NOMBRE: Benjamín Farías Valdés

N.ALUMNO: 17642531

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2213 — Lógica para Ciencias de la Computación — 1' 2021

Tarea 6

1. NP-complete

Para demostrar que CO es NP-complete, hay que demostrar que **CO está en NP** y que **CO es NP-hard.**

1.1. Pertenencia a NP

Para demostrar que el lenguaje CO pertenece a la clase NP, debemos encontrar una **máquina NO determinista** que acepte al lenguaje, y que trabaje en tiempo polinomial. Para lograr esto nos apoyamos en la **tarea 3,** donde se detallaron las siguientes dos máquinas **(ambas se encuentran en el anexo):**

- M: Máquina **determinista** que es capaz de indicar si un conjunto oscuro dado existe dentro de un grafo dado (ambos vienen en el input). Es capaz de lograr esto con una cantidad de cambios de dirección en $O(v)$, donde v es la cantidad de vértices del grafo.
- M': Máquina **NO determinista** que recibe de entrada un número k y un grafo (en la **misma codificación de las palabras w del lenguaje CO**), y se encarga de revisar todos los posibles conjuntos oscuros de dicho tamaño dentro del grafo, para finalmente indicar si existe o no alguno. Es capaz de lograr esto con una cantidad de cambios de dirección en $O(v)$, donde v es la cantidad de vértices del grafo.

La máquina M' es **justamente la que necesitamos,** ya que recibe palabras que codifican un número k y un grafo, y acepta cuando existe un conjunto oscuro de tamaño k en el grafo. Al analizar el tiempo de procesamiento de la máquina M', tenemos lo siguiente:

- La máquina M' sabe a priori el camino para llegar a un conjunto oscuro (de haberlo), por lo que de aquí en adelante se analiza la cantidad de pasos que toma el algoritmo para una rama de ejecución en particular.
- El tamaño de la palabra de entrada depende de la cantidad v de vértices del grafo, ya que por cada vértice se deben agregar v símbolos binarios que representan las conexiones de dicho vértice con el resto (además de los símbolos separadores). Por lo tanto, el tamaño de la palabra de entrada es del orden de $v^2 + c \cdot v + d$, donde $\{c, d\}$ son constantes. Entonces, definimos nuestro tamaño de palabra de entrada como n, y tenemos que n está en $O(v^2)$.

- Cada cambio que realiza la máquina M', en el peor caso, deberá recorrer la palabra de un extremo a otro, es decir tomará $O(n) = O(v^2)$ pasos. Entonces, el total de pasos del algoritmo termina siendo $O(v) \cdot O(v^2) = O(v^3)$, lo que de forma más general es simplemente $O(n^p)$, con p un número entero cualquiera.

Gracias al análisis anterior, sabemos que $T_{M'}(n) \in O(n^p)$, por lo que la máquina M' **acepta el lenguaje y trabaja en tiempo polinomial** con respecto al tamaño n de la palabra de entrada. Queda demostrado que **CO pertenece a la clase de complejidad NP.**

1.2. NP-hard

Para demostrar que el lenguaje CO es NP-hard, debemos encontrar una **reducción** *desde* un lenguaje NP-hard L **conocido** *hacia* CO, la que además **debe poder ser computada en tiempo polinomial.**

El lenguaje L propuesto corresponde al **lenguaje de todos los pares** k, G**, donde** k **es un entero mayor a** 1 **y** G **es un grafo que tiene un clique de tamaño** k**:**

$$L := \{w \mid \exists k > 1, G : w = 1^k \#C(G) \text{ y } G \text{ tiene un clique de tamaño } k\}$$

Este lenguaje L sigue la misma lógica que CO, pero con cliques (conjuntos de vértices que tienen todas las aristas posibles entre sí) en vez de conjuntos oscuros (conjuntos de vértices que **NO** tienen ninguna arista entre sí). Además, en la ayudantía se demostró que es un **lenguaje NP-hard.**

