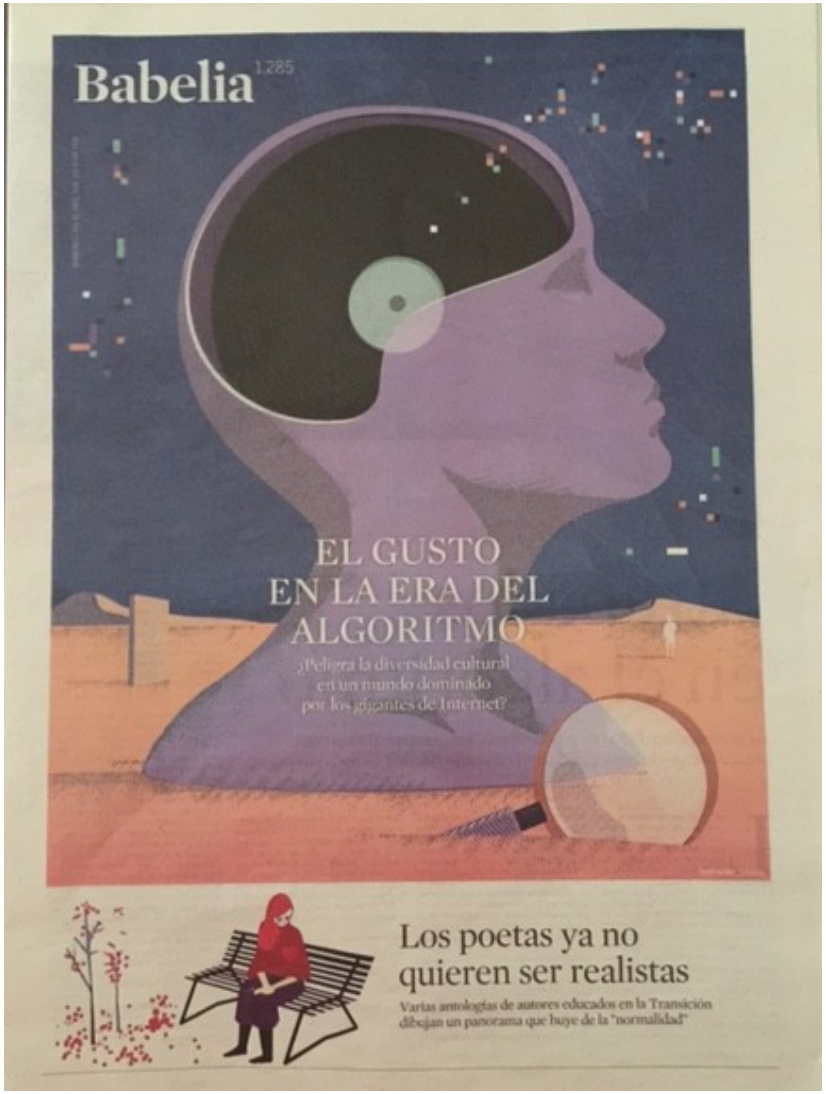
ALGORÍTMICA

Capítulo 1: La Eficiencia de los Algoritmos

Tema 1: Planteamiento general

- Ciencia de la Computación
- El concepto de algoritmo
- Elección de un algoritmo
- Problemas y casos
- Distintos tipos de casos

¿Es actual este tema?



- El gusto en la era del algoritmo
- La prescripción artificial en plataformas digitales como Amazon, Netflix, Google o Facebook eleva el riesgo de homogeneizar la identidad y los hábitos de consumo cultural
- El algoritmo, sostienen sus críticos, nos hace aburridos, previsibles, y empobrece nuestra curiosidad cultural
- De algún modo, Internet y las plataformas de streaming cultural han alumbrado un universo parecido al que describía Borges en La biblioteca de Babel
- Daniel Verdú, 8 de julio de 2016

¿Que es la Ciencia de la Computación?

- **La Ciencia de la Co**
  computadores.
- **La Ciencia de la**
  cómo e **rogram**
- **La Ciencia de la**
  aplicaciones de com

El estudio de cómo usar un computador/software es una parte de la Ciencia de la Computación igual que el manejo de un movil es una parte de la Ingenieria de Telecomunicaciones.

La Ciencia de la Computación es a los computadores lo mismo que la Astronomia es a los telescopios, la Biologia a los microscopios o la Química a las pipetas.

La Ciencia no trata sobre sus herramientas, sino sobre como usarlas y sobre lo que descubrimos cuando las usamos

La programación es una parte muy importante de la Ciencia de la Computación. Pero es una herramienta para implementar ideas y soluciones. Un programa es un medio para lograr un objetivo, pero no un objetivo.

¿Entonces que es la Ciencia de la Computación?

