Permutaciones

Conceptos y técnicas comunes sobre permutaciones

Mateo Carranza Velez

Universidad Nacional de Córdoba - FAMAF

Training Camp 2025

Gracias Sponsors!

Organizador

Diamond Plus

Gracias Sponsors!

Platino

FOLDER IT

INTERNATIONAL
SOFTWARE COMPANY

Gold

NeuralSoft

Oro

JS JERÁRQUICOS

Gracias Sponsors!

Aliado

¿Qué es una permutación?

- Es un arreglo de tamaño n, en el que aparecen todos los enteros entre 1 y n exactamente una vez.

¿Qué es una permutación?

- Es un arreglo de tamaño n, en el que aparecen todos los enteros entre 1 y n exactamente una vez.
- Por ejemplo, `[2,3,1]` y `[5,2,4,1,3]` son permutaciones, pero `[1,3,1]` y `[5,2,4,1]` no lo son.

¿Qué es una permutación?

- Es un arreglo de tamaño n, en el que aparecen todos los enteros entre 1 y n exactamente una vez.
- Por ejemplo, `[2,3,1]` y `[5,2,4,1,3]` son permutaciones, pero `[1,3,1]` y `[5,2,4,1]` no lo son.
- También las podemos pensar como funciones, la segunda permutación corresponde a la función $p : \{1, 2, 3, 4, 5\} \to \{1, 2, 3, 4, 5\}$ tal que $p(1) = 5$, $p(2) = 2$, $p(3) = 4$, $p(4) = 1$ y $p(5) = 3$.

¿Qué es una permutación?

- Es un arreglo de tamaño n, en el que aparecen todos los enteros entre 1 y n exactamente una vez.
- Por ejemplo, `[2,3,1]` y `[5,2,4,1,3]` son permutaciones, pero `[1,3,1]` y `[5,2,4,1]` no lo son.
- También las podemos pensar como funciones, la segunda permutación corresponde a la función $p : \{1, 2, 3, 4, 5\} \to \{1, 2, 3, 4, 5\}$ tal que $p(1) = 5$, $p(2) = 2$, $p(3) = 4$, $p(4) = 1$ y $p(5) = 3$.
- También podemos escribirlas con la notación de dos filas:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 2 & 4 & 1 & 3 \end{pmatrix}$$

Inversa de una permutación

- Para una permutación p su inversa (que llamaremos p^{-1}) es la permutación que nos indica para cada elemento, en qué posición de la permutación original está.

Inversa de una permutación

- Para una permutación p su inversa (que llamaremos p^{-1}) es la permutación que nos indica para cada elemento, en qué posición de la permutación original está.
- Por ej, para $p = [5,2,4,1,3]$ su inversa es $p^{-1} = [4,2,5,3,1]$. Observar que $p^{-1}[1] = 4$ porque el 1 está en la posición 4 de p.

Inversa de una permutación

- Para una permutación p su inversa (que llamaremos p^{-1}) es la permutación que nos indica para cada elemento, en qué posición de la permutación original está.
- Por ej, para $p = [5,2,4,1,3]$ su inversa es $p^{-1} = [4,2,5,3,1]$. Observar que $p^{-1}[1] = 4$ porque el 1 está en la posición 4 de p.
- Para computarla en $\mathcal{O}(n)$ es útil ver que $p[i] = j \iff p^{-1}[j] = i$.

Inversa de una permutación

- Para una permutación p su inversa (que llamaremos p^{-1}) es la permutación que nos indica para cada elemento, en qué posición de la permutación original está.
- Por ej, para $p = [5,2,4,1,3]$ su inversa es $p^{-1} = [4,2,5,3,1]$. Observar que $p^{-1}[1] = 4$ porque el 1 está en la posición 4 de p.
- Para computarla en $\mathcal{O}(n)$ es útil ver que $p[i] = j \iff p^{-1}[j] = i$.
- En la notación de dos filas, si ordenamos la fila de abajo, arriba nos queda la inversa:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 2 & 4 & 1 & 3 \end{pmatrix} \longrightarrow \begin{pmatrix} 4 & 2 & 5 & 3 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

Composición de permutaciones

- La composición de dos permutaciones p y q (que llamaremos $p \circ q$) es el resultado de componer p y q como funciones.

Composición de permutaciones

- La composición de dos permutaciones p y q (que llamaremos $p \circ q$) es el resultado de componer p y q como funciones.
- Es decir si $r = p \circ q$, $r[i] = p[q[i]]$.

Composición de permutaciones

- La composición de dos permutaciones p y q (que llamaremos $p \circ q$) es el resultado de componer p y q como funciones.
- Es decir si $r = p \circ q$, $r[i] = p[q[i]]$.
- Por ejemplo, para $p = [5,2,4,1,3]$ y $q = [2,3,1,5,4]$, entonces $p \circ q = [2,4,5,3,1]$.

Composición de permutaciones

- La composición de dos permutaciones p y q (que llamaremos $p \circ q$) es el resultado de componer p y q como funciones.
- Es decir si $r = p \circ q$, $r[i] = p[q[i]]$.
- Por ejemplo, para $p = [5,2,4,1,3]$ y $q = [2,3,1,5,4]$, entonces $p \circ q = [2,4,5,3,1]$.
- En la notación de dos filas, si escribimos q y debajo p (alineándolas) nos queda $p \circ q$:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 1 & 5 & 4 \\ 2 & 4 & 5 & 3 & 1 \end{pmatrix}$$

Algunas definiciones

- La permutación *identidad* (que llamaremos Id) es la permutación p tal que $p[i] = i$ para todo i. Observar que $p \circ p^{-1} = p^{-1} \circ p = Id$.

Algunas definiciones

- La permutación *identidad* (que llamaremos Id) es la permutación p tal que $p[i] = i$ para todo i. Observar que $p \circ p^{-1} = p^{-1} \circ p = Id$.
- Por ejemplo, `[1,2,3,4,5]` es la permutación identidad para $n = 5$.

Algunas definiciones

- La permutación *identidad* (que llamaremos Id) es la permutación p tal que $p[i] = i$ para todo i. Observar que $p \circ p^{-1} = p^{-1} \circ p = Id$.
- Por ejemplo, `[1,2,3,4,5]` es la permutación identidad para $n = 5$.
- Un *punto fijo* de una permutación es un índice i tal que $p[i] = i$.

Algunas definiciones

- La permutación *identidad* (que llamaremos Id) es la permutación p tal que $p[i] = i$ para todo i. Observar que $p \circ p^{-1} = p^{-1} \circ p = Id$.
- Por ejemplo, `[1,2,3,4,5]` es la permutación identidad para $n = 5$.
- Un *punto fijo* de una permutación es un índice i tal que $p[i] = i$.
- Por ejemplo, la permutación `[5,2,4,1,3]` tiene un solo punto fijo, que es 2.

Algunas definiciones

- La permutación *identidad* (que llamaremos *Id*) es la permutación p tal que $p[i] = i$ para todo i. Observar que $p \circ p^{-1} = p^{-1} \circ p = Id$.
- Por ejemplo, `[1,2,3,4,5]` es la permutación identidad para $n = 5$.
- Un *punto fijo* de una permutación es un índice i tal que $p[i] = i$.
- Por ejemplo, la permutación `[5,2,4,1,3]` tiene un solo punto fijo, que es 2.
- Un *ciclo* es una lista de índices $c_1, c_2, \ldots, c_k$ tales que $p[c_1] = c_2$, $p[c_2] = c_3$, $\ldots$, $p[c_k] = c_1$.

Algunas definiciones

- La permutación *identidad* (que llamaremos Id) es la permutación p tal que $p[i] = i$ para todo i. Observar que $p \circ p^{-1} = p^{-1} \circ p = Id$.
- Por ejemplo, `[1,2,3,4,5]` es la permutación identidad para $n = 5$.
- Un *punto fijo* de una permutación es un índice i tal que $p[i] = i$.
- Por ejemplo, la permutación `[5,2,4,1,3]` tiene un solo punto fijo, que es 2.
- Un *ciclo* es una lista de índices $c_1, c_2, \ldots, c_k$ tales que $p[c_1] = c_2,\ p[c_2] = c_3,\ \ldots,\ p[c_k] = c_1$.
- Al valor de k en la definición anterior lo llamaremos el largo del ciclo. Observar que un punto fijo es un ciclo de largo 1.

