

TAREA 3 - Análisis de Algoritmos

*Gabriela N. Gongora Svartzman, Karen L. Poblete Rodríguez,
Salvador B. Medina Maza, Victor R. Martinez Palacios*

Exponenciación

¿Cuál es la forma óptima de calcular x^{77} , x^{195} y x^{631} , con dos posiciones de memoria?
Encontrar un método.

Hint: Uso en Public Key Encryption. El cómputo requerido para descifrar claves públicas, es el cálculo de $x^e \bmod N$, para x , e y N enteros positivos de hasta 1000 dígitos.

Exponenciación

Espacios de memoria = 2

A. x^{77} . $n=8$.

M1	M2
x	
	$x * x = x^2$
	$x^2 * x^2 = x^4$
	$x^4 * x^4 = x^8$
$x^8 * x = x^9$	
	$x^8 * x^9 = x^{17}$
	$x^{17} * x^{17} = x^{34}$
$x^{34} * x^9 = x^{43}$	
	$x^{43} * x^{34} = x^{77}$

Exponenciación

B. x^{195} . $n = 9$

M1	M2
x	
	$x * x = x^2$
$x^2 * x = x^3$	
	$x^3 * x^3 = x^6$
	$x^6 * x^6 = x^{12}$
	$x^{12} * x^{12} = x^{24}$
	$x^{24} * x^{24} = x^{48}$
	$x^{48} * x^{48} = x^{96}$
$x^{96} * x^3 = x^{99}$	
	$x^{99} * x^{96} = x^{195}$

Exponenciación

C. $x^{631} \cdot n = 12$

M1	M2
x	
	$x * x = x^2$
	$x^2 * x = x^3$
	$x^3 * x^3 = x^6$
$x^6 * x = x^7$	
	$x^7 * x^6 = x^{13}$
	$x^{13} * x^{13} = x^{26}$
	$x^{26} * x^{26} = x^{52}$
	$x^{52} * x^{52} = x^{104}$
	$x^{104} * x^{104} = x^{208}$
$x^{208} * x^7 = x^{215}$	
$x^{215} * x^{208} = x^{423}$	
	$x^{423} * x^{208} = x^{631}$

Exponenciación

Algoritmos de Exponenciación:

1) Técnicas Básicas

2) Exponente Fijo

3) Base Fija

Exponenciación Modular

MODULAR-EXPONENTIATION(a, b, n)

```
1   $c = 0$ 
2   $d = 1$ 
3  let  $\langle b_k, b_{k-1}, \dots, b_0 \rangle$  be the binary representation of  $b$ 
4  for  $i = k$  downto 0
5       $c = 2c$ 
6       $d = (d \cdot d) \bmod n$ 
7      if  $b_i == 1$ 
8           $c = c + 1$ 
9           $d = (d \cdot a) \bmod n$ 
10 return  $d$ 
```

Exponenciación Modular

def modularExp(a,b,n):

c=0	%Inicialización de variable c (Representa un espacio de memoria).
d=1	%Inicialización de variable d. (Representa un espacio de memoria).
bk=parseBin(b)	%Valor de exponente en binario.
temp=d	%Variable temporal para ir guardando el valor de a^b

for i in range(len(bk)):

c=2*c	%c por dos.
d=int((d*d)%n)	% Se obtiene el módulo n del cuadrado de d.
temp=temp*temp	%Realiza misma operación que arriba pero % sin el módulo para obtener su valor

if(bk[i]=='1'):	%IF Sí bk(i) == 1 en su representación binaria.
c=c+1	% c se incrementa en 1.
d=int((d*a)%n)	% Se obtiene el módulo n de la multiplicación de "a" con "d".
temp=temp*a	%Realiza misma operación que arriba pero % sin el módulo para obtener su valor

return (temp)	%END IF, END FOR.
----------------------	-------------------

Exponenciación RL binary

Algorithm Right-to-left binary exponentiation

INPUT: an element $g \in G$ and integer $e \geq 1$.

OUTPUT: g^e .

