Movimento do cavalo

Prof. Antonio Carlos Mariani

INE | CTC

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Problema

Considere o tabuleiro de xadrez, cujas casas podem ser identificadas por um par (x, y), sendo que x e y são números naturais e estão no intervalo fechado [1, 8].

Dadas, então, as coordenadas de duas casas do tabuleiro, (x_1, y_1) e (x_2, y_2), escreva um programa que verifique se é possível um cavalo mover-se de uma para outra casa, retornando verdade em caso afirmativo e falso caso não seja possível.

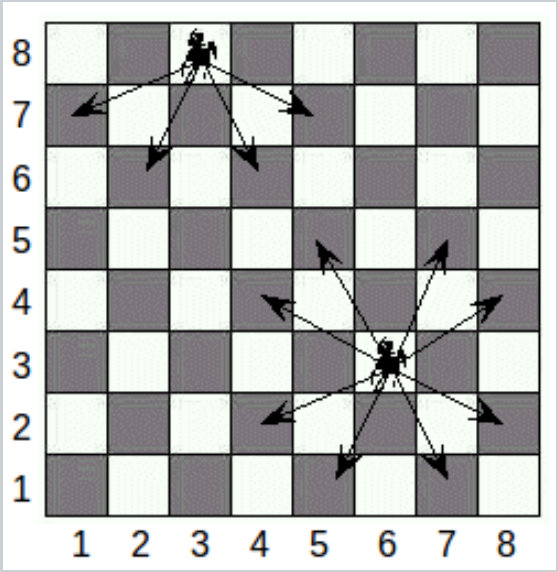
Perguntas:

1) Como representar (modelo, esquema) o problema?
2) O que já é conhecido ou precisamos conhecer para resolver o problema?
3) Quais são os dados de entrada?
4) Qual é a incógnita (o *x* da questão)?
5) Qual estratégia de solução de problemas parece mais adequada para este problema em particular?

Esquema

O problema pode ser representado pelo esquema abaixo no qual as duas dimensões (linhas e colunas) são numeradas de 1 a 8. Cada casa é identificada por um par (x, y), onde x representa a linha e y a coluna.

Sabe-se que o cavalo sempre se movimenta em formato da letra "L", ou seja uma casa numa das dimensões e duas casas na outra.



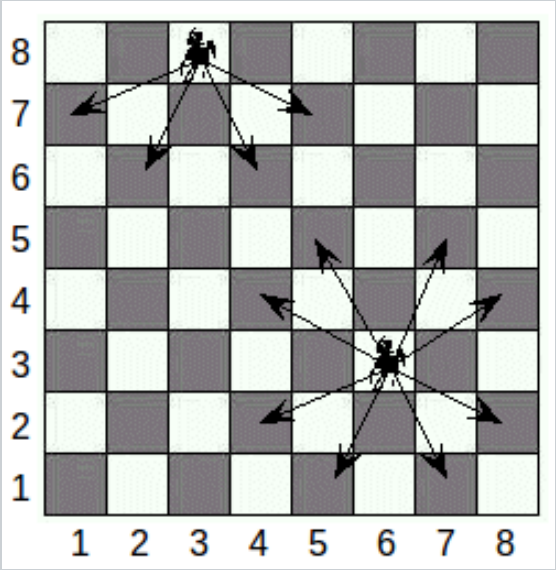
O esquema apresenta dois exemplos típicos:

1. da casa (3, 6) no qual há 8 possibilidades de movimentos;
2. da casa (8, 3) no qual há apenas 4 possibilidades.

Importante notar que há outros casos, como da casa (8, 8) para a qual há apenas duas possibilidades de movimento.

Entrada e saída

Conforme descrito no problema, são dados 4 valores naturais no intervalo [1, 8] como entrada, os quais definem os pares ordenados (x_1, y_1) e (x_2, y_2), nesta ordem. O primeiro par diz onde o cavalo está enquanto que o segundo indica a posição para onde o cavalo deve ser movido. Mas em realidade pode ser também o inverso visto que o movimento do cavalo é reversível.



