Máquinas de Turing y Teoría de Decidibilidad

Pedro O. Pérez M., PhD.

Implementación de métodos computacionales
Tecnológico de Monterrey

pperezm@tec.mx

02-2023

Contenido I

1. Problemas que las computadoras no pueden resolver

2. Máquinas de Turing
   - ¿Porqué estudiamos este tipo de problemas?
   - Notación de una máquina de Turing
   - Ejemplos

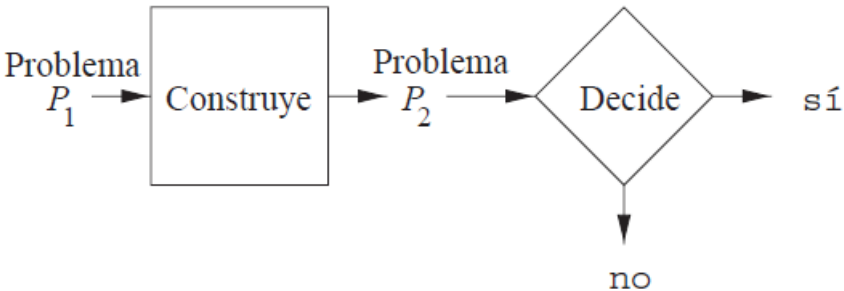
3 Extensiones de Máquinas de Turing
- Técnicas de programación para las MT's
  - Almacenamiento de estado
  - Pistas múltiples
  - Subrutinas
- Extensiones de la máquina de Turing básica
  - Máquina de Turing de varias cintas
  - Tiempo de ejecución
- Máquinas de Turing y computadoras
  - Simulación de una máquina de Turing mediante una computadora
  - Simulación de una computadora mediante máquina de Turing

4 Problemas intratables
- Introducción
- Lenguaje no recursivamente enumerable
- Clases P
  - Problemas resolubles en tiempo polinómico
- Clases NP
  - Tiempo polinómico no determinista
  - Problemas NP-completos
  - Otros problemas $NP - Completos$

Problemas que las computadoras no pueden resolver

- Recuerdan "The Halting Problem: The Unsolvable Problem". Con este problema, Dr. Turing demostró que ningún programa H puede informar si un determinado programa P con entrada I se detiene o se ejecuta por siempre.
- Pero, ¿cómo podemos demostrar que un nuevo problema, P_2 es indecidible?¿Tenemos que usar una estrategia similar?

Basta con demostrar que si pudiéramos resolver el nuevo problema, P_2, entonces utilizaríamos dicha solución para resolver un problema que ya sabemos que es indecidible, P_1. Esta técnica se conoce como *reducción* de P_1 a P_2.


Problema
P_1
Construye
Problema
P_2
Decide
sí
no

El propósito de la teoría de los problemas indecidibles no es sólo establecer la existencia de tales problemas sino proporcionar también una guía a los desarrolladores sobre lo que se puede o no conseguir a través de la programación.

Existe otra razón.:

- La teoría también tiene un gran impacto práctico al tratar problemas que aunque sean decidibles, requieren mucho tiempo para se resueltos. Estos problemas, conocidos como "problemas intratables", suelen plantear una mayor dificultad al programador y el diseñador de sistemas que los problemas indecibles.
- Los problemas intratables son más comunes. Y, a menudo dan lugar a pequeñas modificaciones de los requisitos o soluciones heurísticas.

Veamos porqué decimos esto. Revisemos un segmentos del vídeo: “P vs. NP - The Biggest Unsolved Problem in Computer Science”.

Entonces, necesitamos herramientas que nos permitan determinar cuestiones acerca de la indecidibilidad o intratabilidad todos los días. Como resultado, necesitamos reconstruir nuestra teoría sobre indecidibilidad, basándonos es un modelo de computadora muy simple: la máquina de Turing.

Con la notación de la máquina de Turing, demostraremos que ciertos problemas, que aparentemente no están relacionados con la programación, son indecidibles. Por ejemplo, demostraremos que el "problema de la correspondencia de Post", una cuestión simple que implica dos listas de cadenas, es indecidible y que este problema facilita la demostración de que algunas cuestiones acerca de las gramáticas, como por ejemplo la ambigüedad, son indecidibles.

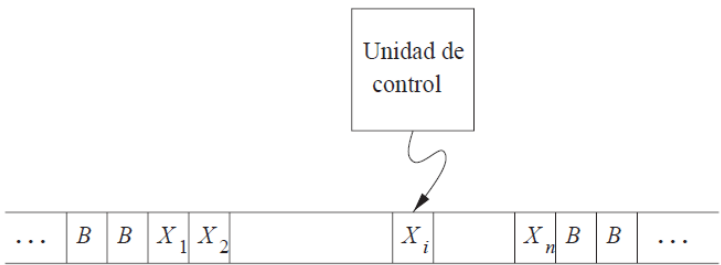
El intento de decidir todas las cuestiones matemáticas

- A finales del siglo XX, el matemático D. Hilbert se preguntó si era posible encontrar un algoritmo para determinar la verdad o falsedad de cualquier proposición matemática. En particular, se preguntó si existía una forma de determinar si cualquier fórmula del cálculo de predicados de primer orden, aplicada a enteros, era verdadera.
- Sin embargo, en 1931, K. Gödel publicó su famoso teorema de la incompletitud. Construyó una fórmula para el cálculo de predicados aplicada a los enteros, que afirmaba que la propia fórmula no podía ser ni demostrada ni refutada dentro del cálculo de predicados.

El cálculo de predicados no era la única idea que los matemáticos tenían para "cualquier computación posible". De hecho, el cálculo de predicados, al ser declarativo más que computacional, entraba en competencia con una variedad de notaciones, incluyendo las "funciones recursivas parciales", una notación similar a un lenguaje de programación, y otras notaciones similares. En 1936, A. M. Turing propuso la máquina de Turing como modelo de "cualquier posible computación".

Lo interesante es que todas las propuestas serias de modelos de computación tienen el mismo potencial; es decir, calculan las mismas funciones o reconocen los mismos lenguajes. La suposición no demostrada de que cualquier forma general de computación no permite calcular sólo las funciones recursivas parciales (o, lo que es lo mismo, que las máquinas de Turing o las computadoras actuales pueden calcular) se conoce como *hipótesis de Church* (por el experto en lógica A. Church) o *tesis de Church*.