Antes de definir la máquina que computará la reducción, revisaremos el concepto de **complemento de un grafo,** aplicado a este contexto. El complemento de un grafo G corresponde a otro grafo G', cuyo conjunto de aristas **posee a todas las aristas** (u, v) **posibles que NO se encontraban en** G**,** y **NO tiene a ninguna de las aristas de** G**.** En el contexto de los cliques y conjuntos oscuros ocurre algo interesante: **los conjuntos de vértices que forman cliques en un grafo** G**, forman conjuntos oscuros en su complemento** G'**.** Esto es debido a que al sacar el complemento de G se **borran todas sus aristas,** y como los cliques son conjuntos de vértices conectados completamente entre sí, al remover estas aristas quedan efectivamente sin conexiones entre sí, lo que **por definición es un conjunto oscuro.** Más aún, al sacar el complemento de G, **todos sus conjuntos oscuros se transforman en cliques, ya que no existía ninguna arista posible entre ellos, y por tanto ahora deben existir todas esas aristas en** G'**.** Entonces, concluimos que al sacar el complemento de un grafo G, se obtiene otro grafo G', en donde **todos los cliques de** G **ahora son conjuntos oscuros,** y **todos los conjuntos oscuros de** G **ahora son cliques.**

Del análisis anterior sobre el complemento de un grafo, podemos ver que dado un par (k, G) que está en L, sacar el complemento del grafo G que está en dicho par es **justamente la reducción que necesitamos,** ya que G (al estar en L junto con el número k) tiene un clique de tamaño k, el que **será convertido en un conjunto oscuro de tamaño** k **al aplicar la transformación.** Además, el complemento es **recíproco,** es decir, si se saca el complemento de G' se vuelve a obtener G, ya que se vuelven a colocar todas las aristas que se habían removido y se eliminan las que se habían agregado. Esto significa que si existe una palabra $f(w)$ en el conjunto CO, entonces la **palabra** w **original se puede obtener aplicando el complemento del grafo presente en** $f(w)$**.** Además, como en el grafo codificado en $f(w)$ hay un conjunto oscuro de tamaño k, **se deduce que el grafo codificado en** w **tendrá un clique de tamaño** k**, y por lo tanto está en** L (ya que sería el complemento del conjunto oscuro tras aplicar la función). De esta forma, se tiene que al tomar f como la **función que saca el complemento del grafo,** se cumplen las condiciones para que la reducción sea válida:

$$w \in L \longleftrightarrow f(w) \in CO$$

Para aplicar la función f se diseña la siguiente máquina M_C:

- Recibe de entrada una palabra $w = 1^k \# C(G)$, donde el grafo codificado por $C(G)$ tiene un clique de tamaño k.
- Avanza en la palabra hasta encontrar el segundo símbolo #, desde donde **comienzan las secciones que codifican las aristas entre vértices.**
- Avanza **reemplazando** todos los 1s por 0s y todos los 0s por 1s. Esto efectivamente transforma al grafo en su **complemento,** ya que todas las aristas que existían (1s) son borradas, y todas las que no existían (0s) son agregadas.
- Una vez que termina de recorrer la palabra la máquina acepta, devolviendo $f(w)$ según lo descrito anteriormente.

A modo de ejemplo, se tiene la siguiente palabra que representa un grafo de 3 vértices con cliques de tamaño 2, la que se encuentra en L:

$$w = 11\#111\#010\#101\#010$$

En el grafo codificado hay 2 cliques de tamaño 2, uno entre los vértices $(1, 2)$, y el otro entre los vértices $(2, 3)$. Ahora se pasa por la máquina M_C:

$$f(w) = 11\#111\#101\#010\#101$$

Se obtiene el complemento, que posee conjuntos oscuros de tamaño 2 entre los vértices $(1, 2)$ y $(2, 3)$. Es claro que $f(w)$ está en CO. Si se vuelve a pasar por la máquina M_C, se obtiene nuevamente la palabra original w, lo que **evidencia** la doble implicancia demostrada más arriba:

$$f(f(w)) = 11\#111\#010\#101\#010$$

Al analizar el **tiempo de ejecución** de la máquina M_C, podemos notar que simplemente recorre la palabra una vez, de izquierda a derecha, es decir $T_{M_C}(n) \in O(n)$, lo que significa que **la reducción se puede computar en tiempo polinomial.**

