INSTITUTO SUPERIOR TÉCNICO

Lógica para Programação

Projecto

7 de Abril de 2018

Solucionador de problemas de termómetros

O objetivo deste projecto é escrever um programa em PROLOG para resolver problemas de termómetros.

Um problema de termómetros de dimensão n (em que n é um natural) é uma grelha $n \times n$. As linhas e colunas da grelha encontram-se numeradas de 1 a n, de cima para baixo (linhas), e da esquerda para a direita (colunas). Na grelha encontram-se dispostos vários termómetros; estes podem estar na vertical ou na horizontal. Para cada linha e coluna é indicado o número total de posições que devem ser preenchidas. Na Figura 1 apresenta-se um problema de dimensão 5.[1]

O objectivo é determinar que posições dos termómetros devem ser preenchidas, de forma a respeitar os totais indicados para cada linha e coluna, e a seguinte regra: uma posição de um termómetro só pode ser preenchida se estiverem preenchidas todas as posições entre essa posição e a base do termómetro.

Na Figura 2 apresenta-se a solução para o problema da Figura 1.[2]

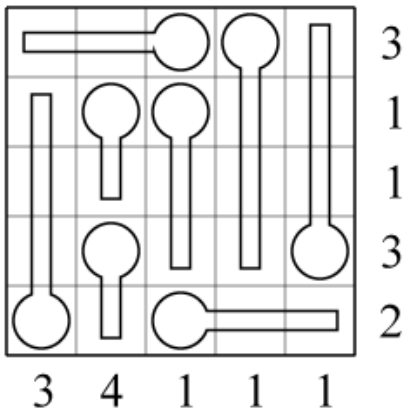

Figura 1: Problema de dimensão 5.

[1] Poderão existir posições da grelha que não estejam preenchidas por termómetros (o que não acontece no exemplo apresentado).

[2] Figuras retiradas de http://syndicate.yoogi.com/thermometer/.

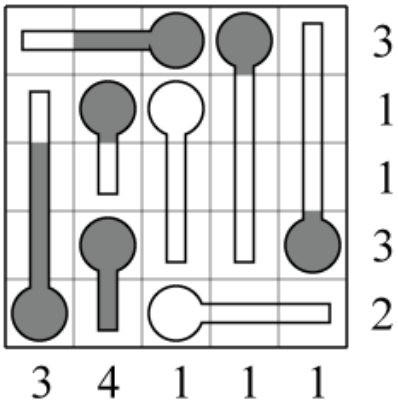

Figura 2: Solução do problema da Figura 1.

1 Abordagem

Nesta secção apresentamos o algoritmo que o seu programa deve usar na resolução de problemas de termómetros, daqui em diante designados simplesmente por *puzzles*.

A abordagem seguida consiste em escolher, para cada linha, uma forma possível de a preencher, respeitando o total dessa linha e as regras do puzzle: uma posição de um termómetro só pode ser preenchida se estiverem preenchidas todas as posições entre essa posição e a base do termómetro. Este processo é repetido para cada linha, até se encontrar uma solução, ou até se encontrar uma inconsistência; neste caso, retrocede-se até à última escolha feita, e faz-se uma escolha diferente.

2 Representação de problemas

Um problema de termómetros é representado em Prolog por uma lista com 3 elementos:

1. Uma lista de termómetros: cada termómetro é representado pela lista de posições ocupadas pelo termómetro (uma posição é representada por um par (linha, coluna)); a primeira posição indica a base do termómetro e a última indica o fim do termómetro. Por exemplo, o termómetro horizontal na primeira linha da Figura 1 é representado pela lista `[(1,3), (1,2), (1,1)]`. A ordem dos vários termómetros na lista de termómetros não é relevante.
2. Uma lista de inteiros positivos, representando os totais das linhas.
3. Uma lista de inteiros positivos, representando os totais das colunas.

Assim, o problema da Figura 1 pode ser representado pela lista:

```
[[[(1,3), (1,2), (1,1)],
  [(1,4), (2,4), (3,4), (4,4)],
  [(4,5), (3,5), (2,5), (1,5)],
  [(5,1), (4,1), (3,1), (2,1)],
  [(2,2), (3,2)],
  [(2,3), (3,3), (4,3)],
  [(4,2), (5,2)],
  [(5,3), (5,4), (5,5)]],
 [3,1,1,3,2],
 [3,4,1,1,1]]
```

3 Trabalho a desenvolver

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

Alguns dos predicados pedidos calculam listas de posições. Para efeitos de avaliação, estas listas devem estar ordenadas. Para ordenar uma lista pode usar o predicado pré-definido `sort/2`, com o seguinte significado: `sort(Lst, Lst_ord)` significa que `Lst_ord` é a lista resultante de ordenar a lista `Lst`, removendo eventuais elementos repetidos de `Lst`. Por exemplo:

```
?- sort([1,2,1,3,0],Lst_ord).
Lst_ord = [0, 1, 2, 3].

?- sort([ (1, 3), (1, 2), (1, 1), (3,1), (2,2)],Lst_ord).
Lst_ord = [ (1, 1), (1, 2), (1, 3), (2, 2), (3, 1)].
```

Os exemplos encontram-se editados para maior facilidade de leitura.

3.1 Predicados auxiliares

Nesta secção são especificados alguns predicados auxiliares, úteis para a implementação do predicado `possibilidades_linha`, descrito na Secção 3.2.

O objectivo deste predicado é determinar as possibilidades existentes para preencher determinada linha de um puzzle. Uma possibilidade para preencher uma linha é representada por uma lista de posições contendo:

- as posições da linha a preencher de modo a perfazer o total da linha, e
- as posições de outras linhas necessárias para respeitar as regras do jogo.

Por exemplo, considerando a linha 1 do puzzle da Figura 1, qualquer possibilidade que contenha a posição `(1, 5)` terá de conter também as posições `(2, 5)`, `(3, 5)`, `(4,`

`5)`, uma vez que estas posições de encontram entre a posição `(1, 5)` e a base do termómetro em questão.

3.1.1 Predicado `propaga/3` (2.0 val.)

Atendendo às regras do puzzle, dada uma posição `Pos`, se quisermos saber quais as posições que devem ser preenchidas para `Pos` poder ser preenchida, devemos procurar na lista de termómetros um termómetro que contenha essa posição; todas as posições desde o início do termómetro até `Pos` terão de ser preenchidas.

Implemente o predicado `propaga/3` que, dados um puzzle e uma posição determina as posições que devem ser preenchidas para que essa posição possa ser preenchida:

`propaga(Puz, Pos, Posicoes)` significa que, dado o puzzle `Puz`, o preenchimento da posição `Pos` implica o preenchimento de todas as posições da lista de posições `Posicoes`.
NOTA: a lista `Posicoes` deve estar ordenada.

Por exemplo, sendo `Puz` o puzzle da Figura 1, temos:

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

?- ..., propaga(Puz, (3,1), Posicoes).
...
Posicoes = [ (3, 1), (4, 1), (5, 1)].

?- ..., propaga(Puz, (1,3), Posicoes).
...
Posicoes = [(1, 3)].
```

3.1.2 Predicado `nao_altera_linhas_anteriores/3` (2.0 val.)

Ao determinar as possibilidades para preencher uma linha é necessário ter em atenção, entre outras restrições, as possibilidades já escolhidas para preencher as linhas anteriores. Isto significa que uma possibilidade para preencher uma linha não deverá implicar o preenchimento de nenhuma posição vazia em linhas anteriores.

Para exemplificar o que foi dito consideremos de novo o puzzle da Figura 1. Suponhamos que para preencher a linha 1 tinha sido escolhida a possibilidade `[(1,1), (1,2), (1,3)]`. Isto significa que a possibilidade `[(2,4), (1,4)]` não pode ser escolhida para preencher a linha 2, pois iria preencher uma posição da linha 1 que se encontrava vazia.

Implemente o predicado `nao_altera_linhas_anteriores/3`:

`nao_altera_linhas_anteriores(Posicoes, L, Ja_Preenchidas)` significa que, dada a lista de posicoes `Posicoes`, representando uma possibilidade de preenchimento para a linha `L`, todas as posições desta lista pertencendo a linhas anteriores a `L`, pertencem à lista de posições `Ja_Preenchidas`. Como o nome indica, esta lista contém todas as posições já preenchidas nas linhas anteriores a `L`.

Por exemplo, sendo `Puz` o puzzle da Figura 1, temos:

```
?- nao_altera_linhas_anteriores([(2,4),(1,4)],
                                2, [(1,1), (1,2), (1,3)]).
false.

?- nao_altera_linhas_anteriores([(2,4),(1,4)],
                                2, [(1,2), (1,3), (1,4)]).
true.
```

3.1.3 Predicado `verifica_parcial/4` (4.0 val.)

O predicado `verifica_parcial`[3] permite verificar se uma possibilidade para preencher uma linha não viola os totais das colunas, tendo em atenção as escolhas feitas anteriormente. Por exemplo, uma possibilidade para preencher a linha 1 do puzzle da Figura 1 seria preencher as posições da lista `[(1,2), (1,3), (1,5), (2,5), (3,5), (4,5)]`.[4] No entanto, esta possibilidade viola o número de posições a preencher na coluna 5.

Implemente o predicado `verifica_parcial/4`:

`verifica_parcial(Puz, Ja_Preenchidas, Dim, Poss)` em que:

- `Puz` é um puzzle,
- `Ja_Preenchidas` é a lista das posições já preenchidas por escolhas anteriores,
- `Dim` é a dimensão de `Puz`,
- `Poss` é lista de posições representando uma potencial possibilidade para preencher uma linha,

significa que `Poss` não viola os totais das colunas, isto é, que se `Poss` for escolhida, nenhuma coluna fica com um número de posições preenchidas superior ao seu total.

Por exemplo, sendo `Puz` o puzzle da Figura 1, temos:

```
?- ..., verifica_parcial(Puz, [], 5, [(1,1), (1,2), (1,3)]).
Puz = [[ (1, 3), (1, 2), ...
```

[3] Este nome deve-se ao facto de a verificação se aplicar a uma solução parcial, isto é, em que ainda não estão preenchidas todas as linhas.

[4] As 3 últimas posições desta lista devem-se ao facto de, para preencher a posição `(1, 5)`, ser necessário também preencher essas posições.

