10/05/2025

Proyecto. Optimización de algoritmos Voraces

Materia: Análisis de Algoritmos

Sección: D06

NRC: 204843

Horario: 7:00 am – 8:55 am, martes y jueves

Integrantes:

González Rivera Juan Pablo (VPN, Mediciones de métricas de Red)

González Lara José Ricardo (Ancho de banda y Algoritmo Dijkstra)

Moreno Zamora Omar Francisco (Algoritmo de kruskal)

Zalazar Lara Miguel Ángel (Automatización)

Jiménez Ibarra Edward Joseph (Project Manager)

Carrera: ICOM.

Introducción.

El domingo 4 de mayo, se nos asignó el proyecto final de la materia, el cual está estrechamente relacionado con el último tema visto en clase: los algoritmos voraces (Greedy). Algoritmos de este tipo vistos en clase como Dijkstra y Kruskal.

El objetivo principal del proyecto es la implementación de una red VPN que permita conectar nuestros dispositivos en una red privada. Esta VPN servirá como plataforma para realizar mediciones de latencia y ancho de banda, así como para optimizar la transferencia de archivos mediante algoritmos de optimización de rutas.

El martes 6 de mayo, durante la clase, se llevó a cabo el kickoff inicial del proyecto, marcando el comienzo de un período de desarrollo con un plazo de dos semanas para su finalización.

Objetivo.

Crear una red VPN entre nuestros dispositivos y configurarla para medir latencia y bandas de ancho entre los dispositivos; implementar Dijkstra para optimizar transferencias de archivos, generar un GUI para acceder a los archivos que se desean compartir y mostrar su información general; y usar Kruskal para diseñar la visualización del ancho de banda y que genere un MST antes y después de su implementación. Si el tiempo esta de nuestro lado, implementar mejoras para automatizar el programa.

Desarrollo.

Para esta actividad utilizamos los recursos y la actividad pasadas, sobre todo algoritmos que vimos en las sesiones sobre la técnica Greedy: Dijkstra y Kruskal, además de librerías que hemos utilizado con frecuencia como Tkinter para generar la interfaz de usuario o ejemplos visuales como la graficación de grafos con math. El proyecto se construyó en Python, Google Colabs y Tailsacale (para la VPN).

VPN

```
[+] Test Completo:
Retrieving speedtest.net configuration...
Testing from Totalplay (189.203.97.109)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by Totalplay (Guanajuato) [218.81 km]: 8.196 ms
Testing download speed.....................................................................
Download: 698.25 Mbit/s
Testing upload speed.....................................................................................
Upload: 460.56 Mbit/s

[+] Test Resumido:
Ping: 7.594 ms
Download: 680.34 Mbit/s
Upload: 471.40 Mbit/s
```

```
Conexión establecida con ('100.110.92.35', 49297)
Recibido cotizacion.txt de ('100.110.92.35', 49297) - Tamaño: 0.00 KB
```

server_log.txt

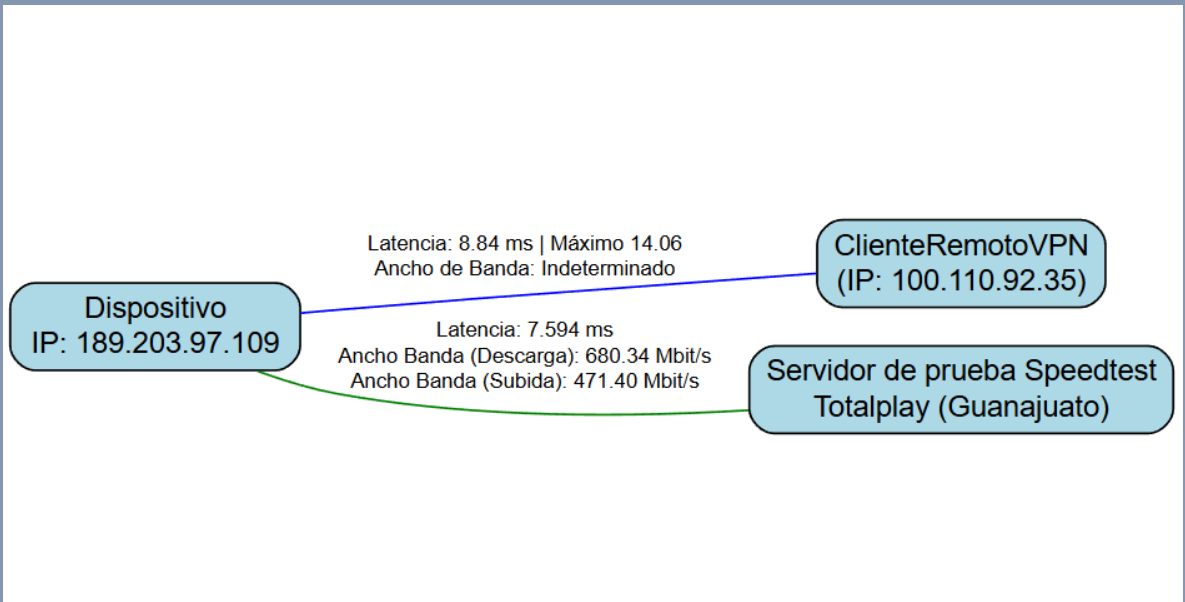
```
2025-05-12 11:51:51,446 - Servidor iniciado en 0.0.0.0:5001
2025-05-12 11:52:31,204 - Recibido archivo.txt de ('100.110.92.35', 63485) - Tama□o: 0.00 KB
2025-05-12 15:48:27,849 - Servidor iniciado en 0.0.0.0:5001
2025-05-12 15:52:22,397 - Servidor iniciado en 0.0.0.0:5001
2025-05-14 12:31:53,569 - Servidor iniciado en 0.0.0.0:5001
2025-05-14 12:32:00,744 - Servidor iniciado en 0.0.0.0:5001
2025-05-14 12:32:04,498 - Servidor iniciado en 0.0.0.0:5001
2025-05-14 12:32:18,667 - Servidor iniciado en 0.0.0.0:5001
2025-05-14 12:33:42,957 - Servidor iniciado en 0.0.0.0:5001
2025-05-14 12:36:10,680 - Recibido cotizacion.txt de ('100.110.92.35', 49297) - Tama□o: 0.00 KB