Como se encontró una reducción **válida** desde L hacia CO, la que es **computable en tiempo polinomial,** podemos argumentar que el lenguaje CO es al menos tan complejo como L, y por lo tanto es **NP-hard.**

Finalmente, se logró demostrar que el lenguaje **CO está en la clase NP y es NP-hard,** por lo que se concluye que **CO es también NP-complete.**

2. Anexo: Tarea 3

2.1. Máquina Determinista

A continuación se describirá **paso a paso** el funcionamiento de una máquina M que cumple con lo pedido, siguiendo un ejemplo en paralelo para poder ilustrar de forma más clara el algoritmo:

- Para ejemplificar cada paso se utilizará el siguiente grafo cuadrado (mismo del enunciado):

  $$C(G) = 1111\#0101\#1010\#0101\#1010$$

  Y la palabra de entrada será $1010\#C(G) \rightarrow 1010\#1111\#0101\#1010\#0101\#1010$, que representa un **conjunto oscuro entre los nodos** $\{1, 3\}$.

- El **primer paso** consiste en un **parser**, el que se encargará de revisar el input y validarlo, pasando a la siguiente fase. En caso de que no cumpla con el formato pedido, simplemente se rechaza y termina la ejecución de M.

  $$B\hat{1}010\#1111\#0101\#1010\#0101\#1010B$$

  * **El símbolo ˆ representa la posición actual del cabezal, B representa el carácter vacío (blanco)**

- Estando parado en el primer bit del input, se avanza hasta llegar al primer #, y luego se procede a **sobre-escribir todos los símbolos entre este y el siguiente # (incluyéndolos), colocando el símbolo** X **(esto es para separar los bits directamente de las secciones** w_i **del grafo).** Una vez listo este paso, se deja el cabezal parado en el primer bit nuevamente (esto sale directo haciendo uso del **blanco** del extremo izquierdo).

  $$B\hat{1}010XXXXXX0101\#1010\#0101\#1010B$$

- Ahora existen 2 posibilidades, una es que el primer bit sea un 1 y la otra es que sea 0. Por cada una de ellas se ejecuta una **serie de pasos diferentes** a continuación **(denominadas variante de bit** 1 **y variante de bit** 0**).** En el ejemplo el primer bit es un 1, por lo que veremos este caso **primero** (después se verá el otro en el 2do bit).

  $$B\hat{1}010XXXXXX0101\#1010\#0101\#1010B$$

- Como el bit es un 1, se avanzará hasta atravesar todas las X. Posterior a esto, **se irá revisando y reemplazando cada casilla hacia adelante por una** X**, hasta encontrar un** 1 **o un** 0, y en caso de que **antes** de encontrar estos símbolos aparezca el **símbolo de exclamación** !, el algoritmo terminará y la máquina **rechazará inmediatamente.** La razón de esto es que ese símbolo será usado para indicar una conexión a otro nodo previamente revisado, el que además pertenece al conjunto oscuro del input, **contradiciendo** la propiedad de que no pueden existir conexiones entre nodos de un conjunto oscuro. En este caso se llega directamente a un 0, avanzando al siguiente paso dentro de la **variante de bit** 1.

  $$B1010XXXXXX\hat{0}101\#1010\#0101\#1010B$$

- En este paso se avanzará **reemplazando** todo por una X hasta llegar al siguiente # (incluyéndolo) o al blanco del extremo derecho. La razón de esto es que se terminó de revisar el w_i correspondiente al nodo/bit actual.