Representación gráfica

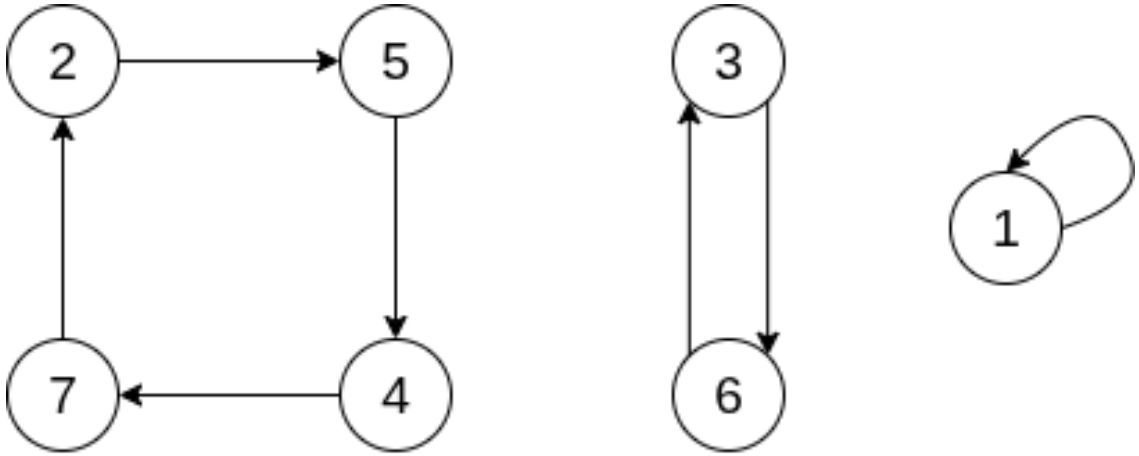
- Una permutación puede verse como un caso particular de un grafo funcional.

Representación gráfica

- Una permutación puede verse como un caso particular de un grafo funcional.
- Esto nos permite representar una permutación como un grafo dirigido, donde para cada índice i dibujamos la arista dirigida $i \to p[i]$.

Representación gráfica

- Una permutación puede verse como un caso particular de un grafo funcional.
- Esto nos permite representar una permutación como un grafo dirigido, donde para cada índice i dibujamos la arista dirigida $i \to p[i]$.
- Por ejemplo, podemos representar la permutación `[1,5,6,7,4,3,2]` gráficamente de la siguiente forma:



Partición en ciclos

Partición en ciclos

Los índices de cualquier permutación pueden particionarse de forma tal que cada elemento de la partición sea un ciclo.

Partición en ciclos

Partición en ciclos

Los índices de cualquier permutación pueden particionarse de forma tal que cada elemento de la partición sea un ciclo.

Por ejemplo, `[1,5,6,7,4,3,2]` puede particionarse en 3 ciclos, que son:

- 1
- 2, 5, 4, 7
- 3, 6

Partición en ciclos

Partición en ciclos

Los índices de cualquier permutación pueden particionarse de forma tal que cada elemento de la partición sea un ciclo.

Por ejemplo, `[1,5,6,7,4,3,2]` puede particionarse en 3 ciclos, que son:

- 1
- 2, 5, 4, 7
- 3, 6

Estos ciclos pueden computarse en $\mathcal{O}(n)$, con un algoritmo similar a DFS.

Swaps

- Un *swap* es una operación sobre una permutación que consiste en intercambiar los valores de dos posiciones.
  Es decir, un swap en (i, j) es intercambiar $p[i]$ y $p[j]$ donde $i \neq j$.

Swaps

- Un *swap* es una operación sobre una permutación que consiste en intercambiar los valores de dos posiciones.
  Es decir, un swap en (i, j) es intercambiar $p[i]$ y $p[j]$ donde $i \neq j$.
- Por ejemplo, en $p = [5,2,4,1,3]$, un swap en $(1, 3)$ produce la permutación $p' = [4,2,5,1,3]$.

Swaps

- Un *swap* es una operación sobre una permutación que consiste en intercambiar los valores de dos posiciones.
  Es decir, un swap en (i, j) es intercambiar $p[i]$ y $p[j]$ donde $i \neq j$.
- Por ejemplo, en $p = [5,2,4,1,3]$, un swap en $(1, 3)$ produce la permutación $p' = [4,2,5,1,3]$.
- Una *permutación swap* es una permutación en la que todas las posiciones son puntos fijos, salvo exactamente dos.
  Por ejemplo, $s = [3,2,1,4,5]$ es una permutación swap.

Swaps

- Un *swap* es una operación sobre una permutación que consiste en intercambiar los valores de dos posiciones.
  Es decir, un swap en (i, j) es intercambiar $p[i]$ y $p[j]$ donde $i \neq j$.
- Por ejemplo, en $p = [5,2,4,1,3]$, un swap en $(1, 3)$ produce la permutación $p' = [4,2,5,1,3]$.
- Una *permutación swap* es una permutación en la que todas las posiciones son puntos fijos, salvo exactamente dos.
  Por ejemplo, $s = [3,2,1,4,5]$ es una permutación swap.
- Viéndolo así, aplicar un swap es componer la permutación con la permutación swap correspondiente.
  En nuestro ejemplo de arriba, tenemos que $p' = p \circ s$.

Swaps

- Un *swap* es una operación sobre una permutación que consiste en intercambiar los valores de dos posiciones.
  Es decir, un swap en (i, j) es intercambiar $p[i]$ y $p[j]$ donde $i \neq j$.
- Por ejemplo, en $p = [5,2,4,1,3]$, un swap en $(1, 3)$ produce la permutación $p' = [4,2,5,1,3]$.
- Una *permutación swap* es una permutación en la que todas las posiciones son puntos fijos, salvo exactamente dos.
  Por ejemplo, $s = [3,2,1,4,5]$ es una permutación swap.
- Viéndolo así, aplicar un swap es componer la permutación con la permutación swap correspondiente.
  En nuestro ejemplo de arriba, tenemos que $p' = p \circ s$.
- Una propiedad importante de las permutaciones swap es que son iguales a su inversa, es decir, $s^{-1} = s$.

Composición de swaps

Composición de swaps

Toda permutación p se puede escribir como $p = s_1 \circ s_2 \circ \ldots \circ s_m$ donde cada s_i es una permutación swap.

Composición de swaps

Composición de swaps

Toda permutación p se puede escribir como $p = s_1 \circ s_2 \circ \ldots \circ s_m$ donde cada s_i es una permutación swap.

Usando el hecho de que $(p \circ q)^{-1} = q^{-1} \circ p^{-1}$ tenemos que

$$
\begin{aligned}
& p = s_1 \circ s_2 \circ \ldots \circ s_m \\
\iff & Id = s_1 \circ s_2 \circ \ldots \circ s_m \circ p^{-1} \\
\iff & Id = (s_1 \circ s_2 \circ \ldots \circ s_m \circ p^{-1})^{-1} \\
\iff & Id = p \circ s_m^{-1} \circ s_{m-1}^{-1} \circ \ldots \circ s_1^{-1} \\
\iff & Id = p \circ s_m \circ s_{m-1} \circ \ldots \circ s_1
\end{aligned}
$$

Composición de swaps

Composición de swaps

Toda permutación p se puede escribir como $p = s_1 \circ s_2 \circ \ldots \circ s_m$ donde cada s_i es una permutación swap.

Usando el hecho de que $(p \circ q)^{-1} = q^{-1} \circ p^{-1}$ tenemos que

$$
\begin{aligned}
& p = s_1 \circ s_2 \circ \ldots \circ s_m \\
\iff & Id = s_1 \circ s_2 \circ \ldots \circ s_m \circ p^{-1} \\
\iff & Id = (s_1 \circ s_2 \circ \ldots \circ s_m \circ p^{-1})^{-1} \\
\iff & Id = p \circ s_m^{-1} \circ s_{m-1}^{-1} \circ \ldots \circ s_1^{-1} \\
\iff & Id = p \circ s_m \circ s_{m-1} \circ \ldots \circ s_1
\end{aligned}
$$

Esto nos dice que "construir" o "destruir" una permutación haciendo swaps es lo mismo, pero haciendo las operaciones al revés.

Mínima cantidad de swaps

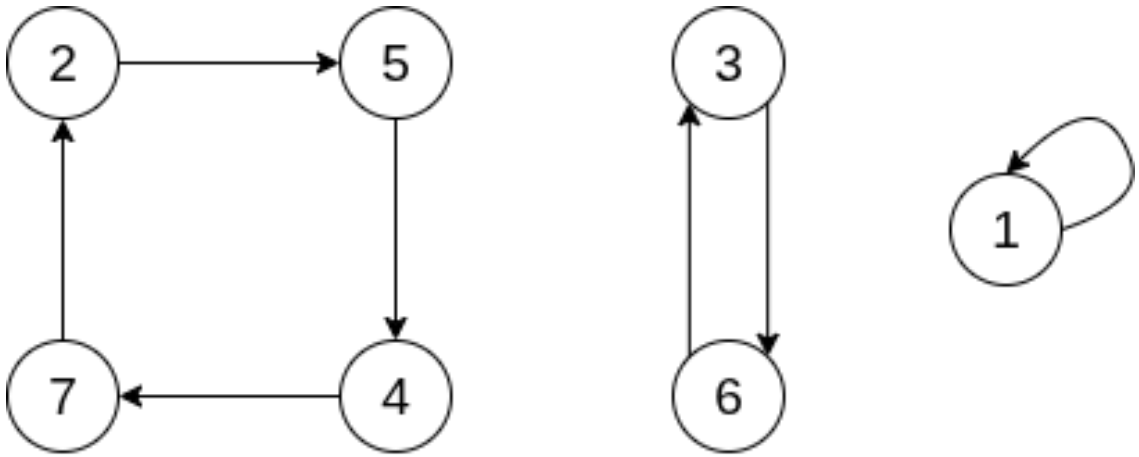
- ¿Cuál es la mínima cantidad de swaps que hay que aplicarle a una permutación para llevarla a la identidad?

