Universidad de Ciencias Aplicadas

INFORME DEL TRABAJO PARCIAL

CURSO DE COMPLEJIDAD ALGORÍTMICA

Carrera de Ingeniería de Software

Sección: WS6D

Proyecto: OptiNet

Código	Nombres y apellidos
u202213208	Christian Renato Espinoza Saenz
u202312230	Daniel José Huapaya Vargas
u202211846	Joaquin David Rivadeneyra Ramos

2024

Contenido

Problema

El crecimiento exponencial de la demanda de servicios de internet de alta velocidad ha impulsado a los proveedores de servicios de internet (ISP) a expandir y mejorar sus redes de fibra óptica para ofrecer conexiones más rápidas y confiables. Sin embargo, el diseño e implementación de estas redes representan un desafío significativo, ya que deben optimizarse para minimizar los costos de infraestructura, manteniendo al mismo tiempo un alto rendimiento y cobertura. El problema principal radica en encontrar la configuración óptima de estas conexiones que minimicen el costo total de la red, sin comprometer la calidad del servicio.

Para resolver este problema, se propone el uso del algoritmo de Kruskal, una técnica que forma parte de los algoritmos de árboles de expansión mínima (MST). Este algoritmo es adecuado cuando se necesita construir una red que minimice el costo total de las conexiones, seleccionando las aristas de menor peso que conecten todos los nodos sin formar ciclos (Cormen, Leiserson, Rivest, & Stein, 2009). La eficiencia del algoritmo de Kruskal se destaca en redes distribuidas geográficamente, donde las distancias entre puntos pueden influir significativamente en los costos de implementación.

El uso del algoritmo de Kruskal en la planificación de redes de fibra óptica es crucial, ya que permite a los ISP reducir los costos de infraestructura y optimizar el uso de recursos. Según Kershenbaum (2003), en redes de telecomunicaciones, la minimización de costos es esencial para la sostenibilidad del proyecto y su expansión futura. Así, la aplicación de OptiNet asegura una red eficiente en términos de costo, sin comprometer la calidad de la conectividad.

Conjunto de Datos (Dataset)

El conjunto de datos utilizado para el análisis corresponde a una representación geográfica de una red de fibra óptica simulada (ver Figura 1). Cada dato en el conjunto incluye ubicaciones geoespaciales de puntos de acceso, representados como nodos, y las posibles conexiones entre ellos, representadas como aristas ponderadas con costos asociados. Los pesos se determinan aleatoriamente.

El dataset está compuesto por las siguientes características principales:

- **Nodos:** Cada nodo representa un punto de acceso a la red de fibra óptica, generalmente ubicaciones estratégicas como estaciones base o centros de distribución de internet.
- **Aristas:** Las posibles conexiones entre nodos, con un costo asociado.
- **Pesos de Conexión:** Cada arista tiene un valor numérico que representa el peso de establecer la conexión entre dos nodos. Los costos pueden variar en función de la distancia, el tipo de terreno, la infraestructura existente y las limitaciones legales o regulatorias.

El origen de los datos es un conjunto simulado de datos (ver Figura 2) donde se planificará un sistema de redes de fibra óptica. Para simulaciones, los datos se generan mediante modelos aleatorios. Estos datos son fundamentales para aplicar el algoritmo de Kruskal, ya que permiten identificar las conexiones más económicas para construir una red de fibra óptica eficiente.

Figura 1

Muestra del dataset (1500 nodos)

	A	B	C
1	Node1	Node2	Weight
2	0	1497	42
3	0	594	177
4	0	1081	164
5	0	497	164
6	1	570	173
7	1	332	132
8	2	512	7
9	2	747	69
10	2	48	77
11	2	68	140
12	3	920	22
13	3	972	52
14	3	586	42
15	3	1455	30
16	3	1430	95
17	4	1440	13
18	4	1488	96
19	4	490	142
20	5	1178	14
21	5	164	85
22	5	1390	162
23	5	1297	73
24	5	855	1
25	6	1435	198
26	6	443	156

