

Si comincia. . .

- 1 Il corso
- 2 Organizzazione del corso
- 3 Qualche richiamo dei concetti di base
 - Problema computazionale
 - Problemi di ottimizzazione
 - Esempio COMMESO VIAGGIATORE
 - Esempio MASSIMO FLUSSO
 - Classificazione dei problemi di ottimizzazione
 - Considerazioni sui problemi computazionali
 - Esempio di modellizzazione
 - Concetti generali sugli algoritmi
 - Complessità di un algoritmo
 - Analisi caso medio o caso peggiore
 - Algoritmi vs. hardware
 - Algoritmi efficienti e problemi trattabili

Concetto di problema computazionale

Esempio CAMMINO HAMILTONIANO

Esempio 2 (Una soluzione algoritmica)

Algoritmo 1: ricerca esaustiva per CAMMINO HAMILTONIANO

Input: $G = (V, E), x, y$

- 1: $P = \{\text{permutazioni dei nodi di } V \text{ con primo nodo} = x \text{ e ultimo} = y\};$
 - 2: **while** ($P \neq \emptyset$) **do**
 - 3: $p = (v_1, v_2, \dots, v_n) \in P;$
 - 4: $P = P \setminus \{p\};$
 - 5: **if** (p è un cammino) **then return** s_i ; // $(v_i, v_{i+1}) \in E \quad 1 \leq i < n$
 - 6: **endw**
 - 7: **return** *no*
-

Problemi di ottimizzazione

Esempio MASSIMO FLUSSO

PROBLEMA MASSIMO FLUSSO (MAX FLOW)

DESCRIZIONE: Una rete di distribuzione $N = (V, E, c, s, t)$.

QUESITO: determinare la funzione flusso $f : E \rightarrow \mathbf{N}$ tale che:

- 1 non può superare la capacità dei singoli archi:
 $f(i, j) \leq c(i, j) \quad \forall (i, j) \in E;$
- 2 in ciascun nodo diverso da s e t deve soddisfare il principio di conservazione del flusso: $\forall i \in V \setminus \{s, t\}$
 $\sum_j f(i, j) = \sum_j f(j, i);$
- 3 massimizzi il flusso uscente da s : $\max_f \sum_i f(s, i).$

MASSIMO FLUSSO = $(I, \text{SOL}, m, \text{goal})$ dove:

- $I = \{\text{rete di distribuzione } N = (V, E, c, s, t)\}.$
- $\text{SOL}(N) = \{f \mid f \text{ soddisfa i vincoli dei punti 1 e 2}\}.$
- $m(N, f) = \sum_i f(s, i).$
- $\text{goal} = \max.$

Problemi di ottimizzazione

Classificazione dei problemi di ottimizzazione

I problemi di ottimizzazione possono essere suddivisi in due categorie naturali:

Problemi nel continuo: i parametri e la funzione obiettivo assumono valori nel campo dei reali.

Problemi nel discreto: i parametri e la funzione obiettivo assumono valori in un campo di valori finito o infinito numerabile (ad esempio \mathbf{Z}).

I problemi di questo tipo sono anche chiamati **problemi di ottimizzazione combinatoriali**.

Nota!

In questo corso si considerano soprattutto algoritmi per problemi combinatoriali.

Classificazione dei problemi di ottimizzazione

Inoltre, dato un problema di ottimizzazione \mathcal{P} , si possono considerare 3 differenti **versioni**:

Versione costruttiva \mathcal{P}_C : la soluzione deve determinare l'**argomento** ottimo della funzione.

Versione di valutazione \mathcal{P}_E : la soluzione deve determinare il **valore** ottimo della funzione.

Versione di decisione \mathcal{P}_D : la soluzione deve determinare se il **valore** ottimo della funzione è maggiore (minore) di un valore soglia dato come parametro.

