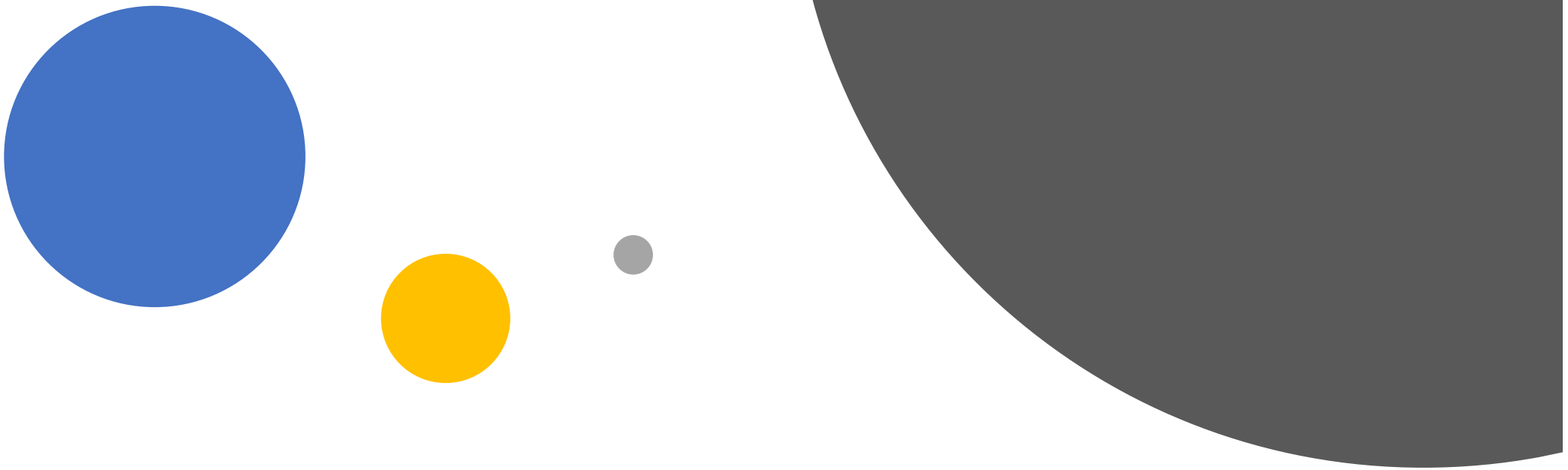
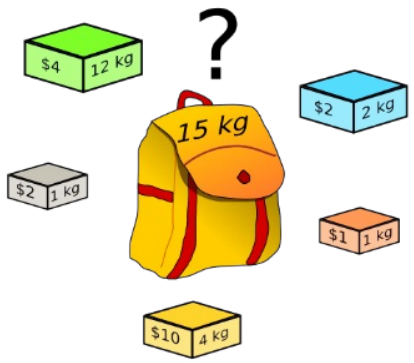


Algoritmos y Programación

Semana 10b: Knapsack
(Programación Dinámica)

Ejercicio

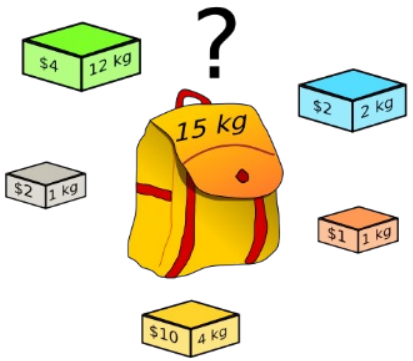


Utilizando programación dinámica programar mediante memoization y tabulation la recurrencia explicada en clase de teoría para resolver el problema de la mochila 0/1

$$t(n,w) \begin{cases} t(n-1, w) & : W_n > w \\ \max (t(n-1,w), t(n-1,w-W_n) + B_n) & \\ 0 & : n <= 0 \end{cases}$$

Formato del fichero de entrada

- La primera línea es un descriptor: número de items, peso máximo
- El resto de las líneas tiene el valor y el peso de cada item



3 items
Peso máximo = 10

El primer item
vale 45 y pesa 5

Ejemplo

3 10
45 5
48 8
35 3

El segundo item
vale 48 y pesa 8

VPL

main.py

```python
from collections       import namedtuple
from solve_memoization import *
from solve_tabulation  import *

Item = namedtuple("Item", ['index', 'value', 'weight'])

first_line = input().split()    # N, Capacity
N          = int(first_line[0])
capacity   = int(first_line[1])

items = []
for i in range(1, N+1):
    parts = input().split()
    items.append(Item(i, int(parts[0]), int(parts[1])))

# Comenzamos programando la recurrencia mediante tabulation
value1, taken1 = solve_tabulation(items, capacity)
print(value1, taken1)

# Cuando termines tabulation, comenta el código anterior
# para desactivarlo (la llamada a solve_tabulation y los
# dos print) y descomenta las siguientes lineas para que
# programes la recurrencia mediante memoization.

# value2, taken2 = solve_memoization(items, capacity)
# print(value2, taken2)

# Cuando termines los dos ejercicios puedes activar estas
# lineas para comprobar que los dos dan exactamente los
# mismos resultados.

# assert value1 == value2
# assert taken1 == taken2
```

VPL

solve_tabulation.py

```python
import numpy as np

def solve_tabulation(items, capacity):
    taken = []
    table = np.zeros((len(items)+1,capacity+1), dtype=int)

    def fill_table():
        return

    def fill_taken():
        return

    fill_table()
    fill_taken()
    return 0, taken
```

VPL

solve_memoization.py

```python
def solve_memoization(items, capacity):
    taken = []
    mem={}

    def t(n,w):
        # Primera fase: Calculamos la recurrencia guardando en
        # el diccionario la solución optima de cada subproblema.
        #   Aviso: Para resolver este ejercicio no es valido
        #          utilizar el soporte de @functools
        # ...
        return 0

    def fill_taken():
        # Segunda fase: Rellenamos la lista 'taken' con el
        # indice de los items elegidos.
        # ...

        return

    n = len(items)-1

    max_benefit = t(n,capacity)
    fill_taken()

    return 0, taken
```