INSTITUTO SUPERIOR TÉCNICO

Lógica para Programação

Projecto

10 de Janeiro de 2022

Solucionador de Puzzles Hashi (Parte I)

Conteúdo

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

Um puzzle hashi é constituído por uma grelha $n \times m$ (n linhas, m colunas). Cada posição da grelha pode estar vazia, ou conter uma *ilha*, com a indicação do número de pontes que essa ilha deverá ter, na solução do puzzle. Na Figura 1 mostra-se um exemplo de um puzzle 7×7, com 9 ilhas.

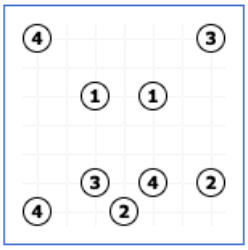


Figura 1: Puzzle de dimensão 7×7.

Na primeira linha deste puzzle existem 2 ilhas: uma na posição (1, 1), com a indicação de 4 pontes, e outra na posição (1, 7), com a indicação de 3 pontes.

Para obter uma solução, as ilhas devem ser ligadas por pontes, de forma a respeitar o número de pontes indicado por cada ilha e as seguintes restrições:

- Não há mais do que duas pontes entre quaisquer duas ilhas.
- As pontes só podem ser verticais ou horizontais e não podem cruzar ilhas ou outras pontes.
- Na solução do puzzle, as pontes permitem a passagem entre quaisquer duas ilhas.

Assim, o puzzle da Figura 1 tem uma única solução, que é apresentada na Figura 2.

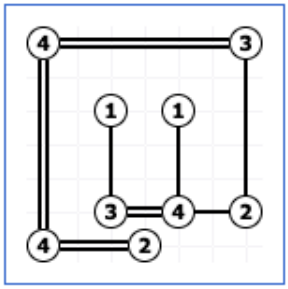


Figura 2: Solução do puzzle da Figura 1.

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 uma ilha é representada por um inteiro positivo, correspondendo ao número de pontes dessa ilha. As posições vazias são representadas por zero. Por exemplo, o puzzle da Figura 1 é representado por

```
[[4, 0, 0, 0, 0, 0, 3],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0],
 [0, 0, 3, 0, 4, 0, 2],
 [4, 0, 0, 2, 0, 0, 0]]
```

2 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.

Os predicados fornecidos no ficheiro `codigo_comum.pl` poderão ser-lhe úteis durante o desenvolvimento do projeto. 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.

2.1 Predicado `extrai_ilhas_linha/3`

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

`extrai_ilhas_linha(N_L, Linha, Ilhas)`, em que `N_L` é um inteiro positivo, correspondente ao número de uma linha e `Linha` é uma lista correspondente a uma linha de um puzzle, significa que `Ilhas` é a lista ordenada (ilhas da esquerda para a direita) cujos elementos são as ilhas da linha `Linha`.

Uma ilha é representada por uma estrutura `ilha(N, (Linha, Coluna))`, em que `N` representa o número de pontes da ilha, e `(Linha, Coluna)` representa a posição da ilha. Por exemplo,

```
?- extrai_ilhas_linha(1, [4, 0, 0, 0, 0, 0, 3], Ilhas).
Ilhas = [ilha(4,  (1, 1)), ilha(3,  (1, 7))].

?- extrai_ilhas_linha(7, [4, 0, 0, 2, 0, 0, 0] , Ilhas).
Ilhas = [ilha(4,  (7, 1)), ilha(2,  (7, 4))].
```

2.2 Predicado `ilhas/2`

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

`ilhas(Puz, Ilhas)`, em que `Puz` é um puzzle, significa que `Ilhas` é a lista ordenada (ilhas da esquerda para a direita e de cima para baixo) cujos elementos são as ilhas de `Puz`.

