Teórica 9: Teoría de complejidad computacional - 1era parte

1. Introducción

Al comienzo de la materia, vimos cómo medir la complejidad de un algoritmo en función de la medida de la instancia, representando a ésta en una forma razonable, como puede ser en representación binaria. Y llamamos polinomial a un algoritmo que al ser ejecutado requiere una cantidad de tiempo que es una función polinomial en la medida de su entrada. Luego, estudiamos algoritmos polinomiales para algunos problemas sobre grafos. Para otros problemas, mencionamos que no se conocen algoritmos polinomiales para resolverlos.

Una idea intuitiva es que los algoritmos polinomiales son útiles en la práctica, mientras que los exponenciales no lo son. Sin embargo, ésto puede no ser verdad si el grado del polinomio es alto. Por otro lado, hay algoritmos cuyo peor caso es exponencial, pero en la práctica es muy raro que aparezca una instancia que sea peor caso y para las instancias que se presentan funciona muy bien, aun mejor que un algoritmo polinomial. Pero en este tema, casi siempre esta intuición funciona, y Edmonds en 1965 propuso la siguiente definición:

Definición 1. Un ***algoritmo eficiente*** es un algoritmo de complejidad polinomial.

El objetivo de esta última clase es analizar la dificultad inherente del ***problema*** que queremos resolver, presentando una introducción a la teoría de la complejidad computacional.

Definición 2. Un problema está ***bien resuelto*** si se conocen algoritmos eficientes para resolverlo.

Dada una instancia I de un problema de optimización Π con función objetivo f, se pueden presentar distintas versiones del problema:

- Versión de ***optimización***: Encontrar una ***solución óptima*** del problema Π para I (de valor mínimo o máximo).
- Versión de ***evaluación***: Determinar el ***valor*** de una solución óptima de Π para I.
- Versión de ***localización***: Dado un número k, determinar una ***solución factible*** de Π para I tal que $f(S) \leq k$ si el problema es de minimización (o $f(S) \geq k$ si el problema es de maximización).
- Versión de ***decisión***: Dado un número k, ¿existe una solución factible de Π para I tal que $f(S) \leq k$ si el problema es de minimización (o $f(S) \geq k$ si el problema es de maximización)?

Ejemplo 1. *TSP: Dado un grafo G con longitudes asignadas a sus aristas, las distintas versiones son:*

- *Versión de **optimización**: Determinar un circuito hamiltoniano de G de longitud mínima.*
- *Versión de **evaluación**: Determinar el valor de una solución óptima, o sea la longitud de un circuito hamiltoniano de G de longitud mínima.*
- *Versión de **localización**: Dado un número $k \in \mathbb{Z}_{\geq 0}$, determinar un circuito hamiltoniano de G de longitud menor o igual a k.*
- *Versión de **decisión**: Dado un número $k \in \mathbb{Z}_{\geq 0}$, ¿existe un circuito hamiltoniano de G de longitud menor o igual a k?*

Hay cierta relación entre la dificultad de resolver las distintas versiones de un mismo problema. En general, la versión optimización es computacionalmente la más difícil, ya que usualmente es fácil (eficiente) evaluar la función objetivo para una solución factible.

Por otro lado, la versión de decisión suele ser la ***más fácil***. Dada la respuesta de cualquiera de las otras versiones es fácil (eficiente) calcular la respuesta de esta versión.

Además, para los problemas de optimización que cumplen que el valor de una solución óptima es un entero K cuyo logaritmo está acotado por un polinomio en la medida de la entrada, la versión de evaluación puede ser resuelta ejecutando una cantidad polinomial de veces la versión de decisión, realizando búsqueda binaria sobre el parámetro k. Muchos problemas cumplen esta propiedad.

Para varios problemas, si resolvemos el problema de decisión, podemos encontrar la solución de la versión de localización resolviendo una cantidad polinomial de problemas de decisión para instancias reducidas de la instancia original. La misma relación ocurre entre las versiones de evaluación y optimización.

