Introduzione agli Algoritmi

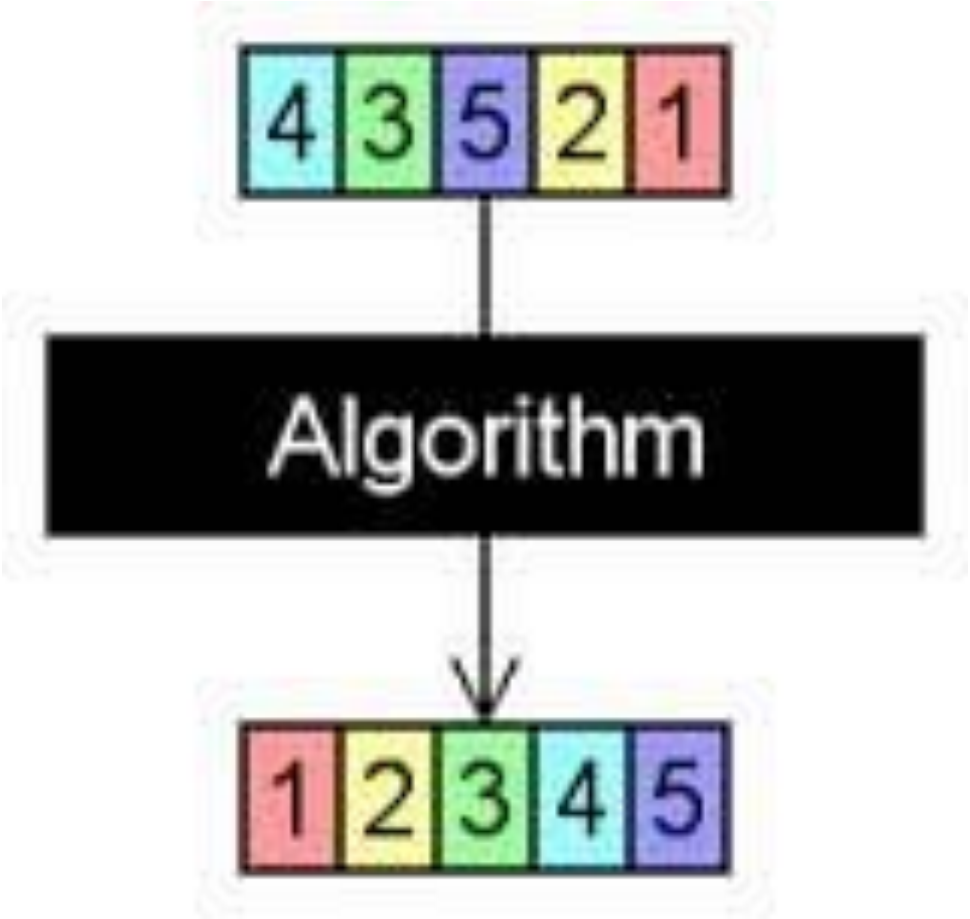
Prof. Emanuela Fachini

- Contenuto del corso
- Motivazioni
- Qualche definizione iniziale

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

Cosa si studierà?

1. Algoritmi per problemi di base come l'ordinamento di elementi memorizzati in una lista (array)

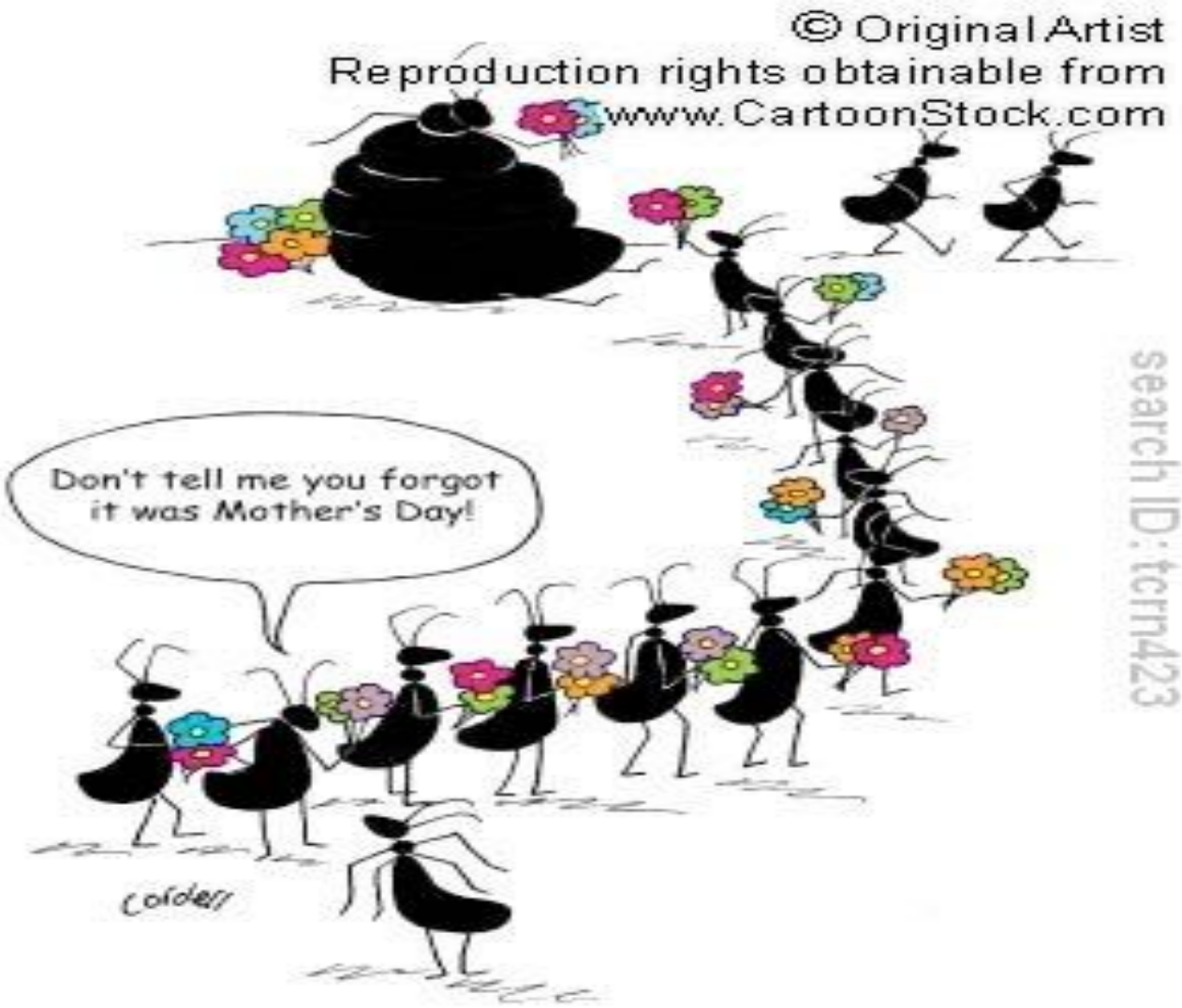


Cosa si studierà?

2. la ricerca di un elemento in un insieme o in una collezione (insieme con ripetizioni),

Cosa si studierà?

3. **la gestione di code**



Cosa si studierà?

Classiche strutture dati e algoritmi elementari

Illustreremo alcune soluzioni algoritmiche efficienti per problemi di base come
l'ordinamento
la gestione di dizionari, cioè insiemi con le operazioni di ricerca, inserimento e cancellazione.
la gestione di code, cioè insiemi con le operazioni di inserimento e cancellazione, dove la cancellazione è determinate dall'ordine di inserimento o da una quantità associata agli elementi.

Cos'è una struttura dati?

