Benemérita Universidad Autónoma de Puebla

Facultad de ciencias de la computación

FRAMEWORK EDUCATIVO PARA LA MODULACIÓN EN LA RESOLUCIÓN LA APLICACIÓN DE ALGORITMOS CON GRAFOS

Tesis
Presentada para obtener el título de:

LICENCIATURA INGENIERÍA EN CIENCIAS DE LACOMPUTACIÓN

Presenta:

Efraín de Jesús Lima Estrada

Asesor:

M.C. Mariano Larios Gómez

Puebla, Pue. Noviembre 2022

Dedicatoria

El camino hasta aquí fue largo, como un túnel en donde vi a varios de mis amigos y compañeros hacerse a un lado porque perdían el interés, y al recordar todo lo que paso muy fácilmente pude haber sido como ellos y apartarme, sin embargo, en mi vida hubo personas que me ofrecieron su apoyo de forma incondicional, ellos fueron mi luz en el túnel oscuro, ya que iluminaron mi camino, cabe aclarar que cuando inicie no tenía idea del trayecto a seguir, pero ellos supieron guiarme y si me refiero a mi familia, amigos y conocidos, sin su ayuda y aportes a mi vida yo no hubiera llegado hasta este punto, ya que en dicha travesía existieron impedimentos y también obstáculos que ellos me ayudaron a superar.

Por lo anterior agradezco a dios por poner en mi camino a esas personas porque sin su apoyo no hubiera llegado tan lejos, esta tesis la dedico a mi familia que con tanto esfuerzo supieron guiarme.

Agradecimientos

Empezaré por agradecer el apoyo de mis padres que hicieron todo lo posible para que mis hermanos y yo tuviéramos estudio, en el caso de mi padre que tuvo que pasar largas jornadas de trabajo en donde sé que no era feliz y sin embargo, soporto los malos tratos de su trabajo con tal y su familia tuviera algo que comer y dinero para mandar a sus hijos a estudiar porque él siempre tuvo la idea de hacer progresar a sus hijos para que tuvieran mejores oportunidades a las que él nunca tuvo acceso, además agradezco el apoyo que como padre me brindo cuando lo necesitaba así como su ejemplo de un hombre trabajador el cual aprendí bien de él y también de sus valores que me transmitió por medio de su ejemplo para no ser una mala persona para la sociedad. En el caso de mi madre agradezco que siempre estuviera junto a mí, enseñándome sus valores, no permitiendo que me desviara del camino, agradezco que cuando sufrí de enfermedades ella siempre estuvo a mi lado velando por mi recuperación, desvelándose y preocupándose por mi bienestar, además debo agradecer que ella fue un modelo de resistencia para mí, nunca se doblegó por más fuerte que fuera la situación para ella o cuan dolorosa fuera, siempre tuvo como mira el éxito de sus hijos aunque eso significara olvidarse de ella, también reconozco que a pesar de que ella siempre sufrió por personas externas toda su vida nunca deseo el mal a nadie.

Agradezco a mis hermanos por el apoyo monetario y emocional que me brindaron en mi etapa infantil, a mi hermana que siempre tuvo palabra de ánimo para mí y que incluso compartía conmigo una parte de su salario para que tuviera la oportunidad de comprarme algo, además agradezco su apoyo como hermana; en cuanto a mi hermano le agradezco su ayuda que me brindo pagando algunas de mis colegiaturas además de sus apoyos que me brindo con tal y saliera adelante así como su apoyo económico que me daba, también agradezco los viajes que organizo él con tal y la familia se despejara de su rutina; A los dos agradezco de tenerlos como mis hermanos y crecer junto a ellos.

Agradezco a los profesores que siempre creyeron en mí específicamente a 3, al profesor Mariano Larios Gómez que me dio la oportunidad de ser su asesorado y ayudarme a realizar esta tesis, al profesor Luis Osvaldo Juárez Cázares quien en un principio creyó en mis habilidades y la última vez que me vio me comento que llegaría tan lejos como yo quisiese y finalmente al profesor Pablo Almazán López, quien me apoyo e ingreso en esta rama de las ciencias; también agradezco a todos mis conocidos que también aportaron a mi vida.

Finalmente, quiero agradecer a un joven quien desde el inicio de su vida tuvo complicaciones y siempre ha luchado por sus sueños, a dicho joven quien siempre vio a otros gozar de su vida y él se centró en estudiar con la esperanza de que el futuro llegara con mejores oportunidades, a él le agradezco su dedicación y aguante durante estos años.

Muchas gracias a todos por su dedicación y esfuerzo para conmigo.

Índice General

DEDICATORIA 2

AGRADECIMIENTOS 3

ÍNDICE GENERAL 4

ÍNDICE DE ILUSTRACIONES 7

RESUMEN 9

CAPÍTULO 1 INTRODUCCIÓN 10

1.1. ANTECEDENTES DEL PROYECTO 10
1.2. OBJETIVOS GENERALES Y ESPECÍFICOS DEL PROYECTO 10
1.3. METODOLOGÍA 11

CAPÍTULO 2 ESTADO DEL ARTE 12

2.1. RECURSOS RELACIONADOS 12
2.2. INVESTIGACIÓN SOBRE LA HISTORIA DE LOS GRAFOS 12
2.3. ALGORITMOS RELACIONADOS AL ESTUDIO DE GRAFOS 13
2.4. CONCEPTOS RELACIONADOS A MODULARIDAD 14
2.5. PROGRAMACIÓN ORIENTADA A OBJETOS 14

CAPÍTULO 3: ANÁLISIS Y DISEÑO 15

3.1. ETAPA DE INVESTIGACIÓN 15
3.2. ANÁLISIS DE LOS ALGORITMOS 15
3.2.1. GRAFOS DIRIGIDOS 15
3.2.1.1. Dijkstra 15
3.2.1.2. Floyd 16
3.2.1.3. Warshall 16
3.2.1.4. BPF 17
3.2.2. GRAFOS NO DIRIGIDOS 18
3.2.2.1. Prim 18
3.2.2.2. Kruskal 18
3.2.3. ARBOLES 19
3.2.3.1. Pre-Orden 19
3.2.3.2. In-Orden 19
3.2.3.3. Post-Orden 20
3.2.3.4. Huffman 20
3.3. SELECCIÓN DE TECNOLOGÍAS DE DESARROLLO 22
3.4. DISEÑO DE LA ESTRUCTURA DEL PROYECTO 26

3.5. ESTRUCTURA DE LOS PAQUETES PRINCIPALES DE LA APLICACIÓN. 26
3.6. ESTRUCTURA DE HERENCIAS. 26
3.7. ESTRUCTURAS DE DATOS 27
3.8. DISEÑO DE LAS INTERFACES GRÁFICAS DE USUARIO 29

CAPÍTULO 4: IMPLEMENTACIÓN DEL SOFTWARE JGRAFOS4 32

4.1. PARÁMETROS GENERALES DEL PROYECTO. 32
4.2. ETAPA DE PRODUCCIÓN DEL SOFTWARE JGRAFOS4. 32
4.2.1. IMPLEMENTACIÓN DE LA CLASE NODEMATRIX (ESTRUCTURA DE DATOS BASE PARA FUNCIONALIDAD) 32
4.2.2. IMPLEMENTACIÓN DE LOS ALGORITMOS DE SOLUCIÓN PARA GRAFOS Y ÁRBOLES 33
4.2.2.1. Clase Algorithm 33
4.2.2.2. Grupo de algoritmos de vector 34
4.2.2.3. Grupo de algoritmos de matriz 34
4.2.2.4. Grupo de algoritmos de estructura 34
4.2.2.5. Implementación de las clases contenedoras de los algoritmos 35
4.2.3. IMPLEMENTACIÓN DE LA GUI (INTERFAZ GRÁFICA DE USUARIO) 35
4.2.3.1. Implementación de componentes 35
4.2.3.1.1. Componentes nativos 35
4.2.3.1.2. Componentes dedicados 36
4.2.3.1.3. Implementación de elementos del canvas 36
4.2.3.1.3.1. Nodo 36
4.2.3.2. *Implementación de un canvas Abstracto* 37
4.2.3.3. *Implementación de canvas dedicados* 37
4.2.3.3.1. Grafo 37
4.2.3.3.2. Árboles 37
4.2.3.3.3. Resultados 38
4.2.3.4. Implementación de las ventanas de trabajo 38
4.2.3.5. *Implementación de la ventana principal* 38
4.2.3.6. *Implementación de la ventana de resultados* 38
4.2.3.7. *Implementación de ventanas secundarias* 38
4.2.4. IMPLEMENTACIÓN DE UNA ESTRUCTURA DE COMUNICACIÓN ENTRE COMPONENTES 38
4.2.5. IMPLEMENTACIÓN DE UN SISTEMA DE TRABAJO PARA MÓDULOS EXTERNOS 38
4.2.5.1. Java Reflection 40
4.2.5.2. Process y ProcessBuilder 40
4.2.5.3. RMI 40
4.2.5.4. Conversor de resultados 41
4.2.6. LANZADOR DE PROCESOS 41

CAPÍTULO 5: PRUEBAS Y RESULTADOS DEL JGRAFOS4 42

5.1. RESULTADOS DE LAS PRUEBAS SOBRE FUNCIONALIDADES GENERALES 42
5.1.1. FUNCIÓN APERTURA DE ARCHIVOS 43
5.1.2. FUNCIÓN DE GUARDADO DE PROYECTO 43
5.1.3. FUNCIÓN DE EXPORTACIÓN DE IMÁGENES 44
5.1.4. FUNCIONAMIENTO DE EXPORTACIÓN DE CONSOLAS 45

5.1.5. FUNCIÓN DE CREACIÓN DE NUEVAS ÁREAS DE TRABAJOS (PESTAÑAS DE TRABAJO) 46
5.1.6. FUNCIÓN DE ASISTENTE DE CREACIÓN DEL PROYECTO. 46
5.1.7. FUNCIÓN DE GENERACIÓN DE GRÁFICOS (CREACIÓN DE NODOS Y ARISTAS) 47
5.1.8. FUNCIÓN DE AGREGACIÓN DE MÓDULOS EXTERNOS. 48
5.1.9. FUNCIÓN DE CAMBIO DE IDIOMAS. 49
5.2. RESULTADOS DE LAS PRUEBAS DE MÓDULOS NATIVOS Y MÓDULOS EXTERNOS 50
5.2.1. RESULTADOS DE LA EJECUCIÓN DE FLOYD (NATIVO) 51
5.2.2. RESULTADOS DE LA EJECUCIÓN DE PRIM (NATIVO) 52
5.2.3. RESULTADOS DE LA EJECUCIÓN DE IN-ORDEN (NATIVO) 52
5.2.4. RESULTADOS DE LA EJECUCIÓN DE FLOYD (MÓDULO EXTERNO) 53
5.3. RESULTADOS DE LAS PRUEBAS SOBRE CONSUMO DE RECURSOS 54

CONCLUSIONES 57

ANEXO 59

Índice de ilustraciones

FIGURA 3.1. DIAGRAMA DE LA ESTRUCTURA GENERAL DEL ALGORITMO DE DIJKSTRA 15
FIGURA 3.2. DIAGRAMA DEL DIAGRAMA DE FLUJO DEL ALGORITMO DE FLOYD. 16
FIGURA 3.3. DIAGRAMA DE FLUJO DEL ALGORITMO DE WARSHALL. 17
FIGURA 3.4. DIAGRAMA DE FLUJO DEL ALGORITMO DE BPF. 17
FIGURA 3.5. DIAGRAMA DE FLUJO DEL ALGORITMO DE PRIM. 18
FIGURA 3.6. DIAGRAMA DE FLUJO DEL ALGORITMO DE KRUSKAL. 19
FIGURA 3.7. DIAGRAMA DE FLUJO DE PRE-ORDEN. 19
FIGURA 3.8. DIAGRAMA DE FLUJO DE IN-ORDEN. 20
FIGURA 3.9. DIAGRAMA DE FLUJO DE POST-ORDEN. 20
FIGURA3.10. DIAGRAMA DE FLUJO DE HUFFMAN. 21
FIGURA 3.11. DIAGRAMA EJEMPLO DE HERENCIAS DE CLASES PARA EL ALGORITMO DE DIJKSTRA. 27
FIGURA 3.12. REPRESENTACIÓN VISUAL DE UNA MATRIZ. 28
FIGURA 3.13. REPRESENTACIÓN VISUAL DE UN CONJUNTO DE MATRICES. 29
FIGURA 3.14. REPRESENTACIÓN DEL GASTO EFICIENTE CON UNA ESTRUCTURA DINÁMICA. 29
FIGURA 3.15. DISEÑO PARA LA VENTANA PRINCIPAL. 30
FIGURA 3.16. DISEÑO GENERAL PARA LA VENTANA DE RESULTADOS EN EL ANALIZADOR GRÁFICO. 30
FIGURA 3.17. DISEÑO GENERAL PARA LA VENTANA DE RESULTADOS EN LA SECCIÓN DE CONSOLA. 31
FIGURA 3.18. DISEÑO PARA LOS ASISTENTES DE CONFIGURACIÓN. 31
FIGURA 3.18. DIAGRAMA DE LA ESTRUCTURA INTERNA DE LOS ATRIBUTOS DENTRO DE LA CLASE NODEMATRIX. 33
FIGURA 4.2. DIAGRAMA QUE REPRESENTA LA EJECUCIÓN DE 2 PROGRAMAS. 39
FIGURA 4.3. DIAGRAMA DE LA PROPUESTA PLANTEADA PARA LA EJECUCIÓN DE MÓDULOS EXTERNOS. 39
FIGURA 5.1. IMAGEN DE LA PRUEBA DE LA FUNCIONALIDAD DE APERTURA DE ARCHIVOS. 43
FIGURA 5.2. PRUEBA DEL SOFTWARE JGRAFOS 4 PARA LA FUNCIONALIDAD DE GUARDADO. 43
FIGURA 5.3. PRUEBA DEL FUNCIONAMIENTO DE GUARDADO DEL PROYECTO. 44
FIGURA 5.4. PRUEBA DE LA FUNCIÓN DE EXPORTACIÓN DE IMÁGENES EN LA VENTANA DE ANÁLISIS USANDO COMO PRUEBA EL ALGORITMO DE FLOYD(NATIVO). 44
FIGURA 5.5. PRUEBA DEL FUNCIONAMIENTO CORRECTO DEL PROCESO DE EXPORTACIÓN DE IMÁGENES. 45
FIGURA 5.6. PRUEBA DEL FUNCIONAMIENTO DE CONSOLA (DESCRIPCIÓN DEL PROCESO INTERNO). 45
FIGURA 5.7. PRUEBA DEL FUNCIONAMIENTO DE EXPORTACIÓN DE CONSOLA EN LA CARPETA CONTENEDORA. 46
FIGURA 5.8. CREACIÓN DE UNA NUEVA PESTAÑA EN EL SOFTWARE JGRAFOS 4. 46
FIGURA 5.9. ASISTENTE DE CONFIGURACIÓN DEL PROYECTO. 47
FIGURA 5.10. GENERACIÓN DE UN GRAFO EN EL SOFTWARE JGRAFOS4. 48
FIGURA 5.11. GENERACIÓN DE UN ÁRBOL EN EL SOFTWARE JGRAFOS4. 48
FIGURA 5.12. ASISTENTE DE INSTALACIÓN DE MÓDULOS EXTERNOS. 49
FIGURA 5.13. INSTALACIÓN DEL MÓDULO EXTERNO CON AYUDA DEL ASISTENTE DE INSTALACIÓN. 49
FIGURA 5.14. CAMBIO DE IDIOMA. 50
FIGURA 5.15. LETREROS TRADUCIDOS AL ESPAÑOL. 50
FIGURA 5.16. GRAFO DIRIGIDO CON 25 NODOS. 51
FIGURA 5.17. RESULTADO DEL CAMINO MÁS CORTO ENTRE EL NODO 0 Y EL NODO 3. 52
FIGURA 5.18. GRAFO NO DIRIGIDO DE 25 NODOS. 52
FIGURA 5.19. ÁRBOL DE EXPANSIÓN MÍNIMA RESULTANTE. 52
FIGURA 5.20. ÁRBOL. 53
FIGURA 5.21. RESULTADO DE LA APLICACIÓN DEL ALGORITMO DE IN-ORDEN. 53

FIGURA 5.22. RESULTADOS DE EJECUCIÓN DEL MÓDULO EXTERNO FLOYD. 54
FIGURA 5.23. VALORES DEL ORDENADOR CUANDO SE ESTÁ DIBUJANDO EN EL ESPACIO DE TRABAJO. 54
FIGURA 5.24. VALORES DEL ORDENADOR CUANDO SE ESTÁ EJECUTANDO EL MÓDULO DE FLOYD CON GRAFO DE 25 NODOS. 55
FIGURA 5.25. VALORES DEL ORDENADOR CUANDO EL SOFTWARE ESTÁ EN ESTADO DE DESUSO. 55

Resumen

Se propone una herramienta de software con la finalidad de resolver problemas sobre caminos más cortos con grafos dirigidos, grafos no dirigidos y árboles n-arios mediante algoritmos básicos y con implementación de técnicas de inteligencia artificial, la herramienta permite la interacción con el usuario mediante una interfaz gráfica amigable y de fácil uso, la cual facilita al usuario ingresar el grafo mediante un lienzo de dibujo para su posterior procesamiento, una vez concluido dicho proceso, el software manda el resultado en 2 formatos, una de forma gráfica dibujando al grafo ingresado en el lienzo inicial y exponiendo la respuesta en otro lienzo, la otra en formato de texto en el cual se visualizan los pasos del proceso realizado así como el tiempo de trabajo del algoritmo, otra de las cualidades del software es la implementación de idiomas el cual permite al programa ser entendible para diversas personas y no solamente entendible con personas que entienden el lenguaje español, esto posibilita llegar a más personas, otra cualidad implementada es un sistema de multi trabajo, el cual facilita trabajar en diferentes pestañas diversos problemas al mismo tiempo, dicho sistema contempla almacenar los grafos en un archivo para su uso posterior, así como, la exportación de imágenes del grafo ingresado y procesado.

Capítulo 1 Introducción

1.1. Antecedentes del proyecto

Este trabajo de tesis fue producto de un software con derechos de autor por el Instituto Nacional del Derecho de Autor (INDAUTOR) y numero: 03-2022-010310403700-01, llamado JGRAFOS 4. Donde se participó con un grupo de investigadores tanto profesores como alumnos. En este software se pone a prueba cada una de estas ideas implica la realización de nuevos sistemas en donde se tienen que ajustar parámetros para replicar ciertos fenómenos y probar dichas ideas, esto es un problema en sí mismo, dado a que varios de esos problemas pueden solucionarse con casi los mismos parámetros con un ajuste adecuado, exponiendo un ejemplo concreto sería en cuanto a la teoría de grafos, actualmente contamos con algoritmos tales como Dijkstra, Floyd (para encontrar caminos más cortos), Prim, Kruskal (para encontrar árboles de expansión mínima), etc. Sin embargo, al ver los procesos que conlleva cada uno de los algoritmos hasta la fecha, se puede llegar a la conclusión de poder optimizarlo en cuanto a recursos (como el tiempo de ejecución o incluso en memoria de operación).

1.2. Objetivos generales y específicos del proyecto

El objetivo general consistió en el desarrollo de funcionalidades a una herramienta de software cuya finalidad fuera la ejecución de módulos creados por fabricante y externos para la programación individual de algoritmos con grafos.

En cuanto a los objetivos específicos:

- Se creo una herramienta de software que realiza el cálculo de grafos y arboles n-arios de forma eficiente, sin hacer uso desmedido de los recursos de una PC como la memoria RAM, CPU y GPU, además del procesamiento de dichos esquemas y hacer entrega de los resultados en forma gráfica mediante diagramas nuevos y en formato de texto.
- Se construyó un software que facilita la resolución de problemas relacionados a los grafos y arboles n-arios mediante módulos incluidos por defecto en el software y mediante nuevos módulos agregados por personas externas al proyecto con el fin de implementar y verificar sus nuevas ideas.
- Se realizó un software que está disponible en la mayoría de las plataformas (sistemas operativos) así como en más de un idioma con la finalidad de ser usado por el mayor número de usuarios posible.

1.3. Metodología

En cuanto a la metodología que se usó para este proyecto, se optó por dividir el trabajo en 4 etapas con la finalidad de tener un proceso ordenado, simplificado y resultando un eficiente manejo de errores.

En la etapa de documentación, se comenzó con una investigación cualitativa referente al estudio de los grafos, esto con la finalidad de obtener un mejor entendimiento del tema y poder hacer una mejor manipulación de las estructuras, dicha investigación impacto en diversas etapas del trabajo, incluyendo el lenguaje de programación empleado en el producto.

En la etapa de planeación se optó en ordenar y sistematizar el estado del arte que comprendió tanto la teoría como las implicaciones prácticas, esto resultó en la validación de la hipótesis inicial en cuanto a su novedad y necesidad.

Para la etapa de implementación se manejó un modelo mixto entre el modelo incremental y el modelo de espiral, debido a que el producto se hizo de forma modular, esto implicó el desarrollo de módulos pequeños de forma independiente, pero con gran importancia en conjunto, cada uno de estos avances tuvo un proceso muy similar, también se intentó ser redundante con la finalidad de minimizar el mayor número de errores posibles.

Finalmente, en la etapa de testeo, se diseñó e implementó una serie de pruebas de caja blanca y caja negra, esto con la finalidad de corroborar la eficiencia y confianza de cada módulo terminado. En dichos diseños se tomaron en cuenta casos extremos, así como casos intermedios en donde el programa funcionó con facilidad y demostró eficiencia.

Capítulo 2 Estado del Arte

2.1. Recursos relacionados

En [1] se describe una herramienta llamada Grafos que permite la construcción, edición y el análisis de grafos, este software permite el modelado mediante una interfaz gráfica construida en Visual Studio y su análisis mediante algoritmos de resolución de grafos como Dijkstra, Bellman-Ford, entre otros, dicho software está disponible únicamente para Windows.

En [2] se tiene la herramienta llamada Gephi, se trata de un software open-source el cual permite la visualización de grafos, este software es usado en estudios científicos en el análisis de datos de internet y de redes sociales, esta herramienta es multiplataforma, además posee un motor de renderizado de alto rendimiento lo cual permite manipular diversos modelos de grafos y encontrar patrones de comportamiento de forma intuitiva. Los algoritmos de diseño en la herramienta permiten optimizar diversos elementos que ayudan al análisis y procesamiento de los datos.

