

# Práctica 8: Autómatas Celulares

## Implementación secuencial .

Karina Flores García y Gabriel Alejandro Herrera Gandarela.

**Abstract—In this practice we will implement an algorithm for the processing of a matrix by means of cellular automata in the programming language c.**

### I. OBJETIVOS.

- Procesar una matriz haciendo uso de autómatas celulares.
- Comprender y realizar la implementación de un átoma celular en el lenguaje C.
- Visualizar el autómata con un mapa de calor.

### II. INTRODUCCIÓN.

*Autómatas celulares.*

Los autómatas celulares son sistemas dinámicos discretos que se definen de una manera sencilla y poseen una gran complejidad, aparte de muchas otras bondades. Nos interesa presentar su calidad de instrumento relativamente sencillo para modelar sistemas complejos como gases fuera de equilibrio, ferromagnetos, reacciones químicas, sistemas inmunológicos, interacción entre genes biológicos, y ácidos nucleicos. Como veremos, un autómata celular es esencialmente una regla de evolución temporal sobre un conjunto discreto, es decir, dado un punto de este conjunto, presentamos una regla que le asocia otro punto del conjunto. El estudio de sistemas cuya construcción es sencilla pero cuyo comportamiento resulta extremadamente complicado ha llevado a una nueva disciplina de estudio llamada teoría de sistemas complejos, en la cual los métodos computacionales juegan un papel central.<sup>1</sup>

Un ejemplo de ellos, es el juego de la vida:



### III. ANÁLISIS DEL PROBLEMA.

Dada una población, por decirlo de alguna forma, necesitamos generar un programa capaz de identificar todas las formas posibles que se pueden dar. Para empezar analizaremos una secuencia se nos asignó y sobre eso, empezar a observar las múltiples formas que pueden ocurrir, es decir,

ir de lo particular a lo general. De primera instancia, es sencillo, sin embargo, lo complicado es que debemos obtener las múltiples formas que pueden ocurrir, entonces, debemos de llevar la lógica adecuadamente para que el objetivo de la práctica se cumpla. Una vez que nuestro programa funcione de manera correcta, debemos de pasarlo a un mapa de calor para que los resultados se visualicen con mayor facilidad.

