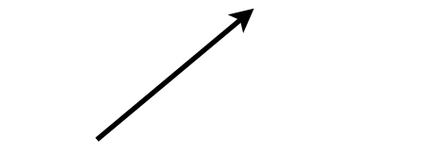
¿como les va con las practicas?

Problemas NP-completos y Estructuras de datos básicas (representación de datos)

mat-151

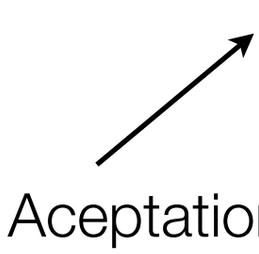
Una maquina de Turing



Aceptation state

Una maquina de Turing

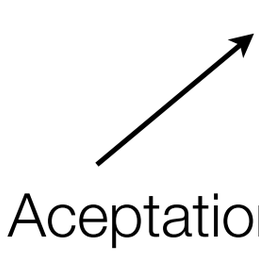
- Maquina de Turing Determinística, que acepta múltiplos de 3 en binario.



Aceptation state

Una maquina de Turing

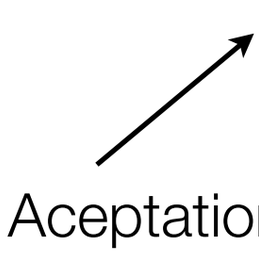
- Maquina de Turing Determinística, que acepta múltiplos de 3 en binario.



Aceptation state

Una maquina de Turing

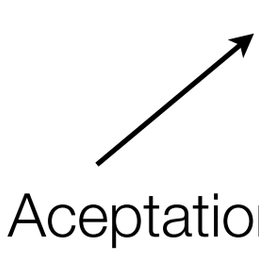
- Maquina de Turing Determinística, que acepta múltiplos de 3 en binario.



Aceptation state

Una maquina de Turing

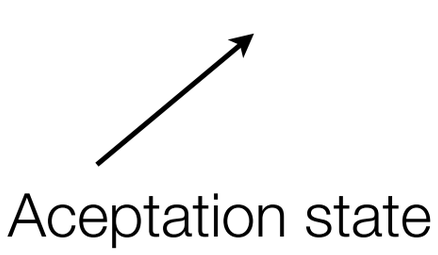
- Maquina de Turing Determinística, que acepta múltiplos de 3 en binario.



Aceptation state

Una maquina de Turing

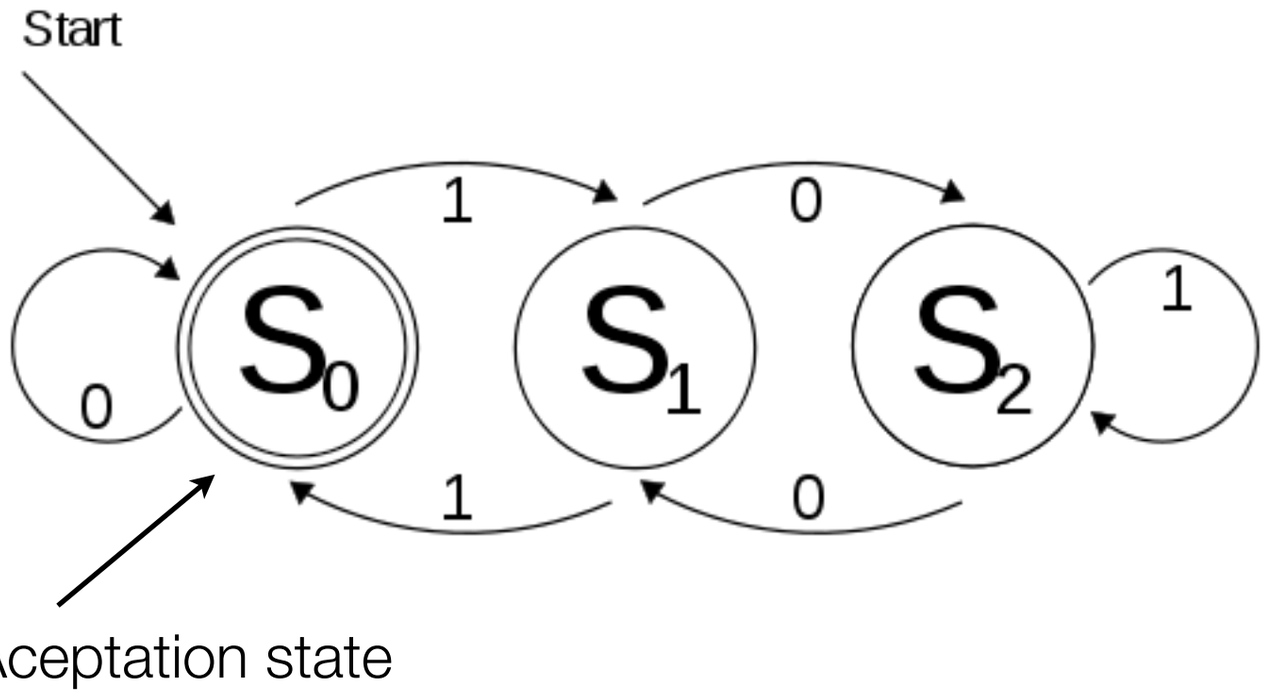
- Maquina de Turing Determinística, que acepta múltiplos de 3 en binario.



