



Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**



Grado en Matemáticas e Informática

Trabajo Fin de Grado

**Desarrollo de una Aplicación para
Ilustrar Contenidos de la Asignatura
Matemática Discreta II: Algoritmo de
Floyd-Warshall**

Autor: Marius Robert Drăghici
Tutor(a): Luis Magdalena Layos

Madrid, Junio 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

*Trabajo Fin de Grado
Grado en Matemáticas e Informática*

Título: Desarrollo de una Aplicación para Ilustrar Contenidos de la Asignatura Matemática Discreta II: Algoritmo de Floyd-Warshall

Junio 2024

Autor: Marius Robert Drăghici

Tutor: Luis Magdalena Layos

Departamento: Matemática Aplicada a las Tecnologías de la Información y las Comunicaciones

Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Grado pertenece al conjunto de trabajos relacionados con el *Proyecto Calculadora*, desarrollado como parte de la actividad del Grupo de Innovación Educativa "Desarrollo de Tecnologías en la Enseñanza de las Matemáticas". *Proyecto Calculadora* trata en construir una aplicación web, cuyo propósito es incorporar los distintos algoritmos que se imparten a lo largo del grado en Matemáticas e Informática, impartido en la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Madrid durante el desarrollo del trabajo. El objetivo es que la aplicación pueda ser usada en las aulas con el fin de ayudar a los estudiantes a entender los algoritmos de una forma más intuitiva, facilitando así el proceso de aprendizaje.

Concretamente, en el trabajo se ha tratado de incorporar un nuevo algoritmo a la aplicación: el algoritmo de Floyd-Warshall, empleado para el cálculo de caminos mínimos en grafos e impartido en la asignatura Matemática Discreta II.

En esta memoria se explicarán unos conceptos previos relacionados con la teoría de grafos. A continuación, se explicará la teoría del algoritmo, seguida de las tecnologías empleadas en la realización del trabajo, del código desarrollado, de la incorporación de dicho código en la aplicación, y finalmente se mostrarán los resultados y se hará un análisis del impacto del trabajo.

Abstract

This Bachelor's Thesis is part of the set of works related to *Proyecto Calculadora*, developed as part of the activity of the Educational Innovation Group "Desarrollo de Tecnologías en la Enseñanza de las Matemáticas". *Proyecto Calculadora* aims to build a web application, whose purpose is to incorporate the various algorithms taught throughout the Mathematics and Computer Science degree, taught at the Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Madrid at the time of writing this thesis. The purpose is for the application to be used in classrooms to help students understand the algorithms in a more intuitive way, thus facilitating the learning process.

Specifically, this work has been dedicated to incorporating a new algorithm into the application: the Floyd-Warshall algorithm, used for calculating shortest paths in graphs and taught in the Discrete Mathematics II course.

In this report, some preliminary concepts related to graph theory will be explained. Next, the theory of the algorithm will be explained, followed by the technologies used in process of the work, the developed code, the incorporation of this code into the application, and finally, the results will be presented, and an analysis of the impact of this work will be conducted.

Tabla de contenidos

1. Introducción	1
1.1. Contexto	1
1.2. Objetivos	1
1.3. Planificación	2
1.4. Diagrama de Gantt	2
1.5. Estructura de la memoria	2
2. Marco teórico	5
3. Algoritmo de Floyd-Warshall	9
3.1. Funcionamiento	9
3.2. Matriz de sucesores	11
3.3. Construcción de caminos	12
3.4. Ejemplo	13
3.4.1. Algoritmo de Floyd-Warshall	13
3.4.2. Construcción del camino mínimo	14
4. Tecnologías empleadas	17
5. Desarrollo del código	21
6. Incorporación del código en la aplicación	31
7. Resultados y conclusiones	39
7.1. Ejemplo 1	39
7.1.1. Algoritmo	39
7.1.2. Camino 1	44
7.1.3. Camino 2	46
7.1.4. Camino 3	47
7.2. Ejemplo 2: Grafo no conexo	48
7.2.1. Algoritmo	48
7.2.2. Camino entre distintas componentes conexas	51
7.3. Varios ejemplos de entradas inválidas	52
7.3.1. Ejemplo 1	52
7.3.2. Ejemplo 2	53
7.3.3. Ejemplo 3	54
7.3.4. Ejemplo 4	55

TABLA DE CONTENIDOS

7.3.5. Ejemplo 5	56
7.4. Conclusión	57
8. Análisis de impacto	59
Bibliografía	61
Anexos	65
A. Informe de originalidad	65

Capítulo 1

Introducción

1.1. Contexto

El presente trabajo trata sobre el desarrollo de una aplicación didáctica que muestre el funcionamiento del algoritmo de Floyd-Warshall, el cual es empleado para el cálculo de caminos mínimos en grafos e impartido en la asignatura Matemática Discreta II.

La matemática discreta es una rama de las Matemáticas que se encarga de estudiar estructuras discretas como los grafos, excluyendo de su estudio conceptos continuos, como los números reales o las funciones reales.

1.2. Objetivos

El principal objetivo del trabajo es desarrollar el algoritmo de Floyd-Warshall, y añadirlo al *Proyecto Calculadora*.

A continuación, se presentan los subobjetivos que se deberán cumplir para alcanzar este objetivo principal:

- Implementar el algoritmo de Floyd-Warshall, el cual reciba como entrada un grafo representado mediante una matriz de pesos, y determine las distancias mínimas entre cada par de vértices. En caso de que el grafo contenga bucles de peso negativo, notificar al usuario de su existencia.
- Extender el algoritmo para incorporar funcionalidades que permitan al usuario la obtención del camino mínimo, y no tan solo la distancia mínima.
- Desarrollar un sistema de ayuda que muestre cada paso del algoritmo hasta llegar a la solución final.
- Incorporar el código desarrollado al *Proyecto Calculadora*.

Capítulo 1. Introducción

1.3. Planificación

A continuación, se muestran las tareas necesarias para la elaboración del trabajo:

1. Estudio de los fundamentos teóricos y documentación.
2. Codificación del algoritmo de Floyd-Warshall.
3. Codificación del sistema de ayuda.
4. Elaboración de la memoria de seguimiento.
5. Incorporación del código en la aplicación.
6. Realización de pruebas.
7. Elaboración de la memoria final.

1.4. Diagrama de Gantt

En el siguiente diagrama de Gantt se muestra el comienzo y la duración de cada una de las tareas mencionadas en el apartado anterior:

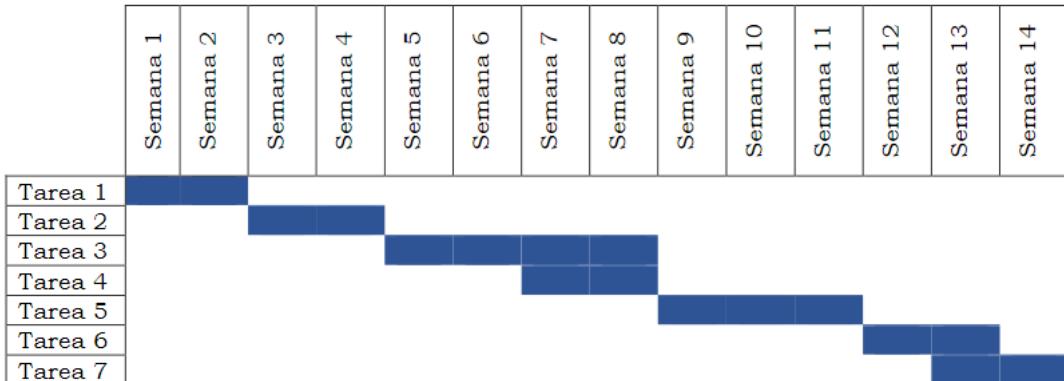


Figura 1.1: Diagrama de Gantt

1.5. Estructura de la memoria

Capítulo 1. Introducción: Se realiza una introducción a los objetivos del proyecto y el plan de trabajo.

Capítulo 2. Marco teórico: Se explica la teoría previa necesaria para poder comprender el contenido del trabajo.

Capítulo 3. Algoritmo de Floyd-Warshall: Se explica el objeto de este trabajo: el algoritmo de Floyd-Warshall, así como una demostración de su validez.

1.5. Estructura de la memoria

Capítulo 4. Tecnologías empleadas: Se explica la elección de las tecnologías empleadas en la realización del trabajo.

Capítulo 5. Desarrollo del código: Se explica el código desarrollado paso a paso, tanto los algoritmos como el sistema de ayuda.

Capítulo 6. Incorporación del código en la aplicación: Se explica la incorporación del trabajo al *Proyecto Calculadora*.

Capítulo 7. Resultados y conclusiones: Se muestran los resultados y conclusiones del trabajo realizado.

Capítulo 8. Análisis de impacto: Se estudia el análisis de impacto del trabajo realizado.

Capítulo 2

Marco teórico

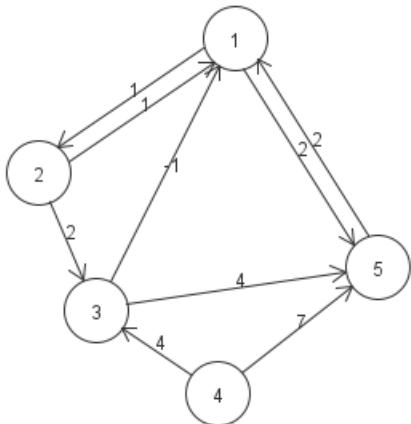
En este capítulo se explica el marco teórico previo necesario para poder comprender el algoritmo de Floyd-Warshall y el problema que resuelve.

- Un **grafo** es una estructura $G = (V, A)$, donde V es un conjunto finito no vacío de **vértices**, y A es un conjunto de pares de vértices de V que llamaremos **aristas**. Hay dos tipos principales de grafos:
 - Los grafos **no dirigidos** son aquellos en los que A es un conjunto de pares no ordenados, es decir, aquellos en los que las aristas no conectan los vértices en una dirección establecida. Gráficamente, las aristas se representan como líneas.
 - Los grafos **dirigidos** son aquellos en los que A es un conjunto de pares ordenados, es decir, aquellos en los que las aristas sí conectan los vértices en una dirección establecida. Gráficamente, las aristas se representan como flechas.
- Un **bucle** es una arista $a = (u, u), u \in V$.
- Las **aristas múltiples** son aquellas aristas que inciden en los mismos vértices en los grafos no dirigidos, o, aquellas aristas con mismo vértice de salida y llegada en los grafos dirigidos. Debido a haber definido A como un conjunto, un grafo no puede contener aristas múltiples.
- Un **grafo simple** es un grafo sin bucles ni aristas múltiples. En el desarrollo de este TFG solo se ha trabajado con este tipo de grafos.
- Un **grafo ponderado** es aquel en el cual las aristas tienen un número asociado, que llamaremos **peso** de la arista y denotaremos por $w(a), a \in A$.
- Una **matriz de pesos** de un grafo es una forma de representar el grafo. La matriz de pesos del grafo $G = (V, A)$ con $V = \{v_1, \dots, v_n\}$ y $A = \{a_1, \dots, a_k\}$ es $M(G) = (w(a_{ij})) \in \mathbb{M}_{n \times n}(\mathbb{R})$ con $i, j \in \{1, \dots, n\}$, donde $a_{ij} \in A$ es la arista que une el vértice i con el vértice j . Para la implementación del código, se comprueba que $a_{ii} = 0, \forall i \in \{1, \dots, n\}$ (ya que en caso contrario el grafo no sería simple), y se ha decidido que $w(a_{ij}) = \infty, \forall a_{ij} \notin A$, con $i, j \in \{1, \dots, n\}$.

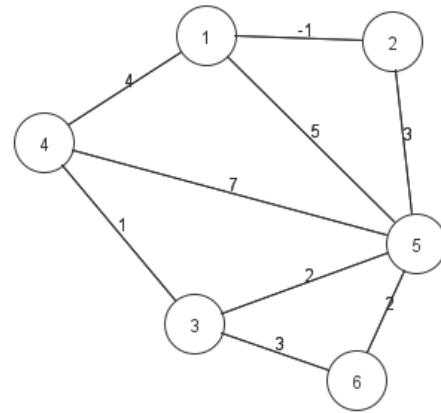
Capítulo 2. Marco teórico

- Un **camino** es una sucesión de vértices distintos $v_1v_2\dots v_{n+1}$ (a excepción quizás de v_1 y v_{n+1}) y aristas distintas $a_{i,i+1}, i \in \{1, \dots, n\}$. La longitud del camino es n . Denotaremos un camino que va del vértice u al vértice v por $C(u, v)$.
- Un **ciclo** es un camino cerrado, es decir, aquel en el que $v_1 = v_{n+1}$.
- El **peso** de un camino es la suma de los pesos de cada arista que lo componen: $w(C) = \sum_{a \in C \cap A} w(a)$. El algoritmo de Floyd-Warshall no permite la existencia de ciclos con peso negativo, ya que en ese caso no sería capaz de encontrar la distancia mínima entre cada par de vértices. Sin embargo, esto no supone un problema, puesto que el algoritmo es capaz de detectar la existencia de dichos ciclos.
- La **distancia** entre dos vértices es el peso mínimo de entre todos los caminos que los unen: $d(u, v) = \min\{w(C(u, v))\}$. Diremos que $C(u, v)$ es un camino mínimo de u a v si $w(C(u, v)) = d(u, v)$. Si no existe ningún camino entre dos vértices, la distancia será ∞ .

Ejemplo:



(a) Grafo dirigido



(b) Grafo no dirigido

La matriz de pesos del grafo dirigido es:

$$\begin{pmatrix} 0 & 1 & \infty & \infty & 2 \\ 1 & 0 & 2 & \infty & \infty \\ -1 & \infty & 0 & \infty & 4 \\ \infty & \infty & 4 & 0 & 7 \\ 2 & \infty & \infty & \infty & 0 \end{pmatrix}$$

La matriz de pesos del grafo no dirigido es:

$$\begin{pmatrix} 0 & -1 & \infty & 4 & 5 & \infty \\ -1 & 0 & \infty & \infty & 3 & \infty \\ \infty & \infty & 0 & 1 & 2 & 3 \\ 4 & \infty & 1 & 0 & 7 & \infty \\ 5 & 3 & 2 & 7 & 0 & 2 \\ \infty & \infty & 3 & \infty & 2 & 0 \end{pmatrix}$$

Capítulo 3

Algoritmo de Floyd-Warshall

3.1. Funcionamiento

El algoritmo de Floyd-Warshall es un algoritmo empleado para calcular la distancia entre cada par de vértices de un grafo dirigido ponderado (*All-pairs shortest paths*). El requisito de un grafo dirigido ponderado no supone un problema, ya que cualquier grafo no dirigido se puede considerar como un grafo dirigido en el cual las aristas son bidireccionales y con un peso igual a 1, a no ser que se les asigne un valor diferente.

Para probar la corrección del algoritmo, vamos antes a probar un lema que será de gran ayuda [1] .

Lema: Sea $C(u, v)$ un camino mínimo entre u y v , y sea $C(u', v') \subset C(u, v)$ un subcamino de u' a v' contenido en $C(u, v)$. Veamos que $C(u', v')$ es un camino mínimo entre u' y v' :

Demostración:

Sea $C(u, u') \subset C(u, v)$ un camino de u a u' , y sea $C(v', v) \subset C(u, v)$ un camino de v' a v . Se tiene que $C(u, v) = C(u, u') \cup C(u', v') \cup C(v', v) \implies w(C(u, v)) = w(C(u, u')) + w(C(u', v')) + w(C(v', v))$.

Sea $C'(u', v')$ un camino mínimo de u' a v' tal que $w(C'(u', v')) \leq w(C(u', v'))$. Se tiene que $C'(u, v) = C(u, u') \cup C'(u', v') \cup C(v', v)$ es un camino de u a $v \implies w(C'(u, v)) = w(C(u, u')) + w(C'(u', v')) + w(C(v', v)) \implies w(C'(u, v)) - w(C(u, v)) = w(C'(u', v')) - w(C(u', v'))$.

Como $w(C(u, v))$ es un camino mínimo entre u y v , se tiene que $w(C(u, v)) \leq w(C'(u, v)) \implies w(C(u', v')) \leq w(C'(u', v')) \implies w(C(u', v')) = w(C'(u', v'))$. Por lo tanto, $C(u', v')$ es un camino mínimo de u' a v' . \square

El algoritmo de Floyd-Warshall se basa en la siguiente observación [2] :

Observación: Sea $V = \{1, \dots, n\}$ los vértices de un grafo G . Consideramos un subconjunto $\{1, \dots, k\}, 1 \leq k \leq n$. Para cualquier par de vértices i, j , considero todos los caminos de i a j cuyos vértices intermedios pertenecen a $\{1, \dots, k\}$ y sea

Capítulo 3. Algoritmo de Floyd-Warshall

$C(i, j)$ el de menor peso. Existe una relación entre $C(i, j)$ y los caminos de menor peso de i a j cuyos vértices intermedios pertenecen a $\{1, \dots, k - 1\}$, en función de si k es un vértice intermedio de $C(i, j)$ o no:

- Si k no es un vértice intermedio de $C(i, j)$, entonces los vértices intermedios de $C(i, j)$ pertenecen a $\{1, \dots, k - 1\}$. Por lo tanto un camino mínimo de i a j con todos sus vértices intermedios en $\{1, \dots, k - 1\}$ es también un camino mínimo de i a j con todos sus vértices intermedios en $\{1, \dots, k\}$
- Si k es un vértice intermedio de $C(i, j)$, entonces podemos descomponer $C(i, j)$ en $C(i, k)$ y $C(k, j)$. Por el lema anterior, $C(i, k)$ es un camino mínimo de i a k , con todos sus vértices intermedios en $\{1, \dots, k - 1\}$ y $C(k, j)$ es un camino mínimo de k a j , con todos sus vértices intermedios en $\{1, \dots, k - 1\}$

A partir esta observación anterior, se puede definir una fórmula recursiva para el camino mínimo. Sea $d_{ij}^{(k)}$ la distancia de un camino mínimo de i a j tal que todos sus vértices intermedios pertenecen al conjunto $\{1, \dots, k\}$. Si $k = 0 \implies d_{ij}^{(0)} = w_{ij}$, donde $w_{ij} = M[i][j]$. Definimos $d_{ij}^{(k)}$ de forma recursiva:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{si } k = 0 \\ \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{si } k \geq 1 \end{cases}$$

De esta forma, $D^{(n)} = (d_{ij}^{(n)})$ da la solución del algoritmo, donde $n = |V|$.

A continuación se muestra el pseudocódigo del algoritmo [3] :

```

function FLOYDWARSHALL(w)
    dist  $\leftarrow$  array  $|V| \times |V|$  inicializado a  $\infty$ 
    for each edge  $(u, v)$  do
        dist[u][v]  $\leftarrow w(u, v)$ 
    end for
    for each vertex v do
        dist[v][v]  $\leftarrow 0$ 
    end for
    for k from 1 to  $|V|$  do
        for i from 1 to  $|V|$  do
            for j from 1 to  $|V|$  do
                if dist[i][j]  $>$  dist[i][k] + dist[k][j] then
                    dist[i][j]  $\leftarrow$  dist[i][k] + dist[k][j]
                end if
            end for
        end for
    end for
end function

```

Se observa que no es necesario construir una nueva matriz para cada iteración, ya que se puede simplemente modificar la que existe, reduciendo así la complejidad espacial del algoritmo.

3.2. Matriz de sucesores

Este algoritmo ofrece una solución al problema *All-pairs shortest paths* más eficiente que aplicar un algoritmo *Single-source shortest path* partiendo de cada vértice del grafo. Gracias a ser un algoritmo de programación dinámica, la complejidad temporal del algoritmo es $O(|V|^3)$.

La programación dinámica hace referencia a dividir un problema en subproblemas y usar los resultados de esos subproblemas para resolver el problema original. En este caso, en la iteración k se usa la información obtenida en la iteración $k - 1$ exclusivamente.

Para detectar ciclos negativos, basta con comprobar si $dist[i][i] < 0$ dentro del if. Si es el caso, se ha encontrado un camino de i a i con peso negativo.

3.2. Matriz de sucesores

Para poder construir los caminos mínimos, y no obtener únicamente las distancias, se deberá emplear una matriz adicional, que llamaremos **matriz de sucesores**. Dicha matriz se construirá sucesivamente en cada iteración similar a la matriz de distancias. Tras la iteración k , la fila i y columna j contendrá el siguiente vértice en el camino mínimo de i a j , con todos los vértices intermedios en $\{1, \dots, k\}$, de la siguiente manera [2]:

Para $k = 0$, el camino mínimo de i a j no puede tener ningún vértice intermedio. Por lo tanto:

$$s_{ij}^{(0)} = \begin{cases} - & \text{si } w_{ij} = \infty \\ j & \text{si } w_{ij} < \infty \end{cases}$$

Para $k > 0$, si el camino mínimo de i a j contiene k como vértice intermedio, entonces debemos elegir como sucesor de i al sucesor de i a k con todos los vértices intermedios en $\{1, \dots, k - 1\}$. Por otro lado, si el camino mínimo de i a j no contiene k como vértice intermedio, entonces el sucesor no cambia; es el mismo que el sucesor i a j con todos los vértices intermedios en $\{1, \dots, k - 1\}$. Formalmente, para $k > 0$:

$$s_{ij}^{(k)} = \begin{cases} s_{ik}^{(k-1)} & \text{si } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ s_{ij}^{(k-1)} & \text{si } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Al igual que antes, $S^{(n)} = (s_{ij}^{(n)})$ da la solución que permite construir los caminos mínimos entre cada par de vértices del grafo.

Con esta modificación, el pseudocódigo queda de la siguiente manera [3] :

Capítulo 3. Algoritmo de Floyd-Warshall

```
function FLOYDWARSHALL(w)
    dist  $\leftarrow$  array  $|V| \times |V|$  inicializado a  $\infty$ 
    succ  $\leftarrow$  array  $|V| \times |V|$  inicializado a -
    for each edge  $(u, v)$  do
         $dist[u][v] \leftarrow w(u, v)$ 
         $succ[u][v] \leftarrow v$ 
    end for
    for each vertex  $v$  do
         $dist[v][v] \leftarrow 0$ 
         $succ[v][v] \leftarrow v$ 
    end for
    for k from 1 to  $|V|$  do
        for i from 1 to  $|V|$  do
            for j from 1 to  $|V|$  do
                if  $dist[i][j] > dist[i][k] + dist[k][j]$  then
                     $dist[i][j] \leftarrow dist[i][k] + dist[k][j]$ 
                     $succ[i][j] \leftarrow succ[i][k]$ 
                end if
            end for
        end for
    end for
end function
```

3.3. Construcción de caminos

Una vez se tiene la matriz de sucesores, la construcción del camino de un vértice i a otro vértice j se puede realizar de la siguiente manera [3] :

```
function CONSTRUIRCAMINO(S, i, j)
    if  $S[i][j] = -$  then
        return []
    end if
    camino  $\leftarrow$  []
    while  $i \neq j$  do
        camino.append(i)
         $i \leftarrow S[i][j]$ 
    end while
    return camino
end function
```

3.4. Ejemplo

3.4.1. Algoritmo de Floyd-Warshall

Sea G el grafo dirigido del capítulo 2, cuya matriz de pesos ya conocemos. Vamos a aplicar el algoritmo para calcular el camino mínimo del vértice 4 al vértice 2. Para no mostrar las 125 iteraciones ($i, j, k = 1, 2, 3, 4, 5$), únicamente se mostrarán los pasos relevantes, en los que se realizan cambios en las matrices.

Inicialmente, se tiene:

$$D^{(0)} = \begin{pmatrix} 0 & 1 & \infty & \infty & 2 \\ 1 & 0 & 2 & \infty & \infty \\ -1 & \infty & 0 & \infty & 4 \\ \infty & \infty & 4 & 0 & 7 \\ 2 & \infty & \infty & \infty & 0 \end{pmatrix} \quad S^{(0)} = \begin{pmatrix} 1 & 2 & - & - & 5 \\ 1 & 2 & 3 & - & - \\ 1 & - & 3 & - & 5 \\ - & - & 3 & 4 & 5 \\ 1 & - & - & - & 5 \end{pmatrix}$$

Tomando el vértice 1 como vértice intermedio:

La distancia actual del vértice $i = 2$ al vértice $j = 5$ es $D[2][5] = \infty$, mientras que pasando por el vértice $k = 1$ pasa a ser $D[2][1] + D[1][5] = 1 + 2 = 3$. Luego $S[2][5]$ pasa a ser $S[2][1] = 1$.

La distancia actual del vértice $i = 3$ al vértice $j = 2$ es $D[3][2] = \infty$, mientras que pasando por el vértice $k = 1$ pasa a ser $D[3][1] + D[1][2] = -1 + 1 = 0$. Luego $S[3][2]$ pasa a ser $S[3][1] = 1$.

La distancia actual del vértice $i = 3$ al vértice $j = 5$ es $D[3][5] = 4$, mientras que pasando por el vértice $k = 1$ pasa a ser $D[3][1] + D[1][5] = -1 + 2 = 1$. Luego $S[3][5]$ pasa a ser $S[3][1] = 1$.

La distancia actual del vértice $i = 5$ al vértice $j = 2$ es $D[5][2] = \infty$, mientras que pasando por el vértice $k = 1$ pasa a ser $D[5][1] + D[1][2] = 2 + 1 = 3$. Luego $S[5][2]$ pasa a ser $S[5][1] = 1$.

Al finalizar la iteración $k = 1$, queda:

$$D^{(1)} = \begin{pmatrix} 0 & 1 & \infty & \infty & 2 \\ 1 & 0 & 2 & \infty & 3 \\ -1 & 0 & 0 & \infty & 1 \\ \infty & \infty & 4 & 0 & 7 \\ 2 & 3 & \infty & \infty & 0 \end{pmatrix} \quad S^{(1)} = \begin{pmatrix} 1 & 2 & - & - & 5 \\ 1 & 2 & 3 & - & 1 \\ 1 & 1 & 3 & - & 1 \\ - & - & 3 & 4 & 5 \\ 1 & 1 & - & - & 5 \end{pmatrix}$$

Tomando el vértice 2 como vértice intermedio:

La distancia actual del vértice $i = 1$ al vértice $j = 3$ es $D[1][3] = \infty$, mientras que pasando por el vértice $k = 2$ pasa a ser $D[1][2] + D[2][3] = 1 + 2 = 3$. Luego $S[1][3]$ pasa a ser $S[1][2] = 2$.

Capítulo 3. Algoritmo de Floyd-Warshall

La distancia actual del vértice $i = 5$ al vértice $j = 3$ es $D[5][3] = \infty$, mientras que pasando por el vértice $k = 2$ pasa a ser $D[5][2] + D[2][3] = 3 + 2 = 5$. Luego $S[5][3]$ pasa a ser $S[5][2] = 1$.

Al finalizar la iteración $k = 2$, queda:

$$D^{(2)} = \begin{pmatrix} 0 & 1 & 3 & \infty & 2 \\ 1 & 0 & 2 & \infty & 3 \\ -1 & 0 & 0 & \infty & 1 \\ \infty & \infty & 4 & 0 & 7 \\ 2 & 3 & 5 & \infty & 0 \end{pmatrix} S^{(2)} = \begin{pmatrix} 1 & 2 & 2 & - & 5 \\ 1 & 2 & 3 & - & 1 \\ 1 & 1 & 3 & - & 1 \\ - & - & 3 & 4 & 5 \\ 1 & 1 & 1 & - & 5 \end{pmatrix}$$

Tomando el vértice 3 como vértice intermedio:

La distancia actual del vértice $i = 4$ al vértice $j = 1$ es $D[4][1] = \infty$, mientras que pasando por el vértice $k = 3$ pasa a ser $D[4][3] + D[3][1] = 4 + (-1) = 3$. Luego $S[4][1]$ pasa a ser $S[4][3] = 3$.

La distancia actual del vértice $i = 4$ al vértice $j = 2$ es $D[4][2] = \infty$, mientras que pasando por el vértice $k = 3$ pasa a ser $D[4][3] + D[3][2] = 4 + 0 = 4$. Luego $S[4][2]$ pasa a ser $S[4][3] = 3$.

La distancia actual del vértice $i = 4$ al vértice $j = 5$ es $D[4][5] = 7$, mientras que pasando por el vértice $k = 3$ pasa a ser $D[4][3] + D[3][5] = 4 + 1 = 5$. Luego $S[4][5]$ pasa a ser $S[4][3] = 3$.

Al finalizar la iteración $k = 3$, queda:

$$D^{(3)} = \begin{pmatrix} 0 & 1 & 3 & \infty & 2 \\ 1 & 0 & 2 & \infty & 3 \\ -1 & 0 & 0 & \infty & 1 \\ 3 & 4 & 4 & 0 & 5 \\ 2 & 3 & 5 & \infty & 0 \end{pmatrix} S^{(3)} = \begin{pmatrix} 1 & 2 & 2 & - & 5 \\ 1 & 2 & 3 & - & 1 \\ 1 & 1 & 3 & - & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 1 & 1 & - & 5 \end{pmatrix}$$

Siguiendo de la misma manera, resulta que ya no se producen más cambios, por lo que $D^{(3)} = D^{(4)} = D^{(5)}$ y $S^{(3)} = S^{(4)} = S^{(5)}$

3.4.2. Construcción del camino mínimo

Ahora, vamos a calcular el camino mínimo del vértice 4 al vértice 2 a partir de $S^{(5)}$:

El algoritmo comienza en el vértice 4. El siguiente paso para llegar al vértice 2 es ir al vértice que se encuentra en $S^{(5)}[4][2] = 3$.

A continuación, desde el vértice 3, el siguiente paso para llegar al vértice 2 es ir al vértice que se encuentra en $S^{(5)}[3][2] = 1$.

3.4. Ejemplo

Finalmente, desde el vértice 1, el siguiente paso para llegar al vértice 2 es ir al vértice que se encuentra en $S^{(5)}[1][2] = 2$.

De esta forma, se ha construido el camino mínimo del vértice 4 al vértice 2: $4 \rightarrow 3 \rightarrow 1 \rightarrow 2$.

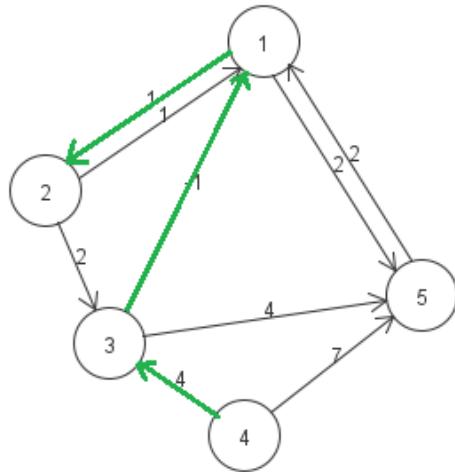


Figura 3.1: Camino mínimo del vértice 4 al vértice 2

Se puede observar que para calcular el camino mínimo de un vértice i a otro vértice j , únicamente se utilizan los valores que se encuentran en la columna j -ésima de la matriz de sucesores. Esto se da ya que en cada paso, el vértice de llegada siempre es el mismo, independientemente del vértice de salida.

Capítulo 4

Tecnologías empleadas

Para el código se ha empleado el lenguaje de programación **Python**, por ser el lenguaje elegido para el desarrollo del backend del *Proyecto Calculadora*. Es uno de los lenguajes de programación más usados en el mundo [4], gracias a su claridad y simplicidad. A pesar de ser un lenguaje más lento que otros, esto no supone un problema en el contexto de este tipo de TFGs, ya que no se busca la eficiencia máxima de los algoritmos, sino su fácil comprensión.

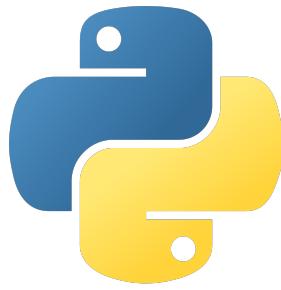


Figura 4.1: Logo de Python

Para el desarrollo del back-end se ha empleado **Flask** [5]. Flask es un framework web ligero y flexible escrito en Python. Está diseñado para facilitar la creación de aplicaciones web de forma rápida y sencilla, con un enfoque minimalista que permite a los desarrolladores tener un alto grado de control sobre la estructura de sus aplicaciones. Una de las características más destacadas de Flask es su simplicidad y su naturaleza no restrictiva. A diferencia de otros frameworks más completos que imponen una estructura rígida, Flask proporciona solo las herramientas básicas necesarias para construir una aplicación web, dejando la elección de herramientas adicionales y la arquitectura de la aplicación en manos del desarrollador.



Figura 4.2: Logo de Flask

Dentro de Python se han empleado varias librerías. A continuación se muestran las principales:

- **networkx** es una librería que permite trabajar con grafos. En el contexto de este TFG, se ha usado para crear el grafo que introduce el usuario a partir de su matriz de adyacencia para su visualización en la aplicación web.
- **matplotlib** es una librería para la generación de gráficas. Sin embargo, en este caso se ha utilizado para guardar la imagen del grafo generado con networkx.
- **base64** es una librería que permite codificar bytes usando el algoritmo base64, que se ha utilizado para codificar la imagen generada con matplotlib para poder mostrarla en el front-end con HTML.
- **array-to-latex** es una librería que se ha usado para facilitar la conversión de arrays a formato L^AT_EX, el cual es empleado para explicar los pasos que se han ido tomando en la ejecución del algoritmo.
- **NumPy** es una librería de Python principalmente utilizada para trabajar con arrays y realizar cálculos matemáticos y científicos. Su uso ha sido necesario para poder trabajar con networkx y con array-to-latex.

Para el desarrollo del front-end se ha empleado **Angular** [6], un framework desarrollado en TypeScript y mantenido por Google. Se trata de un framework para aplicaciones web basado en componentes, lo que significa que la interfaz de usuario se divide en pequeños módulos independientes y reutilizables. Se utiliza principalmente para construir aplicaciones web de una sola página, donde el contenido se carga dinámicamente en la página sin necesidad de recargarla por completo.



Figura 4.3: Logo de Angular

Para la parte de testing, se ha decidido crear una aplicación gráfica en Java usando la librería **Java Swing** que permita al usuario crear y editar grafos de una manera intuitiva y visual. De esta forma, cuando el usuario ha creado el grafo y pulsa el botón de resolver, la aplicación crea un nuevo proceso que ejecuta el código desarrollado en Python y le muestra al usuario el resultado por pantalla. Esta aplicación ha sido desarrollada únicamente para el testing, y es independiente del *Proyecto Calculadora*. En las figuras 2.1 y 2.2 se pueden ver ejemplos de grafos creados con esta aplicación.

Java Swing es una biblioteca gráfica para Java. Incluye widgets que el usuario puede utilizar, tales como botones, listas desplegables, campos de texto, etc. Se ha decidido emplear esta librería de Java para el testing en vez de otras, como Tkinter o PyQt por varias razones:

- Permite al usuario interactuar con la aplicación de una forma muy sencilla. Por lo tanto, no resulta muy complicado dibujar el grafo mientras que el usuario lo va creando.
- Una mayor familiarización con esta librería que con las demás.
- A pesar de haber sido lanzado en 1998, se le sigue dando soporte a día de hoy.

Para el desarrollo del código en Python y TypeScript, se ha empleado **Visual Studio Code** como editor de texto gracias a su gran cantidad de extensiones que ayudan a la hora de desarrollar código y a la libertad que ofrece, mientras que para el desarrollo del código en Java, se ha empleado **Eclipse** como IDE (entorno de desarrollo integrado) por ser el IDE por excelencia a la hora de desarrollar aplicaciones en Java.



Figura 4.4: Logo de Visual Studio Code

Capítulo 5

Desarrollo del código

Como se ha mencionado en el capítulo anterior, para el desarrollo del código del back-end se ha utilizado Flask, una librería de Python. Cuando el usuario desea calcular la solución, se accederá a la ruta `/floyd_marshall`, pasando como parámetro la matriz de adyacencia introducida por el usuario, donde se ejecutará la función `floyd_marshall()`.

Lo primero que se hace dentro de la función es comprobar que la matriz de adyacencia es válida:

```
@app.route("/floyd_marshall/<string:matrix>", methods = ["GET"])
def floyd_marshall(matrix):
    # Convertir el string a una matriz
    matrix = string_to_matrix(matrix)
```

Para ello se utiliza el siguiente código, donde se comprueba que el string tenga un formato específico, determinado por una expresión regular:

```
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False

def string_to_matrix(str):
    str = str.replace(" ", "")
    output = []

    # Comprobar que el string sigue un formato valido
    if not re.match(r"^\[\[[a-zA-Z0-9-]+([, [a-zA-Z0-9-]+)](, \[\[a-zA-Z0-9-]+([, [a-zA-Z0-9-]+)]\])*]\]$", str):
        return []

    for sublist in str.split(", "):
```

Capítulo 5. Desarrollo del código

```
sublist = sublist.replace("[", "").replace("]", "")
data = []

for number in sublist.split(","):
    if number.upper() == "INF":
        data.append(INF)
    elif is_number(number) and number.upper() != "-INF":
        data.append(int(float(number)))
    else:
        return []

output.append(data)

return output
```

En caso de que haya habido algún problema, se devuelve un mensaje de error:

```
# Comprobar si se ha introducido una matriz
if matrix == []:
    return {
        "error": "La matriz de adyacencia es invalida"
    }, 400

# Comprobar que la matriz introducida es cuadrada
if not all (len (row) == len (matrix) for row in matrix):
    return {
        "error": "La matriz de adyacencia ha de ser cuadrada"
    }, 400
```

Una vez que se ha determinado que el formato de la matriz es válido, se comprueba que la matriz introducida no contenga bucles, y, en caso de que los tenga, se devuelve un mensaje de error:

```
# Comprobar que la matriz introducida no contiene bucles
if has_loops(matrix):
    return {
        "error": "La matriz de adyacencia no puede contener bucles"
    }, 400
```

Para ello, se utiliza el siguiente código:

```
def has_loops(matrix):
    n = len(matrix)

    for i in range(n):
        if matrix[i][i] != 0:
            return True
```

```
    return False
```

A continuación, se crean las matrices de distancias y sucesores tal y como se explica en el capítulo 3:

```
n = len(matrix)
steps = []

# Crear matrices de distancias y sucesores
distance_matrix = [[INF] * n for i in range(n)]
successor_matrix = [[-1] * n for i in range(n)]
successors = []
distances = []

# Rellenar matrices con datos iniciales
for i in range(n):
    for j in range(n):
        if i == j:
            distance_matrix[i][j] = 0
            successor_matrix[i][j] = j + 1
        elif matrix[i][j] != INF:
            distance_matrix[i][j] = matrix[i][j]
            successor_matrix[i][j] = j + 1

successors.append(copy.deepcopy(successor_matrix))
distances.append(copy.deepcopy(distance_matrix))

paso1 = str(distance_matrix).replace(" ", " ")
paso2 = str(successor_matrix).replace(" ", " ")
pasoLatex1 = a2l.to_ltx(np.asarray(distance_matrix), print_out =
    False, frmt = "{:6.0f}").replace(" ", "").replace("\n", "") .
    replace("inf", "\infty")
pasoLatex2 = a2l.to_ltx(np.asarray(successor_matrix), print_out
    = False, frmt = "{:6.0f}").replace(" ", "").replace("\n", "")

steps.append({
    "paso": (paso1, paso2),
    "pasoLatex": (pasoLatex1, pasoLatex2),
    "descripcion": "Ha finalizado la iteración k = 0"
})
```

Una vez que se han creado las matrices iniciales, se procede a aplicar el algoritmo:

```
# Aplicar el algoritmo
for k in range(n):
```

Capítulo 5. Desarrollo del código

```
for i in range(n):
    for j in range(n):
        # Se ha encontrado un camino mas corto
        if distance_matrix[i][k] + distance_matrix[k][j] <
           distance_matrix[i][j]:
            if i != j:
                # Ya hay un camino entre los vértices
                if distance_matrix[i][j] != INF:
                    old_path = reconstruct_path(str(successors[-1])).
                        replace(" ", ""), i + 1, j + 1)
                    old_path = old_path[-1]["paso"]

                    explanation = "El camino actual del vértice " + str(
                        i + 1) + " al vértice " + str(j + 1) + ", tomando
                        como vértices intermedios al conjunto " + str(
                        list(range(1, k + 1))) + " es: " + " - ".join(str(
                            x) for x in old_path) + ", con un peso = m[" +
                            str(i + 1) + "][" + str(j + 1) + "] = " + str(
                            distance_matrix[i][j])
                # No hay camino entre los vértices
            else:
                explanation = "Actualmente no existe camino del vé-
                    rtice " + str(i + 1) + " al vértice " + str(j +
                        1) + " ya que " + "m[" + str(i + 1) + "][" + str(
                            j + 1) + "] = inf"

        # Actualizar matrices
        distance_matrix[i][j] = distance_matrix[i][k] +
            distance_matrix[k][j]
        successor_matrix[i][j] = successor_matrix[i][k]

        # Calcular camino de i a k
        new_path1 = reconstruct_path(str(successors[-1])).
            replace(" ", ""), i + 1, k + 1)
        new_path1 = new_path1[-1]["paso"]

        # Calcular camino de k a j
        new_path2 = reconstruct_path(str(successors[-1])).
            replace(" ", ""), k + 1, j + 1)
        new_path2 = new_path2[-1]["paso"][1:]

        # Juntar caminos
        new_path1.extend(new_path2)

        # Detección de ciclo negativo
        if distance_matrix[i][i] < 0:
            return {
```

```

        "error": "La matriz de adyacencia no puede
                  contener ciclos negativos: El ciclo " + " - ".
        join(str(x) for x in new_path1) + " tiene peso
        = " + str(get_path_weight(matrix, new_path1))
    }, 400

signo = "-" if distance_matrix[k][j] < 0 else "+"
explanation += ", mientras que tomando como vértices
intermedios al conjunto " + str(list(range(1, k +
2))) + ", el camino pasa a ser: " + " - ".join(str(
x) for x in new_path1) + ", con un peso = m[" + str(
i + 1) + "][" + str(k + 1) + "] + m[" + str(k + 1)
+ "][" + str(j + 1) + "] = " + str(distance_matrix[
i][k]) + " " + signo + " " + str(abs(
distance_matrix[k][j])) + " = " + str(
distance_matrix[i][k] + distance_matrix[k][j])

distance_copy = copy.deepcopy(distances[-1])
distance_copy[i][j] = distance_matrix[i][j]

successors_copy = copy.deepcopy(successors[-1])
successors_copy[i][j] = successor_matrix[i][j]

# Agregar paso actual
steps.append({
    "paso": (str(distance_copy).replace(" ", ""), str(
        successors_copy).replace(" ", "")),
    "pasoLatex": (matrix_to_latex_algorithm(
        distance_copy, i, j, k, "distance"),
        matrix_to_latex_algorithm(successors_copy, i, j,
        k, "successor")),
    "descripcion": explanation
})

successors.append(copy.deepcopy(successor_matrix))
distances.append(copy.deepcopy(distance_matrix))

paso1 = str(distance_matrix).replace(" ", "")
paso2 = str(successor_matrix).replace(" ", "")
pasoLatex1 = a2l.to_ltx(np.asarray(distance_matrix), print_out
    = False, frmt = "{:6.0f}").replace(" ", "").replace("\n",
    "").replace("inf", "\\\infty")
pasoLatex2 = a2l.to_ltx(np.asarray(successor_matrix),
    print_out = False, frmt = "{:6.0f}").replace(" ", "") .
    replace("\n", "")
steps.append({
    "paso": (paso1, paso2),

```

Capítulo 5. Desarrollo del código

```
"pasoLatex": (pasoLatex1, pasoLatex2),
"descripcion": "Ha finalizado la iteración k = " + str(k +
    1)
}

return steps
```

En el código se han empleado las funciones auxiliares *get_path_weight()* para obtener el peso de un camino, y *matrix_to_latex_algorithm()* para obtener el código *LATEX*de una matriz, resaltando en rojo los cambios, y en azul los datos que se han usado para calcular el cambio:

```
def get_path_weight(matrix, path):
    w = 0

    for i in range(len(path) - 1):
        w += matrix[path[i] - 1][path[i + 1] - 1]

    return w

def matrix_to_latex_algorithm(matrix, i, j, k, case):
    # Caso matriz de distancias
    if case == "distance":
        latex = a2l.to_ltx(np.asarray(matrix), print_out = False,
                           frmt = "{:6.0f}").replace(" ", "").replace("\n", "")

        row_list = latex.split("\\\\")
        row_list[0] = row_list[0][15:]
        row_list[-1] = row_list[-1][:13]

        elem_list = row_list[i].split("&")
        elem_list[j] = "\\\textcolor{red}{\\textbf{" + str(elem_list[
            j]) + "}}"
        row_list[i] = "&".join(elem_list)

        elem_list = row_list[i].split("&")
        elem_list[k] = "\\\textcolor{blue}{\\textbf{" + str(elem_list[
            k]) + "}}"
        row_list[i] = "&".join(elem_list)

        elem_list = row_list[k].split("&")
        elem_list[j] = "\\\textcolor{blue}{\\textbf{" + str(elem_list[
            j]) + "}}"
        row_list[k] = "&".join(elem_list)

    latex = "\\begin{bmatrix}" + "\\\\".join(row_list) + "\\end{
        bmatrix}"
```

```

# Caso matriz de sucesores
elif case == "successor":
    latex = a2l.to_ltx(np.asarray(matrix), print_out = False,
                      frmt = "{:6.0f}").replace(" ", "").replace("\n", "")

    row_list = latex.split("\\\\")
    row_list[0] = row_list[0][15:]
    row_list[-1] = row_list[-1][:13]

    elem_list = row_list[i].split("&")
    elem_list[j] = "\\textcolor{red}{\\textbf{" + str(elem_list[
        j]) + "}}"
    row_list[i] = "&".join(elem_list)

    elem_list = row_list[i].split("&")
    elem_list[k] = "\\textcolor{blue}{\\textbf{" + str(elem_list[
        k]) + "}}"
    row_list[i] = "&".join(elem_list)

return ("\\begin{bmatrix}" + "\\\\".join(row_list) + "\\end{
bmatrix}").replace("inf", "\\infty")

