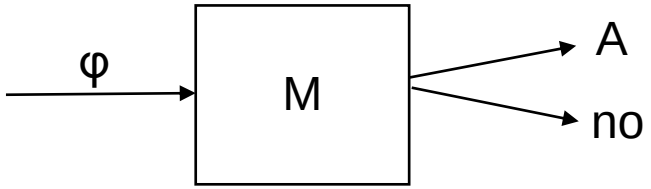
Clase teórica 8

Misceláneas de Teoría de la Computación I

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, para el problema de búsqueda asociado a SAT, conocido como **FSAT**, la MT M asociada, dada una fórmula booleana φ, M hace:

  (a) Devuelve una asignación A que satisface φ, si existe.
  (b) Responde no, si no existe.



M resuelve FSAT

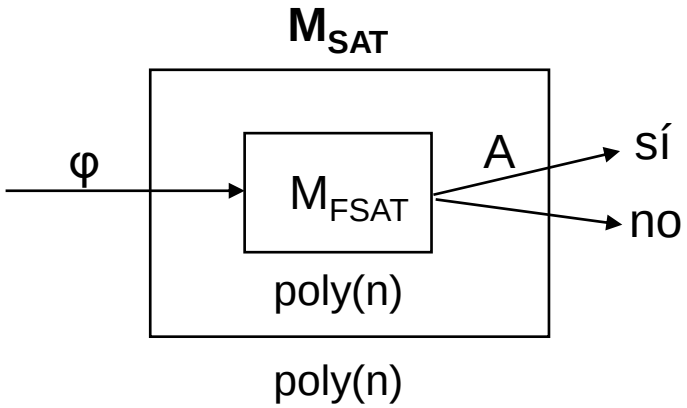
- Las clases de problemas de búsqueda asociadas a las clases de lenguajes P, NP, NPC, etc., se identifican con **FP, FNP, FNPC,** etc. Por ejemplo, **FSAT ∈ FNPC**.
- En general, **decidir la existencia** de una solución es más fácil que **encontrar una.** Pero considerando, por ejemplo, **los lenguajes NP-completos y los problemas FNP-completos**, **¿realmente es así?**

- **Ejemplo 1. Encontrar una solución en el problema FSAT es tan o más difícil que decidir su existencia.**

  Sea una MT M_{FSAT} que puede encontrar una asignación A que satisface φ en tiempo poly(n).

  Vamos a construir a partir de ella una MT M_{SAT} que puede decidir si existe una en tiempo poly(n):

  Dada φ, M_{SAT} ejecuta M_{FSAT} y acepta sii M_{FSAT} devuelve una asignación A.



M_{SAT} decide SAT en tiempo poly(n).
FSAT es tan o más difícil que SAT.
Si FSAT ∈ FP entonces SAT ∈ P.

- **Ejemplo 2. Relación entre encontrar una solución en el problema FSAT y decidir su existencia.**

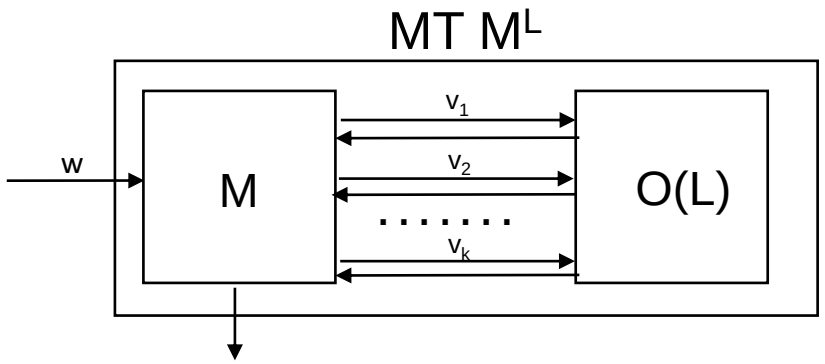
  Sea ahora una MT M_{SAT} que puede decidir si existe una asignación A que satisface φ en tiempo poly(n).

  ¿Se puede construir a partir de ella una MT M_{FSAT} que puede encontrar una en tiempo poly(n)?