- Una no-determinística

Una maquina de Turing

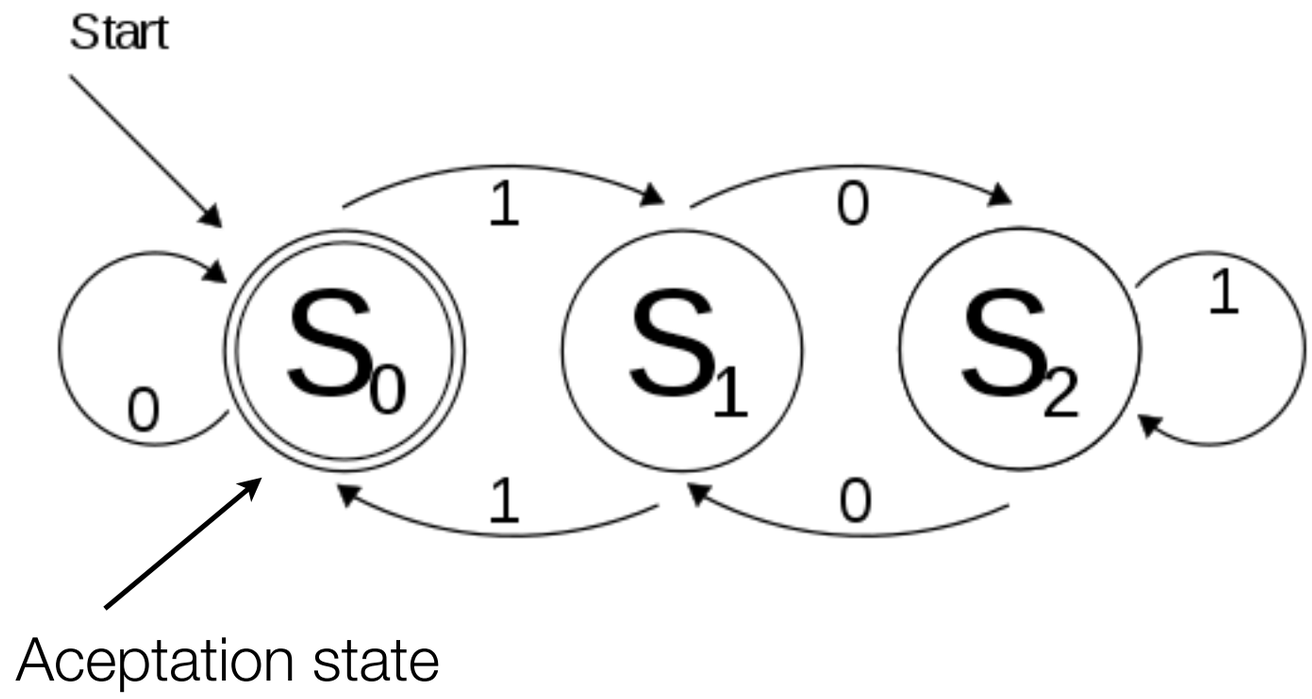
- Maquina de Turing Determinística, que acepta múltiplos de 3 en binario.



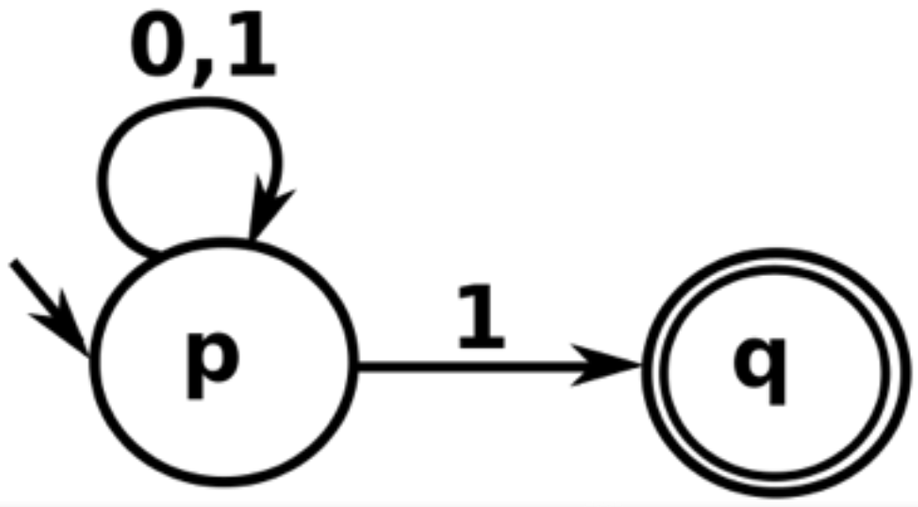
- Una no-determinística

Una maquina de Turing

- Maquina de Turing Determinística, que acepta múltiplos de 3 en binario.

