Clase 04 - Bits, bytes y codificaciones

IIC1001 - Algoritmos y Sistemas Computacionales

Cristian Ruz – cruz@uc.cl

Lunes 18-Marzo-2024

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

Contenidos

Contenidos

Contacto

🥇

ignaciomunoz@uc.cl

Ignacio Muñoz

Ayudante jefe

- Coordinación
- Notas de actividades, interrogaciones
- Todo lo que no sé donde más enviar

🐐

vicente.cabra@uc.cl

Vicente Cabra

Ayudante

- Materia

🐚

fernando.concha@uc.cl

Fernando Concha

Ayudante

- Materia

🐸

alejandro.tapia@uc.cl

Alejandro Tapia

Ayudante

- Materia

Contenidos

Temas del curso

Sistemas computacionales

- Representación datos, números y compresión
- Funcionamiento hardware, procesadores y memoria.
- Funcionamiento de sistemas operativos: ejemplo scheduling
- Funcionamiento de Internet
- Herramientas computacionales: github + latex

Algoritmos

- Algoritmos y resolución de problemas
- Eficiencia algorítmica
- Estructuras secuenciales y ordenamiento
- Grafos y árboles

Contenidos

Sistemas de representación numérica

Los números (naturales) son infinitos.

- Usamos D símbolos (dígitos) para representar lo que podemos.
- Cuando se nos acaban los dígitos, agregamos una **posición** más.
- La nueva posición indica cuántas veces hemos pasado todos los dígitos de la posición anterior.
- En base 10, usamos 10 dígitos ($D = 10$)

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, \ldots$$

- Podría escribirse como:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1 \times D, 1 \times D + 1, 1 \times D + 2, 1 \times D + 3, \ldots, 2 \times D, 2 \times D + 1, \ldots$$

- Un número en base 10 como 13425, es una abreviación de la expresión:

$$1 \times 10000 + 3 \times 1000 + 4 \times 100 + 2 \times 10 + 5 \times 1$$

- Pero también se puede escribir usando la base $D = 10$:

$$1 \times 10^4 + 3 \times 10^3 + 4 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

- Estos sistemas se conocen como sistemas de representación posicional

Lo mismo pero con $D = 2$

- Con $D = 2$ solo tenemos dos dígitos: 0 y 1.
- Cuando se nos acaban los dígitos, agregamos una **posición** más.
- Usamos la misma idea que cuando teníamos $D = 10$.

$$0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, \ldots$$

- Podría escribirse como:

$$0, 1, 1\times D, 1\times D+1, 1\times D\times D, 1\times D\times D+1, 1\times D\times D+1\times D, 1\times D\times D+1\times D+1\times 1, \ldots$$

$$0, 1, 1 \times D, 1 \times D + 1, 1 \times D^2, 1 \times D^2 + 1, 1 \times D^2 + 1 \times D, 1 \times D^2 + 1 \times D + 1, \ldots$$

$$0, 1, 1 \times 2, 1 \times 2 + 1, 1 \times 2^2, 1 \times 2^2 + 1, 1 \times 2^2 + 1 \times 2, 1 \times 2^2 + 1 \times 2 + 1, \ldots$$

$$0, 1, 1 \times 2, 1 \times 2 + 1, 1 \times 4, 1 \times 4 + 1, 1 \times 4 + 1 \times 2, 1 \times 4 + 1 \times 2 + 1, \ldots$$

- Un número en base 2 como 101010, es una abreviación de la expresión:

$$1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$$

- Pero también se puede escribir usando la base $D = 2$:

$$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

Sistemas de representación numérica

Ahora podemos hacer algunas equivalencias:

Base 10	Descomposición	Base 2	Descomposición
0	0×10^0	0	0×2^0
1	1×10^0	1	1×2^0
2	2×10^0	10	$1 \times 2^1 + 0 \times 2^0$
3	3×10^0	11	$1 \times 2^1 + 1 \times 2^0$
4	4×10^0	100	$1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
5	5×10^0	101	$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
6	6×10^0	110	$1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
7	7×10^0	111	$1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
8	8×10^0	1000	$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
9	9×10^0	1001	$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
10	$1 \times 10^1 + 0 \times 10^0$	1010	$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
11	$1 \times 10^1 + 1 \times 10^0$	1011	$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
12	$1 \times 10^1 + 2 \times 10^0$	1100	$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
13	$1 \times 10^1 + 3 \times 10^0$	1101	$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
14	$1 \times 10^1 + 4 \times 10^0$	1110	$1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
15	$1 \times 10^1 + 5 \times 10^0$	1111	$1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
16	$1 \times 10^1 + 6 \times 10^0$	10000	$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
17	$1 \times 10^1 + 7 \times 10^0$	10001	$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
18	$1 \times 10^1 + 8 \times 10^0$	10010	$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
19	$1 \times 10^1 + 9 \times 10^0$	10011	$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$
20	$2 \times 10^1 + 0 \times 10^0$	10100	$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
100	$1 \times 10^2 + 0 \times 10^1 + 0 \times 10^0$	1100100	$1 \times 2^6 + 1 \times 2^5 + 1 \times 2^2$
1000	1×10^3	1111101000	$1 \times 2^9 + 1 \times 2^8 + 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^3$

Sistemas de representación numérica

También podemos abreviar un poco:

Base 10	Descomposición	Base 2	Descomposición
0	0×1	0	0×1
1	1×1	1	1×1
2	2×1	10	$1 \times 2 + 0 \times 1$
3	3×1	11	$1 \times 2 + 1 \times 1$
4	4×1	100	$1 \times 4 + 0 \times 2 + 0 \times 1$
5	5×1	101	$1 \times 4 + 0 \times 2 + 1 \times 1$
6	6×1	110	$1 \times 4 + 1 \times 2 + 0 \times 1$
7	7×1	111	$1 \times 4 + 1 \times 2 + 1 \times 1$
8	8×1	1000	$1 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1$
9	9×1	1001	$1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1$
10	$1 \times 10 + 0 \times 1$	1010	$1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$
11	$1 \times 10 + 1 \times 1$	1011	$1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$
12	$1 \times 10 + 2 \times 1$	1100	$1 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1$
13	$1 \times 10 + 3 \times 1$	1101	$1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1$
14	$1 \times 10 + 4 \times 1$	1110	$1 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1$
15	$1 \times 10 + 5 \times 1$	1111	$1 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1$
16	$1 \times 10 + 6 \times 1$	10000	$1 \times 16 + 0 \times 8 + 0 \times 4 + 0 \times 2 + 0 \times 1$
17	$1 \times 10 + 7 \times 1$	10001	$1 \times 16 + 0 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1$
18	$1 \times 10 + 8 \times 1$	10010	$1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$
19	$1 \times 10 + 9 \times 1$	10011	$1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$
20	$2 \times 10 + 0 \times 1$	10100	$1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1$
100	$1 \times 100 + 0 \times 10 + 0 \times 1$	1100100	$1 \times 64 + 1 \times 32 + 1 \times 4$
1000	1×1000	1111101000	$1 \times 512 + 1 \times 256 + 1 \times 128 + 1 \times 64 + 1 \times 32 + 1 \times 8$

Conversión de binario a decimal

Tip: usaremos la notación $(x)_2$ o $(x)_{10}$ para indicar si la representación de x es binaria o decimal. Si no se usa nada, será por defecto decimal.

Suponiendo que la posición de más a la derecha es la posición 0, y que van aumentando a medida que nos movemos la izquierda, y que la última posición es la $n-1$, podemos convertir cualquier número binario de n bits a su equivalente decimal usando:

$$\sum_{k=0}^{n-1} s_k \times 2^k$$

donde s_k es el símbolo (bit) que se encuentra en la posición k.

Ejemplo: convertir $(100110101)_2$ a decimal:

$$\begin{aligned} & 1 \times 2^0 + 1 \times 2^2 + 1 \times 2^4 + 1 \times 2^5 + 1 \times 2^8 \\ &= 1 + 4 + 16 + 32 + 256 \\ &= 309 \end{aligned}$$

Conversión de decimal a binario

Algoritmo de división sucesiva.

