Ejemplo, generación de #s aleatorios

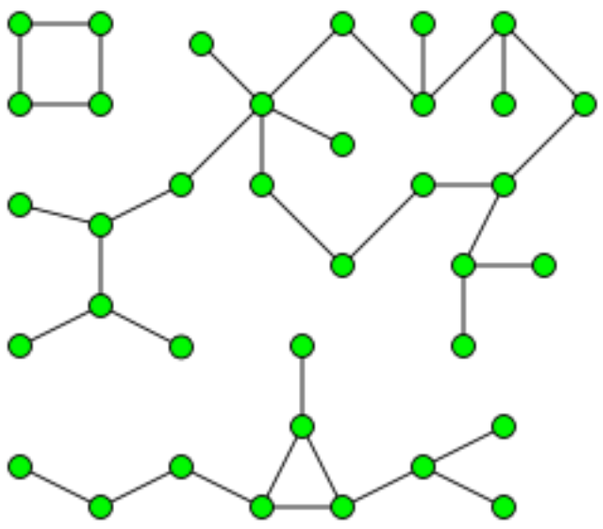
- Supón que tienes un generador de números aleatorios enteros [0,2] con distribución uniforme, y tu necesitas generar números con **distribución uniforme** [0,4].
- Alguien propone generar 2 números A,B y sumarlos C = A + B para generar los números que necesitas.
- Muestra por qué esta solución **está mal**, pues no genera números con distribución uniforme.

Introducción al análisis de algoritmos

Computación y Algoritmos mat-151

Aplicaciones:

- Conexiones en redes, de comunicación, de energía eléctrica, etc.
- Búsquedas en bases de datos gigantescas (transacciones comerciales)
- Encontrar caminos cortos en grafos, en Planeación de movimiento en robótica, en busquedas de rutas en mapas
- The Human Genome Project: identificar los 100,000 genes del DNA humano.
- Encontrar páginas web en donde una información en particular está contenida.



Componentes conectados en VC

Análisis empírico

Análisis empírico

- Si tenemos dos algoritmos que solucionan el mismo problema, ¿cómo podemos saber cuál es mejor?

Análisis empírico

- Si tenemos dos algoritmos que solucionan el mismo problema, ¿cómo podemos saber cuál es mejor?

- ¡Corremos los dos a ver cuál termina primero!

Análisis empírico

- Si tenemos dos algoritmos que solucionan el mismo problema, ¿cómo podemos saber cuál es mejor?

- ¡Corremos los dos a ver cuál termina primero!

- No podría escapar al ojo menos atento si uno de los algoritmos termina en 3 segundos y el otro después de 30 !

Análisis empírico

- Si tenemos dos algoritmos que solucionan el mismo problema, ¿cómo podemos saber cuál es mejor?

- ¡Corremos los dos a ver cuál termina primero!

- No podría escapar al ojo menos atento si uno de los algoritmos termina en 3 segundos y el otro después de 30 !

- Cuando los análisis **empíricos** toman demasiado tiempo, se requiere un análisis **matemático**: esperar una hora o un día para que termine de ejecutarse un programa no es la mejor manera de determinar que es lento.

Análisis empírico: retos

Análisis empírico: retos

- implementar los algoritmos a comparar.

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.
- determinar las entradas apropiadas para probar el algoritmo.

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.
- determinar las entradas apropiadas para probar el algoritmo.
  - datos **reales**.

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.
- determinar las entradas apropiadas para probar el algoritmo.
  - datos **reales**.
  - datos **aleatorios**.

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.
- determinar las entradas apropiadas para probar el algoritmo.
  - datos **reales**.
  - datos **aleatorios**.
  - datos **perversos.**

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.
- determinar las entradas apropiadas para probar el algoritmo.
  - datos **reales**.
  - datos **aleatorios**.
  - datos **perversos.**
- ejemplo: algoritmo de ordenamiento.

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.
- determinar las entradas apropiadas para probar el algoritmo.
  - datos **reales**.
  - datos **aleatorios**.
  - datos **perversos.**
- ejemplo: algoritmo de ordenamiento.
  - datos reales como las palabras en un libro,

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.
- determinar las entradas apropiadas para probar el algoritmo.
  - datos **reales**.
  - datos **aleatorios**.
  - datos **perversos.**
- ejemplo: algoritmo de ordenamiento.
  - datos reales como las palabras en un libro,
  - datos aleatorios generados automáticamente,

Análisis empírico: retos

- implementar los algoritmos a comparar.
- desarrollar una implementación **correcta** y **completa**.
- determinar las entradas apropiadas para probar el algoritmo.
  - datos **reales**.
  - datos **aleatorios**.
  - datos **perversos.**
- ejemplo: algoritmo de ordenamiento.
  - datos reales como las palabras en un libro,
  - datos aleatorios generados automáticamente,
  - datos perversos como archivos que contienen el mismo número.

Análisis empírico: peligros

Análisis empírico: peligros

- comparar implementaciones con diferentes máquinas, compiladores o sistemas.

Análisis empírico: peligros

- comparar implementaciones con diferentes máquinas, compiladores o sistemas.
- comparar implementaciones de sistemas enteros (programas enormes).

Análisis empírico: peligros

- comparar implementaciones con diferentes máquinas, compiladores o sistemas.
- comparar implementaciones de sistemas enteros (programas enormes).
- las diferentes implementaciones se hicieron con diferente cuidado

Análisis empírico: peligros

- comparar implementaciones con diferentes máquinas, compiladores o sistemas.
- comparar implementaciones de sistemas enteros (programas enormes).
- las diferentes implementaciones se hicieron con diferente cuidado
- Algunas soluciones:

Análisis empírico: peligros

- comparar implementaciones con diferentes máquinas, compiladores o sistemas.
- comparar implementaciones de sistemas enteros (programas enormes).
- las diferentes implementaciones se hicieron con diferente cuidado
- Algunas soluciones:
  - comparar algoritmos relativamente similares con los mínimos cambios posibles en el código.

Análisis empírico: peligros

- comparar implementaciones con diferentes máquinas, compiladores o sistemas.
- comparar implementaciones de sistemas enteros (programas enormes).
- las diferentes implementaciones se hicieron con diferente cuidado
- Algunas soluciones:
  - comparar algoritmos relativamente similares con los mínimos cambios posibles en el código.
  - comparar bloques de operaciones básicas.

Algoritmos “buenos” (1)

Algoritmos “buenos” (1)

- No solo nos importa que los algoritmos sean factibles y en se ejecuten en tiempo finito, sino también que el proceso completo de la ejecución del programa que realiza este algoritmo sea **eficiente** en términos de:
    - tiempo de ejecución,
    - cantidad de memoria necesaria.
- Para un problema dado, puede haber gran cantidad de algoritmos que lo resuelvan, **con propiedades muy diferentes**!
- Según el contexto, se preferirá tal o tal algoritmo ....

Algoritmos “buenos” (2)

Algoritmos “buenos” (2)

- Pueden haber otros factores que influyan nuestras elecciones:

Algoritmos “buenos” (2)

- Pueden haber otros factores que influyan nuestras elecciones:
  - **consideraciones de *hardware:*** por ejemplo como usa la memoria el algoritmo es importante si tenemos poca memoria física y usamos memoria virtual.

Algoritmos “buenos” (2)

- Pueden haber otros factores que influyan nuestras elecciones:
  - **consideraciones de *hardware:*** por ejemplo como usa la memoria el algoritmo es importante si tenemos poca memoria física y usamos memoria virtual.
  - el hecho de que el algoritmo sea llamado muchas veces **en una secuencia de llamadas**: esto puede eventualmente *amortiguar* la complejidad, lo que puede resultar “interesante”. (por ejemplo, si se ejecuta rápido 9 veces de cada 10 ...)

Algoritmos “buenos” (2)

- Pueden haber otros factores que influyan nuestras elecciones:
  - **consideraciones de *hardware:*** por ejemplo como usa la memoria el algoritmo es importante si tenemos poca memoria física y usamos memoria virtual.
  - el hecho de que el algoritmo sea llamado muchas veces **en una secuencia de llamadas**: esto puede eventualmente *amortiguar* la complejidad, lo que puede resultar “interesante”. (por ejemplo, si se ejecuta rápido 9 veces de cada 10 ...)
  - consideraciones **globales**: las estructuras usadas en el algoritmo pueden ser reutilizadas en otras partes del programa.

Análisis de algoritmos (1)

Análisis de algoritmos (1)

- Estudiar las especificaciones del algoritmo para sacar conclusiones sobre cómo su implementación - el programa - se comportará en general.

Análisis de algoritmos (1)

- Estudiar las especificaciones del algoritmo para sacar conclusiones sobre cómo su implementación - el programa - se comportará en general.
- ¿Pero que podemos analizar de un algoritmo?

Análisis de algoritmos (1)

- Estudiar las especificaciones del algoritmo para sacar conclusiones sobre cómo su implementación - el programa - se comportará en general.
- ¿Pero que podemos analizar de un algoritmo?
  - determinar el tiempo de ejecución en función de sus entradas.

Análisis de algoritmos (1)

- Estudiar las especificaciones del algoritmo para sacar conclusiones sobre cómo su implementación - el programa - se comportará en general.
- ¿Pero que podemos analizar de un algoritmo?
  - determinar el tiempo de ejecución en función de sus entradas.
  - determinar su espacio total o máximo necesario en memoria.

