Algoritmos e Programação I: Aula 35*

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

Sumário

1 Usando Listas para implementar Problemas com Matrizes **1**
1.1 Umidade Relativa do Ar 1
1.2 Industria Alimentícia - 1 5
1.3 Matriz Transposta 8
1.4 Mapa de Assentos 12

1 Usando Listas para implementar Problemas com Matrizes

Como vimos na aula passada, matrizes surgem naturalmente na modelagem de uma grande variedade de aplicações. Uma matriz $m \times n$ é formada por m linhas e n colunas. Usando listas para representar as matrizes, veremos agora outros exemplos onde a solução de um dado problema necessita de uma matriz como estrutura de dados para a representação de um dado conjunto de entrada e a solução do algoritmo utiliza operações matriciais.

1.1 Umidade Relativa do Ar

Considere o problema da umidade relativa do ar que vimos anteriormente. As medidas da umidade relativa do ar são feitas três vezes ao dia e em quatro lugares distintos. A média dessas quatro leituras é usada como a medida de umidade relativa da cidade para o período medido, valores dessa medida de umidade relativa do ar abaixo de 12% indicam **estado de emergência**, valores maiores ou iguais a 12% e menores ou iguais a 20% indicam **estado de alerta**, enquanto valores maiores que 20% e menores ou iguais a 30% indicam **estado de atenção**. Para valores superiores a 30% e inferiores a 60% indicam **estado aceitável** e superiores ou iguais a 60% indicam **estado ideal**.

As medidas são registradas numa tabela, onde cada coluna contém os dados da umidade relativa do ar nos 4 locais numa dada hora. Para calcular a média da umidade relativa do ar num dado horário, temos que somar a respectiva coluna e dividir por 4.

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

Essa tabela podem ser representadas com uma matriz, onde as linhas representam os locais onde as medições foram realizadas e as colunas os horários em que as medições foram feitas.

	04:00	12:00	20:00
Reserva	.62	.68	.72
Centro	.45	.28	.48
Industria	.42	.22	.18
Bairro	.32	.61	.45

Se usarmos uma lista, composta de 4 sublistas com 3 elementos cada para armazenar essas informações temos:

```
umidade[0] = [umidade[0][0], umidade[0][1], umidade[0][2]]
umidade[1] = [umidade[1][0], umidade[1][1], umidade[1][2]]
umidade[0] = [umidade[2][0], umidade[2][1], umidade[2][2]]
umidade[0] = [umidade[3][0], umidade[3][1], umidade[3][2]]
```

ou seja a matriz pode ser representada por uma lista com 4 elementos, onde cada elemento também é uma lista de 3 medidas (para cada local).

```
umidade = [[.62, .68, .72], [.45, .28, .48], [ .42, .22, .18],
           [.32, .61, .45]]
```

A lista acima indica que `umidade` tem 4 linhas e 3 colunas. No total, a lista tem elementos, onde o elemento `umidade[1][2] = .48` indica o valor da medida da umidade relativa do ar no horário das `20:00` horas e no local `Centro`.

Para computar as médias da umidade relativa do ar em cada um dos horários, basta somarmos as medidas nos quatro locais e calcular a média. No caso da tabela (matriz), temos que somar cada uma das colunas de `umidade` (locais em que a umidade foi registrada em cada um dos horários) e dividir pelo número de locais (4). Por exemplo, a umidade relativa do ar às 12:00 horas fica:

```
soma = .68 + .28 + .22 + .61
media = soma/4.0
```

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

```
# descrição das variáveis utilizadas
# list  umidade - lista das listas das temperaturas
# list  media - lista com as medias
# float soma
```

A entrada é dada pelo arquivo `umidade1.in`. O arquivo é composto por quatro linhas, cada uma com três números de ponto flutuante (`float`), representando a tabela com as umidades relativas do ar para o 4 locais nos três horários. Considerando que a entrada é dada por:

```
# formato da entrada
.62 .68 .72
.45 .28 .48
.42 .22 .18
.32 .61 .45
```

podemos ler essa lista da seguinte forma:

```
# Passo 1. Leia a entrada
# Passo 1.1. Inicialize as estruturas de dados
umidade = [[0.0]*3 for i in range(4)] # tabela 4 x 3
media = [0.0]*3 # média de 3 colunas
# Passo 1.2. Leia a tabela de temperaturas (linha a linha)
for i in range(0,4):
   umidade[i] = list(map(float, input().split())) # leia a linha i
```

Para calcular a média, devemos somar cada coluna da tabela e dividir pelo número de locais (4). Como a tabela está armazenada numa lista de listas, temos que somar os elementos e calcular a média:

```
soma = umidade[0][0]+umidade[1][0]+umidade[2][0]+umidade[3][0]
media[0] = soma/4.0 # média da umidade no horário 0 (4:00)
soma = umidade[0][1]+umidade[1][1]+umidade[2][1]+umidade[3][1]
media[1] = soma/4.0 # média da umidade no horário 1 (12:00)
soma = umidade[0][2]+umidade[1][2]+umidade[2][2]+umidade[3][2]
media[2] = soma/4.0 # média da umidade no horário 2 (20:00)
```

