José Alejandro Pérez Márquez 310109800
Tania Michelle Rubí Rojas 315121719
Erick Bernal Márquez 317042522

# Tarea 4

Complejidad Computacional
Semestre 2023-1

## Ejercicio 1

- Demuestra que $\leq_p$ es una relación de orden en $NP$.

  Sean los lenguajes $A, B, C \in NP$. Para demostrar que $\leq_p$ es una relación de orden, debemos mostrar que $\leq_p$ es reflexiva, antisimétrica y transitiva. Dicho esto, procederemos a demostrar estas tres propiedades:

  i. **Reflexividad**. Queremos mostrar que $A \leq_p A$. Por definición, existe una función computable $f : \Sigma^* \to \Sigma^*$ tal que para cualquier $w$ se cumple que

  $$x \in A \Leftrightarrow f(x) \in A$$

  Tal función es la función `Identidad`. Esto significa que el lenguaje $A$ siempre se puede reducir a sí mismo. Por lo tanto, $A \leq_p A$.

  ii. **Antisimetría**. Queremos mostrar que si $A \leq_p B \wedge B \leq_p A$, entonces $A = B$.
  Supongamos que $A$ y $B$ son lenguajes tal que $A \leq_p B \wedge B \leq_p A$. Por definición, existen funciones computables $f : \Sigma^* \to \Sigma^*$ y $g : \Sigma^* \to \Sigma^*$, respectivamente, tal que para cualquier $x, y$ se cumple que

  $$x \in A \Leftrightarrow f(x) \in B$$
  $$y \in B \Leftrightarrow g(y) \in A$$

  Pero esto significa que, sin importar que elementos $x \in A$ y $y \in B$ tomemos, las imagenes de $f(x)$ y $f(y)$ son iguales, es decir $x \in A = g(y) \in A, y \in B = f(x) \in B$ lo que implica que $A = B$ en cuanto nivel de dificultad se refiere, es decir son equivalentes en dificultad.

  iii. **Transitividad**. Queremos mostrar que si $A \leq_p B \wedge B \leq_p C$, entonces $A \leq_p C$.
  Entonces, supongamos que $A, B$ y $C$ son lenguajes tal que $A \leq_p B$ y $B \leq_p C$. Por definición, existen funciones computables $f : \Sigma^* \to \Sigma^*$ y $g : \Sigma^* \to \Sigma^*$, respectivamente, tal que para cualquier $w, x$ se cumple que

  $$w \in A \Leftrightarrow f(w) \in B$$
  $$x \in B \Leftrightarrow g(x) \in C$$

  De este modo, podemos definir una nueva función $h : \Sigma^* \to \Sigma^*$ tal que para cualquier $w$ se cumple que

  $$w \in A \Leftrightarrow f(w) \in B \Leftrightarrow g(f(w)) \in C$$

  Dicha función $h$ es la composición $g(f(w))$. Sabemos que la composición $g \circ f$ se puede realizar pues el codominio de $f$ es igual al dominio de $g$. Así, esta nueva función es computable pues $f$ y $g$ también lo son. Por lo tanto, podemos concluir que $A \leq_p C$.

- ¿Cómo podríamos definir formalmente que dos problemas en $NP$ son equivalentes en dificultad?

  Utilizando propiedad antisimetrica. Esta propiedad significa

  $$x \in A \Leftrightarrow f(x) \in B$$
  $$y \in B \Leftrightarrow g(y) \in A$$

  Es decir, que de un problema podemos *"pasar"* a otro mediante algún algoritmo de transformación. Es aquí donde nos ayuda tal propiedad, pues si somos capaces de resolver $A$ entonces también podemos resolver $B$ y al revés también. Son igual de "fáciles" o igual de "díficiles".

  La propiedad concluye en $A = B$ lo cual tiene sentido si lo pensamos como niveles de dificultad y no como que son el mismo problema.

- ¿Utilizando $\leq_p$, es posible establecer relación de orden en $P$? ¿Se puede establecer algún otro tipo de orden, respecto a la dificultad de problemas, en $P$?

  Sí, acabamos de mostrar que existe un orden en $NP$ y como $P \subseteq NP$ significa que el orden tambiéen existe en $P$. Tal orden es un orden parcial.
  Podríamos definir un orden *total* si logramos demostrar que para todo problema $A, B \in P$ entonces $A \leq_p B$ o $B \leq_p A$, esto es que todos los problemas se pueden comparar entre sí, que siempre existe alguna reducción entre cuales quiera dos problemas.

