Implementación de un Algoritmo de Búsqueda para Rutas Óptimas en una Red de Metro

Proyecto de Materia

Materia: búsqueda de soluciones e inferencia bayesiana.

Universidad Autónoma del Estado de Morelos

Presentado por:
Luis Josué Cortés Anzurez

Resumen

En este proyecto, se desarrolla un algoritmo de búsqueda para encontrar el mejor camino entre dos puntos en una red de metro. El algoritmo no solo minimiza el número de estaciones y transbordos, sino que también respeta restricciones específicas sobre estaciones por las que se debe o no se debe pasar. La implementación considera la importancia relativa de estas restricciones, proporcionando una solución que puede no ser óptima, pero sí válida en muchos casos. Se describen los datos de entrada, las restricciones y los resultados obtenidos, los cuales demuestran la eficacia del algoritmo en distintos escenarios.

Introducción

En las grandes ciudades, el sistema de transporte público, especialmente el metro, juega un papel crucial en la movilidad de millones de personas diariamente. Sin embargo, encontrar la ruta más eficiente para llegar de una estación a otra puede ser un desafío significativo, especialmente cuando se desconocen las mejores opciones de transbordo o se necesita evitar ciertas estaciones. Este proyecto surge de una experiencia personal al utilizar el metro, donde me encontré en la dificultad de planificar mi trayecto de manera óptima debido a la falta de una guía clara sobre las mejores rutas y transbordos.

El objetivo principal de este proyecto es desarrollar un algoritmo de búsqueda capaz de encontrar la mejor ruta en una red de metro, considerando diversas restricciones. La calidad de la ruta se mide no solo por el número de estaciones y transbordos, sino también por la capacidad de cumplir con restricciones específicas, como evitar ciertas estaciones o pasar por otras obligatorias. Este tipo de herramienta puede ser extremadamente útil tanto para los usuarios del metro como para los planificadores urbanos que buscan mejorar la eficiencia del sistema de transporte.

Recolección de Datos

Para la implementación de este proyecto, los datos sobre las líneas y estaciones del metro fueron recolectados manualmente de las páginas oficiales del gobierno y de fuentes confiables en línea. Esta recopilación manual fue necesaria para asegurar que los datos estuvieran en el formato adecuado para su uso en el algoritmo de búsqueda. Los datos incluyen la lista de estaciones para cada línea de metro y las conexiones entre estaciones donde se pueden realizar transbordos.

La estructura de los datos se define de la siguiente manera:

- **Líneas de metro:** Cada línea se representa como una lista de estaciones. Por ejemplo, la línea 1 se define como

  linea_1 = ["observatorio", "tacubaya", "juanacatlan", "chapultepec", "sevilla", "insurgentes", "cuauhtemoc", "balderas", "salto_del_agua", "isabel_la_catolica", "pino_suarez", "merced", "candelaria", "san_lazaro", "moctezuma", "balbuena", "blvd_puerto_aereo", "gomez_farias", "zaragoza", "pantitlan"].

- **Conexiones entre estaciones:** Se utiliza un diccionario para definir las estaciones donde se pueden realizar transbordos y las líneas a las que se puede cambiar en esas estaciones. Por ejemplo, las conexiones de la línea 1 se definen como conexiones = {"tacubaya": [7, 9], "balderas": [3], "salto_del_agua": [8], "pino_suarez": [2], "candelaria": [4], "san_lazaro": [11], "pantitlan": [5, 9, 10]}.

Descripción del Problema

El problema de búsqueda de rutas en una red de metro se puede definir formalmente de la siguiente manera:

- **Entrada:** Una red de metro con múltiples líneas y estaciones. Cada línea y estación está claramente definida, junto con las conexiones posibles para transbordos.
- **Restricciones:** El trayecto debe comenzar y terminar en estaciones específicas, minimizando el número de estaciones y transbordos. Además, el trayecto debe evitar ciertas estaciones y pasar por otras, con la posibilidad de asignar prioridades a estas restricciones.
- **Salida:** La secuencia de estaciones que forma la ruta óptima, el número de transbordos realizados, y la verificación de si el trayecto respeta todas las restricciones.

