# Repaso

## Algoritmos y Estructuras de Datos III

Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

26 de Septiembre de 2022

# Problema 1

Se arrojan simultáneamente $n$ dados, cada uno con $k$ caras numeradas de 1 a $k$. Queremos calcular todas las maneras posibles de conseguir la suma total $s \in \mathbb{N}$ con una sola tirada.
Consideramos que los dados son **indistinguibles**, es decir que si $n = 3$ y $k = 4$, entonces existen 3 posibilidades que suman $s = 6$: ⚀⚀⚃, ⚀⚁⚂ y ⚁⚁⚁.

1. Definir en forma recursiva la función $f : \mathbb{N}^3 \to \mathbb{N}$ tal que $f(n, s, k)$ es igual a la cantidad de maneras de sumar $s$ con $n$ dados indistinguibles de $k$ caras.
2. Demostrar que $f$ tiene la propiedad de superposición de subproblemas
3. Definir un algoritmo top-down para calcular $f(n, s, k)$ indicando claramente las estructuras de datos utilizadas y la complejidad resultante.
4. Escribir el (pseudo-)código del algoritmo top-down resultante.

¿Cómo hacemos para no contar ⚀⚀⚃, y ⚀⚃⚀ como tiradas diferentes?

¿Cómo hacemos para no contar ⚀⚀⚃, y ⚀⚃⚀ como tiradas diferentes?

$f(n, s, k)$: Cantidad de maneras *ordenadas* de obtener $s$ tirando $n$ dados de $s$ caras

Si consideramos a los dados ordenados, pensemos que pasa con el último dado:

- Es $k$, entonces la cantidad de formas ordenadas que tengo para los otros dados es $f(n-1, s-k, k)$, por lo tanto su resultado es 0
- No es $k$. Entonces es menor o igual a $k-1$, y como está ordenado, los demás también lo son. Podemos deducir entonces que la cantidad formas ordenadas es igual a $f(s, n, k-1)$

Notemos que si $n = 1$, la única configuración de dados es el conjunto de dados el conjunto vacío, por lo tanto es 1 si $s = 0$ y 0 en otros casos.

Si $k \leq 0$ no podemos armar ninguna configuración, ya que las caras de los dados van de 1 a $k$

Finalmente podemos agregar que si $s > nk$, entonces no tenemos ninguna configuración posible, ya que lo máximo que podemos obtener con $n$ dados es $nk$.

$$f(n,s,k) = \begin{cases} 1 & \text{if } n = 0 \wedge s = 0 \\ 0 & \text{if } n = 0 \wedge s \neq 0 \\ 0 & \text{if } k = 0 \\ 0 & \text{if } nk > s \\ f(n-1, s-k, k) + f(n, s, k-1) & \text{en otro caso} \end{cases}$$

# Superposición de Problemas

Notemos que cuando cuando hacemos las llamadas recursivas, o bien el valor de $n$ o el de $k$ disminuye en uno, así consideramos el caso cuando $n$, $k$ y $s$ son suficientemente grandes, como en caso recursivo hacemos 2 llamadas recursivas, el número de llamadas recursivas es $\Omega(2^{min(n,k)})$

La cantidad de subproblemas en cambio es igual a $nk \times min(nk, s) \subseteq O(n^2k^2)$

Así que cuando $2^{min(n,k)} >> n^2k^2$, tenemos superposición de problemas.

# Algorítmo Top down

Vamos a utilizar una matríz $M$ de tamaño $n \times s \times min(nk, s)$ para guardar los valores ya calculados y la inicializamos con un valor de indefinído ($\perp$).

- Si $n = 0$ devolvemos 1 si $s = 0$, si no 0
- Si $k = 0$ o $nk > s$ devolvemos 0, ya que vímos que no hay configuraciones posibles.
- Si $M[s, n, k] = \perp$, llamamos recursivamente y guardamos $M[s, n, k] \leftarrow f(s - k, n - 1, k) + f(s, n, k - 1)$
- Si ya tenemos $M[s, n, k]$ lo devolvemos.

# Problema 3

Dada una matriz de valores enteros de tamaño $m \times n$, $M^0 \in \mathbf{Z}^{m \times n}$, se define el siguiente proceso iterativo.

$$M_{ij}^{t+1} = max\{M_{kl}^t|_{max(0,j-1) \leq l \leq min(n,j+1)}^{max(0,i-1) \leq k \leq min(m,i+1)}\}$$

$$\begin{pmatrix} 1 & 5 & 10 & 5 \\ 6 & 4 & 12 & 4 \\ 10 & 5 & 12 & 11 \\ 5 & 11 & 23 & 9 \end{pmatrix} \qquad \begin{pmatrix} 1 & 5 & 10 & 5 \\ 6 & 4 & 12 & 4 \\ 10 & 5 & 12 & 11 \\ 5 & 11 & 23 & 9 \end{pmatrix}$$

1. Observar que este proceso converge en una cantidad fínita de pasos, i.e., existe $t \geq 0$ tal que $M^{t+1} = M^t$. Dar una justifícación.
2. Diseñar un algoritmo que encuentre la mínima cantidad de pasos temporales necesaria para converger, explicando claramente su implementación y por qué es correcto.

La complejídad del algoritmo debe ser $O(mn)$

¿Que pasa cuando converge?

