


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

Laboratorios de computación sala A y B

| | |
|---|-------------------------------------|
| <i>Profesor:</i> | Ing. Adrian Ulises Mercado Martinez |
| <i>Asignatura:</i> | Estructura de datos y algoritmos I |
| <i>Grupo:</i> | 13 |
| <i>No de Práctica(s):</i> | 11 |
| <i>Integrante(s):</i> | Oscar Tovar Mendoza |
| <i>No. de Equipo de cómputo empleado:</i> | |
| <i>No. de Lista o Brigada:</i> | 8 |
| <i>Semestre:</i> | 2 |
| <i>Fecha de entrega:</i> | Domingo, 7 de junio de 2020 |

CALIFICACIÓN: _____

Introducción

Existen distintas estrategias para la elaboración de un algoritmo, en esta práctica haremos uso de algunas como fuerza bruta, greedy o voraz, bottom-up, top-down, incremental, divide y vencerás.

Desarrollo

Como primera estrategia usamos el de búsqueda de fuerza bruta, en este ejemplo generamos combinaciones para encontrar una contraseña, dependiendo del equipo usado el tiempo que tardará sería mas o menos

```
ejercicio1.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)))
32
```

En el segundo ejercicio usamos la solución con algoritmo voraz, en este se nos daba una cantidad de dinero y dependiendo las denominaciones que tuviésemos nos daría cambio

```
ejercicio2.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]))
```

En este ejercicio usamos la estrategia bottom-up para realizar el número de fibonacci que corresponda al número que le demos

```
ejercicio3.py > ...
1  #Estrategia bottom-up o programacion dinámica
2
3
4  def fibonacci(numero):
5      a = 1
6      b = 1
7      c = 0
8      for i in range (1, numero-1):
9          c = a + b
10         a = b
11         b = c
12     return c
13
14
15  def fibonacci2(numero):
16      a = 1
17      b = 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)
```

En este ejercicio usamos de nueva cuenta el ejemplo de fibonacci, sólo que utilizamos un recursividad

```
ejercicio4.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)
```

En este ejercicio usamos la estrategia incremental para ordenar números de menor a mayor de una lista

```
ejercicio5.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)
31
32
```

En el ejercicio 6 usamos la estrategia divide y vencerás, en donde ordenamos números a partir de un pivote y posiciones, partiendo un arreglo de números para que fuera más fácil

```
ejercicio6.py > ...
1  #Estrategia divide y venceras
2  '''
3  21 10 12 0 34 15
4  p  i      d
5
6
7  '''
8
9  def quicksort(lista):
10     quicksort2(lista, 0, len(lista)-1)
11
12  def quicksort2(lista, inicio, fin):
13     if inicio < fin:
14         pivote = particion(lista, inicio, fin)
15         quicksort2(lista, inicio, pivote-1)
16         quicksort2(lista, pivote+1, fin)
17
18  def particion(lista, inicio, fin):
19     pivote = lista[inicio]
20     print("Valor el pivote {}".format(pivote))
21     izquierda = inicio+1
22     derecha = fin
23     print("Indice izquierda {} y indice derecha {}".format(izquierda, derecha))
24
25     bandera = False
26     while not bandera:
27         while izquierda <= derecha and lista[izquierda] <= pivote:
28             izquierda = izquierda + 1
29         while lista[derecha] >= izquierda and lista[derecha] >= pivote:
30             derecha = derecha - 1
31         if derecha < izquierda:
32             bandera = True
33         else:
34             temp = lista[izquierda]
35             lista[izquierda] = lista[derecha]
36             lista[derecha] = temp
37
38     print(lista)
39     temp = lista[inicio]
40     lista[inicio] = lista[derecha]
41     lista[derecha] = temp
42     return derecha
43
44  lista = [21,10,0,11,9,24,20,14,1]
45  print(lista)
46  quicksort(lista)
47  print(lista)
48
```


Para los ejercicios 7 y 8, graficamos las funciones de quicksort y insertsort, con tiempos generados por un random , además de tiempo en que tardaba en ejecución

```
ejercicio7.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 inserSort {}".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="inserSort", 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] inserSort vs quicksort")
36 plt.show()
```

```

ejercicio8.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()

```


Conclusiones

- Las estrategias son muy importantes, ya que depende del problema que se presente, cada una de estas será mejor que la otra y ayudará a resolver el problema más fácil, así que tener en cuenta los diferentes tipos y sus usos, es indispensable (*Oscar Tovar Mendoza*)