Universidad Nacional Autónoma de México
Facultad de Ciencias

Complejidad Computacional

Práctica 1

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

Esquemas de Codificación

Descripcion de Problemas

- Descripción del problema de decisión
  RUTA HAMILTONIANA (Decisión)
  EJEMPLAR: Una gráfica $G = (V, E)$.
  PREGUNTA: ¿Existe una ruta hamiltoniana en G? Es decir, ¿existe una ruta simple que pase por cada vértice de G exactamente una vez?

- Descripción del problema del problema de optimización
  RUTA HAMILTONIANA (Optimización)
  En este caso, no tendría mucho sentido una pregunta de optimización, ya que todas las rutas hamiltonianas de una gráfica, tienen el mismo número de aristas. Tendria sentido con una gráfica ponderada, donde podríamos definirlo como encontrar la ruta con menor peso, pero en una gráfica sin pesos en las aristas, se reduciría a encontrar cualquier ruta hamiltoniana.

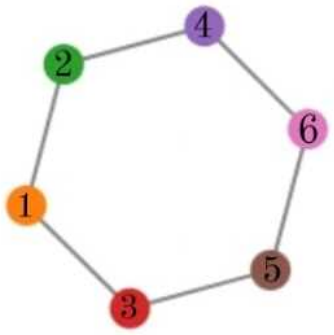
Esquemas

- Esquema de codificación 1 Usara el alfabeto $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (,), \backslash n, -, ,\}$, con $\backslash n$ el salto de linea
  Para la codificación vamos a solicitar que la entrada sea de la forma:
  n
  (r-v),(u-v),(u-t),...
  Donde, $n \geq 1$ y donde r,v,u,t $\in V$, lo cual quiere decir que el número que los denota tiene que ser menor a n, escrito en tuplas delimitadas por paréntesis, cada tupla se separa por un guión.
  Es decir, primero esta un numero entero que representa el numero de vértices de la gráfica
  Después hay un salto de linea para separar al numero y las aristas
  Luego hay un serie de pares separados por , que serán las aristas
  Los pares son de la forma de dos números enteros separados por $-$, los números representaran los vértices que forman la arista, y los pares de números están dentro de paréntesis
  Esta representación usa lista de aristas
  Ejemplo
  La gráfica con los vértices 1, 2 y 3 y las aristas $1-2$ y $1-3$ se representara como: "$3\backslash n(1-2),(1-3)$"

- Esquema de codificación 2 Usara el alfabeto $\{0, 1\}$
  Esta representación usa la matriz de adyacencia de la gráfica, 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
  Ejemplo
  La gráfica con los vértices 1, 2 y 3 y las aristas $1-2$ y $1-3$ se representara como: "011100100"

- Ejemplares

  1. **Ejemplar con una ruta hamiltoniana y una ruta euleriana** Ubicado en el archivo `ejemplar1.txt` para representación 1 y `ejemplar1v2.txt` para representación 2



$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$E = \{v_1v_2, v_1v_3, v_2v_4, v_3v_5, v_4v_6, v_5v_6\}$$

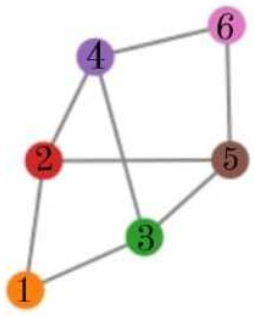
Camino euleriano: $1 \to 3 \to 5 \to 6 \to 4 \to 2 \to 1$
Camino hamiltoniano: $1 \to 3 \to 5 \to 6 \to 4 \to 2$
Representación 1: "$6\backslash n(1-2), (1-3), (2-4), (3-5), (4-6), (5-6)$"
Representación 2: "011000100100100010010001001001000110"
Ejecución ejercicio 3:

```
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$ python practica.py ej3 -i archivos/ejemplar1.txt
-o archivos/ejemplar1v2.txt
Numero de vertices: 6
Numero de aristas: 6
Representacion 2 guardada en archivos/ejemplar1v2.txt
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$
```

Ejecución ejercicio 4:

```
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$ python practica.py ej4 -i archivos/ejemplar1v2.tx
t
Numero de vertices: 6
Numero de aristas: 6
Vertice con mayor grado: 1
Camino euleriano: SI
Camino:
1 -> 3 -> 5 -> 6 -> 4 -> 2 -> 1
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$
```

2. **Ejemplar con una ruta hamiltoniana y que no contenga una ruta euleriana** Ubicado en el archivo `ejemplar2.txt` para representación 1 y `ejemplar2v2.txt` para representación 2