¿Que pasa cuando converge?
Todos los numeros son iguales al máximo.

¿Cuanto tarda una posición en quedar igual al valor máximo?

¿Que pasa cuando converge?
Todos los numeros son iguales al máximo.

¿Cuanto tarda una posición en quedar igual al valor máximo?
Su distancia a uno de los valores máximos

¿Cuando son todos sean iguales?

¿Que pasa cuando converge?
Todos los numeros son iguales al máximo.

¿Cuanto tarda una posición en quedar igual al valor máximo?
Su distancia a uno de los valores máximos

¿Cuando son todos sean iguales?
Cuando el más lejano se convierta en el máximo

Necesitamos encontrar la distancia de los puntos a los máximos.

Necesitamos encontrar la distancia de los puntos a los máximos. Corro BFS desde todos los máximos a cada posición. Tengo que hacer desde todos lo máximos a cada punto.

Necesitamos encontrar la distancia de los puntos a los máximos. Corro BFS desde todos los máximos a cada posición. Tengo que hacer desde todos lo máximos a cada punto.

La distancia máxima va a ser el tiempo de convergencia.

Podemos definir el grafo donde cada posición es un verice y las arístas unen vertices adyacentes.

Agregamos un vertice extra, del cual salen aristas hacia los máximos.

Hacemos BFS desde ese vértice y luego restamos la distancia en 1.

La cantidad de vértices es $O(mn)$.

Como cada vertice tiene a lo sumo 8 vecinos, la cantidad de aristas también es $O(mn)$.

La complejidad es $O(V + E) = O(mn)$

# Problema 4

Dado un grafo pesado y conexo $G$, queremos construir un grafo $H$ que tenga un árbol generador cuya arista de peso máximo tenga peso exactamente $k$, cambiando los pesos de algunas aristas de $G$.
El costo de cambiar el peso $w_G(e)$ que una arista $e$ tiene en $G$ por el peso $w_H(e)$ que $e$ tiene en $H$ es $|w_H(e) - w_H(e)|$. Por lo tanto, el costo de construir $H$ es

$$\sum_{e \in E(G)} |w_H(e) - w_H(e)|$$

Tenemos a nuestra disposición un algoritmo que puede construir un AGM de un grafo con $m$ aristas en $O(m\alpha^{-1}(m))$ tiempo.
Diseñar un algoritmo de tiempo $O(m\alpha^{-1}(m))$ que, dado $G$ y $k \in \mathbb{N}$, encuentre el grafo pesado $H$ que tenga una arista de peso $k$ y cuyo costo de construcción sea mínimo.

# Solución

Sea $T$ el AGM que obtenemos al utilizar el algoritmo que nos dan.

Tenemos los siguientes casos:

Si la máxima arista de $T$ tiene peso $k$, no tengo que cambiar el peso de ninguna arista y el costo es 0.

Si el peso de arista máxima es menor a $T$, tendríamos que cambiar alguna arista de $T$ o el peso de una. Tomo la arista de $e \in G$ con peso más cercano a $k$.

Si $e \notin T$, se lo agrego. Ahora $T$ tiene tengo un ciclo que contiene a $e$. Elimino otra arista del ciclo formado para obtener nuevamente un árbol.

Ahora solo cambiamos el peso de $e$ a $k$. Claramente es el mínimo porque necesitamos cambiar alguna arista y estamos cambiando la arista que nos cuesta menos.

El último caso sería que que hayan aristas con peso mayor a $k$. Propongo cambiar el peso de esas aristas a $k$.

Veamos que es una solución que minimiza el costo total.

Sea $U$ el árbol generador que minimiza el costo.

Miremos las componentes conexas del grafo que se obtiene de mirar solo las aristas de peso menor a $k$. Dos puntos de la misma componente están conectadas por aristas de peso menor o igual a $k$ tanto en $T$ como en $U$.

Notar que podríamos agregar las aristas del camino una a una, eliminando siempre la arista de costo máximo cuando formamos un ciclo. Si hacemos esto, eliminamos alguna arista de coso mayor a $k$, lo que implica bajar el peso de $T$ o el costo de $U$

Observemos que la cantidad de aristas de peso mayor a $k$ en $T$ y $U$ es igual a $n-1$ menos la cantidad de componentes conexas y además tienen que sumar lo mismo.

Si las de $T$ pesaran menos, podríamos reemplazar las de $U$ por ellas y bajar el costo con lo que $U$ no sería el óptimo.

Si las de $U$ pesaran menos, podriamos cambiar las de $T$ por ellas, bajando el peso de $T$, y por lo tanto no sería *AGM*

Luego cambiar las aristas mayores a $k$ en $T$ nos da una solución óptima, ya que cuesta lo mismo que $U$, que es óptimo

# Algoritmo

1. Obtenemos un AGM usando el algoritmo en $O(m\alpha^{-1}(m))$
2. Si la arista máxima pesa $k$, termino
3. Si la arista máxima pesa menos $k$, busco la arista más con el peso más cercano a $k$ en $G$, y la reemplazo por $k$
4. Si no, hay aristas que pesan mas que $k$ y alcanza con cambiar sus peso a $k$

**Costo total:** $O(m\alpha^{-1}(m))$