Algoritmi per Bioinformatica

Tecniche di Progettazione

Programmazione Dinamica
Pt. 2

Programmazione Dinamica

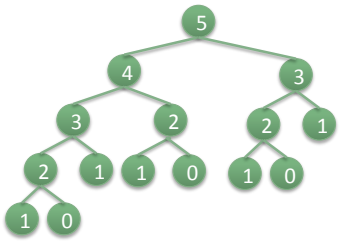
- Troviamo una soluzione ricorsiva al problema
- Identifichiamo l'esistenza di un numero "piccolo" di sottoproblemi
- Risolviamo "bottom-up"

Fibonacci: passi della Prog. Dinamica

- Soluzione ricorsiva

$$F(n) = \begin{cases} 1 & n = 0,1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

- L'algoritmo ricorsivo "naturale" risolve lo stesso sottoproblema più volte



- Usa una tabella per ricordare i sottoproblemi già calcolati

```
Set M[0] = M[1] = 1
Set M[2] = M[3] = … = M[n] =  -1
 // M[] memorizza i valori già calcolati

Procedure MFib(n){
 if M[n] = -1  // non ancora calcolato
      M[n] = MFib(n-1)+ MFib(n-2)
 return M[n]
}
```


Programmazione Dinamica: La ricetta

- Caratterizza la struttura del problema
- Definisci ricorsivamente il valore ottimo
- Calcola il valore ottimo *una volta* per ogni sottoproblema utile
  - Memoization
  - Iterativamente da piccoli a grandi
- Costruisci la soluzione ottima dalle informazioni memorizzate sul valore

Progr. Dinamica – Altri Esempi

- La sottosequenza crescente più lunga
- Calcolo del numero di sottinsiemi di taglia m di un insieme di taglia n
- Il problema dello zaino (knapsack)
  - Programmazione dinamica a due variabili

La sottosequenza crescente più lunga

- Input
  - A=(a(1),a(2), …, a(n)) una sequenza di numeri distinti
- Goal
  - Trovare la sottosequenza *crescente* più lunga
- Esempio: A = (2, 3, 14, 5, 9, 8, 4)
  - (2, 3, 8) e (3, 5, 9) sono sottosequenze *crescenti* di lunghezza 3
  - La sottosequenza *crescente* più lunga è (2, 3, 5, 9)

La sottosequenza crescente più lunga

- Input
  - A=(a(1),a(2), …, a(n)) una sequenza di numeri distinti
- Goal
  - Trovare la sottosequenza crescente più lunga
- Quante sono le sottosequenze possibili?
  -

La sottosequenza crescente più lunga

- Input
  - A=(a(1),a(2), …, a(n)) una sequenza di numeri distinti
- Goal
  - Trovare la sottosequenza crescente più lunga
- Quante sono le sottosequenze possibili?
  - $O(2^n)$ quanto i sottinsiemi di A

La sottosequenza crescente più lunga

- Input
  - A=(a(1),a(2), …, a(n)) una sequenza di numeri distinti
- Goal
  - Trovare la sottosequenza crescente più lunga
- Quante sono le sottosequenze possibili?
  - $O(2^n)$ quanto i sottinsiemi di A

Una ricerca esaustiva è da escludere!!!

La sottosequenza crescente più lunga

- Input
  - A=(a(1),a(2), …, a(n)) una sequenza di numeri distinti
- Goal
  - Trovare la sottosequenza crescente più lunga
- Programmazione Dinamica
  - Cerchiamo una soluzione ricorsiva
  - Identifichiamo un numero piccolo di sottoproblemi
  - Risolviamo bottom-up

La sottosequenza crescente più lunga

- Cerchiamo una soluzione ricorsiva
  - Input: A=(a(1),a(2), …, a(n)) una sequenza di numeri distinti
  - Es.: A = (2, 3, 14, 5, 9, 8, 4)

Definizione
L(i): lunghezza della sottosequenza crescente più lunga che finisce con a(i)

- Esempio: A= (2, 3, 14, 5, 9, 8, 4)
  - L(1) = 1, L(2) = 2, L(3) = 3, L(4) = 3, L(5) = 4, L(6) = 4, L(7) = 3
- La lunghezza della sottosequenza crescente più lunga
  max{ L(1), L(2), L(3), …, L(n) }

La sottosequenza crescente più lunga

- Cerchiamo una soluzione ricorsiva
  - Input: A=(a(1),a(2), …, a(n)) una sequenza di numeri distinti

Definizione
L(i): lunghezza della sottosequenza crescente più lunga che finisce con a(i)

- Esempio: A= (2, 3, 14, 5, 9, 8, 4)
  - L(1) = 1, L(2) = 2, L(3) = 3, L(4) = 3, L(5) = 4, L(6) = 4, L(7) = 3
- Come possiamo definire ricorsivamente ***L(i)***?

