

# Escritura del problema de secuencia creciente en una matriz cuadrada

María Camila Aguirre Collante<sup>1</sup>      Jessica Tatiana Naizaque Guevara<sup>1</sup>

<sup>1</sup>Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana  
Bogotá, Colombia  
{aguirrec.mcamila, j.naizaque}@javeriana.edu.co

29 de septiembre de 2022

## Resumen

En este documento se presenta la formalización del problema de hallar una secuencia creciente dentro de una matriz cuadrada, junto con la descripción de los algoritmos que lo solucionan, los cuales surgen a partir de la estrategia de “programación dinámica”. Adicionalmente, se presenta un análisis experimental para cada uno de los algoritmos. **Palabras clave:** matriz cuadrada, secuencia, elementos únicos, algoritmo, formalización.

## Índice

<b>1. Introducción</b>	<b>1</b>
<b>2. Formalización del problema</b>	<b>2</b>
2.1. Definición del problema de “secuencia creciente en una matriz cuadrada” . . . . .	2
<b>3. Algoritmos de solución</b>	<b>2</b>
3.1. Algoritmo “evidente” . . . . .	2
3.2. Algoritmo “memoización” . . . . .	3
3.3. Algoritmo con “backtracking” . . . . .	3
<b>4. Análisis experimental</b>	<b>4</b>
4.1. Matrices ingresadas . . . . .	4
4.2. Matrices creadas de manera aleatoria . . . . .	7
4.3. Resultados obtenidos para cada caso . . . . .	7
4.3.1. Matrices ingresadas . . . . .	8
4.3.2. Matrices creadas de manera aleatoria . . . . .	8
<b>5. Conclusiones</b>	<b>9</b>

## 1. Introducción

En una matriz que contiene elementos únicos, es posible encontrar desde un punto, un camino de vecinos adyacentes que cumplan la condición de ser consecutivos. Sin embargo, una matriz podría tener más de un camino, por lo tanto, se buscaría conocer el más largo de estos. A lo largo de este informe se evidencia: la formalización del problema (sección 2), la escritura formal de los algoritmos (sección 3) y un análisis experimental de cada algoritmo (sección 4).

## 2. Formalización del problema

Dada una matriz cuadrada natural  $A : \mathbb{R}^{n \times n}$  donde  $n$  es el orden de la misma y donde  $A$  contiene los números únicos en el rango  $[1, n^2]$ , encontrar la secuencia más larga de vecinos que están ordenados y los elementos adyacentes en la matriz tienen una diferencia de  $\pm 1$ .

### 2.1. Definición del problema de “secuencia creciente en una matriz cuadrada”

Así, el problema de secuencia creciente en una matriz cuadrada se define a partir de:

1. Una matriz  $A = [a_{ij}]$ ,
2. donde  $i = 1, \dots, n \wedge j = 1, \dots, n$ , es decir,  $A$  es de orden  $n$ ,
3. además,  $a_{ij} \in \mathbb{N} \wedge 1 \leq a_{ij} \leq n \times n$

encontrar la secuencia creciente más larga de elementos adyacentes en la matriz  $A$ , los cuales están ordenados y tienen una diferencia de 1.

■ Entradas:

- $A = [a_{ij}] \mid (i = 1, \dots, n) \wedge (j = 1, \dots, n) \wedge (1 \leq a_{ij} \leq n \times n)$

■ Salidas:

- $L = \langle e_k \in A_n \rangle \mid (e_k \in \mathbb{N}) \wedge (e_{k+1} - e_k = 1) \wedge (e_k < e_{k+1})$

## 3. Algoritmos de solución

### 3.1. Algoritmo “evidente”

La idea de este algoritmo es: encontrar la longitud de la mayor secuencia con la cantidad de elementos consecutivos y adyacentes utilizando recurrencia para revisar cada una de las posiciones en la matriz.

