Algoritmi e Strutture Dati
Introduzione al corso

Lorenzo Donatiello
Dip. di Informatica – Scienza e Ingegneria-
Università di Bologna

lorenzo.donatiello@unibo.it,

Presentiamoci

Titolare del corso: prof. Lorenzo Donatiello

Modulo A (I sem.) Prof. Lorenzo Donatiello

lorenzo.donatiello@unibo.it

http//www.cs.unibo.it/~donat/

Lezioni primo semestre

mercoledì ore 09.00 –12.00

venerdì ore 10.00 – 13.00

Presentiamoci

- Ricevimento Studenti

- Previo appuntamento da concordare via mail

Informazioni generali sul corso

Obiettivi del corso

Il corso ha lo scopo di introdurre le principali strutture dati e presentare i concetti fondamentali riguardanti la progettazione e realizzazione di algoritmi e l'analisi della loro correttezza ed efficienza.

Prerequisiti

Programmazione (Programmazione Internet + Lab. Di Programmazione)

Nozioni di base di Analisi Matematica e Algebra (sommatorie, disequazioni, polinomi, elementi di calcolo delle probabilità)

Programma del Corso

Algoritmi di ordinamento e ricerca;
Strutture dati elementari: liste, pile, code;
Strutture dati non lineari: alberi e grafi;
Tecniche per l'analisi di algoritmi;
Strutture dati avanzate: alberi di ricerca, tabelle hash, heap;
Tecniche Algoritmiche: divide et impera, metodo *greedy,* programmazione dinamica, backtrack, ricerca locale;
Macchine di Turing e Calcolabilità

Sperimentazione

Implementazione in linguaggio Java di algoritmi analizzati durante le lezioni.

Testo adottato

- Alan Bertossi, Alberto Montresor, *Algoritmi e strutture di dati 2/ed*, Città Studi, ISBN: 9788825173567

Testi consigliati

- Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano, *Algoritmi e strutture dati* 2/ed, McGraw-Hill ISBN: 9788838664687, Giugno 2008

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduzione agli algoritmi e strutture dati 2/ed*, McGraw-Hill, ISBN: 9788838662515, Maggio 2005

Modalità d'esame

L’esame finale prevede lo svolgimento di un progetto e una prova orale.

Progetto:

vengono proposti problemi la cui soluzione richiede la progettazione di algoritmi da implementare in linguaggio Java.

l’efficienza degli algoritmi proposti rappresenterà uno dei criteri principali su sui si baserà la valutazione;

le specifiche del progetto verranno pubblicate 4/5 settimane prima della data stabilita per il primo appello di ogni sessione (estiva/autunnale/estiva) di esame;

una volta decisa la sessione in cui si vuol sostenere l’esame, Il progetto va consegnato entro la data stabilita, riportata nelle specifiche;

Modalità d'esame

Se la valutazione del progetto è positiva, **bisogna** sostenere l'esame in uno degli appelli della sessione;

In caso di valutazione negativa, ci si organizza per l'appello successivo.

Orale

è possibile sostenere l'orale in una sessione solo se si è consegnato il progetto per la relativa sessione (estiva / autunnale / invernale)

L'orale includerà discussione del progetto consegnato (oltre che domande su intero programma)

Un suggerimento

- Nei lucidi troverete una serie di domande che vengono lasciate a voi per esercizio.
  - In alcuni casi la risposta si trova nel libro di testo; in generale dovreste essere in grado di rispondere da soli.
  - Provate a rispondere
- TUTTO il materiale viene inserito anche nella pagina del corso: http://www.cs.unibo.it/~donat/

altri suggerimenti

- Seguite le lezioni;
- Studiate gli argomenti man mano che vengono presentati;
- Esercitatevi il più possibile.

- Descriveremo gli algoritmi mediante pseudocodice
  - Per poterne dare una versione compatta senza perdersi in dettagli implementativi
  - Cercate di implementare alcuni degli algoritmi che vedremo a lezione. Niente più della pratica aiuta a fissare i concetti

FAQ

- È sufficiente studiare sui lucidi?
  - I lucidi sono fatti per integrare lo studio individuale, e soprattutto lo studio sul libro di testo.

