Título:

EUREKA: Producto Informático para la Solución a problemas de Modelos de Redes.

Autor:

Raúl René Alpizar Gamboa.

Tutores:

Dr. Manuel Cortés Cortés.

Msc. Daimarelys Acevedo Cardoso.

Consultantes:

Dr. Raúl Alpizar Fernández

Cienfuegos, 2010.

"Año 52 de la Revolución"

Declaración de autoría

Declaro que soy el único autor de este trabajo y autorizo al Departamento de Matemática y al Departamento de Informática de la Facultad de Informática en la Universidad de Cienfuegos “Carlos Rafael Rodríguez”, para que hagan el uso que estimen pertinente con el trabajo de diploma.

Para que así conste firmo la presente a los ____ días del mes de ___ del ___.

_____

Nombre completo del primer autor

Los abajo firmantes certificamos que el presente trabajo ha sido revisado según acuerdo de la dirección de nuestro centro y el mismo cumple los requisitos que debe tener un trabajo de esta envergadura referente a la temática señalada.

---------- ----------

Firma Tutor Firma Tutor

---------- ----------

Firma ICT Firma Vicedecano

Agradecimientos

A todo el mundo pero en especial a mis tutores que me deben de estar revisando el documento en este preciso momento.

Dedicatoria

A mi CDR.

Resumen

La presente investigación tiene como título “EUREKA: Producto Informático para la Solución a problemas de Modelos de Redes” y ha sido realizada con el objetivo de crear un producto informático que resuelva, de forma gráfica, los problemas más usuales de la teoría de redes.

Dicho producto permite una mejor interacción siendo más amigable al usuario, es de fácil manejo de los gráficos para la construcción e interpretación de las soluciones posibilitando analizar el comportamiento de los algoritmos implementados en el grafo construido y además posee algoritmos de Modelos de Redes no incluidos en los paquetes informáticos existentes.

Se utilizó el Lenguaje Unificado de Modelado (UML) para el análisis, diseño e implementación de la solución propuesta, siguiendo lo establecido por el Proceso Unificado de Desarrollo de Software (RUP). La implementación se realizó con la herramienta Visual C++ utilizando el lenguaje de programación orientado a objetos C++ además de la librería Qt para la optimización del trabajo con los gráficos.

Índice

Índice de tablas

Índice de figuras

Introducción

La Investigación de Operaciones es la disciplina matemática que se ocupa de las aplicaciones en múltiples problemas tales como: Programación Lineal, Programación en enteros, Programación Reticular, Programación Dinámica, Programación no Lineal, Programación Convexa, Teoría de Inventarios, Teoría de la Decisión, Teoría de Colas, entre otras. A su vez se orienta a la resolución de problemas relacionados con la conducción y coordinación de las operaciones o actividades dentrode una organización. Su ámbito de aplicación es muy amplio, aplicándose a problemas de fabricación, transporte, construcción, telecomunicaciones, planificación y gestión financiera, ciencias de la salud, servicios públicos, etc. En general, puede aplicarse en todos los problemas relacionados con la gestión, la planificación y el diseño.

La Investigación de Operaciones incluye un conjunto muy amplio de técnicas orientadas a proporcionar una ayuda cuantitativa a la toma de decisiones. El método empleado es el método científico, y las técnicas que se utilizan son, en buena medida, técnicas matemáticas.

La teoría de redes se enmarca dentro de la Investigación de Operaciones y describe modelos matemáticos que involucran la representación gráfica de diversos problemas de optimización. Como Matemática Aplicada las Redes tienen aplicaciones muy amplias, formulándose modelos matemáticos de Redes en innumerables situaciones. Con motivo de su vasta aplicación y a la valiosa ayuda que proporcionan para el entendimiento de los sistemas, ha habido gran actividad en su estudio. Gracias a esto y a la estructura que presentan los modelos de redes, se han desarrollado algoritmos eficientes para la solución de los problemas formulados.

La historia y desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) en Cuba han evolucionado de manera vertiginosa. En las esferas de la Salud, Comercio, Industria, Educación, entre otras, se ha logrado aumentar el rendimiento en sus metas gracias a la informatización de la gestión. El tratamiento de la información se realiza hoy en día más rápida y oportunamente, de ahí la importancia a escala internacional del crecimiento de estas tecnologías. Se denominan medios de comunicación a los medios desarrollados a partir de las TIC. Estos medios se encargan

del estudio, desarrollo, implementación, almacenamiento y distribución de la información haciendo uso de hardware y software. En general se trata del empleo de las computadoras y las aplicaciones informáticas para la gestión, difusión y localización de los datos necesarios para cualquier actividad humana.

La nueva economía se ha venido caracterizando fundamentalmente por el incremento de inversiones por parte de las empresas en las TIC. Es evidente que su uso es una cuestión clave para la expansión y supervivencia de cualquier organización, pues constituye un elemento imprescindible que permite un continuo desarrollo.

Desde el punto de vista de optimización tanto en la representación como en la descripción, la relación existente entre los Modelos Matemáticos de Redes y los Sistemas Informáticos goza de gran fortaleza, y muy justificado, por la variedad de software y paquetes, de este tipo, existentes.

Situación Problémica.

Existen numerosos productos informáticos que trabajan el tema de la Teoría de Redes. En la mayoría de los mismos el tratamiento de datos y soluciones se da en forma tabular y aquellos que presentan tratamiento gráfico lo hacen de una forma complicada, además no son capaces de desdoblar los algoritmos en su desarrollo. No permiten una buena visibilidad de los gráficos para la entrada del problema, su solución parcial o su solución completa. No implementan las soluciones de la manera más óptima, utilizándose métodos de programación lineal lo que provoca un aumento de la complejidad temporal. Por lo antes expuesto es necesaria la implementación de productos informáticos que sean más amigables a los usuarios, tanto estudiantes como investigadores, y que resuelvan los problemas de la teoría de redes de una manera más eficiente.

Problema de Investigación

La carencia de un Producto Informático para atender las aplicaciones de la Teoría de Redes que sea amigable al usuario, de fácil manejo de los gráficos para la construcción e interpretación de las soluciones y que posea algoritmos no incluidos en los Paquetes Informáticos existentes.

Preguntas de Investigación

- ¿El diseño de algoritmos matemáticos y computacionales de Teoría de Redes, satisface las necesidades que exige hoy la resolución de problemas de la Disciplina de Investigación de Operaciones?
- ¿Se seleccionarán los algoritmos matemáticos más usuales para la implementación del sistema informático?
- ¿Se podrán implementar los algoritmos seleccionados para la resolución de problemas en los modelos de redes de acuerdo a las exigencias del cliente?
- ¿El sistema computacional desarrollado permitirá de una forma más clara la apropiación del conocimiento por parte de los estudiantes e investigadores?

Objeto de Estudio

La aplicación de los Modelos Matemáticos de la Investigación de Operaciones.

Campo de Acción

La aplicación de los Modelos Matemáticos de la Teoría de Redes.

Objetivo General

Elaborar un producto informático que resuelva, de forma gráfica, los problemas más usuales de la teoría de redes.

Objetivos Particulares

1. Realizar el análisis del producto informático propuesto.
2. Diseñar el producto informático propuesto.
3. Implementar el producto informático propuesto.
4. Validar el producto informático propuesto.

Tareas de Investigación

- Estudiar cómo se solucionan matemáticamente los algoritmos desarrollados.

- Analizar a profundidad las herramientas que se aplican actualmente en la solución de problemas de Redes.
- Revisar y analizar la bibliografía contemporánea para caracterizar el estado actual de la problemática planteada.
- Identificar y analizar la aplicabilidad, ventajas y desventajas de algoritmos existente para la solución del problema de modelos de redes.
- Realizar la modelación de la solución, con ayuda de la herramienta Rational Rose, aplicando la metodología RUP y el lenguaje de modelado UML.
- Definir los requerimientos funcionales y no funcionales.
- Crear el diseño de clases e implementar los métodos necesarios para satisfacer los requerimientos propuestos para el producto informático.
- Realizar la interfaz gráfica de la aplicación.
- Estudiar y analizar los costos y beneficios que tiene la puesta en marcha de la solución propuesta.
- Aplicar encuestas a estudiantes con dominio de la Teoría de Redes.
- Procesar las encuestas aplicadas haciendo uso de un programa estadístico.
- Escribir el informe de investigación.

Idea a defender

Si se elabora un producto informático para la solución de problemas de la Teoría de Redes, que desarrolle un nuevo sistema gráfico de introducción de datos y obtención de soluciones, se contribuirá a mejorar la interpretación de estos algoritmos para estudiantes e investigadores.

Justificación.

Luego de haberse consultado todos los paquetes de programas que resuelven redes, conocidos por el autor: Qsb, WinQsb, MathLab, Qmwin2 y TORA, se ha podido

identificar que los mismos presentan insuficiencias en el trabajo con las redes desde el punto de vista gráfico. El aprendizaje de estos se torna demasiado complejo para realizar operaciones sencillas y de poco grado de dificultad. El paquete diseñado es más amigable, más fácil de introducir e interpretar la solución obtenida, es útil como paquete de aplicación, así como para el estudio de las redes en los estudiantes de la Investigación de Operaciones.

Para el adecuado análisis y entendimiento de este documento, se ha estructurado el mismo en 3 capítulos. Los cuales hacen referencia a:

- **Capítulo I.-** "Fundamentación teórica":

  En este capítulo se exponen y detallan las características, conceptos básicos y enfoques en el desarrollo del software; se describen los sistemas ya existentes asociados a la Teoría de Redes y se analizan lenguajes y metodologías aplicadas en el desarrollo del sistema.

- **Capítulo II.-** "Descripción y construcción de la solución propuesta":

  En este capítulo se seleccionó como punto de partida el modelo del dominio. También se describe la solución propuesta utilizando algunos de los artefactos que propone la Metodología RUP. Los artefactos referidos son: los requerimientos funcionales y no funcionales, el diagrama de casos de uso, la descripción de cada uno, diagrama de clases del diseño y diagrama de clases persistentes. Se incluyen además los principios de diseño.

- **Capítulo III.-** "Estudio de factibilidad y validación del sistema":

  En este capítulo se estima el esfuerzo humano, el tiempo de desarrollo y también el costo para la realización del producto informático, determinándose así si es factible o no su desarrollo. También se validará el producto basándose en el análisis de los resultados obtenidos de las encuestas realizadas a los estudiantes que han cursado la asignatura de Investigación de Operaciones II. Se utilizará la prueba de hipótesis para la comparación de los resultados, apreciando así si existe acuerdo o no entre los encuestados.

Capítulo 1 – Fundamentación teórica

1.1 – Introducción

En este capítulo se exponen y detallan las características, conceptos básicos y enfoques en el desarrollo del software; se describen los producto informático ya existentes asociados a la Teoría de Redes y se analizan lenguajes y metodologías aplicadas en el desarrollo del sistema.

1.2 – Descripción del dominio del problema

1.2.1 – Sistema Informático

Un sistema informático como todo sistema, es el conjunto de partes interrelacionadas, hardware, software y de Recurso Humano (humanware). Un sistema informático típico emplea una computadora que usa dispositivos programables para capturar, almacenar y procesar datos **[1]**. La computadora personal o PC, junto con la persona que lo maneja y los periféricos que los envuelven, resultan de por sí un ejemplo de un sistema informático.

1.2.2 – Teoría de Redes

En matemáticas y en ciencias de la computación, la teoría de redes (también llamada teoría de grafos) estudia las propiedades de los grafos. Un grafo es un conjunto, no vacío, de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas que pueden ser orientados o no **[2]**.

El trabajo de Leonhard Euler, en 1736, sobre el problema de los puentes de Königsberg es considerado el primer resultado de la teoría de grafos.

También un grafo es una terna **G = (V,A,j),** en donde **V** y **A** son conjuntos finitos, y **j** es una aplicación que hace corresponder a cada elemento de **A** un par de elementos de **V**. Los elementos de **V** y de **A** se llaman, respectivamente, **"vértices"** y **"aristas"** de **G**, y **j** asocia entonces a cada arista con sus dos vértices **[2]**.

Esta definición da lugar a una representación gráfica, en donde cada vértice es un punto del plano, y cada arista es una línea que une a sus dos vértices, como se muestra en la Figura I.

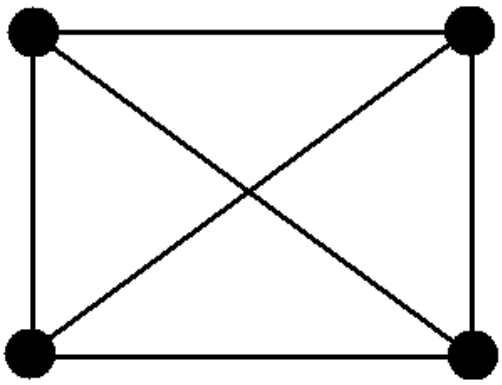


Figura I. Representación Visual de un Grafo

Hoy en día es rara la disciplina científica o humanística que no utiliza la teoría de grafos. Como ejemplos se pueden citar la psicología en dinámica de grupos, la física teórica, que usa los diagramas de Feynmann, donde se representan mediante líneas las partículas elementales, el estudio de flujos en redes en programación lineal e investigación operativa, los cambios de variable en el cálculo diferencial, entre otros.

TIPOS DE GRAFOS

Existen dos tipos de grafos los no dirigidos y los dirigidos.

- **NO DIRIGIDOS:** son aquellos en los cuales los lados no están orientados. Cada lado se representa entre paréntesis, separando sus vértices por comas, y teniendo en cuenta **$(V_i,V_j)=(V_j,V_i)$**. Figura II.
- **DIRIGIDOS:** son aquellos en los cuales los lados están orientados. Cada lado se representa entre ángulos, separando sus vértices por comas y teniendo en cuenta **$<V_i, V_j>!=<V_j, V_i>$**. En grafos dirigidos, para cada lado **<X, Y>**, **X**, el cual es el vértice origen, se conoce como la cola del lado y **B,** el cual es el vértice destino, se conoce como cabeza del lado. Figura III.

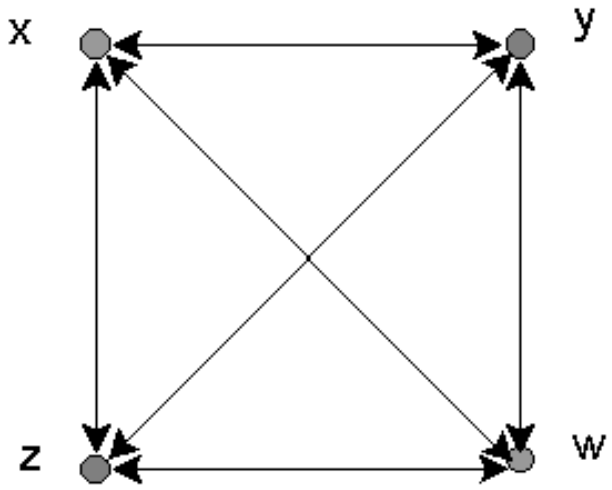


Figura II. Grafo No Dirigido

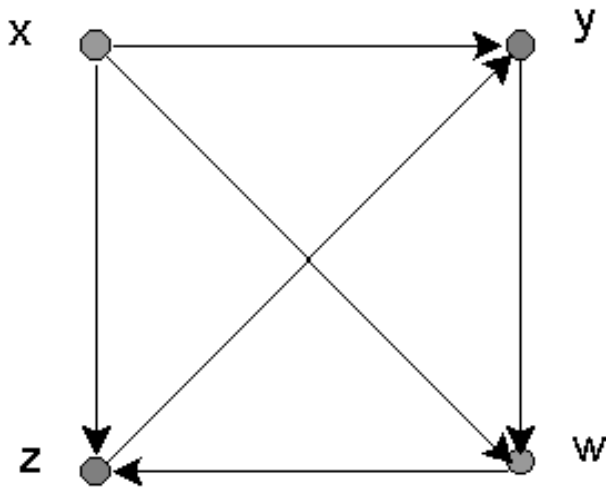


Figura III. Grafo Dirigido

Terminología de grafos

ADYACENCIA: Para no dirigidos se dice que dos vértices son adyacentes si forman un lado. En dirigidos se extiende el concepto: sea el lado **<V_i, V_j>:** se dice **V_j es adyacente desde V_i** y **V_i es adyacente hacia V_j [3]**.

INCIDENCIA: se dice que un lado es incidente sobre el par de vértices que lo conforman **[3]**.

GRADO DE UN VÉRTICE (NODO): es el número de lados incidentes sobre él**.**

Para grafos dirigidos el grado se diferencia así: **grado entrante** que es el número de lados que llegan al nodo y el **grado saliente** que es el número de lados que salen del nodo. La suma de ambos es el grado total.

TRAYECTORIA

Es una secuencia de lados para ir desde $\mathbf{V_i}$ a $\mathbf{V_j}$, en la cual cada pareja consecutiva de vértices debe ser un lado definido para el grafo **[3]**. Si alguna pareja de vértices consecutivos no está definida en el conjunto de lados del grafo se dice que la trayectoria es **inválida.**

Si en un grafo es posible viajar de cada nodo hacia todos los demás se dice que es un **grafo conectado**, si el grafo es no dirigido. Si el grafo es dirigido se dice que es **fuertemente conectado**.

El número de lados presentes en una trayectoria se denomina **longitud de la trayectoria**.

1.2.2.1 – Aristas

Son las líneas con las que se unen las aristas de un grafo y con la que se construyen también caminos **[4]**.

Si la arista carece de dirección se denota indistintamente **<X, Y>** o **< Y, X>**, siendo X y Y los vértices que une.

Es importante destacar que las aristas tienen varias clasificaciones, dentro de ellas la más importante es el término de aristas adyacentes el cual se dice que dos aristas son adyacentes si convergen en el mismo vértice.

1.2.2.2 – Vértices

Son los puntos o nodos con los que está conformado un grafo. Se denomina grado de un vértice al número de aristas de las que es extremo. Se dice que un vértice es *par* o *impar* según lo sea su grado **[4]**.

- Vértices Adyacentes: si se tiene un par de vértices de un grafo (U, V) y una arista que los une, entonces U y V son vértices adyacentes y se dice que U es el vértice inicial y V el vértice adyacente.

- Vértice Aislado: Es un vértice de grado cero.
- Vértice Terminal: Es un vértice de grado uno.

1.2.2.3 – Caminos

Sean X, Y dos vértices, se dice que hay un camino en G de *x* a *y* si existe una sucesión finita no vacía de aristas $\{X,v_1\}$, $\{v_1,v_2\}$,..., $\{v_n,Y\}$. En este caso X e Y se llaman los extremos del camino, el número de aristas del camino se llama la longitud del camino. Si los vértices no se repiten el camino se dice propio o simple. Si hay un camino no simple entre 2 vértices, también habrá un camino simple entre ellos. Cuando los dos extremos de un camino son iguales, el camino se llama circuito o camino cerrado. Se le llama ciclo a un circuito simple y un vértice X se dice accesible desde el vértice Y si existe un camino entre ellos **[4]**.

1.2.2.4 – Árbol

Un grafo que no tiene ciclos y que conecta a todos los puntos, se llama árbol. En un grafo con n vértices, los árboles tienen exactamente n - 1 aristas, y hay n^{n-2} árboles posibles. Su importancia radica en que los árboles son grafos que conectan todos los vértices utilizando el menor número posible de aristas. Un importante campo de aplicación de su estudio se encuentra en el análisis filogenético, el de la filiación de entidades que derivan unas de otras en un proceso evolutivo, que se aplica sobre todo a la averiguación del parentesco entre especies; aunque se ha usado también, por ejemplo, en el estudio del parentesco entre lenguas **[4]**.

1.2.2.5 – Algoritmos

Algoritmo de Dijkstra (Camino Mínimo – Camino Máximo)

En 1956, Edsger Wybe Dijkstra anunció su **algoritmo de caminos mínimos**, después de haber estado trabajando con el ARMAC, el ordenador que el Centro Matemático poseía.

Una posible definición de algoritmo es un conjunto de reglas que permite obtener un resultado determinado a partir de ciertas reglas definidas **[5]**. Otra definición sería,

algoritmo es una secuencia finita de instrucciones, cada una de las cuales tiene un significado preciso y puede ejecutarse con una cantidad finita de esfuerzo en un tiempo finito. Ha de tener las siguientes características: legible, correcto, modular, eficiente, estructurado, no ambiguo y a ser posible se ha de desarrollar en el menor tiempo posible. El término proviene del matemático árabe Al'Khwarizmi, que escribió un tratado sobre los números. Este texto se perdió, pero su versión latina, *Algoritmi de Numero Indorum*, sí se conoce **[5]**.

Una red de comunicaciones involucra un conjunto de nodos conectadas mediante arcos, que transfiere vehículos desde determinados nodos origen a otros nodos destino. La forma más común para seleccionar la trayectoria (o ruta) de dichos vehículos, se basa en la formulación de la ruta más corta. En particular a cada arco se le asigna un escalar positivo el cual se puede ver como su longitud.

Un algoritmo de trayectoria más corta, rutea cada vehículo a lo largo de la trayectoria de longitud mínima (ruta más corta) entre los nodos origen y destino. Hay varias formas posibles de seleccionar la longitud de los enlaces. La forma más simple es que cada enlace tenga una longitud unitaria, en cuyo caso, la trayectoria más corta es simplemente una trayectoria con el menor número de enlaces. De una manera más general, la longitud de un enlace puede depender de su capacidad de transmisión y su carga de tráfico.

La solución es encontrar la trayectoria más corta. Esperando que dicha trayectoria contenga pocos enlaces no congestionados; de esta forma los enlaces menos congestionados son candidatos a pertenecer a la ruta. Hay algoritmos de ruteo especializados que también pueden permitir que la longitud de cada enlace cambie en el tiempo, dependiendo del nivel de tráfico de cada enlace. De esta forma un algoritmo de ruteo se debe adaptar a sobrecargas temporales y rutear paquetes alrededor de nodos congestionados. Dentro de este contexto, el algoritmo de ruta más corta para ruteo opera continuamente, determinando la trayectoria más corta con longitudes que varían en el tiempo **[6]**.

El problema de la ruta más corta se puede resolver utilizando programación lineal, sin embargo, debido a que el método simplex es de complejidad exponencial, se prefiere

utilizar algoritmos que aprovechen la estructura en red que se tiene para estos problemas.

Para ello, el algoritmo mantiene un conjunto S de nodos cuyos pesos finales de camino mínimo desde el nodo origen ya han sido determinados.

El algoritmo de Dijkstra aunque fue diseñado para encontrar la ruta más corta se puede transformar fácilmente para encontrar la ruta más larga (camino máximo), cambiando simplemente su función objetivo **[6]**. Del mismo modo, se encuentra el árbol máximo desde un nodo origen.

El camino máximo estará formado por tareas críticas (nodos) cuya duración (coste del arco sucesor) determina la duración total de un proyecto. Si una tarea crítica se retrasa o su duración cambia durante la realización del proyecto, afectaría directamente a la duración total del proyecto y a su fecha de finalización.

Encontrar el camino máximo de la planificación de un proyecto es lo mismo que encontrar el camino más largo desde el nodo inicial (tarea inicial) al nodo final (última tarea); esto es, la mínima cantidad de tiempo necesaria para finalizar un proyecto.

Algoritmo de Bellman-Ford (Camino Mínimo – Camino Máximo)

El algoritmo de Bellman-Ford (algoritmo de Bell-End-Ford), fue desarrollado por Richard Bellman, Samuel End y Lester Ford. Soluciona el problema de la ruta más corta o camino mínimo desde un nodo origen, de un modo más general que el Algoritmo de Dijkstra, ya que permite valores negativos en los arcos.

El algoritmo devuelve un valor booleano si encuentra un circuito o lazo de peso negativo. En caso contrario calcula y devuelve el camino mínimo con su coste.

Para cada vértice v perteneciente a V, se mantiene el atributo d[v] como cota superior o coste del camino mínimo desde el origen s al vértice v **[7]**.

Si el grafo contiene un ciclo de coste negativo, el algoritmo lo detectará, pero no encontrará el camino más corto que no repite ningún vértice. La complejidad de este problema es al menos la del problema del camino más largo de complejidad NP-Completo.

El problema de la ruta más corta puede ser transformado en el de ruta más larga cambiando el signo de los costes de los arcos.
De manera alternativa se puede transformar también cambiando los procesos de inicialización y relajación. En este caso el problema es inconsistente para circuitos de peso positivo.

Algoritmo de Kruskal

Joseph B. Kruskal investigador del Math Center (Bell-Labs), que en 1956 descubrió su algoritmo para la resolución del problema del Árbol de coste total mínimo (minimum spanning tree - MST) también llamado árbol recubridor euclídeo mínimo **[8]**.
Dado un grafo G con nodos conectados por arcos con peso (coste o longitud): el peso o coste total de un árbol será la suma de pesos de sus arcos. Obviamente, árboles diferentes tendrán un coste diferente. El problema es entonces ¿cómo encontrar el árbol de coste total mínimo?
Una manera de encontrar la solución al problema del árbol de coste total mínimo, es la enumeración completa. Aunque esta forma de resolución es eficaz, no se puede considerar un algoritmo, y además no es nada eficiente.
Este problema fue resuelto independientemente por Dijkstra (1959), Kruskal (1956) y Prim (1957) y la existencia de un algoritmo polinomial (que todos ellos demostraron) es una grata sorpresa, debido a que un grafo con N vértices puede llegar a contener N^{N-2} sub-árboles. A lo largo de la historia se ha hecho un gran esfuerzo para encontrar un algoritmo rápido para este problema. El algoritmo de Kruskal es uno de los más fáciles de entender y probablemente el mejor para resolver problemas a mano.
El algoritmo se basa en una propiedad clave de los árboles que permite estar seguros de si un arco debe pertenecer al árbol o no, y usar esta propiedad para seleccionar cada arco. Nótese en el algoritmo, que siempre que se añade un arco (U,V), éste será siempre la conexión más corta (menor coste) alcanzable desde el nodo U al resto del grafo G. Así que por definición éste deberá ser parte del árbol **[9]**.
Este algoritmo es de tipo greedy, ya que a cada paso, éste selecciona el arco más barato y lo añade al sub-grafo. Este tipo de algoritmos pueden no funcionar para

resolver otro tipo de problemas, por ejemplo para encontrar la ruta más corta entre los nodos A y B **[10]**.

Para simplificar, se asumirá que existe un único árbol de coste total mínimo, aunque en muchos problemas puede existir más de una solución óptima de igual valor total mínimo.

Algoritmo de Prim

Robert Prim en 1957 descubrió un algoritmo para la resolución del problema del Árbol de coste total mínimo (minimum spanning tree - MST).

Consiste en dividir los nodos de un grafo en dos conjuntos: procesados y no procesados. Al principio, hay un nodo en el conjunto procesado que corresponde al equipo central; en cada interacción se incrementa el grafo de procesados en un nodo (cuyo arco de conexión es mínimo) hasta llegar a establecer la conexión de todos los nodos del grafo a procesar **[11]**.

Este algoritmo ha sido aplicado para hallar soluciones en diversas áreas (diseño de redes de transporte, diseño de redes de telecomunicaciones - TV por cable, sistemas distribuidos, interpretación de datos climatológicos, visión artificial - análisis de imágenes - extracción de rasgos de parentesco, análisis de clusters y búsqueda de superestructuras de quasar, plegamiento de proteínas, reconocimiento de células cancerosas, y otros). También se ha utilizado para encontrar soluciones aproximadas a problemas NP-Hard como el del 'viajante de comercio'.

Algunos trabajos han comparado la eficiencia entre los algoritmos de Kruskal y de Prim: Sea *a* el número de arcos y *n* el número de nodos. El algoritmo de Kruskal requiere un tiempo que está en O(a log n) **[9]**. Para un grafo denso, a tiende a n(n-1)/2, por lo que el algoritmo requiere un tiempo que está en $O(n^2 \log n)$, y el algoritmo de Prim puede ser mejor $O(n^2)$. En un grafo disperso, a tiende a n-1, por lo que el algoritmo de Kruskal requiere un tiempo que está en O(n log n) y el algoritmo de Prim es menos eficiente. Sin embargo, si el algoritmo de Prim se implementa con montículos, el tiempo requerido por este algoritmo está, como el de Kruskal, en O(a log n).

- Si a es aproximadamente igual a n (grafo con pocos arcos) conviene usar Kruskal

- Si a es aproximadamente igual a n^2 (grafo denso) conviene usar Prim

En cualquier caso, la complejidad del algoritmo de Kruskal depende de la técnica de ordenación empleada. El algoritmo de Prim requiere más memoria que el algoritmo de Kruskal **[11]**.

1.3 – Descripción de los sistemas existentes

En la actualidad el gran avance de las tecnologías de la informática, ha permitido desarrollar infinidad de sistemas que optimizan el trabajo con Teoría de Redes, durante la revisión bibliográfica para la realización del presente trabajo se encontraron algunas herramientas que se corresponden en parte con el objeto de estudio del mismo pero no resuelven a totalidad las necesidades del usuario que las consulta. A continuación se muestran los más representativos:

1.3.1 – QSB

QSB es un sistema desarrollado en Pascal, el cual ayuda en la toma de decisiones, contiene herramientas muy útiles para resolver distintos tipos de problemas en el campo de la investigación operativa **[12]**. Programas específicos para resolver el problema del transbordo, el problema del transporte, el de asignación, el problema del camino más corto, flujo máximo, árbol generador y problema del agente viajero. Su principal problema radica en el hecho que no posee representación gráfica alguna, por lo que su comprensión radica en el entendimiento de los resultados literales. Tampoco desarrolla la solución de los algoritmos de forma óptima, afectando la complejidad de los mismos, resolviendo los problemas pero no de mejor manera.

1.3.2 – WinQSB

WinQSB es un sistema interactivo de ayuda a la toma de decisiones que contiene herramientas muy útiles para resolver distintos tipos de problemas en el campo de la investigación operativa. El sistema está formado por distintos módulos, uno para cada tipo de modelo o problema **[13]**. Entre ellos se destacará el que se ocupa de la Teoría de Redes:

Network Modeling (NET): el mismo resuelve los problemas que se presentaron en su versión en consola, de no representar gráficamente el grafo desarrollado pero mantiene la solución de los problemas mediante métodos de programación lineal, los cuales no son tan eficientes como los resultados que nos brindan los algoritmos.

WinQSB utiliza los mecanismos típicos de la interfaz de Windows, es decir, ventanas, menús desplegables, barras de herramientas, etc. Por lo tanto el manejo del programa es similar a cualquier otro que utilice el entorno Windows. Pero en la representación de grafos superiores a 5 nodos se vuelve de muy difícil comprensión ya que representa automáticamente los mismos y no ofrece posibilidad alguna de modificación por el usuario. Además en el tratamiento de los problemas antes mencionados, se pierde el concepto de Redes Orientadas o no Orientadas, creando dudas sobre su utilización a todo aquel que lo consulte en caso de no poseer los conocimientos sobre la Teoría de Redes.

1.3.3 – Qmwin2

Qmwin2 es un sistema desarrollado por la Universidad de Guayaquil, su objetivo fundamental radica en la resolución y descripción de algunos algoritmos en la Teoría de Redes **[14]**. Este software a pesar de que describe en su desarrollo la resolución del problema no es capaz de representar gráficamente la solución propuesta. El proceso de entrada de datos es mediante ramas por lo que el usuario debe introducir uno por uno todos los arcos que componen el grafo. Tiene límite de hasta 50 nodos, por lo que no es un software para grafos de gran volumen.

1.3.4 – POM

POM es un sistema desarrollado por Howard J. Weiss en una universidad de Estados Unidos, en su contenido, sobre el tema Teoría de Redes, posee solamente un módulo de transporte, en el que tanto la entrada como la representación de los datos se realiza de manera literal y de muy alta complejidad para un usuario de nivel básico en el tema. No posee representación gráfica **[15]**.

1.4 – Tendencias, metodologías, lenguaje, herramientas y tecnologías actuales

Las etapas en el desarrollo de una herramienta de software deben estar soportadas por las indicaciones de algunas metodologías para poder garantizar su calidad y eficiencia. Conjuntamente con esto se debe tener presente que antes de llevarse a cabo cualquier aplicación es necesario realizar un estudio de las tecnologías actuales, con el objetivo de seleccionar la mas adecuada según los requerimientos que se debe cumplir para el desarrollo de la nueva propuesta. Otra elemento importante a analizar es el estado de las herramientas afines a escala global.

1.4.1 – Tendencias actuales a considerar.

Programación Orientada a Objetos (POO)

La programación orientada a objetos (POO), por su parte, fue creada precisamente para superar las insuficiencias de la programación estructurada, aprovechando sus mejores ideas. Su principal preocupación durante el desarrollo de programas es determinar los objetos que representarán, de una forma más adecuada, los elementos presentes en un problema, una vez determinados estos, se pasa a determinar cuáles son sus características o atributos principales y cuáles son las acciones (procedimientos) que son realizados con tales atributos y que por tanto caracterizan su comportamiento.

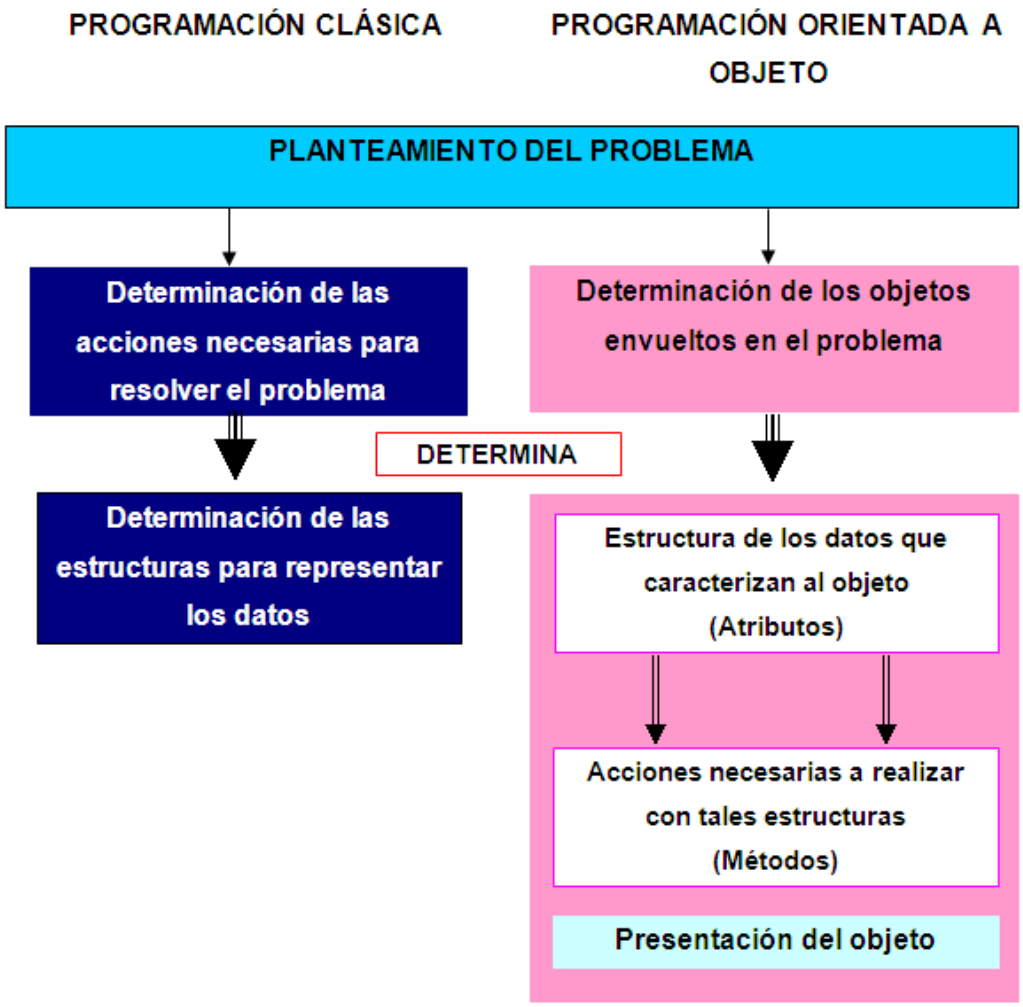


Figura IV. Características de la Programación Clásica y la POO.

Por ello la programación orientada a objeto tiene como objetivos facilitar el desarrollo y comprensión de programas de gran porte y posibilitar la reutilización de código. Las principales características que identifican a un lenguaje orientado a objeto son los conceptos de clases, herencia y polimorfismo.

A diferencia de otros paradigmas de programación, el paradigma orientado a objetos es tanto un estilo de programación como una metodología, que cuenta como principales características:

1. Flexibilidad y adaptabilidad.
2. Reutilización del software.
3. Notación e integración consistente de las fases del diseño.

En la POO estas características también deben estar presentes y esta forma de programar por sus características y mecanismos ayuda a conservarlas y las afianza. Estas características están muy interrelacionadas con las cuatro propiedades básicas del modelo de objetos, que son la abstracción, encapsulación, modularidad y jerarquía, tal y como se presenta a continuación.

- Claridad

La claridad del código fuente de los programas ayuda a la comprensión no sólo de quienes los confeccionan, sino también de otras personas que le pueden dar mantenimiento. Muchas veces un programa se ve limitado porque no hay quien lo entienda y pueda mejorarlo o hacer otras versiones y otras veces no se puede reusar su código por el mismo motivo. A la claridad ayuda, poner nombres adecuados a las rutinas y variables, ni muy largos ni muy cortos y carentes de sentido, también ayuda la modularidad, la abstracción y el encapsulamiento.

- Modularidad

La modularidad, la abstracción, y el encapsulamiento están estrechamente relacionadas entre sí.

La **modularidad** consiste en dividir las tareas del programa de una forma y lógica razonable, generalmente siguiendo un diseño t*op-down*, desde lo más general a lo más específico. Un programa con modularidad puede dividirse en módulos separados a los cuales se les puede dar mantenimiento también por separado.

La **abstracción** es la propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las restantes características (no esenciales).

Una abstracción se centra en la vista externa de un objeto, de modo que sirva para separar el comportamiento esencial de un objeto de su implementación. Definir una abstracción significa describir una entidad del mundo real, no importa lo compleja que pueda ser, y a continuación utilizar esta descripción en un programa.

El elemento clave de la programación orientada a objetos es la **clase**. Una **clase** se puede definir como una descripción abstracta de un grupo de objetos, cada uno de los cuales se diferencia por su *estado* específico y por la posibilidad de realizar una serie de *operaciones*.

La idea de escribir programas definiendo una serie de abstracciones no es nueva, pero el uso de clases para gestionar dichas abstracciones en lenguajes de programación ha facilitado considerablemente su aplicación.

La **encapsulación** *o* **encapsulamiento** es la propiedad que permite asegurar qué contenido de la información de un objeto está oculta al mundo exterior: El objeto A no conoce lo que hace el objeto B, y viceversa. La encapsulación (también se conoce como *ocultamiento de la información),* en esencia, es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales.

La encapsulación permite la división de un programa en módulos. Estos módulos se implementan mediante clases, de forma que una clase representa la encapsulación de una abstracción. En la práctica, esto significa que cada clase debe tener dos partes: una interface y una implementación. La *interface* de una clase captura sólo su vista externa y la *implementación* contiene la representación de la abstracción, así como los mecanismos que realizan su comportamiento.

- Protección

La protección consiste en cuidar de las manos inexpertas las partes de los programas más sensibles o que no se desean que se tenga acceso. En C esto se logra colocando en los ficheros de encabezamiento (.h) declaraciones de funciones y de los datos y ofrecer los módulos compilados (.obj) donde están las definiciones de las funciones y los datos.

- Reusabilidad

La reusabilidad consiste en hacer los programas de tal manera que sus módulos, estructuras de datos y funciones, puedan usarse en otros programas con un mínimo de cambios. A esto ayudan las antes mencionadas: modularidad, abstracción, y encapsulamiento.

- Extensibilidad

La extensibilidad es parecida a la reusabilidad, pero persigue otro objetivo. Cuando se diseña un programa este debe ser fácilmente modificable de manera que se le puedan agregar nuevas funcionalidades, es decir, extender el programa, con un mínimo de cambios en el código fuente.

1.4.2 – Metodología de Desarrollo de Software

1.4.2.1 – Proceso Unificado de Desarrollo (RUP)

En la actualidad existe una tendencia a la creación de un software más grande y complejo, esto impulsado sin duda alguna por el vertiginoso avance de la tecnología que pone al alcance de todos máquinas más potentes con una capacidad de procesamiento y almacenamiento de información que crece casi sin límites. Hoy en día se necesita un software más complejo que satisfaga la demanda de los usuarios, que cada día son mayores. Ante estos retos los desarrolladores de software necesitan nuevos métodos para el desarrollo de aplicaciones informáticas.

La comunidad de desarrolladores de software necesita una forma coordinada de trabajar. Necesitan un proceso que integre las múltiples facetas de desarrollo. Este proceso debe: **[18]**

- Proporcionar una guía para ordenar las actividades de un equipo.
- Dirigir las tareas de cada desarrollador por separado y del equipo como un todo.
- Ofrecer criterios para el control y la medición de los productos y actividades del proyecto.

La actual solución al problema del software lo constituye el Proceso Unificado de Desarrollo (RUP, siglas en ingles de Rational Unified Process). El proceso Unificado surge en 1998 y se identifica por ser un proceso de desarrollo de software definido por tres fases claves.

Dirigido por Casos de Uso: Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Estos guían el proceso de desarrollo incluyendo el diseño, la implementación y las pruebas del sistema. "Los casos de uso no solo inician el proceso de desarrollo sino que le proporcionan un hilo conductor. Dirigido por casos de uso quiere decir que el proceso de desarrollo sigue un hilo – avanza a través de una serie de flujos de trabajos que parten de los casos de uso". **[18]**

Centrado en la Arquitectura: La arquitectura invoca los elementos más significativos del sistema y está influenciada entre otros por plataformas de software, sistemas operativos, gestores de base de datos, protocolos, consideraciones generales y requisitos no funcionales. La arquitectura abarca decisiones importantes sobre la

organización del sistema, los elementos estructurales, sus interfaces y comportamientos. Es de resaltar la interacción que debe existir entre arquitectura y caso de uso, estos dos aspectos deben evolucionar en paralelo **[18]**.

Iterativo e Incremental: para hacer más manejable un proyecto se recomienda dividirlo en ciclos. Para cada ciclo se establecen fases de referencia, cada una de las cuales debe ser considerada como un mini - proyecto, cuyo núcleo fundamental está constituido por una o más iteraciones de las actividades principales básicas de cualquier proceso de desarrollo **[18]**.

Se puede decir que el Proceso unificado atraviesa por una serie de ciclos, los cuales se dividen en las fases:

Inicio: En esta fase inicial se desarrolla una descripción del producto final partiendo de una buena idea. Es en esta fase donde se definen las principales funciones del sistema para sus usuarios, se obtiene una vista preliminar de la arquitectura del sistema y se define cuál es el plan del proyecto y su costo.

Elaboración: En esta fase se especifican los casos de uso y se diseña la arquitectura del sistema, la cual se expresa en forma de vista de todos los modelos, los que en su totalidad conforman el sistema entero.

Construcción: Es una fase de desarrollo donde se emplean la mayor cantidad de recursos para poder entregar a los usuarios un producto.

Transición: En esta fase el producto es probado por un grupo pequeño de usuarios con experiencias para informar las deficiencias y que estas son corregidas. En esta fase donde se forma al cliente, se proporciona una línea de ayuda y asistencia y se corrigen los defectos encontrados tras la entrega.

Cada fase se divide en iteraciones o mini – proyectos los cuales atraviesan por los flujos de trabajo: requisitos, análisis, diseño, implementación y prueba.

Es acertado decir que el Proceso Unificado guía a los equipos de proyecto en cómo administrar el desarrollo iterativo de un modo controlado mientras se balancean los requerimientos del negocio, el tiempo de desarrollo y los riesgos del proyecto. El proceso describe los diversos pasos involucrados en la captura de los requerimientos y el establecimiento de una guía arquitectónica, para diseñar y probar el sistema. Además el Proceso Unificado es soportado por herramientas que automatizan entre otras cosas,

el modelado visual, la administración de cambios y las pruebas. [Zalazar, 2003] Estas razones hacen del RUP la metodología seleccionada para el desarrollo de numerosas aplicaciones.

1.4.3 – Lenguajes

1.4.3.1 – UML

El Proceso Unificado utiliza el Lenguaje Unificado de Modelado (Unified Modeling Lenguaje, UML) para preparar todos los esquemas de un sistema de software **[19]**. El Lenguaje UML fue creado por un grupo de estudiosos de la Ingeniería del Software en el año 1995 y es un lenguaje gráfico de modelado orientado a objetos. Este lenguaje tiene una sintaxis y una semántica bien definidas, sirviendo además para todas las etapas de desarrollo. UML ofrece un estándar para describir un plano del sistema (modelo), incluyendo aspectos concretos como expresiones de lenguajes de programación, esquemas de base de datos y componentes de software reutilizables.

Estereotipo UML

Los estereotipos son el mecanismo de extensibilidad incorporado más utilizado dentro de UML. Un estereotipo representa una distinción de uso. Puede ser aplicado a cualquier elemento de modelado, incluyendo clases, paquetes, relaciones de herencia, etc. Por ejemplo, una clase con estereotipo \' actor \' es una clase usada como un agente externo en el modelado de negocio. Una clase patrón es modelada como una clase con estereotipo parametrizado, lo que significa que puede contener parámetros.

¿Por qué es importante UML?

Hoy en día, UML está consolidado como el lenguaje estándar en el análisis y diseño de sistemas de cómputo. Mediante UML es posible establecer la serie de requerimientos y estructuras necesarias para plasmar un sistema de software previo al proceso intensivo de escribir código.

En otros términos, así como en la construcción de un edificio se realizan planos previo a su construcción, en Software se deben realizar diseños en UML previa codificación de un sistema, ahora bien, aunque UML es un lenguaje, éste posee más características visuales que programáticas, las mismas facilitan a integrantes de un equipo

multidisciplinario participar e intercomunicarse fácilmente, estos integrantes siendo los analistas, diseñadores, especialistas de área y desde luego los programadores **[20]**.

Complejidad / Objetos UML

Entre más complejo es el sistema que se desea crear más beneficios presenta el uso de UML, las razones de esto son evidentes, sin embargo, existen dos puntos claves: El primero se debe a que mediante un plano/visión global resulta más fácil detectar las dependencias y dificultades implícitas del sistema, y la segunda razón radica en que los cambios en una etapa inicial (Análisis) resultan más fáciles de realizar que en una etapa final de un sistema como lo sería la fase intensiva de codificación.

Puesto que UML es empleado en el análisis para sistemas de mediana y alta complejidad, era de esperarse que su base radique en otro paradigma empleado en diseños de sistemas de alto nivel que es la orientación a objetos, por lo que para trabajar en UML puede ser considerado un pre-requisito tener experiencia en un lenguaje orientado a objetos **[20]**.

1.4.3.2 – C++

El lenguaje C++ se comenzó a desarrollar en 1980. Su autor fue B. Stroustrup, formado en la ATT. Al comienzo era una extensión del lenguaje C que fue denominada C with classes. Este nuevo lenguaje comenzó a ser utilizado fuera de la ATT en 1983. El nombre C++ es también de ese año, y hace referencia al carácter del operador incremento de C (++). Ante la gran difusión y éxito que iba obteniendo en el mundo de los programadores, la ATT comenzó a estandarizarlo internamente en 1987. En 1989 se formó un comité ANSI (seguido algún tiempo después por un comité ISO) para estandarizarlo a nivel americano e internacional.

En la actualidad, el C++ es un lenguaje versátil, potente y general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original **[21]**. La evolución de C++ ha continuado con la aparición de Java, un lenguaje creado simplificando algunas cosas de C++ y añadiendo otras, que se utiliza para realizar aplicaciones en Internet.

Hay que señalar que el C++ ha influido en algunos puntos muy importantes del ANSI C, como por ejemplo en la forma de declarar las funciones, en los punteros a void, etc. En efecto, aunque el C++ es posterior al C, sus primeras versiones son anteriores al ANSI C, y algunas de las mejoras de éste fueron tomadas del C++.

El C++ es a la vez un lenguaje procedural (orientado a algoritmos) y orientado a objetos. Como lenguaje procedural se asemeja al C y es compatible con él, aunque ya se ha dicho que presenta ciertas ventajas. Como lenguaje orientado a objetos se basa en una filosofía completamente diferente, que exige del programador un completo cambio de mentalidad. Las características propias de la Programación Orientada a Objetos (Object Oriented Programming) de C++ son modificaciones mayores que sí cambian radicalmente su naturaleza anterior **[21]**.

1.4.4 – Herramientas de desarrollo

1.4.4.1 – Rational Rose

La complejidad de los proyectos de software hoy en día, el constante cambio de requerimientos y la falta de una documentación durante el proceso de desarrollo provoca que los proyectos se retrasen en tiempo y se incrementen en costo. La solución a esta problemática es implantar una arquitectura de desarrollo que permita hacer seguimiento a los proyectos desde su etapa de requerimientos, hasta su implantación. Rational ofrece un Proceso Unificado (RUP) para el desarrollo de los proyectos de software, desde la etapa de Ingeniería de Requerimientos hasta la etapa de pruebas. Para cada una de estas etapas existe una herramienta que ayuda en la administración de los proyectos, Rose es la herramienta de Rational para la etapa de análisis y diseño de sistemas **[22]**.

Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Una de las grandes ventajas de Rose es que utiliza la notación estándar en la arquitectura de Software (UML), la cual permite a los arquitectos de software y desarrolladores visualizar el sistema completo utilizando un lenguaje común. Otra ventaja de Rose es que los diseñadores pueden modelar sus componentes e interfaces en forma individual y luego unirlos con otros componentes del proyecto.

Además Rose soporta la construcción de componentes en lenguajes como C++, Visual Basic, Java, Ada, genera IDL's para aplicaciones CORBA. Por todo lo anterior Rose es la herramienta de Análisis, Diseño, Modelado y Construcción de software Orientado a Objetos líder en el mercado y es por todo esto que fue escogida para ser utilizada en este trabajo **[22]**.

1.4.4.2 – Visual C++

Microsoft Visual C++ 2008 proporciona un entorno de desarrollo eficaz y flexible para crear aplicaciones basadas en Microsoft Windows y en Microsoft .NET. Se puede utilizar como un sistema de desarrollo integrado o como un conjunto de herramientas individuales. Visual C++ se compone de estos componentes: **[23]**

- El compilador de Visual C++ 2008 tiene nuevas características que ayudan a los desarrolladores que trabajan con plataformas de equipo virtual como Common Language Runtime (CLR). Hay ahora compiladores diseñados para x64 e Itanium. El compilador sigue admitiendo directamente equipos x86 de destino, y optimiza el rendimiento para ambas plataformas.

- Las bibliotecas de Visual C++ 2008. Incluyen las siguientes bibliotecas: Active Template Library (ATL) estándar del sector, Microsoft Foundation Class (MFC), la Biblioteca estándar de C++ y la Biblioteca en tiempo de ejecución de C (CRT), que se ha ampliado para proporcionar alternativas de seguridad mejorada a funciones que sufrían problemas de seguridad conocidos. Una nueva biblioteca, la Biblioteca de compatibilidad de C++, está diseñada para simplificar programas destinados al CLR.
- Las bibliotecas del entorno de desarrollo de Visual C++ 2008. Aunque las herramientas y bibliotecas del compilador de C++ se pueden utilizar desde la línea de comandos, el entorno de desarrollo proporciona una eficaz ayuda para la administración y configuración de proyectos (incluida una mejor compatibilidad con grandes proyectos), edición y exploración del código fuente y herramientas de depuración. Este entorno también admite IntelliSense, que realiza sugerencias contextuales y bien fundamentadas cuando se crea el código.

Además de las aplicaciones de interfaz gráfica de usuario convencionales, Visual C++ permite a los desarrolladores generar aplicaciones Web, aplicaciones smart-client basadas en Windows y soluciones para dispositivos móviles thin-client y smart-client. C++ es el lenguaje de nivel de sistemas más popular del mundo, y Visual C++ ofrece a los desarrolladores una herramienta universal con la que generar software **[23]**.

1.4.5 – Tecnologías actuales

1.4.5.1 – Librería QT

Qt es un producto de la empresa noruega de software Trolltech AS, esta empresa se dedica a desarrollar librerías y herramientas de desarrollo de software, además es experta en servicios de consultoría. Qt es un conjunto de librerías multi-plataforma para el desarrollo del esqueleto de aplicaciones GUI, escritas en código C++.

Qt comenzó a distribuirse comercialmente en 1996 y desde entonces ha sido la base para numerosas aplicaciones incluyendo la popular interfaz gráfica para Linux llamada KDE, disponible en todas las grandes distribuciones de Linux. Es una librería para la creación de interfaces gráficas. Se distribuye bajo una licencia libre GPL (o QPL) que permite incorporar QT en aplicaciones open-source. Se encuentra disponible para una gran número de plataformas: Linux, MacOs X, Solaris, HP-UX, UNIX con X11. Además, existe también una versión para sistemas empotrados. Es orientado a objetos, lo que facilita el desarrollo de software. El lenguaje para el que se encuentra disponible es C++ aunque han aparecido versiones de prueba a otros lenguajes como Python o Perl. Es una librería que se basa en los conceptos de widgets (objetos), Señales-Slots y Eventos. Las señales y los slots son el mecanismo para que unos widgets se comuniquen con otros. Los mismos pueden contener cualquier número de hijos. El widget "top-level" puede ser cualquiera, sea ventana, botón, etc. Algunos atributos como el texto de etiquetas se modifican de modo similar al lenguaje html. Además proporciona otras funcionalidades como:

- Librerías básicas -> Entrada/Salida, Manejo de Red, XML
  Interface con bases de datos -> Oracle, MySQL, PostgreSQL, ODBC
- Plugins, librerías dinámicas (Imágenes, formatos, gráficos, etc.)
  Unicode, Internacionalización.

1.5 – Conclusiones

A partir de lo analizado en este capítulo se concluye que:

- No existe un sistema, que dé solución gráfica a los algoritmos, para una mayor comprensión e interpretación de la solución.
- La programación orientada a objetos, como tendencia actual a considerar, es la de mayor aceptación por programadores y desarrolladores.
- Se selecciona la metodología RUP, como guía en el proceso de desarrollo del software propuesto, así como el uso de UML como lenguaje de modelado por ser un estándar internacional. Para la representación gráfica la herramienta Rational Rose.
- Como lenguaje de programación seleccionado está C++, además de ser considerado como el mejor lenguaje concebido de la programación orientada a objetos, se apoya su selección en la dependencia de la librería Qt, ya que la misma logra su mejor funcionamiento en este lenguaje.
- Como herramientas de desarrollo de la aplicación se optó por Visual C++ por tener funcionalidades superiores a cualquier herramienta que trabaje con el lenguaje C++ las cuales fueron expuestas anteriormente.

Capítulo 2 – Descripción y construcción de la solución propuesta.

2.1 – Introducción

Existen al menos dos aproximaciones fundamentales para expresar el contexto de una aplicación: modelado del dominio y modelado del negocio. Seleccionándose como punto de partida el modelo del dominio, se obtiene como resultado una descripción de entidades y los conceptos principales, así como elementos necesarios para la buena comprensión del problema

En el presente capítulo se recogen además, la descripción de la solución propuesta utilizando algunos de los artefactos que propone la Metodología RUP. Los artefactos referidos son: el Modelo del Dominio, los Requerimientos Funcionales y No Funcionales, el Diagrama de Casos de Uso y la descripción de cada uno.

2.2 – Modelación del dominio

El modelo del dominio permite comprender el contexto del sistema a través de la representación de los objetos más importantes en el mismo, dichos objetos pueden representar entidades físicas (cosas que existen), así como eventos que suceden en el entorno del sistema. El mismo se describe a través de diagramas de UML (específicamente diagramas de clases). A este nivel no se representan clases del software con atributos y responsabilidades, sino clases que describen los conceptos fundamentales que se manejan entre desarrolladores, clientes y usuarios. Las tres formas típicas de aparición de las clases del dominio son: **[18]**

- Objetos del negocio que representan cosas manipulables en el negocio.
  Objetos del mundo real y conceptos de los que el sistema debe hacer un seguimiento.
- Sucesos que ocurrirán o han ocurrido.

A continuación se muestra la representación del modelo del dominio del sistema:

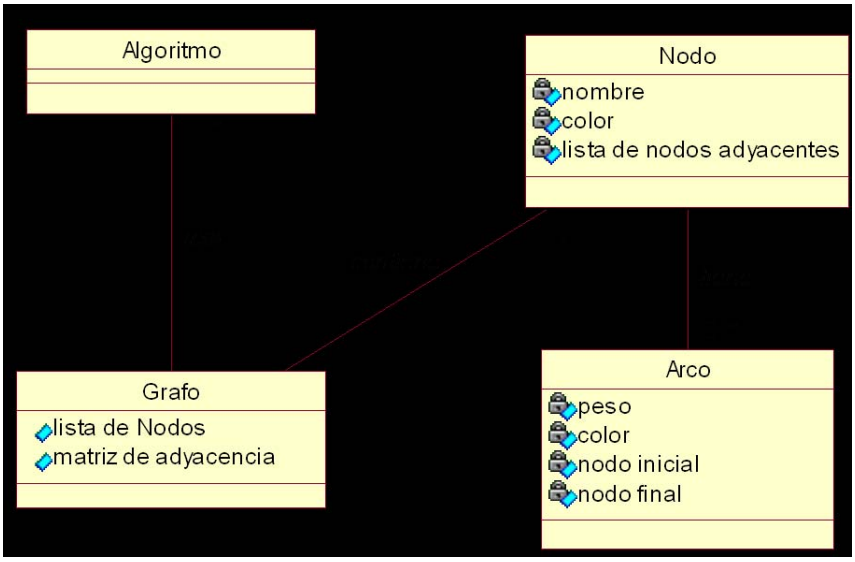


Figura V. Modelo del dominio.

2.3 – Requerimientos funcionales.

Los requerimientos funcionales permiten expresar una especificación más detallada de las responsabilidades del sistema que se propone. Ellos permiten determinar, de una manera clara, lo que debe hacer el mismo **[18]**.

