

Universidad de Oriente Núcleo Anzoátegui

Materia: Teoría de Autómatas y Lenguajes Formales

Profesor: José Bastardo

Sección: 01

Alumno: Luis Correa CI: xx.xxx.xxx

Departamento de Computación y Sistemas

Ingeniería en Computación

MAQUINAS DE TURING Y COMPLEJIDAD

Fecha 06 – 08 - 12

# INDICE

**1.- Maquinas de Turing**

## 1.1.- Defina formalmente una Maquina de Turing Determinística…………………….5

## 1.2.- Defina formalmente una Maquina de Turing No Determinística……….………..5

1.3 ¿Cómo sabe una máquina no determinista qué acción tomar de las varias…...6

1.4.-Defina algoritmo, según la teoría de Autómatas

Algoritmos y Máquinas de Turing……………………………………………..….7

1.5.-Como se denominan los lenguajes reconocibles usando Maquinas de Turing?.....9

1.6.- Construya una Maquina de Turing Determinística capaz de reconocer

el lenguaje L definido sobre Σ={0,1} y que está formado por todas las cadenas

de Σ\* que responden al siguiente patrón 1n 0n, con n>0……………………..…….…11

1.7.- Construya una Maquina de Turing Determinística capaz de reconocer

el lenguaje L definido sobre Σ={0,1} y que está formado por todas las cadenas

de Σ\* que responden al siguiente patrón 1n 02n1n, con n>0…………………………...12

1.8.- Que es el Orden ( ejecución)de un algoritmo…………………………………….14

2.-La teoría de la complejidad computacional

2.1.-Defina problemas P y problemas NP……..…………..……………………….….15

2.2.-Que es un problema NP-Completos?....................................................

2.3.- Haga una lista de 5 problemas NP-completos……………………………………..

2 .4.-P=NP?.....................................................................................................................

**ANEXOS**

# INTRODUCCIÓN

Los autómatas son modelos para diseñar circuitos digitales, Analizadores léxicos de un compilador. Buscar por palabras claves en un archivo o en internet, como en protocolos de comunicación.

Los autómatas son muy importantes ya que sin ellos no podría saberse los cambios de estado de una maquina.

Por ejemplo tenemos la Tesis de Church: Todo lo que es computable se puede calcular con una Máquina de Turing. Existen problemas que no son computables.

Es decir, la Teoría de los Lenguajes Formales (y de los Autómatas) permite responder a preguntas esenciales de la Informática.

**1.- Maquinas de Turing**

Una máquina de Turing  (MT) es un modelo  [computacional](http://es.wikipedia.org/wiki/Ciencias_de_la_computaci%C3%B3n) que realiza una [lectura](http://es.wikipedia.org/wiki/Lectura)/ [escritura](http://es.wikipedia.org/wiki/Escritura) de manera automática sobre una [entrada](http://es.wikipedia.org/wiki/Entrada) llamada cinta, generando una [salida](http://es.wikipedia.org/wiki/Salida_(inform%C3%A1tica)) en esta misma.

Este modelo está formado por un [alfabeto](http://es.wikipedia.org/wiki/Alfabeto) de entrada y uno de salida, un símbolo especial llamado blanco (normalmente b, \Delta o 0), un conjunto de [estados](http://es.wikipedia.org/wiki/Estado_(inform%C3%A1tica)) finitos y un conjunto de transiciones entre dichos estados. Su funcionamiento se basa en una [función de transición](http://es.wikipedia.org/wiki/Funci%C3%B3n_de_transici%C3%B3n), que recibe un estado inicial y una [cadena de caracteres](http://es.wikipedia.org/wiki/Cadena_de_caracteres) (la cinta, la cual puede ser infinita) pertenecientes al [alfabeto](http://es.wikipedia.org/wiki/Alfabeto) de entrada. La máquina va leyendo una celda de la cinta en cada paso, borrando el símbolo en el que se encuentra posicionado su cabezal y escribiendo un nuevo símbolo perteneciente al alfabeto de salida, para luego desplazar el cabezal a la izquierda o a la derecha (solo una celda a la vez). Esto se repite según se indique en la [función de transición](http://es.wikipedia.org/wiki/Funci%C3%B3n_de_transici%C3%B3n), para finalmente detenerse en un estado final o de aceptación, representando así la salida.