Nota. Elaboración propia

Figura 2

Código de generación del dataset

```
import networkx as nx
import pandas as pd
import random

# Function to generate a connected graph with n_nodes
def generate_connected_graph(n_nodes):
    # Create a random spanning tree to ensure connectivity
    G = nx.generators.random_tree(n_nodes)

    # Assign random integer weights between 1 and 200 to the
spanning tree edges
    for u, v in G.edges():
        G[u][v]['weight'] = random.randint(1, 200)

    # Randomly add more edges to increase connectivity/density
    additional_edges = int(n_nodes * 1.5)  # Adjust the factor to
control edge density
    edges_added = 0

    while edges_added < additional_edges:
        u = random.randint(0, n_nodes - 1)
        v = random.randint(0, n_nodes - 1)

        # Check that u and v are different nodes and not already
connected
        if u != v and not G.has_edge(u, v):
            weight = random.randint(1, 200)  # Assign random
positive weight
            G.add_edge(u, v, weight=weight)
            edges_added += 1  # Increment only when an edge is
successfully added

    return G
```

Nota. Elaboración propia.

Propuesta

Objetivo

El objetivo de esta propuesta es diseñar una red de fibra óptica optimizada para proveedores de servicios de internet (ISP), que conecte múltiples puntos de acceso geográficamente distribuidos, minimizando los costos de infraestructura. Esto se logrará mediante la aplicación del algoritmo de Kruskal, que permite reducir el costo total de las conexiones entre los puntos de acceso, manteniendo una red eficiente y escalable (GeeksforGeeks, 2023).

Técnica Utilizada

La técnica utilizada será el algoritmo de Kruska**l**, un método eficiente para la resolución de problemas de árboles de expansión mínima (MST). Kruskal se caracteriza por conectar los nodos del grafo de forma que el costo total de las conexiones sea el mínimo posible, evitando ciclos y redundancias. Este algoritmo es ideal para diseñar redes donde se requiere minimizar costos en infraestructuras de telecomunicaciones (Ravikiran, 2024).

Metodología

1. **Recopilación de Datos:** Se utilizarán conjuntos de datos simulados que especifique los puntos de acceso y los costos asociados a las posibles conexiones entre ellos.

2. **Modelado del Grafo:** Los puntos de acceso se representarán como nodos en un grafo, y las posibles conexiones entre ellos como aristas ponderadas por su costo.

3. **Aplicación del Algoritmo de Kruskal:** Se aplicará el algoritmo de Kruskal para seleccionar las conexiones con menor costo total, asegurando que todos los puntos de acceso queden conectados sin crear ciclos.