O que queremos saber (a incógnita) é se o cavalo pode mover-se da posição (x_1, y_1) para a posição (x_2, y_2), e vice-versa, segundo as regras do jogo de xadrez. O resultado é valor lógico (verdade ou falso), conforme o caso.

	Origem	Destino	Resultado
Ex:	(8, 3)	(7, 5)	verdade
	(3, 6)	(4, 2)	falso

Primeira versão

Baseado naquilo que já levantamos, podemos propor uma primeira versão parcial para nosso programa que considera os 3 passos mais gerais:

1) Entrada de dados
2) Processamento
3) Saída

cavalo.py *

```
1  # Entrada de dados
2  x1 = int(input())
3  y1 = int(input())
4  x2 = int(input())
5  y2 = int(input())
6
7  # Computar o resultado
8  pode_moverse = False
9
10 # Mostrar o resultado
11 print(pode_moverse)
```

É claro que o passo 3 (linhas 7 e 8) está incompleto e precisa necessariamente ser desenvolvido.

Estratégia de solução

Há diferentes formas de resolver este problema, sendo provavelmente trivial para alguém experiente. Mas considerando que não somos, como buscar solução?

Note que para este problema, dadas duas coordenadas (x_1, y_1) e (x_2, y_2) só há duas possibilidades de resultado (verdade ou falso), indicando se o cavalo pode ou não mover-se entre as casas. Podemos tirar proveito disto visto que se conseguirmos determinar todos os casos favoráveis (verdade), por complemento todos os demais são desfavoráveis (falso). Ou o contrário, podemos determinar todos os casos negativos e por complementos temos os positivos.

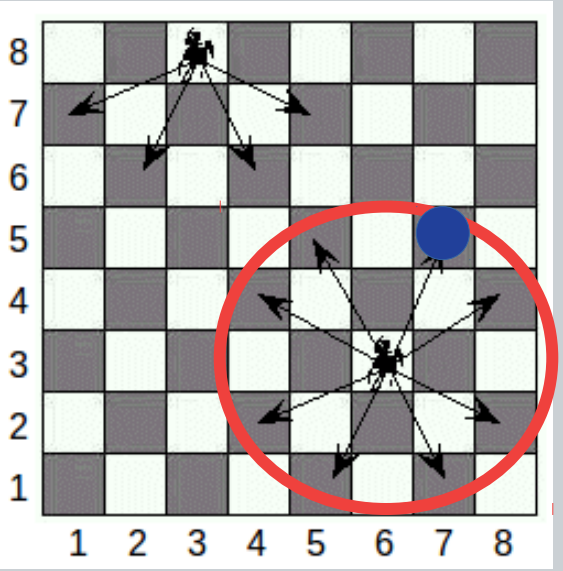
Qual abordagem adotar?

Estratégia de solução

Tomemos o caso da casa (3, 6). No nosso esquema é possível ver que há 8 casas para as quais o cavalo pode mover-se (os casos favoráveis) e 55 casas para as quais ele não pode mover-se (os casos desfavoráveis). Isto sugere que para este problema talvez seja mais profícuo pensar nos casos favoráveis.

Duas estratégias de resolução de problemas que podem ser úteis são:

- Faça uma lista
- Procure por padrões



Lista de casos favoráveis, em sentido horário a partir da 1º casa acima e à direita:

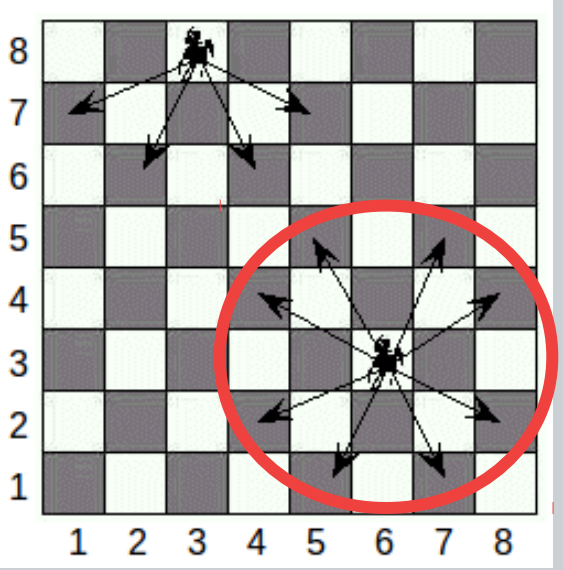
x_1	y_1	x_2	y_2
3	6	5	7
3	6	4	8
3	6	2	8
3	6	1	7
3	6	1	5
3	6	2	4
3	6	4	4
3	6	5	5

