Proyecto Final

30/11/2024

Ingeniería en Computación

Análisis de Algoritmos

Jorge Ernesto López Arce

- *Luis Angel Lozano Reyes*
- *Jesús Antonio Torres Contreras*

Introducción

El siguiente proyecto consta de una investigación exhaustiva y un análisis sobre las redes urbanas a través de herramientas tales como los grafos que nos representa una gran oportunidad para poder analizar este tipo de ciudades mediante algoritmos los cuales nos ayudan a visualizar y entender de mejor manera todo nuestro proyecto.

En este proyecto se nos pidió implementar algoritmos para encontrar la distancia optima para lo cual usamos el algoritmo de Dijkstra para encontrar la mejor ruta mientras que para construir nuestro árbol de expansión mínima (MST) nos ayudan a visualizar y analizar la eficiencia de dichos algoritmos en tareas de búsqueda y profundidad.

El tema por el cual estamos realizando dicho proyecto se centra en modelar caminos entre dos puntos para encontrar la ruta mas eficiente la cual nos ahorre temas de distancia y tiempo para esto usamos los nodos o puntos de interés y las aristas que en este código nos simbolizan la ruta entre nodo y nodo este enfoque hacia este tipo de problemas nos ayuda a analizar y resolver problemas en los cuales se necesita encontrar la ruta mas cercana entre dos puntos.

La resolución de este tipo de problemas mediante técnicas de programación nos proporciona una visión mas clara acerca de las limitaciones, fortalezas y mejoras en un grafo urbano.

Conceptos Claves:

- Grafo urbano: Es la representación gráfica y matemática de una red formada por notos y aristas. Es decir, un grafo urbano es el modelado mediante nodos y aristas de cualquier espacio en este caso una ciudad.
- Algoritmo de Dijkstra: Este algoritmo fue creado para encontrar una solución eficiente al buscar una ruta de dos puntos.
- Algoritmo de Prim: Es un subgrafo que conecta todos los nodos de un grafo con el valor mínimo de cada uno así con esto garantizando que no haya ciclos.

Objetivo general:

Implementar un grafo urbano para encontrar la ruta optima y con esta información observar el comportamiento de los algoritmos implementados (Dijkstra, Prim) y evaluar las limitaciones que estos conllevan así mismo analizar la escalabilidad, precisión y eficiencia de dichos algoritmos.

Objetivos específicos:

- Emplear algoritmos para encontrar la ruta más corta entre dos puntos.
- Construir un grafo urbano para su posterior análisis.
- Visualizar los grafos urbanos con la menor distancia entre dos puntos.
- Analizar mediante las técnicas de análisis de algoritmos para encontrar su complejidad temporal.

Aplicación del algoritmo

Herramientas iniciales:

- osmnx: Manipular grafos urbanos.
- random: Realizar selecciones de números al azar
- heapq: Implementar colas de prioridad para que funcionen de manera correcta los algoritmos de Dijkstra y Prim
- time: Para medir los tiempos de ejecución
- matplotlib: Grafica los resultados obtenidos

Dataset:

- El grafo urbano de Tamazula de Gordiano, Jalisco, México

1. Implementación de las siguientes librerías las cuales nos sirven para exportar nuestro grafo urbano

```python
import osmnx as ox
import random
import heapq
import time
import matplotlib.pyplot as plt
```

2. Función que nos ayuda a importar nuestra ciudad seleccionada pero ahora siendo un grafo urbano.

```python
# Cargar el grafo
place_name = "Tamazula de Gordiano, Jalisco, Mexico"
G = ox.graph_from_place(place_name, network_type="drive")
```

3. Se asignan velocidades máximas a las aristas del grafo (en caso de que no estén ya presentes), y se calculan los "pesos" de las aristas como la longitud de la carretera dividida por la velocidad máxima permitida.

```python
# Configuracion inicial de aristas
for edge in G.edges:
    Velocidad_mx = 40
    if "Velocidad_mx" in G.edges[edge]:
        Velocidad_mx = G.edges[edge]["Velocidad_mx"]
        if isinstance(Velocidad_mx, list):
            Velocidad_mx = min([int(vel) for vel in Velocidad_mx])
        elif isinstance(Velocidad_mx, str):
            Velocidad_mx = int(Velocidad_mx)
    G.edges[edge]["Velocidad_mx"] = Velocidad_mx
    G.edges[edge]["Pes"] = G.edges[edge]["length"] / Velocidad_mx