Definizione: una struttura dati è un modo di organizzare i dati in memoria

1. arrays

A =

4	18	22	15	9	7	2
0	1	2	3	4	5	6

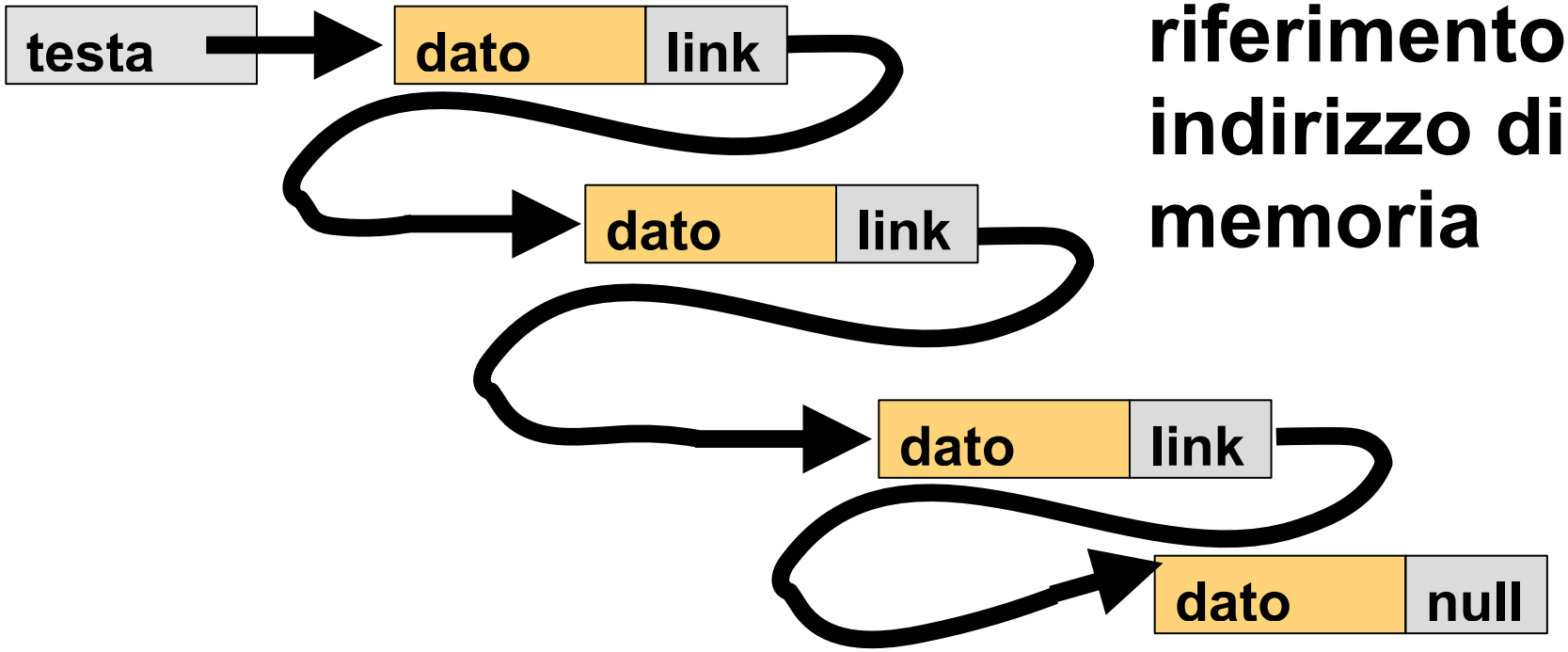
elementi di A

indici di A

L'elemento A[3] è 15, è il quarto elemento nell'array ed è l'elemento di indice 3.
A differenza della lista di Python, un array può contenere solo elementi dello stesso tipo.
La sua dimensione (lunghezza) è il numero degli elementi.

