ITC-ADA-C1-2023: Assignment #3

Luis Ballado
luis.ballado@cinvestav.mx

CINVESTAV UNIDAD TAMAULIPAS — February 10, 2023

1 Considerando el algoritmo recursivo mostrado a continuación, responda las siguientes preguntas

Algorithm 1: Algoritmo Misterio

```
entrada: Un arreglo A[0..n − 1] de números reales
if n==1 then
    return A[0];
else
    temp ← Misterio(A[0..n − 2])
    if temp <= A[n-1] then
        return temp;
    else
        return A[n-1];
```

Pregunta 1

¿Qué calcula el algoritmo?
El algoritmo recursivo calcula el elemento de minimo valor del arreglo

Pregunta 2

¿Cuál es el parámetro que indica el tamaño de la entrada del algoritmo? *n, siendo este el tamaño del arreglo de entrada. Apartir de este valor se puede llegar al caso base.*

Pregunta 3

¿Cuál es la operación básica del algoritmo?
La comparación es la operación básica de tiempo constante $O(1)$ en cada llamada, el algoritmo compara el valor minimo del subarreglo con el último elemento del subarreglo

Pregunta 4

¿Cuáles son el mejor caso y el peor caso para este algoritmo? *El mejor caso ocurre cuando el valor minimo es encontrado en la primera llamada donde el arreglo de entrada es pequeño A[0] y seria de tiempo constante $O(1)$*
*El peor caso ocurre cuando el minimo valor es encontrado en la última llamada a la función. En este caso el algoritmo hace **n** llamadas a función, siendo n el tamaño del arreglo. Convirtiendose en una complejidad lineal $O(n)$*

Pregunta 5

Proporcione una expresión matemática (relación de recurrencia), en función del tamaño de la entrada del algoritmo, que permita calcular cuántas veces se ejecuta la operación básica en este algoritmo.
T(n) = 1 cuando n = 1
T(n) = T(n-1)+1 cuando n > 1
T(n-1)+1 = T(n-2)+2 = T(n-3)+3 podemos decir que la recurrencia esta expresada como: $T(n-i)+i$ *al ser* $n-i=0$ *para llegar al caso base* $n=i$
T(0)+n = n+1; por lo tanto $T(n) \in \Theta(n)$

Pregunta 6

Resuelva la relación de recurrencia propuesta mediante substitución hacia atrás
Partiendo de la relación propuesta: $T(n) = T(n-1) + O(1)$
$T(n)$ *es el tiempo de complejidad del algoritmo para tamaños de entrada n,* $T(n-1)$ *es el tiempo de complejidad para tamaños* $n-1$
$O(1)$ *tiempo de complejidad por las comparaciones. La relación de recurrencia mediante substitución hacia atrás:*
$T(n) = T(n-1) + O(1)$
$T(n) = (T(n-2) + O(1)) + O(1)$
$T(n) = (T(n-3) + O(1) + O(1)) + O(1)$
$T(n) = (T(n-4) + O(1) + O(1) + O(1)) + O(1)$
podemos concluir que $T(n) = (T(1) + O(1) + O(1) + ... + O(1)) + O(1))$ donde $T(n)$ es el tiempo de complejidad para un array de tamaño n, $T(1)$ es la complejidad para tamaño 1 por la comparación constante

Pregunta 7

¿Cuál es la clase de eficiencia? $T(n) \in O(n)$

2 Dado el problema de encontrar el determinante de una matriz A de nxn , desarrolle los siguiente puntos:

Pregunta 8

Programe las versiones iterativa y recursiva del algoritmo para resolver el problema

2.1 Implementación

Código completo repo github `https://raw.githubusercontent.com/luisballado/ADA/main/assignments/assignment_3/determinante.py`

Por la facilidad de leer argumentos de entrada y guardar a texto los resultados de las corridas de los códigos, cambie la implementación

Ejecutar desde una terminal, depende de la versión de python instalada desde el ordenador. Ya que se incluye una solución de la libreria de numpy para la verificación de resultados, es probable la instalación de la biblioteca con el uso de pip **$ pip install numpy**

