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

Complejidad Computacional
Semestre 2023-1

Práctica 1

Ejercicio 1

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?

- Transforma el problema anterior a su versión de optimización:

Para transformalo a un problema de optimización debemos añadir un parametro que servirá como cota superior o inferior, digamos m, el cual depende si queremos maximizar o minimizar la solución buscada. Así, el problema quedaría como sigue:

EJEMPLAR: Sea una gráfica $G = (V, E)$, un entero positivo $K \leq |V|$ y una cota m (a saber)

PREGUNTA: Encuentre 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 sea un clan, además tenemos que:

$m \leq i$ si queremos una cantidad de conjuntos que maxime el resultado

$i \leq m$ si queremos una cantidad de conjuntos que minimce el resultado.

Ejercicio 2

Describir e implementar un esquema de codificación razonable para ejemplares del problema PARTICION EN CLANES (en su versión de decisión).

Descripción:

El problema de Particion en clanes en su versión de decisión consiste en determinar si 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.

Para codificar el problema, se iniciará leyendo la gráfica (no dirigida) como una matriz de adyacencias. Recordemos que por ejemplo, la matriz de adyacencia siguiente:

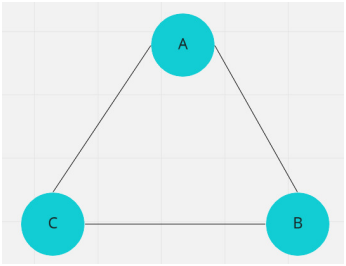
011
101
110

representa una gráfica de 3 vértices, digamos (A, B, C) en la que todos los vértices están

conectados entre sí, ya que el par ordenado $(x, y) = 0$ significa que x no es adyacente a y, y el par ordenado $(x, y) = 1$ significa que x es adyacente a y. Además notemos que ningún vértice está conectado así mismo.

Por ejemplo, en la matriz siguiente $(A, B) = 1$ significa que el vértice A es adyacente al vértice B.

_	A	B	C
A	0	1	1
B	1	0	1
C	1	1	0

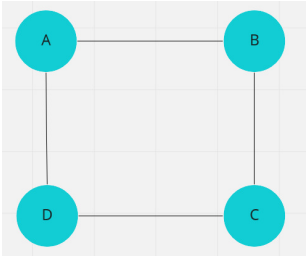


Salta a la vista el hecho de que si tenemos una grafica con n nodos, todos estos nodos pueden estar conectados entre sí (salvo así mismo), por lo cual para contemplar todos estos casos la matriz tiene que ser de $n \times n$.

Entonces nuestro alfabeto de entrada sería el siguiente: $\{1, 0, \backslash n\}$, donde \n significa un salto de linea.
Otro ejemplo es el siguiente

Una gráfica que forme un cuadrado (con vértices A, B, C, D) su matriz de adyacencia es

_	A	B	C	D
A	0	1	0	1
B	1	0	1	0
C	0	1	0	1
D	1	0	1	0



De esta forma podemos modelar nuestro programa de la siguiente manera

ENTRADA: El número de particiones codificado en binario, después de dos saltos de linea la Matriz de adyacencias.

0 0 1 0

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

En este caso particular la longitud es de 16 por la matriz más 4 del numero en binario y por último 5 saltos de linea, siendo un total de 25 caracteres.

NOTA: Los espacios en blanco **no** los contamos, **no** forman parte de nuestro lenguaje, los ocupamos solo para que no se vea tan encimado tal como

0010
0101
1010
0101
1010

Se ve más bonito.

Asi entonces podemos obtener cada uno de los siguientes puntos de la siguientes formas

SALIDA:

1. **Número de vértices**: Longitud de las columnas de la matriz, aunque también podría ser de las filas quitando el primero que lo ocupamos para codificar el número k. En este caso particular es igual a 4.

2. **Número de aristas**: Número de 1's de la matriz entre 2. Para este caso 4. No contamos el primer renglón ya que no forma parte de la matriz de adyacencias.

3. **Valor de K**: Directo del primer renglón.