```

Además, como se ha mencionado en la introducción, uno de los objetivos del trabajo es que el usuario pueda obtener el camino mínimo entre 2 vértices usando la matriz de sucesores. Para ello, se ha creado otro endpoint /construir_camino, donde se ejecuta la función *reconstruct_path()*, que también se utiliza durante el algoritmo para la explicación y para mostrar el ciclo de peso negativo en caso de que lo haya:

```

@app.route("/construir_camino/<string:matrix>&<int(signed=True) :
    start_node>&<int(signed=True):end_node>", methods = ["GET"])
def reconstruct_path(matrix, start_node, end_node):
    # Convertir el string a una matriz
    matrix = string_to_matrix(matrix)

    # Comprobar si se ha introducido una matriz
    if matrix == []:
        return {
            "error": "La matriz es invalida"
        }, 400

    current_node = start_node - 1

    # Comprobar si la matriz es cuadrada
    if not all (len (row) == len (matrix) for row in matrix):
        return {
            "error": "La matriz ha de ser cuadrada"
        }, 400

```

Capítulo 5. Desarrollo del código

```
n = len(matrix)

# Comprobar que el nodo de salida introducido es valido
if start_node > n or start_node < 1:
    return {
        "error": "Nodo de salida invalido"
}, 400

# Comprobar que el nodo de llegada introducido es valido
if end_node > n or end_node < 1:
    return {
        "error": "Nodo de llegada invalido"
}, 400

# Comprobar si los nodos de salida y llegada son iguales
if start_node == end_node:
    return {
        "error": "Los nodos de salida y llegada son el mismo"
}, 400

path = []
steps = []

while current_node != end_node - 1:

    # Comprobar que se puede llegar del nodo de salida al de llegada
    if matrix[current_node][end_node - 1] < 1 or matrix[
        current_node][end_node - 1] > n:
        return {
            "error": "No se puede llegar del nodo " + str(start_node)
                    + " al nodo " + str(end_node)
}, 400

    # Comprobar que no he vuelto a un nodo ya visitado
    if current_node + 1 in path:
        return {
            "error": "No se puede llegar del nodo " + str(
                start_node) + " al nodo " + str(end_node)
}, 400

    path.append(current_node + 1)

    # Agregar paso actual
    steps.append({
        "paso": copy.deepcopy(path),
```

```

    "pasoLatex": matrix_to_latex_path(matrix, current_node,
        end_node - 1),
    "descripcion": "El nodo actual es " + str(current_node +
        1) + ". Para llegar al nodo " + str(end_node) + "
        hay que ir al nodo s[" + str(current_node + 1) + "] ["
        + str(end_node) + "] = " + str(matrix[current_node] [
        end_node - 1]))
}

current_node = matrix[current_node] [end_node - 1] - 1

# Agregar último paso
steps.append({
    "paso": path + [end_node],
    "pasoLatex": matrix_to_latex_path(matrix, end_node - 1,
        end_node - 1),
    "descripcion": "Ya se ha llegado al nodo " + str(end_node)
})

return steps

```

De forma similar al caso anterior, se ha empleado la función auxiliar `matrix_to_latex_path()` para obtener el código \LaTeX de una matriz, resaltando en rojo los cambios:

```

def matrix_to_latex_path(matrix, i, j):
    latex = a21.to_ltx(np.asarray(matrix), print_out = False, frmt
        = "{:6.0f}").replace(" ", "").replace("\n", "")

    row_list = latex.split("\\\\")
    row_list[0] = row_list[0][15:]
    row_list[-1] = row_list[-1][:-13]
    elem_list = row_list[int(i)].split("&")
    elem_list[j] = "\\textcolor{red}{\\textbf{" + str(elem_list[j])
        + "}}"
    row_list[int(i)] = "&".join(elem_list)

    return "\\begin{bmatrix}" + "\\\\".join(row_list) + "\\end{
        bmatrix}"

```


Capítulo 6

Incorporación del código en la aplicación

Como se ha mencionado en el capítulo 4, la parte del front-end se ha programado con la librería Angular. En primer lugar, se ha creado una entrada en la ruta `/matDiscreta2` de la aplicación:

Matemática Discreta II



Figura 6.1: Botón para acceder a la página creada

Una vez que el usuario pulsa sobre el botón “Abrir“, se le redirige a la página `/matDiscreta2/floydWarshall`, que se muestra a continuación:

Capítulo 6. Incorporación del código en la aplicación

Algoritmo de Floyd-Warshall

El algoritmo de Floyd-Warshall se emplea para calcular los caminos mínimos entre cada par de vértices de un grafo.

Floyd-Warshall

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:

```
[[0,3,8,inf,-4],[inf,0,inf,1,7],[inf,4,0,inf,inf],[2,inf,-5,0,inf],[inf,inf,inf,6,0]]
```

Introducir matriz por celdas:

Filas: 5 Columns: 5

$$\begin{pmatrix} 0 & 3 & 8 & \text{inf} & -4 \\ \text{inf} & 0 & \text{inf} & 1 & 7 \\ \text{inf} & 4 & 0 & \text{inf} & \text{inf} \\ 2 & \text{inf} & -5 & 0 & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & 6 & 0 \end{pmatrix}$$

Importar matriz de CSV

Calcular

Figura 6.2: Página Algoritmo de Floyd-Warshall

En esta página, el usuario puede introducir su matriz de adyacencia, ya sea en formato lineal o por celdas. También existe la opción de importar una matriz desde un fichero csv:

Importar matriz de CSV

El archivo que contenga la matriz deseada debe seguir el siguiente formato:

- Los números de una misma fila en una única línea.
- Las columnas se indicarán separando con coma/punto y coma los números de la fila.
- Una nueva fila se introducirá con un salto de línea.

Browse... csv.txt

Figura 6.3: Importar matriz de CSV

Una vez que el usuario ha introducido la matriz que desea, deberá pulsar en el botón “Calcular”, que le mostrará una imagen del grafo junto con la solución:

El grafo introducido es:

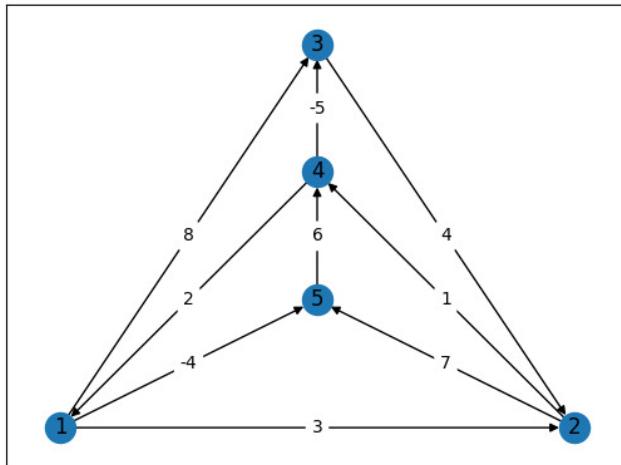


Figura 6.4: Imagen del grafo a partir de la matriz introducida

Para la generación de la imagen se ha usado el siguiente código [7]:

```
@app.route("/imagen_grafo/<string:matrix>", methods = ["GET"])
def imagen_grafo(matrix):
    def my_draw_networkx_edge_labels(G, pos, edge_labels):
        ax = plt.gca()
        text_items = {}
        for (n1, n2), label in edge_labels.items():
            (x1, y1) = pos[n1]
            (x2, y2) = pos[n2]
            (x, y) = (x1 * 0.5 + x2 * 0.5, y1 * 0.5 + y2 * 0.5)
            pos_1 = ax.transData.transform(np.array(pos[n1]))
            pos_2 = ax.transData.transform(np.array(pos[n2]))
            linear_mid = 0.5 * pos_1 + 0.5 * pos_2
            d_pos = pos_2 - pos_1
            rotation_matrix = np.array([(0, 1), (-1, 0)])
            ctrl_1 = linear_mid + 0.08 * rotation_matrix@d_pos
            ctrl_mid_1 = 0.5 * pos_1 + 0.5 * ctrl_1
            ctrl_mid_2 = 0.5 * pos_2 + 0.5 * ctrl_1
            bezier_mid = 0.5 * ctrl_mid_1 + 0.5 * ctrl_mid_2
            (x, y) = ax.transData.inverted().transform(bezier_mid)
            text_items[(n1, n2)] = ax.text(x, y, label, size = 10,
                color = "k", family = "sans-serif", weight = "normal",
                alpha = None, horizontalalignment = "center",
                verticalalignment = "center", rotation = 0.0, transform
```

Capítulo 6. Incorporación del código en la aplicación

```
= ax.transData, bbox = dict(boxstyle = "round", ec =
(1.0, 1.0, 1.0), fc = (1.0, 1.0, 1.0)), zorder = 1,
clip_on = True)

return text_items

matrix = string_to_matrix(matrix)

if matrix == []:
    return {
        "error": "La matriz de adyacencia es invalida"
    }, 400

# Comprobar que la matriz introducida es cuadrada
if not all (len (row) == len (matrix) for row in matrix):
    return {
        "error": "La matriz de adyacencia ha de ser cuadrada"
    }, 400

# Comprobar que la matriz introducida no contiene bucles
if has_loops(matrix):
    return {
        "error": "La matriz de adyacencia no puede contener bucles"
    }
}, 400

G = nx.DiGraph()

for i in range(len(matrix)):
    G.add_node(i + 1)
    for j in range(len(matrix[0])):
        if i != j and matrix[i][j] != INF:
            G.add_edge(i + 1, j + 1, weight = str(matrix[i][j]))

pos = nx.planar_layout(G)

curved_edges = [edge for edge in G.edges() if reversed(edge)
    in G.edges()]
straight_edges = list(set(G.edges()) - set(curved_edges))
edge_weights = nx.get_edge_attributes(G, "weight")
curved_edge_labels = {edge: edge_weights[edge] for edge in
    curved_edges}
straight_edge_labels = {edge: edge_weights[edge] for edge in
    straight_edges}
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_edges(G, pos, edgelist=straight_edges)
```

```

nx.draw_networkx_edges(G, pos, edgelist=curved_edges,
    connectionstyle = "arc3, rad = 0.08")
nx.draw_networkx_edge_labels(G, pos, edge_labels =
    straight_edge_labels, rotate = False)
my_draw_networkx_edge_labels(G, pos, edge_labels =
    curved_edge_labels)

img_buffer = io.BytesIO()
plt.savefig(img_buffer, format = "png")
img_buffer.seek(0)
plt.clf()

return base64.b64encode(img_buffer.getvalue()).decode("utf-8")

```

Solución:

Las matrices resultantes son:

Forma lineal de matriz de pesos:

`[[0,1,-3,2,-4],[3,0,-4,1,-1],[7,4,0,5,3],[2,-1,-5,0,-2],[8,5,1,6,0]]`

Forma LaTeX de matriz de pesos:

$$\begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

Forma lineal de matriz de sucesores:

`[[1,5,5,5,5],[4,2,4,4,4],[2,2,3,2,2],[1,3,3,4,1],[4,4,4,4,5]]`

Forma LaTeX de matriz de sucesores:

$$\begin{bmatrix} 1 & 5 & 5 & 5 & 5 \\ 4 & 2 & 4 & 4 & 4 \\ 2 & 2 & 3 & 2 & 2 \\ 1 & 3 & 3 & 4 & 1 \\ 4 & 4 & 4 & 4 & 5 \end{bmatrix}$$

MostrarPasos

Figura 6.5: Solución del problema

Para mostrar el algoritmo paso a paso, el usuario deberá pulsar sobre el botón “Mostrar Pasos“. Esto se mostrará en el siguiente capítulo, cuando se muestren algunos ejemplos del funcionamiento.

Una vez que la solución ha sido calculada, el usuario puede copiar al portapapeles la forma lineal de la matriz de sucesores, e introducirla en la página Construir camino:

Capítulo 6. Incorporación del código en la aplicación

Forma lineal de matriz de sucesores:

[[1,5,5,5,5],[4,2,4,4,4],[2,2,3,2,2],[1,3,3,4,1],[4,4,4,4,5]]

Figura 6.6: Copiar solución al portapepeles

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:
[[1,5,5,5,5],[4,2,4,4,4],[2,2,3,2,2],[1,3,3,4,1],[4,4,4,4,5]]

Introducir matriz por celdas:
Filas: 5 Columns: 5

$$\begin{pmatrix} 1 & 5 & 5 & 5 & 5 \\ 4 & 2 & 4 & 4 & 4 \\ 2 & 2 & 3 & 2 & 2 \\ 1 & 3 & 3 & 4 & 1 \\ 4 & 4 & 4 & 4 & 5 \end{pmatrix}$$

Introducir vértice de salida: 1

Introducir vértice de llegada: 3

Importar matriz de CSV

Calcular

Figura 6.7: Página Construir camino

De nuevo, el usuario puede introducir su matriz de sucesores a mano, ya sea en formato lineal o por celdas, o también tiene la opción de importar una matriz desde un fichero csv. Una vez que ha introducido la matriz, y ha seleccionado los vértices de salida y de llegada, pulsando sobre el botón “Calcular” se obtiene la solución:

Solución

El camino es:

Forma lineal:

[1,5,4,3]

MostrarPasos

Figura 6.8: Camino mínimo

Al igual que en el caso anterior, en el capítulo siguiente se mostrarán ejemplos del funcionamiento paso a paso.

Capítulo 7

Resultados y conclusiones

En este capítulo se mostrarán los resultados obtenidos mediante ejemplos y al final se discutirán las conclusiones extraídas del trabajo.

7.1. Ejemplo 1

7.1.1. Algoritmo

Llamada a la función floyd_marshall("[[0,4,inf,5,inf],[inf,0,1,inf,6],[2,inf,0,3,inf],[inf,inf,1,0,inf],[1,2,inf,4,0]]"):

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:
[[0,4,inf,5,inf],[inf,0,1,inf,6],[2,inf,0,3,inf],[inf,inf,1,0,inf],[1,2,inf,4,0]]

Introducir matriz por celdas:
Filas: 5 Columns: 5

$$\begin{pmatrix} 0 & 4 & \text{inf} & 5 & \text{inf} \\ \text{inf} & 0 & 1 & \text{inf} & 6 \\ 2 & \text{inf} & 0 & 3 & \text{inf} \\ \text{inf} & \text{inf} & 1 & 0 & \text{inf} \\ 1 & 2 & \text{inf} & 4 & 0 \end{pmatrix}$$

Importar matriz de CSV
Calcular

Figura 7.1: Introducción del input

Capítulo 7. Resultados y conclusiones

El grafo introducido es:

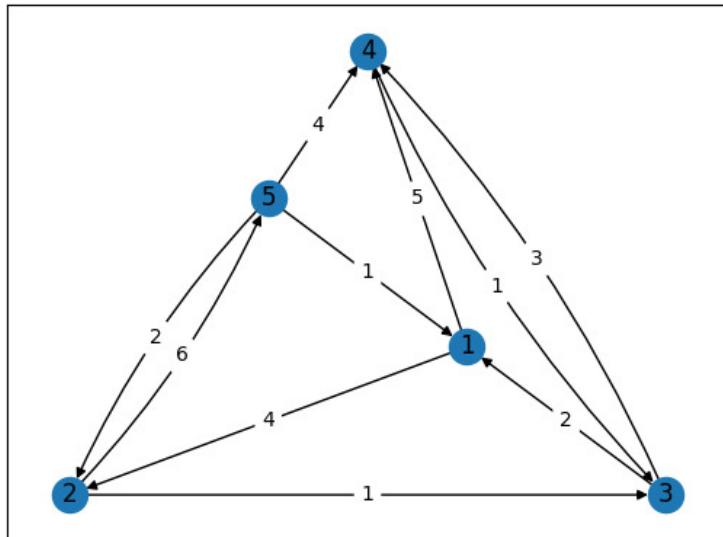


Figura 7.2: Imagen generada del grafo introducido

Solución:

Las matrices resultantes son:

Forma lineal de matriz de pesos:

`[[0,4,5,5,10],[3,0,1,4,6],[2,6,0,3,12],[3,7,1,0,13],[1,2,3,4,0]]`

Forma LaTeX de matriz de pesos:

$$\begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ 3 & 0 & 1 & 4 & 6 \\ 2 & 6 & 0 & 3 & 12 \\ 3 & 7 & 1 & 0 & 13 \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}$$

Forma lineal de matriz de sucesores:

`[[1,2,2,4,2],[3,2,3,3,5],[1,1,3,4,1],[3,3,3,4,3],[1,2,2,4,5]]`

Forma LaTeX de matriz de sucesores:

$$\begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

MostrarPasos

Figura 7.3: Solución del problema

7.1. Ejemplo 1

Pasos

Paso nº 1: Ha finalizado la iteración $k = 0$

$$m : \begin{bmatrix} 0 & 4 & \infty & 5 & \infty \\ \infty & 0 & 1 & \infty & 6 \\ 2 & \infty & 0 & 3 & \infty \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & \infty & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & -1 & 4 & -1 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & -1 & 3 & 4 & -1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 2: Actualmente no existe camino del vértice 3 al vértice 2 ya que $m[3][2] = \infty$, mientras que tomando como vértices intermedios al conjunto $[1]$, el camino pasa a ser: $3 \rightarrow 1 \rightarrow 2$, con un peso = $m[3][1] + m[1][2] = 2 + 4 = 6$

$$m : \begin{bmatrix} 0 & 4 & \infty & 5 & \infty \\ \infty & 0 & 1 & \infty & 6 \\ \textcolor{blue}{2} & \textcolor{red}{6} & 0 & 3 & \infty \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & \infty & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & -1 & 4 & -1 \\ -1 & 2 & 3 & -1 & 5 \\ \textcolor{blue}{1} & \textcolor{red}{1} & 3 & 4 & -1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 3: Ha finalizado la iteración $k = 1$

$$m : \begin{bmatrix} 0 & 4 & \infty & 5 & \infty \\ \infty & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & \infty \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & \infty & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & -1 & 4 & -1 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & -1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & -1 & 4 & 5 \end{bmatrix}$$

Figura 7.4: Pasos 1-3 explicados

Paso nº 4: Actualmente no existe camino del vértice 1 al vértice 3 ya que $m[1][3] = \infty$, mientras que tomando como vértices intermedios al conjunto $[1, 2]$, el camino pasa a ser: $1 \rightarrow 2 \rightarrow 3$, con un peso = $m[1][2] + m[2][3] = 4 + 1 = 5$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & \infty \\ \infty & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & \infty \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & \infty & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & \textcolor{blue}{2} & \textcolor{blue}{2} & 4 & -1 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & -1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 5: Actualmente no existe camino del vértice 1 al vértice 5 ya que $m[1][5] = \infty$, mientras que tomando como vértices intermedios al conjunto $[1, 2]$, el camino pasa a ser: $1 \rightarrow 2 \rightarrow 5$, con un peso = $m[1][2] + m[2][5] = 4 + 6 = 10$

$$m : \begin{bmatrix} 0 & 4 & \infty & 5 & \textcolor{red}{10} \\ \infty & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & \infty \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & \infty & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & \textcolor{blue}{2} & -1 & 4 & \textcolor{red}{2} \\ -1 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & -1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 6: Actualmente no existe camino del vértice 3 al vértice 5 ya que $m[3][5] = \infty$, mientras que tomando como vértices intermedios al conjunto $[1, 2]$, el camino pasa a ser: $3 \rightarrow 1 \rightarrow 2 \rightarrow 5$, con un peso = $m[3][2] + m[2][5] = 6 + 6 = 12$

$$m : \begin{bmatrix} 0 & 4 & \infty & 5 & \infty \\ \infty & 0 & 1 & \infty & \textcolor{blue}{6} \\ 2 & \textcolor{blue}{6} & 0 & 3 & \textcolor{red}{12} \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & \infty & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & -1 & 4 & -1 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & \textcolor{blue}{1} & 3 & 4 & \textcolor{red}{1} \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & -1 & 4 & 5 \end{bmatrix}$$

Figura 7.5: Pasos 4-6 explicados

Capítulo 7. Resultados y conclusiones

Paso nº 7: Actualmente no existe camino del vértice 5 al vértice 3 ya que $m[5][3] = \inf$, mientras que tomando como vértices intermedios al conjunto $[1, 2]$, el camino pasa a ser: $5 \rightarrow 2 \rightarrow 3$, con un peso = $m[5][2] + m[2][3] = 2 + 1 = 3$

$$m : \begin{bmatrix} 0 & 4 & \infty & 5 & \infty \\ \infty & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & \infty \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & -1 & 4 & -1 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & -1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 8: Ha finalizado la iteración $k = 2$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ \infty & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & 12 \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 9: Actualmente no existe camino del vértice 2 al vértice 1 ya que $m[2][1] = \inf$, mientras que tomando como vértices intermedios al conjunto $[1, 2, 3]$, el camino pasa a ser: $2 \rightarrow 3 \rightarrow 1$, con un peso = $m[2][3] + m[3][1] = 1 + 2 = 3$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ 3 & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & 12 \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Figura 7.6: Pasos 7-9 explicados

Paso nº 10: Actualmente no existe camino del vértice 2 al vértice 4 ya que $m[2][4] = \inf$, mientras que tomando como vértices intermedios al conjunto $[1, 2, 3]$, el camino pasa a ser: $2 \rightarrow 3 \rightarrow 4$, con un peso = $m[2][3] + m[3][4] = 1 + 3 = 4$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ \infty & 0 & 1 & 4 & 6 \\ 2 & 6 & 0 & 3 & 12 \\ \infty & \infty & 1 & 0 & \infty \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ -1 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ -1 & -1 & 3 & 4 & -1 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 11: Actualmente no existe camino del vértice 4 al vértice 1 ya que $m[4][1] = \inf$, mientras que tomando como vértices intermedios al conjunto $[1, 2, 3]$, el camino pasa a ser: $4 \rightarrow 3 \rightarrow 1$, con un peso = $m[4][3] + m[3][1] = 1 + 2 = 3$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ \infty & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & 12 \\ 3 & \infty & 1 & 0 & \infty \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & -1 & 3 & 4 & -1 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 12: Actualmente no existe camino del vértice 4 al vértice 2 ya que $m[4][2] = \inf$, mientras que tomando como vértices intermedios al conjunto $[1, 2, 3]$, el camino pasa a ser: $4 \rightarrow 3 \rightarrow 1 \rightarrow 2$, con un peso = $m[4][3] + m[3][2] = 1 + 6 = 7$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ \infty & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & 12 \\ 7 & 1 & 0 & \infty & \infty \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ -1 & 3 & 3 & 4 & -1 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Figura 7.7: Pasos 10-12 explicados

7.1. Ejemplo 1

Paso nº 13: Actualmente no existe camino del vértice 4 al vértice 5 ya que $m[4][5] = \inf$, mientras que tomando como vértices intermedios al conjunto $[1, 2, 3]$, el camino pasa a ser: $4 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow 5$, con un peso = $m[4][3] + m[3][5] = 1 + 12 = 13$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ \infty & 0 & 1 & \infty & 6 \\ 2 & 6 & 0 & 3 & 12 \\ \infty & \infty & 1 & 0 & 13 \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ -1 & 2 & 3 & -1 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ -1 & -1 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 14: Ha finalizado la iteración $k = 3$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ 3 & 0 & 1 & 4 & 6 \\ 2 & 6 & 0 & 3 & 12 \\ 3 & 7 & 1 & 0 & 13 \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 15: Ha finalizado la iteración $k = 4$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ 3 & 0 & 1 & 4 & 6 \\ 2 & 6 & 0 & 3 & 12 \\ 3 & 7 & 1 & 0 & 13 \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Figura 7.8: Pasos 13-15 explicados

Paso nº 16: Ha finalizado la iteración $k = 5$

$$m : \begin{bmatrix} 0 & 4 & 5 & 5 & 10 \\ 3 & 0 & 1 & 4 & 6 \\ 2 & 6 & 0 & 3 & 12 \\ 3 & 7 & 1 & 0 & 13 \\ 1 & 2 & 3 & 4 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Figura 7.9: Paso 16 explicado

Capítulo 7. Resultados y conclusiones

7.1.2. Camino 1

Llamada a la función construir_camino("[[1,2,2,4,2],[3,2,3,3,5],[1,1,3,4,1],[3,3,3,4,3],[1,2,2,4,5]]",3,5)

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:

```
[[1,2,2,4,2],[3,2,3,3,5],[1,1,3,4,1],[3,3,3,4,3],[1,2,2,4,5]]
```

Introducir matriz por celdas:

Filas: 5 Columns: 5

$$\begin{pmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{pmatrix}$$

Introducir vértice de salida:

```
3
```

Introducir vértice de llegada:

```
5
```

Figura 7.10: Introducción del input

Solución

El camino es:

Forma lineal:

```
[3,1,2,5]
```

MostrarPasos

Figura 7.11: Solución del problema

7.1. Ejemplo 1

Pasos

Paso nº 1: El nodo actual es 3. Para llegar al nodo 5 hay que ir al nodo $s[3][5] = 1$

$$s = \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & \textcolor{red}{1} \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 2: El nodo actual es 1. Para llegar al nodo 5 hay que ir al nodo $s[1][5] = 2$

$$s = \begin{bmatrix} 1 & 2 & 2 & 4 & \textcolor{red}{2} \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 3: El nodo actual es 2. Para llegar al nodo 5 hay que ir al nodo $s[2][5] = 5$

$$s = \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & \textcolor{red}{5} \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{bmatrix}$$

Paso nº 4: Ya se ha llegado al nodo 5

$$s = \begin{bmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & \textcolor{red}{5} \end{bmatrix}$$

Figura 7.12: Pasos explicados

Capítulo 7. Resultados y conclusiones

7.1.3. Camino 2

Llamada a la función construir_camino("[[1,2,2,4,2],[3,2,3,3,5],[1,1,3,4,1],[3,3,3,4,3],[1,2,2,4,5]]",3,3)

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:
[[1,2,2,4,2],[3,2,3,3,5],[1,1,3,4,1],[3,3,3,4,3],[1,2,2,4,5]]]

Introducir matriz por celdas:
Filas: 5 Columns: 5

$$\begin{pmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{pmatrix}$$

Introducir vértice de salida:
3

Introducir vértice de llegada:
3

Importar matriz de CSV

Calcular

Los nodos de salida y llegada son el mismo

Figura 7.13: Input introducido y error: mismos vértices de salida y llegada

7.1. Ejemplo 1

7.1.4. Camino 3

Llamada a la función construir_camino("[[1,2,2,4,2],[3,2,3,3,5],[1,1,3,4,1],[3,3,3,4,3],[1,2,2,4,5]]",6,3)

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:
[[1,2,2,4,2],[3,2,3,3,5],[1,1,3,4,1],[3,3,3,4,3],[1,2,2,4,5]]]

Introducir matriz por celdas:
Filas: 5 Columns: 5

$$\begin{pmatrix} 1 & 2 & 2 & 4 & 2 \\ 3 & 2 & 3 & 3 & 5 \\ 1 & 1 & 3 & 4 & 1 \\ 3 & 3 & 3 & 4 & 3 \\ 1 & 2 & 2 & 4 & 5 \end{pmatrix}$$

Introducir vértice de salida:
6

Introducir vértice de llegada:
3

Importar matriz de CSV

Calcular

Nodo de salida invalido

Figura 7.14: Input introducido y error: vértice de salida inválido

Capítulo 7. Resultados y conclusiones

7.2. Ejemplo 2: Grafo no conexo

7.2.1. Algoritmo

Llamada a la función floyd_warshall("[[0,1,2,inf,inf],[2,0,1,inf,inf],[1,inf,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]")

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:

```
[[0,1,2,inf,inf],[2,0,1,inf,inf],[1,inf,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]
```

Introducir matriz por celdas:

Filas: 5 Columnas: 5

$$\begin{pmatrix} 0 & 1 & 2 & \text{inf} & \text{inf} \\ 2 & 0 & 1 & \text{inf} & \text{inf} \\ 1 & \text{inf} & 0 & \text{inf} & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & 0 & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & -1 & 0 \end{pmatrix}$$

Importar matriz de CSV
Calcular

Figura 7.15: Introducción del input

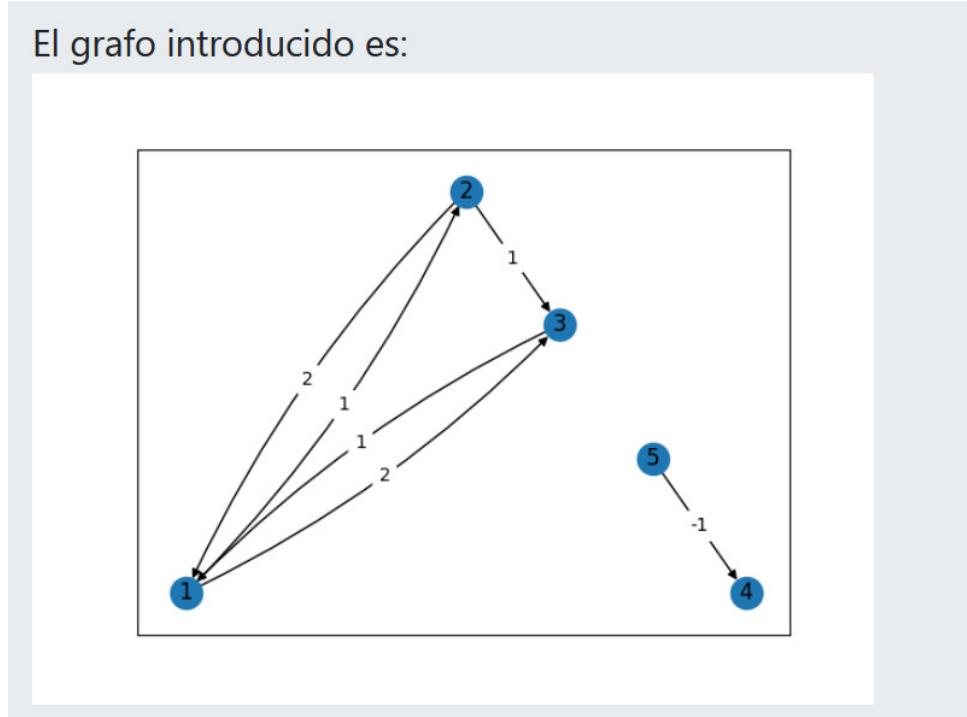


Figura 7.16: Imagen generada del grafo introducido

7.2. Ejemplo 2: Grafo no conexo

Solución:

Las matrices resultantes son:

Forma lineal de matriz de pesos:

```
[[0,1,2,inf,inf],[2,0,1,inf,inf],[1,2,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]
```

Forma LaTeX de matriz de pesos:

$$\begin{bmatrix} 0 & 1 & 2 & \infty & \infty \\ 2 & 0 & 1 & \infty & \infty \\ 1 & 2 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & -1 & 0 \end{bmatrix}$$

Forma lineal de matriz de sucesores:

```
[[1,2,3,-1,-1],[1,2,3,-1,-1],[1,1,3,-1,-1],[-1,-1,-1,4,-1],[-1,-1,-1,4,5]]
```

Forma LaTeX de matriz de sucesores:

$$\begin{bmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ 1 & 1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{bmatrix}$$

[MostrarPasos](#)

Figura 7.17: Solución del problema

Pasos

Paso nº 1: Ha finalizado la iteración $k = 0$

$$m : \begin{bmatrix} 0 & 1 & 2 & \infty & \infty \\ 2 & 0 & 1 & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & -1 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ 1 & -1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 2: Actualmente no existe camino del vértice 3 al vértice 2 ya que $m[3][2] = \text{inf}$, mientras que tomando como vértices intermedios al conjunto [1], el camino pasa a ser: 3 \rightarrow 1 \rightarrow 2, con un peso = $m[3][1] + m[1][2] = 1 + 1 = 2$

$$m : \begin{bmatrix} 0 & \color{blue}{1} & 2 & \infty & \infty \\ 2 & 0 & 1 & \infty & \infty \\ \color{blue}{1} & \color{red}{2} & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & -1 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ \color{blue}{1} & \color{red}{1} & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 3: Ha finalizado la iteración $k = 1$

$$m : \begin{bmatrix} 0 & 1 & 2 & \infty & \infty \\ 2 & 0 & 1 & \infty & \infty \\ 1 & 2 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & -1 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ 1 & 1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{bmatrix}$$

Figura 7.18: Pasos 1-3 explicados

Capítulo 7. Resultados y conclusiones

Paso nº 4: Ha finalizado la iteración k = 2

$$m : \begin{bmatrix} 0 & 1 & 2 & \infty & \infty \\ 2 & 0 & 1 & \infty & \infty \\ 1 & 2 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & -1 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ 1 & 1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 5: Ha finalizado la iteración k = 3

$$m : \begin{bmatrix} 0 & 1 & 2 & \infty & \infty \\ 2 & 0 & 1 & \infty & \infty \\ 1 & 2 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & -1 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ 1 & 1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 6: Ha finalizado la iteración k = 4

$$m : \begin{bmatrix} 0 & 1 & 2 & \infty & \infty \\ 2 & 0 & 1 & \infty & \infty \\ 1 & 2 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & -1 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ 1 & 1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{bmatrix}$$

