Algoritmos e Programação I: Aula 36 (Prev.)*

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

Sumário

1 Resolvendo Problemas de Matrizes com Listas

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

1.1 Industria Alimentícia - 2

A empresa Mandioca & Cia tem uma linha de produtos baseada em derivados de mandioca. Ela produz talharim, nhoque e mandioca para fritura. A empresa possui duas fábrica, uma em Coxim e outra em Três Lagoas. A Mandioca & Cia possui 4 depósitos para distribuições de seus produtos no estado. Os depósitos ficam nas cidades de Campo Grande, Dourados, Corumbá e Naviraí. O custo para transportar uma tonelada dos produtos das duas unidades fabris para cada um dos depósitos é dado pela matriz abaixo:

$$Custo = \begin{array}{c} \\ \text{Três Lagoas} \\ \text{Coxim} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Campo Grande} & \text{Corumbá} & \text{Dourados} & \text{Naviraí} \end{array} \\ \begin{pmatrix} 20.1 & 47.3 & 38.2 & 41.1 \\ 18.1 & 40.2 & 35.3 & 60.1 \end{pmatrix} \end{array}$$

Os depósitos de Campo Grande, Corumbá, Dourados e Naviraí precisam repor seus respectivos estoques. A soma dos pesos produtos (talharim, nhoque e fritas) para repor os estoques é 120, 50, 80 e 40 toneladas, para os depósitos de Campo Grande, Corumbá, Dourados e Naviraí, respectivamente. A carga deve ser única, isto é, sair de uma única unidade fabril e a Mandioca & Cia deseja gastar o mínimo possível com o transporte. Compute o custo do transporte de determinar qual é a melhor opção de transporte. Esses dados podem ser representados pelo vetor

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

$$Quantidades = \begin{array}{l} \\ \text{Campo Grande} \\ \text{Corumbá} \\ \text{Dourados} \\ \text{Naviraí} \end{array} \begin{array}{c} \text{Toneladas} \\ \left(\begin{array}{c} 120 \\ 50 \\ 80 \\ 40 \end{array} \right) \end{array}$$

Para solucionarmos o problema acima, temos que efetuar uma multiplicação da matriz (representado pela lista) `custo` com o vetor (representado com a lista) das `quantidades` e verificar no vetor resultante (representado por uma lista) qual é o menor valor.

Como nos exemplos anteriores, vamos usar listas para representar a tabela (matriz) dos custos do transporte (`custo`) e a tabela (vetor) das quantidades (`quantidades`).

```
custos = [[20.1, 47.3, 38.2, 41.1], [18.1, 40.2, 35.3, 60.1]]

quantidade = [120, 50, 80, 40]
```

e a solução é dada pela lista `total`:

```
total = [9477.00, 9410.00]
```

onde o primeiro elemento da lista refere-se ao custo de transportar os itens da fábrica de "Três Lagoas" e o segundo elemento da lista da fábrica de "Coxim". Calculamos o menor valor da lista `total` e a fábrica associada. Neste caso teremos:

```
9410.0
```

e nesse caso, a melhor opção é transportar os itens da fábrica de Coxim.

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

```
# descrição das variáveis utilizadas
# list   custos[[]] - lista de listas para representar os custos
# list   quantidades[] - lista para representar a tabela de quantidades
# list   total[] - lista para representar os custos totais
# float  valor - valor mínimo do transporte
# string unidade - unidade de produção
# int    lin, col - dimensões da tabela de custos
```

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
2 4 # tamanho da tabela (linhas, colunas)
20.1 47.3 38.2 41.1 # linha 1 - custos do transporte da fábrica 1
18.1 40.2 35.3 60.1 # linha 2 - custos do transporte da fábrica 2
120 50 80 40 # tabela com as unidades a serem transportadas
```

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 custos
custos = [[0.0]*col for i in range(lin)] # lin linhas, col colunas
# Passo 1.3. Inicialize a lista das quantidades
precos = [0]*col # col quantidades
# Passo 1.3. Inicialize a lista dos totais
total = [0.0]*lin # lin fábricas
# Passo 1.4. Leia a tabela custos (linha a linha)
for i in range(0,lin):
   custos[i] = list(map(float, input().split())) # leia a linha i de custos
# Passo 1.5. Leia a lista das quantidades
quantidades = list(map(int, input().split()))[:col]
```

Considerando o nosso exemplo, onde `lin == 2` e `col == 4`, os elementos das listas `custos` e `quantidades` são dados por:

```
vendas=[[custos[0][0],custos[0][1],custos[0][2],custos[0][3]],
        [custos[1][0],custos[1][1],custos[1][2],custos[1][3]]]
quantidades = [quantidades[0],quantidades[1],quantidades[2],quantidades[3]]
```

Para computar o custo total do transporte da fábrica 1, temos que multiplicar os preços do transporte para cada um dos depósitos a partir da fábrica 1, pelas respectivas quantidades que serão transportadas.

```
total[0] = custos[0][0]*quantidades[0] + custos[0][1]*quantidades[1] +
           custos[0][2]*quantidades[2] + custos[0][3]*quantidades[3]
```

De uma forma geral, para a fábrica `i+1` temos:

```
total[i] = custos[i][0]*quantidades[0] + custos[i][1]*quantidades[1] +
           custos[i][2]*quantidades[2] + custos[i][3]*quantidades[3]
```

Um implementação da computação dos custos totais de cada fábrica pode ser dado por:

```
# Passo 2. Compute o custo total
for i in range(lin):  # custo total transporte da fábrica i+1
   total[i] = 0.0
   for j in range(col): # compute o valor das vendas do item j+1
      total[i] = total[i] + custos[i][j]*quantidades[j]
```

Uma vez calculado os custos totais do transporte a partir de cada fábrica (`custos`), temos que computar o menor custo e de que fábrica deverão ser transportados os alimentos. Podemos obter com as seguintes instruções:

```
# Passo 2.2. Compute de que unidade fabril será feito o transporte
valor = min(total) # computa o menor valor da lista
if total.index(valor) == 0: # computa o índice do menor valor da lista
   msg = 'Três Lagoas'
else:
   msg = 'Coxim'
```

No nosso caso, a lista `total` será `[9477.0, 9410.0]`, o menor valor é `9410.0` referente a fábrica de Coxim. Falta imprimir o resultado de acordo com o formato abaixo:

```
9410.0 Coxim

```

Essa saída pode ser obtida por:

```
# Passo 3. Imprima o resultado
print(valor, msg)
```

Abaixo apresentamos um programa em Python para resolver o problema do custo do transporte.

```
# -*- coding: utf-8 -*-
# Programa: transporte.py
# Programador:
# Data: 16/11/2019
# Este programa lê uma tabela com os custos do transporte e a
# quantidade a ser transportada entre duas fábricas e os depósitos de
# uma dada industria. O programa computa qual o menor custo do
# transporte e de qual fábrica será feito o transporte.
# início do módulo principal
# descrição das variáveis utilizadas
# list   custos[[]] - lista de listas para representar os custos
# list   quantidades[] - lista para representar a tabela de quantidades
# list   total[] - lista para representar os custos totais
# float  valor - valor mínimo do transporte
# string unidade - unidade de produção
# int    lin, col - dimensões da tabela de custos

# pré: lin col custos[0][0] custos[0][1]..custos[lin-1][col-1]
#      quantidade[0]..quantidade[col-1]

# 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 custos
custos = [[0.0]*col for i in range(lin)] # lin linhas, col colunas
# Passo 1.3. Inicialize a lista das quantidades
precos = [0]*col # col quantidades
```

```
# Passo 1.3. Inicialize a lista dos totais
total = [0.0]*lin # lin fábricas
# Passo 1.4. Leia a tabela custos (linha a linha)
for i in range(0,lin):
   custos[i] = list(map(float, input().split())) # leia a linha i de custos
# Passo 1.5. Leia a lista das quantidades
quantidades = list(map(int, input().split()))[:col]
# Passo 2. Compute o custo total
for i in range(lin):  # custo total transporte da fábrica i+1
   total[i] = 0.0
   for j in range(col): # compute o valor das vendas do item j+1
      total[i] = total[i] + custos[i][j]*quantidades[j]
# Passo 2.2. Compute de que unidade fabril será feito o transporte
valor = min(total) # computa o menor valor da lista
if total.index(valor) == 0: # computa o índice do menor valor da lista
   msg = 'Três Lagoas'
else:
   msg = 'Coxim'
# Passo 3. Imprima o resultado
print(valor, msg)

# pós: valor == min{total[0],total[1]} and para i em {0,1} total[i] =
#      {0<= j < 4}: custos[i][j]*quantidades[j]
# fim do módulo principal
```