Los requerimientos funcionales del producto informático propuesto son los siguientes:

1. Crear un nuevo documento.
2. Abrir un fichero con la información de un grafo.
3. Crear un fichero con extensión propia, con la información del grafo que se elaboró.
4. Salvar una copia de resguardo del grafo que se está elaborando en un fichero con extensión propia.
5. Insertar un nodo.
6. Modificar un nodo.
7. Eliminar un nodo.
8. Insertar un arco orientado entre dos nodos.
9. Modificar un arco orientado entre dos nodos.
10. Eliminar un arco orientado entre dos nodos.
11. Insertar un arco no orientado entre dos nodos.

12. Modificar un arco no orientado entre dos nodos.
13. Eliminar un arco no orientado entre dos nodos.
14. Aplicar a un grafo el algoritmo Dijkstra – Camino mínimo.
15. Aplicar a un grafo el algoritmo Dijkstra – Camino máximo.
16. Aplicar a un grafo el algoritmo Bellman Ford – Camino mínimo.
17. Aplicar a un grafo el algoritmo Bellman Ford – Camino máximo.
18. Aplicar a un grafo no orientado el algoritmo de Prim.
19. Aplicar a un grafo no orientado el algoritmo de Kruskal.
20. Modificar el grafo para la inicialización de los algoritmos.
21. Mostrar las iteraciones por las que transita el grafo al aplicarle los algoritmos.
22. Modificar el ambiente de trabajo.
23. Salir de la aplicación.

2.4 – Requerimientos no funcionales.

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener, como restricciones del entorno o de implementación, rendimiento, etc **[18]**.

Requerimientos de apariencia o interfaz externa

En el diseño de la interfaz de este producto se tuvo como premisa la idea de que el software fuese lo más legible y sencillo de usar, siempre que no se afectase ninguna de las funcionalidades del mismo. Un diseño de este tipo persigue el objetivo de que el producto sea aceptado y asimilado rápidamente por los usuarios del mismo, independientemente de que tengan habilidades en el manejo de aplicaciones informáticas.

Requerimientos de Usabilidad

Es importante aclarar que el producto está dirigido a usuarios con los conocimientos mínimos necesarios en el campo de acción de los modelos de redes.

La utilización de la aplicación no requiere de gran experiencia en el uso de las computadoras por parte de los usuarios, además se utilizó un lenguaje familiar para los mismos lo que facilita la utilización del software.

Requerimientos de Soporte

Con el fin de lograr el mejoramiento del software y su evolución se desarrolló previendo las facilidades para su mantenimiento. Con este fin se garantiza toda la documentación técnica, así como el código fuente, el cual está escrito sobre la base de estándares adecuadamente definidos y documentados.

Otro aspecto que se tuvo en cuenta durante el desarrollo de la aplicación fue la posibilidad de extender su uso gracias a una arquitectura que permite adaptar el mismo a otros algoritmos que se le integren.

Requerimientos de Software

El software está realizado para sistema operativo Windows, se requiere Windows 98 o superior.

Requerimientos de Hardware

Para la explotación del sistema los requerimientos mínimos de hardware son:

- Procesador Pentium a 133MHz de velocidad.
- 64 Mb de memoria RAM.
- 100 Mb de espacio disponible en disco.

2.5 – Descripción del sistema propuesto.

2.5.1 – Concepción general del sistema

El resultado que se desea alcanzar fruto de esta investigación es la obtención de un producto informático que dé solución a algunos problemas de la Teoría de Redes, que desarrolle un nuevo sistema gráfico de introducción de datos y obtención de soluciones, contribuyendo así a mejorar la aplicación de estos algoritmos para estudiantes e investigadores. Esta herramienta permitirá a los usuarios el análisis e interpretación de los resultados devueltos por la ejecución de alguno de los algoritmos.

Con este software, se trata de eliminar algunos de los inconvenientes de las herramientas existentes, analizadas como antecedentes en el *Capítulo 1: Fundamentación Teórica* y se le adicionan funcionalidades propias que permitirán la construcción de los grafos o redes de una manera más fácil y rápida.

2.5.2 – Definición de los actores y casos de uso del sistema

Actores del sistema.

Un actor no es más que un conjunto de roles que los usuarios de Casos de Uso desempeñan cuando interaccionan con estos Casos de Uso. Los actores representan terceros fuera del sistema que colaboran con el mismo. Una vez que hemos identificado los actores del sistema, tenemos identificado el entorno externo del sistema **[18]**.

Nombre del Actor	Descripción
Usuario	Cualquier usuario que interactúe con el sistema, puede ser un estudiante, un profesor o cualquier otra persona que necesite elaborar un grafo e interactuar con los algoritmos implementados. Este usuario tendrá acceso a todos los requerimientos funcionales del sistema.

Casos de Uso del Sistema.

Los actores interactúan y usan el sistema a través de Casos de Uso. Los Casos de Uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario**.** De manera más precisa, un Caso de Uso especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia **[18]**.

En el presente trabajo los casos de uso del sistema quedan representados por:

1. Crear nuevo documento.
2. Abrir documento.
3. Salvar documento.
4. Gestionar nodo.
5. Gestionar un arco orientado.
6. Gestionar un arco no orientado.
7. Aplicar Dijkstra Mínimo.
8. Aplicar Dijkstra Máximo.
9. Aplicar Bellman Ford Mínimo.
10. Aplicar Bellman Ford Máximo.
11. Aplicar Prim.

12. Aplicar Kruskal.
13. Inicializar algoritmos.
14. Mostrar iteraciones.
15. Modificar entorno.
16. Salir de la aplicación.

2.5.3 – Diagrama de casos de uso del sistema

Las formas en que los actores utilizan el sistema se representa como casos de uso, que no son más que un grupo de funcionalidades del sistema que devuelven un resultado de valor al actor. O sea que es una secuencia de acciones que el sistema lleva a cabo cuando el actor interactúa con él **[18]**.

La manera de representar la interacción entre actores y sistema es a través del diagrama de casos de uso del sistema, en el cual se describe la forma en que cada usuario se relaciona con el software.

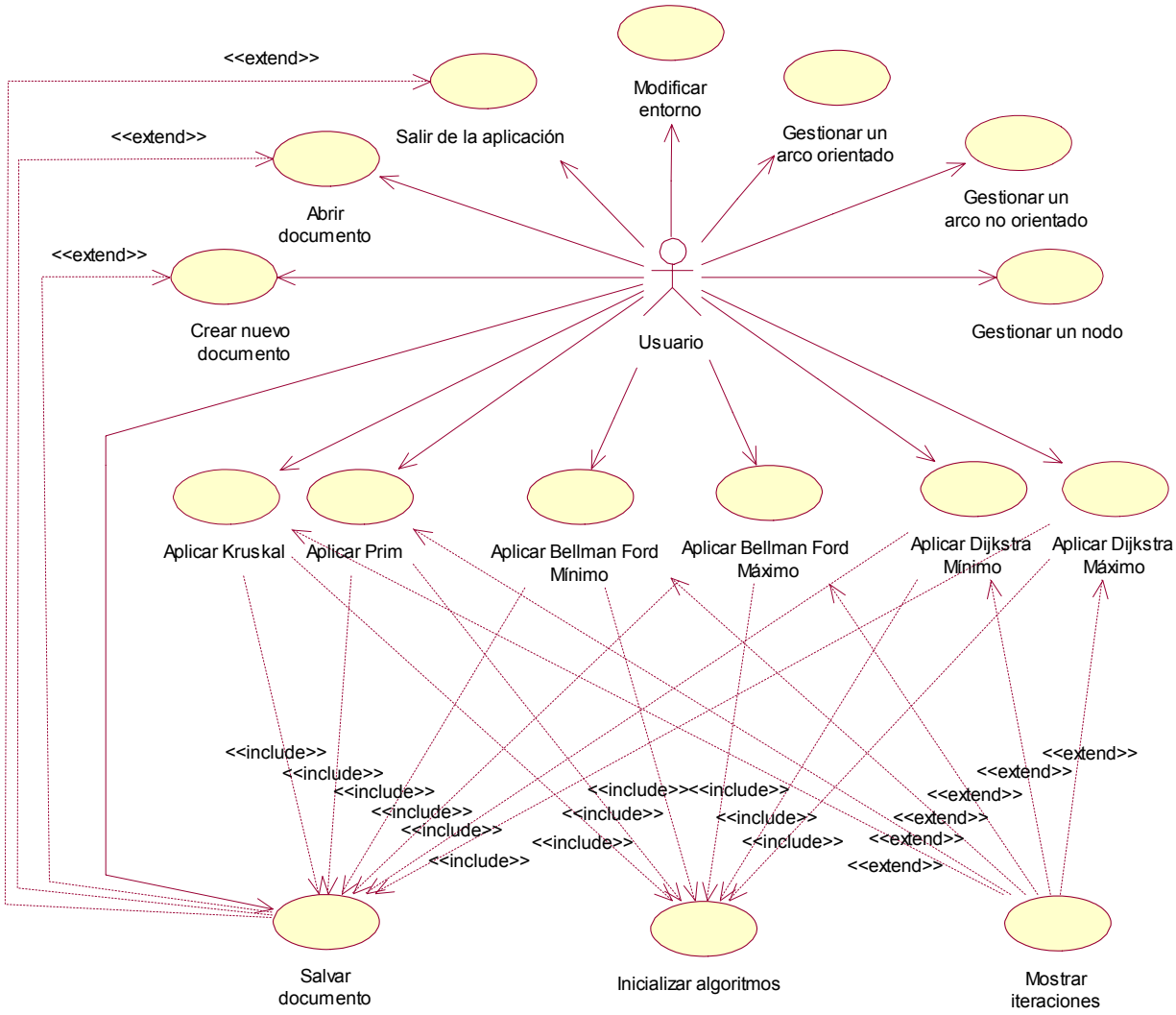


Figura VI. Diagrama de casos de uso del sistema.

2.5.4 – Descripción de los Casos de Uso

Caso de uso: Crear nuevo documento.
Actores: Usuario (inicia).
Propósito: Permitir la creación de un nuevo documento.
Resumen: El caso de uso se inicia cuando el usuario desea elaborar un nuevo documento y cuenta con toda la información necesaria basada en nodos

y arcos que conectan los mismos. El sistema verifica si hay algún documento creado sin guardar, dando la posibilidad al usuario de realizar el caso de uso **Salvar un documento** para mantener la información o desechar la misma. Se concluye este caso de uso con la creación de un nuevo documento.
Precondiciones:
Referencias: R1 Salvar documento (Extend).
Poscondiciones: ■ Se creó un nuevo documento.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 1

Caso de uso: Abrir documento.
Actores: Usuario (inicia).
Propósito: Cargar de un fichero la información almacenada de un grafo y representarla mediante un documento.
Resumen: El caso de uso se inicia cuando el usuario desea cargar un grafo almacenado en un fichero creado con antelación. El sistema verifica si hay algún documento creado sin guardar, dando la posibilidad al usuario de realizar el caso de uso **Salvar documento** para mantener la información o desechar la misma. Se concluye este caso de uso con la representación en un documento de la información del grafo almacenada en el fichero.
Precondiciones: ■ El fichero debe ser del tipo GRDM.
Referencias: R2

Salvar documento (Extend).
Poscondiciones: ■ Se culmina con la representación en un documento de la información del grafo almacenada en el fichero.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 2

Caso de uso: Salvar documento.
Actores: Usuario (inicia).
Propósito: Permitir el almacenamiento de un grafo en un fichero con extensión propia (GRDM).
Resumen: El caso de uso se inicia cuando el usuario desea salvar en un fichero con extensión propia la información de un grafo que está elaborando. El usuario entra los datos del fichero como nombre y ubicación del mismo. Se concluye este caso de uso con la salva de un grafo en un fichero con extensión propia.
Precondiciones: ■ Debe existir un grafo con al menos un nodo.
Referencias: R3, R4
Poscondiciones: ■ Se culmina con la salva de un grafo en un fichero con extensión propia (GRDM).
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 3

Caso de uso: Gestionar un nodo.
Actores: Usuario (inicia).

Propósito: Permitir gestionar un nodo.
Resumen: El caso de uso se inicia cuando el usuario desea gestionar un nodo, el sistema le brinda la posibilidad de **Insertar un nodo**, **Modificar un nodo y Eliminar un nodo** del grafo. El usuario puede especificar las características (color y posición) con las que desea insertar un nodo, el sistema asigna un nombre verificando que el mismo no se repita y se establece una prioridad superior al nodo anterior, en caso de existir. Para modificar un nodo, inicialmente se debe seleccionar y aplicar la modificación deseada (nombre, color, posición y prioridad). Si se desea eliminar un nodo, se selecciona el mismo y antes de eliminar se da la posibilidad de reafirmar si se continúa o se cancela la operación. Se concluye este caso de uso con la actualización del grafo asociado al nodo.
Precondiciones: Para Modificar un nodo y Eliminar un nodo: ▌ Debe de existir al menos un nodo. ▌ Debe estar seleccionado algún nodo.
Referencias: R5, R6, R7
Poscondiciones: ▌ Se concluye con la actualización del grafo asociado al nodo.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 4

Caso de uso: Gestionar un arco orientado.
Actores: Usuario (inicia).
Propósito: Permitir gestionar un arco orientado entre dos nodos.

Resumen: El caso de uso se inicia cuando el usuario desea gestionar un arco orientado, el sistema le brinda la posibilidad de **Insertar un arco orientado**, **Modificar un arco orientado y Eliminar un arco orientado** del grafo. El usuario puede especificar las características (color, nodo inicial y nodo final) con las que desea insertar un arco orientado, el sistema asigna un peso inicial de valor 1, además de, invalidar la posibilidad de insertar arcos orientados. Para modificar un arco orientado, inicialmente se debe seleccionar y aplicar la modificación deseada (peso y color). Si se desea eliminar un arco orientado, se selecciona el mismo y antes de eliminar se da la posibilidad de reafirmar si se continúa o se cancela la operación. Se concluye este caso de uso con la actualización del grafo asociado al arco orientado.
Precondiciones: Para Insertar un arco orientado: ■ Debe de existir al menos dos nodos. Para Modificar un arco orientado y Eliminar un arco orientado: ■ Debe de existir al menos un arco orientado. ■ Debe estar seleccionado algún arco orientado.
Referencias: R8, R9, R10
Poscondiciones: ■ Se concluye con la actualización del grafo asociado al arco orientado.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 5

Caso de uso: Gestionar un arco no orientado.
Actores: Usuario (inicia).
Propósito: Permitir gestionar un arco no orientado entre dos nodos.

Resumen: El caso de uso se inicia cuando el usuario desea gestionar un arco no orientado, el sistema le brinda la posibilidad de **Insertar un arco no orientado**, **Modificar un arco no orientado y Eliminar un arco no orientado** del grafo. El usuario puede especificar las características (color, nodo inicial y nodo final) con las que desea insertar un arco no orientado, el sistema asigna un peso inicial de valor 1, además de, invalidar la posibilidad de insertar arcos orientados. Para modificar un arco no orientado, inicialmente se debe seleccionar y aplicar la modificación deseada (peso y color). Si se desea eliminar un arco no orientado, se selecciona el mismo y antes de eliminar se da la posibilidad de reafirmar si se continúa o se cancela la operación. Se concluye este caso de uso con la actualización del grafo asociado al arco no orientado.
Precondiciones: Para Insertar un arco no orientado: ■ Debe de existir al menos dos nodos. Para Modificar un arco no orientado y Eliminar un arco no orientado: ■ Debe de existir al menos un arco no orientado. Debe estar seleccionado algún arco no orientado.
Referencias: R11, R12, R13
Poscondiciones: ■ Se concluye con la actualización del grafo asociado al arco no orientado.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 6

Caso de uso: Inicializar algoritmos.
Actores: Usuario (inicia).
Propósito:

Permitir modificar algunos atributos del grafo para lograr un mejor entendimiento al desarrollar el algoritmo seleccionado.
Resumen: El caso de uso se inicia cuando el usuario desea aplicar un algoritmo al grafo. El sistema asigna valores predefinidos a los atributos color y valor para todos los nodos del grafo y color para todos los arcos, logrando así un mejor entendimiento por parte del usuario en el desarrollo del mismo. Se concluye este caso de uso con la modificación de los atributos necesarios.
Precondiciones: Debe existir un grafo con al menos un nodo. Debe seleccionarse un algoritmo a aplicar.
Referencias: R20
Poscondiciones: Se modificó el grafo con valores estándares.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 7

Caso de uso: Mostrar iteraciones.
Actores: Usuario (inicia).
Propósito: Permitir recorrer paso a paso la solución del algoritmo seleccionado.
Resumen: El caso de uso se inicia cuando el usuario desea recorrer paso a paso la solución del algoritmo seleccionado. El sistema brinda la posibilidad de ir al inicio, al final, además de moverse al paso siguiente o regresar al paso anterior. En cada una de las iteraciones se modifican los atributos de nodos y arcos implicados. Este caso de uso concluye cuando el usuario está conforme con la información que necesitaba del algoritmo

seleccionado.
Precondiciones: ▪ Debe existir un grafo con al menos un nodo. ▪ Debe seleccionarse un algoritmo a aplicar.
Referencias: R21
Poscondiciones: ▪ Se modifican los atributos de nodos y arcos implicados en la iteración.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 8

Caso de uso: Aplicar Dijkstra Mínimo.
Actores: Usuario (inicia).
Propósito: Permitir aplicar el Algoritmo Dijkstra, en su variante camino mínimo, al grafo construido por el usuario.
Resumen: El caso de uso se inicia cuando el usuario desea aplicar a un grafo el Algoritmo Dijkstra, en su variante camino mínimo; este método comienza luego de seleccionarse un nodo inicial, posteriormente se realiza el caso de uso **Salvar documento** para que el grafo se pueda modificar mediante el caso de uso **Inicializar algoritmos**. Seguidamente se ejecuta el algoritmo, donde se obtiene el menor camino del nodo inicial al resto de los nodos, así como el recorrido que se realiza para obtener este resultado. Luego si se desea, se puede recorrer paso a paso la solución del algoritmo, mediante el caso de uso **Mostrar iteraciones**. Este caso de uso concluye con el cierre de la ventana de algoritmos y el retorno a la ventana principal.
Precondiciones:

Debe existir un grafo con al menos un nodo. Debe seleccionarse un nodo inicial. El peso de los arcos no puede ser negativo.
Referencias: R14 Salvar documento (Include). Inicializar algoritmos (Include). Mostrar iteraciones (Extend).
Poscondiciones: Se retornó a la ventana principal.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 9

Caso de uso: Aplicar Dijkstra Máximo.
Actores: Usuario (inicia).
Propósito: Permitir aplicar el Algoritmo Dijkstra, en su variante camino máximo, al grafo construido por el usuario.
Resumen: El caso de uso se inicia cuando el usuario desea aplicar a un grafo el Algoritmo Dijkstra, en su variante camino máximo; este método comienza luego de seleccionarse un nodo inicial, posteriormente se realiza el caso de uso **Salvar documento** para que el grafo se pueda modificar mediante el caso de uso **Inicializar algoritmos**. Seguidamente se ejecuta el algoritmo, donde se obtiene el mayor camino del nodo inicial al resto de los nodos, así como el recorrido que se realiza para obtener este resultado. Luego si se desea, se puede recorrer paso a paso la solución del algoritmo, mediante el caso de uso **Mostrar iteraciones**. Este caso de uso concluye con el cierre de la ventana de algoritmos y el retorno a la ventana principal.

Precondiciones: Debe existir un grafo con al menos un nodo. Debe seleccionarse un nodo inicial. El peso de los arcos no puede ser negativo.
Referencias: R15 Salvar documento (Include). Inicializar algoritmos (Include). Mostrar iteraciones (Extend).
Poscondiciones: Se retornó a la ventana principal.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 9

Caso de uso: Aplicar Bellman Ford Mínimo.
Actores: Usuario (inicia).
Propósito: Permitir aplicar el Algoritmo Bellman Ford, en su variante camino mínimo, al grafo construido por el usuario.
Resumen: El caso de uso se inicia cuando el usuario desea aplicar a un grafo el Algoritmo Bellman Ford, en su variante camino mínimo; este método comienza luego de seleccionarse un nodo inicial, posteriormente se realiza el caso de uso **Salvar documento** para que el grafo se pueda modificar mediante el caso de uso **Inicializar algoritmos**. Seguidamente se ejecuta el algoritmo, donde se obtiene el menor camino del nodo inicial al resto de los nodos, así como el recorrido que se realiza para obtener este resultado. Luego si se desea, se puede recorrer paso a paso la solución del algoritmo, mediante el caso de uso **Mostrar iteraciones**. Este caso de uso concluye con el cierre de la ventana de algoritmos y el retorno a la ventana principal.

Precondiciones: Debe existir un grafo con al menos un nodo. Debe seleccionarse un nodo inicial.
Referencias: R16 Salvar documento (Include). Inicializar algoritmos (Include). Mostrar iteraciones (Extend).
Poscondiciones: Se retornó a la ventana principal.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 9

Caso de uso: Aplicar Bellman Ford Máximo.
Actores: Usuario (inicia).
Propósito: Permitir aplicar el Algoritmo Bellman Ford, en su variante camino máximo, al grafo construido por el usuario.
Resumen: El caso de uso se inicia cuando el usuario desea aplicar a un grafo el Algoritmo Bellman Ford, en su variante camino máximo; este método comienza luego de seleccionarse un nodo inicial, posteriormente se realiza el caso de uso **Salvar documento** para que el grafo se pueda modificar mediante el caso de uso **Inicializar algoritmos**. Seguidamente se ejecuta el algoritmo, donde se obtiene el mayor camino del nodo inicial al resto de los nodos, así como el recorrido que se realiza para obtener este resultado. Luego si se desea, se puede recorrer paso a paso la solución del algoritmo, mediante el caso de uso **Mostrar iteraciones**. Este caso de uso concluye con el cierre de la ventana de algoritmos y el retorno a la ventana principal.
Precondiciones:

■ Debe existir un grafo con al menos un nodo. ■ Debe seleccionarse un nodo inicial.
Referencias: R17 Salvar documento (Include). Inicializar algoritmos (Include). Mostrar iteraciones (Extend).
Poscondiciones: ■ Se retornó a la ventana principal.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 9

Caso de uso: Aplicar Prim.
Actores: Usuario (inicia).
Propósito: Permitir aplicar el Algoritmo de Prim al grafo construido por el usuario.
Resumen: El caso de uso se inicia cuando el usuario desea aplicar a un grafo el Algoritmo de Prim; este método comienza luego de seleccionarse un nodo inicial, posteriormente se realiza el caso de uso **Salvar documento** para que el grafo se pueda modificar mediante el caso de uso **Inicializar algoritmos**. Seguidamente se ejecuta el algoritmo, donde se obtiene el árbol de cubrimiento mínimo partiendo del nodo inicial, así como el recorrido que se realiza para obtener el árbol. Luego si se desea, se puede recorrer paso a paso la solución del algoritmo, mediante el caso de uso **Mostrar iteraciones**. Este caso de uso concluye con el cierre de la ventana de algoritmos y el retorno a la ventana principal.
Precondiciones: ■ Debe existir un grafo no dirigido con al menos un arco. ■ Debe seleccionarse un nodo inicial.

Referencias: R18 Salvar documento (Include). Inicializar algoritmos (Include). Mostrar iteraciones (Extend).
Poscondiciones: Se retornó a la ventana principal.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 9

Caso de uso: Aplicar Kruskal.
Actores: Usuario (inicia).
Propósito: Permitir aplicar el Algoritmo de Kruskal al grafo construido por el usuario.
Resumen: El caso de uso se inicia cuando el usuario desea aplicar a un grafo el Algoritmo de Kruskal; este método comienza luego de seleccionarse un nodo inicial, posteriormente se realiza el caso de uso **Salvar documento** para que el grafo se pueda inicializar mediante el caso de uso **Inicializar algoritmos**. Seguidamente se ejecuta el algoritmo donde se obtiene el árbol de coste total mínimo, así como el recorrido que se realiza para obtener el árbol. Luego si se desea, se puede recorrer paso a paso la solución del algoritmo, mediante el caso de uso **Mostrar Iteraciones**. Este caso de uso concluye con el cierre de la ventana de algoritmos y el retorno a la ventana principal.
Precondiciones: Debe existir un grafo no dirigido con al menos un arco. Debe seleccionarse un nodo inicial.
Referencias: R19 Salvar documento (Include). Inicializar algoritmos (Include).

Mostrar iteraciones (Extend).
Poscondiciones: ■ Se retornó a la ventana principal.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 9

Caso de uso: Modificar entorno.
Actores: Usuario (inicia).
Propósito: Permitir modificar el ambiente de trabajo.
Resumen: El caso de uso se inicia cuando el usuario desea aumentar o disminuir de tamaño el entorno de trabajo. Para esto se utiliza la herramienta Zoom. Este caso de uso concluye cuando el usuario está conforme con el entorno de trabajo que se visualiza.
Precondiciones: ■ Debe existir un documento activo.
Referencias: R22
Poscondiciones: ■ Se modificó el entorno de trabajo.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 9

Caso de uso: Salir de la aplicación.
Actores: Usuario (inicia).
Propósito: Permitir cerrar la aplicación.
Resumen: El caso de uso se inicia cuando el usuario desea cerrar la aplicación. El

sistema verifica si hay algún documento creado sin guardar, dando la posibilidad al usuario de realizar el caso de uso **Salvar documento** para mantener la información o desechar la misma. Se concluye el caso de uso con el cierre de la aplicación.
Precondiciones: ▌ Debe existir un grafo con al menos un nodo. ▌ Debe seleccionarse un algoritmo a aplicar.
Referencias: R23 Salvar documento (Extend).
Poscondiciones: ▌ Se retornó a la ventana principal.
Requerimientos especiales: -
Prototipo: Ver Anexo 1 - Prototipo # 10

2.6 – Construcción del sistema propuesto.

2.6.1 – Diagrama de clases del diseño.

Seguidamente aparecen los diagramas de clases del diseño propuestos para la construcción del sistema.

Figura VII. Diagrama de clases del diseño.

2.6.2 – Diagrama de clases persistentes.

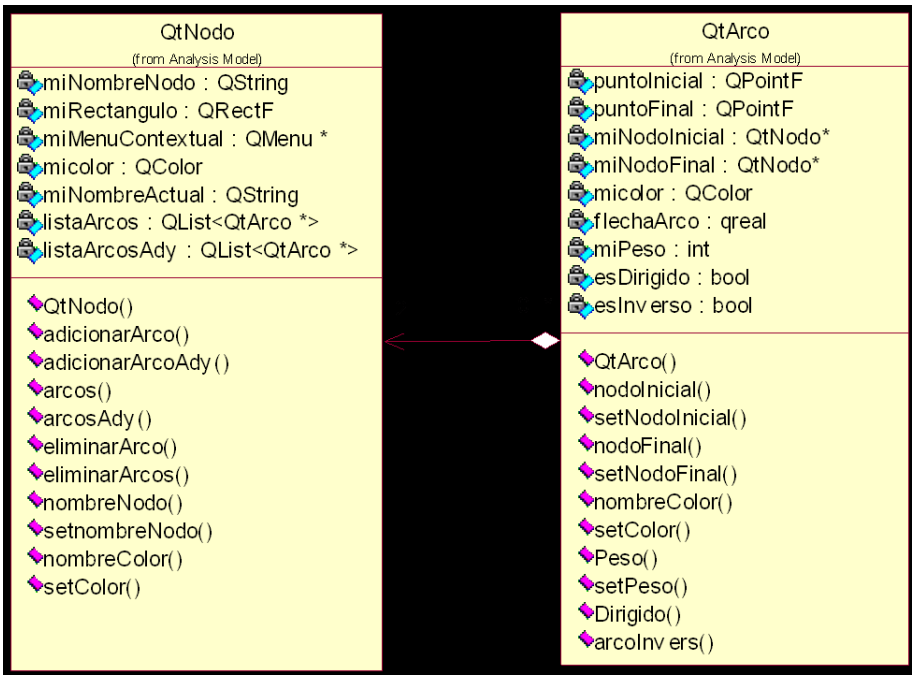


Figura VIII. Diagrama de clases persistentes.

Las clases representadas en el Diagrama de Clases Persistentes son guardadas en un fichero. El mismo está estructurado de la siguiente manera: contiene un encabezamiento formado por un identificador para verificar que fue generado por el sistema. A continuación un bloque con la cantidad de nodos que posee y la clasificación del grafo a construir. Luego un bloque dentro del anterior que contiene, para cada nodo, nombre, color, posición y la cantidad de arcos asociados al nodo. Seguidamente presenta un bloque que contendrá tantos sub-bloques como cantidad de arcos asociados tenga el nodo. Luego, en este sub-bloque se almacenará la información perteneciente a cada arco que será nombre del nodo inicial, nombre del nodo final, color y peso. En el **Anexo 2** se representa de forma gráfica toda la estructura del fichero que se genera y que ha sido descrito anteriormente.

2.6.3 – Principios de diseño.

2.6.3.1 – Estándares en la interfaz de la aplicación.

La interfaz diseñada para el sistema está basada en el estándar de ventanas de Windows. El tipo de letra a utilizar será MS Sans Serif de estilo regular y tamaño 9 y el diseño de la aplicación será conservador. La carga visual es adecuada y el lenguaje de las opciones que se ha utilizado es de fácil comprensión para el usuario. El sistema posee una barra de herramientas que brinda acceso rápido a la mayor parte de las opciones y en cada una de estas se han utilizado iconos para una mayor comprensión de la funcionalidad que realiza. El icono asociado a la aplicación será el logotipo que identifica al sistema. En cuanto a los mensajes de error e informativos que se muestran son breves, pero informando siempre en qué consiste el error.

Las figuras V y VI muestran ejemplos de lo expuesto anteriormente.

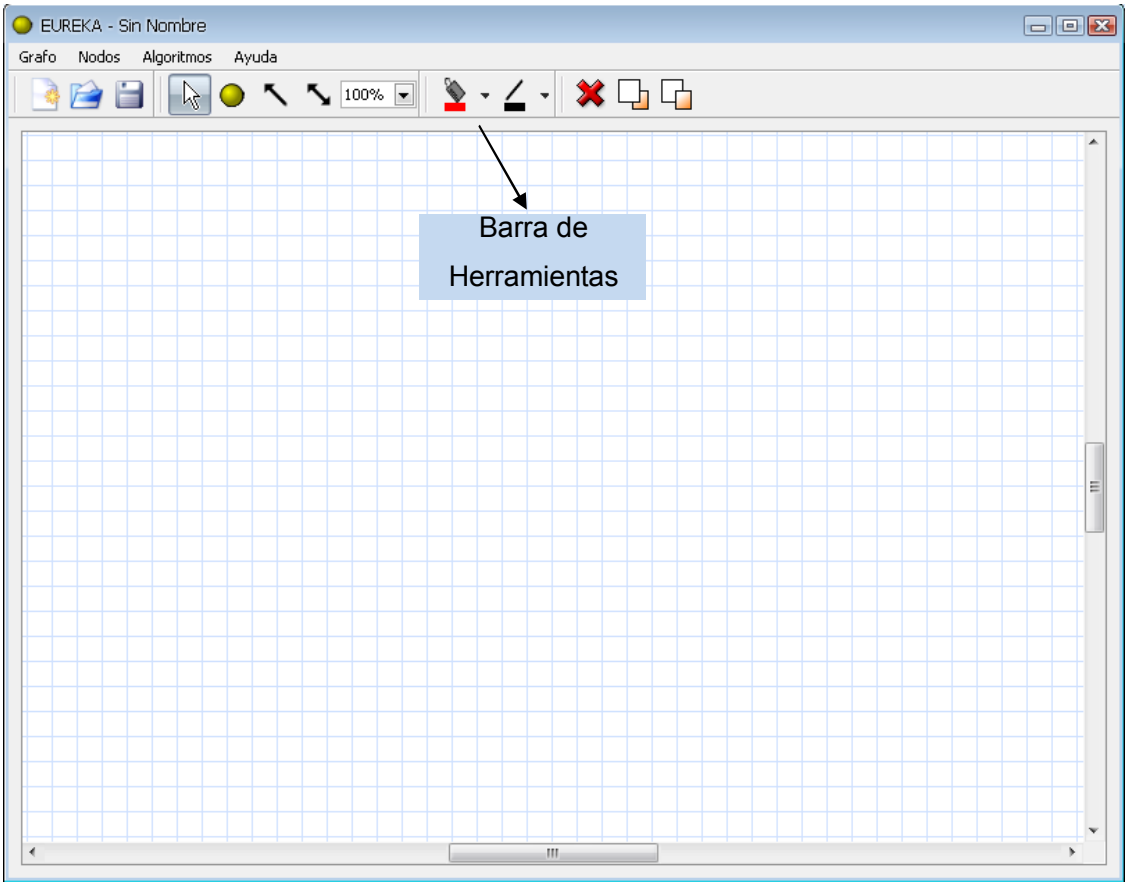


Figura IX. Vista del Diseño de la Interfaz.

Figura X. Tipos de mensajes del sistema.

2.6.3.2 Concepción general de la ayuda.

El sistema brinda una ayuda contextual que se muestra al pasar el mouse sobre cada botón de la barra de herramientas, de esta forma se le hace saber al usuario cómo operar con el sistema, según se muestra en la imagen a continuación.

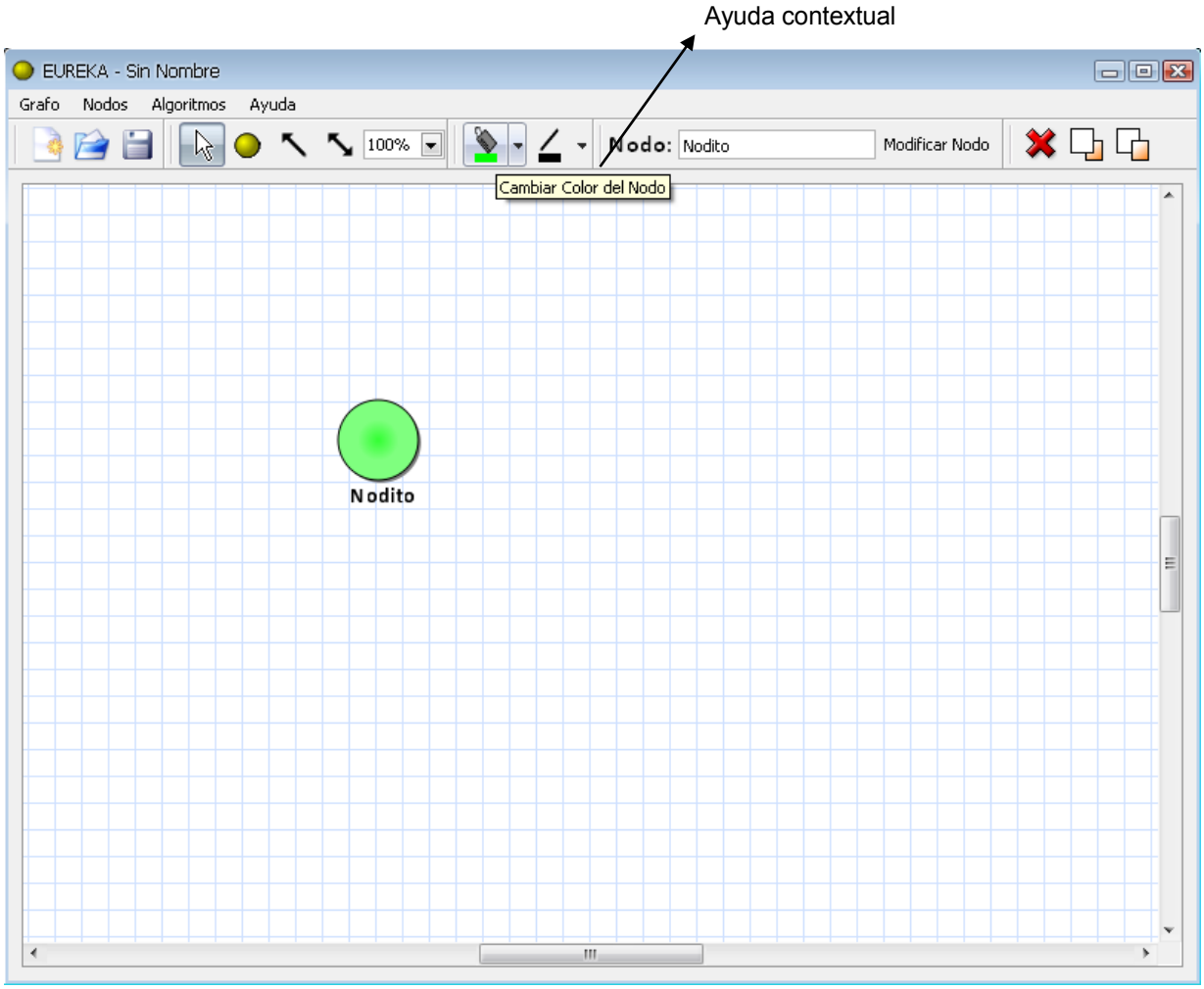


Figura XI. Ayuda del sistema.

2.6.3.3 Tratamiento de excepciones.

El diseño de la interfaz del sistema está dirigido a evitar errores, teniendo en cuenta siempre la creación de interfaces amigables. Los mensajes de error que emite el sistema se muestran en un lenguaje de fácil comprensión para los usuarios, además se le permite al usuario confirmar las operaciones que impliquen riesgos, tal es el caso de eliminar algún nodo o arco.

2.6.3.4 Estándares de codificación.

Para llevar a cabo el ejercicio de la buena ingeniería de software debe seguirse un buen estilo de código. Es necesario escribir código que sea fácil de entender y que reduzca el tiempo y esfuerzo a la hora de realizar alguna modificación al mismo.

Existen varios aspectos que pueden hacer un código más legible; algunos de estos son el empleo de nombres descriptivos, el uso de una indentación coherente y de comentarios informativos, entre otros. Se describen a continuación algunas convenciones tomadas con relación a estos aspectos.

Convención de Nombres

Los nombres de las variables, los controles, los procedimientos y funciones fueron adoptados lo más explicativos posibles, siempre respondiendo a su propósito.

Para cada tipo de control se tuvo en cuenta el uso de los prefijos más utilizados, una muestra de esto se muestra en la siguiente tabla:

Control	Prefijo	Ejemplo
QCheckBox	chbox	chbox_TipoEnlace
QButton	bt	bt_Aceptar
QForm	fr	fr_Principal
QLabel	lb	lb_Nombre
QMenu	mn	mn_Principal
QEdit	edt	edt_PosX
QToolButton	tb	tb_Imprimir
QListBox	lbx	Lbx_Categorias
QTimer	tm	tm_SalvaTemp
QAction	at	at_Accion

Tabla 1. Nombres de algunos controles utilizados

En el caso de las variables comenzarán todas con la letra "v" seguido de un guión bajo y luego el nombre deseado, por ejemplo: *v_Cantidad*.

Indentación

En el caso de la indentación, se establece que todas las líneas dentro de un procedimiento o función, estarán indentadas con respecto a la instrucción que encabeza a este, lo mismo ocurre con todas las líneas que conformen el cuerpo de un ciclo estructural condicional.

Comentarios

Un buen comentario añade información al código de una manera clara y ayuda a entender el objetivo del mismo. Se tomó como regla, por tanto, comentar todos los procedimientos y funciones al principio de los mismos para explicar cómo se deben usar sin necesidad de leer el código. Se comentaron además algunos algoritmos que pudieran resultar de difícil comprensión.

2.7 – Conclusiones.

A partir de lo analizado en este capítulo se concluye que:

Existen en la solución 23 requerimientos funcionales de los cuales se extrajeron 16 casos de uso, descritos posteriormente. Se identificó además un actor del sistema así como la labor que realiza en el mismo. Fueron expuestas algunas de las especificaciones sobre el tratamiento de errores, principios de codificación y de diseño que ayudan a una mejor implementación del producto informático.

Capítulo 3 – Estudio de factibilidad y validación del sistema

3.1 – Introducción

En el inicio de la elaboración de un software, resulta imprescindible determinar si el mismo resultará factible o no. La estimación del tiempo y los esfuerzos asociados a la realización del proyecto constituyen la base para el análisis de la factibilidad. Estas estimaciones serán realizadas a través del método planificación basada en caso de uso del modelo COCOMO II.

COCOMO es una herramienta utilizada para la estimación de algunos parámetros (costes en personas, tiempo, etc) en el diseño y construcción de programas y de la documentación asociada requerida para desarrollarlos, operarlos y mantenerlos, es decir, en la aplicación práctica de la Ingeniería del Software **[24]**.

En este capítulo se analizarán también los resultados obtenidos de las encuestas realizadas a los estudiantes que han cursado la asignatura de Investigación de Operaciones II, utilizando el programa estadístico SPSS v.15.0 para el procesamiento de estos datos. Además se utilizará la prueba de hipótesis para la comparación de los resultados, apreciando así si existe acuerdo o no entre los encuestados.

3.1 – Planificación basada en caso de uso

Método de estimación del esfuerzo de desarrollo de un producto de software a partir de los Casos de Uso y algunos factores de complejidad técnica y ambiente que influyen en el desarrollo **[25]**.

Pasos para la Estimación:

1. Calcular los Puntos de Casos de Uso **(PCU)**

- **1.1** Calcular el Factor de Peso de los Actores **(FPA)**
- **1.2** Calcular el Factor de Peso de los Casos de Uso **(FPCU)**

2. Calcular los Puntos de Casos de Usos Ajustados **(PCUA)**

- **2.1** Calcular el Factor de Complejidad Técnica **(FCT)**
- **2.2** Calcular el Factor de Ambiente **(FA)**

3. Calcular el Esfuerzo de desarrollo **(E)**

3.1.1 – Obtención de los Puntos de Casos de Uso sin ajustar

El primer paso para la estimación consiste en el cálculo de los Puntos de Casos de Uso sin ajustar. Este valor, se calcula a partir de la ecuación.

PCU = FPA + FPCU

Donde:

PCU: Puntos de Casos de Uso **(UUCP)**

FPA: Factor de Peso de los Actores **(UAW)**

FPCU: Factor de Peso de los Casos de Uso **(UUCW)**

3.1.1.1 – Obtención del Factor de Peso de los Actores sin ajustar (FPA)

Este valor se calcula mediante un análisis de la cantidad de actores presentes en el sistema y la complejidad de cada uno de ellos. La complejidad de los actores se establece teniendo en cuenta en primer lugar si se trata de una persona o de otro sistema, y en segundo lugar, la forma en la que el actor interactúa con el sistema. Los criterios son:

Tipo	Descripción	Peso	Cant*Peso
Simple	Otro sistema mediante una interfaz de programación.	1	0 * 1
Medio	Otro sistema mediante un protocolo o una interfaz basada en texto.	2	0 * 2
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica.	3	1 * 3
		Total:	3

Tabla 2. Factor de Peso de los Actores (FPA)

3.1.1.2 – Obtención del Factor de Peso de CU (FPCU)

Se calcula teniendo en cuenta la cantidad de casos de usos y su complejidad.

Complejidad: Se determina a partir de la cantidad de transacciones que se realizan.

Transacción: Es una secuencia atómica de actividades, las cuales se realizan completamente o no se realiza ninguna.

Tipo de CU	Transacciones	Peso
Simple	menos de 4	5
Medio	de 4 a 7	10
Complejo	más de 7	15

Tabla 3. Factor de Peso de CU (FPCU)

Para calcular UUCW:

No.	Caso de Uso	No. Transiciones	Tipo de CU	Peso
1	Crear nuevo documento.	2	Simple	5
2	Abrir documento.	2	Simple	5
3	Salvar documento.	2	Simple	5
4	Gestionar nodo.	5	Medio	10
5	Gestionar un arco orientado.	6	Medio	10
6	Gestionar un arco no orientado.	6	Medio	10
7	Aplicar Dijkstra Mínimo.	2	Simple	5
8	Aplicar Dijkstra Máximo.	2	Simple	5
9	Aplicar Bellman Ford Mínimo.	2	Simple	5
10	Aplicar Bellman Ford Máximo.	2	Simple	5
11	Aplicar Prim.	2	Simple	5
12	Aplicar Kruskal.	2	Simple	5
13	Inicializar algoritmos.	2	Simple	5
14	Mostrar iteraciones.	3	Simple	5
15	Modificar entorno.	2	Simple	5
16	Salir de la aplicación.	1	Simple	5

Tabla 4. Variables por casos de uso.

3.1.1.3 – Obtención de los Puntos de Casos de Uso sin ajustar (PCU)

Se tienen 13 casos de uso con la clasificación simple y 3 casos de uso con la clasificación media por lo que se le aplican como factor de peso 5 y 10 respectivamente.

PCU = FPA + FPCU

Luego: **PCU** = 3 + 95

PCU = 98

3.1.2 – Obtención de los Puntos de Casos de Uso ajustados

Después de calculados los PCU (sin ajustar) estos se deben ajustar teniendo en cuenta un grupo de factores técnicos y ambientales. Este valor, se calcula a partir de la ecuación.

PCUA = PCU X FCT X FA

Donde:

PCUA: Puntos de Casos de Usos Ajustados **(UCP)**

FCT: Factor de Complejidad Técnica **(TCF)**

FA: Factor de Ambiente **(EF)**

3.1.2.1 – Obtención del Factor de Complejidad Técnica (FCT)

Este coeficiente se calcula mediante la cuantificación de un conjunto de factores (13) que determinan la complejidad técnica del sistema. Cada uno de los factores se cuantifica con un valor de 0 a 5, donde 0 significa un aporte irrelevante y 5 un aporte muy importante. Se estima de la forma:

FCT = 0.6 + 0.01 x Σ ($Peso_i$ x $Valor_i$)

Factor	Descripción	Peso	Valor	Σ ($Peso_i$ * $Valor_i$)
T1	Sistema distribuido	2	4	8
T2	Objetivos de performance o tiempo de respuesta	1	4	4
T3	Eficiencia del usuario final	1	5	5
T4	Procesamiento interno complejo	1	5	5

T5	El código debe ser reutilizable	1	4	4
T6	Facilidad de instalación	0.5	4	2
T7	Facilidad de uso	0.5	4	2
T8	Portabilidad	2	4	8
T9	Facilidad de cambio	1	4	4
T10	Concurrencia	1	2	2
T11	Incluye objetivos especiales de seguridad	1	0	0
T12	Provee acceso directo a terceras partes	1	3	3
T13	Se requieren facilidades especiales de entrenamiento a usuarios	1	2	2
			Total:	49

Tabla 5. Factores Técnicos

FCT = 0.6 + 0.01 * 49

FCT = 1.09

3.1.2.2 – Obtención del Factor de Ambiente (FA)

Después se multiplican los pesos asociados a cada factor por el valor puesto y se aplica la fórmula para obtener el factor de Ambiente. Luego se pueden calcular los puntos de función ajustados multiplicando los desajustados por los factores de complejidad técnica y de ambiente. Se estima de forma similar al FCT:

FA = 1.4 - 0.03 x Σ ($Peso_i$ x $Valor_i$)

Factor	Descripción	Peso	Valor	Σ ($Peso_i$ * $Valor_i$)
E1	Familiaridad con el modelo de proyecto utilizado	1.5	3	4.5

E2	Experiencia en la aplicación	0.5	3	1.5
E3	Experiencia en orientación a objetos	1	5	5
E4	Capacidad del analista líder	0.5	4	2
E5	Motivación	1	5	5
E6	Estabilidad de los requerimientos	2	4	8
E7	Personal a tiempo compartido	-1	0	0
E8	Dificultad del lenguaje de programación	-1	4	-4
			Total:	22

Tabla 6. Factor Ambiente.

FA = 1.4 - 0.03 * 22

FA = 0.74

3.1.2.3 – Obtención de los Puntos de Casos de Uso ajustados (PCUA)

Se tienen 13 casos de uso con la clasificación simple y 3 casos de uso con la clasificación media por lo que se le aplican como factor de peso 5 y 10 respectivamente.

PCUA = PCU X FCT X FA

Luego: **PCUA** = 98 **X** 1.09 **X** 0.74

PCUA = 79.05

3.1.3 – Calcular el Esfuerzo de desarrollo

Convertir los Puntos de Casos de Uso Ajustados a Esfuerzo de desarrollo.

E = PCUA x FC

Donde:

FC: Factor de Conversión

El valor de FC según Karner es de 20 H/H

Puede ser calibrado entre 15 y 30 H/H en dependencia de los FA.

Karner originalmente sugirió que cada Punto de Casos de Uso requiere 20 horas-hombre. Posteriormente, surgieron otros refinamientos que proponen mayor detalle, según el siguiente criterio:

Sumar el número de factores de ambiente que están por debajo de la media de E1 a E6 y los que están por encima de E7 y E8

- Si la suma da menor o igual a 2 se usa como factor de conversión 20 horas-hombre.
- Si la suma da 3 o 4 se usa como factor de conversión 28 horas-hombre.
  Mayor o igual que 5 se debe ajustar pues se considera de alto riesgo con posibilidad de fracaso.

Media = 3

Total EF = Cant EF < 3 (entre E1 – E6) + Cant EF > 3 (entre E7 – E8)

Como Total EF = 2 + 0

Total EF = 2

CF = 20 horas-hombre (porque Total EF = 2)

Luego **E = PCUA x FC**

E = 79.05* 20 horas-hombre

E = 1581 horas-hombre

La Programación ocupa el 40 % del proyecto. Existe una aproximación estimada de la distribución del esfuerzo en función de las etapas del desarrollo de software.

Tipo de Actividad	Porciento	Esfuerzo (H/H)
Análisis	10	395.25
Diseño	20	790.5
Implementación	40	1581
Pruebas	15	592.875
Sobrecarga (Otras actividades)	15	592.875
Total	100	3952.5

Tabla 7. Distribución del esfuerzo estimado entre los flujos de trabajo de RUP.

3.1.4 – Duración

Trabajando los 26 días al mes y 9 horas al día como promedio, se puede decir que:
Duración (días) = Total de Horas/Hombre entre 9 horas al día = 3952.5/9 = 439 días.
Duración (meses) = Total de días/26 días por mes = 439/26 = 16.88 ≈ 17 meses.

3.1.5 – Cálculo de costos

Tomando como salario promedio mensual 345.00 MN

Costo = 17 * 354 = **6018 MN**

3.1.6 – Beneficios tangibles e intangibles

El desarrollo de un producto informático tiene asociado un costo y el llevarlo a cabo está en dependencia de los beneficios que el mismo puede reportar. Los beneficios pueden ser económicos y de orden social, resaltando que los últimos mencionados tienen tanta connotación como los primeros.

Dentro de los beneficios tangibles que se desprenden del proyecto se puede mencionar el incremento de la productividad en la actividad que se aplique ya que esta herramienta podrá ser utilizada en cualquier entidad estatal del país para beneficio del mismo y de sus recursos.

Como beneficios intangibles se puede citar la obtención de una novedosa herramienta para la optimización de tareas, además del ahorro de tiempo que significa la realización de estos algoritmos de forma manual.

Análisis de costos y beneficios

Indudablemente la utilización de este producto informático traería grandes beneficios en cualquier sector que se desee, debido a la posibilidad que brinda para la optimización y descripción de las tareas a las que se aplique, lo que se traduce en un incremento productivo, además de los beneficios tanto tangibles como intangibles, descritos en el inicio del epígrafe.

Un aspecto importante para determinar la factibilidad de este producto, independientemente de los beneficios aparejados al mismo, es el costo, el cual fue estimado en 6018 MN, además supone un tiempo de desarrollo de 17 meses. Para la realización de la aplicación no se incurrió en gastos adicionales de equipamiento,

materiales de oficina, compra de otros sistemas necesarios, ni de herramientas de desarrollo, además no hubo necesidad de contratar personal calificado que realizara el trabajo imprescindible para obtener el producto final.

Analizando los costos se puede apreciar que los mismos son relativamente bajos, este aspecto, unido a los grandes beneficios que resultarían de la realización y posterior utilización del software propuesto, determina la factibilidad del desarrollo del producto.

3.2 –Validación del problema

El cálculo del número de elementos de la muestra según, el Muestreo Aleatorio Simple de Proporciones, utiliza la fórmula siguiente:

$$n = \frac{Npq}{(N-1)D + pq}$$

Donde:

p = proporción de elementos que cumplen la condición.

q = 1- p proporción de elementos que no cumplen la condición.

D = B2/4 donde B = Error dado por el investigador.

N = Tamaño de la población.

Ahora se demuestra que si p = q = 0.5, entonces, se encuentra el máximo número de elementos de la muestra.

Para validar el producto informático se realizó una encuesta a los estudiantes que han cursado la asignatura de Investigación de Operaciones II en la Universidad de Cienfuegos (Cuarto año de II), se tomaron los siguientes datos:

p = q = 0.5

N = 36 Estudiantes (Total de la Población)

B = Error de muestreo = 0.13

Aplicando la fórmula anteriormente estudiada se calcula n = 16 estudiantes (muestra escogida, en forma aleatoria, para hacer el análisis estadístico del Producto informático).

3.2.2 – Resultados del Procesamiento Estadístico

Los resultados de la encuesta se analizan mediante el programa estadístico SPSS v.15.0. Además se aplica la prueba de hipótesis para apreciar la existencia o no de acuerdo entre los encuestados (Prueba no paramétrica de Kendall). En el Anexo 2 se muestra la encuesta realizada.

El análisis Estadístico reportó los siguientes resultados:

	N	Minimum	Maximum	Mean	Std. Deviation
Utilidad del Software en forma general	16	1	2	1,19	,403
Utilidad como apoyo a la Enseñanza de la Teoría de Redes	16	1	2	1,13	,342
Relacionado con otras Aplicaciones sobre Redes en cuanto al contenido	16	1	3	1,44	,629
Relacionado con otras Aplicaciones sobre Redes en cuanto a la presentación	16	1	2	1,19	,403
En cuanto al USO	16	1	2	1,06	,250
Ventajas en la Entrada de Datos	16	1	2	1,25	,447
Ventajas en la facilidad de los gráficos	16	1	2	1,13	,342
Ventajas en los métodos	16	1	2	1,50	,516
Ventajas en la calidad de la visualización Gráfica	16	1	2	1,13	,342
Valor dado a la Aplicación en escala del 1 al 5	16	4	5	4,94	,250

Tabla 1. Estadística Descriptiva de todas las Variables

Se observa en la Tabla 8 que todas las variables, excepto la última, se encuentran en el rango de 1 a 2 (Muy Bueno a Bueno) con una valor medio muy cercano a 1 (Muy Bueno) y una desviación estándar pequeña, lo que indica la aceptación de los encuestados por el software.

La última variable Valor presenta un rango entre 4 y 5 con una media muy cercana a 5 (máxima puntuación posible) y una desviación estándar muy pequeña.

3.2.2 – Comprobación de la existencia de acuerdos entre los encuestados

Prueba de Hipótesis:

H_0 : Los encuestados no están de acuerdo con las respuestas a las preguntas sobre el Software.

H_1: Los encuestados están de acuerdo con las respuestas a las preguntas sobre el software.

Para decidir cuál hipótesis aceptar se debe comparar el nivel de significación con la significación asintótica del estadígrafo, si esta última es menor que la significación entonces se acepta H_1.

N	16
Kendall's W(a)	,504
Chi-Square	72,646
df	9
Asymp. Sig.	,000

Tabla 2. Prueba Kendall

En la Tabla 9 podemos comprobar que se cumple la Hipótesis Alternativa H_1 con una aprobación moderadamente fuerte. De acuerdo a la prueba de concordancia W de Kendall que se realizó tomando en cuenta la hipótesis H_0 (Los encuestados no están de acuerdo con las respuestas a las preguntas sobre el Software) y H_1 (Los encuestados están de acuerdo con las respuestas a las preguntas sobre el Software) para así decidir cuál de las hipótesis aceptar, se adoptó un nivel de significación de 0.05 mayor que la significación asintótica calculada en el SPSS de 0.00, por lo que se puede concluir que la hipótesis a aceptar es H_1 y sí existe un acuerdo entre las opiniones de los encuestados definiéndose como moderadamente fuerte, véase en la Tabla 9.

3.3 –Conclusiones

Luego del análisis de este capítulo se concluye que:

- Es factible realizar el sistema propuesto, teniendo en cuenta el costo y los beneficios que aportará con su implementación; resultando así un costo de 6018 MN y desarrollándose por una persona en un tiempo de 17 meses.
- En la validación del sistema la hipótesis a aceptar es H_0 y sí existe un acuerdo entre las opiniones de los encuestados definiéndose como moderadamente fuerte luego de ser analizadas en el programa estadístico SPSS.

Conclusiones

Después de realizado todo el estudio y elaborado el producto informático se arribaron a las siguientes conclusiones:

- Del análisis del Estado del Arte se constató que las aplicaciones Informáticas: WinQSB, QSB, QmWin2 y POM no resuelven parcialmente las necesidades de los usuarios que interactúan con ellos.
- Entre los métodos importantes de la Teoría de Redes estudiados se encuentran algunos como el Cartero Chino, Dijkstra y Bellman Ford que no se implementan en las aplicaciones anteriormente mencionadas y que son ampliamente tratados en la Aplicación Informática Eureka.
- En cuanto a la complejidad (O) de los métodos estudiados, los paquetes de programas existentes, poseen O (N^2) mientras que el propuesto tiene O (N Log N), en cuanto a los Algoritmos de Caminos Extremales.
- Se logró una eficiente estructuración y concepción del sistema, gracias al empleo de los diferentes flujos de trabajos ofrecidos por la metodología RUP.
- La herramienta implementada sobre cumple los objetivos inicialmente planteados en cuanto a: facilidad de la introducción de los datos, trabajo con los gráficos, aplicación de nuevos algoritmos de Redes, interpretación de los resultados.
- El Estudio de Factibilidad realizado en el Capitulo 3, constató la factibilidad del sistema propuesto.
- El Muestreo Aleatorio aplicado validó la encuesta de satisfacción del cliente y la Prueba de Hipótesis conjuntamente con la W de Kendall proporcionó el acuerdo significativo de los resultados de las encuestas.

Recomendaciones

Con el fin de brindarle al sistema desarrollado una mayor funcionalidad, acorde con las necesidades reales de estudiantes y profesores que utilizan la herramienta, se recomienda:

- ✓ Realizar un nuevo análisis de necesidades para ampliar el producto propuesto con otros módulos que lo enriquezcan.
- ✓ Implementar más algoritmos enmarcados en la Teoría de Redes con el objetivo de fortalecerlo y alcanzar una mejora continua ampliando sus funcionalidades.
- ✓ Dotar al sistema con una entrada de datos tabular para matrices de incidencia y de adyacencia.
- ✓ Implantar el sistema en la Universidad de Cienfuegos, como herramienta de apoyo a los estudiantes y profesores que cursen o impartan la asignatura Investigación de Operaciones II.
- ✓ Con las recomendaciones anteriores el sistema propuesto podría formar parte de un estudio de post grado.

Referencias bibliográficas

[1] Computing-dictionary. [En línea] [Citado el: 13 de febrero de 2010.] computing-dictionary.thefreedictionary.com/computer+system.

[2] Ginés Garcia Mateos, Apuntes Algoritmos y Estructura de Datos, Madrid: Editorial Barcelo, 2004.

[3] Páginas blancas de la Teoría de grafos. [En línea] [Citado el: 13 de febrero de 2010.] www.math.gatech.edu/~sanders/graphtheory/.

[4] Gupta, Karypis, "Introduction to Parallel Computing. Design and Analysis of Algorithms." The Benjamin Cumming Publishing Company. 1994.

[5] Explicación Algoritmo de Dijkstra. [En línea] [Citado el: 17 de febrero de 2010.] www.ece.northwestern.edu/~guanghui/Transportation/spt/section3_1.html.

[6] Dijkstra, E., A note on two problems in connexion with graphs, Numerische Mathematik 1, 1959.

[7] Bellman, R. E., On a routing problem, quarterly of Applied Mathematics, 1958.

[8] Kruskal (Bell Labs). [En línea] [Citado el: 17 de febrero de 2010.] cm.bell-labs.com/cm/ms/departments/sia/kruskal/index.html.

[9] Kruskal, J., "On the shortest spanning subtree of a graph and the traveling salesman problem", Proc. of the American Mathematical Society, 1956.

[10] Minimum Spanning Tree. [En línea] [Citado el: 17 de febrero de 2010.] www2.toki.or.id/book/AlgDesignManual/BOOK/BOOK4/NODE161.HTM.

[11] Priml, R., "Shortest connection networks and some generalizations", Bell System Technical Journal, 36, 1957.

[12] Introducción al QSB. [En línea] [Citado el: 24 de febrero de 2010.] network-modeling.qsb.com/.

[13] Introducción al WinQSB. [En línea] [Citado el: 24 de febrero de 2010.] network-modeling.winqsb.com/.

[14] Garzón, F., "Manual de Qm2 para windows", Guayaquil: Rosario S.A., 2000.

[15] Software for Production and Operations Management. [En línea] [Citado el: 24 de febrero de 2010.] www.prenhall.com/weiss_pomwin.

[16] Goyanes, L., "Programación orientada a objetos", Mc Graw-Hill Interamericana, 1996.

[17] Katrib, M., "Programación orientada a objetos con C++". Editorial Infosys, México, 1994.

[18] I. Jacobson, El Proceso Unificado de Desarrollo de Software, La Habana: Editorial Félix Varela, 2008.

[19] I. Jacobson, El Lenguaje Unificado de Modelado: Manual de referencia. Madrid: Pearson Educación S.A, 2000.

[20] *Applying UML in The Unified Process.* [En línea] [Citado el: 12 de marzo de 2010.] www.rational.com/uml.

[21] Stroustrup, B., The C++ Programming Language, Addison-Wesley, 3ª edición, 1997.

[22] *The Unified Process.* [En línea] [Citado el: 12 de marzo de 2010.] www.rational.com.

[23] Programación con Visual C++. [En línea] [Citado el: 27 de marzo de 2010.] http://www.monografias.com/trabajos5/visualcurso/visualcurso.shtml

[24] Troltech. [En línea] [Citado el: 28 de marzo de 2010.] www.troltech.com.

[25] Ruiz, F., Modelo de Estimación de Costes para proyectos software. Ciudad Real: Universidad de Castilla – La macha, 1999.

Bibliografía

Bellman, R. E., On a routing problem, quarterly of Applied Mathematics, 1958.

Dijkstra, E., A note on two problems in connexion with graphs, Numerische Mathematik 1, 1959.

Ginés Garcia Mateos, Apuntes Algoritmos y Estructura de Datos, Madrid: Editorial Barcelo, 2004.

Goyanes, L., "Programación orientada a objetos", Mc Graw-Hill Interamericana, 1996.
Gupta, Karypis, "Introduction to Parallel Computing. Design and Analysis of Algorithms." The Benjamin Cumming Publishing Company. 1994.

I. Jacobson, El Lenguaje Unificado de Modelado: Manual de referencia. Madrid: Pearson Educación S.A, 2000.

Jacobson, I.; Booch, G. y Rumbaugh, J.; "El Proceso Unificado de Desarrollo de software". 2000. Addison-Wesley.

Katrib, M., "Programación orientada a objetos con C++". Editorial Infosys, México, 1994.
Kruskal, J., "On the shortest spanning subtree of a graph and the traveling salesman problem", Proc. of the American Mathematical Society, 1956.

Priml, R., "Shortest connection networks and some generalizations", Bell System Technical Journal, 36, 1957.

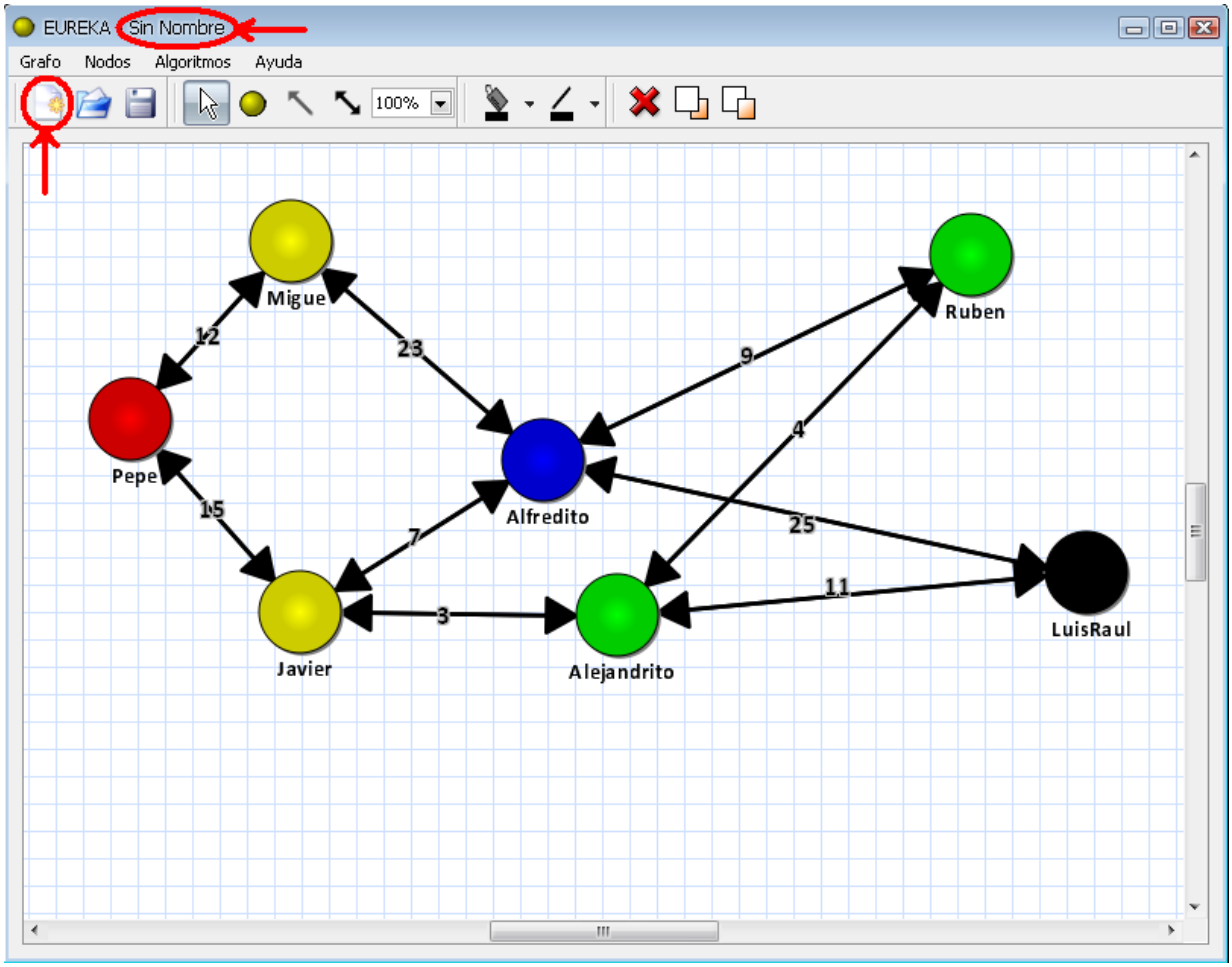
Ruiz, F., Modelo de Estimación de Costes para proyectos software. Ciudad Real: Universidad de Castilla – La macha, 1999.

Stroustrup, B., The C++ Programming Language, Addison-Wesley, 3ª edición, 1997.

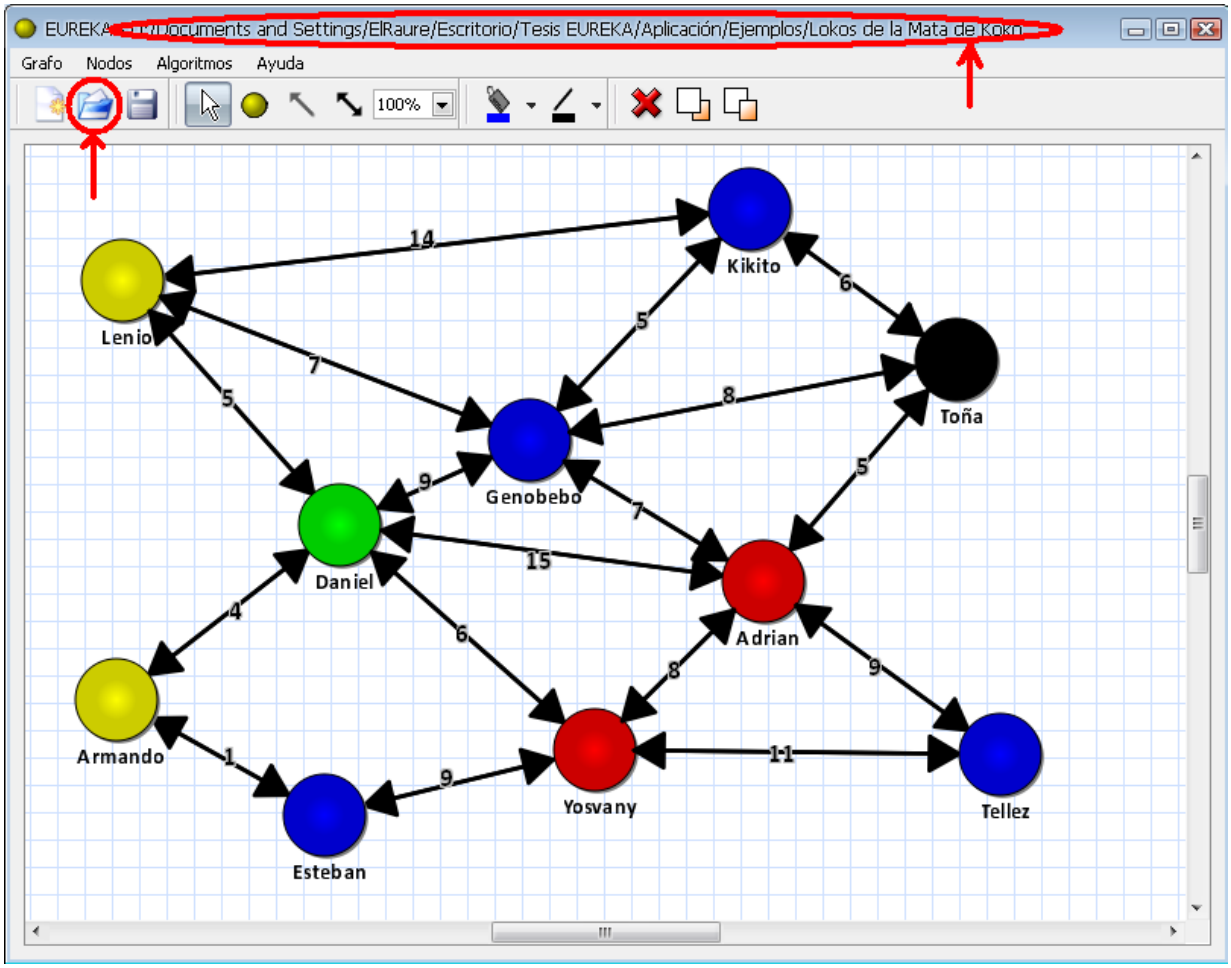
Anexos

Anexo 1 - Prototipos

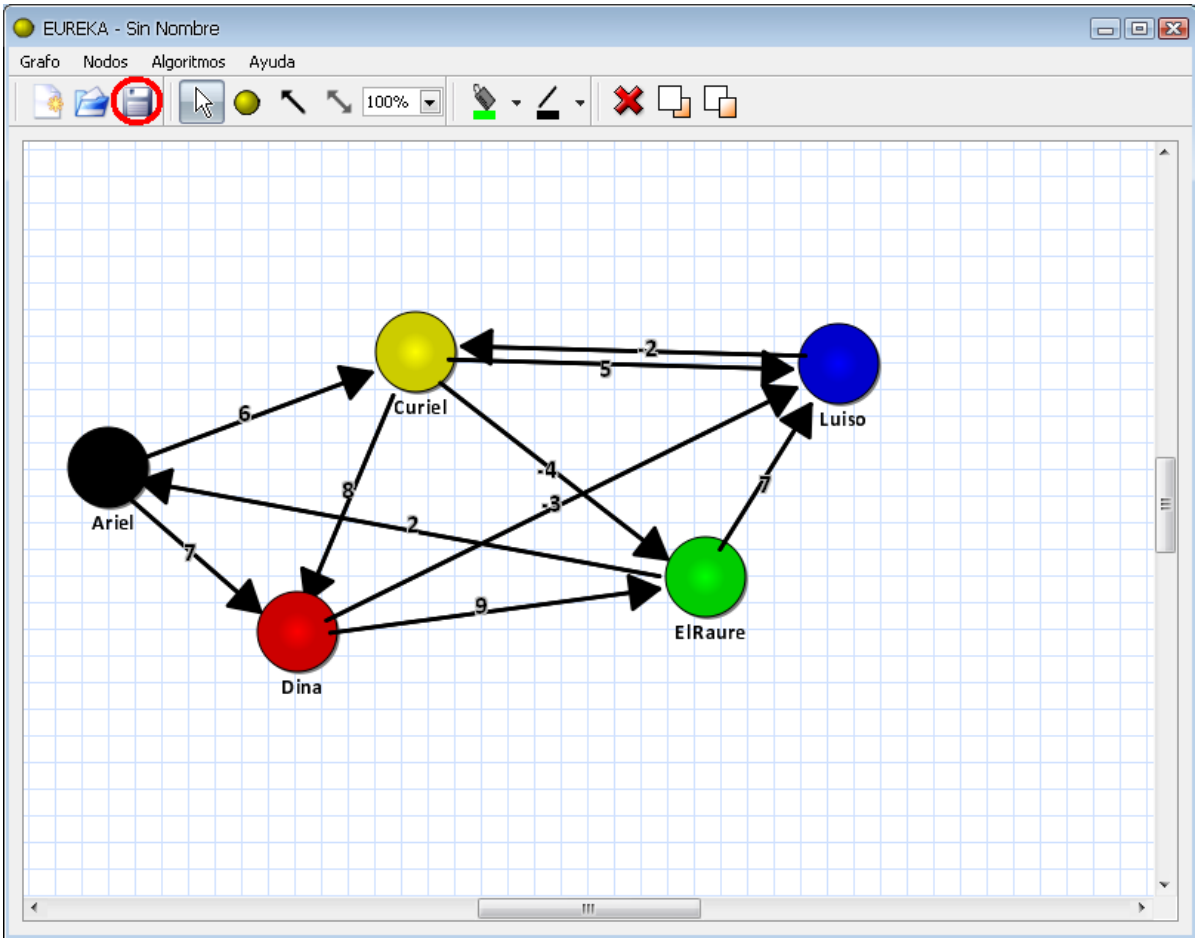
Prototipo #1 Crear nuevo documento.



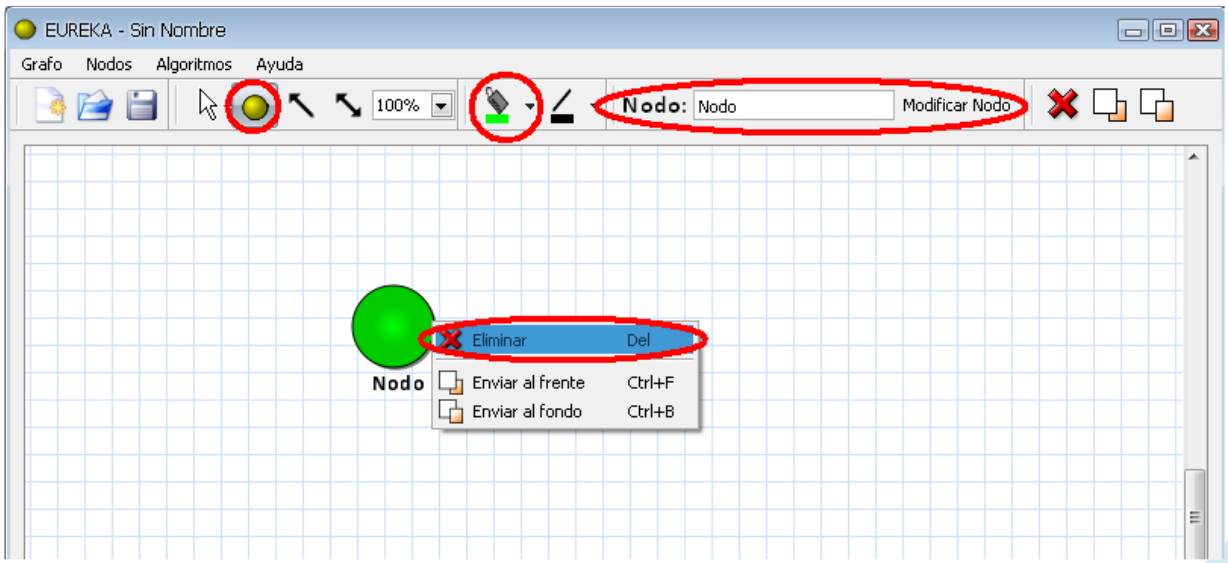
Prototipo #2 Abrir documento.



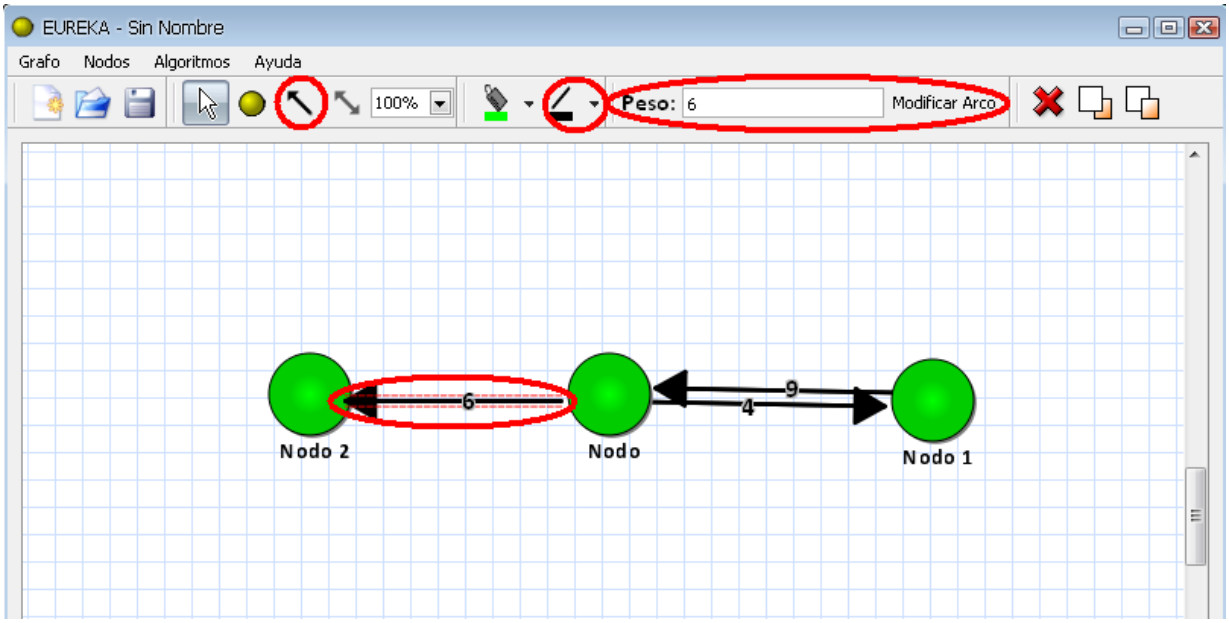
Prototipo #3 Salvar documento.



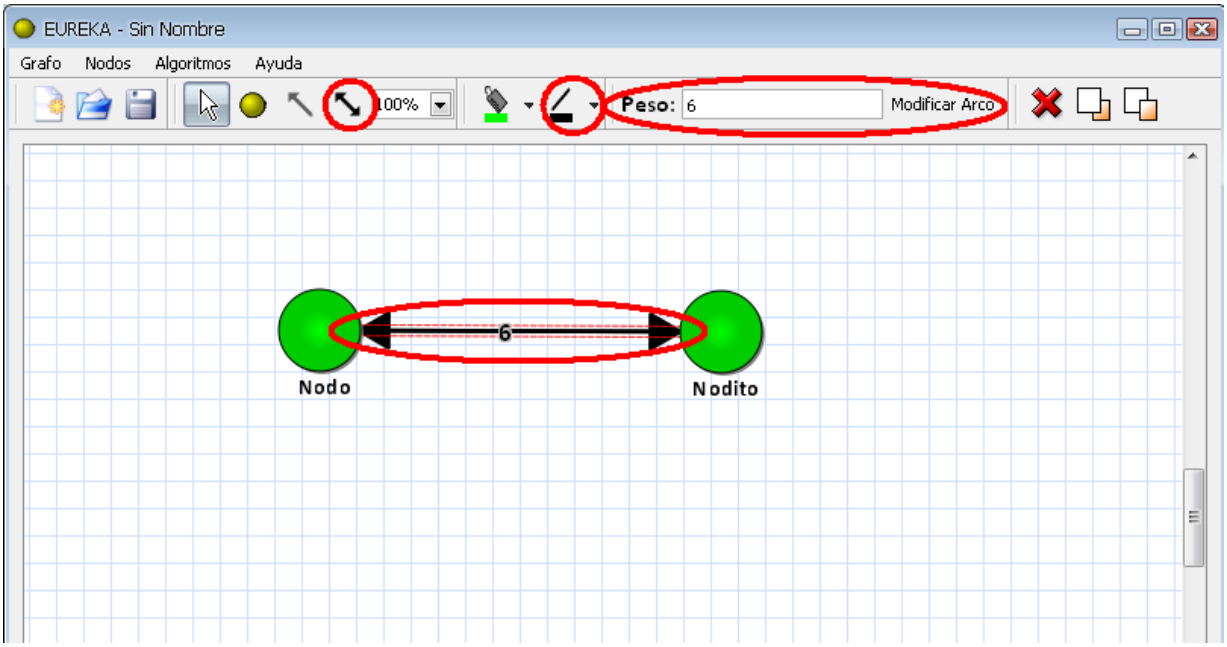
Prototipo #4 Gestionar un nodo.



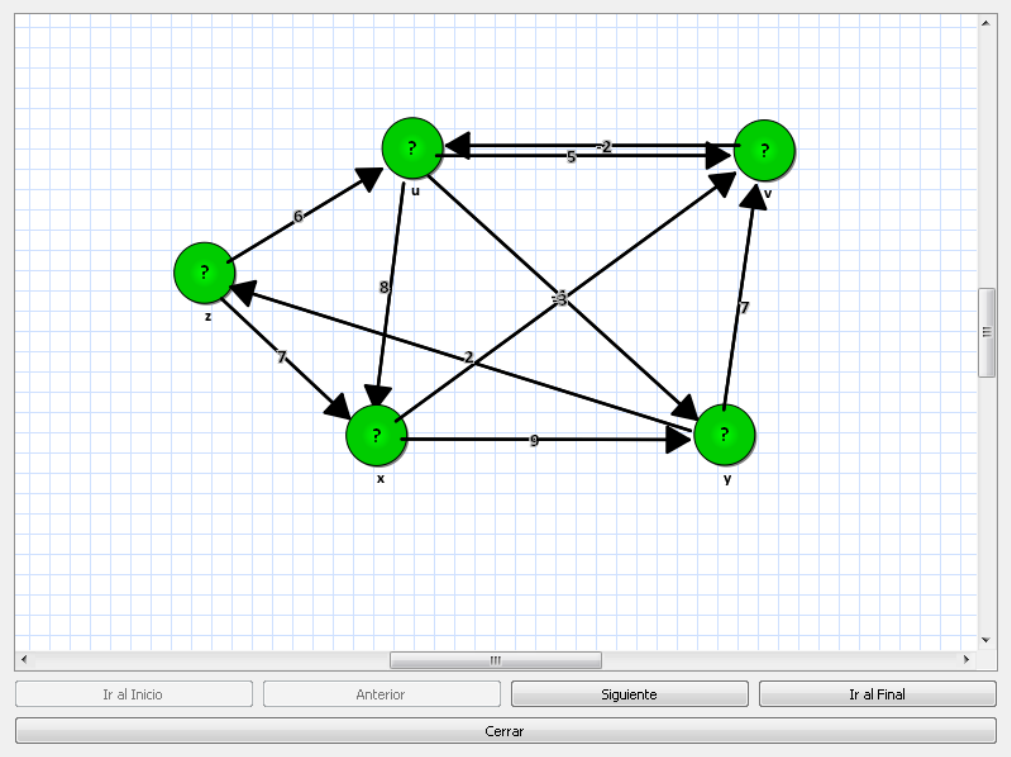
Prototipo #5 Gestionar un arco orientado.



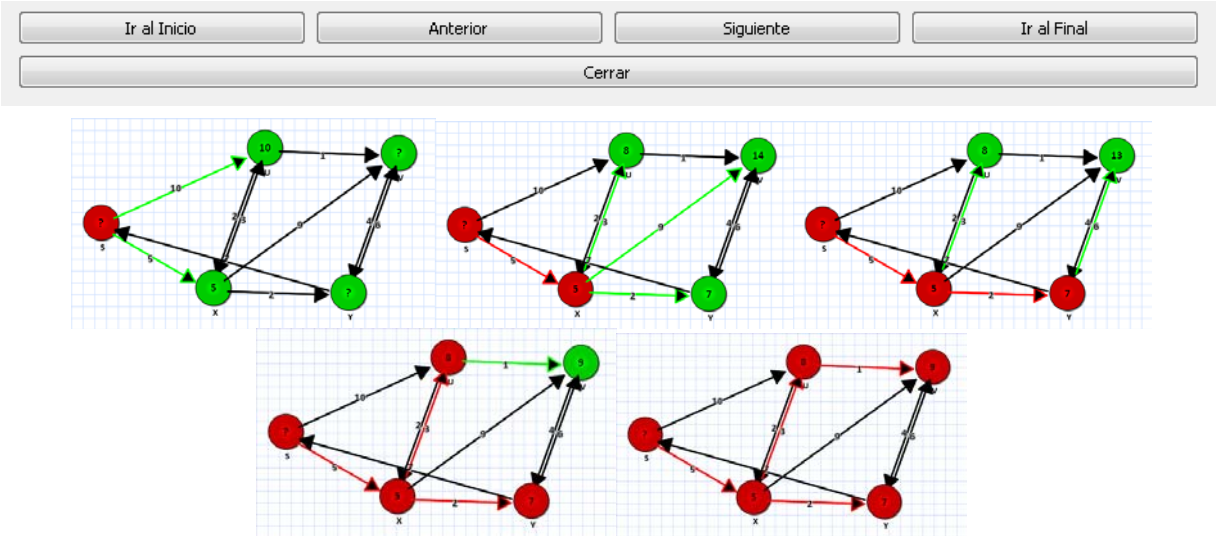
Prototipo #6 Gestionar un arco no orientado.



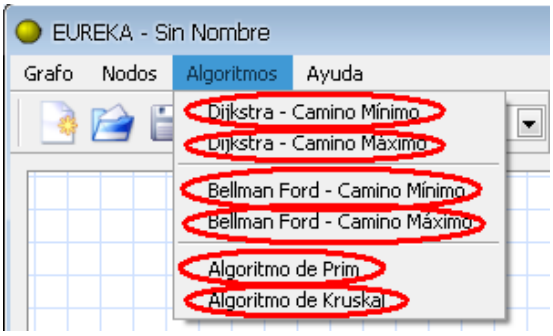
Prototipo #7 Inicializar algoritmos.

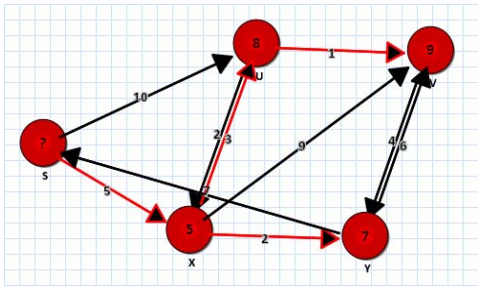


Prototipo #8 Mostrar iteraciones.

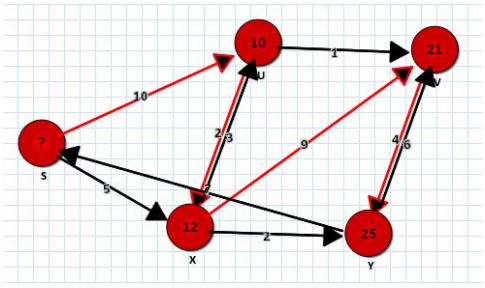


Prototipo #9 Algoritmos.

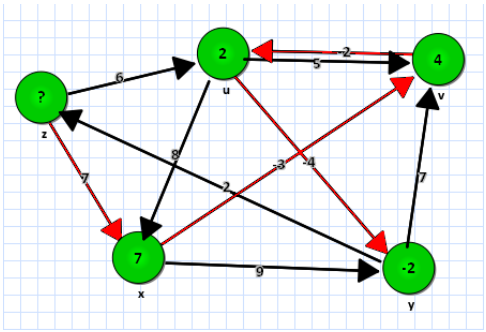




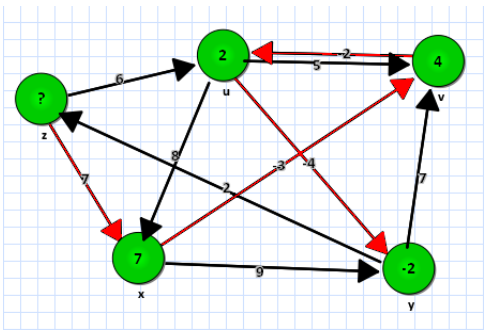
Dijkstra – Mínimo



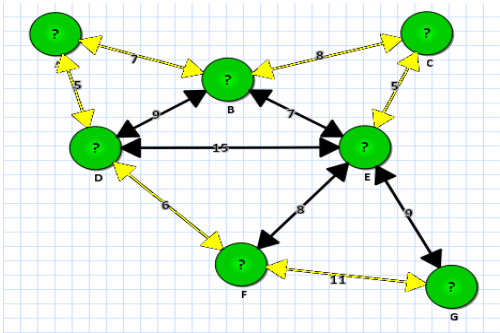
Dijkstra – Máximo



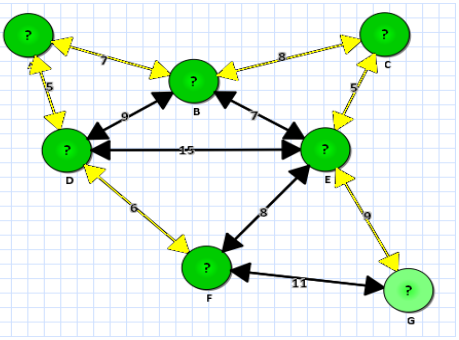
Bellman Ford – Mínimo



Bellman Ford – Máximo

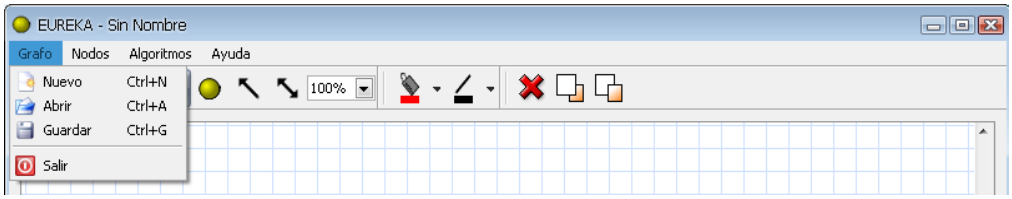


Kruskal



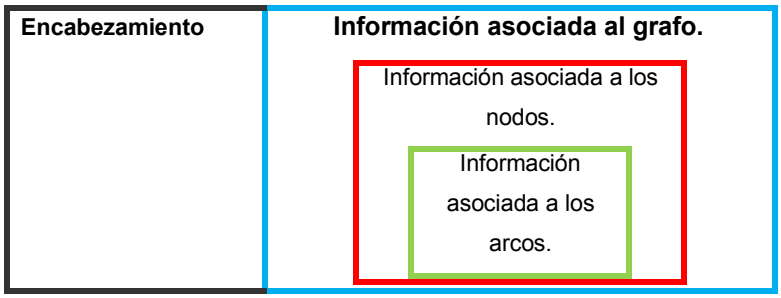
Prim

Prototipo #10 Salir de la aplicación.

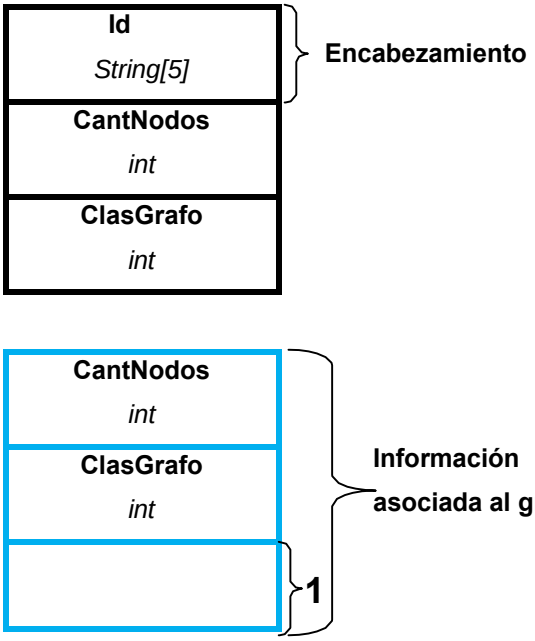


Anexo 2 – Estructura del fichero

La estructura general del fichero que crea el sistema propuesto para salvar un MC, es la siguiente:



A continuación se describe con bastante nivel de detalle cada parte del fichero.



Descripción de los datos del encabezamiento:

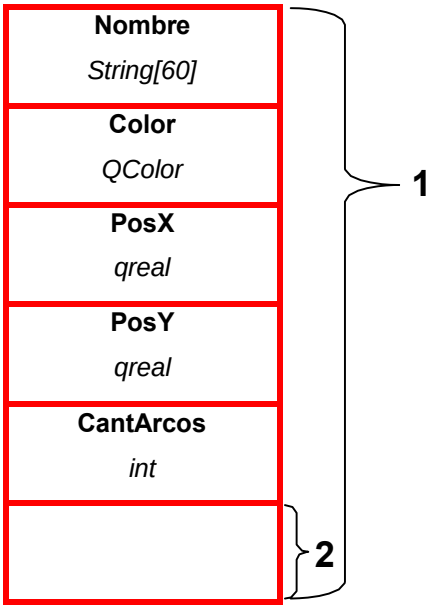
Id: Es un identificador para verificar que el fichero fue creado por el sistema.

Descripción de la información asociada al grafo:

CantNodos: Contiene el numero de nodos que posee el grafo.

ClasGrafo: Puede ser 0 ó 1 y nos informa si el grafo es orientado o no orientado, respectivamente.

Información asociada al los nodos.



Descripción de los nodos

Nombre: Almacena el nombre del nodo.

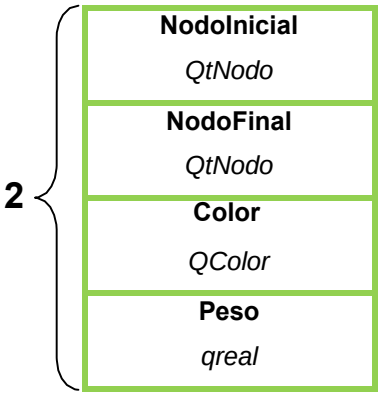
Color: Almacena el color del nodo.

PosX: Guarda la posición en el eje de la X.

PosY: Guarda la posición en el eje de la Y.

CantArcos: Guarda la cantidad de arcos adyacentes al nodo.

Información asociada a los arcos



Descripción de los nodos

NodoInicial: Almacena el nombre del nodo inicial.

NodoFinal: Almacena el nombre del nodo final.

Color: Almacena el color del arco.

Peso: Guarda el peso del arco.

Anexo 3 – Encuesta

Encuesta sobre Producto Informático de Teoría de Redes.

Estimado Usuario la presente encuesta forma parte de la Validación de un Producto Informático para un Trabajo de Diploma en la carrera de Ingeniería Informática.

Muchas Gracias por su participación.

1.- Utilidad del Producto Informático:

a.- Como Software de Teoría de Redes:

Muy Buena: ____ Buena: _____ Regular: _____ Mala: ____

b.- Como apoyo a la Enseñanza de la Teoría de Redes.

Muy Buena: ____ Buena: _____ Regular: _____ Mala: ____

2.- Relacionado con otras Aplicaciones Informáticas sobre Redes:

a.- En cuanto los contenidos:

Es Novedoso: ____ Tiene Mejoras: ____ Es Igual: _____ Es más malo: _____

b.- En cuanto a la presentación:

Muy Buena: ____ Buena:_____ Regular:____ Mala:____

3.- En cuanto al uso:

a.- Es más fácil de usar que otros: ____ b.- Es igual a los otros: ____ c.- Es más difícil: _____

4.- En qué radican las ventajas:

a.- En la Entrada de Datos: ___ **b.-** En la Facilidad de los Gráficos: ___ **c.-** En los Métodos: ____

d.- En la calidad de la Visualización Gráfica: ____ **e.-** No tiene ventajas: ____

5.- Si usted lo fuera a valorar en una escala de 5 cuántos puntos le daría: _____

6.- Algún comentario al respecto: