DISEÑO DE UN *SOFTWARE* EN JAVA PARA LA TOMA DE DECISIONES EN EL MANEJO DE INVENTARIOS DETERMINÍSTICOS[1]

Design of Java Software for decisions in the handling of deterministic inventories

Recibido: 25 de julio del 2012
Aprobado: 30 de septiembre del 2012

Carlos Carvajal T.*

Resumen

En este artículo de investigación se analiza el modelo de inventarios con demanda determinística de entubamiento con los algoritmos de programación dinámica, con el fin de optimizarlos. El objetivo es el diseño de un *software* para la toma de decisiones en el manejo de inventarios de entubamiento con demanda determinística. La metodología es la orientada a objetos. Los resultados son: diseño del modelo matemático, algoritmos recursivos de programación dinámica, programa en Java, diagramas en el Lenguaje de Modelamiento Unificado (UML). Como conclusión se puede afirma que para desarrollar un *software* con un alto componente matemático, este se debe hacer en dos fases: en la primera fase se desarrolla el modelo matemático y en la segunda se implementa el Proceso Racional Unificado (RUP). El artículo forma parte del proyecto de investigación "Diseño de un producto de *software* en Java, para modelos de inventarios determinísticos de lote económicos de pedidos", realizado en el 2011 en la Universidad Cooperativa, sede Cali, por el grupo de investigación "Herramientas para apoyo de procesos educativos (Gihaped)".

Palabras clave: demanda determinista, Lenguaje de Modelamiento Unificado (UML), modelos matemáticos, programación dinámica, programación orientada a objetos.

Abstract

This research paper studies a tunneling inventory model with deterministic demand, using dynamic programming algorithms to optimize inventories. The objective is to design software for decision-making in the management of tunneling inventories with deterministic demand. The methodology is Object Oriented. The results consisted in the design of a mathematical model; recursive algorithms implemented with dynamic programming; the Java application and Unified Modeling Language (UML) diagrams. As a conclusion we state that the development of software with important mathematical components should be done in two phases. In the first phase the mathematical model must be developed. The second phase is the implementation of the Unified Rational Process (RUP). The paper is part of the research project: "Design of a Java software product for lot, deterministic, ordering inventory models", developed at the Universidad Cooperativa at Cali in 2011, by the research group "Tools for the Support of Educational Processes (Gihaped)".

Keywords: deterministic demand, Unified Modeling Language (UML), mathematical models, dynamic programming, object oriented programming.

Cómo citar este artículo: C. Carvajal T. "Diseño de un *software* en Java para la toma de decisiones en el manejo de inventarios determinísticos". *Revista Ingeniería Solidaria*, vol. 8, núm. 15, 2012, pp. 33-45.

[1] Artículo de investigación derivado del proyecto de investigación "Diseño de un producto de *software* en Java, para modelos de inventarios determinísticos de lote económicos de pedidos", realizado en el 2011 en la Universidad Cooperativa, sede Cali, por el grupo de investigación "Herramientas para apoyo de procesos educativos (Gihaped)".

* Ingeniero de Sistemas de la Universidad del Valle. Especialista en Sistemas de Información de la Universidad del Valle. Especialista en Ingeniería de Sistemas de la Universidad del Valle. Magíster en Ingeniería de Sistemas de la Universidad del Valle. Docente de la Facultad de Ingeniería de la Universidad Cooperativa de Colombia, sede Cali. Correo electrónico: carlos.carvajal@campusucc.edu.co

Introducción

Actualmente, en muchas Pymes (pequeñas y medianas empresas) de bienes o servicios es necesario adquirir, vender y transportar inventarios de materia prima o producto terminado, desde los proveedores hasta las empresas, y desde la empresa hasta los clientes.

En muchos casos, el inventario debe ser transportado recorriendo varias ciudades (etapas) por diferentes carreteras hasta su destino final. Todo este desplazamiento y mantenimiento del bien durante el trayecto conlleva costos, dado por las diferentes rutas entre la ciudad de origen y la ciudad de destino. Para optimizar los costos es necesario saber cuál es la ruta más corta (ruta óptima). Para ello se deben hacer los cálculos y modelos matemáticos de programación dinámica en el contexto de la investigación de operaciones, pero se presenta un gran problema, y es que, debido al gran caudal de datos y a la complejidad de los modelos matemáticos, los cálculos pueden tener errores, o no se pueden hacer.

Teniendo en cuenta lo anterior, muchas Pymes desean comprar un *software*, pero este resulta muy costoso o difícil de usar. Para ayudar a resolver el problema planteado se desarrolló un producto de *software* para modelos de inventarios determinísticos de entubamiento de lote económico de pedidos de un ítem, usando modelos de programación dinámica, que sirve para la toma de decisiones. Para alcanzar este objetivo se cumplieron los siguientes objetivos específicos: desarrollar los modelos matemáticos de programación dinámica en el contexto de los inventarios; diseñar un modelo orientado a objetos usando el Lenguaje de Modelamiento Unificado (UML); traducir los modelos matemáticos y el UML al lenguaje de programación Java; crear las interfaces para facilitar el manejo al usuario y, por último, diseñar y realizar las pruebas del y al producto de *software*.

Para desarrollar el producto de *software* se utilizó el paradigma de Desarrollo de *Software* Orientado a Objetos (DSOO) con el Proceso Racional Unificado (RUP) y el Lenguaje de Modelamiento Unificado (UML).

Es importante resaltar que en el programa de Ingeniería de Sistemas de la Universidad Cooperativa de Colombia, sede Cali, se dictan las asignaturas de Investigación de operaciones, Simulación, Modelamiento, Estructura de datos y Teoría de grafos, entre otras, que aportaron la parte conceptual y epistemológica, ya que en estas asignaturas se ven contenidos que atañen con el proyecto y que sirvieron para el desarrollo del producto de *software*.

Una vez construida la primera versión del producto de *software*, se desea el siguiente impacto: el producto de *software* podrá ser usado por las Pymes para la toma de decisiones en la gestión de inventarios. En el contexto académico podrá servir como herramienta pedagógica en las áreas y asignaturas de Investigación de operaciones, Administración de inventarios y Programación dinámica, entre otras. También podrá servir de ejemplo aplicativo para el desarrollo de *software* en las aéreas y asignaturas de Ingeniería de *software*, Desarrollo de *software*, Teoría de grafos, Programación, Estructura de datos, etcétera. El desarrollo del proyecto ha servido para la formación del talento humano y el fomento de la cultura investigativa en la universidad, ya que el estudiante pudo ver y practicar lo visto en clase con el desarrollo y funcionamiento del *software*.

