Universidad Nacional
Autónoma de México

Facultad de Ciencias

# Complejidad Computacional

2024-2

# Práctica 1

Linares Gil, Daniel 420490056
Chávez Zamora, Mauro Emiliano 111001079

2 de marzo de 2024

# 1 Descripción del problema de optimización:

El problema de coloración plantea que si tenemos una gráfica $G = (V, E)$ y un número $k$ de colores es posible "colorear" la gráfica de manera que no existan vértices adyacentes con el mismo color asignado. Algunas definiciones de gráfica bipartita parten del problema de coloración, estableciendo $k = 2$ para su definición formal. Vamos a plantearlo con el siguiente formato visto en clase.

**K-coloración (optimización):**
**Ejemplar:** Una gráfica finita $G$.
**Pregunta:** ¿Cuál es la menor cantidad de colores que se pueden utilizar para colorearla?

# 2 Descripción del problema de decisión:

**K-coloración (decisión):**
**Ejemplar:** Una gráfica finita $G$ y un entero no negativo $k$.
**Pregunta:** ¿Existe una coloración de la gráfica con k colores?

# 3 Codificación:

Para la codificación de nuestras gráficas elegimos hacer una codificación donde cada línea de un archivo representará un vértice y las aristas se expresarán como apariciones del número de otro vértice, sabremos que hemos tenido un cambio de vértice por tener un salto de línea y si tenemos una arista hacia otro vértice simplemente aparecerá el número de ese otro vértice dentro de la línea que representa nuestro vértice inicial. De manera que si nuestra gráfica aparece compuesta por las vértices $A, B, C$ y las aristas $(A, B), (B, C)$ podemos codificar con números para $A = 1$, $B = 2$ y $C = 3$ de la siguiente manera:

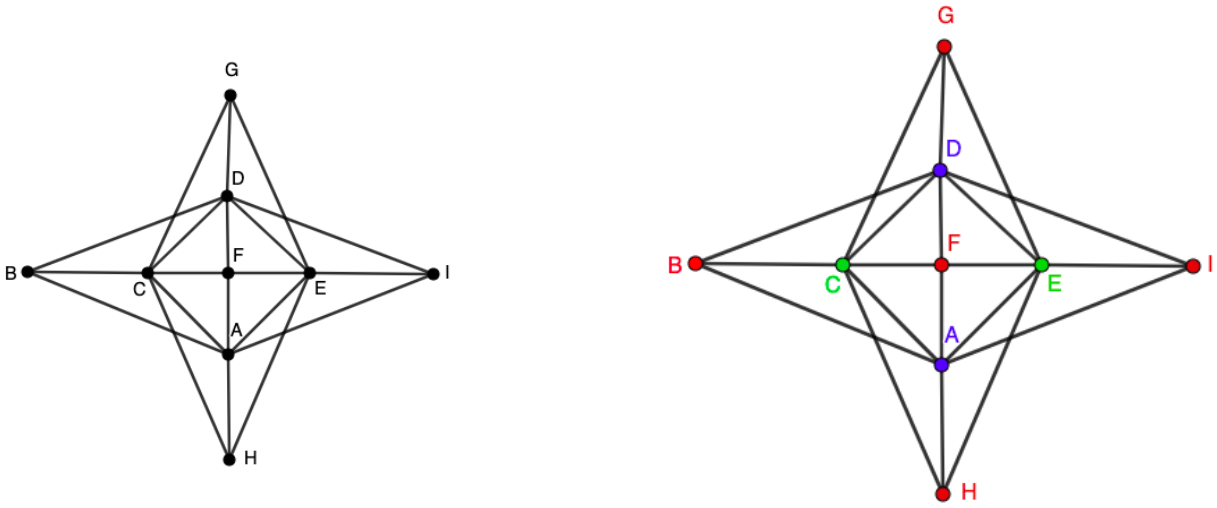
```
2 \n
1 3 \n
3 \n
```

Donde \n representa un salto de línea. Hemos elegido representar los saltos de líneas en diferentes líneas por claridad, pero podemos expresar nuestra cadena en un solo nivel de la siguiente manera:

```
2 \n 1 3 \n 3
```

Ahora veremos algunos ejemplos que codifican las gráficas bajo este esquema para nuestro programa que verifica si una gráfica es $2 - coloreable$.

