Introduzione agli Algoritmi

Prof. Emanuela Fachini

- Contenuto del corso
- Motivazioni
- Qualche definizione iniziale

Pagina del corso: http://twiki.di.uniroma1.it/twiki/view/Intro_algo/AD/WebHome

Cosa si studierà?

Classiche strutture dati e algoritmi elementari

Illustreremo alcune soluzioni algoritmiche efficienti per problemi di base come
l'ordinamento di elementi memorizzati in un vettore (array)
la gestione di dizionari, cioè insiemi con le operazioni di ricerca, inserimento e cancellazione.

Gli algoritmi presentati saranno analizzati dal punto di vista della correttezza e dell'uso delle risorse di tempo e spazio di memoria necessarie per l'esecuzione.
Queste analisi esemplificano i metodi di analisi proposti e quindi contribuiscono alla loro comprensione.

Perchè ci interessano?

Gli algoritmi e delle strutture dati che studieremo in questo corso sono ben noti e implementati.

Ma è opportuno conoscerli bene perchè

- **possono essere utilizzati come moduli in algoritmi più complessi**
- **sono abbastanza semplici da essere utilizzati per introdurre importanti metodi e tecniche di**
  - ✓ **analisi degli algoritmi**
- **Il loro studio è un allenamento mentale al problem solving: l'arte di costruire nuove soluzioni algoritmiche per problemi**

Il materiale di studio

T.H. Cormen, C.E. Leiserson, R.L. Rivest e C. Stein , Introduction to algorithms 3/ed, MIT Press, 2009.

Risorse in rete:

Giancarlo Bongiovanni e Tiziana Calamoneri, dispense per il corso di Informatica generale:

http://twiki.dsi.uniroma1.it/twiki/view/Infogen/DispenseELibriDiTesto

Le implementazioni (e non solo)

Risorse in rete(inglese):

per le implementazioni in python, e non solo, il sito del testo "Problem solving with algorithms and data structures using Python" di B.N. Miller e D.L. Ranum:

http://interactivepython.org/runestone/static/pythonds/index.html

o anche

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/

per le implementazioni in java, non solo, il sito del testo "Algoritmi in Java" di R. Sedgewick:

http://algs4.cs.princeton.edu/home/

Gli esami

5 appelli: giugno, luglio, settembre, gennaio e febbraio. Sempre nella prima e quarta settimana dopo la fine delle lezioni

Modalità: l'esame consiste in una prova scritta ed un'eventuale prova orale.

Prova intermedia: è prevista a metà semestre e riguarderà gli argomenti sviluppati fino a quel momento. Chi supera la prova intermedia all'esame farà solo la seconda parte della prova scritta.

Bonus: mezzo punto in più a chi conclude l'esame nella sessione estiva e un altro mezzo punto a coloro che hanno anche superato la prova intermedia.

Risolvere un problema

Definizione del problema

Progetto dell'algoritmo che lo risolve

Analisi dell'algoritmo (correttezza, complessità di tempo e di spazio)

Corsi di Algoritmi

Implementazione in un linguaggio di programmazione

Testing

[manutenzione]

Corsi di Programmazione

Definizione di un problema

Un problema è ben definito quando è chiaro che cosa ci aspettiamo in output in funzione dell' input.

Un'istanza di un problema è un esempio di input.

Esempio di problema ben definito:

Problema 1: Si vuole calcolare il massimo comun divisore tra due numeri interi

Input: due numeri interi

Istanza: 27,18

Output: il MCD dei due numeri in input

Il problema dell'ordinamento

Definizione del problema dell'ordinamento: data una lista di n elementi $A = \langle a_1,a_2,...,a_n \rangle$ presi da un insieme totalmente ordinato si tratta di produrre una lista ordinata degli n elementi, secondo uno dei possibili versi (crescente o descrescente) dell'ordinamento, cioè una permutazione $\langle b_1,b_2,\ldots,b_n \rangle$ dei dati di $\langle a_1,a_2,...,a_n \rangle$ tale che $b_1 \leq b_2 \leq \ldots, \leq b_n$

Problema dell'ordinamento di una sequenza di numeri naturali secondo l'ordinamento naturale.