Básicamente, una máquina de Turing es un autómata finito que dispone de una única cinta de longitud infinita en la que se pueden leer y escribir datos. Una ventaja de la máquina de Turing sobre los problemas como representación de lo que se puede calcular es que la máquina de Turing es lo suficientemente simple como para que podemos representar su configuración de manera precisa, utilizando una notación sencilla muy simular a las descripciones de un autómata de pila.



- Inicialmente, la *entrada*, que es una cadena de símbolos de longitud finita elegidos del *alfabeto de entrada*, se coloca en la cinta. Las restantes casillas de la cinta, que se extiende infinitamente hacia la izquierda y la derecha, inicialmente almacenan un símbolo especial denominado *espacio en blanco*.
- Existe una *cabeza de la cinta* que siempre está situada en una de las casillas de la cinta. Se dice que la máquina de Turing *señala* dicha casilla. Inicialmente, la cabeza de la cinta está en la casilla más a la izquierda que contiene la entrada.
- Un *movimiento* de la máquina de Turing es una función del estado de la unidad de control y el símbolo de cinta al que señala la cabeza.
  - Cambiará de estado.
  - Escribirá un símbolo de cinta en la casilla que señala la cabeza.
  - Moverá la cabeza de la cinta hacia la izquierda o hacia la derecha.

La notación formal de una *máquina de Turing* (MT) es similar a la que hemos empleado para los autómatas finitos o los autómatas de pila. Describimos un MT mediante la siguiente séptula: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ cuyos componentes tienen el siguiente significado:

- Q: El conjunto finito de *estados* de la unidad de control.
- Σ: El conjunto finito de *símbolos de entrada*.
- Γ: El conjunto completo de *símbolos de cinta*; Σ siempre es un subconjunto de Γ.
- δ: La *función de transición*. Los argumentos $\delta(q, X)$ son un estado q y un símbolo X. La función regresa, si está definido, la tripleta (p, Y, D), donde p es el nuevo estado, Y es el símbolo de Γ que se escribe en la casilla y D es una *dirección* y puede ser L o R.
- q_0: El *estaod inicial*, un elemento de Q.
- B: El símbolo *espacio en blanco*.
- F: El conjunto de estados *de aceptación*.

Ejemplo 1

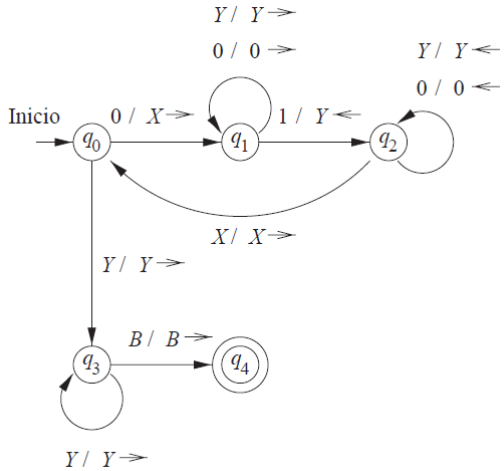
Vamos a diseñar una máquina de Turing y a ver cómo se comporta una entrada típica. La MT que vamos a construir aceptará el lenguaje $\{0^n1^n | n \geq 1\}$. Inicialmente, se proporciona a la cinta una secuencia infinita de ceros y unos, precedida y seguida por secuencias de infinitas de espacios en blanco.

Comenzando por el extremo izquierdo de la entrada, se cambia sucesivamente un 0 por una X y se mueve hacia la derecha pasando por encima de todos los ceros y las letras Y que ve, hasta encontrar un 1. Cambia el 1 por una Y y se mueve hacia la izquierda pasando sobre todas las letras Y y ceros hasta encontrar una X. En esta situación, busca un 0 colocado inmediatamente la derecha y, si lo encuentra, lo cambia por una X y repite el proceso, cambiando el 1 correspondiente por una Y.

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q4\})$$

	Símbolo				
Estado	0	1	X	Y	B
q_0	(q_1, X, R)	–	–	(q_3, Y, R)	–
q_1	$(q_1, 0, R)$	(q_2, Y, L)	–	(q_1, Y, R)	–
q_2	$(q_2, 0, L)$	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	–

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q4\})$$



Ejemplo 2

Ahora, veamos cómo una máquina de Turing puede calcular la función $\dot{-}$, que recibe el nombre de *sustracción propia* y se define mediante $m \dot{-} n = \text{máx}(m - n, 0)$. Es decir, $m \dot{-} n$ es $m - n$ si $m \geq n$ y 0 si $m < n$.

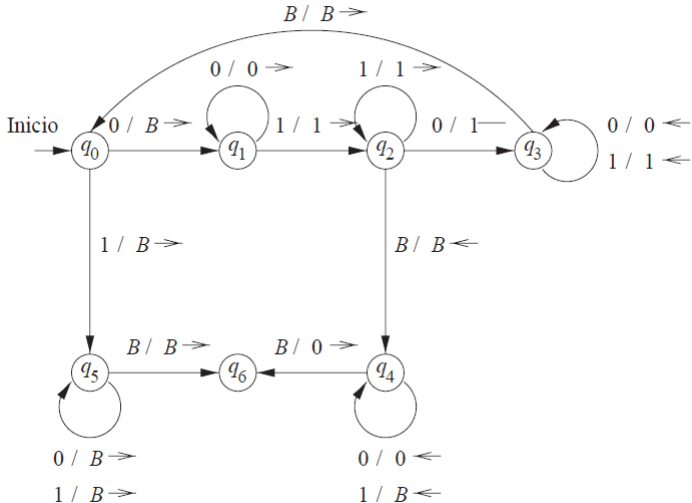
La MT empezará a trabajar con una cinta que conste de $0^m 1 0^n$ rodeada de espacios en blanco. La MT se parará cuando el contenido de la cinta sea $0^{m \dot{-} n}$ rodeado de espacios en blanco.

La MT encuentra repetidamente el 0 más a la izquierda que queda y lo reemplaza por un espacio en blanco. A continuación, se mueve hacia la derecha, en busca de un 1. Después de encontrar un 1, continúa moviéndose hacia la derecha hasta que llega a un 0, que sustituye por un 1. M vuelve entonces a moverse hacia la izquierda, buscando el 0 más a la izquierda, el cual identifica cuando encuentra un espacio en blanco a su izquierda, y luego se mueve una casilla hacia la derecha.

Este proceso se termina si:

- Buscando hacia la derecha un 0, encuentra un espacio en blanco. Esto significa que los n ceros de 0^m10^n han sido todos ellos sustituidos por unos, y $n+1$ de los m ceros han sido sustituidos por B. Se reemplazan los $n+1$ unos por un 0 y n símbolos B, dejando $m-n$ ceros en la cinta. Dado que $m \geq n$ en este caso, $m-n = m \dot{-} n$.
- Si al comenzar un ciclo, la MT no puede encontrar un 0 para sustituirlo por un espacio en blanco, porque los m primeros ceros ya han sido sustituidos por símbolo B. Entonces $n \geq m$, por lo que $m \dot{-} n = 0$. Se reemplazan todos los unos y ceros restantes por símbolos B y termina con una cinta completamente en blanco.

Estado	Símbolo 0	1	B
q_0	(q_1,B,R)	(q_5,B,R)	—
q_1	$(q_1,0,R)$	$(q_2,1,R)$	—
q_2	$(q_3,1,L)$	$(q_2,1,R)$	(q_4,B,L)
q_3	$(q_3,0,L)$	$(q_3,1,L)$	(q_0,B,R)
q_4	$(q_4,0,L)$	(q_4,B,L)	$(q_6,0,R)$
q_5	(q_5,B,R)	(q_5,B,R)	(q_6,B,R)
q_6	—	—	—


B / B
0 / 0
1 / 1
Inicio
0 / B
1 / 1
0 / 1
0 / 0
1 / 1
q0
q1
q2
q3
1 / B
B / B
B / B
B / 0
q5
q6
q4
0 / B
1 / B
0 / 0
1 / B

Ejercicio 1

Diseño una máquina de Turing que reconozca el lenguaje: $\{a^n b^n c^n | n \geq 1\}$.

Podemos usar una máquina de Turing para hacer cálculos de una manera muy diferente a como lo hace una computadora convencional. En particular, veremos que las máquinas de Turing pueden realizar, sobre otras máquinas de Turing, el ´tipo de cálculos que puede realizar mediante un programa al examen otros problemas.

Podemos emplear la unida de control no sólo para representar una posición en el "programa" de la máquina de Turing, sino también para almacenar una cantidad finita de estados.

Por ejemplo, diseñemos una máquina de Turing que recuerda en su unidad de control el primer símbolo (0 o 1) que vea y compruebe que no aparece en ningún otro lugar de su entrada. Por tanto, la MT acepta el lenguaje $\{01^*|10^*\}$.

Para resolver este problema, consideraremos que la unidad de control no sólo consta de un estado de "control" (estado actual), sino que, además, puede almacenar tres elementos de datos A, B y C. Es decir, podríamos ver el estado como la tripleta $[q, A, B, C]$.

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0, B], \{[q_1, B]\})$$

El conjunto de estado Q es $\{q_o, q_1\} \times \{0,1,B\}$. Es decir, puede pensarse en los estados como en pares de dos componentes:

a) Una parte de control, q_0 o q_1, que recuerda lo que está haciendo la MT. El estado de control q_0 indica que todavía no se ha leído el primer símbolo, mientras que q_1 indica que ya se ha leído, y está comprobando que no aparece en ningún otro lugar, moviéndose hacia la derecha con la esperanza de alcanzar una casilla con un espacio en blanco.

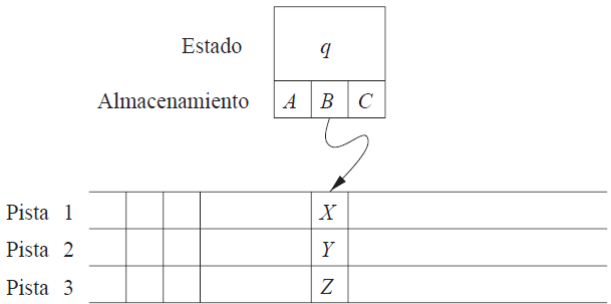
b) Una parte de datos, recuerda el primer símbolo que se ha visto, que tiene que ser 0 o 1. El símbolo B en este componente indica que no se ha leído ningún símbolo.

La función de transición δ es la siguiente:

1. $\delta([q_0, B], a) = ([q_1, a], a, R)$ para $a = 0$ o $a = 1$. Inicialmente, q_0 es el estado de control y la parte de los datos es B. El símbolo que está siendo señalada se copia al segundo componente del estado y la MT se mueve hacia la derecha entrando en el estado de control q_1.
2. $\delta([q_1, a], \bar{a}) = ([q_1, a], \bar{a}, R)$ donde $\bar{a}$ es el "complementario" de a, es decir, 0 si $a = 1$ y 1 si $a = 0$. En el estado q_1, la MT salta por encima de cada símbolo 0 o 1 que es diferente del que se tiene almacenado en su estado y continúa moviéndose hacia la derecha.
3. $\delta([q_1, a], B) = ([q_1, B], B, R)$ para $a = 0$ o $a = 1$. Si llega al primer espacio en blanco, entra en el estado de aceptación $[q_1, B]$.

Observa que la MT no tiene definida una transición $\delta([q_1, a], a)$ para $a = 0$ o $a = 1$. Es decir, si encuentra una segunda aparición del símbolo almacenado inicialmente en su unidad de control, se para sin entrar en el estado de aceptación.

Otro "truco" práctico es el pensar que la cinta de una máquina de Turing está compuesta por varios pistas. Cada pista puede almancenar un símbolo, y el alfabeto de la cinta consta de tuplas, con una componente para cada "pista".



Un uso común de las pistas múltiples consiste en considerar una pista que almacena los datos y una segunda pista que almacena una marca. Podemos marcar cada uno de los símbolos como "utilizado", o podemos seguir la pista de un número pequeño de posiciones dentro de los datos marcando sólo dichas posiciones. Por ejemplo, utilizaremos explícitamente una segunda pista para reconocer el lenguaje independiente de contexto:

$$L_{wcw} = \{wcw | w \in (0|1)^+\}$$

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_9, B]\})$$

donde:

- Q: El conjunto de estados es $\{q_1, q_2, ..., q_9\} \times \{0, 1\}$, es decir, pares que constan de un estado de control q_i y una componente de datos, 0 o 1. De nuevo, utilizaremos la técnica de almacenamiento en la unidad de control, permitiendo que el estado recuerde y símbolo de entrada 0 o 1.
- Γ: El conjunto de símbolos de cinta es $\{B, *\} \times \{0, 1, c, B\}$. El primer componente, o pista, puede ser o un espacio en blanco o un símbolo "marcado" (que se ha revisado), y se representan respectivamente mediante los símbolos B y $*$. Utilizaremos $*$ para marcar como revisados los símbolos del primer y segundo grupos de ceros y unos, confirmando así que la cadena a la izquierda del marcador central c es la misma que la cadena situada a su derecha. El segundo componente del símbolo de cinta es el propio símbolo de cinta.
- Σ: Los símbolo de entrada son $[B, 0]$ y $[B, 1]$.