Mínima cantidad de swaps

- ¿Cuál es la mínima cantidad de swaps que hay que aplicarle a una permutación para llevarla a la identidad?
- Para esto, veamos primero qué pasa con los ciclos de la permutación al hacer un swap.

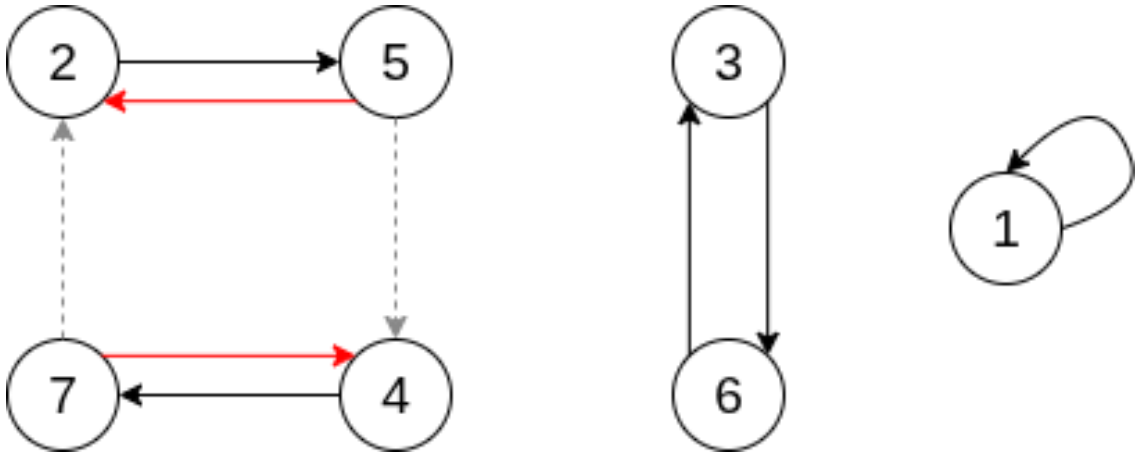
Mínima cantidad de swaps

- ¿Cuál es la mínima cantidad de swaps que hay que aplicarle a una permutación para llevarla a la identidad?
- Para esto, veamos primero qué pasa con los ciclos de la permutación al hacer un swap.
- Graficamos nuevamente `[1,5,6,7,4,3,2]`:



Mínima cantidad de swaps

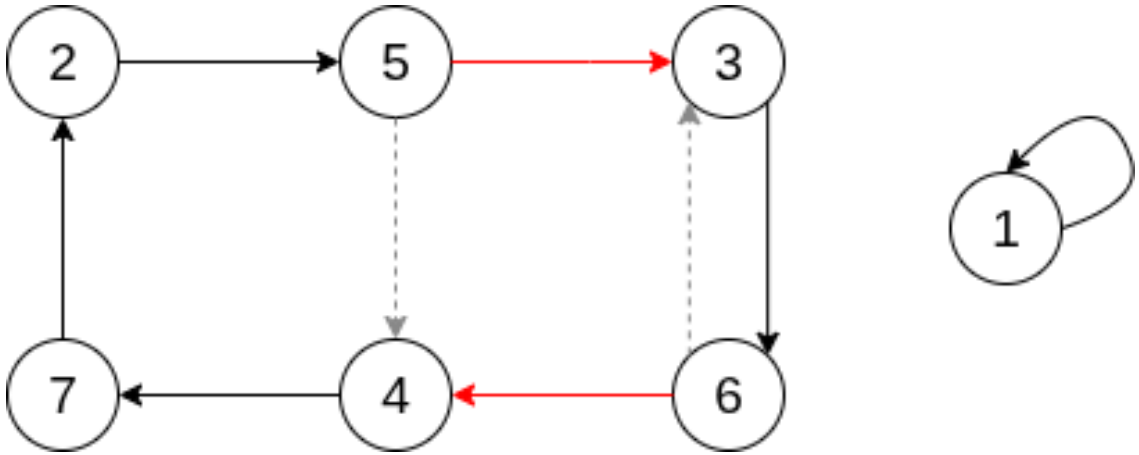
- ¿Cuál es la mínima cantidad de swaps que hay que aplicarle a una permutación para llevarla a la identidad?
- Para esto, veamos primero qué pasa con los ciclos de la permutación al hacer un swap.
- Si hacemos el swap (5,7) se divide el ciclo en dos:



La permutación resultante sería `[1,5,6,7,2,3,4]`.

Mínima cantidad de swaps

- ¿Cuál es la mínima cantidad de swaps que hay que aplicarle a una permutación para llevarla a la identidad?
- Para esto, veamos primero qué pasa con los ciclos de la permutación al hacer un swap.
- Si en cambio hacemos el swap (5,6) se unen dos ciclos en uno:



La permutación resultante sería `[1,5,6,7,3,4,2]`.

Mínima cantidad de swaps (continuación)

- Queremos llegar a la identidad, es decir, que haya n ciclos en total.

Mínima cantidad de swaps (continuación)

- Queremos llegar a la identidad, es decir, que haya n ciclos en total.
- Si tenemos un ciclo de largo k, llevar sus posiciones a la identidad puede hacerse en $k - 1$ operaciones.

Mínima cantidad de swaps (continuación)

- Queremos llegar a la identidad, es decir, que haya n ciclos en total.
- Si tenemos un ciclo de largo k, llevar sus posiciones a la identidad puede hacerse en $k - 1$ operaciones.
- Si las posiciones del ciclo son $c_1, c_2, \ldots, c_k$ una forma de hacerlo es haciendo los swaps $(c_1, c_2), (c_1, c_3), \ldots, (c_1, c_k)$.

Mínima cantidad de swaps (continuación)

- Queremos llegar a la identidad, es decir, que haya n ciclos en total.
- Si tenemos un ciclo de largo k, llevar sus posiciones a la identidad puede hacerse en $k - 1$ operaciones.
- Si las posiciones del ciclo son $c_1, c_2, \ldots, c_k$ una forma de hacerlo es haciendo los swaps $(c_1, c_2), (c_1, c_3), \ldots, (c_1, c_k)$.
- Por lo tanto, si c es la cantidad de ciclos de la permutación, podemos hacerlo en $n - c$ swaps (por cada ciclo nos ahorramos un swap).

Mínima cantidad de swaps (continuación)

- Queremos llegar a la identidad, es decir, que haya n ciclos en total.
- Si tenemos un ciclo de largo k, llevar sus posiciones a la identidad puede hacerse en $k - 1$ operaciones.
- Si las posiciones del ciclo son $c_1, c_2, \ldots, c_k$ una forma de hacerlo es haciendo los swaps $(c_1, c_2), (c_1, c_3), \ldots, (c_1, c_k)$.
- Por lo tanto, si c es la cantidad de ciclos de la permutación, podemos hacerlo en $n - c$ swaps (por cada ciclo nos ahorramos un swap).
- Esa es la mínima cantidad ya que, como vimos antes, podemos crear máximo un ciclo con cada swap.

Mínima cantidad de swaps (continuación)

- Queremos llegar a la identidad, es decir, que haya n ciclos en total.
- Si tenemos un ciclo de largo k, llevar sus posiciones a la identidad puede hacerse en $k - 1$ operaciones.
- Si las posiciones del ciclo son $c_1, c_2, \ldots, c_k$ una forma de hacerlo es haciendo los swaps $(c_1, c_2), (c_1, c_3), \ldots, (c_1, c_k)$.
- Por lo tanto, si c es la cantidad de ciclos de la permutación, podemos hacerlo en $n - c$ swaps (por cada ciclo nos ahorramos un swap).
- Esa es la mínima cantidad ya que, como vimos antes, podemos crear máximo un ciclo con cada swap.
- La complejidad es $\mathcal{O}(n)$ (contar los ciclos).

Potencias y orden de una permutación

- Denotamos $p^m = \overbrace{p \circ p \circ \ldots \circ p}^{m \text{ veces}}$.

Potencias y orden de una permutación

- Denotamos $p^m = \overbrace{p \circ p \circ \ldots \circ p}^{m \text{ veces}}$.
- Esto puede calcularse en $\mathcal{O}(n \log m)$ con binary lifting, o en $\mathcal{O}(n)$ si calculamos para cada elemento cuánto avanza en su ciclo.

