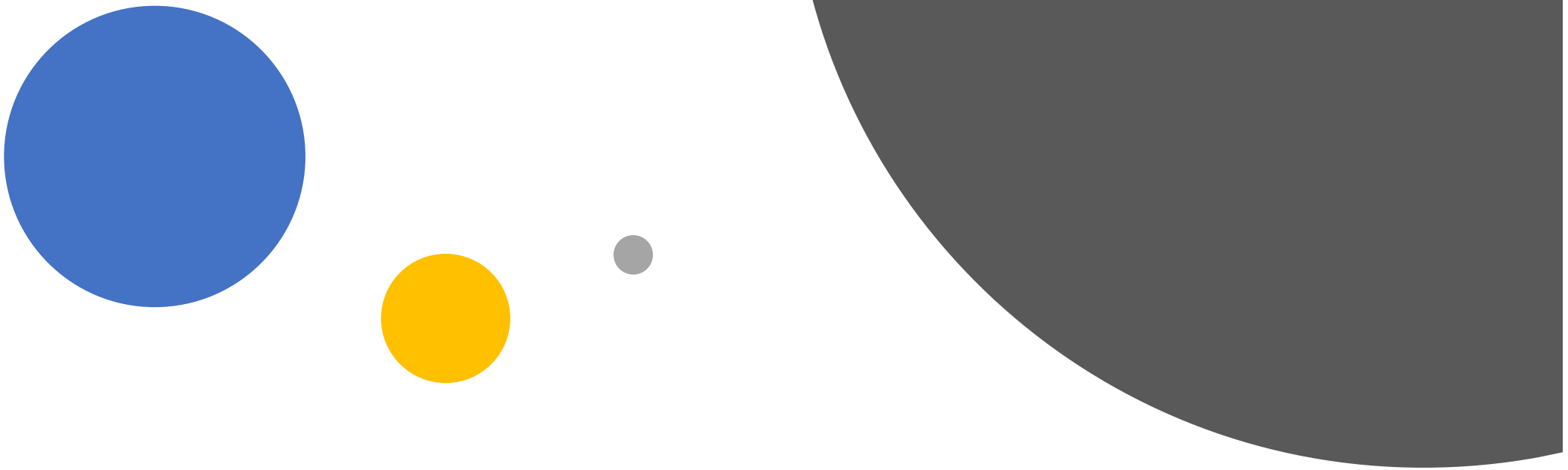
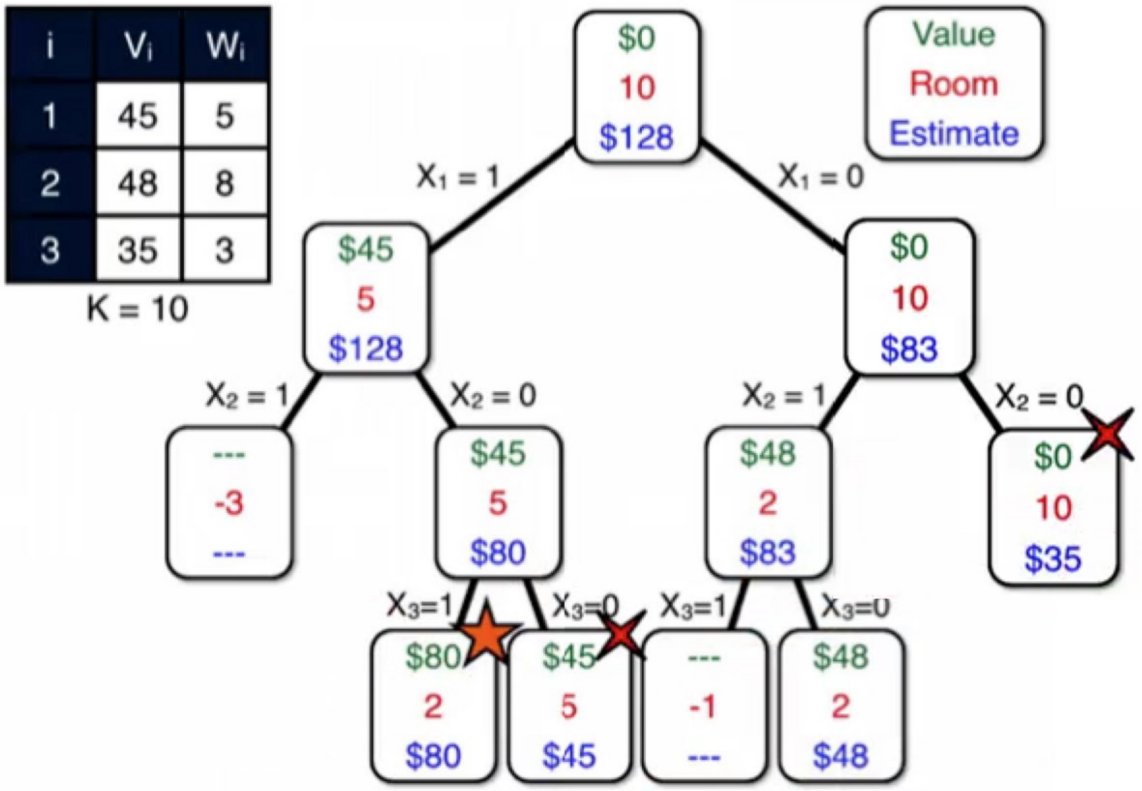


