Escritura de Algoritmo para hallar secuencia creciente dentro de una matriz cuadrada

Pablo Santander Álvarez, Juan José Bolaños

Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
santander_p_@javeriana.edu.co, bolanos.jj@javeriana.edu.co

29 de septiembre de 2022

Resumen

En este documento se presenta la formalización e implementación de un algoritmo basado en la estrategia de programación dinámica, que soluciona el caso de: dada una matriz cuadrada natural de tamaño N x N, que contiene los números únicos en el rango $[1, N\text{x}N]$, pero que no están forzosamente en orden, encontrar la secuencia más larga de vecinos que están ordenados y los elementos adyacentes en la matriz tienen una diferencia de +1.Además se presentan análisis experimentales. **Palabras clave:** programación dinámica, secuencia, matriz, algoritmo, formalización, experimentación, complejidad.

Índice

1. Introducción **1**

2. Formalización del problema **1**
2.1. Definición del problema de secuencia creciente dentro de una matriz cuadrada 2

3. Algoritmo de solución **2**
3.1. Secuencia creciente dentro de una matriz cuadrada - Bottom-Up con Backtracking 2
3.1.1. Análisis de complejidad 6
3.1.2. Invariante 6

4. Análisis experimental **6**

1. Introducción

El presente documento tiene como objetivo mostrar el proceso para resolver un algoritmo que tiene como objetivo encontrar la secuencia creciente mas larga dentro de una matriz cuadrada, utilizando la técnica de programación dinámica donde partiendo de una ecuación recursiva se busca construir diferentes algoritmos hasta poder llegar a aquel que nos indique el tamaño de la secuencia mas larga creciente dentro de la matriz (optimización) y que a su vez nos muestre cada uno de los elementos que conforman la secuencia encontrada. Se pretende mostrar: la formalización del problema (sección 2), la escritura formal de tres algoritmos (sección 3) y un análisis experimental de la complejidad de estos.

2. Formalización del problema

Para el problema de *secuencia creciente* inicialmente podemos partir de una matriz $A : \mathbb{R}^{(n*n)}$, en la cual nos piden hallar la secuencia creciente mas larga. Para llevar a cabo el objetivo de encontrar la ruta óptima(la

mas larga) y que a su vez es de manera creciente (sus elementos crecen de uno en uno) vamos a comenzar desde cualquiera de la celdas que componen la matriz, y en cada una de estas encontraremos la ruta mas larga a la que podemos ir, de manera que en cada celda, vamos a poder realizar 4 movimientos: derecha, izquierda, arriba y abajo. Con los cuales podremos ir explorando cada uno de los elementos adyacentes de la celda actual, hasta que podamos alcanzar los limites de la matriz A o de lo contrario lleguemos a que las celdas adyacentes contengan elementos menores a los de la actual, Ambos casos hacen parte de la condición base para detener la recursividad. Es importante denotar también, que, existe la posibilidad de que la secuencia creciente más larga sea de un solo elemento, por ejemplo en una matriz donde no existan elementos con diferencia de 1 para ningún elemento.

2.1. Definición del problema de secuencia creciente dentro de una matriz cuadrada

Así, el problema de hallar la secuencia creciente mas larga dentro de una matriz cuadrada se define a partir de:

1. Una matriz cuadrada natural de tamaño NxN, donde encontramos los números únicos en el rango $[1, N$x$N]$, los cuales no están forzosamente en orden

Produciendo un número que indica la mayor cantidad de elementos que conforman la secuencia creciente mas larga dentro de la matriz A.

- Entradas:
  - Inicialmente podemos partir de una matriz $A : \mathbb{R}^{(n*n)}$.
- Salidas:
  - Teniendo en cuenta la entrada, la primera salida seria entonces: tamaño M de la secuencia S creciente mas larga . La segunda salida, la cual es producto del Backtracking, seria una secuencia $S = \langle a_i \in Sm \rangle$.

3. Algoritmo de solución

3.1. Secuencia creciente dentro de una matriz cuadrada - Bottom-Up con Backtracking

Este algoritmo es el resultado final de realizar los diferentes pasos necesarios para construir un algoritmo de programación dinámica, siendo estos: Algoritmo inocente, Algoritmo con Memoización, Algoritmo bottom-up y finalmente bottom-up con backtracking. La idea de este algoritmo es obtener el tamaño de la secuencia creciente mas larga y a su vez, mostrar exactamente los elementos que conforman la secuencia que llega a dicha optimización. Esto se logra mediante la implementación de un algoritmo de programación dinámica encargado de realizar el backtracking, construyendo entonces la secuencia de números que forman el camino por el cual se obtuvo la solución. Esto es importante ya que el solo hecho de mostrar el tamaño de la secuencia mas larga no es suficiente para que un usuario pueda entender y saber cómo se llega al resultado y cuales elementos conforman esa secuencia, allí es donde entra en juego el backtracking y la posibilidad visual que brinda para entender de una mejor manera la solución.

Algoritmo 1 Algoritmo Inocente Recursivo.

Require: matriz $A : \mathbb{R}^{(n*n)}$

```
1: global variable movimientos[4][2] = [-1, 0, 1, 0, 0, -1, 0, 1];
2: procedure SECUENCIACRECIENTE(Matriz)
3:     rutaMax ← 1
4:     for i ← 0 to size(Matriz) do
5:         for j ← 0 to size(Matriz[0]) do
6:             rutaMax ← max(rutaMax, recursividad(Matriz, i, j))
7:         end for
8:     end for
9:     RETURN(rutaMax)
10: end procedure
11:
12: procedure RECURSIVIDAD(Matriz, i, j)
13:     maximo ← 1 longitud maxima 1.
14:     for k ← 0 to K < 4 do
15:         auxi ← i + adyacentes[k][0]
16:         auxj ← j + adyacentes[k][1]
17:         if auxi < 0||auxj < 0||auxi >= size(Matriz)||auxj >= size(Matriz[0])||Matriz([auxi][auxj])
    then
18:             maximo ← max(maximo, 1 + recursividad(Matriz, auxi, auxj));
19:         end if
20:     end for
21:     RETURN(maximo)
22: end procedure
```

Algoritmo 2 Algoritmo Memoizado.

Require: matriz $M : \mathbb{R}^{(n*n)}$

```
1: global variable movimientos[4][2] = [-1, 0, 1, 0, 0, -1, 0, 1];
2: procedure secuenciaCreciente(Matriz)
3:     rutaMax ← 0
4:     n ← n
5:     m ← size(Matriz[0])
6:     for i ← 0 to n do
7:         for j ← 0 to m do
8:             rutaMax ← max(rutaMax, recursividad(Matriz, i, j))
9:         end for
10:    end for
11:    Return(rutaMax)
12: end procedure
13:
14: procedure recursividad(Matriz, i, j)
15:    if M[i][j] then
16:        Return(M[i][j])
17:    end if
18:    M[i][j] ← 1
19:    for k ← 0 to K < 4 do
20:        auxi ← i + movimientos[k][0]
21:        auxj ← j + movimientos[k][1]
22:        if auxi < 0||auxj < 0||auxi >= n||auxj >= m||Matriz([auxi][auxj]) then
23:            M[i][j] ← max(M[i][j], 1 + recursividad(Matriz, auxi, auxj));
24:        end if
25:    end for
26:    Return(M[i][j])
27: end procedure
```

Algoritmo 3 Algoritmo con Backtracking

```
1: procedure DFS(matriz, dp, i, j)
2:     if dp[i][j]! = 0 then
3:         RETURN(dp[i][j])
4:     end if
5:     max ← 1, n ← size(Matriz)
6:     if i > 0 and matrix[i − 1][j] == matriz[i][j] + 1 then
7:         len ← 1 + dfs(matriz, dp, i − 1, j)
8:         if len > max then
9:             max ← len
10:        end if
11:    end if
12:    if i < n − 1 and matriz[i + 1][j] == matriz[i][j] + 1 then
13:        len ← 1 + dfs(matriz, dp, i + 1, j)
14:        if len > max then
15:            max ← len
16:        end if
17:    end if
18:    if j > 0  matriz[i][j − 1] == matriz[i][j] + 1 then
19:        len ← 1 + dfs(matriz, dp, i, j − 1)
20:        if len > max then
21:            max ← len
22:        end if
23:    end if
24:    if j < n − 1 matriz[i][j + 1] == matriz[i][j] + 1 then
25:        len ← 1 + dfs(matriz, dp, i, j + 1)
26:        if len > max then
27:            max ← len
28:        end if
29:    end if
30:    dp[i][j] ← max
31:    RETURN(max)
32: end procedure
33:
34: procedure SECUENCIACRECIENTE(dp, i, j)
35:    max ← 0, int: path
36:    for i ← 0 to n do
37:        for j ← 0 to n do
38:            len ← dfs(matriz, dp, i, j)
39:            if len > max then
40:                max ← len
41:            end if
42:        end for
43:    end for
44:    for i ← 0 to n do
45:        for j ← 0 to n do
46:            if dp[i][j] == max then
47:                vector < int > p[i, j]
48:                path.pushBack(p), max − −
49:                for x ← 0 to n do
50:                    if dp[i][x] == max then
51:                        vector < int > p[i, jx]
52:                        path.pushBack(p), max − −
53:                        x ← 0
54:                    end if
55:                end for
56:                i ← 0
57:            end if
58:        end for
59:    end for
```

