PauloHenriqueSantosLima_desafio_engenharia

April 22, 2022

- **Nome:** Paulo Henrique Santos Lima
- **Universidade:** Universidade Federal de Goiás
- **Curso:** Ciência da computação
- **Semestre atual:** 10
- **Previsão de formação:** 2023/2

0.1 IDEIA BASE

1. Criar uma estrutura que ligue o **nome do pacote** com o **código**, como **vetores** por exemplo, neste desafio se usou **dicionário**;
2. Quebrar o código em **trincas**, comparar com **restrições** e realizar resolução do questionário.

As questões Todas as questões estão reunidas em um único arquivo chamado "PauloHenriqueSantosLima_desafio_engenharia.ipynb".

As questões do desafio foram desenvolvidas na linguagem Python, para cada uma tem os seguintes tópicos: * Ideia da lógica de como chegar ao resultado; * Breve descrição da estrutura do código; * Quais questões são necessárias para executar a questão a ser executada.

Requisitos

- Importe o arquivo "PauloHenriqueSantosLima_desafio_engenharia.ipynb" para um programa que execute arquivos do tipo ".ipynb", uma sugestão de uso é o Google Colab (colab.research.google.com/) que pode ser executado pelo navegador;
- Python 3.

0.2 0. Dicionários Pacotes

- Para cada pacote se associa seu respectivo código.

[97]:
```
regiao = {"Centro-Oeste":0,
          "Nordeste": 0,
          "Norte":0,
          "Sudeste": 0,
          "Sul":0,
          "Região Invalida": 0}
```

```
# nome do pacote com seu respectivo código
pacotes = {"Pacote 1": "288355555123888",
           "Pacote 2": "335333555584333",
           "Pacote 3": "223343555124001",
           "Pacote 4": "002111555874555",
           "Pacote 5": "111188555654777",
           "Pacote 6": "111333555123333",
           "Pacote 7": "432055555123888",
           "Pacote 8": "079333555584333",
           "Pacote 9": "155333555124001",
           "Pacote 10": "333188555584333",
           "Pacote 11": "555288555123001",
           "Pacote 12": "111388555123555",
           "Pacote 13": "288000555367333",
           "Pacote 14": "066311555874001",
           "Pacote 15": "110333555123555",
           "Pacote 16": "333488555584333",
           "Pacote 17": "455448555123001",
           "Pacote 18": "022388555123555",
           "Pacote 19": "432044555845333",
           "Pacote 20": "034311555874001"}
```

0.3 1. Identificar a região de destino de cada pacote, com totalização de pacotes (soma região);

ideia: Quebrar o código de cada pacote em uma trinca, tal que essa seja a trinca de destino [3:6]. Em seguida, compara se essa trinca está em algum intervalo de alguma região de destino, se sim imprime dados da região e incrementa o contador daquela região que será sua soma, se não imprime erro e incrementa o contador do erro que representa a soma das regiões com destino inválido.

Estrutura programa

1. Para cada pacote se faz a comparação da segunda trinca, destino, com o intervalo de destino do pacote, se for um destino válido imprime dados e incrementa o contador de soma da região, se for um destino inválido se incrementa o contador de região inválida e imprime destino inválido;
2. Imprime a região com sua respectiva soma;
3. Zera soma das regiões para que não some o mesmo valor na região caso execute o programa.

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes.

