El triángulo

María de Lourdes Choy Fernández
Alejandro Yero Valdéz
Leismael Sosa

Contents

1 Problema

El Triángulo
Leismalel estaba un día practicando con su instrumento favorito: El triángulo. El triángulo es un instrumento musical tan espectacular que sus notas se escriben como números enteros. Este día Leismalel se propuso componer una canción de una forma bastante peculiar. Tomó n enteros (notas) aleatorios y los escribió en una lista a. Una melodía válida es una subsecuencia de a en la que todos sus números adyacentes cumplen que:

- Se diferencian en 1.
- Son congruentes módulo 7.

La canción de Leismalel debe contener exactamente 4 melodías que cumplan con lo anterior y además no se intercepten entre sí. Ayude a Leismalel encontrando una canción que maximice las notas usadas.

2 Modelación

El objetivo es maximizar la cantidad de subsecuencias que se pueden extraer de la lista de notas a sin que estas subsecuencias se intercepten entre sí. Para ello, se propone la construcción de un grafo dirigido G, donde todos los elementos a_i son representados como vértices. La idea detrás de esta construcción es que cada elemento por sí solo puede ser una subsecuencia válida de a.

Cada vértice a_i podrá formar una subsecuencia con otros vértices a_j que aparezcan en posiciones posteriores a i en la lista, siempre que se cumplan las condiciones del problema. Es decir, los vértices a_i y a_j estarán conectados por una arista dirigida si:

- $a_i < a_j$ (i.e., a_j está en una posición posterior en la lista),
- $|a_i - a_j| = 1$ o $(a_i - a_j)\%7 = 0$ (las diferencias entre a_i y a_j cumplen con las restricciones del problema).

Definamos formalmente los componentes de nuestro modelo:

- G: Grafo que modela el problema.
- $\forall i, j \in a$: Existe una arista $i \to j \in E(G)$ si $i < j$ y se cumple que $|a_i - a_j| = 1$ o $(a_i - a_j)\%7 = 0$.
- $cap(u, v)$: Función que asigna capacidad a las aristas de G.
- $cost(u, v)$: Función de costo asociada a las aristas de G, que representa el costo de tomar la arista (u, v) en la subsecuencia.

Para resolver este problema, implementaremos un algoritmo de flujo máximo con costo mínimo, donde el flujo a maximizar será 4, ya que se requieren exactamente 4 subsecuencias válidas. El grafo original será transformado en una red de flujo, de la siguiente forma:

- $G_f = \langle V, E \rangle$: Una red de flujo derivada del grafo G.
- s: Fuente $\in V(G_f)$.
- t: Receptor $\in V(G_f)$, hacia el cual debe llegar un flujo máximo de 4, representando las 4 subsecuencias deseadas.
- $\forall v_i \in V(G)$, se crean dos nodos $v_{i1} \in G_f$ y $v_{i2} \in G_f$, con una arista $v_{i1} \to v_{i2} \in E(G_f)$ tal que $cap(v_{i1}, v_{i2}) = 1$ y $cost(v_{i1}, v_{i2}) = -1$. Esto representa la decisión de incluir el elemento v_i como parte de una subsecuencia.
- $\forall v_i \in V(G)$, se conecta la fuente $s \to v_{i1} \in E(G_f)$, ya que cualquier elemento puede ser el inicio de una subsecuencia. La capacidad de esta arista es $cap(s, v_{i1}) = 1$ y el costo es $cost(s, v_{i1}) = 0$.
- $\forall i, j$ tal que $v_i \to v_j \in E(G)$, existe una arista $v_{i2} \to v_{j1} \in E(G_f)$, con $cap(v_{i2}, v_{j1}) = 1$ y $cost(v_{i2}, v_{j1}) = 0$, lo que permite continuar la subsecuencia.
- $\forall v_i \in V(G)$, se tiene una arista $v_{i2} \to t \in E(G_f)$ para completar la subsecuencia, con $cap(v_{i2}, t) = 1$ y $cost(v_{i2}, t) = 0$.