La sottosequenza crescente più lunga

- Cerchiamo una soluzione ricorsiva
  - Input: A=(a(1),a(2), …, a(n)) una sequenza di numeri distinti

Definizione

L(i): lunghezza della sottosequenza crescente più lunga che finisce con a(i)

- Esempio: A= (2, 3, 14, 5, 9, 8, 4)
  - L(1) = 1, L(2) = 2, L(3) = 3, L(4) = 3, L(5) = 4, L(6) = 4, L(7) = 3

- ***L(i)*** = max {L(j)+1 | j < i e a(j) < a(i) }
  - L(1) = 1

La sottosequenza crescente più lunga

L(i) *= max {L(j)+1 | j < i and a(j) < a(i) }*

- *L(1) = 1*

```
Input: n, a(1),…,a(n)

For j=1 to n
  L(j)←1, pre(j)←0
  For i=1 to j-1
       If a(i) < a(j) and 1+L(i)>L(j) then
           L(j) ← 1+L(i)
           pre(j)←i
       End If
  End For
End For

MSC ← 0
For i=1 to n
  MSC ← max{ MSC, L(i) }
End For
```

Complessità: $O(n^2)$

pre(j): contiene il predecessore di a(j) nella sottoseq. crescente più lunga che termina con a(j)

La sottosequenza crescente più lunga

Procedura per ricostruire la soluzione

```
Input: n, L(1),…,L(n),pre(1),…,pre(n),MSC

j←0
While L(j)<> MSC
   j++
End While

While j>0
  Aggiungi j all'inizio della soluzione;
  j← pre(j)
End While
```

Complessità: O(n)

MSC: contiene la lunghezza della sottosequenza più lunga

La soluzione viene costruita in ordine inverso: dall'ultimo elemento al primo

Esempio 2

Combinazioni

e

Sottoinsiemi

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n
- Calcolare il numero di modi di scegliere m elementi tra n
- choose(n,m) = $\binom{n}{m}$

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n
- Calcolare il numero di modi di scegliere m elementi tra n
- choose(n,m) = $\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$

Il primo elemento unito ad ogni sottinsieme di m-1 tra i rimanenti n-1

Non inserisco il primo elemento e scelgo ogni sottinsieme di m tra i rimanenti n-1

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$$

1. Soluzione ricorsiva

$$choose(n,m) = \begin{cases} 1 & m = 0 \text{ o } m = n \\ choose(n-1,m-1) + choose(n-1,m) & altrimenti \end{cases}$$

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n

1. Soluzione ricorsiva $choose(n,m) = \begin{cases} 1 & m = 0 \text{ o } m = n \\ choose(n-1,m-1) + choose(n-1,m) & altrimenti \end{cases}$

```
Choose(n, m)
    if m = 0 or m=n
        return 1
    else
        return ( Choose(n-1,m-1) + Choose(n-1, m) )
```

Complessità:

T(n,m) complessità di *Choose(n,m)*

T(n) = $\max_m$ T(n,m)

$$T(n) = \begin{cases} 1 & n = 0,1 \\ 2T(n-1) & altrimenti \end{cases} = \Theta(2^n)$$

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n

1. Soluzione ricorsiva $choose(n,m) = \begin{cases} 1 & m=0 \text{ o } m=n \\ choose(n-1,m-1)+choose(n-1,m) & altrimenti \end{cases}$

```
Choose(n, m)
    if m = 0 or m=n
        return 1
    else
        return ( Choose(n-1,m-1) + Choose(n-1, m) )
```

Complessità:

$$T(n) = \begin{cases} 1 & n=0,1 \\ 2T(n-1) & altrimenti \end{cases} = \Theta(2^n)$$

La ricorsione ricalcola gli stessi valori più volte
Es.: choose(7,4)
= choose(6,3) + choose(6,4)
= choose(5,2)+choose(5,3)+ choose(5,3) +choose(5,4)
=...

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n

1. Soluzione ricorsiva con memoization $choose(n,m) = \begin{cases} 1 & m=0 \text{ o } m=n \\ choose(n-1,m-1)+choose(n-1,m) & altrimenti \end{cases}$

```
// T[i,j]  contiene il valore di choose(i,j) se già calcolato

Choose(n, m)
    if m = 0 or m=n
        T[n,m] = 1
    else
        if T[n,m] non definito
            T[n,m] =  Choose(n-1,m-1) + Choose(n-1, m) )
    return T[n,m]
```

Complessità:
Tempo: O(nm)
Spazio: O(nm)

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n

1. Soluzione ricorsiva con memoization $choose(n,m) = \begin{cases} 1 & m=0 \text{ o } m=n \\ choose(n-1,m-1)+choose(n-1,m) & altrimenti \end{cases}$

```
// T[i,j]  contiene il valore di choose(i,j) se già calcolato

Choose(n, m)
    if m = 0 or m=n
        T[n,m] = 1
    else
        if T[n,m] non definito
            T[n,m] =  Choose(n-1,m-1) + Choose(n-1, m) )
    return T[n,m]
```

Analizziamo La tabella T[i,j]

i \ j	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1		1				
3	1			1			
4	1				1		
5	1					1	
6	1			+			1
...							

T[i,j] = T[i-1,j-1] + T[i-1,j]

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n

1. Soluzione iterativa per colonne $choose(n,m) = \begin{cases} 1 & m=0 \text{ o } m=n \\ choose(n-1,m-1)+choose(n-1,m) & altrimenti \end{cases}$

```
// T[i,j]  contiene il valore di choose(i,j) se già calcolato
Choose(n, m)
    for i = 0 to n
        T[i,0] = 1
    for i = 0 to m
        T[i, i] = 1

    for  j = 1 to m                 // O(m)
        for i = j+1 to n            // O(n-j)
            T[i,j] =  T[i-1, j-1] + T[i-1, j]
    return T[n,m]
```

Analizziamo La tabella T[i,j]

i \ j	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1		1				
3	1			1			
4	1				1		
5	1					1	
6	1			+			1
...							

Complessità: tempo O(nm), spazio O(nm)

Combinazioni e sottinsiemi

- Calcolare il numero di sottinsiemi di taglia m di un insieme di taglia n

1. Soluzione iterativa per righe

$$choose(n,m) = \begin{cases} 1 & m = 0 \text{ o } m = n \\ choose(n-1,m-1) + choose(n-1,m) & altrimenti \end{cases}$$

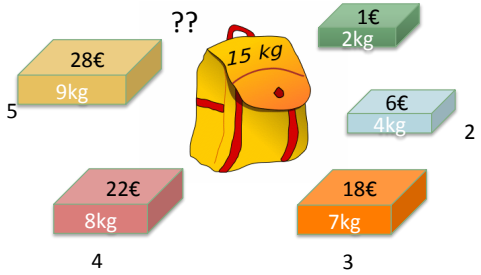
```
// T[i,j]  contiene il valore di choose(i,j) se già calcolato
Choose(n, m)
    for i = 0 to n
        T[i,0] = 1
    for i = 0 to m
        T[i, i] = 1

    for  i = 1 to n                    // O(n)
        for j = 1 to min{ i, m }       // O(m)
            T[i,j] =  T[i-1, j-1] + T[i-1, j]
    return T[n,m]
```

Analizziamo La tabella T[i,j]

i \ j	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1		1				
3	1			1			
4	1				1		
5	1					1	
6	1			+			1
...							

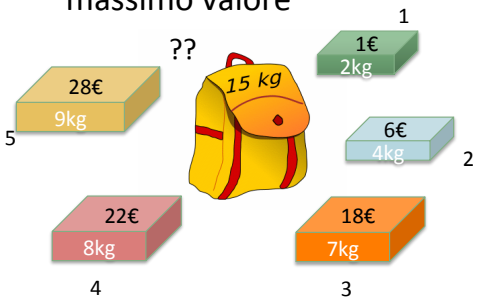
Complessità: tempo O(nm), spazio O(nm)

Il problema dello Zaino



Il Problema dello Zaino (Knapsack)

