Introducción a las Menciones II: Computación Científica - Automata Celular

Graeme Candlish

graeme.candlish@ifa.uv.cl

Contenido

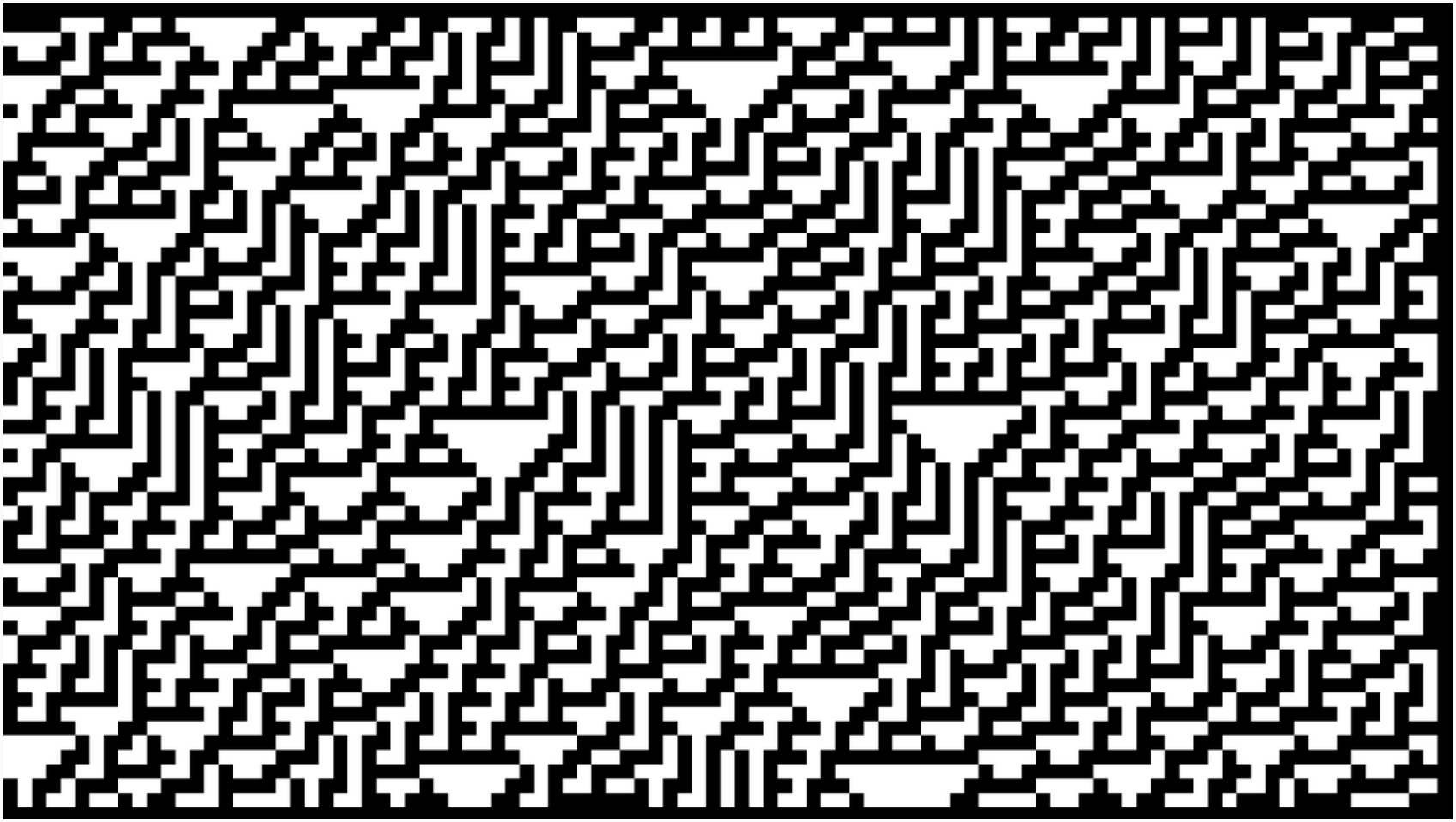
Contenido

Automata celular

- Un **automata celular** es un sistema compuesto de "células" con reglas que definen su comportamiento.
- Típicamente estas reglas son muy simples, pero la combinación de muchas células resulta en un comportamiento global que es complejo.

