Complejidad Temporal

Máquinas de Turing No Determinísticas (MTN)

Una MTN es una Máquina de Turing que para un estado y un símbolo barrido por el cabezal, tiene un conjunto finito de alternativas para el siguiente movimiento. La MTN acepta su entrada si cualquier sucesión de alternativas de movimiento se detiene en q_A

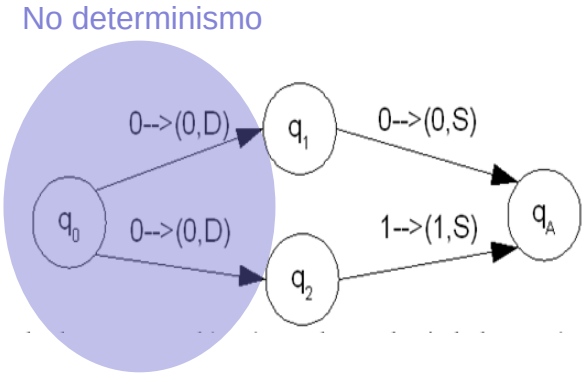
$M=<Q, \Sigma, \Gamma, \Delta, q_0, q_A, q_R>$, $\Delta: Q \times \Gamma \rightarrow \rho((Q \cup \{q_A, q_R\}) \times \Gamma \times \{D, I, S\})$

M acepta su entrada *w*, sii existe al menos una computación a partir de *w* alcanza el estado q_A

Nota: Para referirnos a las máquinas de Turing determinísticas utilizaremos MTD. Obsérvese que una MTD es un caso particular de una MTN donde el conjunto de alternativas es siempre un conjunto unitario

Máquinas de Turing No Determinísticas (MTN)

Ejemplo: Construir una MTN M tal que
L(M)={$w \in$ {0,1,2,3}* / w comienza con 00 ó 01 }



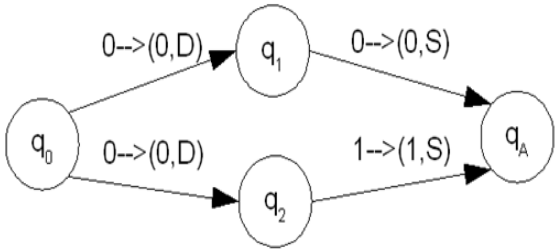
En el gráfico faltan todas las otras combinaciones de estado símbolo que tienen que llevar al estado q_R.

¿Donde se observa el no determinismo en el grafo de la función de transición?

Claramente hay dos posibilidades de movimiento para el mismo caso (q_0,0)

Máquinas de Turing No Determinísticas (MTN)

Ejemplo: Construir una MTN M tal que
L(M)={$w \in$ {0,1,2,3}* / w comienza con 00 ó 01 }



En el gráfico faltan todas las otras combinaciones de estado símbolo que tienen que llevar al estado q_R.

$\Delta(q_0,0) = \{(q_1,0,D),(q_2,0,D)\}$ ← No determinismo

$\Delta(q_0,x) = \{(q_R,x,S)\}$ para todo $x \in \Gamma - \{0\}$

$\Delta(q_1,0) = \{(q_A,0,S)\}$

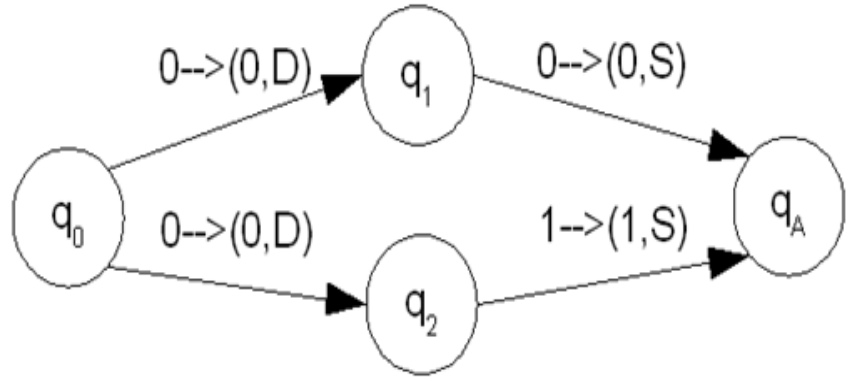
$\Delta(q_1,x) = \{(q_R,x,S)\}$ para todo $x \in \Gamma - \{0\}$

$\Delta(q_2,1) = \{(q_A,1,S)\}$

$\Delta(q_2,x) = \{(q_R,x,S)\}$ para todo $x \in \Gamma - \{1\}$

Máquinas de Turing No Determinísticas (MTN)

Traza de computación. Para una MTN la traza tendrá forma de árbol dónde cada rama representa una computación (una sucesión de alternativas de movimiento)



traza para w= 021

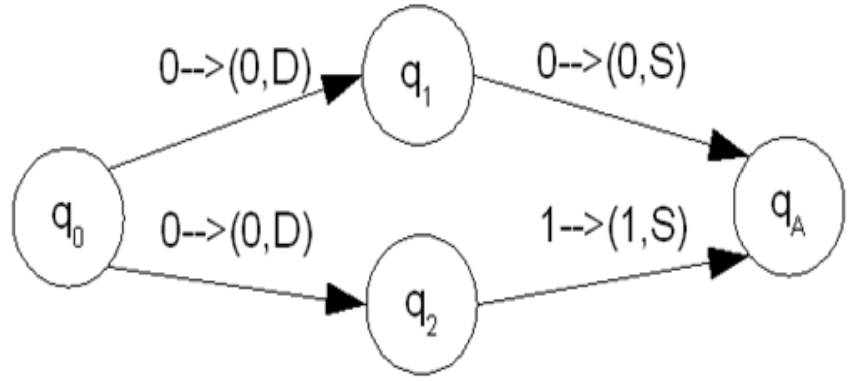
$q_0021 \vdash 0q_121 \vdash 0q_R21$

$0q_221 \vdash 0q_R21$

M rechaza a 021

Máquinas de Turing No Determinísticas (MTN)

Traza de computación. Para una MTN la traza tendrá forma de árbol dónde cada rama representa una computación (una sucesión de alternativas de movimiento)

