Programación Dinámica Avanzada

Resumen de la clase

- Repaso DP - Estados
- Bitmask DP
- DP + Segment Tree / estructuras
- Rerooting
- Recurrencias lineales con matrices
- Recurrencias lineales con matrices de polinomios

Subproblemas y todo eso

- Cuando hablamos de programación dinámica, normalmente pensamos en recurrencias o agarrar problemas chicos y mezclarlos para resolver un problema grande.
- Una manera de formalizar esto, es pensando en términos de **estados**

Hablemos de estados

Pensemos en el siguiente problema:

Tenemos una grilla de $N \times M$ con algunas casillas bloqueadas y queremos llegar desde $(1, 1)$ hasta (N, M).

Hablemos de estados

Un **estado** es una forma de representar únicamente una configuración del problema mediante conjunto de variables. Por ejemplo, en este problema, podemos caracterizar cada configuración con la posición en las filas y las columnas; es decir, una tupla (i, j).

Otro ejemplo: CSES 1635

Hay un sistema monetario con N monedas con algún valor.
Dada una cantidad objetivo X, calcula el número de formas distintas de sumar X utilizando alguna cantidad de las monedas.

Otro ejemplo: CSES 1635

Aquí los estados son un poco más implícitos.
Podemos caracterizar cada configuración del problema según la suma que tengamos hasta el momento.

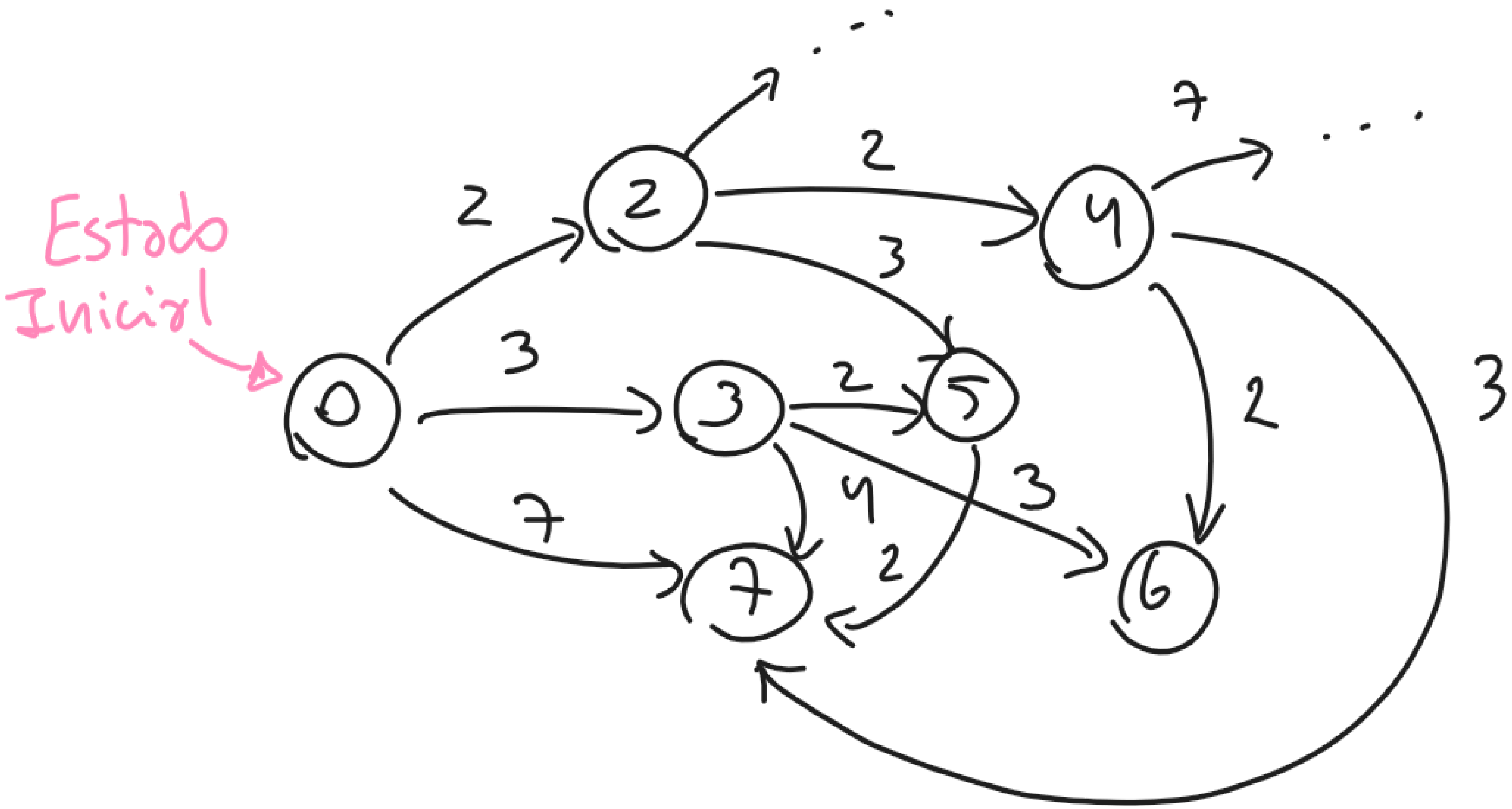
Ahora, **¿cómo nos movemos entre los estados?**

Transiciones

- Supongamos que tenemos las monedas $[2, 3, 7]$.
- Cuando tenemos $\$2$, por ejemplo, podemos escoger alguna de estas monedas y movernos al estado con $\$4$, $\$5$ o $\$9$. A esto se le llama **transición**: Movernos de un estado al otro.

¡El espacio de estados de un problema se puede modelar como un grafo!

