AMBIENTE INTERACTIVO PARA LA VISUALIZACIÓN DE CONCEPTOS ASOCIADOS A LAS TÉCNICAS DE DISEÑO DE ALGORITMOS

LIBRO DE PROYECTO PRESENTADO PARA OPTAR AL TÍTULO DE INGENIERO DE SISTEMAS

Mauricio Giraldo Acosta

Director: Raúl Alfredo Chaparro Aguilar

Escuela Colombiana de Ingeniería Julio Garavito

Proyecto de Grado

Programa Ingeniería de Sistemas

Bogotá D.C.

2016

Hoja de aprobación:

Aprobado por el director de Proyecto Raúl Alfredo Chaparro Aguilar en cumplimiento de los requisitos exigidos por la Escuela Colombiana de Ingeniería Julio Garavito para optar al título de Ingeniero de Sistemas.

Raúl Alfredo Chaparro Aguilar
Director Proyecto de Grado

DEDICATORIA

Este proyecto de Grado lo dedico a mi familia, por quien soy quien soy. A mis padres, por su apoyo incondicional, consejos, comprensión, amor, ayuda y confianza en mis proyectos personales hasta ahora emprendidos. A mis hermanos, formadores de confianza, respeto, valores y ejemplo a seguir como personas y profesionales.

AGRADECIMIENTOS

Agradezco a la Escuela Colombiana de Ingeniería Julio Garavito, por permitirme ser parte de ella y haberme abierto las puertas a una experiencia de aprendizaje inigualable, formándome, así como profesional de Ingeniería de Sistemas, así como a mis docentes quienes brindaron su conocimiento, experiencia y apoyo para desarrollar con solvencia las habilidades necesarias para convertirme en un profesional de calidad.

Agradezco también a mi Director de Proyecto de Grado, matemático Raúl Alfredo Chaparro Aguilar, por haberme brindado la oportunidad de recurrir a su capacidad, experiencia y conocimiento para el buen desarrollo de este proyecto y para guiar el desarrollo de las actividades necesarias que llevaran a un producto terminado de manera satisfactoria.

Finalmente, agradezco a todos mis compañeros de la Escuela Colombiana de Ingeniería Julio Garavito, quienes hicieron parte esencial de mi proceso de formación como ingeniero durante los últimos cinco años.

TABLA DE CONTENIDO

RESUMEN

En este libro se presenta una alternativa lúdica, gráfica e interactiva para el aprendizaje de las técnicas asociadas al diseño de algoritmos, basado en un conjunto de situaciones predefinidas, con los algoritmos enseñados de manera frecuente a los estudiantes, que les permitirá desarrollar comparaciones, talleres y asignaciones académicas, logrando así apoyar las metodologías tradicionales de enseñanza, aprovechando de mejor manera las herramientas tecnológicas de aprendizaje disponibles, con el fin de estimular su fácil aprendizaje y aplicación.

ABSTRACT

This book presents a playful, graphic, and interactive alternative for learning techniques associated with algorithm design, based on a set of predefined situations, with algorithms taught frequently to students, which will allow them to develop comparisons, workshops, and academic assignments, thus helping to support traditional teaching methodologies, making better use of the available technological learning tools, to stimulate their easy learning and application.

INTRODUCCIÓN

La palabra algoritmo proviene de la palabra Latina medieval *algorism* y la palabra griega *arithmos* (Blass *et al.,*2003). En matemáticas y ciencias de la computación, un algoritmo es un conjunto de pasos contenidos, los cuales cada uno representa una operación a realizarse. Los algoritmos realizan cálculos, procesamiento de datos y tareas de razonamiento automático (Bell *et al.,*1971).

Ahora bien, la educación virtual, también conocida como *e-learning* o educación a distancia asistida por computadora, consiste en el uso de internet y de herramientas capaces de ser desplegadas en dispositivos computacionales con el fin de apoyar los procesos de educación tradicionales, y, en algunos casos, de reemplazarlos (Saldarriaga, 2011).

Partiendo de los conceptos anteriormente descritos, y de la evidente falta de inclusión de tecnologías de información con fines pedagógicos durante el proceso de aprendizaje de técnicas asociadas al diseño de algoritmos en las universidades, a nivel mundial y, por supuesto, en la decanatura de Ingeniería de Sistemas de la Escuela Colombiana de Ingeniería Julio Garavito, proponemos un ambiente visual, contextualizado, virtual y que utiliza herramientas tecnológicas de vanguardia, con el fin de facilitar la enseñanza de las técnicas asociadas al diseño y construcción de algoritmos; concretamente algoritmos Voraces, algoritmos Divide y Conquista y algoritmos de Programación Dinámica.

