Algoritmi (modulo di laboratorio)

Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano

Lezioni:	Martedì 9.30 - 11.30 in aula Z5	Mercoledì 11.30 - 13.30 in aula 2 e Z5
	Giovedì 14.30 - 17.30 in aula 2 e Z5	Venerdì 11.30 - 13.30 in aula Z5
Ricevimento:	su appuntamento (Dipartimento di Informatica)	
E-mail:	roberto.cordone@unimi.it	
Pagina web:	http://homes.di.unimi.it/~cordone/courses/2021-algo/2021-algo.html	
Sito Ariel:	https://mgoldwurma.ariel.ctu.unimi.it	

Lezione 15: Programmazione dinamica — Milano, A.A. 2020/21

Ottimizzazione Combinatoria

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 $f : X \to \mathbb{N}$ 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

Algoritmi risolutivi

Algoritmo esaustivo: trova una soluzione ottima scorrendole tutte

E allora dove sta il problema?

L'algoritmo esaustivo

- valuta $O\left(2^{|B|}\right)$ 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, il risultato 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 $v : B \to \mathbb{N}$ che descrive il volume di ogni oggetto
- un numero $V \in \mathbb{N}$ che descrive la capacità di uno zaino
- una funzione $\phi : B \to \mathbb{N}$ che descrive il valore di ogni oggetto

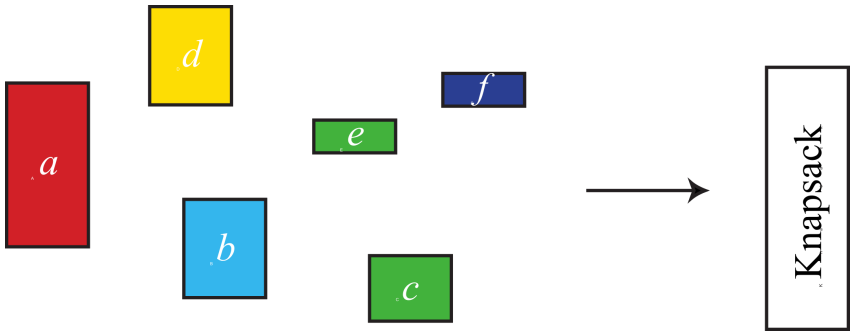
La regione ammissibile contiene i sottoinsiemi di oggetti di volume totale non superiore alla capacità dello zaino

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

L'obiettivo è massimizzare il valore complessivo degli oggetti scelti

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

Esempio

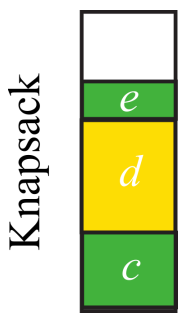


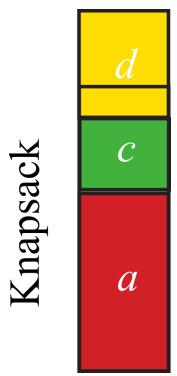
B	a	b	c	d	e	f
ϕ	7	2	4	5	4	1
v	5	3	2	3	1	1

$V = 8$

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

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





Un algoritmo ricorsivo per il problema dello zaino

Qualsiasi problema di zaino (B, ϕ, 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

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

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 $B_n = \{1,\ldots,n\} \subseteq B$
- la restrizione delle funzioni ϕ e v da B a B_n

Un algoritmo ricorsivo per il problema dello zaino

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

I vari problemi si costruiscono facilmente riducendo n e V:

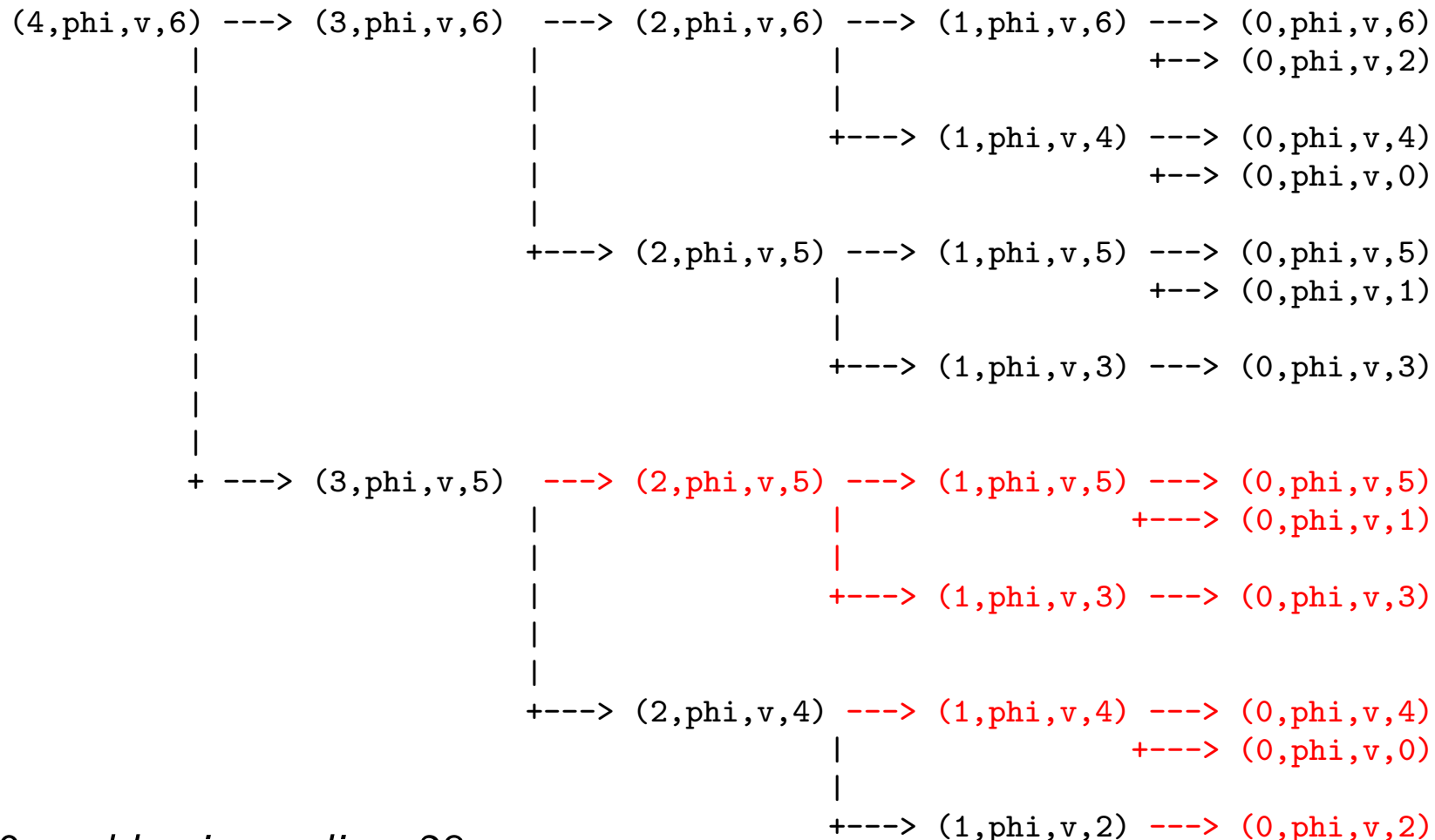
- per restringere l'insieme base B e i vettori `phi` e `v` agli elementi leciti
- per ottenere la capacità residua

```
if (n == 0)
  return 0;
else if (v[n] > V)
  return AlgoritmoRicorsivoKP(n-1,phi,v,V);
else
{
  phi0 = AlgoritmoRicorsivoKP(n-1,phi,v,V);
  phi1 = phi[n] + AlgoritmoRicorsivoKP(n-1,phi,v,V-v[n]);
  return max(phi0,phi1);
}
```

Esempio

È un algoritmo molto inefficiente: risolve ripetutamente problemi uguali

Per esempio, $\phi = [\ 6\ \ 8\ \ 3\ \ 2\ ]$, $v = [\ 4\ \ 2\ \ 1\ \ 1\ ]$ e $V = 6$



10 *problemi uguali su* 28

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
  - elenca 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 (V *occupa spazio* $\log V$)
- polinomiale per V limitata

In gergo si definisce pseudopolinomiale