Una lezione (quasi) universitaria di Informatica.

Liceo Scientifico Statale "Galileo Ferraris".

Torino, 27 aprile 2007

Parte II

Elio Giovannetti
Dipartimento di Informatica
Università di Torino

Università di Torino – Facoltà di Scienze MFN
Corso di Studi in Informatica
Curriculum SR (Sistemi e Reti)

Algoritmi e Laboratorio a.a. 2006-07
Lezioni

prof. Elio Giovannetti

Lezione 2 – Cicli e invarianti
Adattata per lezione Galileo Ferraris.

Il principio di induzione matematica semplice.

Sia P(n) una proposizione che dipende da un numero naturale n variabile. Ad esempio:

Schumacher è in testa dopo n giri di pista.

Supponiamo che Schumacher sia in testa dopo 0 giri di pista (cioè parta in pole position), cioè:

$P(0)$

Supponiamo che sia vero che, per qualunque intero $k \geq 0$, se Schumacher è in testa dopo k giri, allora è in testa dopo k+1 giri; cioè:

$\forall k \,.\, (P(k) \rightarrow P(k+1))$

Allora, essendo in testa dopo 0 giri, sarà in testa dopo 1 giro; ma, essendo in testa dopo 1 giro, sarà in testa dopo 2 giri; ecc.

Possiamo concludere che Schumacher sarà in testa dopo qualunque numero n di giri.

Il principio di induzione matematica semplice.

$P(0) \wedge \forall k \,.\, (P(k) \rightarrow P(k+1)) \quad \rightarrow \quad \forall n \,.\, P(n)$

Data una proposizione P(n) "dipendente" da un numero naturale n, se:

- dimostro che essa vale per n = 0;
- assumendo che essa valga per un naturale non specificato k, riesco a dimostrare che essa vale per k+1;

allora ho dimostrato che essa vale per qualunque naturale n (cioè per tutti i naturali n).

Lo schema generale di una dimostrazione per induzione è allora:

Base: P(0)

dimostrazione della proposizione P(0)

Passo di induzione:

Ipotesi induttiva: P(k)

Tesi induttiva: P(k+1)

dimostrazione della tesi P(k+1) usando l'ipotesi P(k)

Cioè una dimostrazione per induzione è fatta di due dimostrazioni: quella della base e quella del passo.

Di solito (ma non sempre) la dimostrazione della base è ovvia, mentre la cosa non banale è la dimostrazione del passo.

Base diversa da zero

Come base dell'induzione si può prendere un naturale m > 0; in tal caso naturalmente ciò che si dimostra è che la proposizione considerata è vera per tutti i naturali n >= m.

Un esempio di dimostrazione per induzione

La proposizione P(n) da dimostrare è:

$1+2+ \ldots +n = n(n+1)/2$

Base: P(1), cioè $1 = 1(1+1)/2$

Dimostrazione della base:

ovvia, perché $1 = 1(1+1)/2$ è ovviamente vero.

Passo di induzione:

Ipotesi induttiva: $1+2+ \ldots +k = k(k+1)/2$

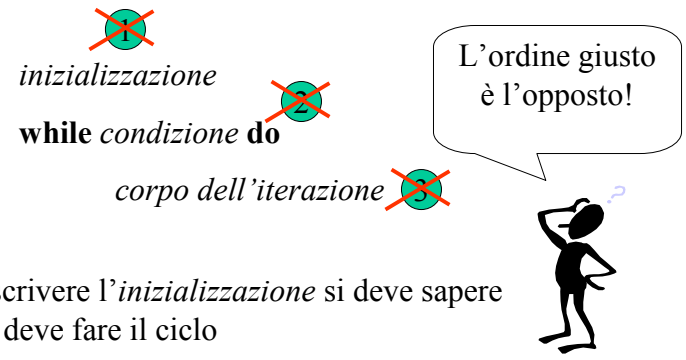
Tesi induttiva: $1+2+ \ldots +k + (k+1) = (k+1)(k+1+1)/2$

Dimostrazione del passo:

$1+2+ \ldots +k + (k+1) =$ per ip.ind.

$$= \frac{k(k+1)}{2} + (k+1) = \frac{k(k+1) + 2(k+1)}{2} = \frac{(k+2)(k+1)}{2}$$

Come si inventa un ciclo?



1. Per scrivere l'*inizializzazione* si deve sapere cosa deve fare il ciclo
2. Per scrivere la *condizione* (*guardia*) si deve conoscere cosa farà il corpo

La generica iterazione

- Per scrivere correttamente il corpo del ciclo non ci si deve porre agli estremi (inizio o fine del ciclo) ma in un ideale punto intermedio: la generica iterazione !

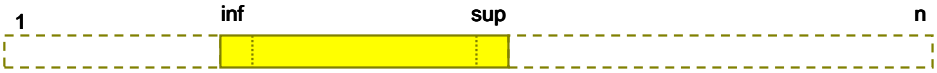
Nel seguito del corso vedremo esempi di problemi per i quali è quasi impossibile scrivere correttamente la soluzione iterativa senza partire col pensare alla "situazione al passo generico", cioè all'invariante di ciclo (ad es. ricerca binaria, problema della bandiera, eliminazione di elementi da un array con compattazione, ecc.).

Un algoritmo classico, eseguito manualmente da tutti nella vita di ogni giorno:

la ricerca binaria (o dicotomica).

Ricerca binaria in array (pieno) ordinato

INVARIANTE



Il valore **x** da cercare, se è presente nell'array, si trova nella porzione di array compresa fra gl'indici **inf** e **sup** (inclusi).

Ricerca binaria in array ordinato

INVARIANTE

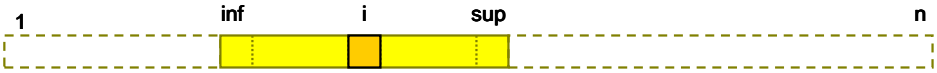
Il valore **x** da cercare, se è presente nell'array, si trova nella porzione di array compresa fra gl'indici **inf** e **sup** (inclusi).

PASSO GENERICO

Calcolo il valore dell'indice **i** dell'elemento centrale della porzione di array; confronto **x** con tale elemento `a[i]`; sono possibili tre casi:

Ricerca binaria in array ordinato

INVARIANTE



Il valore **x** da cercare, se è presente nell'array, si trova nella porzione di array compresa fra gl'indici **inf** e **sup** (inclusi).

PASSO GENERICO

Calcolo il valore dell'indice **i** dell'elemento centrale della porzione di array; confronto **x** con tale elemento `a[i]`; sono possibili tre casi:

- **x < a[i]** x, se c'è, si trova nella porzione compresa fra inf e i-1 (inclusi);

Ricerca binaria in array ordinato

INVARIANTE

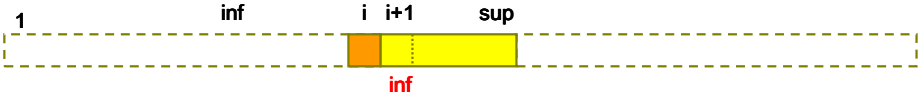
Il valore **x** da cercare, se è presente nell'array, si trova nella porzione di array compresa fra gl'indici **inf** e **sup** (inclusi).

PASSO GENERICO

Calcolo il valore dell'indice **i** dell'elemento centrale della porzione di array; confronto **x** con tale elemento `a[i]`; sono possibili tre casi:

- **x < a[i]** x, se c'è, si trova nella porzione compresa fra inf e i-1 (inclusi);

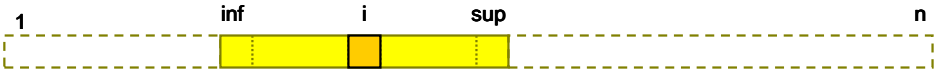
- **x > a[i]** x, se c'è, si trova nella porzione compresa fra i+1 e sup (inclusi);



Ricerca binaria in array ordinato

terzo caso:

- **x = a[i]** ho trovato il valore cercato, quindi termino la funzione restituendo true oppure l' indice **i**



Qual è la condizione per cui il ciclo deve continuare?

La porzione di array su cui fare la ricerca non deve essere vuota; cioè

inf <= sup

NOTA: se inf = sup la porzione non è vuota, ma contiene un elemento.

Quali sono i valori iniziali di inf e sup ?

Inizialmente la porzione di array su cui effettuare la ricerca è l'intero array.

inf := 1; sup := indice dell'ultimo elemento;

Ricerca binaria con risultato l'indice (in pseudo-Pascal)

```
function ricBin(x:integer; a:array): integer;
var inf, sup, i: integer;
begin
  inf := 1; sup := lunghezza di a;
  while(inf <= sup) {
    i := (inf + sup) div 2;
    if(x < a[i]) sup := i-1;
    else if(x > a[i]) inf := i+1;
    else fornisci_il_risultato i e termina
  }
  fornisci_il_risultato -1
}
```

Ricerca binaria con restituzione dell'indice: versione Java.

```
public static int ricercaBin(int x, int[] a) {
  int inf = 0, sup = a.length - 1;
  while(inf <= sup) {
    int i = (inf + sup)/2;
    if(x < a[i]) sup = i-1;
    else if(x > a[i]) inf = i+1;
    else return i;
  }
  return -1;
}
```


Complessità computazionale della ricerca binaria.

Ad ogni passo la porzione di array su cui effettuare la ricerca si divide per 2; se n è la lunghezza dell'array, il numero di iterazioni del ciclo è, nel caso peggiore, circa $\log_2 n$; ad esempio, se n = 1024, al secondo passo la porzione su cui ricercare sarà di dimensione 512, al terzo 256, e così via; dopo 10 passi si sarà ridotta a 1, poiché $1024 = 2^{10}$.

Tale caso peggiore si ha quando il valore da cercare non esiste, oppure quando viene trovato all'ultimo passo.

Raffinamento (versione Java)

Se il valore da ricercare è minore del primo elemento o maggiore dell'ultimo il metodo precedente compie $\log_2 n$ passi (dove n è la lunghezza dell'array).