3.1.1. Análisis de complejidad

Para determinar la complejidad del algoritmo podemos usar la técnica de inspección de código, donde podemos identificar que el algoritmo posee tres ciclos que se encuentran anidados. Por esta razón, el algoritmo tiene una notación de $O(|N|^3)$

3.1.2. Invariante

Como se analizó y dijo en clase, la Invariante para los algoritmos desarrollados mediante la técnica de Programación Dinámica es el correcto llenado de la tabla o estructura de datos a usar, donde se almacenan distintos cálculos permitiendo que los elementos que ya han sido calculados no tengan necesidad de calcularse de nuevo.

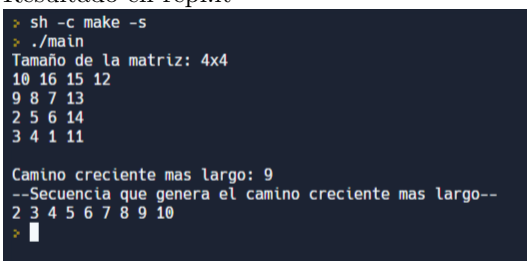
4. Análisis experimental

En esta sección se presenta el comportamiento del algoritmo. Para el experimento se decidió usar diferentes matrices que se encuentren en un rango de tamaño entre 3 y 100, con valores entre 1 y 10. El objetivo no se basa precisamente en medir la velocidad de solución del algoritmo a medida que el tamaño de la matriz cuadrada aumenta, sino medir la precisión con la cual responde sobre cual es el tamaño del máximo camino creciente posible y a su vez, la impresión de la secuencia de números naturales que permite llegar a dicho camino máximo.

Para comprobar y corrobar la respuesta obtenida por el algoritmo, se decidió analizar visualmente la respuesta probando con matrices que no tuvieran un tamaño excesivo, con el fin de poder identificar fácilmente el camino mas largo posible y los elementos crecientes que lo constituyen.

Experimento 1, Matriz $M : \mathbb{R}^{(4*4)}$ (Matriz del enunciado del Taller 4)

Resultado en repl.it



Experimento 2, Matriz $M : \mathbb{R}^{(23*23)}$
Resultado en repl.it

```
> sh -c make -s
> ./main
Tamaño de la matriz: 23x23
6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9
2 0 2 3 7 5 9 2 2 8 9 7 3 6 1 2 9 3 1 9 4 7 8
4 5 0 3 6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7
4 5 2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9 2 0 6
8 9 2 6 6 4 9 5 0 4 8 7 1 7 2 7 2 2 6 1 0 6 1
5 9 4 9 0 9 1 7 7 1 1 5 9 7 7 6 7 3 6 5 6 3 9
4 8 1 2 9 3 9 0 8 8 5 0 9 6 3 8 5 6 1 1 5 9 8
4 8 1 0 3 0 4 4 4 4 7 6 3 1 7 5 9 6 2 1 7 8 5
7 4 1 8 5 9 7 5 3 8 8 3 1 8 9 6 4 3 3 3 8 6 0
4 8 8 8 9 7 7 6 4 3 0 3 0 9 2 5 4 0 5 9 4 6 9
2 2 4 7 7 5 4 8 1 2 8 9 3 6 8 0 2 1 0 5 1 1 0
8 5 0 6 4 6 2 5 8 6 2 8 4 7 2 4 0 6 2 9 9 0 8
1 3 1 1 0 3 4 0 3 9 1 9 6 9 3 3 8 0 5 6 6 4 0
0 4 6 2 6 7 5 6 9 8 7 2 8 2 9 9 6 0 2 7 6 1 3
2 1 5 9 9 1 4 9 1 0 7 5 8 7 0 4 8 0 4 2 9 6 1
0 4 2 2 2 0 5 5 2 9 0 2 8 3 8 0 4 0 9 1 9 6 2
5 4 4 9 9 3 6 0 5 0 2 9 4 3 5 1 7 4 3 1 4 6 9
4 2 2 6 4 1 2 8 8 9 2 8 8 8 6 8 3 8 3 3 3 8 0
4 7 6 8 9 0 6 8 7 9 0 3 3 3 7 3 2 6 5 2 6 5 8
7 9 6 0 4 1 0 4 8 7 0 8 6 2 4 7 9 3 9 2 8 3 0
1 7 8 9 1 5 4 9 2 5 7 4 9 9 4 5 9 3 5 7 0 8 1
9 9 7 8 2 5 3 4 9 0 2 0 1 9 6 2 1 2 0 7 3 1 1
9 0 5 6 7 7 4 0 6 4 7 4 8 5 8 2 6 0 6 6 4 6 2

Camino creciente mas largo: 4
--Secuencia que genera el camino creciente mas largo--
4 5 6 7
>
```

