Programación Avanzada

Introducción al Análisis de Algoritmos

Referencias útiles

- **Introduction to Algorithms**. Cormen T.H., Leiserson, C.E., Rivest R.L. and Stein, C. The MIT Press
- **Algorithm Design.** Kleinberg J. and Tardos E. Addison Wesley
- **Computational Geometry, Algorithms and Applications.** de Berg M. , Cheong O., van Kreveld M. and Overmars M. Springer.
- **Algorithms Course Materials**. Erickson J. http://www.cs.uiuc.edu/~jeffe/teaching/algorithms
- **LaTeX algorithms package, Cormen et. al pseudo-code style** http://algorithms.berlios.de/, http://www.cs.dartmouth.edu/~thc/clrscode/

Algoritmo

¿Qué es un algoritmo?

Un algoritmo es una secuencia explícita, precisa, no ambigua de instrucciones elementales que toma un valor o conjunto de valores como entrada y produce un valor o conjunto de valores como salida.

- Todo algoritmo debe tener las siguientes 5 características principales:
    - finito
    - bien definido
    - entradas bien definidas
    - salidas bien definidas
    - factible

```
ElefantSong(n)
1  for i ← 1 to n
2       if i = 1
3            Sing "Un elefante se columpiaba sobre la tela de una araña"
4            Sing "como veia que resistia fue a llamar a otro elefante"
5       else
6            Sing "i elefantes se columpiaban sobre la tela de una araña"
7            Sing "como veian que resistian fueron a llamar a otro elefante"
```

- La palabra algoritmos no viene de la raíz griega αλγος (dolor).
- Viene del matemático persa del siglo IX Abū 'Abd Allāh Muhammad ibn Mūsā al-Khwārizmī.
- De su tratado Al-Kitāb al-mukhtasar fīhīsāb al-ğabr wa'l-muqābala deriva la palabra moderna álgebra.
- Hasta recientemente, la palabra algoritmo se refería exclusivamente a métodos para cálculos numéricos realizados con papel y lápiz.
- Las personas entrenadas para hacer estos métodos se llamaban: computadores.

Ejemplo

- Multiplicación del campesino (en ruso):
  - Una variante de este algoritmo fue copiada en el Papiro de Rhind (http://mathworld.wolfram.com/RhindPapyrus.html) por el escriba egipcio Ahmes alrededor de 1650 A.C. de un documento que en esa época tenía como 350 años de antiguedad.
  - Éste era el método de cálculo más común en Europa antes de que Fibonacci introdujera los números Arábigos.
  - Se utilizaba aún en Rusia, junto con el calendario Juliano a principios del siglo XX.
  - Éste algoritmo se utilizó en las primeras computadoras digitales que no implementaban en hardware la multiplicación de enteros.

```
PEASANTMULTIPLY(x, y):
  prod ← 0
  while x > 0
      if x is odd
          prod ← prod + y
      x ← ⌊x/2⌋
      y ← y + y
  return p
```

x	y	$prod$
		0
123	+456	= 456
61	+912	= 1368
30	~~1824~~	
15	+3648	= 5016
7	+7296	= 12312
3	+14592	= 26904
1	+29184	= 56088

- Este algoritmo convierte la tarea de multiplicar números grandes en una secuencia de operaciones:
  - determinar la paridad
  - suma
  - duplicar un número
  - dividir un número entre dos y redondear.
- Lo exacto (correcto) del algoritmo viene de la recursión que se mantiene para cualquier x,y enteros no negativos.

$$x \cdot y = \begin{cases} 0 & \text{if } x = 0 \\ \lfloor x/2 \rfloor \cdot (y+y) & \text{if } x \text{ is even} \\ \lfloor x/2 \rfloor \cdot (y+y) + y & \text{if } x \text{ is odd} \end{cases}$$

Ejemplo

- Un mal ejemplo:

BecomeAMultimillionarieAndNeverPayTaxes

1 Get a million dollars
2 Don't pay taxes
3 If you get caught
4 Say "I forgot"

- Ambiguo y no trivial. Demasiado vago para ser considerado algoritmo.
- Reduce el problema de ser millonario y no pagar impuestos al problema "mas sencillo" de ganar un millón de dólares.
- Si sabes resolver el problema más simple, una reducción te indica cómo resolver el problema más difícil.

¿Cómo escribir algoritmos?

- Los programas en computadora son las representaciones concretas de los algoritmos pero **los algoritmos no son programas**: no deben ser escritos en algún lenguaje de programación dado.
- La sintaxis de cada lenguaje no tiene importancia cuando se diseña o se analiza un algoritmo.
- Escribirlos en lenguaje natural (inglés, español ...) tampoco ayuda por la estructura propia de los algoritmos: condicionales, ciclos, recursiones, etc.
- Para expresar algoritmos utilizamos pseudo-código, que utiliza la estructura de los lenguajes formales de programación y las matemáticas además de un lenguaje natural.