El presente proyecto forma parte de la investigación sobre inventarios determinísticos en un sitio, variables en el tiempo, que está adelantando el grupo Gihaped.

Metodología

Para alcanzar los objetivos, lo primero que se hizo fue una búsqueda bibliográfica en la que se encontraran los modelos de inventarios; después se diseñaron los modelos matemáticos; en segundo lugar, cuando estuvieron claros los modelos matemáticos, se comenzaron a desarrollar los algoritmos recursivos en programación dinámica; finalmente, cuando los algoritmos desarrollados se probaron con modelos típicos de inventarios determinísticos, se procedió a construir el *software*.

Para la construcción del *software* en Java se usó la metodología para el desarrollo de *software* orientado a objetos [1], con el RUP [2], con sus diferentes fases y artefactos en UML, del siguiente modo: *modelado del negocio,* se definió el problema en términos de su planteamiento, diseño del modelo y su solución; *requisitos,* se identificaron las tareas y se definieron los responsables; *diseño,* se analizaron los requerimientos del proyecto y los recursos; *implementación,* se efectuó la implementación física de los algoritmos y modelos matemáticos; *despliegue,* se hizo todo lo concerniente a las pruebas de caja blanca y caja negra.

Fundamentación teórica

Las teorías que se usaron para desarrollar el producto de *software* fueron:

- Teoría de inventarios determinísticos.
- Programación dinámica.
- Desarrollo de Software Orientado a Objetos (DSOO).

Teoría de inventarios determinísticos

La inversión en inventarios puede representar más del 25% de los activos totales [3]. Inventario es todo activo que se encuentra almacenado y disponible para cuando sea requerido por un cliente. También se puede decir que inventario [4] son las existencias de cualquier artículo o recurso utilizado en una organización. Un sistema de inventarios es la serie de políticas y controles que monitorean los niveles de inventarios y determinan los niveles que se deben mantener, el momento en que las existencias se deben reponer y el tamaño que deben tener los pedidos. El solo hecho de mantener o no un inventario genera los costos que se describen en los siguientes apartados [5].

Costos de pedido o alistamiento

Son todos los costos asociados con el reabastecimiento del inventario. Estos costos varían con el número de pedido y son:

- Costos de requisición.
- Costos de emitir y seguir la orden de compra.
- Costos de inspección al recibir y colocar los artículos en inventario y transporte.

Costos de mantenimiento

Estos son los costos asociados con mantener una cantidad disponible. Varían con el nivel y los periodos que se mantengan, y son:

- Costos de oportunidad de la inversión.
- Costos de almacenamiento (alquiler, calefacción, refrigeración, vigilancia, etcétera).
- Deterioro del producto u obsolescencia.
- Impuestos depreciación y seguro.
- Costos de transporte.

Costos de quedarse corto o agotado

Estos son los costos de penalización en que incurre la empresa cuando se queda sin la mercancía, y los costos por pedido del cliente.

Teniendo en cuenta el análisis de costos surgen las siguientes preguntas:

- ¿Cuánto producir?
- ¿Cuándo producir?

Es importante recordar que los productos pueden presentar cierto tipo de demandas que, como fenómeno, pueden ser probabilísticas o determinísticas. Si la demanda es determinística surgen los siguientes casos: que la demanda sea uniforme a lo largo del tiempo, es decir, la misma demanda durante todos los meses (figura 1); que la demanda sea de tipo estacional, es decir, la demanda se da en determinadas épocas del año (figura 2); que los inventarios se deban transportar de un origen a un destino por varios caminos, y se deba escoger el camino más corto (figura 3).

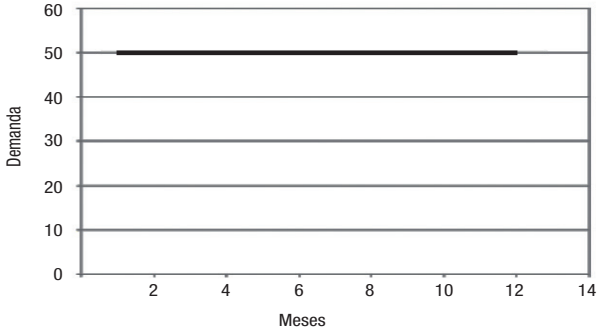

Figura 1. Demanda uniforme
Fuente: el autor

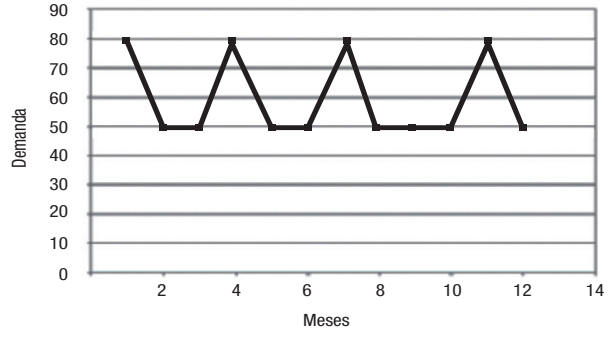

Figura 2. Demanda estacional
Fuente: el autor

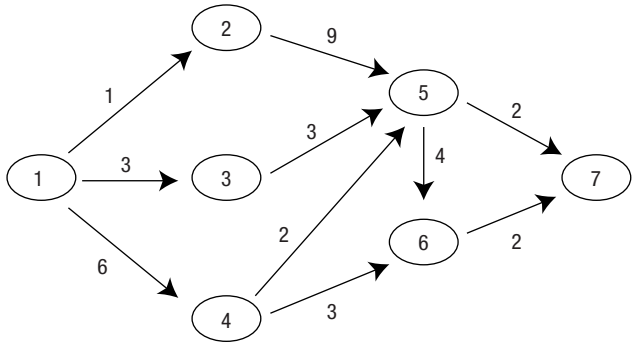

Figura 3. Inventarios que se deben transportar (inventarios de entubamiento)
Fuente: el autor

Si la demanda es uniforme (figura 1), el inventario se puede controlar fácilmente, ya que se puede predecir el *cuánto* y el *cuándo*.