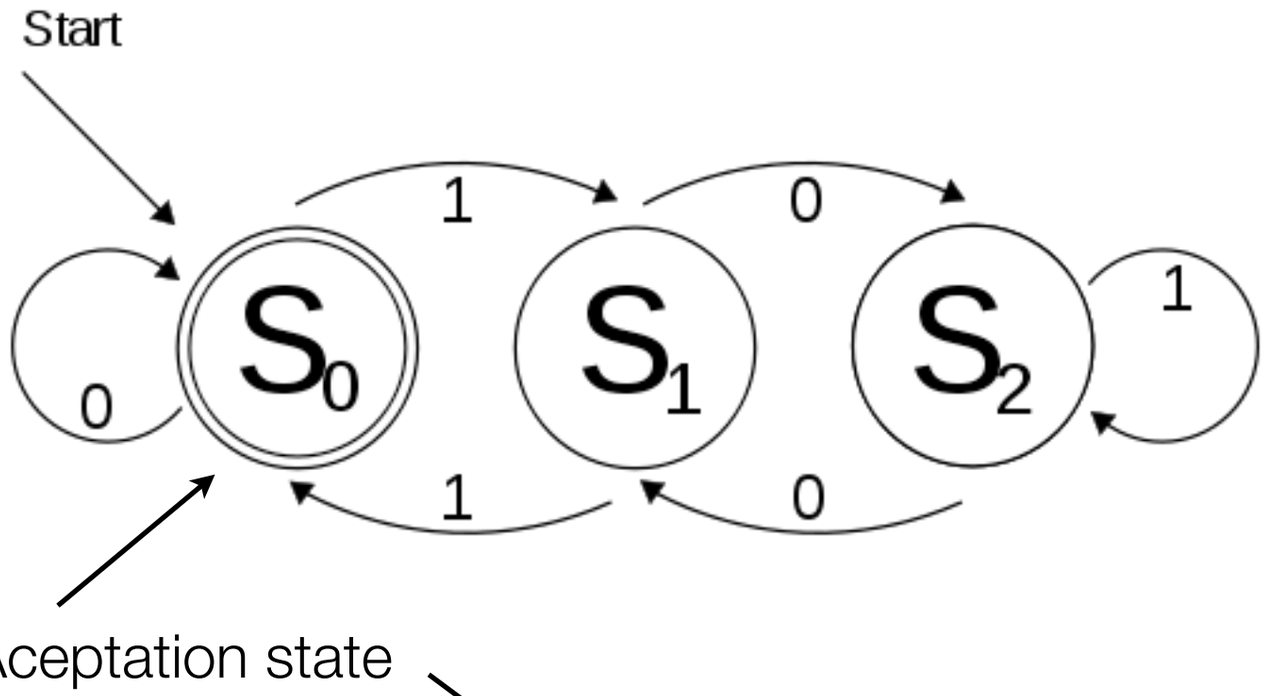


- Una no-determinística

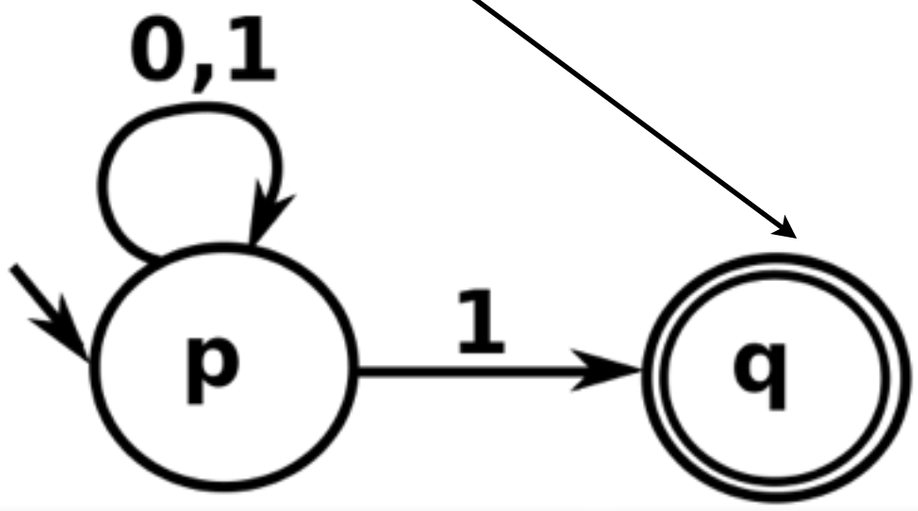


Una maquina de Turing

- Maquina de Turing Determinística, que acepta múltiplos de 3 en binario.



- Una no-determinística



Problemas P y NP

Problemas P y NP

- Problemas **P** se pueden solucionar en tiempo **polinomial** por una máquina de Turing determínistica.

Problemas P y NP

- Problemas **P** se pueden solucionar en tiempo **polinomial** por una máquina de Turing determínistica.

- Problemas **NP (**nondeterministic polynomial time**)** se pueden resolver en tiempo polinomial **PERO por una maquina de Turing no determinística**, si alguien nos da una **solución, certificado o testigo polinómico**, se puede comprobar si es solución del problema en tiempo **polinomial**. Ejemplo: ¿Existe un subconjunto de { –7, –3, –2, 5, 8} tal que su suma es 0?, ¿cuantos subconjuntos debemos de probar?

Problemas P y NP

- Problemas **P** se pueden solucionar en tiempo **polinomial** por una máquina de Turing determínistica.

- Problemas **NP (**nondeterministic polynomial time**)** se pueden resolver en tiempo polinomial **PERO por una maquina de Turing no determinística**, si alguien nos da una **solución, certificado o testigo polinómico**, se puede comprobar si es solución del problema en tiempo **polinomial**. Ejemplo: ¿Existe un subconjunto de { –7, –3, –2, 5, 8} tal que su suma es 0?, ¿cuantos subconjuntos debemos de probar?

- Problemas **NP-completos** los más “dificiles” de los **NP,** no es posible encontrar un algoritmo mejor que simplemente realizar una búsqueda exhaustiva. Si **cualquier** problema **NP-completo** se puede resolver en tiempo polinomial, entonces **todos** los problemas NP-completos se pueden resolver en tiempo polinomial.

Problemas P y NP

- Problemas **P** se pueden solucionar en tiempo **polinomial** por una máquina de Turing determínistica.