[98]:
```
# 1. compara e incrementa
print("REGIÃO DE DESTINO POR PACOTE")
for chave in pacotes.keys():
  i = 3
  j = 6
```

```
  if(int(pacotes[chave][i:j])>=201 and int(pacotes[chave][i:j])<=299):
    print("Região de destino {}: Cidade {}, região Centro-oeste" .
↪format(str(chave), pacotes[chave][i:j]))
    regiao["Centro-Oeste"] = regiao["Centro-Oeste"]+1
  elif(int(pacotes[chave][i:j])>=300 and int(pacotes[chave][i:j])<=399):
    print("Região de destino {}: Cidade {}, região Nordeste" .
↪format(str(chave), pacotes[chave][i:j]))
    regiao["Nordeste"] = regiao["Nordeste"]+1
  elif(int(pacotes[chave][i:j])>=400 and int(pacotes[chave][i:j])<=499):
    print("Região de destino {}: Cidade {}, região Norte" .format(str(chave),␣
↪pacotes[chave][i:j]))
    regiao["Norte"] = regiao["Norte"]+1
  elif(int(pacotes[chave][i:j])>=1 and int(pacotes[chave][i:j])<=99):
    print("Região de destino {}: Cidade {}, região Sudeste" .format(str(chave),␣
↪pacotes[chave][i:j]))
    regiao["Sudeste"] = regiao["Sudeste"]+1
  elif(int(pacotes[chave][i:j])>=100 and int(pacotes[chave][i:j])<=199):
    print("Região de destino {}: Cidade {}, região Sul" .format(str(chave),␣
↪pacotes[chave][i:j]))
    regiao["Sul"] = regiao["Sul"]+1
  else:
    print("Região de destino {}: Cidade {} - região Inválida" .
↪format(str(chave), pacotes[chave][i:j]))
    regiao["Região Invalida"] = regiao["Região Invalida"]+1

# 2. imprime soma regiões
print("\nSOMA DE PACOTES POR REGIÃO")
for chave, valor in regiao.items():
    print("{} = {}" .format(chave, valor))

# 3. zera soma das regiões
for key, value in regiao.items():
  regiao[key] = 0
```

```
REGIÃO DE DESTINO POR PACOTE
Região de destino Pacote 1: Cidade 355, região Nordeste
Região de destino Pacote 2: Cidade 333, região Nordeste
Região de destino Pacote 3: Cidade 343, região Nordeste
Região de destino Pacote 4: Cidade 111, região Sul
Região de destino Pacote 5: Cidade 188, região Sul
Região de destino Pacote 6: Cidade 333, região Nordeste
Região de destino Pacote 7: Cidade 055, região Sudeste
Região de destino Pacote 8: Cidade 333, região Nordeste
Região de destino Pacote 9: Cidade 333, região Nordeste
Região de destino Pacote 10: Cidade 188, região Sul
Região de destino Pacote 11: Cidade 288, região Centro-oeste
Região de destino Pacote 12: Cidade 388, região Nordeste
```

```
Região de destino Pacote 13: Cidade 000 - região Inválida
Região de destino Pacote 14: Cidade 311, região Nordeste
Região de destino Pacote 15: Cidade 333, região Nordeste
Região de destino Pacote 16: Cidade 488, região Norte
Região de destino Pacote 17: Cidade 448, região Norte
Região de destino Pacote 18: Cidade 388, região Nordeste
Região de destino Pacote 19: Cidade 044, região Sudeste
Região de destino Pacote 20: Cidade 311, região Nordeste

SOMA DE PACOTES POR REGIÃO
Centro-Oeste = 1
Nordeste = 11
Norte = 2
Sudeste = 2
Sul = 3
Região Invalida = 1
```

0.4 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos;

Critérios

1. A Loggi não envia produtos que não sejam dos tipos acima informados;
2. Não é possível despachar pacotes contendo jóias tendo como região de origem o Centro-oeste;
3. O vendedor 367 está com seu CNPJ inativo e, portanto, não pode mais enviar pacotes pela Loggi, os códigos de barra que estiverem relacionados a este vendedor devem ser considerados inválidos;
4. O código do produto deve ser válido.

ideia: Armazenar em 2 listas se um pacote possui código válido ou não. Para isso testa uma condição envolvendo todas as restrições, se algum pacote testar verdadeiro para alguma restrição, este pacote é armazenado na lista de código invalido, mas se algum pacote testar falso para todas as restrições, então ele vai para a lista de pacotes válidos.

Estrutura programa

1. Cria 2 listas e usando um laço testa com uma condição para todas as restrições, se alguma restrição for verdadeira, armazena na lista de invalido o pacote que sinalizou, caso contrário armazena na lista de válidos;
2. imprime os pacotes com códigos válidos e em seguida os inválidos.

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes.

[99]:
```
# 1. cria 2 listas
valido = []
invalido = []

# 1. compara restrições
```

```
for chave in pacotes.keys():
  if( (len(pacotes[chave]) != 15)
      or  ( (int(pacotes[chave][12:15]) == 1) and
          (int(pacotes[chave][0:3])>=201 and int(pacotes[chave][0:3])<=299) )
      or  (int(pacotes[chave][9:12]) == 367)
      or  (int(pacotes[chave][12:15]) != 1 and int(pacotes[chave][12:15]) !=␣
 ↪111 and
           int(pacotes[chave][12:15]) != 333 and int(pacotes[chave][12:15]) !=␣
 ↪555 and
           int(pacotes[chave][12:15]) != 888) ):
    invalido.append(chave)
  else:
    valido.append(chave)

# 2. imprime pacotes
print("CÓDIGOS VÁLIDOS")
for val in valido:
    print(val)
print("\nCÓDIGOS INVÁLIDOS")
for inval in invalido:
    print(inval)
```

```
CÓDIGOS VÁLIDOS
Pacote 1
Pacote 2
Pacote 4
Pacote 6
Pacote 7
Pacote 8
Pacote 9
Pacote 10
Pacote 11
Pacote 12
Pacote 14
Pacote 15
Pacote 16
Pacote 17
Pacote 18
Pacote 19
Pacote 20

CÓDIGOS INVÁLIDOS
Pacote 3
Pacote 5
Pacote 13
```

0.5 3. Identificar os pacotes que têm como origem a região Sul e Brinquedos em seu conteúdo;

ideia: Comparar se a primeira trinca, de origem [0:3], está no intervalo da região sul e verificar se trinca de produto [12:15] é igual ao código de brinquedos.

Estrutura programa

1. Com um laço, se testa se produto é válido e se a trinca [0:3] está no intervalo da região sul e se a trinca final [12:15] é igual brinquedo, se sim imprime dados do pacote e incrementa a flag;
2. Caso nenhum produto passe na condicional, então a flag é ativada e imprime mensagem de zero pacotes.

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes; * 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos.

