LECTURA 8.1
¿CÓMO DIBUJAR UNA ESTRUCTURA DE DATOS UTILIZANDO GRAPHVIZ Y SU LENGUAJE DOT?

¿Cómo dibujar una estructura de datos utilizando Graphviz?

Muchos estudiantes de la materia muestran especial interés en mostrar gráficamente las estructuras de datos en sus aplicaciones; sin embargo, este proceso demanda una cantidad considerable de tiempo y esfuerzo de programación que puede desviar la atención del objetivo fundamental del curso. Por esta razón, en esta sección se analiza una forma de lograrlo sin sacrificar tiempo dedicado en sí a la estructura por medio de la utilización de un software de libre distribución con este fin: Graphviz y el lenguaje *dot*.

Graphviz y el lenguaje *dot*

Graphviz es un software abierto de libre distribución para graficar, que presenta información estructural en forma de diagramas y puede aplicarse en diversas áreas como el análisis de redes, bioinformática, ingeniería de software, bases de datos, diseño de sitios web, aprendizaje por computadora y tiene interfaces gráficas para otros dominios.

Su modo de utilización se basa en el diseño de pequeños programas que toman descripciones de los diagramas de un lenguaje de texto simple y los dibuja en diversos formatos tales como archivos de imágenes, SVG, PDF ó para desplegarse en exploradores.

En la práctica, generalmente las gráficas se generan de fuentes de datos externas, (típicamente mediante el recorrido de la estructura de datos correspondiente) para crear un programa que pueda ser interpretado y ejecutado por alguno de los componentes de Graphviz y generar la imagen correspondiente.

Un componente de Graphviz: el lenguaje dot

El paquete Graphviz consiste de una variedad de programas para dibujar esquemas, de las cuales el lenguaje *dot* es la herramienta principal para dibujar estructuras jerárquicas

o de capas dirigidas que puede ejecutarse como un programa desde la línea de comandos, servicio de visualización web ó mediante una interfase gráfica compatible.

Una característica es que contiene diseños de algoritmos bien afinados para colocar los nodos, bordes, etiquetas, formas, etc. para dibujar estructuras de datos, diseños jerárquicos y un lenguaje subyacente para manejar herramientas gráficas orientadas a archivos.

Básicamente *dot* dibuja figuras con nodos dirigidos mediante líneas ó flechas. Puede leer archivos de texto que contienen las descripciones de las figuras y crea imágenes que pueden ser almacenadas en archivos gráficos con diversos formatos como JPG, GIF, PNG, SVG, PDF, PostScript, etc.

Sintaxis de un programa en lenguaje dot

La sintaxis del lenguaje *dot* tiene una gramática muy completa que permite manipular varios aspectos de la imagen generada, sin embargo, en esta sección se muestra la manera más simple de generar una imagen que muestre una estructura de datos. Para crear un programa en lenguaje *dot*, basta con escribir un archivo de texto (utilizando cualquier procesador de textos) y seguir los siguientes pasos:

1.- Definir el tipo y el nombre de la gráfica: En la primera línea se debe colocar el tipo y el nombre de la figura que se desea graficar. Se pueden nodos unidos mediante líneas ó flechas dirigidas. Si se desean dibujar nodos unidos mediante flechas se usa el tipo `digraph`, en cambio si las relaciones entre los nodos no requieren dirección, se usa un esquema de tipo `graph`.

2.- Colocar los nodos y sus relaciones: Después de definir el tipo y nombre de la figura, las siguientes líneas de código crean los nodos y sus relaciones mediante líneas ó flechas. Las definiciones de los nodos y sus relaciones se colocan entre los símbolos **{** y **}**. Un nodo se crea cuando su nombre aparece al principio de la línea y su relación con otro nodo se genera por medio del operador **–>** para relaciones dirigidas (con punta de flecha) ó **--** para relaciones no dirigidas (sin punta de flecha). Cuando se desea etiquetar cada relación con un valor, se coloca la sentencia `[label=`*`valor`*`]`. Cada definición de los nodos y sus relaciones debe terminar con el símbolo de puntuación del punto y coma (**;**).