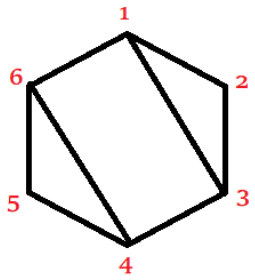
4. **Codificación del primer vértice de G**: A partir del 3 renglón podemos saber la cantidad de aristas que indicen en él y en qué vértice, además los podríamos etiquetar con letras para que sea más fácil de leer, siendo el primero A, el segundo B, y así sucesivamente.

5. **Codificación de la primera arista**: El primer 1 leido a partir del tercer renglón cuyos vértices igual podemos deducir a partir de su posición.

Veamos algunos ejemplos para entender mejor el funcionamiento.

A) Ejemplar con al menos 6 vértices, $K = 2$ y con respuesta SÍ

Consideremos la siguiente gráfica:



Entonces nuestra matriz de adyacencia es la siguiente

_	1	2	3	4	5	6
1	0	1	1	0	0	1
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	0	0	0	1	0	1
6	1	0	0	1	1	0

Ahora nuestra entrada al programa será el número k seguido de dos saltos de linea y de la matriz de adyacencia. Por lo cual la cadena de entrada es

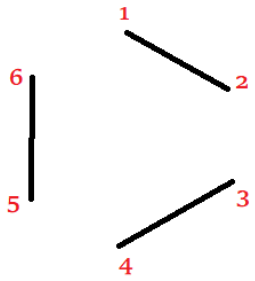
0 0 0 0 0 1 0

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

Así podemos obtener los siguientes puntos

1. **Número de vértices**: Longitud de las columnas de la matriz, en este caso 6.
2. **Número de aristas**: Número de 1's de la matriz entre 2. En este caso 8, no estamos contando el primer renglón puesto que no forma parte de la matriz de adyacencias.
3. **Valor de K**: 000010 es el binario de 2.
4. **Codificación del primer vértice de G**: A partir del 3^{er} renglón. 011001
5. **Codificación de la primera arista**: La codificación corresponde a la posición del primer 1 en la matriz. Pongamos una tupla (x, y) donde x es de izquierda a derecha y y de arriba abajo empezando desde el 1 y sin contar el primer renglón. Entonces la primer arista es $(2, 1)$, quiere decir que hay una arista que conecta los vértices 2 y 1.

B) Ejemplar con al menos 6 vértices, $K = 3$ y con respuesta SÍ.
Consideremos la siguiente gráfica:



Cuya matriz de adyacencia es la siguiente:

_	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	0	0	0	0	0
3	0	0	0	1	0	0
4	0	0	1	0	0	0
5	0	0	0	0	0	1
6	0	0	0	0	1	0

Nuestra entrada al programa será el número k seguido de dos saltos de linea y de la matriz de adyacencia. Por lo cual la cadena de entrada es

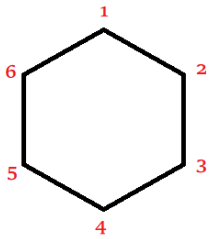
$$
\begin{array}{c}
0\ 0\ 0\ 0\ 1\ 1 \\
0\ 1\ 0\ 0\ 0\ 0 \\
1\ 0\ 0\ 0\ 0\ 0 \\
0\ 0\ 0\ 1\ 0\ 0 \\
0\ 0\ 1\ 0\ 0\ 0 \\
0\ 0\ 0\ 0\ 0\ 1 \\
0\ 0\ 0\ 0\ 1\ 0
\end{array}
$$

Así podemos tener que

1. **Número de vértices**: Longitud de las columnas de la matriz, en este caso 6.
2. **Número de aristas**: Número de 1's de la matriz entre 2. En este caso 3, no estamos contando el primer renglón puesto que no forma parte de la matriz de adyacencias.
3. **Valor de K**: 000011 es el binario de 3.
4. **Codificación del primer vértice de G**: A partir del 3^{er} renglón. 010000
5. **Codificación de la primera arista**: La codificación corresponde a la posición del primer 1 en la matriz. Pongamos una tupla (x, y) donde x es de izquierda a derecha y y de arriba abajo empezando desde el 1 y sin contar el primer renglón. Entonces la primer arista es $(2, 1)$, quiere decir que hay una arista que conecta los vértices 2 y 1.

