Luis Gerardo Estrada García (319013832)
Cielo López Villalba (422050461)
Dulce Julieta Mora Hernández (319236448)
Marcos Julián Noriega Rodríguez (319284061)

Esquemas de Codificación

Considera el siguiente problema:
RUTA HAMILTONIANA
EJEMPLAR: Una gráfica $G = (V, E)$
PREGUNTA: ¿G contiene una ruta hamiltoniana?

1. ¿Cómo podemos expresar este problema en su versión de búsqueda?, ¿Tiene sentido plantearse una versión de optimización?
   Para plantear el problema en su versión de búsqueda debemos definir primero el ejemplar y la pregunta

   - **Ejemplar:** Una gráfica $G = (V, E)$
   - **Pregunta:** ¿Cuál es una secuencia de vértices (de manera que estén todos los vértices de V en dicha secuencia) que forme una ruta hamiltoniano?

   No tiene sentido plantearse una versión de optimización dado que cuando buscamos una versión para optimizar un problema, lo que se busca es encontrar una solución que "mejore" al problema. Como nuestro problema es acerca de rutas hamiltonianas, solamente requirimos ver la existencia o no de una ruta hamiltoniana, es decir, un camino que visita todos los vértices de la gráfica una unica vez. Por lo que, no es sumamente útil plantearnos una versión de optimización ya que no hay algún requisito que nos indique que podemos "mejorar" la solución del problema, ya que las rutas hamiltonianas tienen las mismas características, es decir, el mismo número de vértices y aristas, aunque sean diferentes.

2. Describir e implementar un esquema de codificación para ejemplares del problema el anterior (en su versión de decisión). El esquema de codificación utilizado para gráficas, no puede ser una matriz.
   Para poder implementar un esquema de codificación para los ejemplares del problema anterior (ruta hamiltoniana) en su versión de decisión, primero debemos plantearnos cual es la versión de decisión. Veamos como plantearlo

   Problema de decisión

   - **Ejemplar:** Una gráfica $G = (V, E)$
   - **Pregunta:** ¿Existe una ruta hamiltoniana en G?

   Cadena que codifica a cada ejemplar
   Para la codificación de las gráficas tomamos archivos .txt en los cuales tenemos gráficas de la forma:

   a b
   b c
   a s
   ...

   En la que cada renglón representa cada arista de la gráfica, por lo que nuestro codificador separa los vértices que recibe de cada renglón y verifica que no sean lazos y omite aristas de la forma "a b" y "b a", dejando solo la primer arista encontrada.

```
# Separamos la arista en dos valores
v1, v2 = arista[0], arista[1]

# Omitimos lazos
if v1 == v2:
    continue

# Agregamos los vertices a la lista
if v1 not in vertices:
    vertices.append(v1)
if v2 not in vertices:
    vertices.append(v2)

# aristas de la forma 'a b' es igual que 'b a'
if v1 > v2:
    v1, v2 = v2, v1
```

```
if f"{v1}-{v2}" not in aristas:
    aristas.append(f"{v1}-{v2}")
```

Resultado de la ejecución del programa
Si tuvimos éxito al leer el archivo, imprimimos las aristas, los vértices y el número de estos en forma de lista.

```
print("Lista de aristas: [", ', '.join(aristas), "]")
print(f"Numero de vertices: {len(vertices)}")
print(f"Lista de vertices: {vertices}")
```

Complejidad
Veamos que el algoritmo realiza lo siguiente
El algoritmo recibe un archivo con la lista de aristas. Sin embargo, acotaremos la complejidad respecto al número de vértices por practicidad. Se inicializan dos listas, una para aristas y otra para vértices.

Para cada arista que se encuentra en nuestro archivo de entrada se realiza lo siguiente

- Separar la arista en dos vértices `O(1)`
- Verificar si el vértice ya se encuentra en la lista de vértices `O(n)`
  - En caso de que el vértice no esta en la lista, lo agregamos a la lista. `O(1)`
- Agrega la arista a la lista de aristas `O(1)`

Por lo que nuestro algoritmo para generar la lista de aristas toma $O(|E|)$ y la mayor cantidad de aristas que podemos tener en una gráfica respecto al número de vértices es $\frac{n(n-1)}{2}$ y generar la lista de vértices nos toma tiempo $O(n)$

3. Describir e implementar un algoritmo para transformar el esquema de codificación propuesto, a uno que utilice codificación de gráficas como matrices. El programa implementado debe recibir como entrada:

   - Nombre del archivo de entrada con el ejemplar a leer.
     El ejemplar debe seguir el esquema de codificación propuesto
   - Nombre del archivo de salida con el resultado de la codificación como matriz.

   Como resultado de la ejecución, el programa deberá producir como salida:

   - Imprimir en consola el número de vértices y aristas en la gráfica G
   - Escribir en el archivo de texto (indicado en el segundo parámetro del programa) una cadena (como texto plano) que represente los parámetros del ejemplar (aplicando un esquema de codificación como matriz para la gráfica).

   Nota: Para optar por el 10, deberán implementar la salida en el archivo con un esquema de codificación que haga uso únicamente del alfabeto binario; en caso de que hagan la implementación con otra codificación, podrían tener una penalización.

   Cadena que codifica a cada ejemplar
   En este caso, para la codificación de las gráficas recibimos archivos .txt en los cuales se reciben gráficas como listas de aristas, de la forma:

   a b
   b c
   a s
   ...

   En esta representación, se representa una arista por renglón. Nuestro codificador se encarga de dadas dichas aristas, separar los vértices a partir de ellas, verifica que no sean lazos y en caso de encontrar aristas que tengan la forma "a b" y "b a", toma en cuenta solo la primera encontrada. Una vez realizado lo anterior, el programa crea una matriz de $n \times n$ a partir de la cantidad de vértices que obtengamos. Una vez creada dicha matriz, lee una vez más el archivo que recibimos como entrada(el cual contiene la lista de aristas), y renglon por renglon, separa los vertices de cada arista, y marca que existe la arista en la matriz de adyacencia para M[v_1][v_2] y M[v_2][v_1]

```
# Separamos la arista en dos valores
v1, v2 = arista[0], arista[1]

# Omitimos lazos
if v1 == v2:
```

```
            continue

        # Agregamos los vertices a la lista
        if v1 not in vertices:
            vertices.append(v1)
        if v2 not in vertices:
            vertices.append(v2)

        # Normalizamos la representacion para que 'a b' sea igual que 'b a'
        if v1 > v2:
            v1, v2 = v2, v1

        # Agregamos la arista si no estaba en la lista
        if f"{v1}-{v2}" not in aristas:
            aristas.append(f"{v1}-{v2}")


        # Crear una matriz de adyacencia de tamano n x n
        cantidad_vertices = len(vertices)
        matriz_adyacencia = [[0] * cantidad_vertices for _ in
        range(cantidad_vertices)]

        # Volver a leer el archivo para llenar la matriz
        with open(archivo_entrada, 'r') as archivo:
            for linea in archivo:
                arista = linea.strip().split("-")

                # Separamos la arista en dos valores
                v1, v2 = arista[0], arista[1]

                # Obtener los indices de los vertices
                indice1 = vertices.index(v1)
                indice2 = vertices.index(v2)

                # Marcar las posiciones en la matriz de adyacencia
                matriz_adyacencia[indice1][indice2] = 1
                matriz_adyacencia[indice2][indice1] = 1
```

Resultado de la ejecución del programa
Si tuvimos éxito al leer el archivo y no tuvimos problemas en la ejecución, en terminal se regresa lo que sigue:

- Lista de vértices

- Número de vértices
- Lista de aristas
- Represenación visual de la matriz de adyacencia; además, se indica el archivo en el que se guardo la matriz cuyo nombre es proporcionado al ejecutar el programa.

Además, el programa escribe en el archivo de salida la matriz utilizando un alfabeto binario, a saber, $\{ 0, 1 \}$, por lo que simplemente se muestra un renglon que contiene a la matriz

```
# Escribir la matriz en el archivo de salida
    with open(archivo_salida, 'w') as archivo:
        for fila in matriz_adyacencia:
            archivo.write("".join(map(str, fila)))
```

Complejidad
Veamos que el algoritmo realiza lo siguiente

- Primero se lee el archivo que recibe como entrada el programa y notemos que como el archivo de entrada contiene una lista de $|E|$ aristas. Sin embargo, acotaremos la complejidad respecto al número de vértices

  El programa realiza lo siguiente para cada renglón
  - Separa la arista en dos vértices. $O(1)$
  - Verifica si el vértice ya se encuentra en la lista de vértices, lo que nos toma tiempo lineal respecto a la lista que se creo
  - En caso de no estar lo agregamos a la lista. $O(1)$
  - Agrega la arista a la lista de aristas $O(1)$
- Después, el programa crea una matriz de $n \times n$ donde n indica la cantidad de vértices que tiene nuestra gráfica y vuelve a leer el archivo. Para cada arista se realiza lo siguiente
  - Separamos la arista en dos vértices y guardamos cada vértice. $O(1)$
  - Accedemos a la matriz en la posición M$[v_1][v_2]$ y M$[v_2][v_1]$. $O(1)$
  - Marcamos que existe la arista. $O(1)$

La complejidad de nuestra función para agregar toma $O(n^2)$ respecto a la cantidad de vértices, ya que tenemos que rellenar una matriz nos toma dicho tiempo y todas las demás operaciones a realizar toman menor tiempo, por lo que concluimos que realizar nuestra matriz de adyacencias toma tiempo cuadratico.

4. Describir e implementar un algoritmo eficiente para determinar si una gráfica tiene una ruta Euleriana. La implementación del algoritmo debe hacer uso del esquema de codificación implementado en el inciso anterior. El programa implementado debe recibir como entrada (desde consola) el nombre del archivo que contiene el ejemplar a probar, y como salida deberá imprimir:

   - Número de Vértices
   - Número de Aristas
   - Vértice de mayor grado (nombre o etiqueta del vértice y valor del grado)
   - ¿La gráfica del ejemplar tiene un camino euleriano? [Responder con un SI o un NO]
   - En caso de que la gráfica sea SI, indicar el camino euleriano.

   Este programa recibe un archivo codificado con el esquema propuesto para luego transformar la representación a una matriz y con ella poder obtener todo lo que buscamos.

```
def informacion_grafica(archivo_entrada):
    vertices, aristas, matriz_adyacencia =
    transformar_archivo(archivo_entrada)

    if vertices and aristas and matriz_adyacencia:
        print(f"Numero de Vertices: {len(vertices)}")
        print(f"Numero de Aristas: {len(aristas)}")

        vertice_mayor, grado_mayor =
        vertices_mayor_grado(vertices, matriz_adyacencia)
        print(f"Vertice(s) de mayor grado:
--------{vertice_mayor} (grado {grado_mayor})")

        resultado_euleriano, impares =
        ruta_euleriana(vertices, matriz_adyacencia)
        print(f" La grafica tiene una ruta euleriana?
--------{resultado_euleriano}")

        if resultado_euleriano.startswith("SI"):
            inicio = impares[0] if impares else 0
            ruta = encontrar_ruta_euleriana(vertices,
            matriz_adyacencia, inicio)
            print(f"Ruta Euleriana: {' -> '.join(ruta)}")
```

   Este primer paso de obtener la matriz toma $O(n^2)$ con $n = |V|$, es decir, el número de vértices de la gráfica, aunque la entrada esté en términos del número de aristas, este

número se puede acotar en el peor caso por $O(n^2)$ ya que a lo más hay $|E| = \frac{n(n-1)}{2}$ aristas en la gráfica.

```
vertices, aristas, matriz_adyacencia =
transformar_archivo(archivo_entrada)
```

Una vez hecho esto, obtenemos una lista de vértices, otra de aristas y la matriz de adyacencia de la gráfica.

Con esto podemos calcular el número de vértices y de aristas, con solo tomar la longitud de las listas respectivamente, lo cual toma $O(n)$ para la de vértices y en el peor caso $O(n^2)$ para la de aristas.

```
print(f"Numero de Vertices: {len(vertices)}")
print(f"Numero de Aristas: {len(aristas)}")
```

Para obtener el vértice o vértices de mayor grado, suma cada renglón de la matriz, lo cual da en total el grado de cada vértice, ya que los renglones representan vértices y solo se toma en cuenta una arista por vértice y no hay lazos. Con esto recorremos toda la matriz lo que es $O(n^2)$, además guardamos el grado de cada vértice y para encontrar el mayor recorremos dicha lista en $O(n)$, para finalmente volverla a recorrer toda para obtener todos los vértices que tengan grado igual al mayor, guardandolos en una lista para devolver la lista y el número del grado mayor. Con lo cual esto en total toma $O(n^2)$.

```
def vertices_mayor_grado(vertices, matriz_adyacencia):
    grados = grado_vertices(matriz_adyacencia)
    max_grado = max(grados)
    vertices_mayores_grado = [vertices[i] for i, grado in
    enumerate(grados) if grado == max_grado]
    return vertices_mayores_grado, max_grado
```

Para verificar si la gráfica tiene o no un camino euleriano, vemos si tiene más de 2 vértices de grado impar, con lo cual nos lleva $O(n^2)$ obtener los grados y $O(n)$ recorrer toda la lista en el peor caso, después si se cumple la condición devolvemos "SI" o "No" dependiendo del resultado. En caso de que tenga exactamente 2 vértices de grado impar, los devolvemos junto con la respuesta de decisión para identificarlos más fácil al buscar el camino.

```
# Verificar si la grafica tiene un ruta Euleriana.
def ruta_euleriana(vertices, matriz_adyacencia):
    grados = grado_vertices(matriz_adyacencia)
    impares = [i for i, grado in enumerate(grados) if grado % 2 != 0]
```

```
    if len(impares) == 0:
        return "SI, tiene una ruta Euleriana", None
        #si esto sucede, es ademas un ciclo Euleriano.
    elif ((len(impares) == 2) or (len(impares) == 1)):
        return "SI, tiene una ruta Euleriana", impares
    else:
        return "NO, no tiene una ruta Euleriana", None
```

En caso de que no tenga camino euleriano terminamos, sino, buscamos la ruta euleriana en cuestión. Para esto partimos de un vértice "*inicio*" y buscamos en una copia de la matriz un vértice adyacente a él u, después eliminamos la arista de la matriz y repetimos el proceso recursivamente con u. Esto hasta que no haya más aristas en la gráfica o que todas las entradas de la matriz sean 0. Después agregamos a una lista el índice del último vértice del camino, para terminar ese proceso recursivo y hacemos lo mismo con los demás vértices que estuvieron involucrados en el proceso recursivo, al final obtenemos la ruta en orden inverso en el que la calculamos.

Con esto, obtenemos una lista de esa ruta, pero en términos de los vértices y no de sus índices de la matriz, para luego invertirla y devolver la lista como resultado del camino euleriano.

Todo esto se hace en $O(n^2)$, ya que a pesar de que es un proceso recursivo, lo que termina pasando en el peor caso, es que se recorre toda la matriz ya que la gráfica es completa, es decir, tiene $O(n^2)$ aristas, y como la matriz tiene tamaño $n \times n$, entonces, recorrerla toda lleva $O(n^2)$. Después cambiar la lista del camino en términos de los vértices e invertirla, toma $O(n^2)$ igualmente ya que recorremos toda la lista que tiene tamaño $O(n^2)$ porque hay $O(n^2)$ aristas en la gráfica.

```
# Algoritmo para encontrar una ruta Euleriana. O(v^2)
def encontrar_ruta_euleriana(vertices, matriz_adyacencia, inicio):
    matriz_aux = [fila[:] for fila in matriz_adyacencia]
    ruta = []

    def recorrer(v):
        for u in range(len(vertices)):
            if matriz_aux[v][u] > 0:
                matriz_aux[v][u] -= 1
                matriz_aux[u][v] -= 1
                recorrer(u)
        ruta.append(v)

    recorrer(inicio)
```

```
    ruta = [vertices[v] for v in ruta]
    return ruta[::-1]
    # Invertir la ruta para obtener el orden correcto.
```

Como vemos varios pasos toman $O(n^2)$, pero no más, con lo cual como la complejidad está en términos del tamaño de la entrada que a grandes rasgos es la gráfica, entonces la complejidad final de nuestro algoritmo es $O(n^2)$ con $n = |V|$ y por ende también es un algoritmo eficiente.

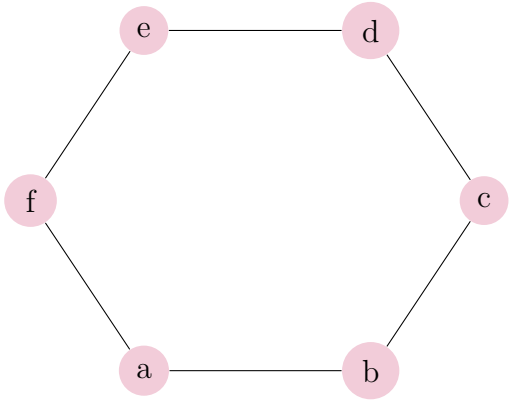
5. Incluir ejemplos de ejecución, así como ejemplos gráficos de los ejemplares con los que prueben su programa. Al menos deben agregar ejemplos de los siguientes casos (con al menos 6 vértices en cada caso)
   Antes de representar las gráficas, vamos a tomar la siguiente preposición que nos dice si una gráfica tiene o no es una ruta euleriana (tomada del libro "Introduction to Graph Theory" de Douglas B. West)

4.2.21. *If G is $2k$-edge-connected and has at most two vertices of odd degree, then G has a k-edge-connected orientation.* It suffices to orient the edges so that at least k edges leave each nonempty proper subset of the vertices. When $k = 0$, the statement is trivial, so we may assume that $k > 0$.

Since G has at most two vertices of odd degree, G has an Eulerian trail. Choose an Eulerian trail T. Let D be the orientation obtained by orienting each edge of G in the direction in which T traverses it. Let $[S, \overline{S}]$ be an edge cut of G. When crossing the cut, the trail alternately goes from one side and then from the other, so it alternately orients edges leaving or entering S. Since G is $2k$-connected, $|[S, \overline{S}]| \geq 2k$, and the alternation means that at least k edges leave each side in the orientation.

La cual, usaremos para demostrar cuando una gráfica no tiene una ruta euleriana (si encontramos dos o más vértices de grado impar)

- Ejemplar con una ruta hamiltoniana y una ruta euleriana.
  Una gráfica $G = (V, E)$ donde $V = \{ a, b, c, d, e, f \}$ y $E = \{ ab, bc, cd, de, ef, fa \}$



Veamos que la gráfica propuesta cuenta con una ruta hamiltoniana ya que si es posible visitar todos los vértices exactamente una vez porque la gráfica es un ciclo; además, la ruta euleriana son todas las aristas ya que al ser un ciclo podemos recorrer cada arista exactamente una vez obteniendo asi la ruta euleriana

Tomamos la siguiente ruta Hamiltoniana H (representada por los vértices) y la siguiente ruta Euleriana E (representada por las aristas, la cual en sí es un ciclo, pero la tomaremos como ruta también):

$H = (e, f, a, b, c, d, e)$
$E = (ab, bc, cd, de, ef, fa)$

La cadena que codifica este ejemplar se encuentra en el archivo "Ejemplar1.txt", el cual contiene lo siguiente:

a b
b c
c d
d e
e f
f a

Al pasar el ejemplar 1 en el transformador obtenemos lo siguiente en terminal:

```
→  Practica01 python3 transformador.py Ejemplar1.txt 1Solucion.txt
Lista de vértices: ['a', 'b', 'c', 'd', 'e', 'f']
Número de vértices: 6
Lista de aristas: [ a b, b c, c d, d e, e f, a f ]
Matriz de adyacencia:
0 1 0 0 0 1
1 0 1 0 0 0
0 1 0 1 0 0
0 0 1 0 1 0
0 0 0 1 0 1
1 0 0 0 1 0
Matriz guardada correctamente en el archivo: 1Solucion.txt
```

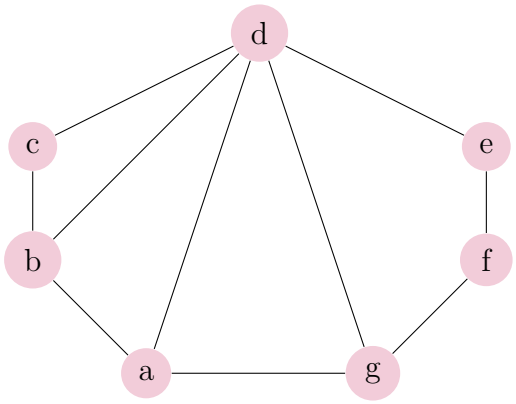
El archivo que elegimos para regresar el resultado sí nos lo da en binario. En este caso, "1Solucion.txt" tiene lo siguiente:

010001101000010100001010000101100010

Al final, el Ejemplar 1 con el archivo "euler.py" nos regresa lo siguiente:

```
→  Practica01 python3 euler.py Ejemplar1.txt
Número de Vértices: 6
Número de Aristas: 6
Vértice(s) de mayor grado: ['a', 'b', 'c', 'd', 'e', 'f'] (grado 2)
¿La gráfica tiene una ruta euleriana? SI, tiene una ruta Euleriana
Ruta Euleriana: a -> b -> c -> d -> e -> f -> a
```

- Ejemplar con una ruta hamiltoniana y que NO contenga una ruta euleriana.
  Una gráfica $G = (V, E)$, donde $V = \{ a, b, c, d, e, f, g \}$ y
  $E = \{ ab, bc, cd, de, ef, fg, ga, bd, ad, gd \}$



Notemos que esta gráfica cumple con tener una ruta hamiltoniana ya que contamos con un ciclo que une a todos los vértices; sea H el ciclo Hamiltoniano:

$H = (d, c, b, a, g, f, e, d)$

Sin embargo, no es posible tener una ruta euleriana ya que tenemos los vértices b, a y g tienen grado impar.

La cadena que codifica este ejemplar se encuentra en el archivo "Ejemplar2.txt", el cual contiene lo siguiente:

a b
b c
c d
d e
e f
f g
g a
b d
a d
g d

Al pasar el ejemplar 2 en el transformador obtenemos lo siguiente en terminal:

```
→  Practica01 python3 transformador.py Ejemplar2.txt 2Solucion.txt
Lista de vértices: ['a', 'b', 'c', 'd', 'e', 'f', 'g']
Número de vértices: 7
Lista de aristas: [ a b, b c, c d, d e, e f, f g, a g, b d, a d, d g ]
Matriz de adyacencia:
0 1 0 1 0 0 1
1 0 1 1 0 0 0
0 1 0 1 0 0 0
1 1 1 0 1 0 1
0 0 0 1 0 1 0
0 0 0 0 1 0 1
1 0 0 1 0 1 0
Matriz guardada correctamente en el archivo: 2Solucion.txt
```

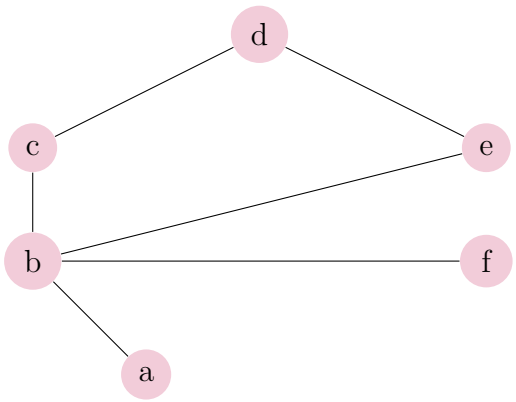
El archivo que elegimos para regresar el resultado sí nos lo da en binario. En este caso, "2Solucion.txt" tiene lo siguiente:

01010011011000010100011101010001010000010110010 10

Al final, el Ejemplar 2 con el archivo "euler.py" nos regresa lo siguiente:

```
→  Practica01 python3 euler.py Ejemplar2.txt
Número de Vértices: 7
Número de Aristas: 10
Vértice(s) de mayor grado: ['d'] (grado 5)
¿La gráfica tiene una ruta euleriana? NO, no tiene una ruta Euleriana
```

- Ejemplar que NO contenga una ruta hamiltoniana y que contenga una ruta euleriana.
  Una gráfica $G = (V, E)$ donde $V = \{a, b, c, d, e, f\}$ y $E = \{ab, bc, cd, de, eb, bf\}$



Notemos que esta gráfica cumple con no tener una ruta hamiltoniana ya que, si queremos pasar por los vértices f y a, tendremos que regresar al vértice b para poder ir al otro vértice (ya sea a o f); como pasamos por el vértice b más de una vez, ya no es una ruta hamiltoniana.

En cambio, tomamos la siguiente ruta euleriana representada por aristas:

$E = (ab, bc, cd, de, eb, bd)$

Lo cual es una ruta válida ya que pasa por todas las aristas de la gráfica.

