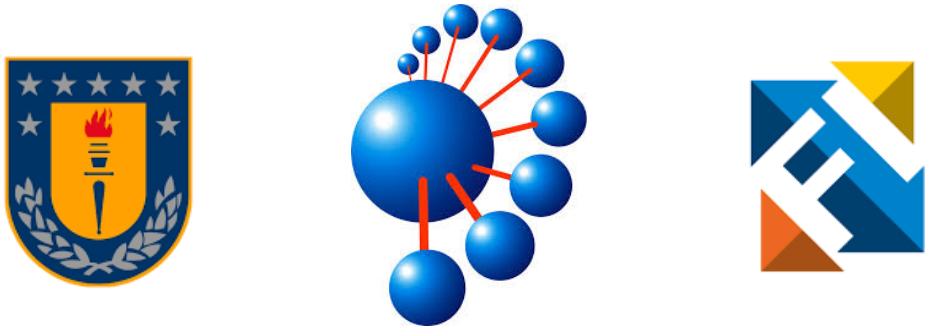
TAREA COMPUTACIONAL DEL CURSO DE MATEMÁTICAS DISCRETAS SEMESTRE 2025-2



Integrantes del Grupo:
Felipe Osvaldo Grandón Contreras
Leonardo Rafael Guerrero Ortega
Matías Alberto Concha Aguilera
Vicente Andrés Ramírez Torrealba

Curso: Matemáticas Discretas
Docentes a Cargo: Pierluigi Cerulo, Waldo Gálvez
Fecha: 23/10/2025

Universidad de Concepción
Facultad de Ingeniería
Departamento de Ingeniería Informática y Ciencias de la Computación

Índice

1. Introducción

En este informe, se aborda una tarea computacional fundamental en la rama de las matemáticas discretas, la cual tiene como propósito implementar y aplicar el algoritmo Dijkstra para la búsqueda del camino más corto en diversos tipos de grafos conexos. El principal objetivo de esta tarea se centra en desarrollar un programa en C que sea capaz de determinar la secuencia más óptima de vértices, es decir el camino más corto entre un vértice inicial y un vértice final del grafo.

El problema que vamos a solucionar se enmarca en la teoría de grafos y consiste en encontrar el camino con mejor costo entre dos nodos dados de un grafo con aristas de peso no negativo, en el cual, para simplicidad del problema consideraremos aristas con peso igual a 1. Esta tarea se aplica a tres estructuras de grafos no dirigidos: un árbol, un grafo planar y un grafo Euleriano. Además, la implementación debe considerar tanto los grafos no dirigidos originales de la tarea como en 4 de sus orientaciones arbitrarias de cada uno. Esto introduce el desafío de manejar grafos dirigidos y considerar la importancia de la conexidad fuerte.

La solución que proponemos se basa en el diseño e implementación del algoritmo Dijkstra, capaz de resolver el problema del camino más corto en un conjunto de vértices de un grafo conexo.

- Modelamiento del grafo: Para representar el grafo en el código, se utilizará la matriz de adyacencia de este, lo que permitirá almacenar eficientemente los vértices que están conectados. La matriz de adyacencia facilitará la exploración de vecinos, esencial para el algoritmo de Dijkstra.

- Implementación del Algoritmo Dijkstra: El algoritmo se implementará utilizando una cola de prioridad para seleccionar de manera eficiente el vértice no visitado con la menor distancia conocida desde la fuente. Este enfoque garantiza que se explore primero el camino más prometedor en cada iteración.

- Manejo de orientaciones: El programa debe ser capaz de procesar grafos dirigidos, donde la orientación de las aristas se maneja a partir de la matriz adyacencia (es decir, una arista (u, v) no implica la existencia de una arista (v, u)). Esto permitirá investigar las diferencias en la existencia y longitud del camino más corto entre los grafos no dirigidos y sus orientaciones, especialmente ante la posible falta de conexidad fuerte.

La estructura de este informe está organizada en secciones donde se encuentra el trabajo realizado. Tras esta introducción, la Sección 2 presenta los Objetivos de la tarea. En la Sección 3 se describe el modelo implementado donde el punto 3.1 es la Representación del grafo donde se definen los vértices y las aristas, y el punto 3.2 es la Implementación del Algoritmo Dijkstra para encontrar el camino más corto. En la Sección 4 se encuentran los Resultados obtenidos de las ejecuciones del programa, donde se discuten las diferencias entre los grafos dirigidos y no dirigidos y la importancia de una conexidad fuerte. Finalmente la Sección 5 contiene las conclusiones de la tarea, en donde se evalúa el cumplimiento de los objetivos propuestos y donde se reportan las dificultades e impresiones encontradas de cada uno de los integrantes del grupo de trabajo.

2. Objetivos

El objetivo principal de esta tarea se centra principalmente en el modelamiento de algoritmos y representación de grafos según su definición. Más que la ejecución de código, nuestro propósito es medir la capacidad de programar el algoritmo Dijkstra y la importancia que tiene este en la Teoría de Grafos. Entender como el algoritmo puede implementarse desde grafos simples a complejos, y sobre todo, como puede aplicarse a problemas de la vida real.

Identificar las diferencias que hay al aplicar el algoritmo Dijkstra entre un grafo Árbol, un grafo Planar y un grafo Euleriano.

Analizar el impacto que tiene la Orientación y la Conexidad a la hora de aplicar el algoritmo. Comprender la diferencia de resultados del camino más corto al momento de operar sobre grafos no dirigidos frente a grafos dirigidos. A raíz de esto, se deriva la necesidad de analizar el impacto de la falta de conexidad fuerte en las orientaciones, determinando si un camino puede existir o no según ciertas direcciones dadas.

Por último, a nivel individual, esperamos mejorar nuestras habilidades en redacción e interpretación, como también nuestro nivel de análisis en problemas de alta complejidad, por ejemplo, a la hora interactuar con el algoritmo Dijkstra.

A nivel de grupo, esperamos mejorar la habilidad de comunicación en un trabajo técnico a través de la redacción de un informe de carácter formal, que es lo requerido en una asignatura de tal complejidad e importancia como lo es Matemáticas Discretas.

3. Modelo

3.1. Representación del Grafo

Para explicar como se representan los grafos de la tarea, primero definimos sus nodos o vértices a través de una estructura de datos tipo Struct que almacena toda la información necesaria para el algoritmo. Guarda el nombre de cada vértice (a, h, l, etc) y la función inicializarVertice() establece los parámetros de Dijkstra:

- distancia: La distancia se inicializa en infinito (99999) para todos los vértices excepto el inicial que es 0.
- visitado: Dato booleano inicializado en false
- IindicePadre: Inicializado en -1, que se usa para la reconstrucción del camino.

Luego los vértices se gestionan en el arreglo dinámico Vertice *vertices, cuyo tamaño se va ajustando con realloc durante la lectura.

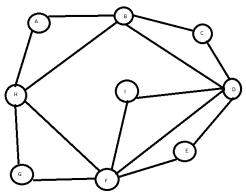
Luego, la relación entre los vértices se define por la matriz de adyacencia dinámica del Grafo (int **matriz):

- Representación: Un valor matriz[i][j] = 1 significa que existe una arista dirigida desde el vértice con índice i al vértice con índice j.
- Pesos de las aristas: En el enunciado de la tarea, se nos indica que los pesos de las aristas serán 1. Esto se implementa directamente en la relación: nuevaDist: vertices[u].distancia + 1.

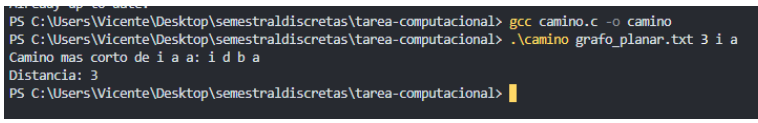
- Carga de Datos: La función leerArchivo carga los grafos no dirigidos (Árbol, Planar, Euleriano), asegurando que se cumpla que matriz[i][j] = 1 y matriz[j][i] = 1
- Orientaciones: La función aplicarOrientacion permite generar las 4 orientaciones requeridas (dirigida i–>j, j–>i, bidireccional i<–>j o aleatoria) a partir del grafo no dirigido seleccionado, manipulando la simetría de la matriz.

3.2. Implementación del Algoritmo de Dijkstra

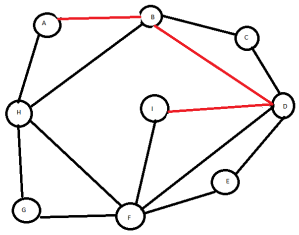
1. Iteración principal: Con un ciclo for que se ejecuta V - 1 veces, es decir la cantidad de vértices del grafo -1. En cada iteración se busca el vértice u no visitado (!vertices[i].visitado) con la menor distancia registrada (distancia_min).
2. Visita y Selección: Si se encuentra un vértice alcanzable (u != -1), se marca el vértice como visitado (vertices[u].visitado = true), para garantizar así que su distancia es definitiva.
3. Vecindad: Se recorren todos los vecinos posibles de v y de u. Solo se consideran los vecinos alcanzables, es decir matriz [u][v] == 1 y se consideran también los vertices no visitados (!vertices[v].visitado).
4. Actualización: Si la nueva distancia nuevaDist es menor que vertices[v].distancia, se actualiza la distancia de v y se establece el puntero al predecesor (vertices[v].IndicePadre = u).
5. Finalización del camino: Si el camino existe, se reconstruye la ruta recorriendo el IndicePadre desde el vértice final hasta el de origen, almacenando los índices en el arreglo camino y luego imprimiéndolos en orden inverso.



(a) Grafo planar



(b) Camino Grafo planar más corto



(c) Camino Grafo planar

Figura 1: Comparación de los grafos utilizados.