1. $A \leftarrow 1, S \leftarrow g$.
 2. While $e \neq 0$ do the following:
 - 2.1 If e is odd then $A \leftarrow A \cdot S$.
 - 2.2 $e \leftarrow \lfloor e/2 \rfloor$.
 - 2.3 If $e \neq 0$ then $S \leftarrow S \cdot S$.
 3. Return(A).
-

Exponenciación

RL binary

def RLbinary(g,e):

A=1 %Inicializamos el primer espacio de memoria A en 1

S=g %Ponemos en S el valor de la base g.

while (e!=0): %WHILE e (exponente) sea diferente de 0

 if ((e%2)!=0): %IF Modulo de e diferente de 0

 A=A*S % A lo que había en A se multiplica por lo que hay en S.

 e=int(e/2) %e es función piso de e/2.

 if e!=0: %IF e diferente de 0

 S=S*S %S vale su cuadrado.

return (A) %END WHILE y RETURN

Exponenciación RL binary

Ejemplo: 2^{77}

Iteraciones / Operaciones	Iteración	Num Mult.	
$A = A * S = 1 * 2 = 2$ $S = S * S = 2 * 2 = 4$	1	1 2	$e = 77/2 = 38$ Even
$S = 4 * 4 = 16$	2	3	$e = 38/2 = 19$ Odd
$A = 2 * 16 = 32$ $S = 16 * 16 = 256$	3	4 5	$e = 19/2 = 9$ Odd
$A = 32 * 256 = 8192$ $S = 256 * 256 = 65536$	4	6 7	$e = 9/2 = 4$ Even
$S = 65536 * 65536 = 4294967296$	5	8	$e = 4/2 = 2$ Even
$S = 4294967296 * 4294967296 = 18446744073709551616$	6	9	$e = 2/2 = 1$ Odd
$A = 8192 * 18446744073709551616 =$ 151115727451828646838272	7	10	$e = 1/2 = 0$ Stop

Exponenciación LR binary

Algorithm Left-to-right binary exponentiation

INPUT: $g \in G$ and a positive integer $e = (e_t e_{t-1} \cdots e_1 e_0)_2$.

OUTPUT: g^e .

1. $A \leftarrow 1$.
 2. For i from t down to 0 do the following:
 - 2.1 $A \leftarrow A \cdot A$.
 - 2.2 If $e_i = 1$, then $A \leftarrow A \cdot g$.
 3. Return(A).
-

Exponenciación

LR binary

Función auxiliar para manejar números binarios.

```
def parseBin(num):  
    divs=num  
    numB=[]  
    temp=divs  
  
    while (divs!=0):  
        if((temp!=1)&((divs%2)==0)):  
            numB.append(str(0))  
        else:  
            numB.append(str(1))  
  
        temp=divs  
        divs=int(divs/2)  
  
    return numB[::-1]
```

Exponenciación

LR binary

def LRbinary(g,e):

A=1 % Inicialización de variable A.
% (Representa un espacio de memoria).

E=parseBin(e) % Valor de exponente en binario.

for i in range(len(E)): %FOR Para cada valor en E
 A=A*A %A es igual a A al cuadrado.

 if (E[i]=='1'): %IF Sí E(i) == 1 en su representación binaria.
 A=A*g %A es igual a A*g, g siendo la base

return (A) %END IF, END FOR.

Exponenciación LR binary

Ejemplo: 2^{77} , E = 1001101

Iteraciones / Operaciones	Rep. Binaria	Num Multiplicación
$A * A = 1 * 1 = 1$ $A = A * g = 1 * 2 = 2$	1	1 2
$A = 2 * 2 = 4$	0	3
$A = 4 * 4 = 16$	0	4
$A = 16 * 16 = 256$ $A = 256 * 2 = 512$	1	5 6
$A = 512 * 512 = 262144$ $A = 262144 * 2 = 524288$	1	7 8
$A = 524288 * 524288 = 274877906944$	0	9
$A = 274877906944 * 274877906944 = 75557863725914323419136$ $A = A * 2 = 151115727451828646838272$	1	10 11

Exponenciación

Resultados

LR BINARY

TIME: 24 microseconds

Result: 151115727451828646838272

Number of mult = 11

Add Chain = [0, 1, 2, 4, 8, 9, 18, 19, 38, 76, 77]