Esta metáfora lúdica, visual e interactiva, procura que los estudiantes se apropien de conceptos que tradicionalmente se enseñan en las aulas de clase de manera teórica, abstracta y difícil de aprehender de un modo sencillo, que va en concordancia a las teorías tradicionales, es decir, este ambiente propone una versión análoga de la teoría tradicional de manera tangible y sencilla de entender, con el fin de facilitar el aprendizaje del estudiante, y que esté en capacidad de utilizar conceptos base de diseño y construcción de algoritmos en el corto, mediano y largo plazo.

OBJETIVOS

- Contextualizar, diseñar y construir un ambiente visual (basado en el color y figuras) de interacción y experimentación para problemas y conceptos asociados a las técnicas de diseño de Algoritmos (las técnicas a estudiar son: Dividir y Conquistar, Algoritmos Voraces y Programación Dinámica).

- Diseñar y construir un ambiente informático basado en figuras con componentes visuales donde prime el color, figuras y objetos familiares.

- Contextualizar, definir alcances, documentar el ambiente con talleres y laboratorios de aplicación

- Investigar sobre trabajos existentes en la misma línea.

- Integrar el proyecto a la línea de la conceptualización a través de la experimentación, interacción y conjetura con elementos basado en propiedades de color y figuras geométricas básicas.

- Integrar el proyecto en el contexto de los juegos discretos. Para aprovechar experiencias realizadas.

JUSTIFICACIÓN

La interacción que permite la tecnología y los modelos concretos enriquece las maneras de aprender. Muchas veces el obstáculo de la conceptualización es el lenguaje simbólico. Proponer un ambiente para que se haga la misma reflexión formal, pero que se ayude del sentido de la vista para pensar, enriquece el contexto de aprendizaje. Esto le permite al estudiante pensar y conceptualizar con elementos familiares y al profesor repensar en la nueva creación de problemas y pautas de evaluación. Permitir la interacción en el diseño fortalece el aprendizaje de los principios del diseño algorítmico.

ESTADO DEL ARTE

En matemáticas, lógica, ciencias de la computación y disciplinas relacionadas, un algoritmo (del griego y latín, dixit algorithmus y éste a su vez del matemático persa Al-Juarismi) es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución. Los algoritmos son el objeto de estudio de la algoritmia (Cormen *et al.,*2001).

En la vida cotidiana, se emplean algoritmos frecuentemente para resolver problemas. Algunos ejemplos son los manuales de usuario, que muestran algoritmos para usar un aparato, o las instrucciones que recibe un trabajador por parte de su patrón. Algunos ejemplos en matemáticas, son el algoritmo de multiplicación, para calcular el producto, el algoritmo de la división para calcular el cociente de dos números, el algoritmo de Euclides para obtener el máximo común divisor de dos enteros positivos, o el método de Gauss para resolver un sistema de ecuaciones lineales (Sipser,2005).

Los algoritmos pueden ser expresados de muchas maneras, incluyendo al lenguaje natural, pseudocódigo, diagramas de flujo y lenguajes de programación entre otros. Las descripciones en lenguaje natural tienden a ser ambiguas y extensas. El usar pseudocódigo y diagramas de flujo evita muchas ambigüedades del lenguaje natural. Dichas expresiones son formas más estructuradas para representar algoritmos; no obstante, se mantienen independientes de un lenguaje de programación específico (Kelley, Dean, 1995).

La descripción de un algoritmo usualmente se hace en tres niveles:

- Descripción de alto nivel. Se establece el problema, se selecciona un modelo matemático y se explica el algoritmo de manera verbal, posiblemente con ilustraciones y omitiendo detalles.
- Descripción formal. Se usa pseudocódigo para describir la secuencia de pasos que encuentran la solución.
- Implementación. Se muestra el algoritmo expresado en un lenguaje de programación específico o algún objeto capaz de llevar a cabo instrucciones.

También es posible incluir un teorema que demuestre que el algoritmo es correcto, un análisis de complejidad o ambos (Kelley, Dean, 1995).

Algoritmos Voraces

Un algoritmo voraz (también conocido como ávido, devorador o goloso) es aquel que, para resolver un determinado problema, sigue una heurística consistente en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. Este esquema algorítmico es el que menos dificultades plantea a la hora de diseñar y comprobar su funcionamiento. Normalmente se aplica a los problemas de optimización (Brassard *et al.,*1997).

