Algoritmi e Strutture Dati

Introduzione al Corso

Maria Rita Di Berardini (Camerino), Emanuela Merelli (Ascoli)[1]

[1]Dipartimento di Matematica e Informatica
Università di Camerino

Struttura del corso

Il corso consiste di

- 45 ore di lezione
- 2 ore di ricevimento settimanali

Il corso viene valutato in

- 6 CFU (Crediti Formativi Universitari)

Docente del corso di
Laboratorio di Algoritmi e strutture dati:
Dott. Ezio Bartocci (Camerino)
Dott. Francesco De Angelis (Ascoli)

Orario del Corso – Camerino

I semestre: dal 10 ottobre 2006 al 31 gennaio 2007

I periodo:	10 ottobre - 17 novembre	6 settimane	22 (20) h lezione e 18 (15) h laboratorio
II periodo:	4 dicembre - 31 gennaio	6 settimane	28 (25) h lezione e 21 (15) h laboratorio
pausa natalizia:	21 dicembre al 6 gennaio		

Dal 9 ottobre 2006 al 31 gennaio 2006

Martedì dalle 08:00 alle 10:00

Giovedì dalle 09:00 alle 12:00 (Lab)

Venerdì dalle 09:00 alle 11:00

Orario del Corso – Ascoli

I semestre: dal 10 ottobre 2006 al 31 gennaio 2007

I periodo:	10 ottobre - 17 novembre	6 settimane	22 (20) h lezione e 18 (15) h laboratorio
II periodo:	4 dicembre - 31 gennaio	6 settimane	28 (25) h lezione e 21 (15) h laboratorio
pausa natalizia:	21 dicembre al 6 gennaio		

Dal 9 ottobre 2006 al 31 gennaio 2006

Giovedì dalle 11:00 alle 13:00

Venerdì dalle 9:00 alle 11:00

Orario di ricevimento

Camerino

Mercoledì dalle 15:00 alle 17:00

Primo Piano, Polo Informatico, stanza 9

Ascoli

Giovedì dalle 15:00 alle 17:00

Polo di Scienze

Materiale del Corso

Pagina Web

http://www.cs.unicam.it/merelli/algoritmi/algoritmi.htm

Struttura del corso

Il corso è strutturato in sette parti:

1. **Introduzione agli algoritmi, ai modelli di calcolo e alle metodologie di analisi** (6 ore)
2. **Strutture dati elementari** (8 ore)
3. **Algoritmi di ordinamento** (8 ore)
4. **Alberi binari di ricerca** (6 ore)
5. **Algoritmi su stringhe** (6 ore)
6. **Tecniche avanzate di analisi e progettazione di algoritmi** (5)
7. **Grafi e visite di grafi** (6)

Obiettivi del Corso

Il corso si prefigge di fornire i saperi necessari per:

- Analizzare le principali tecniche di progettazione degli algoritmi
- Affrontare in maniera integrata la classificazione, l'analisi, la progettazione e la realizzazione di algoritmi
- Identificare le scelte algoritmiche fondamentali e valutare i costi in termini di efficienza computazionale
- Scegliere e realizzare strutture dati adeguate al caso specifico che si sta affrontando
- Raggiungere adeguati compromessi tra esigenze conflittuali (costo, semplicità, efficienza)

Obiettivi del Corso

- Progetto di algoritmi **corretti**, ovvero che risolvono sempre e solo il problema a cui si è interessati, attraverso l'esame di diversi paradigmi
- Progetto di algoritmi **efficienti**, ovvero che risolvono il problema il più velocemente possibile o usano il minor spazio di memoria possibile
- Sono risultati attesi, alla fine del corso, la conoscenza delle tecniche di base di progettazione e analisi degli algoritmi

Prerequisiti del corso

Per seguire il corso è necessario conoscere

- concetto di ricorsione
- strutture dati fondamentali
- dimostrazioni per induzione e calcolo infinitesimale

Programma del Corso

- **Introduzione agli algoritmi**
- **Modelli di calcolo e metodologie di analisi**
- **Strutture dati elementari**
- **Algoritmi di Ordinamento**
- **Alberi binari di ricerca**
- **Algoritmi su stringhe**
- **Tabelle Hash**
- **Code con priorità**
- **Tecniche avanzate di analisi e progettazione**
- **Grafi e visite di grafi**

Testi di Riferimento e consultazione

C1. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, **Introduzione agli Algoritmi**, McGrawHill, seconda edizione

C0. L. Margara, V. Maniezzo, **Lezioni di algoritmi**, Pitagora 2002

R1. C.Demetrescu, I. Finocchi, G. Italiano, **Algoritmi e Strutture Dati**, McGraw-Hill, 2004

C2. Aiello, Albani, Attardi, Monatanari, **Teoria della computabilità, logica, teoria dei linguaggi formali**, ETS, 1976 o succ.

C3. G. Ausiello, A. Marchetti-Spaccamela, M. Protasi, **Teoria e Progetto di Algoritmi Fondamentali**, Franco Angeli, 1994 o succ.

C4. N. Wirth, **Algortimi + Strutture Dati = Programmi**, TecnicheNuove, 1987 o succ.

C5. M.R. Garey and D.S. Johnson, **Computers and Intractability: A Guide to the Theory of NP-Completeness**, W.H. Freeman and Co ed. – 1979

Modalità d'esame

- L'esame consiste in una prova scritta ed una eventuale prova orale
- La valutazione della prova scritta vale solo per la sessione d'esame in cui è stata sostenuta
- Ogni studente può provare l'esame al più due volte in un anno accademico
- Per sostenere l'esame è necessario iscriversi all'esame tramite pagina web: web.unicam.it/matinf

È possibile sostenere l'esame sostenendo due prove parziali

Appelli d'Esame

I semestre
7 febbraio 2007
21 febbraio 2007

II semestre
19 giugno 2007
10 luglio 2007

Sessione di recupero
18 settembre 2007
9 ottobre 2007

Parte I

Il concetto di Algoritmo

Il concetto di "algoritmo"

Informalmente, un algoritmo è una procedura computazionale ben definita eseguita per risolvere un dato problema computazionale

È una sequenza di passi computazionali che prende dei valori come input e produce altri valori come output

Più precisamente, un algoritmo è un procedimento di calcolo esplicito, descrivibile con un numero finito di regole che conduce al risultato dopo un numero finito di operazioni, cioè di applicazioni delle regole

Un esempio

Definizione del problema: ricerca del minimo in un array:

$$min(A[1 \dots n]) = a \text{ sse } a \leq A[i] \text{ per ogni } i = 1, \dots, n$$

(stabilisce una relazione tra input e output)

Algoritmo (descrive una procedura computazionale per realizzare tale relazione):

```
min(A)
    a ← A[1]
    for i ← 2 to length[A]
        do if A[i] < a
                a ← A[i]
    return a
```

```
min(A) //se A è ordinato
    return A[1]
```

Un pò di storia

Etimologia:
Il termine **algoritmo** significa procedimento di calcolo

Deriva dal termine latino medievale **algorismus**, che a sua volta deriva dal nome del matematico usbeco *Abu Jafar Mohammad ibn-Musa al-Khowarismi*, vissuto nel IX (?) secolo

Algoritmi nella storia

- Algoritmi di tipo numerico sono stati studiati da babilonesi e indiani
- Algoritmi in uso ancora oggi sono stati studiati da matematici greci 2000 anni fa (Algoritmo di Euclide per il MCD, algoritmi geometrici, ...)

Perchè parliamo di algoritmi

La teoria degli algoritmi ha iniziato a stabilizzarsi agli inizi del XX secolo, mentre ...

Le tecniche di progettazione di algoritmi e di analisi di correttezza e di efficienza si sono evolute nella seconda metà del XX secolo grazie alla diffusione dei calcolatori elettronici

Ovunque si impieghi un calcolatore occorrono algoritmi corretti e efficienti che ne utilizzino al massimo le possibilità. Esempi di algoritmi efficienti:

- controllo dei voli aerei
- regolazione reattori nucleari
- reperimento d'informazioni da archivi
- smistamento di comunicazioni telefoniche
- gioco degli scacchi
- controllo della produzione di una catena di montaggio

Come valutiamo gli algoritmi

Risolve correttamente il problema?

- un algoritmo si dice corretto se, per ogni istanza di input, si ferma con l'output corretto
- un algoritmo corretto risolve il problema computazionale dato
- dimostrazione matematica, descrizione informale

Risolve il problema in maniera efficiente (analisi di algoritmi)?

- definizione di **efficienza** (tempo o memoria)
- alcuni problemi non possono essere risolti in maniera efficiente
- esistono delle soluzioni ottime: non è possibile fare di meglio

Algoritmi e Programmi

Gli algoritmi vengono descritti tramite programmi, che si avvalgono di istruzioni e costrutti dei linguaggi di programmazione per essere eseguiti da calcolatori elettronici

I programmi sono formulazioni concrete di algoritmi astratti che si basano su particolari rappresentazioni dei dati, e utilizzano operazioni di manipolazione dei dati, messe a disposizione da uno specifico linguaggio di programmazione

Le proprietà degli algoritmi sono talmente fondamentali, generali e robuste, da essere indipendenti dalle caratteristiche di specifici linguaggi di programmazione o di particolari calcolatori elettronici

Strutture Dati

Il concetto di algoritmo è inscindibile da quello di **dato**: per risolvere un problema computazionale, occorre organizzare ed elaborare dati

Un algoritmo può essere visto come un manipolatore di dati: a fronte di dati in ingresso che descrivono il problema producono dati in uscita come risultato del problema

È fondamentale che i dati siano ben organizzati e **strutturati** in modo che il calcolatore li possa elaborare efficientemente

"Clever" e "Efficient"

Obiettivo:

Studiare i modi più appropriati di organizzare i dati di un problema al fine di realizzare un algoritmo efficiente

Domanda:

- Che cosa intendiamo per appropriato "clever"?
- Che cosa intendiamo per efficiente "efficient"?

"Clever" e "Efficient"

Data Structure	*Algorithms*
Liste, pile, code	Insert
Heaps	Delete
Alberi binari di ricerca	Find
B-trees	Merge
Tabelle Hash	Shortest path
Grafi	

Analisi di Algoritmi

Analizzare un algoritmo vuol dire determinare le risorse necessarie all'algoritmo in termini di

spazio di memoria
(quantità di memoria utilizzata durante l'esecuzione)
e
tempo computazionale
(tempo di esecuzione)

Analisi di Algoritmi

L'analisi della complessità di un algoritmo in termini di tempo di esecuzione consente di:

- stimare il tempo impiegato
- stimare il più grande input gestibile in termini ragionevoli
- confrontare l'efficienza di due algoritmi diversi
- ottimizzare le parti "critiche"

Definiamo una funzione T : **dimensione** $\longrightarrow$ **tempo** impiegato

Dimensione dell'input

Per molti problemi (ex. l'ordinamento) la misura più naturale è il numero di elementi (criteri di costo uniforme)

Per altri (ex. moltiplicazione di numeri interi) la misura migliore è il numero totale di bit necessari per la rappresentazione dell'input (criteri di costo logaritmico)

In realtà ciascun elemento è rappresentato da un numero costante di bit, quindi le due misure coincidono a meno di una costante moltiplicativa

In altri casi ancora, è più appropriato descrivere la dimensione con due numeri; ex: se l'input è una matrice bidimensionale la dimensione dell'input è $\#righe \times \#colonne$

Modello di Calcolo

Prima di analizzare un algoritmo abbiamo bisogno di stabilire quale sarà la tecnologia di riferimento utilizzata per eseguire gli algoritmi quando saranno realizzati come programmi

Assumiamo di utilizzare

Mono-Processore + RAM (Random Access Memory)

assenza di concorrenza e parallelismo

Definizione di tempo

Tempo = "wall-clock" time ossia il tempo effettivamente impiegato per eseguire un algoritmo

Dipende da troppi fattori (non sempre prevedibili)

1. bravura del programmatore
2. linguaggio di programmazione utilizzato
3. processore, memoria (cache, primaria, secondaria)
4. sistema operativo, processi attualmente in esecuzione

Dobbiamo usare un modello astratto: introduciamo un concetto di tempo legato al "# di operazioni elementari" o di passi eseguiti per il calcolo dell'output corrispondente

Tempo di esecuzione

Numero di operazioni elementari o "passi" eseguiti per il calcolo dell'output

passo $\cong$ una linea di pseudo-codice

Hp: ogni passo riferito ad una linea i, è eseguito in un tempo costante c_i

Parte II

Il problema dell'ordinamento

Il problema dell'ordinamento

Definition

Dato un insieme di n numeri $\langle a_1, a_2, \ldots, a_n \rangle$, trovare un'opportuna permutazione $\langle a'_1, a'_2, \ldots, a'_n \rangle$ tale che $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Input: $\langle a_1, a_2, \ldots, a_n \rangle$

Output: $\langle a'_1, a'_2, \ldots, a'_n \rangle$ oppure

$\langle a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)} \rangle$

dove π è un'opportuna permutazione degli indici $1, \ldots, n$

Istanza del problema

Input: (31, 41, 59, 26, 41, 58)

Output: (26, 31, 41, 41, 58, 59)

La scelta del migliore algoritmo dipende:

- dal numero di elementi da ordinare
- da quanto gli elementi siano già ordinati
- dal dispositivo di memoria (metodo d'accesso)

Idea per ordinare

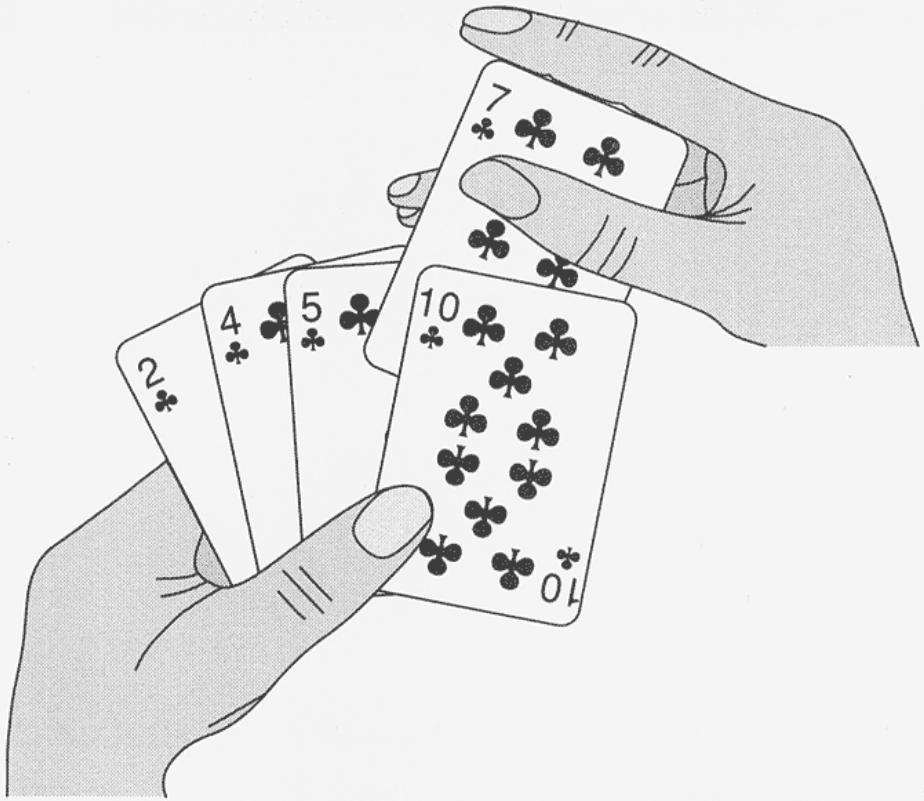
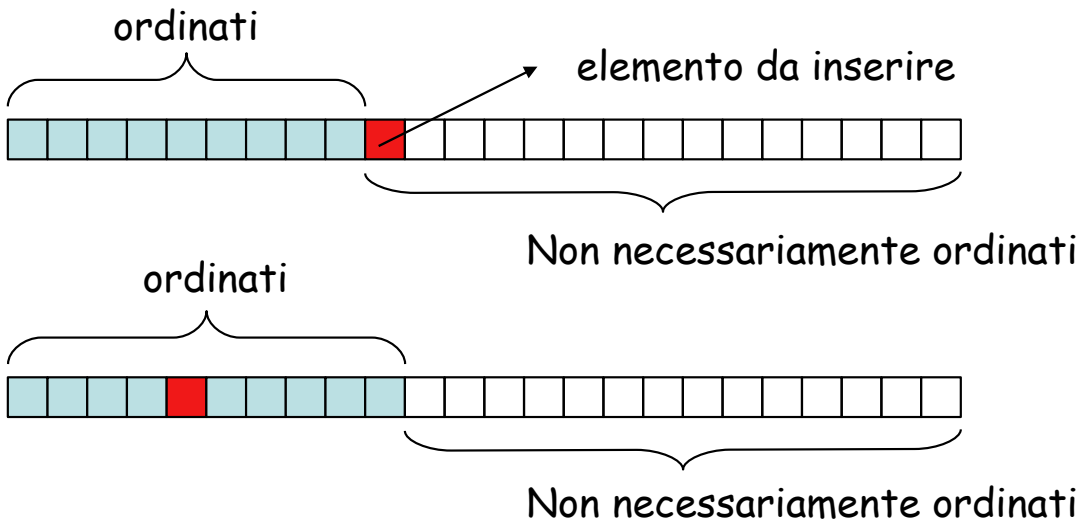


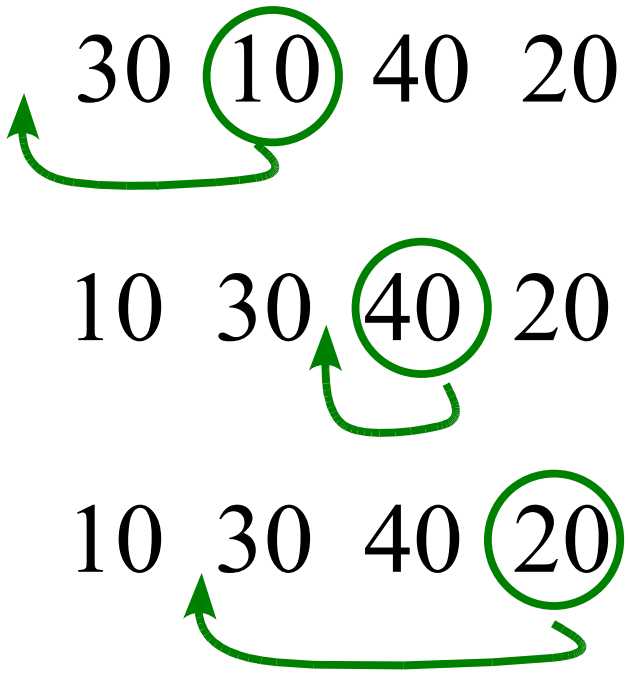
Figure 2.1 Sorting a hand of cards using insertion sort.

Idea per ordinare

Ad ogni passo si ha una sottosequenza ordinata in cui inserisco un nuovo elemento dell'input:



Idea per ordinare



Insertion Sort

L'algoritmo di ordinamento Insertion Sort risulta efficiente nel caso in cui il numero di elementi (n) da ordinare è piccolo

```
for j ← 2 to lenght[A]
    do key ← A[j]
        ▷ Si inserisce A[j] nella sequenza ordinata A[1..j-1]
        i ← j-1
        while i > 0 e A[i] > key
            do   A[i+1] ← A[i]
                 i ← i-1
        A[i+1] ← key
```

Istanza $A = \{5, 2, 4, 6, 1, 3\}$

$j = 2$	5	2	4	6	1	3
$j = 3$	2	5	4	6	1	3
$j = 4$	2	4	5	6	1	3
$j = 5$	2	4	5	6	1	3
$j = 6$	1	2	4	5	6	3
$j = 7$	1	2	3	4	5	6