.UBAfiuba
FACULTAD DE INGENIERÍA

Backtracking: K coloreo de grafos

Teoría de Algoritmos I (75.29 / 95.06)
Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

K coloreo de grafos

- **Contamos con un**

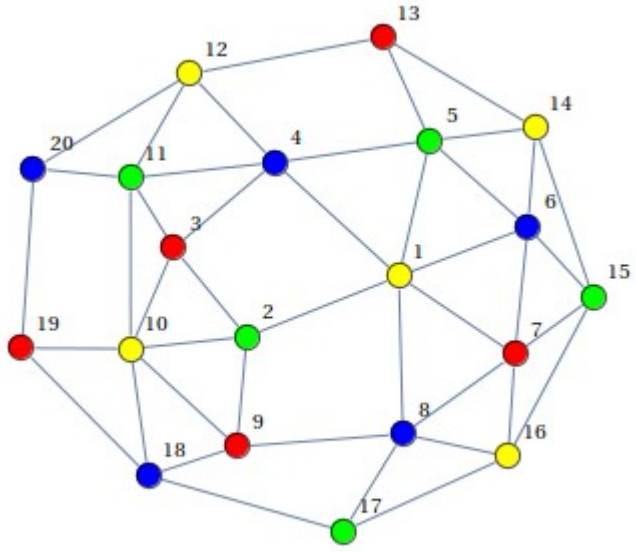
  grafo G=(V,E) con “n” vértices y “m” ejes.

- **Queremos**

  asignarles no más de K colores a sus vértices

- **De modo que**

  para cualquier par de vértices adyacentes no compartan el mismo color.



Representación de un coloreo

Estableceremos un orden de 1 a “n” para los vértices del grafo.