Notación

- Utilizar palabras imperativas de los lenguajes de programación: if/then/else, while, for, repeat/until, case, return).
- Utilizar la notación de asignación de variables: variable←valor, Arreglo[índice], función(argumento), mayor>menor, etc.
- Usar notación matemática para operaciones matemáticas estándar: $\sqrt{x}$, a^b y π en lugar de sqrt(x), power(a,b) y PI.
- No utilizar $\forall$ para representar un ciclo for.
- Evitar notación matemática si en lenguaje natural es más claro: "Insert a in X" es preferible a INSERT(X,a) o X←X∪{a}.
- Identar cuidadosa y consistentemente: la estructura de bloques debe notarse especialmente en ciclos y condicionales anidados.
- Evitar usar "azucar" sintáctica: paréntesis, llaves, diferentes tipos de fuentes.
- Cada instrucción debe aparecer en una línea y debe contener exactamente un elemento estructural (for, while, if)
- Utilizar nombres cortos de variables, no usar pronombres.

Exactitud (correctness)

- Probar que los algoritmos son correctos para **todas** las entradas posibles, no solo para algunos casos de prueba.
- Se probará en general con inducción.

- Expresar el algoritmo como un problema formal, con estructuras de datos conocidas (números, arreglos, listas, gráficas, árboles ...)
- Considerar todas las suposiciones necesarias para las entradas del algoritmo (p.e. que la entrada n sean números enteros positivos)

Tiempo de cálculo

- ¿Cuánto tiempo toma cantar la canción de los elefantes?
- Función de la entrada n.
- También depende de qué tan rápido puedas cantar.
- Algunos cantantes tomarán 20 segundos en cantar un verso, otros tomarán 10 segundos. (lenguaje de programación)
- Depende de la tecnología utilizada: dictar la canción a un telegrafista, bajar un mp3 de internet o copiarla a otro directorio en una computadora.

- Lo que nos interesa es cómo crece el tiempo de canto cuando aumenta el número de elefantes de entrada n.
- Cantar la canción de los elefantes con 2n elefantes de entrada, ¿cómo modifica el tiempo de canto?
  - no importa el tipo de tecnología o lenguaje de programación utilizado.
- ¿Cómo se escribe en notación asintótica?
  - $\Theta(n)$

Tiempo de cálculo

- ¿Cuánto tiempo toma cantar la canción de los elefantes?

```
ElefantSong(n)
1  for i ← 1 to n
2       if i = 1
3            Sing "Un elefante se columpiaba sobre la tela de una araña"
4            Sing "como veia que resistia fue a llamar a otro elefante"
5       else
6            Sing "i elefantes se columpiaban sobre la tela de una araña"
7            Sing "como veian que resistian fueron a llamar a otro elefante"
```

- Podemos medir el tiempo de ejecución por el número de veces que el algoritmo ejecuta cierta instrucción o llega a un cierto lugar del código.
- Por ejemplo, la palabra elefante se canta 2 veces en cada verso de ElefantSong por lo que utiliza exactamente:
  - 2n elefantes

- ¿Cúanto tiempo toma cantar Alouette en función de n?

```
ALOUETTE(lapart[1...n])
1  Sing "Allouete, gentille alouette, alouette, je te plumerai"
2  for i ← 1 to n
3        Sing "Je te plumerai lapart[i]. Je te plumerai lapart[i]."
4        for j ← i − 1 to 1
5              Sing "Et lapart[j]!, et lapart[j]!"
6        Sing "Aaaaaaaaa"
7        Sing "Allouete, gentille alouette, alouette, je te plumerai"
```

- $\Theta(n^2)$
- Se menciona el nombre de una parte del pájaro $\sum_{i=1}^{n} 2i = n(n+1)$ veces.

Matrimonio Estable

- Grupo de demandantes (un trabajo, un internado, un novio/a).
- Grupo de ofertantes (empresas, hospitales, otro novia/o).
- Por simplicidad suponemos n demandantes y n ofertantes.
- Cada ofertante ofrece solo un empleo, internado, noviazgo...
- No hay empates en la lista de preferencias de los demandantes y ofertantes.
- Cada grupo tiene una lista de preferencias.
- El problema trata de asignar parejas con el criterio de estabilidad siguiente:

- Decimos que la asignación es inestable si hay dos demandantes α y β y dos ofertantes A y B tal que:

  - α se asigne a A y β se asigne a B.
  - α prefiere B sobre A y β prefiere A sobre B.

Matrimonio Estable

- Un algoritmo para resolver este problema es el Boston Pool Algorithm.
- El algoritmo hace ciclos hasta que cada posición se llene. En cada vuelta:

  - Algún ofertante ofrece una posición a su demandante preferido que no lo haya rechazado anteriormente.
  - Cada demandante acepta la posición de la mejor oferta que le hayan hecho hasta el momento de acuerdo a sus preferencias.