```

4. Configuración visual de las aristas, estas funciones actualizan los atributos de color y tamaño de los nodos y las aristas, dependiendo de si estamos visualizando un grafo sin modificar, el árbol de expansión mínima (MST) o la ruta óptima entre dos puntos.

```python
# Estilos para nodos y aristas
def estilo_nov_edge(edge):
    G.edges[edge].update({"color": "#cccccc", "alpha": 0.3, "Linea": 0.5})

def estilo_mst_edge(edge):
    G.edges[edge].update({"color": "#34a853", "alpha": 1, "Linea": 2})
```

5. Con esta función nos permite tener una visualización del grafo que estamos utilizando y con esta función aplicamos los estilos que previamente habíamos definido y mostrando el grafico con la función matplotlib.

```python
# Funcion para dibujar el grafo
def plot_graph(title):
    fig, ax = ox.plot_graph(
        G,
        node_size=[G.nodes[node].get("tamano", 0) for node in G.nodes],
        edge_color=[G.edges[edge]["color"] for edge in G.edges],
        edge_alpha=[G.edges[edge]["alpha"] for edge in G.edges],
        edge_linewidth=[G.edges[edge]["Linea"] for edge in G.edges],
        node_color=[G.nodes[node].get("color", "#f0f0f0") for node in G.nodes],
        bgcolor="white",
        show=False,
        close=False,
    )
    ax.set_title(title, color="black", fontsize=15)
    plt.show()
```

6. Esta es la función que nos genera el árbol mínimo de expansión (MST) y este comienza desde un nodo origen y el cual expande el MST tomando en cuenta las aristas de menor peso.

```python
def prim(orig, dest, title):
    for node in G.nodes:
        G.nodes[node]["tamano"] = 0
    for edge in G.edges:
        estilo_nov_edge(edge)

    visitado = {orig}
    pq = []
    mst_edges = []
```

7. Esta función nos ayuda a encontrar el camino mínimo entres dos nodos con esta hacemos uso de una cola de prioridad para mapear nuestro grafo desde el nodo origen hasta nuestro destino

```python
# Algoritmo de Dijkstra
def dijkstra(orig, dest, title):
    for node in G.nodes:
        G.nodes[node].update({"visitado": False, "distancia": float("inf"), "Anterior": None, "tamano": 0})
    for edge in G.edges:
        estilo_nov_edge(edge)

    G.nodes[orig]["distancia"] = 0
    estilo_punto_inicio(orig)
    estilo_punto_destino(dest)

    pq = [(0, orig)]
```

8. Con la función del análisis del crecimiento lo que hace es evaluar el tiempo de ejecución de los algoritmos de Dijkstra y Prim.

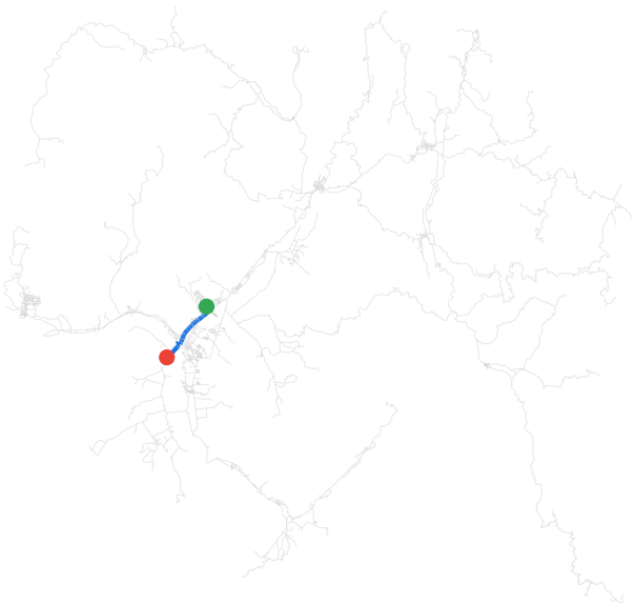
```python
# Funcion para medir tiempos y generar una grafica consolidada
def ejecutar_analisis_crecimiento_grafo(ejemplos=3):
    tamaños_grafo = []
    tiempos_dijkstra_global = []
    tiempos_prim_global = []
```

9. Con esta línea de código lo que hacemos es llamar a la función para ejecutar la función que nos corre todo el programa.

