Reflexión previa: ¿inteligencia razonable o inteligencia empírica?

Los neófitos que nos topamos con el dilema de las vacas lecheras no podemos evitar caer en la tentación intuitiva de hallar la solución en el índice de productividad de cada vaca. Con dicho índice, aventuramos, haremos un ranking de las más productivas el cual nos dará la clave para cargar el camión y obtener, sin mucha duda, la combinación de reses más productiva. Este razonamiento salta por los aires tan solo en unas pocas pruebas hecha con tan solo media docena de reses; a mano, con lápiz y papel. Efectivamente, a la luz de una mayor atención al problema descubrimos que, en cierto punto, puede ser más determinante la producción bruta de una res que su índice de producción (el cociente litros y peso). Algo nos dice que nuestra solución no es tan racional como pensamos y que tan solo podremos calificar de *razonable,* a priori.

Una breve búsqueda en la web nos pone en alerta ya que la solución pasa por algoritmos más complejos, dejando nuestras altas esperanzas iniciales en nada o casi nada. Y es que la mera idea de que la solución pase por una iteración de todas las combinaciones nos hace pensar en una inteligencia sin racionalidad, en un mero cotejo exhaustivo de todas las combinaciones que finaliza en una lista ordenada en la que en su cima se hallará la solución al problema planteado. Confiar la solución a lo empírico, es decir, que no hay razón sino hechos, en línea con el empirismo actual en el que arraiga la AI y aun de la ciencia moderna,

Así que antes de dar nuestro brazo a torcer, quisiéramos ver hasta dónde nos lleva la que denominamos *solución razonable* (la cual también podríamos perfectamente denominar "solución ingenua"). Buscaremos por ello una conjetura que nos de cuenta de la, a primera vista, mejor explicación de porqué la solución se construye con razones y no con cotejos exhaustivos. Esa conjetura se basa en la convicción de que detrás de las cosas hay una razón, y no simplemente hechos y algoritmos eficaces; es un pensamiento con tufo a racionalismo rancio, lo sabemos, pero démonos el gusto de ir aún un poco más allá.

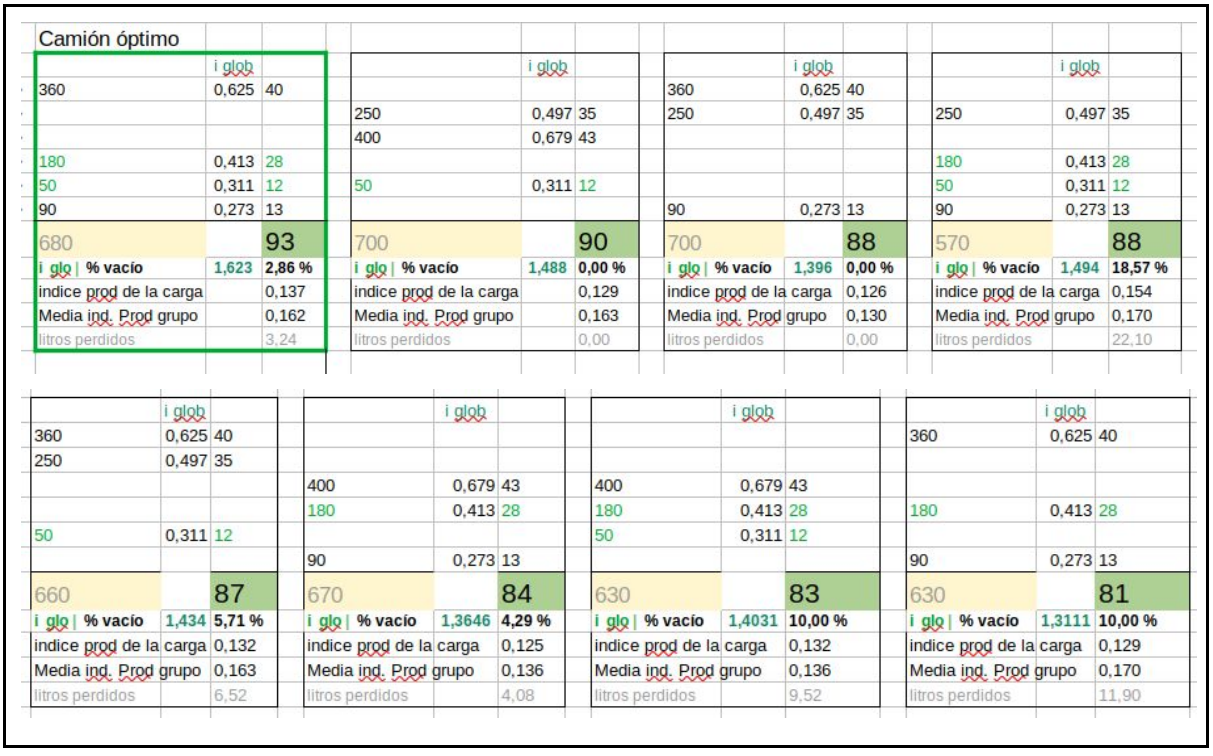
PARTE I: La solución razonable (o ingenua)

