Universidad Nacional Autónoma de México
Facultad de Ciencias

Complejidad Computacional

Práctica 2

Bonilla Reyes Dafne - 319089660
García Ponce José Camilo - 319210536
Juárez Ubaldo Juan Aurelio - 421095568
Moreno Castro Fernanda - 319128727

Algoritmos de Verificación

1. Ruta Hamiltoniana

Este problema pertenece a *NP*, ya que no se conoce un algoritmo en tiempo polinomial que lo resuelva.

Para esto diremos que el certificado u serán una permutan de los los vértices de G (sin repetición), donde la permutación representará el orden en el aparecen los vértices en una ruta (que sería Hamiltoniana, ya que pasa por todos los vértices).

Notemos que el certificado tendra n vértices, ya que n es el número de vértices en G y el certificado es una permutación sin repetición de estos, por lo que u tendrá una longitud de $O(n)$ y, por lo tanto, será un certificado válido (ya que $O(n) \in O(n^2)$ y eso es polinomial en relación con la entrada original, es decir es polinomial en relación con G, ya que la entrada que represente a G debe contener todos sus vértices, ya sea explícitamente u obtenerlos con base en las aristas).

Ahora, veamos una Máquina de Turing que reciba la gráfica G y el certificado u y revise si la ruta inducida por el certificado si existe en la gráfica G (ya que la ruta ya sabemos que es Hamiltoniana), para esto usaremos un algoritmo en el modelo RAM (recordemos que dependiendo de la representación de las gráficas usada, este algoritmo puede cambiar algo, pero no mucho, la idea general se mantiene y la complejidad seguirá siendo polinomial).

- `Input:` Una gráfica G, para este caso diremos que la gráfica se va a dar como la matriz de adyacencias, y certificado u con una permutacion de los vértices de la gráfica, para esto usaremos un arreglo de tamaño n, donde en la posición i esta el número del vértice en la posición i de la permutacion.
- `Output:` 1 si la ruta dada por el certificado es válida, 0 en otro caso.

1. Iteramos una variable j de 0 a $n-1$ (donde n es el numero de vértices en la gráfica) y para cada vértice j haremos lo siguiente:
   - Obtenemos los valores $[j]$ y $[j+1]$ del arreglo que representan al certificado, digamos que son a y b respectivamente.
   - Luego obtenemos el valor en la posición $[a][b]$ de la matriz de adyacente, digamos que es c.
   - SI c es 1 entonces seguimos iterando sobre las js, y en otro caso (c es 0), regresamos 0 y termina el algoritmo.
2. Luego de iterar sobre todas las js (los elementos de 0 a $n-1$), regresamos un 1 y termina el algoritmo.

Complejidad del algoritmo:

- **Paso 1:** Toma $O(n)$ (con n la cantidad de vértices en G), ya que iteramos la variable j de 0 a $n-1$ y ademas en cada iteración revisamos las posición en arreglos y matrices, lo cual sabemos que toma $O(1)$, entonces cada iteración nos toma $O(1)$, entonces todo el paso 1 nos toma $O(n)$.
- **Paso 2:** Como solo es el regreso de un valor, nos toma $O(1)$.

Por lo tanto, todo el algoritmo nos tomó $O(n)$, lo cual es una complejidad polinomial, ya que la complejidad es un polinomio.

Por último, usando la tesis de Church-Turing (*complexity-theoretic Church–Turing thesis*), sabemos que existe una Máquina de Turing determinista que resuelve/computa lo mismo que este algoritmo en a lo más una diferencia de tiempo polinomial, por lo tanto, podemos concluir que la Máquina de Turing tendrá tiempo polinomial, entonces concluimos que el problema Ruta Hamiltoniana pertenece a *NP* usando la definición alternativa de *NP*, ya que dimos un certificado de a los más longitud polinomial y una Máquina de Turing determinista que sirve para verificar en tiempo polinomial.

2. Descripción del certificado y esquema de codificación a utilizar

El certificado sera una permutación sin repetición de los vértices de la gráfica (suponemos que los vértices de la gráfica son números del 0 al $n-1$, con n el numero de vértices en la gráfica), de esta manera el certificado representará el orden de los vértices en una ruta (que es Hamiltoniana, ya que pasa por todos los vértices).

Codificación

Usará el alfabeto $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, , \}$

Como el certificado es una permutacion de los vértices, lo podemos ver como una lista de vértices (sin repetición), donde cada certificado representara el orden de los vértices de la gráfica en una ruta (la cual es Hamiltoniana, ya que pasa por todos los vértices), pero notemos que la ruta generada por el certificado puede no existir en la gráfica.

Por lo tanto, la representación sera una cadena de números (son los números naturales de 0 a n), y entre cada dos números son separados por una coma.

- **Ejemplo:**

  La gráfica con los vértices 1, 2, 3 y 4 puede tener el certificado $[2, 4, 1, 3]$ se representará como: "$2, 4, 1, 3$".

3. Descripción del algoritmo de generación aleatoria de certificados

El algoritmo de generación aleatoria de certificados representa una ruta Hamiltoniana en una gráfica utilizando su matriz de adyacencia, así que veamos como funciona este algoritmo:

1. **Entrada**: Se recibe una matriz de adyacencia que representa la gráfica. Esta matriz es una lista de listas donde el valor 1 indica la existencia de una arista entre los vértices y 0 indica su ausencia.
2. **Inicialización**: Se crea una lista que contiene todos los vértices de la gráfica (por ejemplo, si hay 4 vértices, la lista será $[0, 1, 2, 3]$).
3. **Generación de Permutación**:

- Mientras haya vértices en la lista:
  - Se selecciona un índice aleatorio dentro del rango de la longitud de la lista.
  - Se extrae el vértice en ese índice y se elimina de la lista.
  - Se agrega el vértice extraído a una nueva lista que representa el certificado.

4. **Salida**: El certificado, que es una lista de vértices representando una permutación de todos los vértices, se devuelve como resultado.

Complejidad

Ahora, análicemos la complejidad del algoritmo en dos partes:

1. **Creación de la lista de vértices**:

   - Esto tiene una complejidad de $O(n)$, donde n es el número de vértices.

     La creación de la lista de vértices es $O(n)$ porque se está generando una lista que contiene todos los vértices de la gráfica, donde n es el número de vértices. Veamoslo a detalle:

     a) **Inicialización de la lista**:
     - Se inicia con una lista vacía.
     - Se itera desde 0 hasta $n-1$ (donde n es el número total de vértices) y se agrega cada vértice a la lista.

     b) **Operación de añadir a la lista**:
     - La operación de añadir un elemento a la lista (en este caso, un vértice) se realiza en tiempo constante $O(1)$.
     - Como estamos añadiendo n elementos, el tiempo total para completar esta operación es $n \cdot O(1) = O(n)$.

     Por lo tanto, la complejidad para crear la lista de vértices es $O(n)$.

2. **Generación de la permutación**:

   - En cada iteración del bucle `while`, se selecciona un índice aleatorio (que se considera $O(\log n)$ debido a la generación del número aleatorio) y se extrae el vértice (que es $O(n)$ porque el `pop` se hace en el medio de la lista). Sin embargo, a medida que se extraen vértices, el tamaño de la lista se reduce.
   - En total, el bucle se ejecuta n veces. Por lo tanto, aunque cada extracción tiene un costo variable, la suma de las complejidades por todas las extracciones resulta en $O(n^2)$.

Por lo tanto, la complejidad del algoritmo para la generación aleatoria de certificados es $O(n^2)$.

Este ejercicio (el de generar un certificado a partir de una gráfica, el ejercicio 2) se implemento mediante el uso de los métodos `obtener_matriz_representacion2` y `generar_certificado`, por lo tanto sabemos que sus complejidades son de (los análisis de complejidad se encuentran al final de este documento) son $O(l)$ y $O(n^2)$ respectivamente, con l la longitud de la cadena que representa a la gráfica en representación 2 y n la cantidad de vértices en la gráfica, por lo tanto todo toma $O(n^2 + l)$.

4. Descripción y pseudocódigo del algoritmo de verificación

Para esto usaremos las representación 2 de la práctica 1:

Codificación

Usará el alfabeto $\{0, 1\}$

Esta representación usa la matriz de adyacencia de la gráfica (nótese que las gráficas no tienen lazos), si existe una arista entre los vértices i y j se representara con un 1 y si no con un 0 en la posición i, j de la matriz.

La representación sera una cadena de ceros y unos, donde solo se pondrán los valores de la matriz de adyacencia (y distinguimos las filas de la matriz de adyacencia usando la raíz de la longitud total de la cadena que codifica la matriz).

