Problemas NP-completos: coloreo de grafos, partición de grafos, viajero de comercio, el problema de morral

Complejidad de problemas

- ¿Cuál es el costo del “mejor” algoritmo (en tiempo de ejecución) para resolver un problema?
- Eso es el costo del problema.
- ¿Podemos conocer el costo de un problema sin conocer el “mejor” algoritmo?
- Si no conocemos el costo podemos acotarlo inferior y superiormente.

Complejidad de un algoritmo

- Todo algoritmo tiene una serie de características, **entre otras que requiere de recursos**, algo que es fundamental considerar a la hora de implementarlos en una maquina.

- Estos recursos son principalmente:
  - **El tiempo:** período transcurrido entre el inicio y la finalización del algoritmo.
  - **La memoria:** la cantidad (la medida varía según la máquina) que necesita el algoritmo para su ejecución.

Complejidad de un algoritmo

- **Los modelos de complejidad son normalmente usados para**
  - Mostrar que un algoritmo dado es optimo o
  - Determinar la complejidad mínima de un problema (establecer el limite superior del numero de operaciones necesarias para resolver un problema dado).
- **Complejidad Computacional:** considera globalmente todos los posibles algoritmos para resolver un problema planteado

Complejidad Computacional,

La complejidad de la ordenación

- Cual es el número mínimo de comparaciones necesarias para ordenar n elementos.

- Tomaremos en cuenta los algoritmos de ordenación basados en comparaciones. La pregunta en si seria:

¿ Cuál es el numero mínimo de comparaciones en cualquier algoritmo para ordenar n elementos por comparación?

Para ello se recurre a los árboles de decisión

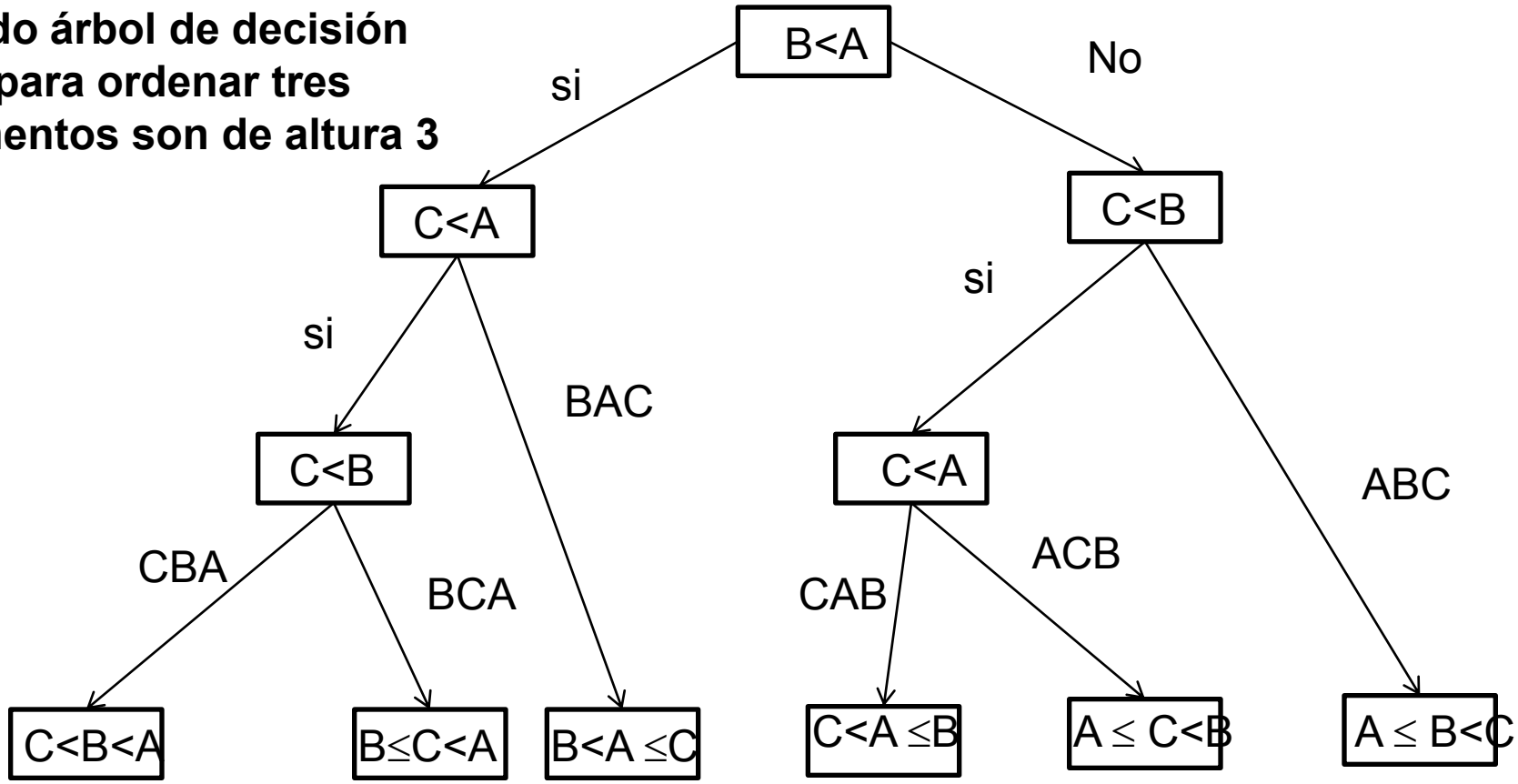
Complejidad Computacional,

Algoritmo para ordenar 3 elementos

```
Procedimiento ordenacioninsitu3(T[1..3])
    A <- T[1]; B <- T[2]; C <- T[3]
    si A < B entonces
        si B < C entonces { ya están ordenados }
        sino si A < C entonces T <- A, C, B
                sino T <- C, A, B
    sino si B < C entonces
                Si A < C entonces T <- B, A, C
                sino T <- B, C, A
        sino T <- C, B, A
```

Complejidad Computacional,

Todo árbol de decisión para ordenar tres elementos son de altura 3



cada árbol de decisión valido para ordenar n elementos debe contener al menos n! hojas, así como tener una altura que sea mínimo $[\lg_2 n!]$ y una altura media que sea como mínimo $\lg_2 n!$.

P y NP

- La clase de complejidad **P** corresponde a todos los problemas que pueden **ser resueltos en tiempo polinomial**.

- La clase de complejidad **NP** (viene de tiempo polinomial no-determinístico) corresponde a la clase de problemas cuya **solución puede ser verificada en tiempo polinomial**.

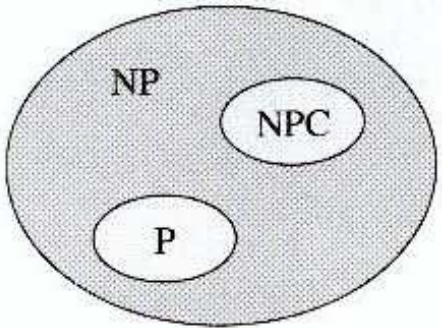
- Por ejemplo: Consideremos el problema del **ciclo Hamiltoniano**:
  - Un ciclo Hamiltoneano es aquél que **recorre todos los nodos de un grafo en un camino simple.**
  - ¿Tiene un grafo G un ciclo Hamiltoniano? Un posible candidato para **este problema puede ser verificado en tiempo polinomial**, este problema está en la categoría NP.
  - Los **problemas P son todos NP.**

¿P = NP?

- Se desconoce si estos dos conjuntos son equivalentes, pero muchos científicos creen que estas dos clases son distintas.
- Científicos creen que la clase NP incluye problemas que no están en P.
- Generalmente es más fácil verificar una posible solución que encontrar la solución de un problema.
- Esta pregunta lleva a la aparición de la clase NP-completo (NPC). Aquellos **problemas cuyo status es desconocido pero se cree no tienen soluciones polinomiales**.

Hasta ahora no se ha encontrado solución polinomial para ninguno de ellos.

- Se puede probar que: si un problema NP-completo tiene solución polinomial, todos la tendrían.



Problemas NP-completos,

P es la clase de problemas de decisión que se pueden resolver mediante un algoritmo de tiempo polinómico.

- Un algoritmo en tiempo polinómico es eficiente si existe un polinomio p(n) tal que el algoritmo puede resolver cualquier caso de tamaño n en un tiempo O(p(n))

  ⇒ Algoritmo de tiempo polinómico.

- O(nlgn), $O(n^2)$?

Problemas NP-completos

- Teoría de la NP-Completitud:
  - **Problemas sin algoritmo eficiente**, de dificultad intrínseca que en algunos casos aún no ha sido demostrada.
  - Se cree que algoritmos eficientes para ellos no existen.

Lo que sí es eficiente es la validación de una supuesta solución.

Ineficiencia e Intratabilidad

Problemas algorítmicos para los que no existe una solución satisfactoria

- Ejemplos
  - Torres de Hanoi
  - Puzzle del mono

Problemas NP_Completos

- Quizás es cuestión de esperar que los computadores sean más rápidos

- ¿Puede ser causa de nuestra incompetencia para idear buenos algoritmos?

- No tiene valor el esfuerzo, este problema es un problema específico, no es importante.

Problemas NP_Completos

- Existen cerca de 1000 problemas algorítmicos con características parecidas

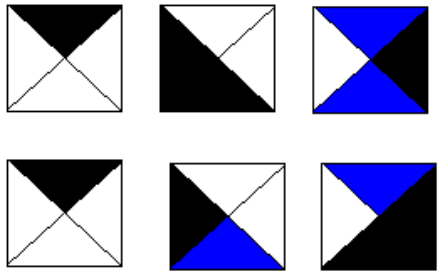
- Sus limites inferiores son lineales y sus limites superiores exponenciales.

NP conjunto de problemas que no se pueden resolver en tiempo polinómico por una máquina no determinista

El problema del embaldosado o dominó

- Tenemos baldosas cuadradas divididas en cuatro por dos diagonales, cada división de un color. Las baldosas tiene una orientación fija
- Dado un conjunto T de baldosas,

¿se puede embaldosar cualquier área de cualquier tamaño?



El problema del embaldosado o dominó

- Este tipo de razonamiento no puede ser mecanizado. No existe un algoritmo ni lo habrá para solucionar el problema del embaldosado.

