MAC122 – Princípios de Desenvolvimento de Algoritmos
Segundo Semestre de 2025 – BMAC – IMEUSP – Prof. Marcilio
Exercício Programa - Entregar até 23/11/2025

Dado um arquivo de texto, onde cada linha contém os seguintes campos separados por vírgula:

<identidade>,<nome>,<data>

Exemplo:

```
32343245100,Jose de Castro,10/12/2001
43456790236,Maria das Dores e Silva,05/01/2005
22589349476,Antonio Paixao dos Santos,30/05/2012
32343245111,Jose de Castro,10/12/2010
43456790222,Maria da Silva,25/01/2015
22589348876,Antonio dos Santos,30/06/2002
32343245144,Jose de Campos,15/15/2007
43453330236,Maria da Rocha,27/11/2000
22555549476,Armando da Silva,20/06/2002
...
```

O programa deve ler o arquivo, colocá-lo numa lista **TAB**. Cada elemento de **TAB** é uma lista com 3 strings (exemplo abaixo). Em seguida o programa deve classificar esta lista **TAB** em ordem crescente usando o método Quick ou o método Merge e mostrar o tempo gasto em cada algoritmo.

O objetivo é classificar a lista TAB de todas as formas possíveis, a escolher, em ordem crescente ou decrescente. O nome em ordem alfabética, na data a antiga é menor que a recente e a identidade também em ordem alfabética.

Especial atenção para a comparação da data.

TAB:

```
[["32343245100","Jose de Castro","10/12/2001"]
 ["43456790236","Maria das Dores e Silva","05/01/2005"]
 ["22589349476","Antonio Paixao dos Santos","30/05/2012"]
 . . .
]
```

Algoritmos de Classificação

Escreva 2 funções:

1) **ClassificaQuick(TAB,comp)** – que classifica TAB pelo método Quick não recursivo
2) **ClassificaMerge(TAB,comp)** – que classifica TAB pelo método Merge não recursivo

comp é a função de comparação entre dois elementos. É uma função externa às funções de classificação que deve ser importada.

Exemplo:

Módulo `Compara.py` que contém as várias funções `comp`.

```
def comp1(a, b):
    # Devolve True se a vem antes de b na classificação
    # . . .

def comp2(a, b):
    # Devolve True se a vem antes de b na classificação
    # . . .

def comp3(a, b):
    # Devolve True se a vem antes de b na classificação
    # . . .

   . . .
```

Exemplo de função `comp`:

```
def data(x):
    # converte a data para comparação
    # . . .

def compx(a, b):

    # ordem crescente de nome
    #   se os nomes forem iguais em ordem decrescente de data
    #      se as datas forem iguais em ordem crescente de identidade

    # nome
    if a[1] == b[1]: # nomes iguais
        # data
        if data(a[2]) == data(b[2]): # datas iguais
            # identidade
            return a[0] <= b[0] # crescente de identidade
        else: return data(a[2]) >= data(b[2]) # decrescente de data
    else: return a[1] <= b[1] # crescente de nome
```

O seu programa:

```
from Compara import comp1, comp2, ...

def ClassificaQuick(TAB,comp):
. . .
def ClassificaMerge(TAB,comp):
. . .
```

Faça pelo menos as seguintes funções **`comp`**:
`comp1` - crescente por nome – crescente por data – crescente por id.
`comp2` - crescente por nome – decrescente por data – crescente por identidade
`comp3` - decrescente por data – crescente por nome – crescente por identidade.
`comp4` - crescente por identidade – crescente por data – crescente por nome.

Entregue seu programa com essas 4 versões:

O Programa

O programa terá então a seguinte estrutura:

1. Pedir o nome do arquivo de origem. Até que seja digitado **"fim"**
2. Ler o arquivo e colocar o seu conteúdo em **TAB**
3. Solicitar digitação do método (q ou m) e da ordem (1 a 4) de classificação
4. **ClassificaQuick(TAB, comp 1 2 3 ou 4)** ou
   ClassificaMerge(TAB, comp 1 2 3 ou 4)
5. Mostrar o tempo de classificação.
6. Listar os 100 primeiros itens de TAB classificada para conferência visual
7. Repetir a partir do passo 3