Espacio de estados



Espacio de estados

Viéndolo de esta forma, el problemea de las monedas se reduce a contar caminos entre el estado inicial 0 y X (sabemos hacer eso!!1)

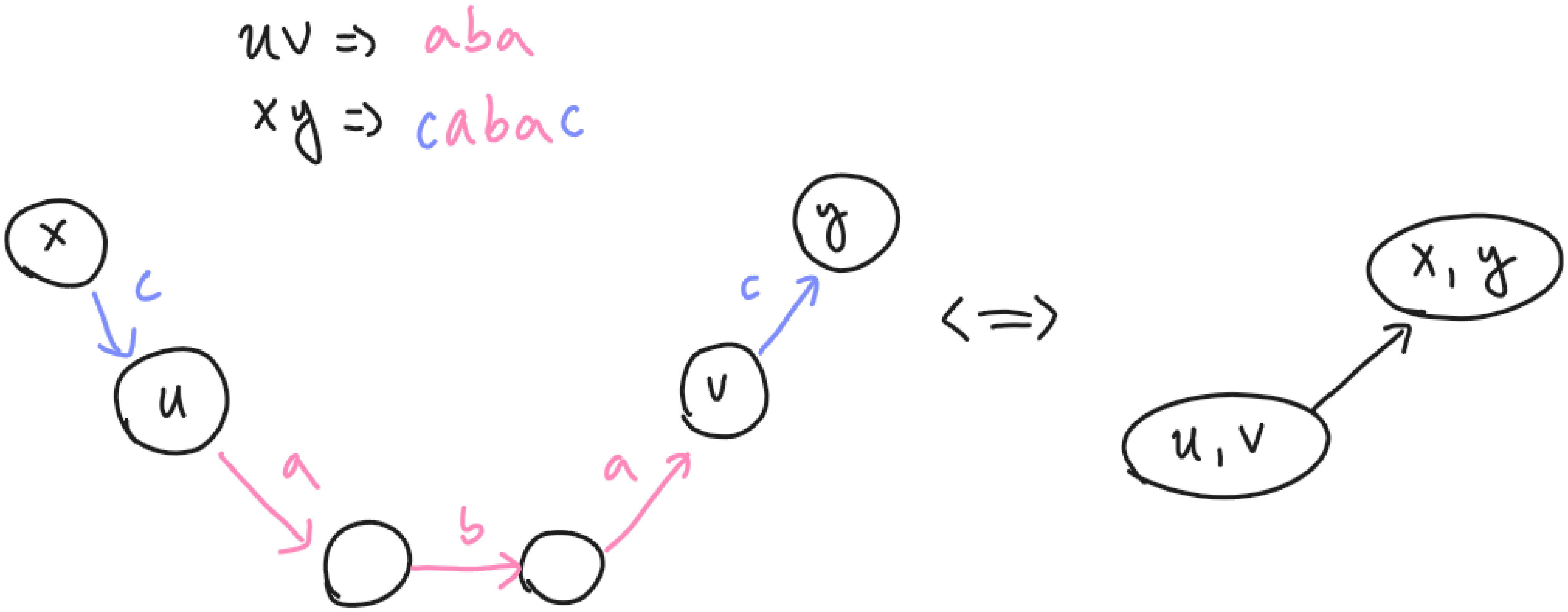
Muchas veces el grafo de estados es acíclico; es decir, no podemos visitar un estado dos veces. Sin embargo, hay casos en los que esto sí ocurre y no podemos hacer DP de forma estándar!

Ejemplo (PdA 2025 - Problem A):

Dado un grafo dirigido con caracteres en las aristas, contar el número de pares (u, v) tal que existe un paseo **palíndromo** entre u y v

Ejemplo (PdA 2025 - Problem A)

- Los palíndromos tienen una estructura increíblemente recursiva
- caracter + palíndromo + mismo caracter
- Podemos pensar en que cada tupla (u, v) es un estado: Tenemos un palíndromo formado entre u y v
- Si encontramos una arista entrante hacia u y otra saliente desde v v con el mismo caracter hacia vértices x, y respectivamente, podemos movernos al estado (x, y)! pues podemos expandir el palíndromo que ya teníamos
- Idea preliminar: $f(u, v) = \operatorname*{or}_{x \xrightarrow{c} u, v \xrightarrow{c} y} \{f(x, y)\}$
- Casos base $f(e.u, e.v) = f(u, u) = 1$

