

# UNIVERSIDAD DE GUADALAJARA



## CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

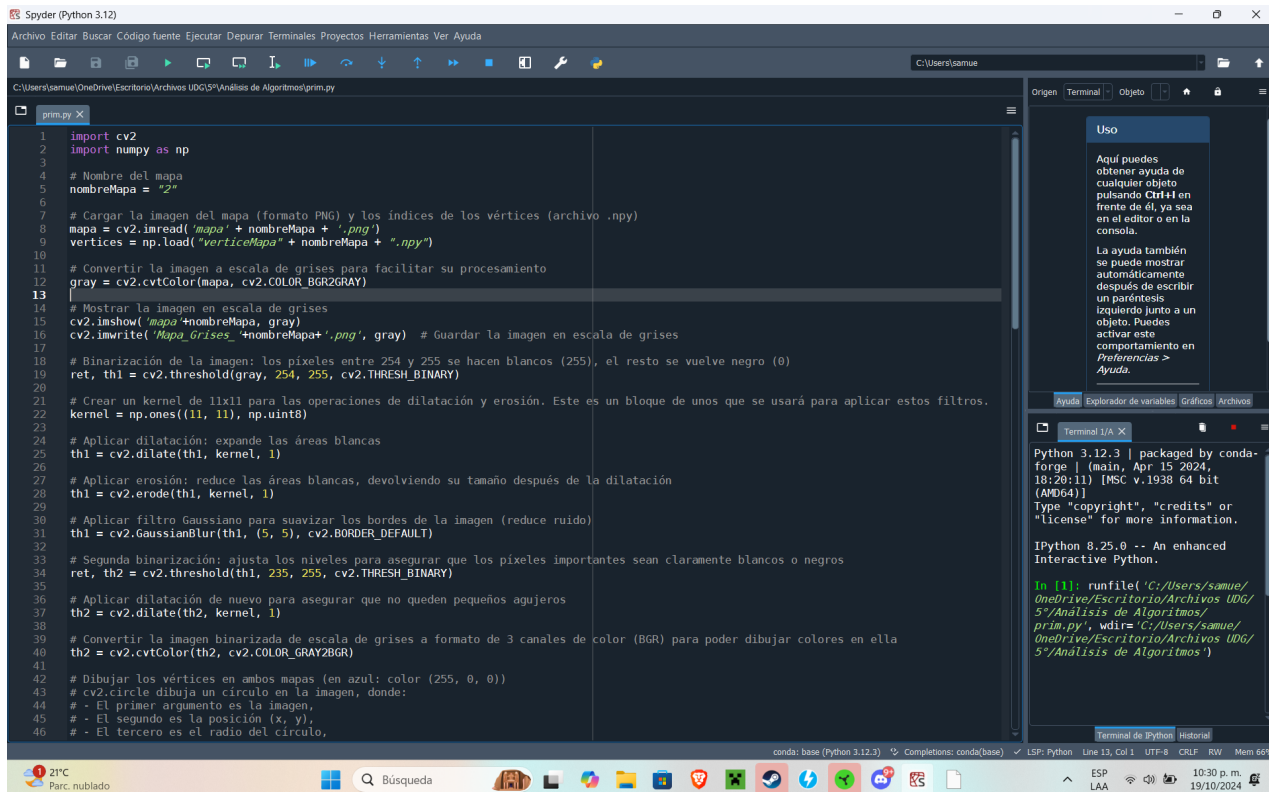
### Análisis de Algoritmos

#### Reporte de práctica

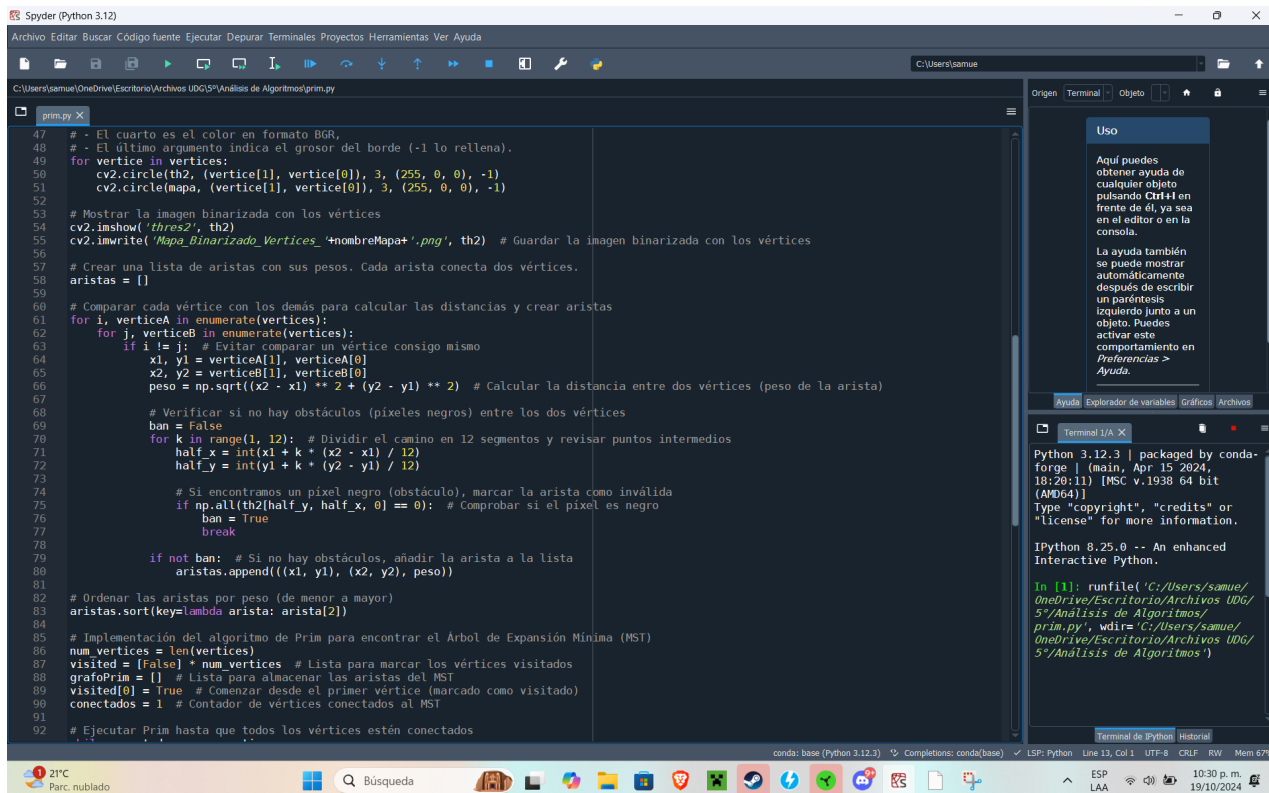
Nombre del alumno:	Samuel David Pérez Brambila
Código del alumno:	222966286
Profesor:	Erasmus Gabriel Martínez Soltero
Título de la práctica:	"Prim"
Fecha:	04 de Noviembre de 2024

