



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Adrian Ulises Mercado Marínez

Asignatura: Estructura de Datos y Algoritmos I

Grupo: 13

No de Práctica(s): 11

Integrante(s): Martínez Salgado Elean Jim

*No. de Equipo de
cómputo empleado:* -

*No. de Equipo de
cómputo empleado:*

No. de Lista o Brigada:

Semestre: 2020-2

Fecha de entrega: 7 de junio de 2020

Observaciones:

CALIFICACIÓN: _____

introducción

En esta practica se abarcan 6 estrategias para la construcción de algoritmos: Fuerza bruta, greedy, bottom-up, top-down, incremental y por ultimo divide y vencerás

Desarrollo

- 1- Utilizamos la estrategia de fuerza bruta generando combinaciones con el fin de encontrar una contraseña, dependiendo de la potencia del equipo este tardara mas o menos, primero se abre un archivo de texto en el que se guardaran todas la combinaciones y para obtener dichas combinaciones utilizamos la función producto, contenida en la biblioteca itertools y por ultimo con un ciclo for se prueba cada una.

```
Bienvenido  1.py  x  2.py  3.py  4.py
1.py
1  #Estrategia de busqueda de fuerza bruta
2  #realiza una busqueda exhaustiva
3
4  from string import ascii_letters, digits
5  from itertools import product
6  from time import time
7
8  caracteres = ascii_letters + digits
9
10 def buscar(con):
11     #Abrir el archivo con las cadenas generadas
12     archivo = open("combinaciones.txt", "w")
13
14     if 3<= len(con) <= 4:
15         for i in range(3, 5):
16             for comb in product(caracteres, repeat = i):
17                 prueba = "".join(comb)
18                 archivo.write(prueba+"\n")
19                 if prueba == con:
20                     print("La contraseña es {}".format(prueba))
21                     break
22             archivo.close()
23     else:
24         print("Ingresa una contraseña de longitud 3 o 4")
25
26
27 if __name__=="__main__":
28     con = input("Ingresa una contraseña\n")
29     t0 = time()
30     buscar(con)
31     print("Tiempo de ejecucion {}".format(round(time()-t0,6)))
```

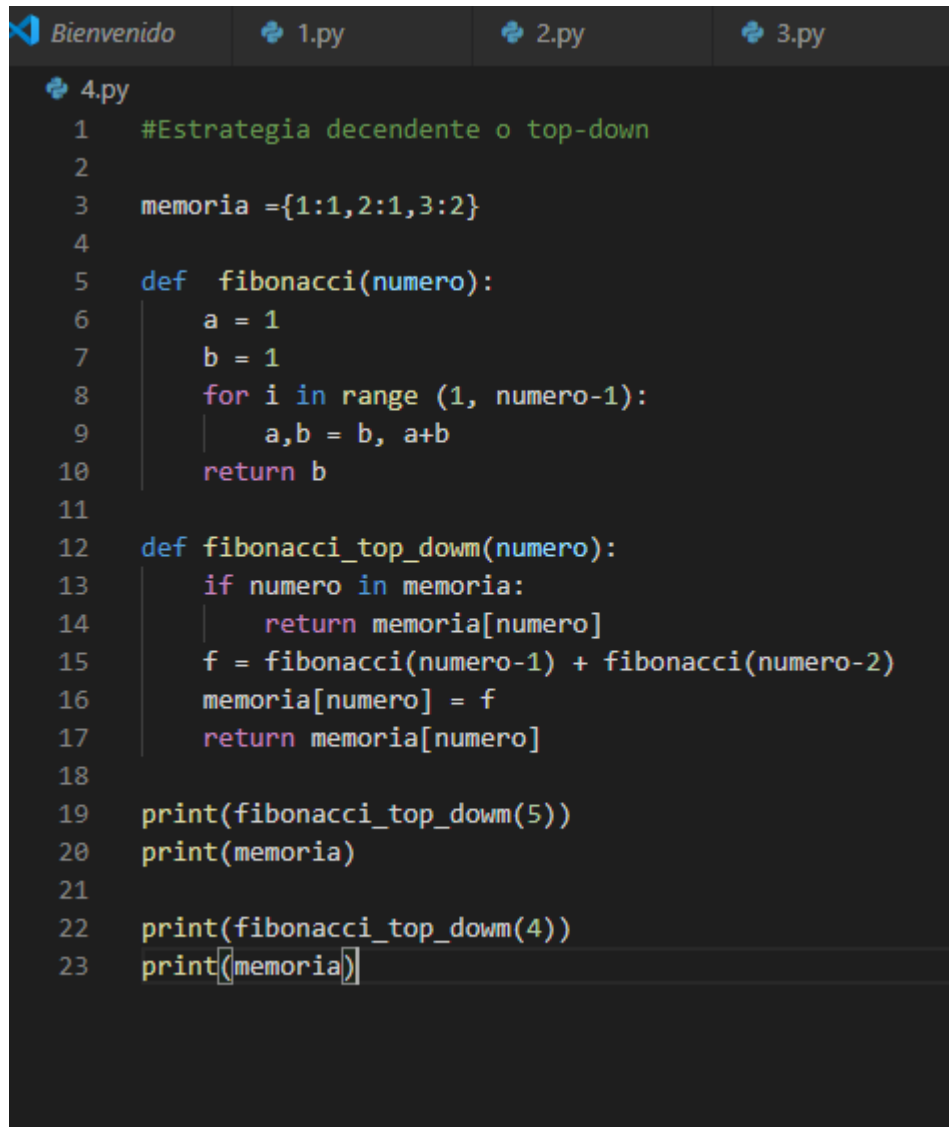
- 2- Utilizamos la estrategia greedy, buscábamos crear un programa que devolviera el cambio de monedas de acuerdo con la cantidad de dinero ingresado y el número de monedas que teníamos para dar cambio, con el objetivo de devolver el menor número de monedas, creamos un algoritmo que toma las monedas del mayor hasta que llegue al límite y entonces puede usar las monedas del siguiente valor.

```
2.py
1  #solucion con algoritmo greegy o voraz
2
3  def cambio(cantidad, monedas):
4      resultado = []
5      while cantidad > 0:
6          if cantidad >= monedas[0]:
7              num = cantidad // monedas[0]
8              cantidad = cantidad - (num*monedas[0])
9              resultado.append([monedas[0], num])
10             monedas = monedas[1:]
11         return resultado
12
13
14  if __name__ == "__main__":
15      print(cambio(1000, [20, 10, 5, 2, 1]))
16      print(cambio(20, [20, 10, 5, 2, 1]))
17      print(cambio(30, [20, 10, 5, 2, 1]))
18      print(cambio(98, [5, 20, 1, 50]))
19      print(cambio(98, [50, 20, 5, 1]))
```

- 3- Utilizamos la estrategia bottom-up aquí el objetivo es obtener el termino n de la sucesión de Fibonacci, las primeras dos funciones son la solución iterativa del problema mientras que la tercer es la que hace referencia a la estrategia de bottom-up en ella hay un arreglo de soluciones parciales de los primeros números, de manera que cuando se pida un cierto numero n de la sucesión, se calculan los resultados desde esos casos base, hasta llegar a n, de abajo hacia arriba.

```
3.py
17     u = 1
18     for i in range (1, numero-1):
19         a,b = b, a+b
20     return b
21
22 def fibonacci_bottom_up(numero):
23     fib_parcial = [1, 1, 2]
24     while len(fib_parcial) < numero:
25         fib_parcial.append(fib_parcial[-1]+fib_parcial[-2])
26         print(fib_parcial)
27     return fib_parcial[numero-1]
28
29
30 f = fibonacci(10)
31 f2 = fibonacci2(10)
32 f3 = fibonacci_bottom_up(10)
33
34 print(f)
35 print(f2)
```

- 4- Utilizamos la estrategia top-down en esta se resuelve lo mismo que en la anterior solo que haciendo uso de un diccionario que guarda los resultados generados para cuando se requiera un termino n, se comprueba si existe en el diccionario , si no existe se calcula y guarda.



```
Bienvenido  1.py  2.py  3.py
4.py
1  #Estrategia decendente o top-down
2
3  memoria ={1:1,2:1,3:2}
4
5  def  fibonacci(numero):
6      a = 1
7      b = 1
8      for i in range (1, numero-1):
9          a,b = b, a+b
10     return b
11
12  def fibonacci_top_down(numero):
13     if numero in memoria:
14         return memoria[numero]
15     f = fibonacci(numero-1) + fibonacci(numero-2)
16     memoria[numero] = f
17     return memoria[numero]
18
19  print(fibonacci_top_down(5))
20  print(memoria)
21
22  print(fibonacci_top_down(4))
23  print(memoria)
```

- 5- Utilizamos la estrategia incremental aquí el algoritmo ordena una lista, y parte de que el primer elemento está ordenado después compara con el segundo elemento y así sucesivamente.

```
5.py
1  #Estrategia incremental
2  #Algoritmo de ordenacion por insercion
3  '''
4  21 10 12 0 34 15
5  Parte ordenada
6  21                10 12 0 34 15
7  10 21            12 0 34 15
8  10 12 21        0 34 15
9  0 10 12 21      34 15
10 0 10 12 21 34   15
11 0 10 12 15 21 34
12 '''
13
14 def insertSort(lista):
15     for index in range(1, len(lista)):
16         actual = lista[index]
17         posicion = index
18         #print("Valor a ordenar {}".format(actual))
19         while posicion > 0 and lista[posicion-1] > actual:
20             lista[posicion] = lista[posicion-1]
21             posicion = posicion-1
22         lista[posicion] = actual
23         #print(lista)
24         #print()
25     return lista
26
27 lista = [21, 10, 12, 0, 34, 15]
28 #print(lista)
29 insertSort(lista)
30 #print(lista)
```

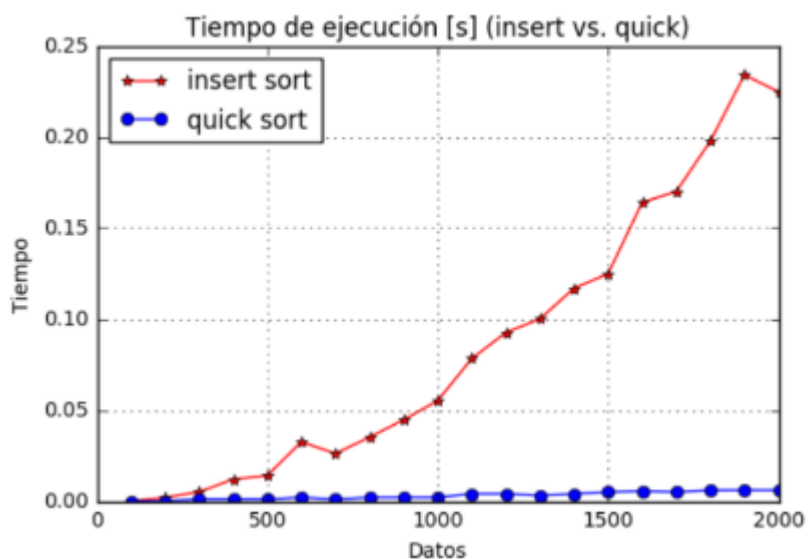
- 6- Utilizamos la estrategia divide y venceras en donde ordenamos números a partir de un pivote y posiciones, partiendo de un arreglo de números con el fin de que sea mas sencillo.

```
6.py
1  #Estrategia divide y venceras
2  '''
3  21 10 12 0 34 15
4  p   i           d
5
6  '''
7
8  def quicksort(lista):
9      quicksort2(lista, 0, len(lista)-1)
10
11  def quicksort2(lista, inicio, fin):
12      if inicio < fin:
13          pivote = particion(lista, inicio, fin)
14          quicksort2(lista, inicio, pivote-1)
15          quicksort2(lista, pivote+1, fin)
16
17  def particion(lista, inicio, fin):
18      pivote = lista[inicio]
19      print("Valor el pivote {}",format(pivote))
20      izquierda = inicio+1
21      derecha = fin
22      print("Indice izquierda {} y indice derecha {}".format(izquierda, derecha))
23
24      bandera = False
25      while not bandera:
26          while izquierda <= derecha and lista[izquierda] <= pivote:
27              izquierda = izquierda + 1
28          while lista[derecha] >= izquierda and lista[derecha] >= pivote:
29              derecha = derecha - 1
30          if derecha < izquierda:
31              bandera = True
32          else:
33              temp = lista[izquierda]
34              lista[izquierda] = lista[derecha]
35              lista[derecha] = temp
36
37      print(lista)
38      temp = lista[inicio]
39      lista[inicio] = lista[derecha]
40      lista[derecha] = temp
41      return derecha
42
43  lista = [21,10,0,11,9,24,20,14,1]
44  print(lista)
45  quicksort(lista)
46  print(lista)
```

- 7- Los programas 7 y 8 son acerca de medición y graficas en tiempos de ejecución aquí se comparan las dos funciones de ordenación vistas.

Ejercicio 7

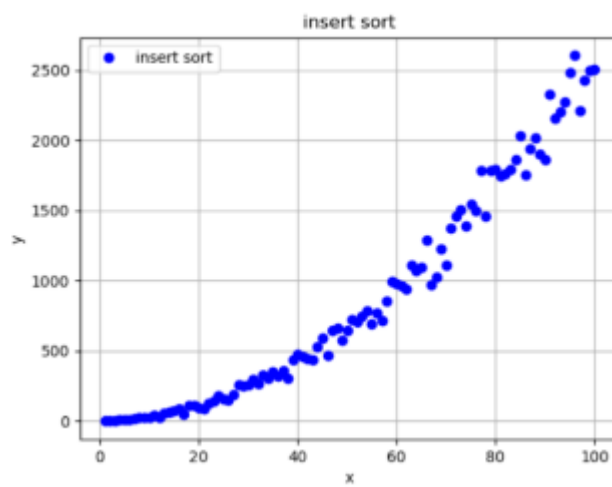
```
7.py
1  import matplotlib.pyplot as plt
2  from mpl_toolkits.mplot3d import Axes3D
3  import random
4  from time import time
5  from ejercicio5 import insertSort
6  from ejercicio6 import quicksort
7
8  datos = [ii+100 for ii in range(1,21)]
9  tiempo_is = []
10 tiempo_qs = []
11 for ii in datos:
12     lista_is = random.sample(range(0,10000000), ii)
13     lista_qs = lista_is.copy()
14
15     t0 = time()
16     insertSort(lista_is)
17     tiempo_is.append(round(time()-t0,6))
18
19     t0 = time()
20     quicksort(lista_qs)
21     tiempo_qs.append(round(time()-t0,6))
22
23 print("Tiempos parciales de ejecucion en insertSort {}".format(tiempo_is))
24 print("Tiempos parciales de ejecucion en quicksort {}".format(tiempo_qs))
25
26 fig, ax = plt.subplots()
27 ax.plot(datos, tiempo_is, label="insertSort", marker="*", color="r")
28 ax.plot(datos, tiempo_qs, label="quicksort", marker="o", color="b")
29
30 ax.set_xlabel("Datos")
31 ax.set_ylabel("Tiempo")
32 ax.grid(True)
33 ax.legend(loc=2)
34
35 plt.title("Tiempos de ejecucion [s] insertSort vs quicksort")
36 plt.show()
```



Utilizando la herramienta de graficas de datos de Python podemos apreciar una comparativa entre insert sort e Quicksort para ver cual es mas eficiente y podemos ver que gana Quicksort en cuanto a tiempo pero en datos tenemos un empate.

Ejercicio 8

```
8.py
1 import matplotlib.pyplot as plt
2 from mpl_toolkits.mplot3d import Axes3D
3 import random
4
5 times = 0
6
7 def insertSort(lista):
8     global times
9     for i in range(1, len(lista)):
10         times += 1
11         actual = lista[i]
12         posicion = i
13         while posicion > 0 and lista[posicion-1] > actual:
14             times += 1
15             lista[posicion] = lista[posicion-1]
16             posicion = posicion-1
17         lista[posicion] = actual
18     return lista
19
20 TAM = 101
21 eje_x = list(range(1, TAM, 1))
22 eje_y = []
23
24 lista_variable = []
25
26 for num in eje_x:
27     lista_variable = random.sample(range(0, 1000), num)
28     times = 0
29     lista_variable = insertSort(lista_variable)
30     eje_y.append(times)
31
32
33 fig, ax = plt.subplots(facecolor='w', edgecolor='k')
34 ax.plot(eje_x, eje_y, marker="o", color="b", linestyle="None")
35 ax.set_xlabel('x')
36 ax.set_ylabel('y')
37 ax.grid(True)
38 ax.legend(["insert sort"])
39
40 plt.title("Insert sort")
41 plt.show()
```



Aquí observamos una prueba de testeo de la función insert sort de matplotlib y otras de Python, en la grafica se observa que varia la línea exponencial de insert sort, lo que indica que hay pocos datos y a su vez que requiere mas tiempo de realización.

Conclusión

Esta práctica resulta muy útil para conocer estrategias de solución de problemas utilizando el lenguaje Python además de que pusimos en práctica conocimiento de anteriores prácticas como lo de la elaboración de graficas además de conocer los métodos de ordenamiento existente en el lenguaje Python.