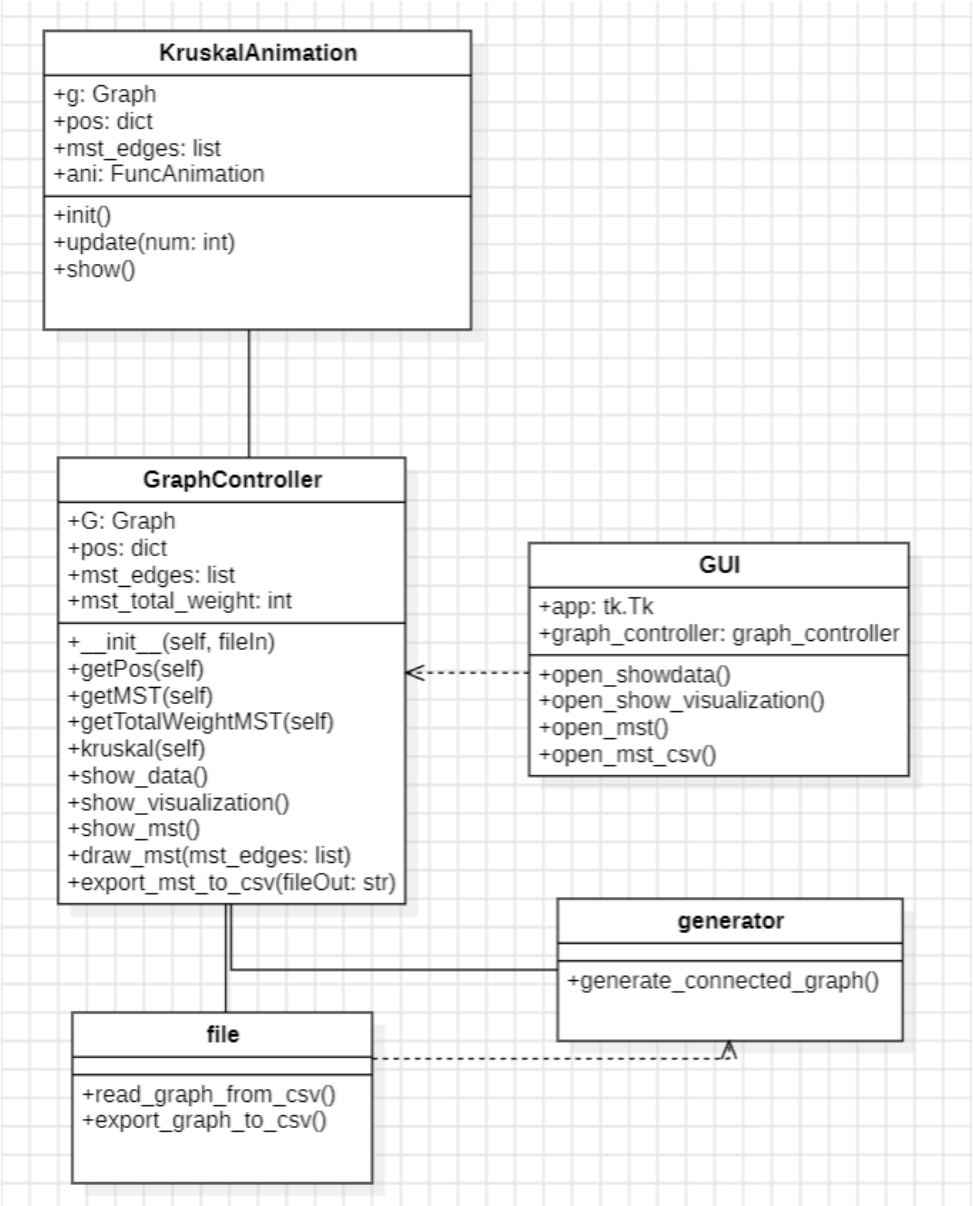
4. **Evaluación del Diseño:** Se validará la red propuesta para asegurar que minimiza los costos y cumple con los requisitos de conectividad, escalabilidad y eficiencia de la red.

Diseño del aplicativo

Diagrama de clase y diagrama de base de datos

Figura 3

Diagrama de clases UML para el sistema completo.



Nota. El diagrama muestra cómo KruskalAnimation se asocia con GraphController para animar el algoritmo de Kruskal, usando el gráfico y las posiciones de los nodos. GraphController controla el gráfico, calculando el MST y su peso total, mostrando los datos, visualizaciones, y exportando el MST a CSV. file se asocia con GraphController para leer gráficos desde archivos CSV y exportarlos, mientras que generator genera gráficos conectados aleatorios. La GUI depende de GraphController para abrir ventanas que muestran los datos, visualizaciones y exportar el MST.

Módulo para la adecuación de datos fuente a estructura de datos tipo grafo

Figura 4

Código de adecuación de datos fuente

```
def read_graph_from_csv(file_path):
    try:
        # Attempt to read the CSV file
        df = pd.read_csv(file_path)

        # Create an empty undirected graph
        G = nx.Graph()

        # Add edges to the graph from the DataFrame
        for index, row in df.iterrows():
            node1 = row['Node1']
            node2 = row['Node2']
            weight = row['Weight']
            G.add_edge(node1, node2, weight=weight)

        return G

    except FileNotFoundError:
        print(f"Error: The file '{file_path}' was not found.")
    except PermissionError:
        print(f"Error: The file '{file_path}' is already open or you
don't have permission to read it.")
    except pd.errors.EmptyDataError:
        print(f"Error: The file '{file_path}' is empty.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

Nota. Esto se encuentra en el archivo file.py y se invoca en GraphController.py en el constructor. Elaboración propia.

Módulo para la representación de datos en el grafo

Figura 5

Función para dibujar el MST directamente