- Las definiciones anteriores:

  Aunque no necesariamente (completamente) equivocadas, son incompletas, y por tanto incorrectas

- Una definición mas precisa:

  La Ciencia de la Computación es el estudio de los Algoritmos, incluyendo sus propiedades, su hardware, sus aspectos lingüisticos (sintácticos y semánticos) y sus aplicaciones.

¿Que es un algoritmo?

- **Definición formal:**

  Una secuencia finita y ordenada de pasos, exentos de ambigüedad, tal que al llevarse a cabo con fidelidad, dará como resultado que se realice la tarea para la que se ha diseñado (se obtenga la solución del problema planteado) con recursos limitados y en tiempo finito.
- O, mas informalmente:
  - Un método por etapas para realizar alguna tarea.
- No hay una única definición de algoritmo (hay un “algoritmo para calcular el número π, pero …)
- Un programa es una serie de instrucciones ordenadas, codificadas en lenguaje de programación que expresa un algoritmo y que puede ser ejecutado en un computador.

Algoritmo y programa son conceptos diferentes.
Nuestro interés se centra en los algoritmos.

¿Que es un algoritmo?

- Esta definición es un tanto imprecisa
  - ¿Cuando un algoritmo estará completamente exento de ambigüedad?
  - ¿Como se define fidelidad?
  - Un gran número de años es un tiempo finito?
- Se puede formalizar mucho mas:
- **Definición de Bazaraa, Sheraly y Shetty (2006) :**

  Un método de solución (un algoritmo) para resolver un problema es un proceso iterativo que genera una sucesión de puntos, conforme a un conjunto dado de instrucciones, y un criterio de parada

¿Que es un algoritmo?

- **Definición de Knuth (1997)**
- Un método computacional es una cuaterna (Q, I, Ω, f) en la que Q es un conjunto que contiene a I y a Ω como subconjuntos y f es una función de Q en Q tal que

$$f(q) = q, \quad \forall q \in \Omega$$

- Y donde Q es el conjunto de los estados del cálculo, I el input, Ω el output y f la regla de cálculo que se esté aplicando. Cada input $x \in I$ define una sucesión computacional, x_0, x_1, x_2, …, como sigue:

$$x_0 = x \text{ y } x_{k+1} = f(x_k) \text{ cuando } k \geq 0$$

- Se dice que la sucesión computacional termina en k etapas si k es el menor entero para el que x_k está en Ω, y entonces en ese caso se dice que a partir de x se obtiene como output x_k.
- Algunas sucesiones computacionales pueden no terminar nunca.
- **Un algoritmo es un método computacional que termina en un número finito de etapas $\forall x \in I$.**

Ejemplo de la definición de Knuth

- El Algoritmo de Euclides (“Elementos de Euclides”, libro 7, proposiciones 1 y 2) calcula el máximo común divisor (mcd) de dos números.

Algoritmo E (**Algoritmo de Euclides**). Dados dos números enteros positivos m y n, encontrar su mcd, es decir, el mayor número entero positivo que divide exactamente tanto a m como a n.

E1. [Cálculo del resto] Dividir m por n. Sea r el resto ($0 \leq r < n$)
E2. [¿Es cero?] Si r = 0 el algoritmo termina. La respuesta es n.
E3. [Descenso] Tomar m = n, n = r y volver a la etapa E1.

- Sea Q el conjunto de todos los singletones (n), todos los pares ordenados (m,n) y todas las cuaternas ordenadas (m,n,r,1), (m,n,r,2) y (m,n,p,3), donde m, n y p son números enteros positivos y r es un entero no negativo. Sea I el subconjunto de todos los pares (m,n) y sea Ω el subconjunto de todos los singletones (n).

- ¿Cómo definir la función f?

Ejemplo de la definición de Knuth

- Si definimos f del siguiente modo,

  f((m,n)) = (m,n,0,1); f((n)) = (n)

  f((m,n,r,1)) = (m,n, resto de dividir m por n, 2)

  f((m,n,r,2)) = (n) si r = 0

  = (m,n,r,3) en otro caso

  f((m,n,p,3)) = (n,p,p,1)

- Entonces la correspondencia entre esta notación y el Algoritmo de Euclides es evidente.

D. Knuth: “The Art of Computer Programming. Fundamental Algorithms”. Addison Wesley 1997

Happening Now | August 01, 2016

Print

Donald E. Knuth Awarded SIAM's Highest Honor, Delivers the John von Neumann Lecture

Philadelphia, PA- The Society for Industrial and Applied Mathematics (SIAM) awards the 2016 John von Neumann Lecture prize to Donald E. Knuth of Stanford University for his transformative contributions to mathematics and computer science. Knuth delivered the associated prize lecture, "Satisfiability and Combinatorics," at the SIAM Annual Meeting in Boston, Massachusetts, on July 12. The highest honor awarded by SIAM, the flagship lecture recognizes outstanding and distinguished contributions to the field of applied mathematical sciences and the effective communication of these ideas to the community.