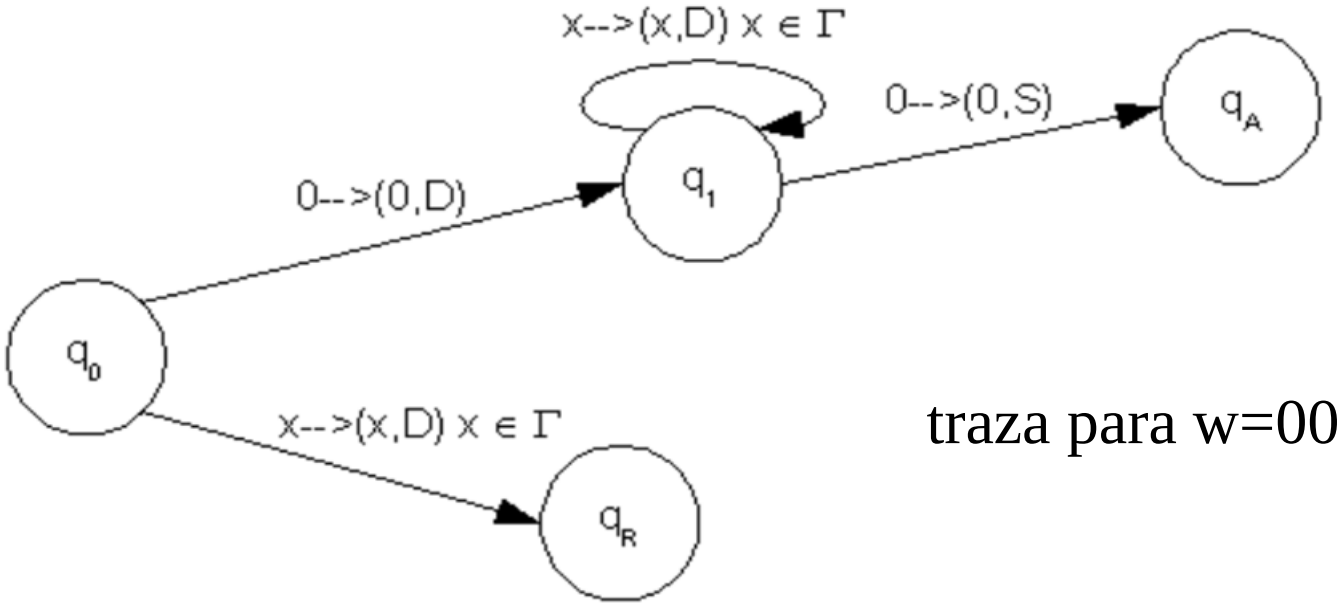


traza para w=0054

$q_0 0054 \vdash 0q_1 054 \vdash 0q_A 054$ ✓

M acepta a 0054

$0q_2 054 \vdash 0q_R 054$



traza para w=0010

$q_0$0010 ⊢ 0$q_1$010 ⊢ 00$q_1$10 ⊢ 001$q_1$0 ⊢ 0010q_1B ⊢ · · ·

0$q_1$010 ⊢ 0q_A010 ✓

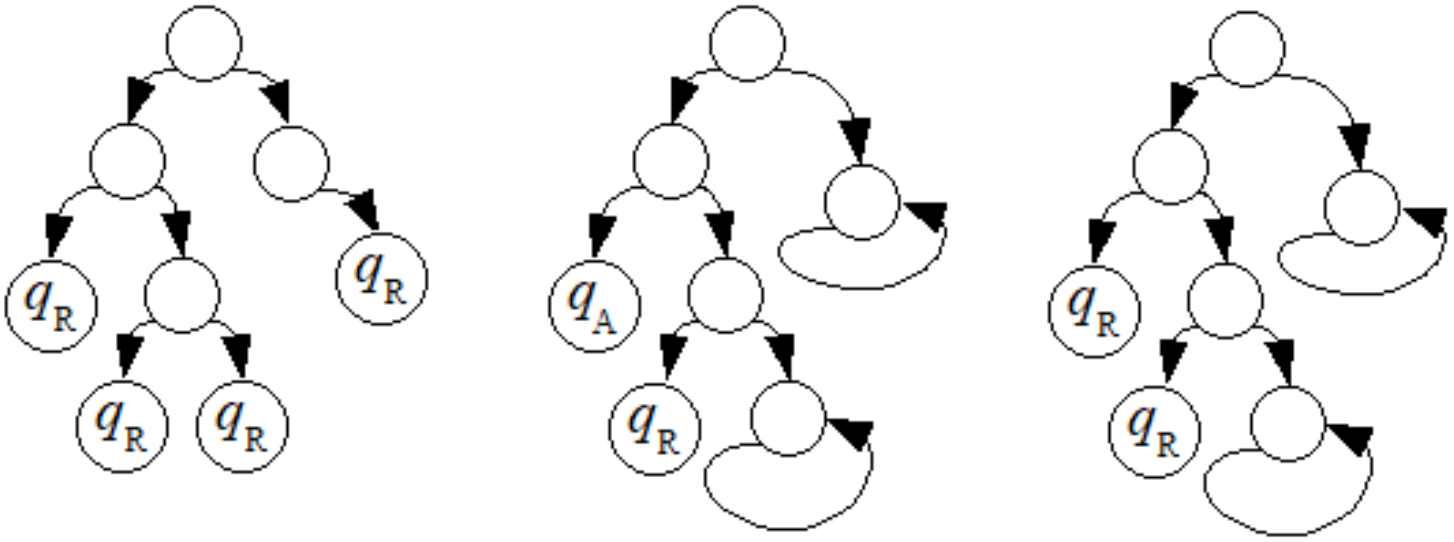
0q_R010

001q_A0 ✓

M acepta a 0010

Máquinas de Turing No Determinísticas (MTN)

Gráfico de Aceptación / Rechazo



Rechaza | **Acepta** | **Rechaza**

Máquinas de Turing No Determinísticas (MTN)

Ejercicio: Construir una MTN tal que:

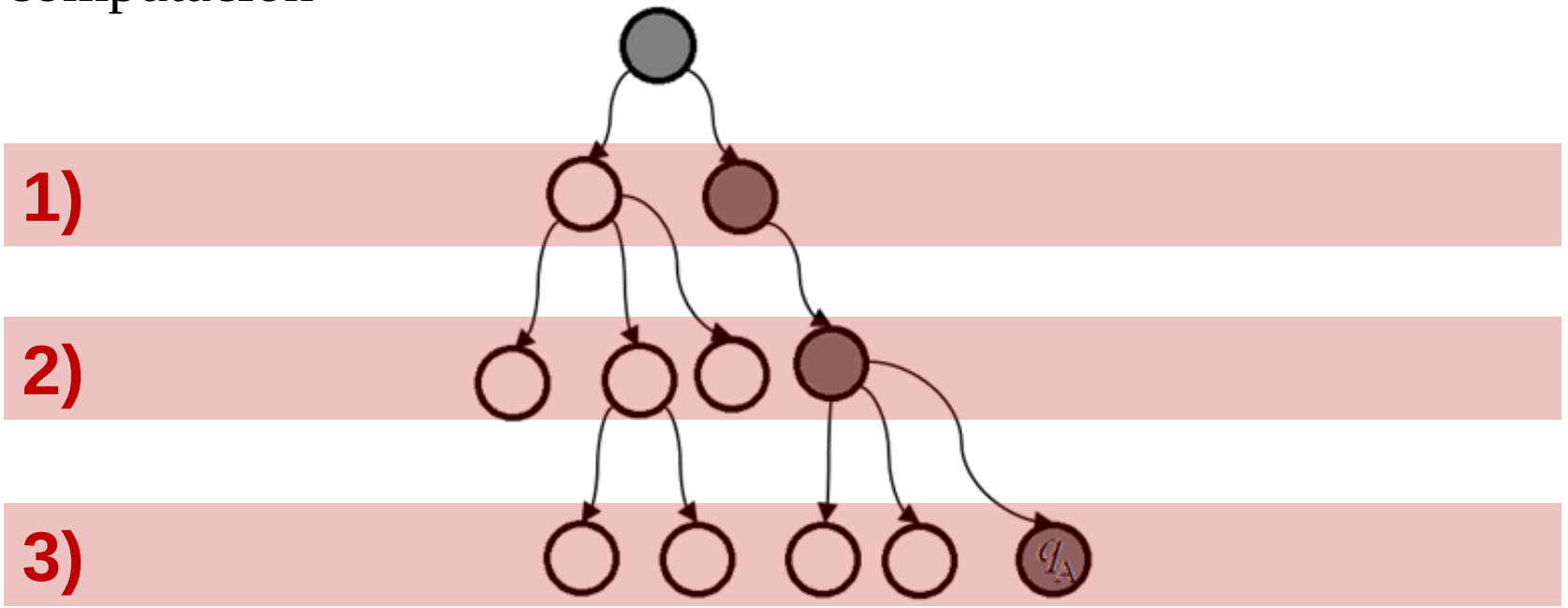
$L(M) = \{w \in \Sigma^* / \text{cant}_1(w)$ *sea divisible por* 2 *o por* 3$\}$

$\Sigma = \{0,1\}$

Máquinas de Turing No Determinísticas (MTN)

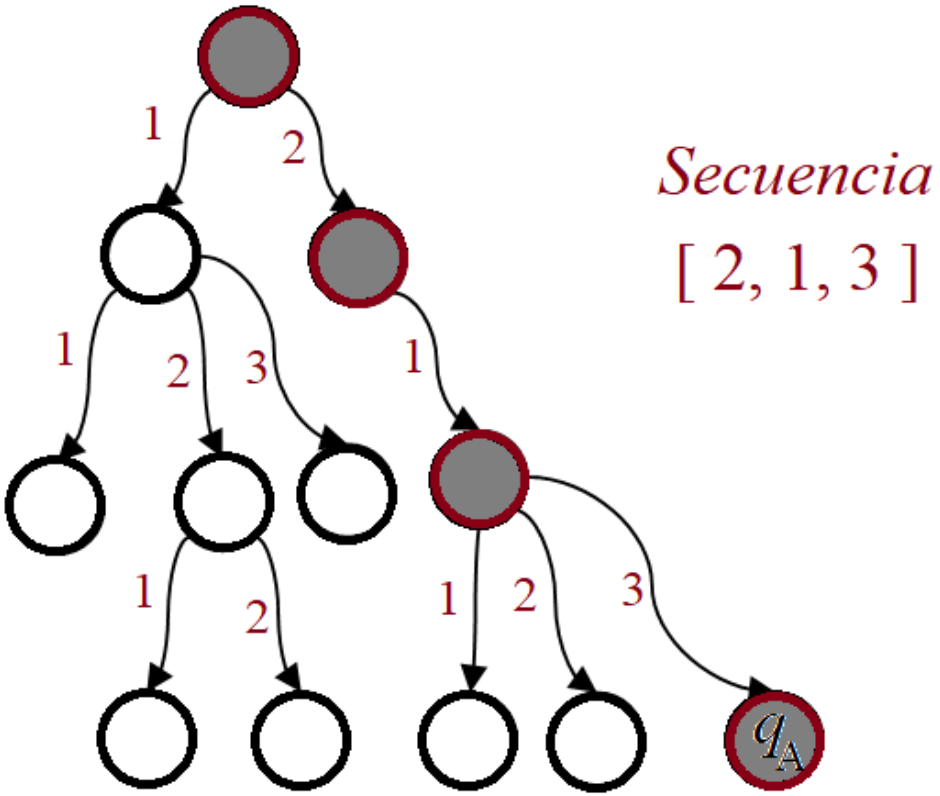
Teorema: Si L es un lenguaje aceptado por una MTN M_1 entonces L es aceptado por una MTD M_2.

Idea de la demostración: Realizar un recorrido BFS del árbol de computación



Máquinas de Turing No Determinísticas (MTN)

Sea *r* el número máximo de alternativas para cualquier par (estado, símbolo), cualquier computación (rama del árbol) puede representarse por una secuencia de dígitos del 1 a *r*.



