Universidad de las Américas Puebla


# Proyecto Final de Matemáticas Discretas y Base de datos

## Camino más corto para edificios de la Universidad

Matemáticas Discretas y Base de datos

Nombre del Profesor

Nombre del Departamento

*Trabajo presentado por:*

Isabela Constantino Aguilar

Víctor Dávila Desdier

Emma Duarte Mercado

Diego Iván Martínez Escobar

Ingeniería en Sistemas Computacionales

San Andrés Cholula, 14 de junio de 2024

## Introducción

Los grafos son estructuras de datos que sirven como una herramienta para demostrar las conexiones de elementos (nodos) con otros. Con los grafos es posible identificar los caminos (aristas) entre varios elementos. Dentro de este trabajo se representarán los edificios dentro de nuestra universidad y se asignarán las distancias y relaciones válidas. A partir de esta ilustración y recordando que a estos caminos se les asignará un peso, esto significa que existirá un camino más eficiente, y este se podrá encontrar implementando herramientas como Dijkstra, para analizar el camino con menor costo.

¿Por qué utilizar Dijkstra? Gracias a este enfoque, en la actualidad, herramientas de navegación modernas (p. ej. Google Maps) pueden ofrecer rutas dinámicas que se adaptan a obras, cierres de calles o tráfico en tiempo real. En la aplicación al campus universitario, donde múltiples senderos y edificaciones se entrelazan, los grafos son la pieza clave para brindar indicaciones precisas y rápidas para obtener el camino más eficiente.

En este proyecto, los edificios se representan como nodos y los pesos como distancias. Gracias a una base de datos relacional se tienen todos los edificios y las distancias de un edificio origen a uno destino. Aprovechando sus características se logró modelar la información de manera ordenada y lograr la aplicación correctamente. Dentro del siguiente reporte se encontrará nuestra documentación sobre la realización del proyecto.

### Objetivo general

Generar una herramienta de búsqueda de rutas entre edificios en el campus UDLAP, con el propósito de encontrar la más optima de un edificio origen a un edificio destino, con la implementación de una base de datos, y una interfaz gráfica para una interacción con el usuario.

### Objetivos específicos

- Extraer las relaciones de lo edificios para la base de datos.
- Generar la Base de datos con datos medidos

- Programar un código en base a grafos con el algoritmo de Dijkstra.
- Implementar la base de datos al código para probarlo con datos reales.
- Diseñar una Interfaz para la visualización de el campus y las rutas.

**Integrantes**

- **Diego Iván:** Diseñador principal de Interfaz gráfica.
- **Isabela Constantino:** Investigadora de datos y creadora de base de datos
- **Víctor Dávila:** Programador de código con el algoritmo de Dijkstra.
- **Emma Duarte:** Encargada de documentación del código, interfaz y base de datos del proyecto.

**Base de datos**

**Obtención de los datos**

Para lograr obtener las relaciones y los datos, y mantener la par a la realidad lo más posible. Para la extracción de datos se utilizó la herramienta ”Distancia a este punto” en Google Maps, utilizando como puntos las entradas principales de los edificios. Para determinar las relaciones por edificios, se siguieron líneas rectas entre sus entradas. En caso de que el camino se obstruyera por algún otro edificio o cuerpo de agua, la relación no tomaba lugar. El último punto a recalcar es que las mediciones fueron entregadas en formato de metros a dos decimales. Para la obtención de los nombres y abreviación de los nombres de los edificios se utilizaron los nombres oficiales que se pueden encontrar en cualquier mapa de la universidad, en este caso los extrajimos de UDLAPlife.

La implementación de la base de datos cuenta con dos tablas, y se logra respetar 3FN, pues cumple con 2FN y no se observan dependencias transitivas.

- **Edificios**: esta tabla tendrá todos los edificios que se encuentran en el campus de la UDLAP, además tendrá su correspondiente identificador.

- **Conexiones**: esta tabla tendrá todas las conexiones entre 2 edificios, no dirigido, evitando redundancia. Para definir esas conexiones tendremos los campos de edifico origen, edificio destino y su distancia (que funcionara como el peso).