Si la demanda presenta características estacionales (figura 2), se da la situación de manejar los costos variables asociados con el almacenamiento, costos por depreciación, costos de transporte, costos por alistamiento de equipos y los costos administrativos por órdenes de producción.

Si el caso es el tercero, inventarios que se deben transportar (inventarios de entubamiento) (figura 3), la situación se complica aún más, ya que los costos asociados con los inventarios comienzan a variar. El inventario se debe transportar por diferentes etapas y lo que varía es el costo de transporte, por lo que se debe buscar la distancia más corta desde el origen hasta el final para reducir costos.

Programación dinámica

Para resolver este tipo de problemas se han planteado modelos matemáticos en el contexto de la *investigación de operaciones* [5], y dentro de la *investigación de operaciones* está la *programación dinámica,* en la cual se elaboran modelos matemáticos con las siguientes características:

- Ocurren periódicamente en un tiempo y un espacio.
- Se pueden definir procesos (etapas).
- Los procesos tienen una entrada y una salida.
- Presentan características recursivas, es decir, todo proceso depende de los anteriores.

Un problema típico de programación dinámica es el de la figura 3, en la que se muestran siete procesos o etapas que van de la etapa 1 (origen) a la etapa 7 (destino).

El problema consiste en hallar la menor ruta entre la estación (etapa) 1 y la 7. Para calcular la menor ruta se utilizó el algoritmo de Dijkstra [6], también conocido como algoritmo del viajante [5], que se basa en calcular los caminos por cada uno de los nodos (etapas), del primero al último, y después del último al primero (en forma recursiva).

Para ilustrar, se explican los cálculos, paso por paso, utilizando grafos (figura 4) con la siguiente notación:

Óvalo es la etapa -> Etapa (n)

Arista es la distancia entre cada etapa -> arista (ni, nj) (las longitudes de las aristas no están a escala).

Los cálculos se realizan en dos fases: en la primera se realizan los cálculos de la etapa 1 a la 7, en la cual se calculan los óptimos acumulados para cada etapa; en la segunda fase se realizan los cálculos recursivos de la etapa 7 a la 1, en la cual se deduce la ruta para el óptimo.

Fase 1

Se comienzan a hacer los cálculos por etapas, de la etapa 1 a la 7.

Cálculos para la etapa 1

Como la etapa 1 es la inicial, su valor es igual a cero.

Cálculos para la etapa 2

Para hacer los cálculos se tienen en cuenta las etapas que están conectadas a ella, en este caso es la etapa 1, el valor acumulado y el valor de su arista (tabla 1, figura 4).

Tabla 1. Etapa 2

Etapa	Acumulado	Costo/ arista	Total	Óptimo
1	0	1	1	Óptimo

Fuente: el autor

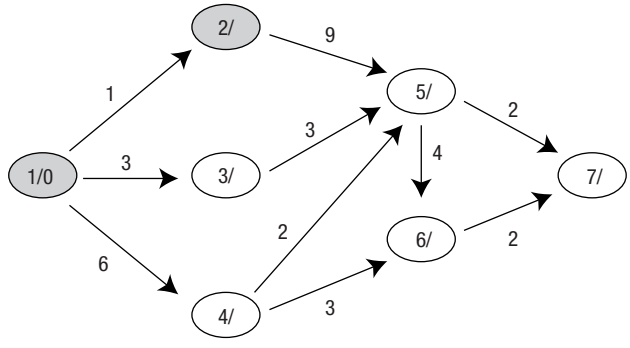

Figura 4. Inventario etapa 2
Fuente: el autor

Cálculos para la etapa 3

Para hacer los cálculos se tienen en cuenta las etapas que están conectadas a ella, en este caso es la etapa 1, el valor acumulado y el valor de su arista (tabla 2, figura 5). Se escoge la que tiene el menor total (óptimo).

Tabla 2. Etapa 3

Etapa	Acumulado	Costo/ arista	Total	Óptimo
1	0	3	3	Óptimo

Fuente: el autor

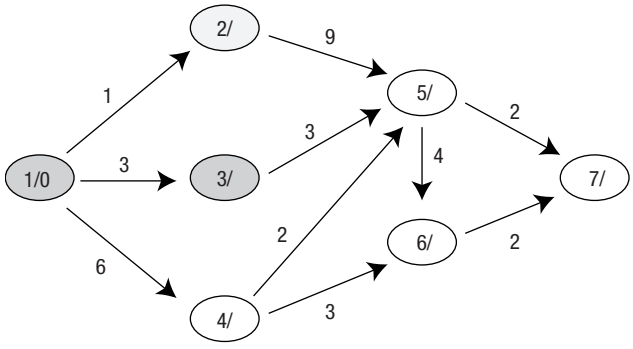

Figura 5. Inventario etapa 3
Fuente: el autor

Cálculos para la etapa 4

Para hacer los cálculos, se tienen en cuenta las etapas que están conectadas a ella, en este caso es la etapa 1, el valor acumulado y el valor de su arista (tabla 3, figura 6). Se escoge la que tiene el menor total (óptimo).

Tabla 3. Etapa 4

Etapa	Acumulado	Costo/arista	Total	Óptimo
1	0	6	6	Óptimo

Fuente: el autor

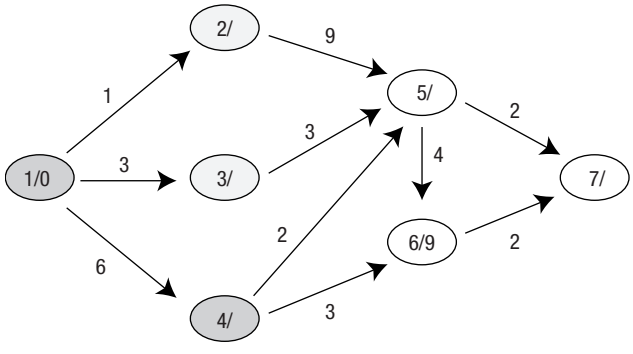

Figura 6. Inventario etapa 4
Fuente: el autor

Cálculos para la etapa 5

Para hacer los cálculos se tienen en cuenta las etapas que están conectadas a ella, en este caso son las etapas 2, 3 y 4, el valor acumulado de la etapa, y el valor de su arista (tabla 4, figura 7). Se escoge la que tiene el menor total (óptimo).