Para somar os elementos (i, j) da coluna j, fixamos cada um dos índices j e variamos o índice i. Isso pode ser feito com dois laços `for` aninhados, sendo que o laço externo controla o índice j e o laço interno controla o índice i. Dessa forma, para cada coluna de índice j em $\{0, 1, 2\}$, os valores de todas as linhas de índice i em $\{0, 1, 2, 3\}$ (`umidade[i][j]`) são acessados.

```
# Passo 2. Calcule as médias para cada horário
for j in range(0,3): # fixa coluna
   soma = 0.0
   for i in range(0,4): # varia a linha
      soma += umidade[i][j] # soma dos elementos da coluna
   media[j] = soma/4.0 # media da coluna j
```

Queremos que a impressão da lista resultante seja feita com apenas duas casas decimais. Nesse caso, ou usamos a função `print` com o `format` para imprimir cada um dos elementos, ou fazemos o ajuste diretamente na lista e depois imprimimos a lista. Isso pode ser feito da seguinte maneira:

```
# Passo 3. Imprima a lista da umidade média em cada horário
# Passo 3.1. Ajuste os elementos da lista com duas casas decimais
saida = [round(item,2) for item in media]
# Passo 3.2. Imprima a saída
print(saida)
```

```
# -*- coding: utf-8 -*-
# Programa: umidade01.py
# Programador:
# Data: 21/11/2016
# Este programa lê 3 medidas diárias de umidade do ar em 4 locais
# distintos, computa a média da umidade para cada um dos
# horários de medição e imprime o resultado
# início do módulo principal
# descrição das variáveis utilizadas
# list  umidade - lista das listas das umidades
# list  media - lista com as medias
# float soma

# pré: umidade[0] == umidade[0][0] umidade[0][1] umidade[0][2]
#      umidade[1] == umidade[1][0] umidade[1][1] umidade[1][2]
#      umidade[2] == umidade[2][0] umidade[2][1] umidade[2][2]
#      umidade[3] == umidade[3][0] umidade[3][1] umidade[3][2]

# Passo 1. Leia a entrada e crie as estruturas de dados
# Passo 1.1. Inicialize as estruturas de dados
umidade = [[0.0]*3 for i in range(4)] # tabela 4 x 3
media = [0.0]*3 # média de 3 colunas
# Passo 1.2. Leia a tabela das umidades
for i in range(0,4):
   umidade[i] = list(map(float,input().split())) # leia a linha i
# Passo 2. Calcule as médias para cada horário
for j in range(0,3): # fixa coluna
   soma = 0.0
   for i in range(0,4): # varia a linha
      soma = soma + umidade[i][j] # soma dos elementos da coluna
   media[j] = soma/4.0 # media da coluna j
# Passo 3. Imprima a lista da umidade média em cada horário
# Passo 3.1. Ajuste os elementos da lista com duas casas decimais
saida = [round(item,2) for item in media]
# Passo 3.2. Imprima a saída
print(saida)

# pós: para j em {0,...,2}:media[j] and media[j] == (soma)
#      i em {0,...,3}:tabtemp[i][j])/4.0
# fim do módulo principal
```

Você pode criar dois arquivos para testar o seu programa, uma para a matriz da entrada e outro para a lista das médias da saída. No nosso caso, vamos denominar o arquivo de entrada como `umidade01.in` e o arquivo de saída como `umidade01.out`. O programa pode ser executado com:

```
$ python umidade01.py < umidade01.in > umidade01.out
```

Exemplo

```
# formato da entrada
.62 .68 .72
.45 .28 .48
.42 .22 .18
.32 .61 .45

# formato da saída
[0.45, 0.45, 0.46]
```

Veremos agora outros exemplos onde a solução de um dado problema é modelado com o uso de uma matriz, representada como uma listas de listas, cujas soluções usam operações matriciais.

1.2 Industria Alimentícia - 1

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 fábricas e sua produção anual é registrada a cada três meses. As tabelas (matrizes) `TL` e `CX` indicam as produções das unidades fabris de Três Lagoas e Coxim, respectivamente.

$$TL = \begin{array}{c} \\ \text{Talharim} \\ \text{Nhoque} \\ \text{Fritas} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Jan-Mar} & \text{Abr-Jun} & \text{Jul-Set} & \text{Out-Dez} \end{array} \\ \begin{pmatrix} 150 & 250 & 180 & 220 \\ 50 & 100 & 80 & 120 \\ 150 & 100 & 100 & 200 \end{pmatrix} \end{array}$$

$$CX = \begin{array}{c} \\ \text{Talharim} \\ \text{Nhoque} \\ \text{Fritas} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Jan-Mar} & \text{Abr-Jun} & \text{Jul-Set} & \text{Out-Dez} \end{array} \\ \begin{pmatrix} 50 & 120 & 80 & 140 \\ 150 & 200 & 180 & 220 \\ 250 & 200 & 200 & 300 \end{pmatrix} \end{array}$$

A empresa deseja saber qual é a produção total de cada um dos produtos. Como a produção é representada por duas tabelas (matrizes), necessitamos somar as duas matrizes para obter a produção total de cada um dos produtos.

