SIMPAR

Simulação de passageiros em Partida Aérea

Licenciatura em Informática de Gestão
1º Ano – Pós-Laboral

FASES 1 e 2 do trabalho de Algoritmos e Estruturas de Dados
Junho de 2018

Ana Rita Carvalho – 30001683
Bernardo Santa Clara Gomes – 30001687
Emanuel Pedrosa - 30000656
Rui Ramos - 30000402

Prof. Dr. Paulo Enes da Silveira

UNIVERSIDADE AUTÓNOMA DE LISBOA

ÍNDICE

INTRODUÇÃO

PROBLEMAS E OBJECTIVOS A ATINGIR

Com o objectivo de resolver o problema colocado neste trabalho tentamos inicialmente colocar em práctica os conhecimentos, conceitos e competências adquiridos no âmbito desta cadeira nas aulas teoricas e práticas.

Aquando da interpretação do enunciado, é imperativo identificar os parâmetros pelos quais este se rege, estabelecendo desde cedo tarefas e compreender o objectivo final da Simulação. Vários destes parâmetros serão definidos aleatoriamente, outros serão estabelecidos pelo utilizador final.

O principal objectivo a que este grupo se propôs foi o de conseguir finalizar a simulação em código Phyton, através do *interface Spyder*, com todas as funções e correspondendo a todos os requisitos que constam do enunciado.

UNIVERSIDADE AUTÓNOMA DE LISBOA

DESCRIÇÃO DAS ESTRUTURAS DE DADOS

Optámos pela criação de duas classes:

- class **Passageiro**: Descreve um passageiro, número de bagagens, instante em que foi colocado na queue.

- class **Balcao**: Descreve um balcão e a respectiva fila de passageiros.
  Nesta classe estão definidas várias funções, para inicializar o balcão, aumular o tempo de atendimento do passageiro. No fim, demonstra o total de passageiros atendidos por balcão, o tempo médio de atendimento e o número médio de bagagens por passageiros.

- class **TreeNode:** Descreve a estrutura de dados da árvore.

- class **BinarySearchTree:** Define como os dados são adicionados na estrutura, e como é feita a sua pesquisa na mesma.

UNIVERSIDADE AUTÓNOMA DE LISBOA

DESCRIÇÃO SUMÁRIA DO CÓDIGO

O código que consta do nosso projecto,foi elaborado com recurso a Classes, como tal criámos duas Classes – *Passageiro* e *Balcao*.

Tem como objectivo simular o atendimento de passageiros num aeroporto. Começa por solicitar ao utilizador os dados para a simulaçao:

- Número máximo de passageiros a atender
- Número máximo e bagagens por passageiro
- Número de balcões de atendimento
- Número de ciclos de tempo a executar (enquanto atende e recebe passageiros nas filas dos balcões).
- Percentagem de passageiros a colocar na fila de espera antes da abertura dos balcões.

O balcões são gerados, sendo guardado o seu numero e o numero de bagagens que atende por ciclo (feito de forma aleatória).

A colocação de passageiros na fila dos balcões, pode ser feita numa fase inicial, consoante escolha do utilizador, utilzando para isso uma determinada percentagem que já estará em fila quando começar o atendimento (como as filas estão vazias, optou-se por colocar lá os passageiros de maneira distribuida uniformemente).

Após este passo iniciamos os ciclos de tempo (cada ciclo de tempo percorre todos os balcões, tanto para colocar passageiros na fila como para atende-los).

Colocamos passageiros na fila de espera (caso existam), segundo uma lógica de procurar as filas fazias ou com menos pessoas e escolher uma delas.

Uma vez iniciado o atendimento, o passageiro é atendido e retirado da fila, se o seu numero de bagagens for menor ou igual ao numero de bagagens que o balcão, ficando para o(s) ciclo(s) seguintes, se for caso disso.

Se todos os passageiros forem atendidos antes dos ciclos chegarem ao fim, não se executam mais ciclos, não se aceitam mais passageiros nas filas e fazemos o output das estatisticas pedidas.

Chegando ao fim dos ciclos previstos, temos de verificar se existe algum balcão, ou balcões, com filas com passageiros por atender. Nesse caso fazemos os ciclos de tempo necessários para poder esvaziar (atender) as mesmas. De notar que não percorremos os balcões à procura de passageiros , e atendemos (isso criaria um numero de ciclos incorrecto), mas percorremos os balcões enquanto existem filas cheias.

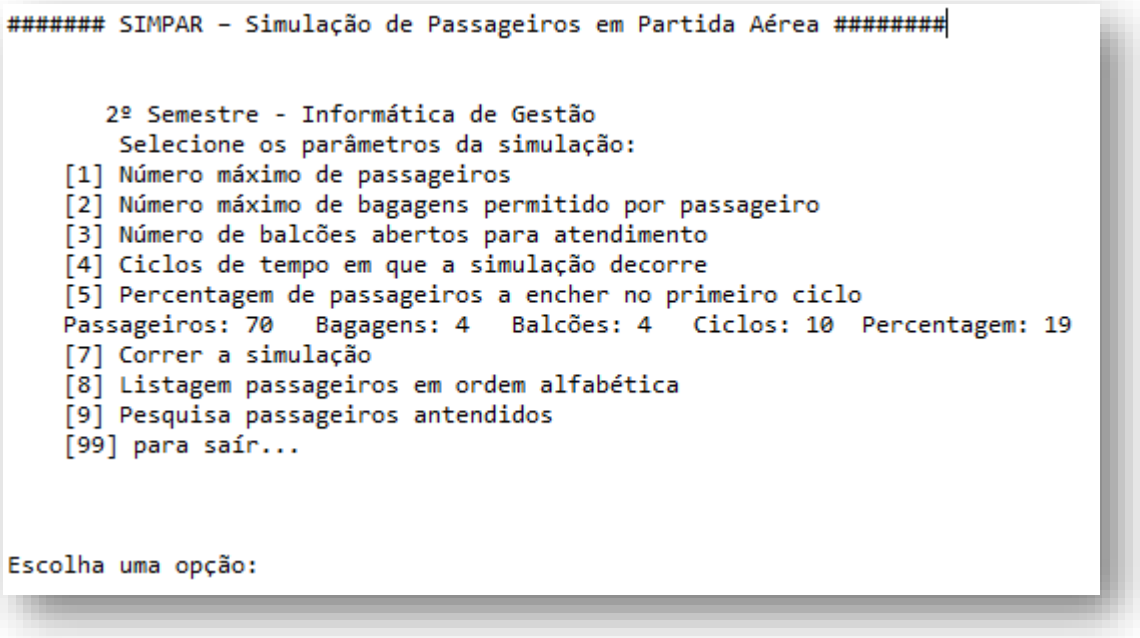
Foram criadas 2 classes, *TreeNode* e *BinarySearchTree,* que trabalhando em conjunto, são a base da nossa árvore. O seu principal objectivo é ordenar os nome dos passageiros (gerados aleatoriamente).

No final são apresentados os outputs pedidos, explicados na proxima secção.

MANUAL DO UTILIZADOR

O manual tem como principal objectivo ajudar o utilizador a navegar no programa SIMPAR. A opção **[99]** tem como objectivo sair do menu; e a tecla <ENTER> continuar.

No Menu Inicial de Chegada o utilizador escolhe uma das opções consoante o seu objectivo.



[1 – 5] – Cada número corresponde a um input dado pelo utilizador. Caso não seja introduzido qualquer valor, o programa inclui números pré-definidos.

[1] Número maximo de passageiros
[2] Número máximo de bagagens permitido por passageiro
[3] Número de balcões abertos para atendimento
[4] Ciclos de tempo em que a simulação decorre
[5] Percentagem de passageiros a encher no primeiro ciclo

Depois de introduzidos os valores requeridos, o programa vai mostrando os mesmos entre as opções [5] e [7].

```
######## SIMPAR – Simulação de Passageiros em Partida Aérea ########


       2º Semestre - Informática de Gestão
        Selecione os parâmetros da simulação:
    [1] Número máximo de passageiros
    [2] Número máximo de bagagens permitido por passageiro
    [3] Número de balcões abertos para atendimento
    [4] Ciclos de tempo em que a simulação decorre
    [5] Percentagem de passageiros a encher no primeiro ciclo
    Passageiros: 56   Bagagens: 3   Balcões: 3   Ciclos: 9  Percentagem: 19
    [7] Correr a simulação
    [8] Listagem passageiros em ordem alfabética
    [9] Pesquisa passageiros antendidos
    [99] para saír...


Escolha uma opção:
```

[7] – Para correr a Simulação com os dados inseridos previamente.

```
Escolha uma opção: 7
«««« CICLO n.º 1 »»»»
Balcão 1 tempo 0 : - [b:1 t:1] [b:3 t:0] [b:1 t:0] [b:1 t:0] [b:1 t:0] [b:3 t:0] [b:3 t:0]
[b:1 t:0] [b:2 t:0] [b:3 t:0] [b:2 t:0] -
Balcão 2 tempo 0 : - [b:2 t:1] [b:3 t:0] [b:1 t:0] [b:1 t:0] [b:2 t:0] [b:3 t:0] [b:3 t:0]
[b:1 t:0] [b:3 t:0] [b:3 t:0] [b:1 t:0] -
Balcão 3 tempo 0 : - [b:2 t:1] [b:2 t:0] [b:2 t:0] [b:3 t:0] [b:3 t:0] [b:1 t:0] [b:2 t:0]
[b:2 t:0] [b:1 t:0] [b:2 t:0] [b:1 t:0] -
```

Durante a execução é mostrado:
«««« CICLO n.º 1 »»»» - Ciclo a ser executado
Balcão 1 tempo 0 : - [b:1 t:1] [b:3 t:0] [b:1 t:0] – A fila de espera do balcão, com b= bagagens do passageiro e t=ciclo em que foi gerado, neste caso 3 passageiros.
Atendido passageiro com 2 bagagens no balcão 2 com tempo de espera 1 – O atendimento realizado com a informação do numero de bagagens e tempo de espera 1 (ciclos de atendimento).

Balcão 1 despachou 1 bagagens por ciclo: - número de balcão e máximo de bagagens que o balcão despacha por ciclo
8 passageiros atendidos com média de bagagens / passageiro = 2.8 – Total de passageiros atendidos e a média de bagagens por passageiro
Tempo médio de espera = 0.8 – O tempo médio de espera

```
********** Fechou a chegada de novos passageiros **********
«««« CICLO ESVAZIA n.º 10 »»»»
Atendido passageiro  com 1 bagagens no balcão 1 com tempo de espera 8
Atendido passageiro  com 1 bagagens no balcão 2 com tempo de espera 8
Atendido passageiro  com 2 bagagens no balcão 3 com tempo de espera 8
«««« CICLO ESVAZIA n.º 11 »»»»
Atendido passageiro  com 3 bagagens no balcão 1 com tempo de espera 8
Atendido passageiro  com 3 bagagens no balcão 2 com tempo de espera 8
Atendido passageiro  com 2 bagagens no balcão 3 com tempo de espera 8
```

[8] – Listagem passageiros em ordem alfabética.

```
Escolha uma opção: 8
Alicia Hsu
Amanda Fisher
Amy Martin
Benjamin Kolb
Coy Smith
Crystal Beekman
Debra Brooks
Ernest Cheek
Henry Valenzuela
Humberto Schreckengost
Irene Dobson
James Bain
John Arnold
Leonard Koonce
Mayme Smith
Orlando Smith
Paul Dyer
Tammy Sutton
Virginia Hernandez
```

[9] – Pesquisa passageiros atendidos.

```
Escolha uma opção: 9

Qual o nome do passageiro a pesquisar?
Sue Haans
Pesquisa - Sue Haans: passageiro não embarcado

Prima <ENTER> para continuar . . .
```

[99] – Para sair do programa SIMPAR.

```
...adeus :(
```

CONCLUSÕES

DIFICULDADES

Desde já uma grande dificuldade que existiu na execução deste trabalho foi conciliar as nossas vidas profissionais e pessoais, com a exigência que este trabalho nos colocou.

A questão dos ciclos foi considerada a mais desafiante. Quando o número de ciclos pré-determinado acaba, mas ainda existirem passageiros por atender causou-nos algum transtorno.

Sentimos algumas dificuldades na interpretação do enunciado e ao decidir o "aspecto final" do programa. Foi tomando diversas formas à medida que fomos progredindo com o mesmo.

Na segunda fase do trabalho, já com a uma árvore BST, deparámo-nos com algumas dificuldades, especialmente na sua ordenação. O código originalmente fornecido, no exemplo das aulas práticas, utilizava um valor numérico para a *key* e um *val* para o valor uma string associada. O que implicava que para haver uma ordenação alfabética, essa teria que estar definida no valor colocado na *key*. Após vários testes ao código percebemos que a *key* pode ser uma string (isto porque a linguagem "sabe" que 'a'<'b', logo que 'Bernardo < 'Rui'. De modo a não existirem variáveis sem uso, optámos por retirar o *val* do algoritmo.

METAS ATINGIDAS

Criação de um menu interactivo permitindo ao utilizador final inserir os dados que desejar. Após a inserção dos mesmos, é possivel calcular o tempo de espera por balcão consoante o número de bagagens; calcular o número de bagagens despachada por balcão por ciclo.

Na segunda fase, conseguimos cumprir os objectivos a que nos propusemos. Criação de um ficheiro com o output final. Criação de uma árvore BST onde o nome de cada passageiro (gerado aleatoriamente) é inserido. É possivel também a pesquisa de passageiros, inserindo o nome no teclado, confirmando se este mesmo existe na árvore.

