El Castor Laborioso y la Matemática Fundamental

GGA, jubilado UNAM.
UNAM, CdMx

January 2, 2021

Abstract

Texto de divulgación matemática dirigido a estudiantes y profesores del bachillerato en adelante.

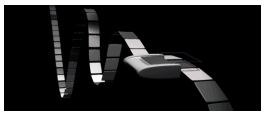
Part I

Parte I: El juego del "castor laborioso".

1 El problema del 'castor laborioso'.

De todo mundo es conocida la constancia de los *castores (marmotas, nutrias)* en mantener a la hora de elaborar sus famosas '*represas*' y así poder detener el flujo de la corriente y crear estanques de aguas tranquilas, donde poder construir sus madrigueras.

Esta idea de la consistencia del *castor* tuvo que ser la que llevo al matemático de origen húngaro *Tibor Radó* (1895-1965) a bautizar como el *juego o problema del* '*cas tor laborioso*' a cierto problema que planteó para una '*Máquina de Turing'* [1]



[1] La '*Máquina de Turing' (MT)* es un *dispositivo* que *manipula símbolos* sobre una *cinta 'infinita'* de acuerdo con una *tabla de reglas*. Puede ser adaptada para simular la lógica de cualquier algoritmo de computadora. Definida inicialmente por el matemático *Alan Turing* como una "*maquina automáti ca*" en *1936* en [1] no está diseñada como una tecnología de computación práctica,sino como un artefacto teórico hipotético que representa a una máquina de computo. Éstas máquinas ayudan a los científicos, entre otras cosas, a entender los límites de *cálculos mecánicos*. Con una limitada capacidad de memoria obtenida en la forma de una cinta infinita marcada con cuadrados en cada uno de los cuáles se podría imprimir un simbolo. Sin embargo la cinta puede tener movimientos elementales hacia adelante o hacia atrás.

Fig.1. Escultura con foto al fondo de *Alan Turing*. Fig1A. Imagen visual de una Máquina de Turing.

1.1 Planteamiento del problema.

El hungaro-norteamericano *Tibor Radó* definió al "*castor laborioso*" en [A] como una "Máquina de *Turing" (MT)* que satisfase dos condiciones:

1. Al poner el castor a laborar sobre una cinta totalmente ocupada por 0's (ceros), ésta termina por detenerse.

2. El número de 1's (unos) que imprime no es inferior al que pueda imprimir cualquier '*Máquina de Turing' (MT)* de igual *número de estados N* que lle gue a detenerse. Este número de 1's (unos) se denota por S (ó $\blacksquare(N)$)

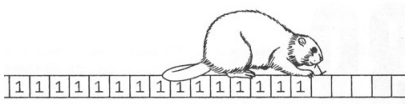


Fig.1A. Busy Beaver (castor laborioso) pone otro 1 en la cinta de una máquina de Turing (imagen del libro The New Turing Omnibus).

En otras palabras, el problema del castor laborioso es un divertido problema teóri- co de la informática. En forma intuitiva, el problema es encontrar el programa más pe- queño que genere tantos datos como sea posible y finalmente se detenga.

Resulta que este problema no se puede resolver. En el caso de estados más pequeños, puede razonar al respecto y hacer estimaciones, pero no se puede resolver en general. Los teóricos llaman a estos problemas no computables.

Actualmente, las personas han logrado resolverlo para $N = 1, 2, 3, 4$ (para '*Máquinas de Turing*' (MT) con $1, 2, 3$ y 4 estados) razonando y ejecutando todas las Máquinas de Turing (MT) posibles, pero para $N = 5$ esta tarea hasta ahora ha sido imposible. Si bien es probable que se resuelva para $N = 5$, los teóricos del tema dudan que alguna vez se pueda calcular para $N = 6$ en forma exacta.

Denotemos por $B(N)$ al número de 1'os (unos) que el atareado castor logró poner en la cinta de una MT. Esta función $B(N)$ se denomina *función de castor ocupado* y es la *solución al problema del castor ocupado.* La función del castor ocupado también es interesante: *crece más rápido* que cualquier *función computable.* Crece así:

$B(1) = 1$

$B(2) = 4$

$B(3) = 6$

$B(4) = 13$

$B(5) = 4098$ (cota inferior, ya que aún no se ha encontrado el resultado exacto)

$B(6) = 4.6 \cdot 10^{1439}$ (cota inferior, ya que el exacto al parecer nunca se conocerá)

Si se usára un átomo por cada 1 (uno) que el castor ocupado pone en la cinta de la MT, entonces para $N = 6$, ¡se llenaría todo el Universo! Así de rápido crece la función del castor ocupado.

1.2 La definición original.

Más formalmente, es así: dada una '*máquina de Turing*' (MT) de N estados con un alfabeto de dos símbolos $\{0, 1\}$, ¿cuál es el número máximo de 1's (**unos)** que la MT pueda imprimir en una cinta inicialmente en blanco (llena de 0's) antes de parar?

La definición original dada por *Radó* de '*Castor Laborioso*' como una '*Máquina de Turing' (MT)* en el formato de una *quínteta fue la siguiente:* con **i)** y **ii)** $N + 1$ estados (N *estados usuales* y el *estado final de parada*), con **iii)** y **iv)** El *alfabeto de la cinta* que tiene *dos símbolos: blanco* y 1 ($\blacksquare \in \{blanco, 1\}$) y **v)** el *alfabeto de entrada* formado por sólo 1's ($\blacksquare = \{1\}$). La quinteta es: $(E, \Gamma, \Pi) \stackrel{def}{=} (\{e_N, e_f\}, \{blanco, 1\}, \{1\})$.

1.3 Productividad.

La productividad de una MT se representa como el numero de 1's (unos) que se tiene en el resultado, a partir de una cinta en *blanco* (o en 0's), cuando ésta se para. Las MT que *no se detienen* se les consideran de *productividad cero.* La *productividad* de una MT se representa como la cantidad de 1's que se obtiene como resultado, a partir de una cinta en *blanco*, cuando esta se detiene. La $S = \Sigma(N)$ se define como la máxima productividad que se puede obtener a partir de una *máquina de Turing* (MT) de N estados (N reglas). Para resolver este problema se utiliza una *Máquina de Turing* (MT). La cinta de esta MT puede tomar sólo 2 valores $\{0, 1\}$ (o $\{B, 1\}$, con B como *blanco*). La cinta inicial debe estar llena de *blancos* ó 0's. La '*Máquina de Turing*' (MT) puede desplazarse en cualquiera de sus dos direcciones sin restricciones (en particular, es válido el desplazamiento a la derecha desde la posición inicial).

1.4 ¿Para qué se usa un castor laborioso?

Primero recordar que en *1936 Turing* definió las máquinas que ahora llevan su nombre ('*Máquinas de Turing' (MT)*) como un *modelo universal de cómputo* en el que todas las funciones computables imaginables pueden ser calculadas en ese modelo universal de cómputo. Entonces, *¿cómo se puede encontrar una función que no sea computable*? Aquí es donde entra el '*castor laborioso'* ya que el mismo *Tibor Radó* demostró que si $P : N \longrightarrow N$ es una función computable, entonces existe $\Sigma(n)$, tal que $\Sigma(n) > P(n)$ para toda n suficientemente grande, y por lo tanto Σ **no** es una función computable.

1.5 ¿Cuál es el problema?

El problema consiste principalmente en la utilización de una '*Máquina de Turing*' (MT) de un cierto número de estados finito, digamos N, para generar la mayor cantidad de 1's (unos) posibles según el número de estados escogidos y que finalmente la MT se detenga. (La cantidad de 1's (unos) que se puede generar varía en función de la cantidad de estados N que se utilicen para la generación de la MT)

Lo curioso de este problema es que el *número de estados* de la *Maquina de Turing N (mayor o igual que* 5) no se conoce con total certeza cuál es tal número máximo de 1's (unos) que se pueden generar sobre la cinta.

1.6 ¿Qué se debe cumplir?

Como ya se dijo La '*Máquina de Turing*' (MT) tiene que cumplir dos condiciones esenciales (equivalente a lo ya mencionado):

·El alfabeto de la cinta solo puede tener dos valores *Blanco* y 1: $\{Blanco, 1\}$.