## 3.1. Ejemplar 3-coloreable




La figura mostrada es $3 - colorable$, si bien no es tan claro de observar en la imagen a la izquierda, el lado derecho nos muestra cómo se vería una coloración así. Esta coloración es la mínima, ya que observamos que hay un triángulo (un clan de 3 vértices) dentro de la gráfica, por lo que no es posible colorear esa sección con menos de 3 colores. Con $A = 1$, $B = 2$, $C = 3$, $D = 4$, $E = 5$, $F = 6$, $G = 7$, $H = 8$, $I = 9$ podemos codificarla para nuestro programa.

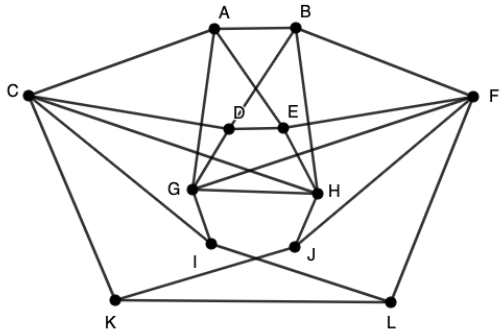
**Cadena codificada:**

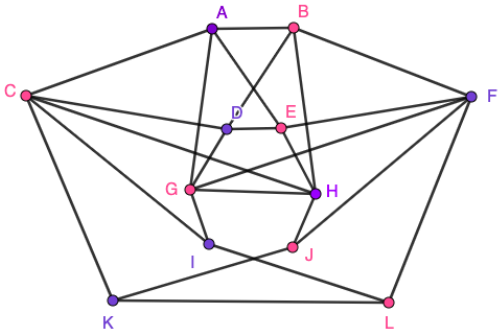
```
2 3 5 6 8 9 \n 1 3 4 \n 1 2 4 6 7 8 \n 2 3 5 6 7 9 \n 1 4 6 7 8 9 \n
1 3 4 5 \n 3 4 5 \n 1 3 5 \n 1 4 5
```

**Salida del programa:**

```
$ python src/practica.py ejemplares/ej3-3_3-coloreable.tx
Número de Vértices: 9
Número de Aristas 20
El vértice de grado máximo es 1 y tiene grado 6
¿La gráfica del ejemplar es 2-coloearble?: NO
```

## 3.2. Ejemplar 2-coloreable




Recordemos que los nombres de los vértices no son importantes, sino únicamente las relaciones entre ellos. Acá utilizaremos de nuevo la codificación $A = 1$, $B = 2$, $C = 3$, $D = 4$, $E = 5$, $F = 6$, $G = 7$, $H = 8$, $I = 9$, $J = 10$, $K = 11$ y $L = 12$.