A entrada e 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 custos de transporte de cada uma das $i+1$ fábricas $0 \leq i < lin$ para cada um dos *col* depósitos, seguido de uma linha com os valores de cada um dos *col* itens a serem transportados. A saída consiste em imprimir o menor custo de transporte entre uma das fábricas e os depósitos e o nome da cidade da fábrica.

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

```
$ python transporte.py < transporte.in
```

Exemplo

```
# formato da entrada
2 4
20.1 47.3 38.2 41.1
18.1 40.2 35.3 60.1
120 50 80 40

# formato da saída
9410.0 Coxim
```

1.2 Matriz de Permutação

Dizemos que uma matriz de números inteiros $n \times n$ é uma matriz de permutação se em cada linha e em cada coluna houver $n - 1$ elementos nulos e um único elemento 1. Abaixo, um exemplo de uma matriz de permutação 4×4:

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

Projete e implemente um programa que leia uma matriz de tamanho $n \times n$, compute se a matriz é de permutação. A entrada é dada por um conjunto de linhas. A primeira linha indica a dimensão n matriz $n \times n$, e as próximas n linhas representam as linhas da matriz. A saída é dada por uma string `'P'` caso a matriz seja de permutação e `'N'` caso contrário. Obs: Os comentários não fazem parte da entrada, estão apenas tentando facilitar a compreensão do exercício.

Exemplo 1

```
# formato da entrada
4 # tamanho da matriz seguido da matriz.
0 1 0 0
0 0 1 0
1 0 0 0
0 0 0 1

# formato da saída
P
```

Exemplo 2

```
# formato da entrada
3 # tamanho da matriz seguido da matriz.
2 -1 0
-1 2 0
0 0 1

# formato da saída
N
```

```
# -*- coding: utf-8 -*-
# Programa: permutacao.py
# Programador:
# Data: 16/11/2019
# Este programa lê uma matrize de tamanho n x n, e imprime 1 caso a
# matriz seja de permutação e 0 caso contrário.
# início do módulo principal
# descrição das variáveis utilizadas
# list mat[[]]
# int n, linha, coluna, i
# str msg
# bool permutacao
```

```
13
14 # pré: n mat
15 # Passo 1. Leia a matriz de entrada
16 # Passo 1.1. Leia as dimensões da matriz
17 n = int(input())
18 # Passo 1.2. Inicialize a estrutura de dados
19 mat = [[0]*n for _ in range(n)]
20 # Passo 1.3. Leia os elementos da matriz (linha a linha)
21 for i in range(n):
22    mat[i] = list(map(int,input().split()))
23 # Passo 2. Verifique se a matriz é de permutação
24 # Passo 2.1. Inicialize as variáveis
25 permutacao = True
26 msg = 'P'
27 i = 0
28 # Passo 2.2. Verifique as linhas e colunas da matriz
29 while i < n and permutacao:
30 for i in range(n):
31    linha = 0 # contar o número de 1´s nas linhas
32    coluna = 0 # contar o número de 1´s nas colunas
33 # Passo 2.2.1. Para linha i coluna j faça
34    for j in range(n):
35 # Passo 2.2.1.1 Verifique os elementos da linha i
36       if mat[i][j] != 0 and mat[i][j] != 1:
37          linha = 2 # tem um número diferente de 0 ou 1 na linha
38       elif mat[i][j] == 1:
39          linha = linha + 1
40 # Passo 2.2.1.2. Verifique os elementos da coluna j
41       if mat[j][i] != 0 and mat[j][i] != 1:
42          coluna = 2 # tem um número diferente de 0 ou 1 na coluna
43       elif mat[j][i] == 1:
44          coluna = coluna + 1
45 # Passo 2.2.2. Verifique as condições de matriz de permutação
46    if linha == 0 or linha > 1 or coluna == 0 or coluna > 1:
47       permutacao = False
48       msg = 'N'
49    i += 1 # testar a próxima linha
50 # Passo 3. Imprime o resultado
51 print(msg)
52
53 # pós: 'P' and para cada i em {0..m-1} e j em {0..n-1}:
54 #      soma(mat[i][j])==1 and soma(mat[j][i])==1 or 'N'
55 # fim do módulo principal
```

Como observamos acima, a entrada é dada por um número `n` representado o tamanho (formato) da matriz ($n \times n$ – linhas e colunas), seguido por linhas, cada uma com `n` inteiros representando as `n` linhas da matriz (de `n` colunas). A saída consiste em imprimir `'P'` caso a matriz seja de permutação e `'N'` caso contrário. Você pode criar um arquivo para testar o seu programa. No nosso caso, vamos denominar o arquivo com `permutacao.in`. Obs: Os comentários não fazem parte da entrada, são apenas para facilitar a compreensão do

exercício. O programa pode ser executado com:

```
$ python permutacao.py < permutacao.in
```

Exemplo

```
# formato da entrada
4 4 # 4 linhas e 4 colunas
0 1 0 0 # linha 0
0 0 1 0 # linha 1
1 0 0 0 # linha 2
0 0 0 1 # linha 3

# formato da saída
P
```

1.3 Somas Constantes

Dizemos que uma matriz quadrada de números inteiros possui *somas constantes* se a soma dos elementos de cada linha, a soma dos elementos de cada coluna e a soma dos elementos das diagonais principal e secundária são todas iguais. Se os números da matriz variarem de 1 a n^2, dizemos que a matriz é um quadrado mágico.

Exemplo de uma matriz com soma constantes:

$$M = \begin{pmatrix} 8 & 0 & 7 \\ 4 & 5 & 6 \\ 3 & 10 & 2 \end{pmatrix}$$

Projete e implemente um programa que leia uma matriz M de dimensão $n \times n$, $0 < n \leq 50$, e compute se a matriz possui soma constantes. Se A possuir somas constantes, imprima `'CONSTANTE'`, caso contrário imprima `'NÃO É CONSTANTE'`.

A entrada é dada por um conjunto de linhas, sendo que a primeira linha contém um número inteiro `n` que representa a dimensão n da matriz, seguido de n linhas da matriz. A saída consiste em imprimir `'CONSTANTE'` se a matriz possui somas constantes e `'NÃO É CONSTANTE'` caso contrário. Obs: Os comentários não fazem parte da entrada, servem apenas para informar o significado dos dados da entrada.

Exemplo 1

```
# formato da entrada
3 # dimensão da matriz
8 0 7 # linha 1 da matriz
4 5 6 # linha 2 da matriz
3 10 2 # linha 3 da matriz

# formato da saída
CONSTANTE
```

Exemplo 2

```
# formato da entrada
2 # dimensão da matriz
3 5 # linha 1 da matriz
1 1 # linha 2 da matriz

# formato da saída
NÃO É CONSTANTE
```

$$A = \begin{pmatrix} a[0][0] & a[0][1] & a[0][2] & a[0][3] & a[0][4] \\ a[1][0] & a[1][1] & a[1][2] & a[1][3] & a[1][4] \\ a[2][0] & a[2][1] & a[2][2] & a[2][3] & a[2][4] \\ a[3][0] & a[3][1] & a[3][2] & a[3][3] & a[3][4] \\ a[4][0] & a[4][1] & a[4][2] & a[4][3] & a[4][4] \end{pmatrix}$$

para $n = 5$, temos que a diagonal principal, os elementos são da forma $a[i][i]$ e na diagonal secundária $a[4][0]$, $a[3][1]$, $a[2][2]$, $a[1][3]$ e $a[0][4]$. Neste caso, temos que para $a[i][j]$, i varia de $n-1$ a 0 (`i = n-1; i >= 0; i--`) e j de 0 a $n-1$ (`j = 0; j < n; j++`).

Podemos projetar os laços para percorrer as diagonais da seguinte forma:

```
# Inicialize a soma das diagonais
diag1 = 0
diag2 = 0
# inicialize a linha
i = 0
# percorra as linhas
while i < n:
# Some matriz[i][i] e matriz[i][n-1-i] a soma das diagonais
   diag1 = diag1 + matriz[i][i]
   diag2 = diag2 + matriz[i][n-1-i]
```

O programa abaixo ilustra uma implementação em Python.

```
1  # -*- coding: utf-8 -*-
2  # Programa: somasconst.py
3  # Programador:
4  # Data: 08/07/2019
5  # Este programa lê uma matriz quadrada e verifica se ela possui somas
6  # constantes. Ou seja, se a soma das linhas, colunas e diagonais
7  # principal e secundária tem o mesmo valor.
8  # início do módulo principal
9  # descrição das variáveis utilizadas
10 # list  matriz[[]] - lista das listas da produção
11 # int n - dimensão da matriz
12 # int soma - soma das linhas e colunas
13 # bool somaconst - variável booleana
14
15 # pré: n matriz[0][0]..matriz[lin][col]
16
17 # Passo 1. Crie a matriz e leia os elementos
```

```
# Passo 1.1. Leia as dimensões das matrizes
n = int(input())
# Passo 1.2. Inicialize a matriz
matriz = [[0]*n for i in range(n)]
# Passo 1.3. Leia a matriz linha a linha
for i in range(0,n):
   matriz[i] = list(map(int, input().split()))
# Passo 2. Verifique se a matriz possui somas constantes
# Passo 2.1. Inicialize as variáveis
somaconst = True
diagonal1 = 0
diagonal2 = 0
# Passo 2.2. Compute a soma das linhas e colunas de matriz
i = 0
while i < n and somaconst:
# Passo 2.2.1. Inicialize a sama das linhas e colunas
   linha = 0
   coluna = 0
# Passo 2.2.2. Compute a soma da linha i e coluna j
   for j in range(0,n):
      linha =  linha + matriz[i][j]
      coluna = coluna + matriz[j][i]
   diagonal1 = diagonal1 + matriz[i][i]
   diagonal2 = diagonal2 + matriz[i][n-1-i]
# Passo 2.2.3. Regitre a primeira soma
   if i == 0:
      soma = linha
# Passo 2.2.4. Verifique se a soma das linhas é igual a das colunas
   if soma != linha or soma != coluna:
      somaconst = False
   i = i + 1
# Passo 2.3. Verifique os elementos das diagonais
if somaconst and soma == diagonal1 and soma == diagonal2:
   msg = 'CONSTANTE'
else:
   msg = 'NÃO É CONSTANTE'
# Passo 3. Imprima o resultado
print(msg)

# pós: 'CONSTANTE' and matriz tem somas constantes or `'NÃO É CONSTANTE'
# fim do módulo principal
```

Como observamos acima, a entrada é dada por um bloco de linhas. A primeira linha do bloco contém um número inteiro representando a dimensão n da matriz, seguido das n linhas da matriz. A saída consiste em imprimir `'CONSTANTE'` caso a matriz tenha as somas constantes e `'NÃO É CONSTANTE'` caso contrário. Você pode criar um arquivo para testar o seu programa. No nosso caso, vamos denominar o arquivo com `somaconst.in`. Obs: Os comentários não fazem parte da entrada, são apenas para facilitar a compreensão do exercício. O programa pode ser executado com:

```
$ python somaconst.py < somaconst.in
```

Exemplo 1

```
# formato da entrada
# somaconstantes.in
3 # dimensão da matriz
8 0 7 # linha 0
4 5 6 # linha 1
3 10 2 # linha2

# formato da saída
CONSTANTE
```

Exemplo 2

```
# formato da entrada
2 # dimensão da matriz
3 5 # linha 0
1 1 # linha 1

# formato da saída
NÃO É CONSTANTE
```