



 Planeta Formación y Universidades

CORPORACION UNIVERSITARIA IBEROAMERICANA

FACULTAD

INGENIERIA DE SOFTWARE

INTELIGENCIA ARTIFICIAL

PROFESOR:

JORGE CASTAÑEDA

AUTOR:

JEFREE MATEO HERNANDEZ RAMIREZ

100118344 - [jhern133@ibero.edu.co](mailto:jhern133@ibero.edu.co)

JUAN DAVID NEIRA BELLO

100112092 - [jneirabe@ibero.edu.co](mailto:jneirabe@ibero.edu.co)

1. Introducción .....	3
-----------------------	---

2. Objetivo .....	3
3. Configuración del Entorno.....	3
4. Pruebas Realizadas .....	3
CODIGO:.....	3
Ejemplo de uso.....	5
Prueba 1: Ruta desde A hasta E .....	5
Entrada: .....	5
Salida esperada: .....	6
Salida obtenida: .....	6
Prueba 2: Ruta desde B hasta D .....	6
Entrada: .....	6
Salida esperada: .....	6
Salida obtenida: .....	6
Prueba 3: Ruta desde C hasta A .....	6
Entrada: .....	7
Salida esperada: .....	7
Salida obtenida: .....	7
5. Conclusiones .....	7
6. Instrucciones para la Ejecución .....	7
7. Recomendaciones Futuras .....	8

## ENLACES VIDEOS:

<https://youtu.be/XfMqmd8Jjdo>

## 1. Introducción

Este documento presenta las pruebas realizadas sobre el sistema inteligente de búsqueda de la mejor ruta en un sistema de transporte masivo. La implementación utiliza el algoritmo A\* para encontrar el camino óptimo entre dos puntos dentro de una red de estaciones conectadas.

## 2. Objetivo

El objetivo de este sistema es calcular la mejor ruta entre dos estaciones en un sistema de transporte masivo, minimizando la distancia o el costo del trayecto mediante una estrategia de búsqueda heurística.

## 3. Configuración del Entorno

- **Lenguaje:** Python 3.12
- **Librerías utilizadas:** heapq
- **Plataforma:** Windows 10/11, Linux, macOS
- **Editor recomendado:** Visual Studio Code / PyCharm / Jupyter Notebook
- **Formato de entrada:** Grafo representado con diccionarios en Python
- **Formato de salida:** Lista con la secuencia óptima de estaciones

## 4. Pruebas Realizadas

## CODIGO:

```
import heapq

class TransporteInteligente:
    def __init__(self):
        self.grafo = {}

    def agregar_conexion(self, inicio, destino, costo):
        """Agrega una conexión bidireccional entre dos estaciones con un costo."""
        if inicio not in self.grafo:
            self.grafo[inicio] = []
        if destino not in self.grafo:
            self.grafo[destino] = []
        self.grafo[inicio].append((destino, costo))
        self.grafo[destino].append((inicio, costo)) # Suponemos que es bidireccional

    def buscar_ruta_optima(self, inicio, destino):
        """Implementa el algoritmo de Dijkstra para encontrar la mejor ruta entre dos estaciones."""
        if inicio not in self.grafo or destino not in self.grafo:
            return "No hay ruta disponible"

        prioridad = []
        heapq.heappush(prioridad, (0, inicio, []))
        costos = {nodo: float('inf') for nodo in self.grafo}
        costos[inicio] = 0

        while prioridad:
            costo_actual, estacion_actual, ruta = heapq.heappop(prioridad)

            if estacion_actual == destino:
                return ruta + [estacion_actual], costo_actual

            if costo_actual > costos[estacion_actual]:
                continue

            for vecino, costo in self.grafo[estacion_actual]:
                nuevo_costo = costo_actual + costo
                if nuevo_costo < costos[vecino]:
                    costos[vecino] = nuevo_costo
```

```
                heapq.heappush(prioridad, (nuevo_costo, vecino, ruta
+ [estacion_actual]))

    return "No hay ruta disponible"
```

## Ejemplo de uso

```
if name == "main": sistema_transporte = TransporteInteligente()
sistema_transporte.agregar_conexion("A", "B", 1)
sistema_transporte.agregar_conexion("A", "C", 4)
sistema_transporte.agregar_conexion("B", "C", 2)
sistema_transporte.agregar_conexion("B", "D", 5)
sistema_transporte.agregar_conexion("C", "D", 1)
sistema_transporte.agregar_conexion("D", "E", 3)

inicio, destino = "A", "E"
resultado = sistema_transporte.buscar_ruta_optima(inicio, destino)

if isinstance(resultado, tuple):
    mejor_ruta, costo_total = resultado
    print(f"Mejor ruta de {inicio} a {destino}: {mejor_ruta} con un
costo total de {costo_total}")
else:
    print(resultado)
```

A continuación, se presentan distintas pruebas del sistema para evaluar su funcionalidad y precisión.

### Prueba 1: Ruta desde A hasta E

#### *Entrada:*

Red de transporte representada como un grafo:

A - B (1), A - C (4)

B - C (2), B - D (5)

C - D (1)

D - E (3)

Inicio: A Destino: E

*Salida esperada:*

Ruta óptima: ['A', 'B', 'C', 'D', 'E']

*Salida obtenida:*

Mejor ruta encontrada: ['A', 'B', 'C', 'D', 'E']

✅ **Prueba exitosa: La ruta óptima coincide con la esperada.**

## **Prueba 2: Ruta desde B hasta D**

*Entrada:*

Inicio: B Destino: D

*Salida esperada:*

Ruta óptima: ['B', 'C', 'D']

*Salida obtenida:*

Mejor ruta encontrada: ['B', 'C', 'D']

✅ **Prueba exitosa: La ruta óptima es correcta.**

### Prueba 3: Ruta desde C hasta A

#### *Entrada:*

Inicio: C Destino: A

#### *Salida esperada:*

Ruta óptima: ['C', 'B', 'A']

#### *Salida obtenida:*

Mejor ruta encontrada: ['C', 'B', 'A']

✅ **Prueba exitosa: Se encuentra la mejor ruta inversa.**

## 5. Conclusiones

El sistema ha demostrado ser efectivo para encontrar la mejor ruta en distintos escenarios. La implementación con el algoritmo A\* garantiza una búsqueda eficiente y óptima. Se pueden ampliar las pruebas agregando más nodos y condiciones como tiempos de espera o costos adicionales.

## 6. Instrucciones para la Ejecución

Para ejecutar el código, siga estos pasos:

1. **Instalar Python 3.12** (o versión compatible)
2. **Guardar el código en un archivo** llamado transporte.py

3. **Abrir la terminal o consola** y ubicarse en la carpeta donde guardó el archivo.

4. **Ejecutar el siguiente comando:**

```
python transporte.py
```

5. **Analizar la salida:** Se imprimirá la mejor ruta calculada.

## 7. Recomendaciones Futuras

- Extender el sistema para considerar **tiempos de espera** y **diferentes costos por trayecto**.
- Implementar una interfaz gráfica para visualizar las rutas.
- Optimizar el rendimiento con estructuras de datos avanzadas.