	Carátula para entrega de prácticas	
Facultad de Ingeniería		Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Adrian Ulises Mercado

Asignatura: Estructura de datos y Algoritmos I

Grupo: 13

No de Práctica(s): Práctica 11

Integrante(s): Escobar Ventura Jesus Emiliano

No. de Equipo de cómputo empleado:

No. de Lista o Brigada: Brigada 5

Semestre: 2020-2

Fecha de entrega: **07.**06.2020

Observaciones:

CALIFICACIÓN: _____

INTRODUCCIÓN

Se van a desarrollar diversos programas en python en los cuales se van a utilizar algunos algoritmos como: fuerza bruta, greedy, Bottom-up, top-down, Incremental, divide y vencerás y se desarrollaran las gráficas de los tiempos de ejecución.

DESARROLLO

Fuerza bruta

El objetivo del programa es buscar una contraseña de 3 a 4 caracteres; Se utilizó el algoritmo de fuerza bruta porque va realizar una búsqueda exhaustiva de todas las posibilidades que puedan dar solución al problema. Cabe mencionar que la contraseña son de caracteres alfanuméricos y va a generar un archivo de texto con las posibles combinaciones generadas y el tiempo que le tomó encontrar la contraseña.

En este caso la contraseña que se ingresó due al3 y le tomó 57.82 segundos encontrar la contraseña.

```
# Estrategia de fuerza bruta,
# realiza una busqueda exhaustiva
from string import ascii_letters, digits
from itertools import product
from time import time

caracteres = ascii_letters + digits

def buscar(con):
    #Abrir el archivo con las cadenas generads
    archivo  = open("combinaciones.txt", "w")
    if 3 <= len(con) <= 4:
        for i in range(3, 5):
            for comb in product(caracteres, repeat = i):
                prueba = "".join(comb)
                archivo.write(prueba + "\n")
                if prueba == con:
                    print("La contraseña es {}" .format(prueba))
                    break
        archivo.close()

    else:
        print("Ingresa un contraseña de longitud 3 o 4")

if __name__ == "__main__":
    con = input("ingresa una contraseña\n")
    t0 = time()
    buscar(con)
    print("El tiempo de ejecucion {}" .format(round(time()- t0,6)))
```

Greedy

Para el algoritmo greedy, se va a realizar un problema en cual consiste en regresar monedas de alguna denominación, usando el menor número de estas.
Su solución consta de de dar las monedas de mayor denominación hasta que ya no haya, para utilizar la siguiente moneda de mayor denominación.
En este caso se ingresan casos como:

- cambio de 1000 con monedas de 20, 10, 5, 2 en el cual te da 50 monedas de 20
- cambio de 20 con monedas de 20, 10, 5,2 en el cual te dan 1 moneda de 20
- cambio de 30 con monedas de 20, 10, 5, 2 en el cual te dan una moneda de 10 y una de 20.
- Cambio de 98 con monedas de 50, 20, 5, 1 en el cual te dan una moneda de 50, 2 monedas de 20, 1 moneda de 5 y 3 monedas de 1

```
# Algoritmo greedy
def cambio(cantidad, monedas):
    resulatdo = []
    while cantidad > 0:
        if cantidad >= monedas[0]:
            num = cantidad//monedas[0]
            cantidad = cantidad - (num*monedas[0])
            resulatdo.append([monedas[0], num])
        monedas = monedas[1:]
    return resulatdo

if __name__== "__main__":
    print(cambio(1000,[20, 10, 5, 2 , 10]))
    print(cambio(20,[20, 10, 5, 2 , 10]))
    print(cambio(30,[20, 10, 5, 2 , 10]))
    print(cambio(98,[50, 20, 5,1]))
```

Bottom-up

Se utiliza el ejemplo de encontrar un número n de la sucesión de Fibonacci. Se utiliza este algoritmo porque guarda los que ya han sido solicitados y la solución de un número es forma a partir de la combinación de una o más soluciones que se hayan guardado.

```
#Algoritmo bottom up/ programacion dinamica
def fibonacci (numero):
    a = 1
    b = 1
    c = 0
    for i in range(1, numero-1):
        c = a + b
        a = b
        b = c
    return c
```

```
def fibonacci2 (numero):
    a = 1
    b = 1
    for i in range(1, numero-1):
        a, b = b, a+b
    return b

def fibonacci_bottom_up(numero):
    fib_parcial = [1, 1 , 2]
    while len(fib_parcial) < numero:
        fib_parcial.append(fib_parcial[-1] + fib_parcial[-2])
        print(fib_parcial)
    return fib_parcial[numero-1]

f = fibonacci_bottom_up(4)
print(f)

f2 = fibonacci2(5)
print(f2)
```

Top-down

Se vuelve a utilizar el problema de encontrar un número n de la sucesión de Fibonacci, solo que esta vez con otro método.

El algoritmo Top-down, empieza a hacer el calculo del numero n hacia abajo, por otro lado guarda los resultados que ya han sido calculados, para que no tenga que repetir las operaciones.

Para hacer más visual lo que pasa con el algoritmo top-down, imprimimos la memoria con su respectiva respuesta del numero a buscar, para cerciorarnos que los valores se están guardando en memoria.

En este caso se utilizaron los número 4, 6 y 8

```
# Estrategia top-down
# n = 5
# f(n-1) = f(4) + f(3) + f(2) + f(1)
# f(n-2) =  f(3) + f(2) + f(1)
memoria = {1:1, 2:1, 3:2}

def fibonacci(numero):
    a = 1
    b = 1
    for i in range(1 , numero -1):
```

```
        a , b= b , a+b
    return b

def fibonacci_top_down(numero):
    if numero in memoria:
        return memoria[numero]
    f = fibonacci(numero-1) + fibonacci(numero-2)
    memoria[numero] = f
    return memoria[numero]


print(fibonacci_top_down(4))
print(memoria)

print(fibonacci_top_down(6))
print(memoria)

print(fibonacci_top_down(8))
print(memoria)
```

Incremental

El problema que se realizó fue Insert sort, el cual ordena los elementos de una lista de manera ascendente, se utiliza este algoritmo porque se quiere cerciorar que cada movimiento sea correcto y por ello lo hace de manera paulatina. En donde se toma el primer elemento de la lista y se compara con los demás elementos para saber si es mayor o menor, la lista a ordenar fue : 21, 10, 12, 0, 34, 15, 6, 50

```
#Estrategia incremental
# Algoritmo de ordenacion por insercion
"""
21 10 12 0 34 15
parte ordenada
21   10 12 0 34 15
10 21            12 0 34 15
10 12 21         0 34 15
0 10 12 21       34 15
0 10 12 15 21    34
0 10 12 15 21 34
"""

def insertSort (lista):
    for index in range(1, len(lista)):
        actual = lista[index]
```

```
        posicion = index
        print("valor a ordenar {}".format(actual))
        while posicion > 0 and lista[posicion-1]>actual:
            lista[posicion] = lista[posicion-1]
            posicion = posicion-1
        lista[posicion] = actual
        print(lista)
        print()
    return lista

lista = [21, 10, 12, 0, 34, 15, 6, 50]
print(lista)
insertSort(lista)
print(lista)
```

Divide y vencerás

El problema que se realizó fue quick sort, el cual a ordenar una lista de elementos, por medio de la partición de la lista un valor pivote. Este algoritmo consiste en dividir el problemas en subproblemas que tengan las mismas características pero son mas faciles de resolver.

Una vez que se dividieron los datos, se mueven los que no están ordenados con respecto al pivote.

En este caso la lista a ordenar fue: 25, 5, 17, 21, 10, 12, 0, 34, 15

```
#Estrategia divide y vencerás
"""
21 10 12 0 34 15
p   i          d

i es es mayor que p
d es menor que p

15 10  12 0 34 21
p   i        d

15 10  12 0 34 21
p       i d

0 10 12 15 34 21
"""
def quickSort(lista):
    quickSort2(lista, 0, len(lista)-1)
```

```
def quickSort2 (lista, inicio, fin):
    if inicio < fin:
        pivote = particion(lista, inicio, fin)
        quickSort2(lista, inicio, pivote-1)
        quickSort2(lista, pivote+1, fin)

def particion(lista, inicio, fin):
    pivote = lista[inicio]
    print("Valor del pivote {}".format(pivote))
    izquierda = inicio + 1
    derecha = fin
    print("indice izquierdo {} y indice derecho {}" .format(izquierda,
derecha))

    bandera = False
    while not bandera:
        while izquierda <= derecha and lista[izquierda] <= pivote:
            izquierda = izquierda + 1
        while derecha >= izquierda and lista[derecha] >= pivote:
            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 = [25, 5, 17,21, 10, 12, 0, 34, 15]
print(lista)
quickSort(lista)
print(lista)
```

Medición y gráficas de los tiempo de ejecución

Insert sort vs quick sort
Se importan los ejercicios que tienen las funciones quick sort e insert sort para que las podamos graficar con la ayuda de la biblioteca maplotlib y time

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import random
from time import time
from ejercicio5 import insertSort
from ejercicio6 import quickSort

datos =[ii*100 for ii in range(1,21)]
tiempo_is = []
tiempo_qs = []
for ii in datos:
    lista_is = random.sample(range(0,1000000),ii)
    lista_qs = lista_is.copy()

    t0 = time()
    insertSort(lista_is)
    tiempo_is.append(round(time()-t0, 6))

    t0 = time()
    quickSort(lista_qs)
    tiempo_qs.append(round(time()-t0, 6))

print("Tiempos parciales de ejecucion en insertsort {} [s]"
.format(tiempo_is))

print("Tiempos parciales de ejecucion en quick sort{} [s]" .format(tiempo_qs))

fig, ax = plt.subplots()
ax.plot(datos, tiempo_is, label="insert sort", marker = "*", color = "r")
ax.plot(datos, tiempo_qs, label="quick sort", marker = "o", color = "g")


ax.set_xlabel("Datos")
ax.set_ylabel("Tiempo")
ax.grid(True)
ax.legend(loc = 2)
plt.title("Tiempo de ejecucion [s] inser sort vs quick sort")
plt.show()
```

Gráfica insert sort

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

times = 0

def insertSort(lista):
    global times
    for i in range(1, len(lista)):
        times +=1
        actual = lista[i]
        posicion = i
        while posicion > 0 and lista[posicion-1] > actual:
            times += 1
            lista[posicion] = lista[posicion-1]
            posicion = posicion-1
        lista[posicion] = actual
    return 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, 100), num)
    times = 0
    lista_variable = insertSort(lista_variable)
    eje_y.append(times)

# Grafica
fig, ax = plt.subplots(facecolor ='w', edgecolor= 'k')
ax.plot(eje_x, eje_y, marker="*", linestyle = "none", color="g")

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

plt.title("insert sort")
plt.show()
```

EJECUCIÓN

Fuerza bruta

```
SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN  PROBLEMAS                    1: Python
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python> & C:/Users/Emiliano/AppData/Local/Programs/Python/Python38-32/python.exe
 "c:/Users/Emiliano/Desktop/Estructura de datos/practica 11/ejercicio1.py"
ingresa una contraseña
al3
La contraseña es al3
El tiempo de ejecucion 57.824835
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python>
```

Greedy

```
SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN  PROBLEMAS                    1: Python
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python> & C:/Users/Emiliano/AppData/Local/Programs/Python/Python38-32/python.exe "c:/Users/Emiliano/Desk
top/Estructura de datos/practica 11/ejercicio2.py"
[[20, 50]]
[[20, 1]]
[[20, 1], [10, 1]]
[[50, 1], [20, 2], [5, 1], [1, 3]]
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python>
```

Bottom-up

```
Archivo  Editar  Selección  Ver  Ir  Ejecutar  Terminal  Ayuda   ejercicio3.py - recursividas con python - Visual Studio Code
SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN  PROBLEMAS                    1: Python

[1, 1, 2, 3]
[1, 1, 2, 3, 5]
[1, 1, 2, 3, 5, 8]
[1, 1, 2, 3, 5, 8, 13]
13
13
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python> & C:/Users/Emiliano/AppData/Local/Programs/Python/Python38-32/python.exe "c:/Users/Emiliano/Desk
top/Estructura de datos/practica 11/ejercicio3.py"
[1, 1, 2, 3]
[1, 1, 2, 3, 5]
[1, 1, 2, 3, 5, 8]
[1, 1, 2, 3, 5, 8, 13]
[1, 1, 2, 3, 5, 8, 13, 21]
[1, 1, 2, 3, 5, 8, 13, 21, 34]
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
55
55
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python>
```

Top-down

```
SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN  PROBLEMAS                    1: Python
--------------------------------------------------------------------------------
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python>  C:/Users/Emiliano/AppData/Local/Programs/Python/Python38-32/python.exe "c:/Users/Emiliano/Deskt
op/Estructura de datos/practica 11/ejercicio4.py"
3
{1: 1, 2: 1, 3: 2, 4: 3}
8
{1: 1, 2: 1, 3: 2, 4: 3, 6: 8}
21
{1: 1, 2: 1, 3: 2, 4: 3, 6: 8, 8: 21}
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python>
```

Incremental

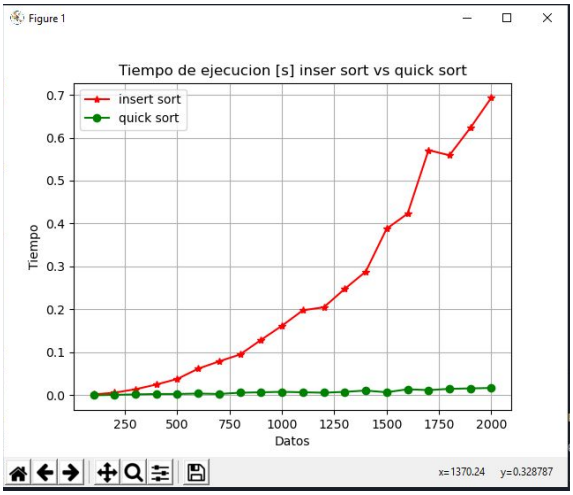
```
SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN  PROBLEMAS                    1: Python

valor a ordenar 15
[0, 10, 12, 15, 21, 34, 6, 50]

valor a ordenar 6
[0, 6, 10, 12, 15, 21, 34, 50]

valor a ordenar 50
[0, 6, 10, 12, 15, 21, 34, 50]

[0, 6, 10, 12, 15, 21, 34, 50]
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python>
```

Divide y vencerás

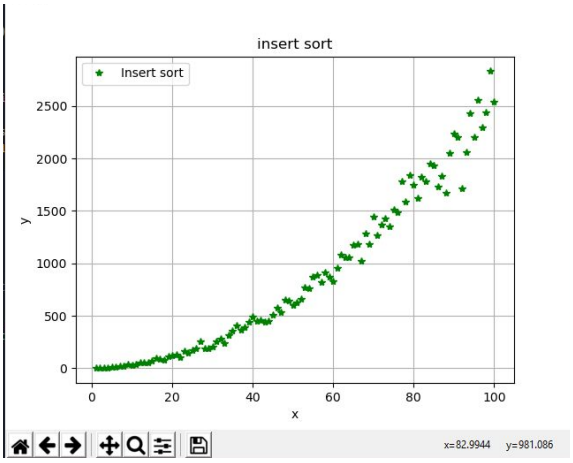
```
SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN  PROBLEMAS                    1: Python

indice izquierdo 1 y indice derecho 1
[0, 5, 10, 12, 15, 21, 17, 25, 34]
Valor del pivote 21
indice izquierdo 6 y indice derecho 6
[0, 5, 10, 12, 15, 21, 17, 25, 34]
[0, 5, 10, 12, 15, 17, 21, 25, 34]
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividas con python>
```

Gráficas de tiempo de ejecución

Gráficas insert sort vs quick sort



gráfica insert sort



CONCLUSIÓN

El problema insert sort es de orden n^2 mientras que quick sort es de orden lineal.

COMENTARIOS

BIBLIOGRAFÍAS