

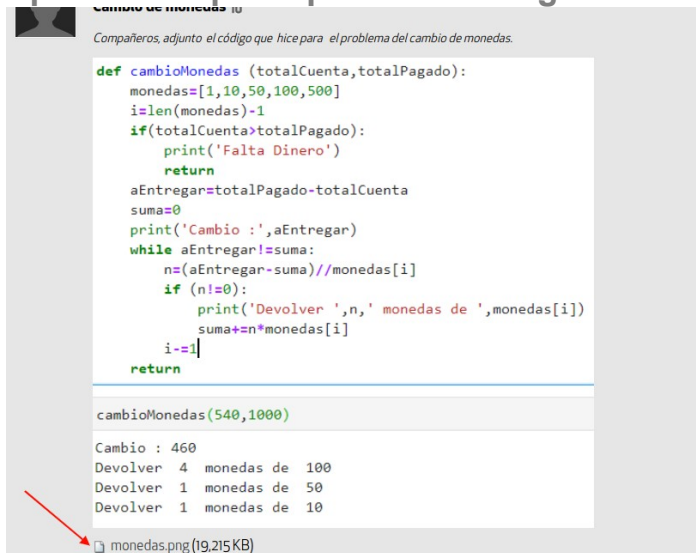
VC3 – Problemas tipo y Algoritmos de Búsqueda

03MIAR – Algoritmos de Optimización

Aportaciones de código en el foro

Incluir código importante con formato

Opción 1: captura parcial en imagen



Opción 2: pegado con formato

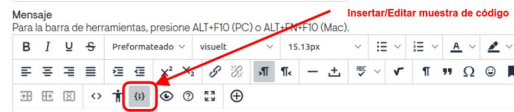
Os adjunto otro algoritmo:

se supone sistema monetario está ordenado menor a mayor (0.1,0.2,0.5,1,2,10,20,50,100,200,500)

```
import numpy as np

def cambio monedas(cantidad, sistema monetario):
    total_monedas = len(sistema_monetario)
    solucion = np.zeros(total_monedas)
    valor_acumulado = 0
    for i in range(total_monedas-1,0,-1):
        monedas = int((cantidad-valor_acumulado)/sistema_monetario[i])
        valor_acumulado = valor_acumulado + monedas * sistema_monetario[i]
        solucion[i] = monedas
        if valor_acumulado == cantidad:
            return solucion
    return []
sistema_monetario = (0.1,0.2,0.5,1,2,5,10,20,50,100,200,500)

solucion = cambio_monedas(1111,sistema_monetario)
if len(solucion) == 0:
    print("NO HAY SOLUCIÓN")
else:
    for i in range(len(solucion)):
        if solucion[i]>0:
            cantidad = int(solucion[i])
            texto_cantidad = ""
            if cantidad == 1:
                texto_cantidad = " moneda de: "
            else:
                texto_cantidad = " monedas de: "
            print(cantidad, texto_cantidad, sistema_monetario[i])
```



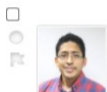
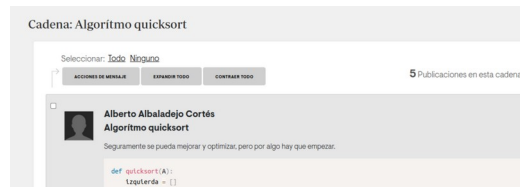
Añadir enlace al cuaderno completo

Algoritmo voraz:

<https://colab.research.google.com/drive/1t4krkJdGvkhaCQgQ100tAOTUrClvRcq>

Aportaciones en el foro

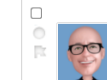
Quick Sort



Miguel Mejía Cabello RE: Algoritmo quicksort

Hace 4 días

Hola Alberto, lo que yo hago primero es encontrar el caso base que es cualquier arreglo o lista menor a dos elementos de ser así devuelvo el arreglo que ya estaría ordenado. En segundo lugar elijo un pivote en mi caso el primero del arreglo, luego recorro el arreglo separando los que son mayores y menores al pivote. Finalmente convino todo en orden retornando

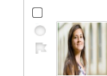


Manuel Esteban-Infantes RE: Algoritmo quicksort

Hace 4 días

En ambos casos pueden usar la siguiente función para elegir un pivot aceptable - selecciona dos números al azar de la lista, cuenta cuántos elementos hay mayores al primero, menores al segundo, y se queda con el pivot más cercano al medio. No garantiza el mejor pivot, pero sí que no nos quedemos con el peor.

```
from random import sample
```



María José Toledo Arcic RE: Algoritmo quicksort

Hace 4 días


Hola chicos,

también estuve investigando acerca del algoritmo de quicksort y esta implementación me parece bastante buena, de hecho es la mejor que he encontrado... (No la he implementado yo... 😊).

El algoritmo que les presento recorre la lista inicial por ambos lados. El valor low, que es el valor correspondiente al valor de la lista que está en la posición 0, se recorre de izquierda a derecha, aumentando +1 en cada iteración, siempre y cuando el valor de la izquierda sea menor que el pivote. El parámetro high toma inicialmente el mayor índice de la lista, y avanza de derecha a izquierda, siempre y cuando el valor sea más grande que el pivote, es decir, `input_list[high] >= pivot`. El algoritmo termina cuando los índices low y high se cruzan.

Aportaciones en el foro

Tiempos de ejecución

**Manuel Esteban-Infantes**Hace 2 días

Heap Sort, y tiempos de algunas contribuciones

Os dejo el [enlace a un libro que tiene una implantación de los algoritmos de ordenación comentados](#), incluido un heap sort y varias de las contribuciones a este foro, para que probéis y veáis los tiempos para distintos tamaños de lista. En cuanto se aumenta el tamaño se aprecian notables diferencias de tiempos, y si seguimos subiendo provocamos el error de stack overflow en los algoritmos que usan recursividad (no en el heap sort). También se nota el mayor rendimiento de las funciones que bajan a C frente a las manipulaciones directas de las listas por índices.

