Algoritmos y Estructuras de Datos II
Clase 2

Carreras:

Licenciatura en Informática

Ingeniería en Informática

2024

Algoritmo
Definición de Donald Knuth

Hasta aquí la definición sobre algoritmos ha sido bastante imprecisa y hasta muy poco rigurosa desde el punto de vista matemático.

Donald Knuth afirma que esto constituye una base muy inestable sobre la cual erigir cualquier teoría sobre algoritmos.

Según este autor el concepto de algoritmo puede basarse firmemente en términos de teoría matemática de conjuntos.

De este modo propone una ***noción formal para los algoritmos***.

Algoritmo
Definición de Donald Knuth

Noción "formal":

Un ***método de cálculo*** es una cuaterna (Q,I,W,f) donde:

- Q es un conjunto de estados de cálculo
- Q contiene a I y a W
- I es el conjunto de estado de entrada
- W es el conjunto de estados de salida
- f es la regla de cálculo
- $f: Q \rightarrow Q$ con $f(w)=w$ para todo w perteneciente a W

Algoritmo. Definición de Knuth

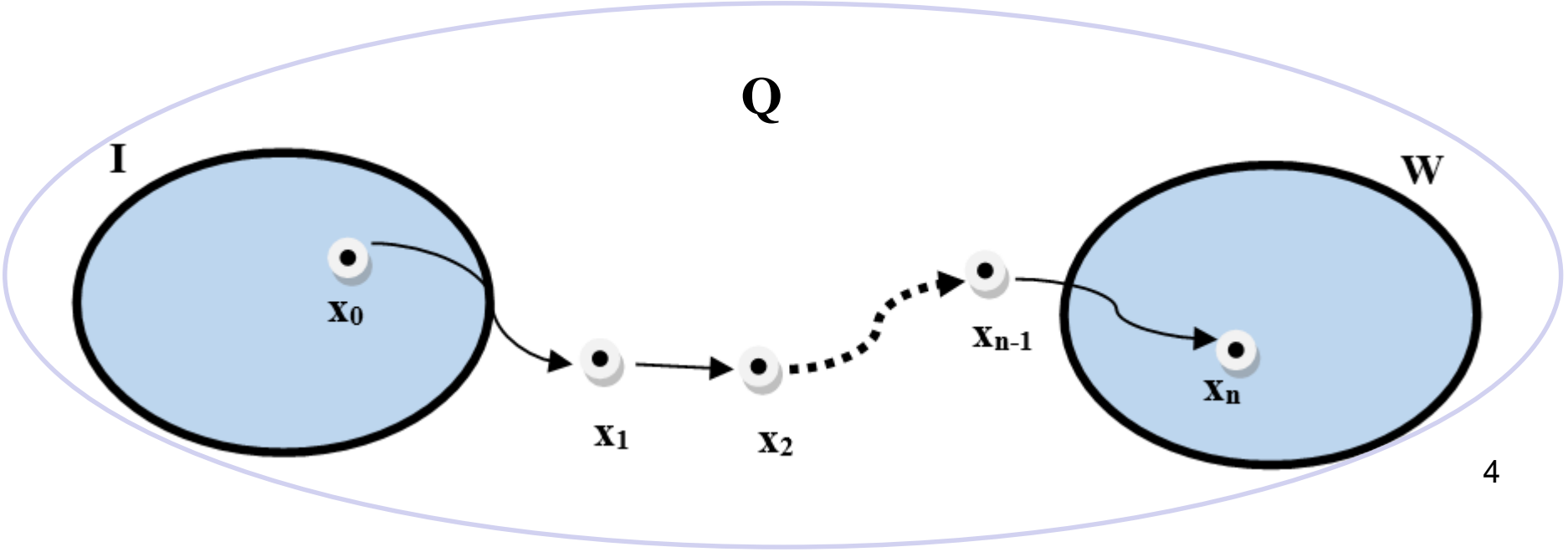
Cada entrada x del conjunto I define una ***secuencia de cálculo***:

$$x_0, x_1, x_2, \ldots,$$

Donde $x_0 = x$ pertenece a I , y para todo $k \geq 0$:

$$x_{k+1} = f(x_k)$$

La secuencia de cálculo finaliza en n pasos si n es el menor entero con $x_n \in W$, y en ese caso se dice que proporciona la salida x_n de x .



Algoritmo. Definición de Knuth

Dada una secuencia de cálculo y su entrada x , entonces pueden ocurrir 2 cosas:

- Que la secuencia de cálculo **finalice después de n pasos**.
- Que la secuencia de cálculo **NO finalice nunca**.

DEFINICIÓN:

Un *algoritmo* es una secuencia de cálculo que finaliza en un número finito de pasos para todas sus entradas, es decir para todo $x \epsilon I$.

Algoritmo de Euclides

Entrada: m,n, dos numeros enteros positivos

Salida: n, entero positivo MCD de m y n

Auxiliar: r, entero positivo o nulo

E0. Leer (m,n)

E1. r ← resto(m/n)

E2. Mientras r ≠ 0 Hacer

m ← n

n ← r

r ← resto (m/n)

E3. Escribir (n)

E4. Fin

Es un algoritmo ?

El algoritmo de Euclides siempre termina y el último resto no nulo que se obtiene es el máximo común divisor entre **m** y **n**.

