

# Matemática Discreta I

## Clase 23 - Algoritmos greedy en grafos

FAMAF / UNC

10 de junio de 2021

# Algoritmo greedy para coloración de vértices

- No se conoce ningún algoritmo general para encontrar el número cromático de un grafo que trabaje en “tiempo polinomial”
- Sin embargo hay un método simple de hacer una coloración cromática usando un “razonable” número de colores.

El algoritmo es muy sencillo y se puede describir en una sola línea.

- Si hay vértices no coloreados, elegimos un vértice no coloreado y le otorgamos un color que no tengan sus vecinos.

En este algoritmo insistimos en hacer la mejor elección que podemos en cada paso, sin mirar más allá para ver si esta elección nos traerá problemas luego.

Un algoritmo de esta clase se llama a menudo un *algoritmo greedy (goloso)*.

El algoritmo greedy para coloración de vértices es fácil de programar.

Supóngase que hemos dado a los vértices algún orden  $v_0, v_1, \dots, v_n$ .

- Asignemos el color 0 a  $v_0$ .
- Tomamos  $v_i$  el siguiente vértice de la lista y
  - $S$  = el conjunto de colores asignados a los vértices  $v_j$  ( $0 \leq j < i$ ) que son adyacentes a  $v_i$ .
  - Le damos a  $v_i$  el primer color que no está en  $S$ .
- si  $i < n$  volvemos hacemos el procedimiento del paso anterior para  $i = i + 1$ .

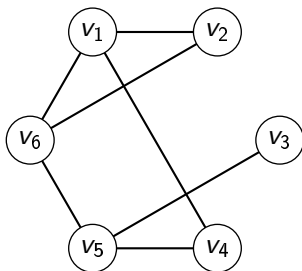
Mostramos en pseudocódigo del algoritmo greedy para coloración de vértices.

```
# pre: 0,...,n los vértices de un grafo G
# post: devuelve v[0],...,v[n] una coloración de G
color = [] # color[j] = c dirá que el color de j es c.
for i = 0 to n:
    S = [] # S conjunto de colores asignados a los vértices j
           # (1 <= j < i) que son adyacentes a i (comienza vacío)
    for j = 0 to i-1:
        if j es adyacente a i:
            S.append(color[j]) # agrega el color de j a S
    k = 0
    while k in S:
        k = k + 1
    color.append(k) # Asigna el color k a i, donde k es el primer
                   # color que no esta en S.
```

Debido a que la estrategia greedy es corta de vista, el número de colores que usará será normalmente más grande que el mínimo posible.

## Ejemplo

Aplicar el algoritmo greedy a



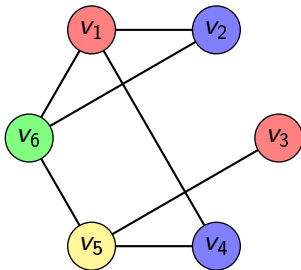
## Solución

El orden de los vértices es  $v_1, v_2, v_3, v_4, v_5$  y  $v_6$ .

El algoritmo es:

- **Paso 1.**  $v_1$  tiene colores vecinos  $S = \emptyset \Rightarrow v_1$  color 0.
- **Paso 2.**  $v_2$  tiene colores vecinos  $S = \{0\} \Rightarrow v_2$  color 1.
- **Paso 3.**  $v_3$  tiene colores vecinos  $S = \emptyset \Rightarrow v_3$  color 0.
- **Paso 4.**  $v_4$  tiene colores vecinos  $S = \{0\} \Rightarrow v_4$  color 1.
- **Paso 5.**  $v_5$  tiene colores vecinos  $S = \{0, 1\} \Rightarrow v_5$  color 2.
- **Paso 6.**  $v_6$  tiene colores vecinos  $S = \{0, 1, 2\} \Rightarrow v_6$  color 3.

Es decir el coloreo queda:



El orden que se elige inicialmente para los vértices es fundamental para establecer la coloración.