RL BINARY

RLbinary 151115727451828646838272

TIME: 16 microseconds

mults 10

Add Chain = [1, 2, 4, 5, 8, 13, 16, 32, 64, 77]

Modular Exponentiation

TIME: 31 microseconds

Result: 151115727451828646838272

Number of mult = 11

Add Chain = [0, 1, 2, 4, 8, 9, 18, 19, 38, 76, 77]

Exponenciación

Resultados

RL BINARY

TIME: 16 microseconds

Result:

50216813883093446110686315385661331328818843555712276103168

Number of mult = 11

Add Chain = [1, 2, 3, 4, 8, 16, 32, 64, 67, 128, 195]

Modular Exponentiation

TIME: 31 microseconds

Result:

50216813883093446110686315385661331328818843555712276103168

Number of mult = 12

Add Chain = [0, 1, 2, 3, 6, 12, 24, 48, 96, 97, 194, 195]

LR BINARY

TIME: 26 microseconds

Result:

50216813883093446110686315385661331328818843555712276103168

Number of mult = 12

Add Chain = [0, 1, 2, 3, 6, 12, 24, 48, 96, 97, 194, 195]

Exponenciación

Resultados

Modular Exponentiation

TIME: 41 microseconds

Result:

891101683129335003640853829238338149393208692821984361441248538652202181095444802051936095960424
1015192660760885926576778688876408936402340337229140082449586429677098359892480630613656731648

Number of mult = 17

Add Chain = [0, 1, 2, 4, 8, 9, 18, 19, 38, 39, 78, 156, 157, 314, 315, 630, 631]

LR BINARY

TIME: 32 microseconds

Result:

891101683129335003640853829238338149393208692821984361441248538652202181095444802051936095960424
1015192660760885926576778688876408936402340337229140082449586429677098359892480630613656731648

Number of mult = 17

Add Chain = [0, 1, 2, 4, 8, 9, 18, 19, 38, 39, 78, 156, 157, 314, 315, 630, 631]

RL BINARY

TIME: 20 microseconds

Result:

891101683129335003640853829238338149393208692821984361441248538652202181095444802051936095960424
1015192660760885926576778688876408936402340337229140082449586429677098359892480630613656731648

Number of mult = 16

Add Chain = [1, 2, 3, 4, 7, 8, 16, 23, 32, 55, 64, 119, 128, 256, 512, 631]

Exponenciación

Resultados

Exponente	x^{77}	x^{195}	x^{631}
LRbinary			
Tiempo (microsegundos)	24	26	32
Num. Mult	11	12	17
RLbinary			
Tiempo (microsegundos)	16	16	20
Num. Mult	10	11	16
Modular Exp			
Tiempo (microsegundos)	31	31	41
Num. Mult	11	12	17

Producto de Cadenas Matriciales

Producto de cadena de matrices. Dada las matrices $A_1 \times A_2 \times \dots A_n$, donde las matrices tienen dimensiones compatibles - pero no son cuadradas - ¿en qué orden deben ser multiplicadas para minimizar el tiempo de computación?

$$A_1 = 100 \times 4$$

$$A_2 = 4 \times 50$$

$$A_3 = 50 \times 20$$

$$A_4 = 20 \times 100$$

Exponer ejemplos, mejor y peor caso, y describir el método empleado.

Programación Dinámica

1. Caracterizar la estructura de una solución óptima
2. Definir recursivamente el valor de una solución óptima
3. Computar el valor de la solución óptima
4. Construir una solución óptima de la información computada

Caracterizar la estructura

- $A_{i...j}$ matriz resultante del producto $A_i A_{i+1} \dots A_j$
- $A_{i...j}$ se puede dividir en $A_{i...k}$ y $A_{k+1...j}$
- El costo es:

$$C(A_{i...j}) = C(A_{i...k}) + C(A_{k+1...j}) + C(A_{i...k} \times A_{k+1...j})$$

$C(A_{i...k})$: Costo de computar $A_{i...k}$

$C(A_{k+1...j})$: Costo de computar $A_{k+1...j}$

$C(A_{i...k} \times A_{k+1...j})$: Costo del producto entre $A_{i...k}$ y $A_{k+1...j}$

