Aula 16: 09/OUT/2018

Aulas passadas

Nas aulas passadas tratamos de recursão usando os problemas:

- torres de Hanoi (aula 13): mostrou como o raciocínio recursão auxilia na solução de problemas;
- `fatorialR()` e `fatorialI()` (aula 14): ilustrou o mecanismo recursivo através de um exemplo simples;
- `hanoi()` (aula 14): análise de algoritmos, relações de recorrência, solução exponencial;
- `fibonacciR()` e `fibonacciI()`(aula 14): resolver o mesmo problema várias vezes; *memoization*;
- `binomialR()` e `binomialI()` (aula 14): resolver o mesmo problema várias vezes; *memoization*;
- `maximoR()` (aula 14): exercitou raciocínio recursivo.
- `mdc()` ou `euclides()` (aula 15): mostrou que recursão não é o problema

Hoje

Mais recursão ainda: problema do labirinto (Next: An Army of Cris (2007)) e curvas de Hilbert.

Problema

Dada uma posição em um labirinto, encontrar um caminho até a saída. Cada posição do labirinto é formada por um quadrado que pode estar vazio ou conter uma parede.

Representação do labirinto

Um labirinto pode ser representado, essencialmente, por uma lista de listas de caracteres. Uma posição com um `'+'` indica uma parede e uma posição vazia é representada por um `' '`. Um `'I'` pode indicar a posição inicial.

Vamos supor o labirinto todo murado, exceto pela saída que contém um `'X'`.

```
CAMINHO = 'C'
TENTOU  = '*'
PAREDE  = '#'
BECO    = 'b'
SAIDA   = 'X'
ORIGEM  = 'M'
```

```
    lab = [
        ['#','#','#','#','#','#','#','#','#','#','#'],
        ['#',' ',' ',' ',' ',' ',' ','#',' ',' ','X'],
        ['#',' ','#',' ','#','#',' ','#',' ','#','#'],
        ['#',' ','#',' ',' ',' ',' ','#',' ','#','#'],
        ['#','#','#',' ','#','#',' ','#',' ',' ','#'],
        ['#',' ',' ',' ','#','#',' ',' ',' ',' ','#'],
        ['#','#','#',' ','#','#','#','#','#',' ','#'],
        ['#',' ',' ',' ','#','#',' ',' ','#',' ','#'],
        ['#',' ','#','#',' ',' ','#',' ',' ',' ','#'],
```

```
    ['#',' ',' ',' ',' ',' ','#',' ','#','#','#'],
    ['#','#','#','#','#','#','#',' ','#','#','#']
]
lin_ini, col_ini = 9, 5
lab[lin_ini][col_ini] = ORIGEM
procure_saida(lab, lin_ini, col_ini)
```

Solução

```
#--------------------------------------------------------------------
def procure_saida(lab, lin, col):
    '''(list, int, int) -> bool

    Recebe uma matriz `lab` de strings representando um labirinto e
    uma posição [lin][col] para onde desejamos nos mover.

    Os símbolos encontrados em cada posição de `lab` são

        SAIDA   = 'X'
        TENTOU  = '*'
        PAREDE  = '#'
        BECO    = '-'
        VAZIA   = ' '
        CAMINHO = 'C'
        ORIGEM  = 'M'
    '''

    #----------------------------------------------------------------
    # BASE
    # 1. encontramos a saída
    if lab[lin][col] == SAIDA:
        lab[lin][col] = CAMINHO
        return True

    # 2. 'entramos' em uma parede
    if lab[lin][col] == PAREDE:
        return False

    # 3. chegamos em um posição que já foi examinada
    if lab[lin][col] == TENTOU or lab[lin][col] == BECO:
        return False

    lab[lin][col] = TENTOU
    #----------------------------------------------------------
    # tente cada uma das quatro direções a partir da [lin][col],
    # se necessário: NORTE, OESTE, SUL, LESTE
    encontrou = procure_saida(lab, lin-1, col) or \
                procure_saida(lab, lin,   col-1) or \
                procure_saida(lab, lin+1, col) or \
                procure_saida(lab, lin,   col+1)

    if encontrou:
        lab[lin][col] = CAMINHO
    else:
        lab[lin][col] =  BECO
    return encontrou
```