Sobre algoritmos, máquinas de Turing y complejidad

Juan Carlos Martínez

Facultad de Matemáticas e Informática
Universidad de Barcelona

Congreso en homenaje a Luis Martínez

Objetivo de la charla

Mostraremos el concepto de máquina de Turing, que es una formulación matemática del concepto intuitivo de algoritmo.

A continuación, definiremos las principales clases de complejidad y mostraremos algunos problemas abiertos importantes sobre ellas.

Complejidad Computacional

La complejidad computacional es un campo nuevo, reciente para la Matemática.

Parte del principio de que no basta con disponer de un algoritmo que resuelva un problema. Se necesita, además, que el comportamiento del algoritmo sea "razonable" en relación con el tiempo de ejecución del algoritmo y la memoria requerida.

El concepto de algoritmo

Para poder profundizar en las clases de complejidad y obtener resultados sobre ellas, necesitamos definirlas rigurosamente. Y para ello, necesitamos una definición matemática del concepto informal de algoritmo.

En la primera mitad del siglo XX, se introdujeron diversas formulaciones matemáticas del concepto de algoritmo. Y se pudo demostrar que todas las formulaciones introducidas son equivalentes, es decir, todas las diferentes formulaciones computan la misma clase de funciones.

En esta charla, utilizaremos la formulación más conocida del concepto del algoritmo, que es la noción de máquina de Turing.

Conceptos básicos de lenguajes formales

Para poder definir el concepto de máquina de Turing, necesitamos introducir previamente algunos conceptos básicos referentes a lenguajes formales.

Un alfabeto es un conjunto finito no vacío de símbolos.

Una palabra sobre un alfabeto Σ es una secuencia finita de símbolos de Σ.

Denotamos por λ a la palabra vacía, es decir, a la palabra que no tiene ningún símbolo.

Un lenguaje es un conjunto de palabras sobre un alfabeto.

Si Σ es un alfabeto, denotamos por Σ^* al conjunto de todas las palabras sobre Σ.

Si $x \in \Sigma^*$, denotamos por $|x|$ a la longitud de x, es decir, al número de símbolos de x (contando repeticiones).

Orden lexicográfico

Si $\Sigma = \{a_1, \ldots, a_n\}$ es un alfabeto donde $a_1 < \cdots < a_n$, el orden lexicográfico de Σ^* es el siguiente:

$$\lambda$$
$$a_1$$
$$\vdots$$
$$a_n$$
$$a_1a_1$$
$$a_1a_2$$
$$\vdots$$
$$a_1a_n$$
$$a_2a_1$$
$$a_2a_2$$
$$\vdots$$
$$a_na_n$$
$$a_1a_1a_1$$
$$\vdots$$

El modelo de máquina de Turing

Una máquina de Turing tiene asociada una cinta, que está dividida en celdas. Cada celda contiene un símbolo de un alfabeto de cinta asociado a la máquina de Turing. Suponemos que la cinta es infinita por la derecha. La información de la máquina se encuentra entonces almacenada en las celdas, y la máquina accede a dicha información mediante un puntero que señala en un instante dado el contenido de una celda de la cinta. Al producirse un paso de cómputo, el puntero puede escribir un símbolo en la celda que esté señalando y a continuación puede eventualmente moverse a la siguiente celda a la derecha o la siguiente celda a la izquierda de la cinta.

La máquina tiene además asociada una "unidad de control", que en un paso de cómputo se encuentra en un cierto estado. Cuando se pasa al siguiente paso de cómputo, el estado puede variar.

El modelo de máquina de Turing

La primera celda de la cinta contendrá siempre un carácter especial, que denotaremos por $\$$.

Representamos por $*$ el carácter blanco. Cuando una celda de la cinta contenga $*$, dicha celda no contiene información.

Denotaremos por:

$-1 =$ movimiento del puntero a la izquierda,

$1 =$ movimiento del puntero a la derecha,

$0 =$ no hay movimiento del puntero.