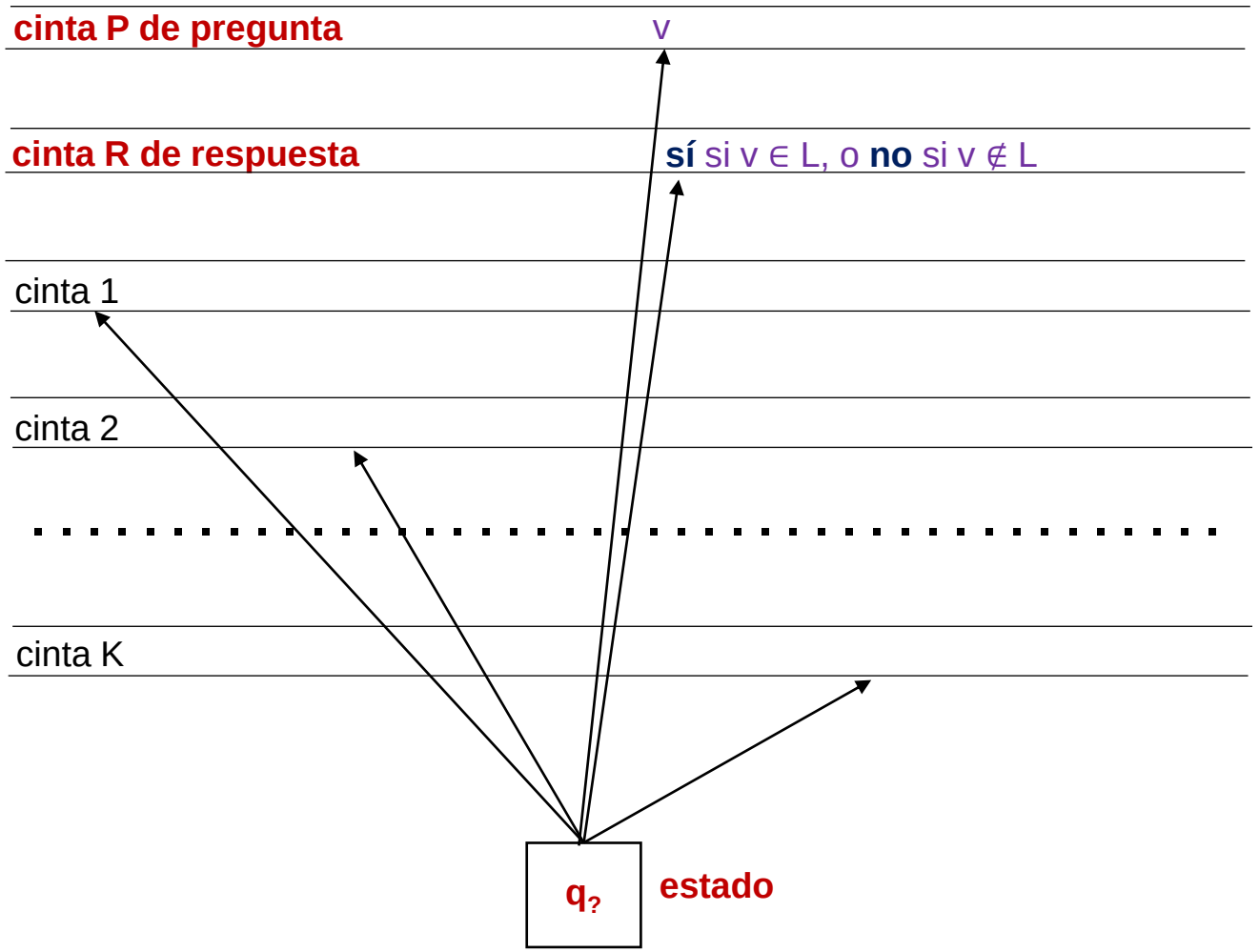
MT con oráculo

- **Una MT M con un oráculo del lenguaje L** es una MT M común pero con dos cintas y un estado especiales (ver dibujo):

  Si una cadena v está en su cinta de pregunta P y su estado es $q_?$, en el paso siguiente recibe en su cinta de respuesta R un **sí** si v ∈ L o un **no** si v ∉ L.



M^L es una MT con un oráculo O(L)

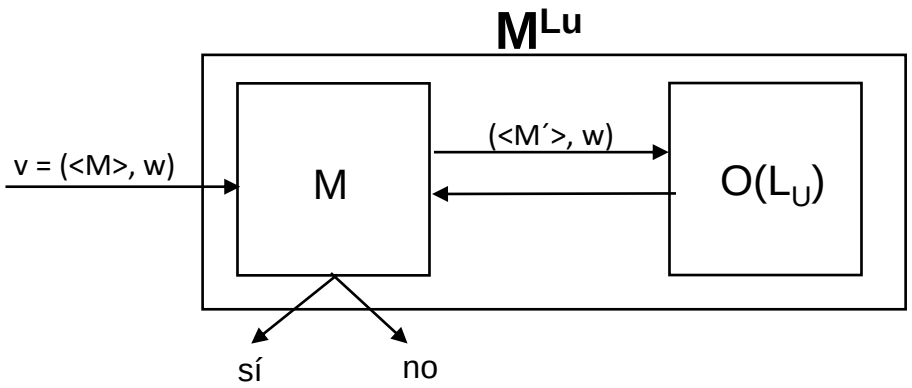


Las MT con oráculo las creó Turing para su tesis doctoral de 1939 sobre la lógica de ordinales.