Uma ilha é representada por uma estrutura `ilha(N, (Linha, Coluna))`, em que `N` representa o número de pontes da ilha, e `(Linha, Coluna)` representa a posição da ilha. Por exemplo,

```
?- puzzle_publico(2, Puz), ilhas(Puz, Ilhas), maplist(writeln, Ilhas).
ilha(4,(1,1))
ilha(3,(1,7))
ilha(1,(3,3))
ilha(1,(3,5))
ilha(3,(6,3))
ilha(4,(6,5))
ilha(2,(6,7))
ilha(4,(7,1))
ilha(2,(7,4))
Puz = [[4, 0, 0, 0, 0, 0, 3],
                ...
```

Considere agora o conceito de "ilhas vizinhas". Duas ilhas dizem-se vizinhas sse:

- se encontrarem na mesma linha ou mesma coluna;
- entre elas não existir outra ilha;
- entre elas não existir uma ponte que una outras duas ilhas quaisquer.

2.3 Predicado `vizinhas/3`

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

`vizinhas(Ilhas, Ilha, Vizinhas)`, em que `Ilhas` é a lista de ilhas de um puzzle e `Ilha` é uma dessas ilhas, significa que `Vizinhas` é a lista ordenada (ilhas de cima para baixo e da esquerda para a direita) cujos elementos são as ilhas vizinhas de `Ilha`.

Por exemplo, sendo `Puz` em `puzzle_publico(2, Puz)` o puzzle representado na Secção 1:

```
?- puzzle_publico(2, Puz), ilhas(Puz, Ilhas),
   vizinhas(Ilhas, ilha(4,(1,1)), Vizinhas) .
                ...
Vizinhas = [ilha(3,  (1, 7)), ilha(4,  (7, 1))].
```

```
?- puzzle_publico(2, Puz), ilhas(Puz, Ilhas),
   vizinhas(Ilhas, ilha(4,(6,5)), Vizinhas).
                ...
Vizinhas = [ilha(1,  (3, 5)), ilha(3,  (6, 3)), ilha(2,  (6, 7))].
```

2.4 Predicado `estado/2`

Para representar o estado de um puzzle durante a sua resolução é utilizada uma lista de entradas. Uma *entrada* é uma lista em que:

- o 1º elemento é uma ilha.
- o 2º elemento é lista de vizinhas dessa ilha.
- o 3º elemento é a lista de pontes da ilha; esta lista é vazia no estado inicial.

Um *estado* é uma lista de entradas.

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

`estado(Ilhas, Estado)`, em que `Ilhas` é a lista de ilhas de um puzzle, significa que `Estado` é a lista ordenada cujos elementos são as entradas referentes a cada uma das ilhas de `Ilhas`.

Por exemplo,

```
?- puzzle_publico(2, Puz), ilhas(Puz, Ilhas), estado(Ilhas, Estado),
   maplist(writeln, Estado).
[ilha(4,(1,1)),[ilha(3,(1,7)),ilha(4,(7,1))],[]]
[ilha(3,(1,7)),[ilha(4,(1,1)),ilha(2,(6,7))],[]]
[ilha(1,(3,3)),[ilha(1,(3,5)),ilha(3,(6,3))],[]]
[ilha(1,(3,5)),[ilha(1,(3,3)),ilha(4,(6,5))],[]]
[ilha(3,(6,3)),[ilha(1,(3,3)),ilha(4,(6,5))],[]]
[ilha(4,(6,5)),[ilha(1,(3,5)),ilha(3,(6,3)),ilha(2,(6,7))],[]]
[ilha(2,(6,7)),[ilha(3,(1,7)),ilha(4,(6,5))],[]]
[ilha(4,(7,1)),[ilha(4,(1,1)),ilha(2,(7,4))],[]]
[ilha(2,(7,4)),[ilha(4,(7,1))],[]]
                ...
```

2.5 Predicado `posicoes_entre/3`

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

`posicoes_entre(Pos1, Pos2, Posicoes)`, em que `Pos1` e `Pos2` são posições, significa que `Posicoes` é a lista ordenada de posições entre `Pos1` e `Pos2` (excluindo `Pos1` e `Pos2`). Se `Pos1` e `Pos2` não pertencerem à mesma linha ou à mesma coluna, o resultado é `false`.

