E. Hanoi Factory

Eric Luis López Tornas C411

Contents

1 Definiendo el Problema

Seguro que has oído hablar de la famosa tarea de las Torres de Hanoi, pero ¿sabías que existe una fábrica especial que produce los anillos para este maravilloso juego? Hace mucho tiempo, el gobernante del antiguo Egipto ordenó a los trabajadores de la Fábrica de Hanoi crear una torre tan alta como fuera posible. No estaban preparados para cumplir con una orden tan extraña, por lo que tuvieron que crear esta nueva torre utilizando los anillos ya producidos.

Hay n anillos en el almacén de la fábrica. El i-ésimo anillo tiene un radio interior a_i, un radio exterior b_i y una altura h_i. El objetivo es seleccionar un subconjunto de anillos y organizarlos de tal manera que se cumplan las siguientes condiciones:

1. Los radios exteriores formen una secuencia no creciente, es decir, uno puede colocar el anillo j-ésimo sobre el anillo i-ésimo solo si $b_j \leq b_i$.
2. Los anillos no deben caer uno dentro del otro. Eso significa que se puede colocar el anillo j-ésimo sobre el anillo i-ésimo solo si $b_j > a_i$.
3. La altura total de todos los anillos utilizados debe ser la máxima posible.

Encuentre la altura de la torre que cumple las tres condicciones anteriores.

2 Observaciones

Ordenamiento inicial: Dado que la selección de los anillos para construir una solución al problema depende de una relación de orden, primero se considerarán los radios exteriores y, en segundo lugar, los radios interiores. La lista de discos se ordenará de manera descendente según sus radios exteriores. Una vez hecho esto, para discos con igual radio exterior, se ordenarán de manera descendente según su radio interior. Esto asegura que cualquier secuencia de anillos elegida cumple al menos con la primera condición para ser una solución válida.

Torre (Secuencia) válida: Es una secuencia de anillos que cumple las dos primeras condiciones para formar una secuencia óptima.

A: Es la lista inicial de anillos, la cual ha sido ordenada de manera descendente, primero por radio exterior y luego por radio interior.

3 Primera Solución

3.1 Idea

Se toman todas las posibles subsecuencias s de A y, por cada una de ellas, se verifica que cumpla la segunda condición, es decir, que todo anillo es sostenido por el anillo que está más a la izquierda. Luego, para cada subsecuencia valida s se registra su altura, manteniendo un registro de la altura máxima alcanzada. Dado que el algoritmo explora todas las subsecuencias posibles que cumplen las dos primeras condiciones, al finalizar, se habrá registrado la secuencia óptima de anillos.

3.2 Complejidad Temporal

Dado que el algoritmo se compone de dos ciclos y que las operaciones realizadas en el interior de estos son $O(1$, la complejidad temporal del algoritmo propuesto es $O(n^2)$.

4 Segunda Solución

4.1 Idea

Se utiliza un enfoque de programación dinámica empleando una lista auxiliar dp del mismo tamaño que A. En esta lista, $dp[i]$ representa la altura máxima que puede alcanzar una torre cuyo anillo superior sea $A[i]$.

4.2 Valor de dp[i]

$\mathbf{dp}[i] = dp[j] + h_i$, donde j es el anillo a la izquierda de i con el mayor valor en dp, al cual se le puede colocar el anillo i, es decir, $r_j \leq b_i$. Si $i = 1$, entonces $\text{dp}[i] = h_i$.

4.3 Algoritmo propuesto

Se recorre la lista A de izquierda a derecha y, por cada anillo i, se recorren de derecha a izquierda los $i-1$ anillos anteriores. En este recorrido, se registra el valor máximo que puede tener dp en los anillos que capaces de sostener a i, es decir, aquellos cuyo radio interior sea menor que b_i. Sea j el índice correspondiente al valor máximo encontrado, se asigna $\text{dp}[i] = \text{dp}[j] + h_i$, y se continúa el ciclo externo con el anillo $i+1$, si existe.

Para el momento en que se analiza el anillo i, todos los valores de dp de los anillos $i-1$ a su izquierda ya habrán sido calculados correctamente.

4.4 Complejidad Temporal

Puesto que al momento de seleccionar el anillo número i se puede verificar con el anillo i - 1 que la addición de i a la respuesta válida que tenemos hasta el momento forma una nueva respuesta válida, entonces el propio recorrido del backtrack asegura que en todo momento tengamos una secuencia valida por tanto la complejidad temporal del primer algoritmo propuesto es $O(2^n)$.

