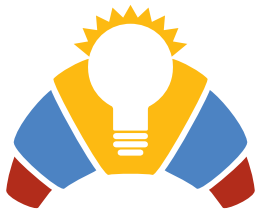
Programación Dinámica

De la Fuerza Bruta al Estado Justo y Necesario

Ivo Pajor

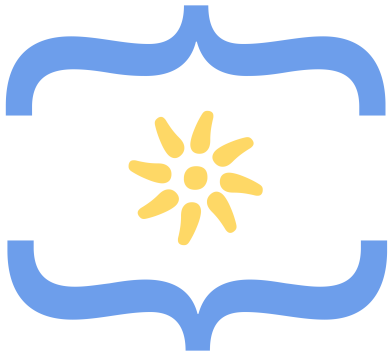
Universidad de Buenos Aires

Training Camp 2026



Gracias Sponsors!

Organiza



Facultad de INFORMÁTICA

UNIVERSIDAD NACIONAL DE LA PLATA

Diamond Sponsor

Outline

Outline

Cuchau

- Primero armo un **backtracking**.
- Miro la recursión: ¿qué información **necesita** para decidir el futuro?
- Agrupo estados que, para el futuro, son **indistinguibles**.
- Ese agrupamiento *es* la DP.

Problema

Another Filling the Grid (Codeforces 1228E)

Dados n y k, hay que llenar una grilla de $n \times n$ con enteros en $[1, k]$ tal que:

- el mínimo de **cada fila** sea exactamente 1,
- el mínimo de **cada columna** sea exactamente 1.

Contar la cantidad de formas de llenarla, módulo $10^9 + 7$.

- $1 \leq n \leq 250$
- $1 \leq k \leq 10^9$

Ejemplo

Con $n = 3$, $k = 3$, una grilla válida (celdas con 1 resaltadas):

1	2	1
2	1	3
3	3	1

Cada fila y cada columna tiene, entre sus valores, al menos un 1.

Backtracking

S: columnas que ya vieron un 1 en alguna fila anterior.

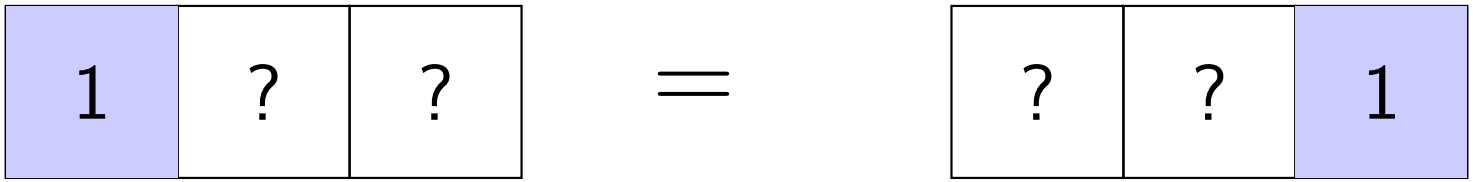
Algorithm 1 COUNT(fila i, S)

if $i = n + 1$ **then**
  return $[S = \{1, \ldots, n\}]$
end if
$total \leftarrow 0$
for cada forma de llenar la fila i con al menos un 1 **do**
  $T \leftarrow$ columnas que quedaron en 1 en esta fila
  $total \leftarrow total +$ (formas de llenar el resto de la fila) $\cdot$ COUNT$(i + 1,\ S \cup T)$
end for
return $total$

El problema

S toma hasta 2^n valores distintos: exponencial.

Información Necesaria



$S = \{1\}$ $S = \{3\}$

$\textsc{Count}(i, \{1\})$ y $\textsc{Count}(i, \{3\})$ cuentan exactamente lo mismo: renombrar columnas manda una situación a la otra.

- $\textsc{Count}$ nunca necesita saber *cuáles* columnas están en S, sólo **cuántas**.

