

# Tema-1-Maquinas-de-Turing-Funcio...



ParmigianoReg



Modelos Avanzados de Computación (Especialidad  
Computación y Sistemas Inteligentes)



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada

**QUIERES  
CONSEGUIR  
15E??**

→ TRÁENOS A TU  
**CRUSH DE APUNTES** ❤  
ANTES DE QUE  
LOS QUEME 🔥



**WUOLAH**

NO  
QUEMES  
TUS  
APUNTES

**GANAS  
0,25 €**

por subir tus  
apuntes en PDF  
a Wuolah

## Tema 1 - Máquinas de Turing, Funciones y Lenguajes Calculables

Profesor Serafín Moral Callejón, Curso 21-22

↓  
si juegas  
con fuego  
te fuegos



\* válido

hasta el 3 de

junio de

2022 o hasta

llegar al

tope de

documentos

para esta

promoción

### Máquina de Turing:

- Una máquina de Turing es una séptupla  $(Q, A, B, \delta, q_0, \#, F)$  en la que
  - $Q$  es un conjunto finito de estados.
  - $A$  es un alfabeto de entrada.
  - $B$  es un alfabeto de trabajo.
  - $\delta$  es la función de transición que asigna a cada estado  $q \in Q$  un símbolo  $b \in B$ , entonces el valor  $\delta(q, b)$  puede ser vacío o una tripleta  $(p, c, M)$  donde  $p \in Q, c \in B, M \in \{I, D\}$  donde  $I$  es Izquierda y  $D$ , derecha.
  - $q_0$  es el estado inicial.
  - $\#$  es el símbolo de los alfabetos  $B, A$  llamado símbolo blanco o vacío.
  - $F$  es el conjunto de estados finales.

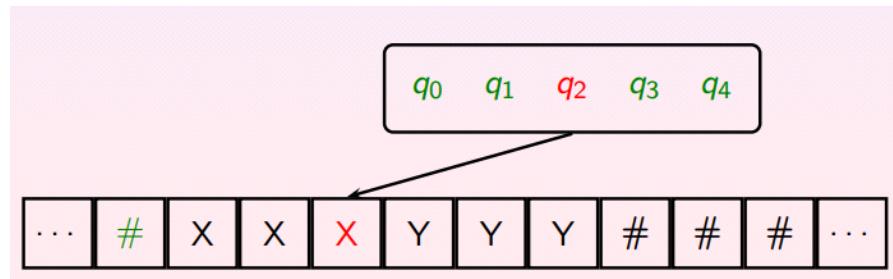
Consideremos la MT

$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, \#\}, \delta, q_0, \#, \{q_4\})$  donde las transiciones no nulas son las siguientes:

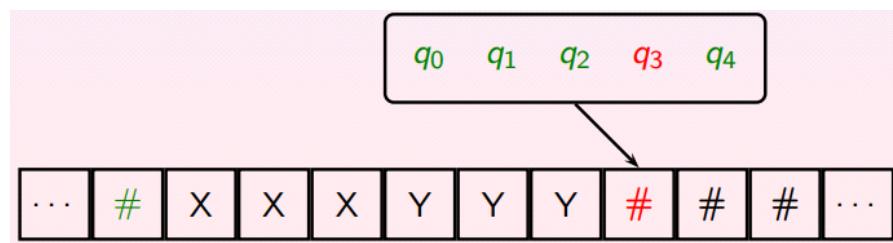
$$\begin{array}{ll} \delta(q_0, 0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) \\ \delta(q_1, 0) = (q_1, 0, D) & \delta(q_1, 1) = (q_2, Y, I) \\ \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) \end{array}$$

### Proceso de cálculo

- La **configuración** de una MT es una tripleta  $(q, w_1, w_2)$  donde:
  - $q$  es el estado en el que se encuentra la máquina.
  - $w_1$  es la representación de la parte de la palabra en la cinta a la izquierda del cabezal, puede ser vacío y se ignoran los blancos consecutivos.
  - $w_2$  es la representación de la parte de la palabra en la cinta a la derecha del cabezal, incluyendo la palabra que se encuentra bajo el cabezal, no puede ser vacía. Se eliminan las casillas blancas consecutivas.



- La configuración aquí sería  $(q_2, XX, YYYY)$ .



- La configuración sería  $(q_3, XXXYYY, #)$

### Configuración Inicial

- Si  $u \in A^*$ , o sea, si  $u$  es cualquier palabra del alfabeto  $A$ , la configuración inicial de una MT  $(Q, A, C, \delta, q_0, \#, F)$  asociada a esa palabra es  $(q_0, \epsilon, u)$  siendo  $(q_0, \epsilon, \#)$  si  $u = \epsilon$ .

### Paso de Cálculo

- Un paso de cálculo hacia la izquierda es cuando se tiene una transición de forma que  $\delta(q, a) = (p, b, l)$ , se dice entonces que la configuración de la MT pasa de  $(q, c_1 \dots c_n, ad_2, \dots, d_m)$  a la configuración  $(p, c_1 \dots c_{n-1}, c_n bd2, \dots, d_m)$ .
  - Se denota como  $(q, c_1 \dots c_n, ad_2, \dots, d_m) \vdash (p, c_1 \dots c_{n-1}, c_n bd2, \dots, d_m)$ .
  - Si  $c_1 \dots c_n = \epsilon$  entonces  $c_1 \dots c_{n-1} = \epsilon, c_n = \#$ .
- Si  $\delta(q, a) = (p, b, D)$  entonces  $(q, c_1 \dots c_n, ad_2, \dots, d_m)$  pasa a  $(p, c_1 \dots c_n b, d_2 d_3, \dots, d_m)$

### Relación de Pasos de Cálculo

- Si  $R$  y  $R'$  son configuraciones de MTs  $M = (Q, A, B, \delta, q_0, \#, F)$  se dice que desde  $R$  se puede llegar en una sucesión de pasos de cálculo a  $R'$ , esto se denota como  $R \vdash R'$ .
  - Esto solamente sucede si y solo si existen una sucesión finita de configuraciones  $R_1, \dots, R_n$  tal que  $R_1 = R$  y  $R' = R_n$  tal que  $\forall i < n$  se cumple que  $R_i \vdash R_{i+1}$ .

### Lenguaje Aceptado por una MT

- Si  $M$  es una MT, entonces el lenguaje aceptado es el conjunto de palabras  $L(M)$  tales que  $u \in L(M)$  si y solo si existen  $w_1, w_2 \in A^*$  y  $q \in F$  tales que  $(q_0, \epsilon, u) \vdash^* (q, w_1, w_2)$ .
  - O sea, desde la configuración inicial asociada a  $u$  se puede llegar mediante una sucesión de pasos de cálculo a una configuración en la que se está en un estado final.

### Lenguaje Recursivamente Enumerable:

- Un lenguaje  $L \subseteq A^*$  se dice recursivamente enumerable (r.e) si y solo si existe una MT  $M = (Q, A, B, \delta, q_0, \#, F)$  tal que  $L(M)$  existe.
- Una MT para cuando en el estado actual y símbolo de la cinta no hay ninguna transición definida.
  - Cuando se llega al estado final  $q \in F$ , se puede suponer que para porque no tiene más transiciones.
- Una palabra es aceptada por la MT cuando la MT para, o bien llega a un estado de aceptación. Podría seguirse ejecutando, pero eso es irrelevante. Se podrá suponer que en cualquier estado final la MT se detendrá.
- Una MT puede no aceptar una palabra, de dos formas distintas:
  - Llegando a un estado no final donde no se tiene una transición para el símbolo de la cinta, la MT para y rechaza.
  - Se puede quedar en un bucle de forma indefinida sin llegar a un estado de aceptación, en este caso nunca se podría llegar a aceptar una palabra, aunque técnicamente no se rechaza tampoco pues la palabra no se acepta ni rechaza.

### Lenguaje Recursivo:

- Un lenguaje recursivo es aquel que es aceptado por una MT que siempre termina: todas las palabras se aceptan o rechazan.
  - Un lenguaje recursivo siempre es recursivamente Enumerable.
  - Estos lenguajes siempre pueden ser resueltos por un algoritmo.
- En el caso de los lenguajes recursivos, se puede suponer que hay dos tipos de estados finales: de aceptación y rechazo, la máquina acepta cuando se llega a un estado de aceptación y rechaza cuando llega a un estado de rechazo.

### Máquinas de Turing Calculadoras:

- Dada una MT  $M = (Q, A, B, \delta, q_0, \#, F)$ , la función  $f$  calculada por esta MT es una función  $f: D \rightarrow B^*$  tal que  $D \subseteq A^*$  es el conjunto de entradas para los que la MT termina y si  $u \in D$ , entonces  $f(u)$  es el contenido de la cinta cuando la MT termina excluyendo lo símbolos en blanco.

#### **Función Parcialmente Calculable**

- Una función  $f$  se dice es parcialmente calculable cuando existe una MT que la calcula.

#### **Función Totalmente Calculable**

- Si una función es parcialmente calculable y  $D = A^*$ , o sea que la MT termina en todas las entradas, se dice que es calculable total.
- Un ejemplo es una MT que calcule la resta de números en unario, la función  $f$  mapea las entradas a una salida particular.

### Programación de Máquinas de Turing:

#### **Recordando símbolos / Memoria**

- Se puede diseñar una MT para que recuerde un símbolo del alfabeto de trabajo o el de entrada.
  - Si se quiere recordar un símbolo de  $B$  cuando está en el estado  $q$ , entonces basta con cambiar el estado  $q$  por las parejas de estados  $[q, b]$  donde  $b \in B$ .
- Si se quieren recordar símbolos en cualquier estado, entonces el conjunto de estados sería el conjunto de parejas  $Q' \times B$  formadas por un elemento  $q \in Q'$  y un símbolo  $b \in B$ .
  - Se puede considerar que el estado  $[q', b]$  donde  $q'$  es el estado básico y  $b$  el símbolo recordado.
- Se pueden escribir MTs de esta forma para ayudar a comprender el significado de los estados y se usa para describir MTs sin llegar al detalle de las transiciones.

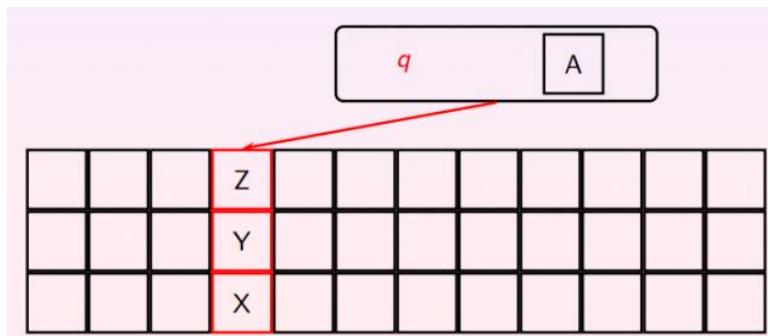
Vamos a hacer una máquina de Turing que reconozca el lenguaje  $01^* + 10^*$ : la Máquina tiene que recordar el primer símbolo leído y después comprobar que nunca más aparezca.

La Máquina es  $M = (Q, \{0, 1\}, \{0, 1, \#\}, \delta, [q_0, \#], \{[q_1, \#]\})$  donde

- $Q = \{q_0, q_1\} \times \{0, 1, \#\}$
- Las posibles transiciones de  $\delta$  son:
  - 1  $\delta([q_0, \#], a) = ([q_1, a], a, D)$  para  $a = 0$  o  $a = 1$ .
  - 2  $\delta([q_1, a], \bar{a}) = ([q_1, a], \bar{a}, D)$ , donde  $\bar{a}$  es el complementario de  $a$  (esto es,  $\bar{a} = 1$  si  $a = 0$  y  $\bar{a} = 0$  si  $a = 1$ ).
  - 3  $\delta([q_1, a], \#) = ([q_1, \#], \#, D)$

### Pistas Múltiples

- Se puede pensar también que una cinta de una MT posee varias casillas en vez de una sola, donde se puede escribir entonces más de un símbolo.
- Tener dos casillas o pistas equivale a suponer que el alfabeto de trabajo está formado por los elementos de  $B \times B$  y tener  $k$  pistas equivale a suponer que el alfabeto de trabajo es  $B^k$ .



### Subrutinas

- Una subrutina en una MT es un conjunto de estados que realiza una acción concreta, en este conjunto de estados habrá un estado inicial y otro estado que sirve como estado de retorno.
- No se añade ninguna funcionalidad nueva, sólo es una forma de organizar los estados de una MT agrupando aquellos que realizan una tarea concreta y suponiendo que siempre podemos movernos a ese conjunto de estados.
- La MT no tiene un sistema de llamadas que permita saber a qué posición y en qué estado hay que volver: la posición se puede recordar con una pista adicional y un símbolo extra que indique la casilla en la que se tiene que posicionar.
  - El estado se puede determinar haciendo varias copias del último estado de la subrutina, una para cada estado al que haya que volver. El número de copias es finito.
- Siempre que se haga un conjunto de estados para una tarea determinada, por ejemplo, desplazar el contenido de todas las casillas a partir de la posición actual un lugar a la derecha, se supondrá que esta tarea siempre la podemos hacer en una MT sin necesidad de especificar los estados.

### Complejidad en Tiempo

- En una MT, el número de pasos o tiempo para una entrada  $u$  es el número de pasos de cálculo entre la configuración de entrada y la última configuración.
- Una MT tiene una complejidad  $t(n)$  en tiempo si para toda entrada de longitud  $n$ , la MT termina en  $t(n)$  o menos pasos.

### Variantes de la MT básica

- Ninguna de estas modificaciones cambia la potencialidad de las MT
- **Extensiones**
  - MTs que se pueden quedar en la misma posición, se introduce el movimiento  $S$  que hace que la

NO  
QUEMES  
TUS  
APUNTES

GANAS  
0,25 €  
por subir tus  
apuntes en PDF  
a Wuolah

↓  
si juegas  
con fuego  
te fregas

\* válido  
hasta el 3 de  
junio de  
2022 o hasta  
llegar al  
tope de  
documentos  
para esta  
promoción

WUOLAH

máquina se quede en la misma parte de la cinta.

- Esto no supone ninguna potencialidad adicional porque un estado con el movimiento  $S$  se puede simular con un estado que se mueva a la derecha seguido de otro que se mueva a la izquierda.
- **MTs con múltiples cintas:** Se tienen multiples cintas, cada una con un cabezal que se mueve independiente de los otros.
- **MTs no deterministas,** hay distintas transiciones que puede realizar una MT con una configuración dada.
- **Limitaciones**
  - **MT con cintas semilimitadas,** o sea, la cinta por la derecha es ilimitada, pero limitada por la izquierda.

#### Máquinas de Turing con Múltiples Cintas:

- Se suponen que tienen  $k$  cintas ilimitadas en las que leer y escribir.
- Se diferencia con la multipista es que la multipista no supone una modificación de la definición de una MT, simplemente es una forma de visualizar la cinta de una MT en la que el alfabeto de trabajo es el producto cartesiano  $B = B' \times B'$ , es decir, cada símbolo está formado por una pareja de símbolos básicos, el tener múltiples cintas si implicará una modificación de la definición.
- Ahora a cada  $\delta(q, b_1 \dots, b_k)$  se le podrá asignar un vector  $(p, c_1, \dots, c_k, M_1, \dots, M_k)$ . Cuando hay múltiples cintas, el cabezal de lectura podrá estar en una posición distinta en cada cinta, por eso se tendrá que especificar el movimiento  $M_i$  que hay que realizar en cada cinta  $i$  además de lo que se ve en la cinta  $b_i$  y lo que se escribe en cada una  $c_i$ , el movimiento podrá ser  $\{I, D, S\}$ .

#### **Configuración**

- Será un vector  $(q, u_1, w_1, u_2, w_2, \dots, u_k, w_k)$  donde  $q$  es el estado en el que está la MT,  $u_i$  es la parte de la palabra a la izquierda del cabezal en la cinta  $i$ , y  $w_i$  la parte de la palabra a partir del cabezal hacia la derecha en la cinta  $i$ , incluyendo el símbolo encima del cabezal.

#### **Lenguaje Aceptado**

- El lenguaje aceptado por una MT multicinta es el conjunto de palabras  $u$  tales que empezando en una configuración en la que la primera cinta está la palabra  $u$  y el resto de cintas son vacías y el cabezal de lectura de la primera cinta está en el primer símbolo de  $u$  y en cualquier casilla de las otras cintas termina en un estado de aceptación.

#### **Función Calculada**

- La función parcial  $f$  calculada por una MT es la función definida en todas las entradas en las que la MT termina, si  $u \in A^*$  es una entrada para la que la MT termina, entonces lo hace con  $f(u)$  como contenido de la última cinta excluyendo blancos.

#### **Equivalencia**

- **Teorema:** Todo lenguaje aceptado por una MT con varias cintas es también aceptado por una MT de una cinta.

#### **Demostración**

- Si se supone que se tiene una MT  $M$  con  $k$  cintas, se puede simular el funcionamiento en una MT  $N$  de una cinta.
- La MT  $N$  tendrá una cinta con  $2k$  pistas, en cada par de pistas se simula una cinta de  $M$ , en una de esas pistas se coloca el símbolo especial  $*$  en el lugar en el que se encuentre el cabezal de la MT  $M$ , en la otra pista, se añade el contenido de esa cinta.
- La MT  $N$  almacenará en su unidad de control los  $k$  símbolos que contiene  $M$ , para eso comienza a revisar la cinta de izquierda a derecha y cada vez que encuentra un símbolo lo almacena en su correspondiente lugar en la unidad de control, posee un contador de que va desde 0 y se aumenta en 1 cada vez que se encuentra un símbolo hasta llegar a  $k$ , que es un valor fijo y se puede almacenar.
- Hecho esto se tiene todo lo necesario para realizar una transición de la MT de  $k$  cintas, para eso se va

- colocando en cada una de las posiciones señaladas y realiza la transición correspondiente, escribiendo el símbolo que corresponde y moviendo la señal de posición del cabezal de lectura.
- Los estados de aceptación de  $N$  corresponden con los estados de aceptación de  $M$ .

#### **Complejidad en Tiempo:**

- Si la MT  $M$  emplea un número de pasos inferior o igual a  $t(n)$  para una entrada de longitud  $n$ , entonces la MT  $N$  de una cinta empleará un número de pasos del orden de  $O(t^2(n))$

### **Máquinas de Turing No Deterministas:**

- Una MTND tiene la misma definición que una MT normal con la única diferencia que ahora  $\delta(q, a)$  puede ser conjunto finito de triplets  $\{(q_1, b_1, M_1), \dots, (q_k, b_k, M_k)\}$ .

