# Proyecto final: Recursividad en potencia de matrices

Sonya Castro - 2020
Algoritmia y Programación II - IST 2089
Profesor: PhD. Daladier Jabba M

**Resumen**

En el siguiente informe se soluciona la potencia de una matriz cuadrada A elevada a L (con L > 0) de dos formas, una iterativa y recesiva. Con esto, se busca compara la eficiencia de las dos versiones con base al tiempo de ejecución y la seguridad y, de esta forma, concluir cual versión es mejor. Para ello, se implementó un programa en java en la que se obtuvieron los datos de ejecución. Con ello se descubrió que la versión recursiva tendía a fallar desde cierto tamaño de la matriz y que, además, con exponente muy grandes los errores aumentaban. Teniendo eso en cuenta, se concluye que la versión iterativa es favorable, por su velocidad y seguridad.

**Palabras clave:** Potencia de matrices, recursividad.

## 1. Descripción del proyecto

La programación modular es un esquema ideal para resolver problemas muy complejos al dividirlos en subprogramas o procedimientos. Bajo esta misma filosofía de *dividir y vencer* surge la técnica de recursividad, en donde un procedimiento se llama a sí mismo. Por un lado, esta técnica, a pesar de no tener una gran aplicación en la industria, es necesaria para resolver ciertos problemas. Por otro lado, aumenta la posibilidad de un *StackOverFlow* o rebosamiento de pila. Los sistemas trabajan con una pila LIFO (Last Inupt, Fist Output) que almacena todos los procesos que quedan por hacer. Al trabajar con procedimientos recursivos, mientras más veces se llame el procedimiento a sí mismo, mas aumenta la posibilidad de que la pila de sistema llegue a su tope (esto depende de la memoria disponible) y ocurra un rebosamiento de cadena.

En este contexto, el objetivo del siguiente proyecto es comparar dos versiones de un algoritmo , una versión iterativa y otra recursiva, que den solución al problema de la potencia de matrices $A^L$, donde A es una matriz cuadrada de N x N elementos y L es un entero positivo mayor a cero. El problema de $A^L$ puede, usando las propiedades la potenciación, ser visto como realizar L veces la multiplicación de matrices, es decir, **un proceso repetitivo**. Para realizar el producto matricial, el número de filas del primer factor debe coincide con el número de columnas del segundo factor, generado una segunda matriz cuyos elementos son el producto escalar de las filas de A y las columnas transpuestas de B (Gutiérrez, Barrios, & Castañeda, 2020). Es decir, la multiplicación de dos matrices da como resultado una tercera matriz cuyos elementos están dados por la ecuación 1 (Lázaro, 2005).

$$A_{mn}B_{np} = C_{mp} = [c_{ij}], \text{tal que } c_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj} \quad (1)$$

Como se desea hallar la potencia de una matriz cuadrada, la condición de las filas y columnas siempre se va a cumplir. Con todo esto, se deben hallar las soluciones algorítmicas, ambas empleando procedimientos. En la implantación en Java, el exponente de la matriz (L) debe ser leído y en el

algoritmo principal habrá un ciclo que aumente el valor del tamaño de la matriz (N) iniciando con una matriz de 10x10 aumentando N con incremento de 5 hasta llegar a 10 casos. Además, la matriz debe ser generada de forma aleatoria desde el algoritmo principal. Se crearán dos clases, *iterativa* y *recursiva*, que resuelven el problema como indica su nombre, para ello, contiene un método constructor, un método de escritura y un método de potencia (que llama a un cuarto método llamado multiplicación). El tiempo de ejecución de este último método, potencia, será medido desde el *main*, para ambas versiones. Luego de resolver las 10 pruebas, los datos del tiempo ejecución serán mostrados en pantalla y guardados en archivos CSV. Además, la salidas en pantallas también serán guardas y, juntos a los CSV, se encontrar en el repositorio publico https://github.com/Sonya-c/Potencia-de-Matrices donde también estarán los algoritmos y la implementación. Al finalizar la fase experimental, se hará el respectivo análisis para hallar las conclusiones finales.