Como ya hemos apuntado, nuestra conjetura se basa, tercamente, en otorgar relevancia a la razón, es decir a la *ratio*, en este caso en forma de *índice de producción*, *índice de peso* y, como complemento necesario, a la *aportación bruta* de leche de cada res. Creemos que esos índices están en la base de la conformación del grupo óptimo de vacas y eso es lo que nos hemos propuesto demostrar. Esos índice serán los que nos soporten la conjetura porque, y esta es nuestra tesis central, ella no era una elección intuitiva ni caprichosa. Asi hemos dado con la siguiente tabla en la que se recogen los tres indice anteriormente referidos y en el que hemos anotado también un *índice global,* resultante de las suma de los *índices de producción* y *de peso.*

peso	prod	i- l/kg	l % aport	i glo	l % PM	i comp
360	40	0,111	0,234	0,625	0,514	
250	35	0,140	0,205	0,497	0,357	
400	43	0,108	0,251	0,679	0,571	
180	28	0,156	0,164	0,413	0,257	
50	12	0,240	0,070	0,311	0,071	
90	13	0,144	0,076	0,273	0,129	

Con los datos enfrente, de forma rápida, vemos que las tres vacas que más leche dan en bruto y que mayor índice de aportación tienen no pueden entrar en el camión conjuntamente; una al menos ha de quedar fuera. Por descarte nos quedamos con dos y, seguidamente, completamos el camión de reses lecheras con otra vaca con índice de producción notable y cuyo peso no haga exceder el máximo permitido. Hemos llegado a una solución satisfactoria que logra llenar el camión con los 700 kg máximos permitidos y con una cifra de producción de leche nada desdeñable de 90 litros.

No contentandonos con ello, y sabiendo que el reto encierra más miga de la que parece, hemos hecho unos cálculos en tabla Excel, sacando algunas de ela combinaciones más productivas. Ahora vemos como la solución primera no es la mejor: resulta que el *índice de producción* marca su valía y vemos que los tres vacas con mayor índice, completadas con una res de gran peso y poco *índice de producción* pero gran producción bruta, como es la vaca 1, son la solución óptima con 93 litros de producción.



Las condiciones tienen que ver con el *índice global*, claro, pero con un valor no menor al porcentaje de peso que la vaca en cuestión representa dentro del total de carga del camión. También tiene un papel decisivo el peso sin carga que deja la combinación de reses considerada. Sería cuestión de ajustar estos valores de decisión con pruebas varias (experiencia, evidencia empírica, estadística o decir generalización a partir del casos dados) pero en un principio hemos estimado en 2,5% y 96% respectivos para las dos condiciones referidas.