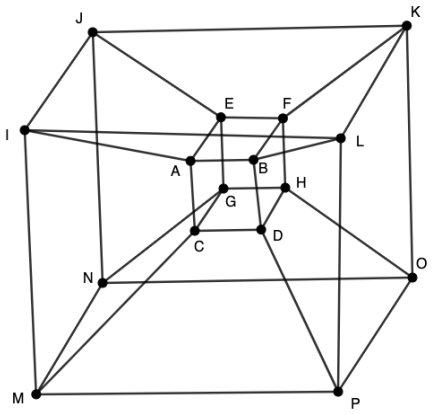
**Cadena codificada:**

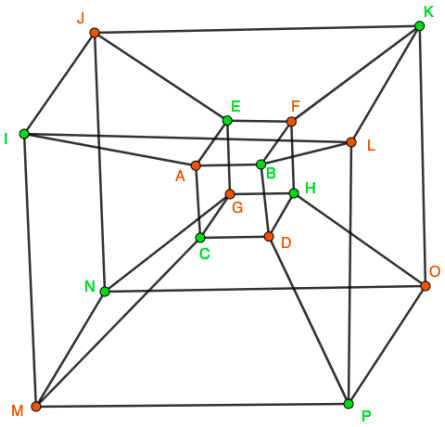
```
2 3 5 7 \n 1 4 6 8 \n 1 4 8 9 11 \n 2 5 7 3 \n 1 4 6 8 \n 2 5 7 10 12 \n
1 4 6 8 9 \n 2 3 5 7 10 \n 3 7 12 \n 6 8 11 \n 3 10 12 \n 6 9 11
```

**Salida del programa:**

```
$ python src/practica.py ejemplares/ej3-2_2-coloreable.txt
Número de Vértices: 12
Número de Aristas 24
El vértice de grado máximo es 3 y tiene grado 5
¿La gráfica del ejemplar es 2-coloreable?: SI
Vértice Color
1 0
2 1
3 1
4 0
5 1
6 0
7 1
8 0
9 0
10 1
11 0
12 1
```

## 3.3. Ejemplar 2-coloreable




Utilizamos la codificación $A = 1$, $B = 2$, $C = 3$, $D = 4$, $E = 5$, $F = 6$, $G = 7$, $H = 8$, $I = 9$, $J = 10$, $K = 11$, $L = 12$, $M = 13$, $N = 14$, $O = 15$ y $P = 16$.

**Cadena codificada:**

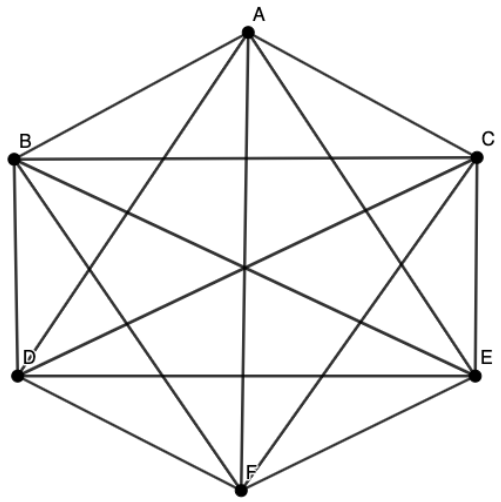
```
2 3 5 9 \n 1 4 6 12 \n 1 4 7 13 \n 2 3 8 16 \n 1 6 7 10 \n 2 5 8 11 \n
3 5 8 14 \n 4 6 7 15 \n 1 10 12 13 \n 5 9 11 14 \n 6 10 12 15 \n
2 9 11 16 \n 3 9 14 16 \n 7 10 13 15 \n 8 11 14 16 \n 4 12 13 15
```

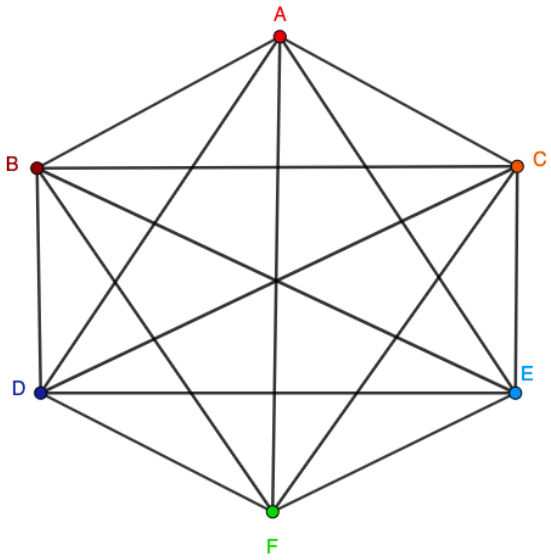
**Salida del programa:**

```
$ python src/practica.py ejemplares/ej3-3_2-coloreable.txt
Número de Vértices: 16
Número de Aristas 32
El vértice de grado máximo es 1 y tiene grado 4
¿La gráfica del ejemplar es 2-coloreable?: SI
Vértice Color
1 0
2 1
3 1
4 0
5 1
6 0
7 0
8 1
9 1
10 0
11 1
```

```
12 0
13 0
14 1
15 0
16 1
```