Análisis de algoritmos (1)

- Estudiar las especificaciones del algoritmo para sacar conclusiones sobre cómo su implementación - el programa - se comportará en general.
- ¿Pero que podemos analizar de un algoritmo?
  - determinar el tiempo de ejecución en función de sus entradas.
  - determinar su espacio total o máximo necesario en memoria.
  - determinar el tamaño del código

Análisis de algoritmos (1)

- Estudiar las especificaciones del algoritmo para sacar conclusiones sobre cómo su implementación - el programa - se comportará en general.
- ¿Pero que podemos analizar de un algoritmo?
  - determinar el tiempo de ejecución en función de sus entradas.
  - determinar su espacio total o máximo necesario en memoria.
  - determinar el tamaño del código
  - determinar si el resultado calculado es el correcto.

Análisis de algoritmos (1)

- Estudiar las especificaciones del algoritmo para sacar conclusiones sobre cómo su implementación - el programa - se comportará en general.
- ¿Pero que podemos analizar de un algoritmo?
  - determinar el tiempo de ejecución en función de sus entradas.
  - determinar su espacio total o máximo necesario en memoria.
  - determinar el tamaño del código
  - determinar si el resultado calculado es el correcto.
  - determinar qué tan fácil es de leer, entender, modificar

Análisis de algoritmos (1)

- Estudiar las especificaciones del algoritmo para sacar conclusiones sobre cómo su implementación - el programa - se comportará en general.
- ¿Pero que podemos analizar de un algoritmo?
  - determinar el tiempo de ejecución en función de sus entradas.
  - determinar su espacio total o máximo necesario en memoria.
  - determinar el tamaño del código
  - determinar si el resultado calculado es el correcto.
  - determinar qué tan fácil es de leer, entender, modificar
  - determinar su robustez, es decir, qué tan bien se comporta con entradas equivocadas o inesperadas.

Análisis de Algoritmos (2)

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada
    - el hardware

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada
    - el hardware
        - el tipo de procesador utilizado (tipo y velocidad)

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada
    - el hardware
        - el tipo de procesador utilizado (tipo y velocidad)
        - memoria disponible (caché y RAM)

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada
    - el hardware
        - el tipo de procesador utilizado (tipo y velocidad)
        - memoria disponible (caché y RAM)
        - espacio de disco disponible

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada
    - el hardware
        - el tipo de procesador utilizado (tipo y velocidad)
        - memoria disponible (caché y RAM)
        - espacio de disco disponible
    - el lenguaje de programación en el que se ha especificado el algoritmo

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada
    - el hardware
        - el tipo de procesador utilizado (tipo y velocidad)
        - memoria disponible (caché y RAM)
        - espacio de disco disponible
    - el lenguaje de programación en el que se ha especificado el algoritmo
    - el compilador / intérprete utilizado

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada
    - el hardware
        - el tipo de procesador utilizado (tipo y velocidad)
        - memoria disponible (caché y RAM)
        - espacio de disco disponible
    - el lenguaje de programación en el que se ha especificado el algoritmo
    - el compilador / intérprete utilizado
    - el sistema operativo utilizado.

Análisis de Algoritmos (2)

- En este curso analizaremos principalmente el ***tiempo de ejecución.***
- Existen muchos factores que pueden afectar el tiempo de ejecución de un programa:
    - los datos de entrada
    - el hardware
        - el tipo de procesador utilizado (tipo y velocidad)
        - memoria disponible (caché y RAM)
        - espacio de disco disponible
    - el lenguaje de programación en el que se ha especificado el algoritmo
    - el compilador / intérprete utilizado
    - el sistema operativo utilizado.
- Se necesita un **modelo** de la **tecnología con la que se va a implementar** como programas computacionales.

No todo es del tipo C y C++

Las cláusulas sin cuerpo (es decir, antecedente) son llamados **hechos** porque siempre son ciertos. Un ejemplo de un hecho es:

```
gato(tom).
```

que es equivalente a la regla:

```
gato(tom) :- true.
```

El predicado predefinido `true/0` siempre es verdad.

Dado el hecho anterior, se puede preguntar:

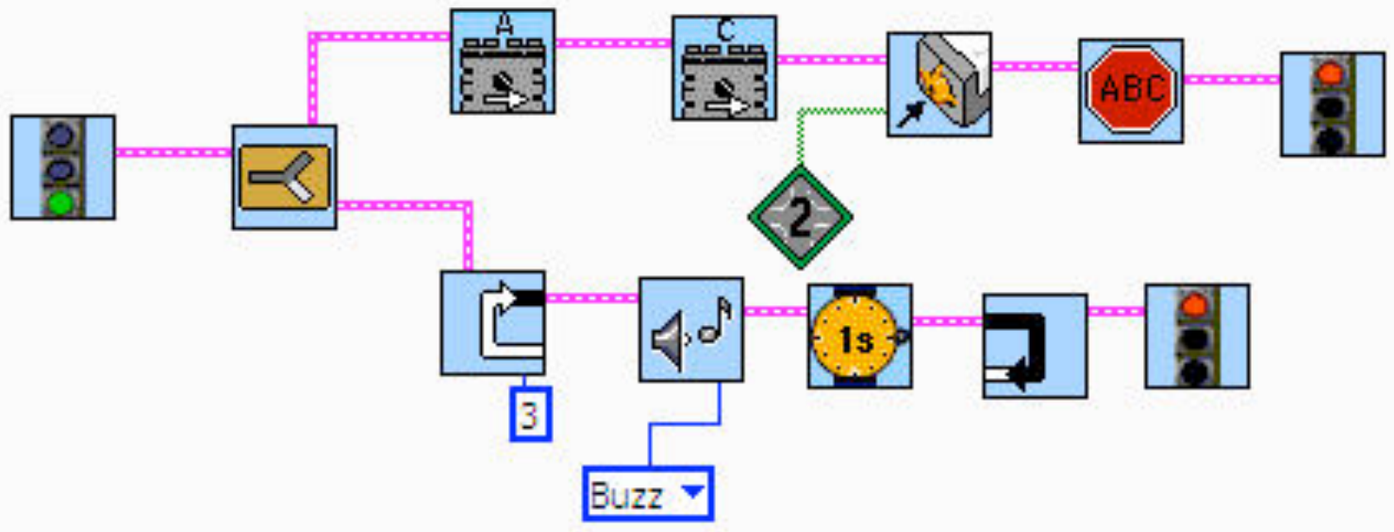
¿ es tom un gato ?

```
?- gato(tom).
Yes
```

¿ que cosas son gatos ?

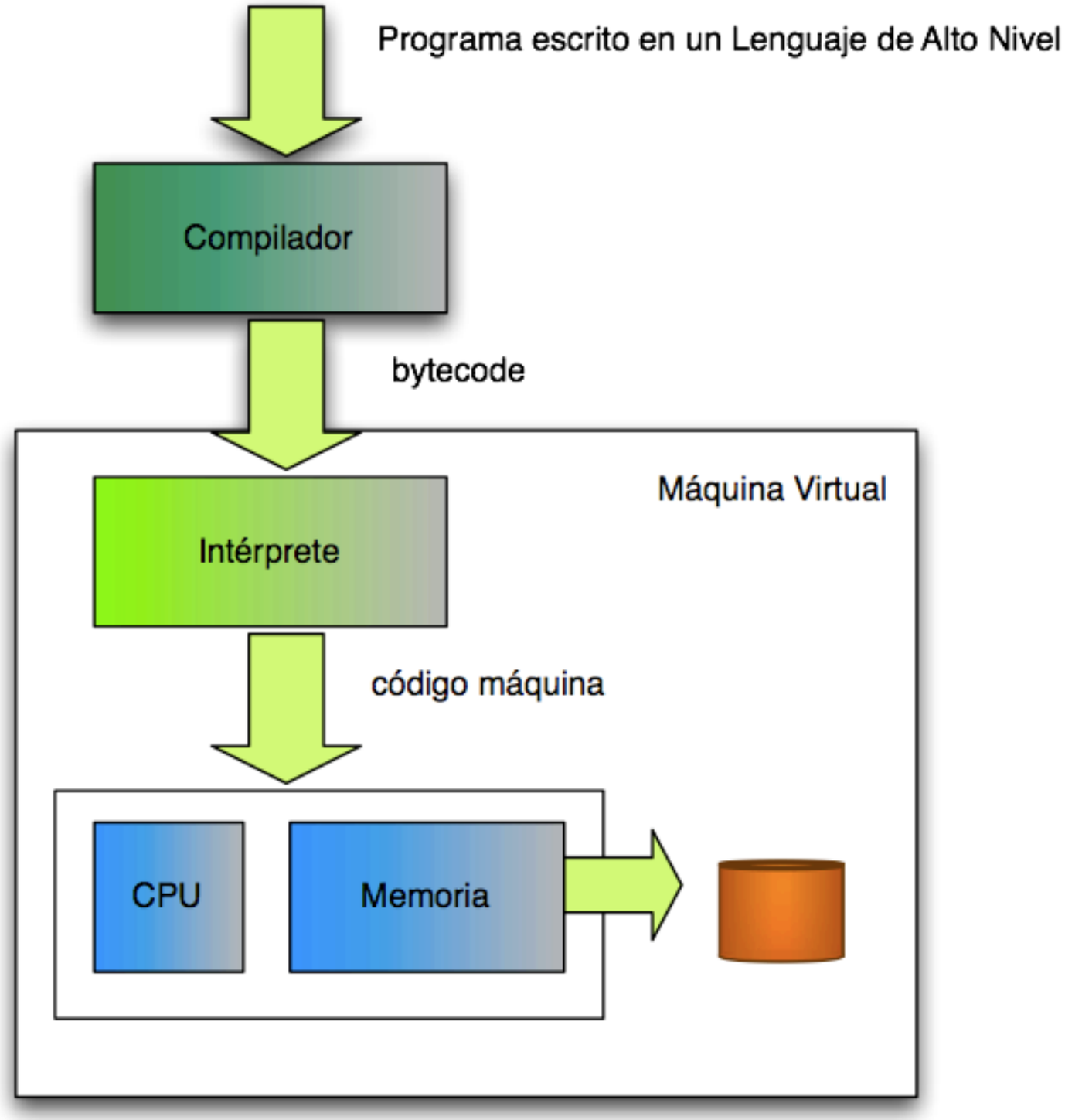
```
?- gato(X).
X = tom
```

Prolog



Robolab
LabView

Las cosas pueden ser complicadas de especificar:



