5. Teoría matemática de la computación

Introducción

- Preguntas que inquietan a los científicos:
  - ¿Qué problemas pueden resolverse usando una computadora? ¿Cuáles no?
  - ¿En cuánto tiempo puede calcularse la solución de un problema usando un lenguaje en particular?
  - ¿Un lenguaje es superior a otro?
  - ¿Cuál es el número mínimo de instrucciones para que un lenguaje resuelva un problema?
  - ¿Puede saberse si un programa terminará o se ejecutará para siempre?

5.1 Lenguaje simple

Instrucciones del lenguaje simple

- Sólo se requieren de 3 instrucciones:
  - Incremento **inc**
  - Decremento **dec**
  - Ciclo **mientras**
- Sólo se utilizan datos enteros
- Otros símbolos '{' y '}'

Instrucción incremento

- Añade 1 a la variable
- Formato:

  incr X

Instrucción decremento

- Sustrae 1 a la variable
- Formato:

  decr X

Instrucción de ciclo

- Repite un conjunto de instrucciones cuando el valor de la variable es no cero
- Formato:

```
mientras X
{
    Acciones
}
```

0 falso
1 verdadero

El lenguaje es poderoso aunque no necesariamente eficiente

Uso de Macros

A través de ellas se puede simular instrucciones más complejas sin la necesidad de repetir el código

Macros

Limpieza: x ← 0

```
mientras x
{
        decr x
}
```

Asignación: x ← n

```
x ← 0  //macro de limpieza
incr x
incr x
....
incr x
```


Copiar: y ← x

```
y ← 0
temp ← 0
mientras x
{
        incr y
        decr x
        incr temp
}
mientras temp
{
        decr temp
        incr x
}
```

Macros

Suma: z ← x + y

```
z ← x  //copiar
temp ← y
mientras temp
{
        incr z
        decr temp
}
```

Macros

Multiplicación: z ← x * y

```
z ← 0
temp ← y
mientras temp
{
        z ← x+y      z ← x+z
        decr temp
}
```

Potencia: z ← x ^ y

```
z ← 1
temp ← y
mientras temp
{
        z ← x*y      z ← x*z
        decr temp
}
```


Macros

Complemento

comp (x)

Si x es cero lo cambia a 1
Si x no es cero lo camiba a 0

```
//cambia x a 0, si x≠0
temp ← 1
mientras x
{
        x ← 0
        temp ← 0
}
//cambia x a 1
mientras temp
{
        incr x
        decr temp
}
```

Macros

Selección

si x
a1
otro
a2

```
temp ← x
//si x ≠0
mientras temp
{
        a1
        temp ← 0
}
temp ←x
comp temp
mientras temp
{
        a2
        temp ← 0
}
```

Conclusión

Si no se puede resolver un problema en este lenguaje no se podrá resolver en ningún otro

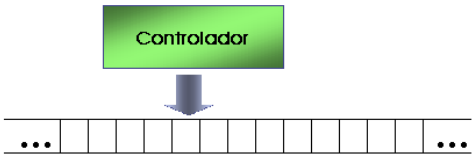
5.2 Máquina de Turing

Introducida por Alan M. Turing en 1936 para resolver problemas computables.

Base de las computadoras actuales

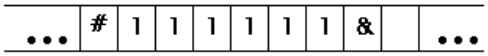
Componentes de la Máquina de Turing

- Una cinta infinita
- Un controlador
- Una cabeza de lectura/escritura



Cinta

- La cinta contiene una secuencia de caracteres válidos:
  - #, principio de un número
  - &, fin de un número
  - 1, número expresado en unario
  - Espacio en blanco



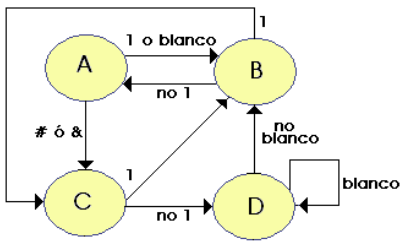
Cabeza de lectura/escritura

- Señala a un símbolo en la cinta, llamado el "símbolo actual"
- Lee y escribe un símbolo a la vez en la cinta
- Después de leer y escribir se mueve:
  - Izquierda
  - Derecha
  - Permance en su lugar
- Todo se realiza bajo instrucciones del controlador

Controlador

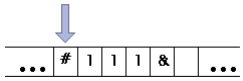
- Equivalente al CPU
- Es un **autómata de estado finito**, máquina con un número finito predeterminado de estados que se mueve de un estado a otro de acuerdo a la entrada. Sólo puede estar en un estado

Autómata y Tabla de Transición

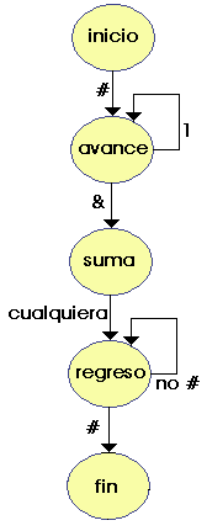


Estado actual	Lee	Escribe	Se mueve	Estado nuevo
A	1 ó blanco	#	derecha	B
A	# ó &	&	izquierda	C
B	1	1	izquierda	C
B	no 1	lo mismo que lee		A
C	1	blanco	derecha	B
C	no 1	1	derecha	D
D	no blanco	lo mismo que lee	derecha	B
D	blanco	1	izquierda	D

Ejemplo: incremento

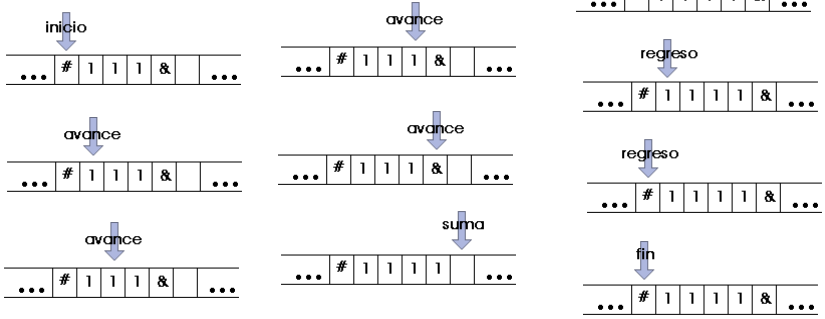


Estado actual	Lee	Escribe	Se mueve	Estado nuevo
inicio	#	#	derecha	avance
avance	1	1	derecha	avance
avance	&	1	derecha	suma
suma	cualquiera	&	izquierda	regreso
regreso	no #	lo mismo que lee	izquierda	regreso
regreso	#	#		fin



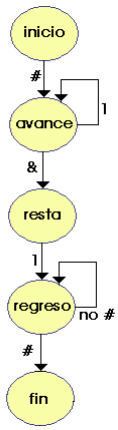
Pasos en la instrucción incr x

Estado actual	Lee	Escribe	Se mueve	Estado nuevo
inicio	#	#	derecha	avance
avance	1	1	derecha	avance
avance	&	1	derecha	suma
suma	cualquiera	&	izquierda	regreso
regreso	no #	lo mismo que lee	izquierda	regreso
regreso	#	#		fin



Instrucción de decremento

Estado actual	Lee	Escribe	Se mueve	Estado nuevo
inicio	#	#	derecha	avance
avance	1	1	derecha	avance
avance	&	blanco	izquierda	resta
resta	1	&	izquierda	regreso
regreso	no #	lo mismo que lee	izquierda	regreso
regreso	#	#		fin



Ejercicio:
Realizar los pasos de la instrucción

Conclusión

- La máquina de Turing es tan poderosa como el lenguaje simple
- Cualquier problema que puede resolverse con el lenguaje simple también puede resolver por una máquina de Turing y viceversa

Números de Gödel

- Un número sin asignar se asigna a cada programa que puede escribirse en un lenguaje y a esto se le llama número de Gödel (Kurt Gödel)
- Ventajas:
  - Los progrmas pueden utilizarse como un solo elmento de datos que sirva como entrada a otro programa
  - Se puede hacer referencia a los progrmas simplemente mediante sus representaciones de enteros
  - La numeración puede usarse para probar que algunos problemas ono pueden resolverse usando una computadora al mostrar que el número total de problemas en el mundo es mucho más grande que el número total de programas que puedan escribirse.

Numeración simple

- Utilizando el lenguaje simple, X, X1, ..., X9 son variables

Símbolo	Código Hexa	Símbolo	Código Hexa
1	1	9	9
2	2	incr	A
3	3	decr	B
4	4	mientras	C
5	5	{	D
6	6	}	E
7	7	x	F
8	8		

Representación de un programa

- Usando la tabla se puede representar cualquier programa escrito en el lenguaje simple mediante un entero positivo único:
  1. Reemplazar cada símbolo con el código hexadecimal que le corresponde en la tabla
  2. Interpretar el número hexadecimal resultante como un entero sin asignar

Ejemplo

- Número de Gödel para el programa incr x
- Solución

  incr x

  A F

  1010 1111

  175 (decimal)

Símbolo	Código Hexa	Símbolo	Código Hexa
1	1	9	9
2	2	incr	A
3	3	decr	B
4	4	mientras	C
5	5	{	D
6	6	}	E
7	7	x	F
8	8		

Interpretación de un número

- Para mostrar que es único, el número de Gödel se interpreta:
  - Convertir el número a hexadecimal
  - Interpretar cada dígito hexadecimal como un símbolo utilizando la tabla (ignorar los 0)

Mientras que cualquier programa en simple puede representarse mediante un número, no todos los números pueden interpretarse como un programa válido

Ejemplo

- Interpretar 3058 como un programa
  - En binario: 1011 1111 0010
  - En hexadecimal: B F 2
  - Se interpreta como: dec x 2

El problema de paro

- Una repetición puede no terminar (parar o detenerse), es decir un programa puede ejecutarse por siempre si éste tiene un ciclo infinito.
- Ejemplo:

```
x ←1
mientras x
{
}
```

¿Puede escribirse un programa que evalúe si un programa, representado por su número de Gödel, terminará o no?

- La existencia de este programa ahorraría a los programadores mucho tiemp.
- La ejecución de un programa sin saber si éste se detiene o no, es una tarea tediosa.
- Se ha probado que este programa no puede existir (gran decepción para los programadores)
- Se dice que “El problema de paro no tiene solución”, en vez que no existe y nunca podrá existir

Problemas con solución y sin solución

- En general los problemas se dividen en dos categorías:
  - Los que tienen solución
    - Polinomiales
    - No polinomiales
  - Los que no tienen solución

Problemas sin solución

- Existe un número infinito de problemas que no pueden resolverse mediante una computadora, uno de ellos es el problema de paro. Un método para probar que un problema no tiene solución es mostrar que si ese problema tiene solución, el problema de paro también tiene solución.

Problemas con solución

- Existen muchos problemas que pueden resolverse mediante una computadoras. Con frecuencia, se desea saber cuánto tiempo tomara resolver ese problema, es decir ¿qué tan complejo es?
- La complejidad se mide en:
  - Tiempo de ejecución
  - Memoria requerida

Complejidad de los problemas con solución

- _Una forma de calcular es a través del número de operaciones que se realizan cuando la computadora ejecuta el programa, de tal forma que es independiente de la velocidad de la máquina
- Esta medida normalmente depende del tamaño de la entrada

Notación O

- Número de operaciones se da como función del tamaño de la entrada, solo es importante entradas grandes
- No se consideran factores constantes ni los que dependen de una máquina en particular
- O(n) significa que un programa realiza n operaciones para un tamaño n de entrada
- $O(n^2)$, se realizan n^2 operaciones para una entrada de tamaño n

Ejemplo

- Suponga 3 programas para resolver el mismo problema:
  - $O(\log_{10} n)$ $\log_{10}$ 1'000,000 = 6 → 6μs
  - O(n) 1'000,000/1'000,000 = 1 → 1μs
  - $O(n^2)$ 10^{12} → 277 horas
- Si el tamaño de la entrada es de un millón
- ¿Cuánto tiempo tardan en ejecutarse? Si se considera que se realiza una instrucción en un milisegundo, es decir un millón por segundo.

Problemas polinomiales

- Son programas con una complejidad:
  - O (log n)
  - O (n)
  - $O(n^2)$
  - $O(n^3)$
  - ...
  - $O(n^k)$
- Con las velocidades actuales para un tamaño razonable de entrada (1000 a 1 millón) se pueden obtener soluciones en un tiempo razonable

Problemas no polinomiales (NP)

- Complejidades:
  - $O(10^n)$
  - O(n!)
  - $O(e^n)$
- Pueden obtenerse soluciones para entradas pequeñas (menor a 100)
- Si son grandes pueden tardar años, siglos, etc.

Ejercicios

- Escribir en el lenguaje simple macros para:
  - z ← x –y
  - si x < y a1 otro a2
  - si x > y a1 otro a2
  - mientras x > y { acciones }
  - mientras x = y {acciones }
- Mostrar el diagrama de transición para la máquina de Turing para simular:
  - x ← 0
  - x ← n
  - x ← y
  - x ← y + z
  - x ← y * z
  - comp x
  - si x < y a1 otro a2
- Calcular el número de Gödel para
  - x ← 0
  - x ← n
  - x ← y + z