

# Esquemas.pdf



Nessy



Modelos Avanzados de Computacion



3º Grado en Ingeniería Informática



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

De la imprenta a la nube, de las descargas a Netflix.

Fuera los temarios infinitos, bienvenido Wuolah.

¿Y para tu futuro? ¿A quién le vas a confiar tu formación?

**The Globe, cursos del siglo XXI**

Descúbrelo  
en estos QR



# Aprende Inglés

## Con nuestros cursos **GRATUITOS** para desempleados



Fórmate con nuestros cursos de Inglés para las titulaciones A1 B1 y B2 100€ subvencionados para desempleados

Pincha aquí e  
inscríbete ya

958 047 283  
621 21 76 50



MINISTERIO  
DE TRABAJO  
Y ECONOMÍA SOCIAL

SERVICIO PÚBLICO  
DE EMPLEO ESTATAL  
**SEPE**



ANDALUCÍA EMPRENDE,  
FUNDACIÓN PÚBLICA ANDALUZA  
Consejería de Empleo,  
Formación y Trabajo Autónomo



## Tema 1 – Máquinas de Turing. Funciones y Lenguajes Calculables

- Problema de la Parada:
  - o Entrada: programa + datos de entrada.
  - o Salida:
    - SI: el programa termina para esos datos.
    - NO: el programa cicla de forma indefinida para esos datos.
- Máquina de Turing (MT):
  - o  $(Q, A, B, \delta, q_0, \#, F)$ 
    - $Q$ : conjunto finito de estados.
    - $A$ : alfabeto de entrada.
    - $B$ : alfabeto de trabajo (incluye  $A$ ).
    - $\delta$ : transición de un estado a otro.
    - $q_0$ : estado inicial
    - $\#$ : símbolo blanco.
    - $F$ : estados finales.
- Configuración:  $(q, w_1, w_2)$ 
  - o  $q$ : estado en la que se encuentra la MT.
  - o  $w_1$ : palabra a la izquierda del cabezal de lectura. Puede ser vacío.
  - o  $w_2$ : palabra a la derecha desde el cabezal de lectura. No puede ser vacío.
- Paso de cálculo:  $(q, w_1, w_2) \vdash (p, v_1, v_2)$ 
  - o Movimiento a la izquierda:
    - Si  $w_1 = \epsilon \Rightarrow v_1 = \epsilon, v_2 = \#w_2$
    - Si  $|w_1| = 1, w_1 = \# \Rightarrow v_1 = w_1 - a, v_2 = a$
  - o Movimiento a la derecha:
    - Si  $|w_2| = 1 \Rightarrow v_1 = w_1 w_2, v_2 = \#$
    - Si  $w_2[0] = \# \Rightarrow v_1 = \epsilon, v_2 = w_2 - w_2[0]$
- Sucesión de pasos de cálculo: llegar de  $R$  a  $R'$ .
  - o  $R \vdash^* R'$
  - o Sucesión finita de configuraciones en la que  $R$  es la primera configuración y  $R'$  la última.
- Lenguaje Recursivamente Enumerable (r.e.):
  - o Un lenguaje  $L$  es r.e. si existe una MT que acepte ese lenguaje (la MT llega a un estado final).
- Lenguaje Recursivo:
  - o Un lenguaje  $L$  es recursivo si es aceptado por una MT que siempre termina (rechace o acepte).
  - o Es r.e.
  - o Su problema de aceptación puede ser resuelto por un algoritmo.
- MT Calculadoras:
  - o  $f: D \rightarrow B^*$
  - o  $f(u) =$  contenido de la cinta cuando la MT termina.
  - o Funciones Parcialmente Calculables := Lenguaje r.e.
  - o Funciones Calculables := Lenguaje Enumerable
- Características de las MT:
  - o Memoria:
    - $\delta([q_i, a], b)$ : la MT está en el estado  $q_i$  recordando  $a$  y lee  $b$ .
  - o Pistas múltiples:
    - $\delta(q_i, [a, b])$ : la MT está en el estado  $q_i$  y lee en la primera pista de la cinta  $a$  y  $b$  en la segunda.
  - o Subrutinas:
    - Conjunto de estados que realiza una acción concreta.
    - El estado final representa el estado de retorno de la subrutina.
    - Para recordar la posición en la que estaba la MT antes de la subrutina se puede usar una pista adicional y un símbolo extra.

**Llévate** Un patinete, unos auriculares o una tablet voom tab pro+teclado. Todos los estudiantes que presenten unos apuntes de **Wuolah** en tienda se les aplicará un 10% de descuento en la compra de cualquiera de nuestros productos.

**Llévate**

G R A N A D A  
Calle Puentezuelas, 49



- Extensiones de las MT:
  - o Movimientos estáticos:
    - El cabezal se queda en la misma posición en la que estaba.
  - o Multicintas:
    - Distintas cintas en la que se puede leer y escribir.
    - Cada cinta cuenta con un cabezal propio.
    - Número de cintas limitadas.
    - Lenguaje aceptado:
      - La MT termina en aceptación.
    - Función calculada:
      - $f(u)$  como contenido de la última cinta.
    - Si una MT multicinta requiere  $t(n)$  pasos para una entrada de tamaño  $n$ , para una MT con una sola cinta necesita  $O(t^2(n))$ .
  - o Máquinas No Deterministas:
    - Puede realizar varias transiciones en una configuración.
    - Lenguaje aceptado:
      - Existe una sucesión de movimientos que acaba en estado de aceptación.
    - Recursivamente Enumerable:
      - Todo lenguaje aceptado por la MTND.
    - El número de pasos que requiere una MTND es el número de pasos para el cálculo más largo posible para esa entrada.
      - Si no termina, es infinito.
    - Si tiene complejidad  $t(n)$  en tiempo significa que todos los posibles cálculos de la MTND terminan en  $t(n)$  o menos pasos.
    - Si la MTND tiene complejidad  $t(n)$ , la MT que la simula tiene complejidad  $O(d^{t(n)})$  ( $d > 1$ ).
    - Ventajas:
      - Útiles para resolver problemas de búsqueda.
      - Se elige un elemento de manera no-determinista y se comprueba si cumple la condición.
        - Si la cumple, acepta.
        - Si no la cumple, rechaza.
      - No hay que describir el proceso de búsqueda.
  - Limitaciones de las MT:
    - o Cintas semiilimitadas:
      - La cinta de la MT es ilimitada solo por la derecha.
      - Todo lenguaje aceptado por una MT ( $MT_2$ ) ilimitada también es aceptado por una MT ( $MT_1$ ) semilimitada si:
        - $MT_1$  nunca escribe el símbolo blanco (#).
        - La cabeza de  $M_1$  nunca se mueve a la izquierda de su posición inicial.
      - Se tiene el símbolo  $\triangleright$  a la izquierda de cada cinta que se use (indica el tope).
  - **Problema ( $\Pi$ ):**
    - o  $X$ : conjunto de entradas.  $x$  es una entrada del conjunto.
    - o  $Y$ : conjunto de soluciones.  $y$  es una solución del conjunto.
    - o  $R: X \times Y \rightarrow \{0,1\}$ : relación entre las entradas y las salidas.
      - $R(x,y) = 1 \Rightarrow R(x,y)$
      - $R(x,y) = 0 \Rightarrow \neg R(x,y)$
    - o Se resuelven con algoritmos.
      - Programa en Python (Ph).
      - Máquina de Turing (Mt).
    - o Un algoritmo ALG resuelve un problema  $\Pi$  si  $R(x,y) = 1$ .