Temos que TL e CX são duas matrizes 3×4, de inteiros, sua soma é definida como segue: Se $tl_{i,j}$ e $cx_{i,j}$ são os elementos da entrada da i-ésima linha e j-ésima coluna de TL e CX, respectivamente, então $tl_{i,j} + cx_{i,j}$ é a entrada da i-ésima linha e j-ésima coluna da sua soma, a qual também é uma matriz 3×4, MC.

$$mc_{i,j} = tl_{i,j} + cx_{i,j}$$

$$\begin{pmatrix} tl_{0,0} & tl_{0,1} & tl_{0,2} & tl_{0,3} \\ tl_{1,0} & tl_{1,1} & tl_{1,2} & tl_{1,3} \\ tl_{2,0} & tl_{2,1} & tl_{2,2} & tl_{2,3} \end{pmatrix} + \begin{pmatrix} cx_{0,0} & cx_{0,1} & cx_{0,2} & cx_{0,3} \\ cx_{1,0} & cx_{1,1} & cx_{1,2} & cx_{1,3} \\ cx_{2,0} & cx_{2,1} & cx_{2,2} & cx_{2,3} \end{pmatrix} = \begin{pmatrix} mc_{0,0} & mc_{0,1} & mc_{0,2} & mc_{0,3} \\ mc_{1,0} & mc_{1,1} & mc_{1,2} & mc_{1,3} \\ mc_{2,0} & mc_{2,1} & mc_{2,2} & mc_{2,3} \end{pmatrix}$$

Como nos exemplos anteriores, vamos usar listas para representar as matrizes. As listas `tl`, `cx` e `mc` representam a produção de Três Lagoas, Coxim e o total da empresa respectivamente. Onde `tl[0]` representa a produção de talharim em Três Lagos e `cx[1][2]` representa a produção de Nhoque no período de julho a setembro. A lista `mc` armazena a soma das listas `tl` e `cs` e nesse caso, temos que:

$$mc[i][j] = tl[i][j] + cx[i][j]$$

$$\begin{pmatrix} tl[0][0] & tl[0][1] & tl[0][2] & tl[0][3] \\ tl[1][0] & tl[1][1] & tl[1][2] & tl[1][3] \\ tl[2][0] & tl[2][1] & tl[2][2] & tl[2][3] \end{pmatrix} + \begin{pmatrix} cx[0][0] & cx[0][1] & cx[0][2] & cx[0][3] \\ cx[1][0] & cx[1][1] & cx[1][2] & cx[1][3] \\ cx[2][0] & cx[2][1] & cx[2][2] & cx[2][3] \end{pmatrix} = \begin{pmatrix} mc[0][0] & mc[0][1] & mc[0][2] & mc[0][3] \\ mc[1][0] & mc[1][1] & mc[1][2] & mc[1][3] \\ mc[2][0] & mc[2][1] & mc[2][2] & mc[2][3] \end{pmatrix}$$

```
[[tl[0][0], tl[0][1], tl[0][2], tl[0][3]], [tl[1][0], tl[1][1], tl[1][2], tl[1][3]],
[tl[2][0], tl[2][1], tl[2][2], tl[2][3]]] + [[cx[0][0], cx[0][1], cx[0][2], cx[0][3]],
[cx[1][0], cx[1][1], cx[1][2], cx[1][3]], [cx[2][0], cx[2][1], cx[2][2], cx[2][3]]] =
[[mc[0][0], mc[0][1], mc[0][2], mc[0][3]], [mc[1][0], mc[1][1], mc[1][2], mc[1][3]],
[mc[2][0], mc[2][1], mc[2][2], mc[2][3]]]
```

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

```
# descrição das variáveis utilizadas
# list  tl, cx, mc - lista das listas da produção
# int   lin, col - dimensão das listas
```

A entrada é dada pelo arquivo `producao.in`. O arquivo é composto por um bloco de linhas. A primeira linha do bloco é dada por dois números inteiros contendo o tamanho (formato) das duas listas (matrizes m – linhas e n – colunas) seguida de dois sub-blocos de m linhas cada um, com n inteiros cada, representando a produção de um dado produto em cada uma das unidades. Obs1: A linha 2 representa a produção de talharim na unidade de Três Lagoas, a linha 6 a produção de nhoque na unidade de coxim. Obs2: Os comentários não fazem parte da entrada, são apenas para facilitar a compreensão do exercício.

Considerando que a entrada é dada por:

Exemplo

```
# formato da entrada
3 4 # dimensões da lista
150 250 180 220 # produção de talharim em Três Lagoas
50 100 80 120 # produção de nhoque em Três Lagoas
150 100 100 200 # produção de fritas em Três Lagoas
50 120 80 140 # produção de talharim em Coxim
150 200 180 220 # produção de nhoque em Coxim
250 200 200 300 # produção de fritas em Coxim
```

podemos ler essas listas da seguinte forma:

```
# Passo 1.Inicialize as estruturas e leia a entrada
# Passo 1.1. Leia as dimensões das listas
lin,col = map(int,input().split())
# Passo 1.2. Inicialize as estruturas de dados
tl = [[0.0]*col for i in range(lin)] # produção de Três Lagoas
cx = [[0.0]*col for i in range(lin)] # produção de Coxim
mc = [[0.0]*col for i in range(lin)] # produção total
# Passo 1.3. Leia a produção de Três Lagoas (linha a linha)
for i in range(0,lin):
   tl[i] = list(map(int, input().split())) # leia a linha i
```

```
# Passo 1.4. Leia a produção de Coxim (linha a linha)
for i in range(0,lin):
   cx[i] = list(map(int, input().split())) # leia a linha i
```

A soma da produção pode ser calculada com:

```
# Passo 2. Calcule a produção total da industria - soma das listas
for i in range(lin): # fixa linha i
   for j in range(col): # percorre as colunas da linha i
      mc[i][j] = tl[i][j]+cx[i][j] # soma dos elementos da linha i
```

e a saída consiste em imprimir as linhas que indicam a produção total de cada um dos itens produzidos:

```
# Passo 3. Imprima a produção total linha a linha
for i in range(lin):
   print(mc[i])
```

Exemplo

```
# formato de saída
[200, 370, 260, 360]
[200, 300, 260, 340]
[400, 300, 300, 500]
```

O programa abaixo ilustra uma implementação em Python:

```
# -*- coding: utf-8 -*-
# Programa: producao.py
# Programador:
# Data: 21/11/2016
# Este programa lê os valores da produção dos itens de duas
# unidades fabris para quatro períodos. O programa calcula e
# imprime a produção total de cada um dos itens produzidos
# pelas fábricas.
# início do módulo principal
# descrição das variáveis utilizadas
# list  tl, cx, mc - lista das listas da produção
# int   lin, col - dimensão das listas

# pré: tl[0][0]..tl[2][3] cx[0][0]..cx[2][3]

# Passo 1.Inicialize as estruturas e leia a entrada
# Passo 1.1. Leia as dimensões das listas
lin,col = map(int,input().split())
# Passo 1.2. Inicialize as estruturas de dados
mc = [[0]*col for i in range(lin)] # produção total
# Passo 1.3. Leia a produção de Três Lagoas (linha a linha)
for i in range(0,lin):
```

```
   tl[i] = list(map(int, input().split())) # leia a linha i
# Passo 1.4. Leia a produção de Coxim (linha a linha)
for i in range(0,lin):
   cx[i] = list(map(int, input().split())) # leia a linha i
# Passo 2. Calcule a produção total da industria - soma das listas
for i in range(lin): # fixa linha i
   for j in range(col): # percorre as colunas da linha i
      mc[i][j] = tl[i][j]+cx[i][j] # soma dos elementos da linha i
# Passo 3. Imprima a produção total linha a linha
for i in range(lin):
   print(mc[i])

# pós: mc and mc[i][i] == tl[i][j]+cx[i][j]
# fim do módulo principal
```

Você pode criar dois arquivos para testar o seu programa, uma para a matriz da entrada e outro para a lista das médias da saída. No nosso caso, vamos denominar o arquivo de entrada como `producao.in` e o arquivo de saída como `producao.out`. O programa pode ser executado com:

```
$ python producao.py < producao.in > producao.out
```

Exemplo

```
# formato da entrada
3 4
150 250 180 220
50 100 80 120
150 100 100 200
50 120 80 140
150 200 180 220
250 200 200 300

# formato da saída
[200, 370, 260, 360]
[200, 300, 260, 340]
[400, 300, 300, 500]
```

1.3 Matriz Transposta

Vamos agora abordar o problema de computar a matriz transposta de uma dada matriz A. A **transposta**[1] A^T de uma matriz A com m linhas e n colunas pode ser obtida por espelhamento de seus elementos ao longo da diagonal principal. A transposta da matriz A^T é a própria matriz A. Em álgebra linear a transposta de uma matriz A é uma matriz A^T obtida por uma das três ações equivalentes:

- espelhe A sobre a sua diagonal principal para obter A^T

[1] matriz transposta

- escreva as linhas de A como colunas de A^T
- escreva as colunas de A como linhas de A^T

Exemplo do cálculo da matriz transposta:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \quad A^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

Vamos usar uma lista de listas a para representar a matriz A. Como nos problemas anteriores, cada linha da matriz A é representado por um elemento da lista a.

```
a = [[a[0][0], a[0][1]], [a[1][0], a[1][1]], [a[2][0], a[2][1]]]
```

```
a = [[1, 2], [3, 4], [5, 6]]
```

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

```
# descrição das variáveis utilizadas
# list  a[[]] - lista de listas para representar a
# list  at[[]] - lista de listas para representar a transposta de a
# int   m, n - dimensões da matriz a
```

