TÉCNICO LISBOA

Lógica para Programação

Projeto

15 de Abril de 2016

Labirintos

O objetivo deste projeto é escrever um programa em PROLOG para encontrar caminhos em labirintos, desde uma posição inicial até uma posição final. Por exemplo, dado o labirinto da Fig. 1, em que as posições inicial e final estão assinaladas com `I` e `F`, respetivamente, uma possível solução está representada na Fig. 2.

1 Representação de labirintos

Consideramos um labirinto como sendo uma grelha, com linhas e colunas numeradas como se mostra na Fig. 3.

Esta grelha é representada por uma lista com um comprimento igual ao número de linhas. Cada elemento desta lista é por sua vez uma lista com um comprimento igual ao número de colunas. Finalmente, cada elemento de uma lista interior corresponde à representação de uma posição, ou célula, da grelha. Designando por `Cij` a representação da célula da linha `i`, coluna `j`, um labirinto com 2 linhas e 3 colunas será representado pela lista

```
[[[C11],[C12],[C13]],
 [[C21],[C22],[C23]]]
```

Consideremos agora a representação de células. Uma célula pode ter paredes em cima (`c`), em baixo (`b`), à esquerda (`e`) e/ou à direita (`d`). Assim, uma célula será representada

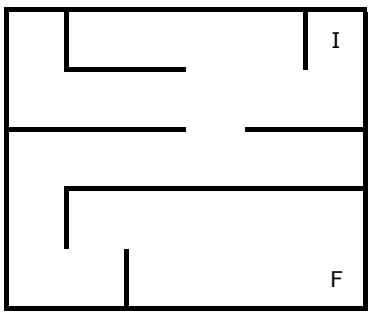

Figura 1: Exemplo de labirinto.

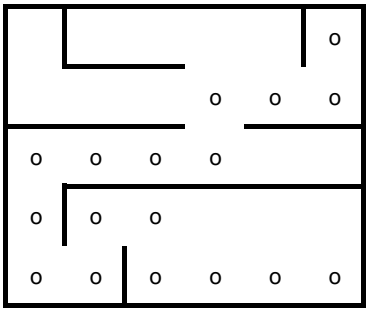

Figura 2: Solução para o labirinto da Fig. 1.

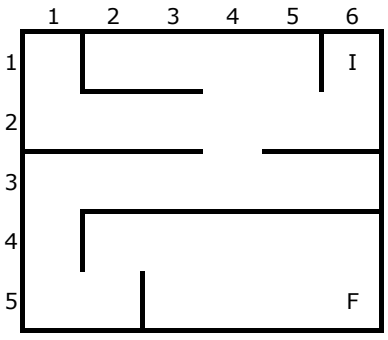

Figura 3: Numeração das linhas e colunas do labirinto da Fig. 1.

por uma lista cujos elementos podem ser `c`, `b`, `e` ou `d`. Por exemplo, o labirinto da Fig. 1 será representado por

```
[[[d,e,c],[e,b,c],[b,c],[c],[d,c],[d,e,c]],
 [[e,b],[b,c],[b,c],[],[b],[d,b]],
 [[e,c],[b,c],[b,c],[b],[b,c],[d,b,c]],
 [[d,e],[e,c],[c],[c],[c],[d,c]],
 [[e,b],[d,b],[e,b],[b],[b],[d,b]]]
```

Embora esta representação apresente redundâncias, facilita a determinação dos movimentos possíveis a partir de uma célula.

2 Representação de soluções

Nos labirintos, as direções dos movimentos permitidos são: para cima (`c`), para baixo (`b`), para a esquerda (`e`), para a direita (`d`). Não são permitidos movimentos na diagonal. Uma solução para um labirinto consiste numa sequência de pares da forma (⟨direção⟩, ⟨posição⟩), em que ⟨direção⟩ é uma das 4 direções permitidas, e ⟨posição⟩ é um par (⟨linha⟩, ⟨coluna⟩). O par (dir, pos) significa que a posição pos foi atingida a partir da posição anterior na solução, fazendo um movimento na direção dir. Daqui em diante referimo-nos a estes pares como movimentos.[1] O primeiro elemento da sequência correspondente a uma solução é da forma (`i`, ⟨posição⟩), em que ⟨posição⟩ representa a posição inicial do labirinto. A posição do último par da sequência é a posição final do labirinto.

