

**UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS  
FACULTAD 9**



**TÍTULO:** Desarrollo de Biblioteca de Estructuras de Datos Avanzadas  
(Árboles, Grafos, Heaps, Matrices Poco Densas).

**TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN  
INFORMÁTICA**

**AUTORES:** Víctor Frank Molina López  
José Alberto Fernández Gómez

**TUTOR:** Lic. José Ángel Lago Graverán

**CO-TUTORES:** Ing. Yancy Martínez Pérez  
Lic. Yusnier Valle Martínez  
Ing. Alexey Díaz Domínguez

**Ciudad de La Habana Julio, 3 2008.  
Año 50 del Triunfo de la Revolución**

## **DEDICATORIA**

*A mi madre,  
la fuente principal de mis realizaciones...*

*A mi padre,  
aunque lejos pero constante...*

*A los que pusieron toda su confianza en mí...*

*A mi tika ninda...*

*... José Alberto*

## **DEDICATORIA**

*A mi madre,  
la culpable principal de todos mis logros...*

*A todos los que estuvieron ahí para mí cuando los necesité  
y a los que no también...*

*A ti, que aunque no aparezca tu nombre,  
eres parte importante de este logro...*

*...Victor Frank*

# Desarrollo de Biblioteca de Estructuras de Datos Avanzadas (Árboles, Grafos, Heaps y Matrices Poco Densas)

---

## AGRADECIMIENTOS

Hasta la más sencilla de las obras no es posible sin el apoyo brindado a quien la realiza; y hacerse profesional, que sin dudas no es sencillo, hubiera sido imposible sin el apoyo de un enorme grupo de personas, por eso quiero agradecer a todos los que de alguna forma contribuyeron a que cumpliera el más grande objetivo que tuviera como estudiante y que me ha ayudado a crecer en todos los aspectos de la vida. Quisiera agradecer de forma especial:

- A mi madre Carmen Victoria, quien ha sido y será la mejor profesora que cualquier hijo soñara tener.
- A mis abuelos, los maternos, los paternos y los postizos, quienes aportaron sus semillas en mi crecimiento.
- A mi padre, para que vea cuanto he crecido y se enorgullezca.
- A mi tío, quien siempre se ha preocupado por mi formación.
- A quienes me sirvieron de guía en todas mis acciones y aún siguen haciéndolo.
- A mis amigos más cercanos, dentro y fuera de la UCI (ustedes saben quienes son), por dejarme ser parte de sus vidas.
- A José, el mejor compañero de tesis que se pudiera tener, quien con su exigencia hizo posible la realización de esta tesis.
- A Lago, mi tutor, por idear esta tesis y darme la posibilidad de sentir el gusto de trabajar en ella.
- A quienes supieron desconectarme del trabajo y contribuyeron a no estresarme.
- A mis profesores, los que me la pusieron difícil y los que no...
- A la Revolución Cubana, que me proporcionó los medios y las opciones, para que escogiera el camino que quisiera.
- A mi madre Lilliam E. por su constante apoyo y mente positiva. Mejor... ni mandándola a hacer.
- A mi abuelita por su mirada constante en mi, extendiendo este agradecimiento al resto de los integrantes de mi árbol genealógico (un día de estos mi objetivo será programarlo).
- A Lago, tutor y amigo, gracias por contar con nosotros, por las brillantes ideas que nos diste.
- A Karenia Jimenez. Gracias por todo mi tika.
- A Vic, eficiente compañero de tesis y amigo.
- A los integrantes de UCISTORE, Vela y Toscano por su manera de liberar mi estrés creando música.
- A Alexander Osorio por las veces que me despertó de la mesa de la pc y me enseñó a programar. Hermano, valió la pena Maisí.
- A Yinet Gaínza, el centro de mis amigos desde que la conozco. Uff Yi, cuánto hace!!!
- A Yari, Yula, Yuya y el equipo que formamos para todas las evaluaciones. Todos más que amigos, hermanos.
- A Alden Hernández Gómez por su ayuda con Python y la pc cómplice de laargas noches.
- Al Comandante en Jefe y a la Revolución por todo un sueño hecho realidad.

...Victor Frank

... José Alberto

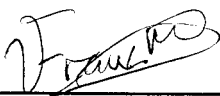
**Desarrollo de Biblioteca de Estructuras de Datos Avanzadas  
(Árboles, Grafos, Heaps y Matrices Poco Densas)**

---

**DECLARACIÓN DE AUTORÍA**

Nosotros, Víctor Frank Molina López y José Alberto Fernández Gómez declaramos que somos los únicos autores de este trabajo y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo.

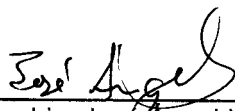
Para que así conste firmamos la presente a los \_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.



\_\_\_\_\_  
Víctor Frank Molina López  
Autor



\_\_\_\_\_  
José Alberto Fernández Gómez  
Autor



\_\_\_\_\_  
Lic. José Ángel Lago Graverán  
Tutor

## **DATOS DE CONTACTO**

**Síntesis del Tutor:** Graduado con Título de Oro de Licenciado en Ciencia de la Computación, Universidad de la Habana año 2004. Concursante de Matemática desde la Secundaria Básica. Ha obtenido resultados a nivel nacional (Medalla de Plata, Bronce y Oro). Cursó estudios en la Universidad de la Habana, aquí integró el grupo UHSIS. En este grupo participó como desarrollador de cuatro productos, los cuales están registrados. Fue integrante del grupo de Geometría Computacional de la facultad de Matemática-Computación. Ha participado en varios eventos nacionales e internacionales, en algunos de ellos ha publicado trabajos. Actualmente trabaja como profesor en la UCI, en la cual ha impartido Matemática Numérica y Gráfico por Computadoras, además de estar vinculado a la producción.

## Desarrollo de Biblioteca de Estructuras de Datos Avanzadas (Árboles, Grafos, Heaps y Matrices Poco Densas)

---

### OPINIÓN DEL TUTOR

El tutor del presente Trabajo de Diploma considera que durante su ejecución los estudiantes mostraron las cualidades que a continuación se detallan:

Ambos estudiantes presentaron una actitud correcta al asumir con un alto grado de responsabilidad todas las tareas realizadas, con un nivel muy alto de independencia en cada una de las actividades, además de alcanzar un profundo dominio del campo donde se realizó la investigación. En la documentación y herramienta obtenidas como resultado se puede apreciar originalidad, creatividad y un alto nivel de responsabilidad por parte de los autores. Es importante destacar que la mayoría de las estructuras de datos abordadas en este trabajo son desconocidas en el ámbito de nuestra universidad, por tal motivo los resultados alcanzados son primer paso para su conocimiento, estudio y posterior uso en proyectos productivos.

En el trabajo realizado se puede apreciar una alta calidad científico-técnica. Los resultados obtenidos presentan un elevado nivel de rigurosidad y formalismo, lo cual corrobora que el objetivo propuesto al inicio de la investigación fue alcanzado.

Por todo lo anteriormente expresado se considera que ambos estudiantes están aptos para ejercer como Ingeniero en Ciencias Informáticas, y se propone:

Calificación máxima de 5 puntos al diplomante Jose Alberto Fernández Gómez.

Calificación máxima de 5 puntos al diplomante Víctor Frank Molina López.

José Angel Lago Graverán

  
Firma

  
Fecha

# **Desarrollo de Biblioteca de Estructuras de Datos Avanzadas (Árboles, Grafos, Heaps y Matrices Poco Densas)**

---

## **RESUMEN**

En la actualidad la mayoría de las aplicaciones informáticas de éxito utilizan Estructuras de Datos Avanzadas con algoritmos eficientes que le garantizan un óptimo funcionamiento. Problemas complejos como la administración de un gran conjunto de datos y su almacenamiento en ficheros, la minimización de costos para construir infraestructuras y cálculos complejos en la simulación de procesos en las diferentes industrias, etc., presentan soluciones eficaces gracias a las estructuras de datos avanzadas.

El presente trabajo propone un aporte con el fin de proveer estas soluciones agrupadas en una biblioteca bajo el concepto de Software Libre, que pretende con su implementación servir de ayuda y soporte al desarrollo de sistemas más competentes y efectivos, tanto para nuestra universidad como para nuestro país y el mundo . Se expone la selección de las tecnologías y herramientas tenidas en cuenta a la hora del desarrollo de la solución propuesta que responde al problema que se plantea.

Se presenta la solución haciendo uso de la Ingeniería de Software vinculada a la metodología seleccionada a través del análisis y diseño de la biblioteca implementada. Se ofrece además una descripción detallada de cada una de las estructuras de datos avanzadas propuestas en dicha solución, recogidas en una documentación adjunta a las implementaciones.

### **PALABRAS CLAVES**

Biblioteca, solución, problemas complejos, Estructuras de Datos Avanzadas, Software Libre, Ingeniería de Software, documentación adjunta, implementación.



## **INDICE DE TABLAS Y FIGURAS**

<b>Figura 1: Grafos que son estructuras tipo árbol binario.....</b>	<b>9</b>
<b>Figura 2: Grafos que no son árboles binarios .....</b>	<b>10</b>
<b>Figura 3: Rotación a la derecha en un AVL .....</b>	<b>12</b>
<b>Figura 4: Rotación a la derecha en un AVL .....</b>	<b>12</b>
<b>Figura 5: Grafo con camino AJLOE y conexo .....</b>	<b>15</b>
<b>Figura 6: Grafo con camino AJLOE y no conexo .....</b>	<b>15</b>
<b>Figura 7: Grafo a) .....</b>	<b>20</b>
<b>Figura 9 a): Árbol Binomial .....</b>	<b>22</b>
<b>Figura 9 b): árboles binomiales desde <math>B_0</math> hasta <math>B_4</math>. .....</b>	<b>22</b>
<b>Figura 9 c) Árboles Binomiales .....</b>	<b>23</b>
<b>Figura 10: Heap de Fibonacci .....</b>	<b>24</b>
<b>Figura 11: Cuatro max pairing Heaps representados ordenadamente. ....</b>	<b>25</b>
<b>Figura 12: Max pairing heap .....</b>	<b>25</b>
<b>Figura 13: Fases de XP .....</b>	<b>39</b>
<b>Figura 14: Fases de FDD.....</b>	<b>41</b>
<b>Figura 15:Fases de RUP. ....</b>	<b>43</b>
<b>Figura 16: Diagramas de clases del Modelo de Dominio .....</b>	<b>52</b>
<b>Figura 17: Diagrama de Casos de Uso del Sistema .....</b>	<b>55</b>
<b>Figura 18: Diagrama General del Diseño organizado por paquetes. ....</b>	<b>73</b>
<b>Figura 19: Diagrama de Clases del Diseño del paquete Nodo. ....</b>	<b>74</b>
<b>Figura 20: Diagrama de Clases del Diseño del paquete Árbol. ....</b>	<b>75</b>
<b>Figura 21: Diagrama de Clases del Diseño del paquete Grafo. ....</b>	<b>76</b>
<b>Figura 22: Diagrama de Clases del Diseño del paquete Heap.....</b>	<b>77</b>
<b>Figura 23: Diagrama de Clases del Diseño del paquete Matriz. ....</b>	<b>78</b>
<b>Figura 24: Diagrama de componentes del Modelo de Implementación. ....</b>	<b>79</b>
<b>Tabla 1. Principales Características de Rational Rose Enterprise .....</b>	<b>34</b>
<b>Tabla 2. Actores del sistema a automatizar.....</b>	<b>55</b>
<b>Tabla 3. Descripción del Caso de Uso Gestionar elementos en un Árbol. ....</b>	<b>56</b>
<b>Tabla 4. Descripción del Caso de Uso Gestionar elementos en un Heap.....</b>	<b>60</b>
<b>Tabla 5. Descripción del Caso de Uso Gestionar elementos en una Matriz.....</b>	<b>62</b>
<b>Tabla 6. Descripción del Caso de Uso Gestionar elementos en un Grafo. ....</b>	<b>65</b>
<b>Tabla 7. Pruebas realizadas a los Casos de Uso Gestionar Elementos en un Heap y Gestionar Elementos en un Grafo.....</b>	<b>80</b>

# Desarrollo de Biblioteca de Estructuras de Datos Avanzadas (Árboles, Grafos, Heaps y Matrices Poco Densas)

---

## INDICE

INTRODUCCIÓN .....	1
CAPÍTULO I: Fundamentación Teórica .....	4
1.1 Introducción. ....	4
1.2 Conceptos asociados al dominio del problema.....	4
1.3 Objeto de Estudio. ....	6
1.4 Descripción general del objeto de estudio. ....	6
1.4.1 Árboles: .....	7
1.4.2 Grafos. ....	14
1.4.4 Matrices poco densas.....	25
1.5 Análisis de otras soluciones existentes. ....	27
1.6 Conclusiones parciales.....	29
CAPÍTULO II: Tendencias y tecnologías actuales a desarrollar .....	30
2.1 Introducción. ....	30
2.2 Las Tecnologías de la Información y las Comunicaciones (TIC).....	30
2.3 Desarrollo de bibliotecas de clases. ....	30
2.4 El Lenguaje Unificado de Modelado (UML) como soporte a la Programación Orientada a Objetos.....	31
2.4.1 Visual Paradigm como herramienta CASE para el modelamiento con UML. ....	33
2.5 El Proceso Unificado de Desarrollo de Software (RUP) como base en el desarrollo de la solución.....	38
2.6 ¿Por qué Python como lenguaje de programación? .....	45
2.7 ¿Por qué IDLE Python 2.5 y PyScripter como IDEs para la programación de la biblioteca?.....	49
2.8 Conclusiones parciales .....	50
CAPÍTULO III: Presentación de la solución propuesta .....	51
3.1 Introducción. ....	51
3.2 Entorno donde desarrollará el sistema.....	51
3.2.1 Diagrama de clases del Modelo de Dominio. ....	52
3.2.2 Glosario de Términos del Dominio.....	52
3.3 Funcionalidades del sistema propuesto.....	53
3.3.1 Requerimientos Funcionales (RF).....	53
3.3.2 Requerimientos No Funcionales (RNF).....	54
3.4 Descripción del sistema propuesto. ....	55
3.4.1 Descripción de los actores del sistema a automatizar. ....	55

## **Desarrollo de Biblioteca de Estructuras de Datos Avanzadas (Árboles, Grafos, Heaps y Matrices Poco Densas)**

---

3.4.2 Diagramas de casos de uso del sistema a automatizar. ....	55
3.5 Conclusiones parciales.....	69
<b>CAPÍTULO IV: Construcción de la solución propuesta.....</b>	<b>70</b>
4.1 Introducción. ....	70
4.2 Patrones.....	70
4.2.1 Patrones de arquitectura utilizados. ....	70
4.2.2 Patrones de diseño utilizados. ....	71
4.3 Diagramas de clases del diseño.....	73
4.3.3 Paquete Grafo .....	76
4.3.4 Paquete Heap. ....	77
4.3.5 Paquete Matriz.....	78
4.4 Modelo de implementación. ....	79
4.4.1 Diagrama de Componentes.....	79
4.5 Pruebas.....	79
4.6 Conclusiones parciales.....	86
<b>RECOMENDACIONES.....</b>	<b>88</b>
<b>REFERENCIAS BIBLIOGRÁFICAS .....</b>	<b>89</b>
<b>GLOSARIO.....</b>	<b>94</b>

## INTRODUCCIÓN

Las estructuras de datos constituyen la formalización de un determinado tipo de dato, el entorno donde este existe y realiza sus procedimientos y funciones (1). Tienen el objetivo de facilitar la manipulación del mismo, ya que no son más que la agrupación de variables sobre las que se han implementado las operaciones definidas para el tipo de dato y que permite finalmente su implementación.

En las carreras de Ingeniería Informática o similares, es casi de estudio obligatorio este tema, pero solo se estudian las estructuras de datos más comúnmente utilizadas (Listas, Pilas, Colas, Árboles y Grafos sencillos [L, P, C, A, G]), lo cual sirve para adentrarse en este interesante tópico y resolver determinados problemas con su uso.

Este tipo de estructuras han sido implementadas a nivel mundial en otros lenguajes diferentes de Python. Este lenguaje incluye las listas, diccionarios y tuplas, aunque se ha podido comprobar la implementación de Árboles Binarios y Árboles Binarios de Búsqueda que resuelven problemas e intereses específicos de clientes. Otros lenguajes de programación como C++, C# y Java, por solo mencionar algunos, tienen incluidas bibliotecas de estructuras de datos pero nunca tan abarcadoras en diversidad, puesto que solo están implementadas las estructuras de datos más simples [L, P, C, A, G].

¿Qué sucedería si los problemas a resolver fueran tan complejos como representar terrenos o implementar algoritmos para gráficos por computadoras, los cuales necesitan manejar una gran cantidad de información en memoria de forma óptima?, las estructuras simples y más conocidas no serían una solución factible en muchos de los casos. Por tal motivo sería de gran utilidad contar con otras estructuras de datos que tengan un mejor aprovechamiento de la memoria y/o mayor eficiencia en la realización de determinadas operaciones, de esta forma se pudiera diseñar algoritmos óptimos para determinados problemas. En este trabajo se brinda la implementación de Estructuras de Datos Avanzadas, las cuales son óptimas para ser utilizadas en problemas con características especiales, ej: grandes volúmenes de datos, gran cantidad de operaciones de almacenamiento, eliminación y consulta de elementos, etc.

Sobre este tipo de estructuras existe documentación tanto a nivel mundial como en esta universidad, pero no se ha implementado aún una biblioteca que recoja gran cantidad de ellas y cuando hace falta utilizarlas hay que implementarlas por separado una y otra vez,

de ahí que el **problema científico** sea justamente esto: “la poca disponibilidad de una biblioteca que agrupe estructuras de datos avanzadas”.

Por lo que para resolverlo es necesario adentrarse en el estudio de *las Estructuras de Datos*, constituyendo estas el **objeto de estudio** de la investigación. El **campo de acción** serían las estructuras de datos avanzadas, específicamente: Árboles, Grafos, Heaps y Matriz Poco Densa.

A partir de las ideas anteriormente expuestas, el **objetivo general** de este trabajo sería: “Desarrollar una Biblioteca de Estructura de Datos Avanzadas en una plataforma de Software Libre y generar un documento de ayuda para el usuario de dicha biblioteca”.

Por lo que queda planteada la siguiente **hipótesis**:

“Si se documenta y desarrolla esta Biblioteca de EDA en una plataforma de Software Libre, es posible que la universidad pueda utilizarla con todas las ventajas que brinda esta filosofía “

Para el desarrollo y término satisfactorio del trabajo de diploma se han desglosado las siguientes **tareas de la investigación**:

1. Estudiar el estado del arte.
2. Escoger herramientas para el desarrollo de la biblioteca
3. Realizar el análisis, diseño e implementación de la biblioteca.
4. Realizar pruebas a la biblioteca
5. Elaborar documentación de la biblioteca.

El producto final sería una biblioteca que agrupe estructuras de datos avanzadas y la documentación asociada a ella, lo cual tendría gran utilidad tanto en la universidad como en cualquier parte del mundo, ya que como se ha dicho, nunca se ha logrado agrupar las implementaciones de estas estructuras de datos avanzadas en una biblioteca y la documentación adjunta podría ser utilizada como guía, aun cuando no se utilicen nuestras implementaciones, ya que en ellas están detalladas las funcionalidades y aplicaciones de cada una de las estructuras de datos incluidas en nuestra biblioteca. Además, está desarrollada en un entorno de Software Libre, lo cual aprovecha todas las ventajas que esta filosofía brinda, como el acceso al código fuente para su posterior mejora y sobre todo lo que implica para la realidad cubana que cada vez se aproxima más a dejar el Software Propietario.

**Métodos de Investigación.**

Los métodos de investigación que se utilizaron para la realización de este trabajo de diploma fueron teóricos y empíricos.

Dentro los teóricos utilizados se encuentra el sistémico por la construcción de los diagramas de flujos de actividades, el histórico por el estudio de los antecedentes de bibliotecas que contengan estructuras de datos avanzadas y el de modelación por la creación de un modelo que se ajuste a nuestro trabajo. Mientras que los empíricos utilizados fueron las entrevistas, por las realizadas en los diferentes proyectos productivos que han utilizado estructuras de datos avanzadas y otros para consultar libros, sitios webs y artículos concernientes al tema.

## CAPÍTULO I: Fundamentación Teórica

### 1.1 Introducción.

En este capítulo se brinda una descripción de los principales conceptos asociados al dominio del problema, profundizando en la investigación realizada acerca del estudio del estado del arte de las estructuras de datos de forma general. Además se ofrece una explicación general del objeto de estudio, particularizando después en las estructuras de datos avanzadas que se utilizan como posible solución al problema.

### 1.2 Conceptos asociados al dominio del problema.

Para un mejor entendimiento del trabajo de diploma, son ofrecidos algunos conceptos necesarios que serán de utilidad a la hora de adentrarse en terminologías técnicas de la especialidad, los mismos son:

#### **Programación Orientada a Objetos (POO):**

Es una forma especial de programar, más cercana a como se expresarían las cosas en la vida real que en otros tipos de programación. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo y encapsulamiento.

Con la POO tenemos que aprender a pensar las cosas de una manera distinta, para escribir programas en términos de objetos, propiedades y métodos de forma general. En otras palabras la POO es la modelación literal de los problemas de la vida real. (1)

#### **Lenguajes orientados a objetos:**

Son lenguajes basados en sentencias y en clases (algunos, llamados mixtos soportan también el modelo procedural). Entre los lenguajes orientados a objetos puros podemos nombrar a Smalltalk, Eiffel y Java. Entre los mixtos se encuentran C++ y Python.

#### **Tipo de Dato:**

Es la clasificación que se le puede dar a los datos de acuerdo con sus características y funcionalidades.

### Tipo de Datos Abstractos (TDA):

La abstracción es el mecanismo que permite seleccionar partes de un todo complejo para su consideración, ignorando el resto. Esto permite filtrar aquellos aspectos relevantes, y obtener soluciones más generales.

El concepto de abstracción nos remite a la posibilidad de considerar la resolución de un problema sin tener en cuenta los detalles por debajo de cierto nivel.

El concepto de Tipo de Dato Abstracto (TDA en español, 'Abstract Data Type' o ADT en inglés) surgió para facilitar el trabajo con tipos de datos haciendo abstracción de la implementación de los mismos.

Un TDA está dado por un grupo de datos que cumplen cierta condición especificada para él, más un conjunto de operaciones que representan su comportamiento. La entidad TDA exporta las características de forma de las operaciones. La implementación de la estructura del TDA, y de las operaciones propias son privadas.

No se permite acceso ni visibilidad a la implementación de un TDA. Esta parte oculta está constituida por la maquinaria algorítmica que implementa la semántica de los operadores. De este modo un TDA encapsula ciertos tipos de datos en cuanto a la definición del tipo y todas las operaciones del mismo en una sección del código.

Los TDA constituyen una forma de generalización y encapsulamiento de los aspectos más importantes de la información que se debe manejar en la resolución de un problema, sin considerar las cuestiones relativas a la implementación.

Un TDA define una nueva clase de concepto que puede manejarse con independencia de la estructura de datos para representarlo. Es una generalización de los tipos de datos básicos y de las operaciones primitivas.

Es decir:

TDA = Representación (estructuras de datos) + Operaciones (métodos)

Los TDA pueden clasificarse en:

Simples: son aquellos que cambian su valor pero no su estructura (el espacio de almacenamiento es constante)



Contenedores: son aquellos que cambian su valor y estructura (son conjuntos o colecciones de elementos de número variable) Es frecuente definir TDA Contenedor como una colección de objetos más un conjunto de métodos para gestionarlos (permite realizar altas y bajas de elementos en el contenedor, acceder a ellos, etc.).

