BENEMÉRITA

UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

SISTEMA DE PLANIFICACIÓN DE RUTAS DE LOGÍSTICA USANDO GRAFOS

TESIS

PARA OBTENER EL TITULO DE

INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA

MARCO POLO OLIVARES GONZALEZ

ASESOR

DR. PEDRO BELLO LÓPEZ

Septiembre 9, 2024

Agradecimientos

Quiero agradecer a mi familia por siempre apoyarme en cada unas de mis metas, en especial a mi madre Yolanda que sin ella no seria el hombre que soy ahora, con su amor, apoyo y cuidado he logrado cada una de mis metas y sueños a lo largo de mi vida.

A mis hermanas Brenda Ivonne y María Fernanda que han sido mi motivación para ser un mejor hombre en todos los aspectos, por apoyarme, aconsejarme y caminar conmigo en este largo camino tanto personal como profesional.

Agradezco a mis hermanos de otra madre Adrian, Daniel y Jonathan que durante tantos años me han soportado, por sus consejos, su conocimiento y por demostrarme que a pesar de todo siempre se puede lograr lo que uno se propone.

Agradezco a mis amigos en general que aún siguen conmigo por siempre estar en los momentos importantes que he vivido, en especial a Felipe por brindarme siempre su apoyo, conocimiento y por creer en mi en el momento en que necesitaba de su ayuda.

Agradezco a mi asesor al Dr. Pedro Bello López por su paciencia, conocimiento y dedicación durante este tiempo, por darme la oportunidad de poder trabajar con el en este proyecto.

Agradecimiento a la Vicerrectoría de Investigación y Estudios de Posgrado por el apoyo económico brindado para terminar este trabajo de tesis dentro del proyecto VIEP 2024: aplicando ciencia de datos para analizar cadenas genómicas.

Tabla de contenido

Resumen

En el ámbito matemático, un grafo emerge como una herramienta fundamental para representar las relaciones entre objetos o entidades. Este concepto consta de nodos (vértices) y aristas (arcos) que conectan dichos nodos y su versatilidad lo convierte en un instrumento clave en la resolución de problemas en diversas disciplinas, desde la computación y las matemáticas hasta la biología, la logística, la ingeniería y las ciencias sociales. La teoría de grafos, por tanto, proporciona una estructura conceptual que permite modelar y analizar situaciones complejas donde existen conexiones y relaciones.

En este contexto, se plantea el desafío de la planificación de rutas de logística utilizando grafos, un problema recurrente en la gestión de la cadena de suministro. Este problema busca determinar la manera óptima de transportar productos o bienes desde un origen hasta un destino final, teniendo en cuenta factores cruciales como la distancia, el tiempo, los costos y las restricciones de capacidad.

La propuesta para abordar este problema implica la modelización de la planificación de rutas mediante la aplicación de la teoría de grafos, específicamente utilizando algoritmos asociados a problemas de flujo máximo. Estos algoritmos, que encuentran su aplicación en una variedad de contextos, se ajustan de manera idónea para resolver eficientemente problemas logísticos complejos.

La innovación clave radica en la implementación de estos algoritmos en un sistema web interactivo, proporcionando una herramienta que no solo resuelve el problema, sino que también permite visualizar de manera dinámica el grafo que representa la problemática de la ruta de transporte. Este enfoque no solo ofrece soluciones eficaces a la planificación logística, sino que también proporciona una herramienta práctica y visualmente comprensible para profesionales y tomadores de decisiones en logística.

En resumen, esta tesis propone un enfoque integral y aplicado para abordar la planificación de rutas logísticas, aprovechando la potencia de la teoría de grafos y los algoritmos de flujo máximo, integrados de manera accesible a través de una plataforma web interactiva. Este enfoque no solo aborda eficazmente el problema logístico, sino que también demuestra el potencial de la teoría de grafos como una herramienta valiosa en la resolución de problemas complejos en el ámbito de la logística y la cadena de suministro.

Capítulo 1. Introducción

La logística y la gestión de la cadena de suministro son elementos cruciales en el funcionamiento eficiente de cualquier organización. La planificación de rutas logísticas se presenta como un desafío complejo, donde la optimización del transporte de bienes desde su origen hasta su destino final no solo impulsa la eficiencia operativa, sino que también tiene un impacto directo en los costos y la satisfacción del cliente. En este contexto, la teoría de grafos surge como una herramienta potente y versátil para abordar este desafío.

Los grafos, con sus nodos y aristas, proporcionan una representación estructurada de las conexiones y relaciones entre elementos (ver ejemplo Figura 1.1), lo que resulta fundamental para modelar y resolver problemas en diversas disciplinas. En particular, la teoría de grafos se ha convertido en un recurso invaluable en la optimización de procesos y toma de decisiones, ofreciendo soluciones eficientes para problemas complejos [1].

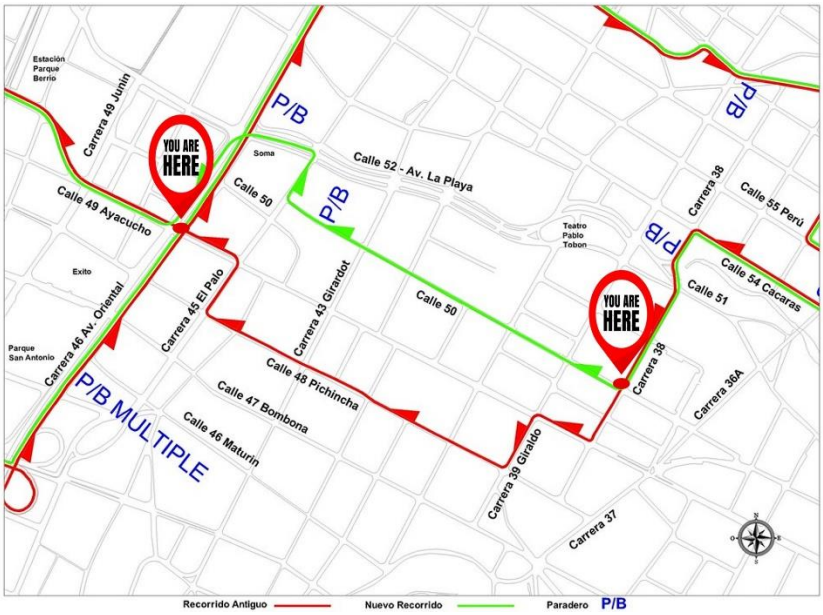


Figura 1.1. Ejemplo de mapa de ruta

En este proyecto de tesis, nos proponemos abordar el problema de la planificación de rutas logísticas utilizando grafos como una herramienta principal. La ineficiencia en los métodos tradicionales de planificación de rutas y la necesidad de una optimización más efectiva del transporte han motivado esta investigación. El enfoque principal consiste en modelar el problema mediante la representación gráfica de la red logística y aplicar algoritmos de flujo máximo [7], aprovechando la capacidad de la teoría de grafos para resolver problemas de optimización.

La teoría de grafos [6] aplicada al problema de flujo máximo tiene diversas aplicaciones como se indica en la Figura 1.2. En particular el trabajo aquí desarrollado tiene que ver con la logística y la cadena de suministro.

Figura 1.2. diagrama de las principales aplicaciones en teoría de grafos para el problema de flujo máximo.

1.1 Planteamiento del problema

El planteamiento del problema se centra en la ineficiencia y complejidad asociadas con la planificación de rutas logísticas. En el ámbito de la logística y la cadena de suministro, la necesidad de optimizar el transporte de productos desde su origen hasta su destino final es esencial para mejorar la eficiencia y reducir costos. Los métodos tradicionales de planificación de rutas a menudo enfrentan desafíos al considerar factores cruciales como la distancia, el tiempo, los costos y las restricciones de capacidad. Este problema se abordará utilizando la teoría de grafos y algoritmos asociados para proporcionar soluciones más eficaces y dinámicas [7].

1.2 Objetivo general del proyecto de tesis

El objetivo general de este proyecto de tesis es desarrollar un Sistema de Planificación de Rutas de Logística utilizando Grafos, que aproveche la teoría de grafos y los algoritmos de flujo máximo. El sistema propuesto buscará optimizar la asignación de recursos en la planificación de rutas, considerando diversos factores, con el fin de mejorar la eficiencia del transporte logístico.

1.3 Objetivos específicos del proyecto de tesis

- Modelización del Problema: Definir y establecer un modelo de grafos que represente de manera precisa la problemática de la planificación de rutas logísticas.
- Selección de Algoritmos: Identificar y aplicar algoritmos específicos de grafos, particularmente aquellos relacionados con el flujo máximo, para resolver eficientemente el problema de planificación de rutas.
- Implementación del Sistema: Desarrollar un sistema en web que integre la modelización de grafos y la aplicación de algoritmos, permitiendo la visualización interactiva de las soluciones propuestas.
- Evaluación y Validación: Evaluar la eficacia y eficiencia del sistema mediante la comparación de resultados obtenidos con métodos tradicionales de planificación de rutas.
- Interfaz de Usuario Intuitiva: Asegurar que el sistema sea accesible y fácil de usar para profesionales en logística, proporcionando una interfaz intuitiva para la entrada de datos y la interpretación de resultados.

1.4 Metodología para el desarrollo del proyecto

La ingeniería de software es esencial para el desarrollo de sistemas de software complejos y de alta calidad, abarca una amplia gama de actividades y disciplinas que contribuyen a la creación de productos de software robustos, eficientes y sostenibles [2].

Enfoque clásico de la Ingeniería de Software. Los modelos clásicos de la ingeniería de software incluyen el modelo en cascada, el modelo en V y el modelo de prototipado, cada uno con enfoques únicos para el desarrollo de software. El modelo en cascada, uno de los más antiguos y conocidos, sigue un proceso secuencial donde cada fase debe completarse antes de iniciar la siguiente, ideal para proyectos con requisitos bien definidos y estables. El modelo en V es una extensión del modelo en cascada que incorpora fases de verificación y validación paralelas a cada etapa del desarrollo, asegurando una mayor calidad del producto. Por otro lado, el modelo de prototipado se centra en la creación rápida de prototipos para comprender y refinar los requisitos del cliente a través de la retroalimentación continua, lo que es especialmente útil cuando los requisitos no están claramente definidos desde el principio. Estos modelos han sido fundamentales en la evolución de las metodologías de desarrollo de software, proporcionando estructuras claras y enfoques disciplinados para la creación de sistemas de software robustos y de alta calidad.

Enfoque ágil de la Ingeniería de Software. El modelo ágil de la ingeniería de software es un enfoque iterativo e incremental que prioriza la flexibilidad, la colaboración y la entrega continua de software funcional. En contraste con los modelos tradicionales, el ágil se adapta rápidamente a los cambios en los requisitos y permite una retroalimentación constante del cliente, lo que mejora la satisfacción del usuario final. Las metodologías ágiles, como Scrum, Kanban y Extreme Programming (XP), fomentan la comunicación abierta y la colaboración estrecha entre todos los miembros del equipo, promoviendo ciclos de desarrollo cortos y la entrega frecuente de incrementos de producto. Esta capacidad para responder rápidamente a las necesidades cambiantes y su enfoque en el valor para el cliente hacen del modelo ágil una opción preferida para proyectos de software complejos y dinámicos.

La metodología utilizada en este proyecto es un modelo clásico de desarrollo de sistemas denominado cascada con el fin de implementar un prototipo funcional desarrollado en PHP con un repositorio de los casos de prueba para verificar el funcionamiento del sistema que se ocupara para el desarrollo del proyecto con los siguientes pasos (ver ejemplo figura 1.3):

- Revisión de la Literatura: Explorar y analizar la literatura existente sobre la planificación de rutas logísticas, teoría de grafos y algoritmos de flujo máximo.
- Diseño del Modelo de Grafos: Definir y diseñar un modelo de grafos que refleje de manera efectiva la problemática de la planificación de rutas.
- Selección de Tecnologías: Identificar las tecnologías y herramientas más apropiadas para la implementación del sistema en web.

- Desarrollo del Sistema: Implementar el sistema, integrando la modelización de grafos y la aplicación de algoritmos de flujo máximo.
- Pruebas y Evaluación: Realizar pruebas exhaustivas para validar la eficacia y eficiencia del sistema utilizando casos de prueba y escenarios logísticos simulados.
- Análisis de Resultados: Comparar los resultados obtenidos con otros métodos de planificación de rutas, analizando la mejora en la eficiencia del transporte logístico.
- Documentación y Redacción: Documentar el proceso de desarrollo y redactar la tesis, incorporando los hallazgos, discusiones y conclusiones derivadas del proyecto.