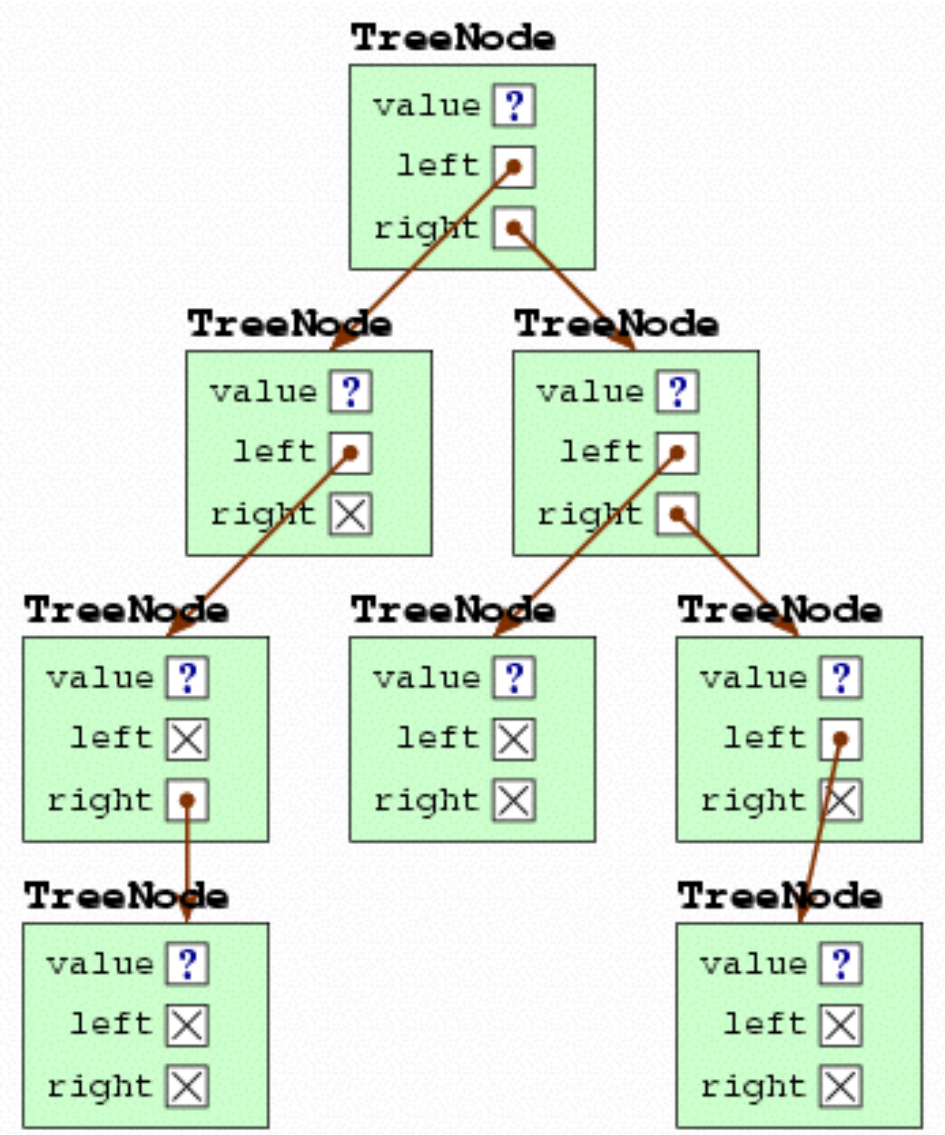
2. Liste concatenate

Definizione: una struttura dati è un modo di organizzare i dati in memoria



link = riferimento, indirizzo di memoria

3. alberi



Tipica rappresentazione in memoria di una struttura gerarchica

Perchè ci interessano?

Gli algoritmi e le strutture dati in generale ci interessano perchè non c'è applicazione informatica che non si basi su un algoritmo e non c'e' algoritmo che non si basi su una struttura dati!

Esempi:

- **trovare informazioni in rete**
- **proteggere le informazioni che sono trasmesse (crittografia)**
- **calcolare il percorso meno costoso per collegare tra loro dei luoghi**
- **assegnare posti ai passeggeri in un aereo, treno, autobus....**
- **...**

Perchè ci interessano?

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

Ma è opportuno conoscerli a fondo perchè

- **possono essere utilizzati come moduli in algoritmi più complessi**

- **sono abbastanza semplici da essere utilizzati per introdurre importanti metodi e tecniche di**
  - ✓ **progettazione e analisi degli algoritmi e**
  - ✓ **dimostrazioni di correttezza**

Perchè ci interessano?

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

Ma è opportuno conoscerli a fondo perchè

- **Il loro studio è un allenamento mentale al problem solving: l'arte di costruire nuove soluzioni algoritmiche per i problemi**

Oltre agli algoritmi

Metodi di analisi della complessità di tempo degli algoritmi

Introdurremo il metodo teorico (asintotico) per l' analisi delle prestazioni di un algoritmo e gli strumenti necessari per realizzarla.