- Problemas **NP (**nondeterministic polynomial time**)** se pueden resolver en tiempo polinomial **PERO por una maquina de Turing no determinística**, si alguien nos da una **solución, certificado o testigo polinómico**, se puede comprobar si es solución del problema en tiempo **polinomial**. Ejemplo: ¿Existe un subconjunto de { –7, –3, –2, 5, 8} tal que su suma es 0?, ¿cuantos subconjuntos debemos de probar?

- Problemas **NP-completos** los más “dificiles” de los **NP,** no es posible encontrar un algoritmo mejor que simplemente realizar una búsqueda exhaustiva. Si **cualquier** problema **NP-completo** se puede resolver en tiempo polinomial, entonces **todos** los problemas NP-completos se pueden resolver en tiempo polinomial.

- No se sabe si los problemas **NP-completos** son tratables, si es NP-completo o P, se puede demostrar por **reducción**.

Reducción

Reducción

- Tenemos un problema de decisión A, que no sabemos si se puede resolver en tiempo **polinomial**.

Reducción

- Tenemos un problema de decisión A, que no sabemos si se puede resolver en tiempo **polinomial**.
- Llamamos a la entrada a un problema particular una **instancia del problema.**

Reducción

- Tenemos un problema de decisión A, que no sabemos si se puede resolver en tiempo **polinomial**.
- Llamamos a la entrada a un problema particular una **instancia del problema.**
- Tenemos un problema de decisión B, diferente a A, que sabemos cómo resolver en tiempo polinomial.

Reducción

- Tenemos un problema de decisión A, que no sabemos si se puede resolver en tiempo **polinomial**.
- Llamamos a la entrada a un problema particular una **instancia del problema.**
- Tenemos un problema de decisión B, diferente a A, que sabemos cómo resolver en tiempo polinomial.
- Conocemos un **procedimiento** que transforma cualquier instancia α de A en una instancia β de B con las características siguientes:

Reducción

- Tenemos un problema de decisión A, que no sabemos si se puede resolver en tiempo **polinomial**.
- Llamamos a la entrada a un problema particular una **instancia del problema.**
- Tenemos un problema de decisión B, diferente a A, que sabemos cómo resolver en tiempo polinomial.
- Conocemos un **procedimiento** que transforma cualquier instancia α de A en una instancia β de B con las características siguientes:
  - La transformación toma tiempo polinomial.

Reducción

- Tenemos un problema de decisión A, que no sabemos si se puede resolver en tiempo **polinomial**.
- Llamamos a la entrada a un problema particular una **instancia del problema.**
- Tenemos un problema de decisión B, diferente a A, que sabemos cómo resolver en tiempo polinomial.
- Conocemos un **procedimiento** que transforma cualquier instancia α de A en una instancia β de B con las características siguientes:
  - La transformación toma tiempo polinomial.
  - Las respuestas son las mismas. Esto es, la respuesta de α es “si” si y solo si la respuesta para β es ¨si¨.

Reducción

- Tenemos un problema de decisión A, que no sabemos si se puede resolver en tiempo **polinomial**.
- Llamamos a la entrada a un problema particular una **instancia del problema.**
- Tenemos un problema de decisión B, diferente a A, que sabemos cómo resolver en tiempo polinomial.
- Conocemos un **procedimiento** que transforma cualquier instancia α de A en una instancia β de B con las características siguientes:
  - La transformación toma tiempo polinomial.
  - Las respuestas son las mismas. Esto es, la respuesta de α es “si” si y solo si la respuesta para β es ¨si¨.
- Nos da una forma para solucionar A en tiempo polinomial.

Reducción

Reducción

- Suponemos tener un problema **A** para el que **sabemos que no** hay solución posible en tiempo polinomial.

Reducción

- Suponemos tener un problema **A** para el que **sabemos que no** hay solución posible en tiempo polinomial.

- Suponemos que tenemos una reducción polinomial para transformar todas las instancias de A en las de otro problema B.

Reducción

- Suponemos tener un problema **A** para el que **sabemos que no** hay solución posible en tiempo polinomial.

- Suponemos que tenemos una reducción polinomial para transformar todas las instancias de A en las de otro problema B.

- Podemos probar por contradicción que no existe un algoritmo polinomial para B.

Reducción

- Suponemos tener un problema **A** para el que **sabemos que no** hay solución posible en tiempo polinomial.

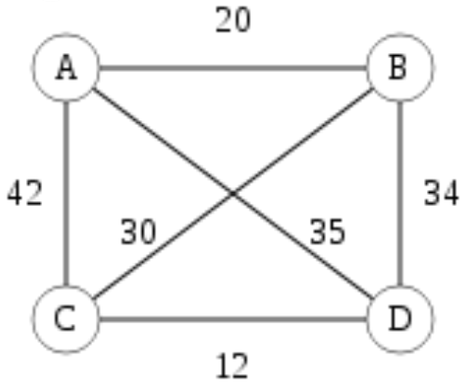
- Suponemos que tenemos una reducción polinomial para transformar todas las instancias de A en las de otro problema B.