È facile e conveniente ottimizzare questi due casi introducendo un test prima del ciclo.

```
public static int ricercaBin(int x, int[] a) {
  int inf = 0, sup = a.length - 1;
  if(x < a[0] || x > a[sup]) return -1;
  while(inf <= sup) {
    int i = (inf + sup)/2;
    if(x < a[i]) sup = i-1;
    else if(x > a[i]) inf = i+1;
    else return i;
  }
  return -1;
}
```


Slides supplementari.
Un elementare problema algoritmico:
risoluzione per mezzo dell'iterazione.

Ideare un ciclo a partire dal passo generico.

Problema. Dato un array di interi (di lunghezza ≥ 2), trovare gl'indici di due elementi (di valore non necessariamente distinto) che siano i due valori più grandi che compaiono nell'array.

Ideare un ciclo a partire dal passo generico.

Problema. Dato un array di interi (di lunghezza ≥ 2), trovare gl'indici di due elementi (di valore non necessariamente distinto) che siano i due valori più grandi che compaiono nell'array.

(in pseudo-Pascal:)

```
function dueMax(a: array_di_interi): coppia_di_interi;
var iMax, iVice, i: integer;
begin
```

PRECONDIZIONE: `a` è un array di interi ∧ `lunghezza_di_a` ≥ 2

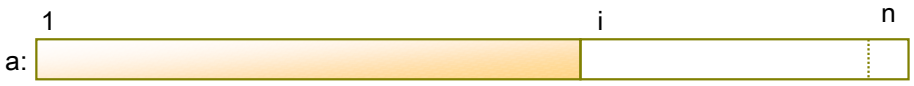
```
  ...
```

Risoluzione: Considero il generico passo:

la variabile i contiene l'indice del prossimo elemento da esaminare, l'array è stato esaminato dall'elemento di indice 1 fino all'elemento di indice i-1 compreso, e la situazione deve essere descritta dal seguente invariante:

SITUAZIONE AL PASSO GENERICO

a[iMax] e a[iVice] contengono risp. il massimo e il secondo valore più grande della parte esaminata dell'array

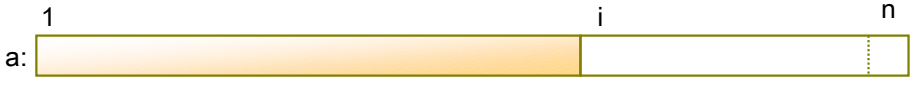


SITUAZIONE AL PASSO GENERICO

a[iMax] e a[iVice] contengono risp. il massimo e il secondo valore più grande della parte esaminata dell'array

cioè

a[iMax] contiene il massimo di a[1 .. i-1],
a[iVice] contiene il secondo valore più grande di a[1 .. i-1]



Scrittura del corpo del ciclo.

Per realizzare un programma corretto bisogna definire il corpo del ciclo in modo che esso "mantenga il significato delle variabili".
Esaminiamo il primo elemento non ancora esaminato a[i], e confrontiamolo con a[iMax] e a[iVice]; sono ovviamente possibili tre casi:

1. a[iMax] < a[i] : allora i è l'indice del nuovo massimo,
   mentre il valore di iMax è l'indice del nuovo vicemassimo;
   istruzioni:

```
iVice := iMax;
iMax := i;
```


Scrittura del corpo del ciclo.

Per realizzare un programma corretto bisogna definire il corpo del ciclo in modo che esso "mantenga il significato delle variabili".
Esaminiamo il primo elemento non ancora esaminato a[i], e confrontiamolo con a[iMax] e a[iVice]; sono ovviamente possibili tre casi:

1. a[iMax] < a[i] : allora i è l'indice del nuovo massimo,
   mentre il valore di iMax è l'indice del nuovo vicemassimo;
   istruzioni:

```
iVice := iMax;
iMax := i;
```

2. a[iVice] < a[i] ≤ a[iMax]: allora i è il nuovo vice-massimo:
   istruzione:

```
iVice := i;
```

Scrittura del corpo del ciclo.

Per realizzare un programma corretto bisogna definire il corpo del ciclo in modo che esso "mantenga il significato delle variabili".
Esaminiamo il primo elemento non ancora esaminato a[i], e confrontiamolo con a[iMax] e a[iVice]; sono ovviamente possibili tre casi:

1. a[iMax] < a[i] : allora i è l'indice del nuovo massimo,
   mentre il valore di iMax è l'indice del nuovo vicemassimo;
   istruzioni:
   ```
   iVice := iMax;
   iMax := i;
   ```

2. a[iVice] < a[i] ≤ a[iMax]: allora i è il nuovo vice-massimo:
   istruzione:
   ```
   iVice := i;
   ```

3. a[i] ≤ a[iVice] ≤ a[iMax]: massimo e vice-massimo sono invariati;

in tutti e tre i casi poi si incrementa `i`.

il ciclo completo

```
...
if a[1] >= a[2] then begin
  imax:= 1; ivice:= 2
end
else begin
  imax:= 2; ivice:= 1
end;

for i:= 3 to n do begin
  if a[i] > a[imax] then begin
    ivice := imax;
    imax:= i
  end
  else if a[i] > a[ivice] then ivice:= i
end;
...
```