Algoritmos Divide y Conquista

En la cultura popular, divide y vencerás hace referencia a un refrán que implica resolver un problema difícil, dividiéndolo en partes más simples tantas veces como sea necesario, hasta que la resolución de las partes se torna obvia. La solución del problema principal se construye con las soluciones encontradas (Reynolds, Tymann, 2008).

En las ciencias de la computación, el término divide y vencerás (DYV) hace referencia a uno de los más importantes paradigmas de diseño algorítmico. El método está basado en la resolución recursiva de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar. El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente. Al final, las soluciones a cada uno de los subproblemas se combinan para dar una solución al problema original (Aho, Hopcroft,1974).

Esta técnica es la base de los algoritmos eficientes para casi cualquier tipo de problema como, por ejemplo, algoritmos de ordenamiento (quicksort, mergesort, entre muchos otros), multiplicar números grandes (Karatsuba), análisis sintácticos (análisis sintáctico top-down) y la transformada discreta de Fourier. Por otra parte, analizar y diseñar algoritmos de DyV son tareas que lleva tiempo dominar. Al igual que en la inducción, a veces es necesario sustituir el problema original por uno más complejo para conseguir realizar la recursión, y no hay un método sistemático de generalización (Manber, 1989).

El nombre divide y vencerás también se aplica a veces a algoritmos que reducen cada problema a un único subproblema, como la búsqueda binaria para encontrar un elemento en una lista ordenada (o su equivalente en computación numérica, el algoritmo de bisección para búsqueda de raíces). Estos algoritmos pueden ser implementados más eficientemente que los algoritmos generales de “divide y vencerás”; en particular, si es usando una serie de recursiones que lo convierten en simples bucles. Bajo esta amplia definición, sin embargo, cada algoritmo que usa recursión o bucles puede ser tomado como un algoritmo de “divide y vencerás”. El nombre decrementa y vencerás ha sido propuesta para la subclase simple de problemas (Knuth, 1997).

La corrección de un algoritmo de “divide y vencerás”, está habitualmente probada una inducción matemática, y su coste computacional se determina resolviendo relaciones de recurrencia (Knuth, 1997).

Algoritmos de Programación Dinámica

Una subestructura óptima significa que se pueden usar soluciones óptimas de subproblemas para encontrar la solución óptima del problema en su conjunto. Por ejemplo, el camino más corto entre dos vértices de un grafo se puede encontrar calculando primero el camino más corto al objetivo desde todos los vértices adyacentes al de partida, y después usando estas soluciones para elegir el mejor camino de todos ellos. En general, se pueden resolver problemas con subestructuras óptimas siguiendo estos tres pasos (Xumari, 1967):

- Dividir el problema en subproblemas más pequeños.

- Resolver estos problemas de manera óptima usando este proceso de tres pasos recursivamente.

- Usar estas soluciones óptimas para construir una solución óptima al problema original.

Los subproblemas se resuelven a su vez dividiéndolos en subproblemas más pequeños hasta que se alcance el caso fácil, donde la solución al problema es trivial.

Decir que un problema tiene subproblemas superpuestos es decir que se usa un mismo subproblema para resolver diferentes problemas mayores. Por

ejemplo, en la sucesión de Fibonacci (F3 = F1 + F2 y F4 = F2 + F3) calcular cada término supone calcular F2. Como para calcular F5 hacen falta tanto F3 como F4, una mala implementación para calcular F5 acabará calculando F2 dos o más veces. Esto sucede siempre que haya subproblemas superpuestos: una mala implementación puede acabar desperdiciando tiempo recalculando las soluciones óptimas a problemas que ya han sido resueltos anteriormente (Gurevich, Yuri, 2000).

Esto se puede evitar guardando las soluciones que ya hemos calculado. Entonces, si necesitamos resolver el mismo problema más tarde, podemos obtener la solución de la lista de soluciones calculadas y reutilizarla. Este acercamiento al problema se llama memoización (no confundir con memorización; en inglés es llamado memoization). Si estamos seguros de que no volveremos a necesitar una solución en concreto, la podemos descartar para ahorrar espacio. En algunos casos, podemos calcular las soluciones a problemas que de antemano sabemos que vamos a necesitar (Gurevich, Yuri, 2000).

En resumen, la programación hace uso de:

- Subproblemas superpuestos
- Subestructuras óptimas
- Memorización

La programación toma normalmente uno de los dos siguientes enfoques (Sedgewick, 2004):

- Top-down: El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursión.
- Bottom-up: Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado.

Originalmente, el término de programación dinámica se refería a la resolución de ciertos problemas y operaciones fuera del ámbito de la Ingeniería Informática, al igual que hacía la programación lineal. Aquel contexto no tiene relación con la programación en absoluto; el nombre es una coincidencia. El término también lo usó en los años 40 Richard Bellman, un matemático norteamericano, para describir el proceso de resolución de problemas donde hace falta calcular la mejor solución consecutivamente (Denardo,2003).

Algunos lenguajes de programación funcionales, sobre todo Haskell, pueden usar la memorización automáticamente sobre funciones con un conjunto concreto de argumentos, para acelerar su proceso de evaluación. Esto sólo es posible en funciones que no tengan efectos secundarios, algo que ocurre en Haskell, pero no tanto en otros lenguajes (Sniedovich, 2010).

Principio de Optimalidad de Bellman

Cuando hablamos de optimizar nos referimos a buscar alguna de las mejores soluciones de entre muchas alternativas posibles. Dicho proceso de optimización puede ser visto como una secuencia de decisiones que nos proporcionan la solución correcta. Si, dada una subsecuencia de decisiones, siempre se conoce cuál es la decisión que debe tomarse a continuación para obtener la secuencia óptima, el problema es elemental y se resuelve trivialmente tomando una decisión detrás de otra, lo que se conoce como estrategia voraz. En otros casos, aunque no sea posible aplicar la estrategia voraz, se cumple el principio de optimalidad de Bellman que dicta que «dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima». En este caso sigue siendo posible el ir tomando decisiones elementales, en la confianza de que la combinación de ellas seguirá siendo óptima, pero será entonces necesario explorar muchas secuencias de decisiones para dar con la correcta, siendo aquí donde interviene la programación dinámica (Bellman, 1957).

Contemplar un problema como una secuencia de decisiones equivale a dividirlo en problemas más pequeños y por lo tanto más fáciles de resolver como hacemos en Divide y Vencerás, técnica similar a la de programación dinámica. La programación dinámica se aplica cuando la subdivisión de un problema conduce a (Bellman, 1952):

- Una enorme cantidad de problemas.
- Problemas cuyas soluciones parciales se solapan.
- Grupos de problemas de muy distinta complejidad.

DESARROLLO TÉCNICO DE AVIAL

A lo largo de los años, las personas relacionadas al rol de la enseñanza del diseño y construcción de algoritmos (ingenieros, matemáticos, computólogos. Etc.), han podido identificar que la imposibilidad de hacer tangibles los conceptos anteriormente mencionados con el fin de que los estudiantes estén en capacidad de explorar y aprehender la conceptualización necesaria, han generado, en variadas ocasiones, la desmotivación y renuncia de los estudiantes a los cursos básicos de diseño de algoritmos en ciencias de la computación, ingeniería de software, ingeniería de sistemas e ingeniería de computación.

Actualmente incluso, los estudiantes deben pasar por la difícil situación de intentar aprehender los conceptos asociados al diseño y construcción de algoritmos al verse obligados a abstraer, en exceso, los conceptos matemáticos y computacionales asociados, lo cual, como se explicó anteriormente, deriva en la desmotivación y desinterés por parte de los estudiantes al estudiar algoritmos como un concepto abstracto y carente de utilidad y de aplicación a su vida real.

Dada la situación ya explicada, hemos enfocado nuestros esfuerzos en construir una solución que permita tanto a estudiante como a profesor, interactuar con los conceptos abstractos de una manera tangible, introduciendo así la capacidad de los estudiantes para manipular y actuar de manera pragmática sobre las técnicas de diseño y construcción de algoritmos “Divide y Conquista”, “Algoritmos Voraces” y “Programación Dinámica”.

Durante el desarrollo y construcción de este proyecto, se tuvieron en cuenta tecnologías web modernas, con el fin de llevar la capacidad del proyecto a cualquier ambiente en el que simplemente se cuente con una conexión a internet, sin librerías adicionales a las típicas que es capaz de manipular cualquier navegador, con el fin de masificar la accesibilidad a nuestra audiencia objetivo, la cual está compuesta por estudiantes de los primeros cursos de diseño y construcción de algoritmos, estructuras de datos de proyectos curriculares tales como: ‘Ingeniería de Software’, ‘Ingeniería de Computación’, ‘Ingeniería de Sistemas’ y ‘Ciencias de la Computación’.

Con el fin de ilustrar y mostrar de manera previa la construcción actual de este proyecto, se presentan a continuación algunas ilustraciones relacionadas a la presentación actual del mismo.

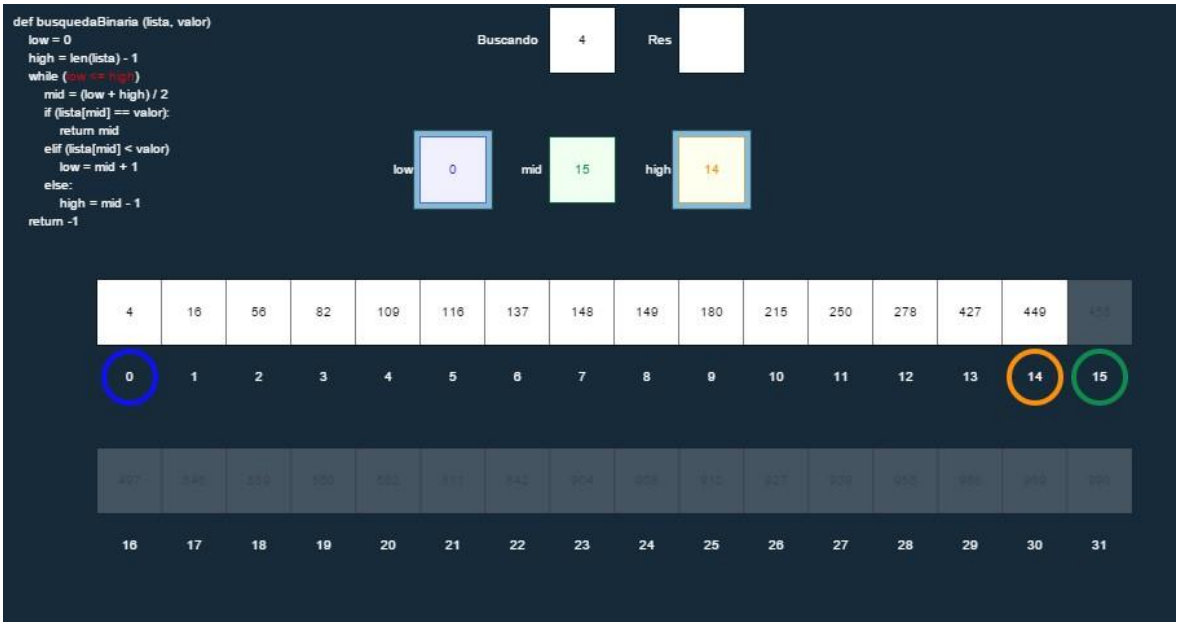

Fig. 1 - Búsqueda Binaria sobre Listas Ordenadas

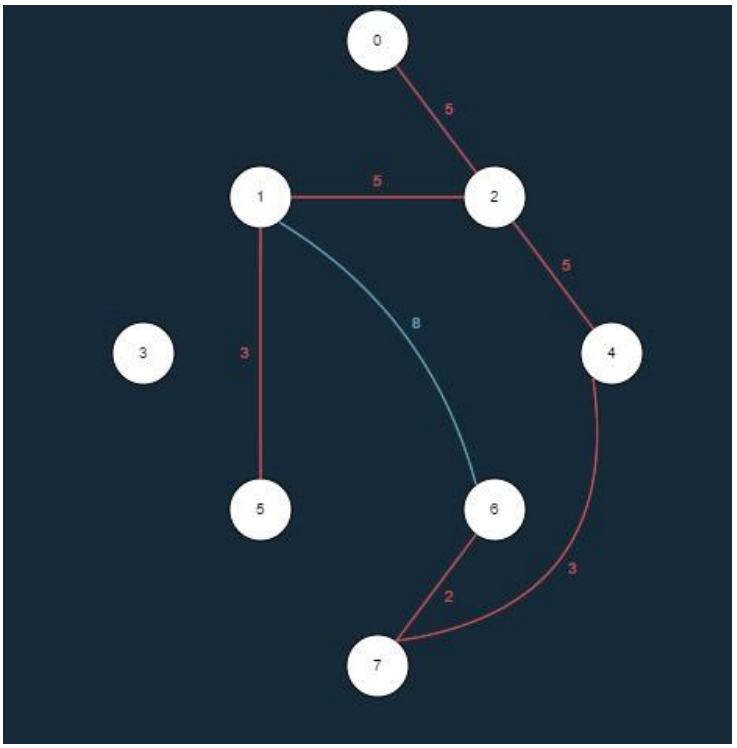

Fig. 2- Algoritmo de Prim aplicado sobre un grafo

Para soportar la correcta operación y desarrollo del uso del proyecto, así como para fomentar la interactividad y la visualización, se ofrece un panel de manipulación que permitirá, en cualquier situación, observar de diferentes maneras la forma en que se 'juega' con las animaciones con el fin de propender y velar por la interactividad y la capacidad de observación del desarrollo de los conceptos tanto para el estudiante, como para el profesor.

Luego del estudio y construcción de este ambiente visual, se espera conseguir un ambiente para la interacción, estudio y aprendizaje del diseño y construcción de algoritmos asociados a las tres técnicas anteriormente descritas para este fin, compuesto por software, diseño de actividades, bancos de problemas de aplicación de los conceptos asociados a nuestra problemática, razón por la cual hemos definido un alcance provisto de todos los anteriores elementos, descrito a continuación.

Alcance del Proyecto

-Software:

Ambiente para definir situaciones problemáticas asociada con el diseño de algoritmos (con las técnicas dividir y conquistar, algoritmos voraces y programación dinámica). Y permitir tener el proceso visual en figuras de colores y figuras geométricas. Que permita mostrar visualmente lo que se conceptualiza.

Que a partir de alguna situación gráfica pueda validar la correspondiente propuesta de solución teórica del problema.

-Documentación:

Correspondiente al proyecto, laboratorios y ejemplos de uso de la herramienta, talleres en los que se evidencie la importancia del trabajo con este tipo de ambiente y recomendaciones y experiencias del trabajo.

CONCLUSIONES