$$B1010XXXXXXXXXXX\hat{1}010\#0101\#1010B$$

- Ahora se avanza hasta encontrar un 1 o 0, y dependiendo de cuál se encontró se **reemplaza por un símbolo específico.** Si es un 1 se reemplaza por un !, mientras que si es un 0 se reemplaza por un |. Luego de esto se avanza hasta el siguiente #, tras el que se **repite este paso,** y así sucesivamente hasta llegar al **blanco del extremo derecho.** Esto se hace para **marcar la conexión del nodo actual con el resto de nodos** del grafo que aún no han sido revisados (marcar con el ! indica que está conectado con el nodo **asociado a ese** w_i, por lo que sirve para poder rechazar a futuro si es que se visita ese nodo como parte del conjunto oscuro, es decir mediante la variante de bit 1) (marcar con el | simplemente indica que **NO hay conexión entre el nodo actual y el asociado al** w_i).

$$B1010XXXXXXXXXXX!010\#|101\#!010\hat{B}$$

- Ahora nos **devolvemos hasta el primer bit** del input (usando el blanco de la izquierda para ubicarlo), lo eliminamos (colocando un blanco en su lugar) y quedamos parados en el **siguiente bit a revisar, terminando así la variante de bit** 1**.**

$$BB\hat{0}10XXXXXXXXXXX!010\#|101\#!010B$$

- El siguiente bit a revisar es un 0, por lo que a continuación se ejecutarán los pasos de la **variante de bit** 0**.** Primero avanzamos hasta atravesar las X (al igual que en la otra variante).

$$BB010XXXXXXXXXXX\hat{!}010\#|101\#!010B$$

- Ahora, **a diferencia de la otra variante, NO se revisarán los símbolos y simplemente se reemplazarán todos por una** X **hasta llegar al siguiente # (incluyéndolo).** La razón de esto es que el nodo actual no está dentro del conjunto oscuro del input, por lo que **no nos interesa** si está o no conectado con nodos de dicho conjunto.

$$BB010XXXXXXXXXXXXXXX\hat{|}101\#!010B$$

- Ahora se avanza hasta encontrar un 1 o 0, pero a diferencia de la otra variante se coloca un | para reemplazarlo, **independiente de si es un** 1 **o un** 0 **(esto debido a que al ser un nodo externo al conjunto oscuro, no nos interesa esa conexión a futuro).** Luego de esto se avanza hasta el siguiente #, tras el que se **repite este paso,** y así sucesivamente hasta llegar al **blanco del extremo derecho.**

$$BB010XXXXXXXXXXXXXXX||01\#!|10\hat{B}$$

- Nos devolvemos al principio y pasamos al siguiente bit a revisar (al igual que la otra variante), **terminando así la variante de bit** 0**.**

$$BBB\hat{1}0XXXXXXXXXXXXXXX||01\#!|10B$$

- Ahora simplemente se ejecuta la **variante correspondiente a cada bit que falta por revisar, hasta que ya no queden más.**

$$BBB10XXXXXXXXXXXXXXXXXXXX!|!0\hat{B}$$

$$BBBB\hat{0}XXXXXXXXXXXXXXXXXXXX!|!0B$$

$$BBBB0XXXXXXXXXXXXXXXXXXXXXXXX\hat{B}$$

$$BBBBB\hat{X}XXXXXXXXXXXXXXXXXXXXXXXXB$$

- Finalmente, ahora que ya se revisaron todos los bits, se **ACEPTA la palabra de input, ya que significa que NO se encontraron conexiones entre ninguno de los nodos del conjunto oscuro indicado.**

La máquina M ilustrada en los pasos de arriba siempre acepta al terminar de revisar todos los bits, y **rechaza cuando está revisando un nodo del conjunto oscuro (variante de bit** 1**) y se encuentra con un** ! **dentro de su** w_i **correspondiente,** debido a que se encontró entonces una conexión entre este nodo y otro del conjunto oscuro que fue visitado previamente **(ya que los bits** 1 **son los únicos que colocan** ! **en sus conexiones con el resto).** Un ejemplo para ilustrar el rechazo de la máquina es con el mismo grafo de antes y el conjunto oscuro $\{1, 2\}$:

$$B\hat{1}100XXXXXX0101\#1010\#0101\#1010B$$

$$B1100XXXXXXXXXX!010\#|101\#!010\hat{B}$$

$$BB\hat{1}00XXXXXXXXXX!010\#|101\#!010B$$

$$BB100XXXXXXXXXX\hat{!}010\#|101\#!010B$$

En este caso la variante de bit 1 encuentra un ! al revisar su w_i asociado (en este caso al nodo 2), por lo que **rechaza inmediatamente, ya que los nodos** 1 **y** 2 **están conectados y no pueden pertenecer al mismo conjunto oscuro.**

