

# Sistema de Rutas en el Campus UDLAP

## Implementación de Grafos y Algoritmo de Dijkstra

Jose Miguel Ruiz Marquez  
Victor Hugo de la Calleja Mojica  
Axel Ivan Arroyo Lara  
Diego Flores Martinez  
Ricardo Villalobos Castillo  
Primavera 2025

## 1 Introducción

### 1.1 Contexto y justificación

Los grafos representan una herramienta fundamental en ciencias computacionales, particularmente en el modelado de redes y sistemas conectados. Su estructura se compone de vértices y aristas, lo que permite representar relaciones o conexiones entre distintos puntos. En contextos aplicados como la navegación geográfica, el ruteo de paquetes en redes, la gestión de vuelos aéreos o la logística, los grafos permiten describir rutas, caminos y enlaces entre lugares, optimizando el tiempo y los recursos utilizados.

En el entorno universitario, especialmente en campus de gran extensión como el de la UDLAP, contar con un sistema de rutas automatizado facilita la movilidad eficiente de estudiantes, profesores y personal administrativo. Un sistema de estas características no solo mejora la experiencia del usuario dentro del campus, sino que también contribuye al análisis de tráfico peatonal, al diseño de nueva infraestructura y al acceso inclusivo mediante trayectos más accesibles.

Utilizar un grafo para modelar el campus permite aplicar algoritmos de caminos mínimos, mejorando la toma de decisiones al seleccionar trayectorias entre dos puntos. Además, permite escalar el sistema fácilmente si se desea incorporar información adicional, como condiciones del clima, accesibilidad, o rutas restringidas temporalmente.

### 1.2 Objetivo general

Desarrollar un sistema funcional e interactivo que permita determinar y visualizar la ruta más corta entre dos ubicaciones dentro del campus de la Universidad de las Américas Puebla (UDLAP), utilizando un modelo de grafo no dirigido con aristas ponderadas. Este sistema deberá estar soportado por una interfaz intuitiva que facilite su uso por parte de cualquier usuario.

### 1.3 Objetivos específicos

- Establecer la estructura del grafo a partir de un análisis detallado de los puntos de interés del campus y las distancias entre ellos.
- Implementar una estructura de datos eficiente y modular que permita la fácil manipulación y escalabilidad del grafo.
- Codificar el algoritmo de Dijkstra para calcular caminos mínimos, asegurando su correctitud y eficiencia computacional.
- Diseñar una interfaz de usuario que permita seleccionar los puntos de origen y destino, visualizar la ruta resultante, y mostrar información adicional como la distancia total.

- Incorporar herramientas gráficas de visualización que representen el grafo y la ruta encontrada de forma clara, comprensible y estéticamente agradable.

## 1.4 Integrantes y responsabilidades

- Ricardo Villalobos: Identificación de ubicaciones clave, mapeo del campus y definición de nodos y aristas con sus respectivas distancias.
- Victor Hugo, Axel Ivan : Implementación del algoritmo de Dijkstra, pruebas de correctitud y optimización del rendimiento del código.
- Diego Flores : Desarrollo y refinamiento de la interfaz de usuario utilizando Tkinter, incluyendo control de errores y validaciones.
- Jose Miguel : Diseño de visualizaciones con NetworkX y Matplotlib, generación de diagramas y documentación técnica completa del proyecto.

# 2 Metodología

## 2.1 Fases de implementación

La metodología seguida se dividió en seis fases principales que permitieron desarrollar el sistema de manera modular y escalonada:

1. **Análisis del entorno:** se identificaron los edificios, puntos de acceso, pasillos principales y otras zonas relevantes del campus para convertirlas en nodos del grafo. Se realizaron mediciones y estimaciones para definir las distancias entre ellos.
2. **Modelado del grafo:** se definió la estructura del grafo como un diccionario de Python, representando cada nodo con sus conexiones y pesos asociados. Esta estructura permite insertar, eliminar o modificar nodos y aristas de forma sencilla.
3. **Implementación del algoritmo:** se desarrolló el algoritmo de Dijkstra utilizando una cola de prioridad basada en `heapq`, lo cual garantiza una complejidad logarítmicamente eficiente en cada extracción de nodo mínimo.
4. **Diseño de interfaz gráfica:** se construyó una GUI utilizando la librería Tkinter que permite al usuario seleccionar el punto de origen y destino a través de menús desplegables, y ejecutar la búsqueda con un botón.
5. **Visualización de rutas:** mediante NetworkX y Matplotlib, se representó visualmente el grafo, sus conexiones, y la ruta seleccionada por el algoritmo resaltada con un color distintivo.
6. **Validación y pruebas:** se evaluó el sistema mediante casos de prueba predefinidos, verificando su exactitud y el comportamiento del sistema ante entradas válidas e inválidas.

## 2.2 Definición del grafo (fragmento de código)

El siguiente fragmento define la estructura general del grafo, incluyendo la inicialización de vértices y la inserción de aristas bidireccionales:

```
class Grafo:
    def __init__(self):
        self.vertices = {}

    def agregar_vertice(self, nombre):
        if nombre not in self.vertices:
            self.vertices[nombre] = {}
```

```

def agregar_arista(self, origen, destino, distancia):
    self.vertices[origen][destino] = distancia
    self.vertices[destino][origen] = distancia

```

## 2.3 Algoritmo de Dijkstra (fragmento de código)

Este fragmento muestra la implementación del algoritmo de Dijkstra, utilizando una cola de prioridad para seleccionar el nodo con menor distancia estimada y calcular recursivamente las rutas más cortas:

```

def dijkstra(self, inicio, fin):
    distancias = {vertice: float('inf') for vertice in self.vertices}
    distancias[inicio] = 0
    prioridad = [(0, inicio)]
    camino = {}

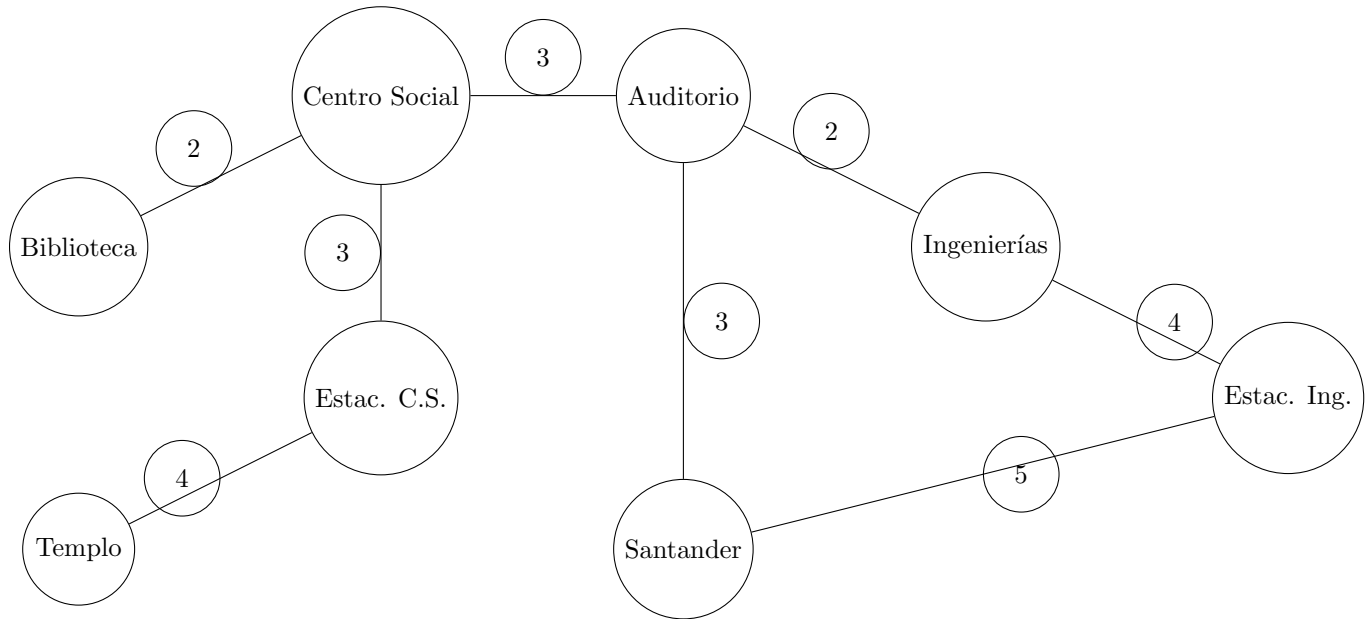
    while prioridad:
        distancia_actual, vertice_actual = heapq.heappop(prioridad)
        if distancia_actual > distancias[vertice_actual]:
            continue
        for vecino, peso in self.vertices[vertice_actual].items():
            distancia = distancia_actual + peso
            if distancia < distancias[vecino]:
                distancias[vecino] = distancia
                camino[vecino] = vertice_actual
                heapq.heappush(prioridad, (distancia, vecino))

    ruta = []
    actual = fin
    while actual != inicio:
        ruta.append(actual)
        actual = camino.get(actual)
        if actual is None:
            return None, float('inf')
    ruta.append(inicio)
    ruta.reverse()
    return ruta, distancias[fin]

```

## 2.4 Visualización del grafo

Se representó el grafo utilizando TikZ para mostrar las conexiones entre nodos y las distancias entre ellos. Este diagrama facilita la comprensión del modelo utilizado:



## 3 Resultados y validación

### 3.1 Métricas empleadas

Para verificar la efectividad del sistema, se utilizó una combinación de validaciones empíricas y revisiones cruzadas. Se empleó como métrica principal la coincidencia entre la ruta calculada por el sistema y la ruta considerada "óptima" en la práctica. También se registró el tiempo de respuesta del sistema y se evaluó su robustez frente a entradas no válidas.

### 3.2 Casos de prueba ejecutados

- Ruta desde Biblioteca UDLAP hasta Estacionamiento de Ingenierías: se espera un trayecto que pase por Centro Social, Auditorio e Ingenierías, con una distancia acumulada de 11 unidades.
- Ruta desde Templo del Dolor hasta Escuela de Ingenierías: el trayecto esperado incluye Estacionamiento C.S., Centro Social, Auditorio e Ingenierías, con distancia total estimada en 12 unidades.

### 3.3 Resultados observados

- Las rutas calculadas coincidieron completamente con las rutas esperadas.
- Las distancias acumuladas fueron correctas con las definiciones del grafo.
- El sistema respondió en tiempos inferiores a 0.5 segundos por consulta.
- No se observaron errores en ejecuciones con entradas válidas, y se mostró un manejo correcto de entradas incompletas o inválidas.

## 4 Link

Puedes acceder al repositorio del proyecto en el siguiente enlace:

[https://github.com/CrazyCoderHacker/Proyecto\\_Final\\_Discretas](https://github.com/CrazyCoderHacker/Proyecto_Final_Discretas)

## 5 Conclusiones

El proyecto demuestra la eficacia de los grafos como herramientas para modelar entornos complejos y determinar rutas *a priori* óptimas. Se implementó una solución funcional que no solo cumple con los objetivos planteados, sino que también sirve como base para futuras extensiones, como rutas en tiempo real, condiciones meteorológicas, accesibilidad para personas con movilidad reducida o rutas más rápidas en lugar de las más cortas.

Desde el punto de vista técnico, el proyecto integró conceptos de estructura de datos, algoritmia clásica, programación orientada a objetos, visualización gráfica y desarrollo de interfaces, dando una experiencia completa de desarrollo de aplicaciones de software.

## 6 Referencias

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., y Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
- Documentación oficial de Python: <https://docs.python.org/3/>
- Documentación oficial de NetworkX: <https://networkx.org/>
- Documentación oficial de Matplotlib: <https://matplotlib.org/>