Por exemplo, a solução apresentada na Fig. 2 é representada por

```
[(i,1,6), (b,2,6), (e,2,5), (e,2,4), (b,3,4), (e,3,3), (e,3,2),
 (e,3,1), (b,4,1), (b,5,1), (d,5,2), (c,4,2), (d,4,3), (b,5,3),
 (d,5,4), (d,5,5), (d,5,6)].
```

Esta solução significa que começando na posição `(1,6)`, fazendo um movimento para baixo (para a posição `(2,6)`), seguido de um movimento para a esquerda (para a posição `(2,5)`), etc., se chega à posição final. Note-se que um dos componentes, direção ou posição, de cada elemento da solução seria suficiente. A representação apresentada facilita a verificação da correção de soluções.

3 Trabalho a desenvolver

Nesta secção são descritos os predicados que deve desenvolver no seu projeto. Estes serão os predicados avaliados e, consequentemente, devem respeitar escrupulosamente as especificações apresentadas. A ordem pela qual são apresentados os predicados é a ordem pela qual os deverá desenvolver.

Para além dos predicados descritos, poderá desenvolver todos os predicados que julgar necessários.

[1] Na realidade, o movimento é definido pelo par e pela posição anterior na solução.

3.1 Predicado `movs_possiveis/4`

Este predicado determina, dados um labirinto, uma posição atual e os movimentos já efetuados, os movimentos possíveis, tendo em conta que cada posição não deve ser visitada mais do que uma vez. `movs_possiveis(Lab, Pos_atual, Movs, Poss)` significa que dados um labirinto `Lab`, uma posição atual `Pos_atual` e uma lista de movimentos já efetuados `Movs`, os movimentos possíveis são `Poss`. Por exemplo, sendo `Lab1` o labirinto da Fig. 1, temos:

```
?- ..., movs_possiveis(Lab1, (2,5),
       [(i, 1, 6), (b, 2, 6), (e, 2, 5)], Poss).
Lab1 = ...,
Poss = [ (c, 1, 5), (e, 2, 4)].
```

Os movimentos possíveis devem ser apresentados pela ordem `c, b, e, d`.

3.2 Predicado `distancia/3`

Para calcular a distância entre duas posições `(L1, C1)` e `(L2, C2)` defina o predicado `distancia/3`, tal que `distancia((L1, C1),(L2, C2),Dist)` afirma que `Dist = abs(L1 - L2) + abs(C1 - C2)`.

3.3 Predicado `ordena_poss/4`

Este predicado ordena os movimentos possíveis determinados pelo predicado `movs_possiveis/4`, definido na Secção 3.1, segundo dois critérios: em primeiro lugar, os movimentos conducentes à menor distância a uma posição final, e, em caso de igualdade, os movimentos conducentes a uma maior distância a uma posição inicial. `ordena_poss(Poss, Poss_ord, Pos_inicial, Pos_final)` significa que `Poss_ord` é o resultado de ordenar os movimentos possíveis `Poss`, segundo os critérios acima, em relação à posição inicial `Pos_inicial` e à posição final `Pos_final`. Por exemplo:

```
?- ordena_poss([ (c, 4, 4), (d, 5, 5)], Poss_ord, (1, 6), (5, 6)).
Poss_ord = [ (d, 5, 5), (c, 4, 4)].

?- ordena_poss([ (c, 1, 5), (e, 2, 4)],  Poss_ord, (1, 6), (5, 6)).
Poss_ord = [ (e, 2, 4), (c, 1, 5)].
```

Sugestão: utilize o predicado `distancia/3`, definido na Secção 3.2.

3.4 Predicado `resolve/4`

Este predicado corresponde ao predicado principal: `resolve(Lab, Pos_inicial, Pos_final, Movs)` significa que a sequência de movimentos `Movs` é uma solução para

o labirinto `Lab`, desde a posição inicial `Pos_inicial` até à posição final `Pos_final`. Deverá desenvolver 2 versões deste predicado:

- `resolve1(Lab, Pos_inicial, Pos_final, Movs)`: a solução `Movs` apenas deve obedecer à restrição de não passar mais do que uma vez pela mesma célula. Os movimentos possíveis devem ser testados pela ordem `c, b, e, d`. Sugestão: utilize o predicado `movs_possiveis/4`, definido na Secção 3.1.
- `resolve2(Lab, Pos_inicial, Pos_final, Movs)`: a solução `Movs` é gerada escolhendo entre os vários movimentos possíveis, aquele que se aproximar mais da posição final. Se existirem vários movimentos conducentes a posições com a mesma distância à posição final, é escolhido o que mais se distanciar da posição inicial. No caso de existirem vários movimentos conducentes a posições com as mesmas distâncias às posições inicial e final, deve ser seguida a ordem `c, b, e, d`. Sugestão: utilize os predicados `movs_possiveis/4` e `ordena_poss/4`, definidos nas Secções 3.1 e 3.3, respetivamente.

Para melhor compreensão do que foi dito anteriormente, consideremos o labirinto da Fig. 4, com a posição inicial `(1,1)` e a posição final `(3,3)`. Teríamos:

```
?- resolve1([[[c, e], [c], [c, d]],
             [[e], [], [d]],
             [[b, e], [b], [b, d]]],
            (1,1), (3,3), Movs).
Movs = [ (i,1,1), (b,2,1), (b,3,1), (d,3,2),
         (c,2,2), (c,1,2), (d,1,3), (b,2,3), (b,3,3)] .

?- resolve2([[[c, e], [c], [c, d]],
             [[e], [], [d]],
             [[b, e], [b], [b, d]]],
            (1,1), (3,3), Movs).
Movs = [ (i,1,1), (b,2,1), (b,3,1), (d,3,2), (d,3,3)] .
```

A solução de `resolve1` foi obtida escolhendo sempre que possível o movimento para cima, se tal não for possível o movimento para baixo, se tal não for possível o movimento para a esquerda, e só se todos os anteriores falharem, o movimento para a direita. Um movimento só é possível se, para além das paredes do labirinto o permitirem, também não conduzir a uma célula já visitada. Certifique-se de que compreende claramente as razões das escolhas feitas.

Consideremos agora a solução de `resolve2`. Nos dois primeiros movimentos, `(b,2,1)` e `(b,3,1)`, havia duas possibilidades: para baixo e para a direita, ambas conducentes a posições à mesma distância das posições inicial e final. Como deve ser seguida a ordem `c, b, e, d`, foi escolhido o movimento para baixo. No terceiro movimento, `(d,3,2)`, só havia uma possibilidade. Finalmente, no quarto movimento, `(d,3,3)`, havia duas possibilidades: para cima e para a direita. Foi escolhida a segunda possibilidade por conduzir a uma posição com menor distância (zero) à posição final.

Consideremos agora a solução apresentada na Fig. 2. Esta solução corresponde à calculada pelo predicado `resolve2`, cujos 4 primeiros elementos são `[ (i,1,6), (b,2,6),`

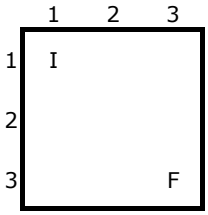

Figura 4: Labirinto apenas com as paredes exteriores.

`(e,2,5), (e,2,4), ...]`. Os dois primeiros movimentos, `(b,2,6)` e `(e,2,5)`, correspondem às únicas possibilidades existentes. Para o terceiro movimento, `(e,2,4)`, havia duas possibilidades: para cima e para a esquerda, ambos conducentes a posições à mesma distância da posição final. Foi escolhida a segunda possibilidade por conduzir a uma posição com maior distância à posição inicial.

4 Avaliação

A nota do projeto será baseada nos seguintes aspectos:

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

`movs_possiveis`	3 val.
`distancia`	1 val.
`ordena_poss`	4 val.
`resolve1`	4 val.
`resolve2`	4 val.

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

5 Recomendações

- Recomenda-se o uso do SWI PROLOG, dado que este vai ser usado para a avaliação do projeto.
- O ficheiro deverá estar em UTF-8 e em lado algum deverão ser usados carateres acentuados ou cedilhas.
- 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.

6 Condições de realização e prazos

O projeto deve ser realizado em grupos de 1 ou 2 alunos. Os alunos de um grupo podem pertencer a qualquer turno das aulas práticas. Os alunos devem proceder à inscrição do grupo através do sistema Fénix.

O código do projeto deve ser entregue obrigatoriamente por via electrónica até às 23:59 do dia 9 de Maio de 2016. O código do projeto deve estar contido num único ficheiro, com o nome `lp-g`x`.pl`, em que x deve ser substituído pelo número do grupo, por exemplo, para o grupo 3 será `lp-g3.pl`. Se durante o desenvolvimento forem usados vários ficheiros, no final, antes de enviar o projeto, todo o código deve ser colocado num único ficheiro. No início do ficheiro deve estar um comentário com o número do grupo e os números e os nomes do alunos. A partir das 24:00 horas do dia 9 de Maio de 2016 não se aceitam quaisquer projetos, seja qual for o pretexto. Pode ou não haver uma discussão oral do projeto e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

7 Cópias

Projetos 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 projeto.