Análisis de Algoritmos, Sem: 2021-2, 3CV11, Práctica 1, Fecha: 17/03/2021

Práctica 1: Determinación experimental de la complejidad temporal de un algoritmo

Muñoz Carbajal Carlos Eduardo, Villalobos Aceves Jonathan Jesus.

cmunozc1500@alumno.ipn.mx, jvillalobosa1500@alumno.ipn.mx

Resumen: A lo largo de la experiencia que hemos adquirido en el desarrollo de programas para la resolución de problemáticas dentro de las diversas unidades de aprendizajes contempladas en nuestra trayectoria escolar, nunca se había profundizado en la comparación de los diversos algoritmos que se empleaban para la solución de un mismo conflicto, esto solo se mencionaba o se podía observar cuando algún programa informático contaba con una menor cantidad de líneas de código o en el tiempo de ejecución en una misma computadora. A continuación se llevara a cabo el desarrollo de dos ejercicios prácticos en los cuales se emplearan el análisis algorítmico para entender la complejidad que se presenta en cada uno de los ejercicios.

Palabras Clave: Lenguaje C++, Algoritmo y Complejidad .

1 Introducción

Antes de comenzar con el desarrollo de la presente práctica, es necesario comenzar colocando las bases de los temas que se abordara a continuación, ya que para la mayoría de personas o de estudiantes que comienzan con el desarrollo de programas informáticos para así ayudarse a la resolución de sus tareas diarias, les es indiferente los conceptos sobre que es en realidad un programa informático, la diferencia entre el desarrollo de un programa que brinde una solución eficaz o uno que brinde una solución eficiente, además de que mientras dichos desarrollos cumplan con su función para la mayoría de personas es suficiente y no es necesario el saber si la solución propuesta consta de una estructura realmente funcional.

"Un algoritmo, . . . , es sencillamente un conjunto de reglas para efectuar algún calculo, bien sea a mamo, o más frecuentemente, en una maquina". (Brassard, 1997)

Los humanos desde que comenzamos a llevar a cabo procesos escolares nos vemos inversos en el uso de algoritmos para el proceso de nuestro aprendizaje y la resolución de problemas, ahora bien desde antes de la formalización de

las ciencias computacionales, se estaba llevando a cabo una exhaustiva investigación por parte de matemáticos de la época para poder así establecer el concepto de lo que era un algoritmo, y fue hasta que se formalizaron las ciencias computacionales que se plantean en libros y artículos relacionados a esta ciencia, definiciones como la citada anteriormente. Ahora bien, antes se mencionó que un algoritmo (programa informático) podría brindar una solución eficaz o eficiente, pero ¿a que hace referencia cada una de estas? Para hablar de esto entran en juego tres definiciones: algoritmo correcto, algoritmo eficaz y algoritmo eficiente. Nosotros decimos que un algoritmo es correcto, si este siempre nos brinda la respuesta correcta, un algoritmo es eficaz si es posible encontrar una respuesta correcta a nuestro problema y un algoritmo es eficiente si realiza su trabajo con la menor cantidad de recursos. Con lo anterior ya nos podemos dar una idea de que cada algoritmo puede contar con una serie de características que lo puedan hacer más útil o favorable en alguna situación con caracterśticas especiales, por lo que ahora es necesario saber el como separar a los algoritmos según sea el caso. Lo primordial a la hora de llevar a cabo un desarrollo, en general, es el tiempo que le tomo a un algoritmo desarrollar sus procesos, por lo cual en esta práctica se hará uso del análisis algorítmico para poder observar las características de los siguientes ejercicios.

2 Conceptos Básicos

A continuación colocaremos las notaciones que se emplean en el análisis algorítmico.

