algoritmi e strutture di dati

definizioni di base

m.patrignani

nota di copyright

definizioni di base

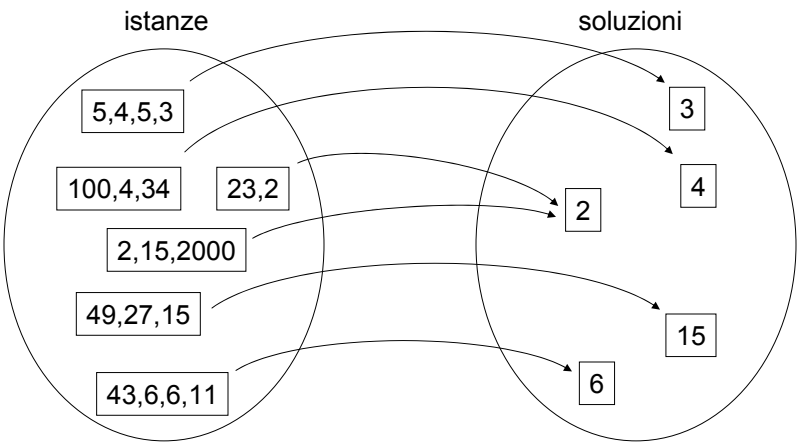
- problemi computazionali
- algoritmi
  - Random Access Machine (RAM)
  - pseudocodifica
- strutture di dati

problemi

- un *problema computazionale* esprime la relazione tra due insiemi di valori, detti rispettivamente di *input* e di *output*
  - un insieme legittimo di valori di input è chiamato *istanza*, e i vincoli che rispetta sono descritti in termini assoluti
    - esempio: un insieme di *n* interi positivi $\{a_1, a_2, ..., a_n\}$
  - un insieme legittimo di valori di output è chiamato *soluzione* ed è descritto in termini dell'istanza
    - esempio: il valore minore tra $a_1, a_2, ..., a_n$

esempio

minimo di un insieme di interi positivi



esempi di problemi computazionali

- ordinamento
  - input: una sequenza di numeri
  - output: la stessa sequenza ordinata
- ricerca
  - input: una collezione di dati ed una chiave k
  - output: il dato associato alla chiave k
- percorsi ottimali
  - input: un grafo pesato e due nodi u e v
  - output: il cammino di costo minimo tra u e v
- prodotto di matrici
  - input: n matrici A_1, A_2, ..., A_n
  - output: il loro prodotto $A_1A_2A_3...A_n$
- involucro convesso
  - input: n punti sul piano
  - output: il più piccolo poligono convesso che li contiene
- ...

importanza dei problemi computazionali

- i problemi computazionali sono il cuore di molte applicazioni
  - motori di ricerca
  - instradamento del traffico in Internet
  - computer aided design
  - computer audio
  - computer grafica
  - simulatori ed emulatori
  - giochi
  - crittografia
  - bioinformatica (sequenziamento, ecc)
  - compressione di dati

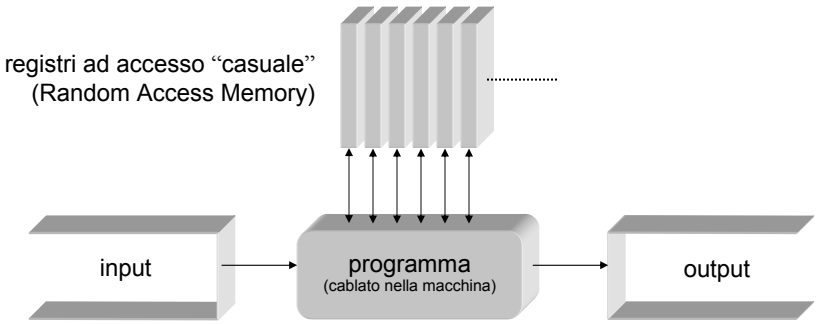
algoritmi

- un *algoritmo* è una procedura di calcolo ben definita che, a partire da un insieme di valori in input, produce valori in output
  - questa definizione sarà più rigorosa quando avremo definito cos'è una "procedura di calcolo ben definita"
- un algoritmo è detto *corretto* per un problema computazionale p se per ogni istanza x di p
  - l'algoritmo termina
  - l'algoritmo produce un output y corretto
    - cioè tra x e y vale la relazione specificata da p

Random Access Machine

che vuol dire "procedura di calcolo ben definita"?