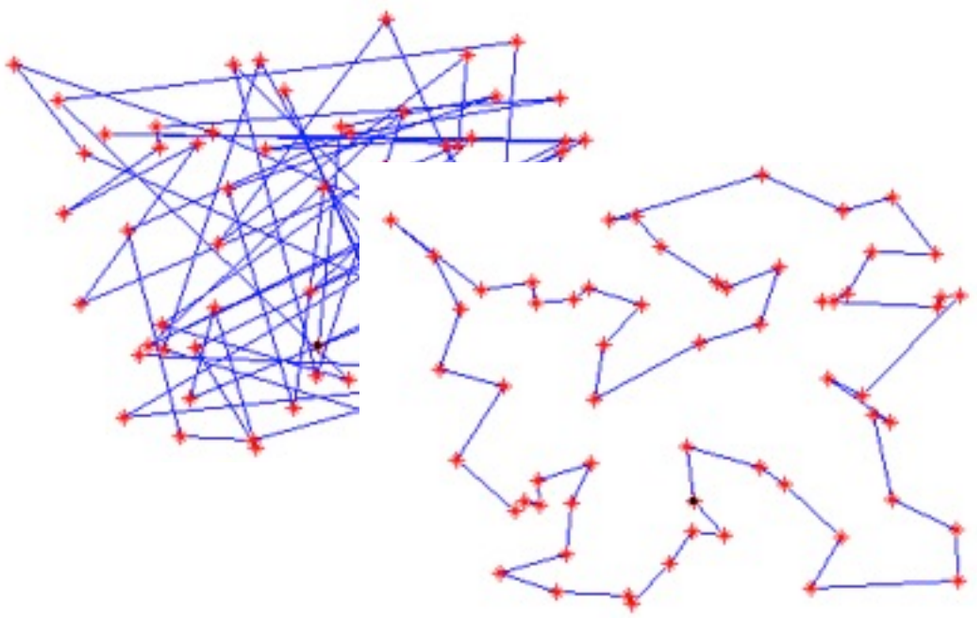
- Podemos probar por contradicción que no existe un algoritmo polinomial para B.

- En caso de existir, podríamos probar que hay un algoritmo polinomial para A.

Problema del vendedor viajero (TSP)

- Dado un número de ciudades y los costos de ir de una ciudad a otra, ¿cuál es el viaje redondo de menor costo que visita cada ciudad **una sola vez** y **regresa** a la ciudad de origen?





- La solución más directa seria intentar todas las **permutaciones** (combinaciones ordenadas) y ver cuál es la más barata (utilizando búsqueda de fuerza bruta).

- Dado que el número de permutaciones es ***n!***, obviamente la solución es NO práctica.

- appletviewer AppAgenteViajero.htm

TSP

- El TSP es uno de los problemas más intensamente estudiados en matemáticas computacionales y hasta ahora no se ha encontrado un método de solución para el caso general.

- Una solución resolvería el asunto de los problemas P vs. NP y les daría un premio de $1,000,000 del Clay Mathematics Institute.

TSP info en

http://www.tsp.gatech.edu/index.html

Tipos de datos

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

- es un ***conjunto*** de valores y **operaciones** en los valores.

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

- es un ***conjunto*** de valores y **operaciones** en los valores.
    - ***conjunto de valores***, e.g. el tipo **entero** puede corresponder al conjunto {...,-1,0,1,...}

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

- es un ***conjunto*** de valores y **operaciones** en los valores.
  - ***conjunto de valores***, e.g. el tipo **entero** puede corresponder al conjunto {...,-1,0,1,...}
  - que presentan un ***comportamiento uniforme ante*** un conjunto de operaciones como suma, multiplicación, comparación,... e.g. el **tipo** {13,naranja,'z'} sería un tipo inusual... (¿el naranja es más chico que 'z'?)

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

- es un ***conjunto*** de valores y **operaciones** en los valores.
  - ***conjunto de valores***, e.g. el tipo **entero** puede corresponder al conjunto {...,-1,0,1,...}
  - que presentan un ***comportamiento uniforme ante*** un conjunto de operaciones como suma, multiplicación, comparación,... e.g. el **tipo** {13,naranja,'z'} sería un tipo inusual... (¿el naranja es más chico que 'z'?)
    - ➡ boolean = {false, true}

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

- es un ***conjunto*** de valores y **operaciones** en los valores.
  - ***conjunto de valores***, e.g. el tipo **entero** puede corresponder al conjunto {...,-1,0,1,...}
  - que presentan un ***comportamiento uniforme ante*** un conjunto de operaciones como suma, multiplicación, comparación,... e.g. el **tipo** {13,naranja,'z'} sería un tipo inusual... (¿el naranja es más chico que 'z'?)
    - ➡ boolean = {false, true}
    - ➡ integer = {..., -1, 0, 1, ... }

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

- es un ***conjunto*** de valores y **operaciones** en los valores.
  - ***conjunto de valores***, e.g. el tipo **entero** puede corresponder al conjunto {...,-1,0,1,...}
  - que presentan un ***comportamiento uniforme ante*** un conjunto de operaciones como suma, multiplicación, comparación,... e.g. el **tipo** {13,naranja,'z'} sería un tipo inusual... (¿el naranja es más chico que 'z'?)
    - ➡ boolean = {false, true}
    - ➡ integer = {..., -1, 0, 1, ... }
    - ➡ real = {...,-1.0, ... , 0.0, ..., 1.0, ...}

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

- es un ***conjunto*** de valores y **operaciones** en los valores.
  - ***conjunto de valores***, e.g. el tipo **entero** puede corresponder al conjunto {...,-1,0,1,...}
  - que presentan un ***comportamiento uniforme ante*** un conjunto de operaciones como suma, multiplicación, comparación,... e.g. el **tipo** {13,naranja,'z'} sería un tipo inusual... (¿el naranja es más chico que 'z'?)
    - ➡ boolean = {false, true}
    - ➡ integer = {..., -1, 0, 1, ... }
    - ➡ real = {...,-1.0, ... , 0.0, ..., 1.0, ...}
    - ➡ character = {'\0\, ...,'a',...,'z',...,'\255'}

Tipos de datos

- uno de los conceptos fundamentales de cualquier lenguaje de programación.