C) Ejemplar con al menos 6 vértices, $K = 2$ y con respuesta NO.

Consideremos la siguiente gráfica:



Cuya matriz de adyacencia es la siguiente:

_	1	2	3	4	5	6
1	0	1	0	0	0	1
2	1	0	1	0	0	0
3	0	1	0	1	0	0
4	0	0	1	0	1	0
5	0	0	0	1	0	1
6	1	0	0	0	1	0

Nuestra entrada al programa será el número k seguido de dos saltos de linea y de la matriz de adyacencia. Por lo cual la cadena de entrada es

0 0 0 0 1 0

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

Así podemos tener que

1. **Número de vértices**: Longitud de las columnas de la matriz, en este caso 6.
2. **Número de aristas**: Número de 1's de la matriz entre 2. En este caso 6.
3. **Valor de K**: 000010 es el binario de 2.
4. **Codificación del primer vértice de G**: A partir del 3^{er} renglón. 010001
5. **Codificación de la primera arista**: La primer arista es $(2, 1)$, quiere decir que hay una arista que conecta los vértices 2 y 1. De nuevo, no contamos el primer rengñon, ya que este es el valor de k

Ejercicio 3

Usaremos algunas propiedades de las gráficas que no servirán para facilitar la solución del problema los cuales podemos consultar en la parte de referencias.

Para resolver el problema queremos un k específico igual a 2, esto es dividir a los vértices de la gráfica en dos conjuntos ajenos diferente del vacío. ϕ, es decir una gráfica bipartita. Para saber si una grafica es bipartita basta con saber si es 2-coloreable. (ver referencias)

Aunque no cualquier gráfica G, tomaremos su complemento G', pero ¿Por qué? Haremos un pequeño analisis antes de pasar al algoritmo y así entenderlo mejor.

Nos queda claro que si la grafica es bipartita entonces se puede dividir en dos subgráficas de tal manera que ningún vértice se conecta con otro del mismo conjunto, de hecho es la defínción. Así de esta idea sacamos la $k = 2$.

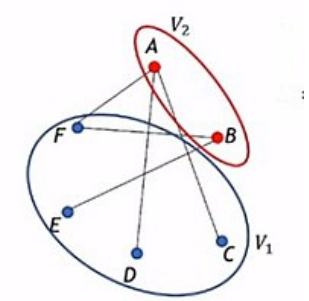
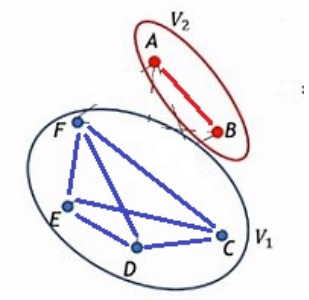


Figure 1: Imagen extraída de google

Ahora, ¿Cómo comprobamos que en efecto son clanes?

Pues bien, la misma defínción nos lo da. Si hacemos el complemento de la gráfica anterior nos daremos cuenta que los vértices encerrados en circulo azul están todos conectados entre sí, al igual que los vértices del círculo rojo. Que por defínción esto significa que ambos son clanes.



Así, para resolver el problema del clan con $k = 2$ nos auxilaremos de este analisis; buscaremos que el complemento de la grafica sea bipartita y de esta manera podemos asegurar que la grafica original contiene 2 clanes.

Algoritmo

Para empezar debemos obtener el complemento de la grafica, esto es relativamente sencillo, basta con recorrer la matriz y cambiar los 0 por 1 y los 1 por 0, a excepción de la diagonal. Para ello ocuparemos dos *for*. Esto tiene complejidad $O(n^2)$, donde n es el numero de renglones del arreglo.