Exemplo de saída do programa:

```
Nome do arquivo de origem:arq1.txt

Quick ou Merge (q ou m)? q

Ordem – comp1a4? 1

Tempo do Quick: 0.002345

100 primeiros registros da tabela
Ordem: cresc. por nome – cresc. por data – cresc. por id.

Ind  Identidade     nome                          data
1    32343245100    Jose de Castro                10/12/2001
2    43456790236    Maria das Dores e Silva       05/01/2005
...

Quick ou Merge (q ou m)? m

Ordem – comp1a4? 3

Tempo do Merge: 0.001234

100 primeiros registros da tabela
Ordem: decr. por data – cresc. Por nome – cresc. por id.

Ind  Identidade     nome                          data
1    43456790222    Maria da Silva                25/01/2015
2    22589349476    Antonio Paixao dos Santos     30/05/2012
3    32343245111    Jose de Castro                10/12/2010
...
```

```
Quick ou Merge (q ou m)? f
*** fim
```

O programa gerador dos arquivos de testes

Use o programa abaixo para gerar arquivos de teste para o seu programa.
Gere por exemplo arquivos com 100, 1.000, 10.000 registros usando o programa abaixo.

```
from random import seed, randrange
# nomes randômicos
n1 = ["Felicia", "Catulo", "Osmund", "Artmio", "Senizio", "Tilenio"]
n2 = ["Cartuxo", "Olambro", "Romulo", "Ambulo", "Atomon", "Virino"]
n3 = ["Sereno", "Soterno", "Moncoes", "Oscaran", "Topovi", "Talento"]
n4 = ["Lasmia", "Mantega", "Casas", "Lorentao", "Melkioz", "Motivio"]
nn = 6

# Gera um registro com IDENT,NOME,DATAN
# Conteúdo randômico baseado em seu NUSP
# pp = '' - gera um registro completo
# pp != '' - gera apenas uma nova datan + ident ou apenas ident
def GeraRegistro(pp):
    global n1, n2, n3, n4, nn
    # nome, datan e ident
    nome = n1[randrange(nn)] + ' ' + n2[randrange(nn)] + ' ' +
n3[randrange(nn)] + ' ' + n4[randrange(nn)]
    dia = randrange(28) + 1
    mes = randrange(12) + 1
    ano = randrange(17) + 2000
    datan = f'{dia:02}' + '/' + f'{mes:02}' + '/' + f'{ano:04}'
    ident = f'{randrange(100000000000):011}'
    kr = randrange(3)
    if pp == '':
        # gera um novo registro completo
        registro = ident + ',' + nome + ',' + datan
        return registro
    elif kr == 0:
        # preserva o nome e gera datan + ident
        campos = pp.split(',')
        registro = ident + ',' + campos[1] + ',' + datan
        return registro
    elif kr == 1:
        # preserva o nome e datan e gera ident
        campos = pp.split(',')
        registro = ident + ',' + campos[1] + ',' + campos[2]
        return registro
    else:
        # preserva ident e nome e gera datan
        campos = pp.split(',')
        registro = campos[0] + ',' + campos[1] + ',' + datan
        return registro

# gera arquivo nomearq com nreg registros
def GeraArquivo(nusp, nomearq, nreg):
    # randomize
    seed(nusp)
    # quantidade de registros - gera 80% do total
    nreg80 = nreg * 80 // 100
    # tabela para guardar registros para repetição
```

```
    tab = ['' for k in range(nreg // 20)] # 5% dos registros
    # abre arquivo para gravação
    arq = open(nomearq, "w")
    # grava metade dos registros
    for k in range(nreg80):
        reg = GeraRegistro('')
        arq.write(reg + '\n')
        print(k + 1, " - ", reg)
        # guarda 5% dos registros para repetição
        if k % 20 == 0:
            # guarda em tab
            tab[k // 20] = reg
    # grava o resto dos 20% dos registros
    cont = nreg80 + 1
    for k in range(len(tab)):
        # para cada registro em tab gera 4 outros
        for j in range(4):
            reg = GeraRegistro(tab[k])
            arq.write(reg + '\n')
            print(cont, " - ", reg)
            cont += 1
    # fecha arquivo
    arq.close()

# Entre com seu NUSP - para randomizar
nusp = int(input("Entre com seu NUSP - para randomizar:"))
# Gera arquivo com uma certa quantidade de registros
while True:
    nome_arq = input("Entre com o nome do arquivo.txt:")
    quant_reg = int(input("Entre com a quantidade de registros:"))
    GeraArquivo(nusp, nome_arq, quant_reg)
```