En [3] se describe el software ALGRAF el cual está diseñado para el estudio de algoritmos en los grafos. Este software tiene fines didácticos como el mejorar la comprensión de algunos conceptos.

En [4] se describe el software ProRouting, el cual visualiza diversas estructuras de proximidad definidas sobre un conjunto de nodos en un plano, permite visualizar estrategias de ruteo en el grafo siguiendo criterios locales, este software sirve como herramienta didáctica para la comprensión del funcionamiento de ciertos algoritmos, además de servir como herramienta para estudio de propiedades que ayuden a encontrar estrategias eficientes orientadas a grafos geométricos.

En [5] se describe LINKURIOUS, es una herramienta de visualización de grafos, este software está orientado a la investigación y análisis, además de ayudar a dependencias gubernamentales contra el lavado de dinero y en cuestiones de seguridad cibernética.

En el ámbito de bibliotecas de visualización de grafos [6] se tiene a SigmaJs el cual es una biblioteca escrita en JavaScript que permite únicamente el modelado gráfico de un grafo en un ambiente web mediante renderización de canvas y WebGl.

Otra biblioteca de visualización y análisis de grafos [7] es JUNG, esta es una biblioteca multiplataforma y Open-Source diseñada en java con el fin de modelar y analizar grafos, además permite representar nodos posicionados en un mapa.

2.2. Investigación sobre la historia de los grafos

Leonhard Euler [8] fue un matemático y físico suizo en el siglo XVIII, es considerado como el padre de la teoría de grafos gracias a su trabajo en 1735 sobre el primer problema registrado sobre la teoría de grafos, los siete puentes de Königsberg, en donde se expresa la siguiente problemática.

“Dado el mapa de Königsberg, con el rio pregel dividiendo el plano a través de los siete puentes, ¿Es posible dar un paseo comenzando desde cualquiera de estas regiones pasando por todos los puentes, recorriendo solo una vez cada uno y regresando al mismo punto de partida?”

Dada la situación anterior Euler expresó las siguientes soluciones:

- Si el punto de llegada y salida coincidían, obligatoriamente debería de existir un número par de aristas, en cuyo caso se hablaría de un “ciclo euleriano”.
- Si el punto de llegada y salida no coincidían, obligatoriamente debería de existir un número impar de aristas, en cuyo caso se estaría hablando de un “camino euleriano”.

Edsger Wybe Dijkstra [9] es un científico de la computación holandés quien recibió un premio por la asociación de sistemas computarizados por sus aportes en la rama de la computación distribuida, entre sus trabajos destacados se encuentra el algoritmo de Dijkstra, por el cual recibió el premio de Turing en 1972.

El algoritmo de Dijkstra [13] es un algoritmo que determina el camino más corto entre un punto A y un punto B, requiere de un vértice de origen (punto A) y una matriz de pesos que describa al grafo, una vez el proceso concluye, mostrará los caminos mínimos del vértice de origen a cada uno de los otros vértices si y sólo si existe al menos un camino.

Paul Erdös [14] fue un matemático húngaro que nació en 1913, fue uno de los matemáticos más prolíficos (después de Euler) tras colaborar con 500 colegas y publicar 1500 artículos, sus aportaciones fueron en la teoría de números, combinatoria, teoría de grafos, análisis, geometría y teoría de grafos, se le adjunta el teorema de Erdös – Stone que es un resultado asintótico por la generalización del teorema de Turán para limitar los números de vértices en un grafo de grado H – libre por un grafo completo H, este teorema es mejor conocido como el teorema fundamental de teoría de grafos extrémales.

Dénes König [15] fue un matemático húngaro muy conocido por su tratado sobre teoría de grafos, König demostró el siguiente teorema: El máximo tamaño de un emparejamiento en un grafo bipartito G es igual al mínimo tamaño de una cobertura en G”.

Kazimierz Kuratowski [16] fue un matemático y lógico polaco, fue uno de los principales representantes de la Escuela de Matemáticas de Varsovia, su aportación a la teoría de grafos fue el Teorema de Kuratowski, el cual se define como: “Un grafo es planar si y solo si no contiene un subgrafo que es una subdivisión elemental K5 (el grafo completo de 5 vértices) o K3,3 (el grafo bipartito completo de 6 vértices)”. Sin embargo, el teorema se puede llegar a simplificar en el siguiente enunciado: “un grafo es planar si y solo si no contiene ningún subgrafo homeomorfo a K5 o K3,3”.

Gerhard Ringel [17] fue un matemático alemán, que obtuvo su doctorado por la Universidad de Bonn en 1951, fue uno de los pioneros en teoría de grafos, en cuanto a su contribución a esta área esta la conjetura de Heawood también llamado teorema de Ringel-Youngs, el cual da un límite inferior a la cantidad de colores que son necesarios para la coloración gráfica en una superficie de dado un género, para superficies de género 0,1, 2, … El número requerido de colores es 4,7,8.

2.3. Algoritmos relacionados al estudio de grafos

En cuanto a la tecnología existente referente al campo de los grafos, se investigó a los siguientes algoritmos:

Algoritmo de Dijkstra [18] creado por Edsger Dijkstra (publicado en 1959). Es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista.

Algoritmo de Floyd [19] (1962). Es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución.

Algoritmo de Prim [20] diseñado por Vojtech Jarnik (1930) y Robert C. Prim (manera independiente 1957). Es un algoritmo perteneciente a la teoría de los grafos para encontrar un árbol que recubra el mínimo grafo conexo, no dirigido y cuyas aristas están etiquetadas.

Algoritmo de Kruskal [20] creado por Joseph Kruskal (1956). Es un algoritmo de la teoría de grafos para encontrar un árbol que recubra el mínimo del grafo conexo y ponderado.

Algoritmo de Ford-Fulkerson [21] creado por L. R. Ford, Jr. y D. R. Fulkerson (1956). Propone buscar caminos en los que se pueda aumentar el flujo, hasta que se alcance el flujo máximo.

Algoritmo de Bellman – Ford [22] creado por Lester Randolph Ford y Richard E. Bellman (entre los años de 1920 y 1940). Genera el camino más corto en un grafo dirigido ponderado (en el que el peso de alguna de las aristas puede ser negativo).

2.4. Conceptos relacionados a modularidad

En cuanto a los antecedentes del objetivo general se encontró la siguiente información:

Modularidad. [23] La modularización es un proceso el cual permite agrupar las instrucciones que cumplen una función en específico en una función, esto permite varias cosas como por ejemplo la reutilización de código, es decir, que se puede usar ese segmento de código más de una vez y que permite simplificar el programa, también mitiga los errores al concentrarlos en una sección del código de un programa.

2.5. Programación orientada a objetos

Es [24] un paradigma de la programación que permite la reutilización de código, así como llevar una mejor organización además de una mejor manutención, además evita la repetición de código para crear programas más eficientes.

Capítulo 3: Análisis y diseño

3.1. Etapa de investigación

De acuerdo con la metodología propuesta, el primer paso fue realizar una investigación en cuanto al objeto de estudio el cual fue el grafo, los cuales describen problemáticas diversas y su estructura es muy flexible y se apoya de las matrices como la principal forma de su desglose, desde este punto podemos ver a la matriz como la principal estructura de datos a usar, y su tamaño dependerá de forma proporcional al número de nodos empleados para describir el grafo, esto juega un papel importante a la hora de obtener resultados, ya que entre mayor sea el número de nodos mayor cantidad de cálculos se realizarán para obtener el resultado al igual que ocupará más almacenamiento.

La investigación se basó en torno al libro de estructura de datos de Alfred V. Aho, John E. Hopcroft, Jefrey D. Ullman, por lo cual se tomaron los 10 algoritmos del libro para poder poner a prueba la Hipótesis expuesta en esta tesis, también cabe aclarar que se consultaron diversas fuentes para poder entender de mejor manera los algoritmos descritos en el libro de estructura de datos, a continuación se hablará la lógica entendida sobre cada algoritmo y cuál sería su posible solución en un ambiente computacional

3.2. Análisis de los algoritmos

3.2.1. Grafos dirigidos

3.2.1.1. Dijkstra

El algoritmo de Dijkstra (véase en figura 3.1) tiene como propósito el encontrar el camino más corto entre un nodo “a” hacia todos los demás nodos, su lógica seria la siguiente:

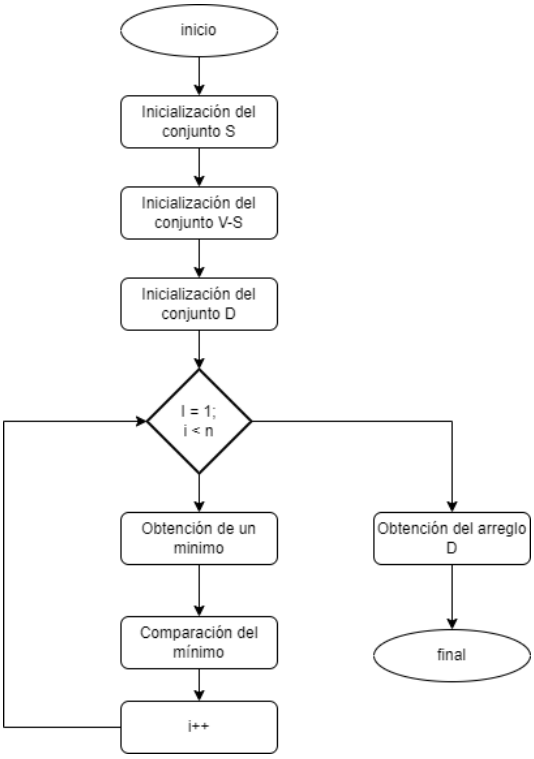


Figura 3.1. Diagrama de la estructura general del algoritmo de Dijkstra

En cuanto a las posibles estructuras de datos a usar para manipulación de la información serian una matriz la cual describa al grafo (Matriz de pesos), un arreglo unidimensional en donde se guarden los costos de los caminos resultantes y por último un conjunto de matrices que describan el camino más corto.

3.2.1.2. Floyd

El algoritmo de Floyd (véase en figura 3.2) tiene como propósito encontrar los caminos más cortos entre los nodos de un grafo, es decir debe buscar todos los caminos más cortos posibles entre 2 nodos, su lógica es la siguiente:

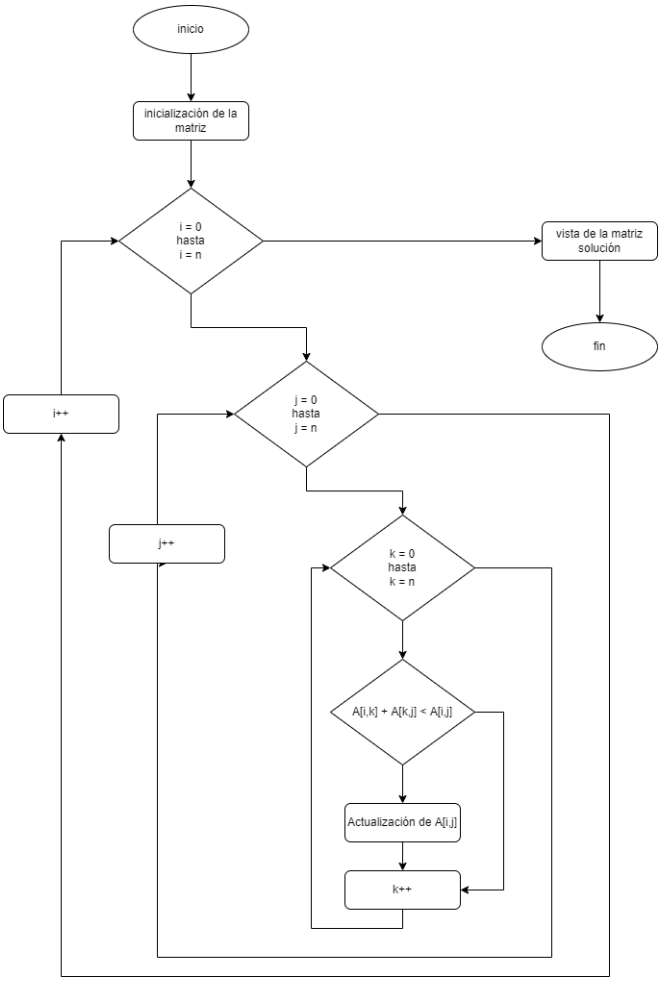


Figura 3.2. Diagrama de la estructura general del algoritmo de Floyd.

En cuanto a las posibles estructuras de datos a usar para manipulación de la información serian una matriz la cual describa al grafo (Matriz de pesos), y por último un conjunto de matrices que describan el camino más corto.

3.2.1.3. Warshall

El algoritmo de Warshall (véase en figura 3.3) verifica la “cerradura transitiva”, es un algoritmo muy similar al algoritmo de Floyd, incluso este algoritmo le antecede, su lógica es la siguiente:

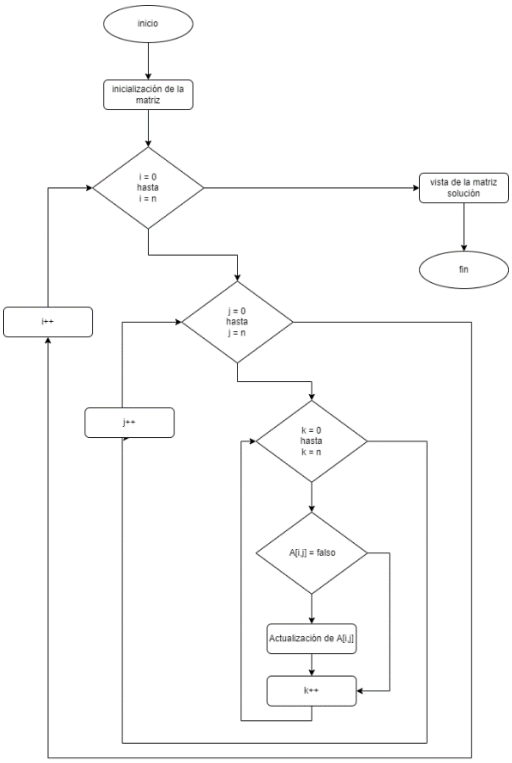


Figura 3.3. Diagrama de flujo del algoritmo de Warshall.

En cuanto a las posibles estructuras de datos a usar para manipulación de la información y dado que es un algoritmo muy similar al algoritmo de Floyd, las estructuras contempladas serian una matriz la cual describa al grafo (Matriz de conexiones), y por último un conjunto de matrices que describan el camino más corto.

3.2.1.4. BPF

El algoritmo BPF (véase en figura 3.4) es un algoritmo que realiza búsqueda de nodos en profundidad en un grafo determinado, se trata de un algoritmo recursivo, su lógica es la siguiente:

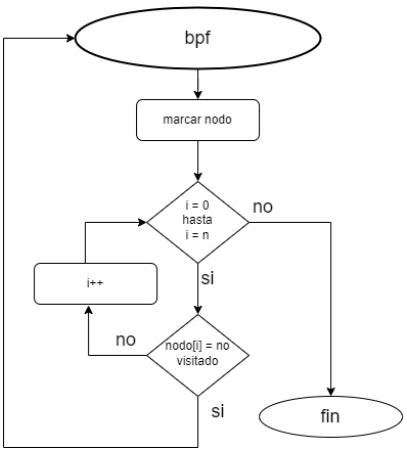


Figura 3.4. Diagrama de flujo del algoritmo de BPF.

Como posible estructura de trabajo se ubica un arreglo con la función de capturar el orden de los nodos con forme se encuentren, este tipo de algoritmos no requieren de una estructura especializada.

3.2.2. Grafos no Dirigidos

3.2.2.1. Prim

El algoritmo de Prim (véase en figura 3.5) tiene como finalidad el encontrar el árbol de expansión mínima, su forma de búsqueda es sistemática, es decir que basa su decisión en base a las decisiones en el momento, su lógica es la siguiente:

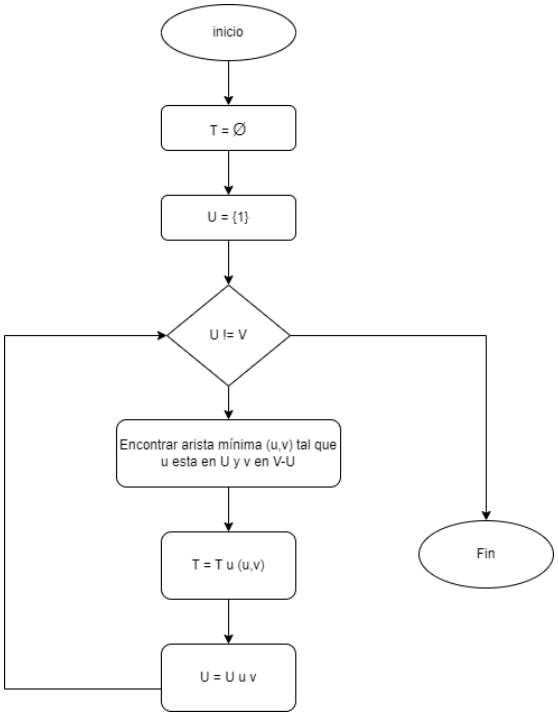


Figura 3.5. Diagrama de flujo del algoritmo de Prim.

3.2.2.2. Kruskal

El algoritmo de Kruskal (véase en figura 3.6) tiene como finalidad el encontrar el árbol de expansión mínima, aunque su fin es similar al del algoritmo de Prim, su forma de búsqueda es distinto, ya que, en lugar de ser sistemático, este es más práctica, ya que va en base a la arista con menos peso, su lógica es la siguiente:

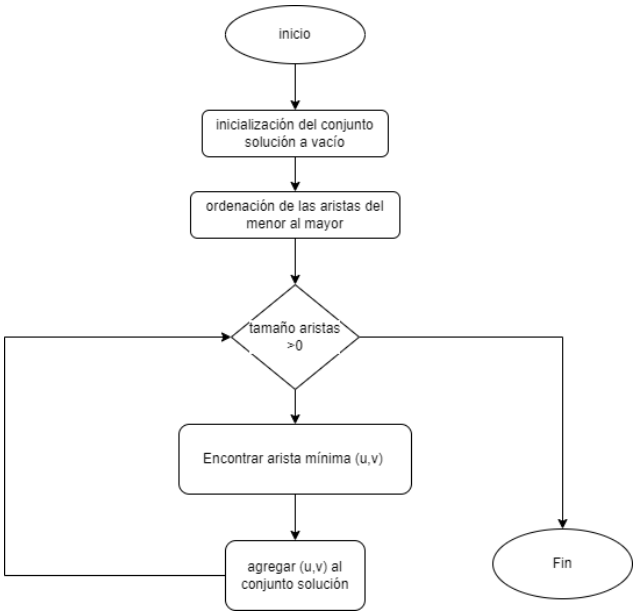


Figura 3.6. Diagrama de flujo del algoritmo de Kruskal.

3.2.3. Arboles

Para este tipo de algoritmos, la estrategia de trabajo varia con respecto a las anteriores, ya que estos únicamente requieren de una estructura que guarde los caminos, por lo que su posible estructura de datos a usar sea un vector que guarde dichos caminos.

3.2.3.1. Pre-Orden

El algoritmo de Pre-Orden (véase en figura 3.7) es un algoritmo para árboles, su función es recorrer el árbol llegando hasta la raíz de este y después a cada uno de sus hijos en caso de tenerlos de forma de orden previo, se trata de un algoritmo recursivo, su lógica es la siguiente:

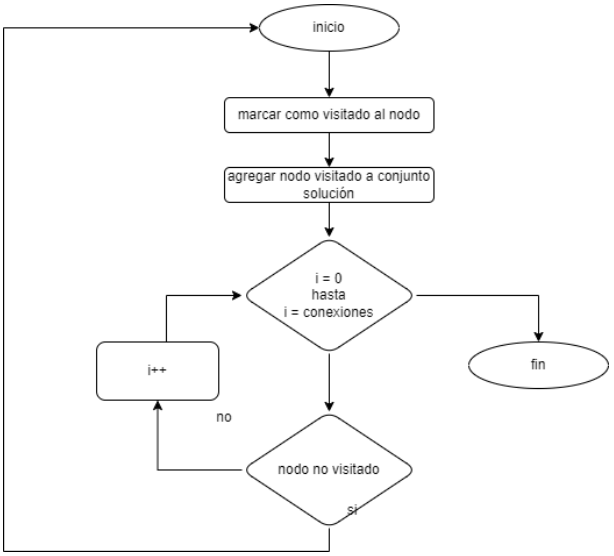


Figura 3.7. Diagrama de flujo de Pre-Orden.

3.2.3.2. In-Orden

El algoritmo de In-Orden (véase en figura 3.8) es un algoritmo para árboles, su función es recorrer el árbol de forma “simétrica”, es decir, primero al hijo de la izquierda luego la raíz y después al hijo de la derecha, se trata de un algoritmo recursivo, su lógica es la

siguiente:

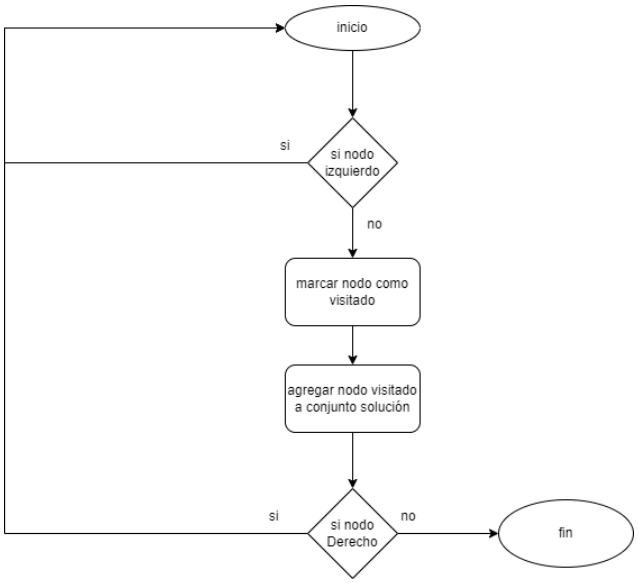


Figura 3.8. Diagrama de flujo de In-Orden.

