Taller 4: Secuencia creciente en una matriz cuadrada

Alejandro Morales Contreras[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
a.moralesc@javeriana.edu.co

29 de septiembre de 2022

Índice

1. Enunciado del problema

Escribir, implementar en C++ y hacer experimentos sobre un algoritmo basado en programación dinámica que resuelva el siguiente problema:

Dada una matriz cuadrada natural de tamaño NxN, que contiene los números únicos en el rango [1,NxN], 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.

2. Formalización del problema

Sea $A \in \mathbb{N}^{n \times n}$ una matriz cuadrada compuesta por los números naturales únicos desde 1 hasta n^2 ($n \in \mathbb{N} \land a_{ij} \in [1, n^2]\ \forall i \in [1, n]\ \forall j \in [1, n]$), encontrar la secuencia más larga de vecinos que cumplen la relación de orden parcial $<$ y cuya diferencia es equivalente a 1 entre elementos adyacentes.

Defínase un vecino en la matriz como un par de elementos $a_{ij}, a_{kl} \in A$ cuya distancia relativa entre ambos es igual a 1. Esto es, que $|i - k| + |j - l| = 1$.

Entonces, una secuencia de vecinos ordenados se puede expresar como:

$$N = \langle a_{ij} \in A \mid N_m = N_{m+1} + 1\ \forall m \in [1, |N| - 1] \land N_m = a_{ij}, N_{m+1} = a_{kl} \implies |i - j| + |k - l| = 1 \rangle$$

Encontrar la secuencia más larga consiste entonces encontrar la secuencia N' que **maximiza** la cantidad de elementos. Es decir, la secuencia cuya cardinalidad $|N|$ es mayor a la de las demás.

2.1. Definición del problema "secuencia más larga de vecinos ordenados"

El problema de encontrar la secuencia más larga de vecinos ornedados se define así:

- Entradas:
  - $A \in \mathbb{N}^{n \times n} \mid a_{ij} \in [1, n^2]\ \forall i, j \in [1, n]$
- Salidas:
  - $N' = \langle a_{ij} \in A \mid N_m = N_{m+1} + 1\ \forall m \in [1, |N| - 1] \land N_m = a_{ij}, N_{m+1} = a_{kl} \Rightarrow |i - j| + |k - l| = 1 \rangle$

3. Algoritmo de solución

3.1. Idea general de la solución

Encontrar la secuencia de vecinos ordenados más larga puede ser abordado como una tarea de fuerza bruta, donde se define un $i \in [1, n]$ y un $j \in [1, n]$ que permiten recorrer todas las celdas de la matriz $A \in \mathbb{N}^{n \times n}$. El problema se convierte en, por cada celda, explorar sus vecinos que cumplan las condiciones establecidas en las 4 direcciones posibles. Este proceso se repite para todas las celdas, buscando aquel camino que recorra la mayor cantidad de celdas.

Explorar los vecinos de una celda de acuerdo a las condiciones establecidas puede ser visto como, para un $i, j \in [1, n]$, moverse hacia cualquiera de las cuatro posibles direcciones que cumplan la condición:

- $A[i, j] + 1 = A[i - 1, j]$, moverse una celda hacia arriba siempre y cuando sea 1 mayor y $1 < i$, porque $i - 1$ no es una posición válida cuando $i = 1$.
- $A[i, j] + 1 = A[i + 1, j]$, moverse una celda hacia abajo siempre y cuando sea 1 mayor y $i < n$, porque $i + 1$ no es una posición válida cuando $i = n$.
- $A[i, j] + 1 = A[i, j - 1]$, moverse una celda hacia la izquierda siempre y cuando sea 1 mayor y $1 < j$, porque $j - 1$ no es una posición válida cuando $j = 1$.
- $A[i, j] + 1 = A[i, j + 1]$, moverse una celda hacia la derecha siempre y cuando sea 1 mayor y $j < n$, porque $j + 1$ no es una posición válida cuando $j = n$.