- Para cualquier algoritmo que podamos diseñar habrá siempre un conjunto de entrada ***T*** para el que el algoritmo no termine o dé una respuesta errónea

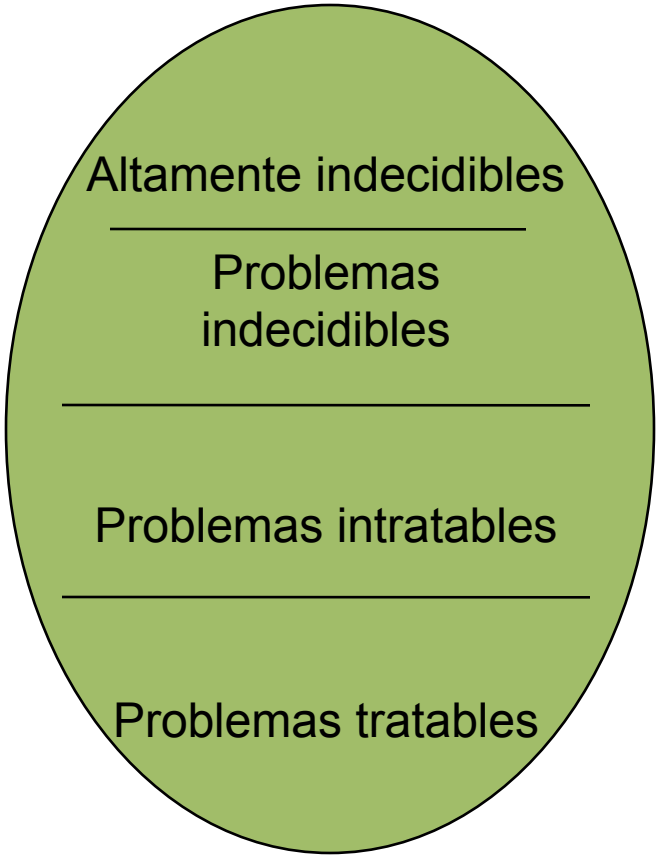
Verificación de programas

- Nos gustaría obtener un algoritmo que, dada la descripción de un problema y un algoritmo, indique si este algoritmo resuelve el problema o no
- Queremos que la respuesta sea **sí** si para cada entrada legal del algoritmo éste termina y da la respuesta correcta, y **no** si existen entradas para las que el algoritmo falla o da respuestas erróneas.

¿Se puede determinar algorítmicamente si dado un problema y un algoritmo, el algoritmo resuelve el problema?

problemas indecidibles

Niveles de comportamiento algorítmico



Problemas NP-completos,

- TSP
- HAM
- Mochila
- Ciclos eulerianos
- Buscaminas, Tetris
- Factorización
- Cobertura de vértices
- Emparejamiento de grafos
- Partición de grafos
- Coloreado de grafos

Problemas de optimización combinatoria

maximizan o minimizan funciones de varias variables sujeto a restricciones de integridad sobre todas o algunas variables.

Un pastor tiene que pasar un lobo, un conejo y una col de una orilla de un río a la otra orilla. Dispone de una barca en la que sólo caben él y una de las tres cosas anteriores. Si deja solos al conejo y al lobo, éste se come a aquél; si deja al conejo con la col, aquél se la come.

¿Cómo debe proceder para llevar las tres cosas a la orilla opuesta?

Problemas NP-Completos

- Problema Pandilla (clique):
  - Una pandilla en un grafo no dirigido G=(V,E) es un subconjunto de vértices que cumplen con tener conexión todos con todos. El tamaño de la pandilla es el número de vértices de él.
  - Problema:

    Existe un pandilla de tamaño k en el grafo?
  - Algoritmo simple: buscar todos los grupos de k nodos y probar para cada uno si están todos conectados.

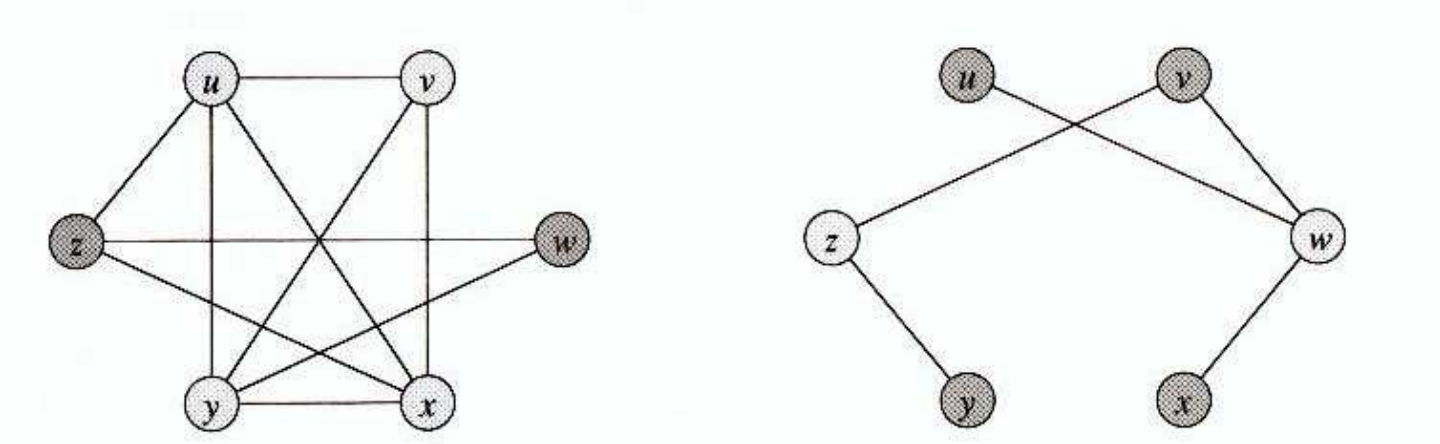
Problemas NP-Completos

- Problema de cubierta de vértices
  - La cubierta de vértices de un grafo no dirigido G=(V,E) es un subconjunto $V' \subseteq V$ tal que si $(u,v) \in E$, entonces $u \in V'$ o $v \in V'$.
  - Cada vértice cubre su arco incidente y la idea es buscar el conjunto que cubra todos los arcos.
  - Problema: encontrar la cubierta de vértices de tamaño mínimo.

Ejemplos de Pandilla (Clique) y cubierta de vértices

Pandilla (clique) para grafo de la izquierda ={u,v,x,y}

Cubierta de vértices para grafo de la derecha = {w,z}



Estos dos problemas son equivalentes. Si se resuelve uno se tiene el otro. La reducción de uno al otro conduce a los dos grafos de la figura. Un grafo es el complemento del otro. La cubierta del grafo complemento son los nodos que no están en la pandilla del grafo inicial.

Más Problemas NP-Completos

- Problema del ciclo hamiltoniano
  - Problema: Existe un ciclo simple en G que contenga a todos los vértices?
  - Vendedor viajero
  - Se tiene un grafo completo (todos los arcos) con costos asociados a cada arco. El vendedor desea hacer un “tour” (ciclo hamiltoniano) tal que el costo sea mínimo.

- El camino más largo
  - ¿ Cuál es el camino simple más largo que conecta dos nodos del grafo?

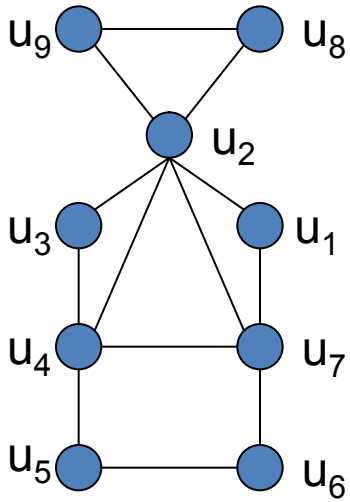
- Conjunto independiente
  - ¿Cuál es el conjunto más grande de vértices que no están conectados entre sí en el grafo?

- Isomorfismo de grafos
  - ¿Podemos hacer dos grafos iguales sólo renombrando los vértices?

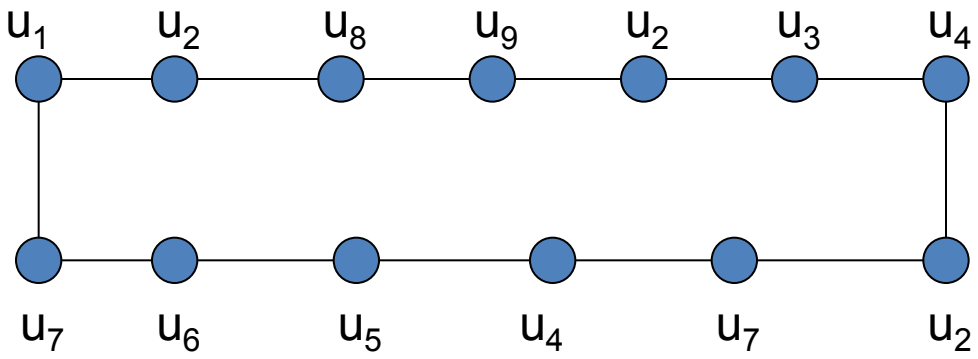
Ciclo eulerianos

Un ciclo es un recorrido con el mismo origen y final.

- Un ciclo euleriano de un multi - grafo conexo G es un ciclo que contiene todas las aristas de G.
- Un grafo G con un ciclo euleriano se denomina un grafo euleriano.



u_1– u_2–u_8-u_9-u_2-u_3-u_4-u_2-u_7-u_4-u_5-u_6-u_7-u_1

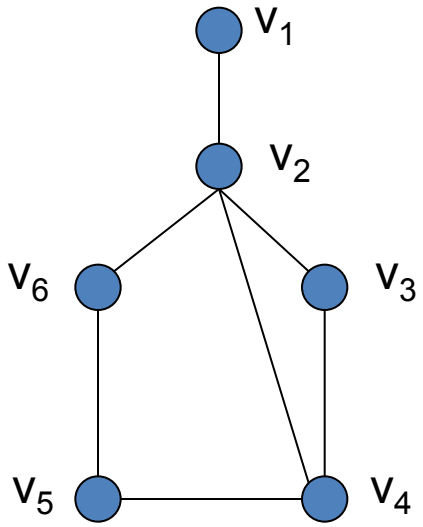


Otra posibilidad:

u_1-u_2-u_7-u_4-u_2-u_8-u_9-u_2-u_3-u_4-u_5-u_6-u_7-u_1

Ciclo eulerianos

Un recorrido es una cadena sin aristas repetidas (se pueden repetir vértices)



v_1—v_2-v_4-v_3-v_2-v_6-v_5-v_4

Otra opción:

v_1-v_2-v_3-v_4-v_2-v_6-v_5-v_4

Algoritmo ciclo eulerianos

- Sea H el grafo que se obtiene de G eliminando las aristas del ciclo T. Por el mismo razonamiento anterior, podemos encontrar un ciclo T' que comience en v.

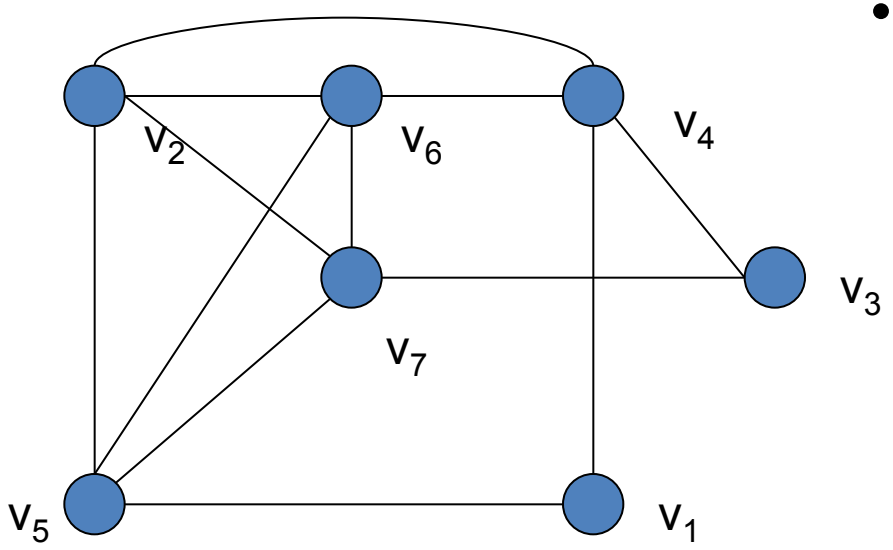
- Este ciclo se puede insertar en T de forma que tengamos un ciclo T_1 que contiene las aristas de ambos ciclos.

- Si este ciclo ha usado todas las aristas de G seria un ciclo euleriano.

- Si no vamos repitiendo el proceso hasta que en algún momento se hayan usado todas las aristas.

ciclo eulerianos

Ejemplo:

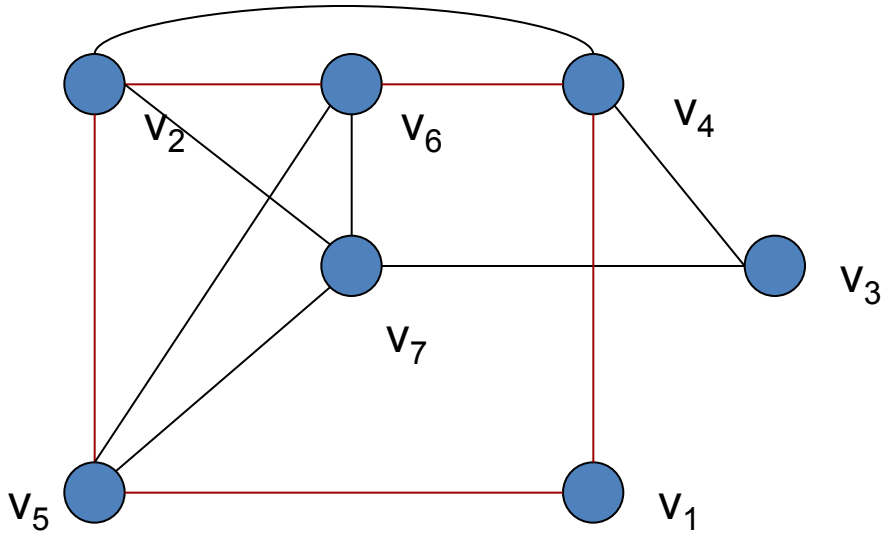


- Es conexo.
- Todos los vértices tienen grado par.

Entonces existe un ciclo euleriano.

ciclo eulerianos

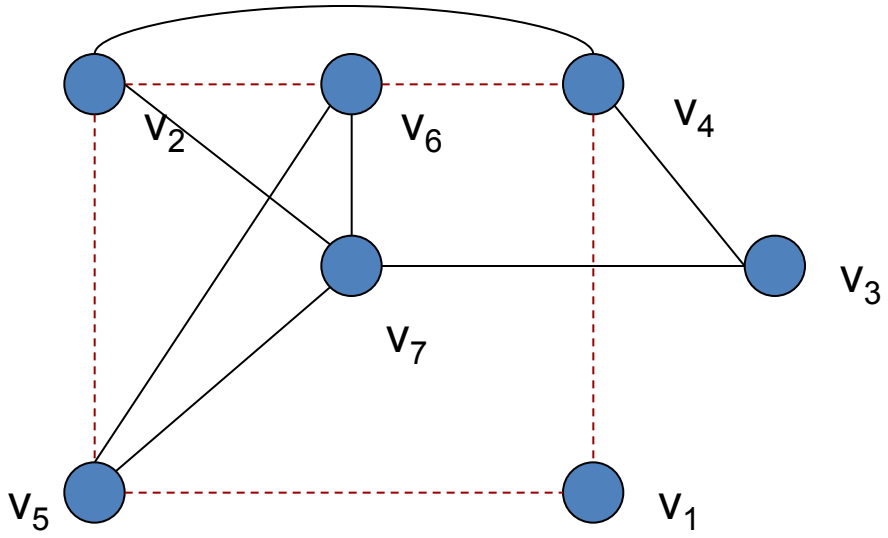
Ejemplo:



v_1-v_4-v_6-v_2-v_5-v_1

ciclo eulerianos

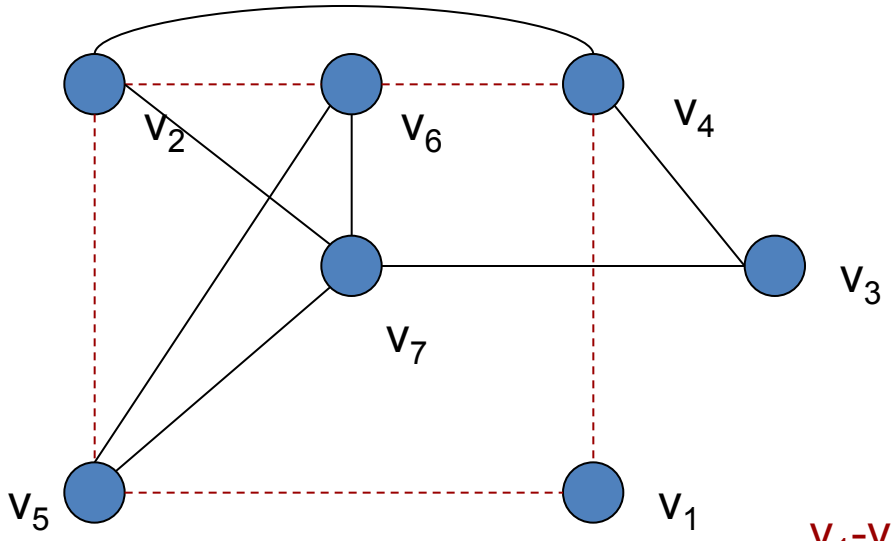
Ejemplo:



v_1-v_4-v_6-v_2-v_5-v_1

ciclo eulerianos

Ejemplo:



v_1-v_4-v_6-v_2-v_5-v_1

v_2-v_7-v_6-v_5-v_7-v_3-v_4-v_2

v_1-v_4-v_6-v_2-v_7-v_6-v_5-v_7-v_3-v_4-v_2- v_5-v_1

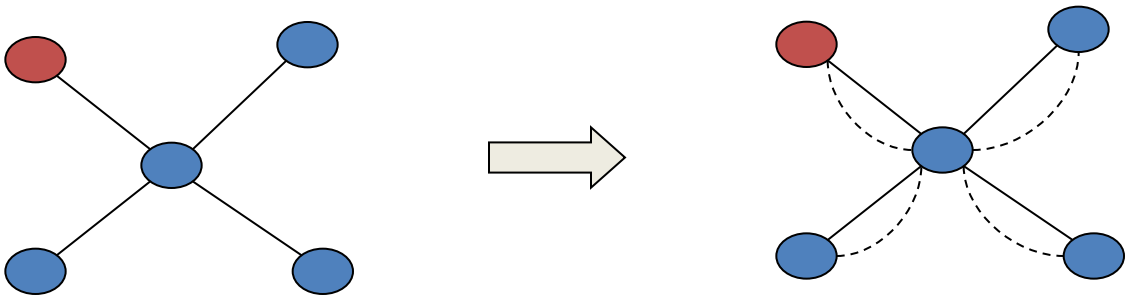
Aplicación de grafos eulerianos: El problema del cartero chino.

- Un cartero quiere repartir las cartas con el menor coste posible.
- Debe recorrer todas las calles que tiene asignadas
- Debe empezar en la oficina de correos y acabar en la oficina de correos.

Objetivo: Encontrar la cadena cerrada mas corta posible que recorre todas las aristas.

Esta cadena se denomina **cadena euleriana**.

Las calles pueden ser representadas por aristas.
Las intersecciones son los vértices.
El problema tiene solución porque si doblamos las aristas para crear un multigrafo
Todos los vértices tendrían grado par.

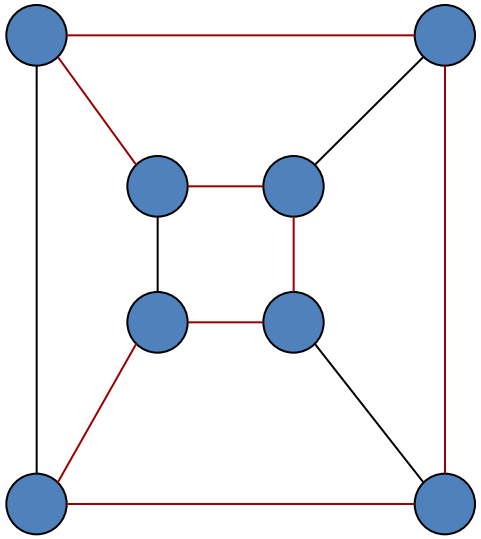


Grafos Hamiltonianos

Definición:

Un grafo G se dice que es Hamiltoniano si tiene un ciclo recubridor.

Es decir, un ciclo que pasa por cada vértice una sola vez.



Aplicación de los gráficos Hamiltonianos: El problema del Vendedor.

- Un vendedor quiere vender su producto en varias ciudades.
- Las ciudades están representadas por vértices.
- El costo para ir de una ciudad a otra por las aristas.
- Se puede suponer que el grafo es completo (no es obligatorio).

Problema del agente viajero (Problema de optimización combinatoria)

"Visitar todas las ciudades importantes de Vzla. por lo menos una vez, comenzando y terminando en Mérida"

- Requiere información sobre el espacio de estados:
  - Ubicación del agente,
  - Un registro de las ciudades visitadas.

- El test objetivo consiste en verificar si el agente está en Mérida y ha visitado todas las ciudades una sola vez.

- El objetivo es determinar cuál es el recorrido más corto,

  es un problema de complejidad NP.

El Agente Viajero

- ***Formalización General:*** Dada N ciudades, el viajero de comercio debe visitar cada ciudad una vez, teniendo en cuenta que el costo total del recorrido debe ser mínimo.

G=(N, A)

N={1, ...,n}: conjunto de *n* nodos (vértices)

A={a_{ij}}: matriz de adyacencia.

d_{ij} = { ∞ Si $a_{ij} = 0$

Lij Si $a_{ij} = 1$

Lij: distancia entre las ciudades i y j.

El Agente Viajero:

Formas de representar soluciones

- Suponiendo que las ciudades son numeradas desde 1 hasta n, una solución al problema puede expresarse a través de una matriz de estado *E*

e_{ij} = { 1 **Si la ciudad *j* fue la i^{esima} ciudad visitada**

0 En otro caso

- La matriz E permite definir un arreglo unidimensional V de dimensión *n*;

$v_j = i$ **Si la ciudad i fue la j^{esima} ciudad visitada**

El Agente Viajero

Función objetivo

$$F1 = \sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{n} L_{ik} e_{ij} e_{kj+1}$$

$$F2 = C\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\left(e_{ij} - n\right) + \sum_{i=1}^{n}\sum_{j=1}^{n}\left(e_{ij} - 1\right) + \sum_{j=1}^{n}\sum_{i=1}^{n}\left(e_{ij} - 1\right)\right)$$

FC = F1 + F2

C = n * Max(L_{ik}) factor de penalización.

Algoritmo que encuentra una solución de bajo coste (no necesariamente la mínima)

1.- p=1

2.- Selecciona cualquier vértice v de G. C_1=v v

Mientras p< n

- Encuentra un vértice v_p que no este en C_p y que u_pv_p sea mínimo siendo u_p pertenece a C_p
- Adjuntar v_p inmediatamente antes que u_p
- p=p+1

Algoritmo que encuentra una solución de bajo coste (no necesariamente la mínima)

Ejemplo:

$$\begin{pmatrix} 0 & 3 & 3 & 2 & 7 & 3 \\ 3 & 0 & 3 & 4 & 5 & 5 \\ 3 & 3 & 0 & 1 & 4 & 4 \\ 2 & 4 & 1 & 0 & 5 & 5 \\ 7 & 5 & 4 & 5 & 0 & 4 \\ 3 & 5 & 4 & 5 & 4 & 0 \end{pmatrix}$$

inicio: v_1 v_1

Primera iteración: v_1 v_4 v_1

Segunda iteración: v_1 v_3 v_4 v_1

Tercera iteración: v_1 v_3 v_4 v_2 v_1

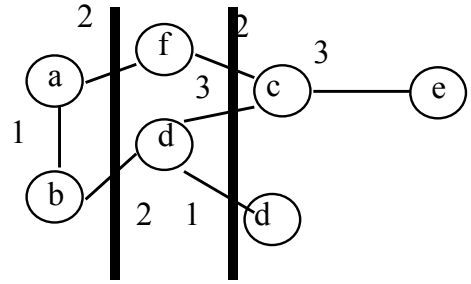
Cuarta iteración: v_1 v_5 v_3 v_4 v_2 v_6 v_1

Coste 26

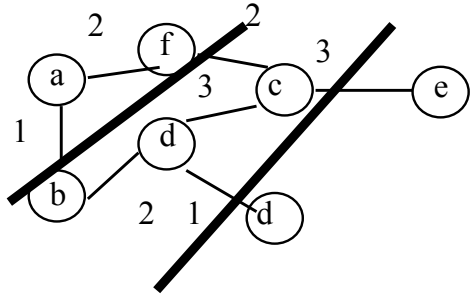
El mínimo que se podría encontrar inicializando con otro vértice seria 21

Problema de k-partición de grafos

- **Objetivo:** asignar los vértices tal que se minimice los arcos en diferentes sub-grafos y se repartan los vértices equitativamente entre los diferentes sub-grafos.



a)



b)

K=3

Problema de partición de grafos

- Definición de una Función de Costo General
  - *Costo de Comunicación*
  - *Costo por el Desequilibrio de la Carga*

$$\mathbf{F = A_1 C_C + A_2 C_D}$$

Objetivo: MIN(*F*)

Algoritmo de Kernighan-Lin

- Este algoritmo de partición, es un algoritmo determinista e iterativos, fue publicado en el año 1970.

- El objetivo de este algoritmo es dividir el grafo en dos partes (k=2), de modo que se minimice el número de arcos que conectan nodos pertenecientes a particiones distintas.

- El método comienza asignando aleatoriamente los nodos a dos grupos A y B. Posteriormente, se van intercambiando nodos entre las dos particiones.

- Estos intercambios, implican un incremento o decremento del número de corte, que se representa mediante la ganancia *g*.

Algoritmo de Kernighan-Lin

- La ganancia *g*, asociada al intercambio de dos nodos, se obtiene a partir del parámetro *D*. Dicho parámetro, se define como la diferencia entre el número de interconexiones de la celda que atraviesan la separación entre particiones, y el número de interconexiones que no atraviesan dicha frontera.

D(ai)= *inedge(ai)-outedge(ai)*

- ***outedge(ai),*** es el número de interconexiones entre la celda *ai* y otras celdas que no pertenecen a la misma partición.
- ***inedge(ai),*** es el número de interconexiones entre la celda *ai* y otras celdas pertenecientes a la misma partición.

Una ganancia *gi*, definida com $\begin{cases} g_i = D(a_i) + D(b_i) & si\ a_i\ y\ b_i\ no\ están conectadas \\ g_i = D(a_i) + D(b_i) + 2 & si\ a_i\ y\ b_i\ están conectadas \end{cases}$

Algoritmo de Kernighan-Lin

- El primer paso es seleccionar dos nodos, a_1 y b_1, pertenecientes, respectivamente, a los grupos A y B, de modo que la ganancia g_1 asociada al intercambio de estos dos elementos sea mínima.
- A continuación, se intercambian los nodos y, durante las iteraciones restantes, se impiden nuevos desplazamientos de las mismas.
- El siguiente paso es seleccionar dos nuevos nodos, a_2 y b_2, cuyo intercambio ha de suponer un incremento mínimo, g_2, del número de corte. Una vez realizado el intercambio, se bloquean nuevos desplazamientos de estos dos elementos.
- Esta serie de pasos continúa hasta que todos los elementos del ciclo han sido bloqueados.

La clave del algoritmo de Kernighan-Lin pues es encontrar aquel valor de *p* que minimiza:

$$\Sigma_{i=1,\, p}\, g_i$$

Aplicación PG

Asignación de tarea

- Las tareas son asignadas a los procesadores de manera de **maximizar la utilización de los procesadores y minimizar los costos comunicacionales**.

- Este problema no existe en sistemas uniprocesadores o computadores a memoria compartida.

- En el último caso, el **sistema operativo** es el que realiza la distribución de las tareas.

Aplicación PG

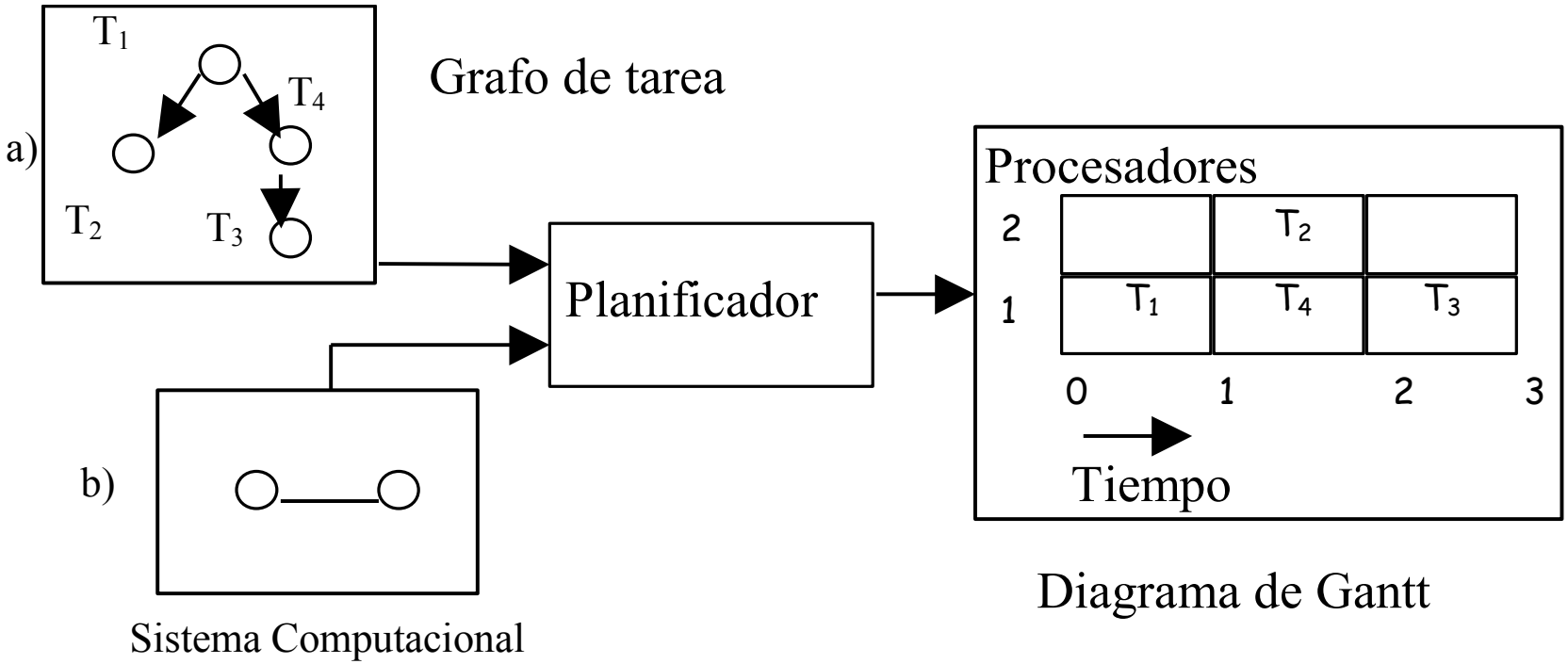
Asignación de tarea

- En general, este es un problema difícil y su objetivo es minimizar el tiempo total de ejecución de la aplicación a través de dos ideas:
  - Colocar tareas concurrentes en diferentes procesadores.
  - Colocar tareas que se comunican frecuentemente en el mismo procesador.

En general, este problema es NP-completo,

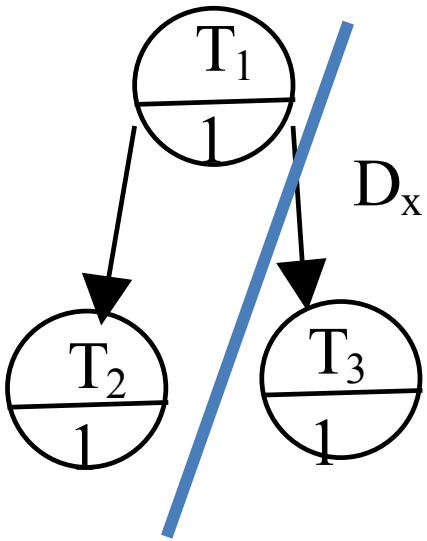
Aplicación PG

Asignación de tarea.



Aplicación PG

Asignación de tarea



Tiempo

1	T_1	
2	T_2	
3	T_3	

D_x $\quad$ $\overline{D_x}$

Tiempo

1	T_1	
2	T_2	T_3
3		

D_x $\quad$ $\overline{D_x}$

Aplicación PG

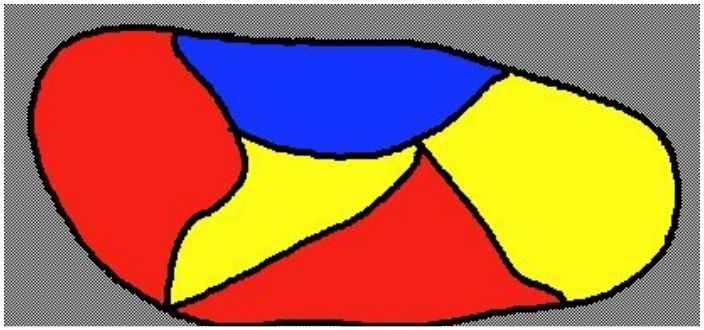
- Problema Replicación de archivos
- Problema descomposición Archivo
- Problema de asignación de archivos

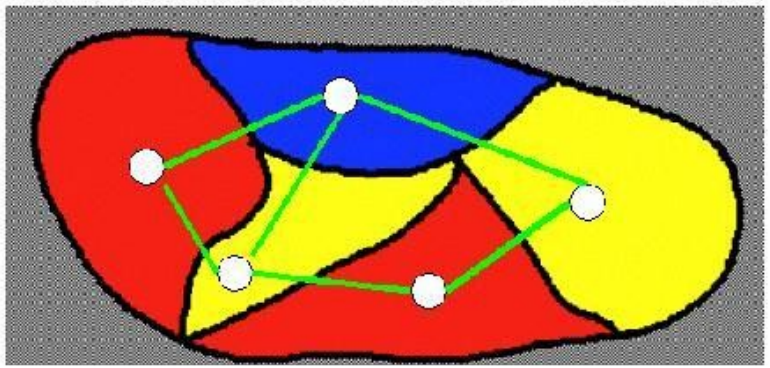
CF = Cs + Cu + Cq + Cli + Cno

Más Problemas NP-Completos

Coloración

Encontrar el mínimo conjunto de colores necesarios para colorear los arcos, tal que dos arcos con el mismo color no compartan un vértice



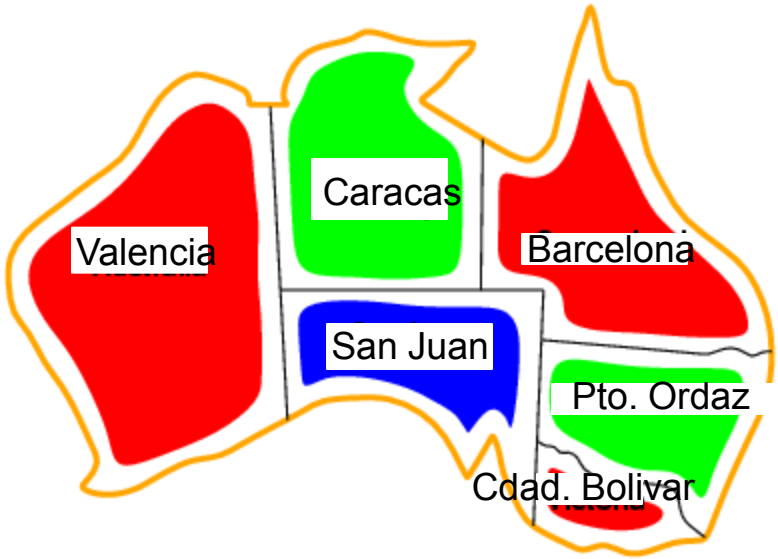


Ejemplo: Coloreado de Mapas

- Variables Valencia, Caracas, San Juan, ..
- Dominio D_i = {rojo, verde, azul}
- Restricciones: regiones adyacentes deben tener diferentes colores

- Ejm., Valencia ≠ Caracas,
  => (Valencia, Caracas) puede ser
  {(rojo,verde),(rojo,azul),(verde,rojo),
  (verde,azul),(azul,rojo),(azul,verde)}

Ejemplo: Coloreado de Mapas



- Soluciones deben ser completas y consistentes,
- Ejm., Valencia = rojo, Caracas = verde, Barcelona = rojo, Pto. Ordaz = verde, Cda. Bolivar = rojo, San Juan = azul,

Búsqueda por profundidad con restricciones

Se puede retroceder en la busqueda para conseguir solución que satisfaga restricciones (Backtracking)

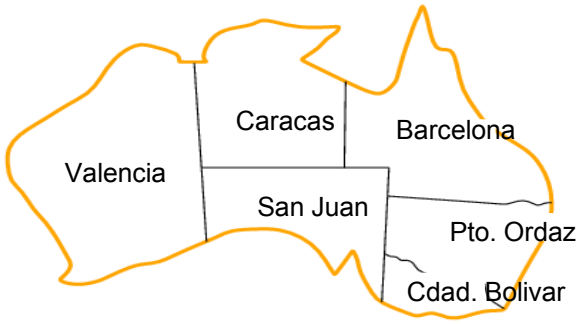
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE( Variables[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment according to Constraints[csp] then
            add { var = value } to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

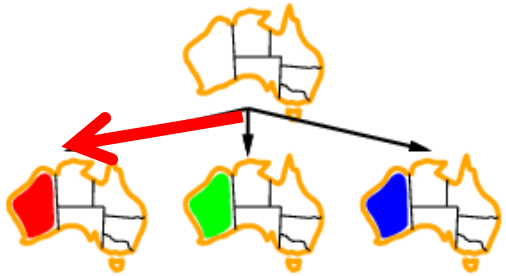
retrocede

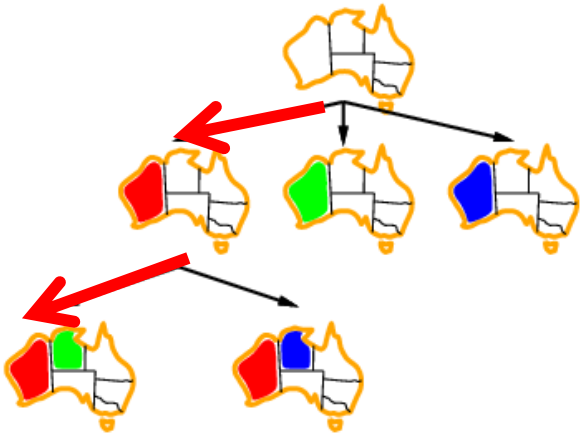
parada

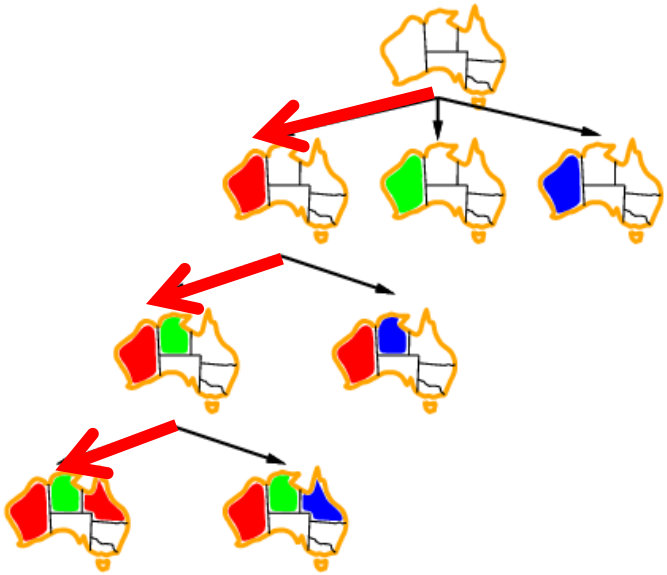
recursividad

Búsqueda por profundidad con restricciones



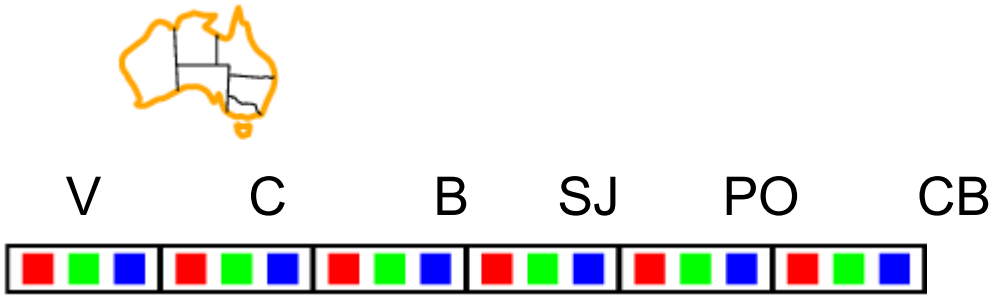




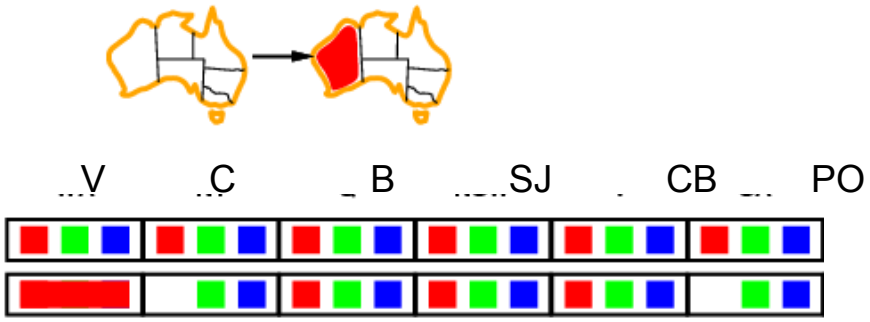


Chequeo Hacia Atras

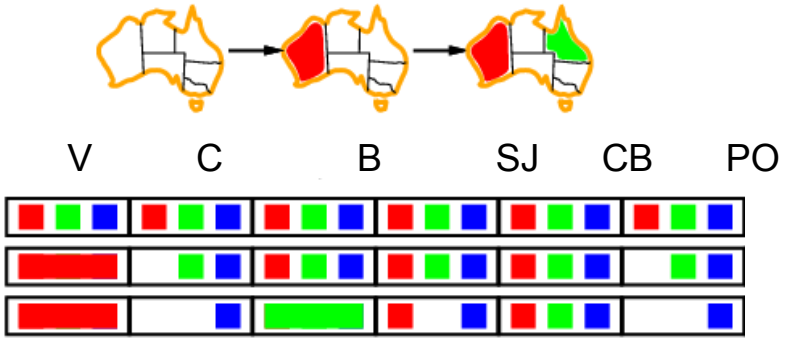
- Ideas:
  - Guardar **traza de valores legales remanentes** para variables no asignadas
  - **Terminar búsqueda** en una rama cuando cualquier variable tenga valores no legales

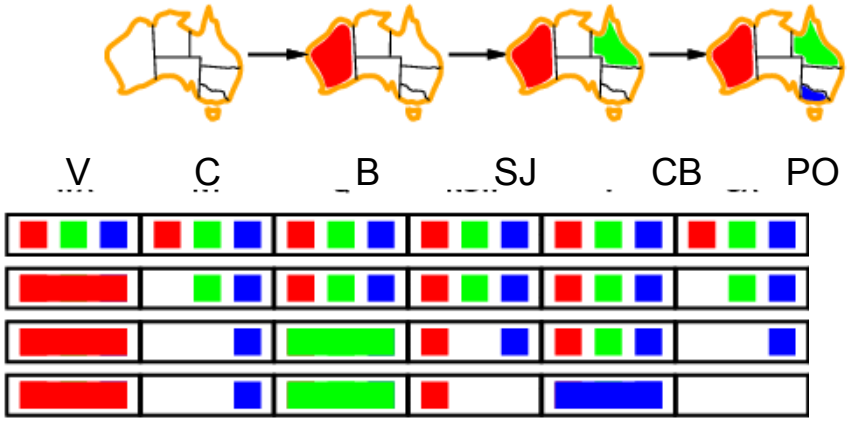


Chequeo Hacia Atras



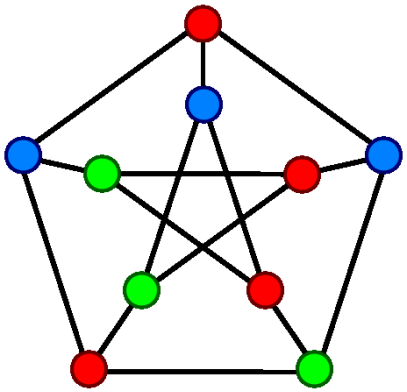
Recorrido de una rama por profundidad





Mínimo grafo k-coloreado

Existe una forma de asignar k colores a los vértices de un grafo de modo que no hayan arcos conectando vértices del mismo color?



Considerar el problema de encontrar un k-coloreo para un grafo G(V, E), para k fijo.

mínimo grafo k-coloreado

```
algoritmom_coloreado(ent k:entero;
                     entsal x:sol)
{Se usa una variable global g de tipo grafo.
 En x se tiene la parte de la solución ya calculad
 (es decir, hasta x[k-1]) y k es el índice del
 siguiente vértice al que se va a asignar color.}
principio
  repetir
  {generar todos los colores 'legales' para x[k]}
    siguienteValor(x,k); {x[k]:=un color legal}
    si x[k]≠0  {se ha encontrado un color legal}
      entonces
        si k=n
          entoncesescribir(x)
          sino m_coloreado(k+1,x)
        fsi
    fsi
  hastaQuex[k]=0
fin
```

mínimo grafo k-coloreado

Planificación de las Operaciones de Entrada/Salida

El procesamiento masivamente paralelo es en la actualidad la respuesta más prometedora en la búsqueda por incrementar el rendimiento de los computadores.

Muchas de las actuales aplicaciones requieren de un supercomputador que sea capaz de procesar grandes cantidades de datos en forma eficiente.

El cuello de botella debido a las operaciones de E/S en los computadores paralelos, ha generado diferentes esquemas para resolver este problema,

- usando paralelismo de bajo nivel (a través de técnicas tales como partición de discos),
- solapando el acceso a los datos con cálculos, paralelizando el acceso a los datos (ya sea explotando el paralelismo a nivel del programa o incrementando los recursos), y otros más.

La planificación de las operaciones de E/S es otra alternativa

mínimo grafo k-coloreado

Planificación de las Operaciones de Entrada/Salida

- El modelo básico de un subsistema de E/S comprende un conjunto de procesadores de E/S y un conjunto de discos.
- Cada procesador de E/S controla uno o más discos.
- Los procesadores de E/S pueden ser procesadores que no corren ningún tipo de trabajo, o pueden ser procesadores que también realizan cómputos.

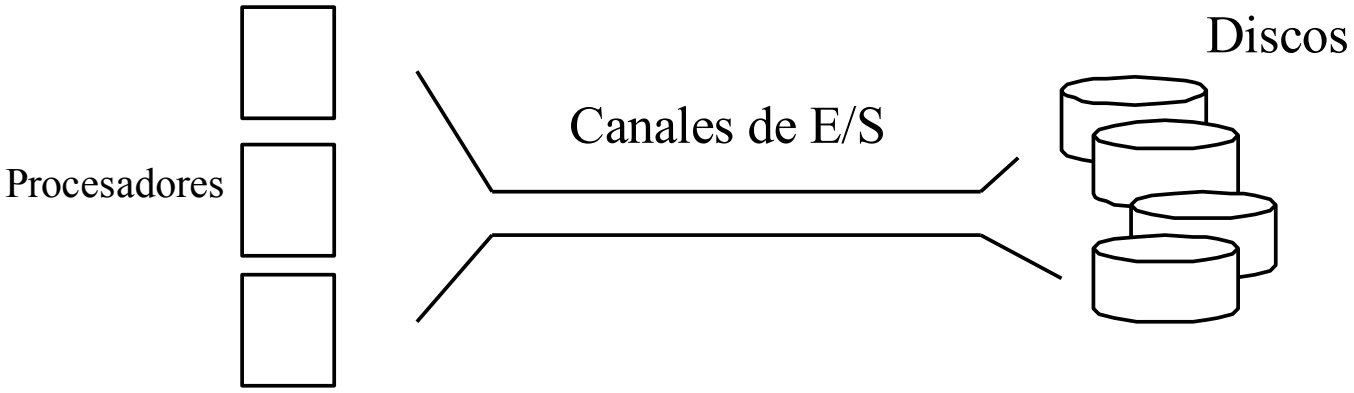
El objetivo del problema de optimización de las operaciones de E/S es proveer un camino de comunicación libre de cuellos de botella entre los procesadores y las unidades de E/S.

mínimo grafo k-coloreado

Planificación de las Operaciones de Entrada/Salida

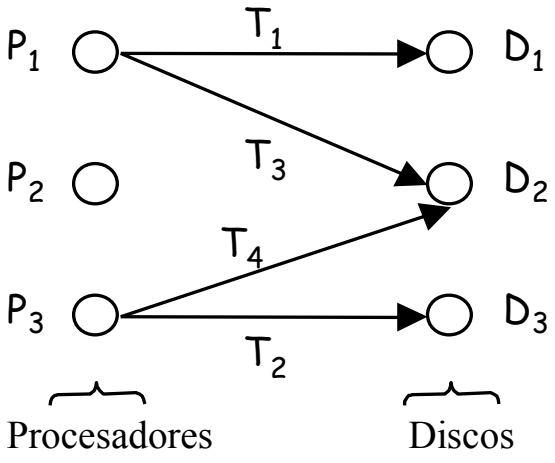
El problema de planificación de las operaciones de E/S puede ser resuelto modelándolo como el problema de comunicación entre los procesadores y las unidades de E/S, tales como los discos.

consiste en maximizar la utilización del camino entre ambos conjuntos y minimizar el tiempo para comenzar cada operación de E/S.

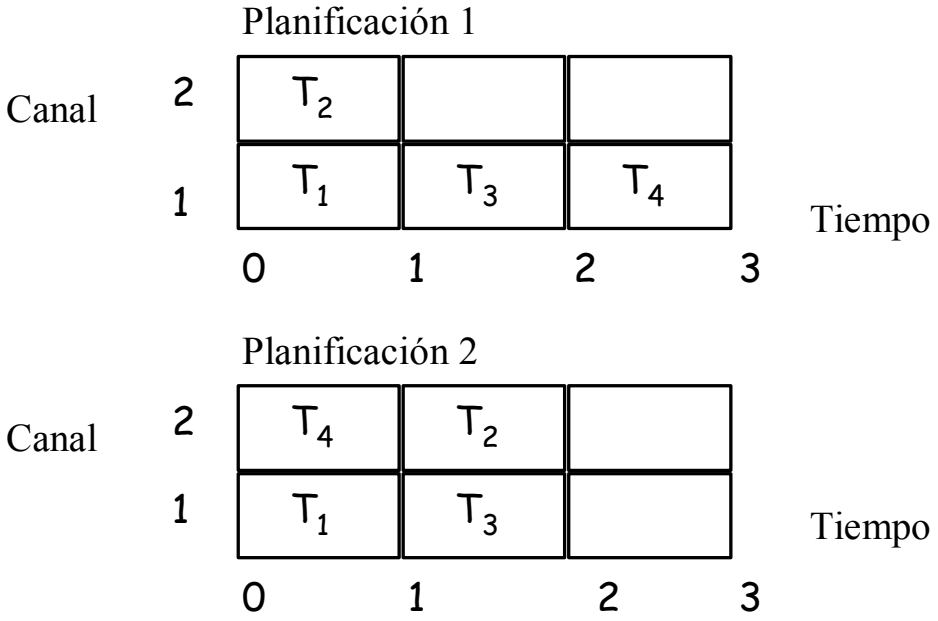


mínimo grafo k-coloreado

Planificación de las Operaciones de Entrada/Salida



a)



b)

mínimo grafo k-coloreado

Planificación de las Operaciones de Entrada/Salida

P1, P2, P3: representan los procesadores.
D1, D2, D3: representan los recursos de E/S.
T1, T2, T3, T4: representan las transacciones entre un procesador y una unidad de E/S. Pueden tener un peso para representar la cantidad de datos a transferir entre ellos (Peso(T_j)).

El problema de planificación modelado de esta forma puede representarse como un problema de mínimo grafo k-coloreado, donde k es el ancho de banda entre los procesadores y los recursos de E/S.

Así, k determina el volumen máximo de datos que se pueden transferir en un momento dado, entre los recursos de E/S y los procesadores. determina la longitud de la planificación.

mínimo grafo k-coloreado

Planificación de las Operaciones de Entrada/Salida

El problema de mínimo grafo coloreado es definido como:

Dado un grafo G=(V, E) no dirigido, el problema de mínimo grafo coloreado consiste en encontrar el mínimo conjunto de colores necesarios para colorear los arcos E, tal que dos arcos con el mismo color no comparten un vértice en común.

Mientras que el problema de mínimo grafo k-coloreado puede ser definido como:

El problema de mínimo grafo k coloreado consiste en resolver el problema de mínimo grafo coloreado, tal que cada color pueda ser usado para colorear k arcos como máximo.

.

mínimo grafo k-coloreado

Planificación de las Operaciones de Entrada/Salida

Resolver el problema del mínimo grafo k-coloreado es equivalente a resolver el problema de optimización de las operaciones de E/S (mínimo número de transacciones, es equivalente al mínimo número de colores), de tal manera de ejecutar en el mínimo tiempo posible el mayor número de transacciones u operaciones de E/S, maximizando el uso de los recursos de E/S y de los canales de comunicación.

Así, un grafo coloreado de G representa una planificación de las operaciones de E/S, donde todos los arcos con *color=i* representan el grupo de transferencias de datos que pueden tomar lugar en el mismo instante de tiempo.

El número de colores requerido para colorear los arcos determina la longitud de la planificación.

Problema del Racimo Mínimo

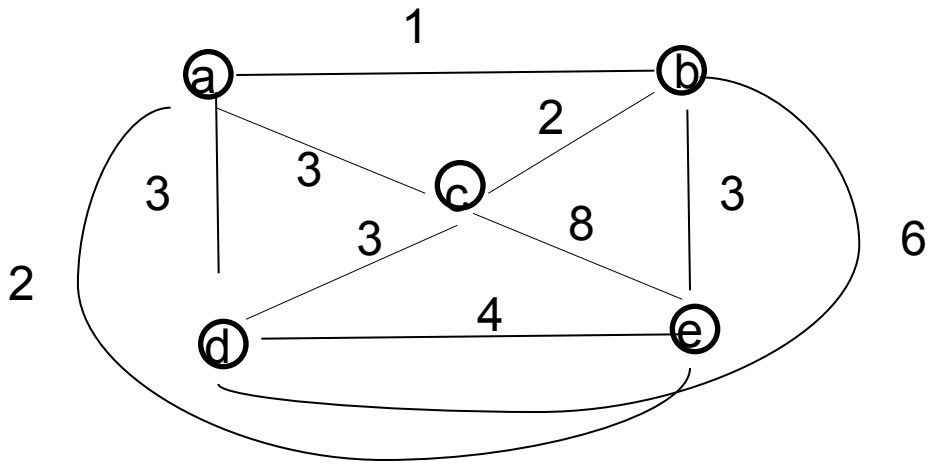
- Sea G=<N,A>, un grafo no dirigido y sea c: A → R+ una función de coste.
- Considere tres sub conjuntos N1, N2, N3, que se llaman racimos, de tal manera que cada nodo N, pertenezca exactamente a uno de los racimos además de ser las *aristas internas aquellas que enlazan nodos de un mismo racimo y las aristas cruzadas, aquellas que enlazan nodos de racimos* distintos,

el problema esta en seleccionar N1, N2, N3, de tal manera que el coste de las aristas cruzadas sea máximo o equivalente de la forma que minimice el coste total la aristas internas.

Este problema tiene dificultad NP

Problema del Racimo Mínimo

La solución óptima consiste en seleccionar N1={a,b}, N2={c,d} y N3={e}, de tal manera que el coste total de las aristas internas es 4 y de las aristas cruzadas es 31.



Problema del Racimo Mínimo

- Es equivalente maximizar el coste total de las aristas cruzadas o minimizar el coste total de las aristas internas, por tanto se consideran dos problemas de optimización siguientes:

- MAX-CORTAR: maximiza el coste total de las aristas cruzadas, para todas las particiones de N.
- MIN-RACIMO: minimiza el coste total de las aristas internas, para todas las particiones de N.

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Problema del Racimo Mínimo

Versión 1.0

MAX-CORTAR_aprox(G=<N,A>, c: A → R+)

pre(c: A → R+) pos(N1, N2, N3 ≠∅) ^ coste aristas cruzadas

```
1   N1,N2,N3 ← ∅
2   racimo, suma ← 0
3   para todo u ∈N hacer
      costemin ← ∞
4     para i ← 1 hasta 3 hacer
        coste ← 0
5       para todo v ∈Ni hacer
          si {u,v} ∈A entonces
            coste ← coste + c({u,v})
          fsi
        frp
        suma ← suma + coste
        si coste < costemin entonces
          costemin ← coste
          k ← i
        fsi
      frp
      Nk ← Nk ∪ {u}
      Racimo ← racimo + costemin
    frp
6   devolver suma – racimo
```

-N1,N2,N3: conjuntos donde se formaran una parición de N cuando concluya el algoritmo.

-suma: acumula el coste total de todas las aristas de G.

-racimo: acumula el coste de todas las aristas internas en la solución aproximada que Seleccionamos

Problema de la mochila

- Problema
  - Se tienen n objetos y 1 morral.
  - Cada objeto i tiene un peso w_i y un valor positivo v_i.
  - Capacidad máxima del morral W.

- Objetivo: llenar el morral para maximizar el valor de los objetos transportados, respetando la limitación de capacidad.

- Formulación: Sea x_i igual a 0 si no se toma el objeto i, o 1 si se incluye el objeto.

Problema de la mochila

El objeto i contribuye en $x_i w_i$ al peso total del morral y en $x_i v_i$ al valor total de la carga.

$$\textbf{Maximizar } (\Sigma_i \, x_i v_i)$$

con la restricción de

$$\Sigma x_i w_i < W,$$

donde v y w son positivos y los X son enteros positivos.

$$v_i > 0,\ w_i > 0 \text{ y } x_i \in \{0,1\} \text{ para } 1 \leq i \leq n$$

En esta primera versión del problema de la mochila vamos a suponer que podemos llevar a lo sumo un objeto de cada tipo o "partes" de los mismos, es decir que consideraremos que las variables x_j son reales positivas, o sea queremos resolver el siguiente problema:

$$\text{Max} \ \Sigma_j c_j x_j$$

sujeto a que

$$\Sigma_j a_j x_j \leq b$$

$$0 \leq x_j \leq 1$$

Ejemplo:

n = 5, b = 100 y los beneficios y pesos están dados en la tabla

a_j	10	30	20	50	40
c_j	20	66	30	60	40

Posibles ideas para un algoritmo goloso para este problema:

- Elegir los objetos en orden decreciente de su beneficio….
- Elegir los objetos en orden creciente de su peso…….

Sirven estas ideas?. Dan el resultado correcto?

Que pasa si ordenamos los objetos por su relación beneficio/peso? O sea en orden decreciente de los c_j/a_j ?.

Lema: Si los objetos se eligen en el orden decreciente de los valores c_j/a_j entonces el algoritmo goloso encuentra la solución óptima al problema de la mochila para variables continuas.

Dem: supongamos que tenemos los elementos ordenados en orden decreciente de su relación c_j/a_j . Sea $X = (x_1,\ldots x_n)$ la solución encontrada por el algoritmo goloso. Si todos los x_i son 1 la solución es óptima. Sino sea j el menor índice tal que sea $x_j < 1$.

Es claro que $x_i = 1$ para $i < j$ y que $x_i = 0$ para $i > j$ y que

$$\sum a_i x_i = b$$

Veamos que ocurre con el problema de la mochila, en el caso de que las variables x_j tengan que tomar necesariamente valores enteros (como ocurre en la práctica, es decir cuando no podemos llevar un "pedazo" de un objeto).

Algoritmo goloso para el problema de la mochila con variables enteras nonegativas:

- Ordenar los elementos de forma que

  $$c_j / a_j \geq c_{j+1} / a_{j+1}$$

- Para j=1,n y mientras $b \neq 0$ hacer

  $$x_j = \lfloor b / a_j \rfloor$$
  $$b = b - a_j x_j$$
  $$z = z + c_j x_j$$

- Parar

Cuando las variables son enteras este algoritmo goloso no da siempre la solución óptima para el problema de la mochila.

Ejemplo: tenemos una mochila de tamaño 10 y 3 objetos, uno de peso 6 y valor 8, y dos de peso 5 y valor 5.

En el caso de variables enteras el algoritmo goloso que presentamos puede considerarse como una heurística para resolver el problema.

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Problema de la mochila

Versión 1.0

mochila (Arreglo[1..n]de Entero+: w, Entero+: W, Arreglo[1..n]de Entero+: v):
Matriz[1..n,0..W]de Entero: V
{Función que crea la tabla con los valores máximos}

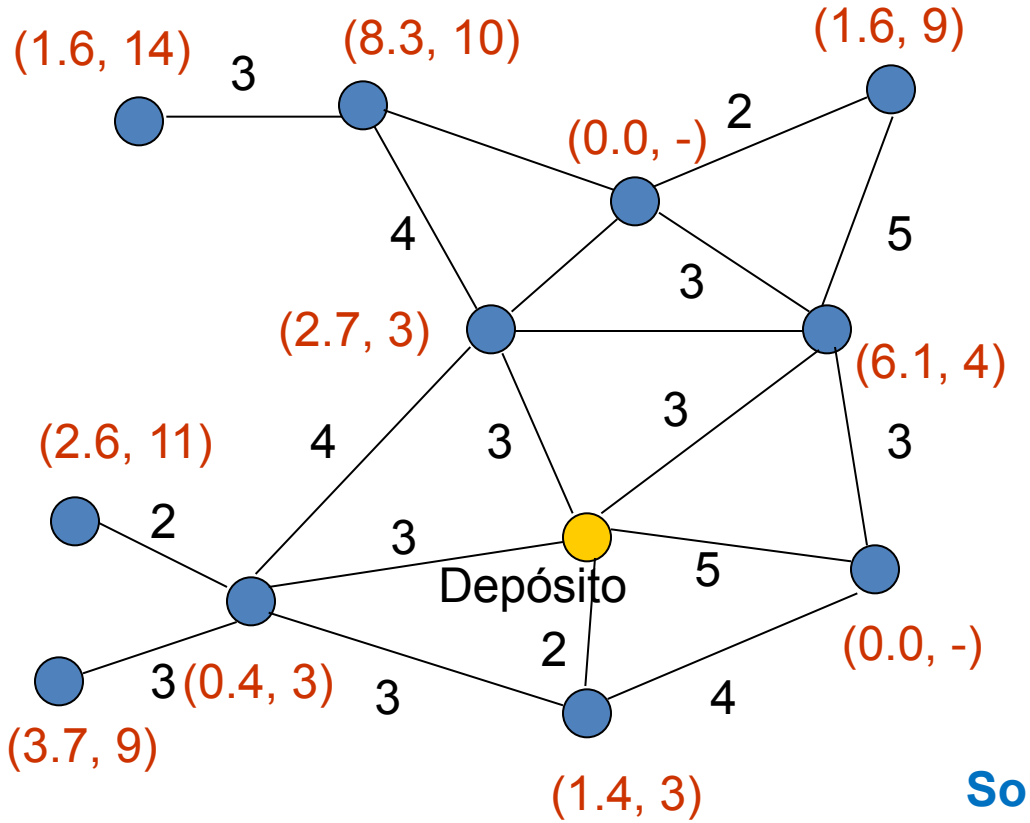
{pre: n>0, W>0} **{pos: matriz con los valores máximos }**

1	[Si(w[1]<W) entonces V[1,j]=w[i] sino V[1,j]=0 fsi]j=0,W	**v:Arreglo[1..n]de Entero+. Contiene los valores de cada uno de los objetos.** **w: Arreglo[1..n]de Entero+. Contiene los pesos de cada uno de los objetos.** **V: Matriz[1..n,0..W]de Entero. Contiene los valores máximos**
2	[[Si(w[i]>j) entonces Si(V[i-1,j]>v[i]) entonces V[i,j]= V[i-1,j] sino V[i,j]= v[i] fsi sino Si(V[i-1,j]>V[i-1,j-w[i]]+v[i]) entonces V[i,j]= V[i-1,j] sino V[i,j]= V[i-1,j-w[i]]+v[i] fsi]j=1,W]i=1,n	
3	**Regrese V**FinPara	

Problema de Enrutamiento de Vehículos

- Grafo G = (V, A), donde el vértice v_0 es el "depósito" desde el cual construyo las rutas, y los demás vértices son "ciudades"
- En el depósito hay m vehículos idénticos
- Cada arco tiene asociada una distancia no negativa
- Para simplificar: "costo" = "tiempo" = "distancia"
- El objetivo es construir un conjunto de costo mínimo de rutas tales que
  - (a) Todas las rutas comienzan y terminan en el depósito
  - (b) Cada ciudad es visitada exactamente una vez por exactamente un vehículo
  - (d) Cada ciudad tiene asociada una demanda q_i. La demanda total asociada a un vehículo no puede exceder la capacidad Q del vehículo.
  - (e) Cada ciudad requiere ser atendida en un tiempo máximo Di, y el tiempo total de cada ruta (incluyendo los servicios) no puede superar un cierto valor L

Problema de Enrutamiento de Vehículos



(demanda, tiempo)

Constantes:

m = 5 (cantidad de vehículos)

Q = 11.5 (capacidad de un vehículo)

L = 25 (duración máxima de una ruta)

Solución:
Híbrido mochila y VC

Problema 8-puzzle

Estado inicial

□	5	2
1	8	3
4	7	6

→

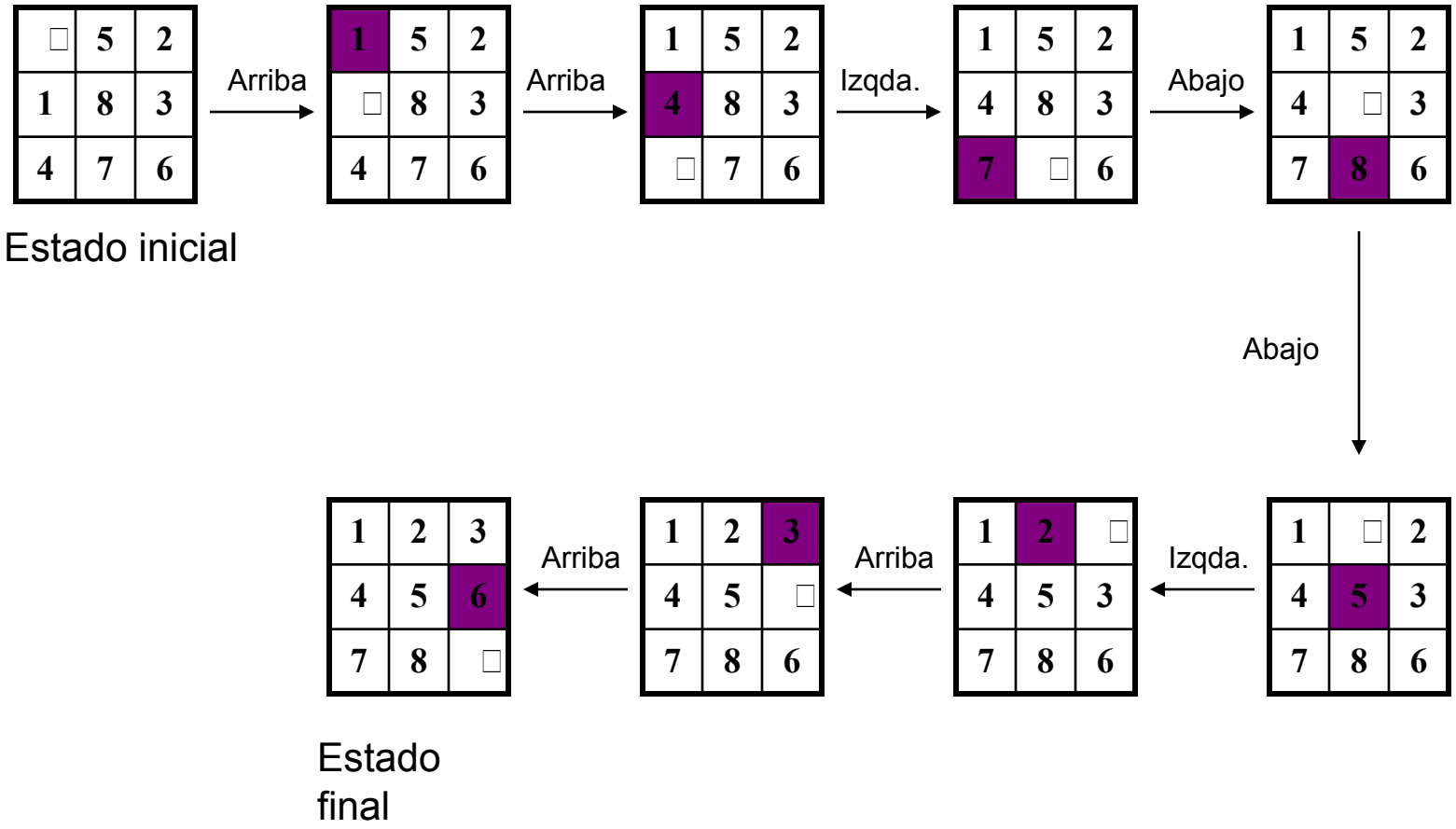
Estado final

1	2	3
4	5	6
7	8	□

S = Conjunto de todas las secuencias de movimientos que conducen del estado inicial al estado final.

f = Número de movimientos en la secuencia.

Problema 8-puzzle



- l(x): Función de búsqueda. Coste de llegar del estado inicial al estado final pasando por el estado x.
- h(x): Función heurística. Coste estimado de llegar desde x al estado final.
- g(x): Coste de llegar desde el estado inicial al estado x.
- l(x) = g(x) + h(x)

Problema N-Reinas

- El problema de las N-Reinas consiste en colocar n reinas en un tablero de ajedrez de tamaño n*n de forma la reinas no se amenacen según las normas del ajedrez. Se busca encontrar una solución o todas las soluciones posibles.

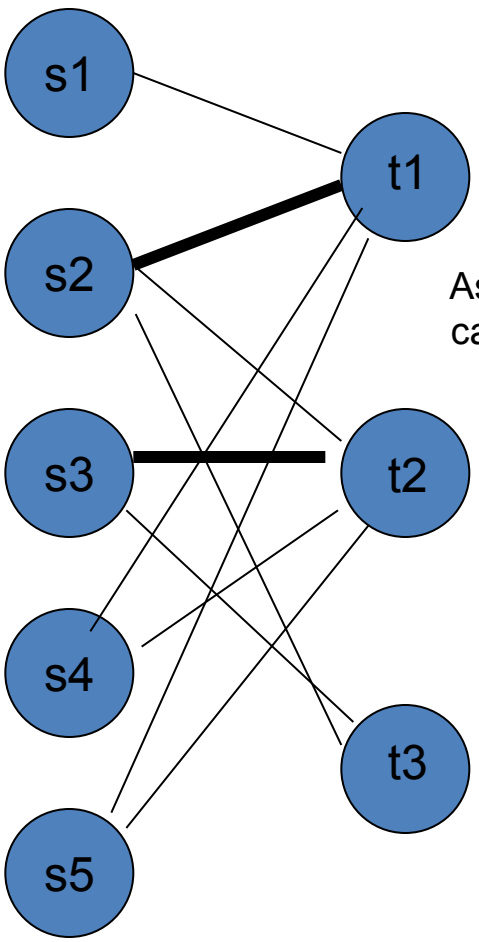
- Cualquier solución del problema estará formada por una n-tupla $(x_1,x_2,\ldots,x_n)$, dónde cada x_i indica la columna donde la reina de la fila i-ésima es colocada.

- Las restricciones para este problema consisten en que dos reinas no pueden colocarse en la misma fila, ni en la misma columna ni en la misma diagonal.
  - Por ejemplo, el problema de las 4-Reinas tiene dos posibles soluciones: [2,4,1,3] y [3,1,4,2].

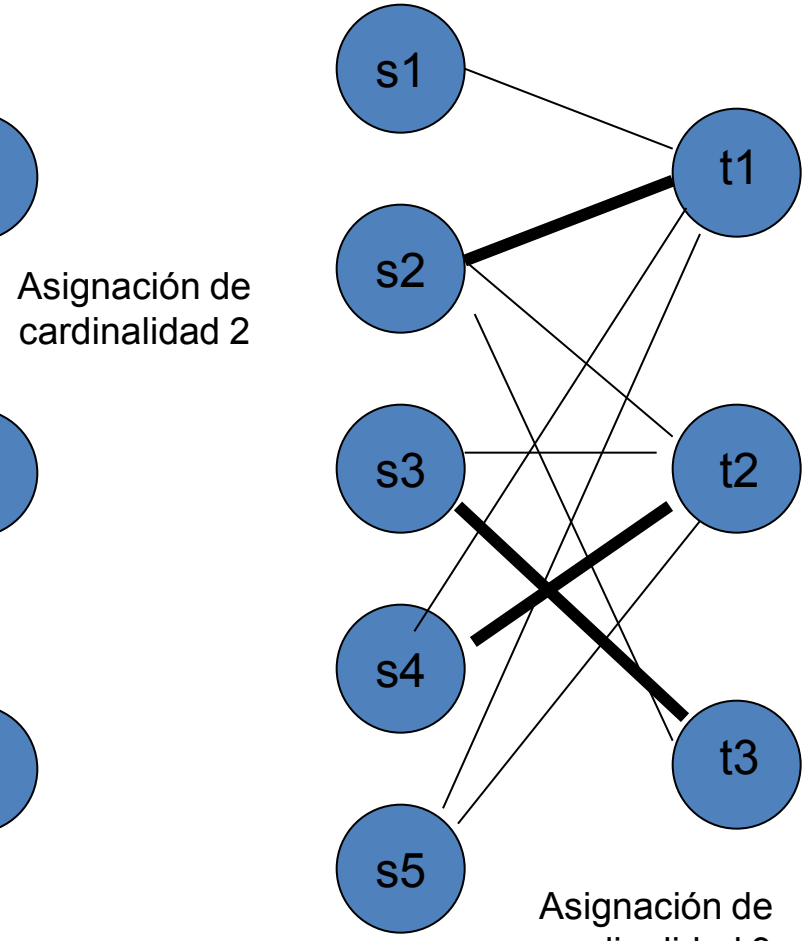
Problema de aparejamiento ("matching“) bipartito máximo

Dado un grafo no dirigido G = (N, A), un "matching" es un subconjunto de aristas M ⊆ A, tal que para todos los nodos v ∈ N, a lo sumo una arista de M incide en v.

Un **"matching" M es máximo** si tiene una cardinalidad máxima tal que para cualquier M' se tiene | M' | ≥ | M |.



Grafo bipartido



Asignación de cardinalidad 2

Asignación de cardinalidad 3

Próxima tarea

Analizar y Desarrollar los algoritmos más utilizados para los siguientes problemas matemáticos:

1. Inversión de Matrices
2. División y Multiplicación de Polinomios
3. Resolución de Sistemas de ecuaciones:
   - Método Jacobi
   - Método Gauss-Seidel
4. Algoritmo RSA de Criptografía
5. Transformada de Fourier
6. Estimación de Mínimos Cuadrados