Máquinas de Turing deterministas

Una máquina de Turing determinista es una estructura $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ donde:

(1) K es un conjunto finito y no vacío de estados,

(2) Σ es al alfabeto de entrada,

(3) Γ es al alfabeto de la cinta,

(4) $\Sigma \cup \{\$, *\} \subseteq \Gamma$ y $\$, * \notin \Sigma$,

(5) $K \cap \Gamma = \emptyset$,

(6) $q_0 \in K$ es el estado inicial,

(7) $q_F \in K$ es el estado aceptador,

(8) δ es una función parcial de $(K \setminus \{q_F\}) \times \Gamma$ en $K \times \Gamma \times \{-1, 1, 0\}$ tal que para todo $q \in K \setminus \{q_F\}$:

   (a) $\delta(q, \$) = (p, b, i)$ implica $b = \$$ e $i \neq -1$,
   (b) si $a \neq \$$, $\delta(q, a) = (p, b, i)$ implica $b \neq \$$.

Cómputos con máquinas de Turing

Supongamos que $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ es una máquina de Turing determinista.

Inicialmente, la máquina M se encuentra en el estado q_0 y se registra una palabra de entrada $a_1 \ldots a_n \in \Sigma^*$ en el comienzo de la cinta, justo a continuación del \$; el resto de las celdas de la cinta contienen el carácter blanco; y el puntero de la cinta señala a la segunda celda.

Los cómputos de la máquina se realizan por medio de la función δ, la cual se aplica para pasar de un paso de cómputo al siguiente. El argumento de δ es el par (q, a) donde q es el estado actual de la máquina de Turing y a es el símbolo al que señala el puntero. Si $\delta(q, a) = (p, b, i)$, entonces la máquina pasa al estado p, escribe en la cinta el símbolo b y realiza el movimiento indicado por i. Y si el par (q, a) no pertenece al dominio de δ, la máquina para.

Cómputos con máquinas de Turing

Sea $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$ una máquina de Turing determinista.

(a) Una configuración de M es una palabra $\alpha q \beta$ donde $q \in K$ y $\alpha, \beta \in \Gamma^*$ tales que β no acaba en $*$.

Si en paso de cómputo, la configuración de M es $\alpha q \beta$, M está en el estado q, el contenido de la cinta es $\alpha\beta$ seguida de blancos a la derecha, y el puntero señala al primer carácter después de α.

(b) Si C y C' son configuraciones de M, decimos que hay un cómputo de C a C', si en M podemos pasar de C a C' aplicando un número finito de veces la función δ. Escribiremos entonces $C \vdash_M^* C'$.

Lenguaje asociado a una máquina de Turing determinista

Sea M una máquina de Turing determinista. Sea Σ el alfabeto de entrada de M. Sea $x \in \Sigma^*$.

(1) Decimos que M para sobre x, si el cómputo de M sobre la entrada x termina.

(2) Decimos que M es de parada segura, si para todo $x \in \Sigma^*$, el cómputo de M sobre la entrada x termina.

(3) Una palabra $x \in \Sigma^*$ es reconocida por M, si M sobre la entrada x para en el estado aceptador, es decir, si $\$q_0x \vdash_M^* \$yq_Fz$ donde $y, z \in \Gamma^*$.

(4) Definimos entonces

$$L(M) = \{x \in \Sigma^* : x \text{ es reconocida por } M\}.$$

Representación de una máquina de Turing mediante un grafo

Consideremos una máquina de Turing determinista $M = (K, \Sigma, \Gamma, \delta, q_0, q_F)$. Para representar M mediante un grafo, seguimos el siguiente algoritmo:

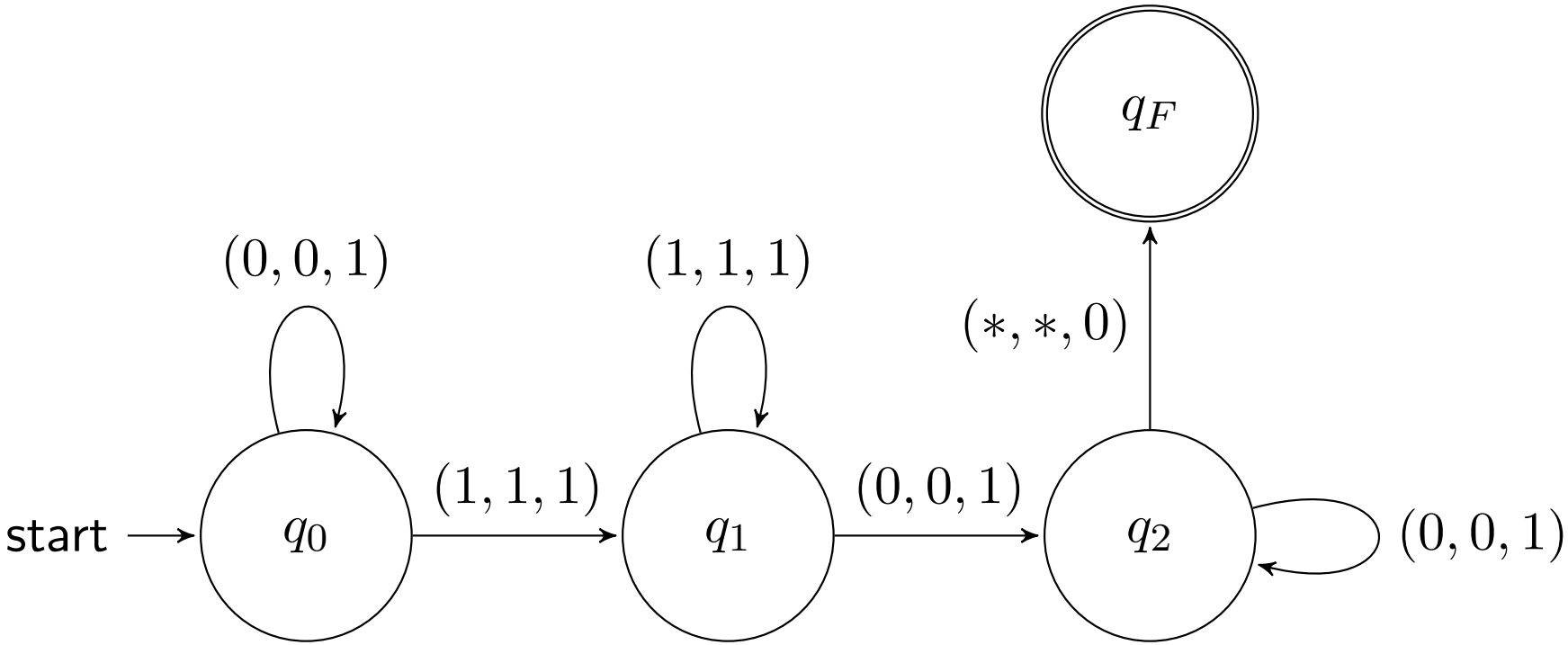
(1) Los nodos del grafo son los estados de K.

(2) Se marca el estado inicial con una flecha, y se marca el estado aceptador con un doble círculo.

(3) Si $\delta(q, a) = (p, b, i)$, se dibuja un arco en el grafo de q a p con etiqueta (a, b, i).

Ejemplo

M:



Tenemos que $L(M) = \{0^n 1^{m+1} 0^{k+1} : n, m, k \geq 0\}$.

Función computada por una máquina de Turing

Sea M una máquina de Turing determinista. Sea Σ el alfabeto de entrada de M.

(1) Si M para sobre una entrada x, denotamos por $M(x)$ a la palabra de Σ^* que aparece en la última configuración del cómputo de M sobre x, que está comprendida entre los dos primeros caracteres de $\Gamma \setminus \Sigma$. Diremos entonces que $M(x)$ es el resultado del cómputo de M sobre la entrada x.