$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$E = \{v_1v_2, v_1v_3, v_2v_4, v_2v_5, v_3v_5, v_3v_4, v_4v_6, v_5v_6\}$$

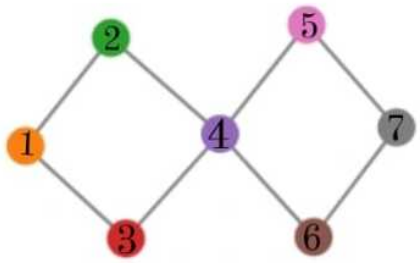
Camino euleriano: No, tiene 4 vértices de grado impar
Camino hamiltoniano: $1 \to 2 \to 5 \to 6 \to 4 \to 3$
Representación 1: "$6\backslash n(1-2),(1-3),(2-4),(2-5),(3-5),(3-4),(4-6),(5-6)$"
Representación 2: "011000100110100110011001011001000110"
Ejecución ejercicio 3:

```
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$ python practica.py ej3 -i archivos/ejemplar2.txt
-o archivos/ejemplar2v2.txt
Numero de vertices: 6
Numero de aristas: 8
Representacion 2 guardada en archivos/ejemplar2v2.txt
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$
```

Ejecución ejercicio 4:

```
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$ python practica.py ej4 -i archivos/ejemplar2v2.tx
t
Numero de vertices: 6
Numero de aristas: 8
Vertice con mayor grado: 2
Camino euleriano: NO
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$
```

3. **Ejemplar que NO contenga una ruta hamiltoniana y que contenga una ruta euleriana** Ubicado en el archivo `ejemplar3.txt` para representación 1 y `ejemplar3v2.txt` para representación 2



$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$$

$$E = \{v_1v_2, v_1v_3, v_2v_4, v_3v_4, v_4v_5, v_4v_6, v_5v_7, v_6v_7\}$$

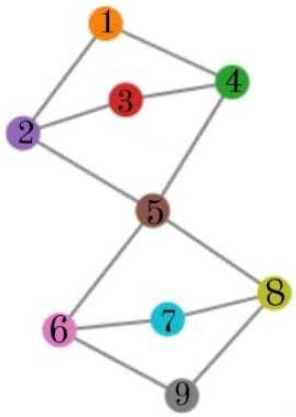
Camino euleriano: $1 \to 3 \to 4 \to 6 \to 7 \to 5 \to 4 \to 2 \to 1$
Camino hamiltoniano: No, después de búsquedas no se encontró alguno
Representación 1: "$7\backslash n(1-2),(1-3),(2-4),(3-4),(4-5),(4-6),(5-7),(6-7)$"
Representación 2: "01100001001000100100001101100001001000010010000110"
Ejecución ejercicio 3:

```
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$ python practica.py ej3 -i archivos/ejemplar3.txt
-o archivos/ejemplar3v2.txt
Numero de vertices: 7
Numero de aristas: 8
Representacion 2 guardada en archivos/ejemplar3v2.txt
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$
```

Ejecución ejercicio 4:

```
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$ python practica.py ej4 -i archivos/ejemplar3v2.tx
t
Numero de vertices: 7
Numero de aristas: 8
Vertice con mayor grado: 4
Camino euleriano: SI
Camino:
1 -> 3 -> 4 -> 6 -> 7 -> 5 -> 4 -> 2 -> 1
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$
```

4. **Ejemplar que NO contenga una ruta hamiltoniana y que NO contenga una ruta euleriana** Ubicado en el archivo `ejemplar4.txt` para representación 1 y `ejemplar4v2.txt` para representación 2



$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$$

$$E = \{v_1v_2, v_1v_4, v_2v_3, v_2v_5, v_3v_4, v_4v_5, v_5v_6, v_5v_8, v_6v_7, v_6v_9, v_7v_8, v_8v_9\}$$

Camino euleriano: No, tiene 4 vértices de grado impar
Camino hamiltoniano: No, después de búsquedas no se encontró alguno
Representación 1: "9\n(1 − 2), (1 − 4), (2 − 3), (2 − 5), (3 − 4), (4 − 5), (5 − 6), (5 − 8), (6 − 7), (6 − 9), (7 − 8), (8 − 9)"
Representación 2:
"010100000101010000010100000101010000010101010000010101000001010000010101000001010"
Ejecución ejercicio 3:

```
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$ python practica.py ej3 -i archivos/ejemplar4.txt
-o archivos/ejemplar4v2.txt
Numero de vertices: 9
Numero de aristas: 12
Representacion 2 guardada en archivos/ejemplar4v2.txt
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$
```