- δ: La función de transición se define de acuerdo con las siguientes reglas, en las que a y b pueden tomar los valores 0 o 1.
  1. $\delta([q_1, B], [B, a]) = ([q_2, a], [*, a], R)$. En el estado inicial, la MT lee el símbolo a, lo almacena en su unidad de control, para al estado de control q_2, "marca como revisado" el símbolo que acaba de leer y se mueve hacia la derecha.
  2. $\delta([q_2, a], [B, b]) = ([q_2, a], [B, b], R)$. La MT se mueve hacia la derecha en busca del símbolo c.
  3. $\delta([q_2, a], [B, c]) = ([q_3, a], [B, c], R)$. Cuando la MT encuentra el símbolo c, continúa moviéndose hacia la derecha, pero cambia al estado de control q_3.
  4. $\delta([q_3, a], [*, b]) = ([q_3, a], [*, b], R)$. En el estado q_3, pasa por encima de todos los símbolos que ya se han revisado.
  5. $\delta([q_3, a], [B, a]) = ([q_4, B], [*, a], L)$. Si el primer símbolo no revisado que encuentra es el mismo que el símbolo que se encuentra en su unidad de control, marca este símbolo como revisado, ya que está emparejado con el símbolo correspondiente del primer bloque de ceros y unos.

6. $\delta([q_4, B], [*, a]) = ([q_4, B], [*, a], L)$. Se mueve hacia la derecha pasando sobre todos los símbolos revisados.
7. $\delta([q_4, B], [B, c]) = ([q_5, B], [B, a], L)$. Cuando encuentra el símbolo c, pasa al estado q_5 y continúa moviéndose hacia la izquierda.En el estado q_5, se tiene que tomar una decisión, que depende de si el símbolo inmediatamente a la izquierda del símbolo está marcado o no como revisado. Si está revisado, entonces quiere decir que ya hemos considerado el primer bloque completo de ceros y unos (a la izquierda de c). Tenemos que asegurarnos que todos los ceros y unos situados a la derecha del símbolo c también están revisados, y si no queda ningún símbolo no revisado a la derecha de c, aceptar. Si el símbolo inmediatamente a la izquierda de la c no está revisado, buscaremos el símbolo no revisado más a la izquierda, lo leeremos e iniciaremos el ciclo que comienza en el estado q_1

8. $\delta([q_5, B], [B, a]) = ([q_6, B], [B, a], L)$. Esta rama cubr e el caso en que el símbolo a la izquierda no está revisado. Se pasa al estado q_6 y continúa moviéndose hacia la izquierda, en busca de un símbolo revisado.
9. $\delta([q_6, B], [B, a]) = ([q_6, B], [B, a], L)$. Siempre y cuando existan símbolos no revisados, permanece en el estado q_6 y continúa moviéndose hacia la izquierda.
10. $\delta([q_6, B], [*, a]) = ([q_1, B], [*, a], R)$. Cuando se encuentra el símbolo revisado, la MT entra en el estado q_1 y se mueve hacia la derecha hasta llegar al primer símbolo no revisado.
11. $\delta([q_5, B], [*, a]) = ([q_7, B], [*, a], R)$. Ahora, elegimos la rama que sale del estado q_5, que corresponde al caso en que se mueva hacia la izquierda de C y se encuentra un símbolo revisado. Comenzamos, nuevamente, a movernos hacia la derecha pasando al estado q_7.
12. $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$. En el estado q_7 seguramente veremos el símbolo c. La MT entrará en el estado q_8 y continúa moviéndose a la derecha.

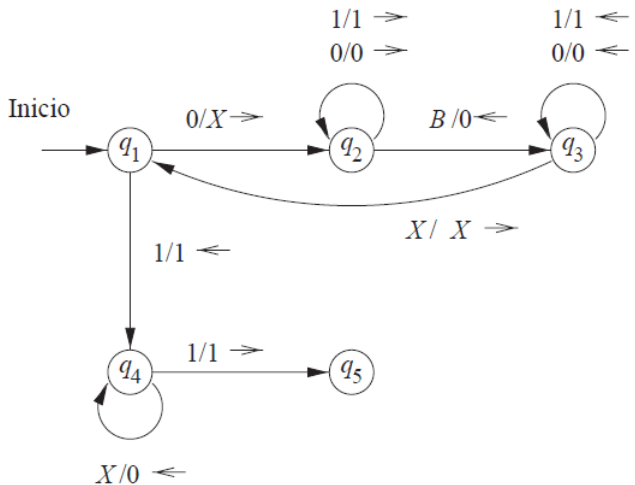
11. $\delta([q_5, B], [*, a]) = ([q_7, B], [*, a], R)$. Ahora, elegimos la rama que sale del estado q_5, que corresponde al caso en que se mueva hacia la izquierda de C y se encuentra un símbolo revisado. Comenzamos, nuevamente, a movernos hacia la derecha pasando al estado q_7.
12. $\delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R)$. En el estado q_7 seguramente veremos el símbolo c. La MT entrará en el estado q_8 y continúa moviéndose a la derecha.
13. $\delta([q_8, B], [*, a]) = ([q_8, B], [*, a], R)$. En el estado q_8, se mueve hacia la derecha, saltando sobre todos los ceros y unos revisados que ya ha encontrado.
14. $\delta([q_8, B], [B, B]) = ([q_9, B], [B, B], R)$. Estado de aceptación.

