# 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.**

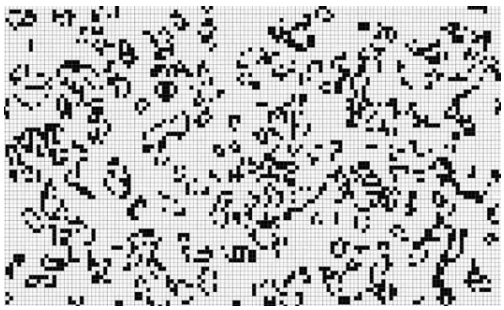
## I. OBJETIVOS.

- Procesar una matriz haciendo uso de autómatas celulares.
- Comprender y realizar la implementación de un atómata 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.[1]
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 funcioné de manera correcta, debemos de pasarlo a un mapa de calor para que los resultados se visualicen con mayor facilidad.

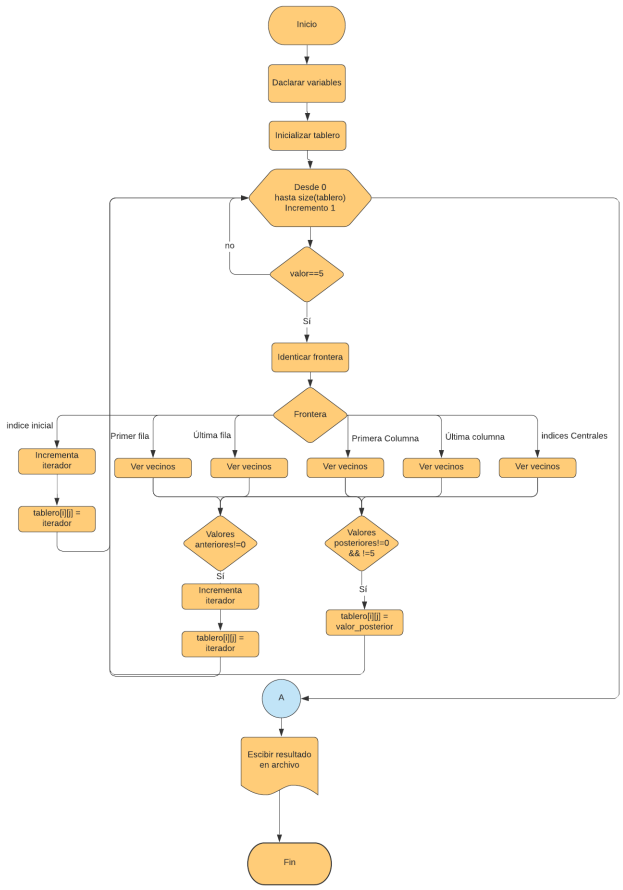
## IV. IMPLEMENTACIÓN

### 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, signica 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*

```
void actualizaVecino(int matriz[8][20], int i, int j,int label){
  matriz[i][j] = label;
}

void vecinos(int matriz[8][20],int i, int j){
  bool bandera = false;

  if(i == 0 && j == 0){
      if(matriz[i][j] ==5){
        matriz[i][j] = iterador;
      }
  }else if(j==0 && i==7){
      int arrSuperiores[2];
      arrSuperiores[0] = matriz[i-1][j];
      arrSuperiores[1] = matriz[i-1][j+1];
      if(arrSuperiores[0] == 0 && arrSuperiores[1] == 0){
          iterador++;
          matriz[i][j] = iterador;
      }else{
          for(int p=0;p<2;p++){
            if(arrSuperiores[p] != 5){
                matriz[i][j] = arrSuperiores[p];
                bandera = true;
            }else if(arrSuperiores[p] == 0){
                continue;
            }else{
                if(bandera != true){
                  matriz[i][j] = iterador;
                }else{
                  continue;
                }
            }
          }
      }
  }

}else if(j==0 && (i>0 && i<8)){
int arrSuperiores[3];
arrSuperiores[0] = matriz[i-1][j];
arrSuperiores[1] = matriz[i-1][j+1];
arrSuperiores[2] = matriz[i][j+1];

if(arrSuperiores[0] == 0 && arrSuperiores[1] == 0 && arrSuperiores[2] == 0){
  iterador++;
  matriz[i][j] = iterador;
}else{
  for(int k = 0; k<3; k++){
    if(arrSuperiores[k] == 0){
      continue;
    }else if (arrSuperiores[k] != 5){
      matriz[i][j] = arrSuperiores[k];
    }else if(arrSuperiores[k] == 5){
      matriz[i][j] = iterador;
    }
  }
}
}else if(i==7 && j==19){
  int arrVecinos[2];
  int arrSuperiores[2];
  int aux1 = i-1;
  int aux2 = j-1;
  int aux3 = j;
  int aux4 = i;

  arrVecinos[0] = matriz[i-1][j-1];
  arrVecinos[1] = matriz[i][j-1];
  arrSuperiores[0] = matriz[i-1][j-1];
  arrSuperiores[1] = matriz[i-1][j];


  if(arrVecinos[0] == 0 && arrVecinos[1]){
    iterador++;
    matriz[i][j] = iterador;
    for(int k=0; k<2;k++){
      if(arrSuperiores[k] != 0){
        if(k == 0){
          actualizaVecino(matriz,aux1,aux2,iterador);
        }else if(k == 1){
          actualizaVecino(matriz,aux1,aux3,iterador);
        }
      }
    }
  }else{
  for(int k=0;k<2;k++){
    if(arrVecinos[k] == 0){
      continue;
    }else if(arrVecinos[k] != 5){
      matriz[i][j] = arrVecinos[k];
      bandera = true;
    }else if(arrVecinos[k] == 5){
