



**Advertencia: Queda prohibido el uso de cualquier modelo de IA para resolver los ejercicios**, su uso será sancionado. Los ganadores deberán sustentar su código en clase, esto para evitar el uso de cualquier Inteligencia Artificial y de cualquier trampa en general.

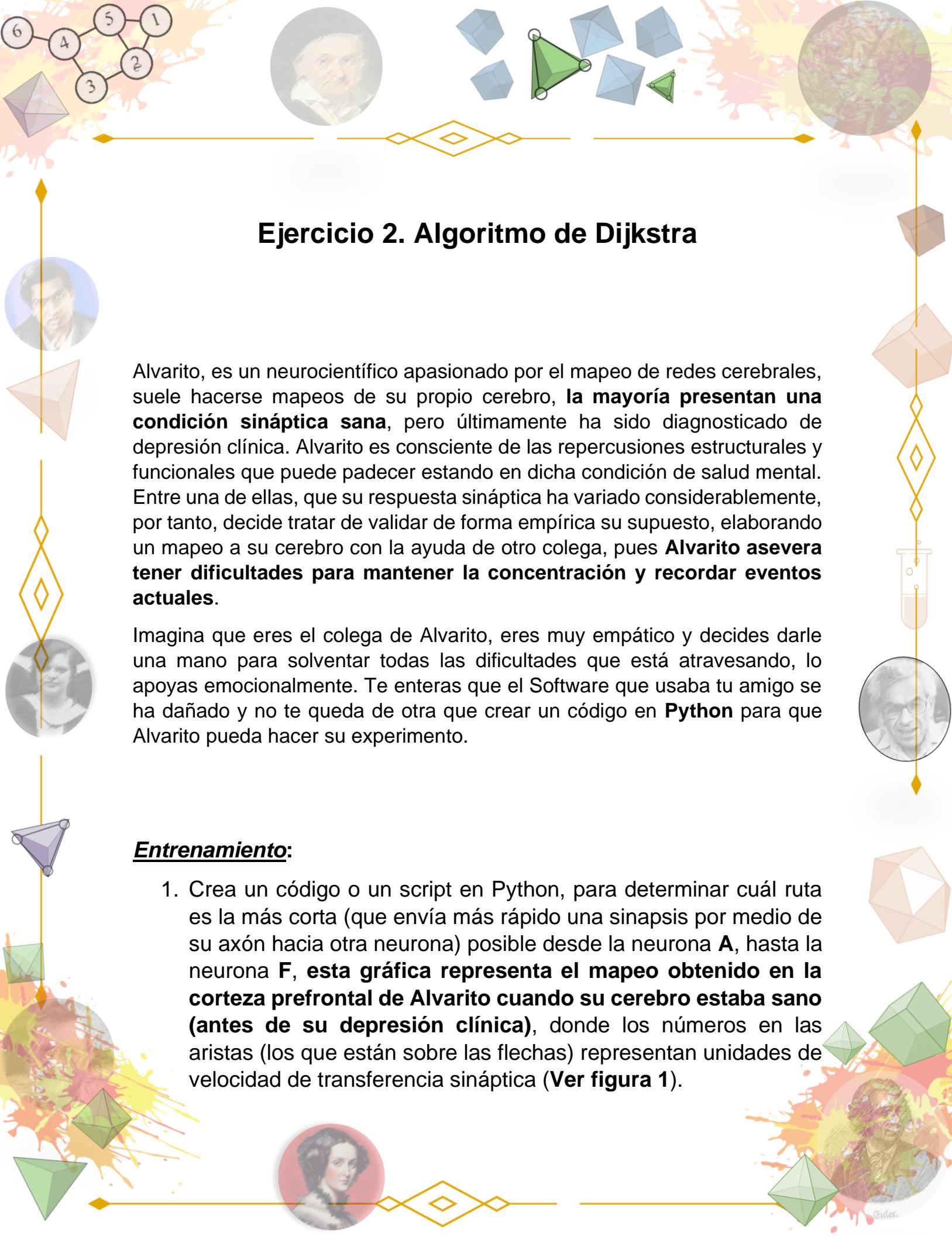
**El ejercicio es individual, está prohibido hacerlo en equipo.**

**No está permitido el uso de ninguna librería externa de Python o Framework para resolver los ejercicios, usa Python puro (Vanilla).**

**Consejo:**

*Evita escribir tu código con tildes, ñ, Ñ, o algún acento, para que el Custom Checker de los Tests, o el calificador, no te de error:*

Your submission contains non ASCII characters, we dont accept submissions with non ASCII characters for this challenge.



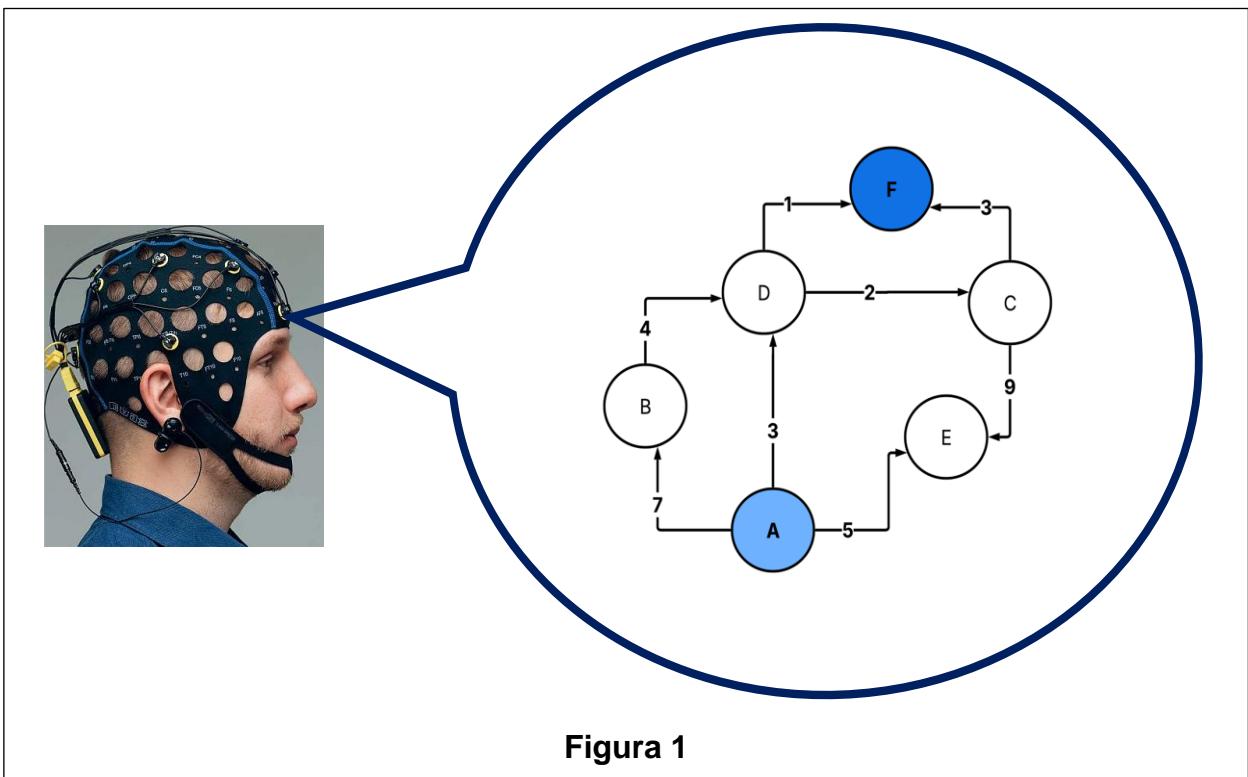
## Ejercicio 2. Algoritmo de Dijkstra

Alvarito, es un neurocientífico apasionado por el mapeo de redes cerebrales, suele hacerse mapeos de su propio cerebro, **la mayoría presentan una condición sináptica sana**, pero últimamente ha sido diagnosticado de depresión clínica. Alvarito es consciente de las repercusiones estructurales y funcionales que puede padecer estando en dicha condición de salud mental. Entre una de ellas, que su respuesta sináptica ha variado considerablemente, por tanto, decide tratar de validar de forma empírica su supuesto, elaborando un mapeo a su cerebro con la ayuda de otro colega, pues **Alvarito asevera tener dificultades para mantener la concentración y recordar eventos actuales.**

Imagina que eres el colega de Alvarito, eres muy empático y decides darle una mano para solventar todas las dificultades que está atravesando, lo apoyas emocionalmente. Te enteras que el Software que usaba tu amigo se ha dañado y no te queda de otra que crear un código en **Python** para que Alvarito pueda hacer su experimento.

### Entrenamiento:

1. Crea un código o un script en Python, para determinar cuál ruta es la más corta (que envía más rápido una sinapsis por medio de su axón hacia otra neurona) posible desde la neurona **A**, hasta la neurona **F**, **esta gráfica representa el mapeo obtenido en la corteza prefrontal de Alvarito cuando su cerebro estaba sano (antes de su depresión clínica)**, donde los números en las aristas (los que están sobre las flechas) representan unidades de velocidad de transferencia sináptica (**Ver figura 1**).



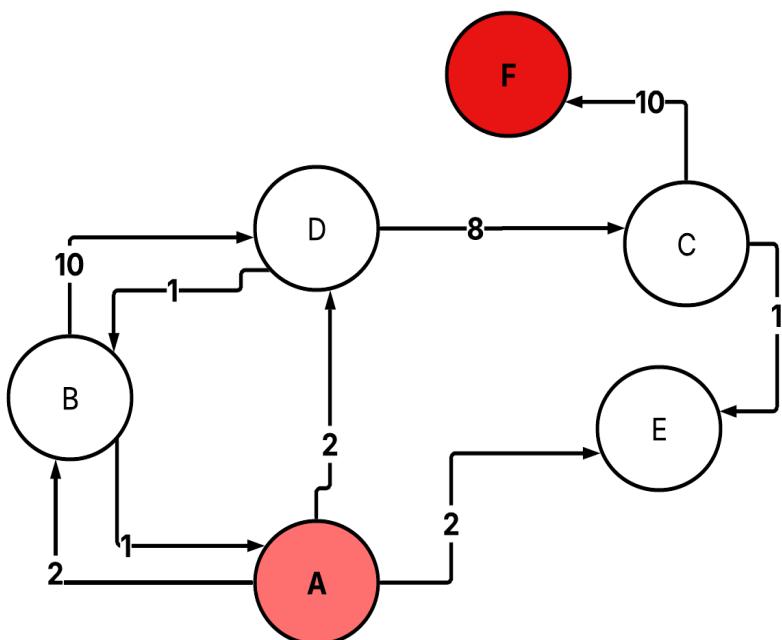
**Figura 1**

El grafo anterior se debe representar como una matriz cuadrada, para este caso inicia desde la letra A y termina hasta la F.

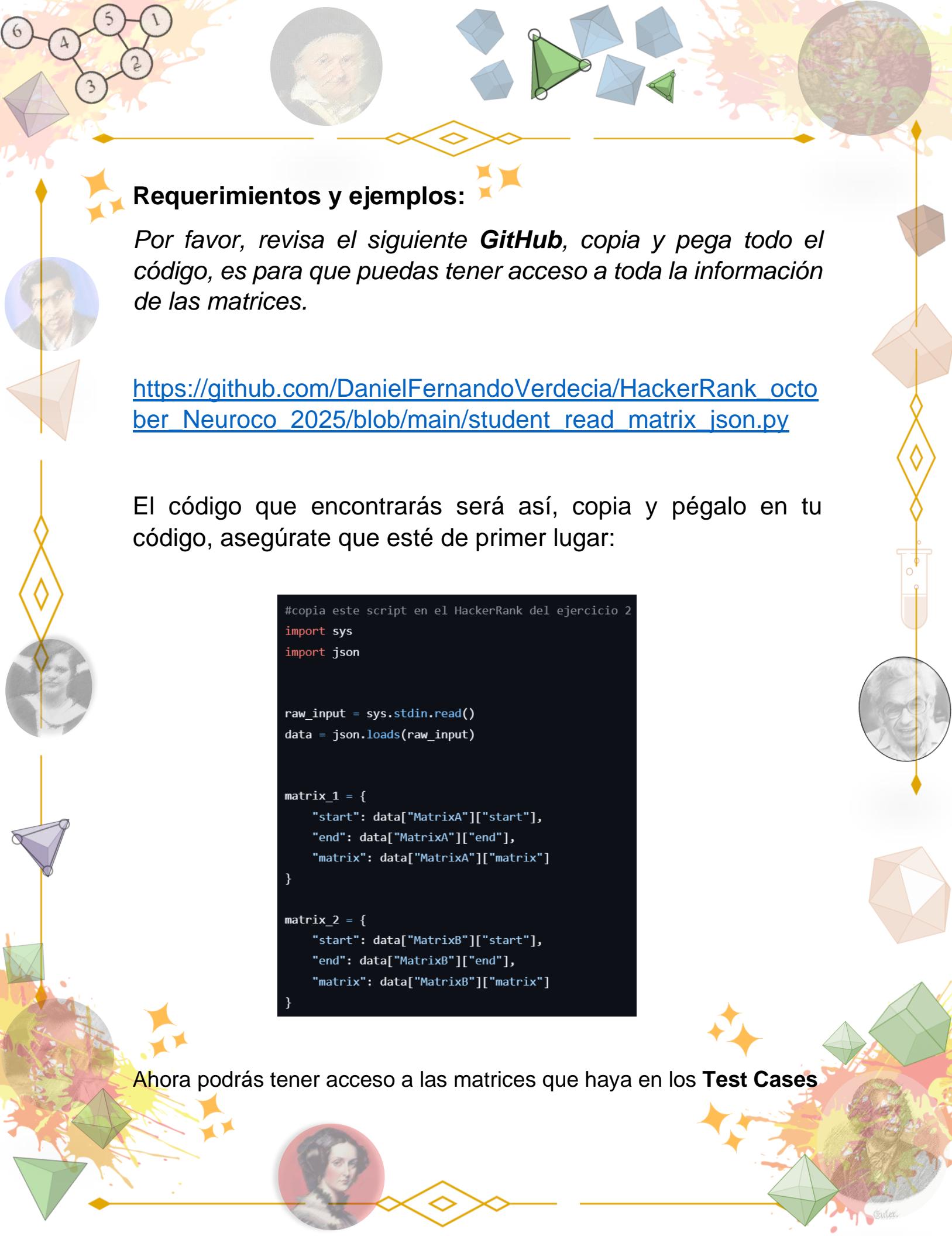
Fíjate que es un grafo dirigido, y nos fijamos es desde donde hacia donde llega cada punto y también tenemos en cuenta los valores de cada arista.

```
[# A B C D E F
 [0, 7, 0, 3, 5, 0],
 [0, 0, 0, 4, 0, 0],
 [0, 0, 0, 0, 9, 3],
 [0, 0, 2, 0, 0, 1],
 [0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0],]
```

2. Usando el código del punto 1 realiza una comparación de la ruta sináptica más corta en la **Figura 1** (Corteza prefrontal sana de Alvarito), con la ruta sináptica más corta del mapeo cerebral de Alvarito teniendo depresión (**Ver figura 2**), compara si realmente hay algún cambio de velocidad entre ambas rutas cortas, calcula la diferencia entre la ruta sana y la ruta con depresión.



**Figura 2**



## Requerimientos y ejemplos:

Por favor, revisa el siguiente **GitHub**, copia y pega todo el código, es para que puedas tener acceso a toda la información de las matrices.

[https://github.com/DanielFernandoVerdecia/HackerRank\\_october\\_Neuroco\\_2025/blob/main/student\\_read\\_matrix\\_json.py](https://github.com/DanielFernandoVerdecia/HackerRank_october_Neuroco_2025/blob/main/student_read_matrix_json.py)

El código que encontrarás será así, copia y pégalo en tu código, asegúrate que esté de primer lugar:

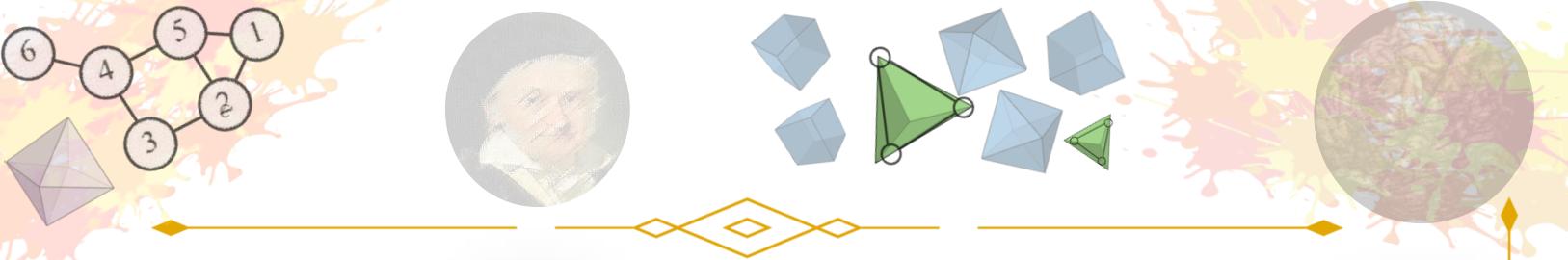
```
#copia este script en el HackerRank del ejercicio 2
import sys
import json

raw_input = sys.stdin.read()
data = json.loads(raw_input)

matrix_1 = {
    "start": data["MatrixA"]["start"],
    "end": data["MatrixA"]["end"],
    "matrix": data["MatrixA"]["matrix"]
}

matrix_2 = {
    "start": data["MatrixB"]["start"],
    "end": data["MatrixB"]["end"],
    "matrix": data["MatrixB"]["matrix"]
}
```

Ahora podrás tener acceso a las matrices que haya en los **Test Cases**



Muy bien, continuemos con lo que realmente deberás programar:

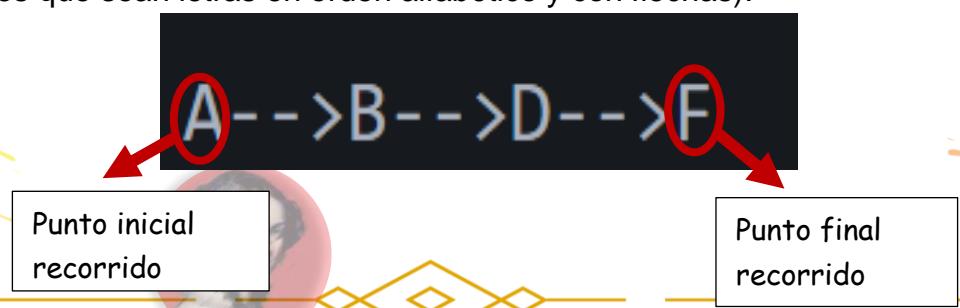
- Crea una función (por convención la denotaremos aquí como **Fun1()**), puede llamarse como quieras, lo más importante es que **reciba 3 parámetros** (léase de izquierda a derecha), el primer parámetro será una matriz cuadrada (recuerda adaptar la matriz al problema, donde cada fila representa letras desde la letra A, B, C, etc. Y cada columna representa las letras A, B, C, etc. **Recuerda cada fila son las horizontales y las columnas verticales**, puedes guiarte con el ejemplo de la matriz de listas de la **Figura 1**). El segundo parámetro será un número entero desde el número cero (cada número representará una letra, ejemplo: 0 es la A, 1 es la B, 2 es la C, ...) estos números representan desde qué punto se comenzará a comprobar cuál es el camino más corto posible.

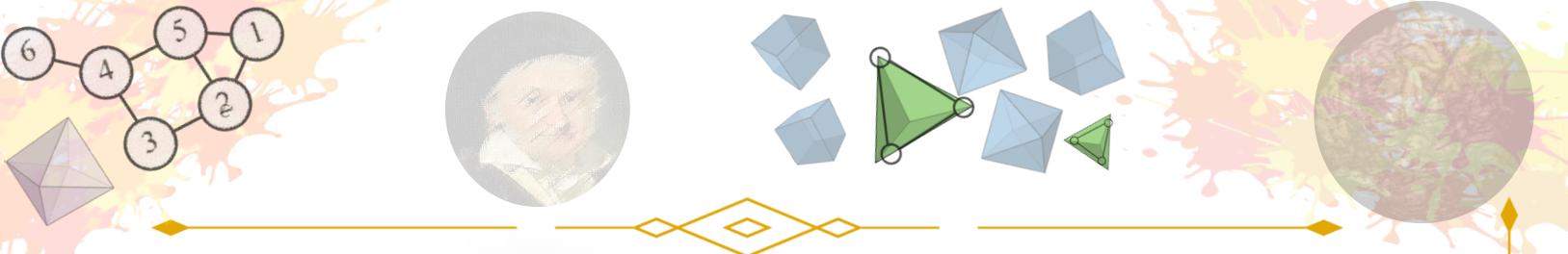
El tercer parámetro representará el punto final donde se acabará el recorrido de la búsqueda de la ruta o camino más corto (también serán números enteros, 0 es la A, 1 es la B, 2 es la C, ...)

*Si copiaste el código anterior de GitHub, verás que tendrás dos variables que son diccionarios, **matrix\_1** y **matrix\_2**, por ejemplo, para **Fun1()**, los tres parámetros están en **matrix\_1**, el primero es el valor de la clave “**matrix**”, el segundo es “**start**” y el tercer parámetro es el valor de la clave “**end**”*

Muy bien, la función **Fun1()** debe devolver 2 valores en orden obligatorio, *primero la distancia total calculada a partir de la ruta o camino más corto entre dos puntos dados* (es decir, suma todos los valores que hay en las aristas o durante el recorrido del camino más corto, y al final obtiene el valor total de la suma), y como segundo valor será un string, que representa primero el primer punto del inicio del recorrido para encontrar el camino más corto, finalmente está el punto final o llegada del recorrido.

Aquí tenemos un pequeño ejemplo de cómo sería el segundo valor que se debe devolver de la anterior función (puede variar, pero lo más importante es que sean letras en orden alfabético y con flechas):





Es decir, si tienes la siguiente matriz:

```
"MatrixA": {  
    "start": 0,  
    "end": 4,  
    "matrix": [  
        [0,1,1,0,0],  
        [0,0,5,0,5],  
        [0,0,0,2,8],  
        [0,0,0,0,1],  
        [0,0,0,0,0]  
    ]  
},
```

Lo que devolverá tu función es:

```
'4, A-->C-->D-->E'
```

**Note que siempre debe colocar las letras en orden alfabético, en mayúscula, sin espacios y con una flecha -->**

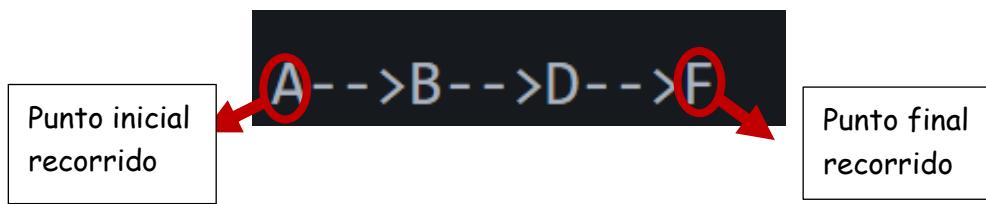
- Creamos otra función, puedes llamarla como quieras, la llamaremos en nuestro caso, **Fun2()**, esta recibirá dos matrices cuadradas (recuerda la estructura de matrices con forma de listas que definimos anteriormente).

*Básicamente tienes en cuenta para los dos parámetros los dos diccionarios dados en el GitHub, el primer parámetro será el diccionario **matrix\_1** y el segundo es **matrix\_2**.*

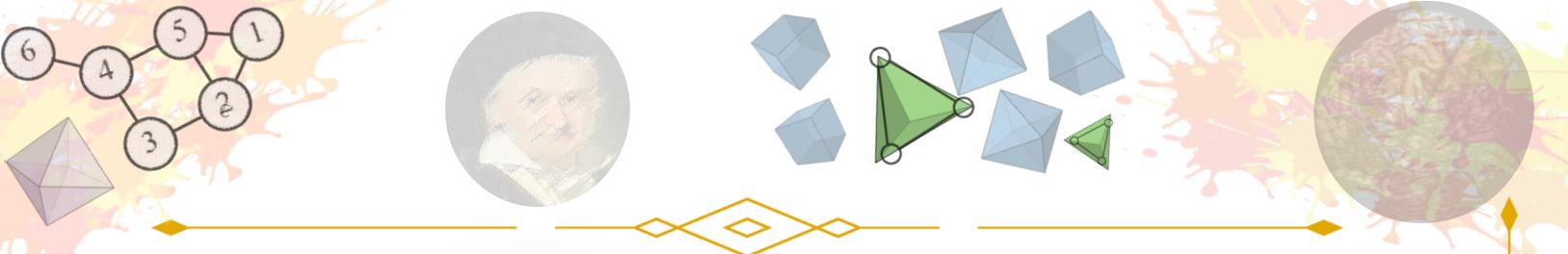
Devolveremos 3 valores como resultados:

El primer valor devuelto es el valor de la resta de la distancia total para hallar el camino más corto, entre las dos matrices, es decir valor de la **matrix\_1**, menos el valor obtenido de **matrix\_2**.

Para el segundo valor devuelto, muestra un string con los puntos recorridos para encontrar el camino más corto entre dos puntos de la **matrix\_1**, similar a la forma:



Y el tercer valor devuelto de la función, es similar al anterior, pero obtenemos los valores para la **matrix\_2**.

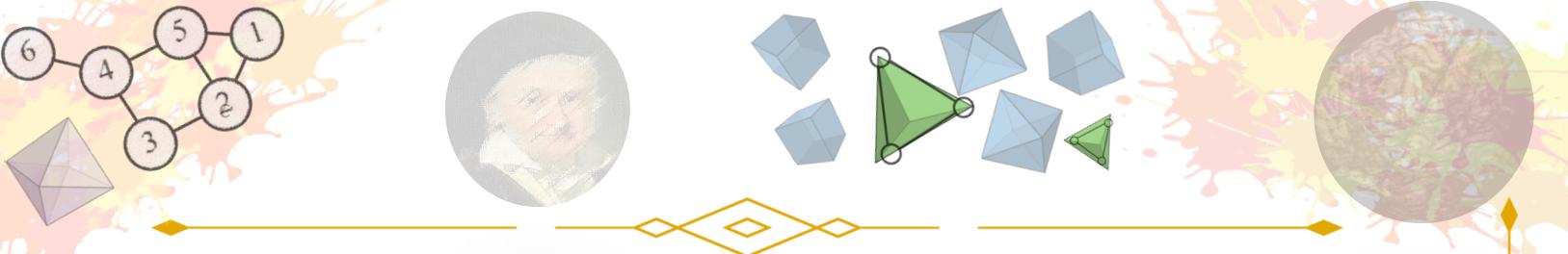


Es decir, si tienes para las dos matrices esto de ejemplo:

```
"MatrixA": {  
    "start": 0,  
    "end": 4,  
    "matrix": [  
        [0,1,1,0,0],  
        [0,0,5,0,5],  
        [0,0,0,2,8],  
        [0,0,0,0,1],  
        [0,0,0,0,0]  
    ]  
},  
"MatrixB": {  
    "start": 0,  
    "end": 4,  
    "matrix": [  
        [0,1,1,0,0],  
        [0,0,5,0,2],  
        [0,0,0,2,8],  
        [0,0,0,0,1],  
        [0,0,0,0,0]  
    ]  
}
```

La función que programes devolverá:

```
'1, A-->C-->D-->E, A-->B-->E'
```



Tranquila o tranquilo, ya terminaremos la explicación, ahora de lo más importante para que la plataforma califique tu código es que debes hacer un **print()**, el cual mostrará una lista, primero con los valores que hay en la **Fun1()** y el segundo valor será los de la **Fun2()**.

En palabras más simples, si tu INPUT es así:

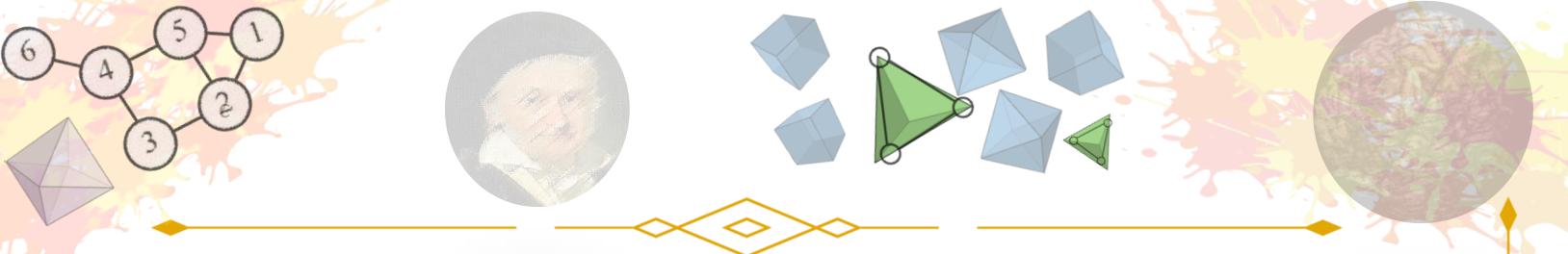
```
{  
    "MatrixA": {  
        "start": 0,  
        "end": 4,  
        "matrix": [  
            [0,1,1,0,0],  
            [0,0,5,0,5],  
            [0,0,0,2,8],  
            [0,0,0,0,1],  
            [0,0,0,0,0]  
        ]  
    },  
    "MatrixB": {  
        "start": 0,  
        "end": 4,  
        "matrix": [  
            [0,1,1,0,0],  
            [0,0,5,0,2],  
            [0,0,0,2,8],  
            [0,0,0,0,1],  
            [0,0,0,0,0]  
        ]  
    }  
}
```

El **print()** que harás será:

```
[ '4, A-->C-->D-->E' , '1, A-->C-->D-->E, A-->B-->E' ]
```

Valores  
retornados por  
**Fun1()**

Valores retornados  
por **Fun2()**



Excelente, ahora puedes empezar a programar.

 **Te deseamos muchos éxitos!!!!!!** 

Te quiero mucho



**Nota:** Debes procurar hacer un código muy óptimo, debido que **hay varios Tests o pruebas ocultas** que se calificarán con gigantescos datos, por tanto, **procura generalizar un código que sirva para todos los casos ocultos que hemos diseñado.**