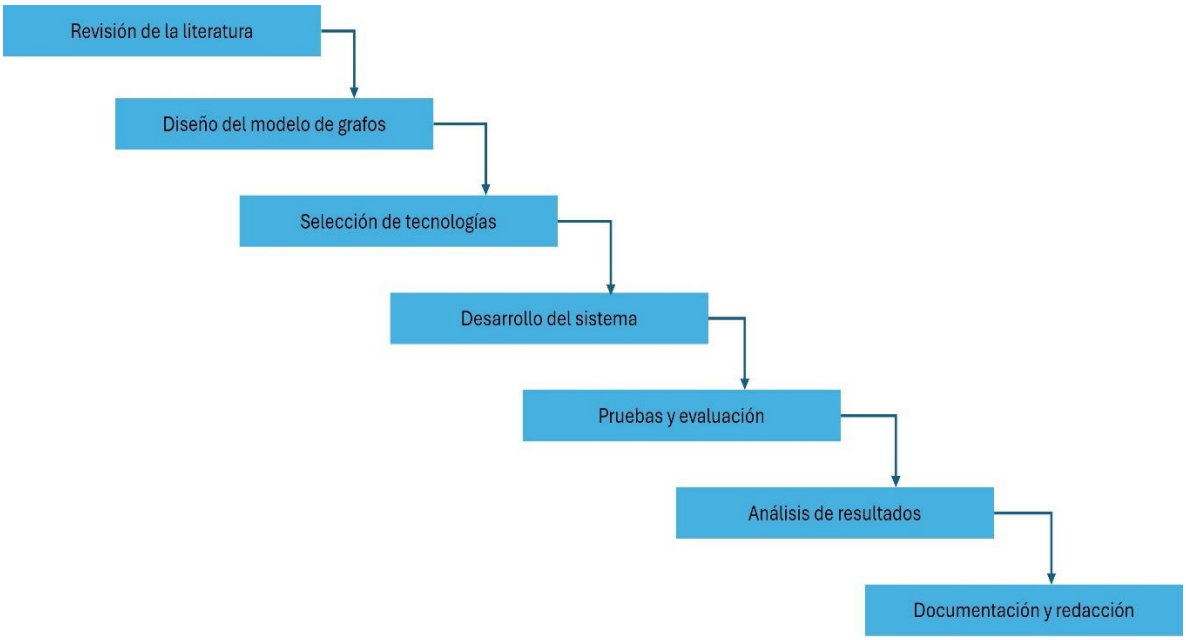


Figura 1.3 Diseño visual del modelo en cascada.

El modelo en cascada [8] es uno de los enfoques tradicionales más utilizados en la ingeniería de software para el desarrollo de sistemas computacionales, el modelo de cascada sigue un proceso secuencial y lineal para el desarrollo de software, en el que cada fase debe completarse antes de que comience la siguiente fase, sin embargo, hay enfoques donde al llegar a la última fase, puede regresar a alguna de las fases anteriores (ver Figura 1.4) con el objetivo de hacer más flexible el modelo.

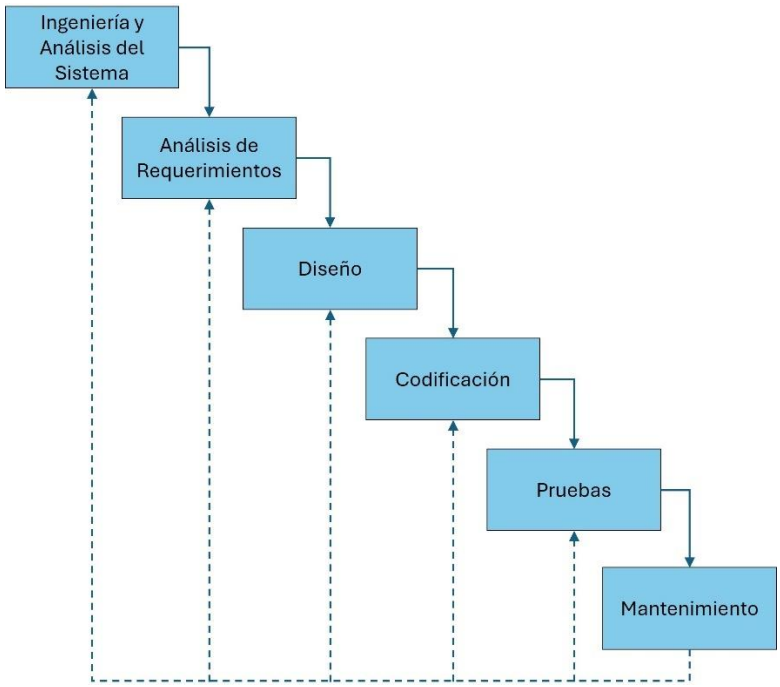


Figura 1.4. diagrama modificado del modelo de cascada

Análisis de Requisitos: En esta fase se obtienen y analizan los requisitos del sistema a desarrollar. Los requisitos pueden ser funcionales (qué debe hacer el sistema) y no funcionales (cómo debe comportarse el sistema), generalmente los requerimientos se obtienen a través de reuniones con el cliente, entrevistas, cuestionarios, análisis de documentos y creación de un documento de especificación de requisitos.

Diseño: En esta fase se elabora un diseño del sistema basado en los requisitos obtenidos en la fase anterior. El diseño incluye la arquitectura del sistema, el diseño de datos, el diseño de interfaces y el diseño de componentes del sistema, en esta etapa se crean diagramas de arquitectura, diagramas de flujo de datos, esquemas de base de datos, y diseño de interfaces entre otros artefactos.

Codificación: En esta fase se lleva a cabo la codificación del sistema. Los desarrolladores escriben el código fuente utilizando el diseño especificado en la fase anterior, es decir se realiza la programación, el desarrollo de módulos y componentes, en algunos casos se realizan aquí las pruebas unitarias.

Pruebas: en esta fase se realizan pruebas exhaustivas para asegurar que el sistema cumple con los requisitos especificados en la etapa de análisis y diseño y está libre de errores. Las pruebas incluyen la verificación de funcionalidades y la validación de requisitos, aquí se aplican pruebas unitarias, pruebas de integración, pruebas de sistema, pruebas de aceptación y depuración de errores. También en esta fase el sistema se pone en funcionamiento en el entorno de producción y se entrega a los usuarios finales. Puede

incluir la instalación, configuración y transferencia de datos, a través de la instalación del software, migración de datos, formación a usuarios y configuración de hardware y software.

Mantenimiento: finalmente en esta fase, después del despliegue, el sistema necesita ser mantenido para corregir errores, mejorar el rendimiento, o adaptarlo a cambios en el entorno operativo o en los requisitos del usuario, este proceso se realizan actividades como corrección de errores, actualizaciones de software, mejoras de funcionalidad y soporte técnico.

Ventajas del modelo de cascada.

1. La naturaleza secuencial del modelo en cascada es fácil de entender y seguir. Cada fase tiene un inicio y un final claramente definidos, lo que facilita la planificación y la gestión del proyecto.

2. El modelo promueve una estructura bien definida y disciplinada para el desarrollo de software. Cada etapa debe completarse antes de pasar a la siguiente, lo que ayuda a mantener el orden en el proceso de desarrollo.

3. La documentación completa y detallada en cada fase facilita el seguimiento del progreso y la comprensión del sistema. Esto es especialmente útil para proyectos de gran escala y para el mantenimiento futuro.

4. En este modelo es más fácil estimar los costos y los tiempos de entrega, ya que cada fase debe completarse antes de avanzar. Esto permite una mejor gestión de los recursos y el cumplimiento de plazos.

En general el modelo de cascada es muy útil cuando los requerimientos del usuario están bien definidos lo que permite avanzar en el desarrollo del sistema de forma consistente, sin embargo, para proyectos más complejos y dinámicos, es posible que se prefieran enfoques más flexibles e iterativos, como los modelos ágiles [2, 8, 11].

Capítulo 2. Algoritmos de Flujo Máximo

Los algoritmos de flujo máximo son herramientas fundamentales en la teoría de grafos y la optimización de redes. Estos algoritmos abordan problemas donde se busca transportar la mayor cantidad posible de flujo desde una fuente hasta un sumidero a través de una red de conexiones, cada una con una capacidad limitada [1, 7].

En términos simples, el flujo máximo en una red se refiere a la cantidad máxima de recursos (como datos, productos o información) que pueden ser transportados desde la fuente hasta el sumidero, respetando las capacidades de las conexiones intermedias.

Componentes Clave:

Red de Flujo: Representa el sistema en forma de grafo dirigido, donde los nodos son puntos de procesamiento y las aristas son las conexiones entre ellos. Cada arista tiene una capacidad que limita la cantidad de flujo que puede pasar a través de ella.

Fuente y Sumidero: La fuente es el nodo de inicio del flujo, mientras que el sumidero es el nodo destino. El objetivo es maximizar el flujo desde la fuente hasta el sumidero.

Características Principales:

Flujo Válido: Para ser válido, el flujo debe cumplir con las capacidades de las aristas y respetar la conservación del flujo en cada nodo, es decir, la cantidad de flujo que entra en un nodo debe ser igual a la cantidad que sale, excepto en la fuente y el sumidero.

Red Residual: Durante la ejecución del algoritmo, se trabaja con una red residual que representa las capacidades no utilizadas en la red original. Esta red es modificada iterativamente para encontrar el flujo máximo.

Algoritmos Destacados:

- Algoritmo de Ford-Fulkerson
- Algoritmo de Edmonds-Karp
- Algoritmo de Dinic

Los algoritmos de flujo máximo tienen diversas aplicaciones en la vida real, como la optimización de rutas en logística, la asignación eficiente de recursos en redes de comunicación, la maximización del flujo de datos en redes de computadoras, entre otros.

Los algoritmos de flujo máximo son esenciales para resolver problemas de optimización en redes. Su capacidad para determinar la cantidad óptima de flujo en sistemas complejos tiene un impacto significativo en la eficiencia y la toma de decisiones en diversas disciplinas.

2.1 Conceptos básicos de la teoría de grafos

La teoría de grafos es una rama de las matemáticas y la informática que estudia las relaciones entre objetos abstractos mediante la representación gráfica de conexiones entre ellos. Este campo es esencial para modelar y resolver problemas en una variedad de disciplinas, desde la informática hasta la logística y la biología [6]. A continuación, se exploran en detalle algunos conceptos fundamentales de la teoría de grafos:

Un grafo *G* es una estructura matemática compuesta por un conjunto de nodos (también llamados vértices) y un conjunto de aristas (también llamados arcos) que conectan esos nodos. La representación gráfica de un grafo suele utilizarse para visualizar las relaciones entre los elementos del conjunto [3, 9].

Nodos (Vértices): Son los elementos fundamentales del grafo. Cada nodo representa un punto o entidad en el conjunto y se denota comúnmente por letras, números o cualquier otro símbolo.

Aristas (Arcos): Son las conexiones entre nodos. Pueden ser dirigidas (con una dirección específica) o no dirigidas (bidireccionales). Cada arista representa una relación entre dos nodos.

Típicamente un grafo (Figura 2.1) se representa visualmente de la siguiente manera.

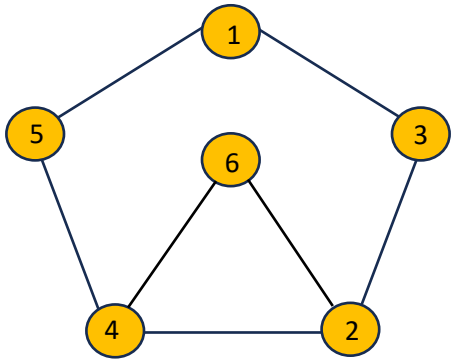


Figura 2.1 Grafo básico

Nodos Adyacentes: Dos nodos son adyacentes si están conectados por una arista. La relación de adyacencia se representa a menudo mediante una matriz de adyacencia o una lista de adyacencia.

Camino: Una secuencia de nodos donde cada par de nodos consecutivos está conectado por una arista. La longitud de un camino es el número de aristas en él.

Ciclo: Un camino cerrado donde el nodo de inicio y el de fin son el mismo. Un grafo sin ciclos se denomina grafo acíclico.

Grafo Dirigido: También conocido como dígrafo (ver figura 2.2), tiene aristas con una dirección específica. La relación de conexión entre dos nodos puede ser unidireccional.

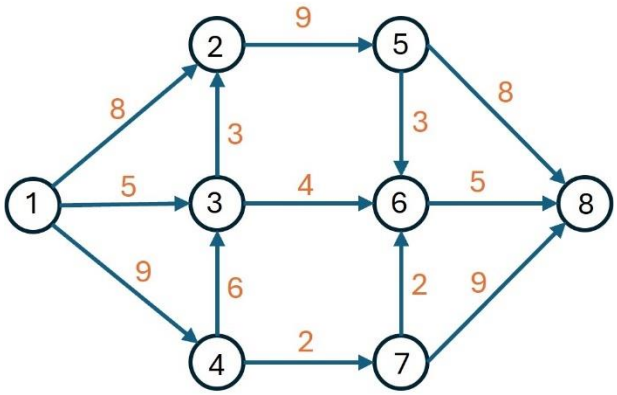


Figura 2.2. Grafo Dirigido o Dígrafo

Grafo No Dirigido: Las aristas no tienen una dirección específica, lo que significa que la relación es bidireccional.

Grado de Entrada: En un grafo dirigido, es el número de aristas entrantes a un nodo.

Grado de Salida: En un grafo dirigido, es el número de aristas salientes de un nodo.

Grado de un Nodo (en un grafo no dirigido): Es el número total de aristas conectadas al nodo.

Grafo Conexo: Un grafo en el que hay un camino entre cada par de nodos.

Componentes Conectados: En un grafo no dirigido, son subgrafos conexos. Cada componente conectado es un subconjunto de nodos conectados entre sí.

En algunos casos, las aristas pueden tener un peso o valor asociado. Estos grafos ponderados son útiles para modelar situaciones donde las conexiones tienen costos o medidas específicas.

Matriz de Adyacencia: Es una matriz cuadrada donde cada entrada

$A[i][j]$ representa si hay una arista entre los nodos i y j.

Puede incluir información adicional como pesos o capacidades.

Lista de Adyacencia: Se representa como una lista de listas, donde cada nodo tiene una lista que contiene sus nodos adyacentes. Es más eficiente en términos de espacio para grafos dispersos.

Ejemplo de grafos. Aplicado en Redes Sociales:

Supongamos que los nodos son personas y las aristas representan amistades (figura 2.3). Un camino podría indicar una cadena de amistades conectadas, y la detección de componentes conectados podría identificar grupos de amigos dentro de la red social.

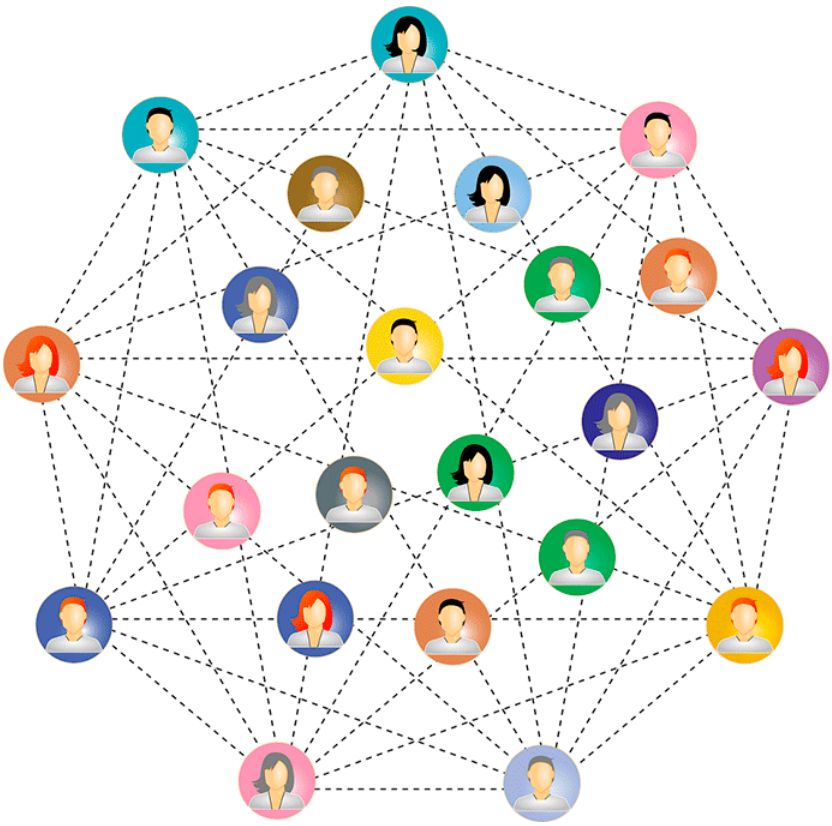


Figura 2.3. Ejemplo de grafo de redes sociales (https://www.inesem.es/revistadigital/informatica-y-tics/teoria-grafos/)

Ejemplo Aplicado en Sistemas de Carreteras:

Imaginemos un grafo donde los nodos son ciudades y las aristas son carreteras (figura 2.4). Este grafo modela la conectividad entre ciudades y puede utilizarse para encontrar la ruta más corta entre dos ubicaciones o identificar ciudades con mayor grado de conexión.

La teoría de grafos proporciona un marco conceptual para modelar y entender relaciones entre elementos en diversos contextos. Su aplicación se extiende desde la informática hasta la logística, la biología y más allá, ofreciendo herramientas poderosas [1,7].

Figura 2.4. Ejemplo de grafo de carreteras (https://marzomates.webs.ull.es/grafos-y-redes-las-comunicaciones-y-la-logistica/)

Recorrido a lo Ancho (Búsqueda en Amplitud):

El recorrido a lo ancho es un algoritmo utilizado para recorrer o buscar en un grafo de manera sistemática. A diferencia del recorrido a lo largo (profundidad), donde se explora una rama completa antes de retroceder, el recorrido a lo ancho explora todos los vecinos directos de un vértice antes de pasar a los vecinos de los vecinos. Este enfoque permite encontrar la ruta más corta entre dos vértices en un grafo no ponderado.

Proceso del Recorrido a lo Ancho:

- Se visita el nodo inicial.
- Después de visitar el nodo inicial se visitan todos los sucesores.

- Después los sucesores de los sucesores.
- Se repiten los pasos 1 y 2 hasta encontrar un nodo sin arcos salientes o ya visitados.
- Si después de visitar todos los descendientes del primer nodo, todavía quedan mas nodos por visitar, se repite el proceso reiniciando.

Aplicaciones del Recorrido a lo Ancho:

- Búsqueda de Rutas: Encuentra el camino más corto entre dos vértices en un grafo no ponderado.
- Análisis de Redes Sociales: Identifica la conectividad entre personas en redes sociales.
- Resolución de Problemas de Grafos: Útil en la resolución de problemas como el problema del vendedor viajero o la planificación de rutas de transporte.

Complejidad del Recorrido a lo Ancho:

- La complejidad temporal del recorrido a lo ancho es O(V + E), donde V es el número de vértices y E es el número de aristas en el grafo. Esto se debe a que cada vértice y arista se visita una vez durante el proceso.

El recorrido a lo ancho es un algoritmo fundamental en la teoría de grafos que permite explorar la estructura de un grafo de manera sistemática y eficiente. Su capacidad para encontrar la ruta más corta entre dos vértices en un grafo no ponderado lo convierte en una herramienta valiosa en una variedad de aplicaciones, desde la planificación de rutas hasta el análisis de redes sociales y la resolución de problemas de optimización.

Para el siguiente ejemplo, habría varios recorridos en anchura posibles, entre los cuales podríamos mencionar los siguientes (figura 2.5)

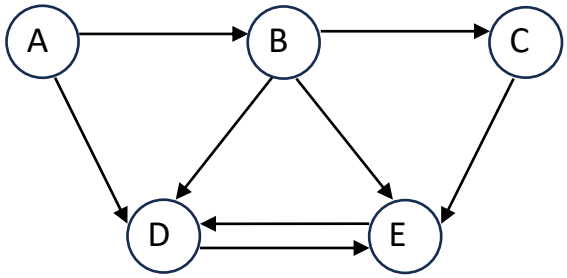
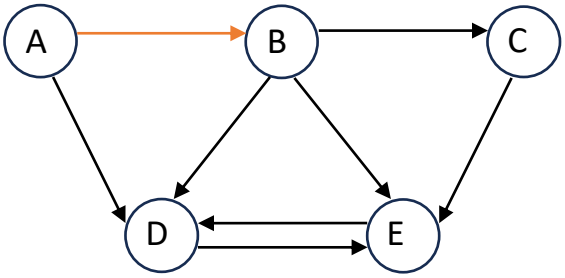


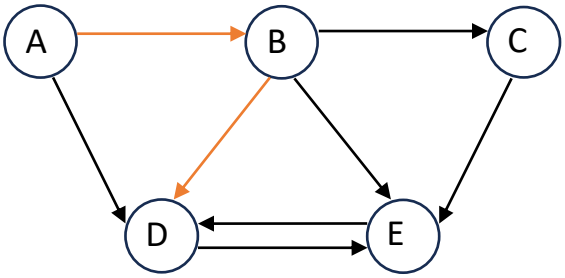
Figura 2.5 Recorrido a lo ancho

Visitando primero el nodo izquierdo empezando por el nodo A:

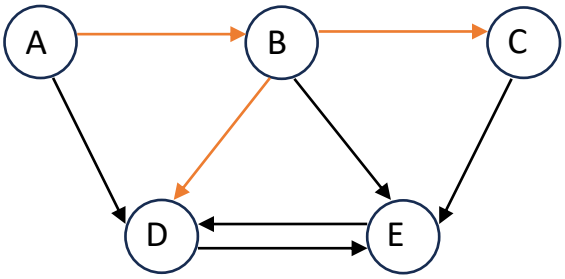
A → B



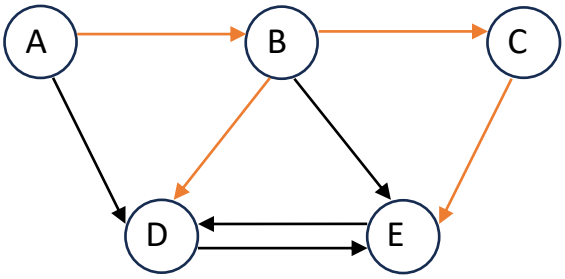
A → B → D



A → B → D → C

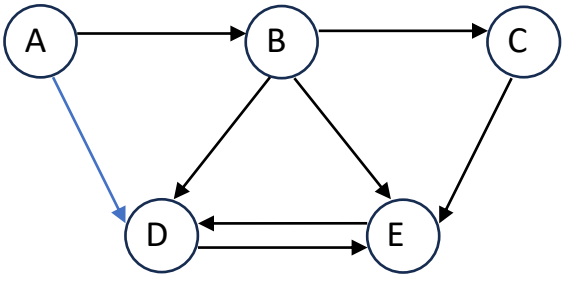


A → B → D → C → E

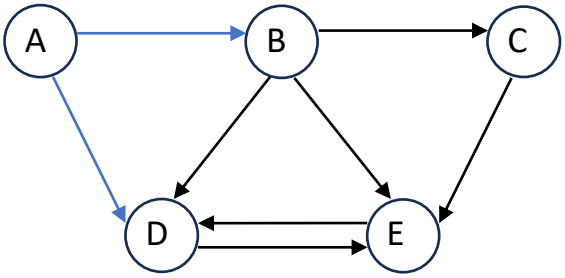


Visitando primero al nodo derecho empezando desde el nodo A:

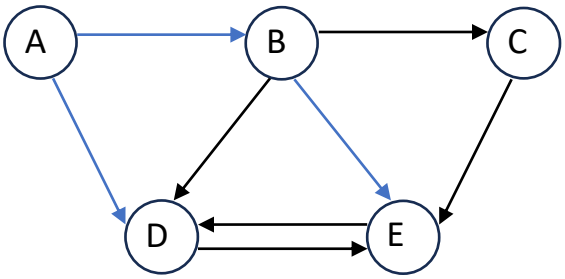
A → D



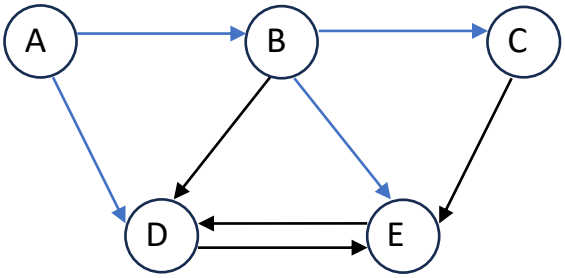
A → D → B



A → D → B → E



A → D → B → E → C



Recorrido en Profundidad (Búsqueda en Profundidad):

El recorrido a la profundidad es un algoritmo utilizado para recorrer o buscar en un grafo de manera sistemática. A diferencia del recorrido a lo ancho, donde se explora todos los vecinos directos de un vértice antes de pasar a los vecinos de los vecinos, el recorrido a la profundidad explora tanto como sea posible a lo largo de cada rama antes de retroceder.

Proceso del Recorrido en Profundidad:

- Se toma un nodo “S” como inicial y se marca
- A continuación, se toma y se marca un nodo no marcado adyacente a “S”, y ese nodo pasa a ser el nuevo nodo de partida, dejando posiblemente por el momento al nodo inicial original con nodos no explorados.
- La búsqueda continua por el grafo hasta que el camino en curso finalice con un grafo de salida igual a cero, o bien en un nodo en que todos los nodos adyacentes estén marcados.
- A continuación, la búsqueda vuelve al último nodo que todavía tenga nodos adyacentes sin marcar, y continúa marcando todos los nodos de forma recursiva hasta que ya no queden nodos sin marcar [3, 9].

Aplicaciones del Recorrido en Profundidad:

- Búsqueda de Rutas: Explora todas las posibles rutas en un grafo, pero no necesariamente encuentra la más corta.
- Detección de Ciclos: Útil para detectar ciclos en un grafo dirigido o no dirigido.
- Topología: Ayuda a realizar ordenamientos topológicos en grafos dirigidos acíclicos.

Complejidad del Recorrido en Profundidad:

La complejidad temporal del recorrido a la profundidad es O(V + E), donde V es el número de vértices y E es el número de aristas en el grafo. Esto se debe a que cada vértice y arista se visita una vez durante el proceso.

El recorrido a la profundidad es un algoritmo fundamental en la teoría de grafos que permite explorar la estructura de un grafo de manera sistemática y eficiente. A diferencia del recorrido a lo ancho, el recorrido a la profundidad se adentra profundamente en una rama antes de retroceder, lo que lo hace útil para ciertas aplicaciones, como la detección de ciclos y la realización de ordenamientos topológicos.

Para el siguiente ejemplo, habría varios recorridos en anchura posibles, entre los cuales podríamos mencionar los siguientes (figura 2.6)

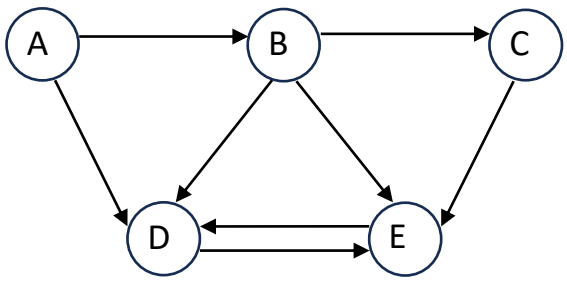
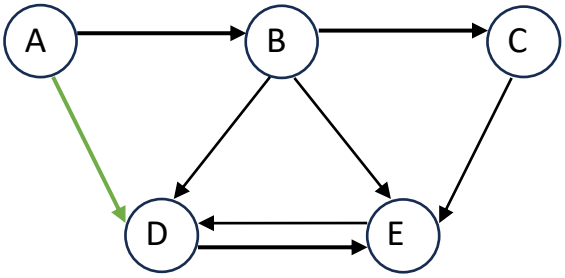


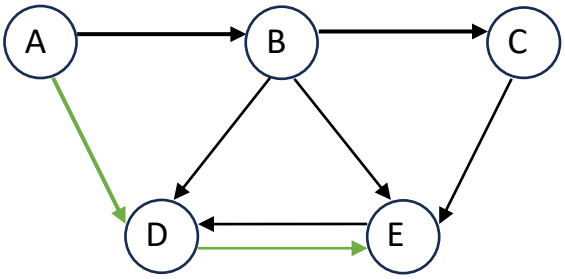
Figura 2.6 Recorrido en profundidad

Visitando primero el nodo izquierdo empezando por el nodo A:

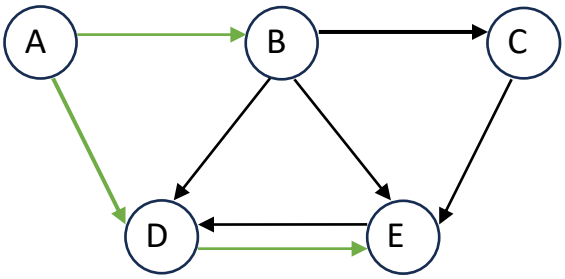
A → D



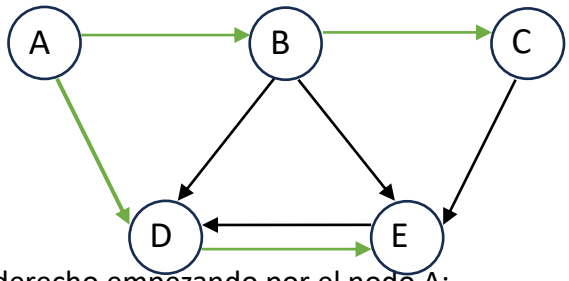
A → D → E



A → D → E → B

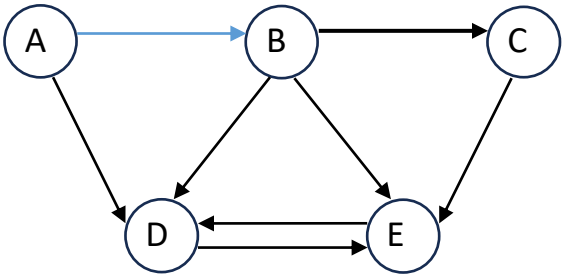


A → D → E → B → C

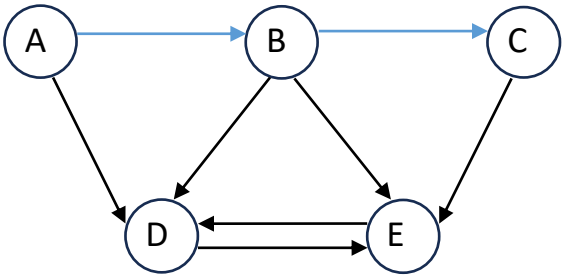


Visitando primero el nodo derecho empezando por el nodo A:

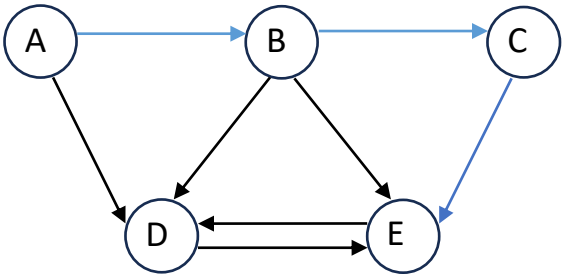
A → B



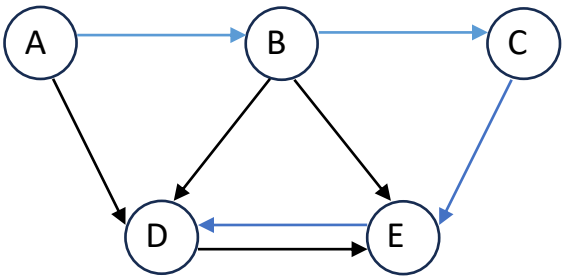
A → B → C



A → B → C → E



A → B → C → E → D



2.2 Algoritmo de Ford-Fulkerson

El Algoritmo de Ford-Fulkerson [7] es un algoritmo fundamental en teoría de grafos utilizado para encontrar el flujo máximo en una red de flujo. Fue propuesto por L.R. Ford, Jr. y D.R. Fulkerson en 1956. Aunque es un algoritmo poderoso, es esencial destacar que su implementación original puede no converger en todos los casos. Para abordar esto, diversas variantes y mejoras, como el Algoritmo de Edmonds-Karp, han sido propuestas.

Funcionamiento del Algoritmo:

Inicialización:

- Se inicia el flujo en todas las aristas a cero.
- Se crea una red residual, inicialmente idéntica a la red original.

Camino de Aumento:

- Se busca un camino desde la fuente hasta el sumidero en la red residual.
- Un camino válido es aquel donde todas las aristas tienen capacidad residual positiva.

Aumento de Flujo:

- Se determina la capacidad residual mínima a lo largo del camino de aumento.
- Se incrementa el flujo a lo largo del camino por la capacidad residual mínima.
- Se actualizan las capacidades residuales en la red residual, disminuyendo la capacidad en las aristas utilizadas y aumentando en las aristas inversas.

Repetición:

- Se repiten los pasos 2 y 3 hasta que no se encuentre ningún camino de aumento.

Flujo Máximo:

- El flujo máximo se encuentra cuando no hay más caminos de aumento en la red residual.

Ejemplo Aplicado en Redes de Computadoras:

En el contexto de redes de computadoras, el algoritmo de Ford-Fulkerson se puede aplicar para determinar el flujo máximo de datos desde un origen hasta un destino, considerando las capacidades de los enlaces de la red. Esto es esencial para optimizar la transmisión de datos en redes complejas.

El algoritmo de Ford-Fulkerson tiene limitaciones, especialmente en casos donde las capacidades no son números enteros. En tales situaciones, pueden surgir problemas de convergencia. Para abordar esto, se han propuesto mejoras, como la variante de Edmonds-Karp, que utiliza la búsqueda de amplitud para garantizar convergencia rápida y eficiente.

A pesar de sus limitaciones, el Algoritmo de Ford-Fulkerson sigue siendo una herramienta valiosa en teoría de grafos y optimización de redes. Su aplicabilidad en diversos problemas del mundo real demuestra su importancia en la resolución de desafíos logísticos, de redes de comunicación y otros problemas de flujo en sistemas complejos.

2.3 Algoritmo de Edmonds-Karp

El Algoritmo de Edmonds-Karp es una mejora específica del algoritmo de Ford-Fulkerson diseñada para abordar algunas de las limitaciones de convergencia y eficiencia del algoritmo original. Este algoritmo se destaca por su capacidad para encontrar de manera eficiente el flujo máximo en una red mediante la utilización de la búsqueda de amplitud (BFS) para hallar caminos de aumento, asegurando la convergencia rápida hacia el flujo máximo [1, 7].

Funcionamiento del Algoritmo:

Inicialización:

- Inicia el flujo en todas las aristas a cero.
- Crea la red residual, que inicialmente es idéntica a la red original.

Búsqueda de Camino Más Corto:

- Utiliza BFS para encontrar el camino más corto desde la fuente hasta el sumidero en la red residual.
- La búsqueda de amplitud garantiza que se encuentre el camino más corto en términos de cantidad de aristas.

Aumento de Flujo y Actualización de Red Residual:

- Aumenta el flujo a lo largo del camino encontrado y actualiza las capacidades residuales en la red residual.
- La actualización de capacidades residuales implica reducir la capacidad de las aristas utilizadas y agregar aristas inversas con capacidades residuales.

Repetición:

- Repite los pasos 2 y 3 hasta que no sea posible encontrar más caminos de aumento.

Ventajas Clave del Algoritmo de Edmonds-Karp:

Convergencia Rápida:

La utilización de BFS asegura la convergencia rápida hacia el flujo máximo, evitando ciclos infinitos que podrían surgir en el algoritmo de Ford-Fulkerson.

Camino Más Corto:

Encuentra caminos de aumento más cortos, garantizando una convergencia eficiente y previniendo problemas de desempeño asociados con caminos más largos.

Ejemplo Aplicado en Logística:

Supongamos una red logística donde nodos representan ubicaciones y aristas indican rutas entre ellas con capacidades asociadas. La fuente sería un centro de distribución, el sumidero podría ser un punto de entrega y las aristas representarían caminos posibles. El algoritmo de Edmonds-Karp podría ayudar a encontrar la mejor ruta para transportar productos, maximizando la eficiencia y minimizando los costos.

2.4 Algoritmo de Dinic

El Algoritmo de Dinic es un algoritmo eficiente para encontrar el flujo máximo en una red de flujo. Fue propuesto por E.A. Dinic en 1970 y se destaca por su complejidad temporal mejorada, siendo más eficiente que el algoritmo clásico de Ford-Fulkerson en ciertos casos. Vamos a explorar detalladamente cómo funciona y cómo se aplica en la práctica [1, 7].

Funcionamiento del Algoritmo de Dinic:

Inicialización:

- Se inicia el flujo en todas las aristas a cero.
- Se crea la red residual, inicialmente idéntica a la red original.

Niveles y Capas:

- Se asignan niveles a los nodos en la red residual utilizando la técnica de la búsqueda de amplitud (BFS). Esto crea capas en la red residual.

Bloqueo de Flujo:

- Se bloquea el flujo en las capas no consecutivas, evitando caminos que no aumentarían el flujo.

Flujo de Bloqueo:

- Se utiliza DFS (búsqueda en profundidad) para encontrar caminos de bloqueo y aumentar el flujo.

Repetición:

- Se repiten los pasos 2-4 hasta que no se pueda encontrar más flujo bloqueante.

Flujo Máximo:

- El flujo máximo se alcanza cuando no hay más caminos de bloqueo en la red residual.

Ventajas Clave del Algoritmo de Dinic:

Complejidad Temporal Mejorada: El algoritmo de Dinic tiene una complejidad temporal de $O(V^2 * E)$, donde V es el número de nodos y E es el número de aristas. Esta complejidad es

más eficiente en comparación con el algoritmo de Ford-Fulkerson en ciertos casos, especialmente en grafos dispersos.

Evita Caminos de Ciclo Infinito: Al utilizar niveles y bloqueo de flujo, el algoritmo evita caminos de ciclo infinito que podrían surgir en el algoritmo de Ford-Fulkerson.

Ejemplo Aplicado en Flujo de Redes:

Optimización de Rutas en Red de Transporte: Consideremos una red de transporte con nodos representando ciudades y aristas denotando rutas entre ellas. Cada arista tiene una capacidad máxima de transporte y un costo asociado. El algoritmo de Dinic puede ayudar a optimizar las rutas de transporte, maximizando el flujo y minimizando el costo total.

Ejemplo Aplicado en Redes de Comunicación:

Optimización de Tráfico de Datos: En una red de comunicaciones, donde los nodos son dispositivos y las aristas son conexiones de red, el algoritmo de Dinic puede ayudar a determinar la capacidad máxima de transmisión de datos entre nodos, optimizando así el tráfico de datos.

Consideraciones Importantes:

Red Ponderada: El algoritmo de Dinic se puede aplicar eficientemente en grafos ponderados, donde las aristas tienen capacidades asociadas.

Uso de Estructuras de Datos Eficientes: Para implementar el algoritmo de Dinic eficientemente, se utilizan estructuras de datos como listas de adyacencia y estructuras de cola que admitan operaciones de búsqueda y actualización de manera eficiente.

El Algoritmo de Dinic se destaca como una opción eficiente para problemas de flujo máximo en redes. Su capacidad para evitar ciclos infinitos y mejorar la complejidad temporal en ciertos casos lo convierte en una herramienta valiosa en la optimización de flujos en sistemas complejos como redes de transporte, comunicaciones y más.

Capítulo 3. Algoritmo de planificación de rutas de logística propuesto

Además de los algoritmos antes descritos en el capítulo anterior, existen otras propuestas algorítmicas para tratar el problema de planificación de rutas de logística que se basan en diferentes estrategias, algunos de estos algoritmos son [1, 6]:

1. Algoritmo de Clarke-Wright. Es un método heurístico para resolver el problema del ruteo de vehículos (VRP). Comienza con una solución inicial en la que cada cliente es atendido por un vehículo individual, y luego combina rutas para reducir la distancia total recorrida.

2. Algoritmo de Búsqueda Tabú. Es un método de optimización que utiliza una lista tabú para evitar ciclos y mejorar iterativamente una solución inicial. Se exploran movimientos en el espacio de soluciones y se prohíben (tabú) ciertos movimientos para evitar caer en mínimos locales.

3. Algoritmo de Colonia de Hormigas. Es un algoritmo basado en el comportamiento de las hormigas que buscan caminos cortos entre sus nidos y fuentes de alimento. Se utiliza una colonia de agentes (hormigas) que depositan feromonas en las rutas recorridas, guiando a otras hormigas hacia rutas prometedoras.

4. Algoritmos Genéticos. Son métodos de búsqueda basados en los principios de la selección natural y la genética. Utilizan una población de soluciones candidatas que evolucionan a través de operaciones como selección, cruce y mutación.

5. Algoritmo de Búsqueda Local. Es una técnica heurística que mejora una solución inicial explorando soluciones vecinas. Si se encuentra una mejor solución en la vecindad, se mueve a esta nueva solución y repite el proceso.

6. Algoritmo de Programación Dinámica. Es un enfoque de optimización que resuelve problemas dividiéndolos en subproblemas más pequeños y combinando sus soluciones. Es especialmente útil para problemas estructurados, como el problema del viajero (TSP).

7. Algoritmo de Branch and Bound. Es una técnica exacta que divide el problema en subproblemas (branching) y utiliza límites para eliminar subproblemas que no pueden contener la solución óptima (bounding).

En este proyecto utilizamos el algoritmo de Ford-Fulkersone para determinar el Flujo máximo modelado con grafos.

3.1 Pseudocódigo del algoritmo de planificación

El algoritmo de Ford-Fulkerson se utiliza para encontrar el flujo máximo utilizando la teoría de grafos para representar la red de flujo. El algoritmo funciona buscando los caminos acumulando los pesos de las aristas en el grafo hasta que no se puedan encontrar más. A continuación, describimos el algoritmo en pseudocódigo.

Algoritmo Ford-Fulkerson(*G, s, t*)

Entrada:

G: grafo de flujo con capacidades *c*(*u, v*)

s: nodo fuente

t: nodo final

Salida:

El valor máximo de flujo *f* y el flujo en cada arista del grafo

1. **Inicializar** el flujo *f*(*u*, *v*) = 0 para todas las aristas (*u*, *v*) en *G*
2. **Mientras** haya un camino aumentante *p* de *s* a *t* en la red residual *G_f*:
    3. **Encontrar** el flujo aumentante *f_p* en el camino *p*: // usando *DFS()*
        4. *f_p* = *min*{*c_f*(*u*, *v*) | (*u*, *v*) está en *p*}
    5. **Para** cada arista (*u*, *v*) en *p*:
        6. *f*(*u*, *v*) = *f*(*u*, *v*) + *f_p* // Aumentar el flujo en las aristas del camino
        7. *f*(*v*, *u*) = *f*(*v*, *u*) - *f_p* // Reducir el flujo en las aristas inversas del camino
8. **Devolver** el flujo total *f*

3.2 Descripción del algoritmo Ford Fulkerson

A continuación, se describe el algoritmo y las principales funciones que fueron implementadas en el sistema desarrollado.

Función FordFulkerson:

- Inicializar el flujo en todas las aristas a cero.

Mientras exista un camino aumentante desde la fuente hasta el destino:

  - Encontrar un camino aumentante utilizando una búsqueda en amplitud (BFS) o búsqueda en profundidad (DFS).
  - Determinar el flujo máximo posible en el camino encontrando la capacidad mínima del camino.
  - Actualizar las capacidades residuales del camino en ida y regreso.
- Calcular el flujo máximo total en el recorrido.

Función EncontrarCaminoAumentante:

- Utilizar BFS o DFS para encontrar un camino desde el origen hasta el destino.

Si se encuentra un camino:

  - Devolver el camino como una lista de caminos accesibles.

Si no se encuentra un camino:

  - Devolver “No es una opción viable”.

Función PrincipalDeLogistica:

Crear un grafo representando la red logística con nodos y aristas. (transporte y caminos)

- Asignar capacidades a las aristas y nodos según la capacidad de transporte y almacenamiento.
- Origen = Nodo de origen.
- Destino = Nodo de destino.
- Llamar a la función FordFulkerson (grafo, origen, destino).

3.3 Prueba de escritorio del algoritmo de planificación

A continuación, se describe una prueba de escritorio del funcionamiento del algoritmo de planificación de rutas

1.- Definimos nuestro punto de origen (S) y nuestro destino (J), así como los valores que tendrán las aristas (que serán nuestros caminos) para determinar el máximo de flujo de la ruta. (ver Figura 3.1)

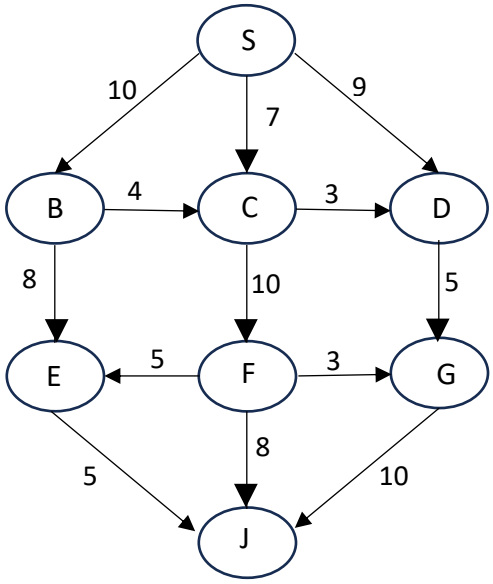
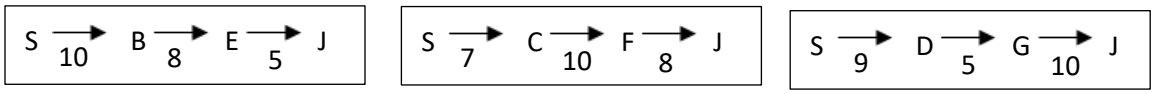


Figura 3.1 Grafo Inicial

2.- Iniciamos el recorrido e identificamos 3 caminos por los cuales podemos empezar que serán los siguientes, cada uno empezamos a hacer el aumento de flujo a lo largo del camino y se ajustan los caminos residuales.



3.-El grafo modificado con las primeras 3 iteraciones que realizo, se repite el proceso de encontrar caminos aumentables, ajustar los flujos y capacidades residuales. (ver Figura 3.2)

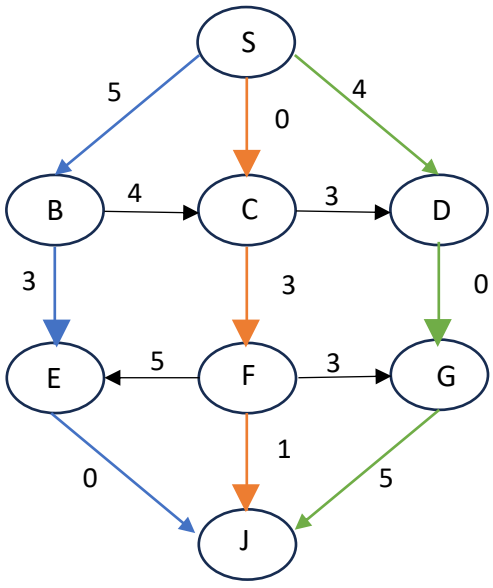


Figura 3.2 Grafo con iteraciones

4.- Después de N iteraciones sobre el grafo, damos con nuestro flujo máximo el cual se determina cuando no se puede encontrar más caminos aumentantes. (ver Figura 3.3) y el camino sería el siguiente (ver Figura 3.4)

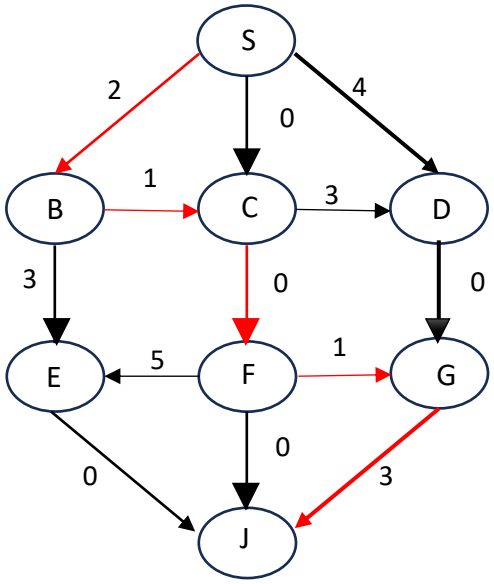


Figura 3.3 Grafo final

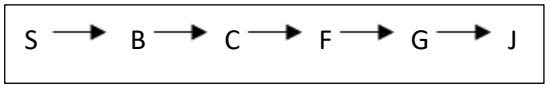


Figura 3.4 Ultima iteración en búsqueda del flujo máximo

3.4 Análisis del Algoritmo propuesto

El algoritmo de Ford-Fulkerson es un método popular para resolver problemas de flujo máximo en redes, que es central en el campo de la investigación operativa y las ciencias de la computación, especialmente en contextos como la planificación de rutas de logística, que es el enfoque de este proyecto.

El desarrollo de sistemas de planificación de rutas de logística que utilizan grafos y algoritmos como Ford-Fulkerson está intrínsecamente vinculado a la ingeniería de software. La ingeniería de software proporciona las metodologías, herramientas y prácticas necesarias

para diseñar, implementar, probar y mantener sistemas de software complejos de manera eficiente y efectiva.

Mejor Caso:

Para el algoritmo de Ford-Fulkerson ocurre cuando el primer camino de aumento encontrado proporciona el flujo máximo posible de la fuente al sumidero. Esto puede suceder en grafos con estructuras simples o cuando las capacidades de las aristas están alineadas de tal manera que el primer camino aumentante encontrado ya maximiza el flujo a través de la red. En este escenario, la complejidad temporal del algoritmo puede ser tan baja como $O(E)$, donde E es el número de aristas, ya que solo necesita encontrar un camino y actualizar las capacidades.

Peor Caso:

Para el algoritmo de Ford-Fulkerson se da en situaciones donde el grafo permite muchos caminos aumentantes con capacidades muy pequeñas. En este caso, el algoritmo puede terminar iterando un número excesivamente grande de veces, especialmente si se elige un camino aumentante no óptimo en cada iteración. La complejidad temporal en el peor de los casos puede ser $O(m * E)$, donde m es el flujo máximo encontrado y E es el número de aristas, porque en el peor de los casos, el algoritmo podría aumentar el flujo en solo 1 unidad en cada iteración.

Capítulo 4. Sistema de planificación de rutas de logística usando grafos

En esta sección se presenta el desarrollo de la planificación del sistema web propuesto con el algoritmo ya presentado para tener una planificación adecuada sobre rutas de logística.

4.1 Herramientas y Lenguaje de desarrollo

Para la implementación del algoritmo se utilizó un servidor Web, el lenguaje de programación PHP, HTML, CSS3 y la librería Vis.js que permite visualizar los grafos de forma gráfica [4, 5, 12, 13].

Servidor Web

Un servidor web es un software o hardware que proporciona servicios de hosting y entrega de contenido en la World Wide Web (WWW) a través del protocolo HTTP (Hypertext Transfer Protocol). Los servidores web son esenciales para la distribución de sitios web y aplicaciones web a los usuarios finales que acceden a través de sus navegadores web. En nuestro trabajo se utilizó el servidor Apache HTTP Server que es uno de los servidores web más utilizados en el mundo. Es de código abierto y se ejecuta en una variedad de sistemas operativos, incluyendo Linux, Windows y macOS. Para este propósito se descargo (Figura 4.1) e instalo WAMPSERVER

Figura 4.1. https://www.wampserver.com/en/

Los servidores web pueden servir diversos tipos de contenido web, como páginas HTML estáticas, imágenes, archivos de estilo, scripts de servidor (como PHP, Python o Ruby), y más. También pueden gestionar la autenticación, la seguridad y otras funciones relacionadas con el servidor. Un servidor web es una parte esencial de la infraestructura de internet, ya que permite que los sitios web y las aplicaciones web sean accesibles para los usuarios en todo el mundo.

Lenguaje PHP

PHP (Hypertext Preprocessor) es un lenguaje de programación ampliamente utilizado en el desarrollo web. PHP desempeña un papel fundamental en el desarrollo web, permitiendo la creación de aplicaciones web dinámicas y accesibles. Su facilidad de aprendizaje, amplia adopción y eficiencia en el procesamiento web lo convierten en una herramienta importante para desarrolladores web de todo el mundo (acceso a las características en la figura 4.2). Al instalar Wampserver se instaló de forma automática el lenguaje PHP para ser utilizado en el desarrollo del sistema.

Figura 4.2. https://itsoftware.com.co/content/lenguaje-de-programacion-php/

Su importancia radica en varias áreas:

1. Desarrollo web dinámico: PHP es especialmente adecuado para la creación de sitios web dinámicos, donde el contenido se genera en tiempo real en función de las solicitudes del usuario. Puede conectarse a bases de datos, procesar formularios, y realizar diversas tareas que permiten la interacción del usuario con la aplicación.

2. Amplia adopción: PHP ha sido ampliamente adoptado en la comunidad de desarrollo web. Muchos sitios web populares, como Facebook y WordPress, utilizan PHP en su infraestructura. Esto significa que hay una gran cantidad de recursos disponibles en línea, como bibliotecas, tutoriales y comunidades de desarrolladores.

3. Facilidad de aprendizaje: PHP es conocido por ser relativamente fácil de aprender para aquellos que están comenzando en el desarrollo web. Su sintaxis es similar a otros lenguajes como C y JavaScript, lo que facilita la transición para muchos programadores.

4. Amplia compatibilidad: PHP se ejecuta en una amplia variedad de plataformas y sistemas operativos, incluyendo Linux, Windows, macOS y más. También es compatible con numerosas bases de datos, como MySQL y PostgreSQL.

5. Eficiencia en el procesamiento web: PHP fue diseñado específicamente para el procesamiento web, por lo que tiende a ser eficiente en esta tarea. Puede manejar grandes cantidades de solicitudes web sin un alto consumo de recursos del servidor.

HTML y CSS

HTML (HyperText Markup Language) y CSS (Cascading Style Sheets) son dos lenguajes fundamentales en el desarrollo web que trabajan juntos para crear y dar formato a páginas web. Cada uno desempeña un papel específico en la construcción de sitios web interactivos y visualmente atractivos. Por default al utilizar desarrollo web se puede utilizar HMTL y CSS a través de los navegadores WEB.

Vis.js

Vis.js es una biblioteca JavaScript de código abierto que se utiliza para crear visualizaciones interactivas en la web. Vis.js es ampliamente utilizado en aplicaciones web que requieren visualizaciones de datos, como aplicaciones de análisis, herramientas de representación gráfica de redes sociales, sistemas de gestión de proyectos y mucho más (figura 4.3).

Figura 4.3. https://visjs.org/

Algunas de las características principales son:

1. Visualización de datos: Vis.js permite representar datos complejos y relaciones de una manera visualmente efectiva. Puedes utilizarlo para crear gráficos, diagramas, redes, líneas de tiempo y otras visualizaciones.

2. Interactividad: Las visualizaciones creadas con Vis.js son interactivas, lo que significa que los usuarios pueden interactuar con los elementos visuales, hacer clic en nodos, arrastrar elementos, acercar y alejar, y más, según el tipo de visualización.

3. Personalización: La biblioteca ofrece una amplia gama de opciones de personalización, lo que te permite controlar la apariencia y el comportamiento de tus visualizaciones. Puedes establecer colores, tamaños, etiquetas y otros atributos de los elementos visuales.

4. Facilidad de uso: Vis.js proporciona una API sencilla de utilizar que facilita la creación de visualizaciones personalizadas. También incluye una documentación detallada y ejemplos que ayudan a los desarrolladores a comenzar rápidamente.

5. Licencia de código abierto: Vis.js se distribuye bajo una licencia de código abierto (MIT License), lo que significa que es gratuito para su uso en proyectos personales y comerciales.

En especial se trabajó con la librería de vis.js en su sección de redes (ver figura 4.4), esta parte de la librería permite generar todo tipo de grafos dinámicos con colores, tamaños y efectos que hacen más agradable la representación del problema modelado.

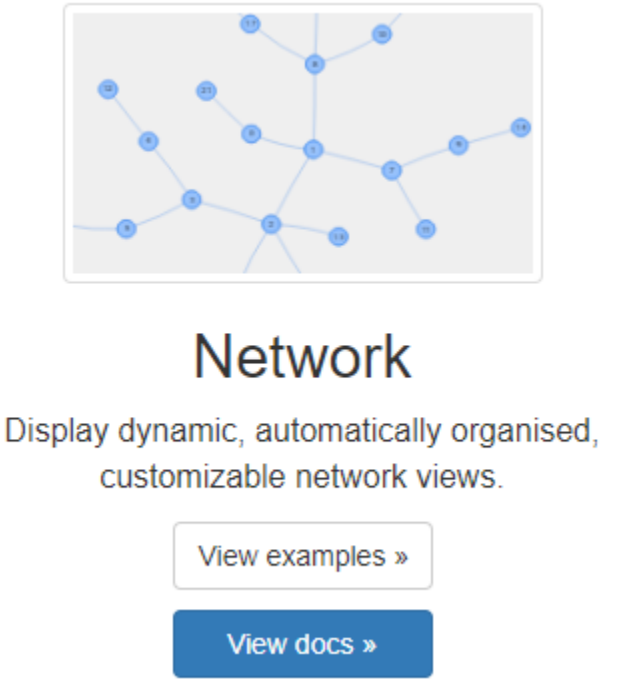


Figura 4.3. https://almende.github.io/vis/network_examples.html

4.2 Formato de los archivos de entrada

Para facilitar el trabajo de las pruebas del algoritmo por parte de los usuarios se estableció el formato del grafo de entrada en un archivo de texto.

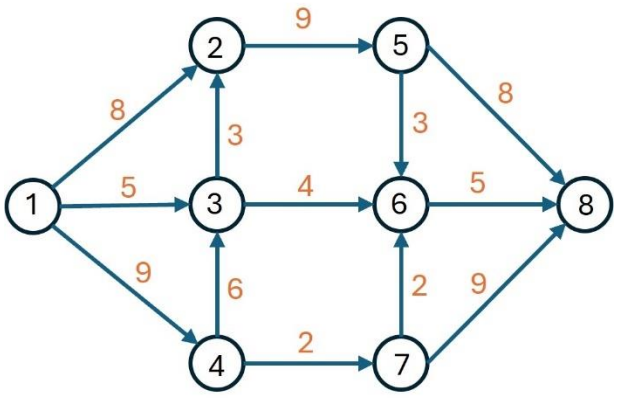


Figura 4.4. Grafo dirigido, ejemplo de entrada

Por ejemplo, si un usuario desea obtener el flujo máximo para el grafo dirigido de la figura 4.4 se tendrá el siguiente archivo de texto (grafoEntrada.txt):

```
-----------------------------------------------

8       ← número total de nodos en la red (grafo)
1 2 8  ← conexión del nodo inicial 1 con el nodo final 2, con un costo=8
1 3 5
1 4 9
2 5 9
3 2 3
3 6 4
4 3 6
4 7 2
5 6 3
5 8 8
6 8 5
7 6 2
7 8 9  ← conexión del nodo inicial 7 con el nodo final 8, con un costo=9
-----------------------------------------------
```

En la primera línea hay un 8 que representa el número de nodos del grafo y a continuación en cada línea se ponen las conexiones (aristas) del grafo dirigido y el tercer dato de cada línea representa el peso, costo o valor que hay entre el nodo inicial y el nodo final. Note que en cada dato se deja solo un espacio entre los diferentes valores.

Posteriormente al leer el archivo de grafoEntrada.txt el sistema lo trasforma a una matriz de adyacencia (tabla 4.1) para poder realizar las operaciones del algoritmo de flujo máximo.

Tabla 4.1. Matriz de adyacencias del grafo de ejemplo

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	0	8	5	9	0	0	0	0
2	0	0	0	0	9	0	0	0
3	0	3	0	0	0	4	0	0
4	0	0	6	0	0	0	2	0
5	0	0	0	0	0	3	0	8
6	0	0	0	0	0	0	0	5
7	0	0	0	0	0	2	0	9
8	0	0	0	0	0	0	0	0

4.3 Pantallas y pruebas del sistema

A continuación, se describe el sistema web desarrollado para el algoritmo de flujo máximo. Al iniciar WampServer se inician también los servicios del servidor web (servidor Apache), el gestor de bases de datos de MySQL (no utilizado aquí) y el lenguaje de programación PHP [10]. Para las pruebas se utilizó un servidor local denominado "localhost" y el sistema se encuentra alojado en la carpeta MarcoPolo2, de esta forma se abre un navegador web y en la URL se especifica: http://localhost/MarcoPolo2/ y se muestra la pantalla inicial (index.php).

