Algoritmi e Strutture Dati (ASD)

Docente: András Horváth
horvath@di.unito.it

1

Il corso

- 48 ore di teoria
- 30 ore in laboratorio
- sito moodle: avvisi, appunti, lucidi, esercizi
- esame composto da un scritto e la discussione dei programmi sviluppati in laboratorio
- 6+3=9 cfu

2

Approccio didattico

- **lucidi non bastano** per studiare!
- appunti disponibili sono più dettagliati ma possono non bastare
- libro: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: **Introduzione agli algoritmi e strutture dati**.
- fate domande!!

3

Problemi e algoritmi

Algoritmi e strutture dati
Lezione 1

Andras Horvath, Ugo de'Liguoro

4

Sommario

- obiettivi:
  - introduzione alla ***terminologia***, allo ***sviluppo*** *ed all'****analisi degli algoritmi***
- argomenti:
  - problemi computazionali
  - algoritmi
  - esempio: peak finding
  - insolubilità ed intrattabilità

5

Problemi computazionali

Un ***problema computazionale*** è una collezione di domande, le ***istanze (ingressi)***, per cui sia stabilito un ***criterio*** (astratto) per riconoscere le ***risposte (uscite)*** corrette.

Massimo comune divisore
- *ingresso*:
  coppia di interi non negativi a e b; non entrambi nulli
- *uscita*:
  un intero c tale che soddisfa il seguente *criterio*:
  1) c divide sia a che b
  2) non esiste $d > c$ che divide sia a che b

6

Problemi computazionali

- un problema è una ***relazione binaria***:

$$R = \{(istanza, risposta) \mid istanza,\ risposta \text{ soddisfano } \ldots\}$$

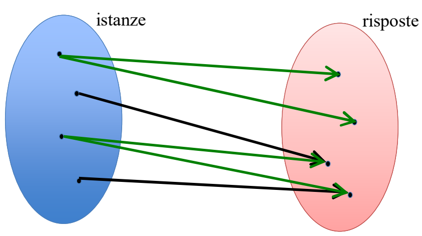
- il ***dominio*** della relazione:

$$dom(R) = \{i \mid \exists\, r.\ (i, r) \in R\}$$

- la relazione R è ***univoca*** se ogni istanza ammette una sola risposta

7

Relazioni



Relazione non univoca.

8

Problemi computazionali

- moltiplicazione fra due interi
- fattorizzazione
- ordinamento (sorting)
- percorso ottimo in un grafo (shortest path)

9

Problemi computazionali

- la ***moltiplicazione*** fra due interi:
  $R = \{((a, b), c) | a \in \mathbb{Z}, b \in \mathbb{Z}, a \cdot b = c\}$
- la ***fattorizzazione***:
  $R = \{(n, (c_1, c_2, \dots, c_k)) | n \in \mathbb{Z}, n \geq 2,$
  $n = c_1 \cdot c_2 \cdot \dots \cdot c_k, c_i \in P\}$

  dove P è l'insieme di numeri primi
- R della moltiplicazione è univoca
- R della fattorizzazione non è univoca, con $n = 10$ sia $c_1 = 2, c_2 = 5$ e $c_1 = 5, c_2 = 2$ sono risposte che soddisfano i requisiti (cioè $(10, (2{,}5)) \in R$ e anche $(10, (5{,}2)) \in R$)

10

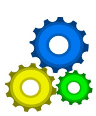
Problemi computazionali

- quali fra i problemi precedenti sono univoci?
- moltiplicazione: SI
- fattorizzazione: come definito due lucidi fa NO, ma lo diventa se richiediamo i fattori in ordine non decrescente
- ordinamento (sorting): SI (a meno di distinguere gli elementi che hanno lo stesso valore)
- percorso ottimo (shortest path): NO

11

Algoritmo

Un ***algoritmo*** è un metodo meccanico per risolvere un problema computazionale.



12

Algoritmo, terminologia

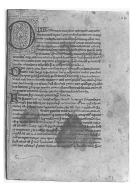
Una ***procedura*** è una sequenza finita di operazioni meccanicamente eseguibili, per produrre un'uscita a partire da certi ingressi.

Un ***algoritmo*** è una procedura che termina per ogni ingresso ammissibile.

13

Il termine **algoritmo**

Abū Jaʿfar Muhammad ibn Mūsā al-Khwārizmī
780 - 850 ca



Algoritmi de numero Indorum

14

L'algoritmo di Euclide

Euclide, 367-283 a.C.

$\textsc{Euclid}(a, b) \quad \triangleright\ a > 0 \vee b > 0$
if $b = 0$ **then**
  return a
else $\quad \triangleright\ b \neq 0$
  $r \leftarrow a \bmod b$
  while $r \neq 0$ **do**
    $a \leftarrow b$
    $b \leftarrow r$
    $r \leftarrow a \bmod b$
  end while
  return b
end if

15

La funzione input-output

- un algoritmo è ***deterministico***: se eseguito più volte sullo stesso input, fornisce sempre lo stesso output
- dunque ad ogni algoritmo deterministico possiamo associare una ***funzione input-output***:

$$A(input) = output$$

16

Algoritmi e problemi

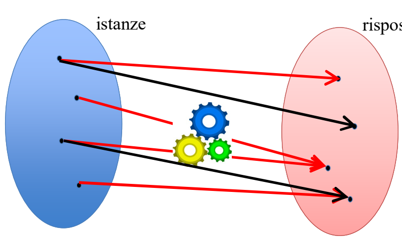
Un algoritmo ***risolve*** un problema R, ossia è ***corretto*** rispetto ad R, se la sua funzione input-output A associa una risposta ad ogni istanza di R:

$$(i, A(i)) \in R \text{ per ogni } i \in dom(R)$$

(A sceglie una risposta corretta per ogni istanza.)

17

Problemi e algoritmi



18

Correttezza di Euclid

```
EUCLID(a, b)        ▷ a > 0 ∨ b > 0
if b = 0 then
    return a
else      ▷ b ≠ 0
    r ← a mod b
    while r ≠ 0 do
        a ← b
        b ← r
        r ← a mod b
    end while
    return b
end if
```

Come fare per dimostrare che Euclid calcola davvero MCD(a,b)?

Lemma. Se $b > 0$ allora MCD(a,b) = MCD(b, a mod b)

19

Algoritmi versus programmi

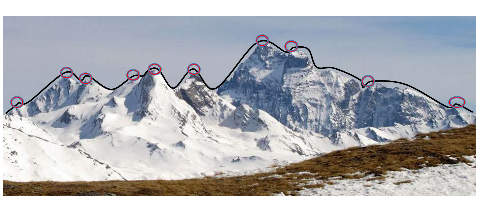
- un ***programma*** può implementare un o più ***algoritmi***
- in un ***programma*** occorre specificare ed implementare opportune ***strutture dati***

Algoritmi + Strutture Dati = Programmi

- un programma è scritto in uno specifico ***linguaggio di programmazione***

20

Peak finding



21

Peak finding

Il problema di ***peak finding***:

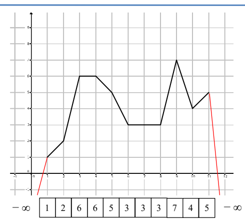
Input: un vettore $A[0..n-1]$ di interi positivi.

Output: un intero $0 \leq p < n$ tale che $A[p-1] \leq A[p] \geq A[p+1]$
dove $A[-1] = A[n] = -\infty$.

(Cioè nella posizione p si trova un picco.)

22

Esempio



- i picchi sono: $A[2] = 6, A[3] = 6, A[6] = 3, A[8] = 7$ e $A[10] = 5$
- l'algoritmo deve trovare uno dei picchi (non tutti e non quello più alto)

23

Left Peak finding

```
PEAK-FIND-LEFT(A, n)        ▷ n ≥ 1
p ← 0
k ← 1
while k ≤ n − 1 ∧ A[p] < A[k] do
    p ← k
    k ← k + 1
end while
return p
```

- Peak-Find-Left trova il picco più a sinistra in $A[0..n-1]$
- nel ***caso migliore*** $p = 0$ è un picco
- nel ***caso peggiore*** il picco più a sinistra è $p = n - 1$ (il vettore $A[0..n-1]$ è una "salita")

Quindi nel caso peggiore si effettuano $n - 1$ confronti.