- Se divide el problema en 2 subproblemas

Optimizar $C(A_{i...k})$ y Optimizar $C(A_{k+1...j})$

Solución recursiva

Subproblemas

Determinar la posición k del paréntesis para obtener el costo mínimo de $A_i A_{i+1} \dots A_j$

Acercamiento

Solución bottom-up tabular

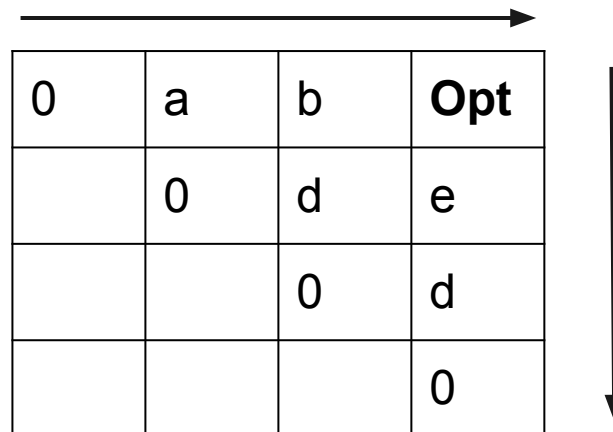
Solución recursiva

Tabla

- Usamos matriz m
- $m[i, j]$ almacena mínimo de multiplicaciones escalares

Por lo tanto, buscamos $m[1, n]$

- $i = j$ es un caso trivial $A_{i \dots i} = A_i$



0	a	b	Opt
	0	d	e
		0	d
			0

Solución recursiva

- $m[i, j] = C(A_{i\dots k}) + C(A_{k+1\dots j}) + C(A_{i\dots k} \times A_{k+1\dots j})$
- Vector p define dimensiones de las matrices
- A_i tiene dimensiones $p_{i-1} \times p_i$
- número de operaciones de $A_{i\dots k} A_{k+1\dots j}$ es $p_{i-1} p_k p_j$

$$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$$

Solución recursiva

$k = ???$

$$k = i, i+1, \dots, j-1$$

Solución:

Evaluar los valores de k

Definición:

$$m[i, j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & \text{if } i < j. \end{cases}$$

Datos extra

En una matriz s se almacenará el valor de k

$$s[i,j] = k$$

para el valor mínimo encontrado de dividir $A_i A_{i+1} \dots A_j$

Producto de Cadenas Matriciales

def MatrixChain(p):

n=len(p)-1

m=[[0 for x in range(0, n+1)] for y in range(0,n+1)]

s=[[0 for x in range(0, n+1)] for y in range(0,n+1)]

for L in range(2, n+1):

for i in range(1, n-L+2):

j = i+L-1

m[i][j] = sys.maxsize

for k in range(i, j-1+1): # check all splits

q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j]

if q < m[i][j]:

m[i][j] = q

s[i][j] = k

return m, s

%Establece el tamaño del arreglo de dimensiones

%Inicialización de matriz de óptimos en 0

%Inicialización de matriz de posiciones en 0

%FOR (L controla la longitud del producto de mtz)

%FOR (i depende del valor de L)

% Inicializa en un valor max, espacio actual de matriz.

%FOR k de i a j. Se evalúan todas las posiciones

% Calcula el costo de cálculo

%IF el costo es menor que el almacenado

% Se actualiza el costo mínimo

% Se guarda la posición

%END IF, END FOR, END FOR, END FOR, RETURN

Resultado

$$A_1 = 100 \times 4$$

$$A_2 = 4 \times 50$$

$$A_3 = 50 \times 20$$

$$A_4 = 20 \times 100$$

El valor inicializado del vector con las dimensiones: $p1=[100,4,50,20,100]$.