La implementación de tecnologías con fines educativos, que están destinadas a apoyar la enseñanza y teoría tradicional, ayudan a los estudiantes a sentir mayor nivel de aprehensión de los conceptos aprendidos y al profesor a facilitar su forma de enseñar y transmitir el conocimiento.

Las plataformas web para el desarrollo de aplicaciones, resultan sumamente convenientes en términos de accesibilidad, portabilidad y requisitos para su buen funcionamiento.

REFERENCIAS Y BIBLIOGRAFÍA

Bell, C. Gordon and Newell, Allen (1971), Computer Structures: Readings and Examples, McGraw–Hill Book Company, New York.

Blass, Andreas; Gurevich, Yuri (2003). "Algorithms: A Quest for Absolute Definitions". Bulletin of European Association for Theoretical Computer Science. 81.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. y Stein, C (2001). Introduction to Algorithms.

Sipser, Michael (2005). Introduction to the Theory of Computation (2 edición). Course Technology. ISBN 978-0534950972.

Kelley, Dean (1995). Teoría de Autómatas y Lenguajes Formales. Prentice Hall. ISBN 0-13-497777-7.

Brassard, Gilles; Bratley, Paul (1997). Algoritmos voraces. Fundamentos de Algoritmia. Madrid: PRENTICE HALL. ISBN 84-89660-00-X.

Carl Reynolds & Paul Tymann (2008). Schaum's Outline of Principles of Computer Science. McGraw-Hill. ISBN 978-0-07-146051-4.

Aho, A. (1974). The Design and Analysis of Computer Algorithms.

Mamber, U. (1989). Introduction to Algorithms. A Creative Approach.

Knuth, D. E. (1997). The Art of Computer Programming.

Xumari, G.L. (1967) Introduction to dynamic programming. Wiley & Sons Inc.

Gurevich, Yuri (2000). Sequential Abstract State Machines capture Sequential Algorithms. ACM Transactions on Computational Logic.

Sedgewick, R. (2004) Algorithms in C (3r ed).

Denardo, E.V. (2003), Dynamic Programming: Models and Applications, Mineola, NY: Dover Publications, ISBN 978-0-486-42810-9.

Sniedovich, M. (2010), Dynamic Programming: Foundations and Principles, Taylor & Francis, ISBN 978-0-8247-4099-3.

Bellman, R.E. (1957). Dynamic Programming. Princeton University Press, Princeton, NJ, Republished 2003: Dover, ISBN 0-486-42809-5.

Bellman, R. (1952). On the Theory of Dynamic Programming, Proceedings of the National Academy of Sciences.