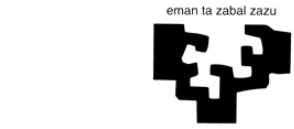


ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Estudio del Problema de la Ordenación Lineal y asociación a digrafos

Trabajo Fin de Grado
Grado en Matemáticas

Malena Urruchua Fernández

Trabajo dirigido por
Leticia Hernando Rodríguez

Leioa, 18 de Junio de 2021

Índice general

Introducción

Desde el comienzo de los tiempos los seres humanos hemos buscado sacar el máximo rendimiento a todo lo que nos rodea, es decir, optimizar los recursos a nuestra disposición. En nuestra vida cotidiana nos enfrentamos a problemas de optimización a diario: buscamos el camino más corto para ir de un lugar a otro, decidimos qué objeto u objetos constituyen la mejor compra, cuál es el mejor candidato para desempeñar cierta tarea... Normalmente, cuando una persona está ante este planteamiento no utiliza fórmulas ni recursos matemáticos para obtener la solución. Sin embargo, cuando tenemos problemas de optimización más complejos, la herramienta para resolverlos son las matemáticas. De manera general, podemos definir la optimización como la selección del mejor elemento (respecto de algún criterio previamente fijado) de entre una colección de elementos.

Durante siglos, grandes matemáticos se han enfrentado a estos problemas tanto en el espacio continuo como en el discreto, desarrollando diferentes métodos. Ya en el siglo III a.C Apolonio de Perge (geómetra griego) estudió las secciones cónicas a fin de obtener la proporción máxima. Pero no fue hasta el siglo XVII, con el desarrollo del cálculo diferencial, cuando la optimización continua dio un gran salto gracias a Newton, Leibnitz, Bernoulli y Lagrange. Un segundo gran hito en la historia de la optimización sucedió en la primera mitad del siglo XX con el desarrollo de la programación lineal por parte de George Dantzig, John Von Neumann y Leonid Kantoróvich. Aún siendo este último un gran avance para la optimización discreta o combinatoria, a día de hoy todavía quedan muchos problemas de optimización combinatoria por resolver de manera eficaz. En las últimas décadas, gracias al desarrollo de la tecnología y las herramientas informáticas, se ha avanzado mucho en este aspecto y se han propuesto nuevos y potentes métodos de resolución.

Algunos de los problemas de optimización combinatoria más relevantes son el problema del viajero/a [1], el problema de la mochila discreta [2], el problema de la ordenación lineal [3]... En todos estos problemas el espacio de búsqueda aumenta de forma exponencial a medida que aumenta el tamaño del problema. A día de hoy, no existe ningún método capaz de resolver de

manera eficaz todos los casos de alguno de estos problemas. De hecho, existe la conjetura de que estos problemas son irresolubles en un tiempo polinomial en la dimensión del problema, lo que se conoce como el problema P versus NP, uno de los problemas del Milenio[1].

De entre los problemas citados, es el de la ordenación lineal al que se dedicará este documento. Comenzaré desarrollando algunos conceptos generales de la optimización combinatoria, para después centrarme en el problema de la ordenación lineal desarrollando y demostrando algunas de sus propiedades más relevantes y proponiendo un nuevo método exacto de resolución nunca anteriormente desarrollado. Este nuevo método recorrerá un número de soluciones, evitando gran parte de ellas que se sabe de antemano que no serán el óptimo global. No obstante, el tiempo de ejecución del algoritmo propuesto aumenta factorialmente con el tamaño del problema, por tanto, el tiempo de ejecución será alto y por ello, solo será aplicable a instancias de tamaño pequeño. Este algoritmo nunca antes ha sido propuesto y toma como base algunas propiedades del problema de la ordenación lineal que nunca antes habían sido estudiadas.

[1] `https://www.claymath.org/millennium-problems/p-vs-np-problem`

Resumen y objetivos

Resumen

En el Capítulo 1 de este documento introducimos conceptos básicos de la optimización, en particular, de la optimización combinatoria. A continuación, desarrollamos algunos de los ejemplos más relevantes de este tipo de problemas, para, a partir del Capítulo 2, centrarnos en el problema de la ordenación lineal (*Linear Ordering Problem* - LOP). En primer lugar desarrollamos y demostramos algunas de sus propiedades más relevantes. En segundo lugar, nombramos algunas de sus aplicaciones más comunes en el mundo actual, centrándonos en la aplicación económica. En el Capítulo 3, se introduce el concepto de *Landscape* del LOP, y para ello, se definen los dos entornos o vecindarios más utilizados para este problema. Se analizarán las características de los *landscapes* bajo estos dos vecindarios, relacionándolas con alguna de las familias de números más conocidas en el campo de la Matemática Discreta. En el Capítulo 4, introducimos conceptos básicos de la Teoría de Grafos y describimos una novedosa asociación entre nuestro problema y los digrafos. Además, formulamos y demostramos un teorema que relaciona los caminos hamiltonianos del digrafo asociado con los óptimos locales de nuestro problema para uno de los vecindarios descritos anteriormente. En el Capítulo 5, valiéndonos de los resultados obtenidos en el capítulo anterior, proponemos un algoritmo de resolución para este problema que aplicaremos a un caso real. La asociación entre el problema LOP y los grafos dirigidos no ha sido realizada en ningún trabajo anterior de la literatura, y por tanto, supone una propuesta completamente nueva. El algoritmo desarrollado en este trabajo es, por tanto, un trabajo original y nunca antes planteado. En el Capítulo 6, presentamos las conclusiones y, a partir de ellas, exponemos el trabajo futuro que puede desarrollarse tomando como base este proyecto.

Objetivos

Estudiar el problema de optimización de la ordenación lineal y sus propiedades. Definir dos vecindarios y relacionarlos con grandes familias de números conocidas. Implementar un nuevo algoritmo que tiene en cuenta las propie-

dades descritas en relación a los vecindarios y se basa en la asociación entre el problema y los digrafos para resolver el problema. Por último, resolver un caso real utilizando el nuevo algoritmo implementado.

Capítulo 1

Problemas de optimización combinatoria

1.1. Descripción matemática y definiciones básicas

Intuitivamente, un problema de optimización consiste en elegir la mejor opción de entre un conjunto de posibles soluciones que cumplen determinadas propiedades.

En general, podemos definir formalmente un problema de optimización como [1]:

$$\begin{cases} Min\, f((x_1, x_2, ..., x_n)) \\ \\ \text{sujeto a:} \\ g_i((x_1, x_2, ..., x_n)) \leq 0,\ i = 1, ..., m \\ h_j((x_1, x_2, ..., x_n)) = 0,\ j = 1, ..., p \end{cases}$$

O análogamente :

$$\begin{cases} Max\, f((x_1, ..., x_n)) \\ \\ \text{sujeto a:} \\ g_i((x_1, x_2, ..., x_n)) \leq 0,\ i = 1, ..., m \\ h_j((x_1, x_2, ..., x_n)) = 0,\ j = 1, ..., p \end{cases}$$

donde: $f, g_i, h_j : S \longrightarrow \mathbb{R}$ son funciones de $(x_1, x_2, ..., x_n) \in S$. Dependiendo de la naturaleza de S tendremos diferentes tipos de problemas de optimización:

- Continuos: en estos problemas S es un espacio continuo, por lo general, $S \subset \mathbb{R}^n$.

- Discretos o combinatorios: en estos problemas S es un espacio discreto, es decir, un espacio finito o infinito numerable que puede estar formado por puntos de $\mathbb{N}^n$, vectores binarios, permutaciones...

En lo que sigue, vamos a tratar con problemas de optimización combinatoria. Para formalizar cualquier problema de optimización combinatoria es imprescindible establecer: la función a optimizar, también denominada función objetivo f, que se define en función de ciertos parámetros del problema; el dominio discreto de la función objetivo que denominamos espacio de búsqueda S; y las restricciones g_i y h_j, que deberán cumplir las variables $x_1, ..., x_n$, en caso de que el problema tenga restricciones. Decimos que tenemos una instancia o ejemplo de un problema de optimización combinatoria cuando estamos ante un caso particular del mismo. Es decir, la instancia es el caso concreto del problema una vez conocidos los datos (los valores que toman los parámetros de los que depende la función objetivo f). A continuación, vamos a establecer algunos conceptos derivados de los problemas de optimización combinatoria.

Definición 1.1.1. Un punto factible de un problema de optimización combinatoria, $(x_1, x_2, ..., x_n)$, es un elemento que pertenece al espacio de búsqueda S que cumple con las restricciones del problema g_i y h_j, donde $i = 1, ..., m$, $j = 1, ..., p$, siendo m el número de desigualdades y p el número de igualdades que deben cumplir las variables $x_1, ..., x_n$.

Buscar la solución al problema de optimización combinatoria consiste en encontrar el valor mínimo o máximo de la función $f : S \longrightarrow \mathbb{R}$. Ahora bien, en la mayoría de los casos esto no es lo único que nos interesa, ya que el punto factible $(x_1, x_2, ..., x_n) \in S$ donde se alcanza esta solución, puede ser tanto o más relevante.

Definición 1.1.2. Sea S el espacio de búsqueda y f la función objetivo de un problema de optimización combinatoria, denominaremos óptimo global al punto factible $(x_1, x_2, ..., x_n) \in S$ donde se alcanza el valor máximo o mínimo de la función f, dependiendo de si el problema trata de maximizar o minimizar, respectivamente.

Nota: el óptimo global puede ser múltiple, es decir, la solución puede alcanzarse en más de un punto factible, o por el contrario, puede que el óptimo global no se alcance en ningún momento, esto es, puede que no exista solución al problema.

Se detallan a continuación algunos ejemplos de problemas de optimización combinatoria más relevantes:

- El problema del viajero/a (*Traveling Salesperson Problem* - TSP) [1]: este problema consiste en recorrer n ciudades comenzando y finalizando en la misma ciudad haciendo que el coste del camino sea mínimo. Dependiendo de la instancia escogida, podemos asociar este coste con diferentes conceptos, siendo los más comunes: la distancia del camino o el tiempo empleado para recorrerlo.
  Para modelizarlo podemos usar diferentes representaciones del problema como por ejemplo: grafos o matrices. El más habitual es un grafo $G = (V, A)$ en el que cada vértice de $V = \{1, 2, ..., n\}$ representa una ciudad, mientras que cada una de las aristas de $A = \{(i, j)|i, j \in V \wedge i \neq j\}$ se identifica con los caminos de una ciudad i, a otra j, y tienen un peso asociado d_{ij} que se identifica con el coste de ir de la ciudad i a la j, con $i, j \in \{1, ..., n\}, i \neq j$. Estos costes asociados, d_{ij}, serán los parámetros bajo los que se define la función objetivo f.
  En este caso, un punto factible se puede representar mediante cualquier permutación $\sigma = (\sigma_1\sigma_2....\sigma_n) \in S_n$ de las n ciudades, en la que cada posición i de la permutación indica la ciudad que debemos visitar en i-ésimo lugar. Sin pérdida de generalidad, podemos fijar la ciudad de partida $\sigma_1 = k$ que es a la que tenemos que regresar tras haber visitado todas. Esta será la única restricción que exijamos para definir este problema, pero según la instancia escogida podríamos exigir más restricciones. Así, podemos definir el espacio de búsqueda S del siguiente modo:
  $S = \{\sigma = (k\sigma_2...\sigma_n) \in S_n\}$ donde S_n es el grupo de permutaciones de n elementos y definimos la función objetivo a minimizar como:

$$\begin{aligned} f : S_n &\longrightarrow \mathbb{R}^+ \\ \sigma &\longmapsto d_{k\sigma_2} + \sum_{j=2}^{n-1} d_{\sigma_j\sigma_{j+1}} + d_{\sigma_n k} \end{aligned}$$

  Aunque la solución a este problema consiste en encontrar el coste mínimo necesario para recorrer todas las ciudades empezando y finalizando en una dada, nos interesa saber en qué orden se han de recorrerse las ciudades para lograr este valor. Esto es, no sólo nos interesa el valor mínimo de f, sino el punto factible $\sigma \in S_n$ en el que se alcanza esta solución.

- El problema de la mochila discreta [2]: en este problema se tiene un conjunto de n objetos, $J = \{1, 2, ..., n\}$, donde cada objeto $j \in \{1, ..., n\}$ tiene un peso ω_j y un valor v_j asociados. Se trata de llenar una mochila de capacidad máxima W para obtener el valor máximo posible, teniendo en cuenta que no todos los objetos caben en la mochila. Esto es, la solución es el valor máximo que podemos llevar en la mochila. En este caso, cualquier combinación de $k < n$ objetos será un punto

factible para este problema siempre y cuando la suma del peso de los k objetos no sea mayor que W. Para representar un punto factible de este problema podríamos usar conjuntos de números enteros o vectores binarios. En este ejemplo, utilizaremos subconjuntos formados por los elementos de J que serán los que metamos en la mochila. Así, definimos S y f del siguiente modo:
$S = \{\{j_1, j_2, ..., j_k\} \subset J | \sum_{i=1}^{k} \omega_{j_i} \leq W\}$,

$$f : S \longrightarrow \mathbb{R}^+$$
$$\{j_1, j_2, ..., j_k\} \longmapsto \sum_{i=1}^{k} v_{j_i}$$

Maximizando f se obtiene el valor máximo que se puede tener en la mochila. No obstante, también interesa saber qué combinación de estos objetos dan dicha solución, esto es, qué subconjunto de J se ha elegido.

- El problema del árbol recubridor mínimo (*Minimal Spanning Tree* - MST) [2]: dado un grafo (V, A), donde $V = \{1, ..., n\}$ representa un conjunto de vértices y $A = \{\{i, j\} | i, j = 1, ..., n\}$ un conjunto de aristas con un valor asociado a_{ij}, se busca encontrar el subgrafo conectado y sin ciclos que contiene todos los elementos de V y un subconjunto de A cuyo valor asociado sea mínimo.
  Para este problema, un punto factible sería cualquier subgrafo compuesto por todos los vértices del grafo original y un subconjunto de k aristas, tales que formen un grafo conectado y sin ciclos (independientemente del coste asociado a sus aristas). Así que podemos definir S y f del siguiente modo:
  $S = \{A' \subset A \mid (V, A')$ es conexo y sin ciclos $\}$,

$$f : S \longrightarrow \mathbb{R}^+$$
$$A' \longmapsto \sum_{\{i,j\} \in A'} a_{ij}$$

  Calculando el mínimo de esta función f encontramos la solución, aunque también nos interesa saber cuáles son estas aristas escogidas.

- El problema de la ordenación lineal (*Linear ordering problem* - LOP): este problema consiste en determinar una permutación $\sigma \in S_n$ de filas y columnas, simultáneamente, de una matriz cuadrada $B \in Mat_{n \times n}(\mathbb{R})$ tal que, la suma de los elementos por encima de la diagonal sea máxima, o análogamente, que la suma de los elementos por debajo de la diagonal sea mínima.

Para este problema un punto factible sería cualquier permutación $\sigma \in S_n$, ya que no tiene ninguna restricción. Si buscamos maximizar podemos definir f del siguiente modo:

$$f : S_n \longmapsto \mathbb{R}$$
$$\sigma = (\sigma_1 ... \sigma_n) \longmapsto \sum_{i<j}^{n} b_{\sigma_i \sigma_j}$$

Para este último ejemplo la solución es el valor máximo de la suma de los elementos por encima de la diagonal de una matriz cuyas filas y columnas han sido permutadas con respecto a la matriz original. Pero en este caso, de nuevo, buscamos también el punto factible en el que se alcanza esta solución, esto es, también nos interesa conocer la permutación $\sigma \in S_n$ en la que se alcanza la solución.

En este trabajo nos centraremos en el problema de la ordenación lineal. En los próximos capítulos se dará una definición más detallada del problema, se desarrollarán y demostrarán algunas de las propiedades, se citarán sus aplicaciones en el mundo real y también propondremos un nuevo método exacto para su resolución basado en las características anteriormente observadas. Pero antes desarrollaremos dos importantes conceptos vinculados a los problemas de optimización combinatoria: métodos de resolución y complejidad temporal.

1.2. Complejidad y métodos de resolución

La complejidad de un problema y los métodos de resolución que se puedan aplicar de manera eficaz son conceptos que están fuertemente relacionados. Precisamente, la complejidad de un problema será la complejidad del método más eficaz que pueda ser utilizado para resolver todas sus instancias.

1.2.1. Métodos de resolución de los problemas de optimización combinatoria

Como hemos citado anteriormente, para conseguir la solución a un problema de optimización combinatoria tenemos que encontrar el máximo o mínimo de la función objetivo. Para ello, podemos valernos de diferentes métodos [4]:

- Métodos exactos: convergen al valor máximo/mínimo, es decir, al óptimo global. Los más relevantes son: el método simplex y el método Branch and Bound.

- Métodos heurísticos: no siempre obtienen el valor máximo/mínimo, esto es, no nos asegura alcanzar el óptimo global, pero a cambio, el tiempo de ejecución que requieren es, por lo general, menor que el de los métodos exactos. Existen métodos heurísticos de naturaleza muy diversos, pero pueden ser clasificados en dos categorías:
  - Métodos basados en una única solución [4]: partimos de una solución inicial y la vamos mejorando paso a paso. El algoritmo más habitual es: la búsqueda local.
  - Métodos basados en una población o un conjunto de soluciones [4]: partimos de un conjunto de soluciones que vamos modificando en cada iteración según la estrategia escogida. Algunos de los algoritmos más comunes son: los algoritmos evolutivos y los algoritmos *swarm intelligence*.

En todos los problemas de optimización combinatoria pueden ser aplicados tanto métodos exactos como métodos heurísticos, pero en numerosas ocasiones, los métodos exactos necesitan un tiempo de ejecución que aumenta de forma exponencial a medida que aumenta el tamaño del problema. Como consecuencia, no siempre es factible aplicarlos. Por el contrario, los métodos heurísticos, en general, necesitan un tiempo de ejecución menor. Como la optimalidad no está garantizada en este último caso, necesitamos medir la calidad de estos. Por este motivo, se busca que un heurístico tenga las siguientes propiedades [2]: eficiencia (necesita de un tiempo de ejecución computacional posible), bondad (solución cercana a la óptima) y robustez (pocas probabilidades de obtener una mala solución).

1.2.2. Complejidad de los problemas de optimización combinatoria

Habitualmente, se establece una clasificación de los problemas de optimización combinatoria en base a la complejidad que encuentran los métodos o algoritmos al tratar de resolverlos [4]:

- Decimos que un problema es de clase P si existe un algoritmo que resuelva todas sus instancias en un tiempo polinómico en la dimensión del problema. Un ejemplo de ello es la ordenación de números naturales.
- Decimos que un problema es NP si no se conoce un algoritmo que resuelva todas sus instancias en un tiempo polinómico en la dimensión del problema, pero dado un punto factible se puede comprobar si es una solución del problema en tiempo polinómico. Por ejemplo, la búsqueda de caminos hamiltonianos en un grafo es un problema NP.

- Decimos que un problema es NP-Completo si no se conoce ningún algoritmo que resuelva todas sus instancias en un tiempo polinómico en la dimensión del problema y tampoco sea posible comprobar en tiempo polinómico si un punto factible es solución al problema. Son ejemplos de ello el problema del viajero (TSP) o el problema de la ordenación lineal (LOP).

Todos los problemas de optimización combinatoria pueden ser abordados mediante algoritmos de tiempo polinómico. Esto no significa que obtengamos el óptimo global, o al menos, no nos lo aseguran. Sin embargo, se considera que un problema es P sólo si se puede asegurar que se obtiene el óptimo global en un tiempo polinómico en la dimensión del problema para todas sus instancias. Además, claramente podemos decir que P⊂NP, pues si podemos encontrar la solución a un problema en tiempo polinómico, dado un punto factible también podemos comprobar en tiempo polinómico si es solución al problema o no. Por el contrario, la inclusión inversa NP⊂P, a día de hoy, no se ha podido demostrar. Este es uno de los problemas del milenio [5] que aún no ha sido resuelto. Pero poder llegar a demostrar $P = NP$, supondría grandes cambios en el mundo de la computación, algunos podrían ser muy ventajosos, mientras que otros podrían desencadenar consecuencias nefastas. Tal y como se cita en [5]: "*poder resolver el problema del viajero mediante un algoritmo que necesite de un tiempo poliómico sería beneficioso para la industria, las comunicaciones y el desarrollo en general. En contraposición, las claves criptográficas se descifrarían con gran facilidad, y muchas cuentas bancarias y comunicaciones cifradas quedarían expuestas, siendo esto un gran peligro para nuestra privacidad*".

Capítulo 2

Problema de la ordenación lineal (LOP)

2.1. Definición

Dada una matriz $B = [b_{ij}]_{n\times n}$ con $b_{ij} \in \mathbb{R}$, el problema de la ordenación lineal consiste en encontrar la permutación $\sigma \in S_n$ (simultáneamente de filas y columnas) que haga que la suma de los valores que se encuentran por encima de la diagonal principal de la matriz resultante sea máxima.
Esto podemos expresarlo como [3]:

$$\begin{aligned} Máx \quad & f : S_n \longrightarrow \mathbb{R} \\ & \sigma \longmapsto f(\sigma) = \sum_{i<j}^{n} b_{ij}^{\sigma} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{ij}^{\sigma} \end{aligned} \tag{2.1}$$

donde S_n representa el grupo de permutaciones de n elementos y b_{ij}^{σ} representa el elemento de la fila i y columna j de la matriz obtenida al permutar B mediante la permutación σ. A esta matriz obtenida la denominaremos $B^{\sigma} \in Mat_{n\times n}(\mathbb{R})$.

Asimismo, podemos calcular el valor de f sin necesidad de construir la matriz $B^{\sigma} \in Mat_{n\times n}(\mathbb{R})$, pues podemos definir nuestro problema de este segundo modo:

$$\begin{aligned} Máx \quad & f : S_n \longrightarrow \mathbb{R} \\ & \sigma \longmapsto f(\sigma) = \sum_{i<j}^{n} b_{\sigma_i\sigma_j} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{\sigma_i\sigma_j} \end{aligned} \tag{2.2}$$

2.2. Espacio de Búsqueda

El espacio de búsqueda estará formado por todas las posibles permutaciones de tamaño n, siendo n el número de filas o columnas de la matriz, ya que el problema no tiene restricciones y por tanto, todas las permutaciones $\sigma \in S_n$ son puntos factibles. Por esto, usaremos para representar cada punto factible $\sigma = (123...k...n) \in S_n$. En cada permutación la posición i-ésima ocupada por el valor k representa la posición que debe ocupar la fila (y columna) k-ésima de la matriz original. Esto es, en la matriz solución, la fila (y columna) i-ésima será la k-ésima de la matriz original.

Ejemplo 2.2.1. Matriz original: B= $\begin{pmatrix} 0 & 1 & 3 & 19 & 2 \\ 3 & 0 & 11 & 15 & 4 \\ 2 & 9 & 0 & 16 & 6 \\ 1 & 18 & 2 & 0 & 5 \\ 15 & 19 & 3 & 7 & 0 \end{pmatrix}$

Como esta es la matriz original de nuestro ejemplo, su permutación asociada es aquella que deja cada fila y columna en la posición en la que estaba, es decir, $e = (12345)$, la canónica o identidad. En este caso el valor de la función objetivo mediante la Ecuación (2.1):

$$f(e) = \sum_{i<j}^{n} b_{ij}^{e} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{ij}^{e} = b_{12}^{e} + b_{13}^{e} + b_{14}^{e} + b_{15}^{e} + b_{23}^{e} + b_{24}^{e} + b_{25}^{e} + b_{34}^{e} + b_{35}^{e} + b_{45}^{e} =$$

$$= 1 + 3 + 19 + 2 + 11 + 15 + 4 + 16 + 6 + 5 = 82.$$

Vamos a tomar la permutación $\sigma = (23145)$, esto es la primera fila/ columna de la matriz que vamos a construir, B^{σ}, será la segunda de nuestra matriz original B y análogamente con el resto de filas/columnas. Construiremos esta matriz en dos pasos, primero modificando filas y después columnas:

$$\begin{matrix} & B & & & & & & & & B^{\sigma} \\ & \| & & & & & & & & \| \end{matrix}$$

$$\begin{pmatrix} 0 & 1 & 3 & 19 & 2 \\ 3 & 0 & 11 & 15 & 4 \\ 2 & 9 & 0 & 16 & 6 \\ 1 & 18 & 2 & 0 & 5 \\ 15 & 19 & 3 & 7 & 0 \end{pmatrix} \xrightarrow{filas} \begin{pmatrix} 3 & 0 & 11 & 15 & 4 \\ 2 & 9 & 0 & 16 & 6 \\ 0 & 1 & 3 & 19 & 2 \\ 1 & 18 & 2 & 0 & 5 \\ 15 & 19 & 3 & 7 & 0 \end{pmatrix} \xrightarrow{columnas} \begin{pmatrix} 0 & 11 & 3 & 15 & 4 \\ 9 & 0 & 2 & 16 & 6 \\ 1 & 3 & 0 & 19 & 2 \\ 18 & 2 & 1 & 0 & 5 \\ 19 & 3 & 15 & 7 & 0 \end{pmatrix}$$

Utilizando la Ecuación (2.1) calculamos el valor de la función objetivo correspondiente a esta permutación:

$$f(\sigma) = \sum_{i<j}^{n} b_{ij}^{\sigma} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} b_{ij}^{\sigma} = b_{12}^{\sigma} + b_{13}^{\sigma} + b_{14}^{\sigma} + b_{15}^{\sigma} + b_{23}^{\sigma} + b_{24}^{\sigma} + b_{25}^{\sigma} + b_{34}^{\sigma} + b_{35}^{\sigma} + b_{45}^{\sigma} =$$

$= 11 + 3 + 15 + 4 + 2 + 16 + 6 + 19 + 2 + 5 = 83.$

Como podemos ver, en este caso el valor obtenido es mayor.

Supongamos ahora que tomamos otra permutación, $\omega = (53421)$. En este caso vamos a calcular primero el valor de función, ya que, valiéndonos de la Ecuación (2.2), podemos usar la matriz original junto con la permutación para calcular este valor, sin necesidad de calcular B^ω. Pues bien, haciendo uso de la Ecuación (2.2) que hemos dado para f:

$$\begin{aligned} f(\omega) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} b_{\omega_i\omega_j} &= b_{\omega_1\omega_2} + b_{\omega_1\omega_3} + \\ &+ b_{\omega_1\omega_4} + b_{\omega_1\omega_5} + b_{\omega_2\omega_3} + b_{\omega_2\omega_4} + \\ &+ b_{\omega_2\omega_5} + b_{\omega_3\omega_4} + b_{\omega_3\omega_5} + b_{\omega_4\omega_5} = \\ &= b_{53} + b_{54} + b_{52} + b_{51} + b_{34} + b_{32} + \\ &+ b_{31} + b_{42} + b_{41} + b_{21} = 3 + 7 + 19 + \\ &+ 15 + 16 + 9 + 2 + 18 + 1 + 3 = 93 \end{aligned}$$

$$B^\omega = \begin{pmatrix} 0 & 3 & 7 & 19 & 15 \\ 6 & 0 & 16 & 9 & 2 \\ 5 & 2 & 0 & 18 & 1 \\ 4 & 11 & 15 & 0 & 3 \\ 2 & 3 & 19 & 1 & 0 \end{pmatrix}$$

Vemos claramente que los valores que hemos considerado al calcular el valor de función objetivo de la solución correspondiente a la permutación ω, coinciden con los valores que se encuentran por encima de la diagonal de la matriz B^ω.

2.3. Propiedades

El problema del LOP tiene algunas particularidades que analizaremos para su mejor comprensión y el avance en desarrollo de técnicas utilizadas en su resolución.
Sean $\sigma = (\sigma_1\sigma_2...\sigma_n) \in S_n$ y $B \in Mat_{n\times n}(\mathbb{R})$:

(i) Los valores que ocupan las posiciones $b_{ii}, i \in \{1, ..., n\}$, no aportan ningún valor a la función a optimizar, sea cual sea la permutación escogida [3].

Ejemplo 2.3.1. Vamos a ver esto en un ejemplo sencillo.

Tomamos una matriz B=$\begin{pmatrix} 10 & 1 & 2 \\ 3 & 10 & 4 \\ 5 & 6 & 10 \end{pmatrix}$.
Podemos calcular todas las posibles permutaciones y el valor de f en cada una de las permutaciones de $S_3 = \{\sigma^1, \sigma^2, \sigma^3, \sigma^4, \sigma^5, \sigma^6\} = \{(123), (132), (213), (312), (231), (321)\}$:

- $\sigma^1 = (123)$
  $B^{\sigma^1} = \begin{pmatrix} 10 & 1 & 2 \\ 3 & 10 & 4 \\ 5 & 6 & 10 \end{pmatrix}$
  $f(\sigma^1) = 1 + 2 + 4 = 7$
- $\sigma^2 = (132)$
  $B^{\sigma^2} = \begin{pmatrix} 10 & 2 & 1 \\ 5 & 10 & 6 \\ 3 & 4 & 10 \end{pmatrix}$
  $f(\sigma^2) = 1 + 2 + 6 = 9$
- $\sigma^3 = (213)$
  $B^{\sigma^3} = \begin{pmatrix} 10 & 3 & 4 \\ 1 & 10 & 2 \\ 6 & 5 & 10 \end{pmatrix}$
  $f(\sigma^3) = 3 + 4 + 2 = 9$
- $\sigma^4 = (312)$
  $B^{\sigma^4} = \begin{pmatrix} 10 & 5 & 6 \\ 2 & 10 & 1 \\ 4 & 3 & 10 \end{pmatrix}$
  $f(\sigma^4) = 5 + 6 + 1 = 12$
- $\sigma^5 = (231)$
  $B^{\sigma^5} = \begin{pmatrix} 10 & 4 & 3 \\ 6 & 10 & 5 \\ 1 & 2 & 10 \end{pmatrix}$
  $f(\sigma^5) = 4 + 3 + 5 = 12$
- $\sigma^6 = (321)$
  $B^{\sigma^6} = \begin{pmatrix} 10 & 6 & 5 \\ 4 & 10 & 3 \\ 2 & 1 & 10 \end{pmatrix}$
  $f(\sigma^6) = 6 + 5 + 3 = 14$

Como podemos ver, al permutar simultáneamente filas y columnas, los elementos de la diagonal principal se mantienen en la misma posición, y como en la función objetivo no entran los elementos de la diagonal principal, nunca se tendrán en cuenta.

Demostración 2.3.1. Por la Ecuación (2.2) sabemos que los valores de la matriz que toman parte en el cálculo de la función objetivo son de la forma $b_{\sigma_i\sigma_j}$ donde $i \in \{1, ..., n-1\}$ y $j \in \{i+1, ...n\}$.

Por reducción al absurdo, supongamos que $\exists l$ tal que: $\begin{cases} b_{ll} = b_{\sigma_i\sigma_j} \\ i \in \{1, ..., n-1\} \\ j \in \{i+1, ...n\} \end{cases}$

$$\iff \begin{cases} l = \sigma_i \wedge l = \sigma_j \\ i \in \{1, ..., n-1\} \\ j \in \{i+1, ...n\} \end{cases} \iff \begin{cases} \exists \sigma_i = \sigma_j \\ i \in \{1, ..., n-1\} \\ j \in \{i+1, ...n\} \end{cases} \iff$$

$$\iff \begin{cases} \exists j \in \{i_0+1, ..., n\} \\ \text{tal que: } \sigma_{i_0} = \sigma_j \end{cases} \iff \begin{cases} \exists j \in \{i_0+1, ..., n\} \\ \text{tal que : } i_0 = j \end{cases} \#$$

□

(ii) Las entradas de la matriz B se pueden organizar por parejas simétricamente localizadas respecto a la diagonal principal [3]. Esto es, cada entrada b_{ij} tiene su par asociado b_{ji}, donde $i, j \in \{1, 2, ...n\}$. Estas parejas permanecen asociadas, sea cual sea la permutación $\sigma \in S_n$.

Ejemplo 2.3.2. Si tomamos la matriz original del ejemplo anterior, tendremos las parejas:
$\{\{b_{12}, b_{21}\}, \{b_{13}, b_{31}\}, \{b_{23}, b_{32}\}\} = \{\{1,3\}, \{2,5\}, \{4,6\}\}$ Si tomamos la permutación $\sigma^3 = (213)$ del ejemplo anterior y la matriz B^{σ^3}, veremos que los pares asociados son claramente los mismos: {{3, 1},{2, 5}, {4, 6}} y esto ocurre para cualquiera de las permutaciones de S_3.

Demostración 2.3.2. Sean $\sigma = (\sigma_1\sigma_2...\sigma_{i-1}\sigma_i\sigma_{i+1}...\sigma_{j-1}\sigma_j\sigma_{j+1}...\sigma_n)$ y la matriz $B = \begin{pmatrix} b_{11} & ... & b_{1i-1} & b_{1i} & b_{1i+1} & ... & b_{1j-1} & b_{1j} & b_{1j+1} & ... & b_{1n} \\ \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots \\ b_{i-11} & ... & b_{i-1i-1} & b_{i-1i} & b_{i-1i+1} & ... & b_{i-1j-1} & b_{i-1j} & b_{i-1j+1} & ... & b_{i-1n} \\ b_{i1} & ... & b_{ii-1} & b_{ii} & b_{ii+1} & ... & b_{ij-1} & b_{ij} & b_{ij+1} & ... & b_{in} \\ \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots \\ b_{j-11} & ... & b_{j-1i-1} & b_{j-1i} & b_{j-1i+1} & ... & b_{j-1j-1} & b_{j-1j} & b_{j-1j+1} & ... & b_{j-1n} \\ b_{j1} & ... & b_{ji-1} & b_{ji} & b_{ji+1} & ... & b_{jj-1} & b_{jj} & b_{jj+1} & ... & b_{jn} \\ b_{j+11} & ... & b_{j+1i-1} & b_{j+1i} & b_{j+1i+1} & ... & b_{j+1j-1} & b_{j+1j} & b_{j+1j+1} & ... & b_{j+1n} \\ \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots & \vdots & \vdots & ... & \vdots \\ b_{n1} & ... & b_{ni-1} & b_{ni} & b_{ni+1} & ... & b_{nj-1} & b_{nj} & b_{nj+1} & ... & b_{nn} \end{pmatrix}$

Por tanto las parejas simétricamente resultantes de la matriz B serán de la forma: $\{b_{ki}, b_{ik}\}$, con $k \neq j$; $\{b_{kj}, b_{ij}\}$, con $k \neq i$; $\{b_{kl}, b_{lk}\}$, con $k, l \neq i, j$ y $\{b_{ji}, b_{ij}\}$. Supongamos ahora que permutamos B mediante

$$\sigma^1 = (\sigma_1\sigma_2...\sigma_{i-1}\sigma_j\sigma_i\sigma_{i+1}...\sigma_{j-1}\sigma_{j+1}...\sigma_n).$$

Permutando primeramente columnas y después filas, obtendríamos:

$$\begin{pmatrix} b_{11} & ... & b_{1j} & b_{1i} & ... & b_{1j-1} & b_{1j+1} & ... & b_{1n} \\ \vdots & ... & \vdots & \vdots & ... & \vdots & \vdots & ... & \vdots \\ b_{i1} & ... & b_{ij} & b_{ii} & ... & b_{ij-1} & b_{ij+1} & ... & b_{in} \\ \vdots & ... & \vdots & \vdots & ... & \vdots & \vdots & ... & \vdots \\ b_{j-11} & ... & b_{j-1j} & b_{j-1i} & ... & b_{j-1j-1} & b_{j-1j+1} & ... & b_{j-1n} \\ b_{j1} & ... & b_{jj} & b_{ji} & ... & b_{jj-1} & b_{jj+1} & ... & b_{jn} \\ b_{j+11} & ... & b_{j+1j} & b_{j+1i} & ... & b_{j+1j-1} & b_{j+1j+1} & ... & b_{j+1n} \\ \vdots & ... & \vdots & \vdots & ... & \vdots & \vdots & ... & \vdots \\ b_{n1} & ... & b_{nj} & b_{ni} & ... & b_{nj-1} & b_{nj+1} & ... & b_{nn} \end{pmatrix}$$

$$\downarrow$$

$$\begin{pmatrix} b_{11} & ... & b_{1j} & b_{1i} & ... & b_{1j-1} & b_{1j+1} & ... & b_{1n} \\ \vdots & ... & \vdots & \vdots & ... & \vdots & \vdots & ... & \vdots \\ b_{i-11} & ... & b_{i-1j} & b_{i-1i} & ... & b_{i-1j-1} & b_{i-1j+1} & ... & b_{i-1n} \\ b_{j1} & ... & b_{jj} & b_{ji} & ... & b_{jj-1} & b_{jj+1} & ... & b_{jn} \\ b_{i1} & ... & b_{ij} & b_{ii} & ... & b_{ij-1} & b_{ij+1} & ... & b_{in} \\ \vdots & ... & \vdots & \vdots & ... & \vdots & \vdots & ... & \vdots \\ b_{j-11} & ... & b_{j-1j} & b_{j-1i} & ... & b_{j-1j-1} & b_{j-1j+1} & ... & b_{j-1n} \\ b_{j+11} & ... & b_{j+1j} & b_{j+1i} & ... & b_{j+1j-1} & b_{j+1j+1} & ... & b_{j+1n} \\ \vdots & ... & \vdots & \vdots & ... & \vdots & \vdots & ... & \vdots \\ b_{n1} & ... & b_{nj} & b_{ni} & ... & b_{nj-1} & b_{nj+1} & ... & b_{nn} \end{pmatrix}$$

De este modo, podemos ver que las posiciones que varían son las siguientes:

- En la posición (k, i), con $k \neq i, j$ de la matriz original B se encontraba el elemento b_{ki} y en la posición (k, i), con $k \neq i, j$ de matriz permutada B^{σ^1} se encuentra el elemento b_{kj}.
- En la posición (k, l), con $k \neq i, j$, $l \in \{i+1, ..., j-1\}$ de la matriz original B se encontraba el elemento b_{kl} y en la posición (k, l), de matriz permutada B^{σ^1} se encuentra el elemento b_{kl-1}.
- En la posición (i, k), con $k \neq i, j$ de la matriz original B se encontraba el elemento b_{ik} y en la posición (i, k), con $k \neq i, j$ de matriz permutada B^{σ^1} se encuentra el elemento b_{jk}.
- En la posición (l, k), con $k \neq i, j$, $l \in \{i+1, ..., j-1\}$ de la matriz original B se encontraba el elemento b_{lk} y en la posición (l, k), de matriz permutada B^{σ^1} se encuentra el elemento b_{l-1k}.

- En la posición (i, j), de la matriz original B se encontraba el elemento b_{ij} y en la posición (i, j), de matriz permutada B^{σ^1} se encuentra el elemento b_{jj+1}.
- En la posición (j, i) de la matriz original B se encontraba el elemento b_{ji} y en la posición (j, i) de matriz permutada B^{σ^1} se encuentra el elemento b_{j+1j}.

Veamos por último cómo afectan estos cambios a la formación de parejas asociadas de la matriz:

- Las posiciones que ocupaban las parejas de la forma $\{b_{ij}, b_{ji}\}$ de la matriz original B, pasan a estar ocupadas por las parejas de la forma $\{b_{jj+1}, b_{j+1j}\}$ de la matriz permutada B^{σ^1}.
- Las posiciones que ocupaban las parejas de la forma $\{b_{ki}, b_{ik}\}$ con $k \neq i, j$ de la matriz original B, pasan a estar ocupadas por las parejas de la forma $\{b_{kj}, b_{jk}\}$ con $k \neq i, j$ de la matriz B^{σ^1}.
- Las posiciones que ocupaban las parejas de la forma $\{b_{lk}, b_{kl}\}$ con $k \neq i, j$; $l \in \{i+1, ..., j-1\}$ de la matriz original B, pasan a estar ocupadas por las parejas de la forma $\{b_{l-1k}, b_{kl-1}\}$ de la matriz permutada B^{σ^1}.
- Las posiciones que ocupaban las parejas de la forma $\{b_{rs}, b_{sr}\}$ con $r, s \in \{1, ..., i-1\} \cup \{j+1, ..., n\}$ no cambian de posición.

Por tanto, las parejas simétricamente resultantes de la matriz permutada B^{σ^1} coinciden exactamente con las obtenidas para la matriz original B.
En el caso en el que la permutación que se quiera aplicar a la matriz B tenga más cambios, bastaría con aplicar el mismo razonamiento tantas veces como cambios de filas y columnas se quieran realizar. □

(iii) Para cada par de entradas asociadas, una de las entradas se encuentra por encima de la diagonal principal y la otra por debajo, $\forall \sigma \in S_n$ [3]. Por lo que, podemos acotar toda solución entre dos valores que puede que no sean alcanzados: $\sum_{i<j} min\{b_{ij},\, b_{ji}\} \leq f(\sigma) \leq \sum_{i<j} max\{b_{ij},\, b_{ji}\}$.

Ejemplo 2.3.3. Si tomamos los pares asociados de la matriz B del ejemplo anterior, $\forall \sigma \in S_3$, podemos acotar la solución entre dos valores del siguiente modo:

$$min\{1,3\}+min\{2,5\}+min\{4,6\} \leq f(\sigma) \leq max\{1,3\}+max\{2,5\}+max\{4,6\}$$
$$1+2+4=7 \leq f(\sigma) \leq 3+5+6=14.$$

Demostración 2.3.3. Como hemos demostrado en la Propiedad (ii) podemos organizar las entradas de la matriz que no están sobre la diagonal principal por parejas simétricamente localizadas $\{b_{\sigma_i\sigma_j}, b_{\sigma_j\sigma_i}\}$ $\forall \sigma \in S_n \wedge \sigma_i \neq \sigma_j$. Por tanto, $\sigma_i > \sigma_j$ o $\sigma_i < \sigma_j$.

Supongamos que $\sigma_j > \sigma_i$, entonces $b_{\sigma_i\sigma_j}$ se encuentra por encima de la diagonal principal $\Longrightarrow$ $b_{\sigma_j\sigma_i}$ se encuentra por debajo de la diagonal. Como en la definición de la función objetivo solo intervienen los valores situados por encima de la diagonal principal, solo uno de los elementos de cada pareja $\{b_{\sigma_i\sigma_j}, b_{\sigma_j\sigma_i}\}$ sumará en la función objetivo.
El mejor de los casos se dará cuando de cada pareja sea el valor máximo el que intervenga en la función objetivo y por tanto, el valor máximo que podríamos obtener para la función objetivo sería: $\sum_{i<j} max\{b_{ij}, b_{ji}\}$.
Por el contrario, el peor de los casos se dará cuando de cada pareja sea el valor mínimo el que intervenga en la función objetivo y por tanto, el valor mínimo que podríamos obtener sería: $\sum_{i<j} min\{b_{ij}, b_{ji}\}$. □

(iv) Si σ es la permutación con la que se obtiene la solución del problema de optimización, es decir, maximiza la suma de los elementos que se encuentren por encima de la diagonal, entonces, dicha permutación σ es la que minimiza la suma de los elementos que se encuentran por debajo de la diagonal [3]. Esto es, si σ es el óptimo global para (2.1), entonces también es el óptimo global de:

$$\begin{aligned} Min \quad & f' : S_n \longrightarrow \mathbb{R} \\ & \sigma \longmapsto f'(\sigma) = \sum_{i>j}^{n} b_{ij}^{\sigma} = \sum_{j=1}^{n-1} \sum_{i=j+1}^{n} b_{ij}^{\sigma} \end{aligned} \tag{2.4}$$

Ejemplo 2.3.4. Si tomamos el ejemplo anterior y calculamos el valor de la función (2.4) $\forall \sigma \in S_n$, tendremos los siguientes resultados:

- $f'(\sigma^1) = 3 + 5 + 6 = 14$
- $f'(\sigma^2) = 5 + 3 + 4 = 12$
- $f'(\sigma^3) = 1 + 5 + 6 = 12$
- $f'(\sigma^4) = 2 + 4 + 3 = 9$
- $f'(\sigma^5) = 6 + 1 + 2 = 9$
- $f'(\sigma^6) = 4 + 2 + 1 = 7$

Vemos que el menor de los valores es 7 que proviene de calcular el valor de f' con la permutación σ^6, que es precisamente la permutación con la que se obtiene el valor máximo de f.

Demostración 2.3.4. Sean $B \in Mat_{n\times n}(\mathbb{R})$ y $\sigma \in S_n$ óptimo global. Supongamos por reducción al absurdo que $\exists \sigma^1 \in S_n$ que no es óptimo global, pero que minimiza la suma de elementos que se encuentran por debajo de la diagonal principal. Como he demostrado en la propiedad (iii), para cada pareja de elementos asociados $\{b_{\sigma_i\sigma_j}, b_{\sigma_j\sigma_i}\}$ solo uno es un sumando de la Ecuación (2.1). Consecuentemente, para una misma permutación $\sigma \in S_n$, de cada pareja un elemento sumará en la definición de f y el otro elemento sumará en la función f'. Por tanto, si $f'(\sigma) > f'(\sigma^1) \Longrightarrow f(\sigma) < f(\sigma^1)\#$ ya que habíamos supuesto que σ era un óptimo global pero σ^1 no lo era. □

(v) Sea $B \in Mat_{n\times n}(\mathbb{R})$ una matriz simétrica, entonces $\forall \sigma \in S_n$ $f(\sigma)$ tiene un valor constante: $f(\sigma) = \sum_{i<j} b_{ij} = \sum_{i>j} b_{ij}$.

Ejemplo 2.3.5. Sea B $= \begin{pmatrix} 10 & 1 & 2 \\ 1 & 10 & 3 \\ 2 & 3 & 10 \end{pmatrix}$ una matriz simétrica.
Claramente $\forall \sigma \in S_3$, $f(\sigma) = 1 + 2 + 3 = 6$ $\forall \sigma \in S_3$.

Demostración 2.3.5. Si B es una matriz simétrica $\implies$ $b_{ij} = b_{ji}$ $\forall i \neq j$. Teniendo en cuenta las propiedades (ii) y (iii):
$\sum_{i<j} min\{b_{ij}, b_{ji}\} \leq f(\sigma) \leq \sum_{i<j} max\{b_{ij}, b_{ji}\} \iff$
$\iff \sum_{i<j} min\{b_{ij}, b_{ij}\} \leq f(\sigma) \leq \sum_{i<j} max\{b_{ij}, b_{ij}\} \iff$
$\iff \sum_{i<j} b_{ij} \leq f(\sigma) \leq \sum_{i<j} b_{ij} \implies \mathrm{f}(\sigma) = \sum_{i<j} b_{ij}$ □

Las demostraciones de estas propiedades no han sido extraídas de ninguna fuente y suponen un aporte personal. Además, la última propiedad también es un aporte propio y por eso consideramos importante también mencionarla.

2.4. Aplicaciones

Como ya hemos citado en la introducción el problema de la ordenación lineal tiene aplicaciones en diversos campos tales como [6]: la economía, la agregación de preferencias individuales, rankings deportivos, teoría de grafos, antropología... [1]

Una de las principales aplicaciones de este problema se da en economía [6], por ello, vamos a describir cómo podemos hacer la asociación entre un conjunto de datos económicos y el LOP. Podemos escribir el problema en los siguientes términos: supongamos que la economía se divide en n sectores y se recogen los valores que aporta cada sector a los demás. Esto es, tenemos una matriz de valores $B = [b_{ij}] \in Mat_{n\times n}(\mathbb{R})$ en la que cada entrada b_{ij} representa el valor que el sector i sirve al j en un tiempo determinado. Estos valores pueden representar diferentes conceptos, tales como: cantidad de mercancías servidas, importaciones o exportaciones de cada sector i a cada sector j. En consecuencia, dependiendo de los datos recogidos el objetivo de nuestro problema será reordenar los sectores de manera que se maximicen beneficios o se minimicen pérdidas.

[1] En este capítulo sólo se desarrolla en profundidad la aplicación en el sector económico, pues es una de las aplicaciones más importantes y uno de los campos de aplicación más ligado a las matemáticas. Por este motivo, será la que utilizaremos en el Capítulo 5 para resolver un ejemplo real. No obstante, se puede encontrar la explicación de algunas otras de las aplicaciones citadas en el Apéndice A.

Capítulo 3

Landscapes LOP

En el problema de la ordenación lineal el espacio de búsqueda crece de forma exponencial a medida que aumenta el tamaño del problema, ya que el espacio de búsqueda consiste en el grupo de permutaciones S_n donde n corresponde al tamaño de la matriz del problema. Sin embargo, el espacio de permutaciones no está ordenado, es decir, a priori no podemos establecer un orden entre ellas, pero lo que sí que podemos es dotar de una estructura a dicho espacio para poder definir una métrica o distancia sobre sus elementos. De este modo, podríamos, además de establecer una relación entre los puntos del espacio de búsqueda y sus valores de función objetivo y establecer también la distancia entre estos puntos.

3.1. Vecindarios en el espacio de permutaciones

Definición 3.1.1. Sea S un espacio discreto, un entorno o vecindario de un punto $(x_1x_2...x_n) \in S$ está definido por la aplicación [1]:

$$N : S \longrightarrow \mathcal{P}(S)\backslash\emptyset$$
$$(x_1x_2...x_n) \longmapsto N((x_1x_2...x_n))$$

Para un mismo punto del espacio se pueden definir vecindarios diferentes. Se describen a continuacón dos vecindarios para el caso en el que S es el grupo simétrico S_n.

3.1.1. Intercambio adyacente

.

Sea $\sigma \in S_n$ $(n \geq 2)$ una permutación, el vecindario intercambio adyacente está formado por el conjunto de todas las permutaciones que se obtienen al

intercambiar dos elementos adyacentes de σ [7].

Esto es, sea $\sigma = (\sigma_1\sigma_2...\sigma_i\sigma_{i+1}...\sigma_n) \in S_n$, entonces el vecindario adyacente será: $N_{ady}(\sigma) = \{(\sigma_1\sigma_2...\sigma_{i+1}\sigma_i...\sigma_n) \in S_n | i \in \{1, ..., n-1\}\}$.

Proposición 3.1.1. El vecindario adyacente de un una permutación de tamaño n, $\sigma \in S_n$, está formdo por $n-1$ permutaciones.

Demostraré esta proposición de dos maneras diferentes, la primera valiéndome de un método constructivo y la segunda por inducción.

Demostración 3.1.1. En primer lugar me basaré en cómo construir de forma ordenada este vecindario. Supongamos que tenemos una permutación $\sigma = (\sigma_1...\sigma_n)$ de la cual queremos calcular el vecindario adyacente. Empezaremos añadiendo al vecindario la permutación obtenida al intercambiar en σ el elemento de la primera posición σ_1 con el elemento de la posición segunda posición σ_2, que se encuentra a su derecha. Repetimos este proceso con el resto de los elementos, es decir añadimos al vecindario las permutaciones obtenidas al intercambiar el elemento de la posición $i-ésima$ con elemento de la posición $i+1-ésima$. Sin embargo, el elemento de la posición n-ésima no puede ser intercambiado con el de su derecha, ya que no hay elemento en la posición $n+1-ésima$. Por lo que, la cantidad de permutaciones que se pueden obtener es $n-1$, ya que son $n-1$ elementos los que hemos podido intercambiar con el elemento a su derecha.
(Podríamos haber hecho un razonamiento análogo intercambiando de derecha a izquierda los elementos de la permutación). □

Demostración 3.1.2. Ahora desarrollaré la demostración por inducción de $|N_{ady}(\sigma)| = n-1, \forall\sigma \in S_n$.

Primero lo comprobamos para las permutaciones de tamaño 2, 3 y 4:
Si $\sigma \in S_2 \rightarrow \sigma = (\sigma_1\sigma_2) \longrightarrow N_{ady}(\sigma) = \{(\sigma_2\sigma_1)\}$
$|N_{ady}(\sigma)| = 1 = 2-1$
Si $\sigma \in S_3 \rightarrow \sigma = (\sigma_1\sigma_2\sigma_3) \longrightarrow N_{ady}(\sigma) = \{(\sigma_2\sigma_1\sigma_3), (\sigma_1\sigma_3\sigma_2)\}$
$|N_{ady}(\sigma)| = 2 = 3-1$
Si $\sigma \in S_4 \rightarrow \sigma = (\sigma_1\sigma_2\sigma_3\sigma_4) \longrightarrow N_{ady}(\sigma) = \{(\sigma_2\sigma_1\sigma_3\sigma_4), (\sigma_1\sigma_3\sigma_2\sigma_4), (\sigma_1\sigma_2\sigma_4\sigma_3)\}$
$|N_{ady}(\sigma)| = 3 = 4-1$

Suponemos que es cierto para $\sigma \in S_{n-1}$: $N_{ady}(\sigma) = \{(\sigma_2\sigma_1\sigma_3...\sigma_{n-1}),$
$(\sigma_1\sigma_3\sigma_2\sigma_4...\sigma_{n-1}), ..., (\sigma_1\sigma_2...\sigma_{i+1}\sigma_i...\sigma_{n-1}), ..., (\sigma_1\sigma_2...\sigma_{n-1}\sigma_{n-2})\} \Longrightarrow$
$\Longrightarrow |N_{ady}(\sigma)| = n-2$ y veamos que se cumple para $\sigma \in S_n$:
$N_{ady}(\sigma) = \{(\sigma_2\sigma_1\sigma_3...\sigma_{n-1}\sigma_n), (\sigma_1\sigma_3\sigma_2\sigma_4...\sigma_{n-1}\sigma_n), ..., (\sigma_1\sigma_2...\sigma_{i+1}\sigma_i...\sigma_{n-1}\sigma_n),$

$$, ..., (\sigma_1\sigma_2...\sigma_{n-1}\sigma_{n-2}\sigma_n)\} \cup \{(\sigma_1\sigma_2...\sigma_{n-2}\sigma_n\sigma_{n-1})\}$$
$$\implies |N_{ady}(\sigma)| = n - 2 + 1 = n - 1 \qquad \square$$

3.1.2. 2-cambio o Intercambio

Sea $\sigma \in S_n$ $(n \geq 2)$ una permutación el vecindario intercambio está formado por el conjunto de todas las permutaciones que se obtienen al intercambiar dos elementos cualesquiera de σ [7].
Esto es, sea $\sigma = (\sigma_1\sigma_2...\sigma_i...\sigma_j...\sigma_n) \in S_n$ e $i < j$, entonces el vecindario intercambio será: $N_{int}(\sigma) = \{(\sigma_1\sigma_2...\sigma_j...\sigma_i...\sigma_n) \in S_n | i, j \in \{1, ..., n\}, i < j\}$.

Este vecindario se puede generalizar y construir el vecindario 2k-cambio, siendo $k \leq \lfloor \frac{n}{2} \rfloor$, donde n es el tamaño de la permutación.

Proposición 3.1.2. El vecindario intercambio de un una permutación de tamaño n, $\sigma \in S_n$, está formado por $\frac{n(n-1)}{2}$ permutaciones.

Al igual que en el caso anterior, voy a dar dos demostraciones la primera valiéndome de cómo formar el vecindario y la segunda demostración mediante inducción.

Demostración 3.1.3. Para esta primera demostración tendré en cuenta cómo construir los elementos de este vecindario de forma ordenada. Sea $\sigma = (\sigma_1...\sigma_n)$ cada una de las n posiciones puede ser intercambiada con las $n - 1$ posiciones restantes. Sin embargo, al intercambiar el elemento de la posición i-ésima, σ_i, con el elemento de la posición j-ésima, σ_j, obtendremos el mismo resultado que al intercambiar el elemento de la posición j-ésima con el elemento de posición la i-ésima, por tanto, la mitad de las permutaciones obtenidas están repetidas. Consecuentemente, la cantidad de elementos que forman este vecindario es: $\frac{n(n-1)}{2}$. $\square$

Demostración 3.1.4. Ahora desarrollaré la demostración por inducción de $|N_{int}(\sigma)| = \frac{n(n-1)}{2}, \forall \sigma \in S_n$.

Primero lo comprobamos para las permutaciones de tamaño: 2, 3 y 4.

Si $\sigma \in S_2 \rightarrow \sigma = (\sigma_1\sigma_2) \rightarrow N_{int}(\sigma) = \{(\sigma_2\sigma_1)\}$
$|N_{int}(\sigma)| = 1 = \frac{2(2-1)}{2}$

Si $\sigma \in S_3 \rightarrow \sigma = (\sigma_1\sigma_2\sigma_3) \rightarrow N_{int}(\sigma) = \{(\sigma_2\sigma_1\sigma_3), (\sigma_3\sigma_2\sigma_1), (\sigma_1\sigma_3\sigma_2)\}$
$|N_{int}(\sigma)| = 3 = \frac{3(3-1)}{2}$

Si $\sigma \in S_4 \rightarrow \sigma = (\sigma_1\sigma_2\sigma_3\sigma_4) \rightarrow N_{int}(\sigma) = \{(\sigma_2\sigma_1\sigma_3\sigma_4), (\sigma_3\sigma_2\sigma_1\sigma_4), (\sigma_4\sigma_2\sigma_3\sigma_1), (\sigma_1\sigma_3\sigma_2\sigma_4), (\sigma_1\sigma_4\sigma_3\sigma_2), (\sigma_1\sigma_2\sigma_4\sigma_3)\}$
$|N_{int}(\sigma)| = 6 = \frac{4(4-1)}{2}$

Suponemos que es cierto para $\sigma \in S_{n-1}$
$N_{int}(\sigma) = \{(\sigma_2\sigma_1\sigma_3...\sigma_{n-1}), (\sigma_3\sigma_2\sigma_1...\sigma_{n-1}), ..., (\sigma_{n-1}\sigma_2\sigma_3...\sigma_1), (\sigma_1\sigma_3\sigma_2...\sigma_{n-1}), ...,$
$, ..., (\sigma_1\sigma_{n-1}\sigma_3...\sigma_2), ..., (\sigma_1\sigma_2\sigma_3...\sigma_{n-1}\sigma_{n-2})\} \Longrightarrow |N_{int}(\sigma)| = \frac{(n-1)(n-2)}{2}$,

y veamos que se cumple para $\sigma \in S_n$:
$N_{int}(\sigma) = \{(\sigma_2\sigma_1\sigma_3...\sigma_{n-1}\sigma_n), (\sigma_3\sigma_2\sigma_1...\sigma_{n-1}\sigma_n), ..., (\sigma_{n-1}\sigma_2\sigma_3...\sigma_1\sigma_n),$
$(\sigma_1\sigma_3\sigma_2...\sigma_{n-1}\sigma_n), ..., (\sigma_1\sigma_{n-1}\sigma_3...\sigma_2\sigma_n), ..., (\sigma_1\sigma_2\sigma_3...\sigma_{n-1}\sigma_{n-2}\sigma_n)\}$
$\cup \{(\sigma_n\sigma_2\sigma_3...\sigma_{n-1}\sigma_1), (\sigma_1\sigma_n\sigma_3...\sigma_{n-1}\sigma_2), ..., (\sigma_1\sigma_2\sigma_3...\sigma_n\sigma_{n-1})\}$

$$\Longrightarrow |N_{int}(\sigma)| = \frac{(n-1)(n-2)}{2} + (n-1) = \frac{n^2-3n+2+2n-2}{2} = \frac{n^2-n}{2} = \frac{n(n-1)}{2}$$

. □

Ejemplo 3.1.1. Sea $\sigma = (12345) \in S_5$ una permutación, podemos calcular los vecindarios intercambio adyacente e intercambio de la siguiente forma:

Vecindario intercambio adyacente:
$N(\sigma)_{ady} = \{(21345), (13245), (12435), (12354)\}$
Vemos que el cardinal de este vecindario es: $|N(\sigma)_{ady}| = 4 = 5 - 1$ elementos.

Vecindario intercambio:
$N(\sigma)_{int} = \{(21345), (32145), (42315), (52341), (13245), (14325), (15342),$
$(12435), (12543), (12354)\}$
Observamos que el cardinal de este vecindario es: $|N(\sigma)_{int}| = 10 = \frac{5(5-1)}{2}$ elementos.

3.2. *Landscapes* LOP

Una vez conocido el concepto de vecindario vamos a unirlo a un problema de optimización combinatoria. Después, estableceremos una conexión entre los vecindarios de S_n descritos y nuestro problema LOP.

Definición 3.2.1. Llamamos *landscape* a la terna (S, f, N) donde S es el espacio de búsqueda de un problema de optimización combinatoria, f es su función objetivo y N un vecindario definido sobre S.

Como para un mismo espacio de búsqueda podemos encontrar diferentes vecindarios, también podemos definir diferentes *landscapes* para un mismo problema.

Definición 3.2.2. Sea (S, f, N) un *landscape*, diremos que un punto factible $(x_1x_2...x_n) \in S$ es un óptimo local si [3]:

- En el caso que se busque maximizar, $\forall (y_1y_2...y_n) \in N((x_1x_2...x_n))$ se cumple que $f((y_1y_2...y_n)) \leq f((x_1x_2...x_n))$.

- En el caso que se busque minimizar, $\forall(y_1y_2...y_n) \in N((x_1x_2...x_n))$ se cumple que $f((y_1y_2...y_n)) \geq f((x_1x_2...x_n))$.

Claramente, el óptimo local depende del vecindario elegido. Esto es, un punto factible puede ser óptimo local bajo un vecindario, pero puede no serlo bajo otro. Por el contrario, los óptimos globales no dependen del vecindario y además, serán óptimos locales bajo cualquier vecindario.

3.2.1. Óptimos locales del LOP

A continuación vamos a describir qué propiedades ha de tener una permutación y los elementos de la matriz del LOP para que sea un óptimo local para cada uno de los dos vecindarios que se ha descrito descrito anteriormente. Además, se van a analizar las implicaciones que tiene el hecho de que una solución sea óptimo local en el LOP.

Óptimo local del vecindario intercambio adyacente

Teorema 3.2.1. Una permutación $\sigma^* = (\sigma_1^*\sigma_2^*....\sigma_n^*) \in S_n$ es un óptimo local para el vecindario adyacente $\iff$ los elementos de la matriz $B = [b_{ij}]_{n\times n}$ del LOP satisfacen [7]:

$$b_{\sigma_i^*\sigma_{i+1}^*} \geq b_{\sigma_{i+1}^*\sigma_i^*} \ , \ \forall i \in \{1, ..., n-1\}.$$

Primero vamos a ver un ejemplo donde se observe esta propiedad y después lo demostraremos formalmente.

Ejemplo 3.2.1. Supongamos que $\sigma^* = (12345) \in S_5$ es un óptimo local del LOP sobre el vecindario de intercambio adyacente que es el siguiente:
$N_{ady}((12345)) = \{(21345), (13245), (12435), (12354)\}$.
Como hemos supuesto que σ^* es un óptimo local del LOP:
$f(\sigma*) \geq f(\sigma)$, $\forall\sigma \in N_{ady}(\sigma^*)$ y donde f está definida mediante la Ecuación (2.2).
Por tanto, aplicando la definición de óptimo local a los cuatro elementos del vecindario:

- $f(\sigma^*) = f((12345)) \geq f((21345)) \iff$
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{21} + b_{23} + b_{24} + b_{25} + b_{13} + b_{14} + b_{15} + b_{34} + b_{35} + b_{45} \iff b_{12} \geq b_{21}$

- $f(\sigma^*) = f((12345)) \geq f((13245)) \iff$
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{13} + b_{12} + b_{14} + b_{15} + b_{32} + b_{34} + b_{35} + b_{24} + b_{25} + b_{45} \iff b_{23} \geq b_{32}$

- $f(\sigma^*) = f((12345)) \geq f((12435)) \iff$
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{12} + b_{14} + b_{13} + b_{15} + b_{24} + b_{23} + b_{25} + b_{43} + b_{45} + b_{35} \iff b_{34} \geq b_{43}$

- $f(\sigma^*) = f((12345)) \geq f((12354)) \iff$
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{12} + b_{13} + b_{15} + b_{14} + b_{23} + b_{25} + b_{24} + b_{35} + b_{34} + b_{54} \iff b_{45} \geq b_{54}$

Como acabamos de comprobar, el Teorema 3.2.1 se cumple para este ejemplo, pero ahora vamos a demostrarlo formalmente.

Demostración 3.2.1. Supongamos que $\sigma^* = (\sigma_1^*\sigma_2^*....\sigma_n^*) \in S_n$ es un óptimo local del LOP sobre el vecindario de intercambio adyacente que es el siguiente:
$N_{ady}(\sigma^*) = \{(\sigma_1^*....\sigma_{i+1}^*\sigma_i^*....\sigma_n^*) | i \in \{1, ..., n-1\}\}$
Ahora calcularemos el valor de la función de cada σ perteneciente a $N_{ady}(\sigma^*)$ en función de $f(\sigma^*)$ que sí conocemos. Además, sabemos que $f(\sigma)$ y $f(\sigma^*)$ solo difieren en un sumando, es por ello que podemos establecer la siguiente igualdad:
$f(\sigma) = f(\sigma^*) - b_{\sigma_i^*\sigma_{i+1}^*} + b_{\sigma_{i+1}^*\sigma_i^*}$
Luego, σ^* es un óptimo local $\iff f(\sigma^*) \geq f(\sigma) = f(\sigma^*) - b_{\sigma_i^*\sigma_{i+1}^*} + b_{\sigma_{i+1}^*\sigma_i^*}$
$\iff f(\sigma^*) \geq f(\sigma^*) - b_{\sigma_i^*\sigma_{i+1}^*} + b_{\sigma_{i+1}^*\sigma_i^*} \iff 0 \geq -b_{\sigma_i^*\sigma_{i+1}^*} + b_{\sigma_{i+1}^*\sigma_i^*}$
$\iff b_{\sigma_i^*\sigma_{i+1}^*} \geq b_{\sigma_{i+1}^*\sigma_i^*}$ □

Con esto, hemos demostrado qué propiedades debe tener una permutación y también los elementos de la matriz del LOP para que esta sea mejor o igual que las $n-1$ permutaciones del vecindario adyacente. No obstante, se puede demostrar que existen otras permutaciones que, aún no perteneciendo al vecindario adyacente, tienen un valor de función objetivo necesariamente peor que el óptimo local conocido.

Ejemplo 3.2.2. Tomamos el ejemplo anterior en el que $\sigma^* = (12345) \in S_5$ es un óptimo local para el vecindario adyacente y en el que hemos deducido que: $b_{12} \geq b_{21}$, $b_{23} \geq b_{32}$, $b_{34} \geq b_{43}$ y $b_{45} \geq b_{54}$.

Además, hemos calculado el valor de su función objetivo que era:
$f((12345)) = b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45}$.

Vamos a calcular ahora el valor de la función para dos permutaciones diferentes que no pertenezcan a $N_{ady}(\sigma^*)$: $\sigma^1 = (13254)$ y $\sigma^2 = (13425)$.

$f((13254)) = b_{13} + b_{12} + b_{15} + b_{14} + b_{32} + b_{35} + b_{34} + b_{25} + b_{24} + b_{54}$
Como $b_{32} \leq b_{23}$ y $b_{54} \leq b_{45}$ por ser $\sigma*$ óptimo local de $N_{ady}(\sigma*)$

$$f((13254)) \leq b_{13} + b_{12} + b_{15} + b_{14} + b_{23} + b_{35} + b_{34} + b_{25} + b_{24} + b_{45} = f((\sigma^*))$$

Vemos que aunque $(13254) \notin N_{ady}(\sigma^*)$ el valor de la función objetivo es necesariamente menor o igual que el óptimo local σ^*. Esto se da porque hemos obtenido σ^1 intercambiando en σ^* dos parejas de posiciones adyacentes que no se solapan. O sea que, cada intercambio de dos posiciones adyacentes es independiente de cualquier otro intercambio. En nuestro ejemplo hemos intercambiado los elementos de las posiciones 2-3 y 4-5, las posiciones de la primera pareja no intervienen en la segunda pareja. Veamos ahora qué ocurre si estos cambios sí se solapan, esto es, si un elemento de una posición es intercambiado en dos o más ocasiones como en $\sigma^2 = (13425)$, donde hemos intercambiado el elemento de la segunda posición con el de la tercera y después el elemento de la tercera posición con el de cuarta.

$$f((13425)) = b_{13} + b_{14} + b_{12} + b_{15} + b_{34} + b_{32} + b_{35} + b_{42} + b_{45} + b_{25}$$

Como $b_{23} \leq b_{32}$ por ser $\sigma*$ óptimo local de $N_{ady}(\sigma*)$

$$f((13425)) \leq b_{13} + b_{14} + b_{12} + b_{15} + b_{34} + b_{23} + b_{35} + b_{42} + b_{45} + b_{25} =$$
$$= f((12345)) - b_{24} + b_{42}$$

Esta desigualdad no nos asegura que $f(\sigma^*) \geq f(\sigma^2)$, por ello no podemos descartar que σ^2 sea una opción mejor y esto se da porque los intercambios realizados para obtenerla se intersecan. Esto es, el elemento de una de las posiciones es intercambiado dos veces. El elemento de la segunda posición σ_2^2 en primer lugar es intercambiado con σ_3^2 y en segundo lugar con σ_4^2.

En este ejemplo hemos visto que si se intercambian parejas de elementos de posiciones adyacentes de forma que los pares no se intersequen, la permutación resultante tendrá un valor menor o igual al óptimo local buscado. Voy a generalizar y demostrar esta idea mediante el Teorema 3.2.2.

Teorema 3.2.2. Dado $\sigma^* = (\sigma_1^*...\sigma_{i-1}^*\sigma_i^*\sigma_{i+1}^*...\sigma_n^*)$ óptimo local del LOP para el vecindario de intercambio adyacente, toda permutación obtenida al intercambiar los elementos de las posiciones adyacentes de k pares de elementos, de manera que un mismo elemento no intervenga en más de un intercambio, dará un valor para la función objetivo menor o igual que el valor de la función objetivo del óptimo local encontrado σ^*.
Esto es, si $\sigma^* = (\sigma_1^*...\sigma_{i_1-1}^*\sigma_{i_1}^*\sigma_{i_1+1}^*...\sigma_{i_2-1}^*\sigma_{i_2}^*\sigma_{i_2+1}^*...\sigma_{i_k-1}^*\sigma_{i_k}^*\sigma_{i_k+1}^*...\sigma_n^*) \in S_n$ es óptimo local para el vecindario de intercambio adyacente entonces:
$f(\sigma^*) \geq f(\sigma)$,
$\forall\sigma \in N = \{(\sigma_1^*...\sigma_{i_1-1}^*\sigma_{i_1+1}^*\sigma_{i_1}^*...\sigma_{i_2-1}^*\sigma_{i_2+1}^*\sigma_{i_2}^*...\sigma_{i_k+1}^*\sigma_{i_k}^*...\sigma_n^*) \in S_n |\ k \leq \lfloor \frac{n}{2} \rfloor \}\ \}$

Demostración 3.2.2. Si $\sigma^* = (\sigma_1^*\cdots\sigma_i^*\cdots\sigma_n^*)$ es un óptimo local para el vecindario de intercambio adyacente, se cumple que:

$$b_{\sigma_i^*\sigma_{i+1}^*} \geq b_{\sigma_{i+1}^*\sigma_i^*}\ ,\ \forall i \in \{1, ..., n-1\}$$

Tomamos ahora la permutación
$\sigma = (\sigma_1^*\cdots\sigma_{i_1+1}^*\sigma_{i_1}^*\cdots\sigma_{i_2+1}^*\sigma_{i_2}^*\cdots\sigma_{i_k+1}^*\sigma_{i_k}^*\cdots\sigma_n^*)$ y calculamos el valor de la función y aplicamos las desigualdades obtenidas en el teorema anterior :

$$\begin{aligned} f(\sigma) &= f(\sigma^*) - b_{\sigma^*_{i_1}\sigma^*_{i_1+1}} - \cdots - b_{\sigma^*_{i_k}\sigma^*_{i_k+1}} + b_{\sigma^*_{i_1+1}\sigma^*_{i_1}} + \cdots + b_{\sigma^*_{i_k+1}\sigma^*_{i_k}} = \\ &= f(\sigma^*) - b_{\sigma^*_{i_1}\sigma^*_{i_1+1}} + b_{\sigma^*_{i_1+1}\sigma^*_{i_1}} + \cdots - b_{\sigma^*_{i_k}\sigma^*_{i_k+1}} + b_{\sigma^*_{i_k+1}\sigma^*_{i_k}} \leq \\ &\leq f(\sigma^*) - b_{\sigma^*_{i_1}\sigma^*_{i_1+1}} + b_{\sigma^*_{i_1}\sigma^*_{i_1+1}} + \cdots - b_{\sigma^*_{i_k}\sigma^*_{i_k+1}} + b_{\sigma^*_{i_k}\sigma^*_{i_k+1}} = f(\sigma^*) \\ \Longrightarrow\ & f(\sigma) \leq f(\sigma^*) \end{aligned}$$ □

En conclusión podemos decir que si encontramos una permutación $\sigma^* \in S_n$ que sea óptima para el vecindario adyacente, además de descartar las $n-1$ permutaciones que forman el vecindario, sabemos que habrá F_n permutaciones cuyo valor de la función objetivo será menor, siendo F_n el n-ésimo número de Fibonacci. Dada una lista de n elementos, F_n cuenta de cuantas formas podemos reordenar los n elementos intercambiando como mucho $\lfloor \frac{n}{2} \rfloor$ pares de elementos adyacentes sin que se intersequen. Recordemos que los números de Fibonacci se podían construir de manera natural de manera recursiva del siguiente modo [7]:
$F_1 = 1$,
$F_2 = 1$,
$F_n = F_{n-1} + F_{n-2}$, $\forall n \geq 3$.
Aunque también podemos darlos de forma no recursiva:
$F_{n+1} = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-i}{i}$, $\forall n \geq 0$.

Óptimo local del vecindario intercambio o 2-cambio

Teorema 3.2.3. Dado $\sigma^* = (\sigma^*_1\sigma^*_2....\sigma^*_n) \in S_n$ un óptimo local (para el vecindario intercambio o 2-cambio) $\Longleftrightarrow$ las entradas de la matriz $B = [b_{ij}]_{n\times n}$ satisfacen [7]:

$$\sum_{k=i+1}^{j} b_{\sigma^*_i\sigma^*_k} + \sum_{k=i}^{j-1} b_{\sigma^*_k\sigma^*_j} \geq \sum_{k=i+1}^{j} b_{\sigma^*_k\sigma^*_i} + \sum_{k=i}^{j-1} b_{\sigma^*_j\sigma^*_k} \ ,$$
$$\forall i \in \{1, ..., n-1\} \ , \ \forall j \in \{i+1, ..., n\}$$

Ejemplo 3.2.3. Al igual que en caso anterior, vamos a comenzar viendo cómo se cumple esto en un ejemplo sencillo.

Sea $\sigma^* = (12345) \in S_5$ un óptimo local para el vecindario intercambio que al ser $\sigma^* \in S_5$, tendrá $\frac{5\cdot(5-1)}{2} = 10$ elementos que serán los siguientes:
$N_{int}(\sigma^*) = \{(21345), (32145), (42315), (52341), (13245), (14325), (15342),$
$(12435), (12543), (12354)\}$
Por ser σ^* un óptimo local: $f(\sigma^*) \geq f(\sigma), \forall \sigma \in N_{int}(\sigma^*)$.
Así que, vamos a comprobar esto con todas las permutaciones del vecindario:

- $f(\sigma^*) = f((12345)) \geq f((21345)) \iff$ (3.1)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{21} + b_{23} + b_{24} + b_{25} + b_{13} + b_{14} + b_{15} + b_{34} + b_{35} + b_{45} \iff$
  $\iff b_{12} \geq b_{21}$

- $f(\sigma^*) = f((12345)) \geq f((32145)) \iff$ (3.2)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{32} + b_{31} + b_{34} + b_{35} + b_{21} + b_{24} + b_{25} + b_{14} + b_{15} + b_{45} \iff$
  $\iff b_{12} + b_{13} + b_{23} \geq b_{32} + b_{31} + b_{21}$

- $f(\sigma^*) = f((12345)) \geq f((42315)) \iff$ (3.3)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{42} + b_{43} + b_{41} + b_{45} + b_{23} + b_{21} + b_{25} + b_{31} + b_{35} + b_{15} \iff$
  $\iff b_{12} + b_{13} + b_{14} + b_{24} + b_{34} \geq b_{42} + b_{43} + b_{41} + b_{21} + b_{31}$

- $f(\sigma^*) = f((12345)) \geq f((52341)) \iff$ (3.4)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{52} + b_{53} + b_{54} + b_{51} + b_{23} + b_{24} + b_{21} + b_{34} + b_{31} + b_{41} \iff$
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{25} + b_{35} + b_{45} \geq b_{52} + b_{53} + b_{54} + b_{51} + b_{21} + b_{31} + b_{41}$

- $f(\sigma^*) = f((12345)) \geq f((13245)) \iff$ (3.5)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{13} + b_{12} + b_{14} + b_{15} + b_{32} + b_{34} + b_{35} + b_{24} + b_{25} + b_{45} \iff$
  $\iff b_{23} \geq b_{32}$

- $f(\sigma^*) = f((12345)) \geq f((14325)) \iff$ (3.6)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{14} + b_{13} + b_{12} + b_{15} + b_{43} + b_{42} + b_{45} + b_{32} + b_{35} + b_{25} \iff$
  $\iff b_{23} + b_{24} + b_{34} \geq b_{43} + b_{42} + b_{32}$

- $f(\sigma^*) = f((12345)) \geq f((15342)) \iff$ (3.7)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{15} + b_{13} + b_{14} + b_{12} + b_{53} + b_{54} + b_{52} + b_{34} + b_{32} + b_{42} \iff$
  $\iff b_{23} + b_{24} + b_{25} + b_{35} + b_{45} \geq b_{53} + b_{54} + b_{52} + b_{32} + b_{42}$

- $f(\sigma^*) = f((12345)) \geq f((12435)) \iff$ (3.8)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{12} + b_{14} + b_{13} + b_{15} + b_{24} + b_{23} + b_{25} + b_{43} + b_{45} + b_{35} \iff$
  $\iff b_{34} \geq b_{43}$

- $f(\sigma^*) = f((12345)) \geq f((12543)) \iff$ (3.9)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$
  $\geq b_{12} + b_{15} + b_{14} + b_{13} + b_{25} + b_{24} + b_{23} + b_{54} + b_{53} + b_{43} \iff$
  $\iff b_{34} + b_{35} + b_{45} \geq b_{43} + b_{53} + b_{54}$

- $f(\sigma^*) = f((12345)) \geq f((12354)) \iff$ (3.10)
  $\iff b_{12} + b_{13} + b_{14} + b_{15} + b_{23} + b_{24} + b_{25} + b_{34} + b_{35} + b_{45} \geq$

$$\geq b_{12} + b_{13} + b_{15} + b_{14} + b_{23} + b_{25} + b_{24} + b_{35} + b_{34} + b_{54} \iff$$
$$\iff b_{45} \geq b_{54}$$

Vemos que claramente se cumple el Teorema 3.2.3 y además, podemos observar que las condiciones (3.1), (3.5), (3.8) y (3.10) son las mismas que en el caso anterior, el caso del vecindario intercambio adyacente, Ejemplo 3.2.2. Efectivamente, es fácil observar que que dichas condiciones provienen de permutaciones que también pertenecen al vecindario de intercambio adyacente, ya que: $N_{ady}(\sigma) \subseteq N_{int}(\sigma), \forall \sigma \in S_n$.

Hemos visto que para un ejemplo concreto el Teorema 3.2.3 se cumple, pero ahora debemos demostrarlo formalmente para cualquier permutación y el vecindario intercambio.

Demostración 3.2.3. Sea $\sigma^* = (\sigma_1^*\sigma_2^* \dots \sigma_n^*)$ es un óptimo local para el vecindario intercambio $N_{int}(\sigma^*) = \{(\sigma_1^*....\sigma_{i-1}^*\sigma_j^*\sigma_{i+1}^*....\sigma_{j-1}^*\sigma_i^*\sigma_{j+1}^*....\sigma_n^*) | i \in \{1, ..., n-1\}, j \in \{i+1, ..., n\}\} \iff f(\sigma^*) \geq f(\sigma), \forall \sigma \in N_{int}(\sigma^*)$

$$\begin{aligned}
\iff f(\sigma^*) \geq f(\sigma^*) &- [b_{\sigma_i^*\sigma_{i+1}^*} + b_{\sigma_i^*\sigma_{i+2}^*} + \cdots + b_{\sigma_i^*\sigma_{j-1}^*} + b_{\sigma_i^*\sigma_j^*}] + \\
&+ [b_{\sigma_j^*\sigma_i^*} + b_{\sigma_j^*\sigma_{i+1}^*} + \cdots + b_{\sigma_j^*\sigma_{j-2}^*} + b_{\sigma_j^*\sigma_{j-1}^*}] + \\
&- [b_{\sigma_i^*\sigma_j^*} + b_{\sigma_{i+1}^*\sigma_j^*} + \cdots + b_{\sigma_{j-2}^*\sigma_j^*} + b_{\sigma_{j-1}^*\sigma_j^*}] + \\
&+ [b_{\sigma_{i+1}^*\sigma_i^*} + b_{\sigma_{i+2}^*\sigma_i^*} + \cdots + b_{\sigma_{j-1}^*\sigma_i^*} + b_{\sigma_j^*\sigma_i^*}] = \\
= f(\sigma^*) - \sum_{k=i+1}^{j} b_{\sigma_i^*\sigma_k^*} &+ \sum_{k=i}^{j-1} b_{\sigma_j^*\sigma_k^*} - \sum_{k=i}^{j-1} b_{\sigma_k^*\sigma_j^*} + \sum_{k=i+1}^{j} b_{\sigma_k^*\sigma_i^*} \iff \\
\iff 0 \geq - \sum_{k=i+1}^{j} b_{\sigma_i^*\sigma_k^*} &+ \sum_{k=i}^{j-1} b_{\sigma_j^*\sigma_k^*} - \sum_{k=i}^{j-1} b_{\sigma_k^*\sigma_j^*} + \sum_{k=i+1}^{j} b_{\sigma_k^*\sigma_i^*} \iff \\
\iff \sum_{k=i+1}^{j} b_{\sigma_i^*\sigma_k^*} &+ \sum_{k=i}^{j-1} b_{\sigma_k^*\sigma_j^*} \geq \sum_{k=i}^{j-1} b_{\sigma_j^*\sigma_k^*} + \sum_{k=i+1}^{j} b_{\sigma_k^*\sigma_i^*}
\end{aligned}$$

Para desarrollar la demostración me he basado en [7]. □

Al igual que en el caso anterior voy a generalizar y demostrar este resultado en el Teorema 3.2.4.

Teorema 3.2.4. Dada $\sigma^* = (\sigma_1^*...\sigma_{i_1-1}^*\sigma_{i_1}^*\sigma_{i_1+1}^*...\sigma_{i_2-1}^*\sigma_{i_2}^*\sigma_{i_2+1}^*...\sigma_{i_k-1}^*\sigma_{i_k}^*\sigma_{i_k+1}^*...\sigma_n^*) \in S_n$ solución óptima para el vecindario intercambio, toda permutación $\sigma \in S_n$ obtenida al realizar intercambios de los elementos de las posiciones de σ^* de modo que los intercambios no se solapen ni se intersequen entre sí, tendrá un valor de la función objetivo menor o igual que el de σ^*.
Es decir, si $\sigma^* = (\sigma_1^*...\sigma_{i_1-1}^*\sigma_{i_1}^*\sigma_{i_1+1}^*...\sigma_{i_2-1}^*\sigma_{i_2}^*\sigma_{i_2+1}^*...\sigma_{i_k-1}^*\sigma_{i_k}^*\sigma_{i_k+1}^*...\sigma_n^*) \in S_n$ es óptimo local de el vecindario intercambio $N_{int}(\sigma^*)$, entonces:
$f(\sigma^*) \geq f(\sigma), \forall \sigma \in N = \{\sigma = (\sigma_1^*...\sigma_{i_1-1}^*\sigma_{i_2}^*\sigma_{i_1+1}^*...\sigma_{i_2-1}^*\sigma_{i_1}^*\sigma_{i_2+1}^*\cdots$
$\sigma_{i_{k-1}-1}^*\sigma_{i_k}^*\sigma_{i_{k-1}+1}^*...\sigma_{i_k-1}^*\sigma_{i_{k-1}}^*\sigma_{i_k+1}^*...\sigma_n^*) \in S_n \mid k \leq \lfloor \frac{n}{2} \rfloor\}$

Demostración 3.2.4. Supongamos que $\sigma^* = (\sigma_1^*\cdots\sigma_{i_1}^*\cdots\sigma_{i_2}^*\cdots\sigma_{i_k}^*\cdots\sigma_n^*)$ es un óptimo local para el vecindario de intercambio, por lo que se cumple que:

$$\sum_{k=i+1}^{j} b_{\sigma_i^*\sigma_k^*} + \sum_{k=i}^{j-1} b_{\sigma_k^*\sigma_j^*} \geq \sum_{k=i+1}^{j} b_{\sigma_k^*\sigma_i^*} + \sum_{k=i}^{j-1} b_{\sigma_j^*\sigma_k^*} \ ,$$
$$\forall i \in \{1,...,n-1\} \ , \ \forall j \in \{i+1,...,n\}$$

Tomamos ahora la permutación
$\sigma = (\sigma_1^* \cdots \sigma_{i_2}^* \cdots \sigma_{i_1}^* \cdots \sigma_{i_4}^* \cdots \sigma_{i_3}^* \cdots \sigma_{i_k}^* \cdots \sigma_{i_{k-1}}^* \cdots \sigma_n^*)$ y calculamos el valor de la función y aplicamos las desigualdades obtenidas en el teorema anterior :

$$\begin{aligned}
f(\sigma) = f(\sigma^*) &- [b_{\sigma_{i_1}^*\sigma_{i_1+1}^*} + b_{\sigma_{i_1}^*\sigma_{i_1+2}^*} + \cdots + b_{\sigma_{i_1}^*\sigma_{i_2}^*}] + \\
&+ [b_{\sigma_{i_2}^*\sigma_{i_2+1}^*} + b_{\sigma_{i_2}^*\sigma_{i_2+2}^*} + \cdots + b_{\sigma_{i_2}^*\sigma_{i_1}^*}] + \\
&- [b_{\sigma_{i_1}^*\sigma_{i_2}^*} + b_{\sigma_{i_1+1}^*\sigma_{i_2}^*} + \cdots + b_{\sigma_{i_2-1}^*\sigma_{i_2}^*}] + \\
&+ [b_{\sigma_{i_2}^*\sigma_{i_1}^*} + b_{\sigma_{i_2+1}^*\sigma_{i_1}^*} + \cdots + b_{\sigma_{i_1-1}^*\sigma_{i_1}^*}] + \cdots\cdots\cdots = \\
= f(\sigma^*) &- \sum_{l=i_1+1}^{i_2} b_{\sigma_{i_1}^*\sigma_l^*} + \sum_{l=i_2+1}^{i_1} b_{\sigma_{i_2}^*\sigma_l^*} - \sum_{l=i_1}^{i_2-1} b_{\sigma_l^*\sigma_{i_2}^*} + \sum_{l=i_2}^{i_1-1} b_{\sigma_l^*\sigma_{i_1}^*} \cdots\cdots\cdots = \\
= f(\sigma^*) &- [\sum_{l=i_1+1}^{i_2} b_{\sigma_{i_1}^*\sigma_l^*} + \sum_{l=i_1}^{i_2-1} b_{\sigma_l^*\sigma_{i_2}^*}] + [\sum_{l=i_2+1}^{i_1} b_{\sigma_{i_2}^*\sigma_l^*} + \sum_{l=i_2}^{i_1-1} b_{\sigma_l^*\sigma_{i_1}^*}] + \cdots \leq \\
\leq f(\sigma^*) &- [\sum_{l=i_1+1}^{i_2} b_{\sigma_{i_1}^*\sigma_l^*} + \sum_{l=i_1}^{i_2-1} b_{\sigma_l^*\sigma_{i_2}^*}] + [\sum_{l=i_1+1}^{i_2} b_{\sigma_{i_1}^*\sigma_l^*} + \sum_{l=i_1}^{i_2-1} b_{\sigma_l^*\sigma_{i_2}^*}] + \cdots\cdots\cdots = f(\sigma^*) \\
\implies f(\sigma) &\leq f(\sigma^*)
\end{aligned}$$

□

Como vemos, gracias a este teorema podemos asegurar que hay más permutaciones de las $\frac{n(n-1)}{2}$ que forman el vecindario intercambio que tendrán un valor menor de la función objetivo. Exactamente, podemos descartar M_n soluciones, donde M_n representa el n-ésimo número de Motzkin. Una de las definiciones de M_n viene dada del siguiente modo: dados n puntos situados equidistantemente sobre una circunferencia, M_n cuenta el número de formas en las que se pueden dibujar sobre la circunferencia cuerdas que unan los n puntos de modo que las cuerdas no se crucen [7]:

$M_0 = 1$,
$M_1 = 1$,
$M_{n+1} = M_n + \sum_{i=0}^{n-1} M_i M_{n-1-i}, \forall n \geq 1.$

Capítulo 4

LOP asociado a grafos dirigidos

En este capítulo asociaremos nuestro problema con grafos dirigidos. Después, nos apoyaremos en el análisis del vecindario adyacente descrito en el capítulo anterior para encontrar relaciones entre óptimos locales de nuestro problema y caminos hamiltonianos del grafo asociado. Nunca antes se había realizado esta asociación entre grafos y LOP. Por tanto, he desarrollado la Sección 4.2 tomando como base lo aprendido y no teniendo en cuenta ningún trabajo anterior.

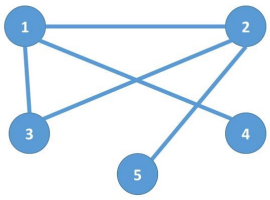
4.1. Definiciones básicas sobre grafos

Definición 4.1.1. Un grafo simple G consiste en un par de conjuntos finitos V y A donde cada elemento de V se llama vértice y cada elemento de A se llama arista, las cuales son conjuntos no ordenados de dos vértices.
$G = (V, A)$ donde $V = \{1, 2, ..., n\}$ y $A = \{\{i, j\} | i, j \in V\}$

Definición 4.1.2. Sea $G = (V, A)$ donde $V = \{1, ..., n\}$ y $A = \{\{i, j\} | i, j \in V\}$ un grafo, definimos la matriz de adyacencia del grafo $M \in Mat_{n \times n}(\{0, 1\})$, donde m_{ij} toma el valor 1 si la arista $\{i, j\} \in A$ o 0 en caso de que la arista $\{i, j\} \notin A$.

Ejemplo 4.1.1. Dado G podemos representarlo mediante dos estructuras. En la Figura 4.1 observamos la representación del grafo por medio de nodos y aristas (a la izquierda) y su matriz de adyacencia (a la derecha).

$$G = (\{1, 2, 3, 4, 5\}, \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 5\}\})$$



$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Figura 4.1: Grafo simple y su matriz de adyacencia

Definición 4.1.3. Un grafo dirigido o digrafo D consiste en dos conjuntos finitos V y A donde cada elemento de V es un vértice y cada elemento de A es una arista. En este caso, las aristas se expresan mediante pares ordenados, es decir: cada arista tiene una orientación específica.
$D = (V, A)$ donde $V = \{1, 2, ..., n\}$ y $A = \{(i, j)|i, j \in V\}$

Nota: La matriz de adyacencia de un grafo simple siempre será simétrica; en un digrafo, por el contrario, la matriz no siempre tendrá por qué ser simétrica.

Definición 4.1.4. Llamaremos grado de un vértice v y representaremos por $g(v)$ al número de aristas que inciden sobre él.
En un digrafo podemos definir otros dos conceptos relacionados con el grado del vértice:
- Grado positivo de v, $g^+(v)$: número de aristas que llegan a v.
- Grado negativo de v, $g^-(v)$: número de aristas que salen de v.
Nota: $g(v) = g^+(v) + g^-(v)$.

Ejemplo 4.1.2. Dado el digrafo

$$D = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 4), (3, 5), (4, 1), (5, 2)\}),$$

se observa su representación en la Figura 4.2, y se muestra en la Tabla 4.1 los grados de sus vértices.

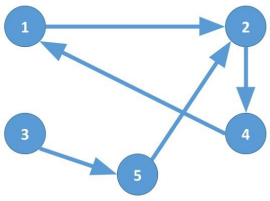


Figura 4.2: Grafo dirigido

vértice	$g^+(v)$	$g^-(v)$	g(v)
1	1	1	2
2	2	1	3
3	0	1	1
4	1	1	2
5	1	1	2

Tabla 4.1: Grados de los vértices del digrafo

Definición 4.1.5. Un camino de un grafo es una secuencia de aristas del grafo tales que cada par de aristas consecutivas comparten un vértice común. Se denota por $(v_1v_2...v_k)$ de modo que: $(v_i, v_{i+1}) \in A, \forall i \in \{1, ..., k-1\}$.

Definición 4.1.6. Un ciclo es un camino en el que el vértice inicial y final son el mismo. Esto es, es un camino tal que $(v_1v_2...v_kv_1)$ donde: $(v_i, v_{i+1}) \in A, \forall i \in \{1, ..., k-1\}$ y $(v_k, v_1) \in A$.

Definición 4.1.7. Un camino es hamiltoniano si cada vértice del grafo aparece exactamente una vez. Es decir: $(v_1, ..., v_n)$ siendo n el número de vértices del grafo y tal que: $v_i \neq v_j, \forall i, j \in \{1, ..., n\}$ y $(v_i, v_{i+1}) \in A$, $\forall i \in \{1, ..., n-1\}$.

Definición 4.1.8. Diremos que un grafo es hamiltoniano si tiene un ciclo hamiltoniano, es decir, si tiene un ciclo que recorre todos los vértices del grafo exactamente una vez a excepción del primero que se toma como inicial y final.

Definición 4.1.9. Diremos que un grafo es semi-hamiltoniano si contiene un camino hamiltoniano.

Nota: todo grafo hamiltoniano es semi-hamiltoniano.

Ejemplo 4.1.3. Si retomamos el Ejemplo 4.1.1 del grafo simple podemos encontrar un camino hamiltoniano, por ejemplo: (41325).

Definición 4.1.10. Un torneo es un digrafo $D = (V, A)$ en el que $\forall v_1, v_2 \in V$, al menos existe $(v_1, v_2) \in A$ o $(v_2, v_1) \in A$.

Teorema 4.1.1. Todo torneo no hamiltoniano es semi-hamiltoniano [8].

Demostración 4.1.1. Lo demostraremos por inducción, para ello tomaremos el torneo con el menor número de aristas posible, es decir, con una única conexión para cada par de vértices $v_1, v_2 \in V$. Si se cumple para estos torneos, también se cumplirá si añadimos más aristas.
Lo vemos para los grafos de dos y tres vértices:
- Para $n = 2$, tendríamos los grafos representados en la Figura 4.3.

Figura 4.3: Torneos de dos nodos con el mínimo de aristas posibles

En este primer caso, vemos fácilmente que existen caminos hamiltonianos para los dos grafos posibles: para a) tenemos (12) y b) tenemos (21).

- Para $n = 3$, tendríamos los 8 grafos representados en la Figura 4.4.

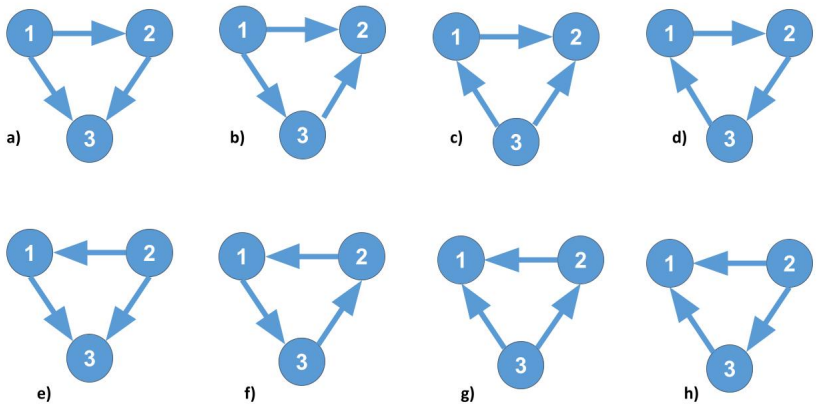


Figura 4.4: Torneos de tres nodos con el mínimo de aristas posibles

En este segundo caso, también es sencillo ver que podemos encontrar caminos hamiltonianos para los ocho grafos posibles. Para a): (123); para b): (132); para c): (312); para d): (123), (231) y (312); para e): (213); para f): (132), (213) y (321); para g): (321) y para h): (231).

- Lo suponemos cierto para $n = k$: $\exists(v_1v_2....v_{k-1}v_k)$ camino hamiltoniano. Lo comprobamos para $n = k + 1$, sabiendo que $\exists(v_1v_2....v_{k-1}v_k)$ camino hamiltoniano.

Como estamos trabajando con un torneo $D = (V, A)$, el vértice v_{k+1} que añadamos será de grado k, puesto que se añadirá una arista (entrante o saliente) uniendo el nuevo vértice v_{k+1} con cada uno de los k vértices restantes. Esto es: $g(v_{k+1}) = g^+(v_{k+1}) + g^-(v_{k+1}) = k$, donde $g^+(v_{k+1}), g^-(v_{k+1}) \in \{0, 1, ..., k\}$.
Distinguimos tres casos:

(i) Si $g^+(v_{k+1}) = k$ y $g^-(v_{k+1}) = 0 \implies$ todas las aristas de v_{k+1} son incidentes $\implies (v_i, v_{k+1}) \in A, \forall i \in \{1, 2, ..., k\} \implies (v_k, v_{k+1}) \in A \implies (v_1v_2...v_kv_{k+1})$ es camino hamiltoniano.

(ii) Si $g^+(v_{k+1}) = 0$ y $g^-(v_{k+1}) = k \implies$ todas las aristas salen de $v_{k+1} \implies (v_{k+1}, v_i) \in A, \forall i \in \{1, 2, ..., k\} \implies (v_{k+1}, v_1) \in A \implies (v_{k+1}v_1v_2...v_k)$ es camino hamiltoniano.

(iii) Si $g^+(v_{k+1}) = l$ y $g^-(v_{k+1}) = k - l$ para $l \in \{1, ..., k-1\}$.
Existen dos opciones:
a) $\exists (v_{k+1}, v_1) \in A \implies (v_{k+1}v_1v_2...v_k)$ es camino hamiltoniano.
b) $(v_{k+1}, v_1) \notin A$ y $\exists$ al menos un vértice v_i tal que $(v_i, v_{k+1}) \in A$ y $(v_{k+1}, v_{i+1}) \in A$ para algún $i \in \{1, 2, 3, ..., k-1\} \implies$ $(v_1v_2...v_{i-1}v_iv_{k+1}v_{i+1}...v_k)$ es camino hamiltoniano. Esto es, si hay unas aristas que entran y otras que salen, existirán dos vértices consecutivos v_i y v_{i+1}, tales que haya una arista que salga del vértice v_i y llegue al vértice v_{k+1} y otra arista que vaya de v_{k+1} a v_{i+1}. □

4.2. Asociación entre grafos dirigidos y LOP

Dada una matriz $B \in Mat_{n\times n}(\mathbb{R})$ de un problema de ordenación lineal, calculamos su grafo asociado, que será dirigido y ponderado, de la siguiente forma :

(i) Localizamos las parejas simétricamente localizadas de la matriz B, esto es: $\{b_{ij}, b_{ji}\}, \forall i, j \in \{1, 2, ..., n\}$, $i \neq j$ descritas en la Propiedad (ii) de la Sección 2.3.

(ii) Seleccionamos el mayor de los valores para cada pareja $\{b_{ij}, b_{ji}\}$.

(iii) Definimos un grafo con n vértices y, al menos, $\frac{n(n-1)}{2}$ aristas que colocaremos del siguiente modo:

- Si b_{ij} es el mayor de los valores de la pareja $\{b_{ij}, b_{ji}\}$, entonces definiremos una arista de i a j.
- Si b_{ji} es el mayor de los valores, entonces dibujaremos una arista de j a i.
- Si $b_{ij} = b_{ji}$ entonces dibujaremos dos aristas de i a j y de j a i.

Nótese que nuestro grafo asociado siempre será un torneo, pues para cada par de vértices $v_1, v_2 \in V$ siempre $\exists (v_1, v_2) \in A$ o $\exists (v_2, v_1) \in A$. Además, gracias al Teorema 4.1.1, sabemos que siempre contendrá un camino hamiltoniano. A la matriz de adyacencia de este grafo asociado le denominaremos matriz de adyacencia de la matriz B del LOP.

Teorema 4.2.1. $\sigma = (\sigma_1\sigma_2...\sigma_n)$ es un camino hamiltoniano del grafo asociado al LOP, si y sólo si $\sigma \in S_n$ es óptimo local del LOP bajo el vecindario adyacente.

Demostración 4.2.1. Supongamos que $\sigma = (\sigma_1\sigma_2...\sigma_n)$ es un camino hamiltoniano del grafo asociado al LOP, por tanto $(\sigma_i, \sigma_{i+1}) \in A, \forall i \in \{1, ..., n-1\}$. Entonces, por la forma en la que hemos construido el grafo:

$$\begin{cases} b_{\sigma_1\sigma_2} \geq b_{\sigma_2\sigma_1} \\ b_{\sigma_2\sigma_3} \geq b_{\sigma_3\sigma_2} \\ \dots \\ b_{\sigma_{n-1}\sigma_n} \geq b_{\sigma_n\sigma_{n-1}} \end{cases} \iff \begin{cases} b_{\sigma_i\sigma_{i+1}} \geq b_{\sigma_{i+1}\sigma_i} \\ \forall i \in \{1, \dots, n-1\} \end{cases} \iff$$

$\iff \sigma$ óptimo local del LOP por el vecindario adyacente $N_{ady}(\sigma)$ por el Teorema 3.2.1. □

Con esto, hemos conseguido demostrar que en nuestro grafo asociado siempre podemos encontrar, al menos, un camino hamiltoniano y que va a ser óptimo locales bajo el vecindario adyacente. En el Ejemplo 4.2.1 que mostramos a continuación, nos valemos de los Teoremas 4.1.1 y 4.2.1 para resolver un problema LOP.

Ejemplo 4.2.1. Si tomamos la matriz $B = [b_{ij}] \in Mat_{4\times4}(\mathbb{N} \cup \{0\})$ y obtenemos las parejas simétricamente localizadas:

$$\mathrm{B} = \begin{pmatrix} 0 & 2 & 1 & 7 \\ 7 & 0 & 8 & 3 \\ 5 & 9 & 0 & 2 \\ 11 & 1 & 3 & 0 \end{pmatrix} \implies \{\{b_{12}, b_{21}\}, \{b_{13}, b_{31}\}, \{b_{14}, b_{41}\}, \{b_{23}, b_{32}\}, \{b_{24}, b_{42}\}, \{b_{34}, b_{43}\}\} = \\ = \{\{2,7\}, \{1,5\}, \{7,11\}, \{8,9\}, \{3,1\}, \{2,3\}\}$$

Analizamos cada pareja para construir nuestro digrafo asociado del modo que hemos descrito al comienzo de esta sección. Como nuestra matriz es de tamaño 4, el conjunto de vértices será: $V = \{1,2,3,4\}$. Para construir el conjunto de aristas debemos analizar cada pareja $\{b_{ij}, b_{ji}\}$, comenzamos con la primera pareja $\{b_{12}, b_{21}\} = \{2,7\}$ donde el mayor de los valores es 7, por tanto la primera arista que incluiremos en A será $(2,1)$. Análogamente, analizamos el resto de las parejas y obtenemos el conjunto de aristas $A = \{(2,1),(3,1),(4,1),(3,2),(2,4),(4,3)\}$. Por lo que el digrafo $D = (V,A)$ asociado sería el representado en la Figura 4.5.

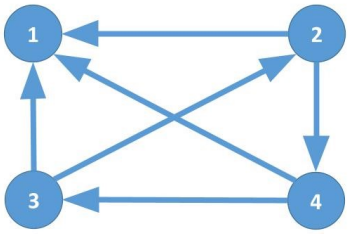


Figura 4.5: Grafo asociado al LOP con matriz B

Posibles caminos hamiltonianos: $\{(4321), (3241), (2431)\}$. Vamos a analizarlos uno a uno:

- $(4321) \Longrightarrow \begin{cases} f((4321)) = 36 \\ N_{ady}((4321)) = \{(3421), (4231), (4312)\} \end{cases}$

Si (4321) es un óptimo local para $N_{ady}((4321))$
$\Longrightarrow f((4321)) = 36 \geq f(\sigma), \forall \sigma \in N_{ady}((4321))$, comprobémoslo:

  - $f(\sigma^1) = f((3421)) = 35 \leq 36 = f((4321))$
  - $f(\sigma^2) = f((4231)) = 35 \leq 36 = f((4321))$
  - $f(\sigma^3) = f((4312)) = 31 \leq 36 = f((4321))$

- $(3241) \Longrightarrow \begin{cases} f((3241)) = 36 \\ N_{ady}((3241)) = \{(2341), (3421), (3214)\} \end{cases}$

Si (3241) es unóptimo local para $N_{ady}((3241))$
$\Longrightarrow f((3241)) = 37 \geq f(\sigma), \forall \sigma \in N_{ady}((3241))$, comprobémoslo:

  - $f(\sigma^1) = f((2341)) = 36 \leq 37 = f((3241))$
  - $f(\sigma^2) = f((3421)) = 35 \leq 37 = f((3241))$
  - $f(\sigma^3) = f((3214)) = 35 \leq 37 = f((3241))$

- $(2431) \Longrightarrow \begin{cases} f((2431)) = 37 \\ N_{ady}((2431)) = \{(4231), (2341), (2413)\} \end{cases}$

Si (2431) es unóptimo local para $N_{ady}((2431))$
$\Longrightarrow f((2431)) = 37 \geq f(\sigma), \forall \sigma \in N_{ady}((2431))$, comprobémoslo:

  - $f(\sigma^1) = f((4231)) = 35 \leq 37 = f((2431))$
  - $f(\sigma^2) = f((2341)) = 36 \leq 37 = f((2431))$
  - $f(\sigma^3) = f((2413)) = 33 \leq 37 = f((2431))$

En este ejemplo hemos comprobado que los caminos hamiltonianos del grafo asociado a la matriz del LOP son óptimos locales, y por tanto el óptimo global será una de estas tres permutaciones. Esto es, calculando únicamente el valor de tres permutaciones hemos conseguido obtener la solución del problema sin necesidad de evaluar las $4! = 24$ permutaciones que forman el espacio de búsqueda S_4. Así, para este ejemplo tenemos que la solución es 37 y las permutaciones con las que obtenemos dicho valor son: (3241) y (2431).

En un ejemplo pequeño es posible calcular la solución del LOP valiéndonos de la relación que hemos establecido entre el grafo asociado a la matriz del LOP, sus caminos hamiltonianos y el vecindario adyacente. Sin embargo, no sabemos si este método es efectivo para ejemplos más grandes y complejos. Por ese motivo, en el siguiente capítulo vamos a implementar un algoritmo que resuelva ejemplos de mayor tamaño midiendo la eficacia.

Capítulo 5

Métodos de resolución y experimentación

De acuerdo con lo que hemos afirmado en los capítulos anteriores el problema de la ordenación lineal es del tipo NP-Completo. En consecuencia, si queremos obtener la solución, debemos aplicar un método exacto cuyo tiempo de ejecución sea no polinómico en la dimensión del problema. Abordaremos el problema utilizando dos métodos exactos: el más simple, aunque también el más ineficiente, es el de la búsqueda exhaustiva; y un segundo método que busque la solución valiéndose de la relación establecida en el capítulo anterior entre el LOP y los digrafos junto con el Teorema 4.1.1. Este segundo nuevo método será mucho más eficiente que una búsqueda exhaustiva, aún siendo también no polinómico. A fin de observar esta diferencia en el tiempo de ejecución: desarrollaremos ambos métodos, los aplicaremos a matrices de diferentes tamaños y mediremos el tiempo necesario para resolver el problema.

5.1. Métodos de resolución

5.1.1. Búsqueda exhaustiva

El primer método que implementamos para resolver el LOP es la búsqueda exhaustiva. Como ya hemos citado a lo largo de capítulos anteriores, no sólo es interesante el valor máximo de la función objetivo, si no también la permutación o permutaciones con las que hemos obtenido ese valor, es por ello que el algoritmo que implementamos devolverá el cálculo de la función objetivo junto con la permutación o permutaciones con las que se obtiene dicho valor. Dada una matriz de tamaño n, el algoritmo calcula el valor de la función objetivo para cada una de las permutaciones del espacio de búsqueda S_n. En cada cálculo de la función objetivo, se compara el nuevo resultado obtenido con el mejor resultado conseguido hasta ese momento

que denominaremos solución actual, pudiéndose dar tres situaciones:

- Si la solución actual es mayor que el nuevo resultado, entonces se desecha el nuevo valor y se mantiene la solución actual.
- Si la solución actual es menor que el nuevo resultado, entonces se desecha la solución actual y se establece el valor calculado como nueva solución actual junto con la nueva permutación.
- Si la solución actual y el nuevo resultado tienen el mismo valor, solamente se guarda la nueva permutación junto con las ya obtenidas hasta ese momento y se mantiene el valor de la solución actual.

5.1.2. Búsqueda basada en los caminos hamiltonianos del grafo asociado

En segundo lugar desarrollaremos un nuevo método nunca usado hasta el momento basándonos en lo desarrollado en el Capítulo 4. Nuestro algoritmo funcionará de la siguiente forma:

(i) Dada una matriz del LOP de tamaño n obtenemos su matriz de adyacencia y construimos el grafo asociado a dicha matriz, tal y como hemos descrito en el Capítulo 4.

(ii) En segundo lugar buscamos los caminos hamiltonianos del grafo que sabemos por el Teorema 4.2.1 que serán óptimos locales del LOP para el vecindario adyacente.

(iii) Se aplica el algoritmo de la búsqueda exhaustiva tomando como espacio de búsqueda únicamente el conjunto de los caminos hamiltonianos.

5.2. Experimentación

Para la experimentación se han usado funciones escritas con Python y Mathematica que se encuentran adjuntas en el Apéndice B. En ellas se podrán ver tanto las pruebas realizadas para obtener los tiempos de ejecución de la Subsección 5.2.1, como los resultados de la instancia de la Subsección 5.2.2.

5.2.1. Comparativa de tiempos de ejecución

Para comparar los dos métodos descritos en la sección anterior tomamos matrices aleatorias de diferentes tamaños y medimos los tiempos de ejecución que recogemos en las gráficas de la Figura 5.1. En ambas gráficas se recoge en el eje horizontal el tamaño de la matriz y en el eje vertical el tiempo en segundos a escala natural y a escala logarítmica, a izquierda y derecha, respectivamente.

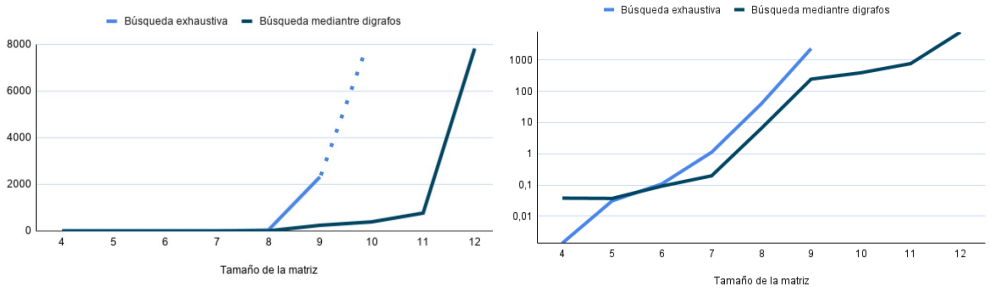


Figura 5.1: Gráfica comparativa del tiempo de ejecución de los métodos descritos a escala natural (izquierda) y a escala logarítmica (derecha)

Como podemos ver para matrices de tamaño pequeño los tiempos de ejecución son muy pequeños y similares. De hecho, en la gráfica a escala logarítmica, podemos apreciar que la búsqueda exhaustiva es más rápida para tamaños menores o iguales a 6. Sin embargo, a medida que aumentamos el tamaño de la matriz, el tiempo de ejecución aumenta de forma exponencial para ambos métodos. No obstante, podemos observar que para matrices de tamaño mayores o iguales a 7 el tiempo de ejecución es mucho mayor para la búsqueda exhaustiva que para la búsqueda mediante digrafos. Además, para matrices de tamaño 10 la búsqueda exhaustiva se detiene sin resultados tras unos días de ejecución; mientras que la búsqueda mediante digrafos puede darse en un tiempo de ejecución factible hasta matrices de tamaño 10, 11, 12 e incluso de tamaño mayor.

5.2.2. Aplicación a un ejemplo real

Teniendo en cuenta la teoría vista en los capítulos anteriores, vamos a resolver un problema real. Para ello, tomamos una matriz que recoge los datos de 15 sectores de economía de Estados Unidos de 1993[1]. La matriz con la que trabajaremos es la siguiente:

$$\begin{pmatrix}
30707 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 75579 & 844 & 0 & 32 & 7 \\
0 & 184 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 \\
0 & 21 & 240 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
21 & 11 & 13 & 445 & 8 & 8 & 1 & 6 & 2 & 0 & 0 & 1 & 2 & 16 & 3 \\
0 & 0 & 0 & 0 & 5725 & 50 & 0 & 0 & 0 & 0 & 6 & 7 & 8 & 0 & 1 \\
241 & 15 & 3 & 0 & 0 & 1 & 89 & 7 & 1917 & 1697 & 0 & 18 & 0 & 2 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 7 & 5 & 0 & 0 & 0 & 26 & 0 & 0 & 0 \\
1059 & 19 & 9 & 146 & 4313 & 0 & 15 & 594 & 2116 & 7 & 78 & 11 & 0 & 48 & 0 \\
18 & 0 & 0 & 0 & 0 & 0 & 0 & 23 & 0 & 1538 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 & 1 & 53820 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 5 & 0 & 1 & 4 & 0 & 0 & 4 & 0 & 10645 & 2036 & 15719 & 14 \\
95 & 0 & 0 & 0 & 0 & 0 & 0 & 1892 & 436 & 3 & 9 & 0 & 769 & 1083 & 46 \\
0 & 0 & 1 & 5 & 11 & 2 & 1 & 125 & 0 & 17 & 70 & 0 & 0 & 10690 & 0 \\
0 & 4 & 22 & 101 & 0 & 0 & 2 & 2 & 0859 & 2722 & 3 & 1 & 3 & 6 & 0 \\
174 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 & 56 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Tras introducir nuestra matriz, lo primero que hacemos es obtener su

[1] `https://grafo.etsii.urjc.es/optsicom/lolib/`

matriz de adyacencia y seguidamente obtenemos el grafo que se muestra en la Figura 5.2.

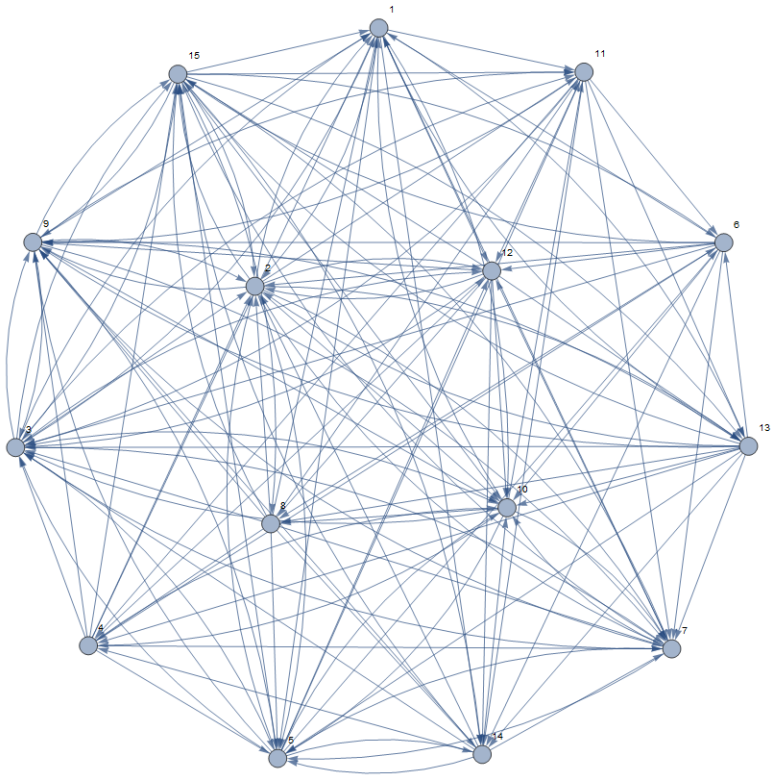


Figura 5.2: Grafo asociado al problema económico de 15 sectores de EE.UU

En segundo lugar buscamos los caminos hamiltonianos del grafo y aplicamos el algoritmo de la búsqueda exhaustiva tomando como espacio de búsqueda $4{,}71449315 \times 10^8$ caminos hamiltonianos. De este modo, aún teniendo que analizar una gran cantidad de permutaciones, es un número menor a las 15! = $1{,}3076744 \times 10^{12}$ permutaciones que componen el espacio de búsqueda S_{15}. De este modo, obtenemos que la solución al problema es 185.216 que se obtiene con más de 100 permutaciones distintas, esto es, en este caso el problema tiene un óptimo global múltiple. Gracias a este nuevo método de resolución hemos conseguido resolver una instancia del problema de la ordenación lineal de tamaño 15, mientras que con la búsqueda exhaustiva el tamaño máximo que hemos conseguido resolver ha sido 10. No obstante, para problemas de tamaño mayor el tiempo de ejecución sería excesivo y, por tanto, sería más adecuado utilizar un método heurístico. Tal vez la asociación realizada en este trabajo, entre el problema de estudio y los digrafos, pueda ayudar al desarrollo y avance de técnicas heurísticas potentes para su resolución.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

En este documento he descrito problemas de optimización combinatoria profundizando en el problema de la ordenación lineal. Aunque aparentemente el problema de la ordenación lineal puede parecer sencillo debido a sus propiedades y a su simetría, esto no es así, pues se trata de un problema NP-Completo. Es decir, no se conoce ningún algoritmo que resuelva todas sus instancias en un tiempo polinómico en la dimensión del problema, y además, dado un punto factible, tampoco se puede comprobar si es solución del problema. En este trabajo, he definido dos vecindarios y he estudiado las propiedades del problema de la ordenación lineal bajo estos vecindarios, relacionándolas con familias importantes de números: los números de Fibonacci y los números de Motzkin. Gracias a ello, he conseguido establecer una nueva relación entre el problema de optimización y los grafos dirigidos. En base a esta relación he desarrollado un algoritmo exacto que se basa en la búsqueda de caminos hamiltonianos del grafo asociado al LOP. He demostrado formalmente que estos caminos hamiltonianos son óptimos locales para vecindario adyacente del LOP. Esta observación nunca se había hecho antes y la he desarrollado por primera vez en este trabajo. Este estudio, me ha permitido implementar un método más eficaz que la búsqueda exhaustiva. Aún así, durante la experimentación he observado algunas carencias que relato a continuación:

- El algoritmo se basa en la búsqueda de caminos hamiltonianos de un grafo que es un problema NP y por tanto el tiempo de ejecución aumenta de forma exponencial en la dimensión del problema.
- El algoritmo no puede ser aplicado a problemas de tamaño grande, ya que necesita varios días de ejecución para problemas de tamaño 15.

- Si los elementos de las parejas simétricamente localizadas $\{b_{ij}, b_{ji}\}$ tienen el mismo valor, el grafo asociado al LOP tendrá dos aristas entre los vértices i, j y por tanto la cantidad de posibles caminos hamiltonianos aumenta notablemente. Lo que da lugar a dos principales problemas: aumento del tiempo de ejecución y aumento del almacenamiento necesario para guardar los caminos hamiltonianos.

6.2. Trabajo futuro

Vistas las debilidades de nuestro algoritmo, sería conveniente buscar un método heurístico eficiente, bueno y robusto que resuelva el problema de la búsqueda de caminos hamiltonianos en un tiempo polinómico. La búsqueda de caminos ha sido mucho más estudiada a fin de encontrar el camino de peso mínimo entre dos vértices de un grafo. Por el contrario, el LOP trata de maximizar, pero en la propiedad (iv) de la Sección 2.3 hemos visto que podemos reconvertir el LOP en un problema de minimización tomando como entrada la transpuesta de la matriz o tomando la función objetivo f' descrita mediante la Ecuación (2.4) que calcula la suma de los elementos situados por debajo de la diagonal principal. Teniendo esto en cuenta, una primera idea podría ser buscar un método heurístico para la resolución del LOP mediante la búsqueda de caminos de peso mínimo en el grafo asociado a la matriz del problema. No obstante, para poder aplicar un heurístico que busque caminos de peso mínimo en un grafo, el grafo debe ser ponderado, por contra, el grafo que hemos asociado al LOP no es ponderado. En conclusión, en un futuro podríamos construir un algoritmo que: en primer lugar, asociara la matriz transpuesta del LOP con un digrafo ponderado y en segundo lugar, que buscara el camino mínimo de dicho digrafo. Este camino sería la solución propuesta para el LOP. La precisión de este nuevo heurístico dependerá de la forma en la que otorguemos los valores a las aristas del grafo asociado.

La realización de este trabajo me ha permitido aplicar conceptos de Matemática Discreta estudiados durante el Grado y profundizar más en esta rama de las Matemáticas. Asimismo, también he podido ampliar mis conocimientos en áreas que no conocía, tales como: la Optimización Combinatoria o la Teoría de la Complejidad Computacional. También ha significado un avance personal poder formular y desarrollar de forma propia teoremas y propiedades con sus respectivas demostraciones. Además, he podido implementar un método basándome en lo aprendido y utilizarlo para la resolución de un pequeño ejemplo real. Pero me gustaría poder seguir mejorando este algoritmo para poder resolver instancias de un tamaño mayor. Creo que el desarrollo de técnicas para la resolución de este tipo problemas puede suponer un gran avance para las Matemáticas y para la vida cotidiana, ya que este campo tiene una aplicación directa en nuestro día a día.

Apéndice A

Aplicaciones LOP

Una de las principales aplicaciones y la más cercana a las matemáticas es la Economía [6]. Esta aplicación es la que ha sido explicada en la Sección 2.4 y la que hemos utilizado para desarrollar un ejemplo real en la Subsección 5.2.2.

Otra de las aplicaciones son los rankings deportivos [6]. Supongamos que en una liga deportiva cada equipo juega contra cada uno de los demás $n-1$ equipos dos veces. El ganador consigue 3 puntos y en caso de empate, ambos consiguen 1 punto. Finalizados todos los encuentros, en caso de que dos equipos tengan la misma puntuación, queda como vencedor aquel equipo que haya conseguido encajar más goles a su adversario. Con estos goles se puede construir una matriz $A = [a_{ij}] \in Mat_{n\times n}(\mathbb{N} \cup \{0\})$ en la que cada entrada a_{ij} representa el número de tantos que el equipo i ha conseguido contra el equipo j. En este caso, el objetivo es maximizar las puntuaciones de nuestro equipo y/o minimizar las de nuestros adversarios. Por lo que, si encontramos el orden de filas y columnas de la matriz que haga que la suma de los elementos por encima de la diagonal sea máxima, encontraríamos un nuevo ranking. En este nuevo ranking el vencedor quedaría en primera posición el equipo que más tantos haya conseguido encajar a sus rivales.

Una aplicación más es la asignación de preferencias individuales [6]. Supongamos que hay n personas y cada una de ellas ordena una lista de n elementos según sus preferencias individuales. Ahora queremos crear una única lista que diste lo menos posible de las listas creadas por las n personas. Para ello, construimos la matriz $B = [b_{ij}] \in Mat_{n\times n}(\mathbb{N} \cup \{0\})$, de modo que en este caso, cada entrada b_{ij} representa el número de personas que prefieren el elemento i al elemento j. Resolviendo el problema y encontrando la reordenación de filas y columnas que maximizan la suma de las entradas por encima de la diagonal principal obtendremos la lista de preferencias que mejor se adapte de forma global a las listas de preferencias individuales.

Otros problemas que podemos modelizar como un LOP son problemas asociados a grafos [2]. Por ejemplo, supongamos que tenemos un grafo dirigido donde $D = (V, A, W)$ donde $V = \{1, ...n\}$ representa el conjunto de vértices, $A = \{(i, j)|i, j \in V\}$ representa el conjunto de aristas y $W = \{w_{ij}|(i, j) \in A\}$ representa los pesos asociados a cada arista. Entonces, el problema consiste en encontrar $A' \subseteq A$ tal que $G = (V, A', W')$ sea un grafo sin ciclos y cuyo peso asociado sea mínimo. Este problema se puede convertir en un LOP construyendo la matriz del siguiente modo: $B = [b_{ij}] \in Mat_{n\times n}(\mathbb{R})$ donde b_{ij} representa el peso de la arista (i, j) si existe y 0 en caso contrario. En este caso nuestro objetivo es minimizar la suma del peso de nuestro grafo. Aunque también podemos definir el problema de forma análoga buscando maximizar el peso del grafo, valiéndonos de la Propiedad (iv) vista en la Sección 2.3.

Un segundo ejemplo de grafos es el de la minimización de cruces [2]. Supongamos que tenemos un grafo bipartito $H = (V, A)$, con la bipartición $V = W_1 \cup W_2$. El problema consiste en dibujar los nodos en dos líneas paralelas en una de ellas situaremos los vértices W_1 y en la otra línea los de W_2. Después, dibujaremos las aristas del grafo que unirán los dos conjuntos de vértices. En este caso, nuestro objetivo es encontrar en qué orden debemos colocar los vértices para que los cruces entre las aristas sean mínimos. Esto, lo podemos convertir en un LOP en el que en el que cada entrada $b_{ij} \in (\mathbb{N} \cup \{0\})$ representa el número de cruces de la arista (i, j). Así que, una vez obtenida nuestra matriz LOP nuestro objetivo será minimizar el valor de la suma de los elementos por encima de la diagonal, o lo que es equivalente, minimizar el número de cruces entre las aristas de nuestro grafo.

Apéndice B

Implementación

En las siguientes hojas, se muestran las funciones escritas con Mathematica y la función que busca caminos hamiltonianos de un grafo escrita con Python. Estas han sido utilizadas en la realización de algunos ejemplos y la experimentación del Capítulo 5. Explico a continuación las funciones descritas en Mathematica.

- Suma: calcula la suma de los elementos de una matriz permutada que se sitúan por encima de la diagonal principal y devuelve el valor de la suma.
- Maximo: dada una matriz resuelve el problema de la ordenación lineal mediante la búsqueda exhaustiva y devuelve la permutación/es donde se alcanza la solución y la solución del problema.
- Matrizady: dada una matriz devuelve la matriz de adyacencia del grafo asociado.
- Diccionario: dada una matriz de adyacencia de un grafo, crea un diccionario en las que las claves son el nombre de los vértices $\{1, ..., n\}$ y cada vértice tiene asociado las aristas que inciden sobre él.
- Maxcamino: dada una matriz y un conjunto de caminos hamiltonianos del grafo asociado, devuelve qué permutación produce el resultado del problema de la ordenación lineal de la matriz y la solución al problema.
- Vecindario: dada una permutación devuelve el vecindario adyacente de dicha permutación.
- Parejas: dada una matriz, devuelve una lista de las parejas asociadas simétricamente localizadas de una matriz.
- Minmax: dada una lista de las parejas asociadas de una matriz, devuelve la cota superior e inferior de un problema de ordenación lineal.

Funciones

In[ ]:=
```
suma[sigma_, matriz_] := Module[{suma, i, j, n}, n = Dimensions[matriz][[1]];
                         módulo                   dimensiones
  suma = 0; For[i = 1, i ≤ (n - 1), i++, For[j = (i + 1), j ≤ n,
             para cada                    para cada
    j++ , suma = suma + matriz[[sigma[[i]]]][[sigma[[j]]]]]; suma]
```

In[ ]:=
```
maximo[mat_] := Module[{n, i, l, sol, res, max = {}}, n = Dimensions[mat][[1]];
                módulo                                 dimensiones
  sol = 0;
  For [i = 1, i ≤ Factorial[n], i++, l = suma[Permutations[Range[n], {n}][[i]], mat];
  para cada          factorial              permutaciones    rango
   If[sol < l, {sol = l;
   si
     max = Permutations[Range[n], {n}][[i]]},
           permutaciones    rango
    If [sol == l, AppendTo[max, Permutations[Range[n], {n}][[i]]]];
    si            añade al final  permutaciones    rango
  {sol, max}]
```

In[ ]:=
```
matrizady[mat_] := Module[{n, i, j, ady}, n = Dimensions[mat][[1]];
                   módulo                  dimensiones
  ady = ConstantArray[0, {n, n}]; For[i = 1, i ≤ (n - 1), i++, For[j = (i + 1),
        arreglo constante           para cada                  para cada
    j ≤ n, j++ , If [mat[[i]][[j]] ≥ mat[[j]][[i]], ady[[i]][[j]] = 1];
                 si
    If [mat[[i]][[j]] ≤ mat[[j]][[i]], ady[[j]][[i]] = 1]]]; ady]
    si
```

In[ ]:=
```
diccionario[matady_] :=
 Module[{i, j, l, sol = {}}, For [i = 1, i ≤ Length[matady], i++, {l = {} ;
 módulo                      para cada                        longitud
    For [j = 1, j ≤ Length[matady], j++, If[ matady[[i]][[j]] == 1, AppendTo[l, j]]];
    para cada          longitud              si                     añade al final
    Print[l]}]]
    escribe
```

```
In[ ]:= maxcamino[caminos_, matriz_] := Module[{i, camax = {}, l, sol = 0},
                                        módulo
  For[i = 1, i ≤ Length[caminos], i++, l = suma[caminos[[i]], matriz];
  para cada        longitud
   If[l > sol, {sol = l, AppendTo[camax, caminos[[i]]]},
   si                    añade al final
    If[l == sol, AppendTo[camax, caminos[[i]]]]];
    si             añade al final
  {sol, camax}]
```

```
In[ ]:= vecindario[sigma_] :=
  Module[{i, lista = {}, n, l = {}}, n = Length[sigma]; For[i = 1, i ≤ n - 1,
  módulo                                  longitud          para cada
    i++, {l = ReplacePart[sigma, {i → sigma[[i + 1]], i + 1 → sigma[[i]]}];
                sustituye una parte
     l = AppendTo[lista, l]}]; lista]
         añade al final
```

```
In[ ]:= parejas[mat_] := Module[{n, i, j, l = {}}, n = Dimensions[mat][[1]];
                         módulo                     dimensiones
  For[i = 1, i ≤ n - 1, i++,
  para cada
   For[j = i + 1, j ≤ n, j++, AppendTo[l, {mat[[i]][[j]], mat[[j]][[i]]}]];
   para cada                  añade al final
  l]
```

```
In[ ]:= minmax[list_] := Module[{i, n, min, max}, min = 0; max = 0; n = Dimensions[list][[1]];
                         módulo                                      dimensiones
  For[i = 1, i ≤ n, i++, {min += Min[list[[i]]], max += Max[list[[i]]]}];
  para cada                 mínimo                  máximo
  {min, max}]
```

Ejemplo capítulo 4

```
In[ ]:= mat = {{0, 2, 1, 7}, {7, 0, 8, 3}, {5, 9, 0, 2}, {11, 1, 3, 0}};
        MatrixForm[mat]
        forma de matriz
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 2 & 1 & 7 \\ 7 & 0 & 8 & 3 \\ 5 & 9 & 0 & 2 \\ 11 & 1 & 3 & 0 \end{pmatrix}$$

```
In[ ]:= Timing[maximo[mat]]
        cronometra
```

```
Out[ ]= {0., {37, {2, 4, 3, 1, {3, 2, 4, 1}}}}
```

In[ ]:=
```
AbsoluteTiming[maximo[mat]]
```
duración absoluta

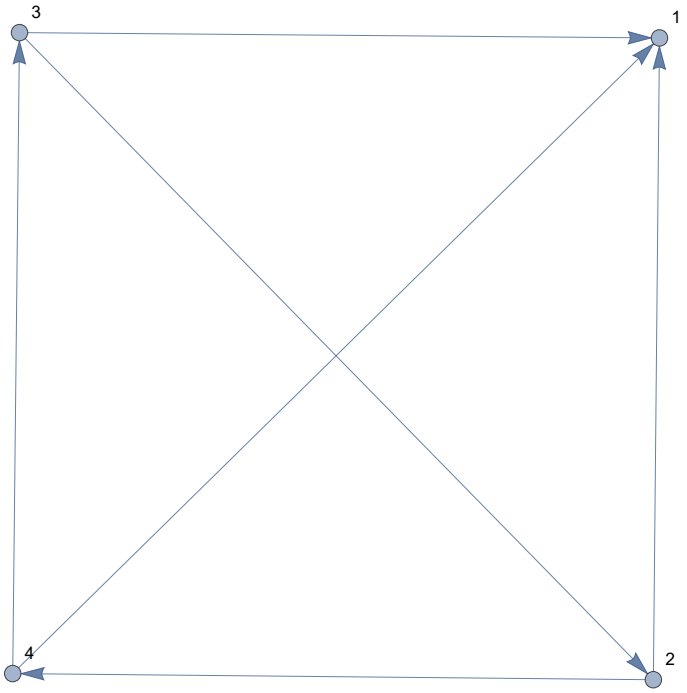
Out[ ]=
```
{0.0014313, {37, {2, 4, 3, 1, {3, 2, 4, 1}}}}
```

In[ ]:=
```
matady = matrizady[mat];
MatrixForm[matady]
```
forma de matriz
```
grafo = AdjacencyGraph[matady, VertexLabels → Automatic, DirectedEdges → True]
```
grafo con adyacencia | etiquetas de vérti… | automático | aristas dirigidas | verdade

Out[ ]//MatrixForm=

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Out[ ]=



In[ ]:=
```
(*Caminos hamiltonianos con python*)
matcaminos = {{2, 4, 3, 1}, {3, 2, 4, 1}, {4, 3, 2, 1}};
(*Calculamos el valor de la función objetivo para camino conseguido*)
For[i = 1, i ≤ Length[matcaminos], i++,
```
para cada | longitud
```
 Print[suma[matcaminos[[i]], mat], matcaminos[[i]]]]
```
escribe

```
37{2, 4, 3, 1}

37{3, 2, 4, 1}

36{4, 3, 2, 1}
```

In[ ]:=
```
AbsoluteTiming[maxcamino[matcaminos, mat]]
```
duración absoluta

Out[ ]=
```
{0.0002844, {37, {{2, 4, 3, 1}, {3, 2, 4, 1}}}}
```

```
(* Comprobamos con la función vecindario que las permutaciones son óptimos locales*)

(*Para {2,4,3,1} con el que obtenemos un valor de 37 *)
For [i = 1, i ≤ Length[vecindario[{2, 4, 3, 1}]],
 para cada          longitud
  i++, Print[suma[vecindario[{2, 4, 3, 1}][[i]], mat]]]
       escribe
```

```
35

36

33
```

```
(*Para {3,2,4,1} con el que obtenemos un valor de 37*)
For [i = 1, i ≤ Length[vecindario[{3, 2, 4, 1}]],
 para cada          longitud
  i++, Print[suma[vecindario[{3, 2, 4, 1}][[i]], mat]]]
       escribe
```

```
36

35

33
```

```
(*Para {4,3,2,1} con el que obtenemos un valor de 36*)
For [i = 1, i ≤ Length[vecindario[{4, 3, 2, 1}]],
 para cada          longitud
  i++, Print[suma[vecindario[{4, 3, 2, 1}][[i]], mat]]]
       escribe
```

```
35

35

31
```

In[ ]:=

```
(*Aunque en este caso hemos podido calcular el valor exacto mediante
 la función máximo, aplicamos también la función minimax para acotar
 la función objetivo y ver que la función funciona correctamente*)
listparejas = parejas[mat]
minmax[listparejas]
```

Out[ ]= {{2, 7}, {1, 5}, {7, 11}, {8, 9}, {3, 1}, {2, 3}}

Out[ ]= {21, 38}

Experimentación capítulo 5

In[ ]:=

```
matriz5 =
  {{3, 2, 4, 6, 8}, {2, 7, 8, 9, 0}, {3, 4, 12, 10, 8}, {23, 5, 6, 8, 0}, {4, 5, 8, 0, 8}};
MatrixForm[matriz5]
```
forma de matriz

```
Timing[maximo[matriz5]]
```
cronometra

```
MatrixForm[ady5 = matrizady[matriz5]];
```
forma de matriz

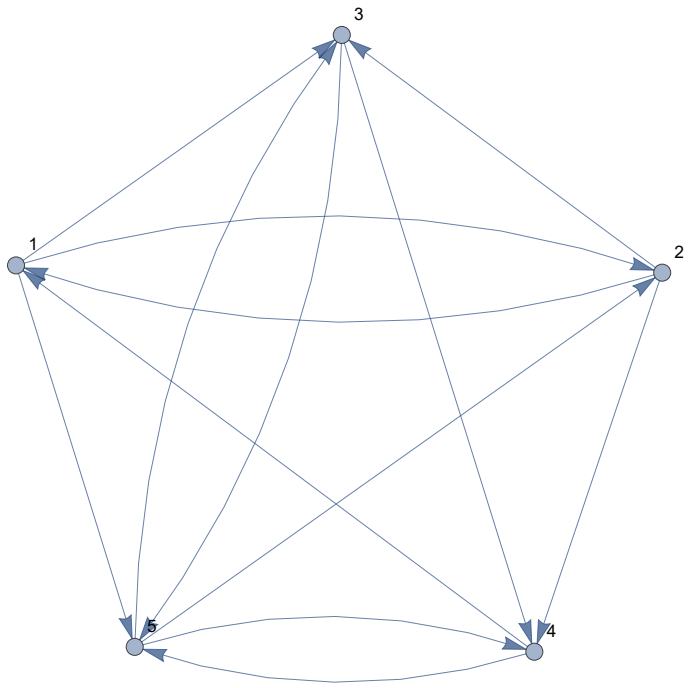
```
AdjacencyGraph[ady5, VertexLabels → Automatic]
```
grafo con adyacencia | etiquetas de vérti··· | automático

Out[ ]//MatrixForm=

$$\begin{pmatrix} 3 & 2 & 4 & 6 & 8 \\ 2 & 7 & 8 & 9 & 0 \\ 3 & 4 & 12 & 10 & 8 \\ 23 & 5 & 6 & 8 & 0 \\ 4 & 5 & 8 & 0 & 8 \end{pmatrix}$$

Out[ ]= {0.015625, {72, {5, 2, 3, 4, 1}}}

Out[ ]=



In[•]:=
```
caminos5 = ReadList["C:/Users/Usuario/Desktop/tfg/graph5.txt"];
           lee lista      constante
Timing[maxcamino[caminos5, matriz5]]
cronometra
```

Out[•]=
```
{0.015625, {72, {{1, 2, 3, 4, 5}, {1, 2, 3, 5, 4}, {1, 3, 5, 2, 4},
    {1, 5, 2, 3, 4}, {2, 3, 4, 1, 5}, {5, 2, 3, 4, 1}, {5, 2, 3, 4, 1}}}}
```

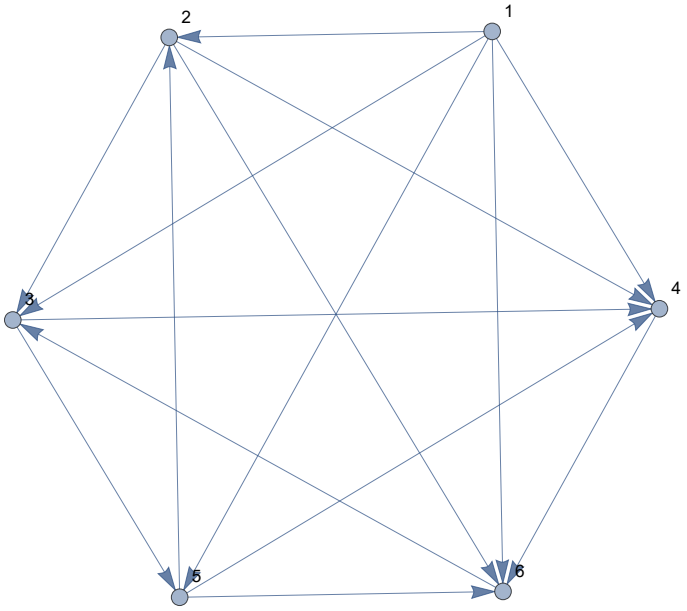
In[•]:=
```
matriz6 = {{3, 4, 9, 6, 8, 10}, {3, 5, 8, 23, 2, 76}, {3, 7, 2, 12, 32, 2},
    {2, 4, 5, 6, 7, 10}, {3, 5, 11, 21, 32, 12}, {2, 3, 4, 6, 5, 1}};
MatrixForm[matriz6]
forma de matriz
Timing[maximo[matriz6]]
cronometra
MatrixForm[ady6 = matrizady[matriz6]];
forma de matriz
AdjacencyGraph[ady6, VertexLabels → Automatic]
grafo con adyacencia   etiquetas de vérti···   automático
```

Out[•]//MatrixForm=

$$\begin{pmatrix} 3 & 4 & 9 & 6 & 8 & 10 \\ 3 & 5 & 8 & 23 & 2 & 76 \\ 3 & 7 & 2 & 12 & 32 & 2 \\ 2 & 4 & 5 & 6 & 7 & 10 \\ 3 & 5 & 11 & 21 & 32 & 12 \\ 2 & 3 & 4 & 6 & 5 & 1 \end{pmatrix}$$

Out[•]=
```
{0.109375, {237, {1, 3, 5, 2, 4, 6}}}
```

Out[•]=



In[•]:=
```
caminos6 = ReadList["C:/Users/Usuario/Desktop/tfg/graph6.txt"];
           lee lista      constante
AbsoluteTiming[maxcamino[caminos6, matriz6]]
duración absoluta
```

Out[•]=
```
{0.0027909, {237, {{1, 2, 3, 5, 4, 6}, {1, 3, 5, 2, 4, 6}, {1, 3, 5, 2, 4, 6}}}}
```

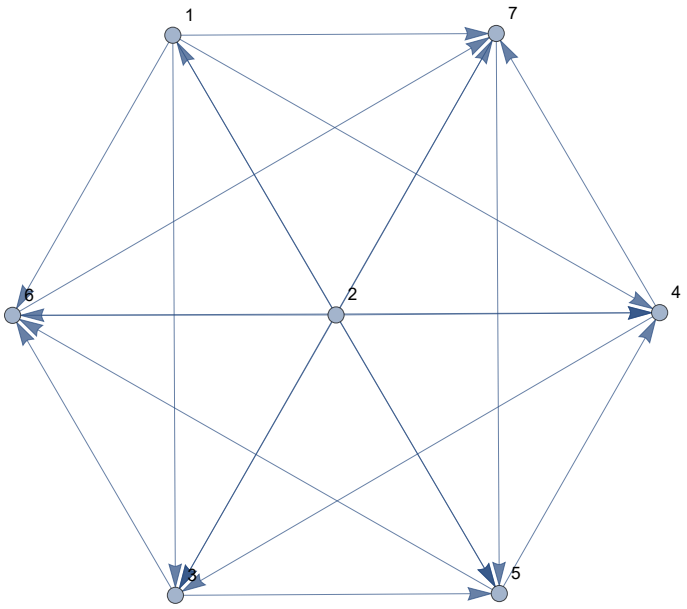
In[ ]:=
```
matriz7 = {{2, 3, 4, 5, 7, 7, 4},
    {4, 4, 6, 6, 7, 8, 87}, {1, 2, 3, 5, 67, 78, 0}, {2, 5, 67, 4, 3, 24, 10},
    {1, 3, 5, 65, 43, 6, 0}, {2, 4, 6, 75, 3, 3, 7}, {3, 35, 6, 8, 5, 5, 1}};
MatrixForm[matriz7]
```
forma de matriz
```
Timing[maximo[matriz7]]
```
cronometra
```
MatrixForm[ady7 = matrizady[matriz7]];
```
forma de matriz
```
AdjacencyGraph[ady7, VertexLabels → Automatic]
```
grafo con adyacencia | etiquetas de vérti··· | automático

Out[ ]//MatrixForm=

$$\begin{pmatrix} 2 & 3 & 4 & 5 & 7 & 7 & 4 \\ 4 & 4 & 6 & 6 & 7 & 8 & 87 \\ 1 & 2 & 3 & 5 & 67 & 78 & 0 \\ 2 & 5 & 67 & 4 & 3 & 24 & 10 \\ 1 & 3 & 5 & 65 & 43 & 6 & 0 \\ 2 & 4 & 6 & 75 & 3 & 3 & 7 \\ 3 & 35 & 6 & 8 & 5 & 5 & 1 \end{pmatrix}$$

Out[ ]= {1.125, {465, {2, 1, 7, 3, 5, 6, 4}}}

Out[ ]=



In[ ]:=
```
caminos7 = ReadList["C:/Users/Usuario/Desktop/tfg/graph7.txt"];
```
lee lista | constante
```
AbsoluteTiming[maxcamino[caminos7, matriz7]]
```
duración absoluta

Out[ ]= {0.0073452, {465,
  {{2, 1, 3, 5, 6, 4, 7}, {2, 1, 3, 6, 7, 5, 4}, {2, 1, 7, 3, 5, 6, 4}, {2, 1, 7, 3, 5, 6, 4}}}}

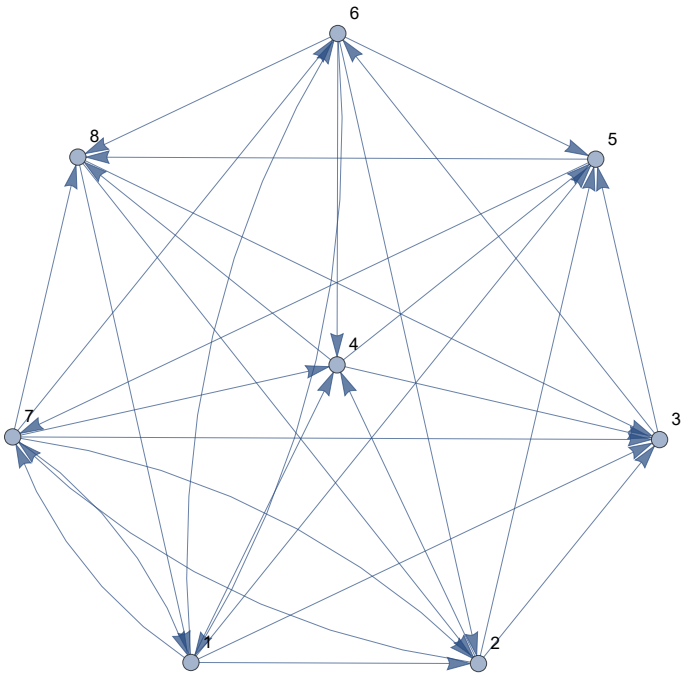
In[ ]:=
```
matriz8 = {{2, 4, 5, 7, 9, 4, 3, 2}, {2, 4, 6, 8, 43, 2, 5, 6},
    {3, 5, 7, 4, 32, 23, 5, 5}, {5, 3, 35, 4, 7, 65, 8, 53}, {5, 6, 8, 5, 3, 3, 6, 9},
    {4, 5, 6, 88, 9, 4, 3, 3}, {3, 5, 7, 9, 5, 5, 6, 7}, {5, 67, 78, 8, 3, 0, 4, 0}};
MatrixForm[matriz8]
```
forma de matriz
```
Timing[maximo[matriz8]]
```
cronometra
```
MatrixForm[ady8 = matrizady[matriz8]];
```
forma de matriz
```
AdjacencyGraph[ady8, VertexLabels → Automatic]
```
grafo con adyacencia | etiquetas de vérti··· | automático

Out[ ]//MatrixForm=

2	4	5	7	9	4	3	2
2	4	6	8	43	2	5	6
3	5	7	4	32	23	5	5
5	3	35	4	7	65	8	53
5	6	8	5	3	3	6	9
4	5	6	88	9	4	3	3
3	5	7	9	5	5	6	7
5	67	78	8	3	0	4	0

Out[ ]= {39.9531, {511, {7, 6, 4, 8, 1, 2, 3, 5}}}

Out[ ]=



In[ ]:=
```
caminos8 = ReadList["C:/Users/Usuario/Desktop/tfg/graph8.txt"];
```
lee lista | constante
```
Timing[maxcamino[caminos8, matriz8]]
```
cronometra

Out[ ]= {0.203125, {508, {{1, 2, 3, 6, 4, 5, 7, 8}, {1, 2, 4, 3, 5, 7, 6, 8},
    {1, 2, 4, 3, 6, 5, 7, 8}, {1, 2, 4, 5, 7, 6, 8, 3}, {1, 2, 4, 5, 7, 8, 3, 6},
    {1, 2, 4, 7, 6, 8, 3, 5}, {1, 2, 4, 7, 8, 3, 6, 5}, {1, 4, 7, 6, 8, 2, 3, 5},
    {1, 4, 7, 8, 2, 3, 6, 5}, {1, 4, 7, 8, 3, 6, 2, 5}, {1, 6, 4, 7, 8, 2, 3, 5},
    {6, 1, 4, 7, 8, 2, 3, 5}, {6, 4, 7, 8, 1, 2, 3, 5}, {6, 4, 7, 8, 1, 2, 3, 5}}}}

```
In[ ]:= matriz9 = {{3, 5, 7, 43, 34, 6, 32, 34, 45}, {54, 4, 4, 53, 67, 7, 7, 7, 0},
    {5, 5, 67, 73, 3, 7, 54, 4, 5}, {0, 5, 6, 8, 67, 6, 54, 53, 42},
    {5, 4, 56, 7, 77, 5, 4, 4, 7}, {4, 6, 6, 53, 4, 42, 3, 42, 42}, {0, 6, 4, 3, 2, 5, 8, 9, 0},
    {0, 12, 5, 56, 45, 6, 42, 3, 2}, {34, 4, 45, 75, 23, 4, 12, 3, 12}};
MatrixForm[matriz9]
  forma de matriz
Timing[maximo[matriz9]]
  cronometra
MatrixForm[ady9 = matrizady[matriz9]];
  forma de matriz
AdjacencyGraph[ady9, VertexLabels → Automatic]
  grafo con adyacencia   etiquetas de vérti···   automático
```

Out[ ]//MatrixForm=

$$\begin{pmatrix} 3 & 5 & 7 & 43 & 34 & 6 & 32 & 34 & 45 \\ 54 & 4 & 4 & 53 & 67 & 7 & 7 & 7 & 0 \\ 5 & 5 & 67 & 73 & 3 & 7 & 54 & 4 & 5 \\ 0 & 5 & 6 & 8 & 67 & 6 & 54 & 53 & 42 \\ 5 & 4 & 56 & 7 & 77 & 5 & 4 & 4 & 7 \\ 4 & 6 & 6 & 53 & 4 & 42 & 3 & 42 & 42 \\ 0 & 6 & 4 & 3 & 2 & 5 & 8 & 9 & 0 \\ 0 & 12 & 5 & 56 & 45 & 6 & 42 & 3 & 2 \\ 34 & 4 & 45 & 75 & 23 & 4 & 12 & 3 & 12 \end{pmatrix}$$

```
In[ ]:= caminos9 = ReadList["C:/Users/Usuario/Desktop/tfg/graph9.txt"];
          lee lista   constante
Timing[maxcamino[caminos9, matriz9]]
  cronometra
```

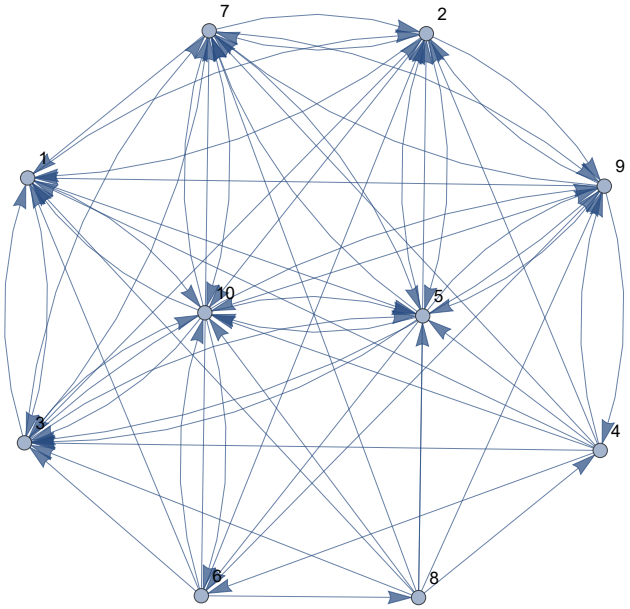
```
Out[ ]= {0.234375,
 {1111, {{1, 3, 2, 4, 5, 6, 9, 8, 7}, {1, 3, 2, 4, 7, 6, 9, 8, 5}, {1, 3, 2, 5, 6, 9, 8, 4, 7},
   {1, 3, 2, 6, 9, 8, 4, 5, 7}, {1, 3, 6, 9, 8, 2, 4, 5, 7}, {1, 6, 9, 8, 2, 4, 5, 3, 7},
   {1, 6, 9, 8, 2, 5, 3, 4, 7}, {1, 6, 9, 8, 3, 2, 4, 5, 7}, {2, 1, 3, 6, 9, 8, 4, 5, 7},
   {2, 1, 6, 9, 8, 3, 4, 5, 7}, {2, 1, 6, 9, 8, 3, 4, 5, 7}}}}
```

In[ ]:=
```
matriz10 = {{30707, 0, 0, 0, 3, 0, 0, 0, 0, 0}, {0, 184, 0, 0, 0, 0, 0, 0, 0, 2},
    {0, 21, 240, 0, 0, 0, 0, 0, 0, 0}, {21, 11, 13, 4458, 8, 16, 2, 0, 0, 12},
    {0, 0, 0, 0, 5725, 50, 0, 0, 0, 0}, {241, 15, 3, 0, 0, 189, 7, 1917, 1697, 0},
    {1, 0, 0, 0, 0, 0, 75, 0, 0, 0}, {1059, 19, 9, 146, 4313, 0, 15, 594, 211, 67},
    {18, 0, 0, 0, 0, 0, 0, 23, 0, 1538}, {0, 0, 0, 0, 0, 0, 0, 10, 0, 1}};
MatrixForm[matriz10]
```
forma de matriz
```
(*Timing[maximo[matriz10]]*)
```
cronometra
```
MatrixForm[ady10 = matrizady[matriz10]];
```
forma de matriz
```
AdjacencyGraph[ady10, VertexLabels → Automatic]
```
grafo con adyacencia | etiquetas de vérti··· | automático

Out[ ]//MatrixForm=

30707	0	0	0	3	0	0	0	0	0
0	184	0	0	0	0	0	0	0	2
0	21	240	0	0	0	0	0	0	0
21	11	13	4458	8	16	2	0	0	12
0	0	0	0	5725	50	0	0	0	0
241	15	3	0	0	189	7	1917	1697	0
1	0	0	0	0	0	75	0	0	0
1059	19	9	146	4313	0	15	594	211	67
18	0	0	0	0	0	0	23	0	1538
0	0	0	0	0	0	0	10	0	1

Out[ ]=



In[ ]:=
```
caminos10 = ReadList["C:/Users/Usuario/Desktop/tfg/graph10.txt"];
```
lee lista | constante
```
Timing[maxcamino[caminos10, matriz10]]
```
cronometra

Out[ ]= {14.0156, {11369, {{1, 2, 5, 6, 8, 3, 7, 9, 4, 10}, {1, 2, 5, 6, 8, 3, 9, 4, 7, 10},
    {1, 2, 5, 6, 8, 3, 9, 4, 10, 7}, {1, 2, 5, 6, 8, 4, 3, 7, 9, 10},
    {1, 2, 5, 6, 8, 4, 3, 9, 7, 10}, {1, 2, 5, 6, 8, 4, 3, 9, 10, 7},
    {1, 2, 5, 6, 8, 4, 7, 3, 9, 10}, {1, 2, 5, 6, 8, 4, 7, 9, 3, 10},

{1, 2, 5, 6, 8, 4, 7, 9, 10, 3}, {1, 2, 5, 6, 8, 4, 9, 3, 7, 10},
{1, 2, 5, 6, 8, 4, 9, 3, 10, 7}, {1, 2, 5, 6, 8, 4, 9, 7, 3, 10},
{1, 2, 5, 6, 8, 4, 9, 7, 10, 3}, {1, 2, 5, 6, 8, 4, 9, 10, 3, 7},
{1, 2, 5, 6, 8, 4, 9, 10, 7, 3}, {1, 2, 5, 6, 8, 9, 4, 3, 7, 10},
{1, 2, 5, 6, 8, 9, 4, 3, 10, 7}, {1, 2, 5, 6, 8, 9, 4, 7, 3, 10},
{1, 2, 5, 6, 8, 9, 4, 7, 10, 3}, {1, 2, 5, 6, 8, 9, 4, 10, 3, 7},
{1, 2, 5, 6, 8, 9, 4, 10, 7, 3}, {1, 2, 7, 3, 9, 4, 6, 8, 5, 10},
{1, 2, 7, 3, 9, 4, 6, 8, 10, 5}, {1, 2, 7, 3, 9, 10, 6, 8, 4, 5},
{1, 2, 7, 3, 10, 6, 8, 4, 5, 9}, {1, 2, 7, 3, 10, 6, 8, 4, 9, 5},
{1, 2, 7, 3, 10, 6, 8, 9, 4, 5}, {1, 2, 7, 10, 6, 8, 3, 5, 9, 4},
{1, 2, 7, 10, 6, 8, 3, 9, 4, 5}, {1, 2, 7, 10, 6, 8, 4, 3, 5, 9},
{1, 2, 7, 10, 6, 8, 4, 3, 9, 5}, {1, 2, 7, 10, 6, 8, 4, 5, 9, 3},
{1, 2, 7, 10, 6, 8, 4, 9, 3, 5}, {1, 2, 7, 10, 6, 8, 9, 4, 3, 5},
{1, 2, 10, 6, 8, 3, 5, 7, 9, 4}, {1, 2, 10, 6, 8, 3, 5, 9, 4, 7},
{1, 2, 10, 6, 8, 3, 7, 9, 4, 5}, {1, 2, 10, 6, 8, 3, 9, 4, 5, 7},
{1, 2, 10, 6, 8, 3, 9, 4, 7, 5}, {1, 2, 10, 6, 8, 4, 3, 5, 7, 9},
{1, 2, 10, 6, 8, 4, 3, 5, 9, 7}, {1, 2, 10, 6, 8, 4, 3, 7, 5, 9},
{1, 2, 10, 6, 8, 4, 3, 7, 9, 5}, {1, 2, 10, 6, 8, 4, 3, 9, 5, 7},
{1, 2, 10, 6, 8, 4, 3, 9, 7, 5}, {1, 2, 10, 6, 8, 4, 5, 7, 3, 9},
{1, 2, 10, 6, 8, 4, 5, 7, 9, 3}, {1, 2, 10, 6, 8, 4, 5, 9, 3, 7},
{1, 2, 10, 6, 8, 4, 5, 9, 7, 3}, {1, 2, 10, 6, 8, 4, 7, 3, 5, 9},
{1, 2, 10, 6, 8, 4, 7, 3, 9, 5}, {1, 2, 10, 6, 8, 4, 7, 5, 9, 3},
{1, 2, 10, 6, 8, 4, 7, 9, 3, 5}, {1, 2, 10, 6, 8, 4, 9, 3, 5, 7},
{1, 2, 10, 6, 8, 4, 9, 3, 7, 5}, {1, 2, 10, 6, 8, 4, 9, 5, 7, 3},
{1, 2, 10, 6, 8, 4, 9, 7, 3, 5}, {1, 2, 10, 6, 8, 9, 4, 3, 5, 7},
{1, 2, 10, 6, 8, 9, 4, 3, 7, 5}, {1, 2, 10, 6, 8, 9, 4, 5, 7, 3},
{1, 2, 10, 6, 8, 9, 4, 7, 3, 5}, {1, 3, 7, 10, 6, 8, 2, 9, 4, 5},
{1, 3, 7, 10, 6, 8, 4, 2, 5, 9}, {1, 3, 7, 10, 6, 8, 4, 2, 9, 5},
{1, 3, 7, 10, 6, 8, 4, 5, 9, 2}, {1, 3, 7, 10, 6, 8, 4, 9, 2, 5},
{1, 3, 7, 10, 6, 8, 9, 4, 2, 5}, {1, 3, 10, 6, 8, 2, 5, 7, 9, 4},
{1, 3, 10, 6, 8, 2, 5, 9, 4, 7}, {1, 3, 10, 6, 8, 2, 7, 9, 4, 5},
{1, 3, 10, 6, 8, 2, 9, 4, 5, 7}, {1, 3, 10, 6, 8, 2, 9, 4, 7, 5},
{1, 3, 10, 6, 8, 4, 2, 5, 7, 9}, {1, 3, 10, 6, 8, 4, 2, 5, 9, 7},
{1, 3, 10, 6, 8, 4, 2, 7, 5, 9}, {1, 3, 10, 6, 8, 4, 2, 7, 9, 5},
{1, 3, 10, 6, 8, 4, 2, 9, 5, 7}, {1, 3, 10, 6, 8, 4, 2, 9, 7, 5},
{1, 3, 10, 6, 8, 4, 5, 7, 2, 9}, {1, 3, 10, 6, 8, 4, 5, 7, 9, 2},
{1, 3, 10, 6, 8, 4, 5, 9, 2, 7}, {1, 3, 10, 6, 8, 4, 5, 9, 7, 2},
{1, 3, 10, 6, 8, 4, 7, 2, 5, 9}, {1, 3, 10, 6, 8, 4, 7, 2, 9, 5},
{1, 3, 10, 6, 8, 4, 7, 5, 9, 2}, {1, 3, 10, 6, 8, 4, 7, 9, 2, 5},
{1, 3, 10, 6, 8, 4, 9, 2, 5, 7}, {1, 3, 10, 6, 8, 4, 9, 2, 7, 5},
{1, 3, 10, 6, 8, 4, 9, 5, 7, 2}, {1, 3, 10, 6, 8, 4, 9, 7, 2, 5},
{1, 3, 10, 6, 8, 9, 4, 2, 5, 7}, {1, 3, 10, 6, 8, 9, 4, 2, 7, 5},
{1, 3, 10, 6, 8, 9, 4, 5, 7, 2}, {1, 3, 10, 6, 8, 9, 4, 7, 2, 5},
{1, 10, 6, 8, 3, 2, 9, 4, 5, 7}, {1, 10, 6, 8, 3, 2, 9, 4, 7, 5},
{1, 10, 6, 8, 3, 5, 7, 9, 4, 2}, {1, 10, 6, 8, 3, 5, 9, 4, 2, 7},
{1, 10, 6, 8, 3, 5, 9, 4, 7, 2}, {1, 10, 6, 8, 3, 7, 9, 4, 2, 5},
{1, 10, 6, 8, 3, 9, 4, 2, 5, 7}, {1, 10, 6, 8, 3, 9, 4, 2, 7, 5},
{1, 10, 6, 8, 3, 9, 4, 5, 7, 2}, {1, 10, 6, 8, 3, 9, 4, 7, 2, 5},
{1, 10, 6, 8, 4, 3, 2, 5, 7, 9}, {1, 10, 6, 8, 4, 3, 2, 5, 9, 7},
{1, 10, 6, 8, 4, 3, 2, 7, 5, 9}, {1, 10, 6, 8, 4, 3, 2, 7, 9, 5},
{1, 10, 6, 8, 4, 3, 2, 9, 5, 7}, {1, 10, 6, 8, 4, 3, 2, 9, 7, 5},
{1, 10, 6, 8, 4, 3, 5, 7, 2, 9}, {1, 10, 6, 8, 4, 3, 5, 7, 9, 2},
{1, 10, 6, 8, 4, 3, 5, 9, 2, 7}, {1, 10, 6, 8, 4, 3, 5, 9, 7, 2},
{1, 10, 6, 8, 4, 3, 7, 2, 5, 9}, {1, 10, 6, 8, 4, 3, 7, 2, 9, 5},
{1, 10, 6, 8, 4, 3, 7, 5, 9, 2}, {1, 10, 6, 8, 4, 3, 7, 9, 2, 5},

{1, 10, 6, 8, 4, 3, 9, 2, 5, 7}, {1, 10, 6, 8, 4, 3, 9, 2, 7, 5},
{1, 10, 6, 8, 4, 3, 9, 5, 7, 2}, {1, 10, 6, 8, 4, 3, 9, 7, 2, 5},
{1, 10, 6, 8, 4, 5, 7, 3, 2, 9}, {1, 10, 6, 8, 4, 5, 7, 3, 9, 2},
{1, 10, 6, 8, 4, 5, 7, 9, 3, 2}, {1, 10, 6, 8, 4, 5, 9, 3, 2, 7},
{1, 10, 6, 8, 4, 5, 9, 3, 7, 2}, {1, 10, 6, 8, 4, 5, 9, 7, 3, 2},
{1, 10, 6, 8, 4, 7, 3, 2, 5, 9}, {1, 10, 6, 8, 4, 7, 3, 2, 9, 5},
{1, 10, 6, 8, 4, 7, 3, 5, 9, 2}, {1, 10, 6, 8, 4, 7, 3, 9, 2, 5},
{1, 10, 6, 8, 4, 7, 5, 9, 3, 2}, {1, 10, 6, 8, 4, 7, 9, 3, 2, 5},
{1, 10, 6, 8, 4, 9, 3, 2, 5, 7}, {1, 10, 6, 8, 4, 9, 3, 2, 7, 5},
{1, 10, 6, 8, 4, 9, 3, 5, 7, 2}, {1, 10, 6, 8, 4, 9, 3, 7, 2, 5},
{1, 10, 6, 8, 4, 9, 5, 7, 3, 2}, {1, 10, 6, 8, 4, 9, 7, 3, 2, 5},
{1, 10, 6, 8, 9, 4, 3, 2, 5, 7}, {1, 10, 6, 8, 9, 4, 3, 2, 7, 5},
{1, 10, 6, 8, 9, 4, 3, 5, 7, 2}, {1, 10, 6, 8, 9, 4, 3, 7, 2, 5},
{1, 10, 6, 8, 9, 4, 5, 7, 3, 2}, {1, 10, 6, 8, 9, 4, 7, 3, 2, 5},
{2, 7, 3, 9, 4, 6, 8, 1, 5, 10}, {2, 7, 3, 9, 4, 6, 8, 1, 10, 5},
{2, 7, 3, 9, 4, 6, 8, 10, 1, 5}, {2, 7, 3, 9, 10, 6, 8, 4, 1, 5},
{2, 7, 3, 10, 6, 8, 1, 5, 9, 4}, {2, 7, 3, 10, 6, 8, 4, 1, 5, 9},
{2, 7, 3, 10, 6, 8, 4, 5, 9, 1}, {2, 7, 3, 10, 6, 8, 4, 9, 1, 5},
{2, 7, 3, 10, 6, 8, 9, 4, 1, 5}, {2, 7, 10, 6, 8, 3, 5, 9, 4, 1},
{2, 7, 10, 6, 8, 3, 9, 4, 1, 5}, {2, 7, 10, 6, 8, 4, 3, 5, 9, 1},
{2, 7, 10, 6, 8, 4, 3, 9, 1, 5}, {2, 7, 10, 6, 8, 4, 9, 1, 3, 5},
{2, 7, 10, 6, 8, 4, 9, 3, 1, 5}, {2, 7, 10, 6, 8, 9, 4, 1, 3, 5},
{2, 7, 10, 6, 8, 9, 4, 3, 1, 5}, {2, 10, 6, 8, 3, 5, 9, 4, 7, 1},
{2, 10, 6, 8, 3, 7, 9, 4, 1, 5}, {2, 10, 6, 8, 3, 9, 4, 1, 5, 7},
{2, 10, 6, 8, 3, 9, 4, 7, 1, 5}, {2, 10, 6, 8, 4, 3, 5, 7, 9, 1},
{2, 10, 6, 8, 4, 3, 5, 9, 7, 1}, {2, 10, 6, 8, 4, 3, 7, 5, 9, 1},
{2, 10, 6, 8, 4, 3, 7, 9, 1, 5}, {2, 10, 6, 8, 4, 3, 9, 7, 1, 5},
{2, 10, 6, 8, 4, 7, 3, 9, 1, 5}, {2, 10, 6, 8, 4, 7, 9, 1, 3, 5},
{2, 10, 6, 8, 4, 7, 9, 3, 1, 5}, {2, 10, 6, 8, 4, 9, 3, 7, 1, 5},
{2, 10, 6, 8, 4, 9, 7, 1, 3, 5}, {2, 10, 6, 8, 4, 9, 7, 3, 1, 5},
{2, 10, 6, 8, 9, 4, 3, 7, 1, 5}, {2, 10, 6, 8, 9, 4, 7, 1, 3, 5},
{2, 10, 6, 8, 9, 4, 7, 3, 1, 5}, {3, 7, 10, 6, 8, 2, 9, 4, 1, 5},
{3, 7, 10, 6, 8, 4, 2, 5, 9, 1}, {3, 7, 10, 6, 8, 4, 2, 9, 1, 5},
{3, 7, 10, 6, 8, 4, 9, 1, 2, 5}, {3, 7, 10, 6, 8, 4, 9, 2, 1, 5},
{3, 7, 10, 6, 8, 9, 4, 1, 2, 5}, {3, 7, 10, 6, 8, 9, 4, 2, 1, 5},
{3, 10, 6, 8, 2, 5, 7, 9, 4, 1}, {3, 10, 6, 8, 2, 5, 9, 4, 7, 1},
{3, 10, 6, 8, 2, 7, 9, 4, 1, 5}, {3, 10, 6, 8, 2, 9, 4, 1, 5, 7},
{3, 10, 6, 8, 2, 9, 4, 7, 1, 5}, {3, 10, 6, 8, 4, 2, 5, 7, 9, 1},
{3, 10, 6, 8, 4, 2, 5, 9, 7, 1}, {3, 10, 6, 8, 4, 2, 7, 5, 9, 1},
{3, 10, 6, 8, 4, 2, 7, 9, 1, 5}, {3, 10, 6, 8, 4, 2, 9, 7, 1, 5},
{3, 10, 6, 8, 4, 7, 2, 9, 1, 5}, {3, 10, 6, 8, 4, 7, 9, 1, 2, 5},
{3, 10, 6, 8, 4, 7, 9, 2, 1, 5}, {3, 10, 6, 8, 4, 9, 2, 7, 1, 5},
{3, 10, 6, 8, 4, 9, 7, 1, 2, 5}, {3, 10, 6, 8, 4, 9, 7, 2, 1, 5},
{3, 10, 6, 8, 9, 4, 2, 7, 1, 5}, {3, 10, 6, 8, 9, 4, 7, 1, 2, 5},
{3, 10, 6, 8, 9, 4, 7, 2, 1, 5}, {4, 6, 8, 1, 2, 5, 7, 3, 9, 10},
{4, 6, 8, 1, 2, 5, 7, 9, 3, 10}, {4, 6, 8, 1, 2, 5, 7, 9, 10, 3},
{4, 6, 8, 1, 2, 5, 9, 3, 7, 10}, {4, 6, 8, 1, 2, 5, 9, 3, 10, 7},
{4, 6, 8, 1, 2, 5, 9, 7, 3, 10}, {4, 6, 8, 1, 2, 5, 9, 7, 10, 3},
{4, 6, 8, 1, 2, 5, 9, 10, 3, 7}, {4, 6, 8, 1, 2, 5, 9, 10, 7, 3},
{4, 6, 8, 1, 2, 7, 3, 5, 9, 10}, {4, 6, 8, 1, 2, 7, 3, 9, 5, 10},
{4, 6, 8, 1, 2, 7, 3, 9, 10, 5}, {4, 6, 8, 1, 2, 7, 5, 9, 3, 10},
{4, 6, 8, 1, 2, 7, 5, 9, 10, 3}, {4, 6, 8, 1, 2, 7, 9, 3, 5, 10},
{4, 6, 8, 1, 2, 7, 9, 3, 10, 5}, {4, 6, 8, 1, 2, 7, 9, 5, 10, 3},
{4, 6, 8, 1, 2, 7, 9, 10, 3, 5}, {4, 6, 8, 1, 2, 9, 3, 5, 7, 10},
{4, 6, 8, 1, 2, 9, 3, 5, 10, 7}, {4, 6, 8, 1, 2, 9, 3, 7, 5, 10},

{4, 6, 8, 1, 2, 9, 3, 7, 10, 5}, {4, 6, 8, 1, 2, 9, 3, 10, 5, 7},
{4, 6, 8, 1, 2, 9, 3, 10, 7, 5}, {4, 6, 8, 1, 2, 9, 5, 7, 3, 10},
{4, 6, 8, 1, 2, 9, 5, 7, 10, 3}, {4, 6, 8, 1, 2, 9, 5, 10, 3, 7},
{4, 6, 8, 1, 2, 9, 5, 10, 7, 3}, {4, 6, 8, 1, 2, 9, 7, 3, 5, 10},
{4, 6, 8, 1, 2, 9, 7, 3, 10, 5}, {4, 6, 8, 1, 2, 9, 7, 5, 10, 3},
{4, 6, 8, 1, 2, 9, 7, 10, 3, 5}, {4, 6, 8, 1, 2, 9, 10, 3, 5, 7},
{4, 6, 8, 1, 2, 9, 10, 3, 7, 5}, {4, 6, 8, 1, 2, 9, 10, 5, 7, 3},
{4, 6, 8, 1, 2, 9, 10, 7, 3, 5}, {4, 6, 8, 1, 3, 2, 5, 7, 9, 10},
{4, 6, 8, 1, 3, 2, 5, 9, 7, 10}, {4, 6, 8, 1, 3, 2, 5, 9, 10, 7},
{4, 6, 8, 1, 3, 2, 7, 5, 9, 10}, {4, 6, 8, 1, 3, 2, 7, 9, 5, 10},
{4, 6, 8, 1, 3, 2, 7, 9, 10, 5}, {4, 6, 8, 1, 3, 2, 9, 5, 7, 10},
{4, 6, 8, 1, 3, 2, 9, 5, 10, 7}, {4, 6, 8, 1, 3, 2, 9, 7, 5, 10},
{4, 6, 8, 1, 3, 2, 9, 7, 10, 5}, {4, 6, 8, 1, 3, 2, 9, 10, 5, 7},
{4, 6, 8, 1, 3, 2, 9, 10, 7, 5}, {4, 6, 8, 1, 3, 5, 7, 2, 9, 10},
{4, 6, 8, 1, 3, 5, 7, 9, 2, 10}, {4, 6, 8, 1, 3, 5, 9, 2, 7, 10},
{4, 6, 8, 1, 3, 5, 9, 2, 10, 7}, {4, 6, 8, 1, 3, 5, 9, 7, 2, 10},
{4, 6, 8, 1, 3, 7, 2, 5, 9, 10}, {4, 6, 8, 1, 3, 7, 2, 9, 5, 10},
{4, 6, 8, 1, 3, 7, 2, 9, 10, 5}, {4, 6, 8, 1, 3, 7, 5, 9, 2, 10},
{4, 6, 8, 1, 3, 7, 9, 2, 5, 10}, {4, 6, 8, 1, 3, 7, 9, 2, 10, 5},
{4, 6, 8, 1, 3, 9, 2, 5, 7, 10}, {4, 6, 8, 1, 3, 9, 2, 5, 10, 7},
{4, 6, 8, 1, 3, 9, 2, 7, 5, 10}, {4, 6, 8, 1, 3, 9, 2, 7, 10, 5},
{4, 6, 8, 1, 3, 9, 2, 10, 5, 7}, {4, 6, 8, 1, 3, 9, 2, 10, 7, 5},
{4, 6, 8, 1, 3, 9, 5, 7, 2, 10}, {4, 6, 8, 1, 3, 9, 7, 2, 5, 10},
{4, 6, 8, 1, 3, 9, 7, 2, 10, 5}, {4, 6, 8, 1, 5, 7, 3, 2, 9, 10},
{4, 6, 8, 1, 5, 7, 3, 9, 2, 10}, {4, 6, 8, 1, 5, 7, 9, 3, 2, 10},
{4, 6, 8, 1, 5, 9, 3, 2, 7, 10}, {4, 6, 8, 1, 5, 9, 3, 2, 10, 7},
{4, 6, 8, 1, 5, 9, 3, 7, 2, 10}, {4, 6, 8, 1, 5, 9, 7, 3, 2, 10},
{4, 6, 8, 3, 1, 2, 5, 7, 9, 10}, {4, 6, 8, 3, 1, 2, 5, 9, 7, 10},
{4, 6, 8, 3, 1, 2, 5, 9, 10, 7}, {4, 6, 8, 3, 1, 2, 7, 5, 9, 10},
{4, 6, 8, 3, 1, 2, 7, 9, 5, 10}, {4, 6, 8, 3, 1, 2, 7, 9, 10, 5},
{4, 6, 8, 3, 1, 2, 9, 5, 7, 10}, {4, 6, 8, 3, 1, 2, 9, 5, 10, 7},
{4, 6, 8, 3, 1, 2, 9, 7, 5, 10}, {4, 6, 8, 3, 1, 2, 9, 7, 10, 5},
{4, 6, 8, 3, 1, 2, 9, 10, 5, 7}, {4, 6, 8, 3, 1, 2, 9, 10, 7, 5},
{4, 6, 8, 3, 1, 5, 7, 2, 9, 10}, {4, 6, 8, 3, 1, 5, 7, 9, 2, 10},
{4, 6, 8, 3, 1, 5, 9, 2, 7, 10}, {4, 6, 8, 3, 1, 5, 9, 2, 10, 7},
{4, 6, 8, 3, 1, 5, 9, 7, 2, 10}, {4, 6, 8, 3, 2, 1, 5, 7, 9, 10},
{4, 6, 8, 3, 2, 1, 5, 9, 7, 10}, {4, 6, 8, 3, 2, 1, 5, 9, 10, 7},
{4, 6, 8, 3, 2, 5, 7, 9, 1, 10}, {4, 6, 8, 3, 2, 5, 7, 9, 10, 1},
{4, 6, 8, 3, 2, 5, 9, 7, 1, 10}, {4, 6, 8, 3, 2, 5, 9, 7, 10, 1},
{4, 6, 8, 3, 2, 5, 9, 10, 7, 1}, {4, 6, 8, 3, 2, 7, 5, 9, 1, 10},
{4, 6, 8, 3, 2, 7, 5, 9, 10, 1}, {4, 6, 8, 3, 2, 7, 9, 1, 5, 10},
{4, 6, 8, 3, 2, 7, 9, 1, 10, 5}, {4, 6, 8, 3, 2, 7, 9, 10, 1, 5},
{4, 6, 8, 3, 2, 9, 7, 1, 5, 10}, {4, 6, 8, 3, 2, 9, 7, 1, 10, 5},
{4, 6, 8, 3, 2, 9, 7, 10, 1, 5}, {4, 6, 8, 3, 2, 9, 10, 7, 1, 5},
{4, 6, 8, 3, 7, 2, 9, 1, 5, 10}, {4, 6, 8, 3, 7, 2, 9, 1, 10, 5},
{4, 6, 8, 3, 7, 2, 9, 10, 1, 5}, {4, 6, 8, 3, 7, 9, 1, 2, 5, 10},
{4, 6, 8, 3, 7, 9, 1, 2, 10, 5}, {4, 6, 8, 3, 7, 9, 2, 1, 5, 10},
{4, 6, 8, 3, 7, 9, 2, 1, 10, 5}, {4, 6, 8, 3, 7, 9, 2, 10, 1, 5},
{4, 6, 8, 3, 9, 2, 7, 1, 5, 10}, {4, 6, 8, 3, 9, 2, 7, 1, 10, 5},
{4, 6, 8, 3, 9, 2, 7, 10, 1, 5}, {4, 6, 8, 3, 9, 2, 10, 7, 1, 5},
{4, 6, 8, 3, 9, 7, 1, 2, 5, 10}, {4, 6, 8, 3, 9, 7, 1, 2, 10, 5},
{4, 6, 8, 3, 9, 7, 2, 1, 5, 10}, {4, 6, 8, 3, 9, 7, 2, 1, 10, 5},
{4, 6, 8, 3, 9, 7, 2, 10, 1, 5}, {4, 6, 8, 7, 3, 2, 9, 1, 5, 10},
{4, 6, 8, 7, 3, 2, 9, 1, 10, 5}, {4, 6, 8, 7, 3, 2, 9, 10, 1, 5},
{4, 6, 8, 7, 3, 9, 1, 2, 5, 10}, {4, 6, 8, 7, 3, 9, 1, 2, 10, 5},

{4, 6, 8, 7, 3, 9, 2, 1, 5, 10}, {4, 6, 8, 7, 3, 9, 2, 1, 10, 5},
{4, 6, 8, 7, 3, 9, 2, 10, 1, 5}, {4, 6, 8, 7, 9, 1, 3, 2, 5, 10},
{4, 6, 8, 7, 9, 1, 3, 2, 10, 5}, {4, 6, 8, 7, 9, 3, 1, 2, 5, 10},
{4, 6, 8, 7, 9, 3, 1, 2, 10, 5}, {4, 6, 8, 7, 9, 3, 2, 1, 5, 10},
{4, 6, 8, 7, 9, 3, 2, 1, 10, 5}, {4, 6, 8, 7, 9, 3, 2, 10, 1, 5},
{4, 6, 8, 9, 3, 2, 7, 1, 5, 10}, {4, 6, 8, 9, 3, 2, 7, 1, 10, 5},
{4, 6, 8, 9, 3, 2, 7, 10, 1, 5}, {4, 6, 8, 9, 3, 2, 10, 7, 1, 5},
{4, 6, 8, 9, 3, 7, 1, 2, 5, 10}, {4, 6, 8, 9, 3, 7, 1, 2, 10, 5},
{4, 6, 8, 9, 3, 7, 2, 1, 5, 10}, {4, 6, 8, 9, 3, 7, 2, 1, 10, 5},
{4, 6, 8, 9, 3, 7, 2, 10, 1, 5}, {4, 6, 8, 9, 7, 1, 3, 2, 5, 10},
{4, 6, 8, 9, 7, 1, 3, 2, 10, 5}, {4, 6, 8, 9, 7, 3, 1, 2, 5, 10},
{4, 6, 8, 9, 7, 3, 1, 2, 10, 5}, {4, 6, 8, 9, 7, 3, 2, 1, 5, 10},
{4, 6, 8, 9, 7, 3, 2, 1, 10, 5}, {4, 6, 8, 9, 7, 3, 2, 10, 1, 5},
{6, 8, 1, 2, 5, 7, 3, 9, 4, 10}, {6, 8, 1, 2, 5, 7, 9, 4, 3, 10},
{6, 8, 1, 2, 5, 7, 9, 4, 10, 3}, {6, 8, 1, 2, 5, 9, 4, 3, 7, 10},
{6, 8, 1, 2, 5, 9, 4, 3, 10, 7}, {6, 8, 1, 2, 5, 9, 4, 7, 3, 10},
{6, 8, 1, 2, 5, 9, 4, 7, 10, 3}, {6, 8, 1, 2, 5, 9, 4, 10, 3, 7},
{6, 8, 1, 2, 5, 9, 4, 10, 7, 3}, {6, 8, 1, 2, 7, 9, 4, 3, 5, 10},
{6, 8, 1, 2, 7, 9, 4, 3, 10, 5}, {6, 8, 1, 2, 7, 9, 4, 5, 10, 3},
{6, 8, 1, 2, 7, 9, 4, 10, 3, 5}, {6, 8, 1, 2, 9, 4, 3, 5, 7, 10},
{6, 8, 1, 2, 9, 4, 3, 5, 10, 7}, {6, 8, 1, 2, 9, 4, 3, 7, 5, 10},
{6, 8, 1, 2, 9, 4, 3, 7, 10, 5}, {6, 8, 1, 2, 9, 4, 3, 10, 5, 7},
{6, 8, 1, 2, 9, 4, 3, 10, 7, 5}, {6, 8, 1, 2, 9, 4, 5, 7, 3, 10},
{6, 8, 1, 2, 9, 4, 5, 7, 10, 3}, {6, 8, 1, 2, 9, 4, 5, 10, 3, 7},
{6, 8, 1, 2, 9, 4, 5, 10, 7, 3}, {6, 8, 1, 2, 9, 4, 7, 3, 5, 10},
{6, 8, 1, 2, 9, 4, 7, 3, 10, 5}, {6, 8, 1, 2, 9, 4, 7, 5, 10, 3},
{6, 8, 1, 2, 9, 4, 7, 10, 3, 5}, {6, 8, 1, 2, 9, 4, 10, 3, 5, 7},
{6, 8, 1, 2, 9, 4, 10, 3, 7, 5}, {6, 8, 1, 2, 9, 4, 10, 5, 7, 3},
{6, 8, 1, 2, 9, 4, 10, 7, 3, 5}, {6, 8, 1, 3, 2, 5, 9, 4, 7, 10},
{6, 8, 1, 3, 2, 5, 9, 4, 10, 7}, {6, 8, 1, 3, 2, 7, 9, 4, 5, 10},
{6, 8, 1, 3, 2, 7, 9, 4, 10, 5}, {6, 8, 1, 3, 2, 9, 4, 5, 7, 10},
{6, 8, 1, 3, 2, 9, 4, 5, 10, 7}, {6, 8, 1, 3, 2, 9, 4, 7, 5, 10},
{6, 8, 1, 3, 2, 9, 4, 7, 10, 5}, {6, 8, 1, 3, 2, 9, 4, 10, 5, 7},
{6, 8, 1, 3, 2, 9, 4, 10, 7, 5}, {6, 8, 1, 3, 5, 7, 9, 4, 2, 10},
{6, 8, 1, 3, 5, 9, 4, 2, 7, 10}, {6, 8, 1, 3, 5, 9, 4, 2, 10, 7},
{6, 8, 1, 3, 5, 9, 4, 7, 2, 10}, {6, 8, 1, 3, 7, 9, 4, 2, 5, 10},
{6, 8, 1, 3, 7, 9, 4, 2, 10, 5}, {6, 8, 1, 3, 9, 4, 2, 5, 7, 10},
{6, 8, 1, 3, 9, 4, 2, 5, 10, 7}, {6, 8, 1, 3, 9, 4, 2, 7, 5, 10},
{6, 8, 1, 3, 9, 4, 2, 7, 10, 5}, {6, 8, 1, 3, 9, 4, 2, 10, 5, 7},
{6, 8, 1, 3, 9, 4, 2, 10, 7, 5}, {6, 8, 1, 3, 9, 4, 5, 7, 2, 10},
{6, 8, 1, 3, 9, 4, 7, 2, 5, 10}, {6, 8, 1, 3, 9, 4, 7, 2, 10, 5},
{6, 8, 1, 5, 7, 9, 4, 3, 2, 10}, {6, 8, 1, 5, 9, 4, 3, 2, 7, 10},
{6, 8, 1, 5, 9, 4, 3, 2, 10, 7}, {6, 8, 1, 5, 9, 4, 3, 7, 2, 10},
{6, 8, 1, 5, 9, 4, 7, 3, 2, 10}, {6, 8, 2, 5, 7, 9, 4, 1, 3, 10},
{6, 8, 2, 5, 7, 9, 4, 1, 10, 3}, {6, 8, 2, 5, 7, 9, 4, 3, 1, 10},
{6, 8, 2, 5, 7, 9, 4, 3, 10, 1}, {6, 8, 2, 5, 7, 9, 4, 10, 1, 3},
{6, 8, 2, 5, 7, 9, 4, 10, 3, 1}, {6, 8, 2, 5, 9, 4, 1, 3, 7, 10},
{6, 8, 2, 5, 9, 4, 1, 3, 10, 7}, {6, 8, 2, 5, 9, 4, 1, 10, 3, 7},
{6, 8, 2, 5, 9, 4, 1, 10, 7, 3}, {6, 8, 2, 5, 9, 4, 3, 1, 10, 7},
{6, 8, 2, 5, 9, 4, 3, 7, 1, 10}, {6, 8, 2, 5, 9, 4, 3, 7, 10, 1},
{6, 8, 2, 5, 9, 4, 3, 10, 7, 1}, {6, 8, 2, 5, 9, 4, 7, 1, 3, 10},
{6, 8, 2, 5, 9, 4, 7, 1, 10, 3}, {6, 8, 2, 5, 9, 4, 7, 3, 1, 10},
{6, 8, 2, 5, 9, 4, 7, 3, 10, 1}, {6, 8, 2, 5, 9, 4, 7, 10, 1, 3},
{6, 8, 2, 5, 9, 4, 7, 10, 3, 1}, {6, 8, 2, 5, 9, 4, 10, 3, 7, 1},
{6, 8, 2, 5, 9, 4, 10, 7, 1, 3}, {6, 8, 2, 5, 9, 4, 10, 7, 3, 1},

{6, 8, 2, 7, 9, 4, 1, 3, 5, 10}, {6, 8, 2, 7, 9, 4, 1, 3, 10, 5},
{6, 8, 2, 7, 9, 4, 1, 5, 10, 3}, {6, 8, 2, 7, 9, 4, 1, 10, 3, 5},
{6, 8, 2, 7, 9, 4, 3, 1, 5, 10}, {6, 8, 2, 7, 9, 4, 3, 1, 10, 5},
{6, 8, 2, 7, 9, 4, 3, 10, 1, 5}, {6, 8, 2, 7, 9, 4, 10, 1, 3, 5},
{6, 8, 2, 7, 9, 4, 10, 3, 1, 5}, {6, 8, 2, 9, 4, 1, 3, 5, 7, 10},
{6, 8, 2, 9, 4, 1, 3, 5, 10, 7}, {6, 8, 2, 9, 4, 1, 3, 7, 5, 10},
{6, 8, 2, 9, 4, 1, 3, 7, 10, 5}, {6, 8, 2, 9, 4, 1, 3, 10, 5, 7},
{6, 8, 2, 9, 4, 1, 3, 10, 7, 5}, {6, 8, 2, 9, 4, 1, 5, 7, 3, 10},
{6, 8, 2, 9, 4, 1, 5, 7, 10, 3}, {6, 8, 2, 9, 4, 1, 5, 10, 3, 7},
{6, 8, 2, 9, 4, 1, 5, 10, 7, 3}, {6, 8, 2, 9, 4, 1, 10, 3, 5, 7},
{6, 8, 2, 9, 4, 1, 10, 3, 7, 5}, {6, 8, 2, 9, 4, 1, 10, 5, 7, 3},
{6, 8, 2, 9, 4, 1, 10, 7, 3, 5}, {6, 8, 2, 9, 4, 3, 1, 5, 7, 10},
{6, 8, 2, 9, 4, 3, 1, 5, 10, 7}, {6, 8, 2, 9, 4, 3, 1, 10, 5, 7},
{6, 8, 2, 9, 4, 3, 1, 10, 7, 5}, {6, 8, 2, 9, 4, 3, 7, 1, 5, 10},
{6, 8, 2, 9, 4, 3, 7, 1, 10, 5}, {6, 8, 2, 9, 4, 3, 7, 10, 1, 5},
{6, 8, 2, 9, 4, 3, 10, 7, 1, 5}, {6, 8, 2, 9, 4, 7, 1, 3, 5, 10},
{6, 8, 2, 9, 4, 7, 1, 3, 10, 5}, {6, 8, 2, 9, 4, 7, 1, 5, 10, 3},
{6, 8, 2, 9, 4, 7, 1, 10, 3, 5}, {6, 8, 2, 9, 4, 7, 3, 1, 5, 10},
{6, 8, 2, 9, 4, 7, 3, 1, 10, 5}, {6, 8, 2, 9, 4, 7, 3, 10, 1, 5},
{6, 8, 2, 9, 4, 7, 10, 1, 3, 5}, {6, 8, 2, 9, 4, 7, 10, 3, 1, 5},
{6, 8, 2, 9, 4, 10, 3, 7, 1, 5}, {6, 8, 2, 9, 4, 10, 7, 1, 3, 5},
{6, 8, 2, 9, 4, 10, 7, 3, 1, 5}, {6, 8, 3, 2, 7, 9, 4, 1, 5, 10},
{6, 8, 3, 2, 7, 9, 4, 1, 10, 5}, {6, 8, 3, 2, 7, 9, 4, 10, 1, 5},
{6, 8, 3, 2, 9, 4, 1, 5, 7, 10}, {6, 8, 3, 2, 9, 4, 1, 5, 10, 7},
{6, 8, 3, 2, 9, 4, 1, 10, 5, 7}, {6, 8, 3, 2, 9, 4, 1, 10, 7, 5},
{6, 8, 3, 2, 9, 4, 7, 1, 5, 10}, {6, 8, 3, 2, 9, 4, 7, 1, 10, 5},
{6, 8, 3, 2, 9, 4, 7, 10, 1, 5}, {6, 8, 3, 2, 9, 4, 10, 7, 1, 5},
{6, 8, 3, 5, 9, 4, 2, 7, 1, 10}, {6, 8, 3, 5, 9, 4, 2, 7, 10, 1},
{6, 8, 3, 5, 9, 4, 2, 10, 7, 1}, {6, 8, 3, 5, 9, 4, 7, 1, 2, 10},
{6, 8, 3, 5, 9, 4, 7, 2, 1, 10}, {6, 8, 3, 5, 9, 4, 7, 2, 10, 1},
{6, 8, 3, 7, 9, 4, 1, 2, 5, 10}, {6, 8, 3, 7, 9, 4, 1, 2, 10, 5},
{6, 8, 3, 7, 9, 4, 2, 1, 5, 10}, {6, 8, 3, 7, 9, 4, 2, 1, 10, 5},
{6, 8, 3, 7, 9, 4, 2, 10, 1, 5}, {6, 8, 3, 9, 4, 1, 2, 5, 7, 10},
{6, 8, 3, 9, 4, 1, 2, 5, 10, 7}, {6, 8, 3, 9, 4, 1, 2, 7, 5, 10},
{6, 8, 3, 9, 4, 1, 2, 7, 10, 5}, {6, 8, 3, 9, 4, 1, 2, 10, 5, 7},
{6, 8, 3, 9, 4, 1, 2, 10, 7, 5}, {6, 8, 3, 9, 4, 1, 5, 7, 2, 10},
{6, 8, 3, 9, 4, 2, 1, 5, 7, 10}, {6, 8, 3, 9, 4, 2, 1, 5, 10, 7},
{6, 8, 3, 9, 4, 2, 1, 10, 5, 7}, {6, 8, 3, 9, 4, 2, 1, 10, 7, 5},
{6, 8, 3, 9, 4, 2, 7, 1, 5, 10}, {6, 8, 3, 9, 4, 2, 7, 1, 10, 5},
{6, 8, 3, 9, 4, 2, 7, 10, 1, 5}, {6, 8, 3, 9, 4, 2, 10, 7, 1, 5},
{6, 8, 3, 9, 4, 7, 1, 2, 5, 10}, {6, 8, 3, 9, 4, 7, 1, 2, 10, 5},
{6, 8, 3, 9, 4, 7, 2, 1, 5, 10}, {6, 8, 3, 9, 4, 7, 2, 1, 10, 5},
{6, 8, 3, 9, 4, 7, 2, 10, 1, 5}, {6, 8, 4, 3, 2, 5, 7, 9, 1, 10},
{6, 8, 4, 3, 2, 5, 7, 9, 10, 1}, {6, 8, 4, 3, 2, 5, 9, 7, 1, 10},
{6, 8, 4, 3, 2, 5, 9, 7, 10, 1}, {6, 8, 4, 3, 2, 5, 9, 10, 7, 1},
{6, 8, 4, 3, 2, 7, 5, 9, 1, 10}, {6, 8, 4, 3, 2, 7, 5, 9, 10, 1},
{6, 8, 4, 3, 2, 7, 9, 1, 5, 10}, {6, 8, 4, 3, 2, 7, 9, 1, 10, 5},
{6, 8, 4, 3, 2, 7, 9, 10, 1, 5}, {6, 8, 4, 3, 2, 9, 7, 1, 5, 10},
{6, 8, 4, 3, 2, 9, 7, 1, 10, 5}, {6, 8, 4, 3, 2, 9, 7, 10, 1, 5},
{6, 8, 4, 3, 2, 9, 10, 7, 1, 5}, {6, 8, 4, 3, 7, 2, 9, 1, 5, 10},
{6, 8, 4, 3, 7, 2, 9, 1, 10, 5}, {6, 8, 4, 3, 7, 2, 9, 10, 1, 5},
{6, 8, 4, 3, 7, 9, 1, 2, 5, 10}, {6, 8, 4, 3, 7, 9, 1, 2, 10, 5},
{6, 8, 4, 3, 7, 9, 2, 1, 5, 10}, {6, 8, 4, 3, 7, 9, 2, 1, 10, 5},
{6, 8, 4, 3, 7, 9, 2, 10, 1, 5}, {6, 8, 4, 3, 9, 2, 7, 1, 5, 10},
{6, 8, 4, 3, 9, 2, 7, 1, 10, 5}, {6, 8, 4, 3, 9, 2, 7, 10, 1, 5},

{6, 8, 4, 3, 9, 2, 10, 7, 1, 5}, {6, 8, 4, 3, 9, 7, 1, 2, 5, 10},
{6, 8, 4, 3, 9, 7, 1, 2, 10, 5}, {6, 8, 4, 3, 9, 7, 2, 1, 5, 10},
{6, 8, 4, 3, 9, 7, 2, 1, 10, 5}, {6, 8, 4, 3, 9, 7, 2, 10, 1, 5},
{6, 8, 4, 7, 3, 2, 9, 1, 5, 10}, {6, 8, 4, 7, 3, 2, 9, 1, 10, 5},
{6, 8, 4, 7, 3, 2, 9, 10, 1, 5}, {6, 8, 4, 7, 3, 9, 1, 2, 5, 10},
{6, 8, 4, 7, 3, 9, 1, 2, 10, 5}, {6, 8, 4, 7, 3, 9, 2, 1, 5, 10},
{6, 8, 4, 7, 3, 9, 2, 1, 10, 5}, {6, 8, 4, 7, 3, 9, 2, 10, 1, 5},
{6, 8, 4, 7, 9, 1, 3, 2, 5, 10}, {6, 8, 4, 7, 9, 1, 3, 2, 10, 5},
{6, 8, 4, 7, 9, 3, 1, 2, 5, 10}, {6, 8, 4, 7, 9, 3, 1, 2, 10, 5},
{6, 8, 4, 7, 9, 3, 2, 1, 5, 10}, {6, 8, 4, 7, 9, 3, 2, 1, 10, 5},
{6, 8, 4, 7, 9, 3, 2, 10, 1, 5}, {6, 8, 4, 9, 3, 2, 7, 1, 5, 10},
{6, 8, 4, 9, 3, 2, 7, 1, 10, 5}, {6, 8, 4, 9, 3, 2, 7, 10, 1, 5},
{6, 8, 4, 9, 3, 2, 10, 7, 1, 5}, {6, 8, 4, 9, 3, 7, 1, 2, 5, 10},
{6, 8, 4, 9, 3, 7, 1, 2, 10, 5}, {6, 8, 4, 9, 3, 7, 2, 1, 5, 10},
{6, 8, 4, 9, 3, 7, 2, 1, 10, 5}, {6, 8, 4, 9, 3, 7, 2, 10, 1, 5},
{6, 8, 4, 9, 7, 1, 3, 2, 5, 10}, {6, 8, 4, 9, 7, 1, 3, 2, 10, 5},
{6, 8, 4, 9, 7, 3, 1, 2, 5, 10}, {6, 8, 4, 9, 7, 3, 1, 2, 10, 5},
{6, 8, 4, 9, 7, 3, 2, 1, 5, 10}, {6, 8, 4, 9, 7, 3, 2, 1, 10, 5},
{6, 8, 4, 9, 7, 3, 2, 10, 1, 5}, {6, 8, 9, 4, 3, 2, 7, 1, 5, 10},
{6, 8, 9, 4, 3, 2, 7, 1, 10, 5}, {6, 8, 9, 4, 3, 2, 7, 10, 1, 5},
{6, 8, 9, 4, 3, 2, 10, 7, 1, 5}, {6, 8, 9, 4, 3, 7, 1, 2, 5, 10},
{6, 8, 9, 4, 3, 7, 1, 2, 10, 5}, {6, 8, 9, 4, 3, 7, 2, 1, 5, 10},
{6, 8, 9, 4, 3, 7, 2, 1, 10, 5}, {6, 8, 9, 4, 3, 7, 2, 10, 1, 5},
{6, 8, 9, 4, 7, 1, 3, 2, 5, 10}, {6, 8, 9, 4, 7, 1, 3, 2, 10, 5},
{6, 8, 9, 4, 7, 3, 1, 2, 5, 10}, {6, 8, 9, 4, 7, 3, 1, 2, 10, 5},
{6, 8, 9, 4, 7, 3, 2, 1, 5, 10}, {6, 8, 9, 4, 7, 3, 2, 1, 10, 5},
{6, 8, 9, 4, 7, 3, 2, 10, 1, 5}, {6, 8, 4, 3, 2, 7, 9, 1, 5, 10},
{6, 8, 4, 3, 2, 7, 9, 1, 10, 5}, {6, 8, 4, 3, 2, 7, 9, 10, 1, 5},
{6, 8, 4, 3, 2, 9, 7, 1, 5, 10}, {6, 8, 4, 3, 2, 9, 7, 1, 10, 5},
{6, 8, 4, 3, 2, 9, 7, 10, 1, 5}, {6, 8, 4, 3, 2, 9, 10, 7, 1, 5},
{6, 8, 4, 3, 7, 2, 9, 1, 5, 10}, {6, 8, 4, 3, 7, 2, 9, 1, 10, 5},
{6, 8, 4, 3, 7, 2, 9, 10, 1, 5}, {6, 8, 4, 3, 7, 9, 1, 2, 5, 10},
{6, 8, 4, 3, 7, 9, 1, 2, 10, 5}, {6, 8, 4, 3, 7, 9, 2, 1, 5, 10},
{6, 8, 4, 3, 7, 9, 2, 1, 10, 5}, {6, 8, 4, 3, 7, 9, 2, 10, 1, 5},
{6, 8, 4, 3, 9, 2, 7, 1, 5, 10}, {6, 8, 4, 3, 9, 2, 7, 1, 10, 5},
{6, 8, 4, 3, 9, 2, 7, 10, 1, 5}, {6, 8, 4, 3, 9, 2, 10, 7, 1, 5},
{6, 8, 4, 3, 9, 7, 1, 2, 5, 10}, {6, 8, 4, 3, 9, 7, 1, 2, 10, 5},
{6, 8, 4, 3, 9, 7, 2, 1, 5, 10}, {6, 8, 4, 3, 9, 7, 2, 1, 10, 5},
{6, 8, 4, 3, 9, 7, 2, 10, 1, 5}, {6, 8, 4, 7, 3, 2, 9, 1, 5, 10},
{6, 8, 4, 7, 3, 2, 9, 1, 10, 5}, {6, 8, 4, 7, 3, 2, 9, 10, 1, 5},
{6, 8, 4, 7, 3, 9, 1, 2, 5, 10}, {6, 8, 4, 7, 3, 9, 1, 2, 10, 5},
{6, 8, 4, 7, 3, 9, 2, 1, 5, 10}, {6, 8, 4, 7, 3, 9, 2, 1, 10, 5},
{6, 8, 4, 7, 3, 9, 2, 10, 1, 5}, {6, 8, 4, 7, 9, 1, 3, 2, 5, 10},
{6, 8, 4, 7, 9, 1, 3, 2, 10, 5}, {6, 8, 4, 7, 9, 3, 1, 2, 5, 10},
{6, 8, 4, 7, 9, 3, 1, 2, 10, 5}, {6, 8, 4, 7, 9, 3, 2, 1, 5, 10},
{6, 8, 4, 7, 9, 3, 2, 1, 10, 5}, {6, 8, 4, 7, 9, 3, 2, 10, 1, 5},
{6, 8, 4, 9, 3, 2, 7, 1, 5, 10}, {6, 8, 4, 9, 3, 2, 7, 1, 10, 5},
{6, 8, 4, 9, 3, 2, 7, 10, 1, 5}, {6, 8, 4, 9, 3, 2, 10, 7, 1, 5},
{6, 8, 4, 9, 3, 7, 1, 2, 5, 10}, {6, 8, 4, 9, 3, 7, 1, 2, 10, 5},
{6, 8, 4, 9, 3, 7, 2, 1, 5, 10}, {6, 8, 4, 9, 3, 7, 2, 1, 10, 5},
{6, 8, 4, 9, 3, 7, 2, 10, 1, 5}, {6, 8, 4, 9, 7, 1, 3, 2, 5, 10},
{6, 8, 4, 9, 7, 1, 3, 2, 10, 5}, {6, 8, 4, 9, 7, 3, 1, 2, 5, 10},
{6, 8, 4, 9, 7, 3, 1, 2, 10, 5}, {6, 8, 4, 9, 7, 3, 2, 1, 5, 10},
{6, 8, 4, 9, 7, 3, 2, 1, 10, 5}, {6, 8, 4, 9, 7, 3, 2, 10, 1, 5},
{6, 8, 9, 4, 3, 2, 7, 1, 5, 10}, {6, 8, 9, 4, 3, 2, 7, 1, 10, 5},

```
{6, 8, 9, 4, 3, 2, 7, 10, 1, 5}, {6, 8, 9, 4, 3, 2, 10, 7, 1, 5},
{6, 8, 9, 4, 3, 7, 1, 2, 5, 10}, {6, 8, 9, 4, 3, 7, 1, 2, 10, 5},
{6, 8, 9, 4, 3, 7, 2, 1, 5, 10}, {6, 8, 9, 4, 3, 7, 2, 1, 10, 5},
{6, 8, 9, 4, 3, 7, 2, 10, 1, 5}, {6, 8, 9, 4, 7, 1, 3, 2, 5, 10},
{6, 8, 9, 4, 7, 1, 3, 2, 10, 5}, {6, 8, 9, 4, 7, 3, 1, 2, 5, 10},
{6, 8, 9, 4, 7, 3, 1, 2, 10, 5}, {6, 8, 9, 4, 7, 3, 2, 1, 5, 10},
{6, 8, 9, 4, 7, 3, 2, 1, 10, 5}, {6, 8, 9, 4, 7, 3, 2, 10, 1, 5}}}}
```

In[ ]:=

```
matriz11 = {{30707, 0, 0, 0, 3, 0, 70, 0, 10, 0, 78}, {0, 184, 0, 02, 03, 70, 0, 0, 0, 2, 54},
   {0, 21, 240, 0, 0, 0, 0, 0, 0, 40, 85}, {21, 11, 13, 4458, 8, 16, 2, 0, 0, 12, 32},
   {0, 0, 0, 0, 5725, 50, 0, 0, 0, 0, 12}, {241, 15, 3, 0, 0, 189, 7, 1917, 1697, 0, 47},
   {1, 0, 0, 0, 70, 0, 75, 0, 0, 0, 54}, {1059, 19, 9, 146, 4313, 0, 15, 594, 211, 67, 15},
   {18, 0, 0, 70, 0, 0, 0, 23, 0, 1538, 45},
   {0, 10, 90, 70, 60, 0, 50, 10, 04, 1, 12}, {5, 5, 32, 3, 6, 35, 35, 3, 54, 8, 8}};
MatrixForm[matriz11]
```

forma de matriz

```
(*Timing[maximo[matriz11]]*)
```

cronometra

```
MatrixForm[ady11 = matrizady[matriz11]];
```

forma de matriz

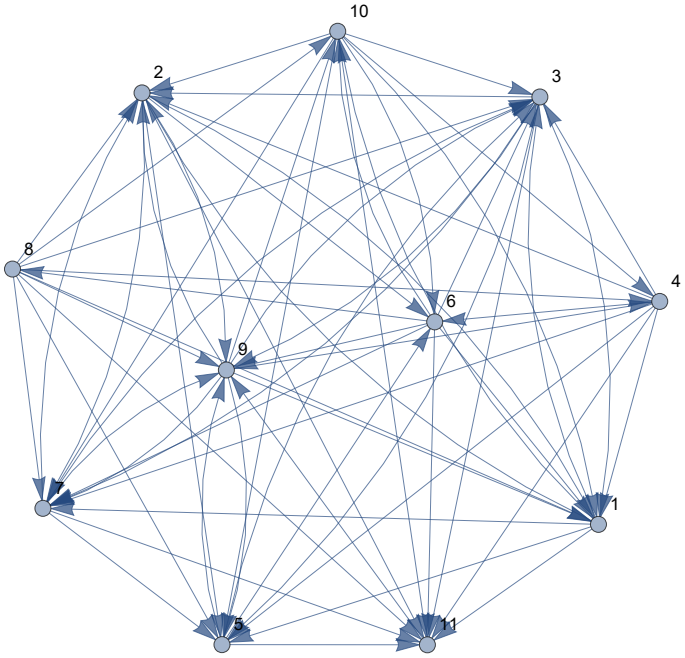
```
AdjacencyGraph[ady11, VertexLabels → Automatic]
```

grafo con adyacencia | etiquetas de vérti··· | automático

Out[ ]//MatrixForm=

$$\begin{pmatrix} 30707 & 0 & 0 & 0 & 3 & 0 & 70 & 0 & 10 & 0 & 78 \\ 0 & 184 & 0 & 2 & 3 & 70 & 0 & 0 & 0 & 2 & 54 \\ 0 & 21 & 240 & 0 & 0 & 0 & 0 & 0 & 0 & 40 & 85 \\ 21 & 11 & 13 & 4458 & 8 & 16 & 2 & 0 & 0 & 12 & 32 \\ 0 & 0 & 0 & 0 & 5725 & 50 & 0 & 0 & 0 & 0 & 12 \\ 241 & 15 & 3 & 0 & 0 & 189 & 7 & 1917 & 1697 & 0 & 47 \\ 1 & 0 & 0 & 0 & 70 & 0 & 75 & 0 & 0 & 0 & 54 \\ 1059 & 19 & 9 & 146 & 4313 & 0 & 15 & 594 & 211 & 67 & 15 \\ 18 & 0 & 0 & 70 & 0 & 0 & 0 & 23 & 0 & 1538 & 45 \\ 0 & 10 & 90 & 70 & 60 & 0 & 50 & 10 & 4 & 1 & 12 \\ 5 & 5 & 32 & 3 & 6 & 35 & 35 & 3 & 54 & 8 & 8 \end{pmatrix}$$

Out[ ]=



In[ ]:=
```
caminos11 = ReadList["C:/Users/Usuario/Desktop/tfg/graph11.txt"];
```
lee lista constante

```
Timing[maxcamino[caminos11, matriz11]]
```
cronometra

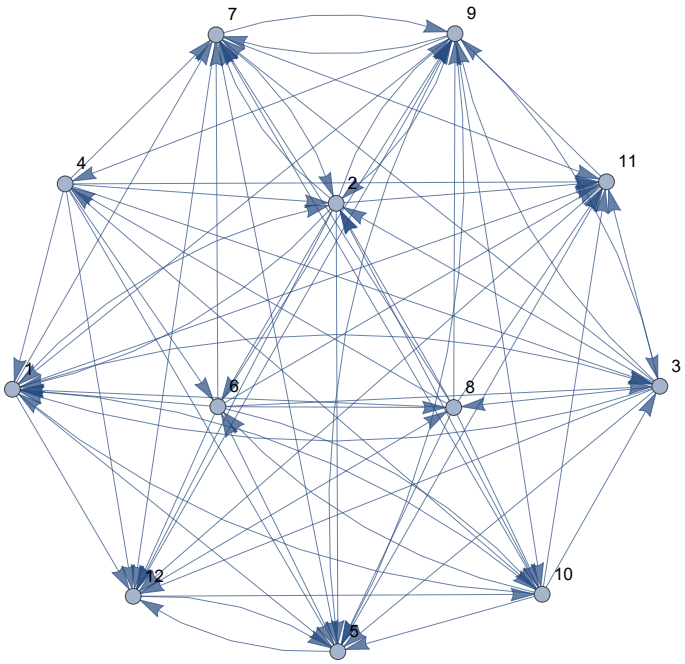
Out[ ]=
```
{28.4219, {12281, {{1, 2, 5, 3, 7, 9, 4, 6, 8, 10, 11}, {1, 2, 5, 3, 7, 9, 10, 4, 6, 8, 11},
    {1, 2, 5, 3, 7, 9, 10, 6, 8, 4, 11}, {1, 2, 5, 3, 9, 10, 6, 8, 4, 7, 11},
    {1, 2, 5, 6, 8, 3, 7, 9, 10, 4, 11}, {1, 2, 5, 6, 8, 3, 9, 10, 4, 7, 11},
    {1, 2, 5, 6, 8, 7, 9, 10, 4, 3, 11}, {1, 2, 5, 6, 8, 9, 10, 4, 3, 7, 11},
    {1, 2, 5, 6, 8, 9, 10, 4, 7, 3, 11}, {1, 2, 6, 8, 3, 5, 9, 10, 4, 7, 11},
    {1, 2, 6, 8, 3, 7, 5, 9, 10, 4, 11}, {1, 2, 6, 8, 3, 7, 9, 10, 4, 5, 11},
    {1, 2, 6, 8, 3, 9, 10, 4, 7, 5, 11}, {1, 2, 6, 8, 7, 9, 10, 4, 3, 5, 11},
    {1, 2, 6, 8, 7, 9, 10, 4, 5, 3, 11}, {1, 2, 6, 8, 9, 10, 4, 3, 7, 5, 11},
    {1, 2, 6, 8, 9, 10, 4, 7, 3, 5, 11}, {1, 2, 6, 8, 9, 10, 4, 7, 5, 3, 11},
    {2, 6, 8, 1, 3, 5, 9, 10, 4, 7, 11}, {2, 6, 8, 1, 3, 7, 5, 9, 10, 4, 11},
    {2, 6, 8, 1, 3, 7, 9, 10, 4, 5, 11}, {2, 6, 8, 1, 3, 9, 10, 4, 7, 5, 11},
    {2, 6, 8, 1, 7, 9, 10, 4, 3, 5, 11}, {2, 6, 8, 1, 7, 9, 10, 4, 5, 3, 11},
    {2, 6, 8, 3, 9, 10, 4, 1, 7, 5, 11}, {2, 6, 8, 9, 1, 10, 4, 3, 7, 5, 11},
    {2, 6, 8, 9, 1, 10, 4, 7, 3, 5, 11}, {2, 6, 8, 9, 1, 10, 4, 7, 5, 3, 11},
    {2, 6, 8, 9, 10, 4, 1, 3, 7, 5, 11}, {2, 6, 8, 9, 10, 4, 1, 7, 3, 5, 11},
    {2, 6, 8, 9, 10, 4, 1, 7, 5, 3, 11}, {2, 6, 8, 9, 10, 4, 3, 1, 7, 5, 11},
    {6, 8, 9, 10, 4, 1, 3, 2, 7, 5, 11}, {6, 8, 9, 10, 4, 1, 3, 7, 2, 5, 11},
    {6, 8, 9, 10, 4, 1, 7, 3, 2, 5, 11}, {6, 8, 9, 10, 4, 3, 1, 2, 7, 5, 11},
    {6, 8, 9, 10, 4, 3, 1, 7, 2, 5, 11}, {6, 8, 9, 10, 4, 3, 2, 1, 7, 5, 11},
    {6, 8, 9, 10, 4, 1, 3, 2, 7, 5, 11}, {6, 8, 9, 10, 4, 1, 3, 7, 2, 5, 11},
    {6, 8, 9, 10, 4, 1, 7, 3, 2, 5, 11}, {6, 8, 9, 10, 4, 3, 1, 2, 7, 5, 11},
    {6, 8, 9, 10, 4, 3, 1, 7, 2, 5, 11}, {6, 8, 9, 10, 4, 3, 2, 1, 7, 5, 11}}}}
```

In[ ]:=
```
matriz12 = {{30707, 0, 0, 0, 3, 0, 70, 0, 10, 0, 78, 54},
    {0, 184, 0, 02, 03, 70, 0, 0, 0, 2, 54, 87}, {0, 21, 240, 0, 45, 0, 40, 10, 0, 40, 85, 78},
    {21, 11, 13, 4458, 8, 16, 2, 0, 0, 12, 32, 768},
    {0, 0, 0, 0, 5725, 50, 0, 0, 0, 0, 12, 56},
    {241, 15, 3, 0, 0, 189, 7, 1917, 1697, 0, 47, 86}, {1, 0, 0, 0, 70, 0, 75, 0, 0, 0, 54, 53},
    {1059, 19, 9, 146, 4313, 0, 15, 594, 211, 67, 15, 89},
    {18, 0, 0, 70, 0, 0, 0, 23, 0, 1538, 45, 5}, {0, 10, 90, 70, 60, 0, 50, 10, 04, 1, 12, 68},
    {5, 5, 32, 3, 6, 35, 35, 3, 54, 8, 8, 546}, {5, 5, 3, 56, 56, 6, 8, 3268, 43, 87, 9, 65}};
MatrixForm[matriz12]
  forma de matriz
(*Timing[maximo[matriz12]]*)
  cronometra
MatrixForm[ady12 = matrizady[matriz12]];
  forma de matriz
AdjacencyGraph[ady12, VertexLabels → Automatic]
  grafo con adyacencia   etiquetas de vérti···   automático
```

Out[ ]//MatrixForm=

30707	0	0	0	3	0	70	0	10	0	78	54
0	184	0	2	3	70	0	0	0	2	54	87
0	21	240	0	45	0	40	10	0	40	85	78
21	11	13	4458	8	16	2	0	0	12	32	768
0	0	0	0	5725	50	0	0	0	0	12	56
241	15	3	0	0	189	7	1917	1697	0	47	86
1	0	0	0	70	0	75	0	0	0	54	53
1059	19	9	146	4313	0	15	594	211	67	15	89
18	0	0	70	0	0	0	23	0	1538	45	5
0	10	90	70	60	0	50	10	4	1	12	68
5	5	32	3	6	35	35	3	54	8	8	546
5	5	3	56	56	6	8	3268	43	87	9	65

Out[ ]=



In[ ]:=
```
caminos12 = ReadList["C:/Users/Usuario/Desktop/tfg/graph12.txt"];
            lee lista      constante
Timing[maxcamino[caminos12, matriz12]]
 cronometra
```

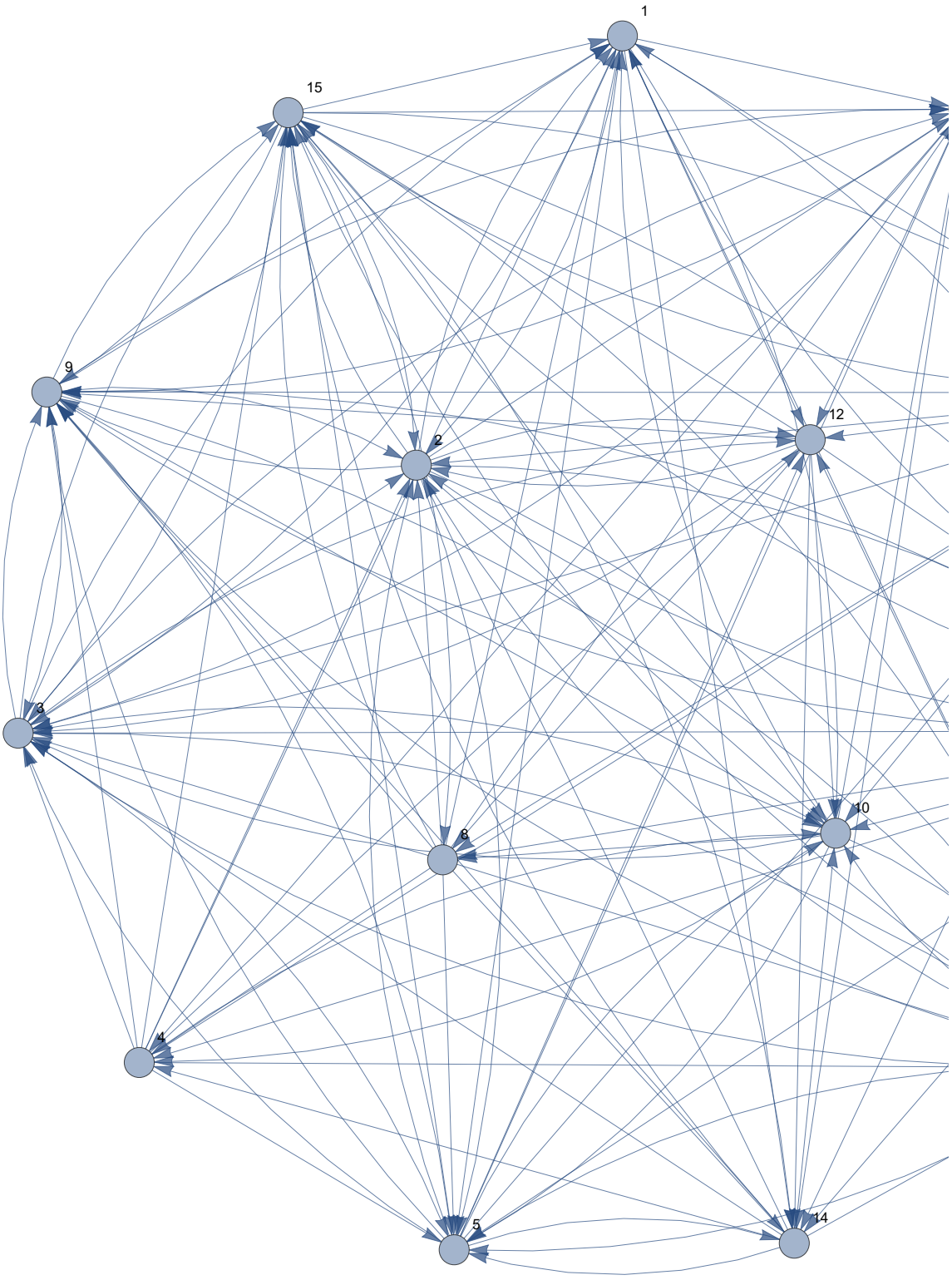
Out[ ]=
```
{232.813,
 {17011, {{1, 2, 5, 6, 3, 7, 9, 4, 11, 12, 8, 10}, {1, 2, 5, 6, 3, 9, 4, 7, 11, 12, 8, 10},
   {1, 2, 5, 6, 3, 9, 4, 11, 12, 8, 10, 7}, {1, 2, 5, 6, 3, 9, 10, 4, 7, 11, 12, 8},
   {1, 2, 5, 6, 4, 3, 7, 11, 12, 8, 9, 10}, {1, 2, 5, 6, 4, 3, 11, 12, 8, 9, 10, 7},
   {1, 2, 6, 3, 7, 9, 4, 11, 12, 8, 10, 5}, {1, 2, 6, 3, 9, 4, 7, 11, 12, 8, 10, 5},
   {1, 2, 6, 3, 9, 4, 11, 12, 8, 10, 7, 5}, {1, 2, 6, 3, 9, 10, 4, 7, 11, 12, 8, 5},
   {1, 2, 6, 4, 3, 7, 11, 12, 8, 5, 9, 10}, {1, 2, 6, 4, 3, 7, 11, 12, 8, 9, 10, 5},
   {1, 2, 6, 4, 3, 11, 12, 8, 9, 10, 7, 5}, {1, 3, 2, 6, 4, 7, 11, 12, 8, 9, 10, 5},
   {1, 3, 2, 6, 4, 11, 12, 8, 9, 10, 7, 5}, {2, 6, 1, 3, 7, 9, 4, 11, 12, 8, 10, 5},
   {2, 6, 1, 3, 9, 4, 7, 11, 12, 8, 10, 5}, {2, 6, 1, 3, 9, 4, 11, 12, 8, 10, 7, 5},
   {2, 6, 1, 3, 9, 10, 4, 7, 11, 12, 8, 5}, {2, 6, 3, 7, 9, 4, 11, 12, 8, 1, 10, 5},
   {2, 6, 3, 7, 9, 4, 11, 12, 8, 10, 1, 5}, {2, 6, 3, 9, 4, 7, 11, 12, 8, 1, 10, 5},
   {2, 6, 3, 9, 4, 7, 11, 12, 8, 10, 1, 5}, {2, 6, 3, 9, 4, 11, 12, 8, 1, 10, 7, 5},
   {2, 6, 3, 9, 4, 11, 12, 8, 10, 1, 7, 5}, {2, 6, 4, 3, 7, 11, 12, 8, 1, 5, 9, 10},
   {2, 6, 4, 3, 7, 11, 12, 8, 5, 9, 1, 10}, {2, 6, 4, 3, 7, 11, 12, 8, 5, 9, 10, 1},
   {2, 6, 4, 3, 7, 11, 12, 8, 9, 1, 10, 5}, {2, 6, 4, 3, 7, 11, 12, 8, 9, 10, 1, 5},
   {2, 6, 4, 3, 11, 12, 8, 1, 7, 9, 10, 5}, {2, 6, 4, 3, 11, 12, 8, 9, 1, 10, 7, 5},
   {2, 6, 4, 3, 11, 12, 8, 9, 10, 1, 7, 5}, {3, 2, 6, 4, 11, 12, 8, 9, 1, 10, 7, 5},
   {3, 2, 6, 4, 11, 12, 8, 9, 10, 1, 7, 5}, {4, 2, 6, 3, 11, 12, 8, 9, 1, 10, 7, 5},
   {4, 2, 6, 3, 11, 12, 8, 9, 10, 1, 7, 5}, {4, 3, 2, 6, 11, 12, 8, 9, 1, 10, 7, 5},
   {4, 3, 2, 6, 11, 12, 8, 9, 10, 1, 7, 5}}}}
```

Experimentación datos economía

In[ ]:=
```
datos15 = {{30707, 0, 0, 0, 3, 0, 0, 0, 0, 0, 75579, 844, 0, 32, 7},
{0, 184, 0, 0, 0, 0, 0, 0, 0, 2, 1, 0, 0, 0, 0},
{0, 21, 240, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0},
{21, 11, 13, 445, 8, 8, 1, 6, 2, 0, 0, 1, 2, 16, 3},
{0, 0, 0, 0, 5725, 50, 0, 0, 0, 0, 6, 7, 8, 0, 1},
{241, 15, 3, 0, 0, 1, 89, 7, 1917, 1697, 0, 18, 0, 2, 0},
{1, 0, 0, 0, 0, 0, 7, 5, 0, 0, 0, 26, 0, 0, 0},
{1059, 19, 9, 146, 4313, 0, 15, 594, 2116, 7, 78, 11, 0, 48, 0},
{18, 0, 0, 0, 0, 0, 0, 23, 0, 1538, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 10, 0, 1, 53820, 0, 0, 0, 0},
{0, 0, 1, 5, 0, 1, 4, 0, 0, 4, 0, 10645, 2036, 15719, 14},
{95, 0, 0, 0, 0, 0, 0, 1892, 436, 3, 9, 0, 769, 1083, 46},
{0, 0, 1, 5, 11, 2, 1, 125, 0, 17, 70, 0, 0, 10690, 0},
{0, 4, 22, 101, 0, 0, 2, 2, 0859, 2722, 3, 1, 3, 6, 0},
{174, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20, 56, 0, 0, 0, 0}};

MatrixForm[datos15];
  forma de matriz
(*Timing[maximo[datos]]*)
    cronometra
MatrixForm[ady15 = matrizady[datos15]];
  forma de matriz
AdjacencyGraph[ady15, VertexLabels → Automatic]
  grafo con adyacencia    etiquetas de vérti···    automático
diccionario[ady15]
```

Out[•]=



Out[•]= diccionario[{{0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0},
{1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1}, {1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1},
{1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1}, {0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1},
{1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1}, {1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1},
{1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1}, {1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1},
{1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0}, {0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0},
{0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1}, {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1},
{0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1}, {1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0}}]

••• Set: Tag Inherited in Inherited[State] is Protected.

```
caminos15 = ReadList["C:/Users/Usuario/Desktop/tfg/graph15.txt"];
            lee lista      constante
```

```
maxcamino[caminos15, datos15]
 {185216, {{1, 2, 5, 3, 7, 9, 10, 4, 6, 8, 11, 12, 13, 14, 15}, {1, 2, 5, 3, 7, 9, 10, 4,
    6, 8, 11, 12, 13, 15, 14}, {1, 2, 5, 3, 7, 9, 10, 4, 6, 8, 11, 12, 15, 13, 14},
   {1, 2, 5, 3, 7, 9, 10, 4, 6, 15, 11, 12, 13, 8, 14},
   {1, 2, 5, 3, 7, 9, 10, 4, 15, 11, 6, 12, 13, 8, 14}, {1, 2, 5, 3, 7, 9, 10, 11,
    4, 6, 12, 13, 8, 14, 15}, {1, 2, 5, 3, 7, 9, 10, 11, 4, 6, 12, 13, 8, 15, 14},
   {1, 2, 5, 3, 7, 9, 10, 11, 4, 6, 12, 13, 15, 8, 14}, {1, 2, 5, 3, 7, 9, 10, 11,
    4, 6, 12, 15, 13, 8, 14}, {1, 2, 5, 3, 7, 9, 10, 11, 6, 12, 8, 4, 15, 13, 14},
   {1, 2, 5, 3, 7, 9, 10, 11, 6, 12, 8, 15, 13, 14, 4}, {1, 2, 5, 3, 7, 9, 10, 11,
    6, 12, 13, 8, 4, 15, 14}, {1, 2, 5, 3, 7, 9, 10, 11, 6, 12, 13, 8, 14, 4, 15},
   {1, 2, 5, 3, 7, 9, 15, 6, 10, 11, 4, 12, 13, 8, 14}, {1, 2, 5, 3, 7, 9, 15, 6,
    10, 11, 12, 13, 8, 14, 4}, {1, 2, 5, 3, 7, 10, 4, 6, 15, 11, 12, 8, 9, 13, 14},
   {1, 2, 5, 3, 7, 10, 4, 6, 15, 11, 12, 13, 8, 14, 9}, {1, 2, 5, 3, 7, 10, 4, 15,
    11, 6, 12, 13, 8, 14, 9}, {1, 2, 5, 3, 7, 10, 11, 4, 6, 12, 13, 8, 14, 9, 15},
   {1, 2, 5, 3, 7, 10, 11, 4, 6, 12, 13, 8, 14, 15, 9}, {1, 2, 5, 3, 7, 10, 11, 4,
    6, 12, 13, 8, 15, 14, 9}, {1, 2, 5, 3, 7, 10, 11, 4, 6, 12, 13, 15, 8, 14, 9},
   {1, 2, 5, 3, 7, 10, 11, 4, 6, 12, 15, 13, 8, 14, 9}, {1, 2, 5, 3, 7, 10, 11, 6,
    12, 8, 4, 15, 13, 14, 9}, {1, 2, 5, 3, 7, 10, 11, 6, 12, 8, 15, 13, 14, 4, 9},
   {1, 2, 5, 3, 7, 10, 11, 6, 12, 13, 8, 4, 15, 14, 9}, {1, 2, 5, 3, 7, 10, 11, 6,
    12, 13, 8, 14, 4, 9, 15}, {1, 2, 5, 3, 7, 10, 11, 6, 12, 13, 8, 14, 4, 15, 9},
   {1, 2, 5, 3, 7, 15, 6, 8, 9, 10, 11, 12, 13, 14, 4}, {1, 2, 5, 3, 7, 15, 6, 10,
    11, 4, 12, 8, 9, 13, 14}, {1, 2, 5, 3, 7, 15, 6, 10, 11, 4, 12, 13, 8, 14, 9},
   {1, 2, 5, 3, 7, 15, 6, 10, 11, 12, 13, 8, 14, 4, 9}, {1, 2, 5, 3, 15, 6, 7, 10,
    11, 12, 13, 8, 14, 4, 9}, {1, 2, 5, 3, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 9},
   {1, 2, 5, 6, 3, 15, 10, 11, 7, 12, 13, 8, 14, 4, 9}, {1, 2, 5, 6, 7, 10, 11, 12,
    13, 8, 14, 4, 3, 9, 15}, {1, 2, 5, 6, 7, 10, 11, 12, 13, 8, 14, 4, 3, 15, 9},
   {1, 2, 5, 6, 7, 10, 11, 12, 13, 8, 14, 4, 9, 3, 15}, {1, 2, 5, 6, 7, 10, 11, 12,
    13, 8, 14, 4, 9, 15, 3}, {1, 2, 5, 6, 7, 10, 11, 12, 13, 8, 14, 4, 15, 3, 9},
   {1, 2, 5, 6, 7, 10, 11, 12, 13, 8, 14, 4, 15, 9, 3}, {1, 2, 5, 6, 7, 15, 10, 11,
    12, 13, 8, 14, 4, 3, 9}, {1, 2, 5, 6, 7, 15, 10, 11, 12, 13, 8, 14, 4, 9, 3},
   {1, 2, 5, 6, 15, 7, 10, 11, 12, 13, 8, 14, 4, 3, 9}, {1, 2, 5, 6, 15, 7, 10, 11,
    12, 13, 8, 14, 4, 9, 3}, {1, 2, 5, 6, 15, 10, 11, 7, 12, 13, 8, 14, 4, 3, 9},
   {1, 2, 5, 6, 15, 10, 11, 7, 12, 13, 8, 14, 4, 9, 3}, {1, 2, 5, 15, 6, 10, 11, 7,
    12, 13, 8, 14, 4, 3, 9}, {1, 2, 5, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 9, 3},
   {1, 2, 7, 3, 9, 15, 6, 10, 11, 4, 12, 13, 8, 5, 14}, {1, 2, 7, 3, 9, 15, 6, 10,
    11, 4, 12, 13, 8, 14, 5}, {1, 2, 7, 3, 9, 15, 6, 10, 11, 12, 13, 8, 4, 5, 14},
   {1, 2, 7, 3, 9, 15, 6, 10, 11, 12, 13, 8, 5, 14, 4}, {1, 2, 7, 3, 9, 15, 6, 10,
    11, 12, 13, 8, 14, 4, 5}, {1, 2, 7, 3, 10, 4, 6, 15, 11, 12, 8, 5, 9, 13, 14},
   {1, 2, 7, 3, 10, 4, 6, 15, 11, 12, 8, 9, 13, 5, 14}, {1, 2, 7, 3, 10, 4, 6, 15,
    11, 12, 8, 9, 13, 14, 5}, {1, 2, 7, 3, 10, 4, 6, 15, 11, 12, 13, 8, 5, 14, 9},
   {1, 2, 7, 3, 10, 4, 6, 15, 11, 12, 13, 8, 14, 5, 9}, {1, 2, 7, 3, 10, 4, 6, 15,
```

```
 11, 12, 13, 8, 14, 9, 5}, {1, 2, 7, 3, 10, 4, 15, 11, 6, 12, 13, 8, 5, 14, 9},
{1, 2, 7, 3, 10, 4, 15, 11, 6, 12, 13, 8, 14, 5, 9}, {1, 2, 7, 3, 10, 4, 15, 11,
 6, 12, 13, 8, 14, 9, 5}, {1, 2, 7, 3, 10, 4, 15, 11, 12, 13, 8, 5, 6, 14, 9},
{1, 2, 7, 3, 10, 11, 4, 12, 13, 8, 5, 6, 14, 9, 15}, {1, 2, 7, 3, 10, 11, 4, 12,
 13, 8, 5, 6, 14, 15, 9}, {1, 2, 7, 3, 10, 11, 4, 12, 13, 8, 5, 6, 15, 14, 9},
{1, 2, 7, 3, 10, 11, 4, 12, 13, 8, 5, 15, 6, 14, 9}, {1, 2, 7, 3, 10, 11, 6, 12,
 8, 5, 15, 13, 14, 4, 9}, {1, 2, 7, 3, 10, 11, 6, 12, 8, 15, 13, 5, 14, 4, 9},
{1, 2, 7, 3, 10, 11, 6, 12, 8, 15, 13, 14, 4, 5, 9}, {1, 2, 7, 3, 10, 11, 6, 12,
 8, 15, 13, 14, 4, 9, 5}, {1, 2, 7, 3, 10, 11, 6, 12, 13, 8, 4, 5, 14, 9, 15},
{1, 2, 7, 3, 10, 11, 6, 12, 13, 8, 4, 5, 14, 15, 9}, {1, 2, 7, 3, 10, 11, 6, 12,
 13, 8, 4, 5, 15, 14, 9}, {1, 2, 7, 3, 10, 11, 6, 12, 13, 8, 5, 14, 4, 9, 15},
{1, 2, 7, 3, 10, 11, 6, 12, 13, 8, 5, 14, 4, 15, 9}, {1, 2, 7, 3, 10, 11, 6, 12,
 13, 8, 14, 4, 5, 9, 15}, {1, 2, 7, 3, 10, 11, 6, 12, 13, 8, 14, 4, 5, 15, 9},
{1, 2, 7, 3, 10, 11, 6, 12, 13, 8, 14, 4, 9, 5, 15}, {1, 2, 7, 3, 10, 11, 12, 13,
 8, 5, 6, 14, 4, 9, 15}, {1, 2, 7, 3, 10, 11, 12, 13, 8, 5, 6, 14, 4, 15, 9},
{1, 2, 7, 3, 10, 11, 12, 13, 8, 5, 14, 4, 6, 9, 15}, {1, 2, 7, 3, 10, 11, 12, 13,
 8, 5, 14, 4, 6, 15, 9}, {1, 2, 7, 3, 10, 11, 12, 13, 8, 5, 14, 4, 15, 6, 9},
{1, 2, 7, 3, 10, 11, 12, 13, 8, 14, 4, 5, 6, 9, 15}, {1, 2, 7, 3, 10, 11, 12, 13,
 8, 14, 4, 5, 6, 15, 9}, {1, 2, 7, 3, 10, 11, 12, 13, 8, 14, 4, 5, 15, 6, 9},
{1, 2, 7, 3, 15, 6, 10, 11, 4, 12, 8, 5, 9, 13, 14}, {1, 2, 7, 3, 15, 6, 10, 11,
 4, 12, 8, 9, 13, 5, 14}, {1, 2, 7, 3, 15, 6, 10, 11, 4, 12, 8, 9, 13, 14, 5},
{1, 2, 7, 3, 15, 6, 10, 11, 4, 12, 13, 8, 5, 14, 9}, {1, 2, 7, 3, 15, 6, 10, 11,
 4, 12, 13, 8, 14, 5, 9}, {1, 2, 7, 3, 15, 6, 10, 11, 4, 12, 13, 8, 14, 9, 5},
{1, 2, 7, 3, 15, 6, 10, 11, 12, 13, 8, 4, 5, 14, 9}, {1, 2, 7, 3, 15, 6, 10, 11,
 12, 13, 8, 5, 14, 4, 9}, {1, 2, 7, 3, 15, 6, 10, 11, 12, 13, 8, 14, 4, 5, 9},
{1, 2, 7, 3, 15, 6, 10, 11, 12, 13, 8, 14, 4, 9, 5}, {1, 2, 7, 15, 6, 3, 10, 11,
 12, 13, 8, 14, 4, 5, 9}, {1, 2, 7, 15, 6, 3, 10, 11, 12, 13, 8, 14, 4, 9, 5},
{1, 2, 7, 15, 6, 10, 3, 11, 12, 13, 8, 14, 4, 5, 9}, {1, 2, 7, 15, 6, 10, 3, 11,
 12, 13, 8, 14, 4, 9, 5}, {1, 2, 7, 15, 6, 10, 11, 3, 12, 13, 8, 14, 4, 5, 9},
{1, 2, 7, 15, 6, 10, 11, 3, 12, 13, 8, 14, 4, 9, 5}, {1, 2, 7, 15, 6, 10, 11, 12,
 13, 8, 3, 5, 14, 4, 9}, {1, 2, 7, 15, 6, 10, 11, 12, 13, 8, 5, 14, 4, 3, 9},
{1, 2, 7, 15, 6, 10, 11, 12, 13, 8, 5, 14, 4, 9, 3}, {1, 2, 7, 15, 6, 10, 11, 12,
 13, 8, 14, 4, 3, 5, 9}, {1, 2, 7, 15, 6, 10, 11, 12, 13, 8, 14, 4, 3, 9, 5},
{1, 2, 7, 15, 6, 10, 11, 12, 13, 8, 14, 4, 5, 3, 9}, {1, 2, 7, 15, 6, 10, 11, 12,
 13, 8, 14, 4, 5, 9, 3}, {1, 2, 7, 15, 6, 10, 11, 12, 13, 8, 14, 4, 9, 3, 5},
{1, 2, 7, 15, 6, 10, 11, 12, 13, 8, 14, 4, 9, 5, 3}, {1, 2, 15, 6, 3, 7, 10, 11,
 12, 13, 8, 5, 14, 4, 9}, {1, 2, 15, 6, 3, 7, 10, 11, 12, 13, 8, 14, 4, 5, 9},
{1, 2, 15, 6, 3, 7, 10, 11, 12, 13, 8, 14, 4, 9, 5}, {1, 2, 15, 6, 3, 10, 11, 7,
 12, 13, 8, 14, 4, 5, 9}, {1, 2, 15, 6, 3, 10, 11, 7, 12, 13, 8, 14, 4, 9, 5},
{1, 2, 15, 6, 7, 10, 11, 12, 13, 8, 5, 14, 4, 3, 9}, {1, 2, 15, 6, 7, 10, 11, 12,
 13, 8, 5, 14, 4, 9, 3}, {1, 2, 15, 6, 7, 10, 11, 12, 13, 8, 14, 4, 3, 5, 9},
{1, 2, 15, 6, 7, 10, 11, 12, 13, 8, 14, 4, 3, 9, 5}, {1, 2, 15, 6, 7, 10, 11, 12,
 13, 8, 14, 4, 5, 3, 9}, {1, 2, 15, 6, 7, 10, 11, 12, 13, 8, 14, 4, 5, 9, 3},
{1, 2, 15, 6, 7, 10, 11, 12, 13, 8, 14, 4, 9, 3, 5}, {1, 2, 15, 6, 7, 10, 11, 12,
 13, 8, 14, 4, 9, 5, 3}, {1, 2, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 3, 5, 9},
{1, 2, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 3, 9, 5}, {1, 2, 15, 6, 10, 11, 7, 12,
 13, 8, 14, 4, 5, 3, 9}, {1, 2, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 5, 9, 3},
{1, 2, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 9, 3, 5}, {1, 2, 15, 6, 10, 11, 7, 12,
 13, 8, 14, 4, 9, 5, 3}, {1, 3, 15, 6, 7, 10, 11, 12, 13, 8, 5, 14, 4, 2, 9},
{1, 3, 15, 6, 7, 10, 11, 12, 13, 8, 5, 14, 4, 9, 2}, {1, 3, 15, 6, 7, 10, 11, 12,
 13, 8, 14, 4, 2, 5, 9}, {1, 3, 15, 6, 7, 10, 11, 12, 13, 8, 14, 4, 2, 9, 5},
{1, 3, 15, 6, 7, 10, 11, 12, 13, 8, 14, 4, 5, 2, 9}, {1, 3, 15, 6, 7, 10, 11, 12,
 13, 8, 14, 4, 5, 9, 2}, {1, 3, 15, 6, 7, 10, 11, 12, 13, 8, 14, 4, 9, 2, 5},
{1, 3, 15, 6, 7, 10, 11, 12, 13, 8, 14, 4, 9, 5, 2}, {1, 3, 15, 6, 10, 11, 7, 12,
 13, 8, 14, 4, 2, 5, 9}, {1, 3, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 2, 9, 5},
```

```
{1, 3, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 5, 2, 9}, {1, 3, 15, 6, 10, 11, 7, 12,
  13, 8, 14, 4, 5, 9, 2}, {1, 3, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 9, 2, 5},
{1, 3, 15, 6, 10, 11, 7, 12, 13, 8, 14, 4, 9, 5, 2}}}
```

```
#FUNCI□ N BUSCA CAMINOS HAMILTONIANOS

def PrintAllHamiltonianPaths(graph, visited , archivo ):
    f = open(archivo,"a")
    if(len(visited) == len(graph)):
        f.write("{" + str(visited)[1:-1]+"}"+"\n")
        return()


    for vertex in graph[visited[-1]]:
        if not (vertex in visited):
            visited.append(vertex)
            PrintAllHamiltonianPaths(graph, visited, archivo)
            visited.pop()
    f.close()




from time import time


##########################################################################################################
##########################################################################################################
########################

tiempo_inicialg = time()

graph = {1: [], 2: [1, 4], 3: [1, 2],4: [1, 3]}

print ("Caminos hamiltonianos para el grafo del ejemplo 4.2.1 del capítulo 4")

for i in range (1,(len(graph)+1)):
    PrintAllHamiltonianPaths(graph, visited = [i], archivo = "graph.txt" )

tiempo_finalg = time()
tiempo_ejecuciong = tiempo_finalg - tiempo_inicialg


print ('El tiempo de ejecucion fue:',tiempo_ejecuciong )

f = open("graph.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 4 v□ rtices")
f.close()

##########################################################################################################
##########################################################################################################
########################

tiempo_inicial5 = time()

graph5 = {1: [2, 3, 5], 2: [1, 3, 4], 3: [4, 5],4: [1, 5], 5: [2, 3, 4]}

print ("Caminos hamiltonianos para el grafo 5")
```

```
for i in range (1,(len(graph5)+1)):
   PrintAllHamiltonianPaths(graph5,[i], "graph5.txt" )

tiempo_final5 = time()
tiempo_ejecucion5 = tiempo_final5 - tiempo_inicial5

print ('El tiempo de ejecucion para el grafo 5 fue:',tiempo_ejecucion5)


f = open("graph5.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 5 v□ rtices")
f.close()

##########################################################################################################
##########################################################################################################
########################

tiempo_inicial6 = time()

graph6 = {1: [2, 3, 4, 5, 6], 2: [ 3, 4, 6], 3: [4, 5],4: [6], 5: [2, 4, 6], 6: [3]}

print ("Caminos hamiltonianos para el grafo 6")

for i in range (1,(len(graph6)+1)):
  PrintAllHamiltonianPaths(graph6,[i], "graph6.txt")

tiempo_final6 = time()
tiempo_ejecucion6 = tiempo_final6 - tiempo_inicial6

print ('El tiempo de ejecucion para el grafo 6 fue:',tiempo_ejecucion6)


f = open("graph6.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 6 v□ rtices")
f.close()

##########################################################################################################
##########################################################################################################
########################


tiempo_inicial7 = time()

graph7 = {1: [ 3, 4, 5, 6, 7], 2: [ 1, 2, 3, 4, 5, 6, 7], 3: [5, 6],4: [3, 7], 5: [4, 6], 6: [4, 7], 7:[3, 5]}

print ("Caminos hamiltonianos para el grafo 7")

for i in range (1,(len(graph7)+1)):
    PrintAllHamiltonianPaths(graph7,[i], "graph7.txt")

tiempo_final7 = time()
tiempo_ejecucion7 = tiempo_final7 - tiempo_inicial7
```

```
print ('El tiempo de ejecucion para el grafo 7 fue:',tiempo_ejecucion7)


f = open("graph7.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 7 v rtices")
f.close()
##########################################################################################################
##########################################################################################################
########################



tiempo_inicial8 = time()

graph8 = {1: [ 2, 3, 4, 5, 6, 7], 2: [3, 4, 5, 7], 3: [5, 6], 4: [3, 5, 7], 5: [7, 8], 6: [1, 2, 4, 5, 8], 7:[1, 2, 3, 4, 6, 8], 8:[1,
2, 3]}

print ("Caminos hamiltonianos para el grafo 8")

for i in range (1,(len(graph8)+1)):
    PrintAllHamiltonianPaths(graph8,[i], "graph8.txt")

tiempo_final8 = time()
tiempo_ejecucion8 = tiempo_final8 - tiempo_inicial8

print ('El tiempo de ejecucion para el grafo 8 fue:',tiempo_ejecucion8)



f = open("graph8.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 8 v rtices")
f.close()

##########################################################################################################
##########################################################################################################
########################

tiempo_inicial9 = time()

graph9 = {1: [3, 4, 5, 6, 7, 8, 9], 2: [1, 4, 5, 6, 7], 3: [2, 4, 6, 7], 4: [5, 7], 5: [3, 6, 7], 6: [4, 8, 9], 7:[6], 8:[2, 3, 4 ,5, 7
], 9:[2, 3, 4, 5, 7, 8]}

print ("Caminos hamiltonianos para el grafo 9")

for i in range (1,(len(graph9)+1)):
    PrintAllHamiltonianPaths(graph9,[i],"graph9.txt")

tiempo_final9 = time()
tiempo_ejecucion9 = tiempo_final9 - tiempo_inicial9

print ('El tiempo de ejecucion para el grafo 9 fue:',tiempo_ejecucion9)


f = open("graph9.txt", "r")
```

```
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 9 v□ rtices")
f.close()

##########################################################################################################
##########################################################################################################
#######################


tiempo_inicial10 = time()

graph10 = {1:[2, 3, 5, 10], 2:[1, 5, 7, 9, 10], 3:[1, 2, 5, 7, 9, 10], 4:[1, 2, 3, 5, 6, 7, 9, 10], 5:[6, 7, 9, 10], 6:[1, 2, 3, 7,
8, 9, 10], 7:[1, 2, 3, 5, 9, 10], 8:[1, 2, 3, 4, 5, 7, 9, 10], 9:[1,2, 3, 4, 5, 7, 10], 10:[1, 3, 5, 6, 7 ]}

print("Caminos hamiltonianos para el grafo 10")


for i in range (1,(len(graph10)+1)):
  PrintAllHamiltonianPaths(graph10,[i],"graph10.txt")


tiempo_final10 = time()
tiempo_ejecucion10 = tiempo_final10 - tiempo_inicial10

print ('El tiempo de ejecucion para el grafo 10:',tiempo_ejecucion10)



f = open("graph10.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 10 v□ rtices")
f.close()

##########################################################################################################
##########################################################################################################
#######################

tiempo_inicial11 = time()

graph11 = {1: [2, 3, 5, 7, 10, 11], 2: [1, 5, 6, 7, 9, 11], 3: [1, 2, 5, 7, 9, 11], 4: [1, 2, 3, 5, 6, 7, 11], 5: [3, 6, 9, 11], 6: [
1, 3, 4, 7, 8, 9, 10, 11], 7:[2, 3, 5, 9, 11], 8:[1, 2, 3, 4 ,5, 7, 9, 10, 11], 9:[1, 2, 3, 4, 5, 7, 10], 10:[1, 2, 3, 4, 5, 6, 7, 11]
, 11:[9]}

print ("Caminos hamiltonianos para el grafo 11")

for i in range (1,(len(graph11)+1)):
   PrintAllHamiltonianPaths(graph11,[i],"graph11.txt")

tiempo_final11 = time()
tiempo_ejecucion11 = tiempo_final11 - tiempo_inicial11

print ('El tiempo de ejecucion para el grafo 11 fue:',tiempo_ejecucion11)


f = open("graph11.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 11 v□ rtices")
```

```
f.close()


##########################################################################################################################################################################

tiempo_inicial12 = time()

graph12 = {1: [2, 3, 5, 7, 10, 11, 12], 2: [1, 5, 6, 7, 9, 11, 12], 3: [1, 2, 5, 7, 8, 9, 11, 12], 4: [1, 2, 3, 5, 6, 7, 11, 12], 5 : [ 6, 9, 11, 12], 6: [1, 3, 4, 7, 8, 9, 10, 11, 12], 7:[2, 5, 9, 11, 12], 8:[1, 2, 3, 4 ,5, 7, 9, 10, 11], 9:[1, 2, 3, 4, 5, 7, 10], 10:[1, 2, 3, 4, 5, 6, 7, 11], 11:[9,12], 12:[5, 8, 9, 10]}

print ("Caminos hamiltonianos para el grafo 12")

for i in range (1,(len(graph12)+1)):
   print(i)
   PrintAllHamiltonianPaths(graph12,[i],"graph12.txt")

tiempo_final12 = time()
tiempo_ejecucion12 = tiempo_final12 - tiempo_inicial12

print ('El tiempo de ejecucion para el grafo 12 fue:',tiempo_ejecucion12)


f = open("graph12.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 12 v rtices")
f.close()

##########################################################################################################################################################################

tiempo_inicial15 = time()

graph15 = {1: [2,3,5,10,11,12,13,14], 2:[1,5,7,9,10,11,12,13,15], 3:[1,2,5,7,9,10,11,12,15], 4: [1,2,3,5,6,7,9,10,12,15], 5:[2,3,6,7,9,10,11,12,14,15], 6: [1,2,3,7,8,9,10,12,14,15], 7:[1,2,3,5,9,10,12,15], 8:[1,2,3,4,5,7,9,11,14,15], 9:[1,2,3,5,7,10,11,13,15], 10:[1,3,4,5,7,8,11], 11:[3,4,6,7,9,12,13,14], 12:[2,3,8,9,10,13,14,15], 13:[1,2,3,4,5,6,7,8,9,10,14,15], 14:[2,3,4,5,7,9,10,15], 15:[1,2,3,6,7,8,9,10,11,13,14]}
print ("Caminos hamiltonianos para el grafo de la experimentacion con los datos de la econom a de 15 sectores")

for i in range (1,(len(graph15)+1)):
   i=1
   print i
   PrintAllHamiltonianPaths(graph15,[i],"2.0graph15.txt")

tiempo_final15 = time()
tiempo_ejecucion15 = tiempo_final15 - tiempo_inicial15

print ('El tiempo de ejecucion para el grafo de los datos econ micos de 15 sectores fue:',tiempo_ejecucion15)


f = open("2.0graph15.txt", "r")
print ("Se han encontrado:", len(f.readlines()), "caminos hamiltonianos para el grafo de 15 v rtices")
```

```
f.close()
```

Bibliografía

[1] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity.* Englewood cliffs, 1998.

[2] Rafael Marti. *Procedimientos metaheuristicos en optimizacion combinatoria.* Universitat de Valencia, 2003.

[3] Josu Ceberio, Alexander Mendiburu, and Jose A. Lozano. The linear ordering problem revisted. Technical report, University of The Basque Country, 2014.

[4] Borja Calvo, Josu Ceberio, and Usue Mori. *Bilaketa heuristikoak - Teoria eta Adibideak R Lengoaian.* Universidad del País Vasco - Euskal Herriko Unibertsitatea, 2017.

[5] Ricardo Peña Marí. El problema que los informáticos no han podido resolver en 45 años: `https://elpais.com/tecnologia/2017/05/19/actualidad/1495202801_698394.html`, May 2017.

[6] Rafael Marti and Gerhard Reinelt. *The linear ordering problem.* Springer, 2011.

[7] Leticia Hernando, Alexander Mendiburu, and Jose A. Lozano. Journey to the center of the linear ordering problem. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, page 201–209, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Robin Wilson. *Introduction to graph theory.* Prentice hall, 1996.