

IIC-2133 — Estructuras de Datos y Algoritmos

Programación Dinámica

Jorge A. Baier

Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile

Santiago, Chile



Problema de la Mochila sin Restricciones

El problema está definido por $\langle I, v, u, V \rangle$, donde:

- Conjunto de items I .
- Función de volumen $v : I \rightarrow \mathbb{N}^+$
- Función de utilidad $u : I \rightarrow \mathbb{N}^+$
- $V \in \mathbb{N}^+$ es el volumen a completar.

Objetivo:

Encontrar un multiconjunto E de elementos en I tal de

$$\begin{array}{l} \text{maximizar } \sum_{e \in E} u(e) \\ \text{sujeto a } \sum_{e \in E} v(e) \leq V \end{array}$$

El problema de decidir si existe un multiconjunto E , que satisface las restricciones, y que es tal que $\sum_{e \in E} u(e) \geq k$ es NP-completo.



Definición Recursiva del Óptimo

Sea $Opt(V)$ la solución óptima para $\langle I, v, u, V \rangle$. Podemos definir $Opt(V)$ recursivamente:

$$Opt(V) =$$



Definición Recursiva del Óptimo

Sea $Opt(V)$ la solución óptima para $\langle I, v, u, V \rangle$. Podemos definir $Opt(V)$ recursivamente:

$$Opt(V) = \max_{e \in I: v(e) \leq V} \{u(e) + Opt(V - v(e))\}$$

donde \max se define como 0 si el conjunto está vacío. Para resolver el problema computamos $Opt(v)$.

Ejercicio: Demuestre que $Opt(V)$ define la solución óptima para $\langle I, v, u, V \rangle$.



Calculando el Óptimo Vía un Enfoque Recursivo

- Directo.
- **Muy** Caro.

En el peor caso el tiempo para calcular $Opt(V)$, $T(V)$, se puede escribir como:

$$T(v) = \begin{cases} |I|(T(v-1) + c) & \text{si } V > 0 \\ 0 & V = 0, \end{cases}$$

de donde concluimos que $T(V)$ es $O(|I|^V)$.



Bottom Up en vez de Top Down

Construimos un arreglo Opt y ahora definimos:

$$Opt[v] = \max_{e \in I: v(e) \leq V} \{u(e) + Opt[v - v(e)]\}$$

y la usamos para calcular $Opt[0], Opt[1], \dots, Opt[V]$.

¿Cuál es el tiempo de ejecución en el peor caso?

¿Por qué esto funciona y cuándo podemos usar este “truco”?

Relacionado: *memoization*.



¿Qué es Programación Dinámica?

- **Programación Dinámica (PD)** es una técnica para resolver problemas de optimización que presentan:
 - 1 **Subestructura Óptima** La solución óptima se calcula usando la solución óptima de sub-problemas.
 - 2 **Problemas traslapados** Al resolverlo en la forma recursiva directa, es necesario resolver el mismo sub-problema múltiples veces.
- Consiste en cambiar la solución recursiva típica (*top-down*) por un enfoque *bottom-up*.
- *Bottom-up* consiste en resolver problemas pequeños antes de los más grandes.



Programación Dinámica: ¿Arte o Técnica?

El problema de la **mochila 0-1** está descrito por $\langle I, v, u, V \rangle$, donde:

- Conjunto de items I .
- Función de volumen $v : I \rightarrow \mathbb{N}^+$
- Función de utilidad $u : I \rightarrow \mathbb{N}^+$
- $V \in \mathbb{N}^+$ es el volumen a completar.

Objetivo:

Encontrar un subconjunto E de I para

$$\begin{aligned} &\text{maximizar } \sum_{e \in E} u(e) \\ &\text{sujeto a } \sum_{e \in E} v(e) \leq V \end{aligned}$$



Planteamiento Recursivo, primer intento

Podemos definir el valor óptimo recursivamente, así:

$$Opt(v, I)$$



Podemos definir el valor óptimo recursivamente, así:

$$Opt(v, I) = \max_{e \in I: v(e) \leq v} \{u(e) + Opt(v - v(e), I \setminus \{e\})\}$$

donde \max se define como 0 si el conjunto está vacío.



