
**Universidad Nacional Autónoma de México**
**Facultad de Ciencias**

## Complejidad Computacional

## Tarea 4

Bonilla Reyes Dafne - 319089660
García Ponce José Camilo - 319210536


# Problemas P y NP

## Ejercicio 1

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

- $K$ - $COLORACIÓN = \{G \mid G$ tiene una coloración de sus vértices, con $K$ colores$\}$

  **Demostración de pertenencia:**

  Este problema pertenece a $NP$, ya que no se conoce un algoritmo en tiempo polinomial que lo resuelva.

  Vamos a suponer que $K$ es menor o igual al número de vértices en $G$, ya que si $K$ es mayor estrictamente que el número de vértices, el problema se vuelve muy fácil. Para esto diremos que el certificado $u$ serán $K$ listas que contengan a los vértices de $G$ (sin que el mismo vértice pueda estar en dos listas distintas), donde si el vértice $v$ está en la $i$-ésima lista, entonces $v$ será coloreado con él $i$-ésimo color.

  Notemos que todos los elementos de todas las listas serán $V$ (con $V$ la cantidad de vértices en $G$), por lo que $u$ tendrá una longitud de $O(V)$ y, por lo tanto, será un certificado válido (ya que $O(V) \in O(V^2)$ y eso es polinomial en relación con la entrada original, es decir es polinomial en relación con $G$, ya que la entrada que represente a $G$ debe contener todos sus vértices, ya sea explícitamente u obtenerlos con base en las aristas).

  Ahora, veamos una Máquina de Turing que reciba la gráfica $G$ y el certificado $u$ y revise si la gráfica tiene una coloración válida usando los valores del certificado, para esto usaremos un algoritmo en el modelo RAM (recordemos que dependiendo de la representación de las gráficas usada, este algoritmo puede cambiar algo, pero no mucho, la idea general se mantiene y la complejidad seguirá siendo polinomial).

  - `Input:` Una gráfica $G$, para este caso diremos que la gráfica se va a dar como la lista de adyacencias de cada vértice, y certificado $u$ con el color que tendrá cada vértice de la gráfica, para esto usaremos $K$ listas (como se dijo cuando se explicó el certificado).
  - `Output:` 1 si la coloración sugerida por el certificado es válida, 0 en otro caso.

  1. Primero, iteraremos sobre las $K$ listas dadas por el certificado y para cada lista $i$ realizaremos lo siguiente:

- Iteramos sobre los vértices de la lista $i$ y por cada vértice $v$ haremos lo siguiente:
  - Recorremos la lista de adyacencias del vértice $v$ y para cada vértice $z$ que está en la lista de adyacencias de $v$ vamos a recorrer la lista $i$ para ver si lo encontramos. Si no lo encontramos, seguimos con el algoritmo, pero si lo encontramos significa que dos vértices que son vecinos tienen el mismo color y, por lo tanto, regresamos 0 y terminamos el algoritmo.

  ***Nota:*** Si queremos que sean $K$ colores y no menos, tenemos que revisar que las $K$ listas del certificado no estén vacías, pero si pueden ser $k$ o menos colores, entonces este paso se puede saltar.

2. Luego de revisar que todos vecinos de cada vértice tengan un color diferente, regresamos un 1 y termina el algoritmo.

**Complejidad del algoritmo:**

En el paso 1 se realizan varias iteraciones (`fors`), entonces veamos cuando veces se realiza cada una:

- La iteración más externa es en la que recorremos las $K$ listas y los vértices de cada una de esas listas, entonces como sabemos que en total todas las listas tienen $V$ elementos (cada vértice está en solo una lista) entonces esta iteración es de $O(V)$.
- La siguiente iteración que es sobre los vértices adyacentes de cada vértice en las $K$ listas. Es decir, estamos iterando sobre los vecinos de cada vértice, y sabemos que un vértice puede tener a lo más $V - 1$ vecinos por lo cual esta iteración es de $O(V)$
- La última iteración es sobre los vértices de la lista de los vértices del mismo color del vértice de la iteración más externa, y la iteración se hace para encontrar si algún vecino de este vértice está en la lista de vértices del color. Notemos que la operación de ver si dos elementos en una lista son iguales nos toma $O(1)$ (esto puede cambiar si los objetos son muy complejos, pero suponemos que para representar los nodos usaremos números, por lo tanto, está bien $O(1)$)

  Ahora, veamos que cada una de las $K$ listas puede tener a lo más $V$ elementos (si todos los vértices solo tienen un color), por lo tanto, esta iteración es de $O(V)$,

Dicho de otra forma, iteramos sobre todos los vértices de la gráfica, luego dentro de esa iteración iteramos sobre todos los vecinos del vértice $v$ y dentro de esa iteración iteramos sobre todos los vértices en la lista del mismo color que $v$, para ver si el vecino $z$ del vértice $v$ está en la lista.

Usando todas las complejidades dadas antes y la extraña explicación de arriba, notamos que todo este paso nos toma $O(V^3)$, y para el paso 2 como solo es el regreso de un valor, nos toma $O(1)$. Por lo tanto, todo el algoritmo nos tomó $O(V^3)$, lo cual es una complejidad polinomial, ya que la complejidad es un polinomio.

Por último, usando la tesis de Church-Turing (*complexity-theoretic Church–Turing thesis*), sabemos que existe una Máquina de Turing determinista que resuelve/computa lo mismo que este algoritmo en a lo más una diferencia de tiempo polinomial, por lo tanto, podemos concluir que la Máquina de Turing tendrá tiempo polinomial, entonces concluimos que el problema K-Coloración pertenece a *NP* usando la definición alternativa de *NP*, ya que dimos un certificado de a los más longitud polinomial y una Máquina de Turing determinista que sirve para verificar en tiempo polinomial.

- $CONECTIVIDAD = \{G \mid G \text{ es una gráfica conexa}\}$

**Demostración de pertenencia:**

Este problema pertenece a $P$.

Veamos una Máquina de Turing que pueda resolver este problema. Para esto usaremos un algoritmo en el modelo RAM (recordemos que dependiendo de la representación de las gráficas usada, este algoritmo puede cambiar algo, pero no mucho, la idea general se mantiene y la complejidad seguirá siendo polinomial).

- `Input:` Una gráfica $G$, para este caso diremos que la gráfica se va a dar como una lista de vértices y una lista de aristas.
- `Output:` 1 si $G$ es conexa, 0 en caso de que no sea conexa.

1. Lo primero que se realizará será usar una lista $A$ de tamaño $V$ (donde $V$ es el número de vértices de la gráfica) y una función recursiva donde se realice un recorrido DFS en el cual cuando marquemos a un vértice como visitado lo agregaremos a la lista $A$. Por lo tanto, haremos un recorrido DFS empezando en el primer vértice de la lista de vértices y de esta manera agregar a la lista los vértices visitados.
2. Ya que terminó el recorrido DFS, vamos a tener en la lista $A$ todos los vértices visitados a partir del primer vértice. Ahora vamos a recorrer la lista de vértices y por cada vértice $v$ en la lista de vértices vamos a recorrer la lista $A$ hasta encontrar al vértice $v$ en la lista $A$, cuando se encuentre al vértice $v$ seguimos con el siguiente vértice en la lista de vértices, pero si no logramos encontrar a $v$ entonces vamos a regresar un 0 y termina el algoritmo.
3. Luego de revisar que todos los vértices en la lista de vértices se encuentran en la lista $A$, regresamos un 1 y termina el algoritmo.

**Complejidad del algoritmo:**

- **Paso 1:** Toma $O(V+E)$ (con $V$ la cantidad de vértices en $G$ y $E$ la cantidad de aristas en $G$), ya que estamos haciendo un recorrido DFS y sabemos que hacerlo nos toma esa complejidad, además agregar elementos a una lista nos toma $O(1)$, por lo tanto, no aumenta la complejidad de realizar el recorrido DFS.
- **Paso 2:** Notemos que la operación de ver si dos elementos en una lista son iguales nos toma $O(1)$ (esto puede cambiar si los objetos son muy complejos, pero suponemos que para representar los nodos usaremos números, por lo tanto, está bien $O(1)$).

  Entonces notemos que se recorre una vez la lista de vértices de $G$ por lo tanto, esto nos toma $O(V)$ y también se recorre la lista $A$ que nos toma a los más $O(V)$ (si se recorrieron todos los vértices), y como por cada vértice de la lista de vértices se recorre la lista $A$, además podemos regresar valores lo cual nos toma $O(1)$, entonces todo el paso 2 nos toma $O(V^2)$.
- **Paso 3:** Como solo es el regreso de un valor, nos toma $O(1)$.

Por lo tanto, todo el algoritmo nos tomó $O(V^2+V+E)$, lo cual es una complejidad polinomial, ya que la complejidad es un polinomio.

Por último, usando la tesis de Church-Turing (*complexity-theoretic Church–Turing thesis*), sabemos que existe una Máquina de Turing determinista que resuelve/computa lo mismo que este algoritmo en a lo más una diferencia de tiempo polinomial, por lo tanto, podemos concluir que la Máquina de Turing tendrá tiempo polinomial, entonces concluimos que el problema Conectividad

pertenece a $P$ usando la definición de $P$, ya que existe una Máquina de Turing determinista que sirve resolver este problema en tiempo polinomial. Y por lo visto en la clase del 8/10/24, sabemos que $P \subseteq NP$, entonces el problema Conectividad también pertenece a $NP$.

- $DOM = \{(G, K) \mid G$ 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 $C$.

  **Demostración de pertenencia:**

  Este problema pertenece a $NP$, ya que no se conoce un algoritmo en tiempo polinomial que lo resuelva.

  Vamos a suponer que $K$ es menor o igual al número de vértices en $G$, ya que si $K$ es mayor estrictamente que el número de vértices, el problema se vuelve muy fácil. Para esto diremos que el certificado $u$ será un conjunto de $K$ vértices de $G$ (sin que el mismo vértice pueda estar dos veces en el certificado), de esta manera los vértices en el certificado serna los vértices que formaran el conjunto dominante.

  Notemos que la cantidad de vértices en el certificado son $K$, entonces el certificado tendrá longitud $O(K)$, y además sabemos que $K \leq V$ (con $V$ la cantidad de vértices en $G$), por lo tanto, es un certificado válido (ya que $O(K) \in O(V)$ y eso es polinomial en relación con la entrada original, es decir es polinomial en relación con $G$, ya que la entrada que represente a $G$ debe contener todos sus vértices, ya sea explícitamente u obtenerlos con base en las aristas).

  Ahora veamos una Máquina de Turing que reciba la gráfica $G$ y el certificado $u$ y revise si los vértices del certificado forman un conjunto dominante para $G$, para esto usaremos un algoritmo en el modelo RAM (recordemos que dependiendo de la representación de las gráficas usada, este algoritmo puede cambiar algo, pero no mucho, la idea general se mantiene y la complejidad seguirá siendo polinomial).