Algunos tiempos:

Tiempo medio (10 bucles) con una lista de 100 elementos.

```
Base (shuffle) : 0:00:00.000083.  
Bubble Sort : 0:00:00.000984.  
Insertion Sort : 0:00:00.000786.  
Selection Sort : 0:00:00.000467.  
Heap Sort : 0:00:00.000336.  
Quick Sort Miguel : 0:00:00.000178.  
Quick Sort Alberto : 0:00:00.000157.  
Quick Sort Mª José : 0:00:00.000179.
```

Tiempo medio (10 bucles) con una lista de 1000 elementos.

```
Base (shuffle) : 0:00:00.000845.  
Bubble Sort : 0:00:00.124506.  
Insertion Sort : 0:00:00.096459.  
Selection Sort : 0:00:00.053687.  
Heap Sort : 0:00:00.006864.  
Quick Sort Miguel : 0:00:00.005859.  
Quick Sort Alberto : 0:00:00.007096.  
Quick Sort Mª José : 0:00:00.008771.
```

Tiempo medio (10 bucles) con una lista de 5000 elementos.

```
Base (shuffle) : 0:00:00.004526.  
Bubble Sort : 0:00:03.138543.  
Insertion Sort : 0:00:02.484802.  
Selection Sort : 0:00:01.257566.  
Heap Sort : 0:00:00.039108.  
Quick Sort Miguel : 0:00:00.015261.  
Quick Sort Alberto : 0:00:00.015152.  
Quick Sort Mª José : 0:00:00.021080.
```

Responder **CITAR** **EDITAR** **ELIMINAR** **ESCRIBIR CORREO ELECTRÓNICO AL AUTOR**

Agenda

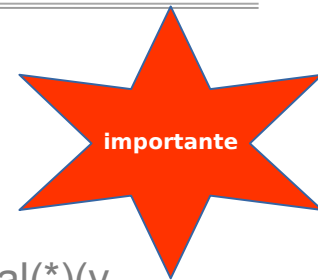
1. Programación lineal
2. Programación lineal entera
3. Problema del agente viajero
4. Problema de la mochila
5. Otros problemas
6. Búsqueda en amplitud
7. Búsqueda en profundidad
8. Ramificación y Poda

VC3 - Programación Lineal

Agenda

1. Programación lineal
2. Programación lineal entera
3. Problema del agente viajero
4. Problema de la mochila
5. Otros problemas
6. Búsqueda en amplitud
7. Búsqueda en profundidad
8. Ramificación y Poda

Programación lineal (I)



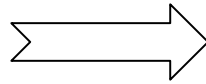
Definición: Son problemas de **programación lineal** aquellos para los que se necesita optimizar(maximizar o minimizar) una función objetivo que es lineal(*) (y por tanto continua) y multivariable sobre unas restricciones que vienen expresadas en igualdades o desigualdades también lineales.

- Existe muchos problemas que pueden ser enunciados en términos lineales(flujo, inventario, gestión de cartera, asignación de recursos, planificación...)
- **Resolución:** Método del **Simplex** y Método del Punto Interior.
- Dependiendo de la región que delimitan las restricciones el problema puede tener una, infinitas, no acotada o ninguna solución.

Programación lineal (II)

- **Método del Simplex(Dantzig, 1949):** Se obtienen soluciones iterativamente a través de la frontera de la región factible hasta llegar al optimo.
- El éxito de su eficiencia se basa en que no es necesario revisar todos los puntos extremos de la región factible(donde se sabe que está el optimo) .
- Aunque hay otros métodos sigue siendo el más utilizado.
- Está incluido como una función en Excel

$$\begin{aligned}\text{Max } & 200X + 150Y + 120Z \\ \text{S.A. } & 15X + 7,5Y + 5Z \leq 315 \\ & 2X + 3Y + 2Z \leq 110 \\ & X + Y + Z \leq 50 \\ & X, Y, Z \geq 0\end{aligned}$$



X	Y	Z	F.OBJETIVO	
4	10	36	6620	
200	150	120		

RESTRICCIONES				
X	Y	Z	LADO IZQ	LADO DER
15	7,5	5	315	315
2	3	2	110	110
1	1	1	50	50

Resultados de Solver

Solver ha hallado una solución. Se han satisfecho todas las restricciones y condiciones.

☒ Utilizar solución de Solver

☐ Restaurar valores originales

Informes

Respuestas

Sensibilidad

Límites

Aceptar

Cancelar

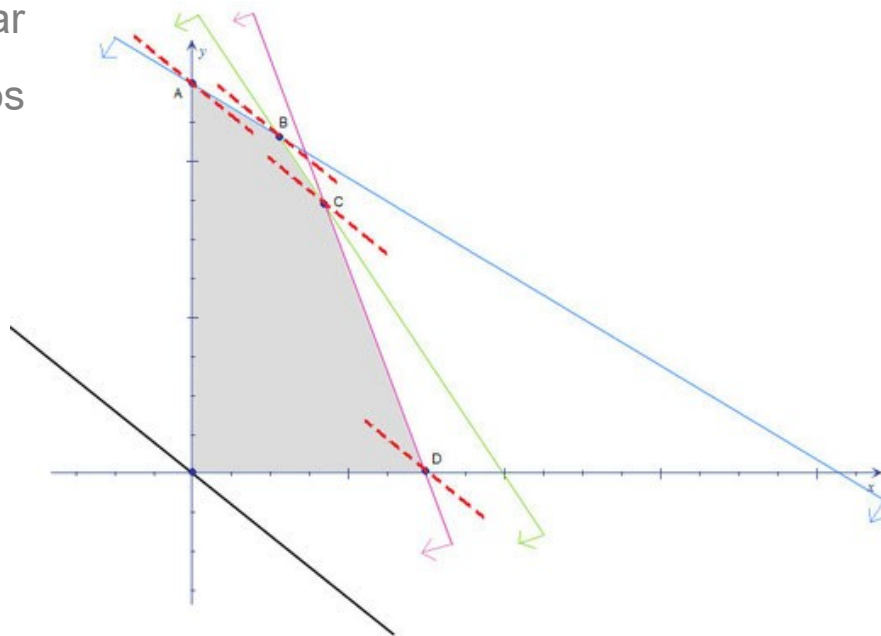
Guardar escenario...