      if(bandera == true){
        continue;
      }else{
        matriz[i][j] = iterador;
        if(k == 0){
          actualizaVecino(matriz,aux1,aux2,iterador);
        }else if(k == 1){
          actualizaVecino(matriz,aux4,aux2,iterador);
        }
```

```
        }
      }
    }

    for(int k = 0;k<2;k++){
      if(arrSuperiores[k] != iterador &&
arrSuperiores[k] !=0){
        if(k == 0){
          actualizaVecino(matriz,aux1,aux2,
iterador);
          matriz[i][j] = arrSuperiores[k];
        }else if(k ==1){
          actualizaVecino(matriz,aux1,aux3,
iterador);
          matriz[i][j] = arrSuperiores[k];
        }
      }
    }
    }

  }else if(i==7 && (j>0 && j<19)){
    int arrVecinos[2];
    int arrSuperiores[3];
    int aux1 = i-1;
    int aux2 = j-1;
    int aux3 = j;
    int aux4 = j+1;
    int aux5 = i;
    arrVecinos[0] = matriz[i-1][j-1];
    arrVecinos[1] = matriz[i][j-1];
    arrSuperiores[0] = matriz[i-1][j-1];
    arrSuperiores[1] = matriz[i-1][j];
    arrSuperiores[2] = matriz[i-1][j+1];

    if(arrVecinos[0] == 0 && arrVecinos[1] == 0){
    if(arrSuperiores[0] == 0 && arrSuperiores
[1]==0 && arrSuperiores[2]==0){
      iterador++;
      matriz[i][j] = iterador;
      for(int k = 0;k<3;k++){
        if(arrSuperiores[k] != 0){
          if(k ==1){
            actualizaVecino(matriz,aux1,aux2,
iterador);
          }else if(k == 2){
            actualizaVecino(matriz,aux1,aux3,
iterador);
          }else if(k == 3){
            actualizaVecino(matriz,aux1,aux4,
iterador);
          }
        }
      }
    }else{
      for(int k = 0;k<3;k++){
        if(arrSuperiores[k] != 0){
          matriz[i][j] = arrSuperiores[k];
        }
      }
    }
    }else{
    for(int k=0;k<2;k++){
      if(arrVecinos[k] == 0){
        continue;
      }else if(arrVecinos[k] != 5){
        matriz[i][j] = arrVecinos[k];
        bandera = true;
      }else if(arrVecinos[k] == 5){
        if(bandera == true){
          continue;
        }else{
          matriz[i][j] = iterador;
          if(k == 0){
            actualizaVecino(matriz,aux1,aux2,
iterador);
          }else if(k == 1){
            actualizaVecino(matriz,aux5,aux2,
iterador);
          }
        }
      }
    }

    }

  }else if(j==19 && (i>0 && i<7)){
    int arrVecinos[3];
    int arrSuperiores[2];
    int aux1 = i-1;
    int aux2 = j-1;
    int aux3 = j;
    int aux4 = i;

  arrVecinos[0] = matriz[i-1][j-1];
    arrVecinos[1] = matriz[i][j-1];
    arrVecinos[2] = matriz[i+1][j-1];
    arrSuperiores[0] = matriz[i-1][j-1];
    arrSuperiores[1] = matriz[i-1][j];

    if(arrVecinos[0] == 0 && arrVecinos[1] == 0
&& arrVecinos[2]==0){
      iterador++;
      matriz[i][j] = iterador;
      for(int k=0; k<2;k++){
        if(arrSuperiores[k] != 0){
          if(k == 0){
            actualizaVecino(matriz,aux1,aux2,
iterador);
          }else if(k == 1){
            actualizaVecino(matriz,aux1,aux3,
iterador);
          }