# Metodología

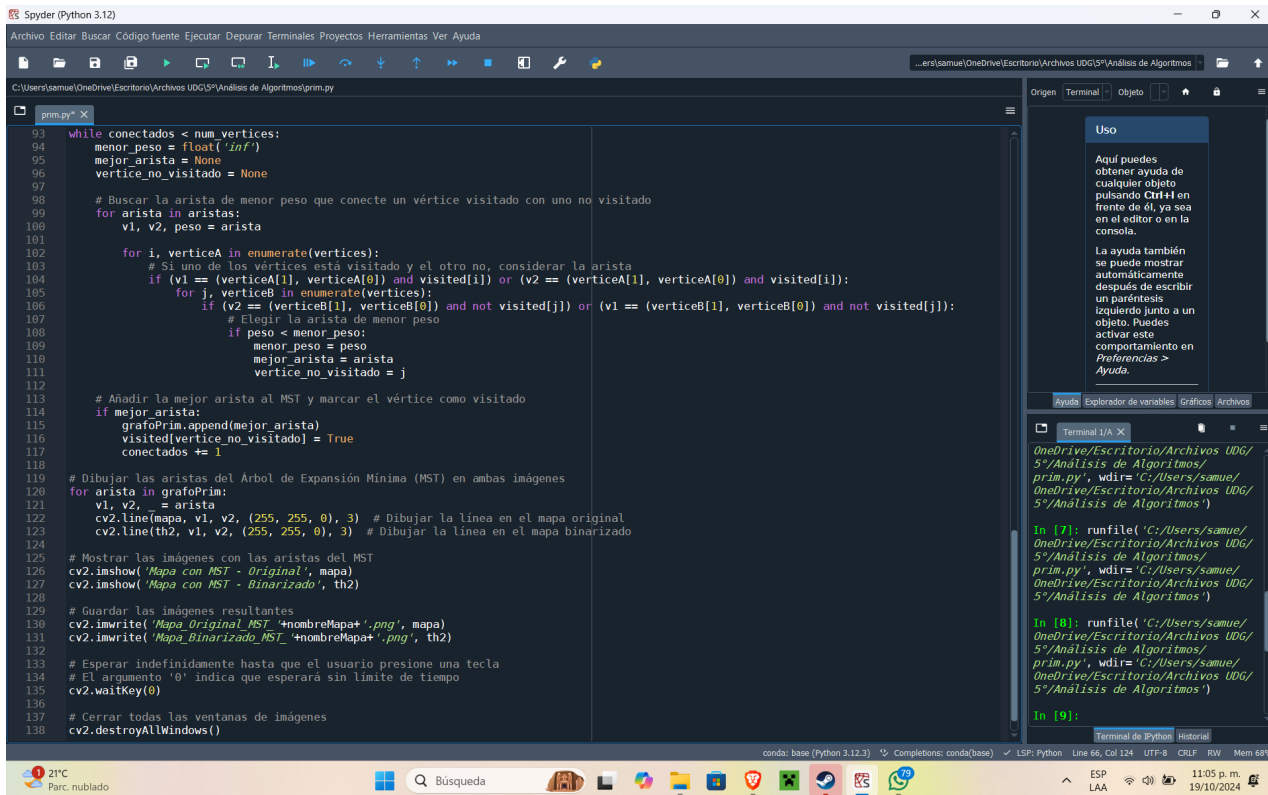
Código (Nota: Solamente se cambia el número o nombre en la variable nombreMapa para aplicar el algoritmo en otro):



```
1 import cv2
2 import numpy as np
3
4 # Nombre del mapa
5 nombreMapa = "2"
6
7 # Cargar la imagen del mapa (formato PNG) y los índices de los vértices (archivo .npy)
8 mapa = cv2.imread('mapa' + nombreMapa + '.png')
9 vertices = np.load('vertices' + nombreMapa + '.npy')
10
11 # Convertir la imagen a escala de grises para facilitar su procesamiento
12 gray = cv2.cvtColor(mapa, cv2.COLOR_BGR2GRAY)
13
14 # Mostrar la imagen en escala de grises
15 cv2.imshow('mapa' + nombreMapa, gray)
16 cv2.imwrite('Mapa_Grises_' + nombreMapa + '.png', gray) # Guardar la imagen en escala de grises
17
18 # Binarización de la imagen: los píxeles entre 254 y 255 se hacen blancos (255), el resto se vuelve negro (0)
19 ret, th1 = cv2.threshold(gray, 254, 255, cv2.THRESH_BINARY)
20
21 # Crear un kernel de 11x11 para las operaciones de dilatación y erosión. Este es un bloque de unos que se usará para aplicar estos filtros.
22 kernel = np.ones((11, 11), np.uint8)
23
24 # Aplicar dilatación: expande las áreas blancas
25 th1 = cv2.dilate(th1, kernel, 1)
26
27 # Aplicar erosión: reduce las áreas blancas, devolviendo su tamaño después de la dilatación
28 th1 = cv2.erode(th1, kernel, 1)
29
30 # Aplicar filtro Gaussiano para suavizar los bordes de la imagen (reduce ruido)
31 th1 = cv2.GaussianBlur(th1, (5, 5), cv2.BORDER_DEFAULT)
32
33 # Segunda binarización: ajusta los niveles para asegurar que los píxeles importantes sean claramente blancos o negros
34 ret, th2 = cv2.threshold(th1, 235, 255, cv2.THRESH_BINARY)
35
36 # Aplicar dilatación de nuevo para asegurar que no queden pequeños agujeros
37 th2 = cv2.dilate(th2, kernel, 1)
38
39 # Convertir la imagen binarizada de escala de grises a formato de 3 canales de color (BGR) para poder dibujar colores en ella
40 th2 = cv2.cvtColor(th2, cv2.COLOR_GRAY2BGR)
41
42 # Dibujar los vértices en ambos mapas (en azul: color (255, 0, 0))
43 # cv2.circle dibuja un círculo en la imagen, donde:
44 # - El primer argumento es la imagen,
45 # - El segundo es la posición (x, y),
46 # - El tercero es el radio del círculo,
```

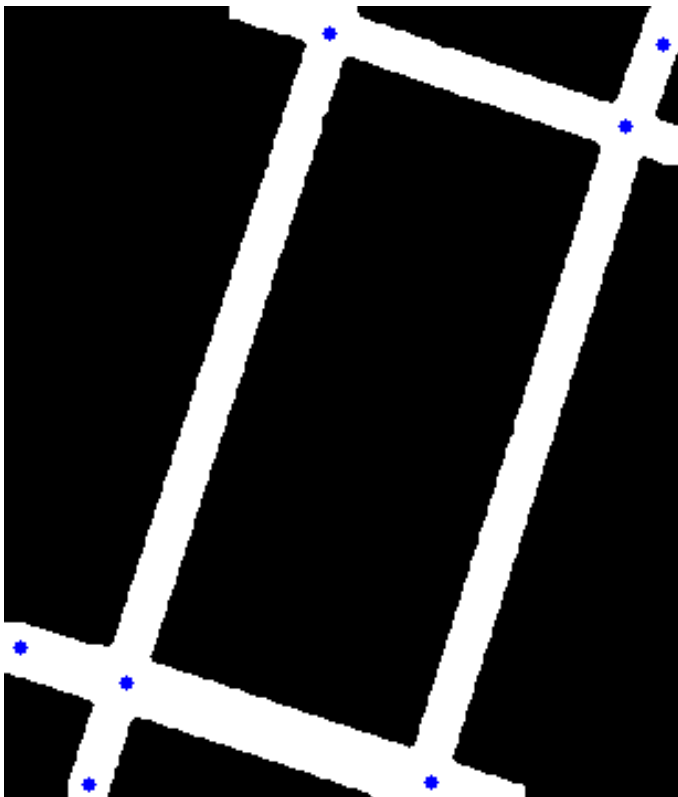


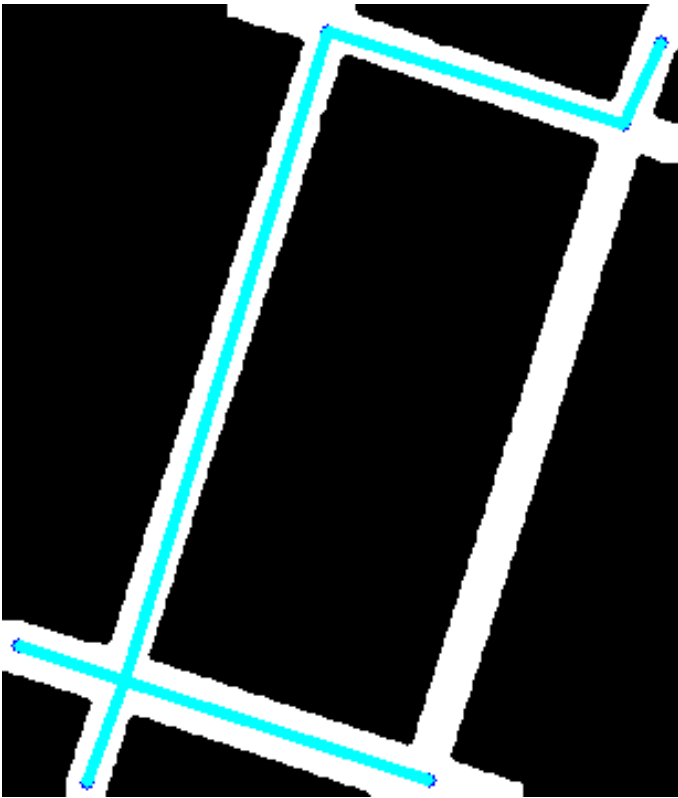
```
47 # - El cuarto es el color en formato BGR,
48 # - El último argumento indica el grosor del borde (-1 lo rellena).
49 for vertice in vertices:
50     cv2.circle(th2, (vertice[1], vertice[0]), 3, (255, 0, 0), -1)
51     cv2.circle(mapa, (vertice[1], vertice[0]), 3, (255, 0, 0), -1)
52
53 # Mostrar la imagen binarizada con los vértices
54 cv2.imshow('thres2', th2)
55 cv2.imwrite('Mapa_Binarizado_Vertices_' + nombreMapa + '.png', th2) # Guardar la imagen binarizada con los vértices
56
57 # Crear una lista de aristas con sus pesos. Cada arista conecta dos vértices.
58 aristas = []
59
60 # Comparar cada vértice con los demás para calcular las distancias y crear aristas
61 for i, verticeA in enumerate(vertices):
62     for j, verticeB in enumerate(vertices):
63         if i != j: # Evitar comparar un vértice consigo mismo
64             x1, y1 = verticeA[1], verticeA[0]
65             x2, y2 = verticeB[1], verticeB[0]
66             peso = np.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2) # Calcular la distancia entre dos vértices (peso de la arista)
67
68             # Verificar si no hay obstáculos (píxeles negros) entre los dos vértices
69             ban = False
70             for k in range(1, 12): # Dividir el camino en 12 segmentos y revisar puntos intermedios
71                 half_x = int(x1 + k * (x2 - x1) / 12)
72                 half_y = int(y1 + k * (y2 - y1) / 12)
73
74                 # Si encontramos un píxel negro (obstáculo), marcar la arista como inválida
75                 if np.all(th2[half_y, half_x, 0] == 0): # Comprobar si el píxel es negro
76                     ban = True
77                     break
78
79             if not ban: # Si no hay obstáculos, añadir la arista a la lista
80                 aristas.append(((x1, y1), (x2, y2), peso))
81
82 # Ordenar las aristas por peso (de menor a mayor)
83 aristas.sort(key=lambda arista: arista[2])
84
85 # Implementación del algoritmo de Prim para encontrar el Árbol de Expansión Mínima (MST)
86 num_vertices = len(vertices)
87 visited = [False] * num_vertices # Lista para marcar los vértices visitados
88 grafoPrim = [] # Lista para almacenar las aristas del MST
89 visited[0] = True # Comenzar desde el primer vértice (marcado como visitado)
90 conectados = 1 # Contador de vértices conectados al MST
91
92 # Ejecutar Prim hasta que todos los vértices estén conectados
```

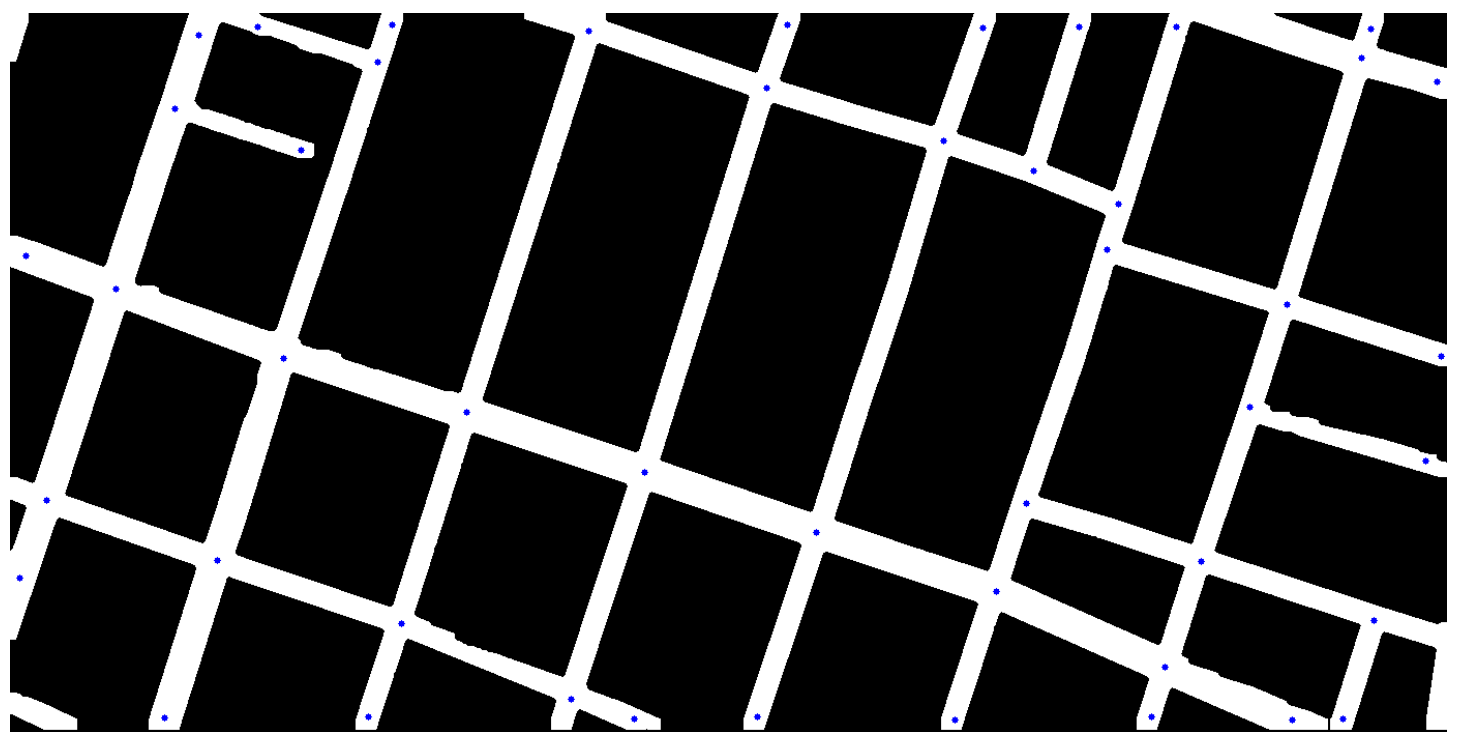


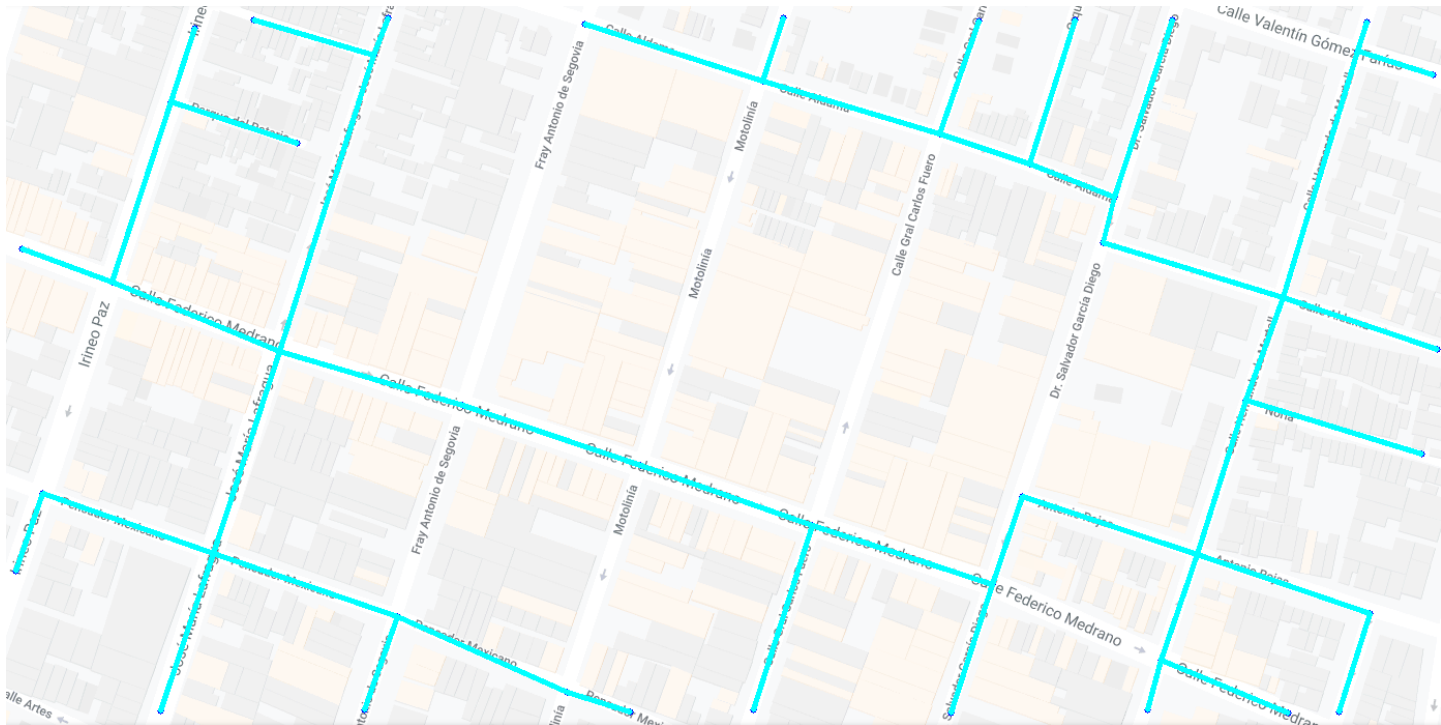
# Resultados

Mapa chico