Máquinas de Turing No Determinísticas (MTN)

Construimos M_2 con 3 cintas. En la cinta 1 está la entrada, en la cinta 2 M_2 va generando a medida que lo necesite secuencias de dígitos de 1 a *r* en orden canónico (primero las más cortas, para igual longitud según orden numérico). Supongamos $r = 3$, se irá generando:

[1] [2] [3] [1-1] [1-2] [1-3] [2-1] [2-2] [2-3] [3-1] [3-2] [3-3] [1-1-1] [1-1-2] …

Para cada secuencia que va generándose en la cinta 2 M_2 copia la entrada en la cinta 3 y simula M_1 sobre la cinta 3 utilizando la secuencia de la cinta 2 para elegir los movimientos de M_1. Algunas secuencias se descartan pues no existen *r* alternativas para todos los casos. Si M_1 alcanza q_A, M_2 acepta parando en q_A.

Máquinas de Turing No Determinísticas (MTN)

¿$L(M_1)=L(M_2)$?

Claramente si existe una sucesión de alternativas que lleva a la aceptación, ésta en algún momento será generada en la cinta 2 y al simultar M_1, M_2 aceptará.

Si ninguna sucesión de alternativas de movimientos lleva a la aceptación de M_1, M_2 tampoco aceptará.

Complejidad Temporal

Def. Sea M una MTD con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n, M hace a lo sumo $t(n)$ movimientos.

Def. Sea M una MTN con k cintas, decimos que M es de complejidad $t(n)$, si para toda entrada de longitud n, ninguna secuencia de alternativas de movimiento ocasiona que M haga más de $t(n)$ movimientos.

Clases de Complejidad

Def. Un lenguaje L $\in$ DTIME($t(n)$) sii existe una MTD M, con L=L(M) tal que la complejidad temporal de M pertenece a O($t(n)$)

Def. Un lenguaje L $\in$ NTIME($t(n)$) sii existe una MTN M, con L=L(M) tal que la complejidad temporal de M pertenece a O($t(n)$)

Corolario: DTIME($t(n)$) está incluido en NTIME($t(n)$)

Clases de Complejidad

Def. La clase de lenguajes P comprende todos los lenguajes para los que existe una MTD que lo acepta en tiempo polinomial

$$P = U_{i \geq 0} \ DTIME(n^i)$$

Def. La clase de lenguajes NP comprende todos los lenguajes para los que existe una MTN que lo acepta en tiempo polinomial

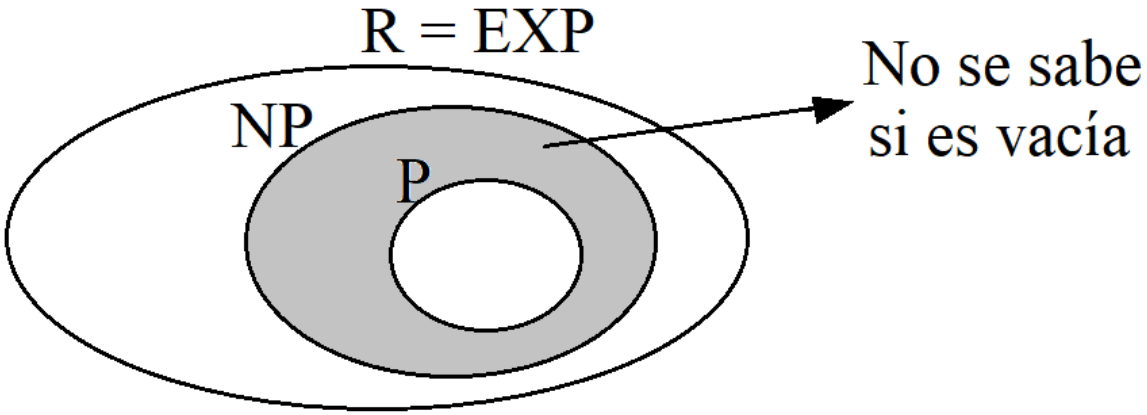
$$NP = U_{i \geq 0} \ NTIME(n^i)$$

Corolario: P es un subconjunto de NP (inmediato, pues las MTD son un caso particular de las MTN)

Clases de Complejidad

Teorema: Si L es un lenguaje recursivo aceptado por una MTN M_1 en tiempo $t(n)$, existe una MTD M_2 que lo acepta en tiempo menor o igual que $d^{t(n)}$, con d una constante mayor que 1 que depende de M_1

NOTA: Se sabe que existen lenguajes recursivos que no pertenecen a NP (R-NP $\neq \varnothing$) pero no se sabe si NP = P