[100]:
```
# 1. verifica se pacote é valido, se é do sul e brinquedo
print("PACOTES COM BRINQUEDOS E COM ORIGEM NO SUL")
flag = 0
for chave in pacotes.keys():
  if(chave in valido):
    if((int(pacotes[chave][0:3])>=100 and int(pacotes[chave][0:3])<=199) and␣
 ↪int(pacotes[chave][12:15]) == 888):
      print("{}: com origem na Cidade {}" .format(str(chave), pacotes[chave][0:
 ↪3]))
      flag +=1

# 2. ativa flag se não houver pacotes
if(flag==0):
  print("Zero Pacotes")
```

```
PACOTES COM BRINQUEDOS E COM ORIGEM NO SUL
Zero Pacotes
```

0.6 4. Listar os pacotes agrupados por região de destino (Considere apenas pacotes válidos);

ideia: Criar uma estrutura auxiliar que receba os dados da estrutura pacotes, para que essa não seja alterada. Em seguida se quebra o código dos pacotes dessa estrutura auxiliar em 2 partes (parte 1 sendo a trinca origem [0:3] e a parte 2 sendo o resto do código do pacote [3:15]), jogando a parte 1 no final da parte 2, assim se organiza os códigos deixando o destino na trinca inicial; em seguida se ordena os pacotes de acordo com esse novo código, pois assim se agrupa por região de destino; em seguida se une as 2 partes ao seu estado original jogando a parte final (neste caso a trinca de origem) [12:15:] no começo da parte 1 [0:12] (restante do código). Para trabalhar apenas com pacotes válidos se usa a lista da questão "2. Saber quais pacotes possuem códigos de barras

válidos e/ou inválidos;" e comparar se o pacote está na lista e se estiver o armazena na estrutura auxiliar.

Estrutura programa

1. Quebra o código do pacote em 2 partes, jogando a primeira trinca (origem [0:3]) para o final da segunda parte (restante código pacote [3:15]);
2. Com o novo código gerado, o que começa com o destino, se ordena, assim fica agrupado por região;
3. Com a nova ordenação da estrutura, se une as 2 partes ao estado original jogando a parte final (origem [12:15]) no começo da parte 1 (restante código pacote [0:12]);
4. Imprime a estrutura que foi agrupada por região comparando o intervalo da região com a trinca destino [3:6].

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes; * 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos.

[101]:
```
from collections import OrderedDict
#cria dicionario pacAux que não referencia mesmo endereco que dicionario pacotes
pacAux = dict(pacotes)

# 1. quebra codigo do pacote em 2 partes
for chave in pacAux.keys():
  if(chave in valido):
    pacAux[chave] = pacAux[chave][3:15]+pacAux[chave][0:3]
    #print(pac[chave])

# 2. ordena de acordo com destino
pacAux = OrderedDict(sorted(pacAux.items(), key=lambda x: x[1]))

# 3. une as 2 partes ao estado original
for chave in pacAux.keys():
  if(chave in valido):
    pacAux[chave] = pacAux[chave][12:15]+pacAux[chave][0:12]
    #print(pac[chave])

# 4. agrupado e imprime
for chave in pacAux.keys():
  if(chave in valido):
    if(int(pacAux[chave][3:6])>=201 and int(pacAux[chave][3:6])<=299):
      print("{}: com destino na Cidade {}, região Centro-Oeste   " .
 ↪format(str(chave), pacAux[chave][3:6]))
    elif(int(pacAux[chave][3:6])>=300 and int(pacAux[chave][3:6])<=399):
      print("{}: com destino na Cidade {}, região Nordeste       " .
 ↪format(str(chave), pacAux[chave][3:6]))
    elif(int(pacAux[chave][3:6])>=400 and int(pacAux[chave][3:6])<=499):
      print("{}: com destino na Cidade {}, região Norte          " .
 ↪format(str(chave), pacAux[chave][3:6]))
```

```
    elif(int(pacAux[chave][3:6])>=1 and int(pacAux[chave][3:6])<=99):
      print("{}: com destino na Cidade {}, região Sudeste        " .
↪format(str(chave), pacAux[chave][3:6]))
    elif(int(pacAux[chave][3:6])>=100 and int(pacAux[chave][3:6])<=199):
      print("{}: com destino na Cidade {}, região Sul            " .
↪format(str(chave), pacAux[chave][3:6]))
pacAux.clear()
```

```
Pacote 19: com destino na Cidade 044, região Sudeste
Pacote 7: com destino na Cidade 055, região Sudeste
Pacote 4: com destino na Cidade 111, região Sul
Pacote 10: com destino na Cidade 188, região Sul
Pacote 11: com destino na Cidade 288, região Centro-Oeste
Pacote 20: com destino na Cidade 311, região Nordeste
Pacote 14: com destino na Cidade 311, região Nordeste
Pacote 6: com destino na Cidade 333, região Nordeste
Pacote 15: com destino na Cidade 333, região Nordeste
Pacote 9: com destino na Cidade 333, região Nordeste
Pacote 8: com destino na Cidade 333, região Nordeste
Pacote 2: com destino na Cidade 333, região Nordeste
Pacote 1: com destino na Cidade 355, região Nordeste
Pacote 18: com destino na Cidade 388, região Nordeste
Pacote 12: com destino na Cidade 388, região Nordeste
Pacote 17: com destino na Cidade 448, região Norte
Pacote 16: com destino na Cidade 488, região Norte
```

0.7 5. Listar o número de pacotes enviados por cada vendedor (Considere apenas pacotes válidos);

ideia: Criar uma estrutura que armazene o vendedor e associa a este a quantidade de pacotes enviados. Para isso se copia cada vendedor válido pegando a trinca do código do vendedor [9:12] e associa o valor 0 a cada um, em seguida se incrementa na estrutura criada a cada pacote enviado por cada vendedor, até que não haja mais pacotes.