Istanza del problema:
una sequenza di numeri A = <5,8,7,6,10,43,67>

Output: A = <5,6,7,8,10,43,67> (crescente)
A = <67,43,10,8,7,6,5> (decrescente)

Altri esempi per l'ordinamento

Problema dell'ordinamento di una sequenza di parole
secondo l'ordinamento alfabetico (lessicografico)
Istanza del problema: una sequenza di parole
B = <pesche, mele, pere, ciliege,albicocche>
Output: B = <albicocche,ciliegie,mele,pere,pesche, uva>
(crescente)

Problema dell'ordinamento di una sequenza di parole
secondo l'ordinamento quasi-lessicografico cioè ordinate prima per lunghezza e a parità di lunghezza in alfabetico)
Istanza del problema: una sequenza di parole
B = <pesche, mele, pere, ciliege,albicocche>,
Output: B = < uva,mele,pere,pesche,ciliegie, albicocche>
(crescente)

Progetto dell'algoritmo

Cos'è un algoritmo?

Definizione informale:
un algoritmo è una sequenza finita di istruzioni, ciascuna eseguibile in tempo finito, tale da produrre un determinato risultato

Esempio:

algoritmo di Euclide (~300 a.c.) per il calcolo del MCD
INPUT: due numeri interi *a* e *b*
precondizione: *a* e *b* ≠ 0
Finché *a* ≠ *b*
se *a* > *b* allora ad *a* assegna *a* - *b*
se *a* < *b* allora a *b* assegna *b* - *a*
OUTPUT *b (qui a=b)*

Descrizione di un algoritmo

1. **Può essere descritto in linguaggio naturale (p.e. italiano), utilizzando uno pseudocodice o dei diagrammi,…**
2. **L'input può mancare ma non l' output!**
3. **Ogni istruzione deve essere precisa e chiara**
4. **L'algoritmo deve terminare dopo un numero finito di passi**
5. **Ogni istruzione che lo compone deve essere abbastanza elementare e eseguibile in tempo finito**

Esempio di non algoritmo:

1. **Impara il cinese**
2. **Traduci in cinese**

Da dove viene la parola?

La traduzione latina, risalente al XII secolo, di un libro di **Muhammad ibn Mūsa 'l-Khwārizmī** era intitolata "l-Khwārizmī sui numeri indiani", che fu interpretato come "algoritmi sui numeri indiani".

Muhammad ibn Mūsa 'l-Khwārizmī è un Matematico persiano, nato nel 780 circa e morto nel 850 , famoso per il libro "Kit āb al-djabr wa 'l-muq ābala " il primo libro che tratta in modo sistematico le equazioni lineari e di secondo grado e dal cui titolo è nato anche il termine algebra (al-djabr)

Analisi di un algoritmo: correttezza

Correttezza:

un algoritmo è (totalmente) corretto quando termina su tutti gli input previsti ed è corretto, cioè produce l'output atteso.

tecniche di prova: (in questo corso)

- **induzione**

Analisi di un algoritmo: complessità

Tra due algoritmi che risolvono correttamente lo stesso problema, con quale criterio decido quale usare?

La prima cosa da esaminare sono le risorse di calcolo necessarie per ottenere la risposta: si sceglierà quello più conveniente nel contesto.

Ma come valutare questo aspetto?

Analisi asintotica: un esempio

Il problema dell'individuazione dei duplicati.

soluzione lenta	soluzione veloce
Ogni elemento viene confrontato con i seguenti alla ricerca di un duplicato	Gli elementi vengono prima ordinati, poi esaminati in sequenza per cercare coppie di elementi consecutivi uguali
Si eseguono n-1 confonti per il primo elemento, n-2 per il secondo, … 1 per il penultimo, per un totale di n(n-1)/2 confronti	Per ordinare n elementi bastano nlg n confronti.

