Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Querétaro

Análisis y diseño de algoritmos avanzados TC2038.601

Actividad Integradora 2

Presenta:

María Fernanda Moreno Gómez | A01708653
Uri Jared Gopar | A01709413
José Ricardo Rosales Castañeda | A01709449

Querétaro, Querétaro 18 de noviembre, 2023

Índice

Alta demanda para los Proveedores de Servicios de Internet (ISP)

Contexto

En el año 2020 una pandemia sacudió al mundo entero, situación por la que se optó parar gran parte de las labores que tenían que ver con salir al exterior, dando como opción el home office y las clases virtuales a los alumnos, de manera que se disminuyera en medida de lo posible todo contacto con otras personas. Al tener a todas las personas encerradas en sus casas, la demanda de Internet se incrementó, pues las familias al tener miembros con distintas actividades que requerían Internet en una misma casa, se necesitaba el Internet cada vez más urgente, pues la transmisión de datos por Internet era enorme.

Por ello, se planea mejorar el servicio de Internet en una población pequeña, sin embargo, se presentan algunas dificultades como el cablear los puntos más importantes, de manera que se necesite la menor cantidad posible de fibra óptica, así como el mejorar el camino para que la persona que visite todos los puntos de red, lo haga con la menor distancia regresando a su origen, y que además, se calcule la cantidad máxima de información que puede pasar de un nodo a otro y la factibilidad de conectar un nuevo punto en el mapa.

Reto

Se debe hacer un programa que, dado un archivo de entrada que contiene la información de un grafo en forma de una matriz de adyacencia, donde el peso de cada arista es la distancia en kilómetros entre colonia y colonia por donde se puede cablear, se calcule cuál es la forma óptima de cablear con fibra óptica conectando las colonias para que todas estén conectadas.

También, se necesita que una persona visite todas las colonias para ver sobre todo lo relacionado con la infraestructura de red, por lo que se necesita la ruta más corta posible para que, esta persona visite cada colonia una vez y al final, regrese a su colonia de origen, por lo que el programa debe desplegar dicha ruta corta, tomando como “A” la primera colonia que visitará y como “B” la segunda colonia que visitará.

Asimismo, el programa debe leer otra matriz de tamaño N x N datos que representen la capacidad máxima de transmisión de datos entre la colonia “i” y la colonia “j”, esto debido a que se pueden generar interferencias debido a los campos electromagnéticos, por lo que se necesita conocer el flujo máximo de información del nodo inicial al final, esto se debe desplegar de igual manera.

Por último, teniendo la ubicación geográfica de centrales a las que se pueden conectar nuevas casas, la empresa quiere contar con una forma de decidir, dada una nueva contratación del servicio, cuál es la central más cercana geográficamente a esa nueva contratación (No necesariamente hay una central por cada colonia). Se pueden tener colonias sin central y colonias con más de una central.

Recursos

Entrada

- Un número entero N que representa el número de colonias en la ciudad

- Una matriz cuadrada de N x N que representa el grafo con las distancias en kilómetros entre las colonias de la ciudad
- Una matriz cuadrada de N x N que representa las capacidades máximas de flujo de datos entre colonia “i” y colonia “j”
- Una lista de N pares ordenados de la forma (x,y) que representaban la ubicación en un plano coordenado de las centrales, junto con la ubicación de la nueva central.

Salida

1. Forma de cablear las colonias con fibra (lista de arcos de la forma (A,B))
2. Ruta a seguir por el personal que reparte correspondencia, considerando inicio y fin en la misma colonia
3. Valor de flujo máximo de información del nodo inicial al nodo final
4. La salida será la distancia más costa entre dos puntos: el de la ubicación de la nueva central con respecto al más cercano

Puntos importantes

- Buscar un algoritmo que, dependiendo de cada peso de la arista, pueda cablear conectando cada colonia con el menor uso posible de fibra óptica
- Buscar un algoritmo que de la rita más corta en la que el técnico pueda visitar todas las colonias desde el punto A al punto B.
- Buscar un algoritmo que pueda obtener el flujo máximo de información del nodo al inicial
- Buscar un algoritmo que nos permita conocer la distancia más corta entre la nueva contratación del servicio y una central.