Ahora falta obtener la **complejidad de la cantidad de cambios producidos durante la ejecución de la máquina:**

- Asumimos que el **parser** produce una cantidad de cambios que está en $O(n)$, ya que no tiene sentido que crezca de forma superior a lineal en la cantidad de bits (sólo debe recorrer el input y asegurarse que sea el formato correcto, no es necesario que dé n vueltas por cada bit).
- El segundo paso se encarga de colocar X entre los 2 primeros #, lo que simplemente requiere de ir y luego volver al comienzo, resultando en un número de cambios **constante** (exactamente 2).
- Para el resto del algoritmo, se **realizarán sólo 2 cambios por cada bit a revisar,** ya que las variantes se mueven hasta cada extremo sin hacer cambios de dirección entre medio. Esto implica que la complejidad de los cambios en esta parte es de $O(2 \cdot n) = O(n)$.
- Sumando todas las complejidades anteriores, se tiene que la complejidad de los cambios de dirección en el **algoritmo completo es de** $O(n)$**.**

Finalmente, como el algoritmo **cumple con lo pedido y su complejidad en cambios es** $O(n)$**,** se logró demostrar que existe la máquina M determinista que acepta el lenguaje pedido en el enunciado.

2.2. Máquina NO Determinista

A continuación se describirá **paso a paso** el funcionamiento de una máquina **NO determinista** M' que cumple con lo pedido, siguiendo un ejemplo en paralelo para poder ilustrar de forma más clara el algoritmo:

- Para ejemplificar cada paso se utilizará el siguiente grafo cuadrado (mismo del enunciado):

  $$C(G) = 1111\#0101\#1010\#0101\#1010$$

  Y la palabra de entrada será $11\#C(G) \rightarrow 11\#1111\#0101\#1010\#0101\#1010$, que representa la existencia de un **conjunto oscuro de tamaño 2** en el grafo G.

- El **primer paso** consiste en un **parser**, el que se encargará de revisar el input y validarlo, pasando a la siguiente fase. Dentro de la validación también se **incluye revisar que la cantidad de** 1**s es a lo más** n**,** es decir, que se cumpla con $k \leq n$ (ya que no tiene sentido buscar un conjunto oscuro de un tamaño mayor al del conjunto de todos los nodos del grafo). En caso de que no cumpla con el formato pedido, simplemente se rechaza y termina la ejecución de M'.

  $$B\hat{1}1\#1111\#0101\#1010\#0101\#1010B$$

  * **El símbolo ˆ representa la posición actual del cabezal, B representa el carácter vacío (blanco)**

- Estando parado en el primer 1, se escribe otro 1 a su izquierda, y luego se avanza hacia la derecha y se borra el **último** 1 **antes del primer #.** Esto lo que hace es **desplazar hacia la izquierda los** k 1**s del input,** dejando un espacio en blanco entre ellos y el primer #. Luego se deja el cabezal en el primer 1 después del #, reemplazándolo por una X.

  $$B11B\#\hat{X}111\#0101\#1010\#0101\#1010B$$

- Ahora hay que moverse hacia la izquierda hasta ver el **primer blanco, reemplazándolo por un** 0**.** Inmediatamente después, se borra el símbolo directamente a la izquierda de este nuevo 0, y luego se avanza hasta el extremo izquierdo y se coloca un nuevo 1. Esto vuelve a **desplazar los** k 1**s a la izquierda, y además coloca un** 0 **que será el bit que representa al primer nodo del grafo** (1 si pertenece al conjunto oscuro y 0 en otro caso), el que de momento será simplemente 0. Finalmente se vuelve hasta el próximo 1 después del #, que sería el que **está a la derecha de la** X **colocada anteriormente.**

  $$B11B0\#X\hat{1}11\#0101\#1010\#0101\#1010B$$

- A continuación se **repite el paso anterior** para cada uno de los 1s que quedan entre medio de los primeros dos #, que corresponden a los nodos del grafo.