Mas o que fazer com esta lista?
R: Procure padrões.

Busca por padrões

Padrões? Como encontrá-los?
R. Por observação e tentativas, partindo de casos específicos.

Vimos anteriormente que o cavalo sempre anda em “L”, ou seja uma casa numa das dimensões e duas casas na outra. Isto deveria chamar a nossa atenção no sentido que possivelmente haja uma relação entre x_1 e x_2, e entre y_1 e y_2. Mas qual? Tentemos subtração pois a frase destacada acima de alguma forma parece remeter para diferença:



x_1	y_1	x_2	y_2	x_1-x_2	y_1-y_2
3	6	5	7	-2	-1
3	6	4	8	-1	-2
3	6	2	8	1	-2
3	6	1	7	2	-1
3	6	1	5	2	1
3	6	2	4	1	2
3	6	4	4	-1	2
3	6	5	5	-2	1

É coincidência que os valores das últimas duas colunas sejam apenas 1, 2, -1 e -2?

Busca por padrões

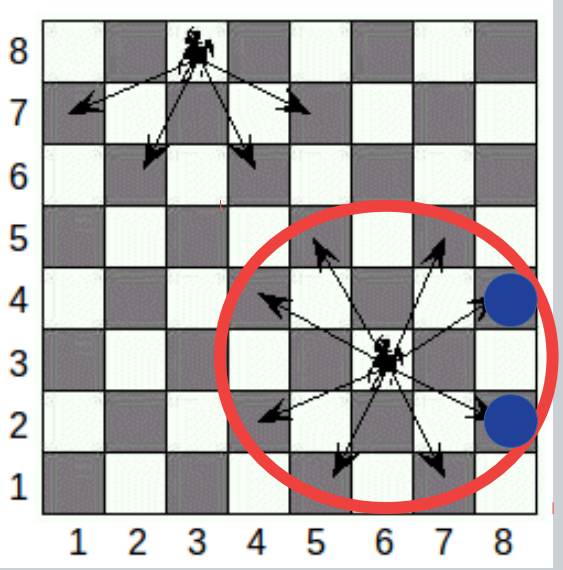
Decididamente não é coincidência o fato dos valores das últimas duas colunas serem apenas 1, 2, -1 e -2, visto que o movimento do cavalo é em “L” (duas casas para um lado, uma para outro), independentemente da casa de origem do movimento. Encontramos, então, um padrão que se repete para qualquer casa, o que nos permite escrever uma segunda versão (ainda ingênua) para o programa, que basicamente replica os 8 casos listados na tabela.

x_1	y_1	x_2	y_2	x_1-x_2	y_1-y_2
3	6	5	7	-2	-1
3	6	4	8	-1	-2
3	6	2	8	1	-2
3	6	1	7	2	-1
3	6	1	5	2	1
3	6	2	4	1	2
3	6	4	4	-1	2
3	6	5	5	-2	1

```
7  # Assumimos, por hipótese,
8  # que o movimento não é valido
9  pode_moverse = False
10
11 # Alteramos a hipótese para os
12 # casos positivos
13 if x1-x2 == -2 and y1-y2 == -1:
14     pode_moverse = True
15 elif x1-x2 == -1 and y1-y2 == -2:
16     pode_moverse = True
17 elif x1-x2 == 1 and y1-y2 == -2:
18     pode_moverse = True
19 elif x1-x2 == 2 and y1-y2 == -1:
20     pode_moverse = True
21 elif x1-x2 == 2 and y1-y2 == 1:
22     pode_moverse = True
23 elif x1-x2 == 1 and y1-y2 == 2:
24     pode_moverse = True
25 elif x1-x2 == -1 and y1-y2 == 2:
26     pode_moverse = True
27 elif x1-x2 == -2 and y1-y2 == 1:
28     pode_moverse = True
```