Por fim, conforme requerido no enunciado, através de um input de menu, afixar os nomes dos passageiros na árvore alfabeticamente.

FUNÇÕES E TAREFAS

É para nós definir as tarefas específicas de cada um devido ao entrosamento que temos entre os 4 elementos. Isto porque aproveitamos ao máximo as competências de cada um (sejam elas por vocação ou por experiência profissional).

Competências que podemos descrever como: análise crítica, análise funcional, programação, manipulação de informação, expressão escrita.

APÊNDICES

CÓDIGO FONTE

```
from pythonds import Queue
from random import randint, choice
from shutil import get_terminal_size
import names, pickle, math


class TreeNode:
    def __init__(self,key,left=None,right=None,parent=None):
        self.key = key
        self.leftChild = left
        self.rightChild = right
        self.parent = parent

    def hasLeftChild(self):
        return self.leftChild

    def hasRightChild(self):
        return self.rightChild

    def isLeftChild(self):
        return self.parent and self.parent.leftChild == self

    def isRightChild(self):
        return self.parent and self.parent.rightChild == self

    def isRoot(self):
        return not self.parent

    def isLeaf(self):
        return not (self.rightChild or self.leftChild)

    def hasAnyChildren(self):
        return self.rightChild or self.leftChild

    def hasBothChildren(self):
        return self.rightChild and self.leftChild

    def replaceNodeData(self,key,lc,rc):
        self.key = key
        self.leftChild = lc
        self.rightChild = rc
        if self.hasLeftChild():
            self.leftChild.parent = self
        if self.hasRightChild():
            self.rightChild.parent = self

    def preorder(self):
        print(self.key)
        if self.leftChild:
            self.leftChild.preorder()
        if self.rightChild:
            self.rightChild.preorder()
```

```
    def inorder(self):
        if self.leftChild:
            self.leftChild.inorder()
        print(str(self.key))
        if self.rightChild:
            self.rightChild.inorder()

    def postorder(self):
        if self.leftChild:
            self.leftChild.postorder()
        if self.rightChild:
            self.rightChild.postorder()
        print(self.key)


class BinarySearchTree:

    def __init__(self):
        self.root = None
        self.size = 0

    def length(self):
        return self.size

    def __len__(self):
        return self.size

    def put(self,key):
        if self.root:
            self._put(key,self.root)
        else:
            self.root = TreeNode(key)
        self.size = self.size + 1

    def _put(self,key,currentNode):
        if key < currentNode.key:
            if currentNode.hasLeftChild():
                self._put(key,currentNode.leftChild)
            else:
                currentNode.leftChild = TreeNode(key,parent=currentNode)
        else:
            if currentNode.hasRightChild():
                self._put(key,currentNode.rightChild)
            else:
                currentNode.rightChild = TreeNode(key,parent=currentNode)

    def __setitem__(self,k):
        self.put(k)

    def get(self,key):
        if self.root:
            res = self._get(key,self.root)
            if res:
                return res
            else:
                return None
        else:
            return None

    def _get(self,key,currentNode):
```

```
  if not currentNode:
     return None
  elif currentNode.key == key:
     return currentNode
  elif key ← currentNode.key:
     return self._get(key,currentNode.leftChild)
  else:
     return self._get(key,currentNode.rightChild)

def __getitem__(self,key):
  return self.get(key)

def __contains__(self,key):
  if self._get(key,self.root):
     return True
  else:
     return False

def delete(self,key):
 if self.size → 1:
    nodeToRemove = self._get(key,self.root)
    if nodeToRemove:
       self.remove(nodeToRemove)
       self.size = self.size-1
    else:
       raise KeyError('Error, key not in tree')
 elif self.size == 1 and self.root.key == key:
    self.root = None
    self.size = self.size - 1
 else:
    raise KeyError('Error, key not in tree')

def __delitem__(self,key):
  self.delete(key)

def spliceOut(self):
  if self.isLeaf():
     if self.isLeftChild():
           self.parent.leftChild = None
     else:
           self.parent.rightChild = None
  elif self.hasAnyChildren():
     if self.hasLeftChild():
           if self.isLeftChild():
             self.parent.leftChild = self.leftChild
           else:
             self.parent.rightChild = self.leftChild
           self.leftChild.parent = self.parent
     else:
           if self.isLeftChild():
             self.parent.leftChild = self.rightChild
           else:
             self.parent.rightChild = self.rightChild