El ejercicio consitiría entonces en repetir este proceso desde todo i, j inicial, buscando aquella celda inicial que maximiza la cantidad de celdas visitadas.

Debido a que el objetivo del problema es encontrar cuál es la secuencia más larga y no su tamaño, es necesario implementar *backtracking*. Es por esto que, para facilitar esta implementación, se modifica un poco esta idea de solución a encontrar la secuencia ordenada al revés. Es decir, empezar considerando la celda inicial i, j como la mayor, y buscar el camino de vecinos como aquellos que sean 1 menor.

Entonces, $N_{i,j}$ es el tamaño de la secuencia más larga de vecinos ordenados descendemente empezando en la celda $A[i,j]$ definido así:

$$N_{i,j} = \begin{cases} max & \begin{cases} N_{i-1,j}+1 & \text{if } 1 < i \wedge A[i,j]-1 = A[i-1,j], \\ N_{i+1,j}+1 & \text{if } i < n \wedge A[i,j]-1 = A[i+1,j], \\ N_{i,j-1}+1 & \text{if } 1 < j \wedge A[i,j]-1 = A[i,j-1], \\ N_{i,j+1}+1 & \text{if } j < n \wedge A[i,j]-1 = A[i,j+1] \end{cases} \\ 1 & \text{else} \end{cases}$$

donde se hace necesario probar todas las posibles permutaciones de celdas iniciales $i, j \in [1, n]$ y maximizar sobre estas también para determinar aquella que consigue el camino más largo.

Finalmente, el problema se puede resolver por fuerza bruta y recursión. Después, se puede optimizar con una solución con programación dinámica por tabla de memoización (de 2 dimensiones) que reduzca el cálculo de operaciones repetidas. Luego, se implementa el backtracking para encontrar cuál es el camino de vecinos que se recorre desde la celda que se determina como la mayor hasta la menor de esa secuencia.

Es decir, para implementar el backtracking, se utiliza una tabla de 2 dimensiones que almacena los índices de la celda (tupla de índices i, j) a la que me tengo que mover después para seguir la secuencia encontrada. Después de realizar el backtracking y generar la secuencia, esta se debe invertir, de acuerdo a que el algoritmo calculó la secuencia de forma descendiente.

3.2. Escritura del algoritmo

Donde:

- MAX(n, m) recibe 2 números y retorna el máximo de ambos

Algoritmo 1 Calcular secuencia más larga de vecinos ordenados

```
1:  procedure LongestSortedNeighbors(A)
2:      let M[1..|A|, 1..|A|] be a matrix filled with 0
3:      let B[1..|A|, 1..|A|] be a matrix of tuples
4:      for i ← 1 to |A| do
5:          for j ← 1 to |A| do
6:              B[i, j] ← (i, j)
7:          end for
8:      end for
9:      max ← 0 ∧ max_pos ← (0, 0)
10:     for i ← 1 to |A| do
11:         for j ← 1 to |A| do
12:             max ← Max(max, LongestSortedNeighbors_Aux(A, i, j, M, B))
13:             if max = M[i, j] then
14:                 max_pos ← (i, j)
15:             end if
16:         end for
17:     end for
18:     let N be an array
19:     while B[max_pos_1, max_pos_2] ≠ max_pos do
20:         N ← A[max_pos_1, max_pos_2] ∪ N
21:         max_pos ← B[max_pos_1, max_pos_2]
22:     end while
23:     N ← A[max_pos_1, max_pos_2] ∪ N
24:     return N
25: end procedure
```