FAQ

.

- Queste slide non ci sono sul sito del corso /
  Queste slide sono diverse sul sito del corso
  - Cerco di mettere a disposizione le slide aggiornate prima della lezione, ma non sempre ci riesco
  - In ogni caso metto sempre i lucidi online dopo la lezione
  - Talvolta trovo (o mi vengono segnalati) errori in lucidi di lezioni passate. Controllate spesso la pagina del corso per aggiornamenti

Algoritmi e strutture dati

Cos'è un algoritmo?

- Un algoritmo è un procedimento effettivo per risolvere un problema mediante una sequenza finita di passi elementari
- Il procedimento deve essere descritto in modo preciso allo scopo di poterne automatizzare l'esecuzione
- L'esecuzione deve terminare dopo un tempo finito
- l'output deve essere univoco

Cos'è un algoritmo?

- Il termine deriva dal nome del matematico persiano Abu Ja'far Muhammad ibn Musa Khwarizmi
-
  - Autore di un primo fondamentale trattato di algebra
  - Un cratere lunare porta il suo nome

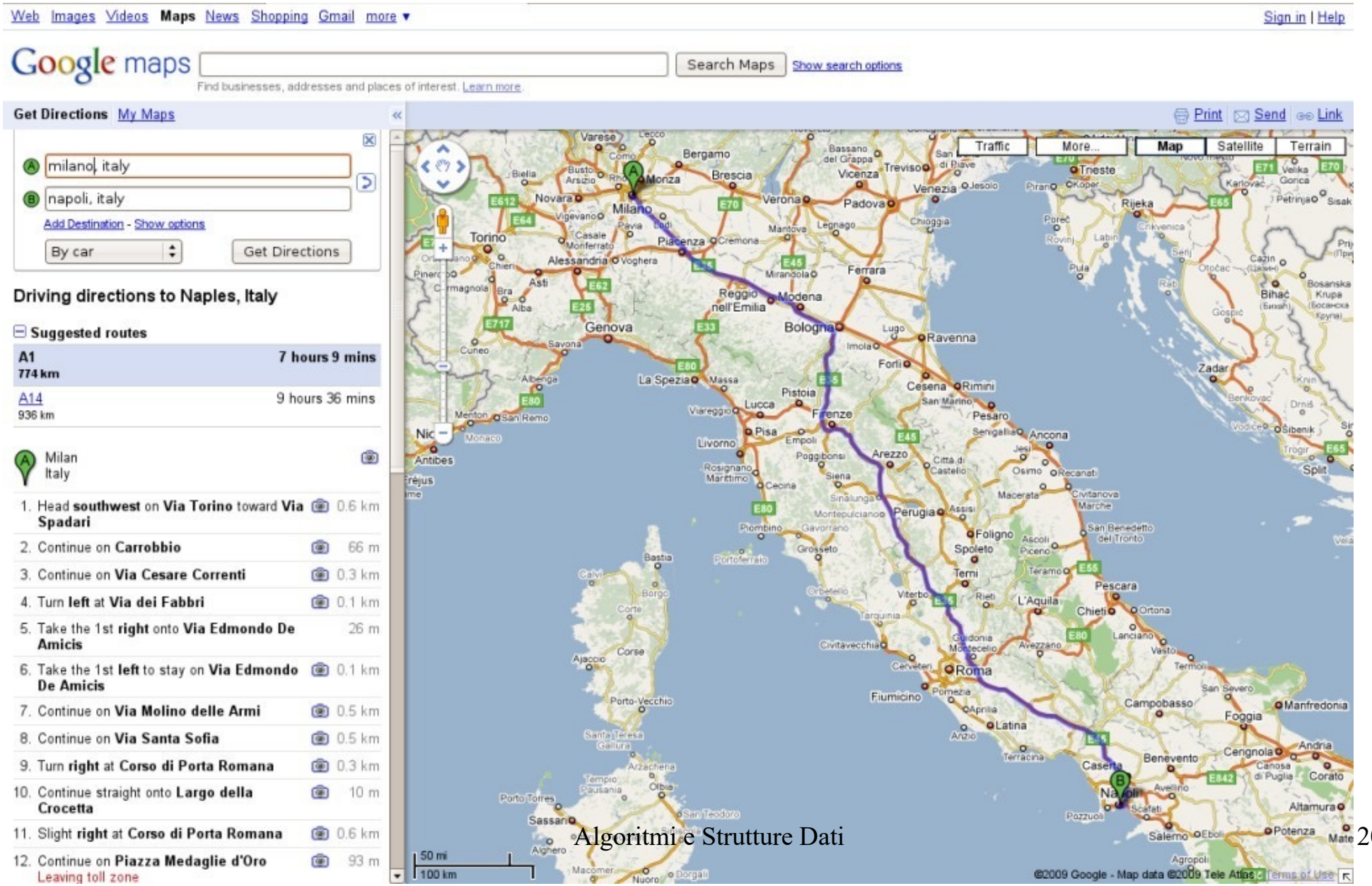