Automata celular

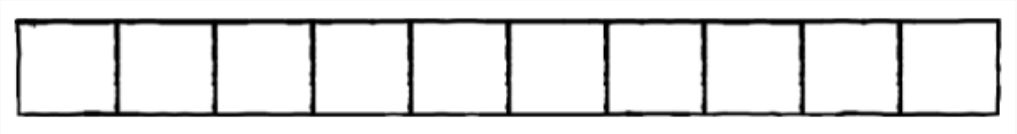
Automata celular



Contenido

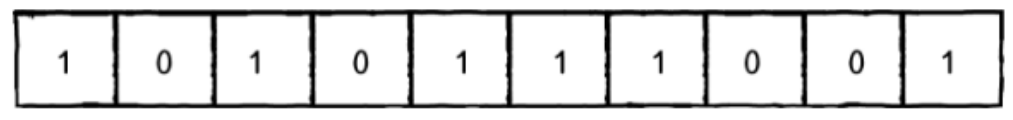
Creando la simulación

Representamos las células como un arreglo de cajitas:



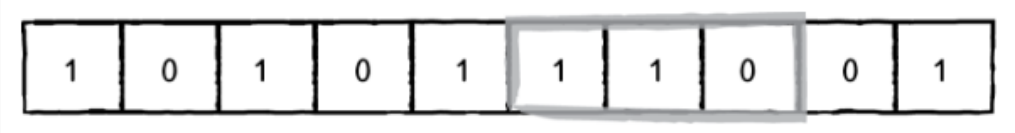
Creando la simulación

Cada célula tiene un *estado* de **vivo** o **muerto**, representado con 1 o 0:



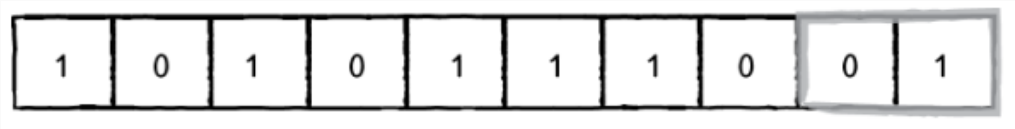
Creando la simulación

Finalmente, necesitamos el concepto de una *vecindad* en el arreglo. La opción más simple es considerar las dos células al lado como las células vecinas:



Creando la simulación

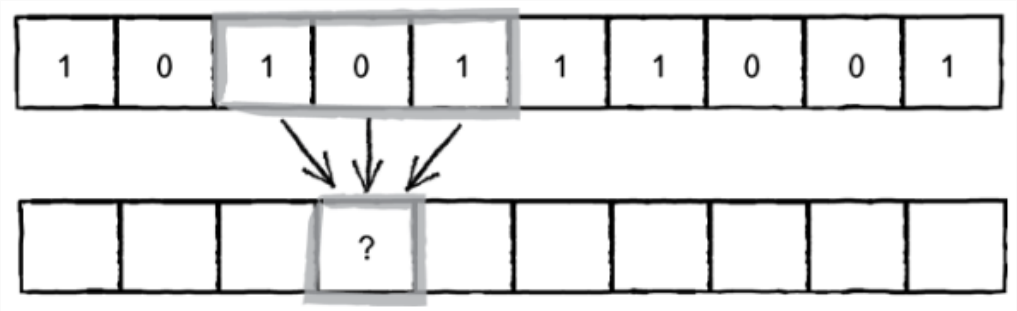
¿Qué hacemos con las células en el borde? Por ahora vamos a ignorar estas células (es decir, tendrán un estado constante).