Ayuda

Programación lineal (III)

- Método del Simplex(Dantzig, 1949)

Gráficamente se traduce en desplazar la función objetivo(línea negra) por los puntos externos de la región factible(líneas rojas) hasta que no se encuentre intersección



Programación lineal (IV)

- Método del Simplex(Dantzig, 1949):
 - Debemos reconocer un problema de programación lineal
 - Admite gran cantidad de variables y restricciones

- Recursos

- Calculadora on-line:

<http://simplex.tode.cz/en/>

- Sitio Web

SIMPLEX CALCULATOR

Maximize

$Z = \square x_1 + \square x_2 + \square x_3$

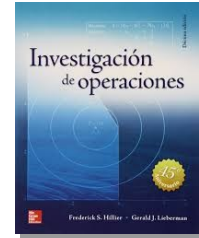
subject to

$\square x_1 + \square x_2 + \square x_3 \leq \square$

$x_{1,2,3} \geq 0$

[+ ADD NEW RESTRICTION](#)

[SOLVE](#)



<https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci%C3%B3n-de-operaciones/m%C3%A9todo-simplex/>

Programación lineal (V)

IBM® Decision Optimization CPLEX® Modeling for Python



Welcome to IBM® Decision Optimization CPLEX® Modeling for Python.

CPLEX de IBM:

<https://en.wikipedia.org/wiki/CPLEX>

Integración con python

<https://rawgit.com/IBMDecisionOptimization/docplex-doc/master/docs/index.html>

Programación lineal (VI)

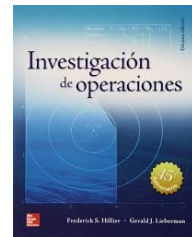
Problemas clásicos de programación lineal:

- Problema de la dieta(1939 - George J. Stigler)
Conseguir una dieta equilibrada(restricciones) al menor coste
- Problema del transporte(1949 - Tjalling C. Koopmans)
Minimizar el coste de transportar mercancías entre diferentes puntos que necesitan abastecimiento
- Problema de Producción
Maximizar las ganancias al producción diferentes productos con materias primas(restricciones)

Programación lineal (VII)

Bibliografía:

- Hillier, F. S., y Lieberman, G. J. (2015): McGraw-Hill Education.



- Kong, Maynard: Investigación de operaciones:

programación lineal, problemas de transporte, análisis de redes(*)

<https://elibro-net.universidadviu.idm.oclc.org/es/ereader/universidadviu/79351?page=1>



VC3 - Programación Lineal entera

Agenda

1. Programación lineal
2. Programación lineal entera
3. Problema del agente viajero
4. Problema de la mochila
5. Otros problemas
6. Búsqueda en amplitud
7. Búsqueda en profundidad
8. Ramificación y Poda

Programación lineal entera (I)

- Son problemas de **programación lineal** en los que se exige que algunas o todas las variables sean **enteras**.
- Tipo
 - Puros: **todas** las variables debe ser enteras
 - Mixtos: **algunas** variables debe ser enteras
- **No existe un algoritmo** específico que los resuelva. Cada problema requiere un diseño específico según la técnica más adecuada.

Programación lineal entera (II)

- **Resolución:**

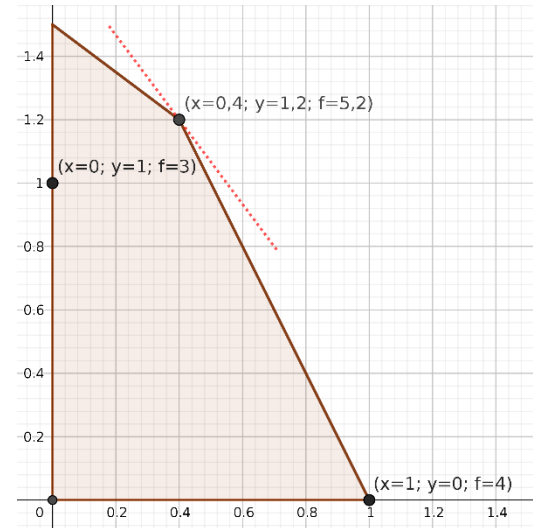
- ✓ Resolver el problema lineal “relajado” sin las restricciones de variables enteras (p.ej SIMPLEX). Si la solución es entera...Hemos tenido suerte!!!
- ✓ Redondear las soluciones del problema “relajado”. **Mala opción!. Descartable!.** Contraejemplo:

$$\max 4x + 3y$$

$$2x + y \leq 2$$

$$3x + 4y \leq 6$$

$$x, y \in [0, 1]$$



Programación lineal entera (III)

- **Resolución:**
 - ✓ Algoritmos de búsqueda con Ramificación y Poda
 - ✓ Algoritmos heurísticos
- **Multitud de problemas:**
 - ✓ Problema de Inversión de capital. Elección de la cartera de valores
 - ✓ Problema de coste fijo.
 - ✓ Cubrimiento de conjuntos
 - ✓ ...

Programación lineal entera (IV)

- **Problema de Inversión de capital. Elección de la cartera de valores**

Se están considerando cuatro posibles inversiones. La primera de ellas proporciona actualmente unos beneficios netos de 16000 euros, la segunda 22000 euros, la tercera 12000 euros, y la cuarta 8000 euros. Cada una de las inversiones requiere cierta cantidad exacta de dinero en efectivo: 5000, 7000, 4000 y 3000 euros, respectivamente. Si solamente se dispone de 14000 euros para invertir. ¿Que modelo permite obtener la combinación de inversiones que proporcionará los máximos beneficios?

$$\text{Max } z = 16x_1 + 22x_2 + 12x_3 + 8x_4$$

$$\text{s.a: } 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14$$

$$x_j \in \{0, 1\}, j = 1, 2, 3, 4$$

$$x_j = \begin{cases} 1 & \text{si se elige la inversión } j \\ 0 & \text{en otro caso} \end{cases}$$



Programación lineal entera (V)

- Problema de coste fijo

He sido abordado por tres compañías de teléfonos para que me suscriba a su servicio. MaBell cobrará una tarifa fija de 16€ al mes , más 0.25 céntimos por minuto. PaBell cobrará 25€ al mes de tarifa fija, pero reducirá el coste por minuto a 0.21 céntimos. En cuanto a BabyBell (compañía recién estrenada) ofrece una tarifa fija mensual de 18€ y un coste por minuto de 0.22 céntimos. Las compañías solamente me cobrarán la tarifa fija si realizo alguna llamada a través de su operador. Teniendo en cuenta que realizo un promedio mensual de 200 minutos en llamadas, y que puedo repartir dichas llamadas entre las tres compañías, ¿Cómo debo utilizar sus servicios de forma que la factura mensual de teléfono me resulte lo más económica posible?