- Se supone:
  - Las entradas están codificadas como palabras.
  - Hay una única palabra de entrada y salida.
- **Problema Computacional:**
  - Problemas en los que las entradas y las salidas son palabras de un alfabeto.
  - Cualquier problema se transforma en un problema computacional con un sistema de codificación de las entradas y salidas.
  - Si no es una entrada correcta, la salida es NO.
- Tipos de problemas:
  - **Problemas de Búsqueda:**
    - Encontrar una solución a una entrada.
    - Si no existe se devuelve NO.
  - **Problemas de Decisión:**
    - Existencia.
    - Las soluciones son SI y NO.
    - Cada entrada tiene una única solución.
    - Son los más simples.
    - Todo problema tiene un problema de decisión asociado.
  - **Problemas de Optimización:**
    - Optimizar una función definida sobre un conjunto de soluciones factibles asociadas a la entrada.
  - **Problemas de función:**
    - Cada entrada tiene una única solución.
- Versión de Problema de Decisión de los problemas:
  - Problema de existencia para problemas de búsqueda.
    - Dada una entrada, determinar si tiene una solución asociada.
  - Problema de umbral para problemas de optimización:
    - Se añade un umbral al problema.
    - Determinar si existe una solución de valor mayor/menor que el umbral (depende de si se quiere minimizar o maximizar).
  - Problema de comprobación para problemas de función y búsqueda:
    - Dada una entrada y una posible solución, determinar si es solución.
- **Problemas de Decisión y Lenguajes:**
  - Las entradas se codifican como palabras del alfabeto  $A$  con el lenguaje de las entradas que tienen respuesta SI.
  - Problemas cuyos lenguajes son recursivos: **decidibles** o **calculables**.
  - Problemas cuyos lenguajes no son recursivos: **indecidibles** o **no calculables**.
- **Problema contrario ( $\bar{\Pi}$ ):**
  - Intercambia las salidas SI y No del problema  $\Pi$ .
  - El lenguaje asociado al problema contrario de  $\Pi$  es el lenguaje complementario del lenguaje asociado a  $\Pi$ .
    - $L(\bar{\Pi}) = \overline{L(\Pi)}$
    - Si  $L$  es recursivo,  $\bar{L}$  también lo es.
    - Si  $L$  y  $\bar{L}$  son r.e.,  $L$  es recursivo.
- Correspondencia Biyectiva:
  - Entre palabras del alfabeto y números naturales.
  - El número de la palabra  $w$  se denota como  $Z(w)$ .
    - $Z(0110) = 1 \times 2^3 + 2 \times 2^2 + 2 \times 2^1 + 1 \times 2^0$
    - $Z(0) = 1 ; Z(1) = 2$
  - Se puede establecer una aplicación biyectiva entre dos alfabetos:
    - Se codifica el alfabeto  $A$  como números naturales.
    - Se codifica el código numérico para el alfabeto  $B$ .

- Se puede establecer un orden total en las palabras del alfabeto con el código numérico de cada palabra.
- **Lenguaje de diagonalización ( $L_d$ ):**
  - **No es r.e.**
  - $w \in L_d$  si la MT con codificación  $w$  **no acepta**  $w$ .
  - $\Pi_d$ : **Problema de Diagonalización**
    - Determinar si, dada una MT, no se acepta a si misma como entrada.
- **Lenguaje Universal ( $L_u$ ):**
  - **No es recursivo**
  - **Es r.e.**
  - Conjunto de todas las cadenas que codifican una MT y su entrada, tales que la MT acepta la entrada.
  - $\Pi_u$ : **Problema Universal**
    - Determinar si, dada una MT  $M$  y una entrada  $w$ ,  $M$  acepta  $w$ .
- **Lenguaje vacío ( $L_e$ ):**
  - Conjunto de palabras  $w$  tales que la MT  $M$  cuya codificación es  $w$  no acepta ninguna palabra.
  - $L(M) = \emptyset$
- **Lenguaje no vacío ( $L_{ne}$ ):**
  - **Es r.e.**
  - **No es recursivo.**
  - Conjunto de palabras  $w$  tales que la MT  $M$  cuya codificación es  $w$  acepta alguna palabra.
  - $L(M) \neq \emptyset$
- **Reducción (problemas):**
  - Si  $P_1$  y  $P_2$  son problemas de decisión,  $P_1$  se reduce a  $P_2$  si existe una MT que siempre para y calcula una función  $f$  tal que para toda entrada  $w$  a  $P_1$ ,  $P_2$  produce la misma respuesta para la entrada  $f(w)$ .
  - Si  $P_1$  se reduce a  $P_2$ :
    - Si  $P_1$  es indecidible,  $P_2$  también lo es.
    - Si  $P_1$  no es semidecidible,  $P_2$  tampoco lo es.
- **Reducción (lenguajes):**
  - Si  $L_1 \subseteq A^*$  y  $L_2 \subseteq B^*$  son lenguajes,  $L_1$  se reduce a  $L_2$  si existe una MT que siempre para y calcula una función  $f: A^* \rightarrow B^*$  tal que para toda entrada  $w \in A^*$ ,  $w \in L_1 \Leftrightarrow f(w) \in L_2$ .
  - Si  $L_1$  se reduce a  $L_2$ :
    - Si  $L_1$  no es recursivo,  $L_2$  tampoco lo es.
    - Si  $L_1$  no es r.e.,  $L_2$  no lo es.
- **Propiedades de los r.e.:**
  - Saber si el lenguaje de una MT verifica esa propiedad.
  - **Propiedad trivial:**
    - Para toda MT, su lenguaje aceptado no verifica la propiedad.
    - Para toda MT, su lenguaje aceptado verifica siempre la propiedad.
- **Teorema de Rice:**
  - Toda propiedad trivial sobre los lenguajes r.e. es indecidible.
- **Problema de las Correspondencias de Post:**
  - **Es indecidible.**
  - Se tiene dos listas de la misma longitud sobre un alfabeto de referencia  $A$ .
  - Determinar si existe una secuencia no vacía de enteros tales que se consigan las mismas palabras en las dos listas.
  - **Problema de la Correspondencia de Post Modificado:**
    - Es indecidible si  $A$  tiene al menos 2 símbolos.
- **Reducción (ambigüedad):**
  - Saber si una gramática es independiente del contexto es ambigua es **indecidible**.