### Modelado de los datos

La representación del diagrama entidad-relación es la siguiente:

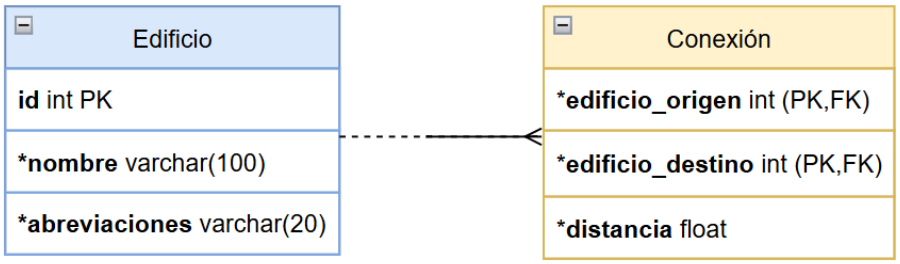



Figura 1: Representación gráfica

### Reglas de negocio

Dentro de este trabajo, ninguna pieza de información podía faltar para lograr mantener la integridad y coherencia de los datos, por lo que las reglas de negocio resultan bastante predecibles.

- `UNIQUE KEY`: de la siguiente forma se asegura que los valores de una columna sean únicos, lo cual es esencial dentro de áreas como identificadores. Se puede observar en:
  - **id**
- `NOT NULL` : con esta instrucción se prohíbe que se asigne un valor NULL dentro del área, lo que garantiza la consistencia de los datos. Dentro de nuestro diseño se observa esta restricción en las áreas:
  - **nombre**
  - **abreviacion**
  - **edificio_origen**

- **edificio_destino**
- **distancia**

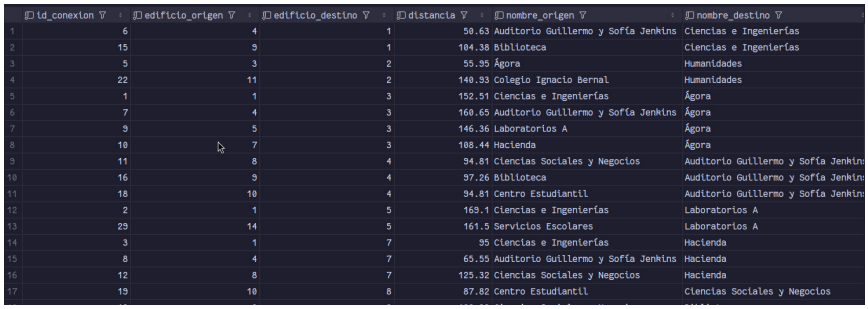
- `REFERENCES`: esta política asegurará que todas las conexiones creadas utilicen edificios existentes. Dentro de nuestra implementación se observa en:
  - **edificio_origen**
  - **edificio_destino**

### Implementación

Aunque se verán más sobre las consultas dentro de la sección desarrollo del sistema, el equipo agregó dentro del MySQL una vista para lograr entender todos los datos que fuéramos agregando. Esta vista tiene el nombre "vista_conexciones" es una consulta que brinda información de las tablas, conexiones y edificios, representándola de manera más legible y útil. La vista se genera con el siguiente código:

```
CREATE TABLE `conexiones` (
  `id` INT PRIMARY KEY AUTO_INCREMENT,
  `edificio_origen` INT NOT NULL,
  `edificio_destino` INT NOT NULL,
  `distancia` FLOAT NOT NULL,
  UNIQUE KEY `uq_origen_destino` (`edificio_origen`, `edificio_destino`),
  FOREIGN KEY (`edificio_origen`) REFERENCES `edificios` (`id`),
  FOREIGN KEY (`edificio_destino`) REFERENCES `edificios` (`id`)
);
```

Proporciona el ID único de la conexión, de los edificios involucrados, la distancia entre ellos y los nombres completos de los edificios. La relación que se genera es entre **edificio_origen** y **edificio_destino** con sus respectivos **edificios.id**.