- **Ejemplo 3. Máquina de Turing con oráculo.**

  La siguiente MT M^{Lu} decide HP. Dada una entrada v, M^{Lu} hace:

  1. Si v ≠ (<M>, w), rechaza.
  2. Transforma (<M>, w) en (<M´>, w) como hicimos cuando probamos HP ≤ L_U:
     los estados q_R de M se reemplazan por estados q_A en M´ para que (<M>, w) ∈ HP sii (<M´>, w) ∈ L_U
  3. Copia (<M´>, w) en la cinta P y pasa al estado $q_?$
  4. Si O(L_U) responde sí (no), acepta (rechaza).



M^{Lu} decide HP.
L_U es tan o más difícil que HP.
Si L_U ∈ R entonces HP ∈ R.

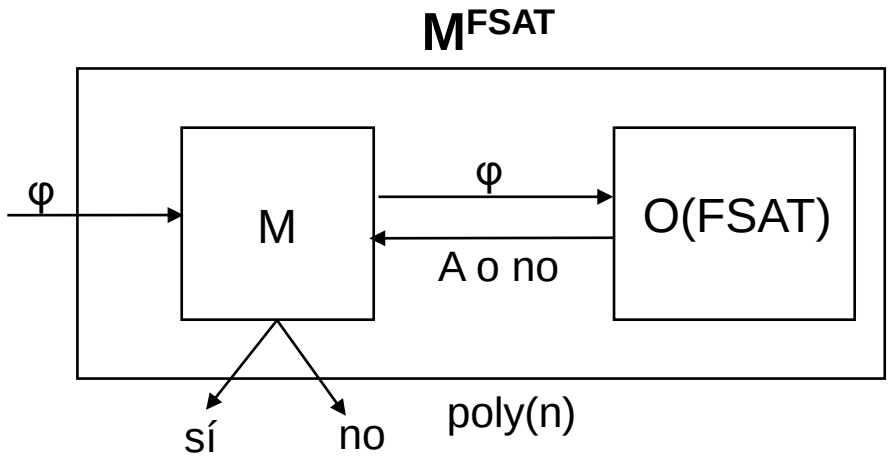
- Se define que la MT M^{Lu} es una **Turing-reducción** de HP a L_U. Se expresa así: **HP ≤^{T} L_U.**

- Como con las reducciones polinomiales, se cumple: **si A ≤^{T} B y B ∈ R entonces A ∈ R.**

Cook-reducciones

- Son una variante de las Turing-reducciones. Son de **tiempo polinomial**, y además **relacionan problemas de búsqueda o decisión.**

- Repetimos el **ejemplo 1:**



M^{FSAT} decide SAT en tiempo poly(n).
FSAT es tan o más difícil que SAT.
Si FSAT ∈ FP entonces SAT ∈ P.

M^{FSAT} es una Cook-reducción de SAT a FSAT. Se expresa así: **SAT $\leq^C$ FSAT.**

Claramente M^{FSAT} decide SAT, y lo hace en tiempo poly(n):

validación sintáctica de φ, copia en la cinta de pregunta P, chequeo de la respuesta de O(FSAT).

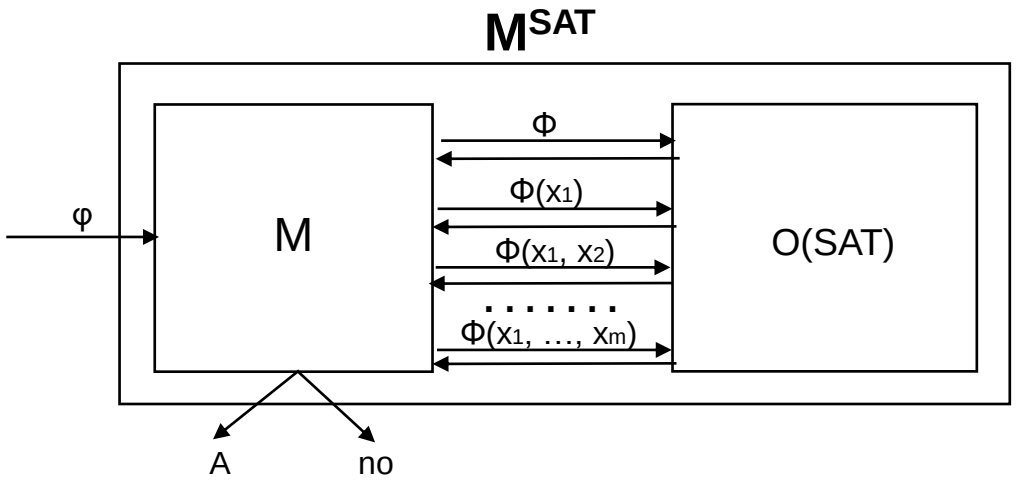
- Volviendo a la pregunta del **ejemplo 2**, reformulada ahora en términos de una Cook-reducción:

¿Se cumple FSAT $\leq^C$ SAT?

- **Ejemplo 4. Decidir la existencia de una solución en FSAT es tan o más difícil que encontrarla.**

La siguiente MT M^{SAT}, dada una fórmula booleana φ con variables x_1, …, x_m, resuelve FSAT en tiempo poly(n):

1. Invoca a O(SAT) con φ. Si O(SAT) responde no, entonces **responde no**.
2. Invoca a O(SAT) con φ y con x_1 = verdadero.
   Si O(SAT) responde sí, **fija para x_1 el valor verdadero**, y si responde no, **fija para x_1 el valor falso**.
3. Invoca a O(SAT) con φ con el valor de x_1 obtenido y con x_2 = verdadero.
   Si O(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 A obtenida**.



M^{SAT} resuelve FSAT en tiempo poly(n).
SAT es tan o más difícil que FSAT.
Si SAT ∈ P entonces FSAT ∈ P.

M^{SAT} resuelve FSAT (trivial)

M^{SAT} tarda tiempo poly(n): hace O(n) invocaciones al oráculo y cada una la procesa en tiempo poly(n).

- **En síntesis:**

  Se cumple tanto SAT $\leq^C$ FSAT como FSAT $\leq^C$ SAT.

  SAT y FSAT son **Cook-equivalentes**.

  En otras palabras, **SAT y FSAT son igualmente difíciles**, uno se resuelve eficientemente sii el otro se resuelve eficientemente.

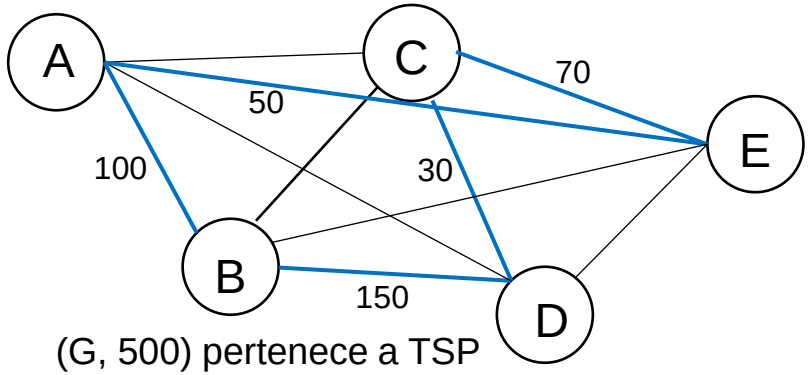
  Todos los problemas FNP-completos tienen esta propiedad, lo que confirma la relevancia del estudio de los problemas de decisión.

  La propiedad también se cumple en problemas no FNP-completos, como por ejemplo el problema de los grafos isomorfos.

- **Ejemplo 5. El problema del viajante de comercio revisitado.**

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

FTSP: “Hallar un CH mínimo de un grafo completo G con arcos con números”.



(G, 500) pertenece a TSP

Se cumple TSP $\leq^{C}$ FTSP:

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

También se cumple FTSP $\leq^{C}$ TSP:

Si se puede decidir si un grafo G tiene un CH cuyos arcos suman ≤ B en tiempo poly(n), también se puede encontrar un CH mínimo de G en tiempo poly(n).

La MT M M^{TSP} que resuelve FTSP en tiempo poly(n) se comporta de la siguiente manera. Dado un grafo completo G con arcos con números, M^{TSP} hace:

Paso 1. Obtiene la suma **N** de los arcos de un CH mínimo de G.
Paso 2. Utilizando N, encuentra un **CH mínimo** de G.

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

- M^{TSP} invoca al oráculo O(TSP) con los parámetros (G, N) varias veces, **sin modificar G pero sí 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 los parámetros (G, $2^n/2$), N se puede calcular luego de $O(\log_2 2^n)$ = **O(n) invocaciones.**

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

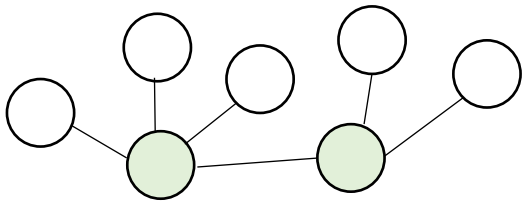
- M^{TSP} invoca al oráculo O(TSP) con los parámetros (G, N) varias veces, **sin modificar N pero sí G.**
- En cada invocación **modifica el número de un arco distinto de G, asignándole el valor N + 1:**
  Si O(TSP) acepta, significa que dicho arco no es parte de un CH mínimo.
  Si O(TSP) rechaza, significa que dicho arco es parte de un CH mínimo. En este caso **marca el arco (es parte del CH mínimo que se va a devolver) y le devuelve su número original**.
- M^{TSP} procesa todos los arcos al cabo de |E| = **O(n) invocaciones**.

