3ª Prova de Algoritmos e Estruturas de Dados I
26/06/2015

- P: O que será corrigido?
  R: A lógica, a criatividade, a sintaxe, o uso correto dos comandos, a correta declaração dos tipos, os nomes das variáveis, a indentação, o uso equilibrado de comentários no código e, evidentemente, a clareza. A modularidade, o correto uso de funções e procedimentos, incluindo passagem de parâmetros e o bom uso de variáveis locais e globais serão especialmente observados.

- **Introdução:** Suponha que você receba o mapa de um labirinto, no qual existem alguns caminhos que levam da entrada dele para a saída e, como em todo labirinto, alguns caminhos que são becos sem saída. O problema é não apenas encontrar um caminho que leva à saída, mas determinar o menor caminho existente.

  A seguir está um trecho de código em *Pascal* que resolve este problema usando um algoritmo clássico encontrado na literatura da computação. O código apresenta algumas declarações de constantes, tipos e variáveis bem como um programa principal que será detalhado em seguida.

```
Program labirinto;
Const
     MAX = 100;
     PAREDE = -1;
     VAZIO = 0;
     LIN = 1;
     COL = 2;
Type
     labirinto= array [1..MAX,1..MAX] of integer;
     coordenada = array [1..2] of integer;
     fila= array [1..MAX] of coordenada;
Var
     L: labirinto;
     tamL, entrada, saida, elem, vizinho: coordenada;
     F: fila; tamF: integer;
     distancia: integer;
Begin
     iniciaFila(F, tamF);
     distancia := 1;
     lerLabirinto(L, tamL, entrada, saida);

     marcaElemLabirinto(L, saida, distancia);
     insereElemFila (F, tamF, saida);

     repeat
          retiraElemFila(F, tamF, elem);
          distancia := valorElemLabirinto(L, elem) + 1;
          "para cada vizinho a ser marcado de elem"
          begin
               marcaElemLabirinto(L, vizinho, distancia);
               insereElemFila(F, tamF, vizinho);
          end;
     until (filaVazia(F, tamF));

     imprimeMenorCaminho(L, entrada);
End.
```

- **Detalhamento:** Este programa foi concebido para receber uma entrada de dados no seguinte formato: a primeira linha da entrada de dados contém as dimensões de uma matriz de inteiros; a segunda linha contém as coordenadas da entrada do labirinto (uma posição da matriz); a terceira linha contém as coordenadas da saída do labirinto (uma outra posição da matriz); as outras linhas contêm os elementos da matriz propriamente dita, constituída de elementos que podem ser 0 (zero) ou -1 (menos 1). Os zeros representam posições pelas quais se pode "passar" na matriz, enquanto que os -1 representam "paredes". Um exemplo de entrada de dados é:

```
6 12
1 1
6 12
 0  0  0 -1  0  0  0 -1  0 -1  0  0
 0 -1  0 -1  0 -1  0  0  0 -1  0 -1
 0  0  0  0  0 -1  0 -1 -1 -1  0 -1
-1  0 -1 -1  0 -1  0  0  0  0  0  0
 0  0  0  0  0  0  0 -1 -1  0 -1  0
 0 -1  0 -1 -1 -1  0  0 -1  0 -1  0
```

  Note que um labirito pode possuir várias possibilidades de entrada e saída, mas o programa deve encontrar um caminho mínimo entre as coordenadas indicadas no arquivo de entrada passando apenas por elementos da matriz propriamente dita.

  A ideia do algoritmo é marcar cada posição da matriz que não seja uma parede com valores inteiros que representam a distância desta posição para a saída. Para isto, ele inicia marcando a posição da saída com o valor 1 e toda outra posição (L,C) é marcada com o valor K desde que exista algum vizinho já marcado com valor (K-1). Os vizinhos válidos são: o da esquerda (L,C-1), o da direita (L,C+1), o da posição de cima (L-1,C) e o da posição de baixo (L+1,C), desde que estas coordenadas estejam dentro dos limites da matriz.

  Após esta etapa, a matriz acima fica como mostrado na figura seguinte:

```
19 18 17 -1 13 12 11 -1 13 -1  7  8
18 -1 16 -1 14 -1 10 11 12 -1  6 -1
17 16 15 14 13 -1  9 -1 -1 -1  5 -1
-1 15 -1 -1 12 -1  8  7  6  5  4  3
15 14 13 12 11 10  9 -1 -1  6 -1  2
16 -1 14 -1 -1 -1 10 11 -1  7 -1  1
```

- **Implementação:** Para implementar esta ideia, o programa principal, contendo ainda uma linha em pseudo-código, usa o seguinte algoritmo (que depende da correta inicialização das estruturas de dados): marca a posição da saída com o valor 1 (um) e insere seus vizinhos válidos no final de uma fila. Em seguida, até que a fila esvazie completamente, retira uma coordenada do início da fila, descobre qual é a distância K que esta tem da saída e para cada um dos seus vizinhos válidos: (1) anota $K + 1$ no vizinho; e (2) insere o vizinho no final da fila.

- **Menor caminho:** Na última linha do programa, após o laço que marca as distâncias na matriz, existe uma chamada a um procedimento que recebe uma matriz preenchida como descrito acima e encontra o menor caminho para se chegar da entrada do labirinto na sua saída. O menor caminho é encontrado partindo-se da entrada do labirinto e escolhendo sucessivamente em cada etapa o vizinho que tem um valor menor que o atual, até se chegar na saída, que tem o valor 1. Basta anotar de alguma maneira este caminho e em seguida imprimí-lo.

- **Questão única (100 pontos):** Sem fazer este último procedimento, sua tarefa é refinar o programa principal e implementar todas as outras funções e procedimentos.

- **Questão bônus (10 pontos extra):** Da maneira como você achar conveniente, implemente o procedimento que imprime o menor caminho.