### IV. IMPLEMENTACIÓN

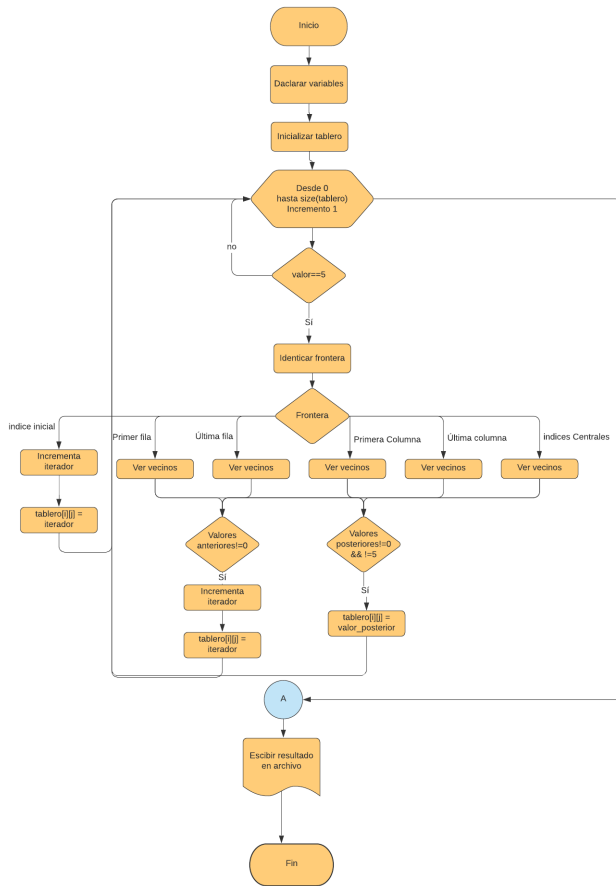
#### A. Desarrollo

Partiendo de la matriz inicial donde los valores inician con 5 y 0 los valores que se ignorarán, esto porque en el mapa de calor, es más visible dicho rango. Tendremos un iterador, este se encargará de modificar conforme encuentra nuevas secuencias, por ejemplo, cada vez que encuentra una nueva secuencia, incrementará en 1 para que tome valores distintos cada secuencia. Con esta lógica, debemos recorrer la matriz, cada vez que encuentre un valor distinto de cero, entonces cada vez que encuentre un valor distinto de 0, entonces, evaluará su posición y de acuerdo en dónde se encuentra, hará cierta acción. Es decir, habrá fronteras para evitar el desbordamiento, la primera es el nodo inicial, luego es la primera fila, la última fila, la primera columna, la última columna, último valor de la primer fila, primer valor de la última fila, el último nodo y el nodo central, el cual, tiene 8 vecinos a su alrededor.

La lógica consta de si encuentra que sus vecinos anteriores a él, son igual a cero, significa que hay una nueva secuencia, por lo que el iterador aumenta en 1, en caso contrario, si encuentra un valor distinto de 0 y 5, tomará el nodo dicho valor, si encuentra un valor igual a 5, tomará el valor que tiene el iterador. Esto se hará con cada nodo hasta llegar al último, con la condición de sus fronteras.

V1	V2	V3
V8	Actual	V4
V7	V6	V5

## B. Flujo del programa



## V. CÓDIGO FUENTE

*Función para el procesamiento de los nodos y sus vecinos*

```

1 void actualizaVecino(int matriz[8][20], int i, int
2   j, int label){
3     matriz[i][j] = label;
4   }
5 void vecinos(int matriz[8][20], int i, int j){
6     bool bandera = false;
7
8     if(i == 0 && j == 0){
9       if(matriz[i][j] == 5){
10        matriz[i][j] = iterador;
11      }
12    }else if(j==0 && i==7){
13      int arrSuperiores[2];
14      arrSuperiores[0] = matriz[i-1][j];
15      arrSuperiores[1] = matriz[i-1][j+1];
16      if(arrSuperiores[0] == 0 && arrSuperiores[1]
17 == 0){
18        iterador++;
19        matriz[i][j] = iterador;
20      }else{
21        for(int p=0;p<2;p++){
22          if(arrSuperiores[p] != 5){
23            matriz[i][j] = arrSuperiores[p];
24            bandera = true;
25          }else if(arrSuperiores[p] == 0){
26            continue;
27          }else{
28            if(bandera != true){
29              matriz[i][j] = iterador;

```

```

29       }else{
30         continue;
31       }
32     }
33   }
34 }
35
36 }else if(j==0 && (i>0 && i<8)){
37   int arrSuperiores[3];
38   arrSuperiores[0] = matriz[i-1][j];
39   arrSuperiores[1] = matriz[i-1][j+1];
40   arrSuperiores[2] = matriz[i][j+1];
41
42   if(arrSuperiores[0] == 0 && arrSuperiores[1] ==
43 0 && arrSuperiores[2] == 0){
44     iterador++;
45     matriz[i][j] = iterador;
46   }else{
47     for(int k = 0; k<3; k++){
48       if(arrSuperiores[k] == 0){
49         continue;
50       }else if (arrSuperiores[k] != 5){
51         matriz[i][j] = arrSuperiores[k];
52       }else if(arrSuperiores[k] == 5){
53         matriz[i][j] = iterador;
54       }
55     }
56   }else if(i==7 && j==19){
57     int arrVecinos[2];
58     int arrSuperiores[2];
59     int aux1 = i-1;
60     int aux2 = j-1;
61     int aux3 = j;
62     int aux4 = i;
63
64     arrVecinos[0] = matriz[i-1][j-1];
65     arrVecinos[1] = matriz[i][j-1];
66     arrSuperiores[0] = matriz[i-1][j-1];
67     arrSuperiores[1] = matriz[i-1][j];
68
69
70     if(arrVecinos[0] == 0 && arrVecinos[1]){
71       iterador++;
72       matriz[i][j] = iterador;
73       for(int k=0; k<2;k++){
74         if(arrSuperiores[k] != 0){
75           if(k == 0){
76             actualizaVecino(matriz,aux1,aux2,
77 iterador);
78           }else if(k == 1){
79             actualizaVecino(matriz,aux1,aux3,
80 iterador);
81           }
82         }else{
83           for(int k=0;k<2;k++){
84             if(arrVecinos[k] == 0){
85               continue;
86             }else if(arrVecinos[k] != 5){
87               matriz[i][j] = arrVecinos[k];
88               bandera = true;
89             }else if(arrVecinos[k] == 5){
90               if(bandera == true){
91                 continue;
92               }else{
93                 matriz[i][j] = iterador;
94                 if(k == 0){
95                   actualizaVecino(matriz,aux1,aux2,
96 iterador);
97                 }else if(k == 1){
98                   actualizaVecino(matriz,aux4,aux2,
99 iterador);
100                }