x_1 = Minutos de larga distancia al mes con *MaBell*

x_2 = Minutos de larga distancia al mes con *PaBell*

x_3 = Minutos de larga distancia al mes con *BabyBell*

$$y_1 = \begin{cases} 1 & \text{si } x_1 > 0 \\ 0 & \text{en otro caso} \end{cases}$$

$$y_2 = \begin{cases} 1 & \text{si } x_2 > 0 \\ 0 & \text{en otro caso} \end{cases}$$

$$y_3 = \begin{cases} 1 & \text{si } x_3 > 0 \\ 0 & \text{en otro caso} \end{cases}$$

$$\text{Min} \quad 0,25x_1 + 0,21x_2 + 0,22x_3 + 16y_1 + 25y_2 + 18y_3$$

$$\text{s.a:} \quad x_1 + x_2 + x_3 = 200$$

$$x_1 \leq 200y_1$$

$$x_2 \leq 200y_2$$

$$x_3 \leq 200y_3$$

$$x_1, x_2, x_3 \geq 0$$

$$y_1, y_2, y_3 \in \{0, 1\}$$

VC3 – Problema del Agente Viajero (TSP, Travelling Salesman Problem)

Agenda

1. Programación lineal
2. Programación lineal entera
- 3. Problema del agente viajero**
4. Problema de la mochila
5. Otros problemas
6. Búsqueda en amplitud
7. Búsqueda en profundidad
8. Ramificación y Poda

Problema del agente viajero (TSP) (I)

- El **TSP** (travelling salesman problem) se enuncia como la búsqueda optima del camino que debe realizar un viajero para recorrer un conjunto de ciudades de tal manera que la distancia recorrida sea mínima pasando una vez y solo una vez por cada una de las ciudades y comenzando y finalizando en la misma ciudad.
- El TSP es uno de los problemas con más estudios dentro de optimización matemática debido a que es muy habitual en el reparto de mercancías.
- El problema se modela con un grafo, donde los nodos son las ciudades y las aristas son las rutas que llevan asociado un valor que representa el coste o la distancia que hay que minimizar.

Problema del agente viajero (TSP) (II)

- Formalmente el problema se puede formular como un problema de programación lineal entera : Dantzing, Fulkerson, Johnson (1954):

$$x_{ij} = \begin{cases} 1 & \text{si se visita la ciudad } j \text{ después de la } i \\ 0 & \text{e.o.c.} \end{cases}$$

c_{ij} el costo de (distancia) de visitar la ciudad i y luego la j

$$\min \sum_{\forall (i,j)} c_{ij} x_{ij}$$

sujeto a

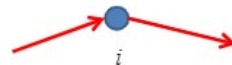
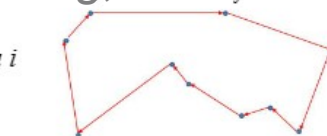
$$\sum_{j=1}^n x_{ij} = 1 \quad \text{desde alguna ciudad } i \text{ se va a una sola ciudad}$$

$$\sum_{i=1}^n x_{ij} = 1 \quad \text{a la ciudad } j \text{ se llega desde una sola ciudad}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \quad \text{restricción de eliminación de sub-tours}$$

$$\forall S \subset \{2, \dots, n\}$$

$\approx 2^n$ restricciones !



Eliminar ciclos!



Problema del agente viajero (TSP) (III)

- Se mejora en el número de restricciones: Miller, Tucker, Zemlin(1960):

$$\min \sum_{\forall(i,j)} c_{ij} x_{ij}$$

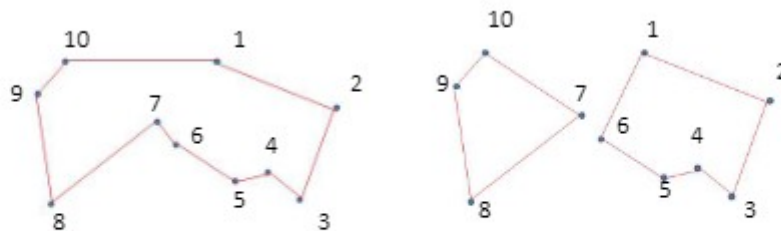
sujeto a

$$\sum_{j=1}^n x_{ij} = 1$$

desde alguna ciudad i se va a una sola ciudad

$$\sum_{i=1}^n x_{ij} = 1$$

a la ciudad j se llega desde una sola ciudad



$$u_i - u_j + n x_{ij} \leq n - 1$$

restricción de eliminación de sub-tours

$$u_i, u_j \in \mathbb{R}, \quad x_{ij} \in \{0,1\}$$

$\approx n^2$ restricciones



Problema del agente viajero (TSP) (IV)

- A pesar de la sencillez del enunciado es uno de los problemas mas difíciles de resolver (es NP-difícil (*)) para casos de gran tamaño.
- Para un grafo de n ciudades la cantidad de posibles soluciones es:

$$\frac{(n-1)!}{2}$$

Ciudades	Rutas posibles
4	3
5	12
6	60
7	360
8	2.520
9	20.160
10	181.440
20	$\approx 60.000.000.000.000.000$

Problema del agente viajero (TSP) (V)

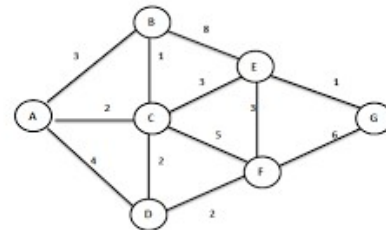
- Resolución:
 - ✓ **Fuerza Bruta:** Explorando todas las posibles soluciones y sus costes o distancias asociadas y quedándose con la mejor. No es viable a partir de cierto número de ciudades.
 - ✓ **Métodos exactos:** Realizando también exploraciones pero descartando ramas del árbol de soluciones porque se hagan evidentes que no conducirán a buenas soluciones (ramificación y poda)
 - ✓ **Métodos heurísticos:** Realizan mejoras a partir de soluciones iniciales pero no aseguran obtener el óptimo. Necesarios para tamaños grandes.

Problema del agente viajero (TSP) (VI)

- Ámbitos de aplicación:
 - ✓ Los evidentes en compañías de reparto
 - ✓ Circuitos electrónicos.
 - ✓ Planificación de tareas
 - ✓ Secuenciación de Genoma
 - ✓
- Variaciones:
 - ✓ Incorporación de varios viajeros
 - ✓ Restricciones con ventanas horarias
 - ✓ ...

