UNIVERSIDAD
NACIONAL
DE LA PLATA

CyC - Practica 8

Facundo Tomatis

(Ejercicio 1)

Consigna:
Determinar para cada funcion $t(n)$ en la siguiente tabla, cual es el mayor tamaño n de una instancia de un problema que puede ser resuelto en cada uno de los tiempos indicados en las columnas de la tabla, suponiendo que el algoritmo para resolverlo utiliza $t(n)$ microsegundos

$t(n)$	1 seg.	1 min.	1 hora	1 dia	1 mes	1 año	1 siglo
$log_2(n)$	2^{10^6}	$2^{6\times10^7}$	$2^{36\times10^8}$	$2^{24\times36\times10^8}$	$2^{30\times864\times10^8}$	$2^{12\times25920\times10^8}$	$2^{31104\times10^{11}}$
$\sqrt{n}$	$[10^6]^2$	$[6 \times 10^7]^2$	$[36 \times 10^8]^2$	$[24 \times 36 \times 10^8]^2$	$[30 \times 864 \times 10^8]^2$	$[12 \times 2592 \times 10^9]^2$	$[31104 \times 10^{11}]^2$
n	10^6	6×10^7	36×10^8	$24 \times 36 \times 10^8$	$30 \times 864 \times 10^8$	$12 \times 25920 \times 10^8$	31104×10^{11}
$n \times log_2(n)$	62746.12	280141 7.88	133378058 .86	2755147513 .21	7187085640 4.05	787089606198 .48	6769949846 3641.914
n^2	$\sqrt{10^6}$	$\sqrt{6 \times 10^7}$	$\sqrt{36 \times 10^8}$	$[24 \times 36 \times 10^8]^{1/2}$	$[30 \times 864 \times 10^8]^{1/2}$	$[12 \times 2592 \times 10^9]^{1/2}$	$[31104 \times 10^{11}]^{1/2}$
2^n	$\frac{ln(10^6)}{ln(2)}$	$\frac{ln(6\times10^7)}{ln(2)}$	$\frac{ln(36\times10^8)}{ln(2)}$	$\frac{ln(24\times36\times10^8)}{ln(2)}$	$\frac{ln(30\times864\times10^8)}{ln(2)}$	$\frac{ln(12\times2592\times10^9)}{ln(2)}$	$\frac{ln(31104\times10^{11})}{ln(2)}$
$n!$	9.4456	11.16635	12.78884	13.99664	15.24888	16.14121	17.75201

Algoritmos para sacar incisos con una precision de 5 digitos

```python
import math
import numpy as np

def fac(n):
    return math.gamma(n)

# time in microseconds
arr = {
    'second':               1000000,
    'minute':          6 * 10000000,
    'hour':           36 * 100000000,
    'day':           864 * 100000000,
    'month':       25920 * 100000000,
    'year':      31104 * 1000000000,
    'century':31104 * 100000000000,
}

# factorial equation with input
def fac_equation(n):
    for i in np.arange(8, 20,0.00001):
        if fac(i) > n:
            return i-1

for k, v in arr.items():
```

```
    print(k, fac_equation(v))

# n times log_2 n equation with input
def nlogn_equation(n):
    return math.exp(sp.lambertw(n*math.log(2)))

for k, v in arr.items():
    print(k, nlogn_equation(v))
```

(Ejercicio 2)

Consigna:
Si el tiempo de ejecucion en el mejor caso de un algoritmo, $t_m(n)$, es tal que $t_m(n) \in \Omega(f(n))$ y el tiempo de ejecucion en el peor caso de un algoritmo, $t_p(n)$, es tal que $t_p(n) \in O(f(n))$, ¿Se puede afirmar que el tiempo de ejecucion del algoritmo es $\Theta(f(n))$?
Respuesta:
Si, ya que nos esta asegurando que el mejor caso va a ser mayor que f(n) con una constante y que en el peor caso va a ser menor que f(n) con una constante. Por definicion $\Theta(f(n))$ quedaria definida con la constante c_1 utilizada en $\Omega()$ y c_2 utilizada en O()

(Ejercicio 3)

Consigna:
Un algoritmo tarda 1 segundo en procesar 1000 items en una maquina determinada. ¿Cuanto tiempo tomara procesar 10000 items si se sabe que el tiempo de ejecucion del algoritmo es n^2? ¿y si se sabe que es $n \times log_2 n$? ¿Que se estaria asumiendo en todos los casos?
Respuesta:
Si tarda 1 segundo en procesar 1000 items con un algoritmo cuyo orden es n^2, va a tardar $1 \times 10^2 = 100$ segundos en procesar 10000 items
Si tarda 1 segundo en procesar 1000 items con un algoritmo cuyo orden es $n \times log_2 n$, va a tardar $\frac{10000 \times log_2 10000}{1000 \times log_2 1000} = 13.33$ segundos en procesar los 10000.
En todos los casos se asume el peor caso

(Ejercicio 4)

Consigna:
Un algoritmo toma n^2 dias y otro n^3 segundos para resolver una instancia de tamaño n de un problema. Mostrar que el segundo algoritmo superara en tiempo al primero solamente en instancias que requieran mas de 20 millones de años para ser resueltas.
Respuesta:
n^3 en 1 segundo
n^2 en 1 dia
paso a dias $n^3 = \frac{1}{86400} n^3$

igualo ambas funciones y me fijo en que punto se cruzan
$n^2 = \frac{1}{86400}n^3 \rightarrow 86400 = n$
n es la entrada en la cual ambas funciones se cruzan, para entradas mayores $\frac{1}{86400}n^3$ dias va a superar en tiempo a n^2 dias
$\frac{86400^2}{365} = 20451945.2055 > 20000000$ por lo que la afirmacion es correcta.

(Ejercicio 5)

Consigna:
¿Cuales y cuantas serian las operaciones elementales necesarias para multiplicar dos enteros n y m por medio del algoritmo enseñado en la escuela primaria? ¿Esta cantidad depende de la entrada? Justifique.
Respuesta:
Son necesarias $(\lfloor log_{10}(n) \rfloor + 1) \times (\lfloor log_{10}(m) \rfloor + 1)$ operaciones. Depende de la entrada si es mayor o menor.

```
sum=0
base_i=1
base_j=1
operand=[3,3,3] # n, len(n) = floor(log10(n)) + 1
multiplier=[1,2]

for i in operand[::-1]:              # 333
    base_j=1                         #  12
    for j in multiplier[::-1]:       # ----
        sum+=j*base_j*i*base_i       # 3996
        base_j*=10
    base_i*=10
print(sum)
```

(Ejercicio 6)

Consigna:
Dar el tiempo de ejecucion en funcion de n de los siguientes algoritmos y una $f(n)$ tal que el tiempo de ejecucion pertenezca a $\Theta(f(n))$. Determine si cada algoritmo o partes del mismo tiene casos de analisis (peor, mejor, etc.).

a) .
$p \leftarrow 0$
for i $\leftarrow 1$ to n do
. *for* j $\leftarrow 1$ to n^2 do
. . *for* k $\leftarrow 1$ to n^3 do
. . . $p \leftarrow p + 1$

b) .
$p \leftarrow 0$
for i $\leftarrow 1$ to n do
. *for* j $\leftarrow 1$ to i do
. . *for* k $\leftarrow 1$ to n do
. . . $p \leftarrow p + 1$

Respuesta:
a) Se hace 1 operacion constante inicializando p, en el for mas exterior hace n iteraciones, el interno a ese n^2 y el mas interior hace n^3 iteraciones multiplicado por una cantidad de operaciones constantes (c=2; 1 por suma, 1 por asignacion) por lo que en todos los casos (no existe peor, ni mejor, etc) el algoritmo va a tardar $t(n) = 1 + 2n^6$, por lo que puedo asegurar que $t(n) \in \Theta(n^6)$

b) Se hace 1 operacion constante inicializando p, en el for exterior se hace n iteraciones, interno a este contiene una instruccion barometro que se ejecuta n veces, y dentro de esta instruccion se realizan n iteraciones multiplicada por una constante c=2, de vuelta no hay distincion entre mejor ni peor caso y $t(n) = 1 + 2n^3$ tal que $t(n) \in \Theta(n^3)$

(Ejercicio 7)

Consigna:
Definir y analizar el tiempo de ejecucion de la multiplicacion de una matriz triangular inferior por una matriz completa (en la que todos sus elementos pueden ser diferentes de 0). ¿Que formas tiene de hallar $t(n)$? ¿De cuantas formas podria encontrar la pertenencia a $O()$ o a $\Theta()$ del algoritmo?
Respuesta:
El tiempo de ejecucion de multiplicar una matriz triangular inferior por una matriz completa tiene el mismo orden que la multiplicacion entre dos matrices "normales" $t(n) \in \Theta(n^3)$.
Lo diferencia en que al ser una matriz triangular inferior se pueden ahorrar unas iteraciones para disminuir la cantidad de operaciones elementales a un $t(n) = (n^2 \times (n+1))/2$.
Si por ejemplo la entrada tiene como $n = 5$ con el algoritmo original haria si o si en 125 operaciones pero aprovechando las matrices triangulares inferiores solo tendria que hacer 75 operaciones

```python
def mult(matrix,lower_triangular):
  make_empty_matrix = lambda n: [[0] * n for i in range(n)]
  result=make_empty_matrix(len(matrix))
  leng=len(matrix)
  for i in range(leng):
      for j in range(leng):
          for k in range(j,leng):
              result[i][j] += matrix[i][k] * lower_triangular[k][j]
  return result
```

(Ejercicio 8)

Consigna:
¿Encuentra algun inconveniente para analizar las iteraciones while y repeat como recurrencias?
Respuesta:
Si, ya que es mas dificil obtener la cantidad de iteraciones, se requiere observar la funcion que realiza el decrecimiento del while/repeat y por lo menos realizar una iteracion para poder encontrar el factor de decrecimiento.

(Ejercicio 9)

Consigna:
Considerar las matrices $A, B, C \in \mathbb{R}^{(n \times n)}$, y la notacion tal que $X_{i,j}$, con $1 \leq i, j \leq 2$ y X cualquiera de las matrices A, B o C, identifica una de las cuatro submatrices de orden $n/2$.

a) Dar el orden del tiempo de ejecucion del algoritmo DC que se describe con las ecuaciones

$$C_{1,1} = A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1}$$

$$C_{1,2} = A_{1,1} \times B_{1,2} + A_{1,2} \times B_{2,2}$$

$$C_{2,1} = A_{2,1} \times B_{1,1} + A_{2,2} \times B_{2,1}$$

$$C_{2,2} = A_{2,1} \times B_{1,2} + A_{2,2} \times B_{2,2}$$

¿Seria necesario definir algo mas?

b) Buscar el algoritmo de Strassen, dar su definicion en funcion de las submatrices $X_{i,j}$ anteriores y dar su orden de tiempo de ejecucion.

c) Comparar los dos algoritmos de multiplicacion de matrices. ¿Los dos son algoritmos DC? ¿Alguno de los dos es "mejor" que el otro en cuanto

Respuesta:

a) Faltarian definir las llamadas recursivas si el tamaño de la matriz no es 2x2

$DC(A_{1,1}, B_{1,1}) + DC(A_{1,2}, B_{2,1})$
$DC(A_{1,1}, B_{1,2}) + DC(A_{1,2}, B_{2,2})$
$DC(A_{2,1}, B_{1,1}) + DC(A_{2,2}, B_{2,1})$
$DC(A_{2,1}, B_{1,2}) + DC(A_{2,2}, B_{2,2})$

Se puede utilizar la receta $t(n) = at(n/b) + f(n)$
siendo $a = 8$, $b = 2$, $f(n) = n^2$ (ya que es una suma de matrices)
$a > b^k = 8 > 2^2$ por lo que
El orden de ejecucion del algoritmo DC es de $\Theta(n^3)$

b) Strassen es un algoritmo de tipo divide and conquer, define el problema mas sencillo como la multiplicacion de una matriz 2x2 pero reduciendo la cantidad de multiplicaciones entre matrices de 8 a 7, utilizando las siguientes formulas:

$P = (A_{11} + A_{22})(B_{11} + B_{22})$
$Q = (A_{21} + A_{22})B_{11}$
$R = A_{11}(B_{12} - B_{22})$
$S = A_{22}(B_{21} - B_{11})$
$T = (A_{11} + A_{12})B_{22}$
$U = (A_{21} - A_{11})(B_{11} + B_{12})$
$V = (A_{12} - A_{22})(B_{21} + B_{22})$

$C_{11} = P + S - T + V$
$C_{12} = R + T$
$C_{21} = Q + S$
$C_{21} = P + R - Q + U$

c) Los dos son algoritmos DC pero Strassen es mejor en notacion asintotica ya que si utilizamos la receta $t(n) = at(n/b) + f(n)$
siendo $a = 7$, $b = 2$, $f(n) = n^2$
$a > b^k = 7 > 2^2$ por lo que
El orden de ejecucion del algoritmo Strassen es de $\Theta(n^{log_2 7}) = \Theta(n^{2.81})$