En la red residual, para cada arista de retroceso $u \to v$, se cumple que $cap_0(v, u) = 0$ y $cost(v, u) = -cost(u, v)$.

La intuición detrás de esta modelación es que cada vez que se utiliza una nota en un camino aumentativo, se disminuye el costo del camino, lo que obliga al algoritmo a encontrar el flujo de 4 con el menor costo posible, es decir, maximizando la cantidad de notas utilizadas en las subsecuencias.

Una diferencia importante con los algoritmos clásicos de flujo máximo es que, en lugar de buscar caminos aumentativos de longitud mínima (utilizando BFS), aquí buscamos caminos de costo mínimo, para lo cual utilizamos el algoritmo de Bellman-Ford, ya que nuestro grafo puede tener aristas con costo negativo.

Cabe mencionar que el grafo original G es un **DAG** (grafo acíclico dirigido), ya que todas las aristas van de notas con menor índice a notas con mayor índice en la lista. Sin embargo, en la red residual, se generan ciclos debido a las aristas de retroceso. A pesar de esto, se garantiza que no existen ciclos de costo negativo en la red residual, lo que nos permite usar Bellman-Ford de manera segura para encontrar los caminos de costo mínimo.

3 Solución

La implementación del algoritmo sigue estos pasos:

3.1 Construcción del Grafo

Dado que las aristas en el grafo solo permiten conexiones unidireccionales entre los vértices correspondientes a los elementos a_i y a_j de la secuencia, siempre bajo la restricción de que $i < j$, resulta conveniente almacenar las posiciones de estos vértices de manera estructurada, utilizando un diccionario o tabla hash. Esta representación nos permite gestionar de manera eficiente las relaciones entre los vértices y facilita la diferenciación entre las aristas directas, que representan las relaciones válidas entre los elementos de la secuencia, y las aristas de retroceso, que serán introducidas posteriormente en la construcción de la red residual.

La construcción del grafo residual requiere un enfoque sistemático que garantice la correcta representación de todas las posibles transiciones entre vértices, respetando las condiciones impuestas por el problema. En este caso, se realiza una iteración inversa sobre la lista de notas, lo que nos permite, para cada vértice, analizar de manera eficiente todas las conexiones hacia vértices situados en posiciones superiores. Este procedimiento involucra la creación de pares de vértices duplicados por cada elemento en la secuencia original, conectando cada vértice inicial a su duplicado, y posteriormente añadiendo aristas hacia aquellos vértices que cumplan las restricciones de diferencias de valor $+1$, -1 o congruentes modulo 7.

Es crucial destacar que la iteración inversa es particularmente efectiva en este contexto, ya que para cada vértice solo se consideran los vértices posteriores, evitando la necesidad de evaluar combinaciones redundantes o inválidas. Además, este enfoque reduce el número de comparaciones necesarias para determinar las conexiones válidas.

En términos de complejidad, el peor escenario ocurre cuando un elemento a_i de la secuencia puede establecer una conexión válida con todos los elementos a_j que le siguen en la lista. En este caso, el número total de aristas posibles entre los vértices asciende a $O(n \cdot (n-1)) \rightarrow O(n^2 - n)$, donde n es el número de elementos en la secuencia. Por tanto la complejidad del algoritmo se puede simplificar a $O(n^2)$.

3.2 Algoritmo Bellman-Ford

El algoritmo de Bellman-Ford es un enfoque clásico para la resolución de problemas de caminos de costo mínimo en grafos con posibles aristas de costo negativo, siendo particularmente útil en la red residual resultante del problema modelado. Este algoritmo itera sobre todas las aristas del grafo, relajando los costos asociados a los caminos, lo que garantiza que se identifiquen los caminos óptimos entre vértices. Una de las principales ventajas de Bellman-Ford es su capacidad para detectar ciclos de costo negativo, que en muchos casos pueden invalidar la solución en otros algoritmos. Sin embargo, en nuestro caso, la construcción del grafo garantiza que no existirán ciclos de costo negativo relevantes, ya que las aristas de retroceso están diseñadas para que sus costos sean el opuesto de las aristas originales, evitando la formación de ciclos con costos netos negativos.

En cuanto a la complejidad temporal, Bellman-Ford requiere realizar una re-

lajación sobre todas las aristas del grafo $|E|$, repitiendo este proceso un número de veces igual al número de vértices $|V|$. Por lo tanto, su complejidad es $O(|V||E|)$. En el contexto de este problema específico, la red residual está formada por un número de vértices proporcional al tamaño de la secuencia de entrada, es decir, $O(n)$ vértices, donde n es la longitud de la secuencia. Sin embargo, como cada vértice puede potencialmente conectarse con un gran número de otros vértices (en el peor de los casos, todos aquellos que le siguen en la secuencia), el número total de aristas en la red residual puede crecer cuadráticamente, resultando en $O(n^2)$ aristas.

Por lo tanto, cuando se aplica Bellman-Ford sobre este grafo residual, el número total de operaciones necesarias para encontrar el camino de costo mínimo se estima como $O(n) \times O(n^2) = O(n^3)$. Este comportamiento cúbico de la complejidad es una característica inherente al problema, dada la densa conectividad posible entre los vértices en la red residual. A pesar de su complejidad, Bellman-Ford es una elección apropiada debido a su capacidad para manejar las aristas de costo negativo, que surgen naturalmente de la estructura del grafo residual y de la necesidad de representar las decisiones de inclusión o exclusión de vértices en las subsecuencias válidas.

Además, el uso de Bellman-Ford está justificado porque, a diferencia de otros algoritmos de caminos mínimos como Dijkstra, que requieren costos no negativos, este es capaz de operar correctamente en redes residuales que incluyen aristas con costos negativos, como es el caso de la modelación presentada. Esto asegura que el algoritmo siempre encontrará una solución óptima sin incurrir en errores debido a la presencia de dichos costos negativos.

3.3 Algoritmo MinCost-MaxFlow

La implementación del algoritmo de flujo máximo con costo mínimo sigue un enfoque iterativo para encontrar caminos aumentativos en la red residual, donde cada camino representa una posible subsecuencia de notas que maximiza el número de elementos utilizados con el menor costo posible. El objetivo es alcanzar un flujo total de 4, que corresponde a las 4 subsecuencias disjuntas que deben extraerse de la lista de notas. En cada iteración del algoritmo, el flujo se incrementa en al menos una unidad, lo que garantiza que el número de iteraciones del ciclo principal nunca excederá las 4, dado que el flujo máximo requerido es exactamente 4.

El proceso de encontrar cada camino aumentativo de costo mínimo en la red residual se realiza utilizando el algoritmo de Bellman-Ford, cuya complejidad, como se analizó previamente, es $O(n^3)$, debido a que la red residual contiene $O(n)$ vértices y $O(n^2)$ aristas. Cada vez que se encuentra un camino aumentativo, es necesario reconstruir dicho camino para actualizar el flujo en la red, lo que implica revisar las aristas involucradas y ajustar las capacidades y los costos asociados. Este proceso de reconstrucción del camino tiene un costo adicional de $O(n^2)$, ya que implica una revisión lineal de los vértices y aristas que forman parte del camino.

Como resultado, la complejidad por iteración está dominada por el costo

de encontrar el camino aumentativo mediante Bellman-Ford, es decir, $O(n^3)$. Dado que el ciclo principal del algoritmo se ejecuta un máximo de 4 veces (el número de subsecuencias requeridas), el costo total del algoritmo es el costo de una iteración multiplicado por 4. Sin embargo, dado que 4 es una constante, la complejidad asintótica global del algoritmo sigue siendo $O(n^3)$.

Es importante destacar que esta complejidad está determinada principalmente por la necesidad de encontrar caminos aumentativos de costo mínimo en una red residual densa, en la cual el número de conexiones posibles entre los vértices puede crecer rápidamente. Aun así, el uso del algoritmo de flujo máximo con costo mínimo es adecuado para este tipo de problemas, ya que permite encontrar una solución óptima que no solo maximiza el número de notas utilizadas, sino que también garantiza que se minimicen los costos asociados a la construcción de las subsecuencias.

