Universidad Nacional Autónoma de México
Facultad de Ciencias

Complejidad Computacional

Tarea 5

García Ponce José Camilo - 319210536

Ejercicio 1

- Demuestra que $\leq_p$ es una relación de orden en NP.
  Para esto tenemos que demostrar tres cosas

  1. Reflexividad
     Sea L un lenguaje cualquiera en NP, ahora veamos que $L_1 \leq_p L_1$.
     Para esto diremos que la función de transformación f sera la función identidad.
     Notemos que claramente f es computable en tiempo polinomio, ya que como es la función de identidad no se realizan operaciones, solo se regresa el mismo valor.
     Ahora veamos que $x \in L \implies f(x) \in L$ y $x \notin L \implies f(x) \notin L$
     Supongamos una cadena $x \in L$ cualquiera, ahora notemos que $f(x) = x$, entonces se cumple que $x \in L \implies f(x) \in L$.
     Ahora, supongamos una cadena $x \notin L$ cualquiera, ahora notemos que $f(x) = x$, entonces se cumple que $x \notin L \implies f(x) \notin L$.
     Por lo tanto usando la definición de $\leq_p$, concluimos que $L_1 \leq_p L_1$, entonces $\leq_p$ cumple con reflexividad.

  2. Transitividad
     Sean L_1, L_2 y L_3 lenguajes cualesquiera en NP, tales que $L_1 \leq_p L_2$ y $L_2 \leq_p L_3$, ahora veamos que $L_1 \leq_p L_3$.
     Notemos que por definición de $\leq_p$ existen dos funciones f y g, tales que son computables en tiempo polinomio y que $x \in L_1 \implies f(x) \in L_2$ y $x \notin L_1 \implies f(x) \notin L_2$ y $y \in L_2 \implies g(y) \in L_3$ y $y \notin L_2 \implies g(y) \notin L_3$.
     Para esto diremos que la función de transformación h sera la composición de g y f, $h(x) = g(f(x))$.
     Notemos que claramente h es computable en tiempo polinomio, ya que f y g son computables en tiempo polinomial y h primero realiza f y luego g, lo cual le toma tiempo polinomial, ya que la suma de dos polinomios es otro polinomio.
     Ahora veamos que $z \in L_1 \implies h(z) \in L_3$ y $z \notin L_1 \implies h(z) \notin L_3$
     Supongamos una cadena $z \in L_1$ cualquiera, ahora notemos que $h(z) = g(f(z))$, entonces como $L_1 \leq_p L_2$ tenemos que $f(z) = a$ tal que $a \in L_2$, ahora notemos que como $L_2 \leq_p L_3$ tenemos que $g(a) = b$ tal que $b \in L_3$, por lo tanto $h(z) = g(f(z)) = g(a) = b$, entonces se cumple que $z \in L_1 \implies h(z) \in L_3$.
     Ahora, supongamos una cadena $z \notin L_1$ cualquiera, notemos que $h(z) = g(f(z))$, entonces como $L_1 \leq_p L_2$ tenemos que $f(z) = a$ tal que $a \notin L_2$, ahora notemos que como $L_2 \leq_p L_3$ tenemos que $g(a) = b$ tal que $b \notin L_3$, por lo tanto $h(z) = g(f(z)) = g(a) = b$, entonces se cumple que $z \notin L_1 \implies h(z) \notin L_3$.
     Por lo tanto usando la definición de $\leq_p$, concluimos que $L_1 \leq_p L_3$, entonces $\leq_p$ cumple con

transitividad.

3. Antisimetria
   Sean L_1 y L_2 lenguajes cualesquiera en NP, tales que $L_1 \leq_p L_2$ y $L_2 \leq_p L_1$, ahora veamos que $L_1 \equiv_p L_2$ (aquí suponemos que $\equiv_p$ significa que ambos problemas son equivalentes).
   Notemos que por definición de $\leq_p$ existen dos funciones f y g, tales que son computables en tiempo polinomio y que $x \in L_1 \implies f(x) \in L_2$ y $x \notin L_1 \implies f(x) \notin L_2$ y $y \in L_2 \implies g(y) \in L_1$ y $y \notin L_2 \implies g(y) \notin L_1$.
   Notemos que L_1 y L_2 son equivalentes en dificultad, ya que si tenemos una solución eficiente (en tiempo polinomio) s_1 para resolver el problema del lenguaje L_1 entonces podemos transformar, en tiempo polinomial, un ejemplar del problema del lenguaje L_2 a un ejemplar del problema del lenguaje L_1 con g y así poder usar la solución s_1 al nuevo ejemplar, así logrando resolver el problema del lenguaje L_2 en un tiempo eficiente (tiempo polinomial), de igual manera si existe una solución eficiente (en tiempo polinomio) s_2 para resolver el problema del lenguaje L_2, podemos usar la transformación f para poder resolver el problema del lenguaje L_1 en tiempo polinomial de manera eficiente, todo gracias a las transformaciones que son computables en tiempo polinomial y la suma de dos polinomios es otro polinomio.
   Entonces $\leq_p$ cumple con antisimetria.

Por lo tanto podemos concluir que $\leq_p$ es una relación de orden en NP.

- ¿Como podríamos definir formalmente que dos problemas en NP son equivalentes en dificultad?
  Dos problemas L_1 y L_2 en NP son equivalentes en dificultad si se cumple lo siguiente: $L_1 \leq_p L_2$ y $L_2 \leq_p L_1$, de manera similar a lo visto en la antisimetria de $\leq_p$, ya que ambos problemas se podrían reducir entre ellos y serian equivalentes en dificultad de resolver.

- ¿Utilizando $\leq_p$, es posible establecer relación de orden en P? ¿Se puede establecer algún otro tipo de orden, respecto a la dificultad de problemas, en P?
  Si es posible posible establecer una relación de orden en P, esto ya que como notamos en el primer ejercicio que $\leq_p$ cumple con lo necesario para ser una relación de orden y sabemos que $\leq_p$ esta definido para los lenguajes no solo para los lenguajes de NP, por lo tanto si es una relación de orden en P (se puede usar la misma transformación solo cambiar que los lenguajes estén en P y no NP).
  Es posible establecer otro tipo de orden en P, esto usando tal vez la complejidad temporal de cada lenguaje, y así poder tener los lenguajes que toman tiempo $O(1)$ siendo problemas triviales, después lenguajes que toman tiempo $O(n)$ siendo problemas fáciles pero más complicados que los problemas que toman $O(1)$, luego lenguajes que toman tiempo $O(n^2)$ siendo problemas un poco más difíciles que los que toman $O(n)$, posteriormente lenguajes que toman tiempo $O(n^3)$ siendo problemas un poco más difíciles que los que toman $O(n^2)$, y así seguir, de esta manera usamos su complejidad $O(n^k)$ para poder ordenar los lenguajes mediante el uso de k ($k \in N$) para poder ordenarlos (gracias al orden de los naturales).
  De manera similar podemos establecer otra relación de orden en P, pero ahora usando la complejidad de espacio de cada lenguaje, de manera similar a como se realizo arriba, solo que ahora tendremos que $O(1) \leq O(log(n)) \leq O(n) \leq O(n^c) \leq O(2^n)$, usamos las funciones de las complejidades para comparar los lenguajes.

Ejercicio 2

Considera la demostración del Teorema de Cook vista en clase. Responde las siguientes preguntas justificando ampliamente.

- ¿Es valido cambiar la demostración considerando ventanas de: 3×3?.
  Si es valido, la demostración y la transformación van a seguir sirviendo con ventanas de 3×3.
  Esto debido a que al agregar una fila a la columna solo vamos a obtener más información del tableau, entonces poder ver/obtener más información no afecta a la demostración, ya que no perdemos información, lo único en lo que si afecta la demostración es que vamos a tener $|C|^9$ (con $|C|$ el tamaño del alfabeto del tableau) posibles ventanas legales, lo cual es mayor a la cantidad con ventanas de 2×6. Por lo tanto el tamaño de ϕ_{move} va a aumentar a $\leq 9 * |C|^9 * (2n^k + 3)n^k$ (con n^k siendo la profundidad del árbol generado de la ejecución de M con la cadena w), pero notamos que siguen siendo $O(n^{2k})$, por lo cual el tamaño de la formula sigue polinomial y entonces la formula sigue siendo computable en tiempo polinomial.

- ¿Es valido cambiar la demostración considerando ventanas de: 2×2?.
  Esto ya no seria tan valido, debido a que al tener ventanas de 2×2 vamos a perder algo de la información obtenida por las ventas de 2×3 (como dice la demostración), esto debido a que la cabeza de la maquina de Turing se puede mover a la izquierda o a la derecha entonces necesitamos al menos 3 columnas para poder ver como se puede mover la cabeza, y como solo tenemos dos columnas con ventanas de 2×2 esto nos puede generar problemas al intentar validar o usar las ventanas legales, por ejemplo si una transición hace cambios en tres posiciones no los podemos validar usando las ventanas de 2×2, por ejemplo si tenemos la configuración baq_1aba, con a y b caracteres de la entrada y q_1 la cabeza de la maquina de Turing indicando que se encuentra en el estado q_1, y si tenemos la siguiente transición $\delta(q_1, a) = (q_2, b, \leftarrow)$, entonces la siguiente configuración de baq_1aba seria bq_2abba, notemos que una a se cambio por b y la cabeza se movió a la izquierda por lo tanto se modificaron tres posiciones lo cual no podemos representar con ventanas 2×2, entonces no son validas para la demostración.