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 il metodo dell' analisi asintotica e ciò ne favorisce la comprensione.

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.

Problem solving

Definizione del problema

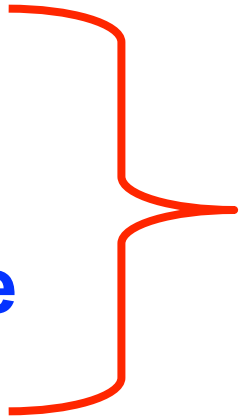
Progetto dell'algoritmo che lo risolve

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

Implementazione in un linguaggio di programmazione

Testing

[manutenzione]



Corsi di Programmazione

Definizione di problema e istanze

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

Per specificare il problema dell'ordinamento bisogna stabilire esattamente su quali dati si vuole operare e con che tipo di ordinamento.

In generale il problema si specifica così:
data una lista di n elementi $A = <a_1,a_2,...,a_n>$ 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 $<b_1,b_2,\ldots,b_n>$ dei dati di $<a_1,a_2,...,a_n>$ 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 (crescente) alfabetico (lessicografico).
Istanza del problema: una sequenza di parole
B = <pesche, mele, pere, ciliege,albicocche>
Output: B = <albicocche,ciliegie,mele,pere,pesche, uva>

Problema dell'ordinamento di una sequenza di parole secondo l'ordinamento (crescente) 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>

Definizione di 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. **Ogni istruzione deve essere precisa e chiara**
3. **L'algoritmo deve terminare dopo un numero finito di passi**
4. **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".

Il matematico persiano, Muhammad ibn Mūsa 'l-Khwārizmī, nato nel 780 circa e morto nel 850, è 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: tempo e spazio

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 della complessità di tempo

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

Vorremmo farlo in modo tale da 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. **destinare ulteriori risorse spazio temporali ad altri requisiti desiderabili del software, aspetti come facilità d'uso (interfacce, etc.) o sicurezza, …**

Tempo di calcolo

Tempo totale di calcolo di un algoritmo è la somma dei tempi di esecuzione di ogni operazione eseguita.

Ma il tempo di esecuzione di un'operazione dipende dalla piattaforma su cui si fa girare l'algoritmo (il programma che lo implementa)

Tempo di calcolo

Tempo totale di calcolo di un algoritmo:
somma dei tempi di esecuzione di ogni operazione eseguita.

1° astrazione/semplificazione: ogni operazione elementare (operazione aritmetica, assegnamento, un confronto, etc.) è considerata di costo costante

Il tempo di calcolo di un algoritmo:
La somma dei tempi di esecuzione delle operazioni di costo costante che l'algoritmo esegue.

Esempio 1

```
if score >= 90:
    print('A')
else:
    if score >=80:
        print('B')
```

Tempo di calcolo di un algoritmo:
La somma dei tempi di esecuzione delle operazioni di costo costante che l'algoritmo esegue.

tempo di calcolo:
il primo if prende un tempo costante, diciamo c
il secondo ancora una costante, diciamo c',
print prende tempo costante b,
mentre il confronto ha un tempo costante d
allora il tempo totale è al più 2d+c+c'+b.
Questo dice che questo frammento di programma ha un tempo di esecuzione costante.

Esempio 2

```
1. wordlist = ['cat','dog','rabbit']
2. letterlist = [ ]
3. for aword in wordlist:
4.     for aletter in aword:
5.         letterlist.append(aletter)
6. print(letterlist)
```

tempo di calcolo:
Le linee 1,2 e 6 sono eseguite in tempo costante (a,b e c).
La linea 5 è eseguita in tempo costante, d.

Il ciclo for della linea 3 è eseguito tante volte quante sono le parole in wordlist, nell'esempio 3 volte.
Ma se vogliamo una previsione su una lista qualunque?
Allora chiamiamo n il numero delle parole nella lista,
il secondo ciclo (linea 4) è eseguito tante volte quant'è la lunghezza della parola scelta nella lista.
Se supponiamo che la più lunga parola nella lista è lunga m, possiamo dire che il tempo di esecuzione su un input formato da una lista di n parole, di lunghezza al più m, è minore o uguale a a+b+c + n*m*d.

Tempo di calcolo

Questi esempi mostrano che la sola astrazione sui tempi reali di esecuzione su una piattaforma non basta a dotarci di strumenti idonei all'analisi del tempo di calcolo.
Nell'ultimo esempio troviamo un limite superiore ma con un'abbondanza di dettagli inutili.
Dire che il tempo è minore o uguale a a+b+c + n*m*d o dire che è minore o uguale a p + n*m*d (dove p è una nuova costante) dà la stessa informazione!
Bisogna dotarsi di altri strumenti per esprimere in modo più efficace la complessità di tempo, che già qui vediamo espressa come funzione di due misure dell'input.

Tempo di calcolo di un algoritmo:
è espresso da una funzione che ha come argomento una o più misure dell'input e come valore la somma dei tempi di esecuzione delle operazioni di costo costante che l'algoritmo esegue per input di quella dimensione.

Esempio 3

Il problema dell'individuazione dei duplicati.

Specifichiamo il problema:

input: una lista di interi e

output: 1 se nella lista c'è almeno un elemento che si ripete e 0 altrimenti.

Qui contiamo solamente il numero di esecuzioni dell' operazione più costosa: il confronto tra elementi.

Esempio 3

Contiamo il numero di confronti necessari per ottenere il risultato

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. Si scorre poi la lista alla ricerca di due elementi consecutivi uguali.

Confrontiamo le soluzioni

	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

tempo di calcolo

In questi esempi abbiamo visto che contare il numero delle operazioni più costose eseguite per un certo input ci consente di fare previsioni e scelte su quale algoritmo usare e che considerare singolarmente il tempo costante di ogni operazione appesantisce il calcolo senza dare reali vantaggi.

Dobbiamo adottare una strategia generale e un modello teorico in modo da ottenere i vantaggi che abbiamo visto nel misurare solo i confronti ma anche che ci consenta di non trascurare completamente le costanti che danno conto dei tempi di esecuzione delle singole operazioni.

Per chiarire definitamente su che tipo di macchina immaginiamo di eseguire i nostri algoritmi, introduciamo il nostro modello di calcolo.

Algoritmi e programmi

	programmi	algoritmi
scritti	linguaggio di programmazione	pseudocodice
correttezza	spesso solo testing	prova
eseguiti	calcolatore	modello di calcolo
efficienza in tempo	calcolo del tempo di esecuzione, 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, 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

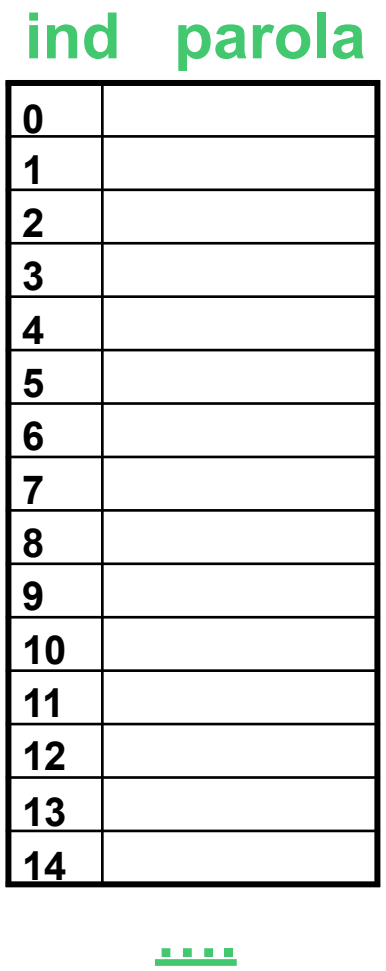
modello RAM

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.