## Tema 2 – Otros modelos de Cálculo: Tesis de Church-Turing

### - Programas de Post Turing:

- Lenguaje de programación para manipulación de cadenas.
- Programas que actúan sobre una cinta en la que se pueden escribir o leer símbolos (ilimitada por ambos lados).
- Siempre hay una casilla de la cinta que está activa en un momento dado.
  - Solo se puede escribir y leer esta casilla.
- En un paso de cálculo:
  - Leer la casilla.
  - Ejecutar una instrucción.
  - Escribir un símbolo.
  - Moverse a derecha/izquierda.
- Conjunto de instrucciones sobre un alfabeto de entrada  $A$  y un alfabeto de trabajo  $B$  (incluye el símbolo blanco).
- Una instrucción puede llevar una etiqueta ([Label]).
- Instrucciones:
  - **PRINT a:** escribir  $a$  en la casilla del cabezal.
  - **IF a GOTO L:** si el símbolo de la casilla es  $a$ , entonces la siguiente instrucción es la de la etiqueta  $L$ .
  - **RIGHT:** mover el cabezal de lectura a la derecha.
  - **LEFT:** mover el cabezal de lectura a la izquierda.
  - **HALT:** termina y acepta.
- Se comienza con el cabezal de lectura a la izquierda de la palabra.
- Las instrucciones se ejecutan en orden.
  - Si se encuentra con un IF a GOTO L, continua en la instrucción etiquetada con  $L$ .
- Termina si no quedan más instrucciones o si llega a HALT.
- **Lenguaje Aceptado:**
  - Conjunto de cadenas del alfabeto de entrada que, comenzando en la configuración inicial, llegan a una instrucción HALT.
- **Función Calculada:**
  - Una función es parcialmente calculable por un programa Post Turing si, para cualquier palabra del alfabeto de entrada, el programa:
    - Llega a HALT con  $f(u)$  en la cinta, si la palabra pertenece a la entrada.
    - Cicla si no pertenece a la entrada.

### - Programas con variables:

- Elementos:
  - $X$ : variable de entrada.
  - $Z$ : conjunto de variables de trabajo.
  - $Y$ : variable de salida.
- 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 está vacía).
  - **IF A ENDS a GOTO L:** si el último símbolo de  $A$  es una  $a$ , la siguiente instrucción es la de la etiqueta  $L$ .
  - **HALT:** termina y acepta.
- Al inicio, la variable de entrada es la única variable no vacía.
- Se acepta una palabra si llega a HALT con  $f(u)$  almacenado en la variable  $Y$ , cuando  $f$  está definida.
  - En otro caso, cicla.
- **Macros:**
  - Conjunto de instrucciones que se usan con frecuencia.
  - **IF  $V \neq \epsilon$  GOTO L:** si la variable  $V$  no está vacía, seguir por la instrucción con la etiqueta  $L$ .

**Llévate**

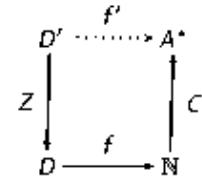
Un patinete, unos auriculares o una tablet voom tab pro+teclado.  
Todos los estudiantes que presenten unos apuntes de **Wuolah** en tienda se les aplicará un 10% de descuento en la compra de cualquiera de nuestros productos.