```

```

99     }
100   }
101 }
102
103   for(int k = 0;k<2;k++){
104     if(arrSuperiores[k] != iterador &&
arrSuperiores[k] !=0){
105       if(k == 0){
106         actualizaVecino(matriz,aux1,aux2,
iterador);
107         matriz[i][j] = arrSuperiores[k];
108       }else if(k ==1){
109         actualizaVecino(matriz,aux1,aux3,
iterador);
110         matriz[i][j] = arrSuperiores[k];
111       }
112     }
113   }
114 }
115
116 }else if(i==7 && (j>0 && j<19)){
117   int arrVecinos[2];
118   int arrSuperiores[3];
119   int aux1 = i-1;
120   int aux2 = j-1;
121   int aux3 = j;
122   int aux4 = j+1;
123   int aux5 = i;
124   arrVecinos[0] = matriz[i-1][j-1];
125   arrVecinos[1] = matriz[i][j-1];
126   arrSuperiores[0] = matriz[i-1][j-1];
127   arrSuperiores[1] = matriz[i-1][j];
128   arrSuperiores[2] = matriz[i-1][j+1];
129
130   if(arrVecinos[0] == 0 && arrVecinos[1] == 0){
131     if(arrSuperiores[0] == 0 && arrSuperiores
[1]==0 && arrSuperiores[2]==0){
132       iterador++;
133       matriz[i][j] = iterador;
134       for(int k = 0;k<3;k++){
135         if(arrSuperiores[k] != 0){
136           if(k ==1){
137             actualizaVecino(matriz,aux1,aux2,
iterador);
138           }else if(k == 2){
139             actualizaVecino(matriz,aux1,aux3,
iterador);
140           }else if(k == 3){
141             actualizaVecino(matriz,aux1,aux4,
iterador);
142           }
143         }
144       }
145     }else{
146       for(int k = 0;k<3;k++){
147         if(arrSuperiores[k] != 0){
148           matriz[i][j] = arrSuperiores[k];
149         }
150       }
151     }
152   }else{
153     for(int k=0;k<2;k++){
154       if(arrVecinos[k] == 0){
155         continue;
156       }else if(arrVecinos[k] != 5){
157         matriz[i][j] = arrVecinos[k];
158         bandera = true;
159       }else if(arrVecinos[k] == 5){
160         if(bandera == true){
161           continue;
162         }else{
163           matriz[i][j] = iterador;
164           if(k == 0){
165             actualizaVecino(matriz,aux1,aux2,
iterador);