5 Tercera Solución

5.1 Idea

Para esta solución, utilizamos un algoritmo de tipo greedy. El objetivo es emplear las propiedades del problema para confeccionar, para cada anillo x, una torre válida S que tenga la altura máxima posible con x como anillo superior. Esto se realiza utilizando una pila, por la cual pasarán todos los anillos exactamente una vez. En el momento de guardar x en la pila, solo estarán presentes los anillos necesarios para formar S.

5.2 Observaciones:

A la hora ordenar una secuencia de anillos con igual radio exterior b es conveniente poner el de menor radio interior encima de todos los demás. Ya que este es el que más opciones tiene de sostener nuevo anillo entre todos los demás de su mismo radio.

Si un anillo de radio exterior b pertenece a la solución óptima, entonces todo anillo de radio b también pertenece a esta solución. Si no es así, sea S una solución óptima. Podemos colocar el anillo x de radio exterior b, que no está en S, justo debajo del último anillo t de radio b que se encuentra en S. Inmediatamente obtendremos una nueva secuencia válida con una altura mayor que S, lo que resultaría en una contradicción.

Por lo tanto, podemos tratar todos los anillos $x_i, x_{i+1}, \ldots, x_{i+k}$ de igual radio exterior b como un solo anillo X con radio exterior b, radio interior $a = \min(a_i, a_{i+1}, \ldots, a_{i+k})$ y altura igual a $h = h_i + h_{i+1} + \ldots + h_{i+k}$.

A partir de este punto podemos considerar que todos los anillos de la lista A tienen radios exteriores distintos. Este convenio facilita las demostraciones siguientes y aunque se trata de una transformación simple no es necesaria para el funcionamiento del algoritmo.

5.3 Algoritmo:

Sea Q una pila. Se recorre la lista A y para cada anillo i, se verifica que el anillo j en la parte superior de la pila cumple que $a_j < b_i$. Si es así, se guarda el anillo i en Q. En caso contrario, se retira j de la parte superior de la pila y se vuelve a realizar la verificación anterior. Si Q está vacía, se guarda i en Q.

5.4 Lema

Sean j e i anillos de A, donde j es el anillo a la izquierda de i más cercano al que se puede colocar i. Entonces, la torre válida de mayor altura que tiene a i en su parte superior tendrá a j sosteniendo a i. Es decir, la secuencia válida que acumula la mayor suma de alturas teniendo a i en su extremo derecho tendrá a j inmediatamente antes que i.

Demostración

Supongamos que existe un anillo t distino a j que ocupa el inmediatamente a la izquierda de i en dicha secuencia. Por hipótesis, se cumple que j es el anillo más cercano a i por la izquierda, por lo que $t < j < i$. Esto implica, por la construcción de A, que $b_i < b_j < b_t$. Además, dado que i puede colocarse tanto sobre t como sobre j, se tiene que $a_j < b_i$ y $a_t < b_i$. De aquí se deduce que $a_t < b_j$. Dado que $b_j < b_t$, el anillo j también podría colocarse sobre t. Sin embargo, dado que j no puede estar debajo de t en ninguna secuencia válida, esto implica que j no se encuentra en S. Por lo tanto, si adicionamos j entre t e i en S, obtendremos una nueva secuencia válida S' con i en el extremo derecho y mayor altura que S, lo que contradice la definición de S.

5.5 Lema

Sean j e i anillos de A, donde j es un anillo a izquierda de i. Si i no sé puede colocar sobre j entonces ningún anillo a la derecha de i en A se puede colocar sobre j.

Demostración Sea l un anillo a la derecha de i. Por construcción de A, se cumple que $b_l < b_i < b_j$. Si el anillo i no se puede colocar sobre j, entonces $b_i < a_j$, lo que implica que $b_l < a_j$.

5.6 Lema

En el momento en que se selecciona el anillo i para ser guardado en la pila Q, en esta se encontrarán todos los anillos que pueden sostener a i.

Demostración
Para el momento de guardar el anillo i en Q, todo anillo a la izquierda de i en A se habrá guardado una vez en Q. Supongamos que existe un anillo x en A que puede sostener a i, pero que en este momento no está en Q. Por construcción del algoritmo, x tuvo que ser sacado de Q por un anillo y que se encuentra en A entre x e i. Solo pudo haber sacado a x de Q si $b_y \leq a_x$, puesto que $y < x$ en A, entonces $b_x < b_y$. Pero $a_x < b_i$ y, por lo tanto, $b_x < b_y < b_x$, lo cual es imposible.