```
?- ..., verifica_parcial(Puz, [], 5,
                         [(1,2), (1,3), (1,5), (2,5), (3,5), (4,5)]).
false.
?- ..., verifica_parcial(Puz, [(1,1), (1,2), (1,3)], 5, [(2,3)]).
false.
```

3.2 Predicado `possibilidades_linha/5` (4.0 val.)

Este predicado permite determinar as possibilidades existentes para preencher uma determinada linha, tendo em atenção as escolhas já feitas para preencher as linhas anteriores, e os totais das colunas.

Uma possibilidade para preencher uma linha `L`, de um puzzle `Puz`, é uma lista de posições, `Poss`, tal que:

- O número de posições de `Poss` pertencentes à linha `L` é igual ao total de posições a preencher na linha `L`.
- Se uma posição `Pos` pertence a `Poss` e `propaga(Puz, Pos, Posicoes)`, então a lista `Posicoes` está contida na lista `Poss`.
- A lista `Poss` respeita as escolhas anteriores. Isto implica que:
  - A lista `Poss` não contém nenhuma posição de linhas anteriores a `L` que não estivesse já preenchida por escolhas anteriores.
  - A lista `Poss` contém todas as posições da linha `L` já preenchidas por escolhas anteriores.
- A escolha de `Poss`, tendo em atenção as escolhas anteriores, respeita (não ultrapassa) o total de posições a preencher de nenhuma coluna do puzzle `Puz`.

Implemente o predicado `possibilidades_linha/5`:

`possibilidades_linha(Puz, Posicoes_linha, Total, Ja_Preenchidas, Possibilidades_L)` em que:

- `Puz` é um puzzle,
- `Posicoes_linha` é uma lista com as posições da linha em questão. Por exemplo, `Posicoes_linha = [(2, 1), (2, 2), (2, 3), (2, 4), (2, 5)]`,
- `Total` é o número total de posições a preencher na linha em questão,
- `Ja_Preenchidas` é a lista das posições já preenchidas por escolhas anteriores,
- `Possibilidades_L` é uma lista de listas de posições,

significa que `Possibilidades_L` é a lista das possibilidades para preencher a linha em questão.
NOTA: a lista `Possibilidades_L` deve estar ordenada.

Por exemplo, sendo `Puz` o puzzle da Figura 1, temos:

```
?- ..., possibilidades_linha(Puz, [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5)],
                             3, [], Possibilidades_L).
Possibilidades_L =
[[(1,1), (1,2), (1,3)],  [(1,2), (1,3), (1,4)]]


?- ..., possibilidades_linha(Puz,[(2, 1), (2, 2), (2, 3), (2, 4), (2, 5)],
                             1, [(1,1), (1,2), (1,3)],
                             Possibilidades_L).
Possibilidades_L = [[(2,2)]]

?- ..., possibilidades_linha(Puz,[(3, 1), (3, 2), (3, 3), (3, 4), (3, 5)],
                             1, [(1,2), (1,3),(1,4), (2,2)],
                             Possibilidades_L).
Possibilidades_L = [[(2, 2), (3, 2)], [(3, 1), (4, 1), (5, 1)]].
```

3.3 Predicado `resolve/2` (4.0 val.)

Este é o predicado principal que, dado um puzzle, nos permite obter (um)a solução, se ela existir.

Implemente o predicado `resolve/2`:

`resolve(Puz, Solucao)` em que:

- `Puz` é um puzzle,
- `Solucao` é uma lista de posições,

significa que `Solucao` é a lista de posições a preencher no puzzle `Puz` para obter uma solução. Na obtenção da solução, deve ser utilizado o algoritmo apresentado na Secção 1. NOTA: a lista `Solucao` deve estar ordenada.

Por exemplo, sendo `Puz` o puzzle da Figura 1 temos:

```
?- ..., resolve(Puz, Solucao).
Solucao = [(1,2), (1,3), (1,4), (2,2), (3,1),
           (4,1), (4,2), (4,5), (5,1), (5,2)]
```

4 Avaliação

A nota do projecto será baseada nos seguintes aspectos:

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

`propaga`	2.0 val.
`nao_altera_linhas_anteriores`	2.0 val.
`verifica_parcial`	4.0 val.
`possibilidades_linha`	4.0 val.
`resolve`	4.0 val.

- 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 11 de Maio** de 2018, através do sistema Mooshak. Depois desta hora, não serão aceites projectos sob pretexto algum.[5]

Deverá ser submetido um único 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 caracter 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. Isto é ainda mais importante quando se estão a testar/comparar os algoritmos. 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.[6] Só são permitidas 10

[5] 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.

[6] 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.

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.

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 às várias 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.

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.