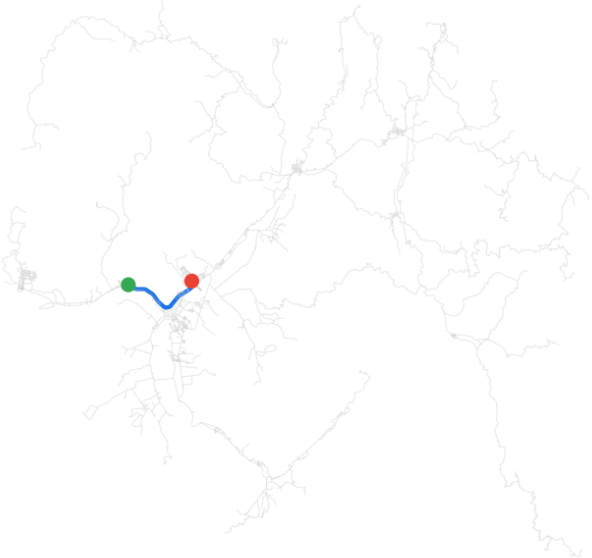
```python
# Ejecutar
ejecutar_analisis_crecimiento_grafo()
```

Resultados

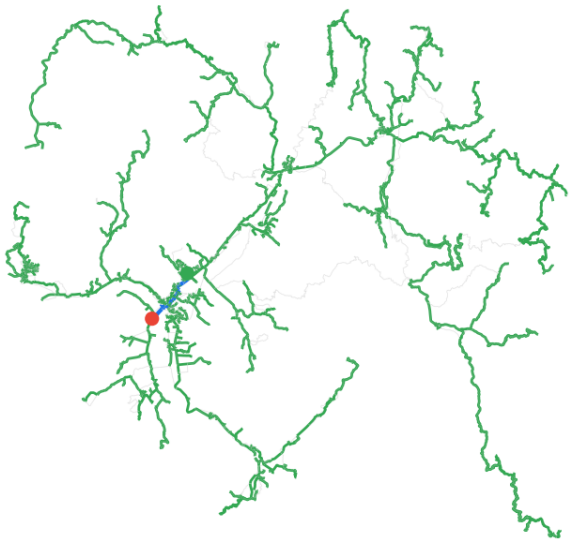
Dijkstra: A → B en subgrafo (50 nodos)



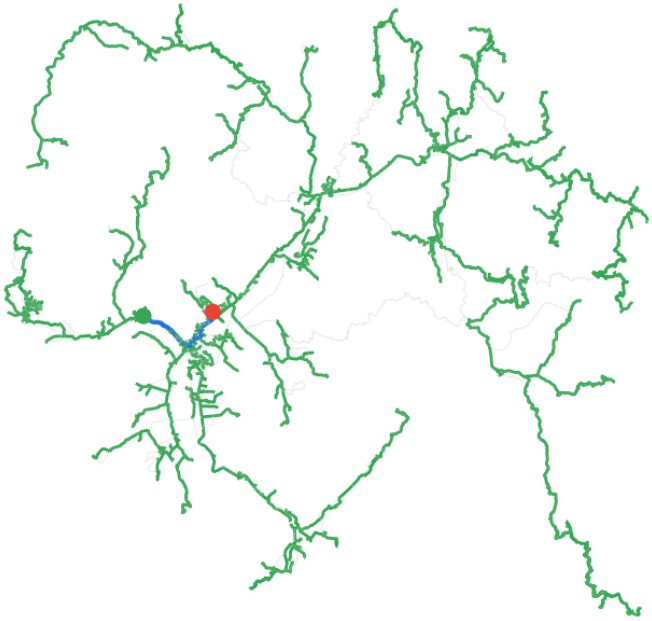
Dijkstra: B → C en subgrafo (50 nodos)