Tabla 4. Etapa 5

Etapa	Acumulado	Costo/arista	Total	Óptimo
2	1	9	10	
4	3	3	6	Óptimo
4	6	2	80	

Fuente: el autor

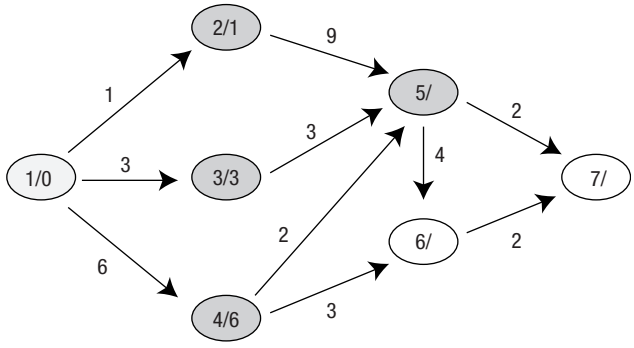

Figura 7. Inventario etapas
Fuente: el autor

Cálculos para la etapa 6

Para hacer los cálculos se tienen en cuenta las etapas que están conectadas a ella, en este caso son las etapas 5 y 4, el valor acumulado de la etapa, y el valor de su arista (tabla 5, figura 8). Se escoge la que tiene el menor total (óptimo).

Tabla 5. Etapa 6

Etapa	Acumulado	Costo/arista	Total	Óptimo
4	6	3	9	Óptimo
5	6	4	10	

Fuente: el autor

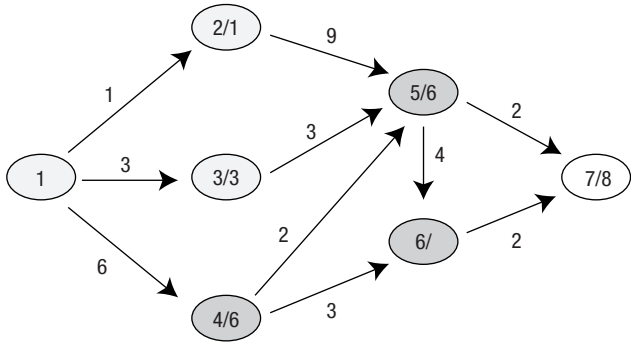

Figura 8. Inventario etapa 6
Fuente: el autor

Cálculos para la etapa 7

Para hacer los cálculos se tienen en cuenta las etapas que están conectadas a ella, en este caso son las etapas 5 y 6, el valor acumulado de la etapa, y el valor de su arista (tabla 6, figura 9). Se escoge la que tiene el menor total (óptimo).

Tabla 6. Etapa 7

Etapa	Acumulado	Costo/arista	Total	Óptimo
5	6	2	8	Óptimo
6	9	1	10	

Fuente: el autor

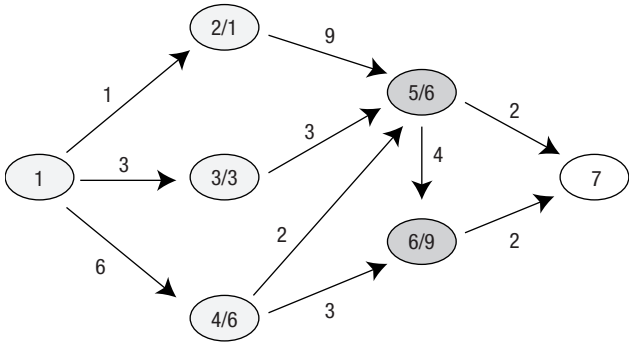

Figura 9. Inventario etapa 7
Fuente: el autor

Fase 2

En forma recursiva nos comenzamos a devolver de la etapa 7 (final) a la etapa 1 (inicial) para establecer la ruta óptima. La solución a la que se llegó es el cálculo de la etapa 7, pero, como podemos ver, para obtener el óptimo de la etapa 7 (figura 10), debemos haber calculado sus anteriores etapas, que son las etapas 5 y 6, y escogemos la menor de las dos (la óptima).

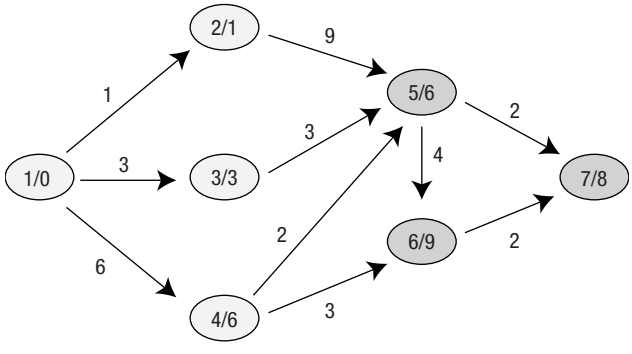

Figura 10. Cálculo etapa 7
Fuente: el autor

Para obtener el óptimo de la etapa 5 (figura 11), debemos haber calculado sus anteriores etapas, que son las etapas 2, 3 y 4, y escogemos el óptimo (la menor).

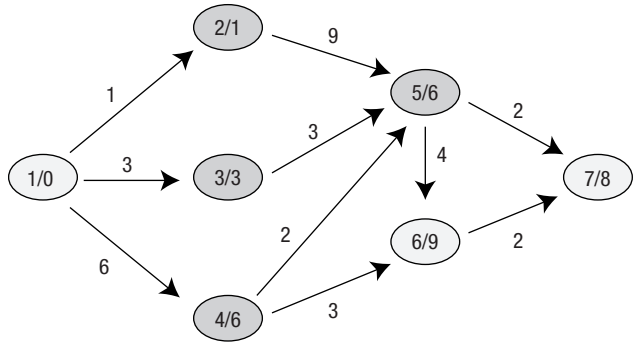

Figura 11. Cálculo etapas
Fuente: el autor

Para obtener el óptimo de la etapa 6 (figura 12), debemos haber calculado sus anteriores etapas, que son las etapas 5 y 4, y escogemos el óptimo (la menor).

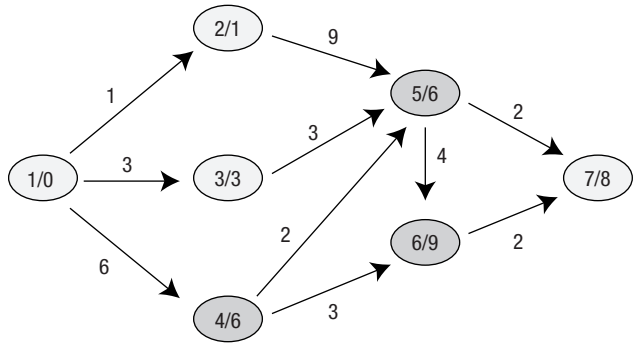

Figura 12. Cálculo etapa 6
Fuente: el autor

Para obtener el óptimo de la etapa 4 (figura 13), debemos haber calculado su etapa anterior, que es la etapa 1 (se escoge por defecto, ya que es la primera.