Algoritmos y Programación

Práctica 8: Branch & Bound (Parte 2)

Ejercicio

Utilizando la solución del recorrido en profundidad de la práctica 4, programa el algoritmo de ramificación y acotación explicado en clase.



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

i	V_i	W_i
1	45	5
2	48	8
3	35	3

K = 10

<u>Ejemplo</u>

3 items
Peso máximo = 10

3 10
45 5
48 8
35 3

El primer item
vale 45 y pesa 5

El segundo item
vale 48 y pesa 8

Branch and Bound, DFS

- Empezamos con el nodo raíz y lo metemos en una lista (*append*) de nodos vivos
- Mientras haya nodos en la lista de nodos vivos, hacemos un pop de la lista
  - Comprobamos si en el nodo actual hay espacio libre en la mochila
  - Comprobamos si la mejor estimación (*bound*) podría mejorar la solución incumbente (el mejor valor obtenido hasta el momento)
  - Si el valor obtenido en el nodo actual mejora la mejor solución hasta el momento, **actualizamos** el **valor** de la mejor solución, **y los ítems elegidos** para obtener esa solución (*taken*)

  - **Si no hemos llegado al final** del árbol:

    - Ramificamos (*branch*) por la derecha (*append*)
    - Ramificamos (*branch*) por la izquierda (*append*)

node

index

taken

value

room

estimate

De esta forma, cuando hagamos el *pop* empezaremos a ramificar por la izquierda

VPL

node.py

```python
# Copia aquí la definición del nodo que resuelve
# el VPL anterior!


```

main.py

```python
from collections import namedtuple

from node import *
from solve import *

first_line = input().split()
item_count = int(first_line[0])
capacity = int(first_line[1])

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

value, taken, visiting_order = solve_branch_and_bound_DFS(capacity, items, True)

print(visiting_order)
print(value)
print(taken)
```

VPL

solve.py

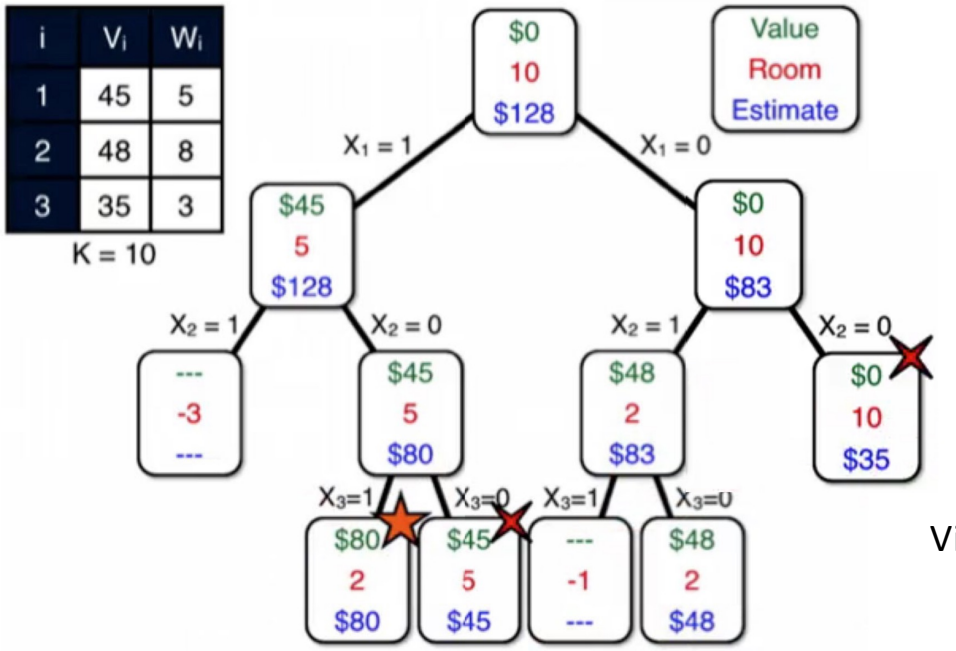
```python
from node import *

def solve_branch_and_bound_DFS(capacity, items, record_visiting_order = False):
    """
    :param capacity: capacidad de la mochila
    :param items: items de la mochila
    :param record_visiting_order: activa/desactiva el registro de nodos visitados
    :return: Por ahora sólo devuelve la lista de nodos visitados
    """

    # Completa este código para realizar el recorrido DFS; tienes
    # indicados los sitios que debes completar con tres puntos
    # suspensivos ("...")

    # Utilizamos la lista 'alive' como nuestra pila de nodos vivos
    # (pendientes de visitar) para programar nuestro recorrido DFS.

    alive = []

    # Utilizamos la lista Visiting_Order como el registro de nodos
    # visitados (el contenido final de esta lista lo utiliza el VPL
    # para comprobar que nuestro recorrido DFS es correcto).

    visiting_order = []
```

VPL

```
25
26      # 1) Creamos el nodo raiz (en este VPL todavía no utilizamos los
27      #    parámetros taken, value, room, con lo que se inicializan con
28      #    lista vacía y 0). El único valor necesario en el nodo es el
29      #    indice al primer elemento de la lista (index = 0).
30      # ...
31
32      # Lo añadimos a la lista de nodos vivos (alive)
33      # ...
34
35      # Mientras haya nodos en la lista de nodos vivos
36      # ...
37      while True:
38          # Avanzamos al siguiente nodo de nuestro recorrido DFS (hacemos un pop
39          # de la lista) y lo registramos en nuestro recorrido DFS.
40
41          current = alive.pop()
42          if record_visiting_order:
43              visiting_order.append(current.index)
44
45          # Si no hemos llegado al final del árbol
46          #    1) Ramificamos (branch) por la derecha (append)
47          #    2) Ramificamos (branch) por la izquierda (append)
48          # ...
49
50      return 0, [], visiting_order
```

Formato de la salida del programa

- Muestra el índice de los items visitados en el recorrido en profundidad, el beneficio conseguido y los items elegidos (‘taken’)
- Por ejemplo, éste es el resultado que genera con nuestro ejemplo



Visitados = [0, 1, 2, 2, 3, 3, 1, 2, 3, 3, 2]
Value = 80
Taken = [1, 3]