A entrada é dada pelo arquivo `matriza.in`. O arquivo é composto por um bloco de linhas. A primeira linha tem dois números inteiros `m, n` indicando o número de linhas e colunas da matriz, seguido de `m` linhas, cada uma com `n` números inteiros, representado as colunas. Os comentário não fazem parte da entrada, são apenas para ilustrar o significado.

```
# formato da entrada
3 2 # dimensões da matriz
1 2 # linha 1
3 4 # linha 2
5 6 # linha 3
```

Para especificar a matriz `at`, temos que inverter as dimensões, pois a matriz transposta tem as dimensões invertidas da matriz original. Ou seja se `a` tem `m` linhas e `n` colunas, a matriz transposta de `a`, `at` será uma matriz com `n` linhas e `m` colunas. Nesse caso, as especificações de `a` e `at` ficam:

```
# Estruturas de dados para armazenar a[[]] e at[[]]
a = [[0]*n for i in range(m)] # m linhas, n colunas
at = [[0]*m for i in range(n)] # n linhas m colunas
```

Uma vez definida a estrutura para armazenar os dados, e a forma que eles serão disponibilizados, temos que lê-los. Isso pode ser feito com:

```
# Passo 1. Leia a entrada
# Passo 1.1. Leia as dimensões da matriz a
m, n = map(int, input().split())
# Passo 1.2. Inicialize as estruturas de dados
a = [[0]*n for i in range(m)] # m linhas, n colunas
at = [[0]*m for i in range(n)] # n linhas m colunas
# Passo 1.3. Leia a matriz a (linha a linha)
for i in range(0,m):
   a[i] = list(map(int, input().split())) # leia a linha i de a
```

Após a leitura da matriz `a` e inicializada a matriz `at`, temos que computar a matriz `at`. Para isso, temos que atribuir `at[i][j] = a[j][i]` para todos os elementos `0 <= i < n` e `0 <= j <m`.

```
at[0][0] = a[0][0]        at[0][1] = a[1][0]        at[0][2] = a[2][0]
at[1][0] = a[0][1]        at[1][1] = a[1][1]        at[1][2] = a[2][1]
```

Para computar a matriz transposta `at` podemos usar o seguinte conjunto de instruções:

```
# Passo 2. Compute a matriz transposta de a
# Passo 2.1. Compute as linhas de at
for i in range(0,n): # fixa linha de at e coluna de a
   for j in range(0,m): # percorra a coluna i de a
      at[i][j] = a[j][i]
```

O `Python` tem uma forma mais sintética de fazer laços aninhados de listas. O conjunto de instruções acima pode ser resumido em:

```
at = [[a[j][i] for j in range(m)] for i in range(n)]
```

Essa instrução é equivalente a anterior. Primeiro a variável `i` recebe o valor 0 e em seguida é computada a lista `at[0]`. A lista recebe os valores de `a[0][0] a[1][0] a[2][0] ... a[m][0]`. A seguir a variável `i` recebe o valor 1 e a lista `at[1] = a[0][1] a[1][1] a[2][1] ... a[m][1]` é computada. Esses passos são repetidos até que o valor de `i == n-1`.

Uma vez calculada a matriz transposta `at`, queremos agora imprimir `at`. Como `at` é representada por uma lista, para melhor visualizar a matriz, vamos imprimir uma linha de cada vez.

A saída deve atender a seguinte especificação:

```
[1, 3, 5]  # linha 1 de at
[2, 4, 6]  # linha 2 de at
```

```
# Passo 3. Imprima a matriz transposta
for i in range(n): # número de linhas de at
   print(at[i])
```

A solução do problema fica:

```
# -*- coding: utf-8 -*-
# Programa: transposta.py
# Programador:
# Data: 16/11/2019
# Este algoritmo lê uma matriz a mxn, calcula sua transposta at
# e imprime o resultado.
# início do módulo principal
# descrição das variáveis utilizadas
# list  a[[]] - lista de listas para representar a
# list  at[[]] - lista de listas para representar a transposta de a
# int   m, n - dimensões da matriz a

# pré: a[0][0]..a[m][n]

# Passo 1. Leia a entrada
# Passo 1.1. Leia as dimensões da matriz a
m, n = map(int, input().split())
# Passo 1.2. Inicialize as estruturas de dados
a = [[0]*n for i in range(m)] # m linhas, n colunas
at = [[0]*m for i in range(n)] # n linhas m colunas
# Passo 1.3. Leia a matriz a (linha a linha)
for i in range(0,m):
   a[i] = list(map(int, input().split())) # leia a linha i de a
# Passo 2. Compute a matriz transposta de a
# Passo 2.1. Compute as linhas de at
for i in range(0,n): # fixa linha de at e coluna de a
   for j in range(0,m): # percorra a coluna i de a
      at[i][j] = a[j][i]
# Passo 3. Imprima a matriz transposta
for i in range(n): # número de linhas de at
   print(at[i])

# pós: at[0][0]..at[n][m] and at[i][j] == a[j][i]
# fim do módulo principal
```

