

# Complejidad computacional (repaso)

Programación III - UNGS

# Algoritmos

---

- ¿Qué es un algoritmo?
- ¿Qué es un buen algoritmo?
- Dados dos algoritmos para resolver un mismo problema, ¿cuál es mejor?
- ¿Cuándo un problema está bien resuelto?

## Complejidad computacional

---

- Una forma habitual de medir el tiempo de ejecución de un algoritmo es evaluar el **peor caso** en función del **tamaño de la entrada** del algoritmo.

# Complejidad computacional

---

- Una forma habitual de medir el tiempo de ejecución de un algoritmo es evaluar el **peor caso** en función del **tamaño de la entrada** del algoritmo.
  1. **Modelo uniforme:** Se asume que cada dato individual ocupa una posición individual de memoria. Por ejemplo, decimos que un arreglo de  $n$  elementos tiene tamaño  $n$ .

# Complejidad computacional

---

- Una forma habitual de medir el tiempo de ejecución de un algoritmo es evaluar el **peor caso** en función del **tamaño de la entrada** del algoritmo.
  1. **Modelo uniforme:** Se asume que cada dato individual ocupa una posición individual de memoria. Por ejemplo, decimos que un arreglo de  $n$  elementos tiene tamaño  $n$ .
  2. **Modelo logarítmico:** Se mide el tamaño en bits de cada dato individual. En este caso, el tamaño de un arreglo  $a$  se define como  $\sum_{i=1}^n \lceil \log_2(a[i]) + 1 \rceil$ .

# Complejidad computacional

---

- Una forma habitual de medir el tiempo de ejecución de un algoritmo es evaluar el **peor caso** en función del **tamaño de la entrada** del algoritmo.
  1. **Modelo uniforme:** Se asume que cada dato individual ocupa una posición individual de memoria. Por ejemplo, decimos que un arreglo de  $n$  elementos tiene tamaño  $n$ .
  2. **Modelo logarítmico:** Se mide el tamaño en bits de cada dato individual. En este caso, el tamaño de un arreglo  $a$  se define como  $\sum_{i=1}^n \lceil \log_2(a[i]) + 1 \rceil$ .
- En esta materia utilizamos el modelo uniforme para los análisis de complejidad computacional.

# Complejidad computacional

---

- **Definición.** La **función de complejidad** de un algoritmo es una función  $f : \mathbb{N} \rightarrow \mathbb{N}$  tal que  $f(n)$  es la cantidad de operaciones que realiza el algoritmo en el peor caso cuando toma una entrada de tamaño  $n$ .

# Complejidad computacional

---

- **Definición.** La **función de complejidad** de un algoritmo es una función  $f : \mathbb{N} \rightarrow \mathbb{N}$  tal que  $f(n)$  es la cantidad de operaciones que realiza el algoritmo en el peor caso cuando toma una entrada de tamaño  $n$ .
- Algunas observaciones:

# Complejidad computacional

---

- **Definición.** La **función de complejidad** de un algoritmo es una función  $f : \mathbb{N} \rightarrow \mathbb{N}$  tal que  $f(n)$  es la cantidad de operaciones que realiza el algoritmo en el peor caso cuando toma una entrada de tamaño  $n$ .
- Algunas observaciones:
  1. Medimos la cantidad de operaciones en lugar del tiempo total (por qué?).

# Complejidad computacional

---

- **Definición.** La **función de complejidad** de un algoritmo es una función  $f : \mathbb{N} \rightarrow \mathbb{N}$  tal que  $f(n)$  es la cantidad de operaciones que realiza el algoritmo en el peor caso cuando toma una entrada de tamaño  $n$ .
- Algunas observaciones:
  1. Medimos la cantidad de operaciones en lugar del tiempo total (por qué?).
  2. Nos interesa el peor caso del algoritmo (por qué??).

# Complejidad computacional

---

- **Definición.** La **función de complejidad** de un algoritmo es una función  $f : \mathbb{N} \rightarrow \mathbb{N}$  tal que  $f(n)$  es la cantidad de operaciones que realiza el algoritmo en el peor caso cuando toma una entrada de tamaño  $n$ .
- Algunas observaciones:
  1. Medimos la cantidad de operaciones en lugar del tiempo total (por qué?).
  2. Nos interesa el peor caso del algoritmo (por qué??).
  3. La complejidad se mide en función del tamaño de la entrada y no de la entrada particular (por qué???)

## Notación “O grande”

---

- **Definición.** Si  $f$  y  $g$  son dos funciones, decimos que  $f \in O(g)$  si existen  $\alpha \in \mathbb{R}$  y  $n_0 \in \mathbb{N}$  tales que

$$f(n) \leq \alpha g(n) \quad \text{para todo } n \geq n_0.$$

## Notación “O grande”

---

- **Definición.** Si  $f$  y  $g$  son dos funciones, decimos que  $f \in O(g)$  si existen  $\alpha \in \mathbb{R}$  y  $n_0 \in \mathbb{N}$  tales que

$$f(n) \leq \alpha g(n) \quad \text{para todo } n \geq n_0.$$

- Intuitivamente,  $f \in O(g)$  si  $g(n)$  “le gana” a  $f(n)$  para valores grandes de  $n$ .

## Notación “O grande”

---

- **Definición.** Si  $f$  y  $g$  son dos funciones, decimos que  $f \in O(g)$  si existen  $\alpha \in \mathbb{R}$  y  $n_0 \in \mathbb{N}$  tales que

$$f(n) \leq \alpha g(n) \quad \text{para todo } n \geq n_0.$$

- Intuitivamente,  $f \in O(g)$  si  $g(n)$  “le gana” a  $f(n)$  para valores grandes de  $n$ .
- Ejemplos:

## Notación “O grande”

---

- **Definición.** Si  $f$  y  $g$  son dos funciones, decimos que  $f \in O(g)$  si existen  $\alpha \in \mathbb{R}$  y  $n_0 \in \mathbb{N}$  tales que

$$f(n) \leq \alpha g(n) \quad \text{para todo } n \geq n_0.$$

- Intuitivamente,  $f \in O(g)$  si  $g(n)$  “le gana” a  $f(n)$  para valores grandes de  $n$ .
- Ejemplos:
  - $n \in O(n^2)$ .

## Notación “O grande”

---

- **Definición.** Si  $f$  y  $g$  son dos funciones, decimos que  $f \in O(g)$  si existen  $\alpha \in \mathbb{R}$  y  $n_0 \in \mathbb{N}$  tales que

$$f(n) \leq \alpha g(n) \quad \text{para todo } n \geq n_0.$$

- Intuitivamente,  $f \in O(g)$  si  $g(n)$  “le gana” a  $f(n)$  para valores grandes de  $n$ .
- Ejemplos:
  - $n \in O(n^2)$ .
  - $n^2 \notin O(n)$ .

## Notación “O grande”

---

- **Definición.** Si  $f$  y  $g$  son dos funciones, decimos que  $f \in O(g)$  si existen  $\alpha \in \mathbb{R}$  y  $n_0 \in \mathbb{N}$  tales que

$$f(n) \leq \alpha g(n) \quad \text{para todo } n \geq n_0.$$

- Intuitivamente,  $f \in O(g)$  si  $g(n)$  “le gana” a  $f(n)$  para valores grandes de  $n$ .
- Ejemplos:
  - $n \in O(n^2)$ .
  - $n^2 \notin O(n)$ .
  - $100n \in O(n^2)$ .

## Notación “O grande”

---

- **Definición.** Si  $f$  y  $g$  son dos funciones, decimos que  $f \in O(g)$  si existen  $\alpha \in \mathbb{R}$  y  $n_0 \in \mathbb{N}$  tales que

$$f(n) \leq \alpha g(n) \quad \text{para todo } n \geq n_0.$$

- Intuitivamente,  $f \in O(g)$  si  $g(n)$  “le gana” a  $f(n)$  para valores grandes de  $n$ .
- Ejemplos:
  - $n \in O(n^2)$ .
  - $n^2 \notin O(n)$ .
  - $100n \in O(n^2)$ .
  - $4n^2 \in O(2n^2)$  (y a la inversa).

## Complejidad computacional

---

- Utilizamos la notación “O grande” para especificar la función de complejidad  $f$  de los algoritmos (por qué?!?).

## Complejidad computacional

---

- Utilizamos la notación “O grande” para especificar la función de complejidad  $f$  de los algoritmos (por qué?!?).
  - Si  $f \in O(n)$ , decimos que el algoritmo es **lineal**.

## Complejidad computacional

---

- Utilizamos la notación “O grande” para especificar la función de complejidad  $f$  de los algoritmos (por qué?!?).
  - Si  $f \in O(n)$ , decimos que el algoritmo es **lineal**.
  - Si  $f \in O(n^2)$ , decimos que el algoritmo es **cuadrático**.

# Complejidad computacional

---

- Utilizamos la notación “O grande” para especificar la función de complejidad  $f$  de los algoritmos (por qué?!?).
  - Si  $f \in O(n)$ , decimos que el algoritmo es **lineal**.
  - Si  $f \in O(n^2)$ , decimos que el algoritmo es **cuadrático**.
  - Si  $f \in O(n^3)$ , decimos que el algoritmo es **cúbico**.

## Complejidad computacional

---

- Utilizamos la notación “O grande” para especificar la función de complejidad  $f$  de los algoritmos (por qué?!?).
  - Si  $f \in O(n)$ , decimos que el algoritmo es **lineal**.
  - Si  $f \in O(n^2)$ , decimos que el algoritmo es **cuadrático**.
  - Si  $f \in O(n^3)$ , decimos que el algoritmo es **cúbico**.
  - En general, si  $f \in O(n^k)$ , decimos que el algoritmo es **polinomial**.

# Complejidad computacional

---

- Utilizamos la notación “O grande” para especificar la función de complejidad  $f$  de los algoritmos (por qué?!?).
  - Si  $f \in O(n)$ , decimos que el algoritmo es **lineal**.
  - Si  $f \in O(n^2)$ , decimos que el algoritmo es **cuadrático**.
  - Si  $f \in O(n^3)$ , decimos que el algoritmo es **cúbico**.
  - En general, si  $f \in O(n^k)$ , decimos que el algoritmo es **polinomial**.
  - Si  $f \in O(2^n)$  o similar, decimos que el algoritmo es **exponencial**.

## Ejemplos

---

- Búsqueda secuencial:  $O(n)$ .
- Búsqueda binaria:  $O(\log(n))$ .
- Ordenar un arreglo (bubblesort):  $O(n^2)$ .
- Ordenar un arreglo (quicksort):  $O(n^2)$  en el peor caso (!).
- Ordenar un arreglo (heapsort, mergesort):  $O(n \log(n))$ .

## Ejemplos

---

- Búsqueda secuencial:  $O(n)$ .
- Búsqueda binaria:  $O(\log(n))$ .
- Ordenar un arreglo (bubblesort):  $O(n^2)$ .
- Ordenar un arreglo (quicksort):  $O(n^2)$  en el peor caso (!).
- Ordenar un arreglo (heapsort, mergesort):  $O(n \log(n))$ .

Es interesante notar que  $O(n \log(n))$  es la complejidad **óptima** para algoritmos de ordenamiento basados en comparaciones.

# Problemas bien resueltos

---

**Definición:** Decimos que un problema está **bien resuelto** si existe un algoritmo de complejidad polinomial para el problema.

## Problemas bien resueltos

---

**Definición:** Decimos que un problema está **bien resuelto** si existe un algoritmo de complejidad polinomial para el problema.

---

$n = 10$

$n = 20$

$n = 30$

$n = 40$

$n = 50$

---

# Problemas bien resueltos

---

**Definición:** Decimos que un problema está **bien resuelto** si existe un algoritmo de complejidad polinomial para el problema.

	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
$O(n)$	0.01 ms	0.02 ms	0.03 ms	0.04 ms	0.05 ms

# Problemas bien resueltos

---

**Definición:** Decimos que un problema está **bien resuelto** si existe un algoritmo de complejidad polinomial para el problema.

	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
$O(n)$	0.01 ms	0.02 ms	0.03 ms	0.04 ms	0.05 ms
$O(n \log n)$	0.01 ms	0.03 ms	0.04 ms	0.06 ms	0.08 ms

# Problemas bien resueltos

---

**Definición:** Decimos que un problema está **bien resuelto** si existe un algoritmo de complejidad polinomial para el problema.

	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
$O(n)$	0.01 ms	0.02 ms	0.03 ms	0.04 ms	0.05 ms
$O(n \log n)$	0.01 ms	0.03 ms	0.04 ms	0.06 ms	0.08 ms
$O(n^2)$	0.10 ms	0.40 ms	0.90 ms	1.60 ms	2.50 ms

# Problemas bien resueltos

---

**Definición:** Decimos que un problema está **bien resuelto** si existe un algoritmo de complejidad polinomial para el problema.

	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
$O(n)$	0.01 ms	0.02 ms	0.03 ms	0.04 ms	0.05 ms
$O(n \log n)$	0.01 ms	0.03 ms	0.04 ms	0.06 ms	0.08 ms
$O(n^2)$	0.10 ms	0.40 ms	0.90 ms	1.60 ms	2.50 ms
$O(n^3)$	1.00 ms	8.00 ms	27.00 ms	64.00 ms	0.12 sg

# Problemas bien resueltos

---

**Definición:** Decimos que un problema está **bien resuelto** si existe un algoritmo de complejidad polinomial para el problema.

	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
$O(n)$	0.01 ms	0.02 ms	0.03 ms	0.04 ms	0.05 ms
$O(n \log n)$	0.01 ms	0.03 ms	0.04 ms	0.06 ms	0.08 ms
$O(n^2)$	0.10 ms	0.40 ms	0.90 ms	1.60 ms	2.50 ms
$O(n^3)$	1.00 ms	8.00 ms	27.00 ms	64.00 ms	0.12 sg
$O(2^n)$	1.02 ms	1.04 sg	17.90 min	12 días	35 años

# Problemas bien resueltos

---

**Definición:** Decimos que un problema está **bien resuelto** si existe un algoritmo de complejidad polinomial para el problema.

	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
$O(n)$	0.01 ms	0.02 ms	0.03 ms	0.04 ms	0.05 ms
$O(n \log n)$	0.01 ms	0.03 ms	0.04 ms	0.06 ms	0.08 ms
$O(n^2)$	0.10 ms	0.40 ms	0.90 ms	1.60 ms	2.50 ms
$O(n^3)$	1.00 ms	8.00 ms	27.00 ms	64.00 ms	0.12 sg
$O(2^n)$	1.02 ms	1.04 sg	17.90 min	12 días	35 años
$O(3^n)$	0.59 sg	58 min	6 años	3855 siglos	$2 \times 10^8$ siglos!

## Problemas bien resueltos

---

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

## Problemas bien resueltos

---

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

92 kIPS	2.66 MIPS	9.7 MIPS	177 MIPS
IBM 4004 (1971)	Intel 286 (1982)	Pentium IV (2000)	Intel Core i7 (2011)

# Problemas bien resueltos

---

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

	92 kIPS IBM 4004 (1971)	2.66 MIPS Intel 286 (1982)	9.7 MIPS Pentium IV (2000)	177 MIPS Intel Core i7 (2011)
$O(n)$	5520	$1.5 \cdot 10^5$	$5.8 \cdot 10^5$	$1.06 \cdot 10^7$ (1923x)

# Problemas bien resueltos

---

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

	92 kIPS IBM 4004 (1971)	2.66 MIPS Intel 286 (1982)	9.7 MIPS Pentium IV (2000)	177 MIPS Intel Core i7 (2011)	
$O(n)$	5520	$1.5 \cdot 10^5$	$5.8 \cdot 10^5$	$1.06 \cdot 10^7$	(1923x)
$O(n \log n)$	4141	$1.0 \cdot 10^5$	$2.7 \cdot 10^5$	$6.19 \cdot 10^6$	(1495x)

# Problemas bien resueltos

---

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

	92 kIPS IBM 4004 (1971)	2.66 MIPS Intel 286 (1982)	9.7 MIPS Pentium IV (2000)	177 MIPS Intel Core i7 (2011)	
$O(n)$	5520	$1.5 \cdot 10^5$	$5.8 \cdot 10^5$	$1.06 \cdot 10^7$	(1923x)
$O(n \log n)$	4141	$1.0 \cdot 10^5$	$2.7 \cdot 10^5$	$6.19 \cdot 10^6$	(1495x)
$O(n^2)$	74	399	762	3258	(43x)

# Problemas bien resueltos

---

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

	92 kIPS IBM 4004 (1971)	2.66 MIPS Intel 286 (1982)	9.7 MIPS Pentium IV (2000)	177 MIPS Intel Core i7 (2011)	
$O(n)$	5520	$1.5 \cdot 10^5$	$5.8 \cdot 10^5$	$1.06 \cdot 10^7$	(1923x)
$O(n \log n)$	4141	$1.0 \cdot 10^5$	$2.7 \cdot 10^5$	$6.19 \cdot 10^6$	(1495x)
$O(n^2)$	74	399	762	3258	(43x)
$O(n^3)$	17	54	83	219	(12x)

# Problemas bien resueltos

---

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

	92 kIPS IBM 4004 (1971)	2.66 MIPS Intel 286 (1982)	9.7 MIPS Pentium IV (2000)	177 MIPS Intel Core i7 (2011)	
$O(n)$	5520	$1.5 \cdot 10^5$	$5.8 \cdot 10^5$	$1.06 \cdot 10^7$	(1923x)
$O(n \log n)$	4141	$1.0 \cdot 10^5$	$2.7 \cdot 10^5$	$6.19 \cdot 10^6$	(1495x)
$O(n^2)$	74	399	762	3258	(43x)
$O(n^3)$	17	54	83	219	(12x)
$O(2^n)$	12	17	19	23	(1.88x)

# Problemas bien resueltos

---

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

	92 kIPS IBM 4004 (1971)	2.66 MIPS Intel 286 (1982)	9.7 MIPS Pentium IV (2000)	177 MIPS Intel Core i7 (2011)	
$O(n)$	5520	$1.5 \cdot 10^5$	$5.8 \cdot 10^5$	$1.06 \cdot 10^7$	(1923x)
$O(n \log n)$	4141	$1.0 \cdot 10^5$	$2.7 \cdot 10^5$	$6.19 \cdot 10^6$	(1495x)
$O(n^2)$	74	399	762	3258	(43x)
$O(n^3)$	17	54	83	219	(12x)
$O(2^n)$	12	17	19	23	(1.88x)
$O(3^n)$	7	10	12	14	(1.88x)

## Problemas bien resueltos

---

**Conclusión:** Los algoritmos polinomiales se consideran satisfactorios (cuanto menor sea el grado, mejor), y los algoritmos supra-polinomiales se consideran no satisfactorios.

## Problemas bien resueltos

---

**Conclusión:** Los algoritmos polinomiales se consideran satisfactorios (cuanto menor sea el grado, mejor), y los algoritmos supra-polinomiales se consideran no satisfactorios.

- Si los tamaños de instancia son pequeños, ¿es tan malo un algoritmo exponencial?

## Problemas bien resueltos

---

**Conclusión:** Los algoritmos polinomiales se consideran satisfactorios (cuanto menor sea el grado, mejor), y los algoritmos supra-polinomiales se consideran no satisfactorios.

- Si los tamaños de instancia son pequeños, ¿es tan malo un algoritmo exponencial?
- ¿Cómo se comparan  $O(n^{85})$  con  $O(1.001^n)$ ?

## Problemas bien resueltos

---

**Conclusión:** Los algoritmos polinomiales se consideran satisfactorios (cuanto menor sea el grado, mejor), y los algoritmos supra-polinomiales se consideran no satisfactorios.

- Si los tamaños de instancia son pequeños, ¿es tan malo un algoritmo exponencial?
- ¿Cómo se comparan  $O(n^{85})$  con  $O(1.001^n)$ ?
- ¿Puede pasar que un algoritmo de peor caso exponencial sea eficiente en la práctica? ¿Puede pasar que en la práctica sea *el mejor*?

## Problemas bien resueltos

---

**Conclusión:** Los algoritmos polinomiales se consideran satisfactorios (cuanto menor sea el grado, mejor), y los algoritmos supra-polinomiales se consideran no satisfactorios.

- Si los tamaños de instancia son pequeños, ¿es tan malo un algoritmo exponencial?
- ¿Cómo se comparan  $O(n^{85})$  con  $O(1.001^n)$ ?
- ¿Puede pasar que un algoritmo de peor caso exponencial sea eficiente en la práctica? ¿Puede pasar que en la práctica sea *el mejor*?
- ¿Qué pasa si no encuentro un algoritmo polinomial?

# Divide and conquer

Programación III - UNGS

## Divide and conquer

---

- La técnica **divide and conquer** (divide y vencerás) es una idea para diseñar algoritmos recursivos.

## Divide and conquer

---

- La técnica **divide and conquer** (divide y vencerás) es una idea para diseñar algoritmos recursivos.
- Un algoritmo divide and conquer **divide recursivamente** el problema en dos o más subproblemas del mismo tipo (o similar), hasta que los subproblemas son tan pequeños que se pueden resolver directamente.

## Divide and conquer

---

- La técnica **divide and conquer** (divide y vencerás) es una idea para diseñar algoritmos recursivos.
- Un algoritmo divide and conquer **divide recursivamente** el problema en dos o más subproblemas del mismo tipo (o similar), hasta que los subproblemas son tan pequeños que se pueden resolver directamente.
- Una vez hecho esto, las soluciones de los subproblemas se **combinan** para obtener una solución del problema original.

## Divide and conquer

---

- La técnica **divide and conquer** (divide y vencerás) es una idea para diseñar algoritmos recursivos.
- Un algoritmo divide and conquer **divide recursivamente** el problema en dos o más subproblemas del mismo tipo (o similar), hasta que los subproblemas son tan pequeños que se pueden resolver directamente.
- Una vez hecho esto, las soluciones de los subproblemas se **combinan** para obtener una solución del problema original.
- Esta técnica es la base de algoritmos eficientes para muchos problemas, como **ordenamiento de arreglos**, **multiplicación de números grandes**, y el cálculo de la **transformada discreta de Fourier**, entre otros.

## Divide and conquer

---

- Si la instancia  $I$  de entrada es pequeña, entonces utilizar un algoritmo ad hoc para el problema.

## Divide and conquer

---

- Si la instancia  $I$  de entrada es pequeña, entonces utilizar un algoritmo ad hoc para el problema.
- En caso contrario:

## Divide and conquer

---

- Si la instancia  $I$  de entrada es pequeña, entonces utilizar un algoritmo ad hoc para el problema.
- En caso contrario:
  1. **Dividir**  $I$  en sub-instancias  $I_1, I_2, \dots, I_k$  más pequeñas.

## Divide and conquer

---

- Si la instancia  $I$  de entrada es pequeña, entonces utilizar un algoritmo ad hoc para el problema.
- En caso contrario:
  1. **Dividir**  $I$  en sub-instancias  $I_1, I_2, \dots, I_k$  más pequeñas.
  2. Resolver **recursivamente** las  $k$  sub-instancias.

## Divide and conquer

---

- Si la instancia  $I$  de entrada es pequeña, entonces utilizar un algoritmo ad hoc para el problema.
- En caso contrario:
  1. **Dividir**  $I$  en sub-instancias  $I_1, I_2, \dots, I_k$  más pequeñas.
  2. Resolver **recursivamente** las  $k$  sub-instancias.
  3. **Combinar** las soluciones para las  $k$  sub-instancias para obtener una solución para la instancia original  $I$ .

## Divide and conquer

---

- Dos de los algoritmos de **ordenamiento de arreglos** más eficientes que se conocen están basados en divide and conquer: el algoritmo **mergesort** (von Neumann, 1945) y el algoritmo **quicksort** (Hoare, 1960).



John von Neumann (1903–1957)



Tony Hoare (1934–)

## Ejemplo: Mergesort

---

Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier método sencillo.

## Ejemplo: Mergesort

---

Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier método sencillo.
- Si  $n$  es grande:

## Ejemplo: Mergesort

---

Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier método sencillo.
- Si  $n$  es grande:
  - $A_1 :=$  primera mitad de  $A$ .

## Ejemplo: Mergesort

---

Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier método sencillo.
- Si  $n$  es grande:
  - $A_1 :=$  primera mitad de  $A$ .
  - $A_2 :=$  segunda mitad de  $A$ .

## Ejemplo: Mergesort

---

Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier método sencillo.
- Si  $n$  es grande:
  - $A_1 :=$  primera mitad de  $A$ .
  - $A_2 :=$  segunda mitad de  $A$ .
  - Ordenar recursivamente  $A_1$  y  $A_2$  por separado.

## Ejemplo: Mergesort

---

Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier método sencillo.
- Si  $n$  es grande:
  - $A_1 :=$  primera mitad de  $A$ .
  - $A_2 :=$  segunda mitad de  $A$ .
  - Ordenar recursivamente  $A_1$  y  $A_2$  por separado.
  - Combinar  $A_1$  y  $A_2$  para obtener los elementos de  $A$  ordenados (apareo de arreglos).

## Ejemplo: Mergesort

---

Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier método sencillo.
- Si  $n$  es grande:
  - $A_1 :=$  primera mitad de  $A$ .
  - $A_2 :=$  segunda mitad de  $A$ .
  - Ordenar recursivamente  $A_1$  y  $A_2$  por separado.
  - Combinar  $A_1$  y  $A_2$  para obtener los elementos de  $A$  ordenados (apareo de arreglos).

## Ejemplo: Mergesort

---

Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier método sencillo.
- Si  $n$  es grande:
  - $A_1 :=$  primera mitad de  $A$ .
  - $A_2 :=$  segunda mitad de  $A$ .
  - Ordenar recursivamente  $A_1$  y  $A_2$  por separado.
  - Combinar  $A_1$  y  $A_2$  para obtener los elementos de  $A$  ordenados (apareo de arreglos).

Este algoritmo contiene todos los elementos típicos de la técnica *divide and conquer*.

## Ejemplo: Mergesort

---

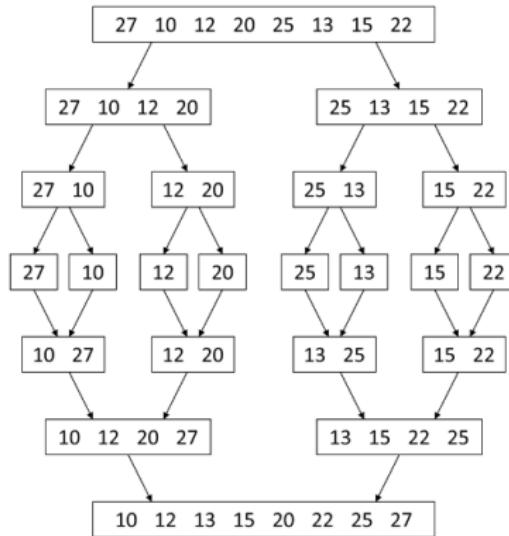
Algoritmo *divide and conquer* para ordenar un arreglo  $A$  de  $n$  elementos.

- Si  $n$  es pequeño, ordenar por cualquier **método sencillo**.
- Si  $n$  es grande:
  - $A_1 :=$  **primera mitad** de  $A$ .
  - $A_2 :=$  **segunda mitad** de  $A$ .
  - Ordenar **recursivamente**  $A_1$  y  $A_2$  por separado.
  - **Combinar**  $A_1$  y  $A_2$  para obtener los elementos de  $A$  ordenados (apareo de arreglos).

Este algoritmo contiene todos los elementos típicos de la técnica *divide and conquer*.

## Ejemplo: Mergesort

---



## Ejemplo: Quicksort

---

- El algoritmo **quicksort** para ordenar un arreglo es otro ejemplo de un algoritmo basado en divide and conquer.

## Ejemplo: Quicksort

---

- El algoritmo **quicksort** para ordenar un arreglo es otro ejemplo de un algoritmo basado en divide and conquer.
  1. Seleccionar un elemento  $x$  del arreglo (llamado el **pivote**).

## Ejemplo: Quicksort

---

- El algoritmo **quicksort** para ordenar un arreglo es otro ejemplo de un algoritmo basado en divide and conquer.
  1. Seleccionar un elemento  $x$  del arreglo (llamado el **pivote**).
  2. Permutar el arreglo para que queden primero los elementos menores o iguales a  $x$ , luego el mismo  $x$  y finalmente los elementos mayores que  $x$ .

