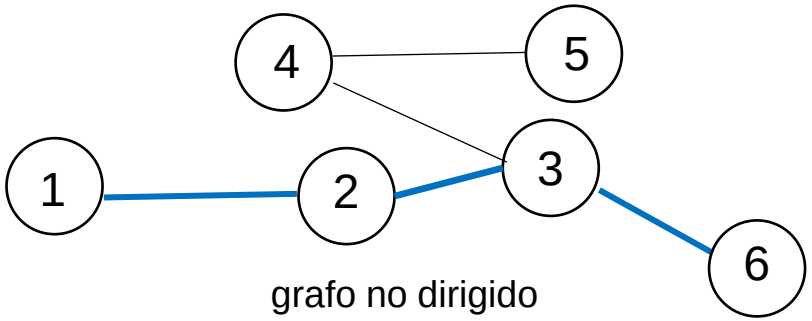
Clase teórica 8

Temas avanzados de complejidad computacional

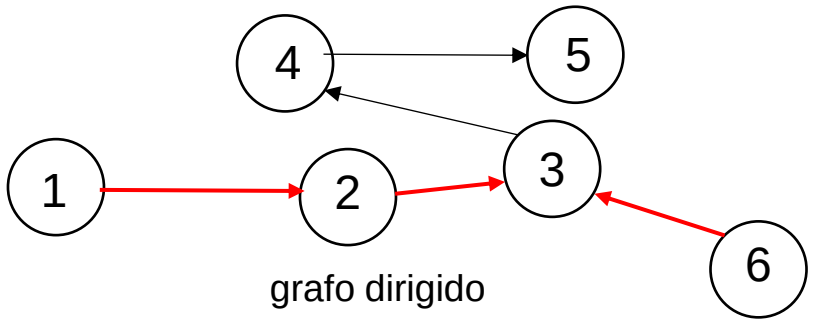
1. Profundizando en la complejidad espacial

- Se prueba (O. Reingold, 2005) que el problema de accesibilidad en grafos no dirigidos está en LOGSPACE:
  ACC = {G | G es un grafo **no dirigido** con un camino del primero al último vértice} **está en LOGSPACE.**

- No pareciera que el mismo problema pero en grafos dirigidos lo cumpla:
  D-ACC = {G | G es un grafo **dirigido** con un camino del primero al último vértice} **no estaría en LOGSPACE:**



grafo no dirigido



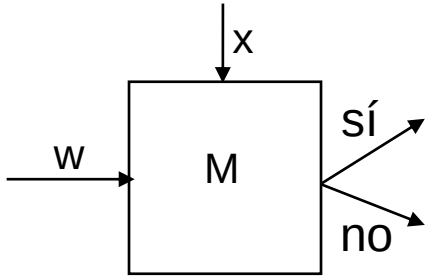
grafo dirigido

En el segundo caso hay que contemplar la dirección de los arcos

- **D-ACC <u>sí está</u> en la clase NLOGSPACE**, definida de la misma manera que hemos definido NP:

- Un lenguaje L pertenece a **NLOGSPACE** sii:

  cuenta con una MT M que para toda cadena w puede **verificar** en espacio **$O(\log_2|w|)$** si $w \in L$,
  con la ayuda de otra cadena (**certificado sucinto x**, es decir que $|x| \leq$ poly($|w|$)),
  tal que x está en una cinta auxiliar de **sólo lectura** y cuyo cabezal **sólo va a la derecha**:



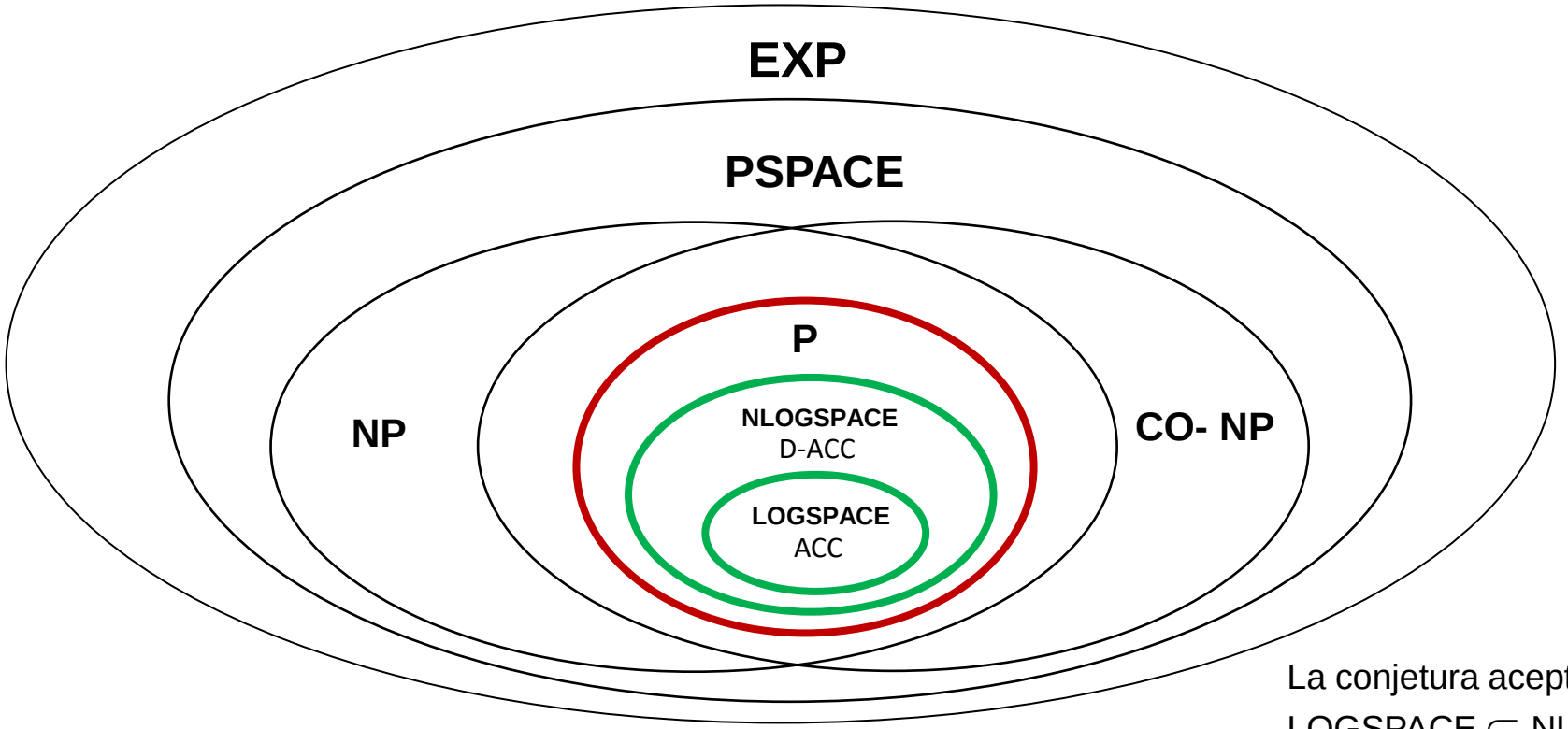
w está en la cinta de entrada, de sólo lectura.
x está en una cinta auxiliar, de sólo lectura, cuyo cabezal sólo va a la derecha (en caso contrario, M ocuparía espacio polinomial).