        }
      }
    }else{
    for(int k=0;k<3;k++){
      if(arrVecinos[k] == 0){
        continue;
      }else if(arrVecinos[k] != 5){
        matriz[i][j] = arrVecinos[k];
        bandera = true;
      }else if(arrVecinos[k] == 5){
        if(bandera == true){
          continue;
        }else{
          matriz[i][j] = iterador;
          if(k == 0){
            actualizaVecino(matriz,aux1,aux2,
iterador);
          }else if(k == 1){
            actualizaVecino(matriz,aux4,aux2,
iterador);
          }else if(k == 2){
            actualizaVecino(matriz,aux3,aux2,
iterador);
          }

        }
      }
    }
  }

    for(int k = 0;k<2;k++){
      if(arrSuperiores[k] != iterador &&
arrSuperiores[k] !=0){
        if(k == 0){
          actualizaVecino(matriz,aux1,aux2,
iterador);
          matriz[i][j] = arrSuperiores[k];
        }else if(k ==1){
          actualizaVecino(matriz,aux1,aux3,
```

```
iterador);
        matriz[i][j] = arrSuperiores[k];
      }
    }
  }

}

}else if((i>0 && i<7) && (j>0 && j<19)){
int arrVecinos[5] = {0,0,0,0,0};
int arrSuperiores[5];
bandera = false;
int aux1 = i-1;
int aux2 = j-1;
int aux3 = i+1;
int aux4 = i;
int aux5 = j;
int aux6 = j+1;
arrVecinos[0] = matriz[i-1][j-1];
  arrVecinos[1] = matriz[i][j-1];
  arrVecinos[2] = matriz[i+1][j-1];
arrVecinos[3] = matriz[i-1][j+1];
arrVecinos[4] = matriz[i-1][j];
arrSuperiores[0] = matriz[i-1][j-1];
arrSuperiores[1] = matriz[i-1][j];
arrSuperiores[2] = matriz[i-1][j+1];
arrSuperiores[3] = matriz[i][j+1];
arrSuperiores[4] = matriz[i+1][j+1];

if(arrVecinos[0] == 0 && arrVecinos[1]==0 &&
arrVecinos[2]==0){
  if(arrSuperiores[0] == 0 && arrSuperiores
[1]==0 && arrSuperiores[2]==0){

    iterador++;
    matriz[i][j] = iterador;
    for(int k = 1;k<5;k++){
      if(arrSuperiores[k] != 0){
        if(k ==1){
          actualizaVecino(matriz,aux1,aux5,
iterador);
        }else if(k == 2){
          actualizaVecino(matriz,aux1,aux6,
iterador);
        }else if(k == 3){
          actualizaVecino(matriz,aux4,aux6,
iterador);
        }else if(k == 4){
          actualizaVecino(matriz,aux3,aux6,
iterador);
        }
      }
    }
  }else{
    for(int k = 0;k<4;k++){
      if(arrSuperiores[k] != 0){
        matriz[i][j] = arrSuperiores[k];
      }
    }
  }

}else{
  for(int k=0;k<3;k++){


    if(arrVecinos[k] == 0){

      continue;
    }else if(arrVecinos[k] != 5 && arrVecinos[k
] != 0){

      matriz[i][j] = arrVecinos[k];
      bandera = true;
    }else if(arrVecinos[k] == 5){

      if(bandera == true){
        continue;
      }else{
        matriz[i][j] = iterador;

        if(k == 0){
          if(arrVecinos[k] == 0){
            continue;
          }else{
            actualizaVecino(matriz,aux1,aux2,
iterador);
          }
        }else if(k == 1){
          actualizaVecino(matriz,aux4,aux2,
iterador);
        }else if(k == 2){
          actualizaVecino(matriz,aux3,aux2,
iterador);
        }

      }
    }
  }
}
```

