Análisis Proyecto 2

Integrantes:

Ignacio Chaparro - 202220577

Juan Diego Osorio – 202220148

Universidad de Los Andes,

Diseño y Análisis de Algoritmos

Explicación de la Solución

Para solucionar este problema, se parte de que un elemento solo puede conectarse con otro (sin considerar la cadena de átomos libres) si estos elementos contienen al menos un átomo en común. Por ejemplo, el elemento (6,3) solo podría conectarse con otro elemento que tenga un átomo 6 o 3, como lo es el elemento (3, 1), con el cual se podría conectar mediante el átomo 3 y con una cadena de átomos libres que todavía no conocemos. Una vez que sabemos esto, podemos verificar si un elemento puede conectarse con algún otro, buscando átomos en común con los demás elementos. Si este elemento es encontrado, significa que si se pueden conectar y pasamos a la siguiente fase del algoritmo. Previamente, se construyó un grafo de átomos libres, en donde cada nodo es un átomo, y cada arco es el LTP entre los átomos que conecta. Ahora, en esta fase del algoritmo, partimos desde el átomo que encontramos en común entre los dos elementos y ejecutamos el algoritmo de Dijkstra sobre el grafo de átomos libres, buscando el camino más corto entre el átomo en común de los elementos y su opuesto. Esto se hace, ya que el elemento al que se desea conectar solo va a poder conectarse mediante un enlace ideal, ósea con un átomo de su misma masa, pero de carga opuesta.

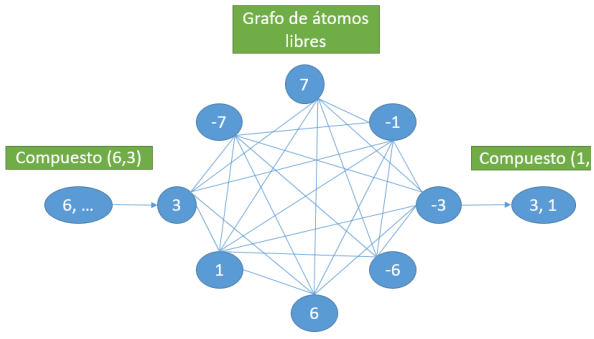
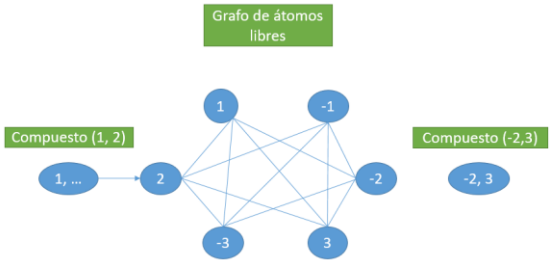


Ilustración 1: Grafo de ejemplo elemento 6,3 y 3,1

Por ejemplo, en la ilustración 1, podemos ver el caso de conectar el elemento (6, 3) con (3, 1). Comenzamos a ejecutar Dijkstra desde el nodo 3, que sería el inicio de la cadena y paramos cuando encuentre su opuesto. Esto nos retornará el camino con menos costo para conectar estos dos elementos.



Por otro lado, tenemos este ejemplo, en donde se busca conectar el elemento (1,2) con (-2,3). Al tener un átomo de la misma masa, pero carga diferente no debería poder conectarse, sin embargo, ejecutaremos el algoritmo para mostrar que nuestro algoritmo da la respuesta indicada. En este caso, la cadena comienza en el átomo 2, y comienza a buscar el camino más corto hacia su opuesto, sin embargo, este opuesto no va a poder conectarse con el elemento ya que tienen la misma carga, por lo tanto, no encontrará ningún camino. Para optimizar estos casos en donde no se va a poder encontrar un átomo con la misma carga y misma magnitud en otro elemento, utilizamos una tabla de hash en donde como llave almacenamos el átomo, y como valor su frecuencia. Una vez que sabemos todos los átomos (de los elementos, no átomos libres) presentes en el caso de prueba, verificamos su concurrencia, y si más de 3 frecuencias son impares, podemos concluir que no se va a poder formar una cadena con esos elementos. Esto se hace ya que máximo, puede haber 2 elementos con átomos que no se puedan conectar a ningún otro elemento, el del inicio de la cadena y el del final de esta. Esto ayudó a optimizar los casos en los que no se pueden formar cadenas y logra retornar una respuesta “NO SE PUEDE” en tiempo O(V) ya que con solo cargar los átomos de los elementos se podría saber si no se puede formar una cadena.

Como alternativas de solución al problema, surgió la solución mediante el uso del algoritmo de camino Euleriano. Este algoritmo se usaría para poder saber si los

elementos se podrían conectar con otros elementos mediante átomos en común, pasando solo una vez por cada elemento, algo parecido a lo que se hace en nuestro planteamiento, pero de manera más eficiente. Con el uso de este camino, se podría determinar con una sola iteración, todas las conexiones entre todos los elementos, en vez de buscar una conexión cada vez que se encuentra un nuevo elemento. Esto habría ayudado a reducir la complejidad de nuestro algoritmo y poder optimizarlo. No se implementó esta solución por cuestiones de tiempo, ya que dimos con ella después de haber implementado nuestro algoritmo actual y no lográbamos implementarla antes de la entrega.

