Algoritmi e Strutture Dati

Introduzione

Alberto Montresor

Università di Trento

2019/01/13

Sommario

Introduzione

Problema computazionale

Dati un dominio di input e un dominio di output, un *problema computazionale* è rappresentato dalla relazione matematica che associa un elemento del dominio di output ad ogni elemento del dominio di input.

Algoritmo

Dato un problema computazionale, un *algoritmo* è un procedimento effettivo, espresso tramite un insieme di passi elementari ben specificati in un sistema formale di calcolo, che risolve il problema in tempo finito.

Algoritmi nella storia

- Papiro di Rhind o di Ahmes (1850BC): algoritmo del contadino per la moltiplicazione
- Algoritmi di tipo numerico furono studiati da matematici babilonesi ed indiani
- Algoritmi in uso fino a tempi recenti furono studiati dai matematici greci più di 2000 anni fa
  - Algoritmo di Euclide per il massimo comune divisore
  - Algoritmi geometrici (calcolo di tangenti, sezioni di angoli, ...)

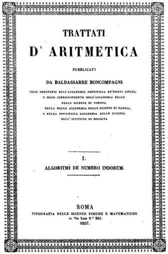
Origine del nome

Abu Abdullah Muhammad bin Musa al-Khwarizmi

- E' stato un matematico, astronomo, astrologo e geografo
- Nato in Uzbekistan, ha lavorato a Baghdad
- Dal suo nome: algoritmo

Algoritmi de numero indorum

- Traduzione latina di un testo arabo ormai perso
- Ha introdotto i numeri indiani (arabi) nel mondo occidentale
- Dal numero arabico sifr $= 0$: zephirum $\rightarrow$ zevero $\rightarrow$ zero, ma anche cifra



TRATTATI

D' ARITMETICA

PUBBLICATI

DA BALDASSARRE BONCOMPAGNI

SOCIO ORDINARIO DELL'ACCADEMIA PONTIFICIA DE'NUOVI LINCEI,
E SOCIO CORRISPONDENTE DELL'ACCADEMIA REALE
DELLE SCIENZE DI TORINO,
DELLA REALE ACCADEMIA DELLE SCIENZE DI NAPOLI,
E DELLA PONTIFICIA ACCADEMIA DELLE SCIENZE
DELL' ISTITUTO DI BOLOGNA

I.

ALGORITMI DE NUMERO INDORUM.

ROMA

TIPOGRAFIA DELLE SCIENZE FISICHE E MATEMATICHE
in Via Lata N.° 211.
1857.

Origine del nome

Abu Abdullah Muhammad bin Musa al-Khwarizmi

- E' stato un matematico, astronomo, astrologo e geografo
- Nato in Uzbekistan, ha lavorato a Baghdad
- Dal suo nome: algoritmo

Al-Kitab al-muhtasar fi hisab al-gabr wa-l-muqabala

- La sua opera più famosa (820 d.C.)
- Tradotta in latino con il titolo: *Liber algebrae et almucabala*
- Dal suo titolo: algebra

Problemi computazionali: esempi

Esempio: Minimo

Il minimo di un insieme S è l'elemento di S che è minore o uguale ad ogni elemento di S.

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

Esempio: Ricerca

Sia $S = s_1, s_2, \ldots, s_n$ una sequenza di dati ordinati e distinti, i.e. $s_1 < s_2 < \ldots < s_n$. Eseguire una ricerca della posizione di un dato v in S consiste nel restituire un indice i tale che $1 \leq i \leq n$, se v è presente nella posizione i, oppure 0, se v non è presente.

$$lookup(S, v) = \begin{cases} i & \exists i \in \{1, \ldots, n\} : S_i = v \\ 0 & \text{altrimenti} \end{cases}$$

Algoritmi: esempi

Algoritmo: 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.

Algoritmo: Ricerca

Per trovare un valore v nella sequenza S, confronta v con tutti gli elementi di S, in sequenza, 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

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, 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"?

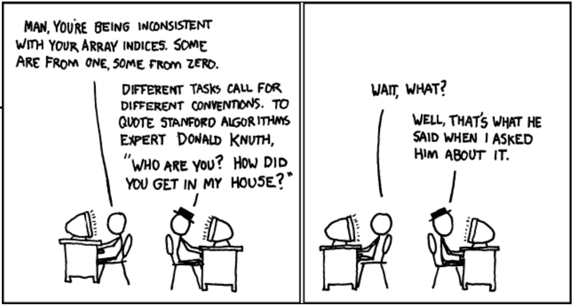
Esempio: pseudo-codice

int min(**int**[] S, **int** n)

```
for i = 1 to n do
    boolean isMin = true
    for j = 1 to n do
        if i ≠ j and S[j] < S[i]
          then
            isMin = false
    if isMin then
        return S[i]
```

int lookup(**int**[] S, **int** n, **int** v)

```
for i = 1 to n do
    if S[i] == v then
        return i
return 0
```