Si può fare meglio?

24

Max Peak finding

```
PEAK-FIND-MAX(A, n)          ▷ n ≥ 1
p ← 0
for k ← 1 to n − 1  do
    if A[p] < A[k] then
        p ← k
    end if
end for
return p
```

- se $A[p]$ è il massimo in $A[0..n\text{-}1]$ allora p è un picco (il picco più "alto")
- in tutti i casi si effettuano $n-1$ confronti
- lo sforzo associato con caso peggiore del Peak-Find-Left basta per trovare il picco più alto

E' possibile trovare un picco qualunque in minor tempo?

25

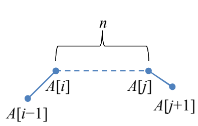
Peak finding

Dato un segmento del vettore, A[i..j], che cosa assicura che un picco esista al suo interno?

26

Peak finding

Teorema. Siano $i \le j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \le p \le j$ tale che $A[p-1] \le A[p] \ge A[p+1]$ ossia p è un picco in $A[i..j]$.



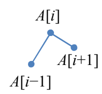
- sia n il numero di elementi nel segmento considerato, cioè $n = j - i + 1$

27

Peak finding

Teorema. Siano $i \leq j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \leq p \leq j$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ ossia p è un picco in $A[i..j]$.

- consideriamo il caso $i = j$, cioè $n = 1$



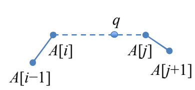
- la posizione i è un picco!

28

Peak finding

Teorema. Siano $i \leq j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \leq p \leq j$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ ossia p è un picco in $A[i..j]$.

- consideriamo il caso $i < j$, cioè $n > 1$



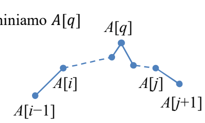
- scegliamo una qualunque posizione q tale che $i \leq q \leq j$

29

Peak finding

Teorema. Siano $i \leq j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \leq p \leq j$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ ossia p è un picco in $A[i..j]$.

- esaminiamo $A[q]$



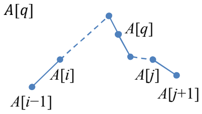
- $A[q]$ stesso può essere un picco

30

Peak finding

Teorema. Siano $i \leq j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \leq p \leq j$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ ossia p è un picco in $A[i..j]$.

- se $A[q]$ non è picco allora $A[q-1]$ oppure $A[q+1]$ è maggiore di $A[q]$



- se $A[q-1] > A[q]$ (come nella figura) ripetiamo il ragionamento su $A[i..q-1]$
- altrimenti su $A[q+1..j]$

31

Peak finding

Teorema. Siano $i \leq j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \leq p \leq j$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ ossia p è un picco in $A[i..j]$.

- se $n=1$ le condizioni garantiscono la presenza di un picco nella posizione $p=i=j$
- se $n>1$ allora scegliamo una posizione q, $i \leq q \leq j$:
  - se q è picco allora il picco c'è
  - se q non è picco procediamo sul segmento $i..q-1$ oppure sul segmento $q+1..j$
- prima o poi o troviamo un picco in una posizione scelta a caso oppure arriviamo ad un segmento che contiene un elemento solo e quindi è un picco

32

Peak finding

Teorema. Siano $i \leq j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \leq p \leq j$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ ossia p è un picco in $A[i..j]$.

Dimostrazione formale con induzione completa su $n = j - i + 1$:

- $\boldsymbol{n = 1}$: allora $i = j$ e $A[i-1] < A[i] > A[i+1]$ allora $p = i$ è un picco.
- $\boldsymbol{n > 1}$: dimostriamo che, **se il teorema è vero per qualunque intero positivo minore di $\boldsymbol{n}$**, allora è vero anche per n:
- sia q, $i \leq q \leq j$, un punto qualsiasi; ci sono tre possibilità:

1. **se $\boldsymbol{q}$ è un picco** allora il picco c'è
2. **se $\boldsymbol{A[q-1] > A[q]}$** : $i..q-1$ è un segmento più piccolo di $i..j$ e soddisfa le ipotesi del teorema, dunque secondo l'ipotesi induttiva il picco c'è
3. **se $\boldsymbol{A[q+1] > A[q]}$** : $q+1..j$ è un segmento più piccolo di $i..j$ e soddisfa le ipotesi del teorema, dunque secondo l'ipotesi induttiva il picco c'è

33

Peak finding

Teorema. Siano $i \leq j$. Se $A[i-1] < A[i]$ e $A[j] > A[j+1]$ allora esiste $i \leq p \leq j$ tale che $A[p-1] \leq A[p] \geq A[p+1]$ ossia p è un picco in $A[i..j]$.