Es bastante fácil ver que si se elige el orden correcto, entonces el algoritmo greedy nos da la mejor coloración posible (ejercicio en el apunte).

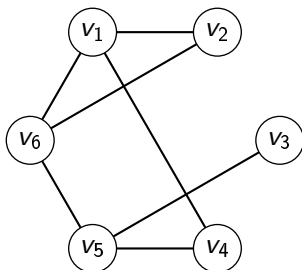
Pero hay  $n!$  órdenes posibles, y si tuviéramos que controlar cada uno de ellos, el algoritmo requeriría “tiempo factorial” (peor aún que tiempo exponencial).



## Ejemplo

Aplicar el algoritmo greedy al siguiente grafo donde el orden de los vértices es

$v_3, v_4, v_6, v_2, v_5, v_1$ .

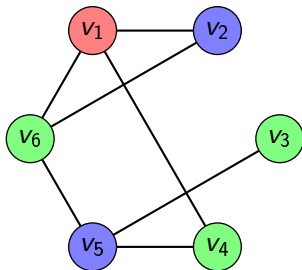


## Solución

El algoritmo es:

- **Paso 1.**  $v_3$  tiene colores vecinos  $S = \emptyset \Rightarrow v_3$  color 0.
- **Paso 2.**  $v_4$  tiene colores vecinos  $S = \emptyset \Rightarrow v_4$  color 0.
- **Paso 3.**  $v_6$  tiene colores vecinos  $S = \emptyset \Rightarrow v_6$  color 0.
- **Paso 4.**  $v_2$  tiene colores vecinos  $S = \{0\} \Rightarrow v_2$  color 1.
- **Paso 5.**  $v_5$  tiene colores vecinos  $S = \{0\} \Rightarrow v_5$  color 1.
- **Paso 6.**  $v_1$  tiene colores vecinos  $S = \{0, 1\} \Rightarrow v_1$  color 2.

Podemos representar en el grafo la coloración:



El coloreo queda igual que en la clase pasada.

El orden fue elegido de la siguiente forma: dado el coloreo con colores  $\chi(G)$ , se elije, en el orden, primero los vértices de un solo color que haya mayor cantidad, luego de otro color que haya mayor cantidad, etc.

Más allá que el algoritmo greedy no soluciona el problema, el algoritmo es útil tanto en la teoría como en la práctica.

Probaremos ahora algunos resultados por medio de la estrategia greedy.

### Teorema

*Si  $G$  es un grafo con valencia máxima  $k$ , entonces*

*a)  $\chi(G) \leq k + 1$ ,*

*b) Si  $G$  es conexo y no regular ,  $\chi(G) \leq k$ .*

## Demostración

a) Sea  $v_1, v_2, \dots, v_n$  un ordenamiento cualquiera de los vértices de  $G$ .

Para cada vértice  $v$ , si  $S$  son los colores de los vecinos a  $v \Rightarrow |S| \leq k$ .

Luego puedo elegir un color entre 0 y  $k$ .

b)

(1) Sea  $v_n$  un vértice con  $\delta(v_n) < k$ .

(2) Sean  $v_{n-1}, v_{n-2}, \dots, v_{n-r}$  los adyacentes a  $v_n$ . Hay a lo más  $k - 1$  de ellos.

(3) Luego se van eligiendo los adyacentes a  $v_i$  que no están listados antes ( $n > i \geq 1$ ).

(4) Si  $i < n$  el vértice  $v_i$  tiene un adyacente a nivel superior  $\Rightarrow v_i$  tiene a lo más  $k - 1$  adyacentes a nivel inferior.

(5) Si  $i < n$ , usando greedy y por (4), se puede colorear  $v_i$  con un color en  $\{1, \dots, k\}$ .

(6) Por (2) se puede colorear  $v_n$  con un color en  $\{1, \dots, k\}$ . □

# Grafos bipartitos



