·La cinta inicialmente está en *Blanco* en todas sus posiciones.

Crear el mayor número de 1's (unos) posibles y la '*Máquina de Turing*' (MT) debe detenerse. La parte variable del problema es el *número de estados* que se puedan utilizar para éllo.

Como ya se dijo se define la función $P : N \longrightarrow N$ donde el valor de entrada es el *número de estados de la MT* y el de salida la cantidad máxima de 1's (unos) que pueda generar la '*Máquina de Turing*' (MT) con los estados dados para la MT.

Este juego no está resuelto es un problema abierto, es decir, no se conoce la función P descrita arriba.

Existen soluciónes para un *número de estados* menor a 5 y también existen cotas inferiores para un *número de estados* menor a 7 y eso es todo lo que se sabe.

También se utilizará la función S igual a P sólo que ésta nueva función devuelve valores, que son el *número de pasos* realizados por la MT.

Estos son los valores exactos y las cotas inferiores conocidas

N	1	2	3	4	5	6
P	1	4	6	13	4098	4.6×10^{1439}
S	1	6	21	107	$47,176,870$	2.5×10^{2879}

Como puede apreciarse es un crecimiento muy difícil de calcular. Si se coloca la cinta y por cada paso de la máquina la fuésemos clonando desplazándola hacia abajo quedarían las siguientes soluciones:

1.7 Soluciones para $N < 5$

Como bien se ha dicho, para el numero de estados N mayor o igual que 5 no se puede saber con total seguridad el número máximo de 1's (unos) que se pueden generar, pero para $N < 5$, sí que se sabe. Hay aficionados que juegan con el

castor laborioso y éllos mismos intentan verificar los resultados conocidos para $N < 5$. Lo usual es que implementan una '*máquina de Turing', digamos* en '*Python*'. Aquí está el programa en *Python* que simula todas estas '*Máquinas de Turing*' (MT). Como ya se dijo, resulta ser demasiado lento y para el castor laborioso de 5 estados, tomó 5 min. generar la solución más conocida actualmente [2], pero usualmente resulta ser demasiado *lenta*, así que también lo usual

[2] Descargar: busy_beaver.py

```
#! / usr / bin / python
#
# Simulador de máquina de Turing para el problema de Busy Beaver.
# Versión 1.0
#
importar sys
error de clase (excepción):
pasar
clase TuringMachine (objeto):
def __init __ (self, program, start, halt, init):
self.program = programa
self.start = inicio
self.halt = detener
self.init = init
self.tape = [self.init]
self.pos = 0
self.state = self.start
self.set_tape_callback (Ninguno)
self.tape_changed = 1
self.movez = 0
def ejecutar (auto):
tape_callback = self.get_tape_callback ()
while self.state! = self.halt:
si tape_callback:
tape_callback (self.tape, self.tape_changed)
lhs = self.get_lhs ()
rhs = self.get_rhs (lhs)
new_state, new_symbol, move = rhs
old_symbol = lhs [1]
self.update_tape (antiguo_símbolo, nuevo_símbolo)
self.update_state (new_state)
self.move_head (mover)
si tape_callback:
tape_callback (self.tape, self.tape_changed)
def set_tape_callback (self, fn):
self.tape_callback = fn
def get_tape_callback (auto):
return self.tape_callback
propiedad (get_tape_callback, set_tape_callback)
@propiedad
def se mueve (auto):
volver self.movez
def update_tape (self, old_symbol, new_symbol):
if old_symbol! = new_symbol:
self.tape [self.pos] = new_symbol
self.tape_changed + = 1
más:
self.tape_changed = 0
```

```
  def update_state (self, state):
  self.state = estado
  def get_lhs (yo):
  under_cursor = self.tape [self.pos]
  lhs = self.state + under_cursor
  volver lhs
  def get_rhs (self, lhs):
  si no está en el autoprograma:
  subir Error ('No se pudo encontrar la transición para el estado "% s".'% lhs)
  return self.program [lhs]
  def move_head (yo, mover):
  if move == 'l':
  self.pos - = 1
  movimiento elif == 'r':
  self.pos + = 1
  más:
  subir Error ('Movimiento desconocido "% s". Solo puede ser hacia la izquierda o hacia la
derecha.'% move)
  si self.pos <0:
  self.tape.insert (0, self.init)
  self.pos = 0
  if self.pos> = len (self.tape):
  self.tape.append (self.init)
  self.movez + = 1
  beaver_programs = [
  {},
  {'a0': 'h1r'},
  {'a0': 'b1r', 'a1': 'b1l',
  'b0': 'a1l', 'b1': 'h1r'},
  {'a0': 'b1r', 'a1': 'h1r',
  'b0': 'c0r', 'b1': 'b1r',
  'c0': 'c1l', 'c1': 'a1l'},
  {'a0': 'b1r', 'a1': 'b1l',
  'b0': 'a1l', 'b1': 'c0l',
  'c0': 'h1r', 'c1': 'd1l',
  'd0': 'd1r', 'd1': 'a0r'},
  {'a0': 'b1l', 'a1': 'a1l',
  'b0': 'c1r', 'b1': 'b1r',
  'c0': 'a1l', 'c1': 'd1r',
  'd0': 'a1l', 'd1': 'e1r',
  'e0': 'h1r', 'e1': 'c0r'},
  {'a0': 'b1r', 'a1': 'e0l',
  'b0': 'c1l', 'b1': 'a0r',
  'c0': 'd1l', 'c1': 'c0r',
  'd0': 'e1l', 'd1': 'f0l',
  'e0': 'a1l', 'e1': 'c1l',
  'f0': 'e1l', 'f1': 'h1r'}
  ]
  def ocupado_beaver (n):
  def tape_callback (tape, tape_changed):
  si tape_changed:
  imprimir " .join (cinta)
  program = beaver_programs [n]
  imprimir "Ejecutando Busy Beaver con% d estados". % n
  tm = TuringMachine (programa, 'a', 'h', '0')
  tm.set_tape_callback (tape_callback)
  tm.run ()
```

es que se vuelve a implementar en digamos $C++$ (véase más adelante).

Se pueden ver cada uno de estos autómatas por separado:

·Problema del '*Castor Laborioso*' resuelto con $(N=)1$ estado. El estado final está excluido por lo que para $(N=)1$ la '*máquina de Turing*' (MT) tiene los estados (inicial y final : q_0 y q_f), pero q_f no se utiliza ya que en el momento que la MT llega a ese estado la máquina se detiene:

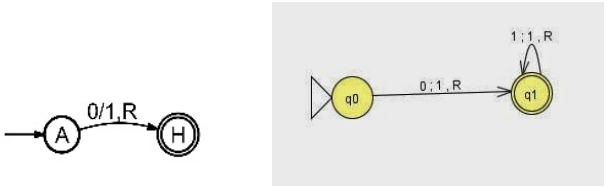


Fig.2. Solución gráfica al problema del 'Castor Laborioso' con una MT de $(N=)1$ estado (2 variantes).

La cinta se queda con un sólo uno.