Tratabilidad

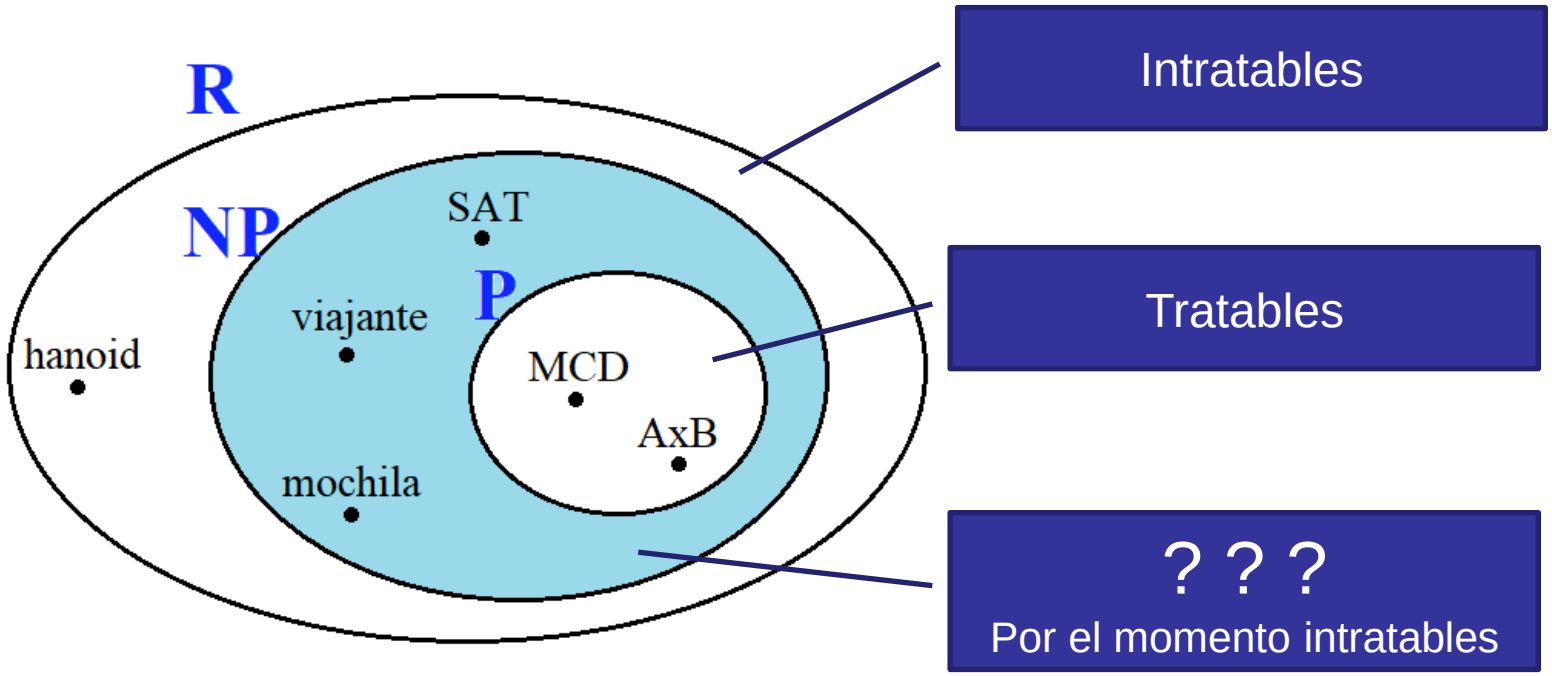
En general se está de acuerdo que la clase P representa a los problemas tratables. La mayoría de los problemas de esta clase tienen algoritmos con implementación razonable (Máximo Común Divisor, multiplicación de matrices, 2-SAT, etc).

Sin embargo existen ciertos problemas en P que realmente no son tratables debido a:

- el grado del polinomio es muy grande
- las constantes ocultas de la notación O son muy grandes
- la demostración de pertenencia a P no es constructiva y no se conoce algoritmo eficiente

Pero estos son casos extremos, y se considera a P una razonable aproximación al concepto de tratabilidad

Tratabilidad

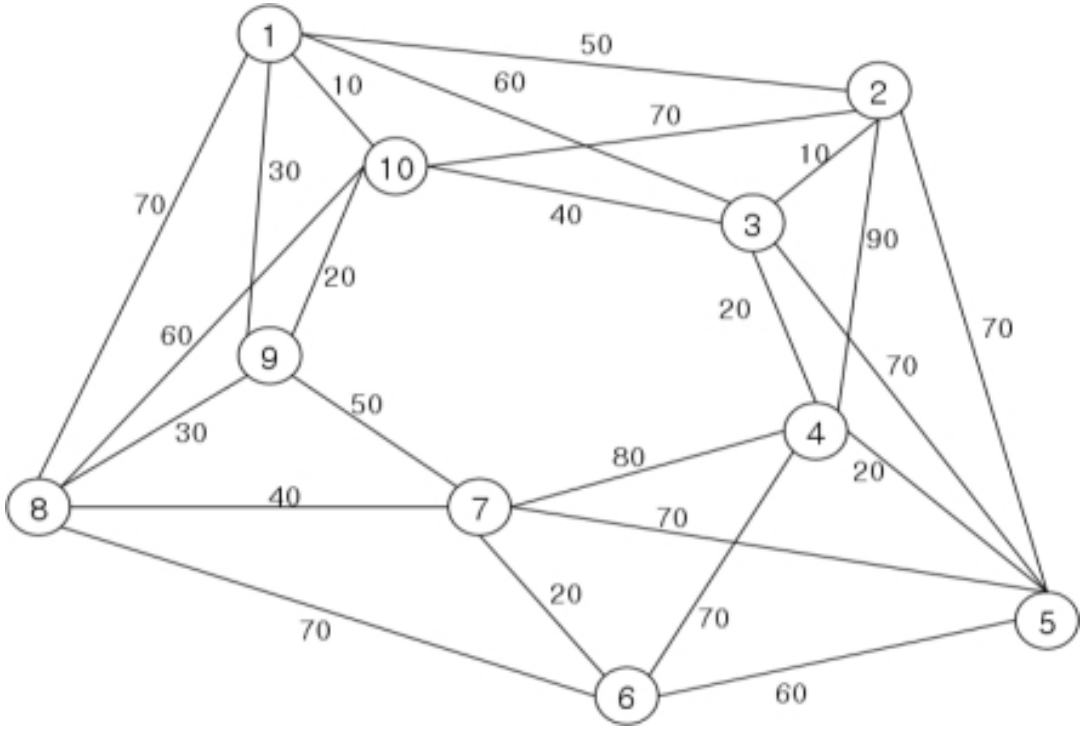


Esta brecha es inaceptablemente grande pues existen muchos problemas importantes que pertenecen a NP a los que no se le conoce una solución tratable, pero tampoco se ha demostrado que esta solución no exista.