## 3.4. Ejemplar NO 2-coloreable




Notamos que es imposible tener coloreada una gráfica que contenga un clan con menos colores que el número de vértices que participan en el clan, así que este ejemplo es claramente no 2-coloreable. Utilizamos la codificación $A = 1$, $B = 2$, $C = 3$, $D = 4$, $E = 5$ y $F = 6$.

**Cadena codificada:**

```
2 3 4 5 6 \n 1 3 4 5 6 \n 1 2 4 5 6 \n 1 2 3 5 6 \n 1 2 3 4 6 \n 1 2 3 4 5
```

**Salida del programa:**

```
$ python src/practica.py ejemplares/ej3-4_no_2-coloreable.txt
Número de Vértices: 6
Número de Aristas 15
El vértice de grado máximo es 1 y tiene grado 5
¿La gráfica del ejemplar es 2-coloreable?: NO
```

# 4 Algoritmo:

- Se elige un vértice, se revisa si tiene un color asignado, si ya tiene color asignado se continua al siguiente vértice. Si no tiene color asignado, se le asigna el primer color y se realizan los siguientes pasos.
  - Se realiza un recorrido DFS a partir de el vértice asignandole colores a los vecinos. Se introduce el vértice en una pila y se itera mientras la pila no se encuentra vacía. En cada iteración se saca un vértice de la pila y se recorren sus vecinos. Tenemos 3 opciones:
    - El vecino ya tiene un color asignado y es el mismo que queríamos asignarle. Entonces sólo continuamos.
    - El vecino ya tiene un color asignado y no es el mismo que queremos asignarle. Entonces terminamos y decimos que NO es $2 - colorable$.
    - El vecino no tiene un color asignado, así que le asignamos el contrario al del vértice, introducimos al vecino en la pila, y continuamos a la siguiente iteración.
  - Se repite el procedimiento con cada uno de los vértices de la gráfica.
- Cuando terminamos sin incongruencias la gráfica SI es $2 - coloreable$

## 4.1. Complejidad del algoritmo:

Sea $G = (V, E)$ una gráfica con $V$ el conjunto de vértices y $E$ el conjunto de aristas. Aunque en cada iteración del ciclo externo se puede llegar a realizar un recorrido DFS y en el peor de los casos recorrer toda la gráfica, el recorrido solo se realiza si el vértice no ha sido coloreado, por lo que solo se realiza un recorrido DFS por cada componente conexa de la gráfica y cada arista solo se recorre una vez. El número aristas recorridas entre todos los recorridos DFS es igual al número de aristas de la gráfica $|E|$, por lo que la complejidad en tiempo de realizar todos los recorridos DFS es de $O(|E|)$. Adicionalmente, es necesario iterar sobre el conjunto de vértices, lo cual tiene complejidad en tiempo $O(|V|)$. La complejidad en tiempo del algoritmo es de $O(|V| + |E|)$, y como el número de aristas de una gráfica crece a lo más $O(|V|^2)$ respecto al número de vértices, la complejidad en tiempo respecto al número de vértices es $O(|V|^2)$.

# Referencias:

[1] Goldreich, O. (2008). Computational complexity: A Conceptual Perspective. Cambridge University Press.

[2] Papadimitriou, C. H. (1994). Computational complexity. Pearson.

[3] Ayudantías y clases con el profesor.