```
    def draw_mst(self, mst_edges):
        plt.close('all')
        plt.figure(figsize=(10, 8))
        pos = self.getPos()
        nx.draw_networkx_nodes(self.G, pos, node_size=50)
        nx.draw_networkx_edges(self.G, pos, edgelist=self.G.edges,
edge_color='lightgray')
        nx.draw_networkx_edges(self.G, pos, edgelist=[(u, v) for u, v,
w in mst_edges], edge_color='blue', width=2)
        plt.title("Minimum Spanning Tree (MST)" + f" | Total Weight:
{self.getTotalWeightMST()}")
        plt.show()
```

Nota. Código de GraphController.py. Elaboración propia.

Figura 6

Función (constructor de KruskalAnimation) para animar una representación del algoritmo Kruskal.

```
class KruskalAnimation:
    def __init__(self, G, pos, edges):
        self.G = G
        self.pos = pos
        self.mst_edges = edges
        self.fig = plt.figure(figsize=(10, 8))
        self.ani = None

        # Initialize the plot with the initial state of the graph
        self.init()

        # Create the animation
        self.ani = animation.FuncAnimation(
            self.fig,
            self.update,
            frames=len(self.mst_edges) + 1,
            interval=50,  # Adjust interval if needed
            repeat=False
        )
```

Nota. El código es de KruskalAnimation.py e inicia cuando se ejecuta la función plt.show(). Elaboración propia.

Interfaz tentativa (visual) para usuarios

Figura 7

Interfaz gráfica inicial

Nota. Los rectángulos en azul son botones presionables. “Mostrar datos” muestra el número de nodos, aristas y una tabla con las conexiones de cada nodo, peso promedio y acumulado del grafo cargado en la base de datos, “Abrir dataset.csv” abre el dataset en excel, “Mostrar visualización” abre la animación del algoritmo hasta encontrar el MST, “Mostrar MST” muestra el MST directamente, y “Mostrar MST CSV” exporta el resultado a un excel y lo abre. Elaboración propia.

Validación de resultados y pruebas

Entradas:

- Archivo dataset.csv que contiene los nodos, conexiones y pesos del grafo.
- MST calculado a partir de los datos del grafo.
- Archivo MST.csv que contiene las conexiones del MST.

Salidas:

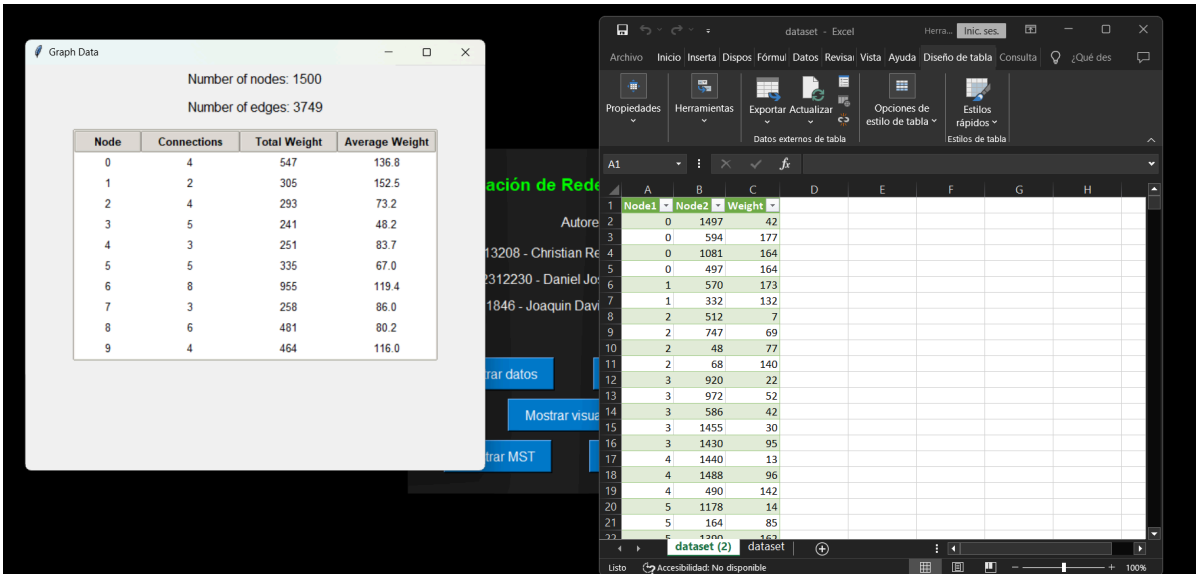
- Visualización de los datos del archivo dataset.csv en la interfaz gráfica del programa.
- El archivo dataset.csv se abre en Excel para visualizar el contenido del grafo.