Por exemplo,

```
?- posicoes_entre((1,1), (1,5), Posicoes).
Posicoes = [(1, 2),  (1, 3),  (1, 4)].

?- posicoes_entre((2,2), (5,2), Posicoes).
Posicoes = [(3, 2),  (4, 2)].

?- posicoes_entre((5,2), (2,2), Posicoes).
Posicoes = [ (3, 2), (4, 2)].

?- posicoes_entre((2,2), (5,3), Posicoes).
false.
```

2.6 Predicado `cria_ponte/3`

Uma ponte entre duas ilhas de posições `Pos1` e `Pos2` é representada pela estrutura `ponte(Pos1, Pos2)`, em que `Pos1` e `Pos2` estão ordenadas. Implemente o predicado `cria_ponte/3`, tal que:

`cria_ponte(Pos1, Pos2, Ponte)`, em que `Pos1` e `Pos2` são 2 posições, significa que `Ponte` é uma ponte entre essas 2 posições.

Por exemplo,

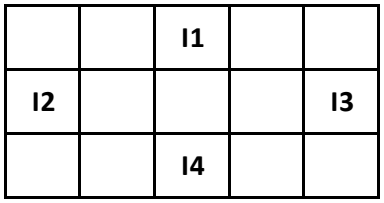
```
?- cria_ponte((1,1), (1,7), Ponte).
Ponte = ponte((1, 1),  (1, 7)).

?- cria_ponte((1,7), (1,1), Ponte).
Ponte = ponte((1, 1),  (1, 7)).
```

2.7 Predicado `caminho_livre/5`

Implemente o predicado `caminho_livre/5`, tal que:

`caminho_livre(Pos1, Pos2, Posicoes, I, Vz)`, em que `Pos1` e `Pos2` são posições, `Posicoes` é a lista ordenada de posições entre `Pos1` e `Pos2`, `I` é uma ilha, e `Vz` é uma das suas vizinhas, significa que a adição da ponte `ponte(Pos1, Pos2)` não faz com que `I` e `Vz` deixem de ser vizinhas.

Note que 2 ilhas vizinhas deixam de o ser, após a adição de uma ponte, se esta não for entre as duas ilhas, e ocupar alguma das posições entre as 2 ilhas. Por exemplo, em relação ao puzzle parcial abaixo, a adição de uma ponte entre as ilhas `I2` e `I3`, faz com que `I1` e `I4` deixem de ser vizinhas, mas `I2` e `I3` continuam a ser vizinhas.



Por exemplo,

```
?- caminho_livre((2,1), (2,5), [(2, 2),  (2, 3),  (2, 4)],
   ilha(_, (1,3)), ilha(_, (3,3))).
false.

?- caminho_livre((2,1), (2,5), [(2, 2),  (2, 3),  (2, 4)],
   ilha(_, (2,1)), ilha(_, (2,5))).
true.
```

2.8 Predicado `actualiza_vizinhas_entrada/5`

Implemente o predicado `actualiza_vizinhas_entrada/5`, tal que:

`actualiza_vizinhas_entrada(Pos1, Pos2, Posicoes, Entrada, Nova_Entrada)`, em que `Pos1` e `Pos2` são as posições entre as quais irá ser adicionada uma ponte, `Posicoes` é a lista ordenada de posições entre `Pos1` e `Pos2`, e `Entrada` é uma entrada (ver Secção 2.4), significa que `Nova_Entrada` é igual a `Entrada`, excepto no que diz respeito à lista de ilhas vizinhas; esta deve ser actualizada, removendo as ilhas que deixaram de ser vizinhas, após a adição da ponte.

