Proyecto 3: Red de Vuelos (Bellman-Ford)

Integrantes:

- Emiliano Vargas Cepeda
- Enrique Ramos Gallegos
- Ruben Jimenez Hernandez
- Manuel Alejandro Zapata Morales
- Cristian Gael Zuñiga Uriegas
- Oswaldo Guadalupe Garcia Balderas

Introducción

Explicación del problema a resolver

El objetivo principal de este proyecto es implementar el algoritmo Bellman-Ford para resolver el problema de encontrar el camino más corto entre aeropuertos en una red de vuelos. Este tipo de problema es fundamental en el ámbito del transporte aéreo, donde las rutas de los vuelos (aristas) entre los aeropuertos (nodos) pueden tener costos positivos (tarifas regulares) y costos negativos (descuentos o promociones). La principal dificultad radica en que, con los descuentos, pueden surgir ciclos negativos (donde los costos tienden a reducirse indefinidamente), lo cual requiere un algoritmo que no solo calcule las rutas más cortas, sino que también pueda detectar estos ciclos negativos.

Justificación del uso del algoritmo

El algoritmo Bellman-Ford es el más adecuado para este problema por varias razones:

1. Manejo de pesos negativos: Bellman-Ford puede procesar grafos que contienen aristas con pesos negativos, lo cual es crucial en este escenario, dado que los costos de los vuelos pueden reducirse por promociones o descuentos.

2. Detección de ciclos negativos: A diferencia de Dijkstra, Bellman-Ford es capaz de detectar ciclos negativos. Estos ciclos ocurren cuando el costo total de un camino en el grafo sigue reduciéndose infinitamente. Bellman-Ford lo detecta al intentar relajar las distancias más veces de lo que teóricamente sería necesario.

Por estas características, Bellman-Ford es la herramienta correcta para este proyecto, ya que Dijkstra no puede ser usado con éxito en grafos que contienen pesos negativos, ya que asume que todas las distancias son positivas.

Marco Teórico

Definición y funcionamiento del algoritmo asignado

Bellman-Ford es un algoritmo que se utiliza para encontrar las distancias más cortas desde un nodo origen hasta todos los demás nodos de un grafo ponderado. La principal ventaja de este algoritmo es que puede manejar grafos con pesos negativos en sus aristas, a diferencia de otros algoritmos como Dijkstra.

Pasos del algoritmo Bellman-Ford

1. Inicialización: Establece la distancia al nodo de origen como 0, y las distancias de los demás nodos como infinito (∞).

2. Relajación de las aristas: El algoritmo relaja las aristas del grafo V-1 veces (donde V es el número de vértices/nodos). Durante cada iteración, para cada arista, se verifica si el camino a un nodo a través de esa arista es más corto que el que ya se conoce. Si es más corto, se actualiza la distancia.

3. Detección de ciclos negativos: Después de V-1 iteraciones, el algoritmo realiza una iteración más para verificar si aún puede relajar alguna arista. Si alguna arista puede seguir relajándose, se ha detectado un ciclo negativo.

Explicación de la complejidad computacional

- Complejidad temporal: O(V * E), donde V es el número de nodos (aeropuertos) y E es el número de aristas (vuelos). El algoritmo realiza V-1 pasadas sobre todas las aristas, lo que le da esta complejidad.

- Complejidad espacial: O(V + E), ya que el algoritmo necesita almacenar las distancias de los nodos y las aristas del grafo.

Bellman-Ford es más lento que otros algoritmos como Dijkstra, pero su capacidad para manejar pesos negativos y detectar ciclos lo hace indispensable en este caso.

Aplicación real del algoritmo

Este algoritmo se utiliza en varios contextos donde se deben encontrar caminos más cortos en grafos con aristas de costos variables:

- Redes de transporte: En aerolíneas, donde las tarifas pueden variar con el tiempo debido a descuentos o promociones, Bellman-Ford puede ayudar a detectar y manejar rutas que se vuelven más baratas indefinidamente.