De manera formal, una máquina de Turing (TM de sus siglas en inglés Turing Machine) se representa por: M=(Q, S , G , d , q0, B, F) en donde:

Q es un conjunto finito de estados.

G es el conjunto finito de símbolos de cinta admisibles.

B símbolo de G , es el espacio en blanco.

S subconjunto de G que no incluye a B, es el conjunto de los símbolos de entrada.

D es la función de movimientos siguiente, una transformación de Q x G a Q x G x {L, R} (d puede sin embargo, permanecer indefinida por algunos argumentos).

q0 en Q es el estado inicial.

F Í Q es el conjunto de estados finales.

## 1.1.- Defina formalmente una Maquina de Turing Determinística.

La entrada de una máquina de Turing viene determinada por el estado actual y el símbolo leído, un par (estado, símbolo), siendo el cambio de estado, la escritura de un nuevo símbolo y el movimiento del cabezal, las acciones a tomar en función de una entrada. En el caso de que para cada par (estado, símbolo) posible exista a lo sumo una posibilidad de ejecución, se dirá que es una máquina de Turing determinista.

## 1.2.- Defina formalmente una Maquina de Turing No Determinística.

Mientras que en el caso de que exista al menos un par (estado, símbolo) con más de una posible combinación de actuaciones se dirá que se trata de una máquina de Turing no determinista.

La función de transición \delta en el caso no determinista, queda definida como sigue:

**δ :Q×Γ→P( Q×Γ×{L,R})**

**1.3 ¿Cómo sabe una máquina no determinista qué acción tomar de las varias posibles?**

Hay dos formas de verlo: una es decir que la máquina es "el mejor adivino posible", esto es, que siempre elige la transición que finalmente la llevará a un estado final de aceptación. La otra es imaginarse que la máquina se "clona", bifurcándose en varias copias, cada una de las cuales sigue una de las posibles transiciones. Mientras que una máquina determinista sigue un único "camino computacional", una máquina no determinista tiene un "árbol computacional". Si cualquiera de las ramas del árbol finaliza en un estado de aceptación, se dice que la máquina acepta la entrada.

La capacidad de cómputo de ambas versiones es equivalente; se puede demostrar que dada una máquina de Turing no determinista existe otra máquina de Turing determinista equivalente, en el sentido de que reconoce el mismo lenguaje, y viceversa. No obstante, la velocidad de ejecución de ambos formalismos no es la misma, pues si una máquina no determinista M reconoce una cierta palabra de tamaño n en un tiempo O(t(n))\!, la máquina determinista equivalente reconocerá la palabra en un tiempo  O(2 ^{t(n)})\!. Es decir, el no determinismo permitirá reducir la complejidad de la solución de los problemas, permitiendo resolver, por ejemplo, problemas de complejidad exponencial en un [tiempo polinómico](http://es.wikipedia.org/wiki/Tiempo_polin%C3%B3mico).

La máquina de Turing determinista es una tulpa formada por varios objetos.

1. Q – un conjunto finito de estados.

2. Σ - El alfabeto de entrada: un conjunto finito de símbolos que no contiene al símbolo blanco.

3. Γ - el alfabeto de cinta, tal que: Σ⊆Γ y en símbolo “\_” está en Γ

4. δ:Q×Γ→Q×Γ×{I,D} – la función de transición.

5. q0 ∈Q – el estado inicial

6. qsi ∈Q – el estado aceptador.

7. qno ∈Q – el estado no aceptador. Qno ≠qsi

## La máquina de Turing es un modelo computacional introducido por Alan Turing en el trabajo “On computable numbers, with an application to the Entscheidungsproblem”, publicado por la Sociedad Matemática de Londres, en el cual se estudiaba la cuestión planteada por David Hilbert sobre si las matemáticas son decidibles, es decir, si hay un método definido que pueda aplicarse a cualquier sentencia matemática y que nos diga si esa sentencia es cierta o no. Turing construyó un modelo formal de computador, la máquina de Turing, y demostró que existían problemas que una máquina no podía resolver. La máquina de Turing es un modelo matemático abstracto que formaliza el concepto de algoritmo. Una máquina de Turing con una sola cinta puede ser definida como una 6-tupla, donde Q es un conjunto finito de estados Γ es un conjunto finito de símbolos de cinta, el alfabeto de cinta es el estado inicial es un símbolo denominado blanco, y es el único símbolo que se puede repetir un número infinito de veces es el conjunto de estados finales de aceptación es una función parcial denominada función de transición, donde L es un movimiento a la izquierda y R es el movimiento a la derecha.

