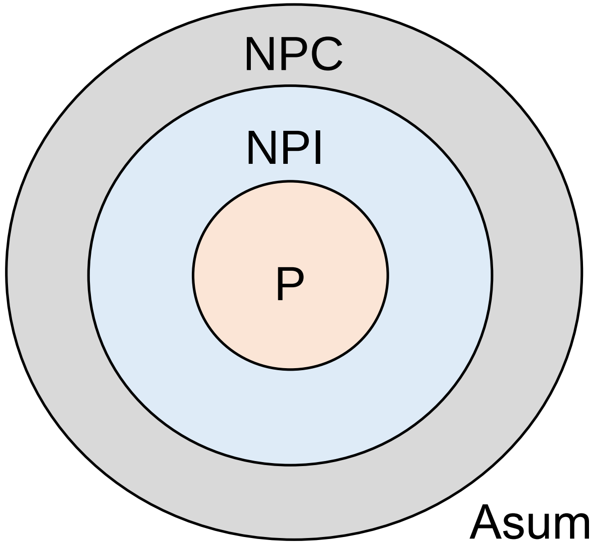
Clase 7. Clases NPI y CO-NP. Complejidad Espacial.

- Asumiendo P ≠ NP, una “foto” más detallada de la clase NP, que ya vimos, es la siguiente:

- Se prueba que si P ≠ NP, entonces **la clase de problemas NPI** ≠ ∅

- ¿Cuándo se “sospecha” que un lenguaje L de NP **pertenece a la clase NPI**?

Clase NP



NPC: los problemas más difíciles

NPI: problemas de dificultad intermedia

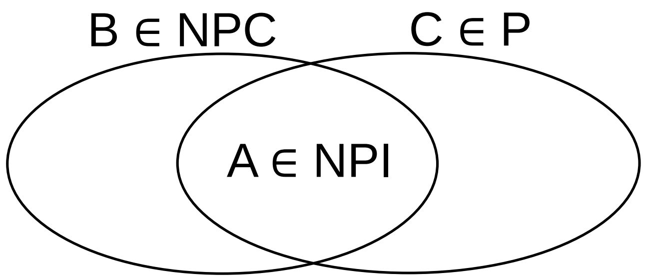
P: los problemas más fáciles

Asumiendo P ≠ NP

1. Cuando no se encuentra una MT que lo acepte en tiempo poly(n). Es decir cuando **no se puede demostrar que L está en P.**

2. Cuando no se encuentra un lenguaje NP-completo L´ tal que L´ α_P L. Es decir cuando **no se puede demostrar que L es un lenguaje NP-completo.**

- Por derivación del **Teorema de Ladner** (1975): Si P ≠ NP, entonces existe un lenguaje A ∈ NPI, obtenido por la intersección de un lenguaje B ∈ NPC y un lenguaje C ∈ P:



Por ejemplo, A podría ser un subconjunto de SAT, con una determinada sintaxis que lo haga más fácil, pero no tanto para que esté en P como el lenguaje 2-SAT.

- ¿Cómo se construiría un conjunto A de NPI en general, a partir de un lenguaje B de NPC? Sacándole a B **muchas cadenas**, pero **no tantas** para que A termine perteneciendo a la clase P, **ni tan pocas** para que A se mantenga en la clase NPC (recordar la densidad de los problemas de NPC).

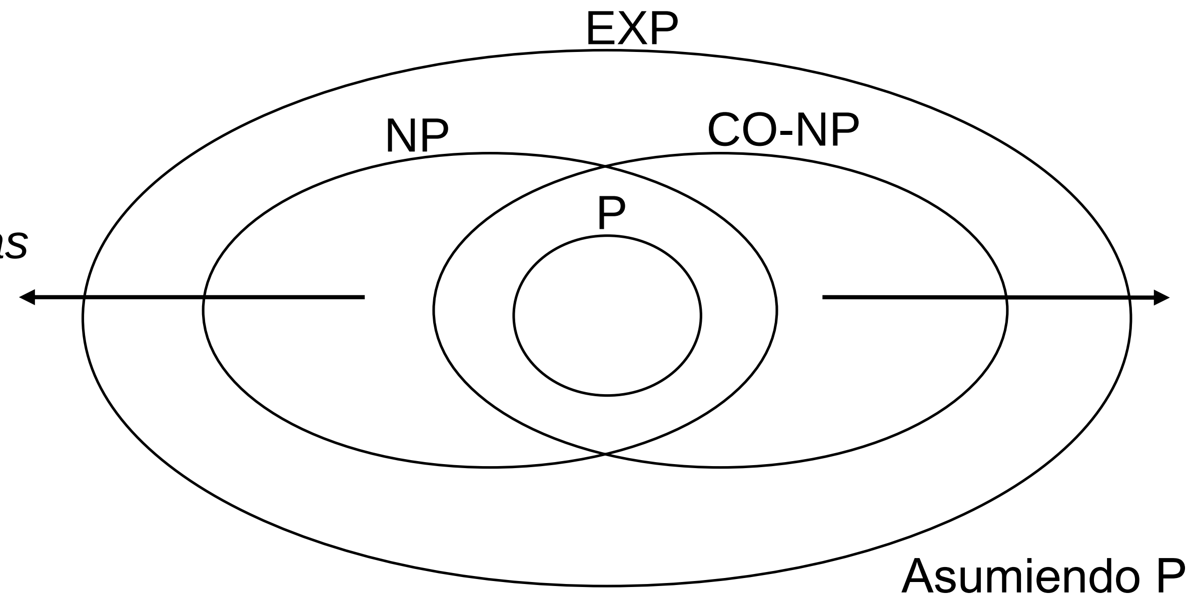
- Podría darse que A sea un lenguaje **disperso.**

- Podrían haber en NPI lenguajes L_1 y L_2 **incomparables**, o sea que no valga $L_1 \alpha_P L_2$ ni $L_2 \alpha_P L_1$.

- Y también podrían haber en NPI secuencias infinitas de la forma: **… $L_5 \alpha_P L_4 \alpha_P L_3 \alpha_P L_2 \alpha_P L_1$**.

- Por otra parte, ya nos hemos referido antes a **la clase de problemas CO-NP** = $\{L \mid L^C \in NP\}$. Habíamos planteado el siguiente mapa:

Por ejemplo los problemas SAT, CH, TSP, CLIQUE, etc.



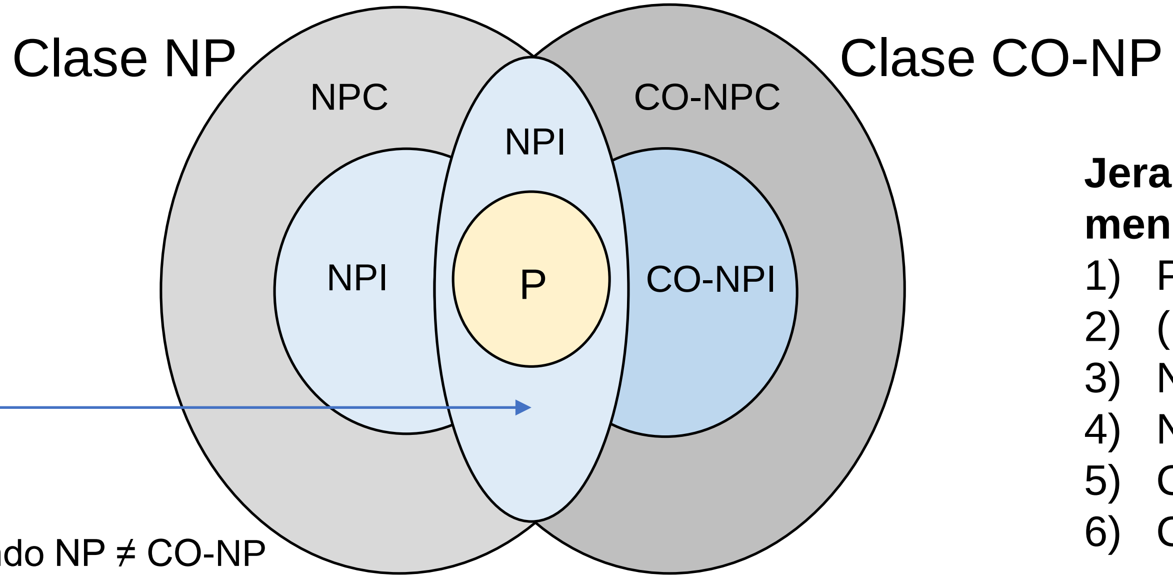
Por ejemplo los problemas SAT ^{C}, CH ^{C}, TSP ^{C}, CLIQUE ^{C}, etc.