| | id_conexion | edificio_origen | edificio_destino | distancia | nombre_origen | nombre_destino |
|---|---|---|---|---|---|---|
| 1 | 6 | 4 | 1 | 50.63 | Auditorio Guillermo y Sofía Jenkins | Ciencias e Ingenierías |
| 2 | 15 | 9 | 1 | 104.38 | Biblioteca | Ciencias e Ingenierías |
| 3 | 5 | 3 | 2 | 55.95 | Ágora | Humanidades |
| 4 | 22 | 11 | 2 | 140.93 | Colegio Ignacio Bernal | Humanidades |
| 5 | 1 | 1 | 3 | 152.51 | Ciencias e Ingenierías | Ágora |
| 6 | 7 | 4 | 3 | 160.65 | Auditorio Guillermo y Sofía Jenkins | Ágora |
| 7 | 9 | 5 | 3 | 146.36 | Laboratorios A | Ágora |
| 8 | 10 | 7 | 3 | 108.44 | Hacienda | Ágora |
| 9 | 11 | 8 | 4 | 94.81 | Ciencias Sociales y Negocios | Auditorio Guillermo y Sofía Jenkins |
| 10 | 16 | 9 | 4 | 97.26 | Biblioteca | Auditorio Guillermo y Sofía Jenkins |
| 11 | 18 | 10 | 4 | 94.81 | Centro Estudiantil | Auditorio Guillermo y Sofía Jenkins |
| 12 | 2 | 1 | 5 | 169.1 | Ciencias e Ingenierías | Laboratorios A |
| 13 | 29 | 14 | 5 | 161.5 | Servicios Escolares | Laboratorios A |
| 14 | 3 | 1 | 7 | 35 | Ciencias e Ingenierías | Hacienda |
| 15 | 8 | 4 | 7 | 65.55 | Auditorio Guillermo y Sofía Jenkins | Hacienda |
| 16 | 12 | 8 | 7 | 125.32 | Ciencias Sociales y Negocios | Hacienda |
| 17 | 19 | 10 | 8 | 87.82 | Centro Estudiantil | Ciencias Sociales y Negocios |

Figura 2: Ejemplo resultado de vista_conexciones

## Desarrollo del sistema

### Fases en el que se descompone

### Servidor (BackEnd)

Para el servidor se utilizará API REST (Representational State Transfer API), una interfaz de programación de aplicaciones (API) que utiliza el protocolo HTTP para interactuar con recursos; en este caso; escuchará el puerto 3000.

**Escuchar al puerto 300**

```
const PORT = 3000;
app.listen(PORT, () => {
    console.log(`Servidor backend corriendo en http://localhost${PORT}`);
});
```

El propósito es poder escuchar peticiones HTTP que produce el usuario o el mismo programa. Las peticiones generarán una query a nuestra base de datos para obtener información dependiendo de la petición. La información se obtendrá en formato JSON para facilitar su representación gráfica.

- La petición "`Conexiones`" devolverá Todas las conexiones entre edificios en el formato JSON

**Obtener conexiones**

```
router.get('/', (_req, res) => {
    const query = 'SELECT id, edificio_origen AS source,
    ↪  edificio_destino AS target, distancia FROM conexiones';
    db.query(query, (err, results) => {
        if (err) {
            console.error('Error al obtener conexiones:', err);
            return res.status(500).json({ error: 'Error al obtener
            ↪  conexiones' });
        }
        res.json(results);
    });
});
```

- La petición "`Edificios`" devolverá Todos los edificios en el formato JSON

**Obtener edificios**

```
router.get('/', (_req, res) => {
    const query = 'SELECT id, nombre, abreviacion FROM edificios ORDER
    ↪  BY nombre';
    db.query(query, (err, results) => {
        if (err) {
            console.error('Error al consultar edificios:', err);
            return res.status(500).json({ error: 'Error al obtener
            ↪  edificios' });
        }
        res.json(results);
    });
});
```

- **Camino**, en la petición camino se tendrán 2 parámetros, el parámetro `inicio` y el

`final`, que serán los ID de los edificios. En esta se hará una query a la nuestra base de datos para la vista de conexiones

```c
onst [conexiones] = await db.promise().query('SELECT * FROM
↪  vista_conexiones');
```

Con las conexiones y los IDs se construirán los grafos, en este caso se representaran como lista de adyacencia.