Potencias y orden de una permutación

- Denotamos $p^m = \overbrace{p \circ p \circ \ldots \circ p}^{m \text{ veces}}$.
- Esto puede calcularse en $\mathcal{O}(n \log m)$ con binary lifting, o en $\mathcal{O}(n)$ si calculamos para cada elemento cuánto avanza en su ciclo.
- ¿Cuál es el mínimo m tal que $p^m = Id$?

Potencias y orden de una permutación

- Denotamos $p^m = \overbrace{p \circ p \circ \ldots \circ p}^{m \text{ veces}}$.
- Esto puede calcularse en $\mathcal{O}(n \log m)$ con binary lifting, o en $\mathcal{O}(n)$ si calculamos para cada elemento cuánto avanza en su ciclo.
- ¿Cuál es el mínimo m tal que $p^m = Id$?
- Observar que en un ciclo de tamaño k, al hacer p^k todas las posiciones del ciclo se convierten en puntos fijos.
  Esto ocurre si y solo si el exponente es múltiplo de k.

Potencias y orden de una permutación

- Denotamos $p^m = \overbrace{p \circ p \circ \ldots \circ p}^{m \text{ veces}}$.
- Esto puede calcularse en $\mathcal{O}(n \log m)$ con binary lifting, o en $\mathcal{O}(n)$ si calculamos para cada elemento cuánto avanza en su ciclo.
- ¿Cuál es el mínimo m tal que $p^m = Id$?
- Observar que en un ciclo de tamaño k, al hacer p^k todas las posiciones del ciclo se convierten en puntos fijos.
  Esto ocurre si y solo si el exponente es múltiplo de k.
- Por lo tanto, el mínimo m tal que $p^m = Id$ es $\text{lcm}(\ell_1, \ell_2, \ldots, \ell_c)$ donde $\ell_1, \ell_2, \ldots, \ell_c$ son los largos de los c ciclos de la permutación.

Raíz cuadrada de una permutación

- ¿Y si queremos calcular $p^{\frac{1}{2}}$? Es decir, alguna r tal que $p = r \circ r$.

Raíz cuadrada de una permutación

- ¿Y si queremos calcular $p^{\frac{1}{2}}$? Es decir, alguna r tal que $p = r \circ r$.
- Veamos qué pasa con los ciclos de una permutación r al hacer $r \circ r$.

Raíz cuadrada de una permutación

- ¿Y si queremos calcular $p^{\frac{1}{2}}$? Es decir, alguna r tal que $p = r \circ r$.
- Veamos qué pasa con los ciclos de una permutación r al hacer $r \circ r$.
- Un ciclo par $c_1, c_2, \ldots, c_{2k}$ se convierte en dos ciclos, $c_1, c_3, \ldots, c_{2k-1}$ y $c_2, c_4, \ldots, c_{2k}$ cada uno largo k.

Raíz cuadrada de una permutación

- ¿Y si queremos calcular $p^{\frac{1}{2}}$? Es decir, alguna r tal que $p = r \circ r$.
- Veamos qué pasa con los ciclos de una permutación r al hacer $r \circ r$.
- Un ciclo par $c_1, c_2, \ldots, c_{2k}$ se convierte en dos ciclos, $c_1, c_3, \ldots, c_{2k-1}$ y $c_2, c_4, \ldots, c_{2k}$ cada uno largo k.
- Un ciclo impar $c_1, c_2, \ldots, c_{2k+1}$ se convierte en un ciclo, $c_1, c_3, \ldots, c_{2k-1}, c_{2k+1}, c_2, c_4, \ldots, c_{2k}$ de largo $2k + 1$.

Raíz cuadrada de una permutación

- ¿Y si queremos calcular $p^{\frac{1}{2}}$? Es decir, alguna r tal que $p = r \circ r$.
- Veamos qué pasa con los ciclos de una permutación r al hacer $r \circ r$.
- Un ciclo par $c_1, c_2, \ldots, c_{2k}$ se convierte en dos ciclos, $c_1, c_3, \ldots, c_{2k-1}$ y $c_2, c_4, \ldots, c_{2k}$ cada uno largo k.
- Un ciclo impar $c_1, c_2, \ldots, c_{2k+1}$ se convierte en un ciclo, $c_1, c_3, \ldots, c_{2k-1}, c_{2k+1}, c_2, c_4, \ldots, c_{2k}$ de largo $2k + 1$.
- Por lo tanto, para que exista la raíz debe pasar que para cada número par haya una cantidad par de ciclos de ese largo.

Raíz cuadrada de una permutación

- ¿Y si queremos calcular $p^{\frac{1}{2}}$? Es decir, alguna r tal que $p = r \circ r$.
- Veamos qué pasa con los ciclos de una permutación r al hacer $r \circ r$.
- Un ciclo par $c_1, c_2, \ldots, c_{2k}$ se convierte en dos ciclos, $c_1, c_3, \ldots, c_{2k-1}$ y $c_2, c_4, \ldots, c_{2k}$ cada uno largo k.
- Un ciclo impar $c_1, c_2, \ldots, c_{2k+1}$ se convierte en un ciclo, $c_1, c_3, \ldots, c_{2k-1}, c_{2k+1}, c_2, c_4, \ldots, c_{2k}$ de largo $2k + 1$.
- Por lo tanto, para que exista la raíz debe pasar que para cada número par haya una cantidad par de ciclos de ese largo.
- La construcción consiste en reordenar los índices de los ciclos de largo impar y juntar de a pares los ciclos de largo par (y con el mismo largo). La complejidad es $\mathcal{O}(n)$.

Inversiones

- Una *inversión* en una permutación p es un par (i, j) tal que $i < j$ y $p[i] > p[j]$.

Inversiones

- Una *inversión* en una permutación p es un par (i, j) tal que $i < j$ y $p[i] > p[j]$.
- Por ejemplo, en $p = [5,2,4,1,3]$, $(3, 5)$ es una inversión ya que $3 < 5$ y $p[3] = 4 > 3 = p[5]$.

Inversiones

- Una *inversión* en una permutación p es un par (i, j) tal que $i < j$ y $p[i] > p[j]$.
- Por ejemplo, en $p =$ `[5,2,4,1,3]`, $(3, 5)$ es una inversión ya que $3 < 5$ y $p[3] = 4 > 3 = p[5]$.
- No es la única, $(1, 2)$ y $(3, 4)$ también lo son.
  En total hay 7 inversiones.

Inversiones

- Una *inversión* en una permutación p es un par (i, j) tal que $i < j$ y $p[i] > p[j]$.
- Por ejemplo, en $p = [5,2,4,1,3]$, $(3, 5)$ es una inversión ya que $3 < 5$ y $p[3] = 4 > 3 = p[5]$.
- No es la única, $(1, 2)$ y $(3, 4)$ también lo son.
  En total hay 7 inversiones.
- En una permutación de tamaño n la mínima cantidad de inversiones posibles es 0 (en la identidad) y la máxima cantidad posible es $\frac{n(n-1)}{2}$ (en la reversa de la identidad). Todas las cantidades entre medio son conseguibles con alguna permutación de tamaño n.

Contar inversiones

- ¿Cómo podemos hacer para contar eficientemente la cantidad de inversiones en una permutación?

Contar inversiones

- ¿Cómo podemos hacer para contar eficientemente la cantidad de inversiones en una permutación?
- Tenemos nuestra permutación p con sus elementos $p[1], p[2], \ldots, p[n]$ en orden.

Contar inversiones

- ¿Cómo podemos hacer para contar eficientemente la cantidad de inversiones en una permutación?
- Tenemos nuestra permutación p con sus elementos $p[1], p[2], \ldots, p[n]$ en orden.
- Los procesamos de izquierda a derecha, y los vamos metiendo a un indexed set de C++. Cada vez que procesamos un elemento, contamos cuántos elementos mayores a él ya agregamos al set, sumamos eso a la respuesta y recién después lo agregamos.

Contar inversiones

- ¿Cómo podemos hacer para contar eficientemente la cantidad de inversiones en una permutación?
- Tenemos nuestra permutación p con sus elementos $p[1], p[2], \ldots, p[n]$ en orden.
- Los procesamos de izquierda a derecha, y los vamos metiendo a un indexed set de C++. Cada vez que procesamos un elemento, contamos cuántos elementos mayores a él ya agregamos al set, sumamos eso a la respuesta y recién después lo agregamos.
- Esto mismo puede hacerse con un segment tree de suma, donde todas las posiciones inicialmente valen 0 y y cuando vemos un elemento le sumamos 1.

