Tecnológico de Monterrey

E2. Actividad Integradora 2

Aaron Hernandez Jimenez A01642529

Jorge Antonio Arizpe Cantú A01637441

Course: Advanced Algorithm Analysis and Design (Group 602)

24 nov 2024

Tecnológico de Monterrey

Tecnológico de Monterrey

1. Algoritmo de Prim para Árbol de Expansión Mínima

- **Descripción**: El algoritmo de Prim encuentra un árbol de expansión mínima en un grafo ponderado. Comienza desde un vértice arbitrario y va creciendo el árbol seleccionando en cada paso la arista de menor peso que conecta el árbol con un vértice no visitado.
- **Aplicación en el código**: Se implementa en el método `find_optimal_cabling()` para determinar la forma más eficiente de conectar todas las colonias con el mínimo total de cable. Utiliza una cola de prioridad (heap) para seleccionar eficientemente la siguiente arista de menor peso.
- **Complejidad computacional**: La implementación usando cola de prioridad tiene una complejidad de O((V + E) log V), donde V es el número de vértices y E el número de aristas. Esta es una mejora significativa sobre la implementación básica de $O(V^2)$.

2. Algoritmo del Vecino Más Cercano para TSP

- **Descripción**: Este es un algoritmo voraz que proporciona una solución aproximada al Problema del Viajante (TSP). En cada paso, selecciona la ciudad no visitada más cercana a la ciudad actual como siguiente destino.
- **Aplicación en el código**: Se utiliza en el método `find_delivery_route()` para determinar una ruta eficiente que visite todas las colonias y regrese al punto de inicio. La implementación mantiene un conjunto de ciudades no visitadas y selecciona iterativamente la más cercana.
- **Complejidad computacional**: El algoritmo tiene una complejidad de $O(n^2)$, donde n es el número de ciudades. Aunque no garantiza la solución óptima, proporciona una aproximación razonable en tiempo polinomial para un problema NP-duro.

3. Algoritmo de Ford-Fulkerson para Flujo Máximo

- **Descripción**: Este algoritmo encuentra el flujo máximo en una red de flujo. Funciona aumentando iterativamente el flujo a lo largo de caminos que tienen capacidad residual disponible entre la fuente y el sumidero.
- **Aplicación en el código**: Implementado en `find_max_flow()`, se utiliza para calcular la capacidad máxima de transmisión de datos en la red. Utiliza BFS para encontrar caminos de aumento y actualiza el grafo residual en cada iteración.

- **Complejidad computacional**: La complejidad es $O(VE^2)$ donde V es el número de vértices y E el número de aristas. En la práctica, el algoritmo suele ser más rápido que su cota teórica, especialmente en redes bien estructuradas.

4. Algoritmos de Voronoi

- **Descripción**: Los diagramas de Voronoi dividen un espacio en regiones basadas en la distancia a un conjunto de puntos específicos (centrales en este caso). Cada punto en una región está más cerca de su central que de cualquier otra.
- **Aplicación en el código**: Implementado en `find_voronoi_regions()`, se utiliza para determinar las áreas de servicio óptimas para cada central. La implementación utiliza un enfoque de fuerza bruta simplificado con un grid discreto.
- **Complejidad computacional**: La implementación actual tiene una complejidad de O(kn), donde k es el número de puntos en el grid y n es el número de centrales. Una implementación más sofisticada usando el algoritmo de Fortune tendría una complejidad de O(n log n).

Conclusiones

La implementación combina varios algoritmos clásicos de optimización y teoría de grafos para resolver un problema complejo de diseño de redes. Cada algoritmo fue elegido por su eficiencia y adecuación a un aspecto específico del problema:

- Prim para minimizar el costo total de conexión
- Vecino más cercano para rutas de mantenimiento eficientes
- Ford-Fulkerson para optimizar la capacidad de la red
- Voronoi para una distribución eficiente de las áreas de servicio