TÉCNICO LISBOA

# Lógica para Programação

Projecto

8 de Abril de 2015

## O puzzle de 8

O puzzle de 8 é uma versão simplificada do puzzle de 15 (também chamado puzzle Gem, puzzle Boss, jogo dos quinze e quadrado místico) que contém 8 quadrados deslizantes, numerados de 1 a 8, montados numa moldura quadrada, na qual um quadrado não está preenchido e corresponde a um espaço vazio (Figura 1). O objectivo do puzzle é movimentar as peças, fazendo movimentos deslizantes que usam o espaço vazio, de modo a partir de uma configuração inicial atingir uma outra configuração.

O puzzle (com 15 quadrados) foi inventado nos Estados Unidos por Noyes Palmer Chapman, no último quartel do Século XIX, e em 1880 era já muito popular nos Estados Unidos, no Canadá e na Europa.

As configurações possíveis do puzzle podem ser agrupadas em dois conjuntos disjuntos, na realidade, duas classes de equivalência, em que cada conjunto contém todas as configurações que podem ser obtidas a partir de qualquer configuração existente nesse conjunto. Não é possivel obter nenhuma configuração existente num dos conjuntos partindo de qualquer configuração do outro conjunto. Só por curiosidade, tendo conhecimento deste facto, o americano Samuel Loyd (1841–1911), ofereceu um prémio de $ 1 000 (cerca de € 24 000 nos nossos dias) a quem conseguisse obter uma solução do puzzle de 15 para uma transformação especificada por Loyd, que consistia em trocar os quadrados com os números 14 e 15, a qual era impossível (Figura 2), desafio esse que contribuiu para aumentar a popularidade do puzzle.

Com este projecto pretende-se desenvolver um programa em SWI PROLOG para resolver o puzzle de 8. O seu programa deve receber duas configurações do puzzle e o objectivo é listar os movimentos que transformam a primeira configuração na segunda.

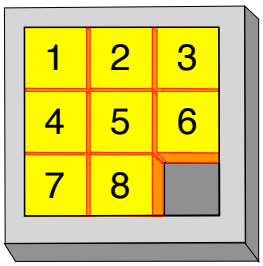


Figura 1: Exemplo do puzzle de 8.

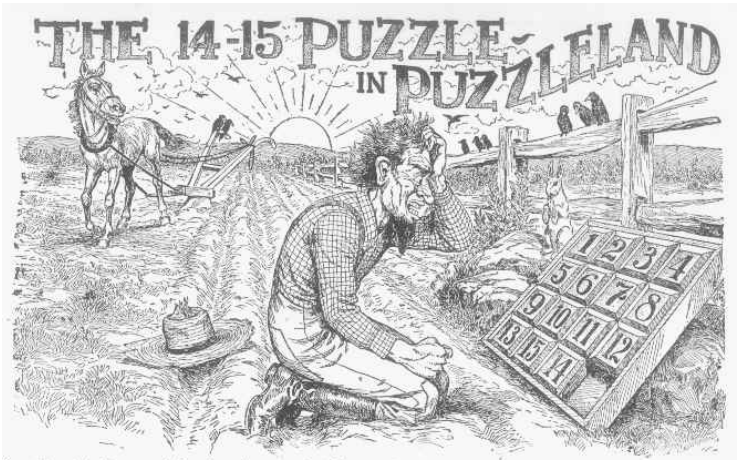


Figura 2: Imagem referindo a transformação de Loyd, 1914 (imagem obtida de Wikipédia).

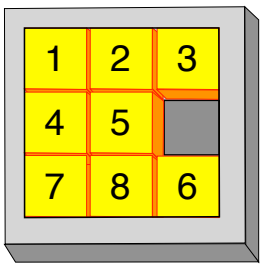


Figura 3: Puzzle de 8 da Figura 1 depois do comando `b`.

A sua solução deve ser obtida através de três métodos distintos: solução manual; solução através de procura cega; e solução através de procura informada.