Experimento 3, Matriz $M : \mathbb{R}^{(35*35)}$

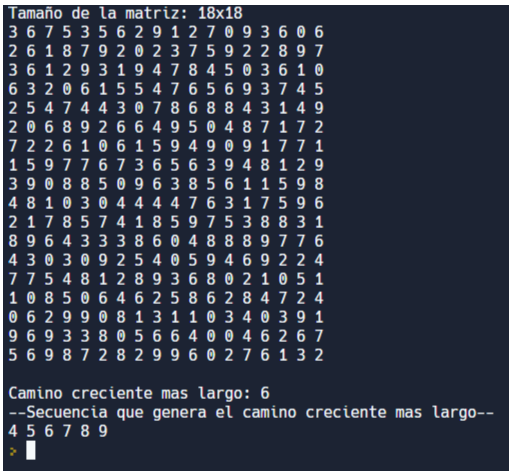
Resultado en repl.it

```
Tamaño de la matriz: 35x35
3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6 2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9
7 3 6 1 2 9 3 1 9 4 7 8 4 5 0 3 6 1 0 6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7
4 5 2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9 2 0 6 8 9 2 6 6 4 9 5 0 4 8 7
1 7 2 7 2 2 6 1 0 6 1 5 9 4 9 0 9 1 7 7 1 1 5 9 7 7 6 7 3 6 5 6 3 9 4
8 1 2 9 3 9 0 8 8 5 0 9 6 3 8 5 6 1 1 5 9 8 4 8 1 0 3 0 4 4 4 4 7 6 3
1 7 5 9 6 2 1 7 8 5 7 4 1 8 5 9 7 5 3 8 8 3 1 8 9 6 4 3 3 3 8 6 0 4 8
8 8 9 7 7 6 4 3 0 3 0 9 2 5 4 0 5 9 4 6 9 2 2 4 7 7 5 4 8 1 2 8 9 3 6
8 0 2 1 0 5 1 1 0 8 5 0 6 4 6 2 5 8 6 2 8 4 7 2 4 0 6 2 9 9 0 8 1 3 1
1 0 3 4 0 3 9 1 9 6 9 3 3 8 0 5 6 6 4 0 0 4 6 2 6 7 5 6 9 8 7 2 8 2 9
9 6 0 2 7 6 1 3 2 1 5 9 9 1 4 9 1 0 7 5 8 7 0 4 8 0 4 2 9 6 1 0 4 2 2
2 0 5 5 2 9 0 2 8 3 8 0 4 0 9 1 9 6 2 5 4 4 9 9 3 6 0 5 0 2 9 4 3 5 1
7 4 3 1 4 6 9 4 2 2 6 4 1 2 8 8 9 2 8 8 8 6 8 3 8 3 3 3 8 0 4 7 6 8 9
0 6 8 7 9 0 3 3 3 7 3 2 6 5 2 6 5 8 7 9 6 0 4 1 0 4 8 7 0 8 6 2 4 7 9
3 9 2 8 3 0 1 7 8 9 1 5 4 9 2 5 7 4 9 9 4 5 9 3 5 7 0 8 1 9 9 7 8 2 5
3 4 9 0 2 0 1 9 6 2 1 2 0 7 3 1 1 9 0 5 6 7 7 4 0 6 4 7 4 8 5 8 2 6 0
6 6 4 6 2 8 9 6 0 7 0 1 0 1 3 7 7 2 6 4 5 2 0 2 9 0 9 9 4 5 1 0 3 7 8
8 6 0 4 6 7 6 0 9 7 5 9 7 8 5 3 3 8 3 7 9 3 7 8 7 4 1 9 0 9 8 8 5 8 4
3 7 1 3 8 0 9 7 9 9 3 2 4 3 7 1 2 2 0 2 1 7 5 1 7 4 1 7 1 1 1 7 0 4 0
8 5 1 6 6 2 1 9 6 4 8 0 8 1 0 2 2 9 7 5 6 4 6 3 7 9 7 4 9 1 7 0 6 0 8
5 3 9 4 1 5 4 1 5 5 4 9 8 3 8 5 2 2 2 7 0 3 4 6 3 6 3 3 4 4 3 9 7 2 5
8 9 0 2 4 7 6 5 7 1 3 3 3 8 5 1 0 8 7 6 3 5 0 8 0 4 1 1 3 5 9 3 4 1 5
0 8 3 5 6 5 9 9 0 7 6 3 7 6 1 5 0 6 5 0 8 1 2 2 6 9 1 0 4 2 7 4 0 3 2
8 0 3 7 0 0 3 4 9 2 7 4 2 5 2 4 4 3 8 6 0 8 9 2 2 3 1 8 3 4 2 2 4 5 1
7 5 7 1 6 9 8 1 3 3 3 7 9 8 6 7 0 4 8 4 8 1 6 8 5 2 1 9 9 6 0 6 4 9 9
0 8 9 3 1 4 8 9 6 7 7 3 7 1 2 2 1 5 0 9 0 4 0 1 3 7 4 1 1 3 0 3 2 1 7
5 6 5 4 2 2 1 7 2 4 1 6 5 7 8 5 9 2 7 3 6 4 7 9 7 2 2 1 6 3 0 2 1 5 6
5 0 0 3 4 6 6 0 2 5 8 9 5 0 6 0 8 3 7 8 2 1 0 5 8 5 5 0 7 3 8 2 3 0 7
7 7 6 9 1 1 9 0 8 9 8 8 0 3 7 0 6 9 2 1 7 7 9 7 6 2 7 1 5 8 8 4 5 6 5
6 8 4 8 8 3 8 7 5 2 4 5 0 5 7 3 2 7 2 1 3 4 9 4 1 9 5 7 4 1 2 2 1 6 2
0 2 2 9 7 4 3 5 6 1 4 0 3 1 2 7 7 9 8 1 0 7 6 0 1 0 2 5 1 9 7 1 3 1 2
0 6 8 7 4 9 2 4 4 5 9 1 2 8 9 6 0 6 4 0 9 4 5 4 6 6 3 9 9 5 2 1 3 0 9
9 1 3 4 5 8 5 7 3 3 8 9 5 5 3 8 6 0 3 3 6 1 8 7 0 5 1 1 0 3 0 0 4 5 4
2 4 9 9 9 4 9 8 1 6 3 9 3 3 4 6 1 5 6 1 7 2 2 9 2 8 1 2 4 7 8 6 3 7 7
2 3 7 2 5 3 7 6 8 1 3 6 4 0 3 5 8 5 0 9 9 8 0 2 4 9 0 1 2 0 0 6 3 9 0
8 3 8 7 3 1 0 0 5 0 3 1 8 0 1 7 9 1 0 3 5 9 4 8 4 6 9 2 9 0 3 0 3 3 7

Camino creciente mas largo: 5
--Secuencia que genera el camino creciente mas largo--
2 3 4 5 6
```