Notación θ: Dada una funcón g(n), θ(g(n)) denota el conjunto de las funciones definidas como:

θ (g(n)) = { f(n): $\exists_n$ $C_0, C_2 > 0$ $n_0 > 0$ tal que $0 \leqslant C_0 g(n) \leqslant$ f(n)$\leqslant$ $C_2 g(n)$ $\forall\, n \geqslant n_0$}

Notació $\mathcal{O}$: Dada un función g(n), $\mathcal{O}$(g(n)) denoma el conjunto de las funciónes definidas como:

$\mathcal{O}$ (g(n)) = { f(n): $\exists_n$ $C > 0$ $n_0 > 0$ tal que $0 \leqslant$ f(n)$\leqslant C g(n)$ $\forall\, n \geqslant n_0$}

Notació Ω: Dada un función g(n), Ω(g(n)) denoma el conjunto de las funciónes definidas como:

Ω (g(n)) = { f(n): $\exists_n$ $C > 0$ $n_0 > 0$ tal que $0 \leqslant$ Cg(n)$\leqslant f(n)$ $\forall\, n \geqslant n_0$}

Ejercicio 1 Suma binaria. Como sabemos el sistema binario esta compuesto por dos símbolos el '0' y el '1', por lo cual un numero binario está conformado por n 0's o 1's seguido de m 0's o 1's; cundo sumamos números binarios necesitamos de un elemento auxiliar que nos ayude a guardar el resultado de la suma de dos 1's, este elemento es denominado comunmente acarreo, esto se explica de la siguiente manera:

A	B	acarreo	Resultado
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 1: Comportamiento de una suma binaria

Como vemos el valor del acarreo de una suma binaria es lo que va a permitir a nuestro algoritmo pasar de un número de n bits a uno de n+1 bits.Para el algoritmo en cuestion tendremos que manter en mente que la mayor parte del tiempo la suma se dara entre tres elementos que son: el valor del elemento A, el del elemento B y el valor de acarreo. Para nuestro programa necesitamos encontrar una ecuación que sea computable y que permita obtener el valor de resultado, está la buscaremos en base al siguiente tabla, cabe aclarar que el primer valor de acarreo es el valor con el que se inicia la suma y el segundo valor de acarreo es el que se obtiene con el resultado:

A	B	acarreo	Resultado	acarreo
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 2: Suma binaria para el algoritmo

A partir de la tabla podemos obtener dos expresiones que nos modelen el comportamiento del valor resultado y del valor del acarreo respectivamente. En

el caso del valor del resultado tenemos que obtenemos un valor de 1 cuando existen números impares de 1, lo anterior es modelado por la operación de modulo 2 aplicado a nuestros 3 valores de entrada; en el caso del acarreo de salida observamos que tendremos un valor de uno para cuando B y el acarreo ambos 1 o para A y acarreo ambos uno o para A y B ambos uno o para A, B y acarreo todos uno, como el ultimo está contenido en los tres primeros solo consideraremos los tres primeros casos.

Pseudo-codigo
Suma (longitud, A, B)
resultado[longitud+1]
for i=0 **to** longitud
resultado[i+1]=(A[i+1]+B[i+1]+acarreo)mod2
acarreo=(A[i+1] and B[i+1]) or (A[i+1] and acarreo) or (B[i+1] and acarreo)
resultado[0]=acarreo.

Ejercicio 2 Algoritmo de Euclides. El propósito principal que tiene este algoritmo es hallar el valor de el maximo cómun divisor de dos números, para ello se toma el número mayor y lo divide entre el menor, en caso de que la división sea exacta (que el residuo sea cero) quiere decir que el divisor es el m.c.d, en caso contrario se divide el divisor entre el resto de la división anterior, éste procedimiento se repite hasta obtener una división exacta. Dado los pasos marcados anteriormente se tiene el siguiente pseudocódigo.
Euclides(m,n):

while n != 0 do
r = m mod n
m = n
n = r
return m

En el pseudocódigo anteriors se considera a r como el residuo, a m como el dividendo y a n como el divisor que abarcaría el valor del m.c.d dentro de la varíable m en el ultimo adentramiento del loop.

3 Experimentación y Resultados

3.1 Suma binaria

Ejecución A continuación se muestra la ejecución del programa desarrollado en lenguaje C++.