- Si tenemos 4 demandantes α, β, γ, δ y 4 ofertantes A, B, C, D con las siguientes preferencias:

α	β	γ	δ	A	B	C	D
A	A	B	D	δ	β	δ	γ
B	D	A	B	γ	δ	α	β
C	C	C	C	β	α	β	α
D	B	D	A	α	γ	γ	δ

El algoritmo procedería como sigue:

- A hace una oferta a δ.
- B hace una oferta a β.
- C hace una oferta a δ, quién rechaza la oferta de A.
- D hace una oferta a γ.
- A hace una oferta a γ, quién rechaza la oferta de D.
- D hace una oferta a β, quién rechaza la oferta de B.
- B hace una oferta a δ, quién rechaza la oferta de C.
- C hace una oferta a α.

Las asignaciones quedarían: $(\alpha,C),(\beta,D),(\gamma,A),(\delta,B)$ que verificando con fuerza bruta son asignaciones estables.

- ¿Cuántas vueltas se necesitan a lo más?
  - n^2 porque un ofertante hace oferta a lo más una vez a cada demandante.
- ¿Qué estructuras de datos podríamos utilizar para la implementación?
  - cada ofertante y demandante se pueden representar con un entero de 1 a n.
  - las listas de preferencias se puede representar como arreglos DemPref[1...n][1...n] y OferPref[1...n][1...n], donde DemPref[α][r] representa al r-ésimo ofertante en la lista del α-ésimo demandante.
  - El tiempo de cálculo, con estas estructuras se puede escribir como $O(n^2)$.
- ¿Por qué es correcto?
  - El algoritmo termina porque cada ofertante ofrece un puesto a un demantante a lo más una vez.
  - Cuando el algoritmo termina, todas las parejas han sido asignadas.
  - Imaginemos que en la asignación final el demandante α fue asignado a A pero prefiere B.
  - Como cada demandante toma la oferta que le gusta más quiere decir que B nunca le hizo una oferta.
  - B hizo una oferta a todos los demandantes que prefiere sobre α y por lo tanto no hay inestabilidad.

- La exactitud del algoritmo no depende del orden en que los ofertantes hagan sus ofertas.

- No importa qué ofertante sin asignar haga su oferta en qué ronda, el algoritmo siempre encuentra el mismo apareamiento.

- Sea α un demandante factible para A si hay un apareamiento estable que asigne al demandante α al hospital A.
- Sea best(A) el demandante factible de mayor preferencia en la lista de A.

Lemma 1. El algoritmo Boston Pool asigna best(A) a A, para cada ofertante A.

Prueba:

- en el apareamiento final a ningún ofertante se le asigna un demandante que prefiera sobre best(A) porque esto crearía inestabilidad.
- a un ofertante A solo se le puede asignar un demandante que esté más abajo en su lista de preferencias que best(A) si best(A) rechazó su oferta.
- Consideramos la primera vuelta donde una oferta del ofertante A es rechazada por su demandante preferido α=best(A).
- en esa ronda α tuvo una oferta de otro ofertante B que está más arriba en la lista de prefencias de α que A.
- B debe tener a α al menos tan alto como A porque estamos en la primera iteración, donde el ofertante es rechazado por su primera opción.
- B definió a α como su preferido.
- Si consideramos el apareamiento estable donde α se asigne a A. α prefiere a B sobre A. A B se le asigna un demandante β factible para B, que implica que B prefiere a α sobre β lo que lleva a inestabilidad, y hay una contradicción.

- Sea worst(α) el ofertante factible de menor preferencia en la lista de α.

Lemma 2. El algoritmo Boston Pool asigna a α a worst(α), para cada demandante α.

Prueba:

- supongamos que el algoritmo asigna a α al ofertante A, que por el Lemma 1 sabemos que α=best(A).
- Para probar el Lemma 2 tenemos que mostrar que A=worst(α).
- Consideremos un apareamiento factible diferente a α, sino a otro demandante β.
- A prefiere a α sobre β.
- Como es apareamiento estable α debe preferir a su pareja actual sobre A.
- Este argumento vale para cualquier emparejamiento estable, por lo que α prefiere cualquier emparejamiento estable sobre A, es decir, A=worst(α).

Análisis de algoritmos

- Estudio teórico del **desempeño** y utilización de recursos de programas computacionales.
- otros recursos como memoria, comunicaciones ...

¿En programación, qué es más importante que el desempeño?

- correctitud
- simplicidad
- mantenimiento
- costo del tiempo del programador
- estabilidad, robustez
- características, funcionalidad
- modularidad
- seguridad
- amigable al usuario

¿Por qué estudiar entonces el desempeño de algoritmos?

