

Algoritmi e Strutture Dati

□ Rocco Aversa

- Tel.: 0815010268
- Email: rocco.aversa@unicampania.it
- Ricevimento (on demand):
lunedì, ore 14-16

□ Testo adottato

Algoritmi e strutture dati in Java

di Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser – **Apogeo**

□ Sito corso

<https://elearning.unicampania.it/course/view.php?id=490>

1

1

Obiettivi generali del corso

- Studio di algoritmi e strutture dati fondamentali
- Progetto e implementazione di algoritmi e strutture dati in Java
- Come si misura l'efficienza degli algoritmi e delle strutture dati
- Come scegliere gli algoritmi e le strutture dati adatte a risolvere in modo efficiente un problema

2

2



Cosa studieremo

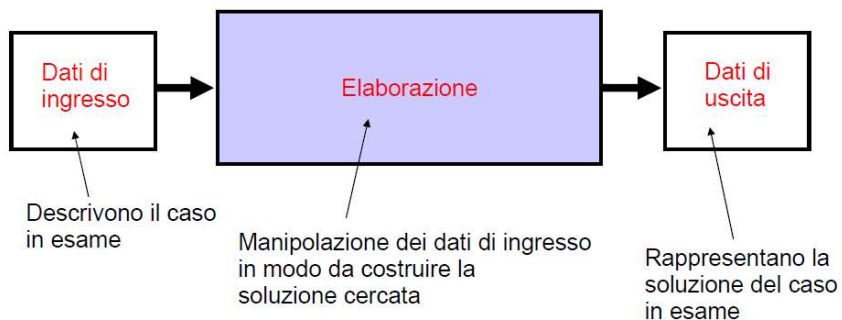
- **Algoritmi** = descrizione precisa di una sequenza di azioni che devono essere eseguite per giungere alla risoluzione di un problema
 - Analisi / progetto / codifica
 - Analisi dell'efficienza
- **Strutture dati** = è fondamentale che i dati siano ben organizzati e strutturati in modo che l'algoritmo li possa elaborare efficientemente



3

3

Risolvere un problema



4

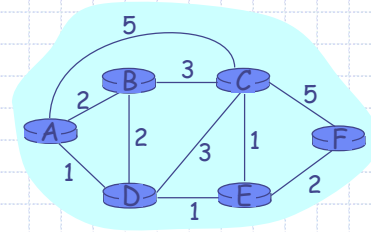
4

Esempio: Routing in Internet

Protocollo di Routing
Obiettivo: determinare "buon"
 cammino sorg.-dest

Astrazione usando grafi:

- ❑ I nodi rappresentano router
- ❑ Gli archi descrivono i link fisici
 - Costi sugli archi (link) : ritardo, costo in €, livello di congestione
- ❑ Cammino "buono":
 - Di solito significa cammino a costo minimo
 - Possibili def. alternative

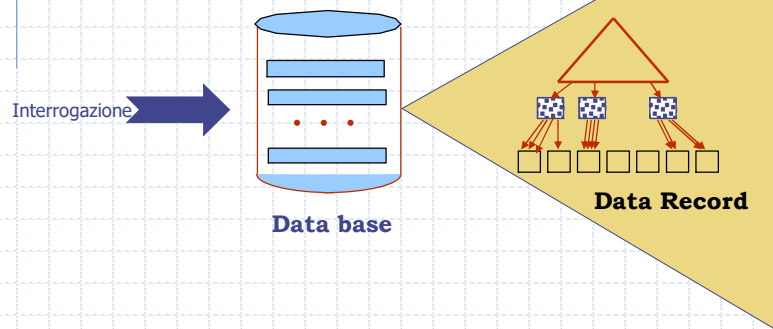


5

5

Esempio: Accesso a basi di dati

Interrogazione
Obiettivo: rispondere
 rapidamente



6

6

Classificare i problems (in base al miglior algor. che li resolve)

Problemi, algoritmi, programmi

- **Problema computazionale** = specifica in termini generali la relazione che deve valere tra input e output
- **Algoritmo** = descrive una procedura computazionale (sequenza di passi) ben definita per trasformare l'input nell'output
- **Programma** = rappresentazione di un algoritmo utilizzando un linguaggio non ambiguo e direttamente comprensibile dal computer

NON ANCORA
ESEGUIBILE

7

⁷ Probl. non polinomiali: non c'è (ancora) una soluz. polinom.

Esempio: problema dell'ordinamento

- **Input:** una sequenza di n numeri
 $\langle a_1, a_2, \dots, a_n \rangle$
- **Output:** una permutazione (riarrangiamento)
 $\langle a'_1, a'_2, \dots, a'_n \rangle$ tale che $a'_1 \leq a'_2 \leq \dots \leq a'_n$
- **Istanza del problema:**
 $\langle 31, 41, 59, 26, 41 \rangle$
- **Soluzione:** $\langle 26, 31, 41, 41, 59 \rangle$

8

Algoritmo: definizione

Un algoritmo è una sequenza **ordinata** di passi elementari **eseguibili** e **non ambigui** che giunge certainemente a **terminazione**

■ Sequenza ordinata

- Un algoritmo deve avere una struttura ben stabilita in termini di ordine di esecuzione dei suoi passi ma i passi non devono necessariamente essere eseguiti secondo una sequenza lineare. Es. algoritmi paralleli

9

9

Algoritmo: requisiti

La descrizione di un procedimento risolutivo di un problema può considerarsi un algoritmo se rispetta alcuni requisiti essenziali:

- **Finitezza:** un algoritmo deve essere composto da una sequenza finita di passi elementari !
- **Eseguibilità:** il potenziale esecutore deve essere in grado di eseguire ogni singola azione in tempo finito con le risorse a disposizione !
- **Non-ambiguità:** l'esecutore deve poter interpretare in modo univoco ogni singola azione !