Para dar continuidad a nuestro experimento, habremos de sacar algún patrón y algunas conclusiones razonables:

- En la configuración óptima de reses en el camión es importante el *índice de producción* de cada una así como el *índice medio de producción* de las vacas que componen el cargamento (1,623)
- En cierto punto, cuando el camión está apunto de llenarse, hay un momento en que puede pasar a ser decisivo el aporte bruto de leche de una res.
- Existen otros datos que podrían ser relevantes en una solución óptima como puede ser que la solución no deje mucho espacio sin asignar en el camión (un 2,86% en el caso óptimo según la prueba nuestra)

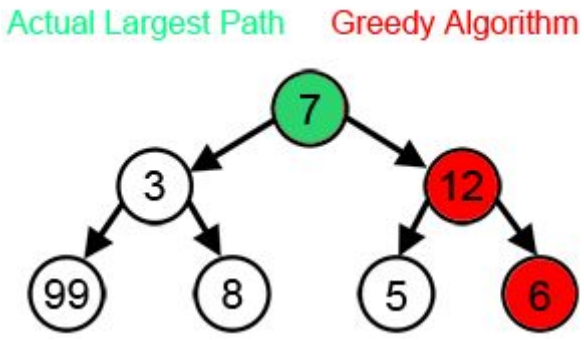
Como es palmario, estos son unos rasgos sacados a posteriori de un mapa de casos, es un patrón que se reproduce en un mundo en exceso reducido (6 vacas) y que no podemos asegurar se cumplan en un caso con decenas de reses en juego.

Decidimos pues crear un script que, configurado para dar prioridad a estos dos criterios (*índice de producción* e *índice global* (peso + producción) nos arroje automáticamente la solución óptima. El resultado por pantalla de dicho script nos ofrece lo siguiente:

```
runfile('/home/alberto-mac/Documentos/theegg_ai-master/Tarea_22/Algoritmo_del_lechero_calculo
_razonable-hilo1.py', wdir='/home/alberto-mac/Documentos/theegg_ai-master/Tarea_22')
INICIO DEL SCRIPT - HILO 1 ////////////////////////////////////////////////////////
////////////////////////////////////////////////////
Ind Prod:  0.111
Ind Prod:  0.14
Ind Prod:  0.107
Ind Prod:  0.156
Ind Prod:  0.24
Ind Prod:  0.144
Ind PMA:  0.514
Ind PMA:  0.357
Ind PMA:  0.571
Ind PMA:  0.257
Ind PMA:  0.071
Ind PMA:  0.129
Ind Glob:  0.625
Ind Glob:  0.497
Ind Glob:  0.678
Ind Glob:  0.413
Ind Glob:  0.311
```

```
Ind Glob:  0.273
[['v1', 360, 40, 0.111, 0.514, 0.625], ['v2', 250, 35, 0.14, 0.357, 0.497], ['v3', 400, 43,
0.107, 0.571, 0.678], ['v4', 180, 28, 0.156, 0.257, 0.413], ['v5', 50, 12, 0.24, 0.071,
0.311], ['v6', 90, 13, 0.144, 0.129, 0.273]]
//////////////////////////////////////////////////////////
HILO 1 - ////////////////////////////////////////////////
Ordenamos el array en función del INDICE GLOBAL:
[['v3', 400, 43, 0.107, 0.571, 0.678], ['v1', 360, 40, 0.111, 0.514, 0.625], ['v2', 250, 35,
0.14, 0.357, 0.497], ['v4', 180, 28, 0.156, 0.257, 0.413], ['v5', 50, 12, 0.24, 0.071,
0.311], ['v6', 90, 13, 0.144, 0.129, 0.273]]
Candidato: v3  y su peso:  400
DESCARTADO el candidato: v1  y su peso:  360
Peso acumulado al camión:  400
Candidato: v2  y su peso:  250
DESCARTADO el candidato: v4  y su peso:  180
Peso acumulado al camión:  650
//////////////////////////////////////////////////////////
Peso acumulado del paso anterior:  650
Ordenamos el array en función del ÍNDICE DE PRODUCCIÓN:
candidato: v5  y su peso  50
Peso acumulado al camión:  700
candidato: v4  y su peso  180
Peso acumulado al camión:  880
El último animal excedería el peso hasta  880  kg. por lo que se descuenta del cargamento
//////////////////////////////////////////////////////////
SOLUCIÓN. ///////////////////////////////////////////////
Vacas escogidas:  ['v3', 'v2', 'v5']
Producción de leche:  90
Peso acumulado FINAL en el camión:  700
//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////
Esta solución puede no ser la óptima, ¿probamos con otra posibilidad no explorada? (responde
SI o NO):
```

Como se ve la solución que reporta es la misma a la que llegamos en nuestros cálculos manuales. Esta solución ingenua tiene mucho de lo que se llama ***Greedy algorithms*** (también conocido como *"algoritmo codicioso"*) aquellos algoritmos que con el objetivo de alcanzar la suma más grande, en cada paso, egirá lo que parece ser la opción inmediata óptima, por lo que, como se muestra en la figura 1, elegirá 12 en lugar de 3 en el segundo paso, y no alcanzará la mejor solución, que contiene 99.



Por eso mismo hemos hecho que en la última línea de la salida de pantalla del script se ofrezca la opción de cargar el segundo hilo del script en el que se realiza el cálculo a partir de las reses que, teniendo alto índice de producción, fueron descartadas en este primer cribado. Si respondemos SÍ a la pregunta, dicho hilo se ejecutará y nos ofrecerá otra solución:

```
Python 3.8.5 (default, Sep  4 2020, 07:30:14)
Type "copyright", "credits" or "license" for more information.
IPython 7.19.0 -- An enhanced Interactive Python.
SI
INICIO DEL SCRIPT - HILO 2 ////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
Ind Prod:  0.111
Ind Prod:  0.14
Ind Prod:  0.107
Ind Prod:  0.156
Ind Prod:  0.24
Ind Prod:  0.144
Ind PMA:  0.514
Ind PMA:  0.357
Ind PMA:  0.571
Ind PMA:  0.257
Ind PMA:  0.071
Ind PMA:  0.129
Ind Glob:  0.625
Ind Glob:  0.497
Ind Glob:  0.678
Ind Glob:  0.413
Ind Glob:  0.311
Ind Glob:  0.273
[['v1', 360, 40, 0.111, 0.514, 0.625], ['v2', 250, 35, 0.14, 0.357, 0.497], ['v3', 400, 43,
0.107, 0.571, 0.678], ['v4', 180, 28, 0.156, 0.257, 0.413], ['v5', 50, 12, 0.24, 0.071,
0.311], ['v6', 90, 13, 0.144, 0.129, 0.273]]
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
Ordenamos el array en función del ÍNDICE GLOBAL:
Descartado el v3 por no ser la vaca 1 400
Peso acumulado al camión:  0
Vaca seleccionada para cargar al camión:  v1  y su peso:  360
['v1']
360
Peso acumulado al camión en esta ronda:  360
HILO 2 - ////////////////////////////////////////////////////////
Peso acumulado del paso anterior:  360
Ordenamos el array en función del ÍNDICE DE PRODUCCIÓN:
candidato: v5  y su peso  50
candidato: v4  y su peso  180
candidato: v6  y su peso  90
candidato: v2  y su peso  250
El último animal excedería el peso hasta  930  kg. por lo que se descuenta del cargamento
////////////////////////////////////////////////////////
SOLUCIÓN. ////////////////////////////////////////////////
Vacas escogidas:  ['v1', 'v5', 'v4', 'v6']
Producción de leche:  93
Peso acumulado FINAL en el camión:  680
```

```
//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
```

Estos dos últimos scripts se hallan, en dos documentos separados (“*solucion_vacas_solucion_ingenua-hilo1.py*” y “*solucion_vacas_solucion_ingenua-hilo2.py*”), en el directorio de la tarea 22 de Github.

Como se aprecia este último hilo del algoritmo da con la solución óptima. No podemos hacer alarde de nada ya que es evidente que la solución está un poco “cocinada”. Es ineludible buscar una forma que nos de la solución al 100% y de una sola tacada.