Idea clave

Cambiamos el estado $S \subseteq \{1, \ldots, n\}$ por un simple número $j = |S|$.

La DP compacta

$dp[i][j]$: formas de llenar las primeras i filas dejando j columnas abiertas.

$$dp[0][n] = 1$$

Si a la fila $i+1$ llegan j columnas abiertas, y $j - j'$ de ellas reciben un 1 en esa fila:

$$dp[i+1][j'] \mathrel{+}= dp[i][j] \cdot \binom{j}{j-j'} (k-1)^{j'} k^{n-j}, \qquad j' < j$$

$$dp[i+1][j] \mathrel{+}= dp[i][j] \cdot (k-1)^{j} \left(k^{n-j} - (k-1)^{n-j}\right)$$

Respuesta: $dp[n][0]$.

Complejidad

$O(n^2)$ estados (i, j), cada uno con una transición $O(n)$ (la suma en j'): $O(n^3)$ en total.

La fórmula secreta de la Cangre Burguer

1. Backtracking correcto.
2. ¿Qué parte del estado **lee** la recursión para decidir el futuro?
3. Agrupar estados equivalentes (simetría: sólo importa una cantidad, no la identidad).
4. Redefinir la DP sobre ese estado compacto.

Outline

Mismo problema, otra construcción

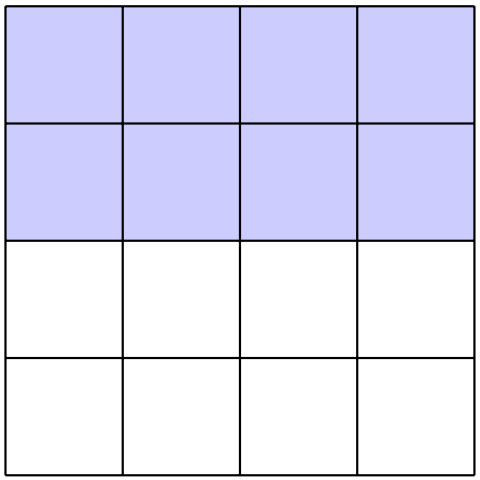
Volvamos a la grilla $n \times n$ (mínimo 1 en cada fila y columna).

- Llenándola **fila por fila** llegamos a $dp[i][j]$: polinomial.
- ¿Qué pasa si la construimos en **otro orden**, por ejemplo diagonal por diagonal?

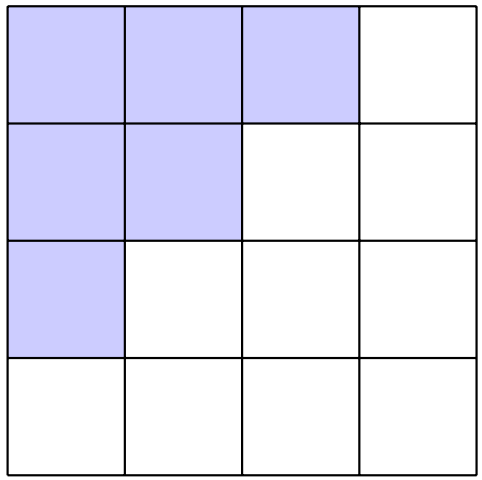
La pregunta

Mismos objetos para construir, misma cuenta final: ¿el orden en que los construimos cambia la complejidad?

Dos formas de construir la grilla



fila por fila



diagonal por diagonal

Celda celeste = ya decidida.

¿Por qué importa la forma de la frontera?

- **Fila por fila**: en cualquier momento, toda fila procesada está **completa** y el resto **vacía**. Alcanza con saber cuántas columnas ya vieron un 1.
- **Diagonal por diagonal**: cada fila queda con una cantidad **distinta** de celdas decididas, y no sabemos si ya tiene su 1. No hay forma evidente de resumir eso en un solo número.

Conclusión

Mismo problema, mismos objetos a construir: el orden de construcción decide si el estado compacta o no.