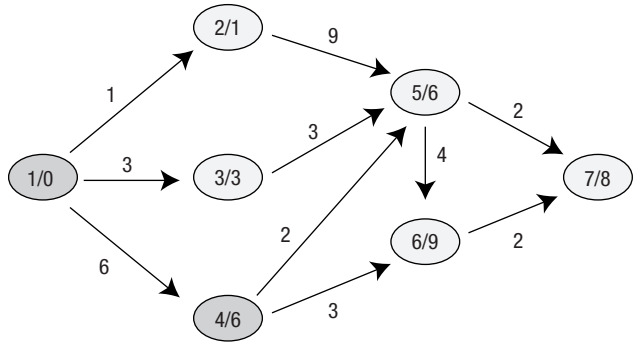

Figura 13. Cálculo etapa 4
Fuente: el autor

Para obtener el óptimo de la etapa 3 (figura 14), debemos haber calculado sus anteriores etapas, que es la etapa 1, y escogemos el óptimo (la menor).

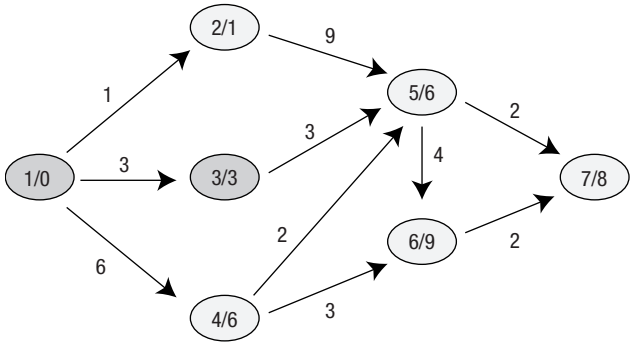

Figura 14. Cálculo etapa 3
Fuente: el autor

Para obtener el óptimo de la etapa 2 (figura 15), debemos haber calculado sus anteriores etapas, que es la etapa 1, y escogemos el óptimo (la menor).

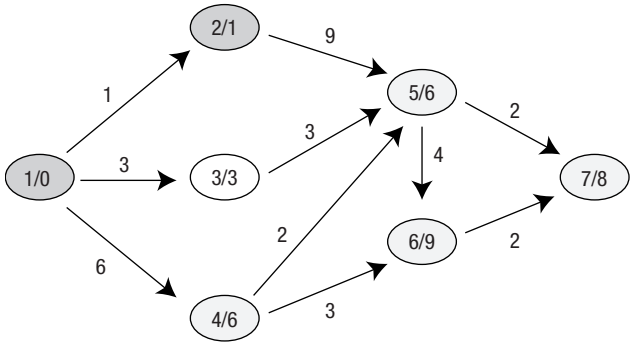

Figura 15. Cálculo etapa 2
Fuente: el autor

Para hallar la ruta crítica [7] del grafo anterior (figura 15), se construyó la representación matemática recursiva siguiente (tabla 7). Dado que va desde la última etapa hasta la primera (en forma recursiva), en este orden se construye el modelo de las funciones matemáticas. Las funciones (nodos) y sus pesos (aristas) se representan de la siguiente manera.

Función (etapa).

Arista (origen, destino).

Tabla 7. Modelo – orden recursivo de las funciones

f(7)	←	menor	{arista(5,7)	+	f(5)
			{arista(6,7)	+	f(6)
f(5)	←		{arista(2,5)	+	f(2)
		menor	{arista(3,5)	+	f(3)
			{arista(4,5)	+	f(5)
f(6)	←	menor	{arista(4,6)	+	f(4)
			{arista(5,6)	+	f(5)
f(4)	←		{arista(1,4)	+	f(1)
f(3)	←		{arista(1,3)	+	f(1)
f(2)	←		{arista(1,2)	+	f(1)
f(1)	←	0	(neutro)		

Fuente: el autor

Los datos de los cálculos anteriores se introducen en el siguiente tabulado (tabla 8), en el que se muestra la ruta más corta del primer nodo (etapa) al último, y del último al primero.

Tabla 8. Cálculo de la ruta más corta

Etapa	f(1)	f(3)	f(5)	f(7)
Peso arista-costo	0	3	3	2
Acumulado	0	3	6	8

Fuente: el autor

Como se puede ver, la construcción del modelo matemático anterior es muy compleja y minuciosa, y hacer los cálculos puede resultar muy enrevesado, sobre todo para personas que no estén familiarizadas con este tipo de soluciones, por lo que se decidió construir un *software* que haga los cálculos automáticamente y que sea fácil de usar.

Para resolver el modelo matemático anterior se utiliza la programación dinámica.

Uso y características de la programación dinámica

Se puede decir que la programación dinámica [8] determina la solución óptima de un problema en

variables, descomponiéndola en etapas, con cada etapa incluyendo un subproblema de una sola variable. La ventaja en el aspecto de los cálculos es que optimizan una sola variable en subproblemas de *n* variables. La principal contribución de la programación dinámica es el principio de optimalidad, un marco de referencia para descomponer el problema en *n* etapas. Los cálculos en la programación dinámica se hacen recursivamente [7], en el sentido de que la solución óptima de un subproblema se utiliza como la entrada para el siguiente subproblema. En muchas ocasiones se usa la simulación para la construcción de los modelos de inventarios [9].

La matriz que se presenta en la tabla 9 almacena los datos del modelo matemático con sus óptimos.

Tabla 9. Matriz- cálculos inventario óptimos

Etapa	1	2	3	4	5	6	7	Óptimo	Etapa
1								0	1
2	1							1	1
3	3							3	1
4	6							6	1
5		9	3	2				6	3
6				3	4			9	4
7					2	1		8	5

Fuente: el autor

- Las columnas (casillas horizontales): 1, 2, 3, 4, 6 y 7 representan el origen de las aristas.
- Las filas (casillas verticales): 1, 2, 3, 4, 5, 6 y 7 son el destino de las aristas.
- Las intercepciones son los costos.
- La columna de los óptimos es el cálculo del mejor valor.
- La columna etapa (nodo) anterior corresponde al cálculo del nodo óptimo.

Como podemos ver, para la etapa 7 el óptimo es igual a 8, que corresponde al acumulado de la etapa 5, que es 6 más el costo, que es 2.

Para obtener el óptimo de la etapa 5, que es 6, debemos hacer el cálculo del acumulado de la etapa 3, que es 3 más el costo, que es 3.

Para obtener el óptimo de la etapa 3, que es 3, debemos hacer al cálculo del acumulado de la etapa 1, que es 0, más el costo, que es 3.

Los cálculos anteriores muestran la ruta óptima de la tabla 7, con la representación gráfica de la figura 16.

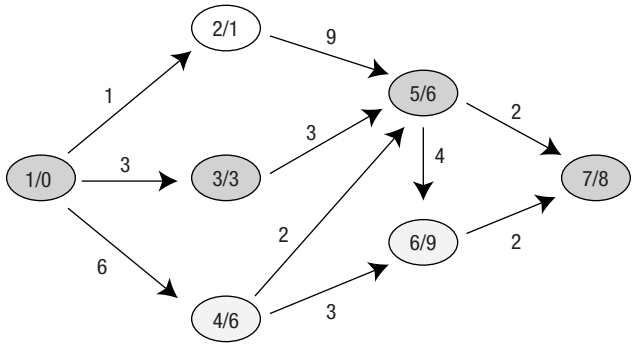

Figura 16. Ruta óptima para los inventarios
Fuente: el autor

Una vez desarrollados los modelos y algoritmos de la programación dinámica se procedió a implementar el modelo en el lenguaje de programación Java (figura 17) (el código se muestra en forma resumida, para facilitar su comprensión, la explicación, está como comentario). Como referente conceptual se utiliza el algoritmo de Dijkstra [10].

```
import java.io.*;
impor java.swing .*;

public class Dijkstra_inventarios
{
 public static void main(String args[])
 {   // las filas son las etapas y las
     // columnas las distancias
   double m[][] = {{0.0.0.0.0.0.0.0.1000.0},
                   {0.0.0.0.0.0.0.0.1000.0},
                   {0.1.0.0.0.0.0.0.1000.0},
                   {0.3.0.0.0.0.0.0.1000.0},
                   {0.6.0.0.0.0.0.0.1000.0},
                   {0.0.9.3.2.0.0.0.1000.0},
                   {0.0.0.0.3.4.0.0.1000.0},
                   {0.0.0.0.0.2.1.0.1000.0}
                  };

    // se recorren las etapas  desde
   //la inicial a la final
 for(int f=2; f<m.length; f++)
  {   // se hacen los calculos de los acumulados
      //y los pesos de las aristas
      for(int c=1; c<(m[0].length-3);  c++)
          {
            if(m[f][c]  != 0)
              {  // se comparan los valores
                 //acumulados y se escoge el
                 //  mejor --
                 if(( m[f][c]  + m[c][m[0].length-2]  )
                                < m[f][m[0].length-2])
                  {
                    m[f][m[0].length-2]  =
                   (m[f][c] + m[c][m[0].length-2]);
                     m[f][m[0].length-1]  = c;
                  }
              }
          }
                        System.out.println("\n");
  }
   System.out.println("-------------------------");
// se inprime la matriz m con los resultados
  imprimir_matriz( m );

}
public static void imprimir_matriz(double k[][])
 {
  for(int f3=0; f3<(k.length); f3++)
      {
        for(int c3=0; c3<(k[0].length); c3++)
          {
            System.out.print( "    " + k[f3][c3]  );}
            System.out.println( "    ");
          }
      }
} // fin del método principal
} // fin de la clase
```

Figura 17. Programa de inventarios
Fuente: el autor

Desarrollo de *Software* Orientado a Objetos (DSOO)

Para desarrollar el prototipo de *software* en consola se usó el paradigma de *Desarrollo de Software Orientado a Objetos* [11] con el RUP, usando el UML para la notación, y el código fuente se escribió en lenguaje de programación Java.

El Proceso Racional Unificado (RUP) consta de las siguientes fases, productos y artefactos [2]:

- *Modelado del negocio*: se planteó el problema y se hizo el análisis de los requerimientos funcionales y no funcionales. A partir del modelo matemático y los objetivos se hizo un análisis de las fortalezas del recurso humano y los recursos disponibles.
- *Requisitos*: se identificaron las tareas y se definieron los responsables.
- *Diseño*: a partir de los requerimientos y los modelos matemáticos se diseñaron los respectivos artefactos (propios del UML).
- *Implementación*: habiendo diseñado todos los artefactos en UML se procedió a traducirlos a un lenguaje de programación.
- *Despliegue*: se efectuó todo lo concerniente a las pruebas de caja blanca y caja negra.

Conceptualización del Lenguaje de Modelamiento Unificado (UML)

Como lo menciona Larman [2]:

> El UML es una notación que se produjo como resultado de la unificación de la técnica de modelado de objetos. Ha sido diseñado para un amplio rango de aplicaciones y actividades enfocándose en tres modelos básicos: *El modelo funcional,* representado en UML con diagramas de casos de uso, describe la funcionalidad del sistema desde el punto de vista del usuario (ver casos de uso). *El modelado de objetos*, representado en UML con diagramas de clases, describe la estructura de un sistema desde el punto de vista de objetos, atributos, asociaciones y operaciones (ver Diagramas de clase). *El modelado dinámico,* representado en UML con Diagramas de secuencia, Diagramas de gráfica de estado y Diagramas de actividades, describe el comportamiento interno del sistema. Los Diagramas de secuencia describen el comportamiento como una secuencia de mensajes intercambiados entre un conjunto de objetos, mientras que los Diagramas de gráficas de estado describen el comportamiento desde el punto de vista de estados de un objeto individual, y las transiciones posibles entre ellos.

A partir de los objetivos se deducen los requerimientos funcionales y no funcionales del producto de *software*.

Requerimientos funcionales

- Realizar las operaciones matemáticas.
- Crear objetos para almacenar y procesar los datos.
- Crear interfaces para la entrada y salida de datos.
- Gestionar archivos.

Requerimientos no funcionales

- El producto de *software* debe ser multiplataforma (correr en los sistemas operativos Windows y Linux).
- El sistema se debe diseñar con compiladores y herramientas case de uso público (Java, NetBeans, etcétera).
- La herramienta debe correr en consola.