10

10

Programma, processo, algoritmo

- **Programma** = rappresentazione fisica formale di un algoritmo progettata per essere eseguita da un computer } EE
- **Processo** = l'attività di esecuzione dell'algoritmo rappresentato dal programma

11

11

Rappresentazione degli algoritmi

- Lo stesso algoritmo può essere rappresentato in vari modi
 - formula, sequenza di istruzioni, disegno, a parole...
 - a diversi livelli di astrazione (linguaggio macchina, assembly, linguaggio ad alto livello: Pascal, C, Java)
 - Si può utilizzare un linguaggio astratto ad alto livello o **pseudocodice**:
 - ♦ per evitare dettagli inutili
 - ♦ per sottolineare il fatto che **un algoritmo è completamente indipendente dal linguaggio**
- Ogni rappresentazione si basa su un insieme di **primitive** ben definite, comprensibili all'esecutore

12

12

Pseudocodice programmazione strutturata

- Il teorema di **Jacopini-Böhm** afferma che qualunque algoritmo può essere descritto utilizzando esclusivamente tre strutture di controllo fondamentali:
 - struttura **sequenziale**
 - struttura **condizionale** (o di selezione)
 - struttura **iterativa**

13

13

Struttura sequenziale

Descrizione	Diagramma di flusso	Pseudocodice
Rappresenta la sequenza di azioni elementari direttamente eseguibili una di seguito all'altra	<pre>graph TD; Entry(()) --> Istr1[Istr 1]; Istr1 --> Istr2[Istr 2]; Istr2 --> Exit(())</pre>	<p>..... Istr 1 Istr 2</p> <p>Es. Linguaggio C</p> <p>A=12; B=14; C=A+B;</p>

14

14

Struttura condizionale

Descrizione	Diagramma di flusso	Pseudocodice
Rappresenta la scelta, in base alla risposta di un Test, tra due esecuzioni poste in alternativa	<pre> graph TD Entry(()) --> Test{Test} Test -- NO --> B[B] Test -- SI --> A[A] B --> Merge(()) A --> Merge Merge --> Exit(()) </pre>	<pre> if Test blocco A else blocco B </pre> <p>Es. Linguaggio C</p> <pre> if(Test){ a=1; b=3; } else{ a=2; b=4; } </pre>

15

15

Struttura condizionale

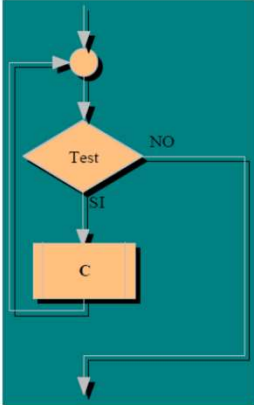
```

if test1
    blocco A
elseif test2
    blocco B
elseif ...
        
```

16

16

Struttura iterativa

Descrizione	Diagramma di flusso	Pseudocodice
Rappresenta la ripetizione di una o più azioni fino a quando la risposta al test rimane affermativa; quando la risposta è negativa l'iterazione termina		<pre>while test blocco C</pre> <p>Es. Linguaggio C</p> <pre>while (test){ a=1; b=2; }</pre>

17

17

Struttura iterativa

```
for i = 0 to A.length
    blocco istr
```

```
for i = A.length downto 0
    blocco istr
```

18

18

Processo di sviluppo di un programma

1. **Specifica funzionale** del problema
2. **Analisi del problema** e definizione di un **algoritmo** risolutivo
3. Descrizione con **diagramma di flusso** e/o **pseudocodice**
4. Traduzione dell'algoritmo in **programma** in linguaggio di programmazione **ad alto livello**
5. **Compilazione**
6. **Esecuzione e Verifica**

19

Specifica funzionale del problema

- **Problema:** calcolare il minimo di un insieme di numeri interi, maggiori o uguali a zero
- **Input:** un insieme di numeri interi
 $\langle a_0, a_1, a_2, \dots, a_{n-1} \rangle$
- **Output:** un numero intero m tale che valga la seguente relazione:

$$\forall 0 \leq i \leq n - 1, m \leq a_i$$

- **Istanza:** $\langle 23, 5, 7, 8, 10, 2, 3 \rangle$
- **Soluzione:** $m=2$

X

20

20

Analisi del problema e definizione di un **algoritmo** risolutivo

- Per trovare l'elemento minimo di un insieme di numeri interi positivi A (che posso implementare come array):
 - Memorizzo in una variabile min il primo elemento dell'insieme
 - Eseguo una scansione dell'insieme A a partire dal secondo elemento e confronto ogni elemento di A con il valore memorizzato nella variabile min
 - Se l'elemento corrente è $< min$, aggiorno la variabile min (cioè assegno a min l'elemento corrente)
 - Alla fine del ciclo di iterazione sull'insieme di input A , nella variabile min sarà memorizzato l'elemento minimo dell'insieme di input

21

21

Rappresentazione algoritmo con pseudocodice

```
Min(A)
  min = A[0]
  for i=1 to A.length
    if A[i] < min
      min = A[i]

  return min
```

22

22

Analisi di algoritmi

Dato un algoritmo A e un problema P dimostrare che A risolve P (correttezza) e valutare la quantità di risorse usate da A (complessità computazionale)

- Un algoritmo è corretto se, per ogni istanza di input, termina con l'output corretto
- Lo studio teorico dell'efficienza (performance) di un programma e dell'uso delle risorse
- Spesso l'efficienza segna il confine tra possibile e impossibile (es. applicazioni real-time)

23

23

Analisi di algoritmi

- Altri aspetti da prendere in considerazione:

modularità	manutenibilità	funzionalità
robustezza	user-friendliness	tempo di
programmazione	semplicità	estendibilità
affidabilità		

24

24

Analisi della complessità degli algoritmi

- Prevedere le risorse richieste dall'algoritmo
 - Analizzeremo
 - **Tempo di calcolo** impiegato da un algoritmo per risolvere un problema
 - **Spazio occupato** durante la computazione (memoria RAM o disco)
- in modo da poter **confrontare algoritmi** diversi e progettare algoritmi efficienti

25

25

A noi interessa comparare algoritmi, non vedere performance effettiva

Analisi di algoritmi: Modello di calcolo

- Modello delle risorse e dei costi dell'uso delle risorse
- **Modello RAM = Random-Access Machine**
 - 1 processore
 - Istruzioni sequenziali
 - Istruzioni aritmetiche (add, sub, mul, div, mod), per spostare dati (load, store), di controllo (salto [in]condizionato, chiamata a subroutine, return) => **costante**
 - Memoria RAM e disco, no cache e memoria virtuale

26

26

Importante è avere la dipendenza da n

Complessità di un algoritmo

- **$T(n)$ = tempo di esecuzione** = numero di operazioni elementari eseguite
- **$S(n)$ = spazio di memoria** = numero di celle di memoria utilizzate durante l'esecuzione
- **n = dimensione dei dati di ingresso**
 - Es. vettore di elementi: n = numero degli elementi
 - Es. grafo: n, m = numero dei vertici, numero archi

27

27

$T(n)$ tempo di elaborazione

- **Caso peggiore:** (spesso)
 $T(n)$ = tempo **massimo** dell'algoritmo su *qualsiasi* input di dimensione n
- **Caso medio:** (talvolta)
 $T(n)$ = tempo **atteso** su tutti gli input di dimensione n = tempo di ogni input \times **la probabilità che ci sia quell'input** (media pesata)
È necessaria un'assunzione sulla distribuzione statistica degli input (spesso *distribuzione uniforme*)
- **Caso migliore:** (fittizio = prob. non si verificherà mai)
Ingannevole per algoritmi lenti che sono veloci su *qualche* input

→ Usiamo lui per analisi

Completato da trovare

28

28

Caso peggiore

- Generalmente si cerca un limite superiore perché:
 - Fornisce una garanzia all'utente
 - ♦ L'algoritmo non potrà impiegare più di così
 - Per alcuni algoritmi si verifica molto spesso
 - ♦ Es. ricerca in un DB di informazione non presente
 - Il caso medio spesso è cattivo quasi quanto quello peggiore
 - ♦ Non sempre è evidente cosa costituisce un input medio

29

29

Tempo di calcolo indipendente dalla macchina

- Qual è il tempo di calcolo di un algoritmo nel caso peggiore?
 - Dipende dal computer usato
 - velocità relativa (confronto sulla stessa macchina)
 - velocità assoluta (su macchine diverse)
- **IDEA:**
 - Ignorare le costanti dipendenti dalla macchina
 - Studiare il tasso di crescita di $T(n)$ con $n \rightarrow \infty$

"Analisi asintotica"

Velocità di divergenza!

30

30

Esempio: $T(n)$ di una funzione iterativa

Min(A)

min = A[0]

for i=1 to A.length

if A[i] < min

min = A[i]

return min

Costo Numero di volte

c_1 1

c_2 n

c_3 n-1

c_4 n-1

$$T(n) = c_1 + n \cdot c_2 + (n-1) \cdot c_3 + (n-1) \cdot c_4 = (c_2 + c_3 + c_4) \cdot n + (c_1 - c_3 - c_4) = \underline{a \cdot n + b}$$

funzione lineare

ultima esec:
i maggiore e
piu esec

c_2 = costo confronto

31

31

Ringraziamenti

Questi lucidi sono un adattamento del materiale preparato dal prof.ssa M. Federico per il corso di Algoritmi e Strutture dati

32

32