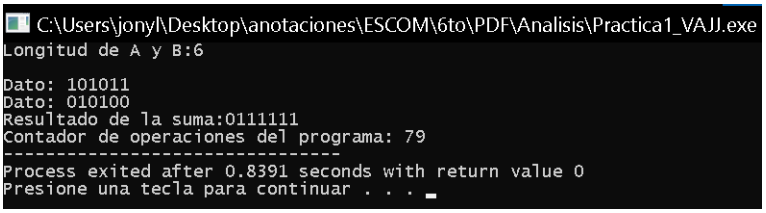

Figure 1: Ejecución del programa SumaDeBits.cpp

Analisis a posteriori. Ahora bien procederemos a generar la tabla de valores para la ejecución de este programa y así poder generar los elementos de analisis necesarios.

Longitud	Operaciones
1	19
2	31
4	55
8	103
32	391
64	775
256	3079
1024	12295
4096	49159
16384	196615
32768	393223
65536	786439
131072	1644871
262144	3145735
524288	6291463
1048576	12582919

Table 3: Datos del algoritmo de suma

Graficas Para concretar nuestro análisis es necesario observar el cómo se distribuyen los puntos del algoritmo en un gráfico. Dentro del comportamiento del algoritmo se observa que los puntos de t sobre r se disparan de una forma acelerada, observamos que para un valor relativamente menor como lo es r=524288, el valor de t llega a los millones con t=6291463.

Analisis Para comenzar a acotar dichos puntos haremos el calculo de la recta que contienen múltiples puntos, para r=1 t=19r, para r=2 t=15.5r, para r=64

t=12.1093...r, para r=256 t=12.0273...r, para r=1048576 t=12.00000668r; a partir de lo anterior podemos darnos cuenta que el mínimo valor contenido debe de ser el 19, de modo que la recta que acota a nuestro algoritmo para r=1 t mayor a 19, y el ultimo dato que debe ser acotado (dato medido) debe ser t mayor a 12582919, de modo que una gráfica que acota nuestro algoritmo es t= 20r; de modo que $T(n) \in \Theta(n)$ al graficar ambos juntos tenemos:

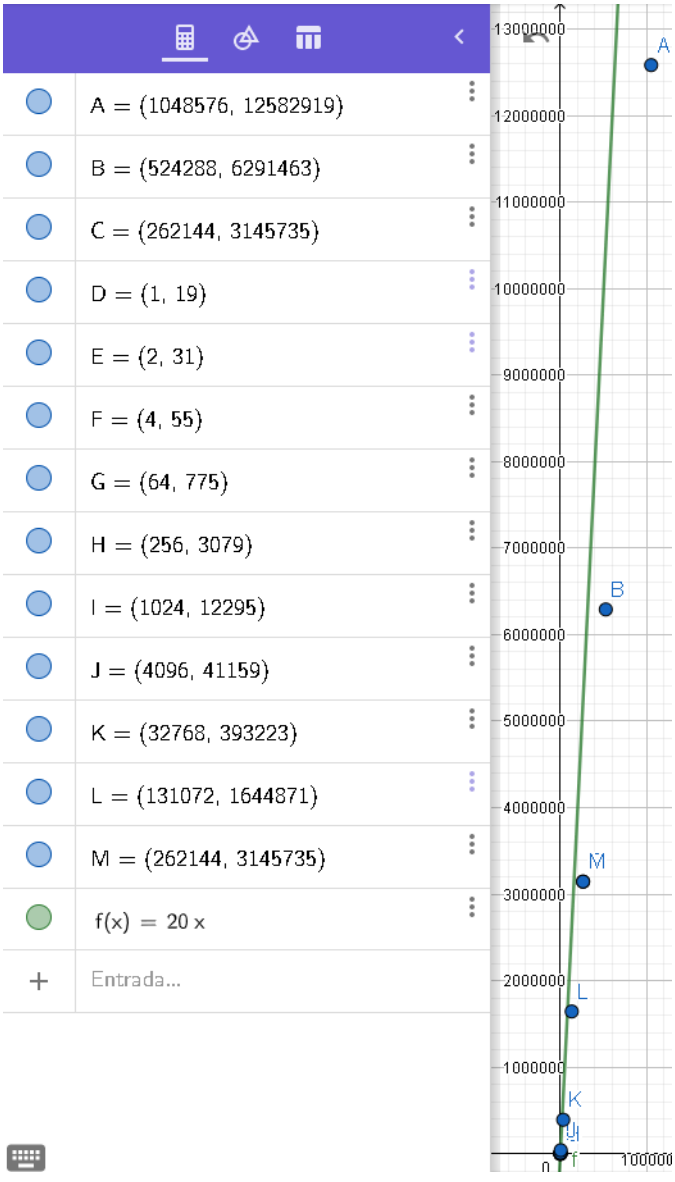

Figure 2: Datos en gráfica

3.2 Algoritmo de Euclides

Para visualizar el comportamiento de éste problema se realizó análisis a posteriori, esto quiere decir que incluímos un contador por cada línea ejecutada en el código de acuerdo a los diferentes valores de entrada; cabe mencionar que para obtener una variación notoria se trabajaron números consecutivos de la sucesión de Fibonacci, por lo que se nos arroja los datos mostrados en la siguiente tabla.

n (Número mayor)	Número de pasos
2	10
3	14
5	18
8	22
13	26
21	30
34	34
55	38
89	42

Table 4: Resultados obtenidos del algoritmo de Euclides

Visualizando la información se observa que por cada número mayor de acuerdo a los números consecutivos de la sucesión de Fibonacci se aprecia que se tiene un incremento de cuatro en cuatro para cada resultado de este algoritmo. De acuerdo a los resultados se obtiene la siguiente gráfica.

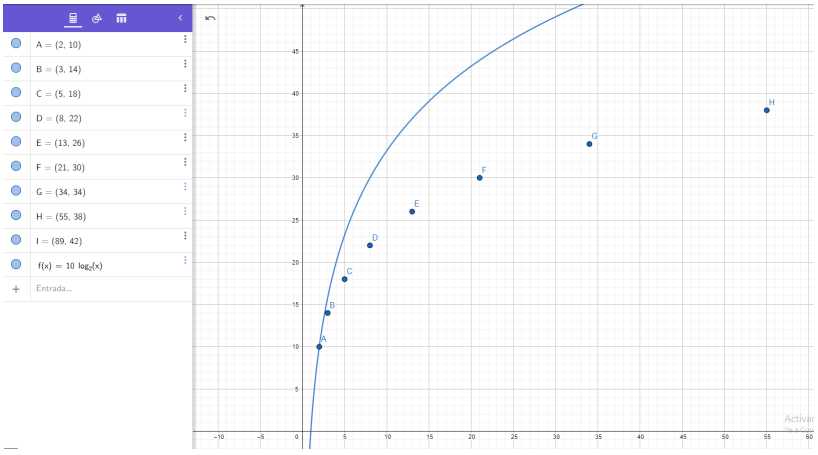

Figure 3: Comportamiento del algoritmo de Euclides

Como se puede observar el algoritmo se asemeja a un comportamiento Logaritmico, y es por ello que se propone la función g(n) = 10Log n, pues de acuerdo a la definición de $\mathcal{O}$ debe presentarse acotada dicha función por encima, y como se puede apreciar en la gráfica utilizando una constante C de 10, éste cumple con la definición para $n_0 \geq 3$.

4 Conclusiones

Conclusión general A lo largo de esta práctica se da un vistazo a como funciona el análisis de la complejidad, ya que con la ayuda de estos procedimientos, podremos definir si es viable o no usar una de varias posibles formas de solucionar un problema utilizando medios computacionales y lenguajes de programación. A pesar de ser un analisis a posteriori podemos notar que el determinar una función puede llegar a ser una tarea compleja de acuerdo a sus entradas como lo es en el ejercicio dos que se analizó de acuerdo a la sucesión de Fibonacci, las entradas depedenden de los dos números vistos anteriormente y pude resultar no tan sencillo llevar a cabo un análisis teórico, sin embargo, trabajandolo como en esta práctica es posible llegar a una función que cumpla con lo solicitando siempre y cuando se especifiquen los valores de las constantes y para n_0. Para finalizar suponemos que el hecho de llevar un análisis de algún algoritmo es de gran importancia ya que trabajar con ellos a futuro puede requerir optimizar lo mejor posible alguna solución a problemas complejos que exigan una basta cantidad de recursos y tiempo, para ello la importancia de esta labor.