Finalmente, la complejidad global del algoritmo está completamente dominada por el costo de aplicar Bellman-Ford en cada iteración, lo que confirma que la complejidad final es $O(n^3)$, independientemente de la naturaleza específica de las subsecuencias encontradas o del número total de iteraciones necesarias para alcanzar el flujo máximo.

4 Mejoras

Una posible forma de mejorar la complejidad del algoritmo es disminuir el orden de la cantidad de aristas del grafo original. Para ello, hay un grupo de cuestiones a tener en cuenta:

Sean:

- n_i la nota que corresponde a la posición i de la secuencia de entrada.
- $n_j \mid n_j = n_i + 1 \land \forall n_k \ (k > i \land n_k = n_i + 1 \Rightarrow k > j)$.
- $n_l \mid n_l = n_i - 1 \land \forall n_k \ (k > i \land n_k = n_i - 1 \Rightarrow k > l)$.
- $n_p \mid n_p \equiv n_i(7) \land \forall n_k \ (k > i \land n_k \equiv n_i(7) \Rightarrow k > p)$.

Entonces $\forall n_k \mid |n_k - n_i| = 0 \lor n_k \equiv n_i(7)$ se cumple que:
Si $|n_k - n_i| = 0$ entonces $n_k = n_i + 1 \lor n_k = n_i - 1$:

- $n_k = n_i + 1 \Rightarrow n_k = n_j$.
- $n_k = n_i - 1 \Rightarrow n_k = n_l$.

Si $n_k \equiv n_i(7)$ entonces $n_k \equiv n_p(7)$.

Luego, si desde cada nota n_i tenemos una arista hacia cada una de las notas n_j, n_l, n_p y desde cada una de ellas tenemos aristas hacia cada una de las notas del mismo valor o que dejan el mismo resto al dividirlas entre 7, logramos que cada una de las notas alcanzables desde n_i sean alcanzables desde alguna de las notas n_j, n_l o n_p.

Por tanto, en el grafo por cada nota n_i tendremos 4 vértices:

- $v_{i,1}$: vértice al que llegarán todas las aristas provenientes de vértices $v_{j,1}$ tales que n_j coincide con n_i y las aristas provenientes de vértices $v_{j,4}$ tales que la diferencia modular entre n_j y n_i es 1. Desde este vértice se tiene una arista hacia el siguiente vértice $v_{j,1}$, tal que $n_i = n_j$.

- $v_{i,2}$: vértice al que llegarán todas las aristas provenientes de vértices $v_{j,2}$ cuyo valor n_j es congruente módulo 7 con n_i. Desde este vértice se tiene una arista hacia el siguiente vértice $v_{j,2}$, tal que $n_i \equiv n_j(7)$.

- $v_{i,3}$: vértice al que están conectados los dos anteriores, mediante una arista de costo 0 y capacidad 1, cada uno respectivamente. El vértice auxiliar usado como origen también tiene una arista de costo 0 y capacidad 1 hacia este vértice.

- $v_{i,4}$: vértice al que está conectado $v_{i,3}$, mediante una arista con costo -1 y capacidad 1, que análogamente a la primera modelación vista, representa la inclusión de esta nota en la canción. Desde este vértice se tienen aristas hacia los próximos vértices $v_{j,1}$, $v_{k,1}$ y $v_{l,2}$ tales que $n_i = n_j - 1$, $n_i = n_k + 1$ y $n_i \equiv n_l(7)$ respectivamente. También se incluye la arista hacia el vértice auxiliar de salida.

Nota: Todas las aristas para las cuales no se especificó costo ni capacidad tienen valores de 0 y 1, respectivamente.

Luego, por cada una de las notas, se tienen a lo sumo 6 aristas que salen de alguno de los vértices especificados anteriormente, de manera que la cantidad de aristas del grafo resultante es $O(n)$, donde n es la cantidad de notas en la secuencia de entrada.