Algoritmo de Euclides

Entrada: m, n: dos números enteros positivos
Salida: n: entero positivo MCD de m y n
Auxiliar: r: enterro positivo o nulo
E0. Leer (m,n)
E1. r ← resto(m/n)
E2. Si r = 0 entonces
Escribir (n)
Fin
E3. Si r ≠ 0 entonces
m ← n
n ← r
volver a paso E1

Algoritmo de Euclides:

Entrada: m, n: dos números enteros positivos
Salida: n: entero positivo MCD de m y n
Auxiliar: r: entero
E0. Leer (m,n)
E1. r ← resto(m/n)
E2. Si r=0 entonces
    Escribir (n)
    Fin
E3. Si r≠ 0 entonces
    m ← n
    n ← r
    volver a paso E1

Entrada: $I = \{(m,n) / m,n \in \mathrm{N}^+\}$

Salida: $W = \{(n) / n \in \mathrm{N}^+\}$

Estados: $Q = I \cup W \cup \{(m,n,r,E1),(m,n,r,E2),(m,n,r,E3) / m,n \in \mathrm{N}^+, r \in \mathrm{N}\}$

Reglas:

$$f(n) = (n)$$

$$f(m,n) = (m,n,r,E1)$$

$$f(m,n,r,E1) = \left(m,n,resto\left(\frac{m}{n}\right),E2\right)$$

$$f(m,n,r,E2) = \begin{cases}(m,n,r,E3) & si\ r \neq 0\\ (n) & si\ r = 0\end{cases}$$

$$f(m,n,r,E3) = (n,r,r,E1)$$

Algoritmo de Euclides

Ejemplo: m=15 y n=9

$x_0 = (15,9)$
$x_1 = f(x_0) = (15,9,0, E1)$
$x_2 = f(x_1) = (15,9,6, E2)$
$x_3 = f(x_2) = (15,9,6, E3)$
$x_4 = f(x_3) = (9,6,6, E1)$
$x_5 = f(x_4) = (9,6,3, E2)$
$x_6 = f(x_5) = (9,6,3, E3)$
$x_7 = f(x_6) = (6,3,3, E1)$
$x_8 = f(x_7) = (6,3,0, E2)$
$x_9 = f(x_8) = (3)$

$$f(n) = (n)$$

$$f(m,n) = (m,n,r,E1)$$

$$f(m,n,r,E1) = \left(m,n,resto\left(\frac{m}{n}\right),E2\right)$$

$$f(m,n,r,E2) = \begin{cases} (m,n,r,E3) & si\ r \neq 0 \\ (n) & si\ r = 0 \end{cases}$$

$$f(m,n,r,E3) = (n,r,r,E1)$$

Algoritmia - Algoritmo

- La algoritmia estudia técnicas para diseñar *algoritmos eficientes*.

- Se dice que un algoritmo es “más eficiente” que otro cuando gasta menos recursos de tiempo y memoria para resolver un ***problema***.

- Por lo tanto la algoritmia estudiará, además de las técnicas de diseño de algoritmos, técnicas para medir la eficiencia de los algoritmos, de modo que en la resolución de un problema concreto se pueda elegir el mejor algoritmo.

Eficiencia de Algoritmos

- La comparación entre algoritmos no es tan sencilla porque puede ser que un algoritmo sea más eficiente que otro para un determinado conjunto de datos de entrada pero menos eficiente para otros.

- Por lo tanto se puede estudiar el rendimiento del algoritmo
    - en el caso ***más favorable***,
    - en el ***peor caso***
    - y también en el ***caso medio***.

Algoritmo - Problema

PROBLEMA:

- *instancia de un problema*
- *datos de entrada de una instancia (I)*
- *solución (O)*

ALGORITMO:

- *técnica para la resolución de un problema*
- *función f tal que f (I) = O*

Análisis de algoritmos

- Se analizarán algunas temas básicos relacionadas con el análisis de algoritmos.
- Esencialmente las siguientes cuestiones:
- **Algunos algoritmos son eficientes y otros no. *¿Cómo medir la eficiencia de un algoritmo?***
- **Algunos problemas son fáciles de resolver y otros no. *¿Cómo medir la dificultad de un problema?***
- ***¿Cómo saber si un algoritmo es óptimo para resolver un problema?***
- Se mostrará que todos estos puntos están relacionados entre sí.

La eficiencia de algoritmos

Cuando se tiene un problema a resolver es posible que estén disponibles varios algoritmos para llegar a la solución.

- Evidentemente se quiere seleccionar el ***mejor***, esto plantea el dilema de decidir entre varios algoritmos cual es preferible.

- El ***enfoque empírico*** (a posteriori) para seleccionar un algoritmo consiste en programar las técnicas e ir probándolas en distintos casos con la ayuda de la computadora.
- El ***enfoque teórico*** (a priori) consiste en determinar matemáticamente la cantidad de recursos necesarios para cada uno de los algoritmos como función del tamaño de los casos considerados.

- Los recursos que más interesan son: el ***tiempo de computación*** y el ***espacio de almacenamiento***.

Enfoque teórico

- La ventaja de la aproximación teórica es que:
  - *no depende ni de la computadora que se esté utilizando,*
  - *ni del lenguaje de programación,*
  - *ni siquiera de las habilidades del programador*.

- Lo más significativo es que permite estudiar la eficiencia del algoritmo cuando se utilizan *casos de todos los tamaños*.

- Dado que suele suceder que los algoritmos recién descubiertos empiezan a comportarse mejor que sus predecesores sólo cuando ambos se utilizan para una *entrada de gran tamaño*, este último punto puede resultar especialmente importante.

Complejidad Computacional

*La **complejidad de espacio** de un algoritmo es una función que calcula los requisitos de almacenamiento en función del tamaño de la entrada de un problema.*

- Para medir la cantidad de espacio que utiliza un algoritmo en función del tamaño de la entrada la unidad natural es el ***bit.***

- Independiente de la máquina que se esté usando, la noción de un bit para almacenamiento está bien definida.

Complejidad Computacional

*La **complejidad de tiempo** de un algoritmo es una función que calcula el tiempo de ejecución en función del tamaño de la entrada de un problema.*

- Para medir un algoritmo en términos del tiempo que necesita para su ejecución, no existe una opción tan evidente como el bit.
- Cómo medir el tiempo de ejecución en función del tamaño de la entrada de los datos del problema?