Quindi nel caso di $A[0..n-1]$ di interi positivi e $A[-1] = A[n] = -\infty$ dal teorema segue che in $A[0..n-1]$ esiste un picco.

34

Peak finding Divide et Impera

- nel provare che A ha sempre un picco abbiamo scelto un punto arbitrario q per iniziare la ricerca
- quale scelta di q sembra vantaggiosa per avere un procedimento veloce?
- potrebbe essere vantaggioso scegliere la posizione centrale

35

Peak finding Divide et Impera

```
PEAK-DI(A, i, j)        ▷ i ≤ j
p ← (i + j)/2
if A[p − 1] ≤ A[p] ≥ A[p + 1] then
    return p
else      ▷ A[p − 1] > A[p] ∨ A[p] < A[p + 1]
    if A[p − 1] > A[p] then
        return PEAK-DI(A, i, p − 1)
    else
        return PEAK-DI(A, p + 1, j)
    end if
end if

PEAK-FIND-DI(A, n)      ▷ n ≥ 1
return PEAK-DI(A, 0, n − 1)
```

Quanti confronti fa? Quanto tempo impiega?

36

Analisi di Peak-DI

```
PEAK-DI(A, i, j)        ▷ i ≤ j
p ← (i + j)/2
if A[p − 1] ≤ A[p] ≥ A[p + 1] then
    return p
else      ▷ A[p − 1] > A[p] ∨ A[p] < A[p + 1]
    if A[p − 1] > A[p] then
        return PEAK-DI(A, i, p − 1)
    else
        return PEAK-DI(A, p + 1, j)
    end if
end if

PEAK-FIND-DI(A, n)      ▷ n ≥ 1
return PEAK-DI(A, 0, n − 1)
```

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ T\left(\frac{n}{2}\right) + 1 & \text{se } n > 1 \end{cases}$$

37

Analisi di Peak-DI

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ T\left(\frac{n}{2}\right) + 1 & \text{se } n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + 1 \\ &= T\left(\frac{n}{4}\right) + 1 + 1 \\ &= T\left(\frac{n}{8}\right) + 1 + 1 + 1 \\ &= T\left(\frac{n}{2^3}\right) + 3 \\ &= T\left(\frac{n}{2^k}\right) + k \qquad \text{per } 1 \leq k \leq \log_2 n \\ &= T(1) + \log_2 n \\ &= 1 + \log_2 n \end{aligned}$$

- provare con $n = 2^5 = 32$ per crederci

38

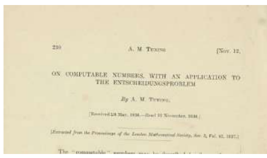
Analisi di Peak-DI

- Peak-Find-Left (nel caso peggiore) e Peak-Find-Max affettuano *n-1* confronti
- Peak-DI effettua all'incirca $log_2 n$ confronti
- con un vettore di 1000 elementi servono circa 10 confronti invece di circa 1000

39

Problemi insolubili (o indecidibili)

E' forse vero che tutti i problemi computazionali ammettano una soluzione algoritmica?


230 A. M. TURING [Nov. 12,

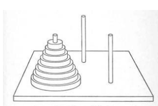
ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

Alan M. Turing
(1912 - 1954)

40

Problemi intrattabili



Torri di Hanoi

Input: n dischi sovrapposti di diametro decrescente su di un piolo
Output: spostare tutti i dischi su un piolo diverso, movendo un disco per volta senza mai sovrapporre un disco più grande ad uno più piccolo, usando solo un terzo piolo d'appoggio

Sono necessarie $2^n - 1$ mosse.

41