  - `Input:` Una gráfica $G$, para este caso diremos que la gráfica se va a dar como la lista de adyacencias de cada vértice y una lista donde están todos los vértices de $G$, y certificado $u$ con un conjunto de $K$ vértices, para esto usaremos una lista, la cual llamaremos $A$.
  - `Output:` 1 si el conjunto de vértices del certificado son un conjunto dominante en $G$, 0 en otro caso.

  1. Lo primero que se va a realizar es revisar que la longitud de la lista $A$ (la que representa al certificado $u$) tenga longitud igual a $K$, si no lo tiene regresamos 1 y termina el algoritmo, pero la longitud es igual a $K$, entonces seguimos.
  2. Ahora vamos a crear una nueva lista a la cual llamaremos $B$, después vamos a iterar sobre los vértices de la lista $A$, y para cada vértice $v$ de la lista $A$ realizaremos lo siguiente:
     - Metemos al vértice $v$ a la lista $B$ y vamos a iterar en la lista de adyacencias del vértice $v$ y para cada vértice $u$ en la lista de adyacencias del vértice $v$ haremos lo siguiente:
       - A cada vértice $z$ (los vecinos de $v$) lo meteremos a la lista $B$, para que al final tengamos en la lista $B$ a todos los vértices de la lista $A$ y a sus vecinos.
  3. Después vamos a iterar sobre todos los vértices de $G$ (se puede usar a lista de vértices de $G$ o de otro modo, dependiendo de como se codifique $G$), y para cada vértice $v$ en $G$ realizaremos lo siguiente:

- Iteramos sobre los vértices de la lista $B$ hasta encontrar al vértice $v$ (es decir, estamos buscando si $v$ está en la lista $B$), si no logramos encontrar a $v$ entonces regresamos 1 y termina el algoritmo, pero si lo logramos encontrar entonces seguimos iterando sobre los vértices de $G$.

4. Luego de revisar que todos los vértices de $G$ estén en la lista $B$, regresamos un 1 y termina el algoritmo.

**Complejidad del algoritmo:**

- **Paso 1:** Solo se revisa que la longitud de una lista sea $K$, por lo tanto, sabemos que esto nos toma $O(1)$.
- **Paso 2:** Se itera sobre los vértices de la lista $A$ que sabemos que son $K$, por lo tanto, esta iteración nos toma $O(K)$, luego mientras se itera estamos agregando elementos a una lista lo cual nos toma $O(1)$ y también se itera sobre los vecinos de un vértice. Sabemos que un vértice puede tener a lo más $V-1$ vecinos, por lo tanto, iterar sobre los vecinos nos toma $O(V)$ (con $V$ la cantidad de vértices en $G$), y también agregamos a una lista, por lo que todo el paso 2 nos va a tomar $O(KV)$.
- **Paso 3:** Primero iteramos sobre los vértices en $G$, entonces esta iteración nos toma $O(V)$ y en la iteración también vamos a iterar sobre la lista $B$. Ahora, notemos que $B$ puede tener a lo más $KV$ elementos (ya que tiene los $K$ elementos de la lista $A$ y a lo más cada uno de estos elementos agrego $V-1$ vecinos), entonces iterar sobre la lista $B$ nos toma $O(KV)$ y también vamos a estar buscando si un vértice está en una lista, esto se hace con comparaciones.

  Notemos que la operación de ver si dos elementos en una lista son iguales nos toma $O(1)$ (esto puede cambiar si los objetos son muy complejos, pero suponemos que para representar los nodos usaremos números, por lo tanto, está bien $O(1)$), además podemos regresar valores lo cual nos toma $O(1)$, entonces con todo esto junto notamos que el paso 3 nos toma $O(KV^2)$, ya que por cada vértice en $G$ vamos a recorrer la lista $B$ una vez.
- **Paso 4:** Como solo es el regreso de un valor, nos toma $O(1)$.

Por lo tanto, todo el algoritmo nos tomó $O(KV^2)$ (recordando que $K \leq V$, podríamos verlo como $O(V^3)$), lo cual es una complejidad polinomial, ya que la complejidad es un polinomio.

Por último, usando la tesis de Church-Turing (*complexity-theoretic Church–Turing thesis*), sabemos que existe una Máquina de Turing determinista que resuelve/computa lo mismo que este algoritmo en a lo más una diferencia de tiempo polinomial, por lo tanto, podemos concluir que la Máquina de Turing tendrá tiempo polinomial, entonces concluimos que el problema Dom pertenece a *NP* usando la definición alternativa de *NP*, ya que dimos un certificado de a los más longitud polinomial y una Máquina de Turing determinista que sirve para verificar en tiempo polinomial.

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

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

**Demostración de pertenencia:**

Este problema pertenece a *NP*, ya que no se conoce un algoritmo en tiempo polinomial que lo resuelva.

Primero recordemos una defunción de isomorfismo, dos gráficas $G[V, E]$ y $G'[V', E']$ son isomorfas si existe una función biyectiva $f : V \Rightarrow V'$ tal que la arista $\{e_1, e_2\} \in E$ si y solo si la arista

$\{f(e_1), f(e_2)\} \in E'$. Para esto diremos que el certificado $u$ será una permutación sin repetición de los vértices de $G_2$, de esta manera va a ser como una representación de una función biyectiva $f$ de los vértices de $G_1$ a los vértices de $G_2$ (notemos que esto pasa si $G_1$ y $G_2$ tienen la misma cantidad de vértices), esto debido a que la permutación la vamos a "guardar.[en] un arreglo para que si el vértice $v$ de la gráfica $G_1$ está en la posición $i$ de la lista de vértices de $G_1$ entonces $f(v)$ va a ser el vértice en la posición $i$ de la permutación de vértices de $G_2$, de esta manera notamos que a cada vértice de $G_1$ le corresponde un único vértice de $G_2$, de esta manera esa una función biyectiva.

Notemos que la cantidad de vértices en la permutación son $V_2$ (con $V_2$ la cantidad de vértices en $G_2$), entonces el certificado tendrá longitud $O(V_2)$, por lo tanto, es un certificado válido (ya que $O(V_2)$ eso es polinomial en relación con la entrada original, es decir es polinomial en relación con $G_1$ y $G_2$, ya que la entrada que represente a $G_1$ y $G_2$ debe contener todos sus vértices, ya sea explícitamente u obtenerlos con base en las aristas).

Ahora veamos una Máquina de Turing que reciba la gráfica $G_1$, la gráfica $G_2$ y el certificado $u$ y revise si la función dada por el certificado cumple la definición que dimos de isomorfismo, para esto usaremos un algoritmo en el modelo RAM (recordemos que dependiendo de la representación de las gráficas usada, este algoritmo puede cambiar algo, pero no mucho, la idea general se mantiene).

- `Input:` Una gráfica $G_1$, para este caso diremos que la gráfica se va a dar como la matriz de adyacencias de $G_1$, una gráfica $G_2$, para este caso diremos que la gráfica se va a dar como la matriz de adyacencias de $G_2$, y certificado $u$ con una permutación de los vértices de $G_2$, para esto usaremos un arreglo, el cual llamaremos $A$. Y además diremos que todos los vértices son representados como números, esto solo para facilitar la explicación del algoritmo.
- `Output:` 1 si la función dada por el certificado cumple la definición de isomorfismo dada, 0 en otro caso.

1. Lo primero que se va a realizar es revisar la gráfica $G_1$ y $G_2$ tengan la misma cantidad de vértices, para esto usamos un contador (una variable) para cada matriz y la vamos recorriendo aumentando el contador, si las dos matrices tienen el mismo tamaño seguimos el algoritmo, en otro caso regresamos 0 y termina el algoritmo.
2. Ahora vamos a iterar $i$ empezando desde 0 hasta $V_1 - 1$ (el número de vértices en $G_1$, esto se puede pedir en la entrada o en el paso 1 podemos sacar la raíz de la cantidad de casillas que tiene la matriz) y dentro de esta iteración vamos a iterar $j$ empezando desde 0 hasta $V_1 - 1$ (es decir, estamos haciendo un `for` de 0 a $V_1 - 1$ y dentro otro `for` de 0 a $V_1 - 1$).

   Con esto, vamos a revisar la posición $[i][j]$ de la matriz de adyacencias de $G_1$ (es decir, estamos revisando si existe una arista entre él $i$-ésimo vértice de $G_1$ y él $j$-ésimo vértice de $G_1$) y también vamos a revisar la posición $[A[i]][A[j]]$ de la matriz de adyacencias de $G_2$ (es decir, estamos revisando si existe una arista entre el vértice de $G_2$ que se obtiene como resultado de aplicar la función dada por el certificado al $i$-ésimo vértice de $G_1$ y el vértice de $G_2$ que se obtiene como resultado de aplicar la función dada por el certificado al $j$-ésimo vértice de $G_1$).

   Entonces si tenemos el mismo valor en ambas posiciones revisadas (por ejemplo si ambas posiciones tienen el valor 0 o ambas tiene el valor 1) seguimos el algoritmo, pero si obtenemos dos valores diferentes (por ejemplo que la primera posición tenga un 0 y la otra posición tenga un 1) entonces regresamos 0 y termina el algoritmo, en este paso estamos revisando que la arista entre los vértices $v$ y $z$ de $G_1$ esta en $G_1$ sí y solo si la arista entre los vértices $f(v)$ y $f(z)$ de $G_2$ esta en $G_2$.
3. Después de revisar todas las posiciones de la matriz de adyacencia (es decir, todas las aristas presentes y no presentes en $G_1$) regresamos un 1 y termina el algoritmo.

**Complejidad del algoritmo:**

1. **Paso 1:** Solo se recorre toda la matriz de adyacencia de cada gráfica para revisar que tengan el mismo tamaño.

   Entonces para recorrer la matriz de $G_1$ nos toma $O(V_1^2)$ (con $V_1$ la cantidad de vértices en $G_1$) y para la matriz de $G_2$ nos toma $O(V_2^2)$ (con $V_2$ la cantidad de vértices en $G_2$), y también se revisa si dos números son iguales, lo que nos toma $O(1)$ (esto puede cambiar dependiendo del modelo de cómputo utilizado, pero sigue siendo polinomial, y en el modelo RAM nos toma $O(1)$. Además, esto puede cambiar si los objetos son muy complejos, pero suponemos que para representar los nodos usaremos números, por lo tanto, está bien $O(1)$), además podemos regresar valores lo cual nos toma $O(1)$, por lo que todo el paso 1 nos toma $O(V_3^2)$ (con $V_3$ siendo el máximo entre $V_1$ y $V_2$).
2. **Paso 2:** Se itera sobre todos los elementos de la matriz o si lo vemos como dice el paso 2 explícitamente iteramos de 0 a $V_1 - 1$ y dentro de la iteración otra iteración de 0 a $V_1 - 1$, por lo tanto, podemos notar que la iteración externa nos toma $O(V)$ y la interna nos toma $O(V)$, entonces iterar sobre toda la matriz nos toma $O(V^2)$ y también se revisan posiciones en arreglos y matrices lo cual sabemos que toma $O(1)$

   Por último se revisa si dos números son iguales lo cual nos toma $O(1)$ (ya se vio arriba), además podemos regresar valores lo cual nos toma $O(1)$, por lo que todo el paso nos toma $O(V_1^2)$ o $O(V_3^2)$ (para simplificar el análisis un poco).
3. **Paso 3:** Como solo es el regreso de un valor, nos toma $O(1)$.