Ejemplo 2. *TSP: Supongamos que tenemos un grafo G y un entero $k \in \mathbb{Z}_{\geq 0}$ tales que la respuesta de la versión de decisión es **SI** y queremos resolver la versión de localización.*

Para esto seleccionamos una arista del grafo y resolvemos la versión de decisión colocando peso infinito a esa arista:

- *Si la respuesta sigue siendo **SI** significa que G tiene un circuito hamiltoniano de peso menor o igual a k que no utiliza esa arista. Dejamos el peso de esa arista en infinito.*
- *Si la respuesta pasa a ser **NO** significa que esa arista pertenece a todo circuito hamiltoniano de peso menor o igual a k. Volvemos a poner el peso original de esa arista.*

Realizamos este proceso sobre todas las aristas. Al finalizar, las aristas que quedan con peso distinto de infinito formarán un ciclo hamiltoniano de peso menor o igual a k.

Para muchos problemas de optimización combinatoria las cuatro versiones son equivalentes en el sentido que si existe un algoritmo eficiente para una de ellas, entonces existe para todas.

La clasificación y el estudio de los problemas dentro de esta teoría se realiza sobre problemas de decisión.

Definición 3. Un ***problemas de decisión*** es un problema cuya respuesta es ***SI*** o ***NO***.

Por un lado, ésto permite uniformar el estudio, ya que hay problemas, como saber si un grafo es hamiltoniano o satisfabilidad, que no tienen versión de optimización. Por otro lado, un resultado negativo para la versión de decisión, por lo que vimos anteriormente, se traslada a un resultado negativo a la versión de optimización.

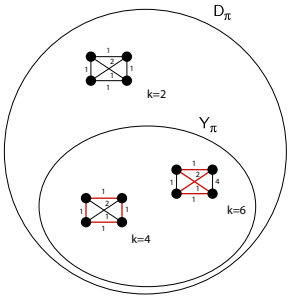
Hasta ahora estuvimos trabajando con instancias de un problema y nos entendimos, pero ahora definiremos formalmente este concepto.

Definición 4.

- Una ***instancia*** de un problema es una especificación de sus parámetros.
- Un problema de decisión Π tiene asociado:

- un conjunto D_Π de instancias
- y un subconjunto $Y_\Pi \subseteq D_\Pi$ de instancias cuya respuesta es SI.

Ejemplo 3. *TSP: Dado un grafo completo con peso en las aristas y un número $k \in \mathbb{Z}_{\geq 0}$, ¿existe un circuito Hamiltoniano de longitud a lo sumo k?*



Nos interesa reconocer los problemas computacionalmente difíciles.

Definición 5. Un problema es ***intratable*** si no puede ser resuelto por algún algoritmo eficiente.

Un problema puede ser intratable por distintos motivos:

- El problema requiere una repuesta de longitud exponencial (ejemplo: pedir todos los circuitos hamiltonianos de longitud a lo sumo k).
- El problema es ***indecidible*** (ejemplo: problema de la parada).
- El problema es decidible pero no se conocen algoritmos polinomiales que lo resuelvan (no se sabe si es intratable o no).

2. Problema de satisfabilidad (SAT)

Una variable booleana o lógica es una variable que puede tomar valor verdadero (V) o falso (F). Un literal es una variable booleana o su negación y una expresión o fórmula booleana está formada por variables, conjunciones, disyunciones y negaciones. Una cláusula es una disyunción de literales y una fórmula está en forma normal conjuntiva (FNC) si es una conjunción de cláusulas. Toda fórmula lógica puede ser expresada en FNC.

Una fórmula se dice satisfacible si existe una asignación de valores de verdad a sus variables tal que la haga verdadera.

Ejemplo 4. *Dados $X = \{x_1, x_2, x_3\}$ un conjunto de variables booleanas y las fórmulas en FNC:*

- $\varphi_1 = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_3)$ *es satisfacible, ya que asignando los valores de verdad $x_1 = x_2 = V$ y $x_3 = F$, φ_1 es verdadera.*
- $\varphi_2 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2) \wedge (\neg x_3)$ *no es satisfacible, porque no existe asignación de valores de verdad a las variables de X que hagan a φ_2 verdadera.*

Veremos que SAT tiene un rol central en el estudio de la teoría de la complejidad computacional. Su definión es la siguiente.

Definición 6. ***Problema de satisfabilidad (SAT):*** Dado un conjunto de cláusulas $C_1, \ldots, C_m$ formadas por literales basados en las variables booleanas $X = \{x_1, \ldots, x_n\}$, ¿la expresión $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ es satisfacible?

