UTEM

Universidad Tecnológica Metropolitana

Informe Análisis de Algoritmos
Serie Fibonacci

Autores:
Benjamin Vargas
Eduardo Osorio

Profesor:
Dr. Adrían Jaramillo
Correo: adrian.jaramillo@utem.cl

19 de diciembre de 2018

Índice

Lista de Tablas

Lista de Figuras

1. Introducción

Un algoritmo es considerado eficiente si su consumo de recursos está en la media o por debajo de los niveles aceptables. Hablando a grandes rasgos, 'aceptable' significa: que el algoritmo corre en un tiempo razonable en una computadora dada. Desde 1950 hasta la actualidad las computadoras han tenido un avance impresionante tanto en poder computacional como en la capacidad de memoria disponible, lo que indica que los niveles aceptables de eficiencia en la actualidad hubieran sido inadmisibles 10 años atrás.

Existen muchas maneras para medir la cantidad de recursos utilizados por un algoritmo: las dos medidas más comunes son la complejidad temporal y espacial; otras medidas a tener en cuenta podrían ser la velocidad de transmisión, uso temporal del disco duro, así como uso del mismo a largo plazo, consumo de energía, tiempo de respuesta ante los cambios externos, etc. Muchas de estas medidas dependen del tamaño de la entrada del algoritmo (cantidad de datos a ser procesados)

En este experimento informático, realizaremos la implementación algorítmica de la sucesión matemática fibonacci de forma lineal y recursiva. utilizaremos C++ como lenguaje de programación y CodeBlocks como IDE.

Obtendremos resultados experimentales que provienen de la ejecución de ambos algoritmos, evaluaremos su complejidad temporal analizando por caso. Para ello se probarán 31 casos por elementos introducidos a la función lineal y recursiva.

Luego analizaremos teóricamente los resultados finales de cada función aplicando los conocimientos visto en clases, pudiendo discernir qué algoritmo es mejor o menos complejo según su complejidad temporal.

2. Desarrollo

2.1. Conceptos previos

Algoritmo

Es la descripción unívoca y finita de la secuencia de acciones a ejecutar para resolver un problema.

Unívoca: esto significa que luego de ejecutada una determinada acción, la siguiente está indicada sin ambigüedades. Si después de ejecutada una acción existen dos o más que podrían ejecutarse y no existe un criterio para seleccionar la que corresponde, no es un algoritmo.

Finita: se refiere a que la secuencia de acciones debe finalizar en algún momento, cuando el problema esté resuelto. Una secuencia de acciones que podría llegar a ejecutarse indefinidamente, no es un algoritmo.

Para resolver un problema: Antes de desarrollar un algoritmo, debemos conocer el problema que se quiere resolver, es decir, debemos haber hecho la especificación del mismo.

Programación Lineal

Es el campo de la programación matemática dedicado a maximizar o minimizar (optimizar) una función lineal, denominada función objetivo, de tal forma que las variables de dicha función estén sujetas a una serie de restricciones expresadas mediante un sistema de ecuaciones o inecuaciones también lineales.

Programación Recursiva

Es la forma en la cual se especifica un proceso basado en su propia definición, la recursión tiene ésta característica discernible en términos de autorreferencialidad. En programación, un método usual de simplificación de un problema complejo es la división de este en subproblemas del mismo tipo. Esta técnica de programación se conoce como divide y vencerás y es el núcleo en el diseño de numerosos algoritmos de gran importancia. El ejemplo más común, es el cálculo factorial recursivo de un número.

Serie de fibonacci

La serie de fibonacci o sucesión de fibonacci, es una sucesión matemática infinita. Consta de una serie de números naturales que se suma a dos a partir de 0 y 1 ,Básicamente, la sucesión de fibonacci se realiza sumando siempre los 2 últimos números(Todos los números presentes en la sucesión se llaman números de fibonacci) de la siguiente manera:

- 0,1,1,2,3,5,8,13,21,34...

