

**Estructura de datos**

ADA7 - Algoritmos Grafos

3SB

Briceño Mukul Adriana Isabel- E23081253

Naranjo Couoh Valeria- E23081251

González López Roberto Daniel - E23081042

Sansores Negron German Gael - E23081296

Sosa Chan Gadiel Antonio - E23081226

Ing. Armando López Valadez

31 de octubre de 2024.

Índice

[Repositorio Github 2](#_Toc181301943)

[Algoritmo Dijkstra 3](#_Toc181301944)

[Algoritmo en Python 3](#_Toc181301945)

[¿Cómo funciona? 4](#_Toc181301946)

[Algoritmo Kruskal 5](#_Toc181301947)

[Algoritmo en Python 6](#_Toc181301948)

[¿Cómo funciona? 7](#_Toc181301949)

[Algoritmo Floyd 8](#_Toc181301950)

[Algoritmo en Python 9](#_Toc181301951)

[¿Cómo funciona? 9](#_Toc181301952)

[Algoritmo Warshall 11](#_Toc181301953)

[Algoritmo en Python 11](#_Toc181301954)

[¿Cómo funciona? 12](#_Toc181301955)

[Bibliografía 13](#_Toc181301956)

# Repositorio Github

# Algoritmo Dijkstra

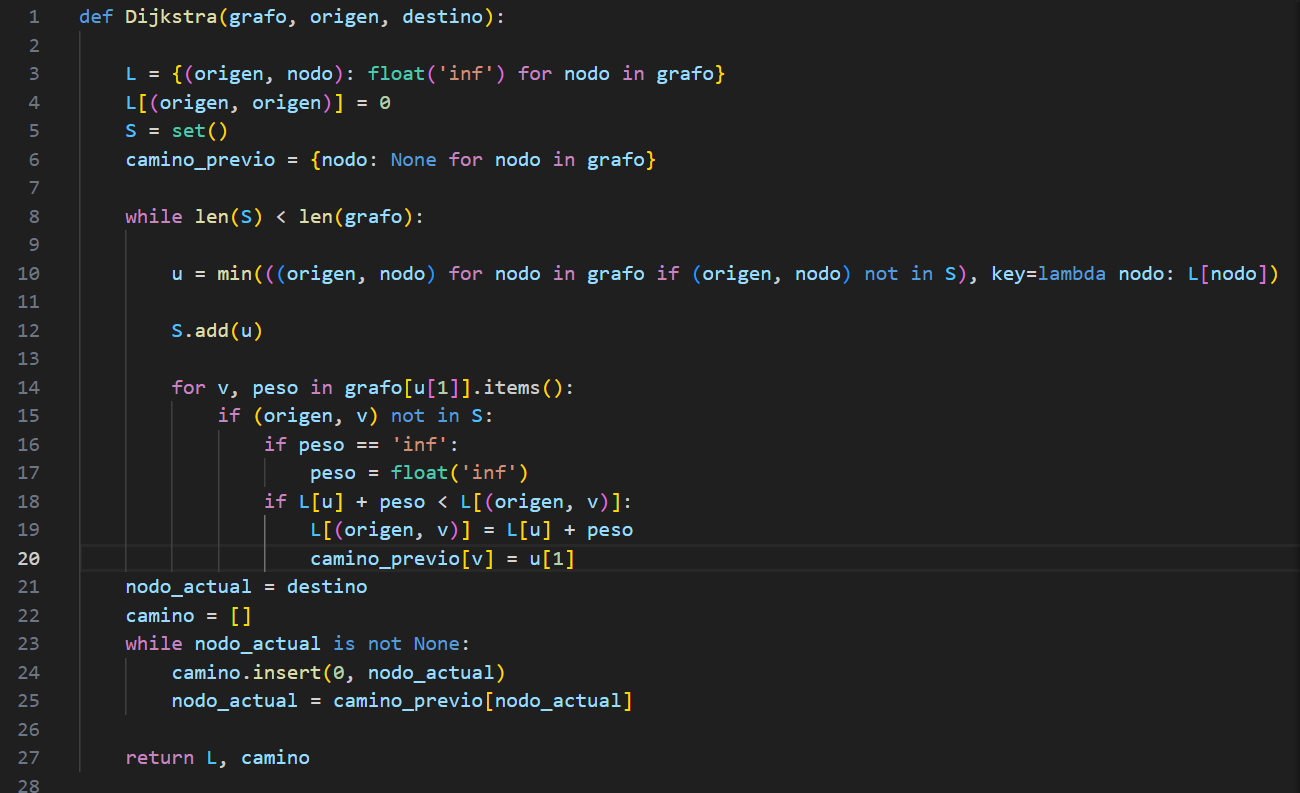
Al algoritmo de Dijkstra se lo conoce también como algoritmo de caminos mínimos. Fue creado e ideado por el científico computacional holandés Edsger Wybe Dijkstra, en 1959. Este algoritmo es utilizado para determinar el camino más corto para ejecutar desde un vértice origen hasta el resto de los vértices ubicados en un grafo con pesos en cada arista.

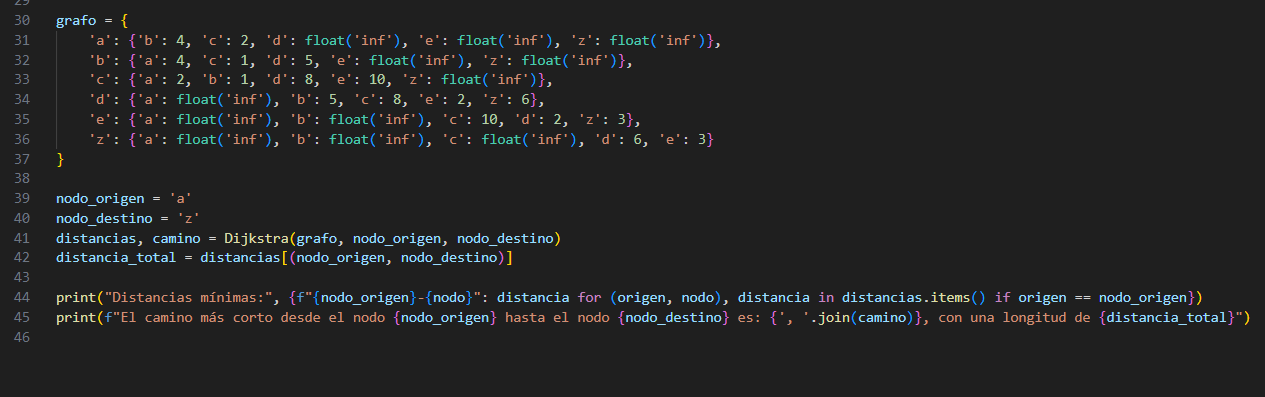
El algoritmo de Dijkstra puede tener diversas utilidades cuando es aplicada dentro de los grafos. Puede ser utilizado para:

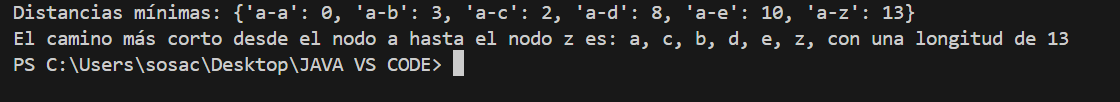
* La distribución de los productos a una o varias redes de establecimientos comerciales.
* Para los servicios de distribución de los correos postales.
* Para un grafo dirigido ponderado (G= (V, A)).

El problema del camino más corto de un vértice a otro consiste en determinar el camino de menor costo, desde un vértice u a otro vértice v. El costo de un camino es la suma de los costos (pesos) de los arcos que lo conforman.

## Algoritmo en Python







## ¿Cómo funciona?

En el código se define una función llamada Dijkstra, el cual recibe parámetros de grafo, origen y destino. Se inicializa el diccionario L que almacenará las distancias mínimas desde el nono origen a cada nodo del grafo, inicialmente todas las distancias (pesos) serán infinitos con excepción de la distancia del nodo origen que será 0.

s representa un conjunto que se usará para almacenar los nodos que ya han sido procesados.

Camino\_previo representa un diccionario que se usará para rastrear el nodo anterior en el camino más corto para cada nodo, lo que ayudará a marcar o generar el camino más corto más adelante.

Se usa un ciclo While para que se ejecute hasta que todos los nodos sean procesados. La idea es seleccionar a los nodos con la distancia menor que aún no ha sido procesado y así actualizar la distancia de sus nodos vecinos.

u representa el nodo que se selecciona para ser procesado. Se busca entre todos los nodos que no están en s (los que ya se procesaron) y se selecciona a quien tiene la distancia mínima. El método s.add(u) se encarga de añadir el nodo u al conjunto de nodos procesados. Se itera sobre los nodos vecinos al nodo u en la que para cada vecino v se verifica que aún no esté procesado, si el peso de la arista a v es infinito este se convierte en float(‘inf’). Se calcula la nueva distancia posible hacia v a través de u con L[u] + peso, si esta nueva distancia es menor que la distancia actualmente registrada en L[(origen, v)], se actualiza, es decir que en L[(origen, v)] se establece a la nueva distancia y en camino\_previo[v] se establece al nodo u, para poder reconstruir el camino más corto después.

Después de procesar todos los nodos, se reconstruye el camino más corto desde el nodo de destino hasta el nodo de origen: se empieza desde el nodo\_actual (el destino), se inserta cada nodo en el inicio de la lista camino hasta que nodo\_actual sea None (vacío), es decir llegar hasta el origen, todo esto con la ayuda de un ciclo While que se va a ejecutar siempre y cuando nodo\_actual no esté vacío.

Después de eso se hace uso de un return para devolver la distancia mínima (L) y el camino más corto (camino) que está dado en el arreglo camino = [].

Al final del código se define al grafo como un diccionario, se establece el nodo origen y el nodo destino. Se llama a la función Dijkstra para encontrar el camino más corto desde el nodo 'a' hasta 'z'. Luego se imprimen las distancias mínimas y el camino más corto.

La salida mostrará las distancias mínimas desde el nodo de origen hasta cada nodo y el camino más corto desde el origen al destino especificado, junto con la longitud total de ese camino.

# Algoritmo Kruskal

El algoritmo Kruskal es un algoritmo de la teoría de grafos descubierto por Joseph Kruskal que sirve para encontrar el árbol de peso mínimo dentro de un grafo conexo y ponderado G. El árbol que resulta del algoritmo Kruskal se conoce como árbol de costo total mínimo o Minimum Spanning Tree (MST) y no contiene ciclos.

Incluye todos los vértices y utiliza una cantidad de aristas n-1. Las aristas son ordenadas y agregadas sucesivamente de acuerdo con su peso siguiendo el orden de menor a mayor. Siempre selecciona el arco con menor coste para añadir al árbol; es un algoritmo voraz.

## Algoritmo en Python

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## ¿Cómo funciona?

Utilizando la estructura Union-Find, la cual nos permite verificar que dos vértices son conexos y unir componentes, declaramos dos atributos iterables que contienen los elementos del grafo. Esta estructura aloja dos métodos.

El método raíz() nos permite encontrar la raíz del conjunto en el que se encuentra “u”. Este método es recursivo. La raíz no es la misma “u”, se volverá a correr hasta encontrar los valores que coincidan.

El método unir() nos permite conectar dos conjuntos, el primero (raíz de “u”) con el segundo (raíz de “v”). Siempre que la raíz no sea la misma, comparará el tamaño del rango de ambas. La raíz del rango mayor será el padre del rango menor. En caso de ser iguales, la raíz de “v” se vuelve la raíz de “u” y al rango “u” se le suma uno.

Procedemos con la implementación del algoritmo de Kruskal. Declaramos un arreglo vacío que contendrá las aristas y usamos el diccionario del grafo ponderado no dirigido. En este diccionario se contendrán los nodos en las claves y los valores de peso en forma de lista. Utilizamos un par de ciclos for para convertir el grafo en una lista de aristas, las cuales son agregadas al arreglo siempre y cuando no se hayan agregado de forma previa. Las ordenamos según su peso, inicializamos la estructura usando el método key() de Python.

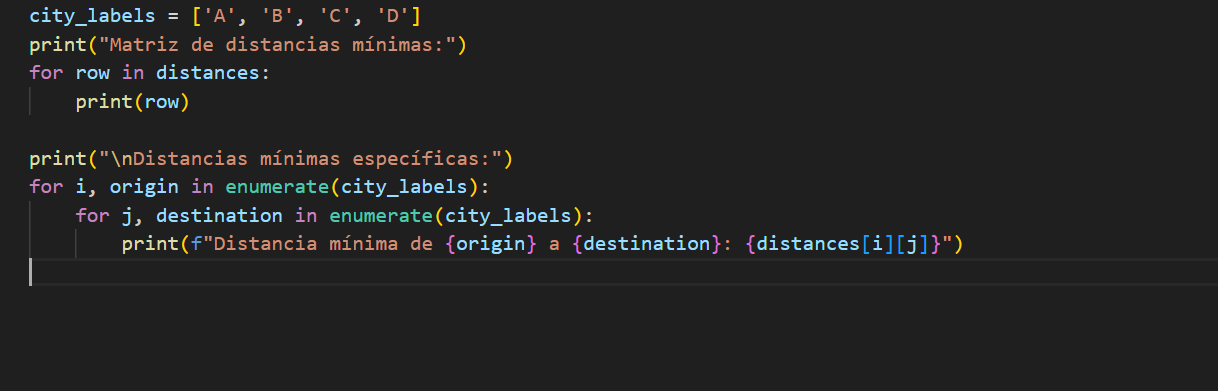
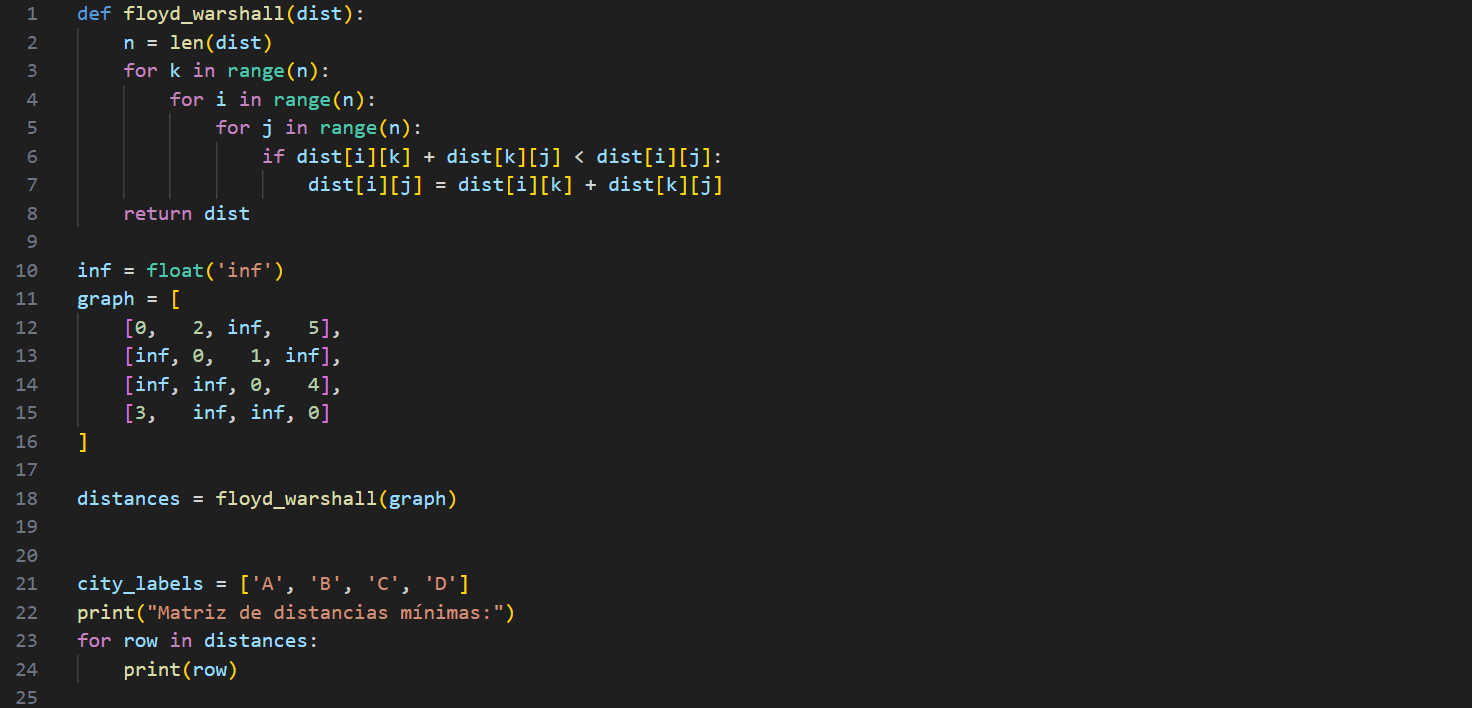
Después, declaramos el arreglo que alojará el MST. En un for que evaluará nuestra lista de aristas, comparamos que la raíz de “u” y la raíz de “v” no sean iguales. Si no lo son, se unirán “u” y “v” a la estructura UF y agregaremos ambos nodos y el peso de su arista al arreglo agm. Este arreglo es el return de nuestro método.

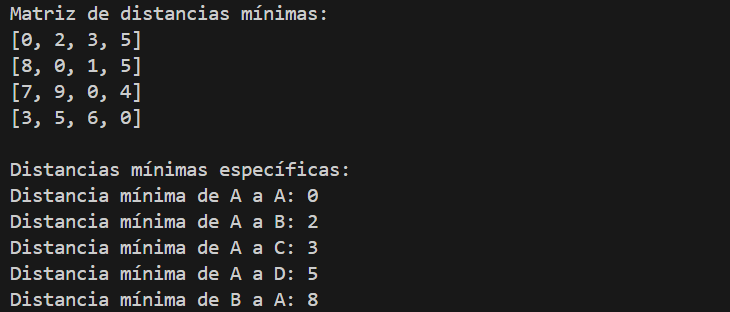
Se utiliza este algoritmo para evaluar el diccionario “Ciudades”. La salida nos muestra los nodos y las aristas más cortas entre ellos.

# Algoritmo Floyd

El Algoritmo de Floyd, también conocido como Algoritmo de Floyd-Warshall o Algoritmo de Roy-Floyd, es un enfoque popular para encontrar el camino más corto entre todos los pares de vértices de un grafo ponderado. Es una técnica de [programación dinámica](https://www.studysmarter.es/resumenes/matematicas/matematicas-de-la-decision/programacion-dinamica/) que maneja eficazmente los pesos negativos de las aristas y detecta los ciclos negativos. Inventado por Robert Floyd, un destacado informático, este algoritmo está diseñado para grafos representados mediante [matrices](https://www.studysmarter.es/resumenes/matematicas/matematicas-puras/matrices/) de adyacencia. El Algoritmo de Floyd desempeña un papel importante en diversos campos de la matemática de la decisión, que es una rama de la matemática aplicada que se ocupa de la construcción y el análisis de métodos para tomar decisiones informadas. Algunas de estas aplicaciones son el encaminamiento de redes, la [investigación operativa](https://www.studysmarter.es/resumenes/matematicas/matematicas-aplicadas/investigacion-operativa/), la teoría de juegos y los gráficos por ordenador.

## Algoritmo en Python





## ¿Cómo funciona?

Este código implementa el algoritmo de Floyd para encontrar la distancia mínima entre todos los pares de nodos en un grafo. Es especialmente útil en grafos ponderados, donde necesitamos conocer el camino más corto entre cada par de nodos.

La función floyd\_warshall, recibe como entrada dist, que es una matriz de adyacencia del grafo, cada elemento dist[i][j] representa la distancia directa entre el nodo i y el nodo j. Si no existe un camino directo, usamos inf (infinito) para indicar que no hay conexión directa.

Usamos graph para representar el grafo con 4 nodos (etiquetados de A a D), aquí, graph[i][j] es el peso o costo directo entre los nodos i y j. Si es inf, significa que no hay un camino directo entre esos nodos. Tiene tres bucles anidados, cada uno recorriendo los nodos del grafo, el k es el nodo intermedio que se considera para intentar reducir la distancia entre otros nodos, el i y j son los nodos de origen y destino.

if dist[i][k] + dist[k][j] < dist[i][j]:

dist[i][j] = dist[i][k] + dist[k][j]

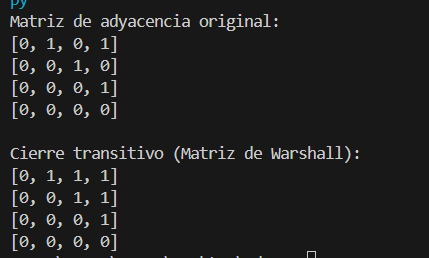
Si encontramos un camino más corto entre i y j pasando por k, actualizamos dist[i][j] con este valor menor y al final, dist contendrá las distancias mínimas entre todos los pares de nodos.

Se imprimen dos cosas la matriz de distancias mínimas que muestra todas las distancias mínimas entre cada par de nodos y las distancias mínimas específicas que presenta las distancias entre nodos etiquetados (de A a D) de una manera más descriptiva.

# Algoritmo Warshall

El algoritmo de Warshall de Stephen Warshall se utiliza para encontrar el cierre transitivo de una relación binaria. El cierre transitivo de una relación binaria es encontrar la relación binaria más pequeña que, siendo esta transitiva, contiene al conjunto de pares de la relación binaria original. Este algoritmo encuentra si es posible un camino entre cada uno de los vértices del grafo dirigido, es decir, no presenta las distancias entre los vértices. Se basa en un concepto llamado cerradura transitiva de la matriz de adyacencia. El algoritmo de Warshall sirve para encontrar la cerradura transitiva de una relación binaria en el conjunto A. La clausura transitiva de una relación binaria es la relación binaria más pequeña que, siendo transitiva, contenga el conjunto de pares de la relación binaria original.

## Algoritmo en Python



## ¿Cómo funciona?

Se define la función warshall que calcula el cierre transitivo de un grafo, lo que significa que encuentra todas las conexiones indirectas entre nodos. La función recibe como parámetro una matriz de adyacencia grafo, donde cada posición grafo[i][j] indica si existe una conexión directa entre el nodo i y el nodo j (representada con 1 para una conexión y 0 si no hay conexión).

Dentro de la función, se obtiene la cantidad de nodos n a partir de la longitud de la matriz de adyacencia. Luego, se crea una copia de la matriz original llamada cierre\_transitivo, que se construye copiando cada fila de grafo para evitar modificar original. Esta copia almacenará las actualizaciones realizadas durante el proceso y representará el cierre transitivo final.

El algoritmo utiliza tres bucles anidados. El primer bucle recorre cada nodo k como nodo intermediario entre los demás nodos i y j. El segundo y tercer bucle recorren cada par de nodos i y j. Para cada combinación de i, j, y k, se evalúa si existe un camino de i a j pasando por k. Si cierre\_transitivo[i][k] y cierre\_transitivo[k][j] son 1, esto significa que hay un camino indirecto de i a j a través de k, y se actualiza cierre\_transitivo[i][j] a 1 para reflejar esta nueva conexión.

Por último, se imprime la matriz de cierre transitivo resultante. Cada fila y columna de esta matriz indica si hay un camino (directo o indirecto) entre cada par de nodos.

# Bibliografía

Algoritmo Dijkstra. (s/f). Mycompiler.Io. Recuperado el 31 de octubre de 2024, de <https://www.mycompiler.io/view/2QMGtd9VNzs>

*algoritmo\_kruskal [Grafos - software para la construcción, edición y análisis de grafos.]*. (s. f.). <https://arodrigu.webs.upv.es/grafos/doku.php?id=algoritmo_kruskal>

*Árboles de peso mínimo: algoritmos de Prim y Kruskal — Matemáticas Discretas para Ciencia de Datos*. (s. f.). <https://madi.nekomath.com/P5/ArbolPesoMin.html>

EcuRed. (s. f.). *Algoritmo de Kruskal - ECUReD*. <https://www.ecured.cu/Algoritmo_de_Kruskal>

Ivaldi, T. (2022, junio 7). Lo que necesitas saber sobre el algoritmo de Dijkstra. TFG Online. <https://tfgonline.es/algoritmo-de-dijkstra/>

*Algoritmo de Floyd: ejemplos, aplicaciones | StudySmarter*. (s. f.). StudySmarter ES. <https://www.studysmarter.es/resumenes/matematicas/matematicas-de-la-decision/algoritmo-de-floyd/>

*You’re using an unsupported browser*. (s. f.). <https://www.mycompiler.io/unsupported-browser>

Algoritmo de Floyd-Warshall \_ AcademiaLab. (s. f.). <https://academia-lab.com/enciclopedia/algoritmo-de-floyd-warshall/>

Caminos más cortos de todos los pares: algoritmo de Floyd Warshall. (s. f.). <https://www.techiedelight.com/es/pairs-shortest-paths-floyd-warshall-algorithm/>