Experimento 4, Matriz $M : \mathbb{R}^{(18*18)}$

Resultado en repl.it


Tamaño de la matriz: 18x18
3 6 7 5 3 5 6 2 9 1 2 7 0 9 3 6 0 6
2 6 1 8 7 9 2 0 2 3 7 5 9 2 2 8 9 7
3 6 1 2 9 3 1 9 4 7 8 4 5 0 3 6 1 0
6 3 2 0 6 1 5 5 4 7 6 5 6 9 3 7 4 5
2 5 4 7 4 4 3 0 7 8 6 8 8 4 3 1 4 9
2 0 6 8 9 2 6 6 4 9 5 0 4 8 7 1 7 2
7 2 2 6 1 0 6 1 5 9 4 9 0 9 1 7 7 1
1 5 9 7 7 6 7 3 6 5 6 3 9 4 8 1 2 9
3 9 0 8 8 5 0 9 6 3 8 5 6 1 1 5 9 8
4 8 1 0 3 0 4 4 4 4 7 6 3 1 7 5 9 6
2 1 7 8 5 7 4 1 8 5 9 7 5 3 8 8 3 1
8 9 6 4 3 3 3 8 6 0 4 8 8 8 9 7 7 6
4 3 0 3 0 9 2 5 4 0 5 9 4 6 9 2 2 4
7 7 5 4 8 1 2 8 9 3 6 8 0 2 1 0 5 1
1 0 8 5 0 6 4 6 2 5 8 6 2 8 4 7 2 4
0 6 2 9 9 0 8 1 3 1 1 0 3 4 0 3 9 1
9 6 9 3 3 8 0 5 6 6 4 0 0 4 6 2 6 7
5 6 9 8 7 2 8 2 9 9 6 0 2 7 6 1 3 2
Camino creciente mas largo: 6
--Secuencia que genera el camino creciente mas largo--
4 5 6 7 8 9
>