Es decir (0+1=1 / 1+1=2 / 1+2=3 / 2+3=5 / 3+5=8 / 5+8=13 / 8+13=21 / 13+21=34...) y así hasta el infinito. De tal forma que podemos obtener las siguientes expresiones.

$$f(0) = 1$$

$$f(n) = f(n-1) + f(n-2), \forall n \in R^+$$

Complejidad temporal

La complejidad temporal es un término usado para hablar sobre los recursos requeridos durante el cómputo para resolver un problema. Estos pueden ser dos:

- Tiempo: Número de pasos de un algoritmo para resolver un problema.
- Espacio: Cantidad de memoria utilizada para resolver el problema.

La complejidad es la relación entre n y el número de pasos. Se refiere al ratio de crecimiento de los recursos con el tamaño de la entrada:

- Del tiempo de ejecución (complejidad temporal): T(n).
- Del espacio de almacenamiento necesario (complejidad espacial): S(n).

La complejidad se determina por el análisis asintótico del coste y es relativa a la peor instancia. Para el análisis asintótico del coste se emplea la cota superior asintótica (O).

Para este informe solo nos concentramos en la complejidad temporal.

Orden de Complejidad

En el estudio teórico de un algoritmo, lo normal es estimar su complejidad de forma asintótica, usa notación O grande para representar la complejidad de un algoritmo como una función que depende del tamaño de la entrada n, esto es generalmente acertado cuando n es lo suficientemente grande, pero para n pequeños podría ser erróneo.

En análisis de algoritmos una cota superior asintótica es una función que sirve de cota superior de otra función cuando el argumento tiende a infinito. Usualmente se utiliza la notación Landau: O(g(x)), Orden de g(x), coloquialmente llamada Notación O Grande, para referirse a las funciones acotadas superiormente por la función g(x).

2.2. Explicación del experimento

En este trabajo se implementa un algoritmo tanto recursivo y lineal para la ejecución de la sucesión de fibonacci con el fin de observar el cambio de los tiempos de ejecución que requieren cada una en sus iteraciones. Para ello se probaran 31 casos por elementos introducidos a la función implementada.

2.2.1. Recursos utilizados en el experimento

Para la implementación de los algoritmos se utilizara el lenguaje c++ usando el IDE de codeblocks versión 17.12, utilizando un computador con un sistema operativo windows 10 con una arquitectura de 64 bit, con 8GB de memoria ram y un procesador intel pentium quad core processor N3540 de 2.66 GHz.

Para medir el tiempo de ejecución del programa se utilizo la libreria time.h ,la función clock().

2.3. Implementación lineal

Para la el desarrollo de la implementación lineal de la sucesión de fibonacci se utilizó la siguiente en c++:

```
long long l_fibonacci(long long n){
    long long x=0,y=1,z=1;

    if(n == 0)
        return n;
    if(n == 1 || n == 2)
        return 1;

    for(long long i= 1; i<n; i++){
        z = x+y;
        x = y;
        y = z;
    }
    return z;
}
```

Figura 1: implementación función lineal

2.3.1. Resultados obtenidos de la implementación lineal

De la implementación lineal de la sucesión de fibonacci aplicada a los 31 casos con n= 5,10,15,20,25,30,35,40,45,50, se obtiene a partir del cálculo de la mediana en cada caso la siguiente tabla:

n	S (mediana de Tiempo expresado en microsegundos)
5	0.001
10	0.001
15	0.003
20	0.003
25	0.005
30	0.005
35	0.007
40	0.007
45	0.009
50	0.009

Tabla 1: Resultados implementación lineal fibonacci

2.3.2. Gráfico Obtenido de implementación lineal

A partir de los resultados obtenidos en la sección anterior (sección 2.3.1) se obtiene la siguiente curva.