Contar inversiones

- ¿Cómo podemos hacer para contar eficientemente la cantidad de inversiones en una permutación?
- Tenemos nuestra permutación p con sus elementos $p[1], p[2], \ldots, p[n]$ en orden.
- Los procesamos de izquierda a derecha, y los vamos metiendo a un indexed set de C++. Cada vez que procesamos un elemento, contamos cuántos elementos mayores a él ya agregamos al set, sumamos eso a la respuesta y recién después lo agregamos.
- Esto mismo puede hacerse con un segment tree de suma, donde todas las posiciones inicialmente valen 0 y y cuando vemos un elemento le sumamos 1.
- En ambos casos la complejidad queda $\mathcal{O}(n \log n)$.

Mínima cantidad de swaps adyacentes

- ¿Cuál es la mínima cantidad de swaps **entre elementos adyacentes** que hay que aplicarle a una permutación para llevarla a la identidad?

Mínima cantidad de swaps adyacentes

- ¿Cuál es la mínima cantidad de swaps **entre elementos adyacentes** que hay que aplicarle a una permutación para llevarla a la identidad?
- Observemos que al aplicar un swap entre elementos adyacentes, la cantidad de inversiones aumenta o disminuye en exactamente uno.

Mínima cantidad de swaps adyacentes

- ¿Cuál es la mínima cantidad de swaps **entre elementos adyacentes** que hay que aplicarle a una permutación para llevarla a la identidad?
- Observemos que al aplicar un swap entre elementos adyacentes, la cantidad de inversiones aumenta o disminuye en exactamente uno.
- Por lo tanto debemos hacer $\geq \text{inv}(p)$ (cantidad de inversiones de p) operaciones.

Mínima cantidad de swaps adyacentes

- ¿Cuál es la mínima cantidad de swaps **entre elementos adyacentes** que hay que aplicarle a una permutación para llevarla a la identidad?
- Observemos que al aplicar un swap entre elementos adyacentes, la cantidad de inversiones aumenta o disminuye en exactamente uno.
- Por lo tanto debemos hacer $\geq \text{inv}(p)$ (cantidad de inversiones de p) operaciones.
- Además, si todavía no llegamos a la identidad, debe existir un índice i tal que $p[i] > p[i+1]$. Luego podemos aplicar ahí el swap y disminuimos en 1 la cantidad de inversiones.

Mínima cantidad de swaps adyacentes

- ¿Cuál es la mínima cantidad de swaps **entre elementos adyacentes** que hay que aplicarle a una permutación para llevarla a la identidad?
- Observemos que al aplicar un swap entre elementos adyacentes, la cantidad de inversiones aumenta o disminuye en exactamente uno.
- Por lo tanto debemos hacer $\geq \text{inv}(p)$ (cantidad de inversiones de p) operaciones.
- Además, si todavía no llegamos a la identidad, debe existir un índice i tal que $p[i] > p[i+1]$. Luego podemos aplicar ahí el swap y disminuimos en 1 la cantidad de inversiones.
- Entonces, haciendo eso tenemos un algoritmo que lo resuelve en $\text{inv}(p)$ operaciones. Luego esta es la mínima cantidad de operaciones.

Mínima cantidad de swaps adyacentes

- ¿Cuál es la mínima cantidad de swaps **entre elementos adyacentes** que hay que aplicarle a una permutación para llevarla a la identidad?
- Observemos que al aplicar un swap entre elementos adyacentes, la cantidad de inversiones aumenta o disminuye en exactamente uno.
- Por lo tanto debemos hacer $\geq \text{inv}(p)$ (cantidad de inversiones de p) operaciones.
- Además, si todavía no llegamos a la identidad, debe existir un índice i tal que $p[i] > p[i+1]$. Luego podemos aplicar ahí el swap y disminuimos en 1 la cantidad de inversiones.
- Entonces, haciendo eso tenemos un algoritmo que lo resuelve en $\text{inv}(p)$ operaciones. Luego esta es la mínima cantidad de operaciones.
- La complejidad es $\mathcal{O}(n \log n)$ (contar las inversiones).

Problema de ejemplo: Kill Anton

Codeforces 1526D: Kill Anton (Problema C contest 2 TC Arg 2025)

Anton's DNA can be represented as a string a which only contains the characters "ANTON" (there are only 4 distinct characters).
Errorgorn can change Anton's DNA into string b which must be a permutation of a. However, Anton's body can defend against this attack. In 1 second, his body can swap 2 adjacent characters of his DNA to transform it back to a. Anton's body is smart and will use the minimum number of moves.
To maximize the chance of Anton dying, Errorgorn wants to change Anton's DNA the string that maximizes the time for Anton's body to revert his DNA. But since Errorgorn is busy, he needs your help to find the best string b. Can you help him? Constraints: $1 \leq |a| \leq 10^5$.

`https://codeforces.com/contest/1526/problem/D`

Problema de ejemplo: Kill Anton

Codeforces 1526D: Kill Anton (Problema C contest 2 TC Arg 2025)

Anton's DNA can be represented as a string a which only contains the characters "ANTON" (there are only 4 distinct characters).
Errorgorn can change Anton's DNA into string b which must be a permutation of a. However, Anton's body can defend against this attack. In 1 second, his body can swap 2 adjacent characters of his DNA to transform it back to a. Anton's body is smart and will use the minimum number of moves.
To maximize the chance of Anton dying, Errorgorn wants to change Anton's DNA the string that maximizes the time for Anton's body to revert his DNA. But since Errorgorn is busy, he needs your help to find the best string b. Can you help him? Constraints: $1 \leq |a| \leq 10^5$.

`https://codeforces.com/contest/1526/problem/D`

Solución en el pizarrón.

Paridad de una permutación

- La paridad de una permutación es la paridad de su cantidad de inversiones.

Paridad de una permutación

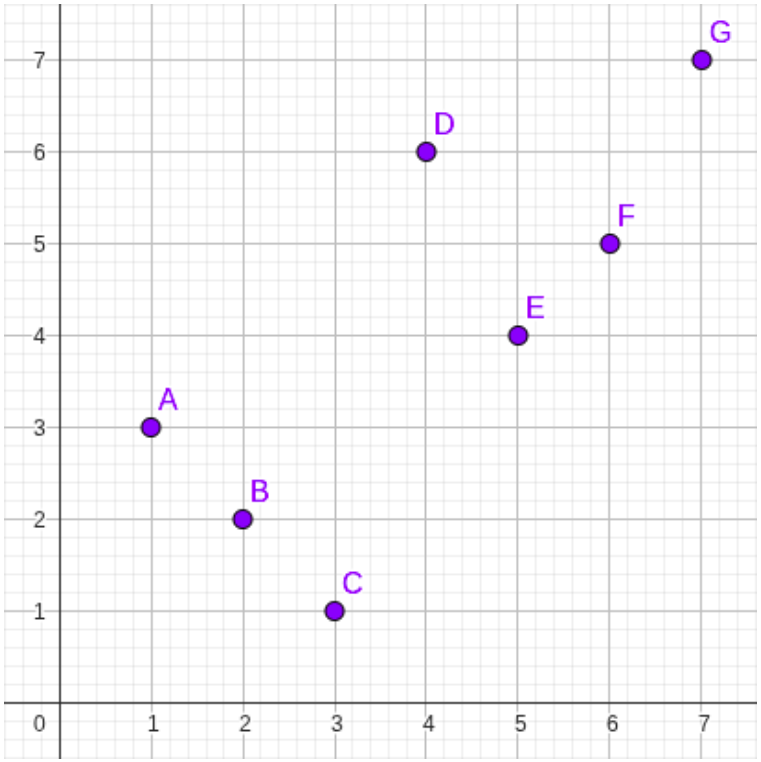
- La paridad de una permutación es la paridad de su cantidad de inversiones.
- Cada vez que aplicamos un swap, la paridad de la permutación cambia.