Motivación

La motivación personal para este proyecto provino de la necesidad de planificar viajes en el metro de manera más eficiente. En múltiples ocasiones, me encontré en la situación de no saber cuál era la mejor ruta para llegar a una estación dada o dónde debía transbordar para llegar a mi destino final. Esta experiencia me llevó a considerar la creación de una herramienta que no solo planificara la ruta más corta, sino que también tuviera en cuenta diversas restricciones que pueden ser importantes para los usuarios del metro.

Relevancia

La relevancia de este proyecto se extiende más allá de la experiencia personal. En el contexto urbano actual, donde las redes de transporte son cada vez más complejas, contar con herramientas que faciliten la planificación de rutas puede mejorar significativamente la eficiencia del transporte público. Además, este tipo de algoritmos pueden ser utilizados por los planificadores urbanos para optimizar las redes de metro y mejorar la experiencia del usuario.

En resumen, este proyecto combina la necesidad personal de una mejor planificación de rutas en el metro con la aplicación de técnicas avanzadas de búsqueda y optimización, proporcionando una solución práctica y eficiente para un problema común en las grandes ciudades.

Objetivos

General:

El objetivo general de este proyecto es desarrollar e implementar un algoritmo de búsqueda que sea capaz de encontrar la ruta más eficiente entre dos estaciones en una red de metro, tomando en consideración múltiples restricciones impuestas por el usuario. Este algoritmo debe ser capaz de manejar tanto las restricciones de minimización de estaciones y transbordos como las restricciones de inclusión y exclusión de estaciones específicas.

Específicos:

1. **Minimización del Número de Estaciones:**
    - Diseñar un algoritmo que identifique la ruta más corta en términos de la cantidad de estaciones a recorrer desde una estación inicial hasta una estación final especificada. Este objetivo busca optimizar el tiempo de viaje del usuario al reducir la distancia total recorrida en la red de metro.
2. **Minimización del Número de Transbordos:**
    - Asegurar que el algoritmo minimice el número de transbordos necesarios durante el trayecto. Dado que los transbordos pueden aumentar el tiempo de viaje y la incomodidad para el usuario, este objetivo es crucial para mejorar la eficiencia y la comodidad del viaje.
3. **Implementación de Restricciones de Estaciones:**
    - Incorporar en el algoritmo la capacidad de evitar ciertas estaciones especificadas por el usuario. Estas restricciones pueden ser debidas a obras, estaciones congestionadas, o preferencias personales del usuario. El algoritmo debe respetar estas restricciones y encontrar rutas que las eviten.
4. **Incorporación de Estaciones Obligatorias:**
    - Desarrollar la funcionalidad para incluir estaciones específicas que deben ser parte de la ruta. Esto puede ser necesario por diversas razones, como puntos de interés turísticos, estaciones de intercambio importantes, o necesidades del usuario. El algoritmo debe asegurar que la ruta pase por estas estaciones obligatorias.

5. **Priorización de Restricciones:**
    - Permitir al usuario asignar prioridades a las restricciones de estaciones obligatorias y prohibidas. Algunas restricciones pueden ser más críticas que otras, y el algoritmo debe poder manejar estas prioridades adecuadamente para generar la ruta más adecuada según las preferencias del usuario.

6. **Flexibilidad en la Entrada de Datos:**
    - Diseñar el algoritmo de tal manera que pueda procesar datos sobre las líneas y estaciones del metro de forma flexible. El usuario debe poder introducir la información de la red de metro en un formato específico y el algoritmo debe ser capaz de interpretar estos datos correctamente para realizar la búsqueda de la ruta óptima.
7. **Validación y Verificación del Algoritmo:**
    - Probar y validar el algoritmo con diversos escenarios de prueba que representen diferentes configuraciones de la red de metro y restricciones. Este objetivo es fundamental para garantizar la robustez y fiabilidad del algoritmo en situaciones reales.
8. **Documentación y Soporte:**
    - Desarrollar una documentación detallada que explique el uso del algoritmo, la estructura de los datos de entrada, y ejemplos de uso. Además, proporcionar soporte para la resolución de problemas y la implementación del algoritmo en diferentes entornos.
9. **Optimización de la Heurística:**
    - Mejorar la función heurística utilizada por el algoritmo para asegurar que no solo sea eficiente en encontrar la ruta más corta, sino también en términos de tiempo computacional. La optimización de la heurística es esencial para manejar grandes redes de metro sin comprometer el rendimiento.