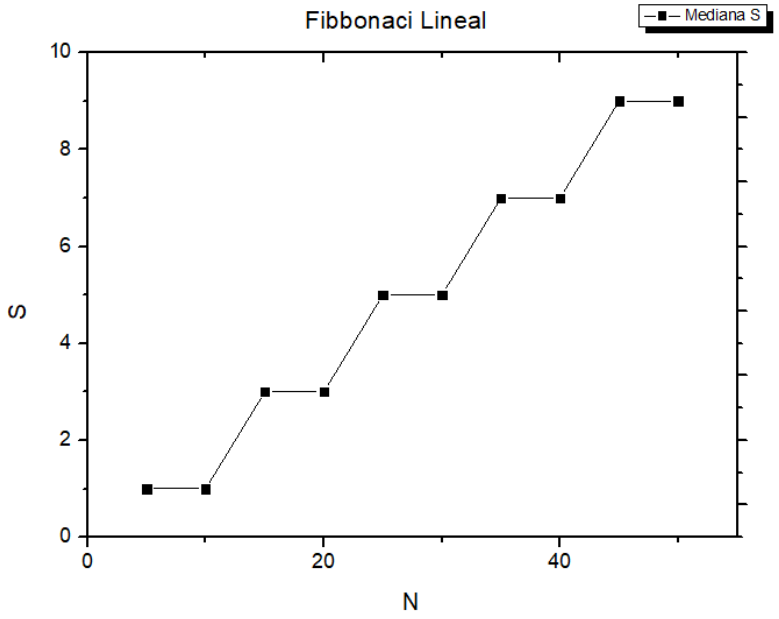

Figura 2: Gráfico función lineal

2.3.3. Análisis de resultados de implementación lineal

Tras ver el gráfico de la sección 2.3.2 hemos podido ver que la función se ejecuta las n veces con tiempo igual a 0.001s las 31 veces de los n casos, lo que nos dice que la complejidad temporal de este algoritmo es una función lineal que pertenece a un orden de complejidad de O(n).

2.4. Implementación recursiva

Para la el desarrollo de la implementación recursiva de la sucesión de fibonacci se utilizó la siguiente en c++:

```
long long r_fibonacci(long long n)
{
    if(n == 0 || n == 1)
        return n;
    else
        return (r_fibonacci(n - 1) + r_fibonacci(n - 2));
}
```

Figura 3: implementación función Recursiva

2.4.1. Resultados obtenidos de la implementación Recursiva

De la implementación Recursiva de la sucesión de fibonacci aplicada a los 31 casos con n= 5,10,15,20,25,30,35,40,45,50, se obtiene a partir del cálculo de la mediana en cada caso la siguiente tabla:

n	S (mediana de Tiempo expresado en microsegundos)
5	0.003
10	0.003
15	0.003
20	0.003
25	0.006
30	0.038
35	0.399
40	4.523
45	56.061
50	563.255

Tabla 2: Resultados implementación recursiva fibonacci

2.4.2. Gráfico Obtenido de implementación Recursiva

A partir de los resultados obtenidos en la sección anterior (sección 2.4.2) se obtiene la siguiente curva:

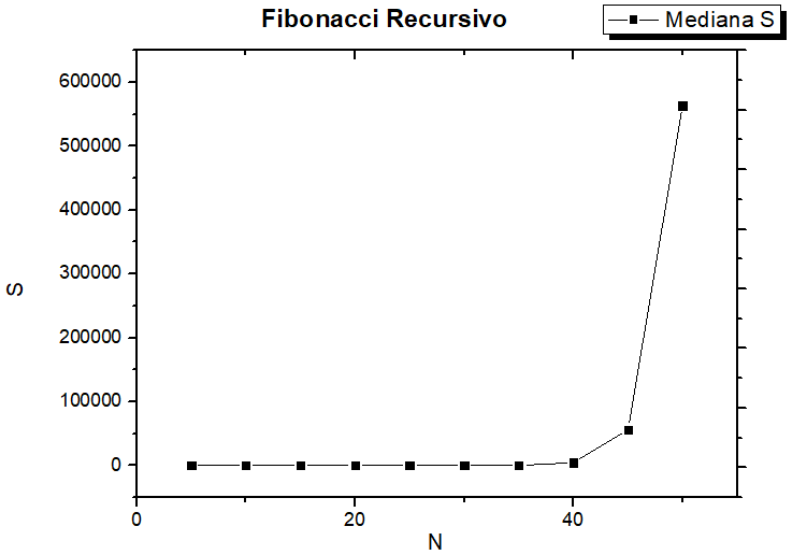

Figura 4: Gráfico función Recursiva

2.4.3. Análisis de resultados de implementación Recursiva

Tras ver el gráfico de la sección 2.4.2 hemos podido ver que la función se ejecuta 31 veces en cada caso n con un tiempo que crece conforme ha ido aumentando n.lo que se traduce que la implementación tiene una complejidad temporal mayor conforme aumente el argumento de la función recursiva, la complejidad temporal del algoritmo tiene un comportamiento exponencial y pertenece a un orden de complejidad O(n^2)