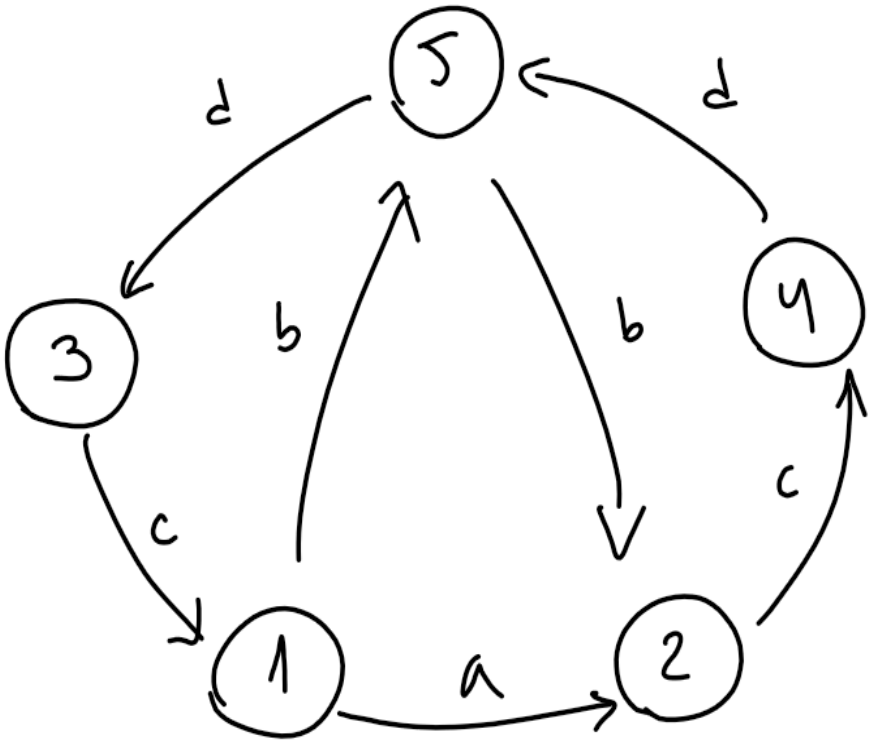


uv => aba
xy => cabac
x
c
u
a
b
a
v
c
y
<=>
x, y
u, v

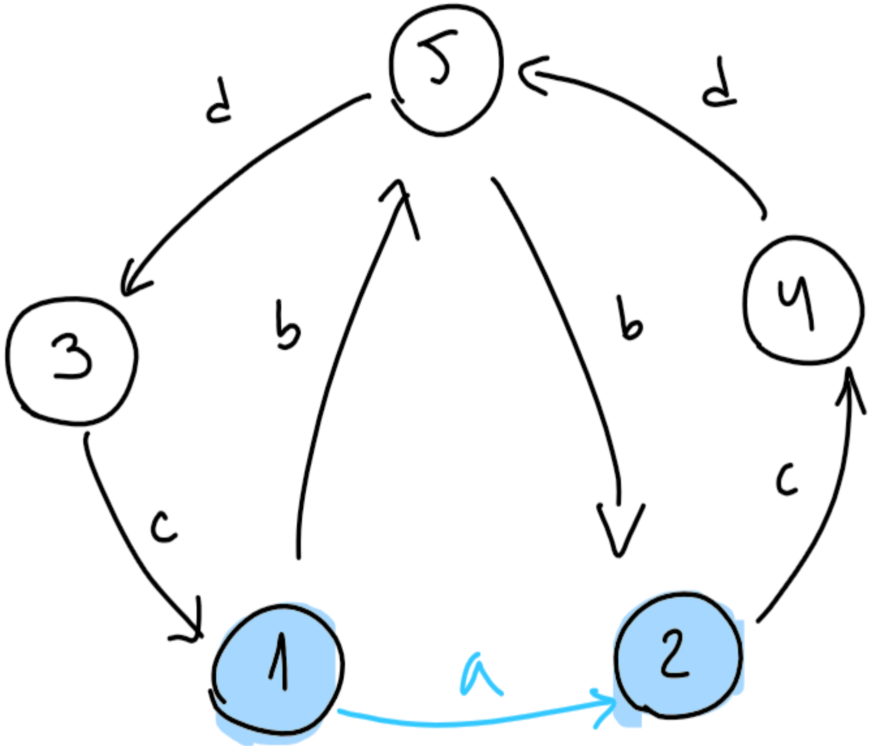
Ejemplo (PdA 2025 - Problem A)

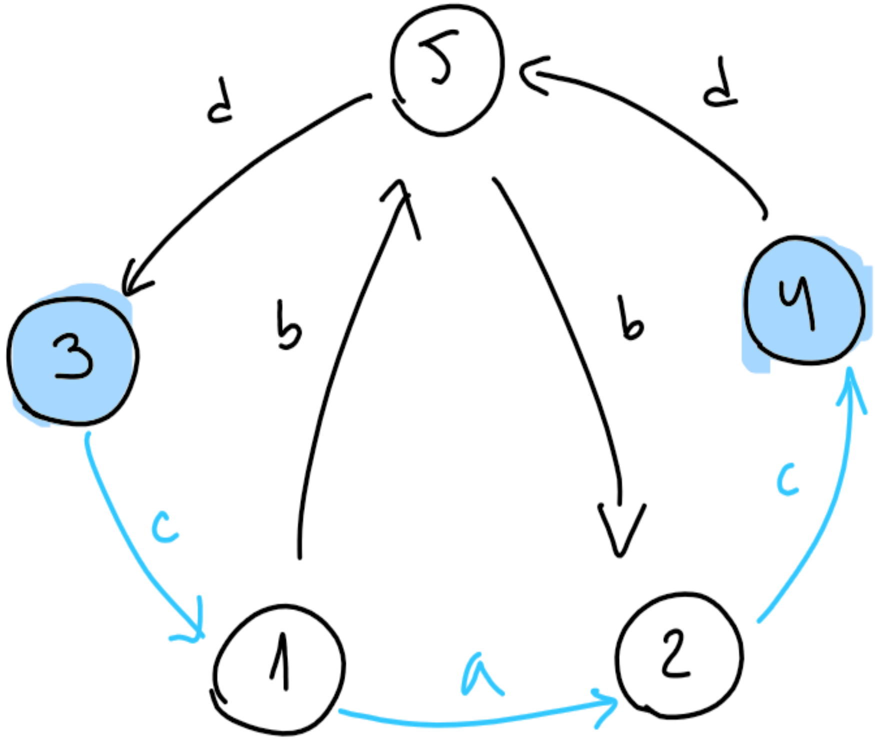
- Los palíndromos tienen una estructura increíblemente recursiva
- caracter + palíndromo + mismo caracter
- Podemos pensar en que cada tupla (u, v) es un estado: Tenemos un palíndromo formado entre u y v
- Si encontramos una arista entrante hacia u y otra saliente desde v v con el mismo caracter hacia vértices x, y respectivamente, podemos movernos al estado (x, y)! pues podemos expandir el palíndromo que ya teníamos
- Idea preliminar: $f(u, v) = \underset{x \xrightarrow{c} u, v \xrightarrow{c} y}{\text{or}} \{f(x, y)\}$
- Casos base $f(e.u, e.v) = f(u, u) = 1$
- **Problema: Pueden haber ciclos!**