Classificazione dei problemi di ottimizzazione

Teorema 1

Per qualsiasi problema \mathcal{P} in cui le funzioni $\text{SOL}()$ e $m()$ possono essere computate in tempo polinomiale e i valori di $\text{SOL}()$ sono limitati polinomialmente, si dimostra che

$$\mathcal{P}_D \equiv_T^P \mathcal{P}_E \leq_T^P \mathcal{P}_C$$

dove \leq_T^P è la riduzione di Turing calcolabile in tempo polinomiale e \equiv_T^P è l'equivalenza di Turing (ovvero, \leq_T^P e \geq_T^P).

Classificazione dei problemi di ottimizzazione

Mediante la ricerca binaria si può dimostrare che vale anche $\mathcal{P}_E \leq_T^P \mathcal{P}_D$ visto che $\text{SOL}()$ è limitata polinomialmente.

Per quanto riguarda $\mathcal{P}_C \leq_T^P \mathcal{P}_E$, in generale non è detto che ci sia un modo per costruire la versione costruttiva dalla versione di valutazione. Si dimostra però che

Teorema 2

Se un problema \mathcal{P} ha la versione di decisione NP-completa, allora $\mathcal{P}_C \leq_T^P \mathcal{P}_D$,

Rimane aperta la questione di dimostrare se esistono problemi reali con versione costruttiva più difficile della versione di valutazione. Ci sono dei risultati che portano a pensare che esistano!

Problemi di ottimizzazione

Considerazioni sui problemi computazionali

Processo di risoluzione

Problema reale $\xrightarrow{1}$ Modello del problema $\xrightarrow{2}$ Soluzione algoritmica

- Tutti i modelli sono semplificazioni dei problemi reali.
- La soluzione quindi è una soluzione per il modello e non per il problema reale.
- Solo se il modello è molto “fedele” al problema originale, allora si può dire che si è trovata una soluzione significativa anche per il problema reale!

Nota!

Capita spesso di vedere modellazioni troppo semplicistiche di problemi reali \Rightarrow soluzioni praticamente inutili!

Problemi di ottimizzazione

Esempio di modellazione

Esempio 3 (Ordine di colorazione di auto)

In una fabbrica di auto esiste solo un impianto di colorazione e l'esigenza di produrre auto in diversi colori. Ogni cambio di colore ha un costo di produzione. Si deve determinare l'ordine dei colori con cui colorare le auto (*schedule*) tale da minimizzare il costo totale.

Si supponga che passare dal rosso al nero costi 30 unità di lavoro mentre il viceversa costi 80, dal rosso al bianco 80, il viceversa 10, ecc.

Come si può modellare il problema?

Si può usare il **COMMESSE VIAGGIATORE**?

Se sì, quali aspetti del problema possono non essere rappresentati?

Problemi di ottimizzazione

Esempio di modellazione 2

Esempio 4 (Distribuzione di carta)

Una società ha n magazzini di carta che devono servire k centri di distribuzione. Ciascuna consegna da un magazzino i a un centro j ha un costo $c_{i,j}$. La funzione c è specifica per ciascuna coppia (i,j) .

Ad esempio:

$$c_{2,3}(x) = \begin{cases} 0 & \text{se } x = 0 \\ 4 + 3,33x & \text{se } 0 < x \leq 3 \\ 0,5 + 10\sqrt{x} & \text{se } 3 < x \end{cases}$$

Determinare il costo minimo di consegna della carta ai centri di distribuzione.

Problemi di ottimizzazione

Esempio di modellazione 2

Un possibile modello per il problema:

$$\min \sum_{i=1}^n \sum_{j=1}^k c_{i,j}(x_{i,j})$$

con i vincoli

$$\sum_{j=1}^k x_{i,j} \leq \text{source}(i), \quad 1 \leq i \leq n$$

$$\sum_{i=1}^n x_{i,j} \geq \text{dest}(j), \quad 1 \leq j \leq k$$

$$x_{i,j} \geq 0 \quad 1 \leq i \leq n, \quad 1 \leq j \leq k$$

dove $\text{source}()$ è la disponibilità di un magazzino e $\text{dest}()$ è la richiesta di un centro.

Troppo difficile da risolvere per molti algoritmi di ottimizzazione tradizionali solo perché $c()$ è discontinua!

Problemi di ottimizzazione

Meglio un modello preciso difficile o uno più semplice facile?