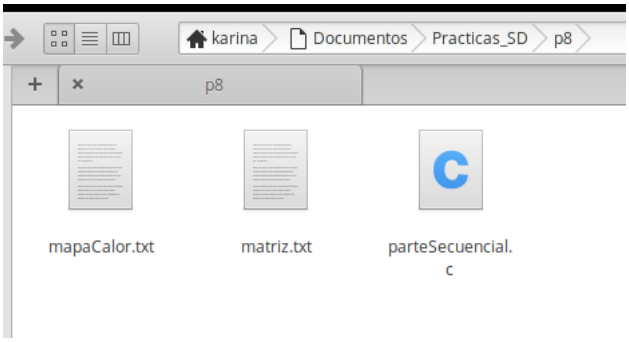
### *Main del programa*

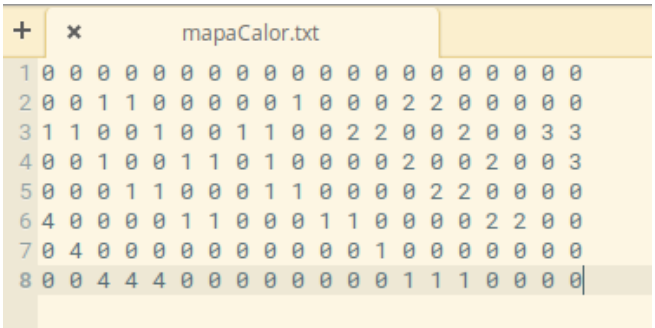
```
int main(int argc, char* argv[]){
  FILE *archivo = fopen("mapaCalor.txt","w");
  int valor = 0;
  int tiempo_inicio = 0;
  int tablero
    [8][20]={{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

    {0,0,5,5,0,0,0,0,0,5,0,0,0,5,5,0,0,0,0,0},

    {5,5,0,0,5,0,0,5,5,0,0,5,5,0,0,5,0,0,5,5},

    {0,0,5,0,0,5,5,0,5,0,0,0,0,5,0,0,5,0,0,5},

    {0,0,0,5,5,0,0,0,5,5,0,0,0,0,5,5,0,0,0,0},

    {5,0,0,0,0,5,5,0,0,0,5,5,0,0,0,0,5,5,0,0},

    {0,5,0,0,0,0,0,0,0,0,0,0,5,0,0,0,0,0,0,0},

    {0,0,5,5,5,0,0,0,0,0,0,0,0,5,5,5,0,0,0,0}};

  tiempo_inicio = clock();

  for(int i=0;i<8;i++){
    for(int j=0;j<20;j++){
      valor = tablero[i][j];
        if(valor!=0){
          vecinos(tablero, i, j);
        }
    }
  }
  iterador = 1;
  for(int i=0;i<8;i++){
    for(int j=0;j<20;j++){
      valor = tablero[i][j];
        if(valor!=0){
          vecinos(tablero, i, j);
        }
    }
  }

  printf("\n");
  for(int i=0;i<8;i++){
      for(int j=0;j<20;j++){
        printf("%d |", tablero[i][j]);
    }
    printf("\n");
```

```
  }

  if(archivo == NULL){
    printf("Error en la apertura del archivo\n");