3. Modelos de cómputo

Los modelos de cómputo son modelos abstractos para expresar cualquier algoritmo. Describen formalmente cómo se calcula la salida en función de la entrada. Como lo hicimos en la primera clase, un modelo de cómputo brinda el marco formal, independiente de la implementación particular, para calcular la complejidad de un algoritmo. Dos de los más utilizados son:

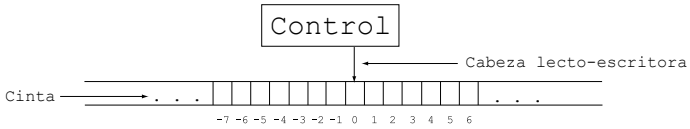
- Máquina de Turing (1937, Alan Turing)
- Máquinas de Acceso Random - RAM (1974, Aho, Hopcroft y Ullman).

Estos dos modelos son polinomialmente equivalentes. Es decir, se puede simular uno a otro con un costo polinomial. Por practicidad e historia, al comienzo de la materia utilizamos las máquinas RAM para formalizar el cálculo de la complejidad de un algoritmo y ahora utilizaremos las máquinas de Turing para estudiar una introducción a la Teoría de la complejidad computacional.

3.1. Máquina de Turing Determinística

Una ***máquina de Turing*** consiste de:

- Una cabeza lecto-escritora.
- Una cinta infinita con el siguiente esquema:



- La cabeza lectora se puede mover en ambas direcciones: $M = \{+1, -1\}$ son los movimientos de la cabeza a derecha $(+1)$ o izquierda (-1).
- Un conjunto finito de estados, Q.
- Hay una celda distinguida, la celda 0, que define la celda incial.
- Un alfabeto finito Σ y un símbolo especial $*$ que indica ***blanco***, definiendo $\Gamma = \Sigma \cup \{*\}$.
- Sobre la cinta está escrita la entrada, que es un string de símbolos de Σ y el resto de las celdas tiene $*$ (blancos).
- Hay un estado distinguido, $q_0 \in Q$, que define el estado inicial.
- Y un conjunto de estados finales, $Q_f \subseteq Q$ (q_{si} y q_{no} para problemas de decisión)

Definición 7.

- Una ***instrucción*** en una MT es una quíntupla $S \subseteq Q \times \Gamma \times Q \times \Gamma \times M$. La quíntupla $(q_i, s_h, q_j, s_k, +1)$ se interpreta como:

Si la máquina está en el estado q_i y la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

- Se define un ***programa*** en una MT como un conjunto finito de instrucciones.
- Una MT es ***determinística***, MTD, si para todo par (q_i, s_h) existe en el programa a lo sumo una quíntupla que comienza con ese par.

El funcionamiento de una MT es el siguiente:

- Arranque
  - máquina posicionada en el estado distinguido q_0, estado inicial
  - cabeza lectora-escritora ubicada en la celda inicial (0) de la cinta.
- Transiciones
  - si la máquina está en el estado q_i y lee en la cinta s_h, busca en la tabla de instrucciones un quíntupla que comience con el par (q_i, s_j).
  - si existe esta quíntupla, (q_i, s_h, q_j, s_k, m), la máquina pasa al estado q_j, escribe en la cinta el símbolo s_k y se mueve a derecha o izquieda según el valor de m.
- Terminación
  - cuando no se puede inferir nuevas acciones para seguir: no existe la quíntupla que comienza con (q_i, s_h)
  - cuando se alcanza un estado final: si el estado final es de SI, entonces la respuesta es ***SI***, caso contrario la respuesta es ***NO***.

Definición 8.

