MAC2166 Introdução à Computação - Grande áreas Civil, Mecânica e Química

EP3: Jogo dos Ladrilhos (com busca em profundidade - tentativa e erro / novos casos-teste)

Due date: sexta, 28 junho 2019, 11:59
Maximum number of files: 1
Type of work: Individual work

Atenção: não deve ser usada qualquer bibliotecas adicional, e.g., não deve ser usada a biblioteca numpy do *Python*.

Atenção 2: CUIDADO com laco infinito na **opção 2**, vocês precisam "quebrar" a recorrência com a variável para profundidade máxima. Se o sistema responder *falta de memória* ou *tempo esgotado*, provavelmente é **laço infinito** (ou quase), nesse caso, se não conseguir resovler, o melhor é comentar sua opção 2 (e garantir as notas das opções 0 e 1)!!!

1. Jogo dos ladrilhos

No *jogo dos ladrilhos* nós temos uma disposição dos números 0,1,...,n^2 -1 colocados em uma matriz *n* x *n*, com *n*>=*2*. Por exemplo, para *n*=3, uma possível configuração do jogo é mostrada na Figura 1 (com o 0 representado com "espaço em branco"):

1	3	4
8	6	2
7		5

Figura 1: exemplo de configuração de ladrilho 3 por 3.

A localização do espaço vazio (branco), associado ao valor 0, faz parte da configuração, indicando que essa posição pode ser ocupada por outra "peça". O objetivo do jogo é, a partir de uma configuração inicial, por exemplo a configuração acima, chegar a uma configuração final, também definida por uma disposição qualquer dos valores 0,1,...,n^2-1. Por exemplo, uma configuração final poderia ser:

1	2	3
8		4
7	6	5

Figura 2: exemplo de configuração final para a configuração inicial da figura 1.

A solução desse quebra-cabeças é feita por meio de movimentações do espaço em branco para a esquerda, para direita, para cima ou para baixo. Na verdade, o movimento do "branco" é uma troca de posições entre o branco e um dos números vizinhos; por exemplo, mover o branco para a direita, na configuração inicial mostrada na Figura 1, é o mesmo que trocar o branco com o número 5. Portanto temos quatro possíveis movimentos, alguns dos quais não podem ser aplicados em determinadas configurações.
Por exemplo, somente três movimentos (esquerda, cima, direita) podem ser aplicados na configuração inicial acima.

2. Sobre os movimentos

Existem 4 tipos de movimentos, que representaremos por suas letras iniciais (*ordem de movimentação*):

- 'c' = mover o branco para cima (trocar 0 com a peça de cima).
- 'd' = mover o branco para direita (troca 0 com a peça da direita).
- 'b' = mover o branco para baixo (troca 0 com a peça de baixo).
- 'e' = mover o branco para esquerda (trocar 0 com a peça da esquerda).

Assim, uma sequência de movimentos (válidos) poderia ser representada por uma lista de caracteres. Por exemplo, a sequência {'c', 'c', 'b', 'd', 'e'} significa: dois movimentos para cima, depois para baixo, depois para a direita e finalmente para a esquerda.

Antes de realizar o movimento, deve-se verificar se é um **movimento válido**, pois um movimento inválido poderia gerar erro de acesso. Por exemplo, considerando o tabuleiro 3 x 3 da figura 2:

- 'c' ou 'b' ou 'd' ou 'e' é um movimento válido (realizando apenas um deles);
- {'c', 'd', 'b', 'b'} é uma sequência de movimentos válidos (simule);
- {'c', 'c', 'b', 'b'} NÃO é uma sequência de movimentos válidos (erro no segundo movimento - simule);

Importante: o movimento só pode ser realizado se for válido.

3. Tarefa

Sua tarefa é escrever um programa que implemente 3 opções de jogo (0, 1 ou 2), descritas abaixo. **Sugerimos fortemente que implemente seu código de modo incremental** usando os casos de teste a seu favor, ou seja, comece pela *opção 0* e só tente a opção 1 após conseguir fazer com que seu código passe nos 2 testes iniciais do VPL. Depois de conseguir passar nos 2 primeiros testes, comece a implementar a *opção 1* e só inicie a opção 2 após conseguir passar nos teste 3, 4, 5 e 6. Abaixo o que cada opção deve fazer:

Opção 0:

1. Leia a opção de jogo (supor digitado 0);
2. Leia um inteiro $n>=2$ (dimensão do tabuleiro);
3. Leia uma configuração inicial qualquer de uma matriz n x n contendo os números de 0 a n^2 -1;
4. Leia uma configuração final qualquer de uma matriz n x n contendo os números de 0 a n^2 -1;
5. Imprima **SIM** se ambas as matrizes forem idênticas, caso contrário imprima **NAO**.