Busca por padrões

Mas pensando no movimento em “L” do cavalo, se fixarmos uma determinada coluna para a casa de destino, a de nº 8, por exemplo, não parece fazer diferença se o cavalo se move para para a linha de cima ou para a de baixo (valores 1 e -1 indicados na tabela abaixo) uma vez que os dois movimentos são válidos. O que importa é que a diferença deve dar 1 (um), para cima ou para baixo, o que lembra valor absoluto (também conhecido como módulo). Fato similar ocorre se fixarmos uma linha em vez de uma coluna, conduzindo às duas últimas colunas da tabela abaixo.



x_1	y_1	x_2	y_2	x_1-x_2	y_1-y_2	$\lvert x_1-x_2\rvert$	$\lvert y_1-y_2\rvert$
3	6	5	7	-2	-1	2	1
3	6	4	8	-1	-2	1	2
3	6	2	8	1	-2	1	2
3	6	1	7	2	-1	2	1
3	6	1	5	2	1	2	1
3	6	2	4	1	2	1	2
3	6	4	4	-1	2	1	2
3	6	5	5	-2	1	2	1

Busca por padrões

Observando as duas últimas colunas da tabela abaixo, notamos que há apenas dois casos distintos que se repetem ao longo das 8 linhas, os quais são derivados do movimento em “L” do cavalo. Isto conduz a uma terceira versão do programa.

A função “*abs()*” em Python calcula o Valor absoluto.

x_1	y_1	x_2	y_2	x_1-x_2	y_1-y_2	$\|x_1-x_2\|$	$\|y_1-y_2\|$
3	6	5	7	-2	-1	2	1
3	6	4	8	-1	-2	1	2
3	6	2	8	1	-2	1	2
3	6	1	7	2	-1	2	1
3	6	1	5	2	1	2	1
3	6	2	4	1	2	1	2
3	6	4	4	-1	2	1	2
3	6	5	5	-2	1	2	1

```python
# Entrada de dados
x1 = int(input())
y1 = int(input())
x2 = int(input())
y2 = int(input())

# Assumimos, por hipótese,
# que o movimento não é valido
pode_moverse = False

# Alteramos a hipótese para os
# casos positivos
if abs(x1-x2) == 2 and abs(y1-y2) == 1:
    pode_moverse = True
elif abs(x1-x2) == 1 and abs(y1-y2) == 2:
    pode_moverse = True

# Mostrar o resultado
print(pode_moverse)
```

Busca por padrões

Continuando a busca por padrões, é possível notar que a soma da penúltima e antepenúltima colunas resultam num valor constante 3, fato que possibilita uma redução significativa do programa, conforme mostrado abaixo.

x_1	y_1	x_2	y_2	x_1-x_2	y_1-y_2	$\|x_1-x_2\|$	$\|y_1-y_2\|$	$\|x_1-x_2\|+\|y_1-y_2\|$
3	6	5	7	-2	-1	2	1	3
3	6	4	8	-1	-2	1	2	3
3	6	2	8	1	-2	1	2	3
3	6	1	7	2	-1	2	1	3
3	6	1	5	2	1	2	1	3
3	6	2	4	1	2	1	2	3
3	6	4	4	-1	2	1	2	3
3	6	5	5	-2	1	2	1	3