(2) Definimos la función $f_M : \Sigma^* \to \Sigma^*$ por:

$$f_M(x) = \begin{cases} M(x), & \text{si } M \text{ para sobre la entrada } x. \\ \text{indefinido}, & \text{en caso contrario.} \end{cases}$$

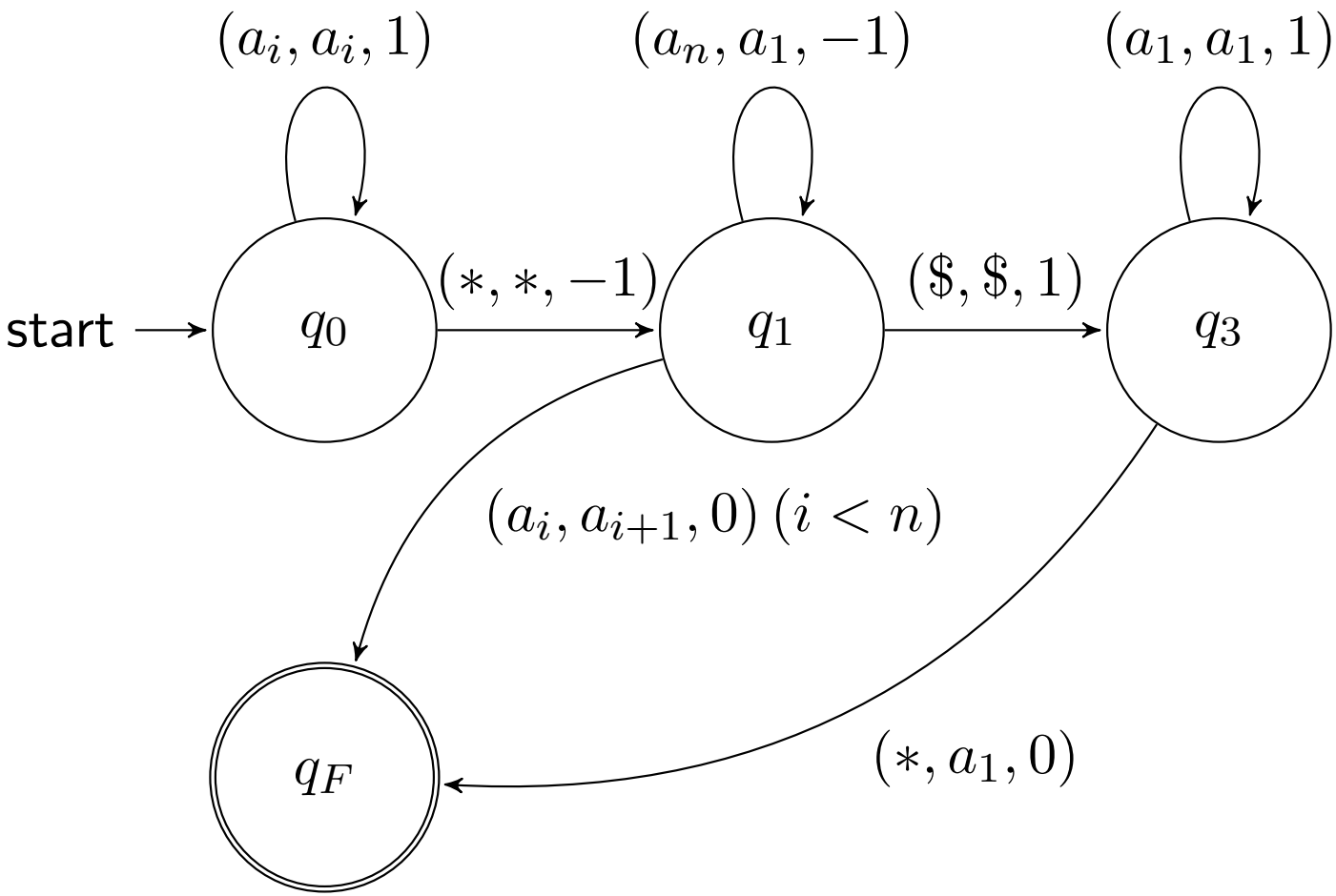
para todo $x \in \Sigma^*$.