1. Dividir el número n por 2. El cuociente es el nuevo n, y el resto (0 ó 1) se ubica a la izquierda del resultado.
2. Repetir el proceso hasta que el cuociente sea 0.

$n = 42$

n	Cuociente	Resto	Resultado
42	21	0	0
21	10	1	10
10	5	0	010
5	2	1	1010
2	1	0	01010
1	0	1	101010

Base 16: Hexadecimal

Si bien los computadores utilizan 0 y 1, muchas representaciones son más fáciles de manipular si se trabajan en grupos más grandes. La representación en base 16 se conoce como **hexadecimal** y requiere de 16 dígitos. Como no conocemos más que 10, agregamos las primeras 6 letras (*A*, *B*, *C*, *D*, *E*, *F*).

Base 10	Base 16	Base 2
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	10000
17	11	10001
18	12	10010
19	13	10011
20	14	10100
42	2A	101010
100	64	1100100
1000	3E8	1111101000

Base 16: Hexadecimal

Convertir desde base 16 a base 2 es fácil. Reemplazamos cada dígito hexadecimal por su correspondiente valor en binario usando 4 bit

Base 16	Base 2	Base 10
A	1010	10
10	0001 0000	16
22	0010 0010	34
7A	0111 1010	122
4B3	0100 1011 0011	1203
83C	1000 0011 1100	2108

Cada dígito hexadecimal se convierte **exactamente** a 4-bit. Los 0's que quedan a la izquierda del resultado final podrían no escribirse. En la tabla se presentan por claridad.

Por ejemplo, el $(22)_{16} \rightarrow (00100010)_2$ podría escribirse como $(100010)_2$, pero **no** como $(1010)_2$.

Base 16: Hexadecimal

Similarmente, convertir desde base 2 a base 16 consiste en agrupar, de derecha a izquierda, grupos 4-bit y reemplazarlos por su correspondiente dígito hexadecimal. Si se necesitan más bit a la izquierda para completar grupos de 4, se usan 0's.

Base 16	Base 2	Base 10
1010	A	10
1 0000	10	16
10 0010	22	34
111 1010	7A	122
100 1011 0011	4B3	1203
1000 0011 1100	83C	2108

Bytes

Los computadores no almacenan bits individuales. Es más fácil manipular grupos de bit de un tamaño fijo.

Los primeros sistemas computacionales agruparon de a 8 bits. Esta unidad (8 bit) se conoce como **byte**, y suele ser la unidad más pequeña con que los sistemas trabajan.

Un **byte** está compuesto de 8 bit. Cada bit puede tomar 2 valores posibles, por lo tanto la cantidad de maneras en que se pueden combinar 8 bit son:
$2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8 = 256$.

Bytes para representar enteros (positivos)

Si utilizamos 1 byte para representar un entero positivo (y el 0), la primera combinación es $(00000000)_2 = (0)_{10}$. La última combinación, con todos los bit en 1, es el $(11111111)_2 = (255)_{10}$. Por lo tanto, 1 byte puede representar 256 enteros:

Base 10	Base 2
0	0000 0000
1	0000 0001
2	0000 0010
...	...
15	0000 1111
16	0001 0000
17	0001 0001
...	...
62	0011 1110
63	0011 1111
64	0100 0000
65	0100 0001
...	...
126	0111 1110
127	0111 1111
128	1000 0000
129	1000 0001
...	...
253	1111 1101
254	1111 1110
255	1111 1111

Bytes para representar enteros (positivos)

1 byte son 8 bit. Escribir 8 bit ocupa espacio. Podemos utilizar 2 dígitos hexadecimales (cada uno con 4 bit) para representar un byte. Por lo tanto, cada byte puede ser escrito también usando 2 dígitos hexadecimales:

Valor base 10	Byte en repr binaria	Byte en repr hexadecimal
0	0000 0000	00
1	0000 0001	01
2	0000 0010	02
...	...	...
15	0000 1111	0F
16	0001 0000	10
17	0001 0001	11
...	...	...
62	0011 1110	3E
63	0011 1111	3F
64	0100 0000	40
65	0100 0001	41
...	...	...
126	0111 1110	7E
127	0111 1111	7F
128	1000 0000	80
129	1000 0001	81
...	...	...
253	1111 1101	FD
254	1111 1110	FE
255	1111 1111	FF

Bytes para representar texto

Un computador no almacena letras (caracteres). Solo números. ¿Cómo representar letras?

Para representar letras, se **asocian** valores numéricos con ciertos patrones (caracteres) en pantalla que identifican una letra. Estas asociaciones se guardan en **mapas de caracteres**, o **codificación de caracteres** (*character encoding*).

Una de estas representaciones estandarizadas es la representación ASCII (*American Standard Code for Information Interchange*) que considera que se puede utilizar 1 byte para representar todas las 26 letras mayúsculas y minúsculas del alfabeto inglés. Solo se necesitan 52 combinaciones.

Valor base 10	Byte binario	Byte hexadecimal	Caracter ASCII
65	0100 0001	41	A
66	0100 0010	42	B
67	0100 0011	43	C
...	...	...	...
88	0101 1000	58	X
89	0101 1001	59	Y
90	0101 1010	5A	Z
...	...	...	...
97	0110 0001	61	a
98	0110 0010	62	b
99	0110 0011	63	c
...	...	...	...
120	0111 1000	78	x
121	0111 1001	79	y
122	0111 1010	7A	z

Bytes para representar texto

Las letras mayúsuculas y minúsculas no son suficientes. También usamos el “espacio en blanco”.

Valor base 10	Byte binario	Byte hexadecimal	Caracter ASCII
32	0010 0000	20	⟨space⟩

Pero también utilizamos caracteres para mostrar dígitos en pantalla.

Valor base 10	Byte binario	Byte hexadecimal	Caracter ASCII
48	0011 0000	30	0
49	0011 0001	31	1
50	0011 0010	32	2
51	0011 0011	33	3
52	0011 0100	34	4
53	0011 0101	35	5
54	0011 0110	36	6
55	0011 0111	37	7
56	0011 1000	38	8
57	0011 1001	39	9

Y también algunos símbolos: !, @, %, +, -, _, ∗, &, (,), :, ;, ,, etc

Bytes para representar texto

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

- ASCII tiene 128 combinaciones. Solo utiliza 7 bit.
- De los 128, 95 son “imprimibles” y los otros son carácteres invisibles, pero con algún significado
- Algunos de los invisibles existen por razones históricas y vienen de sistemas antiguos (teletipos): “cambio de líneas”, “retorno de carro”, “backspace”, “delete”, “bell”, etc

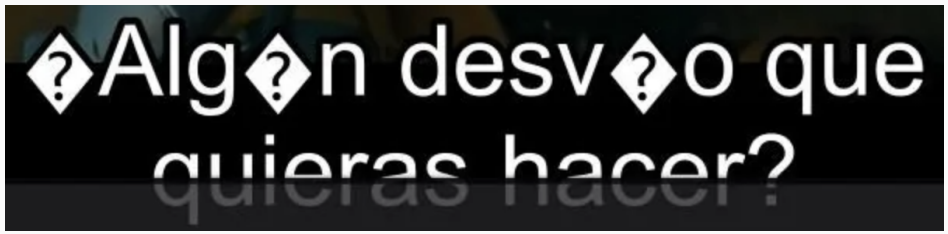
Bytes para representar texto

¿Qué pasa con los tildes, ñ, Ñ, etc?

¡Si aún queda 1 bit no usado, entonces hay 128 valores (desde el 128 al 255) aún disponibles en 1 Byte!

Muchos tuvieron la misma idea y surgieron muchas extensiones de ASCII con tildes, caracteres especiales, caracteres cirílicos, hebreos, etc... Muchos *encoding* que coincidían en los primeros 128 caracteres, pero diferentes en el resto.

Abrir un archivo de texto codificado un *encoding* (ej: UTF-8), con un editor que espera otro *encoding* (ej: ISO-8859-1) generar que los caracteres especiales se vean distintos, o no puedan ser interpretados.



Encodings modernos (como UTF8) extienden el mapa de caracteres, pero conservando los primeros 128 caracteres ASCII.