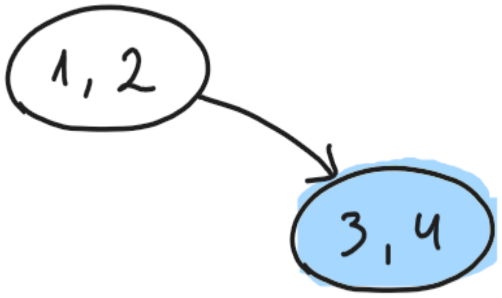
Caso feo

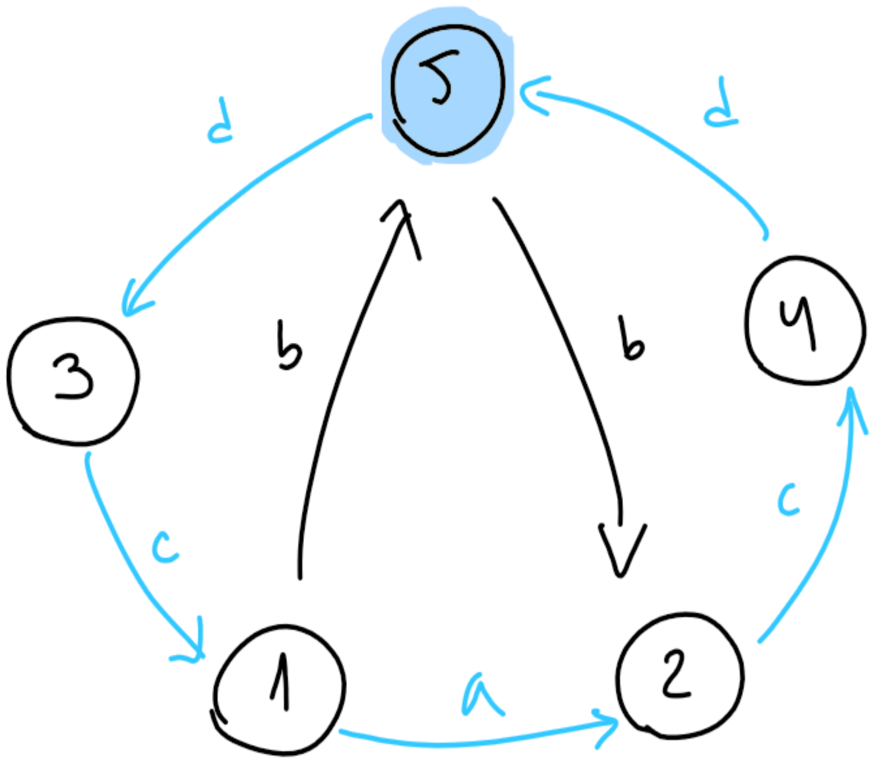


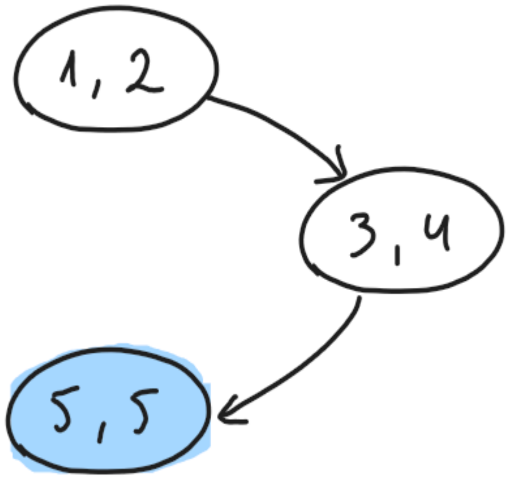
Caso feo



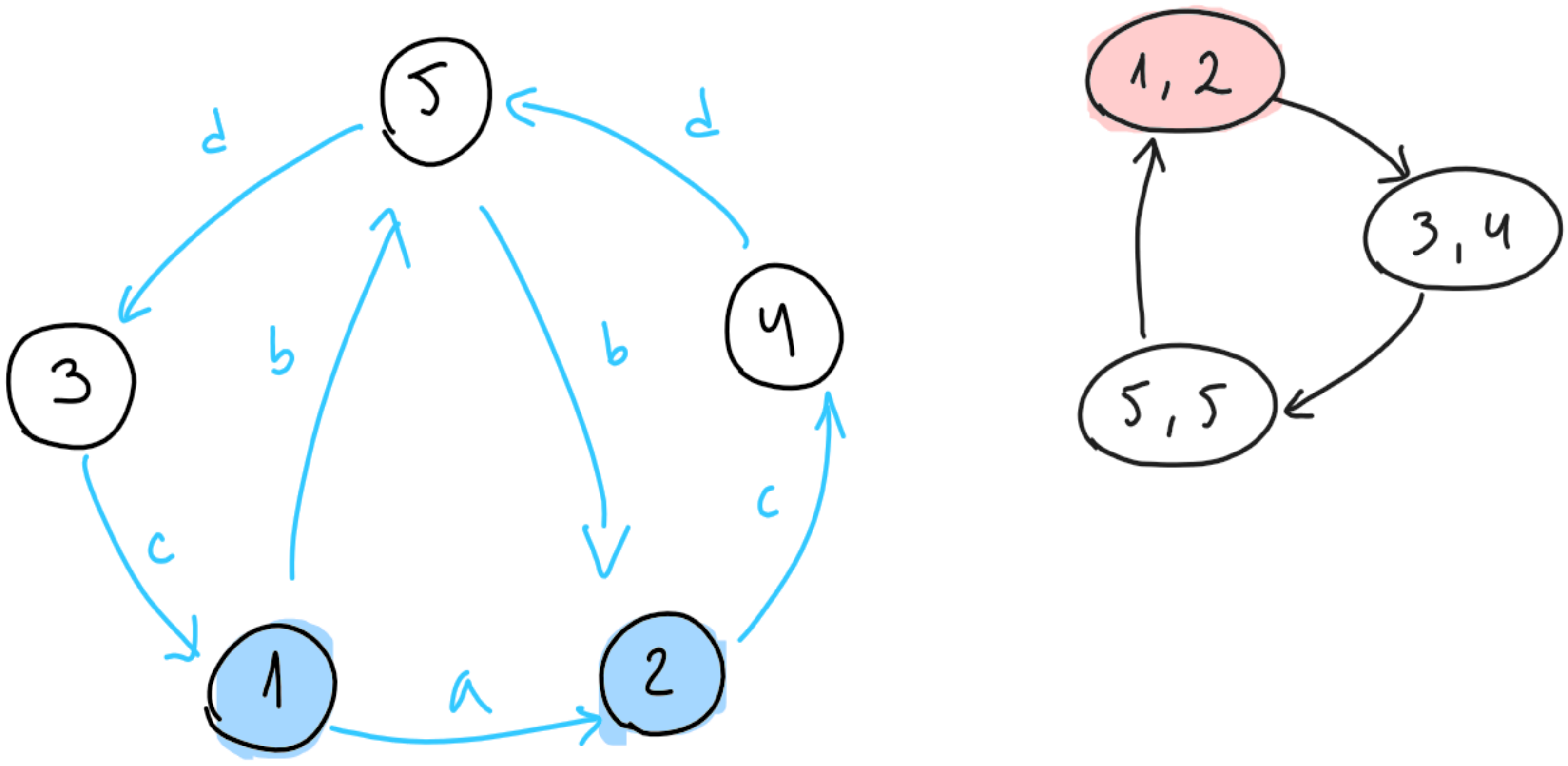

5
d
d
b
b
3
4
c
c
1
a
2


1, 2
3, 4


5
d
d
3
b
b
4
c
c
1
a
2


1, 2
3, 4
5, 5

Caso feo



Ejemplo (PdA 2025 - Problem A)

- Como tal no podemos hacer DP, pero podemos implementar esta misma idea revisando qué estados son **alcanzables** desde los casos base
- Las restricciones del problema no nos permiten construir el grafo, hay que hacer un bfs implícito pero la idea no cambia.