Ejemplo

Dado un alfabeto $\Sigma = \{a_1, \ldots, a_n\}$ donde $a_1 < a_2 < \cdots < a_n$, definimos una máquina de Turing M que computa la función que a una palabra $x \in \Sigma^*$ le asigna la siguiente palabra según el orden lexicográfico.

Ejemplo

M:



La función tiempo de cálculo de una máquina de Turing determinista

Si M es una máquina de Turing determinista, Σ es el alfabeto de entrada de M, $x \in \Sigma^*$ y M para sobre la entrada x, definimos el tiempo de cálculo de M sobre x por:

$T(M, x) =$ número de pasos de cómputo que M realiza sobre x.

Si M es una máquina de Turing de parada segura, definimos la función tiempo de cálculo de M por $T_M : \mathbb{N} \to \mathbb{N}$ dada por

$$T_M(n) = \max\{T(M, x) : |x| = n\}.$$

Es el "peor" de los tiempos de todas las entradas cuya longitud es n.

Máquinas de Turing indeterministas

Una máquina de Turing indeterminista es una estructura $M = (K, \Sigma, \Gamma, \Delta, q_0, q_F)$ donde $K, \Sigma, \Gamma, q_0, q_F$ son como en la definición de máquina de Turing determinista y Δ es un subconjunto de $((K \setminus \{q_F\}) \times \Gamma) \times (K \times \Gamma \times \{1, -1, 0\})$.

Una configuración de M se define como en el caso determinista.

Los cómputos en una máquina de Turing indeterminista M se realizan por medio del conjunto Δ de transiciones de la máquina. Sin embargo, en este caso los cómputos de M no están unívocamente determinados, ya que para una entrada que suministremos a M, tendremos en general un árbol de cómputos, en el que las ramas del árbol representan los posibles cómputos para la entrada.

La función tiempo de cálculo de una máquina de Turing indeterminista

Sea M una máquina de Turing indeterminista. Sea Σ el alfabeto de entrada de M. Decimos que M es de parada segura, si para todo $x \in \Sigma^*$, se tiene que todas las ramas del árbol de cómputos asociado a x son finitas.

Si M es una máquina de Turing indeterminista de parada segura, definimos la función tiempo de cálculo de M como la función $T_M : \mathbb{N} \to \mathbb{N}$ dada por:

$T_M(n) =$ el máximo número de nodos de una rama del árbol de cómputos de una entrada de longitud n.

Lenguaje asociado a una máquina de Turing indeterminista

Sea M una máquina de Turing indeterminista. Sea Σ el alfabeto de entrada de M. Una palabra $x \in \Sigma^*$ es reconocida por M, si el árbol de cómputos asociado a x tiene una hoja que contiene q_F. Definimos entonces

$$L(M) = \{x \in \Sigma^* : x \text{ es reconocida por } M\}.$$

La tesis de Church-Turing

Afirma que todo algoritmo se puede representar por una máquina de Turing.

Razones que avalan la tesis de Church-Turing:

(1) Se ha demostrado que todas las formalizaciones matemáticas que se han hecho del concepto de algoritmo son equivalentes al modelo de la máquina de Turing.

(2) No se ha podido demostrar ningún teorema que exprese una limitación en la capacidad de cómputo de las máquinas de Turing.

(3) No se ha podido encontrar ningún algoritmo que no se pueda representar por una máquina de Turing.

Los órdenes de complejidad

Denotamos por $\mathbb{R}^*$ al conjunto de los números reales no negativos. Si $h : \mathbb{N} \to \mathbb{R}^*$, definimos

$O(h) = \{f : \mathbb{N} \to \mathbb{R}^* :$ existen números naturales c y n_0 tales que para todo $n \geq n_0$, $f(n) \leq c \cdot h(n)\}$.

Definición de las clases principales de complejidad

Si $f : \mathbb{N} \to \mathbb{R}^*$, definimos:

TIME$(f) = \{L$: existe una máquina de Turing determinista M de parada segura tal que $L(M) = L$ y $T_M \in O(f)\}$,

NTIME$(f) = \{L$: existe una máquina de Turing indeterminista M de parada segura tal que $L(M) = L$ y $T_M \in O(f)\}$.