- **Ejemplo:**

  La gráfica con los vértices 1, 2 y 3 y las aristas $1 - 2$ y $1 - 3$ se representara como: "011100100".

Descripción del algoritmo

El algoritmo verificador debe decirnos dada la representación de la gráfica, la cual es una matriz de adyacencia, y un certificado, si el certificado es una solución valida a nuestro problema, en este caso, el de ruta Hamiltoneana, por lo que iniciamos declarando una lista donde guardaremos nuestras aristas. Lo cual toma $O(1)$.

Luego recorremos desde 0 hasta `len(certificado)-1` el certificado añadiendo las aristas, ya que el certificado tiene vértices, creamos las aristas mediante: (certificado[i], certificado[i+1]) que seria la arista que va de certificado[i] hasta certificado[i+1], es por ese $i + 1$ que solo recorremos hasta la penúltima posición del certificado ya que si tomamos la ultima $i + 1$ va a ser indice invalido.

Posteriormente debemos verificar que nuestras aristas creadas a partir del certificado existan en la gráfica, por lo que para cada tupla de aristas verificamos en la gráfica que exista, en el caso de que no exista regresamos un False ya que ese certificado no es valida. Todo esto nos llevo $O(n)$ debido a que las aristas son creadas a partir de los vértices del certificado donde n es la longitud del certificado que coincide con el numero de vértices en la gráfica. Por lo que todo el algoritmo verificador nos toma $O(n)$.

Una vez que el algoritmo paso todas estas verificaciones entonces podemos decir que el certificado es una solución valida para el problema de ruta Hamiltoneana y regresamos `True`.

Aclaración, suponemos que el certificado que recibe este algoritmo es uno generado con el algoritmo de generación aleatoria de certificados, es decir el certificado si es una permutacion sin repeticiones de los vértices de la gráfica, en caso de que no fuera uno generado por nuestro algoritmo, se tendría que verificar que cada vértice solo aparezca una vez y que si sea un vértice real.

Una versión de pseudocódigo sería la siguiente (la complejidad se sigue manteniendo).

Algoritmo de verificación

- `Input:` Una gráfica G, para este caso diremos que la gráfica se va a dar como la matriz de adyacencias y certificado u con una permutacion de los vértices de la gráfica, para esto usaremos un arreglo (o lista) de tamaño n (con n el numero de vértices de la gráfica), donde en la posición i esta el número del vértice en la posición i de la permutacion.

- `Output:` True si la ruta generada por el certificado si existen en G, False en otro caso.

Pasos del algoritmo

1. Creamos una lista L

2. Se obtiene la longitud del certificado, digamos que es l

3. Se itera una variable i desde 0 hasta $l - 1$
   a) Se obtiene el elemento en la posición i del certificado, digamos que es a
   b) Se obtiene el elemento en la posición $i + 1$ del certificado, digamos que es b
   c) Se agrega la tupla (a, b) a la lista L
4. Se itera sobre cada tupla de la lista L, para cada tupla x se hace lo siguiente
   a) Se obtiene el primer elemento de la tupla x, digamos que es c
   b) Se obtiene el segundo elemento de la tupla x, digamos que es d
   c) Se obtiene el elemento en la posición $[c][d]$ de la matriz de adyacencia, digamos que es e
   d) Si e es 0 se regresa False
5. Se regresa True

Este ejercicio (verificar dado un certificado y una gráfica, el ejercicio 3) se implemento mediante el uso de los métodos:

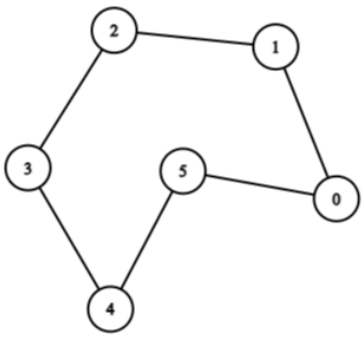
- `obtener_matriz_representacion2`
- `obtener_certificado`
- `obtener_numero_vertices_aristas_representacion2`
- `obtener_primer_ultimo_vertice`
- `verificar_certificado`

Por lo tanto sabemos que sus complejidades son de (los análisis de complejidad se encuentran al final de este documento) son $O(l)$, $O(n^2 + l)$, $O(n^2)$, $O(1)$ y $O(n)$ respectivamente, con l la longitud de la cadena que representa a la gráfica en representación 2 y n la cantidad de vértices en la gráfica, por lo tanto todo toma $O(n^2 + l)$.

5. Resumen de las pruebas ejecutadas, indicando el ejemplar y certificado usado, así como capturas de pantalla que muestren los resultados de la ejecución

1. **Ejemplar 1**

   Ubicado en el archivo `Ejemplar1.txt`, se usa representación 2 (de la práctica 1)



Representación 2: "01000110100001010000101000010110001O"

Ejecución ejercicio 2:

```
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar1/Ejemplar1.txt -o Ejemplar1/Certificado1-1.txt
Certificado guardado en Ejemplar1/Certificado1-1.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar1/Ejemplar1.txt -o Ejemplar1/Certificado1-2.txt
Certificado guardado en Ejemplar1/Certificado1-2.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar1/Ejemplar1.txt -o Ejemplar1/Certificado1-3.txt
Certificado guardado en Ejemplar1/Certificado1-3.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar1/Ejemplar1.txt -o Ejemplar1/Certificado1-4.txt
Certificado guardado en Ejemplar1/Certificado1-4.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar1/Ejemplar1.txt -o Ejemplar1/Certificado1-5.txt
Certificado guardado en Ejemplar1/Certificado1-5.txt
camilo@wowi:~/pyint/com$
```

Los certificados generadores se encuentran en los archivos: `Certificado1-1.txt`, `Certificado1-2.txt`, `Certificado1-3.txt`, `Certificado1-4.txt` y `Certificado1-5.txt`

Ejecuciones del ejercicio 3:

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Ejemplar1.txt
01000110100001010000101000010110001O
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Certificado1-1.txt
1,3,2,0,4,5
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar1\Ejemplar1.txt -c Ejemplar1\Certificado1-1.txt
Numero de vertices: 6
Numero de aristas: 6
Primer vertice de la ruta inducida: 1
Ultimo vertice de la ruta inducida: 5
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Ejemplar1.txt
01000110100001010000101000010110001O
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Certificado1-2.txt
3,4,2,5,1,0
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar1\Ejemplar1.txt -c Ejemplar1\Certificado1-2.txt
Numero de vertices: 6
Numero de aristas: 6
Primer vertice de la ruta inducida: 3
Ultimo vertice de la ruta inducida: 0
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Ejemplar1.txt
01000110100001010000101000010110001O
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Certificado1-3.txt
2,0,5,3,4,1
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar1\Ejemplar1.txt -c Ejemplar1\Certificado1-3.txt
Numero de vertices: 6
Numero de aristas: 6
Primer vertice de la ruta inducida: 2
Ultimo vertice de la ruta inducida: 1
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

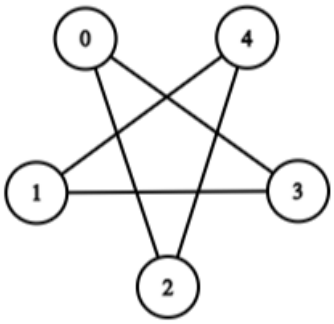
```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Ejemplar1.txt
0100011010000101000010100000101100010
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Certificado1-4.txt
3,0,5,1,2,4
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar1\Ejemplar1.txt -c Ejemplar1\Certificado1-4.txt
Numero de vertices: 6
Numero de aristas: 6
Primer vertice de la ruta inducida: 3
Ultimo vertice de la ruta inducida: 4
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Ejemplar1.txt
0100011010000101000010100000101100010
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar1\Certificado1-5.txt
1,0,5,4,3,2
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar1\Ejemplar1.txt -c Ejemplar1\Certificado1-5.txt
Numero de vertices: 6
Numero de aristas: 6
Primer vertice de la ruta inducida: 1
Ultimo vertice de la ruta inducida: 2
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  Si
```

El certificado que si cumple es: "$1, 0, 5, 4, 3, 2$"

2. **Ejemplar 2**

   Ubicado en el archivo `Ejemplar2.txt`, se usa representación 2 (de la practica 1)



Representación 2: "001100001110001110000110 0"