Pseudo-codice

- $a = b$
- $a \leftrightarrow b \;\equiv\; tmp = a; a = b; b = tmp$
- $T[\,]\ A =$ **new** $T[1 \ldots n]$
- $T[\,][\,]\ B =$ **new** $T[1 \ldots n][1 \ldots m]$
- **int**, **float**, **boolean**, **int**
- **and**, **or**, **not**
- ==, $\neq$, $\leq$, $\geq$
- $+$, $-$, $\cdot$, $/$, $\lfloor x \rfloor$, $\lceil x \rceil$, $\log$, x^2, $\ldots$
- iif$(condizione, v_1, v_2)$

- **if** *condizione* **then** istruzione
- **if** *condizione* **then** istruzione1 **else** istruzione2
- **while** *condizione* **do** istruzione
- **foreach** *elemento* $\in$ *insieme* **do** istruzione
- **return**
- % *commento*

Pseudo-codice

- **for** *indice* = *estremoInf* **to** *estremoSup* **do** istruzione

  int *indice* = *estremoInf*
  while *indice* $\leq$ *estremoSup* **do**
      istruzione
      indice = *indice* + 1

- **for** *indice* = *estremoSup* **downto** *estremoInf* **do** istruzione

  int *indice* = *estremoSup*
  while *indice* $\geq$ *estremoInf* **do**
      istruzione
      indice = *indice* − 1

- RETTANGOLO r = **new** RETTANGOLO
- $r.altezza = 10$
- **delete** r
- r = **nil**

RETTANGOLO
int *lunghezza*
int *altezza*

Come valutare l'algoritmo

Risolve il problema in modo efficiente?

- Dobbiamo stabilire come valutare se un programma è efficiente
- Alcuni problemi non possono essere risolti in modo efficiente
- Esistono soluzioni "ottime": non è possibile essere più efficienti

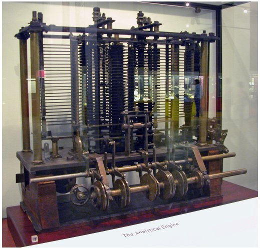
Risolve il problema in modo corretto?

- Dimostrazione matematica, descrizione "informale"
- Nota: Alcuni problemi non possono essere risolti
- Nota: Alcuni problemi vengono risolti in modo approssimato

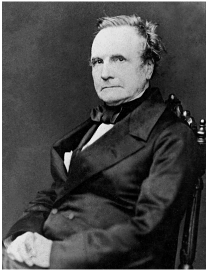
Charles Babbage

Passages from the Life of a Philosopher, Charles Babbage, 1864

As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise — By what course of calculation can these results be arrived at by the machine in the shortest time?



Modello della macchina analitica, Museo di Londra, foto Bruno Barral



Charles Babbage, 1860

Valutazione algoritmi – Efficienza

Complessità di un algoritmo

Analisi delle risorse impiegate da un algoritmo per risolvere un problema, in funzione della dimensione e dalla tipologia dell'input

Risorse

- Tempo: tempo impiegato per completare l'algoritmo
  - Misurato con il cronometro?
  - Misurato contando il numero di operazioni rilevanti?
  - Misurato contando il numero di operazioni elementari?
- Spazio: quantità di memoria utilizzata
- Banda: quantità di bit spediti (algoritmi distribuiti)

Definizione di 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 una rappresentazione più astratta!

Definizione di tempo – A grandi linee

Tempo ≡ n. operazioni rilevanti

Numero di operazioni "rilevanti", ovvero il numero di operazioni che caratterizzano lo scopo dell'algoritmo.

Esempio

- Nel caso del minimo, numero di confronti <
- Nel caso della ricerca, numero di confronti ==

Proviamo!

Valutazione algoritmi – Minimo

Contiamo il numero di confronti per il problema del minimo

```
int min(int[] S, int n)
for i = 1 to n do
    boolean isMin = true
    for j = 1 to n do
        if i ≠ j and S[j] < S[i] then
            isMin = false
    if isMin then
        return S[i]
```

Algoritmo “naïf”: $n^2 - n$

Si può fare meglio di così?

Valutazione algoritmi – Un algoritmo migliore

Contiamo il numero di confronti per il problema del minimo

```
int min(int[] S, int n)
% Partial minimum
int min = S[1]
for i = 2 to n do
    if S[i] < min then
        % Update partial minimum
        min = S[i]
return min
```

Algoritmo "naïf": $n^2 - n$

Algoritmo efficiente: $n-1$

Valutazione algoritmi – Ricerca

Contiamo il numero di confronti per il problema della ricerca

int lookup(**int**[] S, **int** n, **int** v)

for $i = 1$ **to** n **do**
  if $S[i] == v$ **then**
    return i
return 0

Algoritmo "naïf": n

Si può fare meglio di così?

Valutazione algoritmi – Un algoritmo migliore

Una soluzione più efficiente

Analizzo l'elemento centrale (indice m) del sottovettore considerato:

- Se $A[m] = v$, ho trovato il valore cercato
- Se $v < A[m]$, cerco nella "metà di sinistra"
- Se $A[m] < v$, cerco nella "metà di destra"

1	5	12	15	20	23	32

21?

