Problemas de Diseño y Análisis de Algoritmos

Olivia González Peña
Juan Carlos Casteleiro Wong
Adrian Tubal Páez Ruiz

2018 - 2019

Problema 2. Bolos con penalización

Una cierta cantidad de bolos se encuentran ubicados uno al lado del otro. Cada bolo tiene un número impreso, el cual representa la cantidad de puntos que se obtiene al derribarlo. Un jugador tiene una cierta cantidad de bolas y cada una es lo suficientemente ancha como para derribar un número determinado de bolos consecutivos. El juego de bolos se torna un poco más difícil al introducir bolos con penalización. Los bolos con penalización tienen puntuación negativa, de forma tal que al derribarlos la puntuación del jugador disminuye. Esto puede cambiar la estrategia de un jugador, se pueden utilizar los espacios vacios a la izquierda y a la derecha de los bolos, así como los espacios creados por disparos anteriores. De esta forma se podrían evitar bolos con penalización.

Considere el siguiente ejemplo:

$$2 \quad 8 \quad -5 \quad 3 \quad 5 \quad 8 \quad 4 \quad 8 \quad -6$$

Si el jugador tiene tres bolas, cada una de ellas capaz de derribar tres bolos consecutivos, la puntuación máxima que se puede alcanzar es 38. La suma de los tres tiros serán: 2+8, 3+5+8 y 4+8. El primer disparo del jugador sería intencionadamente a la izquierda para derribar solamente los dos primeros bolos, 2 y 8, evitando el -5. El segundo disparo derribará al 3, 5 y 8 y el último al 4 y al 8, aprovechando el espacio dejado por el disparo anterior para evitar el -6. Se necesita un programa que dada la secuencia de bolos, la cantidad de bolas y su ancho, determine la cantidad máxima de puntos que puede ser alcanzada.

Solución 1: Fuerza Bruta

Para resolver el problema, revisaremos todas las posibles combinaciones de k tiros calculando la puntuación alcanzada con cada una de ellas. La solución será la máxima puntuación obtenida al terminar el algoritmo.

Cada una de las combinaciones puede ser generada como una máscara de *bits*, es decir, si la posición i contiene un 1 en la máscara es porque se debe realizar el tiro que abarca los bolos de las posiciones en el intervalo $[i, i + d - 1]$. Si en la máscara hay menos de k unos, es porque los tiros faltantes fueron realizados por fuera.

Correctitud

Como el algoritmo se basa en calcular todas las posibles combinaciones de tiros, y cada una es analizada para ver si es mejor que las anteriores, no cabe posibilidad de dejar de contar alguna distribución de tiros.

Complejidad Temporal

Como se buscan todas las combinaciones posibles el algoritmo es:

$$O(n \cdot d \cdot 2^{n+d+1})$$

Donde d es el ancho de la bola y n la cantidad de bolos.

Solución 2: Dinámica

Esta solución dinámica se apoya en una matriz dp donde $dp[i][j]$ contiene la mejor puntuación que se puede alcanzar gastando un tiro por j y los $i-1$ restantes por posiciones anteriores a j. Para ello, se tienen en cuenta los tiros que se interceptan con él y los que no, en este rango. Donde lanzar por la posición j es tumbar los bolos en el intervalo $[j-d+1, j]$. La solución sería el mayor valor que se encuentra en la fila k-ésima.

Correctitud

Como las soluciones de la fila i se apoya en las de la $i-1$ y en la 1, es necesario llenar la primera fila de la siguiente forma: $dp[1][i]$ es la suma de todos los valores de $dp[1][i-d+1]$ hasta $dp[1][i]$.

Lema 1

Sea j, tal que $1 \leq j \leq n$, $\forall t$, $j-d+1 \leq t < j$ se cumple que $I_j \cap I_t \neq \emptyset$. Entonces los puntos acumulados tras tirar por j y por t es igual a $S(I_j) + S(I_t) - S(I_j \cap I_t)$.Donde:

$$S(I_k) = \sum_{i=k-d+1}^{k} bolos[i]$$

$$S(I_t \cap I_j) = sum[j] - sum[t] + bolos[t]$$

Siendo sum el arreglo de las sumas acumulativas de los valores de los bolos.

En el algoritmo se lleva de forma paralela una matriz `max_dp` de igual tamaño que `dp`, donde, $max\_dp[i][j]$ contiene el valor máximo que existe en la fila i desde la posición 0 hasta la j en la matriz dp.

Para toda posición i,j se puede saber cuál es el mejor comienzo con un tiro menos, por un intervalo que no se solapa con I_j: $max\_dp[i-1][j-d]$

Sea $dp[i][j]$ el i-ésimo tiro por la posición j, $dp[i-1][s]$ tal que $s < j$ el mejor disparo que puede realizarse con un tiro menos.

- Si $I_s \cap I_j = \emptyset$ entonces $dp[i][j] = dp[1][j] + max\_dp[i-1][s]$
- Si $I_s \cap I_j \neq \emptyset$ entonces

$$dp[i][j] = dp[1][j] + max\{dp[i-1][s] - S(I_s \cap I_j) \mid I_s \cap I_j \neq \emptyset\}$$

Si la cantidad de tiros posibles es k, la solución va a ser el mayor valor que se encuentra en la fila k del arreglo dp.

El algoritmo garantiza que no se excluyan soluciones, ya que, aunque se revisen siempre los posibles tiros solo a la izquierda, tras llegar a la posición $dp[k][n]$ está calculado la mejor puntuación para cada comienzo posible con k tiros que se pueden realizar; además, el orden de los tiros no es relevante.

Complejidad Temporal

1. Inicialmente se calcula el arreglo de las sumas acumulativas sum, que es $O(n)$.
2. Se llena la primera fila de la matriz dp con un costo $O(n \cdot k)$.

3. El algoritmo realiza $n \cdot k$ iteraciones para rellenar la matriz dp, calculando en cada una el mejor próximo tiro, y con ello cada uno de los intervalos que se solapan, con costo $O(n \cdot k \cdot d)$

Entonces la complejidad del algoritmo es:

$$O(n) + O(n \cdot k) + O(n \cdot k \cdot d)$$

Que por el principio de la suma es:

$$O(n \cdot k \cdot d)$$