PARTE II: La solución estilo “fuerza bruta”

En una solución tipo experimental, empírica, por fuerza bruta, se realizan todas las combinaciones de agrupamientos de vacas, se suman sus producciones conjuntas y el agrupamiento que de más litros será el elegido. La función *max* de Python nos vale para, sobre una iteración de la lista completa, arrojar el valor más alto:

```
runfile('/home/alberto-mac/Documentos/theegg_ai-master/Tarea_22/solucion_vacas_Fuerza-br
uta.py', wdir='/home/alberto-mac/Documentos/theegg_ai-master/Tarea_22')
1330 171
Peso 680 Valor 93
[('vaca 1', 360, 40), ('vaca 4', 180, 28), ('vaca 5', 50, 12), ('vaca 6', 90, 13)]
```

El script que da esta salida de pantalla (“*solucion_vacas_Fuerza-bruta.py*”) está en el directoria de la tarea 22 de Github.

Para un número pequeño de combinaciones, es suficiente pero para listas grandes no. En este caso de 6 vacas el algoritmo lo hace en 79 iteraciones pero con 54 vacas de producción y peso muy similares entre sí tarda 16 segundos y requiere de 50049314 iteraciones. Curiosamente con los mismos ítems (54 vacas) pero con menos repeticiones entre los distintos pesos y litros, el algoritmo realiza más iteraciones: 61881532. Alterando a la baja solo los litros de cada vaca, sin tocar los peso, respecto a la anterior prueba, el resultado de producción, como es de esperar, es menor: 244 pero las iteraciones son las mismas: 61881532, tardando en ambos vascos 20 segundos.

La cuestión está al parecer en que en listas grandes las combinaciones suben exponencialmente y la complejidad del proceso se vuelve inmanejable. A partir de ciertas cantidad de elementos, este tipo de algoritmos crecen de forma exponencial, con 400, las combinaciones son 2^{400} lo que supone un número demasiado largo. En esto es donde entra en juego la notación *O grande* y lo que se conoce como análisis y optimización de algoritmos.

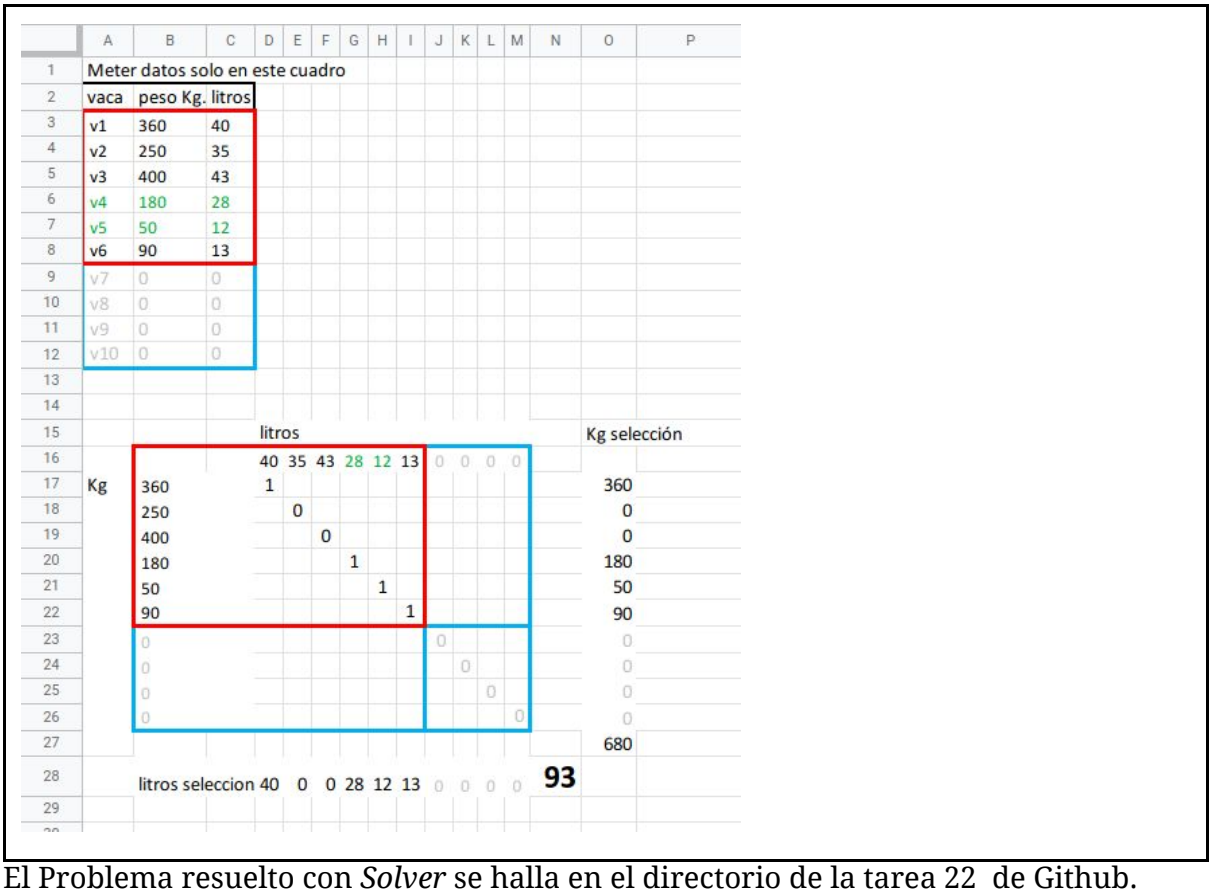
PARTE III: Programación lineal

La teoría de la programación lineal reduce drásticamente el número de posibles soluciones factibles que deben ser revisadas representando una mejor opción que las soluciones de fuerza bruta. Requerimos de una maypr introspección en este tipo de soluciones. tan solo diremos que se usan de forma habitual en entornos de ingeniería y economía para buscar la optimización de producción, costes, etc. Un ejemplo es la maximización, algo similar al caso que nos atañe. La base matemática de la programación lineal se basa en una función objetivo como:

$$Max! = \sum_{i=1}^{N} f_i \times X_i$$

Siendo ***A*** el valor de la restricción conocido a ser respetado estrictamente, ***a*** es el coeficiente técnico conocido, ***X*** son las incógnitas, de 1 a N e ***i*** es número de la incógnita (variable de 1 a N).

A continuación se muestra un ejemplo de solución del problema de las vacas con *Solver*, la herramienta de análisis de Excel. No lo hemos conseguido documentar pero creo que se basa en el *método simplex* de programación lineal.



El Problema resuelto con *Solver* se halla en el directorio de la tarea 22 de Github. (“*solucion_vacas_program_lineal_solver.xlsx*”)

PARTE IV: Programación dinámica

La programación dinámica se apoya en diversos enfoques para reducir la complejidad del problema, haciendo uso de recursos computacionales como la *memoizazion*, una técnica para agilizar los programas informáticos y cálculos mediante el almacenamiento de resultados de cálculos de funciones voraces. Un segundo recurso que utiliza es la descomposición de los problemas en diferentes subproblemas, con dos aproximaciones opcionales: *Top-down* o *Bottom-u*p. En el primero el problema se divide en subproblemas y estos se resuelven y se guardan sus sus cálculos (aquí se apoya en la *memoizacion* anteriormente descrita) por si fueran necesarias nuevamente. En el enfoque *Bottom-up* "todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado" (citado literal de la entrada de la Wikipedia).