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

```
$ python transposta.py < transposta.in
```

```
# formato da entrada
5 7
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
```

```
# formato da saída
[1, 8, 15, 22, 29]
[2, 9, 16, 23, 30]
[3, 10, 17, 24, 31]
[4, 11, 18, 25, 32]
[5, 12, 19, 26, 33]
[6, 13, 20, 27, 34]
[7, 14, 21, 28, 35]
```

1.4 Mapa de Assentos

As Linhas Aéreas Tuiuiú necessita de um programa para reservar assentos em suas aeronaves. Suas aeronaves possuem 12 fileiras de assentos, sendo que cada fileira possui 4 lugares. Os lugares são definidos pela fileira e poltrona. Por exemplo, a fileira 4 possui 4 lugares, 4A, 4B, 4C e 4D. O programa deve ler um mapa com a situação existente e fazer marcações de assentos ou informar "assento ocupado" quando o assento já foi reservado anteriormente, "assento inexistente" quando o usuário digitar um assento que não exista, ou reservar o assento e informar "reserva efetuada" quando o assento estiver disponível.

O programa deve ler as informações relativas ao número de fileiras e cadeiras em cada fileira. Após essas informações, o programa deve ler a situação atual das poltronas ocupadas e disponíveis. Podemos usar uma tabela com 12 linhas e com 4 colunas em cada linha para representar o mapa dos assentos do avião. Para os assentos que estiverem ocupados (reservados), usamos o número 1 e para os que estiverem disponíveis, usamos o número 0. Uma vez lido o mapa de assentos, o programa pede uma entrada do usuário ("`S`" ou "`s`") para fazer uma nova reserva. Qualquer entrada diferente de "`S`" ou "`s`" o programa encerra as reservas e imprime o mapa de assentos atual. Se a resposta for "`S`" ou "`s`", o programa pede uma entrada do usuário relativa a poltrona desejada. O usuário deve digitar de `1A, 1B, 1C, 1D, 2A, ..., 11D, 12A, 12B, 12C, 12D`. Se a entrada do usuário não estiver nessa faixa, o programa imprime "`assento inexistente`" e aguarda uma nova entrada do usuário ("`S`" ou "`s`") para fazer uma nova reserva. Caso a entrada do usuário esteja na faixa `1A, 1B, 1C, 1D, 2A, ..., 11D, 12A, 12B, 12C, 12D`, o programa verifica se o assento está disponível, Se o assento estiver disponível, o programa faz a reserva e imprime "`reserva efetuada`". Caso contrário, o programa imprime "`assento ocupado`". Após fazer a reserva ou informar que "`assento ocupado`", o programa aguarda uma nova entrada do usuário ("`S`" ou "`s`") para fazer uma nova reserva. Caso a resposta não seja "`S`" ou "`s`" o programa encerra as reservas e imprime o mapa de assentos atual.

Como nos casos anteriores, podemos usar uma lista `assentos` para representar o mapa de assentos da aeronave. O assento `1A` corresponde a `assentos[0][0]`, o assento `1B` corresponde a `assentos[0][1]`, o assento `1C` corresponde a `assentos[0][2]`, o assento `1D` corresponde a `assentos[0][3]`, o assento `2A` corresponde a `assentos[1][0]`, o assento `2B` corresponde a `assentos[1][1]`, e assim sucessivamente até o assento `12D` que corresponde a `assentos[11][3]`.

Para associar a identificação do assento desejado com os índices da lista `assentos`, podemos usar uma string, `reserva`, para armazenar a identificação do assento desejado e usando *slicing*, obter o número da fileira `fila` e a letra do assento `pos`. Como a string `reserva` pode ter tamanho 2 ou 3, vamos computar o tamanho `tam` de `reserva`. A número da fila pode ser obtido com `reserva[0:tam-1]`. Como a numeração dos assentos começa na fileira 1 e o índice da lista começa com 0, fazemos:

```
fila = reserva[0:tam-1] - 1
```

No caso da posição, temos que transformar `A, B, C, D` em `0, 1, 2, 3`. Isso pode ser feito com:
`pos = ord(reserva[tam-1:tam].upper())-ord('A')`
neste caso, `reserva[tam-1:tam]` corresponde a letra na fileira, e a função `ord()` computa o código `ASCII` da letra correspondente e a subtração por `ord('A')`, garante que o valor ficará entra 0 e 3.

Com essas operações, temos uma associação entre as poltronas da aeronave e os índices da lista `assentos` utilizada para armazenar as informações das reservas dos assentos na aeronave. Usaremos `assentos[i][j] == 1` para informar que o assento `[i][j]` está ocupado, e `assentos[i][j] == 1` para informar que o assento `[i][j]` está livre.