Paso nº 7: Ha finalizado la iteración k = 5

$$m : \begin{bmatrix} 0 & 1 & 2 & \infty & \infty \\ 2 & 0 & 1 & \infty & \infty \\ 1 & 2 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & -1 & 0 \end{bmatrix}, s : \begin{bmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ 1 & 1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{bmatrix}$$

Figura 7.19: Pasos 4-7 explicados

7.2. Ejemplo 2: Grafo no conexo

7.2.2. Camino entre distintas componentes conexas

Llamada a la función construir_camino("[[1,2,3,-1,-1],[1,2,3,-1,-1],[1,1,3,-1,-1],[-1,-1,-1,4,-1],[-1,-1,-1,4,5]",4,3)

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:

```
[[1,2,3,-1,-1],[1,2,3,-1,-1],[1,1,3,-1,-1],[-1,-1,-1,4,-1],[-1,-1,-1,4,5]]
```

Introducir matriz por celdas:

Filas: 5 Columns: 5

$$\begin{pmatrix} 1 & 2 & 3 & -1 & -1 \\ 1 & 2 & 3 & -1 & -1 \\ 1 & 1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{pmatrix}$$

Introducir vértice de salida:

```
4
```

Introducir vértice de llegada:

```
3
```

Importar matriz de CSV

Calcular

No se puede llegar del nodo 4 al nodo 3

Figura 7.20: Input introducido y error: camino inexistente

Capítulo 7. Resultados y conclusiones

7.3. Varios ejemplos de entradas inválidas

7.3.1. Ejemplo 1

Llamada a la función floyd_warshall("[[0,1,2,inf,inf],[-3,0,1,inf,inf],[1,inf,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]")

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:

```
[[0,1,2,inf,inf],[-3,0,1,inf,inf],[1,inf,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]
```

Introducir matriz por celdas:

Filas: 5 Columns: 5

$$\begin{pmatrix} 0 & 1 & 2 & \text{inf} & \text{inf} \\ -3 & 0 & 1 & \text{inf} & \text{inf} \\ 1 & \text{inf} & 0 & \text{inf} & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & 0 & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & -1 & 0 \end{pmatrix}$$

Importar matriz de CSV

Calcular

La matriz de adyacencia no puede contener ciclos negativos: El ciclo 2 - 1 - 2 tiene peso = -2

Figura 7.21: Input introducido y error: grafo con ciclo de peso negativo

7.3. Varios ejemplos de entradas inválidas

7.3.2. Ejemplo 2

Llamada a la función floyd_marshall("[[0,1,2,inf,inf,0],[-3,0,1,inf,inf,0],[1,inf,0,inf,inf,0],[inf,inf,inf,0,inf,0],[inf,inf,inf,-1,0,0]]")

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:

```
[[0,1,2,inf,inf,0],[-3,0,1,inf,inf,0],[1,inf,0,inf,inf,0],[inf,inf,inf,0,inf,0],[inf,inf,inf,-1,0,0]]
```

Introducir matriz por celdas:

Filas: 5 Columns: 6

$$\begin{pmatrix} 0 & 1 & 2 & \text{inf} & \text{inf} & 0 \\ -3 & 0 & 1 & \text{inf} & \text{inf} & 0 \\ 1 & \text{inf} & 0 & \text{inf} & \text{inf} & 0 \\ \text{inf} & \text{inf} & \text{inf} & 0 & \text{inf} & 0 \\ \text{inf} & \text{inf} & \text{inf} & -1 & 0 & 0 \end{pmatrix}$$

Importar matriz de CSV

Calcular

La matriz de adyacencia ha de ser cuadrada

Figura 7.22: Input introducido y error: matriz de adyacencia no cuadrada

Capítulo 7. Resultados y conclusiones

7.3.3. Ejemplo 3

Llamada a la función floyd_warshall("[[1,1,2,inf,inf],[-3,0,1,inf,inf],[1,inf,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]")

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:

```
[[1,1,2,inf,inf],[-3,0,1,inf,inf],[1,inf,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]
```

Introducir matriz por celdas:

Filas: 5 Columns: 5

$$\begin{pmatrix} 1 & 1 & 2 & \text{inf} & \text{inf} \\ -3 & 0 & 1 & \text{inf} & \text{inf} \\ 1 & \text{inf} & 0 & \text{inf} & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & 0 & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & -1 & 0 \end{pmatrix}$$

Importar matriz de CSV

Calcular

La matriz de adyacencia no puede contener bucles

Figura 7.23: Input introducido y error: matriz de adyacencia con bucle

7.3. Varios ejemplos de entradas inválidas

7.3.4. Ejemplo 4

Llamada a la función floyd_marshall("[[abc,1,2,inf,inf],[-3,0,1,inf,inf],[1,inf,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]")

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:

```
[[abc,1,2,inf,inf],[-3,0,1,inf,inf],[1,inf,0,inf,inf],[inf,inf,inf,0,inf],[inf,inf,inf,-1,0]]
```

Introducir matriz por celdas:

Filas: 5 Columns: 5

$$\begin{pmatrix} \text{abc} & 1 & 2 & \text{inf} & \text{inf} \\ -3 & 0 & 1 & \text{inf} & \text{inf} \\ 1 & \text{inf} & 0 & \text{inf} & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & 0 & \text{inf} \\ \text{inf} & \text{inf} & \text{inf} & -1 & 0 \end{pmatrix}$$

Importar matriz de CSV
Calcular

La matriz de adyacencia es invalida

Figura 7.24: Input introducido y error: matriz de adyacencia con valor inválido

Capítulo 7. Resultados y conclusiones

7.3.5. Ejemplo 5

Llamada a la función construir_camino("[[1,2,2,-1,-1],[2,2,1,-1,-1],[1,1,3,-1,-1],[-1,-1,-1,4,-1],[-1,-1,-1,4,5]",1,3)

El error se encuentra en que para ir del vértice 1 al 3 hay que ir al 2, mientras que para ir del 2 al 3 hay que ir al 1, lo que genera un bucle infinito.

Algoritmo de Floyd-Warshall
 Construir camino

Introducir matriz forma lineal:
[[1,2,2,-1,-1],[2,2,1,-1,-1],[1,1,3,-1,-1],[-1,-1,-1,4,-1],[-1,-1,-1,4,5]]

Introducir matriz por celdas:
Filas: 5 Columns: 5

$$\begin{pmatrix} 1 & 2 & 2 & -1 & -1 \\ 2 & 2 & 1 & -1 & -1 \\ 1 & 1 & 3 & -1 & -1 \\ -1 & -1 & -1 & 4 & -1 \\ -1 & -1 & -1 & 4 & 5 \end{pmatrix}$$

Introducir vértice de salida:
1

Introducir vértice de llegada:
3

Importar matriz de CSV

Calcular

No se puede llegar del nodo 1 al nodo 3

Figura 7.25: Input introducido y error: matriz de sucesores con bucle infinito

7.4. Conclusión

Como conclusión final, se han conseguido cumplir todos los objetivos planteados para la elaboración de este trabajo: el desarrollo del algoritmo extendido de Floyd-Warshall para calcular distancias y caminos, el desarrollo de un sistema de ayuda mostrando los pasos del algoritmo, y la incorporación del código al *Proyecto Calculadora* para que pueda ser accesible por todos aquellos alumnos que deseen hacer uso de él para comprobar las soluciones de los ejercicios realizados durante la preparación para los exámenes.

En cuanto a posibles mejoras, cabe destacar que la librería empleada para la visualización de los grafos no permite la visualización de grafos con aristas bidireccionales, para lo cual se ha hecho uso de un código adicional para resolver este problema. Sin embargo, en algunas ocasiones, esto lleva a que la visualización del grafo no sea la más adecuada debido a la forma de posicionar los vértices. Este es un aspecto que podría ser mejorado en futuras versiones del proyecto mediante la exploración de alternativas más robustas para la visualización gráfica de los grafos.

Capítulo 8

Análisis de impacto

En este capítulo se realizará un análisis del impacto potencial de los resultados obtenidos durante la realización de este Trabajo de Fin de Grado.

A nivel personal, se ha obtenido un mayor aprendizaje del lenguaje de programación Python, el cual no se enseña a lo largo de la carrera. También se ha conseguido una mejor comprensión sobre la elaboración de APIs, así como de la comunicación entre front-end y back-end con JSON para el desarrollo de páginas web. Además, he aprendido sobre el algoritmo de Floyd-Warshall, el cual no fue explicado en clase durante la carrera.

A nivel académico, se espera que el desarrollo de este trabajo pueda servir de ayuda a los estudiantes a comprender el algoritmo y los pasos que hay que tomar para llegar a la solución, así como a comprobar sus soluciones y poder ver dónde se han equivocado, no únicamente si su resultado es el correcto o no.

Relacionando este impacto con los objetivos de desarrollo sostenible elaborados por la ONU [8], los cuales son 17 objetivos creados para promover la prosperidad a la vez que se protege el planeta, se tiene que el trabajo se enmarca principalmente en el objetivo 4: Garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje.

Bibliografía

- [1] E. Sharma. «Subpath of shortest path is shortest path». (2019), dirección: <https://sharmaeklavya2.github.io/theoremdep/nodes/graph-theory/shortest-paths/shortest-subpath.html>.
- [2] T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, *Introduction to algorithms, 4th edition*. The MIT Press, 2022.
- [3] «Floyd-Warshall algorithm». (2024), dirección: https://en.wikipedia.org/wiki/Floyd%20Warshall_algorithm.
- [4] «Top 20 Most Popular Programming Languages in 2024 & Beyond». (2024), dirección: <https://www.orientsoftware.com/blog/most-popular-programming-languages/>.
- [5] «Qué es Flask». (2024), dirección: <https://openwebinars.net/blog/que-es-flask/>.
- [6] «Angular». (2024), dirección: <https://angular.io/>.
- [7] «Drawing multiple edges between two nodes with networkx». (2024), dirección: <https://stackoverflow.com/questions/22785849/drawing-multiple-edges-between-two-nodes-with-networkx/>.
- [8] «Objetivos y metas de desarrollo sostenible». (2024), dirección: <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>.

Anexos

Capítulo A. Informe de originalidad

Apéndice A

Informe de originalidad



Recibo digital

Este recibo confirma que su trabajo ha sido recibido por Turnitin. A continuación podrá ver la información del recibo con respecto a su entrega.

La primera página de tus entregas se muestra abajo.

Autor de la entrega: MARIUS ROBERT DRAGHICI
Título del ejercicio: Turnitin Memoria Final (Moodle PP)
Título de la entrega: TFG Marius Robert Draghici.pdf
Nombre del archivo: 33259_MARIUS_ROBERT_DRAGHICI_TFG_Marius_Robert_Drag...
Tamaño del archivo: 4.58M
Total páginas: 69
Total de palabras: 8,759
Total de caracteres: 42,432
Fecha de entrega: 01-jun.-2024 07:25p. m. (UTC+0200)
Identificador de la entre... 2393189825



Este documento esta firmado por

	Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Sun Jun 02 20:20:06 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)