- es un ***conjunto*** de valores y **operaciones** en los valores.
  - ***conjunto de valores***, e.g. el tipo **entero** puede corresponder al conjunto {...,-1,0,1,...}
  - que presentan un ***comportamiento uniforme ante*** un conjunto de operaciones como suma, multiplicación, comparación,... e.g. el **tipo** {13,naranja,'z'} sería un tipo inusual... (¿el naranja es más chico que 'z'?)
    - ➡ boolean = {false, true}
    - ➡ integer = {..., -1, 0, 1, ... }
    - ➡ real = {...,-1.0, ... , 0.0, ..., 1.0, ...}
    - ➡ character = {'\0\, ...,'a',...,'z',...,'\255'}
    - ➡ colors, months, etc.....

Tipos de datos

Tipos de datos

- La memoria de una computadora está organizada en unidades llamadas ***bytes***.

Tipos de datos

- La memoria de una computadora está organizada en unidades llamadas ***bytes***.

- En la mayor parte de las arquitecturas, un ***byte (octeto)*** está formado por ***ocho bits.*** Cada ***bit*** toma valores de **1** o **0***.*

Tipos de datos

- La memoria de una computadora está organizada en unidades llamadas ***bytes***.
- En la mayor parte de las arquitecturas, un ***byte (octeto)*** está formado por ***ocho bits.*** Cada ***bit*** toma valores de **1** o **0***.*
- Cada octeto tiene una **dirección única** en la memoria, identificada por un número de 16, 32 o 64 bits, dependiendo de la máquina.

Tipos de datos

- La memoria de una computadora está organizada en unidades llamadas ***bytes***.
- En la mayor parte de las arquitecturas, un ***byte (octeto)*** está formado por ***ocho bits.*** Cada ***bit*** toma valores de **1** o **0***.*
- Cada octeto tiene una **dirección única** en la memoria, identificada por un número de 16, 32 o 64 bits, dependiendo de la máquina.

Tipos de datos

Tipos de datos

- La unidad procesada por las instrucciones del código máquina, se llama **una palabra (word)**. Hoy, en la mayoría de las computadoras, las palabras son de **32** o **64** bits. Las palabras son interpretadas como un número binario.
  - una palabra de 32 bits puede representar valores enteros sin signo desde el 0 hasta el 2^{32}-1 o valores enteros con signo desde -2^{31} a 2^{31}-1

Tipos de datos

- El tipo de datos define dos cosas:
  - ➡ el **número de octetos** (8 bits) requeridos para representar el dato y
  - ➡ la **manera de usar** estos octetos.
- Los tipos elementales en muchos lenguajes de programación son:
  - enteros - **int**
  - flotantes (reales) - **float**
  - caracteres - **char**
- El rango/precisión de los tipos depende del número de bits que se utilicen para representarlos.

Tipos de datos

• En C++ el compromiso entre espacio y precisión se decide eligiendo entre los tipos **int, long int, short int** para enteros y entre **float** o **double** para números de punto flotante.

• No hay estándar sobre el tamaño que toman estos tipos y es dependiente del hardware, sin embargo C++ ofrece ciertas garantías:

Tipos de datos

• En C++ el compromiso entre espacio y precisión se decide eligiendo entre los tipos **int, long int, short int** para enteros y entre **float** o **double** para números de punto flotante.

• No hay estándar sobre el tamaño que toman estos tipos y es dependiente del hardware, sin embargo C++ ofrece ciertas garantías:

$$1 == \text{sizeof(char)} <= \text{sizeof(short)} <= \text{sizeof(int)} <= \text{sizeof(long)}$$

$$\text{sizeof(float)} <= \text{sizeof(double)} <= \text{sizeof(long double)}$$

$$\text{sizeof(I)} == \text{sizeof(signed I)} == \text{sizeof(unsigned I)}$$

donde I puede ser char, short, int o long.

además se garantiza que un char tiene al menos 8 bits, un short al menos 16 bits y un long al menos 32 bits

Tipos de datos

- Generalmente se piensa en el rango de valores que podremos almacenar en lugar de el número de octetos que tiene disponible el tipo.

- Para una máquina de 32 bits típica:

Tipos de datos

• Generalmente se piensa en el rango de valores que podremos almacenar en lugar de el número de octetos que tiene disponible el tipo.

• Para una máquina de 32 bits típica:

Tipo	Tamaño en bytes	Valores
char	1	[-128,127]
short	2	[-32768,32767]
int	4	[-2147483648, 2147483647]
long	4	[-2147483648, 2147483647]
float	4	$[3.4x10^{-38}, 3.4x10^{38}]$
double	8	$[1.7x10^{-308}, 1.7x10^{308}]$
long double	10	$[3.4x10^{-4932}, 3.4x10^{4932}]$

Representación de números **enteros** (sin signo)