Pseudocódigo

```
complemento(matriz):
    for i in matriz:
        for j in matriz[i]:
            if matriz[i][j] = 1 then matriz[i][j] = 0
            if matriz[i][j] = 0 then matriz[i][j] = 1 //cambiamos los numeros
            if i == j then matriz[i][j] = 0 //dejamos la diagonal con 0
    retun matriz
```

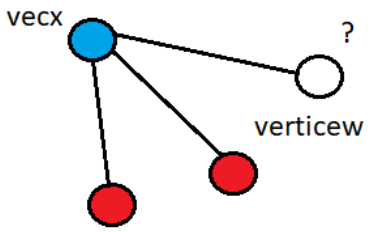
Una vez obtenido el complemento de la grafica nos ocuparemos de saber si dicho complemento es bipartito, basta con saber si la gráfica es 2-coloreable.

Después ocuparemos los dos siguientes algoritmos: *colorea(vertice), explora(vertice).*

El primero se encarga de empezar la coloración con un vertice dado, y explorar la coloración de los demás. Generalmente mandaremos a llamar este algoritmo con el vértice incial Vertices[0], aunque bien podría ser cualquier otro. No tenemos ningún ciclo ni llamadas recursivas, solo asignaciones así el algoritmo tiene complejidad constante $O(1)$

```
colorea(vertice):
    vertice.color = 1
    if explora(vertice) == True:
        return True
    else:
        return 'La grafica no contiene dos clanes'
```

A continuación explicaremos el segundo algoritmo. El algoritmo se manda a llamar con un vértice incial y de este sacamos los vecinos, *vecinos*.
Ahora para cada uno de sus vecinos *vecx* comprobamos si tienen el mismo color, que el vecino que los mandó a llamar. En caso de que tengan el mismo color podemos asegurar que no es bipartita y por lo tanto no contiene dos clanes.



En caso contrario verificamos si el vértice ha sido coloreado, sino lo fue entonces tenemos dos opciones:

- El vértice tiene color 1 por lo cual a su vecino le asginamos el color 2
- El vértice tiene color 2 por lo cual a su vecino le asginamos el color 1

En cualquiera de los dos casos mandamos a llamar el método *explora* con el vértice vecino para hacer recursión y así recorrer la gráfica. Aseguramos que el algoritmo termina pues notemos que nuestra clausula de escape es cuando todos los vértices hayan sido coloreados.

Cuando esto suceda es porque obtuvimos una 2-coloración y por lo tanto la gráfica contiene dos clanes.

```
def explora(verticew):
    vecinos = verticew.get_vecinos()
    for vecx in vecinos:
        vecx = lista_vert[vecx]
        #si tiene la misma coloracion entonces falso
        if vecx.color == verticew.color:
            return False
        else:
            if vecx.color == 0: #si el vértice aun no es coloreado
                if verticew.color == 1: #si tiene color de 1
                    vecx.color = 2 #lo coloreamos con el otro
                    explora(vecx) #recursión sobre los vecinos del vecino
                else:
                    vecx.color = 1 #lo coloreamos con el color 1
                    explora(vecx)
    #en algún momento todos estan coloreados, por lo que podemos regresar True
    return True
```

El algoritmo anterior recorre la gráfica y en el peor de los casos recorremos la grafica completa, todos los vértices y todas las aristas por lo cual tiene complejidad $O(|v| + |E|)$ donde V es el número de vértices y E el numero de aristas.

Así la complejidad del algoritmo para resolver el problema del clan cuando $k = 2$ es la suma de la complejidad de cada uno de los algoritmos anteriores.

Aunque en nuestro código utilizamos más algoritmos para lectura de archivos, o conversiones no tomaremos en cuenta estos ya que no forman parte de la solución del problema, son unicamente auxiliares para facilitar la modelación del problema a la solución.

Funcionamiento y ejemplos

En la terminal debemos ejecutar la siguiente linea

```
python main.py graficaN.txt
```

Donde *graficaN.txt* es el nombre del archivo a ejecutar y N es la letra de la grafica correspondiente. Por ejemplo si ponemos

```
python main.py graficaA.txt
```

Nuestra salida es:

```
El numero de vertices es 6
El numero de aristas es 6
El valor de K es 2
La representacion de nuestra matriz es:
[[0 1 1 0 0 0]
 [1 0 1 0 0 0]
 [1 1 0 1 0 0]
 [0 0 1 0 1 0]
 [0 0 0 1 0 0]
 [1 0 0 1 1 0]]
Aplicacion del algoritmo
Clan 1: [1, 2, 3]
Clan 2: [4, 5, 6]
```

Nuestro archivo *graficaA.txt* corresponde a nuestro ejemplo **A)** visto anteriormente. En efecto la implementación coincide con lo que queríamos resolver.

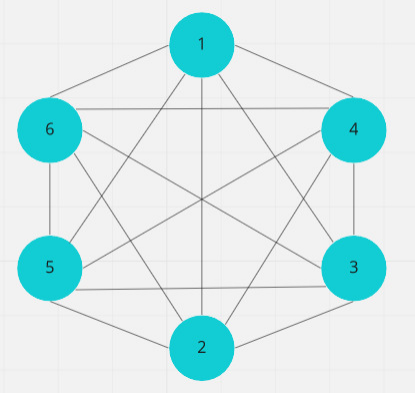
De la misma manera para el ejemplo **C)**. Entrada

```
python main.py graficaC.txt
```

Salida:

```
El numero de vertices es 6
El numero de aristas es 6
El valor de K es 2
La representacion de nuestra matriz es:
[[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]]
Aplicacion del algoritmo
No contiene dos clanes
```

Veamos ejemplos más interesantes como lo podría ser un grafica completa, en ella es evidente que cualquiera dos subconjuntos ajenos que agarremos forman un clan. Así que nos preguntamos ¿Cómo tomará a los vecinos de la grafica el programa?. Pues bien, pongamos la siguiente grafica completa con 6 vértices.



Su correspondiente codificación es.

$$
\begin{array}{c}
0\ 0\ 0\ 0\ 1\ 0 \\
0\ 1\ 1\ 1\ 1\ 1 \\
1\ 0\ 1\ 1\ 1\ 1 \\
1\ 1\ 0\ 1\ 1\ 1 \\
1\ 1\ 1\ 0\ 1\ 1 \\
1\ 1\ 1\ 1\ 0\ 1 \\
1\ 1\ 1\ 1\ 1\ 0
\end{array}
$$

Entrada al programa

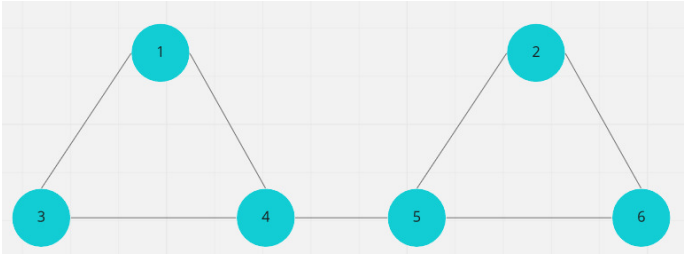
```
python main.py graficaCC.txt
```

Salida

```
El numero de vertices es 6
El numero de aristas es 15
El valor de K es 2
La representacion de nuestra matriz es:
[[0 1 1 1 1 1]
 [1 0 1 1 1 1]
 [1 1 0 1 1 1]
 [1 1 1 0 1 1]
 [1 1 1 1 0 1]
 [1 1 1 1 1 0]]
Aplicacion del algoritmo
Clan 1: [1]
Clan 2: [2, 3, 4, 5, 6]
```

Vemos que hecho la salida no es muy interesante, ya que el algoritmo parte del primer vértice y a este lo toma como el clan trivial.

Otro ejemplo esperemos ahora sí más interesante. Tomemos la siguiente grafica I



Pasemos directo a su codificación

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

Entrada del programa

```
python main.py graficaI.txt
```

Salida:

```
El numero de vertices es 6
El numero de aristas es 7
El valor de K es 2
La representacion de nuestra matriz es:
[[0 0 1 1 0 0]
 [0 0 0 0 1 1]
 [1 0 0 1 0 0]
 [1 0 1 0 1 0]
 [0 1 0 1 0 1]
 [0 1 0 0 1 0]]
Aplicacion del algoritmo
Clan 1: [1, 3, 4]
Clan 2: [2, 5, 6]
```

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/