Valutazione algoritmi – Un algoritmo migliore

Una soluzione più efficiente

Analizzo l'elemento centrale (indice m) del sottovettore considerato:

- Se $A[m] = v$, ho trovato il valore cercato
- Se $v < A[m]$, cerco nella "metà di sinistra"
- Se $A[m] < v$, cerco nella "metà di destra"

			m			
1	5	12	15	20	23	32

21?

Valutazione algoritmi – Un algoritmo migliore

Una soluzione più efficiente

Analizzo l'elemento centrale (indice m) del sottovettore considerato:

- Se $A[m] = v$, ho trovato il valore cercato
- Se $v < A[m]$, cerco nella "metà di sinistra"
- Se $A[m] < v$, cerco nella "metà di destra"

			m			
1	5	12	15	20	23	32

21?

Valutazione algoritmi – Un algoritmo migliore

Una soluzione più efficiente

Analizzo l'elemento centrale (indice m) del sottovettore considerato:

- Se $A[m] = v$, ho trovato il valore cercato
- Se $v < A[m]$, cerco nella "metà di sinistra"
- Se $A[m] < v$, cerco nella "metà di destra"

1	5	12	15	20	23	32

21?

Valutazione algoritmi – Un algoritmo migliore

Una soluzione più efficiente

Analizzo l'elemento centrale (indice m) del sottovettore considerato:

- Se $A[m] = v$, ho trovato il valore cercato
- Se $v < A[m]$, cerco nella "metà di sinistra"
- Se $A[m] < v$, cerco nella "metà di destra"

					m	
1	5	12	15	20	23	32

21?

Valutazione algoritmi – Un algoritmo migliore

Una soluzione più efficiente

Analizzo l'elemento centrale (indice m) del sottovettore considerato:

- Se $A[m] = v$, ho trovato il valore cercato
- Se $v < A[m]$, cerco nella "metà di sinistra"
- Se $A[m] < v$, cerco nella "metà di destra"

				m		
1	5	12	15	20	23	32

21?

Valutazione algoritmi – Un algoritmo migliore

Una soluzione più efficiente

Analizzo l'elemento centrale (indice m) del sottovettore considerato:

- Se $A[m] = v$, ho trovato il valore cercato
- Se $v < A[m]$, cerco nella "metà di sinistra"
- Se $A[m] < v$, cerco nella "metà di destra"

1	5	12	15	20	23	32

21?

Valutazione algoritmi – Un algoritmo migliore

Una soluzione più efficiente

Analizzo l'elemento centrale (indice m) del sottovettore considerato:

- Se $A[m] = v$, ho trovato il valore cercato
- Se $v < A[m]$, cerco nella "metà di sinistra"
- Se $A[m] < v$, cerco nella "metà di destra"

				m		
1	5	12	15	20	23	32

21?

Valutazione algoritmi – Un algoritmo migliore

Contiamo il numero di confronti per il problema della ricerca

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

Algoritmo "naïf": n

Algoritmo efficiente: $\lceil \log n \rceil$

Valutazione algoritmi – Correttezza

Invariante

Condizione sempre vera in un certo punto del programma

Invariante di ciclo

- Una condizione sempre vera all'inizio dell'iterazione di un ciclo
- Cosa si intende per "inizio dell'iterazione"?

Invariante di classe

- Una condizione sempre vera al termine dell'esecuzione di un metodo della classe

Valutazione algoritmi – Correttezza

Il concetto di invariante di ciclo ci aiuta a dimostrare la correttezza di un algoritmo iterativo.

- Inizializzazione (caso base):
  La condizione è vera alla prima iterazione di un ciclo
- Conservazione (passo induttivo):
  Se la condizione è vera prima di un'iterazione del ciclo, allora rimane vera al termine (quindi prima della successiva iterazione)
- Conclusione:
  Quando il ciclo termina, l'invariante deve rappresentare la "correttezza" dell'algoritmo

Valutazione algoritmi – Correttezza

All'inizio di ogni iterazione del ciclo **for**, la variabile *min* contiene il minimo parziale degli elementi $S[1 \ldots i-1]$.

int min(**int**[] S, **int** n)

int $min = S[1]$
for $i = 2$ **to** n **do**
  if $S[i] < min$ **then**
    $min = S[i]$
return min

Inizializzazione

Conservazione

Conclusione

Valutazione algoritmi – Correttezza

La dimostrazione per induzione è utile anche con gli algoritmi ricorsivi

```
int binarySearch(int[] A, int v, int i, int j)
if i > j then
    return 0
else
    int 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)
```

Per induzione sulla dimensione n dell'input

- Caso base: $n = 0$ $(i > j)$
- Ipotesi induttiva: vero per tutti gli $n' < n$
- Passo induttivo: dimostrare che è vero per n

Altre proprietà

Semplicità, modularità, manutenibilità, espandibilità, robustezza, ...

- Secondari in un corso di algoritmi e strutture dati
- Fondamentali per un corso di ingegneria del software

Commento

Alcune proprietà hanno un costo aggiuntivo in termini di prestazioni

- Codice modulare → costo gestione chiamate
- Java bytecode → costo interpretazione

Progettare algoritmi efficienti è un prerequisito per poter pagare questo costo