- **D-ACC ∈ NLOGSPACE**. Dado un grafo dirigido G de m vértices y una secuencia $C = (i_1, i_2, \ldots, i_k)$ de vértices, la siguiente MT M verifica en espacio $O(\log_2|G|)$ si C es un camino en G del primero al último vértice:

  1. Si G no es una entrada válida, rechaza.
  2. Hace a := 1 (con *a* recorre C).
  3. Si $i_a \neq 1$, rechaza (C no empieza con el vértice 1).
  4. Si (i_a, i_{a+1}) no es un arco de G, rechaza, y si $i_{a+1} = m$, acepta (C es un camino).
  5. Hace a := a + 1. Si a = m, rechaza (C no es un camino).
  6. Vuelve a (4).

  Ejercicio: comprobar que M verifica D-ACC y lo hace en espacio $O(\log_2|G|)$.

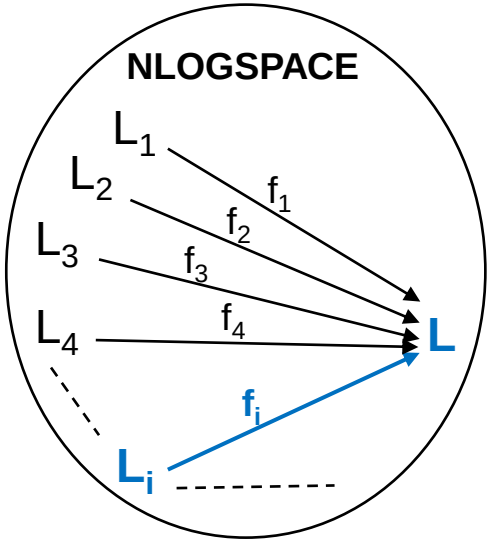
- Mayor detalle de la **Jerarquía Espacial**:



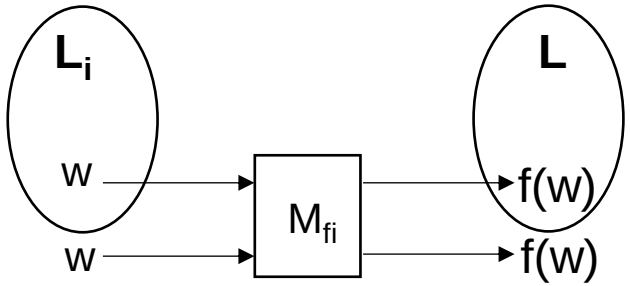
La conjetura aceptada es
LOGSPACE $\subset$ NLOGSPACE $\subset$ P

- D-ACC está **entre los lenguajes más difíciles de NLOGSPACE.**
- Formalmente, **D-ACC es NLOGSPACE-completo respecto de las reducciones logarítmicas (o log-space).**
- **Definición:** Una reducción log-space es una reducción polinomial (o poly-time) que **ocupa espacio logarítmico.**
- **Notación:** $L_1 \leq_{log} L_2$

- **Si un lenguaje L es NLOGSPACE-completo y está en LOGSPACE, entonces LOGSPACE = NLOGSPACE.**

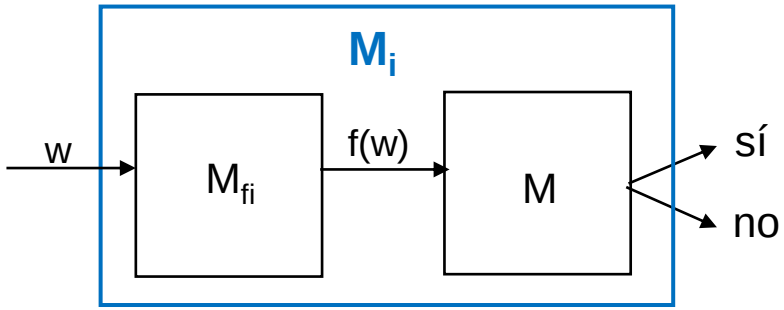


Toda f_i es una reducción log-space



M_{fi} computa f_i en espacio $O(\log_2|w|)$
(|w| y |f(w)| no cuentan)

Si M es una MT que decide L en espacio logarítmico, la siguiente MT M_i decide L_i en espacio logarítmico:



M_{fi} computa f_i en espacio logarítmico
M decide L en espacio logarítmico
¡Pero f(w) puede medir poly(|w|)! (¿por qué?)