Opção 1: (só tente essa opção após seu código passar nos 2 primeiros testes do VPL!!!)

1. Leia a opção de jogo (supor digitado 1);
2. Leia um inteiro $n>=2$ (dimensão do tabuleiro);
3. Leia uma configuração inicial qualquer de uma matriz n x n contendo os números de 0 a n^2 -1
4. Leia uma "string" com uma sequência de movimentos (dentre {'c', 'd', 'b', 'e'}) a serem tentados (supor $m>=1$ movimentos);
5. Realize os m movimentos (se todos forem válidos) e ao final imprima a matriz resultante.
   Se o movimento i ($1<=i<=m$) for o primeiro movimento inválido, deve-se imprimir **NAO: i**.

Se você conseguiu fazer com que seu código tenha passado nos 6 primeiros testes do VPL, então pode começar a tentar a opção 3. Lembre-se, sua nota será no máximo aquela nota dada pelo VPL (após fechado o processo de submissão, verificaremos se não houve casos de plágio e não foi usado algum truque para forçar passar nos casos de teste do VPL - em qualquer desses dois casos, a nota será reduzida).

Opção 2: (essa opção envolve busca recursiva em profundidade)

1. Leia a opção de jogo (supor digitado 2);
2. Leia um inteiro $n>=2$ (dimensão do tabuleiro);
3. Leia a profundidade máxima *pmax* (0<=pmax<=n*n);
4. Leia uma configuração inicial (qualquer matriz n x n contendo os números de 0 a n*n -1);
5. Leia uma configuração final (qualquer matriz n x n contendo os números de 0 a n*n -1);
6. Implementar uma busca recursiva em profundidade que siga a ordem de movimentação 'c', 'd', 'b', 'e', e que em cada passo da recursão imprima a matriz após o movimento.
7. A profundidade máxima de recorrência será `pmax` (note que se 0 o programa nada faz). Se o programa, em algum passo recursivo, conseguir a configuração final, deve-se interromper a execução dessa função de busca recursiva, voltar ao programa principal e imprimir SIM e a configuração da matriz obtida.

Se em nenhum passo for conseguida a configuração final, esgotadas todas as opções até a profundidade máxima de recorrência pmax, deve-se voltar ao programa principal e imprimir NAO.

Veja os exemplos de entrada e saída no final do enunciado.

A função da opção 2, que realiza a busca em profundidade, deve obrigatoriamente ter o nome `ladrilho(...)` e ser recursiva, usando o método de *tentativa e erro* (*backtracking*), **obrigatoriamente** da forma: `ladrilho (Mat, Matfim, p, pmax, pos, mov, ListaMov)`.

Essa função deve devolver um valor que indique que encontrou ou não a configuracáo final (e.g. devolver `True` se encontrou a configuração final e `False` em caso contrário). Os dois primeiro parâmetros (`Mat` e `Matfim`) são as matrizes atual (que começa com a inicial) e final (alvo), `p` é nível de recorrência, `pmax` acima definido, `pos` é o par (linha,coluna) de onde está o branco (0), `mov` é último movimento (para evitar desfazer) e `LisatMov` a lista de todos os movimentos realizados (assim, se realizá-los a partir da configuração inicial, deve-se obter a configuração atual - pode usá-lo para testar).

A profundidade é o número de chamadas recursivas. Por exemplo, para a função fatorial recursiva abaixo,

```
Python                              C
def fat :                           int fat (int n) {
  if (n==0) : return 1;               if (n==0) return 1;
  else : return n*fat(n-1)            else return n*fat(n-1); }
```

ao usar `print(fat(4))`, ou `printf("%d\n",fat(4))`, a maior profundidade de recorrência será 5, correspondendo à 5 níveis de recursividade, pois `fat(4)` chama `fat(3)`, que chama `fat(2)`, que chama `fat(1)` e que chama `fat(0)`, como ilustrado abaixo:

```
fat(4) :..................... nivel 0
  4*fat(3) :................. nivel 1
    3*fat(2) :............... nivel 2
      2*fat(1) :............. nivel 3
        1*fat(0) :........... nivel 4
          devolve 1 :........ nivel 5
```