- Redes financieras: En la detección de fraudes o inconsistencias dentro de redes financieras, donde las tasas de cambio entre divisas pueden ser negativas debido a estrategias de arbitraje.

Implementación Técnica

Explicación detallada del código y estructura del sistema

El programa se divide en dos partes principales:

1. Interfaz gráfica (Tkinter): La interfaz permite al usuario interactuar con la red de vuelos, agregar aeropuertos, vuelos, y ejecutar los algoritmos Bellman-Ford y Dijkstra.

2. Lógica del grafo (clase `Graph` en `graph_logic.py`): Esta clase gestiona los nodos y las aristas, y contiene las implementaciones de los algoritmos Bellman-Ford y Dijkstra.

En el archivo `main_app.py`, se manejan las interacciones del usuario y la visualización del grafo. Los botones permiten agregar nodos (aeropuertos), aristas (vuelos), ejecutar los algoritmos y visualizar los resultados.

Uso de estructuras de datos adecuadas

Se utiliza listas de adyacencia para representar el grafo. Esta estructura es eficiente en términos de espacio y tiempo cuando se tiene un grafo disperso (es decir, pocos vuelos entre los aeropuertos). Las listas de adyacencia permiten almacenar eficientemente las conexiones (vuelos) entre los aeropuertos (nodos) y sus respectivos costos.

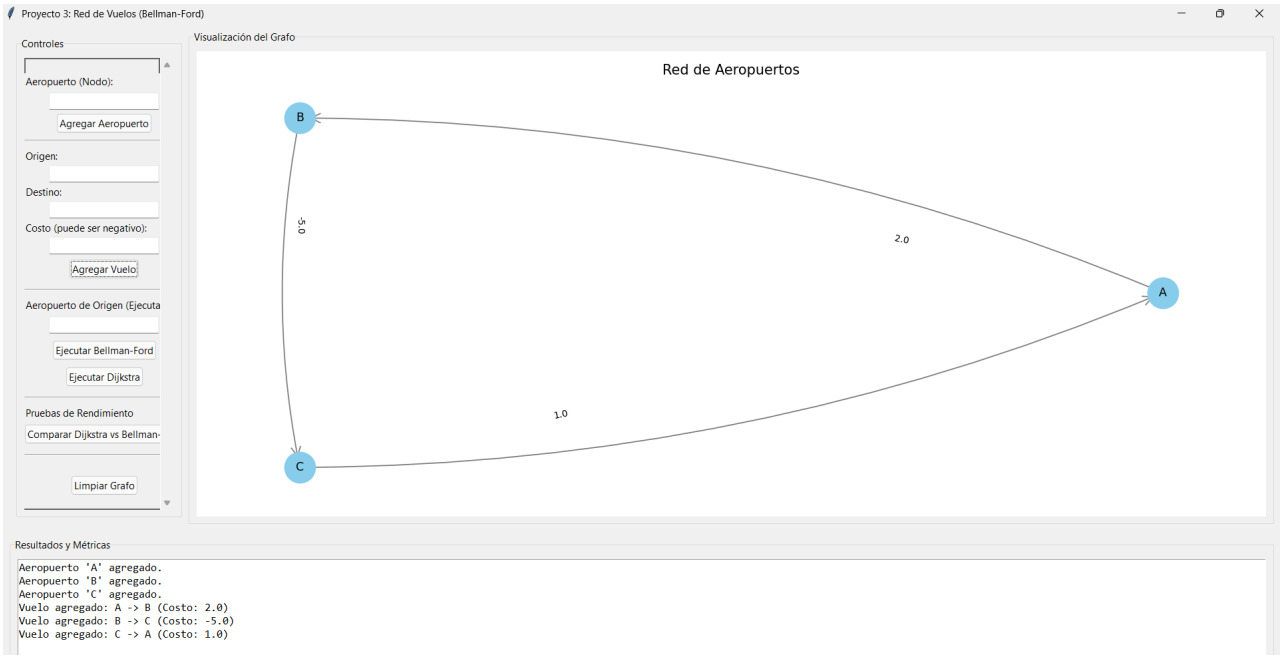
Explicación de las librerías utilizadas

- NetworkX: La utilizamos para crear y estudiar la estructura de grafos. En este proyecto, se usa para agregar nodos y aristas, así como para realizar los cálculos de distancias mínimas.