La cadena que codifica este ejemplar se encuentra en el archivo "Ejemplar3.txt", el cual contiene lo siguiente:

a b
b c
c d
d e
e b
b f

Al pasar el ejemplar 3 en el transformador obtenemos lo siguiente en terminal:

```
→  Practica01 python3 transformador.py Ejemplar3.txt 3Solucion.txt
Lista de vértices: ['a', 'b', 'c', 'd', 'e', 'f']
Número de vértices: 6
Lista de aristas: [ a b, b c, c d, d e, b e, b f ]
Matriz de adyacencia:
0 1 0 0 0 0
1 0 1 0 1 1
0 1 0 1 0 0
0 0 1 0 1 0
0 1 0 1 0 0
0 1 0 0 0 0
Matriz guardada correctamente en el archivo: 3Solucion.txt
```

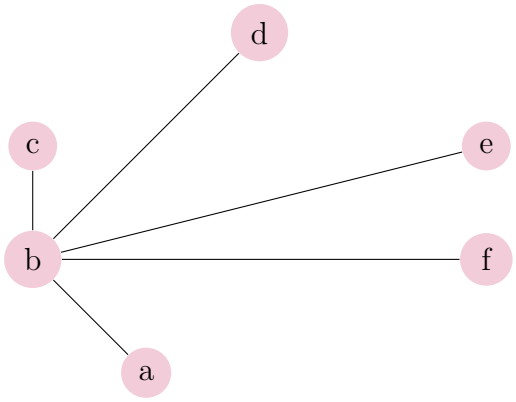
El archivo que elegimos para regresar el resultado sí nos lo da en binario. En este caso, "3Solucion.txt" tiene lo siguiente:

010000101011010100001010010100010000

Al final, el Ejemplar 3 con el archivo "euler.py" nos regresa lo siguiente:

```
→  Practica01 python3 euler.py Ejemplar3.txt
Número de Vértices: 6
Número de Aristas: 6
Vértice(s) de mayor grado: ['b'] (grado 4)
¿La gráfica tiene una ruta euleriana? SI, tiene una ruta Euleriana
Ruta Euleriana: a -> b -> c -> d -> e -> b -> f
```

- Ejemplar que NO contenga una ruta hamiltoniana y que NO contenga una ruta euleriana.
  Una gráfica $G = (V, E)$ donde $V =\{ a, b, c, d, e, f \}$ y $E =\{ ab, bc, bd, be, bf \}$



Esta gráfica no tiene una ruta hamiltoniana ya que b es el único vértice que conecta un vértice con otro, por lo que al querer pasar por todos los vértices obligatoriamente repetiremos a b en la ruta, por lo que no es posible una ruta hamiltoniana.

Y tampoco es posible una ruta euleriana ya que todos los vértices tienen grado impar.

La cadena que codifica este ejemplar se encuentra en el archivo "Ejemplar4.txt", el cual contiene lo siguiente:

a b
b c
b d
b e
b f

Al pasar el ejemplar 4 en el transformador obtenemos lo siguiente en terminal:

```
→  Practica01 python3 transformador.py Ejemplar4.txt 4Solucion.txt
Lista de vértices: ['a', 'b', 'c', 'd', 'e', 'f']
Número de vértices: 6
Lista de aristas: [ a b, b c, b d, b e, b f ]
Matriz de adyacencia:
0 1 0 0 0 0
1 0 1 1 1 1
0 1 0 0 0 0
0 1 0 0 0 0
0 1 0 0 0 0
0 1 0 0 0 0
Matriz guardada correctamente en el archivo: 4Solucion.txt
```

El archivo que elegimos para regresar el resultado sí nos lo da en binario. En este caso, "4Solucion.txt" tiene lo siguiente:

01000010111101000001000001000001000

Al final, el Ejemplar 4 con el archivo "euler.py" nos regresa lo siguiente:

```
→  Practica01 python3 euler.py Ejemplar4.txt
Número de Vértices: 6
Número de Aristas: 5
Vértice(s) de mayor grado: ['b'] (grado 5)
¿La gráfica tiene una ruta euleriana? NO, no tiene una ruta Euleriana
```

Referencias

- West, Douglas B. Introduction to Graph Theory. Prentice Hall, 2ª edición, 2001. Recuperado el 8 de septiembre del 2024 de: `https://bayanbox.ir/view/134038771254298574/West-2nd-Edition-Solution-Manual.pdf` sección 1.2.35 y sección 4.2.21