Algoritmos e Programação I: Aula 37*

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
79070-900 Campo Grande, MS
`http://ava.ufms.br`

Sumário

1 Resolvendo problemas com Matrizes - Exemplos

O material desta aula está ainda numa fase preliminar de preparação.

Vimos anteriormente como utilizar listas para implementar soluções computacionais para problemas cujos dados eram dados por uma tabela. Vimos também que podemos utilizar listas para representar matrizes e resolver problemas modelados com matrizes. Vamos agora continuar a solução de problemas envolvendo operações com matrizes. Utilizaremos listas para representar as matrizes. Esses problemas também podem ser resolvidos com a utilização da biblioteca `numpy`. Como o nosso objetivo é a manipulação individual dos elementos das matrizes, vamos usar apenas as funções de manipulação de listas.

1.1 Campo Minado

Vamos agora abordar o problema do Campo Minado. Nesse problema um mini campo minado $m\ timesn$ é gerado e o usuário deve selecionar coordenadas (x, y), $1 \leq x \leq m$ e $1 \leq y \leq n$ e o programa verifica se na posição (x, y) tem uma bomba. O usuário seleciona coordenadas até encontrar uma bomba ou acertar 3 posições em que não tenha bombas no campo minado.

Um mini campo minado pode ser interpretado como um conjunto de coordenadas (x, y), onde cada coordenada representa uma posição do campo minado e armazenam os números inteiros 0 ou 1. O número (0) indica que a posição está livre e o número (1) indica que há uma bomba no local. Como no exemplo:

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas da disciplina.

$$Campo = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Nesse caso, as coordenadas $(2,2)$, $(2,4)$, $(3,4)$, $(4,1)$, $(4,2)$ e $(5,4)$ possuem bombas. O objetivo do jogo é dada uma coordenada (x,y), verificar se na posição tem uma bomba ou não. O jogador ganha se conseguir fazer três tentativas e não encontrar nenhuma bomba.

Vamos descrever uma solução que implementa uma solução para o problema do mini campo minado. A nossa solução usa uma lista para representar o conjunto de coordenadas do campo. No exemplo acima temos:

```
campo = [[0, 0, 0, 0], [0, 1, 0, 1], [1, 1, 1, 0], [1, 1, 0, 0],
         [0, 0, 0, 1]]
```

Abaixo uma sugestão de estrutura de dados em Python para resolver o problema.

```
# descrição das variáveis utilizadas
# list   campo[[]] - lista de listas para representar campo
# int    m, n - dimensões do campo minado
# int    x, y - coordenadas selecionadas
# int    acertos - número de acerto do jogador
# bool   bomba - indica que tem uma bomba
# string msg - mensagem ao jogador
```

Para tornar o jogo mais atrativo vamos gerar o campo minado de forma aleatória. Quando utilizamos a função `random.randint(0,1)`, temos a ela gera um número inteiro 0 ou 1 com igual probabilidade. Queremos gerar as posições das bombas com uma probabilidade menor. Para isso podemos usar a função `random.choices()`:

```
valores = [0,1]
random.choices(valores,weights=[7,3],k=num)
```

que gera uma lista de `num` elementos onde a probabilidade do elemento ser 0 é 0.7 e a de ser 1 é 0.3.

Para garantir que a cada execução o campo seja diferente, usamos a semente do relógio do computador (`random.seed()`. Para usar essas funções necessitamos importar o módulo `random`.

```
# Passo 1. Crie o campo minado
# Passo 1.1. Leia as dimensões do campo
lin,col = map(int,input().split())
# Passo 1.2. Gere o campo minado
random.seed()
valores = [0,1]
campo = [random.choices(valores,weights=[7,3],k=col) for i in range(lin)]
```

Dessa forma, não precisamos inicializar a lista `campo`. A leitura da entrada no `Passo 1` resume-se a apenas ler as dimensões da lista `campo`.

Após a geração do campo minado temos que ler as tentativas do jogador. As tentativas são lidas até que o jogador encontre uma bomba ou consiga três tentativas sem achar uma bomba.