Paridad de una permutación

- La paridad de una permutación es la paridad de su cantidad de inversiones.
- Cada vez que aplicamos un swap, la paridad de la permutación cambia.
- Para entender porqué pasa esto, grafiquemos los puntos $(i, p[i])$ en el plano, veamos la permutación `[3,2,1,6,4,5,7]` y el swap $(2, 6)$:

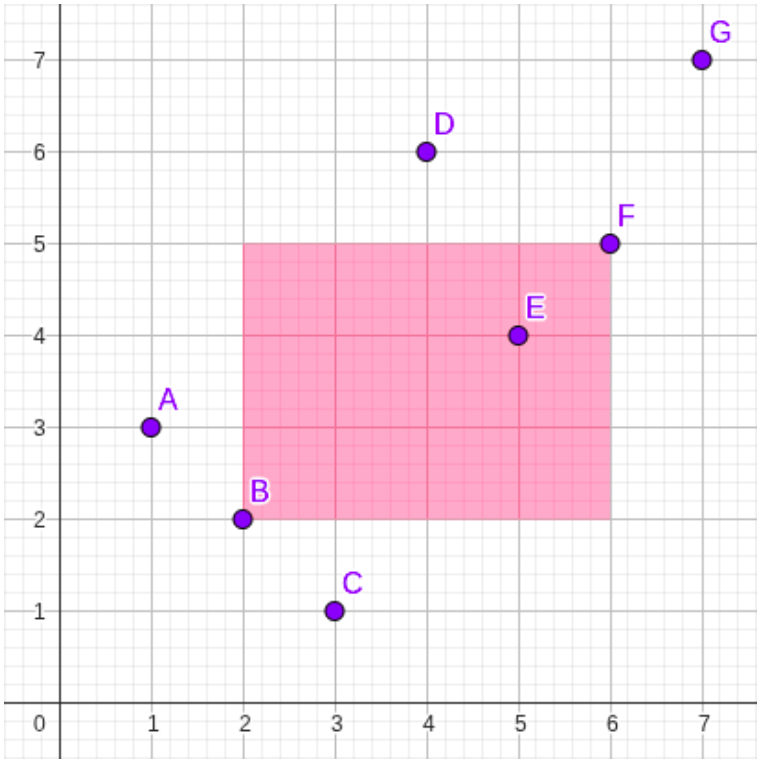
Paridad de una permutación

- La paridad de una permutación es la paridad de su cantidad de inversiones.
- Cada vez que aplicamos un swap, la paridad de la permutación cambia.



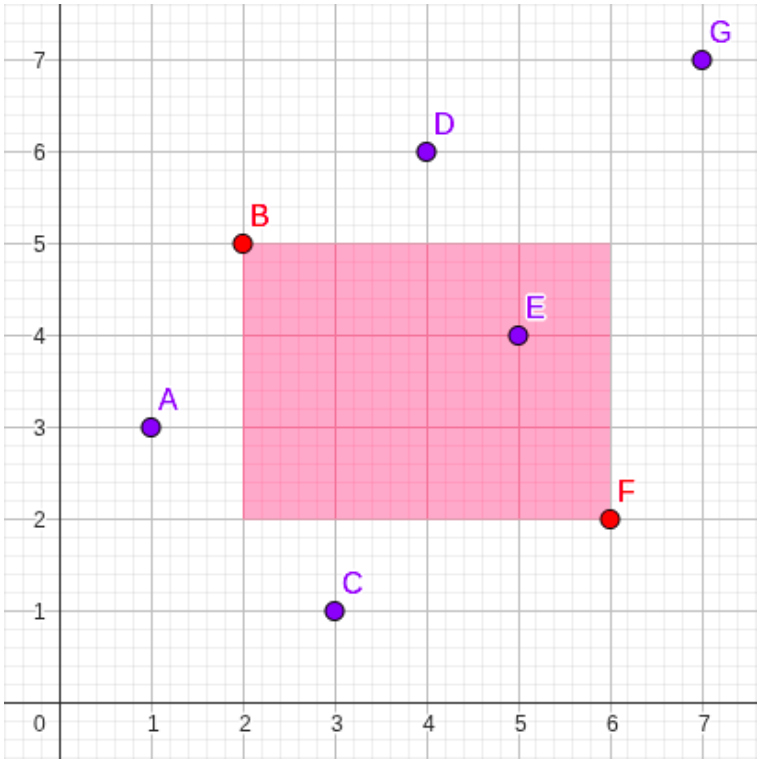
Paridad de una permutación

- La paridad de una permutación es la paridad de su cantidad de inversiones.
- Cada vez que aplicamos un swap, la paridad de la permutación cambia.



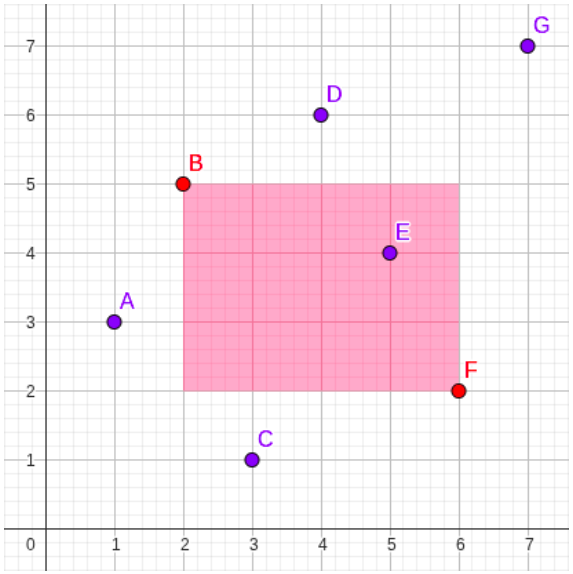
Paridad de una permutación

- La paridad de una permutación es la paridad de su cantidad de inversiones.
- Cada vez que aplicamos un swap, la paridad de la permutación cambia.



Paridad de una permutación

- La paridad de una permutación es la paridad de su cantidad de inversiones.
- Cada vez que aplicamos un swap, la paridad de la permutación cambia.



- Conclusión: Viendo geométricamente todos los casos, la paridad siempre cambia al aplicar un swap.

Paridad y ciclos

- Usando lo anterior y que todas las permutaciones son composición de swaps, podemos ver que al componer permutaciones sus paridades se suman.

Paridad y ciclos

- Usando lo anterior y que todas las permutaciones son composición de swaps, podemos ver que al componer permutaciones sus paridades se suman.
- Si tenemos una permutación compuesta por un solo ciclo de largo k, ya vimos que podemos construirla con $k-1$ swaps.

Paridad y ciclos

- Usando lo anterior y que todas las permutaciones son composición de swaps, podemos ver que al componer permutaciones sus paridades se suman.
- Si tenemos una permutación compuesta por un solo ciclo de largo k, ya vimos que podemos construirla con $k-1$ swaps.
- Por lo tanto, un ciclo de largo par es una permutación impar y un ciclo de largo impar es una permutación par.

Paridad y ciclos

- Usando lo anterior y que todas las permutaciones son composición de swaps, podemos ver que al componer permutaciones sus paridades se suman.
- Si tenemos una permutación compuesta por un solo ciclo de largo k, ya vimos que podemos construirla con $k-1$ swaps.
- Por lo tanto, un ciclo de largo par es una permutación impar y un ciclo de largo impar es una permutación par.
- Recordemos que podemos construir cualquier permutación con $n-c$ swaps, donde c es la cantidad de ciclos. Luego la paridad de una permutación coincide con la paridad de $n-c$.

Paridad y ciclos

- Usando lo anterior y que todas las permutaciones son composición de swaps, podemos ver que al componer permutaciones sus paridades se suman.
- Si tenemos una permutación compuesta por un solo ciclo de largo k, ya vimos que podemos construirla con $k-1$ swaps.
- Por lo tanto, un ciclo de largo par es una permutación impar y un ciclo de largo impar es una permutación par.
- Recordemos que podemos construir cualquier permutación con $n-c$ swaps, donde c es la cantidad de ciclos. Luego la paridad de una permutación coincide con la paridad de $n-c$.
- Esto nos permite calcularla en $\mathcal{O}(n)$ en lugar de en $\mathcal{O}(n \log n)$ contando las inversiones.

Problema de ejemplo: Swap Dilemma

Codeforces 1983D: Swap Dilemma

Given two arrays of distinct positive integers a and b of length n, we would like to make both the arrays the same. Two arrays x and y of length k are said to be the same when for all $1 \leq i \leq k$, $x_i = y_i$.
Now in one move, you can choose some index l and r in a ($l \leq r$) and swap a_l and a_r, then choose some p and q ($p \leq q$) in b such that $r - l = q - p$ and swap b_p and b_q.
Is it possible to make both arrays the same?
Constraints: $1 \leq n \leq 10^5, 1 \leq a_i, b_i \leq 2 \cdot 10^5$.

`https://codeforces.com/contest/1983/problem/D`

Problema de ejemplo: Swap Dilemma

Codeforces 1983D: Swap Dilemma

Given two arrays of distinct positive integers a and b of length n, we would like to make both the arrays the same. Two arrays x and y of length k are said to be the same when for all $1 \leq i \leq k$, $x_i = y_i$.
Now in one move, you can choose some index l and r in a ($l \leq r$) and swap a_l and a_r, then choose some p and q ($p \leq q$) in b such that $r - l = q - p$ and swap b_p and b_q.
Is it possible to make both arrays the same?
Constraints: $1 \leq n \leq 10^5, 1 \leq a_i, b_i \leq 2 \cdot 10^5$.

`https://codeforces.com/contest/1983/problem/D`

Solución en el pizarrón.

Permutaciones sin puntos fijos (Derangements)

- ¿Cuántas permutaciones sin puntos fijos de largo n existen? Llamemos a este número D_n.

Permutaciones sin puntos fijos (Derangements)

- ¿Cuántas permutaciones sin puntos fijos de largo n existen? Llamemos a este número D_n.
- Para $n = 0, 1, 2, \ldots$ da $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$.