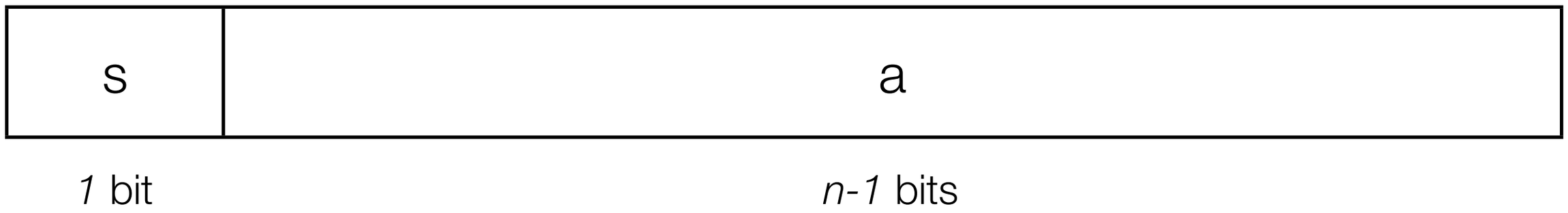
- el término **entero** se utiliza para referirse a un tipo de datos que representa un subconjunto finito de los enteros matemáticos.

- un tipo ***entero*** con **n** bits puede codificar $\mathbf{2^n}$ números; por ejemplo un ***entero sin signo*** representa típicamente los valores no-negativos de 0 a 2^n-1.

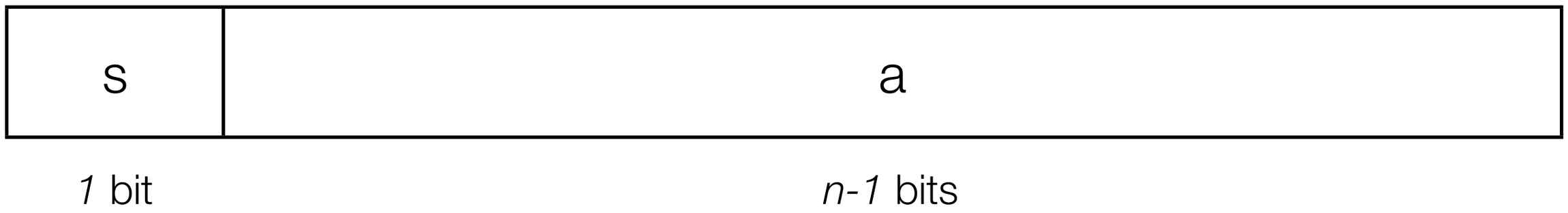
$$b = \sum_{i=0}^{n-1} b_i 2^i$$

- ¿y si queremos representar números con signo?

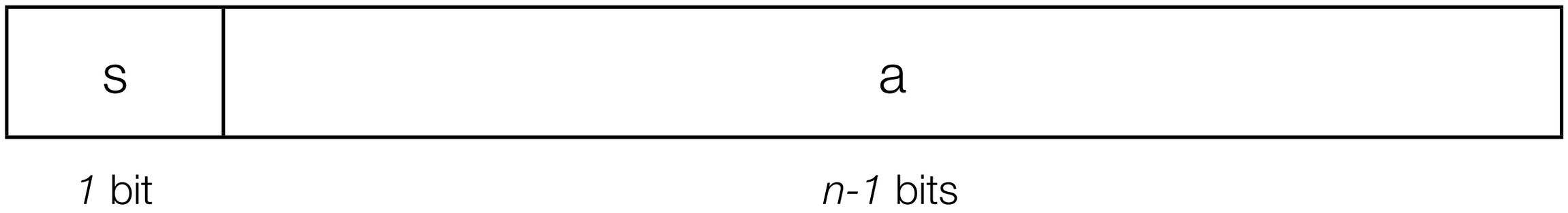
Representación de números **enteros** (con signo)



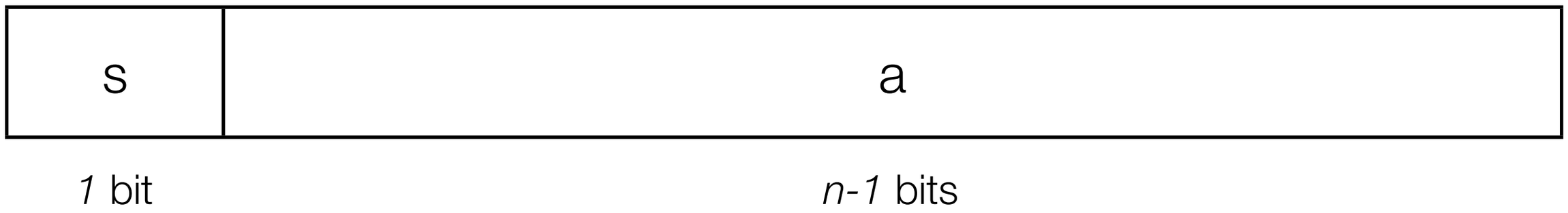
Representación de números **enteros** (con signo)



Representación de números **enteros** (con signo)

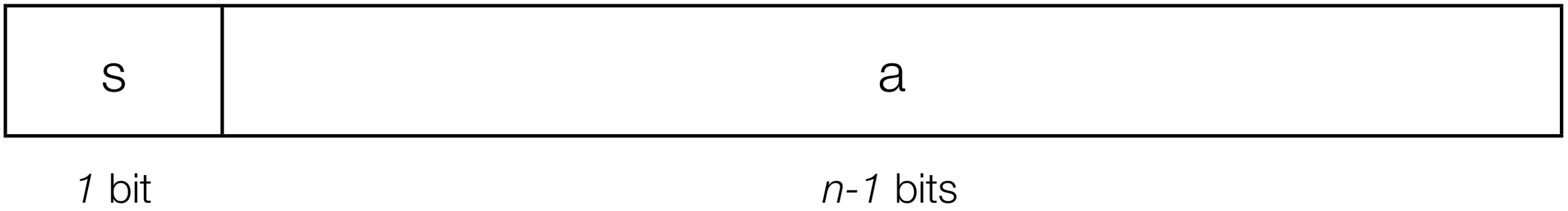


Representación de números **enteros** (con signo)



- ¿cómo sumar 3 + (-3) con esta representación?