Usamos as variáveis `acertos` e `bomba` para controlar a execução do jogo. A cada tentativa do jogador que não encontra uma bomba, incrementamos a variável `acertos`. A variável booleana `bomba` indica se até o momento o jogador encontrou uma bomba. Inicializamos as variáveis com 0 e `False`, respectivamente. Usamos um laço `while` que será executado para ler as tentativas do jogador enquanto não encontrar uma bomba ou o número de acertos for menor que 3. A entrada da tentativa é feita com a leitura de um par de inteiros.

```
# Passo 2. Inicie o jogo
# Passo 2.1. Inicialize as variáveis
bomba = False # antes do início não encontrou bomba
acertos = 0
# Passo 2.2. Enquanto bomba = 0 e acertos < 3  faça
while not bomba and acertos < 3:
# Passo 2.2.1. Leia as coordenadas da tentativa
   x,y = map(int,input().split())
```

Após a tentativa do jogador o programa tem que analisar a coordenada no campo minado. As coordenadas `x` e `y` variam de 1 a `lin` e de 1 a `col`, respectivamente, mas os índices da lista `campo` variam de 0 a `lin -1` (linhas) e `col - 1` (colunas). Em função disso temos que testar `x - 1` para linhas e `y - 1` para as colunas do campo minado. Se a `x` e `y` tem uma bomba, o programa atribui `bomba = True` e com isso o laço será finalizado. Se a posição não possui uma bomba, o programa atualiza a variável `acertos = acertos + 1` e atribui o número 2 para a posição da tentativa. Pode ocorrer de o jogador entrar com uma coordenada já tentada anteriormente e que não possui uma bomba. Para evitar que ele ganhe o jogo com apenas uma coordenada, o programa informa que essa posição já foi testada e aguarda uma nova entrada. Se o número de acertos for igual a 3, o laço é encerrado.

```
# Passo 2.2.2. Verifique se a posição tem bomba
   if campo[x-1][y-1] == 1:
      bomba = True
      print('BOMBA!!!')
# Passo 2.2.3. Se a posição não é bomba
   elif campo[x-1][y-1] == 0:
      print('ACERTOU!!')
      acertos = acertos + 1
      campo[x-1][y-1] = 2
# Passo 2.2.4. Posição já testada
   else:
      print('POSIÇÃO JÁ TESTADA!!!')
```

Após a saída do laço o programa verifica o resultado do jogo (se o jogador ganhou ou perdeu). Isso é feito com a verificação do número de acertos do jogo.

```
# Passo 2.2.5. Verifique se o jogador ganhou o jogo
if acertos == 3:
```

```
   msg = 'GANHOU!!!'
else:
   msg = 'PERDEU!!!'
```

A saída imprime a mensagem (**msg**) se o jogador ganhou ou perdeu e a situação atual do campo minado. de acordo com o formato abaixo (exemplo de um campo 7 x 5):

```
    1  2  3  4  5
 1 [1, 0, 0, 1, 0]
 2 [0, 1, 0, 0, 1]
 3 [1, 2, 0, 0, 0]
 4 [0, 1, 0, 0, 1]
 5 [1, 0, 0, 1, 2]
 6 [0, 1, 1, 0, 0]
 7 [0, 1, 0, 2, 0]
```

Para obter esse formato de saída temos que usar propriedades da função `print()`, além de projetar como ficam os espaços entre os elementos em cada uma das linhas da saída. Isso pode ser obtido com o conjunto de instruções abaixo:

```
# Passo 3. Imprima o resultado
# Passo 3.1. Imprima a mensagem
print(msg)
# Passo 3.2. Imprima o campo minado resultante
print('  ',end='')  # imprime dois espaços e não muda de linha
for i in range(colunas):  # cabeçalho superior
   print(':3d'.format(i+1), end='')
print()  # começa uma nova linha
for i in range(linhas): # número da linha e campo[i]
   print('0:2d'.format(i+1),campo[i])
```