Propuesta: Algoritmos utilizados

Algoritmo de Dijkstra

El algoritmo de Dijkstra es un método utilizado para encontrar el camino más corto entre nodos en un grafo, ya sea un grafo con peso (es decir, un grafo cuyas aristas tienen cierto peso, que es nuestro caso) o un grafo sin peso. Su complejidad algorítmica de nuestra implementación es de O(n^2), donde n es el número de nodos o colonias en el grafo. En O(n^2) porque el ciclo principal se hace n-1 veces, además de que en la función para tener la distancia mínima, esta tiene una complejidad de O(n) para recorrer todos los nodos con la distancia mínima, y, en el ciclo principal, hay otro ciclo anidado que recorre los nodos para actualidad la distancia, por lo que O(n) x O(n) = O(n^2).

Usamos Dijkstra debido a que, se requiere encontrar la forma más eficiente de poner fibra óptica entre las distintas colonias, Dikstra al trabajar con grafos con peso, es ideal para nuestro escenario, ya que se puede calcular eficientemente el camino más corto entre un nodo inicial y todos los demás nodos del grafo, por lo que podemos utilizar esto para que se use la menos cantidad posible de fibra óptica

El funcionamiento con el algoritmo es el siguiente:

1. Se crea un vector “dist” con distancias inicializadas a INT_MAX (infinito), menos el nodo fuente “src” que se inicializa en 0.

2. Se crea un vector “inMST” para localizar sin un nodo ya ha sido incluido en el árbol de camino mínimo (MST).
3. Se selecciona el nodo más cercano, de manera que, la función “minDistance” recorre todos los nodos y selecciona el nodo con la distancia mínima que aún no es´ta incluido en “inMST”.
4. Una vez que se selecciona el nodo más cercano, el algoritmo actualiza las distancias de los nodos adyacentes a este nodo, de modo que si el nuevo camino calculado a un nodo adyacente es más corto que la distancia conocida anteriormente, se actualiza, esto se repite n-1 veces a los nodos, menos al nodo “src” (pues de este partimos)
5. Una vez que se terminó de actualizarse las distancias, la función “dijkstra” devuelve el vector “dist” que contiene las distancias más cortas desde el nodo fuente a todos los demás nodos.
6. Finalmente, se imprime el resultado con la función “printMatrix”, que utiliza “dijkstra” para calcular y mostrar las distancias más cortas desde cada nodo a todos los demás nodos, dando así una matriz completa con las distancias más cortas.

Algoritmo de Nearest Neighbor

El algoritmo de Nearest Neighbor lo que hace es seleccionar la ruta más corta para una persona que debe de visitar cada nodo una vez exactamente y regresar al nodo inicial. Lo que hace es iniciar de un nodo, y, en cada paso, se va al nodo no visitado más cerca hasta que todos los nodos hayan sido visitados. La complejidad algorítmica de nuestra implementación es O(n^2), donde n nuevamente es el número de nodos o colonias: El primer ciclo se repite n veces para visitar cada colonia o nodo, mientras que el ciclo anidado, busca el nodo más cercano no visitado entre todos los n nodos o colonias, por lo que ambos ciclos tienen una complejidad de O(n), por lo que O(n) x O(n) = O(n^2).

Decidimos usar Nearest Neighbor debido a que es fácil de implementar y entender, además de que es ideal para planificar las rutas que el técnico va a hacer para visitar las colonias, donde, si alguna colonia es añadida, este algoritmo rápidamente puede calcular la nueva ruta, pues es rápido para grafos de tamaño moderado, además de que es un algoritmo que es rápido para obtener las soluciones.

El funcionamiento con el algoritmo es el siguiente:

1. Se inicia el nodo 0, marcando este nodo inicial como visitado y se agrega al recorrido “tour”.
2. Se busca el vecino más cercano, de manera que, en cada iteración, el algoritmo busca entre los nodos no visitados el que está más cerca del nodo actual.
3. Una vez que se encuentra el nodo más cercano, se actualiza el nodo actual a este nuevo nodo, se marca como visitado y se agrega a “tour”.
4. Una vez que se terminan de visitar todos los nodos, el algoritmo agrega el nodo inicial al final del recorrido para completar el ciclo.
5. Finalmente, se imprime el recorrido y el costo con la función “printTour”, que calcula y muestra el recorrido completo y su costo total

Como podemos notar, Nearest Neighbor ofrece simplicidad y eficacia, para una solución rápida de la planificación de rutas más cortas.