- ¿Que pasa si en vez de considerar máquinas de Turing No Deterministas, consideramos máquinas de Turing Verificadoras?. ¿Cómo cambiara la transformación propuesta?
  Si se usan maquinas Verificadora en vez de No Deterministas la demostración va a seguir funcionan, esto principalmente ya que podemos definir NP con maquinas No Deterministas y con maquinas Verificadoras, y son definiciones equivalentes. También notemos que la forma del tableau va a cambiar algo, esto debido que en las configuraciones que componen al tableau vamos a tener al certificado, lo cual no pasa con maquinas No Deterministas. Otra cambio importante es que en No Determinismo tenemos un árbol de cómputos y seleccionamos uno de aceptación para forma la tableau pero ahora como usamos una maquina Verificadora que es Determinista solo vamos a tener un solo computo y por lo tanto tendremos que seleccionar solo ese computo y obtener su secuencia de configuraciones para generar de igual manera el tableau (usar la secuencia de configuraciones desde la primera hasta la de aceptación y repetir la de aceptación si es necesario, como se mostró en la demostración), por lo tanto la demostración va a seguir sirviendo. Notemos que también va a cambiar quien es C, ya que vamos a tener que agregarle el alfabeto del certificado (aunque esto tal vez se podría omitir ya que C ya contiene el alfabeto de la cinta y podemos suponer que el certificado va a estar en la cinta). Otra cosa a notar es que la maquina Verificadora también debe ser de tiempo polinomial entonces tendremos que n^k va a ser la longitud del computo que genera la maquina Verificadora, por lo tanto vamos a seguir teniendo algo polinomial y la demostración se va a seguir manteniendo. Y el ultimo posible cambio a la transformación seria que ϕ_{start} ahora tendría que considerar al cer-

tificado (ya sea antes o después de la cadena del ejemplar que se va a verificar), pero notemos que el tamaño de ϕ_{start} no va a cambiar, va a seguir siendo polinomio ya que solo aumentamos la longitud del certificado y sabemos que debe ser polinomial en relación al ejemplar (o la cadena que se va a verificar) por lo tanto el tamaño de ϕ_{start} seguir siendo polinomial y entonces ϕ sigue siendo polinomial y se puede computar en tiempo polinomial, aquí recordemos que en la demostración original tenemos que $n = |w|$ y ahora agregamos el certificado pero tiene longitud polinomial en relación a w por lo tanto usar n^k va a seguir sirviendo, ya que seguimos usando polinomios, en ningún momento logramos salirnos de polinomios, ya que la suma y productos de dos polinomios es otro polinomio.

- ¿Qué cambiaría en la demostración si consideramos el modelo en el que la cinta solo es infinita en un sentido?
  Si se usan cintas solo infinitas en un sentido abría cambios pequeños pero ninguno muy grava, esto ya que sabemos (por la tesis de Church-Turing) que las maquinas de cintas infinitas hacia ambos lados y las maquinas de cintas infinitas hacia solo un lado tienen el mismo poder de computo. Solo seria notar donde empieza la cinta y revisar que no existan símbolos a la izquierda de donde inicia la cinta. Las principales diferencias que tendríamos seria en la forma en la que se ve la tableau y sobre la construcción de algunas parte de ϕ. Notemos que en la tableau vamos a tener longitud de $n^k + 3$ en vez de $2n^k + 3$, ya que como solo usamos una cinta infinita hacia un lado y como el computo toma n^k secuencias de configuraciones, tenemos que la cabeza de la maquina solo puede moverse a lo más n^k posiciones, entonces por eso ahora el tableau tendrá longitud $n^k + 3$, entonces podemos notar que el tamaño de ϕ_{cell} también va a cambiar a $(n^k + 3)n^k * (|C| + |C|^2)$, lo cual notamos que sigue siendo polinomial $O(n^{2k})$. Y por ultimo cambiaría también ϕ_{start} de esta manera $\phi_{start} = X_{1,1,\#} \wedge X_{1,2,q_0} \wedge X_{1,3,\sigma_1} \wedge \cdots \wedge X_{1,n+2,\sigma_n} \wedge X_{1,n+3,\diamond} \wedge \cdots \wedge X_{1,n^k+2,\diamond} \wedge X_{1,n^k+3,\#}$, y notamos que el tamo de ϕ_{start} seguiría siendo polinomial, es $o(n^k)$, ya que tenemos $n^k + 3$ variables $X_{i,j,s}$. Otro posible cambio seria en ϕ_{move} ya que va a depender de la cantidad de ventanas legales que existan en la maquina, pero como usa un cinta infinita a solo un lado entonces vamos a tener diferentes ventanas legales que una maquina infinita a ambos sentidos, ya que si la cabeza esta en la primera posición de la cinta (en el caso de la cinta infinita solo hacia un lado), no se puede mover a la izquierda en cambio si la cinta en infinita hacia ambos lados, la cabeza si se puede mover a la izquierda al inicio. Por lo tanto notamos que no muchas cosas van a cambiar en la demostración y seguirá funcionando de la misma manera, solo con pequeños ajustes.

Ejercicio 3

De acuerdo al ejercicio 1, demuestra que el siguiente par de problemas son equivalentes en dificultad. Ejemplifica las transformaciones correspondientes.

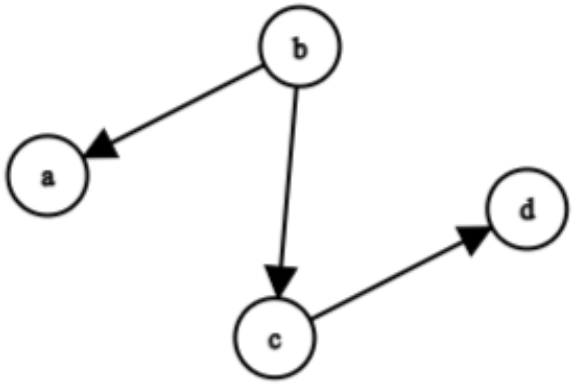
- Problema: CICLO HAMILTONIANO DIRIGIDO (CHD)
  Ejemplar: Una gráfica dirigida
  Pregunta: ¿Existe un ciclo dirigido que pase por todos los vertices exactamente una vez?

- Problema: CICLO HAMILTONIANO NO DIRIGIDO (CHND)
  Ejemplar: Una gráfica no dirigida
  Pregunta: ¿Existe un ciclo no dirigido que pase por todos los vértices exactamente una vez?

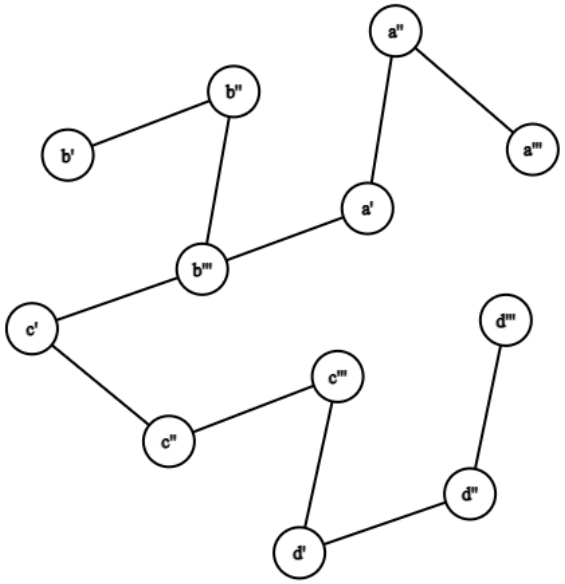
Para esto veremos que CHD $\leq_p$ CHND y CHND $\leq_p$ CHD, de esta manera viendo que ambos problemas son equivalentes en dificultad.

