PROGRAMACIÓN III

TALLER SEMANA 13

Integrantes:
Eduardo Andrade
Martin Jimenez
Manosalvas Roberto
Soledispa Ashlee

PROFESOR: PhD (c). Luis Fernando Aguas Bucheli

QUITO – ECUADOR

DESARROLLO

Explicación de cambios realizados

- **Clase Nodo**

```
public class Nodo {  21 usages
    int x, y;  3 usages
    String etiqueta;  17 usages
   Nodo izquierda1, izquierda2, centro, derecha1, derecha2;  14 usages

    public Nodo(int x, int y, String etiqueta) {  1 usage
        this.x = x;
        this.y = y;
        this.etiqueta = etiqueta;
        this.izquierda1 = null;
        this.izquierda2 = null;
        this.centro = null;
        this.derecha1 = null;
        this.derecha2 = null;
    }
```

Explicación: En principio teníamos únicamente 2 nodos declarados en esta clase, sin embargo, para lograr nuestro árbol quintinario, aumentamos 3 nodos más, actualizamos el constructor y mantenemos en valor “null”, la etiqueta y valores x, y se mantienen igual.

- **Clase Árbol: Función anadirNodo**

```
public void anadirNodo(Nodo nodo, Nodo padre, String posicion) {  2 usages
    if (padre == null) {
        if (raiz == null) {
            raiz= nodo;
        } else {
            throw new IllegalArgumentException("La raiz ya existe");
        }
    } else {
        if (posicion.equalsIgnoreCase( anotherString: "izquierda1")) {
            if (padre.izquierda1 == null) {
                padre.izquierda1 = nodo;
            } else {
                throw new IllegalArgumentException("Hoja Izq ya existe");
            }
        } else if (posicion.equalsIgnoreCase( anotherString: "izquierda2")) {
            if (padre.izquierda2 == null) {
                padre.izquierda2 = nodo;
            } else {
                throw new IllegalArgumentException("Hoja Izq ya existe");
            }
        } else if (posicion.equalsIgnoreCase( anotherString: "centro")) {
        if (padre.centro == null) {
            padre.centro = nodo;
```

```
        if (padre.centro == null) {
            padre.centro = nodo;
        } else {
            throw new IllegalArgumentException("Hoja Cen ya existe");
        }
        } else if (posicion.equalsIgnoreCase( anotherString: "derecha1")) {
            if (padre.derecha1 == null) {
                padre.derecha1 = nodo;
            } else {
                throw new IllegalArgumentException("Hoja Der ya existe");
            }
        } else if (posicion.equalsIgnoreCase( anotherString: "derecha2")) {
            if (padre.derecha2 == null) {
                padre.derecha2 = nodo;
            } else {
                throw new IllegalArgumentException("Hoja Der ya existe");
            }
        } else {
            throw new IllegalArgumentException("Posición inválida: " + posicion);
        }

    }
    nodos.add(nodo);
}
```

Explicación: La función ‘anadirNodo’ fue modificada para adaptarse a la estructura que cambiamos en la clase Nodo, permitiendo añadir nodos hijos en cinco posiciones específicas: izquierda1, izquierda2, centro, derecha1 y derecha2.

Para cada posición, se verifica si el espacio correspondiente en el nodo padre está vacío (null); de lo contrario, se lanza una excepción con un mensaje específico indicando que la hoja ya existe.

Además, se añadió una validación que lanza una excepción si la posición proporcionada no coincide con las opciones válidas. La lógica para añadir un nodo como raíz permanece intacta, asegurando que solo se establece si aún no existe una raíz.

Finalmente, cada nodo añadido se registra en la colección nodos para mantener la estructura global.

- **Clase ArbolGrafico : Función dibujarArbol**

```
public void dibujarArbol(Graphics2D g2d, Nodo nodo, int x, int y, int dimensionX, int dimensionY) {  7 usages
    if (nodo != null) {
        // Dibuja el nodo actual
        g2d.fillOval( x: x - 15,  y: y - 15,  width: 30,  height: 30);
        g2d.drawString(nodo.etiqueta,  x: x - 10,  y: y + 5);

        // Asigna las coordenadas del nodo actual
        nodo.x = x;
        nodo.y = y;

        // Dibujar líneas y nodos hijos - Las hojas esas
        int offset = dimensionX / 2; // Espaciado

        // Nodo izquierda1
        if (nodo.izquierda1 != null) {
            g2d.drawLine(x, y,  x2: x - dimensionX,  y2: y + dimensionY);
            dibujarArbol(g2d, nodo.izquierda1,  x: x - dimensionX,  y: y + dimensionY, offset, dimensionY);
        }

        // Nodo izquierda2
        if (nodo.izquierda2 != null) {
            g2d.drawLine(x, y,  x2: x - offset,  y2: y + dimensionY);
            dibujarArbol(g2d, nodo.izquierda2,  x: x - offset,  y: y + dimensionY,  dimensionX: offset / 2, dimensionY);
        }
```

```
        // Nodo centro
        if (nodo.centro != null) {
            g2d.drawLine(x, y, x,  y2: y + dimensionY);
            dibujarArbol(g2d, nodo.centro, x,  y: y + dimensionY, offset, dimensionY);
        }

        // Nodo derecha1
        if (nodo.derecha1 != null) {
            g2d.drawLine(x, y,  x2: x + offset,  y2: y + dimensionY);
            dibujarArbol(g2d, nodo.derecha1,  x: x + offset,  y: y + dimensionY,  dimensionX: offset / 2, dimensionY);
        }

        // Nodo derecha2
        if (nodo.derecha2 != null) {
            g2d.drawLine(x, y,  x2: x + dimensionX,  y2: y + dimensionY);
            dibujarArbol(g2d, nodo.derecha2,  x: x + dimensionX,  y: y + dimensionY, offset, dimensionY);
        }
    }
}
}
```

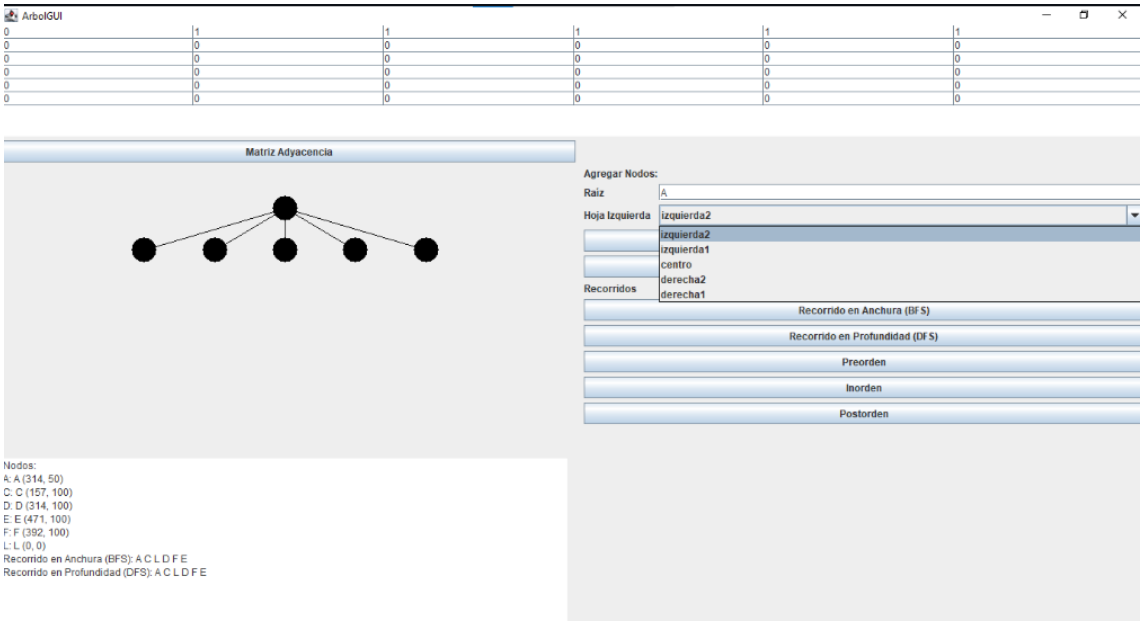
Explicación: La función ‘dibujarArbol’ fue modificada para reflejar la nueva estructura de la clase Nodo, permitiendo visualizar las cinco posiciones posibles de nodos hijos (izquierda1, izquierda2, centro, derecha1, derecha2). Para cada hijo, se añadió lógica que dibuja una línea desde el nodo actual hacia la posición correspondiente, utilizando coordenadas calculadas dinámicamente en función de los parámetros dimensionX y dimensionY, ajustando el espaciado horizontal (offset) para mantener una distribución visual coherente.

Además, para cada hijo existente, se realiza una llamada recursiva a la función, pasando las nuevas coordenadas y ajustando el espaciado horizontal según la posición del nodo.

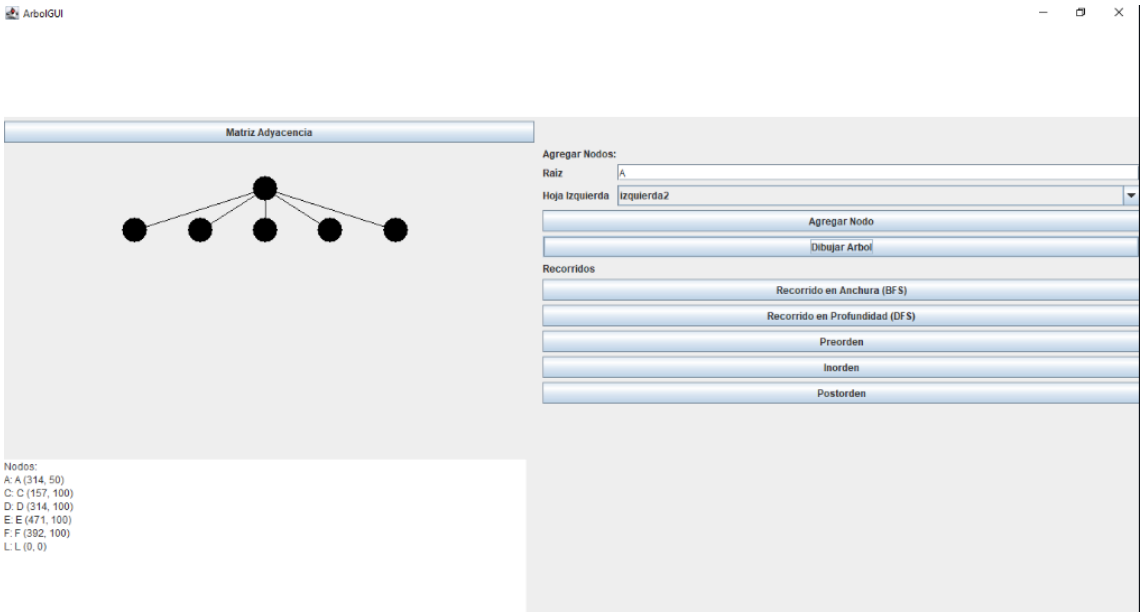
Finalmente modificamos el combobox del GUI Form para poder añadir cada nodo, colocando 5 elementos en el combo box, los cuales corresponden a los elementos definidos en la case nodo, y así ir dibujando cada uno de estos nodos al seleccionarlo desde el combo box.

Resultados

Interfaz Gráfica



Dibujo del árbol quintinario



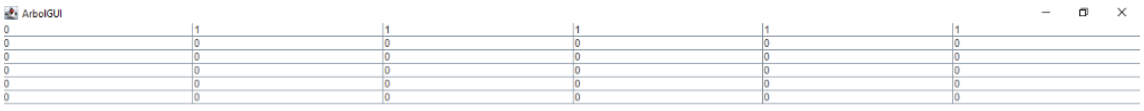
BFS

Nodos:
A: A (314, 50)
C: C (157, 100)
D: D (314, 100)
E: E (471, 100)
F: F (392, 100)
L: L (0, 0)
Recorrido en Anchura (BFS): A C L D F E

DFS

Nodos:
A: A (314, 50)
C: C (157, 100)
D: D (314, 100)
E: E (471, 100)
F: F (392, 100)
L: L (0, 0)
Recorrido en Anchura (BFS): A C L D F E
Recorrido en Profundidad (DFS): A C L D F E

Matriz de adyacencia



CONCLUSIONES

- Se amplió la clase Nodo para soportar un árbol quintinario, permitiendo que cada nodo tenga hasta cinco hijos definidos como izquierda1, izquierda2, centro, derecha1 y derecha2. Esto incrementó significativamente la flexibilidad del modelo, habilitándolo para representar la estructura solicitada
- La función de dibujo del árbol (dibujarArbol) se modificó para reflejar gráficamente la estructura quintinaria. Esto incluyó cálculos precisos de coordenadas y espaciado para distribuir de manera lógica y visualmente clara los nodos en el espacio.
- Se actualizó el combobox en el formulario GUI para incluir las cinco posiciones de hijos definidas en la clase Nodo, facilitando la adición de nodos al árbol. Este cambio no solo mejoró la funcionalidad del programa, sino que también lo hizo más intuitivo y adaptable para gestionar la estructura quintinaria directamente desde la interfaz gráfica.

BIBLIOGRAFÍA

El repositorio base de donde se partió para realizar el taller fue:
https://github.com/Aguaszoft/UDLA-GRAFOS-GRAFICO-ANCHURA-PROFUNDIDAD.git