La solución al problema que nos ocupa se puede comprender de manera increíblemente accesible en el libro "*Grokking algorithms. An illustrated guide for programmers and other curious peopl*e". Siguiendo sus explicaciones hemos podido hacer, con papel y bolígrafo, la la matriz de resolución dinámica del problema de las vacas:

Ahora estamos en disposición de comprender de forma muy didáctica la forma en que la programación dinámica subdivide el problema en problema más pequeños y avanza aprovechado la solución (siempre más sencilla) a esos subproblemas para escalar la soluciones parciales a una solución última. Estamos ahora en condiciones

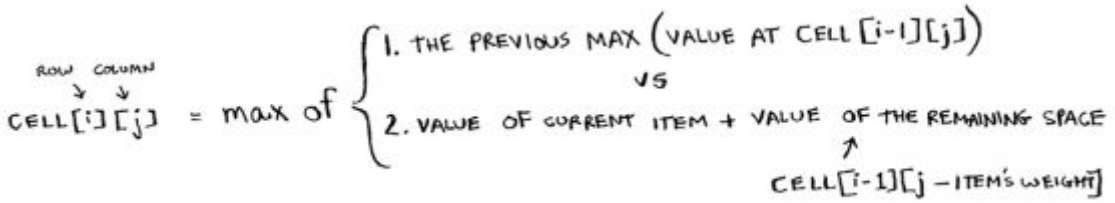
de entender la construcción dinámica de la matriz que se hace en el siguiente código cortesía de Jayant Verma (acomodado al caso de las vacas).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Jan  3 18:54:58 2021

@author:Jayant Verma en  https://www.askpython.com/author/jayant
acomodado al caso por alberto-mac
"""

val = [40,35,43,28,12,13]
wt = [360,250,400,180,50,90]
W = 600

def knapSack(W, wt, val):
        n=len(val)
        table = [[0 for x in range(W + 1)] for x in range(n + 1)]
        for i in range(n + 1):
        for j in range(W + 1):
                if i == 0 or j == 0:
                table[i][j] = 0
                elif wt[i-1] <= j:
                table[i][j] = max(val[i-1]  + table[i-1][j-wt[i-1]],  table[i-1][j])
                else:
                table[i][j] = table[i-1][j]

        return table[n][W]

print(knapSack(W, wt, val))
```

Este script ("*solucion_vacas_program_dinamica*.py") está en el directoria de la tarea 22 de Github. Código original tomado de https://www.askpython.com/author/jayant

PARTE IV: Conclusión

El recorrido realizado solo nos ha hecho ser un poco conscientes de lo que hay detrás de la programación dinámica y de las técnicas de optimización de algoritmos; queda mucho por descubrir. A pesar de que lo aprendido nos ha desbaratado gran parte de nuestras expectativas iniciales depositadas en el planteamiento ingenuo, sí que apreciamos que se mantiene una diferencia entre una proceder razonable y un proceder exhaustivo. La solución dinámica no solo evita el inconveniente de la solución exhaustiva (fuerza bruta) a partir de cierto número de items, sino que también atesora un proceder " inteligente". Nos explicamos: el núcleo "inteligente" de la función que da con la solución es el siguiente:



la función max devuelve el máximo de una sencilla comparación entre la pseudo solución que representa la anterior celda y el valor accedido en curso más el valor que puede albergar el espacio un vacío por delante. Con eso queremos destacar que las "decisiones inteligentes" que se realizan sobre la matriz dinámica no dejan de ser unas meras comparaciones de valores, sin embargo el diseño del procedimiento dinámico que lo hace posible trasluce lo que reconocemos como la marca inconfundible de la inteligencia: no pocas dosis de creatividad, la ocurrencia de ver las cosas desde sus lados menos habituales, la de los modos imaginativos de desvelar la matemática subyacente a los hechos etc. Esa marca que al menos a día de hoy solo aparece a partir de un agente humano.

Referencias

Dynamic programming in Python:
https://www.askpython.com/python/examples/knapsack-problem-dynamic-programming

Solución de problemas con algoritmos y notación O Grande:
https://runestone.academy/runestone/static/pythoned/AlgorithmAnalysis/QueEsAnalisisDeAlgoritmos.html

Algoritmos:
Aditya Y. Bhargava (2016) *"Grokking algorithms. An illustrated guide for programmers and other curious people"*, Manning Publications Co. NY 11964