  $$B11B0000\#XXXX\hat{\#}0101\#1010\#0101\#1010B$$

- Podemos ver que **se formó una palabra de** n 0**s, que será utilizada para la siguiente fase.** Ahora se revierten las X a los 1s que estaban originalmente.

  $$B11B0000\#\hat{1}111\#0101\#1010\#0101\#1010B$$

- En esta fase se utilizará el **NO determinismo** para generar todas las **combinaciones de palabras binarias de largo n que tienen exactamente un** 1**.** Primero se reemplaza el 1 por una X (igual que en la fase anterior), y luego se avanza hacia la izquierda hasta llegar al primer blanco **(el que separa los k** 1**s de la palabra de** 0**s).** Si el símbolo a la izquierda de este blanco es **otro blanco,** simplemente nos devolvemos hacia la sección de los nodos, dejando el cabezal al lado de la X. En cambio, **si el símbolo a la izquierda del blanco es un** 1**, se va hasta el extremo izquierdo y se borra el primer** 1 **presente. A continuación se deberá colocar este** 1 **(que fue borrado) en alguno de los** 0**s de la palabra de tamaño n.** Aquí se definen las siguientes **condiciones NO deterministas:**

  1. Si se ve un 1, **entonces simplemente se sigue avanzando.**
  2. Si se ve un 0, **existen 2 posibles decisiones. Una es colocar aquí el** 1**,** tras lo que se devuelve a la sección de nodos (al lado de la última X). **La otra es ignorar el** 0 y seguir avanzando.
  3. Si se llega hasta el # y **aún no se ha colocado el** 1 **en ninguno de los** 0**s,** se devuelve y se **coloca por default en el primer** 0 **que se encuentre,** posteriormente volviendo a la sección de los nodos.

  La condición 1 **asegura que cada vez que se coloque un** 1 **se aumentará la cantidad de** 1**s en la palabra** (de otra forma se podrían colocar 1s en lugares que ya tenían uno, y no llegar nunca a los k 1s necesarios). La condición 2 es la **responsable de generar todas las combinaciones posibles gracias al NO determinismo,** ya que al considerar todas las ramas de ejecución posibles, en cada posición de la palabra de largo n **existirá una ejecución que colocó su** 1 **allí e ignoró los otros.** La condición 3 **evita que ocurran loops infinitos en caso de que siempre se desee ignorar los** 0**s y no colocar ningún** 1**.** A continuación se muestran los resultados posibles para el ejemplo, omitiendo palabras repetidas (que pueden ocurrir debido a la condición 3):

$$B1B1000\#X\hat{1}11\#0101\#1010\#0101\#1010B$$
$$B1B0100\#X\hat{1}11\#0101\#1010\#0101\#1010B$$
$$B1B0010\#X\hat{1}11\#0101\#1010\#0101\#1010B$$
$$B1B0001\#X\hat{1}11\#0101\#1010\#0101\#1010B$$

- En este paso se **repetirá el procedimiento del paso anterior para cada uno de los nodos restantes (hasta que todos sean reemplazados por X).** Esto colocará los k 1s en la palabra, generando así **todas las posibles combinaciones de exactamente k** 1**s en palabras binarias de largo n (de hecho habrán muchas repetidas, pero eso claramente NO afecta la salida de la máquina).** NO colocará más de k 1s **debido a que se van a acabar los k** 1**s del extremo izquierdo** y entonces en esos casos simplemente se devolverá sin cambiar nada **hasta completar las n repeticiones.** A modo de ejemplo, tomaremos de aquí en adelante la **primera opción de las 4 mostradas en el paso anterior,** llegando a los siguientes posibles resultados:

$$B1100\#XXXX\hat{\#}0101\#1010\#0101\#1010B$$
$$B1010\#XXXX\hat{\#}0101\#1010\#0101\#1010B$$
$$B1001\#XXXX\hat{\#}0101\#1010\#0101\#1010B$$

