Secuencia más larga de vecinos que están ordenados

Jorge Luis Esposito Albornoz[1] Juan Sebastián Herrera Guaitero[1]

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

29 de septiembre de 2022

Resumen

En este documento se presenta la formalización del problema de encontrar la secuencia más larga de vecinos que están ordenados. **Palabras clave:** matriz, adyacentes.

Índice

1. Formalización problema

El problema de encontrar la secuencia mas larga de vecinos que estan ordenados, es un problema muy utilizado en el mundo empresarial para realizar entrevistas de conocimiento a los candidatos. Su dificultad particular radica en la solucíon del problema, los primeros acercamientos a la solucíon suelen ser a través de la recursión ingenua, sin embargo, utilizar programación dinámica es una buena opción.

1.1. Definición del 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 que tienen una diferencia de $+1$.

El problema de la secuencia mas larga de vecinos ordenados se define apartir de:

1. Una matriz M de elementos $a \in \mathbb{Z}$.

Generar una secuencia S cuyos elementos cumplan con una relacíon $a < b$.

- **Entradas:**
  - $M^{nxn} \mid \forall M_{ij} \in \mathbb{Z}$
- **Salidas:**
  - $S = \langle s_i \in M \rangle \mid \forall i \in M \; s_i \leq s_{i+1}$

2. Algoritmo de solución

2.1. Idea solución

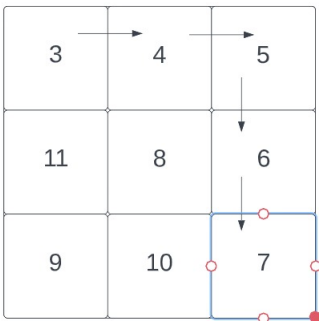

Figura 1: Matriz inicial

Se planteo la siguiente ecuación como posible solución del problema:

$$M_{ij} = \begin{cases} 0 & \text{if } |M| = 0 \\ max = \begin{cases} M_{i-1j} & \text{if } \; i > 1 \; \wedge \; M_{ij} + 1 = M_{i-1j} \\ M_{i+1j} & \text{if } \; i < |M| \; \wedge \; M_{ij} + 1 = M_{i+1j} \\ M_{ij-1} & \text{if } \; j > 1 \; \wedge \; M_{ij} + 1 = M_{ij-1} \\ M_{ij+1} & \text{if } \; j < |M| \; \wedge \; M_{ij} + 1 = M_{ij+1} \end{cases} \end{cases}$$

Teniendo en cuenta que para cada posición ij dentro de la matriz pueden existir valores adyacentes se busca obtener el valor máximo.

2.2. Escritura del algorítmo

Algoritmo 1 Max num

```
1: function max(a, b)
2:     if a > b then
3:         return a
4:     else
5:         return b
6:     end if
7: end function
```

Algoritmo 2 aux

```
1: procedure max(T, M, B, i, j)
2:     q ← 0
3:     if |T| = 0 then
4:         return 0
5:         if M_ij ≠ 0 then
6:             return M_ij
7:         else
8:             if i > 0 ∧ T_ij + 1 = T_{i-1 j} then
9:                 q ← MAX(q, AUX(T, M, B, i − 1, j))
10:                if q = M_{i-1 j} then
11:                    B_ij ← pair(i − 1, j)          ▷ Pair pareja de elementos
12:                end if
13:            end if
14:            if i < |T| ∧ T_ij + 1 = T_{i+1 j} then
15:                q ← MAX(q, AUX(T, M, B, i + 1, j))
16:                if q = M_{i-1 j} then
17:                    B_ij ← pair(i + 1, j)          ▷ Pair pareja de elementos
18:                end if
19:            end if
20:            if j > 0 ∧ T_ij + 1 = T_{ij-1} then
21:                q ← MAX(q, AUX(T, M, B, i, j − 1))
22:                if q = M_{ij-1} then
23:                    B_ij ← pair(i, j − 1)          ▷ Pair pareja de elementos
24:                end if
25:            end if
26:            if j < |T| ∧ T_ij + 1 = T_{ij+1} then
27:                q ← MAX(q, AUX(T, M, B, i, j + 1))
28:                if q = M_{ij+1} then
29:                    B_ij ← pair(i, j + 1)          ▷ Pair pareja de elementos
30:                end if
31:            end if
32:            M_ij ← q + 1
33:        end if
34:        return M_ij
35:
```

Análisis de complejidad: O(N*N)
Invariante: La tabla se esta llenando correctamente.