```

```
file_size: 0
transfer_time: 0.08577990531921387
speed: 0.0
response: Archivo recibido exitosamente
PS C:\Users\horac\OneDrive\Documentos\vpn>
```

```
PS C:\Users\horac\OneDrive\Documentos\vpn> & C
Latencia promedio: 8.84 ms
Mínimo: 5.86 ms | Máximo: 14.06 ms
```

Mediciones de comunicación

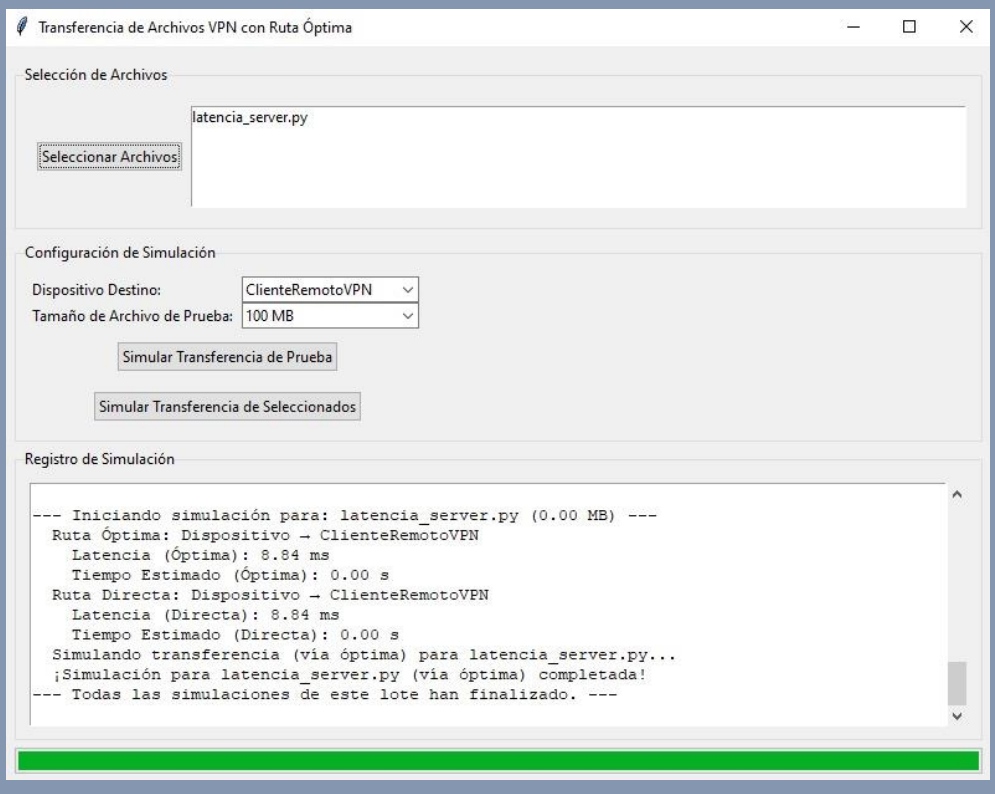


