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

Objetivo:
Encontrar un multiconjunto E de elementos en I tal de

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

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 $Opt[0], Opt[1], \ldots, Opt[V]$.

¿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.

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

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

Objetivo:
Encontrar un subconjunto E de I para

$$\text{maximizar } \textstyle\sum_{e \in E} u(e)$$
$$\text{sujeto a } \textstyle\sum_{e \in E} v(e) \leq V$$

Planteamiento Recursivo, primer intento

Podemos definir el valor óptimo recursivamente, así:

$$Opt(v, I)$$

Planteamiento Recursivo, primer intento

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.

Otra Formulación

Elementos del problema:

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

Otra Formulación

Elementos del problema:

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

Si definimos

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

podemos calcular $Opt(V, n)$.

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

$Opt[0,0] \leftarrow 0$
for $v \leftarrow 1$ **to** n **do**
  for $i \leftarrow n$ **down to** 1 **do**
    if $v - v(e_i) < 0$ **then** $opt_i \leftarrow -\infty$
    else $opt_i \leftarrow Opt[v - v(e_i), i-1]$
    $Opt[v,i] = \max\{u(e_i) + opt_i, Opt[v, i-1])\}$

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

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

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

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

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)

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:

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

La definición recursiva es ahora directa:

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

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 $V = \{1, \ldots, n\}$.
- 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 $i = j$.

```
function Slow-All-Pairs-Shortest-Paths(W)
    D^(1) ... D^(n) matrices de n × n
    D^(1) ← W
    n = número de filas/columnas de W
    for k ← 2 to n do
        for s ← 1 to n do
            for t ← 1 to n do
                min ← ∞
                for v ← 1 to n do
                    dist ← d_sv^(k−1) + w_vt
                    if min > dist then min ← dist
                d_st^(k) ← min
    return D^(n)
```

Que es $O(n^4)$.

Otra Formulación

Definition

Una camino $v_0 v_1 \ldots v_n$ **atraviesa** u si $u = v_i$ para algún $i \in \{1, \ldots, n-1\}$.

- Supongamos $V = \{1, \ldots, n\}$.
- 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:

Otra Formulación

Definition

Una camino $v_0 v_1 \ldots v_n$ **atraviesa** u si $u = v_i$ para algún $i \in \{1, \ldots, n-1\}$.

- Supongamos $V = \{1, \ldots, n\}$.
- 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:

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

El Algoritmo Floyd-Warshall

function *Floyd-Warshall*(W)

  $D^{(1)} \ldots D^{(n)}$ matrices del tamaño de W

  $D^0 \leftarrow W$

  for $k \leftarrow 1$ *to* n **do**

    for $i \leftarrow 1$ *to* n **do**

      for $j \leftarrow 1$ *to* n **do**

        $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$

  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 óptima en $O(n)$.