Quali sono le alternative?

- si semplifica il modello in modo da poter usare algoritmi di ottimizzazione noti in grado di fornire soluzioni di qualità.

Problema reale $\xrightarrow{1}$ Modello_a $\xrightarrow{2}$ Soluzione_p(Modello_a)

- si mantiene il modello e si provano metodi non tradizionali per determinare soluzioni quasi ottimali.

Problema reale $\xrightarrow{1}$ Modello_p $\xrightarrow{2}$ Soluzione_a(Modello_p)

dove _a indica “approssimato” e _p indica “preciso”.

In entrambi i casi si trovano soluzioni approssimate!

Alcuni autori riferiscono che spesso il secondo approccio porta a migliori risultati.

Concetti generali sugli algoritmi

Risorse computazionali di un algoritmo

- L'esecuzione di un algoritmo richiede il consumo di risorse computazionali: spazio, **tempo**, processori, ecc.
- In questo corso si valuterà il **costo o complessità** di un algoritmo quasi sempre in termini di tempo richiesto per la sua esecuzione.
- In generale, il tempo richiesto da un algoritmo cresce con la **dimensione dell'input**.
- Ribadiamo quindi il concetto di **dimensione dell'input** e **tempo di esecuzione**.

Concetti generali sugli algoritmi

Dimensione di un input

Definizione 3 (Dimensione di un input)

Dimensione di un input = lunghezza della stringa che rappresenta l'input secondo una codifica "ragionevole".

(vedi Corso di complessità).

In pratica, si adotta come dimensione dell'input il valore o i valori dei parametri più significativi dell'istanza, a seconda del problema:

- Per il problema della moltiplicazione di due interi, la dimensione è il numero totale di bit necessari per rappresentare i due interi.
- Per il problema dell'ordinamento, la dimensione è il numero totale di bit per rappresentare gli elementi, **ma spesso si considera solo il numero degli elementi.**
- ...

Concetti generali sugli algoritmi

Tempo di esecuzione

Una definizione *robusta* del concetto di tempo di esecuzione deve essere indipendente dal particolare modello di calcolo che si vuole usare.

Definizione 4 (Tempo di esecuzione per un dato input)

Dato un input e un algoritmo,
tempo di esecuzione su tale input = # operazioni elementari o passi eseguiti

Un'operazione è **elementare** quando il suo tempo di esecuzione è indipendente dalla dimensione degli operandi.

La definizione di operazione **elementare** è critica in quanto potrebbe escludere operazioni semplici come la moltiplicazione se non si fanno opportune assunzioni.
(vedi Corso di complessità).

Concetti generali sugli algoritmi

Analisi caso medio o caso peggiore

A parità di dimensione, due istanze diverse possono richiedere tempi diversi per essere risolte.

Algoritmo 2: Ordinamento per inserimento

Input: $A = (a_1, \dots, a_n)$

```
1: for (i = 2; i ≤ n; i++) do
2:   x = A[i]; j = i - 1;
3:   while (j > 0 && x < A[j]) do
4:     A[j + 1] = A[j]; j --;
5:   endw
6:   A[j + 1] = x;
7: endfor
8: return A
```

Dimensione dell'istanza = n .

tempo = #confronti + #assegnamenti:

- se input = **vettore ordinato**

allora tempo =

$$\underbrace{5(n-1)}_{\text{righe 2-6}} + \underbrace{1+n+n-1}_{\text{riga 1}} = O(n)$$

- se input = **vettore inversamente ordinato** allora

tempo =

$$\sum_{i=2}^n (\underbrace{3}_{\text{righe 2,6}} + \underbrace{4(i-1)+1}_{\text{righe 3-5}}) + \underbrace{2n}_{\text{riga 1}} =$$

$$4(n-1) + 2n(n-1) + 2n = O(n^2)$$

Concetti generali sugli algoritmi

Analisi caso medio o caso peggiore

- Stima complessità “conservativa”:

Analisi del caso peggiore

Il tempo è determinato considerando, per ciascuna dimensione, l'istanza che richiede maggior tempo di calcolo.