Por exemplo, em relação ao puzzle da figura da secção anterior,

```
?- actualiza_vizinhas_entrada((2,1), (2,5), [(2, 2),  (2, 3),  (2, 4)],
        [ilha(2,  (1, 3)), [ilha(2,  (3, 3))], [], Nova_Entrada).
                                ...
 Nova_Entrada = [ilha(2,  (1, 3)), [], []].

?- actualiza_vizinhas_entrada((2,1), (2,5), [(2, 2),  (2, 3),  (2, 4)],
        [ilha(1,  (2, 1)), [ilha(2,  (2, 5))], [], Nova_Entrada).
                                ...
Nova_Entrada = [ilha(1,  (2, 1)), [ilha(2,  (2, 5))], []].
```

2.9 Predicado `actualiza_vizinhas_apos_pontes/4`

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

`actualiza_vizinhas_apos_pontes(Estado, Pos1, Pos2, Novo_estado)` , em que `Estado` é um estado (ver Secção 2.4), `Pos1` e `Pos2` são as posições entre as quais foi adicionada uma ponte, significa que `Novo_estado` é o estado que se obtém de `Estado` após a actualização das ilhas vizinhas de cada uma das suas entradas.

Por exemplo, em relação ao puzzle da figura da Secção anterior,

```
?- Puz = [[0,0,2,0,0], [1,0,0,0,2], [0,0,2,0,0]],
   ilhas(Puz, Ilhas), estado(Ilhas, Estado), maplist(writeln, Estado),
   actualiza_vizinhas_apos_pontes(Estado,(2,1), (2,5), Novo_estado),
   nl, maplist(writeln, Novo_estado).
[ilha(2,(1,3)),[ilha(2,(3,3))],[]]
[ilha(1,(2,1)),[ilha(2,(2,5))],[]]
[ilha(2,(2,5)),[ilha(1,(2,1))],[]]
[ilha(2,(3,3)),[ilha(2,(1,3))],[]]

[ilha(2,(1,3)),[],[]]
[ilha(1,(2,1)),[ilha(2,(2,5))],[]]
[ilha(2,(2,5)),[ilha(1,(2,1))],[]]
[ilha(2,(3,3)),[],[]]
                                  ...
```

2.10 Predicado `ilhas_terminadas/2`

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

`ilhas_terminadas(Estado, Ilhas_term)`, em que `Estado` é um estado (ver Secção 2.4), significa que `Ilhas_term` é a lista de ilhas que já têm todas as pontes associadas, designadas por *ilhas terminadas*. Se a entrada referente a uma ilha for `[ilha(N_pontes, Pos), Vizinhas, Pontes]`, esta ilha está terminada se `N_pontes` for diferente de `'X'` (a razão para esta condição ficará aparente mais à frente) e o comprimento da lista `Pontes` for `N_pontes`.

Por exemplo,

```
?- Estado = [[ilha(3,_),_,[_, _, _]],
             [ilha(2,_),_,[_, _]],
             [ilha(4,_),_,[_,_]],
             [ilha('X',_),_,[_]]],
   ilhas_terminadas(Estado, Ilhas_term).
                                  ...
Ilhas_term = [ilha(3, _2318), ilha(2, _2306)].
```

2.11 Predicado `tira_ilhas_terminadas_entrada/3`

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

`tira_ilhas_terminadas_entrada(Ilhas_term, Entrada, Nova_entrada)`, em que `Ilhas_term` é uma lista de ilhas terminadas e `Entrada` é uma entrada (ver Secção 2.4), significa que `Nova_entrada` é a entrada resultante de remover as ilhas de `Ilhas_term`, da lista de ilhas vizinhas de entrada.

Por exemplo,

```
?- Entrada = [ilha(4,(6,5)),[ilha(1,(3,5)),ilha(3,(6,3)),ilha(2,(6,7))],[]],
   Ilhas_term = [ilha(1,(3,5)), ilha(2,(6,7))],
   tira_ilhas_terminadas_entrada(Ilhas_term, Entrada, Nova_Entrada).
                                    ...
Nova_Entrada = [ilha(4,  (6, 5)), [ilha(3,  (6, 3))], []].
```

2.12 Predicado `tira_ilhas_terminadas/3`

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

`tira_ilhas_terminadas(Estado, Ilhas_term, Novo_estado)`, em que `Estado` é um estado (ver Secção 2.4) e `Ilhas_term` é uma lista de ilhas terminadas, significa que `Novo_estado` é o estado resultante de aplicar o predicado `tira_ilhas_terminadas_entrada` a cada uma das entradas de `Estado`.