·Problema del Castor Laborioso resuelto con $(N=)2$ estados:

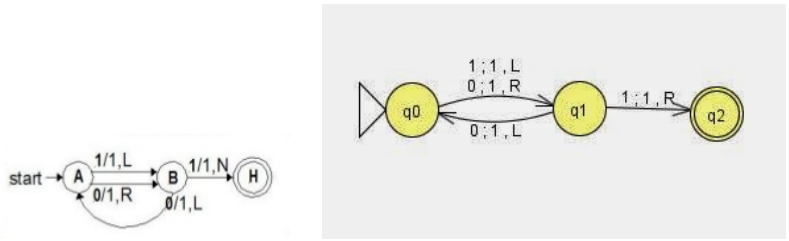
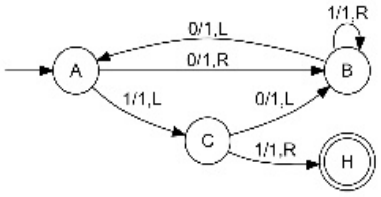


Fig.3. Solución gráfica al problema del 'Castor Laborioso' con una MT de $(N=)2$ estados (3 variantes).

```
imprimir "Castor ocupado terminado en% d pasos". % tm.movimientos
def uso ():
imprimir "Uso:% s [1 | 2 | 3 | 4 | 5 | 6]"% sys.argv [0]
imprimir "Ejecuta un problema de Busy Beaver en 1 o 2 o 3 o 4 o 5 o 6 estados".
sys.exit (1)
if __name__ == "__main__":
si len (sys.argv [1:]) <1:
uso()
n = int (sys.argv [1])
si n <1 o n> 6:
print "n debe estar entre 1 y 6 inclusive"
impresión
uso()
ocupado_beaver (n)
```

·Problema del 'Castor Laborioso' resuelto con $(N =) 3$ estados:



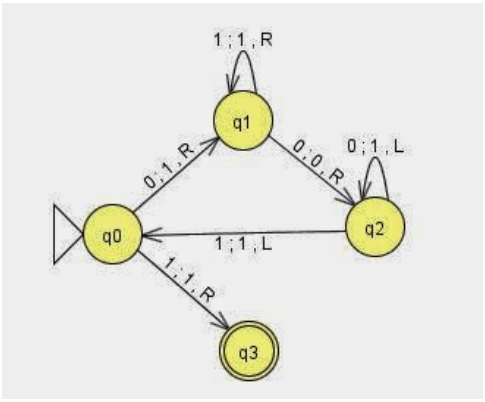
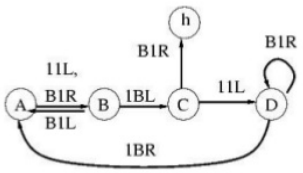


Fig.4. Solución gráfica al problema del 'Castor Laborioso' con una MT de $(N =) 3$ estados. (3 variantes)

Problema del Castor Laborioso resuelto con $(N =) 4$ estados:



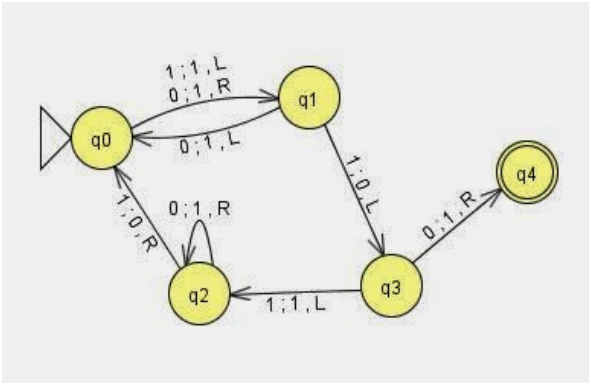


Fig.5. Solución gráfica al problema del 'Castor Laborioso' con una MT de $(N =) 4$ estados (3 variantes)

·El problema (juego) del '*Castor Laborioso'* (o The Busy Beaver Game) es un jue- go fácil de entender pero difícil de jugar. Este juego lo propuso *Tibor Radó* [3] y está pensado para resolverse en una '*Máquinas de Turing' (MT).*

[3] El hungaro Tibor Radó y entre 1913 y 1915 asistió al Instituto Politécnico donde estudió ingeniería civil. En la primera guerra mundial fue capturado en el frente ruso. Logró escapar del campo de prisioneros y, viajando miles de kilometros logró regresar a Hungaria. A partir de eso se le considera un científico aventurero.

Él consiguió un doctorado en la universidad 'Franz Joseph' en 1923. Ejerció la docencia en la universidad en un período breve y empezo como adjunto a la fundación Rockefeller en Alemania.

En 1929 se trasladó a los Estados Unidos impartiendo conferencias en la universidad de Harvard y en el Instituto Rice antes de por fin aterrizar y obtener un puesto en el departamento

Lo propuesto por *Radó* no es un simple *juego de entretenimiento*. Se considera que una solución general de este problema implicaría un paso importante en el área de la *computación*.

1.8 Aplicaciones.

Se sabe que una máquina, en general, puede ser representada como una '*Máquina de Turing'* (MT) de N estados, donde la cinta es la memoria y la memoria sólo puede tener 0's y 1's (en la '*Máquina de Turing'* sería *blancos* y 1's)

Se sabe que una *máquina de Turing* con N estados no puede realizar más de $S(N)$ *pasos* ya que, si los realizase, querría decir que habría entrado en un *bucle infinito*.

Sabiendo lo anterior, se puede saber si una máquina parará o no ejecutándola, ya que si la máquina en algún momento realiza más de $S(N)$ pasos querría decir que ha entrado en un *bucle infinito*.

Las técnicas usadas actualmente para $N > 4$ solo llevan a cabo una búsqueda parcial en el espacio de soluciones, buscando la MT que establezcan un mejor límite inferior para el valor de S (ó $\Sigma(N)$. Por ejemplo, *Marxen* estableció: $\Sigma(5) \geq 4098$. Otros estudios han conseguido establecer límites inferiores del tipo: $\Sigma(7) \geq 102$ mediante algoritmos genéticos y escalada de colina.

Como podemos observar se trata de *un problema no computable* para N grandes, el cual sigue sin solución tras más de medio siglo de su planteamiento y no parece haber visos de lo contrario.

Las cintas de las '*Maquinas de Turing'* (MT) están inicialmente llenas de ceros. Su estado inicial es "a" y el estado de detención es "h". La notación "$a_0 -> b_1 I$" significa "si se está en el estado "a" y el símbolo actual en la cinta es "0", entonces ponga un "1" en esa celda, cambie al estado "b" y muévase hacia la izquierda "l" Este proceso se repite (se itera) hasta que la máquina termina en el estado de *parada*.

·'*Máquina de Turing*' (MT) para castor ocupado de 1 estado: $a_0 -> h_1 r$

·'*Máquina de Turing*' (MT) para castor ocupado de2estados $\begin{cases} a_0 -> b_1 r a_1 -> b_1 l \\ b_0 -> a_1 l b_1 -> h_1 r \end{cases}$

Así es como se ven los cambios en la cinta para el castor ocupado de 2 estados:

(véase la Fig.3) Cambios de cinta para castor ocupado de 2 estados.

---

de matemáticas en la universidad estatal de Ohio en 1930.

Durante la segunda guerra mundial, ya como norteamericano aparece como científico del gobierno norteamericano. En 1948.lo convirten en coordinador del departamento de matemáticas de la universidad estatal de Ohio.

En la década de 1920 demostró que las superficies tenían una triangulación esencial única. En 1933 publicó sobre el problema de Plateau, donde dio una solución al problema de Plateau, y en 1935, publicó algo sobre funciones subarmónicas. Su trabajo se centró en la ciencia informática en la última década de su vida. Al parecer la filosofía de Radó fue la de aquel personaje que consideraba que lo mejor variante de lo que podía conseguir en Hungría no era mejor que la peor variante de lo que le ofrecía el sueño americano.

En Mayo de 1962 publicó uno de sus más famosos artículos en la revista "Bell System Technical": "the busy beaver funciton" (La función del castor laborioso) y su no computabilidad recogidos en su publicación "On Non - Computable Functions".

·MT para Castor Laborioso de 3 estados: $\begin{cases} a_0 -> b_1 r a_1 -> h_1 r \\ b_0 -> c_0 r b_1 -> b_1 r \\ c_0 -> c_1 l c_1 -> a_1 l \end{cases}$

(véase la Fig.4)

·Máquina de Turing para castor ocupado de 4 estados: $\begin{cases} a_0 -> b_1 r a_1 -> b_1 l \\ b_0 -> a_1 l b_1 -> c_0 l \\ c_0 -> h_1 r c_1 -> d_1 l \\ d_0 -> d_1 r d_1 -> a_0 r \end{cases}$

(véase la Fig.5)

·Máquina de Turing para el castor laborioso de 5 estados: $\begin{cases} a_0 -> b_1 r a_1 -> b_1 l \\ b_0 -> a_1 l b_1 -> c_0 l \\ c_0 -> h_1 r c_1 -> d_1 l \\ d_0 -> d_1 r d_1 -> a_0 r \\ e_0 -> h_1 r e_1 -> c_0 r \end{cases}$

Esta imagen es enorme (6146×14293 píxeles, pero solo 110 KB de tamaño).