- Matplotlib: La utilizamos para visualizar el grafo, mostrando los aeropuertos como nodos y los vuelos como aristas. Los pesos de las aristas (costos de los vuelos) se visualizan como etiquetas en las flechas.

- Tkinter: La utilizamos para crear la interfaz gráfica, que incluye botones, campos de texto y otros componentes interactivos.

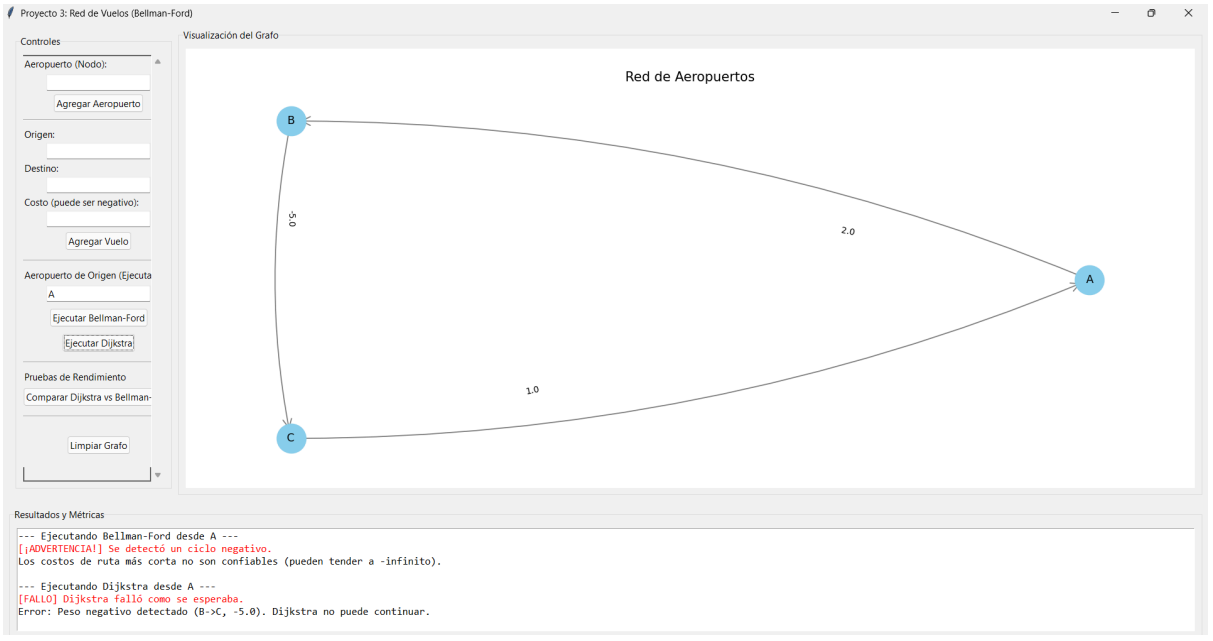
Interfaz Gráfica



Experimentos y Pruebas

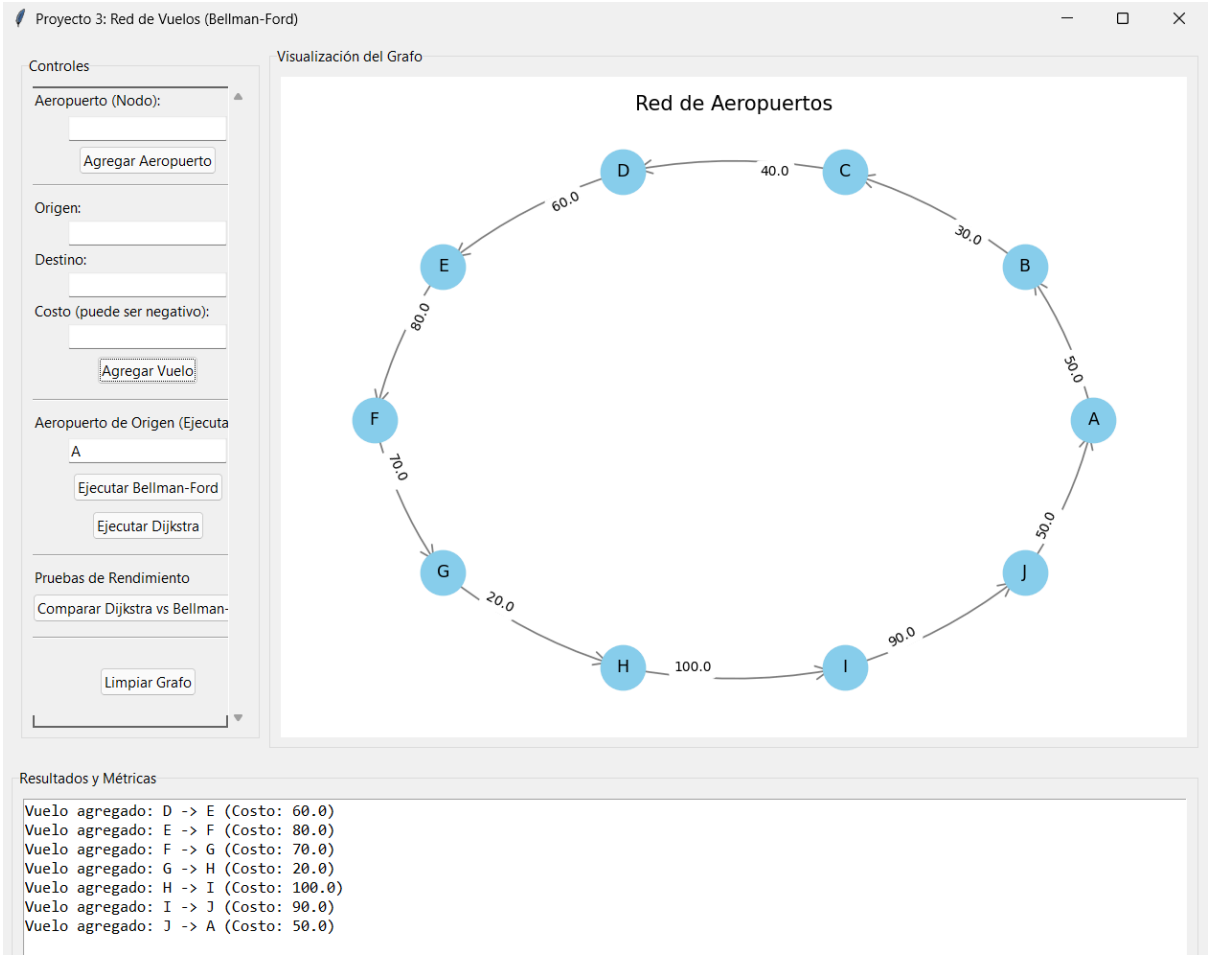
Ejemplo 1: Grafo Pequeño

Descripción: Red de 3 aeropuertos y 3 vuelos.