Command Line

```
$ python3.10 determinante.py -size 12 > 12.txt
```

determinante.py

```python
#version recursiva
def recursivo(matrix):

    n = len(matrix)

    #caso base
    if n == 1:
        return matrix[0][0]
    if n == 2:
        return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]

    det = 0
    for i in range(n):
        sub_matrix = [row[:i] + row[i+1:] for row in matrix[1:]]
        det += ((-1) ** i) * matrix[0][i] * recursivo(sub_matrix)

    return det

#version iterativa
def iterativo(matrix):
    determinant = 1
    n = len(matrix)
    for i in range(n):
        # Encontrar el valor maximo en la columna
        maxRow = i
        for j in range(i + 1, n):
            if abs(matrix[j][i]) > abs(matrix[maxRow][i]):
                maxRow = j
        # intercambiar el maximo valor con el valor actual
        if maxRow != i:
            matrix[i], matrix[maxRow] = matrix[maxRow], matrix[i]
            determinant *= -1

        # Multiplicar el determinante por la diagonal
        determinant *= matrix[i][i]

        # terminar si los elementos de la diagonal son ceros
        if matrix[i][i] == 0:
            return 0

        # Eliminar los elementos abajo de la diagonal
        for j in range(i + 1, n):
            if matrix[i][i] == 0:
                scale = 0
            else:
                scale = matrix[j][i] / matrix[i][i]

            for k in range(i, n):
                matrix[j][k] -= scale * matrix[i][k]

    return determinant
```

Pregunta 9

Analice matemáticamente cada versión el algoritmo por separado usando las metodologías vistas en clase. *versión recursiva $T(n) = T(n-1) + O(n^2)$ donde $O(n^2)$ es el tiempo de complejidad del calculo de las submatrices, y $T(n-1)$ tiempo de complejidad cuando se reduce la matriz $n-1$ en un análisis por cofactores*

El tiempo de complejidad del cálculo de submatrices puede estar expresado por $O(n^2)$ ya que cada elemento pertenece a la primera fila, se va creando una submatriz de tamaño reducido $n-1xn-1$ para ir llegando al caso base.

$T(n) = T(n-1) + O(n^2)$

$T(n) = (T(n-2) + O(n^2)) + O(n^2)$

hasta llegar al caso base

$T(n) = (T(1) + O(n^2) + O(n^2) + ... + O(n^2) + O(n^2))$

podemos concluir que $T(n) = T(1) + n * O(n^2)$

$T(n) = O(n^3)$, pero para un peor caso donde la matriz es muy grande,

el algoritmo en su versión recursiva $T(n) \in \Theta(n!)$

Pregunta 10

En base a los resultados obtenidos en el punto anterior determine cuál de los dos algoritmos es más eficiente *A pesar del crecimiento del orden polinomial, la versión iterativa tiene un mejor comportamiento contra su versión recursiva*

Pregunta 11

Para cada uno de los dos algoritmos desarrollados aplique el método descrito en el apartado "Doubling ratio experiments" del libro Algorithms de Sedgewick y Wayne, generando instancias de tamaños 1000,2000,etc; hasta lograr un radio de 2^b, ejecutando 20 pruebas con cada tamaño y con cada algoritmo. Registre sus resultados. *Al ejecutar los algoritmos dentro del mismo programa y guardar los resultados en un archivo de texto con la salidad resultante de cada experimento, el orden de crecimiento factorial del algoritmo recursivo disparo la ejecución del análisis de una matriz de tamaño 24x24 llegando a no ser rentable para esperar el resultado*

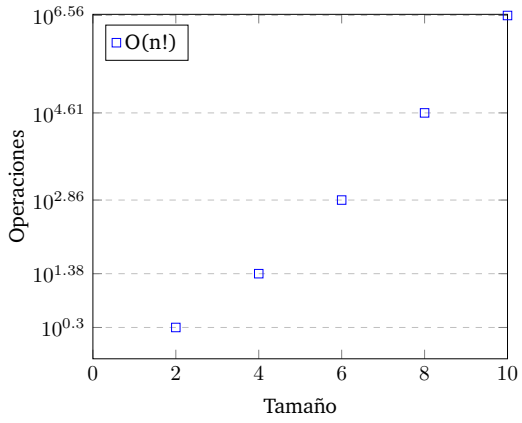
Pregunta 12

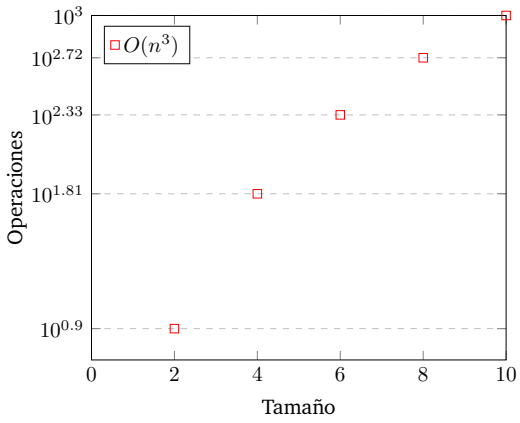
Para cada algoritmo comparado realice una tabla con 5 predicciones, posteriores al tamaño con que se logró obtener el radio 2^b