Cambios de cinta para castor ocupado de 5 estados.

·Máquina de Turing para castor ocupado de 6 estados: $\begin{cases} a_0 -> b_1 r a_1 -> b_1 l \\ b_0 -> a_1 l b_1 -> c_0 l \\ c_0 -> h_1 r c_1 -> d_1 l \\ d_0 -> d_1 r d_1 -> a_0 r \\ e_0 -> h_1 r e_1 -> c_0 r \\ f_0 -> e_1 l f_1 -> h_1 r \end{cases}$

¡He aquí el simulador de la MT reescrito en $C++$ y la aceleración fue enorme! Ahora solo tomó 14 *segundos* el ejecutarse el mismo castor 5 laborioso [4].

[4] He aquí el simulador de Turing Machine reescrito en C++ y la aceleración fue enorme! Ahora solo tomó 14 segundos en ejecutarse el mismo castor laborioso 5.

Descargar: busy_beaver.cpp

```
/ *
** Simulador de máquina de Turing para el problema de Busy Beaver.
** Versión 1.0
* /
#include <cstdlib>
#include <iostream>
#include <utilidad>
#include <vector>
#include <cadena>
#include <map>
usando el espacio de nombres std;
typedef vector <char> Tape;
typedef map <cadena, cadena> Programa;
class TuringMachine {
privado:
Cinta adhesiva;
Programa de programa;
char iniciar, detener, iniciar, estado;
bool tape_changed;
int se mueve;
int pos;
público:
TuringMachine (programa de programa, inicio de caracteres, detención de caracteres, inicio
```

```
de caracteres):
  tape (1, init), programa (programa), inicio (inicio), detener (detener),
  init (init), estado (inicio), movimientos (0), tape_changed (1), pos (0)
  {}
  void run () {
  while (estado! = detener) {
  print_tape ();
  cadena lhs = get_lhs ();
  cadena rhs = get_rhs (lhs);
  char new_state = rhs [0];
  char new_symbol = rhs [1];
  movimiento de char = rhs [2];
  char old_symbol = lhs [1];
  update_tape (antiguo_símbolo, nuevo_símbolo);
  estado_actualización (estado_nuevo);
  move_head (mover);
  }
  print_tape ();
  }
  int get_moves () {
  movimientos de retorno;
  }
  privado:
  inline void print_tape () {
  if (tape_changed) {
  para (int i = 0; i <tape.size (); i ++)
  cout << cinta [i];
  cout << endl;
  }
  }
  cadena en línea get_lhs () {
  char sp [3] = {0};
  sp [0] = estado;
  sp [1] = cinta [pos];
  return string (sp);
  }
  cadena en línea get_rhs (cadena & lhs) {
  programa de retorno [lhs];
  }
  inline void update_tape (char old_symbol, char new_symbol) {
  if (old_symbol! = new_symbol) {
  tape [pos] = new_symbol;
  tape_changed ++;
  }
  else {
  tape_changed = 0;
  }
  }
  inline void update_state (char new_state) {
  state = new_state;
  }
  inline void move_head (movimiento de caracteres) {
  si (mover == 'l')
  pos - = 1;
  más si (mover == 'r')
  pos + = 1;
  más
```

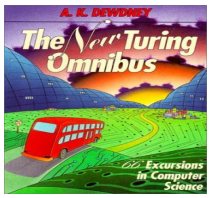
```
lanzar cadena ("estado desconocido");
si (pos <0) {
tape.insert (tape.begin (), init);
pos = 0;
}
if (pos> = tape.size ()) {
tape.push_back (init);
}
se mueve ++;
}
};
vector <Programa> busy_beavers;
vacío init_bb6 ()
{
Programa bb6;
bb6.insert (make_pair ("a0", "b1r"));
bb6.insert (make_pair ("b0", "c1l"));
bb6.insertar (make_pair ("c0", "d1l"));
bb6.insert (make_pair ("d0", "e1l"));
bb6.insert (make_pair ("e0", "a1l"));
bb6.insert (make_pair ("f0", "e1l"));
bb6.insert (make_pair ("a1", "e0l"));
bb6.insert (make_pair ("b1", "a0r"));
bb6.insert (make_pair ("c1", "c0r"));
bb6.insert (make_pair ("d1", "f0l"));
bb6.insert (make_pair ("e1", "c1l"));
bb6.insert (make_pair ("f1", "h1r"));
busy_beavers.push_back (bb6);
}
vacío init_bb5 ()
{
Programa bb5;
bb5.insert (make_pair ("a0", "b1l"));
bb5.insert (make_pair ("b0", "c1r"));
bb5.insert (make_pair ("c0", "a1l"));
bb5.insert (make_pair ("d0", "a1l"));
bb5.insert (make_pair ("e0", "h1r"));
bb5.insert (make_pair ("a1", "a1l"));
bb5.insert (make_pair ("b1", "b1r"));
bb5.insert (make_pair ("c1", "d1r"));
bb5.insert (make_pair ("d1", "e1r"));
bb5.insert (make_pair ("e1", "c0r"));
busy_beavers.push_back (bb5);
}
vacío init_bb4 ()
{
Programa bb4;
bb4.insert (make_pair ("a0", "b1r"));
bb4.insert (make_pair ("b0", "a1l"));
bb4.insert (make_pair ("c0", "h1r"));
bb4.insert (make_pair ("d0", "d1r"));
bb4.insert (make_pair ("a1", "b1l"));
bb4.insert (make_pair ("b1", "c0l"));
bb4.insert (make_pair ("c1", "d1l"));
bb4.insert (make_pair ("d1", "a0r"));
busy_beavers.push_back (bb4);
}
```

```
vacío init_bb3 ()
{
Programa bb3;
bb3.insert (make_pair ("a0", "b1r"));
bb3.insert (make_pair ("b0", "c0r"));
bb3.insert (make_pair ("c0", "c1l"));
bb3.insert (make_pair ("a1", "h1r"));
bb3.insert (make_pair ("b1", "b1r"));
bb3.insert (make_pair ("c1", "a1l"));
busy_beavers.push_back (bb3);
}
vacío init_bb2 ()
{
Programa bb2;
bb2.insert (make_pair ("a0", "b1r"));
bb2.insert (make_pair ("b0", "a1l"));
bb2.insert (make_pair ("a1", "b1l"));
bb2.insert (make_pair ("b1", "h1r"));
busy_beavers.push_back (bb2);
}
vacío init_bb1 ()
{
Programa bb1;
bb1.insert (make_pair ("a0", "h1r"));
busy_beavers.push_back (bb1);
}
vacío init_busy_beavers ()
{
busy_beavers.push_back (Programa ());
init_bb1 ();
init_bb2 ();
init_bb3 ();
init_bb4 ();
init_bb5 ();
init_bb6 ();
}
void busy_beaver (int n)
{
cout << "Ejecutando Busy Beaver con" << n << "estados". << endl;
TuringMachine tm (ocupado_beavers [n], 'a', 'h', '0');
tm.run ();
cout << "Busy Beaver terminó en" << tm.get_moves () << "pasos". << endl;
}
uso nulo (const char * prog)
{
cout << "Uso:" << prog << "[1 | 2 | 3 | 4 | 5 | 6] \ n";
cout << "Ejecuta el problema Busy Beaver en 1 o 2 o 3 o 4 o 5 o 6 estados." << endl;
salida (1);
}
int main (int argc, char ** argv)
{
si (argc <2) {
uso (argv [0]);
}
int n = atoi (argv [1]);
si (n <1 || n> 6) {
cout << "n debe estar entre 1 y 6 inclusive! \ n";
```

También hay variaciones de este problema. Por ejemplo, para el castor más laborioso con 3 o más símbolos [B]-[H]; puede verse "*The Busy Beaver Competition*".

Por otro lado, el desarrollo histórico también es interesante; para ello, se puede consultar "*The Busy Beaver Historical Survey*" para mayor información.