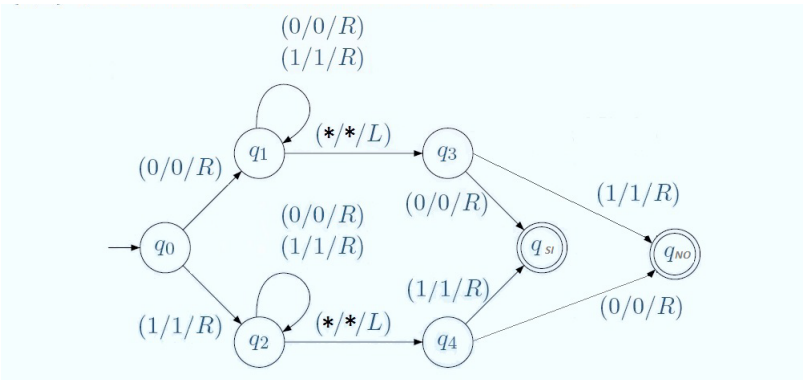
- Una máquina M resuelve el problema Π si para toda instancia alcanza un estado final y responde de forma correcta (o sea, termina en un estado final correcto).
- La complejidad de una MTD está dada por la cantidad de movimientos de la cabeza, desde el estado inicial hasta alcanzar un estado final, en función del tamaño de la entrada.

$$T_M(n) = \text{máx}\{m \text{ tq } x \in D_\Pi, |x| = n \text{ y } M \text{ con entrada } x \text{ hace } m \text{ movimientos}\}$$

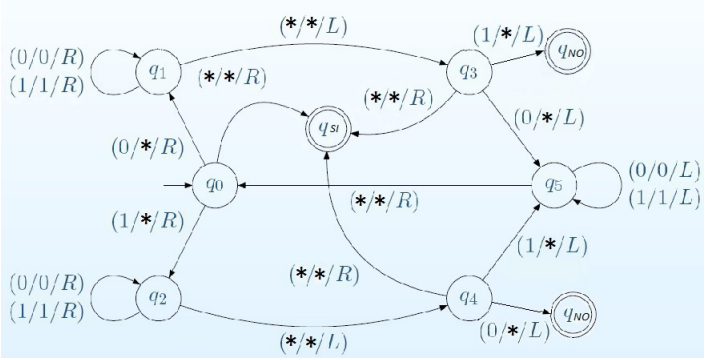
Existen otros modelos de computadoras determinísticas (máquina de Turing con varias cintas, Random Access Machines, etc.) pero puede probarse que son equivalentes en términos de la polinomialidad de los problemas a la MTD.

Una forma útil de expresar una MD es mediante un gráfico, donde los estados se representan mediante nodos y las transiciones mediante los enlaces. Veamos dos ejempplos.

Ejemplo 5. *Dada una palabra sobre* $\{0, 1\}$*, ¿comienza y termina con el mismo símbolo?*



Ejemplo 6. *Dada una palabra sobre* $\{0, 1\}$*, ¿es palíndromo?*



3.2. Máquinas de Turing no-determinísticas (MTND)

Una ***máquina de Turing no-determinística***, MTND, es una generalización de una MTD. Tiene los mismos componentes que vimos para una máquina de Turing determinística, pero no se pide unicidad de la quíntupla que comienza con cualquier par (q_i, s_j).

Las instrucciones dejan de ser quíntuplas para pasar a ser un mapeo multivaluado. Un programa correspondiente en una MTND es una tabla que mapea un par (q_i, t_i) a un ***conjunto*** de ternas $(q_f, t_f, \{0, +1, -1\})$.

Una MTND permite la ejecución en paralelo de las distintas alternativas. Cuando una MTND llega a un punto en su ejecución donde son posibles múltiples alternativas, la MTND examina todas las alternativas en paralelo, en lugar de hacerlo secuencialmente como una MTD. Podemos pensarlo como que se abre en k copias cuando se encuentra con k alternativas. Cada copia continua su ejecución independientemente. Si una copia acepta la entrada, todas las copias paran.

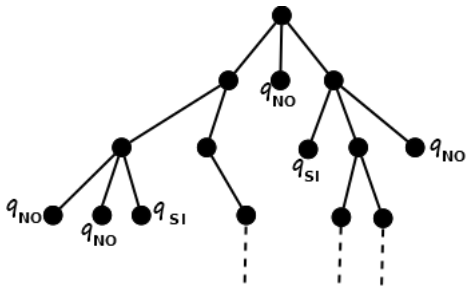
Una interpretación equivalente es que cuando se presentan múltiples alternativas, la MTND selecciona la ***alternativa correcta***.

Definición 9. Una MTND resuelve el problema de decisión Π si:

- existe una secuencia de alternativas que lleva a un estado de respuesta ***SI*** si, y sólo si, la respuesta es ***SI***, o bien
- alguna de las copias se detiene en un estado de respuesta ***SI*** si, y sólo si, la respuesta es ***SI***.