Complejidad Computacional

Para responder a la pregunta:

Cómo medir el tiempo de ejecución en función del tamaño de la entrada de los datos del problema?

Se debe especificar un modelo de computación para la ejecución de esos algoritmos.

MODELOS DE COMPUTACION

- RAM *(Máquina de acceso aleatorio)*
- RASP *(Máquina de acceso aleatorio con programa almacenado)*
- MAQUINA DE TURING

RAM (random access machine)

- Es el modelo que se usará para evaluar la complejidad de un algoritmo.

- *Un modelo de máquina de acceso directo RAM es una computadora de un acumulador cuyas instrucciones no pueden modificarse a sí mismas.*

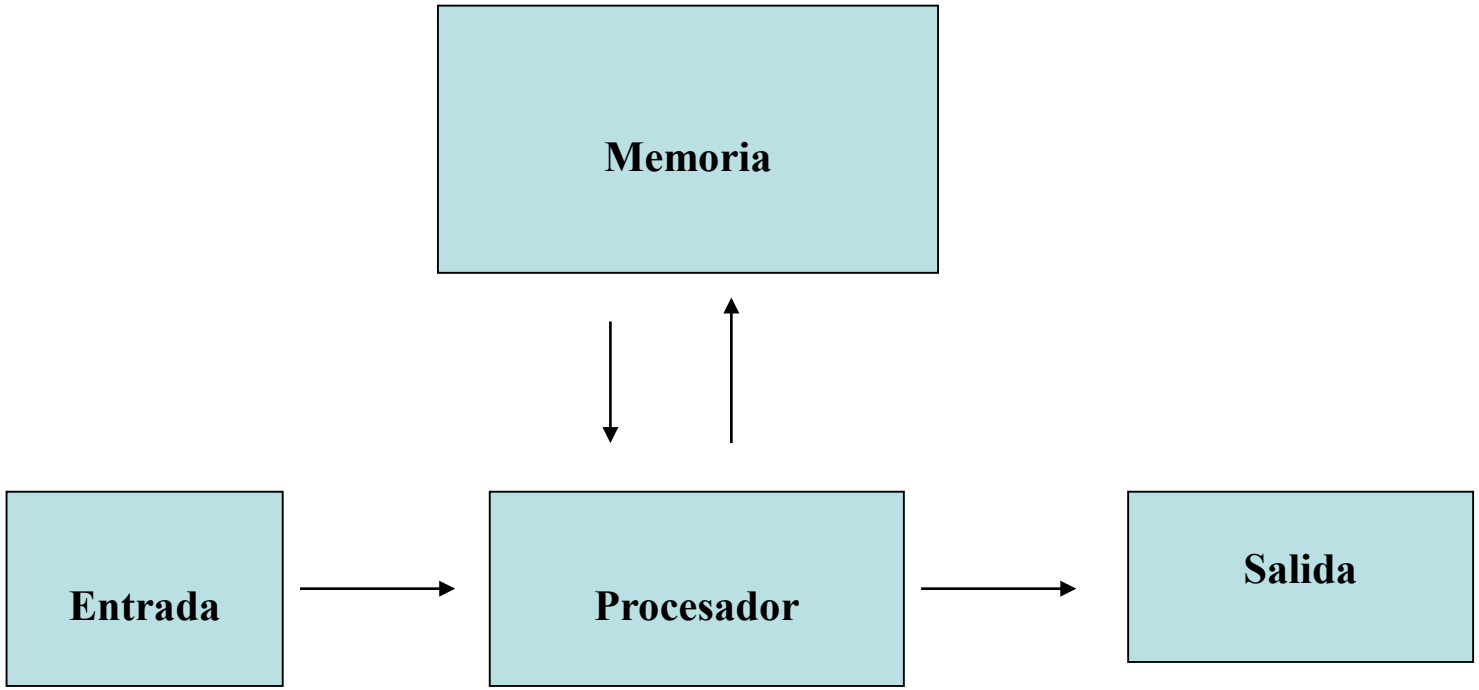
- Consta de:

  unidad de memoria
  unidad de entrada
  procesador
  unidad de salida
  conjunto de instrucciones

- Un ***programa*** RAM es una *secuencia finita de estas instrucciones*

RAM

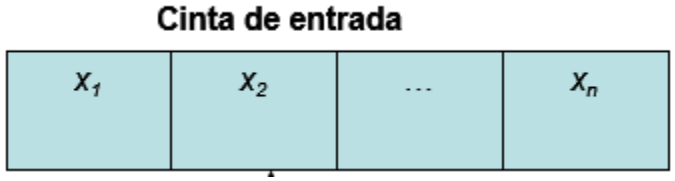
En un esquema:



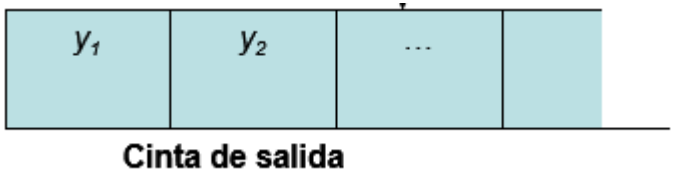
RAM

En este modelo se supone que:

- La ***entrada*** es una secuencia representada por una cinta de cuadrados en cada uno de los cuales está almacenado un número entero.
- Cuando un símbolo se lee de la cinta, la cabeza ***avanza*** al próximo cuadrado.