## Existen en la literatura un abundante número de definiciones alternativas, pero todas ellas tienen el mismo poder computacional, por ejemplo se puede añadir el símbolo como símbolo de “no movimiento” en un paso de cómputo.

**1.4.-Defina algoritmo, según la teoría de Autómatas**

En [programación](http://www.alegsa.com.ar/Dic/programacion.php), los algoritmos se implementan en forma de [sentencias](http://www.alegsa.com.ar/Dic/sentencia%20de%20programacion.php) en algún [lenguaje de programación](http://www.alegsa.com.ar/Dic/lenguaje%20de%20programacion.php). De esta manera, la forma de escribir los algoritmos depende del lenguaje de programación, y del [paradigma](http://www.alegsa.com.ar/Dic/paradigma.php) usado. Estos son los algoritmos que pueden ser interpretados por una [computadora](http://www.alegsa.com.ar/Dic/computadora.php) y así ser [ejecutados](http://www.alegsa.com.ar/Dic/ejecutar.php).

Los algoritmos también pueden representarse gráficamente empleando [diagramas de flujo](http://www.alegsa.com.ar/Dic/diagrama%20de%20flujo.php) o formas similares. De esta manera, son fácilmente comprensibles, especialmente para personas que no son [programadores](http://www.alegsa.com.ar/Dic/programador.php). También, de esta manera, los algoritmos son más "universales", pues no dependen de un lenguaje de programación específico.  
  
Los algoritmos también pueden escribirse en [pseudocódigo](http://www.alegsa.com.ar/Dic/pseudocodigo.php), lo que también los hace fáciles de entender.

Se hacen intentos para que las computadoras interpreten y ejecuten los diagramas de flujo y los pseudocódigos, pero no logran la flexibilidad, potencia y velocidad de los algoritmos puramente escritos en un lenguaje de programación específico.

Un algoritmo también puede expresarse en lenguaje natural, aunque esto puede traer ambigüedades e interpretaciones erróneas (la ambigüedad es propia del lenguaje humano).  
  
Algoritmos en programación

Un [programa](http://www.alegsa.com.ar/Dic/programa.php) de computadora es un algoritmo que le dice a la computadora los pasos específicos para llevar a cabo una tarea. Los algoritmos son rigurosamente definidos para que la computadora pueda interpretarlos. El orden en que se ejecuta cada uno de los pasos que constituyen un algoritmo es fundamental. El orden más básico es de arriba hacia abajo, ejecutándose una instrucción tras otra de un [código](http://www.alegsa.com.ar/Dic/codigo%20fuente.php). Pero un algoritmo puede variar en su flujo u orden de ejecución de pasos dependiendo de los valores de inicio o que entran durante su ejecución. El flujo es manejado por las [estructuras de control](http://www.alegsa.com.ar/Dic/estructura%20de%20control.php).

Algunos autores consideran que el flujo de ejecución de un algoritmo debe detenerse correctamente alguna vez, y que esto forma parte de la definición de algoritmo. En tanto, otros no lo consideran así.

Un algoritmo es la implementación de una máquina de Turing tal que el conjunto de sus entradas es un lenguaje decidible. La resolución práctica de un problema exige por una parte un algoritmo o método de resolución y por otra un programa o codificación de aquel en un ordenador real. Ambos componentes tienen su importancia.

Es decir, si dado un conjunto de entradas bajo las cuales una MT logra un estado de parada para cada entrada, la máquina corresponde a la implementación de un algoritmo.  Esta es la Tesis de Church – Turing. No es un teorema pues no se puede demostrar matemáticamente, de manera general y categórica. Es solo la afirmación de que el concepto informal de algoritmo corresponde a un objeto matemático. Al ser solo una afirmación no demostrable, puede suceder que luego fuera refutada. Para que esto ocurra, se necesitaría encontrar un autómata más potente que uno de Turing tal que fuese la implementación de un algoritmo[[9]](http://www.filosoficas.unam.mx/~morado/LogicaHoy/alvarado.htm" \l "_ftn9" \o "). Si bien hay algunas propuestas interesantes que pretende generalizar a la MT, hasta la fecha ninguna de ellas ha sido aceptada para sustituir nuestro actual concepto de procedimiento computable.