Primero veamos el caso para CHD $\leq_p$ CHND
Para esto diremos que f es una transformación que convierte ejemplares de CHD a ejemplares de CHND, es decir transforma gráficas dirigidas a gráficas no dirigidas.
Para esto diremos que f es una transformación que convierte ejemplares de CHD a ejemplares de CHND, es decir transforma gráficas dirigidas a gráficas no dirigidas.
Para esto veamos como funciona f, va recibir una gráfica $G = (V, E)$ dirigida y genera una gráfica $G' = (V', E')$ no dirigida.
Para esto tendremos que por cada vértice v en G se agregaran los vértices $v^{'}$, $v^{''}$ y $v^{'''}$ a V' (los vértices de G') y se agregaran las aristas no dirigidas $(v^{'}, v^{''})$ y $(v^{''}, v^{'''})$ a E' (las aristas no dirigidas de G'). Y por cada arista dirigida (u, v) en G se agregara la arista no dirigida $(u^{'''}, v^{'})$ a E' (las aristas no dirigidas de G').
De esta manera f solo agregara tres vértices y dos aristas no dirigidas a G' por cada vértice de G, y una arista no dirigida por cada arista dirigida en G.
Notemos que $3|V| = |V'|$ (ya que por cada vértice de G se agregaron tres vértices a G') y $|E|+2|V| = |E'|$ (ya que por cada arista dirigida en G se agrega una arista no dirigida en G' y por cada vértice en G se agregaron dos aristas no dirigidas a G'), y como solo se agregan aristas y vértices, tenemos que f es computable y en tiempo polinomial, ya que toma $O(5n + e)$, con n la cantidad de vértices en G y e la cantidad de aristas en G, esto ya que agregar los vértices es $O(3n)$ y agregar las aristas es $O(e + 2n)$
Ahora veamos que $G \in$ CHD $\implies f(G) \in$ CHND y $G \notin$ CHD $\implies f(G) \notin$ CHND
Digamos que $G \in$ CHD, entonces existe un ciclo Hamiltoniano en G, por lo cual sea
$(u_1, u_2), (u_2, u_3), (u_3, u_4), ..., (u_{n-1}, u_n), (u_n, u_1)$ la secuencia de aristas dirigidas de G que forman el ciclo Hamiltoniano (pasa por todos los vértices exactamente una vez), ahora notemos que la secuencia de aristas dirigidas
$(u_1^{'}, u_1^{''}), (u_1^{''}, u_1^{'''}), (u_1^{'''}, u_2^{'}), (u_2^{'}, u_2^{''}), (u_2^{''}, u_2^{'''}), (u_2^{'''}, u_3^{'}), (u_3^{'}, u_3^{''}), (u_3^{''}, u_3^{'''}), (u_3^{'''}, u_4^{'}), ..., (u_{n-1}^{'''}, u_n^{'}),$
$(u_n^{'}, u_n^{''}), (u_n^{''}, u_n^{'''}), (u_n^{'''}, u_1^{'})$ existe en $f(G)$, esto debido a la forma de construir $f(G)$, ya que las aristas no dirigidas $(u_i^{'}, u_i^{''})$ y $(u_i^{''}, u_i^{'''})$ existen por la forma en que se transformo el vértice v_i y la arista no dirigida $(u_i^{'''}, u_j^{'})$ existe por la forma en la que transformamos la arista dirigida (u_i, u_j), y por lo tanto es un ciclo Hamiltoniano en $f(G)$, ya que pasa por todos los vértices de $f(G)$ exactamente una vez, entonces se cumple $G \in$ CHD $\implies f(G) \in$ CHND
Después, digamos que $G \notin$ CHD, entonces no existe un ciclo Hamiltoniano en G, ahora supongamos que $f(G) \in$ CHND para generar una contradicción.
Como $f(G) \in$ CHND entonces tiene un ciclo Hamiltoniano, notemos que todos los vértices $v_i^{''}$ de $f(G)$ tienen grado dos (por la forma en la que se construye $f(G)$), entonces en la secuencia de vertices que

son el ciclo Hamiltonianos tenemos que tener que $v_i^{'}$ va antes de $v_i^{''}$ y $v_i^{'''}$ va después de $v_i^{''}$ (o puede ser el caso de que $v_i^{'''}$ va antes de $v_i^{''}$ y $v_i^{'}$ va después de $v_i^{''}$, pero este caso es muy similar, solo el orden de los vértices/aristas esta al revés), por lo tanto tenemos que el ciclo Hamiltoniano de $g(G)$ se debe ver algo así $v_1^{'}, v_1^{''}, v_1^{'''}, v_2^{'}, v_2^{''}, v_2^{'''}, v_3^{'}, v_3^{''}, v_3^{'''}, v_4^{'}, ..., v_{n-1}^{'''}, v_n^{'}, v_n^{''}, v_n^{'''}, v_1^{'}$ (o en el orden inverso) y vemos este ciclo Hamiltoniano como una secuencia de aristas no dirigidas seria algo asi
$(u_1^{'}, u_1^{''}), (u_1^{''}, u_1^{'''}), (u_1^{'''}, u_2^{'}), (u_2^{'}, u_2^{''}), (u_2^{''}, u_2^{'''}), (u_2^{'''}, u_3^{'}), (u_3^{'}, u_3^{''}), (u_3^{''}, u_3^{'''}), (u_3^{'''}, u_4^{'}), ..., (u_{n-1}^{'''}, u_n^{'}),$
$(u_n^{'}, u_n^{''}), (u_n^{''}, u_n^{'''}), (u_n^{'''}, u_1^{'})$ (o en el orden inverso), entonces observemos que la secuencia de aristas dirigidas $(u_1, u_2), (u_2, u_3), (u_3, u_4), ..., (u_{n-1}, u_n), (u_n, u_1)$ debe existir en G por la forma en la que se construye $g(G)$, notemos que también la secuencia de aristas no dirigidas
$(u_1, u_n), (u_n, u_{n-1}), (u_{n-1}, u_{n-2}), ..., (u_3, u_2), (u_2, u_1)$ debe existir en G por la forma en la que se construye $f(G)$ y pasan exactamente solo una vez por cada vértice y por lo tanto existiría un ciclo Hamiltoniano en G, lo cual es una contradicción a $G \notin$ CHD, ya que si $G \notin$ CHD significa que no tiene ciclos Hamiltonianos, y la contradicción surge de suponer que $f(G) \in$ CHND, por lo tanto tenemos que tener que $f(G) \notin$ CHND, entonces se cumple $G \notin$ CHD $\implies f(G) \notin$ CHND
Por lo cual f es valida y podemos concluir que CHD $\leq_p$ CHND
Veamos un ejemplo, sea G la siguiente gráfica



Entonces aplicando los pasos de f, para el vértice a se crean los vértices a', a'' y a''' y se agregan las aristas no dirigida (a', a'') y (a'', a'''), luego para el vértice b se crean los vértices b', b'' y b''' y se agregan las aristas no dirigida (b', b'') y (b'', b'''), después para el vértice c se crean los vértices c', c'' y c''' y se agregan las aristas no dirigida (c', c'') y (c'', c''') y para el vértice d se crean los vértices d', d'' y d''' y se agregan las aristas no dirigida (d', d'') y (d'', d'''). Luego por la arista dirigida (b, a) se agrega la arista no dirigida (b''', a'), después por la arista dirigida (b, c) se agrega la arista no dirigida (b''', c') y por la arista dirigida (c, d) se agrega la arista no dirigida (c''', d')
Entonces obtenemos la gráfica $f(G)$



Ahora veamos el caso para CHND $\leq_p$ CHD

Para esto diremos que g es una transformación que convierte ejemplares de CHND a ejemplares de CHD, es decir transforma gráficas no dirigidas a gráficas dirigidas.

Para esto veamos como funciona g, va recibir una gráfica $G = (V, E)$ no dirigida y genera una gráfica $G' = (V', E')$ dirigida.

Tendremos que $V = V'$, es decir G y G' tiene los mismos vértices. Y para E' tendremos que por cada arista no dirigida (u, v) en E se va a agregar las aristas dirigidas (u, v) y (v, u) (es decir son dos aristas dirigidas en opuestas direcciones).

Por lo tanto g solo agrega todas los vértices de G a G' y luego por cada arista no dirigida (u, v) en G se agregan las aristas dirigidas (u, v) y (v, u).

Notemos que $|V| = |V'|$ (ya que son los mismos vértices) y $2|E| = |E'|$ (ya que por cada arista no dirigida en G se agregan dos en G'), y como solo se agregan aristas y vértices, tenemos que g es computable y en tiempo polinomial, ya que toma $O(n + 2e)$, con n la cantidad de vértices en G y e la cantidad de aristas en G, esto ya que agregar los vértices es $O(n)$ y agregar las aristas es $O(2e)$

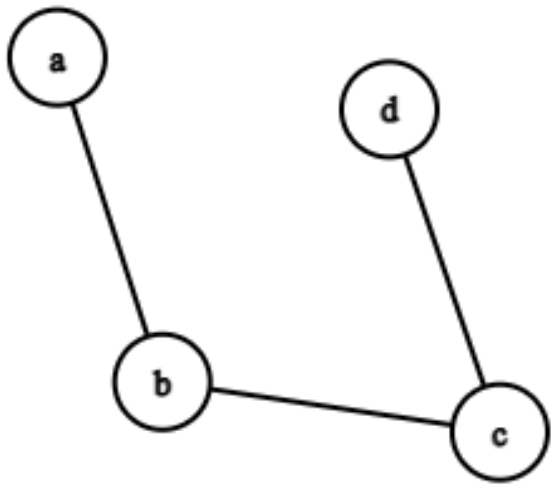
Ahora veamos que $G \in$ CHND $\implies g(G) \in$ CHD y $G \notin$ CHND $\implies g(G) \notin$ CHD