Ejecución ejercicio 4:

```
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$ python practica.py ej4 -i archivos/ejemplar4v2.tx
t
Numero de vertices: 9
Numero de aristas: 12
Vertice con mayor grado: 5
Camino euleriano: NO
camilo@wowi:~/CC/septimoSemestre/Com/Complejidad-Computacional/Practicas/Practica01$
```

Algoritmos

1. `obtener_listas_representacion1` Este método se encarga de transformar una cadena de la representación 1 a una lista de vértices y aristas
   Esto lo hace primero revisando que el texto si tenga caracteres, en $O(1)$
   Luego se parte el texto (usando el salto de linea como separador), lo cual nos toma $O(n)$ donde n es la longitud del texto, ya que se recorre todo el texto
   Después se revisa que sea generaron 1 o 2 partes, toma $O(1)$
   Posteriormente sea guarda en una lista la cantidad de vértices de la gráfica, empezando en 1 a v (la cantidad de vértices), esto toma $O(v)$ ya que agregamos un elemento a la lista por cada vértice de la gráfica
   Luego se revisa si hay aristas o no, toma $O(1)$, si no tiene aristas terminamos
   Después a la parte de las aristas la dividimos (usando la coma como separador) generando las parejas que serán las aristas, toma $O(n)$, ya que se recorre todo el texto
   Posteriormente a las parejas se les quitan los paréntesis, toma $O(n)$, ya que se recorre todo el texto
   Luego iteramos sobre los pares para ir generando los números de las aristas, lo cual toma $O(log(v))$ debido a que los vértices en las aristas están escritos como números decimales y tenemos que separar dos números separados por un guión medio, luego las aristas se van guardando en la lista de aristas como pares de números y todo nos toma entonces $O(elog(v))$ con e el numero de aristas, ya que repetimos el proceso de separar los números para cada arista, o también podríamos decir que toma $O(v^2log(n))$ ya que sabemos que una gráfica tiene a lo más $\frac{n(n-1)}{2}$
   Después, se itera sobre todas las aristas para eliminar las aristas repetidas lo cual toma $O(e^2)$, ya que para cada aristas tenemos que revisar si ya esta
   Posteriormente revisamos que las aristas tengan vértices con valor numérico valido (menor o igual al numero de vértices), lo cual toma $O(ve)$, ya que revisamos ambas listas, o podríamos decir que toma $O(v^3)$
   Al final regresamos la lista de aristas y vertices, y todo nos tomo $O(n + v + elog(v) + e^2 + ve)$ o para simplificar las cosas toma $O(n + v^4)$

2. `obtener_numero_vertices_aristas` Este método se encarga de devolver el numero de vértices y aristas dada la lista de vértices y aristas
   Esto lo hace revisando la longitud de las lista, lo cual toma $O(1)$
   Al final se regresa el numero de vértices y aristas y todo toma $O(1)$

3. `obtener_matriz_adyacencia` Este método se encarga de transformar una lista de vértices y aristas en una matriz de adyacencia
   Esto lo hace creando una matriz de n por n, donde n es el tamaño de la lista de vértices o el numero de vértices, y se llena de ceros la matriz, toma $O(n^2)$, ya que se recorre toda la matriz
   Luego se iteran las aristas y se va marcando con un uno en la posición de la matriz que corresponde a la arista, lo cual toma $O(e)$ donde e es el numero de arista, o toma $O(n^2)$, ya que se recorren todas las aristas
   Al final se devuelve la matriz y todo toma $O(n^2)$

4. `obtener_representacion2` Este método se encarga de obtener la representación 2 usando una matriz de adyacencias
   Esto lo hace recorriendo toda la matriz y escribiendo el valor de cada entrada en una cadena, lo cual toma $O(n)$ donde n es el numero de vértices, ya que se recorre toda la matriz
   Al final se devuelve la cadena y todo toma $O(n)$

5. `obtener_matriz_representacion2` 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 $O(n)$ donde n 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 $O(n)$, ya que se recorre toda la matriz
   Al final se regresa la matriz de adyacencia y todo toma $O(n)$

6. `obtener_numero_vertices_aristas_representacion2` 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 $O(n)$ donde n es el numero de vértices, ya que se recorre toda la matriz
   Al final se regresa el numero de vértices y aristas y todo toma $O(n)$