Ésto es equivalente a: Para toda instancia de Y_Π existe una rama que llega a un estado final q_{si} y para toda instancia en $D_\Pi \setminus Y_\Pi$ ninguna rama llega a un estado final q_{si}.

Podemos interpretar la ejecución de una MTND como un árbol de alternativas. Para una instancia de Y_Π será:



Definición 10.

- La complejidad temporal de una MTND M se define como el máximo número de pasos que toma como mínimo reconocer una instancia de Y_Π en función de su tamaño:

$$T_M(n) = \text{máx}\{m \text{ tq } x \in Y_\Pi, |x| = n \text{ y alguna de las ramas de } M \text{ termina en estado } q_{si} \text{ en } m \text{ movimientos cuando la entrada es } x\}$$

- Una MTND M es ***polinomial*** para Π cuando $T_M(n)$ es un función polinomial.

4. Las clases P y NP

Definición 11. Un problema de decisión Π pertenece a la clase **P** (polinomial) si existe una MTD de complejidad polinomial que lo resuelve:

$$\mathbf{P} = \{\Pi \text{ tq } \exists M \text{ MTD tq } M \text{ resuelve } \Pi \text{ y } T_M(n) \in O(p(n)) \text{ para algún polinomio } p\}$$

Ésto es equivalente a: existe un algoritmo polinomial que lo resuelve.

Ejemplos de problemas en **P** (en su versión de decisión):

- Grafo conexo.
- Grafo bipartito.
- Árbol generador mínimo.
- Camino mínimo entre un par de nodos.
- Matching máximo.
- Grafo euleriano.

Una clase más general de problemas de decisión es la siguiente:

Definición 12. Un problema de decisión Π pertenece a la clase **NP** (polinomial no-determinístico) si las instancias de Π con respuesta ***SI*** son reconocidas por una MTND polinomial.

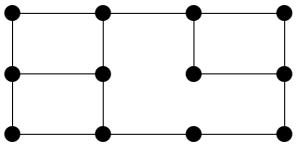
Equivalentemente, dada una instancia de Π con respuesta ***SI*** se puede dar un certificado de longitud polinomial que garantiza que la respuesta es ***SI***, y esta garantía puede ser verificada en tiempo polinomial.

Esta clase es de interés porque incluye a la gran mayoría de los problemas de optimización combinatoria en su versión decisión.

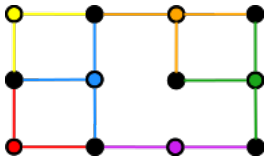
4.1. Conjunto independiente y recubrimiento de aristas

Analicemos algunas situaciones:

Ejemplo 7. *El grafo de la figura representa el mapa de una ciudad. Se quiere ubicar policías en las esquinas de modo que todas las cuadras estén bajo vigilancia, o sea, cada cuadra tiene que tener un policía al menos en una de las esquinas. ¿Cuál es el mínimo número de policías necesarios?*



Si ubicamos un policía en cada uno de los vértices coloreados, todas las cuadras estarán cubiertas.

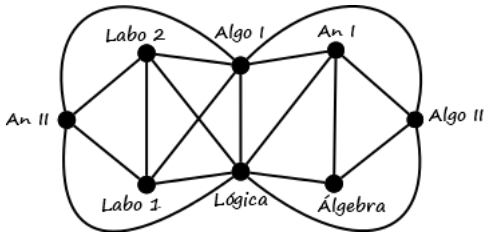


En este caso estamos necesitando 6 policías. Pero, ¿habrá forma de hacerlo con 5?

Ahora nuestro objetivo es encontrar un conjunto de vértices de cardinal mínimo, tal que toda arista del grafo incida, al menos, sobre un vértice de este conjunto.

Volvamos al problema de las aulas de la clase de coloreo, pero con un objetivo distinto.

Ejemplo 8. *Supongamos que no tenemos restricción en cantidad de aulas y que el DC equipó una sala (sólo una por ahora) con la última tecnología y quiere que sea utilizada por la mayor cantidad de materias posible. Volvamos a dibujar el grafo que representa nuestra instancia:*



Si le asignamos la sala a la materia Lógica, ninguna otra materia podrá usarla, porque Lógica se solapa en horario con todas las demás.

