UNIVERSIDAD NACIONAL DE GENERAL SARMIENTO

LICENCIATURA EN SISTEMAS

PROGRAMACIÓN III

TRABAJO PRÁCTICO III: CLIQUES GOLOSAS

PROFESORES:

BAGNES, PATRICIA

SOTELO, IGNACIO

ALUMNA:

GOMEZ, MELINA BELEN

2024

INTRODUCCIÓN

Este informe detalla la creación de una aplicación que aborda el conocido problema de encontrar una clique máxima en un grafo utilizando un algoritmo goloso. Una clique en un grafo es un conjunto de vértices donde cada par de vértices en dicho conjunto está conectado mediante una artista. El programa permite al usuario crear un grafo desde cero, indicando cantidad de aristas, pesos y las conexiones que desea que tenga. Luego de hacer el cálculo mencionado mostrará el resultado en pantalla para una mejor interpretación del problema.

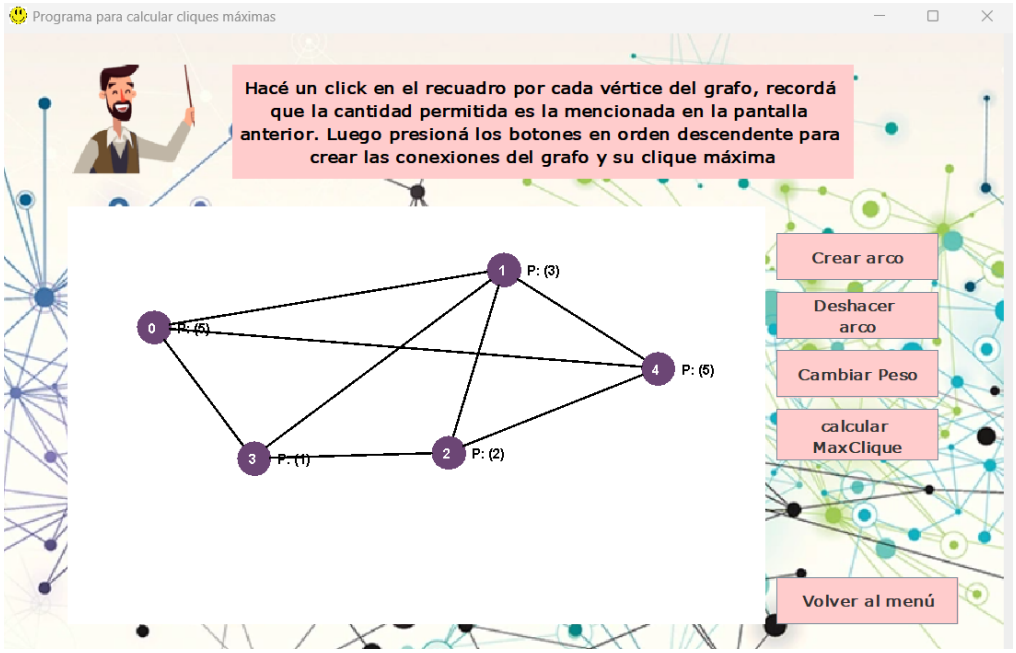
A continuación se detallan las clases implementadas en la aplicación, resaltando la separación de interfaz visual e interfaz lógica entre otros.

IMPLEMENTACIÓN

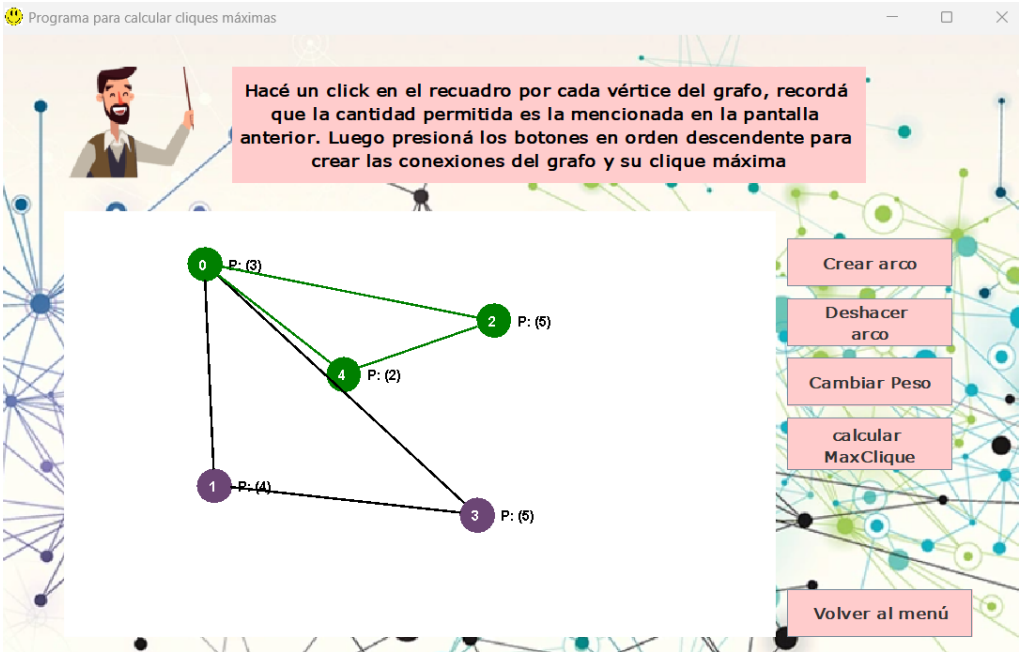
Interfaz Visual:

La aplicación presenta una interfaz visual agradable e interactiva. Comienza con una ventana de bienvenida que solicita el primer dato necesario para acceder a la ventana principal. En esta, los gráficos cambian manteniendo un equilibrio de colores para evitar ruidos visuales.

Luego se accede a la ventana principal en la que hay una guía para el usuario para agregar vértices y su peso mediante clics en el panel. Además, ofrece opciones para manipular el grafo antes de calcular su clique máxima o regresar al menú anterior.

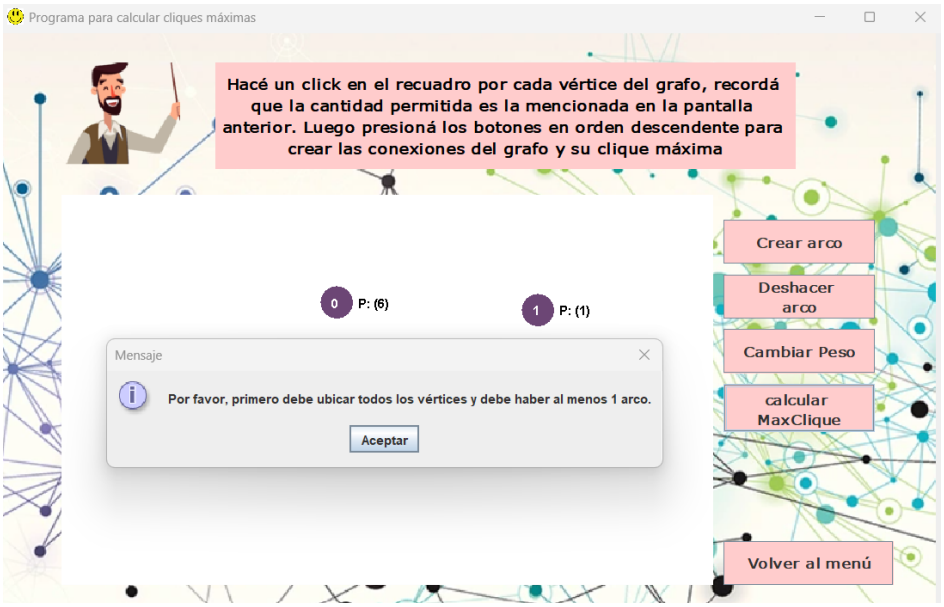


Cuando el usuario presiona el botón calcularMaxClique, además de mostrar la clique máxima en pantalla, esta se pinta de un color acorde al entorno de la interfaz para mantener la armonía visual.



En la clase ventanaPrincipal, al igual que el resto de las clases se asegura la correcta implementación de código mediante clean code según lo visto en clase, de esta manera tenemos variables de corto alcance, métodos públicos primeros según el orden declarado, espacios entre bloques, entre otros.

En la actual pantalla también se implementaron cuadros de “errores” para que, en caso de detectar información no válida especificada en el código, se pueda mostrar dicha información al usuario y éste entienda que es lo que estaría faltando o que dato erróneo brindado no permite que el algoritmo funcione de la manera que está aplicada.



Si evaluamos el código de la pantalla principal, podremos encontrar también clases anónimas en cada evento correspondiente a los botones. Recordamos que las mismas se instancian e inmediatamente se abren llaves que nos informarán el comportamiento. Si bien no se implementó de manera manual el código de acción del evento, es bueno recalcar el conocimiento aprendido cuando se puede visualizar.

```
public void actionPerformed(ActionEvent e) {

    if (verticesMarcados == cantVertices && algoritmo.cantidadArcos() > 0) {
        cliqueMaxima = new HashSet<>();
        cliqueMaxima = algoritmo.resolverCliqueMaxima();
        panel.repaint();

    } else {
        JOptionPane.showMessageDialog(null,
                "Por favor, primero debe ubicar todos los vertices y debe haber al menos 1 arco");
    }

}
```

Interfaz Lógica:

La lógica de la aplicación está estructurada de manera que las acciones generales están separadas de las específicas del grafo y del algoritmo goloso para calcular la clique máxima.

La clase Grafo ofrece acceso constante a la cantidad de arcos y almacena los vecinos del grafo en un ArrayList para un acceso rápido. Permite agregar y borrar arcos, asignar pesos a los vértices y realizar diversas operaciones sobre el grafo. Se destaca por su capacidad para manejar eficientemente la estructura y las relaciones entre los nodos del grafo.

La clase MaxClique contiene el algoritmo goloso que resuelve el problema de la clique máxima, básicamente el motor de este trabajo práctico. Este algoritmo selecciona vértices basándose en su peso y verifica si al agregar un vértice a la clique actual, esta se hace más grande. Si es así, el vértice se añade a la clique y el proceso continúa con el siguiente vértice de mayor peso.

```
public HashSet<Integer> resolverCliqueMaxima() {
    HashSet<Integer> hashADevolver = new HashSet<>();

    for (int vertice : ordenarPesosVertices()) {

        int posicion = grafo.obtenerPosicionPorPeso(vertice);
        verticesVisitados.add(posicion);

        HashSet<Integer> vecinos = grafo.getVecinos(posicion);
        if (esClique(vecinos)&& grafo.getVecinos(posicion).size() > 1) {
            conjuntoClique.addAll(vecinos);
            hashADevolver.addAll(vecinos);

            return hashADevolver;
        }
    }
    throw new IllegalArgumentException("No es posible calcular una clique maxima");
}
```

A continuación, se presenta el método esClique(), que utiliza Streams y Lambdas aprendidos en clases de Tecnología Java para verificar si un conjunto de vértices forma una clique:

```
public boolean esClique(HashSet<Integer> vertices) {
    return vertices.stream().allMatch(vertice1 -> vertices.stream()
            .allMatch(vertice2 -> vertice1.equals(vertice2) || grafo.existeArco(vertice1, vertice2)));
}
```

También se implementó el método ordenarPesosVertices(), el cual ordena el ArrayList de pesos de los vértices utilizando el método sort para colecciones, y luego invierte el orden para obtener una lista de pesos de mayor a menor. Esta implementación se aprovechó mediante la observación de videos relacionados con la materia, lo que facilitó la lógica necesaria para obtener este resultado.

```
public ArrayList<Integer> ordenarPesosVertices() {
    ArrayList<Integer> ret = grafo.getPesosVertices();
    Collections.sort(ret, Collections.reverseOrder());
    return ret;
}
```

Test:

Se crearon las clases de prueba GrafoTest y MaxCliqueTest, en las que se implementaron tests para cada método, incluyendo casos positivos (happy paths) y casos negativos (assertFalse cases). Luego, la clase LogicaTest incluye pruebas para las excepciones y errores posibles en las clases Grafo y MaxClique. Éstos fueron útiles para encontrar errores en tiempo de ejecución, algunos de ellos sirvieron para comprender errores que no se tenían en cuenta al momento de ejecutar los métodos llamados. Se separó mediante líneas de comentarios las pruebas de cada clase para mantener un orden adecuado.

```
public class logicaTest {

    //--------- Excepciones en MaxCliqueTest ---------


    public void cliqueMaximaFallaTest() {
        Grafo grafo = new Grafo(3);
        MaxClique cliqueTest= new MaxClique(grafo);
                assertThrows(IllegalArgumentException.class, () ->
                {
                    cliqueTest.resolverCliqueMaxima();
                });
    }



    //--------- Excepciones en Grafo ---------

    @Test
    public void verticeNegativoTest()
    {
```

CONCLUSIÓN

En este proyecto, se desarrolló una aplicación capaz de encontrar una clique máxima en un grafo mediante un algoritmo goloso. La separación clara entre la interfaz visual y la lógica del programa facilitó el desarrollo y mantenimiento del código. El uso de tecnologías modernas como Streams y Lambdas en Java demostró ser eficiente para el procesamiento de grafos y los tests aseguraron la confiabilidad del programa implementado.

Personalmente, poder implementar en el programa lo aprendido en las clases fue una experiencia enriquecedora, especialmente trabajar con los colores y el aspecto visual del proyecto. Esta experiencia me hizo darme cuenta de que disfruto mucho del diseño y la personalización de interfaces gráficas. En cuanto a la lógica, crear el algoritmo me ayudó a entender mejor los conceptos de grafos y algoritmos golosos. En conclusión, la aplicación no solo cumple con los objetivos propuestos, sino que también me permitió aplicar y consolidar mis conocimientos de una manera práctica y efectiva.