Problema del camino más corto(I)

- No confundir con el **TSP** (travelling salesman problem).
- Se trata de encontrar el camino más corto(o menor coste) entre dos nodos de un grafo dirigido, no dirigido o mixto.
- Diferentes algoritmos dependiendo de :
 - Ponderaciones negativas
 - Grafos muy densos o poco densos
 - Encontrar todos los caminos más cortos desde/hasta un vértice dado



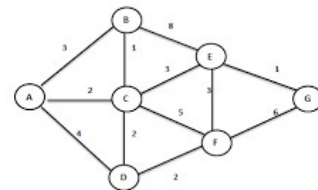
Problema del camino más corto(II)

- Algoritmos:
 - Algoritmo de Dijkstra :
 - Explora todos los caminos más cortos desde vértice origen a todos los demás vértices
 - No es válido con aristas con coste negativo.

https://arodrigu.webs.upv.es/grafos/doku.php?id=algoritmo_dijkstra
 - Algoritmo de Bellman – Ford:
 - Más lento que Dijkstra pero válido para ponderaciones negativas.

Johnson: <http://www.columbia.edu/~cs2035/courses/ieor6614.S16/johnson.pdf>
 - Búsqueda A^* (en la siguiente sección)
 - Uso de heurísticas
 - Algoritmo de Floyd – Warshall
 - Para obtener el camino mas corto entre todos los pares de vértices

https://arodrigu.webs.upv.es/grafos/doku.php?id=algoritmo_floyd_warshall



VC3 – Problema de la Mochila (KP, Knaspack problem)

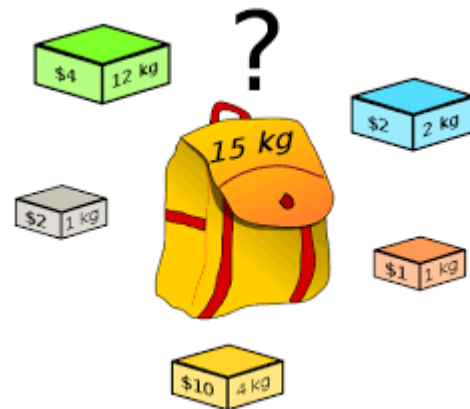
Agenda

1. Programación lineal
2. Programación lineal entera
3. Problema del agente viajero
4. Problema de la mochila
5. Otros problemas
6. Búsqueda en amplitud
7. Búsqueda en profundidad
8. Ramificación y Poda

Problema de la mochila (I)

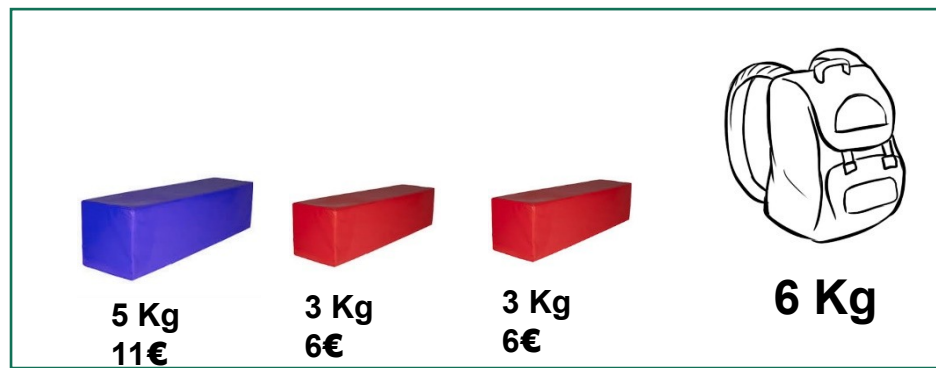
- El **KP** (knapsack problem) se enuncia como la manera óptima de seleccionar un subconjunto de objetos que tienen asociado un valor y un peso cada uno de tal manera que maximicen el valor pero cuyo peso no supere un cierto límite.
- El KP también es uno de los problemas clásicos debido a que es modelo de muchas situaciones en la práctica.
- También es un problema difícil de resolver para datos de entrada grandes.

$$\begin{aligned} \text{Max} \quad & \sum_{i=1}^n x_i \cdot v_i \\ \text{Sujeto a} \quad & \sum_{i=1}^n x_i \cdot w_i \leq M \\ & \text{con } x_i = 0, 1 \end{aligned}$$



Problema de la mochila (II)

- Resolución:
 - ✓ PLE. Es posible resolverlo como un problema de programación lineal entera.
 - ✓ Programación Dinámica y Vuelta atrás con $O(2^n)$ de complejidad
 - ✓ Métodos heurísticos:
 - ✓ Técnica Voraz no funciona!



Problema de la mochila (III)

- Aplicaciones:
 - ✓ Finanzas: equilibrio entre capital y rendimiento
 - ✓ Cargas en transporte de materiales (logística en general)
 - ✓ Corte de materiales
 - ✓ Robótica y Fabricación
- ✓ Similares:
 - ✓ Problema de la suma de subconjuntos(https://es.wikipedia.org/wiki/Problema_de_la_suma_de_subconjuntos)

Problema del enrutamiento (I)

- El **VRP** (vehicle routing problem) se enuncia como una variante del **problema del viajero(TSP)** pero en este caso no es un único agente el que debe realizar el recorrido sino que pueden ser varios.
- Como en el caso del TSP, es un problema complejo de resolver en el además se pueden incluir otras restricciones:
 - ventanas de tiempo
 - orígenes y/o destinos obligados para algunos o todos los agentes
- **Resolución:** Los mismos que en el TSP(fuerza bruta, búsqueda con ramificación y poda y heurísticos)
- Es un ámbito de estudio continuo.

VC3 – Otros Problemas Tipo

Agenda

1. Programación lineal
2. Programación lineal entera
3. Problema del agente viajero
4. Problema de la mochila
5. Otros problemas
6. Búsqueda en amplitud
7. Búsqueda en profundidad
8. Ramificación y Poda

Otros problemas

- Problema de la dieta (Stigler, 1930): minimizar el gasto cubriendo necesidades nutricionales.
- Problema de **satisfacibilidad booleana**(SAT). P.ej

Existe x_1, x_2, x_3, x_4 binarias (0,1) que den como resultado verdadero a:

$$(x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$$

- Problema de asignación: asignar tareas a agentes para maximizar producción
- Coloreado de Grafos:

- Compendio de problemas NP

<http://web.archive.org/web/20200218105451/http://www.nada.kth.se/~viggo/wwwcompendium/>

A compendium of NP optimization problems

Editors:

[Pierluigi Crescenzi](#), and [Viggo Kann](#)

Subeditors:

Magnús Halldórsson (retired)

Graph Theory: Covering and Partitioning, Subgraphs and Supergraphs, Sets and Partitions.

[Marek Karpinski](#)

Graph Theory: Vertex Ordering, Network Design: Cuts and Connectivity.

[Gerhard Woeginger](#)

Sequencing and Scheduling.

VC3 – Algoritmos de Búsqueda

Agenda

1. Programación lineal
2. Programación lineal entera
3. Problema del agente viajero
4. Problema de la mochila
5. Otros problemas
- 6. Búsqueda en amplitud**
7. Búsqueda en profundidad
8. Ramificación y Poda

Algoritmos de Búsqueda

- **Definición:** Técnicas que persiguen la localización de uno o varios elementos con ciertas propiedades dentro de la estructura de datos.
- Dado que una cantidad importante de problemas se pueden modelar como estructuras de datos tipo árbol, es importante conocer técnicas para explorar arboles.
- Técnicas principales:
 - Búsqueda en amplitud o anchura
 - Búsqueda en profundidad
 - Ramificación y Poda

Algoritmos de Búsqueda. Búsqueda en amplitud (I)

- La búsqueda en amplitud o en anchura o **BFS** (breadth first search) se basa en analizar todos los nodos del árbol de un mismo nivel antes de seguir con los del siguiente nivel.
- Esquema general:
 1. Formar una cola(*) con el elemento raíz del árbol.
 2. Probar si el primer elemento de la cola es solución final. En caso afirmativo, terminamos. En caso contrario, seguimos con el paso 3.
 3. Quitar el nodo explorado en el paso 2 y añadir al final de la cola todos sus nodos hijos.
 4. Si en la cola no hay nodos, entonces no podemos encontrar solución. En caso contrario, repetimos el paso 2.

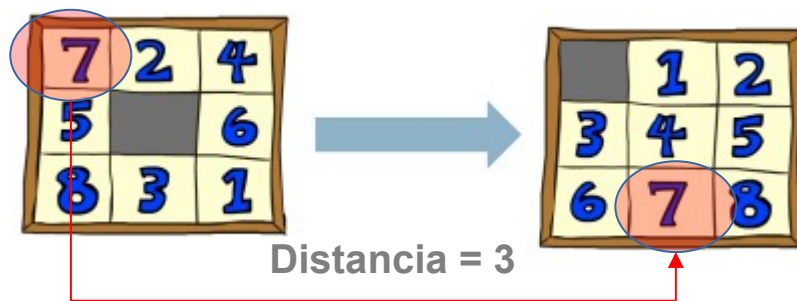
Algoritmos de Búsqueda. Búsqueda en amplitud (II)

- Algoritmo A*
- Es una variación de búsqueda en amplitud a través de la introducción de un componente heurístico en la valoración de los nodos para determinar si la exploración por dicho nodo merece o no la pena.
- Formalmente la función de evaluación es : $f(n) = g(n) + h(n)$ donde
 - $f(n)$ es la función que evalúa cada nodo
 - $g(n)$ es el coste actual desde la raíz hasta el nodo explorado
 - $h(n)$ es un **valor heurístico** del coste desde el nodo actual hasta el final

Algoritmos de Búsqueda. Búsqueda en amplitud (III)

Algoritmo A*. Ejemplo de heurísticas:

- 1) Número de fichas mal colocadas
- 2) Distancia Manhattan de todas las piezas: $3 + 1 + 2 + \dots$



Algoritmos de Búsqueda. Búsqueda en amplitud (IV)

- **Valor heurístico**
- “Una heurística es una regla para engañar, simplificar o para cualquier otra clase de ardid el cual limita drásticamente la búsqueda de soluciones en grandes espacios de estados”.(Feigenbaum y Feldman)
- Un conjunto de **reglas** que según el problema permiten construir una función para **evaluar la mejor posibilidad** entre un conjunto de ellas

VC3 – Algoritmos de Búsqueda

Agenda

1. Programación lineal
2. Programación lineal entera
3. Problema del agente viajero
4. Problema de la mochila
5. Otros problemas
6. Búsqueda en amplitud
7. **Búsqueda en profundidad**
8. Ramificación y Poda

Algoritmos de Búsqueda. Búsqueda en profundidad (I)

- La búsqueda en profundidad o **DFS** (depth first search) se basa en analizar primero todos los nodos más profundos del árbol.
- Esquema general:
 1. Formar una cola(*) con el elemento raíz del árbol.
 2. Probar si el primer elemento de la cola es solución final. En caso afirmativo, terminamos. En caso contrario, seguimos con el paso 3.
 3. Quitar el nodo explorado en el paso 2 y añadir al principio de la cola todos sus nodos hijos.
 4. Si en la cola no hay nodos, entonces no podemos encontrar solución. En caso contrario, repetimos el paso 2.

Algoritmos de Búsqueda. Búsqueda en profundidad (II)

- Ascenso de colina(hill climbing)
- Es una variación de búsqueda en profundidad a través de la introducción de un componente heurístico para tratar de ordenar los nodos hijos introducidos según merezcan más o menos la pena ser explorados.
- Formalmente la función de evaluación es : $f(n) = g(n) + h(n)$ donde:
 - $f(n)$ es la función que evalúa cada nodo
 - $g(n)$ es el coste actual desde la raíz hasta el nodo explorado
 - $h(n)$ es un **valor heurístico** del coste desde el nodo actual hasta el final

VC3 - Ramificación y Poda

Agenda

1. Programación lineal
2. Programación lineal entera
3. Problema del agente viajero
4. Problema de la mochila
5. Otros problemas
6. Búsqueda en amplitud
7. Búsqueda en profundidad
8. Ramificación y Poda

Algoritmos de Búsqueda. Técnicas de Ramificación y Poda(I)

- Durante la exploración exhaustiva es posible **eliminar definitivamente ramas del árbol** que sabemos a ciencia cierta que no van mejorar la solución encontrada.
- En cada nodo se establece una **cota** para todas las soluciones alcanzables desde dicho nodo.
- Implementación en 3 etapas:
 - **Selección** de uno de los nodos pendiente de analizar
 - **Ramificación** de todos los nodos hijo
 - **Poda** o eliminación de los nodos que no mejoran
- La clave es encontrar la **función de coste** para estimar la **cota** desde cada nodo

Algoritmos de Búsqueda. Técnicas de Ramificación y Poda(II)