#### **Cálculo**

- El cálculo asociado a una MTND se define de forma similar que una MT, ahora la configuración del estado  $q$ , viendo  $a$  en la cinta puede evolucionar en cualquiera de las triplets  $(q_i, b_i, M_i)$ , puede ir a cualquiera  $q_i$ , escribir  $b_i$  y hacer el movimiento  $M_i$ .

#### **Lenguaje Aceptado**

- El lenguaje aceptado es el conjunto de todas las palabras aceptadas, se acepta una palabra cuando para la configuración inicial asociada a la palabra, existe una sucesión de movimientos posibles que permiten llegar a un estado de aceptación y parar. No importa que haya otras computaciones posibles que no lleguen a un estado de aceptación.

#### **Equivalencia entre MTND y MT**

- Toda MT es una MTND por lo que todo lenguaje aceptado por una MT determinista es aceptado por una MTND, también se da el inverso.

#### **Teorema**

- Si un lenguaje  $L$  es aceptado por una MTND, entonces es recursivamente enumerable.

#### **Demostración**

- Se supone que  $m$  es el número máximo de opciones en la MTND, se utiliza una MT determinista de dos cintas.
  - En la primera cinta se tendrán una sucesión de configuraciones  $(q, u, v)$  separadas por un símbolo  $\square$ , también existe una marca \* en la configuración activa.
  - Inicialmente hay solamente una configuración, la inicial.
- En cada momento, se coge la configuración activa, se copia en la cinta auxiliar; se va al final de la cinta original y se realizan todas las transiciones posibles sobre la configuración de la cinta auxiliar colocándolas al final de la primera cinta.
- Se busca la configuración marcada y se pasa a procesar la siguiente.
  - Si aparece en algún momento un estado de aceptación, se acepta y termina.

- Este procedimiento hace una búsqueda en anchura exhaustiva de todos los posibles cálculos de la MTND; si en uno acepta será encontrado, si en un nivel todos los cálculos terminan sin aceptar, se rechaza.

#### **Complejidad en Tiempo**

- En una MTND, el número de pasos o tiempo para una entrada  $u$  es el número de pasos para el cálculo más largo posible para esa entrada, si hay una secuencia de cálculos que no termina, entonces el tiempo es infinito.
- Si se dice que una MTND tiene complejidad  $t(n)$  en tiempo, quiere decir que todos los posibles cálculos de la MTND terminan en  $t(n)$  o menos pasos donde  $n$  es la longitud de la entrada; si una MT simula a una MTND, tendrá una complejidad  $O(d^{t(n)})$ , con  $d > 1$ .

#### **Ventajas de las MTND**

- Son útiles para resolver problemas donde se tiene que buscar un elemento en una población que cumpla una condición.
- La MTND elige de forma no-determinista un elemento de la población y comprueba si cumple la condición.

- Si cumple, acepta y sino, rechaza.
- Son problemas que llevan un proceso de búsqueda asociado, la MTND no tiene por qué describir el procedimiento de búsqueda.
- La simulación de una MTND con una MT determinista añade este tipo de búsqueda de anchura válida.

### Máquinas de Turing con Cintas Semilimitadas:

- Se puede suponer que la cinta de una MT está limitada por la izquierda y es ilimitada por la derecha, donde la palabra se escribe en la primera casilla de la cinta.
- Se puede demostrar que cualquier lenguaje se puede aceptar sin escribir a la izquierda de la primera casilla que ocupa la palabra de entrada, lo que equivale a suponer que no existe cinta a la izquierda de la palabra de entrada.

#### **Teorema**

- Todo lenguaje aceptado por una MT  $M_2$  es también aceptado por una MT  $M_1$  con las siguientes restricciones:
  - $M_1$  nunca escribe el espacio en blanco #.
  - El cabezal de  $M_1$  nunca se mueve hacia la izquierda de su posición inicial.
- La primera condición se solventa añadiendo un nuevo símbolo de espacio en blanco para  $M_1$ , por ejemplo, #'.
- Si  $M_2$  escribe un espacio en blanco,  $\delta_2(q, a) = (p, \#, M)$  entonces  $M_1$  escribe  $\delta_1(q, a) = (p, \#', M)$ .
  - Cada transición  $\delta_1(q, \#')$  se hace idéntica a  $\delta_2(q, \#)$ .
- La segunda condición se logra, una cinta ilimitada se puede simular como una semilimitada haciendo uso de dos pistas, tal que:

$X_0$	$X_1$	$X_2$							
$\triangleright$	$X_{-1}$	$X_{-2}$							

#### **Funcionamiento**

- El estado de  $M_2 = (Q_2, A_2, B_2, \delta_2, q_0, \#, F_2)$  se traduce a  $M_1 = (Q_1, A_2 \times \{\#\}, B_1, \delta_1, q_0, [\#, \#], F_1)$ .
  - $Q_1$  se compone de  $\{q_0, q_1\} \cup (Q_2 \times \{S, I\})$ ; los estados  $q_0, q_1$  preparan la cinta de entrada, para poner el símbolo  $\triangleright$ .
  - En los otros estados se tiene que especificar la pista,  $S$ , superior o  $I$ , inferior.
  - Los símbolos de trabajo de  $M_1$  son  $B_2 \times B_2$ , todas las parejas de símbolos de  $M_2$ , cada símbolo  $a \in A_2$  se identifica como  $[a, \#]$  en  $M_1$ .
  - En  $B_1$  están todas las parejas  $[b, \triangleright]$  donde  $b \in B_2$ , el símbolo  $\triangleright$  sirve para saber que se está en el extremo izquierdo de la cinta.
- Los primeros estados serían siempre:
  - $\delta_1(q_0, [a, \#]) = (q_1, [a, \triangleright], D)$  para cualquier  $a \in B_2$ .
  - $\delta_1(q_1, [a, \#]) = ([q_2, S], [a, \#, I], I)$ , se mueve hacia la izquierda y se indica que se está en la pista de arriba.

- Si  $\delta_2(q, a) = (p, b, M)$ , entonces para todo  $c \in B_2$

①  $\delta_1([q, S], [a, c]) = ([p, S], [b, c], M)$

②  $\delta_1([q, I], [c, a]) = ([p, I], [c, b], \overline{M})$ , donde  $\overline{M}$  es el movimiento de sentido opuesto a  $M$ .

- O sea, aquí se manejan las transiciones entre cualquiera de los estados; si se lee el símbolo en la

- pista superior, se escribe en la superior.
- En la inferior igual, con el movimiento invertido del estado original.

- Si  $\delta_2(q, a) = (p, b, D)$  entonces  
 $\delta_1([q, I], [a, \rhd]) = \delta_1([q, S], [a, \rhd]) = ([p, S], [b, \rhd], D)$

Si estamos al principio de la cinta, siempre se supone que el símbolo activo es el superior. Allí se escribe y si nos movemos a la derecha el superior es el activo.

- Si  $\delta_2(q, a) = (p, b, I)$  entonces  
 $\delta_1([q, I], [a, \rhd]) = \delta_1([q, S], [a, \rhd]) = ([p, I], [b, \rhd], D)$

Si estamos al principio de la cinta, siempre se supone que el símbolo activo es el superior. Allí se escribe y si nos movemos a la izquierda el inferior es el activo.

Los estados de aceptación  $F_1$  de  $M_1$  es el conjunto de estados  $F_2 \times \{S, I\}$ .

- Cuando se utilicen cinta semilimitadas, se supondrá desde el principio que se tiene el símbolo  $\rhd$  a la izquierda de cada una de las cintas que se usen y que si se llega a ese símbolo, se va inmediatamente a la derecha del mismo en el próximo movimiento.
- No supone una limitación, ya que toda MT se puede simular con esta restricción, siguiendo el algoritmo descrito.

### Problemas:

- Un  $PROBLEMA(x)$  o  $PROBLEMA()$  consta de:
  - Un conjunto  $X$  de entradas, un elemento  $x \in X$  se llama una entrada.
  - Un conjunto  $Y$  de soluciones, con  $y \in Y$  siendo una solución.
  - Una aplicación  $R: X \times Y \rightarrow \{0,1\}$  que representa la relación que debe existir entre las entradas y las soluciones, si  $R(X, Y) = 1$  es una solución, se puede escribir como  $R(X, Y)$ , y  $R(X, Y) = 0$  como  $\neg R(X, Y)$ .

### Ejemplo

Búsqueda de caminos en grafos dirigidos:

- Entradas  $X$ : conjunto formado por las tripletas  $(G, ns, nl)$ , donde  $G$  es un grafo dirigido,  $ns$  es un nodo de salida,  $nl$  es un nodo de llegada.
- Conjunto  $Y$ : lista de nodos  $(n_1, \dots, n_k)$
- La relación que debe de haber entre las entradas y las soluciones es que  $n_1 = ns, n_k = nl$  y todas las parejas  $(n_i, n_{i+1})$  sean arcos de  $G$ .

### Resolución de Problemas

- Los problemas se resuelven mediante algoritmos.
- Un algoritmo se considerará...
  - Un programa bien escrito sintácticamente en algún lenguaje de programación con al menos una función definida.
  - Una Máquina de Turing.
- Un algoritmo  $ALG$  resuelve un  $PROBLEMA(x)$  cuando el argumento de dicho algoritmo  $x \in X$  y  $ALG(x)$  es un  $y \in Y$  tal que  $R(x, y) = 1$ .

### Codificación

- Para resolver problemas deben de codificarse las entradas y las salidas a lo que puede soportar un

NO  
QUEMES  
TUS  
APUNTES

GANAS  
0,25 €

por subir tus  
apuntes en PDF  
a Wuolah

↓  
si juegas  
con fuego  
te fuegos

ordenador.

- Se supondrá que siempre estarán codificadas como palabras o cadenas de caracteres.
- Esto no supone ninguna restricción ya que cualquier tipo de entrada se puede representar mediante una cadena de caracteres, así como cualquier salida, también se supone que solo hay una palabra de entrada y una de salida.
- En términos generales, se hablarán de problemas donde las entradas y salidas son palabras sobre un alfabeto, esto es un problema computacional.
- Cualquier problema se puede transformar en un problema computacional con un sistema de entradas y salidas.
  - Pueden haber palabras que no sean una codificación correcta de una entrada del problema original, tendrán como salida siempre *NO*.

#### Tipos de Problemas

- **Problemas de Búsqueda:** Para una entrada  $x$ , el problema consiste en encontrar una solución  $y$  tal que  $R(x, y) = 1$  si existe  $y$  y *NO* en caso contrario.
- **Problemas de Decisión:** Aquellos con las soluciones  $Y = \{SI, NO\}$  y cada entrada  $x$  tiene una única solución.
- **Problema de Optimización:** La solución optimiza (minimiza o maximiza) una función definida sobre un conjunto de soluciones factibles asociadas a la entrada.
- **Problemas de Función:** Cada entrada  $x$  tiene siempre una y solo una solución.

#### Problemas de Decisión

- Son especialmente importantes ya que son simples y es fácil razonar sobre ellos, además que cualquier problema tiene asociado un problema de decisión.
  - **Problema de Umbral para Problemas de Optimización:** Con los datos originales, más un umbral  $K$  se pregunta si existe solución que sea mayor o menor que  $K$ , dependiendo si es maximizar o minimizar.
  - **Problemas de Existencia para Problemas de Búsqueda:** Dado un  $x$ , determinar si existe un  $y$  tal que sea solución de  $x$ .
  - **Problemas de Comprobación para Problemas de Función y Búsqueda:** Dado  $x$  y una posible solución  $y$  determinar si  $y$  es una solución de  $x$ .

#### Problemas de Decisión y Lenguajes

- Un problema computacional de decisión en el que las entradas se codifican como palabras de un alfabeto  $A$  con el lenguaje de las entradas que tienen respuesta *SI*.
$$L(\text{PROBLEMA}(x)) = \{x \in A^*: \text{PROBLEMA}(x) = SI\}$$
  - Es decir, los casos con respuesta afirmativa.
  - Un lenguaje  $L$  sobre un alfabeto  $A$  siempre define también un problema de decisión: dada  $x \in A^*$  determinar si  $x \in L$ .
- Se puede resumir como que, un problema de decisión con una codificación es un problema computacional *SI / NO*, y a su vez, esto es un lenguaje.
- Las definiciones sobre lenguajes se puede transformar en definiciones sobre los problemas asociados.
  - Los problemas cuyos lenguajes son recursivos se llaman decidibles o calculables.
    - Los que no, son indecidibles o no calculables.
  - Los problemas cuyos lenguajes son recursivamente enumerables se llaman semidecidibles o parcialmente calculables.
    - También existen los no semidecidibles.
- Esta terminología también se aplica a los lenguajes mismos.
- Identificar que las codificaciones sean correctas no se considera importante desde el punto de vista computacional porque se pueden reconocer en tiempo lineal.

#### Problema Contrario

- Dado un problema de decisión *PROBLEMA*, el problema contrario de *PROBLEMA* es el *C - PROBLEMA*, que intercambia las salidas *SI* y *NO*.

## Problemas contrarios y lenguajes

El lenguaje asociado al problema contrario de PROBLEMA es el lenguaje complementario del lenguaje del lenguaje asociado a PROBLEMA:

$$L(C - \text{PROBLEMA}) = \overline{L(\text{PROBLEMA})}$$

### Palabras y Números

- Suponiendo un alfabeto  $A = \{a_1, \dots, a_n\}$ , se puede establecer una correspondencia biyectiva entre palabras sobre este alfabeto y los números naturales.
- Suponiendo que  $w = a_{i_k}, \dots, a_{i_1}$  entonces el número  $w$  que se denotará como  $Z(w)$  es  $\sum_{j=1}^k i_j n^{j-1}$  siendo  $Z(\epsilon) = 0$ .
- Suponiendo nuevamente  $A$ , si  $m$  es un número natural, siempre se puede encontrar una cadena que se denotará como  $C(m)$  o  $w_m$  cuya codificación sea  $m$ .
  - Se puede conseguir al realizar divisiones y almacenando el resto.
- Si se tienen dos alfabetos  $A$  y  $B$ , se puede establecer una aplicación biyectiva entre las palabras de ambos.
  - Sea  $Z_A$  la aplicación que codifica las palabras de  $A$  como números naturales.  $Z_A: A^* \rightarrow \mathbb{N}$ .
  - Sea  $C_B$  la aplicación que transforma números naturales en palabras sobre  $B$ ,  $C_B: \mathbb{N} \rightarrow B^*$ .
  - La composición de  $C_B$  o  $Z_A$  es una aplicación biyectiva de  $A^*$  en  $B^*$ , se calcula el código numérico de una palabra de  $B$  y luego se obtiene la palabra de  $A$  que corresponde a ese código.
  - Se puede calcular esto por una MT.
- El orden total de las palabras de un alfabeto se considerará de la siguiente forma:
  - Si  $A = \{a_1, \dots, a_n\}$ , se considerará que  $u_1 \leq u_2$  si se da que...
    - $u_1 = u_2$
    - $|u_1| < |u_2|$
    - $|u_1| < |u_2|$  y  $u_1$  precede a  $u_2$  en orden alfabetico, donde  $a_1 < a_2 < \dots < a_n$ . Si  $u_1$  tiene la misma secuencia que  $u_2$  y el primer simbolo donde son distintas es menor.

### Codificando Máquinas de Turing

- Se puede asignar cada MT sobre un alfabeto  $\{0,1\}$  una cadena y un número natural, para ello...
  - Los estados son  $\{q_1, \dots, q_k\}$ , el estado inicial es  $q_1$  y hay un único estado final  $q_2$ , esto es siempre posible.
  - Los símbolos de  $B$  son  $\{a_1, a_2, \dots, a_m\}$  donde  $a_1$  es 0 y  $a_2$  es 1 y  $a_3$  es el símbolo blanco.
  - Al movimiento a la izquierda se le asigna un 1 y a la derecha un 2, este número será  $u(M)$ .
- La codificación de una MT se realiza tal que...
  - Cada transición  $\delta(q_i, a_j) = (q_k, a_l, M)$  se transforma a  $0^i 10^j 10^k 10^l 10^{u(M)}$ .
  - Todas las transiciones se añaden a la codificación separadas por un 11.
- Una vez calculada la cadena  $w = R(M)$  se puede calcular su número,  $Z(w)$  con el alfabeto  $\{0,1\}$ , este número se denotará como  $Z(M)$ .
- Cada número natural se corresponderá a una MT o corresponderá a una cadena sin sentido alguno.
  - Sea  $T(n)$  la MT correspondiente al número  $n$  o "nula" si no hay MT asociada al número  $n$ .
  - Sea  $T(w)$  la MT cuyo código es  $w$ .
- Para codificar una MT  $M$  y una palabra  $w$  sobre el alfabeto  $\{0,1\}$  se puede codificar  $M$  como se ha visto y después añadir 111 seguido de  $w$  para dar a lugar la cadena  $\langle M, w \rangle$ .

### Lenguaje de Diagonalización

- Se define el lenguaje  $L_d$  sobre  $\{0,1\}$  que no es recursivamente enumerable.
- Sea  $w \in \{0,1\}^*$  con  $w \in L_d$  si y solo si la MT cuya codificación es  $w$  no acepta  $w$ .
  - Se interpreta como que si  $w$  no es una MT correcta, entonces representa una MT con un solo estado sin transiciones, siempre rechaza.

- **Demostración**

- Suponer que existe una MT  $M$  que acepta  $L_d$ , esa MT estará definida sobre  $\{0,1\}$  y acepta palabras  $w$  tales que la MT cuya codificación es  $w$  acepta  $w$ , sea  $w_M$  la codificación de la MT  $M$ .
  - Si  $w_M \in L_d$  entonces  $M$  acepta  $w_M$  y la MT cuya codificación es  $w_M$  acepta la palabra  $w_M$ , por lo tanto,  $w_M \notin L_d$ .
  - Si  $w_M \notin L_d$  entonces  $M$  no acepta  $w_M$  y la MT cuya codificación es  $w_M$  no acepta la palabra  $w_M$  y por definición,  $w_M \in L_d$ .
- Con la existencia de  $M$  que acepte  $L_d$  se llega a una contradicción.

- **Problema**

- La versión del problema de decisión se llama el Problema de Diagonalización y se denota como  $DIAGONAL(M)$ , consiste en determinar si dada una MT  $M$ , entonces  $M$  no acepta cuando se lee así misma.