4. Resultados

En esta sección se presentan los resultados obtenidos de la ejecución del programa implementado en lenguaje C, el cual aplica el algoritmo de Dijkstra sobre tres tipos de grafos conexos: un árbol, un grafo planar y un grafo Euleriano. Para cada grafo se probaron las cuatro orientaciones definidas en la tarea y distintos pares de vértices de origen y destino. Las aristas poseen peso unitario, por lo que la distancia calculada corresponde al número mínimo de aristas que conectan ambos vértices.

4.1. Ejecución sobre el Árbol

El grafo tipo árbol se caracteriza por no contener ciclos, por lo que siempre existe un único camino entre dos vértices mientras la orientación lo permita. Por ejemplo, al ejecutar:

```
./camino grafo_arbol.txt 1 a h
```

la salida fue:

```
Camino más corto de a hacia h: a b d h
Distancia: 3
```

En la orientación opuesta (tipo 2), no se encontró camino desde a hacia h, lo que demuestra que la orientación de las aristas puede romper la conectividad fuerte. En la orientación bidireccional (tipo 3), el resultado se mantiene igual a la versión no dirigida.

4.2. Ejecución sobre el Grafo Planar

En el grafo planar, debido a la existencia de múltiples rutas, el algoritmo de Dijkstra seleccionó correctamente el camino más corto. Ejemplo:

```
./camino grafo_planar.txt 3 a i
```

Salida:

```
Camino más corto de a hacia i: a b d i
Distancia: 3
```

En la orientación aleatoria (tipo 4), los resultados variaron según las direcciones generadas, observándose en algunos casos la imposibilidad de alcanzar ciertos vértices. Esto refuerza la importancia de la conectividad fuerte en grafos dirigidos: incluso si el grafo original es conexo, su orientación puede dejar vértices inaccesibles.

4.3. Ejecución sobre el Grafo Euleriano

El grafo Euleriano permitió verificar que, aunque todas sus aristas participan en un ciclo cerrado, el algoritmo de Dijkstra identifica correctamente los caminos más cortos sin necesidad de recorrer todas las aristas. Por ejemplo:

```
./camino grafo_euleriano.txt 1 b i
```

Salida:

```
Camino más corto de b hacia i: b h i
Distancia: 2
```

En orientaciones inversas o aleatorias, el grafo mantuvo mejor conectividad que el planar y el árbol, lo que se debe a su mayor número de aristas y ciclos, reduciendo la probabilidad de desconexión.

4.4. Comparación general

En la Tabla 1 se resume el comportamiento de los tres grafos bajo las distintas orientaciones. Se observa que los grafos con más ciclos tienden a conservar la conectividad en mayor número de casos, mientras que los árboles son los más sensibles a la dirección de las aristas.

Grafo	**Tipo 1**	**Tipo 2**	**Tipo 3**	**Tipo 4 (aleatorio)**
Árbol	Conexo	No Conexo	Conexo	Variable
Planar	Conexo	Parcialmente conexo	Conexo	Variable
Euloriano	Conexo	Conexo	Conexo	Mayoritariamente conexo

Cuadro 1: Resumen de conectividad según orientación.

5. Conclusiones

En el desarrollo de esta tarea, se han cumplido exitosamente los objetivos de aprendizaje propuestos al principio de la tarea. Se implementó con éxito el Algoritmo de Dijkstra en C, validando su aplicación sobre diferentes topologías de grafos conexos (Árbol, Planar y Euleriano).

El análisis de las ejecuciones confirmó que la orientación de las aristas es el factor más crítico. Los resultados demostraron que el Árbol es la estructura más sensible frente a las orientaciones, el Grafo Euleriano demostró ser el más resiliente debido a la densidad de aristas y ciclos, manteniendo la conectividad en una mayor variedad de orientaciones. La detección de la falta de conexidad fuerte se realizó al verificar si la distancia al destino permanecía en infinito tras la ejecución.

A nivel técnico, la principal dificultad se centró en la representación del grafo. Inicialmente, se consideraron estructuras como listas enlazadas con punteros, pero se optó por la matriz de adyacencia por su simplicidad de implementación. Esta elección resultó afortunada, ya que facilitó enormemente el segundo gran desafío: la modificación de la orientación de los grafos dirigidos. La manipulación de las orientaciones se simplificó a modificar los valores 0 y 1 al recorrer la triangular superior de la matriz, evitando complejidades innecesarias en el manejo de punteros y estructuras enlazadas. La lógica del Algoritmo de Dijkstra se simplificó gracias a esta representación.

Respecto al trabajo en equipo, las dificultades iniciales para la organización y definición de tareas fueron superadas. El equipo se dividió de manera eficiente: dos integrantes se enfocaron en la lógica y la codificación (incluyendo el diseño del algoritmo y la optimización de la matriz) y dos en la redacción y análisis del informe. Esta coordinación final permitió cumplir con los plazos y alcanzar un resultado óptimo, evidenciando la importancia de la comunicación y la división del trabajo. El proceso reforzó la comprensión individual del Algoritmo de Dijkstra y la relevancia de la estructura del grafo en el resultado algorítmico.