- Claramente, M^{TSP} resuelve FTSP y lo hace en tiempo poly(n). Así, **FTSP $\leq^C$ TSP.**
- Por lo tanto, **FTSP y TSP son Cook-equivalentes**.

Aproximaciones polinomiales (para búsquedas de óptimos)

Ejemplo 6. Búsqueda del cubrimiento de vértices mínimo de un grafo (OCV).

- Problema de decisión: CV = {(G, K): G tiene un cubrimiento de vértices de tamaño K}.



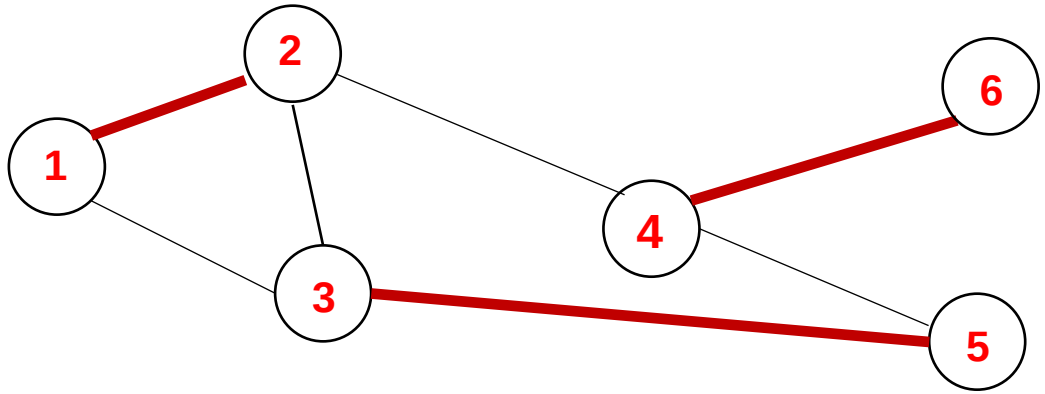
Cubrimiento de tamaño 2.

- CV es NP-completo. Si P ≠ NP, entonces OCV ∉ FP (ejercicio).
- **La idea es remediarlo construyendo un algoritmo de tiempo polinomial que produzca una solución "buena", "cercana al óptimo". En otras palabras, una aproximación polinomial.**

- La siguiente MT, dada una entrada G = (V, E), genera en tiempo poly(n) un cubrimiento de vértices V´ de G **cercano al mínimo** (luego indicamos **cuán cercano**):

1. V´ := ∅ y E' := E.
2. Si E' = ∅, acepta.
3. E' := E' – {(i, j)}, siendo (i, j) algún arco de E'.
4. Si i ∉ V´ y j ∉ V´, entonces V´ := V´ U {i, j}.
5. Vuelve al paso 2.

Es decir, la MT genera un CV V´ **con los vértices de los arcos no adyacentes** que encuentra. Por ejemplo:



A = {(1, 2), (3, 5), (4, 6)}

V´ = {1, 2, 3, 5, 4, 6}

- Sea A el conjunto de los arcos no adyacentes encontrado. Naturalmente, **todo cubrimiento C de G tiene que incluir por lo menos un vértice de todo arco de A** (si no, C no sería un cubrimiento).
- De esta manera, para todo cubrimiento C, debe cumplirse |C| ≥ |V']/2, y así, |V´| ≤ 2.|C|, es decir, |V'| es **a lo sumo el doble del óptimo** (en el ejemplo, el cubrimiento mínimo mide 3, p.ej. C = {2, 3, 4}).
- La MT construida trabaja en tiempo O(|E|.|V|) = **poly(n).**

- Dada una cadena w,
  - si **opt(w)** es la medida de la solución óptima para w,
  - y **m(M(w))** es la medida de la solución obtenida por la MT M para w,
  - se define el **error relativo** ε_w de M con respecto a w del siguiente modo:

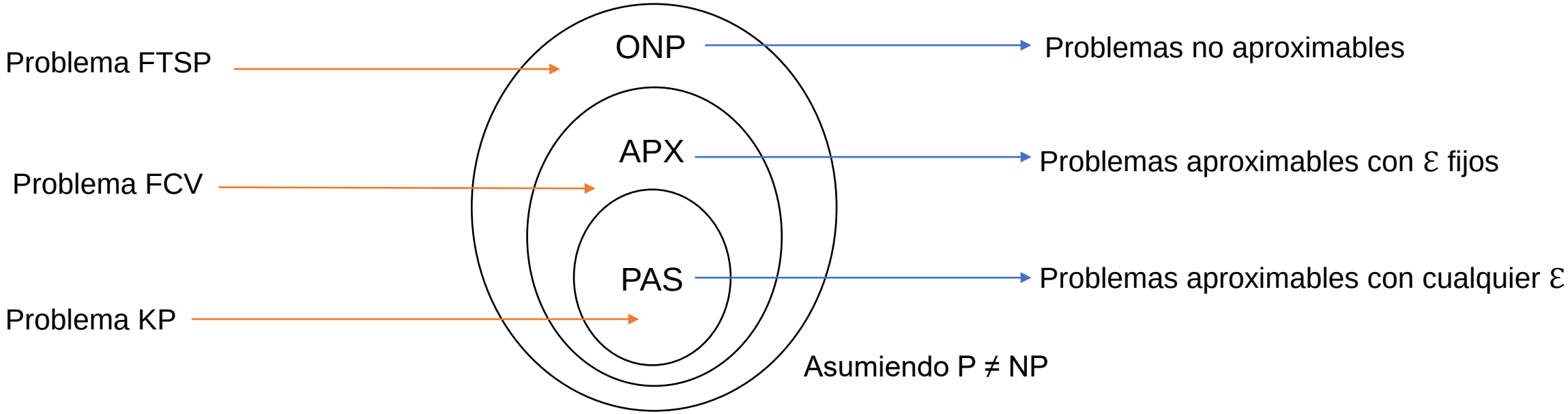
$$\varepsilon_w = \frac{|m(M(w)) - opt(w)|}{\max(m(M(w)), opt(w))}$$

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

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

- Asumiendo P ≠ NP:
  - Hay problemas aproximables con **cualquier** ε, problemas aproximables con ε **fijos** (como el que vimos), y problemas no aproximables (**no existen MT que los resuelvan con ningún** ε).
  - Hay una jerarquía de problemas correspondiente:
    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).

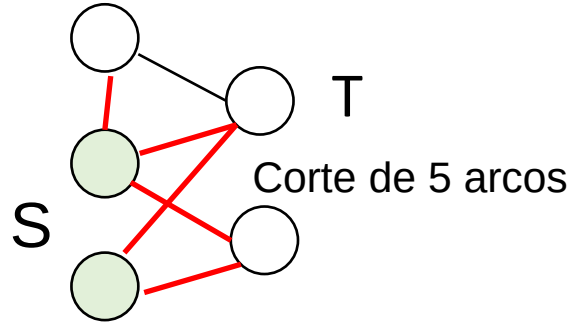
- Algunos problemas representativos de la jerarquía mencionada son los siguientes:



- El **problema de la mochila** (o KP por *Knapsack Problem*) plantea optimizar el armado de una mochila:
  - Se define el peso máximo que soporta un individuo.
  - Se considera un conjunto de objetos para colocar en la mochila, cada uno con peso y volumen.
  - El problema consiste en maximizar el volumen ocupado sin exceder el peso máximo.
  - KP es más fácil que FCV, a su vez más fácil que FTSP, en términos de las aproximaciones polinomiales.
  - También en este caso se plantean problemas completos, usando reducciones de determinadas características (APX-reducciones). No se conocen muchos problemas completos en este marco.

- **Ejemplo 7. Búsqueda del máximo corte de un grafo (OMC).**

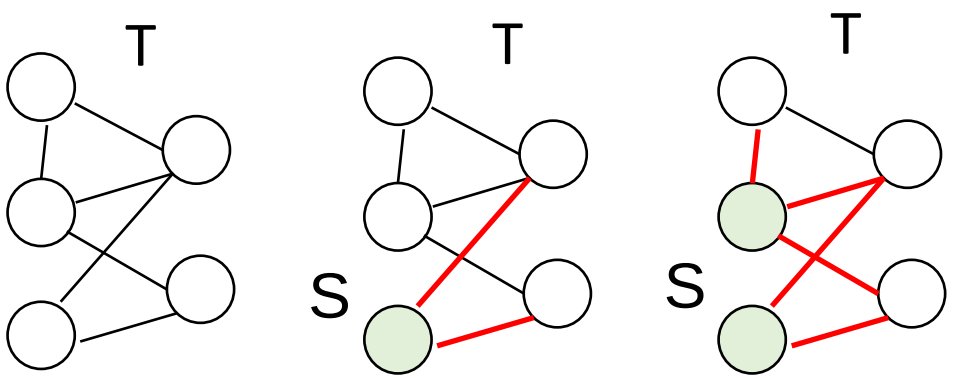
  Dado un grafo G = (V, E), un **corte** de G es una partición {S, T} de V.
  Un **arco de corte** conecta un vértice de S con un vértice de T.
  El problema consiste en encontrar el **corte con el máximo número de arcos**.



- El problema de decisión se define con el lenguaje MC = {(G, K) | G tiene un corte de K arcos}.
  Se prueba que MC es NP-completo. Vamos a construir una **aproximación polinomial de umbral 1/2.**

- Dado G = (V, E), S = ∅ y T = V, la MT M hace:

  1. Si moviendo un vértice i de S a T o de T a S crece el nro de arcos de corte, entonces mover i.
  2. Volver al paso 1.



- El tiempo de M es **polinomial** porque en cada paso (1) a lo sumo se agregan |E| arcos al corte.

- El corte obtenido es **mayor igual que la mitad del máximo**:

  Para todo vértice i se cumple que **el número de arcos de corte que salen de i es al menos el número de arcos que no son de corte que salen de i** (si no, i estaría en el otro conjunto). A partir de esto se obtiene el umbral 1/2.

Anexo de la clase teórica 8

Misceláneas de Teoría de la Computación I

Otros usos de los oráculos en la complejidad temporal

- Sean P^L y NP^L las clases asociadas a P y NP tales que las MT asociadas tienen un oráculo de L.
  1. Un camino para probar P = NP podría ser partir de un lenguaje difícil L, ir comparando P^L vs NP^L a medida que se va simplificando L, hasta llegar a un L tan fácil que permita concluir P = NP.
  2. Otro camino posible para probar P = NP: si NP_p^L es la clase de lenguajes en los que las MT asociadas invocan a L una cantidad poly(n) de veces, se prueba que P = NP sii para todo L se cumple $P^L = NP_p^L$.

- Existen lenguajes A y B tales que $P^A = NP^A$ y $P^B \neq NP^B$ (**Teorema de Baker, Gill y Solovay**). Esto sugiere qué tipo de técnicas utilizar y cuáles no para resolver la relación entre P y NP. Por ejemplo:
  1. Se prueba que si toda MT M que verifica un lenguaje L de NP en tiempo poly(n) puede ser **simulada** con una MT M´ que decide L en tiempo poly(n) (y así P = NP), entonces también vale $P^L = NP^L$ para todo L, y así en particular para B (absurdo). Por lo tanto, **la técnica de simulación no aplica en este caso**.
  2. Se prueba que si se encuentra por diagonalización un lenguaje en NP – P (y así P ≠ NP), entonces también vale $P^L \neq NP^L$ para todo L, y así en particular para A (absurdo). Por lo tanto, tampoco l**a técnica de diagonalización aplica en este caso.**
  3. Un camino viable podría ser encontrar **alguna propiedad algebraica en P que no valga en NP**.

Problemas de conteo (*counting problems*)

- **#P** es la clase de las funciones que calculan la cantidad de soluciones (certificados) que tiene una instancia de un problema de FNP. **Se prueba fácilmente que #P está incluido en FPSPACE**, la clase de funciones asociadas a PSPACE (ejercicio). Ejemplos de funciones de #P: #SAT, #CH, #CLIQUE, etc.

- Si las funciones de #P pudieran calcularse en tiempo polinomial, se cumpliría P = NP (ejercicio).

- Observación: La cantidad de soluciones de un problema de FP no tiene por qué poder calcularse eficientemente. Un ejemplo es el problema del **emparejamiento perfecto (*perfect matching*) en grafos bipartitos**. Un grafo bipartito es un grafo que se puede particionar en dos subconjuntos de vértices, cada uno independiente (no hay aristas que los conectan). En un grafo bipartito, un emparejamiento perfecto es un conjunto de aristas no adyacentes que conectan a todos los vértices de ambos subconjuntos. El problema está en FP, pero calcular la cantidad de emparejamientos es mucho más difícil.

- Se definen también en este caso reducciones entre problemas de conteo A y B. Cada una consiste en un par de funciones f y g computables en tiempo polinomial, tales que, dada una instancia w de A, f(w) es una instancia de B, y g devuelve el número de soluciones de w a partir del número de soluciones de f(w). En términos de estas reducciones se define la **#P-completitud**. Se prueba que #SAT es #P-completo. Así, probar que un problema de conteo es #P-completo, en la práctica es demostrar que es intratable: si se lo resolviera en tiempo polinomial valdría P = NP. En este contexto resulta conveniente usar reducciones **parsimoniosas**, caracterizadas por tener como función g a la función identidad (conservan el número de soluciones). Muchas de las clásicas reducciones del área de NP-completitud son parsimoniosas.

