Organización de Computadoras

Trabajo práctico 0: Infraestructura básica

Nombre	Padron	Email
Pareja, Facundo Jose	99719	fpareja@fi.uba.ar

2024

Implementación del programa

En esta sección se detallan dos aspectos de la implementación considerados de interés.

Determinación del siguiente estado de una celda

La matriz de estados fue implementada como un vector de vectores de caracteres (tipo char). Luego de leerse y cargarse el estado inicial en la primera fila de la misma, se llama iterativamente a la función next_state (se cambió el nombre para mantener consistencia) para cada celda a actualizar hasta finalizar la cantidad de iteraciones total.

El funcionamiento de next_state es el siguiente: En primer lugar, para la celda a actualizar, se obtienen sus celdas vecinas, teniendo cuidado con las celdas de los extremos.

```
if (j == 0) {
        left_neighbor = state_matrix[i][N-1];
    }
    else {
        left_neighbor = state_matrix[i][j-1];
    }
    if (j == N-1) {
        right_neighbor=state_matrix[i][0];
    }
    else {
        right_neighbor = state_matrix[i][j+1];
    }
```

Luego, para cada grupo de 3 celdas con el siguiente esquema, donde la celda principal es aquella cuyo valor está siendo actualizado, se calcula que número de bit representan en la regla representada en binario, y se devuelve en base al número correspondiente de la regla convertida a binario.

```
    int bit_number = 4 * left_neighbor + 2 * cell + right_neighbor;
    return (regla & ( 1 << bit_number )) >> bit_number;
```

Por ejemplo, supongamos que que se quiere actualizar la celda en 0 con vecinos 1 a ambos lados

1	0	1

Este caso representa el número 5, y se obtiene el valor correspondiente de la representación en binario de la regla utilizada. En este caso, asumiendo regla 30:

111	110	101	100	011	010	001	000
0	0	0	1	1	1	1	0

Por lo tanto, el valor de la celda no cambia.

Impresion de la matriz

Si se imprime cada elemento de la matriz directamente se obtiene un pixel por valor, lo cual es demasiado chico para apreciarlo fácilmente. Por lo tanto, se decidió cuadruplicar cada pixel, imprimiendo cada valor dos veces horizontalmente y luego repitiendo cada línea verticalmente.

```
for (int i = 0; i < cell_number; i++) {
    write_row(state_matrix, file_pointer, cell_number, i);
    write_row(state_matrix, file_pointer, cell_number, i);
    fprintf(file_pointer, "\n");
}
```

Y dentro de `write_row`:

```
for (int j = 0; j < cell_number; j++) {
    fprintf(file_pointer, "%d ", state_matrix[i][j]);
    fprintf(file_pointer, "%d", state_matrix[i][j]);
    if (j != cell_number-1) {
        fprintf(file_pointer, " ");
    }
}
```

Esto convierte la matriz de 2x2 que se observa a continuación

1	0
0	1

En la siguiente matriz de 4x4:

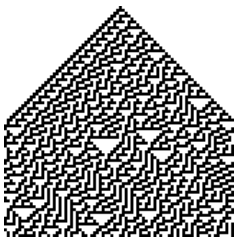
1	1	0	0
1	1	0	0
0	0	1	1
0	0	1	1

Esto facilita la visualización de la imagen resultado.

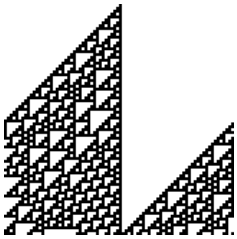
Corridas de prueba

Se muestran a continuación las corridas de prueba pedidas para una matriz de lado 80 con reglas 30, 110 y 126, con una celda ocupada en el centro como estado inicial.

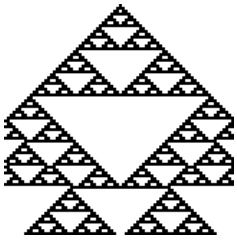
Regla 30



Regla 110



Regla 126



Para replicarlas a mano, ejecutar

```
./autcel R 80 inicial [-o F]
```

Donde R es el número de regla y F (opcional) el nombre de archivo de salida deseado.

Código

El código se encuentra en el siguiente repositorio, junto a instrucciones para su compilación y ejecución mediante QEMU. Se agrega también un .zip con el mismo.