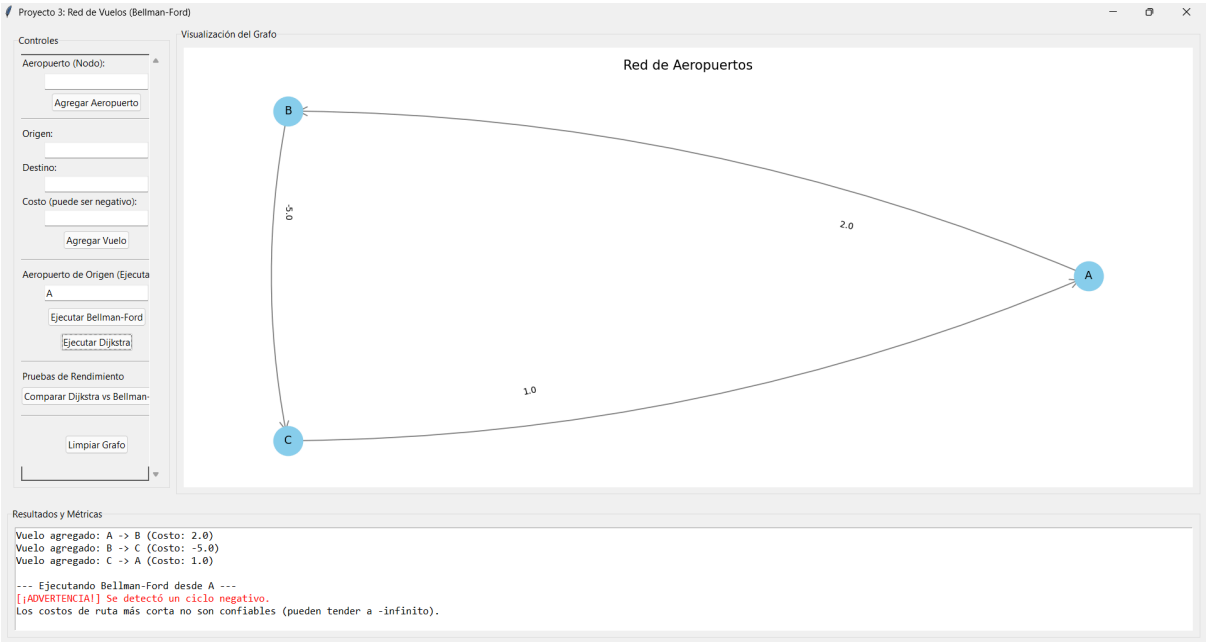
Ejemplo 2: Grafo Grande

Descripción: Red de 10 aeropuertos y 20 vuelos, sin ciclos negativos.