## 2. Algoritmos

### 2.1. Versión Iterativa

Algoritmo conductor

```
1:  INICIO
2:      leer L, N
3:
4:      EJECUTAR leer (N, A)
5:      B ← A
6:      EJECUTAR escribir (N, A)
7:      EJECUTAR potencia (N, A, B, L)
8:      EJECUTAR escribir (N, B)
9:  FIN
```

Algoritmo de lectura de matrices

```
1:  SUBRUTINA leer (tamaño, matriz)
2:      para f = 1, tamaño
3:          para c = 1, tamaño
4:                  leer matriz(f)(c)
5:              fin-para
6:      fin-para
7:  FIN-SUBRUTINA
```

Algoritmo de escritura de matrices

```
1:  SUBRUTINA escribir (tamaño, matriz)
2:      para f = 1, tamaño
3:          para c = 1, tamaño
4:                  escribir matriz(f)(c)
5:              fin-para
6:      fin-para
7:  FIN-SUBRUTINA
```

Algoritmo de potencia: este algoritmo espera la potencia de matrices como la multiplicación de factores iguales, en este caso una matriz, por L veces. La motivación detrás de esto es aplicar uno de los conceptos más importantes en el desarrollo de procedimientos: *divide y veceras*. De esta forma, el proceso de multiplicación es realizado por aparte.

```
1:  SUBRUTINA potencia (tamaño, matriz, potencia, exp)
2:      para e = 1, exp
3:          EJECUTAR producto (tamaño, matriz, potencia)
4:      fin-para
5:  FIN-SUBRUTINA
```

Algoritmo de multiplicación de matrices. En esta unidad del código se aplica la fórmula 1.

```
1:  SUBRUTINA producto (tamaño, A, C)
2:      B <- C
3:      para f = 1, tamaño
4:          para c = 1, tamaño
5:              suma = 0
6:              para k = 1, n
7:                  suma <-- suma + A(f)(k) * B(k)(c)
8:                          fin-para
9:                          C(f)(c) = suma
10:         fin-para
11:     fin-para
12: FIN-SUBRUTINA
```

## 2.2. Versión Recursiva

Algoritmo conductor

```
1:  INICIO
2:          leer L
3:          leer N
4:          EJECUTAR leer (N, A, 1, 1)
5:          EJECUTAR escribir (N, A, 1, 1)
6:          EJECUTAR potencia (N, A, B, L, 1)
7:          EJECUTAR escribir (N, B, 1, 1)
8:  FIN
```

Algoritmo de lectura de matrices

```
1:  SUBRUTINA leer (tamaño, matriz, f, c)
2:    si (f < tamaño)
3:      si (c < tamaño)
4:        leer matriz(f)(c)
5:        EJECUTAR leer (tamaño, matriz, f, c + 1)
6:      sino
7:        EJECUTAR leer (tamaño, matriz, f + 1, 1)
8:      fin-si
9:    fin-si
10: FIN-SUBRUTINA
```

Algoritmo de escritura de matrices

```
1:  SUBRUTINA escribir (tamaño, matriz, f, c)
2:    si (f < tamaño)
3:      si (c < tamaño)
4:        escribir matriz(f)(c)
5:        EJECUTAR escribir (tamaño, matriz, f, c + 1)
6:      sino
7:        EJECUTAR escribir (tamaño, matriz, f + 1, 1)
8:      fin-si
9:    fin-si
10: FIN-SUBRUTINA
```

Algoritmo potencia

```
1:  SUBRUTINA potencia (tamaño, matriz, potencia, exp, e)
2:      si e < exp
3:          EJECUTAR producto (tamaño, matriz, potencia, C, 0,1,1,1)
4:                  EJECUTAR potencia (tamaño, matriz, potencia, exp, e + 1)
5:      fin-si
6:  FIN-SUBRUTINA
```