           self.rightChild.parent = self.parent

def findSuccessor(self):
 succ = None
 if self.hasRightChild():
    succ = self.rightChild.findMin()
 else:
    if self.parent:
```

```
            if self.isLeftChild():
               succ = self.parent
            else:
               self.parent.rightChild = None
               succ = self.parent.findSuccessor()
               self.parent.rightChild = self
     return succ

   def findMin(self):
     current = self
     while current.hasLeftChild():
        current = current.leftChild
     return current

   def remove(self,currentNode):
       if currentNode.isLeaf(): #leaf
         if currentNode == currentNode.parent.leftChild:
            currentNode.parent.leftChild = None
         else:
            currentNode.parent.rightChild = None
       elif currentNode.hasBothChildren(): #interior
         succ = currentNode.findSuccessor()
         succ.spliceOut()
         currentNode.key = succ.key
         currentNode.payload = succ.payload

       else: # this node has one child
         if currentNode.hasLeftChild():
           if currentNode.isLeftChild():
              currentNode.leftChild.parent = currentNode.parent
              currentNode.parent.leftChild = currentNode.leftChild
           elif currentNode.isRightChild():
              currentNode.leftChild.parent = currentNode.parent
              currentNode.parent.rightChild = currentNode.leftChild
           else:
              currentNode.replaceNodeData(currentNode.leftChild.key,
                                  currentNode.leftChild.payload,
                                  currentNode.leftChild.leftChild,
                                  currentNode.leftChild.rightChild)
         else:
           if currentNode.isLeftChild():
              currentNode.rightChild.parent = currentNode.parent
              currentNode.parent.leftChild = currentNode.rightChild
           elif currentNode.isRightChild():
              currentNode.rightChild.parent = currentNode.parent
              currentNode.parent.rightChild = currentNode.rightChild
           else:
              currentNode.replaceNodeData(currentNode.rightChild.key,
                                  currentNode.rightChild.payload,
                                  currentNode.rightChild.leftChild,
                                  currentNode.rightChild.rightChild)

# ********** O código em baixo vai limpar o ecrã de forma a facilitar a leitura ********** #

def limpa():
   print("\n" * get_terminal_size().lines, end="")


# ********** Menu Inicial de Chegada ********** #

def menu():
```

```
    limpa()
    limpa()
    print("\n")
    print("######## SIMPAR – Simulação de Passageiros em Partida Aérea ########")
    print("\n")
    print("""      2º Semestre - Informática de Gestão
       Selecione os parâmetros da simulação:
    [1] Número máximo de passageiros
    [2] Número máximo de bagagens permitido por passageiro
    [3] Número de balcões abertos para atendimento
    [4] Ciclos de tempo em que a simulação decorre
    [5] Percentagem de passageiros a encher no primeiro ciclo
    Passageiros: {}   Bagagens: {}   Balcões: {}   Ciclos: {}  Percentagem: {}
    [7] Correr a simulação
    [8] Listagem passageiros em ordem alfabética
    [9] Pesquisa passageiros antendidos
    [99] para saír...""".format(passa, bag, balc, cicl, pench))


class Passageiro:
    """
    Descreve um passageiro
    """

    def __init__(self, bag_pass, ciclo_in):
        """
        Inicializa um passageiro
        :param bag_pass: número de bagagens do passageiro
        :param ciclo_in: instante em que foi colocado na fila (número do ciclo da simulação)
        """

        self.bag_pass = bag_pass
        self.ciclo_in = ciclo_in
        # self.atendidos = 0

    def obtem_bag_pass(self):
        """
        Devolve o valor de bag_pass
        :return: bag_pass
        """

        return self.bag_pass

    def obtem_ciclo_in(self):
        """
        devolve o valor de ciclo_in
        :return: ciclo_in
        """

        return self.ciclo_in

#    def incr_atendidos(self):
#        """
#        Incrementa em 1 o passt_atend - total de passageiros atendidos
#        :return: None
#        """
#
#        self.atendidos += 1

    def __str__(self):
        """
```

```
        Retorna o passageiro como uma string legivel para o utilizador
        Output esperado:
            [b:4 t:2]
        :return: string
        """

        return "[b:{} t:{}]".format(self.obtem_bag_pass(), self.obtem_ciclo_in())


class Balcao:
    """
    Descreve um balcão e a respectiva fila de passageiros
    """

    def __init__(self, n_balcao, num_bag):
        """
        Inicializa um balcão com o número indicado
        :param n_balcao: número do balcão
        :param num_bag: o número máximo de bagagens permitido por passageiro
        """

        self.n_balcao = n_balcao

        self.fila = Queue()
        self.inic_atend = 0
        self.passt_atend = 0
        self.numt_bag = 0
        self.tempt_esp = 0
        self.bag_utemp = randint(1, num_bag)

    def obtem_n_balcao(self):
        """
        Devolve o valor de n_balcao
        :return: n_balcao
        """

        return self.n_balcao

    def obtem_fila(self):
        """
        Devolve o valor da fila
        :return: fila
        """

        return self.fila

    def muda_inic_atend(self, tempo_atendimento):
        """
        Acumula em inic_atend o "valor" do tempo de atendimento do passageiro
        :param tempo_atendimento: tempo de atendimento
        :return: None
        """

        self.inic_atend = tempo_atendimento

    def incr_passt_atend(self):
        """
        Incrementa em 1 o passt_atend - total de passageiros atendidos por este balcão
        :return: None
        """
```

```
        self.passt_atend += 1

    def muda_numt_bag(self, passageiro):
        """
        Acumula em numt_bag do balcão, o bag_pass do passageiro quando este termina de ser atendido
        :param passageiro: passageiro processado
        :return: None
        """

        self.numt_bag += passageiro.obtem_bag_pass()

    def muda_tempt_esp(self, tempo_espera):
        """
        Acumula em tempt_esp o "t" tempo de espera do passageiro
        :param tempo_espera: Tempo de espera
        :return: None
        """

        self.tempt_esp += tempo_espera

    def __str__(self):
        """
        Retorna o balcão como uma string legível para o utilizador
        Output esperado:
            Quando tem passageiros na fila:
                Balcão 2 tempo 2 : - [b:4 t:1] [b:2 t:2] -
            Quando não tem passageiros na fila:
                Balcão 0 tempo 1 : -
        :return: string
        """

        # Formata a lista de passageiros consoante as especificações
        if self.fila.isEmpty():
            str_pass = "-"
        else:
            passageiros_como_str = [str(passageiro) for passageiro in self.fila.items]
            str_pass = "- {} - ".format(" ".join(passageiros_como_str))

        return "Balcão {} tempo {} : {}".format(self.obtem_n_balcao(), self.tempt_esp, str_pass)


def mostra_balcoes(balcoes):
    """
    Mostra os detalhes dos balcoes
    :param balcoes: Lista de balcões
    :return: None
    """

    for balcao in balcoes:
        print(str(balcao))



#ponto 4.3
def atende_passageiros(tempo, balcoes):
    """
    Atende passageiros nos balcões indicados
    :param tempo: Ciclo de simulação
    :param balcoes: Lista de balcões
    :return: Passageiros colocados em fila
    """
```

```
    atendidos = 0
    p_nome=''
    for b in balcoes:
        if b.obtem_fila().isEmpty():
            # Sem passageiros a processar
            print('BALCÃO ' + str(b) + ' sem passageiros a processar')
            b.muda_inic_atend(tempo)
            continue

        fila = b.obtem_fila()
        p = fila.items[-1]  # Para ser Fifo, tem de ser desta forma porque Queue.enqueue() acrescenta no inicio da lista
        tempo_atendimento = tempo + b.inic_atend
        ut_bag = math.ceil(p.bag_pass / b.bag_utemp)
        if ut_bag <- tempo_atendimento:
            tempo_de_espera = tempo - p.ciclo_in
            # p_nome=names.get_full_name()
            print("Atendido passageiro {} com {} bagagens no balcão {} com tempo de espera {}".format(
                    p_nome,
                    p.bag_pass,
                    b.obtem_n_balcao(),
                    tempo_de_espera
                )

            )
            b.muda_inic_atend(tempo + 1)
            b.incr_passt_atend()
            b.muda_numt_bag(p)
            b.muda_tempt_esp(tempo_de_espera)
            fila.items.remove(p)
            atendidos += 1
            #passTree.put(p_nome)

    return atendidos

#ponto 4.4
def apresenta_resultados(balcoes):
    """
    Apresenta os resultados estatísticos finais
    :param balcoes: Lista de balcões
    :return: None
    """
    lista_tmp=[]
    for i in balcoes:
        if i.passt_atend -> 0:
            tmp="Balcão {} despachou {} bagagens por ciclo:".format(i.obtem_n_balcao(), i.bag_utemp)
            lista_tmp.append(tmp)
            print (tmp)

            #print("Balcão {} despachou {} bagagens por ciclo:".format(i.obtem_n_balcao(), i.bag_utemp))
            tmp=(
                "{} passageiros atendidos com média de bagagens / passageiro = {}".format(
                    i.passt_atend,
                    round(i.numt_bag / i.passt_atend, 1)
                )
            )
            lista_tmp.append(tmp)
            print(tmp)
            tmp=("Tempo médio de espera = {}".format(round(i.passt_atend / i.inic_atend, 1)))
            lista_tmp.append(tmp)
            print(tmp)
        else:
```

```
        tmp=("Balcão {} não atendeu passageiros".format(i.obtem_n_balcao()))
        lista_tmp.append(tmp)
        print(tmp)
    return lista_tmp
#ponto 4.2
def simpar_simula(num_pass, num_bag, num_balcoes, ciclos, p_enche):
    """
    Corre uma simulação
    :param num_pass: o número de passageiros com bagagem previsto para este voo
    :param num_bag: o número máximo de bagagens permitido por passageiro
    :param num_balcoes: o número de balcões abertos para atendimento e despacho de bagagem
    :param ciclos: os ciclos de tempo em que a simulação decorre.
    :param p_enche:  % de passageiros a encher de arranque
    :return: None
    """
    global passTree
    atendidos = 0
    total = num_pass
    balcoes = []
    terco = ciclos / 3
    passTree = BinarySearchTree()
    for n_balcao in range(1, num_balcoes + 1):  # gera balcões
        balcoes.append(Balcao(n_balcao, num_bag))
    # passageiros iniciais
    enche = int((num_pass * p_enche) / 100)
    for i in range(0, enche):
        for j in balcoes:
            j.obtem_fila().enqueue(Passageiro(randint(1, num_bag), 0))  # aqui tempo é 0
            num_pass -= 1
# mostra_balcoes(balcoes)
    # Ocupar das filas
    for ciclo in range(0, ciclos):

        # Verifica se temos passageiros para criar
        if num_pass > 0:

            for n_balcao in range(1, num_balcoes + 1):  # aqui precorremos todos os balcões para colocar pessoas na
fila
                    # Calcula a probabilidade de acrescentar passageiro
                if ciclo <= terco:
                    probabilidade = 100
                elif ciclo <= terco * 2:
                    probabilidade = 80
                else:
                    probabilidade = 60

                temp = randint(0, 100)
                #print('Terço ' + str(terco)+ ' Probabilidade '+str(probabilidade) +' temp '+ str(temp)) #só para perceber
como está a funcionar a probabilidade
                if probabilidade >= temp:
                    # Obtem tamanho da fila com menos passageiros
                    fila_mais_curta = min([balcao.obtem_fila().size() for balcao in balcoes])

                    # Obtem apenas os balcões com o tamanha de fila mais curto
                    # (podem por exemplo existir vários balcões com 0 passageiros)
                    balcoes_filas_curtas = [balcao for balcao in balcoes if balcao.obtem_fila().size() == fila_mais_curta]

                    # E escolhemos de forma aleatória qual usamos
                    balcao_pretendido = choice(balcoes_filas_curtas)

                    # Cria passageiro
```

```
                balcao_pretendido.obtem_fila().enqueue(Passageiro(randint(1, num_bag), ciclo + 1))
                num_pass -= 1
                #print('criei um passageiro no b ' + str (balcao_pretendido)) #este print é de controle
        print("«««« CICLO n.º {} »»»».format(ciclo + 1))

        atendidos = atendidos + atende_passageiros(ciclo + 1, balcoes)
        p_nome=names.get_full_name()
        passTree.put(p_nome)
        mostra_balcoes(balcoes)
        if atendidos >= total:
            break
    print('ATENDIDOS ' + str(atendidos) + ' total ' + str(total))
    # Esvaziar das filas
    print('********************** Fechou a chegada de novos passageiros **********************')
    conta = 0
    esvazia_ciclo = 1
    esvazia = True
    while esvazia == True:
        for balcao in balcoes:  # vamos aos balcões ver se há filas de espera
            if balcao.obtem_fila().isEmpty() == False:  # se a fila não estiver vazia
                conta += 1  # conta é incrementado
        if conta == 0:  # Se não há filas cheias, sai
            esvazia = False
        else:
            esvazia_ciclo += 1
            print("«««« CICLO ESVAZIA n.º {} »»»».format(ciclo + esvazia_ciclo))
            atendidos = atendidos + atende_passageiros(ciclo, balcoes)
            p_nome=names.get_full_name()
            passTree.put(p_nome)
            #atende_passageiros(ciclo, balcoes)
            conta = 0  # Volta a zero para controlar o próximo ciclo

    #apresenta resultados balcoes e guarda no fich SimOutput
    with open("SimOutput",'wb') as f:
            pickle.dump(apresenta_resultados(balcoes),f)
            f.close()


def fazPesquisa():
    global passTree
    if passTree == None:
        print ('Não tem passageiros para pesquisar')
    else:
        #aqui vai pedir o intput da pesquisa
        pesquisa = input(str('Qual o nome do passageiro a pesquisar? \n'))

        if passTree.__contains__(pesquisa) == True:
            print('Pesquisa - {}:'.format(pesquisa) + ' passageiro embarcado')
        else:
            print('Pesquisa - {}:'.format(pesquisa) + ' passageiro não embarcado')

def fazListagem():
    global passTree
    if passTree == None:
        print ('Não tem passageiros para listar')
    else:
        passTree.root.inorder()


if __name__ == "__main__":
```

```
passa = 70
bag = 4
balc = 4
cicl = 10
pench = randint(0,100)
passTree = None
invalid = False  # Inicialização da variável de verificação de erro na Escolha
while True:
    menu()  # Chamada do Menu
    if invalid:  # Verificação se o utilizador escolheu uma opção incorrecta
        print('A opção não é válida')
        invalid = False  # Limpar a variável
    try:
        print("\n")
        escolha = int(input("Escolha uma opção: "))
    except ValueError:  # Se o Valor não for um inteiro estamos em estado de erro e tentamos novamente
        invalid = True
        continue  # Volta ao início do ciclo While

    if escolha == 1:
        limpa()
        limpa()
        # while True:
        if invalid:  # Verificação se o utilizador escolheu uma opção incorrecta
            print('A opção não é válida')
            invalid = False
        try:
            a = passa
            aux = int(input("O valor default é " + str(a) + ", indique o novo valor: "))  # display do valor antigo
            print("O valor passou de: " + str(a))
            print("Para: " + str(aux))
            if aux != a:  # se diferente substitui
                passa = aux
            input()
        except ValueError:  # Se o Valor não for um inteiro estamos em estado de erro e tentamos novamente
            invalid = True
            continue  # Volta ao início do ciclo While

    elif escolha == 2:
        limpa()
        limpa()
        # while True:
        if invalid:  # Verificação se o utilizador escolheu uma opção incorrecta
            print('A opção não é válida')
            invalid = False
        try:
            a = bag
            aux = int(
                input("O valor default é " + str(a) + ", indique o novo valor: "))  # display do valor antigo
            print("O valor passou de: " + str(a))
            print("Para: " + str(aux))
            if aux != a:  # se diferente substitui
                bag = aux
            input()
        except ValueError:  # Se o Valor não for um inteiro estamos em estado de erro e tentamos novamente
            invalid = True
            continue  # Volta ao início do ciclo While

    elif escolha == 3:
        limpa()
        limpa()
```

```
    # while True:
    if invalid:  # Verificação se o utilizador escolheu uma opção incorrecta
        print('A opção não é válida')
        invalid = False
    try:
        a = balc
        aux = int(
            input("O valor default é " + str(a) + ", indique o novo valor: "))  # display do valor antigo
        print("O valor passou de: " + str(a))
        print("Para: " + str(aux))
        if aux != a:  # se diferente substitui
            balc = aux
        input()
    except ValueError:  # Se o Valor não for um inteiro estamos em estado de erro e tentamos novamente
        invalid = True
        continue  # Volta ao início do ciclo While

elif escolha == 4:
    limpa()
    limpa()
    # while True:
    if invalid:  # Verificação se o utilizador escolheu uma opção incorrecta
        print('A opção não é válida')
        invalid = False
    try:
        a = cicl
        aux = int(
            input("O valor default é " + str(a) + ", indique o novo valor: "))  # display do valor antigo
        print("O valor passou de: " + str(a))
        print("Para: " + str(aux))
        if aux != a:  # se diferente substitui
            cicl = aux
        input()
    except ValueError:  # Se o Valor não for um inteiro estamos em estado de erro e tentamos novamente
        invalid = True
        continue  # Volta ao início do ciclo While

elif escolha == 5:
    limpa()
    limpa()
    # while True:
    if invalid:  # Verificação se o utilizador escolheu uma opção incorrecta
        print('A opção não é válida')
        invalid = False
    try:
        a = pench
        aux = int(
            input("O valor default é " + str(a) + ", indique o novo valor: "))  # display do valor antigo
        print("O valor passou de: " + str(a))
        print("Para: " + str(aux))
        if aux != a:  # se diferente substitui
            pench = aux
        input()
    except ValueError:  # Se o Valor não for um inteiro estamos em estado de erro e tentamos novamente
        invalid = True
        continue  # Volta ao início do ciclo While

elif escolha == 7:
    simpar_simula(passa, bag, balc, cicl, pench)
elif escolha == 8:
    fazListagem()
```

```
elif escolha == 9:
   fazPesquisa()

elif escolha == 99:
   limpa()
   limpa()
   print("...adeus :( ")
   break  # Finalizar o programa

else:
   invalid = True
   continue  # Volta ao início do ciclo While
input('Prima ←ENTER→ para continuar . . .')
```