Por exemplo,

```
?- Estado = [[ilha(1,(1,1)),[ilha(4,(1,3))],[_]],
[ilha(4,(1,3)),[ilha(1,(1,1)),ilha(1,(1,5)),ilha(2,(3,3))],[_, _, _, _]],
[ilha(1,(1,5)),[ilha(4,(1,3))],[]],
[ilha('X',(3,3)),[ilha(4,(1,3))],[_, _]]],
Ilhas_term = [ilha(1,(1,1)),ilha(4,(1,3))],
tira_ilhas_terminadas(Estado, Ilhas_term, NovoEstado).
...
NovoEstado = [[ilha(1,(1,1)),[],[_8392]],
[ilha(4,(1,3)),[ilha(1,(1,5)),ilha(2,(3,3))],[_8488,_8494,_8500,_8506]],
[ilha(1,(1,5)),[],[]],
[ilha('X',(3,3)),[],[_8620,_8626]]].
```

2.13 Predicado `marca_ilhas_terminadas_entrada/3`

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

`marca_ilhas_terminadas_entrada(Ilhas_term, Entrada, Nova_entrada)`, em que `Ilhas_term` é uma lista de ilhas terminadas e `Entrada` é uma entrada (ver Secção 2.4), significa que `Nova_entrada` é a entrada obtida de `Entrada` da seguinte forma: se a ilha de `Entrada` pertencer a `Ilhas_term`, o número de pontes desta é substituído por `'X'`; em caso contrário `Nova_entrada` é igual a `Entrada`.

Por exemplo,

```
?- Entrada = [ilha(4,(6,5)),[ilha(1,(3,5)),ilha(3,(6,3)),ilha(2,(6,7))],[]],
   Ilhas_term = [ilha(4,(6,5)), ilha(2,(6,7))],
   marca_ilhas_terminadas_entrada(Ilhas_term, Entrada, Nova_entrada).
                                    ...
Nova_entrada =
[ilha('X',  (6, 5)),
 [ilha(1,  (3, 5)), ilha(3,  (6, 3)), ilha(2,  (6, 7))], []].

?- Entrada = [ilha(4,(6,5)),[ilha(1,(3,5)),ilha(3,(6,3)),ilha(2,(6,7))],[]],
   Ilhas_term = [ilha(2,(6,7))],
   marca_ilhas_terminadas_entrada(Ilhas_term, Entrada, Nova_entrada).
Entrada = Nova_entrada,
Nova_entrada =
[ilha(4,  (6, 5)),
 [ilha(1,  (3, 5)), ilha(3,  (6, 3)), ilha(2,  (6, 7))], [],
                                    ...
```

2.14 Predicado `marca_ilhas_terminadas/3`

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

`marca_ilhas_terminadas(Estado, Ilhas_term, Novo_estado)`, em que `Estado` é um estado (ver Secção 2.4) e `Ilhas_term` é uma lista de ilhas terminadas, significa que `Novo_estado` é o estado resultante de aplicar o predicado `marca_ilhas_terminadas_entrada` a cada uma das entradas de `Estado`.

Por exemplo,

```
?- Estado = [[ilha(1,(1,1)),[ilha(4,(1,3))],[_]],
[ilha(4,(1,3)),[ilha(1,(1,1)),ilha(1,(1,5)),ilha(2,(3,3))],[_, _, _, _]],
[ilha(1,(1,5)),[ilha(4,(1,3))],[]],
[ilha('X',(3,3)),[ilha(4,(1,3))],[_, _]]],
Ilhas_term = [ilha(1,(1,1)),ilha(4,(1,3))],
marca_ilhas_terminadas(Estado, Ilhas_term, NovoEstado).
...
NovoEstado = [[ilha('X',(1,1)),[ilha(4,(1,3))],[_13874]],
[ilha('X',(1,3)),[ilha(1,(1,1)),ilha(1,(1,5)),ilha(2,(3,3))],
```

```
     [_13970,_13976,_13982,_13988]],
[ilha(1,(1,5)),[ilha(4,(1,3))],[]],
[ilha('X',(3,3)),[ilha(4,(1,3))],[_14102,_14108]]].
```