Casos de uso

Con los objetivos y los requerimientos se diseñaron los *Casos de uso* (figura 18) (por espacio del documento se presentan los más relevantes). Para su diseño se emplearon los siguientes formularios:

- Crear interfaz (formulario 1)
- Gestionar archivos (formulario 2)
- Realizar cálculos (formulario 3)
- Validar datos (formulario 4)
- Crear objetos matriz de datos (formulario 5)

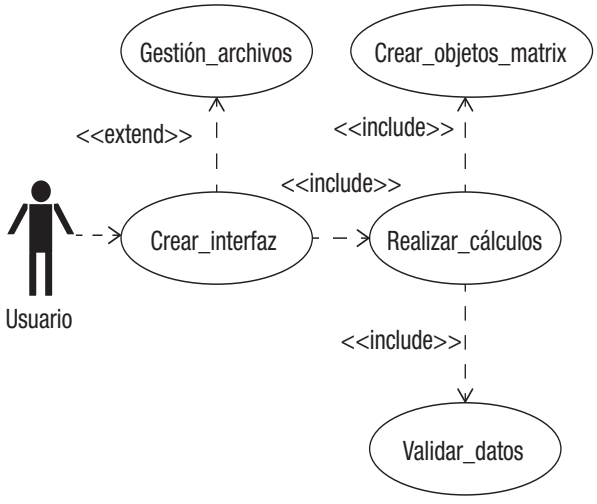

Figura 18. Casos de uso generales
Fuente: el autor

Tabla 10. Formulario 1. Caso de uso: crear interfaz

Caso de uso	
Código	cu1
Nombre	Crear interfaz
Actor	Usuario
Descripción	Acá se digitan todos los datos iniciales y se ejecuta la aplicación.
Referencia	Es el caso de uso primario.
Precondición	Se debe haber instalado la herramienta.
Poscondición	Se presentan los datos en formato tipo hoja de cálculo.
Excepciones	Los objetos deben estar construidos, en caso de error, se muestra un mensaje.
Entrada	Salida
Se pulsan los objetos de la interfaz	Elementos emergentes

Fuente: el autor

Tabla 11. Formulario 2. Caso de uso: gestionar archivos

Caso de uso	
Código	cu2
Nombre	Gestionar archivos
Actor	Usuario
Descripción	Crea la interfaz y presenta las opciones propias de los archivos.
Referencia	cu1
Precondición	La interfaz debe estar abierta. Debe haber conexión a la base de datos.
Poscondición	Datos reales
Excepciones	El sistema se bloquea en caso de no haber conexión.
Entrada -	Salida
Pulsar opción archivos	Se presenta la interfaz con todas las opciones de archivo y gestionan los archivos.

Fuente: el autor

Tabla 12. Formulario 3. Caso de uso: realizar cálculos

Caso de uso	
Código	cu3
Nombre	Realizar cálculos
Actor	Usuario
Descripción	Con los datos abiertos del archivo, o digitados, se realizan los cálculos.
Referencia	cu1
Precondición	Debe estar un archivo abierto, o haber digitado todos los datos.
Poscondición	Se entregan los resultados de los cálculos.
Excepciones	En caso de haber digitado mal un dato, el sistema muestra el error, y no se realizan los cálculos que el error que el dato no se haya corregido.
Entrada	Salida
Se digitan todos los datos de la interfaz y se pulsa el botón “realizar cálculos”.	Aparecen los cálculos realizados en las interfaces. Cálculos costo óptimo. Compra óptima todos los cálculos.

Fuente: el autor

Tabla 13. Formulario 4. Caso de uso: validar datos

Caso de uso	
Código	cu4
Nombre	Validar datos
Actor	Usuario
Descripción	Cuando se van a hacer los cálculos se revisan todos los datos de los campos de texto.
Referencia	cu3
Precondición	Se debe de haber pulsado el botón “realizar cálculos”.
Poscondición	En caso de haber un error, se muestra el mensaje de error.
Excepciones	En caso de que algún dato esté errado, o se sobrepase la capacidad de inventario o compra, se muestra un error.
Entrada	Salida
El usuario digita los datos y pulsa el botón “realizar cálculos”.	En caso de haber un dato errado, el sistema muestra un mensaje de error.
El usuario corrige el dato	Aparecen los cálculos realizados en las interfaces. Cálculos costo óptimo. Compra óptima todos los cálculos.

Fuente: el autor

Tabla 14. Formulario 5. Caso de uso: crear objetos

Caso de uso	
Código	cu5
Nombre	Crear objetos matriz de datos
Actor	
Descripción	Cuando se validan los datos, se deben realizar los cálculos y, dado el gran caudal de datos, se deben almacenar ordenadamente.
Referencia	cu3
Precondición	Los datos deben cumplir con las restricciones.
Poscondición	Se crean los objetos matrices con las dimensiones requeridas.
Excepciones	El sistema se detiene en caso de error.
Entrada	Salida
Dimensiones para los objetos matrices	Creación de los objetos en memoria

Fuente: el autor

Diagrama de clases

Para la construcción de la herramienta se diseñó el siguiente diagrama de clases (figura 19) (se presenta un modelo minimalista), con sus objetos principales:

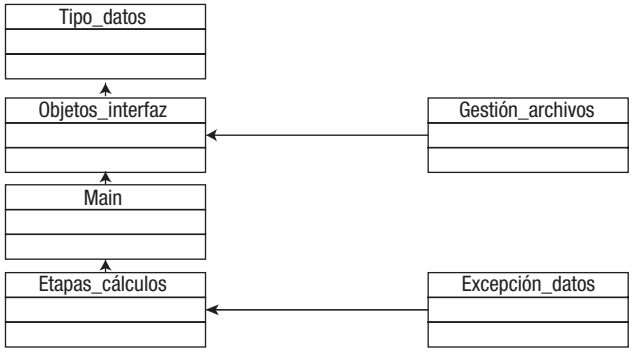

Figura 19. Diagrama de clases
Fuente: el autor

Descripción de los objetos:

- Tipo datos: se denotan todos los tipos de datos que se usarán en el *software*.
- Objetos interfaz: se denotan los objetos y se hace el diseño de la interfaz, con la que se comunicará el usuario.
- Gestión archivos: se agrupan todos los objetos y métodos, y los objetos para los archivos.
- Main: es la clase principal desde donde se ejecuta todo el *software*.
- Etapas cálculos: es la clase en donde se ejecutan todos los cálculos del *software*.
- Excepción_datos: se prueban los datos de entrada y los cálculos de los parámetros.

