Algoritmos y Estructuras de Datos II
Clase 22

Carreras:

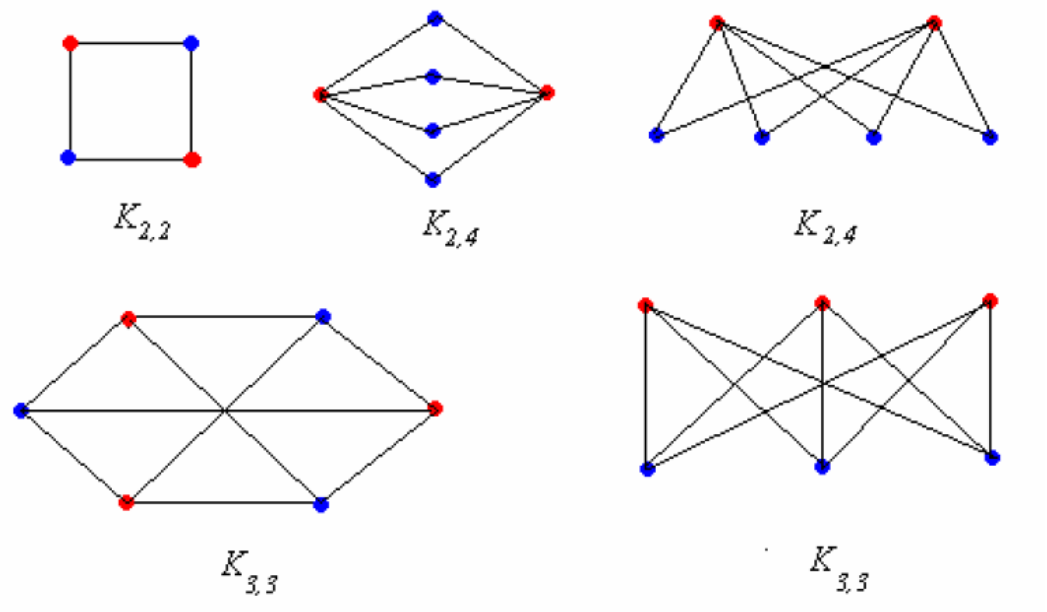
Licenciatura en Informática

Ingeniería en Informática

2024

Unidad IV

Algoritmos en Grafos(2)



Problema del viajante
TSP- Traveling Salesman Problem

Se tiene un mapa de ciudades con las distancias (no negativas) entre ellas.

El viajante quiere salir de una de las ciudades, visitar todas las demás ciudades exactamente una vez y volver al punto de partida habiendo recorrido la menor distancia posible.

El problema entonces consiste en encontrar ***el ciclo hamiltoniano de menor distancia*** de un grafo conexo y ponderado.

No se conocen algoritmos polinomiales para resolver el problema del viajante de comercio.

Este famoso problema tiene dificultad NP.

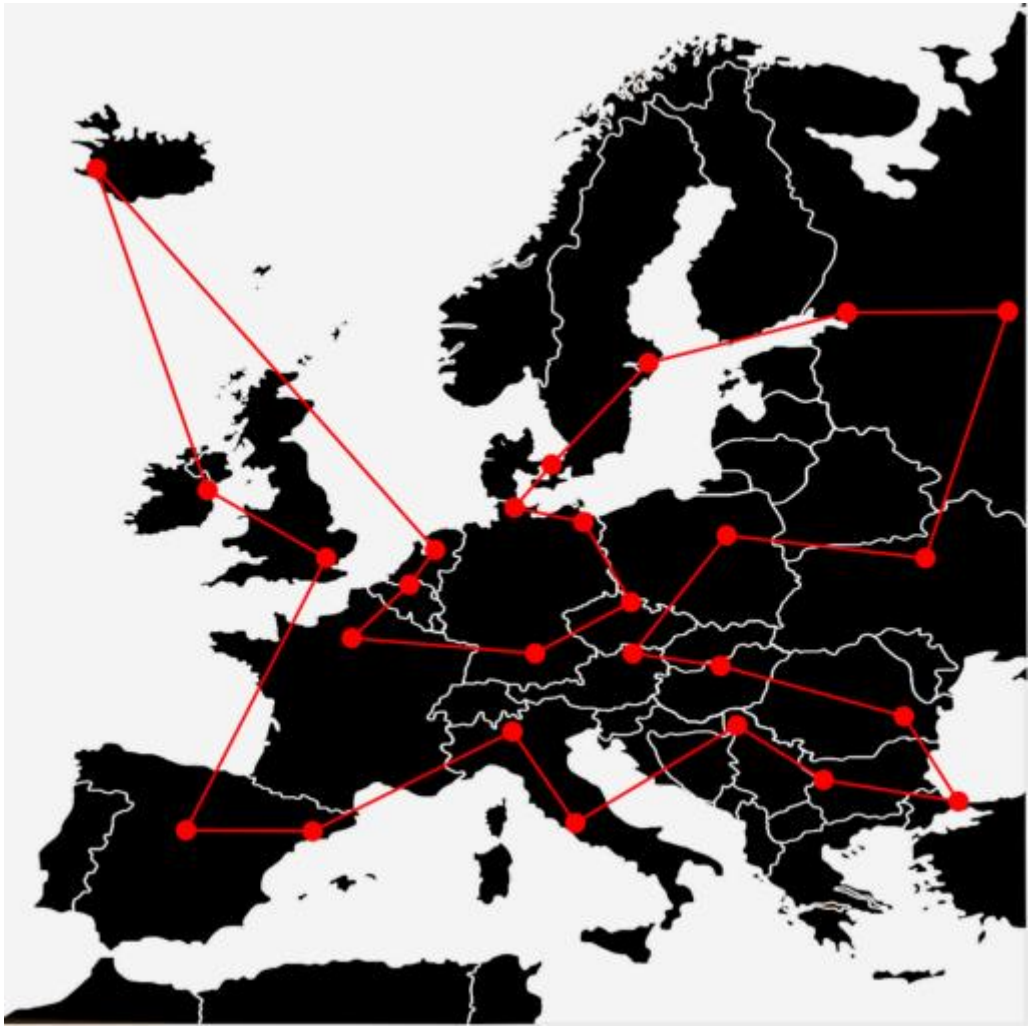
*Es "**el problema**" de optimización combinatoria más estudiado.*

Problema del viajante
TSP- Traveling Salesman Problem

Ejemplo:

Problema del viajante
TSP- Traveling Salesman Problem

Ejemplo:



Problema del viajante
TSP- Traveling Salesman Problem

- El problema fue definido en los años 1800s por el matemático irlandés Hamilton y por el matemático británico Kirkman.
- Es uno de los problemas de optimización más estudiados.
- Se conocen una gran cantidad de métodos exactos y heurísticas.
- El TSP tiene diversas aplicaciones: la planificación, la logística, en la fabricación de microchips, y también en la secuencia de ADN.
- La evidente dificultad computacional para encontrar recorridos óptimos tiene su explicación matemática.
- En la teoría de la complejidad pertenece a la clase de los problemas NP- completos, según lo mostró en 1972 Richard Karp.
- Aun así, hoy en día pueden ser resueltas algunas instancias desde cien hasta miles de ciudades.

Problema del viajante
TSP- Traveling Salesman Problem

- Gran progreso en los algoritmos a finales de los 70s y principios de los 80s, donde se comenzaron a usar métodos de Ramificación y Poda. Se llego a resolver el problema para cerca de 2500 ciudades.
- En los 90s, un equipo en el que participaba Cook desarrolló el programa llamado Concorde.
- En 1991 se publicó el TSPLIB, una colección de instancias de pruebas de dificultad variable, la cual es usada por muchos grupos de investigación para comparar resultados.
- En 2006, Cook resolvió un recorrido óptimo para 85900 ciudades dado por un problema de diseño de microchip.
- Hoy en día se puede resolver para millones de ciudades con algoritmos que si bien no obtienen la solución optima son capaces de obtener una solución “casi optima”.

Problema del viajante
TSP- Traveling Salesman Problem

Por qué es un problema “difícil de resolver” ?:

- Para N ciudades en principio existen N! caminos posibles.
- Como es un ciclo, no importa el vértice de partida se simplifica a (N-1)! caminos
- Si No importa la dirección, se reducen a (N-1)!/2 caminos posibles.

Para el problema del viajante de tamaño:

- N=5 ciudades hay 12 caminos diferentes.
- N=10 ciudades hay 181.440 caminos diferentes.
- N=30 ciudades hay más de $4 \cdot 10^{31}$ rutas posibles. (si se calcula un millón de caminos por segundo, se necesitaría 10^{18} años para resolverlo).

Problema del viajante

- Este problema también puede ser enunciado más formalmente como sigue: dado un grafo *g* conexo y ponderado y dado uno de sus vértices v_0, encontrar el ciclo Hamiltoniano de costo mínimo que comienza y termina en v_0.

Para resolverlo por un ***algoritmo greedy***, se pueden plantear las siguientes estrategias:

a)

Sea v_0 el vértice origen.

Sea (C,v) el camino construido hasta el momento que comienza en v_0 y termina en v.

Inicialmente C es vacío y $v = v_0$.

Si el camino C contiene todos los vértices de g, el algoritmo incluye el arco (v,v_0) y termina.

Si no, debe incluir el arco (v,w) de ***longitud mínima*** entre todos los arcos desde v a los vértices w que no están en el camino C.

Problema del viajante

Ejemplo1:Dada la matriz de costos de un grafo *g*1 con 4 vértices.

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

Partiendo del vértice 1, el algoritmo greedy encuentra la solución óptima, que está formada por los arcos:

(1,2),(2,3),(3,4),(4,1)

lo que da lugar al ***ciclo óptimo*** (1,2,3,4,1), cuyo costo es:

1 + 4 + 3 + 2 = 10,

Ya que otras soluciones poseen costos superiores o iguales a él:

15, 17, 14, 17 y 10.

Problema del viajante

Ejemplo2: Dada la matriz de costos de un grafo *g2* con 5 vértices.

Partiendo del vértice 1 el algoritmo greedy
Elige la secuencia de arcos:
(1,2),(2,3),(3,5),(5,4),(4,6),(6,1)
lo que da lugar al ciclo: (1,2,3,5,4,6,1),
cuyo costo es:
3 + 6 + 4 + 5 + 15 + 25 = 58

Ese no es el ciclo con menor costo,
el camino definido por los arcos:
(1,2),(2,3),(3,6),(6,4),(4,5),(5,1)
tiene un costo de:

3 + 6 + 20 + 15 + 5 + 7 = 56

	1	2	3	4	5	6
1	0	3	10	11	7	25
2	3	0	6	12	8	26
3	10	6	0	9	4	20
4	11	12	9	0	5	15
5	7	8	4	5	0	18
6	25	26	20	15	18	0

- Que significa esto?

Problema del viajante

b) Otro posibilidad para una metodología greedy es ordenar de menor a mayor las aristas y *elegir en cada iteración el arco de menor costo aún no considerado* que cumpla las dos condiciones siguientes:

(i) no forme un ciclo con los arcos ya seleccionados, excepto en la última iteración, que es donde completa el viaje;

(ii) no sea el tercer arco que incide en un mismo vértice de entre los ya elegidos.

Problema del viajante

Ejemplo1: El grafo *g*1 es un ejemplo para el cual el algoritmo encuentra la solución óptima, lo mismo que ocurre con la estrategia anterior.

Sin embargo, este algoritmo no encuentra la solución óptima en todos los casos, como ocurre por ejemplo2.

	1	2	3	4	5	6
1	0	3	10	11	7	25
2	3	0	6	12	8	26
3	10	6	0	9	4	20
4	11	12	9	0	5	15
5	7	8	4	5	0	18
6	25	26	20	15	18	0

Ejemplo2: Para el grafo *g2*, partiendo del vértice 1, el algoritmo va a ir eligiendo la secuencia de arcos:

(1,2),(3,5),(4,5),(2,3),(4,6),(1,6)

que da lugar al mismo ciclo que se obtenía antes, (1,2,3,5,4,6,1), de costo 58 y por tanto no óptimo.

Ninguna estrategia greedy da solución al problema planteado.

Problema del viajante

¿Podría aplicarse la técnica de Programación Dinámica al problema del viajante de comercio?

- En primer lugar, se plantea la solución del problema como una sucesión de decisiones que verifique el *principio de óptimo*.

- La idea va a consistir en construir una solución mediante la búsqueda sucesiva de recorridos mínimos de tamaño 1, 2, 3, etc.

Problema del viajante

- Representando el problema a través de un grafo:

  $g = (V,E)$, siendo A su matriz de adyacencia,

- Cada recorrido del viajante que parte del vértice v_1 estará formado por un arco (v_1,v_k) para algún vértice v_k perteneciente a $V–\{v_1\}$ y un camino de v_k al vértice v_1.

- Pero si el recorrido es óptimo también tiene que ser óptimo el camino de v_k al vértice v_1, pues si no lo fuese llegaríamos a una contradicción. Si no lo fuese y existiera otro camino mejor, incluyendo a éste en el recorrido original se obtendría un camino mejor que el óptimo, lo cual es imposible.

- Por tanto, se cumple el principio de óptimo.

Problema del viajante

- Se plantea entonces la relación en recurrencia. Para ello, se llama $D(v_i, S)$ a la longitud del camino mínimo que partiendo del vértice v_i pasa por todos los vértices del conjunto S y vuelve al vértice v_i.

- La solución al problema del viajante vendrá dada por : $D(v_1, V - \{v_1\})$

$$D(v_1, V - \{v_1\}) = \underset{2 \le k \le n}{Min} \{A(v_1, v_k) + D(v_k, V - \{v_1, v_k\})\}$$

- Generalizando para comenzar el recorrido desde cualquier vértice:

$$D(v_i, V) = \underset{i \notin V, j \in V}{Min} \{A(v_i, v_j) + D(v_j, V - v_j)\}$$

$$D(v_i, \{\}) = A(v_i, v_1) \quad para \quad 1 \le i \le n$$

Problema del viajante

Obsérvese la diferencia que existe entre la estrategia de este algoritmo y los algoritmos que resuelven el problema siguiendo la técnica greedy.

- En los algoritmos greedy se elige una de las posibles opciones en cada paso (la mejor), y una vez tomada o descartada, ya no vuelve a ser considerada nunca.

- Son algoritmos que no guardan “historia”, y por tanto no siempre funcionan.

- En la Programación Dinámica la solución al problema se va construyendo a partir de las soluciones óptimas para problemas más pequeños, que ya están guardadas.

Problema del viajante

- El diseño con programación dinámica tiene un serio inconveniente: su implementación necesita una estructura de datos que permita reutilizar los cálculos.

- Esa estructura debe contener las muchas soluciones intermedias necesarias para el cálculo de $D(v_1, V-\{v_1\})$.

- En este caso la tabla debe tener n filas, y 2^n columnas, porque éste es el cardinal de las partes del conjunto V, que son todas las posibilidades que puede tomar el segundo parámetro de D en su definición.

- Por lo tanto: ***existe una solución al problema del viajante utilizando Programación Dinámica***, pero no es eficiente ni simple su implementación.

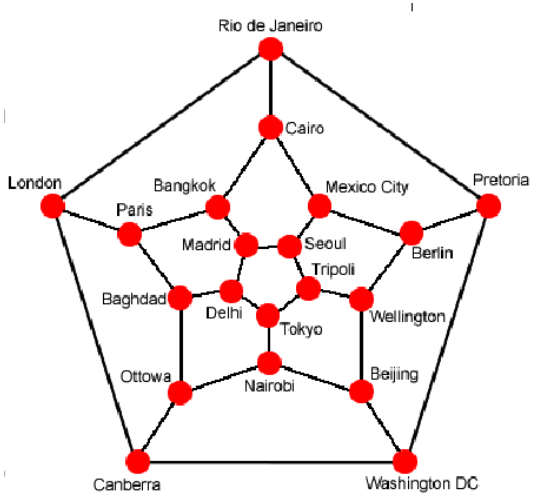
- La solución clásica es usando Vuelta Atrás y se presentará más adelante.

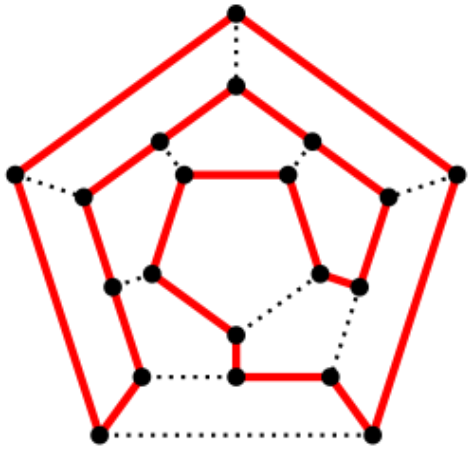
Ciclo Hamiltoniano

Un ***ciclo hamiltoniano*** es un ciclo simple que contiene todos los vértices del grafo en consideración.

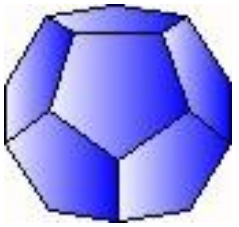
El nombre de "ciclo hamiltoniano" proviene de un juego de ingenio conocido como " La Vuelta al Mundo " y producido comercialmente a mediados del siglo XIX por Sir William Rowan Hamilton (1805-1865), famoso fisicomatemático irlandés.

El juego consistía básicamente en la determinación de un viaje cerrado que incluía todas las ciudades una sola vez, en un dodecaedro regular (12 caras), cuyos vértices (20) representaban ciudades importantes del mundo y sus aristas (30) eran las rutas que comunicaban dichas ciudades.





Ciclo Hamiltoniano



- Dado un grafo conexo, se llama *Ciclo Hamiltoniano* a aquel ciclo que visita exactamente una vez cada vértice del grafo y vuelve al punto de partida.

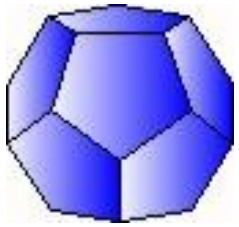
- **Problema:** detectar la presencia de ciclos Hamiltonianos en un grafo dado.

- **Solución usando *técnica de vuelta atrás (backtracking)***

- Sean los vértices del grafo numerados desde 1 hasta n, la solución al problema puede expresarse como un vector de n componentes de valores $X = (x_1, x_2, ..., x_n)$, donde x_i representa el i-ésimo vértice del ciclo Hamiltoniano.

- El algoritmo que resuelve el problema trabaja por etapas, decidiendo en cada etapa qué vértice del grafo de los aún no considerados puede formar parte del ciclo.

Ciclo Hamiltoniano

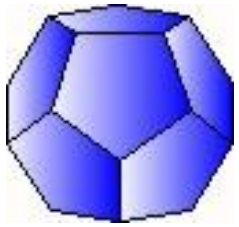


```
// Programa principal

// variables y constantes:
n = ...                                  //numero de vertices
X ϵ vector (1..n) de enteros             // solucion
A ϵ matriz (1..n,1..n) de BOOLEAN        // matriz del grafo
existe ϵ BOOL                            //existe ciclo hamiltoniano

// inicializa las variables y hace la invocación inicial:
    existe←FALSE
    X(1)←1
    Hamiltoniano1(X, 2, existe)
    Si existe entonces
        Escribir(X)          // solucion encontrada
    sino
        Escribir("no se encontró solución")
Fin
```

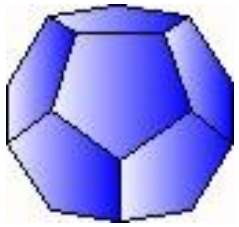
Ciclo Hamiltoniano



```
ALGORITMO Hamiltoniano1(X,k, existe) : vector x entero → BOOLEAN
// comprueba si existe un ciclo Hamiltoniano
        NuevoVertice(k,X)
        Si  X(k) = 0 entonces
                retorna  // sin solucion
        Sino
                Si  k = n entonces
                        existe←TRUE
                        retorna // con solución en X
                 Sino
                        Hamiltoniano1(X, k+1,existe)
Fin
```

El algoritmo ***NuevoVertice*** es el que busca el siguiente vértice libre que pueda formar parte del ciclo y lo almacena en el vector solución X en la posición k.

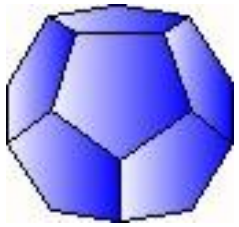
Ciclo Hamiltoniano



```
ALGORITMO NuevoVertice(k, X):entero → vector
        X(k)←(X(k-1)+1) MOD (n+1)
        Si X(k)=0 entonces
                retorna                         // sin solución
        Sino Si A(X(k-1),X(k)) entonces
                j←1
                sigue← verdadero
                Mientras (j≤k-1) AND sigue hacer
                        sigue←(X(j)≠X(k))
                        j←j+1
                Si (j=k)AND((k<n)OR((k=n)AND(A(X(n),1)))) entonces
                        Retorna                 // solución
        NuevoVertice(k, X)                      // sigue buscando
Fin
```

El algoritmo termina cuando encuentra un ciclo o bien cuando ha analizado todas las posibilidades y no encuentra solución.

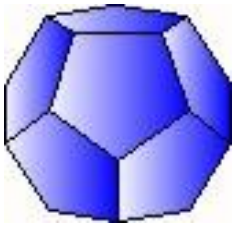
Ciclo Hamiltoniano



Se puede plantear también una modificación al algoritmo para que encuentre todos los ciclos Hamiltonianos si es que hubiera más de uno. La modificación en este caso es simple:

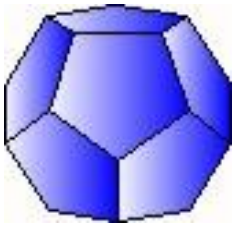
```
ALGORITMO Hamiltoniano2(X, k) : vector x entero → solucion
// determina todos los ciclos Hamiltonianos
        NuevoVertice(k,X)
        Si  X(k) = 0 entonces
                retorna                          // sin solución
        Sino
                Si  k = n entonces
                        Escribir(X)              // la solución es X
                Sino
                        Hamiltoniano2(X, k+1)    // otra solución
Fin
```

Ciclo Hamiltoniano



- Como saber si un dado grafo G admite un ciclo hamiltoniano?
- No se conocen condiciones necesarias y suficientes para la existencia de circuitos hamiltonianos.
- Si se conocen teoremas que dan condiciones suficientes para la existencia de circuitos hamiltonianos.

Ciclo Hamiltoniano



Condiciones suficientes para la existencia de circuitos hamiltonianos:

Teorema de DIRAC (Gabriel A. Dirac en 1952)

Sea *G* un grafo simple con *n* vértices con $n \geq 3$ tal que todos los vértices de *G* tienen grado mayor o igual que $n/2$.

Entonces, *G* contiene un circuito hamiltoniano.

Teorema de ORE (Oystein Ore en 1960)

Sea *G* un grafo simple con *n* vértices con $n \geq 3$ tal que:
grado(*u*)+g*rado*(*v*) $\geq n$ para cada par de vértices no adyacentes *u* y *v* de *G*.

Entonces, *G* contiene un circuito hamiltoniano.

Problema del viajante

- Generalizar el algoritmo del ciclo hamiltoniano para tratar con grafos ponderados.

- El problema es ahora de optimización: diseñar un algoritmo que, dado un grafo ponderado con pesos positivos, encuentre el ciclo Hamiltoniano de costo mínimo.

- En este caso hay que encontrar todos los ciclos Hamiltonianos, siendo necesario calcular el costo de cada solución y actualizar para obtener la solución óptima (el ciclo con mínimo costo).

- Este problema coincide con el conocido problema del viajante cuya solución mediante Vuelta Atrás basada en el algoritmo anterior se presenta a continuación.

```
ALGORITMO Viajante(X, k, XMIN) : vector x entero → solucion
// calcula el ciclo Hamiltoniano de minimo costo
        OtroVertice(k,X)
        Si  X(k) = 0 entonces                  // sin solución
                retorna
        Sino Si  k = n entonces                // actualizar costo de solución
                        costo←CalcularCosto(X)
                        Si costo<minimo entonces
                                minimo←costo
                                XMIN←X
                Sino
                        Viajante(X, k+1,XMIN)  // otra solución
Fin
```

- La función CalcularCosto obtiene la suma de los elementos del vector solución construido.
- Se hace uso de dos variables para almacenar el costo mínimo y el camino:

    minimo Є entero, que almacena el costo mínimo actual

    XMIN Є SOLUCION, que almacena la solución de costo min.

Problema del viajante

El algoritmo OtroVertice trabaja con un grafo ponderado, y por tanto los elementos de su matriz de adyacencia *A* serán enteros no negativos:

```
ALGORITMO OtroVertice(k, X):entero → vector
    X(k)←(X(k-1)+1) MOD (n+1)
    Si X(k)=0 entonces
        retorna
    Sino Si A(X(k-1),X(k)) ≠ ∞ entonces
        j←1
        sigue← verdadero
        Mientras (j≤k-1) AND sigue hacer
            sigue←(X(j)≠X(k))
            j←j+1
        cierraciclo←A(X(n),1)≠ ∞
        Si (j=k) AND ((k<n) OR ((k=n) AND cierraciclo)) entonces
            retorna
    OtroVertice(k, X)
Fin
```

Problema del viajante

El algoritmo Viajante debe ser invocado desde el programa principal después de inicializar la variable *minimo* y el primer elemento del vector *X* con el vértice desde donde parte el ciclo.

```
...
minimo← ∞
X(1)←1
Viajante(X,2,XMIN)
Si minimo < ∞ entonces
        Escribir(XMIN)
...
```

Problema del viajante

- El problema del viajante admite numerosas estrategias de *ramificación y poda.*

- Se usa una representación del grafo en donde los vértices están numerados consecutivamente comenzando por 1, y los arcos vienen definidos mediante una matriz de adyacencia, no necesariamente simétrica, aunque sí de elementos no negativos.

- Se comienza con la construcción del árbol de expansión para el problema.

- En primer lugar, se tiene que plantear la solución como una *secuencia de decisiones*, una en cada paso o etapa.

Problema del viajante

- La solución está formada por un vector que indica el orden en el que deberán ser visitados los vértices.
- Cada elemento del vector contiene un número entre 1 y *N*, siendo *N* el número de vértices del grafo que define el problema.
- Inicialmente el vector solución está compuesto por un solo elemento, el 1 (que es el vértice origen).
- En cada paso *k* se toma la decisión de cual vértice se incluye en el recorrido.
- Los valores que puede tomar el elemento en posición *k* del vector ($1 \leq k \leq N$) están comprendidos entre 1 y *N* pero sin poder repetirse, esto es, no puede haber dos elementos iguales en el vector.
- Por tanto, cada nodo podrá generar hasta $N-k$ hijos.
- Este mecanismo es el que va construyendo el árbol de expansión para el problema.

Problema del viajante

Información que debe contener cada uno de los nodos:

Se tiene que conseguir que cada nodo sea "autónomo", esto es, que cada uno contenga toda la información relevante para poder realizar los procesos de bifurcación, poda y reconstrucción de la solución encontrada hasta ese momento.

Cada ***nodo*** tiene que llevar almacenada información acerca de:

- el nivel en donde se encuentra
- el vector solución construido hasta el momento
- información para realizar la poda:
  - el costo acumulado hasta ese momento
  - la *matriz de costos reducida*

Problema del viajante

Reducción de matrices:

- Se dice que una fila (columna) de una matriz está *reducida* si contiene al menos un elemento cero, y el resto de los elementos son no negativos.

- Una matriz se dice *reducida* si y sólo si todas sus filas y columnas están reducidas.

ALGORITMO

- Dada una matriz de adyacencia de un grafo ponderado se puede obtener su matriz reducida calculando los mínimos de cada una de las filas y restándoselos a los elementos de esas filas, haciendo después lo mismo con las columnas.

Problema del viajante

Por ejemplo, dada la matriz de adyacencia:

	1	2	3	4	5
1	∞	15	7	4	20
2	1	∞	16	6	5
3	8	20	∞	4	10
4	4	7	14	∞	3
5	10	35	15	4	∞

restar respectivamente 4, 1, 4, 3 y 4 a cada fila:

	1	2	3	4	5	-
1	∞	11	3	0	16	4
2	0	∞	15	5	4	1
3	4	16	∞	0	6	4
4	1	4	11	∞	0	3
5	6	31	11	0	∞	4

restar 4 y 3 a las columnas 2 y 3 que no tienen 0, obteniendo la *matriz reducida*:

	1	2	3	4	5
1	∞	7	0	0	16
2	0	∞	12	5	4
3	4	12	∞	0	6
4	1	0	8	∞	0
5	6	27	8	0	∞
-	0	4	3	0	0

Problema del viajante

- En total se ha restado un valor de 23 = (4 + 1 + 4 + 3 + 4 + 4 + 3), que es lo que se denomina el *costo* de la matriz.
- Respecto a la interpretación del *costo*, restando una cantidad *t* a una fila o a una columna se reduce en esa cantidad el costo de los recorridos del grafo.
- Por tanto, un camino mínimo lo seguirá siendo tras una operación de sustracción de filas o columnas.
- En cuanto a la cantidad total restada al reducir una matriz, ésta será una *cota inferior del costo total* de sus recorridos.
- Para el ejemplo anterior se ha obtenido que 23 es una cota inferior para la solución al problema del viajante.

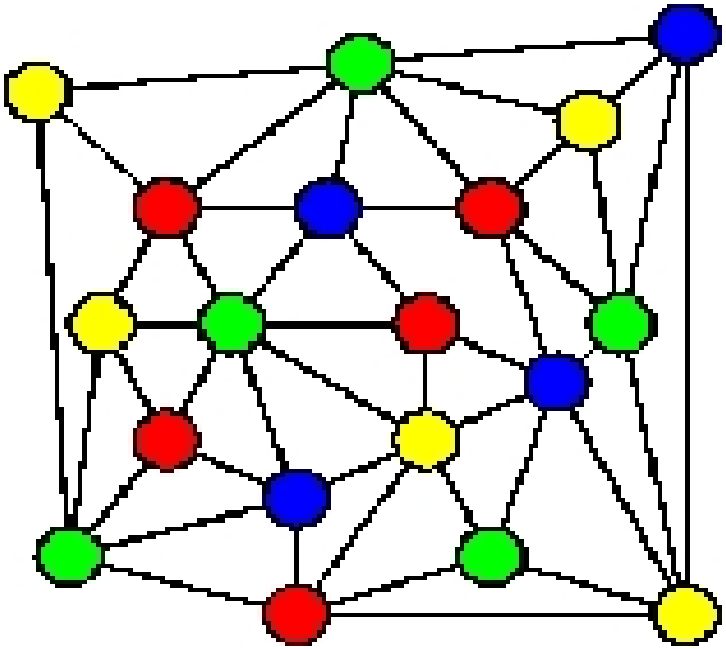
Problema del viajante

- Esto es lo se usa como función de costo para realizar la poda.
- A cada nodo se le asocia una matriz reducida y un costo acumulado.
- Entonces cada **nodo** debe llevar lo siguiente:
  - Costo acumulado
  - Matriz de adyacencia reducida asociada al nodo
  - k, el nivel del árbol
  - S: la solución armada hasta ese nodo.
- La búsqueda de *buenas funciones de costo* para reducir la exploración del árbol es una de las partes más delicadas e importantes de la solución, porque incide directamente en el tamaño del árbol generado y por lo tanto en la complejidad de tiempo y de espacio del algoritmo que obtiene la solución.-

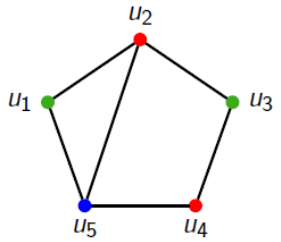
Coloreado

Sea *G* un grafo no dirigido, un *coloreado* de G es una asignación de colores a los nodos del grafo de tal manera que si dos nodos están conectados por una arista se le asigna diferentes colores.

Por ej.:



Coloreado



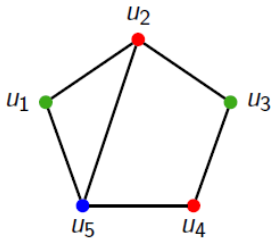
Un coloreo de los nodos de un grafo G (V,E) es una asignación f:

f : V →C, tal que f (v) ≠ f (u) para toda arista (u,v) ϵ E.

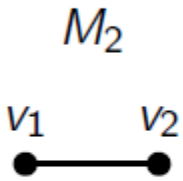
Los elementos de C son llamados colores. Generalmente los colores son enteros positivos.

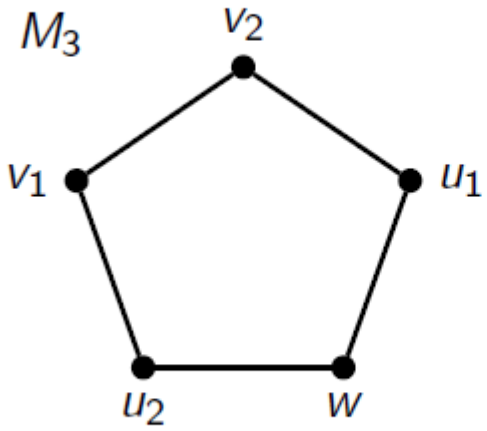
- Para todo entero positivo k, un k-coloreo de G es un coloreo de los nodos de G que usa exactamente k colores diferentes.
- Un grafo G se dice k-coloreable si existe un k-coloreo de G.
- El numero cromático de G: χ (G) , es el **menor numero de colores** necesarios para colorear los nodos de G y se denota con la letra griega χ (ji o chi, correspondiente a la ch en el alfabeto latino).
- Un grafo G se dice k-cromático si χ (G) = k.

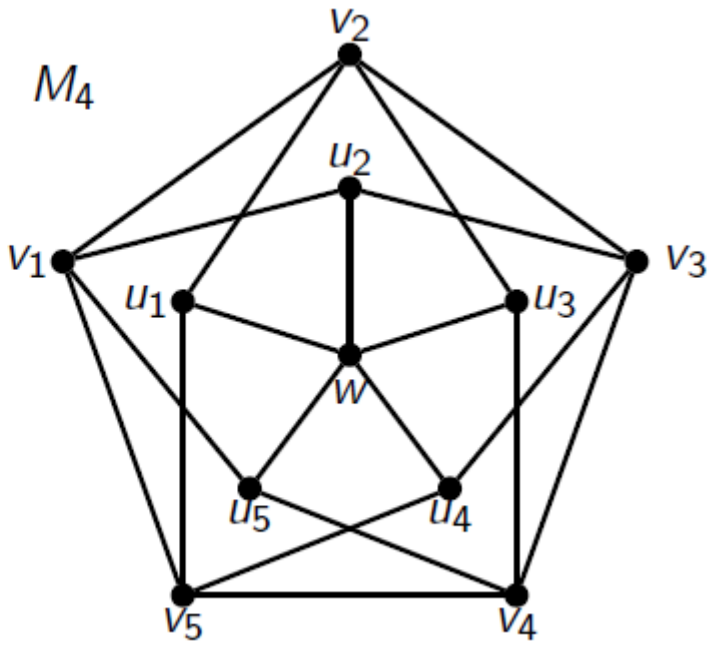
Algoritmos de Coloreado



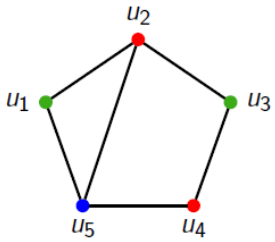
Ejemplo: Cuál es el número cromático de cada grafo?







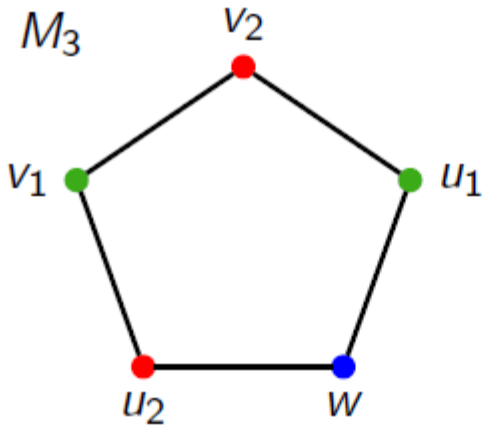
Algoritmos de Coloreado



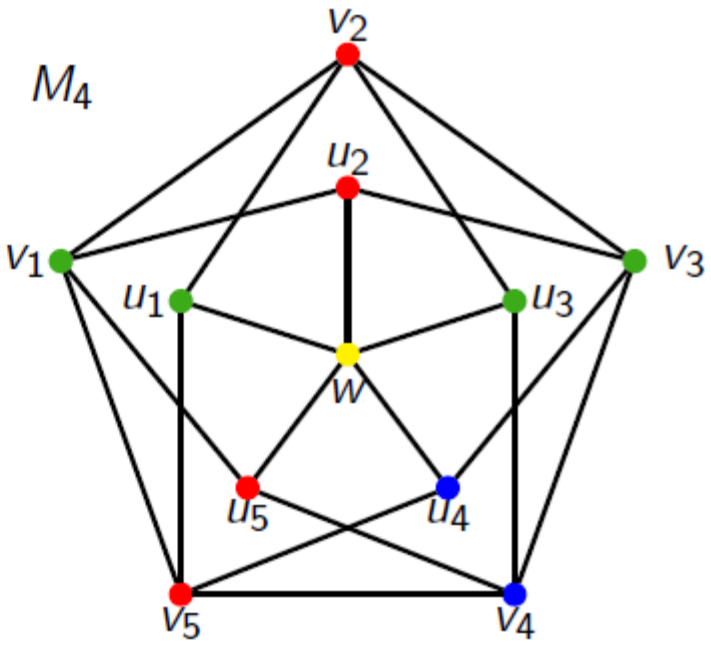
El número cromático de cada grafo es: $\chi(M_i) = i$

M_2

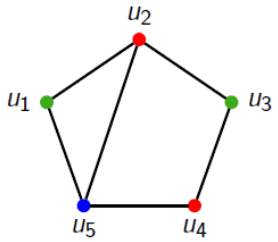
M_3



M_4



Problemas de Coloreado



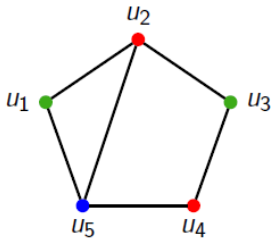
Dados los cuatro problemas siguientes:

- Es un grafo G 3-coloreable
- Es un grafo G k-coloreable, k dado
- Hallar el número cromático de G
- Coloreado óptimo de G (con su número cromático)

Hoy en día existen muchísimos desarrollos de algoritmos que los resuelven, especialmente heurísticas para coloreo de grafos.

Pero…No se conoce ningún algoritmo capaz de dar una solución óptima en tiempo polinomial. No tienen algoritmos polinómicos que los resuelvan. Son problemas NP completos.

Algoritmos de Coloreado



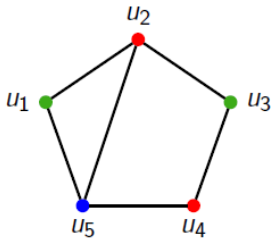
Existen varias heurísticas de coloreado de los vértices de un grafo basadas en la intuición de que un vértice de grado alto será más difícil de colorear que uno de grado bajo.

Las más sencillas son las estrategias greedy:

Se ordenan los vértices. Se van asignando en orden un color a cada vértice considerando que ese color no haya sido asignado a sus vértices adyacentes.

Estos algoritmos no garantizan el mínimo número de colores si no se realiza la asignación en el orden adecuado. Como hay n! posibles ordenes requiere un orden exponencial probarlos a todos.

Algoritmos de Coloreado



Un primer esquema del algoritmo secuencial de coloreado:

Algoritmo Color

Entrada: G , grafo con n nodos(v1; …,vn)

Salida: f , función que a cada nodo le corresponde un color

f (v1) ← 1

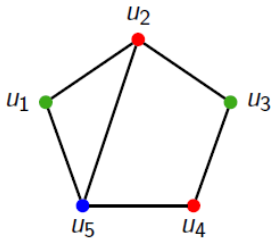
Para i = 2, n hacer

f (vi) = mínimo {h / h≥1 y f(vj)≠h para todo (vj, vi) ϵ E, 1≤j≤i-1}

Escribir (coloreado definido por f)

Fin

Algoritmos de Coloreado



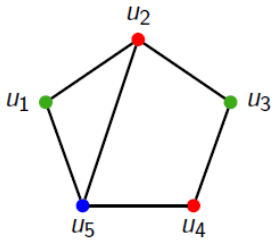
Algoritmo con la técnica vuelta atrás para colorear el grafo con **m** colores:

```
n = ...                                  //número de vertices
m = ...                                  //número de colores
X Є vector (1..n) de enteros             // solución con color asignado a c/vértice
A Є matriz (1..n,1..n) de bool           // matriz del grafo
existe Є BOOL                            // existe o no existe coloreado
```

El programa debe definir las variables y los parámetros de su invocación inicial:

```
// Programa principal ...
    existe←falso
    Color1 (1,X,existe)
    Si existe entonces
            Escribir(X)          // solución encontrada
    sino
            Escribir("no se encontró solución")
    Fin
```

Algoritmos de Coloreado

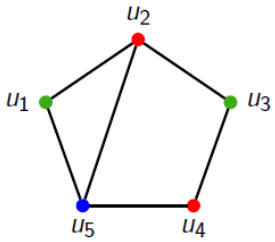


ALGORITMO Color1(k,X,existe)

// busca una solución

```
X(k)←0
Repetir
    X(k)← X(k)+1
    Si Factible(k,X) entonces
        Si k<n entonces
            Color1(k+1,X,existe)
        Sino
            existe←verdadero
Hasta que (existe) OR (X(k)=m)
Fin
```

Algoritmos de Coloreado



Funcion Factible (k,X) : entero x Solucion → bool

// control de la asignacion de color

Para j=1 , k-1 hacer

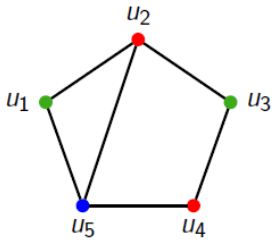
    Si (A(k,j) AND X(k)=X(j)) entonces // son adyacentes y tiene mismo color

        retorna falso //se repite color

retorna verdadero

Fin

Algoritmos de Coloreado



Modificando el algoritmo se puede obtener todas las maneras de colorear el grafo:

```
ALGORITMO Color2(k,X,existe)   // busca todas las soluciones
X(k)←0
Repetir
    X(k)← X(k)+1
    Si Factible(k,X) entonces
        Si k<n entonces  Color2(k+1,X,existe)
        Sino  Escribir (X)   //una solución
Hasta que X(k)=m
Fin
```

Con otra modificación se puede calcular el mínimo número de colores.

Aplicaciones de coloración de vértices

Problema de **Coloreado de un mapa**:

Son suficientes 4 colores para colorear bien cualquier mapa plano, sin importar la forma o número de países que éste tenga?

En lenguaje de Teoría de Grafos, se expresa del modo siguiente:

Es posible asignar una 4–coloración al grafo asociado a un mapa cualquiera?

Conjetura. Todo mapa se puede colorear usando 4 colores, de modo tal que regiones adyacentes usen colores distintos.

- Las regiones deben ser contiguas.
- Dos regiones no se consideran adyacentes si solo se intersecan en un punto.

Aplicaciones de coloración de vértices

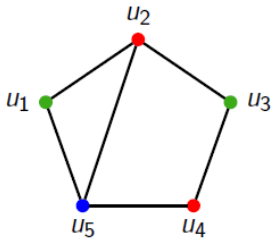
Conjetura de los 4 colores: Todo mapa se puede colorear usando 4 colores, de modo tal que regiones adyacentes usen colores distintos.

Esta conjetura fue planteada por el matemático alemán August Möbius en una conferencia que dictó en 1840, sin una demostración válida.

Alfred Kempe (1849-1922) dio una demostración en 1879, pero Percy Heawood (186-1955) encontró en 1890 un error. Al mismo tiempo, demostró el teorema de los cinco colores.

Francis Guthrie (1831-1899) redescubrió la conjetura mientras coloreaba un mapa de Inglaterra, y su hermano la comunico a Augustus De Morgan (1806-1871).

Teorema de los cuatro colores



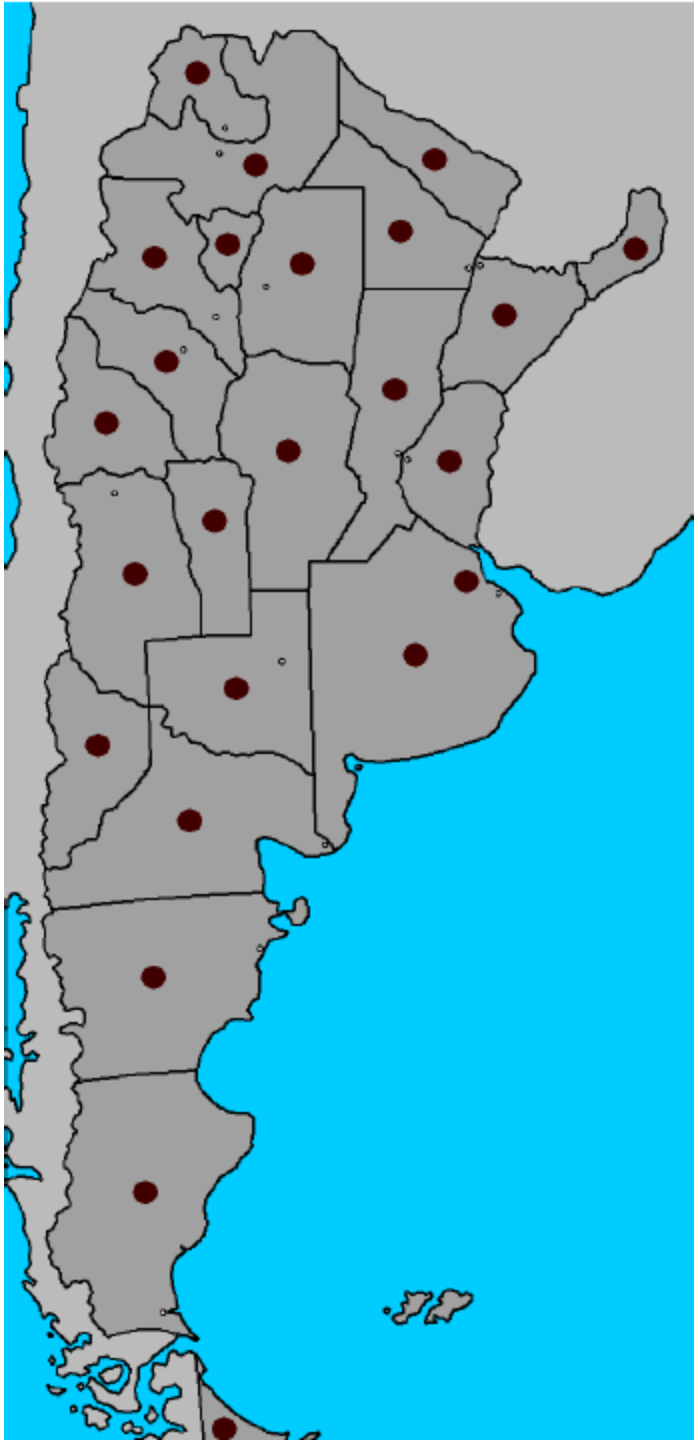
Pasó casi un siglo para enunciar el Teorema:

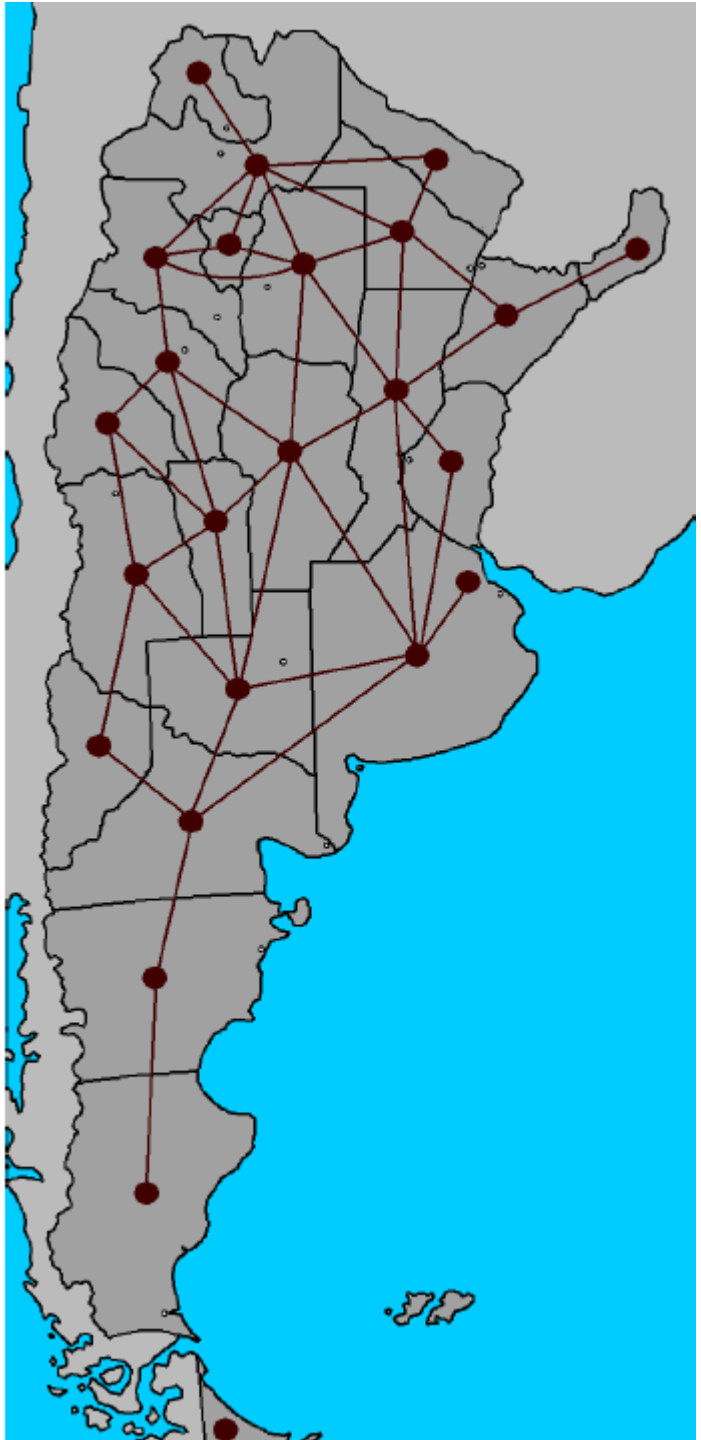
Teorema de los 4 colores: Si G es un grafo plano, entonces $\chi(G) \leq 4$

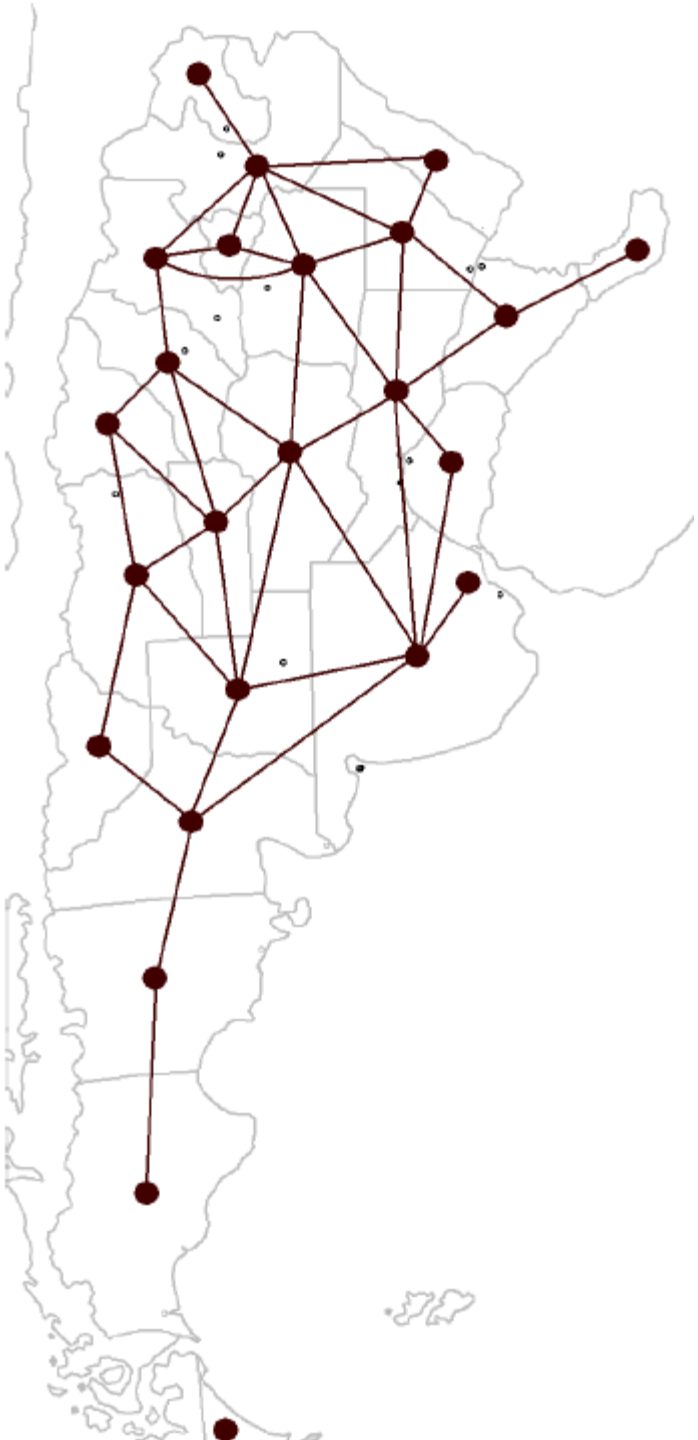
En 1976 Kenneth Appel y Wolfgang Hanken, lograron la primera demostración todavía cuestionada, haciendo uso de los avances de la rama de la matemática denominada topología y de una computadora (usaron 1200 horas de cálculo). (K. APPEL and W. HAKEN, Every planar map is four colorable, Contemp. Math., vol. 98, Amer. Math. Soc., Providence, RI, 1989)

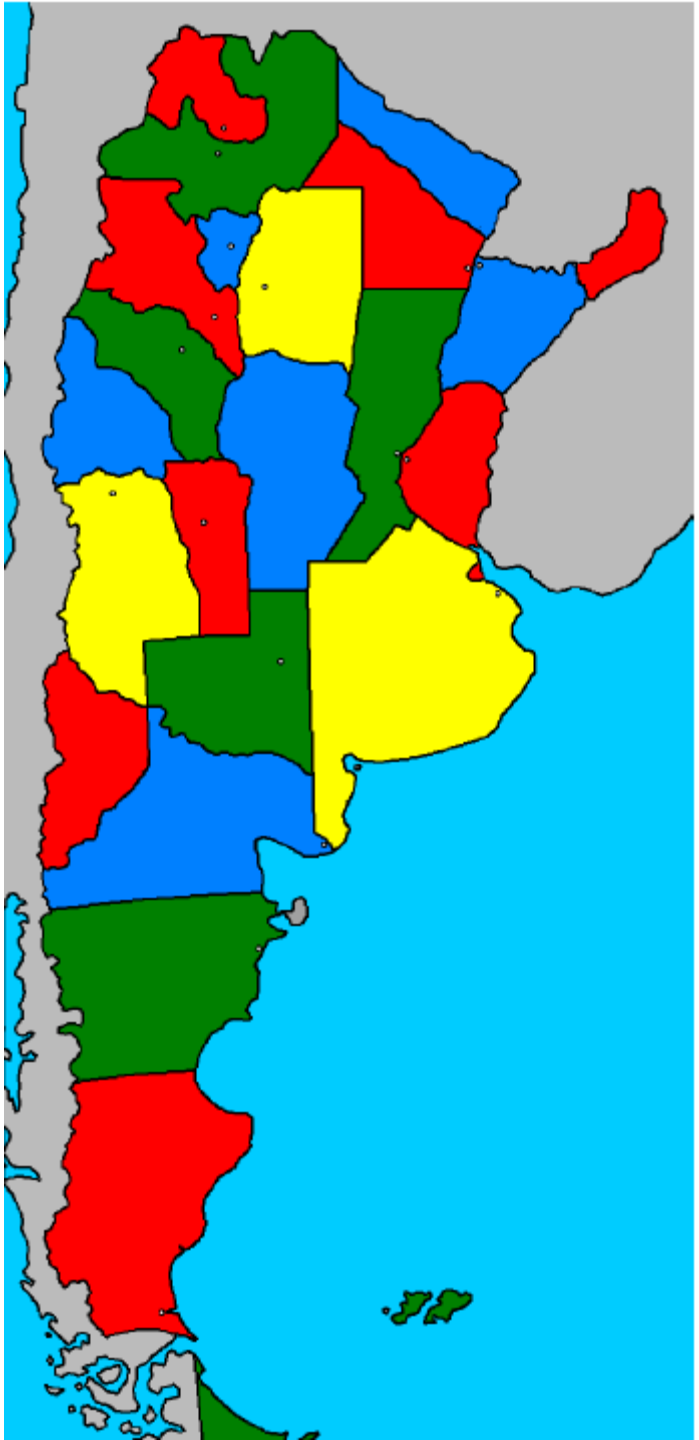
Estado actual: Algoritmo $O(n^2)$ para colorear un mapa con 4 colores (N. Robertson, D. Sanders, P. Seymour y R. Thomas, 1996).

En 2005, B. Werner y G. Gonthier formalizaron una prueba del teorema dentro del asistente de pruebas Coq. (Gonthier, Georges (2008), «Formal Proof--The Four-Color Theorem», Notices of the American Mathematical Society 55 (11): 1382-1393)









Aplicaciones de coloración de vértices

Asignación de frecuencias de radios:

Los vértices representan las estaciones de radio.

Dos emisoras son adyacentes (están unidas por un arco) si sus áreas de emisiones se solapan, lo que significa una interferencia si las emisoras están en la misma frecuencia.

Dos estaciones adyacentes deben tener entonces distintas frecuencias.

Es un problema de coloreo, donde las frecuencias son los colores, en el cual cada clase del mismo color contiene estaciones que no se solapan.

En este modelo el número cromático: $\chi(G)$ es el mínimo número de frecuencias que se requieren para no tener interferencia de emisoras.

Aplicaciones de coloración de vértices

Almacenamiento de productos químicos peligrosos

Los vértices representan los diferentes compuestos químicos que se tienen que almacenar en un laboratorio.

Dos productos son adyacentes (están unidos por un arco) si pueden explotar en el caso de que se almacenen juntos.

Es un problema de coloreo, donde los productos reciben distintos colores, en el cual cada clase del mismo color contiene solamente químicos que se puedan almacenar juntos.

En este modelo el número cromático: χ (G) es el mínimo número de lugares de almacenamiento que se requieren para no almacenar dos productos químicos de mezcla explosiva juntos.

Aplicaciones de coloración de vértices

Horarios de cursos en la Licenciatura en Informática

Los vértices representan las diferentes asignaturas de la licenciatura.

Dos materias son adyacentes (están unidas por un arco) si tiene al menos un alumno inscripto en ambas. De modo que no se pueden dictar en el mismo horario.

Es un problema de coloreo, donde las asignaturas con alumnos comunes deben tener distintos colores.

En este modelo el número cromático: χ (G) es el mínimo número de horarios diferentes de horarios que se requieren para que ningun alumno tenga conflicto de superposición de clases.-