Algoritmo vs Programma

- Un algoritmo descrive (ad alto livello) una procedura di calcolo che, se seguita, consente di ottenere un certo risultato
- Un programma è l'implementazione di un algoritmo mediante un opportuno linguaggio di programmazione.
  - Un programma può essere direttamente eseguito da un calcolatore (processo in esecuzione); un algoritmo solitamente no.

Gli algoritmi sono ovunque!

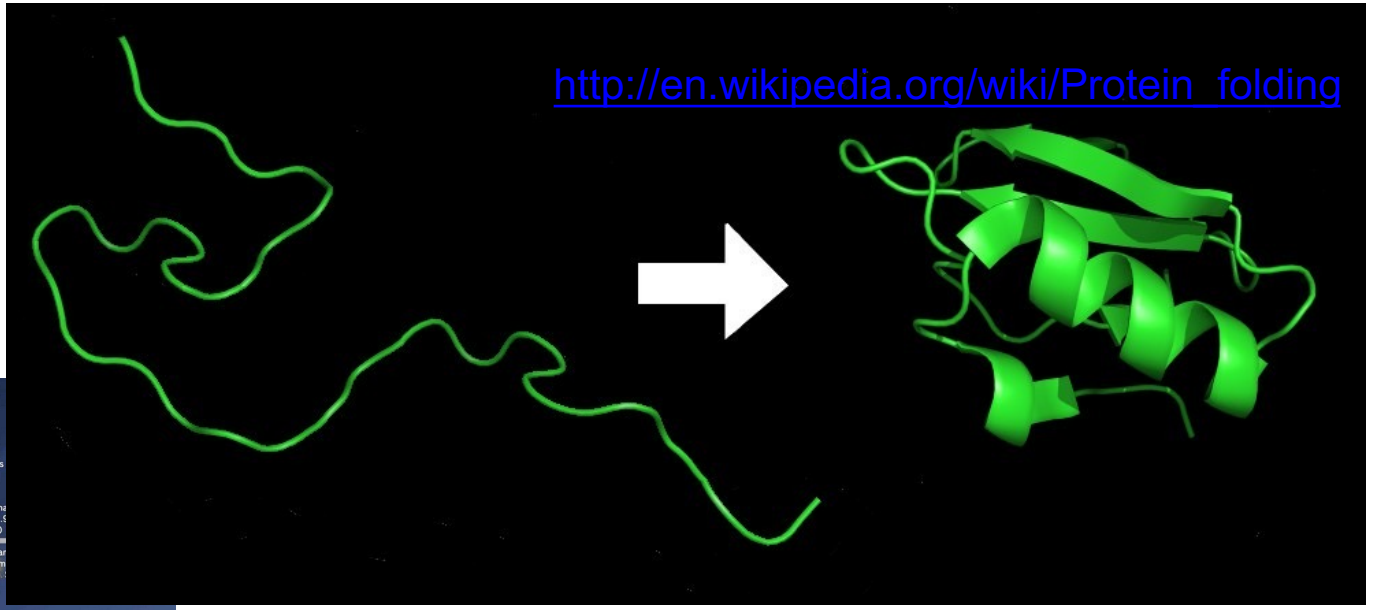
- Internet. Web search, packet routing, distributed file sharing.
- Artificial Intelligence. Machine Learning, Deep Learning...
- Biology. Human genome project, protein folding.
- Computers. Circuit layout, file system, compilers.
- Computer graphics. Movies, video games, virtual reality.
- Security. Cell phones, e-commerce, voting machines.
- Multimedia. CD player, DVD, MP3, JPG, DivX, HDTV.
- Transportation. Airline crew scheduling, map routing.
- Physics. N-body simulation, particle collision simulation.
- Social Networks Analysis.