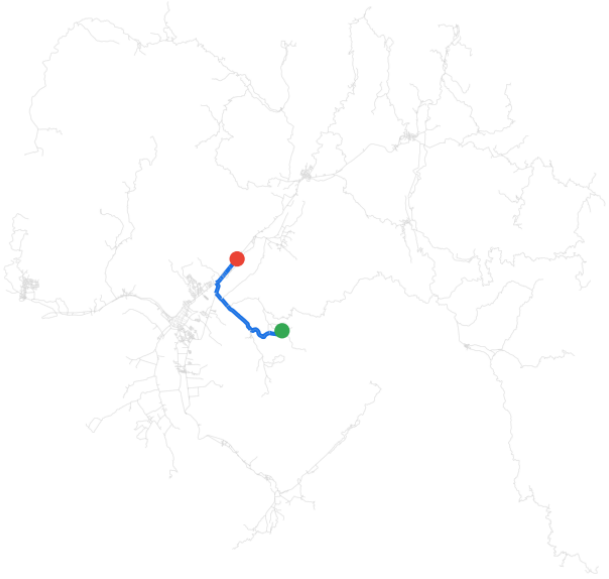
Prim: Arbol y ruta (A → B, 50 nodos)



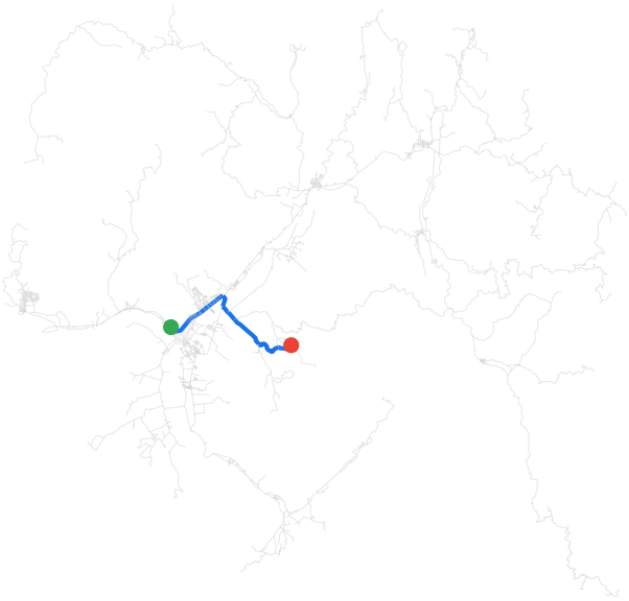
Prim: Arbol y ruta (B → C, 50 nodos)



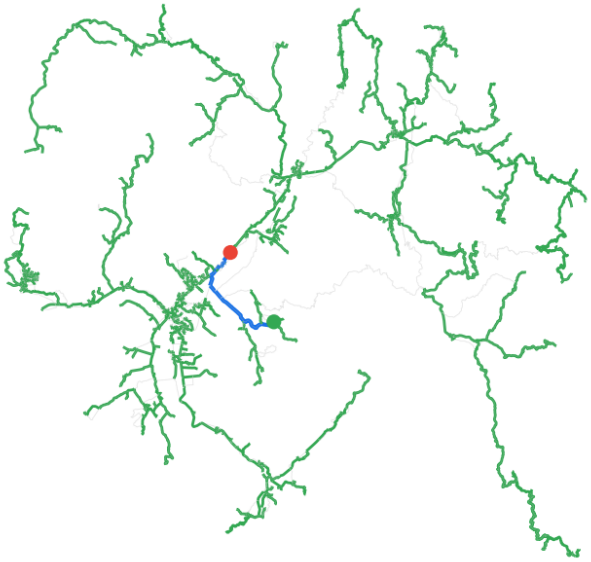
Dijkstra: A → B en subgrafo (453 nodos)



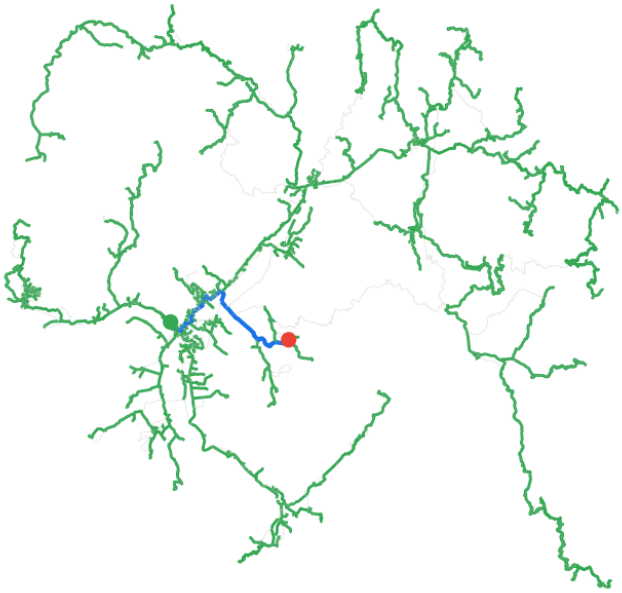
Dijkstra: B → C en subgrafo (453 nodos)



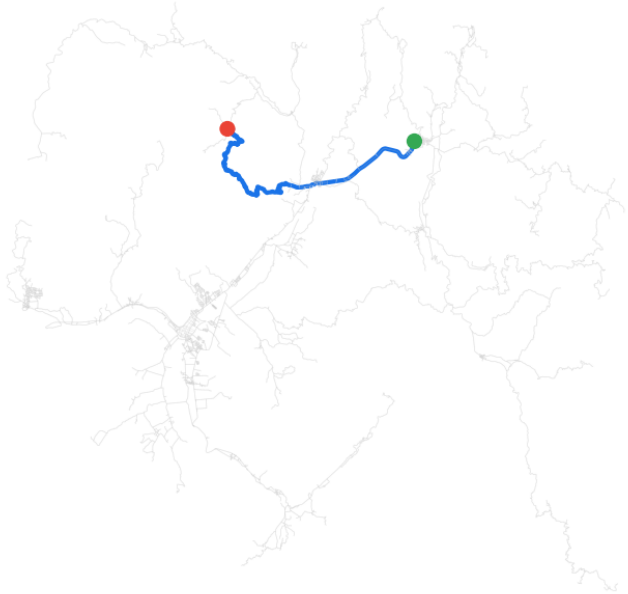
Prim: Arbol y ruta (A → B, 453 nodos)



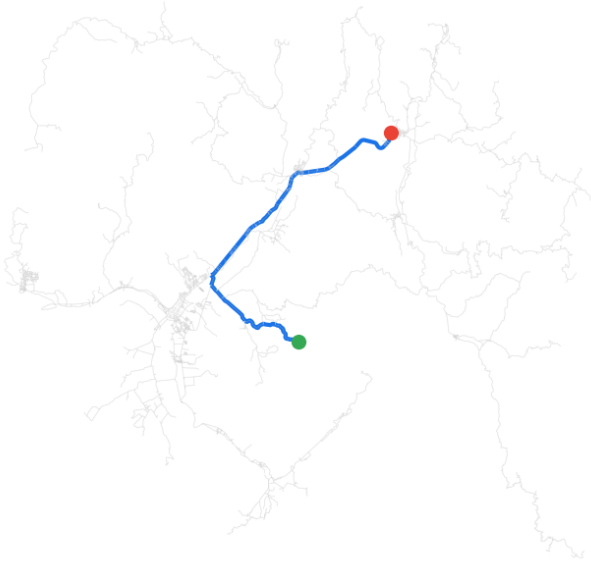
Prim: Arbol y ruta (B → C, 453 nodos)



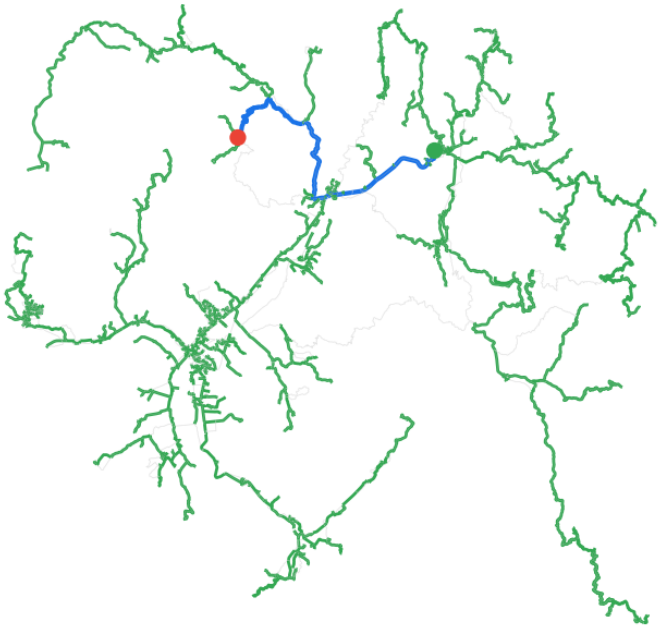
Dijkstra: A → B en subgrafo (1662 nodos)



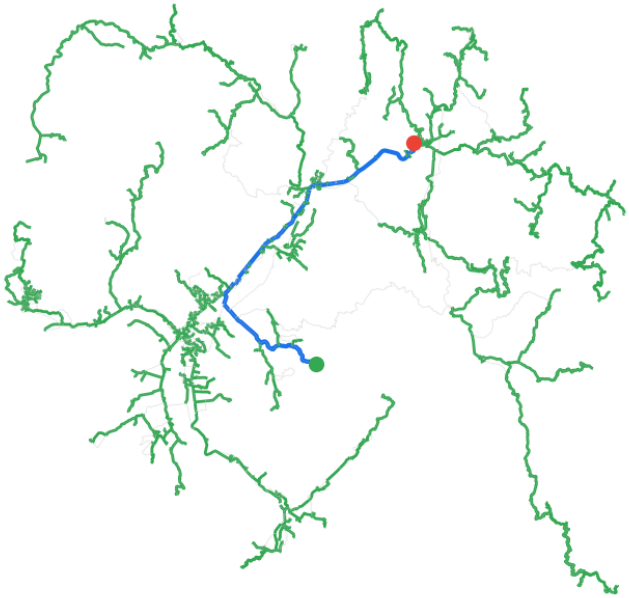
Dijkstra: B → C en subgrafo (1662 nodos)



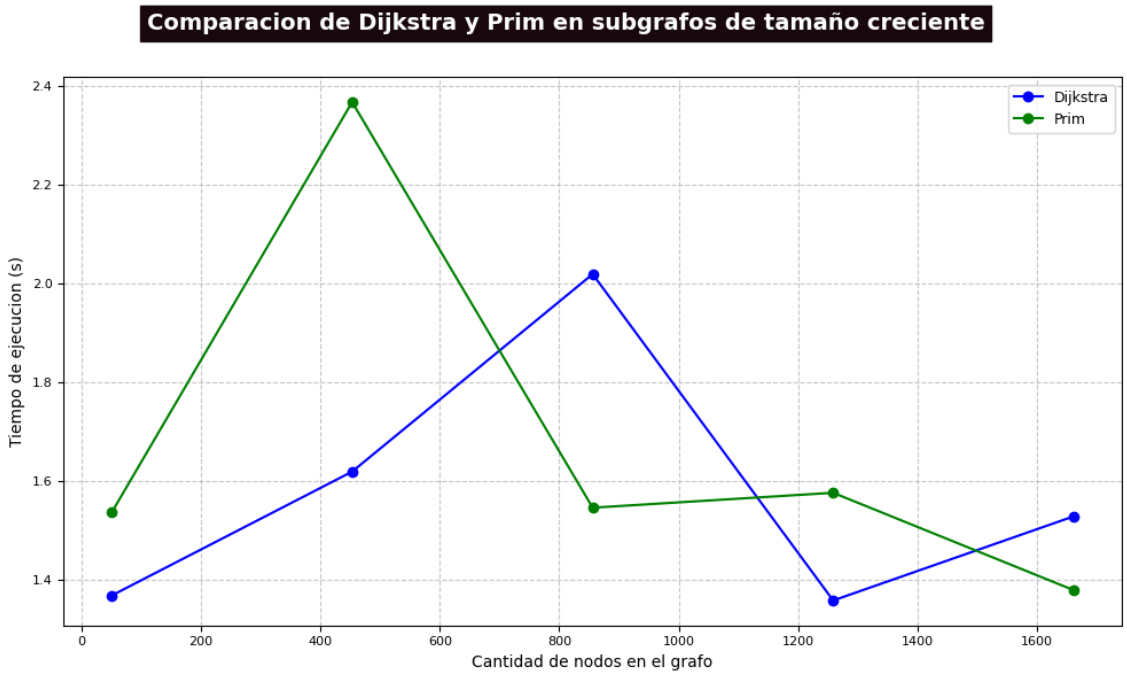
Prim: Arbol y ruta (A → B, 1662 nodos)



Prim: Arbol y ruta (B → C, 1662 nodos)



Comparación de los tiempos de ejecución de los algoritmos de Dijkstra y Prim



Análisis critico

¿Cómo se comportan al aumentar el número de nodos y aristas?