Asumiendo P ≠ NP

- Otro ejemplo clásico en CO-NP es VAL = {φ | φ es una fórmula booleana válida, es decir que toda asignación de valores de verdad la satisface}:
  - ✓ Se cumple que $VAL^C \in NP$: dada una fórmula φ, verificar que **alguna** asignación A **no** la satisface tarda tiempo poly(n). A es un **certificado suscinto**.
  - ✓ En cambio para VAL **no habría un certificado suscinto**: φ ∈ VAL sii **todas** las asignaciones la satisfacen (esto no es suscinto, mide exp(n)).
  - ✓ Notar entonces que lo que sí tiene VAL es un **descalificador suscinto** (una asignación A).

- $NP \neq CO\text{-}NP$ es una hipótesis más fuerte que $P \neq NP$: **$NP \neq CO\text{-}NP \longrightarrow P \neq NP$.**
  Prueba (por el contra-recíproco): Si **$P = NP$**, como $P = CO\text{-}P$ (¿por qué?), entonces **$NP = CO\text{-}NP$**.
- Podría suceder que **$P \neq NP$ aún si $NP = CO\text{-}NP$**, pero no parecería.

- También se cumple: **$NP \neq CO\text{-}NP \longrightarrow NPC \cap CO\text{-}NP = \emptyset$.**



El conjunto $NP \cap CO\text{-}NP$ no tiene problemas muy difíciles, o son fáciles o de dificultad intermedia.

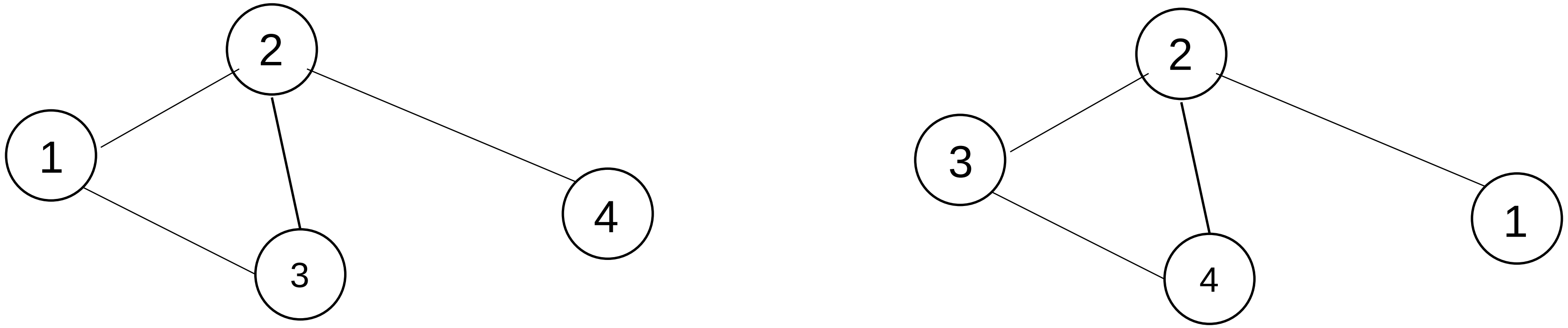
Asumiendo $NP \neq CO\text{-}NP$

Jerarquía de problemas (de menor a mayor dificultad):
1) P
2) $(NP \cap CO\text{-}NP) – P$
3) NPI - CO-NP
4) NPC
5) CO-NPI – NP
6) CO-NPC

- **Prueba** (por el contra-recíproco): Si **$NPC \cap CO\text{-}NP \neq \emptyset$**, entonces **$NP = CO\text{-}NP$**:
  - ✓ Supongamos que existe un L en $NPC \cap CO\text{-}NP$. Llegaremos a la igualdad $NP = CO\text{-}NP$. Probaremos en lo que sigue $NP \subseteq CO\text{-}NP$. La otra inclusión se prueba similarmente (ejercicio).
  - ✓ Sea $L' \in NP$. Veamos que $L' \in CO\text{-}NP$. Se cumple por definición que $L' \alpha_P L$. Y por propiedad de α_P también $L'^C \alpha_P L^C$. Como $L^C \in NP$ por hipótesis, entonces también $L'^C \in NP$, o sea $L' \in CO\text{-}NP$.
  - ✓ Intuitivamente, **los problemas de la región $NP \cap CO\text{-}NP$ son "más fáciles" que los de NPC**.

Problemas candidatos a estar en la clase NPI y eventualmente en NP ∩ CO-NP.

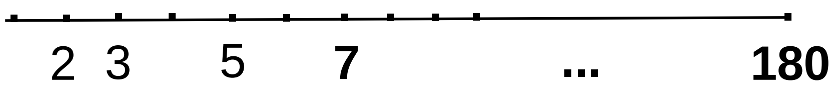
1. El problema de los grafos isomorfos. ISO = {(G_1, G_2) | los grafos G_1 y G_2 son isomorfos, es decir son idénticos salvo por la numeración de sus vértices}



- En este ejemplo, **permutando** la numeración de los vértices 1,2,3,4 de la forma 3,2,4,1, se obtienen dos grafos idénticos. O sea, la permutación Π solución es: Π(1) = 3, Π(2) = 2, Π(3) = 4, Π(4) = 1.
- **ISO "no estaría" en P** (en el peor caso hay que probar con todas las permutaciones de V).
- **ISO está en NP** (la prueba queda como ejercicio - ¿cuál es el certificado suscinto? -).
- **ISO "no estaría" en NPC (no se encuentra un lenguaje NP-completo L que cumpla L α_P ISO).**
- **ISO "no estaría" en CO-NP (**un certificado para ISO^C debe tener todas las permutaciones).

***Curiosidad**. En cambio, el problema de si un grafo G_1 es isomorfo a un subgrafo de G_2 es NP-completo (como que la redundancia de información en G_2 hace que el problema sea más difícil - nuevamente asoma el concepto de densidad de un lenguaje -).*

2. El problema de la factorización. Dado un número natural N, hay que encontrar sus factores primos. P.ej, $180 = 2^2 \times 3^2 \times 5$. Este es un problema de búsqueda, que llevado a la forma de problema de decisión se suele formular así: FACT = {(N,M) | N tiene un factor primo menor que M}.



P.ej., el par (180,7) ∊ FACT

- **FACT "no estaría" en P** (similar al caso del divisor que termina en 3). En el peor caso hay que probar con todos los números menores que M, chequear si son primos, y si lo son, si dividen a N.

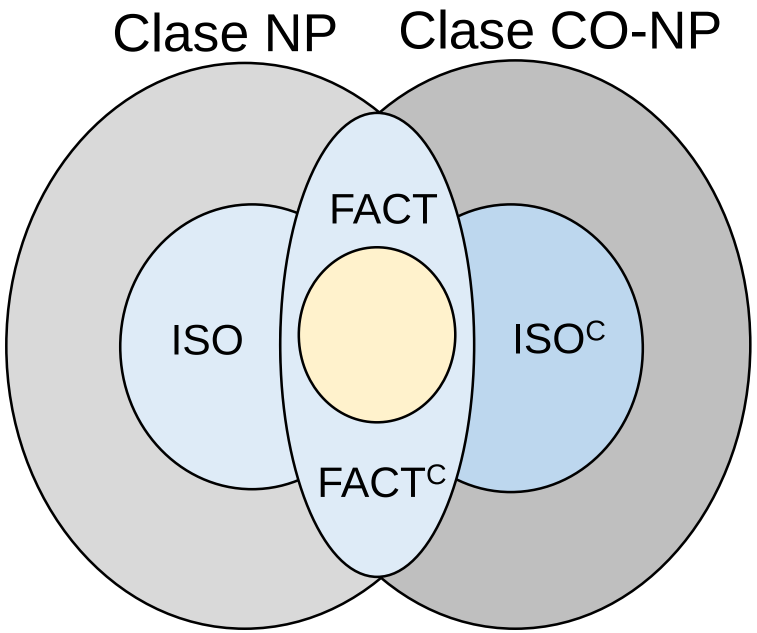
- **FACT está en NP** (la prueba queda como ejercicio - ¿cuál es el certificado suscinto? -).

- **FACT "no estaría" en NPC** (no se encuentra un lenguaje NP-completo L que cumpla $L \; \alpha_P$ FACT).

- **FACT está en NP ∩ CO-NP** (a diferencia de ISO, que no cumpliría esta propiedad).
  La demostración se basa en el Teorema Fundamental de la Aritmética, que establece que todo número tiene una factorización y es única (queda como ejercicio).