Definimos ahora:

Definición de las clases principales de complejidad

$$P = \bigcup\{\mathsf{TIME}(n^k) : k \in \mathbb{N}\},$$

$$NP = \bigcup\{\mathsf{NTIME}(n^k) : k \in \mathbb{N}\},$$

$$EXP = \bigcup\{\mathsf{TIME}(2^{n^k}) : k \in \mathbb{N}\}.$$

En la clase P se encuentran los problemas que pueden ser resueltos por un programa en un tiempo "razonable".

El problema central

Tenemos que $P \subseteq NP \subseteq EXP$. Sin embargo, está abierto el siguiente problema:

Problema

¿ Es cierto que $P = NP$?

Este problema tiene un gran interés, porque hay muchos problemas en la clase NP, que no se sabe si están en la clase P.

Ejemplos de problemas en la clase P

(1) El problema de determinar, dados un grafo dirigido G y dos nodos x, y de G, si hay un camino en G que conecta x con y.

(2) El problema de determinar si un número natural es primo.

(3) El problema de determinar si dos números naturales son primos entre sí.

(4) Dado un autómata finito M, el problema de determinar si una palabra x pertenece a $L(M)$.

(5) Dada una gramática incontextual G, el problema de determinar si una palabra x pertenece a $L(G)$.

Ejemplos de problemas en la clase NP

(1) El problema SAT de determinar si una fórmula $\Phi(X_1, \ldots, X_n)$ del cálculo proposicional es satisfactible, es decir, si existe una asignación de los valores de verdad $x = (x_1, \ldots x_n) \in \{0, 1\}^n$ tal que $\Phi(x) = 1$.

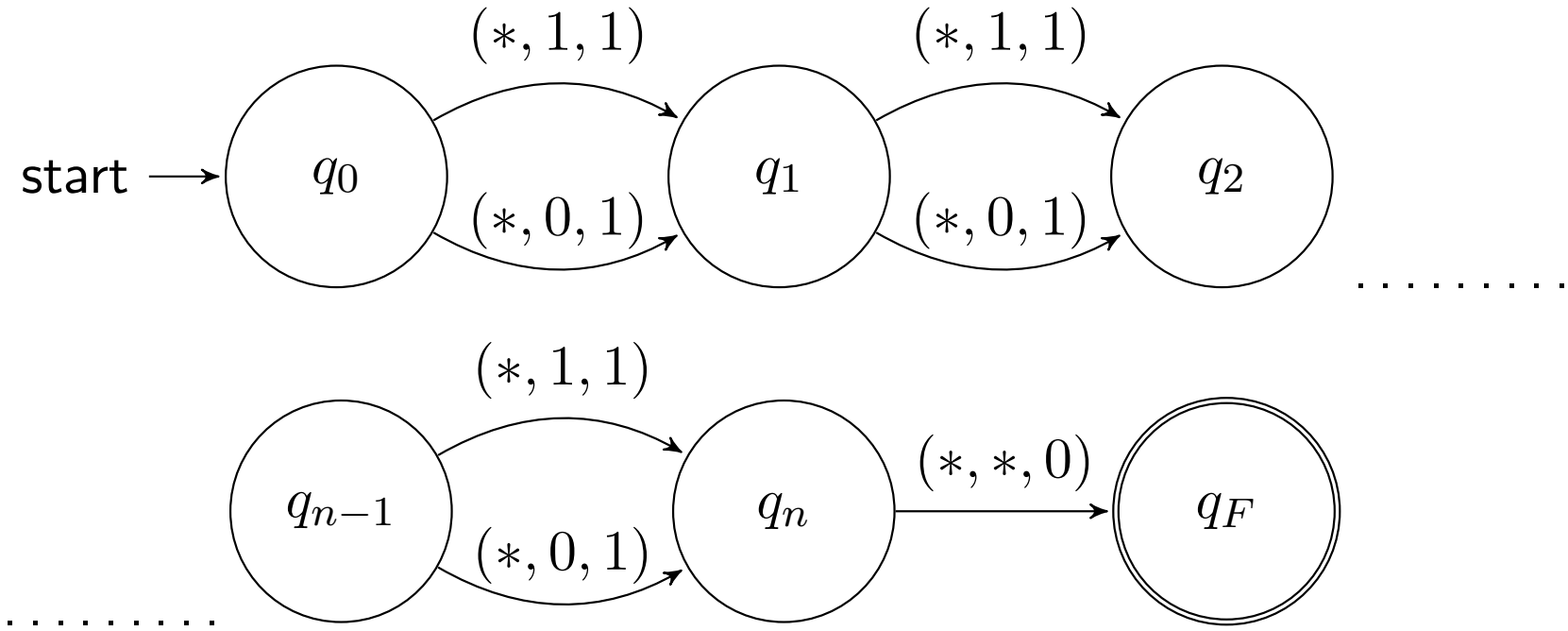
Para comprobar que SAT $\in NP$, utilizamos el siguiente algoritmo:

1. Generamos una asignación de los valores de verdad de manera indeterminista.

2. Verificamos si la interpretación generada satisface la fórmula.

La etapa 2 se realiza de manera determinista. Y para la etapa 1, podemos utilizar la siguiente máquina de Turing indeterminista:

Ejemplos de problemas en la clase NP



Ejemplos de problemas en la clase NP

(2) El problema de colorear un mapa con 4 colores de manera que no haya dos países vecinos que tengan el mismo color.

(3) Dado un conjunto finito $A = \{a_1, \ldots, a_n\}$ de números naturales y un número natural k, determinar si existe un subconjunto $B = \{b_1, \ldots, b_m\}$ de A tal que $b_1 + \cdots + b_m = k$.

(4) Dado un grafo dirigido G, determinar si hay un camino en G que pasa por todos los nodos exactamente una vez.

(5) Dados dos grafos G_1 y G_2, determinar si G_1 y G_2 son isomorfos.

Reducibilidad de lenguajes

Sean A, B lenguajes sobre un alfabeto Σ. Decimos que A es polinómicamente reducible a B, en símbolos $A \leq_P B$, si existe una función $f : \Sigma^* \to \Sigma^*$ total y computable en tiempo polinómico tal que para todo $x \in \Sigma^*$:

$x \in A$ si y sólo si $f(x) \in B$.

Proposición

(a) Si $A \leq_P B$ y $B \in P$, entonces $A \in P$.

(b) Si $A \leq_P B$ y $B \in NP$, entonces $A \in NP$.

Lenguajes NP-completos

Sea $L \in NP$. Se dice que L es NP-completo, si $A \leq_P L$ para todo lenguaje $A \in NP$.

Proposición

Si existe un lenguaje L NP-completo tal que $L \in P$, entonces $P = NP$.

Sin embargo, no es sabido si existe un lenguaje NP-completo en la clase P.

Lenguajes NP-completos

Teorema de Cook-Levin (1971)

SAT es NP-completo.

La siguiente proposición es fácil de comprobar.

Proposición

Si $A, B \in NP$, A es NP-completo y $A \leq_P B$, entonces B es NP-completo.

Utilizando entonces el teorema de Cook-Levin y la anterior proposición, se puede demostrar que muchos de los lenguajes NP son NP-completos.

Otros problemas abiertos en complejidad

Es sabido que $P \subsetneq EXP$. Sin embargo, el siguiente problema está abierto.

Problema

¿ Es cierto que $NP = EXP$?

Se tiene que el lenguaje complementario de un lenguaje en la clase P también está en la clase P. Sin embargo, no es sabido si el lenguaje complementario de un lenguaje en la clase NP está también en NP.

Definimos la clase $co - NP = \{L \subseteq \Sigma^* : \Sigma^* \setminus L \in NP\}$.

Otros problemas abiertos en complejidad

Proposición

$NP \neq co - NP$ implica $P \neq NP$.

Problema

¿ Es cierto que $NP = co - NP$?