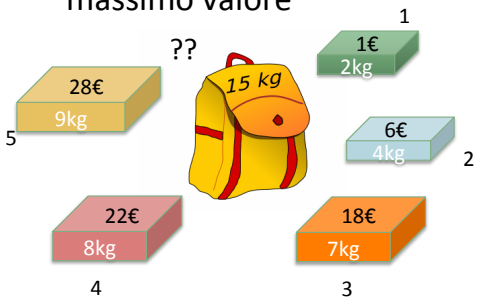
- Input: n oggetti e uno "zaino"
  - Oggetto i pesa $w_i > 0$ kg e ha valore $v_i > 0$
  - lo zaino ha una portata di W kg
- Output: un insieme di oggetti di *peso totale* ≤ *W* e massimo valore



Es.: {3, 4} ha valore 40

Il Problema dello Zaino (Knapsack)

- Input: n oggetti e uno "zaino"
  - Oggetto i pesa $w_i > 0$ kg e ha valore $v_i > 0$
  - lo zaino ha una portata di W kg
- Output: un insieme di oggetti di *peso totale* ≤ *W* e massimo valore



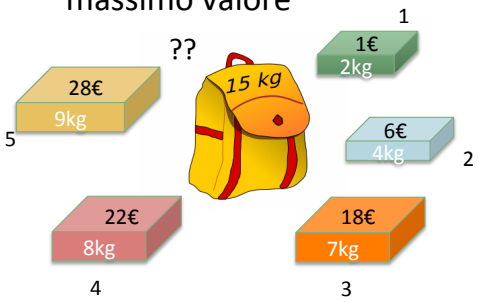
Es.: {3, 4} ha valore 40

Greedy1: aggiungi l'oggetto di *massimo valore possibile*

Es.: {5, 2, 1} ha valore 35 ⇒ Greedy1 non dà una soluzione ottima!

Il Problema dello Zaino (Knapsack)

- Input: n oggetti e uno "zaino"
  - Oggetto i pesa $w_i > 0$ kg e ha valore $v_i > 0$
  - lo zaino ha una portata di W kg
- Output: un insieme di oggetti di *peso totale* $\leq W$ e massimo valore



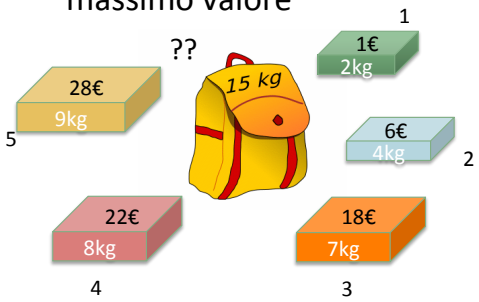
Es.: {3, 4} ha valore 40

Greedy2: aggiungi l'oggetto di *minimo peso*

Es.: {1, 2, 3} ha valore 25 ⇒ Greedy2 non dà una soluzione ottima!

Il Problema dello Zaino (Knapsack)

- Input: n oggetti e uno "zaino"
  - Oggetto i pesa $w_i > 0$ kg e ha valore $v_i > 0$
  - lo zaino ha una portata di W kg
- Output: un insieme di oggetti di *peso totale* $\leq W$ e massimo valore



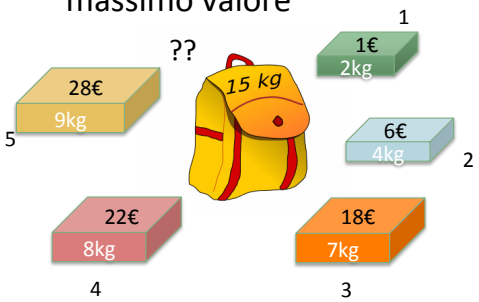
Es.: {3, 4} ha valore 40

Greedy3: aggiungi un oggetto con *massimo rapporto* v_i/w_i

Es.: {5, 2, 1} ha valore 35 ⇒ Greedy3 non dà una soluzione ottima!

Il Problema dello Zaino (Knapsack)

- Input: n oggetti e uno "zaino"
  - Oggetto i pesa $w_i > 0$ kg e ha valore $v_i > 0$
  - lo zaino ha una portata di W kg
- Output: un insieme di oggetti di *peso totale* $\leq W$ e massimo valore



Item	Valore	Peso
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

opt: {3, 4} ha valore 40

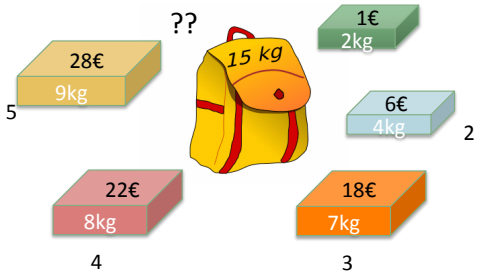
Proviamo con la programmazione dinamica

Probema dello Zaino – 1° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Proviamo come con il problema sugli intervalli