Llamaremos vi al vértice en la posición i

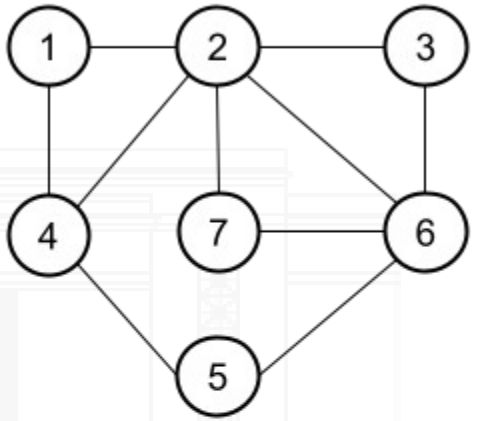
Utilizaremos k etiquetas

para representar los color

Utilizaremos un vector de n posiciones

Para representar un coloreo utilizando como mucho k etiquetas

En la posición i del vector se definirá la etiqueta del vértice vi



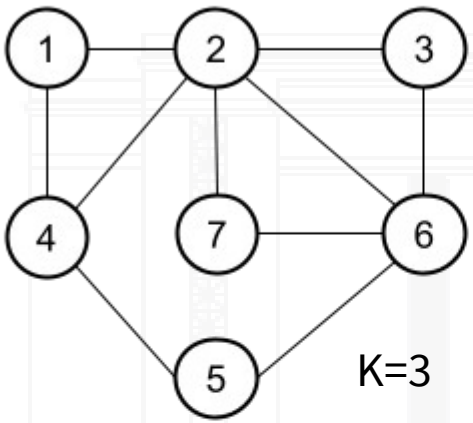
1	2	3	4	5	6	7
1	2	1	3	1	3	1

Árbol de estados del problema

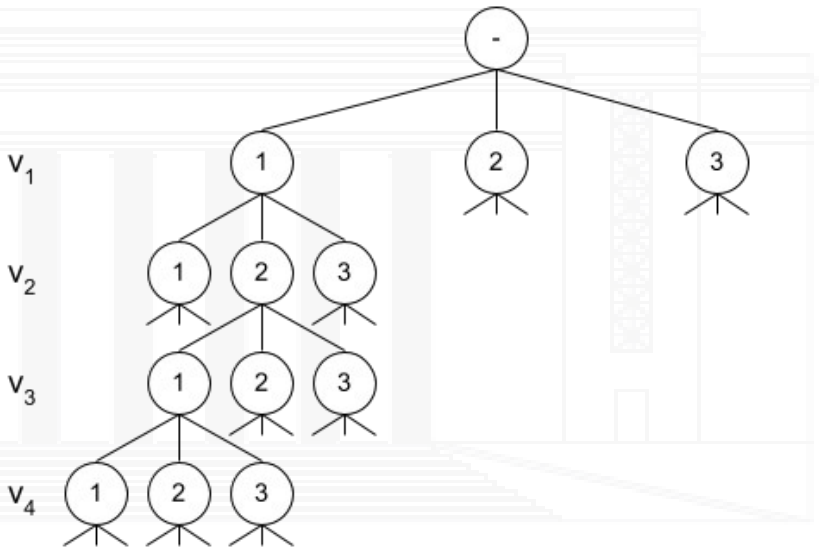
- **La raíz representa al grafo sin ninguna etiqueta en sus vértices**
- **Un nodo a profundidad i corresponde a la elección de una etiqueta al vértice vi**
  - Se agrega a los anteriores incluidos
- **Los descendientes de cada nodo**
  - Son k nodos, uno por cada etiqueta
  - Al etiquetar (colorear) un nodo podemos determinar si es un coloreo compatible con los antes coloreados
- **El árbol resultando corresponderá a un k-ario.**
  - Los estados a profundidad n del arbol corresponde a los estados solución
  - El total de estados del problema corresponde a k^n

Ejemplo: Árbol de estados del problema

Supongamos la siguiente instancia del problema:

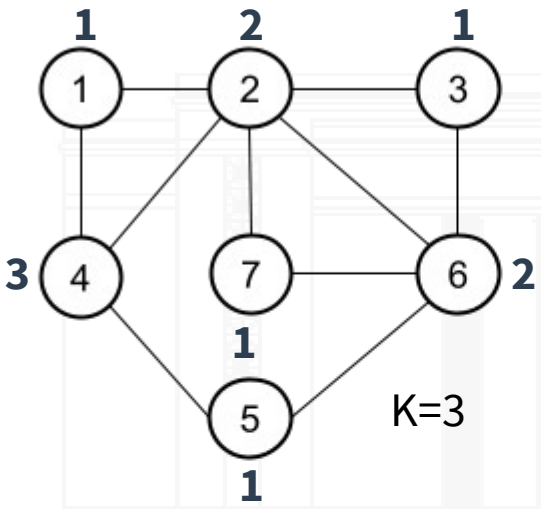


Los estados se pueden representar como un árbol k-ario (porción):

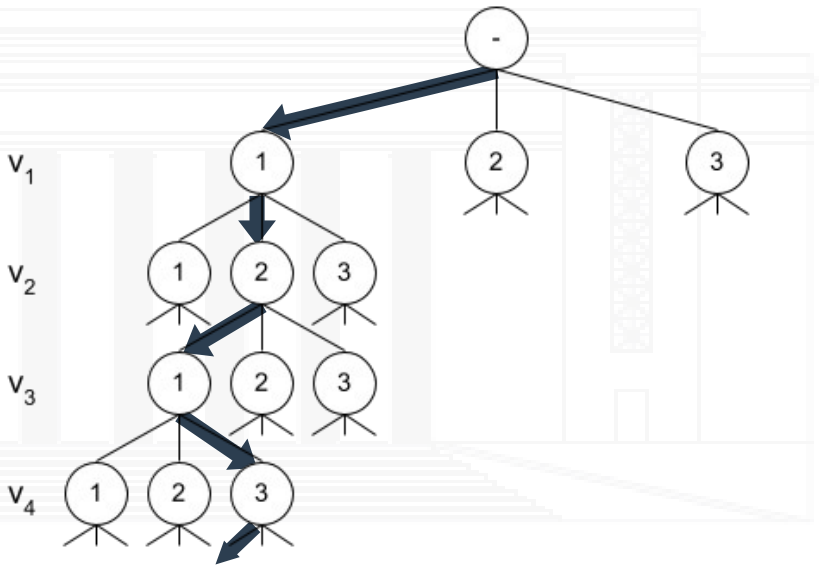


Ejemplo: Árbol de estados del problema

Posible coloreo:



Equivale al recorrido en el árbol (porción):



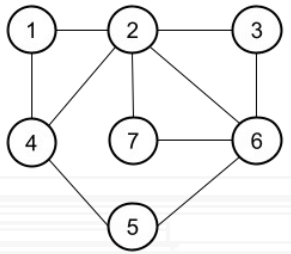
Poda del árbol

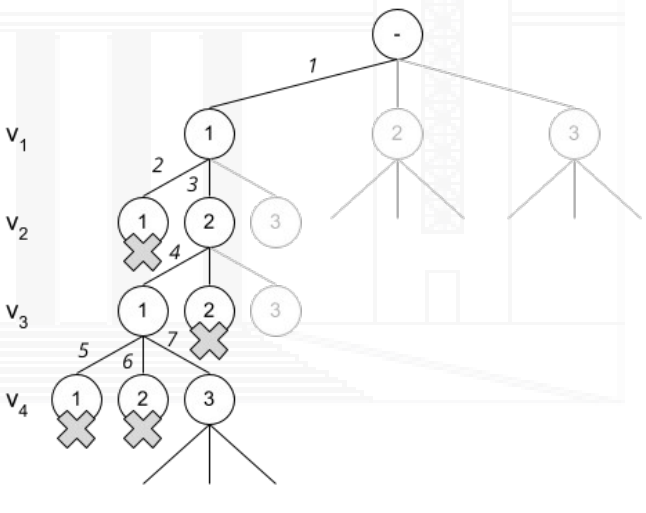
No todos los coloreos son posibles

Si asignamos un color a un vértices del grafo que tiene el mismo que un vértice previamente asignado, podemos podar todos los coloreos que los utilicen

La función limite verifica las etiquetas de los vértices adyacente del último vértice etiquetado

En caso de coincidencia, poda el nodo y desestima la inspección de todas las ramas descendientes del mismo





Backtracking - Pseudocódigo

```
Sea C[x] el color asignado al vértice x
Sea A[x] la lista de vértices adyacentes de la vertice x.

vertice=1
Si Backtrack(vertice)
     Imprimir C[x] para todo vertice x del grafo
Sino
     Imprimir ‘no hay un coloreo posible’
```

Backtracking – Pseudocódigo (II)

```
Backtrack (nroVertice):

    Por cada color disponible
        C[nroVertice]=color

        Sea coloreoValido = true
        Por cada vertice x en A[nroVertice]
            Si C[nroVertice] == C[x]
                coloreoValido =false

        Si (coloreoValido y nroVertice==n)
            retornar true
        Si (coloreoValido y nroVertice<n)
            Si Backtrack(nroVertice+1)
                retornar true
        Quitar color a C[nroVertice]
    Retornar false
```

Complejidad temporal

En el peor de los casos no es posible podar nodos del arbol

Debemos recorrer cada uno de los nodos del árbol con una complejidad $O(k^n)$

En los nodos en el peor de los casos debemos hacer un trabajo O(n)

Correspondiente a comparar el etiquetado de los vértices adyacentes del grafo

En los nodos debemos hacer un trabajo O(k)

Para generar los nodos descendientes por cada color posible

La Complejidad final corresponde a la multiplicación de estas complejidades

Complejidad espacial

Por la implementación recursiva

Por cada llamado en profundidad en el árbol incluimos el consumo de memoria adicional

La memoria utilizada

Es proporcional a la profundidad máxima de la recursión generada

Para el árbol combinatorio la profundidad máxima es “n”.

En cada nivel de profundidad

Se define un color a un vértice

Se realizan cálculos que requieren O(1) de almacenamiento

Por lo que la complejidad espacial es O(n)