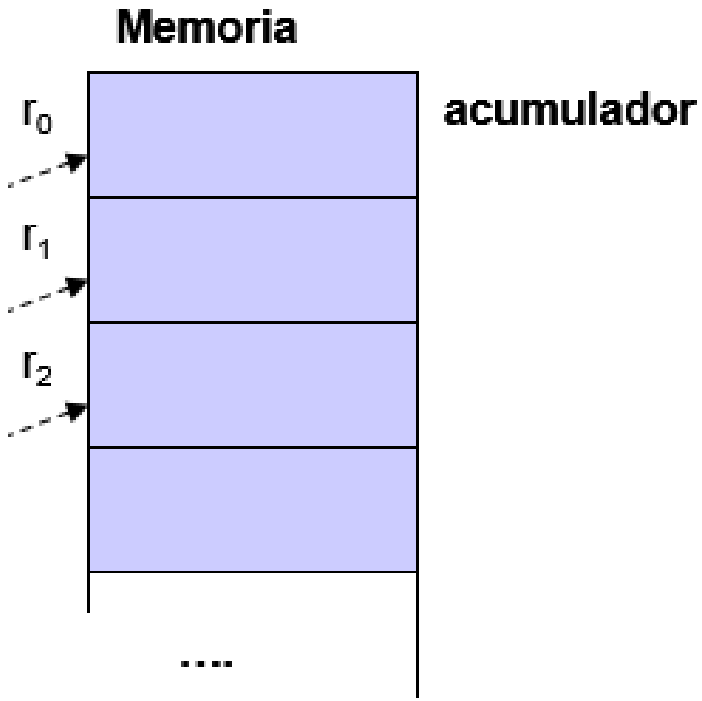
- La ***salida*** es una cinta de cuadrados, que está inicialmente en blanco, en cada uno de los cuales se puede escribir un entero.
- Cuando un símbolo se escribe en la cinta, la cabeza avanza al próximo cuadrado. Cuando ya se escribió un símbolo, no se puede cambiar.



RAM

- La ***memoria*** es una sucesión (cantidad ilimitada) de registros **r0,r1,..,** cada uno de los cuales puede almacenar un entero de tamaño arbitrario.

- Los cálculos se hacen en el primero de ellos, $r0$, llamado ***acumulador.***

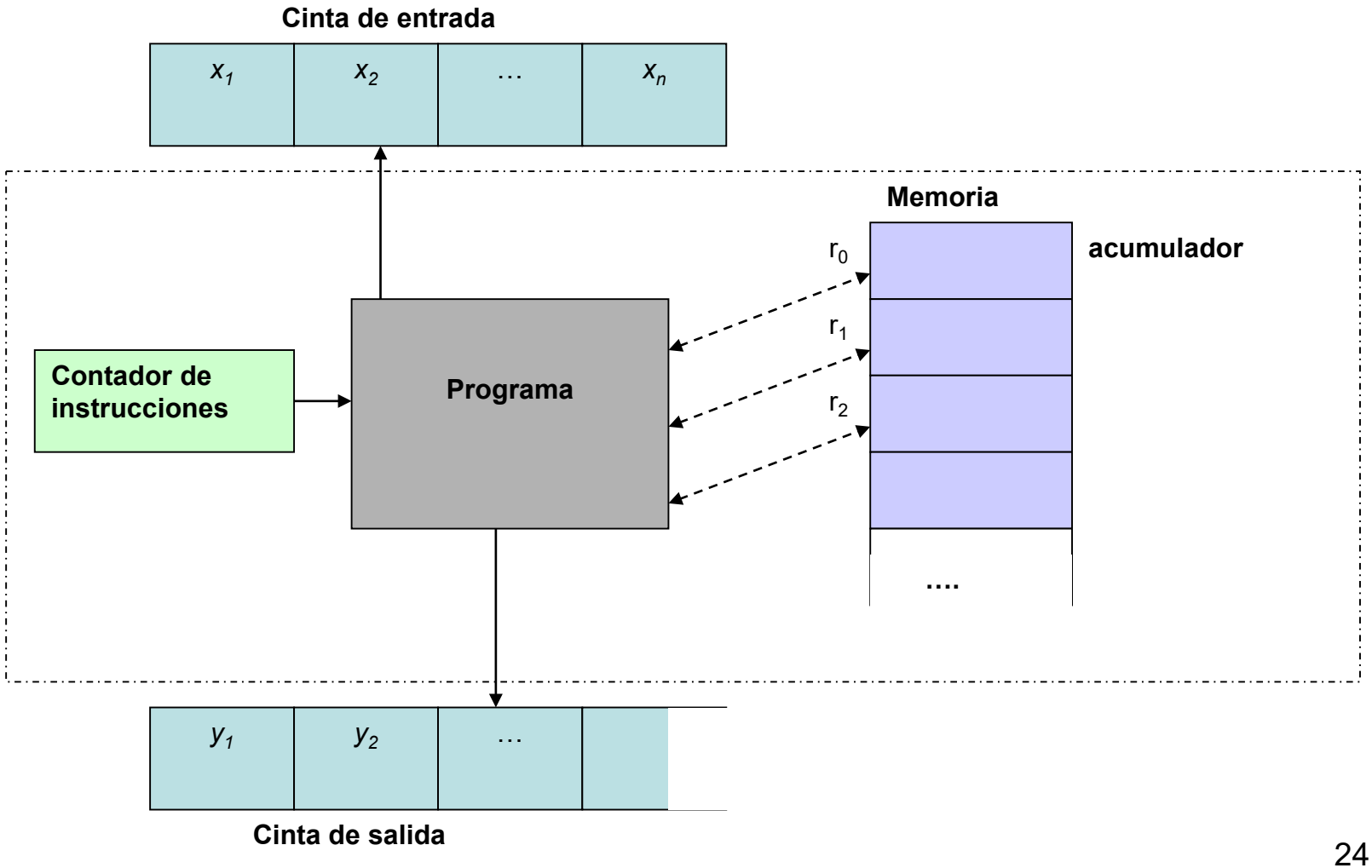


RAM

- El ***programa*** es una secuencia de instrucciones.
- El programa no se almacena en la memoria y no se modifica a si mismo.