Definizione: OPT(i) := max valore con un sottinsieme di {1, …, i}



Probema dello Zaino – 1° tentativo di Prg Din

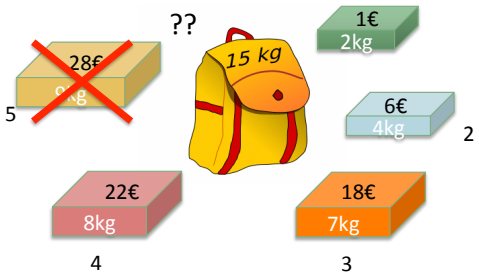
Step 1: L'insieme dei sottoproblemi / ricorsione

- Proviamo come con il problema sugli intervalli

Definizione: OPT(i) := max valore con un sottinsieme di {1, …, i}

Caso 1: La soluzione ottima non sceglie i

- OPT(i) = OPT(i-1)



Probema dello Zaino – 1° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Proviamo come con il problema sugli intervalli

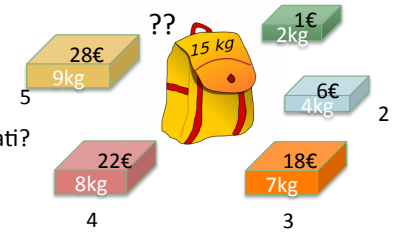
Definizione: OPT(i) := max valore con un sottinsieme di {1, …, i}

Caso 1: La soluzione ottima non sceglie i

- OPT(i) = OPT(i-1)

Caso 2: La soluzione ottima sceglie i

- Quali altri oggetti vanno scartati?
- Come facciamo a sapere che possiamo scegliere i senza gli oggetti già inseriti prima di ridurci a {1,…,i}



Probema dello Zaino – 1° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Proviamo come con il problema sugli intervalli

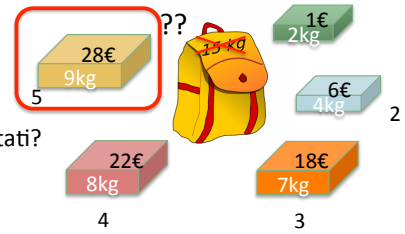
Definizione: OPT(i) := max valore con un sottinsieme di {1, …, i}

Case 1: La soluzione ottima non sceglie i

- OPT(i) = OPT(i-1)

Caso 2: La soluzione ottima sceglie i

- Quali altri oggetti vanno scartati?
- Come facciamo a sapere che possiamo scegliere i senza gli oggetti già inseriti prima di ridurci a {1,…,i}



Probema dello Zaino – 1° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Proviamo come con il problema sugli intervalli

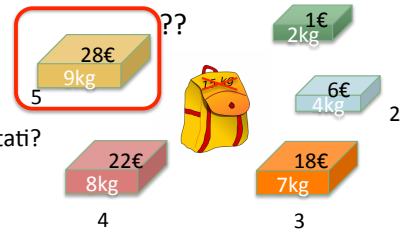
Definizione: OPT(i) := max valore con un sottinsieme di {1, …, i}

Case 1: La soluzione ottima non sceglie i

- OPT(i) = OPT(i-1)

Caso 2: La soluzione ottima sceglie i

- Quali altri oggetti vanno scartati?
- Come facciamo a sapere che possiamo scegliere i senza gli oggetti già inseriti prima di ridurci a {1,…,i}



Probema dello Zaino – 1° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Proviamo come con il problema sugli intervalli

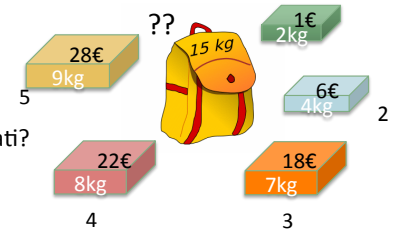
Definizione: OPT(i) := max valore con un sottinsieme di {1, …, i}

Case 1: La soluzione ottima non sceglie i

- OPT(i) = OPT(i-1)

Caso 2: La soluzione ottima sceglie i

- Quali altri oggetti vanno scartati?
- Come facciamo a sapere che possiamo scegliere i senza gli oggetti già inseriti prima di ridurci a {1,…,i}



Dobbiamo cambiare la definizione dei sottoproblemi

Probema dello Zaino – 2° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Aggiungiamo un'altra variabile!

Definizione: OPT(i, w) := max valore con un sottinsieme di {1, …, i} e con uno zaiono di capacità w

Probema dello Zaino – 2° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Aggiungiamo un'altra variabile!

Definizione: OPT(i, w) := max valore con un sottinsieme di {1, …, i} e con uno zaiono di capacità w

Caso 1: La soluzione ottima non sceglie i

- OPT(i,w) = OPT(i-1,w)

Probema dello Zaino – 2° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Aggiungiamo un'altra variabile!

Definizione: OPT(i, w) := max valore con un sottinsieme di {1, …, i} e con uno zaiono di capacità w

Caso 1: La soluzione ottima non sceglie i

- OPT(i,w) = OPT(i-1,w)

Caso 2: La soluzione ottima sceglie i

- La capacità rimanente è ora $w-w_i$
- La soluzione ottima sceglie gli oggetti di massimo valore totale tra {1,…,i-1}, per uno zaino con la nuova capacità

Probema dello Zaino – 2° tentativo di Prg Din

Step 1: L'insieme dei sottoproblemi / ricorsione

- Aggiungiamo un'altra variabile!

Definizione: OPT(i, w) := max valore con un sottinsieme di {1, …, i}
e con uno zaiono di capacità w

Caso 1: La soluzione ottima non sceglie i

- OPT(i,w) = OPT(i-1,w)

Caso 2: La soluzione ottima sceglie i

- La capacità rimanente è ora $w-w_i$
- La soluzione ottima sceglie gli oggetti di massimo valore totale tra {1,…,i-1}, per uno zaino con la nuova capacità

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0															
{1,2}	0															
{1,2,3}	0															
{1,2,3,4}	0															
{1,2,3,4,5}	0															

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

W = 15

Oggetto	Valore	Peso
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	0	1	1	6	6	7	7	7	7	7	7	7	7	7	7
{1,2,3}	0	0	1	1	6	6	7									
{1,2,3,4}	0															
{1,2,3,4,5}	0															

+18

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

W = 15

Oggetto	Valore	Peso
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	0	1	1	6	6	7	7	7	7	7	7	7	7	7	7
{1,2,3}	0	0	1	1	6	6	7	18								
{1,2,3,4}	0															
{1,2,3,4,5}	0															

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

W = 15

Oggetto	Valore	Peso
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	0	1	1	6	6	7	7	7	7	7	7	7	7	7	7
{1,2,3}	0	0	1	1	6	6	7	18	18	20	20	24	24	25	25	25
{1,2,3,4}	0	0	1	1	6	6	7	18	22	22	24	24	28	28	29	40
{1,2,3,4,5}	0	0	1	1	6	6	7	18	22	28	28	29	29	34	34	40

+28

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

Oggetto	Valore	Peso
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

W = 15

Il Problema dello Zaino: algoritmo DP

Step 2: Calcola Ottimo una volta per ogni sottoproblema

- Riempiamo la tavola iterativamente "per righe"

```
Input: n, w1, …, wn, v1, …, vn
for w = 0 to W
    M[0,w] = 0

for i = 1, …, n              // numero di oggetti a disposizione
    for w = 0, 1, …, W       // massimo peso
        if (wi > w)
            M[i,w] = M[i-1, w]
        else
            M[i,w] = max{M[i-1, w], vi + M[i-1, w-wi ]
return M[n, W]
```

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > W \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i & altrimenti \end{cases}$$

Il Problema dello Zaino: algoritmo DP

Step 2: Calcola Ottimo una volte per ogni sottoproblema

- Riempiamo la tavola iterativamente "per righe"

```
Input: n, w1, …, wn, v1, …, vn
for w = 0 to W
    M[0,w] = 0

for i = 1, …, n
    for w = 0, 1, …, W
        if (wi > w)
            M[i,w] = M[i-1, w]
        else
            M[i,w] = max{M[i-1, w], vi + M[i-1, w-wi ]
return M[n, W]
```

Il running time è O(nW)

L'algoritmo è solo pseudopolinomiale

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > W \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i & altrimenti \end{cases}$$

