



5

Instituto Politécnico Nacional
Escuela Superior de Computo

Análisis de algoritmos

Sandra Díaz Santiago

Practica 6: Algoritmos ávidos III

García González Aarón Antonio

Noviembre 11, 2019



Índice

Ejercicio 1.....	3
Avance en laboratorio	3
Implementación.....	4
Funcionamiento.....	5
Ejemplo	6
Ejecución.....	6
Justificación.....	6

Ejercicio 1

El problema de la mochila entera pide encontrar una manera de llenar una mochila de cierta capacidad con algunos elementos que pertenecen a un conjunto dado. De tal manera que se obtenga la mayor ganancia. La entrada del problema es la siguiente:

- a) C : la capacidad total de la mochila.
- b) N : el número de elementos
- c) Un vector Q , donde $Q[I]$ es el número de elementos disponibles de tipo I
- d) Un vector W , donde $W[I]$ es el peso de cada elemento de tipo I , tal que $0 < W[I] \leq C$
- e) un vector P donde $P[I]$ es la ganancia obtenida al tener el elemento I en la mochila.

El problema consiste en guardar elementos en la mochila de tal manera que el peso de estos no exceda la capacidad de la mochila, de tal manera que la ganancia total sea máxima.

La salida debe ser el vector $F[I]$ que contiene el número de elementos de tipo I que se guardan en la mochila. Diseña un algoritmo ávido para solucionar el problema anterior. Justifica por qué tu algoritmo funciona y si la solución que obtienes es óptima o no. Implementa tu solución en tu lenguaje de programación favorito.

Avance en laboratorio

Sesión 6 Algoritmos Ávidos III
Problema de la mochila

C → capacidad de la mochila.
 N → número de elementos.
 Q → vector, donde $Q[I]$ es el número de elementos disponibles tipo I
 W → donde $W[I]$ es el peso de cada elemento de tipo I ,
tal que $0 \leq W[I] \leq C$
 P , vector, donde $P[I]$ es la ganancia al tener el elemento I
en la mochila.
 $F[I]$, vector que Número de elementos de tipo I que se
guardan en la mochila.

Q, W, P, F tienen la misma longitud ya que hacen referencia
a los mismos tipos de artículos.

hayEspacioDisponible(C):
if ($C > 0$)
return 1
else
return 0.

llenarMochila(C, Q, W, P, F):
if (hayEspacioDisponible):
Si actividad mayor da mas ganancia que todo lo
de abajo con el mismo peso.
entonces añadimos un artículo de ese tipo
decrementamos en una unidad al artículo tipo
llenarMochila(C, Q, W, P, F)
y hay artículos de ese tipo.

* Ordenamos todos los
arreglos en base
a la ganancia de cada
artículo, el mayor tendrá
siempre la
posición [0]

Abajo

# de artículos	peso	ganancia
4	7	20
1	10	22

capacidad 14

Implementación

Se implementa en el lenguaje de programación Python.

Definimos la clase articulo con sus respectivos métodos:

```
class articulo:
    def __init__(self,existencias,peso,valor,tipo):
        self.id_tipo = tipo
        self.stock = existencias
        self.peso = peso
        self.ganancia = valor
        self.razon_peso_ganancia = valor/peso

    def quitarArticulo(self):
        if self.stock >= 1:
            self.stock -= 1

    def hayDisponibilidad(self):
        if self.stock >= 1:
            return True
        else:
            return False

    def mostrarInfo(self):
        print("%d\t%d\t%d\t%d\t%f" %
(self.id_tipo,self.stock,self.peso,self.ganancia,self.razon_peso_ganancia))
```