Por lo tanto, todo el algoritmo nos tomó $O(V_3^2)$, lo cual es una complejidad polinomial, ya que la complejidad es un polinomio.

Por último, usando la tesis de Church-Turing (*complexity-theoretic Church–Turing thesis*), sabemos que existe una Máquina de Turing determinista que resuelve/computa lo mismo que este algoritmo en a lo más una diferencia de tiempo polinomial, por lo tanto, podemos concluir que la Máquina de Turing tendrá tiempo polinomial, entonces concluimos que el problema Iso pertenece a *NP* usando la definición alternativa de *NP*, ya que dimos un certificado de a los más longitud polinomial y una Máquina de Turing determinista que sirve para verificar en tiempo polinomial.

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

**Demostración de pertenencia:**

Este problema pertenece a $P$.

Pero antes de seguir, primero veremos un algoritmo que usaremos para dar una solución a este problema, por lo cual daremos una explicación sencilla de como funciona.

***La Criba de Eratóstenes (Sieve of Eratosthenes)***

Este algoritmo genera los números primos menores o iguales a un número $D$ ($D$ siendo la entrada del algoritmo). Lo primero que hará, será generar un arreglo de tamaño $D - 2$ y escribirá los números del 2 a $D$ en el arreglo y después, marcará con una palomita al número 2 (o se puede marcar de la forma que se prefiera).
A continuación, se recorrerá el arreglo marcando con un tache a todos los múltiplos de 2 (se puede marcar de cualquier forma, solo que no sea de la misma forma como se marcó a 2).
Después regresamos al inicio del arreglo y buscamos al siguiente número sin marcar,

en este caso será el 3, el cual se marca con una palomita y se recorre el arreglo marcando con un tache a los múltiplos de 3 y así sigue hasta que no queden números sin marcar. De esta manera, en cada iteración se regresa al inicio, se marca con una palomita, al siguiente número sin marca y a los múltiplos de este número se tachan con un tache. Así hasta que no existan números sin marcar. Luego de tachar todos los números, tendremos que los números con palomitas son los primos.

Notemos que la complejidad de este algoritmo es $O(D\ log\ log\ D)$,[1] que es una complejidad polinomial.

Ahora veamos una Máquina de Turing que pueda resolver este problema, para esto usaremos un algoritmo en el modelo RAM (recordemos que dependiendo de la representación de las gráficas usada, este algoritmo puede cambiar algo, pero no mucho, la idea general se mantiene y la complejidad seguirá siendo polinomial).

- `Input:` Un natural $n$ y entero positivo $K$.
- `Output:` 1 si $n$ tiene un factor primo menor o igual a $K$, 0 en otro caso.

1. Lo primero que se va a realizar es usar el algoritmo de la Criba de Eratóstenes para generar una lista con todos los primos menores o iguales a $K$, llamemos a esta lista como $E$.
2. Ahora iteramos sobre los números de la lista $E$. Para cada número $e$ de la lista $E$ realizaremos lo siguiente:
   - Revisamos sí $n \,\%\, e \equiv 0$, es decir, revisamos si $n$ es divisible por $e$ sin dejar un residuo. Si esta condición se cumple, entonces regresamos un 1 y termina el algoritmo, pero en otro caso seguimos iterando sobre los demás números en la lista de $E$.
3. Luego de revisar que todos los números de la lista $E$, regresamos un 0 y termina el algoritmo.

**Complejidad del algoritmo:**

1. **Paso 1:** Toma $O(K\ log\ log\ K)$, esto lo notamos cuando explicamos la Criba de Eratóstenes, ya que solo se generan los primos menores o iguales a $K$.
2. **Paso 2:** Notemos que la operación de ver si dos números son el mismo y hacer módulos/divisiones nos toma $O(1)$ por el tipo de modelo de cómputo que estamos usando (pero en otros modelos tomaría algo de complejidad polinomial, por lo cual sigue estando bien), entonces en este paso iteramos sobre los elementos de la lista $E$ que a lo más pueden ser $K$, además podemos regresar valores lo cual nos toma $O(1)$, por lo cual esta iteración nos toma $O(K)$. Entonces todo el paso 2 nos toma $O(K)$.
3. **Paso 3:** Como solo es el regreso de un valor, nos toma $O(1)$.

Por lo tanto, todo el algoritmo nos tomó $O(K\ log\ log\ K)$, lo cual es una complejidad polinomial, ya que la complejidad es un polinomio.

Por último, usando la tesis de Church-Turing (*complexity-theoretic Church–Turing thesis*), sabemos que existe una Máquina de Turing determinista que resuelve/computa lo mismo que este algoritmo en a lo más una diferencia de tiempo polinomial, por lo tanto, podemos concluir que la Máquina de Turing tendrá tiempo polinomial, entonces concluimos que el problema Fact pertenece a $P$ usando la definición de $P$, ya que existe una Máquina de Turing determinista que sirve resolver este problema en tiempo polinomial. Y por lo visto en la clase del 8/10/24, sabemos que $P \subseteq NP$, entonces el problema Conectividad también pertenece a $NP$.

## Ejercicio 2

Sea $\Sigma = \{s_1, s_2, \ldots, s_k\},$ con $k \in N$.

- Demuestra que $\Sigma^*$ es contable.

  Para realizar nuestra demostración, primero recordemos que significa que un conjunto sea contable.

  > ***Contabilidad.*** *Un conjunto es contable si existe una función biyectiva entre los elementos de ese conjunto y los números naturales. Dicho de otra forma, un conjunto es contable si sus elementos se pueden enumerar, es decir, se pueden poner en una lista infinita, donde cada elemento tiene un número o índice asociado.*[2]

  *Demostración.*

  Recordemos que $\Sigma^*$ es el conjunto de todas las cadenas finitas que se pueden formar a partir de los símbolos de $\Sigma$ (incluyendo la cadena vacía $\varepsilon$), por lo que podemos expresar a $\Sigma^*$ como la unión de los conjuntos de palabras de longitud fija, es decir:

  $$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

  Con esto, veamos que para cada $n \geq 0$, el número de palabras en $\Sigma^n$ es $k^n$, ya que hay $k$ opciones para cada una de las $n$ posiciones. En este punto, notemos que $k^n$ es un número finito, por lo que cada conjunto $\Sigma^n$ también será finito, y por ende, contable, ya que podemos ordenar esas $k^n$ palabras de manera lexicográfica (al inicio van las palabras formadas con caracteres con menor numero de posición en $\Sigma$, por ejemplo para ver si el carácter $s_i$ va antes que el carácter $s_j$ tenemos que tomar en cuenta si $i \leq j$) y así cada palabra tendrá una posición que sera el numero que la represente.

  Aquí, veamos que al ser cada $\Sigma^n$ un conjunto finito (contable), entonces $\Sigma^*$ es la unión sobre todos los $n \in \mathbb{N}$, lo cual es una unión contable de conjuntos finitos.[3] Dicho esto, ahora usaremos la siguiente propiedad de conjuntos:

  > *La unión contable de conjuntos finitos es un conjunto contable.*[4]
  > o
  > *La unión de conjuntos finitos o un sistema contable de conjuntos contables es contable*[6]

  Por lo tanto, si tenemos la unión de varios conjuntos finitos contables podemos enumerarlo los elementos del primer conjunto luego a los del segundo conjunto (sumándole a cada número que representa a cada elemento la cantidad de elementos antes del segundo conjunto), luego el tercer conjunto y así por todos los conjuntos, logrando que la unión de todos sea contable.

  $\therefore \Sigma^*$ es contable. □

- ¿El conjunto de máquinas de Turing NO-Deterministas es contable?

  *Demostración.*

  Para comenzar con nuestra demostración, primero recordemos como se describe una MTND (Máquina de Turing No-Determinista).

  Para ello, recordemos que la principal diferencia entre las MTND y las MTD ((Máquina de Turing Determinista)) es que, en las MTND, la función de transición puede tener múltiples transiciones

posibles para un mismo par de estado y símbolo de entrada, permitiendo que la máquina siga diferentes caminos simultáneamente. Sin embargo, a pesar de esta diferencia, una MTND todavía tiene una descripción finita, ya que:

- El conjunto de estados es finito.
- Los alfabetos de entrada y de cinta son finitos.
- Aunque la función de transición es no determinista, sigue siendo finitamente especificable, pues para cada estado y símbolo de entrada hay un número finito de posibles transiciones.
- El estado inicial y los conjuntos de aceptación y rechazo también son finitos.

Por lo tanto, cada MTND también puede ser descrita como una cadena finita sobre algún alfabeto finito.

Esta va a ser de igual manera de la que se mostró en la case de 12/9/24, donde se mostró que cualquier MT puede ser descrita por una cadena finita de 0s y 1s.

Entonces para MTND, seria igual.

- Codificar los estados
  $Q = \{q_0, q_1, q_1, q_2, ...\}$ cada estado se codifica poniendo una cantidad de 0s igual al numero del estado más uno, por ejemplo $q_0$ seria 0, $q_1$ seria 00, $q_2$ seria 000, etc.
- Codificar los tipos de movimientos
  Se codifican así: $-$ sera 0, $\leftarrow$ sera 00 y $\rightarrow$ sera 000
- Codificar el alfabeto
  $\Sigma = \{s_1, s_2, s_3, ...\}$ cada carácter se codifica poniendo una cantidad de 0s igual al numero del carácter, por ejemplo $s_1$ seria 0, $s_2$ seria 00, $s_3$ seria 000, etc. Notemos que $\sqcup$ se codifica como la cadena vacia.
- Codificar la relación de transición
  Cada transición se codifica usando las codificaciones anteriores y separando usando 1s, por ejemplo si tenemos la transición $< (q_0, s_2), (q_1, s_3, \leftarrow) >$ se codifica como 01001001000100, así para todas las transiciones

Y al final la codificación para la MTND sera la codifican de su relación de transición (obteniendo que son un conjunto finito como lo visto en la clase, para esto podemos usar la función $f(x) = 2^{longitud(x)} + y$ con $x$ la cadena que representa a la MTND y $y$ siendo la cadena $x$ como un número natural, o podemos usar la explicación siguiente).