- Cada instrucción tiene un ***código de operación*** y una ***dirección.***
- El ***conjunto de instrucciones*** disponibles es similar al de las *computadoras reales:* **LOAD, STORE, ADD, SUB, MULT, DIV, READ, WRITE, JUMP, JGTZ, JZERO** y **HALT** (operaciones aritméticas, de entrada/salida, de bifurcación*…)*

RAM


Cinta de entrada
x_1
x_2
…
x_n
Memoria
acumulador
r_0
r_1
r_2
....
Contador de instrucciones
Programa
y_1
y_2
…
Cinta de salida

RAM

Instrucciones de una máquina RAM

(código – dirección - explicación)

LOAD	*operando*	Carga el operando en el acumulador
STORE	*operando*	Carga el acumulador en un registro
ADD	*operando*	Suma el operando al acumulador
SUB	*operando*	Resta el operando al acumulador
MULT	*operando*	Multiplica el acumulador por el operando
DIV	*operando*	Divide el acumulador por el operando
READ	*operando*	Lee un nuevo dato de entrada y carga operando
WRITE	*operando*	Escribe el operando a la salida
JUMP	*rotulo*	Salto incondicional
JGTZ	*rotulo*	Salto a rotulo si el acumulador es positivo
JZERO	*rotulo*	Salto a rotulo si el acumulador es cero
HALT		Termina ejecucion del programa

RAM

Los ***operandos*** pueden ser*:*

= i indicando el entero de valor i en si mismo

i un entero nonegativo indicando el contenido del registro ri

***i** un puntero, el operando es el contenido de un registro rk, donde rk es el entero que se encuentra en el registro ri. Si rk<0 entonces fin.

RAM

LOAD = a : Carga en el acumulador el valor entero a.

LOAD i : Carga en el acumulador el contenido del registro ri .

LOAD *i : Carga en el acumulador el contenido del registro indexado por el valor del registro ri .

Ejemplos:

LOAD =25

LOAD 1

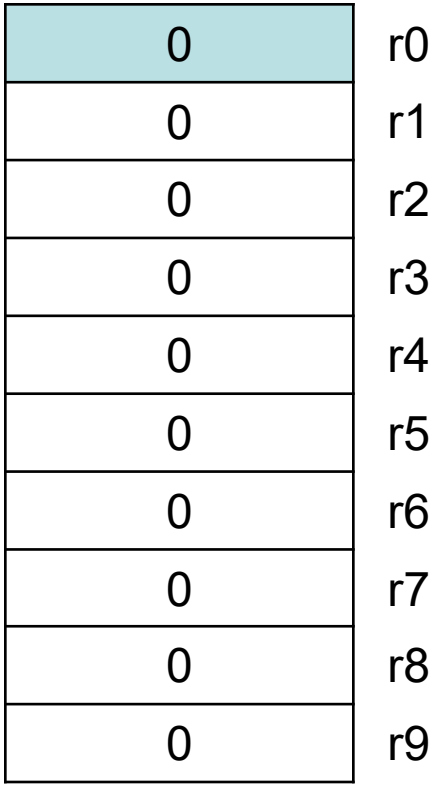
LOAD *1

RAM

Ejemplo 1:

Antes de:

LOAD =25

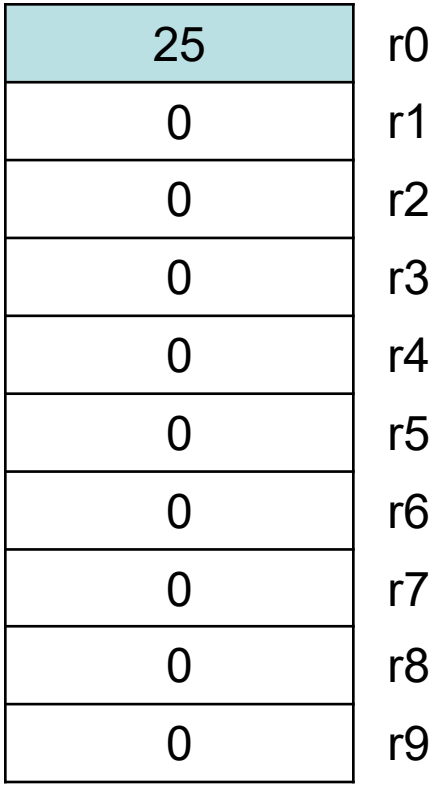


RAM

Ejemplo 1:

Después de:

LOAD =25



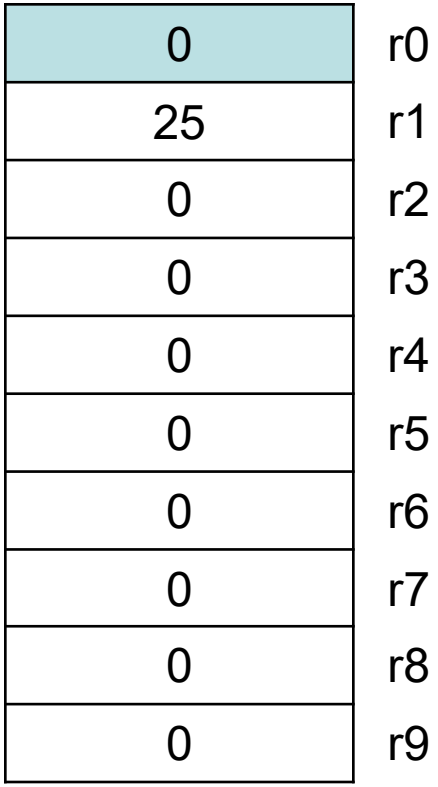
Carga el acumulador con la constante de valor 25.