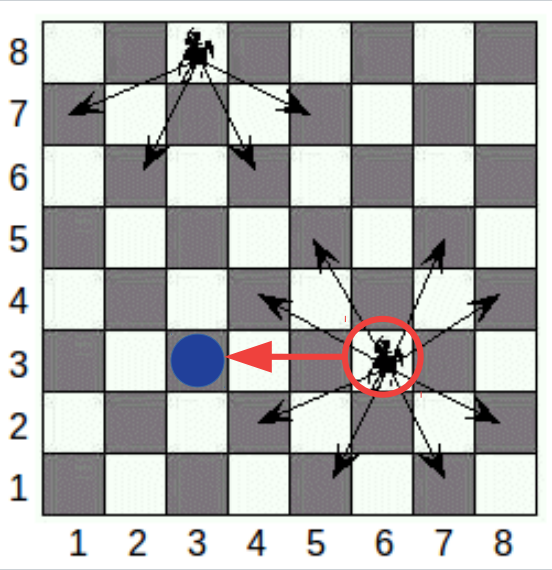
```
1  # Entrada de dados
2  x1 = int(input())
3  y1 = int(input())
4  x2 = int(input())
5  y2 = int(input())
6
7  if abs(x1-x2) + abs(y1-y2) == 3:
8      pode_moverse = True
9  else:
10     pode_moverse = False
11
12 # Mostrar o resultado
13 print(pode_moverse)
```

Apesar de correto o raciocínio, infelizmente esta simplificação acabou introduzindo um erro visto que agora o programa gera resultado "*True*" para entradas como (3, 6) e (3, 3), quando, em realidade, o valor deveria ser "*False*". Por que isto ocorre?

Busca por padrões

O erro introduzido para casos como dos pontos (3, 6) e (3, 3) pode ser entendido se olharmos para o esquema abaixo. Note que estamos tentando mover o cavalo ao longo da mesma linha (um movimento inválido), mas a equação $|x_1-x_2|+|y_1-y_2| = 3$ permanece válida. Fato similar ocorreria se tentássemos movê-lo ao longo da mesma coluna. Para resolver isto é necessário garantir que haja mudança de linha e de coluna nos dois pontos, conforme alteração inserida no programa abaixo, que reflete a equação:

$|x_1-x_2|+|y_1-y_2| = 3$ e $x_1 \neq x_2$ e $y_1 \neq y_2$




```
1  # Entrada de dados
2  x1 = int(input())
3  y1 = int(input())
4  x2 = int(input())
5  y2 = int(input())
6
7  if abs(x1-x2) + abs(y1-y2) == 3 and x1 != x2 and y1 != y2:
8      pode_moverse = True
9  else:
10     pode_moverse = False
11
12 # Mostrar o resultado
13 print(pode_moverse)
```

Melhoria

Note que a expressão destacada na imagem abaixo é a implementação em Python para a equação: $|x_1-x_2|+|y_1-y_2| = 3$ e $x_1 \neq x_2$ e $y_1 \neq y_2$

cujo resultado é um valor lógico *True* ou *False*.

```
6
7  if abs(x1-x2) + abs(y1-y2) == 3 and x1 != x2 and y1 != y2
8      pode_moverse = True
9  else:
10     pode_moverse = False
```

Apesar dela não estar “errada”, na prática ela não faz sentido pois é como se estivéssemos escrevendo:

```
7  if True
8      pode_moverse = True
9  else:
10     pode_moverse = False
```

Neste caso a estrutura de seleção <if-else> não é necessária, podendo a expressão ser escrita simplesmente como:

```
6
7  pode_moverse = abs(x1-x2) + abs(y1-y2) == 3 and x1 != x2 and y1 != y2
```