Permutaciones sin puntos fijos (Derangements)

- ¿Cuántas permutaciones sin puntos fijos de largo n existen? Llamemos a este número D_n.
- Para $n = 0, 1, 2, \ldots$ da $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$.
- Podemos calcularlo recursivamente. Si p no tiene puntos fijos, $p[n] = k \neq n$, y hay $n - 1$ posibilidades para el valor de k.

Permutaciones sin puntos fijos (Derangements)

- ¿Cuántas permutaciones sin puntos fijos de largo n existen? Llamemos a este número D_n.
- Para $n = 0, 1, 2, \ldots$ da $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$.
- Podemos calcularlo recursivamente. Si p no tiene puntos fijos, $p[n] = k \neq n$, y hay $n - 1$ posibilidades para el valor de k.
- Hay dos casos:
  - Si $p[k] = n$, entonces los otros $n - 2$ valores forman una permutación. Hay D_{n-2} formas en este caso.
  - Si $p[k] \neq n$, entonces podemos renombrar n a k y los primeros $n - 1$ valores forman una permutación. Hay D_{n-1} formas en este caso.

Permutaciones sin puntos fijos (Derangements)

- ¿Cuántas permutaciones sin puntos fijos de largo n existen? Llamemos a este número D_n.
- Para $n = 0, 1, 2, \ldots$ da $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$.
- Podemos calcularlo recursivamente. Si p no tiene puntos fijos, $p[n] = k \neq n$, y hay $n - 1$ posibilidades para el valor de k.
- Hay dos casos:
  - Si $p[k] = n$, entonces los otros $n - 2$ valores forman una permutación. Hay D_{n-2} formas en este caso.
  - Si $p[k] \neq n$, entonces podemos renombrar n a k y los primeros $n - 1$ valores forman una permutación. Hay D_{n-1} formas en este caso.
- Luego la recurrencia que obtenemos es $D_n = (n - 1)(D_{n-1} + D_{n-2})$.

Permutaciones sin puntos fijos (Derangements)

- ¿Cuántas permutaciones sin puntos fijos de largo n existen? Llamemos a este número D_n.
- Para $n = 0, 1, 2, \ldots$ da $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$.
- Podemos calcularlo recursivamente. Si p no tiene puntos fijos, $p[n] = k \neq n$, y hay $n - 1$ posibilidades para el valor de k.
- Hay dos casos:
  - Si $p[k] = n$, entonces los otros $n - 2$ valores forman una permutación. Hay D_{n-2} formas en este caso.
  - Si $p[k] \neq n$, entonces podemos renombrar n a k y los primeros $n - 1$ valores forman una permutación. Hay D_{n-1} formas en este caso.
- Luego la recurrencia que obtenemos es $D_n = (n - 1)(D_{n-1} + D_{n-2})$.
- La cantidad de perm con **exactamente** k puntos fijos es $\binom{n}{k} D_{n-k}$.

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4]`

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4,2]`

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4,2]`
  - `[6]`

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4,2]`
  - `[6]`
  - `[8]`

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4,2]`
  - `[6,3]`
  - `[8]`

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4,2,1]`
  - `[6,3]`
  - `[8]`

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4,2,1]`
  - `[6,3]`
  - `[8,5]`

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4,2,1]`
  - `[6,3]`
  - `[8,5]`
  - `[7]`

Descomposición en subsecuencias decrecientes

- Podemos descomponer una permutación en la mínima cantidad de subsecuencias decrecientes con un algoritmo greedy.
- Procesamos los elementos de a uno y en cada paso agregamos el elemento actual a la primera secuencia que podemos.
- Por ejemplo veamos la permutación `[4,2,6,8,3,1,5,7]`:
  - `[4,2,1]`
  - `[6,3]`
  - `[8,5]`
  - `[7]`
- La complejidad de esto es $\mathcal{O}(n \log n)$ ya que en cada paso debemos hacer una búsqueda binaria para saber cuál es la primera secuencia en la que encajamos.

Máxima subsecuencia creciente (LIS)

Máxima subsecuencia creciente = Mínima descomposición en subsecuencias decrecientes

En toda permutación, el largo de la máxima subsecuencia creciente es igual a la cantidad de secuencias en la mínima descomposición en subsecuencias decrecientes.

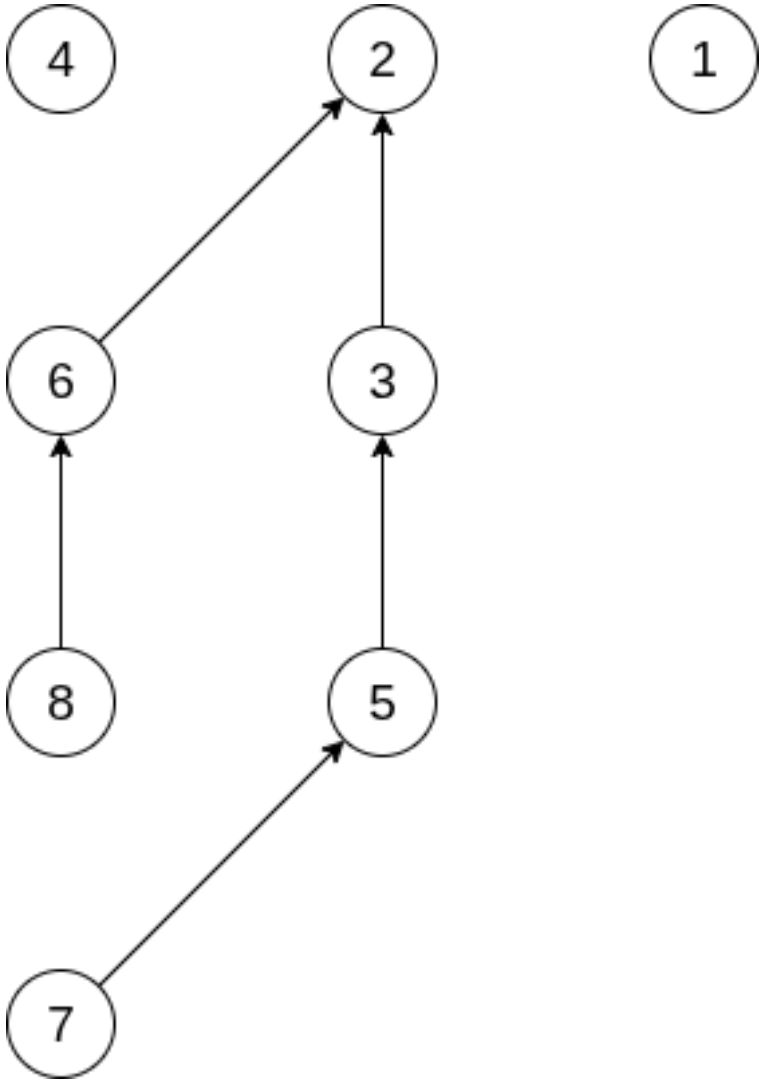
Máxima subsecuencia creciente (LIS)

Máxima subsecuencia creciente = Mínima descomposición en subsecuencias decrecientes

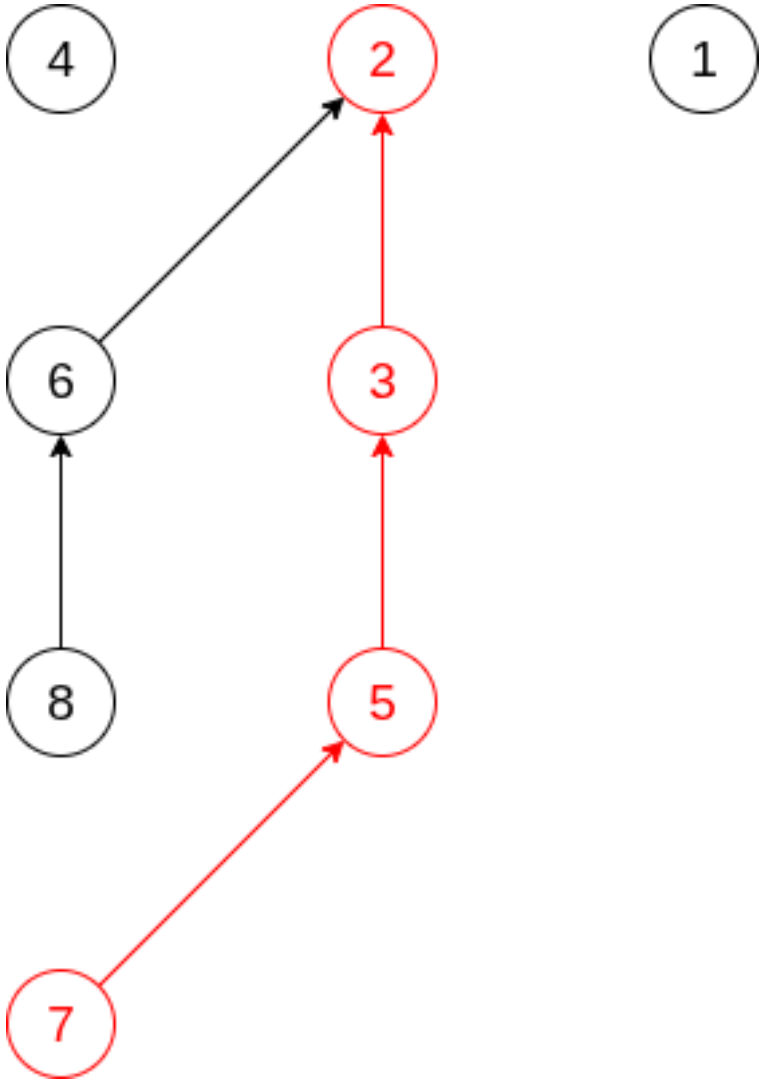
En toda permutación, el largo de la máxima subsecuencia creciente es igual a la cantidad de secuencias en la mínima descomposición en subsecuencias decrecientes.