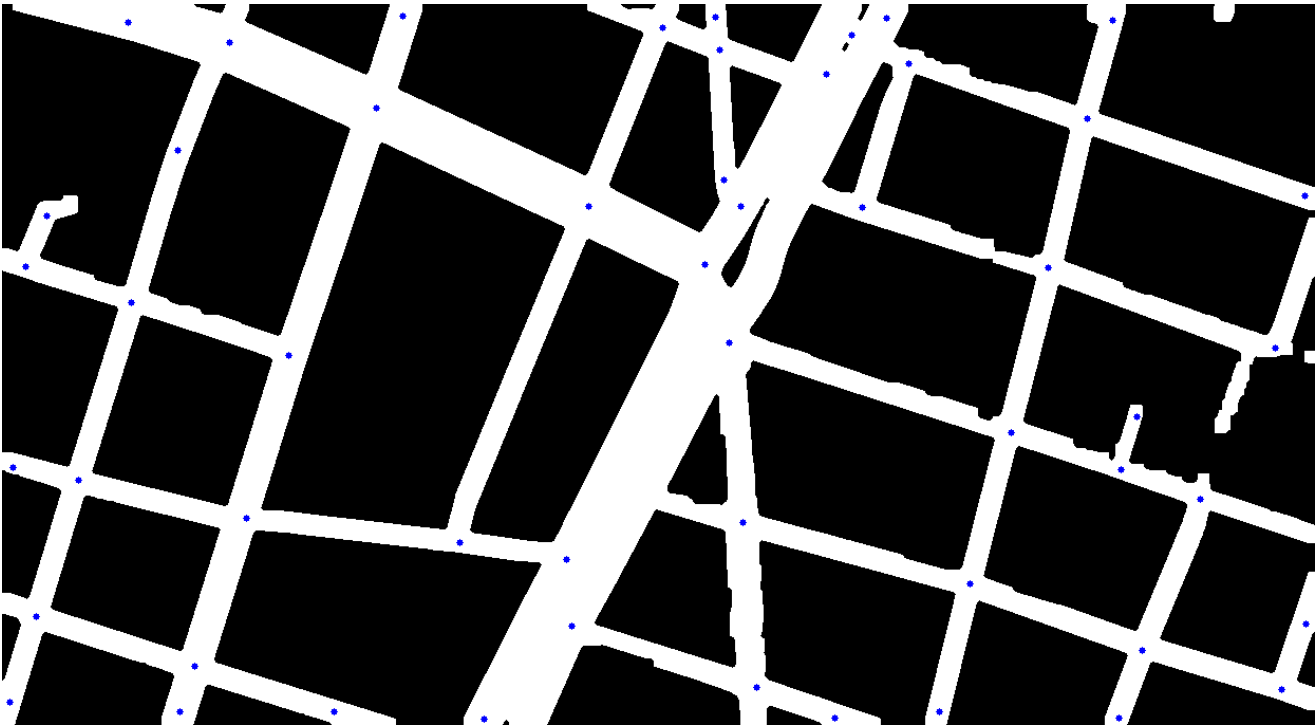




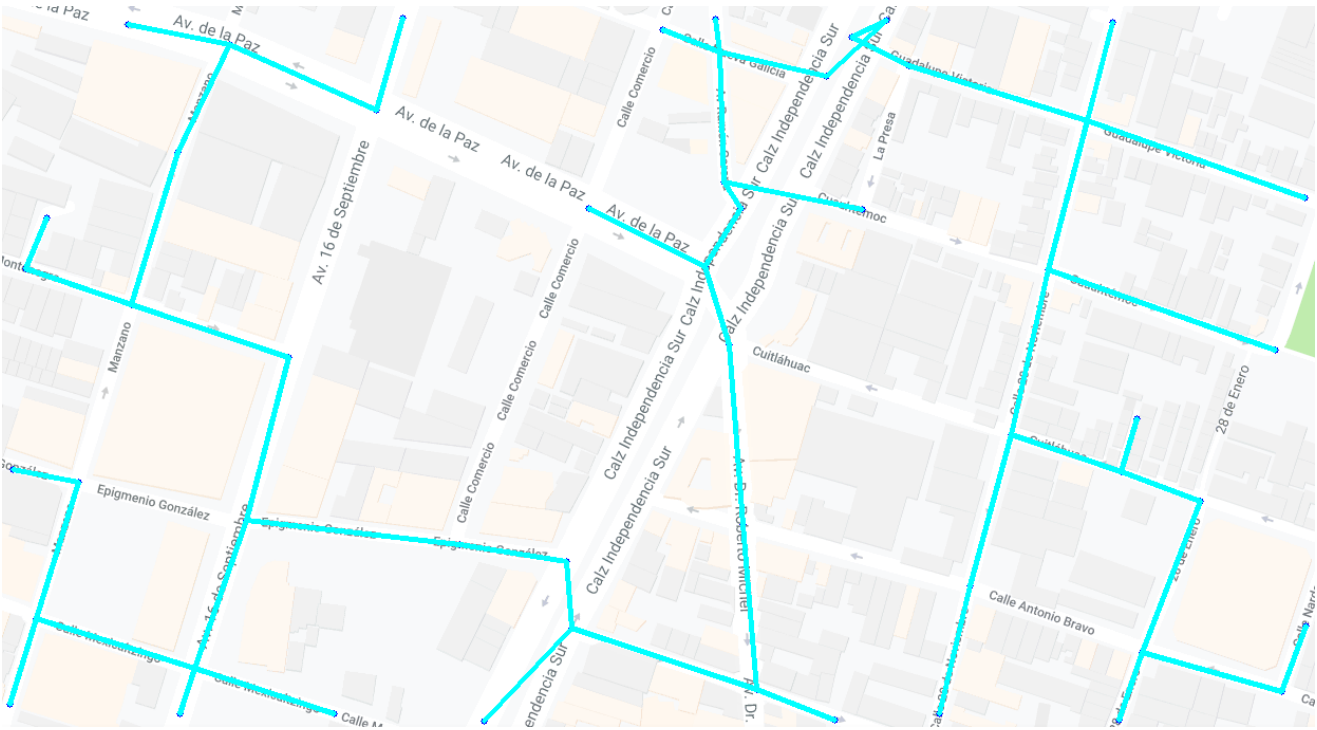
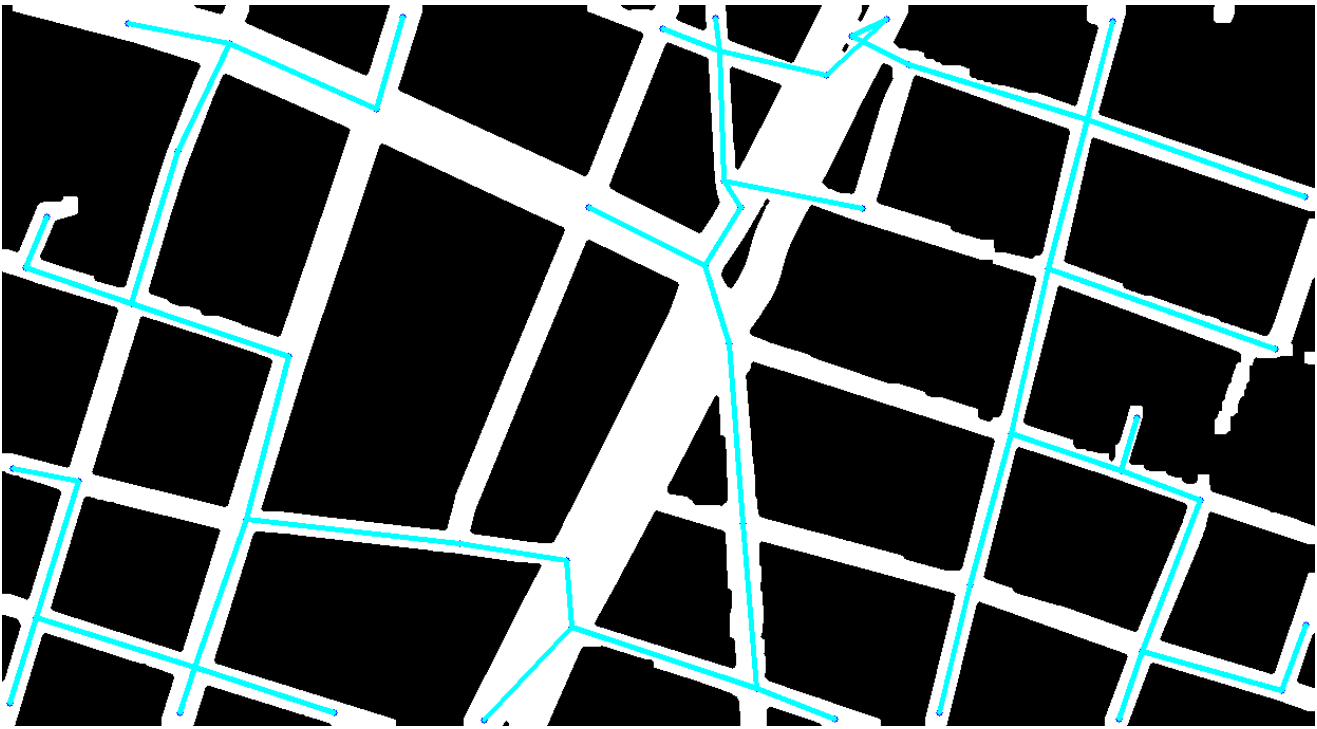




Mapa 2







A grayscale map of a city grid in Mexico City, showing streets like Calle Gigantes, Calle Valentín Gómez Farías, and Calle Reyes Flores. The map is tilted and includes street names in Spanish.