Digamos que $G \in$ CHND, entonces existe un ciclo Hamiltoniano en G, por lo cual sea
$(u_1, u_2), (u_2, u_3), (u_3, u_4), ..., (u_{n-1}, u_n), (u_n, u_1)$ la secuencia de aristas no dirigidas de G que forman el ciclo Hamiltoniano (pasa por todos los vértices exactamente una vez), ahora notemos que en G por cada arista no dirigida (u_i, u_j) se agregan a $g(G)$ las aristas dirigidas (u_i, u_j) y (u_j, u_i), por lo tanto la secuencia de aristas dirigidas
$(u_1, u_2), (u_2, u_3), (u_3, u_4), ..., (u_{n-1}, u_n), (u_n, u_1)$ existe en $g(G)$ y por lo tanto es un ciclo Hamiltoniano en $g(G)$, ya que pasa por todos los vértices exactamente una vez, entonces se cumple $G \in$ CHND $\implies g(G) \in$ CHD

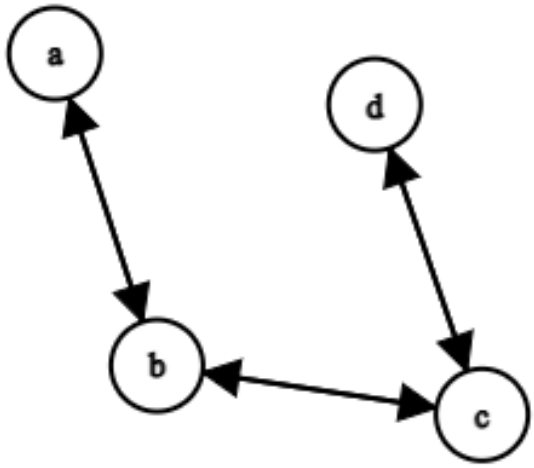
Después, digamos que $G \notin$ CHND, entonces no existe un ciclo Hamiltoniano en G, ahora supongamos que $g(G) \in$ CHD para generar una contradicción.

Como $g(G) \in$ CHD entonces tiene un ciclo Hamiltoniano, por lo cual sea
$(u_1, u_2), (u_2, u_3), (u_3, u_4), ..., (u_{n-1}, u_n), (u_n, u_1)$ la secuencia de aristas dirigidas de $g(G)$ que forman el ciclo Hamiltoniano (pasa por todos los vértices exactamente una vez), entonces notemos que el ciclo generado por las aristas no dirigidas
$(u_1, u_2), (u_2, u_3), (u_3, u_4), ..., (u_{n-1}, u_n), (u_n, u_1)$ debe existir en G, ya que si la arista dirigida (u_i, u_j) existe

en $g(G)$ significa que la arista no dirigida (u_i, u_j) existe en G (o puede ser la arista (u_j, u_i) pero como son no dirigidas, no importa tanto el orden) por la manera en la que se construye $g(G)$, por lo tanto esa secuencia de aristas existe generando un ciclo y es un ciclo Hamiltoniano (ya que es un ciclo Hamiltoniano en $g(G)$, al pasar por todos los vértices exactamente una vez), por lo cual tenemos que en G existe un ciclo Hamiltoniano, lo cual es una contradicción a $G \notin$ CHND, ya que si $G \notin$ CHND significa que no tiene ciclos Hamiltonianos, y la contradicción surge de suponer que $g(G) \in$ CHD, por lo tanto tenemos que tener que $g(G) \notin$ CHD, entonces se cumple $G \notin$ CHND $\implies g(G) \notin$ CHD
Por lo cual g es valida y podemos concluir que CHND $\leq_p$ CHD
Veamos un ejemplo, sea G la siguiente gráfica



Entonces aplicando los pasos de g, todos los vértices de G los agregamos, luego para la arista no dirigida (a, b) se agregan las aristas dirigidas (a, b) y (b, a), después para la arista no dirigida (b, c) se agregan las aristas dirigidas (b, c) y (c, b) y por ultimo para la arista no dirigida (c, d) se agregan las aristas dirigidas (c, d) y (d, c)
Entonces obtenemos la gráfica $g(G)$



Y como CHD $\leq_p$ CHND y CHND $\leq_p$ CHD, podemos concluir que ambos problemas son equivalentes en dificultad.

Ejercicio 4

Considera el siguiente par de problemas:

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

- Problema: 3SAT-IGUALDAD
  Ejemplar: Una fórmula booleana ϕ en FNC en la cual el numero de clausulas es igual al numero de variables
  Pregunta: ¿Existe una asignación que satisface ϕ?

1. Demuestra que ambos problemas con equivalentes en dificultad. Ejemplifica las transformaciones correspondientes.
   Para esto veremos que 3SAT-BALANCEADO $\leq_p$ 3SAT-IGUALDAD y 3SAT-IGUALDAD $\leq_p$ 3SAT-BALANCEADO, de esta manera viendo que ambos problemas son equivalentes en dificultad.

   Primero veamos que 3SAT-BALANCEADO $\leq_p$ 3SAT-IGUALDAD
   Para esto diremos que f es una transformación que convierte ejemplares de 3SAT-BALANCEADO a ejemplares de 3SAT-IGUALDAD.
   Para esto veamos como funciona f, va recibir una formula booleana ϕ en FNC en la cual cada cláusula contiene 3 literales, y cada variable en ϕ aparece negada y sin negación el mismo número de veces y genera una formula booleana ϕ' en FNC en la cual el numero de clausulas es igual al numero de variables.
   Veamos como funcionara f, lo primero que se va a realizar es recorrer ϕ (o ϕ y su lista de variables, dependiendo de la representación) para poder contar cuantas clausulas tiene y cuantas variables tiene (notemos que dependiendo de la representación de ϕ la manera de obtener la cantidad de variables y clausulas puede cambiar, por ejemplo si solo es la formula o si tenemos la formula y una lista de variables), entonces sea a la cantidad de clausulas en ϕ y sea b la cantidad de variables de ϕ.
   Luego todas las variables y clausulas (agregando $\wedge$ entre cada clausula para que este en FNC, de igual manera que en ϕ) de ϕ las copiamos o ponemos en ϕ'.
   Y por ultimo revisamos si $a = b$, en el caso de que sean iguales entonces ya terminamos y regresamos ϕ'. En el caso de que $a < b$, significa que tenemos menos clausulas que variables, por lo tanto sea $d = b - a$ (la diferencia entre variables y clausulas), entonces agregamos la clausula $(Verdadero \vee Verdadero \vee Verdadero)$ a ϕ' (agregando $\wedge$ entre cada clausula para que este en FNC) d veces, es decir agregamos d veces la clausula $(Verdadero \vee Verdadero \vee Verdadero)$ a ϕ' y terminamos regresando ϕ'. Y en el caso de que $a > b$, significa que tenemos menos variables que clausulas, por lo tanto sea $c = a - b$ (la diferencia entre variables y clausulas), entonces agregamos las variables v_{b+1}, v_{b+2}, ..., v_{b+c-1}, v_{b+c} a ϕ', es decir agregamos c nuevas variables a ϕ' y terminamos regresando ϕ'.
   Notemos que la si $a = b$, entonces la cantidad de clausulas y variables en ϕ' es la misma que en ϕ.
   En el caso de que $a < b$, la cantidad de variables en ϕ' y ϕ son la misma, y la cantidad de clausulas en ϕ' es la cantidad de variables en ϕ.
   Y en el caso de que $a > b$, la cantidad de clausulas en ϕ' y ϕ son la misma, y la cantidad de variables en ϕ' es la cantidad de clausulas en ϕ. En general la cantidad de clausulas y variables en

ϕ' es el máximo entre a y b. Por lo tanto tenemos que solo agregamos una cantidad polinomial de variables y clausulas a ϕ' podemos concluir que f es computable y en tiempo polinomial, ya que toma $O(2 * max\{a, b\})$, con a la cantidad de clausulas en ϕ y b la cantidad de variables de ϕ.
Ahora veamos que $\phi \in$ 3SAT-BALANCEADO $\implies f(\phi) \in$ 3SAT-IGUALDAD y $\phi \notin$ 3SAT-BALANCEADO $\implies f(\phi) \notin$ 3SAT-IGUALDAD
Digamos que $\phi \in$ 3SAT-BALANCEADO, entonces existe una asignación de valor para las variables de ϕ que la satisfacen, llamemos a esta asignación de valores x.
Ahora tenemos tres casos:
Caso 1, tenemos que la cantidad de clausulas y variables en ϕ es la misma, entonces $f(\phi)$ sera igual a ϕ, por lo tanto la asignación de valores x satisface a $f(\phi)$.
Caso 2, tenemos que la cantidad de clausulas en ϕ es mayor a la cantidad de variables en ϕ, entonces sabemos que $f(\phi)$ tendrá c (con c siendo la cantidad de clausulas en ϕ menos la cantidad de variables en ϕ) variables extras que no tiene ϕ, digamos que son v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} (con b siendo la cantidad de variables en ϕ), por lo tanto la asignación de valores x', con x' siendo la misma asignación que x solo agregando las asignaciones $v_{b+1} = Verdadero$, $v_{b+2} = Verdadero$, ..., $v_{b+c-1} = Verdadero$ y $v_{b+c} = Verdadero$ satisface a $f(\phi)$, esto ya que no se agregaron clausulas nuevas y las variables v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} no aparecen en ninguna clausula, por lo tanto x' satisface a $f(\phi)$.
Caso 3, tenemos que la cantidad de variables en ϕ es mayor a la cantidad de clausulas en ϕ, entonces sabemos que $f(\phi)$ tendrá d (con d siendo la cantidad de variables en ϕ menos la cantidad de clausulas en ϕ) clausulas extras que no tiene ϕ, las cuales sabemos que son $(Verdadero \vee Verdadero \vee Verdadero)$ todas, por lo tanto la asignación de valores x satisface a $f(\phi)$, esto ya que las clausulas que se agregaron a $f(\phi)$ siempre se satisfacen (son un $\vee$ de $Verdaderos$) por lo tanto x' satisface a $f(\phi)$.
Entonces concluimos que $\phi \in$ 3SAT-BALANCEADO $\implies f(\phi) \in$ 3SAT-IGUALDAD se cumple.
Después, digamos que $\phi \notin$ 3SAT-BALANCEADO, entonces no existe una asignación de valores para sus variables que la satisfaga, ahora supongamos que $f(\phi) \in$ 3SAT-IGUALDAD para generar una contradicción.
Como $f(\phi) \in$ 3SAT-IGUALDAD entonces existe una asignación de valor para las variables de $f(\phi)$ que la satisfacen, llamemos a esta asignación de valores x'.
Ahora tenemos tres casos:
Caso 1, tenemos que la cantidad de clausulas y variables en ϕ es la misma, entonces $f(\phi)$ sera igual a ϕ, por lo tanto la asignación de valores x' satisface a ϕ, lo cual genera una contradicción ya que $\phi \notin$ 3SAT-BALANCEADO.
Caso 2, tenemos que la cantidad de clausulas en ϕ es mayor a la cantidad de variables en ϕ, entonces sabemos que $f(\phi)$ tendrá c (con c siendo la cantidad de clausulas en ϕ menos la cantidad de variables en ϕ) variables extras que no tiene ϕ, digamos que son v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} (con b siendo la cantidad de variables en ϕ), por lo tanto la asignación de valores x, con x siendo la misma asignación que x' solo quitando las asignaciones de valores de v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} satisface a ϕ, esto ya que no se agregaron clausulas nuevas a $f(\phi)$ y si quitamos las variables v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} a $f(\phi)$ obtenemos a ϕ y sabemos que las variables v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} no aparecen en ninguna clausula, lo cual genera una contradicción ya que $\phi \notin$ 3SAT-BALANCEADO.
Caso 3, tenemos que la cantidad de variables en ϕ es mayor a la cantidad de clausulas en ϕ, entonces sabemos que $f(\phi)$ tendrá d (con d siendo la cantidad de variables en ϕ menos la cantidad de clausulas en ϕ) clausulas extras que no tiene ϕ, las cuales sabemos que son $(Verdadero \vee Verdadero \vee Verdadero)$ todas, por lo tanto la asignación de valores x' satisface a ϕ, esto ya que las clausulas que se agregaron a $f(\phi)$ siempre se satisfacen (son un $\vee$ de $Verdaderos$) y si le quitamos a $f(\phi)$ las d clausulas de la forma $(Verdadero \vee Verdadero \vee Verdadero)$ que se agregaron en f obtenemos a ϕ, entonces la asignación x' debería satisfacer a ϕ, lo cual genera una contradicción ya que $\phi \notin$ 3SAT-BALANCEADO.

Notemos que en todos los casos llegamos a un contradicción generada por suponer que $f(\phi) \in$ 3SAT-IGUALDAD, por lo tanto concluimos que $f(\phi) \notin$ 3SAT-IGUALDAD.
Entonces concluimos que $\phi \notin$ 3SAT-BALANCEADO $\implies f(\phi) \notin$ 3SAT-IGUALDAD se cumple.
Por lo cual f es valida y podemos concluir que 3SAT-BALANCEADO $\leq_p$ 3SAT-IGUALDAD.
Veamos un ejemplo, sea ϕ esta formula $(x_1 \vee \neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3 \vee \neg x_3)$ y la lista de variables (x_1, x_2, x_3), ahora contamos la cantidad de clausulas que es 2 y la cantidad de variables es 3, entonces $f(\phi)$ seria la formula $(x_1 \vee \neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3 \vee \neg x_3) \wedge (True \vee True \vee True)$ y la lista de variables (x_1, x_2, x_3), esto ya que solo se agrego una clausula $(True \vee True \vee True)$.
Otro ejemplo, sea ϕ esta formula $(x_1 \vee \neg x_1 \vee x_1) \wedge (\neg x_1 \vee x_1 \vee \neg x_1)$ y la lista de variables (x_1), ahora contamos la cantidad de clausulas que es 2 y la cantidad de variables es 1, entonces $f(\phi)$ seria la formula $(x_1 \vee \neg x_1 \vee x_1) \wedge (\neg x_1 \vee x_1 \vee \neg x_1)$ y la lista de variables (x_1, x_2), esto ya que solo se agrego la variable x_2 que no es usada en la formula.

Ahora veamos que 3SAT-IGUALDAD $\leq_p$ 3SAT-BALANCEADO
Para esto diremos que g es una transformación que convierte ejemplares de 3SAT-IGUALDAD a ejemplares de 3SAT-BALANCEADO.
Para esto veamos como funciona g, va recibir una formula booleana ϕ en FNC en la cual el numero de clausulas es igual al numero de variables y genera una formula booleana ϕ' en FNC en la cual cada cláusula contiene 3 literales, y cada variable en ϕ aparece negada y sin negación el mismo número de veces.
Veamos como funcionara g, lo primero que se va a realizar es que a todas las variables y clausulas (agregando $\wedge$ entre cada clausula para que este en FNC, de igual manera que en ϕ) de ϕ las copiamos o ponemos en ϕ'.
Luego por cada variable v_i de ϕ vamos a recorrer la formula ϕ y tendremos una variable z (que inicialmente vale 0), entonces recorremos cada clausula de ϕ y si encontramos la variable v_o sin negar entonces sumamos uno a z y si encontramos la variable v_i negada entonces restamos uno a z.
Luego de recorrer toda la formula ϕ tendremos a z y tendremos tres casos:
Caso 1, cuando z es cero, entonces seguimos revisando las demás variables.
Caso 2, cuando z es negativo, entonces sea $z' = z * -1$, ahora agregamos la clausula $(v_i \vee \neg v_i \vee v_i)$ a ϕ' un total de z' veces, para poder balancear la variable v_i, ya que tiene z' más apariciones negada que no negada, y luego seguimos revisando las demás variables.
Caso 3, cuando z es positivo, ahora agregamos la clausula $(\neg v_i \vee v_i \vee \neg v_i)$ a ϕ' un total de z veces, para poder balancear la variable v_i, ya que tiene z más apariciones no negada que negada, y luego seguimos revisando las demás variables.
Y por ultimo, luego de revisar todas las variables, regresamos ϕ'.
Notemos que la cantidad de variables en ϕ y ϕ' es la misma, ya que no se agregan variables extras.
Ahora notemos de clausulas en ϕ' es la cantidad de clausulas en ϕ más la cantidad de variables en ϕ multiplicado por tres veces la cantidad de clausulas de ϕ', ya que en el peor de los casos para cada variable (la variable solo aparece sin negar o solo negada) se agregan tantas clausulas como veces que aparece.
Por lo tanto tenemos que solo agregamos una cantidad polinomial de variables y clausulas a ϕ' podemos concluir que g es computable y en tiempo polinomial, ya que toma $O(b + a + b * a * 3)$, con a la cantidad de clausulas en ϕ y b la cantidad de variables de ϕ.
Ahora veamos que $\phi \in$ 3SAT-IGUALDAD $\implies g(\phi) \in$ 3SAT-BALANCEADO y $\phi \notin$ 3SAT-IGUALDAD $\implies g(\phi) \notin$ 3SAT-BALANCEADO
Digamos que $\phi \in$ 3SAT-IGUALDAD, entonces existe una asignación de valor para las variables de ϕ que la satisfacen, llamemos a esta asignación de valores x.
Ahora notemos que la asignación x debe satisfacer a $g(\phi)$ esto debido a que ϕ y $g(\phi)$ tienen las mismas variables por la forma en que funciona g, ademas $g(\phi)$ tiene todas las clausulas de ϕ y puede

tener una clausulas extras (esto por la forma en que funciona g) que son de la forma $(v_i \vee \neg v_i \vee v_i)$ o $(\neg v_i \vee v_i \vee \neg v_i)$ (con v_i una variable de ϕ), entonces notemos que las clausulas $(v_i \vee \neg v_i \vee v_i)$ y $(\neg v_i \vee v_i \vee \neg v_i)$ siempre van a ser verdaderas mientras que v_i tenga un valor, por lo tanto la asignación x debería satisfacer $g(\phi)$.
Entonces concluimos que $\phi \in$ 3SAT-IGUALDAD $\implies g(\phi) \in$ 3SAT-BALANCEADO se cumple.
Después, digamos que $\phi \notin$ 3SAT-IGUALDAD, entonces no existe una asignación de valores para sus variables que la satisfaga, ahora supongamos que $g(\phi) \in$ 3SAT-BALANCEADO para generar una contradicción.
Como $f(\phi) \in$ 3SAT-BALANCEADO entonces existe una asignación de valor para las variables de $g(\phi)$ que la satisfacen, llamemos a esta asignación de valores x'.
Entonces notemos que la asignación x' también satisface a ϕ, esto debido a que ϕ y $g(\phi)$ tienen las mismas variables, por la forma en que g funciona y notemos ademas que la diferencia entre las clausulas de ϕ y $g(\phi)$ es que $g(\phi)$ puede tener algunas clausulas extras (esto por la forma en que funciona g) de la forma $(v_i \vee \neg v_i \vee v_i)$ o $(\neg v_i \vee v_i \vee \neg v_i)$ (con v_i una variable de ϕ) y todas las clausulas de ϕ están en $g(\phi)$, entonces notemos que si solo tenemos las clausulas que están en ϕ y $g(\phi)$ vamos a quitar las clausulas que agrego g (las que son de la forma $(v_i \vee \neg v_i \vee v_i)$ o $(\neg v_i \vee v_i \vee \neg v_i)$), entonces observemos que las formulas que agrega g siempre son verdaderas mientras sus variables tengan un valor (v_i tenga un valor de verdad), por lo tanto la asignación x' va a satisfacer a ϕ, ya que todas las clausulas y variables de ϕ están en $g(\phi)$, por lo tanto existiría una asignación de verdad paras las variables de ϕ que la satisfacen, lo cual es una contradicción ya que $\phi \notin$ 3SAT-IGUALDAD y esta contradicción es generada por suponer que $f(\phi) \in$ 3SAT-BALANCEADO, por lo tanto concluimos que $f(\phi) \notin$ 3SAT-BALANCEADO.
Entonces concluimos que $\phi \notin$ 3SAT-IGUALDAD $\implies g(\phi) \notin$ 3SAT-BALANCEADO se cumple.
Por lo cual g es valida y podemos concluir que 3SAT-IGUALDAD $\leq_p$ 3SAT-BALANCEADO.
Veamos un ejemplo, sea ϕ esta formula $(x_1 \vee x_2 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_2)$ y la lista de variables (x_1, x_2), ahora a la variable $z_{x_1} = 0$ le sumamos uno por cada x_1 sin negar y le restamos uno por cada x_1 negada, entonces vemos que $z_{x_1} = 2$, por lo tanto tendremos que agregar las clausulas $(\neg x_1 \vee x_1 \vee \neg x_1)$ y $(\neg x_1 \vee x_1 \vee \neg x_1)$ a $g(\phi)$, luego sea la variable $z_{x_2} = 0$ le sumamos uno por cada x_2 sin negar y le restamos uno por cada x_2 negada, entonces vemos que $z_{x_2} = 2$, por lo tanto tendremos que agregar las clausulas $(\neg x_2 \vee x_2 \vee \neg x_2)$ y $(\neg x_2 \vee x_2 \vee \neg x_2)$ a $g(\phi)$, entonces $g(\phi)$ seria la formula $(x_1 \vee x_2 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee x_2) \wedge (\neg x_1 \vee x_1 \vee \neg x_1) \wedge (\neg x_1 \vee x_1 \vee \neg x_1) \wedge (\neg x_2 \vee x_2 \vee \neg x_2) \wedge (\neg x_2 \vee x_2 \vee \neg x_2)$ y la lista de variables (x_1, x_2), ya que solo se agregaron todas las clausulas y variables de ϕ y las clausulas extras que vimos arriba.
Otro ejemplo, sea ϕ esta formula $(\neg x_1 \vee \neg x_1 \vee \neg x_1)$ y la lista de variables (x_1), ahora a la variable $z_{x_1} = 0$ le sumamos uno por cada x_1 sin negar y le restamos uno por cada x_1 negada, entonces vemos que $z_{x_1} = -3$, por lo tanto tendremos que agregar las clausulas $(x_1 \vee \neg x_1 \vee x_1)$, $(x_1 \vee \neg x_1 \vee x_1)$ y $(x_1 \vee \neg x_1 \vee x_1)$ a $g(\phi)$, entonces $g(\phi)$ seria la formula $(\neg x_1 \vee \neg x_1 \vee \neg x_1) \wedge (x_1 \vee \neg x_1 \vee x_1) \wedge (x_1 \vee \neg x_1 \vee x_1) \wedge (x_1 \vee \neg x_1 \vee x_1)$ y la lista de variables (x_1), ya que solo se agregaron todas las clausulas y variables de ϕ y las clausulas extras que vimos arriba.

Y como 3SAT-BALANCEADO $\leq_p$ 3SAT-IGUALDAD y 3SAT-IGUALDAD $\leq_p$ 3SAT-BALANCEADO, podemos concluir que ambos problemas son equivalentes en dificultad.

2. Demuestra que ambos problemas son NP-Completos
   Para esto veremos que 3SAT-BALANCEADO y 3SAT-IGUALDAD estan en NP, luego que 3SAT es NP-Completo y por ultimo que 3SAT $\leq_p$ 3SAT-IGUALDAD

   Primero veamos que 3SAT-BALANCEADO es NP
   Este problema pertenece a *NP*, ya que no se conoce un algoritmo en tiempo polinomial que lo

resuelva.
Para esto diremos que el certificado u será cada variable de ϕ (donde ϕ es el ejemplar de 3SAT-BALANCEADO) 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 ϕ) variables en ϕ, 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), ya que es polinomial en relación con la entrada original (la formula ϕ), ya que la entrada que representa a ϕ debe contener todas sus variables y clausulas o una forma de poder obtenerlos.
Ahora veamos una Máquina de Turing que reciba ϕ y el certificado u y evalué a ϕ usando los valores del certificado para saber si ϕ 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:` Una fórmula boolena ϕ en FNC en la cual cada cláusula contiene 3 literales, y cada variable en ϕ aparece negada y sin negación el mismo número de veces y certificado u con los valores para las variables de ϕ, para facilitar un poco las cosas digamos que el certificado es un arreglo donde en la posición i del arreglo se encuentra el valor de verdad de la i-esima variable de ϕ (notemos que el certificado puede ser presentado de varias maneras y esto no afectara tanto la complejidad).
- `Output:` 1 si ϕ se evalúa verdadero usando los valores de u, 0 en caso de que se evalúa falso.

a) Recorremos la fórmula, si encontramos una variable la sustituimos por su valor asignado por el certificado (si es verdadera o falsa).

b) 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 α, β, γ son funciones booleanas y x es una variable.
La función $eval(\phi, u)$ recibe una función booleana ϕ 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:

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

Ahora veamos la complejidad del algoritmo de verificación.
El paso 1 (a)) toma $O(n)$ ya que solo se recorre toda la fórmula una vez $O(n)$, se realizan asignaciones $O(1)$ y lecturas del certificado $O(1)$, esto a cambiar variables por su valor de verdad, con n siendo la longitud de la cadena de entrada o podemos ver lo como que se toma $O(x + y)$ con x la cantidad de variables en ϕ y y la cantidad de operadores en ϕ, ya que tenemos que recorrer toda la formula, por lo tanto todo tomaría $O(x + y)$.
Para el paso 2 (b)), notemos que se realiza una llamada de $eval()$ para la fórmula ϕ y luego se hace otra llamada para cada variable y operador de ϕ por lo tanto se hacen a lo más n llamadas extras

y con esto sabemos que el paso 2 toma $O(n)$, con n siendo la longitud de la cadena de entrada o podemos ver lo como que se hacen $x + y$ llamadas extras con x la cantidad de variables en ϕ y y la cantidad de operadores en ϕ, por lo tanto todo tomaría $O(x + y)$.
Entonces todo el algoritmo nos tomó $O(n)$, lo cual es una complejidad polinomial, con n siendo la longitud de la cadena de entrada o si lo vemos en términos de variables y operadores todo nos toma $O(x + y)$ con x la cantidad de variables en ϕ y y la cantidad de operadores en ϕ, lo cual también es 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 3SAT-BALANCEADO 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.

Ahora veamos que 3SAT-IGUALDAD es NP
Esto es de manera muy similar a como vimos que 3SAT-BALANCEADO es NP (usaremos el mismo certificado y algoritmo para verificar visto en el ejercicio de arriba)
Este problema pertenece a *NP*, ya que no se conoce un algoritmo en tiempo polinomial que lo resuelva.
Para esto diremos que el certificado u será cada variable de ϕ (donde ϕ es el ejemplar de 3SAT-IGUALDAD) 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 ϕ) variables en ϕ, 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), ya que es polinomial en relación con la entrada original (la formula ϕ), ya que la entrada que representa a ϕ debe contener todas sus variables y clausulas o una forma de poder obtenerlos.
Ahora veamos una Máquina de Turing que reciba ϕ y el certificado u y evalué a ϕ usando los valores del certificado para saber si ϕ 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:` Una fórmula boolena ϕ en FNC en la cual cada cláusula contiene 3 literales, y cada variable en ϕ aparece negada y sin negación el mismo número de veces y certificado u con los valores para las variables de ϕ, para facilitar un poco las cosas digamos que el certificado es un arreglo donde en la posición i del arreglo se encuentra el valor de verdad de la i-esima variable de ϕ (notemos que el certificado puede ser presentado de varias maneras y esto no afectara tanto la complejidad).
- `Output:` 1 si ϕ se evalúa verdadero usando los valores de u, 0 en caso de que se evalúa falso.

a) Recorremos la fórmula, si encontramos una variable la sustituimos por su valor asignado por el certificado (si es verdadera o falsa).

b) 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 α, β, γ son funciones booleanas y x es una variable.
La función $eval(\phi, u)$ recibe una función booleana ϕ 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:

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

Ahora veamos la complejidad del algoritmo de verificación.
El paso 1 (a)) toma $O(n)$ ya que solo se recorre toda la fórmula una vez $O(n)$, se realizan asignaciones $O(1)$ y lecturas del certificado $O(1)$, esto a cambiar variables por su valor de verdad, con n siendo la longitud de la cadena de entrada o podemos ver lo como que se toma $O(x + y)$ con x la cantidad de variables en ϕ y y la cantidad de operadores en ϕ, ya que tenemos que recorrer toda la formula, por lo tanto todo tomaría $O(x + y)$.
Para el paso 2 (b)), notemos que se realiza una llamada de $eval()$ para la fórmula ϕ y luego se hace otra llamada para cada variable y operador de ϕ por lo tanto se hacen a lo más n llamadas extras y con esto sabemos que el paso 2 toma $O(n)$, con n siendo la longitud de la cadena de entrada o podemos ver lo como que se hacen $x + y$ llamadas extras con x la cantidad de variables en ϕ y y la cantidad de operadores en ϕ, por lo tanto todo tomaría $O(x + y)$.
Entonces todo el algoritmo nos tomó $O(n)$, lo cual es una complejidad polinomial, con n siendo la longitud de la cadena de entrada o si lo vemos en términos de variables y operadores todo nos toma $O(x + y)$ con x la cantidad de variables en ϕ y y la cantidad de operadores en ϕ, lo cual también es 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 3SAT-IGUALDAD 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.

Veamos que 3SAT es NP-Completo
Esto lo vimos en la clase del 24/10/24 y 29/10/24, entonces sabemos que 3SAT es NP-Completo.

Ahora veamos que 3SAT $\leq_p$ 3SAT-IGUALDAD
Esto sera muy similar a 3SAT-BALANCEADO $\leq_p$ 3SAT-IGUALDAD
Para esto diremos que h es una transformación que convierte ejemplares de 3SAT a ejemplares de 3SAT-IGUALDAD.
Para esto veamos como funciona h, va recibir una formula booleana ϕ en FNC en la cual cada cláusula contiene 3 literales y genera una formula booleana ϕ' en FNC en la cual el numero de clausulas es igual al numero de variables.
Veamos como funcionara h, lo primero que se va a realizar es recorrer ϕ (o ϕ y su lista de variables, dependiendo de la representación) para poder contar cuantas clausulas tiene y cuantas variables tiene (notemos que dependiendo de la representación de ϕ la manera de obtener la cantidad de variables y clausulas puede cambiar, por ejemplo si solo es la formula o si tenemos la formula y una lista de variables), entonces sea a la cantidad de clausulas en ϕ y sea b la cantidad de variables de ϕ.
Luego todas las variables y clausulas (agregando $\wedge$ entre cada clausula para que este en FNC, de igual manera que en ϕ) de ϕ las copiamos o ponemos en ϕ'.

Y por ultimo revisamos si $a = b$, en el caso de que sean iguales entonces ya terminamos y regresamos ϕ'. En el caso de que $a > b$, significa que tenemos menos variables que clausulas, por lo tanto sea $c = a - b$ (la diferencia entre variables y clausulas), entonces agregamos las variables v_{b+1}, v_{b+2}, ..., v_{b+c-1}, v_{b+c} a ϕ', es decir agregamos c nuevas variables a ϕ' y terminamos regresando ϕ'. Y en el caso de que $a < b$, significa que tenemos menos clausulas que variables, por lo tanto sea $d = b - a$ (la diferencia entre variables y clausulas), entonces agregamos la clausula $(Verdadero \vee Verdadero \vee Verdadero)$ a ϕ' (agregando $\wedge$ entre cada clausula para que este en FNC) d veces, es decir agregamos d veces la clausula $(Verdadero \vee Verdadero \vee Verdadero)$ a ϕ' y terminamos regresando ϕ'.
Notemos que la si $a = b$, entonces la cantidad de clausulas y variables en ϕ' es la misma que en ϕ. En el caso de que $a < b$, la cantidad de variables en ϕ' y ϕ son la misma, y la cantidad de clausulas en ϕ' es la cantidad de variables en ϕ. Y en el caso de que $a > b$, la cantidad de clausulas en ϕ' y ϕ son la misma, y la cantidad de variables en ϕ' es la cantidad de clausulas en ϕ. En general la cantidad de clausulas y variables en ϕ' es el máximo entre a y b. Por lo tanto tenemos que solo agregamos una cantidad polinomial de variables y clausulas a ϕ' podemos concluir que f es computable y en tiempo polinomial, ya que toma $O(2 * max\{a, b\})$, con a la cantidad de clausulas en ϕ y b la cantidad de variables de ϕ.
Ahora veamos que $\phi \in$ 3SAT $\implies h(\phi) \in$ 3SAT-IGUALDAD y $\phi \notin$ 3SAT $\implies h(\phi) \notin$ 3SAT-IGUALDAD
Digamos que $\phi \in$ 3SAT, entonces existe una asignación de valor para las variables de ϕ que la satisfacen, llamemos a esta asignación de valores x.
Ahora tenemos tres casos:
Caso 1, tenemos que la cantidad de clausulas y variables en ϕ es la misma, entonces $h(\phi)$ sera igual a ϕ, por lo tanto la asignación de valores x satisface a $h(\phi)$.
Caso 2, tenemos que la cantidad de clausulas en ϕ es mayor a la cantidad de variables en ϕ, entonces sabemos que $h(\phi)$ tendrá c (con c siendo la cantidad de clausulas en ϕ menos la cantidad de variables en ϕ) variables extras que no tiene ϕ, digamos que son v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} (con b siendo la cantidad de variables en ϕ), por lo tanto la asignación de valores x', con x' siendo la misma asignación que x solo agregando las asignaciones $v_{b+1} = Verdadero$, $v_{b+2} = Verdadero$, ..., $v_{b+c-1} = Verdadero$ y $v_{b+c} = Verdadero$ satisface a $h(\phi)$, esto ya que no se agregaron clausulas nuevas y las variables v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} no aparecen en ninguna clausula, por lo tanto x' satisface a $h(\phi)$.
Caso 3, tenemos que la cantidad de variables en ϕ es mayor a la cantidad de clausulas en ϕ, entonces sabemos que $h(\phi)$ tendrá d (con d siendo la cantidad de variables en ϕ menos la cantidad de clausulas en ϕ) clausulas extras que no tiene ϕ, las cuales sabemos que son $(Verdadero \vee Verdadero \vee Verdadero)$ todas, por lo tanto la asignación de valores x satisface a $h(\phi)$, esto ya que las clausulas que se agregaron a $h(\phi)$ siempre se satisfacen (son un $\vee$ de $Verdaderos$) por lo tanto x' satisface a $h(\phi)$.
Entonces concluimos que $\phi \in$ 3SAT $\implies h(\phi) \in$ 3SAT-IGUALDAD se cumple.
Después, digamos que $\phi \notin$ 3SAT, entonces no existe una asignación de valores para sus variables que la satisfaga, ahora supongamos que $h(\phi) \in$ 3SAT-IGUALDAD para generar una contradicción.
Como $h(\phi) \in$ 3SAT-IGUALDAD entonces existe una asignación de valor para las variables de $h(\phi)$ que la satisfacen, llamemos a esta asignación de valores x'.
Ahora tenemos tres casos:
Caso 1, tenemos que la cantidad de clausulas y variables en ϕ es la misma, entonces $h(\phi)$ sera igual a ϕ, por lo tanto la asignación de valores x' satisface a ϕ, lo cual genera una contradicción ya que $\phi \notin$ 3SAT.
Caso 2, tenemos que la cantidad de clausulas en ϕ es mayor a la cantidad de variables en ϕ, entonces sabemos que $h(\phi)$ tendrá c (con c siendo la cantidad de clausulas en ϕ menos la cantidad de variables en ϕ) variables extras que no tiene ϕ, digamos que son v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} (con b

