ReflexActInt2_A00573664-A01643411-A01644024

Unidad de formación:

Análisis y diseño de algoritmos avanzados (Gpo 604)

Profesor:

Ivan Reyes

Alumnos:

Daniela Rocha Muñoz | A00573664

Santiago López | A01643411

Antoine Ganem | A01644024

Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Guadalajara Av. Gral Ramón Corona No 2514, Col Nuevo México, Código Postal 45201 Zapopan, Jal.

21 Octubre 2024

Nos enfrentamos a muchos desafíos durante el desarrollo del proyecto que pusieron a prueba nuestras habilidades; teníamos que implementar algoritmos complejos que eran cruciales para resolver problemas específicos. Uno de los primeros problemas fue elegir el algoritmo correcto para el cableado de fibra óptica. Debido a que el algoritmo de Kruskal nos ayudó a crear un árbol de expansión mínima que conectaba todas las colonias de manera eficiente, decidimos usarlo. Su complejidad de O(E log E), donde E es el número de aristas, fue razonable para nosotros. Cuando vimos las aristas y sus pesos, fue una luz para nosotros porque así pudimos encontrar rápidamente las conexiones más económicas entre ellas, lo que nos permitió optimizar el uso de recursos.

Resolver el problema del flujo máximo fue lo siguiente en nuestra lista, el algoritmo de Ford-Fulkerson se utiliza para resolver este problema. Al principio parecía complicado, pero una vez que lo dividimos en sus partes más simples, descubrimos que funciona. La complejidad de este algoritmo es $O(E \cdot f)$, donde f es el flujo máximo entre dos nodos en un grafo. Aplicar este algoritmo para maximizar el flujo de datos entre las colonias mapeando nuestras conexiones en un grafo fue interesante, y eso era esencial para que el proyecto funcionara como debía.

Para el problema de la central más cerca, se nos ocurrió usar el algoritmo de Dijkstra. La forma en la que hicimos esto fue en sacar la distancia del nodo inicial a todos los demás nodos usando las propiedades de dijkstra. Para esto usamos un priority queue, en el cual, iba a estar hecha la fila desde la distancia más pequeña a la más grande. El problema de esto es que la librería de C + + hace que la prioridad sea de más grande a más chico. Para eso simplemente usamos otra librería que nos permitió modificar para que fuera como queríamos. En este caso, lo único que tuvimos que hacer fue agregarle otra propiedad a la declaración del priority queue que como se ve en el código, es donde dice greater<DistNodo> (NeilZ, 2023). Para la complejidad del Dijkstra, nos dio O((V + E)log V), esto es por que para inicializar el arreglo de las distancias nos toma V que son los nodos. Para sacar un elemento o actualizar la priority queue nos toma O(logv). Para concluir, el E es el número de conexiones que hay. Juntándolo todo, nos da lo anterior. En el caso de la función para encontrar la central más cercana, simplemente iteramos sobre el arreglo de las centrales por lo cual la complejidad final acaba siendo de O((V + E)log V + C) que C es el número de centrales que hay. Dada esta implementación, podemos observar el poder que tiene el algoritmo de Dijkstra.

Se puede ver claramente que no solo vimos sobre algoritmos, sino también sobre cómo aplicar esas ideas a situaciones reales al pensar en la experiencia, ya que también tenía aplicaciones en el mundo real. Cada uno de los algoritmos que usamos nos dio algo diferente y nos ayudó a enfrentar los problemas de una manera más efectiva.

Citas:

NeilZ. (2023, April 29). *How to change C++ std::priority_queue ordering from the default of Max-heap to min-heap?*. Medium. https://medium.com/@aiii/how-to-change-c-std-priority-queue-ordering-from-the-default-of-max-heap-to-min-heap-e95d77f7bdd4