Elementos del problema:

- Conjunto de items $I = \{e_1, \dots, e_n\}$.
- Función de volumen $v : I \rightarrow \mathbb{N}^+$
- Función de utilidad $u : I \rightarrow \mathbb{N}^+$
- $V \in \mathbb{N}^+$ es el volumen a completar



Elementos del problema:

- Conjunto de items $I = \{e_1, \dots, e_n\}$.
- Función de volumen $v : I \rightarrow \mathbb{N}^+$
- Función de utilidad $u : I \rightarrow \mathbb{N}^+$
- $V \in \mathbb{N}^+$ es el volumen a completar

Si definimos

$$Opt(v, i) = \begin{cases} \text{máx}\{u(e_i) + Opt(v - v(e_i), i - 1), Opt(v, i - 1)\} & \text{si } v, i > 0 \\ 0 & \text{si } i = 0, v \geq 0 \\ -\infty & \text{si } v < 0 \end{cases}$$

podemos calcular $Opt(V, n)$.



Pseudo-Código para el Cálculo de $Opt(V, v)$

```
1  $Opt[0, 0] \leftarrow 0$ 
2 for  $v \leftarrow 0$  to  $V$  do
3   for  $i \leftarrow 1$  to  $n$  do
4     if  $v - v(e_i) < 0$  then  $opt_i \leftarrow -\infty$ 
5     else  $opt_i \leftarrow Opt[v - v(e_i), i - 1]$ 
6      $Opt[v, i] \leftarrow \text{máx}\{u(e_i) + opt_i, Opt[v, i - 1]\}$ 
```

El algoritmo es $O(V \cdot n)$, que es *pseudo-polinomial* en el tamaño del problema.



Las Rutas Más Cortas (definiciones)

Un **grafo dirigido** es un par (V, E) , donde:

- V es un conjunto de *nodos*.
- $E \subseteq V \times V$ es un conjunto de *arcos*. $(u, v) \in E$ significa que podemos movernos desde u hasta v .

Un **grafo dirigido con pesos** considera además una función $w : E \rightarrow \mathbb{R}$ que asigna un peso/costo a cada arco.



Las Rutas Más Cortas (definiciones)

Un **grafo dirigido** es un par (V, E) , donde:

- V es un conjunto de *nod*os.
- $E \subseteq V \times V$ es un conjunto de *arcos*. $(u, v) \in E$ significa que podemos movernos desde u hasta v .

Un **grafo dirigido con pesos** considera además una función $w : E \rightarrow \mathbb{R}$ que asigna un peso/costo a cada arco.

Un **camino entre u y v** en un grafo $G = (V, E)$ es una secuencia $v_0 v_1 \cdots v_n$ de vértices en V , donde $u = v_0$ y $v = v_n$ tales que $(v_{i-1}, v_i) \in E$ para todo $i \in \{1, \dots, n\}$.



Las Rutas Más Cortas (definiciones)

Un **grafo dirigido** es un par (V, E) , donde:

- V es un conjunto de *nodos*.
- $E \subseteq V \times V$ es un conjunto de *arcos*. $(u, v) \in E$ significa que podemos movernos desde u hasta v .

Un **grafo dirigido con pesos** considera además una función $w : E \rightarrow \mathbb{R}$ que asigna un peso/costo a cada arco.

Un **camino entre u y v** en un grafo $G = (V, E)$ es una secuencia $v_0 v_1 \cdots v_n$ de vértices en V , donde $u = v_0$ y $v = v_n$ tales que $(v_{i-1}, v_i) \in E$ para todo $i \in \{1, \dots, n\}$.

El **costo de un camino** $v_0 v_1 \cdots v_n$ es $\sum_{i=1}^n w(v_{i-1}, v_i)$.



Las Rutas Más Cortas (planteamiento recursivo)

La **distancia** entre u y v , $\delta(u, v)$, es el costo de un camino de costo mínimo (*shortest path*) entre u y v .

Problema: Calcular $\delta(u, v)$ para cada $u, v \in V$.