Metodología

La metodología seguida en este proyecto se divide en varias etapas clave, cada una de las cuales es fundamental para el desarrollo, implementación y validación del algoritmo de búsqueda para rutas óptimas en una red de metro. A continuación, se describen en detalle las diferentes etapas y grandes pasos del proyecto.

1. Definición y Modelado de la Red de Metro

Objetivo: Crear una representación precisa y manejable de la red de metro, incluyendo las estaciones y las conexiones entre ellas.

Pasos:

- **Recolección de Datos:**
  - Recolectar manualmente datos sobre las estaciones y líneas del metro desde páginas oficiales del gobierno y otras fuentes confiables.
  - Asegurarse de que los datos recolectados estén actualizados y sean precisos.
- **Estructuración de Datos:**
  - Definir las líneas del metro como listas de estaciones. Por ejemplo:

$$linea_1 = [\text{observatorio}, \text{tacubaya}, \text{juanacatlan}, \text{chapultepec}, ...]$$

  - Crear un diccionario para representar las conexiones entre estaciones que permiten transbordos. Por ejemplo:

$$conexiones = \{\text{tacubaya}: [1, 7, 9], \text{balderas}: [1, 3], ...\}$$

2. Diseño del Algoritmo de Búsqueda

Objetivo: Desarrollar un algoritmo que pueda encontrar la ruta más eficiente entre dos estaciones, considerando las restricciones especificadas.

Pasos:

- **Selección del Algoritmo:**
  - Optar por el algoritmo de búsqueda A* debido a su eficacia en la búsqueda de rutas óptimas en grafos y su capacidad para incorporar una función heurística.
- **Definición de la Clase Nodo:**
  - Crear una clase Nodo que represente un estado en el espacio de búsqueda, incluyendo la estación actual, el nodo padre, el costo acumulado y la heurística.

$$class\ Nodo:$$

$$¿_{¿}\ init\ _{¿}¿$$

$$self\ .estado = estado$$

$$self\ .padre = padre$$

$$self\ .costo = costo$$

$$self\ .heuristica = heurística$$

Implementación de la Función Heurística:

- Definir una heurística basada en la distancia de Manhattan para estimar el costo restante desde una estación actual hasta la estación objetivo.

$$¿\ heuristica\left(estacion_{actual}, estacion_{final}\right):$$

$$return\left|\left(estacion_{actual\ .linea} - estacion_{final\ .linea}\right)\right| + \left|\left(estacion_{actual\ .estacion} - estacion_{final\ .estacion}\right)\right|$$

3. Integración de Restricciones

Objetivo: Incorporar restricciones de estaciones obligatorias y prohibidas en el algoritmo de búsqueda.

Pasos:

- **Manejo de Restricciones:**
  - Modificar el algoritmo A* para que pueda manejar listas de estaciones que se deben evitar y listas de estaciones por las que se debe pasar.
  - Permitir al usuario asignar prioridades a estas restricciones para que el algoritmo pueda ponderar su importancia durante la búsqueda.

- **Adaptación del Algoritmo:**
  - Ajustar el algoritmo para evitar estaciones prohibidas y asegurar que se pase por estaciones obligadas.

$$if\ nombre_{¿} \in est_{prohibidas}:$$

$$continue$$

$$if\ not\ all\left(est \in visitados\ for\ est \in est_{obligadas}\right):$$

$$continue$$

4. Implementación en Python

Objetivo: Programar el algoritmo y las funciones auxiliares en Python para su ejecución y pruebas.

Pasos:

- **Programación del Algoritmo:**
  - Codificar el algoritmo A* en Python, asegurando que maneje adecuadamente las restricciones y utilice la heurística definida.
  - Desarrollar funciones auxiliares para obtener el nombre de una estación y las conexiones posibles desde una estación dada.

```
def obtener_nombre_estacion(tgen, lineas):

    # Código para obtener el nombre de la estación
```

5. Pruebas y Validación

Objetivo: Verificar que el algoritmo funciona correctamente y cumple con los objetivos planteados.

Pasos:

- **Diseño de Casos de Prueba:**
  - Crear varios casos de prueba que representen diferentes configuraciones de la red de metro y distintas combinaciones de restricciones.
  - Asegurar que los casos de prueba cubran escenarios comunes y extremos para validar la robustez del algoritmo.

- **Ejecución de Pruebas:**
  - Ejecutar el algoritmo con los casos de prueba definidos, verificando que los resultados sean correctos y que las restricciones se respeten.
  - Analizar los resultados para identificar posibles mejoras o ajustes necesarios en el algoritmo.

6. Documentación y Presentación

Objetivo: Documentar el proyecto de manera detallada y preparar la presentación final.

Pasos:

- **Desarrollo de la Documentación:**
  - Redactar una documentación completa que incluya la descripción del problema, la metodología seguida, el diseño del algoritmo, y ejemplos de uso.

Marco Teórico

El marco teórico de este proyecto se basa en el uso de algoritmos de búsqueda y técnicas de optimización aplicadas a grafos. En particular, el algoritmo A* (A estrella) es el enfoque central debido a su eficiencia y eficacia en la búsqueda de rutas óptimas en grafos. Además, se utilizan estructuras de datos y conceptos de teoría de grafos para modelar y resolver el problema planteado.

Algoritmo A (A Estrella)*

El algoritmo A* es una técnica de búsqueda informada utilizada ampliamente en problemas de búsqueda de rutas.

- *Funcionamiento del Algoritmo A:**
  - A* utiliza dos funciones de costo:
    - g(n): El costo real desde el nodo inicial hasta el nodo n.
    - h(n): La estimación del costo desde el nodo n hasta el nodo objetivo, conocida como heurística.
  - La función de evaluación f(n) se define como:

    $$f(n)=g(n)+h(n)$$

    donde f(n) representa el costo total estimado del camino que pasa por el nodo n.

  - En cada iteración, A* selecciona el nodo con el valor f(n) más bajo para su expansión, asegurando que se exploren primero los caminos más prometedores.

- **Heurística Utilizada:**
  - En este proyecto, se utiliza la distancia de Manhattan como heurística. Esta se define como la suma de las diferencias absolutas en las coordenadas (en este caso, las líneas y posiciones de las estaciones) entre el nodo actual y el nodo objetivo.

    $$h(n)=¿x1-x2\vee+¿y1-y2\vee¿$$

Teoría de Grafos

El problema de encontrar la ruta óptima en una red de metro se modela como un grafo, donde:

- **Nodos:** Representan las estaciones del metro.
- **Aristas:** Representan las conexiones entre estaciones, que pueden ser directas (mismo línea) o indirectas (transbordos).
- **Modelado del Grafo:**
    - Cada línea del metro se representa como una lista de estaciones, y las conexiones entre estaciones se modelan utilizando un diccionario de listas de adyacencia.
    - Las conexiones permiten definir dónde se pueden realizar transbordos, indicando las líneas a las que se puede cambiar en cada estación.

- **Conexiones y Transbordos:**
    - Las conexiones directas entre estaciones se manejan avanzando o retrocediendo en la misma línea.
    - Los transbordos se manejan identificando las estaciones comunes entre diferentes líneas y añadiendo costos adicionales por cambio de línea.

