|  |  |  |
| --- | --- | --- |
|  | **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | | Laboratorio de docencia |

Laboratorios de computación

salas A y B

|  |  |
| --- | --- |
| *Profesor:* | Marco Antonio Martinez Quintana |
| *Asignatura:* | Estructura de Datos y Algoritmos I |
| *Grupo:* | 7 |
| *No de Práctica(s):* | 11 |
| *Integrante(s):* | Rodriguez Tellez Leonardo |
| *No. de Equipo de cómputo empleado:* | 47 |
| *No. de Lista o Brigada:* | 31 |
| *Semestre:* | 2020-2 |
| *Fecha de entrega:* | 21/04/2020 |
| *Observaciones:* |  |
| Calificación: |  |

**Cuéllar Uribe Fernando  
Grupo 17  
Estructura de Datos y Algoritmos I**

**Guía práctica de estudio 0:**

**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:**

¿Qué es un algoritmo? Simplemente una serie de instrucciones sencillas que se llevan a cabo para solventar un problema. La regla de multiplicar que aprendimos en el colegio y que permite sacar el producto de dos números de varias cifras, con papel y lápiz, es un sencillo algoritmo.

**Desarrollo:**

**from** **string** **import** ascii\_letters , digits

**from** **itertools** **import** product

*#Concatenar letas y dígitos 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 regresado por la función product().*

*#Como join necesita una cadena inicial para hacer la concatenación, se pone una cadena vacía*

*#al principio*

prueba = "".join(comb)

*#Escribiendo al archivo cada combinación 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("Tiempos de ejecución **{}**".format(round(time()-t0, 6)))

**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

**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

**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]

**def** fibonacci\_top\_down(numero):

**if** numero **in** memoria: *#Si el número ya se encuentra calculado, se regresa el valor ya ya no se hacen más cálculos*

**return** memoria[numero]

f = fibonacci\_iterativo\_v2(numero-1) + fibonacci\_iterativo\_v2(numero-2)

memoria[numero] = f

**return** memoria[numero]

**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

**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

**Conclusiones:**

Por último, a la fuerza bruta, es decir, se hace una amplia búsqueda de todas las opciones que hay para resolver un problema siendo esta la menos eficaz y por consiguiente más lenta

La estrategia llamada divide y vencerás, consiste en dividir tareas en mas tareas para que asi el proceso sea más eficaz

La estrategia incremental consiste en implementar y probar poco a poco constantemente creando cierto incremento.

En top-down los cálculos, como su nombre lo dice, se hacen de arriba para abajo, en cierto problema. Por ejemplo en una serie de números como en la sucesión de Fibonacci.

Bottom-up como su nombre lo dice, hace cálculos de abajo hacia arriba resolver un problema a partir de subproblemas que ya han sido resueltos. La solución final se forma a partir de la combinación de las soluciones antes obtenidas

La estrategia de algoritmos ávidos (greedy) se basa en decisiones en orden específico, pero una vez que se ejecuta una decisión, esta ya no se vuelve a ejecutar.

**Bibliografía:**El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.  
  
Laboratorio de Computación Salas A y B Manual de prácticas de Estructura de Datos y Algoritmos I MADO-19\_EDAI  
Guía práctica de estudio 11:

https://retina.elpais.com/retina/2018/03/22/tendencias/1521745909\_941081.html