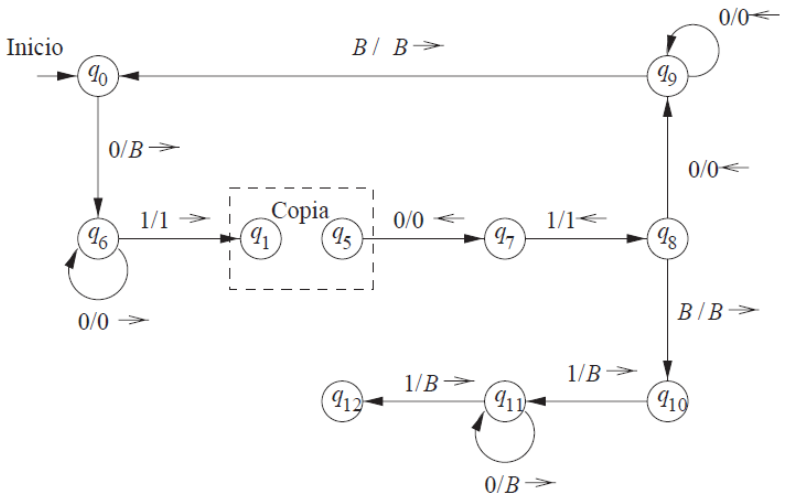
Subrutinas

- Una subrutina de una máquina de Turing es un conjunto de estados que realizar un determinado procesos útil. Este conjunto de estados incluye un estado inicial y otro estado en el que, temporalmente, no existen movimientos, y que sirve como estado de "retorno".
- La "llamada" a una subrutina se produce cuando existe una transición a su estado inicial.
- Dado que la MT no tiene ningún mecanismo que le permita recordar una "dirección de retorno"; es decir, un estado al que volver una vez que haya terminado, si la llamada a una MT que actúa como subrutina tiene que hacerse desde varios estados, se pueden hacer copias de la subrutina utilizando un nuevo conjunto de estados para cada copia.

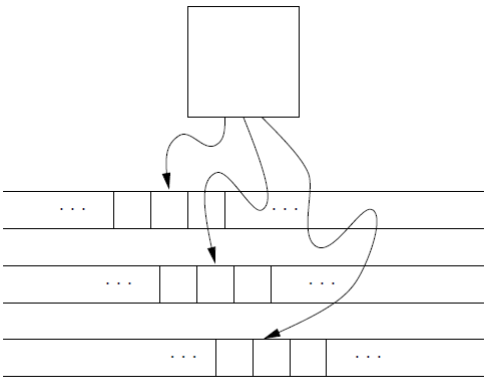
Vamos a diseñar una MT para implementar la función "multiplicación". La MT tendrá inicialmente en la cinta 0^m10^n y al final deberá tener 0^{mn}. Un esquema de la estrategia es la siguiente:

1. En general, la cinta contendrá una cadena sin ningún espacio en blando de la forma $0^m10^n10^{kn}$ para un cierto k.
2. En el paso básico, cambiamos un 0 del primer grupo por un espacio en blanco B y añadimos n ceros al último grupo, obteniendo una cadena de la forma $0^{m-1}10^n10^{(k+1)n}$.
3. Como resultado, se habrá copiado m veces el grupo de n ceros al final, una por cada vez que hayamos cambiado un 0 por un espacio en blanco B. Cuando todo el primer grupo de ceros se haya cambiado por espacios en blanco, habrá mn ceros en el último grupo.
4. El último paso consiste en cambia la cadena 10^n1 del principio por espacios en blanco.


1/1 →
0/0 →
1/1 ←
0/0 ←
Inicio
0/X →
B /0 ←
q_1
q_2
q_3
X/ X →
1/1 ←
1/1 →
q_4
q_5
X/0 ←


Inicio
q_0
$B/\ B$
q_9
0/0
0/B
0/0
Copia
q_6
1/1
q_1
q_5
0/0
q_7
1/1
q_8
0/0
B / B
q_{12}
1/B
q_{11}
1/B
q_{10}
0/B

Máquina de Turing de varias cintas



El dispositivo tiene una unidad de control (estado) y un número finito de cintas. Cada cinta está dividida en casillas, y cada casilla puede contener cualquier símbolo del alfabeto finito. Al igual que en la MT de una sola cinta, el conjunto de símbolos de la cinta incluye el espacio en blanco y también dispone de un subconjunto de símbolos de entrada, al que no pertenece el espacio en blanco. Con este tipo de máquinas puede simularse computadoras reales.

El conjunto de estados incluye un estado inicial y varios estados de aceptación. Inicialmente:

1. La entrada, una secuencia finita de símbolos de entrada, se coloca en la primera cinta.
2. Todas las casillas de las demás cintas contienen espacios en blanco.
3. La unidad de control se encuentra en el estado inicial.
4. La cabeza de la primera cinta apunta al extremo izquierdo de la entrada.
5. Las cabezas de las restantes cintas apunta a una casilla arbitraria.

Un movimiento de la MT de varias cintas depende del estado y del símbolo señalado por las cabezas de cada una de las cintas. En un movimiento, esta MT hace lo siguiente:

1. La unidad de control entra en un nuevo estado, que puede ser el mismo que en el que se encontraba anteriormente.
2. En cada cinta, se escribe un nuevo símbolo de cinta en la casilla señalada por la cabeza. Estos símbolos pueden ser los mismos que estaban escritos anteriormente.
3. Cada una de las cabezas de las cintas realiza un movimiento, que puede ser hacia la izquierda, hacia la derecha o estacionario. Las cabezas se mueven de manera independiente, por lo que pueden moverse en direcciones diferentes y alguna puede no moverse en absoluto.

Tiempo de ejecución

El *tiempo de ejecución* de la máquina de Turing M para la entrada w es el número de pasos que M realiza antes de pararse. Si M no se para con la entrada w, entonces el tiempo de ejecución de M es w es infinito.

Simulación de una máquina de Turing mediante una computadora

Dada una MT concreta M, debemos escribir un programa que actúe como M. Un aspecto de M es su unidad de control. Dado que sólo existen un número finito de estados y un número finito de reglas de transición, nuestro programa puede codificar los estados como cadenas de caracteres y utilizar una tabla de transiciones, que consultará para determinar cada movimiento. Del mismo modo, los símbolos de cinta pueden codificarse como cadenas de caracteres de longitud fija, ya que el número de estos símbolos es finito.