Estrutura programa

1. Criar um dicionário e armazenar nele o vendedor (válido) obtido pela trinca [9:12] e para cada vendedor, inicializar eles com o valor 0;
2. Conta a quantidade de pacotes enviados por cada vendedor através de um laço que incrementa 1 ao vendedor por pacote enviado, isso no dicionário criado;
3. Imprime o vendedor e a quantidade de unidades, se ”unidades” for no singular há uma condição de como se deve imprimir.

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes; * 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos.

[102]:
```
dictVendedores = {}
# 1. inicia dicionario vendedores e com 0
for chave in pacotes.keys():
    if(chave in valido):
      dictVendedores[pacotes[chave][9:12]] = 0

#2. conta pacotes enviados por cada vendedor
for chave in pacotes.keys():
    if(chave in valido):
      dictVendedores[pacotes[chave][9:12]] += 1

#3. Imprime o vendedor e unidades
print("VENDEDOR | PACOTES ENVIADOS")
for chave in dictVendedores.keys():
  if(dictVendedores[chave] >= 1):
    print("  {} \t     {} unidades" .format(chave, dictVendedores[chave]))
  else:
    print("  {} \t     {} unidade" .format(chave, dictVendedores[chave]))
```

```
VENDEDOR | PACOTES ENVIADOS
  123        8 unidades
  584        4 unidades
  874        3 unidades
  124        1 unidades
  845        1 unidades
```

0.8 6. Gerar o relatório/lista de pacotes por destino e por tipo (Considere apenas pacotes válidos);

ideia: Se usa a questão "4. Listar os pacotes agrupados por região de destino (Considere apenas pacotes válidos)" como base, assim se agrupa por destino. No final se compara a trinca do produto [12:15] com os códigos dos produtos, se forem o mesmo, então imprime o nome do produto.

Estrutura programa

1. Quebra o código do pacote em 2 partes, jogando a primeira trinca (origem [0:3]) para o final da segunda parte (restante código pacote [3:15]);
2. Com o novo código gerado, o que começa com o destino, se ordena, assim fica agrupado por região;
3. Com a nova ordenação da estrutura, se une as 2 partes ao estado original jogando a parte final (origem [12:15]) no começo da parte 1 (restante código pacote [0:12]);
4. Imprime a estrutura que foi agrupada por região comparando o intervalo da região com a trinca destino [3:6];
5. Imprime o tipo de produto comparando o código do produto com a trinca produto [12:15].

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes; * 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos.

[103]:
```
from collections import OrderedDict
#cria dicionario pacAux que não referencia mesmo endereco que dicionario pacotes
pacAux = dict(pacotes)

# 1. quebra codigo do pacote em 2 partes
for chave in pacAux.keys():
  if(chave in valido):
    pacAux[chave] = pacAux[chave][3:15]+pacAux[chave][0:3]
    #print(pacAux[chave])

# 2. ordena de acordo com destino
pacAux = OrderedDict(sorted(pacAux.items(), key=lambda x: x[1]))

# 3. une as 2 partes ao estado original
for chave in pacAux.keys():
  if(chave in valido):
    pacAux[chave] = pacAux[chave][12:15]+pacAux[chave][0:12]
    #print(pacAux[chave])


# 4. agrupado e imprime
print("\t\t\tRELATÓRIO\n PACOTE \t         DESTINO   \t\t      TIPO")
for chave in pacAux.keys():
    if(chave in valido):
      # Região destino
      if(int(pacAux[chave][3:6])>=201 and int(pacAux[chave][3:6])<=299):
        print("{}: \t Cidade {}, região Centro-Oeste   " .format(str(chave),␣
↪pacAux[chave][3:6]), end =" ")
      elif(int(pacAux[chave][3:6])>=300 and int(pacAux[chave][3:6])<=399):
        print("{}: \t Cidade {}, região Nordeste       " .format(str(chave),␣
↪pacAux[chave][3:6]), end =" ")
      elif(int(pacAux[chave][3:6])>=400 and int(pacAux[chave][3:6])<=499):
        print("{}: \t Cidade {}, região Norte          " .format(str(chave),␣
↪pacAux[chave][3:6]), end =" ")
      elif(int(pacAux[chave][3:6])>=1 and int(pacAux[chave][3:6])<=99):
        print("{}: \t Cidade {}, região Sudeste        " .format(str(chave),␣
↪pacAux[chave][3:6]), end =" ")
      elif(int(pacAux[chave][3:6])>=100 and int(pacAux[chave][3:6])<=199):
        print("{}: \t Cidade {}, região Sul            " .format(str(chave),␣
↪pacAux[chave][3:6]), end =" ")

# 5. imprime tipo do produto
      # Tipo produto
      if(int(pacAux[chave][12:15]) == 1):
        print("Jóias")
      elif(int(pacAux[chave][12:15]) == 111):
        print("Livros")
```

```
      elif(int(pacAux[chave][12:15]) == 333):
        print("Eletrônicos")
      elif(int(pacAux[chave][12:15]) == 555):
        print("Bebidas")
      elif(int(pacAux[chave][12:15]) == 888):
        print("Brinquedos")

pacAux.clear()
```

```
                    RELATÓRIO
 PACOTE                  DESTINO                        TIPO
Pacote 19:       Cidade 044, região Sudeste         Eletrônicos
Pacote 7:        Cidade 055, região Sudeste         Brinquedos
Pacote 4:        Cidade 111, região Sul             Bebidas
Pacote 10:       Cidade 188, região Sul             Eletrônicos
Pacote 11:       Cidade 288, região Centro-Oeste    Jóias
Pacote 20:       Cidade 311, região Nordeste        Jóias
Pacote 14:       Cidade 311, região Nordeste        Jóias
Pacote 6:        Cidade 333, região Nordeste        Eletrônicos
Pacote 15:       Cidade 333, região Nordeste        Bebidas
Pacote 9:        Cidade 333, região Nordeste        Jóias
Pacote 8:        Cidade 333, região Nordeste        Eletrônicos
Pacote 2:        Cidade 333, região Nordeste        Eletrônicos
Pacote 1:        Cidade 355, região Nordeste        Brinquedos
Pacote 18:       Cidade 388, região Nordeste        Bebidas
Pacote 12:       Cidade 388, região Nordeste        Bebidas
Pacote 17:       Cidade 448, região Norte           Jóias
Pacote 16:       Cidade 488, região Norte           Eletrônicos
```