Algoritmo de Ford-Fulkerson

El algoritmo Ford-Fulkerson es un método para encontrar el flujo máximo en una red de flujo. Se usa normalmente en redes, lo cual es perfecto para nuestro proyecto, pues buscamos maximizar el flujo de datos entre dos puntos en una red. Este algoritmo busca aumentos de flujo a lo largo de los caminos desde el nodo fuente al nodo final hasta que ya no se puedan encontrar más caminos en los que el flujo pueda ser aumentado. La complejidad algorítmica de nuestra implementación es O(n^2), pues tenemos ciclos anidados para recorrer los nodos y los caminos para buscar el flujo máximo.

Decidimos usar Ford-Fulkerson porque es un algoritmo bueno para calcular la capacidad máxima de flujo de datos en la red, además de que como anteriormente y recientemente lo implementamos en clase, lo tenemos bastante fresco, además de que es bastante eficiente para redes de tamaño moderado, pues es bastante flexible.

El funcionamiento con el algoritmo es el siguiente:

1. Se crea un grafo residual "rGraph" que es una copia del grafo original.
2. Luego, se usa BFS para encontrar los caminos aumentantes desde del nodo inicial hasta el final, donde, si no se encuentra un camino, se termina el algoritmo.
3. Cuando se encuentra un camino aumentante, se actualiza el flujo a lo largo de ese camino en el grafo desigual, disminuyendo la capacidad en el camino directo y aumentándola en el inverso.
4. Finalmente, se calcula el flujo máximo sumando los flujos de todos los caminos aumentantes encontrados.

Algoritmo de Distancia Euclidiana

Este algoritmo se utiliza para calcular la distancia más corta entre dos puntos en un plano o un espacio euclidiano, o sea, la longitud de un segmento de recta que conecta 2 puntos. La fórmula es la siguiente:

$$d_{(p1,p2)} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

La complejidad algorítmica es O(n), donde n es el número de puntos en una lista, esto es porque se deben de recorrer todos los puntos para encontrar el más cercano.

Escogimos este algoritmo por la simplicidad para encontrar la distancia entre dos puntos, que, además, es bastante eficiente, pues no se tienen que hacer tantas operaciones, además de que la distancia euclidiana es un estándar que se ha seguido mundialmente para obtener la distancia entre dos puntos.

El funcionamiento con el algoritmo es el siguiente:

1. Se toman las coordenadas de dos puntos (x1, y1) y (x2, y2)
2. Se calcula la distancia entre dos puntos utilizando la fórmula de distancia euclidiana que mostramos anteriormente.
3. Se devuelve la distancia obtenida con esta fórmula.

4. Posteriormente, para encontrar el punto más cercano, se recibe un vector de puntos, donde cada uno es un par de coordenadas “x” y “y” y las coordenadas de otro punto
5. Se inicializa la variable para almacenar el índice del punto más cercano “closestPoint” y una variable para la distancia mínima encontrada “minDistance”, que se inicializa en infinito.
6. Se iteran los puntos en el vector, para calcular la distancia entre ese punto y el punto de referencia
7. Se compara la distancia con “minDistance”, donde, si es menor, se actualiza “minDistance” y se guarda el índice del punto actual como “closestPoint”.
8. Finalmente, se devuelve el índice del punto más cercano al punto de referencia y se imprime.

Link al video

En YouTube

https://youtu.be/gYU-cDbRbx8

En Drive

https://drive.google.com/file/d/1UCjy0iflMMIAgFad4F4b4Y0VyBhVcb2D/view?usp=sharing

Reflexión

Después de este reto, hemos logrado un mejor entendimiento y práctica de cómo es que los algoritmos que vemos en clase, pueden aplicarse en la vida real, y no en un ambiente aislado, sino en algo donde hemos estado en bastante contacto, que es en el tema de redes. A través de este reto, pudimos enlazar los conocimientos teóricos vistos en clase con la parte práctica de ponerlos a prueba y del trabajo en equipo, que consideramos, todo esto es crucial para nuestro futuro laboral, pues son competencias que nos servirán para ello.

