**INTEGRANTES: Alexandra Hurtado**

**Mariana Valderrama**

**Descripción del problema**

Se desarrolló un programa que genera un código QR básico a partir de una cadena de texto. Para ello, se implementamos los elementos fundamentales de un QR estándar, como los marcadores de posición, los patrones de temporización, los datos codificados en binario, y una máscara que mejora la legibilidad. Además, se utilizó un algoritmo de corrección de errores (Reed-Solomon) para garantizar que el QR sea legible incluso si hay pequeños daños o errores.

**Paso a paso del proceso que seguimos en nuestro código**

Creación de una matriz base

Función: create\_qr\_matrix

Generamos una matriz cuadrada de 21x21 (versión 1 del QR) inicializada con ceros. Esta matriz representaría el QR final, donde:

1 indica un píxel negro.

0 indica un píxel blanco.

Adición de patrones de posición

Función: add\_position\_markers

Añadimos los patrones de posición (esquinas superior izquierda, superior derecha, e inferior izquierda). Estos consisten en cuadrados de 7x7 con bordes negros y un cuadrado blanco en el centro.

Adición de patrones de temporización

Función: add\_timing\_patterns

Colocamos líneas alternadas de píxeles blancos y negros

Definición de zonas reservadas

Función: is\_reserved

Establecimos reglas para identificar las áreas de la matriz que no se pueden modificar, como las que contienen: patrones de posición, patrones de temporización.

Conversión del texto en datos binarios

Función: text\_to\_binary

Generación de corrección de errores

Función: reed\_solomon\_correction

Utilizamos la librería Reed-Solomon para calcular los bytes de corrección de errores.

Inserción de datos en la matriz

Función: add\_data\_to\_qr

Insertamos la secuencia de datos binarios (texto + corrección de errores) en la matriz QR:

Los datos se colocaron en columnas alternas de derecha a izquierda, en un patrón de zigzag y

nos aseguramos de no sobrescribir las zonas reservadas.

Aplicación de una máscara

Función: apply\_mask

Aplicamos una máscara básica que invertía los bits de la matriz si (fila + columna) % 2 == 0.

Generación de la imagen

Función: matrix\_to\_image

Convertimos la matriz QR en una imagen de píxeles donde:

1 representa píxeles negros.

0 representa píxeles blancos.

*Guardado del QR como imagen*

Función: save\_qr\_image

Guardamos la imagen generada como un archivo PNG para visualizar el resultado final.

**Aprendizajes:**

A pesar de implementar todos los elementos esenciales para generar un código QR funcional como mencionamos anteriormente, no logramos producir un QR que pudiera ser leído por escáneres, este resultado nos dejó importantes aprendizajes:

1. Generar un QR funcional no es solo cuestión de colocar datos en una matriz; requiere un profundo entendimiento y precisión en la implementación de los estándares internacionales, incluyendo reglas específicas sobre el formato, las versiones, los niveles de corrección de errores y las máscaras.
2. Pequeños errores en la colocación de datos, patrones de sincronización o marcadores pueden hacer que el QR sea ilegible.
3. Cada intento, aunque no sea exitoso nos aporta información valiosa para refinar el código. cada error es una oportunidad para identificar que podemos mejorar y reforzarlo.

**Referencias de las cuales basamos los pasos a implementar en nuestro código:**

[**https://chatgpt.com/**](https://chatgpt.com/)

[**https://search.app?link=https%3A%2F%2Fwww.nayuki.io%2Fpage%2Fcreating-a-qr-code-step-by-step&utm\_campaign=aga&utm\_source=agsadl1%2Csh%2Fx%2Fgs%2Fm2%2F4**](https://search.app?link=https%3A%2F%2Fwww.nayuki.io%2Fpage%2Fcreating-a-qr-code-step-by-step&utm_campaign=aga&utm_source=agsadl1%2Csh%2Fx%2Fgs%2Fm2%2F4)

[**https://www.youtube.com/watch?v=4XTkiudd-\_E&t=882s**](https://www.youtube.com/watch?v=4XTkiudd-_E&t=882s)

[**https://www.quora.com/How-do-I-Develop-my-Own-Qr-Code-generator-without-using-Any-3rd-party-Qrcode-generator**](https://www.quora.com/How-do-I-Develop-my-Own-Qr-Code-generator-without-using-Any-3rd-party-Qrcode-generator)