<u>Veamos cómo construir M_i para que no viole la cota espacial logarítmica</u>:

M_{fi} no le envía completa su salida a M, sino que le pasa cada vez **sólo el símbolo que M requiere**:

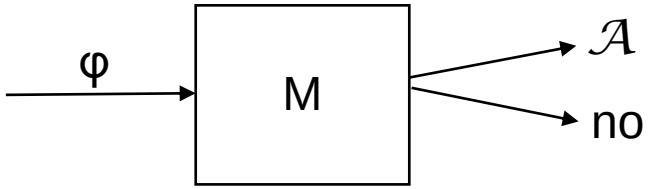
- Si M se mueve a la derecha en su cinta de entrada, M_{fi} le envía el símbolo siguiente de f(w).
- Si M se mueve a la izquierda en su cinta de entrada, M_{fi} se reinicia, obtiene el símbolo de f(w) que M requiere y se lo envía.

- Por otra parte, la clase **NPSPACE** también se define de manera similar a cómo se definen NP y NLOGSPACE.
  En este caso, **PSPACE = NPSPACE**. También se cumple **NLOGSPACE = CO-NLOGSPACE** y **NPSPACE = CO-NPSPACE.**
  (notar la diferencia con lo que ocurre en la jerarquía temporal)

2. Profundizando en la complejidad temporal de los problemas de búsqueda

- Los problemas más generales son los de **búsqueda**.
- Las MT asociadas no sólo aceptan sino que también **devuelven una solución** si existe.
- Por ejemplo, en el caso del problema de búsqueda asociado a SAT, conocido como **FSAT**, la MT M correspondiente, dada una fórmula booleana de entrada φ, hace:

  (a) Devuelve una asignación $\mathcal{A}$ que satisface φ, si existe.
  (b) Responde no, si no existe.



MT que resuelve FSAT

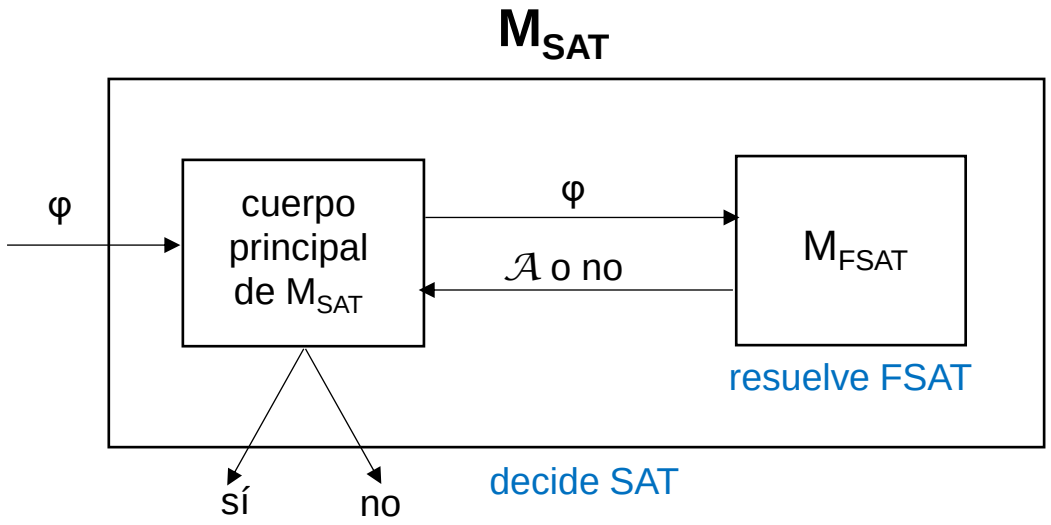
- Las clases de problemas de búsqueda asociadas a las clases de lenguajes P, NP, etc., se identifican con **FP, FNP,** etc. Por ejemplo, **FSAT ∈ FNP**.
- Intuitivamente, **decidir la existencia** de una solución es más fácil que **encontrar una.** Pero en el caso, por ejemplo, de los lenguajes **NP-completos**, ¿realmente es así? Mostramos que **no** con un ejemplo.

- **El problema FSAT es tan o más difícil que el problema SAT (intuitivo).**

  Sea M_{FSAT} una MT que obtiene en tiempo poly(n), si existe, una asignación $\mathcal{A}$ que satisface una fórmula φ.

  La siguiente MT M_{SAT} decide en tiempo poly(n) si una fórmula φ es satisfactible:

  Dada φ, M_{SAT} invoca a M_{FSAT} y acepta sii M_{FSAT} devuelve una asignación $\mathcal{A}$.



¿Por qué M_{SAT} tarda tiempo polinomial?

Si FSAT ∈ FP entonces SAT ∈ P.
O lo mismo: si SAT ∉ P entonces FSAT ∉ FP.