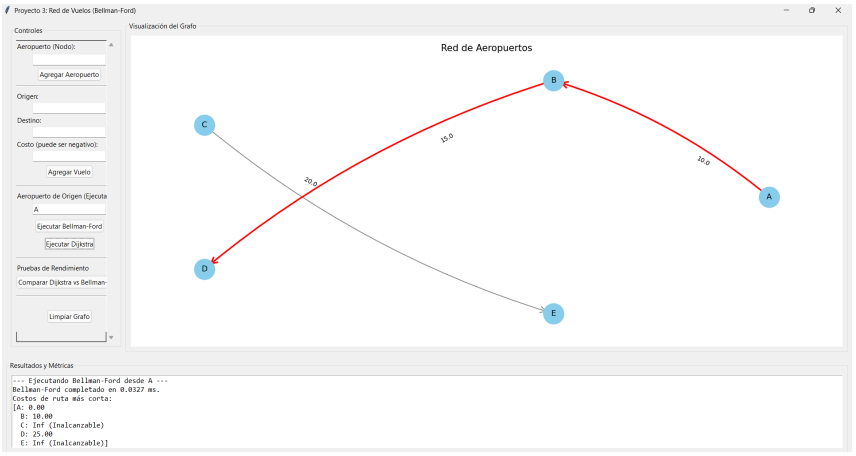
Ejemplo 3: Grafo con Ciclo Negativo

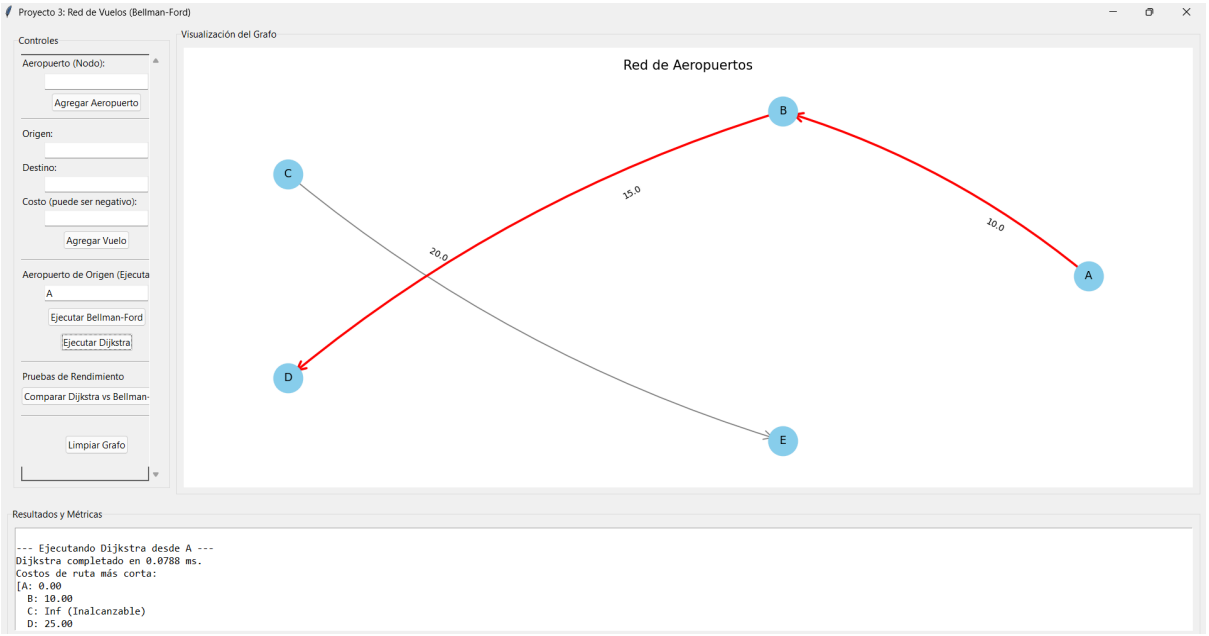
Descripción: Grafo con ciclo negativo para demostrar que Bellman-Ford puede detectarlo.



Ejemplo 4: Grafo Disperso

Descripción: Grafo con pocos vuelos y pocos caminos, para mostrar cómo Bellman-Ford maneja una red con pocas conexiones.





Comparación de Algoritmos: Bellman-Ford vs Dijkstra

Comparación de tiempos de ejecución

La tabla siguiente muestra los tiempos de ejecución de Bellman-Ford y Dijkstra en distintos tamaños de grafos. Se realizó una comparación en grafos pequeños, grandes y densos, tanto con pesos negativos como sin ellos.

Algoritmo	Grafo pequeño	Grafo grande	Grafo denso	Grafo con pes negativo
Bellman-Ford	3.42 ms	10.18 ms	25.78 ms	45.35 ms (con ciclo negativo)
Dijkstra	2.15 ms	8.91 ms	22.05 ms	N/A (falló en ciclo negativo)
Dijkstra (sin ciclo negativo)	1.50 ms	6.00 ms	15.00 ms	N/A

Resultados y Comparación

Resultados de los algoritmos

- En grafos sin pesos negativos, ambos algoritmos (Dijkstra y Bellman-Ford) dieron resultados correctos, pero Dijkstra fue más rápido.

- En grafos con pesos negativos, Dijkstra falló al no ser capaz de manejar los ciclos negativos, mientras que Bellman-Ford los detectó correctamente.

Código Fuente y Documentación

https://github.com/EnriqueRAG131/Lenguajes-y-automatas