G R A N A D A  
Calle Puentezuelas, 49

**WUOLAH**

- **GOTO L:** continuar la ejecución del programa por la instrucción con la etiqueta L.
- **IF V ENDS  $\alpha_i$  GOTO  $L_i$  ( $i = 1, \dots, n$ ):** ir a la instrucción determinada en función del símbolo que se lea.
- **$U \leftarrow V$ :** copiar el contenido de la variable V en U.
- Entradas múltiples:
  - Se quiere calcular una función con varias variables.
  - Juntarlas en una sola variable y con un símbolo separador  $c$ .
- Equivalencia de modelos:
  - Si  $f$  es parcialmente calculable por una MT, lo es también por un programa Post-Turing y por un programa con variables.
  - Si  $L$  es aceptado por una MT, lo es también por un programa Post-Turing y por un programa con variables.
- **Cálculo con números enteros**
  - $f: D \rightarrow \mathbb{N}, D \subseteq \mathbb{N}$ 
    - **$f$  es parcialmente calculable** si existe un alfabeto  $A$  donde  $f' = C \circ f \circ Z$  es calculable.
    - $D' = C(D)$
  - Si una función numérica es calculable sobre un alfabeto  $A$ , entonces lo es sobre cualquier alfabeto  $B$ .
  - Una función es **recursiva o calculable** cuando es parcialmente calculable y total (está definida en los números naturales).
  - **Problema con Variables Numéricas:**
    - Elementos:
      - Conjunto de variables de entrada ( $X_1, \dots, X_k$ ).
      - Conjunto finito de variables de trabajo ( $Z_1, \dots, Z_l$ ).
      - Variable de salida ( $Y$ ).
    - Instrucciones:
      - **$A \leftarrow A+1$ :** suma uno al valor entero de la variable A.
      - **$A \leftarrow A-1$ :** resta uno al valor entero de la variable A (si es 0, no).
      - **IF  $A \neq 0$  GOTO L:** si el valor de A no es 0, seguir por la instrucción con la etiqueta L.
      - **HALT:** termina y acepta.
    - Todas las variables que no sean las de entrada empiezan a 0.
    - Se acepta si llega a HALT. La variable  $Y$  contiene  $f(n_1, \dots, n_k)$  cuando  $f$  está definida.
      - En otro caso, cicla.
    - **Macros:**
      - **$V \leftarrow 0$ :** poner el valor de V a 0.
      - **GOTO L:** seguir por la instrucción con la etiqueta L.
      - **$V \leftarrow U$ :** copiar en V el valor de U.
    - **Función Parcialmente Calculable por un Programa con Números:**
      - $f$  es parcialmente calculable por un programa con números si existe un programa con variables numéricas que dada  $n$  como entrada calcula  $f(n)$  como salida si  $n$  pertenece al conjunto de palabras aceptadas.
        - En caso contrario, cicla.
      - Una función  $f: A \rightarrow \mathbb{N}$  es parcialmente calculable por un programa con número si y solo si es parcialmente calculable por una MT.
- **Tesis de Church Turing:**
  - Toda función calculable mediante un proceso mecánico bien definido (efectivamente calculable) puede ser calculada por una Máquina de Turing.



## Tema 3 – Clases de Complejidad

- Complejidad:
  - o **La complejidad se mide en función de la longitud de la entrada.**
  - o Algoritmo de Primalidad (número primo):
    - Complejidad exponencial.
      - $O(x)$ : número de divisiones.
      - $\log(x)$ : número de casillas que ocupa en binario un número  $x$ .
      - $O(n^2)$ : orden para la división de un número entre otro, ambos de longitud  $n$ .
      - $O(2^n)$ : número de divisiones. El número es  $n = \log(x)$ , por lo que  $x$  es de orden  $2^n$ .
        - $a^b = c \Rightarrow \log_a c = b$
      - $O(n^2 2^n)$ : complejidad exponencial. Multiplicar una división por el número de divisiones.
  - Problema del Flujo Máximo:
    - o Grafo dirigido en el que los arcos están etiquetados por su capacidad. Hay un origen ( $s$ ) y un destino ( $t$ ).
    - o Se quiere asignar un valor a cada arco de manera que:
      - No supere la capacidad del arco.
      - La suma de valores que entran a un nodo es la misma que la que sale.
      - Se consigue el flujo máximo.
    - o Algoritmo de Ford-Fulkerson:
      - Inicio:  $(V, E, s, t, c)$ 
        - $V$ : conjunto de vértices.
        - $E$ : conjunto de aristas.
        - $s$ : nodo inicial.
        - $t$ : nodo final
        - $c$ : función que asigna a cada pareja de nodos  $(x, y)$  su capacidad  $c(x, y)$ .
      - 1. Se empieza con un flujo cualquiera, como 0.
      - 2. Se calcula el grafo diferencia entre el grafo original y el flujo.
        - Por ejemplo, si se tiene una capacidad de 4 y se está pasando 2, la diferencia sería 2.
        - Se añade un arco en sentido contrario indicando el flujo que lleva.
        - Si el flujo es 0, se quita el arco.
      - 3. Se lanza un algoritmo que busque un camino entre el nodo origen y el nodo destino.
        - Se calcula el mínimo de todos los valores asignados de ese camino y se escribe ese valor en todos los arcos del camino.
        - Se suma el flujo origina.
        - Por ejemplo, si se tiene 4 en una dirección y se suma 3 en la dirección contraria, se queda 1 en la dirección de 4.
      - 4. Se vuelve al paso 2.
      - 5. Cuando ya no quedan más caminos, se ha encontrado el flujo óptimo.
    - o  $O(n)$ : complejidad del algoritmo.
      - La complejidad se corresponde con una iteración.
      - Si se tiene  $n$  nodos, una iteración tendría una complejidad de  $n^2$  (por cada nodo se tienen 2 aristas).
    - o Elección del camino mínimo:
      - Escoger el camino mínimo entre el origen y el destino.
      - El número de iteraciones está limitado por  $n^3$  ( $n$  = número de nodos).
      - La complejidad total es del orden  $O(n^5)$ .
    - o Reducciones:

- Coloreo de un grafo.
- Problema de las parejas.
  - Se tiene dos conjuntos de vértices del mismo tamaño y un conjunto de aristas que representa las relaciones entre los vértices de los dos conjuntos.
  - Se quiere saber si existe un subconjunto de aristas tal que cada elemento de un conjunto se relacione con un único elemento del otro conjunto.
  - La reducción es:
    - Añadir un nuevo nodo inicial y conectarlo con todos los nodos del conjunto 1.
    - Mantener las conexiones entre los nodos del primer conjunto y el segundo conjunto.
    - Añadir un nodo final y conectarlo con todos los nodos del grupo 2.
    - Asignar una capacidad de 1 a todos los arcos.
    - ¿Existe un flujo máximo de tamaño  $m$ ?
      - $m = \text{número de elementos en cada conjunto.}$
- **Clases de Complejidad:**
  - Complejidad de una MT:
    - Es de complejidad  $f(n)$  si para toda entrada  $x$  de longitud  $|x| = n$ , la MT la acepta o rechaza consumiendo menos de  $f(n)$  unidades.
      - Unidades **pasos de cálculo: complejidad en tiempo.**
      - Unidades **casillas de la cinta: complejidad de la cinta.**
  - Complejidad de un Lenguaje o Problema de Decisión:
    - Es de complejidad  $f(n)$  si existe una MT que acepta el lenguaje y tiene complejidad  $f(n)$ .
  - Órdenes de Complejidad:
    - Una medida  $g(n)$  es de orden  $O(f(n))$  si existe  $n_0$  y  $c > 0$  tal que  $\forall n \geq n_0, g(n) \leq c \cdot f(n)$ .
    - Si  $L$  es aceptado en tiempo  $t(n)$  por una MT con  $k$  cintas, entonces  $\forall n \geq 0$  existe una MT con  $k + 1$  cintas que acepta el mismo lenguaje en tiempo  $\frac{1}{2^m} t(n) + n$ .
  - Complejidad de problemas de grafos:
    - No importa la representación que se utilice para saber la complejidad.
    - Una función es de orden polinómico como función de  $n$  si lo es en función de  $v$ .
      - Longitud de entrada:  $v \leq n \leq v^3$ .
      - Si es de orden  $O(n^k)$ , será a lo más  $O(v^{3k})$ .
      - Si es de orden  $O(v^k)$ , será a lo más  $O(n^k)$ .
  - Medidas de **Complejidad en Espacio**:
    - Reglas:
      - Contar las casillas que se usan (escribir o pasar sobre ellas).
      - Si nunca se escribe sobre la cinta de entrada, 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.
    - Los algoritmos de búsqueda en profundidad o en anchura son del orden  $O(n^2)$  en tiempo y  $O(n)$  en espacio.
      - $n = \text{número de nodos.}$
  - **Espacio Determinista:**
    - Teorema de Savitch:
      - La complejidad en espacio de existencia de caminos en grafos dirigidos es del orden  $O(\log^2(n))$ .
        - $n = \text{número de todos del grafo.}$
  - **Clases No-Deterministas:**

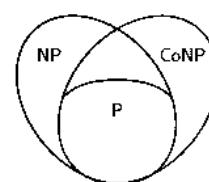


**Llévate** Un **patinete**, unos **auriculares** o una **tablet** voom tab pro+teclado.  
Todos los estudiantes que presenten unos apuntes de **Wuolah** en tienda se les aplicará un **10% de descuento** en la compra de cualquiera de nuestros productos.



G R A N A D A  
Calle Puentezuelas, 49

- Una MTND tiene una complejidad  $f(n)$  en tiempo (espacio) si para una entrada  $x$  le longitud  $n$ , todas las posibles opciones terminan en  $f(n)$  pasos (terminan y no usan más de  $f(n)$  casillas).
- Búsqueda de Caminos en Grafos:
  - Espacio  $O(\log(n))$ : espacio ND.
    - $n$  = número de vértices.
- Clases de Complejidad:
  - **TIEMPO(f):**
    - Todos los lenguajes aceptados por una MTD en tiempo  $O(f(n))$ .
  - **ESPACIO(f):**
    - Todos los lenguajes aceptados por una MTD 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  $O(f(n))$ .
  - Clase **polinómica:**
    - Determinista:
      - Tiempo: **P**
      - Espacio: **PESPACIO**
    - No Determinística:
      - Tiempo: **NP**
      - Espacio: **NPESPACIO**
  - Clase **espacio logarítmico:**
    - Determinista: **L**
    - No determinística: **NL**
  - Clase **exponencial en tiempo: EXP**
- Dependencias 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 MT, con una sobrecarga polinómica en el número de pasos.
      - Si se dan  $f(n)$  pasos, la MT puede dar  $f^k(n)$  pasos.
- Simulación de Máquinas ND:
  - Si  $L$  es decidido por una MTND en tiempo  $f(n) \geq n$ , entonces es decidido por una MT determinística con 3 cintas en tiempo  $O(d^{f(n)})$ 
    - $d > 1$ . Constante que depende de la MND inicial.
- **Complementarios de Clases:**
  - **CoC:** clase de lenguajes complementarios de los lenguajes en la clase **C**.
    - $L \in C \Leftrightarrow \bar{L} \in CoC$
  - El complementario de una clase determinista coincide con la propia clase.
    - **CoP = P**.
    - No ocurre con las clases ND.
  - La clase **CoNP** es el conjunto de problemas cuyo complementario está en **NP**.
- Relaciones Binarias:
  - Relación en  $A^* \times A^*$ :
    - Una relación binaria  $R$  es un subconjunto  $R \subseteq A^* \times A^*$ .
      - $R: A^* \times A^* \rightarrow \{0,1\}$  (función característica) donde  $R(x,y) = 1$  cuando  $(x,y) \in R$ .
        - $(x,y) \in R \cong xRy$
  - Relación calculable polinómicamente:



- $R$  en  $A^* \times A^*$  se dice calculable polinómicamente si existe una MT que calcula la función característica de  $R$  en tiempo polinómico (o acepta  $R$  en tiempo polinómico)
- **NP:**
  - Un lenguaje  $L \subseteq A^*$  está en NP si existe una relación  $R$  en  $A^* \times A^*$  calculable en tiempo polinómico y un polinomio  $p(n)$  tal que  

$$L = \{x \in A^* : \exists y \in A^* \text{ con } |y| \leq p(|x|), R(x, y) = 1\}.$$
  - Los lenguajes/problemas de NP son los problemas que se pueden **verificar** en tiempo polinómico (de forma eficiente).
  - **Verificador:** algoritmo que calcula  $R$ .
  - **Certificado:**  $y$ .
- **CoNP:**
  - Un lenguaje  $L \subseteq A^*$  está en CoNP si existe una relación  $R$  en  $A^* \times A^*$  calculable en tiempo polinómico y un polinomio  $p(n)$  tal que  