---

# Ejercicio 2

¿Cuáles de los siguientes problemas pertenecen a $NP$? ¿Cuáles a $coNP$? Justifica tu respuesta dando la demostración de pertenencia a las clases correspondientes:

- $DOM = \{(\text{G, K}) \mid \text{G tiene un conjunto dominante de K nodos}\}$

  Un subconjunto de vértices C de G es un conjunto dominante de G, si todo otro vértice de G es adyacente a algún vértice de C.

  Afirmamos que $DOM$ es NP

  Sea una grafica $G = (V, E)$ donde $|V| = n, |E| = m$ y $C \subseteq V$
  Proponemos el ceritifcado $w$ como un conjunto $C \subseteq V$ de k vértices de G.

  $$w = \{v_1, v_2, ..., v_k\}$$

Para validar esta solución tenemos verificar que todos los vértices que no forman parte de $C$ (es decir los vértices de $V - C$) son adyacentes a alguno de los vértices de este conjunto. Esto se puede hacer con el siguiente algoritmo.

1. Tomamos los vértices de la gráfica que no están en $C$, es decir el complemento del certificado $w$ como $w^c$.
2. Para cada vértice de $w^c$. verificamos que sea adyacente a algún vértice de $C$, o sea a $w$.
3. Si el vértice **no** es adyacente a algún vértice de $w$ entonces rechazamos, si no entonces continuamos verificando con el siguiente vértice.
4. Una vez hayamos verificado que todos los vértices de $w^c$ son adyacentes a algún vértice en $w$ aceptamos.

En el peor caso nos dan un certificado con un solo vértice, por lo que tenemos que recorrer $n-1$ vértices del certificado $w^c$ lo que hace la complejidad $O(n)$, Sin embargo estamos verificando con cada elemento en $w$ con longitud $k$, así la complejidad es $O(n \times k)$. Por lo tanto el problema es *NP*.

- $ISO = \{(G_1, G_2) \mid G_1 \text{ y } G_2 \text{ son grafos isomorfos}\}$

  Dos grafos son isomorfos si son iguales salvo por los nombres de sus arcos (pares de vértices).

  Recordemos que dos gráficas $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ finitas son isomorfas si existe una biyección entre los vértices que preserva la estructura de las adyacencias, es decir, si existe $f : V_1 \to V_2$ tal que si $u, v \in V_1$ son adyacentes en $G_1$ si y sólo si $f(u), f(v)$ son adyacentes en $G_2$.

  El certificado serán dos matrices de adyacencias.

$$w = \left\{ \begin{bmatrix} e^1_{11} & e^1_{12} & \dots & e^1_{1n} \\ e^1_{21} & e^1_{22} & \dots & e^1_{2n} \\ e^1_{n1} & e^1_{n2} & \dots & e^1_{nn} \end{bmatrix}, \begin{bmatrix} e^2_{11} & e^2_{12} & \dots & e^2_{1n} \\ e^2_{21} & e^2_{22} & \dots & e^2_{2n} \\ e^2_{n1} & e^2_{n2} & \dots & e^2_{nn} \end{bmatrix} \right\}$$

  Lo que hará nuestro algoritmo de verificación es tomar dos vértices $v^1_i$ y $v^1_j$ de la matriz de $G_1$ que forman una entrada $e_{i,j}$ en la matriz, computar la función biyectiva para cada vértice y verificar que la entrada que forman dicho computo de vértices $e^2_{i',j'}$ en la matriz de la gráfica $G_2$ tenga el mismo valor que la entrada $e^1_{i,j}$

  1. Recorremos cada entrada de la matriz de $G_1$ y por cada entrada hacemos lo siguiente
  2. Obtenemos los vértices que forman dicha entrada $v^1_i$ y $v^1_j$ y guardamos el valor de la entrada en una variable llamada $valor1$
  3. Computamos la función para cada vértice. $f(v^1_i) = v^2_{i'}$ y $f(v^1_j) = v^2_{j'}$

4. Buscamos el valor de la entrada $v^2_{i'}$ y $v^2_{j'}$, o bien $e^2_{i',j'}$ en la matriz de $G_2$
5. Si el valor obtenido es el mismo que el da la variable $valor1$ continuamos con el otro vértice. Si no lo son rechazamos. Así hasta finalizar todas las entradas de la matriz de $G_1$