3.2.3.3. Post-Orden

El algoritmo de Post-Orden (véase en figura 3.9) es un algoritmo para árboles, su función es recorrer el árbol priorizando a los hijos y después a la raíz, se trata de un algoritmo recursivo, su lógica es la siguiente:

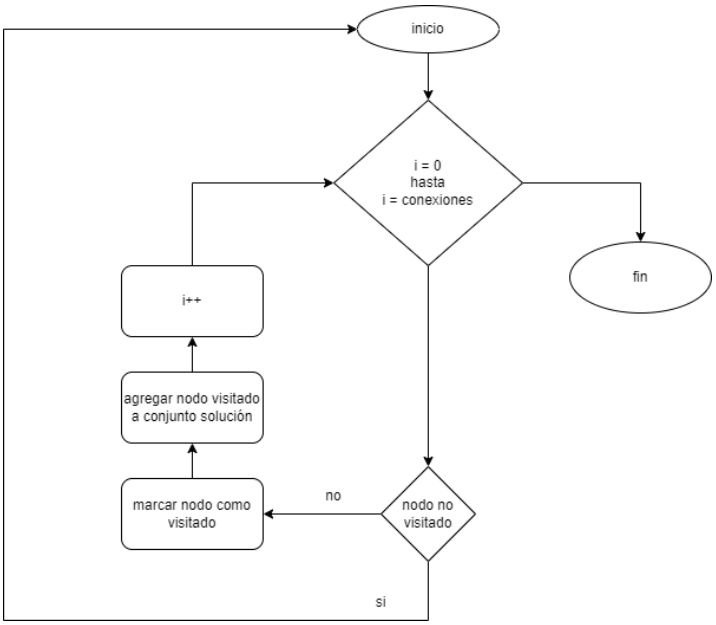


Figura 3.9. Diagrama de flujo de Post-Orden.

3.2.3.4. Huffman

El algoritmo de Huffman (véase en figura 3.10) es diferente a los algoritmos relatados hasta ahora, ya que este en lugar de buscar caminos o recorrer el árbol, este construye un árbol, una de sus características más notorias es que crea nodos con un valor, su lógica es la siguiente:

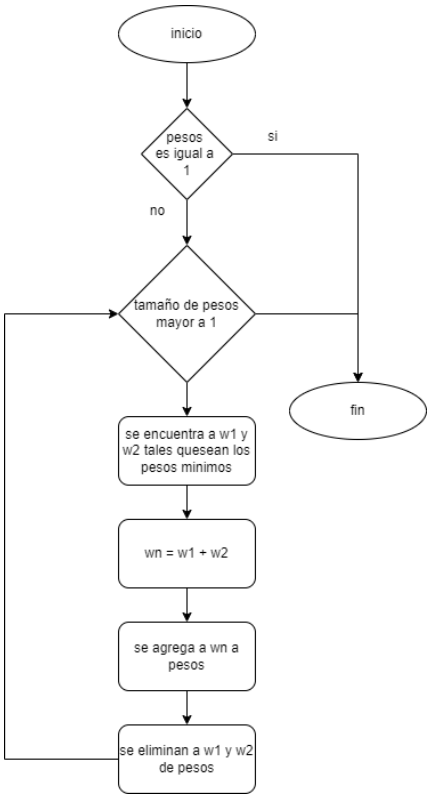


Figura3.10. Diagrama de flujo de Huffman.

3.3. Selección de tecnologías de desarrollo

Para el desarrollo del software se contemplaron diversas tecnologías (véase tabla 3.1), las cuales fueron categorizadas por ambientes de trabajo indicando sus ventajas y sus desventajas, lo cual permitió tomar una decisión apta en cuanto a la plataforma de destino.

Tabla 3.1 Tecnologías de desarrollo (por plataforma)

Ambiente de trabajo	Lenguajes disponibles	Ventajas (pros)	Desventajas (contras)
Mobile	• Java (jvm) • Kotlin (jvm) • JavaScript/Typescript • Xamarin • Flutter (Dart) • C/C++	• Al ser ejecutados en dispositivos móviles se tiene una cercanía más próxima al usuario, por lo que se llegaría a más usuarios.	• Los dispositivos móviles tienen una capacidad de procesamiento muy limitado, por lo que el uso del procesador en cálculos específicos y complejos podría ser más tardado.
Desktop	• Java (jvm) • Javascript/Typescript • C# • C/C++ • Python • Swift (Apple)	• Los ordenadores son capaces de realizar cálculos más complejos en un tiempo relativamente corto. • Permiten el uso “offline” o mejor conocido como ejecución local, lo que permite ejecutar programas sin depender de cómputo externo (servidores). • Permite la ejecución de máquinas virtuales y de programas nativos.	• Depende del rendimiento de los componentes principales (CPU, GPU y Memoria RAM). • No es un dispositivo compacto.
Web	• JavaScript/Typescript • Java (jvm) • Flutter (Dart) • PHP • Rubi • Python • HTML/CSS	• Se puede ejecutar en cualquier dispositivo que admita tecnologías web. • No es necesario cargar con un dispositivo único, ya que el software está alojado en un dispositivo remoto y permite su uso en diversos dispositivos. • Los cálculos complejos se pueden distribuir en microservicios.	• Depende de la conectividad de internet. • Para realizar cálculos complejos es necesario contratar servicios externos y de paga. • Por lo regular las tecnologías web son interpretadas por lo que son más lentas. • Las tecnologías web necesitan de una aplicación de navegador para su

			uso, por lo que se lleva un costo agregado en el procesamiento y almacenamiento en memoria RAM.

Como se pudo observar en la tabla anterior (tabla 3.1) cada uno de los ambientes contemplados tienen sus ventajas y desventajas, sin embargo, para poder tener un veredicto en cuanto al ambiente de trabajo, es necesario analizar sus bases haciendo referencia a los dispositivos finales (ordenadores, laptops, celulares, etc.), ya que, hay que recordar que la tecnología esta estrechamente relacionada con el hardware, ya que este le permitirá al software cierta ventaja o desventaja frente a otras debido a su potencia.

Para realizar dicho análisis, es necesario tomar en cuenta las limitaciones de cada uno de los dispositivos finales (laptops, teléfonos celulares, tabletas y ordenadores de sobremesa), ya que, en la actualidad si bien es cierto que los teléfonos celulares se van acercando a lo que es un ordenador común, estos no están diseñados para una carga de trabajo tan pesada, hay que recordar las características de un smartphone común, estas serían, un chip de gama media de aproximadamente entre 4 y 8 núcleos, con una frecuencia de reloj de hasta 2.0 GHz y con una memoria RAM de hasta 4 GB como máximo (hablando en términos de una gama media), en cuanto al chip, este tal vez podría cumplir con la tarea de cálculos, sin embargo, aquí viene la primera limitación en cuanto a este dispositivo, hay que recordar que cada calculo en una CPU representaría una serie de pulsos eléctricos en un conductor, estos pulsos eléctricos al pasar generan calor y el calor en exceso puede llegar a inutilizar al dispositivo, por lo que, se necesitaría de una unidad de refrigeración para este tipo de circunstancias, sin embargo, en los smartphones y tabletas, estos no existen, por lo que, hacer uso de este tipo de dispositivos para el software seria casi imposible, ya que los procesos demandan recursos y cálculos, lo que para fines prácticos significaría una fuente de calor creciente a tal punto de inutilizar el equipo, sin embargo, hay otra salida y esta seria diversificarlo en servidores para que se pueda acceder a las herramientas del software y este pueda ser usado en dispositivos móviles, sin embargo, aquí existe otra limitación, aunque si se podría adquirir ciertas herramientas de cómputo en la nube para poder distribuir la carga de trabajo y así poder quitar esta parte a los dispositivos móviles, el problema comienza en el aumento de costos, ya que no se cuentan con recursos para poder rentar las herramientas de cómputo que ofrece Amazon y Oracle, además, otro problema seria la velocidad de conexión a internet, ya que esta velocidad tiene un impacto fuerte en cuanto el tiempo de respuesta del software, ya que una baja velocidad de internet supone un incremento en el tiempo de respuesta, finalmente, en el caso de las laptops y los ordenadores de sobremesa, estos tienen limitaciones en cuanto a sus componentes, si bien es muy similar al caso de los dispositivos móviles, aquí aparece la gran diferencia y es que en la mayoría de dispositivos (hablando únicamente de los ordenadores de sobremesa y de las laptops) se encuentra el dispositivo de refrigeración (ventilador o sistema de refrigeración líquido, según sea el caso) que permite que el procesador no sufra un sobrecalentamiento, además en los procesadores actuales se está implementando una protección que actúa frente a un sobrecalentamiento, limitando su frecuencia de reloj para evitar temperaturas mayores a las soportadas por cada CPU.

Para reafirmar lo comentado con anterioridad de una forma sencilla, se realizó una rúbrica para determinar cuál plataforma de trabajo era óptima para realizarlo, dicha rúbrica solo contempló los rubros relacionados con las exigencias del proyecto, (las cuales se encuentran en la tabla 3.2).

Tabla 3.2 Rubrica de selección de ambiente de trabajo.

Rubrica/Ambiente	Mobile	Desktop	Web
Facilidad de realización de cálculos complejos (CPU)	X	✓	✓
Permite el uso Offline (sin acceso a internet)	✓	✓	X
Disponibilidad de bancos de memoria (RAM)	X	✓	✓

Como se puede observar en la tabla anterior (tabla 3.2) las mejores plataformas de trabajo son la Web y Desktop, sin embargo, como se comentó con anterioridad, para usar la plataforma Web, es necesario rentar ciertos servicios de cómputo en la nube para poder llevar a cabo las tareas pesadas, además, como uno de los parámetros de este proyecto requiere la ejecución local de algoritmos, entonces eso convierte a la plataforma de Desktop como la idónea, una vez elegida la plataforma de trabajo, entra otra cuestión en cuanto al lenguaje de desarrollo (vea comparativas en tabla 3.3), el cual es un punto fundamental ya que el lenguaje debe cumplir ciertos criterios como multiplataforma, ser rápido y estable.

Tabla 3.3 Tabla sobre comparativas entre lenguajes.

Lenguaje	Pros	Contras	Sistemas operativos
Java	• Es multiplataforma. • Es de propósito general. • Es estructurado. • Es rápido. • Es compilado. • Se ejecuta en máquina virtual. • Existen librerías nativas para creación de interfaces gráficas de usuario.	• Es más lento que C/C++. • Requiere más código para ser funcional.	• Windows. • Linux (todas las distribuciones). • IOS.
JavaScript /Typescript	• Es multiplataforma. • Es de propósito general. • Es relativamente rápido. • Requiere de menos código para ser funcional.	• Es más Lento que Java. • Es estrictamente interpretado. • Para dar estructura se necesita de un compilador extra (Typescript).	• Windows. • Linux (todas las versiones). • IOS.

	• Estructuras más sencillas.	• Es más fácil tener errores de lógica. • Requiere más memoria RAM para funcionar (electrón). • Requiere de frameworks para tener una interfaz gráfica.	
C#	• Es multiplataforma. • Es de propósito general. • Es rápido. • Se ejecuta en forma nativa.	• Requiere más código para ser funcional. • Es más lento que C/C++.	• Windows. • Linux (todas las versiones). • IOS.
C/C++	• Es de propósito general. • Es el más rápido de todos (sin contar al ensamblador). • Es compilado.	• Requiere más código para ser funcional.	• Windows. • Linux (todas las versiones). • IOS.
Python	• Es multiplataforma. • Es de propósito general. • Es rápido (Sin embargo, en [25] se expone un documento en donde se prueba la lentitud de Python frente a lenguajes como Java y C). • Es interpretado. • Es funcional con menos código escrito.	• Es mas lento de C/C++ y Java (como se expone en [25]).	• Windows. • Linux (todas las versiones). • IOS.

Dada la tabla anterior se podría decir que C/C++, Java y Python podrían ser las elecciones de lenguajes más indicadas para crear el producto final, sin embargo, tomando en cuenta la información, cada uno presenta ciertos inconvenientes, por lo que se trató de elegir uno mediante los siguientes rubros:

- Complejidad de cálculos (rapidez).
- Interfaz gráfica (componentes gráficos).
- Ergonomía (facilidad de codificación).

Con base a la nueva rubrica se eligió usar Java como lenguaje de programación, esto debido a que Java es más rápido que Python (según la tabla 3.3), y aunque C/C++ es más rápido que Java en este rubro, C/C++ tiene inconvenientes en cuanto al proceso de diseño de interfaces gráficas, ya que es más laborioso y dependiente del sistema operativo, además las librerías de javax.swing son más robustas y son multiplataforma, lo cual permite trabajar mejor en él ambiente gráfico.

3.4. Diseño de la estructura del proyecto

Una vez elegidas las herramientas de trabajo, así como las plataformas de destino, se procedió a diseñar la estructura del proyecto, esto con el fin de ahorrar tiempo de trabajo, así como de tener un proyecto optimizado y estructurado, cabe destacar que la aplicación tuvo actualizaciones a lo largo del tiempo lo que también influyó en la estructura de la aplicación.

3.5. Estructura de los paquetes principales de la aplicación.

La aplicación fue dividida por paquetes que contienen grupos de clases relacionadas a un funcionamiento determinado, por lo que la estructura de paquetes de la aplicación fue de la siguiente forma:

- JController: Contiene todas las clases de procesos activos (hilos de ejecución).
- JGui: Contiene Todas las clases necesarias para la interfaz gráfica (IU).
- JServices: Contiene clases de servicios de la aplicación.
- JStruct: Contiene las clases necesarias para las estructuras de datos.
- JSystem: Contiene las clases necesarias para el funcionamiento interno de la aplicación.
- Jgrafos5: Contiene la clase que ejecuta a la aplicación (método main).

Internamente cada uno de los paquetes contienen otros paquetes con grupos más pequeños de clases especializadas.

3.6. Estructura de Herencias.

Debido a que Java resulto electo como el lenguaje de trabajo, facilito el uso de el paradigma de la programación orientada a objetos, la cual resulta muy acorde al tipo de trabajo desempeñado, además permitió la reutilización de código debido a las propiedades de este paradigma (modularidad y herencia de clases), en un ejemplo simple, la clase Dijkstra (clase que ejecuta el algoritmo de Dijkstra) tiene 2 super clases, de las cuales permite heredar todas sus características como se muestra en la siguiente imagen (véase en figura 3.11):

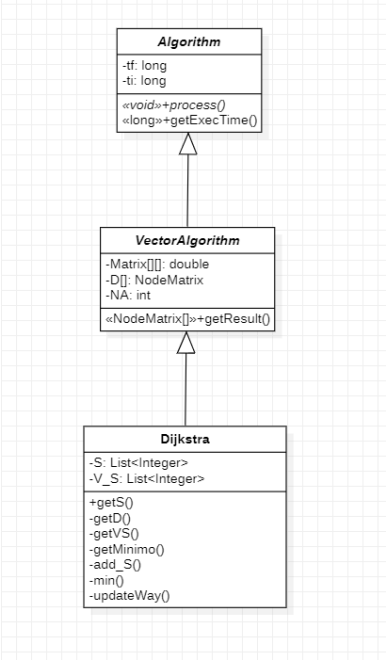


Figura 3.11. Diagrama ejemplo de herencias de clases para el algoritmo de Dijkstra.

Como se puede apreciar en la imagen, la clase “Algorithm” es la clase con la jerarquía más alta, y además es abstracta por lo que se pueden definir unos algunos prototipos de funciones que serán de utilidad en clases hijas, además que permitirá heredar tanto los atributos como los métodos de esta clase, después, siguiendo la misma jerarquía se encuentra la clase “VectorAlgorithm”, esta clase es especializada para las clases que resuelven problemas con vectores como el caso del Algoritmo de Dijkstra, posee métodos más especializados, como es el caso del método de “getResult”, este método devuelve un Arreglo de elementos además de declarar como atributos una matriz, un arreglo y una variable de tipo entera, finalmente se encuentra la clase final, la clase de Dijkstra que finalmente resuelve los problemas con el algoritmo de Dijkstra, y que hereda los atributos y métodos de sus antecesoras, sin embargo, el punto importante de tal división es la reutilización de código ya que la clase “VectorAlgorithm” no es la única clase que hereda de la clase de “Algorithm”, ya que también se encontraron otros grupos de algoritmos que comparten características, los cuales se pueden englobar en algoritmos de matrices y algoritmos de estructuras, y esas clases sirven a otras clases contenedoras de algoritmos de resolución de grafos, además, otra de las cualidades de este paradigma es que permite agrupar clases en base a su superclase, esto también fue útil a la hora de hacer la implementación del software, otro caso muy similar fue en la UI, ya que también se usó la herencia de clases y clases abstractas para poder hacer la reutilización de código y además poder obtener características muy similares entre clases, el paradigma orientado a objetos jugo un papel muy importante en esta etapa del desarrollo del proyecto ya que permitió ahorrar código al momento de crear una estructura generalizada dependiendo del caso de aplicación.

3.7. Estructuras de datos

Como se comentó en la etapa de análisis de los algoritmos, se encontraron formas de manejo para el grafo, uno de los más recurrentes fue la matriz, ya que está podría ser usada para describir al grafo e incluso para poder encontrar las soluciones a los problemas planteados (como encontrar el camino más corto), sin embargo, haciendo un

análisis profundo sobre la recomendación del uso de matrices para la resolución de los grafos, este tipo de solución puede ser vista como una arma de doble filo, si bien permite hacer una descripción del grafo en cuanto a sus conexiones, el uso en su método de recolección de respuestas puede ser muy perjudicial en cuanto al costo de memoria ya que se tendría desperdicio, para poder comprender esta idea es necesario ver a la matriz de forma literal (véase en figura 3.12) el cual es el conjunto de bloques de memoria de un tamaño predefinido (32, 64 bits, etc.), las matrices una vez declaradas de un tamaño son inmutables en tal aspecto durante el resto de la ejecución, por lo tantoes una buena idea si se sabe con certeza si todos los bloques que conforman a la matrizse podrán ocupar, pero si no hay certeza de ello podría jugar un papel en contra.

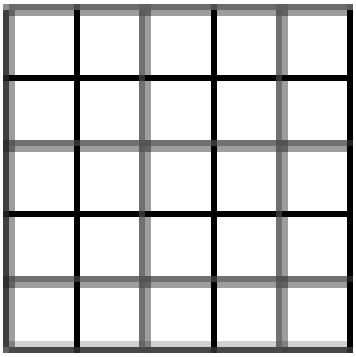


Figura 3.12. Representación visual de una matriz.

Como ejemplo de lo expuesto anteriormente, se analizó el caso del algoritmo de Floyd, ya que en este algoritmo es más visible el tema que se está tratando, hay que recordar que Floyd funciona con matrices principalmente, y una vez que se termina con dicho proceso, el algoritmo necesita indicar el camino más corto, en esta parte se hace visible el desperdicio de las matrices (véase en figura 3.13) en cuanto a su utilización, ya que, cuando se hace un cambio en la matriz de pesos en la ejecución del algoritmo, esté va haciendo una réplica de la matriz y crea una marca para poder realizar una regresión, y este proceso se repite varias veces hasta que finaliza el programa, por lo tanto, se tendrían “n” matrices siendo “n” el número de ejecuciones, además hay que agregar el peso de cada uno de los bloques apartados en la memoria, y también que no siempre existen cambios, por lo que, estos factores hacen una ejecución pesada en términos de la memoria RAM, en ordenadores actuales y de “alta gama” esto podría pasar desapercibido en la mayoría de casos siempre y cuando el grafo no sea tan grande (esto puede ser relativo dependiendo de las características del Hardware), sin embargo, en un ordenador de bajos recursos o de “gama baja” sería muy perceptible y esto podría alterar de una forma desagradable la experiencia de usuario en una ejecución de incluso un grafo pequeño.

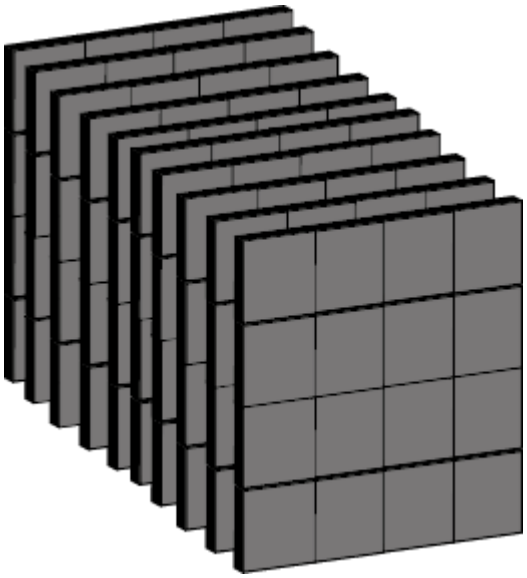


Figura 3.13. Representación visual de un conjunto de matrices.

Ante aquella situación se llegó a la conclusión de crear una estructura dinámica que permitiera ahorrar “los espacios en blanco no usados” (véase en figura 3.14), por lo que, se ideo en base a una celda de la matriz adherida a una lista dinámica la cual permitiera almacenar el camino de un “nodo A” al “nodo B”, el uso de estructuras dinámicas permite hacer uso especifico de memoria RAM, lo que en términos de rendimiento es bien visto, ya que las listas dinámicas con forme es necesario el uso de bloques este los pide.

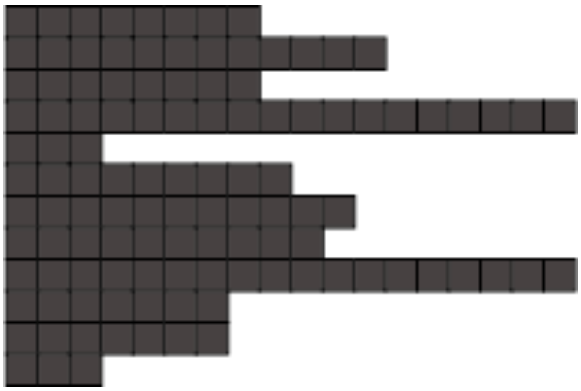


Figura 3.14. Representación del gasto eficiente con una estructura dinámica.

El uso de las matrices podría llegar a afectar el rendimiento del software en el ámbito de memoria RAM, incluso en una época actual o futura es una buena idea el preocuparse por el rendimiento de los insumos, ya que esta medida también ayuda a permitir el acceso a esta tecnología a personas con escasos insumos computacionales.