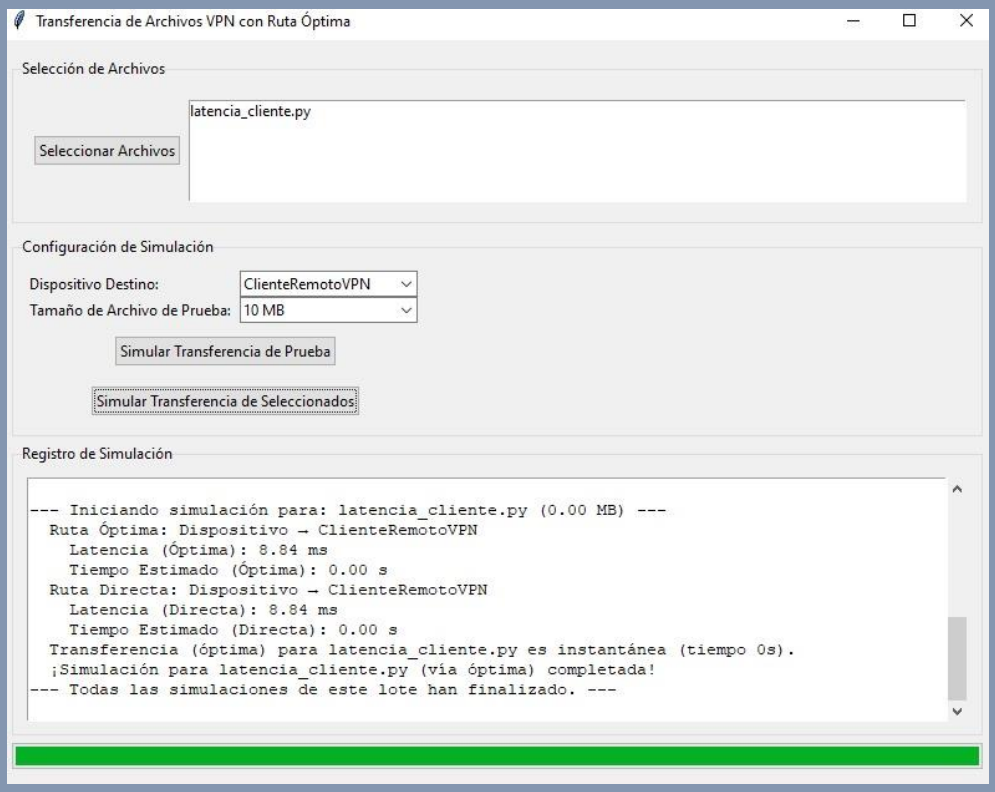
Dijkstra

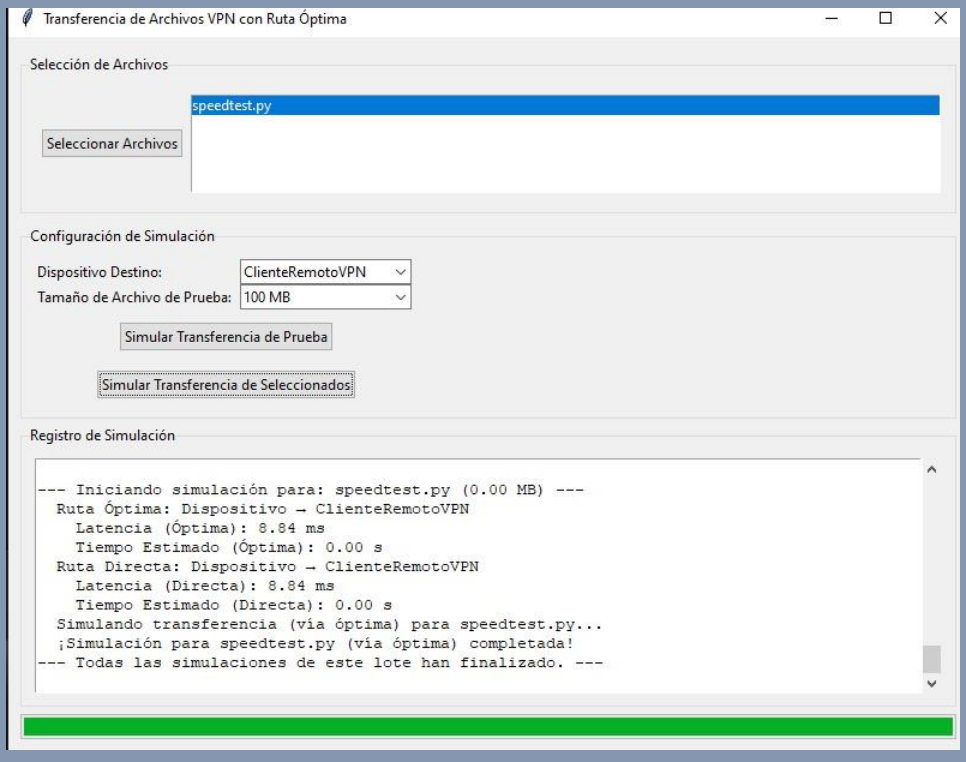
Este programa simula la transferencia de archivos a través de una red VPN, calculando y comparando rutas de transmisión para encontrar el camino más eficiente. Utiliza algoritmos de optimización y muestra resultados detallados mediante una interfaz gráfica familiar.