$$L = \{x \in A^* : \forall y \in A^* \text{ con } |y| \leq p(|x|), R(x, y) = 1\}.$$
- **Problemas NP:**
  - Dados unos datos  $x$  comprobar si existe un objeto  $y$  (con tamaño limitado a un polinomio de tamaño de  $x$ ) tal que se cumple una condición  $R(x, y) = 1$  que es verificable en tiempo polinómico.
- **Sistemas Interactivos de Demostración:**
  - Condiciones:
    - **Demostrador:** capacidad ilimitada de cálculo (no siempre honesto).
    - **Verificador:** capacidad polinómica de cálculo (honesto).
    - Se pueden intercambiar mensajes de un tamaño polinómico en función de una palabra inicial  $x$ .
  - Clase NP:
    - Clase de problemas que el verificador puede decidir, teniendo en cuenta que la respuesta es positiva, el demostrador tratará de convencer al verificador de lo mismo.
- **Relaciones entre Clases de Complejidad:**
  - $\text{ESPACIO}(f(n)) \subseteq \text{NESPACIO}(f(n))$
  - $\text{TIEMPO}(f(n)) \subseteq \text{NTIEMPO}(f(n))$
  - $\text{NTIEMPO}(f(n)) \subseteq \text{ESPACIO}(f(n))$
  - $\text{NESPACIO}(f(n)) \subseteq \text{TIEMPO}(k^{f(n)})$ 
    - Para un  $k > 1$ .
    - El Método de la Alcanzabilidad:
      - Construir un grafo en el que los nodos son las posibles configuraciones de una MT, y los arcos conectan configuraciones tales que se puede llegar de una a otra en un paso de cálculo.
        - Se supone MT con una sola cinta.
        - Una configuración es 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  $f(n)$ .
        - El número de configuraciones del orden  $|Q||B|^{2f(n)} \simeq t^{f(n)}$  con  $t = |B|^2$ .
          - $B$ : alfabeto de trabajo
        - Saber si una palabra es aceptada es equivalente a saber si existe un camino desde el estado inicial a una configuración que contenga un estado final.
          - Se puede comprobar en tiempo  $O(m^2)$ 
            - $m$  = número de nodos.
- **Teorema de la Jerarquía:**

- Funciones de Complejidad Propias:
  - Verifican:
    - $f(n+1) \geq f(n)$
    - $g(u) = f(|u|)$  es calculable en espacio y tiempo  $f(n)$ .
- Teorema de la Jerarquía en Tiempo:
  - Si  $f(n) \geq n$  es propia:
    - $\text{TIEMPO}(f(n)) \subset \text{TIEMPO}(f^2(n))$
    - $\text{TIEMPO}(f(n)) \neq \text{TIEMPO}(f^2(n))$
  - Corolario:  $P \neq EXP$
- Jerarquía en Espacio:
  - Si  $f(n)$  es una función de complejidad propia:
    - $\text{ESPACIO}(f(n)) \subset \text{ESPACIO}(f(n) \log f(n))$
    - $\text{ESPACIO}(f(n)) \neq \text{ESPACIO}(f(n) \log f(n))$
- Inclusión de Clases:
  - $L \subseteq NL \subseteq P \subseteq NP \subseteq PESPACE$ 
    - No se sabe si alguna inclusión es igualdad.
    - $NL$  está incluida estrictamente en  $PESPACE$ .
- Espacio ND:
  - Si  $f(n) \geq \log(n)$  y es propia
    - $\text{NESPACIO}(f(n)) \subseteq \text{ESPACIO}(f^2(n))$
    - $PESPACE = NPSPACE$
  - Si  $f(n) \geq \log(n)$  es propia
    - $\text{NESPACIO}(f(n)) = \text{CoNESPACIO}(f(n))$

## Tema 4 – NP-completitud

### - Reducción:

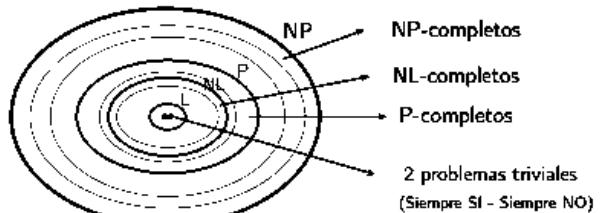
- $L_1 \leq L_2$ 
  - $L_1$  es **reducible** a  $L_2$  si existe una MT determinista que en **espacio logarítmico** calcula una función  $R: A^* \rightarrow B^*$  de tal manera que  $x \in L_1 \Leftrightarrow R(x) \in L_2$ .
  - Problema  $L_1$  reducible a  $L_2$ : Algoritmo  $L_2 \rightarrow$  Algoritmo  $L_1$
- Reducción en términos de problemas:
  - $\Pi_1 \leq \Pi_2$ 
    - $\Pi_1$  es reducible a  $\Pi_2$  si existe un algoritmo (MTD) que en espacio logarítmico calcula una función  $f: D_{\Pi_1} \rightarrow D_{\Pi_2}$  de manera que  $x \in Y_{\Pi_1} \Leftrightarrow f(x) \in Y_{\Pi_2}$ .
    - Problema  $\Pi_1$  reducible a  $\Pi_2$ : Algoritmo  $\Pi_2 \rightarrow$  Algoritmo  $\Pi_1$
  - La reducción entre los problemas equivaldrá a la reducción entre los lenguajes asociados.
  - $\Pi_2$  es **al menos tan difícil** como  $\Pi_1$
- Las reducciones deben poder hacerse en **Espacio Logarítmico**.
- Pasos para reducir un problema:
  - Para demostrar que  $\Pi_1 \leq \Pi_2$ :
    1. Dar un algoritmo que transforme cada ejemplo,  $X_1$ , de  $\Pi_1$  en un ejemplo,  $X_2$ , de  $\Pi_2$ .
    2. Comprobar que dicho algoritmo es logarítmico en espacio.
    3. Comprobar que si  $X_1$  tiene solución positiva,  $X_2$  también la tiene.
    4. Comprobar que si  $X_2$  tiene solución positiva,  $X_1$  también la tiene.
- Problema del Viajante de Comercio (VC):
  - ¿Existe un circuito que visite todas las ciudades una sola vez y de coste no superior a  $B$ ?
  - Problema del Circuito Hamiltoniano (CH):
    - ¿Existe un circuito hamiltoniano?
  - $CH \leq VC$ 
    - CH:  $G = (V, E)$  con  $|V| = m$ .
    - VC:
      - Ciudades =  $V$
      - Distancias: 1 si están conectadas, 2 si no.
      - $B = m$
    - La transformación se puede calcular en espacio logarítmico.
    - Si existe un circuito hamiltoniano, existe un orden de viaje de valor menor o igual a  $m$ .
    - Si existe un orden de viaje de valor menor o igual a  $m$ , ese circuito es un circuito hamiltoniano en el grafo original.
- Composición de reducciones:
  - Si  $L_1 \leq L_2$  y  $L_2 \leq L_3$ ,  $L_1 \leq L_3$
  - $L_1$  y  $L_2$  son equivalentes si  $L_1 \leq L_2$  y  $L_2 \leq L_1$ .

### - Completitud:

- **Lenguajes NP-Complejos:**
  - Es un lenguaje de NP y cualquier otro lenguaje de NP se reduce a él.
  - Son los más difíciles o típico dentro de la clase NP.
- **Lenguaje P-Completo:**
  - Es un lenguaje de P y cualquier otro lenguaje de P se reduce a él.

### - Problema de la consistencia en lógica proposicional (**SAT**):

- Un conjunto de  $U = \{p_1, \dots, p_m\}$  de símbolos proposicionales y una colección  $C$  de cláusulas sobre estos símbolos.





**Llévate** Un **patinete**, unos **auriculares** o una **tablet** voom tab pro+teclado. Todos los estudiantes que presenten unos apuntes de **Wuolah** en tienda se les aplicará un **10% de descuento** en la compra de cualquiera de nuestros productos.

G R A N A D A  
Calle Puentezuelas, 49



- ¿Son consistentes las cláusulas?
- **Teorema de Cook:**
  - SAT es NP-Completo.
  - SAT es NP:
    - Para cada símbolo en  $U$  se selecciona un valor de verdad ( $V$  o  $F$ )
    - Para cada cláusula comprobar si se satisface.
    - Si todas se satisfacen, SI; si no, NO.
  - Reducción:
    - Dada una palabra  $x \in A^*$ , ¿pertenece  $x$  a  $L$ ?
      - La palabra pertenece al lenguaje si el ejemplo es consistente.
    - Transformación en espacio logarítmico.
- **3-SAT:**
  - Dos variantes:
    - 3 literales como máximo.
    - Exactamente 3 literales (3-SAT restringido).
    - Las dos son NP-Completo
  - Es NP porque es un caso particular de SAT, por lo que vale el mismo algoritmo.
  - SAT es NP-Completo:
    - Reducir SAT a 3-SAT.
- **2-SAT:**
  - Se construye en grafo:
    - Nodos: variables y sus negaciones (todos los posibles literales).
    - Arcos:  $(\alpha, \beta)$  es un arco si  $(\neg\alpha \vee \beta)$  está en el problema.
- **Cláusulas de Horn:**
  - Literales en los que hay ninguno o un literal positivo, como mucho.
- **MAX2SAT:**
  - Un conjunto de cláusulas con dos literales y un valor  $K \geq 0$ .
  - ¿Pueden satisfacerse, al menos,  $K$  cláusulas?
  - Reducción: 3-SAT  $\propto$  MAX2SAT
    - Construcción de Gadgets.
- **Frontera entre P y NP:**
  - Algunos problemas, cuando se plantean muy generales, son NP-Complejos.
  - Si se restringe el número de casos a resolver lo suficiente, se llega a un problema polinomial.
- **Restricción de 3-SAT:**
  - 3-SAT es NP-completo si hacemos 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.
- **Problema 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
    - Si se pueden asignar de forma ND 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  $\propto$  NAESAT
    - Se añade una nueva variable  $z$ .
    - Para cada cláusula con menos de 3 literales, añadimos  $z$ .
    - Si los literales son distintos,  $p \vee q \vee r$ , añadimos:
      - Una variable  $s$ .
        - $p \vee q \leftrightarrow s$
      - 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, si se cambia el valor de verdad de todas las variables, sigue teniendo solución.

- **Isomorfismo de Grafos:**
  - o  $G_1 = (V_1, E_1)$ ,  $G_2 = (V_2, E_2)$  dos grafos no dirigidos.
  - o Determinar si existe un isomorfismo entre  $G_1$  y  $G_2$ 
    - Aplicación biyectiva  $f: V_1 \rightarrow V_2$ , tal que  $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$ .
  - o No se ha demostrado que sea NP-completo ni se conoce ningún algoritmo polinómico para resolverlo.
  - o Se supone que no está en P ni es NP-Completo.
  - o Clase IG:
    - Todos los problemas que se pueden reducir al problema del isomorfismo de grafos.
    - Este problema es GI-Completo.
- Acoplamiento de Tripletas (**ACTRI**):
  - o Tres conjuntos distintos,  $W, X, Y$ , del mismo tamaño,  $q$ .
  - o Un subconjunto  $M \subseteq W \times X \times Y$  de compatibilidades.
  - o ¿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  $w_1 \neq w_2, x_1 \neq x_2, y_1 \neq y_2$ ?
  - o Es NP
    - De manera ND se elige un subconjunto de  $M$  y comprueba que si hay en cada elemento del subconjunto solo un elemento de cada conjunto.
  - o Reducción: 3-SAT  $\propto$  ACTRI
- Cubrimiento con conjuntos de tamaño 3 (**3-SET**):
  - o Un conjunto finito  $X$  con  $|X| = 3q$  y un subconjunto  $C$  de subconjuntos de  $X$  de tres elementos.
  - o ¿Existe  $C' \subseteq C$  tal que  $X = \bigcup_{A \in C'} A$  y los elementos de  $C'$  son disjuntos dos a dos?
  - o Es NP
  - o Reducción: ACTRI  $\propto$  3-SET
- **Problemas de grafos:**
  - o **Clique:**
    - Dado un grafo  $G = (V, E)$ , un clique es un subconjunto maximal totalmente conectado.
    - Dado un grafo  $G = (V, E)$  y un número natural  $J \leq |V|$ , determinar si existe un clique de tamaño mayor o igual que  $J$ .
  - o **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 toda arista del grafo tiene un extremo en  $V_c$ .
    - Dado un grafo  $G = (V, E)$  y un número natural  $K \leq |V|$ , determinar si existe un cubrimiento por vértices de tamaño menor o igual que  $K$ .
  - o **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 no hay ninguna arista que una vértices de  $V_i$ .
    - Dado un grafo  $G = (V, E)$  y un número natural  $J \leq |V|$ , determinar si existe un conjunto independiente de tamaño mayor o igual que  $J$ .
  - o Si  $G = (V, E)$  es un grafo y  $V^* \subseteq V$ , entonces:
    - $V^*$  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  $\bar{G} = (V, \bar{E})$ 
      - $\bar{E} = V \times V \setminus E$
  - o Los tres problemas son equivalentes.
  - o CV es NP-Completo
    - NP:
      - 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

- Circuito Hamiltoniano:
  - NP-Completo.
  - Reducción: CV  $\propto$  CH
- Problema de la Partición:
  - Un conjunto  $A$  y un tamaño para cada uno de sus elementos  $s: A \rightarrow \mathbb{N}$
  - Determinar si existe  $A' \subseteq A$  tal que se verifique:
$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$$
  - Es NP
  - Reducción: ACTRI  $\propto$  Problema de la Partición

#### - Técnicas de Reducción de Problemas:

- Restricción:
  - Si un problema  $\Pi_1$  es NP y un subproblema suyo,  $\Pi_2$ , es NP-Completo, entonces  $\Pi_1$  es NP-Completo
- Reemplazamiento Local:
  - Cada elemento de un problema NP-Completo  $\Pi'$  se transforma en una estructura del problema que estamos considerando,  $\Pi$ .
  - Reemplazamiento Local con Refuerzo:
    - Corresponde al caso en el 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.
- 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.

#### - Reducibilidad Turing:

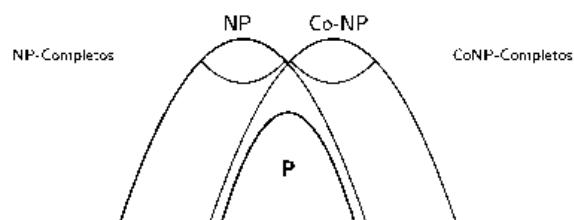
- Un problema  $\Pi$  se reduce Turing a  $\Pi'$ ,  $\Pi \leq_T \Pi'$ , si  $\Pi$  se puede resolver en tiempo polinómico mediante un algoritmo que puede llamar a una función que resuelve  $\Pi'$  contando cada llamada como un paso de cálculo.

#### - Problemas NP-Difíciles:

- Un problema  $\Pi$  es NP-difícil si existe un problema NP-Completo  $\Pi'$  que se pueda reducir (Turing) a  $\Pi$ ,  $\Pi' \leq_T \Pi$
- Si un problema NP-difícil se resuelve en tiempo polinómico, entonces  $P = NP$ .
  - Clase CoNP: complementarios a los problemas NP
  - Clase FNP: problemas que buscan una solución cuando saber si existe es NP.

#### - Clase Co-NP:

- $CoNP = \{\bar{L} : L \in NP\}$
- Tienen una MTND polinómica en la que la respuesta es afirmativa si **todas** las opciones responden **SI**.
  - Respuesta negativa si al menos una acaba en **NO**.
- Existen reducciones Turing entre los problemas de NP y los de Co-NP.
- Co-NP Completos
  - Un problema es CoNP Completo si está en la clase CoNP y cualquier otro problema CoNP se reduce a él.
  - $L$  es NP Completo  $\Leftrightarrow \bar{L}$  es CoNP Completo
  - Si un problema CoNP completo está en NP, entonces CoNP = NP.
  - Si  $P = NP$ , entonces CoNP = NP.



- Un lenguaje  $L \subseteq A^*$  está en NP si existe una relación  $R$  en  $A^* \times A^*$  calculable en tiempo polinómico y un polinomio  $p(n)$  tal que
- $$L = \{x \in A^* : \exists y \in A^* \text{ con } |y| \leq p(|x|), R(x, y) = 1\}$$

- Un lenguaje  $L \subseteq A^*$  está en CoNP si existe una relación  $R$  en  $A^* \times A^*$  calculable en tiempo polinómico y un polinomio  $p(n)$  tal que
 
$$L = \{x \in A^* : \forall y \in A^* \text{ con } |y| \leq p(|x|), R(x, y) = 1\}$$
- **Problemas de funciones FNP:**
  - Un problema de función está en FNP 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:**
    - La clase de problemas de funciones FNP tales que existe una MTD que las calcula en tiempo polinómico.
  - **Clase FNPT:**
    - La clase de los problemas FNP totales. Si para todo  $x$  existe un  $y$  con  $|y| \leq p(|x|)$  tal que  $R(x, y) = 1$
  - Reducciones en FNP:
    - Un problema de funciones  $\Pi$  se reduce a un problema  $\Pi'$  si existen funciones  $R$  y  $S$  calculables en espacio logarítmico tal que, para toda cadena  $x$  y  $z$  ocurre:
      - Si  $x$  es un ejemplo de  $\Pi$ , entonces  $R(x)$  es un ejemplo de  $\Pi'$ .
      - Si  $z$  es una solución correcta de  $R(x)$ , entonces  $S(z)$  es una solución correcta de  $x$ .
  - Problemas FNP-Completos:
    - Un problema es FNP-Completo si está en FNP y cualquier otro problema de esta clase se puede reducir a este problema.
  - $NP = P$  si  $FNP = FP$
  - Problemas de Funciones Totales:
    - Clase FNPT.
    - Para todo  $x$  existe un  $y$  tal que  $R(x, y)$ .
      - El problema de decisión no es difícil, siempre tiene respuesta positiva.
    - Muchos no se conocen que estén en FP.
  - La Red Feliz:
    - Un grafo  $G = (V, E)$  y un peso  $w$  (positivo o negativo) para cada arco.
    - Un estado de aplicación  $s: V \rightarrow \{-1, 1\}$
    - Un nodo  $i$  es feliz si:
 
$$s(i) \cdot \sum_{(i,j) \in E} s(j) \cdot w(i,j) \geq 0$$
    - Solución: un estado  $s$  en el que todos los nodos sean felices.
    - Está en FNP y es total.