- risposta rigorosa
  - computabile da una Random Access Machine
    - astrazione ragionevole di un calcolatore
    - più elementare del modello di von Neumann



RAM

- caratteristiche della Random Access Machine
  - memoria costituita da un numero arbitrario di registri indirizzabili individualmente (Random Access Memory)
  - input e output sono sequenze di dati (streams o nastri)
  - ogni operazione (trasferimento, somma, sottrazione, moltiplicazione, divisione) ha costo unitario
  - le istruzioni sono eseguite in maniera sequenziale
- limitazioni
  - i valori in input, output e memoria RAM sono supposti illimitati
    - questa ipotesi è talvolta irragionevole
  - la RAM modella solo macchine mono-processore

algoritmi e Random Access Machines

- specificare una RAM che implementi un dato algoritmo è molto oneroso
  - le operazioni di base della RAM costituiscono un linguaggio macchina elementare
- esempio di RAM (somma di due numeri)

```
READ R1
READ R2
LOAD R1
ADD R2
STORE R3
WRITE R3
```

→ legge un dato di input e lo copia in R1
→ legge un dato di input e lo copia in R2
→ copia il registro R1 nell'accumulatore
→ somma l'accumulatore con R2
→ copia l'accumulatore in R3
→ scrive il contenuto di R3 in output

pseudocodice

che vuol dire "procedura di calcolo ben definita"?

- risposta pragmatica (ma equivalente alla precedente)
  - descrivibile con uno pseudocodice elementare
    - che dispone di variabili, array, procedure, istruzioni condizionali, istruzioni ripetitive, ecc
  - esempio di pseudocodice

```
CALCOLO-MINIMO(A)
1.  min = A[0]
2.  for i = 1 to A.length-1
3.      if A[i] < min then
4.          ▷ aggiorno il valore temporaneo di min
5.          min = A[i]
6.  return min
```


strutture di dati

- i dati su cui opera l'algoritmo sono conservati in un insieme di variabili atomiche, oggetti ed array
- scopo
  - salvataggio temporaneo di risultati parziali
  - preprocessing sull'input per rendere più facile la loro elaborazione successiva
- i dati in memoria possono essere organizzati secondo diverse strategie o discipline
  - esempio: un array può essere ordinato
- tali discipline sono chiamate *strutture di dati*

strutture di dati

- una *struttura di dati* è un contenitore di dati
  - è concepita per rendere il dato disponibile nella forma più utile e veloce per produrre l'output
    - il dato contenuto nella struttura di dati è a tutti gli effetti un semi-lavorato
  - esempio
    - supponiamo che un insieme di numeri sia conservato in un array ordinato
    - l'elemento minimo dell'insieme sarà il primo elemento dell'array

pseudocodice: variabili e assegnazioni

- le variabili dello pseudocodice hanno associato un tipo (intero, reale, booleano, …) ma non necessitano di dichiarazione
  - in caso di equivoco si ricorre ai commenti
- le assegnazioni sono denotate da un segno di ugualianza (=)
  - equivalentemente si può usare una freccia verso sinistra (←)
- possibili assegnazioni multiple: i = j = k
  - equivalente a j = k e i = j
- esempi

```
1.  max = -1          ▷ max è un intero
2.  max = max + 1     ▷ incremento max
```

```
1.  ▷ definisco un reale che chiamo area
2.  area = base * altezza / 2
```


pseudocodice: istruzioni condizionali

- if-then-else, con ovvio significato
  - talvolta per brevità si tace il "then"
- un blocco di istruzioni è denotato tramite indentazione
- esempi

```
1.  if i < 0 then
2.      ▷ prendo il valore assoluto di i
3.      i = -i
```

```
1.  if i < 0 then
2.      abs = -i
3.      minore_zero = true
4.  else
5.      abs = i
6.      minore_zero = false
```


pseudocodice: espressioni booleane

- sono espressioni booleane
  - le variabili booleane e le costanti **`true`** e **`false`**
  - il risultato di operatori booleani come: and, or, not
  - il risultato di operatori relazionali come: ==, >, ≥, <, ≤
- attenzione
  - talvolta viene usato il simbolo uguale (=) per denotare l'operatore relazionale e la freccia (←) per l'assegnazione
    - è però indispensabile utilizzare i simboli in modo coerente!

```
1.  if i < max and trovato == false then
2.       i = i+1
```


pseudocodice: istruzione ripetitiva for

- esempi

```
1.  for i = 0 to A.length-1
2.      ▷ pongo a zero tutte le posizioni di A
3.      A[i] = 0
```

```
1.  for i = A.length-1 down to 1
2.      ▷ traslo in avanti tutti i valori di A
3.      A[i] = A[i-1]
4.  A[0] = new    ▷ inserisco new in testa
```

- non è possibile porre condizioni aggiuntive ad un for
  - esempio errato

```
1.  for i = 0 to A.length-1 and trovato == false
```


pseudocodice: istruzione ripetitiva while

- il blocco viene eseguito finché (=mentre) la condizione è verificata
  - la condizione è una condizione "di permanenza nel ciclo"
  - se la condizione è falsa il blocco non viene eseguito mai
  - a volte per brevita si tace il "do"

```
1. ▷ pongo a zero tutte le posizioni di A
2. i = 0
3. while i < A.length do
4.     A[i] = 0
5.     i = i+1
```

```
1. while i < A.length and not trovato do
2.     if A[i] == quello_che_cerco then
3.         trovato = true
4.     else
5.         i = i+1
```


istruzione ripetitiva repeat until

- il blocco viene eseguito finché (=mentre) la condizione è falsa
  - la condizione è una condizione "di uscita dal ciclo"
  - il blocco viene eseguito almeno una volta
- esempio

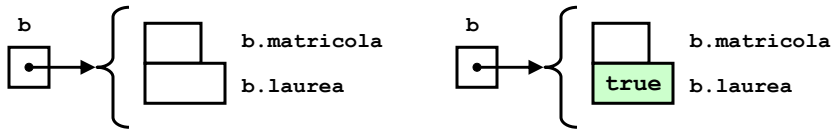
```
1. ▷ pongo a zero tutte le posizioni di A
2. i = 0
3. repeat
4.     A[i] = 0
5.     i = i+1
6. until i == A.length
```


pseudocodice: oggetti

- è possibile definire dati compositi (oggetti) stutturati in attributi
- si accede all'attributo **`campo`** dell'oggetto **`x`** tramite **`x.campo`**

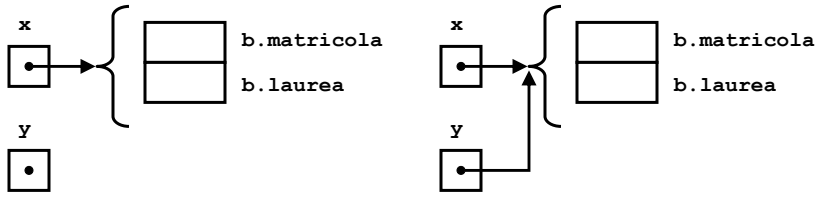
```
1.  ▷ b è un oggetto con gli attributi
2.  ▷ matricola (intero) e laurea (booleano)
3.  b.laurea = true      ▷ con mille auguri!
```

- una variabile che rappresenta un oggetto è trattata come un riferimento (puntatore) all'oggetto



oggetti e operazioni sui riferimenti

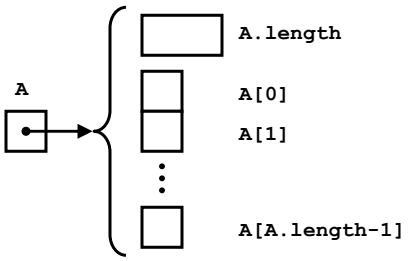
- se **`x`** e **`y`** sono due (riferimenti ad) oggetti
  - dopo l'assegnazione **`y = x`** si ha **`y.campo = x.campo`**
  - se si modifica un attributo di **`x`** (**`x.campo = 3`**), si modifica anche il corrispondente attributo di **`y`** (**`y.campo = 3`**)



- per porre a zero un riferimento si usa la costante **`NULL`**
- ATTENZIONE:
  - in linguaggio C si fa distinzione tra un oggetto ed il suo riferimento
  - in pseudocodice abbiamo solo il riferimento

pseudocodice: array

- si possono definire array di qualunque tipo
- A[i] identifica l'elemento dell'array A in posizione i
- la lunghezza dell'array A è data da A.length
  - A.length è un campo, non una funzione
  - ATTENZIONE: non c'è equivalente di A.length in linguaggio C
- le posizioni dell'array A vanno da 0 ad A.length-1