Algoritmo 3 LSNBacktracking

```
1: procedure bactraking(T)
2:     let M be a matrix filled with 0
3:     let B be a matrix filled with pairs of integers
4:     let rt be an array
5:     let current be a pair of integers
6:     q ← 0
7:     for i ← 1 to |T| do
8:         for j ← 1 to |T| do
9:             q ← MAX(q, AUX(T, M, B, i, j))
10:            if q = M_ij then
11:                let current be a pair that stored i and j
12:            end if
13:        end for
14:    end for
15:    while B_current1current2 ≠ current do
16:        let rt add T_current1current2
17:        current ← B_current1current2
18:    end while
19:    let rt add T_current1current2
20:    return rt
21: end procedure=0
```

3. Análisis experimental

En esta sección encontrará el análisis experimental del problema de la secuencia mas larga de vecinos adyacentes con una diferencia de +1 en una matriz NxN.

3.1. Protocolo

1. Elegir uno de los archivos de la carpeta input para probar en el programa.
2. Pasar como parámetro de lanzamiento la ruta del archivo elegio.
3. Correr el algoritmo para obtener la secuencia más larga de vecinos ordenados.

3.2. Matrices definidas

3.2.1. Procedimiento

1. Se eligió como archivo de entrada el archivo input/test.csv que contiene el ejemplo dado en el enunciado.
2. Se procedió a compilar el archivo main.cpp de la siguiente manera:

$$g++ \; main.cpp - o \; main \; \&\& \; ./main \; ruta\_archivo.$$

3. Se obtuvo como resultado $2, 3, 4, 5, 6, 7, 8, 9, 10$.

3.2.2. Resultado

1. **Entrada:** test.csv

$$\begin{matrix} 4, & 4 & & \\ 10, & 16, & 15, & 12 \\ 9, & 8, & 7, & 13 \\ 2, & 5, & 6, & 14 \\ 3, & 4, & 1, & 11 \end{matrix}$$

2. Al leer el archivo la matriz queda guardada de la siguiente manera:

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

3. **Resultado esperado:** $2, 3, 4, 5, 6, 7, 8, 9, 10$.
4. **Resultado obtenido:** $2, 3, 4, 5, 6, 7, 8, 9, 10$.

3.3. Matrices aleatorias

3.3.1. Procedimiento

1. Ejecutar script de python de la siguiente manera:

$$py \;\; create\_files.py \;\; n \gg nombre\_archivo$$

n representa cualquiera número entero.

2. Elegir el archivo de entrada generado por el script de python.
3. Se procedió a compilar el archivo main.cpp de la siguiente manera:

$$g++ \; main.cpp - o \; main \;\&\&\; ./main \; ruta\_archivo.$$

4. Se obtuvo como resultado $3, 4$.

3.3.2. Resultado

1. **Entrada:** random5.csv

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

2. Al leer el archivo la matriz queda guardada de la siguiente manera:

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

3. **Resultado esperado:** $3, 4$.

4. **Resultado obtenido:** $3, 4$.

3.4. Análisis

Durante el camino a la solución logramos deducir que la complejidad temporal es $O(n * n)$, donde n es el número de filas y de columnas para nuestra caso en particular sabiendo que estamos trabajando con matrices cuadradas. Esto se debe a que estamos calculando los valores de todas las celdas una vez y hay un total de $n * n$ celdas.

De igual modo, la complejidad espacial es $O(n * n)$, puesto que estamos almacenando los valores de todas las celdas, lo que ocupará un espacio $O(n * n)$ y la propia matriz ocupará otro espacio $O(n * n)$. Por tanto, la complejidad espacial total es $O(2(n * n))$, pero omitiendo las constantes tenemos $O(n * n)$.