## Por otro lado, mientras que los lenguajes computables son una infinidad numerable, los lenguajes no computables son una infinidad no numerable[[10]](http://www.filosoficas.unam.mx/~morado/LogicaHoy/alvarado.htm" \l "_ftn10" \o "). Por ello, son más los lenguajes no computables o indecidibles que los computables o decidibles.

## 

**1.5.-Como se denominan los lenguajes reconocibles usando Maquinas de Turing**

Un lenguaje que es aceptado por una máquina de Turing se conoce como lenguaje **recursivamente enumerable** (r.e.). El término enumerable proviene que dichos lenguajes son aquellos cuyas cadenas pueden ser listadas (enumeradas) por una máquina de Turing. Esta clase de lenguajes es bastante grande, incluyendo los lenguajes libres de contexto.

Hay lenguajes r.e. para los cuales ninguna máquina de Turing que los acepte para con todas las entradas (naturalmente, cualquier máquina de Turing para dichos lenguajes debe parar para toda cadena que pertenezca realmente al lenguaje). La subclase de los lenguajes recursivamente enumerables que son aceptados por al menos una máquina de Turing que para con toda cadena de entrada (dependiendo de si la cadena es aceptada o no), se conoce por la clase de los lenguajes recursivos.Puesto que las máquinas de Turing pueden leer y escribir sobre su cinta pueden convertir la entrada en salida. La transformación de la entrada en salida es el primer propósito de las computadoras digitales; por tanto, una máquina de Turing se considera como un modelo abstracto de una computadora. Se supone que la entrada para la máquina de Turing está formada por todos los símbolos de la cinta que no son blancos. La salida está formada por cualquiera de los símbolos que queden en la cinta cuando la computación termina.

Las máquinas de Turing pueden ser consideradas como la implementación de una función de cadena f definida mediante f(w) = u cuando se cumple qsw \* q f u, donde qs es el estado inicial y q f es un estado final, por lo que se requiere que la cabeza de lectura/escritura empiece y termine, respectivamente, sobre el símbolo de las cadenas de entrada y salida que está situado más a la izquierda.

Definiendo lo anterior se dice que una función de cadena f es Turing computable si existe una máquina de Turing M = (Q, S , G , q1,B,F,d ) para la cual q1w \* q f u para algún q f Î F, cuando f(w) = u.

Se puede extender la anterior definición de funciones integrales, como se muestra en el siguiente ejemplo.

Ejemplo. Supongamos que tenemos S ={a,b} y que representamos los enteros positivos mediante cadenas de a’s. Así, el entero positivo n estaría representado por an. La función suma f(n,m) = n + m podría ser implementada mediante la transformación de anbam en an+mb. Podríamos obtener una máquina de Turing para la suma, que estaría representada por M = (Q, S , G , q0, B,F,d ), donde

**Q = {q1, q2, q3, q4, q5}**

**q0 = {q5}**

**y d dada por las siguientes transformaciones:**

**d (q1,a) = (q1,a,R)**

**d (q1,b) = (q2,a,R)**

**d (q2,a) = (q2,a,R)**

**d (q2,B) = (q3,B,L)**

**d (q3,a) = (q4,b,L)**

**d (q4,a) = (q4,a,L)**

**d (q4,B) = (q5,B,R)**

## Esta máquina de Turing desplaza la b hacia el final, a la derecha de an+m. Para ello se crea una a extra. La máquina de Turing "recordará" que se ha creado una a al pasar al estado q2 una vez que se ha encontrado la b, y entonces se escribirá una b sobre la a que está al final de la cadena. Cuando termina la máquina de Turing la cabeza de lectura/escritura está sobre la a que encuentra más a la izquierda.

