	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Quintana Martínez Marco Antonio
Asignatura:	Estructura de Datos y Algoritmos I
Grupo:	17
No de Práctica(s):	11-Estrategias para la construcción de algoritmos.
Integrante(s):	Galdamez Pozos Yoav Farid
Semestre:	2020-2
Fecha de entrega:	27/04/2020

CALIFICACIÓN: _____

Objetivo:

El objetivo de esta guía es implementar, al menos, dos enfoques de diseño (estrategias) de algoritmos y analizar las implicaciones de cada uno de ellos.

Introducción:

Un algoritmo es un conjunto de instrucciones donde llevas un orden para llegar a la solución del problema, son independientes de los lenguajes de programación, en cada problema el algoritmo puede escribirse y luego ejecutarse en un lenguaje.

Desarrollo:

```python
from string import ascii_letters, digits
from itertools import product

#Concatenar letras y digitos en una sola cadena

caracteres = ascii_letters + digits

def buscador(con):

    #Archivo con todas las combinaciones generadas
    archivo = open("Combinaciones.txt", "w")

    if 3<= len(con) <= 4:
        for i in range(3,5):
            for comb in product(caracteres, repeat = i):
                #Se utiliza join() para concatenar los caracteres regresados por la funcion produtc().
                #Como join() necesita una cadena inicial para iniciar la concatenacion, se usa una cadena vacía
                #al principio.
                prueba = "".join(comb)
                #Escribiendo al archivo cada combinacion generada
                archivo.write( prueba + "\n" )
                if prueba == con:
                    print('Tu contraseña es: {}'.format(prueba))
                    #Cerrando el archivo
                    archivo.close()
                    break
    else:
        print("Ingresa una contraseña que contenga de 3 a 4 caracteres")

from time import time

t0 = time()
con = 'H0l4'
buscador(con)
print("Tiempo de ejecucion {}".format(round(time() - t0, 6)))
```

```
C:\Users\Invitado2\PycharmProjects\untitled1\venv\Scripts\python.exe
  C:/Users/Invitado2/PycharmProjects/untitled1/Practica11.py
 Tu contraseña es: H0lu
 Tiempo de ejecucion 19.979143
```

```python
def cambio(cantidad, denominaciones):
    resultado = []
    while (cantidad > 0):
        if(cantidad >= denominaciones[0]):

            num = cantidad // denominaciones[0]
            cantidad = cantidad - (num * denominaciones[0])
            resultado.append([denominaciones[0], num])
        denominaciones = denominaciones[1:] #Se va consumiendo la lista de denominaciones
    return resultado
#pruebas del algoritmo

print(cambio(1000, [500, 200, 100, 50, 20, 5, 1]))
print(cambio(500, [500, 200, 100, 50, 20, 5, 1]))
print(cambio(300, [50, 20, 5, 1]))
print(cambio(200, [5]))
print(cambio(98, [50, 20, 5, 1]))


#Aqui no regresa la solucion optima porque no esta de mayor a menor
print(cambio(98, [5, 20, 1, 50]))
```

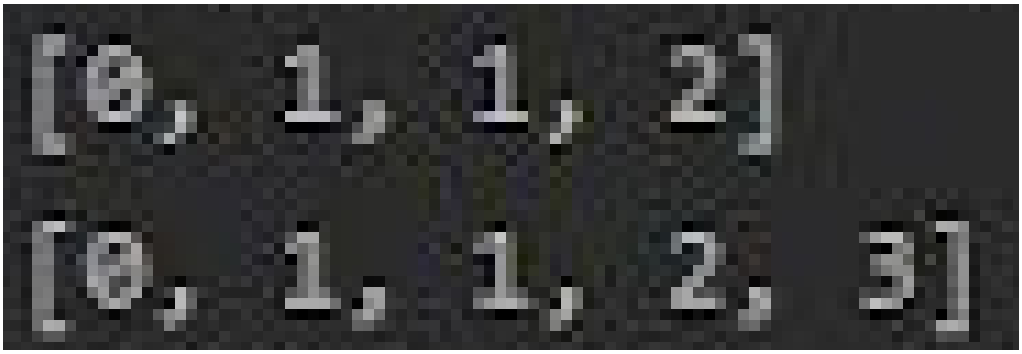
```
[[500, 2]]
[[500, 1]]
[[50, 6]]
[[5, 40]]
[[50, 1], [20, 2], [5, 1], [1, 3]]
[[5, 19], [1, 3]]
```

```python
def fibonacci_iterativo_v1(numero):
    f1=0
    f2=1
    tmp=0
    for i in range(1, numero-1):
        tmp = f1 + f2
        f1=f2
        f2=tmp
    return f2
fibonacci_iterativo_v1(6)

def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range(1, numero-1):
        f1, f2=f2, f1+f2#Asignacion paralela
    return f2
fibonacci_iterativo_v2(6)

def fibonacci_bottom_up(numero):
    f_parciales = [0, 1, 1]#Esta es la lista que mantiene las soluciones previamente calculadas
    while len(f_parciales) < numero:
        f_parciales.append(f_parciales[-1] + f_parciales[-2])
        print(f_parciales)
    return f_parciales[numero-1]

fibonacci_bottom_up(5)
```




```python
def fibonacci_top_down(numero):
    if numero in memoria: #Si el numero eta calculado ya no se hacen operaciones
        return memoria[numero]
    f = fibonacci_iterativo_v2(numero-1) + fibonacci_iterativo_v2(numero-2)
    memoria[numero] = f
    return memoria[numero]
fibonacci_top_down(12)
```

```python
import pickle

#Guardamos la variable con una extension
archivo = open("memoria.p", "rb")
memoria_de_archivo = pickle.load(archivo)
archivo.close()
```

```python
def insertionSort(n_lista):
    for index in range(1, len(n_lista)):
        actual = n_lista[index]
        posicion = index
        print("Valor a ordenar = {}".format(actual))
        while posicion>0 and n_lista[posicion-1]>actual:
            n_lista[posicion]=n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion] = actual
        print(n_lista)
        print()
    return n_lista
#Datos de entrada
lista= [21, 10, 0, 11, 9, 24, 20, 14, 1]
print("Lista desordenada {}".format(lista))
insertionSort(lista)
print("Lista desorndenada {}".format(lista))
```

```
Lista desordenada [21, 10, 0, 11, 9, 24, 20, 14, 1]
Valor a ordenar = 10
[10, 21, 0, 11, 9, 24, 20, 14, 1]

Valor a ordenar = 0
[0, 10, 21, 11, 9, 24, 20, 14, 1]

Valor a ordenar = 11
[0, 10, 11, 21, 9, 24, 20, 14, 1]

Valor a ordenar = 9
[0, 9, 10, 11, 21, 24, 20, 14, 1]

Valor a ordenar = 24
[0, 9, 10, 11, 21, 24, 20, 14, 1]

Valor a ordenar = 20
[0, 9, 10, 11, 20, 21, 24, 14, 1]

Valor a ordenar = 14
[0, 9, 10, 11, 14, 20, 21, 24, 1]

Valor a ordenar = 1
[0, 1, 9, 10, 11, 14, 20, 21, 24]

Lista desorndenada [0, 1, 9, 10, 11, 14, 20, 21, 24]
```

```python
def quicksort(lista):
    quicksort_aux(lista, 0, len(lista)-1)

def quicksort_aux(lista, inicio, fin):
    if inicio < fin:

        pivote = particion(lista, inicio, fin)

        quicksort_aux(lista, inicio, pivote-1)
        quicksort_aux(lista, pivote+1, fin)

def particion(lista, inicio, fin):
    #se asigna como pivote en número de la primera localidad
    pivote = lista[inicio]
    print("Valor del pivote {}".format(pivote))
    #Se crean dos marcadores
    izquierda = inicio+1
    derecha = fin
    print("Índice izquierdo {}".format(izquierda))
    print("Índice derecho {}".format(derecha))

    bandera = False
    while not bandera:
        while izquierda <= derecha and lista[izquierda] <= pivote:
            izquierda = izquierda + 1
        while lista[derecha] >= pivote and derecha >= izquierda:
            derecha = derecha -1
        if derecha < izquierda:
            bandera = True
        else:
            temp=lista[izquierda]
            lista[izquierda]=lista[derecha]
            lista[derecha]=temp
    print(lista)

    temp=lista[inicio]
    lista[inicio]=lista[derecha]
    lista[derecha]=temp
    return derecha
```

```
Lista desordenada [21, 10, 0, 11, 9, 24, 20, 14, 1]
Valor del pivote 21
Índice izquierdo 1
Índice derecho 8
[21, 10, 0, 11, 9, 1, 20, 14, 24]
Valor del pivote 14
Índice izquierdo 1
Índice derecho 6
[14, 10, 0, 11, 9, 1, 20, 21, 24]
Valor del pivote 1
Índice izquierdo 1
Índice derecho 4
[1, 0, 10, 11, 9, 14, 20, 21, 24]
Valor del pivote 10
Índice izquierdo 3
Índice derecho 4
[0, 1, 10, 9, 11, 14, 20, 21, 24]
Lista ordenada [0, 1, 9, 10, 11, 14, 20, 21, 24]
```

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#Cargando modulos
import random
from time import time

#cargando las funciones guardadas en los archivos
from insertionSort import insertionSort_time
from quickSort import quicksort_time

datos = [ii*100 for ii in range(1, 21)]

tiempo_is = []
tiempo_qs = []

for ii in datos:
    lista_is = random.sample(range(0, 10000000), ii)
    #Se hace una copia d ela lista para que se ejecute el algoritmo
    lista_qs = lista_is.copy()

    t0=time() #"Se guarda el tiempo inicial"
    insertionSort_time(lista_is)
    tiempo_is.append(round(time()-t0, 6)) #se le resta al tiempo inicial el tiempo actual

    t0 = time()
    "Se guarda el tiempo inicial"
    quicksort_time(lista_qs)
    tiempo_qs.append(round(time() - t0, 6))  # se le resta al tiempo inicial el tiempo actual

print("Tiempos parciales de ejecucion en INSERT SORT {} [s] \n".format(tiempo_is))
print("Tiempos parciales de ejecucion en QUICK SORT {} [s] \n".format(tiempo_qs))

print("Tiempo total de ejecucion en insert sor {} [s]".format(sum(tiempo_is)))
print("Tiempo total de ejecucion de quick sort {} [s]".format(sum(tiempo_qs)))
```

```python
#Generando la grafica
fig, ax = subplots()
ax.plot(datos, tiempo_is, label="insert sort", marker="*", color="r")
ax.plot(datos, tiempo_qs, label="quick sort", marker="o", color="b")
ax.set_xlabel('Datos')
ax.set_ylabel('Tiempo')
ax.grid(True)
ax.legend(loc=2);

plt.title('Tiempo de ejecucion [s] (insert vs quick)')
plt.show()
```

```
C:\Users\Invitado2\PycharmProjects\untitled1\venv\Scripts\python.exe
  C:/Users/Invitado2/PycharmProjects/untitled1/hashdohad.py
Traceback (most recent call last):
  File "C:/Users/Invitado2/PycharmProjects/untitled1/hashdohad.py", line 9, in <module>
    from insertionSort import insertionSort_time
ModuleNotFoundError: No module named 'insertionSort'

Process finished with exit code 1
```

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

times = 0
def insertionSort_graph(n_lista):
    global times
    for index in range(1, len(n_lista)):
        times += 1
        actual = n_lista[index]
        posicion = index
        while posicion>0 and n_lista[posicion-1]> actual:
            times += 1
            n_lista[posicion]= n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion]=actual
    return n_lista

TAM = 101 eje_x = list(range(1, TAM, 1))
eje_y = []
lista_variable = []

for num in eje_x:
    lista_variable = random.sample(range(0,1000), num)
    times = 0
    lista_variable = insertionSort_graph(lista_variable)
    eje_y.append(times)

fig, ax = plt.subplots(facecolor='w', edgecolor='k')
ax.plot(eje_x, eje_y, marker = 'o', color='b', linestyle='None')

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.grid(True)
ax.legend(["Insertion sort"])

plt.tilte('Insert sort')
plt.show()
```

```
 C:/Users/Invitado2/PycharmProjects/untitled1/hashdohad.py
Traceback (most recent call last):
  File "C:/Users/Invitado2/PycharmProjects/untitled1/hashdohad.py", line 9, in <module>
    from insertionSort import insertionSort_time
ModuleNotFoundError: No module named 'insertionSort'

Process finished with exit code 1
```

Conclusión:

Los algoritmos son la principal herramienta con la que se cuenta en ingeniería a la hora de resolver problemas y que se hayan desarrollado estrategias para afrontarlos de maneras más sencillas me resulta agradable porque solo están dando consejos para poder atacar de mejor manera algún problema, ya que ninguna situación es igual a otra y puede que a algunas personas les resulte mucho más fácil algún método que a otras.

Bibliografía:

Tutorial oficial de Python: https://docs.python.org/3/tutorial/

Galería de notebooks: https://wakari.io/gallery

Matplotlib: http://matplotlib.org/