En cambio, si le asignamos la sala a Álgebra, también la podría utilizar por ejemplo Algo I (o Labo 1, o Labo 2, o Análisis II), pero no Algo II ni Análisis I ni Lógica. En este caso, la sala sería utilizada por dos materias. Pero, ¿será posible que la sala sea utilizada por tres o más materias?
Lo que queremos hallar en este ejemplo es un conjunto de vértices de cardinal máximo tal que no haya dos vértices adyacentes dentro del conjunto.

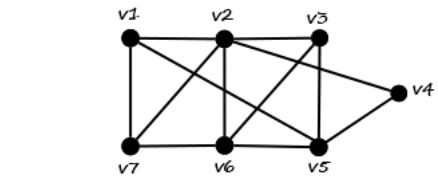
Este tipo de escenarios dan origen a las siguientes definiciones y estudios.

Definición 13. Dado un grafo $G = (V, X)$

- Un ***conjunto independiente*** de vértices de G, es un conjunto de vértices $I \subseteq V$ tal que para toda arista $e \in X$, e es incidente a lo sumo a un vértice $v \in I$.

- Un ***recubrimiento de las aristas*** de G, es un conjunto $R_a \subseteq V$ de vértices tal que para todo $e \in X$, e es incidente al menos a un vértice $v \in R_n$.

Ejemplo 9.



M = {(v1,v7),(v2,v6),(v3,v5)} I = {v1,v6,v4}

Ra = {v1,v2,v5,v6} Rv = {(v1,v7),(v2,v6),(v3,v5),(v4,v5)}

Los problemas de optimización relacionados a estos conceptos son:

Definición 14. Dado un grafo $G = (V, X)$

- El problema de ***conjunto independiente máximo*** consiste en encontrar un conjunto independiente de cardinal máximo entre todos los conjuntos independientes de G.
- El problema de ***recubrimiento de aristas mínimo*** consiste en encontrar un recubrimiento de aristas de cardinal mínimo entre todos los recubrimientos de aristas de G.

Aunque conjunto independiente y matching parezcan problemas muy parecidos, unos sobre vértices y otros sobre aristas, computacionalmente son muy distintos. Para el problema de matching máximo existen algoritmos polinomiales para resolverlo, mientras que para conjunto independiente máximo no se conoce ninguno (y se piensa que no existe).

Podemos enunciar una relación interesante entre dos de estos conceptos:

Lema 1. *Sea $G = (V, X)$ y $S \subseteq V$. S es un conjunto independiente $\iff V \setminus S$ es un recubrimiento de aristas.*

Demostración. $\implies$) Por el absurdo, supongamos que S es un conjunto independiente y $V \setminus S$ no es un recubrimiento de aristas. Entonces existe una arista $(u, v) \in X$ no recubierta, es decir, $u \notin V \setminus S$ y $v \notin V \setminus S$. Ésto implica que $u \in S$ y $v \in S$. Esto contradice que S es conjunto independiente, ya que u y v son adyacentes.

$\impliedby$) Ahora, también por el absurdo, supongamos que $V\S$ es un recubrimiento de aristas y S no es un conjunto independiente. Entonces existen dos vértices u y v en S adyacentes. Ésto implica que $u \notin V \setminus S$ y $v \notin V \setminus S$. Pero entonces la arista (u, v) no está cubierta por $V \setminus S$, contradiciendo que es un recubrimiento de aristas.

■

Corolario 1. *Sea $G = (V, X)$ y $S \subseteq V$. $S \subseteq V$ es un conjunto independiente máximo $\iff V \setminus S$ es un recubrimiento de aristas mínimo.*

Este Lema implica que para el problema de aristas mínimo tampoco se conocen algoritmos polinomiales.

Ejemplo 10. *Dado un grafo $G = (V, X)$, un **conjunto independiente** de vértices de G, es un conjunto de vértices $I \subseteq V$ tal que para toda arista $e \in X$, e es incidente a lo sumo a un vértice $v \in I$. El problema de conjunto independiente máximo en su versión de decisión es* **NP**.

CONJUNTO INDEPENDIENTE: Dados un grafo $G = (V, X)$ y $k \in \mathbb{N}$, ¿G tiene un conjunto independiente de tamaño mayor o igual a k?

