José Alejandro Pérez Márquez 310109800
Tania Michelle Rubí Rojas 315121719
Erick Bernal Márquez 317042522

Práctica 2

Algoritmos de Verificación

- **Esquema de codifcación para el certificado**

  PARTICION EN CLANES

  EJEMPLAR: Una gráfica $G = (V, E)$ y un entero positivo $K \leq |V|$

  PREGUNTA: ¿Existe una partición de G en $k \leq K$ conjuntos disjuntos $V_1, V_2, ..., V_k$ tal que, $\forall i \quad 1 \leq i \leq k$, la subgráfica inducida por V_i es un clan?

  Recordemos nuestro anterior esquema de codificación para gráficas no dirigidas donde el primer renglón corresponde a k y el resto de la cadena corresponde a la matriz de adyacencias de la gráfica G.
  Por ejemplo. La siguiente cadena codifica la gráfica E con k = 3

  0 0 0 0 0 1 1
  0 1 0 0 0 0 0
  1 0 0 0 0 0 0
  0 0 0 1 0 0 0
  0 0 1 0 0 0 0
  0 0 0 0 0 0 1
  0 0 0 0 0 1 0

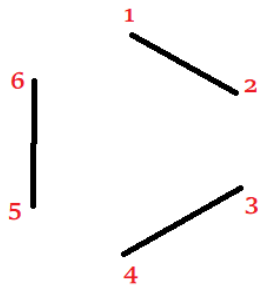

Figure 1: Gráfica E

Sim embargo, por razones de practicidad pondremos las entradas de la matriz como 1 si $i = j$, es decir, la diagonal. Dando a entender que un vértice está conectado así mismo, en el algoritmo explicaremos por qué.
La matriz de entrada sigue siendo la misma, es en el algoritmo donde cambiaremos estos valores.

Nuestro esquema de codificación para el certificado lo vamos a modelar como una lista de los subconjuntos de vértices de la gráfica G, cuya lista es de longitud k, estos vértices no estarán representados de forma binaria, sino por números. La siguiente codificación es un certificado para la Grafica E

$$[\{1,2\},\{3,4\},\{5,6\}]$$

En este caso el algoritmo de verificación tiene que responder con un *Sí*, pues en efecto dicha codificación corresponde a partir la gráfica en 3 clanes. Otro certificado puede ser

$$[\{5,2\},\{3\},\{1,6,4\}]$$

Donde el algoritmo responde con un rotundo *No*, ya que dichos subconjuntos de vértices no están conectados entre sí en la gráfica.

Cabe señalar que no nos importa el orden de los subconjuntos así como tampoco el orden de los elementos de los mismos, debido a que solo nos interesa la conexión entre ellos y no la relación de menor, mayor o tamaño de estos. Así los siguientes certificados son los mismos.

$$[\{5,2\},\{3\},\{1,6,4\}]$$
$$[\{2,5\},\{1,6,4\},\{3\}]$$
$$[\{1,4,6\},\{3\},\{2,5\}]$$

De igual manera para el caso donde responde sí

$$[\{1,2\},\{5,6\},\{4,3\}]$$

- **Algoritmo de generación aleatoria de certificados**

  Como nuestros certificados son una lista de conjuntos, entonces la idea inicial para empezar a programar este algoritmo fue ver estos conjuntos como particiones del conjunto de nuestros vértices, donde el valor de k es el número de particiones que tendrá nuestro certificado.

  El algoritmo que programamos para generar dicho certificado es:

  1. Al momento de crear las particiones hay que evitar crear un número mayor al valor de k, así que esto lo solucionamos dividiendo el número de vértices actuales entre k. Cuando realizamos esta división es posible que la división no sea exacta, entonces lo que hacemos es elegir de manera aleatoria si tomamos el piso o el techo de ese resultado. Así, obtenemos la longitud de nuestra partición en curso.
  2. Después de esto, seleccionamos un elemento dentro de nuestro conjunto de vértices actuales de manera aleatoria, lo agregamos a la partición en curso y eliminamos a dicho elemento de nuestras futuras posibles selecciones de vértices. Este paso lo realizamos hasta llenar de elementos nuestra partición en curso (nos guiamos de la longitud de la partición para saber cuando parar).