Algoritmo 2 Calcular secuencia más larga de vecinos ordenados a partir de la posición

```
1: procedure LongestSortedNeighbors_Aux(A, i, j, M, B)
2:     if M[i, j] ≠ 0 then
3:         return M[i, j]
4:     end if
5:     max ← 0
6:     if 1 < i ∧ A[i, j] − 1 = A[i − 1, j] then
7:         max ← Max(LongestSortedNeighbors_Aux(A, i − 1, j, M, B))
8:         if max = M[i − 1, j] then
9:             B[i, j] ← (i − 1, j)
10:        end if
11:    end if
12:    if i < |A| ∧ A[i, j] − 1 = A[i + 1, j] then
13:        max ← Max(LongestSortedNeighbors_Aux(A, i + 1, j, M, B))
14:        if max = M[i + 1, j] then
15:            B[i, j] ← (i + 1, j)
16:        end if
17:    end if
18:    if 1 < j ∧ A[i, j] − 1 = A[i, j − 1] then
19:        max ← Max(LongestSortedNeighbors_Aux(A, i, j − 1, M, B))
20:        if max = M[i, j − 1] then
21:            B[i, j] ← (i, j − 1)
22:        end if
23:    end if
24:    if j < |A| ∧ A[i, j] − 1 = A[i, j + 1] then
25:        max ← Max(LongestSortedNeighbors_Aux(A, i, j + 1, M, B))
26:        if max = M[i, j + 1] then
27:            B[i, j] ← (i, j + 1)
28:        end if
29:    end if
30:    M[i, j] ← max + 1
31:    return M[i, j]
32: end procedure
```

4. Análisis experimental

En esta sección se presentarán algunos experimentos para probar el algoritmo solución presentado con distintas configuraciones de matrices.

4.1. Matrices por archivo

Acá se presentan los experimentos cuando el algoritmo se ejecuta con matrices preconfiguradas en un archivo.

4.1.1. Protocolo

1. Definir dentro de un archivo CSV, cada configuración de matrices a probar. La primera línea corresponde al tamaño n de la matriz, seguido de n filas con n elementos únicos
2. Cargar el archivo en memoria
3. Imprimir en pantalla la secuencia de vecinos ordenados más larga encontrada
4. Se comentarán las conclusiones necesarias

4.2. Matrices aleatorias

Acá se presentan los experimentos cuando el algoritmo se ejecuta con matrices aleatorias.

4.2.1. Protocolo

1. Definir un número n correspondiente al tamaño de la matriz
2. Generar una matriz $n \times n$ con los números naturales desde 1 hasta n^2 configurados de manera aleatoria
3. Imprimir en pantalla la secuencia de vecinos ordenados más larga encontrada
4. Se comentarán las conclusiones necesarias

5. Resultados de los experimentos

El algoritmo de solución se implementa en Julia y los experimentos se realizan siguiendo los protocolo definidos en la secciones 4.1.1 y 4.2.1.

5.1. Matrices por archivo

5.1.1. Experimento matrices por archivo 1

Entrada

- $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 8 & 7 & 6 & 5 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$

Salida

- $N = \langle 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 \rangle$

Analizando la matriz, se evidencia que esta es la secuencia de vecinos ordenados más larga, empezando en la posición $(1, 1)$ hasta la posición $(3, 4)$.

5.1.2. Experimento matrices por archivo 2

Entrada

- $A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 20 & 21 & 22 & 23 & 24 & 7 \\ 19 & 32 & 33 & 34 & 25 & 8 \\ 18 & 31 & 36 & 35 & 26 & 9 \\ 17 & 30 & 29 & 28 & 27 & 10 \\ 16 & 15 & 14 & 13 & 12 & 11 \end{bmatrix}$

Salida

- $N = \langle 1, 2, \cdots, 36 \rangle$

La matriz se configura como un espiral empezando en la posición $(1, 1)$ hasta la posición $(4, 3)$, y la respuesta del algoritmo es efectivamente la secuencia de vecinos ordenados más larga (todos los elementos de la matriz).

5.1.3. Experimento matrices por archivo 3

Entrada

- $A = \begin{bmatrix} 1 & 2 & 25 & 6 & 7 \\ 23 & 3 & 4 & 5 & 8 \\ 9 & 10 & 11 & 12 & 13 \\ 18 & 21 & 22 & 19 & 20 \\ 14 & 15 & 16 & 17 & 24 \end{bmatrix}$

Salida

- $N = \langle 1, 2, 3, 4, 5, 6, 7, 8 \rangle$

Analizando la matriz, se evidencia que esta es la secuencia de vecinos ordenados más larga, empezando en la posición $(1, 1)$ hasta la posición $(2, 5)$.