Definimos la clase mochila con sus respectivos métodos:

```
class mochila:
    def __init__(self,capacidad):
        self.capacidad = capacidad
        self.capacidad_disponible = self.capacidad
        self.articulos_totales = int(0)
        self.articulos_dentro = {}
        self.ganancia = 0
        # {"tipo":("cantidad","peso","ganancia")}

    def hayEspacioDisponible(self):
        if self.capacidad_disponible >= 0:
            return True
        else:
            return False

    def agregarArticulo(self,objeto):
        if self.hayEspacioDisponible() and self.capacidad_disponible >= objeto.peso:
            self.articulos_totales += 1
            self.capacidad_disponible -= objeto.peso
            self.ganancia += objeto.ganancia
            if not objeto.id_tipo in self.articulos_dentro:
                self.articulos_dentro[objeto.id_tipo] = list([1,objeto.peso,objeto.ganancia])
```

```

        else:
            self.articulos_dentro[objeto.id_tipo][0] += 1

    def mostrarContenido(self):
        print("\nCapacidad: ", self.capacidad)
        print("Espacio disponible: ", self.capacidad_disponible)
        print("Ganancia total: ", self.ganancia)
        print("Objetos dentro: ", self.articulos_totales, "\n")
        print("Tipo\tStock\tValor U\tPeso U\tValor T\tPeso T")
        for tipo in self.articulos_dentro:

print(tipo, "\t", self.articulos_dentro[tipo][0], "\t", self.articulos_dentro[tipo][2], "\t", self.articulos_dentro[tipo][1],

"\t", self.articulos_dentro[tipo][2]*self.articulos_dentro[tipo][0], "\t", self.articulos_dentro[tipo][1]*self.articulos_dentro[tipo][0])

capacidad, numero_articulos, peso_articulos, ganancia_articulos = leerDatos(name_file_INPUT)
backpack = mochila(capacidad)
articulos = []
for i in range(0, len(ganancia_articulos)):
    # lista de articulos, ordenada descendientemente con base a la razon ganancia peso
    articulos.append(articulo(numero_articulos[i], peso_articulos[i], ganancia_articulos[i], i+1))

articulos.sort(key=lambda tup: tup.razon_peso_ganancia, reverse=True)

print("Tipo\tStock\tPeso\tValor\tRazón")
for articulo in articulos:
    articulo.mostrarInfo()

for articulo in articulos:
    if articulo.peso <= backpack.capacidad_disponible:
        for cantidad in range(0, articulo.stock):
            if articulo.hayDisponibilidad():
                articulo.quitarArticulo()
                backpack.agregarArticulo(articulo)

backpack.mostrarContenido()

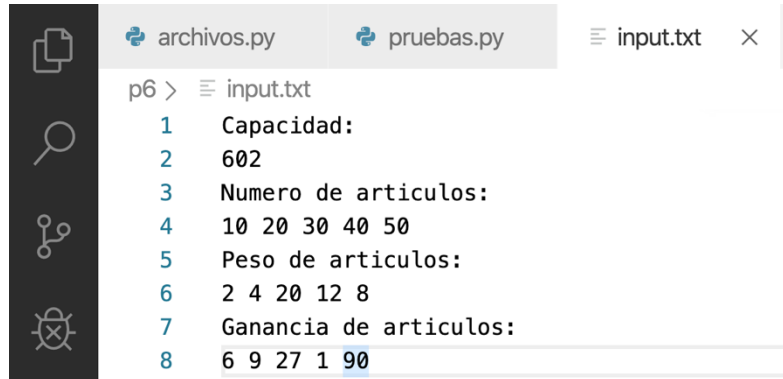
```

Funcionamiento

Dado que cada articulo tiene 3 propiedades; cantidad disponible, peso y ganancia. Con base en la ganancia y el peso, decidí generar un dato calculado a partir de la razón ente ganancia y peso, guardamos de manera descendente los artículos, de acuerdo con su razón, y luego vamos preguntando articulo a articulo por su disponibilidad y si aun hay espacio para uno mas en la mochila, disminuyendo la cantidad de artículos disponibles, aumentando el numero de objetos en la mochila, y disminuyendo el espacio disponible en la mochila.