Como recorremos toda la matriz de la grafica $G_1$ entonces la complejidad es $O(|V|^2)$, podríamos pensar que también recorremos la matriz de la gráfica $G_2$, pero lo hacemos de distinto modo, ya que solo accedemos a sus entradas mediante la función biyectiva la cual se hace por cada iteración de la matriz de $G_1$, por lo cual sigue teniendo complejidad $O(|V|^2)$. De esta manera demostramos que $ISO \in NP$

- $HAM\ MOD = \{(G_1, G_2) \mid G_1$ tiene un ciclo Hamiltoniano y $G_2$ no tiene un ciclo Hamiltoniano$\}$

  Recordemos que un ciclo hamiltoniano es un camino cerrado que visita cada vértice de la gráfica exactamente una vez.

  Observemos que verificar que una gráfica tiene un ciclo hamiltoniano es exactamente el mismo problema que verificar si no lo tiene, por lo que dadas dos gráficas $G_1$ y $G_2$ el problema de verificar si $G_1$ tiene un ciclo hamiltoniano y $G_2$ no tiene un ciclo hamiltoniano es equivalente al problema de verificar si las dos tienen un ciclo hamiltoniano y aprobar sólo cuando $G_1$ sea verdadero y $G_2$ sea falso.

  Sea entonces una gráfica $G_1$, y un ciclo hamiltoniano $C$ de tamaño k : $C = v_1, v_2, ...v_k$ . Para verificar que efectivamente es un ciclo hamiltoniano, tenemos que verificar que para cualesquiera dos vértices $v_i, v_{i+1} \in C$, hay entre estos dos vértices una arista que los une y además verificar que el último y el primer vértice sean el mismo, si no se cumple esto, mandamos falso, si sí mandamos verdadero y en cualquier otro caso falso. Observemos que para hacer esto, primero contamos si todos los vértices de $G_1$ están en el ciclo (k pasos) y luego verificamos si cada uno está conectado con el siguiente (k pasos) y si el último es el mismo que el primero (un paso), lo cual toma tiempo $O(2|V| + 1)$. Lo mismo se puede hacer con $G_2$ por lo que $HAMMOD$ tomaría tiempo $O(2(2|V| + 1))$ el cual es polinomial y por tanto $HAMMOD \in NP$.

- $FACT = \{n \in N, K \in Z^+ \mid n$ tiene un factor primo menor o igual a $K\}$

  Sugerencia: Puedes asumir que $PRIMOS$ (el problema de decidir si un número es primo o no) está en P.

  Recordemos que dado $n \in \mathbb{N}$ y $k \in \mathbb{Z}$, que n tenga un factor primo menor o igual a k es equivalente a que exista $s \in \mathbb{Z}$ que sea primo y que divida a n.

  Así, para demostara que el problema está en $co-NP$ debemos ver si hay un certificado que pueda verificar en tiempo polinomial la negación del problema, es decir, que dado

$n \in \mathbb{N}$ y $k \in \mathbb{Z}$ sus factores primos no son menores o iguales a k. Así, dado un certificado, es decir una factorización en primos de n, dado que $PRIMOS \in P$ podemos verificar en tiempo polinomial que cada factor es primo , que su producto es n y que cada uno es menor o no a k. Con esto demostramos que $FACT \in co - NP$.

---

# Ejercicio 3

Considera el siguiente problema:
Una empresa utiliza listas de correos para enviar comunicados a sus trabajadores. Cada trabajador puede estar dado de alta en diferentes listas de correos, una por cada departamento. Supón que la lista más grande tiene registrados $m$ correos. Dada la naturaleza de la empresa, algunos trabajadores tienen roles asignados en más de un departamento, por tanto cada trabajador puede estar registrado en más de una lista de correos. El gerente de la empresa está organizando una reunión para conocer la opinión de los trabajadores y tomar decisiones importantes en la empresa. Sin embargo, en la sala de juntas solo hay espacio para $k$ personas. El gerente desea que cada departamento esté representado en la reunión por al menos un trabajador, por tanto necesita construir la lista de correos de los $k$ invitados, para enviar el correo con la invitación a la reunión. ¿Es posible construir dicha lista? En otras palabras: ¿Es posible que cada departamento tenga al menos un miembro asistente a la reunión?
Demuestra que dicho problema es NP-completo.

Es posible construir dicha lista sin embargo al demostrar que es $NP$-completo quiere decir que el algoritmo es de tiempo polinomial **no** determinista. Primero mostraremos cómo codificamos el problema.