- Estimación de cota para una solución parcial
 - ✓ Antes de explorar el nodo S, intentamos acotar el beneficio de la mejor opción que “cuelga” de S

$$CI(s) \leq Valor(M) \leq CS(s)$$

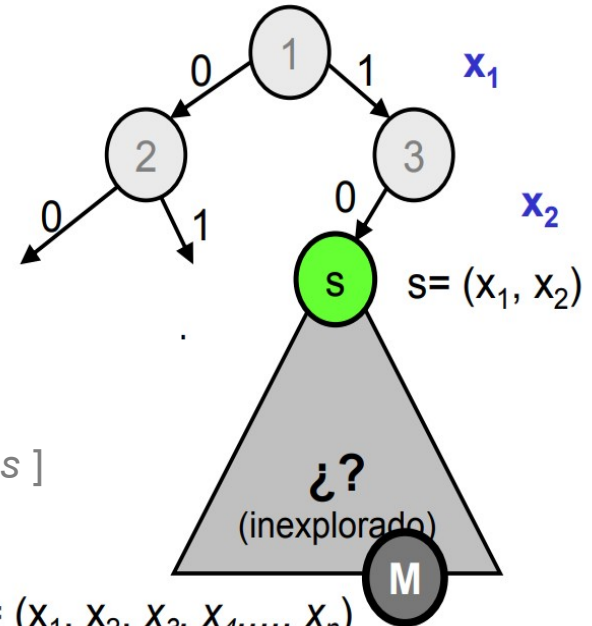
CI(s) = Cota Inferior

CS(s) = Cota Superior

BE(s) = Beneficio esperado

[Las cotas deben ser fiables o podaremos nodos válidos]

[BE ayudará a decidir que parte evaluamos primero]

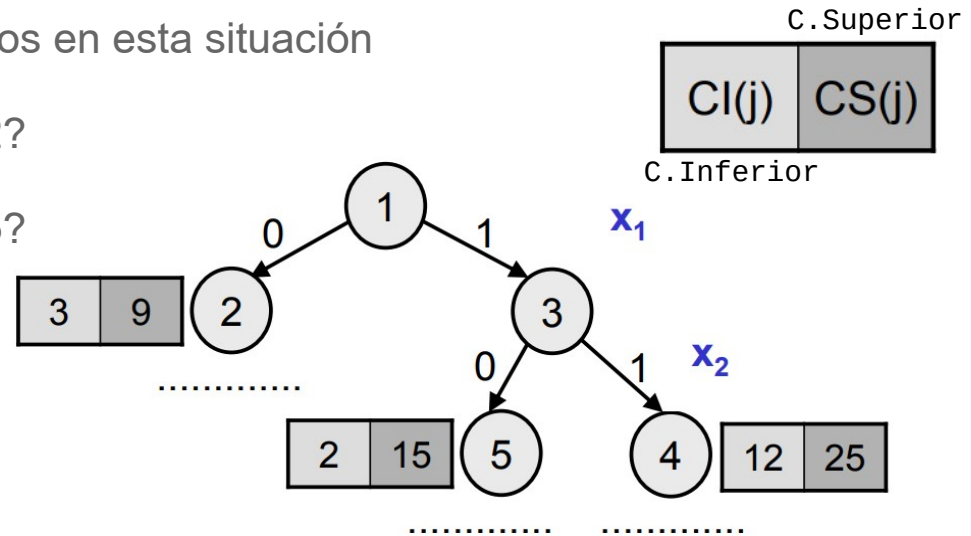


$M = (x_1, x_2, x_3, x_4, \dots, x_n)$

$Valor(M) = ?$

Algoritmos de Búsqueda. Técnicas de Ramificación y Poda(III)

- Estrategia de poda
 - ✓ Ejemplo para maximización. Estamos en esta situación
 - ✓ ¿Merece la pena explorar el nodo 2?
 - ✓ ¿Merece la pena explorar el nodo 5?

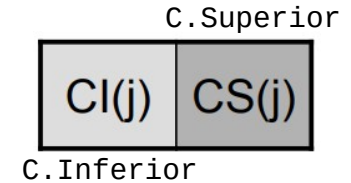


Algoritmos de Búsqueda. Técnicas de Ramificación y Poda(IV)

- Estrategia de poda:

✓ Podar en nodo i si:

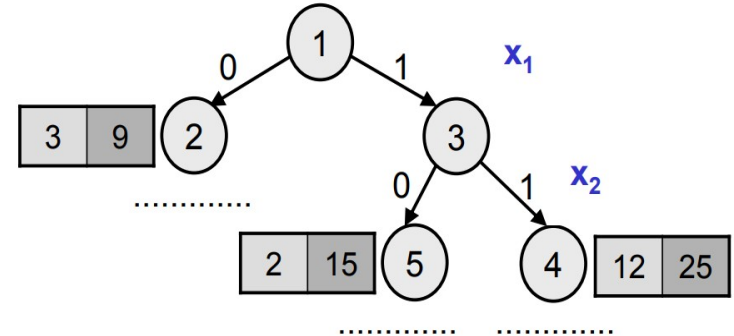
$CS(i) \leq CI(j)$ Para algún j ya generado



- Implementación:

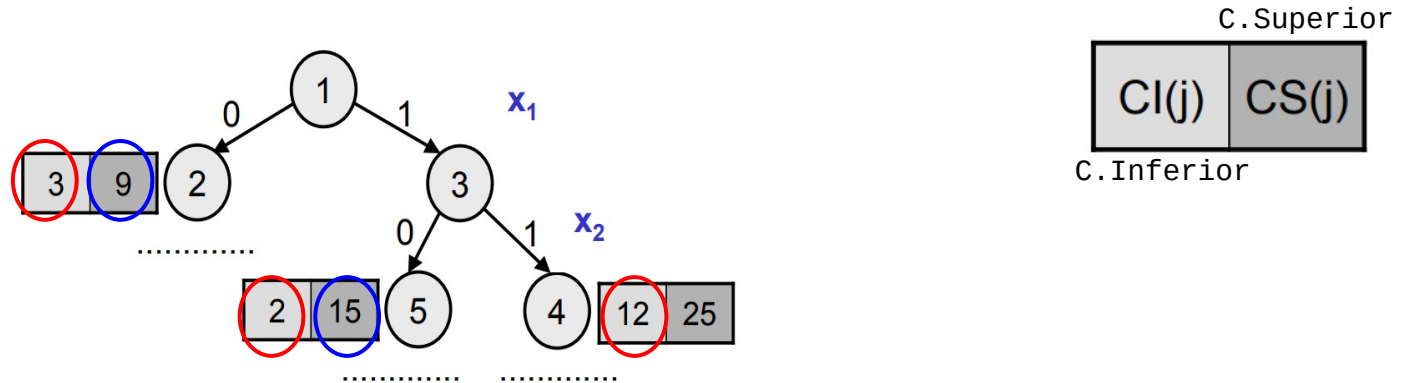
$C = \max(\{CI(j) \mid \forall j \text{ generado}\}, \{Valor(s) \mid \forall s \text{ solución final}\})$

- Podar i si: $CS(i) \leq C$



Algoritmos de Búsqueda. Técnicas de Ramificación y Poda(V)

- Estrategia de poda: Poda i si $CS(i) \leq CI(j)$ Para algún j ya generado



C=12(max. de las cotas inferiores)

El nodo 2 no puede mejorar(valor 9) las peores predicciones de 4(valor 12 = $\max\{\text{todo } CI() \}$)

El nodo 5 quizá pueda mejorar al nodo 4

Algoritmos de Búsqueda. Técnicas de Ramificación y Poda(VI)

- **Propiedades**
 - ✓ Se puede ver como una mejora de la técnica de Vuelta Atrás (4 reinas)
 - ✓ No es necesario almacenar todo el árbol de soluciones. Sino que se guarda un lista con los “nodos vivos” (nodos generados pero que aun no han sido evaluados)
 - ✓ Se utiliza en juegos completos con adversario(MiniMax)

Un buen artículo (*historia, ¿por qué funciona?, cubo de Rubik*):

<https://rjlipton.wordpress.com/2012/12/19/branch-and-bound-why-does-it-work/>

Algoritmos de Búsqueda.

- **MinMax(*)**. Juegos con adversario
- Se trata de la aplicación de algoritmos de búsqueda en juegos completos con adversario(*)
- Poda alfa-beta. Se podan las ramas que se prevé no van a ser buenas soluciones para ambos jugadores

Algoritmos de Búsqueda.

- **Resumen**

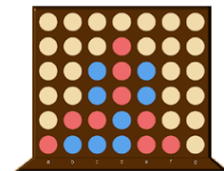
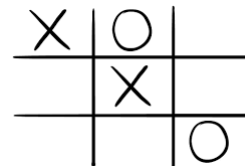
- En general los algoritmos de búsqueda son adecuados cuando el espacio de soluciones posibles es acotado. Dejan de tener efectividad para arboles muy extensos.



- Ej. Tratar de resolver el juego del ajedrez por estos métodos resulta un poco ingenuo dado la **cantidad de posibilidades para los movimientos**.



- Sin embargo si es asequible resolverlos para juegos como 3 en raya o 4 en raya(conecta-4).



Algoritmos de Búsqueda. 8-Puzzle

- Es un puzzle que consiste en un área dividida en una cuadrícula, 3 por 3. Hay ocho fichas en y una vacía. Una ficha que está al lado del cuadrado vacío se puede mover al espacio vacío, dejando por tanto su posición vacía a su vez. Las fichas están numeradas, del 1 al 8 de modo que cada ficha se puede identificar de forma única.
- El objetivo del rompecabezas es lograr la configuración meta de fichas a partir de una configuración inicial con movimientos descritos .

Estado inicial

1	2	3
7	8	4
6		5

Estado meta

1	2	3
8		4
7	6	5



¡Ojo! Hay disposiciones iniciales que no dan lugar a la solución final(la mitad)

Foro.

Armas de destrucción matemática – Cathy O’Neil



Raul Reyero Diez ★

Hace 3 minutos

Algoritmos. Armas de destrucción matemática. Cathy O’Neil

Dado que hemos empezado ya la asignatura quiero plantearos un debate para ser tratado no tanto desde el punto de vista técnico sino ético o filosófico.

¿Estamos haciendo lo correcto, desde otros puntos de vista que no son los puramente técnicos, con "delegar" en los algoritmos los análisis y decisiones importantes?

¿Debemos preocuparnos como una sociedad justa e igualitaria a la que deberíamos aspirar por la puesta en práctica en situaciones que nos afectan en primera persona como la conducción autónoma, diagnósticos médicos, concesión de créditos, selección de personal y otros tantos?

En el libro "Armas de destrucción matemática", Cathy O’Neil nos cuenta algunas situaciones que nos van a hacer, al menos reflexionar, inevitablemente. Podéis encontrarlo en algunas bibliotecas populares.
Hay un buen resumen en:

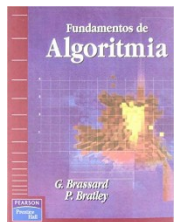
<https://www.revistadelibros.com/resenas/armas-de-destruccion-matematica-cathy-oneil>

Reflexiona sobre este asunto, **busca** información sobre algunas opiniones de diferentes perfiles (políticos, técnicos, filosóficos, afectados,...) y **comparte** tu reflexión.

Responder



Bibliografía



Fundamentos de algoritmia: Una perspectiva de la ciencia de los computadores

Paul Bratley , Gilles Brassard

ISBN 13: 9788489660007



Introducción al diseño y análisis de algoritmos

R.C.T. Lee,...

ISBN 13: 9789701061244



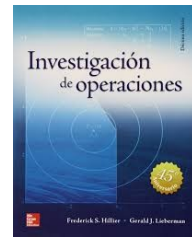
Técnicas de diseño de algoritmos

Guerequeta, R., y Vallecillo, A.

<http://www.lcc.uma.es/~av/Libro/indice.html>

Bibliografía. Programación lineal

- Hillier, F. S., y Lieberman, G. J. (2015): McGraw-Hill Education.



- Kong, Maynard: Investigación de operaciones:
programación lineal, problemas de transporte, análisis de redes(*)

<https://elibro-net.universidadviu.idm.oclc.org/es/ereader/universidadviu/79351?page=1>



Próxima Clase. Actividad Guiada

1. Desarrollar algoritmos voraces para resolver problemas
2. Desarrollar algoritmos con la técnica de vuelta atrás(backtracking) para resolver problemas
3. Desarrollar algoritmos con la técnica de divide y vencerás para resolver problemas
4. Desarrollar algoritmos con la técnica de programación dinámica para resolver problemas

¿Preguntas?



Gracias

raul.reyero@campusviu.es