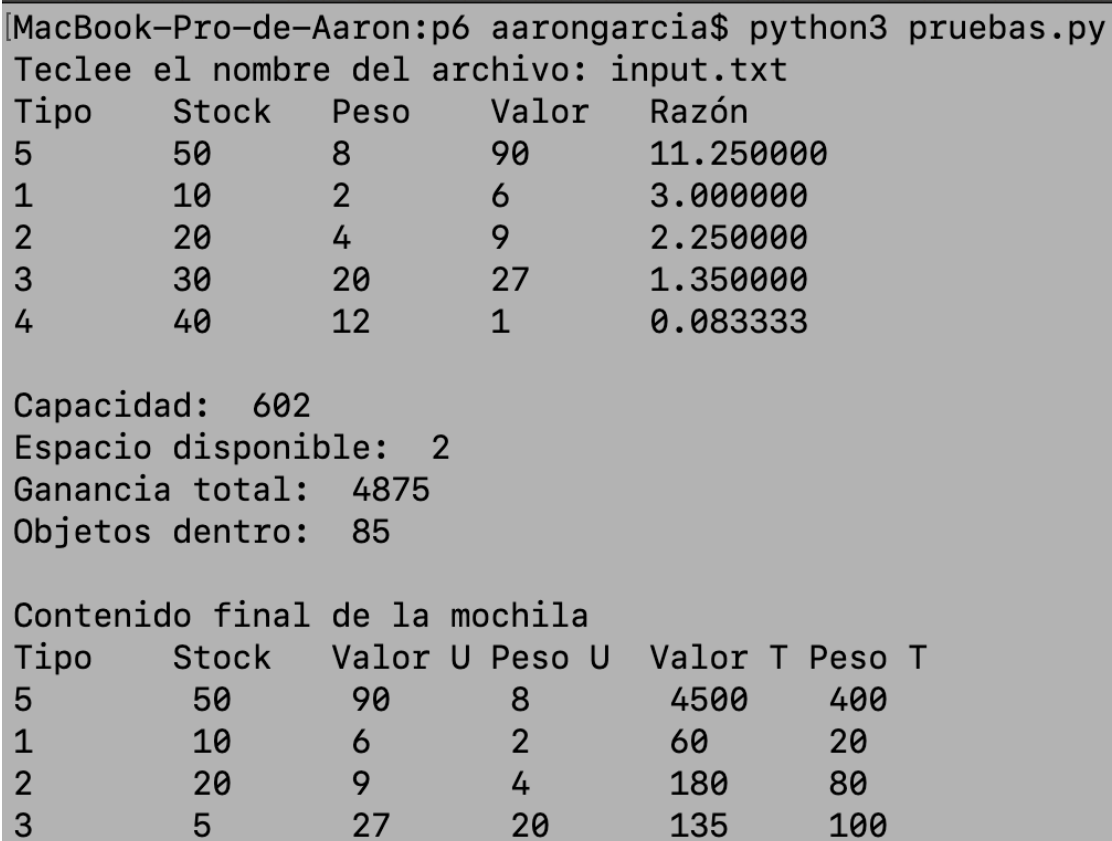
Ejemplo

Desde un archivo de texto se tienen los siguientes artículos y capacidad de la mochila:



```
archivos.py pruebas.py input.txt x
p6 > input.txt
1 Capacidad:
2 602
3 Numero de articulos:
4 10 20 30 40 50
5 Peso de articulos:
6 2 4 20 12 8
7 Ganancia de articulos:
8 6 9 27 1 90
```

Ejecución



```
MacBook-Pro-de-Aaron:p6 aarongarcia$ python3 pruebas.py
Teclee el nombre del archivo: input.txt
Tipo      Stock  Peso  Valor  Razón
5         50     8     90    11.250000
1         10     2      6     3.000000
2         20     4      9     2.250000
3         30    20     27     1.350000
4         40    12      1     0.083333

Capacidad: 602
Espacio disponible: 2
Ganancia total: 4875
Objetos dentro: 85

Contenido final de la mochila
Tipo      Stock  Valor U  Peso U  Valor T  Peso T
5         50     90      8    4500    400
1         10      6      2     60     20
2         20      9      4    180     80
3          5     27     20    135    100
```

En el ejemplo, sobran dos unidades de peso de la mochila, que bien pueden ser ocupados por artículos tipo 1, el problema es que se usó la totalidad de artículos de tipo 1, y ningún otro tipo de artículos tiene peso menor o igual que 2.

Justificación

La complejidad en el peor de los casos es igual al número de artículos por el número máximo de artículos disponibles de alguno de los artículos disponibles, es decir $O(n)$.

No estoy 100% seguro que el algoritmo sea el óptimo, ya que por lo que leí este programa obtiene solución óptima cuando se resuelve por memoria dinámica, pero también probé algunos ejemplos por método Simplex, y los resultados son generalmente los mismos, dado que cumple con el objetivo del programa, y entrega resultados lógicos de acuerdo con las entradas que se le proporcionan al programa.