Ideas

- No hay una forma fácil de deducir los estados; en general hay muchas representaciones de un mismo problema
- Si tenemos alguna idea para los estados de un problema pero no entra en tiempo, tratar de simplificar información redundante.
- Para modelar los estados de un problema, construir una solución de a poquito y observar

Bitmask DP

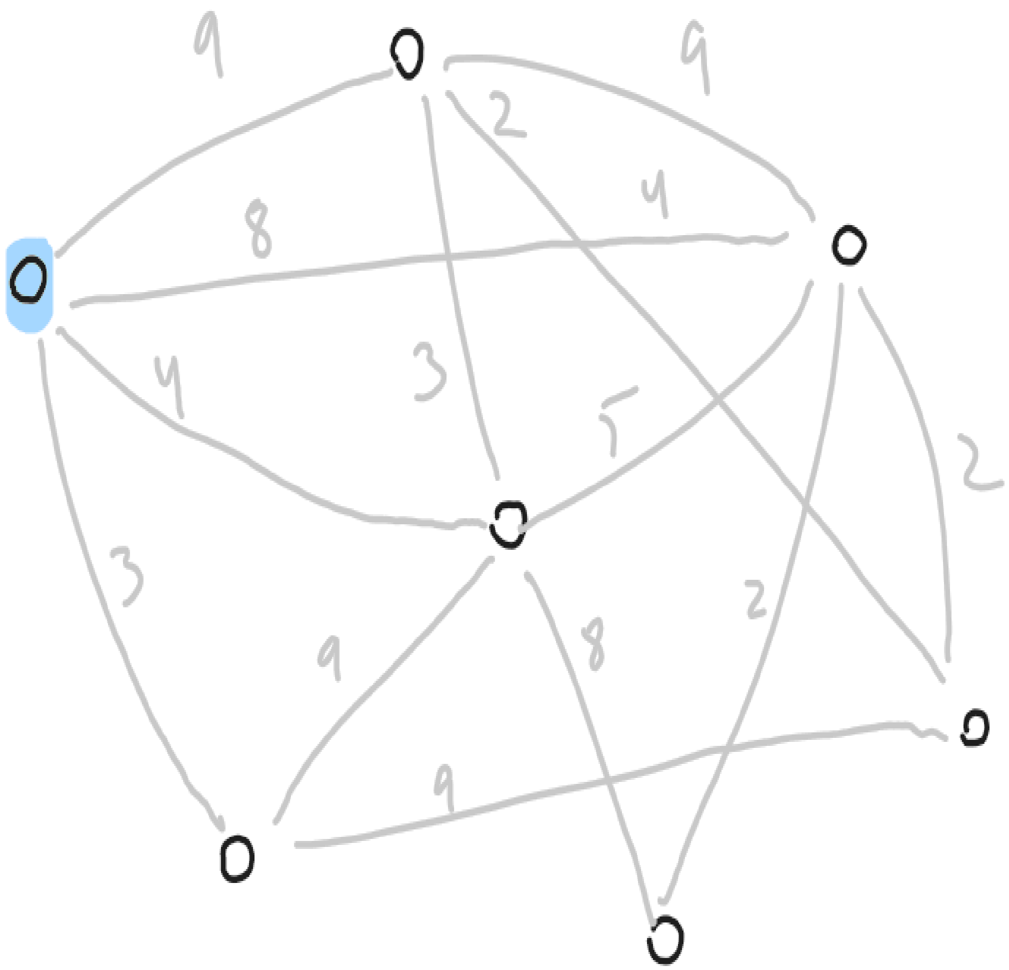
- Cuando las constraints son pequeñas, podemos usar cantidades exponenciales de estados
- Ejemplo: TSP (Problema del vendedor viajero)

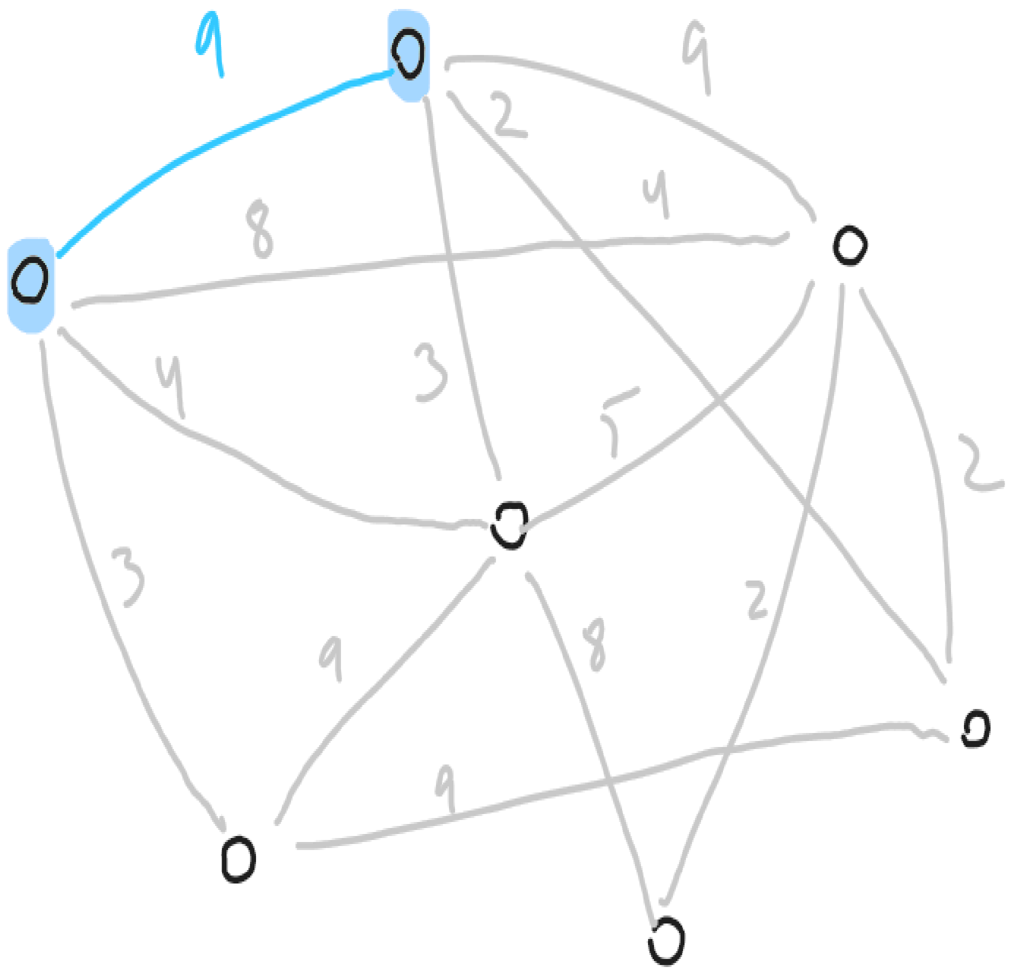
Enunciado:

Se te da un grafo de N nodos y M aristas con $N < 20$ y $M \leq \binom{N}{2}$. Calcula el costo de un camino hamiltoniano de costo mínimo.

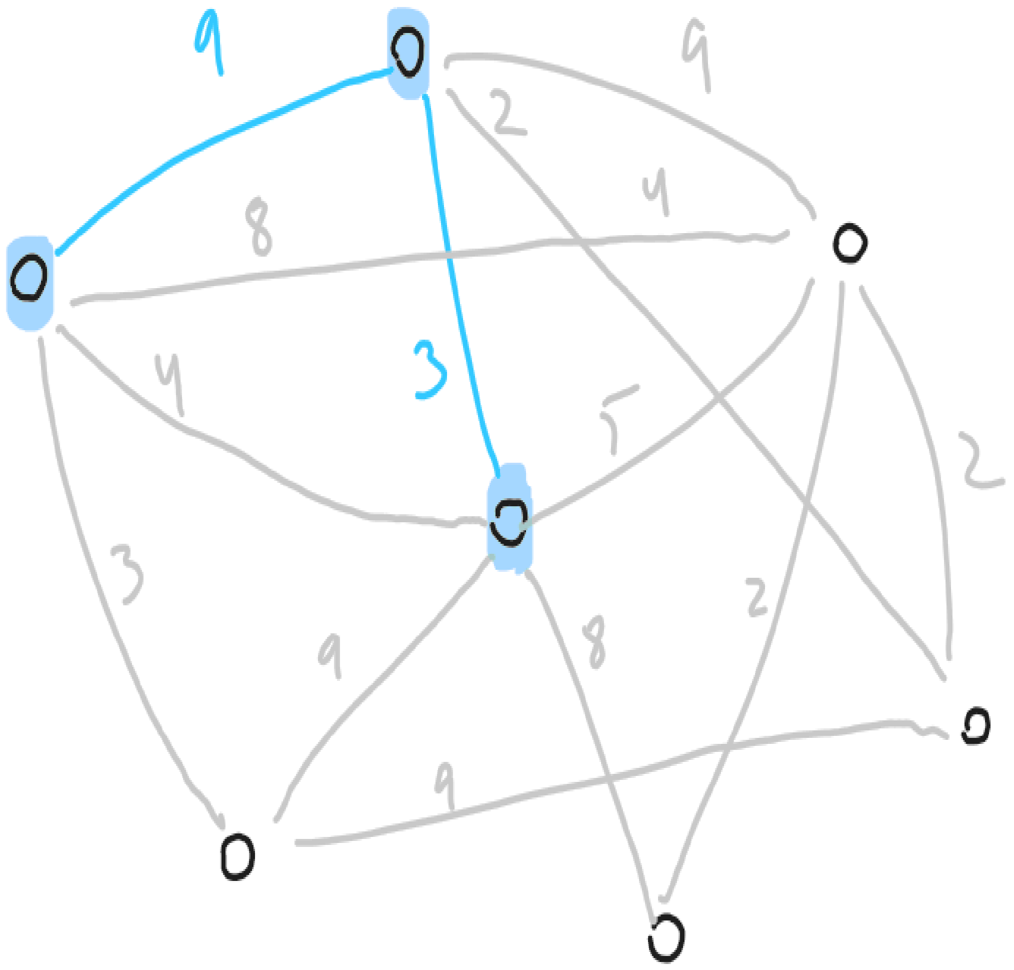
- Pensemos en generar una solución de a poco: Tomemos un nodo y empecemos a agregar vecinos al camino