Contenido

Las reglas

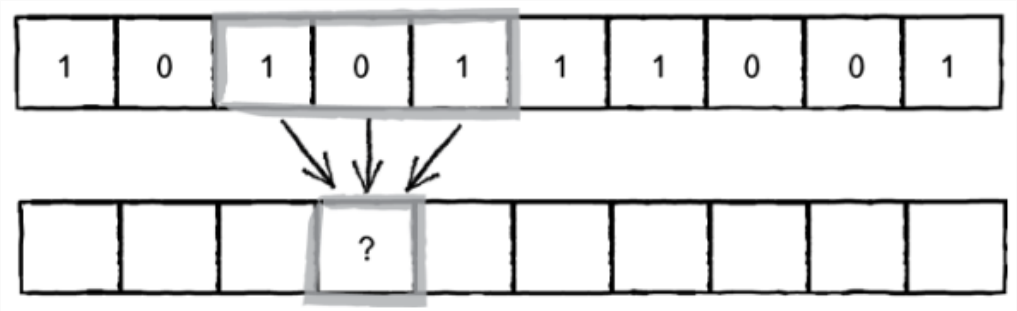
- A partir de un estado inicial de cada célula, queremos ver que pasa con las generaciones subsiguientes, según reglas muy simples.
- Para cada generación de células, decidimos si una célula sigue viviendo según los estados de sus células vecinas.



Las reglas

Pregunta

¿Cuántas combinaciones de estados hay para cada conjunto de 3 células?



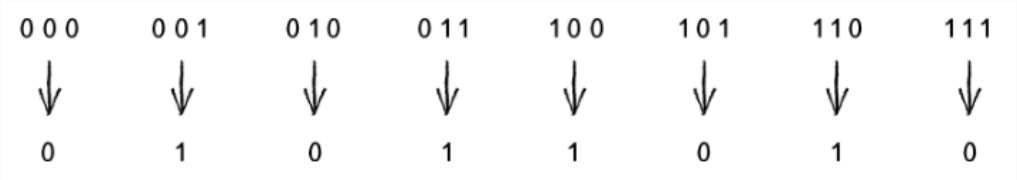
Las reglas

Hay 8 combinaciones posibles: 000, 001, 010, 011, 100, 101, 110 y 111.

- Necesitamos definir, para **cada uno** de estos conjuntos, el valor de la célula en el medio en la generación subsiguiente.

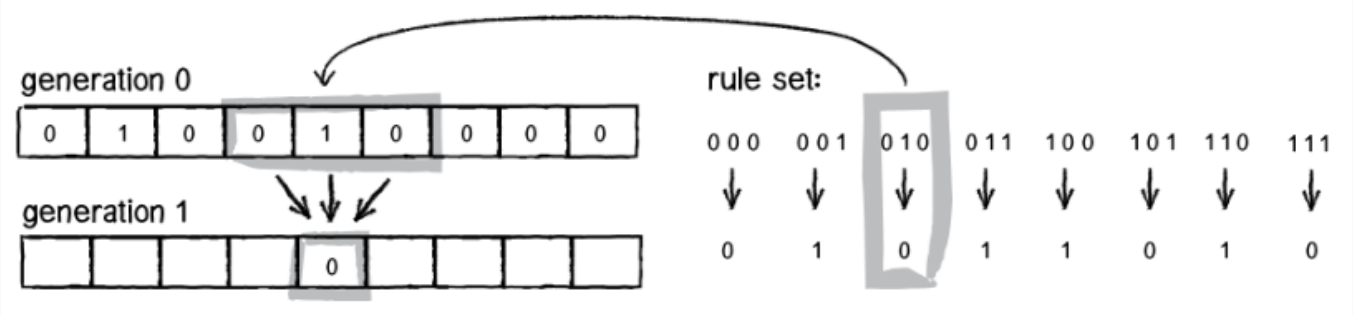
Las reglas: un ejemplo

El término *regla* aplicará al *conjunto* de los 8 “mapeos” mostrados en la figura:



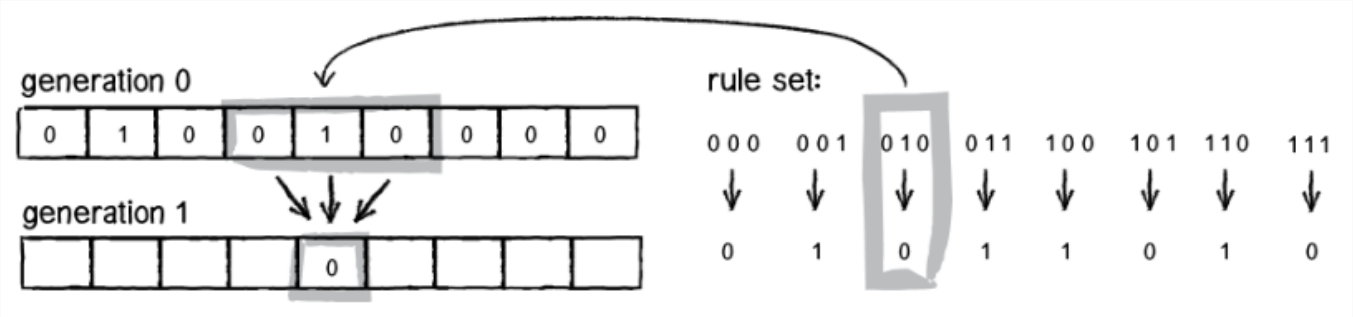
Aplicando las reglas

En la figura mostramos la aplicación de la regla que vimos antes a una parte de un arreglo inicial con 2 células vivas, y todas las otras muertas.



Tarea

Determinar los valores nuevos de todas las otras células en el ejemplo mostrado en la figura. Las células en el borde (i.e. la primera y la última) se mantienen constante en todas las generaciones.



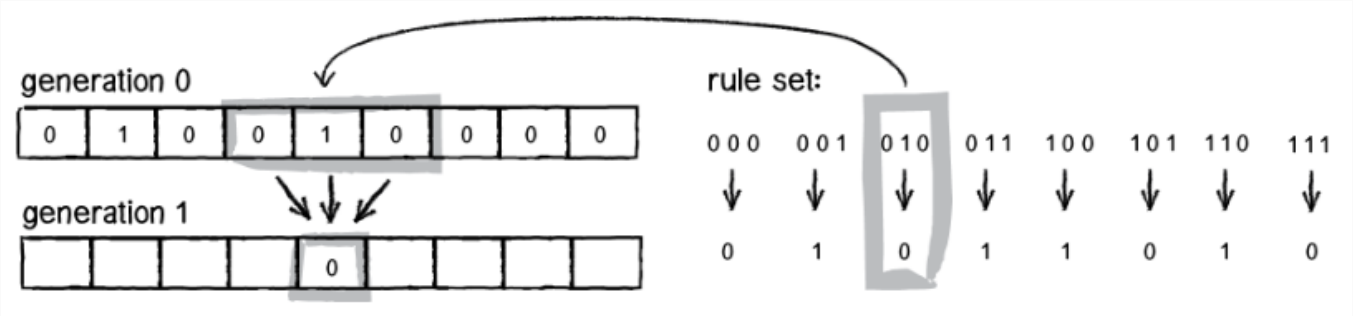
Contenido

Utilizando números binarios

- Es inconveniente tener que especificar una lista de 8 números para definir cada regla.
- Ocupamos los números binarios para expresar cada regla en una forma compacta.