*Para una instancia con respuesta **SI**, podemos exponer $S \subseteq V$ conjunto independiente de G tal que $|S| \geq k$. Es posible chequear polinomialmente que S cumple estas dos propiedades: ser conjunto independiente de G y tener cardinal mayor o igual a k.*

Esto demuestra que CONJUNTO INDEPENDIENTE pertenece a la clase **NP**.

4.2. Más ejemplos de problemas NP

Ejemplo 11. *SAT es* **NP**:

*Una instancia de **SI** de SAT es un conjunto de cláusulas $C_1, \ldots, C_m$ satisfacible. Ésto es, existe una asignación de valores de verdad a las variables booleanas intervinientes en las cláusulas que hacen verdadero el valor de verdad de la expresión $C_1 \wedge C_2 \wedge \ldots \wedge C_m$.*

El certificado que podemos mostrar es una asignación de valores de verdad a las variables que haga verdadera a la expresión $C_1 \wedge C_2 \wedge \ldots \wedge C_m$. Como es posible verificar en tiempo polinomial que esta expresión es verdad con esos valores de las variables, demuestra que $SAT \in$ **NP**.

Ejemplo 12. *El problema de clique máxima en su versión de decisión es* **NP**.

CLIQUE: Dado un grafo $G = (V, X)$ y $k \in \mathbb{N}$, ¿G tiene una clique de tamaño mayor o igual a k?

*Dada una instancia de **SI**, ésto es un grafo $G = (V, X)$ que tiene una clique de tamaño mayor o igual a k, podemos verificar polinomialmente que un conjunto de vértices C cumple que:*

- *$C \subseteq V$*
- *C induce una clique en G*
- *$|C| \geq k$*

La existencia de este algoritmo polinomial demuestra que $CLIQUE \in$ **NP**.

Ejemplo 13. *CIRCUITO HAMILTONAINO está en* **NP**:

La evidencia que soporta una respuesta positiva es un ciclo hamiltoniano de G. Dado un lista de vértices, se puede chequear polinomialmente si define un ciclo hamiltoniano.

Ejemplo 14. *TSP está en* **NP**:

Dada una instancia de TSP, G y $k \in \mathbb{Z}_{\geq 0}$, la evidencia que soporta una respuesta positiva es un ciclo hamiltoniano de G con peso menor o igual a k. Dada una lista de vértices, se puede chequear polinomialmente si define un ciclo hamiltoniano y que la suma de los pesos de las aristas respectivas es menor o igual a k.

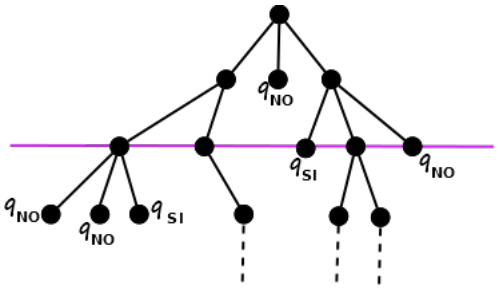
4.3. Resultados sobre la clase NP

Un resultado interesante sobre los problemas de la clase **NP** es el siguiente:

Lema 2. *Si* Π *es un problema de decisión que pertence a la clase* **NP**, *entonces* Π *puede ser resuelto por un algoritmo determinístico en tiempo exponencial respecto del tamaño de la entrada.*

Demostración. Como $\Pi \in$ **NP**, existe una MTND polinomial M que lo resuelve. Sea $T_M(n)$ el polinomio que define la complejidad de esta MTND.

Para resolver Π de forma determinística secuencial podemos recorrer cada rama del árbol de ejecución de M hasta profundidad $T_M(n)$ o llegar a un estado final, lo que ocurra primero. Si en alguna rama se llega a un estado q_{SI} la respuesta será ***SI***, caso contrario será ***NO***. Es decir, es posible cortar el árbol de ejecución a altura $T_M(n)$ para obtener la respuesta correcta.



La cantidad de alternativas en las que se puede abrir la MTND está acotada, ya que la cantidad de ternas está acotada por ser el alfabeto y la cantidad de estados finitos. Si c es una cota de la cantidad de ternas, en el nivel r del árbol de ejecución, habrá, a lo sumo, c^r copias en ejecución paralelamente.