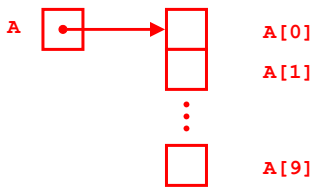
di fatto, un array è visto come un oggetto che ha dei campi indicizzati tramite un intero

esempi di uso di array

```
1.  A[0] = 1              ▷ A è un array 10 di interi
2.  A[A.length-1] = 0     ▷ l'ultimo elemento è zero
```

```
1.  ▷ A è un array contenente 3 array di 10 interi
2.  A[2][9] = 0       ▷ l'ultimo elemento di A[2] è 0
3.  A[2,9] = 0        ▷ equivalente alla precedente
```

- ATTENZIONE: in linguaggio C gli array non hanno il campo length!!



la lunghezza dell'array in linguaggio C deve essere memorizzata in un'altra variabile, come per esempio

lengthA | 10

pseudocodice e tipi delle variabili

- lo psedocodice è privo di definizioni di tipo
- le variabili si suppongono allocate in memoria nel momento in cui vengono menzionate per la prima volta
- i commenti vengono utilizzati per esplicitare il loro tipo, se ciò non è evidente dal contesto (come nelle linee 2 e 5 qui sotto)
  - ciò è evidentemente necessario per gli oggetti

```
1.  i = p - 1          (in assenza di commento il tipo di i è lo stesso di p)
2.  x.info = 1         ▷ x è un oggetto con i campi: info
3.                     ▷ (intero) next e prev (riferimenti ad
4.                     ▷ oggetti dello stesso tipo)
5.  TEMP[1] = 3        ▷ TEMP è un array di 20 interi
6.  y = x              (il tipo di y è lo stesso di x: commento non necessario)
7.  TEMP[2] = 4        (TEMP già noto: commento non necessario)
```


pseudocodice e procedure

- le variabili utilizzate in una procedura sono sempre locali alla procedura stessa
  - non esistono variabili globali
- alcune procedure non ritornano alcun valore
  - la sequenza delle loro istruzioni termina, oppure viene eseguita l'istruzione **`return`**
- alcune procedure ritornano un valore
  - le loro istruzioni terminano sempre con **`return`** <*espressione*>
    - dove la valutazione di <*espressione*> genera il valore ritornato
  - non si possono ritornare due o più valori
    - eventualmente si usano degli oggetti costruiti opportunamente

procedure e parametri

- il passaggio dei parametri ad una procedura è solo per valore
  - la procedura riceve il valore dei *parametri formali* tramite i *parametri attuali* presenti nella chiamata
    - un parametro attuale è in generale un'espressione, che può anche ridursi ad una singola variabile
- un parametro è a tutti gli effetti una variabile locale inizializzata con il valore ricevuto al momento della chiamata
- se il valore di un parametro formale viene modificato, la modifica non ha effetto all'esterno sul parametro attuale
  - non c'è "side effect"
- se però il parametro è un oggetto o un'array, l'oggetto non viene riprodotto, ma viene passato il valore del riferimento (indirizzo, puntatore) all'oggetto/array stesso
  - le assegnazioni `oggetto.campo=valore` e `array[indice]=valore` hanno quindi un side effect sull'oggetto e sull'array passati come parametro

esercizi sullo pseudocodice

- scrivi la procedura `MASSIMO`(*a*,*b*) che ritorni il massimo tra due interi *a* e *b*
- scrivi la procedura `MASSIMO`(*A*) che riceva come parametro un array di interi *A* e ritorni il massimo dei valori contenuti
- scrivi la procedura `SOMMA`(*M*) che riceva come parametro una matrice (un array di array) di interi *M* e ritorni la somma dei valori contenuti
  - puoi usare o meno `MASSIMO`(*A*) per realizzare `SOMMA`(*M*)
- scrivi la procedura `POSITIVO`(*A*) che riceva come parametro un array di interi *A* e verifichi che *A* contenga solo valori maggiori di zero
- scrivi la procedura `POSIZIONE-MASSIMO`(*A*) che riceva come parametro un array di interi *A* e ritorni il valore massimo contenuto e la sua posizione nell'array

