Problema 3: Ajustando el código

Amanda Marrero Santos

Loraine Monteagudo García

Carlos Rafael Ortega Lezcano

Universidad de La Habana

2018-2019

Resumen

En este informe se realiza un análisis de distintas soluciones de un problema que consiste en igualar una lista con otra en la menor cantidad de pasos, teniendo en cuenta que existe un número n que representará el máximo elemento de ambas listas en todo momento. Primero, realizamos una simplificación del problema que, aunque no es solución, constituye una buena aproximación. Se asume que el comportamiento es hacia una sola dirección y se retorna la mínima cantidad de pasos para igualar ambas listas a través de una solución dinámica con un costo de $O(n^2)$. Después, se decide ser mucho menos eficiente pero más correctos con una solución de fuerza bruta que itera sobre los primeros c números, donde c es la cantidad mínima de movimientos esperados. Luego de esto se desarrolla una solución recursiva que tiene un enfoque de divide y vencerás, buscando obtener la respuesta óptima dividiendo la lista en intervalos y retornando de estas la cantidad de movimientos mínima para igualarlos a cierto valor con un costo de $O(n^2|l^3|)$. Finalmente, se es un poco más eficiente transformando la definición recursiva en una dinámica.

I. Análisis del Problema

En el problema se tiene como entrada:

- Un número positivo n
- Lista pw de tamaño m con el código a introducir.

$$\text{pw} = \boxed{p_1}\boxed{p_2}\boxed{p_3}\boxed{...}\boxed{p_{m-1}}\boxed{p_m}$$

  Que cumplirá que $\forall p_i \in pw \;\; 0 \leq p_i < n$
- Lista l de m enteros.

$$\text{l} = \boxed{l_1}\boxed{l_2}\boxed{l_3}\boxed{...}\boxed{l_{n-1}}\boxed{l_n}$$

  Al igual que en con la lista anterior se tendrá que $\forall l_i \in l \;\; 0 \leq l_i < n$

Un movimiento es: dada la lista l, se pueden trasladar números contiguos de la lista, es decir, se pueden mover en bloque. Si es hacia la derecha y en la posición j se encuentra el elemento $l_j = i$, entonces l_j se moverá $(i+1)mod(n)$ y si es hacia la izquierda $(i-1)mod(n)$. Se tiene entonces que determinar:

- El menor número de movimientos posibles para que la lista l coincida con los números de la lista pw que contiene el código.

II. Simplificando el problema

Para empezar a familiarizarnos con el problema e ir comprobando el comportamiento de las soluciones se realiza una simplificación: asumimos que los discos solo pueden moverse en una dirección, hacia la izquierda o hacia la derecha.

La idea principal para resolver este enfoque es a través de una dinámica. Primero, explicaremos el procedimiento para determinar la menor cantidad de movimientos hacia la derecha. La matriz acumuladora dp que utilizaremos tendrá el siguiente significado:

dp[i,j]: la menor cantidad de movimientos hacia la derecha (o izquierda) necesarios para hacer coincidir el bloque de discos del intervalo $[i, j]$ con el equivalente código.

Así, por ejemplo, si se tienen las siguientes entradas:

Código = 3 0 | 1 4 8 | 2 6

Discos = 0 1 | 0 2 6 | 7 1

La menor cantidad de elementos para transformar el intervalo **0 2 6** en **1 4 8** se encontrará en $dp[2, 4]$.

Para los elementos que se encuentran en la diagonal de dp, es decir, $\forall i < n$ se cumplirá que dado un paso el valor l_i cambiará a: $(l_i + 1) mod(n)$, ya que la única manera de transformar un elemento $l_i \in l$ en $p_i \in pw$ moviendo solo a la derecha es desplazándose $(i + 1) mod(n)$. Por lo tanto, la cantidad de pasos que se deben realizar para llegar de l_i a p_i es $(p_i - l_i)\,\%n$. Luego, los elementos de esta diagonal serán los casos base para calcular los restantes elementos.

Además, otro caso especial es contar la cantidad de movimientos necesarios para mover un bloque de 2 elementos.

Demostremos el siguiente lema:

Lema 1

Dado un bloque l_i, l_{i+1}, donde l_i necesita k movimientos para igualarse a p_i y l_{i+1} requiere m para p_{i+1}, supongamos sin pérdida de generalidad que m es menor que k, entonces siempre es posible llevar ambos elementos a p_i y p_{i+1} en k pasos.