RAM

Ejemplo 2:

Antes de:

LOAD 1

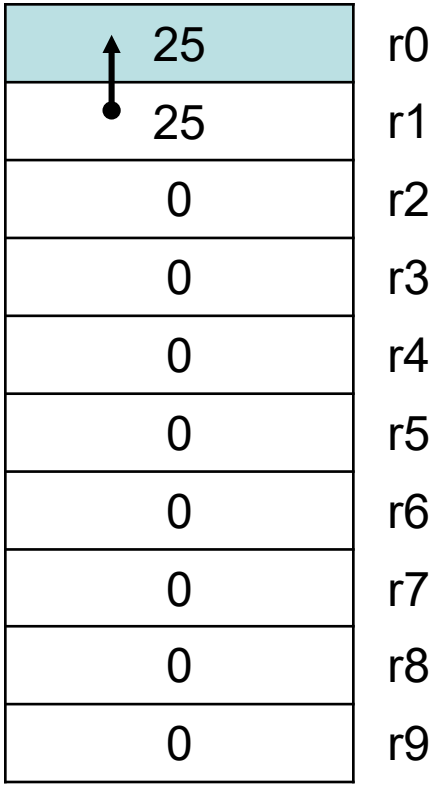


RAM

Ejemplo 2:

Después de:

LOAD 1



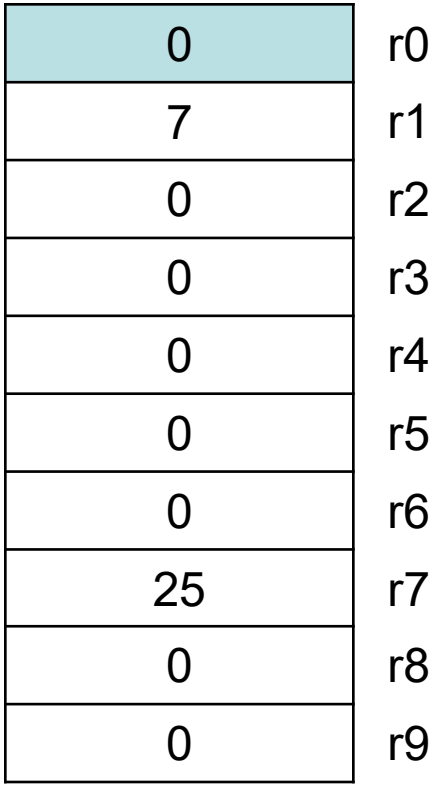
Carga el acumulador con el contenido del r1

RAM

Ejemplo 3:

Antes de:

LOAD *1

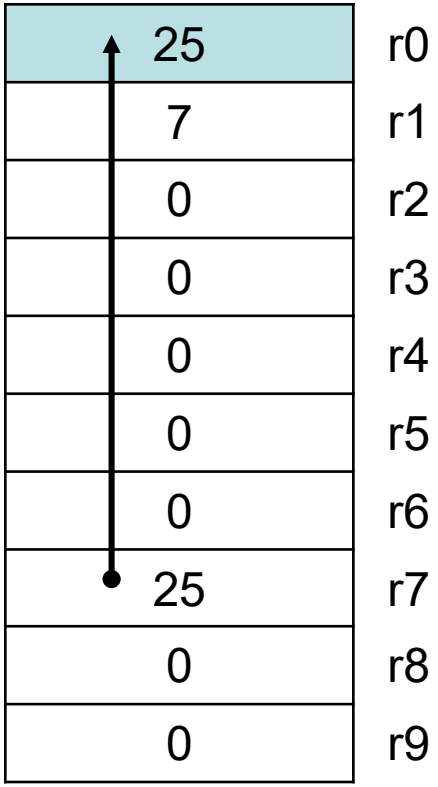


RAM

Ejemplo 3:

Después de:

LOAD *1



Carga el acumulador con el contenido del registro r7 indicado en r1

RAM

Para definir lo que hace un programa se usan 2 funciones:

- Mapa de memoria: c

c(i) es el entero almacenado en el registro i
Inicialmente c(i)=0 para todo i

- Valor del operando: v

v(=i) → i
v(i) → c(i)
v(*i) → c(c(i))

RAM

Ejemplo:

25	r0
8	r1
0	r2
0	r3
0	r4
0	r5
0	r6
25	r7
0	r8
0	r9

Mapa de memoria:

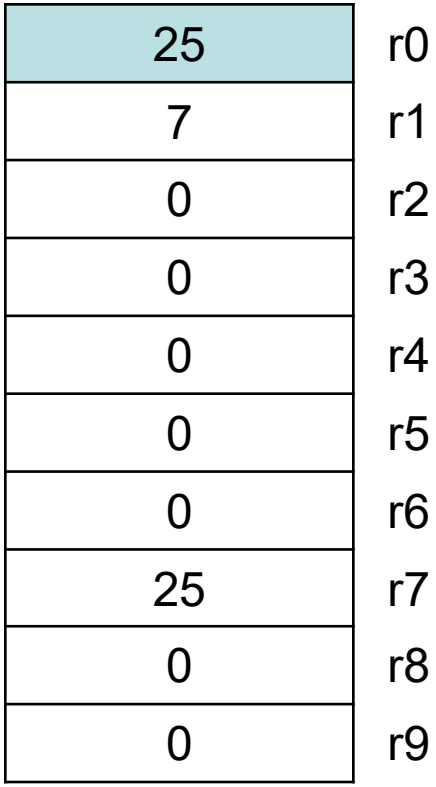
c(0) → 25
c(1) → 8
c(2) → 0
c(3) → 0
c(4) → 0
c(5) → 0
c(6) → 0
c(7) → 25
c(8) → 0
c(9) → 0

RAM

Ejemplo:

Valor del operando:

v(=23) →23
v(1) → 7
v(*1) → 25



RAM

Cada paso del algoritmo consiste de (hasta) tres etapas:

1. LECTURA, durante la cuál el procesador lee un dato desde una posición arbitraria de la memoria en uno de sus registros internos.

2. CALCULO, durante la cuál el procesador realiza una operación básica sobre los contenidos de uno o dos de sus registros.