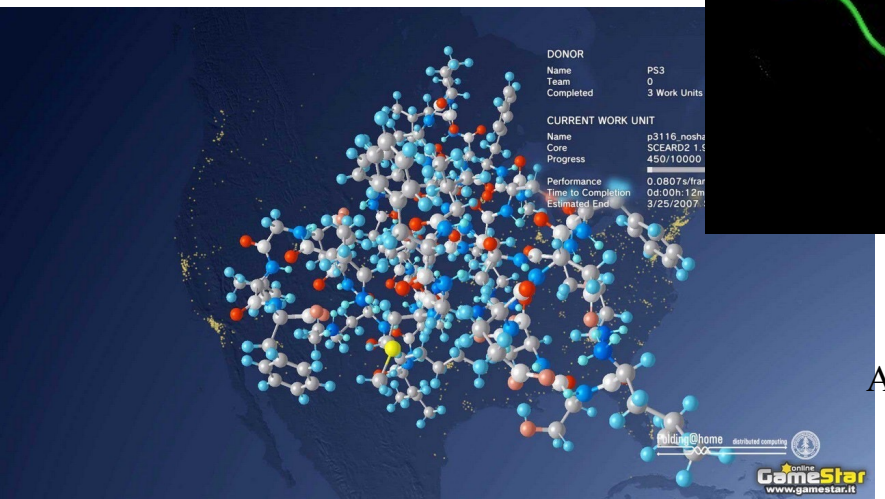
Perché studiare gli algoritmi?



Perché studiare gli algoritmi?

- Le proteine assumono una ben precisa struttura tridimensionale a causa dell'interazione degli aminoacidi che le compongono
- Si ritiene che certe malattie neurodegenerative siano causate dall'accumulo di proteine che si ripiegano in maniera “scorretta”
- Folding@Home





Perché studiare gli algoritmi?

- Eliminazione superfici nascoste, strutture dati per codificare l'ambiente di gioco, simulazioni fisiche (collisioni, movimento dei tessuti, sistemi di particelle—fuoco, nebbia, acqua...)

Perché studiare gli algoritmi?

- Sfida intellettuale

- Per profitto
  - Un algoritmo efficiente può fare la differenza tra il poter risolvere un problema e non poterlo risolvere

Cosa imparerete in questo corso?

- Quali sono gli algoritmi “classici” per risolvere problemi ricorrenti
  - Ordinamento, ricerca, problemi su grafi...
- Come valutare l'efficienza di un algoritmo in termini di prestazioni ed utilizzo di risorse
- Come sviluppare nuovi algoritmi
- Limiti

Valutazione degli algoritmi: altre proprierà

-
-

Valutazione degli algoritmi: altre proprierà

- Correttezza, Semplicità, Modularità, Manutenibilità,
- Espandibilità, Sicurezza, Robustezza, User- Friendly..

- Trede-off tra prestazioni e modularità

- Trade-off tra prestazioni e ….

Come descrivere un algoritmo

- **E' necessario utilizzare una descrizione il più possibile formale**
- **Indipendente dal linguaggio: "*Pseudo-codice*"**
- **Particolare attenzione va dedicata al livello di dettaglio**
  - **Da una ricetta di canederli (google:canederli ricetta), leggo: "... amalgamate il tutto e fate riposare un quarto d'ora..."**
  - **Cosa significa "amalgamare"? Cosa significa "far riposare"?**
  - **E perché non c'è scritto più semplicemente "prepara i canederli"?**