7. `obtener_vertice_mayor_grado` Este método se encarga de devolver el vértice de mayor grado dada la matriz de adyacencia
   Esto lo recorriendo toda la matriz y sumando todos los valores de cada fila y así devolver el numero de la fila con mayor suma total, toma $O(n)$ donde n es el numero de vértices, ya que se recorre toda la matriz
   Al final se regresa el numero de vértices y aristas y todo toma $O(n)$

8. `obtener_vertice_mayor_grado` Este método se encarga de devolver el vértice de mayor grado dada la matriz de adyacencia
   Esto lo recorriendo toda la matriz y sumando todos los valores de cada fila y así devolver el numero de la fila con mayor suma total, toma $O(n)$ donde n es el numero de vértices, ya que se recorre toda la matriz
   Al final se regresa el numero de vértices y aristas y todo toma $O(n)$

9. `obtener_grado_vertice` Este método se encarga de obtener el grado de un vértice dada la matriz de adyacencia y el índice el vértice
   Esto lo sumando todos los valores de la fila del vértice, toma $O(n)$ donde n es el numero de vértices, ya que se recorre todas las entregas de la fila para ver si tiene aristas con algún vértice
   Al final se regresa el grado del vértice y todo toma $O(n)$

10. `dfs` Este método se encarga de hacer un recorrido DFS y guardar los vértices visitados
    Solo es el algoritmo de DFS en su versión recursiva, por lo cual sabemos que toma $O(v + e)$ donde v es el numero de vértices y v es el numero de aristas, o para simplificar las cosas toma $O(v^2)$ Al final se regresa actualiza la lista de vértices visitados y todo toma $O(v^2)$

11. `camino_euleriano_matriz` Este método se regresa falso y una lista vacía si la gráfica no tiene un camino euleriano, pero si tiene un camino entonces regresa verdadero y una lista con el camino, esto dada una matriz de adyacencias

Para esto primero revisamos que el numero de vértices de grado impar sea 2 o 0, esto lo hacemos obteniendo el grado de cada vértice, por lo tal nos toma $O(n^2)$ donde n es el numero de vértices, si el numero de vértices de grado impar no es 2 o 0 regresamos falso, ya que esta es una condición para tener caminos eulerianos
Luego revisamos que la gráfica inducida por los vértices con aristas sea conexa, para esto usamos DFS para ver que vértices podemos visitar, lo cual toma $O(n^2)$, luego revisamos que todos los vértices con grado mayor a 0 hayan sido visitados, para esto revisamos el grado de cada vértice y que este en la lista de vértices visitados por lo cual nos toma $O(n^2)$, si no todos fueron visitados entonces regresamos falso, ya que la gráfica no es conexa (otra condición para caminos eulerianos)
Después usamos un algoritmo para encontrar el camino euleriano, para esto usaremos una cola (usamos una lista para simularla)
Metemos a un vértice a la pila, si no hay vértices de grado impar metemos a cualquier vértice pero si hay vértices de grado impar metemos a uno de ellos, esto toma $O(1)$ ya que sabemos que hay 0 o 2 vértices de grado impar
Luego, mientras la pila no este vacía, repetimos esto, tomamos al vértice del tope y revisamos su grado, lo cual toma $O(n)$, si el grado es cero, entonces agregamos el vértice a la lista del camino y hacemos pop a la pila, lo cual toma $O(1)$, este caso nos toma $O(1)$, pero si el grado es mayor a cero, entonces buscamos una arista que tenga, esto toma $O(n)$ ya que revisamos cada vértice si tiene arista con el, al encontrar la arista la borramos la de la matriz, esto toma $O(1)$, y hacemos push al vértice del otro extremo de la arista, lo cual nos toma $O(1)$, este caso nos toma $O(n)$, y como vamos a repetir este proceso por cada arista de la gráfica nos va a tomar todo $O(ne)$ con e el numero de aristas o para simplificar $O(n^3)$
Al final regresamos verdadero y la lista de vértices y todo tomo $O(n^3)$

Referencias

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

[2] Finding the Eulerian path in O(M) - Algorithms for Competitive Programming. (n.d.). Cp-Algorithms.com. https://cp-algorithms.com/graph/euler_path.html

[3] GeeksforGeeks. (2018, March 27). Eulerian Path in undirected graph. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/eulerian-path-undirected-graph/

[4] Time and Space Complexity of DFS and BFS Algorithm. (2024, March 28). GeeksforGeeks. https://www.geeksforgeeks.org/time-and-space-complexity-of-dfs-and-bfs-algorithm/

[5] GeeksforGeeks. (2021, February). Java Program to Check Whether Undirected Graph is Connected Using DFS. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/java-program-to-check-whether-undirected-graph-is-connected-using-dfs/