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

```
# descrição das variáveis utilizadas
# list assentos - lista das listas do mapa de assentos
# string reserva - número do assento
# int lin, col - dimensão das listas
# int fila, pos - posição do assento no mapa
# string resposta - resposta do usuário
```

A entrada é dada pelo arquivo `assentos.in`. O arquivo é composto por um bloco de linhas. A primeira linha do bloco é dada por dois números inteiros contendo o tamanho (formato) da lista dos assentos (matrizes m – linhas e n – colunas) seguida de um sub-bloco de m linhas (fileiras), com n inteiros (0 ou 1)cada, representando o mapa de assentos de uma dada aeronave. Obs1: A linha 2 representa os assentos da primeira fileira do avião, a linha 3 da segunda fileira e assim sucessivamente, até a linha 13. Da linha 14 diante, são strings indicando se o usuário deseja fazer uma reserva e se for o caso, na linha seguinte, o assento desejado. Se a linha não contiver "S" ou "s", a leitura do programa é encerrada. Obs2: Os comentários não fazem parte da entrada, são apenas para facilitar a compreensão do exercício.

Considerando que a entrada é dada por:

Exemplo

```
# formato da entrada
12 4 # dimensões da aeronave
0 1 0 1 # fileira 1
1 1 1 1 # fileira 2
0 0 0 0 # fileira 3
1 0 0 0 # fileira 4
0 0 0 1 # fileira 5
1 0 1 0 # fileira 6
1 1 1 0 # fileira 7
0 1 1 1 # fileira 8
0 0 0 0 # fileira 9
1 1 1 1 # fileira 10
0 0 1 0 # fileira 11
0 1 0 0 # fileira 12
S
1A
S
7D
```

```
s
15A
S
8B
S
4E
12C
N
```

podemos ler essa lista (linhas 1 a 13 do arquivo) da seguinte forma:

```
# Passo 1.Inicialize as estruturas e leia a entrada
# Passo 1.1. Leia as dimensões das listas
lin,col = map(int,input().split())
# Passo 1.2. Inicialize as estruturas de dados
assentos  = [[0]*col for i in range(lin)] # mapa da aeronave
# Passo 1.3. Leia a a situação inicial (linha a linha)
for i in range(0,lin):
   assentos[i] = list(map(int, input().split())) # leia a linha i
```

Após ler a situação inicial, o programa deve aguardar a entrada do usuário para fazer novas reservas ou finalizar o programa. Isso pode ser feito da seguinte forma:

```
# Passo 2. Leia as reservas de assentos
# Passo 2.1. Leia se o usuário quer fazer uma reserva
resposta = input()
# Passo 2.2. Enquanto o usuário responder S ou s faça
while resposta.upper() == 'S':
```

Se a resposta do usuário for "S" ou "s", o programa deve ler a informação relativa ao assento desejado. A entrada será do um número (fileira), seguido de uma letra `A, B, C, D`. Como vimos anteriormente, temos que transformar essa entrada para um par de índices na lista `assentos`. Isso pode ser feito da seguinte forma:

```
# Passo 2.2.1. Leia o assento desejado
   reserva = input()
# Passo 2.2.2. Compute o tamanho da entrada
   tam = len(reserva)
# Passo 2.2.3. Obtenha o número da fileira desejada
   fila = int(reserva[0:tam-1])-1
# Passo 2.2.4. Obtenha o lugar desejado (último char da entrada)
   pos = ord(reserva[tam-1:tam].upper())-ord('A')
```

Uma vez convertido o número do assento para os índices `fila` e `pos` na lista `assentos`, temos que analisar a situação do assento desejado. Se `fila` e `pos` não estiverem dentro do intervalo dos índices de `assentos`, o programa deve imprimir "`assento inexistente`". Se `fila` e `pos` estiverem dentro do intervalo dos índices de `assentos`, o programa deve verificar se o assento está disponível. Se o assento estiver disponível `assentos[fila][pos] == 0`, o programa deve reservar o assento, `assento[fila][pos] = 1` e imprimir "`reserva`

`efetuado`". Caso o assento não estiver disponível, imprimir "`assento ocupado`". Após isso, o programa deve ler uma nova entrada para ver se o usuário quer fazer uma reserva. Esse conjunto de instruções continua sendo executado até que a entrada do usuário para `resposta` seja diferente de "`S`" ou "`s`". Nesse caso, o programa passa para o `Passo 3` e imprime a nova situação das reservas de assentos da aeronave.

```
 # Passo 2.2.5. Verifique se o assento desejada não existe
   if fila >= lin or pos >= col:
      print('assento inexistente')
# Passo 2.2.6. Verifique se o assento desejado está disponível
   elif assentos[fila][pos] == 0:
      assentos[fila][pos] = 1 # reserve o assento
      print('reserva efetuada')
# Passo 2.2.7. Verifique se o assento desejado está ocupado
   else:
      print('assento ocupado')
# Passo 2.2.8. Leia se o usuário quer fazer uma reserva
   resposta = input()
```

Para melhor visualizar o mapa de assentos, vamos imprimir um cabeçalho com as respectivas letras dos assentos, e imprimir uma linha de cada vez. No início de cada linha, imprimir também o número da linha. A saída deve atender a seguinte especificação:
Exemplo

```
# formato da saída
   ['A', 'B', 'C', 'D']
 1 ['1', '1', '0', '1']
 2 ['1', '1', '1', '1']
 3 ['0', '0', '0', '0']
 4 ['1', '0', '0', '0']
 5 ['0', '0', '0', '1']
 6 ['1', '0', '1', '0']
 7 ['1', '1', '1', '7']
 8 ['0', '1', '1', '1']
 9 ['0', '0', '0', '0']
10 ['1', '1', '1', '1']
11 ['0', '0', '1', '0']
12 ['0', '1', '1', '0']
```