Un buen comienzo para aprender por primera vez sobre el problema del Busy Beaver (Castor Laborioso) es el hermoso libro titulado "*The New Turing Omnibus*" [I], el cual contiene 66 ensayos diferentes sobre diversos temas de informática, como *algoritmos*, *máquinas de Turing*, *gramáticas*, *computabilidad*, *aleatoriedad* y otros muchos temas cual más de divertidos. Estos ensayos están escritos en un estilo accesible que incluso los estudiantes de secundaria y bachillerato pueden entender. Cada ensayo tiene una o dos páginas y no toma más de 10 minutos leerlo. Este libro es altamente recomendable [I] y [J].



Y finalmente también he aquí, en esta probadita de programas, uno en *Perl*, que usa la biblioteca *GD* para dibujar los cambios de cinta de una *MT* [5].

```
  cout << "\ n";
  uso (argv [0]);
  }
  init_busy_beavers ();
  ocupado_beaver (n);
  }
```

[5] Descargar: draw_turing_machine.perl

```
  #! / usr / bin / perl
  #
  # Dada la salida de los programas busy_beaver.py u busy_beaver.cpp,
  # dibuje los cambios de cinta de la máquina de Turing (la celda negra significa 1, el blanco
0).
  #
  use advertencias;
  uso estricto
  utilizar GD;
  $ | ++;
  my $ input_file = shift or die 'Uso: $ 0 <archivo con transiciones de estado TM>';
  my $ cell_size = shift || 4;
  my $ im_file = "$ input_file.png";
  sub line_count {
  my $ count = 0;
  abre mi $ fh, '<', cambia o muere $ !;
  $ cuenta + = tr / \ n / \ n / while sysread ($ fh, $ _, 2 ** 20);
  return $ count;
  }
  sub get_last_line {
  mi $ archivo = turno;
```

```
my $ last_line = 'tail -1 $ file';
chomp $ last_line;
return $ last_line;
}
my $ nr_lines = line_count $ input_file;
my $ last_line = get_last_line $ input_file;
my $ last_width = longitud ($ last_line);
my ($ ancho, $ alto) = ($ tamaño_célula * $ último_ancho, $ tamaño_celda * $ nr_lines);
my $ im = GD :: Imagen-> nuevo ($ ancho, $ alto);
mi $ blanco = $ im-> colorAllocate (255,255,255);
mi $ oscuro = $ im-> colorAllocate (40, 40, 40);
my ($ x, $ y) = (0, $ altura- $ tamaño_célula);
print "Empezando a dibujar la imagen. Estados totales: $ nr_lines. \ n";
print "Tendrá un tamaño de pizels de $ ancho x $ alto. \ n";
my $ prev_line;
my ($ pad_left, $ pad_right) = (0, 0);
sub pad {
my ($ línea, $ izquierda, $ derecha) = @_;
devuelve '0'x $ izquierda. $ línea. '0'x $ derecha;
}
abre mi $ fh, "- |", "/ usr / bin / tac $ input_file" o muere $ !;
while (<$ fh>) {
masticar
impresión "." si $. % 10 == 0;
imprimir "($.)" si $. % 500 == 0;
$ prev_line = $ _ a menos que esté definido $ prev_line;
my $ new_line;
if (longitud $ _! = longitud $ prev_line) {
if ($ prev_line = ~/ 0 $ /) {
$ pad_right ++;
}
elsif ($ prev_line = ~/ ^0 /) {
$ pad_left ++;
}
else {
muere "datos inesperados en $. en el archivo $ input_file";
}
}
$ nueva_linea = pad ($ _, $ pad_left, $ pad_right);
$ prev_line = $ _;
my @cells = split //, $ nueva_línea;
para mi $ celda (@cells) {
$ im-> llenadoRectángulo ($ x, $ y, $ x + $ tamaño_célula, $ y + $ tamaño_celda,
$ celda? $ oscuro: $ blanco);
$ x + = $ tamaño_célula;
}
$ y - = $ tamaño_célula;
$ x = 0;
}
imprimir "\ n";
{
abre mi $ fh, ">", $ im_file o muere $ !;
imprimir $ fh $ im-> png;
cerrar $ fh;
}
imprimir "Listo. Imagen guardada en $ im_file. \ n";
```

No dude en probar estos programas usted mismo. Aquí es cómo. Ejecute "busy_beaver.py

1.9 Bibliografía consultada para la parte I:

References

[1] ·*Radó T.*; **On non-computable functions**; *Bell System Technical Journal, 41(3): 877-884, May 1962*, https://archive.org/details/bstj41–3–877/mode/2up. Google Scholar.

[2] http://en.wikipedia.org/wiki/Tibor_Rad%C3%B3

[3] http://blog.alphasmanifesto.com/2010/04/05/link-del-dia-el-castor-ocupado/

[4] http://www.catonmat.net/blog/busy-beaver/

[5] http://pt.wikipedia.org/wiki/Algoritmo_do_castor

[6] http://clickdefinicion.com/letra-a/algoritmo-beaver.php

[7] http://en.wikipedia.org/wiki/Busy_beaver

[8] http://en.wikipedia.org/wiki/Computable_function.

[9] *A.K. Dewdney*; **The New Turing Omnibus: 66 Excursions In Computer Science;** (1993), Editorial: W.H. Freeman & Co.

[10] https://get-download.club/ ¡Acceso gratuito e ilimitado a programas, juegos, musica, libros, peliculas y más! Ahí se pueden registrar. New Turing Omnibus.

---

n" con n = 1, 2, 3, 4, 5, 6. Esto ejecutará la máquina de Turing beaver de estado n ocupado. La salida serán los cambios de cinta. Luego use "draw_turing_machine.pl" para visualizar los cambios en la cinta.

Por ejemplo:
$ busy_beaver.py 4> bb4
$ draw_turing_machine.pl bb4

Part II

Parte II: Programas lentos explicitan los límites fundamentales de la matemática.

2 Lo inesperado entre el juego del 'Castor Laborioso' y los límites fundamentales de la matemática.

En la modernidad el objetivo del *juego* o *problema* del "*castor laborioso*" [1] es encontrar el programa informático de mayor duración posible. Y en su búsqueda resulta tener conexiones sorprendentes con algunas de las preguntas y conceptos más profundos de la matemática.

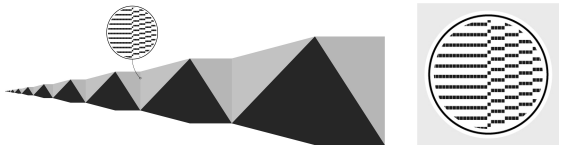


Fig.6. Una visualización de la máquina de Turing de cinco reglas más antigua que se conoce actualmente. Cada co- lumna de píxeles representa un paso en el cálculo, moviéndose de izquierda a derecha. Fig.6A. Los cuadrados ne- gros, del recuadro circular, muestran dónde la máquina ha impreso un 1. La columna del extremo derecho muestra el estado del cálculo cuando la '*Máquina de Turing*' (*MT*) se detiene. Cortesía de *Peter Krumins*.

Los programadores normalmente quieren minimizar el tiempo que tarda su código en ejecutarse. Pero en *1962*, el matemático húngaro *Tibor Radó* [6] [1] planteó el *problema contrario*. Se preguntó: *¿Cuánto tiempo se puede ejecutar un programa computacional simple antes de que termine*? *Radó* llamó a estos *programas sumamente ineficientes* pero aún funcionales "*Castores Laboriosos*" (*BusyBeaver, BB*)

Encontrar estos programas ha resultado un rompecabezas diabólicamente atracti- vo y divertido para los programadores y otros aficionados a las matemáticas desde que se popularizó en la columna '*Computer Recreations*' de la revista '*Scientific American*' en *1984*. Pero en los últimos años, el ajetreado '*juego de los castores*', como se le conoce, se ha convertido en un objeto de estudio por derecho propio, porque ha proporcionado conexiones, quién iba a pensarlo, con *algunos de los conceptos más elevados y problemas abiertos* de las matemáticas, ¡así como suena de extraordinario!

[6] Quién iba a pensar que un matemático sagaz y aventurero podía acercarse a los organismos apropiados y con su fantaciosa imaginación llegar hasta los EE UU, dónde conquistó y ahora al parecer se vislumbra conexiones interesantes con sus planteamientos.

Las declaraciones y los diálogos han sido tomados del especialista en divulgación matemática Pablus [16]. "*En matemáticas, hay un límite muy permeable entre lo que es una recreación divertida y lo que es realmente importante*" (*Scott Aaronson* - científico informático teórico de la Univ. de Texas, Austin, quien recientemente publicó una encuesta sobre el progreso en "Busy Beaver" [2].

Fig.7. Antes de llegar a Univ. de Texas, *Scott Aaronson* dictó cátedra durante nueve años en Ingeniería Eléctrica y Ciencias de la Computación en el MIT. Su área principal de investigación es la informática teórica.

La función del '*Castor Laborioso'* (*Busy Beaver* (BB)), con su crecimiento rápido incomprensible, ha cautivado a generaciones de informáticos, matemáticos y aficionados. En la encuesta de *Aaranson* [2], se ofrece una visión personal de la función *Castor Laborioso* (*Busy Beaver* (BB)) BB 58 años y otras muchas, con las cuales se llega a conclusiones sorprendentes.

En trabajo reciente sugiere que la búsqueda de programas informáticos de larga duración puede iluminar el estado del conocimiento matemático e incluso decirnos lo que se puede conocer. Según los investigadores, el ajetreado '*juego del castor*' proporciona un punto de referencia concreto para evaluar la dificultad de ciertos problemas, como la '*Conjetura de Goldbach'* (Todo número par mayor que 2, puede ser escrito como la suma de dos números primos), aún sin resolver y la '*hipótesis de Riemann*' (formulada en 1859, sobre la existencia de una relación entre la distribución de los números primos y la distribución de los ceros de la función zeta ■ de Riemann) Incluso ofrece una idea de dónde se rompe la base lógica subyacente de la matemática.

2.1 De Gödel un poco.

El lógico *Kurt Gödel* demostró la existencia de tal '*terra incognita*' matemática hace casi un siglo. Pero el ajetreado '*juego de los castores*' puede mostrar dónde se encuentra realmente en una recta numérica, como un mapa antiguo que representa la frontera del mundo.

Se tiene en mente la visión general de los célebres *teoremas de incompletez* de *Kurt Gödel*. Comenzando con la *paradoja de Richard*, el antinomio lógico que motivó a *Gödel* a mirar la codificación de las *metamatemáticas* en la *aritmética de los enteros*. (Pero primero se intenta comprender cómo *Gödel* se sintió motivado para pensar en codificar *metamatemáticas*. Esto vino de examinar el razonamiento involucrado en la presentación de la *paradoja de Richard*. Ésta a su vez puede ilustrarse con la "*paradoja del mentiroso*": Sea s la oración -"Esta oración es falsa"-. Dado que la frase "Esta oración ... se refiere a s, entonces se tiene la siguiente cadena de equivalencias "Esta oración es falsa"

sii[7] s es falsa **sii** no es s. Por lo tanto, se concluye que la *oración s es verdadera* en las oraciones en lengua española, luego *no es definible* en español. En *1905*, el matemático francés *Jules Richard* intentó matematizar '*la paradoja del mentiroso*'. Argumentó lo siguiente: Considere un *idioma* en el que se pueden formular y definir propiedades puramente aritméticas de los números naturales. Dado que cada una de estas definiciones contendrá sólo un número finito de palabras del alfabeto afín, se pueden ordenar las definiciones y ponerlas en orden serial (usando, por ejemplo, el orden léxicográfico). Por tanto, se puede, dado cualquier conjunto coherente de axiomas aritméticos, hay afirmaciones en aritmética que son verdaderas pero que no son deducibles de los axiomas.

Asociar un único entero positivo (indicando su posición) con cada definición en la lista. Supóngase que 15 corresponde a la definición '*es un múltiplo de* 5' y 17 a '*es un número compuesto*'. Observese que 15 en realidad satisface la propiedad correlacionada con élla, mientras que 17 no. Esto sugiere una nueva definición: 'n es *richardiano* como abreviatura de -n no tiene la propiedad designada por la enésima definición en nuestra lista de definiciones. Sea x el número asignado a la definición anterior (de ser *richardiano*). La paradoja se revela en la pregunta ¿a es *Richardiano*? ya que x es *Richardiano* sii x no tiene la propiedad (de ser *Richardiano*). Por tanto, el enunciado 'x es *Richardiano*' es tanto *verdadero* como *falso*. El lector atento según [?] puede haber captado la falacia en los argumentos anteriores. La definición de la propiedad de ser *Richardiano* (en el cuadro arriba descrito) es una propiedad que involucra notación y, por lo tanto, es *metamatemática* y nuestra lista de definiciones se restringió a propiedades puramente matemáticas. Por tanto, no pertenece en absoluto a nuestra lista y la paradoja se evapora. Sin embargo, la construcción motivó a *Gödel* a considerar cómo sería posible mapear *enunciados metamatemáticos* dentro de la *aritmética de números enteros*.

2.2 Numeración de Gödel.

El primer paso fue elegir un cálculo formal dentro del cual se puedan expresar todas las notaciones aritméticas habituales). En 1931, un joven matemático de la Universidad de Viena, *Kurt Gödel*, publicó un artículo en [18] : $-$*Acerca de proposiciones formalmente indecidibles de 'Principia Mathematica' y sistemas relacionados*$-$. Este artículo representa uno de los avances más importantes de la lógica en los tiempos modernos. Atacó un problema central de los fundamentos de las matemáticas.

Gödel demostró que el método axiomático de razonamiento deductivo tiene ciertas limitaciones inherentes. En particular, demostró que incluso la *aritmética ordinaria de números enteros* nunca puede axiomatizarse por completo.

Gödel demostró, en realidad, *dos resultados sustantivos*. Demostró que es imposi- ble dar una *prueba metamatemática* de la *consistencia de la aritmética de números enteros* a menos que se supongan reglas de inferencia que son esencialmente diferentes de las reglas de transformación usadas para deducir teore-

[7] *sii* significa si y sólo si.

mas en la aritmética.

Al tomar a la *metamatemática (Hilbert)* como el lenguaje que trata acerca de las *matemáticas*. El enunciado $\precsim \not\approx 0^1 0 \precsim$ es un enunciado matemático, pero "este enuncia- do $\precsim \not\approx 0^1 0 \precsim$ no es un teorema" es *metamatemática*. "La *aritmética* es *consistente*" es una declaración *metamatemática* más interesante. El 2do. resultado de *incomplez de Gödel* es aún más convincente. Demostró que dado cualquier conjunto consistente de axiomas aritméticos existen afirmaciones aritméticas que son verdaderas, pero que no pueden deducirse de los axiomas. Así demostró una *limitación* fundamental del poder del *método axiomático*. Incluso si los axiomas de la aritmética fueran extendidos por un número indefinido de otros enunciados verdaderos, siempre habría más verdades aritméticas que no se puedan deducir formalmente del conjunto aumentado de axiomas. El artículo original [18] de *1931* es difícil de leer. Antes de poder acceder a los resultados principales, deben comprenderse 46 definiciones junto con varios resultados preliminares importantes. Sin embargo existen libros de divulgación recientes como *Hofstadter (1979)* [18A] y *Penrose (1990)* [18B], que describen estos resultados de *Gödel* y algunas de sus consecuencias y finalmente el de *Nagel y Newman* [19] en su libro clásico de *1958* sobre el artículo [18] de *Gödel.*

2.3 Un juego de computadora inconfundible.

El ajetreado '*juego del castor*' tiene que ver con el comportamiento de las '*máquinas de Turing*' (*MT*): las computadoras primitivas e idealizadas concebidas por *Alan Turing* en *1936*. Una '*máquina de Turing' realiza acciones* sobre una *tira interminable de cinta dividida en cuadrados*. Lo hace de acuerdo con una *lista de reglas*. La *regla 1* podría decir: Si el cuadrado contiene un 0, reemplácelo con un 1, mueva un cuadrado a la derecha *R* y consulte la *regla 2*. Si el cuadrado contiene un 1, deje el 1, mueva un cuadrado a la izquierda *L* y consulte la *regla 3*.

Cada regla tiene este *estilo de bifurcación*, ahora elige tu propia aventura.Algunas reglas pueden indicar volver a las reglas anteriores; eventualmente hay una regla que contiene una instrucción para "*detenerse*". Turing demostró que *este tipo de computadora* simple es capaz de realizar cualquier *cálculo* posible, con las *instrucciones correctas* y el *tiempo suficiente*.

Como señaló *Turing* en *1936*, para calcular algo, una '*Máquina de Turing*' (*MT*) debe *detenerse* eventualmente, *no puede quedar atrapada en un ciclo infinito*. Pero también *demostró* que *no existe un método confiable y repetible para distinguir las máquinas que se detienen de las máquinas que simplemente funcionan para siempre* (hecho conocido como el *problema del paro*)

El ajetreado '*juego del castor*' pregunta: Dada una *cierta cantidad de reglas*, *¿cuál es la cantidad máxima de pasos que puede dar una 'máquina de Turing' (MT) antes de detenerse*?

Fig.?. Foto sin fecha en blanco y negro de Tibor Radó con traje y gafas, sentado en un escritorio y leyendo un li- bro, inventor el ajetreado '*juego del castor*' como una forma de concretar la noción teórica de la incomputable. Cortesía archivos de la Univ. Estatal de Ohio.

Por ejemplo, si solo se le permite una regla y se desea asegurarse de que la *má- quina de Turing* (MT) se detenga, *se verá obligado a incluir la instrucción (regla) de la detención de inmediato.* El *número de 'castor laborioso'* (BB) de una '*máquina de Turing* con una sola regla, o $BB(1)$, es por lo tanto 1.

Pero agregar algunas reglas más instantáneamente aumenta la cantidad de '*máquinas de Turing'* a considerar. De $6,561$ máquinas posibles con *dos reglas*, la que corre más tiempo (6 pasos) antes de detenerse es su '*castor laborioso'*. Pero otros simplemente corren para siempre. Ninguno de estos son los '*castores ocupados*', pero *¿cómo los descartas definitivamente*? *Turing demostró* que *no hay forma de saber en forma automática si una máquina que funciona durante mil o un millón de pasos no se detendrá finalmente.*

Por eso es *tan difícil encontrar 'castores laboriosos'*. No existe un enfoque general para identificar las '*Máquinas de Turing*' de mayor duración con un número arbitrario de reglas (instrucciones); tienes que descifrar los detalles de cada caso por sí solo. En otras palabras, el ajetreado '*juego de los castores*' es, en general, "incuestionable"

Demostrar que $BB(2) = 6$ y que $BB(3) = 107$ fue lo suficientemente difícil como pa- ra que el estudiante de *Radó*, *Shen Lin*, obtuviera un doctorado por ese trabajo en *1965.* Radó consideró a $BB(4)$ como "*totalmente descabellado*", pero el caso finalmente se resolvió en *1983* . Más allá de eso, los valores virtualmente explotan; Los investigadores han identificado una '*Máquina de Turing*' con 5 *reglas*, por ejemplo que se ejecuta durante $47,176,870$ *pasos* antes de *detenerse*, por lo que $BB(5)$ es al menos así de grande. $BB(6)$ es de al menos $7.4 \times\ 1,036,534$. Probar los valores exactos "*necesitará nuevas ideas y nuevos conocimientos, si es que se puede hacer*" (*Aaronson*). Todas las citas de entrevistas y declaraciones han sido tomadas de [17].

2.4 Umbral de incognoscibilidad.

El científico informático *William Gasarch* de la Univ. de Maryland, College Park, expresó que está menos intrigado por la posibilidad de precisar números de '*castores laboriosos*' que por "*el concepto general de que en realidad es incuestionable*". Él y otros matemáticos están interesadosprincipalmente en *usar el 'juego de castores' co- mo patrón para medir la dificultad de problemas abiertos importantes en matemáticas, o para descubrir qué es matemáticamente cognoscible.*

Por ejemplo la "Conjetura de Goldbach", que se pregunta si todo *entero par* ma- yor que 2 *es la suma de dos primos.* Demostrar que la conjetura es verdadera o falsa sería un evento trascendental en la teoría de números, que permitiría a los matemáti- cos *comprender mejor la distribución de los números primos.* En *2015*, un usuario anónimo de '*GitHub'* llamado '*Code Golf Addict*'

publicó un *código* para una '*Máquina de Turing*' (con *27 reglas* que se detiene *si, y solo si*, la '*Conjetura de Goldbach' es falsa.* Funciona contando hacia arriba a través de todos los enteros pares mayores que *4*; para cada uno, *analiza todas las formas posibles* de obtener ese N^o entero agregando otros dos, verificando si la dupla es de primos. Cuando encuentra una pareja adecuada de números primos, avanza al siguiente entero par y se repite el procedimieto. Si encuentra un número entero *par* que no se puede sumar con un par de números primos se detiene.

Hacer funcionar esta máquina sin sentido no es una forma práctica de resolver la '*Conjetura de Golbach*', porque no se puede saber si alguna vez se detendrá hasta que lo haga. Pero el ajetreado '*juego de Castores*' arroja algo de luz sobre el proble- ma. Si fuera posible calcular $BB(27)$, eso proporcionaría un techo sobre cuánto tiempo tendríamos que esperar para que la "*Conjetura de Goldbach*" se resuelva automáticamente. Eso es porque $BB(27)$ corresponde al *número máximo de pasos* que esta '*Máquina de Turing*' (MT) de *27* reglas tendría que ejecutar para detenerse (si es que alguna vez lo hace). Si se conociéra ese número, se podría ejecutar la '*Máquina de Turing*' exactamente esa cantidad de pasos. Si se detuviera en ese punto, se sabría que *la conjetura de Goldbach es falsa.* Pero si diera tantos pasos y no se detuviera, se sabría con certeza que nunca lo haría, demostrando así que la Conjetura es cierta.

El problema es que $BB(27)$ es un número tan incomprensiblemente enorme que *nisiquiera escribirlo*, y *mucho menos ejecutar* la *máquina falsificadora de Goldbach durante tantos pasos*, *no es ni remotamente posible en nuestro universo físico.* Sin embargo, ese *número incomprensiblemente enorme sigue siendo* un *número exacto* cuya magnitud, según *Aaronson*, representa "*una declaración sobre el conocimiento actual*" de la teoría de números.

El mismo *Aaronson* en *2016*, estableció un resultado similar en colaboración con *Yuri Matiyasevich* y *Stefan O'Rear.* Identificaron una 'Máquina de Turing' de 744reglas que se detiene *si y solo si la 'Hipótesis de Riemann'* es falsa.La Hipótesis de Riemann también se refiere a la distribución de números primos y es uno de los "*Problemas del Milenio*" del *Clay Mathematics Institute* con valor de $ 1 millón de USD. La máquina de *Aaronson* entregaría una solución automática en pasos de $BB(744)$. (Funciona esencialmente mediante *el mismo procedimiento sin sentido* que la máquina de Gold- bach, iterando hacia arriba hasta encontrar un contraejemplo).

Por supuesto, $BB(744)$ es un número *aún más inalcanzable* que $BB(27)$. Pero traba jar para precisar algo más fácil, como $BB(5)$, "*puede generar algunas nuevas pregun- tas* de la *Teoría de Números* que son interesantes por derecho propio" (Aaronson) Por ejemplo, el matemático *Pascal Michel* demostró en 1993 que la máquina de Turing de *cinco reglas* que posee registros exhibe un comportamiento similar al de la función descrita en la '*Conjetura de Collatz*', otro famoso problema abierto en la Teoría de Números.

"*Gran parte de las matemáticas se pueden codificar como una pregunta: "¿Ésta 'Máquina de Turing' se detiene o no?*" (*Aaronson*) "*Si conociera todos los números de los 'castores laboriosos', entonces podría resolver todas esas conjeturas*".

Más recientemente, *Aaronson* ha utilizado un criterio deducido de un 'castor *laborioso*' para medir lo que él llama "*el umbral de la incognoscibilidad* " para sistemas completos de matemáticas.

Los famosos *Teoremas de Incompletez* de *Gödel* de *1931* demostraron que cualquier conjunto de axiomas básicos que pudieran servir como una posible base lógica para las matemáticas está condenado a uno de dos destinos: o los *axiomas* serán *inconsistentes*, lo que conducirá a *contradicciones* (tipo $0 = 1$), o estarán *incompletos*, incapaces de probar algunas afirmaciones verdaderas sobre los números (como $2 + 2 = 4$). El sistema axiomático que sustenta a casi todas las matemáticas modernas, conocido como *Teoría de Conjuntos* de *Zermelo-Fraenkel* (*ZF*), tiene sus propios límites de *Gödel*; y *Aaronson* quería usar el ajetreado '*juego del castor*' para establecer en dónde se encuentran.

2.5 De lo último.

En 2016, *Aaronson* y su estudiante de posgrado *Adam Yedidia* [16] especificaron una *máquina de Turing* con 7910 *reglas* que solo se detendría si la teoría de conjun- tos *Zermelo-Fraenkel* (*ZF*) fuera *inconsistente*. Esto significa que $BB(7910)$ es un cálculo que elude los axiomas de la teoría de conjuntos *ZF*. Esos axiomas no pueden usarse para probar que $BB(7910)$ representa un número en lugar de otro, lo que es como no poder probar que $2 + 2 = 4$ en lugar de 5.

Luego de lo anterior ya planteado, apareció *O'Rear*, quien ideó una máquina mucho más simple con 748 reglas y *se detendría* si *Zermelo-Fraenkel* (*ZF*) fuera *inconsistente*, esencialmente moviendo el umbral de incognoscibilidad más cerca, de$BB(7910)$ a $BB(748)$. "*Eso es algo dramático, que la cantidad [de reglas] no sea completamente ridícula*" (*Harvey Friedman* - lógico matemático y profesor emérito de la Univ. Estatal de Ohio) "*Creo que tal vez* 50 *reglas es la respuesta correcta*". *Aaronson* sospecha que el verdadero umbral puede estar tan cerca como $BB(20)$.

Ya sea cerca o lejos, esos *umbrales de incognoscibilidad* definitivamente existen. "*Esta es la visión del mundo que hemos tenido desde Gödel* " (Aaronson) "La función del *castor laborioso* es otra forma de concretarlo".

Las declaraciones y afirmaciones citadas fueron tomadas al divulgador de las ciencias *John Pavlus* en [17].

La función 'Castor Laborioso' (Busy Beaver (BB)), con su incomprensible crecimiento ultrarápido, sigue cautivado a generaciones de informáticos, matemáticos y aficionados. En el panorama actual, se ofrece una visión personal de Aaranson sobre la función BB. A 58 años después de su introducción, haciendo hincapié en las percepciones menos conocidas, el progreso reciente y, especialmente, los problemas abiertos favoritos. Los ejemplos de tales problemas incluyen: ¿cuándo la función BB excede por primera vez a la función de Ackermann? ¿Es el valor de $BB(20)$ independiente de la teoría de conjuntos? ¿Se puede probar que $BB(n + 1) > 2BB(n)$ para n suficientemente grande? Dado $BB(n)$, ¿cuántos bits de aviso se necesitan para calcular $BB(n+1)$? ¿Todos los 'Castores Laboriosos' se detienen en todas las entradas, no solo en la entrada 0? ¿Es decidible, dado n, si $BB(n)$ es par o impar? [3].

2.6 REFERENCIAS.

References

[1] *Shen Lin, Tibor Radó*; **Computer Studies of Turing Machine Problems**; *Journal of the ACM, Vol. 12, No. 2, April 1965, https://doi.org/10.1145/321264.321270.*

[2] *Alan Turing*; **On Computable Numbers, With An Application To The ENTSCHEIDUNGS Problem;** *Proceedings of the London MathematicalSocie- ty, Volume s2-42, Issue 1, 1937, Pages 230–265, 12 November, 1936. https://doi.org/10.1112/plms/s2-42.1.230*

[3] *Scott Aaronson*; **The Busy Beaver Frontier**; *News ACM SIGACT NewsVol. 51, No. 3, News September 2020*, https://doi.org/10.1145/3427361.3427369

[4] *A. M. Ben-Amram and H. Petersen.* **Improved bounds for functions related to Busy Beavers;** *Theory Comput. Syst., 35(1):1-11, 2002.* http://citeseerx.ist.psu.edu/ viewdoc/download? doi=10.1.1.136.5997&rep=rep1&type=pdf. Google ScholarCross Ref

[5] *A. H. Brady*; **The determination of the value of Rado's noncomputable function (k) for four-state Turing machines**; Mathematics of Computation, 40(162):647-665, 1983. Google Scholar

[6] *G. Chaitin*; **To a mathematical theory of evolution and biological creativity**; *Technical Report 391, Centre for Discrete Mathematics and Theoretical Computer Science, 2010.* www.cs.auckland.ac.nz/research/groups/CDMTCS/researchreports/391greg.pdf. 91greg.pdf. Google Scholar

[7] *J. H. Conway*; **Unpredictable iterations**; In Proc. 1972 Number Theory Conference, University of Colorado, Boulder, pages 49-52, 1972. Google Scholar

[8] *R. Goodstein*; **On the restricted ordinal theorem**. J. Symbolic Logic, 9: 33-41, 1944. https://pdfs.semanticscholar.org/4d96/c256994f190617a34aac56c9b9bfb23f43d9.pdf. Google ScholarCross Ref

[9] *M. W. Green*; **A lower bound on Rado's Sigma function for binary Turing machines**. *In Proc. IEEE FOCS, pages 91-94, 1964.* Google Scholar Digital Library.

[10] *L. Kirby and J. Paris*; **Accessible independence results for Peano arithmetic**; *Bulletin of the London Mathematical Society, 14:285-293, 1982.* https://faculty.baruch.cuny.edu/lkirby/accessible_independence_results.pdf. Google Scholar Cross Ref

[11] *P. Kropitz*; **Busy Beaver Problem**; *Bachelors thesis, Charles University in Prague, 2011. In Czech.* https://is.cuni.cz/webapps/zzp/detail/49210/. Google Scholar

[12] *H. Marxen* and *J. Buntrock*; **Attacking the Busy Beaver 5**; *Bulletin of the EATCS, 40:247{251, 1990.* http://turbotm.de/~heiner/BB/mabu90.html. Google Scholar.

[13] *P. Michel*; **Busy beaver competition and Collatz-like problems**; *Archive for Mathematical Logic, 32:351-367, 1993.* Google ScholarCross Ref

[14] *P. Michel*; **The Busy Beaver competition: a historical survey**; https://arxiv.org/abs/0906. 3749, 2019. Google Scholar

[15] *R. M. Solovay*; **Hyperarithmetically encodable sets**; *Transactions of the AMS, 239:99-122, 1978.* https://www.ams.org/journals/tran/1978–239-00/S0002–9947–1978-0491103–7/ S0002–9947–1978-0491103–7.pdf. Google ScholarCross Ref

[16] *A. Yedidia* and *S. Aaronson*; **A relatively small Turing machine whose behavior is independent of set theory**; *Complex Systems, (25):4, 2016.* https://arxiv.org/abs/1605.04343. Google Scholar.

[17] *John Pavlus*; **How the Slowest Computer Programs Illuminate Math's Fundamental Limits**; *Quanta Magazine, Abstractions Blog, 10 december 2020.*

[18] *KurtGödel*;**On Formally Undecidable Propositions Of 'Principia Mathematica' and Related Systems**, *Translated by B Meltzer, Basic Books,1962.*

[19] *Douglas Hofstadter;* **Gödel, Escher, Bach: An Eternal Golden Braid**, *1979* (existe versión castellana de Gedisa)

[20] *Roger Penrose;* **Emperor's New Mind;** (*1990*)

[21] *Ernst Nagel and James Newman*, **Gödel's Proof**, NY, Univ-ty Press, 1958. (existe versión en español de la UNAM)

[22] *Scott Aaronson*; **The Busy Beaver Frontier**; Association for Computing Machinary (ACM) SIGACT News September 2020, Vol. 51, No. 3 https://doi.org/10.1145/3427361.3427369

2.7

.1 The First Appendix

The appendix fragment is used only once. Subsequent appendices can be created using the Section Section/Body Tag.