Backtracking – II

Clase #26

IIC1103 – Introducción a la Programación

Marcos Sepúlveda (marcos@ing.puc.cl)

Veremos hoy …

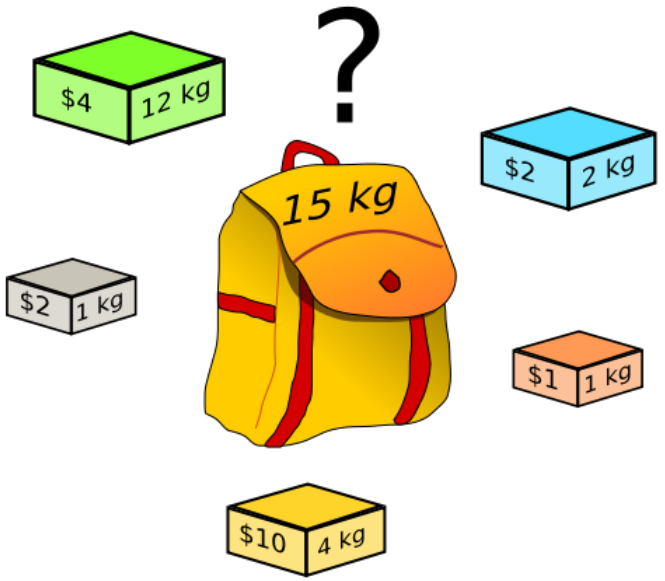
- Ejemplo problema de la Mochila

Fuente: http://controversia-oax.com.mx/?p=13287

Problema de la mochila

- “El **problema de la mochila**, comúnmente abreviado por **KP** (del inglés *Knapsack problem*) es un problema de optimización combinatoria, es decir, que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema. Modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de objetos, cada uno con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo.”

Fuente: https://es.wikipedia.org/wiki/Problema_de_la_mochila

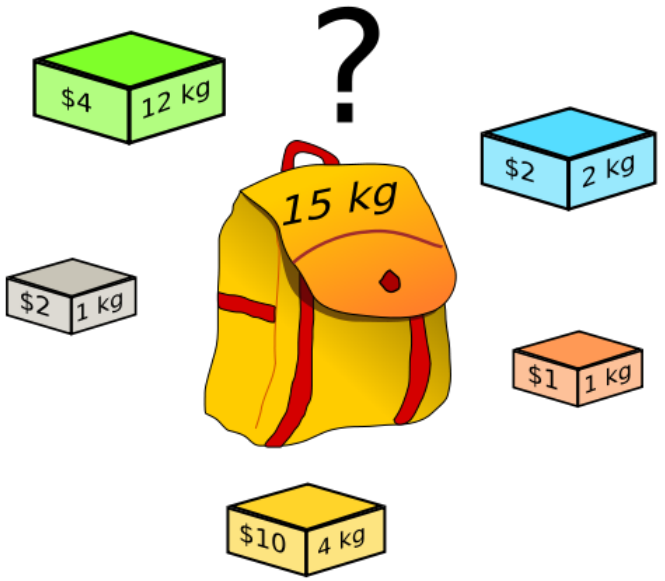


Fuente: https://commons.wikimedia.org/w/index.php?curid=985491

Problema de la mochila

- Ejemplo:
  Dada una mochila con una capacidad de 15 kg que puedo llenar con cajas de distinto peso y valor, ¿qué cajas elijo de modo de maximizar mis ganancias y no exceder los 15 kg de peso permitidos?

Fuente: https://es.wikipedia.org/wiki/Problema_de_la_mochila



Fuente: https://commons.wikimedia.org/w/index.php?curid=985491

Backtracking – estructura genérica

```
# Retorna True si vale la pena explorar la solución candidata
def SirveComoSolucionParcial(soln_c):
    pass

# Retorna True si la solución candidata resuelve el problema
def SirveComoSolucionFinal(soln_c):
    pass

# Usa la solución candidata como una solución valida al problema
def MostrarSolucionFinal(soln_c):
    pass

# Genera una lista de soluciones extendidas, derivadas de la solución candidata
def ObtenerSolucionesCandidatas(soln_c):
    pass

# Retorna una solución candidata inicial
def SolucionInicial():
    pass

# Backtracking genérico
def Backtracking(soln_c):
    if not SirveComoSolucionParcial(soln_c):
        return False
    if SirveComoSolucionFinal(soln_c):
        MostrarSolucionFinal(soln_c)
        return True
    lista_candidatas = ObtenerSolucionesCandidatas(soln_c)
    for solucion in lista_candidatas:
        if Backtracking(solucion):
            return True
    return False

Backtracking(SolucionInicial())
```

Backtracking – estructura genérica, visitando todas las posibles soluciones

```
# Retorna True si vale la pena explorar la solución candidata
def SirveComoSolucionParcial(soln_c):
    pass

# Retorna True si la solución candidata resuelve el problema
def SirveComoSolucionFinal(soln_c):
    pass

# Usa la solución candidata como una solución valida al problema
def MostrarSolucionFinal(soln_c):
    pass

# Genera una lista de soluciones extendidas, derivadas de la solución candidata
def ObtenerSolucionesCandidatas(soln_c):
    pass

# Retorna una solución candidata inicial
def SolucionInicial():
    pass

# Backtracking genérico, visitando todas las posibles soluciones
def Backtracking(soln_c):
    if not SirveComoSolucionParcial(soln_c):
        return False
    if SirveComoSolucionFinal(soln_c):
        MostrarSolucionFinal(soln_c)
        return True
    lista_candidatas = ObtenerSolucionesCandidatas(soln_c)
    for solucion in lista_candidatas:
        if Backtracking(solucion):
            # calculamos todas las soluciones
            pass
            # return True
    return False

Backtracking(SolucionInicial())
```

Problema de la mochila – descripción general de solución

- **Solución candidata parcial:**
  - Una solución parcial es aquella en que se ha tomado una decisión de si agregar, o no, cada uno de los primeros **n** elementos a la mochila.
  - Se representa por una lista de tamaño **n** en que la **i**-ésima posición es **True** si el elemento **i**-ésimo se agregará a la mochila, o **False** en caso contrario.
  - Ejemplo: `[True, False, True]`; se agregarán el primer y tercer elemento.

- **Solución inicial:**
  - Una solución parcial inicial es un **lista vacía**, que indica que aún no tomamos una decisión sobre agregar ningún elemento a la mochila.

- **Soluciones candidatas:**
  - A partir de una solución parcial de largo **n**, se crea una lista de candidatas que contiene **dos** soluciones parciales de largo **n+1**, que son iguales a la solución parcial previa hasta la posición **n**, y en que en la posición **n+1** contienen el valor **True** si se opta por agregar el elemento **n+1** a la mochila, o **False** si se opta por no agregarlo.

- **Solución parcial sirve cuando:**
  - La suma de los pesos de los elementos que se ha decidido agregar a la mochila es menor o igual al peso máximo.

- **Solución parcial es final cuando:**
  - Ya se ha tomado una decisión para todos los elementos, independiente de cuál sea dicha decisión.

Problema de la mochila – obtención del problema

- Datos del problema se leerán de un archivo que contiene en la primera fila el peso máximo a cargar en la mochila, y en las siguientes, los datos de los elementos a considerar, incluyendo su nombre, su valor entregado, y peso

- Ejemplo:

problema15.txt

```
15
Libro;4;12
Computador;10;4
Almuerzo;2;2
Cuaderno;2;1
Agenda;1;1
```

EJERCICIO

Examen 2016.2 – pregunta 4

Pregunta 4 (1/4)

Para preparar el enunciado de un examen, los profesores cuentan con una lista de problemas. Cada problema es una lista `[nombre, tipo, dificultad]`. Donde `nombre` identifica a la pregunta. El tipo indica qué contenido se está evaluando; solo hay 3 tipos: Objetos (`O`), Listas (`L`) y Recursión (`R`). La dificultad es un número que va de 0 a 100, donde 0 es un problema dificultad baja y 100 alta.

En esta pregunta se te pide escribir una función **recursiva `generar`** que permita generar una propuesta de examen válida a partir de una lista de problemas y límites superior inferior (`inf`) y (`sup`) e de dificultad total. El examen tendrá 3 problemas, uno de cada tipo (`O`, `L`, y `R`). Para que una propuesta de examen sea válida, la suma de la dificultad de los 3 problemas de la propuesta deben ser más grande o igual que `inf` y menor o igual que `sup`. En caso de no encontrar una solución, la función debe retornar una lista vacía.

Nota: Tú decides qué parámetros recibe esta función recursiva.
A modo de ejemplo, considera la siguiente lista de problemas y límites inferior y superior:

```
problemas = [["Don Yandran", "O",90], ["Substrings", "L",75],
             ["Castellers", "O",70], ["ADN", "L",30],
             ["UCarpool", "O",50], ["Examenes", "R",10],
             ["Viajes", "R",60], ["Diofanticas", "R",70],
             ["Baile Chino", "L",30], ["Monumentos", "O",40],
             ["Robot", "L",30],

inf = 210
sup = 230
```

La función **`generar`** podría retornar:

```
[['Don Yadran', 'O', 90], ['Substrings','L',75],['Viajes','R',60]]
```