Estructuras de Datos

- **Arrays de NumPy:** Se utilizan para representar las líneas del metro.
- **Diccionarios:** Se utilizan para representar las conexiones y los transbordos posibles entre las estaciones

Restricciones y Priorización

La inclusión de restricciones en el problema añade una capa de complejidad adicional:

- **Estaciones Prohibidas:** Estaciones que deben ser evitadas durante el trayecto. El algoritmo debe verificar constantemente que la estación actual no esté en la lista de prohibidas.
- **Estaciones Obligatorias:** Estaciones que deben ser incluidas en el trayecto. El algoritmo debe asegurarse de que todas las

estaciones obligatorias sean visitadas antes de llegar a la estación final.

- **Priorización de Restricciones:** El algoritmo permite asignar prioridades a las restricciones, ponderando su importancia para generar una ruta que satisfaga mejor las necesidades del usuario.

Propuesta

Descripción del Modelo

El modelo propuesto utiliza el algoritmo de búsqueda A* para encontrar la ruta más eficiente entre dos estaciones en una red de metro. Este algoritmo es adecuado debido a su capacidad para incorporar una función heurística que guía la búsqueda hacia la solución óptima de manera eficiente.

Variables del Modelo

Las principales variables del modelo son las siguientes:

1. **Estación:**
    - Representada como un objeto con atributos de línea y posición.

$$class\ Estacion:$$

$$¿¿init¿(self, linea, posicion):$$

$$self.linea = linea$$

$$self.posicion = posicion$$

2. **Nodo:**
    - Representa un estado en el espacio de búsqueda, con atributos para el estado actual, el nodo padre, el costo acumulado, la heurística y el conjunto de estaciones visitadas.

$$class\ Nodo:$$

$$¿¿init¿¿$$

$$self.estado = estado$$

$$self.padre = padre$$

$$self.costo = costo$$

$$self.heuristica = heuristica$$

$$self.visitados = visitados\ if\ visitados\ is\ not\ ¿$$

3. **Funciones Principales:**
   - **obtener_nombre_estacion:** Devuelve el nombre de la estación dada una línea y una posición.

$$¿obtener_{¿}(estacion, lineas):$$
$$return\ lineas[estacion.linea][estacion.posicion]$$

   - **obtener_conexiones:** Obtiene las posibles líneas a las que se puede cambiar desde una estación.

$$¿obtener_{conexiones}(estacion, linea_{actual}, conexiones):$$
$$if\ estacion \in conexiones[linea_{actual}]:$$
$$return\ conexiones[linea_{actual}][estacion]$$
$$return[\,]$$

4. **Estructuras de Datos:**
   - **Listas de líneas:** Cada línea del metro se representa como una lista de estaciones.

$$linea_1 = [\text{observatorio}, \text{tacubaya}, \text{juanacatlan}, \text{chapultepec}, ...]$$

   - **Diccionario de conexiones:** Define las conexiones entre estaciones para transbordos.

$$conexiones = \{$$
$$\text{tacubaya}: [1, 7, 9],$$
$$\text{balderas}: [1, 3],$$
$$...$$
$$\}$$

Implementación del Algoritmo

Proceso del Algoritmo:

1. **Inicialización:**
    - Crear el nodo inicial con la estación de partida, costo inicial de 0 y la heurística calculada hacia la estación objetivo.
    - Insertar el nodo inicial en una cola de prioridad (min-heap) basada en el valor f(n).

2. **Búsqueda:**
    - En cada iteración, extraer el nodo con el menor valor f(n) de la cola de prioridad.
    - Si el nodo extraído corresponde a la estación objetivo y todas las estaciones obligatorias han sido visitadas, se retorna la solución.
    - Expandir el nodo, generando nodos sucesores para estaciones adyacentes y conexiones posibles para transbordos, actualizando costos y heurísticas.
    - Insertar los nodos sucesores en la cola de prioridad si no han sido explorados anteriormente.

3. **Manejo de Restricciones:**
    - Verificar en cada expansión de nodo si la estación actual está en la lista de estaciones prohibidas, en cuyo caso se descarta.
    - Asegurar que las estaciones obligatorias sean visitadas antes de alcanzar la estación final.

4. **Construcción de la Ruta:**
    - Una vez que se encuentra la estación objetivo, reconstruir la ruta desde el nodo final hacia el nodo inicial siguiendo los nodos padre.