0.9 7. Se o transporte dos pacotes para o Norte passa pela Região Centro-Oeste, quais são os pacotes que devem ser despachados no mesmo caminhão?

ideia: Considerando apenas essas duas regiões, se deve despachar os pacotes com destino Norte e Centro-oeste. Para isso quebra em trinca de destino [3:6] e compara se o pacote é válido e se passa pela região centro-oeste ou norte através do intervalo de cada região.

Estrutura programa

1. Se usa parte do código da questão "4. Listar os pacotes agrupados por região de destino (Considere apenas pacotes válidos)" como base para agrupar de acordo com a região, quebrar em 2 partes, ordenar e por fim unir as 2 partes ao estado original;
2. Verifica se pacote é válido, e compara se tem como destino a região Centro-oeste ou norte, se sim imprime dados do pacote.

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes; * 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos.

[104]:
```
from collections import OrderedDict

# cria dicionario pacAux que não referencia mesmo endereco que dicionario␣
 ↪pacotes
pacAux = dict(pacotes)

# 1. quebra codigo do pacote em 2 partes
for chave in pacAux.keys():
  if(chave in valido):
    pacAux[chave] = pacAux[chave][3:15]+pacAux[chave][0:3]
    #print(pacAux[chave])

# 1. ordena de acordo com destino
pacAux = OrderedDict(sorted(pacAux.items(), key=lambda x: x[1]))

# 1. une as 2 partes ao estado original
for chave in pacAux.keys():
  if(chave in valido):
    pacAux[chave] = pacAux[chave][12:15]+pacAux[chave][0:12]
    #print(pacAux[chave])

# 2. compara se o pacote é valido e tem destino Centro-oeste ou Norte
print("    PACOTES A SEREM DESPACHADOS NO MESMO CAMINHÃO")
for chave in pacAux.keys():
    if(chave in valido):
      # centro oeste
      if(int(pacAux[chave][3:6])>=201 and int(pacAux[chave][3:6])<=299):
        print("{}: despachar na Cidade {}, região Centro-Oeste" .
 ↪format(str(chave), pacAux[chave][3:6]))
      elif(int(pacAux[chave][3:6])>=400 and int(pacAux[chave][3:6])<=499):
        print("{}: despachar na Cidade {}, região Norte" .format(str(chave),␣
 ↪pacAux[chave][3:6]))

#pacAux.clear()
```

```
    PACOTES A SEREM DESPACHADOS NO MESMO CAMINHÃO
Pacote 11: despachar na Cidade 288, região Centro-Oeste
Pacote 17: despachar na Cidade 448, região Norte
Pacote 16: despachar na Cidade 488, região Norte
```

0.10 8. Se todos os pacotes fossem uma fila qual seria a ordem de carga para o Norte no caminhão para descarregar os pacotes da Região Centro Oeste primeiro;

ideia: Pensando como o tipo de estrutura fila, se removeria os pacotes colocados primeiro no veículo, porém como os pacotes estão em um caminhão e supondo que a porta de desembarque seja

na traseira, remover os pacotes iniciais pode ser muito trabalhoso. Neste caso os pacotes próximos a porta podem ser mais fáceis de remover, assim usamos uma "fila inversa", no qual o primeiro que entra é o último que sai. Assim se insere no caminhão os pacotes de acordo com a ordem de entrega, assim temos a seguinte ordem de embarque: 1. Sul = 100 - 199 2. Sudeste = 001 - 099 3. Norte = 400 - 499 4. Nordeste = 300 - 399 5. Centro-oeste = 201 - 299

Isso facilita o desembarque dos pacotes. Uma forma de resolver seria criar uma nova estrutura que iria armazenar primeiro todos os produtos válidos do sul, depois sudoeste, até chegar no Centro-oeste, um laço para cada região resolve a tarefa.