Abaixo apresentamos um programa em Python para resolver o problema do campo minado.

```
1  # -*- coding: utf-8 -*-
2  # Programa: campominado.py
3  # Programador:
4  # Data: 16/11/2019
5  # Este programa lê as dimensão e gera um mini campo minado. O
6  # programa lê as tentativas do usuário e encerra o programa se a
7  # tentativa encontrou uma bomba ou se o usuário conseguiu achar 3
8  # posições em bombas. O programa imprime uma mensagem
9  # informado o resultado.
10 # módulos utilizados
11 import random
12 # descrição das variáveis utilizadas
13 # list  campo[[]] - lista de listas para representar campo
14 # int   m, n - dimensões do campo minado
15 # int   x, y - coordenadas selecionadas
```

```
# int   acertos - número de acerto do jogador
# bool  bomba - indica que tem uma bomba
# str   msg - mensagem ao jogador

# pré: lin col

# Passo 1. Crie o campo minado
# Passo 1.1. Leia as dimensões do campo
lin,col = map(int,input().split())
# Passo 1.2. Gere o campo minado
random.seed()
valores = [0,1]
campo = [random.choices(valores,weights=[7,3],k=col) for i in range(lin)]
# Passo 2. Inicie o jogo
# Passo 2.1. Inicialize as variáveis
bomba = False # antes do início não encontrou bomba
acertos = 0
# Passo 2.2. Enquanto bomba = 0 e acertos < 3  faça
while not bomba and acertos < 3:
# Passo 2.2.1. Leia as coordenadas da tentativa
   x,y = map(int,input().split())
# Passo 2.2.2. Verifique se a posição tem bomba
   if campo[x-1][y-1] == 1:
      bomba = True
      print('BOMBA!!!')
# Passo 2.2.3. Se a posição não é bomba
   elif campo[x-1][y-1] == 0:
      print('ACERTOU!!')
      acertos = acertos + 1
      campo[x-1][y-1] = 2
# Passo 2.2.4. Posição já testada
   else:
      print('POSIÇÃO JÁ TESTADA!!!')
# Passo 2.2.5. Verifique se o jogador ganhou o jogo
if acertos == 3:
   msg = 'GANHOU!!!'
else:
   msg = 'PERDEU!!!'
# Passo 3. Imprima o resultado
# Passo 3.1. Imprima a mensagem
print(msg)
# Passo 3.2. Imprima o campo minado resultante
print('  ',end='')  # imprime dois espaços e não muda de linha
for i in range(colunas):  # cabeçalho superior
   print(':3d'.format(i+1), end='')
print()  # começa uma nova linha
for i in range(linhas): # número da linha e campo[i]
   print('0:2d'.format(i+1),campo[i])

# pós: 'GANHOU!!!' && acertos == 3 || 'PERDEU!!!'
```

```
66 #       para i em 0,..,lin-1: para j em 0,..,col-1: campo[i][j]
67 # fim
```

O programa pode ser executado com:

```
$ python transposta.py
```

neste caso, as jogadas e as saídas ficam:
Exemplo

```
Exemplo
5 5
ACERTOU!!!
3 2
ACERTOU!!!
7 4
ACERTOU!!!
GANHOU!!!
    1  2  3  4  5
 1 [1, 0, 0, 1, 0]
 2 [0, 1, 0, 0, 1]
 3 [1, 2, 0, 0, 0]
 4 [0, 1, 0, 0, 1]
 5 [1, 0, 0, 1, 2]
 6 [0, 1, 1, 0, 0]
 7 [0, 1, 0, 2, 0]
```

1.2 Vendas

A companhia de Arreios Mula Brava tem uma linha de produtos com cinco itens cujos preços de venda são R$ 100, R$ 75, R$ 120, R$ 150 e R$ 35. Existem quatro vendedores trabalhando para a companhia, e a tabela abaixo fornece o relatório de vendas referente a uma semana:

Número do Vendedor	**Número Item**				
	1	**2**	**3**	**4**	**5**
1	10	4	5	6	7
2	7	0	12	1	3
3	4	9	5	0	8
4	3	2	1	5	6

Usando os dados do relatório de vendas acima, projete e implemente um programa que compute o total de reais faturados por cada vendedor.