Algoritmo multiplicación de matrices

```
1:  SUBRUTINA producto (tamaño, A, B, C, suma, f, c, k)
2:      si (f < tamaño)
3:          si (c < tamaño)
4:              si (k < tamaño)
5:                  suma <-- A(f)(k) * B(k)(c)
6:                  EJECUTAR producto (A, B, C, suma, f, c, k + 1)
7:                          sino
8:                  producto(f)(c) = suma
9:                  EJECUTAR producto (A, B, C, 0, f, c + 1, 1)
10:             fin-si
11:         sino
12:             EJECUTAR producto (A, B, C, 0, f + 1, 1, 1)
13:         fin-si
14:     fin-si
15: FIN-SUBRUTINA
```

## 3. Resultados experimentales

En las primeras pruebas de la implantación, se notó que la versión recursiva tendía a fallar, por ello, a la tabla de datos se la añadió una nueva columna que guardara los errores en ejecución de la versión recursiva. Además, para obtener los promedios solo ser tomaron en consideración los caso en los que la versión recursiva funciona, es decir, los casos donde los errores son ceros. En un primer momento, se repitieron las pruebas con L muy cercanos, pero, al momento de hacer la toma final de datos, se dedicó ampliar el rango de valores que tomaba L. Los tiempos que se encuentra en las tablas se hallaron con *System.nanoTime()*.

| EXPERIMENTO # 1; L = 5 | | | | | |
|---|---|---|---|---|---|
| Prueba # | N | Tiempo total (ns) | Iterativa (ns) | Recursiva (ns) | Errores |
| 1,00 | 10,00 | 2,56E+12 | 289600,00 | 723400,00 | 0,00 |
| 2,00 | 15,00 | 1,94E+12 | 706900,00 | 1535199,00 | 0,00 |
| 3,00 | 20,00 | 1,52E+12 | 1418801,00 | 519899,00 | 0,00 |
| 4,00 | 25,00 | 2,94E+12 | 1286200,00 | 1129600,00 | 0,00 |
| 5,00 | 30,00 | 4,08E+12 | 2056701,00 | | 1,00 |
| 6,00 | 35,00 | 5,75E+12 | 1275600,00 | | 1,00 |
| 7,00 | 40,00 | 6,36E+14 | 1244000,00 | | 1,00 |
| 8,00 | 45,00 | 1,62E+16 | 2293499,00 | | 1,00 |
| 9,00 | 50,00 | 1,31E+16 | 3150801,00 | | 1,00 |
| 10,00 | 55,00 | 1,47E+13 | 4192501,00 | | 1,00 |
| promedios (10:25) | | 2,24E+12 | 9,25E+05 | 9,77E+05 | 0,60 |

Tabla 1. Experimento 1 con L = 5

| EXPERIMENTO # 2; L = 50 | | | | | |
|---|---|---|---|---|---|
| Prueba # | N | Tiempo total (ns) | Iterativa (ns) | Recursiva (ns) | Errores |
| 1,00 | 10,00 | 2,34E+14 | 2,24E+06 | 1,94E+06 | 0,00 |
| 2,00 | 15,00 | 3,36E+16 | 9,41E+06 | 2,79E+06 | 0,00 |
| 3,00 | 20,00 | 1,63E+16 | 1,18E+12 | 1,33E+14 | 0,00 |
| 4,00 | 25,00 | 5,56E+14 | 1,72E+12 | 1,28E+14 | 0,00 |
| 5,00 | 30,00 | 4,56E+14 | 9,99E+06 | | 1,00 |
| 6,00 | 35,00 | 7,65E+12 | 1,46E+14 | | 1,00 |
| 7,00 | 40,00 | 8,70E+12 | 3,39E+12 | | 1,00 |
| 8,00 | 45,00 | 9,67E+12 | 3,27E+12 | | 1,00 |
| 9,00 | 50,00 | 1,01E+14 | 3,83E+12 | | 1,00 |
| 10,00 | 55,00 | 1,53E+12 | 6,58E+12 | | 1,00 |
| promedios N (10:25) | | 1,27E+16 | 7,27E+11 | 6,55E+13 | 0,60 |