## 1 Solução manual

A solução manual é gerada através de comandos fornecidos ao programa que indicam quais os movimentos a realizar. Os movimentos podem ser considerados como movimentando uma das peças adjacentes ao espaço vazio e são `c` (move a peça para cima); `b` (move a peça para baixo); `e` (move a peça para a esquerda); e `d` (move a peça para a direita). Por exemplo, partindo da configuração apresentada na Figura 1, o comando `c` origina a configuração apresentada na Figura 3.

A solução manual deverá ser originada pelo predicado `resolve_manual/2` que recebe como primeiro argumento a configuração inicial e como segundo argumento a configuração desejada. Por exemplo,

```
?- resolve_manual([1, 2, 3,  4, 5, 6,  7, 8, 0],
|                 [1, 0, 2,  4, 5, 3,  7, 8, 6]).

Transformacao desejada:

 1  2  3      1     2
 4  5  6  ->  4  5  3
 7  8         7  8  6

Qual o seu movimento?
|: b.

 1  2  3
 4  5
 7  8  6

Qual o seu movimento?
|: d.

 1  2  3
 4     5
 7  8  6

Qual o seu movimento?
|: e.

 1  2  3
 4  5
 7  8  6

Qual o seu movimento?
|: b.

 1  2
 4  5  3
 7  8  6

Qual o seu movimento?
|: e.
Movimento ilegal

Qual o seu movimento?
|: d.

 1     2
 4  5  3
 7  8  6

Parabens!
true
```

## 2 Solução através de procura cega

Uma procura é dita "cega" quando aplica de modo sistemático as regras existentes, sem se preocupar com o facto de se estar a aproximar ou a distanciar da solução.

A solução através de procura cega deverá ser originada pelo predicado `resolve_cego/2` que tem como primeiro argumento a configuração original e como segundo argumento a configuração desejada. Por exemplo,

```
?- resolve_cego([1, 2, 3,  4, 5, 6,  7, 8, 0],
|               [1, 0, 2,  4, 5, 3,  7, 8, 6]).

Transformacao desejada:
 1  2  3      1     2
 4  5  6  ->  4  5  3
 7  8         7  8  6

mova a peça 6 para baixo
mova a peça 3 para baixo
mova a peça 2 para a direita.
true .
```

De modo a evitar a entrada em ciclo, ou seja, fazer repetitivamente os mesmos movimentos, o seu algoritmo de procura cega deve lembrar-se de quais as configurações já geradas, evitando gerar duas vezes a mesma configuração.

## 3 Solução através de procura informada

O algoritmo de procura informada a utilizar corresponde a um algoritmo inventado por Peter Hart, Nils Nilsson e Bertram Raphael do Stanford Research Institute[1], como uma extensão ao algoritmo de Edsger Dijkstra[2], e conhecido por A* (A asterisco).

O algoritmo A* explora de modo sistemático os possíveis estados (no nosso caso, um estado corresponde a uma configuração do puzzle) de um mundo que podem ser obtidos a partir de um estado inicial, aplicando as transformações possíveis. Os estados são organizados numa árvore em que cada nó corresponde a um estado do mundo e os sucessores desse nó correspondem aos estados que podem ser obtidos a partir dele, aplicando todas as transformações possíveis. Por exemplo, na Figura 4 apresentamos os estados (configurações do puzzle) que podem ser obtidos a partir da configuração apresentada na Figura 1 apenas com um movimento.

Cada estado é associado a um número, $f$ que corresponde à soma do número de transformações realizadas para obter esse estado desde o estado inicial, $g$, e de uma estimativa do número de transformações que vão ser necessárias para atingir a solução a partir desse

[1] Hart, P. E., Nilsson, N. J., Raphael, B., "Correction to a Formal Basis for the Heuristic Determination of Minimum Cost Paths", *SIGART Newsletter 37*, pp. 28–29. 1972.

[2] Dijkstra, E. W., "A note on two problems in connexion with graphs", *Numerische Mathematik 1*, pp. 269–271, 1959.