Basta aumentare la potenza di calcolo?

	calcolatore veloce + soluzione lenta	calcolatore lento + soluzione veloce
	10^9 istruzioni al secondo	10^7 istruzioni al secondo
vettore di n = 10^6 interi	tempo1 = n. tot. di confronti da eseguire/ n confronti al secondo = $(10^6(10^6-1)/2)/10^9$ cioè circa 500 secondi pari a 8,3 minuti	tempo2 = n. tot. di confronti da eseguire/ n confronti al secondo = $(10^6 lg\ 10^6 + 10^6)/10^7$ cioè meno di 2 secondi
vettore è di n = 10^7 interi	tempo1 = n. tot. di confronti da eseguire/ n confronti al secondo = $(10^7(10^7-1)/2)/10^9$ cioè circa 50.000 secondi poco meno di 14 ore	tempo2 = n. tot. di confronti da eseguire/ n confronti al secondo = $(10^7 lg 10^7 + 10^7)/10^7$ cioè poco meno di 24 secondi

Algoritmi e programmi

	programmi	algoritmi
scritti	linguaggio di programmazione	pseudocodice
correttezza	spesso solo testing	prova
eseguiti	calcolatore	modello di calcolo
efficienza in tempo	profiler	analisi della complessità asintotica

Analisi dell'algoritmo

Si veda http://interactivepython.org/runestone/static/pythonds/AlgorithmAnalysis/WhatIsAlgorithmAnalysis.html per esempi di uso di istruzioni python di profiling.

Modello di calcolo

Un modello di calcolo è definito precisando un modello architetturale e quali operazioni sono permesse per costruire un algoritmo e con quale costo.

Il costo è in termini di tempo di calcolo, ma anche di spazio di memoria.

Un modello serve per semplificare e astrarre dalle specificità di un'architettura di calcolatore.

Modello RAM

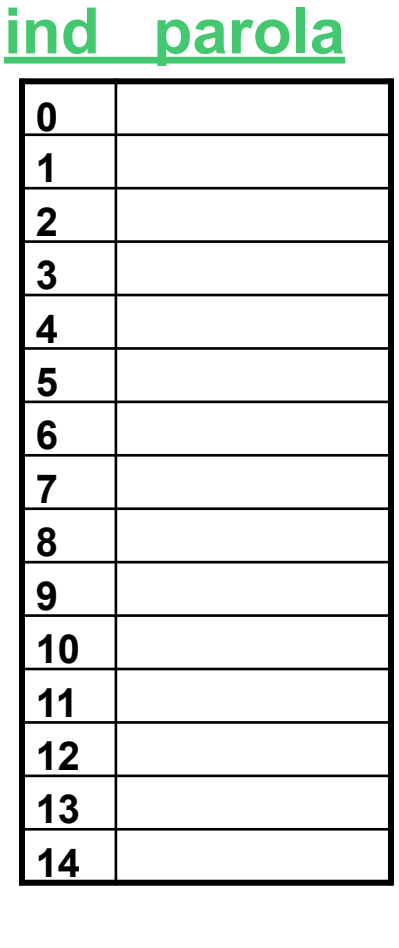
modello RAM

RAM = Random Access Machine.

In una RAM la memoria centrale è

una Random Access Memory cioè un array di celle di memoria, ciascuna accessibile via l'indirizzo e in grado di contenere una word (*parola*), cioè una sequenza di un numero finito di bit (nella realtà 32 o 64).

Anche l'indirizzo è una word. Quindi se la parola è di 32 bits si possono indirizzare 2^{32} (circa quattro miliardi) parole di memoria. Infatti per rappresentare un numero n occorrono lg n bits.

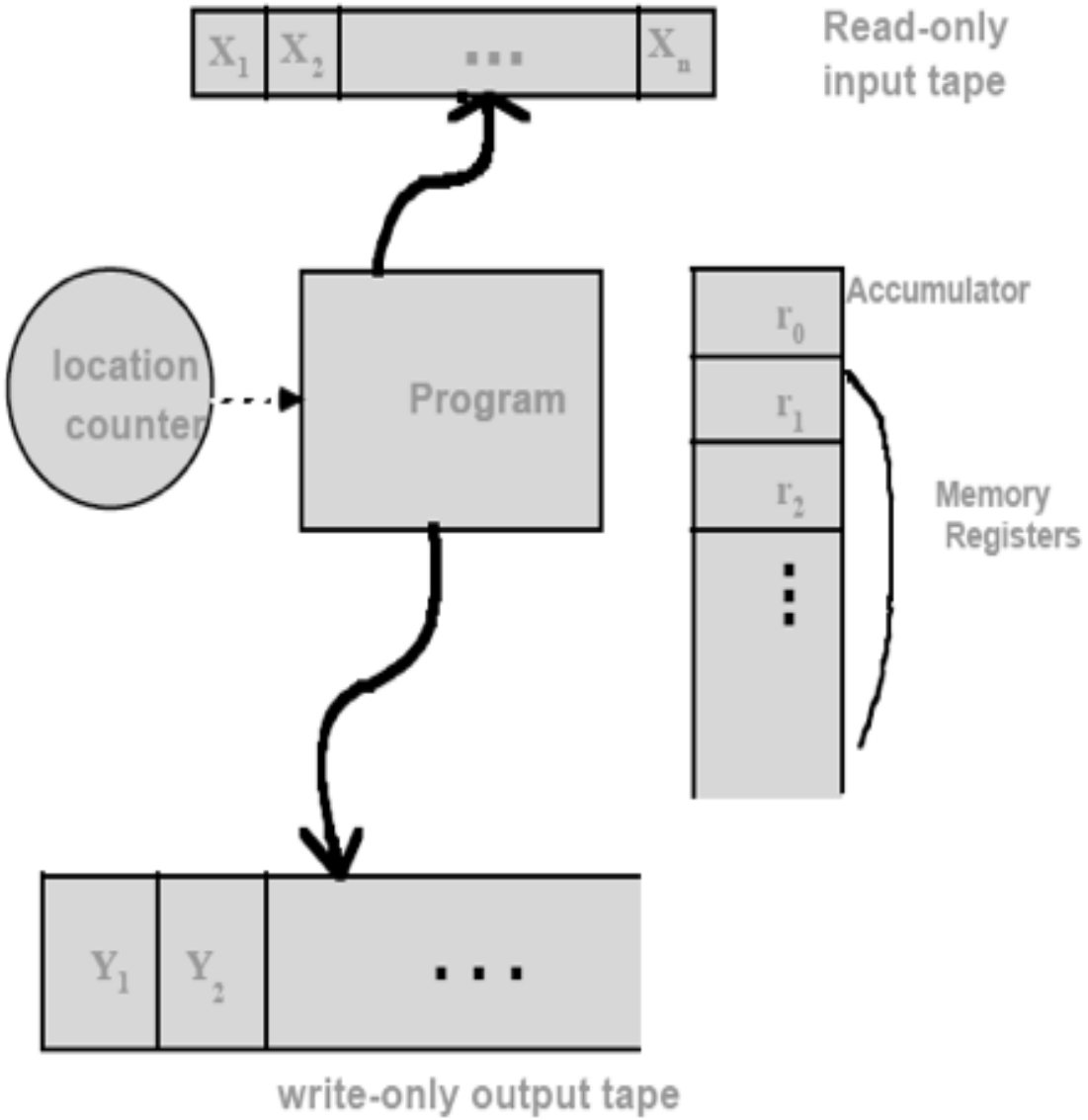


RAM: le operazioni

La RAM dispone di un unico processore e di un numero finito di registri. Ogni registro può memorizzare una parola e su di essi sono possibili le operazioni logico/aritmetiche (come ADD, INC, CMP); i dati in memoria vengono caricati (load) nei registri e da questi i dati vengono memorizzati (store) nella memoria RAM. Inoltre ci sono delle istruzioni per dirigere il flusso di calcolo (per es. JMP).

Tutte queste operazioni sono eseguite in sequenza e in tempo costante.