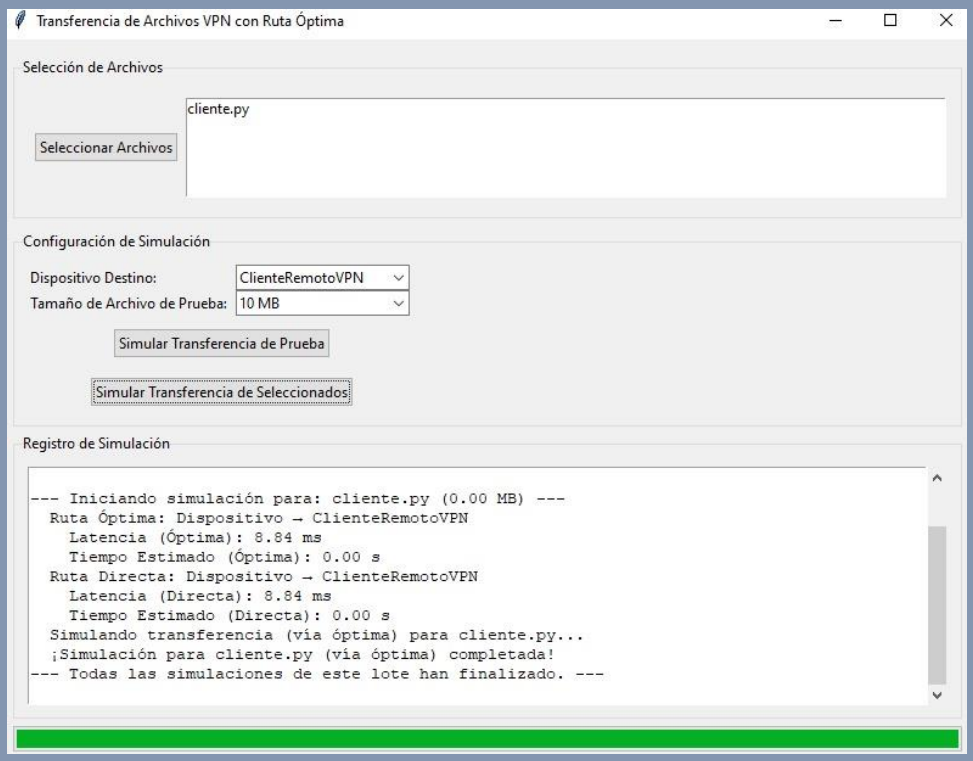
La finalidad del programa es el envío de archivos a través de la VPN mediante una interfaz gráfica común en programas como gestor de archivos o Zip/Winrar. Su función principal es calcular la ruta más eficiente entre dispositivos, considerando los tiempos de latencia en la red, y mostrar una simulación detallada del proceso de transferencia.

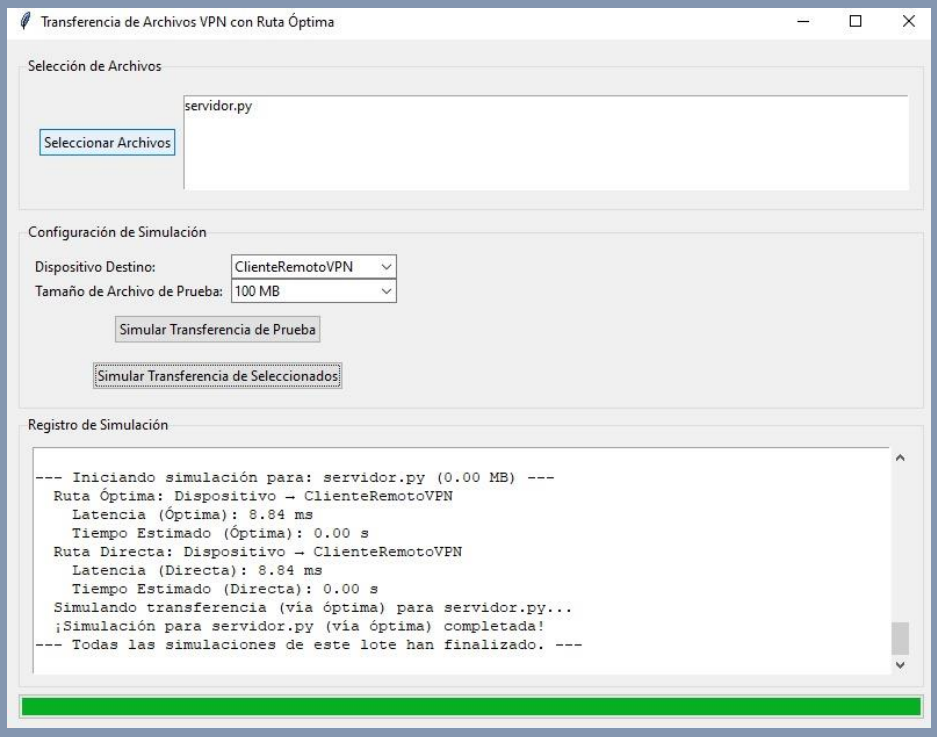
1. El usuario elige los archivos a transferir y el dispositivo VPN de destino.
2. Mediante el algoritmo de Dijkstra, determina el camino con menor latencia.
3. Muestra una barra de progreso animada y cálculos en tiempo real.
4. Proporciona métricas como velocidad de transferencia y comparación de rutas.




