



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación
Salas A y B

Profesor:

Martínez Quintana Marco Antonio

Asignatura:

Estructura de Datos y Algoritmos 1.

Grupo:

17

No de Práctica(s):

11

Integrante(s):

Ruiz Godoy Franco

*No. de Equipo de
cómputo empleado:*

7

No. de Lista o Brigada:

Semestre:

2020-2

Fecha de entrega:

26 de Abril de 2020

Observaciones:

CALIFICACIÓN:

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.

Fuerza bruta

El objetivo de resolver problemas por medio de fuerza es bruta es hacer una búsqueda exhaustiva de todas las posibilidades que lleven a la solución del problema. Un ejemplo de esto es encontrar una contraseña haciendo una combinación exhaustiva de caracteres alfanuméricos generando cadenas de cierta longitud. La desventaja de resolver problemas por medio de esta estrategia es el tiempo que toman.

Algoritmos ávidos (greedy)

Esta estrategia se diferencia de fuerza bruta porque va tomando una serie de decisiones en un orden específico, una vez que se ha ejecutado esa decisión, ya no se vuelve a considerar. En comparación con fuerza bruta, ésta puede ser más rápida; aunque una desventaja es que la solución que se obtiene no siempre es la más óptima.

Bottom-up (programación dinámica)

El objetivo de esta estrategia es resolver un problema a partir de subproblemas que ya han sido resueltos. La solución final se forma a partir de la combinación de una o más soluciones que se guardan en una tabla, ésta previene que se vuelvan a calcular las soluciones. Como ejemplo, se va a calcular el número n de la sucesión de Fibonacci.

La sucesión de Fibonacci es una sucesión infinita de números enteros cuyos primeros dos elementos son 0 y 1, los siguientes números son calculados por la suma de los dos anteriores.

Top-down

A diferencia de bottom-up, aquí se empiezan a hacer los cálculos de n hacia abajo. Además, se aplica una técnica llamada memorización la cual consiste en guardar los resultados previamente calculados, de tal manera que no se tengan que repetir operaciones. Para aplicar la estrategia top-down, se utiliza un diccionario (memoria) el cual va a almacenar valores previamente calculados. Una vez que se realice el cálculo de algún elemento de la sucesión de Fibonacci, éste se va a almacenar ahí.

Incremental

Es una estrategia que consiste en implementar y probar que sea correcto de manera paulatina, ya que en cada iteración se va agregando información hasta completar la tarea. [Insertion sort](#)

Insertion sort ordena los elementos manteniendo una sublista de números ordenados empezando por las primeras localidades de la lista. Al principio se

considera que el elemento en la primera posición de la lista está ordenado. Después cada uno de los elementos de la lista se compara con la sublista ordenada para encontrar la posición adecuada.

Divide y vencerás

Es una estrategia que consiste en: Dividir el problema en subproblemas hasta que son suficientemente simples que se pueden resolver directamente. Después las soluciones son combinadas para generar la solución general del problema.

Quick sort

Quicksort es un ejemplo de resolver un problema por medio de la estrategia divide y vencerás. En Quicksort se divide en dos el arreglo que va a ser ordenado y se llama recursivamente para ordenar las divisiones. La parte más importante en Quicksort es la partición de los datos. Lo primero que se necesita es escoger un valor de pivote el cual está encargado de ayudar con la partición de los datos. El objetivo de dividir los datos es mover los que se encuentran en una posición incorrecta con respecto al pivote.

Modelo RAM

Cuando se realiza un análisis de complejidad utilizando el modelo RAM, se debe contabilizar las veces que se ejecuta una función o un ciclo, en lugar de medir el tiempo de ejecución

Desarrollo.

Fuerza Bruta.

```
from string import ascii_letters , digits
from itertools import product

caracteres = ascii_letters + digits

def buscador(con):
    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('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 = 'H014'
buscador (con)
print("Tiempo de ejecucion {}".format(round(time()-t0, 6)))
```

H01y
H01z
H01A
H01B
H01C
H01D
H01E
H01F
H01G
H01H
H01I
H01J
H01K
H01L
H01M
H01N
H01O
H01P
H01Q
H01R
H01S
H01T
H01U
H01V
H01W
H01X
H01Y
H01Z
H010
H011
H012
H013
H014

```

from string import ascii_letters , digits
from itertools import product

caracteres = ascii_letters + digits

def buscador(con):
    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('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 = 'H014'
buscador (con)
print("Tiempo de ejecucion {}".format(round(time()-t0, 6)))

```

```

File Edit Shell Debug Options Window
Python 3.8.1 (tags/v3.8.1:1b293b
tel)] on win32
Type "help", "copyright", "credi
>>>
===== RESTART: C:/Users/Franc
===== RESTART: C:/Users/Franc
Tu contraseña es H014
Tiempo de ejecucion 107.280136
>>> |

```

Algoritmos ávidos (greedy)

```

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:]
    return resultado

print (cambio(1000, [500, 200, 100, 50, 20, 5, 1]))
print (cambio(500, [500, 200, 100, 50, 20, 5, 1]))
print (cambio(300, [50, 20, 5, 1]))
print (cambio(200, [5]))
print (cambio(98, [50, 20, 5, 1]))
print (cambio(98, [5, 20, 1, 50]))

```

```

Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MS
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more i
>>>
===== RESTART: C:/Users/Franco/Desktop/EDA/practica 11/algori
[[500, 2]]
[[500, 1]]
[[50, 6]]
[[5, 40]]
[[50, 1], [20, 2], [5, 1], [1, 3]]
[[5, 19], [1, 3]]
>>> |

```

Bottom-up (programación dinámica)


```
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
```

```
fibonacci_iterativo_v1(6)
```

```
def fibonacci_iterativo_v2(numero):  
    f1=0  
    f2=1  
    for i in range (1, numero-1):  
        f1,f2=f2,f1+f2  
    return f1
```

```
fibonacci_iterativo_v2(6)
```

```
def fibonacci_bottom_up(numero):  
    f_parciales = [0, 1, 1]  
    while len (f_parciales) < numero:  
        f_parciales.append(f_parciales[-1] + f_parciales [-2])  
        print(f_parciales)  
    return f_parciales[numero-1]  
fibonacci_bottom_up(5)
```

 Python 3.8.1 Shell

File Edit Shell Debug Options Window Help

Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v
tel)] on win32

Type "help", "copyright", "credits" or "license()" for more info

>>>

===== RESTART: C:/Users/Franco/Desktop/EDA/practica 11/Botto

[0, 1, 1, 2]

[0, 1, 1, 2, 3]

>>>

Top-down

```
memoria = {1:0, 2:1, 3:1}
def fibonacci_iterativo_v2(numero):
    f1=0
    f2=1
    for i in range (1, numero-1):
        f1,f2=f2,f1+f2
    return f1

fibonacci_iterativo_v2(6)

def fibonacci_top_down (numero):
    if numero in memoria:
        return memoria[numero]
    f = fibonacci_iterativo_v2(numero-1) + fibonacci_iterativo_v2(numero-2)
    memoria[numero] = f
    return memoria [numero]

fibonacci_top_down(12)
memoria
fibonacci_top_down(8)
memoria

import pickle
archivo = open('memoria.p', 'wb')
pickle.dump(memoria, archivo)
archivo.close()

archivo = open ('memoria.p', 'rb')
memoria_de_archivo = pickle.load(archivo)
archivo.close()
memoria
memoria_de_archivo
```