Problema computazionale: esempi

- **Minimo**
  - **Il minimo di un insieme *A* è l'elemento di *A* che è minore o uguale ad ogni elemento di *A***

$$min(A) = a \Leftrightarrow \forall b \in A : a \leq b$$

- **Ricerca**
  - **Sia $A=a_1,\ldots,a_n$ una sequenza di dati ordinati e distinti, $a_1 < a_2 < \cdots < a_n$. Eseguire una ricerca della posizione di un dato *v* in *A* consiste nel restituire l'indice corrispondente, se *v* è presente, oppure 0, se *v* non è presente**

$$ricerca(A, v) = \begin{cases} i & \exists i \in \{1, \ldots, n\} : a_i = v \\ 0 & altrimenti \end{cases}$$

Algoritmo: esempio

- **Minimo**
  - **Per trovare il minimo di un insieme, confronta ogni elemento con tutti gli altri; l'elemento che è minore di tutti è il minimo.**
- **Ricerca**
  - **Per trovare un valore *v* nella sequenza *A*, confronta *v* con tutti gli elementi di *A*, in ordine, e restituisci la posizione corrispondente; restituisci 0 se nessuno degli elementi corrisponde.**

Problemi

Le descrizioni precedenti presentano diversi problemi:

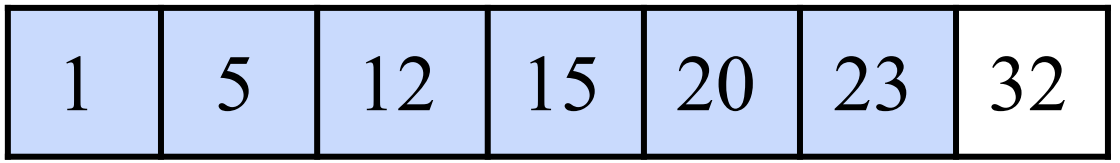
- **Descrizione**
  - **Descritti in linguaggio naturale, imprecisi**
  - **Abbiamo bisogno di un linguaggio più formale**
- **Valutazione**
  - **Esistono algoritmi "migliori" di quelli proposti?**
  - **Dobbiamo definire il concetto di migliore**
- **Progettazione**
  - **Questi problemi sono semplici**
  - **Problemi più complessi devono essere affrontati con opportune tecniche di programmazione**

Esempio: ricerca del minimo in un vettore

```
ITEM min(ITEM[ ] A, integer n)
    ITEM min ← A[1]                                  % Minimo parziale
    for integer i ← 2 to n do
        if A[i] < min then
            min ← A[i]                               % Nuovo minimo parziale
    return min
```

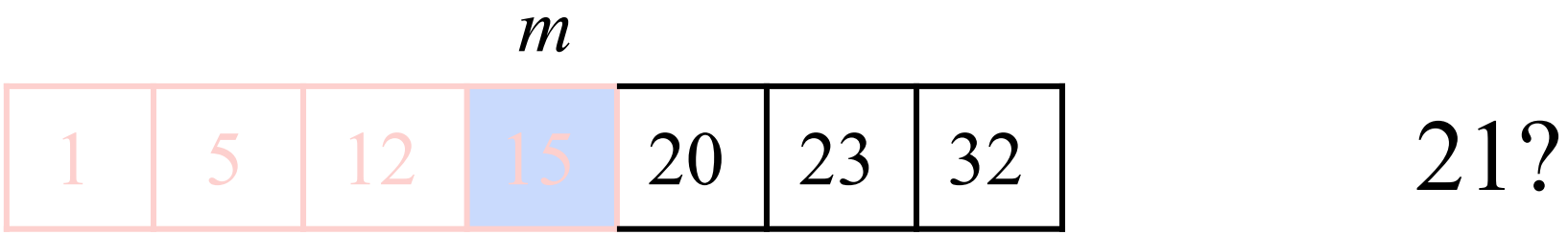
Ricerca in un array ordinato

- **Problema**
  - **Dato un vettore *A* contenente *n* elementi, verificare se un certo elemento *v* è presente**
  - **Esempi: elenco del telefono, dizionario**
- **Una soluzione "banale"**
  - **Scorro gli elementi in ordine, finché non trovo un oggetto "maggiore o uguale" a *v***