Una máquina de Turing se puede comportar como un aceptador de un lenguaje. Si colocamos una cadena w en la cinta, situamos la cabeza de lectura/escritura sobre el símbolo del extremo izquierdo de la cadena w y ponemos en marcha la máquina a partir de su estado inicial. Entonces w es aceptada si, después de una secuencia de movimientos, la máquina de Turing llega a un estado final y para. Por tanto w es aceptada. Si qw \* w1pw2 para algún estado final p y unas cadenas w1 y w2. Entonces, se obtiene la siguiente definición:

Sea M = (Q, S , G , q0=q1, B, F, d ) una máquina de Turing. Entonces el lenguaje aceptado por M es: L(M) = {wÎ S \*½ q1w \* w1pw2 para pÎ F y wiÎ G \*}.

Ejemplo. Diseñar una TM que acepte el lenguaje regular a\* sobre S ={a, b}. Comenzando con el símbolo que está más a la izquierda en una cadena, realizaremos un análisis hacia la derecha, leyendo cada símbolo y comprobando que es una a; si lo es, realizaremos un desplazamiento hacia la derecha. Si encontramos un blanco (B) sin que se haya leído ningún símbolo que no fuera a, paramos y aceptamos la cadena. Si por el otro lado, encontramos un símbolo que no es ni a ni B, podemos parar en un estado que no es de aceptación.

**Sea Q = {q1,q2}, q0=q1 y F={q2}, y sea d definida por:**

**d (q1,a) = (q1,a,R)**

**d (q1,B) = (q2,B,R)**

## Esta máquina de Turing para en el estado q2, sólo si se analiza una cadena de 0 ó más a’s.

## Para rechazar una cadena que no es aceptable, lo único que hay que hacer es evitar que se llegue a un estado final. En este ejemplo las cadenas que no son aceptables, provocan que la máquina pare en un estado que no es final. Otra alternativa para rechazar una cadena es entrar en un bucle infinito.

**1.6.- Construya una Maquina de Turing Determinística capaz de reconocer el lenguaje L definido sobre Σ={0,1} y que está formado por todas las cadenas de Σ\* que responden al siguiente patrón 1n 0n, con n>0.**

**1.7.- Construya una Maquina de Turing Determinística capaz de reconocer el lenguaje L definido sobre Σ={0,1} y que está formado por todas las cadenas de Σ\* que responden al siguiente patrón 1n 02n1n, con n>0.**

**1.8.- Que es el Orden (ejecución) de un algoritmo**

El orden de ejecución será casi siempre crítico para su funcionamiento en general, se asume que las instrucciones se enumeran explícitamente, y deben ejecutarse desde arriba hacia abajo.

La solución de los más variados problemas en un agente de cómputo se reduce inevitablemente a la realización de simples operaciones aritméticas y lógicas. Sin embargo, en muchas situaciones una misma expresión matemática admite varias posibilidades para su cálculo, que difieren solamente en el orden de ejecución de las operaciones. Ahora bien, los resultados de los cálculos en un agente de cómputo dependen del orden de realización de las operaciones y la diferencia en los errores puede ser altamente significativo.

**2.-La teoría de la complejidad computacional**

Es la rama de la [teoría de la computación](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_computaci%C3%B3n) que estudia, de manera teórica, la complejidad inherente a la resolución de un problema computable. Los recursos comúnmente estudiados son el tiempo (mediante una aproximación al número y tipo de pasos de ejecución de un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) para resolver un problema) y el espacio (mediante una aproximación a la cantidad de memoria utilizada para resolver un problema). Se pueden estudiar igualmente otros parámetros, tales como el número de procesadores necesarios para resolver el problema en paralelo. La teoría de la complejidad difiere de la [teoría de la computabilidad](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_computabilidad) en que ésta se ocupa de la factibilidad de expresar problemas como algoritmos efectivos sin tomar en cuenta los recursos necesarios para ello.

Los problemas que tienen una solución con orden de complejidad lineal son los problemas que se resuelven en un tiempo que se relaciona linealmente con su tamaño.

Hoy en día las computadoras resuelven problemas mediante algoritmos que tienen como máximo una complejidad o coste computacional polinómico, es decir, la relación entre el tamaño del problema y su tiempo de ejecución es polinómica. Éstos son problemas agrupados en la clase [P](http://es.wikipedia.org/wiki/Tiempo_polin%C3%B3mico). Los problemas que no pueden ser resueltos por nuestras computadoras (las cuales son Máquinas Determinísticas), que en general poseen costes[factorial](http://es.wikipedia.org/wiki/Factorial) o [combinatorio](http://es.wikipedia.org/wiki/Combinatoria) pero que podrían ser procesados por una máquina no-determinista, están agrupados en la clase [NP](http://es.wikipedia.org/wiki/NP_(Complejidad_computacional)). Estos problemas no tienen una solución práctica, es decir, una máquina determinística (como una computadora actual) no puede resolverlos en un tiempo razonable.