Conclusiones Alumno 1 De acuerdo al desarrollo visto en la práctica, en algunas ocasiones puede ser más viable utilizar el análisis a posteriori, tal es el caso de la segunda parte, pues para poder calcular su complejidad puede verse como una tarea muy difícil ya que tenemos valores muy variados para cada una de las entradas, entonces es necesario encontrar un patrón correcto para visualizar variacianos dentro del tiempo que aborda ejecutarse el programa ,sin embargo, teniendo valores que varían de algún modo especial como es el caso de Fibonacci puede ser una tarea muy compleja, ya que el resultado depende de un número anterior y por ello considero de alguna manera más viable trabajar problemas de este estilo con a posteriori para así estimar un orden de complejidad. Por otra parte, en otros casos puede verse más factible hacer un analísis teórico (a priori) ya que de esta forma tendemos a obtener resultados precisos.

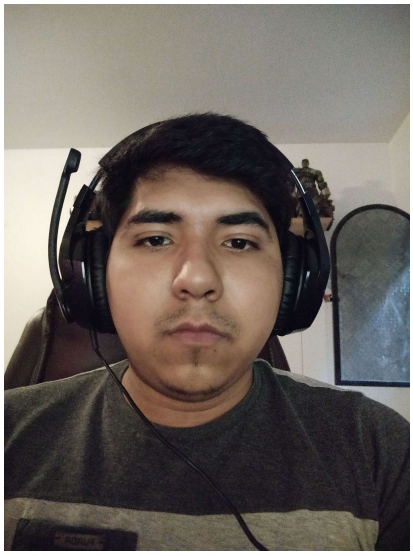

Figure 4: Alumno 1

Conclusiones Alumno 2 Dentro del desarrollo del algoritmo de suma me encontré con que las soluciones más factibles analíticamente hablando, son más complicadas de codificar, por lo cual es mas factible el seguir el procedimiento que se maneja para le desarrollo de u algoritmo, sin embargo como primera impresión del uso y análisis de esta metodología me fue necesario el buscar información extra que se asemejara para la resolución del algoritmo correspondiente, esta búsqueda extra te hace dilucidar que al final lo que mas predomina es aquello para lo cual tienes un mayor entendimiento, la resolución o creación de un software que realice sumas binarias es aparentemente muy estudiado y posee una cantidad de alternativas de solución que van desde el camino más básico la aplicación de caracteres y desarrollar un proceso de comparación de elementos, hasta el uso de librerías especificas las cuales requieren de llevar un estudio a fondo de las mismas. En general diría que el analisis realizado es bastante pobre pero es una gran experiencia que puede sentar las bases de la asimilación que es necesaria en esta unidad de aprendizaje.

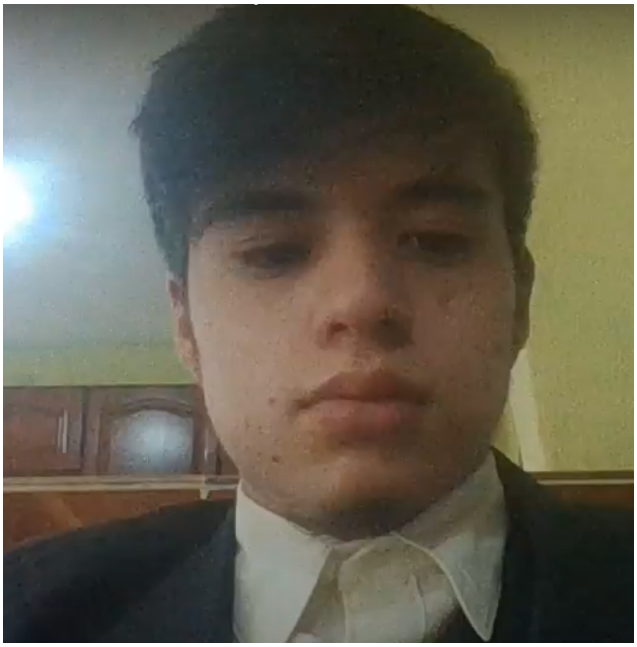

Figure 5: Alumno 2

5 Bibliografía

Baase, V. G. (2001). Algoritmos computacionales. México: Pearson.
Brassard, G. (1997). Fundamentos de Algoritmia. España: Prentice Hall.