---

**Algoritmo 1** Algoritmo “evidente”

---

```
1: procedure LONGESTSEQUENCE(matriz, i, j)
2:   recursion  $\leftarrow$  0
3:   n  $\leftarrow$  matriz.length
4:   if matriz[i, j] - matriz[i - 1, j] = 1  $\wedge$  i > 1 then
5:     recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i - 1, j)
6:   end if
7:   if matriz[i, j] - matriz[i, j + 1] = 1  $\wedge$  j + 1 < n then
8:     recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j + 1)
9:   end if
10:  if matriz[i, j] - matriz[i + 1, j] = 1  $\wedge$  i + 1 < n then
11:    recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i + 1, j)
12:  end if
13:  if matriz[i, j] - matriz[i, j - 1] = 1  $\wedge$  j > 1 then
14:    recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j - 1)
15:  end if
16:  return recursion + 1
17: end procedure
```

---

---

**Algoritmo 2** Algoritmo “evidente”

---

```
    procedure LONGESTSEQUENCE_AUX(matriz)
2:   actual  $\leftarrow$  0
      n  $\leftarrow$  matriz.length
4:   for i  $\leftarrow$  1 to n do
      for j  $\leftarrow$  1 to n do
6:       len  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j)
      if len > actual then
8:         actual  $\leftarrow$  len
      end if
10:    end for
    end for
12:    return actual
end procedure
```

---

### 3.2. Algoritmo “memoización”

La idea de este algoritmo es: llenar una tabla de memoización en la que en cada una de las posiciones almacene la longitud de la mayor secuencia con la cantidad de elementos consecutivos y adyacentes que termine en esta posición, esto, además de utilizar una tabla, presenta el uso de recurrencia para evaluar cada posición en la matriz.

---

**Algoritmo 3** Algoritmo “memoización”

---

```
1: procedure LONGESTSEQUENCE(matriz, i, j, M)
2:   recursion  $\leftarrow$  0
3:   n  $\leftarrow$  matriz.length
4:   if M[i, j]  $\neq$  0 then
5:     return M[i, j]
6:   end if
7:   if matriz[i, j] - matriz[i - 1, j] = 1  $\wedge$  i > 1 then
8:     recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i - 1, j, M)
9:   end if
10:  if matriz[i, j] - matriz[i, j + 1] = 1  $\wedge$  j + 1 < n then
11:    recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j + 1, M)
12:  end if
13:  if matriz[i, j] - matriz[i + 1, j] = 1  $\wedge$  i + 1 < n then
14:    recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i + 1, j, M)
15:  end if
16:  if matriz[i, j] - matriz[i, j - 1] = 1  $\wedge$  j > 1 then
17:    recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j - 1, M)
18:  end if
19:  M[i, j]  $\leftarrow$  recursion + 1
20:  return M[i, j]
21: end procedure
```

---

### 3.3. Algoritmo con “backtracking”

La idea de este algoritmo es: llenar una tabla de memoización en la que en cada una de las posiciones almacene la longitud de la mayor secuencia con la cantidad de elementos consecutivos y adyacentes que termine en esta posición. Además, hacer uso de una tabla de backtracking que almacene inicialmente una tupla con dos elementos primordiales: la columna y la fila de dicha posición. Y, que adicionalmente, durante el recorrido de las revisiones para cada una de las posiciones de la matriz, si se encuentra algún dato adyacente

---

**Algoritmo 4** Algoritmo “memoización”

---

```
procedure LONGESTSEQUENCE_AUX(matriz, M)
2:   actual  $\leftarrow$  0
   n  $\leftarrow$  matriz.lenght
4:   for i  $\leftarrow$  1 to n do
       for j  $\leftarrow$  1 to n do
6:         len  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j, M)
           if len > actual then
8:             actual  $\leftarrow$  len
           end if
10:    end for
    end for
12:   return actual
end procedure
```

---

con un valor menor en 1, se almacene su “dirección” o su posición con la tupla mencionada anteriormente.

## 4. Análisis experimental

En esta sección se presentarán algunos de los experimentos para confirmar la efectividad de los algoritmos presentados en la sección 3. Para realizar la prueba de cada uno de estos, se decidió probar los siguientes casos, con sus respectivos resultados esperados:

### 4.1. Matrices ingresadas

- Dada la matriz  $A_{4 \times 4}$

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

se espera obtener la secuencia: [2, 3, 4, 5, 6, 7, 8, 9, 10]

- Dada la matriz  $A_{8 \times 8}$

$$[htb!] \begin{bmatrix} 1 & 42 & 43 & 20 & 19 & 16 & 15 & 12 \\ 5 & 2 & 44 & 21 & 18 & 17 & 14 & 13 \\ 6 & 45 & 3 & 22 & 23 & 59 & 60 & 61 \\ 7 & 29 & 28 & 4 & 24 & 58 & 63 & 62 \\ 8 & 30 & 27 & 26 & 25 & 57 & 41 & 64 \\ 9 & 31 & 32 & 54 & 55 & 56 & 40 & 39 \\ 10 & 53 & 33 & 34 & 35 & 36 & 37 & 38 \\ 11 & 52 & 51 & 50 & 49 & 48 & 47 & 46 \end{bmatrix} \quad (2)$$

se espera obtener la secuencia: [12, 13, 14, 15, 16, ..., 35, 36, 37, 38, 39, 40, 41]

---

**Algoritmo 5** Algoritmo “backtracking”

---

```
1: procedure LONGESTSEQUENCE(matriz, i, j, M, B)
2:   recursion  $\leftarrow$  0
3:   maximo  $\leftarrow$  0
4:   n  $\leftarrow$  matriz.length
5:   if M[i, j]  $\neq$  0 then
6:     return M[i, j]
7:   end if
8:   if matriz[i, j] - matriz[i - 1, j] = 1  $\wedge$  i > 1 then
9:     recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i - 1, j, M, B)
10:    if maximo < recursion then
11:      maximo  $\leftarrow$  recursion
12:    end if
13:    if maximo = M[i - 1, j] then
14:      coordenada  $\leftarrow$  (i - 1, j)
15:      B[i, j]  $\leftarrow$  coordenada
16:    end if
17:  end if
18:  if matriz[i, j] - matriz[i, j + 1] = 1  $\wedge$  j + 1 < n then
19:    recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j + 1, M, B)
20:    if maximo < recursion then
21:      maximo  $\leftarrow$  recursion
22:    end if
23:    if maximo = M[i, j + 1] then
24:      coordenada  $\leftarrow$  (i, j + 1)
25:      B[i, j]  $\leftarrow$  coordenada
26:    end if
27:  end if
28:  if matriz[i, j] - matriz[i + 1, j] = 1  $\wedge$  i + 1 < n then
29:    recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i + 1, j, M, B)
30:    if maximo < recursion then
31:      maximo  $\leftarrow$  recursion
32:    end if
33:    if maximo = M[i + 1, j] then
34:      coordenada  $\leftarrow$  (i + 1, j)
35:      B[i, j]  $\leftarrow$  coordenada
36:    end if
37:  end if
38:  if matriz[i, j] - matriz[i, j - 1] = 1  $\wedge$  j > 1 then
39:    recursion  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j - 1, M, B)
40:    if maximo < recursion then
41:      maximo  $\leftarrow$  recursion
42:    end if
43:    if maximo = M[i, j - 1] then
44:      coordenada  $\leftarrow$  (i, j - 1)
45:      B[i, j]  $\leftarrow$  coordenada
46:    end if
47:  end if
48:  M[i, j]  $\leftarrow$  recursion + 1
49:  return M[i, j]
50: end procedure
```

---

---

**Algoritmo 6** Algoritmo “backtracking”

---

```
procedure LONGESTSEQUENCE_AUX(matriz, M, B)
2:   actual  $\leftarrow$  0
   n  $\leftarrow$  matriz.lenght
4:   for i  $\leftarrow$  0 to n do
     for j  $\leftarrow$  0 to n do
6:       len  $\leftarrow$  LONGESTSEQUENCE(matriz, i, j, M, B)
       if len > actual then
8:         coordenadaFin  $\leftarrow$  (i, j)
         actual  $\leftarrow$  len
10:      end if
     end for
12:   end for
   return actual
14: end procedure
```

---

■ Dada la matriz  $A_{16 \times 16}$

$$\begin{bmatrix} 1 & 2 & 49 & 71 & 111 & 112 & 113 & 114 & 115 & 116 \\ 117 & 118 & 119 & 120 & 81 & 52 & & & & \\ 109 & 3 & 73 & 72 & 193 & 194 & 195 & 196 & 197 & 198 \\ 199 & 200 & 201 & 202 & 82 & 53 & & & & \\ 110 & 4 & 5 & 137 & 249 & 250 & 251 & 252 & 253 & 254 \\ 255 & 256 & 248 & 247 & 83 & 54 & & & & \\ 8 & 7 & 6 & 138 & 139 & 140 & 141 & 142 & 143 & 144 \\ 145 & 146 & 147 & 148 & 84 & 55 & & & & \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 227 & 226 \\ 221 & 220 & 59 & 58 & 57 & 56 & & & & \\ 130 & 149 & 74 & 75 & 76 & 77 & 80 & 17 & 228 & 225 \\ 222 & 219 & 60 & 242 & 85 & 246 & & & & \\ 131 & 150 & 180 & 203 & 79 & 78 & 19 & 18 & 229 & 224 \\ 223 & 218 & 61 & 243 & 244 & 245 & & & & \\ 132 & 151 & 181 & 204 & 207 & 21 & 20 & 101 & 102 & 103 \\ 104 & 105 & 62 & 106 & 107 & 108 & & & & \\ 133 & 152 & 182 & 205 & 206 & 22 & 23 & 24 & 25 & 26 \\ 27 & 217 & 63 & 230 & 231 & 232 & & & & \\ 134 & 153 & 183 & 184 & 185 & 121 & 208 & 211 & 212 & 215 \\ 28 & 216 & 64 & 239 & 240 & 233 & & & & \\ 135 & 154 & 190 & 189 & 186 & 122 & 209 & 210 & 213 & 214 \\ 29 & 30 & 31 & 238 & 241 & 234 & & & & \\ 136 & 155 & 191 & 188 & 187 & 123 & 38 & 37 & 36 & 35 \\ 34 & 33 & 32 & 237 & 236 & 235 & & & & \\ 90 & 91 & 192 & 99 & 100 & 124 & 39 & 40 & 41 & 42 \\ 43 & 44 & 45 & 46 & 47 & 48 & & & & \\ 89 & 92 & 93 & 98 & 51 & 125 & 167 & 168 & 169 & 172 \\ 173 & 176 & 65 & 177 & 178 & 179 & & & & \\ 88 & 87 & 94 & 97 & 129 & 126 & 157 & 158 & 170 & 171 \\ 174 & 175 & 66 & 67 & 68 & 69 & & & & \\ 50 & 86 & 95 & 96 & 128 & 127 & 156 & 159 & 160 & 161 \\ 162 & 163 & 164 & 165 & 166 & 70 & & & & \end{bmatrix} \quad (3)$$

se espera obtener la secuencia:  $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \dots, 41, 42, 43, 44, 45, 46, 47, 48]$

- Dada la matriz  $A_{18 \times 18}$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & & \\ 86 & 87 & 100 & 101 & 102 & 103 & 104 & 105 & 106 & 107 \\ 108 & 109 & 110 & 111 & 112 & 113 & 114 & 19 & & \\ 85 & 88 & 99 & 204 & 205 & 206 & 207 & 208 & 209 & 210 \\ 211 & 212 & 213 & 214 & 145 & 144 & 115 & 20 & & \\ 84 & 89 & 98 & 203 & 244 & 243 & 242 & 241 & 240 & 239 \\ 238 & 237 & 236 & 215 & 146 & 143 & 116 & 21 & & \\ 83 & 90 & 97 & 202 & 245 & 292 & 293 & 294 & 295 & 296 \\ 271 & 270 & 235 & 216 & 147 & 142 & 117 & 22 & & \\ 82 & 91 & 96 & 201 & 246 & 291 & 312 & 313 & 324 & 297 \\ 272 & 269 & 234 & 217 & 148 & 141 & 118 & 23 & & \\ 81 & 92 & 95 & 200 & 247 & 290 & 311 & 314 & 323 & 298 \\ 273 & 268 & 233 & 218 & 149 & 140 & 119 & 24 & & \\ 80 & 93 & 94 & 199 & 248 & 289 & 310 & 315 & 322 & 299 \\ 274 & 267 & 232 & 219 & 150 & 139 & 120 & 25 & & \\ 79 & 78 & 77 & 198 & 249 & 288 & 309 & 316 & 321 & 300 \\ 275 & 266 & 231 & 220 & 151 & 138 & 121 & 26 & & \\ 60 & 61 & 76 & 197 & 250 & 287 & 308 & 317 & 320 & 301 \\ 276 & 265 & 230 & 221 & 152 & 137 & 122 & 27 & & \\ 59 & 62 & 75 & 196 & 251 & 286 & 307 & 318 & 319 & 302 \\ 277 & 264 & 229 & 222 & 153 & 136 & 123 & 28 & & \\ 58 & 63 & 74 & 195 & 252 & 285 & 306 & 305 & 304 & 303 \\ 278 & 263 & 228 & 223 & 154 & 135 & 124 & 29 & & \\ 57 & 64 & 73 & 194 & 253 & 284 & 283 & 282 & 281 & 280 \\ 279 & 262 & 227 & 224 & 155 & 134 & 125 & 30 & & \\ 56 & 65 & 72 & 193 & 254 & 255 & 256 & 257 & 258 & 259 \\ 260 & 261 & 226 & 225 & 156 & 133 & 126 & 31 & & \\ 55 & 66 & 71 & 192 & 191 & 190 & 189 & 188 & 187 & 186 \\ 185 & 184 & 183 & 182 & 157 & 132 & 127 & 32 & & \\ 54 & 67 & 70 & 171 & 172 & 173 & 174 & 175 & 176 & 177 \\ 178 & 179 & 180 & 181 & 158 & 131 & 128 & 33 & & \\ 53 & 68 & 69 & 170 & 169 & 168 & 167 & 166 & 165 & 164 \\ 163 & 162 & 161 & 160 & 159 & 130 & 129 & 34 & & \\ 52 & 51 & 50 & 49 & 48 & 47 & 46 & 45 & 44 & 43 \\ 42 & 41 & 40 & 39 & 38 & 37 & 36 & 35 & & \end{bmatrix} \quad (4)$$

se espera obtener la secuencia:  $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \dots, 321, 322, 323, 324]$

## 4.2. Matrices creadas de manera aleatoria

En este caso, el usuario es quien ingresa el tamaño de la matriz que se va a generar de manera aleatoria. A continuación, se presentan los tamaños ingresados:

- $A_{20 \times 20}$
- $A_{50 \times 50}$
- $A_{100 \times 100}$

## 4.3. Resultados obtenidos para cada caso

De acuerdo con lo descrito al inicio de esta sección, se ejecutaron los casos en el algoritmo *bottom-up* con *backtracking*, obteniendo así los siguientes resultados:

#### 4.3.1. Matrices ingresadas

- Considerando la Figura 1, es posible decir que para este caso, el algoritmo funciona correctamente.

La secuencia creciente más larga de la matriz es: [2, 3, 4, 5, 6, 7, 8, 9, 10]

Figura 1: Resultados para la matriz ingresada  $A_{4 \times 4}$

- Considerando la Figura 2, es posible decir que para este caso, el algoritmo funciona correctamente.

La secuencia creciente más larga de la matriz es: [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]

Figura 2: Resultados para la matriz ingresada  $A_{8 \times 8}$

- Considerando la Figura 3, es posible decir que para este caso, el algoritmo funciona correctamente.

La secuencia creciente más larga de la matriz es: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48]

Figura 3: Resultados para la matriz ingresada  $A_{16 \times 16}$

- Considerando la Figura 4, es posible decir que para este caso, el algoritmo funciona correctamente.

La secuencia creciente más larga de la matriz es: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324]

Figura 4: Resultados para la matriz ingresada  $A_{18 \times 18}$

#### 4.3.2. Matrices creadas de manera aleatoria

- Para la matriz  $A$  de orden 20, se obtuvo:

La secuencia creciente más larga de la matriz es: [331, 332]

Figura 5: Resultados para la matriz ingresada  $A_{20 \times 20}$

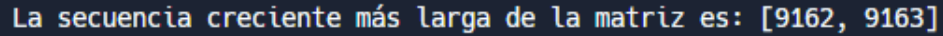
- Para la matriz  $A$  de orden 50, se obtuvo:

La secuencia creciente más larga de la matriz es: [2102, 2103]

Figura 6: Resultados para la matriz ingresada  $A_{50 \times 50}$

- Para la matriz  $A$  de orden 100, se obtuvo:





La secuencia creciente más larga de la matriz es: [9162, 9163]

Figura 7: Resultados para la matriz ingresada  $A_{100 \times 100}$

Teniendo en cuenta las figuras 5, 6 y 7 se puede evidenciar que al generar de manera aleatoria cada matriz el camino más largo que se produce tiene una longitud de dos.

## 5. Conclusiones

De acuerdo con los resultados obtenidos es posible indicar que el algoritmo funciona correctamente, pues en cada uno de los casos encuentra la secuencia y la muestra en el orden correcto. Además, cumple con las restricciones de diferencia de +1 entre los elementos y de asegurar que esta sea la secuencia más larga.

Adicional a esto, es posible identificar que el algoritmo funciona tanto para matrices propuestas por el usuario como para las que se crean aleatoriamente. Sin embargo, en estas últimas, la probabilidad de que salga un camino que pueda considerarse “largo” es bastante baja. Por lo tanto, la efectividad se puede evaluar realmente mediante las matrices propuestas por las estudiantes.

Finalmente, es posible decir que sin importar el tamaño de la matriz, el algoritmo responde correctamente y logra identificar la secuencia creciente más larga, por lo que podría decirse que tiene una alta eficacia y, del mismo modo, una alta eficiencia.