```

```

166           }else if(k == 1){
167             actualizaVecino(matriz,aux5,aux2,
iterador);
168           }
169         }
170       }
171     }
172   }
173 }
174
175 }else if(j==19 && (i>0 && i<7)){
176   int arrVecinos[3];
177   int arrSuperiores[2];
178   int aux1 = i-1;
179   int aux2 = j-1;
180   int aux3 = j;
181   int aux4 = i;
182
183   arrVecinos[0] = matriz[i-1][j-1];
184   arrVecinos[1] = matriz[i][j-1];
185   arrVecinos[2] = matriz[i+1][j-1];
186   arrSuperiores[0] = matriz[i-1][j-1];
187   arrSuperiores[1] = matriz[i-1][j];
188
189   if(arrVecinos[0] == 0 && arrVecinos[1] == 0
&& arrVecinos[2]==0){
190     iterador++;
191     matriz[i][j] = iterador;
192     for(int k=0; k<2;k++){
193       if(arrSuperiores[k] != 0){
194         if(k == 0){
195           actualizaVecino(matriz,aux1,aux2,
iterador);
196         }else if(k == 1){
197           actualizaVecino(matriz,aux1,aux3,
iterador);
198         }
199       }
200     }
201   }else{
202     for(int k=0;k<3;k++){
203       if(arrVecinos[k] == 0){
204         continue;
205       }else if(arrVecinos[k] != 5){
206         matriz[i][j] = arrVecinos[k];
207         bandera = true;
208       }else if(arrVecinos[k] == 5){
209         if(bandera == true){
210           continue;
211         }else{
212           matriz[i][j] = iterador;
213           if(k == 0){
214             actualizaVecino(matriz,aux1,aux2,
iterador);
215           }else if(k == 1){
216             actualizaVecino(matriz,aux4,aux2,
iterador);
217           }else if(k == 2){
218             actualizaVecino(matriz,aux3,aux2,
iterador);
219           }
220         }
221       }
222     }
223   }
224
225   for(int k = 0;k<2;k++){
226     if(arrSuperiores[k] != iterador &&
arrSuperiores[k] !=0){
227       if(k == 0){
228         actualizaVecino(matriz,aux1,aux2,
iterador);
229       }matriz[i][j] = arrSuperiores[k];
230     }else if(k ==1){
231       actualizaVecino(matriz,aux1,aux3,

```

```

232     iterador);
233         matriz[i][j] = arrSuperiores[k];
234     }
235 }
236
237 }
238
239 }else if((i>0 && i<7) && (j>0 && j<19)){
240     int arrVecinos[5] = {0,0,0,0,0};
241     int arrSuperiores[5];
242     bandera = false;
243     int aux1 = i-1;
244     int aux2 = j-1;
245     int aux3 = i+1;
246     int aux4 = i;
247     int aux5 = j;
248     int aux6 = j+1;
249     arrVecinos[0] = matriz[i-1][j-1];
250     arrVecinos[1] = matriz[i][j-1];
251     arrVecinos[2] = matriz[i+1][j-1];
252     arrVecinos[3] = matriz[i-1][j+1];
253     arrVecinos[4] = matriz[i-1][j];
254     arrSuperiores[0] = matriz[i-1][j-1];
255     arrSuperiores[1] = matriz[i-1][j];
256     arrSuperiores[2] = matriz[i-1][j+1];
257     arrSuperiores[3] = matriz[i][j+1];
258     arrSuperiores[4] = matriz[i+1][j+1];
259
260     if(arrVecinos[0] == 0 && arrVecinos[1]==0 &&
261        arrVecinos[2]==0){
262         if(arrSuperiores[0] == 0 && arrSuperiores
263            [1]==0 && arrSuperiores[2]==0){
264             iterador++;
265             matriz[i][j] = iterador;
266             for(int k = 1;k<5;k++){
267                 if(arrSuperiores[k] != 0){
268                     if(k ==1){
269                         actualizaVecino(matriz,aux1,aux5,
270 iterador);
271                     }else if(k == 2){
272                         actualizaVecino(matriz,aux1,aux6,
273 iterador);
274                     }else if(k == 3){
275                         actualizaVecino(matriz,aux4,aux6,
276 iterador);
277                     }else if(k == 4){
278                         actualizaVecino(matriz,aux3,aux6,
279 iterador);
280                     }
281                 }
282             }
283         }
284     }
285 }else{
286     for(int k=0;k<3;k++){
287
288
289         if(arrVecinos[k] == 0){
290
291
292             continue;
293         }else if(arrVecinos[k] != 5 && arrVecinos[k
294 ] != 0){
295
296             matriz[i][j] = arrVecinos[k];
297             bandera = true;
298         }else if(arrVecinos[k] == 5){

```

```

299         if(bandera == true){
300             continue;
301         }else{
302             matriz[i][j] = iterador;
303
304             if(k == 0){
305                 if(arrVecinos[k] == 0){
306                     continue;
307                 }else{
308                     actualizaVecino(matriz,aux1,aux2,
309 iterador);
310                 }
311             }else if(k == 1){
312                 actualizaVecino(matriz,aux4,aux2,
313 iterador);
314             }else if(k == 2){
315                 actualizaVecino(matriz,aux3,aux2,
316 iterador);
317             }
318         }
319     }
320 }

```

### Main del programa

```

1 int main(int argc, char* argv[]){
2     FILE *archivo = fopen("mapaCalor.txt", "w");
3     int valor = 0;
4     int tiempo_inicio = 0;
5     int tablero
6         [8][20]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
7
8         {0,0,5,5,0,0,0,0,0,5,0,0,0,5,5,0,0,0,0,0},
9
10        {5,5,0,0,5,0,0,5,5,0,0,5,5,0,0,5,0,0,5,5},
11
12        {0,0,5,0,0,5,5,0,5,0,0,0,0,5,0,0,5,0,0,5},
13
14        {0,0,0,5,5,0,0,0,5,5,0,0,0,0,5,5,0,0,0,0},
15
16        {5,0,0,0,0,5,5,0,0,0,5,5,0,0,0,0,5,5,0,0},
17
18        {0,5,0,0,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0},
19
20        {0,0,5,5,5,0,0,0,0,0,0,0,0,5,5,5,0,0,0,0}};
21
22     tiempo_inicio = clock();
23
24     for(int i=0;i<8;i++){
25         for(int j=0;j<20;j++){
26             valor = tablero[i][j];
27             if(valor!=0){
28                 vecinos(tablero, i, j);
29             }
30         }
31     }
32     iterador = 1;
33     for(int i=0;i<8;i++){
34         for(int j=0;j<20;j++){
35             valor = tablero[i][j];
36             if(valor!=0){
37                 vecinos(tablero, i, j);
38             }
39         }

```

```

34 printf("\n");
35 for(int i=0;i<8;i++){
36     for(int j=0;j<20;j++){
37         printf("%d |", tablero[i][j]);
38     }
39     printf("\n");

```

```

40 }
41
42 if(archivo == NULL){
43     printf("Error en la apertura del archivo\n");
44 }else{
45     char linea[80];
46     for(int i=0;i<8;i++){
47         linea[0] = '\0';
48         for(int j =0; j<20;j++){
49             char buffer[10];
50             sprintf(buffer,"%d ", tablero[i][j]);
51             strcat (linea,buffer);
52         }
53         int len = strlen(linea);
54         linea[len-1] = '\n';
55         fputs(linea,archivo);
56     }
57 }
58
59 fclose(archivo);
60 printf("El tiempo que tardo es: %f\n", (clock()-
    tiempo_inicio)/(double)CLOCKS_PER_SEC);
61
62 }

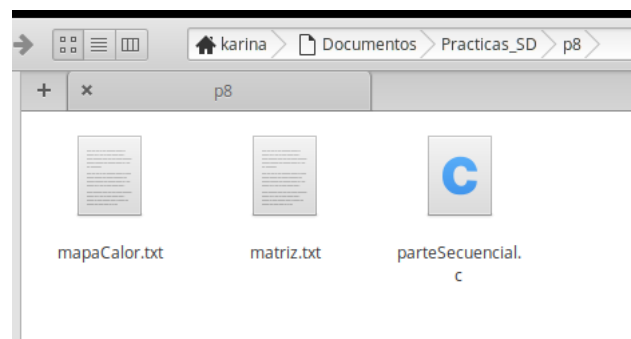
```

```

+ x p8
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p8$ gcc pa
al.c -o secuencial
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p8$ ./secu
0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 |
1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 2 | 0 | 0 | 2 | 0 | 0 |
0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 0 |
0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 |
4 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 2 | 0 |
0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
El tiempo que tardo es: 0.000157
karina@karina-VirtualBox:~/Documentos/Practicas_SD/p8$