Tabla 2. Experimento 2 con L = 50

| EXPERIMENTO # 3; L = 500 | | | | | |
|---|---|---|---|---|---|
| Prueba # | N | Tiempo total (ns) | Iterativa (ns) | Recursiva (ns) | Errores |
| 1,00 | 10,00 | 4,15E+11 | 8274500 | 8780701 | 0,00 |
| 2,00 | 15,00 | 3,81E+12 | 1,28E+12 | 1,51E+12 | 0,00 |
| 3,00 | 20,00 | 6,72E+12 | 2,53E+14 | 3,45E+12 | 0,00 |
| 4,00 | 25,00 | 1,39E+14 | 4,61E+12 | 7,36E+14 | 0,00 |
| 5,00 | 30,00 | 1,14E+14 | 8,37E+12 | | 1,00 |
| 6,00 | 35,00 | 1,61E+14 | 1,34E+16 | | 1,00 |
| 7,00 | 40,00 | 2,48E+14 | 1,97E+14 | | 1,00 |
| 8,00 | 45,00 | 3,38E+13 | 2,83E+16 | | 1,00 |
| 9,00 | 50,00 | 4,94E+13 | 4,07E+16 | | 1,00 |
| 10,00 | 55,00 | 1,03E+18 | 9,25E+13 | | 1,00 |
| promedios N (10:25) | | 3,74E+13 | 6,46E+13 | 1,85E+14 | 0,60 |

Tabla 3. Experimento 3 con L = 500

### 3.1. El exponente

Tomando en consideración los caso en los que la versión recursiva no presenta errores, se creó una gráfica (figura 1) que muestra la relación entre el tiempo de ejecución y el exponente. Con esto, podemos notar que el tiempo de ejecución, en ambos casos, se ve notoriamente incrementado, por lo que, efectivamente, el valor del exponente es un factor que influye en la ejecución del programa.

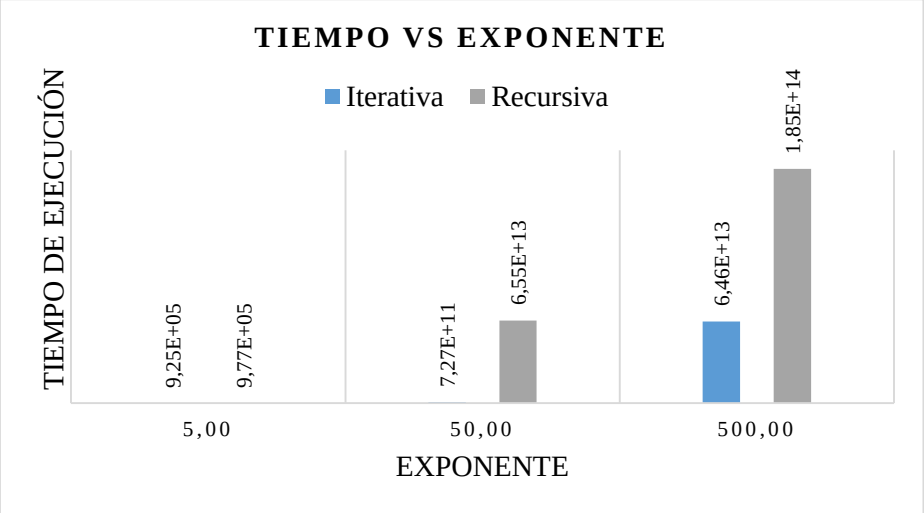


Figura 1. Tiempo de ejecución en relación al exponente

Sin embargo, observando los primeros datos experimentales de las tablas 1, 2, y 3, se podría llegar a la conclusión de que la influencia del tamaño del exponente en la versión recursiva es nula, pues en todos los casos la versión recursiva falla dependiendo del tamaño de la matriz, no del exponente que se escoja. Ante estos resultados se decidió hacer dos nuevos experimentos, aumentando, considerablemente, el valor del exponente.