Análisis de algoritmos (3)

Análisis de algoritmos (3)

- **tamaño de la entrada** - depende del problema estudiado,

Análisis de algoritmos (3)

- **tamaño de la entrada** - depende del problema estudiado,
    - *número de elementos de entrada* (por ejemplo en problemas de sorting).

Análisis de algoritmos (3)

- **tamaño de la entrada** - depende del problema estudiado,
  - *número de elementos de entrada* (por ejemplo en problemas de sorting).
  - *tamaño el bits de la estructura de datos* de entrada (por ejemplo al multiplicar dos números)

Análisis de algoritmos (3)

- **tamaño de la entrada** - depende del problema estudiado,
  - *número de elementos de entrada* (por ejemplo en problemas de sorting).
  - *tamaño el bits de la estructura de datos* de entrada (por ejemplo al multiplicar dos números)
- **tiempo de ejecución** - número de operaciones básicas o primitivas ejecutadas. Es conveniente calcular de manera independiente a la computadora utilizada.

Modelos de computación (1)

Modelos de computación (1)

- Los siguientes dos modelos son ejemplos de máquinas virtuales, ideales.

Modelos de computación (1)

- Los siguientes dos modelos son ejemplos de máquinas virtuales, ideales.
  - Realizan operaciones de forma secuencial (no hay concurrencia)

Modelos de computación (1)

- Los siguientes dos modelos son ejemplos de máquinas virtuales, ideales.
  - Realizan operaciones de forma secuencial (no hay concurrencia)
  - Proporcionan instrucciones básicas que son asociadas a un orden de tiempo de ejecución (un costo)

Modelos de computación (1)

- Los siguientes dos modelos son ejemplos de máquinas virtuales, ideales.
  - Realizan operaciones de forma secuencial (no hay concurrencia)
  - Proporcionan instrucciones básicas que son asociadas a un orden de tiempo de ejecución (un costo)
  - Son modelos simplificados de las computadoras reales.

Máquina de Turing (1)

Máquina de Turing (1)

- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.

Máquina de Turing (1)

- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.
- cinta que se extiende al infinito en ambas direcciones.

Máquina de Turing (1)

- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.
- cinta que se extiende al infinito en ambas direcciones.
- la cinta se divide en bits de tamaño fijo y cada bit puede contener 0s o 1s (o un símbolo de un alfabeto finito)

Máquina de Turing (1)

- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.
- cinta que se extiende al infinito en ambas direcciones.
- la cinta se divide en bits de tamaño fijo y cada bit puede contener 0s o 1s (o un símbolo de un alfabeto finito)
- utilizando este formato la cinta contiene datos e instrucciones.

Máquina de Turing (1)

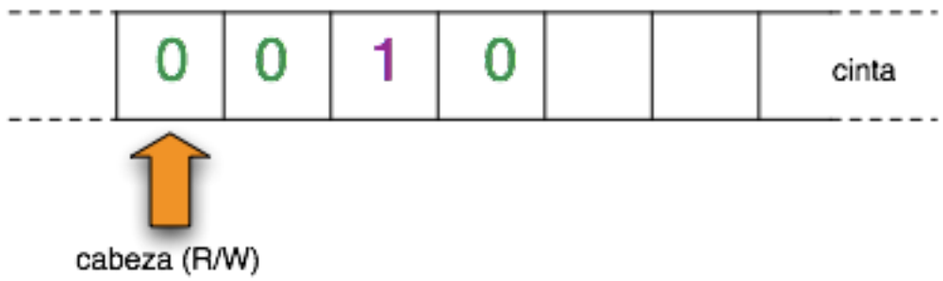
- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.
- cinta que se extiende al infinito en ambas direcciones.
- la cinta se divide en bits de tamaño fijo y cada bit puede contener 0s o 1s (o un símbolo de un alfabeto finito)
- utilizando este formato la cinta contiene datos e instrucciones.
- unidad capaz de leer y escribir datos individuales: la cabeza

Máquina de Turing (1)

- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.
- cinta que se extiende al infinito en ambas direcciones.
- la cinta se divide en bits de tamaño fijo y cada bit puede contener 0s o 1s (o un símbolo de un alfabeto finito)
- utilizando este formato la cinta contiene datos e instrucciones.
- unidad capaz de leer y escribir datos individuales: la cabeza
- controlada por un autómata finito.

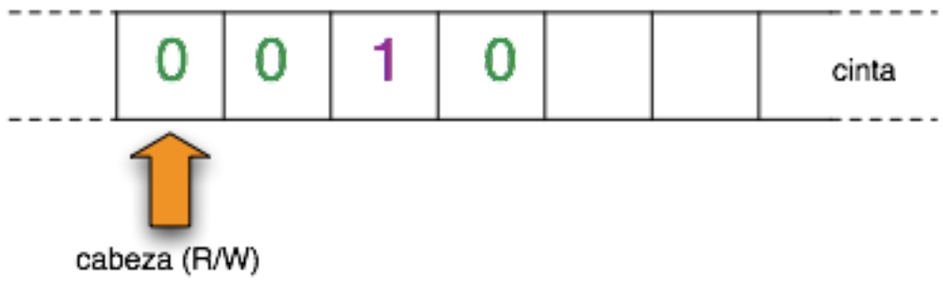
Máquina de Turing (1)

- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.
- cinta que se extiende al infinito en ambas direcciones.
- la cinta se divide en bits de tamaño fijo y cada bit puede contener 0s o 1s (o un símbolo de un alfabeto finito)
- utilizando este formato la cinta contiene datos e instrucciones.
- unidad capaz de leer y escribir datos individuales: la cabeza
- controlada por un autómata finito.



Máquina de Turing (1)

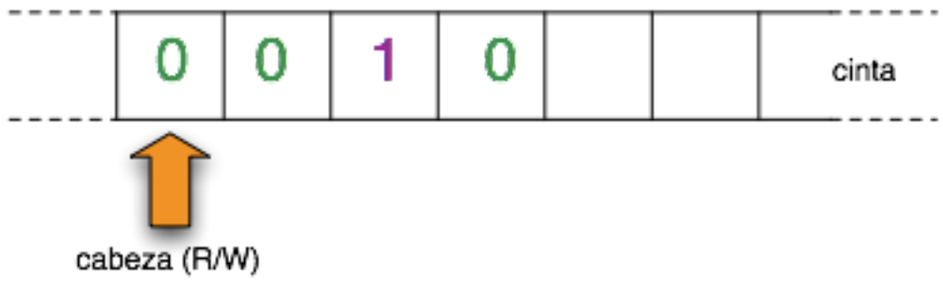
- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.
- cinta que se extiende al infinito en ambas direcciones.
- la cinta se divide en bits de tamaño fijo y cada bit puede contener 0s o 1s (o un símbolo de un alfabeto finito)
- utilizando este formato la cinta contiene datos e instrucciones.
- unidad capaz de leer y escribir datos individuales: la cabeza
- controlada por un autómata finito.



"Si tu estado es 42 y el símbolo que aparece es '0', entonces reemplazar por '1', muevete una localidad a la derecha y actualiza el nuevo estado a 17."

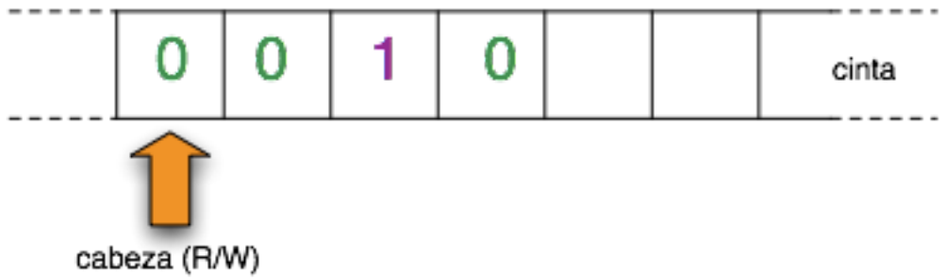
Máquina de Turing (1)

- Alan Turing (1912-1954) propuso el modelo de la **máquina de Turing** en 1936.
- cinta que se extiende al infinito en ambas direcciones.
- la cinta se divide en bits de tamaño fijo y cada bit puede contener 0s o 1s (o un símbolo de un alfabeto finito)
- utilizando este formato la cinta contiene datos e instrucciones.
- unidad capaz de leer y escribir datos individuales: la cabeza
- controlada por un autómata finito.



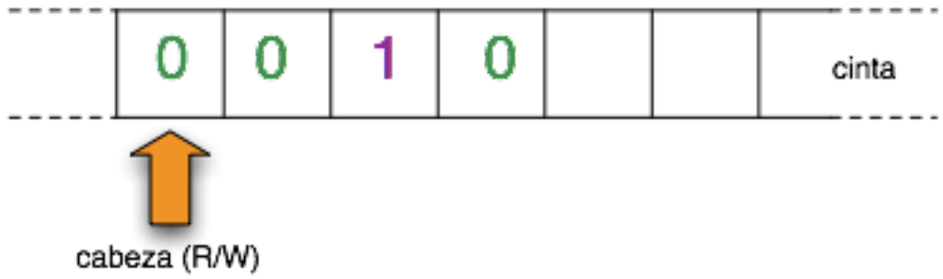
"Si tu estado es 42 y el símbolo que aparece es '0', entonces reemplazar por '1', muevete una localidad a la derecha y actualiza el nuevo estado a 17."

Máquina de Turing (2)



Máquina de Turing (2)

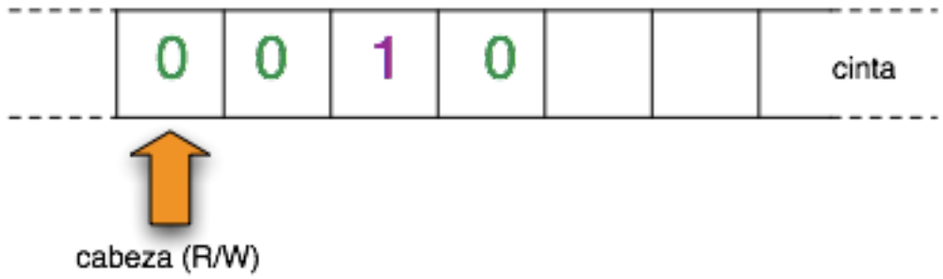
- Operaciones posibles:



Máquina de Turing (2)

- Operaciones posibles:

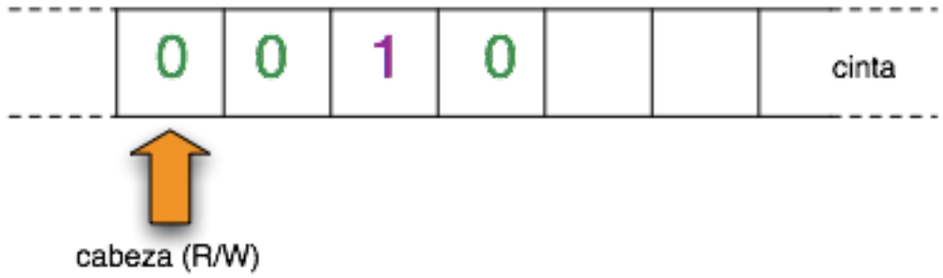
1. **Leer** el bit al que apunta la cabeza.



Máquina de Turing (2)

- Operaciones posibles:

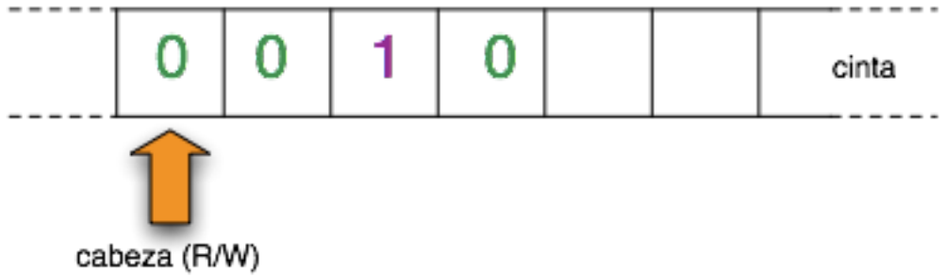
1. **Leer** el bit al que apunta la cabeza.
2. **Escribir** al bit al que apunta la cabeza.



Máquina de Turing (2)

- Operaciones posibles:

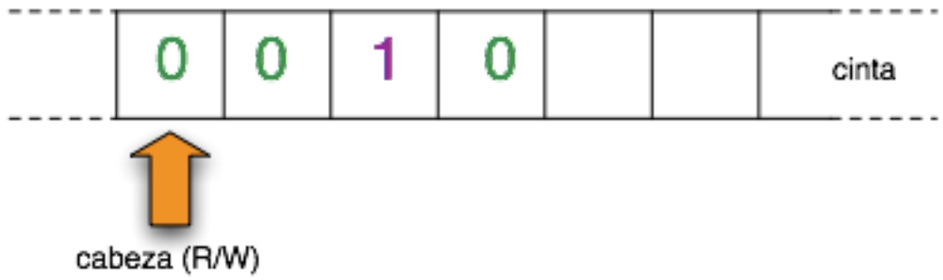
1. **Leer** el bit al que apunta la cabeza.
2. **Escribir** al bit al que apunta la cabeza.
3. **Mover** la cinta a la **derecha**.



Máquina de Turing (2)

- Operaciones posibles:

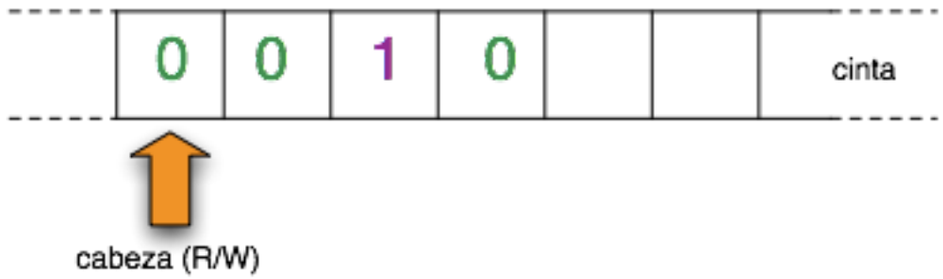
1. **Leer** el bit al que apunta la cabeza.
2. **Escribir** al bit al que apunta la cabeza.
3. **Mover** la cinta a la **derecha**.
4. **Mover** la cinta a la **izquierda**.



Máquina de Turing (2)

- Operaciones posibles:

1. **Leer** el bit al que apunta la cabeza.
2. **Escribir** al bit al que apunta la cabeza.
3. **Mover** la cinta a la **derecha**.
4. **Mover** la cinta a la **izquierda**.
5. **Quedar** en el estado actual.

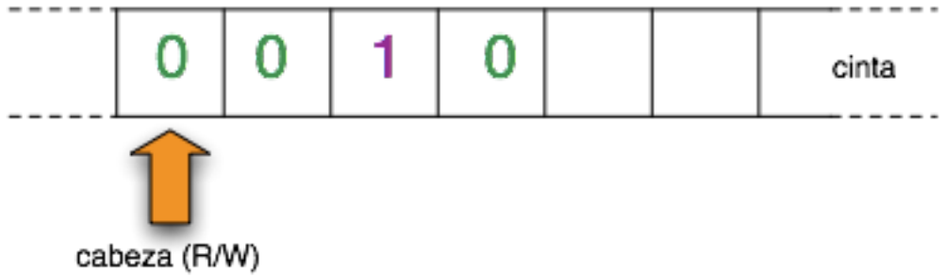


Máquina de Turing (2)

- Operaciones posibles:

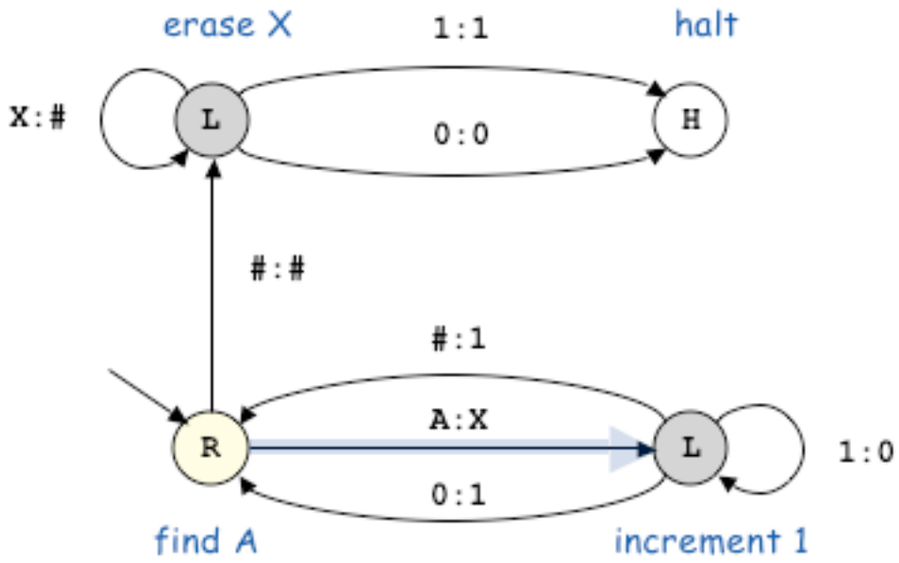
1. **Leer** el bit al que apunta la cabeza.
2. **Escribir** al bit al que apunta la cabeza.
3. **Mover** la cinta a la **derecha**.
4. **Mover** la cinta a la **izquierda**.
5. **Quedar** en el estado actual.

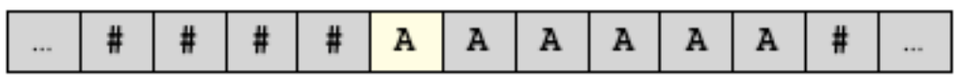
- Cada instrucción toma una unidad de tiempo. *Complejidad* ***constante.***



Ejemplo gráfico de máquina de Turing

http://introcs.cs.princeton.edu/java/74turing/

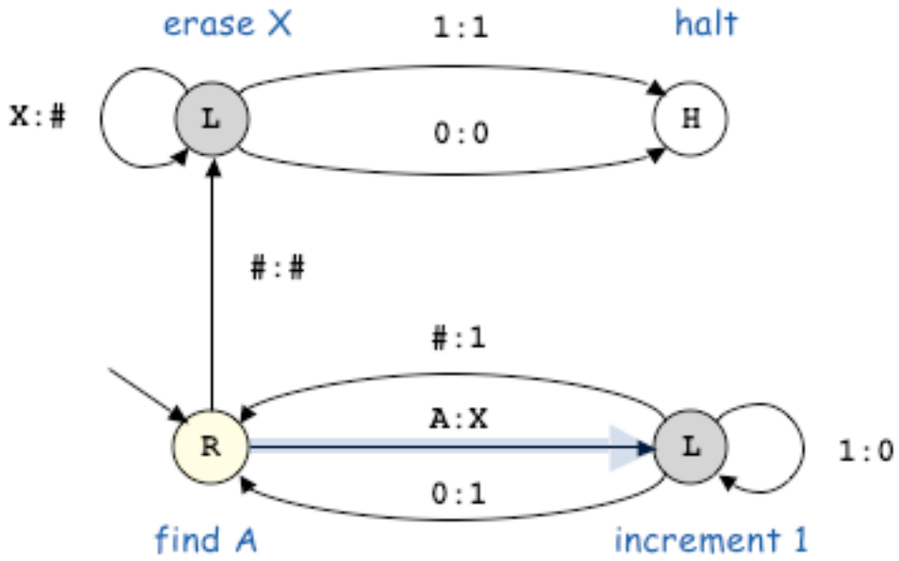


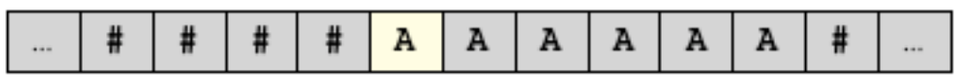


Estado actual en amarillo

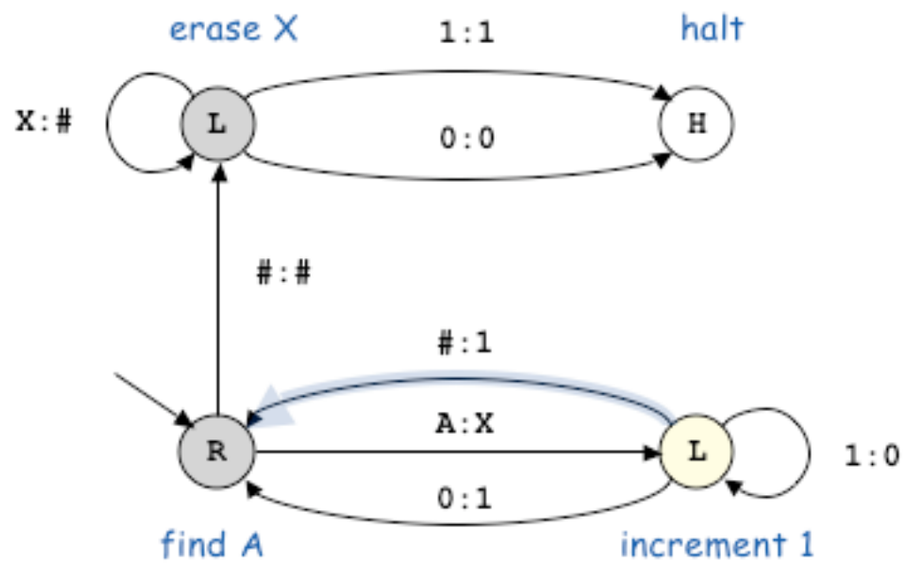
Ejemplo gráfico de máquina de Turing

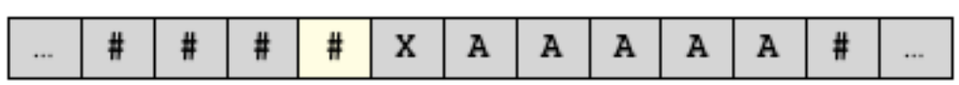
http://introcs.cs.princeton.edu/java/74turing/





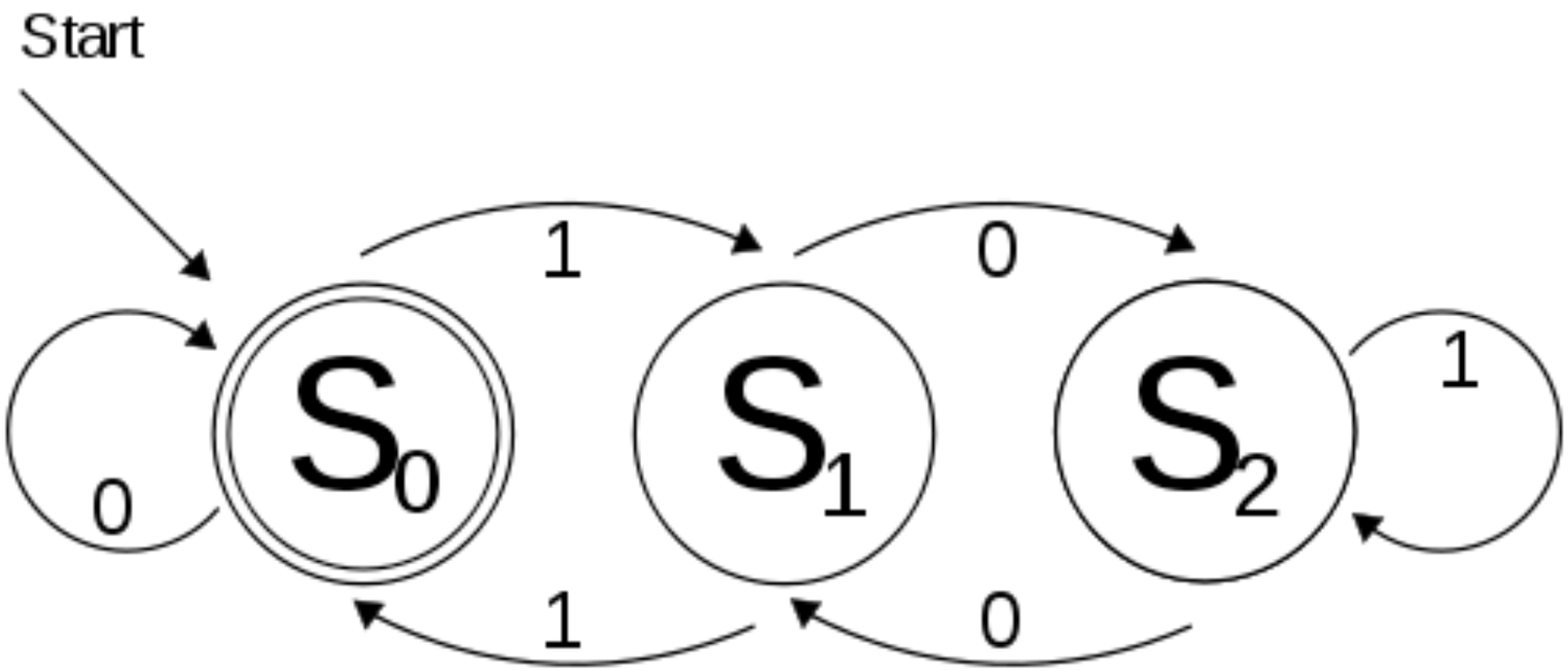
Estado actual en amarillo





Máquina de Turing que acepta múltiplos de 3 en binario

Máquina de Turing que acepta múltiplos de 3 en binario



Máquina de Turing (3)

Una máquina de Turing con una sola cinta puede ser definida como una 6-tupla $M = (Q, \Gamma, s, b, F, \delta)$, donde

- Q es un conjunto finito de estados.
- Γ es un conjunto finito de símbolos de cinta, el alfabeto de cinta.
- $s \in Q$ es el estado inicial.
- $b \in \Gamma$ es un símbolo denominado blanco, y es el único símbolo que se puede repetir un número infinito de veces.
- $F \subseteq Q$ es el conjunto de estados finales de aceptación.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ es una función parcial denominada función de transición, donde L es un movimiento a la izquierda y R es el movimiento a la derecha.

Modelos usados cuando usamos lenguajes de alto nivel

Máquina de acceso aleatorio (RAM) (1)

Máquina de acceso aleatorio (RAM) (1)

- Consta de un número finito de registros y una memoria finita.

Máquina de acceso aleatorio (RAM) (1)

- Consta de un número finito de registros y una memoria finita.
- La memoria se divide en palabras, cada palabra tiene una dirección entera entre 1 y n.

Máquina de acceso aleatorio (RAM) (1)

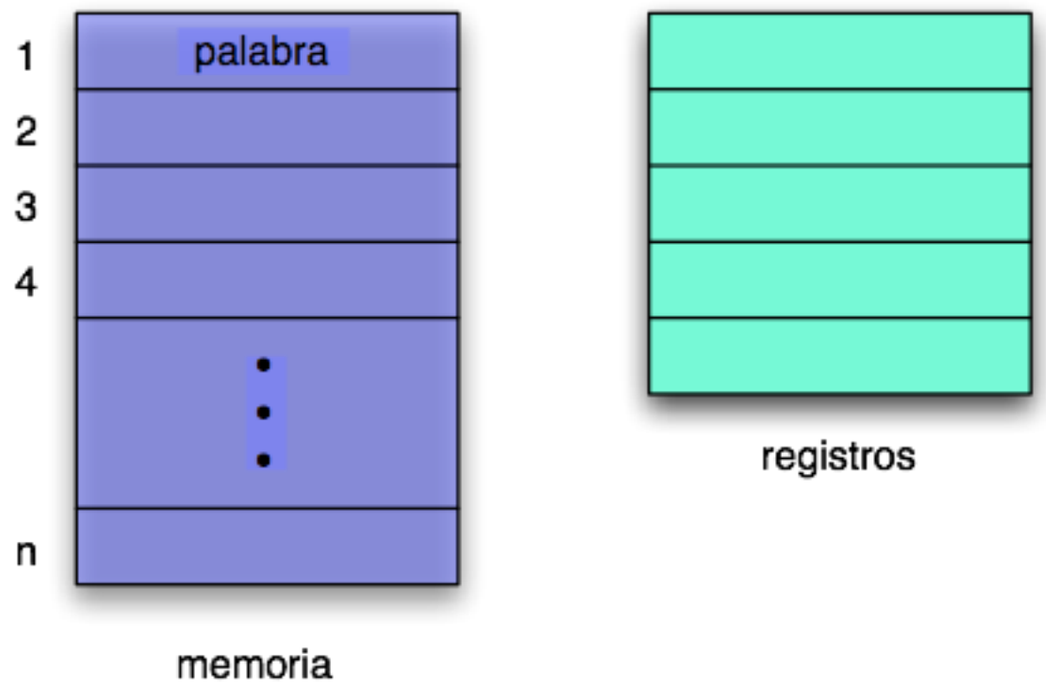
- Consta de un número finito de registros y una memoria finita.
- La memoria se divide en palabras, cada palabra tiene una dirección entera entre 1 y n.
- El contenido de cada palabra puede ser un entero o un número real (tipos de datos)

Máquina de acceso aleatorio (RAM) (1)

- Consta de un número finito de registros y una memoria finita.
- La memoria se divide en palabras, cada palabra tiene una dirección entera entre 1 y n.
- El contenido de cada palabra puede ser un entero o un número real (tipos de datos)
- Estos valores pueden representar datos o instrucciones.

Máquina de acceso aleatorio (RAM) (1)

- Consta de un número finito de registros y una memoria finita.
- La memoria se divide en palabras, cada palabra tiene una dirección entera entre 1 y n.
- El contenido de cada palabra puede ser un entero o un número real (tipos de datos)
- Estos valores pueden representar datos o instrucciones.



Máquina de acceso aleatorio (RAM) (2)

Máquina de acceso aleatorio (RAM) (2)

- Es posible hacer operaciones con las direcciones.

Máquina de acceso aleatorio (RAM) (2)

- Es posible hacer operaciones con las direcciones.
- Buscar o guardar elementos de la memoria significan buscarlos en la dirección actual del registro.

Máquina de acceso aleatorio (RAM) (2)

- Es posible hacer operaciones con las direcciones.
- Buscar o guardar elementos de la memoria significan buscarlos en la dirección actual del registro.
- Las operaciones de posicionamiento y almacenamiento toman el mismo tiempo, sin importar el valor que se encuentre almacenado en la memoria (*complejidad constante*)

Máquina de acceso aleatorio (RAM) (2)

- Es posible hacer operaciones con las direcciones.
- Buscar o guardar elementos de la memoria significan buscarlos en la dirección actual del registro.
- Las operaciones de posicionamiento y almacenamiento toman el mismo tiempo, sin importar el valor que se encuentre almacenado en la memoria (*complejidad constante*)
- operaciones aritméticas: suma, resta, multiplicación, división

Máquina de acceso aleatorio (RAM) (2)

- Es posible hacer operaciones con las direcciones.
- Buscar o guardar elementos de la memoria significan buscarlos en la dirección actual del registro.
- Las operaciones de posicionamiento y almacenamiento toman el mismo tiempo, sin importar el valor que se encuentre almacenado en la memoria (*complejidad constante*)
- operaciones aritméticas: suma, resta, multiplicación, división
- operaciones con datos: cargar, guardar ,copiar

Máquina de acceso aleatorio (RAM) (2)

- Es posible hacer operaciones con las direcciones.
- Buscar o guardar elementos de la memoria significan buscarlos en la dirección actual del registro.
- Las operaciones de posicionamiento y almacenamiento toman el mismo tiempo, sin importar el valor que se encuentre almacenado en la memoria (*complejidad constante*)
  - operaciones aritméticas: suma, resta, multiplicación, división
  - operaciones con datos: cargar, guardar ,copiar
  - operaciones de control: condicionales, llamadas a funciones, regreso de valores.

Máquina de acceso aleatorio (RAM) (2)

- Es posible hacer operaciones con las direcciones.
- Buscar o guardar elementos de la memoria significan buscarlos en la dirección actual del registro.
- Las operaciones de posicionamiento y almacenamiento toman el mismo tiempo, sin importar el valor que se encuentre almacenado en la memoria (*complejidad constante*)
- operaciones aritméticas: suma, resta, multiplicación, división
- operaciones con datos: cargar, guardar ,copiar
- operaciones de control: condicionales, llamadas a funciones, regreso de valores.
- Este modelo es el más utilizado.

Modelos de computación (2)

Modelos de computación (2)

- No simulan la jerarquía de memoria (caché o memoria virtual)

Modelos de computación (2)

- No simulan la jerarquía de memoria (caché o memoria virtual)
- Este tipo de modelos son excelentes predictores del desempeño de un algoritmo en computadoras reales.

Modelos de computación (2)

- No simulan la jerarquía de memoria (caché o memoria virtual)
- Este tipo de modelos son excelentes predictores del desempeño de un algoritmo en computadoras reales.
- Mientras más detallado sea el modelo, más complejo se vuelve el análisis.

Modelos de computación (2)

- No simulan la jerarquía de memoria (caché o memoria virtual)
- Este tipo de modelos son excelentes predictores del desempeño de un algoritmo en computadoras reales.
- Mientras más detallado sea el modelo, más complejo se vuelve el análisis.
- No abusar del modelo.

Nota sobre la implementación

Nota sobre la implementación

- El análisis de algoritmos inicia con la definición de las **operaciones abstractas** necesarias para la implementación.

Nota sobre la implementación

- El análisis de algoritmos inicia con la definición de las **operaciones abstractas** necesarias para la implementación.

- Separar el algoritmo de su implementación. Al implementar (sobre todo en sistemas grandes) se necesitará agregar mecanismos de detección de errores, deberán ser modulares, simples, legibles, con interfaces a otras partes del sistema, etc.

Nota sobre la implementación

- El análisis de algoritmos inicia con la definición de las **operaciones abstractas** necesarias para la implementación.

- Separar el algoritmo de su implementación. Al implementar (sobre todo en sistemas grandes) se necesitará agregar mecanismos de detección de errores, deberán ser modulares, simples, legibles, con interfaces a otras partes del sistema, etc.

- Queremos a pesar de esto los algoritmos más simples y con mejor desempeño.

Nota sobre la implementación

- El análisis de algoritmos inicia con la definición de las **operaciones abstractas** necesarias para la implementación.

- Separar el algoritmo de su implementación. Al implementar (sobre todo en sistemas grandes) se necesitará agregar mecanismos de detección de errores, deberán ser modulares, simples, legibles, con interfaces a otras partes del sistema, etc.

- Queremos a pesar de esto los algoritmos más simples y con mejor desempeño.

- La primera estrategia para aprender sobre el desempeño es realizar un **análisis empírico.**

Errores más comunes al seleccionar un algoritmo

Errores más comunes al seleccionar un algoritmo

1. **No tomar en cuenta su desempeño**: los algoritmos ***más rápidos*** son generalmente ***más complicados***. No tener miedo, a veces un mejor algoritmo solo involucra el cambio de unas cuantas líneas de código !

Errores más comunes al seleccionar un algoritmo

1. **No tomar en cuenta su desempeño**: los algoritmos ***más rápidos*** son generalmente ***más complicados***. No tener miedo, a veces un mejor algoritmo solo involucra el cambio de unas cuantas líneas de código !
2. **Tomar demasiado en cuenta su desempeño:** si un programa toma solo unos microsegundos...¿vale la pena mejorarlo por solo un pequeño factor?

Errores más comunes al seleccionar un algoritmo

1. **No tomar en cuenta su desempeño**: los algoritmos ***más rápidos*** son generalmente ***más complicados***. No tener miedo, a veces un mejor algoritmo solo involucra el cambio de unas cuantas líneas de código !
2. **Tomar demasiado en cuenta su desempeño:** si un programa toma solo unos microsegundos...¿vale la pena mejorarlo por solo un pequeño factor?

Errores más comunes al seleccionar un algoritmo

1. **No tomar en cuenta su desempeño**: los algoritmos ***más rápidos*** son generalmente ***más complicados***. No tener miedo, a veces un mejor algoritmo solo involucra el cambio de unas cuantas líneas de código !
2. **Tomar demasiado en cuenta su desempeño:** si un programa toma solo unos microsegundos...¿vale la pena mejorarlo por solo un pequeño factor?

Hay que considerar entonces

Errores más comunes al seleccionar un algoritmo

1. **No tomar en cuenta su desempeño**: los algoritmos ***más rápidos*** son generalmente ***más complicados***. No tener miedo, a veces un mejor algoritmo solo involucra el cambio de unas cuantas líneas de código !
2. **Tomar demasiado en cuenta su desempeño:** si un programa toma solo unos microsegundos...¿vale la pena mejorarlo por solo un pequeño factor?

Hay que considerar entonces

- aplicación,

Errores más comunes al seleccionar un algoritmo

1. **No tomar en cuenta su desempeño**: los algoritmos ***más rápidos*** son generalmente ***más complicados***. No tener miedo, a veces un mejor algoritmo solo involucra el cambio de unas cuantas líneas de código !
2. **Tomar demasiado en cuenta su desempeño:** si un programa toma solo unos microsegundos...¿vale la pena mejorarlo por solo un pequeño factor?

Hay que considerar entonces

- aplicación,
- veces que el programa será utilizado,

Errores más comunes al seleccionar un algoritmo

1. **No tomar en cuenta su desempeño**: los algoritmos ***más rápidos*** son generalmente ***más complicados***. No tener miedo, a veces un mejor algoritmo solo involucra el cambio de unas cuantas líneas de código !
2. **Tomar demasiado en cuenta su desempeño:** si un programa toma solo unos microsegundos...¿vale la pena mejorarlo por solo un pequeño factor?

Hay que considerar entonces

- aplicación,
- veces que el programa será utilizado,
- tiempo total que tomará al programador hacer las mejoras.

Análisis matemático de algoritmos

Análisis matemático de algoritmos

- Para **comparar** diferentes algoritmos para la misma tarea.

Análisis matemático de algoritmos

- Para **comparar** diferentes algoritmos para la misma tarea.
- Para **predecir** el desempeño en un nuevo ambiente.

Análisis matemático de algoritmos

- Para **comparar** diferentes algoritmos para la misma tarea.
- Para **predecir** el desempeño en un nuevo ambiente.
- Para **establecer** los parámetros de un algoritmo.

Análisis matemático de algoritmos

- Para **comparar** diferentes algoritmos para la misma tarea.
- Para **predecir** el desempeño en un nuevo ambiente.
- Para **establecer** los parámetros de un algoritmo.

Análisis matemático de algoritmos

- Para **comparar** diferentes algoritmos para la misma tarea.
- Para **predecir** el desempeño en un nuevo ambiente.
- Para **establecer** los parámetros de un algoritmo.

- El análisis **empírico** puede ser suficiente para algunas tareas, pero el análisis **matemático** es generalmente **más informativo** y **menos costoso**.

Análisis de algoritmos

Análisis de algoritmos

1. Identificar las **operaciones abstractas** en las que se basa el algoritmo para separar el **análisis** de la **implementación**.

Análisis de algoritmos

1. Identificar las **operaciones abstractas** en las que se basa el algoritmo para separar el **análisis** de la **implementación**.

   ➡ Por ejemplo, ¿cuántas veces se debe ejecutar el fragmento de código `i=a[i]` en la implementación del algoritmo union-búsqueda.

Análisis de algoritmos

1. Identificar las **operaciones abstractas** en las que se basa el algoritmo para separar el **análisis** de la **implementación**.

   ➡ Por ejemplo, ¿cuántas veces se debe ejecutar el fragmento de código `i=a[i]` en la implementación del algoritmo union-búsqueda.

   ➡ ¿Cuántos nanosegundos se necesitan para ejecutar esta porción particular del código?

Análisis de algoritmos

1. Identificar las **operaciones abstractas** en las que se basa el algoritmo para separar el **análisis** de la **implementación**.

   ➡ Por ejemplo, ¿cuántas veces se debe ejecutar el fragmento de código `i=a[i]` en la implementación del algoritmo union-búsqueda.

   ➡ ¿Cuántos nanosegundos se necesitan para ejecutar esta porción particular del código?

- propiedades del algoritmo vs. propiedades de una computadora.

Análisis de algoritmos

1. Identificar las **operaciones abstractas** en las que se basa el algoritmo para separar el **análisis** de la **implementación**.

   ➡ Por ejemplo, ¿cuántas veces se debe ejecutar el fragmento de código `i=a[i]` en la implementación del algoritmo union-búsqueda.

   ➡ ¿Cuántos nanosegundos se necesitan para ejecutar esta porción particular del código?

- propiedades del algoritmo vs. propiedades de una computadora.

Análisis de algoritmos

1. Identificar las **operaciones abstractas** en las que se basa el algoritmo para separar el **análisis** de la **implementación**.

   ➡ Por ejemplo, ¿cuántas veces se debe ejecutar el fragmento de código `i=a[i]` en la implementación del algoritmo union-búsqueda.

   ➡ ¿Cuántos nanosegundos se necesitan para ejecutar esta porción particular del código?

- propiedades del algoritmo vs. propiedades de una computadora.

- determinar exactamente el tiempo de cálculo es una tarea muy compleja y no es útil más que para condiciones específicas. Una idea **aproximada** del desempeño del algoritmo en condiciones generales es más útil.

Análisis de algoritmos

- Es necesario estudiar los datos de entrada que puedan presentarse al algoritmo.
    - entrada aleatoria
    - caso promedio
    - peor caso

Funciones de crecimiento

Funciones de crecimiento

- La mayoría de los algoritmos tienen un ***parámetro primitivo N*** que afecta el tiempo de cálculo significativamente.

Funciones de crecimiento

- La mayoría de los algoritmos tienen un ***parámetro primitivo N*** que afecta el tiempo de cálculo significativamente.

    - ejemplos: el grado de un polinomio, el tamaño del archivo que se va a ordenar o buscar, el número de caracteres en una cadena, o alguna otra medida abstracta relacionada con el tamaño del problema considerado.

Funciones de crecimiento

- La mayoría de los algoritmos tienen un ***parámetro primitivo N*** que afecta el tiempo de cálculo significativamente.
    - ejemplos: el grado de un polinomio, el tamaño del archivo que se va a ordenar o buscar, el número de caracteres en una cadena, o alguna otra medida abstracta relacionada con el tamaño del problema considerado.
    - la meta es expresar los requerimientos de recursos de los programas (principalmente el tiempo de cálculo) en términos de ***N***, usando fórmulas matemáticas de la manera más simple y precisa posible.

Complejidad

Complejidad

- Para poder caracterizar implementaciones, la primera cosa que hacer es **aislar la o las “variables” que permiten caracterizar la complejidad del problema a resolver,** con el fin de expresar la complejidad de las operaciones.

Complejidad

- Para poder caracterizar implementaciones, la primera cosa que hacer es **aislar la o las “variables” que permiten caracterizar la complejidad del problema a resolver,** con el fin de expresar la complejidad de las operaciones.

- Típicamente las variables son el número de elementos dentro del contenedor.

Complejidad

- Para poder caracterizar implementaciones, la primera cosa que hacer es **aislar la o las “variables” que permiten caracterizar la complejidad del problema a resolver,** con el fin de expresar la complejidad de las operaciones.

- Típicamente las variables son el número de elementos dentro del contenedor.

- Ejemplos:
    - tamaño de un arreglo.

Complejidad - Funciones de crecimiento (1)

- Los algoritmos que estudiaremos en este curso tienen típicamente tiempos de cálculo proporcional a una de las siguientes funciones:

Complejidad - Funciones de crecimiento (1)

- Los algoritmos que estudiaremos en este curso tienen típicamente tiempos de cálculo proporcional a una de las siguientes funciones:

$$1$$

Complejidad - Funciones de crecimiento (1)

- Los algoritmos que estudiaremos en este curso tienen típicamente tiempos de cálculo proporcional a una de las siguientes funciones:

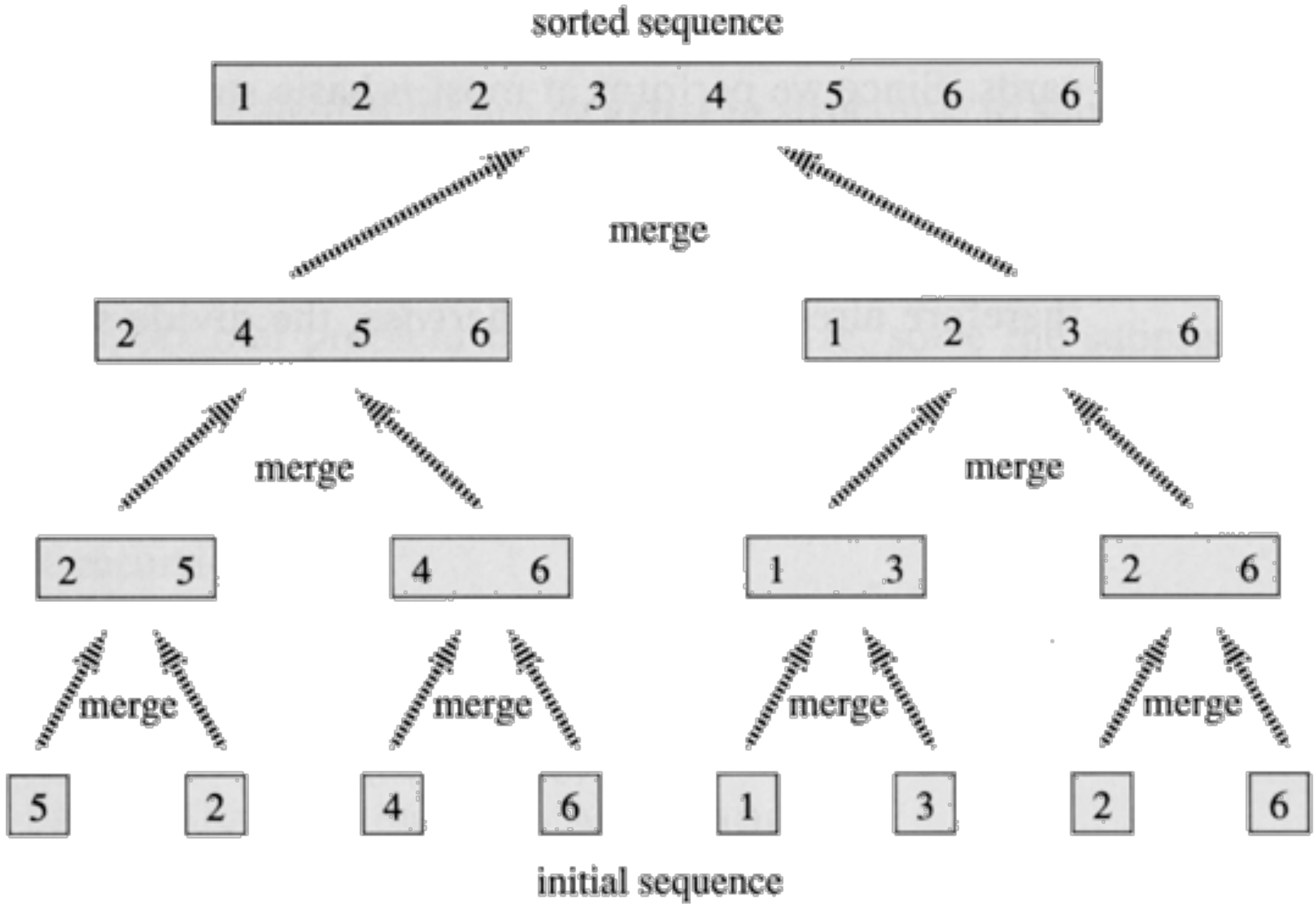
1	La mayoría de las instrucciones de un programa se ejecutan una vez o unas pocas veces. Complejidad **constante**. Independiente al tamaño del problema.
$logN$	Si el ***tiempo de ejecución*** es ***logarítmico***, el programa se vuelve proporcionalmente más lento mientras N crece. Ocurre generalmente en problemas grandes que se dividen en una serie de problemas más pequeños. Típicamente en los problemas de dicotomía donde el problema es compartido por varias instancias y solo una es procesada: búsqueda de mi amigo en el directorio.

Complejidad - Funciones de crecimiento (2)

Complejidad - Funciones de crecimiento (2)

N	Cuando el tiempo es ***lineal***, se da generalmente el caso que se hace un procesamiento por cada elemento que entra: hacer una impresión en la pantalla de cada uno de los elementos, etc...
$NlogN$	El tiempo ***N-logarítmico***. Problemas donde cada iteración se divide en sub-problemas, donde el procesamiento es lineal: los problemas clásicos en este caso son los conocidos algoritmos de ordenamiento: quicksort y mergesort.
N^2	El tiempo de ejecución es de un orden ***cuadrático***. Este es el caso para los algoritmos que tienen cíclos anidados, donde el ciclo interno se realiza sobre todos los datos: bubble sort, multiplicacion de matriz cuadrada por vector.

Ejemplo de $NLog_2(N)$

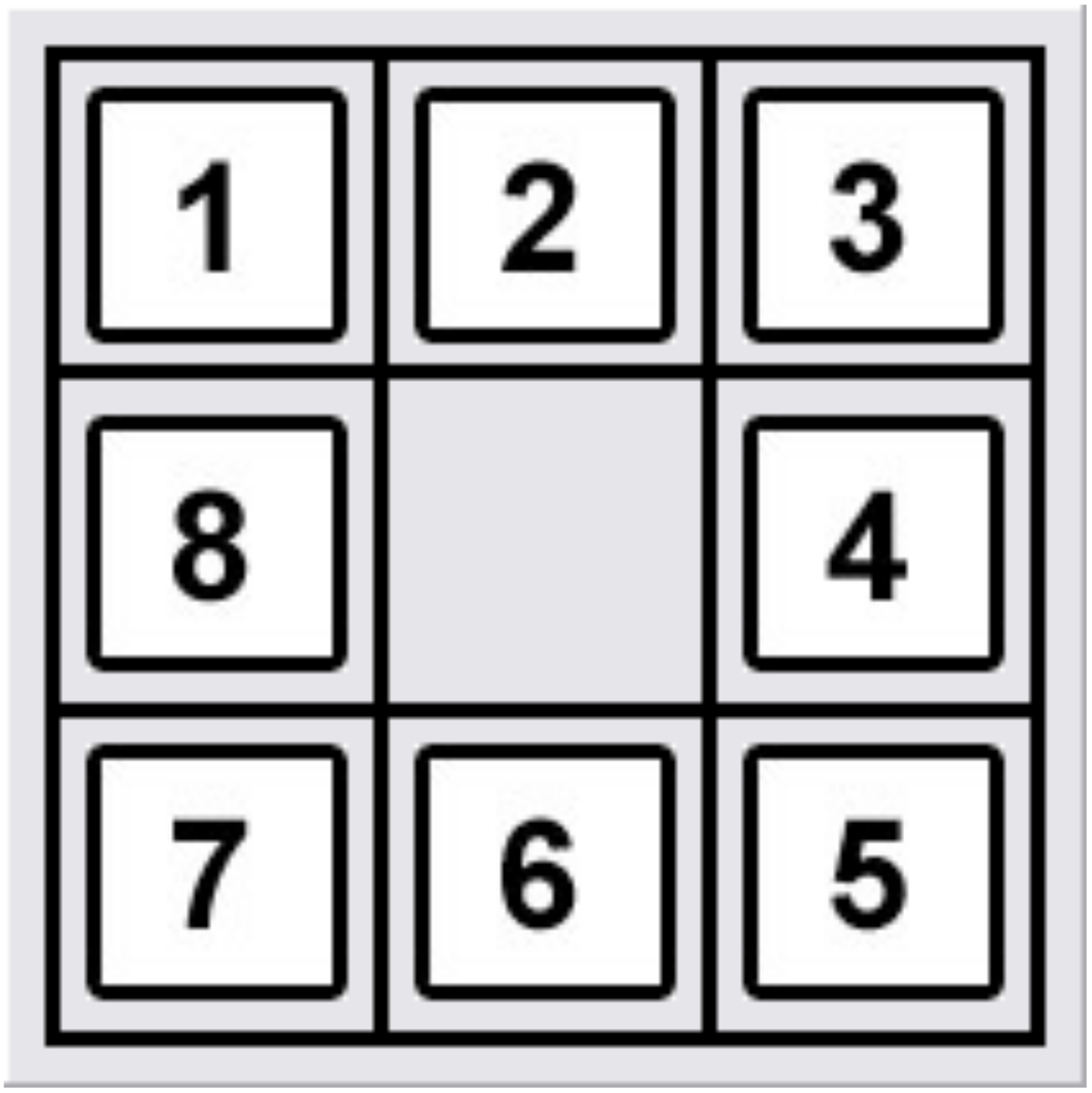


Complejidad - Funciones de crecimiento (3)

Complejidad - Funciones de crecimiento (3)

N^3	Tiempo de ejecución de orden ***cúbico***, implica generalmente tres ciclos anidados, por ejemplo, la multiplicaciones de matrices cuadradas.
2^N	Los algoritmos con funciones de crecimiento ***exponencial***, son generalmente ingenuos y a partir de ciertos tamaños, no demasiado altos, se vuelven ya inutilizables: ejemplo, el problema de 8-puzzle.

Problema del 8 puzzle



Complejidad - Funciones de crecimiento (4)