Las Rutas Más Cortas (planteamiento recursivo)

La **distancia** entre u y v , $\delta(u, v)$, es el costo de un camino de costo mínimo (*shortest path*) entre u y v .

Problema: Calcular $\delta(u, v)$ para cada $u, v \in V$.

Teorema

$$\delta(s, t) = \begin{cases} \min_{(u,t) \in E} \{\delta(s, u) + w(u, t)\} & \text{si } s \neq t \\ 0 & \text{si } u = v \end{cases}$$

¿Podemos usar esta relación para aplicar programación dinámica?



Un Parámetro Adicional

Definimos:

$\delta(u, v, k)$ = distancia entre u y v por caminos de k o menos arcos

La definición recursiva es ahora directa:

$$\delta(s, t, k + 1) = \begin{cases} 0 & \text{si } s = t \\ w(s, t) & \text{si } s \neq t \text{ y } k = 0 \\ \min_{(v,t) \in E} \{ \delta(s, v, k) + w(v, t) \} & \text{si } s \neq t \text{ y } k > 0 \end{cases}$$

Observación: Un camino más corto entre u y v no puede tener más aristas que el total de nodos en el grafo.



Distancias Más Cortas entre Cada Par de Nodos

- Suponemos que el conjunto de nodos $V = \{1, \dots, n\}$.
- Matriz de pesos W donde $w_{ij} \geq 0$ es el costo de arco entre i y j .
- $w_{ij} = \infty$ si no hay un arco entre i y j ; $w_{ij} = 0$ si $i = j$.

```
1 function Slow-All-Pairs-Shortest-Paths( $W$ )
2    $D^{(1)} \dots D^{(n)}$  matrices de  $n \times n$ 
3    $D^{(1)} \leftarrow W$ 
4    $n$  = número de filas/columnas de  $W$ 
5   for  $k \leftarrow 2$  to  $n$  do
6     for  $s \leftarrow 1$  to  $n$  do
7       for  $t \leftarrow 1$  to  $n$  do
8          $min \leftarrow \infty$ 
9         for  $v \leftarrow 1$  to  $n$  do
10           $dist \leftarrow d_{sv}^{(k-1)} + w_{vt}$ 
11          if  $min > dist$  then  $min \leftarrow dist$ 
12           $d_{st}^{(k)} \leftarrow min$ 
13  return  $D^{(n)}$ 
```

Que es $O(n^4)$.



Definition

Una camino $v_0v_1 \dots v_n$ **atraviesa** u si $u = v_i$ para algún $i \in \{1, \dots, n-1\}$.

- Supongamos $V = \{1, \dots, n\}$.
- Definamos $d(u, v, k)$ como el costo del camino de mínimo costo entre u y v que *no atraviesa* por un nodo $v > k$.
- Ahora podemos definir:



Definition

Una camino $v_0v_1 \dots v_n$ **atraviesa** u si $u = v_i$ para algún $i \in \{1, \dots, n-1\}$.

- Supongamos $V = \{1, \dots, n\}$.
- Definamos $d(u, v, k)$ como el costo del camino de mínimo costo entre u y v que *no atraviesa* por un nodo $v > k$.
- Ahora podemos definir:

$$d(i, j, k) = \begin{cases} \min\{d(i, j, k-1), d(i, k, k-1) + d(k, j, k-1)\} & \text{si } k > 0 \\ w(i, j) & \text{si } k = 0 \end{cases}$$



El Algoritmo Floyd-Warshall

```
1 function Floyd-Warshall( $W$ )
2    $D^{(1)} \dots D^{(n)}$  matrices del tamaño de  $W$ 
3    $D^0 \leftarrow W$ 
4   for  $k \leftarrow 1$  to  $n$  do
5     for  $i \leftarrow 1$  to  $n$  do
6       for  $j \leftarrow 1$  to  $n$  do
7          $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$ 
8   return  $D^{(n)}$ 
```

Este algoritmo es $O(n^3)$.

Ejercicios:

- Dé un algoritmo para construir un camino de costo mínimo a partir $D^{(n)}$. ¿De qué orden es su algoritmo?
- Muestre cómo modificar el algoritmo para retornar información para construir un camino óptimo en $O(n)$.