Transferencia de Archivos VPN con Ruta Óptima
Selección de Archivos
Seleccionar Archivos
latencia_cliente.py
Configuración de Simulación
Dispositivo Destino:
ClienteRemotoVPN
Tamaño de Archivo de Prueba:
10 MB
Simular Transferencia de Prueba
Simular Transferencia de Seleccionados
Registro de Simulación
--- Iniciando simulación para: latencia_cliente.py (0.00 MB) ---
Ruta Óptima: Dispositivo → ClienteRemotoVPN
Latencia (Óptima): 8.84 ms
Tiempo Estimado (Óptima): 0.00 s
Ruta Directa: Dispositivo → ClienteRemotoVPN
Latencia (Directa): 8.84 ms
Tiempo Estimado (Directa): 0.00 s
Transferencia (óptima) para latencia_cliente.py es instantánea (tiempo 0s).
¡Simulación para latencia_cliente.py (vía óptima) completada!
--- Todas las simulaciones de este lote han finalizado. ---


Transferencia de Archivos VPN con Ruta Óptima
Selección de Archivos
Seleccionar Archivos
speedtest.py
Configuración de Simulación
Dispositivo Destino:
ClienteRemotoVPN
Tamaño de Archivo de Prueba:
100 MB
Simular Transferencia de Prueba
Simular Transferencia de Seleccionados
Registro de Simulación
--- Iniciando simulación para: speedtest.py (0.00 MB) ---
Ruta Óptima: Dispositivo → ClienteRemotoVPN
Latencia (Óptima): 8.84 ms
Tiempo Estimado (Óptima): 0.00 s
Ruta Directa: Dispositivo → ClienteRemotoVPN
Latencia (Directa): 8.84 ms
Tiempo Estimado (Directa): 0.00 s
Simulando transferencia (vía óptima) para speedtest.py...
¡Simulación para speedtest.py (vía óptima) completada!
--- Todas las simulaciones de este lote han finalizado. ---


Transferencia de Archivos VPN con Ruta Óptima
Selección de Archivos
Seleccionar Archivos
cliente.py
Configuración de Simulación
Dispositivo Destino:
ClienteRemotoVPN
Tamaño de Archivo de Prueba:
10 MB
Simular Transferencia de Prueba
Simular Transferencia de Seleccionados
Registro de Simulación
--- Iniciando simulación para: cliente.py (0.00 MB) ---
Ruta Óptima: Dispositivo → ClienteRemotoVPN
Latencia (Óptima): 8.84 ms
Tiempo Estimado (Óptima): 0.00 s
Ruta Directa: Dispositivo → ClienteRemotoVPN
Latencia (Directa): 8.84 ms
Tiempo Estimado (Directa): 0.00 s
Simulando transferencia (vía óptima) para cliente.py...
¡Simulación para cliente.py (vía óptima) completada!
--- Todas las simulaciones de este lote han finalizado. ---


Transferencia de Archivos VPN con Ruta Óptima
Selección de Archivos
Seleccionar Archivos
servidor.py
Configuración de Simulación
Dispositivo Destino:
ClienteRemotoVPN
Tamaño de Archivo de Prueba:
100 MB
Simular Transferencia de Prueba
Simular Transferencia de Seleccionados
Registro de Simulación
--- Iniciando simulación para: servidor.py (0.00 MB) ---
Ruta Óptima: Dispositivo → ClienteRemotoVPN
Latencia (Óptima): 8.84 ms
Tiempo Estimado (Óptima): 0.00 s
Ruta Directa: Dispositivo → ClienteRemotoVPN
Latencia (Directa): 8.84 ms
Tiempo Estimado (Directa): 0.00 s
Simulando transferencia (vía óptima) para servidor.py...
¡Simulación para servidor.py (vía óptima) completada!
--- Todas las simulaciones de este lote han finalizado. ---

Kruskal

Este código está diseñado para representar y optimizar la topología de una red utilizando un grafo ponderado, donde los nodos representan dispositivos y las aristas representan enlaces entre ellos con un determinado ancho de banda medido en Mbps.