La manera en la que se decidió implementar Dijkstra es por el conocimiento que ya tenemos desde anteriores materias sobre cómo y para qué sirve este algoritmo, por lo que la aplicación acá iba enfocada en la optimización de la infraestructura de red usando el menos cable de fibra óptica posible. Este escenario nos ayudó a poder adaptar este algoritmo que ya lo hemos visto muchas veces para solucionar un problema como es el del alta demanda de internet, y cómo es que necesitamos de algoritmos en nuestra vida cotidiana.

Pasando con el algoritmo de Nearest Neighbor, decidimos que sería el ideal para la planificación de la ruta que va a tomar el técnico porque, se nos hizo bastante fácil de entender y de implementar, además de que es algo que podría ser aplicable en la vida real, por lo que se nos hizo buena forma de abordar este problema.

La implementación del algoritmo de Ford-Fulkerson sin duda se nos hizo más fácil programarlo que hacerlo en papel, ya que teníamos acá la ventaja de hacer un BFS para hacer un mapeo de los vecinos y así recorrerlos mejor que si tuviéramos que estar viendo los vecinos, sumando y restando cargas, donde la probabilidad de error era alta a comparación de tener un programa que haga los cálculos sin error para ti. Además, ya

teníamos la ventaja que, como fue uno de los últimos que vimos y trabajamos, lo teníamos fresco, así que fue fácil de implementar.

Finalmente, se tiene el de la Distancia Euclidiana, que es esa fórmula que desde que vimos geometría nos han estado enseñando, por lo que es muy simple de implementar y fue chistoso en cierto modo porque incluimos conocimientos de matemáticas que habíamos visto en años. A pesar de ser muy sencillo, es bastante preciso, y nos ayudó bastante para encontrar las distancias y así poder conocer la central más cercana al punto de contratación, que es algo que todos queremos, el tener lo más cercano para que sea rápida la instalación.

Integrando todo, este proyecto nos aportó a cada uno de nosotros una puerta para poner en práctica nuestras competencias, de manera que en la práctica pudimos mejorar nuestras habilidades, proponer ideas para resolver problemas, trabajo en equipo y escoger aquellas soluciones que sean más factibles para el proyecto, por lo que fue una experiencia bastante integradora y buena al poder trabajar en un escenario de la vida real.

Referencias

Coding games and programming challenges to code better. (s/f). CodinGame. Recuperado el 18 de noviembre de 2023, de https://www.codingame.com/playgrounds/7656/los-caminos-mas-cortos-con-el-algoritmo-de-dijkstra/el-algoritmo-de-dijkstra

Comprender el análisis de distancia euclidiana—ArcMap. (s/f). Arcgis.com. Recuperado el 19 de noviembre de 2023, de https://desktop.arcgis.com/es/arcmap/latest/tools/spatial-analyst-toolbox/understanding-euclidean-distance-analysis.htm

Dijkstra's Algorithm. (s/f). Programiz.com. Recuperado el 19 de noviembre de 2023, de https://www.programiz.com/dsa/dijkstra-algorithm

Fischer, Q. (s/f). *Ford-Fulkerson algorithm*. Tum.de. Recuperado el 19 de noviembre de 2023, de https://algorithms.discrete.ma.tum.de/graph-algorithms/flow-ford-fulkerson/index_en.html

Ford-Fulkerson algorithm. (s/f). Programiz.com. Recuperado el 19 de noviembre de 2023, de https://www.programiz.com/dsa/ford-fulkerson-algorithm

Ford-Fulkerson algorithm for maximum flow problem. (2013, julio 3). GeeksforGeeks. https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/

GraphEverywhere, E. (2019, octubre 29). *Algoritmo de distancia euclidiana - Algoritmos de Grafos*. GraphEverywhere; Graph Everywhere SL. https://www.grapheverywhere.com/algoritmo-de-distancia-euclidiana/

K-nearest neighbor(KNN) algorithm. (2017, abril 14). GeeksforGeeks. https://www.geeksforgeeks.org/k-nearest-neighbours/

Shubham Agrawal, S. (2016, marzo 21). *Dijkstra's Shortest Path Algorithm using priority_queue of STL*. GeeksforGeeks.

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/

Subramanian, D. (2019, junio 8). *A simple introduction to K-nearest neighbors algorithm*. Towards Data Science. https://towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e