- Cuando Peter Shor en 1994 construyó un algoritmo cuántico que resuelve el problema de la factorización en tiempo poly(n), ante la creencia de que era NP-completo los informáticos se esperanzaron en la computación cuántica para finalmente resolver los problemas NP-completos en tiempo eficiente. **La conjetura más firme es que lo cuántico no resuelve la NP-completitud**.

- Así quedarían las ubicaciones de los problemas ISO y FACT en el mapa de la complejidad computacional temporal:



Asumiendo NP ≠ CO-NP

- ISO “no estaría” en P.
- ISO está en NP.
- ISO “no estaría” en NPC.
- ISO “no estaría” en CO-NP.

- FACT “no estaría” en P.
- FACT está en NP.
- FACT “no estaría” en NPC.
- FACT está en CO-NP.

El uso de la factorización en la criptografía. Sistema de clave semi-pública:

- **A** le envía mensajes a **B**. Para asegurar que un *hacker* no lea los mensajes, **B** crea:
  Dos algoritmos públicos, E para encriptar y D para desencriptar.
  Dos claves, *e*, que es pública, y *d*, que es privada (sólo la conoce **B**).

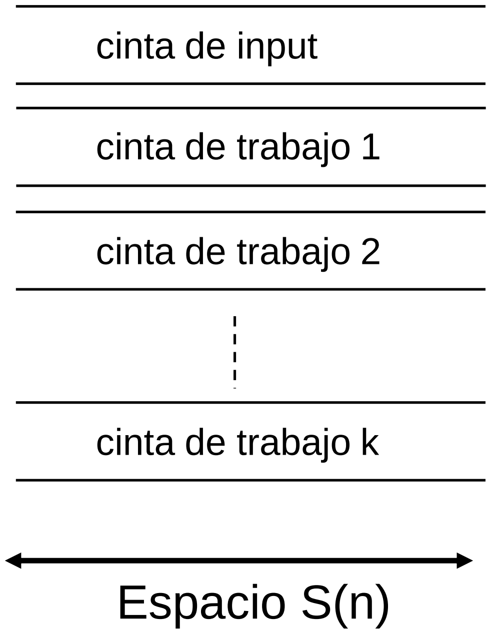
  A le envía encriptado con E un mensaje ***x*** a **B** utilizando la clave pública *e*: $y = E(e, \mathbf{x})$
  B desencripta con D el mensaje ***x*** de **A** utilizando la clave privada *d*: $\mathbf{x} = D(d, y)$

- El sistema se basa en el hecho de que si un número natural N es N_1 x N_2,
  siendo N muy grande y N_1 y N_2 números primos aproximadamente del mismo tamaño,
  es **muy difícil** obtener N_1 y N_2 de N, mientras que es **fácil** obtener N de N_1 y N_2.

- La idea es que la clave *e* incluya N, y que la clave *d* incluya N_1 y N_2.
  Así, se le hará **muy difícil** a un *hacker* obtener los mensajes sin conocer *d*.
  Para **A** y todos los que le envíen mensajes a **B**, encriptarlos con *e* será **fácil**.
  Y para **B** será fácil obtener los mensajes, porque conoce *d*.

- Este mecanismo, de uso masivo en el mundo, se basa en la sospecha de que **el problema de factorización no está en P**. Si se probase que está en P, **habría que cambiar el mecanismo**.

Introducción a la Complejidad Espacial.



- Se utilizan funciones espaciales crecientes $S : N \to N^+$.
- **Una MT trabaja en espacio S(n)** si en todas sus **cintas de trabajo** (no cuenta la **cinta de input**) ocupa a lo sumo S(n) celdas, siendo n = |w|, es decir el tamaño del input.

- La cinta de input es de **solo lectura**. Esto permite considerar espacios **menores que lineales.**

- Podemos utilizar como MT estándar una **MT con 2 cintas, una de input y una de trabajo**:
  - ✓ Por la simulación estudiada en la Clase 1, varias cintas pueden reemplazarse por una sola utilizando pistas o *tracks*, sin aumentar el espacio consumido.

- Que una MT M_1 con 2 cintas trabaje en espacio S(n) no significa que pare siempre, pero seguro existe una MT M_2 equivalente que trabaja en espacio S(n) y para siempre: se construye acotando la cantidad de pasos en $\mathbf{N = (n + 2).S(n).|Q|.\ |\Gamma|^{S(n)}}$ (se puede llevar a la forma $\mathbf{c^{S(n)}}$, con c constante).

- L ∈ SPACE(S(n)) sii existe una MT que acepta L en espacio O(S(n)). Como en el tiempo, también en el espacio **se puede eliminar el factor constante k de k.S(n)** (**teorema de compresión lineal**).

Ejemplo. Volvemos al problema de los palíndromos o "capicúas".

L = {wcw^R, tal que w tiene símbolos *a* y *b* y w^R es la cadena inversa de w}.

L se puede aceptar en espacio O(n) (ejercicio). Pero en verdad alcanza con espacio **O(log n).** Sea la siguiente MT M que con input w hace:

- ✓ Escribir la **cantidad de símbolos a la izquierda** de c, como **contador i,** en la cinta de trabajo 1.
- ✓ Escribir la **cantidad de símbolos a la derecha** de c, como **contador j,** en la cinta de trabajo 2.
- ✓ Si i ≠ j, o si hay símbolos fuera de {a, b, c}, o si no hay exactamente un símbolo c, **rechazar**.
- ✓ Copiar el **símbolo 1 de la parte izquierda** de w en la cinta de trabajo 3 y el **símbolo j de la parte derecha** de w en la cinta de trabajo 4. Si difieren, **rechazar.**
- ✓ Borrar las cintas de trabajo 3 y 4. Copiar el **símbolo 2 de la parte izquierda** de w en la cinta de trabajo 3 y el **símbolo j-1 de la parte derecha** de w en la cinta de trabajo 4. Si difieren **rechazar.**
- ✓ Borrar las cintas de trabajo 3 y 4. Copiar el **símbolo 3 de la parte izquierda** de w en la cinta de trabajo 3 y el símbolo **j-2 de la parte derecha** de w en la cinta de trabajo 4. Si difieren **rechazar.**
- ✓ Y así sucesivamente hasta comparar el **símbolo i de la parte izquierda** de w con el **símbolo 1 de la parte derecha** de w. Si en todos los casos los símbolos fueron iguales, **aceptar.**

La MT M sólo ocupa el espacio de los contadores i y j (que miden **O(log n)**) para moverse a lo largo de la cinta de input, y dos celdas (que miden **O(1)**) para comparar los símbolos. Así, **L ∈ SPACE(log n).**

Nomenclatura y otras características de la complejidad espacial.

- La clase **LOGSPACE** reúne a los lenguajes aceptados por MT que trabajan en espacio O(log n).
  La clase **PSPACE** reúne a los lenguajes aceptados por MT que trabajan en espacio $O(n^k)$.
  La clase **EXPSPACE** reúne a los lenguajes aceptados por MT que trabajan en espacio $O(c^n)$.

- Se utilizan funciones S(n) **espacio-construibles**, es decir que se computan en espacio S(n). Todas las funciones espaciales habituales son construibles: log n, n^k , c^n , n! , n.logn , etc.

- **Tiempo T(n) implica espacio T(n).** ¿por qué?

- **Espacio S(n) implica tiempo $c^{S(n)}$, con c constante** (visto recién). Por lo tanto:

  Si una MT M trabaja en **espacio log n**, entonces trabaja en **tiempo $c^{\log n}$**.
  Y como $c^{\log n} = n^{\log c} = \mathbf{n^k}$**,** con k constante, entonces M trabaja en **tiempo poly(n)**.
  Conclusión: **LOGSPACE $\subseteq$ P**. Un problema es factible, tratable, razonable, considerando complejidad espacial, si su resolución no consume más que espacio logarítmico.

  Y si una MT trabaja en espacio **poly(n),** entonces trabaja en **tiempo $c^{poly(n)}$,** y así: **PSPACE $\subseteq$ EXP**.

- Finalmente, como tiempo T(n) implica espacio T(n), entonces **EXP $\subseteq$ EXPSPACE.**

Jerarquía espacio-temporal.