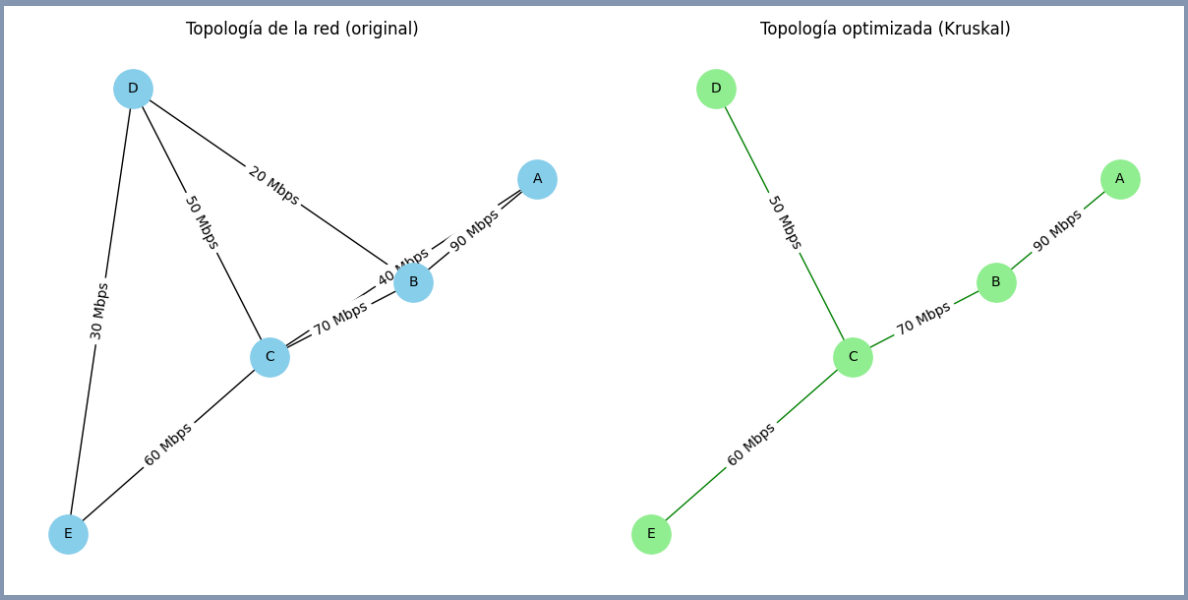
Primero, primero está la lista que contienes las conexiones entre nodos, cada elemento de esta lista es una tupla con tres valores: el nodo de origen, el nodo de destino y el ancho de banda disponible entre ellos.

Con esta lista, se crea el grafo original utilizando la biblioteca networkx. Este grafo contiene todas las conexiones tal como fueron medidas, y cada enlace tiene un peso asociado correspondiente al ancho de banda real. Este grafo permite visualizar cómo está estructurada actualmente la red y cuántos recursos ofrece cada enlace.

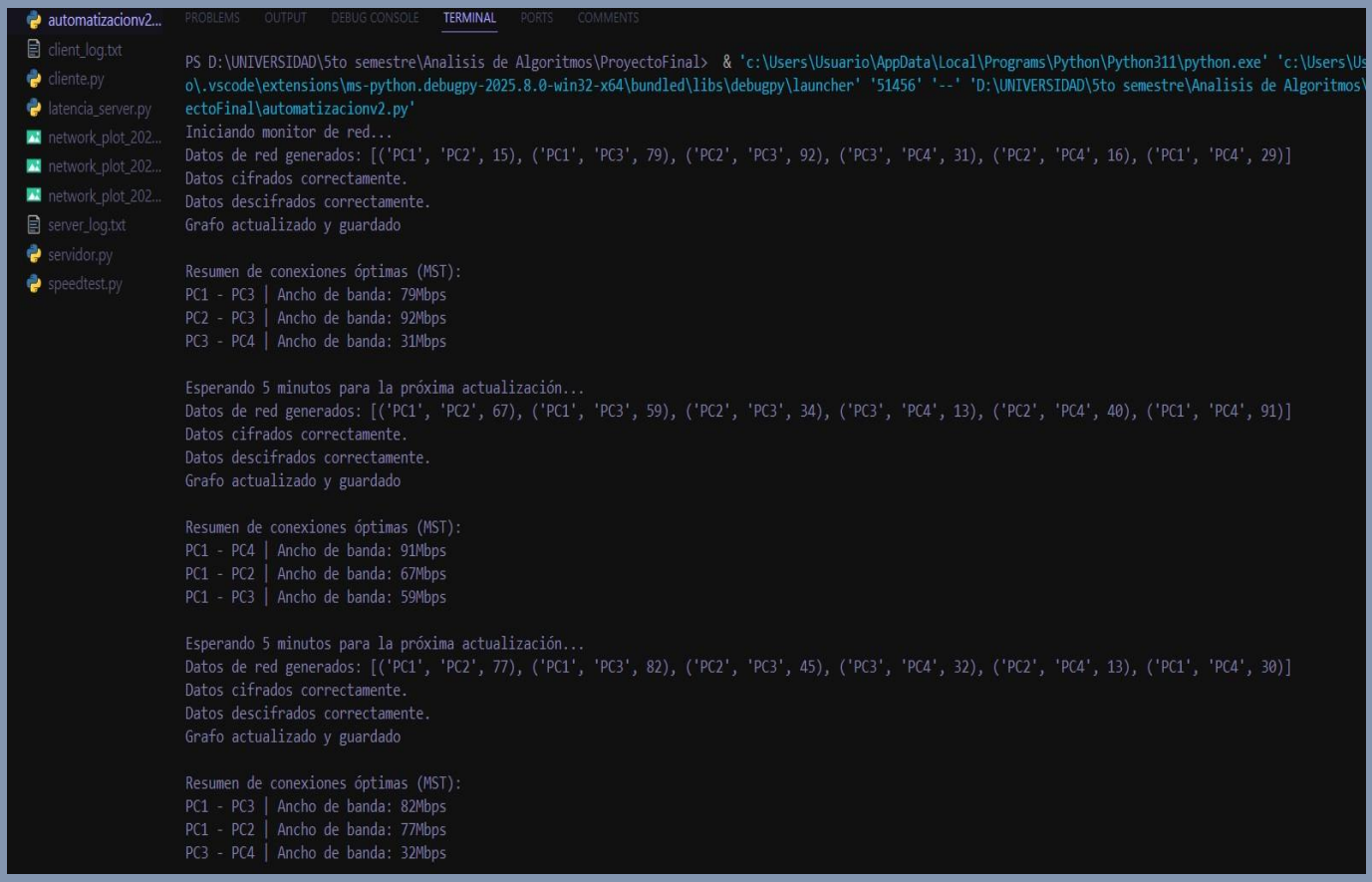
Después, el código genera una segunda versión del grafo donde el ancho de banda se invierte multiplicándolos por -1. esto se hace porque el algoritmo que se usa para optimizar la red, el algoritmo de Kruskal, por defecto selecciona las conexiones de menor peso. Al invertir los valores, se consigue que Kruskal seleccione las conexiones que en realidad tienen el mayor ancho de banda posible, lo cual es deseable para una red eficiente.

Se aplica el algoritmo de Kruskal sobre este grafo invertido para obtener un Árbol de Expansión Mínima, este árbol es una subred que conecta todos los nodos de la forma más eficiente posible, seleccionando solo algunos enlaces pero asegurando que no haya ciclos, y maximizando el rendimiento de la red.

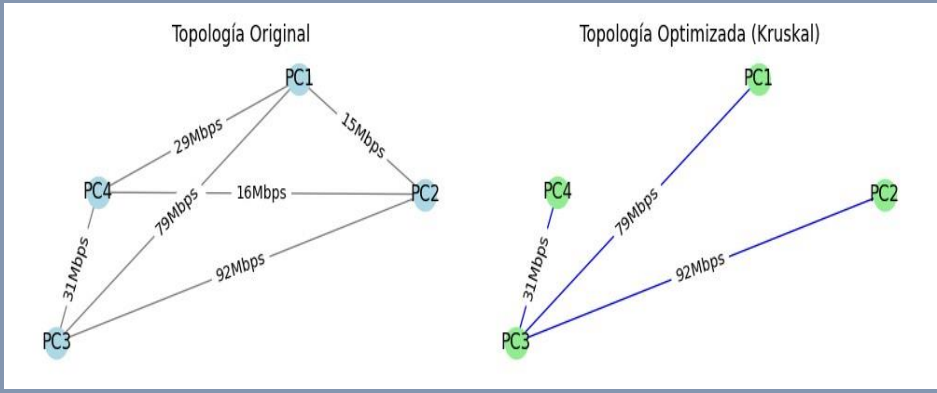
Una vez obtenido el MST, se crea un nuevo grafo con los mismos nodos pero restaurando los valores positivos del ancho de banda. Esto permite interpretar correctamente la información y presentar los resultados en términos comprensibles.



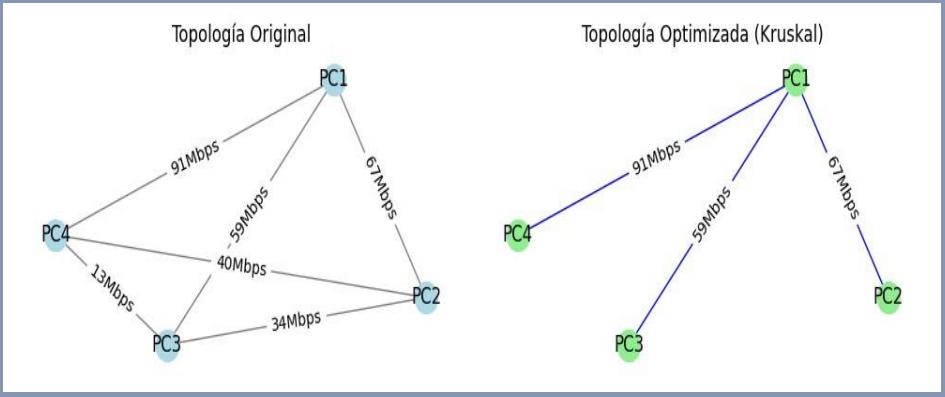
Automatización



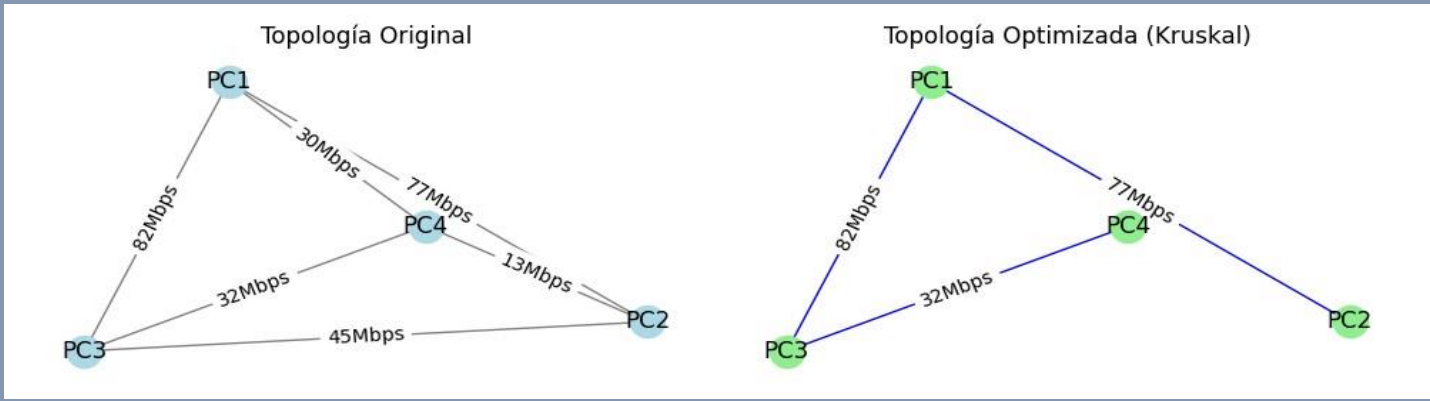
GRAFO GENERADO N1



GRAFO GENERADO N2



GRAFO GENERADO N3



Conclusión.

Los algoritmos greedy como Dijkstra y Kruskal tienen ventajas clave en proyectos como el tuyo. Son rápidos y eficientes para resolver problemas específicos como encontrar caminos más cortos o árboles de expansión mínima, lo que los hace ideales para implementaciones académicas o prototipos. Además, su lógica relativamente sencilla permite integrarlos fácilmente con herramientas como Tkinter para visualización básica y Google Colab para ejecución.

Sin embargo, estos algoritmos también tienen limitaciones importantes. No siempre garantizan soluciones óptimas en problemas más complejos, especialmente si aparecen múltiples restricciones. También pueden volverse incómodos al trabajar con grafos grandes, ya que herramientas como Tkinter no están diseñadas para visualizar eficientemente una gran cantidad de nodos y aristas.

Para ser un proyecto pequeño, fue una buena opción por su equilibrio entre simplicidad y rendimiento. Si el proyecto evoluciona hacia problemas más complejos o grafos más grandes, podrías considerar complementarlos con otras técnicas o herramientas más robustas, como es el casos de Networkxx.