2.5. Análisis general del experimento

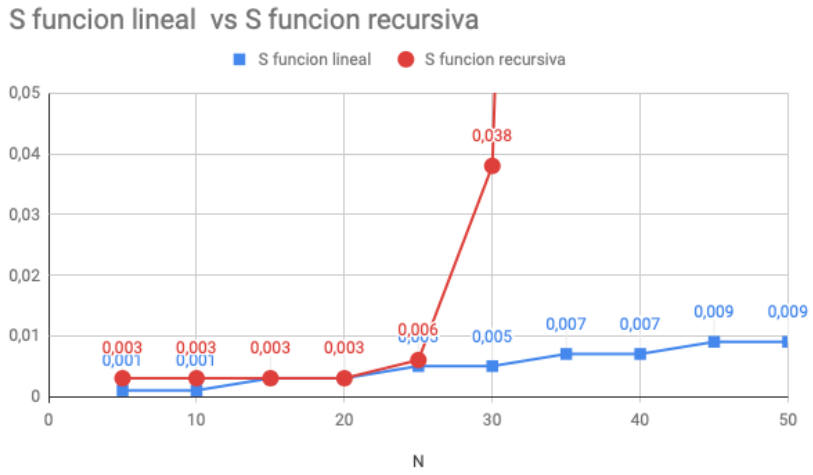

Figura 5: Gráfico función lineal vs función recursiva

Dentro de los resultados obtenidos tanto en la implementación recursiva como en la lineal se puede llegar a las siguientes conclusiones experimentales ratificando lo estudiado en clases:

- Tras observar el crecimiento de la implementación recursiva versus lineal, se puede apreciar que la implementación recursiva requiere de más recursos de tiempo que la lineal, como se puede apreciar en el gráfico de la figura 5.
- Observando el crecimiento de tiempo en la implementación recursiva vs lineal se puede apreciar que la implementación recursiva es más compleja en términos de tiempo, puesto que ocupa este recursos cada vez más conforme realiza una iteración
- Utilizando los conocimientos previamente vistos en clases, a partir de las implementaciones algorítmicas de la sucesión de fibonacci tanto lineal como recursiva, hemos llegado a las siguientes expresiones en relación a la complejidad temporal conforme al número de operaciones elementales de cada implementación algorítmica:
  - Función Lineal:
$$F(n) = 6n + 3 \Rightarrow F(n) \in O(n)$$
  - Función Recursiva:
$$f(0) = 1 \Rightarrow n \leq 1$$
$$f(n) = f(n-1) + f(n-2) + K \Rightarrow f(n) \in O(n^2)$$

3. Conclusión

Durante la realización de este informe, pudimos aplicar el conocimiento entregado en clases. cumpliendo con uno de los principales objetivos de este experimento.

El análisis estadístico de los resultado experimentales nos ayudó a observar el cambio que presenta cada algoritmo implementado en función del tiempo. Ratificando que el algoritmo recursivo consume mucho más tiempo y espacio en memoria en tiempo de ejecución.

La implementación algorítmica recursiva demanda mucho más recurso y tiempo, siendo mas compleja que la lineal.

En conclusión el algoritmo lineal corre en un tiempo más razonable por ende su complejidad temporal es mucho mejor que la recursiva.

Referencias

[1] https://www.vix.com/es/btg/curiosidades/4461/que-es-la-sucesion-de-fibonacci

[2] https://sites.google.com/site/complejidadalgoritmicaes/

[3] https://www.cs.us.es/ jalonso/cursos/i1m/temas/tema-28.html

[4] http://www.lcc.uma.es/ av/Libro/CAP1.pdf

[5] http://www2.elo.utfsm.cl/ lsb/elo320/clases/c4.pdf

[6] https://rodas5.us.es/file/0763b67a-40ea-4203-b430-3f18c74c3387/1/Tema