2,1],[[2,1]]]
```

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

3.3 Predicados para a resolução de puzzles

3.3.1 Predicado `resolve/1`

Implemente o predicado `resolve/1` , tal que:

`resolve(Puz)`, em que `Puz` é um puzzle, resolve esse puzzle, isto é, após a invocação deste predicado a grelha de `Puz` tem todas as variáveis substituídas por números que respeitam as restrições `Puz`.

Por exemplo,

```
?- puzzle_publico(2, Puzzle),
   writeln('Puzzle:'), escreve_Puzzle(Puzzle),
   resolve(Puzzle),
   writeln('Solucao:'), escreve_Puzzle(Puzzle).
Puzzle:
-------------------------
| \  | \  | \  |17\ |10\ |
-------------------------
| \  |24\ |11\3| ?  | ?  |
-------------------------
| \16| ?  | ?  | ?  | ?  |
-------------------------
| \26| ?  | ?  | ?  | ?  |
-------------------------
| \17| ?  | ?  | \  | \  |
-------------------------
Solucao:
-------------------------
| \  | \  | \  |17\ |10\ |
-------------------------
| \  |24\ |11\3| 2  | 1  |
-------------------------
| \16| 7  | 1  | 6  | 2  |
-------------------------
| \26| 8  | 2  | 9  | 7  |
-------------------------
| \17| 9  | 8  | \  | \  |
```

```
-------------------------
```

O predicado `escreve_Puzzle/1` está definido no ficheiro `codigo_comum.pl`.

4 Avaliação

A nota do projecto será baseada nos seguintes aspectos:

- Execução correcta (80% - 16 val.). Estes 16 valores serão distribuídos da seguinte forma:

combinacoes_soma	0.5
permutacoes_soma	0.5
espaco_fila	1
espacos_fila	0.75
espacos_puzzle	0.75
espacos_com_posicoes_comuns	0.75
permutacoes_soma_espacos	0.75
permutacao_possivel_espaco	1
permutacoes_possiveis_espaco	0.75
permutacoes_possiveis_espacos	0.75
numeros_comuns	0.75
atribui_comuns	0.75
retira_impossiveis	1
simplifica	0.75
inicializa	0.5
escolhe_menos_alternativas	1
experimenta_perm	1.25
resolve_aux	1.5
resolve	1

- Qualidade do código, a qual inclui abstracção relativa aos predicados implementados, nomes escolhidos, paragrafação e qualidade dos comentários (20% - 4 val.).

5 Penalizações

- Caracteres acentuados, cedilhas e outros semelhantes: 3 val.
- Presença de *warnings*: 2 val.

6 Condições de realização e prazos

O projecto deve ser realizado individualmente.

O código do projecto deve ser entregue obrigatoriamente por via electrónica até às **23:59 do dia 17 de Maio** de 2021, através do sistema Mooshak. Depois desta hora, não serão aceites projectos sob pretexto algum.[1]

Deverá ser submetido um ficheiro .pl contendo o código do seu projecto. O ficheiro de código deve conter em comentário, na primeira linha, o número e o nome do aluno.

No seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer caractere que não pertença à tabela ASCII, sob pena de falhar todos os testes automáticos. Isto inclui comentários e cadeias de caracteres.

É prática comum a escrita de mensagens para o ecrã, quando se está a implementar e a testar o código. No entanto, **não se esqueçam de remover/comentar as mensagens escritas no ecrã na versão final** do código entregue. Se não o fizerem, correm o risco dos testes automáticos falharem, e irão ter uma má nota na execução.

A avaliação da execução do código do projecto será feita automaticamente através do sistema Mooshak, usando vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. Só poderá efectuar uma nova submissão pelo menos 15 minutos depois da submissão anterior.[2] Só são permitidas 10 submissões em simultâneo no sistema, pelo que uma submissão poderá ser recusada se este limite for excedido. Nesse caso tente mais tarde.

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais. O facto de um projecto completar com sucesso os exemplos fornecidos não implica, pois, que esse projecto esteja totalmente correcto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada aluno garantir que o código produzido está correcto.

Até duas semanas antes do prazo da entrega, serão publicadas na página da cadeira as instruções necessárias para a submissão do código no Mooshak. Apenas a partir dessa altura será possível a submissão por via electrónica. Até ao prazo de entrega poderá efectuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efectuada. Deverão portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projecto que pretendem que seja avaliada. Não serão abertas excepções.

Pode ou não haver uma discussão oral do projecto e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

7 Cópias

Projectos iguais, ou muito semelhantes, originarão a reprovação na disciplina e, eventualmente, o levantamento de um processo disciplinar. Os programas entregues serão testados em relação a soluções existentes na web. As analogias encontradas com os programas da web serão tratadas como cópias. O corpo docente da disciplina será o único

[1] Note que o limite de 10 submissões simultâneas no sistema Mooshak implica que, caso haja um número elevado de tentativas de submissão sobre o prazo de entrega, alguns alunos poderão não conseguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.

[2] Note que, se efectuar uma submissão no Mooshak a menos de 15 minutos do prazo de entrega, fica impossibilitado de efectuar qualquer outra submissão posterior.

juiz do que se considera ou não copiar num projecto.

8 Recomendações

- Recomenda-se o uso do SWI PROLOG, dado que este vai ser usado para a avaliação do projecto.
- Durante o desenvolvimento do programa é importante não se esquecer da Lei de Murphy:
  - Todos os problemas são mais difíceis do que parecem;
  - Tudo demora mais tempo do que nós pensamos;
  - Se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis.