	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Adrian Ulises Mercado Martinez

Asignatura: Estructura de Datos y Algoritmos I

Grupo: 13

No de Práctica(s): 11

Integrante(s): Méndez Bernal Luis Alberto

No. de Equipo de cómputo empleado:

No. de Lista o Brigada: 13

Semestre: 2020.2

Fecha de entrega: 7 de junio del 2020

Observaciones:

CALIFICACIÓN: _____

INTRODUCCION

La practica se llama “Estrategias para la construcción de algoritmos” tiene como objetivo que implementemos, al menos, dos enfoques de diseño de algoritmos y ver que consecuencias tiene cada uno.
La costruccion de algoritmos es el objetivo clave de la materia, y esta practica nos ayudara a ver cuales son las mejores formas de construir estos, sabiendo que vamos a trabajar.

DESARROLLO

FUERZA BRUTA

El objetivo de este método es hacer una búsqueda de todas las posibilidades que nos pueden llevar a una solución. Un ejemplo claro es cuando no sabemos la combinación de una caja con 5 dígitos y probamos todas y cada una de las posibles combinaciones hasta que lleguemos a abrirla. La desventaja de este método es que se toma mucho tiempo en pasar por todas las combinaciones hasta que llegue a alguna que sea real.

```
from string import ascii_lettes, digits
from inertools import product

caracteres = ascii_letters+digits

def buscador(con):
    archivo = open("combinaciones.txt", "w")

    if 3 <= len(con) <= 4:
        for icin range (3,5):
            for comb in product(caracteres, repeat = i):
                prueba = "", join(comb)
                archivo.write(prueba + "\n")
                if prueba == con:
                    print('Tu contraseña es ()', format(prueba))
                    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 ejecucion ()", format(round(time()-t0, 6)))

```

Ejecución:

```
Tu contraseña es H0l4
Tiempo de ejecucion 20.43123
```

ALGORITMOS AVIDOS

Esta estrategia trabaja analizando los valores y tratando de tomar las mejores decisiones para llegar a la respuesta. Esta tiene la ventaja de que suele ser más rápida que lado Fuerza bruta, pero el resultado no siempre es el adecuado.

```
34  def cambio(cantidad, denominaciones):
35      resultado=[]
36      while (cantidad > 0):
37          if (cantidad >= denominaicones [0]):
38              num = cantidad // denominaciones[0]
39              cantidad = canitidad - (num * denominaciones[0])
40              resultado.append([denominaciones[0], num])
41          denominaciones = denominaciones[1:]
42      return resultado
```

Ejemplos

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

[[500, 2]]
```

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

[[50, 1]. [20,2], [5,1], [1,3]]
```

Este caso no siempre regresa el valor optimo, ya que a veces cree que hay una mejor opción

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

[[5,19], [1,3]]
```

BOTTOM-UP

El objetivo de este método es resolver problemas a partir de otros subproblemas ya resueltos. La solución final se formará a partir de la combinación de una o mas soluciones que se guardaran en una tabla para que después se reutilicen.

```
56  def fibonacci(numero):
57      f1=0
58      f2=1
59      tmp=0
60      for i in range(1, numero-1):
61          tmp = f1 + f2
62          f1= f2
63          f2= tmp
64      return f2
```

Aquí creamos un código que usara este método para trabajar con la sucesión Fibonacci

```
fibonacci(6)
5
```

0+1=1 1+1=2 1+2=3 2+3=5

TOP-DOWN

Este método es el contrario al Bottom-up, ya que este empieza a hacer cálculos desde n hacia abajo. También se aplica la técnica de memorización que consiste en guardar resultados previos para no tener que repetir operaciones.

DIVIDE Y VENCERAS

Este es de las mejores estrategias que podemos usar en la vida cotidiana. Este problema consiste en dividir un problema en subproblemas hasta llegar a uno que sea fácil de solucionar directamente.

```
75  def quicksort_aux(lista, inicio, fin):
76      id inicio < fin:
77          pivote = particion(lista, inicio, fin)
78
79          quicksort_aux(lista, inicio, pivote-1)
80          quicksort_aux(lista, pivote+1, fin)
81
82  def particion(lista, inicio, fin):
83      pivote = lista[inicio]
84      print("Valor del pivote ()", format(pivote))
85      izquierda = inicio
86      derecha = fin
87      print("Indice izquierdo {}".format(izquierda))
88      print("Indice derecho {}". format(derecha))
89
90      bandera = False
91      while not bandera:
92          while izquierda <= derecha and lista(izquierda) <= pivote:
93              izquierda = izquierda + 1
94          while lista[derecha] >= pivote and derecha >= izquierda:
95              derecha = derecha -1
96          if derecha < izquierda:
97              bandera = True
98          else:
99              temp = lista[izquierda]
100             lista[izquierda]=lista[derecha]
101             lista[derecha]= temp
102
103         print(lista)
104
105         temp=lista[inicio]
106         lista[inicio]=lista[derecha]
107         lista[derecha]=temp
108         return derecha
```

Este programa lo que hace es dividir una lista de números, en menores cantidades para si poder acomodarla de menora mayor.

```
lista = [21, 10, 0, 11, 9, 24, 20, 14, 1]
print("lista desordenada {}".format(lista))
quicksort(lista)
print("lista ordenada {}".format(lista))
```

```
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]
```

CONCLUCION

Esta practica tenia la finalidad de que aprendiéramos los distintos tipos de algoritmos que podemos aplicar a la computadora. El saber esto me permitirá saber que algoritmo debo usar con base en el problema que tenga. Durante una de las clases vimos como usábamos el juego de los discos como ejemplo y con el algoritmo de divide y vencerás pudimos deducir que solo necesitamos acomodar todas las piezas en un poste auxiliar hasta llegar a la pieza mas grande de estas. Esto me ayuda para saber qué y cómo usar los algoritmos para facilitarme el trabajo.