3.1 Sobre a busca em profundidade a ser implementada

A sua função `ladrilho(...)` deve recursivamente verificar se existe uma sequência de movimentações da configuração inicial até a configuração final, usando como profundidade máximo de recursividade `pmax` ("descer" no máximo ao nível de recursividade `pmax` - no exemplo acima o fatorial "desceu" até o nível 5), fazendo primeiramente a movimentação 'c'. Se a partir daí `ladrilho(...)` devolver False (não encontrou a configuração final), deve desfazer o movimento e, novamente de modo recursivo, tentar com 'd', 'b' e 'e', nessa ordem.

Se em alguma movimentação foi possível chegar à configuração final, a função deve agregar a referida movimentação na lista ListaMov e devolver um valor indicando se conseguiu ou não realizar todos os movimento (e.g. True se conseguiu e False em caso contrário).

Para resolver o problema do *Jogo dos Ladrilhos*, o seu programa deve tentar encontrar uma solução (se esta existir) que use o menor número possível de movimentações. Para isso, começamos verificando se existe uma solução de tamanho 1, fazendo a busca com profundidade com p=1; se não achar uma solução, tentamos, com profundidade p=2, e assim por diante, até chegar ao valor limite colocado pelo usuário (*pmax*).

Atenção: sua função **deve** usar sempre o mesmo tabuleiro para testar todas as movimentações, e não criar cópias do tabuleiro (sob o risco de estourar a memória durante a recursão).

4. Entradas e saídas (1a versão completa em 2019/06/07 18H)

A entrada será feita como nos exemplos abaixo. O primeiro valor corresponde ao n, em seguida vêm as n linhas da configuração inicial, seguidas das n linhas da configuração final e por último, o valor da profundidade máxima.

1. Exemplo de execucão para opção 0. Em azul o que o usuário digita e em verde as saídas esperadas.

```
Opcao de jogo: 0
Tamanho do tabuleiro: 3
Tabuleiro inicial: 1 3 4
8 6 2
7 0 5
Tabuleiro final: 1 3 4
8 6 2
7 0 5
SIM
```

2. Exemplo de execucão para opção 0. Em azul o que o usuário digita e em verde as saídas esperadas.

```
Opcao de jogo: 0
Tamanho do tabuleiro: 3
Tabuleiro inicial: 1 3 4
8 6 2
7 0 5
Tabuleiro final: 1 3 4
8 6 2
5 0 7
NAO
```

3. Exemplo de execucão para opção 1. Em azul o que o usuário digita e em verde as saídas esperadas.

```
Opcao de jogo: 1
Tamanho do tabuleiro: 4
Tabuleiro inicial:  1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15  0
Digite seq mov: ebbc
NAO: 2
```

4. Exemplo de execucão para opção 1. Em azul o que o usuário digita e em verde as saídas esperadas.

```
Opcao de jogo: 1
Tamanho do tabuleiro: 4
Tabuleiro inicial:  2  3  4  8
 1  6  7 12
 5 10 11 15
 9 13 14  0
Digite seq mov: eeecccdddbbb
Tabuleiro final: 3  4  8  12
 2  6  7 15
 1 10 11 14
 5  9 13  0
```

5. Exemplo de execucão para opção 2. Em azul o que o usuário digita e em verde as saídas esperadas.

```
Opcao de jogo: 2
Tamanho do tabuleiro: 4
Profundidade maxima: 3
Tabuleiro inicial: 4 14 3 5
13 12 15 8
 9 10 11 6
 1 2 7 0
Tabuleiro final:  1 2 3 4
 5 6 7 8
 9 10 11 12
13 14 15 0
Nao encontrei solucao
```

6. Exemplo de execucão para opção 2. Em azul o que o usuário digita e em verde as saídas esperadas.

```
Opcao de jogo: 2
Tamanho do tabuleiro: 4
Profundidade maxima: 3
Tabuleiro inicial:  2  3  4  8
 1  6  7 12
 5 10 11 15
 9 13 14  0
Tabuleiro final: 2  3  4  8
 1  6  7 12
 5 10  0 11
9 13 14 15
Sucesso!
Movimentos: c e
2 3 4 8
1 6 7 12
5 10 0 11
9 13 14 15
```

VPL

◄ Instruções para entrega de EPs

Jump to...

Fórum para discussão do EP3 ►