Ahora bien, recordemos que por la definición antes dada, para probar que un conjunto es contable, tendremos que exhibir una función biyectiva entre los elementos de ese conjunto y los números naturales. Para ver que esta función existe en las MTND, veamos que podemos ordenar lexicográficamente sus descripciones, es decir, de forma que primero estén las de longitud 1, luego las de longitud 2, y así sucesivamente. Esto nos permite establecer una función que asigna a cada número natural $i$ una MTND diferente.

Entonces, definamos la siguiente función:

$$f(i) = \text{la } i\text{-ésima MTND en la secuencia ordenada lexicográficamente.}$$

Esta función es una biyección entre los números naturales $\mathbb{N}$ y el conjunto de MTNDs.

$\therefore$ Las MTND son contables. $\square$

- ¿Qué podemos concluir de las afirmaciones de los incisos anteriores?

  Veamos que como cada MTND toma cadenas de entrada de $\Sigma^*$, la entrada que una MTND procesa es una palabra en $\Sigma^*$. Dado que $\Sigma^*$ es contable, esto implica que todas las posibles entradas que una MTND puede procesar son también contables.

  Además, el hecho de que las MTND sean contables, sugiere que la capacidad para describir estas máquinas sigue siendo manejable. Esto implica que, aunque las MTND pueden explorar múltiples caminos computacionales de forma simultánea, su estructura sigue siendo finitamente representable.

  Por otro lado, aunque podemos enumerar todas las posibles máquinas de Turing y entradas ($\Sigma^*$), no existe un algoritmo general que resuelva este problema para todas ellas.

  Esto nos lleva a la idea principal de que la enumerabilidad no implica necesariamente solucionabilidad o decidibilidad, marcando la diferencia entre lo que es conceptualmente manejable (contable) y lo que es computacionalmente alcanzable (computable).

## Ejercicio 3

Considera el siguiente acertijo:

Se tienen 6 personas $P_1, P_2, P_3, P_4, P_5, P_6$ de un club que son elegibles para: Presidente, Vicepresidente y Secretario del Club. Sin embargo, cada uno ha impuesto ciertas restricciones para asumir un cargo:

- $P_1$ no asumirá una carga a menos que $P_5$ sea Presidente.
- $P_2$ no aceptará si supera en rango a $P_3$.
- $P_2$ no trabajará con $P_5$ bajo ninguna condición.
- $P_3$ no quiere trabajar ni con $P_5$ ni con $P_6$.
- $P_3$ no asumirá un cargo si $P_6$ es presidente o si $P_2$ es secretario.
- $P_4$ no trabajará con $P_3$ o $P_5$ a menos que los supere en cargo.
- $P_5$ no quiere ser vicepresidente.
- $P_5$ no quiere ser secretario si $P_4$ asume un cargo.
- $P_5$ no trabajará con $P_1$ a menos que $P_6$ trabaje también.
- $P_6$ no aceptará un cargo a menos que él o $P_3$ sea el presidente.

Para resolver este ejercicio tendremos las siguientes suposiciones:

1. Suponemos que cada persona solo puede tener uno de los siguientes roles: Presidente, Vicepresidente, Secretario o ninguno.
2. Suponemos que solo una persona puede tener el rol de Presidente, Vicepresidente o Secretario.
3. Si una persona no tiene un rol de Presidente, Vicepresidente o Secretario, entonces no tiene un rango para comparar (de esta manera si $P_2$ tiene un cargo y $P_3$ no tiene un cargo, entonces la segunda condición se cumplirá).

¿Cómo podemos elegir a los representantes del club?

Se puede suponer que una persona será el presidente y luego ir acomodando los otros roles para ver si es posible que esa persona sea presidente, de igual manera con las demás personas hasta encontrar una configuración que cumpla las condiciones.

- Expresa el acertijo anterior como un ejemplar concreto de algún problema en *NP*. Enuncia el problema en su versión canónica.

  Lo podemos ver como un ejemplar del problema SAT (*boolean satisfiability problem*).

  **SAT en forma canónica**

  - Ejemplar: Una fórmula booleana $\phi$
  - Pregunta: ¿$\phi$ es satisfacible?

  Notemos que SAT es un problema *NP* (en particular NP-completo), veamos una demostración simple de esto (una demostración más completa es el *Teorema de Cook-Levin* de 1971).

  Para esto tendremos un certificado $u$ que será cada variable de $\phi$ junto con verdadero o falso (es decir, qué valor tomará la variable al evaluar la fórmula). Notemos que a los más hay $n$ ($n$ es la longitud de $\phi$) variables en $\phi$, por lo tanto, podemos notar que como a cada variable le damos un valor, entonces $u$ tendrá longitud $O(n)$ y, por lo tanto, es un certificado válido ($O(n) \in O(n^2)$ y eso es polinomial).

  Ahora veamos una Máquina de Turing que reciba $\phi$ y el certificado y evalué usando los valores del certificado para saber si $\phi$ es satisfacible dado el certificado, para esto usaremos un algoritmo en el modelo RAM (recordemos que dependiendo de la representación de las fórmulas binarias usada, este algoritmo puede cambiar algo, pero no mucho, la idea general se mantiene).

  - `Input:` Fórmula booleana $\phi$ y certificado $u$ con los valores para las variables de $\phi$.
  - `Output:` 1 si $\phi$ se evalúa verdadero usando los valores de $u$, 0 en caso de que se evalúa falso.

  1. Recorremos la fórmula, si encontramos una variable la sustituimos por su valor asignado por el certificado (si la variable es la $y$ entonces la reemplazamos por $u[y]$).
  2. Usamos un método auxiliar para evaluar la fórmula con sus variables reemplazadas, regresamos lo que regrese el método auxiliar.

  Para el método auxiliar usaremos una función recursiva de la siguiente manera (recordemos que una función booleana se puede construir de manera recursiva mediante otras funciones booleanas), tendremos que $\alpha, \beta, \phi$ son funciones booleanas y $x$ es una variable.

  La función $eval(\phi, u)$ recibe una función booleana $\phi$ y el certificado $u$ donde están los valores de las variables (podemos ignorar pasarle $u$ si hacemos el paso 1 del algoritmo anterior), $eval(\phi, u)$ se evaluará de manera recursiva y por casos, de esta manera:

  - $eval(\text{verdadero}, u) = 1$
  - $eval(\text{falso}, u) = 0$
  - $eval(x, u) = u[x]$
  - $eval(\neg\alpha, u) = 1 - eval(\alpha, u)$
  - $eval(\alpha \vee \beta, u) = max(eval(\alpha, u), eval(\beta, u))$
  - $eval(\alpha \wedge \beta, u) = eval(\alpha, u) * eval(\beta, u)$
  - $eval((\phi), u) = eval(\phi, u)$

**Complejidad del algoritmo:**

El paso 1 toma $O(n)$ ya que solo se recorre toda la fórmula una vez $O(n)$, se realizan asignaciones $O(1)$ y lecturas del certificado $O(1)$.

Para el paso 2, notemos que se realiza una llamada de $eval()$ para la fórmula $\phi$ y luego se hace otra llamada para cada variable y operador de $\phi$ por lo tanto se hacen a lo más $n$ llamadas extras y con esto sabemos que el paso 2 toma $O(n)$.

$\therefore$ Todo el algoritmo nos tomó $O(n)$, lo cual es una complejidad polinomial.

Por último, usando la tesis de Church-Turing (*complexity-theoretic Church–Turing thesis*), sabemos que existe una Máquina de Turing determinista que resuelve/computa lo mismo que este algoritmo en a lo más una diferencia de tiempo polinomial, por lo tanto, podemos concluir que la Máquina de Turing tendrá tiempo polinomial, entonces concluimos que SAT es *NP* usando la definición alternativa de *NP* ya que dimos un certificado de a los más longitud polinomial y una Maquina de Turing determinista que sirve para verificar en tiempo polinomial.

Tendremos que las variables de $\phi$ serán:

$P_{1_{pres}}$, $P_{1_{vice}}$, $P_{1_{sec}}$, $P_{1_{nin}}$, $P_{2_{pres}}$, $P_{2_{vice}}$, $P_{2_{sec}}$, $P_{2_{nin}}$, $P_{3_{pres}}$, $P_{3_{vice}}$, $P_{3_{sec}}$, $P_{3_{nin}}$, $P_{4_{pres}}$, $P_{4_{vice}}$, $P_{4_{sec}}$, $P_{4_{nin}}$, $P_{5_{pres}}$, $P_{5_{vice}}$, $P_{5_{sec}}$, $P_{5_{nin}}$, $P_{6_{pres}}$, $P_{6_{vice}}$, $P_{6_{sec}}$ y $P_{6_{nin}}$, donde si $P_{i_{pres}}$ es verdadero significa que la persona $i$ tendrá el rol de presidente, si $P_{i_{vice}}$ es verdadero significa que la persona $i$ tendrá el rol de vicepresidente, si $P_{i_{sec}}$ es verdadero significa que la persona $i$ tendrá el rol de secretario y si $P_{i_{nin}}$ es verdadero significa que la persona $i$ no tendrá ningún rol. Además, $\phi$ será la conjunción ($\wedge$) de todas las fórmulas de las condiciones, y las condiciones son estas:

- Una persona debe tener solo un rol, para esto cada persona tendrá esta fórmula $(P_{i_{pres}} \wedge \neg P_{i_{vice}} \wedge \neg P_{i_{sec}} \wedge \neg P_{i_{nin}}) \vee (\neg P_{i_{pres}} \wedge P_{i_{vice}} \wedge \neg P_{i_{sec}} \wedge \neg P_{i_{nin}}) \vee (\neg P_{i_{pres}} \wedge \neg P_{i_{vice}} \wedge P_{i_{sec}} \wedge \neg P_{i_{nin}}) \vee (\neg P_{i_{pres}} \wedge \neg P_{i_{vice}} \wedge \neg P_{i_{sec}} \wedge P_{i_{nin}})$

  Entonces tendremos la siguiente fórmula:

  $((P_{1_{pres}} \wedge \neg P_{1_{vice}} \wedge \neg P_{1_{sec}} \wedge \neg P_{1_{nin}}) \vee (\neg P_{1_{pres}} \wedge P_{1_{vice}} \wedge \neg P_{1_{sec}} \wedge \neg P_{1_{nin}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{1_{vice}} \wedge P_{1_{sec}} \wedge \neg P_{1_{nin}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{1_{vice}} \wedge \neg P_{1_{sec}} \wedge P_{1_{nin}})) \wedge ((P_{2_{pres}} \wedge \neg P_{2_{vice}} \wedge \neg P_{2_{sec}} \wedge \neg P_{2_{nin}}) \vee (\neg P_{2_{pres}} \wedge P_{2_{vice}} \wedge \neg P_{2_{sec}} \wedge \neg P_{2_{nin}}) \vee (\neg P_{2_{pres}} \wedge \neg P_{2_{vice}} \wedge P_{2_{sec}} \wedge \neg P_{2_{nin}}) \vee (\neg P_{2_{pres}} \wedge \neg P_{2_{vice}} \wedge \neg P_{2_{sec}} \wedge P_{2_{nin}})) \wedge ((P_{3_{pres}} \wedge \neg P_{3_{vice}} \wedge \neg P_{3_{sec}} \wedge \neg P_{3_{nin}}) \vee (\neg P_{3_{pres}} \wedge P_{3_{vice}} \wedge \neg P_{3_{sec}} \wedge \neg P_{3_{nin}}) \vee (\neg P_{3_{pres}} \wedge \neg P_{3_{vice}} \wedge P_{3_{sec}} \wedge \neg P_{3_{nin}}) \vee (\neg P_{3_{pres}} \wedge \neg P_{3_{vice}} \wedge \neg P_{3_{sec}} \wedge P_{3_{nin}})) \wedge ((P_{4_{pres}} \wedge \neg P_{4_{vice}} \wedge \neg P_{4_{sec}} \wedge \neg P_{4_{nin}}) \vee (\neg P_{4_{pres}} \wedge P_{4_{vice}} \wedge \neg P_{4_{sec}} \wedge \neg P_{4_{nin}}) \vee (\neg P_{4_{pres}} \wedge \neg P_{4_{vice}} \wedge P_{4_{sec}} \wedge \neg P_{4_{nin}}) \vee (\neg P_{4_{pres}} \wedge \neg P_{4_{vice}} \wedge \neg P_{4_{sec}} \wedge P_{4_{nin}})) \wedge ((P_{5_{pres}} \wedge \neg P_{5_{vice}} \wedge \neg P_{5_{sec}} \wedge \neg P_{5_{nin}}) \vee (\neg P_{5_{pres}} \wedge P_{5_{vice}} \wedge \neg P_{5_{sec}} \wedge \neg P_{5_{nin}}) \vee (\neg P_{5_{pres}} \wedge \neg P_{5_{vice}} \wedge P_{5_{sec}} \wedge \neg P_{5_{nin}}) \vee (\neg P_{5_{pres}} \wedge \neg P_{5_{vice}} \wedge \neg P_{5_{sec}} \wedge P_{5_{nin}})) \wedge ((P_{6_{pres}} \wedge \neg P_{6_{vice}} \wedge \neg P_{6_{sec}} \wedge \neg P_{6_{nin}}) \vee (\neg P_{6_{pres}} \wedge P_{6_{vice}} \wedge \neg P_{6_{sec}} \wedge \neg P_{6_{nin}}) \vee (\neg P_{6_{pres}} \wedge \neg P_{6_{vice}} \wedge P_{6_{sec}} \wedge \neg P_{6_{nin}}) \vee (\neg P_{6_{pres}} \wedge \neg P_{6_{vice}} \wedge \neg P_{6_{sec}} \wedge P_{6_{nin}}))$

- Solo una persona puede ser presidente, para esto tendremos la siguiente fórmula:

  $(P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge P_{6_{pres}})$

- Solo una persona puede ser vicepresidente, para esto tendremos la siguiente fórmula:

  $(P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge$

$\neg P_{3_{vice}} \wedge P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge P_{6_{vice}})$

- Solo una persona puede ser secretario, para esto tendremos la siguiente fórmula:

  $(P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge P_{6_{sec}})$

- $P_1$ no asumirá un cargo a menos que $P_5$ sea presidente, para esto tendremos la siguiente fórmula:

  $\neg P_{1_{nin}} \Rightarrow P_{5_{pres}}$, que usando equivalencias es $P_{1_{nin}} \vee P_{5_{pres}}$

- $P_2$ no aceptará si supera en rango a $P_3$:

  $(P_{1_{sec}} \Rightarrow (P_{3_{pres}} \vee P_{3_{vice}} \vee P_{3_{nin}})) \wedge (P_{1_{vice}} \Rightarrow (P_{3_{pres}} \vee P_{3_{nin}})) \wedge (P_{1_{pres}} \Rightarrow P_{3_{nin}})$, que usando equivalencias es $(\neg P_{1_{sec}} \vee (P_{3_{pres}} \vee P_{3_{vice}} \vee P_{3_{nin}})) \wedge (\neg P_{1_{vice}} \vee (P_{3_{pres}} \vee P_{3_{nin}})) \wedge (\neg P_{1_{pres}} \vee P_{3_{nin}})$

- $P_2$ no trabajará con $P_5$ bajo ninguna condición, para esto tendremos la siguiente fórmula:

  $\neg P_{2_{nin}} \Rightarrow P_{5_{nin}}$, que usando equivalencias es $P_{2_{nin}} \vee P_{5_{nin}}$

- $P_3$ no quiere trabajar ni con $P_5$ ni con $P_6$, para esto tendremos siguiente la fórmula:

  $\neg P_{3_{nin}} \Rightarrow (P_{5_{nin}} \wedge P_{6_{nin}})$, que usando equivalencias es $P_{3_{nin}} \vee (P_{5_{nin}} \wedge P_{6_{nin}})$

- $P_3$ no asumirá un cargo si $P_6$ es presidente o si $P_2$ es secretario, para esto tendremos la siguiente fórmula:

  $\neg P_{3_{nin}} \Rightarrow (\neg P_{6_{pres}} \wedge \neg P_{2_{sec}})$, que usando equivalencias es $P_{3_{nin}} \vee (\neg P_{6_{pres}} \wedge \neg P_{2_{sec}})$

- $P_4$ no trabajará con $P_3$ o $P_5$ a menos que los supere en cargo, para esto tendremos la fórmula:

  $(P_{4_{sec}} \Rightarrow (P_{3_{nin}} \wedge P_{5_{nin}})) \wedge (P_{4_{vice}} \Rightarrow ((P_{3_{nin}} \vee P_{3_{sec}}) \wedge (P_{5_{nin}} \vee P_{5_{sec}}))) \wedge (P_{4_{pres}} \Rightarrow ((P_{3_{nin}} \vee P_{3_{sec}} \vee P_{3_{vice}}) \wedge (P_{5_{nin}} \vee P_{5_{sec}} \vee P_{5_{vice}})))$, que usando equivalencias es $(\neg P_{4_{sec}} \vee (P_{3_{nin}} \wedge P_{5_{nin}})) \wedge (\neg P_{4_{vice}} \vee ((P_{3_{nin}} \vee P_{3_{sec}}) \wedge (P_{5_{nin}} \vee P_{5_{sec}}))) \wedge (\neg P_{4_{pres}} \vee ((P_{3_{nin}} \vee P_{3_{sec}} \vee P_{3_{vice}}) \wedge (P_{5_{nin}} \vee P_{5_{sec}} \vee P_{5_{vice}})))$

- $P_5$ no quiere ser vicepresidente, para esto tendremos la siguiente fórmula:

  $\neg P_{5_{vice}}$

- $P_5$ no quiere ser secretario si $P_4$ asume un cargo, para esto tendremos la siguiente fórmula:

  $\neg P_{4_{nin}} \Rightarrow \neg P_{5_{sec}}$, que usando equivalencias es $P_{4_{nin}} \vee \neg P_{5_{sec}}$

- $P_5$ no quiere trabajar con $P_1$ a menos que $P_6$ trabaje también, para esto tendremos la siguiente fórmula:

  $(\neg P_{5_{nin}} \wedge \neg P_{1_{nin}}) \Rightarrow \neg P_{6_{nin}}$, que usando equivalencias es $\neg(\neg P_{5_{nin}} \wedge \neg P_{1_{nin}}) \vee \neg P_{6_{nin}}$

- $P_6$ no aceptará un cargo a menos que él o $P_3$ sea presidente, para esto tendremos la siguiente fórmula:

  $\neg P_{6_{nin}} \Rightarrow (P_{6_{pres}} \vee P_{3_{pres}})$, que usando equivalencias es $P_{6_{nin}} \vee (P_{6_{pres}} \vee P_{3_{pres}})$

Por lo tanto, $\phi$ es:

$$(((P_{1_{pres}} \wedge \neg P_{1_{vice}} \wedge \neg P_{1_{sec}} \wedge \neg P_{1_{nin}}) \vee (\neg P_{1_{pres}} \wedge P_{1_{vice}} \wedge \neg P_{1_{sec}} \wedge \neg P_{1_{nin}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{1_{vice}} \wedge P_{1_{sec}} \wedge \neg P_{1_{nin}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{1_{vice}} \wedge \neg P_{1_{sec}} \wedge P_{1_{nin}})) \wedge ((P_{2_{pres}} \wedge \neg P_{2_{vice}} \wedge \neg P_{2_{sec}} \wedge \neg P_{2_{nin}}) \vee (\neg P_{2_{pres}} \wedge P_{2_{vice}} \wedge \neg P_{2_{sec}} \wedge \neg P_{2_{nin}}) \vee (\neg P_{2_{pres}} \wedge \neg P_{2_{vice}} \wedge P_{2_{sec}} \wedge \neg P_{2_{nin}}) \vee (\neg P_{2_{pres}} \wedge \neg P_{2_{vice}} \wedge \neg P_{2_{sec}} \wedge P_{2_{nin}})) \wedge ((P_{3_{pres}} \wedge \neg P_{3_{vice}} \wedge \neg P_{3_{sec}} \wedge \neg P_{3_{nin}}) \vee (\neg P_{3_{pres}} \wedge P_{3_{vice}} \wedge \neg P_{3_{sec}} \wedge \neg P_{3_{nin}}) \vee (\neg P_{3_{pres}} \wedge \neg P_{3_{vice}} \wedge P_{3_{sec}} \wedge \neg P_{3_{nin}}) \vee (\neg P_{3_{pres}} \wedge \neg P_{3_{vice}} \wedge \neg P_{3_{sec}} \wedge P_{3_{nin}})) \wedge ((P_{4_{pres}} \wedge \neg P_{4_{vice}} \wedge \neg P_{4_{sec}} \wedge \neg P_{4_{nin}}) \vee (\neg P_{4_{pres}} \wedge P_{4_{vice}} \wedge \neg P_{4_{sec}} \wedge \neg P_{4_{nin}}) \vee (\neg P_{4_{pres}} \wedge \neg P_{4_{vice}} \wedge P_{4_{sec}} \wedge \neg P_{4_{nin}}) \vee (\neg P_{4_{pres}} \wedge \neg P_{4_{vice}} \wedge \neg P_{4_{sec}} \wedge P_{4_{nin}})) \wedge ((P_{5_{pres}} \wedge \neg P_{5_{vice}} \wedge \neg P_{5_{sec}} \wedge \neg P_{5_{nin}}) \vee (\neg P_{5_{pres}} \wedge P_{5_{vice}} \wedge \neg P_{5_{sec}} \wedge \neg P_{5_{nin}}) \vee (\neg P_{5_{pres}} \wedge \neg P_{5_{vice}} \wedge P_{5_{sec}} \wedge \neg P_{5_{nin}}) \vee (\neg P_{5_{pres}} \wedge \neg P_{5_{vice}} \wedge \neg P_{5_{sec}} \wedge P_{5_{nin}})) \wedge ((P_{6_{pres}} \wedge \neg P_{6_{vice}} \wedge \neg P_{6_{sec}} \wedge \neg P_{6_{nin}}) \vee (\neg P_{6_{pres}} \wedge P_{6_{vice}} \wedge \neg P_{6_{sec}} \wedge \neg P_{6_{nin}}) \vee (\neg P_{6_{pres}} \wedge \neg P_{6_{vice}} \wedge P_{6_{sec}} \wedge \neg P_{6_{nin}}) \vee (\neg P_{6_{pres}} \wedge \neg P_{6_{vice}} \wedge \neg P_{6_{sec}} \wedge P_{6_{nin}}))) \wedge ((P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge P_{5_{pres}} \wedge \neg P_{6_{pres}}) \vee (\neg P_{1_{pres}} \wedge \neg P_{2_{pres}} \wedge \neg P_{3_{pres}} \wedge \neg P_{4_{pres}} \wedge \neg P_{5_{pres}} \wedge P_{6_{pres}})) \wedge ((P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge P_{5_{vice}} \wedge \neg P_{6_{vice}}) \vee (\neg P_{1_{vice}} \wedge \neg P_{2_{vice}} \wedge \neg P_{3_{vice}} \wedge \neg P_{4_{vice}} \wedge \neg P_{5_{vice}} \wedge P_{6_{vice}})) \wedge ((P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge P_{5_{sec}} \wedge \neg P_{6_{sec}}) \vee (\neg P_{1_{sec}} \wedge \neg P_{2_{sec}} \wedge \neg P_{3_{sec}} \wedge \neg P_{4_{sec}} \wedge \neg P_{5_{sec}} \wedge P_{6_{sec}})) \wedge (P_{1_{nin}} \vee P_{5_{pres}}) \wedge ((\neg P_{1_{sec}} \vee (P_{3_{pres}} \vee P_{3_{vice}} \vee P_{3_{nin}})) \wedge (\neg P_{1_{vice}} \vee (P_{3_{pres}} \vee P_{3_{nin}})) \wedge (\neg P_{1_{pres}} \vee P_{3_{nin}})) \wedge (P_{2_{nin}} \vee P_{5_{nin}}) \wedge (P_{3_{nin}} \vee (P_{5_{nin}} \wedge P_{6_{nin}})) \wedge (P_{3_{nin}} \vee (\neg P_{6_{pres}} \wedge \neg P_{2_{sec}})) \wedge ((\neg P_{4_{sec}} \vee (P_{3_{nin}} \wedge P_{5_{nin}})) \wedge (\neg P_{4_{vice}} \vee ((P_{3_{nin}} \vee P_{3_{sec}}) \wedge (P_{5_{nin}} \vee P_{5_{sec}}))) \wedge (\neg P_{4_{pres}} \vee ((P_{3_{nin}} \vee P_{3_{sec}} \vee P_{3_{vice}}) \wedge (P_{5_{nin}} \vee P_{5_{sec}} \vee P_{5_{vice}})))) \wedge (\neg P_{5_{vice}}) \wedge (P_{4_{nin}} \vee \neg P_{5_{sec}}) \wedge (\neg(\neg P_{5_{nin}} \wedge \neg P_{1_{nin}}) \vee \neg P_{6_{nin}}) \wedge (P_{6_{nin}} \vee (P_{6_{pres}} \vee P_{3_{pres}}))$$

- Da una representación visual (diagrama, figura, imagen, etc.) del ejemplar concreto, de acuerdo al problema del inciso anterior.

  La representación visual será $\phi$ (la formula toda gigante que se mostró arriba).

- Resuelve el acertijo exhibiendo un certificado.

  El certificado será el siguiente:

  $P_{1_{pres}}$ = Falso, $P_{1_{vice}}$ = Falso, $P_{1_{sec}}$ = Falso, $P_{1_{nin}}$ = Verdadero, $P_{2_{pres}}$ = Falso, $P_{2_{vice}}$ = Falso, $P_{2_{sec}}$ = Verdadero, $P_{2_{nin}}$ = Falso, $P_{3_{pres}}$ = Falso, $P_{3_{vice}}$ = Falso, $P_{3_{sec}}$ = Falso, $P_{3_{nin}}$ = Verdadero, $P_{4_{pres}}$ = Falso, $P_{4_{vice}}$ = Verdadero, $P_{4_{sec}}$ = Falso, $P_{4_{nin}}$ = Falso, $P_{5_{pres}}$ = Falso, $P_{5_{vice}}$ = Falso, $P_{5_{sec}}$ = Falso, $P_{5_{nin}}$ = Verdadero, $P_{6_{pres}}$ = Verdadero, $P_{6_{vice}}$ = Falso, $P_{6_{sec}}$ = Falso y $P_{6_{nin}}$ = Falso

  Es decir, $P_1$ no tendrá un cargo, $P_2$ será secretario, $P_3$ no tendrá un cargo, $P_4$ será vicepresidente, $P_5$ no tendrá un cargo y $P_6$ será presidente. Con esa asignación se cumplen las 10 condiciones del acertijo.

  Pero notemos algo, si la suposición de "Si una persona no tiene un rol de Presidente, Vicepresidente o Secretario entonces no tiene un rango para comparar (de esta manera si $P_2$ tiene un cargo y $P_3$ no tiene un cargo, entonces la segunda condición se cumplirá)"no se tomara en cuanta significaría que "Si una persona no tiene un rol de Presidente, Vicepresidente o Secretario entonces tiene un rango 0 para comparar, mientras que Presidente tiene rango 3, Vicepresidente rango 2 y Secretario rango 1", y en este caso tendríamos que no existe una solución para este acertijo, esto debido a que ninguna persona puede ser presidente.

  - $P_1$ no puede ser presidente, ya que solo trabaja si $P_5$ es presidente (condición 1)
  - $P_2$ no puede ser presidente, ya que no puede tener rango más alto que $P_3$ (condición 2)

- $P_3$ no puede ser presidente, ya que $P_5$ y $P_6$ no trabajan (condición 4), entonces $P_1$ tampoco trabaja (condición 1) y $P_4$ tampoco trabaja, ya que lo superan en rango (condición 6), entonces no hay suficientes personas
- $P_4$ no puede ser presidente, ya que $P_1$ no trabaja (condición 1), $P_6$ no trabaja (condición 10), $P_5$ no trabaja (condición 7 y 8) y $P_3$ no trabaja (condición 5 y 2) y $P_2$ tampoco trabaja (condición 2 y 5)
- $P_5$ no puede ser presidente, ya que $P_6$ no trabaja (condición 10), $P_1$ tampoco trabaja (condición 1, condición 10 y condición 9), $P_4$ no trabaja (condición 6) y $P_3$ no trabaja (condición 4) y $P_2$ no trabaja (condición 4 y 2)
- $P_6$ no puede ser presidente, ya que $P_1$ no trabaja (condición 1), $P_3$ no trabaja (condición 4), $P_2$ no trabaja (condición 4 y 2) y $P_5$ no trabaja (condición 7 y condición 8), entonces no hay suficientes personas

Por lo tanto, tomamos en cuanta esa suposición para así poder encontrar una solución al acertijo.

## Ejercicio 4

Prueba que la colección de lenguajes decidibles es cerrada bajo las siguientes operaciones:

a. unión

*Demostración.*

Para probar esto, tendremos que mostrar que la unión de dos lenguajes decidibles también es un lenguaje decidible.

Sean $L_1$ y $L_2$ dos lenguajes decidibles. Por definición, esto significa que existen máquinas de Turing $M_1$ y $M_1$ que deciden $L_1$ y $L_2$, respectivamente, es decir:

- La máquina $M_1$ acepta una entrada $w$ si $w \in L_1$ y la rechaza si $w \notin L_1$.
- La máquina $M_2$ acepta una entrada $w$ si $w \in L_2$ y la rechaza si $w \notin L_2$.

Por lo tanto, tendremos que construir una máquina de Turing $M$ que decida $L_1 \cup L_2$, es decir, que acepte una entrada $w$ si $w \in L_1 \cup L_2$, y la rechace si $w \notin L_1 \cup L_2$. De esta forma, tendremos la siguiente Máquina de Turing:

*a)* La máquina $M$ toma una entrada $w$.

*b)* Ejecuta la máquina $M_1$ en la entrada $w$.

- Si $M_1$ acepta $w$, entonces $w \in L_1$ y, por lo tanto, $M$ acepta $w$ y se detiene.

*c)* Si $M_1$ rechaza $w$, entonces ejecuta la máquina $M_2$ en la entrada $w$.

- Si $M_2$ acepta $w$, entonces $w \in L_2$ y, por lo tanto, $M$ acepta $w$ y se detiene.
- Si $M_2$ también rechaza $w$, entonces $w \notin L_1 \cup L_2$, por lo tanto, $M$ rechaza $w$.

Así, la máquina $M$ decide correctamente $L_1 \cup L_2$, ya que siempre se detiene y determina si $w$ pertenece a $L_1 \cup L_2$ y como $M_1$ y $M_2$ son máquinas de Turing que siempre se detienen, la máquina $M$ también siempre se detiene.

$\therefore L_1 \cup L_2$ es decidible.

$\therefore$ los lenguajes decididles están cerrados bajo la unión. □

b. intersección

*Demostración.*

Para probar esto, tendremos que mostrar que la intersección de dos lenguajes decidibles también es un lenguaje decidible.

Sean $L_1$ y $L_2$ dos lenguajes decidibles. Por definición, esto significa que existen máquinas de Turing $M_1$ y $M_2$ que deciden $L_1$ y $L_2$, respectivamente, es decir:

- La máquina $M_1$ acepta una entrada $w$ si $w \in L_1$ y la rechaza si $w \notin L_1$.
- La máquina $M_2$ acepta una entrada $w$ si $w \in L_2$ y la rechaza si $w \notin L_2$.

Por lo tanto, tendremos que construir una máquina de Turing $M$ que decida $L_1 \cap L_2$, es decir, que acepte una entrada $w$ si $w \in L_1 \cap L_2$, y la rechace si $w \notin L_1 \cap L_2$. De esta forma, tendremos la siguiente Máquina de Turing:

*a*) La máquina $M$ toma una entrada $w$.

*b*) Ejecuta la máquina $M_1$ en la entrada $w$.
- Si $M_1$ rechaza $w$, entonces $w \notin L_1$, por lo tanto, $M$ rechaza $w$ y se detiene.