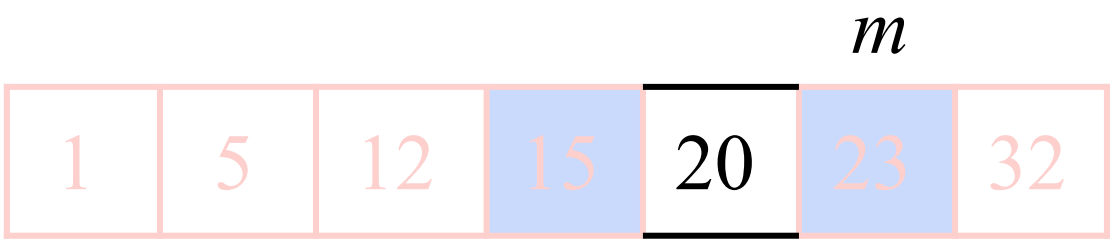
Ricerca in un array ordinato

- **Una soluzione efficiente**
  - **Considero l'elemento centrale (indice *m*) del vettore**
    - **Se *A[m]* = *v*, ho finito**
    - **Se *v* < *A[m]*, cerco nella "metà di sinistra"**
    - **Se *A[m]* < *v*, cerco nella "metà di destra"**



Ricerca in un array ordinato

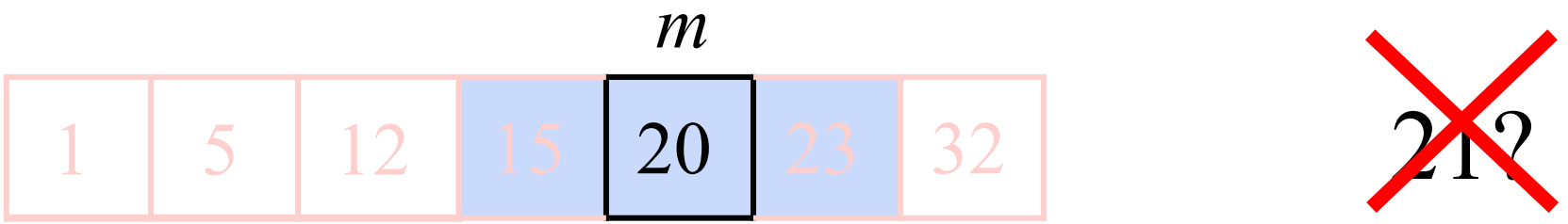
- **Una soluzione efficiente**
  - **Considero l'elemento centrale (indice *m*) del sottovettore che sto analizzando:**
    - **Se *A[m]=v*, ho finito**
    - **Se *v<A[m]*, cerco nella "metà di sinistra"**
    - **Se *A[m]<v*, cerco nella "metà di destra"**



21?

Ricerca in un array ordinato

- **Una soluzione efficiente**
  - **Considero l'elemento centrale (indice *m*) del sottovettore che sto analizzando:**
    - **Se *A[m]=v*, ho finito**
    - **Se *v<A[m]*, cerco nella "metà di sinistra"**
    - **Se *A[m]<v*, cerco nella "metà di destra"**



Ricerca in un array ordinato

```
ITEM binarySearch(ITEM[] A, ITEM v, integer i, integer j)
if i > j then
    return 0
else
    integer m ← ⌊(i + j)/2⌋
    if A[m] = v then
        return m
    else if A[m] < v then
        return binarySearch(A, v, m + 1, j)
    else
        return binarySearch(A, v, i, m − 1)
```

Pseudo-codice

- $a \leftarrow b$
- $a \leftrightarrow b \quad \equiv \quad temp \leftarrow a; a \leftarrow b; b \leftarrow temp$
- **if** *condizione* **then** istruzione
- **if** *condizione* **then** istruzione1 **else** istruzione2
- **while** *condizione* **do** istruzione
- **for** *indice* $\leftarrow$ *estremoInferiore* **to** *estremoSuperiore* **do** istruzione
- **for** *indice* $\leftarrow$ *estremoSuperiore* **downto** *estremoInferiore* **do** istruzione

indice $\leftarrow$ *estremoInferiore*
while *indice* $\leq$ *estremoSuperiore* **do**
  istruzione
  indice $\leftarrow$ *indice* $+ 1$

indice $\leftarrow$ *estremoSuperiore*
while *indice* $\geq$ *estremoInferiore* **do**
  istruzione
  indice $\leftarrow$ *indice* $- 1$

