CÓDIGO PYTHON

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 24 19:56:51 2021

@author: Mariel, Diana
"""

import math
import time
from matplotlib import pyplot



#%% Clase Titulo



# Clase Título
class Titulo:
    # Constructor con valores esperados:
    def __init__(self, id, tipo, titulo, directorx, reparto, pais, fechaN, fechaE, calificacion, duracion, categoria, descripcion):
        self.id = id
        self.tipo = tipo
        self.titulo = titulo
        self.directorx = directorx
        self.reparto = reparto
        self.pais = pais
        self.fechaN = fechaN #Fecha en que se agregó a Netflix
        self.fechaE = fechaE #Fecha de estreno
        self.calificacion = calificacion
        self.duracion = duracion
        self.categoria = categoria
        self.descripcion = descripcion
```

```
    # Getters necesarios
    def getId(self):
        return self.id
    def getTipo(self):
        return self.tipo
    def getTitulo(self):
        return self.titulo
    def getDirectrx(self):
        return self.directorx
    def getReparto(self):
        return self.reparto
    def getPais(self):
        return self.pais
    def getFN(self):
        return self.fechaN
    def getFE(self):
        return self.fechaE
    def getCali(self):
        return self.calificacion
    def getDuracion(self):
        return self.duracion
    def getCategoria(self):
        return self.categoria
    def getDescripcion(self):
        return self.descripcion

    #Get Key:
    #El valor clave asociado a los objetos título va a ser su id
    def getKey(self):
        return self.id

    def __str__(self):
        return "ID: " + str(self.id) + "Tipo: " + str(self.tipo) + "Titulo:
" + str(self.titulo) + "Directorx: " + str(self.directorx) + "Reparto: " +
```

```
str(self.reparto) + "Pais: " + str(self.pais) + "Fecha Netflix: " +
str(self.fechaN) + "Fecha Estreno: " + str(self.fechaE) + "Calificacion: "
+ str(self.calificacion) + "Duracion: " + str(self.duracion) + "Categoria:
" + str(self.categoria) + "Descripcion: " + str(self.descripcion)


    # Método equivalente a toString() para cuando la instancia se
    # encuentra en alguna colección.
    # El 'return' muestra solamente el 'id' del título. Esta variante
    # facilita la lectura del orden de los datos en una colección, pues
    # si se imprime una lista con muchos datos de películas, convendría
    # no tener todos los atributos, sino solo aquellos que nos sirvan para
    # su análisis.
    def __repr__(self):
        #return str(self.id)+" ("+str(self.aho)+") "+" : "+self.nombre
        return str(self.id)


    def __lt__(self, otro):
        return self.id < otro.id



# Lectura de archivos y creación de lista con elementos de clase 'Título'
def lee(archivo):
    id = open(archivo, "r")
    contenido = id.read()
    lista=contenido.split("\n") #cada enter es un nuevo elemento de la
lista
    id.close()
    return lista



# Regresa una lista con los elementos en su interior.
def listaTitulos(nLin):
    lista = []


    i = 0
```

```
    lee =
open('/Users/marielsgtzz/Desktop/ITAM/Otoño2021/EDA/titulosNetflix.txt')
    while i < nLin:
        linea = lee.readline()
        # Se ocupa el método {split()} para separar los valores que se
        # encuentran por línea en el documento de texto.  Regresa una lista
con estos valores.
        valores = linea.split('/')
        titulo = Titulo(int(valores[0]),valores[1], valores[2], valores[3],
valores[4], valores[5], valores[6], valores[7], valores[8], valores[9],
valores[10], valores[11] )
        lista.append(titulo)
        i += 1
    lee.close()
    print (lista)
    return lista




#%% Clase Hash Node


"""
    HASH NODE y HASH TABLE ----------------------------------------------
---
"""


# Clase HashNode
class HashNode:


    # Constructor:
    #   elem - cualquier tipo de dato,
    #          en este caso se almacenan títulos.
    #   next - Hash Node
    def __init__(self, elem):
        self.elem = elem
```

```
        self.next = None


    # Getters necesarios
    def getElem(self):  # Elemento
        return self.elem
    def getNext(self):  # Nodo
        return self.next


    # Setters necesarios
    def setElem(self,e):    # Elemento
        self.elem = e
    def setNext(self,n):    # Nodo
        self.next = n


    # GetKey:
    # Se extrae el valor llave propio del elemento guardado en el nodo.
    def getKey(self):
        return self.elem.getKey()




#%% Clase HashTable


# Clase HashTable
class HashTable:
    import math
    # Constructor
    def __init__(self, tamano):
        # Se crea una lista del tamaño dado como parámetro y en cada
casilla
        # se guarda a un Nodo Hash que contiene un 'None' como elemento.
```

```
        #   Estos nodos son los 'centinelas' de las listas que se guardarán
        #   en cada casilla de la tabla para su veloz acceso.
        self.tabla = [HashNode(None) for i in range(tamano)]
        self.cont = 0


# Getter de la tabla
def getTabla(self):
    return self.tabla


# Inserción de elemento en la tabla utilizando la función de hash
# por el método de la división.
def inserta(self, elem, key):
    nuevo = HashNode(elem)
    pos = self.funcHashDiv(key) % len(self.tabla)
    aux = self.tabla[pos].getNext()
    self.tabla[pos].setNext(nuevo)
    nuevo.setNext(aux)
    self.cont += 1

# Inserción de elemento en la tabla utilizando la función de hash
# por el método de la multiplicación.
def insertaM(self, elem, key):
    nuevo = HashNode(elem)

    pos = self.funcHashMult(key) % len(self.tabla)
    aux = self.tabla[pos].getNext()
    self.tabla[pos].setNext(nuevo)
    nuevo.setNext(aux)
    self.cont += 1


# Método de búsqueda de un elemento en la tabla de hash.
```

```
def busca(self, elem, key):
    # Se crea un nodo auxiliar con el elemento que se desea buscar
    # para poder extraer el valor de hash y encontrar la posición donde
    # debería de encontrarse el elemento.
    aux = HashNode(elem)
    # Bandera de fin de datos inicializada en False
    var = False
    pos = self.funcHashDiv(key) % len(self.tabla)
    actual = self.tabla[pos].getNext()
    while (not var and actual != None):
        if actual.getElem() == elem:
            var = True
        actual = actual.getNext()
    return var

def buscaM(self, elem, key):
    # Se crea un nodo auxiliar con el elemento que se desea buscar
    # para poder extraer el valor de hash y encontrar la posición donde
    # debería de encontrarse el elemento.
    aux = HashNode(elem)
    # Bandera de fin de datos inicializada en False
    var = False
    pos = self.funcHashDiv(key) % len(self.tabla)
    actual = self.tabla[pos].getNext()
    while (not var and actual != None):
        if actual.getElem() == elem:
            var = True
        actual = actual.getNext()
    return var


# Método de borrado de un elemento en la tabla de hash.
def borra(self, elem, key):
    # Se crea un nodo auxiliar con el elemento que se desea buscar
```

```
    # para poder extraer el valor de hash y encontrar la posición donde
    # debería de encontrarse el elemento.
    aux = HashNode(elem)
    pos = self.funcHashDiv(key) % len(self.tabla)
    prev = self.tabla[pos]
    actual = prev.getNext()
    while (actual != None and not actual.getElem() == elem):
        prev = actual
        actual = actual.getNext()
    if actual != None and actual.getElem() == elem:
        aux = actual.getNext()
        prev.setNext(aux)
        self.cont -= 1

def borraM(self, elem, key):
  # Se crea un nodo auxiliar con el elemento que se desea buscar
  # para poder extraer el valor de hash y encontrar la posición donde
  # debería de encontrarse el elemento.
  aux = HashNode(elem)
  pos = self.funcHashMult(key) % len(self.tabla)
  prev = self.tabla[pos]
  actual = prev.getNext()
  while (actual != None and not actual.getElem() == elem):
      prev = actual
      actual = actual.getNext()
  if actual != None and actual.getElem() == elem:
      aux = actual.getNext()
      prev.setNext(aux)
      self.cont -= 1


# Función de Hash por división
def funcHashDiv(self,key):
    res = key % len(self.tabla)
```

```
    return res

# Función de Hash por multiplicación
def funcHashMult(self,key):
    a = 1/((1+2.236)/2)
    res = math.floor(len(self.tabla)*(key*a-math.floor(key*a)))
    return res


# Método que regresa el número de datos contenidos en
# cada una de las casillas de la tabla
def numDatosXCasilla(self):
    res = []
    for i in range(0,len(self.tabla)):
        cont = 0
        actual = self.tabla[i].getNext()
        while actual != None:
            cont += 1
            actual = actual.getNext()
        res.append(cont)
    return res


# Método que devuelve el promedio de la cantidad de datos
# de las casillas de la tabla.
#   Se llama al método {prom}, especificado más abajo.

def promDatosXCasilla(self):
    import statistics as st
    return st.mean(self.numDatosXCasilla())


# Método que devuelve el número de lugares vacíos que se encuentran
# en tabla.
```

```
    def numCasillasVacias(self):
        cont = 0
        for i in range(0,len(self.tabla)):
            actual = self.tabla[i].getNext()
            if actual == None:
                cont +=1
        return cont


    # Método que imprime la tabla de hash.
    def toStr(self):
        for i in range(0,len(self.tabla)):
            actual = self.tabla[i].getNext()
            print(i)
            while actual != None:
                print(actual.getElem())
                actual = actual.getNext()









#%% Métodos obtención de datos


"""
    MÉTODOS DE PRUEBA ----------------------------------------------------
---
"""


    titulos = listaTitulos(n)
    h = HashTable(m)
    ins = []
```

```
for i in range(0,n):
    promIns = []
    for j in range(0,30):
        t_inicio = time.time_ns()
        h.inserta(titulos[i])
        t_fin = time.time_ns()
        t = t_fin - t_inicio
        promIns.append(t)
        h.borra(titulos[i])
    p = prom(promIns)
    if p > tope:
        tiempo = tope
    else:
        tiempo = p
    ins.append(tiempo)
    h.inserta(titulos[i])

bus = []
for i in range(0,n):
    promBus = []
    for j in range(0,30):
        t_inicio = time.time_ns()
        h.busca(titulos[i])
        t_fin = time.time_ns()
        t = t_fin - t_inicio
        promBus.append(t)
    p = prom(promBus)
    if p > tope:
        tiempo = tope
    else:
        tiempo = p
    bus.append(tiempo)

borr = []
```

```
    # NOTA IMPORTANTE: Dado que los elementos están "ordenados" o
"acomodados"
    # en la tabla de hash debido al orden de inserción, solamente podemos
borrar
    # una vez al elemento en la tabla para considerar el tiempo del proceso
    # Por lo tanto, no se tendrá una lista con promedios particulares
    for i in range(0,n):
        t_inicio = time.time_ns()
        h.borra(titulos[i])
        t_fin = time.time_ns()
        t = t_fin - t_inicio
        p = t
        if p > tope:
            tiempo = tope
        else:
            tiempo = p
        borr.append(tiempo)
        h.borra(titulos[i])



    print("")
    print("--------------------------------")
    print("Promedio de inserción en tiempo:")
    print(prom(ins))
    print("")
    print("--------------------------------")
    print("Promedio de búsqueda en tiempo:")
    print(prom(bus))
    print("")
    print("--------------------------------")
    print("Promedio de borrado en tiempo:")
    print(prom(borr))
    print("")


    return ins, bus, borr
```

```
# Método de prueba relacionado al análisis de la cantidad de datos
almacenados
# en cada casilla de la tabla, el número de ellas que están vacías y el
# promedio de la cantidad de datos que hay en las listas.
#
def pruebasDatos(n,m, listaTitulos):

    h = HashTable(m)

    for i in range(0,n):
        h.inserta(listaTitulos[i], listaTitulos[i].getKey())

    datosXCasilla = h.numDatosXCasilla()
    numVacias = h.numCasillasVacias()
    promDatos = h.promDatosXCasilla()

    print("")
    print("--------------------------------")
    print("Número de casillas vacías:")
    print(numVacias)
    print("")
    print("--------------------------------")
    print("Promedio de datos por casilla:")
    print(promDatos)

    return datosXCasilla




    titulos = listaTitulos(n)
    h1 = HashTable(m) # H1 para el método de la división
    h2 = HashTable(m) # H2 para el método de la multiplicación.
```

```
insDiv = []
for i in range(0,n):
    t_inicio = time.time_ns()
    h1.inserta(titulos[i])
    t_fin = time.time_ns()
    t = t_fin - t_inicio
    p = t
    if p > tope:
        tiempo = tope
    else:
        tiempo = p
    insDiv.append(tiempo)


insMult = []
for i in range(0,n):
    t_inicio = time.time_ns()
    h2.insertaM(titulos[i])
    t_fin = time.time_ns()
    t = t_fin - t_inicio
    p = t
    if p > tope:
        tiempo = tope
    else:
        tiempo = p
    insMult.append(tiempo)

datosDiv = h1.numDatosXCasilla()
datosMult = h2.numDatosXCasilla()

print("")
print("--------------------------------")
print("Número vacías por (División)")
```

```
    numVaciasDiv = h1.numCasillasVacias()
    print(numVaciasDiv)
    print("")
    print("--------------------------------")
    print("Número vacías por (Multiplicación)")
    numVaciasMult = h2.numCasillasVacias()
    print(numVaciasMult)

    print("")
    print("--------------------------------")
    print("Promedio datos por casilla (División)")
    promDatosDiv = h1.promDatosXCasilla()
    print(promDatosDiv)
    print("")
    print("--------------------------------")
    print("PromDatosXCasilla (Multiplicación)")
    promDatosMult = h2.promDatosXCasilla()
    print(promDatosMult)

    print("")
    print("--------------------------------")
    print("Promedio de inserción función división en tiempo:")
    print(prom(insDiv))
    print("")
    print("--------------------------------")
    print("Promedio de búsqueda función multiplicación en tiempo:")
    print(prom(insMult))

    return insDiv, insMult, datosDiv, datosMult

def pruebaComparacion2(n, m, tope, listaTitulos):
    import time
    promIns = []
    promBus = []
```

```
promBorr = []
t1 =  HashTable(m)
for i in range(0,20):
    tiempoInicio = time.time()
    for k in range(0,len(listaTitulos)):
        t1.insertaM(listaTitulos[k], listaTitulos[k].getKey())
    tiempoFin = time.time()
    tiempo = tiempoFin - tiempoInicio
    promIns.append(tiempo)

    tiempoInicio = time.time()
    for k in range(0,len(listaTitulos)):
        t1.buscaM(listaTitulos[k], listaTitulos[k].getKey())
    tiempoFin = time.time()
    tiempo = tiempoFin - tiempoInicio
    promBus.append(tiempo)

    tiempoInicio = time.time()
    for k in range(0,len(listaTitulos)):
        t1.borraM(listaTitulos[k], listaTitulos[k].getKey())
    tiempoFin = time.time()
    tiempo = tiempoFin - tiempoInicio
    promBorr.append(tiempo)
import statistics as st
print("Tiempo insertar multiplicacion ",st.mean(promIns))
print("Tiempo buscar multiplicacion ",st.mean(promBus))
print("Tiempo borrar multiplicacion ",st.mean(promBorr))

promIns = []
promBus = []
promBorr = []
t2 =  HashTable(m)
for i in range(0,20):
    tiempoInicio = time.time()
```

```
        for k in range(0,len(listaTitulos)):
            t2.inserta(listaTitulos[k], listaTitulos[k].getKey())
        tiempoFin = time.time()
        tiempo = tiempoFin - tiempoInicio
        promIns.append(tiempo)

        tiempoInicio = time.time()
        for k in range(0,len(listaTitulos)):
            t2.busca(listaTitulos[k], listaTitulos[k].getKey())
        tiempoFin = time.time()
        tiempo = tiempoFin - tiempoInicio
        promBus.append(tiempo)

        tiempoInicio = time.time()
        for k in range(0,len(listaTitulos)):
            t2.borra(listaTitulos[k], listaTitulos[k].getKey())
        tiempoFin = time.time()
        tiempo = tiempoFin - tiempoInicio
        promBorr.append(tiempo)

    print("Tiempo insertar division ",st.mean(promIns))
    print("Tiempo buscar division ",st.mean(promBus))
    print("Tiempo borrar division ",st.mean(promBorr))



#%% Llamados generales

"""
    LLAMADAS A MÉTODOS FINALES ----------------------------------------------
---
"""

# Llamada a creación y obtención de gráficas para casos
#graficasCasos(pruebaTamanioFijo,casoGeneral)
```

```
# Llamada a creación y obtención de gráficas sobre funcionamiento
signo = "<"
cantidad = 0
#graficasFuncionamiento(signo, cantidad)


# llamada a creación y obtención de gráficas para comparar desempeño de las
# funciones de hash de división y de multiplicación
graficasComparacion(100,100,8000)

t = listaTitulos(8000)
import math
pruebaComparacion2(100,100,8000, t)
pruebasDatos(100,100,t)
```