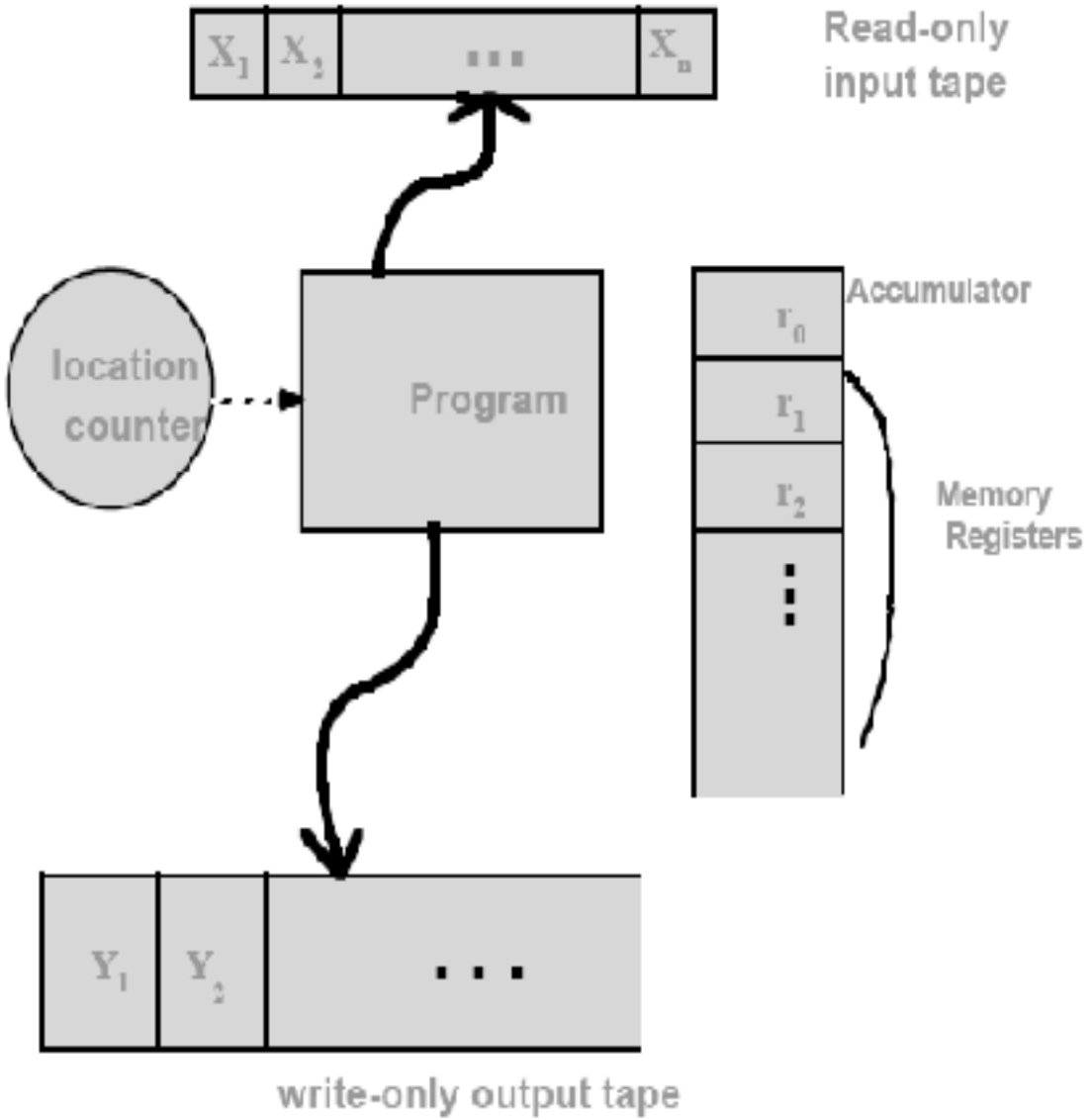


RAM: le operazioni

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.

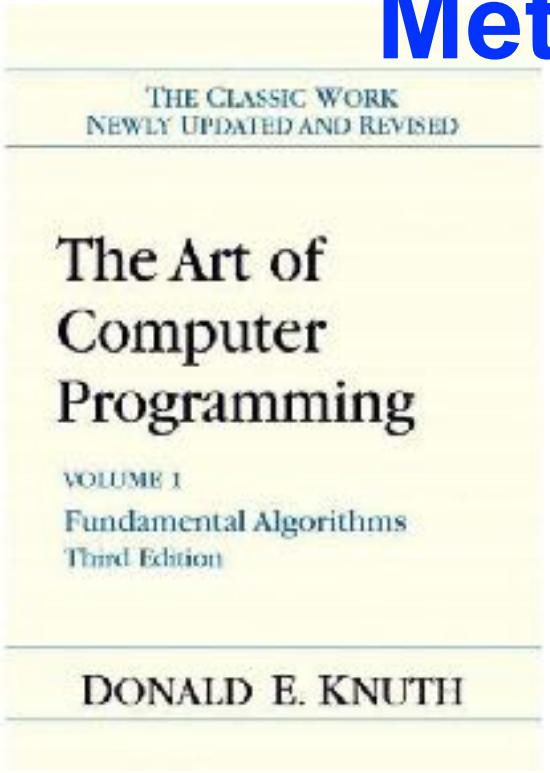
RAM: vista schematica



Random Access Memory

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

Metodo teorico



Si definisce una funzione che associa alla dimensione dell'input (per esempio numero di elementi in una lista, numero di nodi o altezza in un albero)

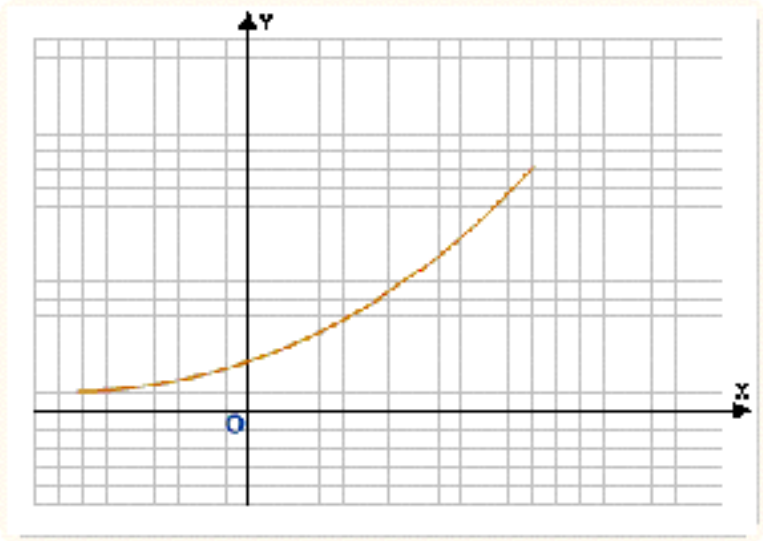
il (un limite superiore al) tempo di calcolo, calcolato sommando i tempi costanti delle operazioni eseguite per un input di quella dimensione.

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**

Analisi degli algoritmi: alternative

- **Approccio teorico**

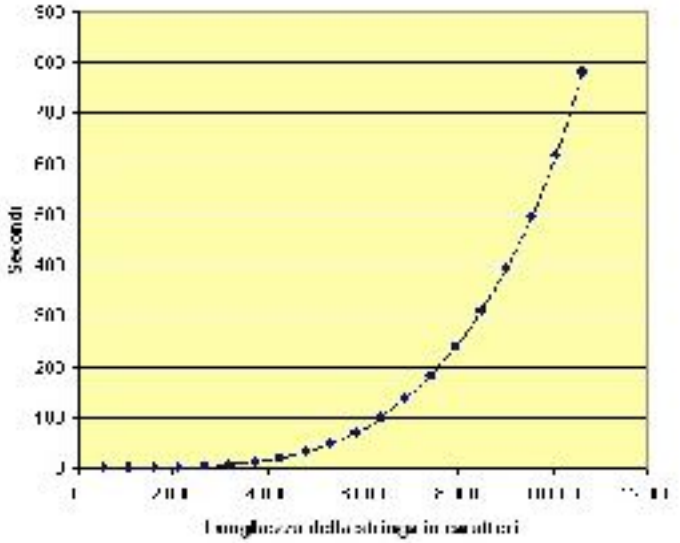


- **Approccio sperimentale**

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,…)**

Meriti dell' approccio sperimentale

Meriti:

- **Molti algoritmi sono difficili da analizzare teoricamente, il limite superiore calcolato 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, 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), o quando l'analisi teorica porta a risultati troppo "pessimisti" o quando si vuole confrontare soluzioni sostanzialmente equivalenti dal punto di vista dell'analisi asintotica.

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.