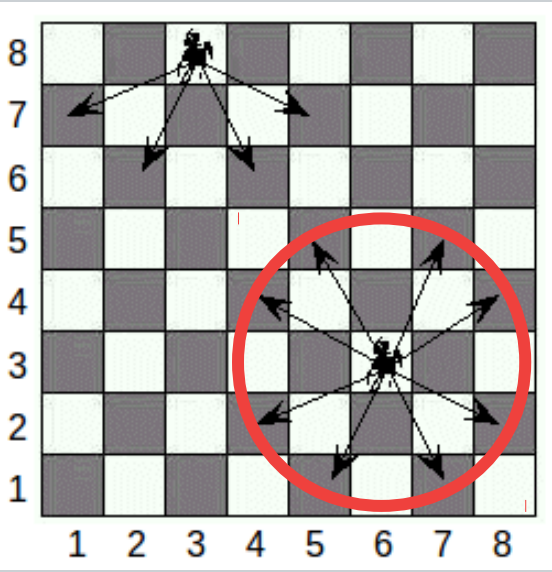
Versão final

A imagem abaixo apresenta a versão final para o programa que verifica se o cavalo pode ser movido entre duas casas (x_1, y_1) e (x_2, y_2):

```
1  # Entrada de dados
2  x1 = int(input())
3  y1 = int(input())
4  x2 = int(input())
5  y2 = int(input())
6
7  # Calcula
8  pode_moverse = abs(x1-x2) + abs(y1-y2) == 3 and x1 != x2 and y1 != y2
9
10 # Mostrar o resultado
11 print(pode_moverse)
```

Outra abordagem

A abordagem até aqui apresentada não é a única possível para a resolução deste problema, nem necessariamente a mais fácil. Para buscarmos outra abordagem, retornemos ao esquema inicial e tentemos "limpar a mente" no sentido de tentar "ignorar" o caminho escolhido e buscar outro tipo de estratégia ou de padrão.

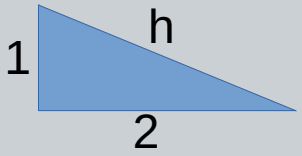


Note que as setas que partem do ponto (3, 6) são todas radiais e parecem ter o mesmo comprimento se tomarmos o centro das casas como referência.

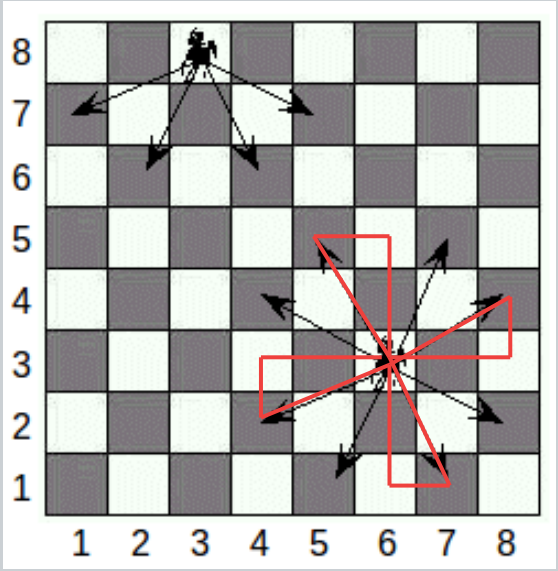
- Isto é de fato um novo padrão?
- Se sim, de alguma forma ele nos indica outro caminho para resolver o problema?
- Isso tem alguma relação com a geometria já que falamos de "radial"?

Outra abordagem

Em realidade, podemos pensar que as setas são as hipotenusas de triângulos retângulos, conforme exemplificado abaixo. Como o cavalo sempre anda em “L” (dois para um lado e um para o outro), as setas do esquema abaixo parecem sempre ser hipotenusas de um triângulo retângulo:



$$h = \sqrt{1^2 + 2^2} = \sqrt{5}$$



Sim, de fato isto é verdade, sendo que o valor de “h” corresponde à distância euclidiana entre os pontos que representam os centros das casas do tabuleiro.

A cálculo acima é condição necessária e suficiente para garantir a possibilidade de movimento entre duas casas, o que leva a um nova abordagem para a resolução do problema.

Outra abordagem

A imagem abaixo apresenta o código desta outra abordagem para o programa que verifica se o cavalo pode ser movido entre duas casas (x_1, y_1) e (x_2, y_2):

```
1  import math
2
3  # Entrada de dados
4  x1 = int(input())
5  y1 = int(input())
6  x2 = int(input())
7  y2 = int(input())
8
9  # Calcula
10 pode_moverse = math.sqrt( (x1-x2)**2 + (y1-y2)**2 ) == math.sqrt(5)
11
12 # Mostrar o resultado
13 print(pode_moverse)
```

Contato

antonio.c.mariani@ufsc.br

CTC/INE

UNIVERSIDADE FEDERAL
DE SANTA CATARINA