- **Codificación**

  Tendremos una colección de listas donde cada lista representa un departamento, y los elementos de las listas representan a los correos en los departamentos. Por ejemplo

  $departamentos = \{$informática, RRHH, ..., salud$\}$

  Y por cada uno de estos tendremos una cantidad a lo más $m$ de correos

  $departamentos = \{[c_1, c_2, ...c_m], [c'_1, c'_2], ..., [c''_1, c''_2, c''_3]\}$

  La lista de invitados de a lo más longitud k es una lista que contenga a los correos de los departamentos. Por ejemplo.

  $lista = [c_1, c'_1, c''_2, ..., c_k]$

  No necesariamente todos los correos pues es justo el problema que queremos resolver, en realidad es nuestro certificado $k$ para verificar que exista al menos una persona por departamento para asistir a tal reunión.

Formalmente

Ejemplar: Sea una colección de subconjunto de $U$ y un entero $k$ con $d_1, d_2, ...d_n \subseteq U$ (los departamentos)
Pregunta: ¿Existe $w \subseteq U$ (la lista) con a lo más $k$ elementos tal que tenemos al menos un elemento de cada $d_i$?

- **Pertenencia NP**

  Proponemos el certificado como una lista de $c_i \in d_n$ elegidas arbitrariamente de a lo más tamaño $k$. $l = [c_1, c_2, ..., c_k]$

  Nuestro algoritmo

  1. Verificamos que la lista (el certificado) sea de a lo más tamaño $k$. Si la lista tiene longitud mayor a $k$ rechazamos
  2. Por cada elemento $c_i$ en la lista verificamos que esté en algún $d_j \subseteq U$. Para ello recoremos todos los subconjuntos $d_j$.
     Si existe algún $c_i$ que no esté en ningún $d_j$ entonces rechazamos
     Si nunca rechazamos es porque obtuvimos que todos los $c_i$ estuvieron en al menos algún $d_j$, por lo que aceptamos

  Como recorremos todo el certificado tenemos que la complejidad es $O(k)$, no obstante la recorremos por cada $d_j$, suponiendo que tengamos $n$ departamentos la complejidad es $O(k \times n)$, por cada $d_j$ recorremos todos los $c_i$, el maximo numero de miembros que tiene cada $d_j$ es $m$, así la complejidad total es $O(k \times n \times m)$ donde $k$ es la longitud del certificado, $n$ la cardinalidad de $d_j$ y $m$ la mayor longitud de algún $d_j$

- **NP-Completo**

  Para ello tomaremos SAT, que sabemos es *NP-Díficil* y lo reduciremos a este problema que adoptaremos como *depa* (a falta de un mejor nombre e imaginación). Es decir SAT $\leq_p$ *depa*

  Nuestra transformación $f(x)$ toma las clausulas de SAT y las mapea a subconjuntos $d_j$ de la siguiente manera

  1. Cada variable en una clausula en SAT es un $c_i$ de los distintos $d_j$. Por ejemplo

$$(p \vee q \vee r) \to \{p, q, r\} = d_1$$

$$(q \vee r) \to \{q, r\} = d_2$$

$$(p \vee s) \to \{p, s\} = d_3$$

$$...$$

$$s \to \{s\} = d_j$$

2. Así cada clausula de SAT es un $d_j$

$$(p \vee q \vee r) \wedge (q \vee r) \wedge (p \vee s) \wedge ... \wedge u \rightarrow \{p, q, r\}, \{q, r\}, \{p, s\}, ..., \{u\}$$

Supongamos que la formula $\phi$ es de longitud $b$, es decir, tiene $b$ símbolos en toda la formula sin importar repetidos, como recorremos toda $\phi$ para hacer el mapeo entonces tomará tiempo $O(b)$, una operación por cada símbolo.

La asignación de valores de verdad para $SAT$ será nuestra lista $w$ de los $c_i$ para *depa* siendo estos los que tenga el valor verdadero, por ejemplo si tenemos que

$$p = True$$

$$q = True$$

$$r = False$$

$$s = False$$

$$\rightarrow w = \{p, q\}$$

esto lo utilizaremos para demostrar que la transformación es correcta, es decir

$$x \in SAT \Leftrightarrow f(x) \in depa$$

$\Rightarrow$ Supongamos $x \in SAT$, esto quiere decir que existe alguna asignación de variables $p, q, r..., s$ tal que hacen verdadera la formula $\phi$ en SAT, hay un valor verdadero en cada cluasula. Como nuestras variables pasan a ser nuestra lista $w$ en el problema de *depa* quiere decir que hay al menos un $c_i$ para cada $d_j$, que proveiene justo de las variables con valor verdadero.

$\Leftarrow$ Por contradicción. Supongamos ahora que $x \notin SAT$, significa que existe al menos una cluasula en $\phi$ tal que cada variable es falsa, tomemos alguna $v_i$ de esta clausula. Esta variable no se verá reflejada en nuestra lista $w$ de *depa* (de hecho niguna de las que están en dicha cluasula) por lo que faltarán $c_i$ en $w$ que estén en algún $d_j$. En particulae el $d_j$ que fue mapeado por tal cluasula. ■

---

# Ejercicio 4

¿Cuál de los siguientes problemas es más difícil? Justifica y demuestra formalmente tu afirmación

- **Problema:** 3SAT-IGUALDAD
  **Ejemplar:** Una fórmula boolena $\phi$ en FNC (Forma Normal Conjuntiva) en la cual el número de cláusulas es igual al número de variables
  **Pregunta:** ¿Existe una asignación que satisface $\phi$?

- **Problema:** 3SAT BALANCIADO
  **Ejemplar:** Una fórmula boolena $\phi$ en FNC en la cual cada cláusula contiene 3 literales, y cada variable en $\phi$ aparece negada y sin negación el mismo número de veces
  **Pregunta:** ¿Existe una asignación que satisface $\phi$?

Observemos que si toda fórmula de $3SAT - IGUALDAD$ es también una fórmula de $3SAT$ por lo que basta demostrar que 3SAT $\leq_p$ 3SAT BALANCEADO.

**Demostración.** Propongamos que $x$ una fórmula $\phi \in 3SAT$ y considere también f(x) como una fórmula $\beta \in 3SAT - BALANCEADO$
Debemos hacer un algoritmo para hacer la transformación y demostrar que ese algoritmo cumple con satisfacibilidad y Balance.

Primero dado que tenemos los elementos de $\phi$ los copiaremos y pondremos en $\beta$, esto nos tomará $O(n)$ operaciones y quedaría balancearlo, para eso, creamos una lista $\gamma$, cabe mencionar que no ocupamos un arreglo pues necesitamos una estructura de datos que pueda ir creciendo. A su vez, sea y una variable para almacenarlos datos de x, si es negada o no.

En la lista $\gamma$ se almacenara duplas de la forma $(x_i\ ,\ y_i)$, el ir creando esta lista estará acotado por $O(n)$. Como apenas se iniciara el algoritmo $y_i$ empieza con valor igual a 0, posteriormente hacemos un recorrido secuencial de izquierda a derecha a $\beta$, el valor de $y_i$ cambiará cada que encontremos una $x_i$ el valor de $y_i$ se actualizara a $y_i+1$, en caso de que este negada se actualizará a $y_i - 1$, esto tomará $O(1)$ operaciones por actualización de *gamma* pues los valores se tendrá que actualizar cada que se encuentre una variable en la formula, posteriormente tomamos una dupla $(x_i, y_i)$ siempre y cuando $y_i \neq 0$.

Aquí tendremos dos opciones. En el caso de que $y_i < 0$ se agregará a $\beta$ una clausula $(x_i \vee True \vee True)$ (esto tomará O(1) operaciones) y actualizaremos $y_i = y_i + 1$, el otro caso es análogo, cuando $y_i > 0$ se agregará a $\beta$ una clausula $(x_i \vee True \vee True)$ y actualizaremos $y_i = y_i + 1$ y así sucesivamente. En este paso $\beta$ esta acotado por las veces que salga n variables en la formula y esto quiere decir que se agregaran n clausulas a $\beta$. Finalmente se regresa $\beta$ el cuál es 3SAT reducido polinomialmente a 3SAT BALANCEADA

Para demostrar la satisfacibilidad notemos que el planteamiento del problema dice que ambas formulas están en FNC, para satisfacer $\gamma$ solo debemos tomar las formulas que comparte con $\phi$, pues True esta en la clausulas y siempre al ser evaluado será True y esto quiere decir que la satisfacibilidad no cambia respecto a la que $\phi$ tiene.