*c*) Si $M_1$ acepta $w$, entonces ejecuta la máquina $M_2$ en la entrada $w$.
- Si $M_2$ acepta $w$, entonces $w \in L_2$, por lo tanto, $w \in L_1 \cap L_2$, y $M$ acepta $w$ y se detiene.
- Si $M_2$ rechaza $w$, entonces $w \notin L_2$, por lo tanto, $w \notin L_1 \cap L_2$, y $M$ rechaza $w$.

Así, la máquina $M$ decide correctamente $L_1 \cap L_2$, ya que siempre se detiene y determina si $w$ pertenece a $L_1 \cap L_2$ y como $M_1$ y $M_2$ son máquinas de Turing que siempre se detienen, la máquina $M$ también siempre se detiene.

$\therefore L_1 \cap L_2$ es decidible.

$\therefore$ los lenguajes decididles están cerrados bajo la intersección. □

c. concatenación

*Demostración.*

Para probar esto, tendremos que mostrar que la concatenación de dos lenguajes decidibles también es un lenguaje decidible.

Sean $L_1$ y $L_2$ dos lenguajes decidibles. Por definición, esto significa que existen máquinas de Turing $M_1$ y $M_2$ que deciden $L_1$ y $L_2$, respectivamente, es decir:

- La máquina $M_1$ acepta una entrada $w$ si $w \in L_1$ y la rechaza si $w \notin L_1$.
- La máquina $M_2$ acepta una entrada $w$ si $w \in L_2$ y la rechaza si $w \notin L_2$.

Por lo tanto, tendremos que construir una máquina de Turing $M$ que decida el lenguaje $L_1 \cdot L_2$, es decir, la concatenación de $L_1$ y $L_2$. Esto significa que $w \in L_1 \cdot L_2$ si existe una forma de dividir $w$ en dos partes, $w = w_1 w_2$, tales que $w_1 \in L_1$ y $w_2 \in L_2$. De esta forma, tendremos la siguiente Máquina de Turing:

*a*) La máquina $M$ toma una entrada $w$.

*b*) Para cada posible forma de dividir $w$ en dos partes, es decir, para cada $w_1$ de longitud $i$ y $w_2$ tal que $w = w_1\, w_2$ (donde $0 \leq i \leq l$ tal que $l$ es la longitud de $w$):

- Ejecuta la máquina $M_1$ en $w_1$.
- Si $M_1$ acepta $w_1$, entonces ejecuta la máquina $M_2$ en $w_2$.
- Si $M_2$ acepta $w_2$, entonces $w \in L_1 \cdot L_2$, por lo tanto, $M$ acepta $w$ y se detiene.

*c*) Si después de probar todas las divisiones posibles de $w$ ninguna combinación de $w_1$ y $w_2$ es aceptada por $M_1$ y $M_2$, entonces $w \notin L_1 \cdot L_2$, y $M$ rechaza $w$.

Así, la máquina $M$ decide correctamente $L_1 \cdot L_2$, ya que siempre prueba todas las divisiones posibles de $w$ y se detiene cuando encuentra una que cumpla $w_1 \in L_1$ y $w_2 \in L_2$ y como $M_1$ y $M_2$ son máquinas de Turing que siempre se detienen, la máquina $M$ también siempre se detiene

$\therefore L_1 \cdot L_2$ es decidible.

$\therefore$ los lenguajes decididles están cerrados bajo la concatenación. □

d. complemento

*Demostración.*

Para probar esto, tendremos que mostrar que el complemento de un lenguaje decidible también es un lenguaje decidible.

Sea $L$ un lenguaje decidible. Por definición, esto significa que existe una máquina de Turing $M$ que decide $L$, es decir:

- La máquina $M$ acepta una entrada $w$ si $w \in L$, y la rechaza si $w \notin L$.

Por lo tanto, tendremos que construir una máquina de Turing $M'$ que decida el complemento de $L$, denotado como $\overline{L}$, es decir, que acepte una entrada $w$ si $w \notin L$ y la rechace si $w \in L$. De esta forma, tendremos la siguiente Máquina de Turing:

*a*) La máquina $M'$ toma una entrada $w$.

*b*) Ejecuta la máquina $M$ en la entrada $w$.

- Si $M$ acepta $w$, entonces $w \in L$, por lo tanto, $M'$ rechaza $w$.
- Si $M$ rechaza $w$, entonces $w \notin L$, por lo tanto, $M'$ acepta $w$.

Así, la máquina $M'$ decide correctamente el complemento de $L$, ya que siempre se detiene y da la respuesta correcta sobre si $w$ pertenece o no a $L$ y como $M$ es una máquina de Turing que siempre se detiene, la máquina $M'$ también siempre se detiene.

$\therefore \overline{L}$ es decidible.

$\therefore$ los lenguajes decididles están cerrados bajo el complemento. □

¿Para qué operaciones la cerradura se mantiene para las clases $P$, $NP$?

Para esto nos usamos estas fuentes de referencias (para ver si se cumple o no) fuente 1, fuente 2 y fuente 3 (no logramos encontrar libro donde se explique claramente, por lo general lo dejan como ejercicios, por ejemplo el problema 7.4.5, pág 155 del libro Computacional Complexity Primera Edición de Christos H. Papadimitriou)

- Para $P$ tenemos lo siguiente

- Unión
  Sean $L_1$ y $L_2$ dos lenguajes en $P$ cualesquiera, por definición de $P$ sabemos que existe una maquina de Turing determinista $M_1$ que decide a $L_1$ en tiempo polinomial $p_1$ y una maquina de Turing determinista $M_2$ que decide a $L_12$ en tiempo polinomial $p_2$, entonces sea $M_3$ una maquina de Turing determinista que dada una entrada $w$ va a realizar lo siguiente.

  1. Ejecuta/simula a $M_1$ con la entrada $w$, si $M_1$ acepta la entrada $w$ entonces $M_3$ acepta a la entrada $w$, pero en caso que $M_1$ no acepte a la entrada $w$ entonces seguimos con el paso siguiente.
  2. Ejecuta/simula a $M_2$ con la entrada $w$, si $M_2$ acepta la entrada $w$ entonces $M_3$ acepta a la entrada $w$, pero en caso que $M_2$ no acepte a la entrada $w$ entonces seguimos con el paso siguiente.
  3. La maquina $M_3$ rechaza a la entrada $w$

  Por lo tanto la maquina $M_3$ va a decidir a la unión de los lenguajes $L_1$ y $L_2$, ya que solo acepta a una cadena si $M_1$ o $M_2$ la aceptan. Ahora notemos la complejidad de $M_3$, el paso 1 sabemos que toma tiempo polinomial, ya que solo ejecuta a $M_1$ (y sabemos que $M_1$ tarda tiempo polinomial), el paso 2 también toma tiempo polinomial, ya que solo ejecuta a $M_2$ (y sabemos que $M_2$ tarda tiempo polinomial), el paso 3 nos toma tiempo constante, ya que solo rechazamos, por lo tanto como sabemos que la suma de dos polinomios es otro polinomio, concluimos que $M_3$ tarda tiempo polinomial. Por lo tanto usando la definición de $P$ tenemos que la unión de los lenguajes $L_1$ y $L_2$ esta en $P$, y concluimos que los lenguajes en $P$ están cerrados bajo la unión.

- Intersección
  Sean $L_1$ y $L_2$ dos lenguajes en $P$ cualesquiera, por definición de $P$ sabemos que existe una maquina de Turing determinista $M_1$ que decide a $L_1$ en tiempo polinomial $p_1$ y una maquina de Turing determinista $M_2$ que decide a $L_2$ en tiempo polinomial $p_2$, entonces sea $M_3$ una maquina de Turing determinista que dada una entrada $w$ va a realizar lo siguiente.

  1. Ejecuta/simula a $M_1$ con la entrada $w$, si $M_1$ acepta la entrada $w$ entonces seguimos con el paso siguiente, pero en caso que $M_1$ no acepte a la entrada $w$ entonces $M_3$ rechaza a la entrada $w$.
  2. Ejecuta/simula a $M_2$ con la entrada $w$, si $M_2$ acepta la entrada $w$ entonces entonces seguimos con el paso siguiente, pero en caso que $M_2$ no acepte a la entrada $w$ entonces $M_3$ rechaza a la entrada $w$.
  3. La maquina $M_3$ acepta a la entrada $w$

  Por lo tanto la maquina $M_3$ va a decidir a la intersección de los lenguajes $L_1$ y $L_2$, ya que solo acepta a una cadena si $M_1$ y $M_2$ la aceptan. Ahora notemos la complejidad de $M_3$, el paso 1 sabemos que toma tiempo polinomial, ya que solo ejecuta a $M_1$ (y sabemos que $M_1$ tarda tiempo polinomial), el paso 2 también toma tiempo polinomial, ya que solo ejecuta a $M_2$ (y sabemos que $M_2$ tarda tiempo polinomial), el paso 3 nos toma tiempo constante, ya que solo acepta, por lo tanto como sabemos que la suma de dos polinomios es otro polinomio, concluimos que $M_3$ tarda tiempo polinomial. Por lo tanto usando la definición de $P$ tenemos que la intersección de los lenguajes $L_1$ y $L_2$ esta en $P$, y concluimos que los lenguajes en $P$ están cerrados bajo la intersección.

- Concatenación
  Sean $L_1$ y $L_2$ dos lenguajes en $P$ cualesquiera, por definición de $P$ sabemos que existe una maquina de Turing determinista $M_1$ que decide a $L_1$ en tiempo polinomial $p_1$ y una maquina de Turing determinista $M_2$ que decide a $L_2$ en tiempo polinomial $p_2$, entonces sea $M_3$ una maquina de Turing determinista que dada una entrada $w$ va a realizar lo siguiente.