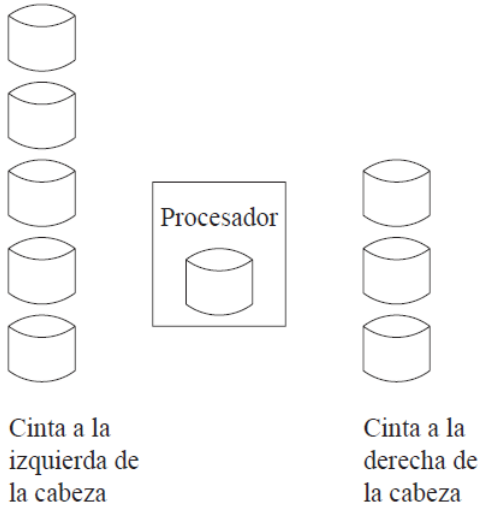


Figura 8.21. Simulación de una máquina de Turing mediante una computadora común.

Simulación de una computadora mediante máquina de Turing

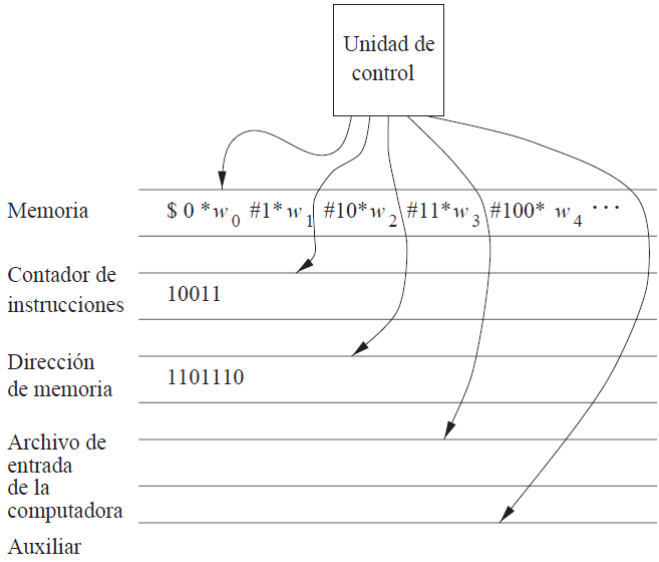
Para simular una computadora mediante máquina de Turing, partiremos de un modelo realista aunque informal del funcionamiento de una computadora típica:

a) En primer lugar, suponemos que el almacenamiento de una computadora consta de una secuencia infinitamente larga de palabras, cada una con una dirección asociada.

b) Suponemos que el programa de la computadora se almacena en algunas de las palabras de memoria. Cada una de estas palabras representan una instrucción, como en el lenguaje máquina o ensamblador de una computadora típica.

c) Suponemos que cada instrucción implica un número limitado (finito) de palabras y que cada instrucción modifica el valor de como máximo una palabra.

d) Una computadora típica dispone de registros, que son palabras de memoria a las que se puede acceder especialmente rápido.


Unidad de control
Memoria
$ 0 *w0 #1*w1 #10*w2 #11*w3 #100* w4 ···
Contador de instrucciones
10011
Dirección de memoria
1101110
Archivo de entrada de la computadora
Auxiliar

- Una de las malas interpretaciones que solemos tener con respecto al tema de indecidibilidad es que, a medida que pase el tiempo, las prestaciones de las computadoras seguirán creciendo indefinidamente en aspectos como el tamaño del espacio de direccionamiento, el tamaño de la memoria principal, etc.
- Si nos centramos en la máquina de Turing, en la que no existen estas limitaciones, podremos entender mejor la idea básica de que algún dispositivo de computación será capaz de hacerlo, si no hoy día, sí en un futuro.
- Entonces, dividiremos los problemas que puede resolver una máquina de Turing en dos categorías: aquellos que tienen un algoritmo (es decir, una maquina de Turing que se detiene en función de si acepta o no su entrada), y aquéllos que sólo son resueltos por máquinas de Turing que pueden seguir ejecutándose para entradas que no aceptan.

Decimos que un lenguaje L es *recursivamente enumerable* (RE) si $L = L(M)$ para alguna máquina de Turing M, tel que:

1. Si w pertenece a L, M la acepta (y, por lo tanto, para).
2. Si w no pertenece a L, entonces M eventualmente para, sin importar si entra o no a un estado de aceptación.

Una MT de este tipo corresponde a nuestra noción informal de un "algoritmo", una secuencia de pasos bien definida que siempre termina y produce una respuesta. Si pensamos en el lenguaje L como un "problema", como será el caso con frecuencia, entonces el problema L se llama *decidible* si es un lenguaje recursivo, e *indecidible* si no lo es.

Nuestro objetivo a largo plazo es demostrar la indecibilidad del lenguaje que consta de pares (M, w) tales que:

1. M es una máquina de Turing (codificada adecuadamente en binario) con el alfabeto de entrada $\{0, 1\}$.
2. w es una cadena de ceros y unos.
3. M acepta la entrada w.

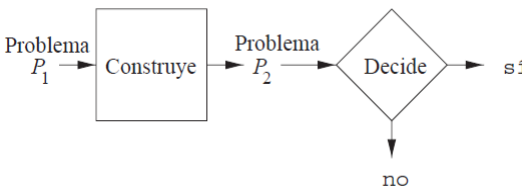
Si este problema con entradas restringidas al alfabeto binario es indecidible, entonces con toda probabilidad el problema más general, en el que la MT puede utilizar cualquier alfabeto, será indecidible.

- Los problemas resolubles en un tiempo polinómico en una computadora típica son exactamente los mismos que los problemas resolubles en un tiempo polinómico en una máquina de Turing.
- La experiencia ha demostrado que existe una línea divisoria fundamental entre los problemas que se pueden resolver en un tiempo polinómico y los que requieren un tiempo exponencial o mayor. Los problemas prácticos que necesitan un tiempo polinómico casi siempre pueden solucionarse en un tiempo que podemos considerar tolerable, mientras que aquellos que precisan un tiempo exponencial, generalmente, no pueden resolverse excepto para casos sencillos.

Problemas resolubles en tiempo polinómico

- Se dice que una máquina de Turing M tiene complejidad temporal $T(n)$ [o tiene un"tiempo de ejecución $T(n)$"] si siempre que M recibe una entrada w de longitud n, M se para después de realizar como máximo $T(n)$ movimientos, independientemente de si la acepta o no.
- Decimos que un lenguaje L pertenece a la clase P si existe alguna $T(n)$ polinómica tal que $L = L(M)$ para alguna máquina de Turing determinista M de complejidad temporal $T(n)$.

Determinar el *árbol de expansión de peso mínimo* (MST, Minimum Spanning Tree) de un grafo. Un *árbol de expansión* es un subconjunto de los arcos tales que todos los nodos están conectados a través de dichos arcos, sin que exista ningún ciclo.