Entonces, considerando que los tipos simples también se corresponden con determinados TDA, se puede decir que un TDA es un contenedor si contiene varias instancias de otros TDA. (2)

### 1.3 Objeto de Estudio.

Una vez hechas las investigaciones y el análisis del problema, se definen **las estructuras de datos** como el objeto de estudio, el cual se describe a continuación.

### 1.4 Descripción general del objeto de estudio.

#### **Estructura de Datos:**

Una estructura de datos es un conjunto de dominios  $\delta$ , un dominio designado  $d \in \delta$ , un conjunto de funciones  $F$  y un conjunto de axiomas  $\Delta$ . La terna  $(\delta, F, \Delta)$  denota la estructura de datos «d» y será abreviada por escrito como «d».

El conjunto de axiomas describe la semántica de las operaciones. La forma en que elegimos para escribir los axiomas es importante. Nuestro objetivo es escribir los axiomas como una representación en forma independiente. A continuación, se discute la forma de aplicar las funciones utilizando un lenguaje de programación convencional. Una implementación de una estructura de datos «d» es el mapeo de «d» a un conjunto de otras estructuras de datos «e». Este mapeo especifica de qué manera cada objeto de «d» es representado por los objetos de «e». En segundo lugar, se requiere que cada función de «d» debe ser escrita usando las funciones de la aplicación de estructuras de datos «e». Por lo tanto, podemos decir que los enteros son representados por cadenas de bits, los booleanos por ceros y unos, un conjunto está representado por un conjunto de palabras consecutivas en la memoria.

En la jerga actual terna  $(\delta, F, \Delta)$  se refiere a un tipo de datos abstracto. Se llama abstracto, porque los axiomas no implican una forma de la representación. Otra forma de ver la aplicación de una estructura de datos constituye el proceso de refinado de un tipo de datos abstracto hasta que todas las operaciones son expresables en términos de

funciones directamente ejecutables. Pero en la primera etapa, una estructura de datos debe diseñarse de modo que se sepa lo que hace, pero no necesariamente la forma en que se hará. Esta división de tareas, llamada especificación y puesta en práctica, es útil porque ayuda a controlar la complejidad de todo el proceso.

Las operaciones básicas que pueden efectuarse en una estructura de datos son:

- **Insertar:** adicionar un nuevo valor a la estructura.
- **Eliminar:** borrar un valor de la estructura.
- **Buscar:** encontrar un determinado valor en la estructura para realizar una operación con este valor.

Otras operaciones que se pueden realizar son:

- **Ordenamiento:** ordenar los elementos pertenecientes a la estructura.
- **Concatenación:** dadas dos estructuras originar una nueva ordenada y que contenga a las concatenadas.

## Estructura de Datos Avanzada (EDA):

Una estructura de datos avanzada es aquella estructura de datos que se puede derivar o no de una simple y que su complejidad y/o eficiencia para resolución de determinados problemas la hacen diferenciarse del resto.

### 1.4.1 Árboles:

Un árbol es una estructura de datos ampliamente usada que emula la forma de un árbol y constituye un conjunto de nodos conectados. Un nodo es la unidad sobre la que se construye el árbol y puede tener cero o más nodos hijos conectados a él. Se dice que un nodo  $a$  es padre de un nodo  $b$  si existe un enlace desde  $a$  hasta  $b$  (en ese caso, también decimos que  $b$  es hijo de  $a$ ). Sólo puede haber un único nodo sin padres, que llamaremos raíz. Un nodo que no tiene hijos se conoce como hoja.

Formalmente, podemos definir un árbol de la siguiente forma recursiva:

- Caso base: un árbol con sólo un nodo (es a la vez raíz del árbol y hoja).
- Un nuevo árbol a partir de un nodo  $n_r$  y  $k$  árboles  $A_1, A_2, \dots, A_k$  de raíces  $n_1, n_2, \dots, n_k$  con  $N_1, N_2, \dots, N_k$  elementos cada uno, puede construirse

estableciendo una relación padre-hijo entre  $n_r$  y cada una de las raíces de los  $k$  árboles. El árbol resultante de  $N = N_1 + N_2 + \dots + N_k$  nodos tiene como raíz el nodo  $n_r$ , los nodos  $n_1, n_2, \dots, n_k$  son los hijos de  $n_r$  y el conjunto de nodos hoja está formado por la unión de los  $k$  conjuntos hojas iniciales. A cada uno de los árboles  $A_i$  se les denota ahora subárboles de la raíz.

Una sucesión de nodos del árbol, de forma que entre cada dos nodos consecutivos de la sucesión haya una relación de parentesco, decimos que es un recorrido árbol. Existen dos recorridos típicos para listar los nodos de un árbol: primero en profundidad y primero en anchura. En el primer caso, se listan los nodos expandiendo el hijo actual de cada nodo hasta llegar a una hoja, donde se vuelve al nodo anterior probando por el siguiente hijo y así sucesivamente. En el segundo, por su parte, antes de listar los nodos de nivel  $n + 1$  (a distancia  $n + 1$  aristas de la raíz), se deben haber listado todos los de nivel  $n$ . Otros recorridos típicos del árbol son preorden, postorden e entreorden:

- El recorrido en preorden, también llamado orden previo consiste en recorrer en primer lugar la raíz y luego cada uno de los hijos  $A_1, A_2, \dots, A_k$  en orden previo.
- El recorrido en entreorden, también llamado orden simétrico (aunque este nombre sólo cobra significado en los árboles binarios) consiste en recorrer en primer lugar  $A_1$ , luego la raíz y luego cada uno de los hijos  $A_2, \dots, A_k$  en orden simétrico.
- El recorrido en postorden, también llamado orden posterior consiste en recorrer en primer cada uno de los hijos  $A_1, A_2, \dots, A_k$  en orden posterior y por último la raíz.

Finalmente, puede decirse que esta estructura es una representación del concepto de árbol en teoría de grafos. Un árbol es un grafo conexo y acíclico. (3)

A continuación definiremos las estructuras de datos avanzadas que implementamos en nuestra biblioteca y que pertenecen a la clasificación de árboles:

a) **Árboles Generales:**

Es la generalización del concepto de árbol donde un árbol  $A$  es un conjunto de elementos (que denominaremos nodos), finito y no vacío, que cumple:

- Existe un nodo especial  $r$  denominado raíz del árbol

– Los restantes nodos están particionados en  $n$  ( $n=0$ ) subconjuntos disjuntos siendo  $A_1, A_2, \dots, A_k$  cada uno de ellos un árbol. (4)

b) **Árbol Binario:**

Un árbol binario es una estructura de datos de tipo árbol en donde cada uno de los nodos del árbol puede tener 0, 1, ó 2 subárboles llamados de acuerdo a su caso como:

- Si el nodo raíz tiene 0 relaciones se llama hoja.
- Si el nodo raíz tiene 1 relación a la izquierda, el segundo elemento de la relación es el subárbol izquierdo.
- Si el nodo raíz tiene 1 relación a la derecha, el segundo elemento de la relación es el subárbol derecho.

La **figura 1** muestra algunas configuraciones de grafos que sí son árboles binarios, y la **figura 2** muestra algunas configuraciones de grafos que no son árboles binarios.

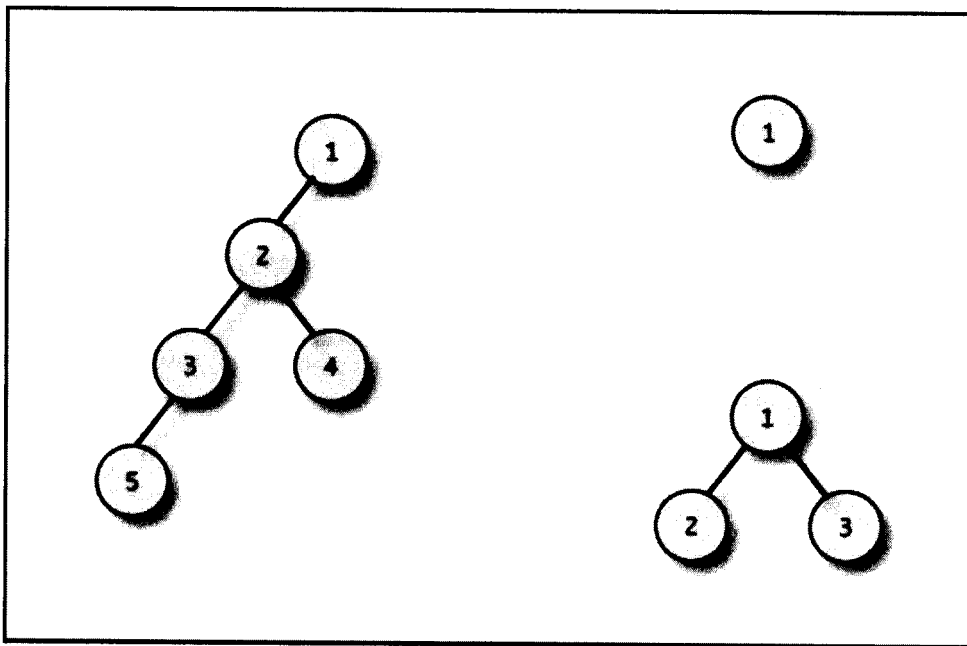


Figura 1: Grafos que son estructuras tipo árbol binario

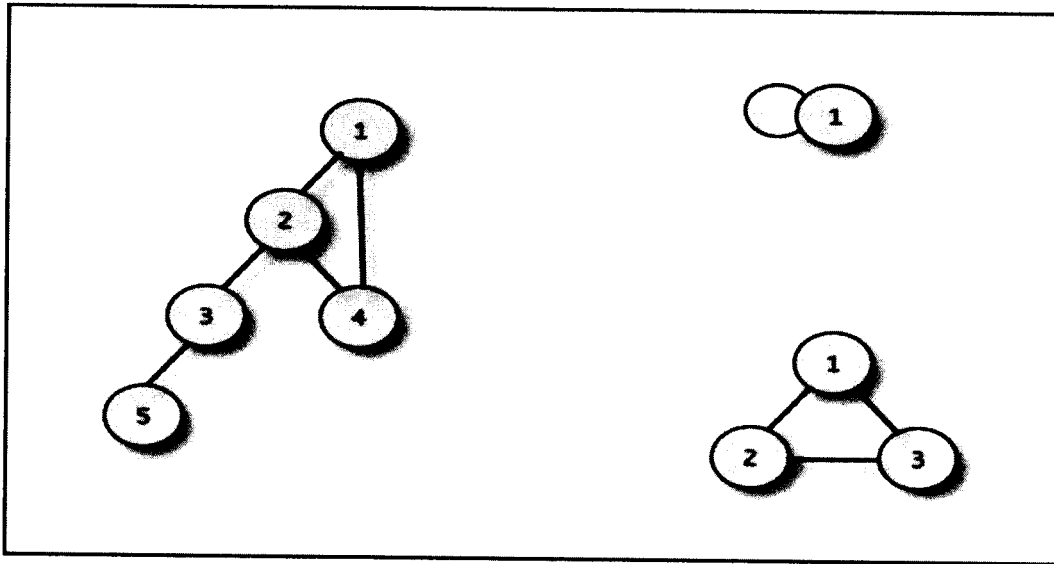


Figura 2: Grafos que no son árboles binarios

En teoría de grafos se usa la siguiente definición: «Un árbol binario es un grafo conexo, acíclico y no dirigido tal que el grado de cada vértice no es mayor a 3». De esta forma sólo existe un camino entre un par de nodos. (5)

**c) Árboles de Búsqueda:**

Un árbol que soporta una búsqueda eficaz y las operaciones de inserción y extracción se llama un árbol de búsqueda. En este contexto, el árbol se utiliza para almacenar un conjunto finito de llaves extraídas de un conjunto totalmente ordenado de llaves  $K$ . Cada nodo del árbol contiene una o más llaves y todas las llaves en el árbol son únicas, es decir, las llaves duplicadas son permitidas.

Lo que hace que un árbol sea un árbol de búsqueda es que las llaves no aparecen en forma arbitraria en los nodos del árbol. En lugar de ello, hay un criterio de ordenamiento de datos que determina dónde una llave puede aparecer en el árbol en relación con las otras llaves en ese árbol. En las secciones siguientes se presentan dos tipos de árboles de búsqueda relacionados, árboles M-Way y árboles binarios de búsqueda. (6)

**– Árboles Binarios de Búsqueda:**

Un árbol binario de búsqueda es un árbol binario que se comporta como un árbol de búsqueda y por definición un árbol binario de búsqueda  $T$  es un conjunto finito de llaves. O bien es el conjunto vacío  $T = \emptyset$ , o el conjunto consta de una raíz  $r$  y exactamente dos

árboles binarios de búsqueda  $T_I$  , y  $T_D$ ,  $T = \{r; T_I; T_R\}$  de tal modo que las siguientes propiedades se cumplen:

Todas las llaves que figura en el sub árbol izquierda  $T_I$ , son menos de  $r$ , es decir,  
 $\forall k \in T_I: k < r$

Todas las llaves que figuran en el derecho sub árbol  $T_D$ , son mayores que  $r$ , es decir,  
 $\forall k \in T_D: k > r$  (7)

– **Árbol AVL:**

Es el árbol binario de búsqueda balanceado por altura. Su nombre se debe a sus inventores Adelson-Velsky<sup>1</sup> y Landis<sup>2</sup>.

Factor de balance:

Es el concepto llave para definir los árboles AVL. El factor de balance para cualquier nodo X se define como la diferencia entre la altura del hijo izquierdo de X y la altura del hijo derecho de X.

Un árbol binario es AVL si:

$|FB(X)| < 2$  donde FB es el factor de balance.

Re balancear un árbol:

Consiste en reubicar los nodos de tal forma que los factores de balance de todos los nodos sean -1, 0, ó 1 y que el recorrido entreorden sea el mismo que antes de reubicarlos.

Operaciones de balanceo:

Antes de ver las operaciones definiremos 2 variables:

P = dirección del registro con factor de balance no permitido. (2 ó -2).  
 Q = dirección del hijo izquierdo o derecho de P dependiendo de si  $FB(P) = 2$  ó  $FB(P) = -2$ , si el factor de balance de P es 2 entonces Q es el hijo izquierdo de P y si el factor de balance de P es -2 entonces Q es el hijo derecho de P.

1. *Rotación a la derecha:*

Se efectúa cuando:  $FB(P) = 2$  y  $FB(Q) = 1$

<sup>1</sup>Georgy Maximovich Adelson-Velsky: matemático y científico de la computación ruso, acreedor del premio Moscú Mathematical Society, por sus inovaciones como estudiante en el campo de las curvaturas en espacios tridimensionales. Con Y.M. Landis, inventó el árbol AVL en 1962.

<sup>2</sup>Yevgeniy Mikhailovich Landis: Matemático ruso que trabajó principalmente en las ecuaciones de las diferenciales parciales. Con G.M. Adelson-Vesky inventó la estructura de datos árbol AVL.

Esta consiste en girar, en el sentido de las manecillas del reloj, el nodo P alrededor del nodo Q. Como consecuencia de este giro, P pasará a ser el nuevo hijo derecho de Q, y el anterior hijo derecho de Q será el nuevo hijo izquierdo de P. Además, Q será la nueva raíz.

2. *Rotación a la izquierda:*

Se efectúa cuando:  $FB(P) = -2$  y el  $FB(Q) = -1$

Consiste en girar, en el sentido contrario de las manecillas del reloj, P alrededor de Q. P será el nuevo hijo izquierdo de Q; Q será la nueva raíz del árbol balanceado, y los factores de balance P y Q quedarán en (0).

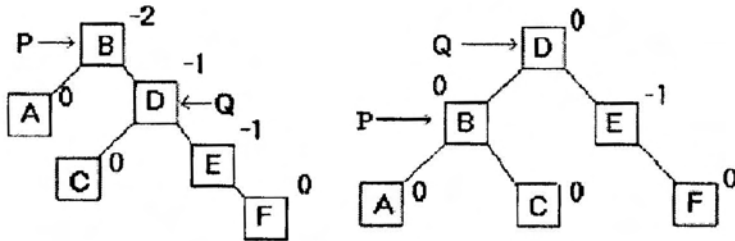


Figura 4: Rotación a la derecha en un AVL

3. *Doble rotación a la derecha:*

Para definir las dobles rotaciones hay una nueva convención:

Sea R el nodo que representa el hijo izquierdo o el hijo derecho de Q dependiendo de si el factor de balance de Q es 1 ó -1. Es decir, si el factor de balance del nodo Q es 1, el nodo R es el hijo izquierdo del nodo Q, y si el factor de balance de Q es -1, R es el hijo derecho del nodo Q.

La doble rotación se efectúa cuando:  $FB(P) = 2$  y el  $FB(Q) = -1$

Consiste en una rotación a la izquierda de Q alrededor de R seguida de una rotación a la derecha de P alrededor de R.

Como consecuencia de estas rotaciones sucede lo siguiente:

R será la nueva raíz del árbol balanceado; P será el nuevo hijo derecho de R; Q será el nuevo hijo izquierdo de R; el anterior hijo derecho de R será el nuevo hijo izquierdo de P; el anterior hijo izquierdo de R será el nuevo hijo derecho de Q; el factor de balance de R será (0), y los factores de balance de P y Q tomarán nuevos valores, los cuales dependerán del factor de balance inicial del nodo R. El factor de balance inicial del nodo puede ser (0), (1) o (-1).

#### 4. Doble rotación a la izquierda:

Se efectúa cuando  $FB(P) = -2$  y el  $FB(Q) = 1$

Consiste en una rotación a la derecha de Q alrededor de R seguida de una rotación a la izquierda de P alrededor de R.

Como consecuencia de estas rotaciones sucede lo siguiente:

R será la nueva raíz del árbol balanceado; P será el nuevo hijo izquierdo de R; Q será el nuevo hijo derecho de R; el anterior hijo derecho de R será el nuevo hijo izquierdo de Q; el anterior hijo izquierdo de R será el nuevo hijo derecho de P; el factor de balance de R será (0), y los factores de balance de P y Q tomarán nuevos valores, los cuales dependerán del factor de balance inicial del registro R. El factor de balance inicial del nodo R puede ser (0), 1 o -1. **(8)**

#### – Árbol B (B-Tree):

Así como son los árboles AVL son árboles binarios de búsqueda balanceados, los árboles-B son árboles multicaminos de búsqueda balanceados. Por la imposición de una condición de equilibrio, la forma de un árbol AVL está limitada de modo que garantice que la búsqueda, y las operaciones de inserción y de extracción sean todas  $O(\log n)$ , donde n es el número de elementos en el árbol. Las formas de B-árboles se ven limitados por las mismas razones y con el mismo efecto.

Un B-Tree de orden M puede ser el vacío, o un árbol multicamino de búsqueda T con las siguientes propiedades:

1. La raíz de T tiene al menos dos subárboles y cuando más M subárboles.
2. Todos los nodos internos de T (con excepción de su raíz) y tienen entre  $\lceil M/2 \rceil$  y M subárboles.
3. Todos los nodos externos de T se encuentran en el mismo nivel.

Un árbol B de orden uno es claramente imposible. Por lo tanto, los árboles B de orden M son en realidad definidos por  $M \geq 2$ . **(7)**



## – **Árbol 2-3:**

El árbol 2-3 es también un árbol de búsqueda como el árbol binario de búsqueda, pero este árbol trata de resolver el problema del árbol no balanceado puesto a que puede convertirse en una lista enlazada. Con un árbol no balanceado como este todas las ventajas de los árboles binarios de búsqueda desaparecen, la búsqueda en el árbol se puede tornar lenta y desperdiciarse mucha memoria debido a los punteros a los hijos izquierdos vacíos.

El 2-3 árbol trata de resolver este problema utilizando una estructura diferente y un ágil procedimiento de agregación y eliminación para ayudar a mantener el árbol más o menos balanceado. El mayor inconveniente con el 2-3 árbol es que se requiere más espacio de almacenamiento que los árboles binarios de búsqueda normales.

El 2-3 árbol se llama así porque el mayor número posible de hijos de cada nodo puede ser 2 o 3. Esto hace que el árbol sea un poco más complejo. (9)

## – **Árbol Rojo-Negro (R-B Tree):**

Un Árbol rojo-negro es un árbol binario en el que cada nodo tiene un color como atributo extra, ya sea rojo o negro. El color de los nodos asegura que la trayectoria más larga de la raíz a una hoja no es más larga que el doble del largo de la más corta. Esto significa que el árbol está fuertemente balanceado. Esto asegura que las operaciones de inserción, eliminación y búsqueda tomen un tiempo  $O(\log n)$ .

Un Árbol rojo-negro debe satisfacer estas propiedades:

- La raíz es negra
- Todas las hojas son negras
- Los nodos rojos sólo pueden tener hijos negros

Todos los caminos de un nodo a sus hojas contienen el mismo número de nodos negros (10)

### **1.4.2 Grafos.**

Un *grafo*  $G$  es una pareja  $G = (V, A)$ , donde  $V$  es un conjunto finito (vértices) y  $A$  es un subconjunto del conjunto de parejas no ordenadas de  $V$  (arcos).

En otras palabras un grafo es un objeto matemático que se utiliza para representar circuitos, redes, etc. Los grafos son muy utilizados en computación, ya que permiten resolver problemas muy complejos.

Un grafo consta de vértices (o nodos) y aristas. Los vértices son objetos que contienen información y las aristas son conexiones entre vértices. Para representarlos, se suelen utilizar puntos para los vértices y líneas para las conexiones, aunque hay que recordar siempre que la definición de un grafo no depende de su representación.

Un camino entre dos vértices es una lista de vértices en la que dos elementos sucesivos están conectados por una arista del grafo. Así, el camino AJLOE es un camino que comienza en el vértice A y pasa por los vértices J, L y O (en ese orden) y al final va del O al E. El grafo será conexo si existe un camino desde cualquier nodo del grafo hasta cualquier otro. Si no es conexo constará de varias componentes conexas.

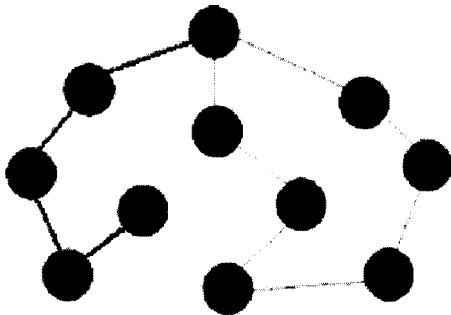


Figura 5: Grafo con camino AJLOE y conexo

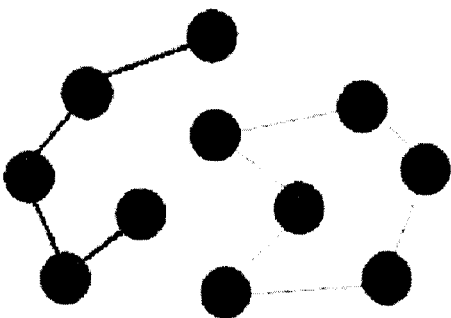


Figura 6: Grafo con camino AJLOE y no conexo

Un camino simple es un camino desde un nodo a otro en el que ningún nodo se repite (no se pasa dos veces). Si el camino simple tiene como primer y último elemento al mismo nodo se denomina ciclo. Cuando el grafo no tiene ciclos tenemos un árbol. Varios árboles independientes forman un bosque. Un árbol de expansión de un grafo es una reducción

del grafo en el que solo entran a formar parte el número mínimo de aristas que forman un árbol y conectan a todos los nodos.