La Fig. 8.22 muestra un pequeño programa en lenguaje *dot* que dibuja un ABB. Esta figura contiene un programa para generar un ABB llamado `Figura` cuyos nodos están unidos mediante flechas (`digraph`), donde la raíz es el nodo 60 y tiene como hijo izquierdo al nodo 40 y su hijo derecho es el nodo 90; a su vez el nodo 40 tiene como hijo izquierdo al 34 y como hijo derecho al 50.

```
digraph Figura {
   Raíz->60;
   60->40;
   60->90;
   40->34;
   40->50;
}
```

Fig. 8.22. Ejemplo de un programa en lenguaje dot para dibujar un ABB.

La Fig. 8.23 muestra la imagen generada con este programa.

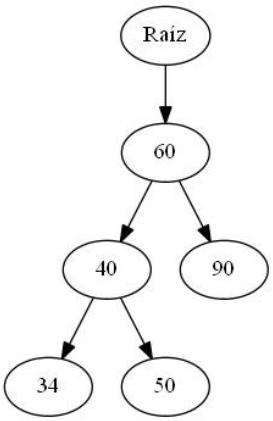

Fig. 8.23. Imagen generada con un programa en lenguaje dot.

¿Cómo ejecutar un programa en lenguaje dot?

Para ejecutar un programa en lenguaje *dot*, es necesario contar con el paquete Graphviz instalado, el cual puede descargarse de manera gratuita del siguiente sitio:

http://www.graphviz.org/Download.php

Un programa en lenguaje *dot* puede ejecutarse desde su editor gráfico (`GVEdit.exe`) ó en desde la línea de comandos; sin embargo, en esta sección se explica ésta última que nos atañe para nuestras necesidades.

Al invocar el interpretador del lenguaje (`dot.exe`) desde la línea de comandos, se le pueden proporcionar varios parámetros con funciones específicas. La Fig. 8.24 muestra los principales parámetros.

Parámetro	Función	Ejemplos

`-T`	Define el tipo de salida de la imagen de acuerdo a sus formatos permitidos, p. ejem. `jpg`, `bmp`, `gif`, `svg`, `png`, `ps`, etc.	`-Tjpg`
		`-Tbmp`
		`-Tgif`
		`-Tps`

`-O`	Genera automáticamente el archivo de salida basado en el nombre del archivo de entrada y el formato definido mediante el parámetro `-T`	

Fig. 8.24. Parámetros necesarios para generar un archivo con la imagen en lenguaje dot.

Una vez creado el programa, es necesario ejecutarlo con los parámetros correspondientes para generar la imagen deseada. P. ejem. suponiendo que el programa de la Fig. 8.22 se almacena en un archivo de texto llamado `Figura` (sin extensión) y desea generar la imagen correspondiente (Fig. 8.23) en un archivo con formato `JPG` llamado `Figura.jpg`, se escribe:

```
DOT -Tjpg -O Figura
```

donde se invoca el programa `dot.exe` y se establecen los parámetros indicados.

¿Cómo generar una imagen desde una aplicación de formas de Windows utilizando un programa en lenguaje dot?

Como se puede apreciar, es sumamente sencillo generar imágenes utilizando el lenguaje *dot* de Graphviz, basta con generar el programa en un archivo de texto con los nodos y sus relaciones y ejecutarlo para generar una imagen en un formato definido, la cual se puede desplegar en una forma de Windows mediante un *PictureBox*. Para ello, se agregan métodos a la clase del ABB de la aplicación de consola para que permita realizar estas operaciones.

Representación de las clases

Para crear el programa en lenguaje *dot*, ejecutarlo para generar un archivo con la imagen del ABB y mostrarla en una forma de Windows, se crea una nueva clase llamada `ClaseABB` que contiene los métodos `CrearArchivoDot()` y `RecorrerNodos()` y hereda los atributos, métodos y propiedades de la `ClaseArbolBinarioBusqueda` de la aplicación de consola resultando las clases mostradas en la Fig. 8.25.