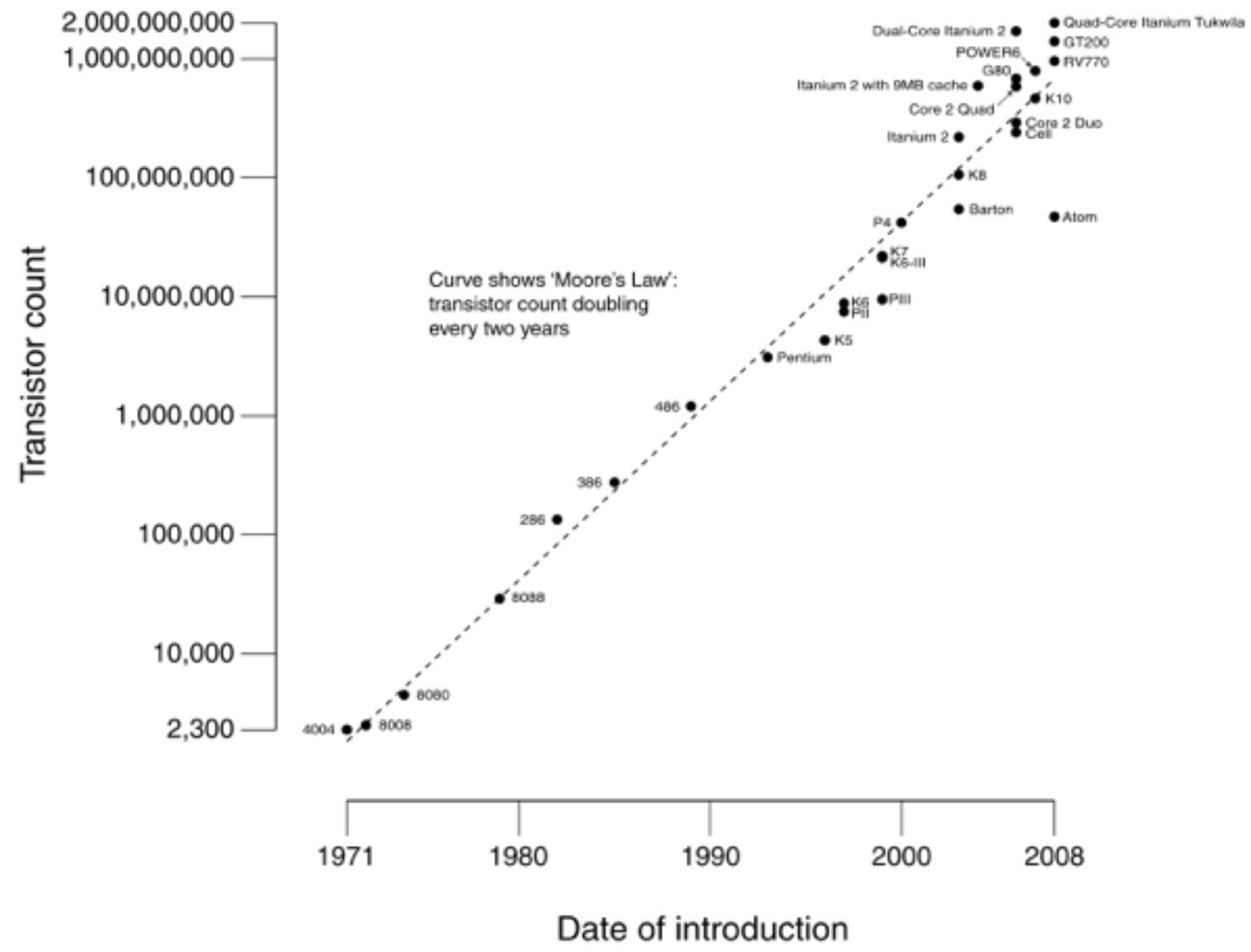
- El desempeño mide la línea entre lo **realizable** y lo no realizable.
- Dan un **lenguaje** para hablar sobre el comportamiento de los programas computacionales.
- Es como la "**moneda de cambio**" para las demás características.
- Base común para las demás características.
- Nos ayuda a comparar algoritmos que ejecutan la misma tarea.

“El poder de cómputo se duplica cada dos años”

- Gordon Moore, co-fundador de Intel

- El número de transistores que se pueden integrar fácilmente en un circuito integrado se dobla aproximadamente cada dos años.

- Se espera que este patrón continue hasta cerca del 2015, ¿y luego?



Algoritmos como tecnología

- Supongamos computadoras infinitamente rápidas y con memoria gratuita.
- ¿Nos sigue importando estudiar algorítmica?
- Cualquier método correcto bastaría para resolver el problema.
- Comparando dos algoritmos de ordenamiento: Insertion Sort y Merge Sort: