Introduzione

Prog. di Algoritmi 2021/2022

Massimo Lauria <massimo.lauria@uniroma1.it>

Benvenuti

Riscaldamento di benvenuto

Il Massimo Comun Divisore (MCD) di due numeri interi positivi è il più grande intero positivo che divida entrambi.

Problema: dati due numeri interi $0 < a \leqslant b$ trovate il MCD

Riscaldamento di benvenuto

Il Massimo Comun Divisore (MCD) di due numeri interi positivi è il più grande intero positivo che divida entrambi.

Problema: dati due numeri interi $0 < a \leqslant b$ trovate il MCD

L'algoritmo di Euclide calcola il MCD come segue:

- calcola il resto r di b diviso a
- se r è uguale a 0 allora il MCD tra a e b è a
- altrimenti il MCD tra a e b è uguale al MCD tra r e a

Il pensiero computazionale

È quella capacità, davanti ad un problema, di effettuare i seguenti quattro passi:

1. formulazione **non ambigua** (astrazione)
2. trovare **passi logici** che portino a soluzione (algoritmo)
3. codificare in un **programma** (programmazione)
4. testare/analizzare la qualità della soluzione (feedback)

Il pensiero computazionale

È quella capacità, davanti ad un problema, di effettuare i seguenti quattro passi:

1. formulazione **non ambigua** (astrazione)
2. trovare **passi logici** che portino a soluzione (algoritmo)
3. codificare in un **programma** (programmazione)
4. testare/analizzare la qualità della soluzione (feedback)

Il corso si focalizza sul **punto 2**, la parte algoritmica.

A cosa serve un algoritmo?

Si parte da un **problema computazionale** e si vuole arrivare ad un insieme di tecniche **logiche** per risolverlo.

Queste tecniche logiche sono **algoritmi** e sono costituiti da

- modi di **organizzare i dati**
- schemi di **manipolazione** e **calcolo**

Il tutto per arrivare ai **risultati** desiderati.

Un problema computazionale

Un problema la cui **formulazione** e **soluzione** può essere descritta in forma non ambigua e che quindi può essere potenzialmente risolta da un programma.

E.g. calcolare la media di questi 100 numeri (preciso)

E.g. trova su facebook il mio più caro amico (ambiguo)

Algoritmi e problemi reali

Nel modo reale tutto è ambiguo e definito in maniera informale. Diventa necessario trovare un **modello** più astratto e non ambiguo del problema reale in campo.

Individuare le componenti **essenziali** di un problema reale è un lavoro *fisolofico*, *sociologico*, *antropologico* con i suoi compromessi e conflitti.

Non è comunque la fine della storia. Il modello può non essere adatto, cosa che si scopre **dopo** aver risolto i problemi algoritmici.

Algoritmi ”classici” e ”moderni”

La potenza e la diffusione dei computer, e la **grande disponibilità di dati** ha permesso di usare metodi statistici per tentare di risolvere problemi **non ben compresi dal punto di vista logico** (e.g. ”trova il gatto nella foto”)

Algoritmi ”classici” e ”moderni”

La potenza e la diffusione dei computer, e la **grande disponibilità di dati** ha permesso di usare metodi statistici per tentare di risolvere problemi **non ben compresi dal punto di vista logico** (e.g. ”trova il gatto nella foto”)

Noi ci occupiamo invece di algoritmi **vecchio stampo**, che però sono alla base di questi metodi più moderni.

- problemi ben definiti
- soluzioni non ambigue

Cosa faremo in questo corso?

Le attività di questo corso

Algoritmi: descriviamo tecniche classiche per risolvere problemi computazionali, soprattutto legati alla teoria dei grafi.

Correttezza: dimostreremo matematicamente la correttezza degli algoritmi proposti.

Efficienza: valuteremo formalmente il costo computazionale

Risoluzione dei problemi: cercheremo di utilizzare le tecniche apprese per risolvere problemi nuovi.

Il punto di partenza: il problema

Abbiamo un **problema computazionale** da risolvere, ovvero una relazione tra degli **input** e degli **output** corrispondenti.

Il punto di partenza: il problema

Abbiamo un **problema computazionale** da risolvere, ovvero una relazione tra degli **input** e degli **output** corrispondenti.

Esempio: ordinare n numeri in modo ascendente

Input: una sequenza di n numeri

$$a_1, a_2, \ldots, a_n$$

Output: una permutazione $a'_1, a'_2, \ldots, a'_n$ dell'input, tale che

$$a'_1 \leqslant a'_2 \leqslant \ldots \leqslant a'_n$$

Il punto di arrivo: l'algoritmo

Una procedura che

- dato un **input**;
- calcola un **output**
- in modo **automatico**

I passi dell'algoritmo sono

- **elementari**
- descritti in maniera **univoca**

Termina sempre in **tempo finito**

Figure: Muḥammad ibn Mūsā al-Khwārizmī (780–850 ca.) Matematico Persiano inventore dell'algebra, e quindi della manipolazione simbolica di elementi matematici. La parola algoritmo deriva dal suo nome.

Differenza tra algoritmo e implementazione

Un algoritmo è una **descrizione precisa** di una procedura di calcolo, ma non è la sua realizzazione attraverso un programma.

L'**implementazione** è la realizzazione di un algoritmo in un programma per computer. Tipicamente si deve occupare di dettagli che sono irrilevanti alla descrizione dell'algoritmo.

```
def MCD(a, b) :                    1
    a, b = min(a,b), max(a,b)      2
    while a > 0 :                  3
        a, b = b % a, a            4
    return b                       5
```

```
def MCD(a,b) :                     1
    a, b = min(a,b), max(a,b)      2
    return MCD_aux(a,b)            3
                                   4
def MCD_aux(a,b) :                 5
    if a == 0 :                    6
        return b                   7
    else :                         8
        return MCD_aux(b%a,a)      9
```

Cosa è importante in un algoritmo

Correttezza: l'algoritmo fa quello vogliamo?

Efficienza: quante operazioni e quanta memoria sono necessarie per arrivare al risultato.

Efficienza computazionale

Una stima **asintotica** dell'**ordine di crescita** del numero di operazioni che impiega l'algoritmo (**tempo**), o della quantità di memoria utilizzata (**spazio**).

- la memoria non è infinita ed ha un costo
- i computer non sono infinitamente veloci

Efficienza computazionale

Una stima **asintotica** dell'**ordine di crescita** del numero di operazioni che impiega l'algoritmo (**tempo**), o della quantità di memoria utilizzata (**spazio**).

- la memoria non è infinita ed ha un costo
- i computer non sono infinitamente veloci

Notazione Θ, Ω, O. Ad esempio

- Bubblesort ha complessità $\Theta(n^2)$
- Mergesort ha complessità $O(n \log n)$ ma richiede spazio $\Theta(n)$
- Radixsort ha complessità $O(n)$ ma non è generale
- Gli ordinamenti per confronto richiedono tempo $\Omega(n \log n)$.

Efficienza ”in pratica”

Le misure di complessità sono teoriche poiché **indipendenti** dagli avanzamenti tecnologici e dalle contingenze. Tuttavia è sempre utile avere in mente una tabella come la seguente, con i tempi di esecuzione *reali* di operazioni tipiche.

Accesso ai dati	tempo (in ns)
memoria cache L1	0.5
memoria cache L2	7
RAM	100
1MB seq. da RAM	250.000
4K random disco rigido SSD	150.000
1MB seq. da disco SSD	1.000.000
disco rigido HDD	8.000.000
1MB seq. da disco HDD	20.000.000
pacchetto dati Europa -> US -> Europa	150.000.000

Fonte: Peter Norvig, `http://norvig.com/21-days.html`

Updated by: Jeff Dean `https://ai.google/research/people/jeff`}

Tecniche algoritmiche che vedremo

La maggior parte degli algoritmi che vedremo riguardano **grafi** semplici o diretti, ma non solo. Svilupperemo varie tecniche **generali**:

- algoritmi greedy (trad. avidi o golosi)
- divide et impera
- programmazione dinamica
- backtracking

Presentazione degli algoritmi

Durante le lezioni presenterò diversi algoritmi in una forma intermedia di

- linguaggio parlato
- pseudo-codice
- python (che spesso è quasi come lo pseudo codice)

Esercizio suggerito:

- implementare gli algoritmi discussi
- scambiatevi degli input utili tra voi

Informazioni Pratiche

Docente del corso

- **Docente**: Massimo Lauria
- **Email:** massimo.lauria@uniroma1.it
- **Dipartimento**: Scienze statistiche (CU002)
- **Ufficio:** Stanza n.9 - 4^o piano
- **Ricevimento:** Giovedì 11.00-13.00 (via Zoom, scrivere prima)

Orari e Aule

Lezioni regolari

- Giovedì, ore 16.00-19.00 (Aula Magna ed. RM111)
- Venerdì, ore 14.00-16.00 (Aula 2L ed. RM018)

Esercitazioni (con Matteo Cinelli <cinelli@di.uniroma1.it>)

- Lunedì, ore 14.00-16.00 (Aula 2L ed. RM018)

Libri di testo

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein
Introduzione agli algoritmi e strutture dati
MacGraw-Hill. Testo in **italiano**.

S.Dasgupta, C. Papadimitriou, U. Vazirani
Algorithms
Disponibile anche online. Testo in **inglese**.

Testi potenzialmente utili

Horowitz, Sahni *Fundamentals of Computer Algorithms*
Kleimberg, Tardos *Algorithm Design*, 2005
Demetrescu, Finocchi, Italiano *Algoritmi e Strutture di Dati*, 2010
Crescenzi, Gambosi, Grossi, Rossi *Strutture di dati e algoritmi*, 2012

Materiale aggiuntivo

Pagina del corso: `https://twiki.di.uniroma1.it/twiki/view/Algoritmi2/WebHome`

Troverete

- aggiornamenti e notizie
- le indicazioni **di massima** del programma
- diario del corso
- libri di testo
- contatti dei docenti
- orari e aule
- materiale aggiuntivo (mio e dei colleghi ;-))

Partecipazione degli studenti

Per superare (quasi) indenni questo corso

- fate domande
- fate gli esercizi (ce ne saranno in abbondanza) man mano che studiate, per **verificare** e **migliorare** lo studio
- lavorare con gli altri studenti, in aula e fuori

Gruppo google (link):

- faremo gli annunci
- potrete discutere tra voi, aiutarvi, ecc…

Esame

Un esame scritto alla fine del corso.