Para resolver esse problema, é necessário fazer uma multiplicação da matriz de vendas (4 x 5) pelo vetor de preços (5 x 1). O resultado será um vetor (4 x 1) com os valores das vendas de cada vendedor.

$$\begin{bmatrix} 10 & 4 & 5 & 6 & 7 \\ 7 & 0 & 12 & 1 & 3 \\ 4 & 9 & 5 & 0 & 8 \\ 3 & 2 & 1 & 5 & 6 \end{bmatrix} \begin{bmatrix} 100.00 \\ 75.00 \\ 120.00 \\ 150.00 \\ 35.00 \end{bmatrix} = \begin{bmatrix} 3045.00 \\ 2395.00 \\ 1955.00 \\ 1530.00 \end{bmatrix}$$

$$\begin{bmatrix} 10 * 100.00 + 4 * 75.00 + 5 * 120.00 + 6 * 150.00 + 7 * 35.00 \\ 7 * 100.00 + 0 * 75.00 + 12 * 120.00 + 1 * 150.00 + 3 * 35.00 \\ 4 * 100.00 + 9 * 75.00 + 5 * 120.00 + 0 * 150.00 + 8 * 35.00 \\ 3 * 100.00 + 2 * 75.00 + 1 * 120.00 + 5 * 150.00 + 6 * 35.00 \end{bmatrix} = \begin{bmatrix} 3045.00 \\ 2395.00 \\ 1955.00 \\ 1530.00 \end{bmatrix}$$

Como nos exemplos anteriores, vamos usar listas para representar a tabela de vendas (`vendas`) e a tabela de preços (`precos`).

```
vendas = [[10, 4, 5, 6, 7], [7, 0, 12, 1, 3], [4, 9, 5, 0, 8],
          [3, 2, 1, 5, 6]]

precos = [100.00, 75.00, 120.00, 150.00, 35.00]
```

A solução é dada pela lista `total`:

```
total = [3045.00, 2395.00, 1955.00, 1530.00]
```

Considerando as listas acima, abaixo uma sugestão de estrutura de dados em Python para resolver o problema.

```
# descrição das variáveis utilizadas
# list   vendas[[]] - lista de listas para representar as vendas
# list    precos[] - lista para representar a tabela de preços
# list    total[] - lista para representar as vendas totais
# int    lin, col - dimensões da tabela de vendas
```

Uma vez definida as estruturas de dados para armazenar as informações necessárias para a solução do problema, temos que especificar o formato da entrada. Obs: Os comentários não fazem parte da entrada, são apenas para facilitar a compreensão da entrada.
Exemplo

```
# formato da entrada
4 5 # tamanho da tabela (linhas, colunas)
10 4 5 6 7 # linha 1 - vendas vendedor 1
7 0 12 1 3 # linha 2 - vendas vendedor 2
4 9 5 0 8 # linha 3 - vendas vendedor 3
3 2 1 5 6 # linha 4 - vendas vendedor 4
100.00 75.00 120.00 150.00 35.00 # tabela com os valores de venda
```

Com a forma de entrada especificada e as estruturas definidas para armazenar os dados, podemos especificar as instruções em Python para inicializar as estruturas e ler os dados:

```
# Passo 1. Inicialize as estruturas e leia os dados
# Passo 1.1. Leia as dimensões da tabela
lin,col = map(int,input().split())
# Passo 1.2. Inicialize a tabela de vendas
vendas = [[0]*col for i in range(lin)] # lin linhas, col colunas
# Passo 1.3. Inicialize a lista de preços
precos = [0.0]*col # col preços
# Passo 1.3. Inicialize a lista dos totais
total = [0]*lin # lin vendedores
# Passo 1.4. Leia a tabela vendas (linha a linha)
for i in range(0,lin):
   vendas[i] = list(map(int, input().split())) # leia a linha i de vendas
# Passo 1.5. Leia a lista de preços
precos = list(map(float, input().split()))[:col]
```

Considerando o nosso exemplo, onde `lin == 4` e `col == 5`, os elementos das listas `vendas` e `precos` são dados por:

```
vendas=[[vendas[0][0],vendas[0][1],vendas[0][2],vendas[0][3],vendas[0][4]],
        [vendas[1][0],vendas[1][1],vendas[1][2],vendas[1][3],vendas[1][4]],
        [vendas[2][0],vendas[2][1],vendas[2][2],vendas[2][3],vendas[2][4]],
        [vendas[3][0],vendas[3][1],vendas[3][2],vendas[3][3],vendas[3][4]]]
precos = [precos[0], precos[1], precos[2], precos[3], precos[4]]
```

Para computar o total de vendas do vendedor 1, temos que multiplicar as quantidades vendidas de cada item do vendedor 1 pelos respectivos preços dos itens.

```
total[0] = vendas[0][0]*precos[0] + vendas[0][1]*precos[1] +
           vendas[0][2]*precos[2] + vendas[0][3]*precos[3] +
           vendas[0][4]*precos[4]
```