FSAT es tan o más difícil que SAT

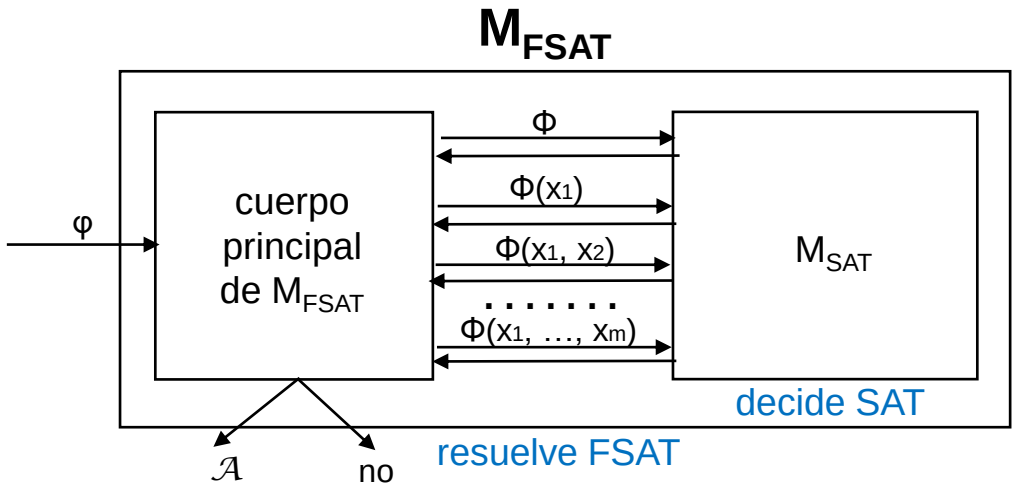
- **Recíprocamente, el problema SAT es tan o más difícil que el problema FSAT (menos intuitivo).**

Sea M_{SAT} una MT que decide en tiempo poly(n) si una fórmula φ es satisfactible.

La siguiente MT M_{FSAT} encuentra en tiempo poly(n), si existe, una asignación $\mathcal{A}$ que satisface una fórmula φ:

Dada φ con m variables, M_{FSAT} invoca a M_{SAT} y devuelve, si existe, una asignación $\mathcal{A}$ que satisface φ:

1. Invoca a M_{SAT} con φ. Si M_{SAT} responde no, entonces **responde no**.
2. Invoca a M_{SAT} con φ haciendo x_1 = verdadero.
   Si M_{SAT} responde sí, **fija para x_1 el valor verdadero**, y si responde no, **fija para x_1 el valor falso**.
3. Invoca a M_{SAT} con φ, utilizando el valor de x_1 obtenido y haciendo x_2 = verdadero.
   Si M_{SAT} responde sí, **fija para x_2 el valor verdadero**, y si responde no, **fija para x_2 el valor falso**.
4. Repite el proceso con las variables **x_3, x_4, …, x_m**, y al final **devuelve la asignación $\mathcal{A}$ obtenida**.



¿Por qué M_{FSAT} tarda tiempo polinomial?

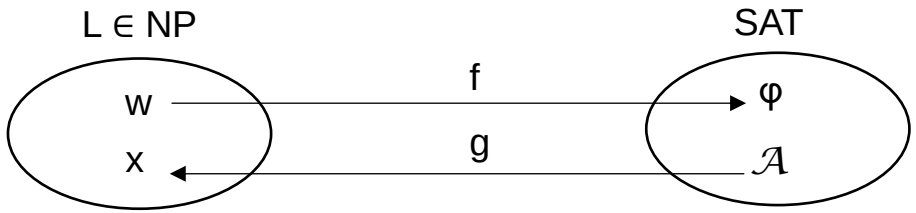
Si SAT ∈ P, FSAT ∈ FP.
Lo mismo: si FSAT ∉ FP, SAT ∉ P.

SAT es tan o más difícil que FSAT

Notar que asumimos que M_{SAT} puede recibir como entrada una fórmula con variables y constantes.
Otra posibilidad es que M_{FSAT} simplifique antes de invocar.

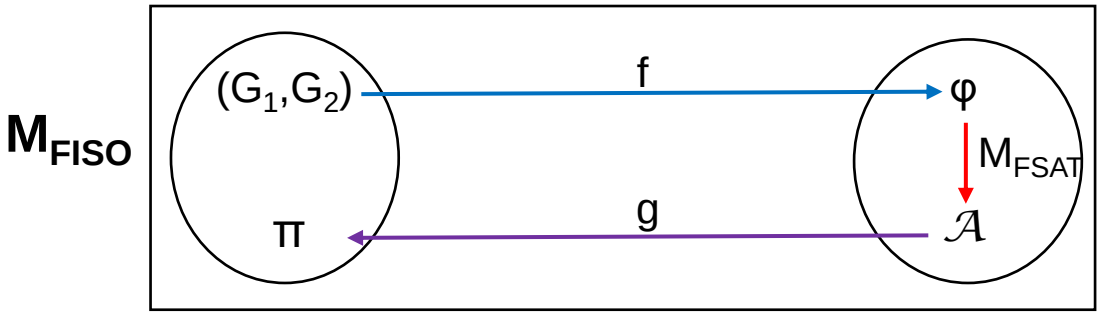
- **Se cumple además que P = NP implica FP = FNP (aún menos intuitivo).**
  En otras palabras, si P = NP, todo problema de búsqueda de FNP se puede resolver en tiempo polinomial.