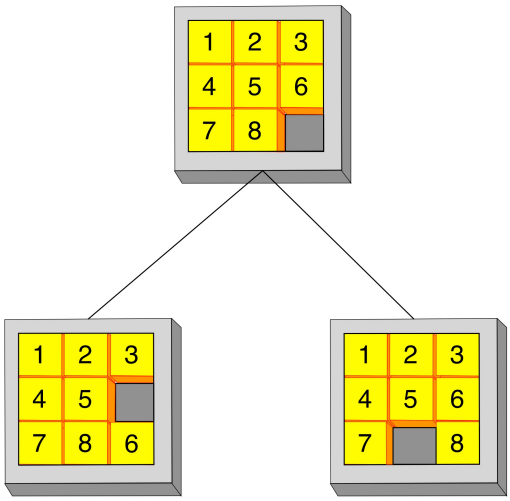


Figura 4: Configurações obtidas a partir da configuração da Figura 1.

estado, $h$. Ao passo que a primeira parcela, $g$, é fácil de calcular, a segunda parcela, $h$, está associada a uma regra empírica ou *heurística*[3].

Duas heurísticas comuns para o puzzle de 8 correspondem: (1) à *distância de Hamming* o número de quadrados fora de posição; ou (2) à *distância de Manhattan* a soma na horizontal e na vertical do número de movimentos que cada quadrado tem que percorrer até ser colocado na posição correcta. No seu projecto pode utilizar qualquer destas heurísticas.

O algoritmo A* utiliza duas listas, a lista de abertos e a lista de fechados. A *lista de abertos* contém todos os estados que já foram gerados mas ainda não foram expandidos. Diz-se que um estado é *expandido* quando são encontrados os estados que podem ser obtidos a partir dele. No início do algoritmo, a lista de abertos contém apenas o nó inicial. A *lista de fechados* contém todos os estados que já foram expandidos. No início do algoritmo, a lista de fechados é vazia.

Em cada passo, o algoritmo escolhe da lista de abertos o nó com menor valor de $f$ (se existirem dois ou mais nós com o mesmo valor de $f$, escolhe o primeiro da lista), remove esse nó da lista de abertos e coloca-o na lista de fechados, expande esse nó (ou seja calcula os seus sucessores), calculando o valor de $f$ para cada um dos seus sucessores, e coloca na lista de abertos todos os sucessores que não se encontram nem na lista de abertos nem na lista de fechados. Estes passos são repetidos até que o nó escolhido para expansão corresponda à solução, caso em que a solução foi encontrada, ou a lista de abertos seja vazia, caso em que não é possível encontar uma solução. No caso de ter sido encontrada a solução, o algoritmo devolve a lista de movimentos necessários para transformar o nó inicial na solução.

A solução através de procura informada deverá ser originada recorrendo a um dos predicados `resolve_info_h/2` ou `resolve_info_m/2` que têm como primeiro argumento a configuração original e como segundo argumento a configuração desejada. O predicado `resolve_info_h` utiliza a distância de Hamming e o predicado `resolve_info_m` utiliza a distância de Manhattan (apenas deverá implementar um destes predicados). Por

[3]Do grego "heuriskein", significando descoberta.

exemplo,

```
?- resolve_info_m([1, 2, 3, 4, 5, 6, 7, 8, 0],
                  [1, 2, 3, 7, 4, 5, 8, 0, 6]).

Transformacao desejada:

 1  2  3      1  2  3
 4  5  6  ->  7  4  5
 7  8         8     6

mova a peça 6 para baixo
mova a peça 5 para a direita
mova a peça 4 para a direita
mova a peça 7 para cima
mova a peça 8 para a esquerda.
true .
```

## 4 Crédito adicional

Procure na internet algoritmos para determinar se é possivel transformar uma configuração noutra. Escreva o predicado `transformacao_possivel/2`. O literal

```
transformacao_possivel(C1, C2)
```

tem o valor verdadeiro apenas se é possível transformar a configuração `C1` na configuração `C2`.

## 5 Sugestões

1. Recomenda-se o uso do SWI PROLOG, dado que este vai ser usado para a avaliação do projecto.
2. O ficheiro deverá estar em UTF-8 e em lado algum deverão ser usados caracteres acentuados ou cedilhas.
3. Utilize como representação do puzzle uma lista com os inteiros de 1 a 8 e o 0 para representar o espaço em branco. A partir desta representação pode escrever os predicados que indicam qual a transformação a realizar e que mostram uma configuração do puzzle.
4. Defina o predicado `mov_legal/4`. O literal

   ```
   mov_legal(C1, M, P, C2)
   ```

   afirma que a configuração `C2` é obtida da configuração `C1`, fazendo o movimento `M`, com a peça `P`. Por exemplo:

```
?- mov_legal([1, 2, 3, 4, 5, 0, 7, 8, 6], M, P, C).
M = c,
P = 6,
C = [1, 2, 3, 4, 5, 6, 7, 8, 0] ;
M = b,
P = 3,
C = [1, 2, 0, 4, 5, 3, 7, 8, 6] ;
M = d,
P = 5,
C = [1, 2, 3, 4, 0, 5, 7, 8, 6] ;
false.
```

Teste cuidadosamente este predicado para garantir que este considera todas as transformações possíveis.

5. Começe por desenvolver a parte do programa que corresponde à solução manual. Utilizando o predicado `mov_legal`, escreva o predicado `resolve_manual/2`. O literal

   `resolve_manual(C1, C2)`

   afirma que certos movimentos transformam a configuração `C1` na configuração `C2`.

   Este predicado origina um ciclo que pede ao utilizador que indique um movimento, verifica se este é um movimento legal, e executa o movimento desejado. Este ciclo termina quando a configuração objectivo é atingida.

6. Concentre-se em seguida no desenvolvimento da solução cega. A procura cega utiliza o mecanismo de controle do PROLOG, gerando movimentos, recorrendo ao predicado `mov_legal`, testando se eles já foram gerados, para cada movimento novo, verifica se ele origina uma solução e, se não for o caso, gera mais movimentos. Para isso, defina o predicado `resolve_cego/2`. O literal

   `resolve_cego(C1, C2)`

   afirma que existe um lista de movimentos que transforma a configuração `C1` na configuração `C2`. Os movimentos podem ser representados por uma lista de pares em que cada par indica um movimento de uma peça e qual a peça movimentada.

   Não se esqueça de recorrer a uma lista que contém as configurações já geradas, evitando que a mesma configuração seja usada mais do que uma vez. Mesmo assim, vai ver que o seu programa *é mesmo cego* podendo gerar movimentos muito irracionais até conseguir chegar à solução.

7. Aborde agora a procura informada. Começe por escolher uma heurística e escreva um predicado que a calcule.

   Poderá representar um estado do problema pela estrutura `no(C, F, G, H, M)` que corresponde ao estado com a configuração `C`, com valor da função $f$, `F`, com valor da função $g$, `G`, com valor da função $h$, `H`, e usando os movimentos `M` para o atingir a partir da configuração inicial.

8. Finalmente, e como trabalho opcional, escreva o predicado `transformacao_possivel` que decide se uma dada transformação é ou não possível e altere os seus programas introduzindo esta verificação antes de começar a resolver o puzzle.

9. Durante o desenvolvimento do programa é importante não se esquecer da *Lei de Murphy*:
   (a) Todos os problemas são mais difíceis do que parecem;
   (b) Tudo demora mais tempo do que nós pensamos;
   (c) Se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis.

## 6 Avaliação

A nota do projecto será baseada nos seguintes aspectos:

- Execução correcta (80%). Destes 80%, 6 valores correspondem à solução manual, 6 valores correspondem à procura cega e 4 valores correspondem à procura informada.
- O predicado que testa se uma solução é possível é opcional e vale 2 valores adicionais, contudo a nota do projecto não poderá ser superior a 20 valores.
- Qualidade do código, a qual inclui abstracção relativa aos predicados definidos, nomes escolhidos, paragrafação e qualidade dos comentários (20%).

## 7 Condições de realização e prazos

O projecto deve ser realizados em grupos de 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 projecto deve ser entregue obrigatoriamente por via electrónica até às 23:59 do dia 19 de Maio de 2015. O código do projecto 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 projecto, 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 19 de Maio de 2015 não se aceitam quaisquer projectos, seja qual for o pretexto.

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

## 8 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.