## Ejemplo: Quicksort

---

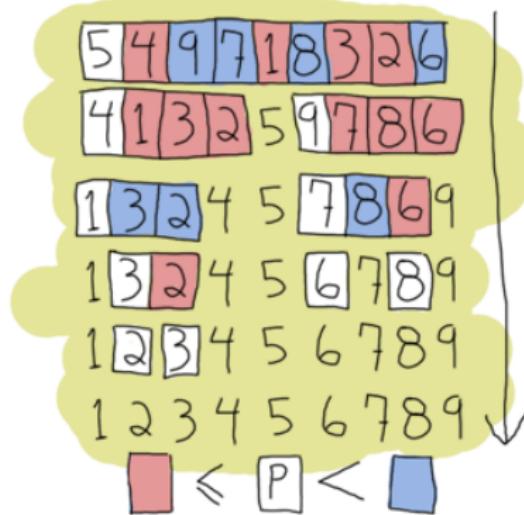
- El algoritmo **quicksort** para ordenar un arreglo es otro ejemplo de un algoritmo basado en divide and conquer.
  1. Seleccionar un elemento  $x$  del arreglo (llamado el **pivote**).
  2. Permutar el arreglo para que queden primero los elementos menores o iguales a  $x$ , luego el mismo  $x$  y finalmente los elementos mayores que  $x$ .
  3. Recursivamente aplicar el mismo procedimiento sobre las dos “mitades” del arreglo.

## Ejemplo: Quicksort

---

- El algoritmo **quicksort** para ordenar un arreglo es otro ejemplo de un algoritmo basado en divide and conquer.
  1. Seleccionar un elemento  $x$  del arreglo (llamado el **pivote**).
  2. Permutar el arreglo para que queden primero los elementos menores o iguales a  $x$ , luego el mismo  $x$  y finalmente los elementos mayores que  $x$ .
  3. Recursivamente aplicar el mismo procedimiento sobre las dos “mitades” del arreglo.
- Es una versión similar a mergesort, con la diferencia de que primero se separa el arreglo y luego se realizan las llamadas recursivas. En este caso, la combinación de las soluciones de los sub-problemas es sencilla.

## Ejemplo: Quicksort



## Ejemplo: Quicksort

---

- Quicksort tiene un **peor caso** de  $O(n^2)$ , que ocurre por ejemplo cuando el arreglo está ordenado al revés (¿por qué?).

## Ejemplo: Quicksort

---

- Quicksort tiene un **peor caso** de  $O(n^2)$ , que ocurre por ejemplo cuando el arreglo está ordenado al revés (**¿por qué?**).
- El **caso promedio**, sin embargo, es de  $O(n \log n)$ .

## Ejemplo: Quicksort

---

- Quicksort tiene un **peor caso** de  $O(n^2)$ , que ocurre por ejemplo cuando el arreglo está ordenado al revés (**¿por qué?**).
- El **caso promedio**, sin embargo, es de  $O(n \log n)$ .
- Se puede mejorar este algoritmo en la práctica con estas ideas:

## Ejemplo: Quicksort

---

- Quicksort tiene un **peor caso** de  $O(n^2)$ , que ocurre por ejemplo cuando el arreglo está ordenado al revés (**¿por qué?**).
- El **caso promedio**, sin embargo, es de  $O(n \log n)$ .
- Se puede mejorar este algoritmo en la práctica con estas ideas:
  1. Seleccionar el pivote **aleatoriamente**.

## Ejemplo: Quicksort

---

- Quicksort tiene un **peor caso** de  $O(n^2)$ , que ocurre por ejemplo cuando el arreglo está ordenado al revés (**¿por qué?**).
- El **caso promedio**, sin embargo, es de  $O(n \log n)$ .
- Se puede mejorar este algoritmo en la práctica con estas ideas:
  1. Seleccionar el pivote **aleatoriamente**.
  2. Cuando el sub-problema es pequeño, utilizar un algoritmo  $O(n^2)$  para ordenarlo, que en la práctica es más eficiente que el divide and conquer.

## Ejemplo: Multiplicando números grandes

---

- Supongamos que necesitamos trabajar con números **enteros arbitrariamente grandes** (por ejemplo, para aplicaciones de criptografía, o para calcular  $\pi$  con muchos decimales).

## Ejemplo: Multiplicando números grandes

---

- Supongamos que necesitamos trabajar con números **enteros arbitrariamente grandes** (por ejemplo, para aplicaciones de criptografía, o para calcular  $\pi$  con muchos decimales).
- No podemos usar los tipos numéricos primitivos tienen límites dados por el hardware.

## Ejemplo: Multiplicando números grandes

---

- Supongamos que necesitamos trabajar con números **enteros arbitrariamente grandes** (por ejemplo, para aplicaciones de criptografía, o para calcular  $\pi$  con muchos decimales).
- No podemos usar los tipos numéricos primitivos tienen límites dados por el hardware.
- En este caso, hay que **emular** la aritmética binaria del microprocesador, implementando números representados como **arreglos de bits** (típicamente arreglos de **boolean**).

## Ejemplo: Multiplicando números grandes

---

- Supongamos que necesitamos trabajar con números **enteros arbitrariamente grandes** (por ejemplo, para aplicaciones de criptografía, o para calcular  $\pi$  con muchos decimales).
- No podemos usar los tipos numéricos primitivos tienen límites dados por el hardware.
- En este caso, hay que **emular** la aritmética binaria del microprocesador, implementando números representados como **arreglos de bits** (típicamente arreglos de **boolean**).
- En Java, la clase **BigInteger** implementa esta idea.

## Ejemplo: Multiplicando números grandes

---

- El producto de dos enteros representados por arreglos de booleanos es  $O(n^2)$  (¿por qué?), y hasta 1960 se pensaba que esta complejidad era la mejor posible.

## Ejemplo: Multiplicando números grandes

---

- El producto de dos enteros representados por arreglos de booleanos es  $O(n^2)$  (¿por qué?), y hasta 1960 se pensaba que esta complejidad era la mejor posible.
- Sin embargo, en 1960 Anatolii Karatsuba descubrió un método más eficiente, basado en divide and conquer.



A. Karatsuba (1937–2008)

## Algoritmo de Karatsuba

---

- Supongamos que tenemos  $x = x_1 10^m + x_2$  e  $y = y_1 10^m + y_2$ . Podemos calcular el producto  $x \times y$  con el siguiente algoritmo:

## Algoritmo de Karatsuba

---

- Supongamos que tenemos  $x = x_1 10^m + x_2$  e  $y = y_1 10^m + y_2$ . Podemos calcular el producto  $x \times y$  con el siguiente algoritmo:
  1.  $A = x_1 \times y_1;$

## Algoritmo de Karatsuba

---

- Supongamos que tenemos  $x = x_1 10^m + x_2$  e  $y = y_1 10^m + y_2$ . Podemos calcular el producto  $x \times y$  con el siguiente algoritmo:
  1.  $A = x_1 \times y_1;$
  2.  $B = x_2 \times y_2;$

## Algoritmo de Karatsuba

---

- Supongamos que tenemos  $x = x_1 10^m + x_2$  e  $y = y_1 10^m + y_2$ . Podemos calcular el producto  $x \times y$  con el siguiente algoritmo:
  1.  $A = x_1 \times y_1;$
  2.  $B = x_2 \times y_2;$
  3.  $C = (x_1 + x_2) \times (y_1 + y_2);$

## Algoritmo de Karatsuba

---

- Supongamos que tenemos  $x = x_1 10^m + x_2$  e  $y = y_1 10^m + y_2$ . Podemos calcular el producto  $x \times y$  con el siguiente algoritmo:
  1.  $A = x_1 \times y_1;$
  2.  $B = x_2 \times y_2;$
  3.  $C = (x_1 + x_2) \times (y_1 + y_2);$
  4.  $K = C - A - B;$  (Notar que  $K = x_1 y_2 + x_2 y_1$ )

## Algoritmo de Karatsuba

---

- Supongamos que tenemos  $x = x_1 10^m + x_2$  e  $y = y_1 10^m + y_2$ . Podemos calcular el producto  $x \times y$  con el siguiente algoritmo:
  1.  $A = x_1 \times y_1;$
  2.  $B = x_2 \times y_2;$
  3.  $C = (x_1 + x_2) \times (y_1 + y_2);$
  4.  $K = C - A - B;$  (Notar que  $K = x_1 y_2 + x_2 y_1$ )
  5. Resultado =  $A \times 10^{2m} + K \times 10^m + B;$

## Algoritmo de Karatsuba

---

- Supongamos que tenemos  $x = x_1 10^m + x_2$  e  $y = y_1 10^m + y_2$ . Podemos calcular el producto  $x \times y$  con el siguiente algoritmo:
  1.  $A = x_1 \times y_1;$
  2.  $B = x_2 \times y_2;$
  3.  $C = (x_1 + x_2) \times (y_1 + y_2);$
  4.  $K = C - A - B;$  (Notar que  $K = x_1 y_2 + x_2 y_1$ )
  5. Resultado =  $A \times 10^{2m} + K \times 10^m + B;$
- Luego de estos pasos,  $A = xy.$

## Algoritmo de Karatsuba

---

- El elemento crucial del algoritmo es que podemos hacer todas las multiplicaciones **recursivamente** utilizando el mismo algoritmo de Karatsuba!

## Algoritmo de Karatsuba

---

- El elemento crucial del algoritmo es que podemos hacer todas las multiplicaciones **recursivamente** utilizando el mismo algoritmo de Karatsuba!
- En este sentido, se trata de un algoritmo de divide and conquer, porque calculamos el producto de dos números en función de los productos de **números más pequeños**.

## Algoritmo de Karatsuba

---

- El elemento crucial del algoritmo es que podemos hacer todas las multiplicaciones **recursivamente** utilizando el mismo algoritmo de Karatsuba!
- En este sentido, se trata de un algoritmo de divide and conquer, porque calculamos el producto de dos números en función de los productos de **números más pequeños**.
- Si  $t(n)$  es el tiempo del algoritmo para números de  $n$  bits, entonces

$$t(n) = 3t(\lceil n/2 \rceil) + 4O(n) + O(1).$$

## Algoritmo de Karatsuba

---

- El elemento crucial del algoritmo es que podemos hacer todas las multiplicaciones **recursivamente** utilizando el mismo algoritmo de Karatsuba!
- En este sentido, se trata de un algoritmo de divide and conquer, porque calculamos el producto de dos números en función de los productos de **números más pequeños**.
- Si  $t(n)$  es el tiempo del algoritmo para números de  $n$  bits, entonces

$$t(n) = 3t(\lceil n/2 \rceil) + 4O(n) + O(1).$$

- Se trata de una **ecuación de recurrencia**. ¿Cuál es la complejidad final?

# Algoritmo de Karatsuba

---

$$t(n) \cong 3t(n/2) + 4n$$

# Algoritmo de Karatsuba

---

$$\begin{aligned}t(n) &\cong 3t(n/2) + 4n \\&= 3\left(3t(n/4) + 4n/2\right) + 4n\end{aligned}$$

## Algoritmo de Karatsuba

---

$$\begin{aligned}t(n) &\cong 3t(n/2) + 4n \\&= 3\left(3t(n/4) + 4n/2\right) + 4n \\&= 3^2 t(n/4) + 4n (3/2 + 1)\end{aligned}$$

## Algoritmo de Karatsuba

---

$$\begin{aligned}t(n) &\cong 3t(n/2) + 4n \\&= 3\left(3t(n/4) + 4n/2\right) + 4n \\&= 3^2 t(n/4) + 4n (3/2 + 1) \\&= 3^2 \left(3t(n/8) + 4n/4\right) + 4n (3/2 + 1)\end{aligned}$$

## Algoritmo de Karatsuba

---

$$\begin{aligned}t(n) &\cong 3t(n/2) + 4n \\&= 3\left(3t(n/4) + 4n/2\right) + 4n \\&= 3^2 t(n/4) + 4n (3/2 + 1) \\&= 3^2 \left(3t(n/8) + 4n/4\right) + 4n (3/2 + 1) \\&= 3^3 t(n/8) + 4n (3^2/2^2 + 3/2 + 1)\end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned}t(n) &\cong 3t(n/2) + 4n \\&= 3\left(3t(n/4) + 4n/2\right) + 4n \\&= 3^2 t(n/4) + 4n (3/2 + 1) \\&= 3^2 \left(3t(n/8) + 4n/4\right) + 4n (3/2 + 1) \\&= 3^3 t(n/8) + 4n (3^2/2^2 + 3/2 + 1) \\&\vdots \\&= 3^k t(n/2^k) + 4n (3^{k-1}/2^{k-1} + \cdots + 3^0/2^0)\end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned}t(n) &\cong 3t(n/2) + 4n \\&= 3\left(3t(n/4) + 4n/2\right) + 4n \\&= 3^2 t(n/4) + 4n (3/2 + 1) \\&= 3^2 \left(3t(n/8) + 4n/4\right) + 4n (3/2 + 1) \\&= 3^3 t(n/8) + 4n (3^2/2^2 + 3/2 + 1) \\&\vdots \\&= 3^k t(n/2^k) + 4n (3^{k-1}/2^{k-1} + \cdots + 3^0/2^0) \\&= 3^k t(n/2^k) + 4n \sum_{i=0}^{k-1} 3^i / 2^i\end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned}t(n) &\cong 3t(n/2) + 4n \\&= 3\left(3t(n/4) + 4n/2\right) + 4n \\&= 3^2 t(n/4) + 4n (3/2 + 1) \\&= 3^2 \left(3t(n/8) + 4n/4\right) + 4n (3/2 + 1) \\&= 3^3 t(n/8) + 4n (3^2/2^2 + 3/2 + 1) \\&\vdots \\&= 3^k t(n/2^k) + 4n (3^{k-1}/2^{k-1} + \cdots + 3^0/2^0) \\&= 3^k t(n/2^k) + 4n \sum_{i=0}^{k-1} 3^i / 2^i \\&= 3^k t(n/2^k) + 4n \sum_{i=0}^{k-1} (3/2)^i\end{aligned}$$

# Algoritmo de Karatsuba

---

$$t(n) = \underbrace{3^{\log_2 n} t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i$$

# Algoritmo de Karatsuba

---

$$\begin{aligned} t(n) &= 3^{\log_2 n} \underbrace{t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i \\ &= 3^{\log_2 n} + 4n \frac{1 - (3/2)^{\log_2 n}}{1 - 3/2} \end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned} t(n) &= 3^{\log_2 n} \underbrace{t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i \\ &= 3^{\log_2 n} + 4n \frac{1 - (3/2)^{\log_2 n}}{1 - 3/2} \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / 2^{\log_2 n}) \end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned}t(n) &= 3^{\log_2 n} \underbrace{t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i \\&= 3^{\log_2 n} + 4n \frac{1 - (3/2)^{\log_2 n}}{1 - 3/2} \\&= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / 2^{\log_2 n}) \\&= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / n)\end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned} t(n) &= 3^{\log_2 n} \underbrace{t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i \\ &= 3^{\log_2 n} + 4n \frac{1 - (3/2)^{\log_2 n}}{1 - 3/2} \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / 2^{\log_2 n}) \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / n) \\ &= 3^{\log_2 n} - 8n + 8 3^{\log_2 n} \end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned} t(n) &= 3^{\log_2 n} \underbrace{t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i \\ &= 3^{\log_2 n} + 4n \frac{1 - (3/2)^{\log_2 n}}{1 - 3/2} \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / 2^{\log_2 n}) \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / n) \\ &= 3^{\log_2 n} - 8n + 8 3^{\log_2 n} \\ &= 9 \times 3^{\log_2 n} - 8n \end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned} t(n) &= 3^{\log_2 n} \underbrace{t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i \\ &= 3^{\log_2 n} + 4n \frac{1 - (3/2)^{\log_2 n}}{1 - 3/2} \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / 2^{\log_2 n}) \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / n) \\ &= 3^{\log_2 n} - 8n + 8 \cdot 3^{\log_2 n} \\ &= 9 \times 3^{\log_2 n} - 8n \\ &= 9 \times n^{\frac{\log_2 n}{\log_3 n}} - 8n \end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned} t(n) &= 3^{\log_2 n} \underbrace{t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i \\ &= 3^{\log_2 n} + 4n \frac{1 - (3/2)^{\log_2 n}}{1 - 3/2} \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / 2^{\log_2 n}) \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / n) \\ &= 3^{\log_2 n} - 8n + 8 3^{\log_2 n} \\ &= 9 \times 3^{\log_2 n} - 8n \\ &= 9 \times n^{\frac{\log_2 n}{\log_3 n}} - 8n \\ &\approx 9 \times n^{1.585} - 8n \end{aligned}$$

# Algoritmo de Karatsuba

---

$$\begin{aligned} t(n) &= 3^{\log_2 n} \underbrace{t(1)}_{=O(1)} + 4n \sum_{i=0}^{\log_2 n - 1} (3/2)^i \\ &= 3^{\log_2 n} + 4n \frac{1 - (3/2)^{\log_2 n}}{1 - 3/2} \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / 2^{\log_2 n}) \\ &= 3^{\log_2 n} - 8n (1 - 3^{\log_2 n} / n) \\ &= 3^{\log_2 n} - 8n + 8 \cdot 3^{\log_2 n} \\ &= 9 \times 3^{\log_2 n} - 8n \\ &= 9 \times n^{\frac{\log_2 n}{\log_3 n}} - 8n \\ &\approx 9 \times n^{1.585} - 8n \end{aligned}$$

- La complejidad del algoritmo de Karatsuba es de  $O(n^{1.585})$ .

## Búsqueda binaria

---

- Recordemos la **búsqueda binaria**, que permite determinar si un elemento está en un arreglo ordenado en tiempo  $O(\log n)$ .

## Búsqueda binaria

---

- Recordemos la **búsqueda binaria**, que permite determinar si un elemento está en un arreglo ordenado en tiempo  $O(\log n)$ .
- Este algoritmo “divide” el problema original en solamente **un subproblema** (cuyo tamaño es aproximadamente la mitad del problema original). Por este motivo, se lo puede considerar como un algoritmo de divide and conquer.

## Búsqueda binaria

---

- Recordemos la **búsqueda binaria**, que permite determinar si un elemento está en un arreglo ordenado en tiempo  $O(\log n)$ .
- Este algoritmo “divide” el problema original en solamente **un subproblema** (cuyo tamaño es aproximadamente la mitad del problema original). Por este motivo, se lo puede considerar como un algoritmo de divide and conquer.
- Sin embargo, habitualmente se utiliza el término “divide and conquer” para el caso en el que se divide el problema original en dos o más subproblemas.

## Búsqueda binaria

---

- Recordemos la **búsqueda binaria**, que permite determinar si un elemento está en un arreglo ordenado en tiempo  $O(\log n)$ .
- Este algoritmo “divide” el problema original en solamente **un subproblema** (cuyo tamaño es aproximadamente la mitad del problema original). Por este motivo, se lo puede considerar como un algoritmo de divide and conquer.
- Sin embargo, habitualmente se utiliza el término “divide and conquer” para el caso en el que se divide el problema original en dos o más subproblemas.
- Cuando se divide el problema en un subproblema más chico, se utiliza el término **decrease and conquer**.

## Programación III - Universidad Nacional de General Sarmiento

### Ejercicios: Complejidad computacional

---

1. ¿Qué es la complejidad computacional de un algoritmo? ¿Por qué se mide el peor caso?
2. ¿Cuándo decimos que un algoritmo es polinomial?
3. ¿Cuándo decimos que un algoritmo es exponencial? Qué algoritmos de complejidad exponencial conoce?
4. ¿Cuándo decimos que un problema está bien resuelto?
5. Si tenemos un algoritmo con complejidad  $O(n \log n)$  y otro algoritmo con complejidad  $O(n)$ , cuál es preferible en la práctica? Existe alguna circunstancia que cambie la respuesta a la pregunta anterior?
6. Calcular la complejidad computacional del siguiente algoritmo (y comentar qué hace el algoritmo!):

```
public Map<Integer, Integer> cantidades(ArrayList<Integer> A)
{
    Map<Integer, Integer> ret = new TreeMap<Integer, Integer>();

    for (Integer elem: A)
        ret.put(elem, apariciones(A, elem));

    return ret;
}

private int apariciones(ArrayList<Integer> A, Integer elem)
{
    int ret = 0;
    for (Integer valor: A)
        ret += elem == valor ? 1 : 0;

    return ret;
}
```

7. ¿Se puede dar un algoritmo de menor complejidad para el problema del ejercicio anterior? ¿Cómo depende la complejidad de la implementación particular de Map que se seleccione?

8. Calcular la complejidad computacional del siguiente algoritmo, sabiendo que el método  $proc(x)$  tiene complejidad  $O(n)$ :

```
public void poochie(ArrayList<Integer> A)
{
    int n = A.size() - 1;
    int ret = 0;

    while( n > 0 )
    {
        ret += proc( A.get(n) );
        n = n/2;
    }

    return ret;
}
```

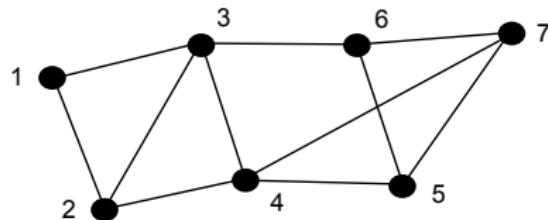
9. ¿Qué es un algoritmo de tipo *divide and conquer*? ¿Qué algoritmos de este tipo conoce? ¿Qué se puede decir de la complejidad computacional de estos algoritmos?
10. ¿Cuál es la complejidad computacional de la búsqueda binaria? ¿Se puede buscar un elemento en un arreglo no ordenado en tiempo menor que  $O(n)$ ?

# Introducción a la teoría de grafos

Programación III - UNGS

# Definiciones

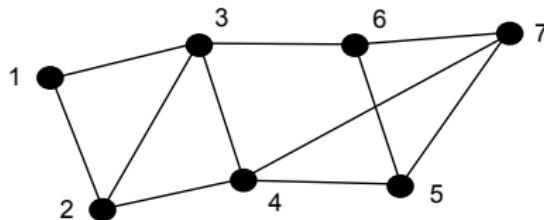
---



- **Definición.** Un **grafo** es un par  $G = (V, E)$  tal que ...

# Definiciones

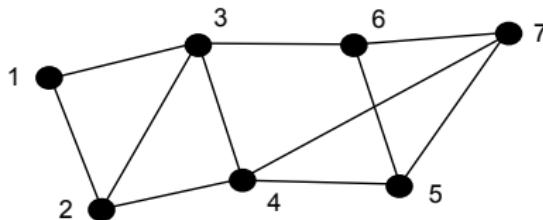
---



- **Definición.** Un **grafo** es un par  $G = (V, E)$  tal que ...
  1.  $V$  es un conjunto finito (llamado el conjunto de **vértices** de  $G$ ), y

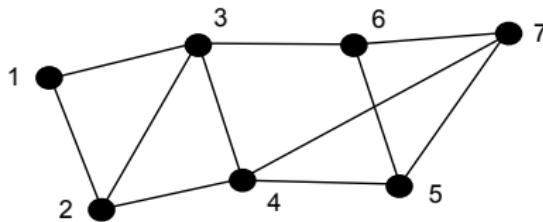
# Definiciones

---



- **Definición.** Un **grafo** es un par  $G = (V, E)$  tal que ...
  1.  $V$  es un conjunto finito (llamado el conjunto de **vértices** de  $G$ ), y
  2.  $E \subseteq \{ij \in V \times V : i \neq j\}$  es un conjunto de **aristas**.

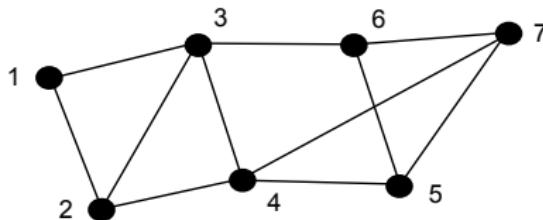
# Definiciones



- **Definición.** Un **grafo** es un par  $G = (V, E)$  tal que ...
  1.  $V$  es un conjunto finito (llamado el conjunto de **vértices** de  $G$ ), y
  2.  $E \subseteq \{ij \in V \times V : i \neq j\}$  es un conjunto de **aristas**.
- En este ejemplo,  $V = \{1, \dots, 7\}$  y  $E = \{12, 13, 23, 24, 34, 36, 45, 47, 56, 57, 67\}$ .

# Definiciones

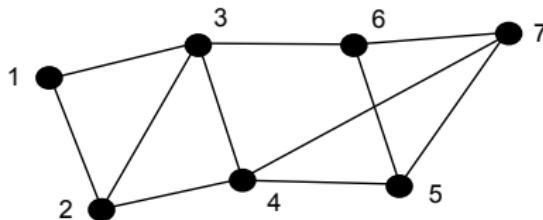
---



- El orden de los **extremos** de una arista no es importante. Nos referimos indistintamente a la arista 34 como 43 o bien  $\{3, 4\}$ .

# Definiciones

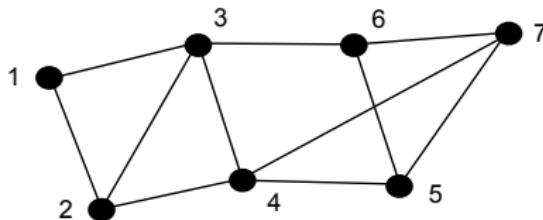
---



- El orden de los **extremos** de una arista no es importante. Nos referimos indistintamente a la arista 34 como 43 o bien  $\{3, 4\}$ .
- **Definiciones:**

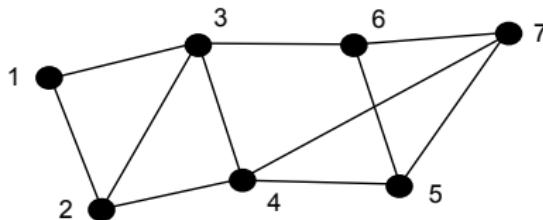
# Definiciones

---



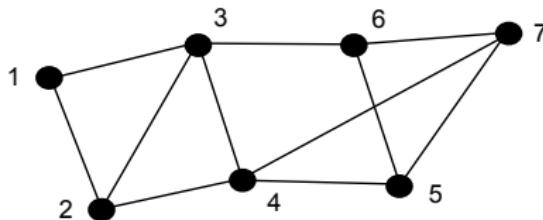
- El orden de los **extremos** de una arista no es importante. Nos referimos indistintamente a la arista 34 como 43 o bien  $\{3, 4\}$ .
- **Definiciones:**
  1. Si  $ij \in E$ , decimos que los vértices  $i$  y  $j$  son **vecinos**.

# Definiciones



- El orden de los **extremos** de una arista no es importante. Nos referimos indistintamente a la arista 34 como 43 o bien  $\{3, 4\}$ .
- **Definiciones:**
  1. Si  $ij \in E$ , decimos que los vértices  $i$  y  $j$  son **vecinos**.
  2. Dado  $i \in V$ , definimos el **vecindario** de  $i$  como
$$N(i) = \{j \in V : ij \in E\}.$$

# Definiciones



- El orden de los **extremos** de una arista no es importante. Nos referimos indistintamente a la arista 34 como 43 o bien  $\{3, 4\}$ .
- **Definiciones:**
  1. Si  $ij \in E$ , decimos que los vértices  $i$  y  $j$  son **vecinos**.
  2. Dado  $i \in V$ , definimos el **vecindario** de  $i$  como  $N(i) = \{j \in V : ij \in E\}$ .
  3. El **grado** de un vértice  $i \in V$  es  $d(i) = |N(i)|$ .

# Motivación

---

- Representamos con grafos relaciones **simétricas** entre entidades.

# Motivación

---

- Representamos con grafos relaciones **simétricas** entre entidades.
  1. Rutas entre un grupo de ciudades (los vértices representan las ciudades).

# Motivación

---

- Representamos con grafos relaciones **simétricas** entre entidades.
  1. Rutas entre un grupo de ciudades (los vértices representan las ciudades).
  2. Mapas de ciudades (los vértices representan las esquinas!).

# Motivación

---

- Representamos con grafos relaciones **simétricas** entre entidades.
  1. Rutas entre un grupo de ciudades (los vértices representan las ciudades).
  2. Mapas de ciudades (los vértices representan las esquinas!).
  3. Relaciones de amistad en una red social (los vértices representan las personas).

# Motivación

---

- Representamos con grafos relaciones **simétricas** entre entidades.
  1. Rutas entre un grupo de ciudades (los vértices representan las ciudades).
  2. Mapas de ciudades (los vértices representan las esquinas!).
  3. Relaciones de amistad en una red social (los vértices representan las personas).
  4. Cursos a dictar y superposiciones entre cursos (dos vértices son vecinos si sus cursos se solapan).

# Motivación

---

- Representamos con grafos relaciones **simétricas** entre entidades.
  1. Rutas entre un grupo de ciudades (los vértices representan las ciudades).
  2. Mapas de ciudades (los vértices representan las esquinas!).
  3. Relaciones de amistad en una red social (los vértices representan las personas).
  4. Cursos a dictar y superposiciones entre cursos (dos vértices son vecinos si sus cursos se solapan).
  5. Antenas en una red de telefonía celular (dos vértices son vecinos si sus antenas tienen áreas de cobertura superpuestas).

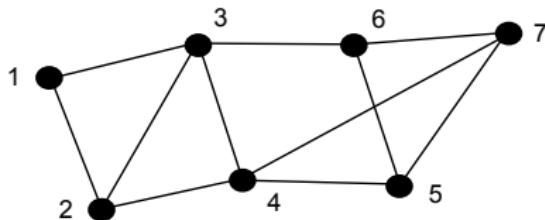
# Motivación

---

- Representamos con grafos relaciones **simétricas** entre entidades.
  1. Rutas entre un grupo de ciudades (los vértices representan las ciudades).
  2. Mapas de ciudades (los vértices representan las esquinas!).
  3. Relaciones de amistad en una red social (los vértices representan las personas).
  4. Cursos a dictar y superposiciones entre cursos (dos vértices son vecinos si sus cursos se solapan).
  5. Antenas en una red de telefonía celular (dos vértices son vecinos si sus antenas tienen áreas de cobertura superpuestas).
  6. Etc.

# Definiciones

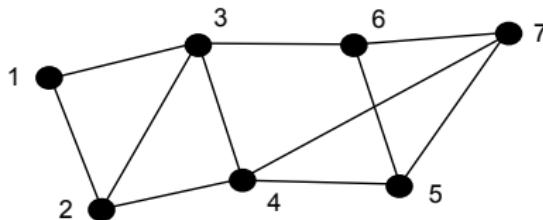
---



- Un **camino** entre dos vértices  $i$  y  $j$  es una secuencia de aristas desde  $i$  hasta  $j$ .

# Definiciones

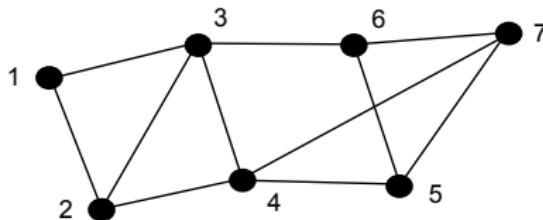
---



- Un **camino** entre dos vértices  $i$  y  $j$  es una secuencia de aristas desde  $i$  hasta  $j$ .
- La **distancia** entre dos vértices es la cantidad de aristas del camino más corto entre ellos.

# Definiciones

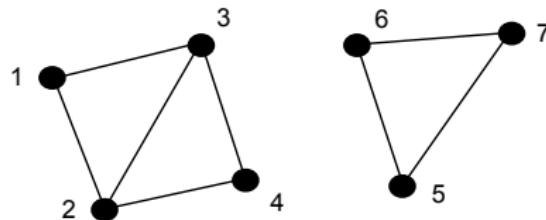
---



- Un **camino** entre dos vértices  $i$  y  $j$  es una secuencia de aristas desde  $i$  hasta  $j$ .
- La **distancia** entre dos vértices es la cantidad de aristas del camino más corto entre ellos.
- Un grafo es **conexo** si existe un camino entre todo par de vértices.

# Definiciones

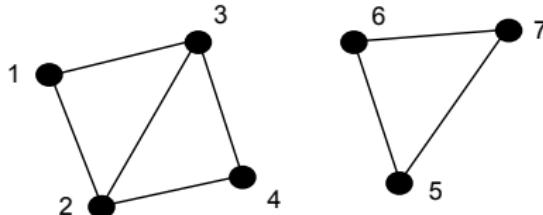
---



- Una **componente conexa** es un subconjunto de vértices conexo, maximal con esta propiedad.

# Definiciones

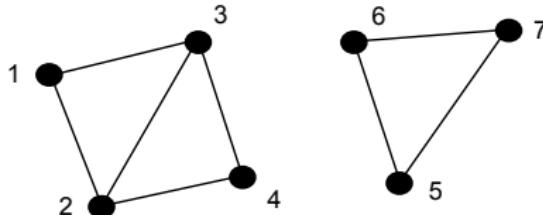
---



- Una **componente conexa** es un subconjunto de vértices conexo, maximal con esta propiedad.
  1. Un grafo conexo tiene exactamente una componente conexa.

# Definiciones

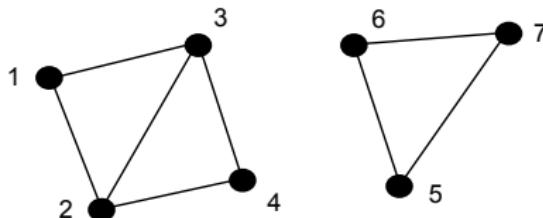
---



- Una **componente conexa** es un subconjunto de vértices conexo, maximal con esta propiedad.
  1. Un grafo conexo tiene exactamente una componente conexa.
  2. Si  $i$  y  $j$  están en componentes conexas distintas, decimos que hay una distancia infinita entre ellos.

# Definiciones

---



- Una **componente conexa** es un subconjunto de vértices conexo, maximal con esta propiedad.
  1. Un grafo conexo tiene exactamente una componente conexa.
  2. Si  $i$  y  $j$  están en componentes conexas distintas, decimos que hay una distancia infinita entre ellos.
- Un **vértice aislado** es un vértice  $i$  con  $d(i) = 0$  (que conforma una componente conexa de tamaño 1).

## Representación de grafos

---

- Nos interesa programar algoritmos que trabajen sobre grafos. Para esto, debemos **representar** adecuadamente grafos en Java (¿cómo lo hacemos?).

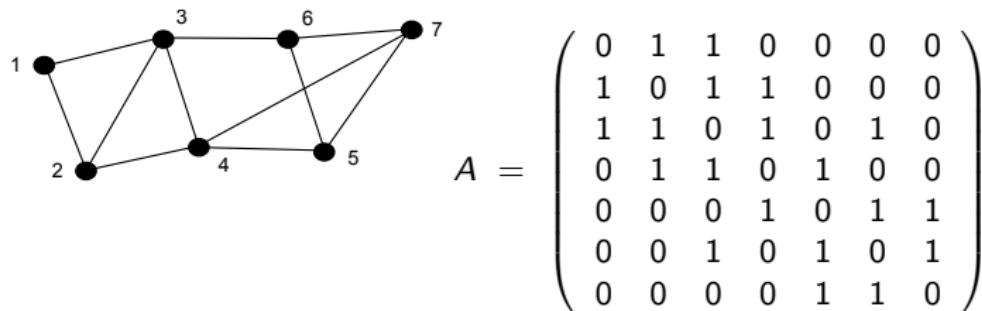
## Representación de grafos

---

- Nos interesa programar algoritmos que trabajen sobre grafos. Para esto, debemos **representar** adecuadamente grafos en Java (**¿cómo lo hacemos?**).
- **Definición.** La **matriz de adyacencia** de un grafo  $G$  es una matriz  $A = (a_{ij}) \in \mathbb{R}^{|V| \times |V|}$  tal que  $a_{ij} = 1$  si  $ij \in E$  y  $a_{ij} = 0$  en caso contrario.

# Representación de grafos

- Nos interesa programar algoritmos que trabajen sobre grafos. Para esto, debemos **representar** adecuadamente grafos en Java (¿cómo lo hacemos?).
- **Definición.** La **matriz de adyacencia** de un grafo  $G$  es una matriz  $A = (a_{ij}) \in \mathbb{R}^{|V| \times |V|}$  tal que  $a_{ij} = 1$  si  $ij \in E$  y  $a_{ij} = 0$  en caso contrario.



# Representación de grafos

---

```
1 class Grafo
2 {
3     private int _vertices;
4     private boolean[][] _adj;
5
6     // Constructor
7     public Grafo(int n)
8     {
9         _vertices = n;
10        _adj = new boolean[n][n];
11    }
12
13    ...
14 }
```

---

# Representación de grafos

---

```
1      ...
2
3  // Getters y setters de aristas
4  public void setArista(int i, int j)
5  {
6      _adj[i][j] = _adj[j][i] = true;
7  }
8  public void deleteArista(int i, int j)
9  {
10     _adj[i][j] = _adj[j][i] = false;
11 }
12 public void isArista(int i, int j)
13 {
14     return _adj[i][j];
15 }
16
17 ...
```

# Representación de grafos

---

- Ventajas de esta representación:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:  $O(1)$ .

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:  $O(1)$ .
- Desventajas de esta representación:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:  $O(1)$ .
- Desventajas de esta representación:
  1. Agregar un vértice:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:  $O(1)$ .
- Desventajas de esta representación:
  1. Agregar un vértice:  $O(n^2)$ .

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:  $O(1)$ .
- Desventajas de esta representación:
  1. Agregar un vértice:  $O(n^2)$ .
  2. Eliminar un vértice:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:  $O(1)$ .
- Desventajas de esta representación:
  1. Agregar un vértice:  $O(n^2)$ .
  2. Eliminar un vértice:  $O(n^2)$ .

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:  $O(1)$ .
- Desventajas de esta representación:
  1. Agregar un vértice:  $O(n^2)$ .
  2. Eliminar un vértice:  $O(n^2)$ .
  3. Obtener todos los vecinos de un vértice:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Agregar una arista:  $O(1)$ .
  2. Eliminar una arista:  $O(1)$ .
  3. Consultar si existe arista:  $O(1)$ .
- Desventajas de esta representación:
  1. Agregar un vértice:  $O(n^2)$ .
  2. Eliminar un vértice:  $O(n^2)$ .
  3. Obtener todos los vecinos de un vértice:  $O(n)$ .

## Representación de grafos

---

- **Definición.** La **matriz de incidencia** de un grafo  $G$  es una matriz  $M = (m_{ij}) \in \mathbb{R}^{|V| \times |E|}$  tal que  $m_{ie} = 1$  si el vértice  $i$  es uno de los extremos de la arista  $e$ , y  $m_{ie} = 0$  en caso contrario.

# Representación de grafos

---

- **Definición.** La **matriz de incidencia** de un grafo  $G$  es una matriz  $M = (m_{ij}) \in \mathbb{R}^{|V| \times |E|}$  tal que  $m_{ie} = 1$  si el vértice  $i$  es uno de los extremos de la arista  $e$ , y  $m_{ie} = 0$  en caso contrario.
- En nuestro ejemplo,  $E = \{12, 13, 23, 24, 34, 36, 45, 47, 56, 57, 67\}$ .

$$M = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

# Representación de grafos

---

- **Definición.** La **matriz de incidencia** de un grafo  $G$  es una matriz  $M = (m_{ij}) \in \mathbb{R}^{|V| \times |E|}$  tal que  $m_{ie} = 1$  si el vértice  $i$  es uno de los extremos de la arista  $e$ , y  $m_{ie} = 0$  en caso contrario.
- En nuestro ejemplo,  $E = \{12, 13, 23, 24, 34, 36, 45, 47, 56, 57, 67\}$ .

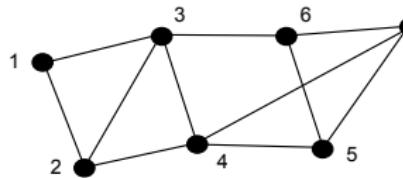
$$M = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

- ¿Es una representación conveniente?

## Representación de grafos

---

- Una tercera opción es por medio de **listas de vecinos** asociadas con cada vértice:



1	→	2, 3
2	→	1, 3, 4
3	→	1, 2, 4, 6
4	→	2, 3, 5, 7
5	→	4, 6, 7
6	→	3, 5, 7
7	→	4, 5, 6

# Representación de grafos

---

```
1 class Grafo
2 {
3     private ArrayList<HashSet<Integer>> _vecinos;
4
5     // Constructor
6     public Grafo(int n)
7     {
8         _vecinos = new ArrayList<HashSet<Integer>>();
9         for(int i=0; i<n; ++i)
10            _vecinos.add(new HashSet<Integer>());
11    }
12
13    ...
14 }
```

---

# Representación de grafos

---

```
1
2     // Getters y setters de aristas
3     public void setArista(int i, int j)
4     {
5         _vecinos.get(i).add(j);
6         _vecinos.get(j).add(i);
7     }
8     public void deleteArista(int i, int j)
9     {
10        _vecinos.get(i).remove(j);
11        _vecinos.get(j).remove(i);
12    }
13    public void isArista(int i, int j)
14    {
15        return _vecinos.get(i).contains(j);
16    }
17
18    14 of.17
```

# Representación de grafos

---

- Ventajas de esta representación:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso,

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.
  4. Eliminar una arista:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.
  4. Eliminar una arista:  $O(1)$  promedio.

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.
  4. Eliminar una arista:  $O(1)$  promedio.
  5. Consultar si existe arista:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.
  4. Eliminar una arista:  $O(1)$  promedio.
  5. Consultar si existe arista:  $O(1)$  promedio.

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.
  4. Eliminar una arista:  $O(1)$  promedio.
  5. Consultar si existe arista:  $O(1)$  promedio.
- Desventajas de esta representación:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.
  4. Eliminar una arista:  $O(1)$  promedio.
  5. Consultar si existe arista:  $O(1)$  promedio.
- Desventajas de esta representación:
  1. Eliminar un vértice:

# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.
  4. Eliminar una arista:  $O(1)$  promedio.
  5. Consultar si existe arista:  $O(1)$  promedio.
- Desventajas de esta representación:
  1. Eliminar un vértice:  $O(n)$ .

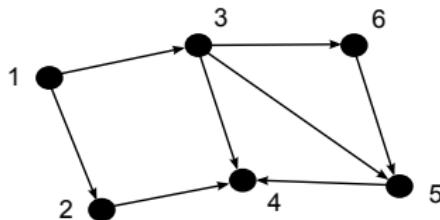
# Representación de grafos

---

- Ventajas de esta representación:
  1. Obtener todos los vecinos de un vértice:  $O(1)$ .
  2. Agregar un vértice:  $O(n)$  en el peor caso, pero  $O(1)$  amortizado.
  3. Agregar una arista:  $O(1)$  promedio.
  4. Eliminar una arista:  $O(1)$  promedio.
  5. Consultar si existe arista:  $O(1)$  promedio.
- Desventajas de esta representación:
  1. Eliminar un vértice:  $O(n)$ .
  - Las operaciones sobre aristas son  $O(\log n)$  en el peor caso si se usan **TreeSets** para representar los vecinos de cada vértice.

# Grafos dirigidos

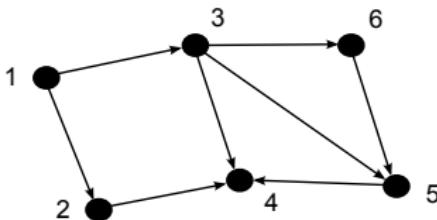
---



- **Definición.** Un **grafo dirigido** (o digrafo) es un par  $D = (N, A)$  tal que ...

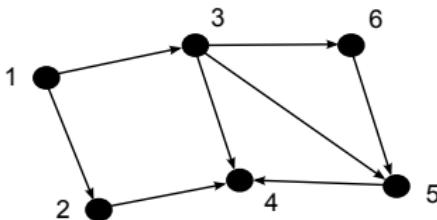
# Grafos dirigidos

---



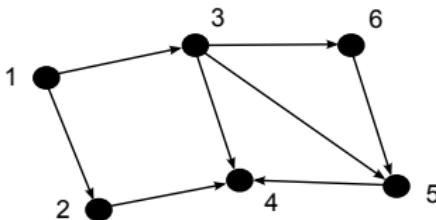
- **Definición.** Un **grafo dirigido** (o digrafo) es un par  $D = (N, A)$  tal que ...
  1.  $N$  es un conjunto finito (llamado el conjunto de **nodos** de  $G$ ), y

# Grafos dirigidos



- **Definición.** Un **grafo dirigido** (o digrafo) es un par  $D = (N, A)$  tal que ...
  1.  $N$  es un conjunto finito (llamado el conjunto de **nodos** de  $G$ ), y
  2.  $A \subseteq \{(i, j) \in V \times V : i \neq j\}$  es un conjunto de pares ordenados, llamados **arcos**.

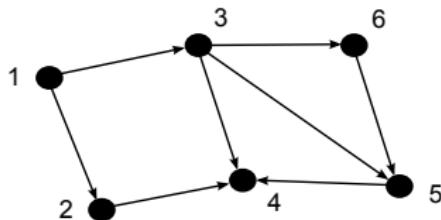
# Grafos dirigidos



- **Definición.** Un **grafo dirigido** (o digrafo) es un par  $D = (N, A)$  tal que ...
  1.  $N$  es un conjunto finito (llamado el conjunto de **nodos** de  $G$ ), y
  2.  $A \subseteq \{(i, j) \in V \times V : i \neq j\}$  es un conjunto de pares ordenados, llamados **arcos**.
- A diferencia de los grafos (no dirigidos), ahora las aristas son **pares ordenados**.

# Definiciones

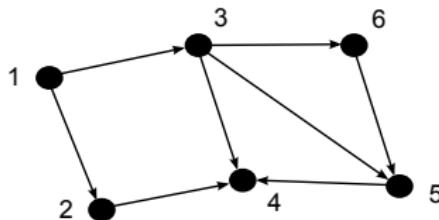
---



- Diferenciamos entre el **grado de entrada** y el **grado de salida** de cada nodo.

# Definiciones

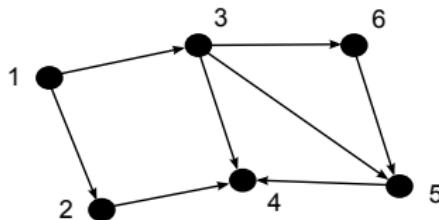
---



- Diferenciamos entre el **grado de entrada** y el **grado de salida** de cada nodo.

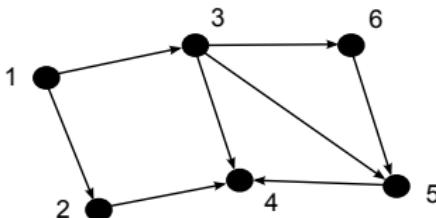
1.  $d^-(i) = |N^-(i)| = |\{j \in N : (j, i) \in A\}|$ .

# Definiciones



- Diferenciamos entre el **grado de entrada** y el **grado de salida** de cada nodo.
1.  $d^-(i) = |N^-(i)| = |\{j \in N : (j, i) \in A\}|$ .
  2.  $d^+(i) = |N^+(i)| = |\{j \in N : (i, j) \in A\}|$ .

# Definiciones



- Diferenciamos entre el **grado de entrada** y el **grado de salida** de cada nodo.
  1.  $d^-(i) = |N^-(i)| = |\{j \in N : (j, i) \in A\}|$ .
  2.  $d^+(i) = |N^+(i)| = |\{j \in N : (i, j) \in A\}|$ .
- Un camino entre dos nodos es una secuencia de arcos entre ellos, respetando el sentido de los arcos.

# Arquitectura en capas

Programación III - UNGS

## Arquitectura en tres capas

---

- Una **arquitectura multi-capa** es una arquitectura de software dividida en niveles, desde la interfaz de usuario hasta el almacenamiento de los datos.

## Arquitectura en tres capas

---

- Una **arquitectura multi-capa** es una arquitectura de software dividida en niveles, desde la interfaz de usuario hasta el almacenamiento de los datos.
- El modelo más habitual es contar con **tres capas** o niveles:

## Arquitectura en tres capas

---

- Una **arquitectura multi-capa** es una arquitectura de software dividida en niveles, desde la interfaz de usuario hasta el almacenamiento de los datos.
- El modelo más habitual es contar con **tres capas** o niveles:
  1. **Nivel de presentación:** Contiene la interfaz de usuario y no realiza procesamiento de información “de negocio”. Sólo realiza el procesamiento necesario para presentar adecuadamente la información.

## Arquitectura en tres capas

---

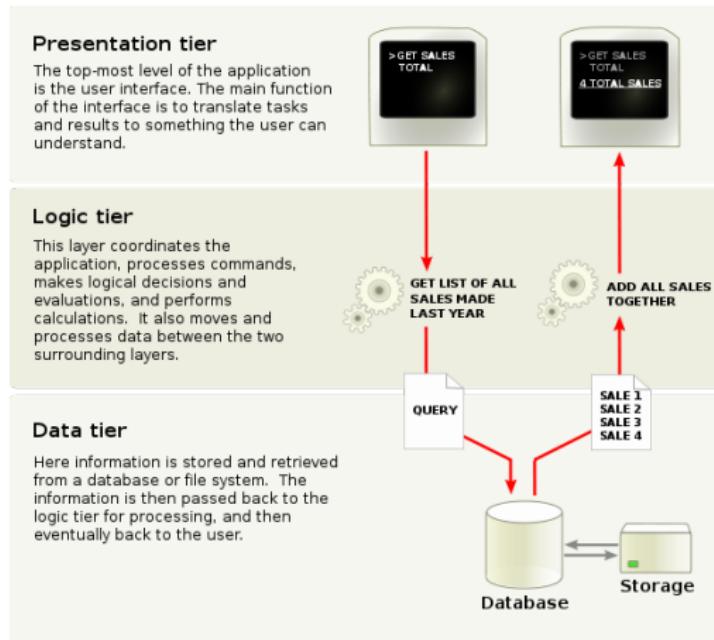
- Una **arquitectura multi-capa** es una arquitectura de software dividida en niveles, desde la interfaz de usuario hasta el almacenamiento de los datos.
- El modelo más habitual es contar con **tres capas** o niveles:
  1. **Nivel de presentación:** Contiene la interfaz de usuario y no realiza procesamiento de información “de negocio”. Sólo realiza el procesamiento necesario para presentar adecuadamente la información.
  2. **Nivel de aplicación o de negocio:** Contiene la funcionalidad del sistema. En un diseño orientado a objetos, incluye las clases de negocio y su implementación.

# Arquitectura en tres capas

---

- Una **arquitectura multi-capa** es una arquitectura de software dividida en niveles, desde la interfaz de usuario hasta el almacenamiento de los datos.
- El modelo más habitual es contar con **tres capas** o niveles:
  1. **Nivel de presentación:** Contiene la interfaz de usuario y no realiza procesamiento de información “de negocio”. Sólo realiza el procesamiento necesario para presentar adecuadamente la información.
  2. **Nivel de aplicación o de negocio:** Contiene la funcionalidad del sistema. En un diseño orientado a objetos, incluye las clases de negocio y su implementación.
  3. **Nivel de datos:** Contiene el almacenamiento de la información y la funcionalidad asociada con el acceso a la información.

# Arquitectura en tres capas



## Arquitectura en tres capas

---

- Contribuye a un diseño modular, con interfaces claras entre las capas.

## Arquitectura en tres capas

---

- Contribuye a un diseño modular, con interfaces claras entre las capas.
- Se puede **reemplazar** cualquiera de los tres niveles por otra implementación, manteniendo los otros dos niveles sin cambios. Los reemplazos habituales se dan en los niveles de presentación y datos.

## Arquitectura en tres capas

---

- Contribuye a un diseño modular, con interfaces claras entre las capas.
- Se puede **reemplazar** cualquiera de los tres niveles por otra implementación, manteniendo los otros dos niveles sin cambios. Los reemplazos habituales se dan en los niveles de presentación y datos.
- Se pueden tener más de tres capas. Por ejemplo, es común tener un **nivel de servicios** entre los niveles de presentación y datos, que es útil cuando hay más de un tipo de interfaz o se proveen servicios a otros procesos.

## Interacción entre las capas

---

- Existen distintas estrategias para implementar la comunicación entre las capas:

## Interacción entre las capas

---

- Existen distintas estrategias para implementar la comunicación entre las capas:
  1. **Interacción top-down:** Cada nivel ejecuta métodos de las clases de los niveles inferiores, y un nivel nunca hace llamadas a los niveles superiores. Esta regla evita las **dependencias circulares**.

## Interacción entre las capas

---

- Existen distintas estrategias para implementar la comunicación entre las capas:
  1. **Interacción top-down:** Cada nivel ejecuta métodos de las clases de los niveles inferiores, y un nivel nunca hace llamadas a los niveles superiores. Esta regla evita las **dependencias circulares**.
  2. **Interacción estricta:** Cada nivel solamente ejecuta métodos de las clases del nivel inmediatamente inferior. Esta regla evita el acoplamiento entre niveles.

# Interacción entre las capas

---

- Existen distintas estrategias para implementar la comunicación entre las capas:
  1. **Interacción top-down:** Cada nivel ejecuta métodos de las clases de los niveles inferiores, y un nivel nunca hace llamadas a los niveles superiores. Esta regla evita las **dependencias circulares**.
  2. **Interacción estricta:** Cada nivel solamente ejecuta métodos de las clases del nivel inmediatamente inferior. Esta regla evita el acoplamiento entre niveles.
  3. **Interacción débil:** Cada nivel puede ejecutar métodos de cualquier otro nivel. Esto mejora la **performance**, pero genera un mayor acoplamiento entre los niveles.

## Funcionalidad transversal

---

- Parte de la funcionalidad del sistema involucra a todas las capas, y es conocida como **funcionalidad transversal** (manejo de logs, manejo de excepciones, validación y autenticación, etc.).

## Funcionalidad transversal

---

- Parte de la funcionalidad del sistema involucra a todas las capas, y es conocida como **funcionalidad transversal** (manejo de logs, manejo de excepciones, validación y autenticación, etc.).
- Es importante detectar qué funcionalidad transversal debemos implementar, y **diseñar módulos separados** para esta funcionalidad, accesibles desde todas las clases.

## Funcionalidad transversal

---

- Parte de la funcionalidad del sistema involucra a todas las capas, y es conocida como **funcionalidad transversal** (manejo de logs, manejo de excepciones, validación y autenticación, etc.).
- Es importante detectar qué funcionalidad transversal debemos implementar, y **diseñar módulos separados** para esta funcionalidad, accesibles desde todas las clases.
  - Por ejemplo, un singleton **Log** con métodos para escribir en el log.

## Funcionalidad transversal

---

- Parte de la funcionalidad del sistema involucra a todas las capas, y es conocida como **funcionalidad transversal** (manejo de logs, manejo de excepciones, validación y autenticación, etc.).
- Es importante detectar qué funcionalidad transversal debemos implementar, y **diseñar módulos separados** para esta funcionalidad, accesibles desde todas las clases.
  - Por ejemplo, un singleton **Log** con métodos para escribir en el log.
- Si no se implementa en módulos separados, se corre el riesgo de duplicar la funcionalidad o entrar en conflicto con las reglas de interacción entre niveles que se hayan fijado.

## Arquitectura en tres capas

---

- Hasta ahora, nuestras implementaciones constan de ...

## Arquitectura en tres capas

---

- Hasta ahora, nuestras implementaciones constan de ...
  1. Una **interfaz de usuario** por consola (típicamente desde la función **main**) o bien con un form desde WindowBuilder.

## Arquitectura en tres capas

---

- Hasta ahora, nuestras implementaciones constan de ...
  1. Una **interfaz de usuario** por consola (típicamente desde la función **main**) o bien con un form desde WindowBuilder.
  2. Un **modelo de objetos** (dado por un diagrama de clases) que representa la lógica del negocio y cuyas funciones son ejecutadas desde la función que implementa la interfaz.