- iif$(condizione, v_1, v_2)$
- **foreach** *elemento* $\in$ *insieme* **do** istruzione
- *% commento*
- **integer**, **real**, **boolean**
- **and**, **or**, **not**
- $+, -, \cdot, /, \lfloor x \rfloor, \lceil x \rceil, \log, x^2, \ldots$

Pseudo-codice

- **Tipi di dato composto**
  - **Vettori, matrici**
  - **Record**
  - **Puntatori**

RETTANGOLO
integer *lunghezza*
integer *altezza*

$T[\,]\ A \leftarrow$ **new** $T[1 \ldots n]$
$T[\,][\,]\ B \leftarrow$ **new** $T[1 \ldots n][1 \ldots m]$
RETTANGOLO $r \leftarrow$ **new** RETTANGOLO
$r.altezza \leftarrow 10$
delete r
$r \leftarrow$ **nil**

- **Procedure e funzioni**
  - Parametri formali
  - Parametri attuali

$\min(V, 1024)$
$\min(E, h)$

- **return**
- **precondition**

Problemi

Le descrizioni precedenti presentano diversi problemi:

- **Descrizione**
  - **Descritti in linguaggio naturale, imprecisi**
  - **Abbiamo bisogno di un linguaggio più formale**
- **Valutazione**
  - **Esistono algoritmi "migliori" di quelli proposti?**
  - **Dobbiamo definire il concetto di migliore**
- **Progettazione**
  - **Questi problemi sono semplici**
  - **Problemi più complessi devono essere affrontati con oppurtune tecniche di programmazione**

Valutazione algoritmi

- **Risolve correttamente il problema?**
  - **Dimostrazione matematica, descrizione "informale"**
  - **Nota: Alcuni problemi non possono essere risolti**
  - **Nota: Alcuni problemi vengono risolti in modo approssimato**
- **Risolve il problema in modo efficiente?**
  - **Definizione di "efficienza"?**
  - **Alcuni problemi non possono essere risolti in modo efficiente**
  - **Esistono soluzioni "ottime": non è possibile essere più efficienti**
- **Quali altre proprietà entrano in gioco?**
  - **Semplicità, modularità, manutenibilità, espandibilità, sicurezza e robustezza**

Valutazione degli algoritmi

Complessità di un algoritmo

Analisi delle risorse impiegate da un algoritmo per risolvere un problema, in funzione della *dimensione* e dal *tipo* dell'input

Risorse

***Tempo*: tempo impiegato per completare l'algoritmo**

***Spazio*: quantità di memoria utilizzata**

***Banda*: quantità di bit spediti**

Valutazione degli algoritmi

Ricerca del minimo in un vettore;

Ricerca all'interno di un vettore di un valore *v*;

Quanti confronti?

(n-1)

(n)

Perché contare i confronti?

Valutazione degli algoritmi: array ordinato

Contiamo il numero di confronti:

Ad ogni confronto dimezziamo lo spazio di ricerca:

$$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{2^2} \rightarrow \frac{n}{2^3} \ldots\ldots \rightarrow \ldots\ldots \frac{n}{2^k}$$

Complessità in tempo

Complessità in tempo: cosa serve?

Per stimare il tempo impiegato da un programma

Per stimare il più grande input gestibile in tempi ragionevoli

Per confrontare l'efficienza di algoritmi diversi

Per ottimizzare le parti più importanti

“Complessità”: “Dimensione” → “Tempo”

Dobbiamo definire “dimensione” e “tempo”!

Valutazione degli algoritmi: array ordinato

Assumiamo:

$$n = 2^k$$

Nel caso peggiore (l'elemento non c'è) l'algoritmo termina dopo k +1 confronti (considerare anche il caso n =1, ovvero k=0)

Avremo k = log n (il logaritmo è in base 2)

Numero di confronti: log (n) + 1

Complessità in tempo

- **Ogni istruzione richiede un tempo costante per essere eseguita**
- **Costante diversa da istruzione a istruzione**
- **Ogni istruzione viene eseguita un certo # di volte, dipendente da *n***

Complessità in tempo

- Tempo di calcolo di min()

integer min(ITEM[] A, **integer** n)	Costo	# Volte
ITEM $min \leftarrow A[1]$	c_1	1
for integer $i \leftarrow 2$ **to** n **do**	c_2	n
if $A[i] < min$ **then**	c_3	$n-1$
$min \leftarrow A[i]$	c_4	$n-1$
return min	c_5	1

Complessità in tempo

Complessità in tempo: cosa serve?