Según el número de aristas que contiene, un grafo es completo si cuenta con todas las aristas posibles (es decir, todos los nodos están conectados con todos), disperso si tiene relativamente pocas aristas y denso si le faltan pocas para ser completo.

Las aristas son la mayor parte de las veces bidireccionales, es decir, si una arista conecta dos nodos A y B se puede recorrer tanto en sentido hacia B como en sentido hacia A: estos son llamados grafos no dirigidos. Sin embargo, en ocasiones tenemos que las uniones son unidireccionales. Estas uniones se suelen dibujar con una flecha y definen un grafo dirigido. Cuando las aristas llevan un coste asociado (un entero al que se denomina peso) el grafo es ponderado. Una red es un grafo dirigido y ponderado. (11)

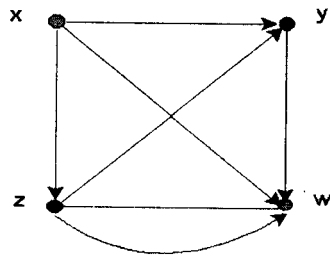
### **Representación de grafos**

Una característica especial en los grafos es que podemos representarlos utilizando dos estructuras de datos distintas. En los algoritmos que se aplican sobre ellos veremos que adoptarán tiempos distintos dependiendo de la forma de representación elegida. En particular, los tiempos de ejecución variarán en función del número de vértices y el de aristas, por lo que la utilización de una representación u otra dependerá en gran medida de si el grafo es denso o disperso. (11)

#### **– Representación por matriz de adyacencia:**

Es la forma más común de representación y la más directa. Consiste en una tabla de tamaño  $V \times V$ , en que la que  $a[i][j]$  tendrá como valor 1 si existe una arista del nodo  $i$  al nodo  $j$ . En caso contrario, el valor será 0. Cuando se trata de grafos ponderados en lugar de 1 el valor que tomará será el peso de la arista. Si el grafo es no dirigido hay que asegurarse de que se marca con un 1 (o con el peso) tanto la entrada  $a[i][j]$  como la entrada  $a[j][i]$ , puesto que se puede recorrer en ambos sentidos. (11)

La matriz de adyacencia siempre ocupa un espacio de  $V \times V$ , es decir, depende solamente del número de nodos y no del de aristas, por lo que será útil para representar grafos densos.



Considere el grafo siguiente "G":

Suponga que los nodos se mantienen en memoria en un arreglo DATOS tal como sigue:

DATOS: X, Y, Z, W

Para hallar la matriz de adyacencia A del grafo "G", tenemos que tomar en cuenta que los nodos están normalmente ordenados de acuerdo con la forma en que aparecen en memoria; o sea, asumimos que  $u_1 = X$ ,  $u_2 = Y$ ,  $u_3 = Z$ , y  $u_4 = W$ , la matriz de adyacencia A de G sería la siguiente:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Aquí  $a_{ij} = 1$  si hay una arista va desde  $u_i$  a  $u_j$ ; si no  $a_{ij} = 0$ . (3)

Así entonces para hallar la matriz de camino P de G mediante las potencias de la matriz de adyacencia A, como G tiene cuatro nodos se calcula:

$$A^2, A^3, A^4 \text{ y } B_4 = A, A^2, A^3, A^4:$$

$$A^2 = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad A^3 = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$A^4 = \begin{pmatrix} 0 & 2 & 2 & 3 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{pmatrix} \quad B_4 = \begin{pmatrix} 0 & 5 & 6 & 8 \\ 0 & 1 & 2 & 3 \\ 0 & 3 & 3 & 5 \\ 0 & 2 & 3 & 5 \end{pmatrix}$$

Por lo tanto la matriz de caminos P se obtiene ahora haciendo  $p_{ij} = 1$  siempre que haya una entrada positiva en la matriz  $B_4$ . Así

$$P = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

La matriz de caminos muestra que no hay camino de u 1 a u 2 de hecho, no hay camino de ningún nodo a u 1 por tanto, G no es fuertemente conexo. (3)

– **Representación por lista de adyacencia:**

Otra forma de representar un grafo es por medio de listas que definen las aristas que conectan los nodos. Lo que se hace es definir una lista enlazada para cada nodo, que contendrá los nodos a los cuales es posible acceder. Es decir, un nodo A tendrá una lista enlazada asociada en la que aparecerá un elemento con una referencia al nodo B si A y B tienen una arista que los une. Obviamente, si el grafo es no dirigido, en la lista enlazada de B aparecerá la correspondiente referencia al nodo A.

Las listas de adyacencia serán estructuras que contendrán un valor entero (el número que identifica al nodo destino), así como otro entero que indica el coste en el caso de que el grafo sea ponderado. (11)

Un grafo "G" se guarda en memoria como sigue:

NODO	A	B	0	E	0	D	C	0
SIG	7	4	0	6	8	0	2	3
ADY	1	2	0	5	0	7	9	0
	1	2	3	4	5	6	7	8

PRINCIPIO = 1, NDISP = 5

DEST	2	6	4	0	6	7	4	0	4	6
ENL	10	3	6	0	0	0	0	4	0	0
	1	2	3	4	5	6	7	8	9	10

ADISP = 8

Para dibujar el respectivo grafo "G", primero debemos buscar todos los vecinos de cada NODO [K] recorriendo su lista de adyacencia que tiene el puntero de adyacencia ADY [J] (3; 3).

Esto da como resultado:

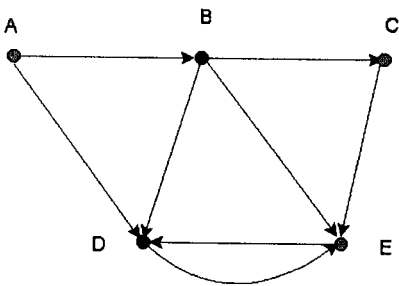
A: 2(B) y 6(D)

B: 6(D), 4(E) y 7(C)

C: 4(E)

D: 4(E)

E: 6(D)



– **Exploración en los grafos:**

A la hora de explorar un grafo, nos encontramos con dos métodos distintos. Ambos conducen al mismo destino (la exploración de todos los vértices o hasta que se encuentra uno determinado), si bien el orden en que éstos son "visitados" decide radicalmente el tiempo de ejecución de un algoritmo, como se verá posteriormente. (11)

En primer lugar, una forma sencilla de recorrer los vértices es mediante una función recursiva, lo que se denomina búsqueda en profundidad. La sustitución de la recursión (cuya base es la estructura de datos Pila) por una Cola nos proporciona el segundo método de búsqueda o recorrido, la búsqueda en amplitud o anchura.

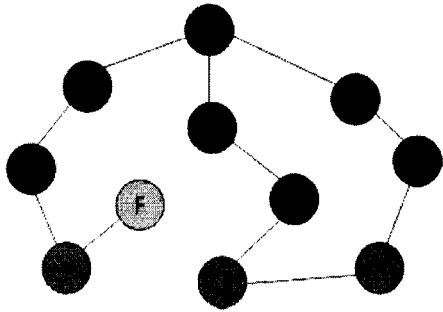


Figura 7: Grafo a)

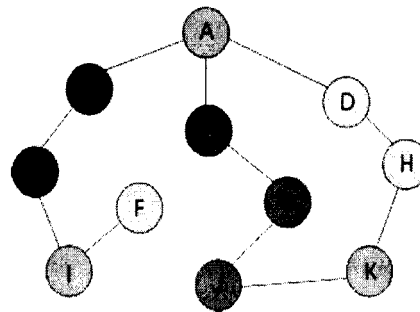


Figura 8: Grafo b)

Suponiendo que el orden en que están almacenados los nodos en la estructura de datos correspondiente es A-B-C-D-E-F... (el orden alfabético), tenemos que el orden que seguiría el recorrido en profundidad sería el siguiente:

A-B-E-I-F-C-G-J-K-H-D

En un recorrido en anchura el orden sería, por contra:

A-B-C-D-E-G-H-I-J-K-F

Es decir, en el primer caso se exploran primero los verdes y luego los marrones, pasando primero por los de mayor intensidad de color. En el segundo caso se exploran primero los verdes, después los rojos, los naranjas y, por último, el rosa.

Es destacable que el nodo D es el último en explorarse en la búsqueda en profundidad pese a ser adyacente al nodo de origen (el A). Esto es debido a que primero se explora la rama del nodo C, que también conduce al nodo D. (11)

En estos ejemplos hay que tener en cuenta que es fundamental el orden en que los nodos están almacenados en las estructuras de datos. Si, por ejemplo, el nodo D estuviera antes que el C, en la búsqueda en profundidad se tomaría primero la rama del D (con lo que el último en visitarse sería el C), y en la búsqueda en anchura se exploraría antes el H que el G.

### 1.4.3 Heaps.

Un heap es una estructura de Árbol con información perteneciente a un conjunto ordenado. Los heaps tienen la característica de que cada nodo tiene un valor mayor que el de todos sus nodos hijos. (12)

Sean A y B dos nodos de un heap tal que B es un hijo de A. El heap debe entonces satisfacer la siguiente condición (Propiedad de heap): llave (A)  $\geq$  llave (B)

Ésta es la única restricción en los heaps. Ella implica que el mayor elemento (o el menor, dependiendo de la relación de orden escogida) está siempre en el nodo raíz. Debido a esto, los heaps se utilizan para implementar colas de prioridad. La eficiencia de las operaciones en los heaps es crucial en diversos algoritmos de recorrido de grafos y de ordenamiento.

Las operaciones normalmente utilizadas en un heap son la inserción de un elemento cualquiera y la eliminación del máximo (el elemento de la raíz).

– **Heap Binario:**

Los heaps binarios (*binary heaps* en inglés) son un caso particular y sencillo de la estructura de datos *Heap* que está basada en un árbol binario balanceado, que puede verse como un árbol binario con dos restricciones adicionales: (7)

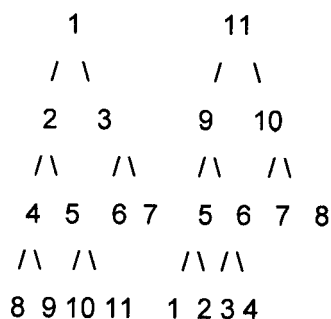
*Propiedad de heap*

Cada nodo contiene un valor superior a los de sus hijos (para un heap por máximos) o más pequeño que los de sus hijos (para un heap por mínimos).

*Árbol completo*

El árbol está balanceado y en un mismo nivel las inserciones se realizan de izquierda a derecha.

Los heaps por máximos se utilizan frecuentemente para representar colas de prioridad. A continuación se muestran dos heaps uno por mínimos y otro por máximos que representan el mismo conjunto de valores.





El orden de los nodos hermanos en un heap no está especificado en la propiedad de heap, de manera que los subárboles de un nodo son intercambiables.

– **Heap Binomial:**

Un heap binomial es una colección de árboles binomiales que están heap-ordenados. A continuación para su mejor comprensión explicamos que es un árbol binomial.

*Árbol Binomial:*

El árbol binomial  $B_k$  es un árbol ordenado definido de forma recursiva. Un árbol binomial  $B_k$  consiste en dos árboles binomiales  $B_{k-1}$  que están enlazados y la raíz de uno es el hijo más a la izquierda de la raíz del otro. (13)

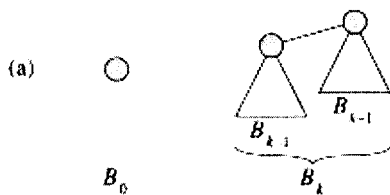


Figura 9 a): Árbol Binomial

La figura 9 a) muestra el árbol binomial  $B_0$  que consiste en un solo nodo y la definición recursiva.

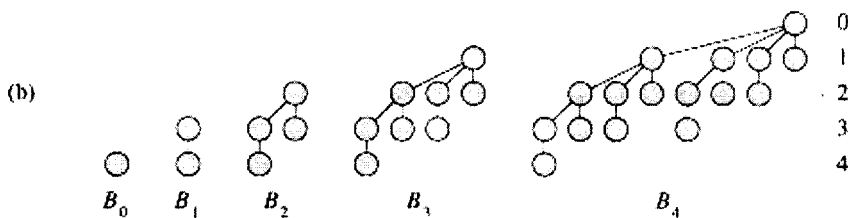


Figura 9 b): árboles binomiales desde  $B_0$  hasta  $B_4$ .

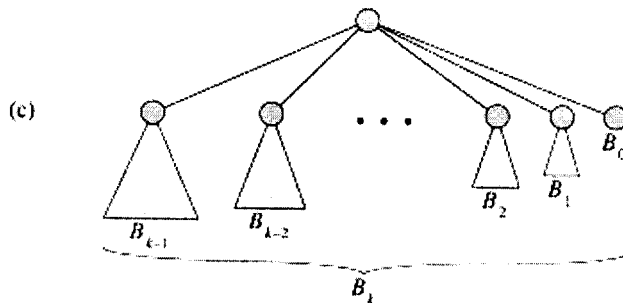


Figura 9 c) Árboles Binomiales

La figura c muestra otra forma de ver los árboles binomiales, donde un árbol binomial  $B_k$  tiene una raíz una colección de hijos que son los árboles binomiales  $B_{k-1}$  hasta  $B_0$ .

Podemos concluir que para todo árbol binomial  $B_k$  se cumple que:

- Hay  $2^k$  nodos
- La altura del árbol es  $k$ .
- Hay exactamente  $\binom{k}{i}$  nodos a profundidad  $i$  para  $i = 0, 1, \dots, k$
- La raíz tiene grado  $k$  y es mayor que cualquier otro grado del árbol.

Una vez que ya se definió el árbol binomial, retomamos la idea de que un heap binomial  $H$  es un conjunto de árboles binomiales y satisface las siguientes propiedades:

1. Cada árbol binomial en  $H$  es un heap ordenado: la llave de un nodo es mayor o igual que la llave de su padre.
2. Existe al menos un árbol binomial en  $H$  cuya raíz tiene un determinado grado.

La primera propiedad nos dice que la raíz de un árbol heap-ordenado contiene la más pequeña llave en el árbol. La segunda propiedad implica que un  $n$ -nodo de un heap binomial  $H$  consiste en al menos  $\lceil \lg n \rceil + 1$  árboles binomiales.

#### – Heap de Fibonacci:

Como un heap binomial, un heap de Fibonacci es una colección de árboles-heaps ordenados. Los árboles en un heap de Fibonacci no están limitados a ser árboles binomiales, sin embargo la figura (a) a continuación muestra un ejemplo de un heap de Fibonacci. (13)

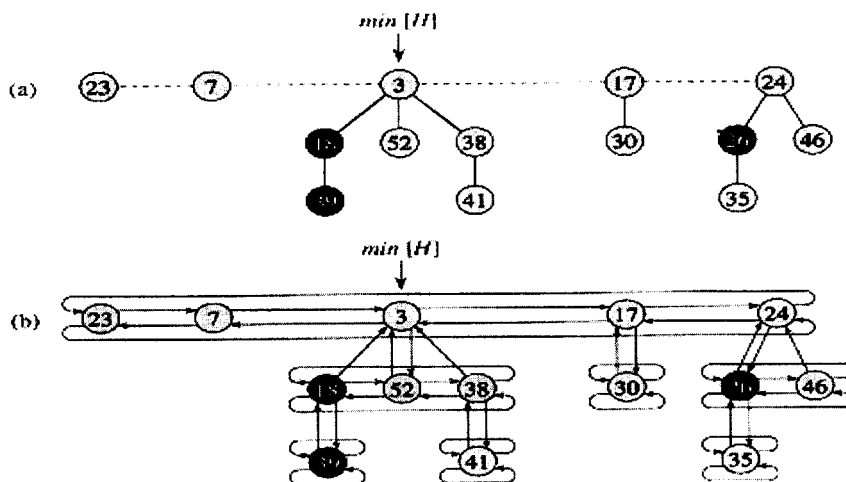


Figura 10: Heap de Fibonacci

A diferencia de los árboles dentro de los heaps binomiales, los cuales están ordenados, los árboles dentro de los heaps de Fibonacci están enraizados, pero desordenados. Como la figura (b) muestra, cada nodo contiene un puntero  $x.p[x]$  a su padre y un hijo puntero  $x.hijo[x]$  a uno de sus hijos. Los hijos de  $x$  están unidos entre sí en una lista circular doblemente enlazada, que llamamos lista hija de  $x$ . Cada hijo y cada lista hija tiene punteros de la izquierda  $[y]$  y de derecha  $[y]$  que apuntan a las  $y$ 's izquierda y derecha hermanas, respectivamente. Si es un nodo y es único hijo, entonces  $izquierda[y] = derecha[y] = y$ . El orden en que aparecen los hermanos en una lista hija es arbitrario.

Las listas circulares doblemente enlazadas tienen dos ventajas para su utilización en los heaps de Fibonacci. En primer lugar, podemos eliminar un nodo de una lista circular doblemente enlazada en un tiempo de orden  $O(1)$ . En segundo lugar, teniendo dos listas así, podemos concatenarlas en una lista circular doblemente enlazada en un tiempo de orden  $O(1)$ .

Otros dos campos en cada nodo serán útiles. El número de hijos en la lista hija de nodo  $x$  se almacena en  $grado[x]$ . El campo booleano evaluado  $marca[x]$  indica cuando el nodo  $x$  ha perdido un hijo desde la última vez  $x$  se creó el hijo de otro nodo.

– **Pairing Heaps (Heaps Autoajustables):**

Los pairing heaps vienen en dos formas el min pairing heap y el max pairing heap. Representan colas con prioridad máximas o mínimas en dependencia del pairing heap utilizado. (14)

A continuación explicaremos el *max pairing heap*, aclarando que el *min pairing heap* es análogo.

Un *max pairing heap* es un árbol heap-ordenado máximo, o sea un árbol donde la información de la raíz es mayor que cualquier otra existente dentro del árbol. La figura 1 muestra cuatro *max pairing heaps*, nótese que no necesariamente los árboles tienen que ser binarios.

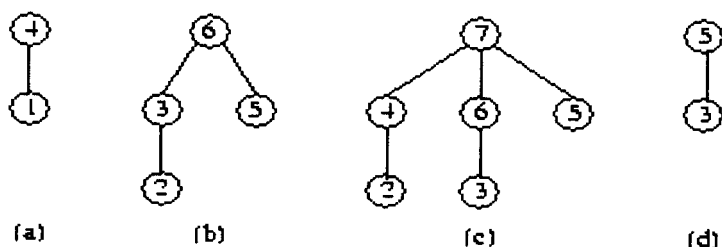


Figura 11: Cuatro *max pairing Heaps* representados ordenadamente.

Lo novedoso de los *pairing heaps* es la posibilidad de mezclar dos heaps en tiempo amortizado de  $O(\log n)$ , con la introducción del procedimiento *compare-link*, el cual compara las raíces de ambos heaps y pone como hijo de la mayor de las raíces al árbol de menor raíz, aclaramos que sería lo contrario en el caso de un *min pairing heap*.

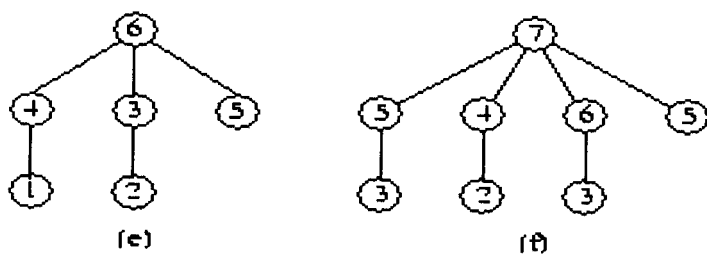


Figura 12: *Max pairing heap*

La figura 12 muestra el *max pairing heap e* que es el resultado de la mezcla de los *max pairing heaps a* y *b* de la figura 1 y el *f* que es la mezcla del *c* y el *d*.

#### 1.4.4 Matrices poco densas.

Una matriz poco densa es aquella que está formada por elementos que en su mayoría son ceros. Este tipo de matrices son matrices cuadradas que se dividen en los siguientes tipos:

1. Matriz triangular superior
2. Matriz triangular inferior
3. Matriz tridiagonal

– **Matriz triangular superior:**

En este tipo de matriz los elementos iguales a cero se encuentran debajo de la diagonal principal. Ejemplo:

$$M = \begin{bmatrix} 5 & 8 & 2 & 7 \\ 4 & -3 & -6 & 0 \\ 9 & -5 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Para evitar el desperdicio de memoria que se ocasionaría al almacenar una matriz en donde la mayoría de los elementos son ceros, es conveniente traspasar a un arreglo unidimensional todos los elementos diferentes de cero.

El arreglo con los elementos distintos de cero de la matriz anterior es el siguiente:

5	8	2	7	4	-3	-6	9	-5	1
---	---	---	---	---	----	----	---	----	---

Una vez que hayamos vaciado la matriz, es indispensable conocer el lugar dentro del arreglo unidimensional en el cual quedaron situados los elementos, y esto se logra con la siguiente formula:

LOC (A [i, j]) =base (A) + (n\*(i-1)) - ((i-2)\*(i-1))/2 + (j-1) donde:

A=Matriz triangular superior

n=No. total de elementos

j= renglones

i=columnas

– **Matriz triangular inferior:**

En este tipo de matrices los elementos iguales a cero se encuentran por encima de la diagonal principal. Ejemplo:

$$M = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 8 & 2 & 0 & 0 \\ 6 & 1 & -4 & 0 \\ 3 & 7 & 6 & 9 \end{bmatrix}$$

Una vez que vaciamos la matriz en un arreglo unidimensional, la formula para obtener las posiciones de los elementos es la siguiente:

5	8	2	6	1	-4	3	7	6	9
---	---	---	---	---	----	---	---	---	---

$$LOC(A[i, j]) = \text{base}(A) + ((i-1)*i)/2 + (j-1)$$

– **Matriz tridiagonal:**

En ésta, los elementos diferentes de cero se encuentran en la diagonal principal ó en las diagonales por debajo ó encima de ésta. Ejemplo:

$$M = \begin{bmatrix} 4 & 5 & 0 & 0 \\ 8 & 3 & 2 & 0 \\ 0 & 7 & 6 & 1 \\ 0 & 0 & 9 & -5 \end{bmatrix}$$

Y el arreglo con los elementos diferentes de cero correspondiente a esta matriz es el siguiente:

4	5	8	3	2	7	6	1	9	5
---	---	---	---	---	---	---	---	---	---

La localización de los elementos distintos de cero en el arreglo unidimensional se realiza aplicando la siguiente formula:

$$LOC(A[i, j]) = \text{base}(A) + 2*i + (j-3)$$

### 1.5 Análisis de otras soluciones existentes.

A nivel mundial existen otras soluciones para el trabajo con las estructuras de datos avanzadas, pero no son todo lo abarcadoras, ni todo lo amigable para el usuario que las necesita.

Ejemplo de estas es la *Standard Template Library* (STL) de la Borland, la cual incluye colecciones de estructuras como las listas, las pilas y las colas, pero solo se limita a estas, siendo esto su principal desventaja.

Otro ejemplo es la librería *Java Development Kit* (JDK) que actualmente está en su versión 1.2, la cual incluye el paquete *Java Collections*, que trae las estructuras siguientes:

- *TreeSet*: implementada mediante un árbol binario ordenado
- *ArrayList*: implementada mediante un arreglo
- *LinkedList*: implementada mediante una lista enlazada.
- *HashMap*: implementada mediante una tabla hash
- *TreeMap*: implementada mediante un árbol binario.

Estas estructuras son mas abarcadoras pero igual están limitadas, al no cubrir todas las estructuras de datos y no brindan el código fuente para su análisis y posterior modificación.

También existe la *Microsoft Base Class Library* (BCL) para C# que es una librería en el .Net Framework que incluye la *System.Collections* que permite el trabajo con estructuras de datos simples como listas, pilas, colas y diccionarios, pero a pesar de no complicar demasiado su uso no se aventura a incluir las variaciones complejas de estas.

Además es válido decir que Python soporta el trabajo con las colecciones listas, tuplas y diccionarios, pero no para grandes volúmenes de datos.

En la UCI las soluciones propuestas para el trabajo con las estructuras de datos son principalmente con fines docentes, donde se enseñan las básicas como listas, pilas, colas, árboles y grafos, pero lo verdaderamente complejo y óptimo no se imparte. Mientras que algunos proyectos productivos se encargan de implementar algunas de las estructuras de datos avanzadas, pero siempre para resolver problemas puntuales en sus trabajos y de forma aislada.

### **1.6 Conclusiones parciales.**

En este capítulo se ha mostrado la fundamentación teórica del trabajo de diploma, donde se fue bastante amplio en conceptos básicos y necesarios para el dominio del tema, además de brindar la descripción del objeto de estudio desglosado en las estructuras de datos que se proponen como posible solución al problema.

También se reflejan los resultados de los estudios e investigaciones realizados sobre otras soluciones existentes, tanto en el marco de la UCI como a nivel mundial, lo que ratifica la no existencia de una biblioteca que agrupe el volumen de estructuras de datos avanzadas que son propuestas.



## CAPÍTULO II: Tendencias y tecnologías actuales a desarrollar

### 2.1 Introducción.

En este capítulo se abordara todo lo referente a las tecnologías, las utilizadas y las que existen como alternativas, haciendo una comparación entre ellas y fundamentando el por qué de la elección. Dentro de ellas se hablará de metodologías de desarrollo, lenguaje de programación, herramientas empleadas, etc.

### 2.2 Las Tecnologías de la Información y las Comunicaciones (TIC).

Las TIC son aquellas tecnologías que permiten transmitir, procesar y difundir información de manera rápida y eficaz. Son consideradas la base para reducir la Brecha Digital sobre la que se tiene que construir una Sociedad de la Información y una Economía del Conocimiento.

Las TIC optimizan el manejo de la información y el desarrollo de la comunicación. Permiten actuar sobre la información y generar mayor conocimiento e inteligencia. Abarcan todos los ámbitos de la experiencia humana. Están en todas partes y modifican los ámbitos de la experiencia cotidiana: el trabajo, las formas de estudiar, las modalidades para comprar y vender, los trámites, el aprendizaje y el acceso a la salud, entre otros. (15)

Una de las habilidades humanas que se beneficia con las TICs es el aprendizaje, es decir el uso de las tecnologías multimedia y la internet para mejorar la calidad de este, darle acceso a las personas que no están cerca de las instituciones educativas y poner a disposición de todos innovadoras formas de educación en cualquier ambiente en que se desarrolle el individuo.

### 2.3 Desarrollo de bibliotecas de clases.

Los lenguajes que soportan la programación orientada a objetos definen todas sus funcionalidades en términos de objetos. Todos los objetos del mismo tipo constituyen una clase de objetos en la cual existen funcionalidades comunes, la cual es codificada dentro de un programa como una clase.

Existen un número de clases que son comunes para muchos programas que son agrupadas en bibliotecas. Un ejemplo podría ser la clase que maneja el control de una ventana en el monitor. Esta clase necesitaría saber como mostrar la ventana, que características comunes mostrar en la ventana (barra de título, minimizar y maximizar botones, etc.) y cómo llevar a cabo funciones tales como mover o restaurar la ventana.

Una biblioteca de clases es un conjunto de objetos reutilizables que pueden ser controlados dentro de una aplicación o programa para lograr objetivos importantes para una organización o individuo y evitar la pérdida de tiempo y esfuerzo que se dedique en programarla. **(16)**

La solución para esto en la vida de las compañías capitalistas y en general, la sociedad capitalista es que cuando se necesite utilizar una de estas bibliotecas se compra. Sin embargo, la solución que se presenta es la de brindarle a la sociedad informática una biblioteca de clases distribuida gratuitamente para el uso personal o de una compañía sin tener que pagar por ello.

### **2.4 El Lenguaje Unificado de Modelado (UML) como soporte a la Programación Orientada a Objetos.**

La ingeniería es la profesión que aplica conocimientos y experiencias para que mediante diseños, modelos y técnicas se resuelvan problemas que afectan a los seres vivos con creatividad e ingenio.

Por otro lado el modelado es una técnica cognitiva que consiste en crear una representación ideal de un objeto real mediante un conjunto de simplificaciones y abstracciones, cuya validez se pretende constatar. La validación del modelo se lleva a cabo comparando las implicaciones predichas por el mismo con observaciones.

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

Es importante resaltar que su propósito es especificar y no describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software, pero no especifica en sí mismo qué metodología o proceso usar.

Tiene una gran cantidad de propiedades que han sido las que, realmente, han contribuido a hacer de UML el estándar de la industria en la actualidad.

Algunas de las propiedades de UML como lenguaje de modelado son:

1. Es un lenguaje distribuido y adecuado a las necesidades de conectividades actuales y futuras. Ampliamente utilizado por la industria del software.
2. Reemplaza a decenas de notaciones empleadas por otros lenguajes.
3. Modela estructuras complejas.
4. Las estructuras más importantes que soporta tienen su fundamento en la tecnología orientada a objeto, tales como objetos, clases, componentes y nodos.
5. Comportamiento del sistema: casos de usos, diagramas de secuencia, de colaboración y otros como los diagramas de clases y de actividades.

Existen otros lenguajes de modelado pero UML es el que más defiende la integridad de modelos estándar. Tal es el caso de BPMML, del cual se describen algunas características a continuación:

BPML [BPML, 2001] es un metalenguaje para el modelado de procesos de negocio que proporciona un modelo de ejecución abstracto para procesos colaborativos y transaccionales basado en el concepto de máquina de estados finita transaccional.

BPML considera un proceso como un conjunto compuesto por una interfaz pública y varias implementaciones, lo que posibilita publicar el proceso en XML independientemente de su implementación privada. De igual modo en que los documentos XML están expresados en un esquema XML, los procesos BPML pueden describirse basándose en el esquema XML de BPML.

Representa los procesos como una mezcla de flujo de control, flujo de datos y flujo de eventos, añadiendo un diseño ortogonal de reglas de negocio, roles de seguridad y

contextos transaccionales. Ofrece también soporte explícito para transacciones distribuidas síncronas y asíncronas.

Se utilizará entonces como notación el Lenguaje Unificado de Modelado para lograr un mayor entendimiento ya que se logra modelar y describir secuencialmente por pasos todos los procesos que se llevan a cabo según la problemática planteada.

### **2.4.1 Visual Paradigm como herramienta CASE para el modelamiento con UML.**

A continuación se muestran características de las herramientas CASE Rational Rose Enterprise y Visual Paradigm Enterprise Edition 3.0 con el fin de que el lector comprenda nuestra decisión de utilizar este último mencionado:

#### *Rational Rose Enterprise*

Rational Rose Enterprise es una buena elección para el ambiente de modelado que soporte la generación de código a partir de modelos en Ada, ANSI C++, C++, CORBA, Java™/J2EE™, Visual C++® y Visual Basic®. Como todos los demás productos Rational Rose, proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad más rápidamente.

Características adicionales incluidas:

- Soporte para análisis de patrones ANSI C++, Rose J y Visual C++ basado en "Design Patterns: Elements of Reusable Object-Oriented Software"
- Característica de control por separado de componentes modelo que permite una administración más granular y el uso de modelos
- Soporte de ingeniería Forward y/o reversa para algunos de los conceptos más comunes de Java 1.5
- La generación de código Ada, ANSI C ++, C++, CORBA, Java y Visual Basic, con capacidad de sincronización modelo- código configurables
- Soporte Enterprise Java Beans™ 2.0
- Capacidad de análisis de calidad de código
- El Add-In para modelado Web provee visualización, modelado y las herramientas para desarrollar aplicaciones de Web
- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos

- Capacidad de crear definiciones de tipo de documento XML (DTD) para el uso en la aplicación
- Integración con otras herramientas de desarrollo de Rational Capacidad para integrarse con cualquier sistema de control de versiones SCC-compliant, incluyendo a Rational ClearCase
- Publicación web y generación de informes para optimizar la comunicación dentro del equipo

Tabla 1. Principales Características de Rational Rose Enterprise

Característica	
Incluye	Rose Enterprise
<b>Integraciones IDE</b>	
Borland JBuilder versiones 7.0 a 10.0	X
Sun Forte for Java Community y Enterprise Editions 3.0	X
Microsoft Visual Studio 6	VB6
Microsoft Visual Studio 2003	MSVC++
Microsoft Visual Studio 2005	MSVC++
Wind River Tornado	
Green Hills MULTI	
<b>Diagramas</b>	
Clases, Componentes, Deployment, Secuencia, Statechart, Caso de Uso	X
Colaboración	X
Physical Storage / Deployment	X
Physical Data / Tables	X
<b>Dominios - Windows</b>	
UML 1.x	X

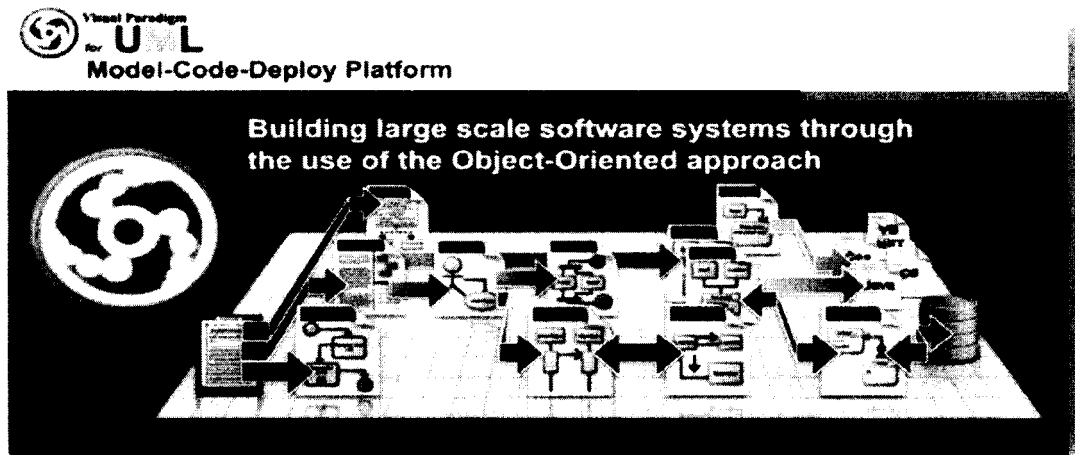
Data Modeling	X
Visual C++ 6.0	X
ANSI C++	X
J2SE, J2EE	Java SE 1.4 con algunas características 1.5 características EJB 1.2
J2ME	
JSP	X
Ada, CORBA IDL y MIDL, XML DTD	X
<b>Desarrollo Orientado al Modelado</b>	
Patterns	X
<b>Build y Deployment</b>	
J2EE deployment	X
Team Support	
Merge Changes	X
Repository / CM	X
<b>Documentación / Reportes</b>	
SoDA Integration	X
Print Diagrams	X
Web Publishing and Report Generation	X
<b>Soportado por Procesos</b>	
Soporte RUP for Systems Engineering	X

Sistemas Operativos y Plataformas de hardware apropiadas para su uso.

- Windows 2000
- Windows NT
- Windows XP

La principal desventaja del Rational Rose es que este no es multiplataforma, es decir, solo defiende los intereses de Microsoft y se utiliza solo en las plataformas que este último desarrolla y no otras como Macintosh y Unix. Además de esta desventaja está la de ser software propietario y todas las desventajas que esto trae consigo.

### Visual Paradigm Enterprise Edition 3.0



Es una herramienta CASE (Computer-Aided Software Engineering) que soporta del UML como lenguaje de modelado. Además provee Modelado del Proceso de Negocio, un generador de mapeo objeto-relación para Java, NET y PHP. Sirve de soporte también para 13 tipos de diagramas para UML.

Una característica clave es su habilidad de generar no solo código del modelo de clases, sino también de la estructura de la base de datos relacional adecuados para sostener persistentemente la información contenida en las clases llamadas "entidad". (17)

Esta herramienta tiene unas características gráficas muy cómodas que facilitan la realización de los diagramas de modelado que sigue el estándar de UML que son:

- Diagramas de clase
- Casos de Uso
- Comunicación
- Secuencia
- Estado

- Actividad
- Componentes

Entre otras características importantes que se tienen tenemos:

- Integración con diversas IDE's como son:
  - Desarrollo drag and drop de aplicaciones web
  - NetBeans(de Sun)
  - JDeveloper(de Oracle)
  - Eclipse (de IBM)
  - JBuilder (de Borland)
  - IntelliJ IDEA
- Ingeniería Inversa para:
  - JAVA
  - .NET
  - XML
  - Hibernate
- Exportación de imágenes jpg, png y svg(w3g estándar)
- Generación de código
- Importación desde Rational Rose
- Exportación e importación XMI
- Generador de informes
- Editor de figuras
- Integración con MS Visio
- Modelado colaborativo con CVS y Subversion

**(18)**

En general ¿cuáles son las características principales de Visual Paradigm?

1. Entorno de creación de diagramas para UML 2.0 y UML 2.1.
2. Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
3. Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
4. Capacidades de ingeniería directa (versión profesional) e inversa. .



5. Disponibilidad en múltiples plataformas incluyendo las relacionadas con el software libre

### **2.5 El Proceso Unificado de Desarrollo de Software (RUP) como base en el desarrollo de la solución.**

En los últimos tiempos la cantidad y variedad de los procesos de desarrollo ha aumentado de forma impresionante, sobre todo teniendo en cuenta el tiempo que estuvo en vigor como ley única el famoso *desarrollo en cascada*. Se podría decir que en estos últimos años se han ido desarrollando dos corrientes en lo referente a los procesos de desarrollo, los llamados *métodos pesados* y los *métodos ligeros*. La diferencia fundamental entre ambos es que mientras los métodos pesados intentan conseguir el objetivo común por medio de orden y documentación, los métodos ligeros (también llamados métodos ágiles) tratan de mejorar la calidad del software por medio de una comunicación directa e inmediata entre las personas que intervienen en el proceso.

A continuación se demostrará por qué razón es escogida RUP como metodología de desarrollo del trabajo de diploma, comparándolo con otras metodologías como XP (Programación Extrema) y FDD (Desarrollo Guiado por Funcionalidad), las cuales se caracterizarán a continuación:

#### Metodología XP (Programación Extrema)

Mientras que RUP intenta reducir la complejidad del software por medio de estructura y la preparación de las tareas pendientes en función de los objetivos de la fase y actividad actual, XP, como toda metodología ágil, lo intenta por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción.

XP intenta minimizar el riesgo de fallo del proceso por medio de la disposición permanente de un representante *competente* del cliente a disposición del equipo de desarrollo. Este representante debería estar en condiciones de contestar rápida y correctamente a cualquier pregunta del equipo de desarrollo de forma que no se retrase al tomar decisiones, de ahí lo de *competente*

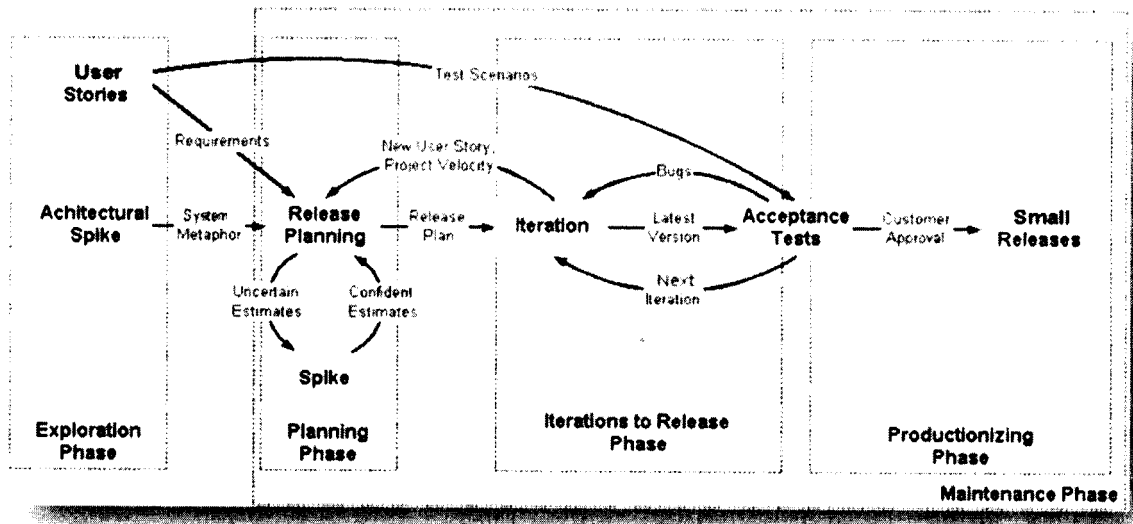


Figura 13: Fases de XP

XP define *UserStories* como la base del software a desarrollar. Estas historias las escribe el cliente y describen escenarios sobre el funcionamiento del software, que no solo se limitan a la GUI si no también pueden describir el modelo, dominio, etc. A partir de las *UserStories* y de la arquitectura perseguida se crea un plan de *releases* o entregas de software entre el equipo de desarrollo y el cliente.

Para cada *release* se discutirán los objetivos de la misma con el representante del cliente y se definirán las iteraciones (de pocas semanas de duración) necesarias para cumplir con los objetivos del *release*. El resultado de cada iteración es un programa que se transmite al cliente para que lo juzgue. En base a su opinión se definen las siguientes iteraciones del proyecto y si el cliente no está contento se adaptará el plan de *releases* e iteraciones hasta que el cliente de su aprobación y el software esté a su gusto.

Junto a los *UserStories* están los escenarios de pruebas que describen el escenario contra el que se comprueba la realización de las *UserStories*. *UserStories* y casos de pruebas son la base sobre la que se asienta el trabajo del desarrollador.

Como primer paso de cada iteración se escribirán las pruebas, de tal forma que puedan ser ejecutadas automáticamente, de manera que pueda comprobarse la corrección del software antes de cada *release*. Esto es de vital importancia en XP debido a su apuesta por las iteraciones cortas que generan software que el cliente puede ver y por la refactorización para mejorar el código constantemente, que hacen más deseable una

cantidad considerable de pruebas lo más automatizables posibles. Así pues, la funcionalidad concreta del software solo se escribe cuando las pruebas para su corrección estén preparadas.

La codificación del software en XP se produce siempre en parejas (dos programadores y un ordenador) por lo que se espera que la calidad del mismo suba en el mismo momento de escribirlo. Al contrario que muchos otros métodos, el código pertenece al equipo en completo, no a un programador o pareja, de forma que cada programador puede cambiar en cualquier parte del código en cualquier momento si así lo necesita, dejándose en todo caso las mejoras orientadas al rendimiento para el final. Las parejas no se mantienen para todo el proyecto si no que rotan cíclicamente a lo largo del mismo, tanto en cuanto a los componentes de la misma como en las partes del software que desarrollan, así cada componente del equipo aprende como trabaja el resto. El objetivo ideal sería que cada componente del equipo trabaje al menos una vez con cada uno de los demás integrantes y con cada componente software, de forma que el conocimiento de la aplicación completa lo posea el equipo entero y no unos pocos miembros.

En XP se programara solo la funcionalidad que es requerida para la *release* actual. Es decir, una gran flexibilidad y capacidad de configuración solo será implementada cuando sea necesaria para cumplir los requerimientos del *release*. Se sigue un diseño evolutivo con la siguiente premisa: conseguir la funcionalidad deseada de la forma más sencilla posible. (19)

### FDD (Desarrollo Guiado por Funcionalidades)

FDD es un proceso diseñado por Peter Coad<sup>3</sup> y Jeff De Luca<sup>4</sup>. Esta metodología se podría considerar a medio camino entre RUP y XP, aunque al seguir siendo un proceso ligero es más similar a este último.

FDD está pensado para proyectos con tiempo de desarrollo relativamente cortos (menos de un año). Se basa en un proceso iterativo con iteraciones cortas (de aproximadamente

---

<sup>3</sup> Peter Coad: Empresario de la Industria del Software y autor de diferentes libros sobre programación. Se dio a conocer por el papel que desempeñó al definir lo que se conoce como los colores de UML, que es una notación de códigos de colores jerárquica muy útil para simplificar la comprensión de un modelo de diseño.

<sup>4</sup> Jeff De Luca: Ejecutivo y estratega de la obtención de resultados en la Tecnología de la Información. Posee más de 20 años de experiencia en Australia y los EE.UU. en el alto nivel de gestión de proyectos, la estrategia, arquitectura de diseño y solución de problemas, con éxito en IBM y en su propia empresa TI.

dos semanas) que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitorizar.

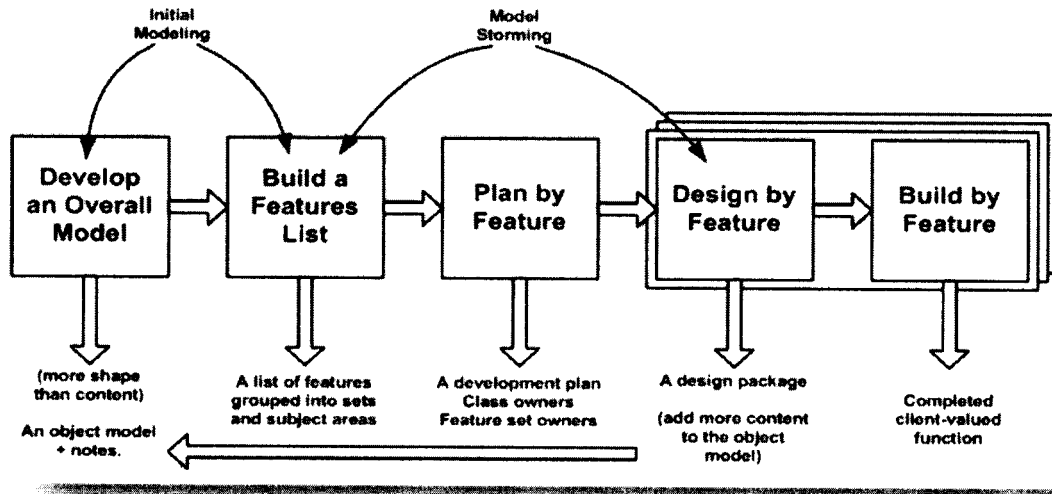


Figura 14: Fases de FDD.

Las iteraciones se deciden en base a features (de ahí el nombre del proceso) o funcionalidades, que son pequeñas partes del software con significado para el cliente.

Un proyecto que sigue FDD se divide en 5 fases:

- 1- Desarrollo de un modelo general
- 2- Construcción de la lista de funcionalidades
- 3- Plan de *releases* en base a las funcionalidades a implementar
- 4- Diseñar en base a las funcionalidades
- 5- Implementar en base a las funcionalidades

Las primeras tres fases ocupan gran parte del tiempo en las primeras iteraciones, siendo las dos últimas la que absorben la mayor parte del tiempo según va avanzando el proyecto, limitándose las primeras a un proceso de refinamiento.

El trabajo (tanto de modelado como de desarrollo) se realiza en grupo, aunque siempre habrá un responsable último (arquitecto jefe o jefe de programadores en función de la fase en que nos encontremos), con mayor experiencia, que tendrá la última palabra en caso de no llegar a un acuerdo. Al hacerlo en grupo se consigue que todos formen parte del proyecto y que los menos inexpertos aprendan de las discusiones de los más

experimentados, y al tener un responsable último, se asignan las responsabilidades que todas las empresas exigen.

Las funcionalidades a implementar en *release* se dividen entre los distintos subgrupos del equipo, y se procede a implementarlas. Las clases escritas tienen propietario(es decir, solo quien las crea puede cambiarlas), es por ello que en el equipo que implementa una funcionalidad dada deberán estar todos los dueños de las clases implicadas, pudiendo encontrarse un programador en varios grupos, implementando distintas funcionalidades.

Habrá también un programador jefe (normalmente el mas experimentado) que hará las funciones de líder del grupo que implementa esa funcionalidad.

En el proceso de implementar la funcionalidad también se contemplan como partes del mismo (en otros métodos se describen como actividades independientes) la preparación y ejecución de pruebas, así como revisiones del código (para distribuir el conocimiento y aumentar la calidad) e integración de las partes que componen el software.

FDD también define métricas para seguir el proceso de desarrollo de la aplicación, útiles para el cliente y la dirección de la empresa, y que pueden ayudar, además de para conocer el estado actual del desarrollo, a realizar mejores estimaciones en proyectos futuros. **(19)**

### Metodología RUP (Proceso Unificado de Desarrollo)

La base de todo desarrollo de software es la calidad, para lograr un producto de buena calidad aplicando RUP se debe ajustar el proceso a las limitaciones que el desarrollo tenga, así como también se debe equilibrar las prioridades.

Esta metodología funciona a través de ciclos, un ciclo está compuesto de 4 fases y al final de este se tiene una versión del producto, en cada fase se pueden tener varias iteraciones.

Las fases mencionadas son las siguientes:

- **Conceptualización:** (Concepción o Inicio): Se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso del sistema.

- **Elaboración:** Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo al alcance definido.
- **Construcción:** Se obtiene un producto listo para su utilización que está documentado y tiene un manual de usuario. Se obtiene 1 o varios release del producto que han pasado las pruebas. Se ponen estos release a consideración de un subconjunto de usuarios.
- **Transición:** El release ya está listo para su instalación en las condiciones reales. Puede implicar reparación de errores.

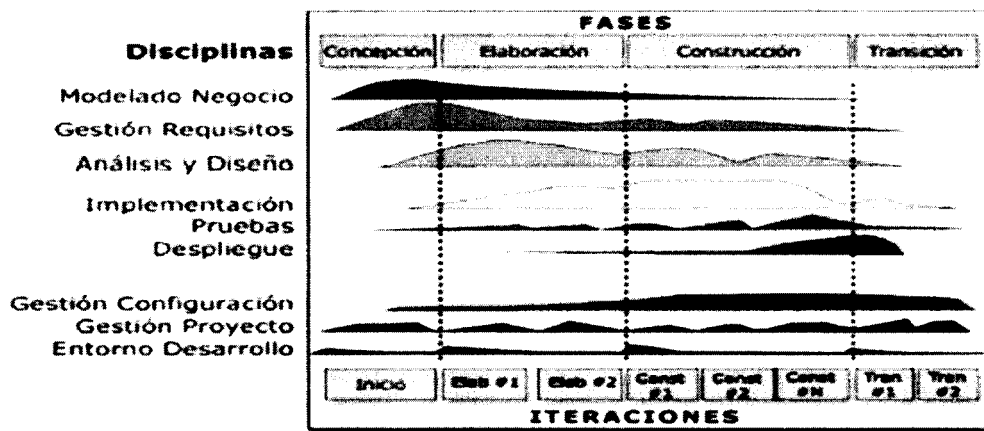


Figura 15:Fases de RUP.

El ciclo de vida de RUP se caracteriza por ser:

1. *Dirigido por casos de uso:* Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

2. *Centrado en la arquitectura*: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los CU relevantes desde el punto de vista de la arquitectura.

3- *Iterativo e Incremental*: Aunque la figura anteriormente mostrada puede sugerir que los flujos de trabajo se desarrollan en cascada, la "lectura" de este gráfico tiene que ser vertical y horizontal. RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Por ejemplo, una iteración de elaboración centra su atención en el análisis y diseño, aunque refina los requerimientos y obtiene un producto con un determinado nivel, pero que irá creciendo incrementalmente en cada iteración. Es práctico dividir el trabajo en partes más pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto. Cada iteración se realiza de forma planificada es por eso que se dice que son miniproyectos

RUP define nueve actividades a realizar en cada fase del proyecto, de las cuales los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de soporte o apoyo. Estas actividades son las mencionadas a continuación:

- 1- Modelo de Negocio
- 2- Requerimientos
- 3- Análisis y diseño
- 4- Implementación
- 5- Prueba
- 6- Instalación
- 7- Administración del proyecto
- 8- Administración de configuración y cambios
- 9- Gestión del entorno o ambiente

Al terminar el ciclo de desarrollo se debe tener una versión funcionando u operando y se podrá empezar un nuevo ciclo. (20)

RUP ofrece las siguientes ventajas comparada con otras metodologías de desarrollo:

- Reducción de riesgos basado en la retroalimentación temprana.
- Pruebas continuas e iterativas promueven una mejor evaluación del estado del proyecto.
- Los patrocinadores reciben evidencia concreta del avance del proyecto.
- Se pueden acomodar mejor los cambios (requerimientos, tácticos y tecnológicos).
- Los problemas más complejos se priorizan.
- Aunque RUP es un proceso de desarrollo de software genérico, se concibió en gran medida para el desarrollo de sistemas basados en programación orientada a objetos.

### 2.6 ¿Por qué Python como lenguaje de programación?

Para el desarrollo del trabajo de diploma es seleccionado el lenguaje de programación Python ya que ofrece la posibilidad de desarrollar aplicaciones o cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation. A continuación les mencionamos varias características de este lenguaje:

- Es un lenguaje multipropósito que al igual que C, C++, C# y Java es utilizado por muchos programadores.
- Posee compiladores en múltiples plataformas como Unix, Windows, OS/2, Mac, Amiga y otros, teniendo más que Java.
- Tiene la posibilidad de embeber su código en la Máquina Virtual de Java, en C y C++ como plug-in, por lo que la biblioteca propuesta pudiera ser utilizada en otros proyectos que utilicen estos lenguajes sin necesidad de migrar a Python obligatoriamente.
- Soporta la Programación Orientada a Objetos donde se destaca la herencia múltiple como uno de sus principales logros y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables.
- Posee una sintaxis clara que se acerca al lenguaje natural, garantizando que todos los programadores tengan un mismo estilo de organizar el código y a su vez permite un fácil entendimiento a los que lo consulten.



- Permite dividir el programa en módulos reutilizables desde otros programas Python, el cual viene con una gran colección de módulos estándar que se pueden utilizar como base de los programas. También hay módulos incluidos que proporcionan E/S de archivos, llamadas al sistema, sockets y hasta interfaces GUI.
- Es un lenguaje de programación multiparadigma. Es decir, permite varios estilos de programación como la Programación Orientada a Objetos, Programación Estructurada, Programación Funcional y Programación Orientada a Aspectos. Otros paradigmas están soportados mediante el uso de extensiones.
- Usa tipado dinámico de datos y *reference counting* (contador de referencias) para el manejo de memoria.
- Es un lenguaje interpretado, lo que quiere decir que no se debe compilar el código antes de su ejecución. En realidad sí que se realiza una compilación, pero esta se realiza de manera transparente para el programador. En ciertos casos, cuando se ejecuta por primera vez un código, se producen unos *bytecodes* que se guardan en el sistema y que sirven para acelerar la compilación implícita que realiza el intérprete cada vez que se ejecuta el mismo código. **(21)**
- Dispone de muchas funciones incorporadas en el propio lenguaje, para el tratamiento de strings, números, archivos, etc. Además, existen muchas bibliotecas que contienen tipos de datos y funciones incorporadas en el propio lenguaje, las cuales ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.
- Posee más chequeos de C ya que al ser un lenguaje interpretado cuando se utiliza el archivo el sistema chequea los errores comunes que cada lenguaje puede tener haciendo énfasis en el tipado ya que este es dinámico. **(22)**

Parte de estas características que son tomadas por muchos programadores e informáticos en general como desventajas y eso los hace optar por otros lenguajes, algunas de ellas son las siguientes:

- Al ser un lenguaje interpretado, trae consigo una menor velocidad de procesamiento del código ya que para ejecutar el código tendría que chequear los errores que puedan haber surgido dentro del código, no así en los demás lenguajes como C o C++ por solo citar dos, ya que estos sí que crean un ejecutable con códigos binarios gracias al compilador asociado a ellos.

- No es un lenguaje apropiado para la programación a bajo nivel (por ejemplo: kernels y drivers) ni tiene el control de la memoria, debido a sus características de lenguaje interpretado y scripting.
- No es adecuado para aplicaciones que requieren una alta capacidad de cómputo, por ejemplo para el procesamiento de imágenes. **(23)**

A continuación se muestran dos de los lenguajes más utilizados en la actualidad para que se tenga en cuenta el por qué de la elección:

### Lenguaje de programación CSharp (C#)

C Sharp es el lenguaje de propósito general diseñado por Microsoft para su plataforma .NET.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET

Este lenguaje tiene las siguientes características:

- Es un lenguaje orientado al desarrollo de componentes (módulos independientes de granularidad mayor que los objetos) ya que los componentes son objetos que se caracterizan por sus propiedades, métodos y eventos y estos aspectos de los componentes están presentes de manera natural en C#.
- En C# todo son objetos: desaparece la distinción entre tipos primitivos y objetos de lenguajes como Java o C++ (sin penalizar la eficiencia como en LISP o Smalltalk).
- El software es robusto y duradero: el mecanismo automático de recolección de basura, la gestión de excepciones, la comprobación de tipos, la imposibilidad de usar variables sin inicializar y hacer conversiones de tipo (castings) no seguras, gestión de versiones, etc. ayudan a desarrollar software de fácil mantenimiento y poco propenso a errores.
- La posibilidad de utilizar C++ puro (código no gestionado o inseguro), la facilidad de interoperabilidad (XML, SOAP, COM, DLLs...) junto con un aprendizaje relativamente sencillo (para los que ya conocen otros lenguajes de programación) hace que el dominio y uso del lenguaje junto a otras tecnologías sea muy apreciado.

Pero como todo lenguaje de software propietario, no es el tipo de lenguaje óptimo para el desarrollo de nuestra biblioteca de estructura de datos avanzadas puesto a que nosotros queremos esta sirva de ayuda a otros sin tener que cobrar por su uso.

### El lenguaje de programación Java

Java es un lenguaje de programación (como C, C++, BASIC, Pascal o Python) que posee características relevantes como las que se presentan a continuación:

- Una misma aplicación puede funcionar en diversos tipos de ordenadores y sistemas operativos: Windows, Linux, Solaris, MacOS-X... así como en otros dispositivos inteligentes.
- Los programas Java pueden ser aplicaciones independientes (que corren en una ventana propia) o "*applets*": pequeños programas interactivos que se encuentran incrustados en una página web y pueden funcionar con cualquier tipo de navegador: Explorer, Netscape, Ópera.
- Se trata de un lenguaje "orientado a objetos". Esto significa que los programas se construyen a partir de módulos independientes, y que estos módulos se pueden transformar o ampliar fácilmente. Un equipo de programadores puede partir de una aplicación existente para extenderla con nuevas funcionalidades.
- Su desarrollo está impulsado por un amplio colectivo de empresas y organizaciones, y conecta con la filosofía de software abierto y entorno colaborativo **(24)**

En resumen, las principales características que posee Java son las siguientes:

*Confiable:* Minimiza los errores que se escapan a la fase de prueba.

*Multiplataforma:* Los mismos archivos binarios funcionan correctamente en Windows/95 y NT, Unix/Motif y Power/Mac.

*Orientado a objetos:* Beneficioso tanto para el proveedor de bibliotecas de clases como para el programador de aplicaciones.

*Robusto:* Los errores se detectan en la etapa de compilación, lo que facilita la depuración. **(25)**

Sin embargo, aunque Java es desde hace poco tiempo licenciada bajo la GPL (General Public Licence), es decir, software libre, también existen problemas con el uso de sus

aplicaciones ya que para utilizar tales aplicaciones hay que tener instalado la Java Virtual Machine (JVM) o Máquina Virtual de Java.

La principal desventaja de los lenguajes basados en máquina virtual, es que efectivamente son más lentos que los lenguajes completamente compilados, debido a la sobrecarga que genera tener una capa de software intermedia entre la aplicación y el hardware de la computadora (26)

### **2.7 ¿Por qué IDLE (Integrated DeveLopment Enviroment) Python 2.5 y PyScripter como IDEs (Integrated Development Enviroment) para la programación de la biblioteca?**

Desde el surgimiento de Python, han sido muchos los IDEs que se han desarrollado para este lenguaje, incluso la consola de Linux soporta sus instrucciones y comandos. Entre los más utilizados se encuentran IDLE Python con sus distintas versiones, que es proporcionado por la organización Python.org y el Eclipse que funciona principalmente para Java pero tiene un plug-in para Python recientemente desarrollado.

Para la programación de todas las estructuras de datos avanzadas se utilizó IDLE Python 2.5, no solo por ser la última versión del IDE que proporciona la organización creadora del lenguaje, Python.org, sino porque, a pesar de que es un poco incómodo, es multiplataforma, y dichas incomodidades desaparecen parcialmente en Linux, ya que el acoplamiento con su consola elimina lo engorroso que puede ser la implementación de estructuras orientadas a objetos en un IDE que optimiza el trabajo procedural y estructurado.

También por problemas de disponibilidad de tiempo y recursos, se utilizó el PyScripter que es un IDE netamente para Microsoft Windows, pero muy cómodo ya que brinda opciones de completamiento de código y traceo, además como ya hemos dicho Python es un lenguaje interpretado y el bytecode que genera es entendible al mismo tiempo por todos los sistemas operativos que lo soportan.

### 2.8 Conclusiones parciales

Una vez realizado el análisis en este capítulo se profundizó en la selección de la metodología y herramientas a utilizar para el desarrollo de este trabajo. Quedando seleccionadas RUP como metodología de desarrollo por su adaptabilidad y documentación, UML como lenguaje de modelado para describir todos los procesos y Visual Paradigm como herramienta CASE para modelar la aplicación. C++ como lenguaje de programación por sus potencialidades y como IDEs de desarrollo IDLE Python 2.5 y PyScripter 2.5. Logrando así un mejor desempeño en el desarrollo del Trabajo de Diploma.

## CAPÍTULO III: Presentación de la solución propuesta

### 3.1 Introducción.

En el presente capítulo se empezará a abordar lo relacionado con la solución propuesta, haciendo uso de las metodologías y herramientas ya seleccionadas.

En el mismo se presentará el modelo de dominio y se ofrecerá su definición a través del glosario de términos, presentando y definiendo además las funcionalidades de dicha solución. También como parte del proceso ingenieril hacemos una descripción del sistema, definiendo los actores y casos de usos que le corresponden, con sus respectivas descripciones para su mejor comprensión.

### 3.2 Entorno donde desarrollará el sistema.

La evaluación del estado del Negocio consiste básicamente en evaluar el estado actual de la organización en la cual el sistema será explotado. Dependiendo de la situación o escenario que se presente, hay varias alternativas de desarrollar este proceso.

Si se determina que no es necesario un modelo completo del negocio se realizará lo que se conoce como Modelo del Dominio.

*Un Modelo del Dominio captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema. (20)*

### 3.2.1 Diagrama de clases del Modelo de Dominio.

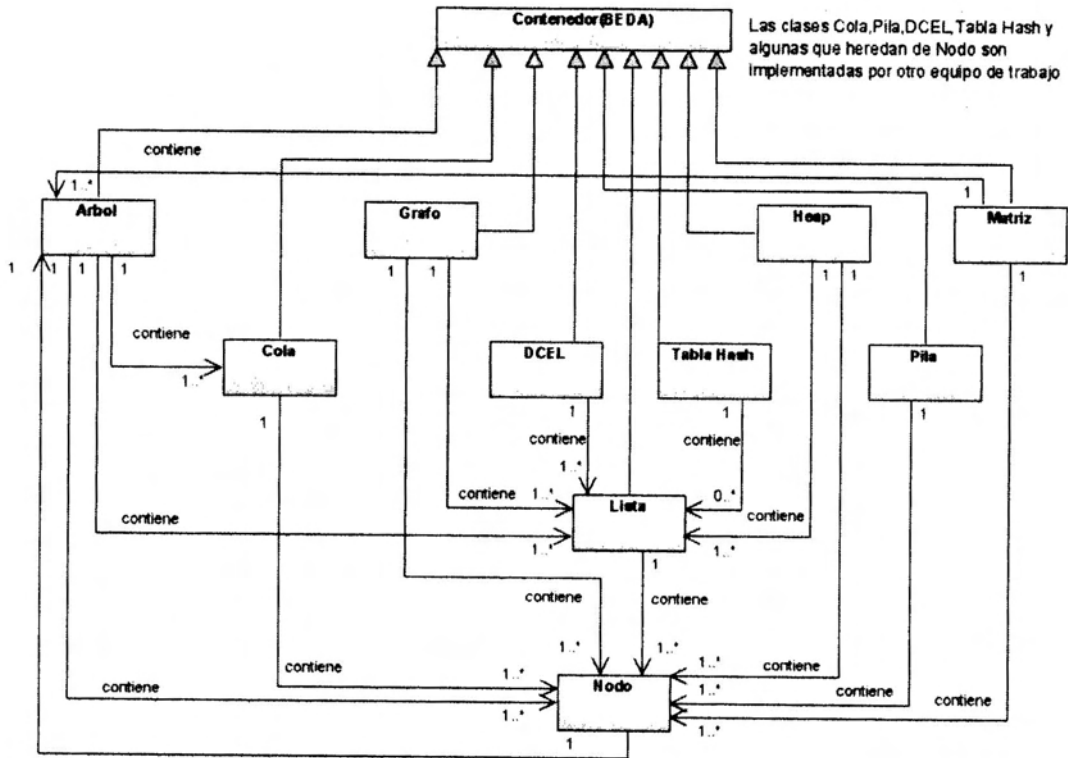


Figura 16: Diagramas de clases del Modelo de Dominio

### 3.2.2 Glosario de Términos del Dominio.

**Nodo:** Es tipo de dato abstracto que posee los atributos necesarios para ser utilizados en diferentes estructuras de datos, la característica principal de estos nodos es la de poseer información valiosa para la estructura y propiciar la navegabilidad entre los otros nodos que componen una estructura. (3)

**Lista:** Una lista es un conjunto de nodos ordenados que tienen la posibilidad de almacenar datos. Una lista tiene un apuntador al elemento siguiente y otro al anterior para garantizar las operaciones relacionadas con este tipo de datos. (3)

**Cola:** Es una estructura de datos que se caracteriza por ser una lista ordenada de datos que se rige por el principio FIFO (first in, first out), es decir, el primero que entra es el primero que sale, donde la función de inserción introduce los datos por el final de la cola y la función de extracción devuelve y elimina los datos de esta. (11)

**Árbol:** Es una estructura de datos que simula a un árbol de la naturaleza, el cual está compuesto por nodos, también son conocidos como subárboles. Cada nodo o subárbol tiene asociado una raíz que porta la información que se quiere almacenar, además de un padre y 0 o varios hijos. (3)

**Grafo:** Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones entre elementos de un conjunto. (3)

**Heap(Montículo):** Es una especie de Árbol Binario sobre una estructura lineal como una Lista o Cola, el cual posee información perteneciente a un conjunto ordenado. Estos tienen la característica de que cada nodo padre tiene un valor mayor que el de todos sus nodos hijos. (3)

**Matriz:** En la Programación Orientada a Objetos es un conjunto de variables del mismo tipo cuyo acceso se realiza por índices. En términos matemáticos una matriz es una ordenación rectangular de elementos algebraicos que pueden sumarse y multiplicarse. En nuestro trabajo utilizaremos las dos versiones aquí expuestas. (27)

**Contenedor(BEDA):** El contenedor BEDA es una abstracción a lo que va a ser esta Biblioteca de Estructuras de Datos, es decir, va a ser el fichero donde se va a guardar el conjunto de clases que se implementarán.

### 3.3 Funcionalidades del sistema propuesto.

#### 3.3.1 Requerimientos Funcionales (RF).

##### Sección de Árboles

El sistema debe ser capaz de:

R1: Insertar elementos en un árbol dado un criterio.

R2: Eliminar elementos en un árbol dado un criterio.

R3: Garantizar la realización óptima de los recorridos en Pre-orden, Pos-orden, Entre-orden y a lo ancho en los árboles a implementar

R4: Reconocer si un árbol está vacío o no.

R5: Devolver el elemento raíz de un árbol.

R6: Realizar la búsqueda de un elemento en un árbol dado un criterio de búsqueda.



### Sección de Grafos:

- R7: Insertar elementos en un grafo dado un criterio de inserción.
- R8: Eliminar elementos en un grafo dado un nodo del grafo.
- R9: Garantizar la realización óptima de los recorridos en profundidad y a lo ancho.
- R10: Reconocer si un grafo está vacío o no.
- R11: Devolver un vértice determinado de un grafo.
- R12: Determinar el camino mínimo entre dos vértices cualquiera.

### Sección de Heaps:

- R13: Insertar elementos en un heap (montículo) dado un criterio de ordenamiento.
- R14: Encontrar el menor elemento en un heap.
- R15: Extraer el menor elemento de un heap.
- R16: Mantener el heap ordenado de acuerdo a un criterio de ordenamiento.

### Sección de Matrices:

- R17: Insertar un elemento en la matriz a través de coordenadas enviadas por el usuario.
- R18: Eliminar un elemento en la matriz a través de coordenadas enviadas por el propio usuario.
- R19: Devolver una fila completa con todos los elementos que pertenezcan a ella y no sean ceros.
- R20: Devolver una columna completa con todos los elementos que pertenezcan a ella y no sean ceros.
- R21: Sumar y multiplicar elementos en la matriz teniendo en cuenta las condiciones para realizar estas operaciones.

### **3.3.2 Requerimientos No Funcionales (RNF).**

Usabilidad: Los futuros usuarios serán programadores o sistemas que utilicen estas estructuras de datos para uso personal o empresarial.

Soporte: Esta biblioteca tendrá asociada una documentación donde se explican en detalles definición, algoritmos, aplicaciones, ventajas y desventajas de cada estructura de datos avanzada.

**Portabilidad:** Esta biblioteca puede ser utilizada en cualquier plataforma y en cualquier aplicación propicia para programar en Python.

**Software:** Sistema Operativo Windows, Sistema Operativo Linux (cualquier distribución), Sistema Operativo Macintosh, Amiga y otros.

**Rendimiento:** Esta biblioteca debe implementar sus métodos de búsqueda, inserción y eliminación de forma tal que estas operaciones sean lo más eficientes posible con el fin de optimizar el espacio ocupado en memoria.