- Visualización animada del proceso paso a paso de cómo el algoritmo de Kruskal selecciona las aristas para formar el MST.
- Visualización del MST final en la interfaz gráfica.
- El archivo MST.csv se abre en Excel, mostrando las conexiones y pesos del MST.

Pruebas:

Mostrar los datos del dataset : Abrir el archivo dataset.csv y verificar que los datos se carguen correctamente en la interfaz gráfica.

Figura 8

Comparación de los datos entre el archivo Excel abierto y la aplicación



Mostrar visualización : Verificar que el programa funcione correctamente y no presente errores al cargar la animación.

Figura 9

Menú inicial y mensaje de espera

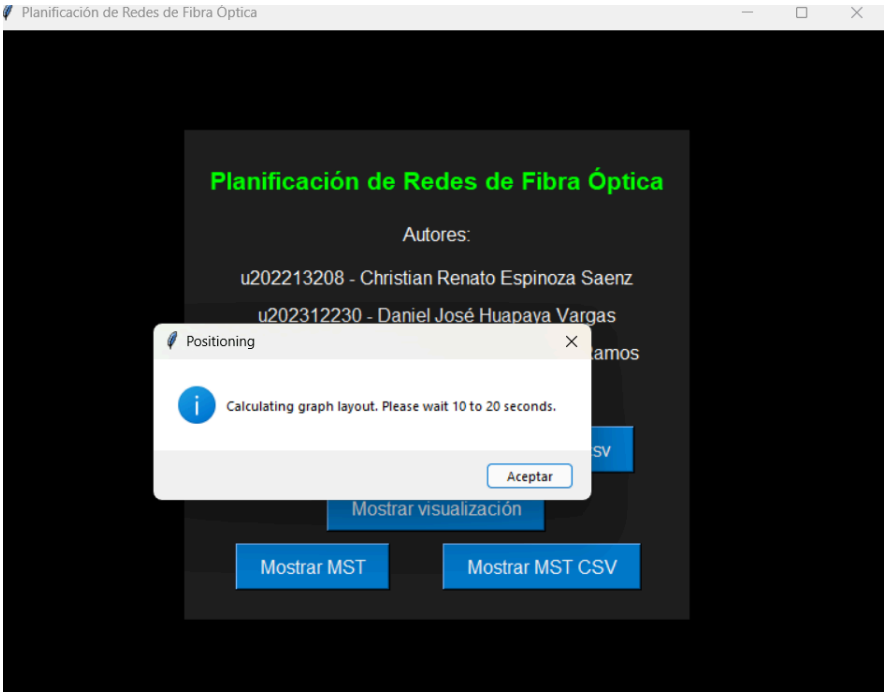


Figura 10

Inicio de la animación

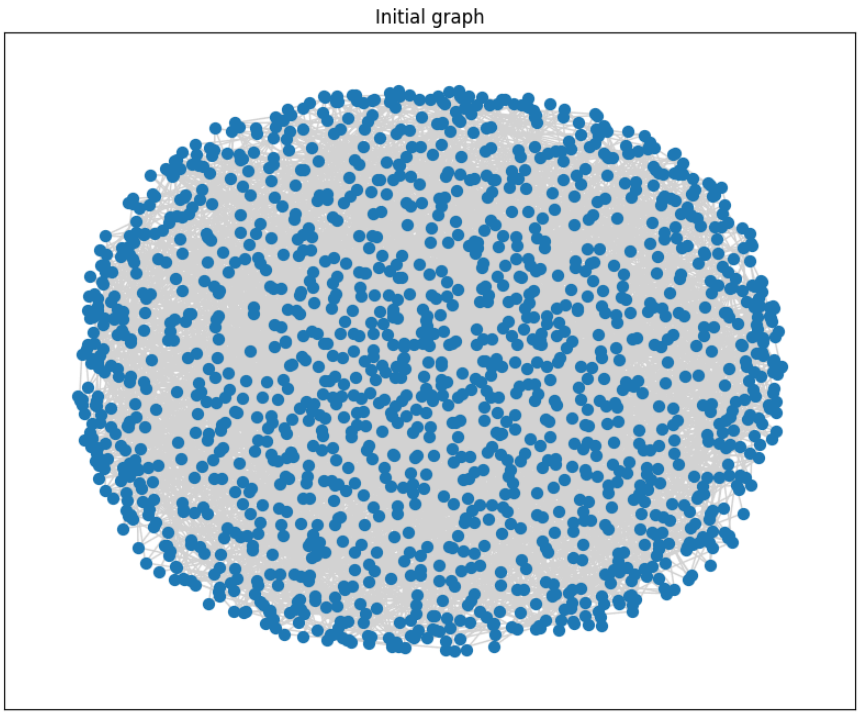
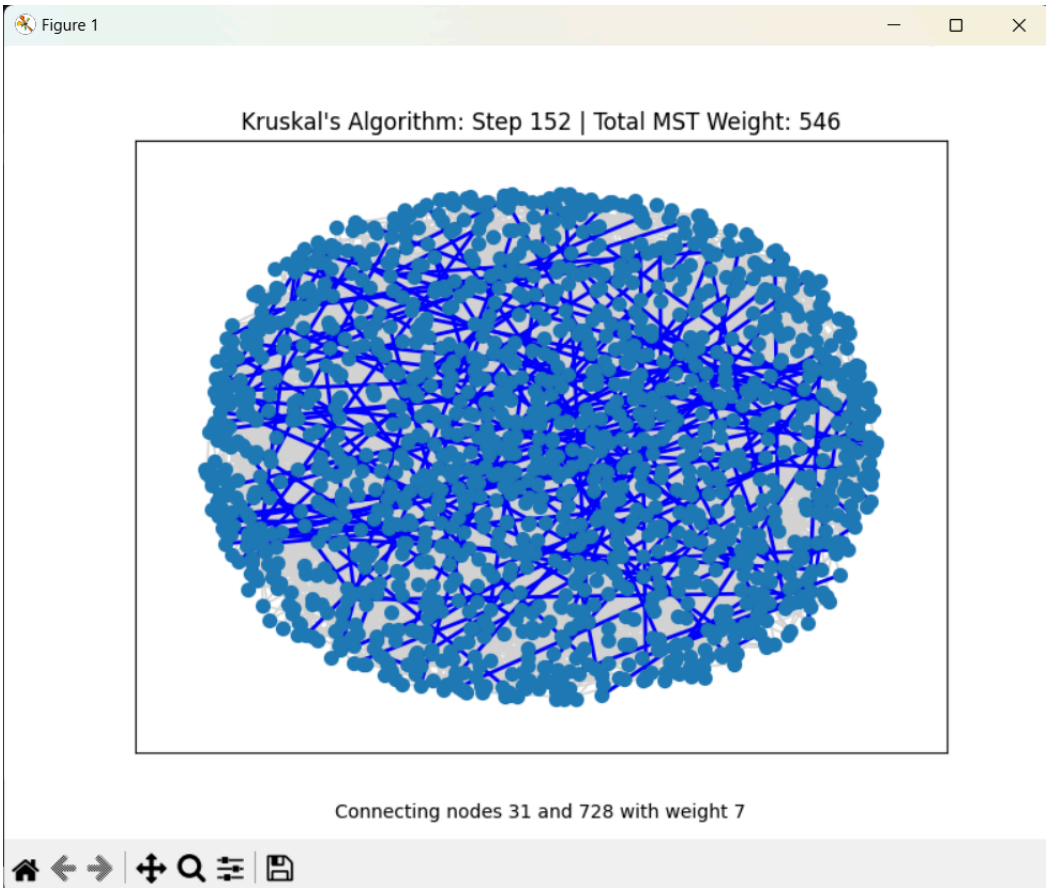


Figura 11

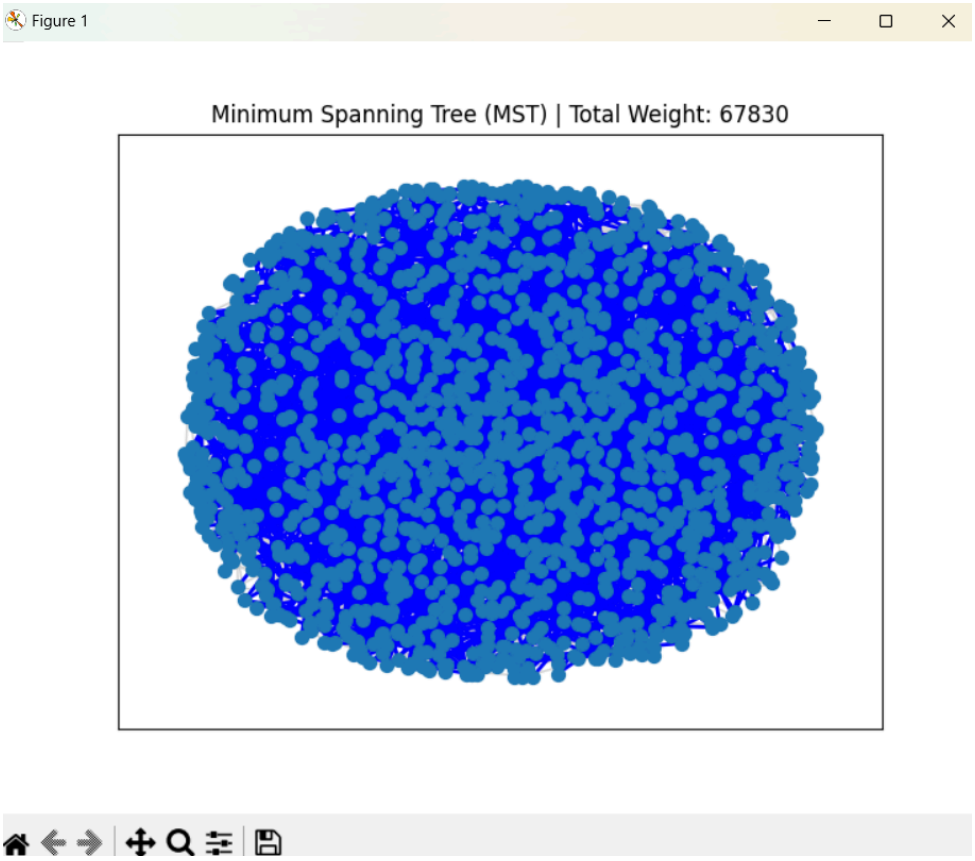
Animación en el paso 152



Mostrar MST: Verificar que el MST mostrado es correcto y que todos los nodos están conectados sin ciclos.

Figura 12

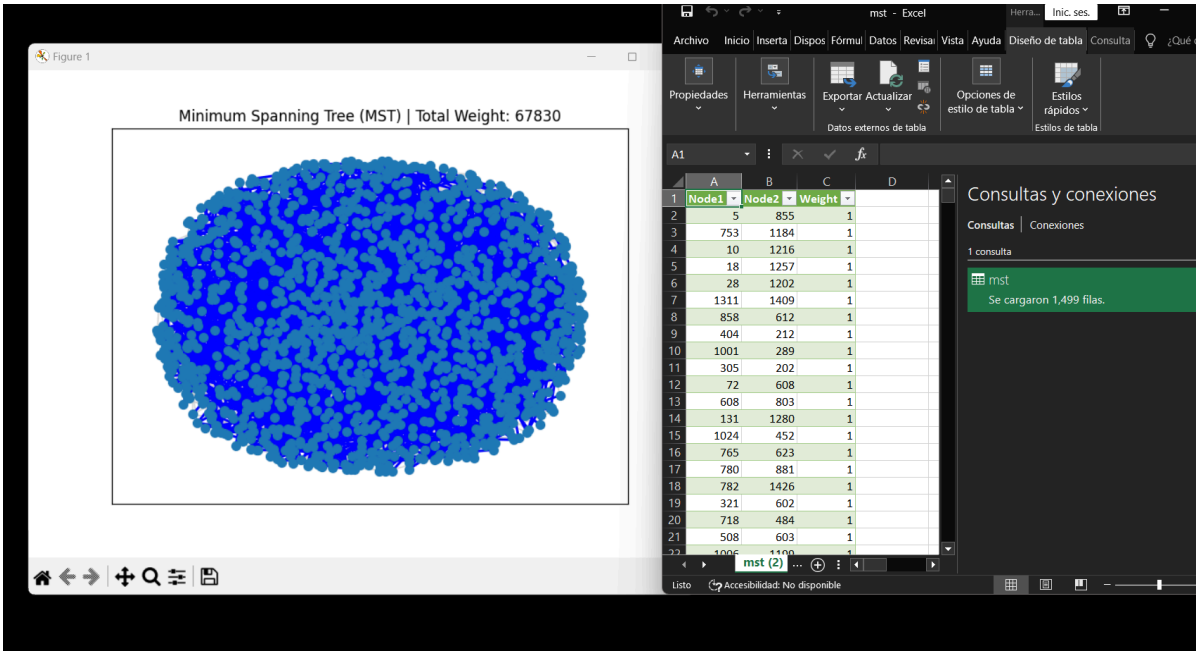
Figura del MST final



Mostrar MST CSV: Abrir el archivo MST.csv y comprobar que se abre correctamente en Excel con todas las conexiones y pesos calculados.

Figura 13