Estrutura programa

1. Cria estrutura e usa um for para cada pacote válido passando dados do sul;
2. Usa um for para cada pacote válido passando dados do Sudeste para a estrutura criada;
3. Usa um for para cada pacote válido passando dados do Norte para a estrutura criada;
4. Usa um for para cada pacote válido passando dados do Nordeste para a estrutura criada;
5. Usa um for para cada pacote válido passando dados do Centro-oeste para a estrutura criada;
6. imprime ordem de embarque de cada pacote.

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes; * 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos.

[105]:
```
# trabalhar com todos os pacotes
dictAux2 = {}

# 1. insere sul
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][3:6])>=100 and int(pacotes[chave][3:6])<=199):
      dictAux2[chave] = pacotes[chave]
# 2. insere sudoeste
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][3:6])>=1 and int(pacotes[chave][3:6])<=99):
      dictAux2[chave] = pacotes[chave]
# 3. insere Norte
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][3:6])>=400 and int(pacotes[chave][3:6])<=499):
      dictAux2[chave] = pacotes[chave]
# 4. insere Nordeste
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][3:6])>=300 and int(pacotes[chave][3:6])<=399):
      dictAux2[chave] = pacotes[chave]
# 5. insere Centro-oeste
for chave in pacotes.keys():
  if(chave in valido):
```

```
    if(int(pacotes[chave][3:6])>=201 and int(pacotes[chave][3:6])<=299):
      dictAux2[chave] = pacotes[chave]

# 6. imprime ordem de embarque
print("    ORDEM DE CARGA DO PACOTES NO CAMINHÃO")
for chave in dictAux2.keys():
    if(chave in valido):
      # centro oeste
      if(int(dictAux2[chave][3:6])>=201 and int(dictAux2[chave][3:6])<=299):
        print("{}: despachar na Cidade {}, região Centro-Oeste" .
 ↪format(str(chave), dictAux2[chave][3:6]))
      elif(int(dictAux2[chave][3:6])>=300 and int(dictAux2[chave][3:6])<=399):
        print("{}: despachar na Cidade {}, região Nordeste" .format(str(chave),␣
 ↪dictAux2[chave][3:6]))
      elif(int(dictAux2[chave][3:6])>=400 and int(dictAux2[chave][3:6])<=499):
        print("{}: despachar na Cidade {}, região Norte" .format(str(chave),␣
 ↪dictAux2[chave][3:6]))
      elif(int(dictAux2[chave][3:6])>=1 and int(dictAux2[chave][3:6])<=99):
        print("{}: despachar na Cidade {}, região Sudeste" .format(str(chave),␣
 ↪dictAux2[chave][3:6]))
      elif(int(dictAux2[chave][3:6])>=100 and int(dictAux2[chave][3:6])<=199):
        print("{}: despachar na Cidade {}, região Sul" .format(str(chave),␣
 ↪dictAux2[chave][3:6]))

#dictAux2.clear()
```

```
    ORDEM DE CARGA DO PACOTES NO CAMINHÃO
Pacote 4: despachar na Cidade 111, região Sul
Pacote 10: despachar na Cidade 188, região Sul
Pacote 7: despachar na Cidade 055, região Sudeste
Pacote 19: despachar na Cidade 044, região Sudeste
Pacote 16: despachar na Cidade 488, região Norte
Pacote 17: despachar na Cidade 448, região Norte
Pacote 1: despachar na Cidade 355, região Nordeste
Pacote 2: despachar na Cidade 333, região Nordeste
Pacote 6: despachar na Cidade 333, região Nordeste
Pacote 8: despachar na Cidade 333, região Nordeste
Pacote 9: despachar na Cidade 333, região Nordeste
Pacote 12: despachar na Cidade 388, região Nordeste
Pacote 14: despachar na Cidade 311, região Nordeste
Pacote 15: despachar na Cidade 333, região Nordeste
Pacote 18: despachar na Cidade 388, região Nordeste
Pacote 20: despachar na Cidade 311, região Nordeste
Pacote 11: despachar na Cidade 288, região Centro-Oeste
```

0.11 9. No item acima considerar que as jóias fossem sempre as primeiras a serem descarregadas;

ideia: Mesma base da ideia da questão "8. Se todos os pacotes fossem uma fila qual seria a ordem de carga para o Norte no para descarregar os pacotes da Região Centro Oeste primeiro", porém neste caso as joias devem ser descarregadas antes dos demais pacotes de uma dada região. Para isso basta copiar para uma nova estrutura se todos os produtos com a uma região que não são jóias, em seguida se todos os pacotes daquela mesma região que tem jóia em seu conteúdo e se repete o mesmo processo para todas as regiões.