Bueno, pues existe un algoritmo "ávido", denominado *algoritmo de Kruskal*, que permite determinar el árbol de expansión de peso mínimo:

1. Inicialmente, considera que cada nodo es el único elemento de un conjunto. Es decir, cada uno de los nodos están en conjuntos disjuntos.
2. Considera el arco de menor costo que aún no se haya tenido en cuenta. Si este arco conecta dos nodos que actualmente están en conjuntos disjuntos, entonces:
   a) Selecciona dicho arco para el árbol de recubrimiento y,
   b) Une los conjuntos de los nodos en un conjunto único.
3. Continúa seleccionando arcos hasta que se hayan tenidos todos ellos en cuenta o el número de para el árbol de recubrimiento sea uno menos que el número de nodos.

Es posible implementar este algoritmo (utilizando una computadora, no una máquina de Turing) sobre un grafo de m nodos y e arcos en un tiempo $O(m + e \log e)$. La implementación más sencilla y fácil de seguir lo hace en e iteraciones. Una tabla proporciona la componente actual de cada nodo. Seleccionamos el arco de menor peso que quede en un tiempo $O(e)$ y determinamos las componentes de los dos nodos conectados por el arco en un tiempo $O(m)$. Si se encuentran en componentes diferentes, unimos todos los nodos con dichos números en un tiempo $O(m)$, explorando la tabla de nodos. El tiempo total invertido por este algoritmo es $O(e(e + m))$. Este tiempo de ejecución es polinómico en función del "tamaño" de la entrada, que informalmente hemos definido como la suma de e y m.

Al trasladar estas ideas a las máquina de Turing, nos enfrentamos con varios problemas:

- Al estudiar algoritmos, nos encontramos con "problema" que exigen generar salidas en distintas formas, tales como la lista de arcos de un árbol EM. Cuando trabajamos con máquinas de Turing, sólo podemos pensar en los problemas como si fueran lenguajes y la única salida que obtenemos es sí o no.
- Con esto en mente, podemos replantear el problema como sigue: "dado el grafo G y el límite W, ¿tiene G un árbol de expansión de peso W o menor?".

1. Informalmente, podemos interpretar que el "tamaño" de un grafo es el número de nodos o arcos del mismo. La entrada a una máquina de Turing es una cadena de un alfabeto finito. Por tanto, los elementos del problema, tales como los nodos y los arcos, tienen que codificarse de la forma adecuada.
   a) La diferencia entre el tamaño de una cadena de entrada de una máquina de Turing y el de la cadena del problema informal nunca es mayor que un factor pequeño, normalmente igual al logaritmo del tamaño de la entrada.
   b) La longitud de una cadena que representa la entrada es realmente una medida más precisa del número de bytes que una computadora real tiene que leer para obtener su entrada.

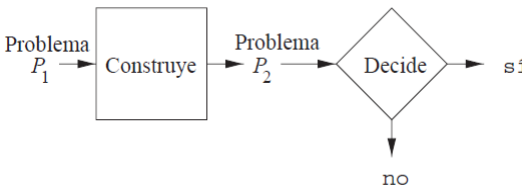
Sigamos con el algoritmo de Kruskal

Consideremos una posible codificación para los grafos y el límite peso que podrían ser la entrada del problema del árbol de recubrimiento de peso mínimo. El código utiliza cinco símbolos: 0, 1, los paréntesis y la coma.

1. Asignamos los enteros 1 hasta m a los nodos.
2. Iniciamos el código con los valores m y el límite de peso W expresados en binario, y separados por una coma.
3. Si existe un arco entre los nodos i y j con peso w, incluimos (i, j, w) en el código. Los enteros i, j y w están codificados en binario. El orden de i y j dentro de un arco y el orden de los arcos en el código no son importantes.

Tomando en cuenta lo anterior, uno de los posibles código para el grado de la figura con el límite de peso $W = 40$ es:

$$100, 101000(1, 10, 1111)(1, 11, 1010)(10, 11, 110$$
$$(10, 100, 10100)(11, 100, 10010)$$



Si representamos las entradas al problema del MST como lo mencionamos antes, entonces una entrada de longitud n puede representar como máximo $O(n/\log n)$ arcos. Es posible que m, el número de nodos, sea exponencial en n, si existen muy pocos arcos. Sin embargo, a menos que el número de arcos, e, sea como mínimo $m-1$, el grafo no tendrá un árbol de expansión de peso mínimo, independientemente de sus arcos. En consecuencia, si el número de nodos no es como mínimo una fracción de $n/\log n$, no es necesario ejecutar el algoritmo de Kruskal; simplemente diremos que "no existe ningún árbol de recubrimiento con dicho peso".

Por tanto, si tenemos un límite superior para el tiempo de ejecución del algoritmo de Kruskal que es una función de m y e, tal como el límite superior $O(e(m+e))$ desarrollado anteriormente, podemos reemplazar tanto m como e por n y decir que el tiempo de ejecución es una función de la longitud de la entrada n es $O(n(n+n))$, es decir, $O(n^2)$. En realidad, una implementación mejor del algoritmo de Kruskal invierte un tiempo $O(n \log n)$, pero aquí no nos interesa esta mejora.

Podemos implementar la versión del algoritmo de Kruskal descrita anteriormente en una MT de varias cintas:

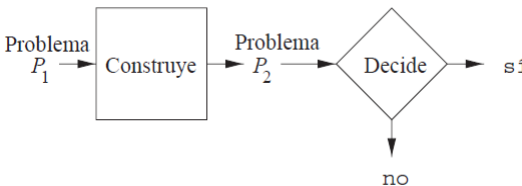
1. Se puede utilizar una cinta para almacenar los nodos y su número de componente actual. La longitud de esta tabla es $O(n)$.
2. Una cinta se puede emplear, a medida que exploramos los arcos sobre la cinta de entrada, para almacenar el arco de menor peso encontrado entre aquellos arcos que no han sido marcados como "usado". Podríamos emplear una segunda pista de la cinta de entrada para marcar aquellos arcos que fueron seleccionados como el arco de menor peso en las anteriores iteraciones del algoritmo. La búsqueda del arco de menor peso no marcado tarda $O(n)$, ya que cada uno de los arcos sólo se considera una vez, y las comparaciones de pesos pueden realizarse mediante una exploración lineal de derecha a izquierda de los números binarios.