3. Como esa partición ya tiene todos los elementos que necesita, entonces lo agregamos a la lista `certificado`, que es la que contendrá todas las particiones necesarias.
4. Ahora tenemos dos posibles escenarios:
   i. La lista `certificado` ya tiene k particiones, así que terminamos y la guardamos en la carpeta `ejemplares/`.
   ii. La lista `certificado` aún no tiene k particiones, así que volvemos a empezar desde el punto 1. para seguir generando y llenando las particiones faltantes.

- **Algoritmo de verificación**

  La idea general del algoritmo es verificar que los vertices de los subconjuntos que hay la lista del certificado estén marcados con un 1 en la matriz de adyacencias. Verbi gratia si tenemos el siguiente subconjunto

  $$\{1, 2, 4\}$$

  entonces solo nos importa que la intersección de las columnas y renglones 1, 2 y 4 de la matriz contengan un 1.

  0 1 x 1
  1 0 x 1
  x x x x
  1 1 x 0

  No nos interesa comprobar la columna ni renglon 3 porque no queremos comprobar si es parte del clan.

  También podríamos tener subconjuntos de tamaño 1 el cual con clan triviales, y podríamos tener $k = |V|$ subconjuntos, donde V es el número de vértices de la gráfica, es decir si tenemos el certificado

  $$[\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}]$$

  Entonces al algoritmo debe responder con *Sí*. Para modelar esto y que sea consistente con la idea anterior pondremos las entradas de la diagonal como 1.

  1 x x x x x
  x 1 x x x x
  x x 1 x x x
  x x x 1 x x
  x x x x 1 x
  x x x x x 1

Las x's no representan nada en el esquema, solo que no nos interesa el valor para el algoritmo.

De esta manera al hacer la comprobación de las entradas, como todas son 1, entonces tendrá como respuesta *Sí*.

Así, lo primero que hay que hacer es modificar la digonal, para ello recorremos cada renglon y fila de esta.

```
for i in filas:
    for j in columnas:
        if i == j then
            matriz[i][j] = 1 #asignamos 1 en la diagonal
```

Dado que recorremos todas las filas n y por cada fila recorremos todas las columnas n, tenemos que en total lo hacemos $n \times n$ veces, es decir la complejidad es $O(n^2)$ donde n es el número de vértices.

Ahora recorremos cada subconjunto en el certificado y los modelamos como filas y columnas para solo comprobar esas entradas en la matriz, en el peor de los caso hay tantos subconjuntos como vértices $k = |V| = n$, por lo cual tenemos que es de tiempo lineal $O(n)$

```
for k in certificado:
    filas = k
    columnas = k
    ...
```

Por cada uno de estos subconjuntos debemos verificar que las entradas sean igual 1.
Suponemos en un principio que todos los subconjuntos del vértice están conectados entre sí, por lo que tendremos una variable *bandera* = *True* y en el momento que alguna entrada tenga 0 (es decir, dichos vértices no están conectados) podemos asegurar que no forman un clan, así nuestro algoritmo queda

```
for k in certificado:
    filas = k
    columnas = k
    for i in filas:
        for j in columnas:
            if matriz[i][j] == 0:
                bandera = False
```

Como vimos anteriormente recorrer la matriz nos toma tiempo cuadrático, pero tengamos en cuenta que la estamos recorriendo por cada elemento k en el certificado. Lo cual nos da un tiempo total cúbico $O(n^3)$.

Para calcular la complejidad total del algoritmo sumamos la complejidad de la modificación de la diagonal más el recorrido de la matriz por cada subconjunto del certificado. $O(n^2) + O(n^3) = O(n^3)$

- **Ejecución de los ejemplares**

  Para ello suponemos ya haber generado los certificados aleatorios como indica el *readme* siguiendo la nomenclatura *certificadoGN.txt*, donde G indica la gráfica y N el numero o letra para especificar el certificado de la gráfica correspondiente

  - **Ejemplar A**
    El primer ejemplar que tenemos para nuestro algoritmo es

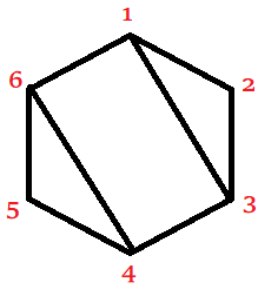

Figure 2: Gráfica A

Cuya codificación se compone de $k = 2$ y nuestra matriz de adyacencia, podemos encontrar la codificación en su archivo correspondiente *graficaA.txt*
Tenemos la siguientes ejecuciones con los certificados

1. **Certificado 1: El certificado: [{2, 4, 6}, {1, 3, 5}]**
   Entrada: `python main.py graficaA.txt certificadoA1.txt`
   Salida:
   El numero de vertices es 6
   El numero de aristas es 8
   El valor de K es 2

La representacion de nuestra matriz es:

0 1 1 0 0 1
1 0 1 0 0 0
1 1 0 1 0 0
0 0 1 0 1 1
0 0 0 1 0 1
1 0 0 1 1 0

El certificado: [{2, 4, 6}, {1, 3, 5}]
El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje

2. **Certificado 2: [{3,5,6},{1,2,4}]**
   Entrada: `python main.py graficaA.txt certificadoA2.txt`
   Salida:
   El numero de vertices es 6
   El numero de aristas es 8
   El valor de K es 2
   La representacion de nuestra matriz es:

   0 1 1 0 0 1
   1 0 1 0 0 0
   1 1 0 1 0 0
   0 0 1 0 1 1
   0 0 0 1 0 1
   1 0 0 1 1 0

   El certificado: [{3, 5, 6}, {1, 2, 4}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje
3. **Certificado 3: [{2,3,5},{1,4,6}]**
   Entrada: `python main.py graficaA.txt certificadoA3.txt`
   Salida:
   El numero de vertices es 6
   El numero de aristas es 8
   El valor de K es 2
   La representacion de nuestra matriz es:

   0 1 1 0 0 1
   1 0 1 0 0 0
   1 1 0 1 0 0
   0 0 1 0 1 1
   0 0 0 1 0 1
   1 0 0 1 1 0

   El certificado: [{2, 3, 5}, {1, 4, 6}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia

al lenguaje

4. **Certificado 4: [{4,5,6},{1,2,3}]**
   Entrada: `python main.py graficaA.txt certificadoA4.txt`
   Salida:
   El numero de vertices es 6
   El numero de aristas es 8
   El valor de K es 2
   La representacion de nuestra matriz es:

   0 1 1 0 0 1
   1 0 1 0 0 0
   1 1 0 1 0 0
   0 0 1 0 1 1
   0 0 0 1 0 1
   1 0 0 1 1 0

   El certificado: [{4, 5, 6}, {1, 2, 3}]
   El ejemplar con el certificado dado satisface la condición de pertenencia al lenguaje
5. **Certificado 5: [{2,3,6},{1,4,5}]**
   Entrada: `python main.py graficaA.txt certificadoA5.txt`
   Salida:
   El numero de vertices es 6
   El numero de aristas es 8
   El valor de K es 2
   La representacion de nuestra matriz es:

   0 1 1 0 0 1
   1 0 1 0 0 0
   1 1 0 1 0 0
   0 0 1 0 1 1
   0 0 0 1 0 1
   1 0 0 1 1 0

   El certificado: [{2, 3, 6}, {1, 4, 5}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje

– **Ejemplar B**
Nuestro segundo ejemplar es

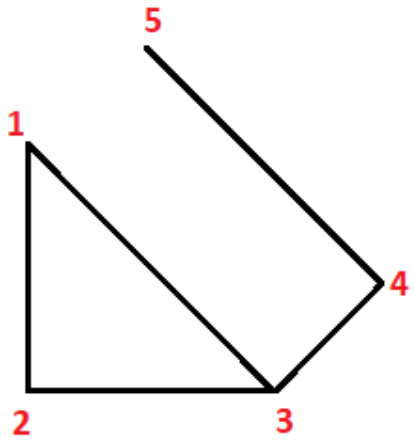

Figure 3: Gráfica B

Cuya codificación se compone de $k = 3$ y nuestra matriz de adyacencia, podemos encontrar la codificación en su archivo correspondiente *graficaB.txt*
Tenemos la siguientes ejecuciones con los certificados

1. **Certificado 1: [{2}, {3, 5}, {1, 4}]**
   Entrada: `python main.py graficaB.txt certificadoB1.txt`
   Salida:
   El numero de vertices es 5
   El numero de aristas es 5
   El valor de K es 3
   La representacion de nuestra matriz es:

   0 1 1 0 0
   1 0 1 0 0
   1 1 0 1 0
   0 0 1 0 1
   0 0 0 1 0

   El certificado: [{2}, {3, 5}, {1, 4}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje

2. **Certificado 2: [{5}, {1, 2}, {3, 4}]**
   Entrada: `python main.py graficaB.txt certificadoB2.txt`
   Salida:
   El numero de vertices es 5
   El numero de aristas es 5
   El valor de K es 3
   La representacion de nuestra matriz es:

   0 1 1 0 0
   1 0 1 0 0
   1 1 0 1 0
   0 0 1 0 1
   0 0 0 1 0

   El certificado: [{5}, {1, 2}, {3, 4}]
   El ejemplar con el certificado dado satisface la condición de pertenencia al lenguaje

   En efecto, dichos subconjuntos de vértices todoas forman un clan. Como dato curioso nos tomó 10 generadores aleatorios para dar con alguno que perteneciera al lenguaje.
3. **Certificado 3: [{2, 5}, {1, 3}, {4}]**
   Entrada: `python main.py graficaB.txt certificadoB3.txt`
   Salida:
   El numero de vertices es 5
   El numero de aristas es 5
   El valor de K es 3
   La representacion de nuestra matriz es:

   0 1 1 0 0
   1 0 1 0 0
   1 1 0 1 0
   0 0 1 0 1
   0 0 0 1 0

   El certificado: [{2, 5}, {1, 3}, {4}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje

4. **Certificado 4: [{2}, {1, 3}, {4, 5}]**
   Entrada: `python main.py graficaB.txt certificadoB4.txt`
   Salida:
   El numero de vertices es 5
   El numero de aristas es 5
   El valor de K es 3
   La representacion de nuestra matriz es:

   0 1 1 0 0
   1 0 1 0 0
   1 1 0 1 0
   0 0 1 0 1
   0 0 0 1 0

   El certificado: [{2}, {1, 3}, {4, 5}]
   El ejemplar con el certificado dado satisface la condición de pertenencia al lenguaje
5. **Certificado 5: [{3, 5}, {2, 4}, {1}]**
   Entrada: `python main.py graficaB.txt certificadoB5.txt`
   Salida:
   El numero de vertices es 5
   El numero de aristas es 5
   El valor de K es 3
   La representacion de nuestra matriz es:

   0 1 1 0 0
   1 0 1 0 0
   1 1 0 1 0
   0 0 1 0 1
   0 0 0 1 0

   El certificado: [{3, 5}, {2, 4}, {1}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje

– **Ejemplar C**
Por último

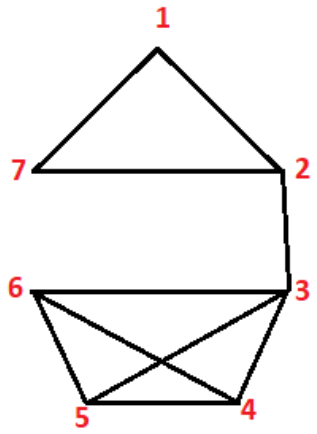

Figure 4: Gráfica C

Cuya codificación se compone de $k = 2$ y nuestra matriz de adyacencia, podemos encontrar la codificación en su archivo correspondiente *graficaC.txt*
Tenemos la siguientes ejecuciones con los certificados

1. **Certificado 1: [{2, 3, 4, 5}, {1, 6, 7}]**
   Entrada: `python main.py graficaC.txt certificadoC1.txt`
   Salida:
   El numero de vertices es 7
   El numero de aristas es 10
   El valor de K es 2
   La representacion de nuestra matriz es:

   0 1 0 0 0 0 1
   1 0 1 0 0 0 1
   0 1 0 1 1 1 0
   0 0 1 0 1 1 0
   0 0 1 1 0 1 0
   0 0 1 1 1 0 0
   1 1 0 0 0 0 0

   El certificado: [{2, 3, 4, 5}, {1, 6, 7}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje

2. **Certificado 2: [{1, 2, 3, 7}, {4, 5, 6}]**
   Entrada: `python main.py graficaC.txt certificadoC2.txt`
   Salida:
   El numero de vertices es 7
   El numero de aristas es 10
   El valor de K es 2
   La representacion de nuestra matriz es:

   0 1 0 0 0 0 1
   1 0 1 0 0 0 1
   0 1 0 1 1 1 0
   0 0 1 0 1 1 0
   0 0 1 1 0 1 0
   0 0 1 1 1 0 0
   1 1 0 0 0 0 0

   El certificado: [{1, 2, 3, 7}, {4, 5, 6}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje
3. **Certificado 3: [{1, 2, 5, 6}, {3, 4, 7}]**
   Entrada: `python main.py graficaC.txt certificadoC3.txt`
   Salida:
   El numero de vertices es 7
   El numero de aristas es 10
   El valor de K es 2
   La representacion de nuestra matriz es:

   0 1 0 0 0 0 1
   1 0 1 0 0 0 1
   0 1 0 1 1 1 0
   0 0 1 0 1 1 0
   0 0 1 1 0 1 0
   0 0 1 1 1 0 0
   1 1 0 0 0 0 0

   El certificado: [{1, 2, 5, 6}, {3, 4, 7}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje

4. **Certificado 4: [{2, 4, 5, 6}, {1, 3, 7}]**
   Entrada: `python main.py graficaC.txt certificadoC4.txt`
   Salida:
   El numero de vertices es 7
   El numero de aristas es 10
   El valor de K es 3
   La representacion de nuestra matriz es:

   0 1 0 0 0 0 1
   1 0 1 0 0 0 1
   0 1 0 1 1 1 0
   0 0 1 0 1 1 0
   0 0 1 1 0 1 0
   0 0 1 1 1 0 0
   1 1 0 0 0 0 0

   El certificado: [{2, 4, 5, 6}, {1, 3, 7}]
   El ejemplar con el certificado dado NO satisface la condición de pertenencia al lenguaje
5. **Certificado 5: [{1, 2, 7}, {3, 4, 5, 6}]**
   Entrada: `python main.py graficaC.txt certificadoC5.txt`
   Salida:
   El numero de vertices es 7
   El numero de aristas es 10
   El valor de K es 2
   La representacion de nuestra matriz es:

   0 1 0 0 0 0 1
   1 0 1 0 0 0 1
   0 1 0 1 1 1 0
   0 0 1 0 1 1 0
   0 0 1 1 0 1 0
   0 0 1 1 1 0 0
   1 1 0 0 0 0 0

   El certificado: [{1, 2, 7}, {3, 4, 5, 6}]
   El ejemplar con el certificado dado satisface la condición de pertenencia al lenguaje

Referencias

- Problema del clique: `https://es.wikipedia.org/wiki/Problema_del_clique`
- Problema del clique: `http://www.cs.ecu.edu/karl/6420/spr16/Notes/NPcomplete/clique.html`
- Teoría de gráficas: `https://targatenet.com/2017/02/06/graph-in-computer-science-and-its-types/`
- Teoría de gráficas: `https://www.baeldung.com/cs/graph-theory-intro`
- Grafica bipartita `https://www.techiedelight.com/es/bipartite-graph/`
- Propiedades de Complejidad `https://www.cs.us.es/~jalonso/cursos/i1m-19/temas/tema-28.html`
- Propiedades de Complejidad. Curso de analisis de algoritmos 2021-2.