Ejecución ejercicio 2:

```
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar2/Ejemplar2.txt -o Ejemplar2/Certificado2-1.txt
Certificado guardado en Ejemplar2/Certificado2-1.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar2/Ejemplar2.txt -o Ejemplar2/Certificado2-2.txt
Certificado guardado en Ejemplar2/Certificado2-2.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar2/Ejemplar2.txt -o Ejemplar2/Certificado2-3.txt
Certificado guardado en Ejemplar2/Certificado2-3.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar2/Ejemplar2.txt -o Ejemplar2/Certificado2-4.txt
Certificado guardado en Ejemplar2/Certificado2-4.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar2/Ejemplar2.txt -o Ejemplar2/Certificado2-5.txt
Certificado guardado en Ejemplar2/Certificado2-5.txt
camilo@wowi:~/pyint/com$
```

Los certificados generadores se encuentran en los archivos: `Certificado2-1.txt`, `Certificado2-2.txt`, `Certificado2-3.txt`, `Certificado2-4.txt` y `Certificado2-5.txt`

Ejecución ejercicio 3:

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Ejemplar2.txt
00110000111000111000011100
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Certificado2-1.txt
0,4,2,3,1
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar2\Ejemplar2.txt -c Ejemplar2\Certificado2-1.txt
Numero de vertices: 5
Numero de aristas: 5
Primer vertice de la ruta inducida: 0
Ultimo vertice de la ruta inducida: 1
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Ejemplar2.txt
00110000111000111000011100
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Certificado2-2.txt
4,1,2,0,3
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar2\Ejemplar2.txt -c Ejemplar2\Certificado2-2.txt
Numero de vertices: 5
Numero de aristas: 5
Primer vertice de la ruta inducida: 4
Ultimo vertice de la ruta inducida: 3
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Ejemplar2.txt
00110000111000111000011100
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Certificado2-3.txt
1,3,0,2,4
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar2\Ejemplar2.txt -c Ejemplar2\Certificado2-3.txt
Numero de vertices: 5
Numero de aristas: 5
Primer vertice de la ruta inducida: 1
Ultimo vertice de la ruta inducida: 4
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  Si
```

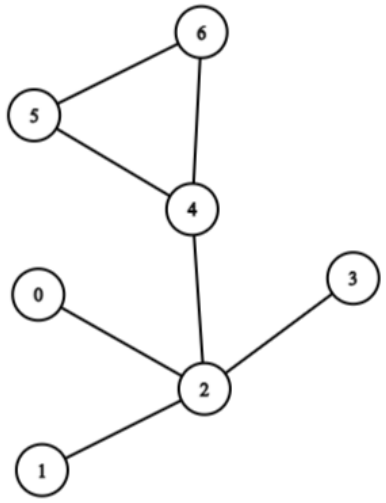
```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Ejemplar2.txt
00110000111000111000011100
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Certificado2-4.txt
0,4,1,3,2
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar2\Ejemplar2.txt -c Ejemplar2\Certificado2-4.txt
Numero de vertices: 5
Numero de aristas: 5
Primer vertice de la ruta inducida: 0
Ultimo vertice de la ruta inducida: 2
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Ejemplar2.txt
00110000111000111000011100
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar2\Certificado2-5.txt
1,4,0,2,3
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar2\Ejemplar2.txt -c Ejemplar2\Certificado2-5.txt
Numero de vertices: 5
Numero de aristas: 5
Primer vertice de la ruta inducida: 1
Ultimo vertice de la ruta inducida: 3
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

El certificado que si cumple es: "$1, 3, 0, 2, 4$"

3. **Ejemplar 3**

   Ubicado en el archivo `Ejemplar3.txt`, se usa representación 2 (de la practica 1)



Representación 2: "$00100000010000110110000100000010011000010100000110$"

Ejecución ejercicio 2:

```
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar3/Ejemplar3.txt -o Ejemplar3/Certificado3-1.txt
Certificado guardado en Ejemplar3/Certificado3-1.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar3/Ejemplar3.txt -o Ejemplar3/Certificado3-2.txt
Certificado guardado en Ejemplar3/Certificado3-2.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar3/Ejemplar3.txt -o Ejemplar3/Certificado3-3.txt
Certificado guardado en Ejemplar3/Certificado3-3.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar3/Ejemplar3.txt -o Ejemplar3/Certificado3-4.txt
Certificado guardado en Ejemplar3/Certificado3-4.txt
camilo@wowi:~/pyint/com$ python3 practica2.py ej2 -i Ejemplar3/Ejemplar3.txt -o Ejemplar3/Certificado3-5.txt
Certificado guardado en Ejemplar3/Certificado3-5.txt
camilo@wowi:~/pyint/com$
```