Para el caso de los algoritmos que resuelven árboles, pasa la misma situación, dado que requieren de un arreglo para guardar los caminos, por lo que, atendiendo a las recomendaciones realizadas en la sección de árboles, una estructura dinámica fue idónea para la creación de estos tipos de algoritmos.

3.8. Diseño de las interfaces gráficas de usuario

En cuanto al diseño de las interfaces gráficas, se procuró diseñar con colores opacos con la finalidad de cuidar la retina de los ojos, ya que se sabe que usar colores muy claros puede llegar a lastimar la visión de las personas en un tiempo prolongado, en cuanto al diseño de las interfaces gráficas se optó por asemejar a las interfaces de un

programa de diseño gráfico, así como de implementar una barra de acceso rápido para facilitar el uso del software al usuario. Estas propuestas tienen como finalidad el hacer crecer la experiencia de usuario de una forma agradable, a continuación, se mostrarán los diseños que se usaron para implementación (véase en figuras 3.15, 3.16, 3.17), cabe aclarar que estos diseños son “diseños base”, en otras palabras, son plantillas que se especializaran dependiendo del trabajo o de la situación:

Figura 3.15. Diseño para la ventana principal.

Figura 3.16. Diseño general para la ventana de resultados en el analizador gráfico.

Figura 3.17. Diseño general para la ventana de resultados en la sección de consola.

Figura 3.18. Diseño para los asistentes de configuración.

Capítulo 4: Implementación del software JGrafos4

Una vez creada la planeación de trabajo, así como de tener planeada la estructura de datos a usar fue posible entrar a la etapa de implementación del software, este capítulo relatara los parámetros del proyecto, el proceso que se llevó a cabo en cuanto a la implementación del software, los problemas que se tuvieron al momento de escribir código, así como la actualización de versiones del software y sus mejoras que obtuvo por el tiempo.

4.1. Parámetros generales del proyecto.

El primer paso fue buscar la versión más reciente de Java, ya que usar una versión más reciente del lenguaje permite tener más beneficios que trabajar con versiones anteriores, además en el caso de Java es recomendable usar versiones LTS (Long Time Support), ya que este tipo de versiones cuentan con soporte a largo plazo, por dichas razones, la versión de Java usada fue Java17 ya que en ese momento era la versión LTS más reciente.

En cuanto al sistema operativo se usó Windows 10 en su última versión, esto fue más por cuestión de estética y comodidad, sin embargo, el código de Java puede ser usado en múltiples plataformas, lo cual no afecta su funcionamiento y permite que el código escrito en una plataforma sea usado en otra, esto por una de las características comentadas con anterioridad sobre Java y de las cuales este lenguaje resulto electo para este trabajo.

En cuanto al hardware que se usó para la implementación de este proyecto, fue un ordenador de escritorio con características de gama media, los cuales se describen a continuación:

- CPU: AMD Ryzen 5 5600 a 3.50GHz como base con 6 núcleos y 12 hilos de ejecución.
- RAM: 32GB DDR4 a 3200MHz.
- Tarjeta de gráficos: Nvidia GT1030 con 2GB VRAM.
- Almacenamiento: Mixto (SSD + HDD).

4.2. Etapa de producción del software JGrafos4.

Una vez que ya se tenía la planeación así como las herramientas de trabajo definidas, se procedió a empezar con la producción del software, en primera instancia creo la estructura de datos base que sirvió para el funcionamiento de todo el software, como se comentó en la sección de Estructura de datos en el capítulo 2, era necesaria una estructura dinámica que fuera flexible para la mayoría de algoritmos, así que se optó por crear una clase específica para dicha estructura de datos, dicha clase tiene como atributos una variable de tipo doble así como una lista dinámica, el funcionamiento

4.2.1. Implementación de la clase NodeMatrix (Estructura de datos base para funcionalidad)

La primera tarea que completar una vez iniciada la etapa de producción fue la creación de la estructura de datos base del software, siguiendo el análisis realizado en la etapa de planeación y descrito en la sección de Estructura de datos en el capítulo 2, era necesaria una estructura dinámica que fuera flexible para la mayoría de los algoritmos,

por lo que se optó crear una clase especifica que contuviera las siguientes características:

- Un bloque de memoria específico para llevar el resultado de una suma o contención de un número, ya que en la mayoría de los algoritmos es necesario guardar un valor actual de una suma o incluso del valor del peso de una arista.
- Un conjunto de bloques que aumenten dependiendo de la demanda de trabajo, en otras palabras, una lista dinámica que contuviera números, ya que es necesario llevar el camino recorrido.
- Un conjunto de métodos que facilitaran el manejo de los atributos de la clase, ya que es necesario realizar validaciones, agregar nuevos datos a las estructuras, actualizar números, etc.

La estrategia usada para satisfacer las características requeridas consistió en declarar atributos los siguientes tipos de datos "atómicos" (véase en figura 4.1):

- Un atributo de tipo doble.
- Un atributo de tipo ArrayList<E>, esta lista es genérica por naturaleza, por lo que se especificó para que fuera compatible con el tipo de dato Entero (Integer).

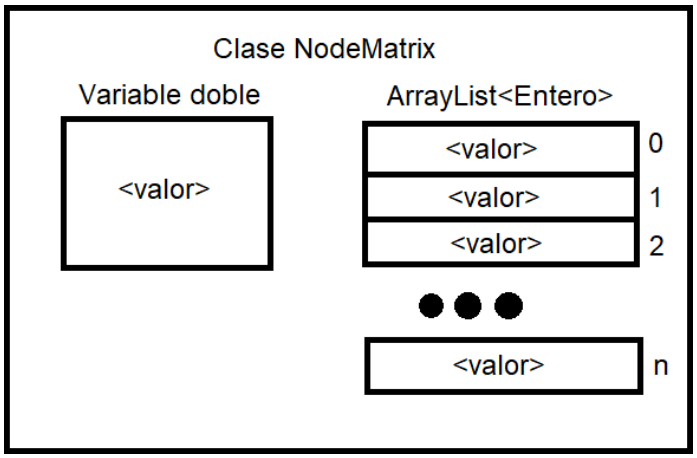


Figura 3.18. Diagrama de la estructura interna de los atributos dentro de la clase NodeMatrix.

Dicha estructura permite la optimización de recursos comentada en la sección de Estructura de datos en el capítulo 3, ya que es una estructura dinámica la cual solo gastara los recursos necesarios bajo la demanda de los algoritmos.

4.2.2. Implementación de los algoritmos de solución para grafos y árboles

En esta etapa del proceso, se hizo la traducción de los algoritmos de seudocódigo a código fuente, sin embargo, al tener un análisis previo sobre cada uno de los algoritmos a implementar facilito la escritura de código, así como encontrar similitudes entre cada uno de los algoritmos, lo que permitió su clasificación en grupos, además retomando lo anteriormente comentado en la sección de Estructura de Herencia del capítulo 3, esto permitió hacer la reutilización de código y de heredar propiedades específicas de un grupo.

4.2.2.1. Clase Algorithm

Como se puede notar en el diagrama de clases, esta clase se encuentra en la parte superior de toda la jerarquía, ya que provee las siguientes características a sus subclases:

- Una interfaz para ejecución en concurrencia.
- Un historial de consola, para observar los procedimientos realizados en algoritmo.
- Un mecanismo de medición de tiempo.

Como se puede observar, esta clase provee características sencillas, sin embargo, la funcionalidad de esta clase no solo se limita a proveer dichas características básicas, sino que aprovechando una característica muy importante de la P.O.O. provee un "grupo de clases" lo cual fue de gran importancia en etapas subsecuentes a esta.

4.2.2.2. Grupo de algoritmos de vector

Este grupo de clases está enfocado a aquellos algoritmos que necesitan de vectores para su funcionamiento, como es el caso del algoritmo de Dijkstra, ya que este necesita de vectores, cabe destacar que esta clase fue influida por los análisis realizados sobre cada algoritmo, por lo que este grupo provee las siguientes características:

- Provee un vector solución aplicando la estructura de datos de NodeMatrix.
- Provee una matriz para el manejo de datos (de tipo doble) para su etapa de funcionamiento.
- Provee un mecanismo de entrega de resultados, sin embargo, el tipo de retorno para este grupo de algoritmos es de un vector de tipo NodeMatrix.

Con base en los análisis realizados previamente y de las características que provee esta clase, se puede realizar herencia a los siguientes algoritmos:

- Dijkstra.
- Prim.
- Kruskal.

4.2.2.3. Grupo de algoritmos de matriz

Este grupo de clases está enfocado para aquellos algoritmos que requieren de matrices para funcionar y además de devolver una matriz como resultado, los análisis anteriormente hechos reflejan las siguientes características necesarias:

- Provee una Matriz de tipo NodeMatrix de n x n elementos que se usara para ejecución del algoritmo.
- Provee un método de limpieza de la matriz, así como de inicialización de la misma.
- Provee un método de obtención de resultados, sin embargo, el tipo de retorno para este grupo es de una matriz de tipo NodeMatrix.

Con base en los análisis previamente realizados y de las características que provee la clase, los algoritmos dentro del grupo fueron los siguientes:

- Floyd
- Warshall

4.2.2.4. Grupo de algoritmos de estructura

Este grupo de clases está enfocado para aquellos algoritmos que basan su ejecución en otra estructura diferente (no especifica), además, no dependen directamente de un array o de una matriz ya que su funcionamiento es diferente, una de sus características más representativas es el uso de la recursividad, sin embargo, no todos la usan, este grupo provee las siguientes características:

- Uso de una estructura no convencional (de tipo Node, se explicará más adelante dicha estructura).
- Uso de una estructura auxiliar, suplente a la estructura de NodeMatrix, pero con un funcionamiento muy similar.
- Mecanismo de recursividad.

Con base en los análisis previamente realizados y de las características que provee la clase, los algoritmos dentro del grupo fueron los siguientes:

- Pre-Orden.
- In-Orden.
- Post-Orden.
- Huffman.
- BPF.

4.2.2.5. Implementación de las clases contenedoras de los algoritmos

Una vez delimitados los grupos de algoritmos, fue más sencillo y rápido la traducción de seudocódigo a código fuente, cada una de las clases heredo de la super clase correspondiente para heredar las características propias de cada superclase, además, ayudo con la estandarización de las salidas, ya que este fue un punto muy importante a la hora de pintar las respuestas en el canvas correspondiente, además, se procuró dividir el código por funciones, esta acción ayudo mucho a la hora de probar cada módulo para hallar con más facilidad los errores, cabe destacar que conforme se terminaba de crear una clase de un algoritmo, se probaba con corridas de escritorio (problemas ya solucionados hechos a mano) y se comparaban las respuestas del proceso con las respuestas de las corridas de escritorio, en este punto solo había 2 caminos, si las respuestas fallaban, la clase se inspeccionaba y se arreglaba para su posterior prueba, pero si la clase se ejecutaba satisfactoriamente, esta se daba por concluida, estos procesos validaban al código después de 10 pruebas seguidas y sin error.

4.2.3. Implementación de la GUI (Interfaz gráfica de usuario)

En cuanto a la implementación de la interfaz gráfica de usuario, se usó las librerías nativas de javax. swing ya que dichas librerías son más robustas que las librerías de awt de java. En cuanto a la estructura de trabajo, se eligió trabajar por componente, esteestilo de trabajo fue inspirado por el framework de angular, en donde toda la interfaz esta creada por componentes, a continuación, se describirá el proceso de creación y su finalidad de los componentes.

4.2.3.1. Implementación de componentes

En cuanto a los componentes de la aplicación se pueden dividir en 2 grupos, componentes nativos, y los componentes dedicados

4.2.3.1.1. Componentes nativos

Este grupo de componentes se refiere a aquellos componentes que no requirieron una extensión de ellos, en otras palabras, solo se usaron como tal, su uso solo se limitó a entrada o salida de datos o en algunas ocasiones como contenedor para otros componentes, los componentes nativos usados fueron los siguientes:

- JTextArea (área de texto).
- JTextField (recuadro de texto).
- JComboBox (lista desplegable).
- JButton (botón simple).

- JCheckBox (recuadro de validación).
- JPanel (panel).
- JOptionPane (ventana emergente para brindar información).
- JLabel (letreros).
- Layouts (gestores de componentes).

El único cambio que se implementó en este grupo fue estético, ya que sólo se cambió fuente, color de fuente, tamaño, fondo y localización.

4.2.3.1.2. Componentes dedicados

Este grupo de componentes se refiere a aquellos componentes que extienden de componentes nativos para hacer componentes específicos para una determinada tarea, su uso depende de la tarea específica para la cual fueron creados, dichos componentes son los siguientes:

- Menú, extiende del componente JMenuBar, es una barra de menú que se colocó en la parte superior de la aplicación antes de la barra de título de la aplicación, su funcionalidad es la de ofrecer un conjunto de opciones.
- Tab, extiende del componente JTabbedPane, es un conjunto de pestañas, permite el uso de varios espacios de trabajo, su funcionalidad es la de poder tener varios espacios de trabajo abiertos al mismo tiempo.
- MenuToolbar, extiende de JToolbar, es un panel de pequeñas dimensiones que cuya única funcionalidad es la de contener botones o listas desplegables,(existen 4, sin embargo, todas son lo mismo).
- OptionRender, extiende de JLabel e implementa la interface ListCellRenderer, su única función es estética, solo es para mostrar un texto y un icono juntos.
- WorkArea, extiende de JPanel, este es un componente muy importante ya que permite dibujar sobre él, pero en pocas palabras es un canvas.
- ResultViewer, extiende de WorkArea, su función es la de mostrar los resultados gráficos del procesamiento de los grafos o árboles.
- TabIcon, extiende de JPanel, su función es la de mostrar un icono.
- TabName, extiende de JPanel, su función es la de renderizar el título de la pestaña con su botón de cierre.
- TopBar, extiende de JPanel, su función es la de ser un contenedor.

4.2.3.1.3. Implementación de elementos del canvas

El canvas es uno de los elementos más importantes dentro del programa JGrafos4, ya que es aquel que permite dibujar e interpretar la información dada por el usuario, ante ello es necesario explicar sus elementos para entender su funcionamiento antes de llegar a su sección, el canvas requiere de 2 estructuras básicas, las líneas o aristas y los nodos.

4.2.3.1.3.1. Nodo

Esta clase fue diseñada para contener información útil relacionada a la ubicación de un nodo en el espacio de trabajo (canvas) así como información sobre sus conexiones con otros nodos (para los algoritmos que requieran recursividad), esta estructura puede ser manejada tanto en arboles como en grafos, además contiene métodos que facilitan su manipulación, así como métodos que afectan su representación en el espacio de trabajo.

4.2.3.1.3.2. Línea

Esta clase fue creada para representar las conexiones entre nodos, además de contener información sobre dicha conexión como es el caso de los pesos, además, provee información para módulos externos.

4.2.3.2. Implementación de un canvas Abstracto

El componente WorkArea (el cual en esta sección será referido como clase) es una de las clases y estructuras más importantes del software, ya que su función no solo se limita a pintar cosas en su lienzo, sino que provee información sobre el tipo de estructura(grafo o árbol), además de indicar información sobre su direccionalidad, el número de nodos, el número de líneas, etc. La información que contiene esta clase es vital para el funcionamiento correcto del software, sin embargo, es una clase abstracta por lo que no puede ser instanciada directamente, aunque una ventaja es que crea de forma indirecta un grupo de estructuras que permite englobarlas como un tipo de dato.

4.2.3.3. Implementación de canvas dedicados

Dado que los grafos se diferencian de los árboles por al menos tener un ciclo, el a ver procedido con un solo espacio de trabajo hubiera afectado la producción del software, ya que sería necesaria un conjunto de validaciones en tiempo real para validar si la estructura escrita fuera un grafo o fuera un árbol, ante dicha situación se decidió manejar a WorkArea como estructura, y dividir dicha estructura en 2, una dedicada para grafos y otra dedicada para árboles, además, dicha división también permitió usar la mismo para mostrar las soluciones y evitar escribir más código similar.

4.2.3.3.1. Grafo

Esta clase hereda directamente de WorkArea, sin embargo, esta se diferencia por tener una forma de crear el esquema, el proceso para crear un grafo es la siguiente:

1. Hacer doble clic izquierdo en cualquier parte del espacio de trabajo.
2. Repetir el paso 1 las veces que se deseé, pero en diferentes partes.
3. Para crear una línea que conecte a 2 nodos, es necesario presionar con el botón derecho del mouse un nodo y arrastrar sin soltar dicho botón hasta llegar al nodo deseado, después estando en el nodo, soltar el botón y listo se creara la línea.

Otro de los beneficios de tener separado al grafo es que se puede identificar de una mejor forma usando Java reflection.

4.2.3.3.2. Árboles

Esta clase hereda directamente de WorkArea, sin embargo, esta se diferencia por tener una forma de crear el esquema, el proceso para crear un Árbol es la siguiente:

1. Hacer doble clic izquierdo en cualquier parte del espacio de trabajo, preferentemente en el centro superior.
2. Después presionar el botón izquierdo del mouse sin soltarlo en el primer nodo, después dirigirse a otra coordenada y soltar el botón, con esta acción se creará una línea y un nodo nuevo.
3. Para hacer más nodos repetir el paso 2.

Otro de los beneficios de tener separado al árbol es que se puede identificar de una mejor forma usando Java reflection.

4.2.3.3.3. Resultados

Esta clase solo tiene como única finalidad el mostrar los resultados de los cálculos de los nodos, pero en formato de nodo o de árbol según sea el caso.

4.2.3.4. Implementación de las ventanas de trabajo

Al igual que otras aplicaciones de escritorio, este software necesitó de ventanas para poder interactuar con el usuario, ante esa situación, se aprovechó de una clase de la librería de javax. swing, la cual permite crear una ventana nativa del sistema operativo en uso llamada JFrame, dicha clase contiene métodos que permiten su administración, así como la manipulación de sus propiedades.

4.2.3.5. Implementación de la ventana principal

La clase JFramework hereda directamente de la clase JFrame heredando todas sus características, esta ventana es la principal de proyecto, se siguieron los diseños ya antes mostrados con el fin de conseguir una interfaz agradable con el usuario y además que le permitiera mostrar ciertas acciones a simple vista, cabe aclarar que esta clase solo es la vista, ya que su controlador se encuentra en la clase FrameController en donde se administran todas las acciones de esa ventana en específico incluyendo las acciones de los componentes en dicha ventana.

4.2.3.6. Implementación de la ventana de resultados

Esta ventana también hereda directamente de la clase JFrame, solo que a diferencia de la clase de JFramework esta clase es genérica, pero no porque tenga un template como otras clases genéricas, sino que esta clase puede ser usada por todos los algoritmos de solución de grafos y árboles, cuenta con ciertos mecanismos que le permiten adaptarse a los algoritmos que se ejecuten, la funcionalidad de esta ventana es únicamente mostrar los resultados de los algoritmos ejecutados en forma de grafo o árbol y en su forma de texto (descripción de los pasos que se siguieron para conseguir el resultado).

4.2.3.7. Implementación de ventanas secundarias

Este tipo de ventanas extienden de la clase JDialog, debido a que no se requiere de un frame como tal, pero una de las ventajas de trabajar bajo esta clase es que bloqueara las acciones en el software hasta que se complete o se cancele el proceso, este tipo de ventanas fueron usadas muy pocas veces y se trataron de evitar en lo posible.

4.2.4. Implementación de una estructura de comunicación entre componentes

Una vez teniendo todos los componentes creados, había la necesidad de comunicarlos, ya que algunos elementos requerían saber de los estados de los componentes para poder pasar a una ejecución o incluso su trabajo dependía de ello, por lo que se incluyó un conjunto de variables de ejecución las cuales solo están activas durante la ejecución del software, estas variables sirven como puentes de comunicación entre componentes con la finalidad de comunicar su estatus mediante una key especifica, cabe aclarar que para evitar escrituras no deseadas se usaron semáforos para evitar problemas con estas zonas críticas, esta estructura permitió un mejor desempeño en el software.

4.2.5. Implementación de un sistema de trabajo para módulos externos

Antes de seguir en esta sección, cabe aclarar lo que significaría un módulo externo, este se definiría como un módulo o programa que no se encuentre dentro del sistema como tal, es decir, este módulo no fue integrado al momento de implementación del software, pero que podría ser implementado como una extensión, ante esta definición podemos visualizar que hay ciertas complicaciones dado que es un módulo no nativo dentro de este software, es decir que se tendrían 2 programas ejecutándose de forma simultánea (véase en figura 4.2).

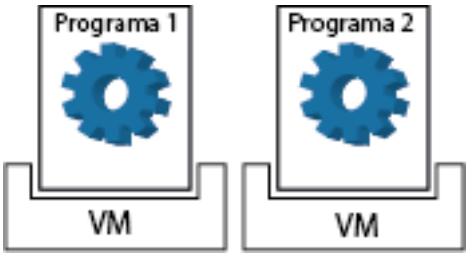


Figura 4.2. Diagrama que representa la ejecución de 2 programas.

Como se puede apreciar en la imagen (véase en la figura 4.2) los programas se ven como 2 bloques independientes que no comparten nada en común, por lo que se analizó lo siguiente:

- Al no ser un módulo nativo, no conoce la estructura que se maneja.
- Al ser un proceso independiente se debe de encontrar la forma de ejecutarlo.
- Se debe de contar con una forma de comunicación para compartir datos.
- Al ser un módulo externo, no se conoce en que lenguaje está escrito, por lo que se tendría que pensar en una forma de poder compartir datos.

De primera instancia se aplicaron ciertas restricciones para poder mitigar dicha situación sobre el último punto descrito, dicha estrategia de mitigación consistió en limitar el lenguaje de programación admitido, el cual tiene que ser java para que puede hacer uso de la JVM (java virtual machine), por lo que el último punto se podría encontrar bajo control, sin embargo, al hacer un análisis más exhaustivo, se dedujo que el primer punto no puede mitigarse completamente, ya que al ser un módulo escrito por otra persona, la lógica de trabajo puede ser muy diferente a la establecida, esto también incluye a las estructuras usadas, pero, la forma de mitigar esta situación es en sus puntos de entrada y de salida de datos, ya que se puede controlar la forma de envió y de recepción de datos, por lo que el primer punto podría visualizarse como bajo control, eso solo deja 2 puntos a mitigar, el lanzamiento del proceso y su forma de compartir datos, ante esta situación, se ideo una forma de poder dar solución a este problema (véase en figura 4.3), se trata de crear un mecanismo interno en el software que permita lanzar un proceso (de Java) y que permita compartir datos en tiempo real, y que además permita retroalimentar al software JGrafos4 con los resultados obtenidos.

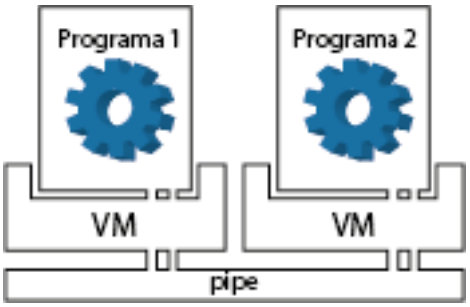


Figura 4.3. Diagrama de la propuesta planteada para la ejecución de módulos externos.

La idea estaba lista, la única incógnita en ese momento fue sobre el método a seguir para llevar a cabo dicha idea, por lo que se realizó una investigación para ver si java tendría ciertos mecanismos para poder llevar a cabo la propuesta anteriormente comentada, aunque fue difícil se encontró cierta documentación al respecto, sin embargo, dichas soluciones se pudieron abarcar en los siguientes puntos:

- Uso de la API de Java Reflection.

- Uso del mecanismo de Process y ProcessBuilder junto con los mecanismos de entrada y salida de java.
- Uso de Java RMI.

4.2.5.1. Java Reflection

Java Reflection es una librería que Java ofrece con la finalidad de poder ver información sobre una determinada clase, con dicha librería se pueden ver los métodos (públicos, privados y protegidos), así como sus atributos, constantes, constructores, etc., en pocas palabras es como un escáner para clases, claro que no se puede ver internamente los métodos, hasta ahí Java reflection tiene un límite, en cuanto a la implementación de la idea comentada, el plan a seguir era de cargar archivos ".class" por medio de un class loader (cargador de clases) y mediante los mecanismos de Java reflection poder lanzar una instancia de la clase y ejecutarla, sin embargo, al probar este camino, se vio un poco más complejo de los que se intuye, por lo que si un proceso se vuelve complejo de entender, existe una mayor probabilidad de fallo, lo cual no es lo ideal, además, otro punto que derribo dicha idea, fue en cuanto si la clase dependiera de otras, lo cual convirtió esta idea a una forma más compleja, lo cual termino por descartar esta solución.

4.2.5.2. Process y ProcessBuilder

Java tiene mecanismos para poder ejecutar otros programas desde el programa actual, existen 2 formas, la primera (que es insegura según la documentación leída) es por medio de System.getRuntime() lo cual permite usar el método exec() el cual admite como parámetros el nombre del programa más los argumentos para el programa y la segunda forma (un poco más segura que la anterior) que usa la clase de ProcessBuilder que admite una serie de parámetros los cuales son el nombre del programa más los argumentos para después aplicar el método start() el cual devuelve un objeto de tipo Process, el plan a seguir con esta solución era el aplicar el método de ProcessBuilder y Process, mandando como parámetros unas cadenas con la información solicitada, redireccionar las entradas y salidas de datos al software y extraer sus consolas para su posterior análisis, sin embargo, esta idea fue descartada, después de analizar la entrega de parámetros, dado a que sería un poco complejo el mandar los datos requeridos, así como de realizar estructura internas más complejas y por último fue un problema con el buffer de entrada y de salida, ya que se observó que cuando hay muchos datos, estos pueden llenar el buffer y por consecuencia perder los datos, sin embargo, con excepción de la forma anterior, lo único rescatable de esta propuesta fue la forma de lanzar los procesos.

4.2.5.3. RMI

Java RMI es un mecanismo de Java que permite realizar computo distribuido, para usar este mecanismo de Java es necesario realizar una serie de interfaces que extiendan de java.rmi.remote y usar dichas interfaces en las terminales (extremos) así como definir un puerto y activar rmi registry, el cual es un registro de nombres que permite compartir objetos de forma remota y acceder a ellos desde otras terminales, el plan a seguir con esta propuesta, era de crear una serie de interfaces en un archivo ".jar" externo, implementarlo como una librería tanto en el JGrafos4 como en el módulo externo y compartir los datos en tiempo real, esta propuesta se probó en un proyecto sencillo y si logro los resultados esperados, por lo que después se creó el archivo JGInterfaces.jar, el cual contiene un paquete llamado interfaces en el cual se encuentran 2 interfaces, un interface llamada Buffer el cual contiene la definición de los métodos para enviar consolas y los resultados, la segunda interface se llama StructureData, el cual contiene los métodos para mandar los datos del software JGrafos4 al módulo externo, dicho

módulo de interfaces remotas se implantaron en el software JGrafos4, cabe aclarar que fue una muy buena idea realizar esta propuesta ya que permite una evolución a futuro, es decir que las interfaces pueden ser actualizadas para dar soporte a más propuestas futuras.

4.2.5.4. Conversor de resultados

Esta sección está muy ligada a la sección inmediata anterior, ya que la clase que implementa la interfaz Buffer fue ocupada para descifrar los resultados y hacer una conversión de cadena a una estructura manejada en el JGrafos4 y así tener salidas de forma estándar.

4.2.6. Lanzador de procesos

Retomando las acciones realizadas en la sección de Process y ProcessBuilder, se implementó la forma de ejecución de un programa externo, usando el segundo mecanismo, sin embargo, esta clase no solo ejecuta módulos externos, también ejecuta los módulos nativos del software, por lo que se implementó otro mecanismo que diferenciara de un módulo nativo de un módulo externo para conseguir las ejecuciones, finalmente esta clase contiene métodos que entregan un respuesta estándar para ser interpretada por el componente de ResultViewer sin importar el tipo de modulo ejecutado.

Capítulo 5: Pruebas y resultados del JGrafos4

Una vez que el software fue funcional, se procedió a hacer una serie de pruebas para demostrar y verificar su funcionalidad con ejemplos concretos, las pruebas fueron diseñadas para experimentar su eficiencia en ciertos puntos críticos en donde los módulos arrojaban altas probabilidades de fallo, además se trató de hacer pruebas en donde se pudiera demostrar su efectividad con las estructuras manejadas, además, cabe recalcar que estas pruebas se realizaron en un sistema operativo Windows 11, los puntos de enfoque de las pruebas fueron los siguientes:

- En cuanto a su estructura (gráfica):
  - Grafos dirigidos.
  - Grafos no dirigidos.
  - Árboles.
- En cuanto al tipo de agrupación (estructural) en el software:
  - Algoritmos de vector.
  - Algoritmos de matriz.
  - Algoritmos de estructura.
- En cuanto a funcionalidades generales:
  - Guardado y apertura de archivos
  - Exportaciones.
  - Cambio de lenguaje.
- En cuanto al comportamiento del software en funcionamiento:
  - Consumo en estado inactivo (no uso o segundo plano).
  - Consumo en estado de uso (en primer plano o en uso).

Para satisfacer los puntos descritos, las pruebas se organizaron en 3 grupos, el primer conjunto de pruebas fue sobre las funcionalidades generales, es decir que las pruebas de este segmento tuvieron como fin el demostrar que las funcionalidades básicas trabajan adecuadamente, el segundo conjunto de pruebas se enfocó a las funcionalidades de los módulos nativos y externos, en otras palabras, se refieren al conjunto de pruebas cuya finalidad fue mostrar que los módulos reaccionan adecuadamente (pruebas de los algoritmos elegidos) y el tercer conjunto de pruebas se refieren a la forma de proceder del software en cuanto a su comportamiento en 2 fases del programa (inactivo o en uso).

5.1. Resultados de las pruebas sobre funcionalidades generales

Este segmento de las pruebas realizadas se refiere a la verificación del correcto funcionamiento del software en cuanto a características básicas tales como las siguientes:

- Función de apertura de archivos de proyecto.
- Función de guardado de proyecto.
- Función de exportación de imágenes.
- Función de exportación de consolas.
- Función de cambio de idiomas.
- Función de creación de nuevas áreas de trabajos (nuevos canvas).
- Función de asistente de creación del proyecto.
- Función de generación de gráficos.
- Función de agregación de módulos externos.

5.1.1. Función apertura de archivos

Al probar esta funcionalidad fue necesario crear un proyecto previamente para poderlo cargar como se muestra en la imagen (véase en figura 5.1)

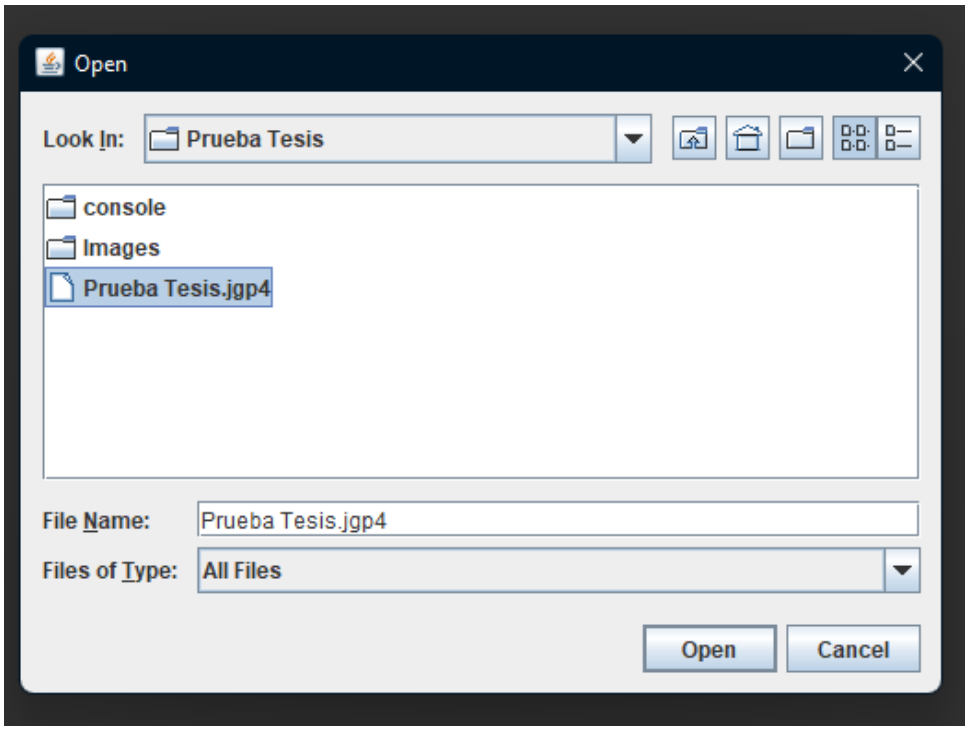


Figura 5.1. Imagen de la prueba de la funcionalidad de apertura de archivos.

En cuanto a su comportamiento, se esperaba que permitiera desglosar en las pestañas de trabajo el proyecto guardado tal y como se mando a guardar, el resultado fue el esperado.

5.1.2. Función de guardado de proyecto

Para probar dicha función, fue necesario crear un proyecto y se realizó 2 grafos y un árbol para después proceder a su guardado.

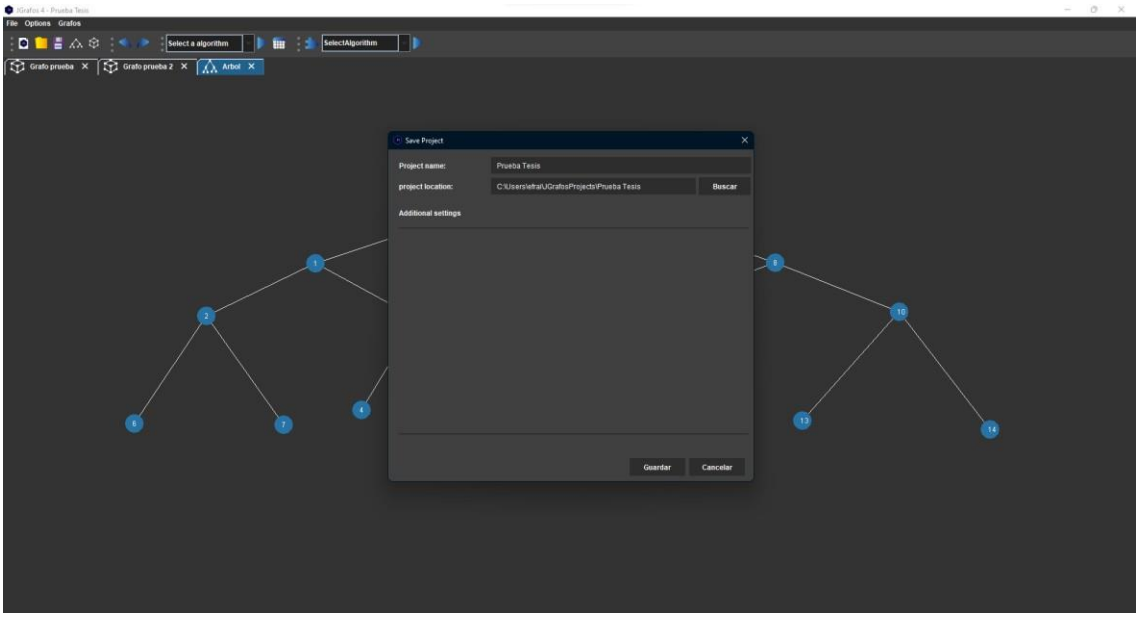


Figura 5.2. Prueba del software JGrafos 4 para la funcionalidad de guardado.

En cuanto al comportamiento de guardado, se esperaba que el software guardara las pestañas tal y como se habían mandado a escribir en disco, además, que creará una carpeta de proyecto para mantener una buena organización.

Después se procedió a verificar si el archivo fue guardado en la localización en donde se configuro en un inicio en el asistente de configuración del proyecto, como se puede apreciar en la imagen (véase en figura 5.3) el comportamiento esperado fue el realizado.

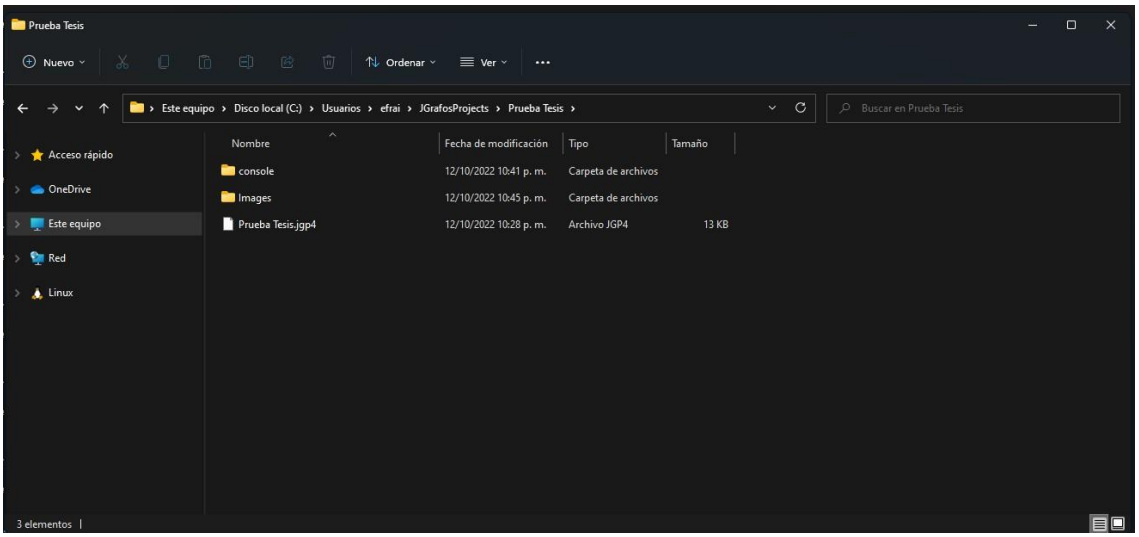


Figura 5.3. Prueba del funcionamiento de guardado del proyecto.

5.1.3. Función de exportación de imágenes

Para probar el funcionamiento de exportación de imágenes, fue necesario el ejecutar un algoritmo para poder generar los resultados y poder acceder a esa opción.

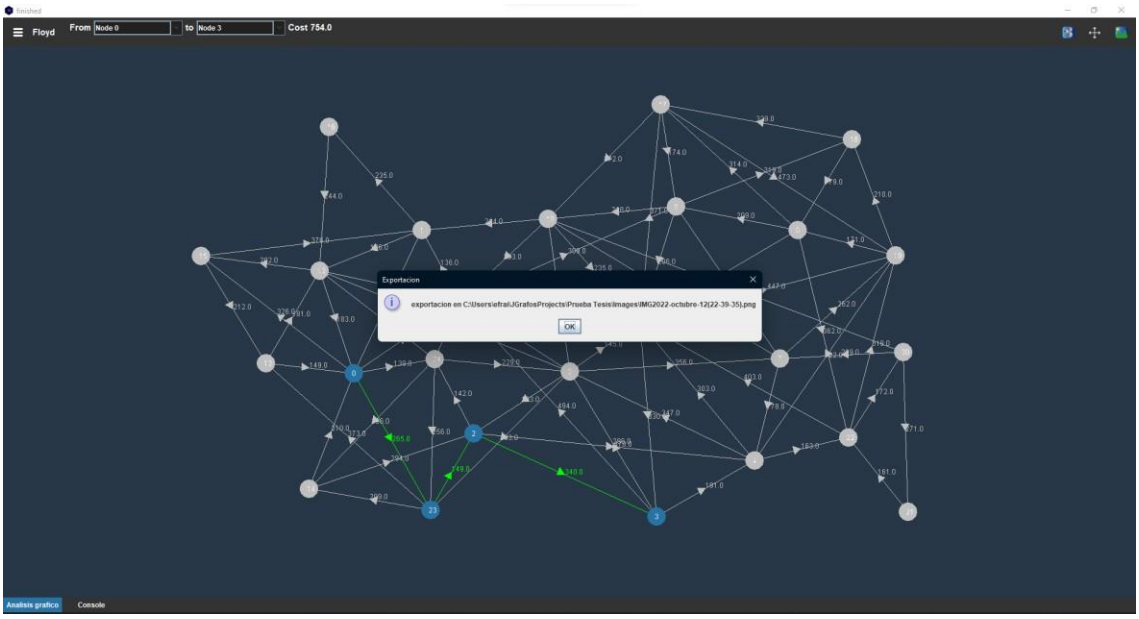


Figura 5.4. Prueba de la función de exportación de imágenes en la ventana de análisis usando como prueba el algoritmo de Floyd(nativo).

En cuanto al comportamiento esperado en este módulo, el software debía de realizar una captura de pantalla al canvas de resultados de la ventana de ejecución de algoritmos, para después realizar su renderizado correspondiente y así poder guardar la imagen en la carpeta de proyecto asignada.

Después de procedió a buscar la imagen en la localización anteriormente configurada en el asistente de configuración del proyecto, como se puede ver en la imagen (véase en figura 5.5), el resultado fue el esperado.

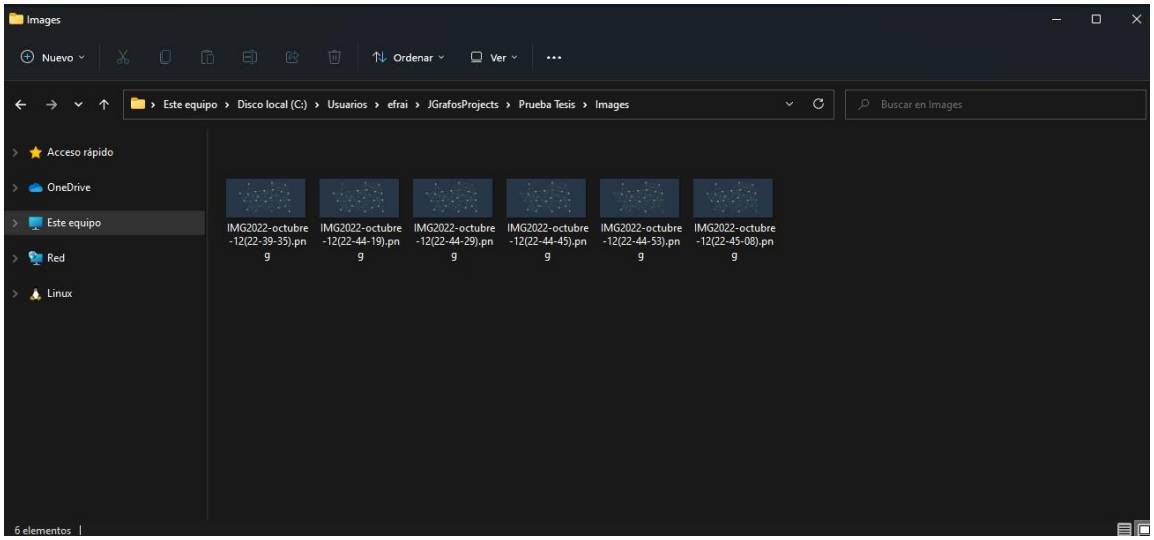


Figura 5.5. Prueba del funcionamiento correcto del proceso de exportación de imágenes.

5.1.4. Funcionamiento de exportación de consolas

Para poder probar el funcionamiento de exportación de consolas de igual forma que con la exportación de imágenes, se tuvo que contar con una ejecución de un algoritmo cualquiera para poder ver los resultados de la consola y poder acceder a esta opción.

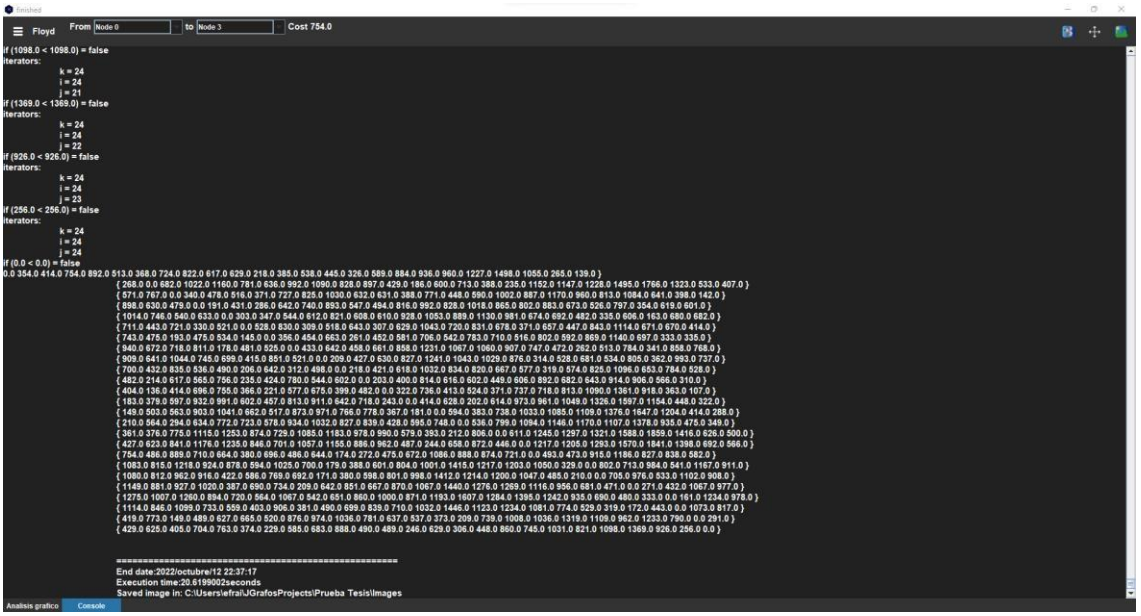


Figura 5.6. Prueba del funcionamiento de consola (descripción del proceso interno).

En cuanto al comportamiento esperado, el software debía guardar todo el contenido de la consola dentro del apartado de consola de la ventana de resultados, además, guardarla en el lugar especificado en el asistente de configuración del proyecto bajo el formato de texto.

Después de realizar el proceso, se procedió a verificar si la consola fue exportada en formato “txt” en la localización configurada en el asistente de configuración del proyecto,

como se puede observar en la imagen (véase en figura 5.7), los resultados fueron los esperados.

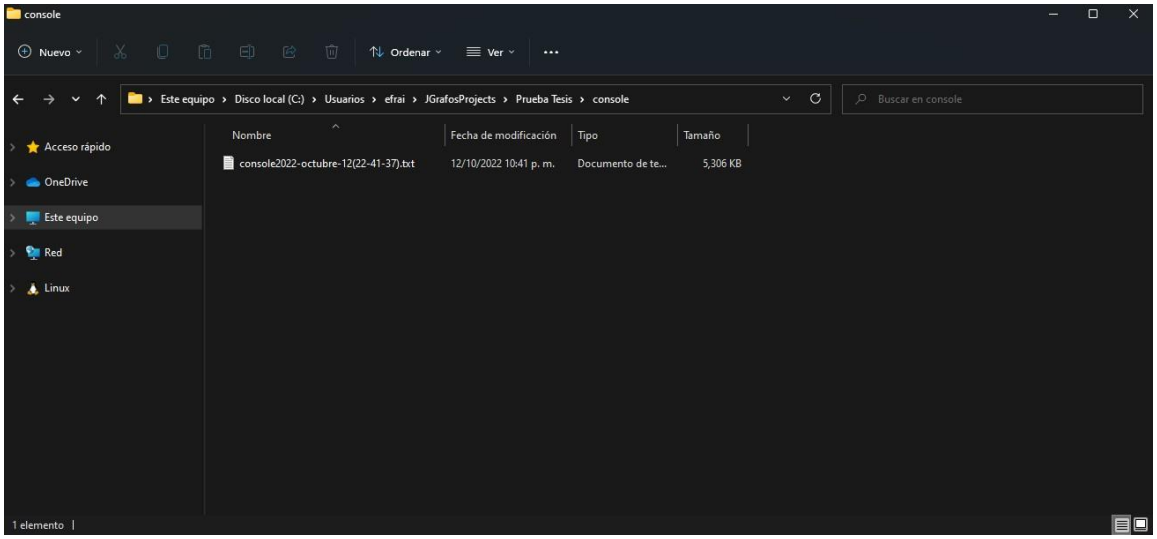


Figura 5.7. Prueba del funcionamiento de exportación de consola en la carpeta contenedora.

5.1.5. Función de creación de nuevas áreas de trabajos (Pestañas de trabajo)

Para realizar la prueba de esta funcionalidad del software es prerrequisito el contar con un proyecto abierto (generar un nuevo proyecto o abrir un proyecto previo), se probaron los 2 tipos de áreas de trabajo (grafo y árbol como se muestra en la figura).

En cuanto al comportamiento esperado, el software tenía que ser capaz de poder abrir múltiples “pestañas de trabajo” a petición del usuario, como se puede observar en la imagen (véase en figura 5.8), el resultado esperado se cumplió.

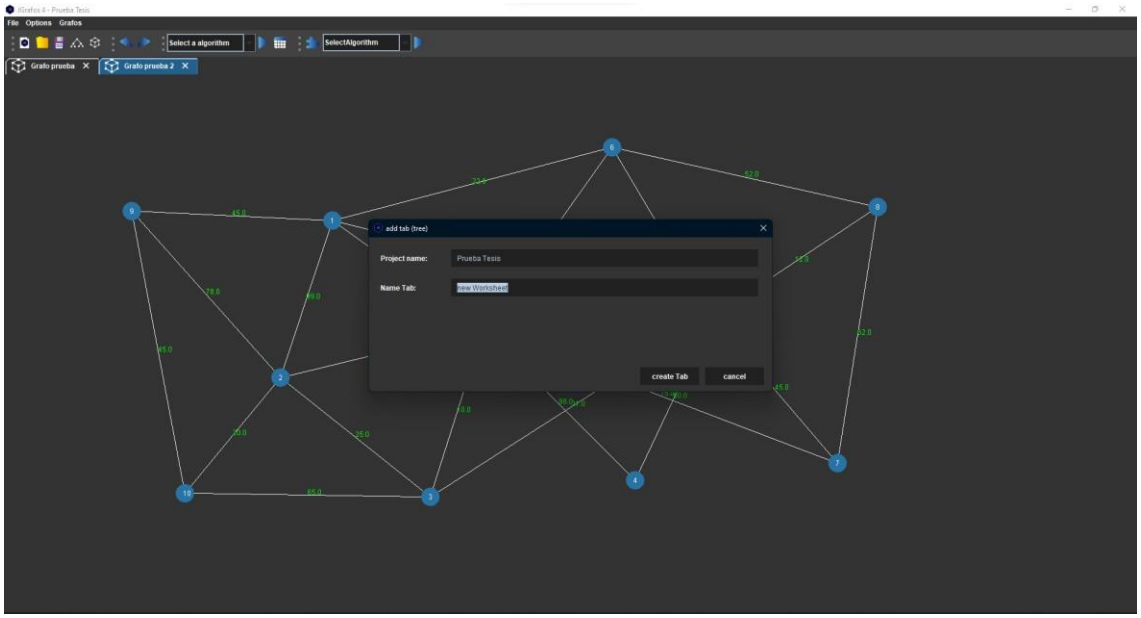


Figura 5.8. Creación de una nueva pestaña en el software JGrafos 4.

5.1.6. Función de asistente de creación del proyecto.

Esta prueba solo tiene como precondición el tener abierto el software JGrafos 4, esta prueba se enfocó a la funcionalidad del asistente de creación de proyecto, ya que este asistente permite la definición de rutas para exportaciones de archivos y de proyectos.

En cuanto a su comportamiento esperado, el software debía ser capaz de abrir el asistente de configuración del proyecto una vez el usuario lo mandara a abrir mediante la instrucción de “nuevo” en el menú o por el atajo de teclas (CTRL + N) y una vez abriendo el asistente, esté debía de ser capaz de capturar las rutas y configuraciones primordiales para el proyecto, como se puede observar en la imagen (véase en figura 5.9), el comportamiento del asistente fue el esperado.

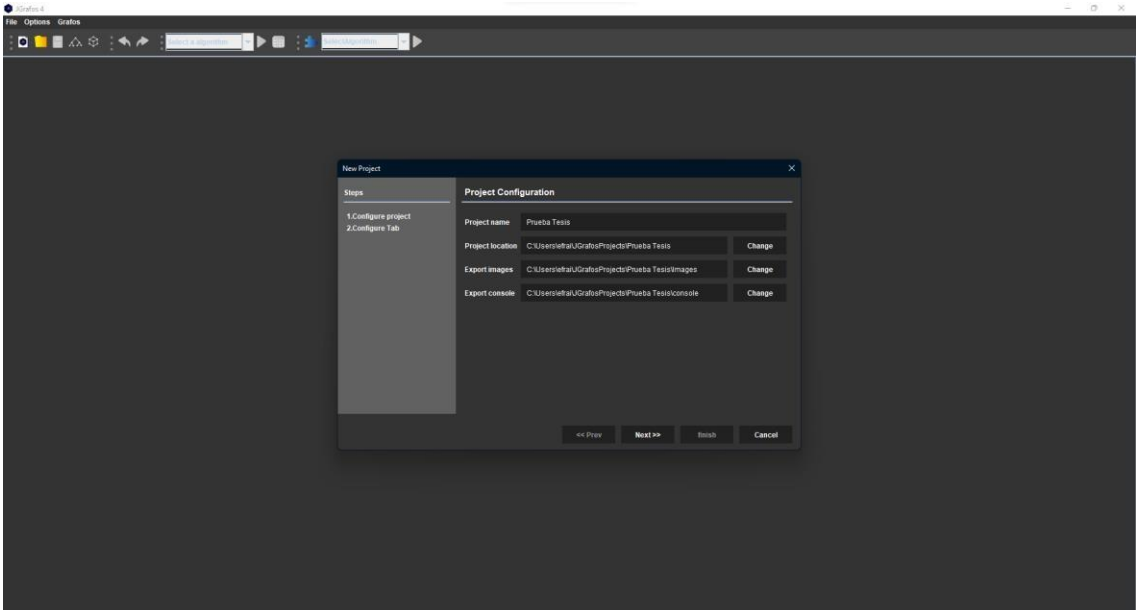


Figura 5.9. Asistente de configuración del proyecto.

5.1.7. Función de generación de gráficos (creación de nodos y aristas)

Para testear esta funcionalidad es necesario tener la precondición de tener un área de trabajo de un tipo, es decir un grafo o un árbol, después de ello se procedió a realizar las pruebas correspondientes.

En cuanto a su comportamiento esperado, este debía de responder mediante los eventos del ratón invocados por el usuario, el software tenia que identificar el tipo de evento del teclado (clic derecho o izquierdo, arrastre del ratón, etc.), con la finalidad de poder renderizar la opción indicada, ya que, hay que recordar que el software usa 2 estructura, los grafos y los árboles, además, otro de los comportamientos esperados en esta funcionalidad, era ver las animaciones de las aristas al momento de unir 2 nodos, se esperaba fluides en dichas animaciones.

Como se puede observar en la imagen (véase en la figuras 5.10 y 5.11), los comportamientos esperados para esta funcionalidad fueron los esperados, así como los resultados de renderizados vistos.

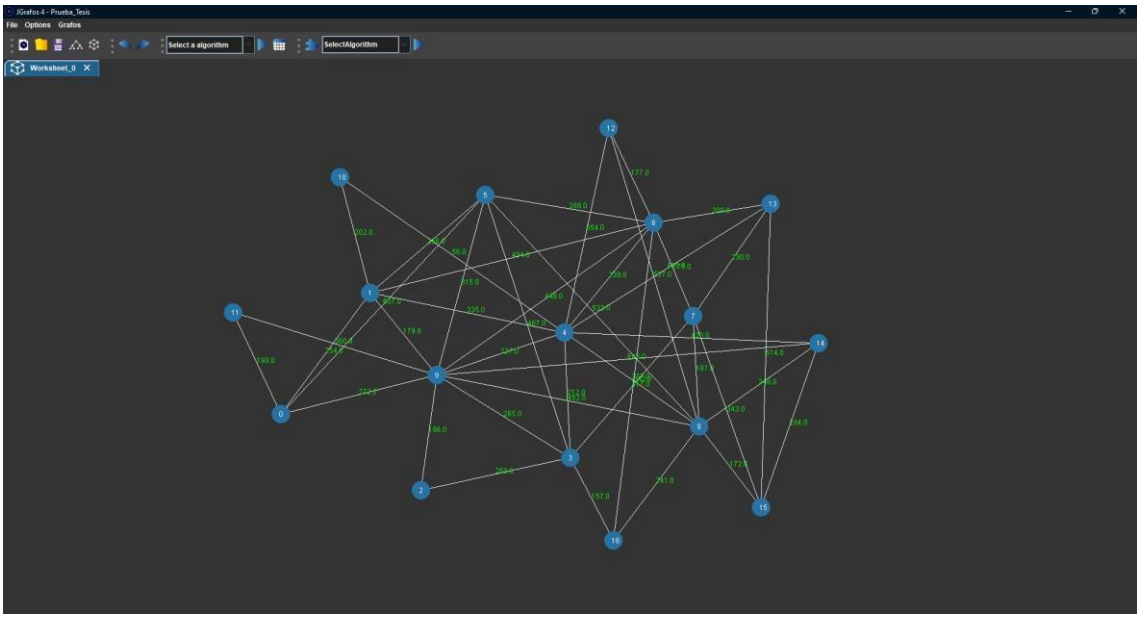


Figura 5.10. generación de un grafo en el software JGrafos4.

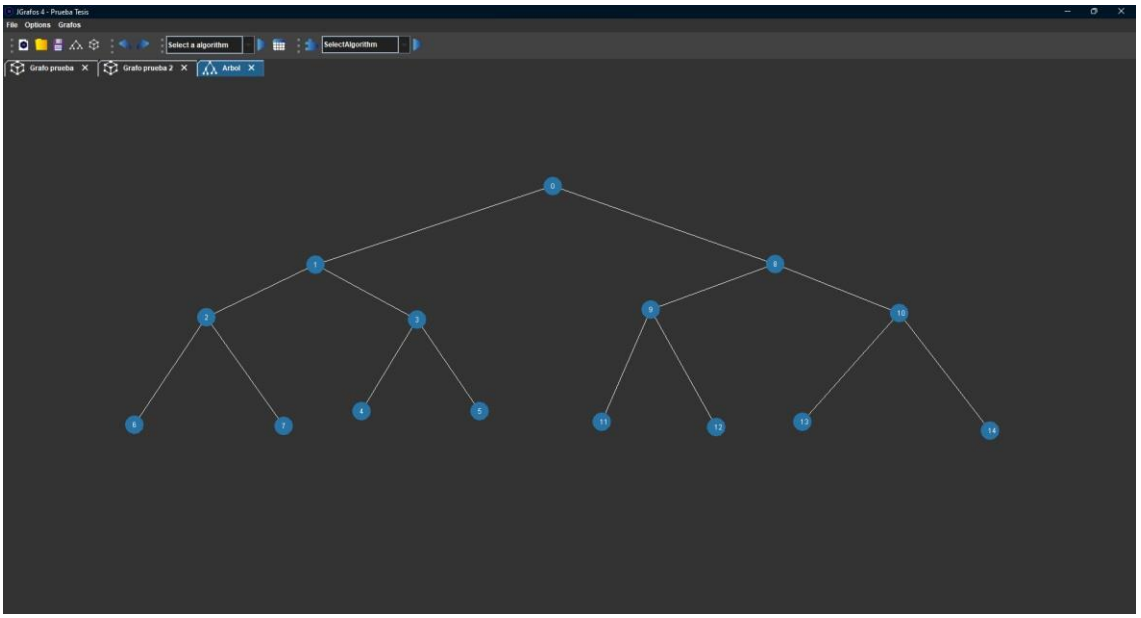


Figura 5.11. Generación de un árbol en el software JGrafos4.

5.1.8. Función de agregación de módulos externos.

La finalidad de esta prueba fue la de verificar que el proceso de instalación de módulos externos se realiza de forma satisfactoria con la ayuda del asistente de instalación de módulos externos.

En cuanto al comportamiento esperado, el software debía mostrar una ventana gráfica que permitiera seleccionar el módulo a implementar y tambien lo anexara en las configuraciones, como se puede ver en las imágenes (véase las figuras 5.12 y 5.13), los comportamientos fueron los esperados.

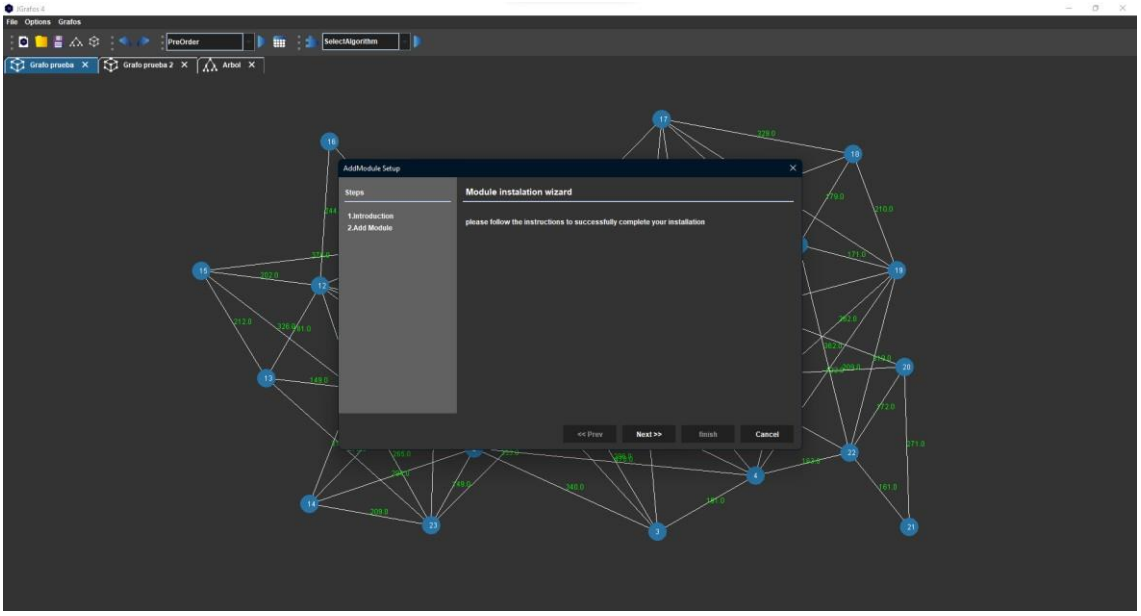


Figura 5.12. Asistente de instalación de módulos externos.

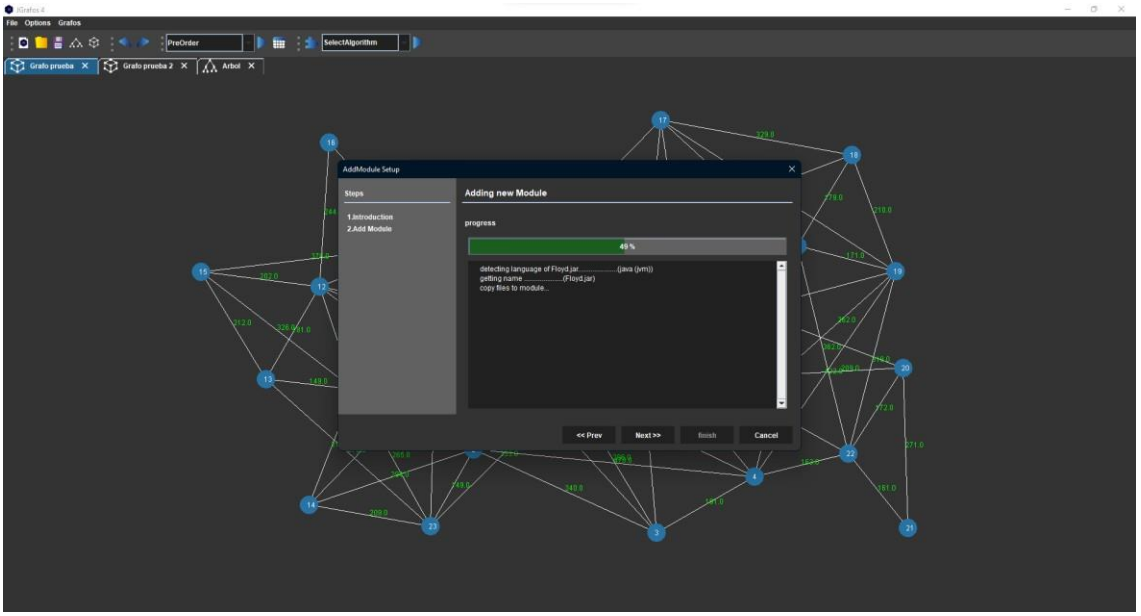


Figura 5.13. Instalación del módulo externo con ayuda del asistente de instalación.

5.1.9. Función de cambio de idiomas.

Esta prueba tubo como finalidad el verificar que el módulo de administración de idiomas funcionada adecuadamente.

En cuanto a su comportamiento esperado, el software debía de mostrar una ventana emergente una vez se invocará mediante el menú de configuraciones, para después mostrar un menú desplegable, en dónde se mostrarían los idiomas disponibles para después seleccionar uno y así grabar los cambios para su próxima apertura delsoftware, como se puede observar en las imágenes (véase las figuras 5.14 y 5.15), el comportamiento fue el esperado al cambiar del idioma inglés al idioma en español.