| EXPERIMENTO # 4; L = 1000 | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| Prueba # | N | Tiempo total (ns) | Iterativa (ns) | Recursiva(ns) | Errores |
| 1,00 | 10,00 | 4,92E+12 | 1,37E+12 | 1,33E+12 | 0,00 |
| 2,00 | 15,00 | 6,26E+14 | 2,32E+12 | 3,10E+12 | 0,00 |
| 3,00 | 20,00 | 1,42E+16 | 5,25E+14 | 8,11E+12 | 0,00 |
| 4,00 | 25,00 | 2,07E+14 | 9,78E+12 | | 1,00 |
| 5,00 | 30,00 | 2,31E+14 | 1,63E+16 | | 1,00 |
| 6,00 | 35,00 | 2,85E+14 | 2,62E+13 | | 1,00 |
| 7,00 | 40,00 | 4,19E+14 | 3,89E+14 | | 1,00 |
| 8,00 | 45,00 | 6,12E+14 | 5,68E+14 | | 1,00 |
| 9,00 | 50,00 | 1,13E+18 | 1,05E+16 | | 1,00 |
| 10,00 | 55,00 | 1,65E+16 | 1,59E+16 | | 1,00 |

Tabla 4. Experimento 4 con L = 1000

| EXPERIMENTO #5; L = 10000 | | | | | |
|---|---|---|---|---|---|
| Prueba # | N | Tiempo total (ns) | Iterativa (ns) | Recursiva (ns) | Errores |
| 1,00 | 10,00 | 1,75E+14 | 7,04E+14 | | 1 |
| 2,00 | 15,00 | 4,92E+14 | 2,00E+16 | 2,79E+14 | 0 |
| 3,00 | 20,00 | 7,54E+14 | 4,74E+14 | | 1 |
| 4,00 | 25,00 | 1,03E+16 | 9,28E+14 | | 1 |
| 5,00 | 30,00 | 2,55E+16 | 2,51E+15 | | 1 |
| 6,00 | 35,00 | 3,32E+16 | 3,26E+16 | | 1 |
| 7,00 | 40,00 | 5,40E+18 | 5,34E+16 | | 1 |
| 8,00 | 45,00 | 7,50E+18 | 7,43E+18 | | 1 |
| 9,00 | 50,00 | 1,11E+20 | 1,10E+18 | | 1 |
| 10,00 | 55,00 | 1,62E+20 | 1,62E+18 | | 1 |

Tabla 5. Experimento 5 con L = 10000

Con estos nuevos resultados se deja en evidencia que la eficacia de la versión recursiva también depende del exponente. En la tabla 4, notamos que los errores empiezan a suceder más pronto (antes empezaban a ocurrir en N = 25, ahora empiezan a hacerlo en N = 20), aunque este parece ser un cambio menor en contraste con lo que sucede en la tabla 5, donde en solo una ocasión el proceso se pudo realizar exitosamente.

### 3.2. El tamaño de la matriz

Como vimos anterior mente, el exponente tiene una alta influencia en el tiempo de ejecución, pero una baja en relación a la eficacia de la versión recursiva. En cambio, promediando los tres primeros casos mostrados, parece que el éxito del proceso recursivo depende únicamente del tamaño de la matriz, más específicamente, si esta es mayor a 25 como podemos ver en la figura 2.