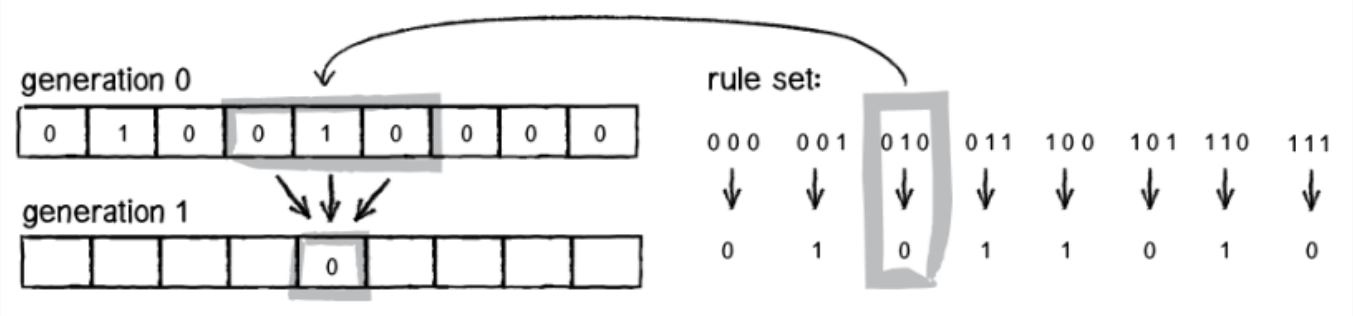
Utilizando números binarios

Cada uno de los conjuntos de estados de las células forma un número binario: tenemos 0, 1, 2, 3, 4, 5, 6 y 7 en sus representaciones binarias. De los 8 dígitos abajo podemos formar un número en binario entre 0 y 255. El nmero en el ejemplo es 90 en decimal.



Utilizando números binarios

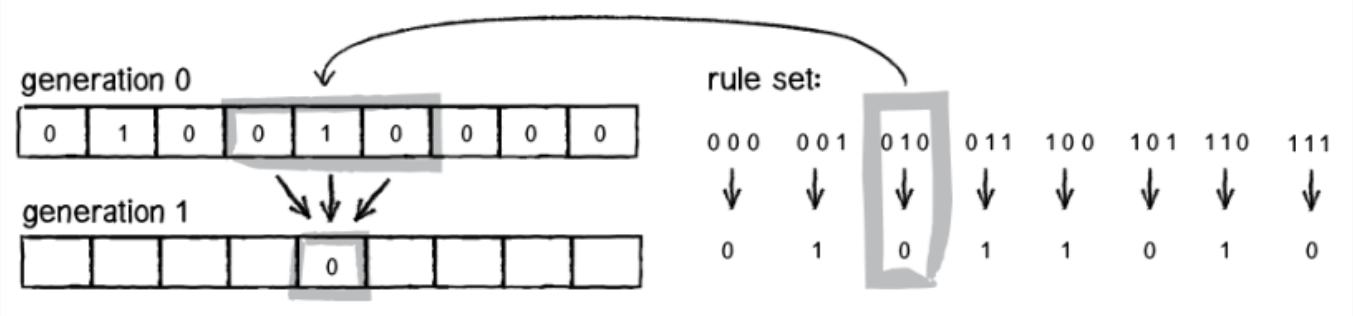
Por lo tanto, podemos representar cada regla (es decir, los 8 mapeos) con un número binario de 8 bits.



¿Qué rango de números enteros podemos representar con 8 bits?

Utilizando números binarios

Por lo tanto, podemos representar cada regla (es decir, los 8 mapeos) con un número binario de 8 bits.



¿Qué rango de números enteros podemos representar con 8 bits?

0 a 255

Tarea

Escribir las representaciones en binario de los siguientes números: 0, 10, 11, 123 y 255. Utilice 8 dígitos para cada número (8 bits).

