
## EI1022, Algoritmia

# Problemas 8 y 9: Programación dinámica

# Introducción

Dividiremos las sesiones en dos partes:

- Implementación de ecuaciones recursivas
- Obtención de ecuaciones recursivas

# Implementación de ecuaciones recursivas

Implementaremos las ecuaciones recursivas de los siguientes problemas:

- Mochila discreta
- Asignación óptima de recursos
- Problema del cambio con limitación del número de monedas

# Mochila discreta

- Disponemos de $N$ objetos, cada uno con un valor $v_i$ y un peso $w_i$. Además disponemos de una mochila con capacidad de carga $C$
- Queremos cargar la mochila, sin sobrepasar su capacidad de carga, de forma que el valor de lo que contenga sea máximo
- El conjunto de soluciones será:

$$X = \left\{ (x_0, \ldots, x_{n-1}) \in \{0,1\}^N \,\middle|\, \sum_{0 \le i < N} x_i w_i \le C \right\}$$

- La ecuación recursiva es:

$$S(c,n) = \begin{cases} 0, & \text{si } n = 0; \\ S(c, n-1), & \text{si } n > 0 \text{ y } w_{n-1} > c; \\ \max\{S(c,n-1), & \\ \qquad S(c - w_{n-1}, n-1) + v_{n-1}\}, & \text{si } n > 0 \text{ y } w_{n-1} \le c. \end{cases}$$

# Implementación

- Edita el programa `knapsack_pd.py` de la carpeta auxiliar
- La función `read_data` ya está escrita y devuelve un elemento del tipo

  ```
  type Data = tuple[int, list[int], list[int]]
  ```

  donde la primera componente es la capacidad de la mochila y las otras dos son las listas de los valores y los pesos

# Implementación (2)

- Hay varias versiones de la función `process`, una por cada posible implementación de la ecuación recursiva:
  - `process_direct`: implementación directa
  - `process_memo`: implementación recursiva con memoización
  - `process_memo_solution`: implementación recursiva con memoización y recuperación de la solución
  - `process_iter`: implementación iterativa con recuperación de la solución
  - `process_iter_red`: implementación iterativa con reducción del coste espacial
- A medida vayas completando las implementaciones, podrás probarlas cambiando los comentarios en el cuerpo de `if __name__ == "__main__":`

# Implementación (3)

- Definimos los siguientes tipos para las puntuaciones, decisiones y soluciones:

```
type Score = int
type Decision = int
type Solution = tuple[Score, list[Decision]|None]
```

- Además, para la implementación de las versiones con memoización, se declaran diccionarios usando los tipos:

```
type SParams = tuple[int, int]
type Mem = dict[SParams, Score]
type MemSolution = dict[SParams, tuple[Score, SParams, Decision]]
```

# Implementación (4)

- Las funciones `process_...` devuelven dos valores, donde el segundo será `None` si es una implementación sin recuperación de la solución, así que `Result` es:

  ```
  type Result = Solution
  ```

- Por último, la función `show_result` muestra el beneficio y la lista de decisiones, si no es `None`

- Puedes probar tu implementación con los problemas del directorio `knapsacks`, teniendo en cuenta que para las versiones sin recuperación de la solución solo te interesa el beneficio

# Asignación óptima de recursos

- Sea $U$ el número de unidades de un recurso del que deseamos asignar cierta cantidad a cada una de $N$ actividades distintas. Además:
  - La posición $v_{i,u}$ de la matriz $v$ contiene el beneficio de asignar $u$ unidades del recurso a la actividad $i$
  - La posición $m_i$ del vector $m$ contiene el número máximo de unidades que podemos asignar la actividad $i$
- El conjunto de soluciones será:

$$X = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \forall i : 0 \leq i < N,\, 0 \leq x_i \leq m_i;\, \sum_{0 \leq i < n} x_i \leq U \right\}$$

- La ecuación recursiva para maximizar el beneficio es:

$$S(u, n) = \begin{cases} 0, & \text{si } n = 0; \\ \max\limits_{0 \leq d \leq \min(m_{n-1}, u)} \{S(u - d, n - 1) + v_{n-1,d}\}, & \text{si } n > 0. \end{cases}$$

# Implementación

- Edita el programa `resources_pd.py`
- Hay las mismas versiones de `process` que en `knapsack_pd.py`
- El formato de la entrada es el siguiente:
  - La primera línea contiene el valor de $U$
  - La segunda línea contiene el valor de $N$
  - La tercera línea tiene los $N$ números de $m$
  - Las siguientes $N$ líneas tienen $U$ números cada una y corresponden a las $N$ filas de $v$

# Implementación (2)

- Los tipos que se definen al principio son:

```
type Score = int
type Decision = int
type Solution = tuple[Score, list[Decision]|None]
```

- De este modo:

```
type Data = tuple[int, int, list[int], list[list[Score]]]
type Result = Solution
```

# Implementación (3)

- Para los diccionarios de memoización se definen los tipos:

```
type SParams = tuple[int, int]
type Mem = dict[SParams, Score]
type MemSolution = dict[SParams, tuple[Score, Decision]]
```

  porque podemos usar la decisión para averiguar los parámetros de la llamada anterior

- En el directorio `resources` de los auxiliares de la práctica tienes los ficheros con extensión `res` con instancias de los problemas y sus soluciones en ficheros con extensión `sol`

# Problema del cambio

- Supongamos que tenemos $N$ tipos de moneda distintos que queremos usar para pagar una cantidad $Q$
- Del tipo $i$ tenemos disponibles $m_i$ monedas cada con un valor $v_i$ y un peso $w_i$
- El conjunto de soluciones será:

$$X = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \forall i : 0 \leq i < N,\, 0 \leq x_i \leq m_i;\, \sum_{0 \leq i < n} x_i v_i = Q \right\}$$

- La ecuación recursiva para minimizar el peso de las monedas usadas es:

$$S(q,n) = \begin{cases} 0, & \text{si } n = 0 \text{ y } q = 0; \\ +\infty, & \text{si } n = 0 \text{ y } q > 0; \\ \min \left\{ S(q - d \cdot v_{n-1}, n-1) + d \cdot w_{n-1} \right. & \\ \qquad \left. \;\middle|\; 0 \leq d \leq \min\left(m_{n-1}, \left\lfloor \frac{q}{v_{n-1}} \right\rfloor\right) \right\}, & \text{si } n > 0. \end{cases}$$

# Implementación

- Edita el programa `coinchange_pd.py`
- Hay las mismas versiones de `process` que en `knapsack_pd.py`
- El formato de la entrada es el siguiente:
  - La primera línea contiene el valor de $N$
  - La segunda línea contiene el valor de $Q$
  - La tercera línea tiene los $N$ números de $v$
  - La cuarta línea tiene los $N$ números de $w$
  - La quinta línea tiene los $N$ números de $m$
- En el directorio `change` de los auxiliares de la práctica tienes los ficheros con extensión `chg` con instancias de los problemas y sus soluciones en ficheros con extensión `sol`

# Obtención de ecuaciones recursivas

# La plantilla de programación dinámica

$$S(\pi) = \operatorname*{opt}_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

# La plantilla de programación dinámica

$$S(\pi) = \operatorname*{opt}_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

- $\operatorname{opt}$: $\operatorname{mín}$ para minimización y $\operatorname{máx}$ para maximización

# La plantilla de programación dinámica

$$S(\pi) = \operatorname*{opt}_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

- $\operatorname{opt}$: $\operatorname{mín}$ para minimización y $\operatorname{máx}$ para maximización
- $\pi$: Parámetros del problema actual

# La plantilla de programación dinámica

$$S(\pi) = \operatorname*{opt}_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

- $\operatorname{opt}$: $\operatorname{mín}$ para minimización y $\operatorname{máx}$ para maximización
- $\pi$: Parámetros del problema actual
- $\pi \ominus d$: Parámetros del problema previo, antes de tomar la decisión $d$

# La plantilla de programación dinámica

$$S(\pi) = \operatorname*{opt}_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

- $\operatorname{opt}$: $\operatorname{mín}$ para minimización y $\operatorname{máx}$ para maximización
- $\pi$: Parámetros del problema actual
- $\pi \ominus d$: Parámetros del problema previo, antes de tomar la decisión $d$
- $\Delta$: Valores válidos para $d$

# La plantilla de programación dinámica

$$S(\pi) = \operatorname*{opt}_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

- $\operatorname{opt}$: $\operatorname{mín}$ para minimización y $\operatorname{máx}$ para maximización
- $\pi$: Parámetros del problema actual
- $\pi \ominus d$: Parámetros del problema previo, antes de tomar la decisión $d$
- $\Delta$: Valores válidos para $d$
- $p(d)$: Puntuación de $d$, término de $f$ correspondiente a $d$

# La plantilla de programación dinámica

$$S(\pi) = \operatorname*{opt}_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

- $\operatorname{opt}$: $\operatorname{mín}$ para minimización y $\operatorname{máx}$ para maximización
- $\pi$: Parámetros del problema actual
- $\pi \ominus d$: Parámetros del problema previo, antes de tomar la decisión $d$
- $\Delta$: Valores válidos para $d$
- $p(d)$: Puntuación de $d$, término de $f$ correspondiente a $d$
- $\oplus$: Combina la puntuación de $d$ con la puntuación de $\pi \ominus d$; "operador principal" de $f$

# Problema del cambio

- Soluciones parametrizadas:

$$X_{q,n} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \sum_{0 \le i < n} x_i v_i = q \right\}$$

- Función objetivo: $f\Big((x_0, \ldots, x_{n-1})\Big) = \sum_{i=0}^{n-1} x_i w_i$

- Ecuación:

$$S(\pi) = \operatorname*{opt}_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

# Problema del cambio

- Soluciones parametrizadas:

$$X_{q,n} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \sum_{0 \leq i < n} x_i v_i = q \right\}$$

- Función objetivo: $f\Big((x_0, \ldots, x_{n-1})\Big) = \sum_{i=0}^{n-1} x_i w_i$

- Ecuación:

$$S(\pi) = \min_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

  - $\mathrm{opt} = \min$

# Problema del cambio

- Soluciones parametrizadas:

$$X_{q,n} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \sum_{0 \le i < n} x_i v_i = q \right\}$$

- Función objetivo: $f\Big((x_0, \ldots, x_{n-1})\Big) = \sum_{i=0}^{n-1} x_i w_i$

- Ecuación:

$$S(q, n) = \min_{d \in \Delta} \left( S(\pi \ominus d) \oplus p(d) \right)$$

  - $\mathrm{opt} = \mathrm{mín}$
  - $\pi = q, n$

# Problema del cambio

- Soluciones parametrizadas:

$$X_{q,n} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \sum_{0 \le i < n} x_i v_i = q \right\}$$

- Función objetivo: $f\Big((x_0, \ldots, x_{n-1})\Big) = \sum_{i=0}^{n-1} x_i w_i$

- Ecuación:

$$S(q, n) = \min_{d \in \Delta} \left( S(q - d \cdot v_{n-1}, n-1) \oplus p(d) \right)$$

  - $\mathrm{opt} = \mathrm{mín}$
  - $\pi = q, n$
  - $\pi \ominus d = q - d \cdot v_{n-1}, n-1$

# Problema del cambio

- Soluciones parametrizadas:

$$X_{q,n} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \sum_{0 \le i < n} x_i v_i = q \right\}$$

- Función objetivo: $f\Big((x_0, \ldots, x_{n-1})\Big) = \sum_{i=0}^{n-1} x_i w_i$

- Ecuación:

$$S(q, n) = \min_{0 \le d \le \lfloor q / v_{n-1} \rfloor} \left( S(q - d \cdot v_{n-1}, n-1) \oplus p(d) \right)$$

  - $\mathrm{opt} = \min$
  - $\pi = q, n$
  - $\pi \ominus d = q - d \cdot v_{n-1}, n-1$
  - $\Delta = \{0, 1, \ldots, \lfloor q / v_{n-1} \rfloor\}$

# Problema del cambio

- Soluciones parametrizadas:

$$X_{q,n} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \sum_{0 \le i < n} x_i v_i = q \right\}$$

- Función objetivo: $f\Big((x_0, \ldots, x_{n-1})\Big) = \sum_{i=0}^{n-1} x_i w_i$

- Ecuación:

$$S(q, n) = \min_{0 \le d \le \lfloor q / v_{n-1} \rfloor} \left( S(q - d \cdot v_{n-1}, n-1) \oplus d \cdot w_{n-1} \right)$$

  - $\mathrm{opt} = \min$
  - $\pi = q, n$
  - $\pi \ominus d = q - d \cdot v_{n-1}, n-1$
  - $\Delta = \{0, 1, \ldots, \lfloor q / v_{n-1} \rfloor\}$
  - $p(d) = d \cdot w_{n-1}$

# Problema del cambio

- Soluciones parametrizadas:

$$X_{q,n} = \left\{ (x_0, \ldots, x_{n-1}) \in \mathbb{N}^n \;\middle|\; \sum_{0 \le i < n} x_i v_i = q \right\}$$

- Función objetivo: $f\Big((x_0, \ldots, x_{n-1})\Big) = \sum_{i=0}^{n-1} x_i w_i$

- Ecuación:

$$S(q, n) = \min_{0 \le d \le \lfloor q / v_{n-1} \rfloor} \left( S(q - d \cdot v_{n-1}, n-1) + d \cdot w_{n-1} \right)$$

  - $\mathrm{opt} = \min$
  - $\pi = q, n$
  - $\pi \ominus d = q - d \cdot v_{n-1}, n-1$
  - $\Delta = \{0, 1, \ldots, \lfloor q / v_{n-1} \rfloor\}$
  - $p(d) = d \cdot w_{n-1}$
  - $\oplus = +$

# Problema del cambio (2)

Añadimos los casos base de la recursión:

$$S(q,n) = \begin{cases} 0, & \text{si } n = 0 \text{ y } q = 0; \\ +\infty, & \text{si } n = 0 \text{ y } q > 0; \\ \min\limits_{0 \le d \le \lfloor \frac{q}{v_{n-1}} \rfloor} \left\{ S(q - d \cdot v_{n-1}, n-1) + d \cdot w_{n-1} \right\}, & \text{si } n > 0. \end{cases}$$

Donde:

- Para el caso base *posible* usamos el neutro de $\oplus$, en este caso $0$
- Para el caso base *imposible* usamos el neutro de $\mathrm{opt}$, en este caso $+\infty$

# Ejercicio (tipo asignación de recursos)

- En una explotación con $N$ campos, se dispone de $A$ metros cúbicos de agua y $B$ sacos de abono
- Se sabe que emplear $a$ metros cúbicos de agua y $b$ sacos de abono en el campo $i$ proporcionará un beneficio en la cosecha igual a $v_{i,a,b}$
- Obtén la ecuación recursiva que maximice el beneficio obtenido e indica los costes temporal y espacial que tendría la implementación con memoización

# Ejercicio (tipo mochila)

- Tenemos dos furgonetas de reparto. La primera puede cargar $C_1$ kilos y la segunda, $C_2$. En el almacén hay $N$ paquetes, cada uno con un peso $w_i$, para $i$ entre $0$ y $N-1$. El envío del paquete $i$ nos reporta un beneficio $v_i$.
- Con cada paquete tenemos tres opciones: cargarlo en la primera furgoneta, en la segunda y dejarlo.
- Obtén la ecuación recursiva que maximice el beneficio obtenido e indica los costes temporal y espacial que tendría la implementación con memoización

# Ejercicio

- Dados dos naturales $K$ y $N$, una descomposición de $K$ en $N$ sumandos es una secuencia $(s_0, \ldots, s_{N-1}) \in \mathbb{N}^N$ tal que

$$K = \sum_{i=0}^{N} s_i$$

- Se quiere obtener la descomposición de $K$ en $N$ sumandos que maximice el producto de sus elementos, es decir que maximice

$$\prod_{i=0}^{N} s_i$$

- Por ejemplo, las descomposiciones de $6$ en $3$ sumandos son $1+1+4$, $1+2+3$ y $2+2+2$, cuyos productos son $4$, $6$ y $8$. Así, la descomposición óptima es $2+2+2$

- Obtén la ecuación recursiva que maximice el producto e indica los costes temporal y espacial que tendría la implementación con memoización

# Ejercicio

- Dados dos naturales $K$ y $N$, una factorización de $K$ en $N$ factores es una secuencia $(q_0, \ldots, q_{N-1}) \in \mathbb{N}^N$ tal que

$$K = \prod_{i=0}^{N} s_i$$

- Se quiere obtener la factorización de $K$ en $N$ factores que minimice la suma de sus elementos, es decir que minimice

$$\sum_{i=0}^{N} s_i$$

- Por ejemplo, las factorizaciones de $12$ en $3$ factores son $1 \cdot 1 \cdot 12$, $1 \cdot 2 \cdot 6$, $1 \cdot 3 \cdot 4$ y $2 \cdot 2 \cdot 3$, que suman $14$, $9$, $8$ y $7$. Así, la factorización óptima es $2 \cdot 2 \cdot 3$

- Obtén la ecuación recursiva que minimice la suma e indica los costes temporal y espacial que tendría la implementación con memoización

# Ejercicio

- Se desea cargar el máximo peso posible de un conjunto de $N$ contenedores, cada uno con peso de $w_i$ toneladas en un barco con una capacidad máxima de $C$ toneladas
- Obtén la ecuación recursiva que maximice el peso cargado e indica los costes temporal y espacial que tendría la implementación con memoización

# Ejercicio

- Para preparar la oferta académica de la universidad se dispone de profesorado con capacidad para impartir un total de $C$ créditos
- Esos créditos se deben repartir entre $N$ titulaciones asegurándose de que los créditos asignados a la titulación $i$ sean mayores o iguales que $t_i$ y menores o iguales que $T_i$
- Se estima que si a la titulación $i$ se le asignan $d$ créditos, el número de estudiantes que se matricularán en ella será $a_{i,d}$
- Obtén la ecuación recursiva que maximice el número de estudiantes matriculados e indica los costes temporal y espacial que tendría la implementación con memoización