Reflexión Actividad Integradora 2

Fernando Doddoli Lankenau - A00827038
Profesor Frumencio Olivas Alvarez

Para esta actividad integradora se pidió que como equipo, resolvieramos una problemática con el objetivo de consolidar nuestro conocimiento y aplicar lo aprendido en la segunda parte del curso. Esto incluye manejo de strings, grafos, diseño de algoritmos avanzados, y geometría computacional.

En resumen, la problemática nos pidió leer un archivo de entrada con información de coordenadas X,Y de colonias y centrales, y una matriz de flujos entre las colonias. Después, utilizando las coordenadas de las colonias, se nos pidió generar una matriz de adyacencias representando un grafo ponderado no dirigido. Donde las distancias entre las colonias son el peso calculado de forma euclidiana. Finalmente, con esta información se nos pidió hacer las siguiente 5 tareas:

- Imprimir la matriz de adyacencias con las distancias entre las colonias
- Imprimir la forma más óptima de conectar las colonias
- Imprimir la ruta de menor costo para visitar todas las colonias desde una colonia origen y regresar a la misma al finalizar
- Imprimir el flujo máximo entre cada colonia i y la colonia j utilizando la matriz de flujo entre colonias
- Imprimir la central y colonia más cercana geográficamente a una nueva contratación

Para la tarea 1, que fue imprimir la matriz de adyacencias con las distancias entre las colonias, inicialmente llenamos la matriz utilizando una función que calcula la distancia entre dos colonias utilizando sus coordenadas con la ecuación euclidiana. La cual es: sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2)). Después, para imprimir esta matriz, utilizamos dos for loops, así logrando una complejidad O(N^2).

Para la tarea 2 imprimimos la forma más óptima de conectar las colonias. Esto lo logramos utilizando el algoritmo de Prim. Un algoritmo greedy donde utilizamos un minimum spanning tree para desplegar la forma de menor costo de conectar las colonias. para imprimir árbol de expansión con peso mínimo. La idea detrás del algoritmo de Prim que implementamos es la siguiente: Después de crear el minimum spanning tree, mantenemos 2 vectores de colonias que representan las colonias ya incluidas en el MST y las colonias aún no incluidas. De este modo, iteramos la matriz de adyacencias y consideramos todas las conexiones de las colonias y seleccionamos la distancia mínima. Esta colonia con menor costo se incluye en el árbol. Finalmente, se despliega la lista de arcos representando pares para cada colonia y la longitud,

mostrando la forma más óptima de conectarlas. La complejidad de este algoritmo terminó siendo O(V^2).

Para la tarea 3 se nos pidió imprimir la ruta de menor costo para visitar todas las colonias desde una colonia origen y regresar a la misma al finalizar. Este problema está basado en el problema del vendedor viajero. La solución implementada de este problema es muy interesante, ya que utiliza una función recursiva auxiliar, y logra una complejidad de O(N^2). El algoritmo funciona de la siguiente manera: primero generamos la ruta con el árbol de expansión mínima utilizando la función auxiliar recursiva inOrder, así filtrando las distancias y manteniendo las menores. Después, iteramos la matriz, y para cada colonia, creamos un vector para representar el current path y otro vector para representar los nodos visitados, al igual que una variable para representar el costo actual. Después iteramos en un for brincando los nodos repetidos en el walk (vector calculado con la función auxiliar recursiva) y actualizamos el current path. Después, cerramos el ciclo regresando al punto de salida. Después, después sumamos la distancia de las colonias elegidas dentro de currentPath (iteramos currentPath) y almacenamos el costo en la variable del costo actual. Finalmente, actualizamos la variable de menor costo si el costo actual es menor. De la misma manera, si esta condición se cumple, entonces el vector de bestPath se actualiza con los valores de currentPath. Al finalizar de iterar la matriz, finalmente imprimimos los resultados.

Para la tarea 4 imprimimos el flujo máximo entre cada colonia i y la colonia j utilizando la matriz de flujo entre colonias. Logramos hacer esto mediante la implementación del algoritmo de Ford-Fulkerson. El cual funciona de la siguiente manera: primero inicializamos las variables de trabajo, un vector de vectores para guardar los valores residuales dos vectores, y un vector para guardar los nodos ancestros. El vector de vectores de nodos residuales se inicializa como la matriz de flujo. Y el vector de nodos ancestros se inicializa de tamaño de la matriz con valores en 0. Después se implementó un while loop que corre mientras exista un path. Esto se decide utilizando con una función auxiliar que utiliza una pila y un vector de visitados y itera los residuales y regresa true si existe un camino del nodo inicial al final. En efecto, mientras esta condición se cumple, por cada nodo X desde el nodo final, hasta que X no sea igual al nodo inicial y con cada iteración cambiando X por el valor de ancestros en el índice X, actualizamos el valor del flujo con el mínimo del valor actual y el valor residual en [U][X], donde U es el valor de ancestros en el índice X. Después, volvemos a iterar con las mismas condiciones que el for loop anterior pero ahora actualizamos los valores residuales. Finalmente, le sumamos a el flujo máximo el valor del flujo. Al finalizar el ciclo desplegamos el flujo máximo. Cabe recalcar que el algoritmo de Ford-Fulkerson originalmente cuenta con una complejidad que depende de la cantidad de caminos que componen al flujo máximo F O(EF). Sin embargo, el uso de una matriz y la función auxiliar BFS hace que la complejidad termine en O(N^5).

Finalmente, para la tarea 5 se nos pidió imprimir la central y colonia más cercana geográficamente a una nueva conexión. Logramos resolver este problema implementando

conceptos de geometría computacional. Utilizamos 2 ciclos anidados, donde uno checa la colonia más cercana y otro la central más cercana, a la nueva conexión. Esto para cada una de las nuevas conexiones. De este modo, en el ciclo para checar la central más cercana a la nueva conexión hacemos uso de una función auxiliar que calcula la distancia entre esta nueva conexión y las centrales disponibles utilizando una ecuación euclidiana. Así almacenando el mínimo en una variable y actualizando cuando la distancia calculada es menor al mínimo actual; si esto se cumple entonces se actualiza también la central más cercana. De igual forma, algo muy similar se utilizó para obtener la colonia más cercana a la nueva conexión. En el ciclo para checar la colonia más cercana a la nueva conexión también hacemos uso de una función auxiliar que calcula la distancia entre esta nueva conexión y las colonias disponibles utilizando una ecuación euclidiana. Así almacenando el mínimo en una variable y actualizando cuando la distancia calculada es menor al mínimo actual; si esto se cumple entonces se actualiza también la colonia más cercana. Finalmente, para cada nueva conexión, desplegamos la colonia más cercana y su respectiva distancia, al igual que la central más cercana y su respectiva distancia. La complejidad de este algoritmo es O(N^2) debido a los dos ciclos anidados.