Estrutura programa

1. Cria estrutura e usa 2 laços para cada pacote válido do Sul, o primeiro laço passa pacotes sem jóia para a estrutura em seguida o outro laço passa pacotes com jóia para a estrutura;
2. Usa 2 laços para cada pacote válido do Sudeste, o primeiro laço passa pacotes sem jóia para a estrutura em seguida o outro laço passa pacotes com jóia para a estrutura;
3. Usa 2 laços para cada pacote válido do Norte, o primeiro laço passa pacotes sem jóia para a estrutura em seguida o outro laço passa pacotes com jóia para a estrutura;
4. Usa 2 laços para cada pacote válido do Nordeste, o primeiro laço passa pacotes sem jóia para a estrutura em seguida o outro laço passa pacotes com jóia para a estrutura;
5. Usa 2 laços para cada pacote válido do Centro-oeste, o primeiro laço passa pacotes sem jóia para a estrutura em seguida o outro laço passa pacotes com jóia para a estrutura;
6. imprime ordem de embarque de tal forma que priorize o desembarque das jóias em cada região.

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes; * 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos.

[106]:
```
dictAux = {}

# 1. insere sul não joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) != 1 and (int(pacotes[chave][3:6])>=100 and␣
 ↪int(pacotes[chave][3:6])<=199)):
      dictAux[chave] = pacotes[chave]
# 1. insere sul  joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) == 1 and (int(pacotes[chave][3:6])>=100 and␣
 ↪int(pacotes[chave][3:6])<=199)):
      dictAux[chave] = pacotes[chave]

# 2. insere sudoeste não joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) != 1 and (int(pacotes[chave][3:6])>=1 and␣
 ↪int(pacotes[chave][3:6])<=99)):
```

```
      dictAux[chave] = pacotes[chave]
# 2. insere sudoeste joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) == 1 and (int(pacotes[chave][3:6])>=1 and␣
 ↪int(pacotes[chave][3:6])<=99)):
      dictAux[chave] = pacotes[chave]

# 3. insere Norte não joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) != 1 and (int(pacotes[chave][3:6])>=400 and␣
 ↪int(pacotes[chave][3:6])<=499)):
      dictAux[chave] = pacotes[chave]
# 3. insere Norte joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) == 1 and (int(pacotes[chave][3:6])>=400 and␣
 ↪int(pacotes[chave][3:6])<=499)):
      dictAux[chave] = pacotes[chave]

# 4. insere Nordeste não joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) != 1 and (int(pacotes[chave][3:6])>=300 and␣
 ↪int(pacotes[chave][3:6])<=399)):
      dictAux[chave] = pacotes[chave]
# 4. insere Nordeste  joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) == 1 and (int(pacotes[chave][3:6])>=300 and␣
 ↪int(pacotes[chave][3:6])<=399)):
      dictAux[chave] = pacotes[chave]

# 5. insere Centro-oeste não joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) != 1 and (int(pacotes[chave][3:6])>=201 and␣
 ↪int(pacotes[chave][3:6])<=299)):
      dictAux[chave] = pacotes[chave]
# 5. insere Centro-oeste joias
for chave in pacotes.keys():
  if(chave in valido):
    if(int(pacotes[chave][12:15]) == 1 and (int(pacotes[chave][3:6])>=201 and␣
 ↪int(pacotes[chave][3:6])<=299)):
      dictAux[chave] = pacotes[chave]
```

```
# imprime pacotes priorizando desembarque de jóias
print("    ORDEM DE CARGA DO PACOTES NO CAMINHÃO")
for chave in dictAux.keys():
    if(chave in valido):
      # centro oeste
      if(int(dictAux[chave][3:6])>=201 and int(dictAux[chave][3:6])<=299):
        if(int(dictAux[chave][12:15]) == 1):
          print("{}: despachar na Cidade {}, região Centro-Oeste (JÓIA)" .
↪format(str(chave), dictAux[chave][3:6]))
        else:
          print("{}: despachar na Cidade {}, região Centro-Oeste" .
↪format(str(chave), dictAux[chave][3:6]))
      # nordeste
      elif(int(dictAux[chave][3:6])>=300 and int(dictAux[chave][3:6])<=399):
        if(int(dictAux[chave][12:15]) == 1):
          print("{}: despachar na Cidade {}, região Nordeste (JÓIA)" .
↪format(str(chave), dictAux[chave][3:6]))
        else:
          print("{}: despachar na Cidade {}, região Nordeste" .
↪format(str(chave), dictAux[chave][3:6]))
      # norte
      elif(int(dictAux[chave][3:6])>=400 and int(dictAux[chave][3:6])<=499):
        if(int(dictAux[chave][12:15]) == 1):
          print("{}: despachar na Cidade {}, região Norte (JÓIA)" .
↪format(str(chave), dictAux[chave][3:6]))
        else:
          print("{}: despachar na Cidade {}, região Norte" .format(str(chave),␣
↪dictAux[chave][3:6]))
      # sudoeste
      elif(int(dictAux[chave][3:6])>=1 and int(dictAux[chave][3:6])<=99):
        if(int(dictAux[chave][12:15]) == 1):
          print("{}: despachar na Cidade {}, região Sudeste (JÓIA)" .
↪format(str(chave), dictAux[chave][3:6]))
        else:
          print("{}: despachar na Cidade {}, região Sudeste" .
↪format(str(chave), dictAux[chave][3:6]))
      # sul
      elif(int(dictAux[chave][3:6])>=100 and int(dictAux[chave][3:6])<=199):
        if(int(dictAux[chave][12:15]) == 1):
          print("{}: despachar na Cidade {}, região Sul (JÓIA)" .
↪format(str(chave), dictAux[chave][3:6]))
        else:
          print("{}: despachar na Cidade {}, região Sul" .format(str(chave),␣
↪dictAux[chave][3:6]))
```

```
   ORDEM DE CARGA DO PACOTES NO CAMINHÃO
```

```
Pacote 4: despachar na Cidade 111, região Sul
Pacote 10: despachar na Cidade 188, região Sul
Pacote 7: despachar na Cidade 055, região Sudeste
Pacote 19: despachar na Cidade 044, região Sudeste
Pacote 16: despachar na Cidade 488, região Norte
Pacote 17: despachar na Cidade 448, região Norte (JÓIA)
Pacote 1: despachar na Cidade 355, região Nordeste
Pacote 2: despachar na Cidade 333, região Nordeste
Pacote 6: despachar na Cidade 333, região Nordeste
Pacote 8: despachar na Cidade 333, região Nordeste
Pacote 12: despachar na Cidade 388, região Nordeste
Pacote 15: despachar na Cidade 333, região Nordeste
Pacote 18: despachar na Cidade 388, região Nordeste
Pacote 9: despachar na Cidade 333, região Nordeste (JÓIA)
Pacote 14: despachar na Cidade 311, região Nordeste (JÓIA)
Pacote 20: despachar na Cidade 311, região Nordeste (JÓIA)
Pacote 11: despachar na Cidade 288, região Centro-Oeste (JÓIA)
```