En la figura 4.5 se muestra la pantalla principal del sistema, en la parte superior se indica las funciones principales del sistema:

- Inicio. Pantalla inicial del sistema para determinar el flujo máximo
- Teoría. Descripción de la teoría asociada al Algoritmo de Ford-Fulkerson
- Flujo máximo. Opción que permite realizar el cálculo del flujo máximo en grafos
- Autores. Nombres de los participantes en este trabajo

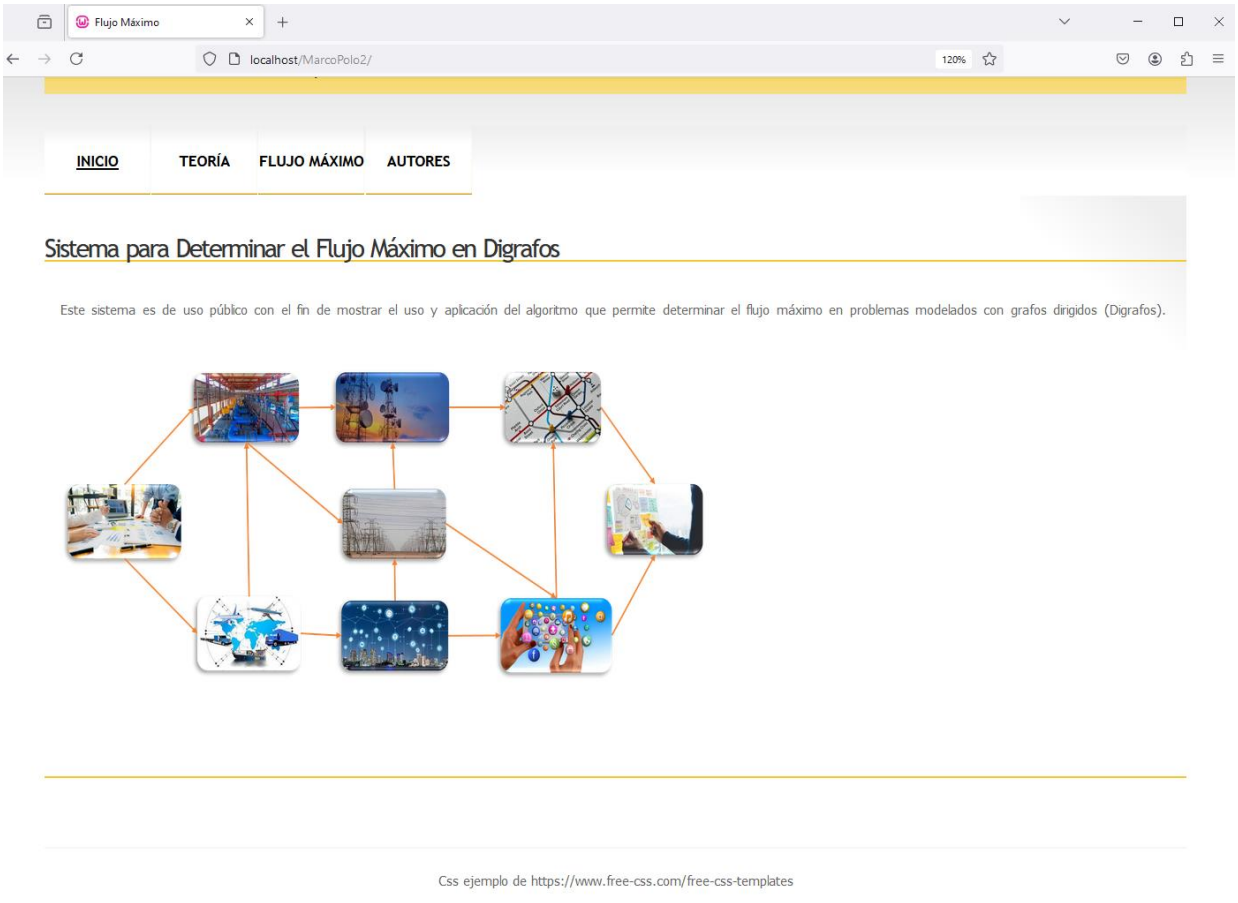


Figura 4.5. Pantalla inicial del sistema

En la figura 4.6 se describe la opción de “Teoría” en el sistema de flujo máximo, donde se explica el funcionamiento del Algoritmo Ford-Fulkerson que nos permite determinar cual es el flujo máximo que se puede distribuir de un nodo inicial a un nodo final modelado en un grafo dirigido.

En el dígrafo los vértices indican los nodos de la red mientras que las aristas representan el flujo que se puede enviar entre dos nodos de red, la dirección del grafo indica que solo es posible enviar flujo de un nodo a otro, pero no en ambos sentidos.

El Algoritmo de Ford-Fulkerson aunque es un algoritmo poderoso, es esencial destacar que su implementación original puede no converger en todos los casos debido a que el grafo es dirigido.

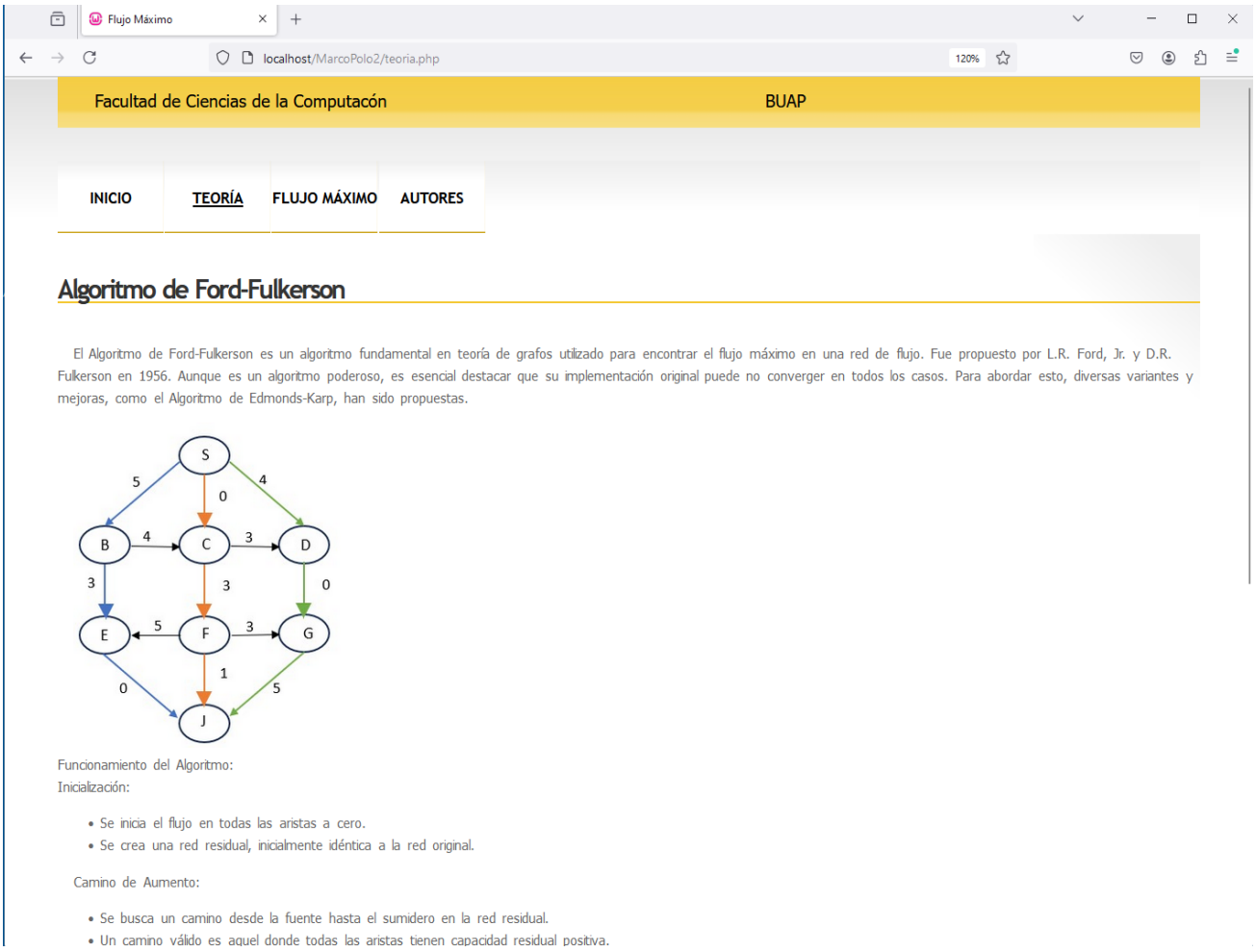


Figura 4.5. Pantalla de la Teoría

En la figura 4.6 se muestra la opción de “FLUJO MÁXIMO” que permite la aplicación del algoritmo de Ford-Fulkerson, para iniciar es necesario contar con un archivo de texto con la especificación del grafo como se indico en la sección 4.2.

En esta pantalla se indica en la parte derecha como esta formado el archivo de entrada y para iniciar el cálculo es necesario dar clic en el botón de “examinar”, donde se apertura un cuadro de dialogo para buscar o seleccionar el archivo de entrada con los datos del grafo. Al elegir el archivo de entrada aparecerá el nombre al lado derecho del botón por lo que ahora es necesario oprimir el botón de “Enviar” para cargar el grafo correspondiente.

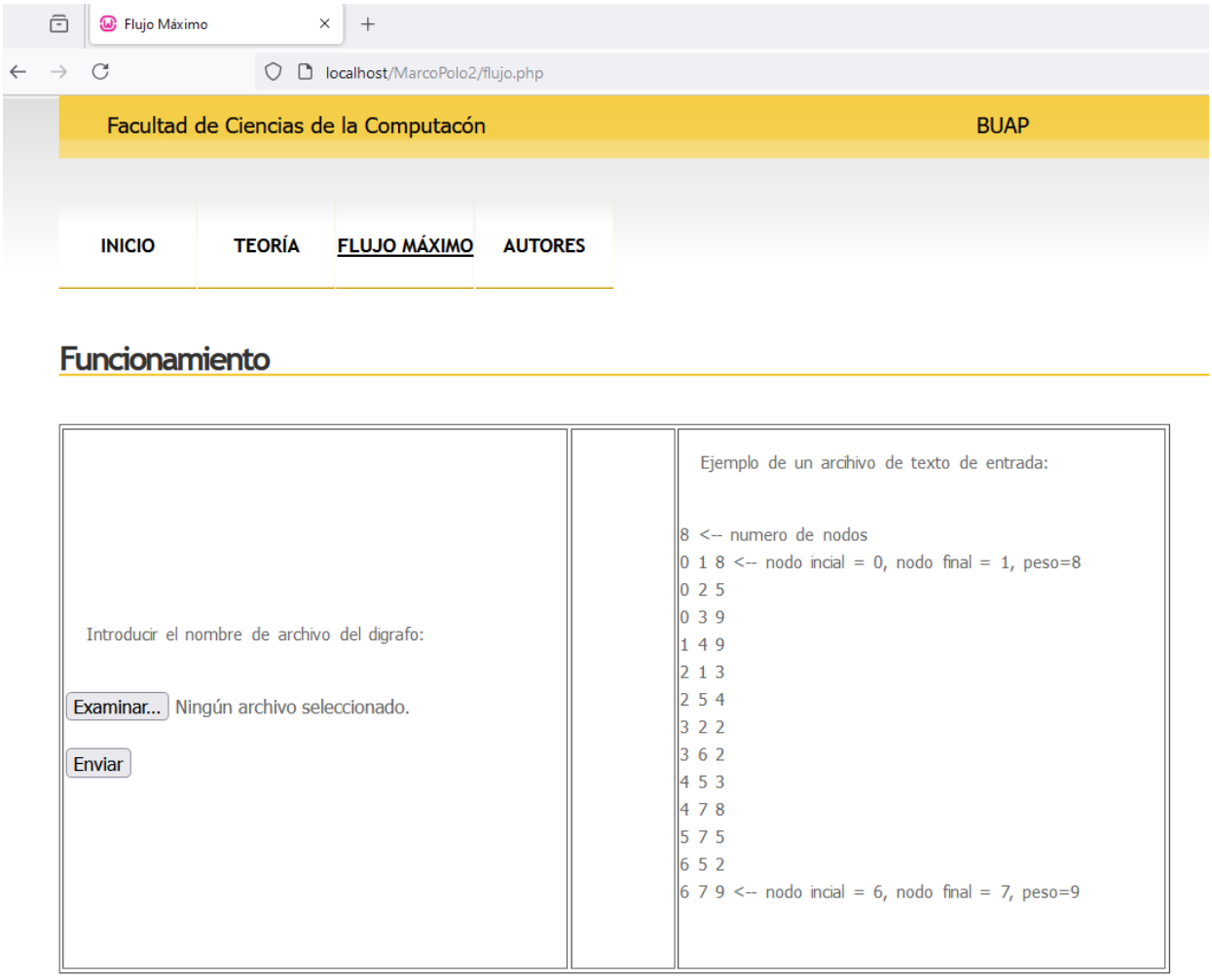


Figura 4.6. Pantalla para la ejecución del algoritmo de Ford-Fulkerson e iniciar con el cálculo del flujo máximo dando como entrada un archivo de texto.