siendo la cantidad de variables en ϕ), por lo tanto la asignación de valores x, con x siendo la misma asignación que x' solo quitando las asignaciones de valores de v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} satisface a ϕ, esto ya que no se agregaron clausulas nuevas a $h(\phi)$ y si quitamos las variables v_{b+1}, v_{b+2}, ..., v_{b+c-1} y v_{b+c} a $h(\phi)$ obtenemos a ϕ, lo cual genera una contradicción ya que $\phi \notin$ 3SAT.
Caso 3, tenemos que la cantidad de variables en ϕ es mayor a la cantidad de clausulas en ϕ es la misma, entonces sabemos que $h(\phi)$ tendrá d (con d siendo la cantidad de variables en ϕ menos la cantidad de clausulas en ϕ) clausulas extras que no tiene ϕ, las cuales sabemos que son $(Verdadero \vee Verdadero \vee Verdadero)$ todas, por lo tanto la asignación de valores x' satisface a ϕ, esto ya que las clausulas que se agregaron a $h(\phi)$ siempre se satisfacen (son un $\vee$ de $Verdaderos$) y si le quitamos a $h(\phi)$ las d clausulas de la forma $(Verdadero \vee Verdadero \vee Verdadero)$ que se agregaron en h obtenemos a ϕ, entonces la asignación x' debería satisfacer a ϕ, lo cual genera una contradicción ya que $\phi \notin$ 3SAT.
Notemos que en todos los casos llegamos a un contradicción generada por suponer que $h(\phi) \in$ 3SAT-IGUALDAD, por lo tanto concluimos que $h(\phi) \notin$ 3SAT.
Entonces concluimos que $\phi \notin$ 3SAT $\implies h(\phi) \notin$ 3SAT-IGUALDAD se cumple.
Por lo cual h es valida y podemos concluir que 3SAT $\leq_p$ 3SAT-IGUALDAD.

Y por ultimo, usando todas las cosas anteriores, veremos que 3SAT-IGUALDAD y 3SAT-BALANCEADO son NP-Completos
En la clase del 24/10/24 y 29/10/24 vimos que si tenemos un problema R es NP-Completo y otro problema S, tal que R $\leq_p$ S, entonces podemos concluir que S es NP-Dificil, esto debido a que si tenemos un lenguaje L cualquiera en NP, etonces como R es NP-Completo sabemos que L $\leq_p$ R, entonces como sabemos que L $\leq_p$ R y R $\leq_p$ S podemos decir que L $\leq_p$ S por la propiedad transitiva de $\leq_p$ vista en el ejercicio 1. Lo cual significa que S es NP-Dificil.
Ahora notemos que tenemos que 3SAT $\leq_p$ 3SAT-IGUALDAD y 3SAT es NP-Completo, por lo tanto 3SAT-IGUALDAD es NP-Dificil y como también vimos que 3SAT-IGUALDAD es NP podemos concluir que 3SAT-IGUALDAD es NP-Completo.
Después notemos que tenemos que 3SAT-IGUALDAD $\leq_p$ 3SAT-BALANCEADO y 3SAT-IGUALDAD es NP-Completo, por lo tanto 3SAT-BALANCEADO es NP-Dificil y como también vimos que 3SAT-BALANCEADO es NP podemos concluir que 3SAT-BALANCEADO es NP-Completo.

Y por lo tanto concluimos con que 3SAT-IGUALDAD y 3SAT-BALANCEADO son NP-Completos.

Ejercicio 5

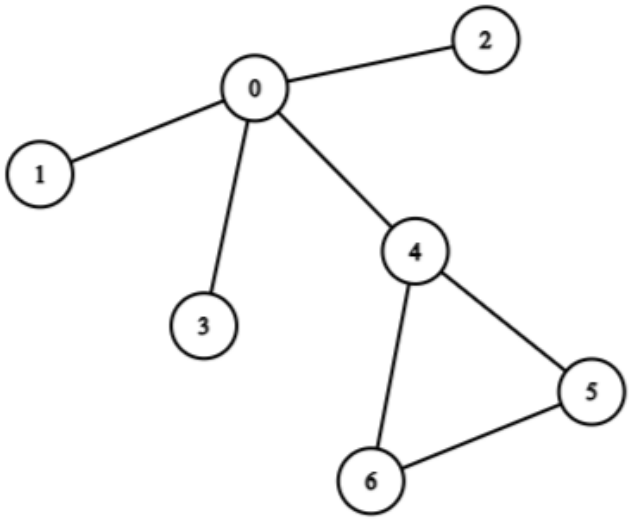
Definimos $coNP$ como la clase de lenguajes cuyo complemento está en NP.

$$coNP = \{L^c | L \in NP\}$$

1. Demuestra que si $coNP \neq NP$ entonces $P \neq NP$.
   Para esto usaremos la contrapositiva de $coNP \neq NP$ entonces $P \neq NP$, la cual seria si $P = NP$ entonces $coNP = NP$.
   Supongamos que $P = NP$, ahora veamos que se cumple $coNP = NP$.
   Sabemos que $P \subseteq NP$ por definición de Py NP y por lo visto en clase.
   También sabemos que P esta cerrado bajo completo, por la tarea 4, eso quiere decir que si tenemos un lenguaje $L \in P$ entonces el complemento de L, digamos L', esta en P, es decir $L' \in P$.
   Ahora, como supusimos que $P = NP$, entonces tenemos que si un lenguaje $L \in P$ entonces $L \in NP$ y de igual manera si $L \in NP$ entonces $L \in P$.
   Por lo tanto sea un lenguaje H cualquiera en NP, $H \in NP$, entonces ya que supusimos que $P = NP$ sabemos que $H \in P$ y por lo tanto también $H' \in P$, con H' el complemento de H, ya que P esta cerrado bajo complemento. Ahora notemos que como $H' \in P$ entonces $H' \in NP$, ya que $P = NP$, entonces como $H' \in NP$ y $H \in NP$ tenemos que $H \in coNP$, ya que el complemento de H, que es H', esta en NP, entonces tenemos que $NP \subseteq coNP$.
   Ahora sea un lenguaje J cualquiera en $coNP$, $J \in coNP$, entonces por definición sabemos que el complemento de J esta en NP, digamos que el complemento es J', $J' \in NP$, entonces tenemos que $J' \in P$ ya que $P = NP$, luego sabemos que $J \in P$ ya que P esta cerrado bajo complemento y por ultimo $J \in NP$ ya que $P = NP$. Por lo tanto llegamos a que $J \in NP$ y por lo tanto tenemos que $coNP \subseteq NP$.
   Notemos que tenemos $NP \subseteq coNP$ y $coNP \subseteq NP$, lo cual significa que $NP = coNP$, entonces logramos obtener que si $P = NP$ entonces $coNP = NP$, y como es la contrapositiva de si $coNP \neq NP$ entonces $P \neq NP$, entonces podemos concluir que si $coNP \neq NP$ entonces $P \neq NP$ se cumple.

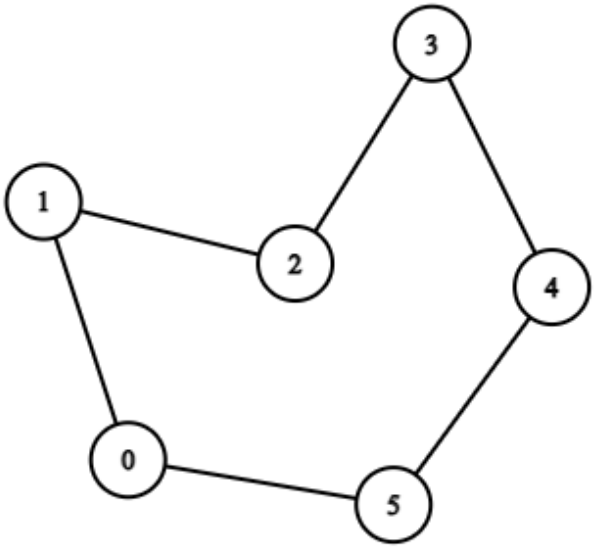
2. Define el complemento del problema $CHND$, definido en el Ejercicio 3. Da un ejemplar que pertenezca al complemento de dicho problema y otro que no pertenezca.

   - Definiremos al complemento del problema $CHND$ así
     Problema: $\overline{CHND}$
     Ejemplar: Una gráfica no dirigida
     Pregunta: ¿Es verdad que no existe un ciclo no dirigido que pase por todos los vértices exactamente una vez?
   - Ejemplares
     - Ejemplar que pertenece a $\overline{CHND}$



Esto debido a que la gráfica que es el ejemplar no tiene ciclos no dirigidos Hamiltonianos (o ciclos no dirigidos que pasen por todos los vértices exactamente una vez), esto ya que para llegar a los vértices 1, 2 y 3 solo tienen de vecino a el vértice 0.

- Ejemplar que no pertenece a $\overline{CHND}$



Esto debido a que la gráfica que es el ejemplar si tiene un ciclo no dirigido Hamiltoniano (o un ciclo no dirigido que pase por todos los vértices exactamente una vez), esto ya que el ciclo generado por los vértices $0, 1, 2, 3, 4, 5, 0$ genera un ciclo Hamiltoniano.