Ejemplo clásico de problema NP

El **Problema del Viajante de Comercio** (TSP por sus siglas en inglés -*Travelling Salesman Problem*-)

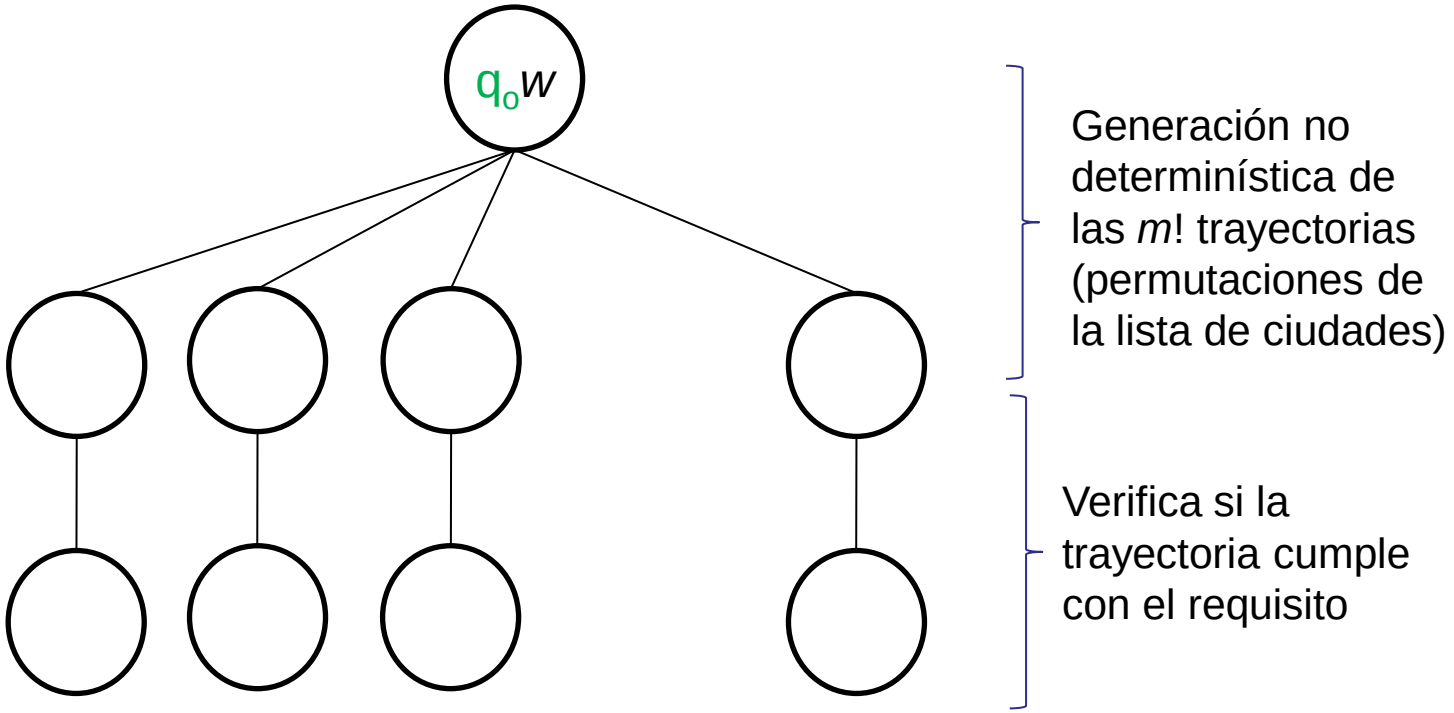
Dado un conjunto de ciudades y caminos que las unen, y una longitud máxima *d*, hay que decidir si existe alguna trayectoria que pase una sola vez por cada ciudad recorriendo una distancia no mayor que *d*



Ejemplo clásico de problema NP

TSP puede resolverse con una MTN de la siguiente manera:

1) El input *w* contiene la lista de *m* ciudades (identificadas con números de 1 a *m*), las distancias de los caminos entre las ciudades y la longitud máxima *d* permitida



Ejemplo clásico de problema NP

TSP puede resolverse con una MTN de la siguiente manera:

1) El input w contiene la lista de m ciudades (identificadas con números de 1 a m), las distancias de los caminos entre las ciudades y la longitud máxima d permitida

2) De manera no determinística se escribe una permutación cualquiera de los números entre 1 y m. Puede hacerse así:

   Se escriben 2 listas, la primera con los números ordenados de 1 a m y la segunda vacía. Se elige de manera no determinista uno de la primera lista que se tacha y pasa a la segunda. Cada ciclo tiene un costo de $O(m)$, que se repite m veces hasta completar la permutación ($O(m^2)$), como la lista de ciudades es una parte de la entrada podemos acotarlo con $O(n^2)$

3) Se calcula la distancia del recorrido elegido. Suponiendo un costo $O(n)$ para localizar en la entrada la distancia entre 2 ciudades cualquiera, el costo de calcular la distancia del recorrido es $O(n^2)$

4) Si la distancia del recorrido es $\leq d$ para en q_A sino para en q_R.

Claramente M trabaja en tiempo $O(n^2)$

CO-NP

Definición: CO-NP = $\{ L / \overline{L} \in NP \}$

Teorema 1: Si $L \in P \Rightarrow L \in (NP \cap CO\text{-}NP)$

a) Si $L \in P \Rightarrow L \in NP$ (por def. de P y NP)

b) Si $L \in P \Rightarrow \overline{L} \in P$ (simplemente intercambiando q_A y q_R en la MTD)

$\Rightarrow \overline{L} \in NP$ (por def. de P y NP)

$\Rightarrow L \in CO\text{-}NP$ (por def. CO-NP)

De a) y b) se tiene que $L \in (NP \cap Co\text{-}NP)$

Nota: No se sabe cómo están relacionados NP y CO-NP. No se puede asumir que son iguales. Reconocer el complemento de un lenguaje, intentando intercambiar estados q_A por q_R de su MTN no funciona, una MTN puede aceptar teniendo computaciones que terminan en q_R pues alcanza que una sola termina en q_A.