**2.1.-** [**Clase de complejidad**](http://es.wikipedia.org/wiki/Clase_de_complejidad)***:***

Los problemas de decisión se clasifican en conjuntos de complejidad comparable llamados [clases de complejidad](http://es.wikipedia.org/wiki/Clase_de_complejidad).

* **La**[**clase de complejidad P**](http://es.wikipedia.org/wiki/Tiempo_polin%C3%B3mico)

Es el conjunto de los problemas de decisión que pueden ser resueltos en una [máquina determinista](http://es.wikipedia.org/wiki/M%C3%A1quina_de_Turing) en tiempo polinómico, lo que corresponde intuitivamente a problemas que pueden ser resueltos aún en el peor de sus casos.

* **La**[**clase de complejidad NP**](http://es.wikipedia.org/wiki/NP_(Complejidad_computacional))

Es el conjunto de los problemas de decisión que pueden ser resueltos por una máquina no determinista en tiempo polinómico. Esta clase contiene muchos problemas que se desean resolver en la práctica, incluyendo el [problema de satisfacibilidad booleana](http://es.wikipedia.org/wiki/Problema_de_satisfacibilidad_booleana) y el [problema del viajante](http://es.wikipedia.org/wiki/Problema_del_viajante), un camino Hamiltoniano para recorrer todos los vértices una sola vez. Todos los problemas de esta clase tienen la propiedad de que su solución puede ser verificada efectivamente.

**2.2.-Que es un problema NP-Completos?**

Un [problema de decisión](http://es.wikipedia.org/wiki/Problema_de_decisi%C3%B3n) C es NP-completo si: C es un problema NP, y Todo problema de NP se puede [transformar polinómicamente](http://es.wikipedia.org/wiki/Transformaci%C3%B3n_polin%C3%B3mica) en C.

Se puede demostrar que C es NP demostrando que un candidato a solución de C puede ser verificado en tiempo polinómico.

Una transformación polinómica de L en C es un algoritmo determinista que transforma instancias de l ∈ L en instancias de c ∈ C, tales que la respuesta a c es positiva si y sólo si la respuesta a l lo es.

Como consecuencia de esta definición, de tenerse un algoritmo en P para C, se tendría una solución en P para todos los problemas de NP.

Esta definición fue propuesta por [Stephen Cook](http://es.wikipedia.org/wiki/Stephen_Cook) en [1971](http://es.wikipedia.org/wiki/1971). Al principio parecía sorprendente que existieran problemas NP-completos, pero Cook demostró ([teorema de Cook](http://es.wikipedia.org/wiki/Teorema_de_Cook)) que el [problema de satisfacibilidad booleana](http://es.wikipedia.org/wiki/Problema_de_satisfacibilidad_booleana) es NP-completo. Desde entonces se ha demostrado que miles de otros problemas pertenecen a esta clase, casi siempre por reducción a partir de otros problemas para los que ya se había demostrado su pertenencia a NP-completo; muchos de esos problemas aparecen en el libro de Garey and Johnson's de 1979 Computers and Intractability: A Guide to NP-completeness.

Un problema que satisface la segunda condición pertenece a la clase [NP-hard](http://es.wikipedia.org/wiki/NP-hard) independientemente de que satisfaga la primera.

**2.3.- Haga una lista de 5 problemas NP-completos.**

Mientras que la pertenencia del problema SAT o de [satisfacibilidad booleana](http://es.wikipedia.org/wiki/Problema_de_satisfacibilidad_booleana) a la clase de los NP-completos fue demostrada utilizando mecanismos particulares, las pertenencias de los 5 problemas siguientes fueron demostradas mediante [reducciones polinomiales](http://es.wikipedia.org/wiki/Reducci%C3%B3n_polinomial). Así, el problema SAT se redujo polinomialmente a los problemas 0-1 INTEGER PROGRAMMING, CLIQUE y 3-SAT, y estos a su vez se redujeron a otros varios

Note que los nombres de los problemas están escritos con letras mayúsculas y corresponden a abreviaciones del nombre en inglés, como es lo usual; junto a ellos, entre paréntesis, se escribe la traducción del nombre en español.