RAM: vista schematica



Random Access Memory

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

Analisi: complessità degli algoritmi

L'analisi della complessità di un algoritmo consiste nel determinare la quantità di tempo (e/o di spazio di memoria) richiesta per la sua esecuzione

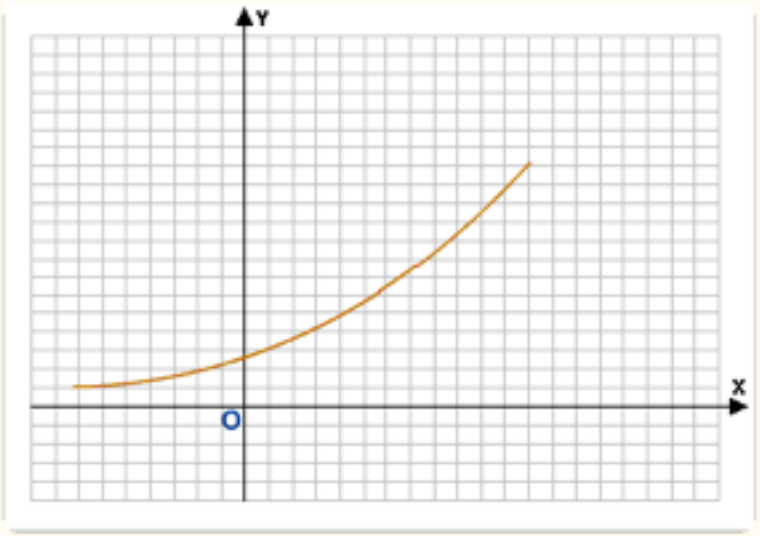
Perchè farlo?
Per poter

1. **predire il comportamento su input di grandezza qualsiasi**
2. **scegliere il giusto algoritmo/struttura dati in un dato contesto confrontando soluzione diverse, dal punto di vista del consumo delle risorse (tempo, spazio)**
3. **decidere se la nostra soluzione è la migliore possibile per un problema**
4. **decidere se il nostro algoritmo è ottimale per il nostro problema (cioè che non è possibile dimostratamente trovarne uno migliore)**

Analisi degli algoritmi: come?

- **Approccio sperimentale**

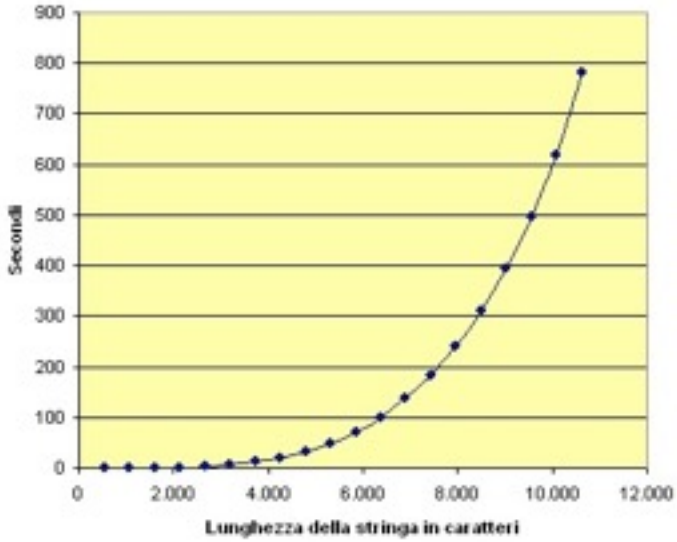
- **Approccio teorico**



Approccio sperimentale

Esperimenti.

Si fa girare un programma che implementa l'algoritmo su input di dimensione crescente e se ne calcolano i tempi di esecuzione. Da questi dati si possono fare previsioni sul comportamento dell'algoritmo su dati di dimensione maggiore che possono essere controllate con nuovi esperimenti.

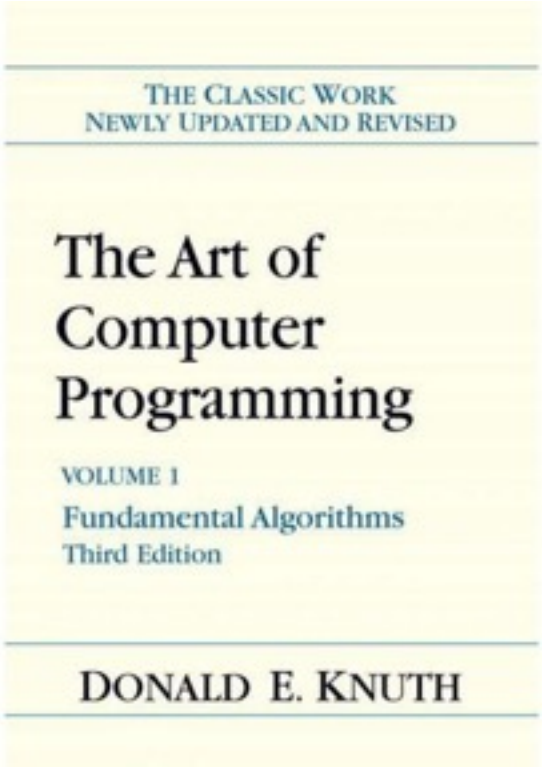


Limiti:

- **Si deve implementare l'algoritmo**
- **Non sempre i risultati sono indicativi del tempo di calcolo su input non compresi nell'esperimento**
- **Il contronto tra algoritmi deve basarsi sullo stesso ambiente hardware/software (processore, tipo e organizzazione memoria, S.O. il compilatore,…)**

Metodo teorico

Tempo totale di calcolo di un algoritmo:
somma dei tempi di ogni operazione eseguita sul modello



Analisi di un algoritmo : si calcola un'approssimazione sul numero di operazioni di costo costante che l'algoritmo esegue.
Si definisce una funzione che associa tale valore approssimato alla dimensione dell'input.

Vantaggi:

- **usa una descrizione ad alto livello dell'algoritmo**
- **dà una risposta per ogni possibile input**
- **permette di valutare l'algoritmo indipendentemente dall'ambiente hard/software**

Ancora su approccio sperimentale

Esperimenti.
Si fa girare un programma che implementa l'algoritmo su input di dimensione crescente e se ne calcolano i tempi di esecuzione. Da questi dati si possono fare previsioni sul comportamento dell'algoritmo su dati di dimensione maggiore che possono essere controllate con nuovi esperimenti.

Meriti:

- **Molti algoritmi recenti sono molto difficili da analizzare teoricamente, lo studio del caso peggiore è troppo "pessimista" e/o i modelli probabilistici non sono abbastanza realistici**
- **La sperimentazione si è rivelata essenziale nella valutazione di euristiche per problemi "difficili", nell'individuazione di istanze dei problemi, nella progettazione di generatori di istanze, nel confronto tra algoritmi che esibiscono grosso modo lo stesso comportamento dal punto di vista teorico**
- **fornisce la chiave per il trasferimento dalla carta alla produzione di codice**

Analisi sperimentale o teorica degli algoritmi? Entrambe!

Recentemente anche gli informatici teorici hanno cominciato ad apprezzare e ad utilizzare un approccio sperimentale all'analisi degli algoritmi (tradizionale in discipline affini come la Ricerca Operativa, ma anche in Intelligenza Artificiale e nel Calcolo Scientifico), soprattutto quando l'analisi teorica porta a risultati troppo "pessimisti" o quando si vuole disporre di dati relativi a input presi dal mondo reale.

Dalla fine del secolo scorso molti studi combinano fruttuosamente i due approcci. Il nuovo campo è stato chiamato Ingegneria degli algoritmi (1997) e ha come obiettivo lo sviluppo di metodi, strumenti e pratiche per valutare e raffinare gli algoritmi attraverso la sperimentazione. L'accento è dunque sul processo che porta da un algoritmo sequenziale scritto su carta a un programma robusto, efficiente, ben testato e di facile uso.

In questo corso si studiano gli strumenti essenziali per l'analisi teorica degli algoritmi.