Una vez seleccionado el archivo de entrada al darle “Enviar”, en la Figura 4.7 se muestra la pantalla donde se indica el grafo con los datos del grafo (dígrafo). Podrá notar que se indica el nombre del archivo de entrada y el numero total de nodos del grafo. En la parte derecha de la pantalla se indica que es necesario seleccionar el nodo inicial y el nodo final para poder realizar el cálculo del flujo máximo utilizando el algoritmo de Ford-Fulkerson, generalmente el nodo inicial es menor al nodo final y también depende del flujo a calcular. También es importante hacer notar que en ocasiones no se podrá obtener el flujo máximo sino existe un camino del nodo inicial al nodo final, por ejemplo, para este ejemplo si el nodo inicial=5 y el nodo final es igual a 7 entonces no existe un camino para enviar un flujo. Note además que la figura del grafo se puede mover, alejar, acercar y rotar.

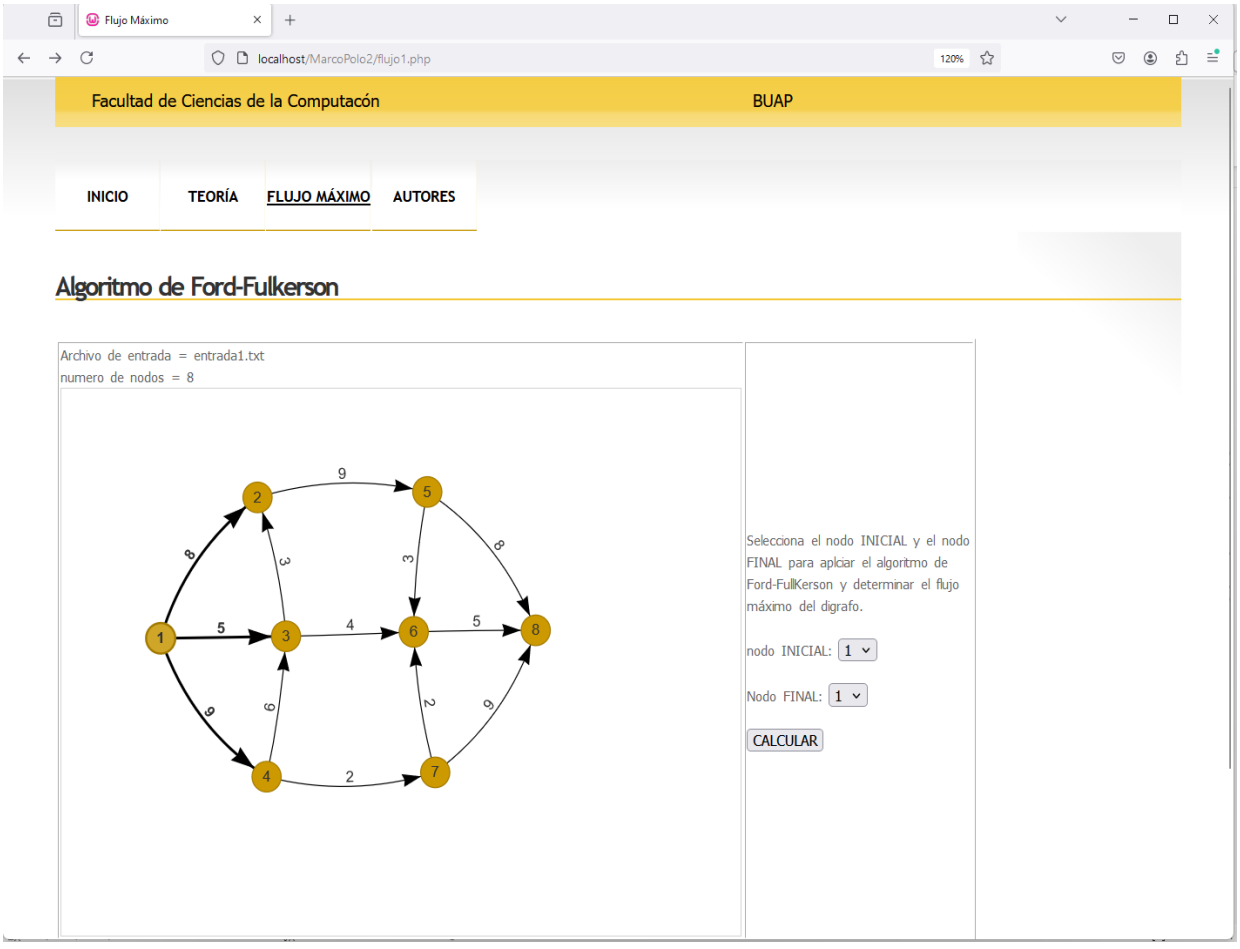


Figura 4.6. Pantalla para la ejecución

Si seleccionamos como nodo inicial=1 y nodo final = 8 en el grafo mostrado en la pantalla anterior (figura 4.7) al dar clic en el botón “CALCULAR” obtenemos el cálculo del flujo máximo del nodo 1 al nodo 8 = 15 como se muestra en la figura 4.7, en esta pantalla se

muestra además el grafo resultante donde el cero en las aristas indican que se utilizó el canal completamente para enviar el flujo mientras que en otros casos se reduce el canal de flujo.

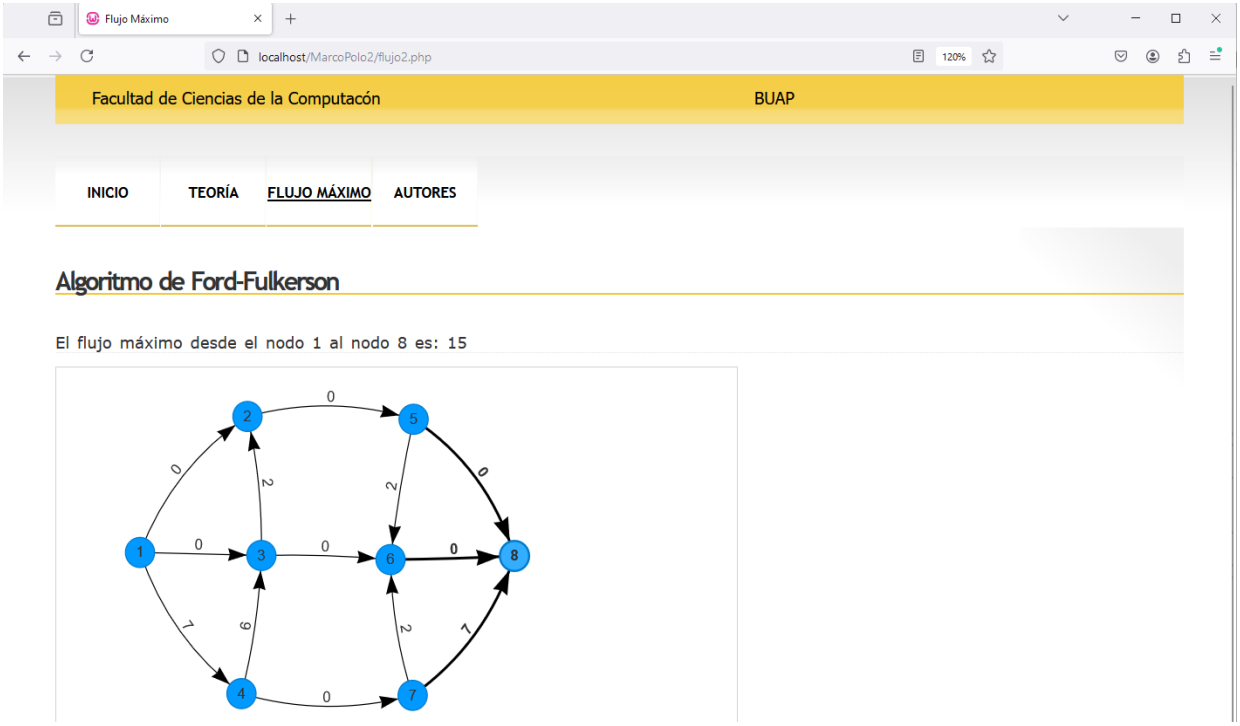


Figura 4.7. Cálculo del flujo máximo

Por ejemplo, si el nodo inicial=3 y el nodo final =8 entonces significa que los canales de flujo 1-->2, 1-->3 y 1-->4 no se utilizan y obtenemos como resultado lo mostrado en la figura 4.8.

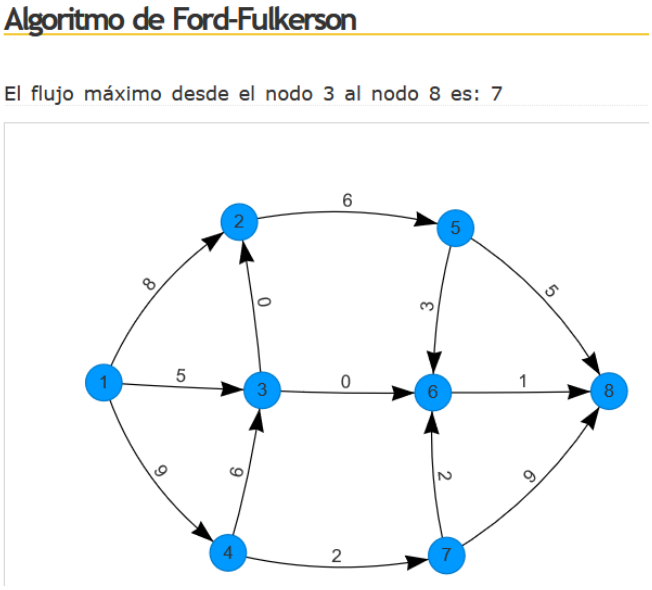


Figura 4.7. Cálculo del flujo máximo

Conclusiones y perspectivas

El trabajo desarrollado está enmarcado en la teoría de grafos y su aplicación en diferentes áreas de la ciencia, los grafos y su teoría asociada es una herramienta versátil y ampliamente aplicable que desempeña un papel fundamental en el modelado y la resolución de problemas en una amplia gama de campos, desde la computación y las matemáticas hasta la biología, la logística, la ingeniería y las ciencias sociales. En este trabajo podemos resaltar los siguientes resultados:

- Un prototipo funcional en web de fácil funcionalidad para los usuarios finales, el cual permitirá realizar una prueba para para determinar la mejor ruta que permita enviar productos de un origen a un destino aplicando una estrategia de flujo máximo en redes.
- Los usuarios podrán genera un repositorio con diferentes ejemplos como estancias de pruebas para determinar la mejor ruta en problemas de logística.
- Al generarse una aplicación web facilita el uso a nivel global de personas interesadas en determinar por ejemplo la mejor ruta para el trasporte de mercancías o problemas similares de logísticas que se puedan modelar con grafos.
- Este proyecto representa una investigación aplicada donde se integran los conocimientos y habilidades de la teoría de grafos y de los algoritmos de flujo máximo para determinar la mejor ruta en el trasporte de procesos de logística.

Perspectivas

- Algunas perspectivas de desarrollo de este sistema consisten en programar más algoritmos de flujo máximo y poder hacer comparaciones en resultados, en tiempos y en general para determinar de forma practica que algoritmo da más y mejores resultados
- También es importante agregar almacenamiento de las pruebas realizadas en alguna base de datos y que el sistema permita el manejo de usuarios

Bibliografía

[1] Beriezina, L. YU. (2013). Grafos y sus aplicaciones: una introducción a la teoría de grafos, editorial URSS.

[2] Braude (2003), Ingeniería de Software, una perspectiva orientada a objetos, Alfaomega, México.

[3] Dorzdek, Adan (2007). Estructuras de datos y algoritmos con Java, International Thomson Editores.

[4] Freeman, E. & Freeman, E. (2006). Head First HTML with CSS & XHTML.

[5] Gutiérrez, A. y Bravo, G. (2005). PHP5, Alfaomega Ra-MA, México.

[6] Meza, O. y Ortega, M. (2006). Grafos y Algoritmos. Editorial Equinoccio, Universidad Simon Bolivar.

[7] López Porta Ángela (2019). El problema del flujo máximo: Teoría, algoritmos y aplicaciones. Universidad de Santiago de Compostela.

[8] Pressman, R. S. (2002). Ingeniería de Software. España: McGraw-Hill Interamericana de España S.L.

[9] Román Martínez y Elda Quiroga (2002). Estructuras de datos: referencia práctica con orientación a objetos, Alfaomega, México.

[10] Spona, H. (2010), Programación de bases de datos con MySQL y PHP, Alfaomega, México.

[11] Sommerville, I (2005). Ingeniería del software, Séptima edición, Pearson Educación. S.A. Madrid.

[12] Ullman, L. (2009). PHP paso a paso. España: Anaya Multimedia.

[13] Welling, Luke y Thomson Laura. (2009). Desarrollo Web con PHP y MySQL Editorial Anaya.