5.7 Lema

El anillo i será guardado en la pila sobre el mejor disco que lo puede sostener.

Demostración
Sea i un anillo a punto de ser guardado. Por el lema anterior, todos los anillos que pueden sostener a i estarán en Q en este punto. Además, todos estos anillos conservarán el mismo orden entre sí que tienen originalmente en A. Luego, el anillo j, que está más cerca de i por la izquierda en A, también estará más cerca al tope de la pila. Si existiera un anillo x entre j y el tope de Q, necesariamente se cumpliría que $b_i \leq a_x$, ya que, de no ser así, x podría sostener al anillo i, lo cual sería una contradicción con lo anterior. Por lo tanto, si existiera tal x, sería removido de Q antes de insertar a i en el tope de la pila.

5.8 Teorema:

Justo después de que el anillo i sea guardado en la pila Q, en esta estará la torre válida más alta posible que puede tener al anillo i en su tope.

Demostración
Como resultado del último lema, al insertar i, este se apoya en el anillo óptimo j que puede sostenerlo. Por la construcción del algoritmo, no se puede modificar la secuencia S de anillos debajo de j en Q sin extraer a j. Dado que los anillos se guardan una única vez en Q, para cuando i es insertado en Q, la secuencia S es la misma que cuando j fue insertado en Q. Así, j también está apoyado sobre el anillo óptimo que puede sostenerlo. Por lo tanto, en este momento del algoritmo, en Q se encuentra la secuencia válida más alta que tiene al anillo i en su tope.

5.9 Complejidad Temporal

Todo anillo entra una vez en la pila Q, y a lo sumo se realizarán $n-1$ extracciones. Usando una pila que realice las operaciones de inserción o extracción en $O(1)$, tenemos que la complejidad del algoritmo greedy propuesto es $O(n)$. Dado que el costo del ordenamiento inicial es $O(n \log(n))$, la complejidad temporal de la tercera solución es $O(n \log(n))$.

6 Anexos

6.1 Código de la primera solución

```python
def backtrack(A, index, subsequence, aux, max_val):
    # Caso base: si ya procesamos todo anillo en A
    if index == len(A):
        return max(max_val, aux)

    # Excluimos el anillo en la posici n actual
    max_val = backtrack(A, index + 1, subsequence, aux, max_val)

    # Incluimos el anillo en la posici n actual si forma una
        secuencia valida
    if len(subsequence) == 0 or (len(subsequence) > 0 and
        subsequence[len(subsequence)-1][0] < A[index][1]):
        subsequence.append(A[index])
        max_val = backtrack(A, index + 1, subsequence, aux + A[
            index][2], max_val)
        subsequence.pop()

    return max_val

# Lectura de entrada
t = int(input())
A = []
for _ in range(t):
    a, b, h = map(int, input().split())
    ring = (a, b, h)
    A.append(ring)

A.sort(key=lambda x: (x[1], x[0]), reverse=True)
max_val = 0
max_val = backtrack(A, 0, [], 0, max_val)
print(max_val)
```

Listing 1: Código solución

6.2 Código de la segunda solución

```
t = int(input())
A = []
dp = [0] * t
for _ in range(t):
    a, b, h = map(int, input().split())
    ring = (a, b, h)
    A.append(ring)

A.sort(key=lambda x: (x[1], x[0]), reverse=True)
max_val = A[0][2]
dp[0] = max_val
for i in range (1, t):
    aux = 0
    for j in range(0, -i):
        if A[i][1] > A[j][0] and dp[j] > aux:
            aux = dp[j]
    dp[i] = aux + A[i][2]
    if dp[i] > max_val:
        max_val = dp[i]
print(max_val)
```

Listing 2: Código solución

6.3 Código de la tercera solución

```python
from collections import deque
t = int(input())
A = []
for _ in range(t):
    a, b, h = map(int, input().split())
    ring = (a, b, h)
    A.append(ring)

A.sort(key=lambda x: (x[1], x[0]), reverse=True)
q = deque([])
max_val = A[0][2]
height_q = 0

for i in range (0, t):
    while True:
        if len(q) == 0 or q[-1][0] < A[i][1]:
            q.append(A[i])
            height_q += A[i][2]
            break
        else:
            x = q.pop()
            height_q -= x[2]
    if height_q > max_val:
        max_val = height_q

print(max_val)
```

Listing 3: Código solución