

# Algoritmi (modulo di laboratorio)

Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano



- Lezioni: Martedì 8.30 - 10.30 in aula 8 Mercoledì 10.30 - 13.30 in aula 309  
Giovedì 16.30 - 18.30 in aula 307 Venerdì 10.30 - 12.30 in aula 4
- Ricevimento: su appuntamento (Dipartimento di Informatica)
- E-mail: [roberto.cordone@unimi.it](mailto:roberto.cordone@unimi.it)
- Pagina web: <http://homes.di.unimi.it/~cordone/courses/2024-algo/2024-algo.html>
- Sito Ariel: <https://mgoldwurmasd.ariel.ctu.unimi.it>

I problemi di Ottimizzazione Combinatoria (OC) sono definiti da

- un insieme base finito  $B$
- una regione ammissibile  $X \subseteq 2^B$  i cui elementi, detti soluzioni, sono opportuni sottoinsiemi  $x \subseteq B$  dell'insieme base
- una funzione obiettivo  $f : X \rightarrow \mathbb{N}$  che dà un valore a ogni soluzione

Si tratta di trovare una soluzione di valore minimo o massimo

$$\min_{x \in X} f(x) \quad \text{oppure} \quad \max_{x \in X} f(x)$$

I problemi di Ottimizzazione Combinatoria hanno moltissime applicazioni

I problemi di *OC* hanno un numero finito di soluzioni

**Algoritmo esaustivo:** valuta tutte le soluzioni e ne trova una ottima

*E allora dove sta il problema?*

**L'algoritmo esaustivo**

- valuta  $O(2^{|B|})$  sottoinsiemi  $x \subseteq B$
- per ognuno valuta se è una soluzione ammissibile ( $x \in X$ )
- per ogni soluzione ammissibile valuta il valore ( $f(x)$ )

Anche se le due valutazioni sono polinomiali, l'algoritmo è **esponenziale**

La ricerca matematica mostra che

- alcuni problemi di *OC* ammettono algoritmi polinomiali esatti
- tutti i problemi di *OC* ammettono algoritmi polinomiali **euristici**,  
cioè che **non garantiscono di trovare l'ottimo su ogni istanza**

*Algoritmi "non corretti", a rigore, però utili in pratica*

# Il problema dello zaino ( $KP$ )

Si vuole scegliere da un insieme di oggetti voluminosi un sottoinsieme di valore massimo che si possa racchiudere in uno zaino di capacità limitata

- un insieme  $B$  di oggetti elementari
- una funzione  $v : B \rightarrow \mathbb{N}$  che descrive il volume di ogni oggetto
- un numero  $V \in \mathbb{N}$  che descrive la capacità di uno zaino
- una funzione  $\phi : B \rightarrow \mathbb{N}$  che descrive il valore di ogni oggetto

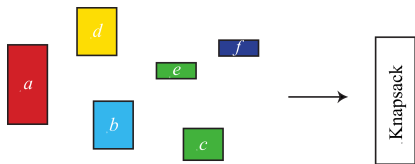
La regione ammissibile contiene i sottoinsiemi di oggetti di volume totale non superiore alla capacità dello zaino

$$X = \left\{ x \subseteq B : \sum_{j \in x} v_j \leq V \right\}$$

L'obiettivo è massimizzare il valore complessivo degli oggetti scelti

$$\max_{x \in X} f(x) = \sum_{j \in x} \phi_j$$

# Esempio



$B$	a	b	c	d	e	f
$\phi$	7	2	4	5	4	1
$v$	5	3	2	3	1	1

$$V = 8$$

$$x' = \{c, d, e\} \in X$$
$$f(x') = 13$$

$$x'' = \{a, c, d\} \notin X$$
$$f(x'') = 16$$



# Un algoritmo ricorsivo per il problema dello zaino

Qualsiasi problema di zaino  $(B, \phi, v, V)$  si può risolvere riconducendolo a due problemi di zaino più piccoli

Considerato un oggetto qualsiasi (per es., l'oggetto  $i = n$ ), si può solo:

- 1 rifiutare l'oggetto
- 2 accettare l'oggetto (se non è troppo grande:  $v_n \leq V$ )

In entrambi i casi, ciò che rimane è ancora un problema di zaino:

- 1 se  $n$  viene rifiutato, si tratta di:
  - riempire uno zaino di capacità  $V$  con oggetti tratti da  $B \setminus \{n\}$
- 2 se  $n$  viene accettato, si tratta di:
  - riempire uno zaino di capacità  $V - v_n$  con oggetti tratti da  $B \setminus \{n\}$
  - aggiungere l'oggetto  $n$  alla soluzione

La soluzione ottima del problema è la migliore fra quelle dei sottoproblemi

# Un algoritmo ricorsivo per il problema dello zaino

Si può quindi scrivere l'equazione ricorrente

$$f^*(n, V) = \begin{cases} 0 & \text{se } n = 0 \\ f^*(n-1, V) & \text{se } n > 0 \text{ e } v_n > V \\ \max[f^*(n-1, V), \phi_n + f^*(n-1, V - v_n)] & \text{se } n > 0 \text{ e } v_n \leq V \end{cases}$$

dove si riconoscono:

- un caso base: insieme di oggetti vuoto
- due casi ricorsivi:
  - ① zaino troppo piccolo per contenere l'ultimo oggetto, per cui si può solo rifiutare l'oggetto stesso
  - ② zaino abbastanza capiente da contenere l'ultimo oggetto, per cui la soluzione è la migliore fra scegliere e rifiutare l'oggetto stesso

I valori  $f^*(n, V)$  sono ottimi di vari problemi di zaino, dove  $n$  identifica

- la restrizione dell'insieme  $B$  al sottoinsieme  $B_n = \{1, \dots, n\} \subseteq B$
- la restrizione delle funzioni  $\phi$  e  $v$  da  $B$  a  $B_n$

# Un algoritmo ricorsivo per il problema dello zaino

$$f^*(n, V) = \begin{cases} 0 & \text{se } n = 0 \\ f^*(n-1, V) & \text{se } n > 0 \text{ e } v_n > V \\ \max[f^*(n-1, V), \phi_n + f^*(n-1, V - v_n)] & \text{se } n > 0 \text{ e } v_n \leq V \end{cases}$$

I vari problemi si costruiscono facilmente riducendo  $n$  e  $V$ :

- per restringere l'insieme base  $B$  e i vettori  $\phi$  e  $v$  agli elementi leciti
- per ottenere la capacità residua

```
if (n == 0)
    return 0;
else if (v[n] > V)
    return AlgoritmoRicorsivoKP(n-1, phi, v, V);
else
{
    f0 = AlgoritmoRicorsivoKP(n-1, phi, v, V);
    f1 = phi[n] + AlgoritmoRicorsivoKP(n-1, phi, v, V-v[n]);
    return max(f0, f1);
}
```





# Programmazione dinamica

La **programmazione dinamica** si basa sull'idea di

- **conservare le soluzioni** dei sottoproblemi **per non ricalcolarle**

Si può realizzare in due modalità

- **top-down**: come **algoritmo ricorsivo** che
  - **divide un problema in sottoproblemi** e li affronta uno per uno
  - **verifica se il sottoproblema corrente è già stato risolto**:
    - **in caso negativo, lo risolve e salva la soluzione** in una struttura
    - **in caso positivo, recupera la soluzione** dalla struttura

*I problemi risolti diventano dei nuovi casi base, risolti in  $\Theta(1)$*

- **bottom-up**: come **algoritmo iterativo** che
  - **elena tutti i sottoproblemi possibili**
  - **risolve i problemi più semplici**
  - **combina le soluzioni dei problemi più semplici per ottenere quelle dei problemi complessi**

# Valutazione di complessità

Per l'**algoritmo ricorsivo**, abbiamo:

- $\Theta(2^n)$  sottoproblemi ammissibili
- tempo  $\Theta(1)$  per risolvere ciascuno

e quindi complessità temporale  $\Theta(2^n)$

Per la **programmazione dinamica** (*top-down* e *bottom-up*), abbiamo

- $\Theta(nV)$  sottoproblemi ammissibili
- tempo  $\Theta(1)$  per risolvere ciascuno

e quindi complessità temporale  $\Theta(nV)$ , che è

- non polinomiale in genere
- polinomiale per  $V$  limitata

( $V$  occupa spazio  $\log V$ )

In gergo si definisce **pseudopolinomiale**