- En esta fase, **cada rama de ejecución ya tiene una de las posibles combinaciones de k** 1**s en palabras de largo n,** y por lo tanto se vuelven a revertir las X a los 1s originales que representaban a cada nodo. Después nos paramos en el primer bit en el extremo izquierdo. Cada opción anterior queda como sigue:

$$B\hat{1}100\#1111\#0101\#1010\#0101\#1010B$$

$$B\hat{1}010\#1111\#0101\#1010\#0101\#1010B$$

$$B\hat{1}001\#1111\#0101\#1010\#0101\#1010B$$

- Si nos fijamos en el estado de la cinta, tenemos **justamente el formato que recibe la máquina M definida en el ejercicio a) de la tarea.** Entonces, en este punto M' **actúa como una máquina universal sobre la máquina M, entregando el mismo output que esta última entregue y terminando así su ejecución.** Revisando las opciones que teníamos del paso anterior, llegamos a que las opciones 1 y 3 son **rechazadas** (ya que el nodo 1 está conectado tanto al 2 como al 4 en el ejemplo), pero la opción 2 **acepta** (ya que el $\{1, 3\}$ es un conjunto oscuro en el ejemplo). Como **una de las ramas logró aceptar finalmente al input, por definición de las máquinas NO deterministas, tenemos una ejecución válida y por lo tanto la máquina M' aceptará y terminará.**

La máquina M' descrita arriba **explora todas las posibles combinaciones de conjuntos oscuros de tamaño k dado por el input, y si cualquiera de estos es aceptado, entonces aceptará también, ya que descubrió que existe al menos un conjunto oscuro de tamaño k dentro del grafo G.** Si **ninguna** de sus ramas del árbol de ejecución aceptan, entonces rechazará, ya que **no existe ningún conjunto oscuro de tamaño k en el grafo.** Además, todos los pasos son finitos, ya que se basan en la cantidad de nodos n, y por lo tanto nunca se quedará pegada.

Un **ejemplo de rechazo** para la máquina M' sería con el siguiente input (y el mismo grafo del ejemplo anterior):

$$111\#1111\#0101\#1010\#0101\#1010$$

Como el grafo es cuadrado, **cada nodo estará conectado con todos los otros MENOS uno**, por lo que los conjuntos oscuros son de tamaño máximo 2. Con este input se explorarán todas las combinaciones de conjuntos oscuros de 3 nodos, pero todas fallarán, y por lo tanto **será rechazado.**

Ahora falta obtener la **complejidad de la cantidad de cambios producidos durante la ejecución de la máquina:**

- Asumimos que el **parser** produce una cantidad de cambios que está en $O(n)$, ya que no tiene sentido que crezca de forma superior a lineal en la cantidad de nodos (sólo debe recorrer el input y asegurarse que sea el formato correcto, no es necesario que dé n vueltas por cada nodo).
- El segundo paso se encarga de formar la palabra de 0s de largo n, **repitiendo por cada nodo el mismo procedimiento con cambios constantes.** Entonces su complejidad es $O(n)$.
- El tercer paso genera las combinaciones posibles de palabras, para lo que ejecuta el mismo procedimiento con cambios constantes **por cada nodo** (una vez que se acaban los k 1s sigue realizando pasos que no tienen efecto hasta completar n repeticiones). Entonces su complejidad es $O(n)$.
- El resto del algoritmo consiste en **invocar a la máquina M de la parte a) de la tarea, la que ya sabemos que es $O(n)$ en la cantidad de cambios.**
- Como todos los pasos son $O(n)$, se tiene que el **algoritmo completo es $O(n)$.**

Finalmente, como el algoritmo **cumple con lo pedido y su complejidad en cambios es $O(n)$,** se logró construir la máquina M' **NO determinista** que acepta el lenguaje pedido en el enunciado.