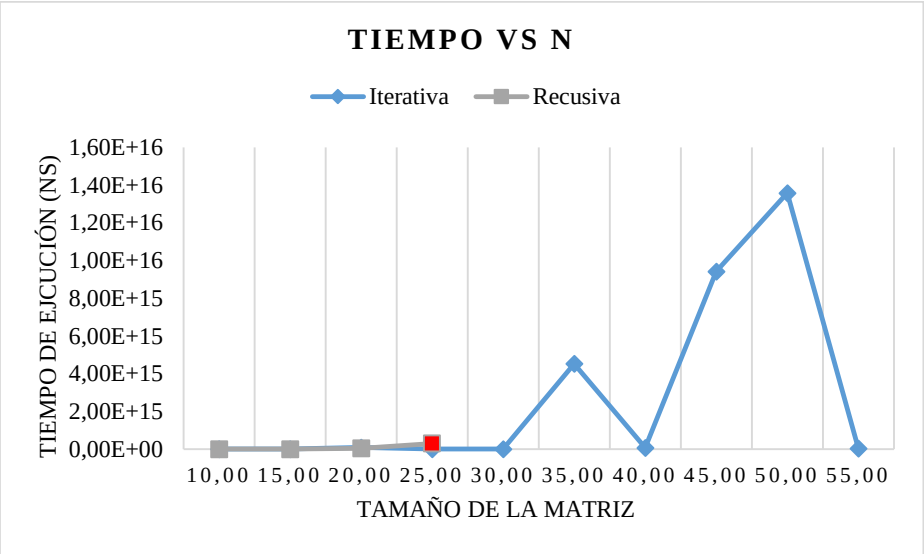


Figura 2. Promedio de caso. Tiempo en relación al tamaño de la matriz

A partir de un N mayor a 25, la versión recursiva empieza a fallar. Con esta observación en mente, se decidió ampliar el rango de experimento, haciendo 20 pruebas iniciando en 10 hasta 29 (con paso de 1) y exponente 50. Los resultados fueron los esperados, a partir de 25 se empieza a fallar, pero, también se mostró un dato que los demás experimentos no habían mostrado: cuando N se está acercando a 25, el tiempo de ejecución de la versión recesiva aumenta, considerablemente, comparándola con la versión iterativa.

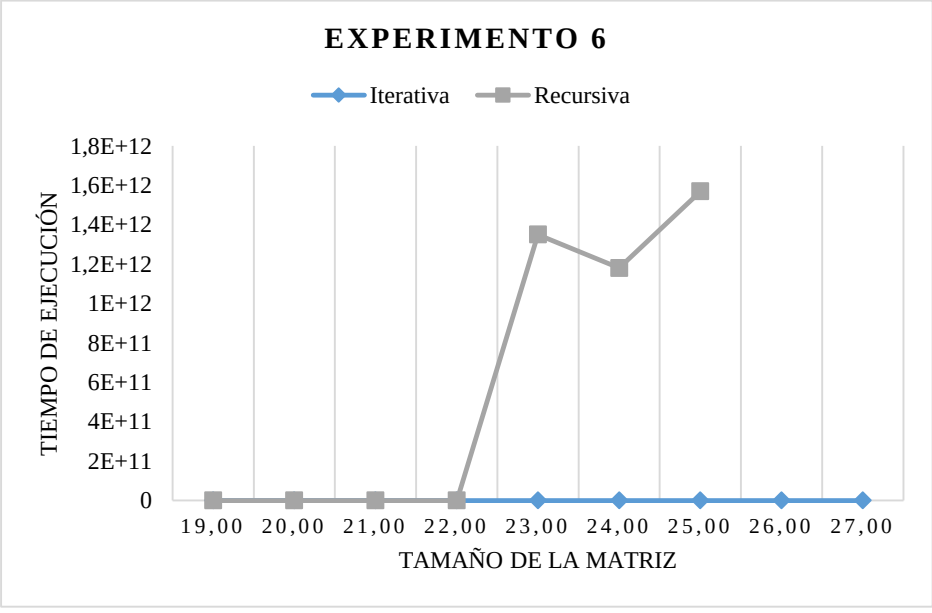


Figura 2. Experimento 6.

## 4. Conclusiones

En el caso de la potenciación de matrices, los algoritmos recursivos pueden que no sean la mejor opción. Con exponente muy grandes o desde determinado tamaño de la matriz (esto dependerá de la memoria disponible) el algoritmo empezará a falla por un desbordamiento de pila. Además, el tiempo de ejecución empezará a aumentar drásticamente comparándolo con su contraparte no recursiva que, de hecho, en los casos estudiados nunca causo un error en la ejecución. Teniendo esto en cuenta, tiempo y seguridad de ejecución, la versión iterativa brinda velocidad y un algoritmo que no fallara, sin importar el tamaño de la matriz o el exponente, haciéndola el mejor camino a tomar.

## 5. Referencias

Gutiérrez, I., Barrios, A., & Castañeda, S. (2020). En *Manual de álgebra lineal 2da edición* (2 ed., págs. 79-85). Barranquilla: Universidad del Norte.

Lázaro, M. (2005). *Algebra Lineal* (2 ed.). Perú: Moshera S.R.L.