En el caso para demostrar balance se debe mostrar que cada variable no negada o negada debe aparecer el mismo número de veces, en este caso con cláusulas de 3 literales. En nuestro algoritmo cada que se agrega una clausula se lleva el conteo en $y_i$, así hasta que nuestra lista ya no haya valores $y_i$ para alguna $x_i$ y cuando se cumpla esto significara que $\beta$ estará balanceada. ■

---

# Ejercicio 5

Sean $L_1, L_2 \in NP$ ¿Cuáles de las siguientes afirmaciones son ciertas o falsas? Justifica tu respuesta:

- $L = L_1 \cup L_2, L \in NP$

  Solución: La afirmación es **verdadera**. Para justificar esto, mostraremos que la clase NP es cerrada bajo la operación de unión. Para mostrar esto, hay que demostrar que existe una máquina de turing no-determinista que decide a $A \cup B$ en tiempo polinomial.

  Entonces, supongamos que $A$ y $B$ son dos lenguajes tales que $A, B \in NP$. Por definición, esto significa que existen máquinas de Turing no-deterministas $M_1$ y $M_2$ que deciden a $A$ y $B$ en tiempo polinomial, respectivamente. Vamos a definir una máquina de turing no-determinista $M$ que decida a $A \cup B$ en tiempo polinomial:

  ```
  M = Para la entrada w
  1. Ejecutar M_1 sobre w. Si M_1 acepta, entonces M acepta.
  2. Ejecutar M_2 sobre w. Si M_2 acepta, entonces M acepta.
  3. En otro caso, rechaza.
  ```

  Recordemos que la definición de la unión es la siguiente:

  $$x \in X \cup Y \Leftrightarrow x \in X \vee x \in Y$$

  Entonces, basta con que $w$ sea aceptada en alguna de las máquinas $M_1$ o $M_2$ (puede ser que sea aceptada en ambas) para que la máquina $M$ acepte a $A \cup B$. La máquina $M$ es no-determinista pues en los pasos uno y dos, cuando las máquinas $M_1$ y $M_2$ se ejecutan, realizan pasos no-deterministas. Y como ambas máquinas $M_1$ y $M_2$ toman tiempo polinomial, entonces $M$ se ejecuta en tiempo polinomial.

  Así, hemos probado que $A \cup B \in NP$.

- $L = L_1 \cap L_2, L \in NP$

  Solución: La afirmación es **verdadera**. Para justificar esto, mostraremos que la clase NP es cerrada bajo la operación de intersección. Para mostrar esto, hay que demostrar que existe una máquina de turing no-determinista que decide a $A \cap B$ en tiempo polinomial.

  Entonces, supongamos que $A$ y $B$ son dos lenguajes tales que $A, B \in NP$. Por definición, esto significa que existen máquinas de Turing no-deterministas $M_1$ y $M_2$ que deciden a $A$ y $B$ en tiempo polinomial, respectivamente. Vamos a definir una máquina de turing no-determinista $M$ que decida a $A \cap B$ en tiempo polinomial:

```
M = Para la entrada w
1. Ejecutar M_1 sobre w. Si M_1 rechaza, entonces M rechaza.
2. Ejecutar M_2 sobre w. Si M_2 rechaza, entonces M rechaza.
3. En otro caso, acepta.
```

  Recordemos que la definición de la intersección es la siguiente:

$$x \in X \cap Y \Leftrightarrow x \in X \wedge x \in Y$$

  Entonces, tenemos que $w$ debe ser aceptaba tanto por $M_1$ como por $M_2$ para que la máquina $M$ acepte a $A \cap B$. La máquina $M$ es no-determinista pues en los pasos uno y dos, cuando las máquinas $M_1$ y $M_2$ se ejecutan, realizan pasos no-deterministas. Y como ambas máquinas $M_1$ y $M_2$ toman tiempo polinomial, entonces $M$ se ejecuta en tiempo polinomial.

  Así, hemos probado que $A \cap B \in NP$.

- Si $L_1$ es $coNP-$ *Difícil* entonces $N = NP$

  Solución:

---

# Referencias


- Orden parcial

  `https://es.wikipedia.org/wiki/Conjunto_parcialmente_ordenado`

- Orden parcial

  `https://blog.nekomath.com/teoria-de-los-conjuntos-ordenes-parciales-y-ordenes-parci`

- Orden total

  `https://blog.nekomath.com/algebra-superior-i-ordenes-parciales/`

- Problemas NP y reducciones

  Temas del presente curso