### Complementarios de Lenguajes Recursivos y Recursivamente Enumerables:

- Si  $L \subseteq A^*$ , el complementario  $L$  con respecto a  $A^*$  se denotará como  $\bar{L}$ .

- **Teorema:**

- Si  $L$  es un lenguaje recursivo, entonces  $\bar{L}$  también lo es.

#### Demostración

Si  $L$  es recursivo, entonces es aceptado por una MT que siempre para. Se construye  $\bar{M}$  con las siguientes características:

- Los estados de aceptación de  $M$  se convierten en estados de rechazo de  $\bar{M}$  donde no se realizan nuevas transiciones.
- $\bar{M}$  tiene un nuevo estado  $r$  que es de aceptación y que no tiene transiciones definidas.
- Para cada estado  $p$  de  $M$  que no sea de aceptación y para cada símbolo de la cinta  $a \in B$  para el que no haya definida una transición, se añade  $\delta(p, a) = (r, a, D)$ .

Está claro que  $\bar{M}$  acepta el lenguaje  $\bar{L}$  y siempre termina, ya que  $M$  lo hace.

- **Teorema:**

- Si  $L$  y  $\bar{L}$  son recursivamente enumerables, entonces  $L$  es recursivo.

#### Demostración

Sea  $M_1$  la MT que acepta  $L$  y  $M_2$  la MT que acepte  $\bar{L}$ . Vamos a construir una MT  $M$  con dos cintas: en una funciona como  $M_1$  y en la otra como  $M_2$ .

$M$  es una máquina que funciona como  $M_1$  y  $M_2$  a la vez (como en el autómata producto). El conjunto de estados de  $M$  es el producto  $Q_1 \times Q_2$  donde  $Q_1$  es el conjunto de estados de  $M_1$  y  $Q_2$  el de  $M_2$ . En cada paso, se pasa al estado que corresponde según  $M_1$  y al estado que corresponde según  $M_2$ .

La MT termina cuando una de las dos máquinas termina.

Los estados de aceptación de  $M$  son las parejas, en las que el primer elemento es un estado de aceptación de  $M_1$ .

Está claro que  $M$  acepta  $L$  y siempre termina, ya que toda palabra  $u \in A^*$  está en  $L$  o en  $\bar{L}$ . En el primer caso,  $M_1$  termina y en el segundo lo hace  $M_2$ . Por lo tanto  $M$  siempre termina.

El lenguaje  $L_d$  no es r.e. Su complementario podría ser r.e., pero nunca recursivo. De hecho, es r.e.

### Lenguaje Universal:

- El lenguaje universal  $L_u$  es el conjunto  $v$  de todas las cadenas  $\{0,1\}$  que codifican parejas  $\langle M, w \rangle$

- donde  $M$  tiene como alfabeto  $\{0,1\}$  y  $w$  está codificado en ese alfabeto y tales que  $M$  acepta  $w$ .
- El problema universal se denota como  $UNIVERSAL(M, w)$ , consiste en determinar si dada una MT  $M$  y entrada  $w$ , entonces  $M$  acepta  $w$ .
- **Teorema:**
  - El lenguaje  $L_u$  es recursivamente enumerable.
- **Demostración:**
  - La idea es construir una MT  $M_u$  que lea  $\langle M, w \rangle$ , y simule  $M$  con entrada  $w$ : es claro que si  $M$  acepta, también  $M_u$  lo hará.
  - $M_u$  tiene varias cintas, en la primera la codificación de  $M$  y  $w$ , en la segunda la cinta  $M$  con la entrada  $w$ , la tercera contiene el estado de  $M$  y la cuarta cinta para cálculos auxiliares.
  - $M_u$  primero analiza  $M$  y  $w$  para determinar que sea una entrada correcta, luego se inicializa la segunda cinta con  $w$ , la tercera con el estado inicial. Hecho esto empieza a simular los movimientos de  $M$ , tendrá que localizar en la primera cinta el estado según lo que lee en la segunda cinta, si no lo encuentra se para, si lo encuentra...
    - Cambia el estado de la tercera cinta
    - Modifica la segunda cinta como indique el estado y mover el cabezal
  - Si  $M$  pasa a un estado de aceptación,  $M_u$  también acepta.
- **Teorema:**
  - El lenguaje  $L_u$  no es recursivo.
- **Demostración:**
  - Si  $L_u$  fuera recursivo, también  $\overline{L_u}$  sería.
  - Si  $M$  es una MT que aceptara  $\overline{L_u}$ , se podría construir  $M'$  que podría aceptar  $L_d$  lo cual no es posible, pues  $L_d$  no es recursivamente enumerable.

### Reducción:

#### **En términos de lenguajes**

- Si  $L_1 \subseteq A^*$  y  $L_2 \subseteq B^*$  son lenguajes, el lenguaje  $L_1$  se reduce al lenguaje  $L_2$  si existe una MT que siempre para y calcula una función  $f: A^* \rightarrow B^*$  tal que para toda entrada  $w \in A^*$ ,  $w \in L_1 \Leftrightarrow f(w) \in L_2$
- **Teorema:**
  - Si  $L_1$  se reduce a  $L_2$  entonces si  $L_1$  no es recursivo, tampoco lo es  $L_2$  y si  $L_1$  no es r.e., tampoco  $L_2$  lo es.

#### **En términos de problemas**

- Sean  $PROBLEMA_1$  y  $PROBLEMA_2$  problemas de decisión, entonces se dice que  $PROBLEMA_1$  se reduce a  $PROBLEMA_2$  si existe un algoritmo  $ALG(w)$  que siempre para y calcula una función  $f(w)$  tal que para toda entrada  $w$  a  $PROBLEMA_1$ , se tiene que  $PROBLEMA_2$  produce la misma respuesta para la entrada  $f(w) = ALG(w)$ .
- Una reducción de  $PROBLEMA_1$  a  $PROBLEMA_2$  es un algoritmo  $ALG(w)$  que transforma las entradas de un problema a las del otro, o sea,  $PROBLEMA_1(x) = PROBLEMA_2(ALG(x))$ .
- **Teorema:**
  - Si  $PROBLEMA_1$  se reduce a  $PROBLEMA_2$  entonces si  $PROBLEMA_1$  es indecidible, también lo es  $PROBLEMA_2$  y si  $PROBLEMA_1$  no es semidecidible, tampoco lo es  $PROBLEMA_2$ .

### Máquinas de Turing que aceptan el Lenguaje vacío:

- Se definen los siguientes lenguajes sobre el alfabeto  $\{0,1\}$ :
  - $L_e$  es el conjunto de palabras  $w$  tales que la MT  $M$  sobre  $\{0,1\}$  cuya codificación es  $w$  no acepta ninguna palabra,  $L(M) = \emptyset$ .
    - $VACIO(x)$  es la versión de problema: Dada una MT  $M$  determinar si acepta el lenguaje vacío.
  - $L_{ne}$  es el conjunto de palabras  $w$  tales que la MT  $M$  sobre  $\{0,1\}$  cuya codificación es  $w$  acepta alguna palabra,  $L(M) \neq \emptyset$ .
    - $C - VACIO(x)$  es la versión de problema: Dada una MT  $M$  determinar si acepta algo

NO  
QUEMES  
TUS  
APUNTES

**GANAS  
0,25 €**

por subir tus  
apuntes en PDF  
a Wuolah

↓  
si juegas  
con fuego  
te fuegos

distinto que vacío.

- Teorema:
  - $L_{ne}$  es recursivamente enumerable.
- Demostración:
  - Se tiene una MTND,  $M$  lee  $w$  que codifica una MT  $N$  y con el no determinismo se genera una palabra de entrada  $u$ , se pone a simular  $\langle N, u \rangle$  en  $M_u$ , la MT Universal, si  $M_u$  para y acepta entonces  $M$  acepta  $w$ .
- Teorema:
  - $L_{ne}$  no es recursivo
- Demostración:
  - Se puede reducir  $UNIVERSAL(M, w)$  a  $C - VACIO(M')$ , por lo tanto,  $UNIVERSAL$  que no es recursivo implica que  $C - VACIO$  tampoco lo será.

#### Demostración

Vamos a demostrarlo usando los problemas asociados. Vamos a reducir  $UNIVERSAL(M, w)$  and  $C - VACIO(M')$ . Eso consiste en un algoritmo  $ALG(M, w)$  que calcula una MT  $M'$  de tal manera que  $UNIVERSAL(M, w)$  tenga la misma solución que  $C - VACIO(M')$ . Este funciona de la siguiente forma:

- Supongamos una entrada  $(M, w)$  vamos a construir una MT  $M'$  que funciona de la siguiente forma.
- $M'$  ignora su entrada  $x$  y coloca en la cinta de entrada  $w$ . Si la longitud de  $w$  es  $n$ , esto se puede hacer con  $n$  estados. Cada estado  $q_i$  escribe el símbolo  $i$  de  $(M, w)$  y se mueve a la derecha. Despues pasaría a un nuevo estado en el que borra lo que quede de  $x$ .
- $M'$  se mueve a la izquierda hasta el primer símbolo de  $w$ .
- $M'$  pasa al estado inicial de  $M$  con  $w$  y funciona como  $M$  para  $w$ .
- La salida de  $M'$  es la misma que la de  $M$  para  $w$ .

Está claro que  $M$  acepta  $w$  si y solo si  $M'$  acepta alguna palabra. De hecho  $L(M') = A^*$  si  $M$  acepta  $w$  y  $L(M') = \emptyset$  si  $M$  no acepta  $w$ .

#### Propiedades de los Lenguajes Recursivamente Enumerados:

- Una propiedad de los lenguajes recursivamente enumerados es una propiedad de los lenguajes de las MTs.
- Una propiedad de los lenguajes r.e. se dice trivial si para toda MT su lenguaje aceptado no verifica la propiedad o para toda MT, su lenguaje aceptado siempre verifica la propiedad.
- Hay que distinguir entre propiedades de una MT o una propiedad de los lenguajes de una MT.

#### **Teorema de Rice**

- Toda propiedad no trivial sobre los lenguajes r.e. es indecidible.

## Demostración

Llamemos NOTRIVIAL( $M$ ) a dicha propiedad no trivial y supongamos una propiedad no trivial y supongamos que el lenguaje vacío aceptado por la MT  $M_e$  no verifica la propiedad que otro lenguaje  $L$  aceptado por la MT  $M_L$  si verifica la propiedad.

Vamos a reducir el problema UNIVERSAL( $M, w$ ) a esta propiedad. Supongamos  $(M, w)$  una MT y su entrada construimos  $M'$  de la siguiente forma (se supone que tiene una entrada  $x$  en la primera cinta):

- $M'$  tiene dos cintas. En la segunda coloca a  $w$  y empieza trabajando sobre esta cinta con las mismas transiciones de  $M$ . Si termina y no acepta, entonces  $M'$  no acepta.
- Si  $M$  termina y acepta con  $w$ , entonces empieza a funcionar como  $M_L$  sobre la entrada  $x$  en la primera cinta y tiene la misma salida que  $M_L$ .

Está claro que si  $M$  acepta  $w$ , entonces el lenguaje de  $M'$  es  $L$  y verifica la propiedad. Si  $M$  no acepta  $w$ , entonces el lenguaje de  $M'$  es vacío y no verifica la propiedad. Con esto se acaba la reducción y la propiedad no es decidible.

Si el lenguaje vacío acepta la propiedad se haría la misma transformación con la propiedad complementaria, demostrando que no es decidible y, por lo tanto, la propiedad original tampoco lo es.

## Otros problemas indecidibles:

### Problema de las Correspondencias de POST / POST( $A_1, A_2$ )

- Se tiene un alfabeto de referencia  $A$  y dos listas con la misma longitud,  $B_1 = w_1, \dots, w_k$  y  $B_2 = u_1, \dots, u_k$  de palabras sobre  $A$ , el problema es determinar si existe una secuencia no vacía de enteros  $i_1, \dots, i_m$  tales que  $w_{i1}, \dots, w_{im} = u_1 \dots u_{im}$ .

### PCP Modificado

- Misma proposición pero se indica ahora el bloque por el cual se comienza y que las palabras no pueden ser vacías
- Este problema es indecidible si  $A$  tiene al menos dos símbolos.

### Gramática Ambigua

- **Teorema:** Saber si una gramática independiente del contexto es ambigua es indecidible.

Se reduce el problema de las correspondencias de Post. Supongamos el alfabeto  $A$  y los bloques  $\begin{array}{|c|}\hline u_1 \\ \hline v_1 \\ \hline \end{array}, \dots, \begin{array}{|c|}\hline u_k \\ \hline v_k \\ \hline \end{array}$ .

Sea  $B = A \cup \{b_1, \dots, b_k\}$  donde  $b_i \notin A$  y construimos la siguiente gramática:

$$S \rightarrow C|D$$

$$C \rightarrow u_i C b_i | u_i b_i, \quad i = 1, \dots, k$$

$$D \rightarrow v_i D b_i | v_i b_i, \quad i = 1, \dots, k$$

La solución al problema de las correspondencias de Post es equivalente a que la gramática sea ambigua.

## Tema-2-Otros-Modelos-de-Cálculo-...



ParmigianoReg



Modelos Avanzados de Computación (Especialidad  
Computación y Sistemas Inteligentes)



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada

**QUIERES  
15€?**



**WUOLAH**



QUIERES  
CONSEGUIR  
**15€??**

TRÁENOS A TU  
CRUSH DE APUNTES  
ANTES DE QUE  
LOS QUEME



## Tema 2 - Otros Modelos de Cálculo: Tesis de Church-Turing

Profesor Serafín Moral Callejón, Curso 21-22

### Programas Post-Turing:

- Se trata de un lenguaje de programación para manipulación de palabras, el sistema Post-Turing. En este modelo se pueden escribir programas que actúan sobre una cinta en la que se pueden escribir o leer símbolos, es ilimitada en ambas direcciones.
- En los cálculos asociados siempre hay una casilla de la cinta que está activa en todo momento, este símbolo y solo este de la cinta se supone observado y en esta casilla solo se puede escribir un símbolo.
  - Es funcionalmente igual que una MT pero la estructura para especificar el control es más parecida a un programa en un lenguaje de programación convencional: una lista de instrucciones.

#### Estructura de los Programas

- Es un conjunto de instrucciones sobre un alfabeto de entrada  $A$  y un alfabeto de trabajo  $B$  que incluye el símbolo en blanco.
- Cada instrucción puede tener una etiqueta opcional, es una palabra de un alfabeto determinado, por ejemplo,  $[L]$ .
- Las instrucciones son:
  - $PRINT\ a$ , imprime el símbolo  $a$  en la casilla de la cinta donde está posicionado el cabezal.
  - $IF\ a\ GOTO\ L$ , si el símbolo que se ve es  $a$  ir a la instrucción etiquetada con  $[L]$ .
  - $RIGHT$ , mueve el cabezal a la derecha.
  - $LEFT$ , mueve el cabezal a la izquierda.
  - $HALT$ , terminar y aceptar.
- El programa comienza con una entrada  $u$  en una cinta ilimitada rodada por símbolos en blanco y el cabezal de lectura a la izquierda de la palabra.
- Se ejecuta instrucción por instrucción de manera descendente excepto si se produce un salto con  $IF\ a\ GOTO\ L$
- Se termina la ejecución cuando no hay más instrucciones o se llega a un  $HALT$ .

#### Lenguaje Aceptado

- El lenguaje que acepta un programa Post-Turing es el conjunto de palabras del alfabeto de entrada que comenzando en la configuración inicial llegan a la instrucción  $HALT$ .

#### Función Calculada

- La función que calcula es  $f: D \rightarrow B^*$  definida en el conjunto  $D$  de las entradas  $u \in A^*$  tal que el programa Post Turing llega a  $HALT$  siendo  $f(u)$  la palabra en la cinta excluyendo blancos  $u \in D$ .
- Una función calculada por un programa Post-Turing se dice **parcialmente calculable Post Turing**, si  $D = A^*$  se dice **calculable total Post Turing**.

#### Macros

- Es un conjunto de instrucciones que realizan una tarea concreta y a las que se les asigna un nombre
- No suponen una funcionalidad adicional del lenguaje Post-Turing, simplemente es una forma de resumir conjuntos de instrucciones para simplificar la escritura.

si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

WUOLAH

WUOLAH

## Ejemplo

La macro **RIGHT TO NEXT BLANK** se expande como,

[A]    **RIGHT**  
      **IF # GOTO E**  
      **GOTO A**

Una vez definida esta macro, ya podríamos usar **RIGHT TO NEXT BLANK** como un resumen de su expansión.

## Programas con Variables:

- Un programa con variables viene dado por un alfabeto  $A$  de entrada, un alfabeto  $B$  de trabajo y los siguientes elementos:
  - Una variable  $X$  de entrada, un conjunto finito de variables de trabajo  $Z_i$  y una variable  $Y$  de salida.
- Posee las siguientes instrucciones:
  - $A \leftarrow aA$ , añadir el símbolo  $a$  al principio de la variable  $A$ .
  - $A \leftarrow A -$ , eliminar el último símbolo de  $A$  si no es vacía.
  - $IF A ENDS a GOTO L$ , si el último símbolo de  $A$  es  $a$  ir a  $L$ .
  - $HALT$ , terminar y aceptar.
- Se supone que se empieza con  $X = u$ , donde  $u \in A^*$  es el valor de entrada y el resto de variables están con la palabra vacía,  $\epsilon$ .
- Se acepta una palabra al llegar a  $HALT$ .
- Calcula una función parcial  $f$  si llega a  $HALT$  con  $f(u)$  almacenando en  $Y$  definida en el conjunto de palabras para las que el programa termina.

### Macros

- Se pueden utilizar igualmente macros, conjuntos de instrucciones que se usan a menudo, se pueden resumir como una nueva instrucción, esto no cambia la definición del lenguaje.

### Entradas Múltiples

- En muchas ocasiones, se desea calcular una función  $f(u_1, \dots, u_n)$  que depende de varias variables, se puede hacer de la siguiente forma:
  - Transformar todo en una sola entrada y añadir un símbolo separador  $c$  que no se encuentre en el alfabeto y calcular  $f'(u_1 c \dots c u_n)$ , el cálculo de  $f$  es equivalente al de  $f'$ .
  - Suponer que se tienen  $n$  variables de entrada  $X_1, \dots, X_n$  y que cada una de ellas contiene al principio uno de los argumentos de entrada  $X_i = u_i$ .