El resultado óptimo es para este caso:

$(A_1 ((A_2 A_3) A_4))$

El número de multiplicaciones es: 52000

Mejor Caso

m =

0	20000	12000	52000
	0	4000	12000
		0	100000
			0

s =

0	1	1	1
	0	2	3
		0	3
			0

$$A_1((A_2 A_3) A_4)$$

Producto de Cadenas Matriciales

```
def printParens(s,i,j):  
    if i == j:  
        print("A%d"%i, end="")  
    else:  
        print ("(", end="")  
        k = s[i][j]  
        printParens( s, i, k)  
        printParens( s, k+1, j )  
        print (")", end="")
```

(A) (B)

Impresión Recursiva

(A1 ((A2A3)A4))

i=1 j=4 level=1 [(
i=1 j=1 level=1 [A1]
i=2 j=4 level=2 [(
i=2 j=3 level=3 [(
i=2 j=2 level=3 [A2]
i=3 j=3 level=3 [A3]
i=2 j=3 level=3 [)]
i=4 j=4 level=2 [A4]
i=2 j=4 level=2 [)]
i=1 j=4 level=1 [)]

0	1	1	1
	0	2	3
		0	3
			0

Impresión Recursiva

(A1 ((A2A3)A4))

i=1 j=4 level=1 [(

i=1 j=1 level=1 [A1]

i=2 j=4 level=2 [(

i=2 j=3 level=3 [(

i=2 j=2 level=3 [A2]

i=3 j=3 level=3 [A3]

i=2 j=3 level=3 [)]

i=4 j=4 level=2 [A4]

i=2 j=4 level=2 [)]

i=1 j=4 level=1 [)]

0	1	1	1
	0	2	3
		0	3
			0

Primer nivel

(A1 ((A2A3)A4))

i=1 j=4 level=1 [(]

i=1 j=1 level=1 [A1]

i=2 j=4 level=2 [(]

i=2 j=3 level=3 [(]

i=2 j=2 level=3 [A2]

i=3 j=3 level=3 [A3]

i=2 j=3 level=3 []]

i=4 j=4 level=2 [A4]

i=2 j=4 level=2 []]

i=1 j=4 level=1 []]

0	1	1	1
	0	2	3
		0	3
			0

Segundo nivel

(A1 ((A2A3)A4))

i=1 j=4 level=1 [(

i=1 j=1 level=1 [A1]

i=2 j=4 level=2 [(

i=2 j=3 level=3 [(

i=2 j=2 level=3 [A2]

i=3 j=3 level=3 [A3]

i=2 j=3 level=3 [)]

i=4 j=4 level=2 [A4]

i=2 j=4 level=2 [)]

i=1 j=4 level=1 [)]

0	1	1	1
	0	2	3
		0	3
			0

Tercer nivel

(A1 ((A2A3)A4))

i=1 j=4 level=1 [(

i=1 j=1 level=1 [A1]

i=2 j=4 level=2 [(

i=2 j=3 level=3 [(

i=2 j=2 level=3 [A2]

i=3 j=3 level=3 [A3]

i=2 j=3 level=3 [)]

i=4 j=4 level=2 [A4]

i=2 j=4 level=2 [)]

i=1 j=4 level=1 [)]

0	1	1	1
	0	2	3
		0	3
			0

Producto de Cadenas Matriciales

def MatrixChain(p):

n=len(p)-1

m=[[0 for x in range(0, n+1)] for y in range(0,n+1)]

s=[[0 for x in range(0, n+1)] for y in range(0,n+1)]

for L in range(2, n+1):

for i in range(1, n-L+2):

j = i+L-1

m[i][j] = -1

for k in range(i, j-1+1): # check all splits

q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j]

if q >= m[i][j]:

m[i][j] = q

s[i][j] = k

return m, s

%Establece el tamaño del arreglo de dimensiones

%Inicialización de matriz de óptimos en 0

%Inicialización de matriz de posiciones en 0

%FOR (L controla la longitud del producto de mtz)

%FOR (i depende del valor de L)

% Inicializa en un valor max, espacio actual de matriz.

%FOR k de i a j. Se evalúan todas las posiciones

% Calcula el costo de cálculo

%IF el costo es menor que el almacenado

% Se actualiza el costo mínimo

% Se guarda la posición

%END IF, END FOR, END FOR, END FOR, RETURN

Peor Caso

m =

0	20000	12000	620000
	0	4000	120000
		0	100000
			0

s =

0	1	2	2
	0	2	2
		0	3
			0

$(A_1 \ A_2)(A_3 \ A_4)$

¡GRACIAS!