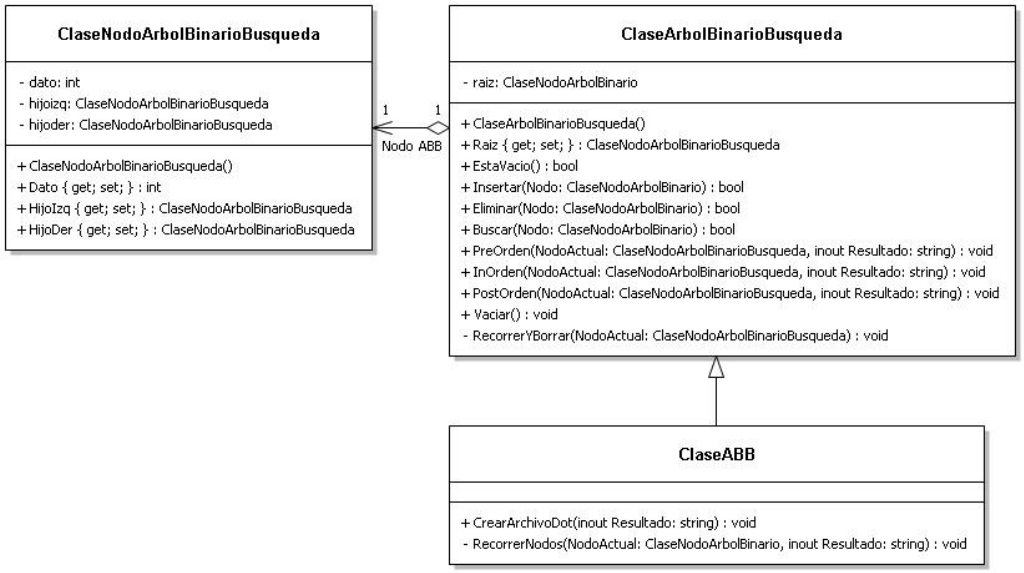

Fig. 8.25. La ClaseABB que permite dibujar utilizando el lenguaje dot.

En virtud de que la `ClaseArbolBinarioBusqueda` es igual que en la aplicación de consola y por lo tanto las operaciones son las mismas, centraremos la atención en el diseño de esta nueva `ClaseABB`.

La declaración de la *ClaseABB* en C# se almacena en un archivo de clase llamado *ClaseABB.cs* cuya codificación se muestra a continuación:

Prog. 8.2 a) Declaración de la clase del ABB (ClaseABB.cs).

```
class ClaseABB: ClaseArbolBinarioBusqueda // ClaseABB hereda de
ClaseArbolBinarioBusqueda
    {
        private void RecorrerNodos(ClaseNodoArbolBinarioBusqueda NodoActual, ref string
Resultado)
        {
            if (NodoActual != null)
            {
                if (NodoActual.HijoIzq != null)
                    Resultado = Resultado + "\n" + NodoActual.Dato.ToString() + "->" +
NodoActual.HijoIzq.Dato.ToString() + ";";

                if (NodoActual.HijoDer != null)
                    Resultado = Resultado + "\n" + NodoActual.Dato.ToString() + "->" +
NodoActual.HijoDer.Dato.ToString() + ";";

                RecorrerNodos(NodoActual.HijoIzq, ref Resultado); // Llamada recursiva
para recorrer el subárbol izquierdo

                RecorrerNodos(NodoActual.HijoDer, ref Resultado); // Llamada recursiva
para recorrer el subárbol derecho
            }
        }

        public void CrearArchivoDot(ref string Resultado)
        {
            if (!EstaVacio())  // Si no está vacío ...
            {
                Resultado = Resultado + "digraph Figura {";
                Resultado = Resultado + "\nRaíz->" + Raiz.Dato.ToString() + ";";
                RecorrerNodos(Raiz, ref Resultado);
                Resultado = Resultado + "\n}";
            }
        }
    }
```

Esta clase hereda de la `ClaseArbolBinarioBusqueda` y solamente contiene dos métodos: El método público `CrearArchivoDot()` que se encarga de llenar una cadena con el programa en lenguaje dot con los comandos necesarios para generar la imagen del ABB y el método privado `RecorrerNodos()` que realiza un recorrido PostOrden del ABB para generar las líneas del programa donde se muestra cada nodo y su relación con otro mediante el comando **->** (tal como el programa mostrado en la Fig. 8.22). Es importante destacar que el objeto `ArbolBinarioBusqueda` que controla el ABB utilizado en esta aplicación es de tipo `ClaseABB` (a diferencia de la aplicación de

consola en la que es de tipo `ClaseArbolBinarioBusqueda`) y se crea de la siguiente forma:

Prog. 8.2 b) Declaración del objeto con el ABB (Form1.cs).

```
        static ClaseABB ArbolBinarioBusqueda = new ClaseABB();
```

La forma mostrada en la Fig. 8.21 muestra un botón que sirve para dibujar un ABB. Cuando se oprime el *button6*, se verifica que el ABB no esté vacío y se procede a invocar el método `DibujarFigura()`. El código de este botón se muestra a continuación:

Prog. 8.2 c) Método para dibujar en pantalla el ABB (Form1.cs).

```
private void button6_Click(object sender, EventArgs e)
        {
            if (!ArbolBinarioBusqueda.EstaVacio())  // Si no está vacío ..
                DibujarFigura();  // Invoca el método para dibujar la figura
            else
                MessageBox.Show("Árbol Binario de Búsqueda vacío", "E R R O R",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
```

El código del método `DibujarFigura()` se muestra a continuación:

Prog. 8.2 d) Método para dibujar la figura del ABB (Form1.cs).

```
private void DibujarFigura()
        {
            //
================================================================================
            // Se invoca el método para crear el archivo Figura que contiene el
programa en lenguaje DOT
            // que crea el archivo Figura.jpg con la imagen del ABB
            string Resultado=CrearArchivoDot();

            //
================================================================================
            // Ejecución del proceso por lotes DIBUJAR.bat para crear la imagen
Figura.jpg
            InvocaDibujar();
```

```
            MessageBox.Show("Código para generar el archivo FIGURA.jpg con la
imagen:\n\n" + Resultado + "\n\nEjecutar mediante: DOT -TJPG -O \\Figura", "CREACIÓN
DEL ARCHIVO EN LENGUAJE DOT", MessageBoxButtons.OK, MessageBoxIcon.Information);
            //
============================================================================
            // Ejecución del método para crear una nueva forma y mostrar en ella la
Figura.jpg con la imagen del ABB
            CrearNuevaFormaConFigura();
        }
```

El proceso para dibujar una figura consta de tres partes:

1.- Crear el archivo con el programa en lenguaje *dot*: Esta parte se encarga de crear un archivo de texto llamado `\Figura` (el símbolo **** indica que se almacena en la carpeta raíz de la unidad de disco correspondiente) con el programa en lenguaje *dot* y la descripción del ABB que se desea graficar. Para ello, requiere una cadena denominada `Resultado` que almacena todas las líneas del código en lenguaje *dot*. Esta cadena se envía por referencia al método `CrearArchivoDot()` de la `ClaseABB` que es el encargado de recorrer los nodos del ABB y obtener sus relaciones con otros nodos, al finalizar, crea el archivo de texto `\Figura` y le almacena la cadena con el programa en *dot*. Enseguida se muestra el código:

Prog. 8.2 e) Método para crear el archivo en lenguaje dot (Form1.cs).

```
private string CrearArchivoDot()
        {
            // Recorrido del ABB para crear la línea con los comandos para el archivo
en lenguaje DOT
            string Resultado = "";
            ArbolBinarioBusqueda.CrearArchivoDot(ref Resultado);

            //
============================================================================

            // Creación del archivo \Figura que contiene el programa en lenguaje DOT)
            System.IO.StreamWriter AliasArchivo = new
System.IO.StreamWriter("\\Figura");
            AliasArchivo.WriteLine(Resultado);
            AliasArchivo.Close();

            return (Resultado);
        }
```

2.- Ejecutar el programa escrito en lenguaje *dot*: Una vez creado el programa `Figura`, se procede a ejecutarlo para generar el archivo con la imagen correspondiente. Para lograrlo, se ejecuta el código del procedimiento `InvocaDibujar()` que simplemente ejecuta un programa de procesamiento por lotes (*batch*) llamado `Dibujar.bat`, el cual contiene los comandos y parámetros necesarios para ejecutar un programa en lenguaje *dot*. El código de este procedimiento se muestra a continuación:

Prog. 8.2 f) Método para ejecutar el archivo en lenguaje dot (Form1.cs).

```
private void InvocaDibujar()
        {
            // El archivo por lotes DIBUJAR.bat contiene el siguiente código:
            //    @echo off
            //    del \Figura.jpg
            //    \"Archivos de programa\Graphviz 2.28"\bin\DOT -Tjpg -O \Figura

            // Ejecuta DIBUJAR.bat ubicado en la carpeta raíz de la unidad de disco
            System.Diagnostics.Process.Start("\\DIBUJAR.bat");
        }
```

Para facilitar la ejecución del programa `Figura` (creado previamente), se crea el archivo de texto `Dibujar.bat` y se almacena en la carpeta raíz de la unidad de disco correspondiente. Este batch contiene el siguiente código:

Prog. 8.2 g) Archivo de procesamiento por lotes (Dibujar.bat).

```
@echo off
del \Figura.jpg
\"Archivos de programa\Graphviz 2.28"\bin\DOT -Tjpg -O \Figura
```

La línea `@echo off` es una sentencia de este tipo de programas que evita desplegar en pantalla los comandos que está ejecutando.

Al ejecutar este programa de procesamiento por lotes, primero se elimina el archivo `\Figura.jpg` (ubicado en la carpeta raíz) para evitar desplegar la imagen de un archivo generado previamente, después se invoca la ejecución del programa `Figura`, a través del interpretador de comandos de programas *dot* (`dot.exe`) ubicado en la carpeta `\"Archivos de programa\Graphviz 2.28"\bin\` y la colocación los parámetros correspondientes. Es importante mencionar que al momento de escribir

esta sección, la versión 2.28 es la más actualizada y está instalada en dicha carpeta. Asegúrese identificar la ruta de directorios ó carpeta de localización del programa `dot.exe` al momento de desarrollar su aplicación.

Con esto, el programa `Dibujar.bat` toma como entrada el archivo `Figura` (en lenguaje *dot*) y genera el archivo `Figura.jpg` con la imagen del ABB.

3.- Mostrar la imagen generada en una forma: Después de haber generado la imagen del ABB en el archivo `Figura.jpg`, se procede a mostrarla en una forma de Windows. Para ello se invoca el procedimiento `CrearNuevaFormaConFigura()`, cuyo código se muestra enseguida:

Prog. 8.2 h) Método para mostrar la figura generada (Form1.cs).

```
private void CrearNuevaFormaConFigura()
        {
            //
============================================================================
            // Creación de la nueva forma
            Form frmNuevaVentana = new Form();  // Creación de una nueva forma para
dibujar el ABB
            frmNuevaVentana.Width = 1000;  // Define el ancho de la nueva forma
            frmNuevaVentana.Height = 600;  // Define la altura de la nueva forma
            frmNuevaVentana.AutoScroll = true;  // Activa el modo AutoScroll
            frmNuevaVentana.Text = "F I G U R A";  // Texto de la nueva forma
            frmNuevaVentana.StartPosition = FormStartPosition.CenterScreen;  // Define
que la nueva forma sea centrada
            frmNuevaVentana.Show();  // Activa la nueva forma

            //
============================================================================
            // Creación de un pictureBox en la nueva forma
            PictureBox picABB = new PictureBox();  // Crea un pictureBox
            picABB.Parent = frmNuevaVentana;  // Establece que el pictureBox está
dentro de la nueva forma
            picABB.Width = 990;  // Define el ancho del pictureBox
            picABB.Height = 565;  // Define la altura del pictureBox
            picABB.SizeMode = PictureBoxSizeMode.Zoom; // Define el modo del tamaño del
pictureBox con ajuste automático

            //
============================================================================
            // Abre el archivo Figura.jpg en modo de sólo lectura
            System.IO.FileStream AliasFigura;  // Declaración del alias del archivo
Figura.jpg

            try  // Intenta abrir el archivo
            {
                AliasFigura = new System.IO.FileStream("\\Figura.jpg",
System.IO.FileMode.Open, System.IO.FileAccess.Read);
            }
            catch (Exception x)  // En caso de error ...
```

```
            {
                MessageBox.Show("No se pudo abrir el archivo \\FIGURA.jpg", "E R R O
R", MessageBoxButtons.OK, MessageBoxIcon.Error);
                frmNuevaVentana.Close();  // Cierra la nueva forma
                return;
            }

            //
============================================================================
            // Carga la imagen del ArbolBinarioBusqueda.jpg en el pictureBox

            try  // Intenta cargar la imagen en el pictureBox
            {
                picABB.Image = System.Drawing.Bitmap.FromStream(AliasFigura);
            }
            catch (Exception x)  // En caso de error ...
            {
                MessageBox.Show("No se pudo cargar la imagen del archivo \\FIGURA.jpg",
"E R R O R", MessageBoxButtons.OK, MessageBoxIcon.Error);
                AliasFigura.Close();  // Cierra el archivo ArbolBinarioBusqueda.jpg
                frmNuevaVentana.Close();  // Cierra la nueva forma
                return;
            }

            AliasFigura.Close();  // Cierra el archivo ArbolBinarioBusqueda.jpg
            //
============================================================================

            picABB.Refresh(); // Refresca la imagen
        }
```

Este procedimiento genera una nueva forma y dentro de ella un *pictureBox* que servirá para desplegar la imagen del ABB. Una vez hecho esto, intenta abrir el archivo `Figura.jpg` (en modo sólo lectura), cargar la imagen que contiene en el *pictureBox* y cerrar el archivo. La ejecución de estas operaciones se verifica a través de sentencias *try-catch* para evitar la posibilidad de errores como la inexistencia del archivo durante la apertura ó la imposibilidad de cargar su imagen.

Al oprimir el *button6* (Dibujar) y ejecutar el código mencionado con anterioridad, primero se despliega un mensaje con los detalles del programa en lenguaje *dot* para generar el archivo `Figura.jpg` con la imagen del ABB (Fig. 8.26).

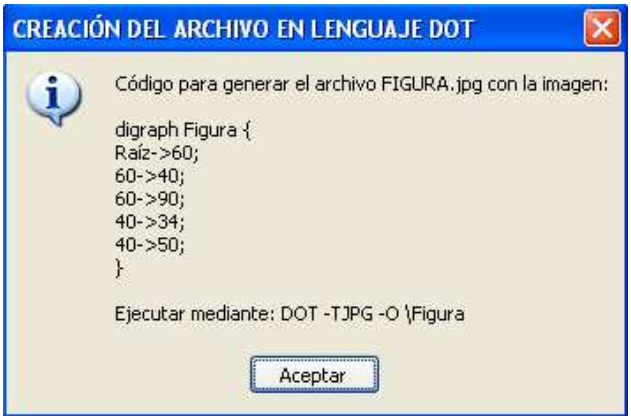

Fig. 8.26. Mensaje de creación del programa dot para generar la Figura.jpg

Después aparece una nueva ventana con una forma de Windows que despliega la imagen del ABB (Fig. 8.27).

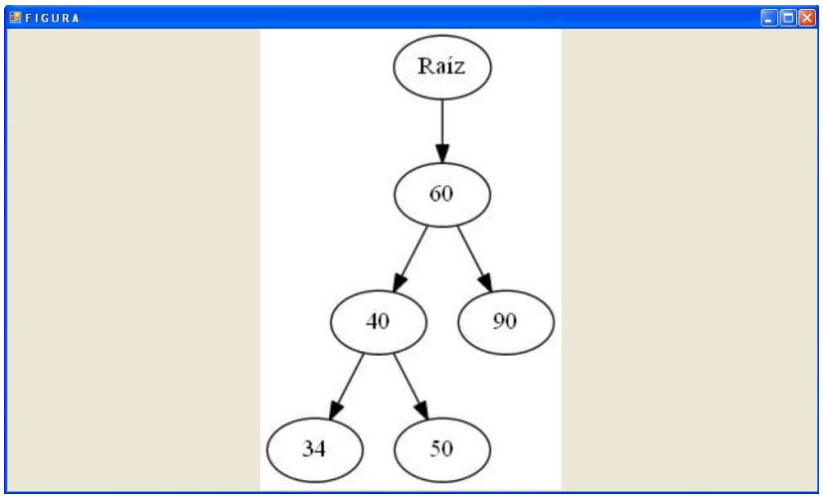

Fig. 8.27. Nueva ventana con la forma que contiene la imagen del ABB en el archivo Figura.jpg