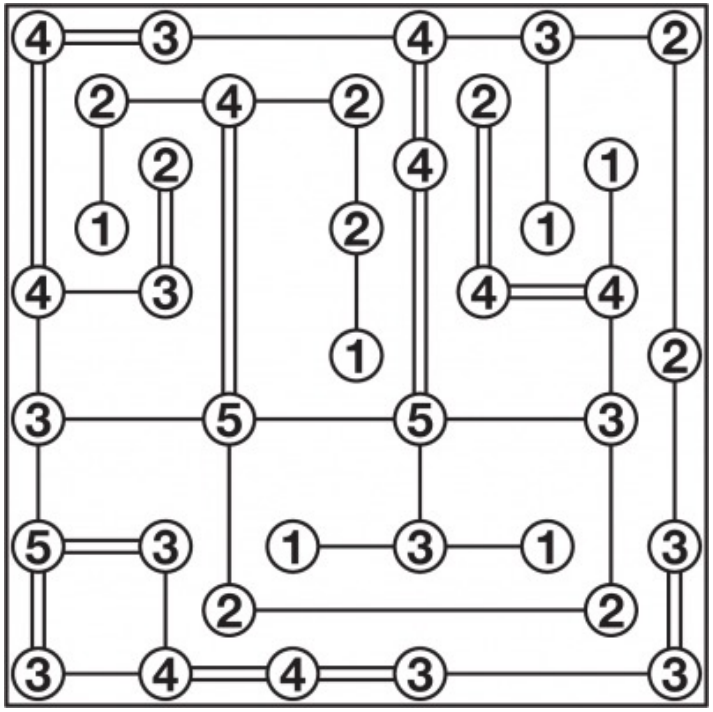
Hashi

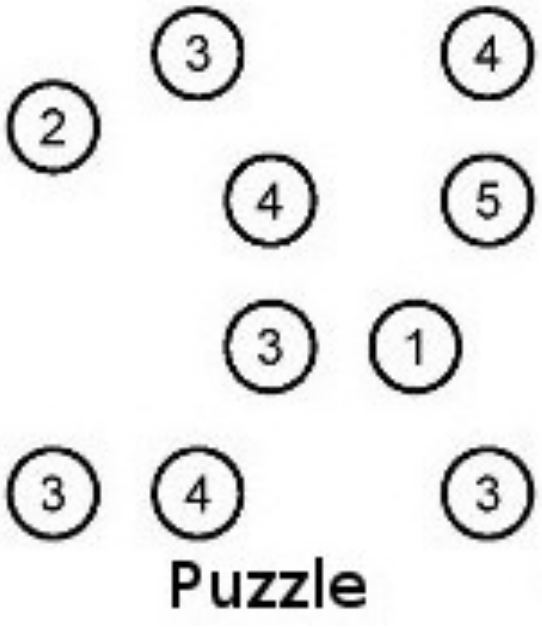
LP, P2



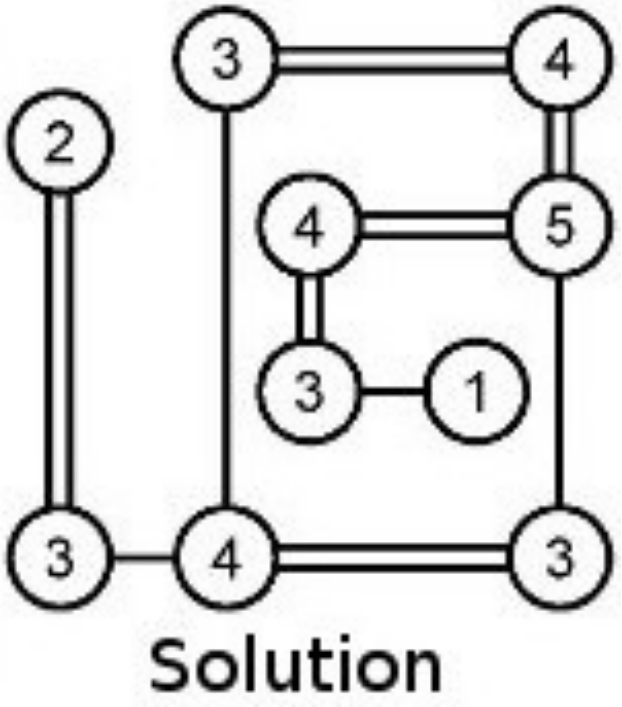
Disclaimer:
esta apresentação não dispensa (de todo) a leitura atenta do enunciado do projecto

Introdução

Hashi – o que é?



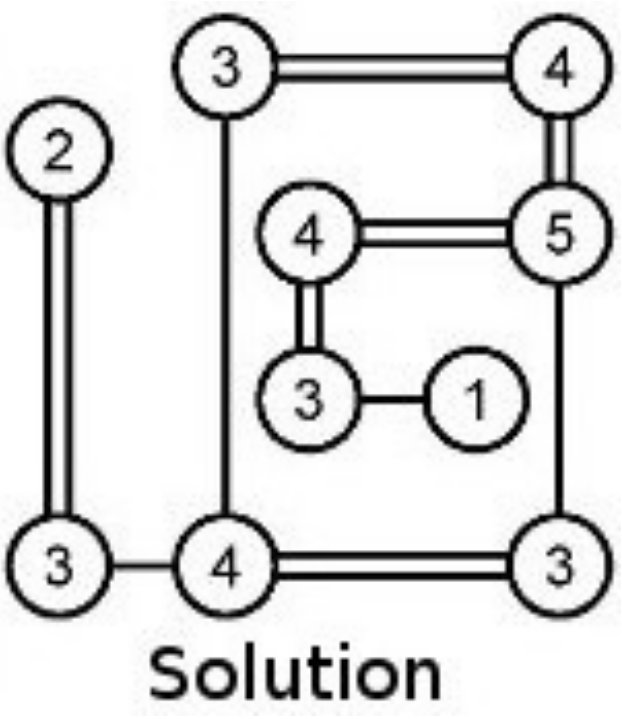
Puzzle



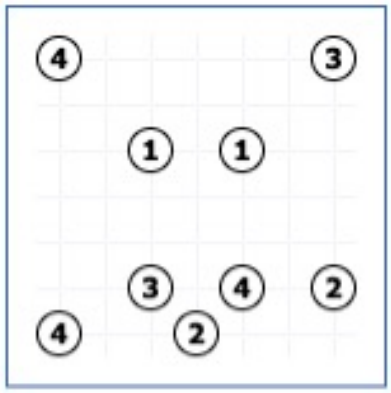
Solution

Para resolver o Puzzle Hashi

- 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.



Representações



Puzzle:

```
[
[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]
]
```

Rep. 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))
],
```

Rep. 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))],[]]
]
```

Estrutura de dados

Predicados que levam à criação da estrutura de dados principal: 1, 2, 3 e 4

Predicados relativos às estruturas de dados

- 1. extrai_ilhas_linha
- 2. ilhas
- 3. vizinhas
- 4.estado

Conceito: “ilha”

Uma ilha é representada pela estrutura de dados

ilha(N, (Linha, Coluna))

em que:

- N é o número de ponte da ilha
- Linha é o número da linha onde está a ilha
- Coluna é o número da coluna onde está a ilha

1. extrai_ilhas_linha(NumLinha, Linha, Ilhas),

O objective deste predicado é criar uma lista com as ilhas de uma dada linha, representada por uma lista. Assim:

- NumLinha é um inteiro que representa o número da linha em causa
- Linha é uma lista, correspondente à linha da qual se querem obter as ilhas
- Ilhas é lista ordenada das ilhas de Linha.

?- extrai_ilhas_linha(1, [4, 0, 0, 0, 0, 0, 3], Ilhas). % Há 4 pontes na coluna 1, 0 na coluna 2, etc.

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. ilhas(Puz, Ilhas)

O objective deste predicado é criar uma lista com as ilhas (Ilhas) de um dado Puzzle (Puz). Assim:

- Ilhas é a lista ordenada (ilhas da esquerda para a direita e de cima para baixo) das ilhas de Puz.

?- 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))

…

```
puzzle_publico(2,
   [[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]]).
```

Conceito: “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 (na implementação de vizinhas não te preocupes com esta condição, dado que não existem pontes inicialmente)

3. vizinhas(Ilhas, Ilha, Vizinhas)

O objective deste predicado é criar uma lista ordenada, Vizinhas, com as vizinhas da ilha Ilha, tendo em conta as outras ilhas existentes (Ilhas).