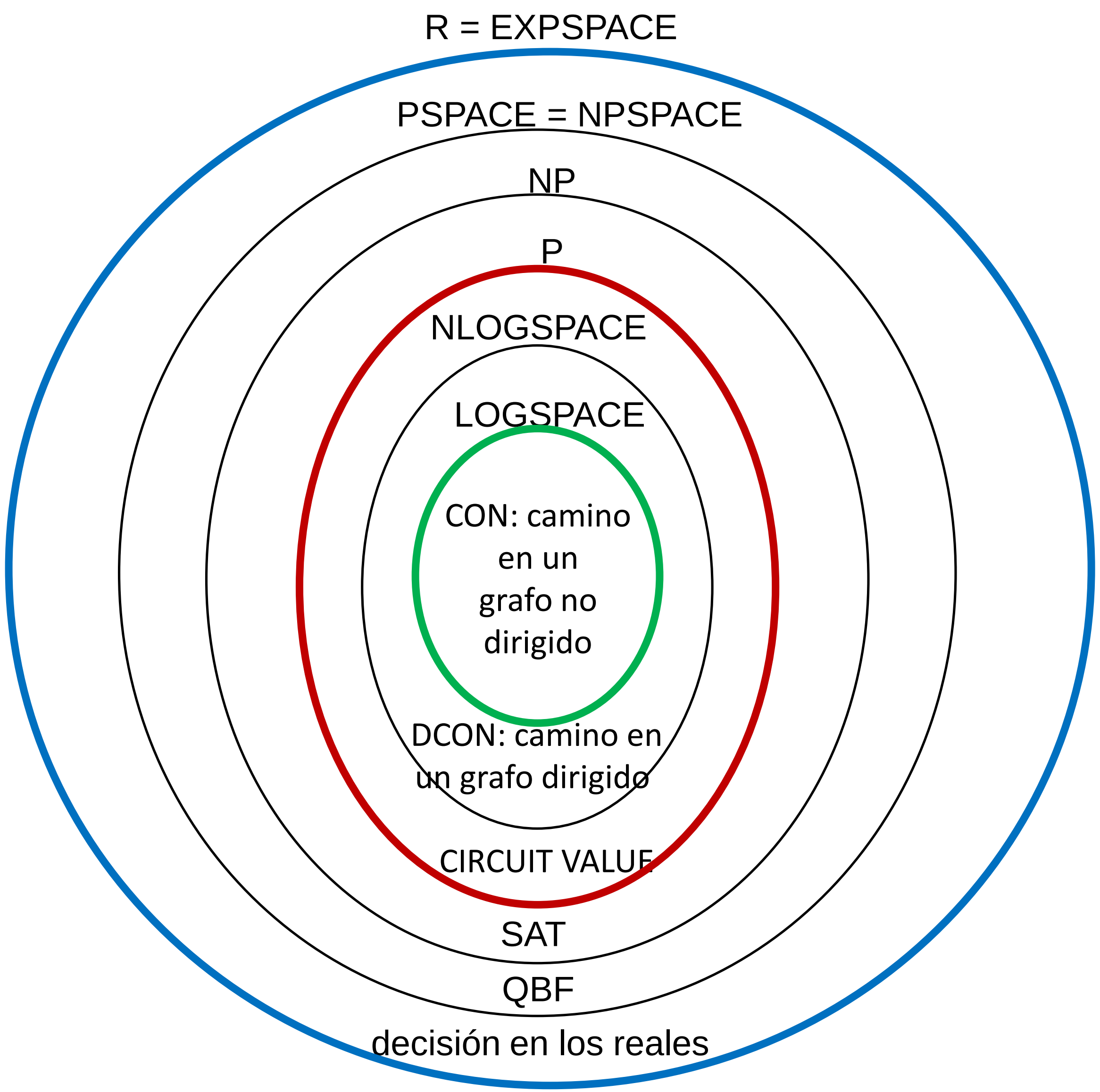
1. Se prueba que PSPACE = NPSPACE (derivado del Teorema de Savitch).

2. Se prueba que también NLOGSPACE $\subseteq$ P.

3. Por (1), NP $\subseteq$ PSPACE (ejercicio).

Dada la inecuación:
LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE, como se prueba que LOGSPACE $\subset$ PSPACE (Teorema de la Jerarquía Espacial), **al menos una de las inclusiones internas es estricta.**

QBF es el problema de satisfactibilidad en las fórmulas booleanas con cuantificadores, y se prueba que es de los **más difíciles de PSPACE**.
CIRCUIT VALUE es el problema de evaluación de un circuito booleano, y se prueba que es de los **más difíciles de P**.
El problema de conectividad en un grafo dirigido u orientado (**DCON**) es de los **más difíciles de la clase NLOGSPACE**.