**Construir el grafo**

```
function construirGrafo(conexiones) {
const grafo = {};

for (const { id_conexion, edificio_origen, edificio_destino,
↪ distancia } of conexiones) {
    if (!grafo[edificio_origen]) grafo[edificio_origen] = [];
    if (!grafo[edificio_destino]) grafo[edificio_destino] = [];

    grafo[edificio_origen].push({
        nodo: edificio_destino,
        peso: distancia,
        conexionId: id_conexion
    });

    grafo[edificio_destino].push({
        nodo: edificio_origen,
        peso: distancia,
        conexionId: id_conexion
    }); // Asumiendo que es bidireccional
}

return grafo;
}
```

Después de construir el grafo se ejecutara Dijkstra para encontrar el mejor camino. con la función `function dijkstra(grafo, inicio, fin)` en esta tendrá como diferencia que cada vez que avance se guardará la información del nodo. Como último paso reconstruirá el camino previo, de regreso.

**FrontEnd**

Para el front end se utilizará REACT JS, y la librería `react-force-graph-2d` para una mejor visualización. Se construirá una interfaz, en el cual se harán querys al servidor para sacar la información. Las diferentes consultas serán:

- Edificios - Con el cual mostrara los edificios
- Conexiones - Con el cual mostrara las conexiones entre los edificios
- Camino - el cual servirá para dibujar la ruta más optima

### Diagrama

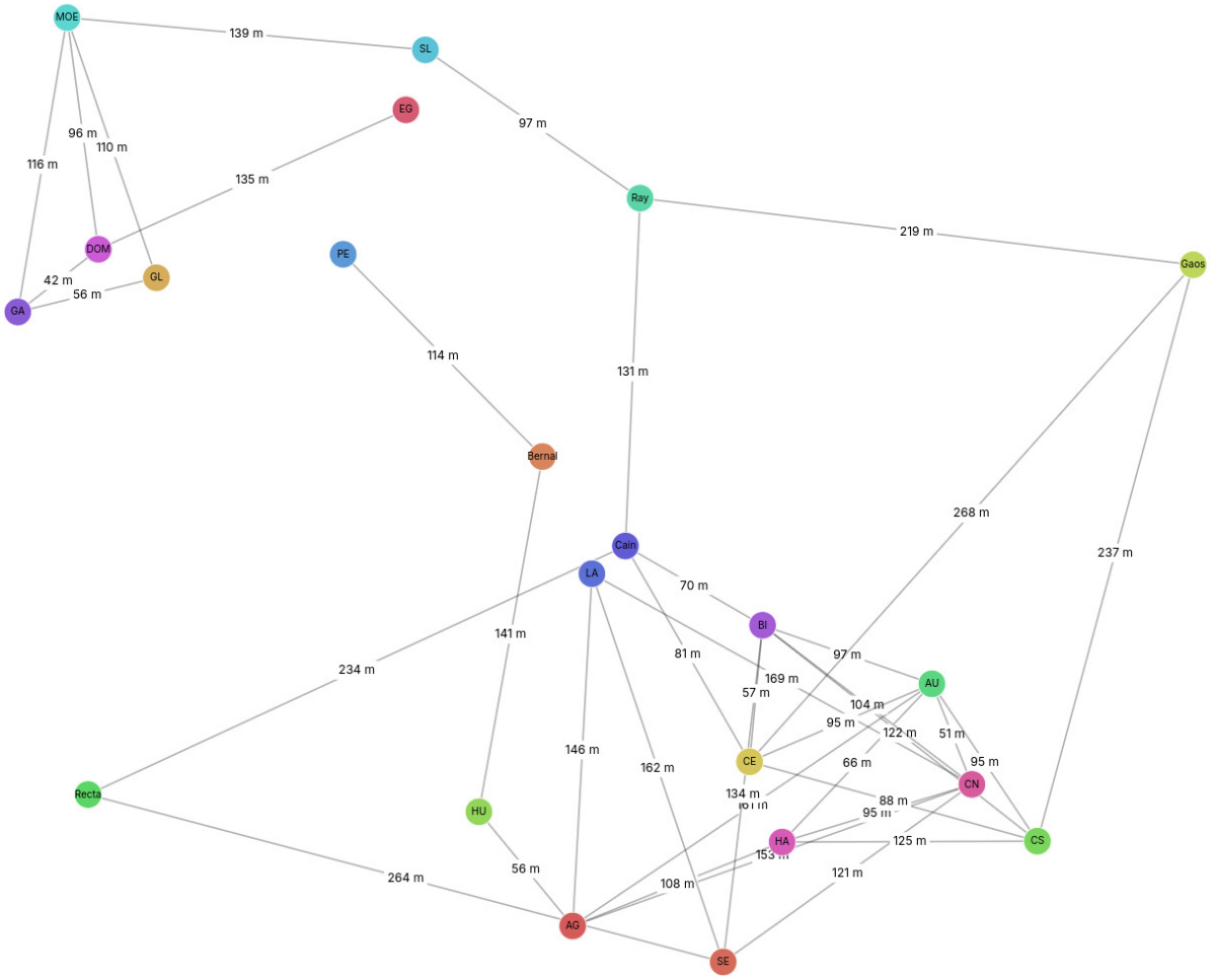



Figura 3: Grafo de los edificios de la universidad

## Evaluación

Para evaluar el funcionamiento del sistema de navegación, se compararon las rutas calculadas con distancias reales del campus UDLAP. Estas distancias fueron obtenidas utilizando Google Maps como referencia, permitiendo estimar con mayor precisión el peso entre nodos. Con la información obtenida fue preciso obtener la mejor ruta por medio de cálculos, así comprobando nuestros resultados.

Además, se registraron los tiempos de respuesta ante distintas consultas y se verificó el comportamiento del sistema ante datos incorrectos o incompletos.

**Validez del algoritmo**

- Se probó que el algoritmo de Dijkstra encuentra siempre la ruta más corta entre dos edificios, validándolo con ejemplos manuales y comparaciones visuales
- Se usaron al menos 10 combinaciones de rutas para verificar que los resultados obtenidos coincidieran con las distancias esperadas.

**Integración con base de datos**

- Se verificó que las consultas a las tablas edificios y conexiones devuelvan datos correctos y en el formato esperado.
- Se realizaron pruebas de inserción, actualización y eliminación para comprobar la consistencia de los datos.

**Rendimiento**

- El sistema respondió a las consultas de caminos en menos de 1 segundo en todas las pruebas.
- No se presentaron errores de conexión o caídas del servidor durante las pruebas.

## Conclusiones

El proyecto permitió diseñar un sistema de navegación para el campus UDLAP, integrando una base de datos relacional con estructuras de grafos. Al representar edificios como nodos y distancias como pesos, se logró calcular rutas óptimas entre distintos puntos del campus utilizando el algoritmo de Dijkstra.

El uso de datos reales obtenidos con Google Maps mejoró la precisión del sistema, y su tiempo de respuesta fue eficiente. Este desarrollo demuestra cómo bases de datos y

algoritmos pueden combinarse para resolver problemas cotidianos de manera funcional y escalable.

## Referencias

[1] GeeksforGeeks. (s.f.). *Graph data structure and algorithms*. Recuperado el 4 de mayo de 2025, de https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

[2] GeeksforGeeks. (s.f.). *Dijkstra's shortest path algorithm*. Recuperado el 4 de mayo de 2025, de https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm/

[3] freeCodeCamp. (s.f.). *A* search algorithm*. Recuperado el 4 de mayo de 2025, de https://www.freecodecamp.org/news/a-star-search-algorithm/

[4] PostgreSQL Global Development Group. (2025). *WITH queries (Common Table Expressions)*. Recuperado el 4 de mayo de 2025, de https://www.postgresql.org/docs/current/queries-with.html

[5] Data to Fish. (2021). *Storing graphs in a relational database*. Recuperado el 4 de mayo de 2025, de https://www.datatofish.com/graph-database-vs-relational-database/