## Equivalencia de Modelos:

### Teorema

- Los siguientes hechos son equivalentes...
  - $f$  es parcialmente calculable por una MT, un programa Post-Turing y un programa con variables.
  - $L$  es aceptado por una MT, un programa Post-Turing y un programa con variables.
- Esto se demuestra simulando los modelos entre sí.
  - **Un programa con variables simulado en un programa Post-Turing.**
    - Se pueden poner todas las variables del programa en una cinta del programa Post-Turing de tal forma que estén separadas por espacios en blanco tal que  $X_1, \dots, X_m, Z_1, \dots, Z_m, Y$  sea  $\#X_1\# \dots \#X_m\#Z_1\# \dots \#Z_m\#Y\#$ .
    - Se crean macros nuevos para automatizar la ubicación de la variable deseada además de añadir o eliminar símbolos de la misma.
      - **GOTO L**

- LEFT TO NEXT BLANK
- MOVE BLOCK RIGHT
- ERASE BLOCK
- **Un programa Post-Turing por una MT.**
  - La MT tendrá los mismos alfabetos de entrada y de trabajo, tendrá un estado  $q_i$  por cada instrucción  $I_i$  del programa más un estado  $q_f$  de estado final y  $q_{k+1}$  sin transiciones,  $k$  siendo el número de instrucciones del programa.
  - El estado inicial corresponde a la primera instrucción.
  - Existen correspondencia directa entre las instrucciones y los estados  $\delta$  de la MT
- **Una MT por un programa con variables.**
  - La MT y el programa tendrán el mismo alfabeto, el programa tendrá las 3 variables básicas  $X, Z, Y$  más las auxiliares.  $X$  tendrá inicialmente la palabra de entrada.
  - La idea es que en cada momento, la variable  $X$  contenga lo que hay en la izquierda del cabezal de lectura,  $Z$  lo que ve el cabezal de lectura e  $Y$  lo que esté a la derecha.
  - Se definen los macros:
    - $V \leftarrow -V$
    - $V \leftarrow V a_j$
    - $IF\ V\ STARTS\ a_i\ GOTO\ L$
  - Se asocia cada estado  $q_i$  a una etiqueta  $A_i$  y a cada par de estados  $(q_i, a_j)$  otra etiqueta  $B_{ij}$  para simular la transición  $\delta(q_i, a_j)$ , las transiciones no definidas se asocian a una etiqueta aparte
  - Dependiendo de la etiqueta se realizarán diferentes acciones...
    - $A_i$  mandan a etiquetas  $B_{ij}$  dependiendo del contenido de  $Z$ .
    - $B_{ij}$  quitan o ponen de  $X$  a  $Y$  dependiendo del movimiento y actualizan  $Z$ .
    - Los estados finales de  $A_i$  ponen todo el contenido de la cinta en  $Y$  antes de finalizar con  $HALT$ .

### Cálculo con Números:

- Se puede, además de realizar programas con palabras, aplicar números también, ya se ha visto que existe una aplicación biyectiva entre las palabras de cualquier alfabeto  $A^*$  y el conjunto de los números naturales  $\mathbb{N}$  que se llamaba  $Z$ , el número asociado a la palabra, siendo su inversa  $C$ , la palabra asociada al número.
- De esta manera cualquier modelo de cálculo con palabras se puede interpretar como un modelo de cálculo con números, una función numérica  $f(n)$  definida sobre los números naturales permite considerar que se calcula con un modelo de palabras donde la entrada se pone a  $u = C(n)$ , la palabra que codifica el número  $n$  y si la salida es  $w$ , se interpreta el número  $f(n) = Z(w)$ , el número que representa  $w$ .
- Se puede así definir el concepto de función parcialmente calculable de números, cuando exista una codificación en un alfabeto que la calcule.
  - Una función es recursiva o calculable cuando es parcialmente calculable y total, está definida dentro de los números naturales.
  - Se puede definir el concepto de conjunto numérico recursivo o recursivamente Enumerable, cuando el conjunto de todas sus codificaciones en un alfabeto sea recursivo o recursivamente enumerable.
- Es posible también diseñar modelos de cálculo que trabajen directamente con números y definir el concepto de función numérica.

### Programas con Variables Numéricas:

- Un programa con variables numéricas posee un conjunto  $X_1, \dots, X_k$  de variables de entrada, un conjunto finito de variables de trabajo  $Z_1, \dots, Z_j$  y una variable  $Y$  de salida.
- Posee un conjunto de instrucciones con etiquetas opcionales
  - $A \leftarrow A + 1$ , añade 1 al valor entero almacenado en  $A$ .
  - $A \leftarrow A - 1$ , resta 1 al valor entero en  $A$ , si es 0 se mantiene en 0.

QUIERES  
CONSEGUIR  
**15€??**

TRÁENOS A TU  
CRUSH DE APUNTES  
ANTES DE QUE  
LOS QUEME



- *IF A ≠ 0 GOTO L*, si el valor de A no es 0, ir a la etiqueta L.
- *HALT*, termina y acepta.
- Se supone que  $X_i = n_i$  donde  $n_i \in \mathbb{N}$  son los valores de entrada y el resto de variables contienen 0.
- Acepta una entrada si llega a *HALT* y calcula una función parcial  $f$  si llega a *HALT* con  $f(n_1, \dots, n_k)$  almacenado en  $Y$  cuando  $f$  está definida y no para en otro caso.
- Poseen macros, iguales a los de Programas con Variables.
- **Teorema:** Una función  $f: A \rightarrow \mathbb{N}$  es parcialmente calculable por un programa con números si y solo si es parcialmente calculable utilizando una codificación con palabras.
  - La demostración se basa en simular un programa con números en un programa con variables que trabaje con las codificaciones de estos números en un alfabeto.

#### Tesis de Church-Turing:

- Se han visto modelos de cálculo en donde todos poseen la misma capacidad para calcular funciones o aceptar lenguajes.
- La generalización de este hecho lleva a la Tesis de Church-Turing
  - *"Toda función efectivamente calculable, calculable mediante un proceso mecánico bien definido, puede ser calculada por una Máquina de Turing"*
- Es una tesis porque no puede ser comprobada matemáticamente si no se da una definición formal de "efectivamente calculable" pero eso es precisamente lo que intenta hacer la tesis; es aceptada como cierta aunque se consideren modelos de computación cuántica, estos no permitirían hacer cálculos más allá de los realizados por una Máquina de Turing, aunque capaz sí en menos tiempo.



si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

# Tema-3-Clases-de-Complejidad.pdf



ParmigianoReg



Modelos Avanzados de Computación (Especialidad  
Computación y Sistemas Inteligentes)



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada

**QUIERES  
15€?**



**WUOLAH**



**QUIERES  
CONSEGUIR  
15€??**

TRÁENOS A TU  
CRUSH DE APUNTES  
ANTES DE QUE  
LOS QUEME



## Tema 3 - Clases de Complejidad

Profesor Serafín Moral Callejón, Curso 21-22

### Objetivo básico:

- Clasificar los problemas en base a su dificultad, con el objetivo final de comprender de manera profunda la naturaleza de los problemas y los algoritmos que lo resuelven.

### Metodología:

- Abstraer los ejemplos, se obtiene la complejidad en función del tamaño de la entrada y en el peor de los casos
  - Abstraer los modelos, se abstrae la codificación de un lenguaje y se trabajan con Máquinas de Turing.
  - Los recursos que se miden son el tiempo y el espacio.
  - Los algoritmos, se elige el mejor posible.
- 
- La complejidad se mide en función de la longitud de la entrada, si se tiene un número  $x$ , ¿cuántos caracteres se necesitan para escribir  $x$ ?
    - La barra de medir un problema es el número de caracteres para la entrada del problema, y en función de eso se mide la complejidad.
    - Se mide el tiempo en función del tamaño de la entrada y lo que tarda eso en ejecutarse; y la forma de medir es cuantos caracteres se necesitan para representarlo. Es algo que se puede aplicar todo.
    - Ya sea en binario, decimal o con alguna otra codificación si el número  $x$  es el que se necesita, el número de caracteres será del orden de  $n = \log(x)$ .
  - La complejidad de un problema no depende del número en sí, sino del número de caracteres necesario para escribirlo
    - Esta medida puede aplicarse a cualquier problema ya sea numérico, de grafos, de conjuntos, o de cualquier tipo.
    - Ejemplo: Si  $n$  es la longitud del número en binario, su valor  $x$  es del orden de  $2^n$ , si fuera decimal sería  $10^n$ .

### Primalidad:

- El algoritmo de primalidad, que es aquel para comprobar la primalidad de un número natural  $x$  va dividiendo ese número por todos los números y menores que  $x$  para ver si existe una división exacta, esto es exponencial.
  - El número de divisiones es  $\mathcal{O}(x)$  pero esa no es la medida correcta de complejidad.
  - Se tiene que tomar en cuenta que la representación en binario de un número  $x$  ocupa el orden de  $n = \log(x)$  caracteres.
  - Un número  $y$  menor que  $x$  ocupa también del orden de  $n$  casillas, una división de un número de longitud  $n$  por otro de longitud  $n$  se hace en orden  $\mathcal{O}(n^2)$ , como el número  $n = \log(x)$  entonces  $x$  es de orden  $2^n$  y el número de divisiones es de orden  $\mathcal{O}(2^n)$ , lo que da como total una complejidad exponencial,  $\mathcal{O}(n^2 2^n)$ .

### Ejemplos de Dos Problemas:

#### **Problema del Flujo Máximo**

- En este algoritmo, se tiene un grafo dirigido en el que los arcos están etiquetados por su capacidad, se tiene un origen ( $s$ ) y un destino ( $t$ ).
- Un flujo es la asignación de valor a cada arco que no supere su capacidad y de forma que la suma de



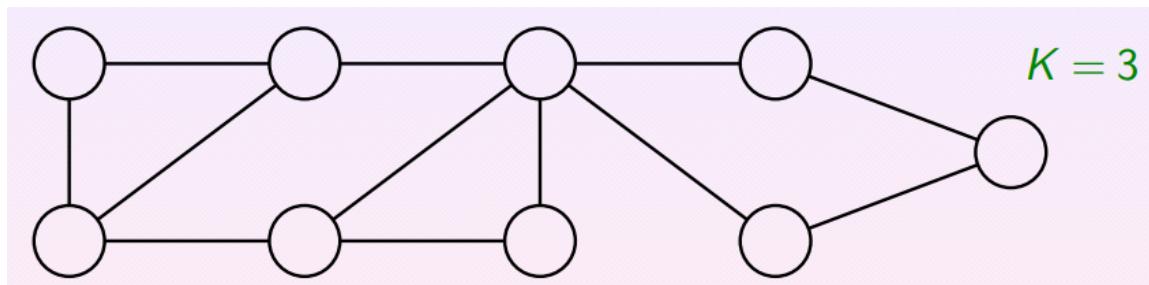
si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

lo que entra a cada nodo intermedio es igual a la suma de lo que sale, el valor del flujo es la suma de lo que sale del origen a lo que llega al destino y el problema es calcular el flujo máximo.

- Existe una forma de elegir el camino que garantiza el mejor crecimiento con complejidad polinómica, basta con elegir en cada caso el camino más corto entre origen y destino, se tiene que la complejidad máxima es del orden de  $\mathcal{O}(n^5)$ , con  $n$  siendo la longitud de la entrada.

### Problema de Coloreo de Grafos

- Dado un grafo  $(G, V)$  y un número entero  $K \dots$



- ... determinar si se puede colorar con  $K$  colores el grafo sin que se toquen dos colores iguales.

### La diferencia

- En el problema del flujo, se conoce un procedimiento que necesita hacer un cálculo polinómico y que se va acercando al óptimo de forma creciente.
- En el problema del grafo, no se conoce una forma de colorear el gráfico en tiempo polinómico y que evite la fuerza bruta: la búsqueda de espacio en todas las opciones.

### Clases de Complejidad:

- Una Máquina de Turing (u otro dispositivo de cálculo) es de complejidad  $f(n)$  si y solo si para toda entrada  $x \in A^*$  de longitud  $|x| = n$ , la máquina acepta esta entrada o la rechaza, consumiendo menos de  $f(n)$  unidades.
  - Unidades de Pasos de Cálculo: Complejidad en Tiempo
  - Unidades de Casillas en Cinta: Complejidad en Espacio.
- Un Lenguaje se dice que es de complejidad  $f(n)$  si existe una MT que acepta el lenguaje y tiene complejidad  $f(n)$ .

### Órdenes de Complejidad

- Se dice que una medida  $g(n)$  es de orden  $\mathcal{O}(f(n))$  si existe  $n_0$  y  $c > 0$  tal que  $\forall n \geq n_0, g(n) \leq cf(n)$ .
- **Teorema:** Si  $L$  es aceptado en tiempo  $f(n)$  por una MT con  $k$  cintas, entonces  $\forall m > 0$  existe una MT con  $k + 1$  cintas que acepta el mismo lenguaje en  $\frac{1}{2^m}t(n) + n$ .
- Demostración:
  - Se puede reducir el tiempo, en principio a la mitad, aunque se puede repetir tantas veces sea necesario, si  $M$  es la máquina original que acepta el alfabeto  $A$  se construye la máquina  $M'$  en la que hay un símbolo nuevo  $w$  por cada 3 símbolos  $abc$  de  $A$ .
  - La máquina es tal que codifica la cinta original en otra cinta de forma que la casilla  $i$  de la cinta representará las casillas  $2i - 1, 2i, 2i + 1$ .
  - La máquina  $M'$  simula lo que haría  $w$  para  $M$  para los símbolos de entrada, por ejemplo,  $abc$ , esto permite resumir dos pasos en uno, para llevar cuenta de los símbolos
  - Con esto, cada dos pasos se hacen en 1 y se dividen los pasos por la mitad.

### Codificando Problemas:

- **Codificando números:** Los números se pueden codificar en cualquier base menos unario, ya que es poco eficiente, en teoría se supondrá binario y en la práctica decimal.
- **Codificando objetos de un conjunto:** Existen diversas maneras de codificar problemas, por ejemplo, la codificación de grafos se puede realizar de distintas maneras.

### Complejidad de Problemas de Grafos

- En esta complejidad, interesa saber si es polinómica, da igual la representación que se utilice y si se mide en función de los vértices, será polinómica en cualquier representación aunque puede que tenga una constante diferente, será siempre polinómica.
- Ocurre lo mismo si se quiere saber si la complejidad es de orden logarítmico,  $O(\log(n))$ , es independiente de la representación o si se mide como una función del número de vértices, el exponente de un logaritmo se transforma en una constante.

### Medidas de Complejidad en Espacio

- En general, para medir el número de unidades que se consumen se siguen las siguientes reglas:
  - Se cuentan las casillas que se usan, las que se escriben o pasan sobre ellas.
  - Si nunca se escribe en la cinta de entrada, entonces las casillas de esta cinta no se cuentan.
  - Si las casillas de la cinta de salida se escriben de izquierda a derecha sin volver nunca hacia atrás, tampoco se cuentan.
- En un algoritmo que se cuenta el espacio, pueden existir cálculos que nunca acaben, pero siempre se puede transformar el algoritmo en uno que no cicle.

## Reconocer palíndromos en espacio $O(\log(n))$

Se hace con una Máquina de Turing con las siguiente estructura de cintas:

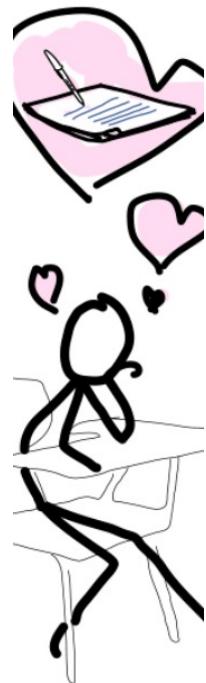
1	2	3	3	2	1	□	Entrada
1	0	1	□	□	Posición que se está comprobando (binario) $N_2$		
1	1	□	□	□	Contador en binario para encontrar posiciones $N_3$		

- Ponemos 1 en la segunda cinta ( $N_2$ ), Ponemos 1 en la tercera cinta ( $N_3$ )
- Nos ponemos al principio de la primera cinta.
- Repetir:
  - Repetir: Incrementar  $N_3$  en 1, mover a la derecha en la primera , hasta  $N_2 = N_3$
  - Copiar el símbolo de la primera cinta en memoria
    - Si el símbolo en memoria es blanco **Aceptar**
    - en otro caso
      - Poner 1 en la tercera cinta ( $N_3 = 1$ ), ir al final de la primera cinta
      - Repetir: Incrementar  $N_3$  en 1, mover izquierda en la primera, hasta  $N_2 = N_3$
      - Si símbolo en primera cinta es distinto al de memoria **Rechazar**
      - Incrementar  $N_2$  en 1

Los números  $N_2$  y  $N_3$  necesitan  $\log(n)$  casillas donde  $n$  es la longitud del número en la cinta 1.

**QUIERES  
CONSEGUIR  
15€??**

TRÁENOS A TU  
CRUSH DE APUNTES  
ANTES DE QUE  
LOS QUEME



G	v <sub>1</sub>	v <sub>2</sub>	Entrada		
u	z	v	Nodos por analizar		
x	y	u	z v	Nodos visitados	

Supongamos  $n$  el tamaño de la entrada en la primera cinta.

Las cintas 2 y 3 siempre son más cortas que la entrada, luego es de  $O(n)$  en espacio.

El número de pasos sobre las tres cintas es:

- La cinta 1 se recorre un número de veces menor o igual al número de vértices  $v$  que a su vez es menor o igual a  $n$  y como su tamaño es  $n$ , el número de pasos es de orden  $O(n^2)$ .
- En la cinta 2 cada nodo se recorre varias veces: para escribirlo, buscarlo en la primera cinta y borrarlo. Buscarlo en la entrada, a lo más una vez por arista. El número de aristas es menor o igual a  $n$ , y recorrer todos los nodos una vez es, a lo más  $n$ , luego la complejidad total es de orden  $O(n^2)$ .
- La cinta 3, hay que recorrerla cada vez que analizamos un nodo  $v$  y encontramos la arista  $(v, v')$ . Como cada nodo se analiza solo una vez, entonces a lo más se recorre una vez por arista. Como el número de aristas es  $O(n)$  y el tamaño de la cinta es  $O(n)$ , obtenemos  $O(n^2)$  en total.

En total, la complejidad en tiempo es  $O(n^2)$  y en espacio  $O(n)$

También se puede hacer en espacio  $O(v)$  y en tiempo  $O(v^3)$ , donde  $v$  es el número de vértices (por ejemplo codificando como matriz 0-1).

### Clases No-Deterministas:

- La función de complejidad se puede medir en una MTND, la misma tiene una complejidad  $f(n)$  en tiempo y espacio si y solo si para una entrada  $x$  de longitud  $n$  todas las posibles opciones de cálculo de la máquina terminan en  $f(n)$  pasos o terminan y no se usan más de  $f(n)$  casillas.