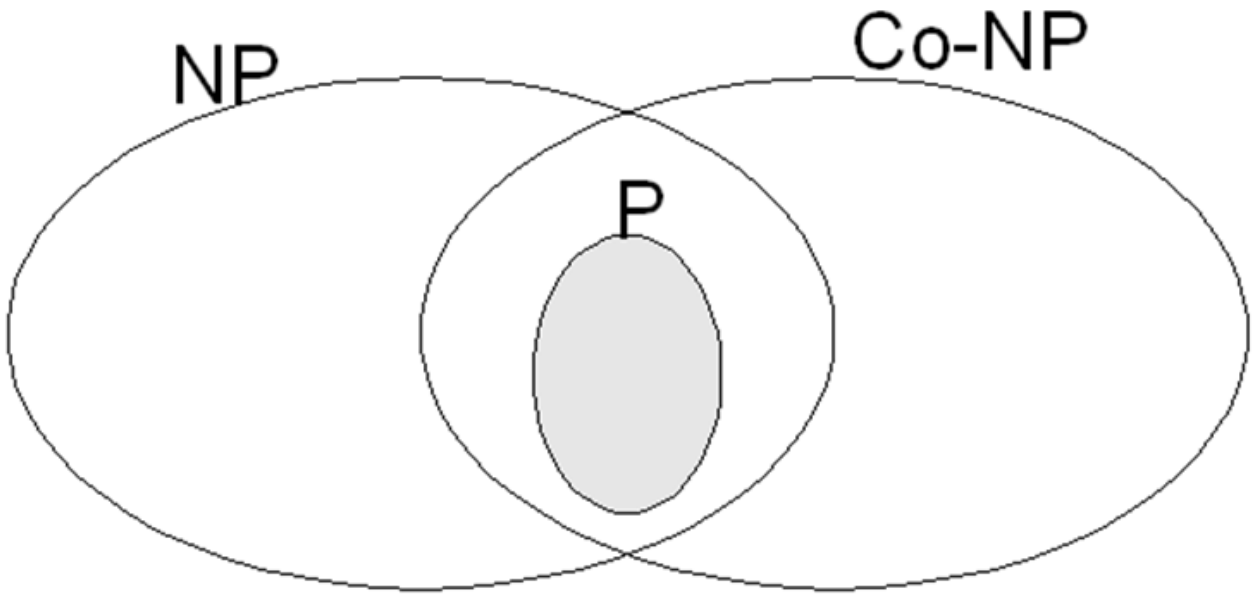
CO-NP

Teorema 2: NP ≠ Co-NP $\Rightarrow$ P ≠ NP

Dem: Si P = NP $\Rightarrow$ NP = CO-NP (porque P = CO-P)

entonces NP ≠ CO-NP $\Rightarrow$ P ≠ NP (por contra-recíproca)

Esta es la situación conocida



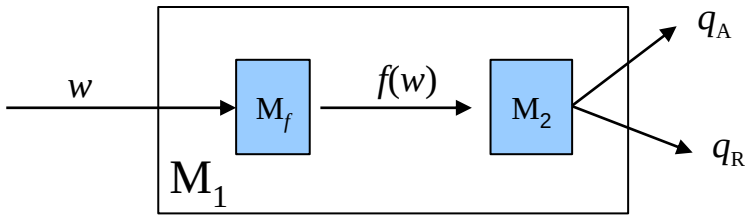
Reducción polinomial

Sean L_1 y L_2 dos lenguajes incluidos en Σ^*, decimos que L_1 se reduce polinomialmente a L_2 (se denota $L_1 \alpha_p L_2$) sii $L_1 \alpha L_2$ y además la función de reducción *f* es computada por una MTD que trabaja en tiempo polinomial ($f \in P$)

Reducción polinomial

Teorema 3: $L_1 \alpha_p L_2$ y $L_2 \in P \Rightarrow L_1 \in P$

Dem: Sea M_f la MTD que computa f en tiempo polinomial. Construimos M_1 una MTD tal que $L_1=L(M_1)$ de la siguiente manera:



$L_1=L(M_1)$?

Dado que $w \in L_1$ sii $f(w) \in L_2$ se tiene que $L_1= L(M_1)$

M_1 trabaja en tiempo polinomial ?

Sea $n = |w|$

M_f computa $f(w)$ en a lo sumo cn^k pasos,

por lo tanto $| f(w) | \leq cn^k$

por lo tanto M_1 hará a lo sumo $cn^k + c_2(cn^k)^{k2}$

por lo tanto M_1 trabaja en tiempo $O(n^{k3})$

Por lo tanto $L_1 \in P$

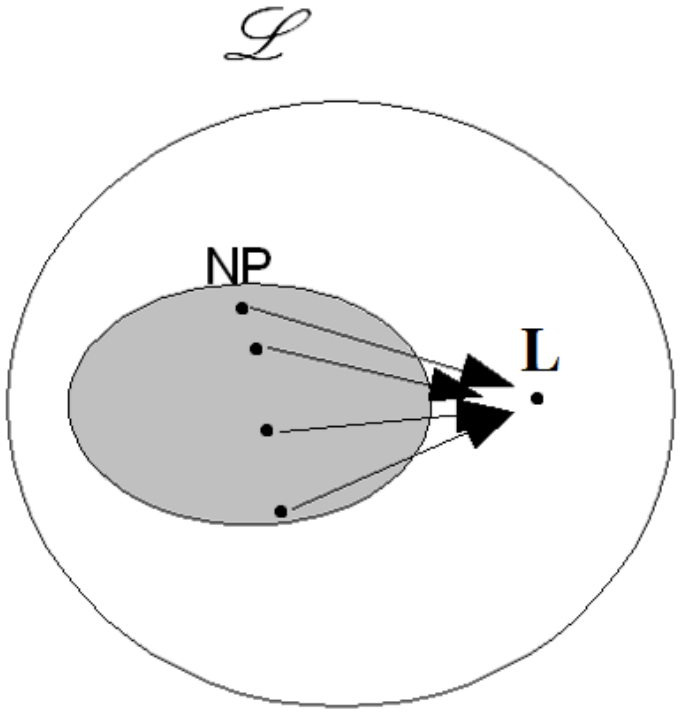
Reducción polinomial

Teorema 4: Sea L un lenguaje tal que $\varnothing \subset L \subset \Sigma^*$ entonces para cualquier lenguaje L' perteneciente a P, vale que $L' \alpha_p L$

Dem: Como $\varnothing \subset L \subset \Sigma^*$ entonces existe algún $x \in L$ y algún $y \notin L$. La reducción consiste para cada *w*, resolver L' con su MTD polinomial, y asignar *x* o *y* según se *acepte* o *rechace* *w* respectivamente.