EXACT COVER ([Problema de la cobertura exacta](http://es.wikipedia.org/w/index.php?title=Problema_de_la_cobertura_exacta&action=edit&redlink=1))

KNAPSACK ([Problema de la mochila](http://es.wikipedia.org/wiki/Problema_de_la_mochila))

JOB SEQUENCING ([Problema de las secuencias de trabajo](http://es.wikipedia.org/w/index.php?title=Problema_de_las_secuencias_de_trabajo&action=edit&redlink=1))

PARTITION ([Problema de la partición](http://es.wikipedia.org/wiki/Problema_de_la_partici%C3%B3n))

MAX-CUT ([Problema del corte máximo](http://es.wikipedia.org/w/index.php?title=Problema_del_corte_m%C3%A1ximo&action=edit&redlink=1))

Tras un tiempo se descubrió que muchos de estos problemas podían ser resueltos si su enunciado se particularizaba a unas ciertas clases, o podían ser resueltos aproximadamente con un error máximo de un cierto porcentaje. Sin embargo [David Zuckerman](http://es.wikipedia.org/w/index.php?title=David_Zuckerman_(mathem%C3%A1tico)&action=edit&redlink=1) demostró en 1996 que cada uno de estos problemas tiene una versión restringida de optimización que es no aproximable a menos que P = NP, demostrando que la versión de la reducción, dada por Karp, generaliza un tipo específico de reducción por aproximación.

**2.4.- La pregunta P=NP**

El saber si las clases P y NP son iguales es el más importante problema abierto en Computación teórica. Incluso hay un [premio de un millón de dólares](http://es.wikipedia.org/wiki/Problemas_del_milenio) para quien lo resuelva.

Preguntas como ésta motivan la introducción de los conceptos de *hard* (difícil) y *completo*. Un conjunto *X* de problemas es *hard* con respecto a un conjunto de problemas *Y* ( 'Y' pertenecientes a NP) si X>Y o X=Y, es decir Y se puede escribir como un conjunto de soluciones de los problemas X. En palabras simples, Y es "más sencillo" que X. El término sencillo se define precisamente en cada caso. El conjunto hard más importante es [NP-hard](http://es.wikipedia.org/wiki/NP-hard). El conjunto *X* es completo para *Y* si es hard para *Y* y es también un subconjunto de *Y* (caso X=Y). El conjunto completo más importante es [NP-completo](http://es.wikipedia.org/wiki/NP-completo). En otras palabras, los problemas del conjunto NP-completo tienen la característica de que, si se llega a encontrar una solución en tiempo P para algún miembro del conjunto (cualquiera de los problemas de NP-completo), entonces de hecho existe una solución en tiempo P para todos los problemas de NP-completo.

# BIBLIOGRAFIA

[http://www.sc.ehu.es/jiwhehum2/TC/temas/[2]turing.pdf](http://www.sc.ehu.es/jiwhehum2/TC/temas/%5b2%5dturing.pdf)

<http://www.mitecnologico.com/Main/LaMaquinaDeTuring>

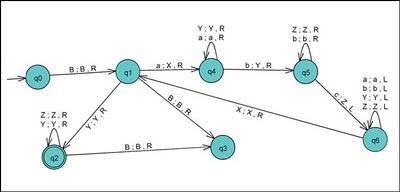
[http://www.mitecnologico.com](http://www.mitecnologico.com/)

<http://www.wikiteka.com/trabajos/algoritmo-4/>

**ANEXOS**



Alan Mathison Turing (1912–1954)



Esta máquina de Turing está definida sobre el [alfabeto](http://es.wikipedia.org/wiki/Alfabeto) \Sigma=\{a,b,c\}, posee el conjunto de [estados](http://es.wikipedia.org/wiki/Estados) Q=\{q_o,q_1,q_2,q_3,q_4,q_5,q_6\}, con las transiciones que se pueden ver. Su estado inicial es q_0 y el estado final es q_2, el lenguaje de salida  
\Gamma=\{X,Y,Z,B\} siendo B el símbolo denominado "blanco". Esta máquina reconoce la expresión regular de la forma a^n b^n c^n con n >= 0.