3. ESCRITURA, durante la cuál el procesador escribe el contenido de un registro en una posición de memoria arbitraria.

Ejemplos en máquina RAM

Leer :

Leer (r1)

READ 1

Escribir:

Escribir (r2)

WRITE 2

Código	Dirección	Explicación
LOAD	operando	*Carga el operando en el acumulador*
STORE	operando	*Carga el acumulador en un registro*
ADD	operando	*Suma el operando al acumulador*
SUB	operando	*Resta el operando al acumulador*
MULT	operando	*Multiplica el acumulador por el operando*
DIV	operando	*Divide el acumulador por el operando*
READ	operando	*Lee dato de entrada y carga operando*
WRITE	operando	*Escribe el operando a la salida*
JUMP	rotulo	*Salto incondicional*
JGTZ	rotulo	*Salto a rotulo si el acumulador es positivo*
JZERO	rotulo	*Salto a rotulo si el acumulador es cero*
HALT		*Termina ejecución del programa*

Ejemplos en máquina RAM

Asignación:

r1←r2

```
LOAD 2
STORE 1
```

Operación aritmética:

r1←r2+3

```
LOAD 2
ADD =3
STORE 1
```

Leer, calcular y escribir:

Leer (r1,r2)

Escribir (r1*r2)

```
READ 1
READ 2
LOAD 1
MULT 2
STORE 1
WRITE 1
```

Código	Dirección	Explicación
LOAD	operando	*Carga el operando en el acumulador*
STORE	operando	*Carga el acumulador en un registro*
ADD	operando	*Suma el operando al acumulador*
SUB	operando	*Resta el operando al acumulador*
MULT	operando	*Multiplica el acumulador por el operando*
DIV	operando	*Divide el acumulador por el operando*
READ	operando	*Lee dato de entrada y carga operando*
WRITE	operando	*Escribe el operando a la salida*
JUMP	rotulo	*Salto incondicional*
JGTZ	rotulo	*Salto a rotulo si el acumulador es positivo*
JZERO	rotulo	*Salto a rotulo si el acumulador es cero*
HALT		*Termina ejecución del programa*

Ejemplos en máquina RAM

Selección:

Si r1 ≤ 0 entonces
    r2 ←r1+r2
Sino
    r2←r1
Fin si

```
carga   LOAD 1
        JGTZ sino
        LOAD 1
        ADD 2
        STORE 2
        JUMP finsi
sino    LOAD 1
        STORE 2
finsi   HALT
```

Código	Dirección	Explicación
LOAD	operando	*Carga el operando en el acumulador*
STORE	operando	*Carga el acumulador en un registro*
ADD	operando	*Suma el operando al acumulador*
SUB	operando	*Resta el operando al acumulador*
MULT	operando	*Multiplica el acumulador por el operando*
DIV	operando	*Divide el acumulador por el operando*
READ	operando	*Lee dato de entrada y carga operando*
WRITE	operando	*Escribe el operando a la salida*
JUMP	rotulo	*Salto incondicional*
JGTZ	rotulo	*Salto a rotulo si el acumulador es positivo*
JZERO	rotulo	*Salto a rotulo si el acumulador es cero*
HALT		*Termina ejecución del programa*

Ejemplos en máquina RAM

Iteración condicional:

Mientras r1>0 hacer
r1←r1-2
Fin mientras

```
carga       LOAD 1
            JGTZ mientras
            JUMP finmientras
mientras    LOAD 1
            SUB =2
            STORE 1
            JUMP carga
finmientras HALT
```

Código	Dirección	Explicación
LOAD	operando	*Carga el operando en el acumulador*
STORE	operando	*Carga el acumulador en un registro*
ADD	operando	*Suma el operando al acumulador*
SUB	operando	*Resta el operando al acumulador*
MULT	operando	*Multiplica el acumulador por el operando*
DIV	operando	*Divide el acumulador por el operando*
READ	operando	*Lee dato de entrada y carga operando*
WRITE	operando	*Escribe el operando a la salida*
JUMP	rotulo	*Salto incondicional*
JGTZ	rotulo	*Salto a rotulo si el acumulador es positivo*
JZERO	rotulo	*Salto a rotulo si el acumulador es cero*
HALT		*Termina ejecución del programa*

Ejemplo Algoritmo en pseudocodigo

Algoritmo potencia
Entrada: r1,número entero
Salida: r2, entero positivo, $r2=r1^{r1}$ si $r1 \geq 1$ sino r2=0
Auxiliar: r3 entero
P0. Leer (r1)
P1. Si r1 ≤ 0 Entonces
Escribir (0)
ir a paso P6
P2. r2←r1
P3. r3←r1-1
P4. Si r3 >0 entonces
r2←r2*r1
r3←r3-1
volver a paso P4
P5. Escribir (r2)
P6. Fin

Ejemplo Algoritmo en máquina RAM

	READ	1	*lee y almacena en el registro r1*
	LOAD	1	*carga el acumulador con el valor almacenado en r1*
	JGTZ	*positivo*	*si r1 > 0 entonces va a positivo*
	WRITE	= 0	*si r1 ≤ 0 entonces escriba 0*
	JUMP	*endif*	*va a endif*
positivo	LOAD	1	*carga el acumulador con el valor almacenado en r1*
	STORE	2	*r2 ← r1*
	LOAD	1	*r1*
	SUB	= 1	*r1 -1*
	STORE	3	*r3 ← r1 -1*
while	LOAD	3	*carga el acumulador con el valor almacenado en r3*
	JGTZ	*continue*	*si r3 > 0 entonces va a continue*
	JUMP	*endwhile*	*si r3 ≤ 0 entonces va a endwhile*
continue	LOAD	2	*carga el acumulador con el valor almacenado en r2*
	MULT	1	*r2 * r1* .
	STORE	2	*r2 ←r2 * r1*
	LOAD	3	*carga el acumulador con el valor almacenado en r3*
	SUB	= 1	*r3-1*
	STORE	3	*r3 ←r3-1*
	JUMP	*while*	*vuelve a while*
endwhile	WRITE	2	*escribe r2*
endif	*HALT*		*fin*

RAM

Este modelo abstracto sirve para modelar situaciones donde:

a) se dispone de memoria suficiente para almacenar el problema

b) los enteros que se usan en los cálculos se pueden almacenar en una palabra.

Modelos

Se puede considerar distintos modelos para determinar la complejidad en función del tamaño de la entrada del problema.

- ***Modelo uniforme***: supone que los valores a almacenar están acotados. Cada instrucción requiere una unidad de tiempo y cada registro requiere una unidad de almacenamiento.

- ***Modelo logarítmico***: más realista, considera la memoria real limitada. Considera la representación binaria de los números a almacenar, que para almacenar el entero n necesita $\lceil \log_2(n+1) \rceil$ bits. El costo de cada instrucción es proporcional a la longitud de los operandos de la instrucción.

Modelos

Los costos obtenidos para un programa serán diferentes según el modelo usado.

Qué modelo usar?

- Si se puede suponer que cada número del problema se puede almacenar en una palabra del computador, el costo uniforme será el más apropiado.

- Si no es así el costo logarítmico será más apropiado para hacer una análisis realista del problema.

Complejidad

Cómo se calcula el tiempo de ejecución de un programa RAM?

Tiempo de ejecución de un Algoritmo:

$T_A(I)$= suma de todos los tiempos de ejecución de las instrucciones realizadas por el algoritmo en la instancia I:

$$T_A(I) = \sum_j t_j$$

t_j : tiempo de ejecución de la instrucción j.

Complejidad de un algoritmo A:

$C_A(n)$=maximo $T_A(I)$, considerando todas las instancias de tamaño n:

$$C_A(n) = \max_{I:\ |I|=n} T_A(I)$$

Este es el modelo que se usa implícitamente cuando se calcula la complejidad de un algoritmo en la práctica

Complejidad

- La ***complejidad del peor caso*** de un programa RAM es la máxima de las sumas T para todas las entradas de tamaño n.

- La ***complejidad del caso esperado*** de un programa RAM es el promedio de las sumas T para todas las entradas de tamaño n.

- En todos los casos la complejidad depende del tamaño de la entrada n.

Tamaño de la entrada de un problema

- Número de símbolos de un alfabeto finito necesarios para codificar todos los datos de un problema.
- El tamaño de la entrada de un problema depende de la base o alfabeto elegidos.
- Para almacenar un entero positivo N en base 2 se necesitan:

$$M = \lceil \log_2(N+1) \rceil \text{ dígitos binarios.}$$

- Para almacenar un entero positivo N en una base b se necesitan:

$$M = \lceil \log_b(N+1) \rceil \text{ dígitos en esa base.}$$

Tiempo de ejecución de un programa

Para dar una medida del tiempo de ejecución de un programa hay que analizar varios factores, entre ellos:

- la ***entrada*** del programa,
- el ***algoritmo*** aplicado.
- la calidad del ***código generado por el compilador*** usado para crear la imagen objeto.
- la naturaleza y velocidad de las ***instrucciones de máquina*** usadas para ejecutar el programa.

Complejidad de Tiempo de un programa

- Se llama **T(n)** al *tiempo de ejecución de un programa con una entrada de tamaño n*.
- **T(n)** es la llamada ***complejidad de tiempo del programa***.
- T(n) depende del algoritmo usado.
- Se dice que el tiempo de ejecución es proporcional a T(n), la constante de proporcionalidad depende del computador.

Tiempo de ejecución de un algoritmo

- Se denota también con T(n) el tiempo de ejecución de un algoritmo para una entrada de tamaño n.

- T(n) es el número de instrucciones ejecutadas por el algoritmo en una computadora ideal, por ejemplo con el modelo RAM.

- Qué pasa con el tiempo de ejecución T(n) estimado cuando ese algoritmo se implementa en un programa y se ejecuta en una computadora real? Y si se cambia de lenguaje y/o de máquina?

- Una respuesta a esta pregunta viene dada por el *principio de invariancia* que afirma que dos implementaciones distintas de un mismo algoritmo no diferirán en su eficiencia en más de alguna constante multiplicativa.

Principio de invarianza

Dos *implementaciones* de un mismo algoritmo no diferirán más que en una constante multiplicativa.

Concretamente si dos implementaciones del mismo algoritmo necesitan t1(n) y t2(n) segundos respectivamente, para resolver un caso de entrada de tamaño n, siempre existen constantes positivas c y d tales que:

$\exists\, c, d \in \mathbf{R}, \qquad t1(n) \leq c.t2(n) \quad y \quad t2(n) \leq d.t1(n)$

siempre que n sea suficientemente grande.

Principio de invarianza

- *En otras palabras, el tiempo de ejecución de cualquiera de las implementaciones está acotado por un múltiplo constante del tiempo de ejecución de la otra.*

- Este principio no es algo que se pueda demostrar, simplemente establece un hecho que puede ser confirmado por observación.

- Además tiene un amplio rango de aplicación.

- El principio sigue siendo cierto sea cual fuere la computadora utilizada para implementar el algoritmo, independiente del lenguaje de programación y del compilador empleado, independiente incluso de la habilidad del programador.

Principio de invarianza

- Bajo este principio un cambio de máquina puede permitir resolver el problema 10 veces o 100 veces más rápido.

- Un cambio de algoritmo sin embargo puede darnos un incremento que se vuelva cada vez más pronunciado a medida que crezca el tamaño de la entrada.

- Respecto a la unidad de tiempo, el principio de invariancia nos permite decidir que *no va a existir unidad.*

- En su lugar, expresamos solamente el tiempo requerido por el algoritmo salvo una constante multiplicativa usando una *notación asintótica.*

Algoritmos y Estructuras de Datos II

Trabajo Práctico 1