Diagrama de secuencia

Se presenta el diagrama de secuencia correspondiente al caso de uso realizar cálculos (figura 20). La lectura debe hacerse de izquierda a derecha y de arriba hacia abajo.

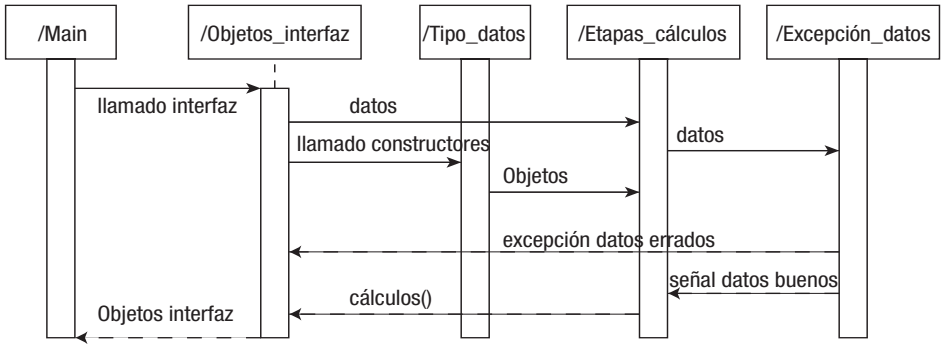

Figura 20. Diagrama de secuencia realizar cálculos
Fuente: el autor

Resultados y conclusiones

Los resultados de esta investigación fueron los siguientes:

- La primera versión del *software* corriendo en consola, que hace los cálculos del costo óptimo de los inventarios de entubamiento.
- El diseño de los programas y algoritmos en Java, así como los modelos matemáticos que los sustentan.
- Los artefactos propios del RUP y los diagramas del UML, como son: modelado del negocio, requerimientos, diagramas de casos de uso, diagrama de clases, etcétera.
- En este documento se describe cómo se deducen e integran los programas, modelos matemáticos, etcétera, con los conceptos propios de la computación, como son estructura de datos, programación orientada a objetos, teoría de grafos, ingeniería de *software*, etcétera, y los conceptos de ingeniería industrial, como son investigación de operaciones, modelamiento matemático de sistemas, manejo de inventarios, etcétera.

Para los efectos del presente proyecto se comprobó que cuando se diseña un *software* con un alto componente matemático, este se debe hacer en dos fases:

- Primera fase: el problema se debe plantear en términos matemáticos y diseñar un modelo con todas sus variables y funciones muy bien definidas. Los datos se deben almacenar en estructuras matriciales. Una vez concluida la fase del modelamiento matemático, se da inicio a la segunda fase.
- Segunda fase: con los modelos matemáticos y las estructuras matriciales que se han diseñado, se aplica el paradigma de desarrollo en retroceso y se implementa el RUP con el UML de la siguiente manera: para el modelado del negocio, se parte del modelo matemático, en el cual los procesos son los diferentes cálculos y funciones que debe realizar el *software* y la forma de manipular los datos, así como el diagrama de procesos; para los requisitos, se identifican las tareas que deben realizar los miembros del proyecto; para el diseño del *software*, con los procesos del *software* definidos y los modelos matemáticos, se hace los requerimientos funcionales y no funcionales, se diseñan los casos de uso, el diagrama de clases, etcétera; para la implementación, habiendo construido los diagramas y artefactos, se hace la traducción al lenguaje de programación (en el proyecto se escogió el Java); para el despliegue, se hace todo lo concerniente a las pruebas de caja negra, caja blanca, manual del usuario, etcétera.

La programación dinámica se debe usar para fenómenos discretos que ocurren en un tiempo y un espacio. Cuando se plantea un problema en el contexto de la programación dinámica, su solución se hace con modelos y estructuras computacionales recursivas.

Aportes

- El producto de *software* es una herramienta muy intuitiva y funcional que puede ser operada por cualquier persona que tenga un conocimiento básico en computación e inventarios.
- El proyecto ha permitido que los estudiantes de la Universidad Cooperativa de Colombia pongan en práctica los conceptos vistos en clase, como teoría de grafos, estructura de datos, algoritmia, programación, ingeniería de *software*, etcétera; además ha permitido que tengan la oportunidad de diseñar, construir y probar un producto de *software* en todas sus etapas.
- Este proyecto ha sido una gran oportunidad para que los estudiantes de la sede de Cali representen a la Universidad en eventos de investigación regional y nacional, como los encuentros de semilleros de investigación, con muy buenos resultados.
- A lo largo del proyecto se ha contribuido a fomentar la cultura investigativa en la Universidad Cooperativa de Colombia.

Reconocimiento

Se agradecen los aportes realizados por la ingeniera Josefina de Llano (directora de investigaciones de la Facultad de Ingenieria de la sede de Cali), en cuanto a la revisión preliminar del documento, así como al doctor José Antonio Carvajal Lombana (decano de la Facultad de Ingenieria de la sede de Cali), por el apoyo brindado al proyecto de investigación.

Referencias

[1] H. Deytel. *Cómo programar en Java*. Quinta edición. México: Prentice Hall, 2004.

[2] C. Larman. *UML y patrones*. Segunda Edición. España: Prentice Hall, 2008.

[3] T. Vollman. *Administración integral de la producción e inventarios*. Primera edición. España: Limusa, 2000.

[4] R. Chase. *Producción y operaciones*. Décima edición. México: McGraw-Hill, 2005.

[5] W. Winstom. *Investigación de operaciones*. Cuarta edición. México: Thomsom, 2005.

[6] R. Kenneth. *Matemáticas discretas y sus aplicaciones*. Quinta edición. México: McGraw-Hill, 2004.

[7] R. De la Llave. *Dynamics of algorithms*. Primera edición. Londres: Springer, 2003.

[8] A. Lew. *Dynamic Programing*. Primera edición. Berlin: Springer, 2007.

[9] B. Coss. *Simulación un enfoque práctico*. Segunda edición. México: Limusa, 2005.

[10] R. Pressman. *Ingeniería de software*. Séptima edición. México: McGraw-Hill, 2010.