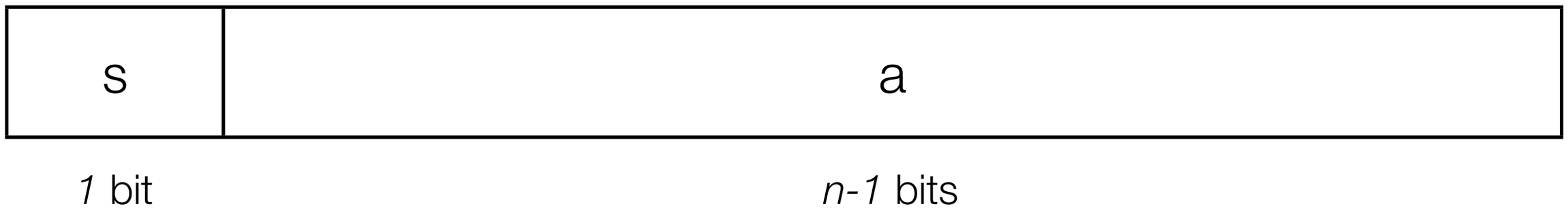
Representación de números **enteros** (con signo)



- ¿cómo sumar 3 + (-3) con esta representación?

- Tenemos

Representación de números **enteros** (con signo)

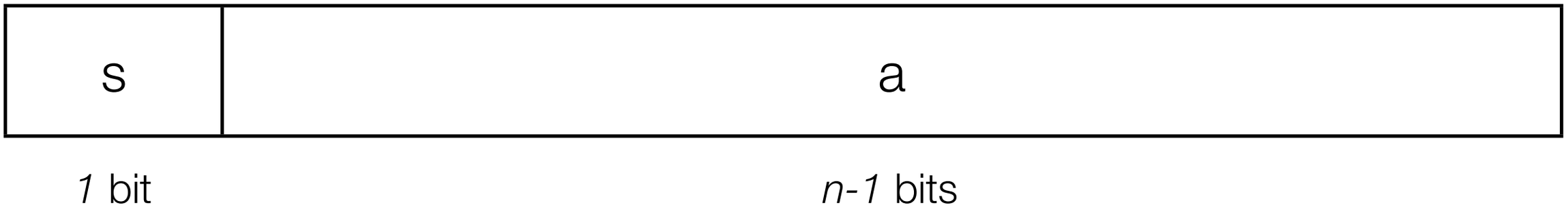


- ¿cómo sumar 3 + (-3) con esta representación?

- Tenemos

  ➡ doble representación del cero!

Representación de números **enteros** (con signo)



- ¿cómo sumar 3 + (-3) con esta representación?

- Tenemos

  ➡ doble representación del cero!

- En el caso de ***enteros con signo,*** los enteros positivos se representan con n-1 bits (por el signo) y los enteros negativos se representan con su **complemento a dos**. $C_2(N) = 2^{n-1} - N$

Representación de números **enteros** (complemento a dos)

Complemento a dos para 8-bits

Representación de números **enteros** (complemento a dos)

signo	**Complemento a dos para 8-bits**								
0	1	1	1	1	1	1	1	=	127
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	0	0	=	0
1	1	1	1	1	1	1	1	=	-1
1	1	1	1	1	1	1	0	=	-2
1	0	0	0	0	0	0	1	=	-127
1	0	0	0	0	0	0	0	=	-128

Representación de números **enteros** (complemento a dos)

Representación de números **enteros** (complemento a dos)

- La aritmética de complemento a dos es conveniente porque

Representación de números **enteros** (complemento a dos)

- La aritmética de complemento a dos es conveniente porque
  - hay una correspondencia uno-a-uno con la representación de valores (representación única de cero)

Representación de números **enteros** (complemento a dos)

- La aritmética de complemento a dos es conveniente porque
  - hay una correspondencia uno-a-uno con la representación de valores (representación única de cero)
  - porque la suma, resta y multiplicación no necesitan distinguir entre valores positivos y negativos.

Representación de números **enteros** (complemento a dos)

- La aritmética de complemento a dos es conveniente porque
  - hay una correspondencia uno-a-uno con la representación de valores (representación única de cero)
  - porque la suma, resta y multiplicación no necesitan distinguir entre valores positivos y negativos.
  - ¿rango?

Representación de números **enteros** (complemento a dos)

- La aritmética de complemento a dos es conveniente porque
  - hay una correspondencia uno-a-uno con la representación de valores (representación única de cero)
  - porque la suma, resta y multiplicación no necesitan distinguir entre valores positivos y negativos.
  - ¿rango?
    - valores de $-2^{(n-1)}$ a $2^{(n-1)}-1$.

Representación de números **enteros** (complemento a dos)

- La aritmética de complemento a dos es conveniente porque
  - hay una correspondencia uno-a-uno con la representación de valores (representación única de cero)
  - porque la suma, resta y multiplicación no necesitan distinguir entre valores positivos y negativos.
  - ¿rango?
    - valores de $-2^{(n-1)}$ a $2^{(n-1)}-1$.
  - Ejercicio: ¿cómo representamos 131 y -131 en un short ?