Escritura de algoritmo Multiplicación matricial

Juan Jose Bolaños Melo[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
{bolanos.jj}@javeriana.edu.co

1 de septiembre de 2022

Resumen

En este documento se presenta la formalización e implementación de un algoritmo basado en la estrategia de programación dinámica, la cual se basa en calcular la parentización que minimice la cantidad de multiplicaciones escalares en una composición de matrices. Para llevar a cabo el objetivo se procede a escribir formalmente los algoritmos a usar a su vez que se trabaja sobre el lenguaje de programación Julia. Además, se presenta un análisis experimental de la complejidad de este algoritmo. **Palabras clave:** multiplicación, matricial, cadena, experimentación, escalares, matrices.

Índice

1. Introducción **1**

2. Formalización del problema **1**
2.1. Definición del problema de "Multiplicación de Matrices' 2

3. Algoritmos de solución **2**
3.1. Método "bottom-up con backtracking " 2
3.1.1. Análisis de complejidad 2

4. Análisis experimental **2**
4.1. Secuencias aleatorias 3

1. Introducción

En el álgebra, pudimos observar que las multiplicaciones de matrices se basan en operaciones binarias que como resultado producen una matriz a partir de dos matrices, teniendo como requisito que el numero de columnas de nuestra primera matriz sea igual al numero de filas de la segunda matriz. A continuación se presenta la escritura formal de el algoritmo basado en la estrategia de programación dinámica y un análisis experimental de la complejidad de cada uno de ellos.

2. Formalización del problema

Para dar solución al problema de multiplicación de matrices, dada la dimensión de una secuencia de matrices en un arreglo, vamos a tener como objetivo principal hallar la forma mas eficiente de multiplicar estas matrices dadas, de manera que mediante una Optimización podamos encontrar no solo el numero de multiplicaciones mínimo sino también el camino (orden de multiplicación) que se debe seguir para cumplir con el mínimo encontrado. Para llevar a cabo este objetivo vamos a trabajar a lo largo del presente documento con los siguientes algoritmos que dan solución a este problema:

1. Algoritmo con Memoización: Para esta solución se dará uso de la técnica de programación en la cual se reduce el tiempo de ejecución de una función a cambio de ampliar el coste del espacio(normalmente memoria)

2. Algoritmo bottom-up : En este solución vamos a evitar la recursividad, ahorrando el costo de memoria en el que incurre la recursividad cuando acumula la pila de llamadas.

3. Algoritmo bottom-up con backtracking :Al estar trabajando sobre un problema de optimización vamos a querer saber no solo la optimización mínima sino también como lograr esta, es por esto que se implementa esta solución.

2.1. Definición del problema de "Multiplicación de Matrices'

Así, el problema de buscar que el numero total de de multiplicaciones de elementos sea mínimo dada la dimensión de una secuencia de matrices en un arreglo,se define a partir de:

1. sea $A_1{,}_k\, A_k + 1{,}_n = (A_1.....A_k)(A_k + 1.....A_n)$ donde $1 \leq k \leq n$

producir un M_ikj el cual sera el número mínimo de multiplicaciones escalares para calcular $A_1{,}_k\, A_k + 1{,}_j$

- Entradas:
  - $S = \langle s_1 : 0 \leq i \leq n \leq s_1 \in \mathbb{N} \rangle$ donde s_0 es la cantidad de filas de la primera matriz y s_1 es la cantidad de columnas de la primera matriz y la cantidad de filas de la segunda matriz.
- Salidas:
  - Cadena de caracteres $F\, \langle 'a'.....'z'a'A'.....Z' \rangle$.

3. Algoritmos de solución

3.1. Método "bottom-up con backtracking "

La idea de este algoritmo es: primero que todo darnos cuenta de que el algoritmo dado realiza repetitivamente cálculos iguales, lo cual termina generando recalculos, para quitar esta propiedad de nuestro algoritmo por el método de memoización vamos hacer uso de una matriz temporal que se usará con el fin de reutilizar cálculos antes hechos luego de esto mediante la técnica del bottom-up vamos a deshacernos de la recursividad implementada, esto se logro cambiando los llamados a las funciones por asignaciones que se harán de forma iterativa. Por ultimo llegamos a utilizar bottom-up con backtracking ya que lo que nos importa en este problema no es solo encontrar el mínimo de multiplicaciones sino también el como llegar a ese resultado, mas exactamente en que posición se deben usar los paréntesis para que sea la mas óptima. <.

3.1.1. Análisis de complejidad

Por inspección de código: el algoritmo cuenta con tres ciclos anidados, mas específicamente tres for,que en todos los casos, va ir recorriendo las matrices para realizar la asignación del calculo; entonces, este algoritmo es $O(|n^3|)$.

El pseudocódigo mostrado a continuación cumple estrictamente los criterios del paradigma de dividir y vencer para especialmente utilizado para la funcionalidad de bottom-up con backtracking .

4. Análisis experimental

En esta sección se presentarán algunos experimentos los cuales no se centran en confirmar el orden de complejidad del algoritmo presentado en la sección sino la comprobación de funcionamiento del algoritmo para diferente entradas de datos(matrices) 3.

Algoritmo 1 Algoritmo "bottom-up con backtracking ".

```
1: procedure MULTIPLICACIONMATRICES(S)
2:     Variables :
3:     n = tamano − 1
4:     T = [N]
5:     B = [N]
6:     for i ← n − 2 to − 1 do
7:         for j ← i + 1 to n do
8:             q ← (−inf)
9:             m ← 0
10:            for k ← i to j do
11:                izq ← T[i][k]
12:                der ← T[k + 1][j]
13:                v ← izq + der + D[i − 1] ∗ D[k] ∗ D[j]
14:                if v < q then
15:                    q ← v
16:                    m ← k
17:                end if
18:                T[i][j] ← q
19:                B[i][j] ← m
20:            end for
21:        end for
22:    end for
23:    return T, B
24: end procedure
```

4.1. Secuencias aleatorias

Acá se presentan los experimentos cuando el algoritmo se ejecuta con secuencias de entrada: .

1. $D = [10, 100, 5, 50]$

Figura 1: Prueba 1".

2. $D = [10, 52354, 560, 100, 2335, 123]$
3. $D = [10, 5, 6, 7, 2, 5, 3, 4]$

Figura 2: Prueba 1".

Figura 3: Prueba 1".