Per stimare il tempo impiegato da un programma

Per stimare il più grande input gestibile in tempi ragionevoli

Per confrontare l'efficienza di algoritmi diversi

Per ottimizzare le parti più importanti

"Complessità": "Dimensione" → "Tempo"

Dobbiamo definire "dimensione" e "tempo"!

Complessità in tempo

Tempo = "# operazioni elementari"

Quali operazioni possono essere considerate elementari?

Esempio: min(A, n)

Modello di calcolo: rappresentazione astratta di un calcolatore

***Astrazione*: deve semplificare dettagli, altrimenti è inutile**

***Realismo*: deve riflettere la situazione reale**

"*Potenza*" matematica: deve permettere di trarre conclusioni "formali" sul costo

Complessità in tempo

Tempo = "Wall-clock" time:

Il tempo effettivamente impiegato per eseguire un algoritmo

Dipende da troppi parametri:

bravura del programmatore

linguaggio di programmazione utilizzato

codice generato dal compilatore

processore, memoria (cache, primaria, secondaria)

sistema operativo, processi attualmente in esecuzione

Dobbiamo considerare un modello astratto

Complessità in tempo

Modello RAM

Random Access Machine (RAM)

***Memoria*:**

Quantità infinita di celle di dimensione finita

Accesso in tempo costante (indipendente dalla posizione)

***Processore* (singolo)**

Set di *istruzioni* elementari simile a quelli reali:
somme, addizioni, moltiplicazioni, operazioni logiche, etc.
istruzioni di controllo (salti, salti condizionati)

***Costo* delle istruzioni elementari**

Uniforme, ininfluente ai fini della valutazione (come vedremo)

Complessità in tempo

Tempo di calcolo di min()

- **Ogni istruzione richiede un tempo costante per essere eseguita**
- **Costante diversa da istruzione a istruzione**
- **Ogni istruzione viene eseguita un certo # di volte, dipendente da *n***

Complessità in tempo

- Tempo di calcolo di min()

integer min(ITEM[] A, **integer** n)	Costo	# Volte
ITEM $min \leftarrow A[1]$	c_1	1
for integer $i \leftarrow 2$ **to** n **do**	c_2	n
if $A[i] < min$ **then**	c_3	$n-1$
$min \leftarrow A[i]$	c_4	$n-1$
return min	c_5	1

Complessità in tempo

- Tempo di calcolo di min()

$$
\begin{aligned}
T(n) &= c_1 + c_2 n + c_3(n-1) + c_4(n-1) + c_5 = \\
&= (c_2 + c_3 + c_4)n + (c_1 + c_5 - c_3 - c_4) \\
&= an + b
\end{aligned}
$$

ESERCIZIO

È dato un array A[1..n-1] contenente una permutazione degli interi da 1 a n (estremi inclusi) a cui è stato tolto un elemento; i valori in A possono comparire in un ordine qualsiasi

Es: A = [1, 3, 4, 5] è una permutazione di 1..5 a cui è stato tolto il numero 2

Es: A = [7, 1, 3, 5, 4, 2] è una permutazione di 1..7 a cui è stato tolto il numero 6

Scrivere una procedura che dato l'array A[1..n-1], individua il valore nell'intervallo 1..n che non compare in A.

Stima del tempo di esecuzione

- Misurare il tempo di esecuzione in secondi?
  - Dipende dalla macchina su cui si fanno le misure
- Misurare il numero di istruzioni in linguaggio macchina eseguite?
  - Anche questo dipende dalla macchina su cui si fanno le misure, e in più è una quantità difficile da desumere a partire dallo pseudocodice
- Stimiamo il tempo di esecuzione calcolando il numero di operazioni elementari eseguite nello pseudocodice