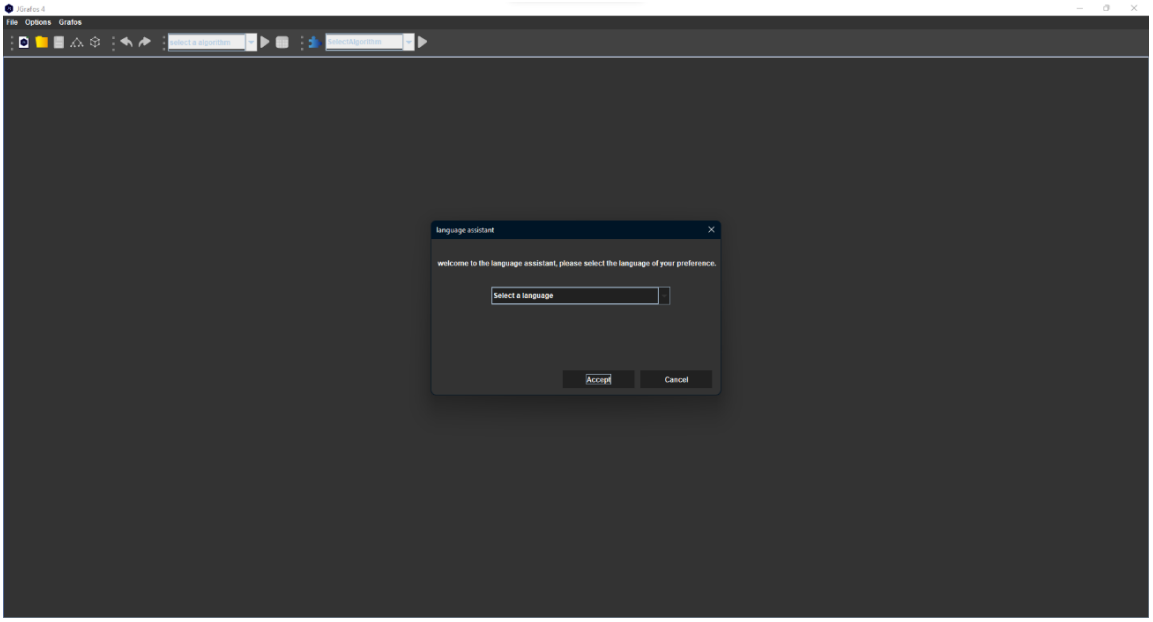


Figura 5.14. Cambio de idioma.

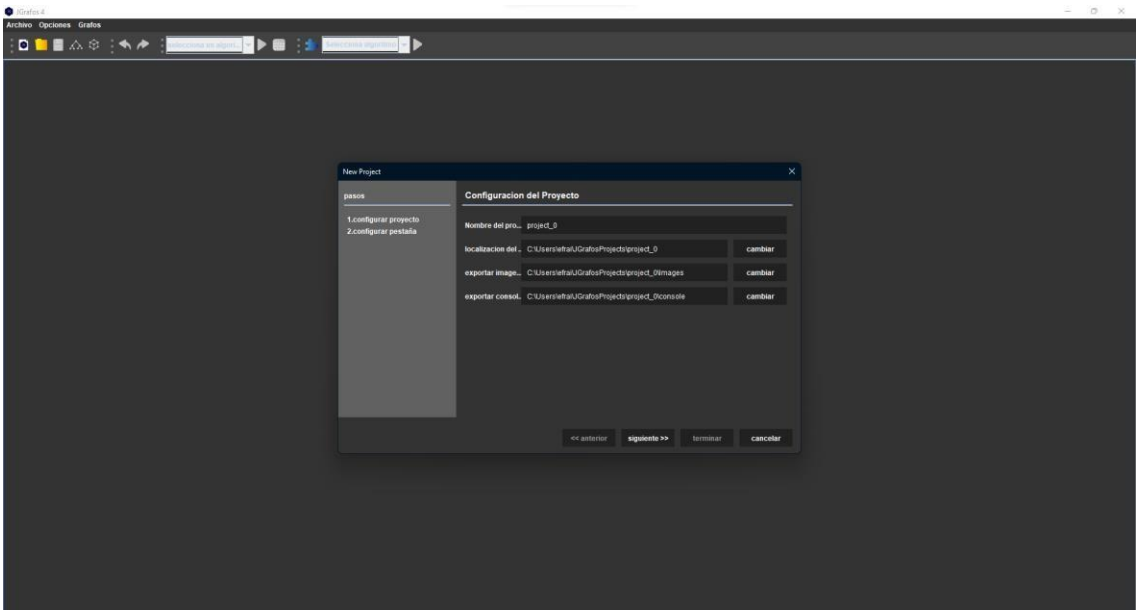


Figura 5.15. Letreros traducidos al español.

5.2. Resultados de las pruebas de módulos nativos y módulos externos

En cuanto a las pruebas de módulos, estas se refieren a la verificación del funcionamiento de los algoritmos implementados de forma nativa, así como de los módulos (implementación de algoritmos) externos, sin embargo, se optó por realizar pruebas a aquellos algoritmos que presentaron una alta taza de probabilidad de fallo en su inicio o que presentan una alta cantidad de cálculos, cabe aclarar que, otra de las finalidades de estas pruebas fueron checar el funcionamiento de las estructuras propuestas e implementadas, a continuación se mostraran los algoritmos que fueron probados así como su justificación:

- Floyd, dado a que este algoritmo realiza una gran cantidad de cálculos (dependiendo de la matriz de pesos), además este algoritmo forma parte del grupo de algoritmos de tipo matriz y usa un grafo dirigido para su ejecución.

- Prim, dado a que este algoritmo presentó en su etapa de realización una alta taza de probabilidad de fallo, además este algoritmo forma parte del grupo de algoritmos de estructura y usa un grafo no dirigido para su ejecución.
- In-Orden, dado que este algoritmo al igual que el anterior presentó una alta taza de probabilidad de fallos en su etapa de realización, además forma parte del grupo de algoritmos de estructura y usa un árbol para su ejecución.

En el caso de la prueba de la ejecución de la estructura para módulos externos, se adaptó un algoritmo de Floyd como módulo externo, cabe aclarar que se implementaron las interfaces correspondientes.

5.2.1. Resultados de la ejecución de Floyd (nativo)

La prueba se realizó con un grafo dirigido de 25 nodos (véase en la figura 5.16) interconectados en su mayoría, con la finalidad de forzar al algoritmo a hacer un consumo alto de CPU, cabe aclarar que el algoritmo puesto aprueba en esta sección fue el algoritmo de Floyd (si se desea ver su lógica, ver en figura 3.2). En cuanto a los resultados pueden verse en las siguientes imágenes (véase en las figuras 5.16 y 5.17):

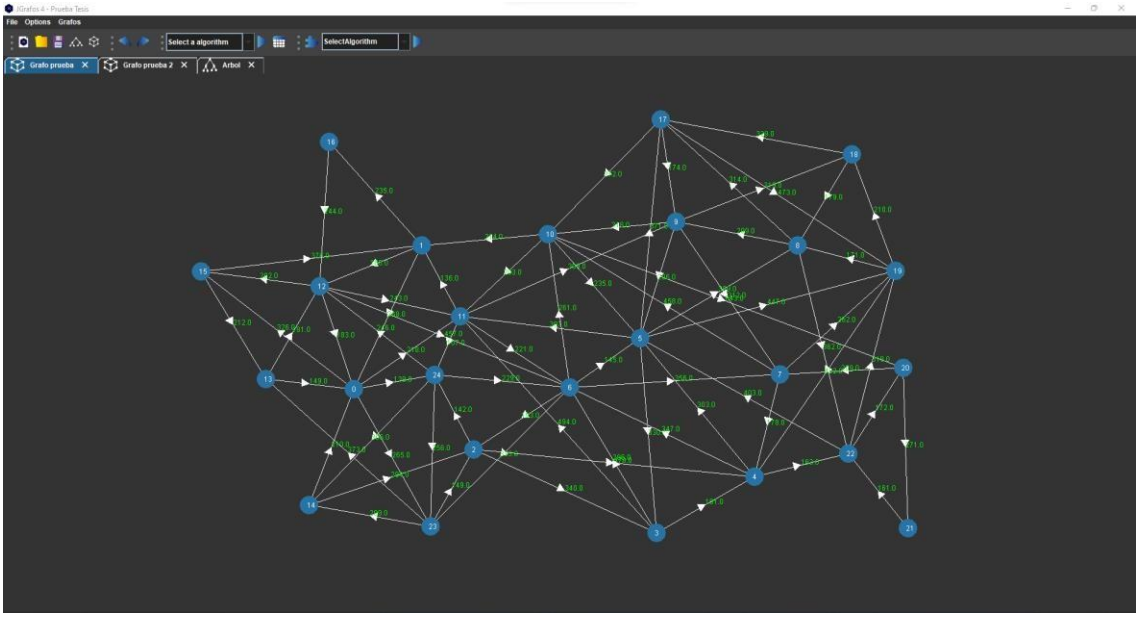


Figura 5.16. Grafo dirigido con 25 nodos.

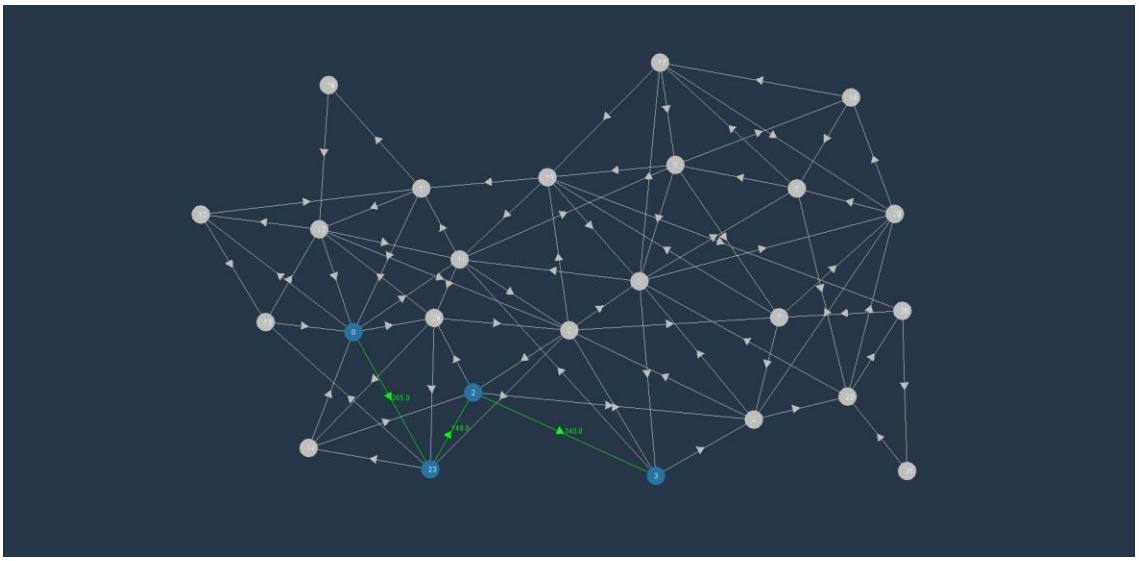


Figura 5.17. Resultado del camino más corto entre el nodo 0 y el nodo 3.

5.2.2. Resultados de la ejecución de Prim (nativo)

Para realizar esta prueba se ocupó el mismo grafo de la prueba anterior (véase en la figura 5.16) pero en su forma no dirigida (véase en figura 5.18), el algoritmo puesto a prueba en esta sección fue el algoritmo de Prim (si se desea ver su lógica, ver en figura 3.5). En cuanto a los resultados (véase en figura 5.19) fueron los siguientes:

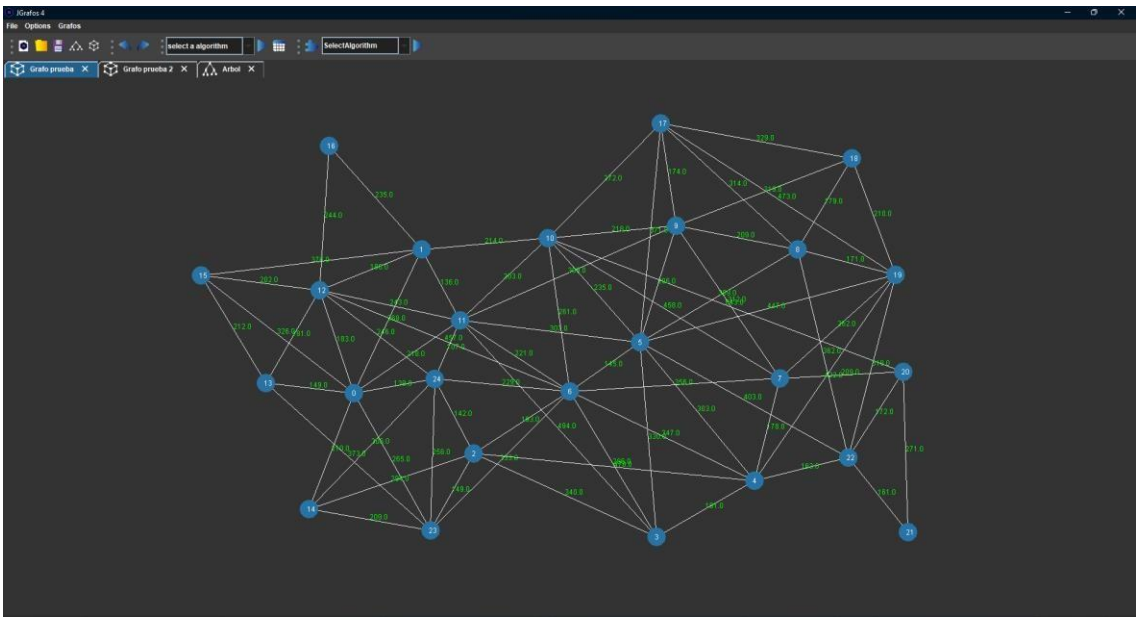


Figura 5.18. Grafo no dirigido de 25 nodos.

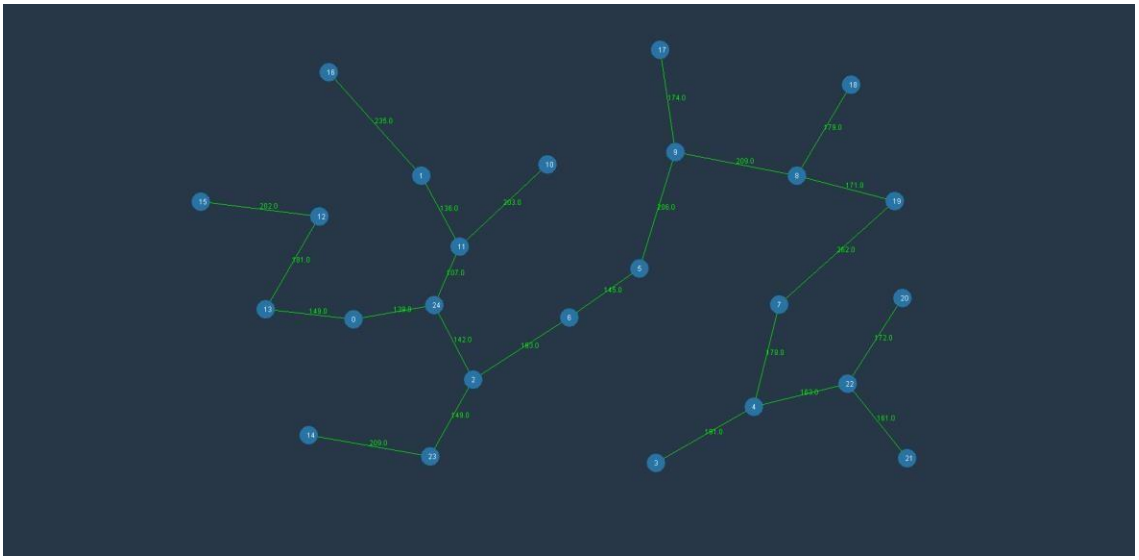


Figura 5.19. Árbol de expansión mínima resultante.

5.2.3. Resultados de la ejecución de In-Orden (nativo)

Para realizar esta prueba se requirió de un árbol (véase en figura 5.20), el algoritmo puesto aprueba en esta sección fue el algoritmo de In-Orden (si se desea ver su lógica, ver en figura 3.8), cabe aclarar que, a diferencia de otros algoritmos, este tiene la particularidad de trabajar con recursividad. En cuanto a los resultados (véase en figura 5.21) de la prueba fue la siguiente:

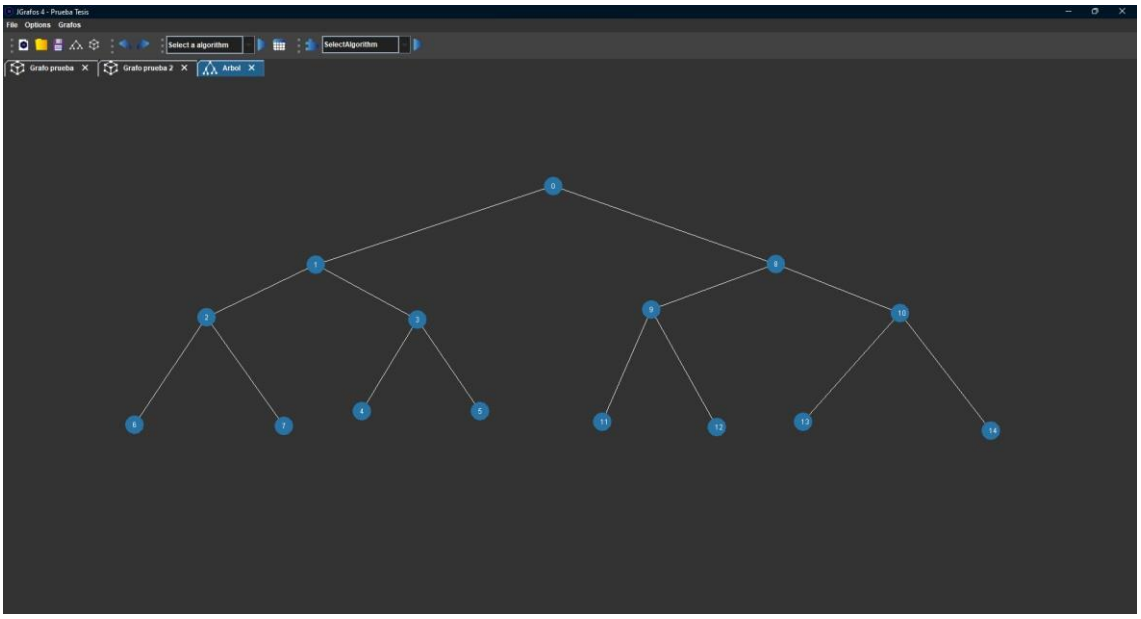


Figura 5.20. Árbol.

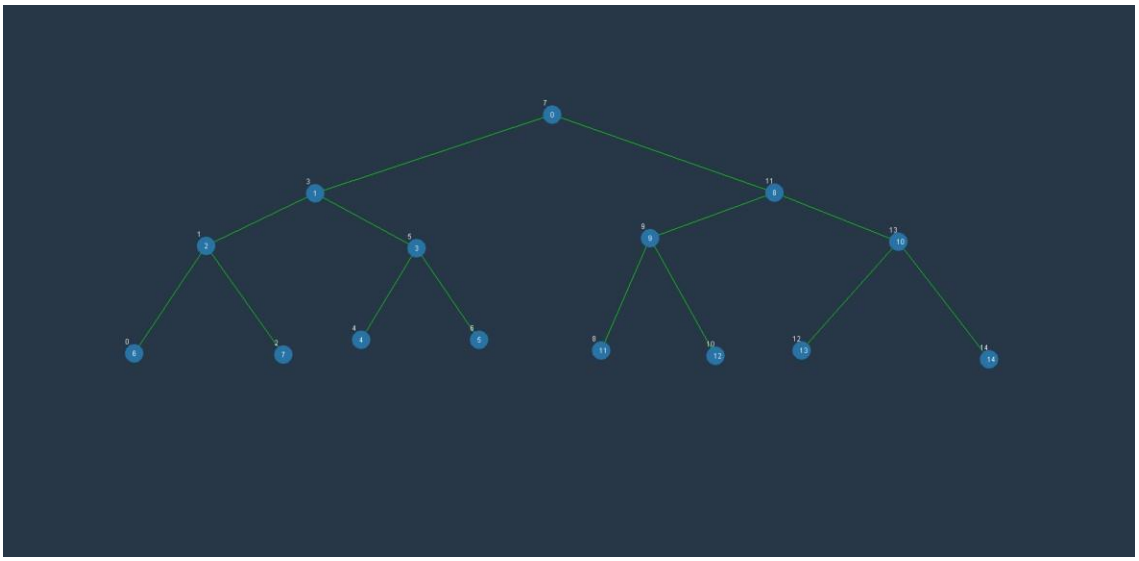


Figura 5.21. Resultado de la aplicación del algoritmo de In-Orden.

5.2.4. Resultados de la ejecución de Floyd (módulo externo)

Para realizar esta prueba se necesitaron de 2 prerrequisitos, el primero fue realizar la instalación del módulo externo, el cual se pudo ver en el apartado de pruebas de funcionalidades generales, en la sección de “Función de agregación de módulos externos” (véase en figuras 5.12 y 5.13), y el segundo de un grafo dirigido, el cual se ocupó el mismo grafo que en la prueba de Floyd (nativo, véase en figura 5.16), cabe aclarar, que el algoritmo elegido para probar en esta sección fue el algoritmo de Floyd (si se desea ver su lógica, ver en figura 3.2), ya que, como se comentó con anterioridad este algoritmo realiza muchos cálculos que van en proporción al número de nodos ingresados por el usuario, los resultados (véase en figura 5.22) fueron los siguientes:

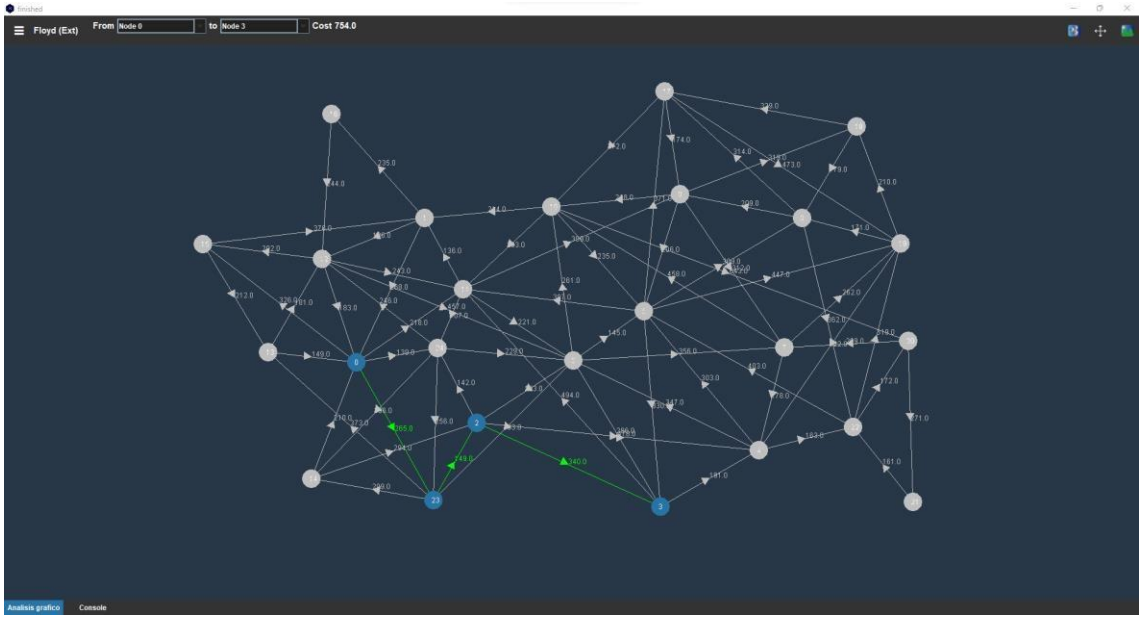


Figura 5.22. Resultados de ejecución del módulo externo Floyd.

5.3. Resultados de las pruebas sobre consumo de recursos

Estas pruebas tienen como finalidad el mostrar el desempeño del software en 2 momentos globales, es decir, en uso y en desuso, sin embargo, el momento de uso se puede subdividir en 2 en uso del canvas y el uso de ejecución de modulo, al realizarlas reflejaron (véase en tabla 5.1 y figuras 5.23, 5.24, 5.25) los siguientes datos:

Tabla 5.1. valores de consumo de recursos en sistema del software JGrafos 4

	Consumo de recursos		
	CPU	RAM	GPU
En uso (canvas)	9.4%	82.9MB	0.8%
En uso (en ejecución de algoritmos)	5.2%	1162.9MB	0%
En desuso	0.4%	66MB	0%

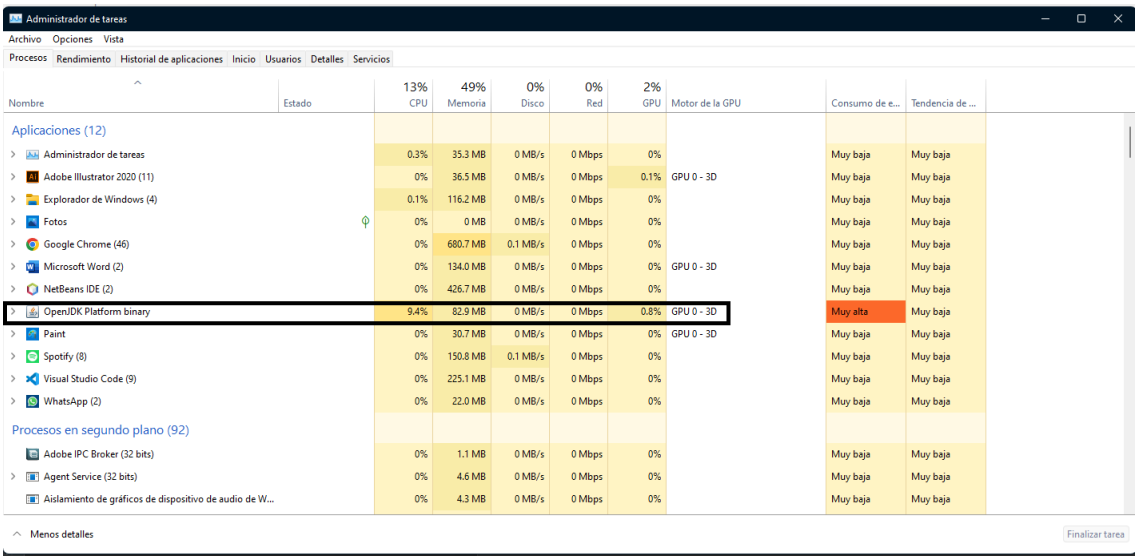
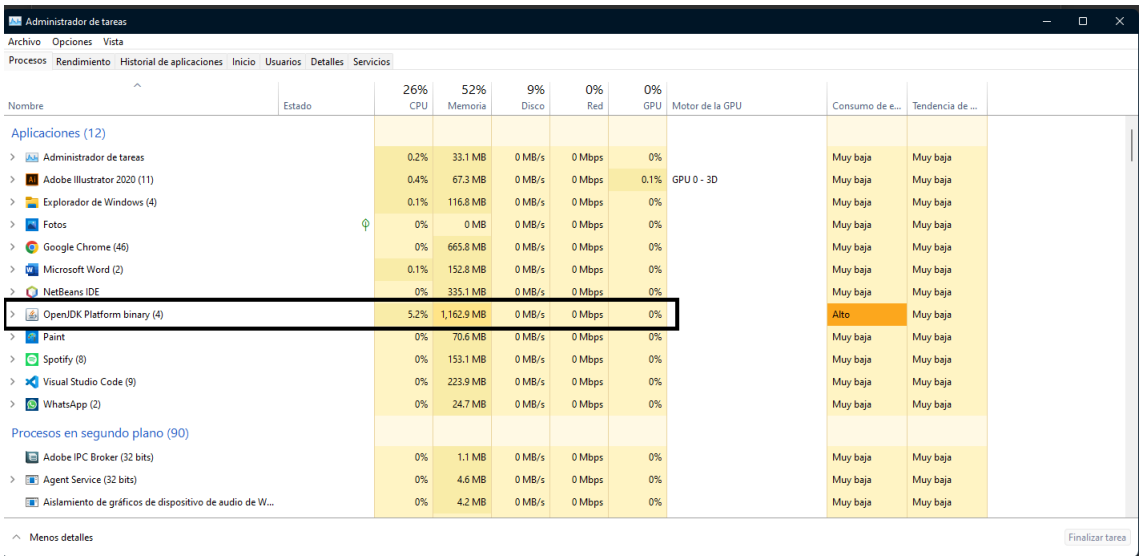


Figura 5.23. Valores del ordenador cuando se está dibujando en el espacio de trabajo.



Administrador de tareas

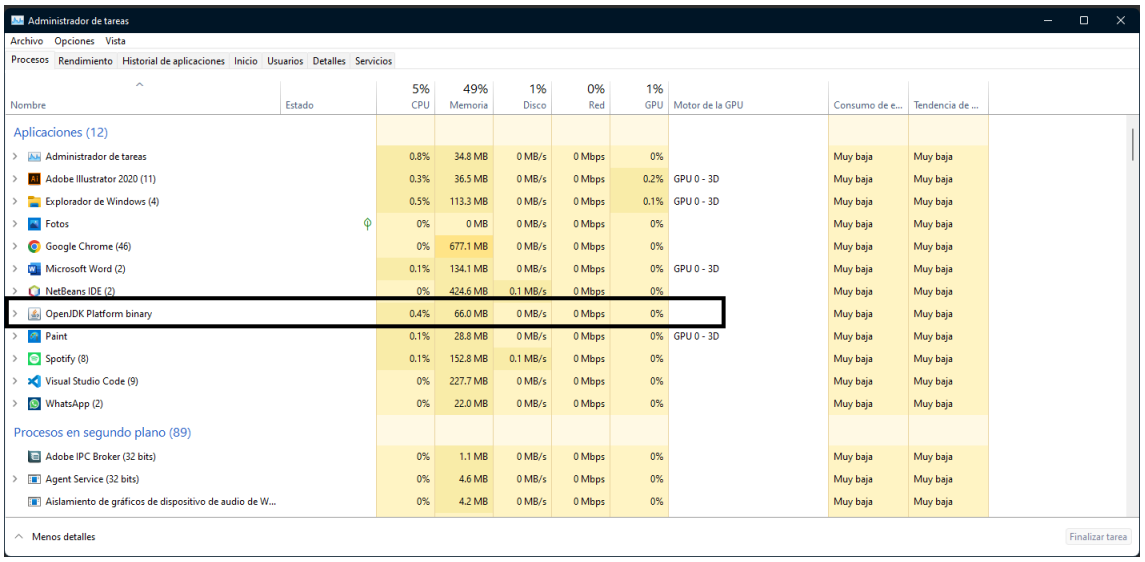
Archivo Opciones Vista

Procesos Rendimiento Historial de aplicaciones Inicio Usuarios Detalles Servicios

Nombre	Estado	26% CPU	52% Memoria	9% Disco	0% Red	0% GPU	Motor de la GPU	Consumo de e...	Tendencia de ...
Aplicaciones (12)									
Administrador de tareas		0.2%	33.1 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Adobe Illustrator 2020 (11)		0.4%	67.3 MB	0 MB/s	0 Mbps	0.1%	GPU 0 - 3D	Muy baja	Muy baja
Explorador de Windows (4)		0.1%	116.8 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Fotos		0%	0 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Google Chrome (46)		0%	665.8 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Microsoft Word (2)		0.1%	152.8 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
NetBeans IDE		0%	335.1 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
OpenJDK Platform binary (4)		5.2%	1,162.9 MB	0 MB/s	0 Mbps	0%		Alto	Muy baja
Paint		0%	70.6 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Spotify (8)		0%	153.1 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Visual Studio Code (9)		0%	223.9 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
WhatsApp (2)		0%	24.7 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Procesos en segundo plano (90)									
Adobe IPC Broker (32 bits)		0%	1.1 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Agent Service (32 bits)		0%	4.6 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Aislamiento de gráficos de dispositivo de audio de W...		0%	4.2 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja

Menos detalles Finalizar tarea

Figura 5.24. Valores del ordenador cuando se está ejecutando el módulo de Floyd con grafo de 25 nodos.



Administrador de tareas

Archivo Opciones Vista

Procesos Rendimiento Historial de aplicaciones Inicio Usuarios Detalles Servicios

Nombre	Estado	5% CPU	49% Memoria	1% Disco	0% Red	1% GPU	Motor de la GPU	Consumo de e...	Tendencia de ...
Aplicaciones (12)									
Administrador de tareas		0.8%	34.8 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Adobe Illustrator 2020 (11)		0.3%	36.5 MB	0 MB/s	0 Mbps	0.2%	GPU 0 - 3D	Muy baja	Muy baja
Explorador de Windows (4)		0.5%	113.3 MB	0 MB/s	0 Mbps	0.1%	GPU 0 - 3D	Muy baja	Muy baja
Fotos		0%	0 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Google Chrome (46)		0%	677.1 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Microsoft Word (2)		0.1%	134.1 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D	Muy baja	Muy baja
NetBeans IDE (2)		0%	424.6 MB	0.1 MB/s	0 Mbps	0%		Muy baja	Muy baja
OpenJDK Platform binary		0.4%	66.0 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Paint		0.1%	28.8 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D	Muy baja	Muy baja
Spotify (8)		0.1%	152.8 MB	0.1 MB/s	0 Mbps	0%		Muy baja	Muy baja
Visual Studio Code (9)		0%	227.7 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
WhatsApp (2)		0%	22.0 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Procesos en segundo plano (89)									
Adobe IPC Broker (32 bits)		0%	1.1 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Agent Service (32 bits)		0%	4.6 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja
Aislamiento de gráficos de dispositivo de audio de W...		0%	4.2 MB	0 MB/s	0 Mbps	0%		Muy baja	Muy baja

Menos detalles Finalizar tarea

Figura 5.25. Valores del ordenador cuando el software está en estado de desuso.

Como se puede observar en la tabla (tabla 5.1) y en las imágenes (véase las figuras 5.23, 5.24 y 5.25), los números reflejaron un consumo regular en los 3 tiempos de uso. Por ejemplo, extrayendo la información del primer tiempo (ejecución, uso del canvas), los números reflejaron un consumo moderado en la CPU (hablando en términos de ejecuciones de programas que logran ocupar un 10% de CPU), sin embargo, esto se comportó de esta forma dado que se estuvo jugando un poco con el arrastre del ratón, esto con la finalidad de ver tanto la animación de las líneas, así como, verificar un caso extremo, ya que se temía que el hacer esa acción consumiría por lo menos un 40% de la CPU, siguiendo el mismo análisis en cuanto a la memoria RAM, esta se puede ver que no pasa de los 100 MB, esto es una buena señal ya que actualmente hay aplicaciones que consumen cerca de 180 MB al inicio. Examinando el segundo caso (en uso), se puede observar que el consumo de la CPU fue moderado en comparación a la lectura anterior, sin embargo, la lectura de la memoria RAM si fue muy elevada a comparación de la anterior lectura (uso del canvas), sin embargo, tiene

congruencia debido a la cantidad de nodos a trabajar y además, que se trataba de un algoritmo pesado (hablando en términos de recursos en CPU, RAM y GPU), como se puede observar, el uso de la GPU es congruente, ya que no se hace ningún renderizado en el canvas, finalmente en el caso de desuso se puede observar que los valores de consumo son inferiores a los de uso, ya que en cuanto CPU solo se gasta un 0.4% (debido a ciertas validaciones), en cuanto a la memoria RAM solo se ocupa 66 MB (por la interfaz gráfica) y en cuanto a la GPU, esta se marca en desuso, como se puede observar los números son congruentes con las diversas etapas de trabajo.

Conclusiones

La presente tesis tuvo como objetivo principal crear una herramienta didáctica que ayude en el estudio y entendimiento de los grafos de una forma más sencilla y visual, además de proporcionar la característica de implementación de módulos externos (módulos creados por terceros) para su funcionamiento en el software.

En cuanto al enfoque del software, se desarrolló pensando en académicos y estudiantes, ya que estos 2 tipos de usuarios son a quienes más les podría interesar este software, por lo que se optó por implementar ciertas herramientas que ayudan a dichos usuarios a tener un mejor entendimiento sobre la resolución de problemas con grafos, como es el caso de la implementación de resultados en formato de texto (consolas que describen el desarrollo de la respuesta) o la implementación de las respuestas gráficas (vista de análisis gráfico).

En cuanto al diseño de la interfaz gráfica, se implementaron bocetos usando iconos descriptivos que permitieran identificar las tareas o funciones a realizar, también, se implementaron ciertas animaciones en la forma de creación esquemas (grafos y árboles), además, abordando desde la perspectiva de la salud visual, se tienen conocimientos que indican que el uso de los colores claros tiene efectos negativos con respecto a la visión de las personas, por lo cual se implementó una paleta de colores obscuros para cuidar la visión de los usuarios.

Sobre la ergonomía en el software, se implementó una barra de acceso rápido que se muestra en la ventana principal la cual contiene un conjunto de funciones las cuales se determinó que serían de mayor uso, por lo que se integró con la finalidad de evitar la fatiga al usuario de buscar las opciones en los menús superiores.

En cuanto a la estructura del software resultó conveniente separarlo en paquetes, ya que esto permitió una mejor administración en el desarrollo del software, sin embargo, esta no fue la única razón por la cual se decidió trabajar de esta manera, ya que dicha separación permite trabajar en secciones específicas, lo cual permitirá en el futuro sobrellevar su manutención, además, teniendo en cuenta que el conocimiento sobre los grafos puede cambiar en un futuro, por lo que la modularización efectuada permitiría actualizar los bloques de código que requieran una actualización, permitiendo al software evolucionar.

Sobre la implementación de módulos externos, los conocimientos sobre Java RMI y la implementación de ProcessBuilder, jugaron un roll importante en la creación del lanzador de procesos y de la forma de comunicación con el software principal, ya que gracias a la clase de ProcessBuilder se permite lanzar programas externos y Java RMI permite realizar puentes de comunicación entre procesos, este último permite crear una interfaz de comunicación, la cual fue aprovechada para compartir los datos, cabe aclarar que dicha interfaz también puede ser modificada en el futuro para acoplarse a las necesidades que se requieran.

En cuanto al tema de la optimización de recursos, se implementaron ciertos mecanismos que permiten gestionar de una forma eficiente los recursos de CPU, GPU y memoria RAM, ya que como se comentó en el capítulo 3, una de las inquietudes era el realizar un software que fuera “cuidadoso” con los recursos de la máquina para así poder ser ejecutado en ordenadores de bajos recursos.

Por último, en cuanto al futuro del software, se tiene contemplado expandirlo a un ambiente web, con el objetivo de realizar mejoras en el rendimiento, así como de mantenerlo en el lenguaje de java, ya que este lenguaje permite un rendimiento alto en cuestión de cálculos pesados e incluso se podría reestructurar de tal forma de distribuir la carga de trabajo en varios servidores para así mejorar los tiempos de respuesta del software.

Anexo

CERTIFICADO

Registro Público del Derecho de Autor

Para los efectos de los artículos 13, 162, 163 fracción I, 164 fracción I, y demás relativos de la Ley Federal del Derecho de Autor, se hace constar que la **OBRA** cuyas especificaciones aparecen a continuación, ha quedado inscrita en el Registro Público del Derecho de Autor, con los siguientes datos:

AUTORES: HERNÁNDEZ BERISTAIN ADRIANA
LARIOS GÓMEZ MARIANO
LIMA ESTRADA EFRAIN DE JESÚS
MARTÍNEZ MIRÓN ERIKA ANNABEL
TOLENTINO LARIOS GABRIEL

TÍTULO: JGRAFOS 4

RAMA: PROGRAMAS DE COMPUTACION

TITULARES: HERNÁNDEZ BERISTAIN ADRIANA
LARIOS GÓMEZ MARIANO
LIMA ESTRADA EFRAIN DE JESÚS
MARTÍNEZ MIRÓN ERIKA ANNABEL
TOLENTINO LARIOS GABRIEL

Con fundamento en lo establecido por el artículo 168 de la Ley Federal del Derecho de Autor, las inscripciones en el registro establecen la presunción de ser ciertos los hechos y actos que en ellas consten, salvo prueba en contrario. Toda inscripción deja a salvo los derechos de terceros. Si surge controversia, los efectos de la inscripción quedarán suspendidos en tanto se pronuncie resolución firme por autoridad competente.

Con fundamento en los artículos 2, 208, 209 fracción III y 211 de la Ley Federal del Derecho de Autor; artículos 64, 103 fracción IV y 104 del Reglamento de la Ley Federal del Derecho de Autor; y artículos 1, 3 fracción I, 4, 8 fracción I y 9 del Reglamento Interior de Instituto Nacional del Derecho de Autor, se expide el presente certificado.

Número de Registro: 03-2022-010310403700-01

Ciudad de México, a 21 de enero de 2022

EL DIRECTOR DEL REGISTRO PÚBLICO DEL DERECHO DE AUTOR

SECRETARÍA DE CULTURA
INSTITUTO NACIONAL DEL DERECHO DE AUTOR
DIRECCIÓN DEL REGISTRO PÚBLICO DEL DERECHO DE AUTOR

JESÚS PARETS GÓMEZ

Bibliografía

1. *Villalobos, Alejandro Rodríguez. "Grafos: herramienta informática para el aprendizaje y resolución de problemas reales de teoría de grafos." X Congreso de Ingeniería de Organización. 2006.*
2. *Bastian, Mathieu, Sebastien Heymann, and Mathieu Jacomy. "Gephi: an open-source software for exploring and manipulating networks." Proceedings of the international AAAI conference on web and social media. Vol. 3. No. 1. 2009.*
3. *Pérez Hernández, Dagoberto, et al. ASISTENTE MATEMÁTICO PARA TEORÍA DE GRAFOS EN LA ASIGNATURA MATEMÁTICA DISCRETA. BS thesis. 2014*
4. *Gil Gil, José María, and Gregorio Hernández Peñalver. "ProRouting: una herramienta para la visualización de grafos de proximidad y estrategias de ruteo." (2008).*
5. *Clark, Mason. Russian hybrid warfare. Washington, DC: Institute for the Study of War, 2020.*
6. *Coene, Jean-Philippe. "sigmajs: An R htmlwidget interface to the sigma. js visualization library." Journal of Open Source Software 3.28 (2018): 814.*
7. *Verdesoto Paz, Vinicio Marcelo. Manejo de grafos sobre el mapa de la ciudad de Quito a través de la herramienta Jung de Java. BS thesis. 2015.*
8. *Núñez Valdés, Juan, et al. "Siete puentes, un camino: Königsberg." Suma: revista sobre la enseñanza y aprendizaje de las matemáticas, 45, 69-78. (2004).*
9. *Apt, Krzysztof R. "Edsger Wybe Dijkstra (1930–2002): A portrait of a genius." (2002): 92-98.*
10. *Aho, Alfred VI, E. John, and D. Jeffrey. Estructuras de datos y algoritmos/por Alfred V. Aho, John E. Hopcroft y Jeffrey D. Ullman. No. 001.642 A4y.*
11. *Ruiz, S. "El algoritmo K-NN y su importancia en el modelado de datos." Ad Tech & Analytics (2017).*
12. *"Algoritmo k-Nearest Neighbor | Aprende Machine Learning". Aprende Machine Learning, 10 de julio de 2018, www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python.*
13. *Aragón Loza, Marco Antonio. "Planificación de rutas accesibles para personas con discapacidad visual utilizando programación lineal entera y el algoritmo de Dijkstra."*
14. *Pérez, Imanol. "Erdös y los inversos de los números primos." 2 Revista πkasle Aldizkaria Aurkibidea Índice: 16.*
15. *Carranza, Edwin Alfredo. "Teorema de Hall y sus equivalencias." (2006): 559-567.*
16. *RHEE, JAIME DAN PORRAS. "EL TEOREMA DE JORDAN Y LOS GRAFOS PLANARES."*
17. *Wilson, Robin. "Cuatro colores bastan o cómo colorear un mapa1." Números: Revista de didáctica de las matemáticas 56 (2003): 49-56.*
18. *Torrubia, G., and V. Terrazas. "Algoritmo de Dijkstra. Un tutorial interactivo." VII Jornadas de Enseñanza Universitaria de la Informática (JENUI 2001) (2012).*
19. *Ortega, María Peñaranda, et al. "Los «Small Worlds» y el algoritmo de Floyd: una manera de estudiar la colaboración científica." Psicothema 18.1 (2006): 78-83.*
20. *Tocto Inga, Paul Miller. "Comparación de los algorítmos Prim y Kruskal." (2012).*
21. *Barelles Menes, Jorge. "Algoritmos para la resolución de problemas en redes." (2017).*
22. *Reyes Esquén, Jeremy Octavio. "Desarrollo de un planificador de rutas para recojo de desechos sólidos utilizando algoritmo de Bellman Ford." (2018).*
23. *LILIA, CHAVEZ SOTO TANIA. "MODULARIDAD."*
24. *Rodríguez, Roberto, Encarna Sosa, and Álvaro Prieto. "Programación orientada a objetos." (2004).*
25. *Requene Estacio, Neider Fabricio. Implementación de algoritmos sobre arquitectura multinúcleo para optimizar el alto coste computacional al procesar grandes volúmenes de datos. BS thesis. 2022.*