  La prueba se basa en una propiedad de la reducción f utilizada para probar que SAT es NP-completo:
  no sólo se cumple que f, de tiempo polinomial, satisface **w ∈ L sii f(w) = φ ∈ SAT,**
  sino que además existe una función g, de tiempo polinomial, que **asigna a un certificado de φ uno de w:**



f y g se computan en tiempo polinomial
$\mathcal{A}$ es un certificado de φ
x es un certificado de w
f y g constituyen una Levin-reducción

Usando tal propiedad, esbozamos la prueba tomando como ejemplo el problema del **isomorfismo de grafos**:



M_{FISO} resuelve FISO en tiempo polinomial

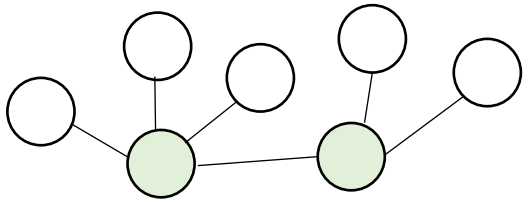
- f asigna en tiempo polinomial al **par (G_1, G_2) de ISO** una fórmula booleana φ de SAT (f y g son una Levin-reducción).
- M_{FSAT} obtiene una asignación $\mathcal{A}$ que satisface φ, invocando a una MT M_{SAT} que decide SAT,
  y como por hipótesis P = NP, entonces M_{SAT} tarda tiempo polinomial (y también M_{FSAT} por lo visto antes).
- g asigna en tiempo polinomial a $\mathcal{A}$ **un isomorfismo π entre G_1 y G_2** (f y g son una Levin-reducción).

3. Profundizando en las aproximaciones polinomiales

- Mencionamos antes el problema de optimización del cubrimiento de vértices de un grafo:

Problema OCV: dado un grafo G, encontrar su cubrimiento de vértices mínimo.

Ejemplo



Cubrimiento mínimo (tamaño 2)

- El lenguaje que representa el problema es **CV = {(G, K) | G tiene un cubrimiento de K vértices}.**
- Como CV es NP-completo, se cumple que si P ≠ NP, entonces **OCV no tiene resolución polinomial** (ejercicio).
- En la práctica, a los problemas de optimización asociados a los lenguajes NP-completos se les busca algoritmos eficientes que generen soluciones “buenas”, “cercanas al óptimo” (**aproximaciones polinomiales**).
- **(Antes vimos un algoritmo que encuentra un cubrimiento de tamaño a lo sumo el doble del óptimo.)**

- Formalizando, dada una MT M que computa una aproximación polinomial para un problema OL:
  - si **opt(w)** es la medida de la solución óptima para la cadena w,
  - y **m(M(w))** es la medida de la solución obtenida por M para w,
  - el **error relativo** ε_w de M con respecto a w es:

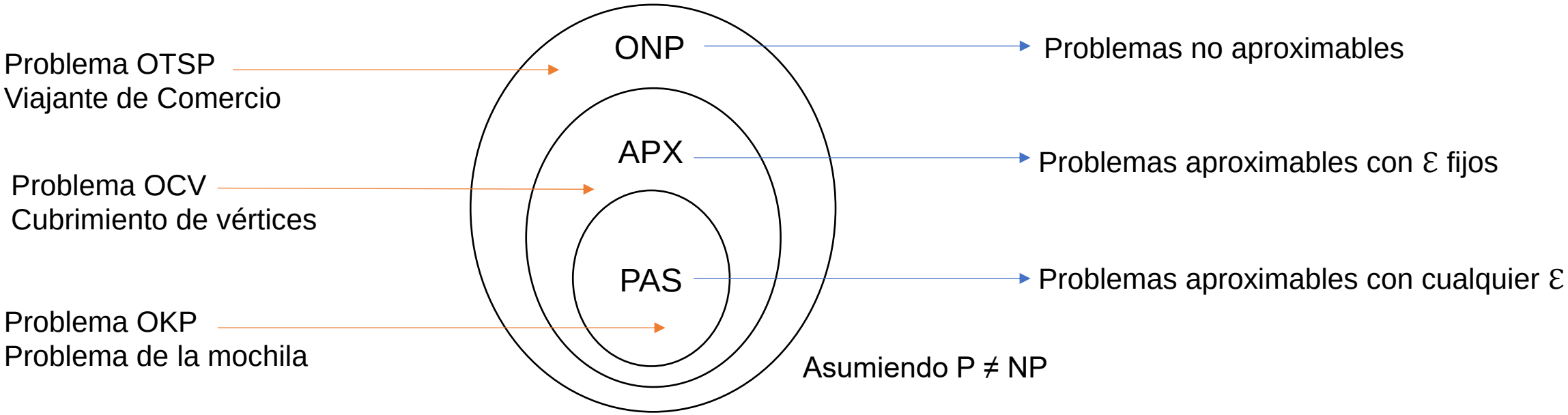
$$\varepsilon_w = \frac{|m(M(w)) - opt(w)|}{\max(m(M(w)), opt(w))}$$

  - ε_w siempre varía entre 0 y 1.
  - Si ε es el máximo de todos los ε_w, se dice que M es una **ε-aproximación polinomial** (o **tiene umbral ε**).
  - P.ej., la aproximación polinomial que construimos para OCV es una **1/2-aproximación polinomial**.

- El objetivo es encontrar aproximaciones polinomiales con un umbral **lo más chico posible**.

- Asumiendo P ≠ NP:
  - Existen problemas aproximables con **cualquier ε**, problemas aproximables con **ε fijos** (como OCV, donde ε = 1/2), y problemas no aproximables (**no existen MT que los resuelvan con ningún ε**).
  - La jerarquía de problemas correspondiente es:
    1. La clase **PAS** contiene los problemas aproximables con cualquier ε.
    2. La clase **APX** contiene los problemas aproximables con ε fijos.
    3. Se cumple **PAS** $\subset$ **APX** $\subset$ **ONP** (ONP es la clase de los problemas de optimización asociados a NP).