Código de Implementación:

$$import\ numpy\ as\ np$$

$$import\ heapq$$

$$¿\ Definir\ clases\ y\ funciones\ como\ se\ describe\ en\ el\ modelo$$

$$¿\ Implementación\ del\ algoritmo\ A*¿$$

$$¿\ a_{estrella}$$

$$(estacion_{inicial}, estacion_{final}, conexiones, lineas, est_{prohibidas}, est_{obligadas}):$$

$$frontera = [\,]$$

$$heapq.heappush$$

$$¿$$

$$(estacion_{inicial}, estacion_{final}), \{obtener_{¿}(estacion_{inicial}, lineas)\}¿¿$$

$$explorados = set()$$

$$while\ frontera:$$

$$nodo_{actual} = heapq.heappop(frontera)$$

$$estado_{actual} = nodo_{actual.estado}$$

$$visitados = nodo_{actual.visitados}$$

$$if\ (estado_{actual.linea}, estado_{actual.posicion}) \in explorados:$$

$$continue$$

$$if\ estado_{actual.linea} = ¿\ estacion_{final.linea} \wedge estado_{actual.posicion} = ¿\ estacion_{final.posicion}:$$

$$if\ all(est \in visitados\ for\ est \in est_{obligadas}):$$

$$return\ obtener_{camino}(nodo_{actual})$$

$$explorados.add((estado_{actual.linea}, estado_{actual.posicion}))$$

$$linea_{actual} = estado_{actual.linea}$$

$$estacion_{pos} = estado_{actual.posicion}$$

$$nombre_{¿} = obtener_{¿}(estado_{actual}, lineas)$$

$$if\ nombre_{¿} \in est_{prohibidas}:$$

$$continue$$

$$¿\ Expandir\ nodos\ sucesores(avanzar, retroceder\ y\ transbordar)$$

$$¿\ Agregar\ nodos\ sucesores\ a\ la\ frontera$$

$$return\ ¿$$

$$¿\ Ejecución\ del\ algoritmo\ con\ datos\ específicos$$

Resultados

Hacemos pruebas poniendo la estación inicial y estación inicial en nuestro programa:

1er Caso

```
# Definir estaciones inicial y final
estacion_inicial = TGen(1, 0)  # Observatorio en línea 1
estacion_final = TGen(3, 3)  # La Raza en línea 3
```

✓ 0.0s

```
Línea: 1, Estación: 0 -> Nombre: observatorio
Línea: 1, Estación: 1 -> Nombre: tacubaya
Línea: 1, Estación: 2 -> Nombre: juanacatlan
Línea: 1, Estación: 3 -> Nombre: chapultepec
Línea: 1, Estación: 4 -> Nombre: sevilla
Línea: 1, Estación: 5 -> Nombre: insurgentes
Línea: 1, Estación: 6 -> Nombre: cuauhtemoc
Línea: 1, Estación: 7 -> Nombre: balderas
Línea: 3, Estación: 8 -> Nombre: balderas
Línea: 3, Estación: 7 -> Nombre: juarez
Línea: 3, Estación: 6 -> Nombre: hidalgo
Línea: 3, Estación: 5 -> Nombre: guerrero
Línea: 3, Estación: 4 -> Nombre: tlatelolco
Línea: 3, Estación: 3 -> Nombre: la_raza
```

2do Caso

```
# Definir estaciones inicial y final
estacion_inicial = TGen(4, 5) # Morelos
estacion_final = TGen(12, 9) # San Andrés Tomatlán
```

✓ 0.0s

```
Línea: 4, Estación: 5 -> Nombre: morelos
Línea: 4, Estación: 6 -> Nombre: candelaria
Línea: 4, Estación: 7 -> Nombre: fray_servando
Línea: 4, Estación: 8 -> Nombre: jamaica
Línea: 4, Estación: 9 -> Nombre: santa_anita
Línea: 8, Estación: 8 -> Nombre: santa_anita
Línea: 8, Estación: 9 -> Nombre: coyuya
Línea: 8, Estación: 10 -> Nombre: iztacalco
Línea: 8, Estación: 11 -> Nombre: apatlaco
Línea: 8, Estación: 12 -> Nombre: aculco
Línea: 8, Estación: 13 -> Nombre: escuadron_201
Línea: 8, Estación: 14 -> Nombre: atlalilco
Línea: 12, Estación: 11 -> Nombre: atlalilco
Línea: 12, Estación: 10 -> Nombre: culhuacan
Línea: 12, Estación: 9 -> Nombre: san_andres_tomatlan
```

3er Caso

```
# Definir estaciones inicial y final
estacion_inicial = TGen(1, 0) # Observatorio
estacion_final = TGen(12, 19) # Mixcoac
```

✓ 0.0s

```
# Definir restricciones y estaciones obligatorias
est_prohibidas = ["tacubaya"]
est_obligadas = ["balderas"]
```

✓ 0.0s

```
Línea: 5, Estación: 0 -> Nombre: politecnico
Línea: 5, Estación: 1 -> Nombre: instituto_del_petroleo
Línea: 5, Estación: 2 -> Nombre: autobuses_del_norte
Línea: 5, Estación: 3 -> Nombre: la_raza
Línea: 3, Estación: 3 -> Nombre: la_raza
Línea: 3, Estación: 4 -> Nombre: tlatelolco
Línea: 3, Estación: 5 -> Nombre: guerrero
Línea: 3, Estación: 6 -> Nombre: hidalgo
Línea: 3, Estación: 7 -> Nombre: juarez
Línea: 3, Estación: 8 -> Nombre: balderas
Línea: 3, Estación: 9 -> Nombre: ninos_heroes
Línea: 3, Estación: 10 -> Nombre: hospital_general
Línea: 3, Estación: 11 -> Nombre: centro_medico
Línea: 3, Estación: 12 -> Nombre: etiopia_plaza_de_la_transparencia
Línea: 3, Estación: 13 -> Nombre: eugenia
Línea: 3, Estación: 14 -> Nombre: division_del_norte
Línea: 3, Estación: 15 -> Nombre: zapata
Línea: 12, Estación: 16 -> Nombre: zapata
Línea: 12, Estación: 17 -> Nombre: hospital_20_de_noviembre
Línea: 12, Estación: 18 -> Nombre: insurgentes_sur
Línea: 12, Estación: 19 -> Nombre: mixcoac
```

Conclusión

El proyecto de desarrollo de un algoritmo de búsqueda para rutas óptimas en una red de metro ha demostrado ser un esfuerzo exitoso y valioso, cumpliendo con los objetivos establecidos y aportando una solución eficaz a un problema común en el transporte urbano. A lo largo del proyecto, se implementó el algoritmo A*, reconocido por su eficiencia y precisión en la búsqueda de caminos óptimos en grafos, adaptándolo para manejar diversas restricciones que los usuarios pueden imponer, tales como evitar ciertas estaciones o incluir otras como obligatorias. Este enfoque ha permitido no solo encontrar la ruta más corta en términos de número de estaciones y transbordos, sino también respetar las preferencias específicas del usuario, lo que añade un valor significativo a la herramienta desarrollada.

Una de las mayores fortalezas del algoritmo es su capacidad para manejar restricciones de manera flexible y eficiente, asignando prioridades a estas restricciones según las necesidades del usuario. Esto se logró mediante la integración de una función heurística basada en la distancia de Manhattan, que guió la búsqueda de manera efectiva y garantizó que el algoritmo fuera capaz de encontrar soluciones óptimas o cercanas a la óptima en la mayoría de los casos. La flexibilidad del modelo también se refleja en su diseño, que permite su adaptación a diversas redes de metro simplemente ajustando los datos de entrada.

Bibliografía

- Metro CDMX. (2022, 5 abril). *Metro de CDMX - Mapas, líneas, estaciones y horarios*. https://metro-mx.com/
- Cdmx, M. (s. f.). *Mapa de la red*. Metro CDMX. https://metro.cdmx.gob.mx/la-red/mapa-de-la-red
- Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th Edition).