2.15 Predicado `trata_ilhas_terminadas/2`

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

`trata_ilhas_terminadas(Estado, Novo_estado)`, em que `Estado` é um estado (ver Secção 2.4), significa que `Novo_estado` é o estado resultante de aplicar os predicados `tira_ilhas_terminadas` e `marca_ilhas_terminadas` a `Estado`.

Por exemplo,

```
Estado = [[ilha(1,(1,1)),[ilha(4,(1,3))],[_]],
[ilha(4,(1,3)),[ilha(1,(1,1)),ilha(1,(1,5)),ilha(2,(3,3))],[_, _, _, _]],
[ilha(1,(1,5)),[ilha(4,(1,3))],[]],
[ilha('X',(3,3)),[ilha(4,(1,3))],[_, _]]],
trata_ilhas_terminadas(Estado, Novo_estado).
...
Novo_estado = [[ilha('X',(1,1)),[],[_19236]],
[ilha('X',(1,3)),[ilha(1,(1,5)),ilha(2,(3,3))],
      [_19332,_19338,_19344,_19350]],
[ilha(1,(1,5)),[],[]],
[ilha('X',(3,3)),[],[_19464,_19470]]].
```

2.16 Predicado `junta_pontes/5`

Implemente o predicado `junta_pontes/5`, tal que:

`junta_pontes(Estado, Num_pontes, Ilha1, Ilha2, Novo_estado)`, em que `Estado` é um estado e `Ilha1` e `Ilha2` são 2 ilhas, significa que `Novo_estado` é o estado que se obtém de `Estado` por adição de `Num_pontes` pontes entre `Ilha1` e `Ilha2`.

Este predicado executa os seguintes passos:

- Cria a(s) ponte(s) entre `Ilha1` e `Ilha2`.
- Adiciona as novas pontes às entradas de `Estado` correspondentes a estas ilhas.
- Actualiza o estado por aplicação dos predicados `actualiza_vizinhas_apos_pontes` e `trata_ilhas_terminadas`.

Exemplo 1:

```
?- puzzle_publico(3, Puz), ilhas(Puz, Ilhas), estado(Ilhas, Estado),
    junta_pontes(Estado, 1, ilha(2,(4,2)), ilha(2,(6,2)), N_Estado),
    nl, writeln('Diferencas:'), escreve_diferencas(Estado, N_Estado),
    escreve_Puzzle(Puz, N_Estado).

Diferencas:
[ilha(2,(4,2)),[ilha(6,(4,5)),ilha(2,(6,2))],[]]
passou a
[ilha(2,(4,2)),[ilha(6,(4,5)),ilha(2,(6,2))],[ponte((4,2),(6,2))]]

[ilha(3,(5,1)),[ilha(6,(3,1)),ilha(1,(5,3)),ilha(1,(7,1))],[]]
passou a
[ilha(3,(5,1)),[ilha(6,(3,1)),ilha(1,(7,1))],[]]

[ilha(1,(5,3)),[ilha(3,(3,3)),ilha(3,(5,1)),ilha(2,(5,6)),ilha(3,(7,3))],[]]
passou a
[ilha(1,(5,3)),[ilha(3,(3,3)),ilha(2,(5,6)),ilha(3,(7,3))],[]]

[ilha(2,(6,2)),[ilha(2,(4,2)),ilha(2,(8,2))],[]]
passou a
[ilha(2,(6,2)),[ilha(2,(4,2)),ilha(2,(8,2))],[ponte((4,2),(6,2))]]

 2     2     5     2
                1     3
 6     3
    2        6     1
 3  |  1        2     6
    2
 1     3     5        3
    2     3        2
                                  ...
```

Podemos ver que as entradas referentes às ilhas entre as quais foi adicionada a ponte mudaram, passando a referir a ponte adicionada. As mudanças nas entradas referentes às ilhas `ilha(3,(5,1))` e `ilha(1,(5,3))`, reflectem o facto de estas ilhas terem deixado de ser vizinhas.