Los certificados generadores se encuentran en los archivos: `Certificado3-1.txt`, `Certificado3-2.txt`, `Certificado3-3.txt`, `Certificado3-4.txt` y `Certificado3-5.txt`

Ejecución ejercicio 3:

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Ejemplar3.txt
00100000010000110110000100000010011000010100000110
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Certificado3-1.txt
5,3,4,1,6,0,2
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar3\Ejemplar3.txt -c Ejemplar3\Certificado3-1.txt
Numero de vertices: 7
Numero de aristas: 7
Primer vertice de la ruta inducida: 5
Ultimo vertice de la ruta inducida: 2
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Ejemplar3.txt
00100000010000110110000100000010011000010100000110
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Certificado3-2.txt
6,2,1,5,0,3,4
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar3\Ejemplar3.txt -c Ejemplar3\Certificado3-2.txt
Numero de vertices: 7
Numero de aristas: 7
Primer vertice de la ruta inducida: 6
Ultimo vertice de la ruta inducida: 4
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Ejemplar3.txt
00100000010000110110000100000010011000010100000110
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Certificado3-2.txt
6,2,1,5,0,3,4
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar3\Ejemplar3.txt -c Ejemplar3\Certificado3-3.txt
Numero de vertices: 7
Numero de aristas: 7
Primer vertice de la ruta inducida: 3
Ultimo vertice de la ruta inducida: 6
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Ejemplar3.txt
00100000010000110110000100000010011000010100000110
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Certificado3-4.txt
6,4,0,3,5,1,2
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar3\Ejemplar3.txt -c Ejemplar3\Certificado3-4.txt
Numero de vertices: 7
Numero de aristas: 7
Primer vertice de la ruta inducida: 6
Ultimo vertice de la ruta inducida: 2
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

```
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Ejemplar3.txt
00100000010000110110000100000010011000010100000110
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ cat Ejemplar3\Certificado3-5.txt
3,5,0,4,1,2,6
C:\Users\Administrador\Desktop\semestre\complejidad\Complejidad-Computacional\Prácticas\Práctica_02 (master -> origin)
λ python practica2.py ej3 -g Ejemplar3\Ejemplar3.txt -c Ejemplar3\Certificado3-5.txt
Numero de vertices: 7
Numero de aristas: 7
Primer vertice de la ruta inducida: 3
Ultimo vertice de la ruta inducida: 6
¿El ejemplar, con el certificado dado, satisface la condicion de pertenencia al lenguaje correspondiente?  No
```

El certificado que si cumple es: para esta gráfica no existe un certificada que cumpla, ya que la gráfica no tiene rutas Hamiltonianas

6. Funciones del código

Para el análisis de complejidad de estos métodos se omite la complejidad de concatenar cadenas (esta complejidad es algo complicada), convertir un numero en una cadena y de convertir una cadena en un numero, esto debido a que solo hace más complicado el análisis, pero estas complejidades no afectan mucho las demás complejidades, ya que estas operaciones tiene complejidad $O(logN)$ (para convertir el numero N a una cadena) y $O(L)$ (para convertir una cadena de longitud L a un numero).

1. `obtener_matriz_representacion2` (Método de la practica pasada)
   Este método se encarga de transformar una cadena de la representación 2 a una matriz de adyacencia
   Esto lo hace primero revisando que el texto si tenga caracteres, en $O(1)$
   Luego se verifica que la longitud del texto sea el cuadrado de algún entero, lo cual toma $O(1)$
   Después se revisa que todo la cadena solo tenga ceros o unos, toma $O(l)$ donde l es la longitud de la cadena, ya que se recorre toda la cadena
   Posteriormente se crea una matriz de g por g donde g es la raíz de la longitud de la cadena, entonces se recorre la matriz poniendo en cada posición de la matriz el valor en la posición del texto que le corresponde, esto nos toma $O(l)$, ya que se recorre toda la cadena de entrada
   Al final se regresa la matriz de adyacencia y todo toma $O(l)$