Ejemplo. La conectividad en los grafos dirigidos u orientados está en NLOGSPACE.

Sea **DCON** el lenguaje que representa el problema de la conectividad en un grafo dirigido. Se define:

DCON = {(G, v_1, v_2) | G es un grafo dirigido y existe un camino en G del vértice v_1 al vértice v_2}.

Vamos a probar que el lenguaje **está en NSPACE(log n) o NLOGSPACE**.

La siguiente **MTN M**, con un contador c en la cinta de trabajo 3 que al comienzo vale 1, trabaja de la siguiente manera a partir de una entrada w (como siempre, se asume que la cantidad de vértices de un grafo es m):

1. Si w no es una entrada válida, rechaza.
2. Hace x := v_1 en la cinta de trabajo 1.
3. Escribe no determinísticamente un vértice z de G en la cinta de trabajo 2.
4. Si (x, z) no es un arco de G, rechaza. Si z = v_2, acepta.
5. Hace c := c + 1 en la cinta de trabajo 3, y si c = m, rechaza. ¿Por qué?
6. Hace x := z en la cinta de trabajo 1, y vuelve al paso 3.

<u>Se cumple que DCON = L(M)</u>.

Sólo si existe un camino de v_1 a v_2 en el grafo G, la MTN M lo irá recorriendo vértice a vértice en la cinta de trabajo 2. En el peor caso, M hace m – 1 iteraciones, hasta aceptar en el paso 4 o rechazar en el paso 5.

<u>M trabaja en espacio no determinístico O(log n)</u>.

La validación sintáctica en el paso 1 se puede hacer en espacio logarítmico (¿por qué?). Con respecto a los pasos 2 a 6, M no construye el camino buscado (esto consumiría espacio lineal), sino que le alcanza con tener en memoria sólo un par de vértices, lo que consume espacio O(log n), con |w| = n. El contador c también ocupa espacio O(log n) (¿por qué?).

Cabe agregar que no hace muchos años que se demostró que la conectividad en los grafos no dirigidos está en LOGSPACE (Omer Reingold, 2008).

Ejemplo. El problema QBF consiste en determinar si una fórmula booleana con cuantificadores y sin variales libres es verdadera. Se prueba que está en PSPACE.

Se define: QBF = {ϕ | ϕ es una fórmula booleana con cuantificadores, no tiene variables libres, y es verdadera}.

Notar que una fórmula booleana ϕ sin cuantificadores, y con variables $x_1, \ldots, x_k$, es satisfactible si y sólo si la fórmula booleana $\exists x_1 \exists x_2 \exists x_3 \ldots \exists x_k (\phi)$ es verdadera, por lo que QBF generaliza SAT (de hecho también se lo denomina QSAT), y así **es NP-difícil**.

QBF "no estaría" en NP. Por ejemplo, para **$\phi = \forall x \forall y \forall z\ (x \wedge y \wedge z)$,** no alcanza con verificar **una** asignación de valores de verdad, sino que **hay que considerar las 2^3 posibilidades**.

La prueba de que **QBF está en PSPACE** se basa en la construcción de una **función recursiva Eval.** Simplificando notación, usaremos Eval con distintas cantidades de argumentos. La idea de la recursión es la siguiente (V es verdadero y F es falso):

Eval(V) = V y Eval(F) = F
Eval(ϕ,$\neg$) = $\neg$Eval(ϕ)
Eval(ϕ_1,ϕ_2,$\wedge$) = Eval(ϕ_1) $\wedge$ Eval(ϕ_2) (el $\vee$ se define de manera similar)
Eval(ϕ,$\forall$x) = Eval(ϕ[x|V]) $\wedge$ Eval(ϕ[x|F])
Eval(ϕ,$\exists$x) = Eval(ϕ[x|V]) $\vee$ Eval(ϕ[x|F])

donde ϕ[x|v] denota la sustitución de la variable x por el valor de verdad v en la fórmula ϕ. El análisis sintáctico de ϕ se puede efectuar claramente en espacio poly(n) (ejercicio). Por otro lado, la cantidad de cuantificadores más la cantidad de conectivos de ϕ es a lo sumo $|\phi| = n$, y así la profundidad de la recursión y el espacio ocupado por los parámetros de una invocación miden O(n). Por lo tanto, **Eval consume espacio $O(n^2)$.**