Exemplo 2: A partir do estado obtido no exemplo 1, vamos agora adicionar uma nova ponte entre as ilhas `ilha(2,(6,2))` e `ilha(2,(8,2))`:

```
?- ...,
   junta_pontes(N_Estado, 1, ilha(2,(6,2)), ilha(2,(8,2)), NN_Estado),
   nl, writeln('Diferencas:'), escreve_diferencas(N_Estado, NN_Estado),
   escreve_Puzzle(Puz, NN_Estado).

Diferencas:
[ilha(2,(4,2)),[ilha(6,(4,5)),ilha(2,(6,2))],[ponte((4,2),(6,2))]]
passou a
[ilha(2,(4,2)),[ilha(6,(4,5))],[ponte((4,2),(6,2))]]
```

```
[ilha(2,(6,2)),[ilha(2,(4,2)),ilha(2,(8,2))],[ponte((4,2),(6,2))]]
passou a
[ilha(X,(6,2)),[ilha(2,(4,2)),ilha(2,(8,2))],
 [ponte((4,2),(6,2)),ponte((6,2),(8,2))]]

[ilha(1,(7,1)),[ilha(3,(5,1)),ilha(3,(7,3))],[]]
passou a
[ilha(1,(7,1)),[ilha(3,(5,1))],[]]

[ilha(3,(7,3)),[ilha(1,(5,3)),ilha(1,(7,1)),ilha(5,(7,5))],[]]
passou a
[ilha(3,(7,3)),[ilha(1,(5,3)),ilha(5,(7,5))],[]]

[ilha(2,(8,2)),[ilha(2,(6,2)),ilha(3,(8,4))],[]]
passou a
[ilha(2,(8,2)),[ilha(3,(8,4))],[ponte((6,2),(8,2))]]
```

```
 2     2     5     2
                1     3
 6     3
    2        6     1
 3  |  1        2     6
    2
 1  |  3     5        3
    2     3        2
```

...

A nova ponte foi adicionada às entradas referentes às ilhas `ilha(2,(6,2))` e `ilha(2,(8,2))`.

A `ilha(2,(6,2))` ficou terminada com a adição da nova ponte e, consequentemente, marcada com `'X'`, e retirada de todas as listas de vizinhas.

As ilhas `ilha(1,(7,1))` e `ilha(3,(7,3))` deixaram de ser vizinhas.

3 Avaliação

A nota do projecto será baseada nos seguintes aspectos:

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

extrai_ilhas_linha	1
ilhas	1
vizinhas	2
estado	1
posicoes_entre	1
cria_ponte	0.5
caminho_livre	1
actualiza_vizinhas_entrada	1
actualiza_vizinhas_apos_pontes	0.5
ilhas_terminadas	1
tira_ilhas_terminadas_entrada	1
tira_ilhas_terminadas	1
marca_ilhas_terminadas_entrada	1
marca_ilhas_terminadas	1
trata_ilhas_terminadas	1
junta_pontes	1

- Estilo de programação e facilidade de leitura (20% - 4 val.). Em particular, serão consideradas as seguintes componentes: .
  - Boas práticas (1,5 valores): serão considerados entre outros a clareza do código e a integração de conhecimento adquirido durante a UC.
  - Comentários (1 valor): deverão incluir comentários para o utilizador (descrição sumária do predicado) e comentários para o programador, quando se justifique.
  - Tamanho de predicados, duplicação de código e abstração procedimental (1 valor).
  - Escolha de nomes (0,5 valores).

4 Penalizações

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

5 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 4 de Fevereiro** de 2022, 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.

[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.

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. Existe ainda um limite temporal entre submissões. 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).

6 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 juiz do que se considera ou não copiar num projecto.

7 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.