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	0	1	1	6	6	7	7	7	7	7	7	7	7	7	7
{1,2,3}	0	0	1	1	6	6	7	18	18	20	20	24	24	25	25	25
{1,2,3,4}	0	0	1	1	6	6	7	18	22	22	24	24	28	28	29	40
{1,2,3,4,5}	0	0	1	1	6	6	7	18	22	28	28	29	29	34	34	40

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

Item	Value	Weight
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

Abbiamo calcolato solo il valore di una soluzione ottima

Eserc.: *Scrivere una procedura che costruisce l'insieme di oggetti in una soluzione ottima, dalla tavola*

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	0	1	1	6	6	7	7	7	7	7	7	7	7	7	7
{1,2,3}	0	0	1	1	6	6	7	18	18	20	20	24	24	25	25	25
{1,2,3,4}	0	0	1	1	6	6	7	18	22	22	24	24	28	28	29	40
{1,2,3,4,5}	0	0	1	1	6	6	7	18	22	28	28	29	29	34	34	40

+28

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

Item	Value	Weight
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

Abbiamo calcolato solo il valore di una soluzione ottima

Eserc.: *Scrivere una procedura che costruisce l'insieme di oggetti in una soluzione ottima, dalla tavola*

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	0	1	1	6	6	7	7	7	7	7	7	7	7	7	7
{1,2,3}	0	0	1	1	6	6	7	18	18	20	20	24	24	25	25	25
{1,2,3,4}	0	0	1	1	6	6	7	18	22	22	24	24	28	28	29	40
{1,2,3,4,5}	0	0	1	1	6	6	7	18	22	28	28	29	29	34	34	40

+22

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

Item	Value	Weight
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

Abbiamo calcolato solo il valore di una soluzione ottima

Eserc.: *Scrivere una procedura che costruisce l'insieme di oggetti in una soluzione ottima, dalla tavola*

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	0	1	1	6	6	7	7	7	7	7	7	7	7	7	7
{1,2,3}	0	0	1	1	6	6	7	18	18	20	20	24	24	25	25	25
{1,2,3,4}	0	0	1	1	6	6	7	18	22	22	24	24	28	28	29	40
{1,2,3,4,5}	0	0	1	1	6	6	7	18	22	28	28	29	29	34	34	40

+18 +22

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

Item	Value	Weight
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

Abbiamo calcolato solo il valore di una soluzione ottima

Eserc.: *Scrivere una procedura che costruisce l'insieme di oggetti in una soluzione ottima, dalla tavola*

Il Problema dello Zaino: algoritmo DP

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
∅	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
{1}	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
{1,2}	0	0	1	1	6	6	7	7	7	7	7	7	7	7	7	7
{1,2,3}	0	0	1	1	6	6	7	18	18	20	20	24	24	25	25	25
{1,2,3,4}	0	0	1	1	6	6	7	18	22	22	24	24	28	28	29	40
{1,2,3,4,5}	0	0	1	1	6	6	7	18	22	28	28	29	29	34	34	40

+18 +22

$$OPT(i,w) = \begin{cases} 0 & i = 0 \\ OPT(i-1,w) & w_i > w \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & altrimenti \end{cases}$$

Item	Value	Weight
1	1	2
2	6	4
3	18	7
4	22	8
5	28	9

Abbiamo calcolato solo il valore di una soluzione ottima

Eserc.: *Scrivere una procedura che costruisce l'insieme di oggetti in una soluzione ottima, dalla tavola*

Conclusioni

Teorema

Esiste un algoritmo che risolve il problema dello zaino in tempo O(nW) e spazio O(nW)

- non è polinomiale
  - Es.: se W è esponenzialmente grande rispetto ad n
- non si conosce alcun algoritmo polinomiale per questo problema
- esiste un algoritmi polinomiali che forniscono una soluzione "molto prossima" all'ottimo
  - Il valore della soluzione fornita è almeno 99% del valore della soluzione ottima

Esercizi

- Scrivere una procedura che costruisce l'insieme di oggetti in una soluzione ottima a partire dalla tavola
- Scrivere una versione ricorsiva dell'algoritmo DP per il problema dello Zaino che usa la tecnica della Memoization
- Scrivere una versione dell'algoritmo DP che usa una tavola di una sola riga: complessità di spazio O(W).
- Non è possibile ricostruire l'intera soluzione se si usa una tavola di una riga sola.
  E' possibile ricostruire l'intera soluzione se si usa una tavola di due righe?