De uma forma geral, para o vendedor `i+1` temos:

```
total[i] = vendas[i][0]*precos[0] + vendas[i][1]*precos[1] +
           vendas[i][2]*precos[2] + vendas[i][3]*precos[3] +
           vendas[i][4]*precos[4]
```

Um implementação da computação dos totais vendidos por cada vendedor pode ser dado por:

```
# Passo 2. Compute o total de vendas
for i in range(lin):  # total de vendas do vendedor i+1
   total[i] = 0.0
   for j in range(col): # compute o valor das vendas do item j+1
      total[i] = total[i] + vendas[i][j]*precos[j]
```

Uma vez calculado os valores totais das vendas de cada vendedor (`vendas`), falta imprimir o resultado de acordo com o formato abaixo:

```
# formato da saída
3045.00
2395.00
1955.00
1530.00
```

Essa saída pode ser obtida por:

```
# Passo 3. Imprima o resultado
for i in range(lin):
   print(':.2f'.format(total[i]))
```

Abaixo apresentamos um programa em Python para resolver o problema das vendas.

```
# -*- coding: utf-8 -*-
# Programa: vendas.py
# Programador:
# Data: 16/11/2019
# Este programa lê uma tabela com as vendas semanais de uma
# uma empresa feitas por um conjunto de vendedores e os preços dos
# produtos comercializados. O programa computa e imprime o total
# vendido por cada vendedor.
# início do módulo principal
# list   vendas[[]] - lista de listas para representar as vendas
# list   precos[] - lista para representar a tabela de preços
# list   total[] - lista para representar as vendas totais
# int    lin, col - dimensões da tabela de vendas

# pré: lin col vendas[0][0] vendas[0][1]..vendas[lin-1][col-1]
#      preco[0]..preco[5]

# Passo 1. Inicialize as estruturas e leia os dados
# Passo 1.1. Leia as dimensões da tabela
lin,col = map(int,input().split())
# Passo 1.2. Inicialize a tabela de vendas
vendas = [[0]*col for i in range(lin)] # lin linhas, col colunas
# Passo 1.3. Inicialize a lista de preços
precos = [0.0]*col # col preços
# Passo 1.3. Inicialize a lista dos totais
total = [0]*lin # lin vendedores
# Passo 1.4. Leia a tabela vendas (linha a linha)
for i in range(0,lin):
   vendas[i] = list(map(int, input().split())) # leia a linha i de vendas
# Passo 1.5. Leia a lista de preços
precos = list(map(float, input().split()))[:col]
# Passo 2. Compute o total de vendas
for i in range(lin):  # total de vendas do vendedor i+1
   total[i] = 0.0
```

```
35      for j in range(col): # compute o valor das vendas do item j+1
36          total[i] = total[i] + vendas[i][j]*precos[j]
37  # Passo 3. Imprima o resultado
38  for i in range(lin):
39      print(':.2f'.format(total[i]))
40
41  # pós: total && para i em 0,..,lin-1: total[i] ==
42  #       sum j em 0,..,col-1: vendas[i][j]*precos[j]
43  # fim do módulo principal
```

A entrada é dada por um bloco de linhas. A primeira linha do bloco contém dois números inteiros (lin e col) indicando o tamanho (formato) da tabela (lin – linhas e col – colunas), seguido de lin linhas da tabela representando as vendas de cada um dos $i + 1$ vendedores $0 \leq i < lin$ para cada um dos col itens comercializados, seguido de uma linha com os valores de venda de cada um dos col itens. A saída consiste em imprimir um conjunto de $0 < i \leq m$ linhas, onde a i-ésima linha contém o número do vendedor com sua respectiva venda total semanal.

Você pode criar um arquivo para testar o seu programa. No nosso caso, vamos denominar o arquivo com `vendas.in`. O programa pode ser executado com:

```
$ python vendas.py < vendas.in
```

Exemplo

```
# formato da entrada
4 5
10 4 5 6 7
7 0 12 1 3
4 9 5 0 8
3 2 1 5 6
100.00 75.00 120.00 150.00 35.00

# formato da entrada
3045.00
2395.00
1955.00
1530.00
```