Podemos transformar l_{i+1} en m pasos, y como $k > m$, entonces l_i al moverse m veces no llegará a p_i. Por lo tanto, podemos mover l_i y l_{i+1} juntos con m pasos, y todavía faltarán $k - m$ pasos para que l_i se iguale a p_i, por lo que estos se realizarán, alcanzando un total de $m + k - m = k$ pasos para llegar al objetivo.

Entonces, para todos los elementos que estén sobre la diagonal (que representan los bloques de tamaño 2), la cantidad de pasos para llegar al objetivo es máx$(dp[i, i], dp[i - 1, i - 1])$, donde $dp[i, i]$ y $dp[i - 1, i - 1]$ tienen la cantidad de pasos necesarios para igualar los elementos que conforman la secuencia de tamaño 2.

Por lo tanto, la ley de formación para llenar los elementos de $dp[i, j]$ será:

$$dp[i,j] = \begin{cases} (pw[i] - a[i])\,\%n, & i = j \\ \text{máx}(dp[i,i], dp[i-1,i-1]) & i = j-1 \\ dp[i+1,j] + dp[i,j-1] - dp[i+1,j-1], & i < j \end{cases}$$

La idea intuitiva detrás de esta dinámica es que para obtener la mejor solución para $[i,j]$ se puede usar la cantidad de movimientos con los que se realizó el bloque $[i+1,j]$, es decir, el mismo intervalo sin incluir al elemento en la posición i, ya que en $dp[i+1,j]$ está la mínima cantidad de elementos para mover dicho bloque. Además, cuando se quiera añadir el elemento en la posición i, se deben de realizar la misma cantidad de movimientos que en el bloque $[i,j-1]$, pero los elementos en las posiciones entre $[i+1,j-1]$ se mueven 2 veces, por lo que estos movimientos son innecesarios, entonces, se debe restar $dp[i+1,j-1]$

La matriz dp depende de los términos que se encuentran a su izquierda, hacia abajo o en la diagonal izquierda hacia abajo, por lo que se llenará diagonalmente, empezando en la columna 2. Recordemos que el resultado final a retornar es la mínima cantidad de movimientos para convertir la secuencia entera de m elementos, por lo que el resultado a retornar es la mínima cantidad de pasos para el bloque $[0,m-1]$, que es el valor que se guarda en $dp[0,m-1]$

Una idea análoga se puede desarrollar para determinar la mínima cantidad de pasos necesarios para que, a través de transformaciones, se lleguen a igualar los términos de l realizando solo movimientos hacia la izquierda. En este caso, dp tendrá la siguiente expresión:

$$dp[i,j] = \begin{cases} (a[i] - pw[i])\,\%n & i = j \\ \text{máx}(dp[i,i], dp[i-1,i-1]) & i = j-1 \\ dp[i+1,j] + dp[i,j-1] - dp[i+1,j-1], & i < j \end{cases}$$

La primera diferencia notable con la definición de dp anterior es el caso base para los elementos que se encuentran en la diagonal. Y es que, como los elementos se mueven hacia la izquierda, entonces dado un movimiento l_i cambiará su valor por $(l_i - 1)mod(n)$, por lo que el movimiento se realiza al sentido contrario que en el ejercicio anterior, entonces la cantidad de movimientos necesarios son $(a[i] - pw[i])\,\%n$. Notemos que el resto de los lemas y las explicaciones dadas para transformar la secuencia con movimientos hacia la derecha son reutilizables para este procedimiento.

Este algoritmo no constituye una solución al problema, ya que en el problema original se deben combinar tanto movimientos a la derecha como a la izquierda. A pesar de esto, se realizó un intento de usar esta información para "mezclar"de alguna manera los resultados para obtener la respuesta original. Aunque dicho objetivo no se cumplió, estos valores constituyen una buena aproximación del problema.

III. Fuerza Bruta

Para el algoritmo de fuerza bruta se define una cantidad de movimientos inicial y se prueba si para todas las posibles operaciones a realizar con los discos se consigue igualar la configuración inicial al código con la cantidad de movimientos predeterminada.

Para esto se realiza una búsqueda comenzando en 0, siendo este la cantidad de movimientos a realizar. Luego, este valor se va aumentando hasta que se encuentre el primer número con el cual se consiga igualar el código. El primer valor con el que se consiga el objetivo será la mínima cantidad de pasos necesarios.

Para una cantidad de movimientos i se pueden tomar las siguientes decisiones: mover hacia la izquierda, o derecha hasta que se agoten los movimientos. Como es posible seleccionar un bloque de discos, entonces se prueba no solo para cada disco, sino para todos los posibles intervalos que se puedan seleccionar.

Correctitud

El algoritmo verifica por cada número a partir de 0 cuál es el primer valor con el cual dada esa cantidad de pasos se logra igualar la secuencia a h; por lo tanto, el primer valor que logre esto se encontrará, lo que garantiza la correctitud del algoritmo.

Complejidad

Primero se generan todas los posibles intervalos, que dado una secuencia de m es en total $\binom{m+1}{2}$ y por cada uno de estos elementos se comprueban 2 opciones: decrementar o incrementar, requiriendo un total de $2^{\binom{m+1}{2}}$ operaciones. Luego, si la cantidad de movimientos óptima es c, entonces se iterará hasta el valor c, como c es un número sobre el que se hacen transformaciones, entonces el costo de la iteración será $O(2^k)$, donde k es la cantidad de bits del número. Como por cada uno de estos subconjuntos formados se itera se tiene un total de $O(2^k * 2^{\binom{m+1}{2}})$ operaciones.

El algoritmo podría tener varias mejoras, como por ejemplo, usar el valor aproximado resultado de la dinámica explicada en el enfoque anterior y luego realizar una búsqueda binaria entre este número y 0 para obtener la respuesta original. Esto daría un costo de $O(k * 2^{\binom{m+1}{2}})$, donde k es la cantidad de bits del número.

IV. Solución Recursiva

Definición: Sea X, Y dos arreglos tal que $|X| = |Y|$, se define distancia entre el arreglo X y Y, como:

$$Dist(X, Y) = D \mid \forall i : 0 \leq i \leq |X| - 1 : D[i] = (X[i] - Y[i] + n) \text{mod}(n)$$

De la definición anterior podemos notar que $X = Y$ si y solo si $\forall x \in Dist(X, Y) : x = 0$, si se cumple que $\forall i : 0 \leq i \leq |X| - 1 : X[i] = Y[i] \implies D[i] = (X[i] - Y[i] + n) \text{mod}(n) \implies D[i] = 0$

Es necesario convertir el arreglo Y (que representa la configuración inicial de los discos) en el arreglo X (el código al que se converge), usando la menor cantidad de movimientos

Lema 1

El mínimo número de transformaciones necesarias para que Y sea igual a X, coincide con el número de operaciones para transformar $Dist(X, Y)$ en un arreglo con todos sus elementos nulos

Sea n el mínimo número de operaciones para convertir Y en X, tal que para el paso k-ésimo conocemos que operación realizar a Y (restar o sumar 1 al intervalo seleccionado), definamos $T = \{Y = Y_0, Y_1, ..., Y_n = X\}$ como la secuencia de estados de Y para convertirse en X. Probemos que podemos transformar $Dist_k(X, Y_k)$ en $Dist_{k+1}(X, Y_{k+1})$ empleando una y solo una operación sobre el mismo intervalo seleccionado para el paso de Y_k a Y_{k+1}, la expresión para D_{k+1} será:

$$D_{k+1}(X, Y_{k+1}) = (X[i] - Y_{k+1}[i] + n)\text{mod}(n)$$

Donde $i \in [l; r]$, y representa el intervalo seleccionado para realizar la operación, podemos obtener los valores de $Y_{k+1}[i]$ a partir de $Y_k[i]$ de la siguiente forma:

$$Y_{k+1}[i] = (Y_k[i] \pm 1)\text{mod}(n)$$

Si sustituimos en la expresión de D_{k+1} se obtendrá:

$$D_{k+1}(X, Y_{k+1}) = (X[i] - (\ (Y_k[i] \pm 1)\text{mod}(n)\ ) + n)\text{mod}(n)$$

Obtenemos entonces una relación entre $D_{k+1}(X, Y_{k+1})$ y $D_k(X, Y_k)$, ya que es posible mediante una única operación realizar la transformación. Si repetimos este mismo razonamiento resulta que luego de n operaciones $Dist(X, Y)$ se transforma en un arreglo donde todos sus elementos son 0.

Lema 2

La mínima cantidad de pasos necesarios para transformar a en b ambos módulo n puede ser expresada como $d(x, y, n) = \text{mín}(|a - b|, n - |a - b|)$

Sea a y b distintos, para poder convertir a en b contamos con dos opciones, sumar o restar 1, tal que $(a \pm 1)\text{mod}(n)$, en este caso la mínima cantidad de movimientos se logra si siempre realizamos la misma operación sobre a, seleccionada acorde a la distancia entre los números. El valor de $|a - b|$ representa la cantidad a sumar en caso que $a > b$ o restar si $a < b$, este valor se compara con $n - |a - b|$, que es el número de operaciones para transformar a en b de forma circular. El mínimo de estos valores es la mínima cantidad de operaciones, ya que para $|a - b|$ próximo a n, es mejor aprovechar la congruencia con n.

El arreglo pw no posee todos sus elementos iguales, esto aumenta la complejidad a la hora de realizar la transformación. Basándonos en el **Lema 2**, podemos conocer como transformar todos los elementos de un arreglo A hacia un mismo número m. Apoyándonos en el **Lema 1**, sabemos que si dados pw, l y $D = Dist(pw, l)$, transformar los elementos de D en 0, con la mínima cantidad operaciones será equivalente a transformar l en pw

Para determinar el número mínimo de operaciones para convertir los elementos de A en un valor m, definimos un algoritmo cuyo objetivo es dividir el arreglo A en dos intervalos, tal que se

minimice en cada uno el número de operaciones para converger a un valor v, el cual cumple que $d(v, m, n)$ minimiza el resultado final.

El algoritmo se divide en dos pasos, primeramente se define el intervalo $[i, j]$ y el valor m, la respuesta para $i = 0,\ j = |A| - 1$ como extremos del intervalo será la solución del problema. A continuación, determinamos la mejor forma de dividir el intervalo y el valor de v al que es necesario converger para minimizar la cantidad de operaciones, para esto debemos probar con todas las formas de dividir el intervalo y para cada una encontrar el valor de v con $0 \leq v < |A|$.

Definamos la función recursiva f, que dados los extremos del intervalo y el valor m, devuelve la mínima cantidad de operaciones:

$$f(A, i, j, m) = \begin{cases} d(A[i], m, n) & i = j \\ \\ \text{mín}\{f(A, i, k, v) + f(A, k+1, j, v) + d(v, m, n)\} & i < j \end{cases}$$

Correctitud

El algoritmo utiliza un enfoque divide y vencerás, donde busca la mejor forma de dividir el intervalo, como caso base tenemos que cuando $i == j$ estamos ante un intervalo de un solo elemento y por Lema 2 se garantiza que esta es la mínima cantidad de pasos. Para cada intervalo se exploran todas las posibilidades y se selecciona la menor de todas, garantizando la correctitud del algoritmo.

Complejidad

Debemos comprobar todos los tríos (i, j, m) para determinar los extremos de los 2 intervalos, por lo tanto, si la cantidad de discos es $|l|$, entonces es necesario $|l|^3$ operaciones, además por cada trío se selecciona el valor de división para el intervalo el cual puede ser n y el valor v, al cual convergen los intervalos, el cual se comprueba en el intervalo $1 \leq v < n$, resultando en n^2 operaciones por cada trío, por tanto la complejidad temporal de nuestro algoritmo es $O(n^2|l|^3)$. En realidad, se itera y se realizan transformaciones sobre un valor n que es de entrada, por lo que nuestro algoritmo será pseudopolinomial.

V. Dinámica

La recursión anterior no aprovecha el calculo realizado en llamadas anteriores, además podemos observar como la solución se apoya en problemas más simples que podemos expresar bajo las mismas condiciones por lo que se ideó una solución dinámica de 3 dimensiones que sigue la misma idea expresada en la recursividad, su expresión es la siguiente:

$$dp[i, j, v] = \begin{cases} d(A[i], v, n) & i = j \\ \\ \text{mín}_{1 \leq k < j}\{dp[i, k, v] + dp[i, k+1, v] + d(v, m, n)\} & i < j \end{cases}$$

$dp[i, j, v]$: Es la menor cantidad de pasos para transformar el intervalo que comienza en i y termina en j al número v.

Los valores base se colocan en aquellas posiciones que cumplen tener sus dos primeras componentes iguales, o sea $dp[i, i, a]$, representando los intervalos de un solo elemento para los

cuales sabemos calcular la cantidad óptima de pasos para convertir el número en la posición i del arreglo en a.

El array se rellena por la diagonal $dp[i, j, v]$ dependiendo de valores mayores que i, y de valores menores que j, además de todos los posibles valores que tome v, de esta forma siempre se dispone de la información para poder determinar para i, j todos los valores de los intervalos que se derivan de este.

La respuesta al problema corresponde a convertir el arreglo $Dist(pw, l)$ a 0, por lo tanto, la solución se encontrará en $dp[0, |l| - 1, 0]$, o sea, transformando el intervalo que cubre todo $Dist(pw, l)$ a 0.

La similitud de la dinámica y la solución recursiva resulta evidente. En realidad, como se mencionó anteriormente, solo se aplican optimizaciones a la definición recursiva para evitar generar soluciones repetidas. Por esto, con la correctitud del enfoque anterior se prueba la de este algoritmo. La complejidad temporal de la solución también será la misma que la anterior.