5.1.4. Experimento matrices por archivo 4

Entrada

- $A \in \mathbb{N}^{100\times100} = \begin{bmatrix} 1 & 2 & \cdots & 100 \\ 200 & 199 & \cdots & 101 \\ \vdots & \vdots & \ddots & \vdots \\ 10000 & 9999 & \cdots & 9901 \end{bmatrix}$

Salida

- $N = \langle 1, 2, \cdots, 10000 \rangle$

La matriz se configura como una matriz semiordenada, donde cada fila impar está en orden ascendente y cada fila par está en orden descendente. Siguiendo este efecto, la secuencia más larga será aquella que reúna todos los 10000 elementos. La respuesta del algoritmo es efectivamente esta.

5.1.5. Experimento matrices por archivo 5

Entrada

- $A \in \mathbb{N}^{100\times100} = \begin{bmatrix} 1 & 2 & \cdots & 100 \\ 101 & 102 & \cdots & 200 \\ \vdots & \vdots & \ddots & \vdots \\ 9901 & 9902 & \cdots & 10000 \end{bmatrix}$

Salida

- $N = \langle 9901, 9902, \cdots, 10000 \rangle$

La matriz se configura como una matriz ordenada, donde cada fila está en orden ascendente. Siguiendo este efecto, la secuencia más larga puede ser cualquier fila de 100 elementos. La respuesta del algoritmo es la última fila, la cual es válida.

5.2. Matrices aleatorias

Para estos experimentos, se define el n y el programa genera la matriz aleatoria, para después encontrar la secuencia. No se hace necesario generar muchos de estos experimentos, ya que las matrices aleatorias de mayor tamaño son a) difíciles de analizar y b) no tienen una configuración aparente.

5.2.1. Experimento matrices aleatorias 1

- $n = 2$, para el cual el programa genera la siguiente matriz:
- $A = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$

Salida

- $N = \langle 1, 2, 3, 4 \rangle$

Analizando la matriz, se evidencia que esta es la secuencia de vecinos ordenados más larga, empezando en la posición $(2, 2)$ hasta la posición $(1, 2)$.

5.2.2. Experimento matrices aleatorias 2

- $n = 4$, para el cual el programa genera la siguiente matriz:
- $A = \begin{bmatrix} 2 & 13 & 15 & 16 \\ 10 & 8 & 4 & 14 \\ 11 & 5 & 3 & 7 \\ 12 & 6 & 9 & 1 \end{bmatrix}$

Salida

- $N = \langle 10, 11, 12 \rangle$

Analizando la matriz, se evidencia que esta es la secuencia de vecinos ordenados más larga, empezando en la posición $(2, 1)$ hasta la posición $(4, 1)$.

5.2.3. Experimento matrices aleatorias 3

- $n = 8$, para el cual el programa genera la siguiente matriz:
- $A = \begin{bmatrix} 40 & 56 & 1 & 51 & 59 & 50 & 27 & 42 \\ 23 & 5 & 37 & 14 & 18 & 54 & 61 & 21 \\ 62 & 9 & 57 & 64 & 6 & 34 & 10 & 8 \\ 25 & 49 & 29 & 52 & 11 & 24 & 55 & 31 \\ 35 & 43 & 20 & 39 & 36 & 41 & 12 & 58 \\ 17 & 44 & 45 & 2 & 53 & 33 & 19 & 3 \\ 30 & 63 & 46 & 32 & 22 & 28 & 15 & 4 \\ 38 & 47 & 16 & 26 & 60 & 13 & 48 & 7 \end{bmatrix}$

Salida

- $N = \langle 43, 44, 45, 46 \rangle$

Analizando la matriz, se evidencia que esta es la secuencia de vecinos ordenados más larga, empezando en la posición $(5, 2)$ hasta la posición $(7, 3)$.