0.12 10. Listar os pacotes inválidos.

ideia: Comparar os pacotes com todas as restrições e adicionar em uma lista a causa caso haja alguma restrição para aquele pacote, por fim associa essa lista com o pacote invalido e imprime na tela os dados do pacote e a causa.

Estrutura programa

1. Compara se tamanho do código é diferente de 15;
2. Compara se jóias tem como região de origem o Centro-oeste;
3. O vendedor 367 é invalido;
4. Verifica se produto é válido;
5. Verifica se região origem é válida;
6. Verifica se região destino é válida;
7. Armazena pacote e com ele associa a causa do erro;
8. Imprime pacotes inválidos.

Para executar este bloco corretamente é necessário executar os seguintes blocos antes: * 0. Dicionários Pacotes; * 2. Saber quais pacotes possuem códigos de barras válidos e/ou inválidos.

[107]:
```
dicInvalido = {}
causaErro = []
for chave in pacotes.keys():
  # print(pacotes[chave])

  if(chave in invalido):
    # 1. compara se tamanho do código é diferente de 15
    if( len(pacotes[chave]) != 15):
      causaErro.append("código diferente de 15 digitos")
```

```
    # 2. compara se jóias tem como região de origem o Centro-oeste
    if((int(pacotes[chave][12:15]) == 1) and
        (int(pacotes[chave][0:3])>=201 and int(pacotes[chave][0:3])<=299) ):
      causaErro.append("não é permitido jóias tendo como origem o Centro-oeste")

    # 3. o vendedor 367 é invalido
    if(int(pacotes[chave][9:12]) == 367):
      causaErro.append("vendedor 367 está inativo")

    # 4. verifica se produto é valido
    if( (int(pacotes[chave][12:15]) != 1 and int(pacotes[chave][12:15]) != 111␣
↪and
        int(pacotes[chave][12:15]) != 333 and int(pacotes[chave][12:15]) != 555␣
↪and
        int(pacotes[chave][12:15]) != 888) ):
      causaErro.append("tipo de produto invalido")

    # 5. verifica se região origem é valida
    if( (int(pacotes[chave][0:3]) == 0) or (int(pacotes[chave][0:3]) == 200)
        or (int(pacotes[chave][0:3]) >= 500 and int(pacotes[chave][0:3]) <=␣
↪999) ):
      causaErro.append("região de origem invalida")

    # 6. verifica se região destino é valida
    if( (int(pacotes[chave][3:6]) == 0) or (int(pacotes[chave][3:6]) == 200)
        or (int(pacotes[chave][3:6]) >= 500 and int(pacotes[chave][3:6]) <=␣
↪999) ):
      causaErro.append("região de destino invalida")

    # 7. armazena pacote e causa do erro
    dicInvalido[chave] = list(causaErro)
    causaErro.clear()

# 8. imprime pacotes invalidos
for chave in dicInvalido.keys():
  if(chave in invalido):
    print("{}: erro {}" .format(str(chave), dicInvalido[chave]))
```

```
Pacote 3: erro ['não é permitido jóias tendo como origem o Centro-oeste']
Pacote 5: erro ['tipo de produto invalido']
Pacote 13: erro ['vendedor 367 está inativo', 'região de destino invalida']
```