Para obtenerla a partir de la descomposición anterior, calculemos para cada elemento quién era el último en la secuencia anterior:

Máxima subsecuencia creciente (LIS)



Máxima subsecuencia creciente (LIS)



Máxima subsecuencia creciente (LIS)

Máxima subsecuencia creciente = Mínima descomposición en subsecuencias decrecientes

En toda permutación, el largo de la máxima subsecuencia creciente es igual a la cantidad de secuencias en la mínima descomposición en subsecuencias decrecientes.

Para obtenerla a partir de la descomposición anterior, calculemos para cada elemento quién era el último en la secuencia anterior:

Si en la última secuencia hay más de un elemento, podemos elegir cualquiera como el último de la LIS.

Máxima subsecuencia creciente (LIS)

Máxima subsecuencia creciente = Mínima descomposición en subsecuencias decrecientes

En toda permutación, el largo de la máxima subsecuencia creciente es igual a la cantidad de secuencias en la mínima descomposición en subsecuencias decrecientes.

Para obtenerla a partir de la descomposición anterior, calculemos para cada elemento quién era el último en la secuencia anterior:

Si en la última secuencia hay más de un elemento, podemos elegir cualquiera como el último de la LIS.

Si quisieramos una más larga, necesariamente tenemos que elegir dos en la misma secuencia de la descomposición. Esto junto con la construcción demuestran el teorema de arriba.

Teorema de Erdős-Szekeres

Teorema de Erdős-Szekeres

En toda permutación de largo $xy + 1$ existe una subsecuencia creciente de largo $x + 1$ o una decreciente de largo $y + 1$.

Teorema de Erdős-Szekeres

Teorema de Erdős-Szekeres

En toda permutación de largo $xy + 1$ existe una subsecuencia creciente de largo $x + 1$ o una decreciente de largo $y + 1$.

- Demostración: Supongamos que no pasa. Entonces tenemos máximo x secuencias en la descomposición, y cada una de largo máximo y. Pero entonces tenemos como máximo xy elementos. Absurdo.

Teorema de Erdős-Szekeres

Teorema de Erdős-Szekeres

En toda permutación de largo $xy + 1$ existe una subsecuencia creciente de largo $x + 1$ o una decreciente de largo $y + 1$.

- Demostración: Supongamos que no pasa. Entonces tenemos máximo x secuencias en la descomposición, y cada una de largo máximo y. Pero entonces tenemos como máximo xy elementos. Absurdo.
- Si hacemos $x = y$, esto nos dice que en una permutación de tamaño $x^2 + 1$ hay una subsecuencia creciente o decreciente de largo $x + 1$.

Teorema de Erdős-Szekeres

Teorema de Erdős-Szekeres

En toda permutación de largo $xy + 1$ existe una subsecuencia creciente de largo $x + 1$ o una decreciente de largo $y + 1$.

- Demostración: Supongamos que no pasa. Entonces tenemos máximo x secuencias en la descomposición, y cada una de largo máximo y. Pero entonces tenemos como máximo xy elementos. Absurdo.
- Si hacemos $x = y$, esto nos dice que en una permutación de tamaño $x^2 + 1$ hay una subsecuencia creciente o decreciente de largo $x + 1$.
- En términos de n, siempre hay una subsecuencia creciente o decreciente de largo $\lceil \sqrt{n} \rceil$.

Teorema de Erdős-Szekeres

Teorema de Erdős-Szekeres

En toda permutación de largo $xy + 1$ existe una subsecuencia creciente de largo $x + 1$ o una decreciente de largo $y + 1$.

- Demostración: Supongamos que no pasa. Entonces tenemos máximo x secuencias en la descomposición, y cada una de largo máximo y. Pero entonces tenemos como máximo xy elementos. Absurdo.
- Si hacemos $x = y$, esto nos dice que en una permutación de tamaño $x^2 + 1$ hay una subsecuencia creciente o decreciente de largo $x + 1$.
- En términos de n, siempre hay una subsecuencia creciente o decreciente de largo $\lceil \sqrt{n} \rceil$.
- Si permitimos repetidos, una de las direcciones debe permitir iguales y la otra no.

Permutaciones aleatorias

- Hay $n!$ permutaciones. Podemos generar una al azar usando `random_shuffle` en C++.

Permutaciones aleatorias

- Hay $n!$ permutaciones. Podemos generar una al azar usando `random_shuffle` en C++.
- En una permutación al azar, agarremos una subsecuencia creciente de forma greedy: Elegimos el primer elemento y en cada paso buscamos el más grande a su derecha.

Permutaciones aleatorias

- Hay $n!$ permutaciones. Podemos generar una al azar usando `random_shuffle` en C++.
- En una permutación al azar, agarremos una subsecuencia creciente de forma greedy: Elegimos el primer elemento y en cada paso buscamos el más grande a su derecha.
- El valor esperado del tamaño de esta secuencia es la suma de las probabilidades de que cada elemento esté en la secuencia (por linealidad de la esperanza).

Permutaciones aleatorias

- Hay $n!$ permutaciones. Podemos generar una al azar usando `random_shuffle` en C++.
- En una permutación al azar, agarremos una subsecuencia creciente de forma greedy: Elegimos el primer elemento y en cada paso buscamos el más grande a su derecha.
- El valor esperado del tamaño de esta secuencia es la suma de las probabilidades de que cada elemento esté en la secuencia (por linealidad de la esperanza).
- $P(p[i] > max(p[1], p[2], \ldots, p[i-1])) = \frac{1}{i}$.
  Entonces el valor esperado es $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \mathcal{O}(\log n)$.

Permutaciones aleatorias

- Hay $n!$ permutaciones. Podemos generar una al azar usando `random_shuffle` en C++.
- En una permutación al azar, agarremos una subsecuencia creciente de forma greedy: Elegimos el primer elemento y en cada paso buscamos el más grande a su derecha.
- El valor esperado del tamaño de esta secuencia es la suma de las probabilidades de que cada elemento esté en la secuencia (por linealidad de la esperanza).
- $P(p[i] > max(p[1], p[2], \ldots, p[i-1])) = \frac{1}{i}$.
  Entonces el valor esperado es $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} = \mathcal{O}(\log n)$.
- El valor esperado del tamaño de la LIS es $\mathcal{O}(\sqrt{n})$.

Problema integrador: Sorting Methods

Sorting Methods

Here are some possible methods using which we can sort the elements of an array in increasing order:

1. At each step, choose two adjacent elements and swap them.
2. At each step, choose any two elements and swap them.
3. At each step, choose any element and move it to another position.
4. At each step, choose any element and move it to the front of the array.

Given a permutation of numbers $1, 2, \ldots, n$, calculate the minimum number of steps to sort the array using the above methods.
Constraints: $1 \leq n \leq 2 \cdot 10^5$.

`http://cses.fi/problemset/task/1162`

Problema integrador: Sorting Methods

Sorting Methods

Here are some possible methods using which we can sort the elements of an array in increasing order:

1. At each step, choose two adjacent elements and swap them.
2. At each step, choose any two elements and swap them.
3. At each step, choose any element and move it to another position.
4. At each step, choose any element and move it to the front of the array.

Given a permutation of numbers $1, 2, \ldots, n$, calculate the minimum number of steps to sort the array using the above methods.
Constraints: $1 \leq n \leq 2 \cdot 10^5$.

`http://cses.fi/problemset/task/1162`

Solución oralmente.

Material de referencia

Bibliografía:

- Blog sobre permutaciones (Gracias Max!)

Problemas de práctica: (Además de los explicados durante la clase)

- `https://codeforces.com/contest/612/problem/E`
- `https://cses.fi/problemset/task/2214`
- `https://codeforces.com/contest/1768/problem/D`

Pueden consultarme durante esta semana, o me pueden enviar un mail a:

- mateocvelez@mi.unc.edu.ar