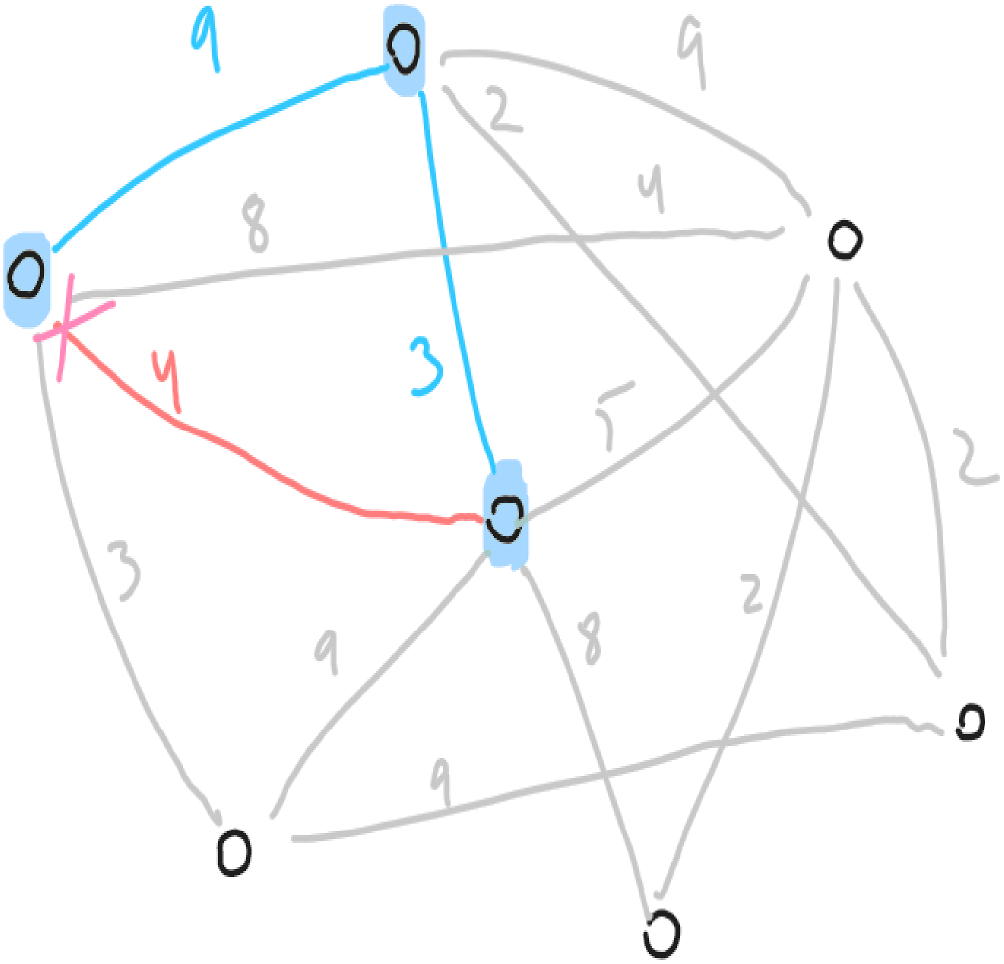
TSP

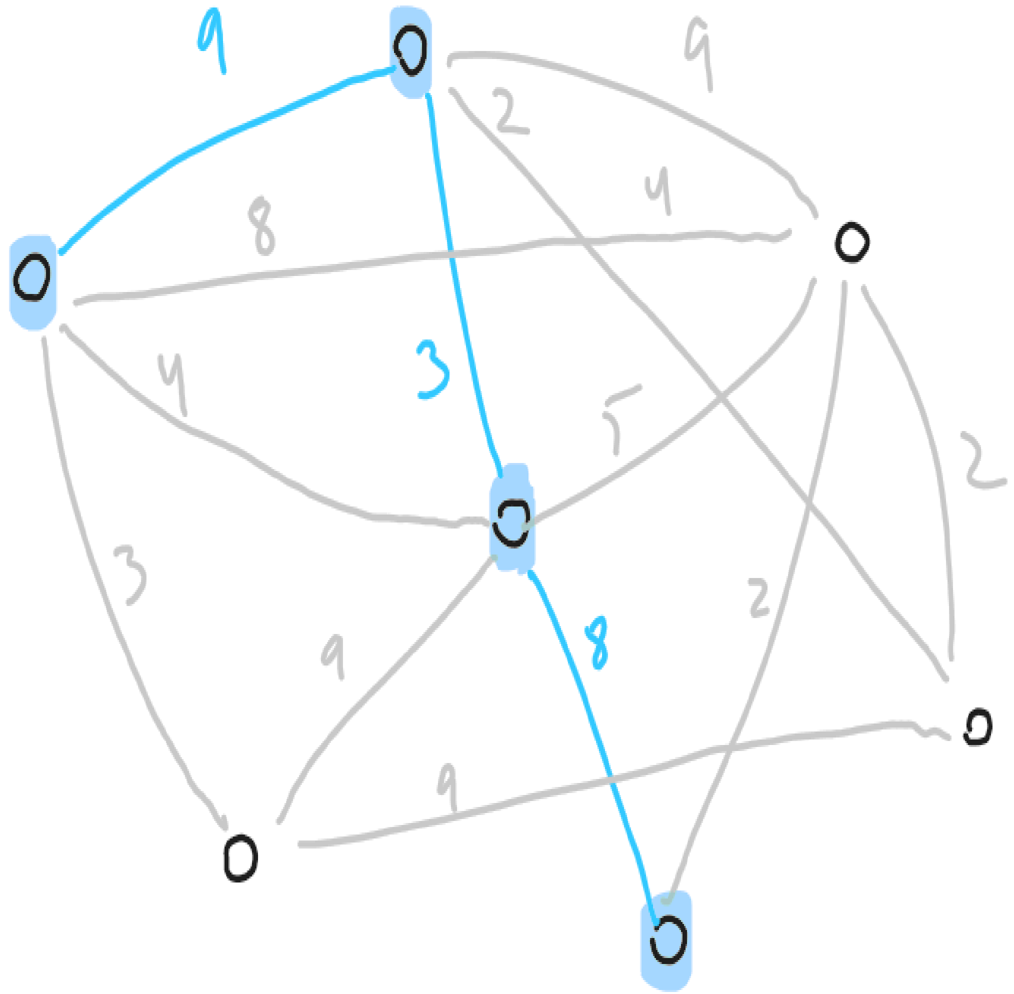



9
9
2
4
8
3
4
5
2
3
9
8
2
9

TSP




9
9
2
8
4
4
3
5
2
3
9
8
2
9


9
9
2
8
4
3
4
5
2
3
9
8
2
9

- Necesitamos saber qué nodos hemos visitado y cuál es nuestro nodo actual
- Hay $2^N \cdot N$ estados posibles!
- Pero N es chico, así que se puede hacer
- Las transiciones quedan así

$$dp(u, S) = \min_{v \in S \cap N(u)} \{dp(v, S - u) + w(u, v)\}$$

- y la respuesta es

$$\min_{U \in V} \{dp(u, V)\}$$

Ideas principales

- Esto es casi como hacer backtracking pero memoizando los estados
- Otro problema que aplica esta idea es este: https://codeforces.com/problemset/problem/1431/G

Backtracking + memo

Alicia y Bob están jugando un juego. Parten con un set de N enteros y juegan por K turnos. En cada turno:

- Alicia escoge un número del conjunto **distinto del máximo** a
- Bob escoge un número más grande que a_i, llamémoslo b
- a y b son borrados del set y se suma $b - a$ al puntaje.

Si Alicia quiere maximizar su puntaje y Bob lo quiere minimizar, calcule el puntaje resultante si juegan óptimamente.

Restricciones

$$n \leq 400$$
$$k \leq \frac{N}{2}$$

Enunciado:

Vi le regaló a Arias un arreglo con $N \leq 3 \cdot 10^5$ elementos para su cumpleaños y Arias quiere encontrar la subsecuencia creciente más larga pero no quiere implementar porque perdió todo en el casino así que te pidió ayuda o algo así como ponen en codeforces

Solución subóptima

$$dp(i) = \max_{\substack{j<i \\ A_j<A_i}} \{dp(j) + 1\}$$

- Corre en $\mathcal{O}(N^2)$
- Podemos hacerlo mejor?