Mientras más nodos haya dentro del grafo más complicado es correr el programa debido a todas las iteraciones que tiene que realizan los algoritmos para encontrar la ruta indicada.

Considera aspectos como cambios dinámicos en los datos de las rutas y la viabilidad de aplicar estos algoritmos en tiempo real.

Para mejorar estos rendimientos se podrían incorporar algunas APIs como Google maps la cual ya hace todo este trabajo y baja considerablemente el rendimiento en la computadora.

Conclusiones

Luis Ángel Lozano Reyes

Al momento de implementar todos los aspectos que se nos describieron en el proyecto me permitió analizar muchos detalles acerca de los grafos urbanos y como se aplican a la vida diaria para resolver los problemas que presentamos todos al momento de movilizarnos por la ciudad. Además de esto termine dándome cuenta lo importante que son las matemáticas en el mundo de la programación y como esta es la base de todo lo que conocemos.

Y bueno creo que para concluir este proyecto quiero destacar algunas de las limitantes que tuvimos al momento de hacer esta implementación ya que en grandes grafos (en este caso ciudades) la eficiencia baja ya que su desempeño se vio afectado en este tipo de grafos urbanos ya que contenían mucha información la cual hizo que mas de una vez tuviéramos problemas.

Jesus Antonio Torres Contreras

Trabajar con los algoritmos de Dijkstra y Prim en este proyecto fue una muy interesante, ya que permitió observar cómo herramientas teóricas pueden aplicarse a problemas del mundo real, como en este caso que lo aplicamos a los grafos de mi ciudad (Tamazula de Gordiano). Ambos algoritmos demostraron ser eficientes en contextos distintos: Dijkstra para encontrar rutas específicas y Prim para conectar puntos de manera global. Fue interesante ver cómo cada uno tiene su propia fortaleza dependiendo del escenario.

Una de las partes más satisfactorias fue visualizar los resultados de manera gráfica. Esto no solo facilitó la interpretación de los algoritmos, sino que hizo evidente cómo el diseño del grafo y los datos que lo componen impactan en las soluciones obtenidas. Sin embargo, también quedó claro que los datos no siempre son perfectos. Por ejemplo, la falta de información precisa en algunos atributos, como las velocidades máximas, puede afectar los resultados finales, algo que sería interesante abordar en proyectos futuros.

En general, este proyecto me ayudó a comprender conceptos clave de este tipo de algoritmos y su interacción con datos geográficos. Además, me dio una nueva perspectiva sobre cómo se pueden usar estas herramientas para resolver problemas prácticos, lo que abre muchas posibilidades, como por ejemplo para aplicaciones de navegación. Fue una experiencia desafiante, pero a la vez fue muy gratificante y sé que me ayudara a futuros proyectos.

Bibliografías

- *OSMnx 2.0.0 documentation*. (s. f.). https://osmnx.readthedocs.io/en/stable/
- *Big-O-calculator*. (2023, 7 enero). PyPI. https://pypi.org/project/big-O-calculator/
- *Plot types — Matplotlib 3.9.2 documentation*. (s. f.). https://matplotlib.org/stable/plot_types/index.html
- Torrubia, G., & Terrazas, V. (2012). Algoritmo de Dijkstra. Un tutorial interactivo. *VII Jornadas de Enseñanza Universitaria de la Informática (JENUI 2001)*.
- Restrepo, J. H., & Sánchez, J. J. (2004). Aplicación de la teoría de grafos y el algoritmo de Dijkstra para determinar las distancias y las rutas más cortas en una ciudad. *Scientia et technica*, *10*(26), 121-126.
- Mostaccio, C. A., & Pérez, G. A. (2016). Algoritmo de Prim.
- Granera, J. A., Valdivia, V. M., & Dávila, M. E. B. (2016). Aplicación informática KPTS (Kruskal, Prim, Tabu Search). *Revista Científica Estelí*, (17), 81-90.