- Algunos problemas representativos de la jerarquía:

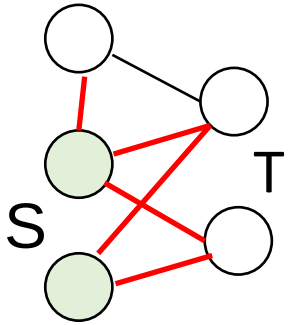


- El **problema de la mochila** (o problema KP, por *Knapsack Problem*) plantea optimizar el armado de una mochila:
  - Se define un peso máximo.
  - Se considera un conjunto de objetos, cada uno con peso y un volumen específicos.
  - Hay que maximizar el volumen ocupado sin exceder el peso máximo.

- También en este caso se plantean problemas completos, usando reducciones de determinadas características (**APX-reducciones**). No se conocen muchos problemas **ONP-completos**.

- **Otro ejemplo. Problema de búsqueda del máximo corte de un grafo (OMC).**

  - Dado un grafo G = (V, E), un **corte** de G es una partición {S, T} de V.
  - Un **arco de corte** conecta un vértice de S con un vértice de T.
  - El problema consiste en encontrar el **corte con el máximo número de arcos**.

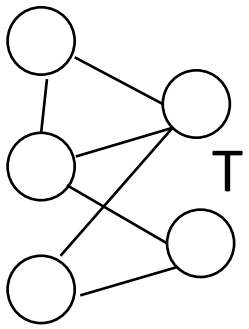


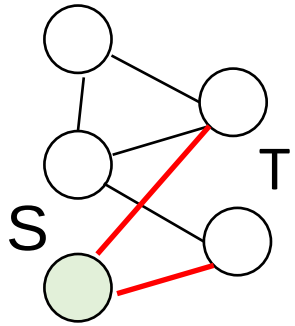
Ej. Corte de 5 arcos

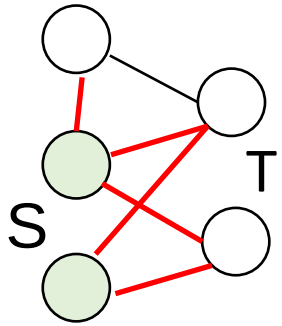
- El lenguaje asociado es MC = {(G, K) | G tiene un corte de K arcos}. Se prueba que MC es NP-completo.

- La siguiente MT M es una **1/2-aproximación polinomial** para el problema. Dado G = (V, E), M hace:

  1. Hace S := ∅ y T := V.
  2. Si moviendo un vértice i de S a T o de T a S crece el nro de arcos de corte, entonces lo mueve.
  3. Vuelve a (2).







- El tiempo de M es **polinomial** porque en cada paso (2) a lo sumo se agregan |E| arcos al corte, y hay |V| vértices.
- El corte obtenido mide **al menos la mitad del máximo:** Todo vértice i cumple que sus arcos de corte suman al menos como sus arcos no de corte (si no, el vértice i estaría en el otro conjunto). A partir de esto no es difícil llegar al umbral 1/2.

4. La jerarquía polinomial

- Vimos que un lenguaje L ∈ NP sii existe una MT M polinomial y un polinomio p tal que para toda cadena w:

  w ∈ L sii (∃x: |x| ≤ p(|w| tal que M acepta (w, x))

- Sean por ejemplo los lenguajes:

  MAX-IND = {(G, K) | el grafo G tiene un conjunto independiente de K vértices (K vértices no adyacentes), y no tiene conjuntos independientes de vértices más grandes}

  MIN-FORM = {(φ, K) | la fórmula booleana φ tiene K símbolos, y no existe ninguna fórmula booleana equivalente más chica}

- Dichos lenguajes también se pueden definir con MT polinomiales y certificados sucintos:

  MAX-IND: Existe una MT M polinomial y un polinomio p tal que para toda cadena w:
  w ∈ MAX-IND sii ($\exists x_1$: $|x_1| \leq p(|w|$, $\forall x_2$: $|x_2| \leq p(|w|)$ tal que M acepta (w, x_1, x_2)), siendo:
  w = (G, K), x_1 es un conjunto de K vértices de G, y x_2 es un conjunto de más de K vértices de G.

  MIN-FORM: Existe una MT M polinomial y un polinomio p tal que para toda cadena w:
  w ∈ MIN-FORM sii ($\forall x_1$: $|x_1| \leq p(|w|$, $\exists x_2$: $|x_2| \leq p(|w|)$ tal que M acepta (w, x_1, x_2)), siendo:
  w = (φ, K), x_1 es una fórmula booleana de menos de K símbolos, y x_2 es una asignación de valores de verdad.

De esta manera, MAX-IND y MIN-FORM no estarían en NP

- Se definen:

  - La clase $\Sigma_2 P$, con lenguajes expresados como se expresa MAX-IND
  - La clase $\Pi_2 P$, con lenguajes expresados como se expresa MIN-FORM
  - Y en general, las clases $\Sigma_i P$ y $\Pi_i P$, de la siguiente manera:

- **$L \in \Sigma_i P$,** con $i \geq 0$, si existen una MT M polinomial y un polinomio p tal que, para toda cadena w:

  $w \in L$ sii ($\exists x_1$: $|x_1| \leq p(|w|)$, $\forall x_2$: $|x_2| \leq p(|w|)$, $\exists x_3$: $|x_3| \leq p(|w|)$, …, M acepta ($w, x_1, x_2, x_3, \ldots$))

- **$L \in \Pi_i P$,** con $i \geq 0$, si existen una MT M polinomial y un polinomio p tal que, para toda cadena w:

  $w \in L$ sii ($\forall x_1$: $|x_1| \leq p(|w|)$, $\exists x_2$: $|x_2| \leq p(|w|)$, $\forall x_3$: $|x_3| \leq p(|w|)$, …, M acepta ($w, x_1, x_2, x_3, \ldots$))

Para todo i, la clase $\Pi_i P$ tiene los complementos de la clase $\Sigma_i P$ (ejercicio)

- Así llegamos a la definición de la **jerarquía polinomial o PH:** es la unión infinita de las clases $\Sigma_i P$, con $i \geq 0$. Se la puede definir también como la unión infinita de las clases $\Pi_i P$. PH se estructura por niveles:
  - **Nivel 0:** las clases $\Sigma_0 P = \Pi_0 P =$ **P**
  - **Nivel 1:** las clases $\Sigma_1 P =$ **NP** y $\Pi_1 P =$ **CO-NP**
  - **Nivel 2:** las clases $\boldsymbol{\Sigma_2 P}$ y $\boldsymbol{\Pi_2 P}$
  - Etc.

- Se cumple, para todo $i \geq 0$, que $\boldsymbol{\Sigma_{i+1} P}$ **y** $\boldsymbol{\Pi_{i+1} P}$ **incluyen a** $\boldsymbol{\Sigma_i P}$ **y** $\boldsymbol{\Pi_i P}$**.**

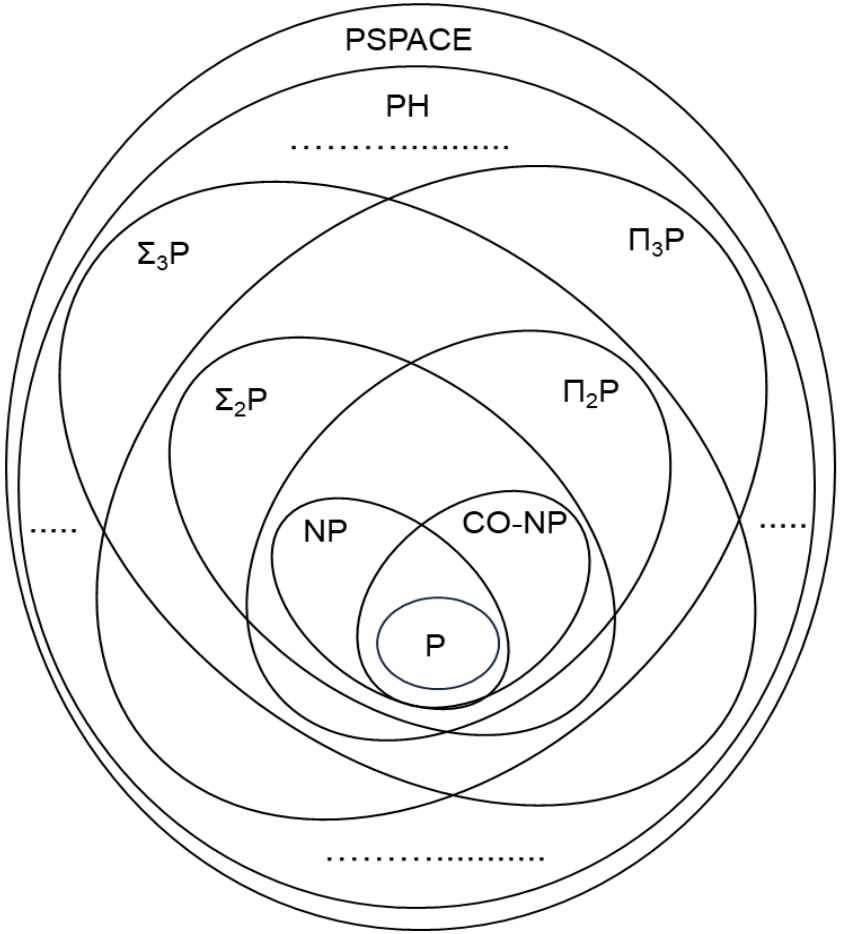
- La conjetura más aceptada es que **las inclusiones son estrictas**, o como se dice habitualmente, que **PH *no colapsa*.**

- Se dice que PH **colapsa en el nivel i** si existe algún i tal que $\Sigma_i P = PH$, es decir, si a partir de un determinado i se cumple $\Sigma_i P = \Sigma_{i+1} P = \ldots$

- Se prueba que **si P = NP, PH colapsa en el nivel 0, es decir, P = PH**

- Aun valiendo P ≠ NP, **podría suceder que PH colapse**.

- Se prueba además que **PH ⊆ PSPACE,** y también en este caso la conjetura más aceptada es **que la inclusión es estricta.**



La conjetura aceptada es PH ⊂ PSPACE

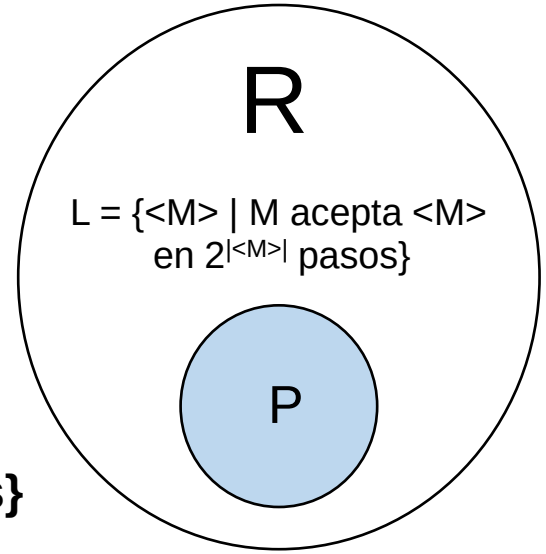
Importante: se prueba que ISO no es NP-completo a menos que la jerarquía polinomial colapse.

5. Prueba de que existen lenguajes recursivos fuera de la clase P

- **El lenguaje L = {<M> | M acepta <M> en $2^{|<M>|}$ pasos} pertenece a R – P.**

- Intuitivamente, decidir si una MT M acepta una cadena w en $2^{|w|}$ pasos no puede llevar menos de $2^{|w|}$ pasos. La prueba de que L ∈ R es muy sencilla (ejercicio).

- Para probar que **L ∉ P** supondremos que no y llegaremos a contradicciones:
  - **Partimos entonces de L ∈ P.**
  - Así, también L^C ∈ P (¿por qué?), con **L^C = {<M> | M no acepta <M> en $2^{|<M>|}$ pasos}**
  - Sea **M^C** una MT que decide L^C en tiempo polinomial
  - Veamos qué sucede en particular **cuando M^C procesa la cadena <M^C>:**



prueba por diagonalización
(L separa P de R)

  - **Si <M^C> ∈ L^C** entonces, como M^C tarda tiempo polinomial, M^C acepta <M^C> en $|<M^C>|^k$ pasos, k constante, y así M^C acepta <M^C> en $2^{|<M^C>|}$ pasos.
    Pero entonces, por la definición de L^C, **<M^C> ∉ L^C** (contradicción).

  - **Si <M^C> ∉ L^C** entonces M^C no acepta <M^C>, y así, en particular, M^C no acepta <M^C> en $2^{|<M^C>|}$ pasos.
    Pero entonces, por la definición de L^C, **<M^C> ∈ L^C** (contradicción).

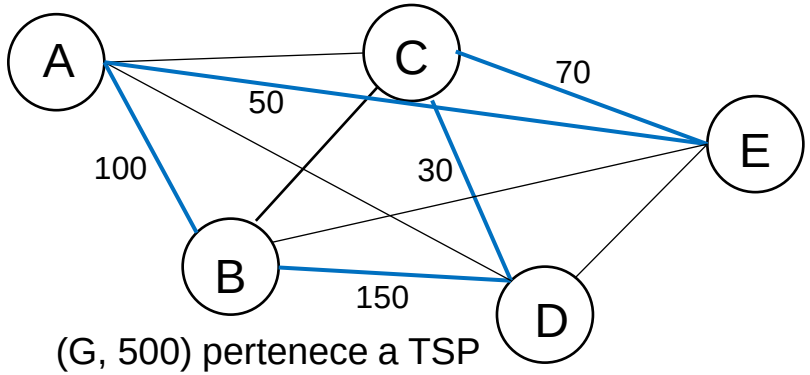
  - En conclusión, **M^C no puede existir,** lo que significa que **L^C ∉ P**, y **por lo tanto L ∉ P.**

Clase práctica 8

- **Ejemplo. El problema del viajante de comercio revisitado.**

TSP = {(G, B) | G es un grafo completo ponderado (arcos con números) y tiene un circuito de Hamilton (CH) cuyos arcos suman ≤ B}.

OTSP: “Hallar un CH mínimo de un grafo completo ponderado G”.



(G, 500) pertenece a TSP

<u>Se cumple que OTSP es tan o más difícil que TSP (intuitivo):</u>

Es decir, si se puede encontrar en tiempo poly(n) un CH mínimo de un grafo completo ponderado G, entonces también se puede decidir en tiempo poly(n) si G tiene un CH cuyos arcos suman ≤ B (ejercicio).

<u>También se cumple que TSP es tan o más difícil que OTSP (menos intuitivo):</u>

Es decir, si se puede decidir en tiempo poly(n) si un grafo completo ponderado G tiene un CH cuyos arcos suman ≤ B (MT M_{TSP}), también se puede encontrar en tiempo poly(n) un CH mínimo de G (MT M_{OTSP}).

La MT M_{OTSP} referida se comporta de la siguiente manera. Dado un grafo completo ponderado G, hace:

Paso 1. Obtiene, invocando a la MT M_{TSP} , la suma **N** de los arcos de un CH mínimo de G.
Paso 2. Utilizando N, encuentra un **CH mínimo** de G.

Sigue en la hoja siguiente

Paso 1. Obtención de la suma N de los arcos de un CH mínimo de G:

- M_{OTSP} invoca a M_{TSP} con cadenas (G, N) varias veces, **sin modificar G pero modificando N:**
- Como el valor de N, codificado en binario, es a lo sumo 2^n siendo n = |G| (¿por qué?),
  entonces por **búsqueda binaria**, arrancando por ejemplo con la cadena (G, $2^n/2$),
  N se puede obtener luego de $O(\log_2 2^n)$ = **O(n) invocaciones** (¿qué hace M_{OTSP} en cada iteración?)

Paso 2. Obtención de un CH mínimo de G utilizando N:

- M_{OTSP} invoca a M_{TSP} con cadenas (G, N) varias veces, ahora **sin modificar N pero modificando G:**
- En cada invocación, **modifica el número de un arco distinto de G, asignándole el valor N + 1:**
  Si M_{TSP} acepta, significa que dicho arco no forma parte de un CH mínimo, y por lo tanto no hace nada.
  Si M_{TSP} rechaza, significa que dicho arco forma parte de un CH mínimo, así que en este caso **marca el arco** (compondrá el CH mínimo que va a devolver) **y le restituye su número original**.
- M_{OTSP} procesa de esta manera todos los arcos, y por lo tanto efectúa |E| = **O(n) invocaciones**.

En verdad, si G tiene más de un CH mínimo, M_{TSP} en el paso 2 puede aceptar aún cuando el arco modificado forme parte de un CH mínimo. ¿Por qué de todos modos el algoritmo es correcto?

Claramente, M_{OTSP} resuelve OTSP y lo hace en tiempo poly(n)