### Clases de Complejidad:

- TIEMPO(f): Todos los lenguajes aceptados por una MT determinista en tiempo  $O(f(n))$ .
- ESPACIO(f): Todos los lenguajes aceptados por una MT determinista en espacio  $O(f(n))$ .
- NTIEMPO(f): Todos los lenguajes aceptados por una MTND en tiempo  $O(f(n))$ .
- NESPACIO(f): Todos los lenguajes aceptados por una MTND en espacio  $O(f(n))$ .
- P, polinómica en tiempo, todos los lenguajes del tipo  $TIEMPO(n^j), j > 0$ .
- NP, polinómica no determinista en tiempo, lenguajes del tipo  $NTIEMPO(n^j), j > 0$ .
- PESPACIO, clase polinómica en espacio,  $ESPACIO(n^j), j > 0$ .
- NESPACIO, clase polinómica no determinista en espacio, todos los lenguajes del tipo  $NESPACIO(n^j), j > 0$ .
- L, logarítmica en tiempo determinista,  $ESPACIO(\log(n))$ .
- NL, logarítmica no determinista en tiempo,  $NESPACIO(\log(n))$ .
- EXP, exponencial en tiempo,  $TIEMPO(2^{n^j})$ .

#### Dependencia del Modelo

- **Tesis de Church-Turing:** Todo procedimiento de cálculo físicamente realizable se puede simular por una MT.
- **Tesis de Church-Turing Fuerte:** Todo procedimiento de cálculo físicamente realizable se puede simular por una máquina de Turing, con una sobrecarga polinómica en el número de pasos, si el mecanismo da  $f(n)$  pasos entonces la MT puede dar  $f^k(n)$  pasos.
  - Esta es la versión más controvertida pues no se piensa que se verifica en ordenadores cuánticos.

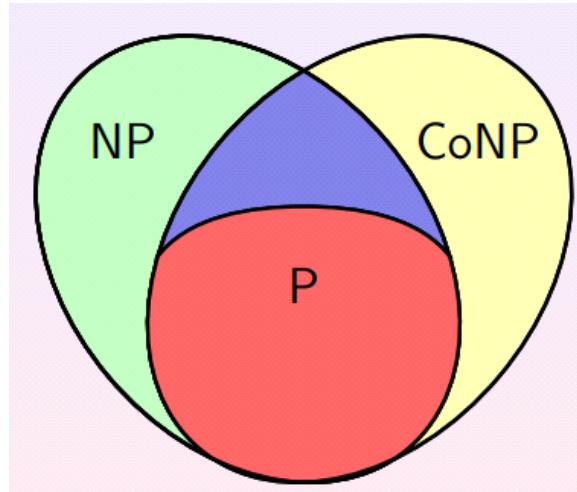
#### Simulación de Máquinas No Deterministas

**WUOLAH**

- **Teorema:** Suponer que  $L$  es decidido por una MTND en tiempo  $f(n)$ , entonces es decidido por una MT determinista con 3 cintas en tiempo  $O(d^{f(n)})$ , donde  $d > 1$  es una constante que depende de la máquina no determinista inicial.
  - Si se supone que  $k$  es el número máximo de opciones de la MTND y que  $k > 1$ , ya que si  $k = 1$  es una máquina determinista y sería trivial.
  - Para  $L = 0, 1, 2 \dots$ 
    - Se coloca en la 3ra cinta todas las secuencias  $a_1, \dots, a_L$  de longitud  $L$  donde cada símbolo se elige entre  $\{1, \dots, k\}$ .
    - Para cada secuencia  $a_1 \dots a_L$  se simula la MTND en  $L$  pasos en una segunda cinta donde en el paso  $i$  se elige la opción  $a_i$ .
    - Si para la secuencia, la simulación acepta, terminar y aceptar.
    - Si para todas las secuencias de longitud  $L$ , se rechaza, entonces terminar y rechazar.
    - Si no ocurre ninguna de las dos, ir al siguiente  $L$ .
- La simulación termina seguro cuando  $L = f(n)$  o antes

### Complementarios de Clases

- La clase de los lenguajes complementarios de los lenguajes en la clase  $C$  se llama  $CoC$ , se verifica que  $L \in C \Leftrightarrow \bar{L} \in CoC$ .
- El complementario de una clase determinista coincide con la propia clase,  $CoP = P$ .
- No ocurre lo mismo en clases no deterministas, la clase  $CoNP$  es el conjunto de problemas cuyo complementario está en  $NP$ .



### Relaciones Binarias

- Una relación binaria  $R$  es un subconjunto  $R \subseteq A^* \times A^*$ , también se puede ver como una aplicación  $R: A^* \times A^* \rightarrow \{0,1\}$  donde  $R(x,y) = 1$  cuando  $(x,y) \in R$ , a veces se escribe como  $xRy$ .

### Relación Calculable Polinómicamente

- $R$  en  $A^* \times A^*$  se dice calculable polinómicamente si existe una MT  $M$  que calcula la función característica  $R$  en tiempo polinómico, o que acepta  $R$  en tiempo polinómico. Es decir, para cada  $x, y$  de entrada calcula si  $R(x,y) = 1$ .

### Definiciones Alternativas

- **NP:** Un lenguaje  $L \subseteq A^*$  está en NP si y solo si existe una relación  $R$  en  $A^* \times A^*$  calculable en tiempo polinómico y un polinomio  $p(n)$  tal que  $L = \{x \in A^*: \exists y \in A^* \text{ con } |y| \leq p(|x|), R(x,y) = 1\}$ .
  - Se dice que los lenguajes o problemas NP se pueden verificar en tiempo polinómico de forma eficiente.
  - Al algoritmo que calcula  $R$  se le llama verificador, a  $y$  se le llama un certificado.
- **CoNP:** Un lenguaje  $L \subseteq A^*$  está en CoNP si y solo si existe una relación  $R$  en  $A^* \times A^*$  calculable en tiempo polinómico y un polinomio  $p(n)$  tal que  $L = \{x \in A^*: \forall y \in A^* \text{ con } |y| \leq p(|x|), R(x,y) = 1\}$

### Ejemplo

- El problema del circuito hamiltoniano está en NP porque se puede expresar como "determinar los  $x$ , que son los grafos, para los que existe un  $y$ , que son la sucesión de nodos, tal que todos los nodos aparecen una vez y solo una vez y existe un arco desde cada nodo al siguiente y del último al primero, que es la relación  $R$ .
  - La condición que pide para  $x$  e  $y$  y la relación  $R(x, y)$  se puede calcular en tiempo polinómico.
  - La longitud del  $y$  que cumple la relación es menor que la de  $x$ , entonces la longitud de  $y$  está acotada por un polinomio de la longitud de  $x$ .

### Problemas NP

- Son aquellos que se pueden expresar como "dado unos datos  $x$ , comprobar si existe un objeto  $y$  con un tamaño limitado a un polinomio de tamaño  $x$ , tal que cumple una condición  $R(x, y) = 1$  que es verificable en tiempo polinómico.
  - "Saber si un número  $x$  es compuesto: si existe  $1 < y < x$  tal que la división entera de  $x$  entre  $y$  es exacta".

### Sistemas Interactivos de Demostración:

#### Condiciones

- Existe un demostrador con capacidad ilimitada de cálculo, siempre quiere convencer de que la respuesta es positiva.
- Existe un verificador con capacidad polinómica de cálculo, quiere saber la verdad.
- Se pueden intercambiar mensajes de un tamaño polinómico en función de una palabra inicial  $x$ .

#### Clase NP

- Es la clase del problemas que el verificador puede decidir, teniendo en cuenta que si la respuesta es positiva, el demostrador tratará convencer al verificador que lo es.

### Relaciones entre Clases de Complejidad:

- a)  $\text{ESPACIO}(f(n)) \subseteq \text{NESPACIO}(f(n))$   
 $\text{TIEMPO}(f(n)) \subseteq \text{NTIEMPO}(f(n))$
- b)  $\text{NTIEMPO}(f(n)) \subseteq \text{ESPACIO}(f(n))$
- c)  $\text{NESPACIO}(f(n)) \subseteq \text{TIEMPO}(k^{f(n)})$  para un  $k > 1$

- Verificar a) es trivial
- Verificar b) se basa en la simulación de una MTND mediante una determinista, donde se ponen palabras  $a_1, \dots, a_L$  donde  $a_i \in \{1, \dots, k\}$  e ir simulando la MT por una determinista que en el paso  $i$  selecciona la opción  $i$ , el espacio que hace falta es el espacio para escribir las palabras que es menor o igual a  $f(n)$  y el espacio para hacer la simulación de  $L$  pasos, que también es de orden  $f(n)$ , en total el espacio necesario es de orden  $f(n)$ .
- Demostrar c) implica utilizar el método de alcanzabilidad, se basa en considerar un grafo en el que todos los nodos son posibles configuraciones de una MT y los arcos conectan configuraciones tales que se puede llegar de una a otra en un paso de cálculo; si se supone una tripleta  $(q, u, v)$ , como no se ocupa más de  $f(n)$  en espacio, la longitud de  $u$  y  $v$  es menor o igual a  $f(n)$ , por lo tanto el número de configuraciones es del orden  $|Q||B|^{2f(n)}$ , donde  $B$  es el alfabeto de trabajo, entonces es del orden  $t^{f(n)}$  donde  $t = |B|^2$ .

### Teorema de la Jerarquía:

**QUIERES  
CONSEGUIR  
15€??**

TRÁENOS A TU  
CRUSH DE APUNTES  
ANTES DE QUE  
LOS QUEME



- **Funciones de Complejidad Propias:** Aquellas que verifican  $f(n+1) \geq f(n)$  y tales que la función  $g(u) = f(|u|)$  expresando  $g(u)$  como una secuencia de longitud  $g(u)$ , es decir usando unario, es calculable en espacio  $O(f(n))$  y tiempo  $O(f(n) + n)$ .
  - O sea que existen problemas en de  $O(n^4)$  que no se pueden resolver en  $O(n^2)$  u  $O(n^8)$  que en  $O(n^4)$  no se pueden.

#### Teorema de la jerarquía en Tiempo

Si  $f(n) \geq n$  es propia, entonces

$$\text{TIEMPO}(f(n)) \subset \text{TIEMPO}(f^2(n))$$

$$\text{TIEMPO}(f(n)) \neq \text{TIEMPO}(f^2(n))$$

- Corolario:  $P \neq EXP$

#### Jerarquía en Espacio

Si  $f(n)$  es una función de complejidad propia, entonces

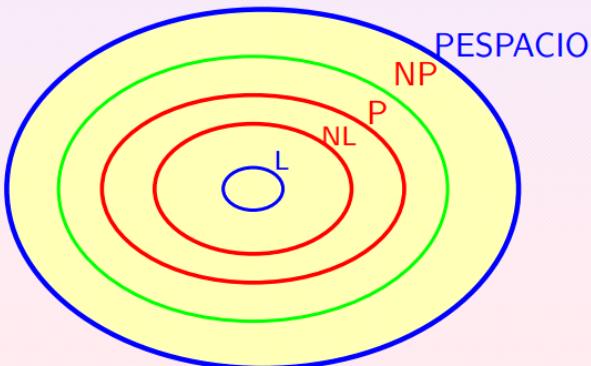
$$\text{ESPACIO}(f(n)) \subset \text{ESPACIO}(f(n) \log f(n))$$

$$\text{ESPACIO}(f(n)) \neq \text{ESPACIO}(f(n) \log f(n))$$

#### Inclusión de Clases:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq \text{PESPACIO}$$

No se sabe si alguna o varias de estas inclusiones son igualdades. Lo único que se sabe es que, **NL** está incluida estrictamente en **PESPACIO**.



Espacio No Determinista



si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

### Teorema

Si  $f(n) \geq \log(n)$  y es propia, entonces

$$\text{NESPACIO}(f(n)) \subseteq \text{ESPACIO}(f^2(n))$$

### Corolario

**PESPACIO = NPESPACIO**

### Teorema

Si  $f(n) \geq \log(n)$  es propia entonces

$$\text{NESPACIO}(f(n)) = \text{CoNESPACIO}(f(n))$$

# Tema-4-NP-Completitud.pdf



ParmigianoReg



Modelos Avanzados de Computación (Especialidad  
Computación y Sistemas Inteligentes)



3º Grado en Ingeniería Informática



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada

**QUIERES  
CONSEGUIR  
15E??**

→ TRÁENOS A TU  
CRUSH DE APUNTES

ANTES DE QUE  
LOS QUEME



**WUOLAH**

**QUIERES  
CONSEGUIR  
15€??**

TRÁENOS A TU  
CRUSH DE APUNTES  
ANTES DE QUE  
LOS QUEME

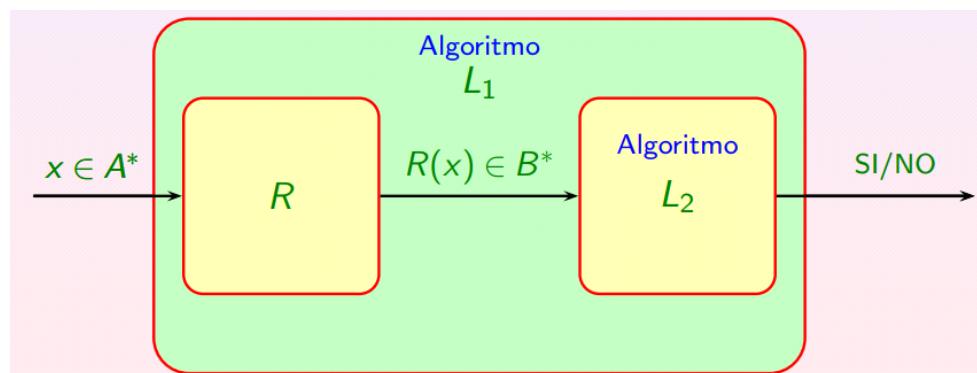


## Tema 4 - NP-Compleitud

Profesor Serafín Moral Callejón, Curso 21-22

### Reducciones:

- Se dice que  $L_1$  es reducible a  $L_2$ ,  $L_1 \propto L_2$  si y solo si existe una MT determinista que en espacio logarítmico calcula una función  $R: A^* \rightarrow B^*$  de tal manera que  $x \in L_1 \Leftrightarrow R(x) \in L_2$ .
- Un lenguaje reducible a otro es también que un algoritmo sea reducible a otro.



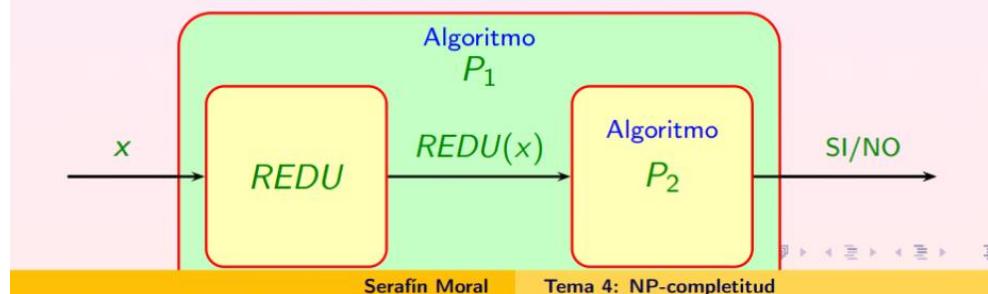
### En términos de problemas

- Un problema de decisión  $P_1(x)$  es reducible a un problema de decisión  $P_2(x)$  donde  $x \in X, y \in Y$ , se puede escribir como  $P_1(x) \propto P_2(x)$  si y solo si existe un algoritmo determinista que en espacio logarítmico calcula la función  $REDU: X \rightarrow Y$  de tal manera que  $P_1(x) = P_2(REDU(x))$



si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

**Problema  $P_1(x)$  reducible a  $P_2(y)$ :** Algoritmo  $P_2 \rightarrow$  Algoritmo  $P_1$



- La reducción de  $P_1$  a  $P_2$  indica que si se obtuviera una solución sencilla para  $P_2$  entonces componiéndola con la reducción se podría obtener una solución para  $P_1$ .
- Esto significa que si se obtuviera una solución sencilla para  $P_2$ , se tendría una para  $P_1$ , pero lo contrario, que se tenga una solución sencilla para  $P_1$  no significa que sea sencilla para  $P_2$ .
- En definitiva,  $P_2$  es al menos tan difícil como  $P_1$ , se resiste más a que se encuentre una solución sencilla.
- Es una forma de comparar problemas.

### Reducciones Espacio Logarítmicas

- Se ha supuesto que las reducciones tienen que ser espacio logarítmicas, en otros textos se suponen de tiempo polinómico. La idea es que la reducción -- la transformación entre problemas -- tiene que

poder hacerse "eficientemente", por lo tanto se supondrá que deben de ser logarítmicas.

### Pasos en la Reducción de Problemas

- Si se tienen problemas  $P_1(x)$  y  $P_2(y)$  para demostrar  $P_1(x) \propto P_2(y)$  hay que:
  - Dar un algoritmo  $REDU$  que transforme cada entrada  $x$  de  $P_1$  en una entrada  $y = REDU(x)$  del problema  $P_2$ .
  - Comprobar que dicho algoritmo es logarítmico en espacio.
  - Comprobar que si  $P_1(x)$  tiene solución positiva,  $P_2(y)$  también la tiene.
  - Comprobar que si  $P_2(y)$  tiene solución positiva,  $P_1(x)$  también la tiene.

Por ejemplo...

#### Reducción de Camino Hamiltoniano a Viajante de Comercio

- El TSP,  $VC$ , indica que, dado un conjunto finito de ciudades, una función de distancia y una cota, ¿existe un circuito que visite todas las ciudades una vez y de coste no superior a la cota?

Es decir, determinar si existe un orden de las ciudades  $(c_{\pi(1)}, \dots, c_{\pi(n)})$  tal que

$$\left( \sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \right) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B$$

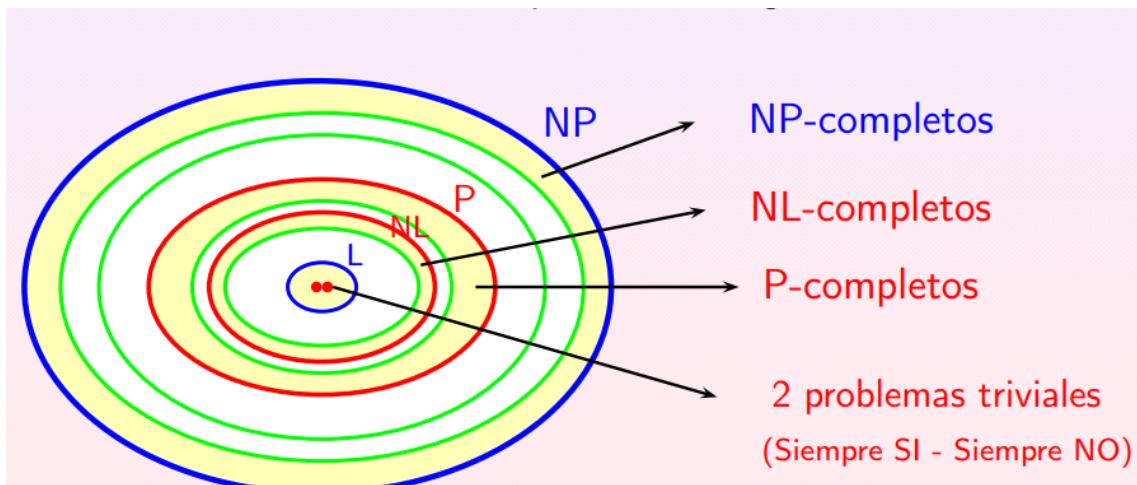
- El circuito hamiltoniano,  $CH$ , indica que, dado un grafo, se puede trazar un camino que visite todos los nodos una sola vez y vuelva al nodo inicial.
- La reducción de  $CH \propto VC$ , se puede hacer tal que el número de ciudades son los nodos, la distancia es 1 si existe conexión entre nodos y 2 sino, y la cota sea el número de ciudades.
- Si existe una solución que sea el valor exacto del número de ciudades, es porque solo existe un arco entre cada ciudad, la transformación se puede calcular en espacio logarítmico.
  - Si existe un circuito hamiltoniano, existe un orden de viaje menor o igual al total de ciudades.
  - Si existe un orden de viaje de valor menor o igual al total de ciudades, todos los arcos son de coste 1 y es un circuito hamiltoniano.

#### Composición de Reducciones

- Si  $L_1 \propto L_2 \propto L_3$ , entonces  $L_1 \propto L_3$ .
- La demostración del porqué, es que si se tiene una MT  $M_1$  que calcula de  $L_1$  a  $L_2$  y otra  $M_2$  que calcula de  $L_2$  a  $L_3$ , se puede tener un espacio intermedio, cada vez que  $M_2$  necesita una casilla,  $M_1$  la calcula directamente.
- $L_1$  y  $L_2$  se les dice equivalentes si y solo si  $L_1 \propto L_2$  y  $L_2 \propto L_1$ .

#### Lenguajes NP-Completos:

- Un lenguaje es NP-Completo si y solo si es un lenguaje NP y cualquier otro lenguaje NP se reduce a él.
  - La primera condición es fácil de comprobar, la segunda no.
  - Los lenguajes NP-Complejos son los más difíciles o típicos dentro de la clase NP.
  - Se puede extender a otras clases, un lenguaje P-Complejo es así solo si es un lenguaje P y cualquier otro lenguaje P se reduce a él.
- La relación de equivalencia que divide el conjunto de los lenguajes en clases de equivalencia, en particular esta relación sobre NP queda tal que:



### Consistencia Lógica Proposicional (SAT):

- **Datos:** Un conjunto  $U = \{p_1, \dots, p_m\}$  de símbolos posicionales y una colección  $C$  de cláusulas sobre estos símbolos.
- **Pregunta:** ¿Son consistentes las cláusulas?

- $U = \{p_1, p_2\}, C = \{p_1 \vee \neg p_2, \neg p_1 \vee p_2\}$

La respuesta es **SI**, ya que todas las cláusulas se satisfacen haciendo  $p_1$  y  $p_2$  verdaderas. En notación matemática  $t(p_1) = t(p_2) = V$ .

#### **Teorema de Cook-Levine**

- Este teorema indica que SAT es NP-Completo

#### **Demostración**

- Primero es demostrar que el problema es NP-Completo, esto se realiza con un algoritmo no-determinista.
  - Para cada símbolo en  $U$  seleccionar un valor de verdad, para cada cláusula comprobar si se satisface: si todas satisfacen, responder SI, sino, NO.
- Ahora hay que demostrar que si existe un lenguaje  $L$  y este está en NP, se puede reducir a SAT, por medio de una MT que traslada este teórico problema del lenguaje  $L$  a una instancia de SAT, entonces dependiendo de la ejecución de esta MTND, si esta acepta entonces es un problema SAT afirmativo, y si se tiene un problema SAT afirmativo, también parará la MNTD aceptando; además la transformación se realiza en espacio logarítmico.

#### **Equivalencia de Problemas**

- La reducción que se realizó para el Teorema de Cook es jodida y está en espacio logarítmico en función de la longitud, pero la estrategia para demostrar problemas con NP-Complejidad, se tiene que...
  - Demostrar que  $L$  está en NP.
  - Determinar que un problema NP-Completo ya conocido,  $L'$  se reduce a  $L$ , o sea  $L' \leq L$ .
    - Como la solución es transitiva, esta última condición garantiza que cualquier otro problema NP se reduce a  $L$ .

### 3-SAT:

- Se tienen dos formas de entender 3-SAT, a los más existen 3 literales o exactamente 3 literales.
  - De ambas formas, son NP-Completas.
- Es NP ya que es un caso particular de SAT y éste ya estaba en NP: Vale el mismo algoritmo.

- Para demostrar que es completo, en vez de hacer el Teorema de Cook, se reduce de SAT a 3-SAT.

### **Reducción SAT $\propto$ 3 – SAT**

- Se tienen los conjuntos  $U$  y cláusulas  $C$ , 3-SAT tiene los mismos conjuntos  $U$  pero con cláusulas  $C'$ , que serán de longitud máxima 3.
  - Las cláusulas con 3 o menor que 3, se añaden directamente.
  - Las cláusulas con más de 3, se dividen las cláusulas iterativamente hasta que sean de longitud 3 o menos; se tiene que añadir un símbolo nuevo para llevar el valor de verdad de la cláusula original, esto se basa en el hecho de que si se tiene  $P \vee Q$ , esto es equivalente a  $P \vee x \vee Q \vee \neg x$ ,  $x$  siendo el símbolo que mantiene el valor de verdad; si las dos fórmulas son ciertas entonces también  $P \vee Q$  es cierta.

- Para cada cláusula en  $C$  de longitud mayor o igual que 4:

$I_1 \vee I_2 \vee \dots \vee I_k, \quad k \geq 4$ , donde  $I_1, \dots, I_k$  son literales, añado símbolos proposicionales  $x_2, \dots, x_{k-2}$  y las cláusulas

$$I_1 \vee I_2 \vee x_2, \quad \neg x_2 \vee I_3 \vee x_3, \quad \dots, \quad \neg x_{k-3} \vee I_{k-2} \vee x_{k-2}, \quad \neg x_{k-2} \vee I_{k-1} \vee I_k,$$

- Para que este algoritmo sea espacio logarítmico, se deben de calcular iterativamente no de forma recursiva, es decir, calculando las cláusulas intermedias.

### **3-SAT Restringido**

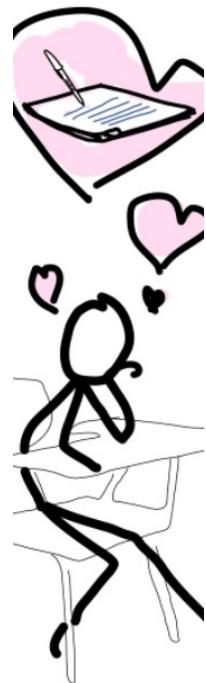
- El problema 3-SAT sigue siendo NP-Completo si se exige que todas las cláusulas tengan exactamente 3 literales distintos.
- 3-SAT se puede reducir a este problema, añadiendo  $x_1, x_2, x_3$  a las cláusulas menores que 3 para que sean 3, por ejemplo,  $\neg t$  se transforma a  $\neg t \vee x_1 \vee x_2$ .

### **2-SAT**

- El problema 2-SAT está en espacio NL, y por lo tanto, en P.
- Se puede ver el problema como un grafo dirigido, donde los nodos son las variables y sus negaciones; y los arcos  $(\alpha, \beta)$  son arcos solo si  $(\neg \alpha \vee \beta)$  está en el problema.

**QUIERES  
CONSEGUIR  
15€??**

TRÁENOS A TU  
CRUSH DE APUNTES   
ANTES DE QUE  
LOS QUEME 



$C = \{x_1 \vee x_2, x_1 \vee \neg x_3, \neg x_1 \vee x_2, x_2 \vee x_3\}$

**Grafo:**

$x_1 \vee \neg x_3$   
 $\neg x_1 \vee x_2$   
 $\neg x_3 \vee x_2$

Las deducciones son los caminos.

**Teorema**

El conjunto de cláusulas  $C$  es consistente si y solo si no existe un par de nodos  $x, \neg x$ , tal que existe un camino de  $x$  a  $\neg x$  y otro camino de  $\neg x$  a  $x$ .

- Si hay un camino de  $x$  a  $\neg x$ , se deduce  $\neg x$ .
- Si hay un camino de  $\neg x$  a  $x$ , se deduce  $x$ .
- Esto demuestra que el complementario de 2-SAT está en NL, pero como NL=CoNL, 2-SAT está en NL.

#### Cláusulas Horn:

- Son aquellas que, a lo más, existe un literal positivo.
- Idea del algoritmo polinómico
  - $V$  es el conjunto de variables
  - Se consideran conjuntos  $C_1$  de literales negativos y  $C_2$ , existe un literal positivo.
  - Sea  $H$  un conjunto de variables proposicionales que inicialmente es igual a las variables que aparecen en las cláusulas de longitud 1 de  $C_2$ . La idea es que este conjunto contenga las variables que tienen que ser necesariamente verdad.
  - Mientras  $H$  cambie, examinar las cláusulas de  $C_2$ , si alguna es de la forma  $\neg z_1 \vee \neg z_2 \vee \dots \vee \neg z_i \vee y$  con todas las variables  $z_1, \dots, z_i$  en el conjunto  $H$ , si  $y$  no está en  $H$ , entonces añadir  $y$  a  $H$ .
  - Una vez calculado  $H$ , se hacen verdad las variables de este conjunto y falsas las de  $V/H$ .
  - La consistencia es equivalente a que para cada cláusula en  $C_1$  exista un literal con su variable en  $V/H$ .

#### MAX2SAT:

- **Datos:** Un conjunto de cláusulas con dos literales y un valor  $K \geq 0$
- **Pregunta:** ¿Puede satisfacerse, al menos,  $K$  cláusulas?
- Es una generalización de 2-SAT y es NP-Completo.

#### **Reducción 3 – SAT $\propto$ MAX2SAT**

- Cada cláusula de longitud 3,  $x \vee y \vee z$  se transforma en 10 cláusulas, junto con una variable  $w$  nueva.



si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

$$\begin{array}{ccccccccc} x, & y, & z, & w, & \neg x \vee \neg y, & \neg y \vee \neg z, & \neg z \vee \neg x, \\ x \vee \neg w, & y \vee \neg w, & z \vee \neg w \end{array}$$

- Se selecciona  $K = 7m$ , con  $m$  siendo el número de cláusulas, se hace con la técnica de construcción de Gadgets.
  - Esto se puede comprobar viendo que si  $x \vee y \vee z$  es verdad, se puede elegir el valor de verdad de  $w$  de manera que se satisfagan 7 pero no más cláusulas; si  $x \vee y \vee z$  es falso, entonces nunca podemos llegar a satisfacer 7 o más de las 10 cláusulas, sea cual sea el valor de  $w$ .

### Frontera entre P y NP:

- Muchos problemas planteados con suficiente generalidad son NP-Completo, hay restricciones que no llegan a convertirlos en problemas P, pero siempre, si se restringe el número de casos a resolver lo suficiente, se llega a un problema polinomial.

#### **Restricción a 3-SAT**

- 3-SAT es NP-Completo si se hace que cada literal nunca aparezca más de dos veces y cada variable más de tres veces, pero pueden existir cláusulas de longitud menor que 3.
- Se puede convertir cualquier ejemplo de 3-SAT para que cumpla esto.
  - Si una variable  $x$  no cumple las condiciones de la restricción apareciendo  $k$  veces en las cláusulas, se substituye  $x$  por  $k$  nuevas variables  $x_1, \dots, x_k$  una por cada aparición.
  - Se añaden las cláusulas  $\neg x_1 \vee x_2, \neg x_2 \vee x_3, \dots, \neg x_k \vee x_1$ , que son equivalentes a  $x_1 \rightarrow x_2, x_2 \rightarrow x_3, \dots, x_k \rightarrow x_1$  con lo que todas las versiones  $x_i$  tienen que tener el mismo valor de verdad.

### NAESAT:

- Dado un conjunto de cláusulas de longitud 3, determinar si existe una asignación de valores de verdad tal que para cada cláusula alguno, pero no todos los literales sean ciertos.
- Es NP, ya que si se pueden asignar de forma no determinista los valores de verdad a las variables y después comprobar en tiempo polinómico si se verifican las condiciones especificadas: en cada cláusula hay un literal cierto y otro falso.

#### **Reducción 3 – SAT $\leq$ NAESAT**

- Se añade una variable  $z$  nueva, para cada cláusula con menos de 3 literales.
- Si los 3 literales son distintos,  $p \vee q \vee r$ , se añade una variable  $s$  tal que  $p \vee q \leftrightarrow s$  y las cláusulas

$$s \vee r \vee z, \quad p \vee q \vee \neg s, \quad \neg p \vee s \vee z, \quad \neg q \vee s \vee z$$

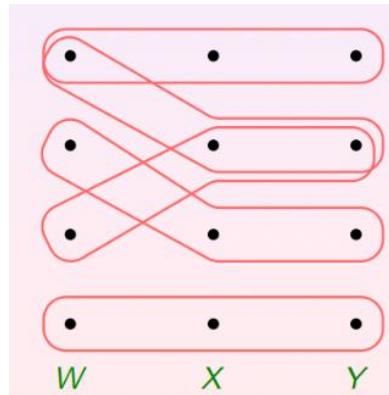
- Si NAESAT tiene solución, entonces cambiando el valor de verdad de todas las variables, sigue teniendo solución, se elige la solución con  $z$  falsa, es una solución al problema original.
- Si se satisface SAT, se pueden satisfacer todas las cláusulas nuevas de acuerdo con NAESAT con el mismo valor de verdad para las variables que existían y haciendo  $z$  falso y  $s$  verdadero si  $p$  o  $q$  son verdaderos.

### Isomorfismo de Grafos:

- Dados dos grafos no dirigidos  $G_1 = (V_1, E_1)$  y  $G_2 = (V_2, E_2)$  determinar si existe un isomorfismo entre  $G_1$  y  $G_2$ , esto es una aplicación biyectiva  $f: V_1 \rightarrow V_2$  tal que  $(u, v) \in E_1 \leftrightarrow (f(u), f(v)) \in E_2$ .
- Este es un problema que no se ha demostrado que sea NP-Completo, tampoco se conoce algún algoritmo polinómico para resolverlo, se supone que no está ni en P ni es NP-Completo.
  - Se ha definido una clase GI de todos los problemas que se pueden reducir al problema de isomorfismo de grafos, por lo tanto el problema es GI-Completo.

### ACTRI:

- El problema de Acoplamiento de Triplets
- **Datos:** Tres conjuntos disjuntos  $W, X, Y$  de tamaño  $q$  y un subconjunto  $M \subseteq W \times X \times Y$  de compatibilidades.
- **Pregunta:** ¿Contiene  $M$  un subconjunto  $M' \subseteq M$  con  $q$  elementos, tal que para cada  $(w_1, x_1, y_1), (w_2, x_2, y_2) \in M'$ , si  $(w_1, x_1, y_1) \neq (w_2, x_2, y_2)$  entonces  $w_1 \neq w_2, x_1 \neq x_2, y_1 \neq y_2$ ?



- Es NP, ya que un algoritmo no determinista que elige un subconjunto de  $M$ , y comprueba si hay en cada elemento del subconjunto uno y solo uno de los elementos de cada conjunto tiene tiempo polinómico.

### **Reducción 3 – SAT $\propto$ ACTRI**

- Se supone un ejemplo de 3-SAT con  $U = \{u_1, u_2, \dots, u_n\}$  y  $C = \{c_1, c_2, \dots, c_m\}$ , de aquí se construye un ejemplo de ACTRI con la misma solución.
  - Por cada variable  $u_i$  se introduce en  $W$  los elementos  $u_i[j], \bar{u}_i[j]$  con  $j = 1, \dots, m$ .
  - En  $X$ ,  $a_i[j]$  con  $j = 1, \dots, m$ .
  - En  $Y$ ,  $b_i[j]$  con  $j = 1, \dots, m$ .

En  $M$  se introducen las tripletas:  $T_i^t = \{(\bar{u}_i[j], a_i[j], b_i[j]) : 1 \leq j \leq m\}$  y  $T_i^f = \{(\bar{u}_i[j], a_i[j+1], b_i[j]) : 1 \leq j < m\} \cup \{(\bar{u}_i[m], a_i[1], b_i[m])\}$ .

#### Ejemplo:

$$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}$$

$\{(\bar{u}_1[1], a_1[1], b_1[1]), (\bar{u}_1[2], a_1[2], b_1[2]), (u_1[1], a_1[2], b_1[1]), (u_1[2], a_1[1], b_1[2]), (\bar{u}_2[1], a_2[1], b_2[1]), (\bar{u}_2[2], a_2[2], b_2[2]), (u_2[1], a_2[2], b_2[1]), (u_2[2], a_2[1], b_2[2]), (\bar{u}_3[1], a_3[1], b_3[1]), (\bar{u}_3[2], a_3[2], b_3[2]), (u_3[1], a_3[2], b_3[1]), (u_3[2], a_3[1], b_3[2]), (\bar{u}_4[1], a_4[1], b_4[1]), (\bar{u}_4[2], a_4[2], b_4[2]), (u_4[1], a_4[2], b_4[1]), (u_4[2], a_4[1], b_4[2]), \}$

- Por cada cláusula en  $C$ , se añade...
  - Un elemento  $s[j]$  a  $X$
  - Un elemento  $t[j]$  a  $Y$
  - Añadir a  $M$  las tripletas

$$\{(u_i[j], s[j], t[j]) : u_i \text{ está en } c_j\} \cup \\ \{(\bar{u}_i[j], s[j], t[j]) : \neg u_i \text{ está en } c_j\}$$

-- Le falta --

### 3-SET:

- **Datos:** Un conjunto finito  $X$  con  $|X| = 3q$  y un subconjunto  $C$  de subconjuntos de  $X$  de tres elementos
- **Pregunta:** ¿Existe  $C' \subseteq C$  tal que  $X = \bigcup_{A \in C'} A$  y los elementos de  $C'$  son disjuntos dos a dos,  $A, B \in C'$  y  $A \neq B$  entonces  $A \cap B = \emptyset$ ?

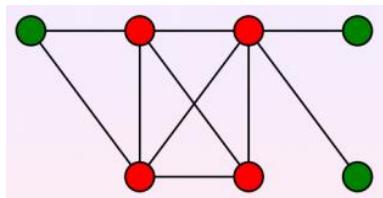
#### **Reducción ACTRI $\propto$ 3 – SET**

- Se tiene un ejemplo de ACTRI dado por  $W, Y, Z$  y el subconjunto de tripletas  $M$ , se construye el ejemplo 3-SET donde  $X = W \cup Y \cup Z$  y  $C = \{\{w, y, z\} : (x, y, z) \in M\}$ .

### Problemas con Grafos:

#### **Problema del Clique Máximo (CLIQUE)**

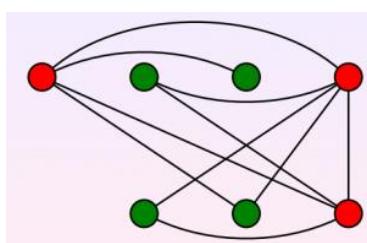
- Un clique es, dado un grafo  $G = (V, E)$ , un subconjunto maximal totalmente conectado, es decir, un subconjunto  $V_t \subseteq V$  tal que  $\forall v_1, v_2 \in V_t$  se tiene que  $(v_1, v_2) \in E$  y que no está estrictamente incluido en otro conjunto que cumpla esta propiedad.
- El problema es entonces, dado un grafo y un número natural  $J \leq |V|$  determinar si existe un clique de tamaño  $\geq J$ .
  - O sea que, sería ver si existe un conjunto totalmente conectado de tamaño  $\geq J$ .



- Ejemplo, Clique de 4.

#### **Cubrimiento por Vértices (CV)**

- Dado un grafo  $G = (V, E)$  y un subconjunto  $V_c \subseteq V$ , se dice que  $V_c$  es un cubrimiento por vértices de  $G$  si y solo si toda arista del grafo tiene un extremo en  $V_c$ :  $\forall (u, v) \in E, (u \in V_c \vee v \in V_c)$
- **Problema:** Dado un grafo y un número natural  $K \leq |V|$ , determinar si existe un cubrimiento por vértices de tamaño menor o igual que  $K$ .
  - O sea que si se tiene un grupo  $\leq K$  de vértices tales que alguna arista esté en ese grupo.



- Ejemplo, Cubrimiento de 3.

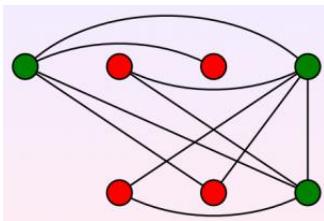
**QUIERES  
CONSEGUIR  
15€??**

TRÁENOS A TU  
CRUSH DE APUNTES



### Conjunto Independiente (CI)

- Dado un grafo  $G = (V, E)$  y un subconjunto  $V_i \subseteq V$ , se dice que  $V_i$  es un conjunto independiente de  $G$  si, y solo si no hay ninguna arista que une vértices de  $V_i$ :  $\forall u, v \in V_i, (u, v) \notin E$ .
- Problema:** Dado un grafo y un número natural  $J \leq |V|$ , determinar si existe un conjunto independiente de tamaño mayor o igual que  $J$ .



- Ejemplo, conjunto independiente de 4

### Lema

- Si  $G = (V, E)$  es un grafo y  $V^* \subseteq V$  entonces las siguientes condiciones son equivalentes:
  - $V^*$  es un cubrimiento por vértices de  $G$
  - $V \setminus V^*$  es un conjunto independiente de  $G$
  - $V \setminus V^*$  es un subgrafo totalmente conectado del grafo complementario,  $\bar{G} = (V, \bar{E})$ .

### Equivalencia de Problemas

- Los tres problemas, Clique, CV y CI, son equivalentes y si uno es NP-Completo, los otros también lo son.
- Si  $n$  es el número de nodos de  $G$  entonces...
  - $G$  tiene un recubrimiento por vértices de tamaño menor o igual que  $K$ , esto equivale a que  $G$  tenga un conjunto independiente de tamaño mayor o igual que  $K$ , esto equivale a que  $\bar{G}$  tenga un clique de tamaño mayor o igual que  $n - K$ .

### NP-Completitud

- Cubrimiento por Vértices es NP-Completo, se elige de forma no-determinista un subconjunto de vértices y se comprueba en tiempo polinomial si es un cubrimiento.

### Reducción 3 – SAT $\propto$ CV

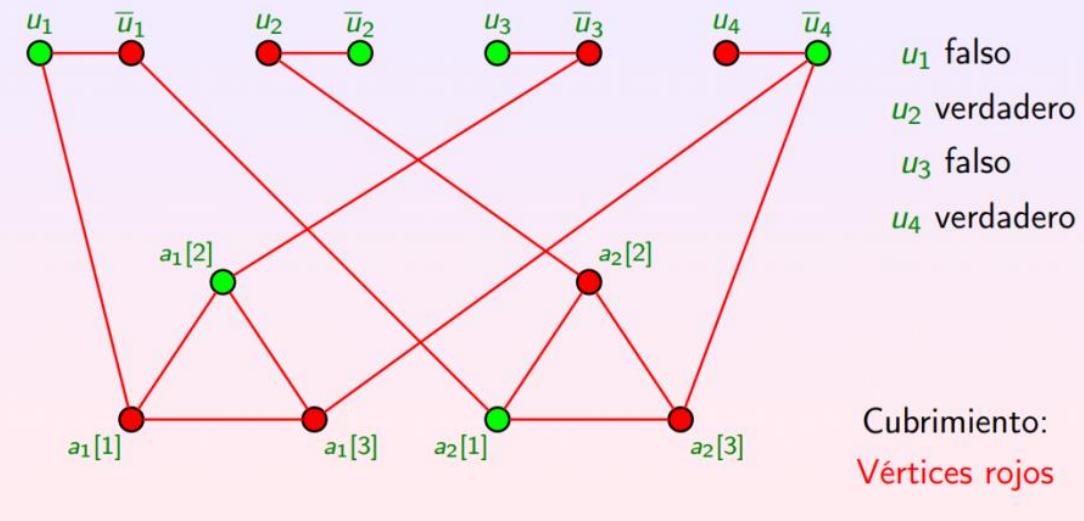
- Sea 3-SAT con las variables  $U$  y cláusulas  $C$ .
- Sea CV un grafo  $G = (V, E)$
- Para cada variable  $u_i \in U$  se añaden dos vértices,  $u_i, \bar{u}_i$  y un arco que los une.
- Para cada cláusula  $c_j \in C$  se añaden tres vértices,  $a_1[j], a_2[j], a_3[j]$  y tres aristas de forma que se cree un triángulo.
- Para cláusula  $c_j \in C$ , si en el literal número  $k$  de esta cláusula aparece la variable  $u_i$ , se añade una arista  $u_k[j]$  al vértice  $u_i$  si la variable aparece positiva y al vértice  $\bar{u}_i$  si la variable aparece negada.
- Se pone el límite de  $K = n + 2m$ , numero de variables + 2 × numero de cláusulas



si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

### Ejemplo de 3-SAT:

$$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}.$$



### CV implica SAT

- Si existe recubrimiento  $V_c$  entonces
  - Debe de haber al menos un vértice de cada pareja,  $u_i, \bar{u}_i$ .
  - Para cada cláusula  $C_j$  deben de existir, al menos, dos vértices de cada conjunto:  $\{a_1[j], a_2[j], a_3[j]\}$ .
- Como el recubrimiento tiene a lo más  $n + 2m$  vértices, entonces habrá exactamente un vértice por cada pareja  $u_i, \bar{u}_i$  y dos por cada conjunto  $\{a_1[j], a_2[j], a_3[j]\}$ .
  - Las cláusulas se pueden satisfacer haciendo cada variable  $u_i$ ...
    - $u_i = V$  si  $u_i \in V_c$
    - $u_i = F$  si  $u_i \notin V_c$
- Cada cláusula  $C_j$  tiene tres vértices, y cada vértice está conectado con un vértice de variable, de estos tres arcos, hay dos que tienen extremos en los dos vértices de  $\{a_1[j], a_2[j], a_3[j]\}$  que están en  $V_c$ , el otro arco tendrá que tener su extremo del cubrimiento en los vértices correspondientes a las variables  $u_i$  o  $\bar{u}_i$ : El valor de verdad asignado a dicha variable hace que esa cláusula se satisfaga.

### SAT implica CV

- Sea una asignación de valores de verdad que haga consistentes las cláusulas, entonces un cubrimiento por vértices del tamaño deseado se consigue de la siguiente forma:

$$V_c = \{u_i : u_i \text{ es verdadero}\} \cup \{\bar{u}_i : u_i \text{ es falso}\} \cup (\bigcup_j (\{a_1[j], a_2[j], a_3[j]\} - \{a_i[j]\}))$$

donde  $a_i[j]$  corresponde al literal que hace verdadera la cláusula  $C_j$ .

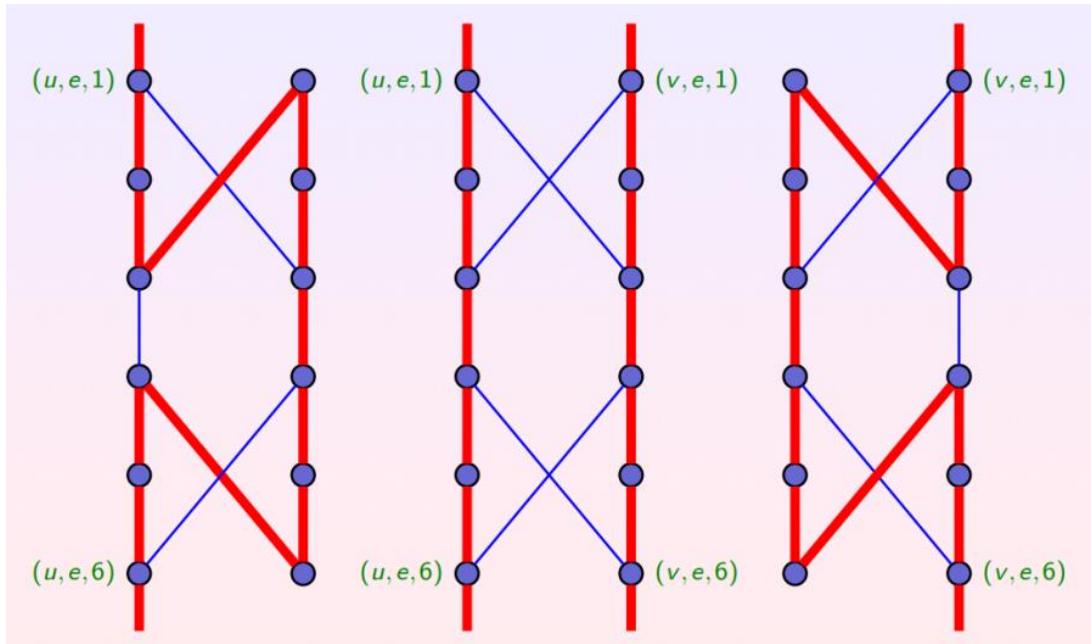
### El Problema del Circuito Hamiltoniano (CH):

- Es inmediatamente NP, el algoritmo no-determinista polinómico solo tiene que elegir  $n$  nodos, el número de nodos en el grafo, y después comprobar que hay arco desde cada nodo al siguiente y del último hasta el primero.

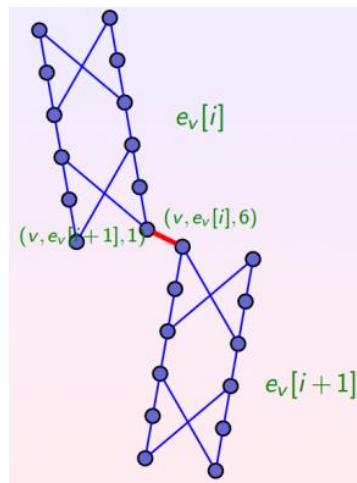
### Reducción $CV \propto CH$

- Para la reducción, se construye un grafo  $G'$  de tal forma que la existencia de un circuito hamiltoniano para  $G'$  sea equivalente a la existencia de un recubrimiento de tamaño  $\leq K$  para  $G$ .

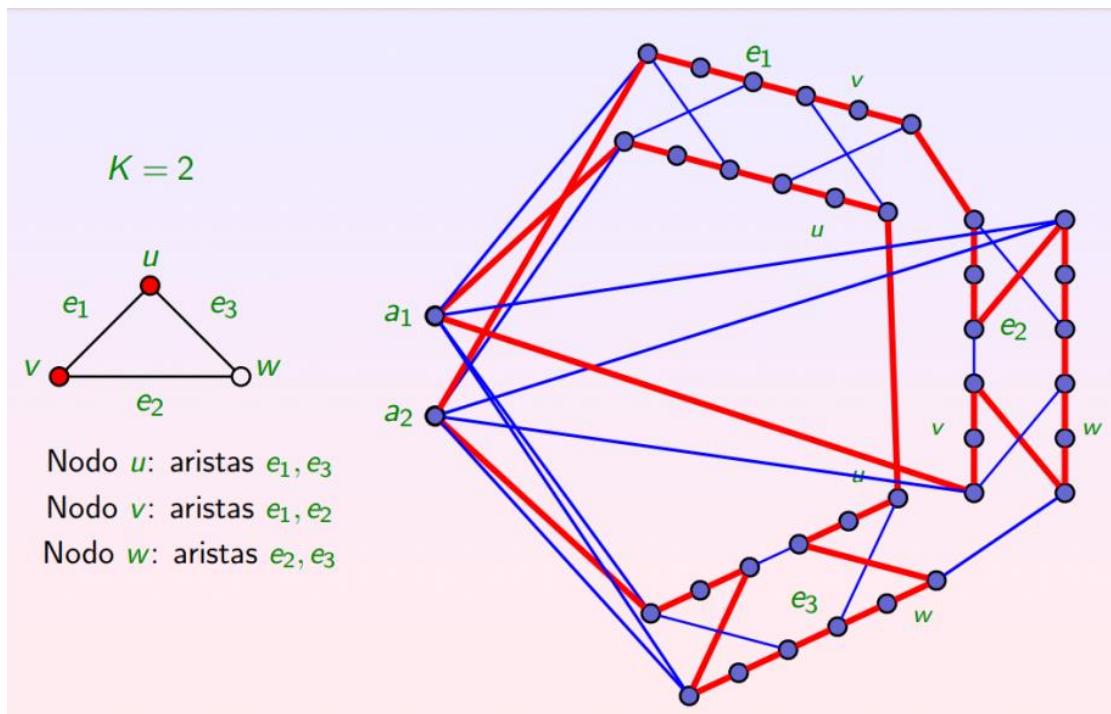
- Para cada  $e = (u, v) \in E$  se añade el siguiente grafo a  $G'$



- Estos son los recorridos válidos que se pueden hacer en estos nodos nuevos.
- Para cada  $v \in V$  sea  $e_v[1], \dots, e_v[r_v]$  una ordenación arbitraria de los arcos que pertenecen a  $v$ , para  $1 \leq i < r_v$  se une el subgrafo asociado a  $e_v[i]$  y el subgrafo  $e_v[i + 1]$  mediante un arco que va de  $(v, e_v[i], 6)$  a  $(v, e_v[i + 1], 1)$



- Como último, se añaden  $K$  vértices  $a_1, a_2, \dots, a_k$  que se unen con los subgrafos de la siguiente forma:
  - Para cada  $v \in V$  sea  $e_v[1], \dots, e_v[r_v]$ , la lista de las aristas que lo contienen en el orden que se consideró anteriormente.
  - Se añade una arista de cada uno de los  $a_i$  a  $(v, e_v[1], 1)$ , vértice extremo de la primera arista de  $v$ .
  - Se añade una arista de cada uno de los  $a_i$  a  $(v, e_v[r_v], 6)$ , vértice extremo de la última arista correspondiente a  $v$ .



### Equivalencia de Soluciones

**CV → CH**

- Sean los  $K$  vértices que forman el recubrimiento,  $\{v_1, \dots, v_K\}$ , un circuito hamiltoniano se puede construir comenzando en  $a_1$ , si se está en  $a_i$ , desde él se recorren todos los grafos asociados a las aristas de  $v_i$ , para cada arista si tiene solo a  $v_i$  en el recubrimiento por vértices, se recorre el subgrafo de esa arista de forma completa, si la arista contiene los dos extremos en el cubrimiento, se recorre solo la mitad de los vértices del subgrafo correspondientes a  $v_i$ , desde el último subgrafo se vuelve a  $a_{i+1}$ , repitiendo el proceso, excepto para  $a_k$  que vuelve a  $a_1$  y termina el circuito.

**CH → CV**

- Si se tiene un circuito hamiltoniano, desde cada  $a_i$  al siguiente  $a_{i+1}$  y desde  $a_K$  a  $a_1$ , se recorren los subgrafos, en cada uno de los recorridos se entra y salen de los subgrafos por un mismo vértice, sea este vértice  $v_i$  (o  $v_k$  si es desde  $v_k$  a  $v_1$ ), el conjunto  $\{v_1, \dots, v_K\}$  es un cubrimiento por vértices ya que cada arista tiene un subgrafo en el que, al menos, el circuito hamiltoniano entra una vez, el vértice por el que se entra, que tiene que coincidir con el de salida, ha de estar en el cubrimiento de vértices.

### Problema SUMA:

- Datos:** Un conjunto  $A$  y un tamaño para cada uno de sus elementos,  $s: A \rightarrow \mathbb{N}$  y un entero  $B$ .
- Pregunta:** Determinar si existe un  $A' \subseteq A$  tal que  $\sum_{a \in A'} s(a) = B$ .
- Esto es NP, se eligen de forma no determinista los elementos de  $A'$  y en tiempo polinómico se determina si la suma de los tamaños de los conjuntos es igual a  $B$ .

### Reducción ACTRI $\propto$ SUMA

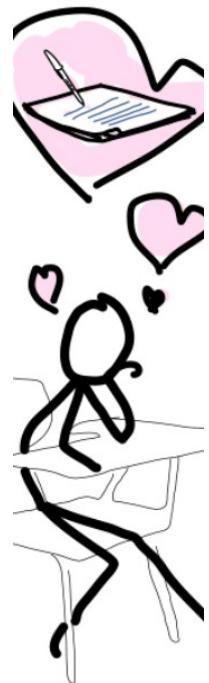
- Sea  $W, X, Y$  con  $|W| = |X| = |Y| = q$  y un subconjunto  $M \subseteq X \times Y \times Z$  un ejemplo del problema ACTRI.

$$\begin{aligned} W &= \{w_1, \dots, w_q\}, & X &= \{x_1, \dots, x_q\} \\ Y &= \{y_1, \dots, y_q\} \\ M &= \{m_1, m_2, \dots, m_k\} \end{aligned}$$

- El conjunto  $A$  va a tener  $k$  elementos  $A = \{a_1, \dots, a_k\}$ , cada elemento  $a_i \in A$  se corresponde con una tripleta  $m_i \in M$ .

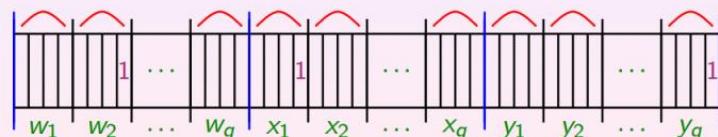
**QUIERES  
CONSEGUIR  
15€??**

TRÁENOS A TU  
CRUSH DE APUNTES  
ANTES DE QUE  
LOS QUEME



- Recordar: Si se suman  $k$  unos, la cifra del número resultante será como máximo  $p = \lceil \log_2(k) \rceil + 1$ .
- Para cada tripleta  $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)})$  se considera el elemento  $a_i$  con un peso...
  - $s(a_i) = 2^{p(3q-f(i))} + 2^{p(2q-g(i))} + 2^{p(q-h(i))}$

En binario podemos ver el número en grupos de  $p$  posiciones. Cada grupo corresponde a un elemento de  $W, X$  o  $Y$ :



Cada tripleta  $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)})$  se corresponde con un número con un **1** a la derecha de las zonas de  $w_{f(i)}, x_{g(i)}, y_{h(i)}$  y **0** en el resto.

- Si  $p = \lceil \log_2(k) \rceil + 1$ , entonces sumando  $k$  o menos unos nunca se obtendrá un número con un 1 más allá de la posición  $p + 1$ .
  - Sumando  $k$  o menos números asociados a las tripletas, nunca pasa un 1 de la zona de un elemento a la zona de otro elemento.
- Por lo tanto se puede fijar  $B = \sum_{j=0}^{3q-1} 2^{p \cdot j}$ , este número tiene un 1 a la derecha de cada zona.
- Como consecuencia, para cada  $A' \subseteq \{a_1, \dots, a_k\}$  se tiene que  $\sum_{a \in A'} s(a) = B$  si y solo si  $M' \in \{m_i : a_i \in A'\}$  es un recubrimiento por tripletas.

#### Problema PARTICIÓN:

- **Datos:** Un conjunto  $C$  y un tamaño para cada uno de sus elementos,  $s: C \rightarrow \mathbb{N}$ .
- **Pregunta:** Determinar si existe un  $C' \subseteq C$  tal que verifica  $\sum_{a \in C'} s(a) = \sum_{a \in C \setminus C'} s(a)$
- Es NP, se elige de forma no determinista los elementos de  $C'$  y en tiempo polinómico se determina si los tamaños totales de los conjuntos  $C'$  y  $C \setminus C'$  son iguales.

#### **Reducción SUMA $\propto$ PARTICION**

- Si se tiene un problema SUMA con  $A$ , tamaños  $s$  y entero  $B$ , entonces se crea un problema de PARTICION en el que  $C = A \cup \{b_1, b_2\}$ , dos valores nuevos.
- Los tamaños  $s$  de partición serán los mismos que en el problema SUMA, los de  $b_1$  y  $b_2$  serían:
  - $s(b_1) = \left( \sum_{i=1}^k s(a_i) \right)$
  - $s(b_2) = 2B$
- Cada uno de estos pesos necesita, máximo,  $(3qp)$  bits, se pueden calcular zona a zona de forma consecutiva.
- El conjunto  $C = \{a_1, \dots, a_k, b_1, b_2\}$ , la suma de los pesos de sus elementos son...



si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

$$\sum_{x \in A} s(x) = \sum_{i=1}^k s(a_i) + s(b_1) + s(b_2) =$$

$$\sum_{i=1}^k s(a_i) + \sum_{i=1}^k s(a_i) + 2B = 2 \sum_{i=1}^k s(a_i) + 2B$$

Este conjunto se parte en dos mitades cuando cada una pesa:

$$\sum_{i=1}^k s(a_i) + B$$

### Equivalencias

#### **PARTICION → SUMA**

- Si  $C' \subseteq C$  tal que  $\sum_{a \in C'} s(a) = \sum_{a \in C \setminus C'} s(a)$  entonces cada una de las sumas es  $\sum_{i=1}^k s(a_i) + B$ , uno de los conjuntos deberá tener  $b_1$  y no  $b_2$ . La suma de los pesos de los elementos de  $C'$  distintos de  $b_1$  tiene que ser  $B$ , por lo tanto, existe solución positiva para SUMA

#### **SUMA → PARTICION**

- Si SUMA tiene solución, con  $A'$  tal que  $\sum_{a \in A'} s(a_i) = B$ , entonces  $C' = \{b_1\} \cup A'$  ya que la suma de los pesos de  $C' = (\sum_{i=1}^k s(a_i) + B)$  que es la mitad del total.

### Técnicas de Reducción:

- Restricción:** Demostrando que un problema NP-Completo es un subproblema del que se está considerando.
- Reemplazamiento Local:** Haciendo una transformación de un problema NP-Completo elemento a elemento.
  - Reemplazamiento Local con Refuerzo:** Haciendo una transformación de un problema NP-Completo elemento a elemento, pero añadiendo algunos elementos adicionales para forzar la equivalencia de los problemas.
- Diseño de Componentes:** Distintos elementos del problema original se transforman en distintos tipos de estructura que se conectan con otros elementos.

#### **Restricción**

- Si un problema  $\Pi_1$  es NP y un subproblema,  $\Pi_2$  es NP-Completo, entonces  $\Pi_1$  es NP-Completo.
  - Con subproblema se entiende que se elige uno de los parámetros concretos de  $\Pi_2$  se obtiene  $\Pi_1$

Tenemos un conjunto finito de objetos  $U$ . Cada objeto,  $u$ , tiene un tamaño,  $s(u) \in \mathbb{N}$ , y un valor  $v(u) \in \mathbb{N}$ . Tenemos, además, dos números naturales:  $B$  (el tamaño máximo) y  $K$  (el valor mínimo). La pregunta es si existe un subconjunto de objetos  $U' \subseteq U$ , tal que

$$\sum_{u \in U'} s(u) \leq B, \quad \sum_{u \in U'} v(u) \geq K$$

El problema de la partición es un caso particular de este problema, en el que  $s(u) = v(u), \forall u$  y  $B = K = (1/2)\sum_{u \in U} s(u)$ .

**REDUCCIÓN** de PARTICIÓN( $C, s$ ) a MOCHILA( $U, s', v', B, K$ ):

$$U = C, s' = s, v' = s, B = K = (1/2)\sum_{u \in C} s(u)$$

**Datos:** Un conjunto  $A$  de tareas, cada tarea,  $a \in A$ , tiene una longitud  $l(a) \in \mathbb{N}$ . Tenemos, además, un número de procesadores  $m$  y un tiempo límite,  $D \in \mathbb{N}$ .

**Pregunta:** ¿Existe una partición de  $A: \{A_1, \dots, A_m\}$  en  $m$  subconjuntos disjuntos, de manera que

$$\max \left\{ \sum_{a \in A_i} l(a) : 1 \leq i \leq m \right\} \leq D$$

---

Si nos restringimos al caso  $m = 2$  y  $D = 1/2\sum_{a \in A} l(a)$  obtenemos el problema de la partición.

#### Reemplazamiento Local

- Cada elemento de un problema NP-Completo  $\Pi'$  se transforma en una estructura del problema que estamos considerando  $\Pi$ .
- Por ejemplo, reducir de SAT a 3-SAT.

#### Reemplazamiento Local con Refuerzo

- Corresponde al caso que, además de los elementos en que se transforman los elementos de  $\Pi'$ , se añaden algunos elementos adicionales para forzar la equivalencia de las soluciones.
- Por ejemplo, la reducción por cubrimiento de tripletas de al problema de la partición, ACTRI  $\propto$  PARTICIÓN.

#### Diseño de Componentes

- Distintas componentes del problema a reducir  $\Pi'$  se transforman en distintas estructuras de  $\Pi$  que se conectan de alguna forma para forzar la equivalencia.
- Por ejemplo, el cubrimiento de vértices, el circuito Hamiltoniano o la demostración del Teorema de Cook.

#### Reducibilidad Turing:

- Un problema  $\Pi$  se reduce Turing a  $\Pi'$  lo que se representa como  $\Pi \leq_T \Pi'$  si y solo si  $\Pi$  se puede resolver en tiempo polinómico mediante un algoritmo que puede llamar a una función que resuelve  $\Pi'$  contando cada llamada como un paso de cálculo.
- Este es el concepto más débil que el que se ha visto de reducibilidad logarítmica, si  $\Pi$  se reduce a  $\Pi'$  entonces se puede construir una reducibilidad Turing.

## Problemas NP-Difíciles:

- Un problema  $\Pi$  es NP-Difícil si y solo si existe un problema NP-Completo  $\Pi'$  que se puede reducir Turing a  $\Pi$ :  $\Pi' \leq_T \Pi$ .
- **Teorema:** Si un problema NP-Difícil se resuelve en tiempo polinómico, entonces,  $P = NP$ .
  - Se van a considerar problemas NP-Difíciles que tienen una dificultad similar a los NP-Completos, donde existe reducibilidad Turing entre ambos.
    - Clase Co-NP: Complementarios de los NP
    - Clase FNP: Problemas que buscan una solución, cuando saber si existe es NP.

## Clase Co-NP:

$$CoNP = \{\bar{L} : L \in NP\}$$

Por ejemplo, dado un grafo, determinar si NO tiene un camino hamiltoniano

- Estos problemas no están en NP, aunque poseen una MTND polinómica en que la respuesta es afirmativa al problema si todas las opciones responden Sí.
  - Si la respuesta es positiva, todas las opciones acaban en Sí.
  - Si la respuesta es negativa, al menos una opción acaba en NO.
- No se reducen a problemas NP con una transformación espacio logarítmica, ya que estas transformaciones exigen que las respuestas a ambos problemas sean las mismas.
  - Si existen reducciones Turing entre problemas NP y CoNP.

## **Problemas CoNP Completos**

- Un problema es CoNP-Completo si y solo si está en la clase CoNP y cualquier otro problema CoNP se reduce a él.

### Teorema

$$L \text{ es NP Completo} \Leftrightarrow \bar{L} \text{ es CoNP Completo}$$

### Teorema

$$\text{Si un problema CoNP completo está en NP, entonces } CoNP = NP.$$

### Teorema

$$\text{Si } P = NP, \text{ entonces } CoNP = NP$$

## **Caracterizaciones**

- **NP:** Un lenguaje  $L \subseteq A^*$  está en NP si y solo si existe una relación  $R$  en  $A^* \times A^*$  calculable en tiempo polinómico y un polinomio  $p(n)$  tal que  $L = \{x \in A^* : \exists y \in A^* \text{ con } |y| \leq p(|x|), R(x, y) = 1\}$ .
  - Se dice que los lenguajes o problemas NP se pueden verificar en tiempo polinómico de forma eficiente.
  - Al algoritmo que calcula  $R$  se le llama **verificador**, a  $y$  se le llama un **certificado**.
- **CoNP:** Un lenguaje  $L \subseteq A^*$  está en CoNP si y solo si existe una relación  $R$  en  $A^* \times A^*$  calculable en tiempo polinómico y un polinomio  $p(n)$  tal que  $L = \{x \in A^* : \forall y \in A^* \text{ con } |y| \leq p(|x|), R(x, y) = 1\}$

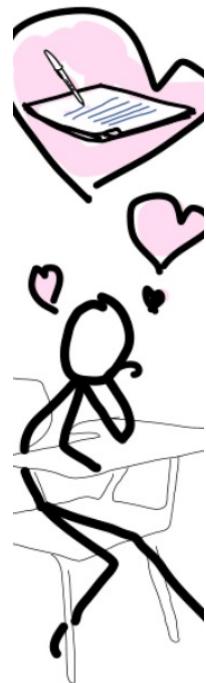
## **La primalidad está en NP**

- Un número  $p > 1$  es primo si y solo si existe un número  $1 < r < p$  tal que  $r^{p-1} = 1 \text{ mod}(p)$  y además  $r^{\frac{p-1}{q}} \neq 1 \text{ mod}(p)$  para todos los divisores primos de  $q$  en  $p - 1$ .
- **Teorema:** El teorema de Pratt reza que la primalidad está en NP.

**QUIERES  
CONSEGUIR  
15€??**

TRÁENOS A TU  
CRUSH DE APUNTES

ANTES DE QUE  
LOS QUEME



### Demostración

Si  $p$  es primo, esto se puede certificar con la existencia de un número  $r$  tal que  $r^{p-1} = 1 \pmod p$  y que además verificase  $r^{\frac{p-1}{q}} \neq 1 \pmod p$ , para todos los divisores primos  $q$  de  $p-1$ .

La comprobación de que  $r^{p-1} = 1 \pmod p$  se puede hacer en tiempo polinomial en función de la longitud de  $p$ , que es de orden  $I = \log(p)$ : Para calcular  $r^{p-1}$ , se calculan  $r^2, r^4, \dots, r^{2^I}$ . Esto tiene un orden de  $O(I^3)$ .  $r$  por sí solo no es un certificado:  $20^{21-1} = 1 \pmod{21}$  y  $21$  no es primo

El certificado tiene que adjuntar todos los divisores primos de  $p-1$ . Es decir constará, en principio de  $r$  y una lista de divisores de  $p-1$ :  $(q_1, \dots, q_k)$ . El comprobar que la lista es completa se hace por divisiones sucesivas de  $p-1$  entre estos números y comprobando que de uno al final.

Nos queda una cuestión, ¿cómo sabemos que los números  $(q_1, \dots, q_k)$  son primos? Pues dando certificados para ellos. Excepto para  $2$  que no necesita certificado.

El certificado sería una estructura recursiva: un certificado del número  $p$  sería  $r$  y una lista de divisores completa de  $p-1$ :  $(q_1, \dots, q_k)$ , y certificados de primalidad para cada uno de ellos.

### Clase FNP:



si  
consigues  
que suba  
apuntes, te  
llevas 15€ +  
5 Wuolah  
Coins para  
los sorteos

#### Caracterización de NP

Un lenguaje  $L \subseteq A^*$  está en NP si y solo si existe una relación  $R$  decidable en tiempo polinómico y un polinomio  $p$  tal que

$$L = \{x \in A^* : \exists y \in A^* \text{ con } |y| \leq p(|x|), R(x, y) = 1\}$$

#### Caracterización de FNP

Un problema de función está en FNP si y solo si está asociado a una relación  $R$  decidable en tiempo polinómico y tal que si  $R(x, y) = 1$ , entonces  $|y| \leq p(|x|)$  para un polinomio  $p$ .

### Clase FP:

- Es la clase problemas de funciones FNP tales que existe una MT que las calcula en tiempo polinómico

## FNPT

**FNPT:** La clase de los problemas de FNP totales.

Si para todo  $x$  existe un  $y$  con  $|y| \leq p(|x|)$  tal que  $R(x, y) = 1$

### Ejemplo

Dado un número  $p$  encontrar una descomposición en números primos

$$p = p_1^{k_1} \cdot p_2^{k_2} \cdots \cdot p_m^{k_m}$$

con un certificado de primalidad para cada número primo.

Seguro que existe, pero no es fácil encontrarlo.

- Los problemas de Funciones Totales están dentro de esta clase; un problema está en esta clase si para todo  $x$  existe un  $y$  tal que  $R(x, y)$ .
  - Es decir, el problema de decisión no es difícil: Siempre tiene una respuesta positiva, no quiere decir que el problema en sí sea fácil, ya que encontrar la solución no puede ser inmediato.
- Por ejemplo, la Red Feliz.

### Reducciones en FNP

#### Reducciones en FNP

Un problema de funciones  $\Pi$  se *reduce* a un problema  $\Pi'$  si y solo si, existen funciones  $R$  y  $S$  calculables en espacio logarítmico, tal que para toda cadena  $x$  y  $z$  ocurre lo siguiente: Si  $x$  es un ejemplo de  $\Pi$  entonces  $R(x)$  es un ejemplo de  $\Pi'$ . Además si  $z$  es una solución correcta de  $R(x)$  entonces  $S(z)$  es una solución correcta de  $x$ .

#### Problemas FNP-completos

Con esto podemos definir el concepto de completitud. Un problema es **FNP-completo** si y solo si está en **FNP** y cualquier otro problema de esta clase se puede reducir a este problema.

### Ejemplo

#### Un problema FNP-completo: FSAT

Dado un conjunto de cláusulas, encontrar, si existe, una asignación de valores de verdad para la que todas las cláusulas sean verdaderas.

- Teorema:** NP=P si y solo si FNP=FP
- Si se pudiera resolver el problema en versión decisión del TSP en tiempo polinómico, se podría resolver el problema del circuito óptimo en tiempo polinómico.
  - Se obtendría primero una cota  $R$  para el coste del circuito óptimo, el valor de la distancia más grande, multiplicado por el número de ciudades.
  - Después, realizando una búsqueda binaria en el intervalo  $[0, R]$  mediante sucesivas llamadas al problema de decisión se calcula el valor del circuito óptimo  $K$
  - Luego, para cada par de ciudades, si el coste de ir de una a otra es  $c$ , se llama al problema de decisión con presupuesto  $K$  e incrementando el coste de  $c$  a  $c + 1$ .
    - Si la respuesta es positiva, no es necesario usar este arco en el óptimo, se deja el coste a  $c + 1$  y continúa.
    - Si la respuesta es negativa, este arco sí se usa, se deja el mismo coste y se continúa.
  - Al final se obtendrán todos los arcos de un circuito óptimo,