?- 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,
   [[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]]).
```

```
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))]
```

Conceito: “entrada” e “estado”

Uma entrada é uma lista (atenção!!!!!) em que:

- O primeiro elemento é uma ilha
- O segundo elemento é uma lista de vizinhos dessa ilha
- O terceiro elemento é uma lista das pontes da ilha. Inicialmente é vazia.

Exemplo: [ilha(4,(1,1)),[ilha(3,(1,7)),ilha(4,(7,1))],[]]

Um estado é uma lista de entradas.

4. estado(Ilhas, Estado)

O objectivo deste predicado é criar um estado (Estado) a partir das ilhas existentes (Ilhas).

?- 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))],[]]

```
puzzle_publico(2,
  [[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]]).
```

Predicados auxiliares (I)

5, 6, 7, 8 e 9

Predicados auxiliares (I)

- 5. posicoes_entre
- 6. cria_ponte
- 7. caminho_livre
- 8. actualiza_vizinhas_entrada
- 9. actualiza_vizinhas_apos_pontes

- Nota: 8 com base em 7 e 9 com base em 8

5. posicoes_entre (Pos1, Pos2, Posicoes)

- Posicoes é a lista ordenada de posicoes entre Pos1 e Pos 2, excluindo Pos 1 e Pos2.
- Se não existir nenhum elemento, falha.
- A ter em atenção: se estamos a tratar de linhas ou se estamos a tratar de colunas.

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

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

?- posicoes_entre((5,2), (2,2), Posicoes). % Orienta-se se aparecerem desordenados
Posicoes = [(3, 2), (4, 2)].

?- posicoes_entre((2,2), (5,3), Posicoes). % Falha se não existirem
false.

6. cria_ponte(Pos1, Pos2, Ponte),

- Uma ponte é uma estrutura ponte(Pos1, Pos2), em que Pos1 e Pos2 são as coordenadas das ilhas que a ponte liga (e estão ordenadas).

?- 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)).

?- cria_ponte((1,7), (4,7), Ponte).

Ponte = ponte((1,7),(4,7)).

7. caminho_livre(Pos1, Pos2, Posicoes, Ilha, Vizinho)

- Posicoes são as posições entre Pos1 e Pos2 (podia ser obtida com o predicado 5, mas assim, evita-se estar sempre a invocá-lo), Ilha é uma ilha e Vizinho uma ilha vizinha dessa ilha. Este predicado avalia se adicionar uma ponte entre Pos1 e Pos2 faz (ou não) com que Ilha e Vizinho deixem de ser vizinhas.
- Nota: se a ponte unir Ilha e Vizinho, estes não deixam de ser vizinhos, mas se passar por uma posição qualquer entre eles, então deixam de ser vizinhos.

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

true.

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

false.

8. actualiza_vizinhas_entrada(Pos1, Pos2, Posicoes, Entrada, NovaEntrada)

- Posicoes são as posições entre Pos1 e Pos2 (que, mais uma vez, podiam ser obtida com o predicado 5) e Entrada é uma entrada. A NovaEntrada é igual a Entrada excepto que lhe foram removidas as ilhas que deixaram de ser vizinhas se fosse adicionada uma ponte entre Pos1 e Pos2.

?- actualiza_vizinhas_entrada((2,1), (2,5), [(2, 2), (2, 3), (2, 4)], [ilha(2, (1, 3)), [ilha(2, (3, 3))], []], NovaEntrada).

...

NovaEntrada = [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))], []], NovaEntrada).

...

NovaEntrada = [ilha(1, (2, 1)), [ilha(2, (2, 5))], []].

9. actualiza_vizinhas_apos_pontes(Estado, Pos1, Pos2, NovoEstado)

Estado é um estado, Pos1 e Pos2 são as posições entre as quais foi adicionada uma ponte (desta vez não são dadas as Posicoes). NovoEstado é o estado que se obtém de Estado após a actualização das ilhas vizinhas de cada uma das suas entradas.

?- 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, NovoEstado).

…

NovoEstado = [ilha(2,(1,3)),[],[]]

[ilha(1,(2,1)),[ilha(2,(2,5))],[]]

[ilha(2,(2,5)),[ilha(1,(2,1))],[]]

[ilha(2,(3,3)),[],[]]

Nota (o que aparece no enunciado é o Estado e o NovoEstado):
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))],[]]]

Predicados auxiliaries (II)

10, 11, 12, 13, 14, 15

Predicados auxiliares (II)

- 10. ilhas_terminadas
- 11. tira_ilhas_terminadas_entrada e 12. tira_ilhas_terminadas
- 13. marca_ilhas_terminadas_entrada e 14. marca_ilhas_terminadas
- 15. trata_ilhas_terminadas

Nota: 12 com base em 11, 14 com base em 13, 15 com base em 10, 11 e 14

Conceito: “Ilha Terminada”

- Se a entrada referente a uma ilha for

[ilha(Num_pontes, Pos), Vizinhas, Pontes],