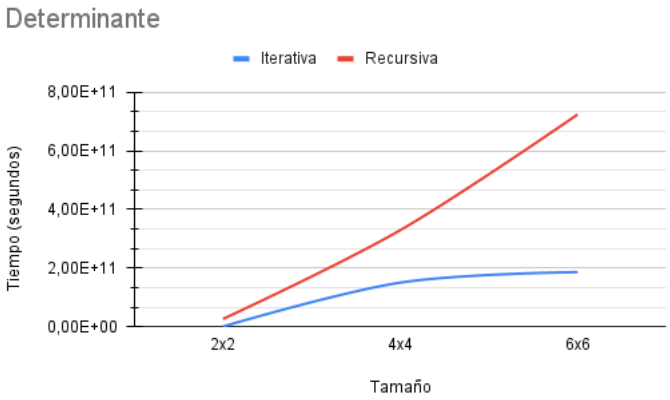
TAMAÑO	Iterativa	Recursiva
14x14	0.0391 ms	-
28x28	0.1227 ms	+7 días
56x56	0.4293 ms	13 años
112x112	0.0052 s	∞
224x224	0.0208 s	∞

Pregunta 13

Para cada algoritmo comparado grafique los siguientes resultados de sus ejecuciones *Los resultados obtenidos de las corridas de los algoritmos son las siguientes:*







TAMAÑO	Iterativa	Recursiva
2x2	0.01120 ms	0.00262 ms
4x4	0.01502 ms	0.03290 ms
6x6	0.01859 ms	0.00072 ms
8x8	0.02670 ms	0.05136 ms
12x12	0.03695 ms	654000 ms
14x14	-	-
16x16	-	-
24x24	-	-

para la corrida de tamaño 24x24 a pesar de que el tiempo de ejecución fue de aproximadamente 5 horas, paré el proceso debido a que no tenia fin

Pregunta 14

Con base en los experimentos realizados y considerando un tiempo máximo de ejecución sobre su computadora de 7 días, ¿Cuál es el tamaño máximo de entrada que puede resolver cada algoritmo analizado? Iterativo *En su versión iterativa el algoritmo tiene una complejidad* $O(n^3)$*. Y asumiendo que de entrada es una matriz de más de mil elementos, para cada iteración el algoritmo requerá* 1000^3 *operaciones y considerando que las ejecuciones por segundo dependen del poder de procesamiendo de la máquina a usar, digamos que puede ejecutar* 10^9 *operaciones por segundo. Calcularíamos el numero de operaciones en un periodo de 7 dias como 7dias * 24horas * 60minutos * 60segundos = 604800 segundos; si los multiplicamos por el número de ejecuciones por segundo aproximadamente* $6.048 * 10^4$ *iteraciones en 7 días*
Recursivo *En su versión recursiva el algoritmo tiene una complejidad* $O(n!)$ *lo que nos dice es que el numero de operaciones que ejecuta el algoritmo crece en un rango proporcional al factorial de la entrada de la matriz. Para largas entradas las operaciones son extremadamente largas y no practicas de realizar. Para una entrada de n = 10, el número de operaciones rondaria en 3,628,800, pero si doblamos la entrada las operaciones se disparán a 2,432,902,000,000,000,000 lo cual es muy lejos del rango que pueda realizar en 7 días*

Pregunta 15

Conclusiones respecto al orden de crecimiento de cada algoritmo observado empíricamente y constrástelas contra los resultados de sus análisis matemático *Debido a los tiempos de ejecución altos para los cálculos de matrices con grandes tamaños los experimientos fueron reducidos a tamaños pequeños, pero de igual forma se pudo apreciar el mejor comportamiento para la versión iterativa del calculo de una determinante. En la práctica la ejecución de algoritmos de orden factorial es una mala idea por el crecimiento rápido y sólo es apreciable con entradas pequeñas.*

3 Referencias

- LATEX Documentación - `https://www.overleaf.com/learn`
- RELACIONES DE RECURRENCIA - `https://medium.com/@matematicasdiscretaslibro/cap%C3%ADtulo-10-relaciones-de-recurrencia-7fe23208184`
- Cálculo de determinantes - `https://www.uv.mx/personal/aherrera/files/2014/08/08d.-MENORES-COFACTORES-pdf`
- Sedgewick And Wayne - Algorithms 4th Edición
- Manejo de Listas python - `https://uniwebsidad.com/libros/algoritmos-python/capitulo-7/listas`
- Determiante de una Matriz - `https://www.geeksforgeeks.org/determinant-of-a-matrix/`
- What is the best algorithm to find a determinant of a matrix? - `https://stackoverflow.com/questions/2435133/what-is-the-best-algorithm-to-find-a-determinant-of-a-matrix`