Comparación entre el MST final y los datos exportados a Excel



Interpretación de resultados:

Nuestro programa logra mostrar datos, visualizar animaciones de manera efectiva, exportar datos a Excel y leer información de archivos Excel de forma satisfactoria. Esta capacidad de manejar y procesar información, junto con la habilidad de dibujar gráficos, asegura la conectividad entre múltiples puntos de acceso y permite una red escalable, lo que convierte al programa en una herramienta valiosa para optimizar las infraestructuras de los proveedores de servicios de internet.

Conclusiones

El diseño de OptiNet sobre una red de fibra óptica mediante el algoritmo de Kruskal resultó ser una estrategia eficiente para optimizar las conexiones entre múltiples puntos de acceso, reduciendo de manera significativa los costos de infraestructura. Este enfoque no solo garantiza una red de bajo costo, sino que también asegura que no existan ciclos redundantes, lo que mejora la eficiencia general del sistema. La implementación de una base de datos estructurada, que gestiona la información de los nodos, las conexiones y los costos, fue clave para facilitar el uso del algoritmo y simplificar la toma de decisiones sobre las mejores rutas a seguir. Este diseño proporciona una base sólida para futuras expansiones de la red. De cara al futuro, sería valioso considerar factores adicionales como la latencia de las conexiones, la redundancia en caso de fallos, y la posibilidad de incorporar algoritmos más avanzados o simulaciones para mejorar la resiliencia y escalabilidad, especialmente en redes de telecomunicaciones más grandes y complejas. Esto permitirá enfrentar desafíos más amplios y mantener la red adaptable a las crecientes demandas de conectividad hoy en día.

Referencias

GeeksforGeeks. (2023, 05 de octubre). *Kruskal s Minimum Spanning Tree MST Algorithm*. Recuperado de https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

Ravikiran, A. (2024, 31 de agosto). *Your One-Stop solution to learn Kruskal algorithm from scratch*. Simplilearn. Recuperado de https://www.simplilearn.com/tutorials/data-structure-tutorial/kruskal-algorithm

Kershenbaum, A. (2003). *Telecommunications Network Design Algorithms*. McGraw-Hill.

DataScientest. (2024, 20 de marzo.). *Kruskal algorithm: Definition and purpose*. DataScientest. Recuperado de https://datascientest.com/en/kruskal-algorithm-definition-and-purpose