2. `obtener_numero_vertices_aristas_representacion2` (Método de la practica pasada)
   Este método se encarga de devolver el numero de vértices y aristas dada la matriz de adyacencia
   Esto lo hace revisando la longitud de la matriz para obtener el numero de vértices, toma $O(1)$
   Luego se recorre toda la matriz, sumando todos los valores uno que se encuentran para devolver el numero de aristas, toma $O(n^2)$ donde n es el numero de vértices de la gráfica, ya que se recorre toda la matriz
   Al final se regresa el numero de vértices y aristas y todo toma $O(n^2)$

3. `generar_certificado`
   Este método se encarga generar un certificado aleatorio dada una gráfica
   Esto lo hace creando una lista con los números de 1 a n, donde n es el número de vértices de la gráfica, toma $O(n)$
   Después se crea otro lista (el certificado)
   Luego se obtiene la longitud de la primera lista, digamos que es le (en $O(1)$) y se itera de le a 1 (se realizan $O(n)$ iteraciones)
   Dentro de la iteración se genera un numero aleatorio i que esta entre 0 y $le-1$ (en $O(le-1)$, pero como $le-1$ al inicio es $n-1$, entonces es $O(n)$), luego se saca el i-esimo elemento de la primera lista (toma $O(n)$) y este elemento se agrega a la segunda lista (en $O(1)$)
   Al final se regresa la segunda lista y todo toma $O(n^2)$

4. `guardar_certificado`
   Este método se encarga de guardar el certificado en un archivo, usando la codificación del certificado
   Para esto se itera sobre el certificado (que como tiene n elemento, donde n es el número de vértices de la gráfica), entonces se itera en $O(n)$
   Y por cada elemento (son números del 1 al $n-1$) del certificado se convierte en una cadena y se le agrega una coma y se uno todo a la cadena que se lleva hasta el momento
   Por ultimo se quita la ultima coma de la cadena generada y se guarda en un archivo la cadena, todo tomando $O(n)$ (o $O(n^2)$ si decimos que transformar el elemento a cadena nos toma $O(n)$)

5. `obtener_primer_ultimo_vertice`
   Este método se encarga de obtener el primer y ultimo vértice de la ruta generada por el certificado
   Para esto solo se revisa el primer y ultimo elemento del certificado, esto en $O(1)$
   Por ultimo se regresan los valores obtenidos, todo en $O(1)$

6. `obtener_certificado`
   Este método se encarga de transformar una cadena de la codificación del certificado a un certificado
   Primero se divide la cadena (usando la coma como separador), para así obtener los elementos (vértices) y su orden del certificado, esto en $O(l)$, donde l es la longitud de la cadena que representa al certificado
   Luego se itera sobre los elementos obtenidos de dividir la cadena y a cada uno se convierte en un número, esto en $O(n)$, donde n es la cantidad de vértices en la gráfica
   Por ultimo se regresa lo obtenido y todo toma $O(n+l)$ (o $O(n^2+l)$ si decimos que transformar una cadena a un número nos toma $O(n)$)

7. `verificar_certificado`
   Este método se encarga de verificar si la ruta generada por el certificado si es una ruta existente en una gráfica
   Primero se crea una lista y se itera sobre el certificado agregando las aristas (tuplas de números) de la ruta inducida por el certificado, esto se hace solo iterando j de 0 a $n-1$ y agregando las posiciones j y $j+1$ en una tupla que se agrega a la lista, iterar sobre el certificado toma $O(n)$, donde n es el numero de vértices en la gráfica, y agregar las tuplas a la lista es $O(1)$
   Después de iteran sobre las aristas (tuplas) obtenidas y usando los números que tienen las aristas, digamos x y y, se revisa si en la matriz de adyacencias de la gráfica la posición x y y tiene un 1 (que existe una arista entre los vértices x y y), si tiene un 1 se sigue iterando sobre las aristas, pero si tiene un 0 se regresa False, todo esto nos toma $O(n)$ en iterar las aristas (ya que la ruta tiene $n-1$ aristas) y en revisar la matriz es $O(1)$
   Por ultimo se regresa True y todo nos toma $O(n)$

Referencias

[1] TimeComplexity - Python Wiki. (2017). Python.org. https://wiki.python.org/moin/TimeComplexity

[2] m81. (2015, April 5). What is Big-O runtime of the standard random number generator in python? (Worst-case). Stack Overflow. https://stackoverflow.com/questions/29461787/what-is-big-o-runtime-of-the-standard-random-number-generator-in-python-worst

[3] Imran. (2024). Cost of len() function. Stack Overflow. https://stackoverflow.com/questions/1115313/cost-of-len-function