  }else{
    char linea[80];
    for(int i=0;i<8;i++){
      linea[0] = '\0';
      for(int j =0; j<20;j++){
        char buffer[10];
        sprintf(buffer,"%d ", tablero[i][j]);
        strcat(linea,buffer);
      }
      int len = strlen(linea);
      linea[len-1] = '\n';
      fputs(linea,archivo);
    }
  }

  fclose(archivo);
  printf("El tiempo que tardo es: %f\n", (clock()-
    tiempo_inicio)/(double)CLOCKS_PER_SEC);

}
```

```
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 |
0 |0 |1 |1 |0 |0 |0 |0 |0 |1 |0 |0 |0 |2 |2 |0 |0 |0 |0 |0 |
1 |1 |0 |0 |1 |0 |0 |1 |1 |0 |0 |2 |2 |0 |0 |2 |0 |0 |3 |3 |
0 |0 |1 |0 |0 |1 |1 |0 |1 |0 |0 |0 |0 |2 |0 |0 |2 |0 |0 |3 |
0 |0 |0 |1 |1 |0 |0 |0 |1 |1 |0 |0 |0 |0 |2 |2 |0 |0 |0 |0 |
4 |0 |0 |0 |0 |1 |1 |0 |0 |0 |1 |1 |0 |0 |0 |0 |2 |2 |0 |0 |
0 |4 |0 |0 |0 |0 |0 |0 |0 |0 |0 |0 |0 |1 |0 |0 |0 |0 |0 |0 |
0 |0 |4 |4 |4 |0 |0 |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$
```

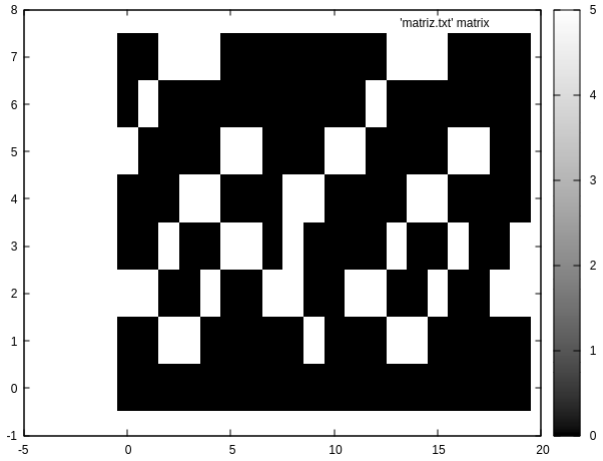
*Archivos generados*



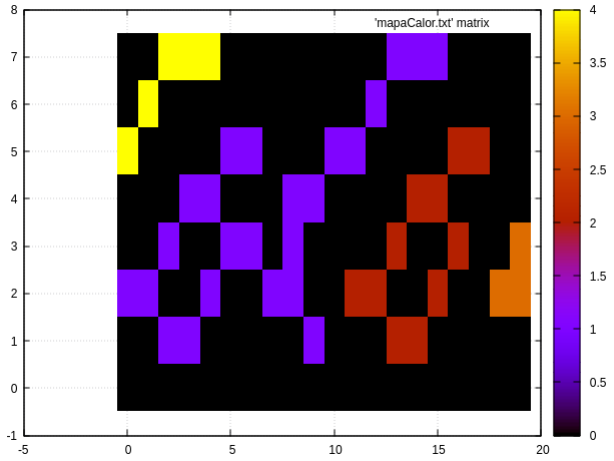

## VI. Mapa de calor

*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 atómata 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.