Pam Cook, University of Delaware; Donald Knuth, Stanford CSD; Jim Crowley, SIAM Executive Director

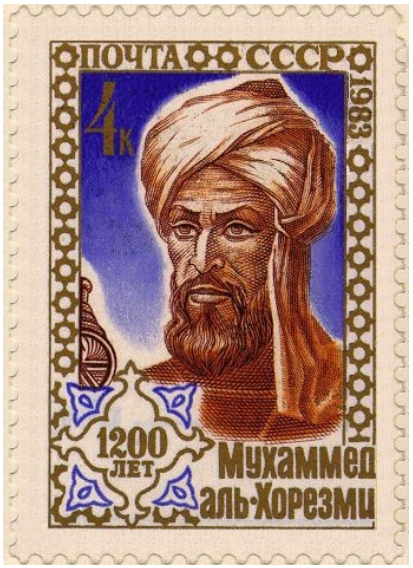
Knuth founded the field of analysis of algorithms and gave it a rigorous mathematical footing. His ongoing *The Art of Computer Programming* book series represents the definitive reference on algorithms and their analysis. His TeX and Metafont typesetting software has changed the face of mathematical publishing and benefited every mathematician. Knuth is recognized as a brilliant communicator at all levels, from his research monographs to his more popular books such as *Surreal Numbers* and his textbook *Concrete Mathematics*.

Knuth is Professor Emeritus of The Art of Computer Programming at Stanford University, where he supervised the PhD dissertations of approximately 28 students since becoming a professor in 1968. He is the author of numerous books, including four volumes (so far) of *The Art of Computer Programming*, five volumes of *Computers and Typesetting*, nine volumes of collected papers, and a non-technical book entitled *Bible Texts Illuminated*.

Knuth received his B.S. and M.S. in Mathematics at Case Institute of Technology (now Case Western Reserve University) in 1960 and received his PhD in Mathematics at California Institute of Technology in 1963.

¿De donde provienen?

- Etimología.
  - "algos" = pena, sufrimiento en griego.
  - "algor" = frio en latín.
  - "arithmos " = número en griego
  - degeneración de "logaritmo"
- Muhammad ibn Musa (Al-Khuwarizmi) fue un matemático persa (siglo 9 adc)



Su libro "Al-Jabr wa-al-Muqabilah" ("Algoritmi de Numero Indorum”) es la clave

Algo de historia

- 300 A. C.
  - Algoritmo de Euclides (mcd)
- 780-850 D.C.
  - Mohammed Ibn Musa al-Khwarizmi.
- 1424 D.C.
  - π = 3.1415926535897932…
- 1845.
  - Lamé: El algoritmo de Euclides hace a lo sumo $1 + \log_{\phi}(n\sqrt{5})$ etapas.
- 1900.
  - Décimo problema de Hilbert (Concepto de algoritmo)
- 1910
  - Pocklington: complejidad en bits
- 1920-1936
  - Post, Goëdel, Church, Turing, Von Neuman …
- 1965
  - Edmonds: Algoritmos polinomiales vs. Algoritmos exponenciales
- 1971
  - Teorema de Cook (SAT es NP-completo), Reducciones de Karp
- 1997
  - Deep Blue vence a Kasparov
- 20xx
  - $P \neq NP$
  - Lee Sedol pierde frente a AlphaGo.

“Algoritmos” en la vida diaria

- Usamos los “algoritmos” todos los días:
  - Cada vez que realizamos una tarea rutinaria (instrucciones para montar aparatos).
  - Comprando por internet
  - Indicaciones para llegar a sitios y recorrer lugares (PRoA)
  - Al conectarnos con el teléfono (contrataciones, averías, …)
  - Buscando y comparando información (donación de órganos, hostelería y turismo, …)
  - Controlando la gestión de los aeropuertos y del tráfico aéreo
  - En el área económica (banca, telebanca, pago ubicuo, …)
  - Diseño y construcción de proyectos de Ingeniería Civil
  - Ámbito hospitalario (datos, diagnósticos, cirugía de precisión, reconocimiento de imágenes y patrones …)
  - Casi cualquier tarea que imaginemos tiene detrás un algoritmo

Un algoritmo no es una receta

- Un algoritmo tiene cinco características primordiales
  a) **Finitud**: ha de terminar después de un tiempo acotado superiormente
  b) **Especificidad:** cada etapa debe estar precisamente definida; las acciones que hay que llevar a cabo deben estar rigurosamente especificadas para cada caso.
  c) **Input**: Un algoritmo tiene cero o mas inputs.
  d) **Output**: uno o mas outputs.
  e) **Efectividad**: todas las operaciones que hay que realizar deben ser tan básicas como para que se puedan hacer exactamente y en un periodo finito de tiempo (usando solo lápiz y papel, ¿tecnología?)