Clase práctica 8

Misceláneas de Teoría de la Computación I

Ejercicio 1. Turing-reducciones.

Se cumple $L_u \leq^T L_\varnothing$.
Sean $L_u = \{(<M>, w) \mid M \text{ acepta } w\}$ y $L_\varnothing = \{<M> \mid L(M) = \varnothing\}$.
La siguiente MT $M^{L_\varnothing}$ decide L_u. Dada una entrada v, $M^{L_\varnothing}$ hace:

1. Si v no tiene la forma (<M>, w), rechaza.
2. Transforma (<M>, w) en $<M_w>$ como hicimos en la reducción $L_u \leq L_{\Sigma^*}$, quedando:
   si $(<M>,w) \in L_u$ entonces $L(M_w) = \Sigma^*$, y si $(<M>,w) \notin L_u$ entonces $L(M_w) = \varnothing$.
3. Copia $<M_w>$ en la cinta P, pasa al estado $q_?$, y acepta sii el oráculo de $L_\varnothing$ rechaza (¿por qué?).

También se cumple la recíproca, $L_\varnothing \leq^T L_u$. La siguiente MT M^{Lu} decide $L_\varnothing$. Dada una entrada w, M^{Lu} hace:

1. Si w no es un código válido <M>, acepta (si la convención es que $L(M) = \varnothing$ cuando <M> no es válido).
2. Genera un código <M'>, tal que M' reemplaza su entrada por <M> y ejecuta una MT M´´ que acepta <M> sii $L(M) \neq \varnothing$ (la construcción de la MT M´´ la vimos en clase).
   Notar que de esta manera, $L(M') = \Sigma^*$ si $L(M) \neq \varnothing$, y $L(M') = \varnothing$ si $L(M) = \varnothing$.
3. Escribe en la cinta P el par $(<M'>, \lambda)$ y pasa al estado $q_?$ (en realidad se puede poner como 2do parámetro cualquier cadena). Si el oráculo de L_u responde sí (no), entonces M^{Lu} rechaza (acepta) (¿por qué?).

- L_u y $L_\varnothing$ son **Turing-equivalentes (o tienen el mismo grado de Turing).** Notar que $L_u \in$ RE pero en cambio $L_\varnothing \in$ CO-RE. Las Turing-reducciones definen otra jerarquía temporal, en términos de **grados de Turing**.

- En cambio, $L_{\Sigma^*} = \{<M> \mid L(M) = \Sigma^*\}$ **no es Turing-equivalente** ni a L_u ni a $L_\varnothing$.

Ejercicio 2. Prueba de que $P^{QSAT} = NP^{QSAT}$ (parte del Teorema de Baker, Gill y Solovay).

- QSAT es PSPACE-completo, por lo tanto más difícil que P y NP asumiendo NP ≠ PSPACE. Intuitivamente, es razonable que P y NP, con QSAT como oráculo, resulten iguales. A continuación vamos a probar $P^{QSAT} = NP^{QSAT}$. Se cumple $\mathbf{P^{QSAT} \subseteq NP^{QSAT}}$ por definición. Para probar $\mathbf{NP^{QSAT} \subseteq P^{QSAT}}$ se demostrará primero $\mathbf{NP^{QSAT} \subseteq PSPACE}$ y luego $\mathbf{PSPACE \subseteq P^{QSAT}}$.

  Nota: Utilizaremos la definición alternativa de NP (en términos de máquinas de Turing no determinísticas).

1. $\mathbf{NP^{QSAT} \subseteq PSPACE}$. Sea $L \in NP^{QSAT}$ y $M_1{}^{QSAT}$ una MTN que acepta L en tiempo poly(n). Dado w, $M_1{}^{QSAT}$ efectúa en cada computación un número poly(n) de invocaciones a QSAT. Toda vez, el tamaño de la pregunta es poly(n) con respecto a |w|, siendo w la entrada. La siguiente MTD M_2 reconoce L en espacio poly(n): simula $M_1{}^{QSAT}$ computación por computación reutilizando espacio, toda invocación a QSAT la reemplaza ejecutando una MTD que acepta QSAT en espacio poly(n), y acepta sii alguna computación acepta.

2. $\mathbf{PSPACE \subseteq P^{QSAT}}$. Sea $L \in PSPACE$ y M_f una MTD que computa una reducción polinomial de L a QSAT (posible porque QSAT es PSPACE-completo). La siguiente MTD $M_3{}^{QSAT}$ acepta L en tiempo poly(n): dada una entrada w, ejecuta M_f para calcular f(w), invoca al oráculo QSAT con f(w), y acepta sii el oráculo acepta.