Solución

- 0: 00000000
- 10: 00001010
- 11: 00001011
- 123: 01111011
- 255: 11111111

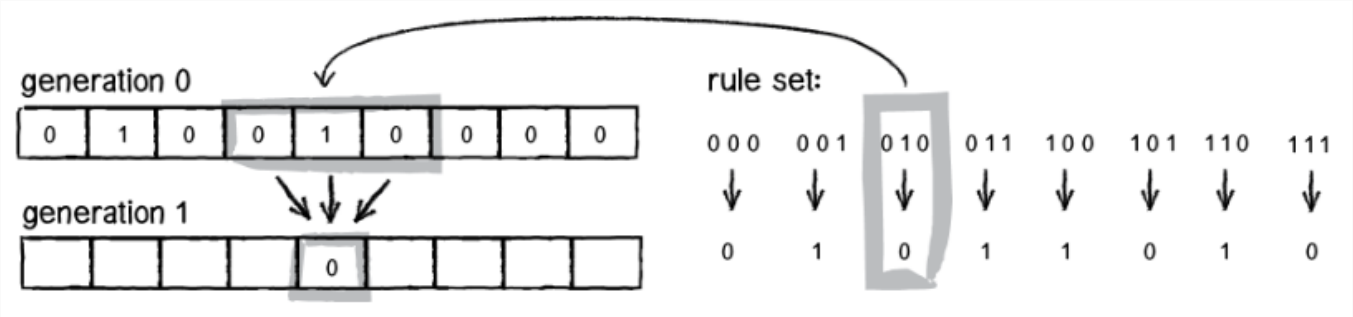
Tarea

Considerar un automata celular donde sólo ocupamos el estado de *una* de las células vecinas, y no las dos.

- ¿Cuántas "mapeos" necesitamos para definir una regla de evolución en este caso?
- Determinar cuantas reglas posibles hay en total en este caso, y por lo tanto cuántos *bits* necesitamos para representar estas reglas con números binarios.

Tarea

Escribir el algoritmo que corresponde a la simulación del automata celular.



Implementando la simulación en Python

Funciones útiles:

- `bin`: convierte un número en decimal a un número en binario.
- `zfill`: agrega 0s a la izquierda de un *string*.
- `matshow`: hace un gráfico de una matriz de NumPy.

Creando las células

Podemos crear un arreglo de N células en NumPy con:

```
import numpy as np
c = np.zeros(N,dtype=int)
```

Tenemos que elegir un valor para N (o obtener el valor del usuario).

Definiendo la regla

Usamos `bin` para convertir un número en decimal a su representación en binario:

```
regla_decimal = 90
regla = bin(regla_decimal)
```

Pero hay un problema...

Definiendo la regla

Usamos `bin` para convertir un número en decimal a su representación en binario:

```
regla_decimal = 90
regla = bin(regla_decimal)[2:]
```

Tarea de Python

Determinar como elegir los 3 elementos del arreglo c, si el elemento en el medio tiene índice i.

Tarea de Python - solución

```
grupo = c[i-1:i+2]
```

Aplicando la regla

Ahora queremos definir una función que aplicará las reglas a cada conjunto de 3 células, y podemos pasar el conjunto "c[i-1:i+2]" a esta función.

```
def aplicar_regla(celulas):
    c1 = celulas[0]
    c2 = celulas[1]
    c3 = celulas[2]

    ...
```

Usando la representación binaria

Los estados de estos 3 células están representados por 3 dígitos, donde cada dígito es 0 (muerta) o 1 (viva). Así que podemos representar la combincación de las 3 células con un número binario entre 0 y 7.

Tarea

Calcule el valor en decimal del número binario que tiene dígitos c_1, c_2 y c_3 (es decir, el número binario $c_1c_2c_3$).

Tarea - solución

```
b = c1*4 + c2*2 + c3*1
```

Tarea

Tenemos el *string* "regla" que es la representación en binario de la regla que queremos aplicar. El valor b de la última tarea es el valor en decimal del conjunto de células. Determinar el nuevo valor de la célula en el medio del grupo de 3 células.

Tarea - solución

```
valor_nuevo = regla[::-1][b]
```

La función “aplicar_regla”

```
def aplicar_regla(celulas):
    c1 = celulas[0]
    c2 = celulas[1]
    c3 = celulas[2]

    b = c1*4 + c2*2 + c3*1
    valor_nuevo = regla[::-1][b]

    return(valor_nuevo)
```

El algoritmo

- Definimos el número de células N.
- Definimos el conjunto de reglas con un número R entre 0 y 255.
- Convertimos R en una cadena de 8 dígitos que corresponde a su representación en binario.
- Creamos dos arreglos de N elementos: el arreglo "celulas_antes" que corresponde a los valores antes de aplicar la regla, y "celulas_despues" que corresponde a los valores después de aplicar las reglas.

El algoritmo

- Asignamos el valor de 1 al elemento en el medio (con índice $N/2$) y todos los otros son 0.
- Usamos un ciclo que elige grupos de 3 células (elementos del arreglo "celulas_antes") comenzando del lado izquierdo.
- Pasamos cada grupo a la función "aplicar_regla" y calculamos el valor nuevo de la célula en el medio.
- Asignamos este valor a la célula en el arreglo "celulas_despues".

Pregunta: ¿Por qué tenemos dos arreglos de células?

Tarea

Escribir un programa que incluya todos estos pasos del algorítmo.

Tarea - solución

```
import numpy as np

def aplicar_regla(celulas):
    c1 = celulas[0]
    c2 = celulas[1]
    c3 = celulas[2]

    b = c1*4 + c2*2 + c3*1
    valor_nuevo = regla[::-1][b]

    return(valor_nuevo)
```

Tarea - solución

```
N = 11 #Por ejemplo
R = 30
regla = bin(R)[2:].zfill(8)

celulas_antes = np.zeros(N,dtype=int)
celulas_despues = np.zeros(N,dtype=int)

celulas_antes[int(N/2)] = 1

for i in range(1,N-1):
    grupo = celulas_antes[i-1:i+2]
    celulas_despues[i] = aplicar_regla(grupo)
```

Evolución en el tiempo

Hasta ahora nuestro programa solamente puede calcular el cambio en las células entre una generación a otra. Para determinar su evolución en el tiempo, necesitamos un ciclo sobre las generaciónes.

Tarea

Modificar el programa para incluir un ciclo sobre “tiempo” pasos del tiempo, donde “tiempo” es un número entero definido al principio del programa.

Tarea - solución

```
...
tiempo = 100
...
for t in range(tiempo-1):
    for i in range(1,N-1):
        grupo = celulas_antes[i-1:i+2]
        celulas_despues[i] = aplicar_regla(grupo)
    celulas_antes = celulas_despues
```

Tarea

Modificar el programa para reemplazar “celulas_antes” y “celulas_despues” con un sólo arreglo de dos dimensiones. Hay que modificar el resto del programa para que funcione bien con este nuevo arreglo.

Tarea - solución

```
import numpy as np

def aplicar_regla(celulas):
    c1 = celulas[0]
    c2 = celulas[1]
    c3 = celulas[2]

    b = c1*4 + c2*2 + c3*1
    valor_nuevo = regla[::-1][b]

    return(valor_nuevo)
```

Tarea - solución

```
tiempo = 100
N = 11 #Por ejemplo
R = 30
regla = bin(R)[2:].zfill(8)

celulas = np.zeros((tiempo,N),dtype=int)

celulas[0,int(N/2)] = 1

for t in range(tiempo-1):
    for i in range(1,N-1):
        grupo = celulas[t,i-1:i+2]
        celulas[t+1,i] = aplicar_regla(grupo)
```

Mostrando el resultado

Podemos usar `matshow` de Matplotlib para mostrar las células como un gráfico. Hay que importar `matplotlib.pyplot` y al final del programa ponemos

```
matshow(celulas,cmap="Greys")
```

La simulación

Ahora tenemos una simulación de este automata celular! Podemos investigar los patrones formados por diferentes elecciones de la regla. Hay una página web sobre este sistema: http://mathworld.wolfram.com/ElementaryCellularAutomaton.html