Tecnología

- Aquí admitiremos como Agente Tecnologico cualquier ente capaz de realizar las etapas que describe el algoritmo, es decir, capaz de ejecutar el algoritmo.
- Por tanto, puede ser una persona, una secuencia de ADN o, claro está, un computador
- En nuestro caso, generalmente, será un computador.
- La necesidad de que una implementación de un algoritmo acabe en un tiempo finito depende de la tecnología

MAÑANA DOMINGO

V DOMINGO

LA ARISTOCRACIA DEL VINO

Las familias bodegueras más importantes del mundo se reúnen en España

XL SEMANAL

TEATRO EN SERIE

Actores de éxito de televisión se estrenan en Mérida

La calculadora Stampede, que ha resuelto la cuestión de la bicoloración de las ternas pitagóricas. :: R. C.

La solución interminable

Tres matemáticos esclarecen un problema que parecía inextricable, aunque se necesitarían 10.000 millones de años para leer la respuesta

:: ANDRÉS JIMÉNEZ

A veces encontrar la solución es un problema. Que se lo digan si no a los tres matemáticos que han esclarecido lo que era un enigma inextricable. Tres sabios han resuelto el misterio de la bicoloración de las ternas pitagóricas. Hasta aquí todo bien. Lo malo es que leer la solución al problema llevaría al ser humano a invertir 10.000 millones de años. Algunos pensarán que mejor permanecer en la bendita ignorancia.

La cuestión de la bicoloración de las ternas pitagóricas tiene tela. Tanta que el problema lleva ocupando a la comunidad de matemáticos desde hace 35 años.

Ayer, en la conferencia científica internacional 'SAT 2016', que se celebraba en Burdeos, por fin tres informáticos británico-estadounidenses lograron dilucidar el problema, gracias a un algoritmo de concepción francesa y a la potencia de una supercalculadora. El problema viene ahora. Si se pusiera por escrito la solución, la enumeración equivaldría a todos los textos digitalizados de la Biblioteca del Congreso de EE UU, es decir, 200 teraoctetos. A la calculadora Stampede de la Universidad de Texas sólo le hicieron falta dos días para dar con el quid de la cuestión.

Para los aficionados a los pasatiempos sesudos, ahí va el problema, cuyo enunciado es sencillo, a decir de los matemáticos. «¿Es posible colorear cada entero positivo (1, 2, 3, 4, 5...) de azul o rojo de forma que en ninguna terna (grupo de tres elementos) de enteros a, b y c que responde a la famosa ecuación de Pitágoras a2+b2=c2 sean todos del mismo color?». Dicho de otra manera, si en la serie 3, 4 y 5, el 3 y el 5 son azules, entonces el 4 debe ser rojo, y así sucesivamente. «A ese enigma, el trío de informáticos respondió que es posible colorear así los enteros hasta 7.824, pero no más allá», asegura Laurent Simon, del Laboratorio de Investigación Informática, dependiente de la Universidad de Burdeos. Una respuesta inalcanzable para un ser humano, dado que existen más de «10 elevado ala potencia 2.300' maneras de colorear esos números hasta 7.825», precisa el investigador. Para llegar a esta conclusión, Marijn Heule, Oliver Kullmann y Victor Marek utilizaron diversas técnicas para reducir las posibilidades a un billón, y luego las trataron por «paquetes»

IDEAL, 9 de Julio de 2016

La Ciencia de la Computación aborda

1) **Propiedades formales y Matemáticas**

- Como diseñar algoritmos para resolver una gran variedad de problemas.
- Como determinar si los problemas son (eficientemente) computables, es decir, ¡si se pueden especificar por un algoritmo!
- Estudiar la conducta de los algoritmos para decidir si trabajan correctamente y con cuanta eficiencia.

2) **Hardware**

- El diseño y construcción de equipos capaces de ejecutar algoritmos.
- Los incesantes avances tecnológicos:
  - Computadores cada vez mas rápidos, redes, …
  - Computación paralela
  - Computación cuántica, molecular...

La Ciencia de la Computación aborda

3) **Aspectos lingüisticos**

- El diseño de lenguajes de programación y de la traducción a estos lenguajes de los algoritmos para que el hardware disponible pueda ejecutarlos.
- Programación funcional, Programación orientada a objetos, Programación visual, …

4) **Aplicaciones cada vez mas novedosas**

- Identificar nuevos problemas importantes para los computadores y del diseño del software que los resuelva.
- Los primeros computadores se usaron sobre todo para cálculos numéricos y tratamiento masivo de datos.
- Ahora, … se usan en negocios, grafismo, multimedia, domótica, Internet, WWW, … ¿qué será los siguiente?

¿Como y donde se estudia todo esto?

Algorítmica

- El estudio de los algoritmos incluye el de diferentes e importantes áreas de investigación y docencia, y se suele llamar **Algorítmica (también Teoría de Algoritmos)**
- La Algorítmica considera:
    1. La construcción
    2. La expresión
    3. La validación
    4. El análisis, y
    5. El test de los programas

1. La construcción de algoritmos

- El acto de crear un algoritmo es un arte que nunca debiera automatizarse.
- Pero, indudablemente, no se puede dominar si no se conocen a la perfección las técnicas de diseño de los algoritmos.
- El área de la construcción de algoritmos engloba el estudio de los métodos que, facilitando esa tarea, se han demostrado en la práctica mas útiles.

2. La expresión de algoritmos

- Los algoritmos han de tener una expresión lo mas clara y concisa posible.
- Como este un tema clave del estudio de los algoritmos, requerirá que se le dedique tanto esfuerzo como sea necesario, a fin de conseguir lo que se suele denominar un buen estilo.

3. Validación de algoritmos

- Cuando se ha construido el algoritmo, se hace necesario demostrar que calcula correctamente sobre inputs legales.
- Este proceso se conoce con el nombre de validación del algoritmo.
- La validación persigue asegurar que el algoritmo trabajará correctamente independientemente del lenguaje empleado, o la tecnología usada.
- Cuando se ha demostrado la validez del método, es cuando puede escribirse el programa, comenzando una segunda fase del trabajo (la verificación del programa).

4. Análisis de algoritmos

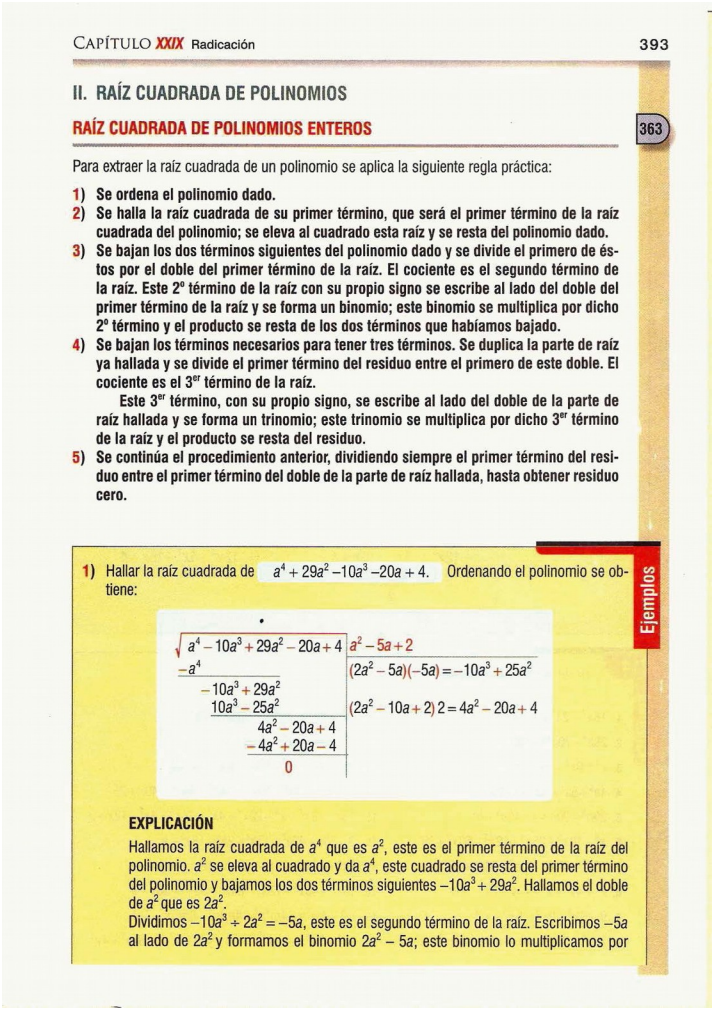
- El análisis de algoritmos se refiere al proceso de determinar cuanto tiempo de cálculo y cuanto almacenamiento requerirá un algoritmo.
- Nos permite hacer juicios cuantitativos sobre el valor de un algoritmo sobre otros.
- A menudo tendremos varios algoritmos para un mismo problema. Habrá que decidir cual es el mejor o cual es el que tenemos que escoger, según algún criterio pre-fijado, para resolverlo
- Puede permitirnos predecir si nuestro software necesitará algún requisito especial.

… y, 5. Test de los programas

- El test de un programa es la última fase que se lleva a cabo y no es propiamente una tarea algorítmica.
- Básicamente supone
  - la corrección de los errores que se detectan, y
  - la comprobación del tiempo y espacio que son necesarios para su ejecución.

Elección de un algoritmo

- A veces no tenemos donde elegir



CAPÍTULO XXIX Radicación 393

II. RAÍZ CUADRADA DE POLINOMIOS

RAÍZ CUADRADA DE POLINOMIOS ENTEROS

363

Para extraer la raíz cuadrada de un polinomio se aplica la siguiente regla práctica:

1) **Se ordena el polinomio dado.**
2) **Se halla la raíz cuadrada de su primer término, que será el primer término de la raíz cuadrada del polinomio; se eleva al cuadrado esta raíz y se resta del polinomio dado.**
3) **Se bajan los dos términos siguientes del polinomio dado y se divide el primero de éstos por el doble del primer término de la raíz. El cociente es el segundo término de la raíz. Este 2º término de la raíz con su propio signo se escribe al lado del doble del primer término de la raíz y se forma un binomio; este binomio se multiplica por dicho 2º término y el producto se resta de los dos términos que habíamos bajado.**
4) **Se bajan los términos necesarios para tener tres términos. Se duplica la parte de raíz ya hallada y se divide el primer término del residuo entre el primero de este doble. El cociente es el 3er término de la raíz.**
   Este 3er término, con su propio signo, se escribe al lado del doble de la parte de raíz hallada y se forma un trinomio; este trinomio se multiplica por dicho 3er término de la raíz y el producto se resta del residuo.
5) **Se continúa el procedimiento anterior, dividiendo siempre el primer término del residuo entre el primer término del doble de la parte de raíz hallada, hasta obtener residuo cero.**

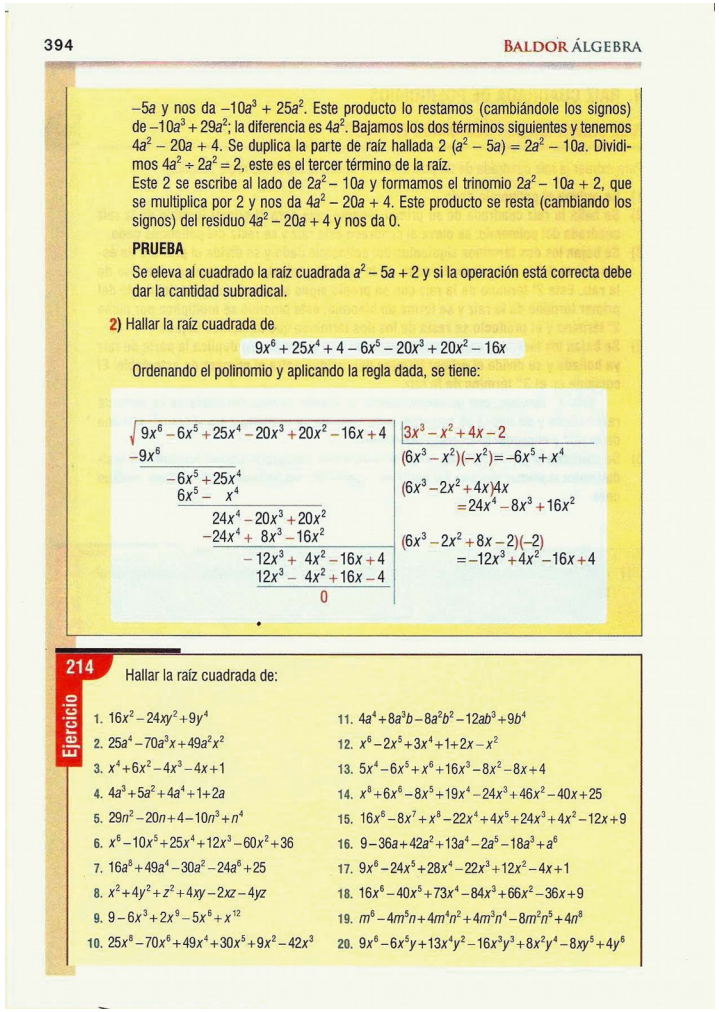
Ejemplos

1) Hallar la raíz cuadrada de $a^4+29a^2-10a^3-20a+4$. Ordenando el polinomio se obtiene:

$$\begin{array}{l|l} a^4-10a^3+29a^2-20a+4 & a^2-5a+2 \\ -a^4 & (2a^2-5a)(-5a)=-10a^3+25a^2 \\ \hline -10a^3+29a^2 & \\ 10a^3-25a^2 & (2a^2-10a+2)\,2=4a^2-20a+4 \\ \hline 4a^2-20a+4 & \\ -4a^2+20a-4 & \\ \hline 0 & \end{array}$$

EXPLICACIÓN

Hallamos la raíz cuadrada de a^4 que es a^2, este es el primer término de la raíz del polinomio. a^2 se eleva al cuadrado y da a^4, este cuadrado se resta del primer término del polinomio y bajamos los dos términos siguientes $-10a^3+29a^2$. Hallamos el doble de a^2 que es $2a^2$.
Dividimos $-10a^3 \div 2a^2 = -5a$, este es el segundo término de la raíz. Escribimos $-5a$ al lado de $2a^2$ y formamos el binomio $2a^2-5a$; este binomio lo multiplicamos por



394 BALDOR ÁLGEBRA

$-5a$ y nos da $-10a^3+25a^2$. Este producto lo restamos (cambiándole los signos) de $-10a^3+29a^2$; la diferencia es $4a^2$. Bajamos los dos términos siguientes y tenemos $4a^2-20a+4$. Se duplica la parte de raíz hallada $2(a^2-5a)=2a^2-10a$. Dividimos $4a^2 \div 2a^2 = 2$, este es el tercer término de la raíz.
Este 2 se escribe al lado de $2a^2-10a$ y formamos el trinomio $2a^2-10a+2$, que se multiplica por 2 y nos da $4a^2-20a+4$. Este producto se resta (cambiando los signos) del residuo $4a^2-20a+4$ y nos da 0.

PRUEBA

Se eleva al cuadrado la raíz cuadrada a^2-5a+2 y si la operación está correcta debe dar la cantidad subradical.

2) Hallar la raíz cuadrada de

$$9x^6+25x^4+4-6x^5-20x^3+20x^2-16x$$

Ordenando el polinomio y aplicando la regla dada, se tiene:

$$\begin{array}{l|l} 9x^6-6x^5+25x^4-20x^3+20x^2-16x+4 & 3x^3-x^2+4x-2 \\ -9x^6 & (6x^3-x^2)(-x^2)=-6x^5+x^4 \\ \hline -6x^5+25x^4 & \\ 6x^5-x^4 & (6x^3-2x^2+4x)4x \\ & =24x^4-8x^3+16x^2 \\ \hline 24x^4-20x^3+20x^2 & \\ -24x^4+8x^3-16x^2 & (6x^3-2x^2+8x-2)(-2) \\ & =-12x^3+4x^2-16x+4 \\ \hline -12x^3+4x^2-16x+4 & \\ 12x^3-4x^2+16x-4 & \\ \hline 0 & \end{array}$$

Ejercicio 214

Hallar la raíz cuadrada de:

1. $16x^2-24xy^2+9y^4$
2. $25a^4-70a^3x+49a^2x^2$
3. $x^4+6x^2-4x^3-4x+1$
4. $4a^3+5a^2+4a^4+1+2a$
5. $29n^2-20n+4-10n^3+n^4$
6. $x^6-10x^5+25x^4+12x^3-60x^2+36$
7. $16a^8+49a^4-30a^2-24a^6+25$
8. $x^2+4y^2+z^2+4xy-2xz-4yz$
9. $9-6x^3+2x^9-5x^6+x^{12}$
10. $25x^8-70x^6+49x^4+30x^5+9x^2-42x^3$
11. $4a^4+8a^3b-8a^2b^2-12ab^3+9b^4$
12. $x^6-2x^5+3x^4+1+2x-x^2$
13. $5x^4-6x^5+x^6+16x^3-8x^2-8x+4$
14. $x^8+6x^6-8x^5+19x^4-24x^3+46x^2-40x+25$
15. $16x^6-8x^7+x^8-22x^4+4x^5+24x^3+4x^2-12x+9$
16. $9-36a+42a^2+13a^4-2a^5-18a^3+a^6$
17. $9x^6-24x^5+28x^4-22x^3+12x^2-4x+1$
18. $16x^6-40x^5+73x^4-84x^3+66x^2-36x+9$
19. $m^6-4m^5n+4m^4n^2+4m^3n^4-8m^2n^5+4n^8$
20. $9x^6-6x^5y+13x^4y^2-16x^3y^3+8x^2y^4-8xy^5+4y^6$

Elección de un algoritmo

- Pero esa no es la situación general
- Supongamos que ante un cierto problema tenemos varios algoritmos para emplear.
- ¿Que algoritmo elegir?.
- Queremos buenos algoritmos en algún sentido propio de cada usuario.
- El problema central que se quiere resolver es el siguiente:
- **Dado un algoritmo, determinar ciertas características que sirven para evaluar su rendimiento.**

Elección de un algoritmo

- Multiplicación de enteros
  - Algoritmo clásico
  - Algoritmo de multiplicación a la rusa (del campesino ruso)

1. Escribir multiplicador y multiplicando en dos columnas
2. Hasta que el número bajo el multiplicador sea 1

REPETIR:

a) Dividir el número bajo el multiplicador por 2, ignorando los decimales

b) Doblar el número bajo el multiplicando sumándolo a si mismo

c) Rayar cada fila en la que el numero bajo el multiplicador sea par, o añadir los números que queden en la columna bajo el multiplicando.

Elección de un algoritmo

1) Escribir multiplicador y multiplicando en dos columnas
2) Repetir las siguientes operaciones hasta que el número bajo el multiplicador sea 1:

   a) Dividir el número bajo el multiplicador por 2, ignorando los decimales

   b) Doblar el número bajo el multiplicando sumándolo a si mismo

   c) Rayar cada fila en la que el número bajo el multiplicador sea par, o añadir los números que queden en la columna bajo el multiplicando.

Multiplicador	**Multiplicando**	**Resultado**	**Resultado acumulado**
	45		19 19
19			
22	38	--	19
11	76	76	95
5	152	152	247
2	304	---	247
1	608	608	**855**

Elección de un algoritmo

- Posibles **criterios** para la elección son:
- La adaptabilidad del algoritmo a los computadores
- Su simplicidad y elegancia
- El costo económico que su confección y puesta a punto puede acarrear.
- La duración del tiempo consumido para llevar a cabo el algoritmo (esto puede expresarse en términos del número de veces que se ejecuta cada etapa).

Problemas y casos

- (19,45) es un **Caso del Problema** de multiplicar dos enteros positivos
- Los problemas mas interesantes incluyen una colección infinita de casos (¿ajedrez?).
- Un algoritmo debe trabajar correctamente en cualquier caso del problema en cuestión.
- Para demostrar que un algoritmo es incorrecto, solo necesitamos encontrar un caso del problema que no produzca la respuesta correcta.
- La diferencia que existe entre **algoritmo y programa** es que cualquier sistema de computo real tiene un límite sobre el tamaño de los casos que puede manejar. Sin embargo, este límite no puede atribuirse al algoritmo que vayamos a usar.

Problemas y casos

- Formalmente:
  - El tamaño de un caso x es el número de bits necesarios para representar el caso en un computador usando un código precisamente definido y razonablemente compacto.
- Informalmente:
  - El tamaño de un caso es cualquier entero que, de algún modo, mida el número de componentes del caso.
- Ejemplos:
  - Ordenación: longitud del array, Matrices: Número de filas y columnas, Grafos: Número de vértices y arcos

Diferentes tipos de casos

- El tiempo consumido por un algoritmo puede variar mucho entre dos casos diferentes del mismo tamaño.
- Consideremos dos algoritmos de ordenación elementales: **Insercion** y **Seleccion.**

Procedimiento Insercion (T[1..n])

```
for i := 2 to n do
    x:= T[i]; j := i-1
    while j > 0 and x < T[j] do
    T[j+1] := T[j]
                          j := j-1
    T[j+1] := x
```

Procedimiento Seleccion (T[1..n])

```
for i:= 1 to n-1 do
minj := i; minx := T[i]
for j := i+1 to n do
    if T[j] < minx then minj := j
                        minx := T[j]
T[minj] := T[i];
T[i] := minx
```

Diferentes tipos de casos

- U y V dos arrays de n elementos: U ordenado en orden ascendente y V en orden descendente.
- **Comportamiento de Seleccion:**
  - Indiferente de U o V, el tiempo requerido para ordenar con Seleccion no varia en mas de un 15%.
- **Comportamiento de Insercion:**
  - Insercion(U) consume menos de 1/5 de segundo si U es un array de 5.000 elementos
  - Insercion(V) consume tres minutos y medio si V es un array de 5.000 elementos.

Si pueden darse esas diferencias, ¿Podremos hablar del tiempo consumido por un algoritmo solo en función del tamaño del caso?

Diferentes tipos de casos

- El tiempo que se obtiene para V da el máximo tiempo que se puede emplear para resolver el problema: V describe el ***peor caso***.
- El peor caso es útil cuando necesitamos una garantía total acerca de lo que durará la ejecucion de un programa. El tiempo del peor caso, para un n dado, se calcula a partir del caso de tamaño n que tarda más.
- Si el problema se ha de resolver en muchos casos distintos, es mas informativo el tiempo del ***caso promedio***: la media de los tiempos de todos los posibles casos del mismo tamaño **(¡hay que conocer la distribución de probabilidad asociada a los casos!)**
- U, sin embargo, describe el **mejor caso** de este problema, es decir, el caso sobre el que se tarda menos tiempo

Diferentes tipos de casos

- Consideramos el **Tiempo de Ejecución**,
- **Tiempo del Peor Caso:**
  - La función definida por el *máximo* número de etapas que se realizan en cualquier caso de tamaño n
- **Tiempo del Mejor Caso:**
  - La función definida por el *mínimo* número de etapas que se realizan en cualquier caso de tamaño n
- **Tiempo del Caso promedio:**
  - La función definida por el número *medio* de etapas que se realizan en cualquier caso de tamaño n
  - El caso promedio no existe como tal

Diferentes tipos de casos

- Es difícil estimar exactamente el tiempo de ejecución
  - El mejor caso depende del input
  - El caso promedio es dificil de calcular
  - Por tanto generalmente nos referiremos al Peor Caso
    - Es fácil de calcular
    - Suele aproximarse al tiempo de ejecución real