Solución optimizada

- Recorramos el arreglo de izquierda a derecha
- Si sólo consideramos los elementos colocados hasta el momento, la restricción $j < i$ es trivial
- Sólo debemos revisar el máximo sobre todos los j donde $A_j \leq A_i$. Cómo podemos hacer esto eficientemente?

Solución optimizada

- Guardaremos en un arreglo B lo siguiente:
- Si existe una subsecuencia creciente de tamaño K que termina en un valor x, actualizaremos $B_x = \max(B_x, K)$
- Calcular $dp(i)$ se convierte en encontrar el máximo en el rango $[0, A_i - 1]$
- Necesitamos actualizar el arreglo y responder estas queries rápidamente
- Respuesta: Segment tree
- Complejidad final: $\mathcal{O}(N \log N)$

Enunciado:

Tienes N bloques y quieres colocarlos en niveles $X_1 \geq X_2 \geq \ldots$ tal que cada nivel esté encima del otro.
De cuántas formas se puede hacer esto? (imprimir módulo $10^9 + 7$)

Es realmente $\mathcal{O}(N^3)$?

- Usamos dp (omg fork found in kitchen)
- $dp(n, k)$ cuenta el número de formas de colocar n bloques donde el piso de abajo tiene k bloques

$$dp(n, k) = \sum_{j=1}^{k} dp(n - j, j) \cdot (k - j + 1)$$

- Formas de colocar el piso de largo j encima del piso de largo k $\times$ formas de colocar el resto de bloques cuando el piso tiene largo j

Es realmente $\mathcal{O}(N^3)$?

$$dp(n,k) = \sum_{j=1}^{k} dp(n-j,j) \cdot (k-j+1)$$

$$dp(n,k) = (k+1)\sum_{j=1}^{k} dp(n-j,j) - \sum_{j=1}^{k} dp(n-j,j) \cdot j$$

$$dp(n,k) = (k+1)S(n,k) + S_2(n,k)$$

- Las sumatorias se pueden calcular al mismo tiempo que la dp
- En total queda $\mathcal{O}(N^2)$

Rerooting

- Hay dps en árboles que dependen de la raíz
- En algunos problemas debemos calcular algo de este estilo para **todas** las raíces
- Cómo lo hacemos sin que nos quede cuadrático?

Rerooting - Idea superficial

- Mover la raíz a un nodo adyacente es muy barato
- Movemos la raíz en orden DFS y queda $\mathcal{O}(N)$
- Sólo hay que recalcular dos valores

Problema: Tree Painting

Jugamos un juego en un árbol de $N \leq 2 \cdot 10^5$ vértices. En el primer turno, escogemos un vértice y lo pintamos de negro. En los turnos siguientes, escogemos un vértice **adyacente** a algún vértice negro y lo pintamos, sumando a nuestro puntaje el tamaño del componente conexo de vértices blancos al que pertenecía.

Cuál es el puntaje máximo que se puede obtener?

Link: https://codeforces.com/contest/1187/problem/E

Recurrencias lineales

Una recurrencia lineal es una recurrencia donde el n-ésimo término es una combinación lineal de terminos anteriores.

Ejemplos:

- $F_n = F_{n-1} + F_{n-2}$
- $A_n = 9A_{n-1} + 9A_{n-4} - 8A_{n-5}$

Esto significa que podemos escribirlas utilizando matrices!

Ejemplo: Fibonacci

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix} = \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix}$$

Recurrencias lineales

$$A_n = 9A_{n-1} + 9A_{n-4} - 8A_{n-5}$$

$$\begin{bmatrix} A_n \\ A_{n-1} \\ A_{n-2} \\ A_{n-3} \\ A_{n-4} \end{bmatrix} = \begin{bmatrix} 9 & 0 & 0 & 9 & -8 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{n-1} \\ A_{n-2} \\ A_{n-3} \\ A_{n-4} \\ A_{n-5} \end{bmatrix}$$

Fibonacci revisited

Queremos calcular F_n, el n-ésimo número de fibonacci.

- Recurrencia sin memoizar: $\mathcal{O}(2^n)$
- DP normal: $\mathcal{O}(n)$

Se puede hacer mejor?

- Exponenciación binaria: $\mathcal{O}(\log n)$

Fibonacci revisited

Si tenemos el vector con los valores $[F_n, F_{n-1}]$, al multiplicarlo por la matriz M de recién **shifteamos** los valores en uno:

$$[F_n, F_{n-1}] \rightarrow [F_{n+1}, F_n]$$

Intuitivamente, multiplicar por M^2 los shiftea en 2 y así sucesivamente. Se tiene que

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^k \begin{bmatrix} F_1 \\ F_0 \end{bmatrix} = \begin{bmatrix} F_{k+1} \\ F_k \end{bmatrix}$$

Fibonacci revisited

Calcular la k-ésima potencia de una matriz de $n \times n$ tiene complejidad $\mathcal{O}(n^3 \log k)$:

- Multiplicar matrices tiene complejidad $\mathcal{O}(n^3)$
- En total se hacen $\mathcal{O}(\log k)$ multiplicaciones

En conclusión, como la matriz es de 2×2, hacemos aproximadamente $8 \cdot \log k$ operaciones.

- Curiosidad: Existe una forma de hacer algo parecido sin matrices para los números de fibonacci que quita la constante, pero no generaliza.

Problema con la misma idea

Estás peleando contra un boss con N de vida y tienes M hechizos distintos representados por una tupla (a_i, b_i), que significa que el hechizo quita a_i de vida y sólo se puede usar si los últimios b_i bits de la vida restante son cero y la vida restante es mayor o igual a a_i.

De cuántas formas se puede hacer que N llegue a cero? (módulo 10^7)

Restricciones:

$N \leq 10^{18}$
$M \leq 10^5$
$a_i \leq 100$
$b_i \leq 60$

Link: https://codeforces.com/gym/105327/problem/D

Recurrencias lineales con polinomios

Se puede usar esta misma idea cuando las recurrencias son bidimensionales?

Link: https://codeforces.com/gym/103960/problem/K