3. Cuando se selecciona un arco en una iteración, se incluyen sus dos nodos en una cinta. Después en la tabla de nodos y componentes se buscan las componentes de estos dos nodos. Esta tarea consume un tiempo $O(n)$.
4. Una cinta se puede emplear para almacenar las dos componentes, i y j, que se unirán cuando se encuentre un arco que conecte dos componentes que anteriormente estaban desconectados. A continuación, exploramos la tabla de nodos y componentes, y para cada nodo que encontremos en la componente i cambiamos su número de componente a j. Este proceso también consume un tiempo $O(n)$.

Una clase fundamental de problemas en el estudio de la intratabilidad es la de aquellos problemas que pueden resolverse mediante una MT no determinista que trabaja en tiempo polinómico. Formalmente, decimos que un lenguaje L pertenece a la clase NP (polinómico no determinista) si existe una MT no determinista M y una complejidad de tiempo polinómico $T(n)$ tal que $L = L(M)$, y cuando M recibe una entrada de longitud n, no existe ninguna secuencia de más de $T(n)$ movimientos de M.

Con el fin de tener una idea de la potencia de NP, vamos a ver un ejemplo de un problema que parece ser de clase NP pero no P: el problema del agente viajero (TSP, Traveling Salesman Problem). La entrada del TSP es la misma que la del árbol MWST, un grafo con pesos enteros en los arcos y un límite de peso W. La cuestión que se plantea es si el grafo contiene un "circuito hamiltoniano" de peso total como máximo igual aW. Un *circuito hamiltoniano* es un conjunto de arcos que conectan los nodos en un único ciclo, en el que cada nodo aparece exactamente una vez. Observe que el número de arcos de un circuito hamiltoniano tiene que ser igual al número de nodos del grafo.

Primero, debemos replantear el problema para que pueda ser resuelto por una MT: "dado el grafo G y el límite W, ¿tiene ciclo hamiltoniano de peso W o menor?". Pro ejemplo, para este grafo, podríamos preguntar ¿existe un ciclo hamiltoniano de peso menor a 64?



Una forma de resolver esta pregunta sería probar todos los ciclos y calcular su peso total. Siendo inteligentes, podemos eliminar algunas de las opciones obviamente malas. Sin embargo, parece que da igual lo que hagamos, tendremos que examinar una cantidad exponencial de ciclos antes de poder concluir que no existe ninguno con el límite de peso deseado W, o de encontrar uno si no tenemos suerte al elegir el orden en el que examinemos los ciclos.

Si disponemos de una computadora no determinista, podremos conjeturar una permutación de los nodos y calcular el peso total para el ciclo de nodos en dicho orden. Si se tratara de una computadora real no determinista, ningún camino podría utilizar más de $O(n)$ pasos si la longitud de la entrada fuera n. En una MT de varias cintas, podemos elegir una permutación en $O(n^2)$ pasos y comprobar su peso total en una cantidad de tiempo similar. Por tanto, una MTN de una sola cinta puede resolver el problema TSP en un tiempo $O(n^4)$ como máximo. Concluimos que el problema TSP pertenece a NP.

A continuación vamos a abordar la familia de problemas que son candidatos bien conocidos para pertenecer a NP pero no a P. Sea L un lenguaje (problema) que pertenece a NP. Decimos que L es $NP - completo$ si las siguientes afirmaciones sobre L son verdaderas:

1. L pertenece a NP.
2. Para todo lenguaje L' perteneciente a NP existe un reducción en tiempo polinómico de L' a L.

Es decir, un problema NP-completo es cualquier problema computacional para el que no se ha encontrado un algoritmo de solución eficiente.

Las *expresiones booleanas se construyen a partir de:*

1. Variables cuyos valores son booleanos; es decir, toman el valor 1 (verdadero) o el valor 0 (falso).
2. Los operadores binarios $\wedge$ y $\vee$, que representan las operaciones lógicas *and* y *or* de dos expresiones.
3. El operador unitario $\neg$ que representa la negación lógica.
4. Paréntesis para agrupar los operadores y los operandos, si fuera necesario para modificar la precedencia predeterminada de los operadores: $\neg$ es el operador de mayor precendencia, le sigue $\wedge$ y, finalmente, $\vee$.

Un ejemplo de una expresión booleana es $x \wedge \neg(y \vee z)$.

- Una *asignación de verdad* para una expresión booleana dada E asigna el valor verdadero o falso a cada de las variables que aparecen en E. El valor de la expresión E para una asignación de verdad T se designa como $E(T)$, y es el resultado de evaluar E reemplazando cada variable x por el valor $T(x)$ (verdadero o falso) que T asigna a x.
- Una asignación de verdad T *satisface* la expresión booleana E si $E(T) = 1$; es decir, la asignación de verdad T hace que la expresión E sea verdadera. Se dice que una expresión booleana E es *satisfacible* si existe al menos una asignación de verdad T que satisface E.

El *problema de la satisfacibilidad* es: Dada una expresión booleana, ¿es satisfacible?

¿Qué sucede cuando descubrimos nuevos problemas *NP – Completos*?

- Cuando descubrimos que un problema es *NP – completo*, sabemos que existen pocas posibilidades de que pueda desarrollarse un algoritmo eficiente para resolverlo. Es aconsejable y animamos a buscar soluciones heurísticas, soluciones parciales, aproximaciones u otras formas con el fin de evitar afrontar el problema directamente. Además, podemos hacerlo con la confianza de que no se nos está "escapando la solución adecuada".

- Cada vez que añadimos un nuevo problema $NP - completo$ a la lista, estamos reforzando la idea de que *todos* los problemas $NP - completos$ requieren un tiempo exponencial. El esfuerzo que sin duda se ha dedicado a encontrar algoritmos en tiempo polinómico para un problema P ha sido, inconscientemente, un esfuerzo dedicado a demostrar que $P = NP$. Es el peso acumulado de los intentos sin éxito hechos por muchos matemáticos y científicos expertos para demostrar algo que es equivalente a $P = NP$ lo que fundamentalmente nos lleva al convencimiento de que es muy improbable que $P = NP$, el lugar de ello lo más probable es que todos los problemas $NP - completos$ requieran un tiempo exponencial.

Ejercicio 2

En grupos pequeños,

1. Revisa que otros problemas NP-completos.
2. Selecciona uno de estos problema (que no sea de los mencionados en clase) y revisa en que situaciones de la vida real podemos encontrar este problema.