Outline

Problema

PolandBall and Gifts (Codeforces 755F)

n bolitas se reparten regalos según una permutación p sin puntos fijos: la bolita i le da un regalo a la bolita p_i. Se elige que k bolitas se olviden el regalo. Una bolita se queda **sin regalo** si ella se olvidó el suyo, o si quien le tenía que regalar a ella se olvidó.
Hallar, sobre todas las formas de elegir esas k bolitas, el mínimo y el máximo número de bolitas sin regalo. $n \leq 10^6$.

La idea

- p se descompone en ciclos (no tiene puntos fijos); la suma de los largos de los ciclos es exactamente n.
- Juntar a las olvidadizas dentro de un mismo ciclo, **completo**, no contagia a nadie fuera de ese ciclo.
- Minimizar se reduce a: ¿existe un subconjunto de ciclos cuyos largos sumen exactamente k?

Pocos valores distintos

Observación clave

Si un multiconjunto de enteros positivos tiene suma total (con repetición) a lo sumo N, entonces tiene a lo sumo $O(\sqrt{N})$ valores **distintos**.

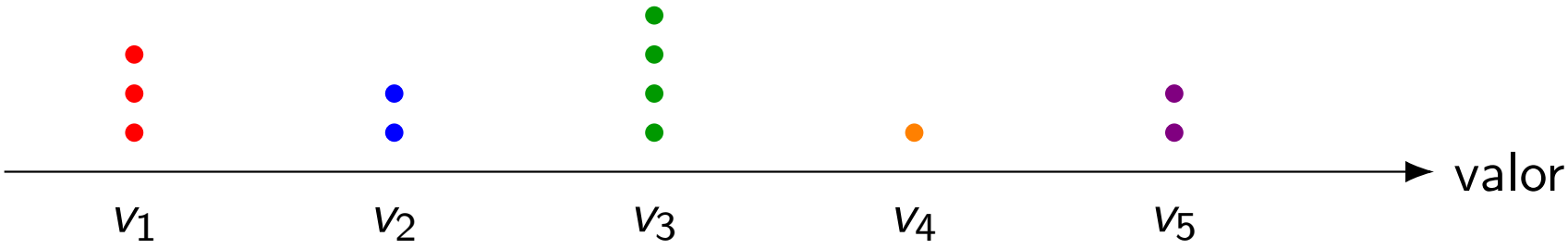
Si hay k valores distintos $v_1 < v_2 < \cdots < v_k$, entonces $v_i \geq i$, así que

$$\sum v_i \geq 1 + 2 + \cdots + k = \frac{k(k+1)}{2}.$$

Como la suma total es a lo sumo N:

$$\frac{k(k+1)}{2} \leq N \quad \Longrightarrow \quad k = O(\sqrt{N}).$$

Agrupando por valor



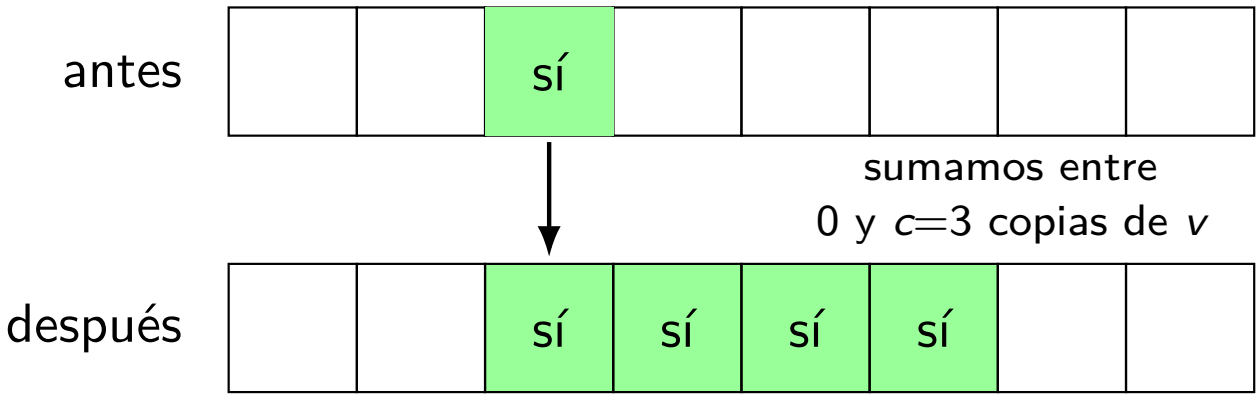
Muchas copias, pocos valores distintos: a lo sumo $O(\sqrt{N})$ grupos. En nuestro problema, los largos de ciclo suman n: sólo hay $O(\sqrt{n})$ largos distintos.

Cerrando la idea

- Agrupamos por largo de ciclo v, con multiplicidad c: usamos entre 0 y c copias de v en la suma.
- Cómo hacer mal las cosas: procesar las c copias una por una, cada una en $O(n) \Rightarrow O(n \cdot c)$.
- Vamos a resolver el grupo **completo** en $O(n)$, sin importar c.

La ventana deslizante

Fijamos un resto $r = s \bmod v$: las sumas con ese resto son la progresión $r,\ r+v,\ r+2v, \ldots$



Un "sí" en la posición p vuelve alcanzables $p, p+1, \ldots, p+c$: eso es sumar entre 0 y c copias de v. Sumando los v restos (largo total $\approx n$): $O(n)$ para **todo** el grupo. Sumando los $O(\sqrt{n})$ grupos: $O(n\sqrt{n})$.

Outline

Problema

Suma de dígitos

Dados N y m, contar cuántos enteros x con $1 \leq x \leq N$ cumplen que la suma de los dígitos de x es múltiplo de m.

Una aclaración

- Muchos problemas de digit DP se pueden resolver de otras formas.
- Pero apenas la condición mezcla los dígitos entre sí (como la suma módulo m), digit DP pasa a ser la herramienta natural.
- Generaliza fácil a muchas variantes con poco esfuerzo extra.

Digit DP

$N[0..|N|-1]$: dígitos de N. *tight*: ¿todavía pegados al prefijo de N?

Algorithm 2 SOLVE(posición *pos*, *tight*, suma acumulada $s \bmod m$)

if $pos = |N|$ **then**
  return $[s = 0]$
end if
if no *tight* y $memo[pos][s]$ ya está calculado **then**
  return $memo[pos][s]$
end if
$total \leftarrow 0, \quad limite \leftarrow tight \ ? \ N[pos] : 9$
for $d = 0$ **to** *limite* **do**
  $total \leftarrow total + \text{SOLVE}(pos + 1, \ tight \wedge (d = limite), \ (s + d) \bmod m)$
end for
if no *tight* **then**
  $memo[pos][s] \leftarrow total$
end if
return *total*

Llamado inicial: SOLVE(0, true, 0).

¿Por qué no memorizar *tight*?

- Con $tight = \text{true}$ hay un único camino posible: el prefijo exacto de N. Son sólo $O(\log N)$ estados así, y cada uno se visita **una sola vez**.

Complejidad

- Estados: $O(\log N \cdot m)$, cada uno con 10 transiciones.
- Total: $O(10 \cdot \log N \cdot m)$, contra $O(N)$ de fuerza bruta.

Outline

Problema

GSS3 (SPOJ)

Un arreglo soporta actualizaciones puntuales y consultas $Query(x, y)$: máxima suma de un subarreglo contenido en $[x, y]$.

¿Esto es DP?

Para **todo** el arreglo, alcanza con Kadane:

$$best[i] = \max\big(a[i],\ best[i-1] + a[i]\big), \qquad \text{respuesta} = \max_i best[i]$$

Es una DP lineal: $best[i]$ sólo depende de $best[i-1]$.

La idea general

- A veces el valor de la DP en un rango se escribe como la **combinación (monoide)** de los valores en sub-rangos.
- Guardamos esa combinación en un segment tree: cada nodo mezcla a sus dos hijos.
- Actualizar un punto o consultar un rango: $O(\log n)$, sin recalcular toda la DP.

En GSS3, cada nodo guarda la tupla $(\mathit{total},\ \mathit{pref},\ \mathit{suf},\ \mathit{sub})$. Mezcla de dos hijos L, R:

$$\mathit{total} = \mathit{total}_L + \mathit{total}_R \qquad \mathit{pref} = \max(\mathit{pref}_L,\ \mathit{total}_L + \mathit{pref}_R)$$

$$\mathit{suf} = \max(\mathit{suf}_R,\ \mathit{total}_R + \mathit{suf}_L) \qquad \mathit{sub} = \max(\mathit{sub}_L,\ \mathit{sub}_R,\ \mathit{suf}_L + \mathit{pref}_R)$$

Outline

Problema

Camino de k vértices

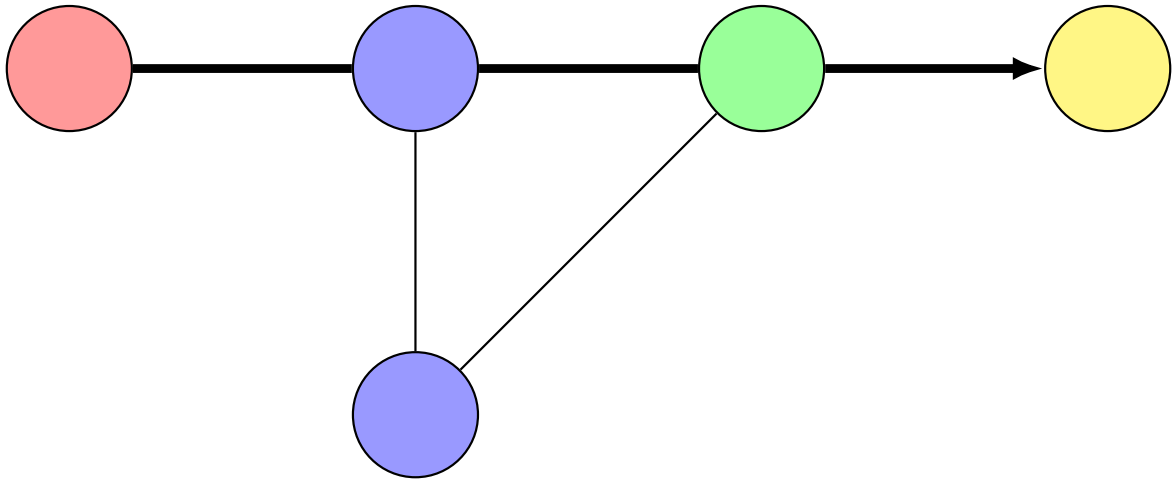
Dado un grafo y un entero k, decidir si existe un camino simple que visite k vértices distintos.

El estado natural, $dp[v]$[subconjunto visitado], es $O(2^n)$.

Un poco de aleatoriedad

- **Color Coding**: coloreamos cada vértice al azar con uno de k colores.
- Nuevo estado: $dp[v]$[máscara de colores usados], con sólo 2^k máscaras.

¿Por qué funciona Color Coding?



Buscamos sólo caminos **coloridos**: todos sus vértices con colores distintos (el camino resaltado lo es; pasar por d repetiría el azul de b).

¿Y si el camino no queda colorido?

- Un camino fijo de k vértices resulta colorido con probabilidad $\frac{k!}{k^k} = \Theta(e^{-k})$.
- Repetimos el coloreo al azar $\sim e^k$ veces para que la probabilidad de éxito sea alta.

Costo

$O(2^k \cdot k \cdot m)$ por intento: exponencial en k, no en n.

Consultas

Pueden enviar un mail a:

- pajorivo@gmail.com