## Arquitectura en tres capas

---

- Hasta ahora, nuestras implementaciones constan de ...
  1. Una **interfaz de usuario** por consola (típicamente desde la función **main**) o bien con un form desde WindowBuilder.
  2. Un **modelo de objetos** (dado por un diagrama de clases) que representa la lógica del negocio y cuyas funciones son ejecutadas desde la función que implementa la interfaz.
  3. Ninguna capacidad de **almacenamiento permanente**, o bien almacenamiento de información en disco.

# Control de versiones

Programación III - UNGS

## Control de versiones

---

- Se llama **control de versiones** (VCS) a las actividades relacionadas con la gestión de cambios en los archivos, configuración y bibliotecas de un producto (en este contexto, de un sistema de software).

## Control de versiones

---

- Se llama **control de versiones** (VCS) a las actividades relacionadas con la gestión de cambios en los archivos, configuración y bibliotecas de un producto (en este contexto, de un sistema de software).
- Habitualmente se realiza con herramientas automatizadas y con servidores centralizados o semi-centralizados:

# Control de versiones

---

- Se llama **control de versiones** (VCS) a las actividades relacionadas con la gestión de cambios en los archivos, configuración y bibliotecas de un producto (en este contexto, de un sistema de software).
- Habitualmente se realiza con herramientas automatizadas y con servidores centralizados o semi-centralizados:
  1. Concurrent Versions System (CVS)
  2. SourceSafe (VSS)
  3. Mercurial
  4. SubVersion (SVN)
  5. Git
  6. ...

# Control de versiones

---

- Síntomas de un mal (o inexistente) control de versiones:
  1. Se destina tiempo a mantener el código actualizado.
  2. Se pierde tiempo buscando la última versión de cada archivo.
  3. Se pierde funcionalidad ya implementada.
  4. Reaparecen bugs ya corregidos.

# Control de versiones

---

- Síntomas de un mal (o inexistente) control de versiones:
  1. Se destina tiempo a mantener el código actualizado.
  2. Se pierde tiempo buscando la última versión de cada archivo.
  3. Se pierde funcionalidad ya implementada.
  4. Reaparecen bugs ya corregidos.
- Beneficios de un buen control de versiones:
  1. Se realiza el control automáticamente.
  2. No hay problema en desarrollar entre varias personas o en varias computadoras.
  3. Se tiene historia de los cambios realizados.
  4. Se pueden revertir los cambios.
  5. Se pueden hacer cambios arriesgados en forma segura.

# SVN

---

- Sistema de control de versiones **centralizado**, con un servidor con un **repositorio** por cada proyecto y un funcionamiento similar a un sistema de archivos.
  1. Se tiene un servidor que centraliza los repositorios.
  2. Cada usuario tiene una **copia local** del repositorio completo, sin la historia previa.

# SVN

---

- **Import:** Subir un proyecto existente al repositorio.
  1. Botón derecho sobre el proyecto, opción *Team → Share project*.
  2. Seleccionar el protocolo SVN.
  3. Especificar la URL del proyecto (proporcionada por el servidor SVN, habitualmente a través de una interfaz web), usuario y clave.
- **Checkout:** Descargar por primera vez un proyecto que ya existe en un repositorio.
  1. En el menú principal, seleccionar la opción *File → Import*.
  2. Seleccionar el tipo de proyecto *SVN → Project from SVN*.
  3. Especificar la URL del proyecto (proporcionada por el servidor SVN, habitualmente a través de una interfaz web), usuario y clave.
  4. Especificar el nombre local con el que se guardará el proyecto.

# SVN

---

- **Update:** Actualizar la versión local descargando los últimos cambios en el repositorio.
  1. Botón derecho sobre el proyecto, opción *Team* → *Update*.
- **Commit:** Registrar en el servidor los últimos cambios realizados.
  1. Botón derecho sobre el proyecto, opción *Team* → *Commit*.
  2. Escribir un comentario y seleccionar los archivos a enviar al repositorio.

# Consejos

---

- Hacer **commits pequeños y puntuales**, con la mayor frecuencia posible.
- Mantener actualizada la copia local del repositorio, para estar sincronizados con el resto del equipo.
- Commitear los **archivos fuente**, nunca los archivos derivados!
- Manejar inmediatamente los **conflictos**.

# Git

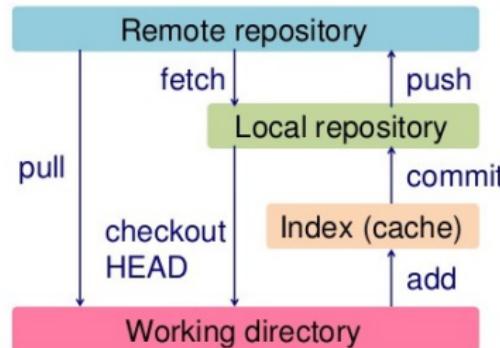
---

- Sistema de control de versiones **distribuido**, orientado a **repositorios** y con énfasis en la eficiencia.
  1. Se tiene un servidor que permite el intercambio de los repositorios entre los usuarios.
  2. Cada usuario tiene una **copia local** del repositorio completo, con toda la historia previa.

# Git

---

- Sistema de control de versiones **distribuido**, orientado a **repositorios** y con énfasis en la eficiencia.
  1. Se tiene un servidor que permite el intercambio de los repositorios entre los usuarios.
  2. Cada usuario tiene una **copia local** del repositorio completo, con toda la historia previa.



# Git

---

- **Clonamos** un repositorio existente en un servidor a nuestro directorio de trabajo:

```
git clone (repositorio) (directorio)
```

# Git

---

- Clonamos un repositorio existente en un servidor a nuestro directorio de trabajo:

```
git clone (repositorio) (directorio)
```

- Creamos un archivo y lo agregamos a la *staging area*:

```
git add (archivo)
```

# Git

---

- Clonamos un repositorio existente en un servidor a nuestro directorio de trabajo:

```
git clone (repositorio) (directorio)
```

- Creamos un archivo y lo agregamos a la *staging area*:

```
git add (archivo)
```

- Hacemos commit de los cambios al repositorio local:

```
git commit
```

# Git

---

- Clonamos un repositorio existente en un servidor a nuestro directorio de trabajo:

```
git clone (repositorio) (directorio)
```

- Creamos un archivo y lo agregamos a la *staging area*:

```
git add (archivo)
```

- Hacemos commit de los cambios al repositorio local:

```
git commit
```

- Enviamos todos los cambios al servidor:

```
git push
```

# Git

---

- Clonamos un repositorio existente en un servidor a nuestro directorio de trabajo:

```
git clone (repositorio) (directorio)
```

- Creamos un archivo y lo agregamos a la *staging area*:

```
git add (archivo)
```

- Hacemos commit de los cambios al repositorio local:

```
git commit
```

- Enviamos todos los cambios al servidor:

```
git push
```

- Traemos todos los cambios del servidor:

```
git pull
```

# Git

---

- **Consultas** sobre el estado actual del repositorio:

`git status`

`git diff`

# Git

---

- **Consultas** sobre el estado actual del repositorio:

`git status`

`git diff`

- Borrar un archivo controlado por Git:

`git rm (archivo)`

# Git

---

- **Consultas** sobre el estado actual del repositorio:

`git status`

`git diff`

- Borrar un archivo controlado por Git:

`git rm (archivo)`

- Mover o renombrar un archivo controlado por Git:

`git mv (archivo) (nuevo)`

## Plugins para eclipse

---

- El acceso a servicios de control de versiones desde eclipse se realiza a través de plugins, accesibles desde la opción “Team” del *package explorer*.
  1. Sin embargo, muchas personas prefieren sincronizar por línea de comandos, para tener más control sobre el procedimiento.

## Plugins para eclipse

---

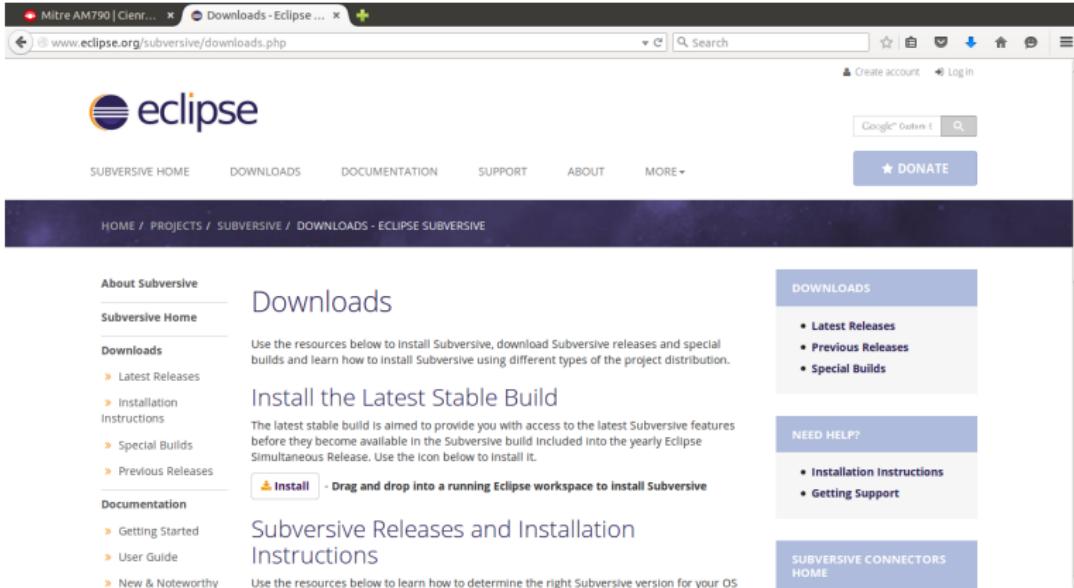
- El acceso a servicios de control de versiones desde eclipse se realiza a través de plugins, accesibles desde la opción “Team” del *package explorer*.
  1. Sin embargo, muchas personas prefieren sincronizar por línea de comandos, para tener más control sobre el procedimiento.
- Las versiones recientes de eclipse (desde 2011) incluyen por defecto plugins para interactuar con Git.

## Plugins para eclipse

---

- El acceso a servicios de control de versiones desde eclipse se realiza a través de plugins, accesibles desde la opción “Team” del *package explorer*.
  1. Sin embargo, muchas personas prefieren sincronizar por línea de comandos, para tener más control sobre el procedimiento.
- Las versiones recientes de eclipse (desde 2011) incluyen por defecto plugins para interactuar con Git.
- Si necesitamos acceso a un servidor SVN, tenemos que instalar los plugins necesarios. A continuación, un tutorial!

# Plugins para eclipse



The screenshot shows a web browser window with three tabs open: "Mitre AM790 | Cien... x", "Downloads - Eclipse ... x", and "www.eclipse.org/subversive/downloads.php" (the active tab). The page content is as follows:

**eclipse**

SUBVERSIVE HOME DOWNLOADS DOCUMENTATION SUPPORT ABOUT MORE ▾

★ DONATE

HOME / PROJECTS / SUBVERSIVE / DOWNLOADS - ECLIPSE SUBVERSIVE

**About Subversive**

- [Subversive Home](#)

**Downloads**

- [Latest Releases](#)
- [Installation Instructions](#)
- [Special Builds](#)
- [Previous Releases](#)

**Documentation**

- [Getting Started](#)
- [User Guide](#)
- [New & Noteworthy](#)

**Downloads**

Use the resources below to Install Subversive, download Subversive releases and special builds and learn how to install Subversive using different types of the project distribution.

**Install** - Drag and drop into a running Eclipse workspace to install Subversive

**Subversive Releases and Installation Instructions**

Use the resources below to learn how to determine the right Subversive version for your OS

**DOWNLOADS**

- Latest Releases
- Previous Releases
- Special Builds

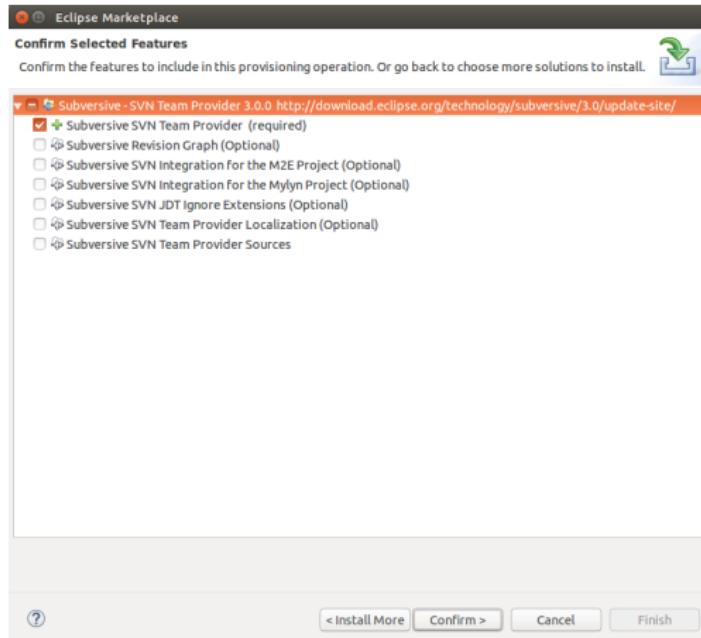
**NEED HELP?**

- Installation Instructions
- Getting Support

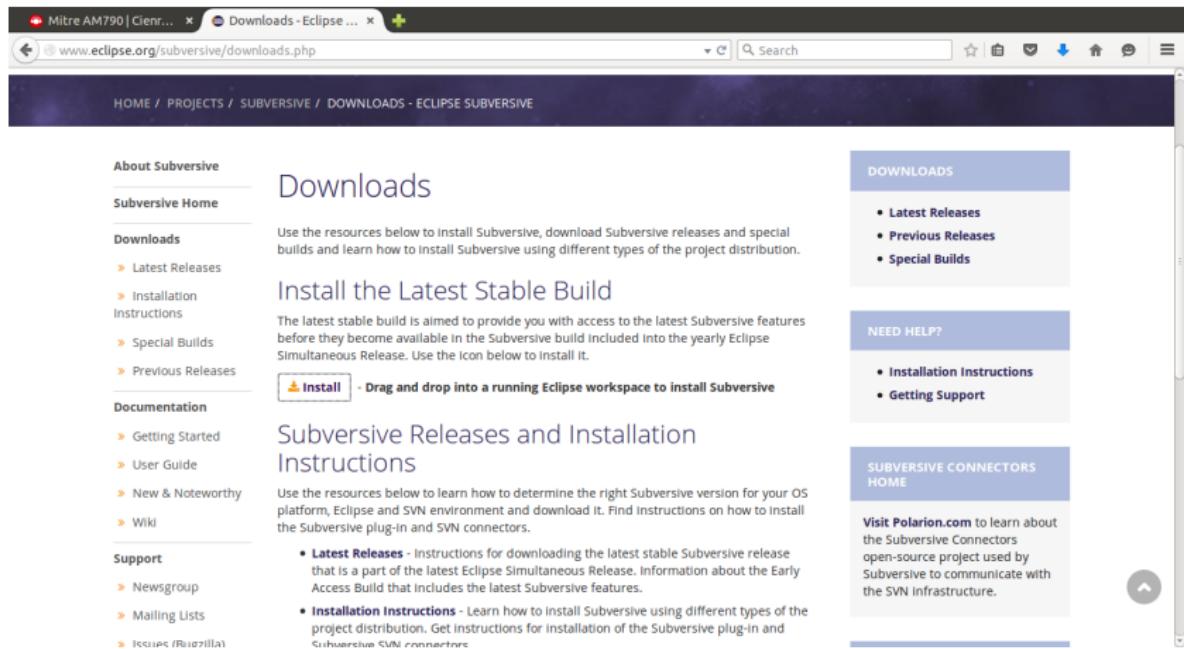
**SUBVERSIVE CONNECTORS HOME**

[www.eclipse.org/subversive](http://www.eclipse.org/subversive)

# Plugins para eclipse



# Plugins para eclipse



The screenshot shows a web browser displaying the 'Downloads - Eclipse Subversive' page from the official Eclipse website. The URL in the address bar is [www.eclipse.org/subversive/downloads.php](http://www.eclipse.org/subversive/downloads.php). The page has a dark header with the title 'Downloads'. On the left, there's a sidebar with links for 'About Subversive', 'Subversive Home', 'Downloads' (which is currently selected), 'Documentation', and 'Support'. The main content area features a large heading 'Downloads' and a sub-section titled 'Install the Latest Stable Build' with an 'Install' button. Below it is another section titled 'Subversive Releases and Installation Instructions' with detailed information and two bulleted lists under 'Latest Releases' and 'Installation Instructions'. To the right, there are three boxes: 'DOWNLOADS' containing links to latest releases, previous releases, and special builds; 'NEED HELP?' containing links to installation instructions and getting support; and 'SUBVERSIVE CONNECTORS HOME' with a link to Polarion.com. The bottom right corner of the page has navigation icons.

Mitre AM790 | Glenr... x Downloads - Eclipse ... x +

www.eclipse.org/subversive/downloads.php

Search

HOME / PROJECTS / SUBVERSIVE / DOWNLOADS - ECLIPSE SUBVERSIVE

About Subversive

Subversive Home

Downloads

- » Latest Releases
- » Installation Instructions
- » Special Builds
- » Previous Releases

Documentation

- » Getting Started
- » User Guide
- » New & Noteworthy
- » Wiki

Support

- » Newsgroup
- » Mailing Lists
- » Eclipse / Mozilla

## Downloads

Use the resources below to install Subversive, download Subversive releases and special builds and learn how to install Subversive using different types of the project distribution.

### Install the Latest Stable Build

The latest stable build is aimed to provide you with access to the latest Subversive features before they become available in the Subversive build included into the yearly Eclipse Simultaneous Release. Use the icon below to install it.

 **Install** · Drag and drop into a running Eclipse workspace to install Subversive

### Subversive Releases and Installation Instructions

Use the resources below to learn how to determine the right Subversive version for your OS platform, Eclipse and SVN environment and download it. Find instructions on how to install the Subversive plug-in and SVN connectors.

- **Latest Releases** - Instructions for downloading the latest stable Subversive release that is a part of the latest Eclipse Simultaneous Release. Information about the Early Access Build that includes the latest Subversive features.
- **Installation Instructions** - Learn how to install Subversive using different types of the project distribution. Get instructions for installation of the Subversive plug-in and Subversive SVN connectors.

### DOWNLOADS

- Latest Releases
- Previous Releases
- Special Builds

### NEED HELP?

- Installation Instructions
- Getting Support

### SUBVERSIVE CONNECTORS HOME

Visit [Polarion.com](#) to learn about the Subversive Connectors open-source project used by Subversive to communicate with the SVN infrastructure.

# Plugins para eclipse

The screenshot shows a web browser window with two tabs: "Mitre AM790 | Glenr..." and "Polarion® : Subversi...". The main content is the Polarion Software website for the Subversive project. The URL is [www.polarion.com/products/svn/subversive.php?utm\\_source=eclipse.org&utm\\_medium=link&utm\\_campaign=link](http://www.polarion.com/products/svn/subversive.php?utm_source=eclipse.org&utm_medium=link&utm_campaign=link). The page title is "Subversive – Subversion Team Provider for Eclipse".

**Polarion Software**  
Requirements Management · Collaborative Test Management · Application Lifecycle Management

**Polarion Solutions**   **Events & Webinars**   **Resources**   **Customer Service**   **Training and Consulting**   **Our Customers**   **Company**   **Try Now**

**New & Noteworthy**

- Subversive Home
- Features
- Screenshots
- New & Noteworthy
- Documentation
- Downloads
- FAQ
- Mailing Lists
- Newsgroup
- Resources
- Bug Tracker »

**Subversive – Subversion Team Provider for Eclipse**

Welcome to the Subversive home page on [www.polarion.com](http://www.polarion.com). Polarion Software sponsors development of Subversive, the Eclipse technology project that aims to provide Subversion (SVN) integration for Eclipse. We have all the information and downloads you need to get started with Subversive – check the menu at left.

142 people like this. [Sign Up](#) to see what your friends like.

**About Subversive Team Provider**

The Subversive plug-in enables you to work with **Subversion** repositories from the Eclipse workbench in almost exactly the same way that has long been possible with CVS repositories via the CVS plug-in bundled in the standard Eclipse distribution. General features of the Subversive plug-in are quite similar to those of the CVS plug-in:

- Browse remote repositories (in this case, Subversion repositories)
- Add a project to the repository and check out projects from the repository
- Synchronize a project to see incoming and outgoing changes
- Commit, update and revert changes
- See resource change history
- Merge changes

Similarity to the CVS plug-in is Subversive's major guiding principle. The goal is to ensure that both plugins use similar concepts, semantics, and interface for similar things. At the same time, Subversive enables you to benefit from all of

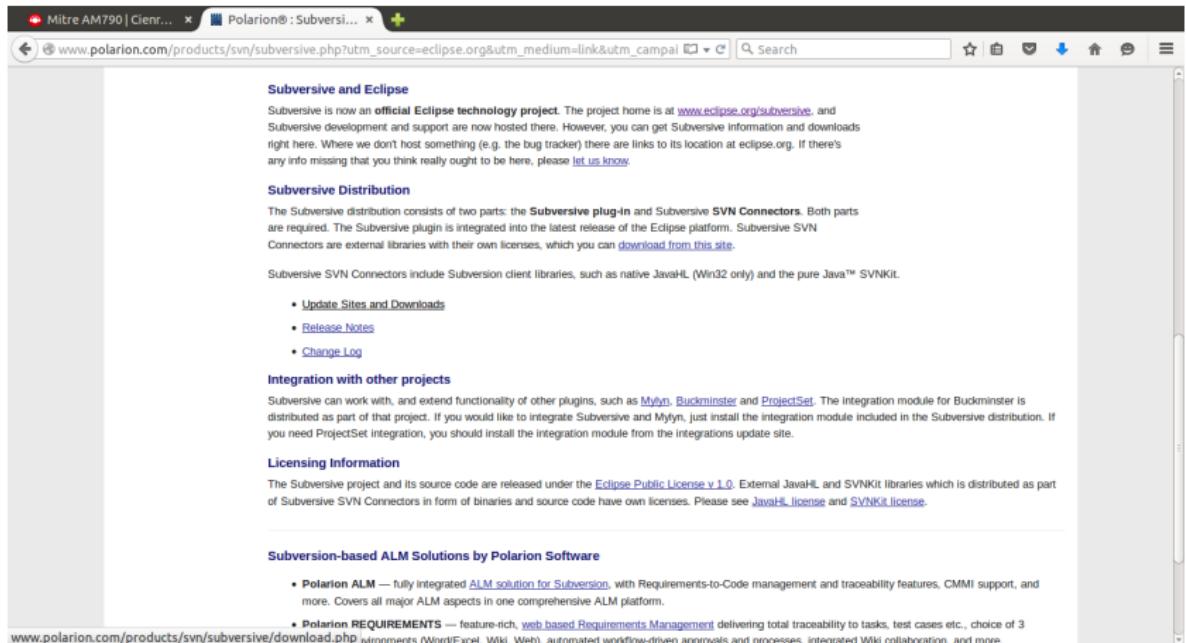
**SUBVERSIVE WORDS**

*"I am in charge of integrating Zend Studio for Eclipse with SVN. After evaluating several alternatives and talking to developers in the Eclipse community, we concluded that Subversive is the most suitable solution for us. I had several interactions with the Subversive team which was very responsive... After deploying the SVN integration within Zend Studio For Eclipse we received many great feedbacks from customers."*

—Yaron Mazor  
Developer, [Zend Technologies Ltd](#)

**DOWNLOAD**

# Plugins para eclipse



**Subversive and Eclipse**  
Subversive is now an **official Eclipse technology project**. The project home is at [www.eclipse.org/subversive](http://www.eclipse.org/subversive), and Subversive development and support are now hosted there. However, you can get Subversive information and downloads right here. Where we don't host something (e.g. the bug tracker) there are links to its location at [eclipse.org](http://eclipse.org). If there's any info missing that you think really ought to be here, please [let us know](#).

**Subversive Distribution**  
The Subversive distribution consists of two parts: the **Subversive plug-in** and **Subversive SVN Connectors**. Both parts are required. The Subversive plugin is integrated into the latest release of the Eclipse platform. Subversive SVN Connectors are external libraries with their own licenses, which you can [download from this site](#).  
  
Subversive SVN Connectors include Subversion client libraries, such as native JavaHL (Win32 only) and the pure Java™ SVNKit.

- [Update Sites and Downloads](#)
- [Release Notes](#)
- [Change Log](#)

**Integration with other projects**  
Subversive can work with, and extend functionality of other plugins, such as [Mylyn](#), [Buckminster](#) and [ProjectSet](#). The integration module for Buckminster is distributed as part of that project. If you would like to integrate Subversive and Mylyn, just install the integration module included in the Subversive distribution. If you need ProjectSet integration, you should install the integration module from the integrations update site.

**Licensing Information**  
The Subversive project and its source code are released under the [Eclipse Public License v 1.0](#). External JavaHL and SVNKit libraries which is distributed as part of Subversive SVN Connectors in form of binaries and source code have own licenses. Please see [JavaHL license](#) and [SVNKit license](#).

**Subversion-based ALM Solutions by Polarion Software**

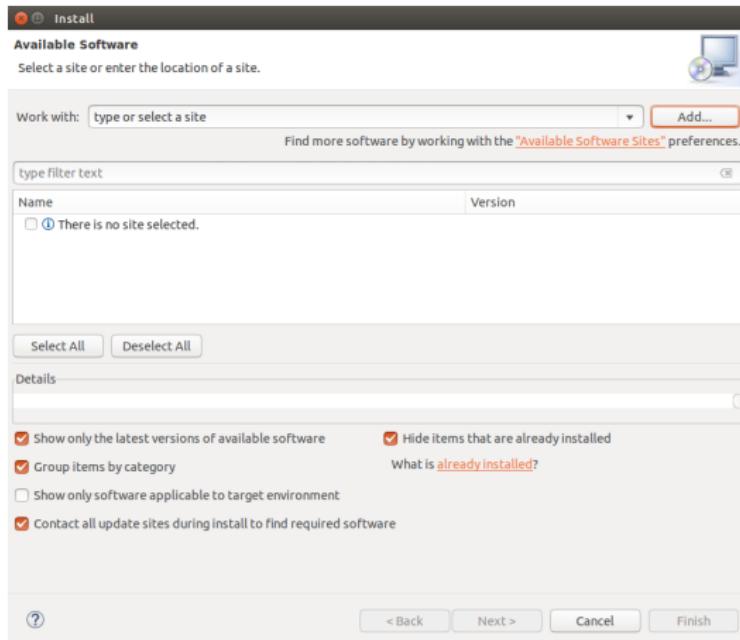
- **Polarion ALM** — fully integrated [ALM solution for Subversion](#), with Requirements-to-Code management and traceability features, CMMI support, and more. Covers all major ALM aspects in one comprehensive ALM platform.
- **Polarion REQUIREMENTS** — feature-rich, [web based Requirements Management](#) delivering total traceability to tasks, test cases etc., choice of 3 environments (Word/Excel, Wiki, Web), automated workflow-driven approvals and processes, integrated Wiki collaboration, and more.

# Plugins para eclipse

The screenshot shows a web browser window with two tabs open: "Mitre AM790 | Glen..." and "Polarion® : Subversi...". The main content area displays the "Subversive Download Page" from [www.polarion.com](http://www.polarion.com/products/svn/subversive/download.php). The page is organized into several sections:

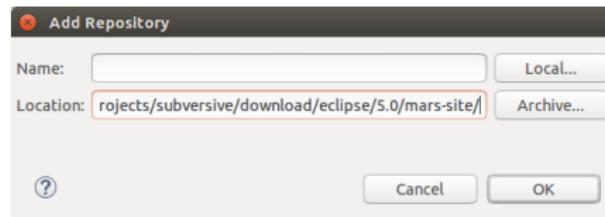
- Mars Release**:
  - A yellow lightbulb icon indicates that The Subversive Update Site is a part of Mars Update Site.
  - To Install:
    - On main menu, choose Help > Install New Software. The available Software dialog appears.
    - In the Work with list, select Mars - <http://download.eclipse.org/releases/mars/>. A list of software packages appears.
    - Expand the Collaboration node, scroll the list and select Subversive features.
    - Check other options in the dialog as desired and click the Next button. The Install Details screen appears in the dialog.
    - Click the Next button, accept the license and click Finish. Subversive will download and install.
  - It is recommended to accept the option to restart Eclipse.
  - Links:
    - <http://community.polarion.com/projects/subversive/download/sites5.0/mars-site/> – [required] Subversive SVN Connectors
    - <http://community.polarion.com/projects/subversive/download/integrations/mars-site/> – [optional] Subversive Integrations
- Latest Stable Build**:
  - <http://download.eclipse.org/technology/subversive/3.0/update-site/> – [required] Subversive plug-in
  - <http://community.polarion.com/projects/subversive/download/integrations5.0/update-site/> – [required] Subversive SVN Connectors
  - <http://community.polarion.com/projects/subversive/download/integrations/update-site/> – [optional] Subversive Integrations
- Subversive Connector Archives**:
  - <http://community.polarion.com/projects/subversive/download/eclipses5.0/builds/> – Subversive Connector Archives for Subversive 3.0
  - <http://community.polarion.com/projects/subversive/download/eclipses4.0/builds/> – Subversive Connector Archives for Subversive 2.0
  - <http://community.polarion.com/projects/subversive/download/eclipses3.0/builds/> – Subversive Connector Archives for Subversive 1.0/1.1
  - <http://community.polarion.com/projects/subversive/download/eclipses2.0/builds/> – Subversive Connector Archives for Subversive 0.7
- Useful Subversive Info from Eclipse**:
  - Here are some links to Subversive information hosted on the [eclipse.org](http://eclipse.org) site. Links open in a pop-up window.
  - To Installation Instructions: [»](#)

# Plugins para eclipse

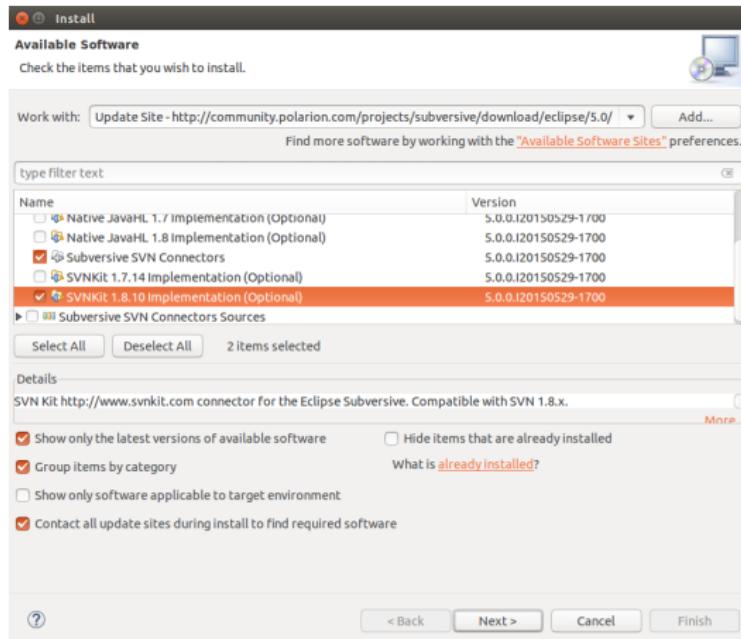


# Plugins para eclipse

---



# Plugins para eclipse



# Técnicas de programación en WindowBuilder

Programación III - UNGS

## Absolute layout

---

- Por defecto, las bibliotecas **Swing** ubican los controles en posiciones predeterminadas (norte, sur, centro, etc.).

## Absolute layout

---

- Por defecto, las bibliotecas **Swing** ubican los controles en posiciones predeterminadas (norte, sur, centro, etc.).
- Para evitar esto, se debe ubicar un **JAbsoluteLayout** como primer componente del frame, para especificar coordenadas absolutas para los controles que se posic和平n a continuaci和平n en el **designer**.

## Look and feels

---

- Se pueden seleccionar distintas apariencias para nuestra aplicación en Java. El conjunto de características que determinan la apariencia de la aplicación se denomina **look and feel**.

## Look and feels

---

- Se pueden seleccionar distintas apariencias para nuestra aplicación en Java. El conjunto de características que determinan la apariencia de la aplicación se denomina **look and feel**.
- Para usar el look and feel del sistema, se deben agregar estas líneas en el constructor del form principal:

## Look and feels

---

- Se pueden seleccionar distintas apariencias para nuestra aplicación en Java. El conjunto de características que determinan la apariencia de la aplicación se denomina **look and feel**.
- Para usar el look and feel del sistema, se deben agregar estas líneas en el constructor del form principal:

1     **try**  
2     {  
3         UIManager.setLookAndFeel(  
4             UIManager.getSystemLookAndFeelClassName());  
5     }  
6     **catch**(Exception e) { ... } // Debe estar en un try/catch

---

## Look and feels

---

- Para usar un look and feel alternativo, es necesario instalar el archivo .jar correspondiente dentro del *build path* del proyecto y cambiar la referencia en el método **setLookAndFeel()**.

## Look and feels

---

- Para usar un look and feel alternativo, es necesario instalar el archivo .jar correspondiente dentro del *build path* del proyecto y cambiar la referencia en el método **setLookAndFeel()**.
- Es importante utilizar un look and feel que se corresponda con las bibliotecas gráficas que estamos usando (en nuestro caso, usamos **Swing**).

## Look and feels

---

- Para usar un look and feel alternativo, es necesario instalar el archivo .jar correspondiente dentro del *build path* del proyecto y cambiar la referencia en el método **setLookAndFeel()**.
- Es importante utilizar un look and feel que se corresponda con las bibliotecas gráficas que estamos usando (en nuestro caso, usamos **Swing**).
- Por ejemplo, para usar el look and feel **JTattoo**, hacemos:

## Look and feels

---

- Para usar un look and feel alternativo, es necesario instalar el archivo .jar correspondiente dentro del *build path* del proyecto y cambiar la referencia en el método **setLookAndFeel()**.
- Es importante utilizar un look and feel que se corresponda con las bibliotecas gráficas que estamos usando (en nuestro caso, usamos **Swing**).
- Por ejemplo, para usar el look and feel **JTattoo**, hacemos:

●

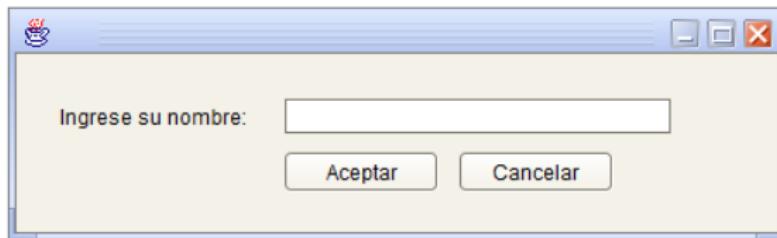
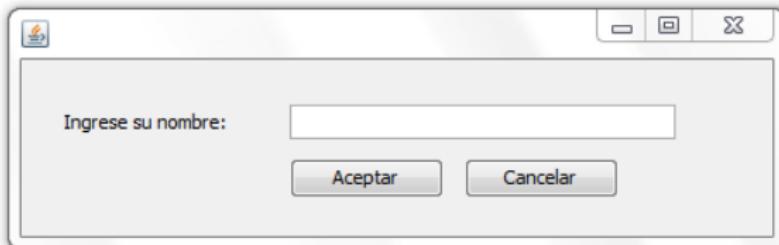
---

```
1 try
2 {
3     UIManager.setLookAndFeel("com.jtattoo.plaf.smart.SmartLookAndFeel");
4 }
5 catch(Exception e) { ... }
```

---

## Look and feels

---



## Text fields

---

- Los **JTextFields** permiten consultar y modificar el texto con **getText()** y **setText()**, respectivamente.

## Text fields

---

- Los **JTextFields** permiten consultar y modificar el texto con **getText()** y **setText()**, respectivamente.
- Es importante recordar que estos dos métodos operan con **Strings**, con lo cual hay que hacer conversiones explícitas hacia/desde los otros tipos de datos:

## Text fields

---

- Los **JTextFields** permiten consultar y modificar el texto con **getText()** y **setText()**, respectivamente.
- Es importante recordar que estos dos métodos operan con **Strings**, con lo cual hay que hacer conversiones explícitas hacia/desde los otros tipos de datos:

---

```
1  String str = textField.getText();
2  int valor = Integer.parseInt(str); // Puede fallar
3
4  valor = valor + 1;
5  textField.setText( Integer.toString(valor) );
```

---

## Combo boxes

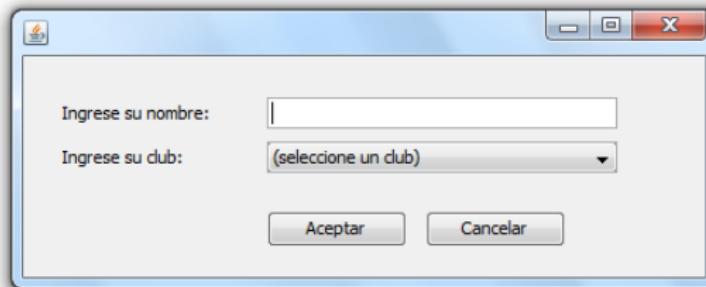
---

- Un **combo box** es adecuado cuando el usuario debe seleccionar entre un conjunto de opciones grande o bien determinado dinámicamente.

## Combo boxes

---

- Un **combo box** es adecuado cuando el usuario debe seleccionar entre un conjunto de opciones grande o bien determinado dinámicamente.
- Se evita que el usuario seleccione una alternativa incorrecta.



## Combo boxes

---

- Para cargar las opciones (*populate*) del combo, se debe generar un nuevo **DefaultComboBoxModel** con un arreglo de strings:

## Combo boxes

---

- Para cargar las opciones (*populate*) del combo, se debe generar un nuevo **DefaultComboBoxModel** con un arreglo de strings:

- ```
1 comboBox.setModel(new DefaultComboBoxModel(new String[])
2 {"selecciona un club"}, "River Plate", "Boca Juniors",
3 "Independiente", "Racing Club", "Otros" }));
```

---

## Combo boxes

---

- Para cargar las opciones (*populate*) del combo, se debe generar un nuevo **DefaultComboBoxModel** con un arreglo de strings:

- ```
1 comboBox.setModel(new DefaultComboBoxModel(new String[])
2 {"seleccione un club", "River Plate", "Boca Juniors",
3 "Independiente", "Racing Club", "Otros"}));
```

---

- El **ComboBoxModel** (y en general, los **models** para los componentes de Swing) es el **modelo de datos** del control, que especifica qué datos se muestran y cómo se seleccionan.

## Combo boxes

---

- Para obtener el ítem seleccionado, se puede consultar por índice o por valor, teniendo en cuenta que en el segundo caso se obtiene un **Object**.

## Combo boxes

---

- Para obtener el ítem seleccionado, se puede consultar por índice o por valor, teniendo en cuenta que en el segundo caso se obtiene un **Object**.

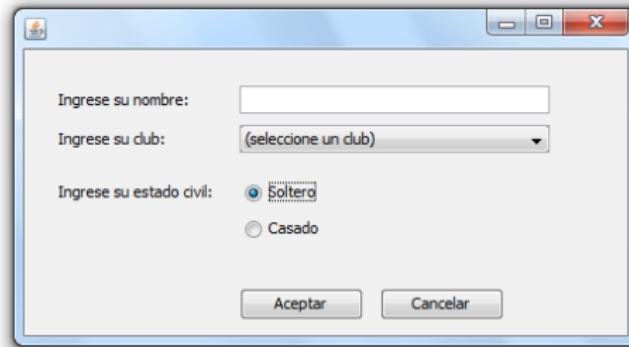
- ```
1 int indice = comboBox.getSelectedIndex();
2 String valor = (String)comboBox.getSelectedItem();
```

---

## Radio buttons

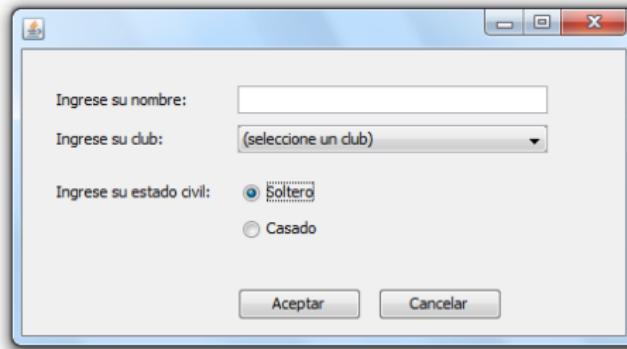
---

- Cuando las opciones son pocas y están prefijadas de antemano, se puede usar un grupo de **JRadioButtons**:



## Radio buttons

- Cuando las opciones son pocas y están prefijadas de antemano, se puede usar un grupo de **JRadioButtons**:



- Se deben ubicar todos los radio buttons dentro de un mismo **ButtonGroup**. Para esto, seleccionar a todos los radio buttons → botón derecho → *Set button group* → *New standard* (o bien seleccionar un **ButtonGroup** existente).

## Radio buttons

---

- Esto genera el siguiente código:

## Radio buttons

---

- Esto genera el siguiente código:

- ---

```
1 private final ButtonGroup buttonGroup = new ButtonGroup();
2 ...
3 JRadioButton rdbtnSoltero = new JRadioButton(" Soltero" );
4 JRadioButton rdbtnCasado = new JRadioButton(" Casado" );
5 buttonGroup.add(rdbtnSoltero);
6 buttonGroup.add(rdbtnCasado);
```

---

## Radio buttons

---

- Esto genera el siguiente código:

- ```
1 private final ButtonGroup buttonGroup = new ButtonGroup();
2 ...
3 JRadioButton rdbtnSoltero = new JRadioButton(" Soltero" );
4 JRadioButton rdbtnCasado = new JRadioButton(" Casado" );
5 buttonGroup.add(rdbtnSoltero);
6 buttonGroup.add(rdbtnCasado);
```

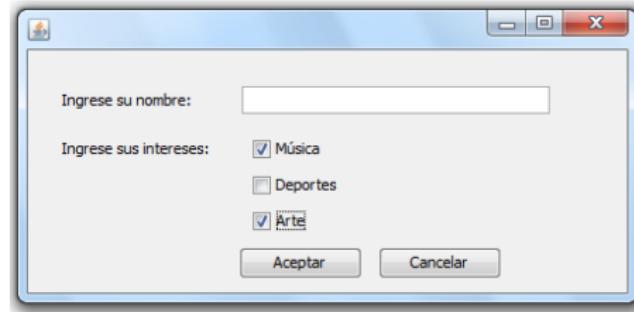
---

- Un **ButtonGroup** controla que haya a lo sumo un radio button seleccionado dentro del grupo. Si no se agrupan los radio buttons de esta forma, puede haber dos de ellos seleccionados al mismo tiempo.

## Check boxes

---

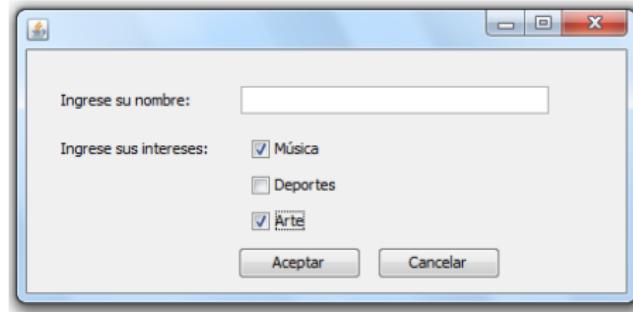
- Cuando las opciones no son excluyentes, se prefieren los **JCheck Boxes**:



## Check boxes

---

- Cuando las opciones no son excluyentes, se prefieren los **JCheck Boxes**:



- Para obtener si un check box está activado, se utiliza el método **boolean isSelected()** sobre el **JCheckBox**.

# Tablas

---

- La visualización de conjuntos de datos se realiza habitualmente por medio de **JTables**:

# Tablas

---

- La visualización de conjuntos de datos se realiza habitualmente por medio de **JTables**:

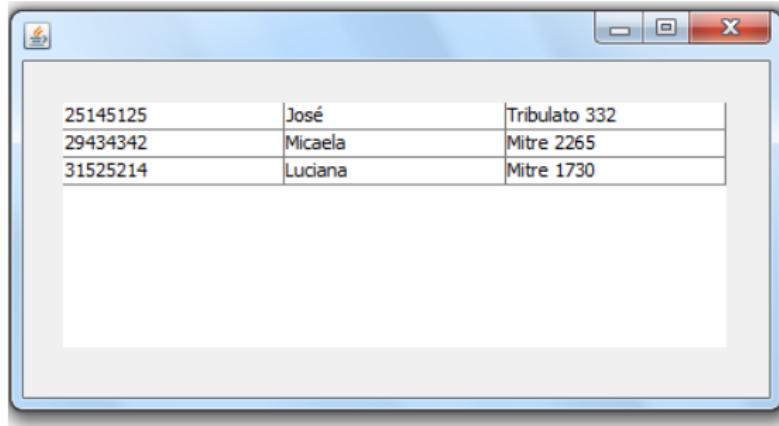
```
1  table = new JTable();
2 ...
3  DefaultTableModel model = new DefaultTableModel();
4  model.addColumn("DNI");
5  model.addColumn("Nombre");
6  model.addColumn("Domicilio");
7
8  model.addRow(new String[] { "25145125" , "José" , "Tribulato 332" });
9  model.addRow(new String[] { "29434342" , "Micaela" , "Mitre 2265" });
10 model.addRow(new String[] { "31525214" , "Luciana" , "Mitre 1730" });
11
12 table.setModel(model);
```

---

# Tablas

---

- Sin embargo, la visualización estándar no muestra las columnas del modelo de datos:



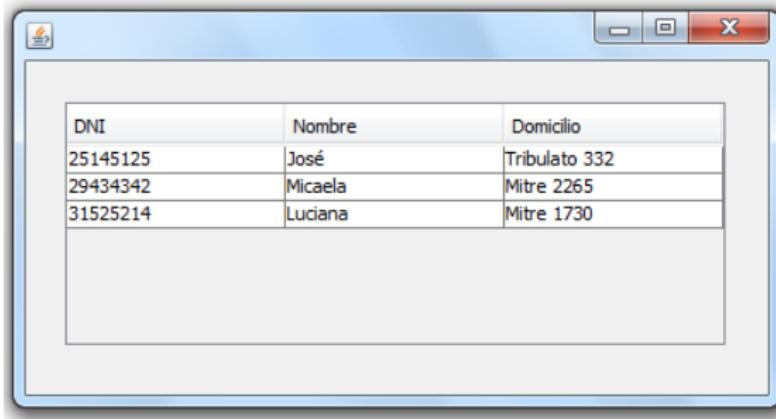
A screenshot of a Windows-style application window titled with a small orange icon. The window contains a 3x3 grid of data, likely representing a table from a database. The columns are labeled with names and addresses, while the rows contain numerical values.

25145125	José	Tribulato 332
29434342	Micaela	Mitre 2265
31525214	Luciana	Mitre 1730

# Tablas

---

- Para ver las columnas se debe ubicar la tabla dentro de un **JScrollPane**, con la opción *sorround with* de eclipse:



A screenshot of a Java Swing application window titled "Eclipse IDE". Inside the window, there is a **JTable** with three columns: "DNI", "Nombre", and "Domicilio". The table contains four rows of data:

DNI	Nombre	Domicilio
25145125	José	Tribulato 332
29434342	Micaela	Mitre 2265
31525214	Luciana	Mitre 1730

# Tablas

---

- Accediendo al modelo de datos de la tabla se pueden consultar y modificar celdas individuales:

# Tablas

---

- Accediendo al modelo de datos de la tabla se pueden consultar y modificar celdas individuales:

- ```
1 TableModel model = table.getModel();
2 String s = (String)model.getValueAt(2,2); // fila y columna
3 model.setValueAt("Nuevo dato", 1, 1);
```

---

# Tablas

---

- Para modificar la visualización de las celdas de la tabla, es necesario crear un **TableCellRenderer** con los atributos personalizados que sean necesarios, y asignarlo a la columna correspondiente.

# Tablas

---

- Para modificar la visualización de las celdas de la tabla, es necesario crear un **TableCellRenderer** con los atributos personalizados que sean necesarios, y asignarlo a la columna correspondiente.
- Por ejemplo, para centrar la primera columna de la tabla:

# Tablas

---

- Para modificar la visualización de las celdas de la tabla, es necesario crear un **TableCellRenderer** con los atributos personalizados que sean necesarios, y asignarlo a la columna correspondiente.
- Por ejemplo, para centrar la primera columna de la tabla:

---

```
1 DefaultTableCellRenderer dtcr = new DefaultTableCellRenderer();
2 dtcr.setHorizontalAlignment( JLabel.CENTER );
3
4 TableColumn col = table.getColumnModel().getColumn(0);
5 col.setCellRenderer(dtcr);
```

---

# Tablas

---

- Por medio del **TableCellRenderer** se controlan todos los aspectos relacionados con la visualización de las celdas.

# Tablas

---

- Por medio del **TableCellRenderer** se controlan todos los aspectos relacionados con la visualización de las celdas.
- Por ejemplo, para cambiar el color:

# Tablas

---

- Por medio del **TableCellRenderer** se controlan todos los aspectos relacionados con la visualización de las celdas.
- Por ejemplo, para cambiar el color:

1      DefaultTableCellRenderer dtcr = new DefaultTableCellRenderer();  
2      dtcr.setHorizontalAlignment( JLabel.CENTER );  
3      dtcr.setBackground( Color.RED );  
4      dtcr.setForeground( Color.WHITE );  
5  
6      TableColumn col = table.getColumnModel().getColumn(0);  
7      col.setCellRenderer(dtcr);

---

# Tablas

---

- El acceso al modelo de las columnas permite modificar otros atributos de las columnas. Por ejemplo, para fijar el ancho de las columnas podemos hacer:

# Tablas

---

- El acceso al modelo de las columnas permite modificar otros atributos de las columnas. Por ejemplo, para fijar el ancho de las columnas podemos hacer:

---

```
1  table.setAutoResizeMode( JTable.AUTO_RESIZE_OFF );
2
3  table.getColumnModel().getColumn(0).setPreferredWidth(40);
4  table.getColumnModel().getColumn(1).setPreferredWidth(150);
5  table.getColumnModel().getColumn(2).setPreferredWidth(200);
```

---

# Tablas

---

- Podemos agregar nuevas filas por código si el modelo de datos es un **DefaultTableModel**.

# Tablas

---

- Podemos agregar nuevas filas por código si el modelo de datos es un **DefaultTableModel**.

- 

```
1 DefaultTableModel tm = (DefaultTableModel) table.getModel();
2 tm.addRow( ... );
```

---

# Tablas

---

- Podemos agregar nuevas filas por código si el modelo de datos es un **DefaultTableModel**.

- ---

```
1 DefaultTableModel tm = (DefaultTableModel) table.getModel();
2 tm.addRow( ... );
```

---

- Si el usuario editó la tabla, accediendo a su modelo de datos se pueden obtener los valores ingresados:

# Tablas

---

- Podemos agregar nuevas filas por código si el modelo de datos es un **DefaultTableModel**.

- ---

```
1 DefaultTableModel tm = (DefaultTableModel) table.getModel();
2 tm.addRow( ... );
```

---

- Si el usuario editó la tabla, accediendo a su modelo de datos se pueden obtener los valores ingresados:

---

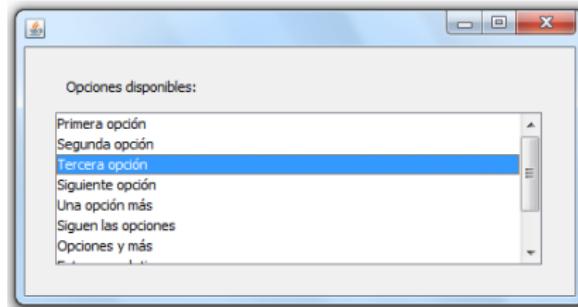
```
1 TableModel tm = table.getModel();
2 System.out.println( tm.getValueAt(2, 2) );
```

---

## Otras opciones para listas de datos

---

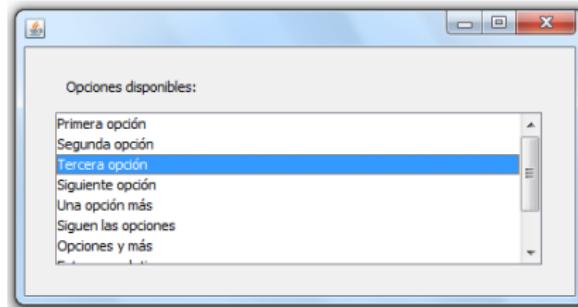
- Las tablas no son siempre la mejor alternativa para visualizar listas de datos. Si hay una única columna y la selección es importante, se puede usar una **JList**:



## Otras opciones para listas de datos

---

- Las tablas no son siempre la mejor alternativa para visualizar listas de datos. Si hay una única columna y la selección es importante, se puede usar una **JList**:



- Para que se muestre la barra de desplazamiento, se debe rodear la lista con un **JScrollPane**.

## Apertura de nuevos forms

---

- Es habitual que se deba **abrir un nuevo form** desde el form principal o desde otros forms.

## Apertura de nuevos forms

---

- Es habitual que se deba **abrir un nuevo form** desde el form principal o desde otros forms.
- Se deben generar nuevos **JFrames** dentro del proyecto, y se crean con el constructor adecuado (que puede recibir parámetros definidos por el programador).

## Apertura de nuevos forms

---

- Es habitual que se deba **abrir un nuevo form** desde el form principal o desde otros forms.
- Se deben generar nuevos **JFrames** dentro del proyecto, y se crean con el constructor adecuado (que puede recibir parámetros definidos por el programador).

●  
1      OtroFrame otro = **new** OtroFrame();  
2      otro.setVisible(**true**);

---

## Apertura de nuevos forms

---

- Es habitual que se deba **abrir un nuevo form** desde el form principal o desde otros forms.
- Se deben generar nuevos **JFrames** dentro del proyecto, y se crean con el constructor adecuado (que puede recibir parámetros definidos por el programador).

●  
1      OtroFrame otro = **new** OtroFrame();  
2      otro.setVisible(**true**);  

---

- La comunicación entre los frames se puede dar por medio de métodos públicos del frame secundario, o bien pasando una referencia al frame llamador.

## Apertura de diálogos

---

- Un **diálogo** (también llamado **form modal**) es una ventana que no permite continuar hasta que no sea cerrada.

## Apertura de diálogos

---

- Un **diálogo** (también llamado **form modal**) es una ventana que no permite continuar hasta que no sea cerrada.
- Se deben generar nuevos **JDialogs** dentro del proyecto, llamando al constructor que recibe el padre y un **boolean** que indica si el diálogo es modal.

# Apertura de diálogos

---

- Un **diálogo** (también llamado **form modal**) es una ventana que no permite continuar hasta que no sea cerrada.
- Se deben generar nuevos **JDialogs** dentro del proyecto, llamando al constructor que recibe el parent y un **boolean** que indica si el diálogo es modal.

---

```
1 public class DialogoTest extends JDialog
2 {
3     public DialogoTest(JFrame parent)
4     {
5         super(parent, true); // true = modal
6         this.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
7         ...
8     }
9 }
```

---

## Cuadros de diálogo

---

- La clase **JOptionPane** permite mostrar **cuadros de diálogo** con formatos pre-especificados.

## Cuadros de diálogo

---

- La clase **JOptionPane** permite mostrar **cuadros de diálogo** con formatos pre-especificados.
- Permite generar mensajes y realizar consultas sencillas, sin agregar una clase por cada tipo de mensaje.

## Cuadros de diálogo

---

- La clase **JOptionPane** permite mostrar **cuadros de diálogo** con formatos pre-especificados.
- Permite generar mensajes y realizar consultas sencillas, sin agregar una clase por cada tipo de mensaje.

1     JOptionPane.showMessageDialog(frame,  
2         "El archivo se escribió correctamente.");  
3  
4     JOptionPane.showMessageDialog(frame,  
5         "El archivo no se escribió correctamente.",  
6         "Problemas", JOptionPane.WARNING\_MESSAGE);  
7  
8     JOptionPane.showMessageDialog(frame, "Error! No se escribió el archivo.",  
9         "Error", JOptionPane.ERROR\_MESSAGE);

---

## Cuadros de diálogo

---

- Se puede especificar una consulta con hasta tres opciones, del siguiente modo:

## Cuadros de diálogo

---

- Se puede especificar una consulta con hasta tres opciones, del siguiente modo:

1     Object[] options = {"Sí, por favor", "No, gracias", "Cancelar!"};  
2  
3     int n = JOptionPane.showOptionDialog(frame,  
4         "Se produjo un problema! Desea continuar?",  
5         "Problemas",  
6         JOptionPane.YES\_NO\_CANCEL\_OPTION,  
7         JOptionPane.QUESTION\_MESSAGE,  
8         **null**, // *Icono*  
9         options, options[2]); // *Valor inicial*

---

## Ventana principal

---

- Es habitual que la ventana principal contenga un **menú principal** y una **barra de herramientas**.

## Ventana principal

---

- Es habitual que la ventana principal contenga un **menú principal** y una **barra de herramientas**.
- Se genera el menú agregando al frame la siguiente jerarquía:

## Ventana principal

---

- Es habitual que la ventana principal contenga un **menú principal** y una **barra de herramientas**.
- Se genera el menú agregando al frame la siguiente jerarquía:
  1. El menú principal está dado por un **JMenuBar**.

## Ventana principal

---

- Es habitual que la ventana principal contenga un **menú principal** y una **barra de herramientas**.
- Se genera el menú agregando al frame la siguiente jerarquía:
  1. El menú principal está dado por un **JMenuBar**.
  2. Cada submenú es un **JMenu**.

## Ventana principal

---

- Es habitual que la ventana principal contenga un **menú principal** y una **barra de herramientas**.
- Se genera el menú agregando al frame la siguiente jerarquía:
  1. El menú principal está dado por un **JMenuBar**.
  2. Cada submenú es un **JMenu**.
  3. Dentro de un submenú, cada opción es un **JMenuItem**.

## Ventana principal

---

- Es habitual que la ventana principal contenga un **menú principal** y una **barra de herramientas**.
- Se genera el menú agregando al frame la siguiente jerarquía:
  1. El menú principal está dado por un **JMenuBar**.
  2. Cada submenú es un **JMenu**.
  3. Dentro de un submenú, cada opción es un **JMenuItem**.
- Un **JMenuItem** se comporta del mismo modo que un **JButton**, y tiene asociado un **ActionListener** con el evento **action → performed**.

## Ventana principal

---

- Las **opciones chequables** dentro de los submenús se generan agregando **JCheckBoxMenuItem**s, y seteando la propiedad **selected**.

## Ventana principal

---

- Las **opciones chequables** dentro de los submenús se generan agregando **JCheckBoxMenuItem**s, y seteando la propiedad **selected**.
- Para generar una barra de herramientas, se debe agregar un **JToolBar** y darle las dimensiones adecuadas.

## Ventana principal

---

- Las **opciones chequables** dentro de los submenús se generan agregando **JCheckBoxMenuItem**s, y seteando la propiedad **selected**.
- Para generar una barra de herramientas, se debe agregar un **JToolBar** y darle las dimensiones adecuadas.
  1. Se agregan los componentes necesarios al **JToolBar**, especialmente **JButtons**.

## Ventana principal

---

- Las **opciones chequables** dentro de los submenús se generan agregando **JCheckBoxMenuItem**s, y seteando la propiedad **selected**.
- Para generar una barra de herramientas, se debe agregar un **JToolBar** y darle las dimensiones adecuadas.
  1. Se agregan los componentes necesarios al **JToolBar**, especialmente **JButtons**.
  2. Si no queremos que el usuario modifique la ubicación de la barra de herramientas, se debe desactivar la propiedad **floatable**.

## Consejos generales

---

- Es una buena práctica agregar **tooltip texts** a los controles, especialmente a los botones de la barra de herramientas.

## Consejos generales

---

- Es una buena práctica agregar **tooltip texts** a los controles, especialmente a los botones de la barra de herramientas.
- Los controles se organizan habitualmente por medio de **JPanels**.

## Consejos generales

---

- Es una buena práctica agregar **tooltip texts** a los controles, especialmente a los botones de la barra de herramientas.
- Los controles se organizan habitualmente por medio de **JPanels**.
  1. La propiedad **border** cuenta con muchas opciones para manejar la visualización de los paneles.

## Consejos generales

---

- Es una buena práctica agregar **tooltip texts** a los controles, especialmente a los botones de la barra de herramientas.
- Los controles se organizan habitualmente por medio de **JPanels**.
  1. La propiedad **border** cuenta con muchas opciones para manejar la visualización de los paneles.
  2. Una opción muy utilizada es usar un **TitledBorder**, que genera un nombre en la parte superior.

## Consejos generales

---

- Es una buena práctica agregar **tooltip texts** a los controles, especialmente a los botones de la barra de herramientas.
- Los controles se organizan habitualmente por medio de **JPanels**.
  1. La propiedad **border** cuenta con muchas opciones para manejar la visualización de los paneles.
  2. Una opción muy utilizada es usar un **TitledBorder**, que genera un nombre en la parte superior.
  3. Cada panel se maneja como un frame y es importante ubicar un **AbsoluteLayout** si queremos organizar los controles por medio de coordenadas absolutas.

**Programación III - Universidad Nacional de General Sarmiento**  
**Ejercicios: Grafos, definiciones elementales**

---

1. ¿Qué es un grafo?
2. ¿Qué es un grafo conexo?
3. ¿Cuál es la complejidad computacional de las siguientes operaciones para la representación de un grafo usando su matriz de adyacencia?
  - Agregar una arista.
  - Eliminar una arista.
  - Consultar si existe una arista.
  - Agregar un vértice.
  - Eliminar un vértice.
  - Obtener todos los vecinos de un vértice.
4. ¿Cuál es la complejidad computacional de las operaciones mencionadas en el punto anterior para la representación de un grafo usando listas de vecinos asociadas con cada vértice?

# Testing unitario – JUnit

Programación III - UNGS

# Testing unitario

---

- Llamamos **testing unitario** (*unit testing*) a las actividades tendientes a testear unidades individuales de código fuente, para determinar si funcionan correctamente.

# Testing unitario

---

- Llamamos **testing unitario** (*unit testing*) a las actividades tendientes a testear unidades individuales de código fuente, para determinar si funcionan correctamente.
  1. Como toda actividad de testing, el testing unitario no puede **garantizar** que no haya errores, pero puede dar cierto grado de seguridad en el código.

# Testing unitario

---

- Llamamos **testing unitario** (*unit testing*) a las actividades tendientes a testear unidades individuales de código fuente, para determinar si funcionan correctamente.
  1. Como toda actividad de testing, el testing unitario no puede **garantizar** que no haya errores, pero puede dar cierto grado de seguridad en el código.
  2. El testing unitario permite encontrar muchos errores antes de que el código se integre al resto del sistema.

# Testing unitario

---

- Llamamos **testing unitario** (*unit testing*) a las actividades tendientes a testear unidades individuales de código fuente, para determinar si funcionan correctamente.
  1. Como toda actividad de testing, el testing unitario no puede **garantizar** que no haya errores, pero puede dar cierto grado de seguridad en el código.
  2. El testing unitario permite encontrar muchos errores antes de que el código se integre al resto del sistema.
- En un lenguaje orientado a objetos, por **unidad individual** nos referimos habitualmente a las clases, pero también los métodos se pueden considerar como unidades individuales.

# Testing unitario

---

- Llamamos **testing unitario** (*unit testing*) a las actividades tendientes a testear unidades individuales de código fuente, para determinar si funcionan correctamente.
  1. Como toda actividad de testing, el testing unitario no puede **garantizar** que no haya errores, pero puede dar cierto grado de seguridad en el código.
  2. El testing unitario permite encontrar muchos errores antes de que el código se integre al resto del sistema.
- En un lenguaje orientado a objetos, por **unidad individual** nos referimos habitualmente a las clases, pero también los métodos se pueden considerar como unidades individuales.
- Los tests unitarios son creados por los programadores, o bien por los **testers de caja blanca**.

# Testing unitario

---

- Beneficios del testing unitario:

# Testing unitario

---

- Beneficios del testing unitario:
  1. Permite **encontrar problemas** en etapas tempranas del desarrollo. Más aún, en un entorno de **test driven development** (TDD), los tests se escriben **antes** de programar el código que testean!

# Testing unitario

---

- Beneficios del testing unitario:

1. Permite **encontrar problemas** en etapas tempranas del desarrollo. Más aún, en un entorno de **test driven development** (TDD), los tests se escriben **antes** de programar el código que testean!
2. Simplifica las actividades de **cambio del código**. Si se realizan cambios en la clase, los tests permiten determinar que el módulo sigue funcionando bien.

# Testing unitario

---

- Beneficios del testing unitario:

1. Permite **encontrar problemas** en etapas tempranas del desarrollo. Más aún, en un entorno de **test driven development** (TDD), los tests se escriben **antes** de programar el código que testean!
2. Simplifica las actividades de **cambio del código**. Si se realizan cambios en la clase, los tests permiten determinar que el módulo sigue funcionando bien.
3. Simplifica la **integración**, en especial si se adopta una metodología de testing *bottom-up*. Sin embargo, el testing unitario no reemplaza al **test de integración**.

# Testing unitario

---

- Beneficios del testing unitario:

1. Permite **encontrar problemas** en etapas tempranas del desarrollo. Más aún, en un entorno de **test driven development** (TDD), los tests se escriben **antes** de programar el código que testean!
2. Simplifica las actividades de **cambio del código**. Si se realizan cambios en la clase, los tests permiten determinar que el módulo sigue funcionando bien.
3. Simplifica la **integración**, en especial si se adopta una metodología de testing *bottom-up*. Sin embargo, el testing unitario no reemplaza al **test de integración**.
4. Proporciona **documentación** de la clase y sus métodos.

# Testing unitario

---

- Limitaciones del testing unitario:

# Testing unitario

---

- Limitaciones del testing unitario:
  1. El testing unitario no permite encontrar errores derivados de la integración entre módulos, dado que cada módulo se testea por separado.

# Testing unitario

---

- Limitaciones del testing unitario:
  1. El testing unitario no permite encontrar errores derivados de la integración entre módulos, dado que cada módulo se testea por separado.
  2. Puede ser complicado/costoso definir tests realistas y suficientemente complejos para detectar problemas no triviales.

# Testing unitario

---

- Limitaciones del testing unitario:
  1. El testing unitario no permite encontrar errores derivados de la integración entre módulos, dado que cada módulo se testea por separado.
  2. Puede ser complicado/costoso definir tests realistas y suficientemente complejos para detectar problemas no triviales.
  3. Requiere de un **control de versiones** riguroso, para mantener registro de los cambios realizados a los tests.

# Testing unitario

---

- Limitaciones del testing unitario:
  1. El testing unitario no permite encontrar errores derivados de la integración entre módulos, dado que cada módulo se testea por separado.
  2. Puede ser complicado/costoso definir tests realistas y suficientemente complejos para detectar problemas no triviales.
  3. Requiere de un **control de versiones** riguroso, para mantener registro de los cambios realizados a los tests.
  4. Requiere de revisión permanente, para evitar que los tests rápidamente queden desactualizados o resulten insuficientes a medida que la funcionalidad del sistema aumenta.

# JUnit

---

- Existen varios paquetes de testing unitario para cada entorno de desarrollo. En Java, el más popular es **JUnit**, desarrollado por Kent Beck, Erich Gamma, David Saff y Mike Clark, de la Universidad de Calgary (Canadá).

# JUnit

---

- Existen varios paquetes de testing unitario para cada entorno de desarrollo. En Java, el más popular es **JUnit**, desarrollado por Kent Beck, Erich Gamma, David Saff y Mike Clark, de la Universidad de Calgary (Canadá).
- Para instalarlo, se debe incluir en el *build path* el archivo **junit.jar** (se realiza automáticamente en eclipse).

# JUnit

---

- Existen varios paquetes de testing unitario para cada entorno de desarrollo. En Java, el más popular es **JUnit**, desarrollado por Kent Beck, Erich Gamma, David Saff y Mike Clark, de la Universidad de Calgary (Canadá).
- Para instalarlo, se debe incluir en el *build path* el archivo **junit.jar** (se realiza automáticamente en eclipse).
- Cada vez que se escribe una clase, por convención se agrega otra clase con el mismo nombre y terminada en “Test”, que contiene los tests unitarios para la clase original.

## Ejemplo

---

- Por ejemplo, tenemos la clase Persona con su nombre y edad:

# Ejemplo

---

- Por ejemplo, tenemos la clase Persona con su nombre y edad:

```
1  class Persona
2  {
3      private String nombre;
4      private int edad;
5
6      public Persona(String n, int e)
7      {
8          nombre = n;
9          edad = e;
10     }
11
12     public void cumplirAnios()
13     {
14         ++edad;
15     }

```

---

# Ejemplo

---

- Escribimos un test unitario para testear cumplirAnios():

# Ejemplo

---

- Escribimos un test unitario para testear cumplirAnios():

---

```
1  class PersonaTest
2  {
3      @Test
4      public void cumplirAniosTest()
5      {
6          Persona p = new Persona("Pepe", 17);
7          p.cumplirAnios();
8          assertEquals(18, p.getEdad());
9      }
10 }
```

---

# Ejemplo

---

- Escribimos un test unitario para testear cumplirAnios():

```
1  class PersonaTest
2  {
3      @Test
4      public void cumplirAniosTest()
5      {
6          Persona p = new Persona("Pepe", 17);
7          p.cumplirAnios();
8          assertEquals(18, p.getEdad());
9      }
10 }
```

---

- Este **caso de test** especifica que una persona creada con 17 años debe tener 18 años luego de ejecutar el método cumplirAnios() sobre ella.

## Ejemplo

---

- Para ejecutar los tests unitarios, dentro de eclipse seleccionamos la acción **Run As** sobre el proyecto y seleccionamos la opción “JUnit Test”.

## Ejemplo

---

- Para ejecutar los tests unitarios, dentro de eclipse seleccionamos la acción **Run As** sobre el proyecto y seleccionamos la opción “JUnit Test”.
- Se ejecutan todos los métodos calificados con `@Test`, y se informa el resultado de cada uno. Si hay tests que fallan, se informa cuáles fallaron y se puede acceder a su código.

## Ejemplo

---

- Para ejecutar los tests unitarios, dentro de eclipse seleccionamos la acción **Run As** sobre el proyecto y seleccionamos la opción “JUnit Test”.
- Se ejecutan todos los métodos calificados con `@Test`, y se informa el resultado de cada uno. Si hay tests que fallan, se informa cuáles fallaron y se puede acceder a su código.
- Es buena práctica ejecutar frecuentemente todos los tests unitarios.

## Ejemplo

---

- Para ejecutar los tests unitarios, dentro de eclipse seleccionamos la acción **Run As** sobre el proyecto y seleccionamos la opción “JUnit Test”.
- Se ejecutan todos los métodos calificados con @Test, y se informa el resultado de cada uno. Si hay tests que fallan, se informa cuáles fallaron y se puede acceder a su código.
- Es buena práctica ejecutar frecuentemente todos los tests unitarios.
  1. Idealmente, los tests unitarios se deben ejecutar completos cada vez que se hace **commit** del código al repositorio. Si hay errores, se debe corregir inmediatamente.

## Ejemplo

---

- Para ejecutar los tests unitarios, dentro de eclipse seleccionamos la acción **Run As** sobre el proyecto y seleccionamos la opción “JUnit Test”.
- Se ejecutan todos los métodos calificados con @Test, y se informa el resultado de cada uno. Si hay tests que fallan, se informa cuáles fallaron y se puede acceder a su código.
- Es buena práctica ejecutar frecuentemente todos los tests unitarios.
  1. Idealmente, los tests unitarios se deben ejecutar completos cada vez que se hace **commit** del código al repositorio. Si hay errores, se debe corregir inmediatamente.
  2. Si esto no es posible, se deben ejecutar con éxito cada vez que se hace **deploy** de una nueva versión.

## Las assertions

---

- Se utilizan **assertions** para testear condiciones que se deben cumplir durante la ejecución de los tests unitarios.

## Las assertions

---

- Se utilizan **assertions** para testear condiciones que se deben cumplir durante la ejecución de los tests unitarios.
  1. **assertEquals(expected, actual)**  
**assertArrayEquals(expected, actual)**  
Requiere que los dos parámetros coincidan.

## Las assertions

---

- Se utilizan **assertions** para testear condiciones que se deben cumplir durante la ejecución de los tests unitarios.
  1. **assertEquals(expected, actual)**  
**assertArrayEquals(expected, actual)**  
Requiere que los dos parámetros coincidan.
  2. **assertTrue(cond)**  
Requiere que la condición sea verdadera.

## Las assertions

---

- Se utilizan **assertions** para testear condiciones que se deben cumplir durante la ejecución de los tests unitarios.
  1. **assertEquals(expected, actual)**  
**assertArrayEquals(expected, actual)**  
Requiere que los dos parámetros coincidan.
  2. **assertTrue(cond)**  
Requiere que la condición sea verdadera.
  3. **assertFalse(cond)**  
Requiere que la condición sea falsa.

## Las assertions

---

- Se utilizan **assertions** para testear condiciones que se deben cumplir durante la ejecución de los tests unitarios.
  1. **assertEquals(expected, actual)**  
**assertArrayEquals(expected, actual)**  
Requiere que los dos parámetros coincidan.
  2. **assertTrue(cond)**  
Requiere que la condición sea verdadera.
  3. **assertFalse(cond)**  
Requiere que la condición sea falsa.
  4. **assertNull(obj)**  
Requiere que la referencia sea **null**.

## Las assertions

---

- Se utilizan **assertions** para testear condiciones que se deben cumplir durante la ejecución de los tests unitarios.

1. **assertEquals(expected, actual)**

**assertArrayEquals(expected, actual)**

Requiere que los dos parámetros coincidan.

2. **assertTrue(cond)**

Requiere que la condición sea verdadera.

3. **assertFalse(cond)**

Requiere que la condición sea falsa.

4. **assertNull(obj)**

Requiere que la referencia sea **null**.

5. **assertNotNull(obj)**

Requiere que la referencia no sea **null**.

## Ejemplo

---

- Test unitarios para una clase que representa una materia, en el contexto de un sistema de registro de inscripciones a materias.

# Ejemplo

---

- Test unitarios para una clase que representa una materia, en el contexto de un sistema de registro de inscripciones a materias.

- 

```
1  class MateriaTest
2  {
3      @Test
4      public void inscribirTest()
5      {
6          Materia m = new Materia("Programación III");
7          Alumno a = new Alumno("José Pérez", "32514521/2011");
8          assertFalse(m.estáInscripto(a));
9          m.inscribir(a);
10         assertTrue(m.estáInscripto(a));
11     }
12     ...
...
```

---

## Ejemplo

---

- Es importante testear los **casos de borde**.

# Ejemplo

---

- Es importante testear los **casos de borde**.
- 

```
1  @Test
2  public void aprobadosTest()
3  {
4      Materia m = new Materia("Programación III");
5      assertEquals(0, m.cantidadAprobados());
6      assertEquals(0, m.promedioAprobados());
7
8      Alumno a = new Alumno("José Pérez", "32514521/2011");
9      assertEquals(0, m.cantidadAprobados());
10
11     m.inscribir(a);
12     m.setNotaFinal(a, 7);
13
14     assertEquals(1, m.cantidadAprobados());
15     assertEquals(7, m.promedioAprobados());
16 }
```

## Las assertions

---

- Se puede escribir código que se ejecuta antes de realizar los tests, por ejemplo para inicializar elementos comunes a los tests.

## Las assertions

---

- Se puede escribir código que se ejecuta antes de realizar los tests, por ejemplo para inicializar elementos comunes a los tests.

---

```
1  class MateriaTest
2  {
3      private List<Alumno> alumnos;
4
5      @Before
6      public void inicializar()
7      {
8          alumnos = new List<Alumno>();
9          alumnos.add(new Alumno("José Pérez", "32514521/2011"));
10         alumnos.add(new Alumno("Josefina López", "34521651/2011"));
11
12     }
13 }
```

---

## Las *annotations*

---

- Se pueden aplicar las siguientes **anotaciones** (*annotations*) a los métodos de la clase de test:

## Las *annotations*

---

- Se pueden aplicar las siguientes **anotaciones** (*annotations*) a los métodos de la clase de test:
  1. **@Test:** Identifica a un método de test.

## Las annotations

---

- Se pueden aplicar las siguientes **anotaciones** (*annotations*) a los métodos de la clase de test:
  1. **@Test:** Identifica a un método de test.
  2. **@Before:** Se ejecuta este método antes de cada test, habitualmente para inicializar los tests o establecer una conexión con la base de datos.

## Las annotations

---

- Se pueden aplicar las siguientes **anotaciones** (*annotations*) a los métodos de la clase de test:
  1. **@Test:** Identifica a un método de test.
  2. **@Before:** Se ejecuta este método antes de cada test, habitualmente para inicializar los tests o establecer una conexión con la base de datos.
  3. **@After:** Se ejecuta este método después de los tests, habitualmente para cerrar conexiones o restaurar los valores originales, si corresponde.

## Las annotations

---

- Se pueden aplicar las siguientes **anotaciones** (*annotations*) a los métodos de la clase de test:
  1. **@Test:** Identifica a un método de test.
  2. **@Before:** Se ejecuta este método antes de cada test, habitualmente para inicializar los tests o establecer una conexión con la base de datos.
  3. **@After:** Se ejecuta este método después de los tests, habitualmente para cerrar conexiones o restaurar los valores originales, si corresponde.
  4. **@Ignore:** No ejecuta el método de test. Se utiliza para omitir tests que pueden llevar demasiado tiempo.

## Las annotations

---

- Se pueden aplicar las siguientes **anotaciones** (*annotations*) a los métodos de la clase de test:
  1. **@Test:** Identifica a un método de test.
  2. **@Before:** Se ejecuta este método antes de cada test, habitualmente para inicializar los tests o establecer una conexión con la base de datos.
  3. **@After:** Se ejecuta este método después de los tests, habitualmente para cerrar conexiones o restaurar los valores originales, si corresponde.
  4. **@Ignore:** No ejecuta el método de test. Se utiliza para omitir tests que pueden llevar demasiado tiempo.
  5. **@Test(expected = Exception.class):** Falla si el método no lanza la excepción especificada.

## Las annotations

---

- Se pueden aplicar las siguientes **anotaciones** (*annotations*) a los métodos de la clase de test:
  1. **@Test:** Identifica a un método de test.
  2. **@Before:** Se ejecuta este método antes de cada test, habitualmente para inicializar los tests o establecer una conexión con la base de datos.
  3. **@After:** Se ejecuta este método después de los tests, habitualmente para cerrar conexiones o restaurar los valores originales, si corresponde.
  4. **@Ignore:** No ejecuta el método de test. Se utiliza para omitir tests que pueden llevar demasiado tiempo.
  5. **@Test(expected = Exception.class):** Falla si el método no lanza la excepción especificada.
  6. **@Test(timeout = 100):** Falla si el método no se ejecuta dentro del límite de tiempo especificado (en milisegundos).

## Consejos

---

1. Visualizar el testing unitario como una herramienta para **manejar los riesgos** asociados con el mantenimiento del software.

## Consejos

---

1. Visualizar el testing unitario como una herramienta para **manejar los riesgos** asociados con el mantenimiento del software.
2. Escribir un caso de test para cada **componente importante**. En particular, se debe tener al menos un método de test para cada método de la clase testeada.

## Consejos

---

1. Visualizar el testing unitario como una herramienta para **manejar los riesgos** asociados con el mantenimiento del software.
2. Escribir un caso de test para cada **componente importante**. En particular, se debe tener al menos un método de test para cada método de la clase testeada.
3. Escribir **tests inteligentes**. No es necesario escribir tests para los getters y setters. En cambio, hay que enfocarse en el comportamiento central de cada componente.

## Consejos

---

1. Visualizar el testing unitario como una herramienta para **manejar los riesgos** asociados con el mantenimiento del software.
2. Escribir un caso de test para cada **componente importante**. En particular, se debe tener al menos un método de test para cada método de la clase testeada.
3. Escribir **tests inteligentes**. No es necesario escribir tests para los getters y setters. En cambio, hay que enfocarse en el comportamiento central de cada componente.
4. **Setear un entorno limpio** para cada test. El énfasis no está puesto en la eficiencia, sino en la efectividad. No es bueno que un test falle porque un test previo dejó datos inconsistentes o inadecuados.

## Consejos

---

1. Visualizar el testing unitario como una herramienta para **manejar los riesgos** asociados con el mantenimiento del software.
2. Escribir un caso de test para cada **componente importante**. En particular, se debe tener al menos un método de test para cada método de la clase testeada.
3. Escribir **tests inteligentes**. No es necesario escribir tests para los getters y setters. En cambio, hay que enfocarse en el comportamiento central de cada componente.
4. **Setear un entorno limpio** para cada test. El énfasis no está puesto en la eficiencia, sino en la efectividad. No es bueno que un test falle porque un test previo dejó datos inconsistentes o inadecuados.
5. Usar **objetos falsos** para no actuar sobre datos importantes.

## Consejos

---

6. Refactorizar los tests cuando se refactoriza el código. Es importante mantener los tests actualizados!

## Consejos

---

6. Refactorizar los tests cuando se refactoriza el código. Es importante mantener los tests actualizados!
7. Escribir tests antes de corregir un bug. Es una forma de asegurarse de que el bug se corrigió, y también permite detectar si el bug reaparece.

## Consejos

---

6. Refactorizar los tests cuando se refactoriza el código. Es importante mantener los tests actualizados!
7. Escribir tests antes de corregir un bug. Es una forma de asegurarse de que el bug se corrigió, y también permite detectar si el bug reaparece.
8. Usar tests unitarios para asegurar que la performance del sistema no empeora.

## Consejos

---

6. Refactorizar los tests cuando se refactoriza el código. Es importante mantener los tests actualizados!
7. Escribir tests antes de corregir un bug. Es una forma de asegurarse de que el bug se corrigió, y también permite detectar si el bug reaparece.
8. Usar tests unitarios para asegurar que la performance del sistema no empeora.
9. Ejecutar los tests continuamente. La situación ideal es que los tests se ejecuten automáticamente, y se envíen avisos a los desarrolladores si se encuentran problemas.

## Consejos

---

6. Refactorizar los tests cuando se refactoriza el código. Es importante mantener los tests actualizados!
7. Escribir tests antes de corregir un bug. Es una forma de asegurarse de que el bug se corrigió, y también permite detectar si el bug reaparece.
8. Usar tests unitarios para asegurar que la performance del sistema no empeora.
9. Ejecutar los tests continuamente. La situación ideal es que los tests se ejecuten automáticamente, y se envíen avisos a los desarrolladores si se encuentran problemas.
10. Divertirse testeando!

# Consejos

---

*When I first encountered unit testing, I was sceptical and thought it was just extra work. But I gave it a chance, because smart people who I trusted told me that it's very useful. Unit testing puts your brain into a state which is very different from coding state. It is challenging to think about what is a simple and correct set of tests for this given component.*

*Once you start writing tests, you'd wonder how you ever got by without them. To make tests even more fun, you can incorporate pair programming. Whether you get together with fellow developers to write tests or write tests for each other's code, fun is guaranteed. At the end of the day, you will be comfortable knowing your system really works because your tests pass.*

*– Alex Iskold, 2008.*

# Recorridos de grafos

Programación III - UNGS

## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.

## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.
- **Idea.** Partir de un vértice  $s \in V$  arbitrario, y obtener todos los vértices que se pueden **alcanzar** a partir de  $s$ .

## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.
- **Idea.** Partir de un vértice  $s \in V$  arbitrario, y obtener todos los vértices que se pueden **alcanzar** a partir de  $s$ .
  1.  $L := \{s\}$ ;

## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.
- **Idea.** Partir de un vértice  $s \in V$  arbitrario, y obtener todos los vértices que se pueden **alcanzar** a partir de  $s$ .
  1.  $L := \{s\}$ ;
  2. Mientras  $L \neq \emptyset$ :

## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.
- **Idea.** Partir de un vértice  $s \in V$  arbitrario, y obtener todos los vértices que se pueden **alcanzar** a partir de  $s$ .
  1.  $L := \{s\}$ ;
  2. Mientras  $L \neq \emptyset$ :
  3.     Seleccionar  $i \in L$  y marcarlo;

## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.
- **Idea.** Partir de un vértice  $s \in V$  arbitrario, y obtener todos los vértices que se pueden **alcanzar** a partir de  $s$ .
  1.  $L := \{s\}$ ;
  2. Mientras  $L \neq \emptyset$ :
    3. Seleccionar  $i \in L$  y marcarlo;
    4. Agregar a  $L$  todos los vecinos no marcados de  $i$ ;

## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.
- **Idea.** Partir de un vértice  $s \in V$  arbitrario, y obtener todos los vértices que se pueden **alcanzar** a partir de  $s$ .
  1.  $L := \{s\}$ ;
  2. Mientras  $L \neq \emptyset$ :
    3. Seleccionar  $i \in L$  y marcarlo;
    4. Agregar a  $L$  todos los vecinos no marcados de  $i$ ;
    5.  $L := L \setminus \{i\}$ ;

## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.
- **Idea.** Partir de un vértice  $s \in V$  arbitrario, y obtener todos los vértices que se pueden **alcanzar** a partir de  $s$ .
  1.  $L := \{s\}$ ;
  2. Mientras  $L \neq \emptyset$ :
    3. Seleccionar  $i \in L$  y marcarlo;
    4. Agregar a  $L$  todos los vecinos no marcados de  $i$ ;
    5.  $L := L \setminus \{i\}$ ;
    6. Fin

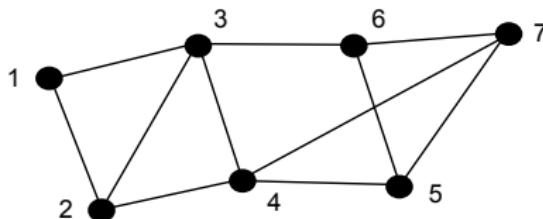
## Recorrido de un grafo

---

- **Problema.** Determinar si un grafo es conexo.
- **Idea.** Partir de un vértice  $s \in V$  arbitrario, y obtener todos los vértices que se pueden **alcanzar** a partir de  $s$ .
  1.  $L := \{s\}$ ;
  2. Mientras  $L \neq \emptyset$ :
    3. Seleccionar  $i \in L$  y marcarlo;
    4. Agregar a  $L$  todos los vecinos no marcados de  $i$ ;
    5.  $L := L \setminus \{i\}$ ;
    6. Fin
    7. Los vértices marcados son los alcanzables desde  $s$ ;

# Definiciones

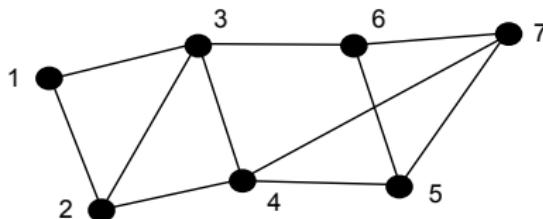
---



- Si la lista  $L$  se implementa con una **cola**, entonces el algoritmo se llama **BFS** (*breadth-first search*), y recorre los vértices en orden de distancia creciente desde  $s$ .

## Definiciones

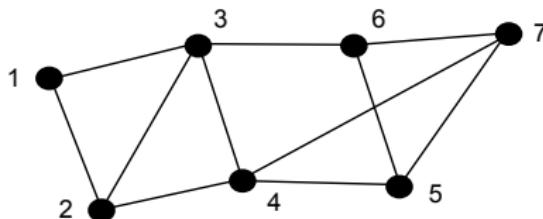
---



- Si la lista  $L$  se implementa con una **cola**, entonces el algoritmo se llama **BFS** (*breadth-first search*), y recorre los vértices en orden de distancia creciente desde  $s$ .
- Si la lista  $L$  se implementa con una **pila**, entonces el algoritmo se llama **DFS** (*depth-first search*), y encuentra primero el vértice más lejano a  $s$ .

## Definiciones

---



- Si la lista  $L$  se implementa con una **cola**, entonces el algoritmo se llama BFS (*breadth-first search*), y recorre los vértices en orden de distancia creciente desde  $s$ .
- Si la lista  $L$  se implementa con una **pila**, entonces el algoritmo se llama DFS (*depth-first search*), y encuentra primero el vértice más lejano a  $s$ .
- Si los vértices marcados son todos los vértices, el grafo es conexo.

**Programación III - Universidad Nacional de General Sarmiento**  
**Ejercicios: Algoritmos sobre grafos**

---

1. Un *triángulo* en un grafo es un conjunto de tres vértices distintos y que además son vecinos dos a dos (alternativamente, un triángulo es una clique de tres vértices).
  - Proponer un algoritmo para determinar la cantidad de triángulos que contiene un grafo.
  - ¿Cuál es la complejidad computacional de este método? ¿Depende esta complejidad de la representación interna del grafo?
  - Implementar el algoritmo propuesto.
2. Dos vértices  $i$  y  $j$  están a *distancia 2* en un grafo  $G = (V, E)$  si existe un vértice  $k$  tal que  $ik \in E$  y  $kj \in E$ .
  - Proponer un algoritmo para determinar si dos vértices están a distancia 2 en un grafo.
  - ¿Cuál es la complejidad computacional del método propuesto? ¿Se puede hacer mejor?
3. Un *vértice universal* es un vértice que es vecino de todos los otros vértices.
  - Escribir un método para determinar si un vértice es universal.
  - ¿Cuál es la complejidad computacional de este método?
  - ¿Se puede implementar este método en  $O(1)$  modificando la representación interna del grafo?

## Programación III - Universidad Nacional de General Sarmiento

### Ejercicios: Testing unitario

---

1. ¿Qué es un test unitario? ¿Qué ventajas tiene la escritura de tests unitarios? ¿Qué limitaciones tiene la técnica de testing unitario?
2. ¿Qué son los “casos de borde”? ¿Por qué es importante tener tests que cubran los casos de borde?
3. ¿Qué *annotations* conoce para los tests unitarios?
4. ¿En qué contexto pueden ser útiles los métodos anotados con @Before y @After?
5. Escribir tests unitarios para el siguiente método:

```
public boolean buscar( int [] b, int x )
{
    int i = 0;
    while( i < b.length && b[ i ] != x )
        ++i;

    if( i < b.length )
        return true;
    else
        return false;
}
```

6. Escribir tests unitarios para el siguiente método, recordando que las excepciones que potencialmente lanza un método forman parte de su interfaz pública:

```
public int segundoMayor( int [] arreglo )
{
    if( arreglo.length < 2 )
        throw new IllegalArgumentException(" Pocos elementos ! ");

    int max = Math.max( arreglo [0] , arreglo [1] );
    int ret = Math.min( arreglo [0] , arreglo [1] );

    for( int i=2; i<arreglo.length; ++i )
    {
        if( arreglo [i] > max )
        {
            ret = max;
            max = arreglo [i];
        }
        else if( arreglo [i] > ret )
            ret = arreglo [i];
    }

    return ret;
}
```

7. Escribir tests unitarios para el siguiente método:

```
public boolean ceroDuplicado(int [] a)
{
    if( a == null )
        throw new IllegalArgumentException();

    int x = 0;
    for( int i=0; i<10; ++i )
        if( a[ i ] == 0 ) ++x;

    return x == 2;
}
```

8. ¿Qué es la metodología de *test driven development*? ¿Qué ventajas y desventajas tiene con relación a las metodologías de desarrollo tradicionales?

# Problema de árbol generador mínimo

Programación III - UNGS

# Árboles

---

- Un **árbol** es un grafo conexo sin circuitos.

# Árboles

---

- Un **árbol** es un grafo conexo sin circuitos.
- **Teorema.** Dado un grafo  $G = (V, X)$ , son equivalentes:
  1.  $G$  es un árbol.
  2.  $G$  es un grafo sin circuitos simples, pero si se agrega cualquier arista  $e$  a  $G$  resulta un grafo con exactamente un circuito simple, y ese circuito contiene a  $e$ .

# Árboles

---

- Un **árbol** es un grafo conexo sin circuitos.
- **Teorema.** Dado un grafo  $G = (V, X)$ , son equivalentes:
  1.  $G$  es un árbol.
  2.  $G$  es un grafo sin circuitos simples, pero si se agrega cualquier arista  $e$  a  $G$  resulta un grafo con exactamente un circuito simple, y ese circuito contiene a  $e$ .
  3. Existe exactamente un camino simple entre todo par de vértices.

# Árboles

---

- Un **árbol** es un grafo conexo sin circuitos.
- **Teorema.** Dado un grafo  $G = (V, X)$ , son equivalentes:
  1.  $G$  es un árbol.
  2.  $G$  es un grafo sin circuitos simples, pero si se agrega cualquier arista  $e$  a  $G$  resulta un grafo con exactamente un circuito simple, y ese circuito contiene a  $e$ .
  3. Existe exactamente un camino simple entre todo par de vértices.
  4.  $G$  es conexo, pero si se quita cualquier arista a  $G$  queda un grafo no conexo.

# Árboles

---

- Una **hoja** de un árbol es un vértice de grado 1.

# Árboles

---

- Una **hoja** de un árbol es un vértice de grado 1.
- **Teorema.** Todo árbol no trivial (es decir, con al menos dos vértices) tiene al menos dos hojas.

# Árboles

---

- Una **hoja** de un árbol es un vértice de grado 1.
- **Teorema.** Todo árbol no trivial (es decir, con al menos dos vértices) tiene al menos dos hojas.
- **Teorema.** Si  $G = (V, E)$  es un árbol, entonces  $|E| = |V| - 1$ .

# Árbol generador mínimo

---

- Dado un grafo  $G$ , un **árbol generador** de  $G$  es un subgrafo de  $G$  que es un árbol y tiene el mismo conjunto de vértices que  $G$ .

## Árbol generador mínimo

---

- Dado un grafo  $G$ , un **árbol generador** de  $G$  es un subgrafo de  $G$  que es un árbol y tiene el mismo conjunto de vértices que  $G$ .
- Sea  $T = (V, E)$  un árbol y  $I : E \rightarrow R$  una función que asigna longitudes (o pesos) a las aristas de  $T$ . Se define la **longitud** de  $T$  como  $I(T) = \sum_{e \in T} I(e)$ .

# Árbol generador mínimo

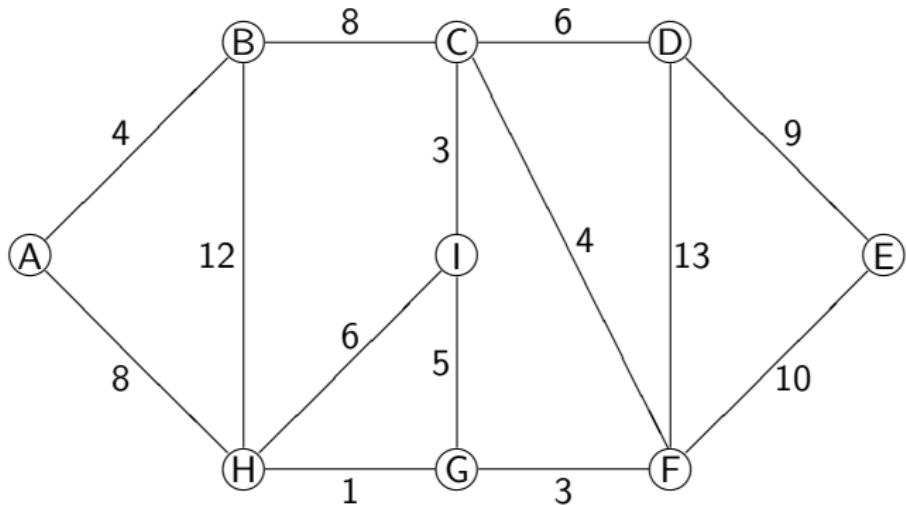
---

- Dado un grafo  $G$ , un **árbol generador** de  $G$  es un subgrafo de  $G$  que es un árbol y tiene el mismo conjunto de vértices que  $G$ .
- Sea  $T = (V, E)$  un árbol y  $I : E \rightarrow R$  una función que asigna longitudes (o pesos) a las aristas de  $T$ . Se define la **longitud** de  $T$  como  $I(T) = \sum_{e \in T} I(e)$ .
- Dado un grafo  $G = (V, E)$  un **árbol generador mínimo**  $T$  de  $G$  es un árbol generador de  $G$  de mínima longitud, es decir

$$I(T) \leq I(T') \quad \forall T' \text{ árbol generador de } G.$$

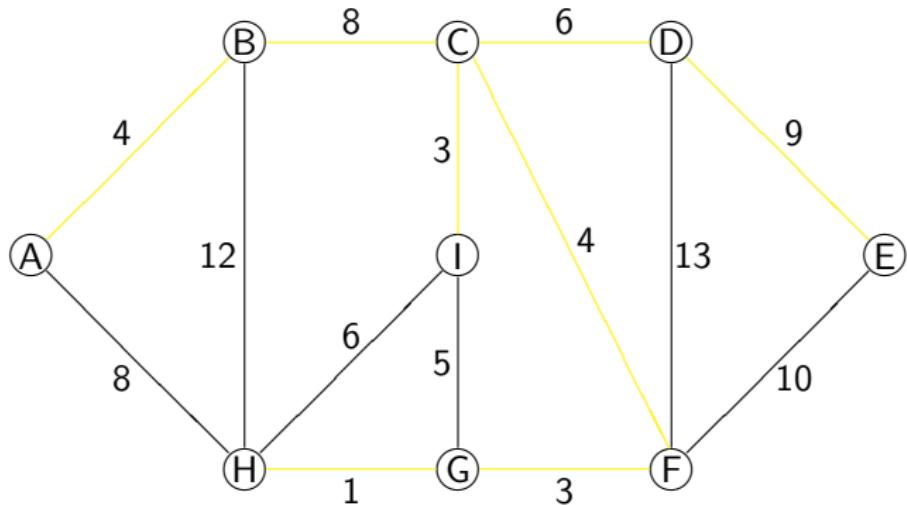
## Ejemplo

---



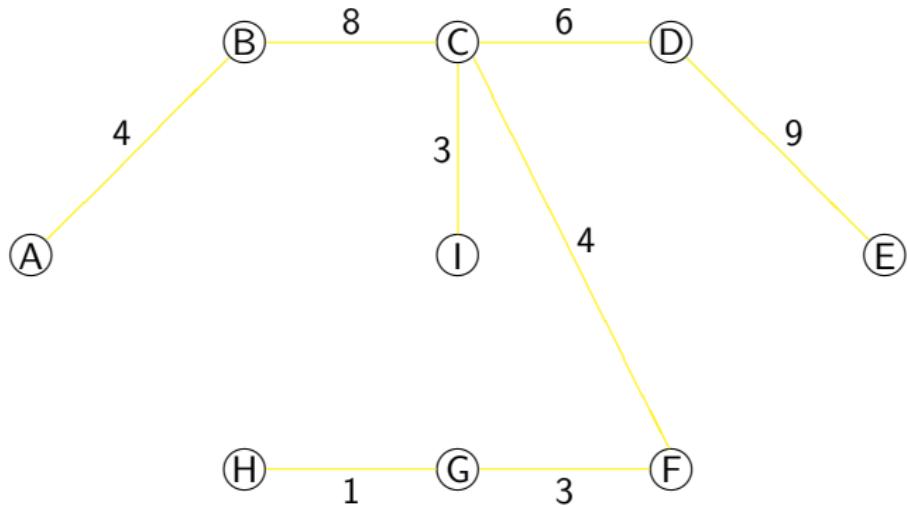
## Ejemplo

---



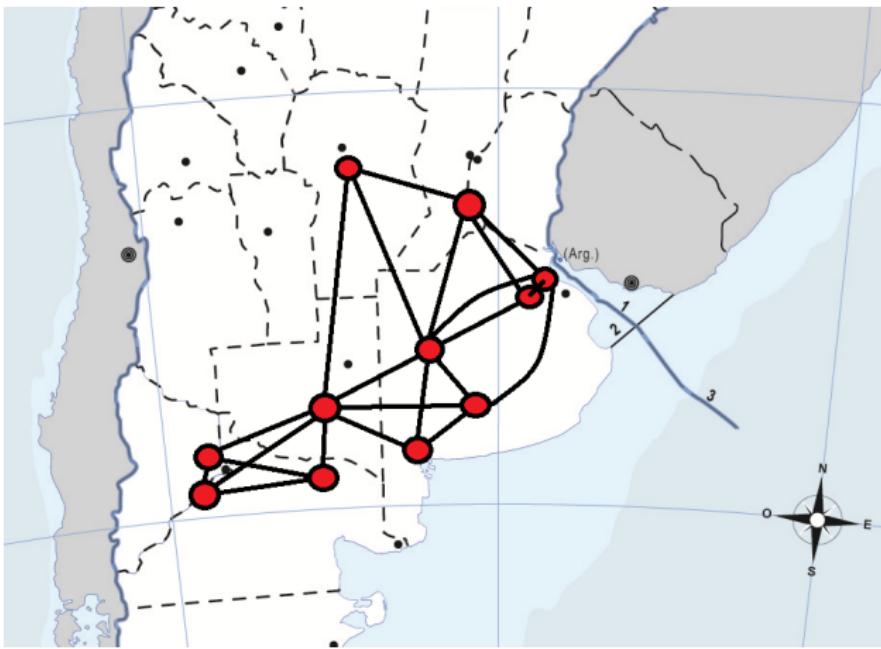
## Ejemplo

---



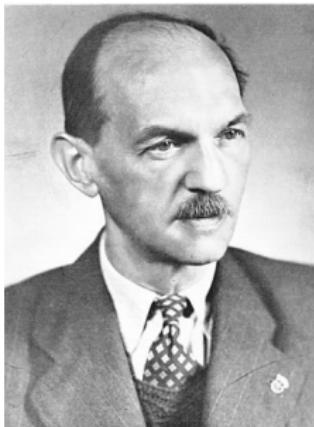
## Aplicación: Diseño de redes eléctricas

---



# Algoritmo de Prim

---



Vojtech Jarnik  
(1897–1970)



Robert Prim  
(1921)



Edsger Dijkstra  
(1930–2002)

## Algoritmo de Prim

---

$V_T := \{u\}$  ( $u$  cualquier vértice de  $G$ )

$E_T := \emptyset$

$i := 1$

**mientras**  $i \leq n - 1$  **hacer**

elegir  $e = (u, v) \in E$  tal que  $l(e)$  sea mínima  
entre las aristas que tienen un extremo

$u \in V_T$  y el otro  $v \in V \setminus V_T$

$E_T := E_T \cup \{e\}$

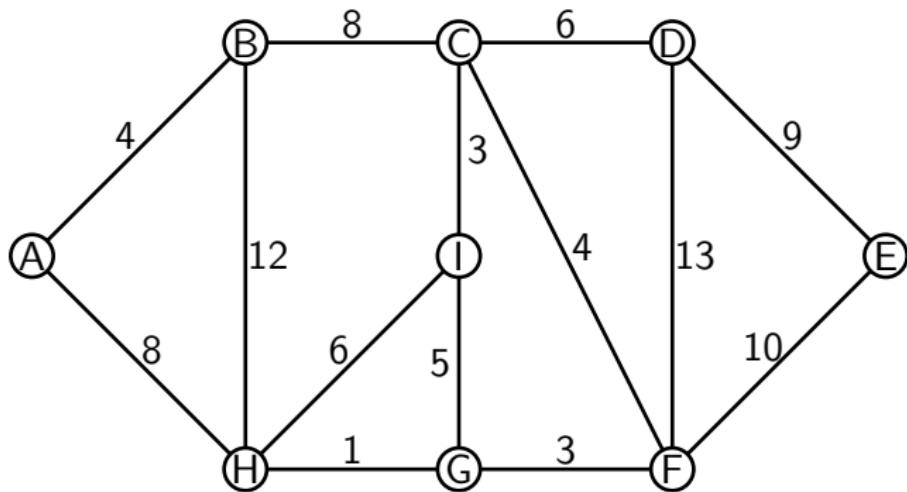
$V_T := V_T \cup \{v\}$

$i := i + 1$

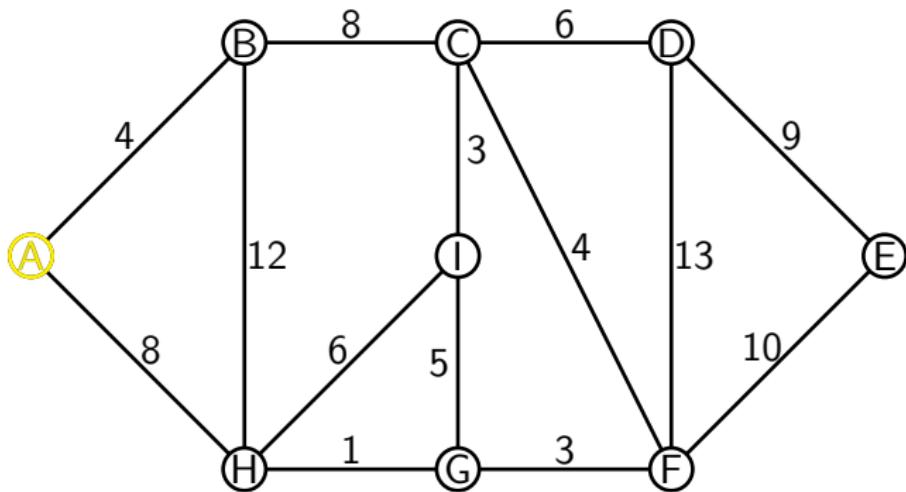
**retornar**  $T = (V_T, E_T)$

## Ejemplo

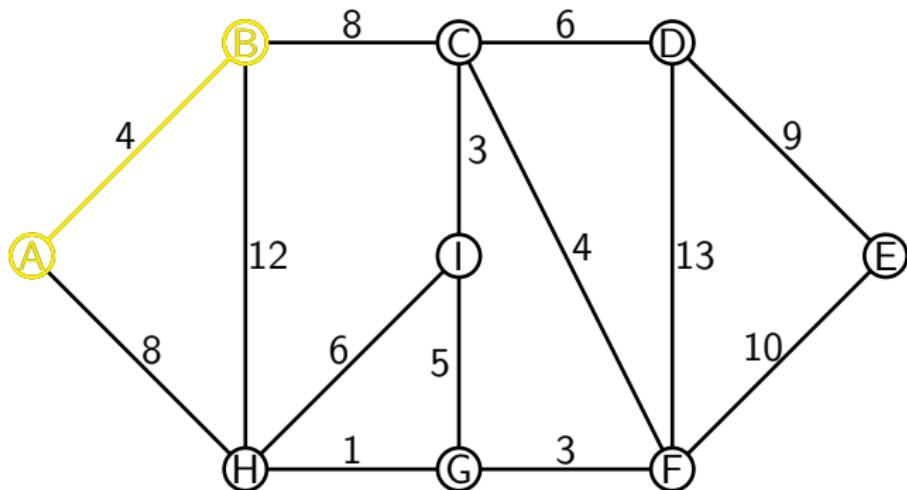
---



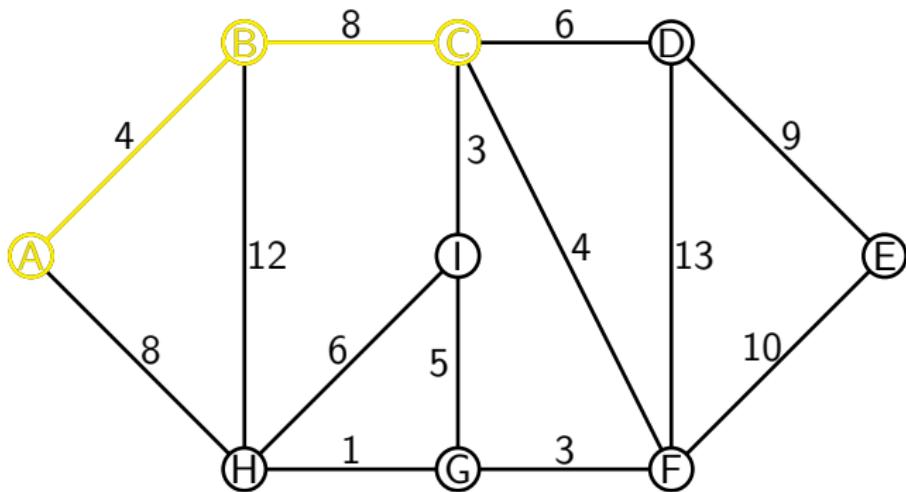
## Ejemplo



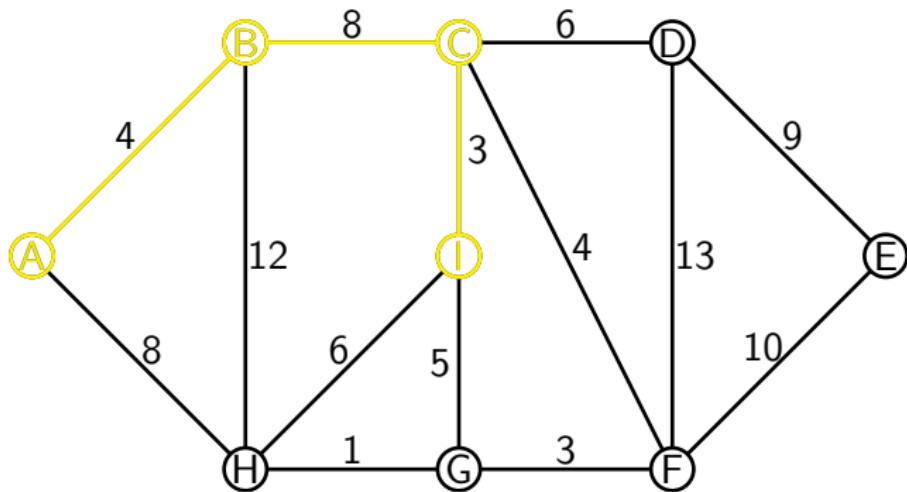
## Ejemplo



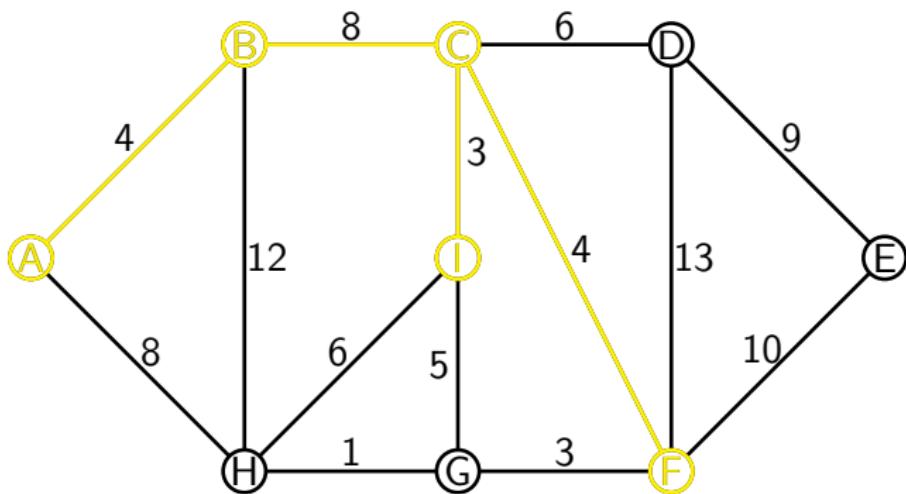
## Ejemplo



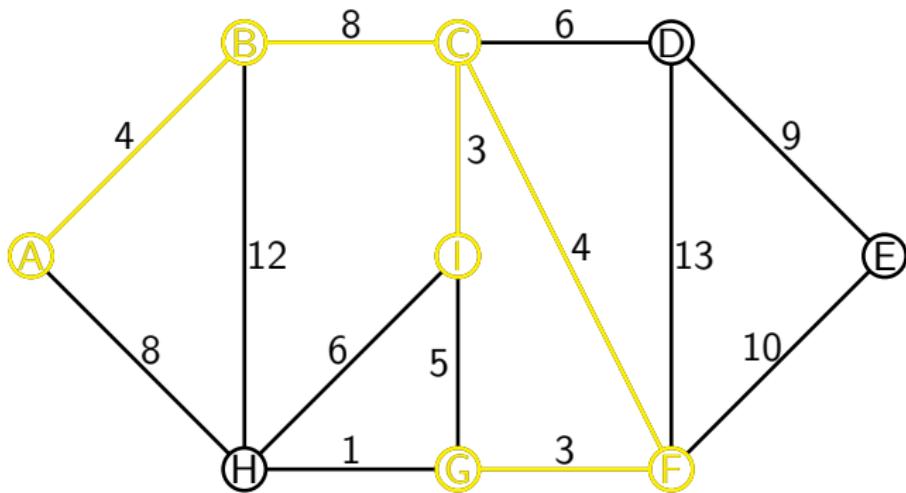
## Ejemplo



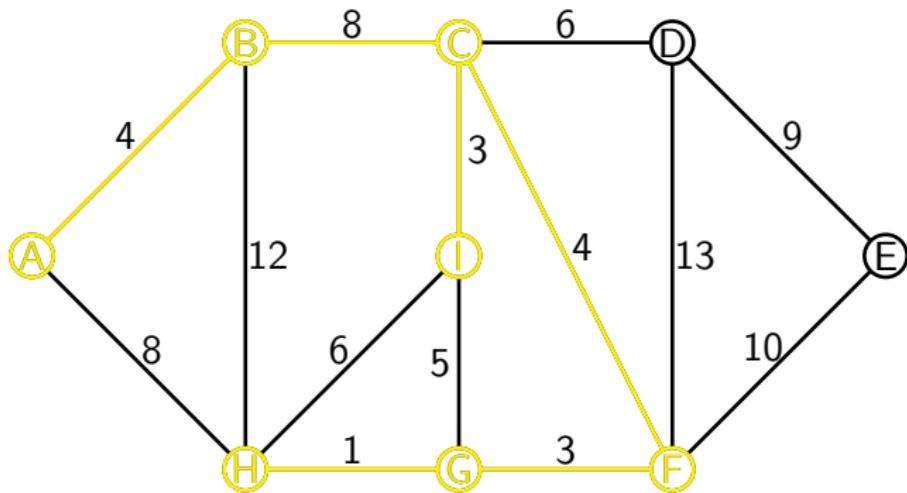
## Ejemplo



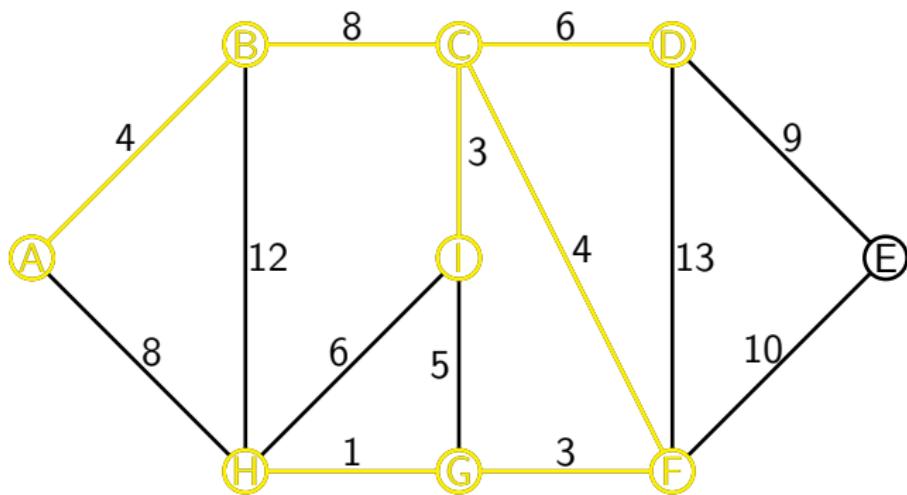
## Ejemplo



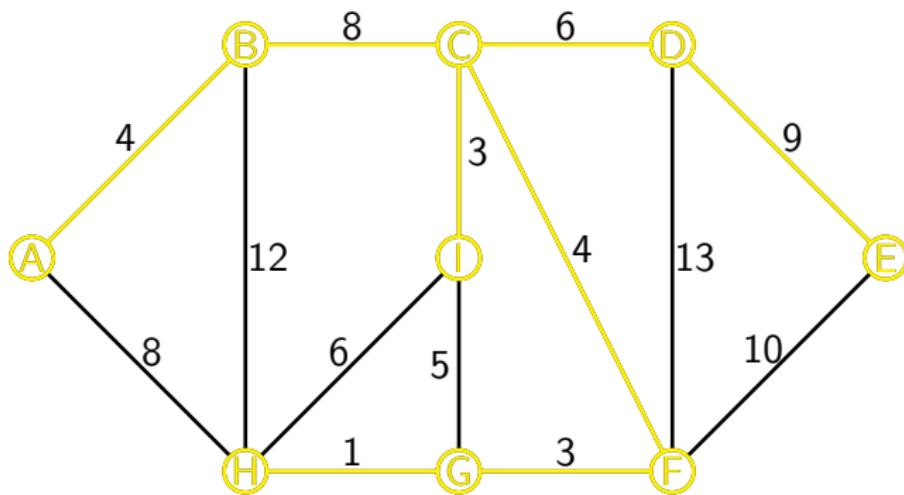
## Ejemplo



## Ejemplo



## Ejemplo



## Algoritmo de Prim

---

- **Teorema.** Sea  $F$  un subconjunto de aristas en un AGM de  $G$ , y sea  $S$  un conjunto de vértices de una componente conexa de  $F$ . Sea  $ij$  una arista de peso mínimo entre  $S$  y  $\bar{S}$ . Entonces, existe un AGM de  $G$  que incluye a todas las aristas de  $F \cup \{ij\}$ .

## Algoritmo de Prim

---

- **Teorema.** Sea  $F$  un subconjunto de aristas en un AGM de  $G$ , y sea  $S$  un conjunto de vértices de una componente conexa de  $F$ . Sea  $ij$  una arista de peso mínimo entre  $S$  y  $\bar{S}$ . Entonces, existe un AGM de  $G$  que incluye a todas las aristas de  $F \cup \{ij\}$ .
- **Teorema.** El algoritmo de Prim es **correcto**. Es decir, dado un grafo  $G$  conexo, determina un árbol generador mínimo de  $G$ .

# Algoritmo de Prim

---

- **Teorema.** Sea  $F$  un subconjunto de aristas en un AGM de  $G$ , y sea  $S$  un conjunto de vértices de una componente conexa de  $F$ . Sea  $ij$  una arista de peso mínimo entre  $S$  y  $\bar{S}$ . Entonces, existe un AGM de  $G$  que incluye a todas las aristas de  $F \cup \{ij\}$ .
- **Teorema.** El algoritmo de Prim es **correcto**. Es decir, dado un grafo  $G$  conexo, determina un árbol generador mínimo de  $G$ .
- ¿Cuál es la complejidad computacional de este algoritmo?

# Algoritmo de Kruskal

---



Joseph Kruskal (1928–2010)

# Algoritmo de Kruskal

---

$E_T := \emptyset$

$i := 1$

**mientras**  $i \leq n - 1$  **hacer**

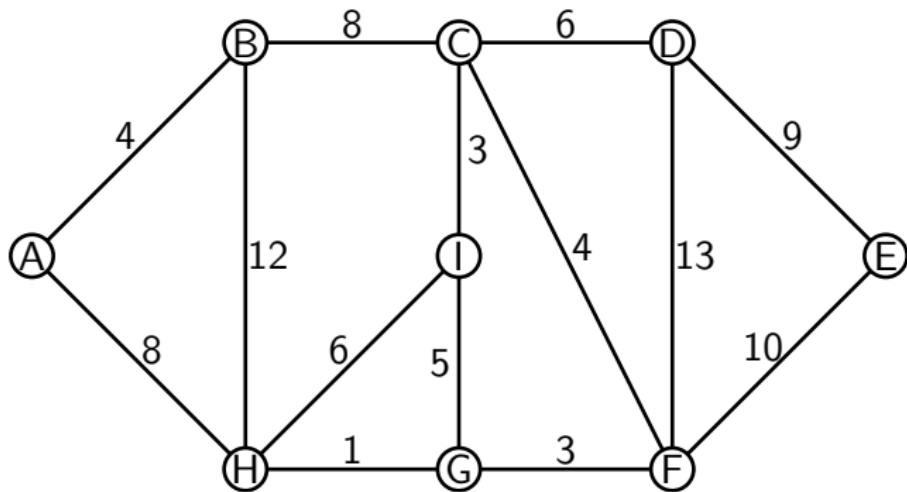
elegir  $e \in X$  tal que  $l(e)$  sea mínima entre las  
aristas que no forman circuito con las  
aristas que ya están en  $E_T$

$X_T := X_T \cup \{e\}$

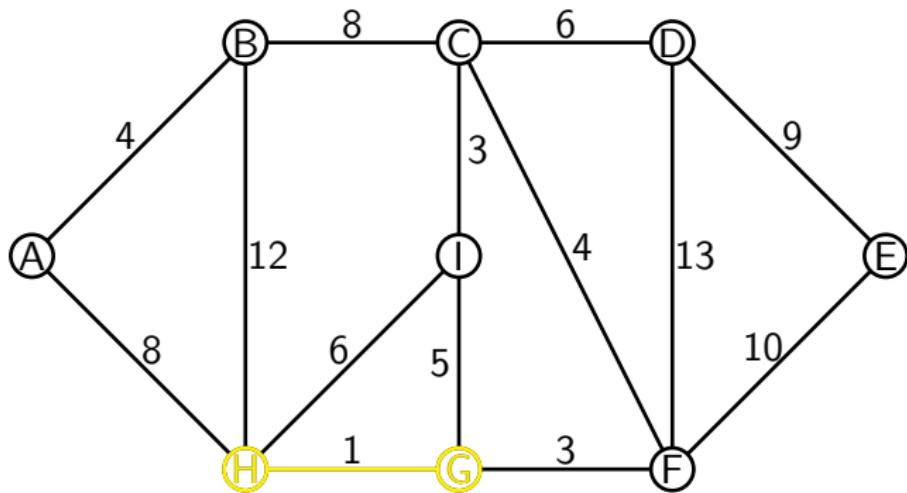
$i := i + 1$

**retornar**  $T = (V, E_T)$

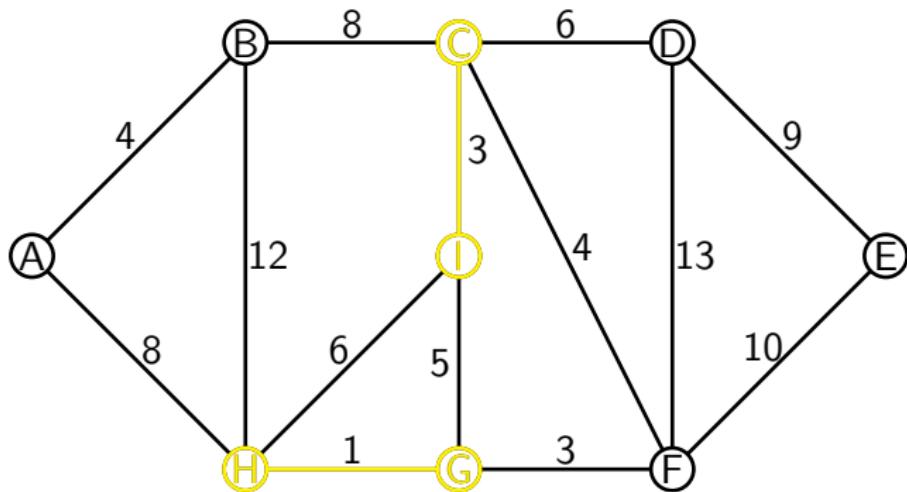
## Ejemplo



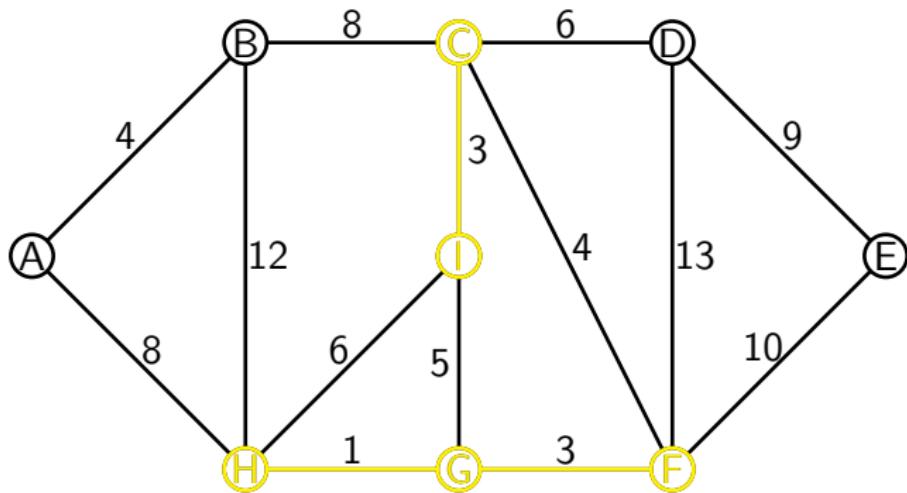
## Ejemplo



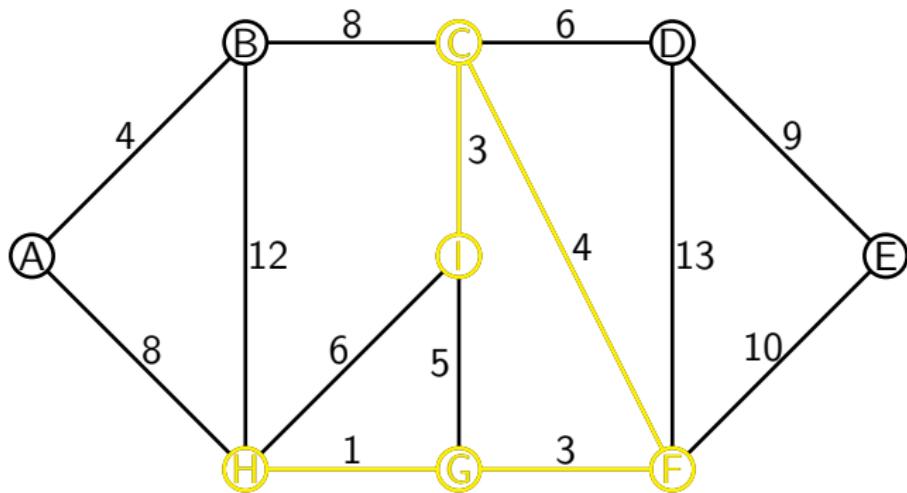
## Ejemplo



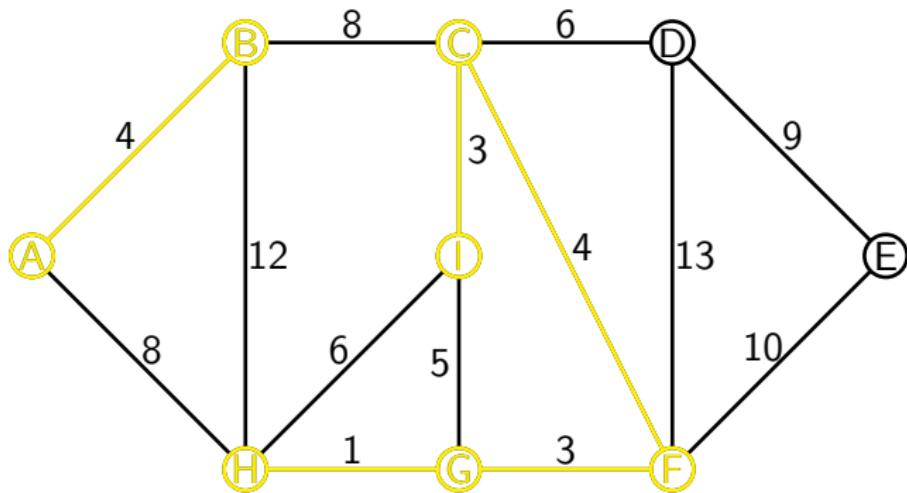
## Ejemplo



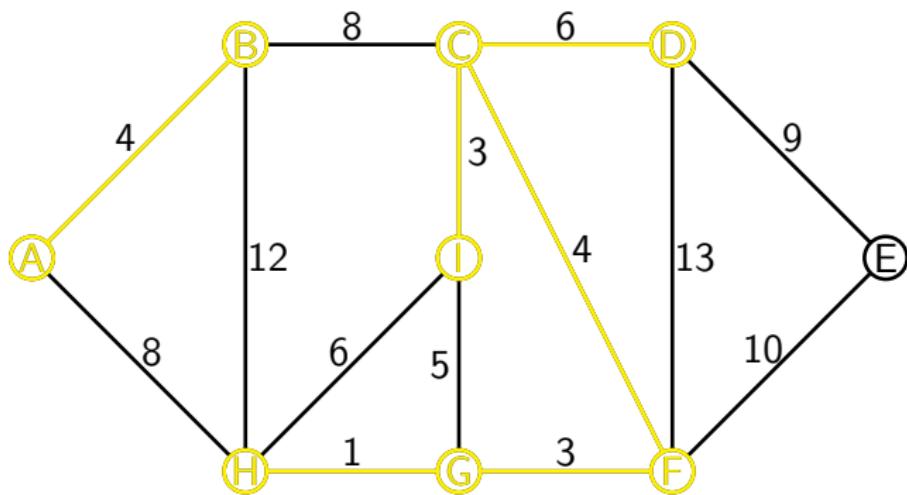
## Ejemplo



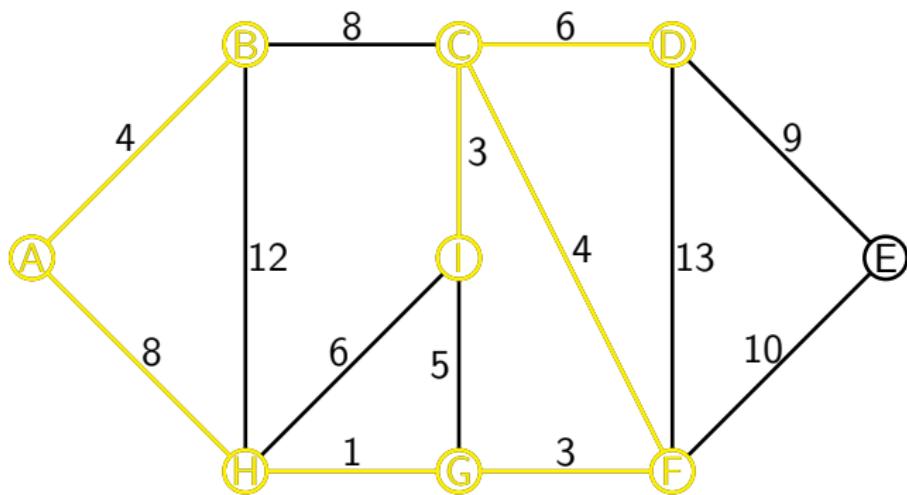
## Ejemplo



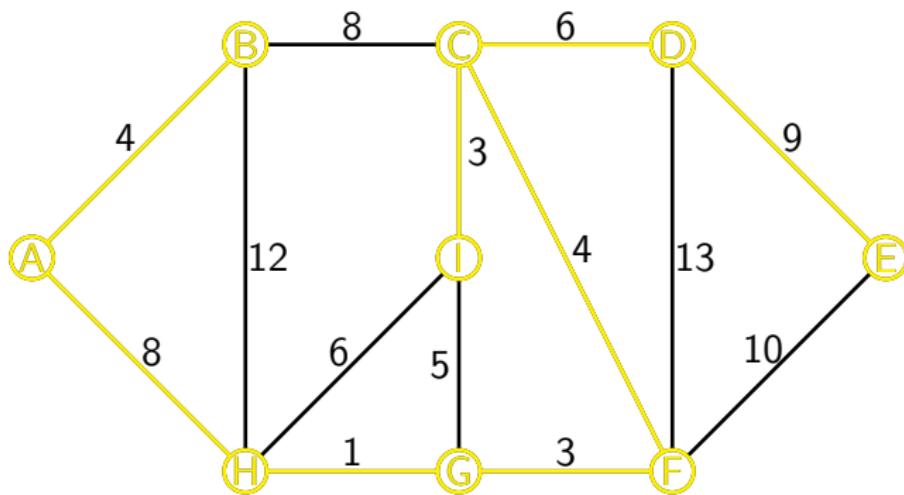
## Ejemplo



## Ejemplo



## Ejemplo



# Manejo de archivos en Java

Programación III - UNGS

## Escritura de archivos de texto

---

- Para escribir en un **archivo de texto** debemos generar un **FileOutputStream** asociado con el archivo, y un **OutputStreamWriter** asociado con el OutputStream.

## Escritura de archivos de texto

---

- Para escribir en un **archivo de texto** debemos generar un **FileOutputStream** asociado con el archivo, y un **OutputStreamWriter** asociado con el OutputStream.
- A diferencia de la serialización, el StreamWriter no es un **ObjectOutputStream**.

## Escritura de archivos de texto

---

- Para escribir en un **archivo de texto** debemos generar un **FileOutputStream** asociado con el archivo, y un **OutputStreamWriter** asociado con el OutputStream.
- A diferencia de la serialización, el StreamWriter no es un **ObjectOutputStream**.
- Es importante tener en cuenta las **excepciones**. Al manejar archivos es más probable la ocurrencia de excepciones y situaciones no previstas en tiempo de ejecución.

# Escritura de archivos de texto

---

```
1 try // Debe estar en un try/catch
2 {
3     FileOutputStream fos = new FileOutputStream("test.txt");
4     OutputStreamWriter out = new OutputStreamWriter(fos);
5
6     out.write("Escribiendo una línea\r\n"); // Notar el salto de linea
7     out.write("Escribiendo otra línea");
8     out.close();
9 }
10 catch(Exception e) { ... }
```

---

## Lectura desde archivos de texto

---

- Para leer desde un archivo de texto se genera una instancia de **Scanner**, que simplifica el **parsing** del archivo.

## Lectura desde archivos de texto

---

- Para leer desde un archivo de texto se genera una instancia de **Scanner**, que simplifica el **parsing** del archivo.

- FileInputStream fis = new FileInputStream("test.txt");  
Scanner scanner = new Scanner(fis);  
  
String s1 = scanner.nextLine();  
String s2 = scanner.nextLine();  
  
...  
scanner.close();

---

## Lectura desde archivos de texto

---

- La clase **Scanner** contiene métodos para leer todos los tipos de datos primitivos.

## Lectura desde archivos de texto

---

- La clase **Scanner** contiene métodos para leer todos los tipos de datos primitivos.
- Por ejemplo, si el archivo contiene un listado de números enteros:

## Lectura desde archivos de texto

---

- La clase **Scanner** contiene métodos para leer todos los tipos de datos primitivos.
- Por ejemplo, si el archivo contiene un listado de números enteros:

1   **while( scanner.hasNextInt() )**  
2     **System.out.println( scanner.nextInt() );**

---

## Lectura desde archivos de texto

---

- La clase **Scanner** contiene métodos para leer todos los tipos de datos primitivos.
- Por ejemplo, si el archivo contiene un listado de números enteros:

1     **while( scanner.hasNextInt() )**  
2        System.out.println( scanner.nextInt() );

---

- El método **nextInt()** saltea espacios en blanco, saltos de línea y tabulaciones. De esta forma, se puede leer cualquier conjunto de datos en un archivo de **texto plano** con un formato predeterminado.

## Lectura desde la consola

---

- La lectura desde la **consola** se realiza por medio de un procedimiento similar:

## Lectura desde la consola

---

- La lectura desde la **consola** se realiza por medio de un procedimiento similar:

- ---

```
1  InputStreamReader stdin = new InputStreamReader(System.in);
2  BufferedReader console = new BufferedReader(stdin);
3  try
4  {
5      String str = console.readLine();
6      System.out.println("El usuario ingresó: " + str);
7  }
8  catch(IOException ex) { ... }
```

---

## Lectura desde la consola

---

- Para leer valores numéricos, se debe realizar la conversión desde **String** explícitamente:

## Lectura desde la consola

---

- Para leer valores numéricos, se debe realizar la conversión desde **String** explícitamente:

```
1  InputStreamReader stdin = new InputStreamReader(System.in);
2  BufferedReader console = new BufferedReader(stdin);
3  try
4  {
5      System.out.print("Ingrese un número: ");
6      String str = console.readLine();
7      double valor = Double.parseDouble(str);
8      ...
9  }
10 catch(IOException ex) { ... }
```

---

## Lectura de archivos xml

---

- El **formato xml** (*extensible markup language*) es un formato desarrollado por el *World Wide Web Consortium* para guardar **información estructurada** en archivos de texto.

## Lectura de archivos xml

---

- El **formato xml** (*extensible markup language*) es un formato desarrollado por el *World Wide Web Consortium* para guardar **información estructurada** en archivos de texto.
- Por ejemplo, del siguiente modo especificamos varios registros de personas:

## Lectura de archivos xml

---

- El **formato xml** (*extensible markup language*) es un formato desarrollado por el *World Wide Web Consortium* para guardar **información estructurada** en archivos de texto.
- Por ejemplo, del siguiente modo especificamos varios registros de personas:

1 <Mensaje>  
2     <Remitente>  
3         <Nombre>Juanito Gonzalez</Nombre>  
4         <Edad>27</Edad>  
5     </Remitente>  
6     <Destinatario>  
7         <Nombre>Jorgelina Perez</Nombre>  
8         <Mail>jperez@yahoo.com</Mail>  
9     </Destinatario>  
10 </Mensaje>

---

## Lectura de archivos xml

---

- Generamos un **Document** asociado con el archivo ...

# Lectura de archivos xml

---

- Generamos un **Document** asociado con el archivo ...

- \_\_\_\_\_

```
1 import javax.xml.parsers.DocumentBuilder;
2 import javax.xml.parsers.DocumentBuilderFactory;
3 import org.w3c.dom.Document;
4
5 DocumentBuilder builder =
6     DocumentBuilderFactory.newInstance().newDocumentBuilder();
7 File f = new File(path);
8 Document documento = builder.parse(f);
9 recorrer(documento);
```

---

## Lectura de archivos xml

---

- ... y recorremos **recursivamente** el documento:

# Lectura de archivos xml

---

- ... y recorremos **recursivamente** el documento:

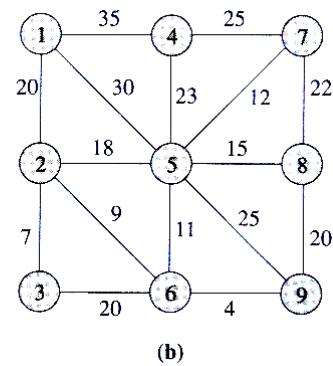
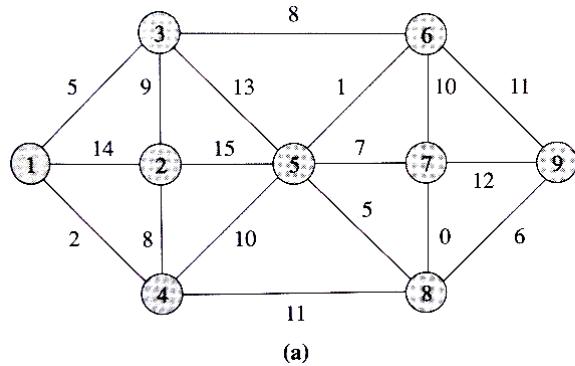
```
1 public void recorrer(Node nodo)
2 {
3     if( nodo != null )
4     {
5         System.out.println(nodo.getNodeName() + " - " +
6             nodo.getNodeValue());
7         NodeList hijos = nodo.getChildNodes();
8
9         for(int i = 0; i < hijos.getLength(); i++)
10            recorrer( hijos.item(i) );
11    }
12 }
```

---

Programación III - Universidad Nacional de General Sarmiento  
Ejercicios: Árbol generador mínimo

---

1. Ejecutar los algoritmos de Prim y Kruskal sobre las siguientes instancias.



2. Dar un grafo con pesos en las aristas que tenga más de un árbol generador mínimo (notar que si hay más de un árbol generador mínimo, entonces todos tienen el mismo peso).
3. Dar un grafo con pesos en las aristas que tenga exactamente dos árboles generadores mínimos distintos.
4. ¿Por qué decimos que el problema de calcular un árbol generador mínimo está bien resuelto?
5. Dar un algoritmo para calcular un árbol generador máximo de un grafo con pesos en las aristas.