Por lo tanto, la implementación secuencial del algoritmo recorrerá, a lo sumo, $\sum_{i=0}^{T_M(n)} c^i$ nodos (cantidad de movimientos).

■

Los primeros hechos que podemos observar sobre la relación entre estas clases son:

- **P** $\subseteq$ **NP**: Porque una MTD es un caso particular de MTND.
- ***Problema abierto:*** ¿Es **P** = **NP**?
  - Todavía no se demostró que exista un problema en NP\P.
  - Mientras tanto, se estudian clases de complejidad ***relativa***, es decir, que establecen orden de dificultad entre problemas.

5. Transformaciones polinomiales

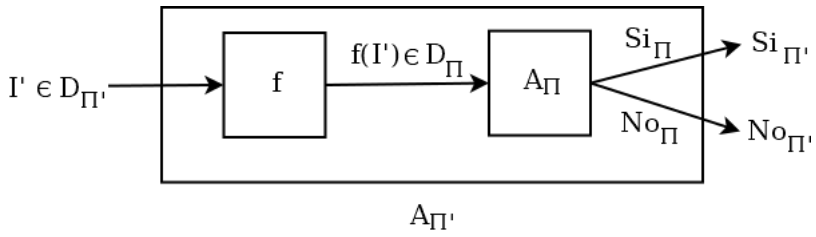
Imaginemos que tenemos un algoritmo A_Π para resolver un problema Π y ahora queremos resolver otro problema Π'. ¿Habrá forma de que podamos aprovechar A_Π?

Para ésto, deberíamos ser capaces de, dada una instancia I' del problema Π', poder transformarla en una instancia I del problema Π, tal que la respuesta para I sea ***SI*** si, y sólo si, la respuesta para I' es ***SI***.

Lo que necesitamos es una transformación $f : D_{\Pi'} \to D_{\Pi}$, tal que

$$I' \in Y_{\Pi'} \iff f(I') \in Y_{\Pi}.$$

Entonces, un algoritmo posible para resolver Π' podría ser $A_{\Pi'}(I') = A_{\Pi}(f(I'))$, donde I' es una instancia de Π'.



Si la función f y el algoritmo A_{Π} son polinomiales, el algoritmo $A_{\Pi'}$ resultará polinomial.

Definamos esta transformación formalmente:

Definición 15.

- Una ***transformación o reducción polinomial*** de un problema de decisión Π' a uno Π es una función polinomial que transforma una instancia I' de Π' en una instancia I de Π tal que I' tiene respuesta ***SI*** para Π' si, y sólo si, I tiene respuesta ***SI*** para Π:

$$I' \in Y_{\Pi'} \iff f(I') \in Y_{\Pi}$$

- El problema de decisión Π' se ***reduce polinomialmente*** a otro problema de decisión Π, $\Pi' \leq_p \Pi$, si existe una transformación polinomial de Π' a Π.

Proposición 1. *La reducción polinomial es una relación transitiva: Si P_1, P_2 y P_3 son problemas de decisión tales que $\Pi_1 \leq_p \Pi_2$ y $\Pi_2 \leq_p \Pi_3$, entonces $\Pi_1 \leq_p \Pi_3$.*

Demostración. Sea $f : D_{\Pi_1} \to D_{\Pi_2}$ una reducción polinomial de P_1 a P_2 y $g : D_{\Pi_2} \to D_{\Pi_3}$ de P_2 a P_3.

Entonces

$$I_1 \in Y_{\Pi_1} \iff f(I_1) \in Y_{\Pi_2} \text{ y } I_2 \in Y_{\Pi_2} \iff g(I_2) \in Y_{\Pi_3}.$$

Por lo tanto, para toda $I_1 \in D_{\Pi_1}$ obtenemos que:

$$I_1 \in Y_{\Pi_1} \iff g(f(I_1)) \in Y_{\Pi_3}.$$

Además, la composición de dos funciones polinomiales resulta en una función polinomial. Resultando que $\Pi_1 \leq_p \Pi_2$. ■

6. Bibliografía recomendada

- M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of NP- Completeness*, W. Freeman and Co., 1979.
- Capítulos 15 y 16 de C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications INC, 1998.

Referencias

[1] C. H. Papadimitriou. On the complexity of edge traversing. *J. ACM*, 23(3):544–554, July 1976.