### 3.4 Descripción del sistema propuesto.

#### 3.4.1 Descripción de los actores del sistema a automatizar.

Tabla 2. Actores del sistema a automatizar

Actores	Descripción
Usuario	Rol que representa un sistema o programador que utiliza y se beneficia con las funcionalidades que brinda esta biblioteca de Estructuras de Datos Avanzadas, sobre la cual debe estar documentado e informado de cada acción de estas clases implementadas.

#### 3.4.2 Diagramas de casos de uso del sistema a automatizar.

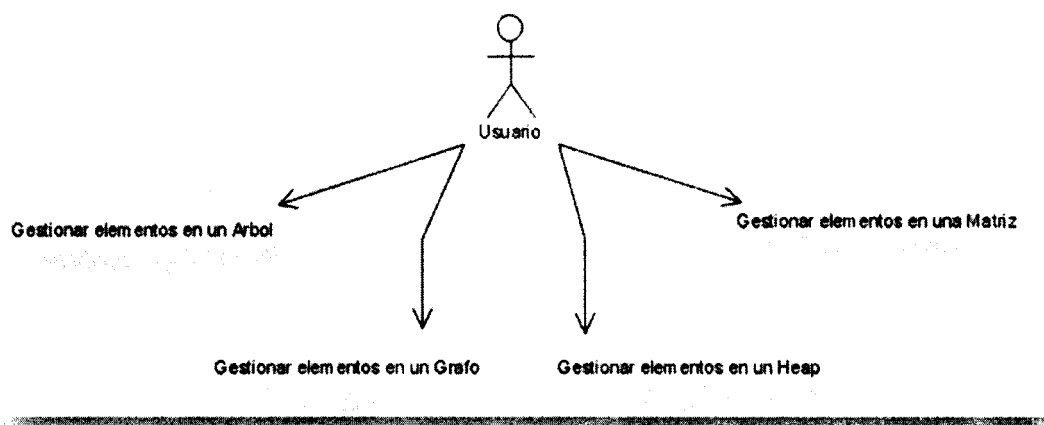


Figura 17: Diagrama de Casos de Uso del Sistema

Tabla 3. Descripción del Caso de Uso Gestionar elementos en un Árbol.

Nombre del CUS	Gestionar elementos en un árbol
Actor	Usuario(Inicia)
Propósito	Permite al usuario Gestionar Elementos en un árbol.
Descripción	El Caso de Uso se inicia cuando el usuario solicita Gestionar Elementos en un árbol, el sistema le pide qué tipo de árbol desea utilizar, el usuario selecciona el tipo de árbol y proporciona los datos necesarios, el sistema realiza la acción seleccionada y termina el CUS
Referencia	R1, R2, R3, R4, R5, R6
Pre-Condiciones	El usuario debe estar documentado e informado acerca de las acciones a solicitar.
Pos-Condiciones	Verdadero si el árbol es hoja Elemento insertado en el árbol Elemento eliminado en el árbol Recorrido realizado en el árbol Elemento raíz del árbol retornado. Elemento Buscado
<b>Flujo normal de los eventos</b>	
<b>Acción del Actor</b>	<b>Respuesta del sistema</b>
1- El usuario solicita gestionar elementos en un árbol	<p>1.1- El sistema permite al usuario escoger el tipo de árbol con el que va a gestionar los elementos:</p> <ul style="list-style-type: none"> <li>• Árbol General</li> <li>• Árbol Binario</li> <li>• Árbol Binario de Búsqueda</li> <li>• Árbol Rojo-Negro</li> <li>• Árbol AVL</li> <li>• Árbol 2 3</li> <li>• Árbol B</li> </ul>

	<p>1.2 - Seleccionado el árbol, el sistema muestra las siguientes opciones:</p> <ul style="list-style-type: none"> <li>▪ Conocer si el árbol es hoja.</li> <li>▪ Insertar elemento</li> <li>▪ Eliminar elemento</li> <li>▪ Realizar recorrido:</li> <li>▪ Pre-Orden</li> <li>▪ Pos-Orden</li> <li>▪ Entre-Orden</li> <li>▪ A lo ancho</li> <li>▪ Buscar un elemento dado</li> <li>▪ Conocer el valor de la raíz del árbol</li> </ul> <p><b>Nota:</b> En el caso del Árbol 2-3 y el Árbol B los recorridos no serán implementados.</p>
<p><b>Sección 1- Conocer si el árbol es hoja.</b></p>	
<p>El usuario selecciona la opción <u>Conocer si el árbol es hoja</u></p>	<p>1.1- El sistema verifica si el árbol tiene hijos 1.2- Si el árbol no tiene hijos o sus hijos son nulos entonces devuelve Verdadero</p>
<p>Flujos Alternos de los Eventos</p>	
<p>Curso Alternativo de los Eventos</p>	<p>1.2- Si el árbol tiene al menos un hijo entonces devuelve Falso.</p>
<p><b>Sección 2- Insertar Elemento</b></p>	
<p>El usuario selecciona la opción <u>Insertar elemento</u></p>	<p>1.1- El sistema verifica si el árbol es hoja. 1.2- Si el árbol no es hoja entonces inserta el elemento</p>
<p>Flujos Alternos de los Eventos</p>	
<p>Curso Alternativo de</p>	<p>1.2-Si el árbol es hoja entonces inserta el elemento en su raíz</p>

los Eventos	
<b>Sección 3- Eliminar Elemento</b>	
El usuario selecciona la opción <u>Eliminar elemento</u>	<p>1.1- El sistema verifica si el árbol es hoja</p> <p>1.2- Si el árbol no es hoja entonces busca el elemento</p> <p>2.1- Si el elemento se encuentra en el árbol, es eliminado</p>
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	<p>1.2-Si el árbol es hoja entonces el sistema muestra un mensaje informando al usuario sobre la situación del árbol.</p> <p>2.1- Si el elemento no se encuentra en el árbol entonces el sistema muestra un mensaje informando al usuario sobre la situación del árbol.</p>
<b>Sección 4- Realizar Recorrido(Pre-Orden)</b>	
El usuario selecciona la opción <u>Realizar Recorrido(Pre-Orden)</u>	<p>1.1- El sistema verifica si el árbol es hoja</p> <p>1.2- Si el árbol no es hoja entonces este es recorrido bajo el criterio del pre-orden.</p>
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si el árbol es hoja entonces el sistema muestra un mensaje informando al usuario sobre la situación del árbol.
<b>Sección 5- Realizar Recorrido(Pos-Orden)</b>	
El usuario selecciona la opción <u>Realizar Recorrido(Pos-Orden)</u>	<p>1.1- El sistema verifica si el árbol es hoja</p> <p>1.2- Si el árbol no es hoja entonces este es recorrido bajo el criterio del pos-orden.</p>
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si el árbol es hoja entonces el sistema muestra un mensaje informando al usuario sobre la situación del árbol.

<b>Sección 6- Realizar Recorrido(Entre-Orden)</b>	
El usuario selecciona la opción <u>Realizar Recorrido(Entre-Orden)</u>	<p>1.1- El sistema verifica si el árbol es hoja</p> <p>1.2- Si el árbol no es hoja entonces este es recorrido bajo el criterio del entre-orden.</p>
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2-Si el árbol es hoja entonces el sistema muestra un mensaje informando al usuario sobre la situación del árbol.
<b>Sección 7- Realizar Recorrido(A lo ancho)</b>	
El usuario selecciona la opción <u>Realizar Recorrido(A lo ancho)</u>	<p>1.1- El sistema verifica si el árbol es hoja</p> <p>1.2- Si el árbol no es hoja entonces este es recorrido bajo el criterio a lo ancho.</p>
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si el árbol es hoja entonces el sistema muestra un mensaje informando al usuario sobre la situación del árbol.
<b>Sección 8- Realizar Búsqueda de un elemento dado</b>	
El usuario selecciona la opción <u>Realizar búsqueda de un elemento dado</u>	<p>1.1- El sistema verifica si el árbol es hoja</p> <p>1.2- Si el árbol es hoja busca el elemento raíz y compara si el elemento que tiene es el elemento buscado</p>
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si el árbol no es hoja entonces busca el elemento por cada uno de los nodos del árbol hasta llegar a las hojas , donde detiene la búsqueda y notifica al usuario del resultado
<b>Sección 9- Conocer el valor de la raíz de un árbol</b>	
El usuario selecciona la opción <u>Conocer el valor de la raíz de un</u>	1.1- El sistema devuelve la raíz del árbol, ya sea nula o con algún valor.

<u>árbol</u>	<u>Nota:</u> Esta sección se toma en cuenta para otras funciones internas de las demás secciones.
--------------	---

Tabla 4. Descripción del Caso de Uso Gestionar elementos en un Heap (Montículo).

Nombre del CUS	Gestionar elementos en un heap(montículo)
Actor	Usuario(Inicia)
Propósito	Permite al usuario Gestionar Elementos en un heap.
Descripción	El Caso de Uso se inicia cuando el usuario solicita Gestionar Elementos en un heap, el sistema le pide qué tipo de heap desea utilizar, el usuario selecciona el tipo de heap y proporciona los datos necesarios, el sistema realiza la acción seleccionada y termina el CUS
Referencia	R13, R14, R15, R16
Pre-Condiciones	El usuario debe estar documentado e informado acerca de las acciones a solicitar.
Pos-Condiciones	<ol style="list-style-type: none"> <li>1- Elemento insertado en el heap</li> <li>2- Menor elemento del heap devuelto</li> <li>3- Menor elemento del heap devuelto y eliminado del mismo</li> <li>4- Mantenido el orden del heap.</li> </ol>

**Flujo normal de los eventos**

Acción del Actor	Respuesta del sistema
1- El usuario solicita gestionar elementos en un heap	<p>1.1- El sistema permite al usuario escoger el tipo de heap con el que va a gestionar los elementos:</p> <ul style="list-style-type: none"> <li>• Heap Binario</li> <li>• Heap Binomial</li> <li>• Heap Fibonacci</li> <li>• Pairing Heap (Heap Autoajustable)</li> </ul> <p>1.2- Seleccionado el heap, el sistema muestra las siguientes</p>

	<p>opciones:</p> <ul style="list-style-type: none"> <li>• Insertar elemento</li> <li>• Encontrar menor elemento</li> <li>• Extraer menor elemento</li> </ul>
<b>Sección 1- Insertar Elemento.</b>	
<p>El usuario selecciona la opción <u>Insertar elemento</u></p>	<p>1- El sistema inserta el elemento</p> <p>2- El sistema ordena y mantiene el heap ordenado</p>
<b>Sección 2- Encontrar Menor Elemento.</b>	
<p>El usuario selecciona la opción <u>Encontrar menor elemento</u></p>	<p>1.1- El sistema encuentra el menor elemento del heap.</p> <p>1.2- El sistema lo devuelve al usuario</p>
<b>Sección 3- Extraer Menor Elemento</b>	
<p>El usuario selecciona la opción <u>Extraer menor elemento</u></p>	<p>1.1- El sistema encuentra el menor elemento del heap.</p> <p>1.2- El sistema lo elimina del heap.</p> <p>2.1- El sistema ordena y mantiene el heap ordenado.</p> <p>3.1- El sistema lo devuelve al usuario.</p>
<b>Sección 4- Decrementar Dato</b>	
<p>El usuario selecciona la opción <u>Decrementar dato (solo para Heap Binomial)</u></p>	<p>1.1- El sistema comprueba que el dato entrado es menor que el existente.</p> <p>1.2- Si es menor asigna el nuevo dato al elemento.</p> <p>2.1- El sistema ordena y mantiene el heap ordenado.</p>
<b>Flujos Alternos de los Eventos</b>	
<p>Curso Alternativo de los Eventos</p>	<p>1.2- Si no es menor no puede realizar la operación y muestra al usuario un mensaje informándole de lo ocurrido</p>



<b>Sección 5- Eliminar Elemento</b>	
1- El usuario selecciona la opción Eliminar elemento(solo para Heap Binomial)	<p>1.1- El sistema utiliza el método de la sección 2 para saber el menor dato existente en el heap.</p> <p>1.2- El sistema utiliza el método de la sección 4 para decrementar el dato del elemento que se va a eliminar a un valor menor que el menor valor existente en el heap.</p> <p>2.1- El sistema ordena y mantiene el heap ordenado.</p> <p>2.2- El sistema utiliza el método de la sección 3 para eliminar el elemento, ya que este tendría el menor valor existente en el heap.</p>

Tabla 5. Descripción del Caso de Uso Gestionar elementos en una Matriz.

Nombre del CUS	Gestionar elementos en una Matriz
Actor	Usuario(Inicia)
Propósito	Permite al usuario Gestionar Elementos en una Matriz.
Descripción	El Caso de Uso se inicia cuando el usuario solicita Gestionar Elementos en una Matriz y realiza las acciones pertinentes con el fin de interactuar con esta estructura de dato avanzada
Referencia	R17, R18, R19, R20, R21
Pre-Condiciones	El usuario debe estar documentado e informado acerca de las acciones a solicitar.
Pos-Condiciones	<p>1- Elemento insertado en la matriz en las coordenadas dadas por el usuario.</p> <p>2- Elemento eliminado en la matriz en las coordenadas dadas por el usuario.</p> <p>3- Elemento buscado en la matriz en las coordenadas dadas por el usuario.</p> <p>4- Fila devuelta según la solicitud del usuario.</p> <p>5- Columna devuelta según la solicitud del usuario.</p> <p>6- Matriz sumada a otra matriz.</p>

	7- Matriz multiplicada por un escalar.
<b>Flujo normal de los eventos</b>	
<b>Acción del Actor</b>	<b>Respuesta del sistema</b>
1- El usuario solicita gestionar elementos en una matriz	<p>1.1- El sistema le permite al usuario crear un objeto de la matriz</p> <p>y le muestra las siguientes opciones:</p> <ul style="list-style-type: none"> <li>• Insertar elemento en la matriz.</li> <li>• Eliminar elemento en la matriz.</li> <li>• Buscar elemento en la matriz.</li> <li>• Devolver cierta columna de la matriz.</li> <li>• Devolver cierta fila de la matriz.</li> <li>• Sumar la matriz con otra matriz.</li> <li>• Multiplicar la matriz por un escalar.</li> </ul>
<b>Sección 1- Insertar elemento en la matriz</b>	
El usuario selecciona la opción <u>Insertar elemento en la matriz</u>	1.2 – El sistema inserta el elemento según las coordenadas dadas por el usuario.
<b>Flujos Alternos de los Eventos</b>	
Curso Alternativo de los Eventos	1.2- Si ya existe elemento en las coordenadas dadas por el usuario en la matriz, el sistema levanta un mensaje dándole a conocer al usuario que ya existe un elemento en las coordenadas dadas.
<b>Sección 2- Eliminar elemento en la matriz</b>	
El usuario selecciona la opción <u>Eliminar elemento en la matriz</u>	1.1- El sistema elimina el elemento según las coordenadas dadas por el usuario.

Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si el elemento no existe en la matriz, el sistema levanta un mensaje de notificación dándole a conocer al usuario que no existe tal elemento que desea eliminar en las coordenadas dadas.
<b>Sección 3- Realizar búsqueda en una matriz</b>	
El usuario selecciona la opción <u>Realizar búsqueda en una matriz</u>	1.1- El sistema realiza la búsqueda de un elemento según las coordenadas dadas por el usuario
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si el elemento no existe en la matriz, el sistema levanta un mensaje de notificación dándole a conocer al usuario de que el elemento no existe en las coordenadas dadas.
<b>Sección 4- Devolver una fila de la matriz</b>	
El usuario selecciona la opción <u>Devolver una fila de la matriz</u>	1.1- El sistema busca los elementos que hay en una fila según el número de fila solicitadas por el usuario.  1.2 - Si existe la fila, entonces es devuelta.
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si no existen elementos en la fila solicitada por el usuario, el sistema envía un mensaje de notificación al usuario dándole a conocer que la fila esta vacía o no existe.
<b>Sección 5- Devolver una columna de la matriz</b>	
El usuario selecciona la opción <u>Devolver una columna de la matriz</u>	1.1- El sistema busca los elementos que hay en una columna según el número de columna solicitadas por el usuario.  1.2 - Si existe la columna, entonces es devuelta.

Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si no existen elementos en la columna solicitada por el usuario, el sistema envía un mensaje de notificación al usuario dándole a conocer que la columna esta vacía o no existe.
<b>Sección 6- Sumar dos matrices</b>	
El usuario selecciona la opción <u>Sumar Matrices</u>	1.1- El sistema verifica si las matrices tienen las mismas dimensiones.
	1.2- Si es tienen las mismas dimensiones, entonces se suman las matrices.
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si no tienen las mismas dimensiones entonces se le notifica al usuario el problema existente.
<b>Sección 7- Multiplicar la matriz con un escalar</b>	
El usuario selecciona la opción <u>Multiplicar la matriz por un escalar</u>	1.1- El sistema recorre la matriz en busca de elementos para multiplicarlos por el escalar enviado por el usuario.
	1.2- Si la matriz no está vacía entonces multiplica por el escalar todos los datos pertenecientes a ella.
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si la matriz está vacía entonces se le notifica al usuario acerca del problema en cuestión.

Tabla 6. Descripción del Caso de Uso Gestionar elementos en un Grafo.

Nombre del CUS	Gestionar elementos en un grafo
Actor	Usuario(Inicia)

<b>Propósito</b>	Permite al usuario Gestionar Elementos en un grafo.
<b>Descripción</b>	El Caso de Uso se inicia cuando el usuario solicita Gestionar Elementos en un grafo, el sistema le pide qué tipo de grafo desea utilizar, el usuario selecciona el tipo de grafo y proporciona los datos necesarios, el sistema realiza la acción seleccionada y termina el CUS
<b>Referencia</b>	R7, R8, R9, R10, R11, R12
<b>Pre-Condiciones</b>	El usuario debe estar documentado e informado acerca de las acciones a solicitar.
<b>Pos-Condiciones</b>	<ol style="list-style-type: none"> <li>1- Elemento insertado en el grafo</li> <li>2- Elemento eliminado en el grafo</li> <li>3- Recorrido realizado en el grafo</li> <li>4- Verdadero si la lista de vértices del grafo esta vacía</li> <li>5- Elemento del grafo retornado.</li> <li>6- Camino mínimo entre vértices devuelto</li> </ol>

**Flujo normal de los eventos**

<b>Acción del Actor</b>	<b>Respuesta del sistema</b>
1- El usuario solicita gestionar elementos en un grafo	<p>1.1- El sistema permite al usuario escoger el tipo de árbol con el que va a gestionar los elementos:</p> <ul style="list-style-type: none"> <li>• Grafo con Lista de Adyacencia</li> <li>• Grafo con Matriz de Adyacencia</li> </ul> <p>1.2 - Seleccionado el árbol, el sistema muestra las siguientes opciones:</p> <ul style="list-style-type: none"> <li>• Insertar elemento.</li> <li>• Eliminar elemento</li> <li>• Realizar recorrido:</li> <li>• En Profundidad</li> <li>• A lo ancho</li> <li>• Saber si el grafo esta vacío o no</li> </ul>

	<ul style="list-style-type: none"> <li>• Obtener un vértice determinado</li> <li>• Determinar el camino mínimo entre vértices.</li> </ul>
<b>Sección 1- Insertar Elemento</b>	
El usuario selecciona la opción <u>Insertar elemento</u>	1.1- El sistema verifica si la lista de vértices esta vacía.
<b>Sección 2- Eliminar Elemento</b>	
El usuario selecciona la opción <u>Eliminar elemento</u>	1.1- El sistema busca el elemento en todas las partes en que se encuentre dentro del grafo. 1.2- Si lo encuentra lo elimina
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si no lo encuentra el grafo permanece invariable
<b>Sección 3- Realizar Recorrido(En Profundidad)</b>	
El usuario selecciona la opción <u>Realizar recorrido(en profundidad)</u>	1.1- El sistema inicializa los vértices para hacer el recorrido 1.2- El grafo es recorrido bajo el criterio en profundidad
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- El sistema notifica al usuario si existen problemas.
<b>Sección 4- Realizar Recorrido(A lo ancho)</b>	
El usuario selecciona la opción <u>Realizar Recorrido(A lo ancho)</u>	1.1- El sistema inicializa los vértices para hacer el recorrido 1.2- El grafo es recorrido bajo el criterio a lo ancho.

<b>Sección 5- Conocer si el grafo está vacío</b>	
El usuario selecciona la opción <u>Insertar elemento</u>	<p>1.1- El sistema verifica si la lista de vértices esta vacía.</p> <p>1.2- Si la lista de vértices está vacía devuelve Verdadero</p>
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si la lista de vértices tiene al menos un vértice entonces devuelve Falso.
<b>Sección 6- Obtener un vértice determinado</b>	
El usuario selecciona la opción <u>Obtener un vértice determinado</u>	<p>1.1- El sistema busca el elemento.</p> <p>1.2- Si lo encuentra lo devuelve</p>
Flujos Alternos de los Eventos	
Curso Alternativo de los Eventos	1.2- Si no lo encuentra el grafo permanece invariable.
<b>Sección 7- Encontrar el camino mínimo entre 2 vértices</b>	
El usuario selecciona la opción <u>Encontrar el camino mínimo entre 2 vértices</u>	<p>El sistema inicializa los vértices para buscar el camino mínimo</p> <p>1.2- El sistema compara sucesivamente el valor de los pesos de cada uno de las aristas formando un árbol de expansión mínimo.</p> <p>1.3- El sistema devuelve el camino entre ambos vértices</p>
<b>Sección 8- Encontrar el camino mínimo entre un vértice y todos los demás</b>	
El usuario selecciona la opción <u>Encontrar el camino mínimo entre un</u>	<p>1.1- El sistema inicializa los vértices para buscar los caminos mínimos</p> <p>1.2- El sistema compara sucesivamente el valor de los pesos de cada uno de las aristas computando las distancias de cada</p>

<p><u>vértice y todos los demás</u></p>	<p>uno de los vértices con respecto al vértice origen</p> <p>1.3- El sistema devuelve todas las distancias mínimas con respecto al vértice origen.</p>
<p><b>Sección 9- Encontrar el camino mínimo entre todos los vértices</b></p>	
<p>El usuario selecciona la opción <u>Encontrar el camino mínimo entre todos los vértices</u></p>	<p>1.1- El sistema inicializa los valores para buscar los caminos mínimos</p> <p>1.2- El sistema compara sucesivamente el valor de los pesos de cada uno de las aristas computando las distancias de todos los vértices.</p> <p>1.3- El sistema devuelve todas las distancias mínimas de todos los vértices.</p>

### 3.5 Conclusiones parciales.

En este capítulo se definieron importantes aspectos correspondientes al desarrollo de la solución propuesta, se comprendió mejor el funcionamiento de la biblioteca a través de la descripción de los diferentes casos de usos, así como de la definición del modelo de dominio. Se ganó claridad en cuanto a las características del sistema que se desea construir y se sentaron las bases para las restantes fases del proceso de diseño e implementación de este.



## CAPÍTULO IV: Construcción de la solución propuesta

### 4.1 Introducción.

En el presente capítulo se diseña el sistema, se exponen los diagramas de clases que muestran las relaciones entre los subsistemas involucrados en estos CU, así como la descripción de los mismos. Se describen también los patrones seguidos a la hora del diseño y de la forma que fueron utilizados.

### 4.2 Patrones.

#### 4.2.1 Patrones de arquitectura utilizados.

##### Características principales de la Arquitectura Orientada a Objetos.

- 1- Los componentes del estilo se basan en principios Orientados a Objetos: encapsulamiento, herencia y polimorfismo.
- 2- Los objetos y sus interacciones son el centro en el diseño de la arquitectura y en la estructura de la aplicación.
- 3- En general la distribución de objetos es transparente y en el estado de arte de la tecnología apenas importa si los objetos son locales o remotos.

##### Ventajas de la Arquitectura Orientada a Objetos.

- 1- La ventaja principal de esta arquitectura es el desarrollo rápido de aplicaciones informáticas con tecnologías orientadas a objetos. Con este soporte se desarrollan sistemas operativos OO, lenguajes OO, sistemas de gestión de bases de datos OO. Además esta arquitectura constituye un buen laboratorio de pruebas de robustez de sistemas software complejo. **(28)**
- 2- Se puede modificar la implementación de un objeto sin afectar a sus clientes. Así mismo es posible descomponer problemas en colecciones de agentes en interacción. Además, un objeto es ante todo una entidad reutilizable en el entorno de desarrollo. **(28)**

### Desventajas de la Arquitectura Orientada a Objetos.

- 1- El principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.

### **4.2.2 Patrones de diseño utilizados.**

Los patrones GRASP (General Responsibility Assignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Es importante entender y poder aplicar estos principios durante la preparación de un diagrama de interacción, pues un diseñador de software sin mucha experiencia en la tecnología de objetos debe dominarlos cuanto antes: constituyen el fundamento de cómo se diseñará el sistema.

Experto: Consiste en asignar una responsabilidad al experto en información: el módulo que cuenta con la información necesaria para cumplir la responsabilidad. Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Hay que tener en cuenta que esto es aplicable mientras estemos considerando los mismos aspectos del sistema:

- Lógica de negocio
- Persistencia a la base de datos
- Interfaz de usuario

Beneficios:

- 1- Se conserva el encapsulamiento ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.
- 2- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase sencillas y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.

Creador: Consiste en asignar a una clase la responsabilidad de crear una instancia de otra clase. Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto

producido en cualquier evento. Al escogerlo como creador, puede soportar un bajo acoplamiento, una mayor claridad, el encapsulamiento y la reusabilidad. **(29)**

Beneficios:

- 1- Brinda soporte a un bajo acoplamiento, lo cual supone menos dependencias y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que conllevaron a elegirla como el parámetro adecuado.

Bajo Acoplamiento: Consiste en asignar una responsabilidad para mantener bajo el acoplamiento. Debe haber pocas dependencias entre las clases. Para determinar el nivel de acoplamiento de clases, son muy buenos los diagramas de colaboración. Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda. Siempre hay que considerar las ventajas de la delegación respecto de la herencia. **(29)**

Beneficios:

- 1- No se afectan por cambios de otros componentes.
- 2- Fáciles de entender por separado.
- 3- Fáciles de reutilizar.

Alta Cohesión: Consiste en asignar una responsabilidad para mantener alta la cohesión. En cuanto al diseño de objetos, la cohesión (o de manera más específica la cohesión funcional) es una medida de fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento. Un elemento con responsabilidad altamente relacionada, y que no hace una gran cantidad de trabajo tiene alta cohesión. A menudo, las clases con baja cohesión representan un “grano grande” de abstracción, o se les ha asignado responsabilidades que debería haberse delegado en otros objetos.

Beneficios:

- 1- Mejoran la claridad y la facilidad con que se entiende el diseño.
- 2- Se simplifican el mantenimiento y las mejoras en funcionalidad
- 3- A menudo se genera un bajo acoplamiento.
- 4- La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico. **(30)**

**Polimorfismo:** El polimorfismo significa “asignar el mismo nombre a servicios en varios objetos”. Cuando por el tipo varían las alternativas o comportamientos afines, las responsabilidades del comportamiento se asignarán mediante operaciones polimórficas a los tipos en que el comportamiento presenta variantes. El uso del patrón Polimorfismo está acorde al espíritu del patrón Experto. Si podemos caracterizar Experto como el patrón fundamental táctico, Polimorfismo será el más importante patrón estratégico en el diseño orientado a objetos.

Beneficios:

- 1- Es fácil agregar las futuras extensiones que requieren las variaciones imprevistas.

**Nota al lector:** Este tipo de patrones se pueden utilizar a la hora de confeccionar los diagramas de interacción y las clases, tanto del análisis como del diseño. Los diagramas de interacción de este trabajo de diploma se encuentran en el tema 1 del Anexo del documento. (30)

#### 4.3 Diagramas de clases del diseño.

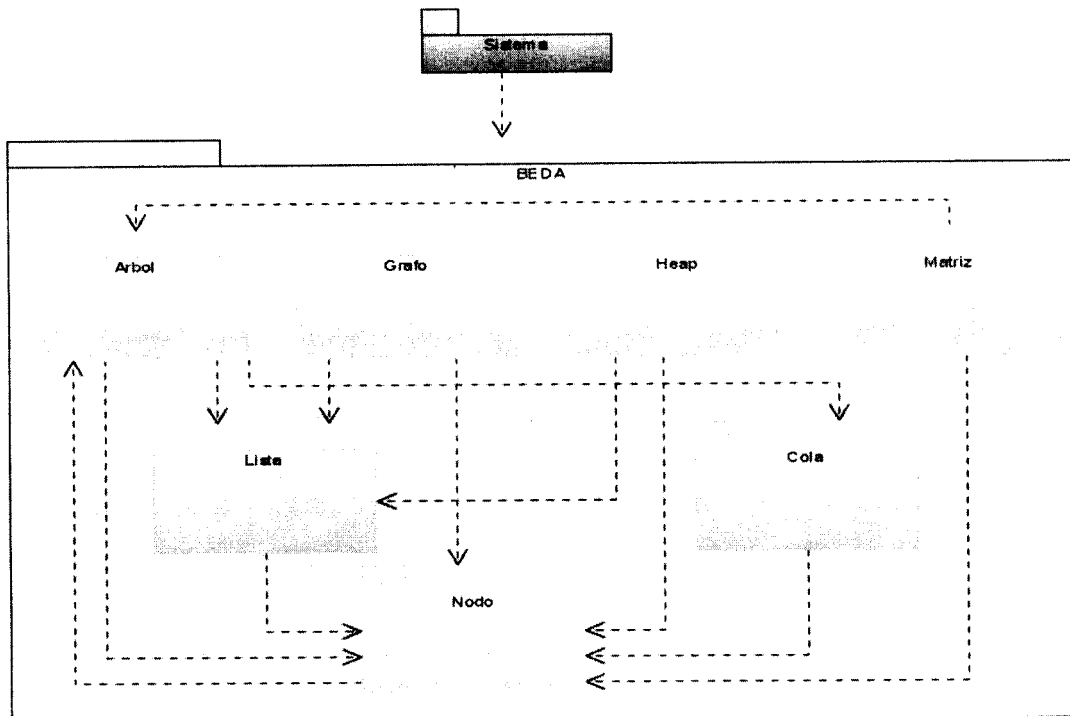


Figura 18: Diagrama General del Diseño organizado por paquetes.

4.3.1 Paquete Nodo.

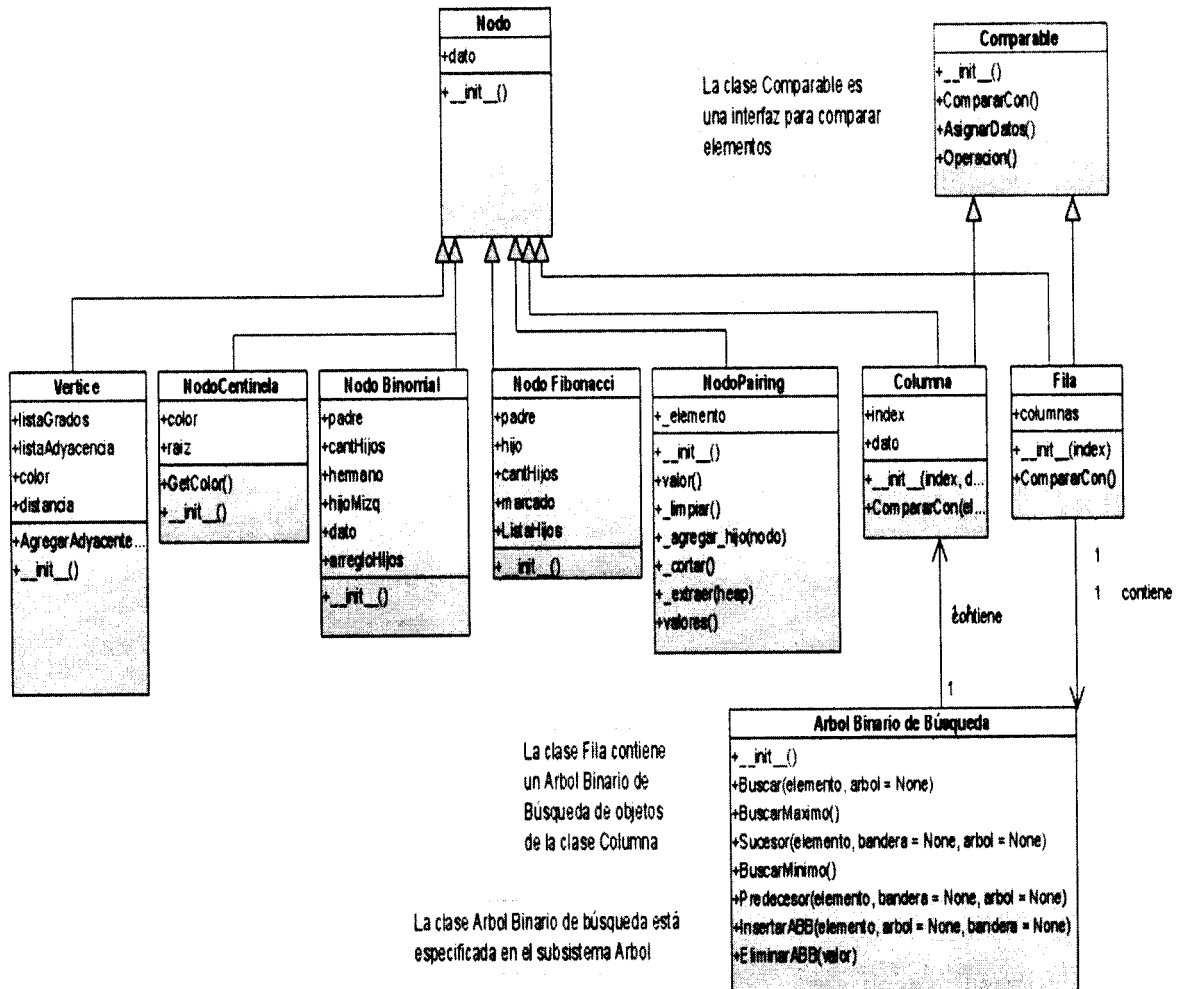


Figura 19: Diagrama de Clases del Diseño del paquete Nodo.

4.3.2 Paquete Árbol.

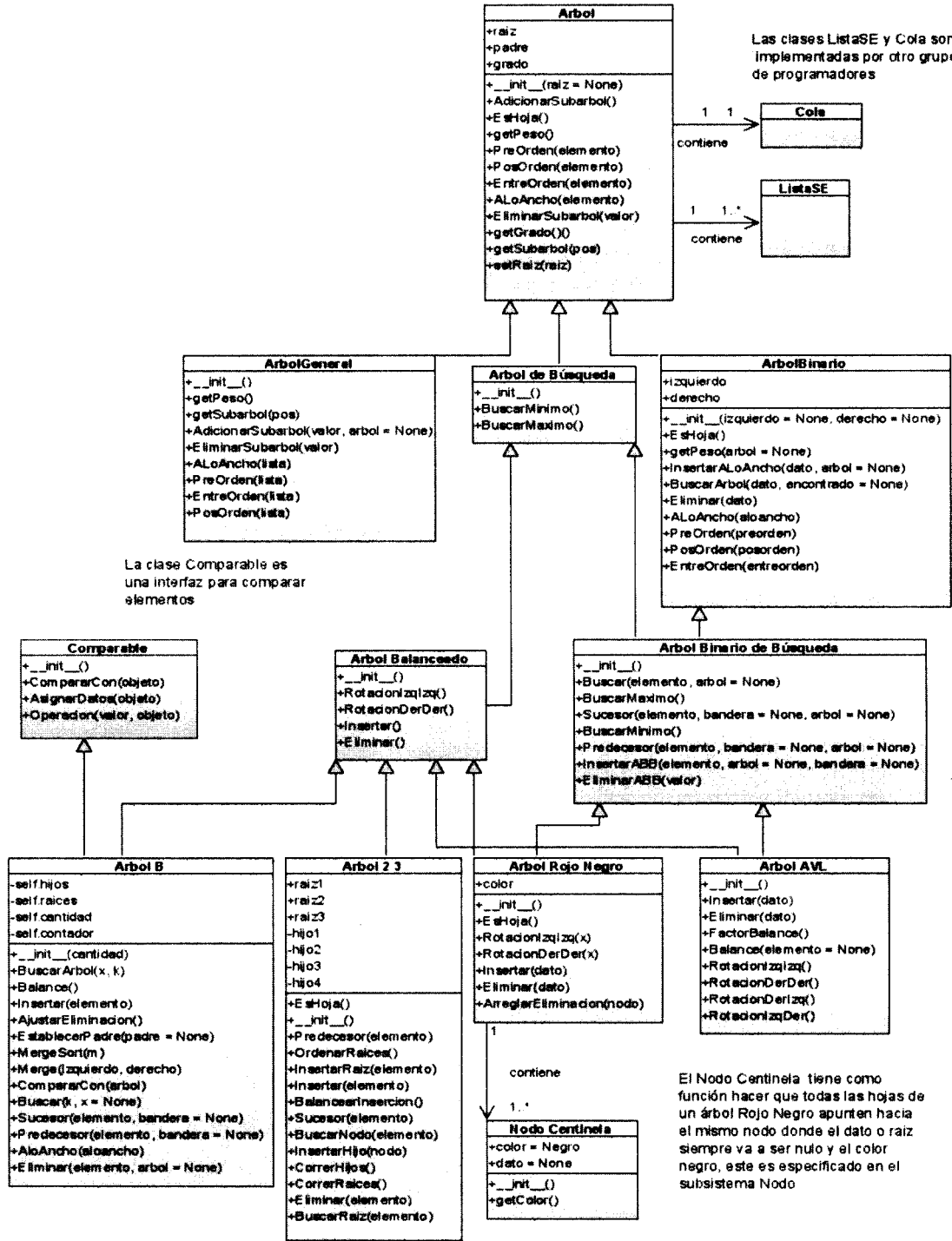


Figura 20: Diagrama de Clases del Diseño del paquete Árbol.

### 4.3.3 Paquete Grafo

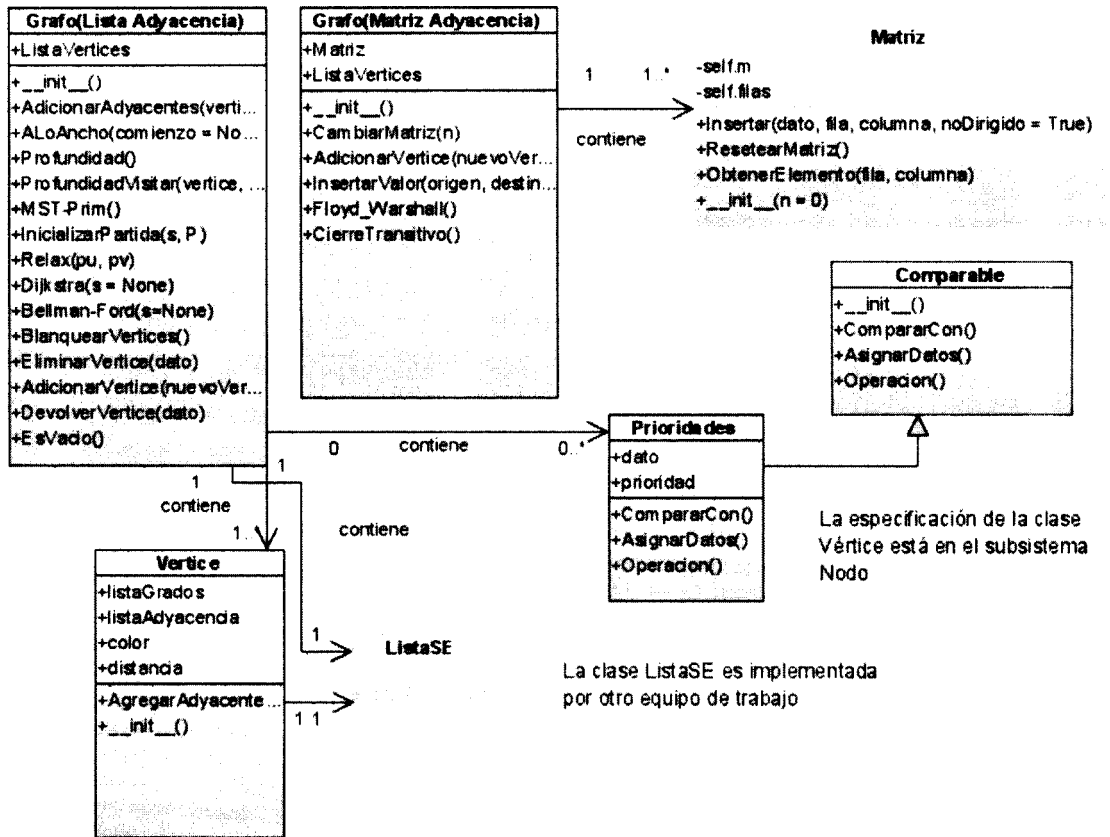


Figura 21: Diagrama de Clases del Diseño del paquete Grafo.

4.3.4 Paquete Heap.

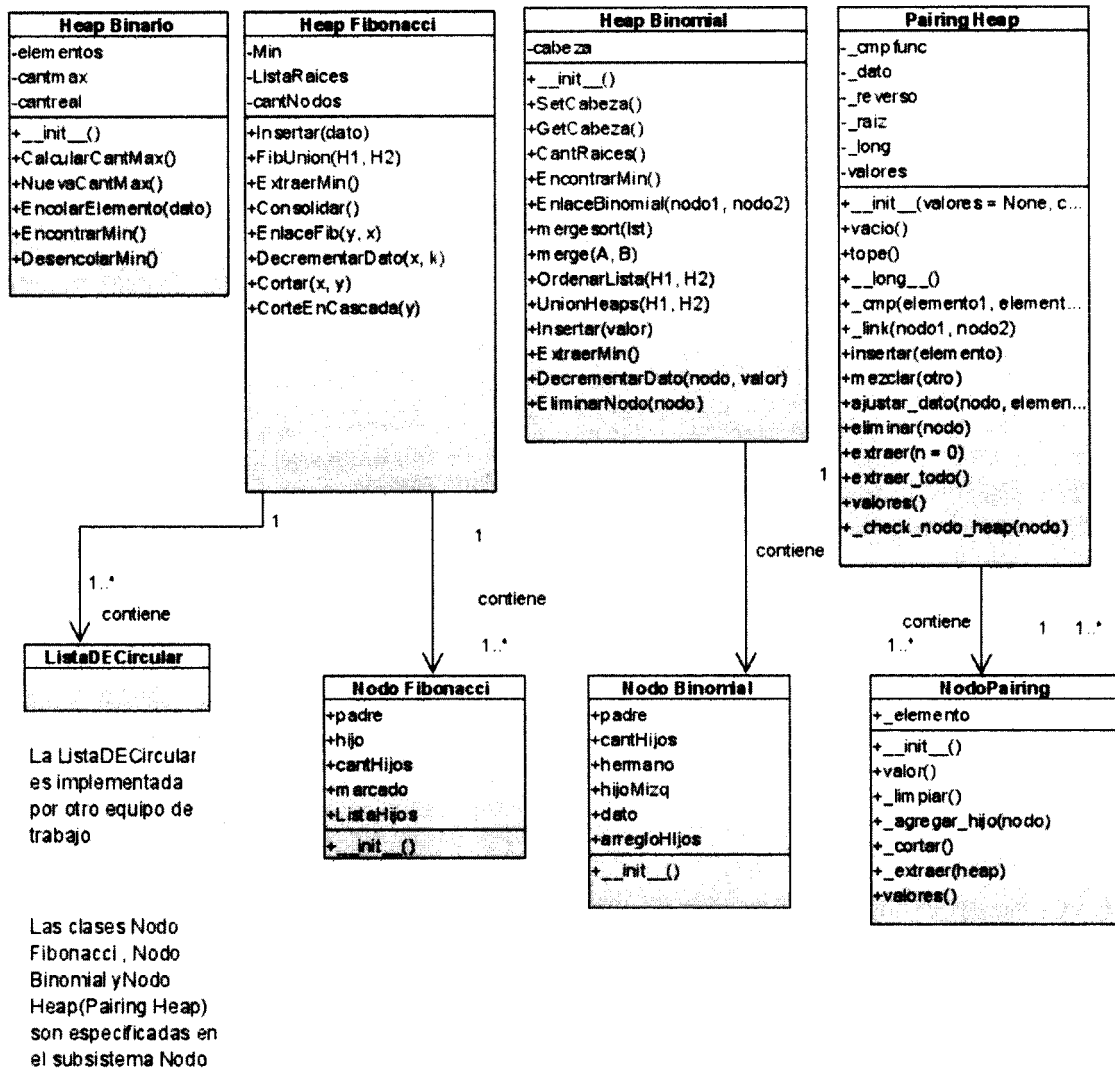


Figura 22: Diagrama de Clases del Diseño del paquete Heap.



4.3.5 Paquete Matriz.

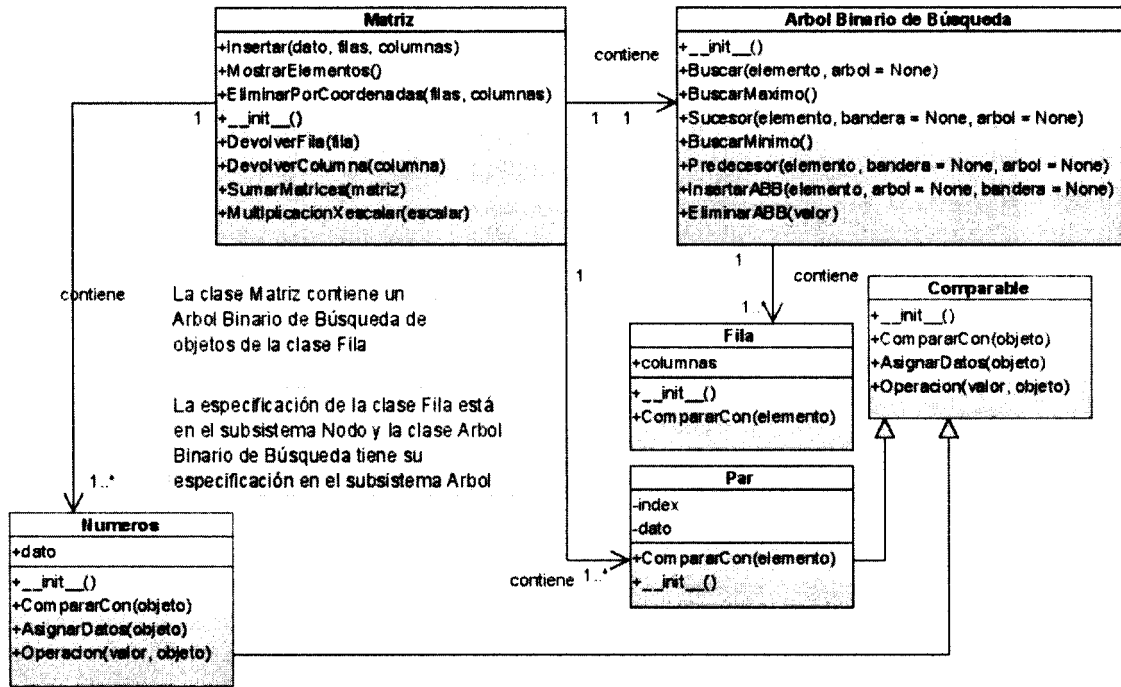


Figura 23: Diagrama de Clases del Diseño del paquete Matriz.



Tabla 7. Pruebas realizadas a los Casos de Uso Gestionar Elementos en un Heap y Gestionar Elementos en un Grafo.

<b>Caso de Uso</b>	Gestionar Elementos en un Heap
<b>Caso de Prueba</b>	Insertar Elementos en un Heap
<b>Entrada (Binario)</b>	1000 elementos de forma aleatoria
<b>Salida</b>	Todos son insertados
<b>Entrada (Binomial)</b>	1000 elementos de forma aleatoria
<b>Resultado</b>	Todos son insertados
<b>Entrada (Fibonacci)</b>	1000 elementos de forma aleatoria
<b>Resultado</b>	Todos son insertados
<b>Entrada (Pairing)</b>	1000 elementos de forma aleatoria
<b>Resultado</b>	Todos son insertados
<b>Caso de Uso</b>	Gestionar Elementos en un Heap
<b>Caso de Prueba</b>	Extraer el Menor Elemento en un Heap con elementos.
<b>Entrada (Binario)</b>	1000 elementos de forma aleatoria y extraer sucesivamente 100 veces el mínimo
<b>Salida</b>	Mínimo = 113
<b>Entrada (Binomial)</b>	1000 elementos de forma aleatoria y extraer sucesivamente 100 veces el mínimo
<b>Resultado</b>	Mínimo = 114
<b>Entrada (Fibonacci)</b>	1000 elementos de forma aleatoria y extraer sucesivamente 100 veces el mínimo
<b>Resultado</b>	Mínimo = 96
<b>Entrada (Pairing)</b>	1000 elementos de forma aleatoria y extraer sucesivamente 100 veces el mínimo
<b>Resultado</b>	Mínimo = 107

<b>Caso de Uso</b>	Gestionar Elementos en un Heap
<b>Caso de Prueba</b>	Extraer el Menor Elemento en un Heap Vacío.
<b>Entrada (Binario)</b>	Extraer el mínimo
<b>Salida</b>	"Heap Vacío"
<b>Entrada (Binomial)</b>	Extraer el mínimo
<b>Resultado</b>	"Heap Vacío"
<b>Entrada (Fibonacci)</b>	Extraer el mínimo
<b>Resultado</b>	None
<b>Entrada (Pairing)</b>	Extraer el mínimo
<b>Resultado</b>	"Heap Vacío"
<b>Caso de Uso</b>	Gestionar Elementos en un Heap
<b>Caso de Prueba</b>	Decrementar Dato Elemento en un Heap Binomial
<b>Entrada (Binomial)</b>	1000 elementos de forma aleatoria y decrementar el dato a todos los hijos de los que se encuentran en la lista de raíces a una unidad menos que sus padres.
<b>Resultado</b>	Padres e hijos cambian de lugar.
<b>Caso de Uso</b>	Gestionar Elementos en un Heap
<b>Caso de Prueba</b>	Eliminar Elemento en un Heap Binomial
<b>Entrada (Binomial)</b>	1000 elementos de forma aleatoria y eliminar todos los hijos hasta que no haya y entonces eliminar el padre y pasar al proximo nodo para repetir operacion
<b>Resultado</b>	Heap Vacio
<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Insertar Elementos en un Grafo(300)
<b>Entrada (Grafo Lista Adyacencia)</b>	300 elementos de forma aleatoria

<b>Salida</b>	Elementos insertados
<b>Entrada(Grafo Matriz)</b>	300 elementos de forma aleatoria
<b>Resultado</b>	Elementos Insertados
<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Insertar Elementos en un Grafo(1000)
<b>Entrada (Grafo Lista Adyacencia)</b>	1000 elementos de forma aleatoria
<b>Salida</b>	Elementos insertados
<b>Entrada(Grafo Matriz)</b>	1000 elementos de forma aleatoria
<b>Resultado</b>	Elementos No Insertados Porque el orden de Inserción es (n2) (Proponer utilizar las Matrices implementadas en las Poco Densas)
<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Eliminar Elementos en un Grafo No Vacío
<b>Entrada (Grafo Lista Adyacencia)</b>	300 elementos de forma aleatoria y se borra el ultimo elemento hasta que el primero sea nulo
<b>Salida</b>	Grafo Vacío
<b>Entrada(Grafo Matriz)</b>	No tiene implementada la funcionalidad ya que se ajusta por el método CambiarMatriz(n)
<b>Resultado</b>	
<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Eliminar Elementos en un Grafo Vacío
<b>Entrada (Grafo Lista Adyacencia)</b>	300 elementos de forma aleatoria y se borra el ultimo elemento hasta que el primero sea nulo
<b>Salida</b>	Grafo Vacío
<b>Entrada(Grafo</b>	No tiene implementada la funcionalidad ya que se ajusta por

<b>Matriz)</b>	el método CambiarMatriz(n)
<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Realizar Recorrido A Lo Ancho
<b>Entrada (Grafo Lista Adyacencia)</b>	1000 elementos de forma aleatoria y se recorre a lo ancho
<b>Salida</b>	Como tal el sistema no muestra un resultado, pero el grafo queda recorrido a lo ancho
<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Realizar Recorrido En Profundidad
<b>Entrada (Grafo Lista Adyacencia)</b>	1000 elementos de forma aleatoria y se recorre en profundidad
<b>Salida</b>	Como tal el sistema no muestra un resultado, pero el grafo queda recorrido en profundidad

<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Formar Árbol de Expansión Mínima
<b>Entrada (Grafo Lista Adyacencia)</b>	500 elementos de forma aleatoria y se ejecuta el método MST_Prim
<b>Salida</b>	Como tal el sistema no muestra un resultado, pero el árbol de expansión mínima queda formado
<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Hallar Camino Mínimo Entre Un Vértice y Todos los Demás En Un Grafo No Dirigido
<b>Entrada (Grafo Lista Adyacencia)</b>	1000 elementos de forma aleatoria y se ejecuta el método Dijkstra
<b>Salida</b>	Como tal el sistema no muestra un resultado, pero la distancia mínima de cada uno de los vértices es hallada y el heap fibonacci donde se guardan los valores es devuelto

<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Hallar Camino Mínimo Entre Un Vértice y Todos los Demás En Un Grafo Dirigido
<b>Entrada (Grafo Lista Adyacencia)</b>	1000 elementos de forma aleatoria especificando que el grafo es dirigido y se ejecuta el método Dijkstra
<b>Salida</b>	Como tal el sistema no muestra un resultado, pero la distancia mínima de cada uno de los vértices es hallada y el heap fibonacci donde se guardan los valores es devuelto

<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Hallar Camino Mínimo Entre Todos Los Vértices En Un Grafo
<b>Entrada (Grafo Lista Adyacencia)</b>	100 elementos de forma aleatoria y se ejecuta el método Floyd_Warshall
<b>Salida</b>	La matriz con los costos mínimos entre todos los vértices
<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Hallar Camino Mínimo Entre Todos Los Vértices En Un Grafo
<b>Entrada (Grafo Lista Adyacencia)</b>	100 elementos de forma aleatoria y se ejecuta el método Floyd_Warshall
<b>Salida</b>	La matriz con los costos mínimos entre todos los vértices

<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Hallar Camino Mínimo Entre Un Vértice y Todos los Demás En Un Grafo Dirigido Pero Con Pesos Negativos Pero Sin

	Ciclos Negativos
<b>Entrada (Grafo Lista Adyacencia)</b>	1000 elementos de forma aleatoria especificando que el grafo es dirigido y se ejecuta el método Bellman_Ford
<b>Salida</b>	Como tal el sistema no muestra un resultado, pero la distancia mínima de cada uno de los vértices es hallada y el heap binomial donde se guardan los valores es devuelto

<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Hallar Camino Mínimo Entre Un Vértice y Todos los Demás En Un Grafo Dirigido Pero Con Pesos Negativos Pero Con Ciclos Negativos
<b>Entrada (Grafo Lista Adyacencia)</b>	1000 elementos de forma aleatoria especificando que el grafo es dirigido y se ejecuta el método Bellman_Ford
<b>Salida</b>	False

<b>Caso de Uso</b>	Gestionar Elementos en un Grafo
<b>Caso de Prueba</b>	Saber Si Hay Camino Entre Todos Los Vértices En Un Grafo
<b>Entrada (Grafo Lista Adyacencia)</b>	100 elementos de forma aleatoria y se ejecuta el método CierreTransitivo
<b>Salida</b>	La matriz con los valores 1 y 0, 1 para decir que hay camino y 0 para cuando no hay



#### **4.6 Conclusiones parciales.**

El principal resultado de este capítulo es el modelo de diseño de la biblioteca que proporciona las bases o esquema para la implementación del sistema propuesto, donde los paquetes de diseño serán implementados por los paquetes de implementación. Se pudieron observar los patrones seguidos y de que forma fueron utilizados, así como el diagrama de componentes correspondiente al modelo de implementación mostrado.

## CONCLUSIONES

Con este trabajo se ha hecho un aporte a la informática tanto para Cuba como para el mundo para impulsar el desarrollo de aplicaciones en Software Libre. De esta forma, esta Biblioteca de Estructuras de Datos Avanzadas se suma a la informatización de nuestro país, propiciando la rapidez de respuesta a los servidores de bases de datos, en los softwares que necesiten de cálculos matemáticos complejos, accesos a información en disco duro, etc.

Luego de todo lo expuesto en este trabajo, es posible concluir que:

- Se hizo un detallado estudio de las diferentes estructuras de datos avanzadas a implementar, lo que propició un amplio conocimiento del tema a los desarrolladores.
- Se reflejaron las principales características de las herramientas y tecnologías a utilizar durante el desarrollo de la biblioteca
- Se logró un producto analizado, diseñado, implementado y probado utilizando la metodología de desarrollo RUP y Python como lenguaje de programación.
- Se demostró la eficiencia de estas estructuras de datos avanzadas, llegando algunas de ellas a insertar, buscar y eliminar miles de datos en pocos segundos según la capacidad de memoria RAM del ordenador donde se utilice.
- Se logró el objetivo general que se había planteado que consistió en desarrollar una biblioteca de estructuras de datos avanzadas en una plataforma de Software Libre y generar un documento que sirve de manual de ayuda a clientes o programadores donde se muestran de cada una de estas estructuras de datos avanzadas sus definiciones, ventajas, desventajas, aplicaciones y pseudocódigo. Este documento tiene el objetivo de orientar y ayudar a aquel programador o cliente que desee utilizarlas según sus propios intereses o que simplemente quiera desarrollarlas en otro lenguaje de programación.

## RECOMENDACIONES

Una vez concluido el trabajo de diploma y habiendo alcanzado el Objetivo General del mismo, se detectaron algunos aspectos que pudieran ser desarrollados y mejorados en función de buscar una mayor eficiencia en sus funcionalidades, además de otros que se consideran muy importantes para que la biblioteca pueda cumplir el objetivo con que fue creada. Por lo que se recomienda:

1. Experimentar el uso de otra metodología de desarrollo, preferiblemente ágil.
2. Implementar variantes de grafos utilizando la propuesta de matrices poco densas desarrollada.
3. Aumentar la cantidad de estructuras de datos implementadas.
4. Incrementar la cantidad de algoritmos y funcionalidades.
5. Poner la biblioteca en explotación en un proyecto real.
6. Estandarizar su uso en la Universidad.

## REFERENCIAS BIBLIOGRÁFICAS

1. TeleformaciónUCI Conferencias de Ingeniería de Software I. [Online]  
<http://teleformación.uci.cu>.
2. Introduction to Algorithms. [book auth.] Thomas H.Cormen, Charles E.Leirserson ,  
Ronald L.Rivest .
3. Algoritmia.net. [Online] <http://www.algoritmia.net/articles.php?id=18>.
4. Descartes Web Page. [Online]  
[descartes.cnice.mecd.es/materiales\\_didácticos/Calculo\\_matricial\\_d3/defmat.htm](http://descartes.cnice.mecd.es/materiales_didácticos/Calculo_matricial_d3/defmat.htm).
5. Programación, Departamento de Sistemas Informáticos . *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*. s.l. : Microsoft, 1999-2000 .
6. Sistemas. Jorge A. Saavedra Gutiérrez Ingeniería de Patrones Grasp (Craig Larman) Parte I .... [Online] [Cited: 5 2, 2008.]  
<http://jorgesaavedra.wordpress.com/2006/08/17/patrones-grasp-craig-larman/>.
7. Sergio Sánchez Rios .Ingeniero en Informática – Licenciado en Informática, Docente Jornada Parcial Universidad Viña del Mar. Metodologías de Análisis y Diseño Unidad VII, Diseño O.O – Diagramas de Interacción“Patrones de Diseño”. [Online] [Cited: 5 2, 2008.]  
[http://www.uvmsf.cl/~ssanchez/images/Metodologias/Unidad8\\_MAD.pdf](http://www.uvmsf.cl/~ssanchez/images/Metodologias/Unidad8_MAD.pdf).
8. *Computer Science Department at Stony Brook University's Web Site*. [Online] Mayo 10, 2005.  
<http://www.cs.sunysb.edu/~skiena/214/lectures/>.
9. La guía de formación online. [Online] 2007.  
[http://www.lectiva.net/curso-superior-en-programacion-uml-poo-y-patrones-de-diseno\\_34495.aspx](http://www.lectiva.net/curso-superior-en-programacion-uml-poo-y-patrones-de-diseno_34495.aspx).
10. Puddu, Susana. Sitio Web del Departamento de Matemáticas de la Universidad de Buenos Aires, Argentina. *Departamento de Matemáticas - Universidad de Buenos Aires*. [Online] Agosto 2006.  
[http://mate.dm.uba.ar/~spuddu/teo\\_de\\_grafos/](http://mate.dm.uba.ar/~spuddu/teo_de_grafos/).

11. Facultad de Ingeniería- Universidad de Buenos Aires. [Online] [Cited: diciembre 14, 2007.]  
<http://www.fi.uba.ar/materias/7504E/material/tda.pdf>.
  
12. Teleformación.uci.cu. [Online] [Cited: diciembre 14, 2007.]  
[http://teleformacion.uci.cu/file.php/57/Clases/Tema\\_III/Actividad\\_12\\_-\\_Conferencia\\_5\\_.pdf](http://teleformacion.uci.cu/file.php/57/Clases/Tema_III/Actividad_12_-_Conferencia_5_.pdf).
  
13. Wapedia. [Online]  
[http://wapedia.mobi/es/%C3%81rboL\\_binario](http://wapedia.mobi/es/%C3%81rboL_binario).
  
14. Prof. Juan Andrés Colmenares, M.Sc. Instituto de Cálculo Aplicado(Universidad de Zulia, Facultad de Ingeniería de Maracaibo, Venezuela). [Online]  
<http://www.ica.luz.ve/juancol/eda/busqueda/index.html>.
  
15. Bruno R. Preiss B.A.Sc., M.A.Sc., Ph.D., P.Eng. Department of Electrical and Computer Engineering, University of Waterloo. Data Structures and Algorithms with Object-Oriented Design Patterns in C++. Waterloo, Canada : s.n., 1997. Vol. 1.
  
16. Facultad de Ciencias Exacta, Ingeniería y Agrimensura y la Fundación Universidad Nacional de Rosario. [Online]  
<http://www.fceia.unr.edu.ar/estruc/2005/introduc.htm>.
  
17. CProgramming.com. [Online]  
<http://www.cprogramming.com/tutorial/computersciencetheory/twothree.html>.
  
18. WUrban(Wiki-Urban). [Online]  
[http://iuscivitas.homeunix.com/mediawiki/index.php?title=%C3%81rboL\\_rojo-negro](http://iuscivitas.homeunix.com/mediawiki/index.php?title=%C3%81rboL_rojo-negro).
  
19. Roberto Flórez Rueda, Profesor Asociado Facultad de Ingeniería , Universidad de Antioquia. Laboratorio Integrado de Sistemas,Universidad. [Online]  
[bochica.udea.edu.co/~rflorez/ed1/arboles/arboles03.html](http://bochica.udea.edu.co/~rflorez/ed1/arboles/arboles03.html).
  
20. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*.
  
21. Sahni, Sartaj. Data Structures, Algorithms, & Applications in Java: Pairing Heaps. [Online]

<http://www.cise.ufl.edu/~sahni/dsaa/enrich/c13/pairing.htm>.

22. Enríquez, Alfredo Campos. UDLA Universidad de las Américas Puebla. [Online] [Cited: marzo 16, 2008.]

<http://hosting.udlap.mx/profesores/miguela.mendez/alephzero/archivo/historico/az26/python.html>.

23. AtiWiki. [Online] [Cited: marzo 16, 2008.]

<http://150.185.75.30/atiwiki/index.php/PYTHON>.

24. Prado, Tomas Javier Robles. Introducción a PostgreSQL. [Online] [Cited: marzo 17, 2008.]

<http://users.servicios.retecal.es/tjavier/docfinal/out-htmls/x686.html>.

25. Altamirano, Ing Alfonso Valdez. Comparativo IDE's. [Online] [Cited: marzo 19, 2008.]

<http://www.obicuos.com>.

26. Tecnologías de la Información y la Comunicación (TIC's). [Online]

<http://consuelomblog.blogspot.com/2007/04/qu-son-las-tics.html>.

27. Chapman, Stephen. Free Computer Help. [Online]

<http://www.felgall.com/class1.htm>.

28. Visual Paradigm.com. [Online] [Cited: 03 20, 2008.]

<http://www.visualparadigm.com>.

29. Alberto Molpeceres. [Online]

<http://javahispano.org>.

30. Mtro. en C. Rolando Menchaca Méndez, Dr. Félix García Carballeira. Revista Digital Universitaria de México. [Online]

<http://www.revista.unam.mx/vol.1/num2/art4/>.

31. Universidad de Chile- Dpto de Ciencias de la Computación. [Online]

<http://www.dcc.uchile.cl/~lmateu/Java/Apuntes/java.htm>.

32. Zona Clic -Java. [Online]

<http://clic.xtec.es/es/jclic/java.htm>.

33. Algoritmos eficientes para Estructuras de Datos. Josh M. Belford Heirwood

34. O'connor, José Luis de la Fuente. Sistemas lineales de grandes dimensiones. Escuela Técnica Superior de Ingenieros Industriales Universidad Politécnica de Madrid. [Online]  
[http://dmaii.etsii.upm.es/~IngElec/downloads/Clase\\_dispersa\\_1\\_06.pdf](http://dmaii.etsii.upm.es/~IngElec/downloads/Clase_dispersa_1_06.pdf).
35. Eduardo A Arbones Malisani, Eduardo A. Arbones. Técnicas gráficas en producción. [Online]  
<http://books.google.com.cu/books?id=3ftqwGr0bAcC&pg=PA35&lpg=PA35&dq=arbol+general&source=web&ots=gHVqHJlt8Z&sig=AzJya6leg-x8Azv4LI5JkNdrwbQ&hl=es#PPA41,M1>.
36. Jiménez, Linda Mariana González Valdez y Marla Yasmín González. Estructura de datos II. [Online] [Cited: 4 2008, 1.]  
<http://members.tripod.com/fcc98/tutores/ed2/ed2.html>.
37. Wapedia. *Wiki: Arbol Binario de Búsqueda*. [Online]  
[http://wapedia.mobi/es/%C3%81rbol\\_binario\\_de\\_b%C3%BAsqueda](http://wapedia.mobi/es/%C3%81rbol_binario_de_b%C3%BAsqueda).
38. Wapedia. *Wiki: Arbol AVL*. [Online]  
[http://wapedia.mobi/es/%C3%81rbol\\_AVL](http://wapedia.mobi/es/%C3%81rbol_AVL).
39. Wapedia. *Wiki: Arbol Rojo Negro*. [Online]  
[http://wapedia.mobi/es/%C3%81rbol\\_rojo-negro#1..](http://wapedia.mobi/es/%C3%81rbol_rojo-negro#1..)
40. I.T.Industry. *Indexacion Arboles B Bases de Datos PHP*. [Online]  
[http://mmengineer.blogspot.com/2007\\_11\\_01\\_archive.html](http://mmengineer.blogspot.com/2007_11_01_archive.html).
41. Algoritmos y Estructuras de Datos. [book auth.] Niklaus Wirth.
42. Estructuras de Datos, Algoritmos y POO. [book auth.] Gregory Heileman.
43. Wapedia. *Wiki: Arbol Binario*. [Online]  
[http://wapedia.mobi/es/%C3%81rbol\\_binario](http://wapedia.mobi/es/%C3%81rbol_binario).
44. Departamento de Arquitectura de Computadores (DAC). Escola Politècnica Superior de Castelldefels (EPSC) (Campus de la Universitat Politècnica de Catalunya). [Online]  
[http://studies.ac.upc.edu/EPSC/LPII/documentos/Puesta\\_al\\_dia/06/MatrizDispersa.pdf](http://studies.ac.upc.edu/EPSC/LPII/documentos/Puesta_al_dia/06/MatrizDispersa.pdf).

45. Alan J. Pelaez, José I. Salas, German Larrazabal, Pablo Guillén. Análisis de Técnicas de Precondicionamiento para los Sistemas Lineales provenientes del simulador SEMIYA de INTEPEP.

[http://alfa.facyt.uc.edu.ve/~glarraza/re2002\\_04.pdf](http://alfa.facyt.uc.edu.ve/~glarraza/re2002_04.pdf).

46. Mauro Castillo Valdés. Politécnica Tecnológica Metropolitana Facultad de Ingeniería .  
[Online]

<http://informatica.utem.cl/~mcast/ESDATOS/TADS/Septiembre2005%20-%20Problemas.pdf>.



## GLOSARIO

### **Brecha Digital**

Brecha digital es una expresión que hace referencia a la diferencia socioeconómica entre aquellas comunidades que tienen Internet y aquellas que no, aunque también se puede referir a todas las nuevas tecnologías de la información y la comunicación (teléfonos móviles, cajeros automáticos, bippers, etc.). Como tal, la brecha digital se basa en diferencias previas al acceso a las tecnologías. Este término también hace referencia a las diferencias que hay entre grupos según su capacidad para utilizar las TIC (Tecnologías de la Información y la Comunicación) de forma eficaz, debido a los distintos niveles de alfabetización y capacidad tecnológica. También se utiliza en ocasiones para señalar las diferencias entre aquellos grupos que tienen acceso a contenidos digitales de calidad y aquellos que no.

### **Costo Amortizado**

Es el tiempo necesario para llevar a cabo una secuencia de operaciones de estructura de datos, es la media de todas las operaciones realizadas. Puede ser utilizado para demostrar que el coste medio de una operación es pequeño, si se promedia sobre las secuencias de operaciones, incluso si una sola operación puede costar bastante. Difiere del caso promedio en que la probabilidad no está involucrada; garantizando así el rendimiento medio de cada operación en el peor de los casos.