Incremental

```
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

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

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 2
tel)] on win32
Type "help", "copyright", "credits" or "license()
>>>
===== RESTART: C:/Users/Franco/Desktop/EDA/pr
Lista desordenada [21, 10, 0, 11, 9, 24, 20, 14,
Valor a ordenar = 10
[10, 21, 0, 11, 9, 24, 20, 14, 1]

Valor a ordenar = 0
[0, 10, 21, 11, 9, 24, 20, 14, 1]

Valor a ordenar = 11
[0, 10, 11, 21, 9, 24, 20, 14, 1]

Valor a ordenar = 9
[0, 9, 10, 11, 21, 24, 20, 14, 1]

Valor a ordenar = 24
[0, 9, 10, 11, 21, 24, 20, 14, 1]

Valor a ordenar = 20
[0, 9, 10, 11, 20, 21, 24, 14, 1]

Valor a ordenar = 14
[0, 9, 10, 11, 14, 20, 21, 24, 1]

Valor a ordenar = 1
[0, 1, 9, 10, 11, 14, 20, 21, 24]

Lista ordenada [0, 1, 9, 10, 11, 14, 20, 21, 24]
>>> |
```

Divide y vencerás.

```
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):
    pivote = lista[inicio]
    print("Valor del pivote {}".format(pivote))
    izquierda = inicio+1
    derecha = fin
    print("Indice izquierdo {}".format(izquierda))
    print("Indice derecha {}".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

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

```
-----
Lista desordenada [21, 10, 0, 11, 9, 24, 20, 14, 1]
Valor del pivote 21
```

Medición y gráficas de los tiempos de ejecución

```
#!/usr/bin/env python
#pylab inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import random
from time import time

from insertionSort import insertionSort_time
from quickSort import quickSort_time

datos = [ii*100 for ii in range(0,21)]

tiempo_is = []
tiempo_qs = []

for ii in datos:
    lista_is = random.sample(range(0,100000000),ii)
    lista_qs = lista_is.copy()

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

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

print("Tiempos parciales de ejecución en INSERT SORT {} [s]\n".format(tiempo_is))
print("Tiempos parciales de ejecución en QUICK SORT {} [s]\n".format(tiempo_qs))

print("Tiempo total de ejecución en INSERT SORT {} [s]\n".format(sum(tiempo_is)))
print("Tiempo total de ejecución en QUICK SORT {} [s]\n".format(sum(tiempo_qs)))

fig, ax =.subplots()
ax.plot(datos, tiempo_is, label="insert sort", marker="x", color="r")
ax.plot(datos, tiempo_qs, label="quick sort", marker="o", color="b")
ax.set_xlabel('Datos')
ax.set_ylabel('Tiempo')
ax.grid(True)
ax.legend(loc=2);
```


Modelo Ram

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

times = 0

def insertionSort_graph(n_lista):
    global times
    for index in range(1, len(n_lista)):
        times += 1
        actual = n_lista[index]
        posicion = index
        while posicion > 0 and n_lista[posicion-1] > actual:
            times += 1
            n_lista[posicion] = n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion] = actual
    return n_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, 1000), num)
    times = 0
    lista_variable = insertionSort_graph(lista_variable)
    eje_y.append(times)

fig, ax = plt.subplots(facecolor='w', edgecolor='k')
ax.plot(eje_x, eje_y, marker="o", color="b", linestyle='None')

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.grid(True)
ax.legend(["Insertion Sort"])

plt.title('Insertion sort')
plt.show()
```

Conclusiones.

Construir algoritmos es esencial al aprender un nuevo lenguaje, saber sus características, las bibliotecas que lleva a cada uno y sus diferentes funciones para reconocer en que se puede aplicar cada uno en la elaboración de un nuevo proyecto, ya que una herramienta sirve para resolver paso a paso un problema. Se trata de una serie de instrucciones ordenadas y secuenciadas para guiar un proceso determinado.

Referencias.

García Cano, E. E. G. C., & Solano Gálvez, J. A. S. G. (2017, enero 20). Guía práctica de estudio 11:Estrategias para la construcción de algoritmos.

Recuperado 26 de abril de 2020, de <http://lcp02.fi->

[b.unam.mx/static/docs/PRACTICAS_EDA1/eda1_p11.pdf](http://lcp02.fi-b.unam.mx/static/docs/PRACTICAS_EDA1/eda1_p11.pdf)