```

## Archivos generados



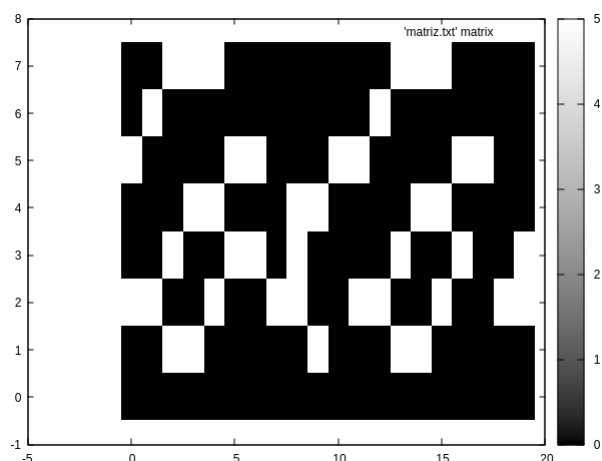
```

+ x mapaCalor.txt
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 0 0 1 1 0 0 0 0 0 1 0 0 0 2 2 0 0 0 0 0
3 1 1 0 0 1 0 0 1 1 0 0 2 2 0 0 2 0 0 3 3
4 0 0 1 0 0 1 1 0 1 0 0 0 0 2 0 0 2 0 0 3
5 0 0 0 1 1 0 0 0 1 1 0 0 0 2 2 0 0 0 0 0
6 4 0 0 0 0 1 1 0 0 0 1 1 0 0 0 2 2 0 0 0
7 0 4 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
8 0 0 4 4 4 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0

```

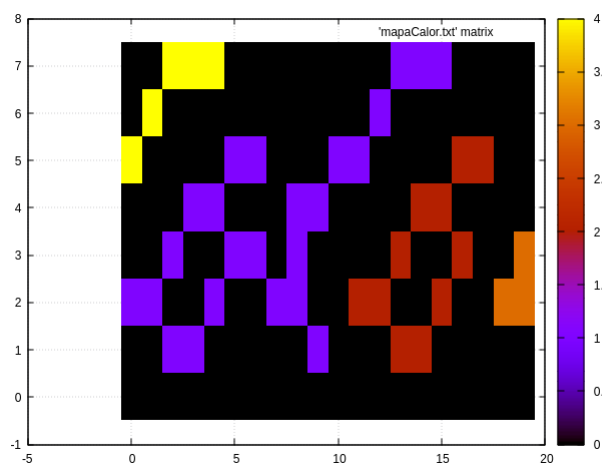
## VI. MAPA DE CALOR

Mapa de la matriz inicial.



**Nota: Gnuplot muestra la matriz al revés.**

Mapa después del procesamiento.



**Nota: Gnuplot muestra la matriz al revés.**

## VII. CONCLUSIONES

Para el procesamiento de la matriz fue de vital importancia tomar en cuenta los diferentes casos que se presentan en cada elemento. Tuvimos que hacer validaciones debido a que las fronteras de la matriz, es decir la primera y última columna, así como la primera y última fila no presentan los ocho vecinos que los demás elementos sí poseen. Esta fue la parte más laboriosa del algoritmo y en la que se invierte más código.

Logramos la implementación de un autómatas celular y pudimos visualizar su comportamiento con un mapa de calor realizado con gnuplot de Linux.

## REFERENCES

- [1] Rechtman, R. Una introducción a autómatas celulares. Revista Ciencias, UNAM. <https://www.revistaciencias.unam.mx/pt/172-revistas/revista-ciencias-24/1572-una-introducci%C3%B3n-a-aut%C3%B3matas-celulares.html>
- [2] Ayala,J.A.(22 abril 2020) Práctica 8:Autómatas Celulares. Sistemas Distribuidos, 1.