- Stima complessità “media”:

Analisi del caso medio

Il tempo è determinato calcolando, per ciascuna dimensione, la media dei tempi di calcolo per le istanze di quella dimensione.

- Il calcolo del caso medio non è sempre possibile: per una buona stima occorre conoscere la distribuzione di probabilità delle istanze del problema.

Concetti generali sugli algoritmi

Ordini di grandezza per la complessità

- La complessità in tempo di un algoritmo permette di stabilire un limite superiore al tempo di calcolo reale dell'algoritmo:
tempo reale = tempo 1 operazione elementare × complessità
- **Principio di invarianza:** L'esperienza dimostra che due differenti implementazioni dello stesso algoritmo hanno tempo di calcolo che non differisce per più di una costante moltiplicativa.
- È possibile quindi definire la complessità di un algoritmo in termini di **ordini di grandezza** del tipo $O()$, $\Omega()$, $\Theta()$.

Concetti generali sugli algoritmi

Classificazione degli algoritmi \Leftrightarrow ordini di grandezza

- Algoritmi che hanno complessità $O(n^k)$, con k costante, sono detti **algoritmi polinomiali in tempo**.
- Tutti gli altri con complessità superiore sono detti, genericamente, **algoritmi esponenziali in tempo**.

Concetti generali sugli algoritmi

Pericoli con gli ordini di grandezza

- Se il principio di invarianza ci permette di trascurare le costanti, **questo non significa che dobbiamo dimenticarle!**
- Se due algoritmi risolutori per un problema richiedono, rispettivamente, n^2 giorni e n^3 secondi di calcolo, affinché il primo algoritmo sia effettivamente migliore del secondo occorre risolvere un'istanza da 20 milioni di anni!
- eppure, asintoticamente, il primo algoritmo è migliore del secondo!

Concetti generali sugli algoritmi

Pericoli con gli ordini di grandezza

In sintesi, nella comparazione di due algoritmi si deve:

- fissare una misura comune per la dimensione dell'istanza;
- fissare quali sono le operazioni elementari da contare;
- determinare la complessità considerando anche le costanti;
- considerare gli ordini di grandezza solo se le costanti moltiplicative sono del medesimo ordine.

Concetti generali sugli algoritmi

Algoritmi vs. hardware

Legge di Moore

Ogni 18 mesi circa la velocità dei processori raddoppia.

- Ha senso allora cercare algoritmi sempre più efficienti?
- Non è sufficiente aspettare un miglioramento tecnologico per risolvere istanze sempre più grandi con il medesimo algoritmo?

Ovviamente NO!

Non sempre un miglioramento dell'hardware determina un pari aumento delle "prestazioni" di un algoritmo.

Concetti generali sugli algoritmi

Algoritmi vs. hardware: esempio

Esercizio 1

Secondo il benchmark *SiSoftware Sandra 2007 - Arithmetic MFLOPS*, il processore Intel Core 2 DUO T7700 (processore di un PowerBook Pro @ 2.4 GHz) ha una prestazione di 15 383 MFLOPS. Determinare quanto tempo richiede un'istruzione aritmetica di tipo floating-point e, assumendo che siano paragonabili a quelle dell'algoritmo A , quanto più veloce è questo processore rispetto al processore C' .

Determinare quindi la nuova tabella dei tempi per l'algoritmo A .

Concetti generali sugli algoritmi

Algoritmi vs. hardware

Cambio strategia: si investe sull'algoritmo!

$A_1 =$ algoritmo con complessità $O(n^3)$.

$C_1 =$ programma-computer che computa A_1 in $10^{-4}n^3$ secondi.

Dimensione	Tempo con C_1
10	$\approx 0.1''$
20	$\approx 1''$
30	$\approx 3''$
38	$\approx 5''$
6 800	≈ 1 anno

In generale, fissato il tempo per risolvere un'istanza di dimensione n con C_1 , C'_1 , computer 100 volte più veloce, permette di risolvere un'istanza di dimensione $\approx \sqrt[3]{100n}$ nel medesimo tempo.