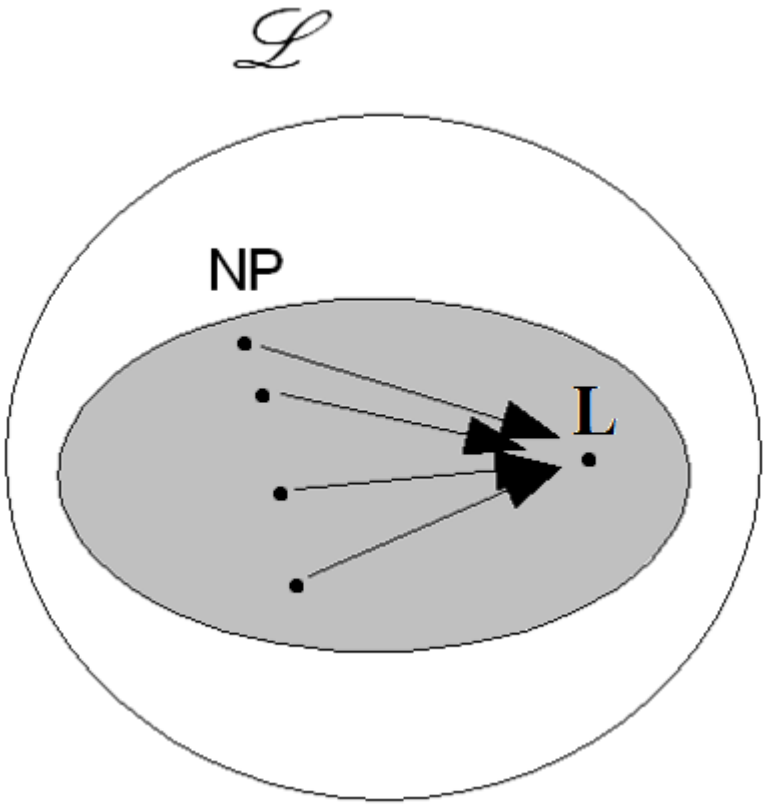
NP-Hard

Definición: L $\in$ NPH (NP-Hard) sii para todo L' $\in$ NP se cumple L' α_p L



NP-Completo

Definición: L ∈ NPC (NP-Completo) sii L ∈ NPH y L ∈ NP



NP-Completo

Hasta ahora no se a encontrado ningún lenguaje que cumpla con esto

Teorema 5: Si $(L \in NPC)$ AND $(L \in P) \Rightarrow P = NP$

Dem: Sea L' un lenguaje arbitrario de NP

$L' \in NP \Rightarrow L' \alpha_p L$ (porque por hipótesis $L \in NPC$)

$\Rightarrow L' \in P$ (por hiótesis $L \in P$ y por teorema 3)

Por lo tanto $NP \subseteq P$

Por lo tanto P = NP

Nota: Según este teorema para demostrar que P = NP alcanzaría con encontrar una solución polinomial para cualquiera de los problemas NPC conocidos.

NP-Completo

Teorema 6: Sean $L_1, L_2 \in NP$. Si $L_1 \in NPC$ y $L_1 \alpha_p L_2$ entonces $L_2 \in NPC$

Dem: Para todo $L \in NP$ existe $L \alpha_p L_1$ (por $L_1 \in NPC$)

Dado que por hipótesis $L_1 \alpha_p L_2$ se tiene que:

Para todo $L \in NP$ existen $L \alpha_p L_1$ y $L_1 \alpha_p L_2$

Por lo tanto, para todo $L \in NP$ existe $L \alpha_p L_2$ (porque α_p es transitiva)

Por lo tanto $L_2 \in NPH$ (por definición de NPH)

Por lo tanto $L_2 \in NPC$ (por hipótesis $L_2 \in NP$)

Ejercicio para el lector: demostrar que α_p es transitiva

Ejemplo de leguaje NPC - Lenguaje SAT

Literal: es una variable proposicional o la negación de una variable proposicional.

FNC: fórmula normal conjuntiva, es una conjunción de cláusulas, donde una cláusula es una disyunción de literales. Ejemplo:

$$\varphi = (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_4)$$

Enunciado del problema SAT: Dada una fórmula proposicional en FNC, determinar si es satisfactible, es decir si existe alguna asignación de verdad para las variables que haga verdadera la fórmula.

$$SAT = \{\varphi \mid \varphi \text{ es una fórmula booleana FNC satisfactible}\}$$

Ejemplo de leguaje NPC - Lenguaje SAT

Se ha demostrado que SAT pertenece a NPC **(Teorema Cook/Levin)**.

Se prueba que SAT pertenece a NP y que para cualquier lenguaje arbitrario L perteneciente a NP existe L α_p SAT.

La idea de la reducción es la siguiente:

Como L pertenece a NP sabemos que existe una MTN M (con L=L(M)) que acepta o rechaza un string *w* en tiempo polinomial.

Se define una función de reducción que puede ser computada en tiempo polinomial por una MTD M_f que a partir de una entrada w y una MTN M define una fórmula bien formada $\varphi = f(M, w)$ con el objetivo que φ sea satisfactible si y solo si M acepta *w*

Ejemplo de leguaje NPC - Lenguaje SAT

Más de 1000 problemas de dominios diferentes, y con variadas aplicaciones se han probado que pertenecen a NPC haciendo reducciones desde un lenguaje que ya se conozca su pertenencia a NPC. Para ninguno de ellos se ha podido encontrar una solución polinomial.

Ejemplos:

SAT α_p 3-SAT

3-SAT α_p VC={(G,k)/G es un grafo que tiene un cubrimiento de vértices de tamaño k}

Sea G=(V,E) un grafo y sea C⊂V un subconjunto de vértices. Decimos que C es un cubrimiento de vértices si cualquier e∈E tiene un extremo que pertenece a C.

VC α_p *k*-clique={(G,k)/ G tiene un clique de tamaño k},

Clique es un subgrafo totalmente conectado

¿¿ P = NP ??

Por varias décadas de estudio los investigadores no han podido responder a esta pregunta.

Por ello se cree que P ≠ NP es mas plausible que P = NP