COSTRUZIONE ALGORITMI PROGRAMMAZIONE DINAMICA E ZAINO 0-1

Problema (dello Zaino 0-1 o zaino intero o knapsack 0-1)

Un ladro entra in un magazzino e trova n oggetti. L'i-esimo oggetto ha un valore di $\boldsymbol{v_i}$ euro e pesa $\boldsymbol{p_i}$ chilogrammi (i pesi sono numeri **interi positivi**).

Gli oggetti NON sono frazionabili. Quindi il ladro può o prendere l'intero oggetto i, o non prenderlo.

Il ladro ha solo uno zaino, che può contenere oggetti per un massimo di $\boldsymbol{P}$ chilogrammi.

Scrivere un algoritmo di programmazione dinamica che restituisca il massimo valore che il ladro può prendere, sapendo che tale valore è dato dall'equazione ricorsiva

$$V(i,j) = \begin{cases} V(i-1,j) & se\ j < p_i \\ \max(V(i-1,j), V(i-1,j-p_i) + v_i) & altrimenti \end{cases}$$

Con $V(i,j)$ che è la soluzione ottima del sottoproblema limitato agli oggetti $1 \dots i$ e con zaino di capienza massima j.

Costruzione di algoritmi – Fasi

Per costruire un algoritmo di programmazione dinamica dato il problema e la funzione ricorsiva per risolverlo, si deve:

1. Descrivere la **struttura dati** necessaria per la **memoizzazione**

2. Definire i **casi base**, e le loro **soluzioni** (banali)

3. Scrivere l'algoritmo che **inizializzi** la struttura di memoizzazione seguendo i casi base, e successivamente la **popoli** in maniera **bottom up**. Alla fine, restituisce il valore che corrisponde alla soluzione.

NOTA: quando si definisce la struttura di memoizzazione, è buona norma individuare anche la posizione della soluzione.

Zaino 0-1 Struttura di memoizzazione

Struttura di memoizzazione.

$V(i, j)$ ha **due parametri**:

- i è l'ultimo oggetto che consideriamo
- j è la capienza

Visto che ci sono **2** parametri, possiamo usare un vettore **bidimensionale**: una matrice V[]. Di quali dimensioni?

Il problema richiede di trovare la soluzione con **n oggetti e P di capienza massima**. Quindi la soluzione sarà contenuta in V[n,P].

Ci servono però anche i **casi base**. In particolare, ci serviranno i V[i,j] tali che i = 0 (nessun oggetto considerato) e/o j = 0 (capienza 0).

Quindi la matrice sarà grande $(n + 1) \times (P + 1)$.

Zaino 0-1 Casi Base

Bisogna considerare i casi base per ciascuna «dimensione» data dai parametri della funzione ricorsiva.

Valori casi base:

i = 0 (**nessun oggetto considerato**) – dato che non abbiamo considerato nessun oggetto, V[0,j] = 0 per ogni $0 \leq j \leq P$.

j = 0 (**capienza 0**) – dato che non possiamo prendere nessun oggetto, il valore massimo raggiungibile sarà 0. Quindi V[i,0] = 0 per ogni $0 \leq i \leq n$.

Zaino 0-1 Algoritmo

$$V(i,j) = \begin{cases} V(i-1,j) & se\ j < p_i \\ \max(V(i-1,j), V(i-1,j-p_i) + v_i) & altrimenti \end{cases}$$

```
Zaino(n,P,v[],p[]) // v[] e p[] sono i vettori dei valori e dei pesi
V[] <- nuova matrice (n+1) x (P+1)
%inizializzazione
for i=0..n do
   V[i,0] = 0
for j=0..P do
   V[0,j] = 0
%riempimento matrice seguendo la funzione ricorsiva
for i=1..n % un ciclo per ogni dimensione della strutt. di memoizzaz.
   for j=1..P do
      if(j<p[i]) then
         V[i,j] = V[i-1,j]
      else
         V[i,j] = max(V[i-1,j], V[i-1,j-p[i]]+v[i])
%soluzione
return V[n,P]
```

Si consideri la seguente tabella che associa ad ogni oggetto i un peso p_i ed un valore v_i. Dato uno zaino di capienza $P = 10$, si trovi una soluzione ottima per il problema dello zaino 0-1.

i	1	2	3	4
p_i	2	7	6	4
v_i	12.7	6.4	1.7	0.3

Soluzione:

		Matrice V										
		j (la capienza)										
		0	1	2	3	4	5	6	7	8	9	10
i	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	12,7	12,7	12,7	12,7	12,7	12,7	12,7	12,7	12,7
	2	0	0	12,7	12,7	12,7	12,7	12,7	12,7	12,7	19,1	19,1
	3	0	0	12,7	12,7	12,7	12,7	12,7	12,7	14,4	19,1	19,1
	4	0	0	12,7	12,7	12,7	12,7	13	13	14,4	19,1	19,1

Extra – è possible anche sapere quali oggetti appartengono alla soluzione dello zaino 0-1?

Sì, si deve utilizzare una matrice ausiliaria K (delle stesse dimensioni di V), che conterrà 1 se l'oggetto i-esimo fa parte della soluzione ottima che ha valore complessivo V[i, j]

```
Zaino(n,P,v[],p[]) // v[] e p[] sono i vettori dei valori e dei pesi
V[] <- nuova matrice (n+1) x (P+1)
K[] <- nuova matrice (n+1) x (P+1)
%inizializzazione
for i=0..n do
   V[i,0] = 0
   K[i,0] = 0
for j=0..P do
   V[0,j] = 0
   K[0,j] = 0
%riempimento matrice
for i=1..n
   for j=1..P do
      V[i,j] = V[i-1,j]
      K[i,j] =  0
      if V[i,j] < V[i-1,j-p[i]]+v[i] then
         V[i,j] = V[i-1,j-p[i]]+v[i]
         K[i,j] = 1
%soluzione
return V[n,P]
```

Extra – è possible anche sapere quali oggetti appartengono alla soluzione dello zaino 0-1?

Per sapere quali oggetti appartengono alla soluzione, visito K partendo dall'ultima cella (in fondo a destra)

d = P
i = n
while(i>0) **do**
    if K[i,d] = 1 **then**
        stampa "Seleziono oggetto" i
        d = d-p[i]
    i = i-1

	Matrice K (in verde le celle visitate)										
	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	0	1	1
3	0	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	1	1	0	0	0