Para imprimirmos o cabeçalho, primeiro temos que gerar uma "lista" `['A', 'B', 'C', 'D']`. Podemos obter isso com a instrução:
`mapa = [chr(65+i) for i in range(colunas)]`

Para imprimir as demais linhas, temos que imprimir o número da linha e depois a sublista `assentos[i]`. Usando as funcionalidades da função `print` temos:

```
# Passo 3. Imprima o mapa de ocupação da aeronave
mapa = [chr(65+i) for i in range(colunas)]
print('  ',mapa)
for i in range(linhas):
   print('0:2d'.format(i+1),list(map(str,assentos[i])))
```

A solução do problema fica:

```
# -*- coding: utf-8 -*-
# Programa: assentos.py
# Programador:
# Data: 29/02/2019
# Este algoritmo lê um conjunto de reservas de assentos em um avião
# e imprime o mapa de ocupação do avião.
# início do módulo principal
# descrição das variáveis utilizadas
# list   assentos - lista das listas do mapa de assentos
# string reserva - número do assento
# int    linhas, colunas - dimensão das listas
# int    fila, pos - posição do assento no mapa
# string resposta - resposta do usuário

# pré: assentos[0][0]..assentos[11][3]

# Passo 1.Inicialize as estruturas e leia a entrada
# Passo 1.1. Leia as dimensões das listas
lin,col = map(int,input().split())
# Passo 1.2. Inicialize as estruturas de dados
assentos  = [[0]*col for i in range(lin)] # mapa da aeronave
# Passo 1.3. Leia a a situação inicial (linha a linha)
for i in range(0,lin):
   assentos[i] = list(map(int, input().split())) # leia a linha i
# Passo 2. Leia as reservas de assentos
# Passo 2.1. Leia se o usuário quer fazer uma reserva
resposta = input()
# Passo 2.2. Enquanto o usuário responder S ou s faça
while resposta.upper() == 'S':
# Passo 2.2.1. Leia o assento desejado
   reserva = input()
# Passo 2.2.2. Compute o tamanho da entrada
   tam = len(reserva)
# Passo 2.2.3. Obtenha o número da fileira desejada
   fila = int(reserva[0:tam-1])-1
# Passo 2.2.4. Obtenha o lugar desejado (último char da entrada)
   pos = ord(reserva[tam-1:tam].upper())-ord('A')
 # Passo 2.2.5. Verifique se o assento desejada não existe
   if fila >= lin or pos >= col:
      print(reserva,'assento inexistente')
# Passo 2.2.6. Verifique se o assento desejado está disponível
   elif assentos[fila][pos] == 0:
      assentos[fila][pos] = 1 # reserve o assento
      print(reserva,'reserva efetuada')
# Passo 2.2.7. Verifique se o assento desejado está ocupado
   else:
      print(reserva,'assento ocupado')
# Passo 2.2.8. Leia se o usuário quer fazer uma reserva
   resposta = input()
```

```
50 # Passo 3. Imprima o mapa de ocupação da aeronave
51 mapa = [chr(65+i) for i in range(col)]
52 print('  ',mapa)
53 for i in range(lin):
54    print('0:2d'.format(i+1),list(map(str,assentos[i])))
55
56 # pós: assentos and assentos[i][j] == (0 or 1)
57 # fim do módulo principal
```

Você pode criar dois arquivos para testar o seu programa, uma para a matriz da entrada e outro para a lista das médias da saída. No nosso caso, vamos denominar o arquivo de entrada como `assentos.in` e o arquivo de saída como `assentos.out`. O programa pode ser executado com:

```
$ python assentos.py < assentos.in > assentos.out
```

Exemplo

```
# formato da entrada
12 4
0 1 0 1
1 1 1 1
0 0 0 0
1 0 0 0
0 0 0 1
1 0 1 0
1 1 1 0
0 1 1 1
0 0 0 0
1 1 1 1
0 0 1 0
0 1 0 0
S
1A
S
7D
s
15A
S
8B
S
4E
12C
N

# formato da entrada
1A reserva efetuada
7D reserva efetuada
15A assento inexistente
8B assento ocupado
```

```
4E assento inexistente
   ['A', 'B', 'C', 'D']
 1 ['1', '1', '0', '1']
 2 ['1', '1', '1', '1']
 3 ['0', '0', '0', '0']
 4 ['1', '0', '0', '0']
 5 ['0', '0', '0', '1']
 6 ['1', '0', '1', '0']
 7 ['1', '1', '1', '1']
 8 ['0', '1', '1', '1']
 9 ['0', '0', '0', '0']
10 ['1', '1', '1', '1']
11 ['0', '0', '1', '0']
12 ['0', '1', '0', '0']
```