1. Vamos a generar todas las posibles formas de dividir en dos partes, la parte $x$ y la parte $y$, a la entrada $w$ (las cuales son $|w|+1$, ya que primero es $x$ vacía y $y$ toda $w$, luego $x$ como el primer carácter de $w$ y $y$ con todos los caracteres de $w$ menos el primero, así seguir hasta que la ultima división sea $x$ toda $w$ y $y$ vacía), entonces realizamos los dos siguientes pasos para cada posible forma de dividir en dos partes, la parte $x$ y la parte $y$, a la entrada $w$, en caso de que todas las posibles divisiones de la entrada ya fueron probadas pasamos al ultimo paso.
2. Ejecuta/simula a $M_1$ con la parte $x$ de la entrada $w$, si $M_1$ acepta la entrada $x$ entonces seguimos con el paso siguiente, pero en caso que $M_1$ no acepte a la entrada $x$ entonces regresamos al primer paso y pasamos a la siguiente posible división de la entrada.
3. Ejecuta/simula a $M_2$ con la parte $y$ de la entrada $w$, si $M_2$ acepta la entrada $y$ entonces $M_3$ acepta a la entrada $w$, pero en caso que $M_2$ no acepte a la entrada $y$ entonces regresamos al primer paso y pasamos a la siguiente posible división de la entrada.
4. La maquina $M_3$ rechaza a la entrada $w$

Por lo tanto la maquina $M_3$ va a decidir a la concatenación de los lenguajes $L_1$ y $L_2$, ya que solo acepta a una cadena si existe una división de la cadena en las partes $x$ y $y$ de tal forma que $M_1$ acepta a $x$ y $M_2$ acepta a $y$. Ahora notemos la complejidad de $M_3$, en el paso 1 generamos todas las posibles divisiones en dos parte de la cadena $w$, lo cual nos toma $O(|w|)$ (ya que existen $|w|+1$ posibles divisiones, lo notamos arriba) por lo tanto los pasos 2 y 3 repiten $O(|w|)$ veces, el paso 2 sabemos que toma tiempo polinomial, ya que solo ejecuta a $M_1$ (y sabemos que $M_1$ tarda tiempo polinomial), el paso 3 también toma tiempo polinomial, ya que solo ejecuta a $M_2$ (y sabemos que $M_2$ tarda tiempo polinomial), el paso 4 nos toma tiempo constante, ya que solo rechaza, por lo tanto como sabemos que la suma de dos polinomios es otro polinomio y que el producto de dos polinomios es otro polinomio es otro polinomio (en particular un polinomio por una constante es un polinomio), concluimos que $M_3$ tarda tiempo polinomial. Por lo tanto usando la definición de $P$ tenemos que la intersección de los lenguajes $L_1$ y $L_2$ esta en $P$, y concluimos que los lenguajes en $P$ están cerrados bajo la intersección.

- Complemento
  Sean $L_1$ un lenguaje en $P$ cualquiera, por defincion de $P$ sabemos que existe una maquina de Turing determinista $M_1$ que decide a $L_1$ en tiempo polinomial $p_1$, entonces sea $M_3$ una maquina de Turing determinista que dada una entrada $w$ va a realizar lo siguiente.

  1. Ejecuta/simula a $M_1$ con la entrada $w$, si $M_1$ acepta la entrada $w$ entonces $M_3$ rechaza a la entrada $w$, pero en caso que $M_1$ no acepte a la entrada $w$ entonces $M_3$ acepta a la entrada $w$.

  Por lo tanto la maquina $M_3$ va a decidir a el complemento del lenguaje $L_1$, ya que solo acepta a una cadena si $M_1$ y $M_2$ la aceptan. Ahora notemos la complejidad de $M_3$, el paso 1 sabemos que toma tiempo polinomial, ya que solo ejecuta a $M_1$ (y sabemos que $M_1$ tarda tiempo polinomial), por lo tanto concluimos que $M_3$ tarda tiempo polinomial. Por lo tanto usando la definición de $P$ tenemos que el complemento de los lenguajes $L_1$ esta en $P$, y concluimos que los lenguajes en $P$ están cerrados bajo el complemento.

- Para $NP$ tenemos lo siguiente

  - Unión
    Sean $L_1$ y $L_2$ dos lenguajes en $NP$ cualesquiera, por definición de $NP$ sabemos que existe una maquina de Turing no determinista $M_1$ que acepta a $L_1$ en tiempo polinomial $p_1$ y una maquina de Turing no determinista $M_2$ que acepta a $L_2$ en tiempo polinomial $p_2$, entonces sea $M_3$ una maquina de Turing no determinista que dada una entrada $w$ va a realizar lo siguiente.

1. Ejecuta/simula a $M_1$ con la entrada $w$, si $M_1$ acepta la entrada $w$ entonces $M_3$ acepta a la entrada $w$, pero en caso que $M_1$ no acepte a la entrada $w$ entonces seguimos con el paso siguiente.
2. Ejecuta/simula a $M_2$ con la entrada $w$, si $M_2$ acepta la entrada $w$ entonces $M_3$ acepta a la entrada $w$, pero en caso que $M_2$ no acepte a la entrada $w$ entonces seguimos con el paso siguiente.
3. La maquina $M_3$ rechaza a la entrada $w$

Por lo tanto la maquina $M_3$ va a decidir a la unión de los lenguajes $L_1$ y $L_2$, ya que solo acepta a una cadena si $M_1$ o $M_2$ la aceptan, por lo tanto si $M_1$ o $M_2$ acepto en alguna rama de su ejecución significa que $M_3$ acepto en alguna rama de su ejecución. Ahora notemos la complejidad de $M_3$, el paso 1 sabemos que toma tiempo polinomial, ya que solo ejecuta a $M_1$ (y sabemos que $M_1$ tarda tiempo polinomial), el paso 2 también toma tiempo polinomial, ya que solo ejecuta a $M_2$ (y sabemos que $M_2$ tarda tiempo polinomial), el paso 3 nos toma tiempo constante, ya que solo rechazamos, por lo tanto como sabemos que la suma de dos polinomios es otro polinomio, concluimos que $M_3$ tarda tiempo polinomial. Por lo tanto usando la definición de *NP* tenemos que la unión de los lenguajes $L_1$ y $L_2$ esta en *NP*, y concluimos que los lenguajes en *NP* están cerrados bajo la unión.

- Intersección
  Sean $L_1$ y $L_2$ dos lenguajes en *NP* cualesquiera, por definición de *NP* sabemos que existe una maquina de Turing no determinista $M_1$ que acepta a $L_1$ en tiempo polinomial $p_1$ y una maquina de Turing no determinista $M_2$ que acepta a $L_2$ en tiempo polinomial $p_2$, entonces sea $M_3$ una maquina de Turing no determinista que dada una entrada $w$ va a realizar lo siguiente.
  1. Ejecuta/simula a $M_1$ con la entrada $w$, si $M_1$ acepta la entrada $w$ entonces seguimos con el paso siguiente, pero en caso que $M_1$ no acepte a la entrada $w$ entonces $M_3$ rechaza a la entrada $w$.
  2. Ejecuta/simula a $M_2$ con la entrada $w$, si $M_2$ acepta la entrada $w$ entonces entonces seguimos con el paso siguiente, pero en caso que $M_2$ no acepte a la entrada $w$ entonces $M_3$ rechaza a la entrada $w$.
  3. La maquina $M_3$ acepta a la entrada $w$

  Por lo tanto la maquina $M_3$ va a decidir a la intersección de los lenguajes $L_1$ y $L_2$, ya que solo acepta a una cadena si $M_1$ y $M_2$ la aceptan, por lo tanto si $M_1$ y $M_2$ aceptaron en alguna rama de su ejecución significa que $M_3$ acepto en alguna rama de su ejecución. Ahora notemos la complejidad de $M_3$, el paso 1 sabemos que toma tiempo polinomial, ya que solo ejecuta a $M_1$ (y sabemos que $M_1$ tarda tiempo polinomial), el paso 2 también toma tiempo polinomial, ya que solo ejecuta a $M_2$ (y sabemos que $M_2$ tarda tiempo polinomial), el paso 3 nos toma tiempo constante, ya que solo acepta, por lo tanto como sabemos que la suma de dos polinomios es otro polinomio, concluimos que $M_3$ tarda tiempo polinomial. Por lo tanto usando la definición de *NP* tenemos que la intersección de los lenguajes $L_1$ y $L_2$ esta en *NP*, y concluimos que los lenguajes en *NP* están cerrados bajo la intersección.

- Concatenación
  Sean $L_1$ y $L_2$ dos lenguajes en *NP* cualesquiera, por definición de *NP* sabemos que existe una maquina de Turing no determinista $M_1$ que acepta a $L_1$ en tiempo polinomial $p_1$ y una maquina de Turing no determinista $M_2$ que acepta a $L_2$ en tiempo polinomial $p_2$, entonces sea $M_3$ una maquina de Turing determinista que dada una entrada $w$ va a realizar lo siguiente.
  1. Vamos a generar de manera no determinista una división de $w$ en dos partes, la parte $x$ y la parte $y$, entonces realizamos los dos siguientes pasos para la parte $x$ y la parte $y$.
  2. Ejecuta/simula a $M_1$ con la parte $x$ de la entrada $w$, si $M_1$ acepta la entrada $x$ entonces seguimos con el paso siguiente, pero en caso que $M_1$ no acepte a la entrada $x$ entonces esta

rama no acepta a la cadena $w$.

3. Ejecuta/simula a $M_2$ con la parte $y$ de la entrada $w$, si $M_2$ acepta la entrada $y$ entonces $M_3$ acepta a la entrada $w$, pero en caso que $M_2$ no acepte a la entrada $y$ entonces esta rama no acepta a la cadena $w$.
4. Si ninguna rama generada de la generación no determinista de la división de $w$ acepta, entonces la maquina $M_3$ rechaza a la entrada $w$

Por lo tanto la maquina $M_3$ va a decidir a la concatenación de los lenguajes $L_1$ y $L_2$, ya que solo acepta si en alguna de las ramas generadas por división de la cadena en las partes $x$ y $y$, se tiene que $M_1$ acepta a $x$ y $M_2$ acepta a $y$, por lo tanto si $M_1$ y $M_2$ aceptaron en alguna rama de su ejecución significa que $M_3$ acepto en esa rama de su ejecución. Ahora notemos la complejidad de $M_3$, en el paso 1 generamos de manera no determinista la división, esto se puede hacer en $O(1)$ ya que solo es elegir en donde partir a $w$, el paso 2 sabemos que toma tiempo polinomial, ya que solo ejecuta a $M_1$ (y sabemos que $M_1$ tarda tiempo polinomial), el paso 3 también toma tiempo polinomial, ya que solo ejecuta a $M_2$ (y sabemos que $M_2$ tarda tiempo polinomial), el paso 4 nos toma tiempo constante, ya que solo rechaza, por lo tanto como sabemos que la suma de dos polinomios es otro polinomio, concluimos que $M_3$ tarda tiempo polinomial. Por lo tanto usando la definición de $NP$ tenemos que la intersección de los lenguajes $L_1$ y $L_2$ esta en $NP$, y concluimos que los lenguajes en $NP$ están cerrados bajo la intersección.

- Complemento
  La ciencia actual no sabe la respuesta de esto.

## Referencias

[1] Computer Sciences Technical Report #909, Department of Computer Sciences University of Wisconsin-Madison, January 2, 1990

[2] GeeksforGeeks. (2024g, mayo 1). Countable set. GeeksforGeeks. https://www.geeksforgeeks.org/countable-set/

[3] Clases de álgebra superior II.

[4] ProofWiki. (s. f.). Countable Union of Finite Sets is Countable. Wikipedia. https://proofwiki.org/wiki/Countable_Union_of_Finite_Sets_is_Countable

[5] set. (2022, March 17). advanced set theory, prove that a countable union of finite set is countable. YouTube. https://youtu.be/599BJPNpPW0?si=oFCF3oFRWFEuadlf

[6] Vladimir Zorich, Matematicheskij Analiz (Part 1,4th corrected edition, Moscow, 2002), pagina 74