esta ilha está terminada se:

- Num_pontes for diferente de ‘X’ (a razão para esta condição ficará aparente mais à frente)
- o comprimento da lista Pontes for Num_pontes .

10. ilhas_terminadas(Estado, IlhasTerminadas)

Estado é um estado. IlhasTerminadas é a lista de ilhas de Estado que já têm **todas** as pontes, isto é, é a lista das ilhas terminadas.

?- Estado =

[[ilha(3,_),_,[_, _, _]], % terminada porque a lista de pontes tem 3 elementos [_, _, _]

[ilha(2,_),_,[_, _]], % terminada porque a lista de pontes tem 2 elementos [_, _]

[ilha(4,_),_,[_,_]], % não está terminada. Tem apenas 2 pontes [_, _] (e não 4)

[ilha('X',_),_,[_]]], % não está terminada. Tem o X.

ilhas_terminadas(Estado, Ilhas_term).

...

Ilhas_term = [ilha(3, _2318), ilha(2, _2306)].

11. tira_ilhas_terminadas_entrada (IlhasTerm, Entrada, NovaEntrada) e 12. tira_ilhas_terminadas(Estado, IlhasTerm, NovoEstado) vs. 13. marca_ilhas_terminadas_entrada (IlhasTerm, Entrada, NovaEntrada) e 14. marca_ilhas_terminadas(Estado, IlhasTerm, NovoEstado),

- IlhasTerm é uma lista de ilhas terminadas
- Entrada é uma entrada
- Nos predicados "tira"
  - NovaEntrada é a entrada resultante de remover as ilhas de IlhasTerm da lista de ilhas vizinhas de Entrada
  - NovoEstado é o estado resultante de aplicar tira_ilhas_terminadas_entrada a cada uma das entradas de Estado.
- No predicados "marca"
  - NovaEntrada é a entrada resultante do seguinte: se a ilha de Entrada pertencer a IlhasTerm, o seu número de pontes deve ser substituído por ´X´
  - NovoEstado é o estado resultante de aplicar marca_ilhas_terminadas_entrada a cada uma das entradas de Estado.

11. tira_ilhas_terminadas_entrada (IlhasTerm, Entrada, NovaEntrada) e 12. tira_ilhas_terminadas(Estado, IlhasTerm, NovoEstado) vs.

?- 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))], []].

?- 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]]].

13. marca_ilhas_terminadas_entrada (IlhasTerm, Entrada, NovaEntrada) e 14. marca_ilhas_terminadas(Estado, IlhasTerm, NovoEstado)

?- Entrada = [ilha(4,(6,5)),[ilha(1,(3,5)),ilha(3,(6,3)),ilha(2,(6,7))],[]], % não mexe nos vizinhos

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))], []].

?- 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]]].

15. trata_ilhas_terminadas(Estado, NovoEstado)

Estado é um estado. NovoEstado é o estado resultante de aplicar os predicados tira_ilhas_terminadas e marca_ilhas_terminadas a Estado. (nota: primeiro tenho de ver quais são as ilhas terminadas)

?- Estado = [[ilha(1,(1,1)),[ilha(4,(1,3))],[_]], % Ilha terminada => marcada com X e desaparece como vizinha

[ilha(4,(1,3)),[ilha(1,(1,1)),ilha(1,(1,5)),ilha(2,(3,3))],[_, _, _, _]], % idem

[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]]].

Predicado “principal”

Predicado principal

16. junta_pontes

Nota: 16 com base em 6, 9 e 15

16. junta_pontes(Estado, NumPontes, Ilha1, Ilha2, NovoEstado)

Estado é um estado e Ilha1 e Ilha2 são 2 ilhas. NovoEstado é o estado que se obtém de Estado por adição de NumPontes 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.

Avaliação

Ver Manual de Sobrevivência em Prolog -- Anexo A do livro do Prof. Pavão Martins

- Execução correcta (80% - 16 valores)
  - Ver cotação de cada predicado no enunciado
- Qualidade do código (20% - 4 valores):
  - 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).
- Penalização
  - Presença de diacríticos (caracteres acentuados, cedilhas e outros semelhantes): 3 valores
  - Warnings: 2 valores

- Código entregue via Mooshak – ver instruções no enunciado e na página)
- Pode haver oral do projecto!
- Vai ser usado **detector de cópias**
  - analogias com programas encontrados na net serão vistos como cópias.

Quanto às cópias

Dicas

Dicas

- Ataquem o projeto quanto antes (e usem o swi-prolog e nada mais)!!!
- Vejam as avaliações de outras cadeira e organizem-se para não ser um stress de última hora
  - vão estar aproximadamente 300 + 160 alunos no Mooshak
  - não faltem às aulas (de nenhuma cadeira)
- Apareçam nas aulas de dúvidas
- Ajudem-se, mas ******nunca***** partilhem código

Alguns predicados interessantes

- append/3
- nth1/3
- flatten/2
- reverse/2
- member/2
- between/3
- findall, bagof, setof
- include, exclude, maplist
- …