A pesar de demorarse más de 2 minutos con casos superiores a 1000 elementos, nuestro algoritmo es capaz de solucionar el problema correctamente, sin embargo, puede llegar a ser lento ya que por cada elemento se va a ejecutar Dijkstra para encontrar el camino más corto hacia el otro elemento con el mismo átomo.

Análisis de Complejidad Espacial y Temporal

Análisis de Complejidad Espacial

Para analizar la complejidad espacial de este algoritmo, se debe considerar el grafo inicial y el algoritmo Dijkstra que se usa para resolver el problema. Al haber implementado el grafo de átomos libres con una lista de adyacencias, la complejidad espacial de este grafo seria de $O(V + E)$. Por otro lado, nuestro algoritmo Dijkstra, está implementado con un heap, en donde se almacenan los nodos del grafo y como resultado tendría una complejidad espacial $O(V)$. Añadido a esto, tenemos dos estructuras auxiliares. La primera es una tabla de hash, que se va a encargar de almacenar el costo mínimo para llegar del nodo "Start" a un nodo en particular, obteniendo así una complejidad espacial $O(V)$ para esta estructura. La segunda estructura también es una tabla de hash, que se va a encargar de almacenar el camino que se recorrió para llegar, en el menor costo posible, a ese nodo en particular, obteniendo una complejidad espacial $O(V)$ para esta estructura.
Por ende, si analizamos el crecimiento espacial total de nuestro algoritmo obtenemos como resultado una complejidad $O(3V + V + E)$, siendo $O(3V)$ la complejidad de Dijkstra y $O(V+E)$ la complejidad del grafo. Sin embargo, solo nos interesa analizar el crecimiento asintótico, por ende, ignoramos las constantes y concluimos que la complejidad espacial de este algoritmo seria de $O(V + E)$.

Análisis de Complejidad Temporal

Al analizar la complejidad temporal hay que considerar 3 momentos de nuestro algoritmo, cuando ejecuta Dijkstra, cuando confirma si se puede formar una cadena entre 2 elementos y cuando "arregla" el camino para que su formato coincida con el formato solicitado. Dijkstra al estar implementado con un heap, tendrá una complejidad temporal $O((V + E) \log V)$. Ahora analizaremos la función principal de nuestro algoritmo, la búsqueda de un elemento con el cual se pueda conectar. Este busca para cada átomo si existe otro elemento con el mismo átomo (complejidad O(V)) con el que se podrá conectar. Si encuentra este átomo, va a ejecutar Dijkstra entre este nodo y su opuesto para poder saber el camino más corto entre esos dos elementos, obteniendo así una complejidad $O(V((V + E) \log V))$ ya que, en el peor caso, para cada elemento existente vamos a encontrar otro elemento con el cual conectar y con el cual se ejecutará Dijkstra. Por último, tenemos la función de revisar la cadena, y ajustar los elementos si no coinciden con la cadena. Por ejemplo, si encuentra la cadena 7, 2, -3, (4,3), sabemos que debemos invertir este elemento para que los átomos 3 y –3 coincidan. Esta

función tiene una complejidad de *O(N)*. En el peor caso, va a tener que recorrer las tuplas que contienen los caminos.
Teniendo en cuenta estas tres funciones, la complejidad final de nuestro algoritmo seria *O(V((V+E) log V) + N)*, pero como solo nos interesa el comportamiento asintótico, su complejidad seria *O(V((V+E) log V))*.

Respuesta a los Escenarios de Comprensión de Problemas Algorítmicos

ESCENARIO 1: Se admiten como respuesta elementos en los que un elemento fundamental aparezca más de una vez.

Con este nuevo escenario surge un problema, y no siempre se podrá concluir que no se puede formar la cadena al no encontrar un camino Euleriano entre los elementos. Esto se debe a que nosotros nos estábamos basando en el principio de que, si el átomo X no puede conectarse con ningún otro átomo en un elemento Y, no se podrá formar la cadena. Con este nuevo escenario no sería cierto, ya que tendría que considerar todos los elementos usados antes y verificar que no se puede conectar con ninguno antes de retornar false. Para poder adaptar nuestra solución a esto, una vez que se verifique que no se puede generar un camino Euleriano con los elementos no usados, toca verificar uno por uno con los elementos repetidos y los no usados.

ESCENARIO 2: Se admiten como respuesta elementos que incluyan directamente enlaces toll.

Con este nuevo escenario surge un nuevo problema, y es que los enlaces toll no tiene costo, son enlaces ideales, por ende, tocaría tener en cuenta estos dentro del camino con menos costo posible. Para esto, primero se ejecutaría el camino Euleriano por los elementos, buscando si es posible conectarlos mediante enlaces toll. Esto se hace primero ya que los enlaces toll no tienen costo, por ende, darían el camino más barato. Si no es posible conectar todos los elementos mediante estos enlaces, obtenemos los elementos que no se lograron conectar y ejecutamos nuestro algoritmo ya planteado, buscando conectar los elementos mediante átomos libres.