Fondamenti di informatica
Oggetti e Java
Luca Cabibbo

Iterazione
(introduzione)

Capitolo 17 (estratto)
ottobre 2015

Contenuti

- Accesso a sequenze
- Accumulazione
- Conteggio
- Altre aggregazioni
- Verifica esistenziale
- Verifica universale

Iterazione

Questo capitolo presenta alcuni problemi comuni, e loro possibili soluzioni

- vengono considerate soluzioni iterative, ovvero basate sull'uso di istruzioni ripetitive

Accesso a sequenze

Questo capitolo mostra alcuni algoritmi fondamentali che possono essere utilizzati nell'elaborazione di sequenze – sequenze che possono essere di diversi tipi, ad es.

- sequenze immesse dall'utente – lette dalla tastiera
  - lunghezza nota
  - c'è un metodo che consente di sapere se ci sono altri elementi della sequenza che devono essere ancora letti
  - è noto l'ultimo elemento o una sua proprietà
- sequenze generate con una variabile di controllo
- sequenze ottenute scandendo gli elementi di una sequenza con accesso posizionale – ad es., caratteri di una stringa, elementi di un array o di una collezione

Il tipo di sequenza ha impatto sul modo con cui *scandire* gli elementi della sequenza

- in questo capitolo ci interessiamo soprattutto a come sia possibile *elaborare* gli elementi della sequenza

Accumulazione

Accumulazione

- lo scopo è calcolare una proprietà sintetica (cumulata) da una sequenza di valori – ad esempio
  - calcolo della somma di una sequenza di valori numerici
  - calcolo del fattoriale di un numero naturale

Tecnica dell'accumulazione

- una variabile di controllo (**accumulatore**) per calcolare l'informazione cumulativa
- un'**operazione di accumulazione**, binaria, che deve essere applicata all'accumulatore e a ciascun elemento della sequenza
- inizialmente all'accumulatore viene assegnato l'elemento neutro dell'operazione di accumulazione

Accumulazione

Calcola la somma di una sequenza di numeri

```
int numero;    // elemento corrente della sequenza
int somma;    // somma degli elementi della sequenza
Scanner in;    // per la lettura dalla tastiera

in = new Scanner ( System.in );

/* leggi una sequenza di numeri interi e
 * calcola la somma di questi numeri */
somma = 0;
while ( in.hasNextInt() ) {
    /* leggi un elemento della sequenza */
    numero = in.nextInt();
    /* somma il numero */
    somma = somma + numero;
}
/* visualizza somma */
System.out.println(somma);
```

Accumulazione

Calcola il fattoriale di un numero naturale

```
int n;         // un numero naturale
int nfatt;     // il fattoriale di n
int i;         // per iterare tra 1 e n
Scanner in;     // per la lettura dalla tastiera

in = new Scanner ( System.in );

/* leggi il numero naturale n */
n = in.nextInt();
/* calcola il fattoriale di n */
nfatt = 1;
i = 1;
while (i<=n) {
    nfatt = nfatt*i;
    i = i+1;
}
/* visualizza nfatt */
System.out.println(nfatt);
```


Accumulazione

Schema risolutivo

```
... dichiarazione della variabile accumulatore ...

accumulatore = elemento neutro dell'operazione
                di accumulazione ;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi o calcola il prossimo elemento ...
    accumulatore = applica l'operazione di accumulazione
                   all'accumulatore e all'elemento
                   corrente ;
}

... altre elaborazioni ...
```

- usare un nome opportuno per l'accumulatore

Invarianti di ciclo

In un'istruzione ripetitiva, un **invariante di ciclo** è una proprietà che risulta verificata

- ad ogni esecuzione del corpo dell'istruzione ripetitiva
- in un particolare punto del corpo dell'istruzione ripetitiva
  - in genere, all'inizio oppure alla fine

Nel caso della lettura e somma di una sequenza di numeri

- un invariante di ciclo è che **somma** è uguale alla somma degli elementi letti fino a quel momento dalla tastiera
- questa proprietà risulta verificata al termine di ciascuna esecuzione del corpo dell'istruzione ripetitiva

L'individuazione degli invarianti di ciclo è importante

- per comprendere/progettare algoritmi
- per motivare la correttezza di algoritmi iterativi

Conteggio

I problemi di **conteggio** sono un caso particolare dei problemi di accumulazione

- ad esempio
  - leggi una sequenza di numeri e calcola la sua lunghezza – conteggio
  - leggi una sequenza di numeri e calcola il numero degli elementi che valgono zero – conteggio condizionale
- l'accumulatore è un **contatore**
- l'operazione di accumulazione è un incremento unitario
- l'aggiornamento deve essere eseguito sempre (conteggio), oppure condizionatamente (conteggio condizionale) al soddisfacimento di una proprietà da parte dell'elemento corrente della sequenza

Conteggio

Calcola la lunghezza di una sequenza di numeri

```
int numero;    // elemento corrente della sequenza
int lunghezza;    // lunghezza della sequenza
Scanner in;    // per la lettura dalla tastiera

in = new Scanner ( System.in );

/* leggi una sequenza di numeri interi e
 * calcola la lunghezza della sequenza */
lunghezza = 0;
while ( in.hasNextInt() ) {
    /* leggi un elemento della sequenza */
    numero = in.nextInt();
    /* incrementa lunghezza */
    lunghezza++;
}
/* visualizza lunghezza */
System.out.println(lunghezza);
```


Conteggio condizionale

Calcola il numero di zeri in una sequenza

```
int numero;    // elemento corrente della sequenza
int zeri;    // numero di zeri della sequenza
Scanner in;    // per la lettura dalla tastiera

in = new Scanner ( System.in );

/* leggi una sequenza di numeri interi e
 * conta il numero di zeri nella sequenza */
zeri = 0;
while ( in.hasNextInt() ) {
    /* leggi un elemento della sequenza */
    numero = in.nextInt();
    /* se è uno zero, contalo */
    if (numero==0) {
        zeri++;
    }
}
/* visualizza zeri */
System.out.println(zeri);
```

Conteggio (condizionale)

Schema risolutivo

```
int contatore;    // numero di elementi che
                  // soddisfano la proprietà

contatore = 0;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi o calcola il prossimo elemento ...
    if ( l'elemento corrente soddisfa la proprietà ) {
        contatore++;
    }
}

... altre elaborazioni ...
```

- usare un nome opportuno per il contatore

Altre aggregazioni

In alcuni casi, bisogna calcolare una proprietà aggregata di una sequenza di numeri – ma questa proprietà aggregata non può essere calcolata mediante l'uso di un singolo accumulatore

- sono possibili diversi casi – ad esempio
  - leggi una sequenza di numeri e calcola la sua media – in questo caso vanno effettuate due accumulazione, e poi vanno combinate
  - leggi una sequenza di numeri non vuota e calcola l'elemento di valore minimo della sequenza – non si può usare un'accumulazione, perché l'operazione di "minimo" non ha un elemento neutro
- esistono pertanto più schemi algoritmici per i problemi di aggregazione

Calcolo della media – non basta un solo accumulatore

Calcola la media di una sequenza non vuota di numeri interi

```
int numero;    // elemento corrente della sequenza
int somma;    // somma degli elementi della sequenza
int lunghezza;    // lunghezza della sequenza
double media;    // media degli elementi della sequenza
Scanner in;    // per la lettura dalla tastiera

in = new Scanner ( System.in );

/* leggi una sequenza di numeri interi e
 * calcolane la somma e la lunghezza */
somma = 0;
lunghezza = 0;
while ( in.hasNextInt() ) {
    numero = in.nextInt();
    somma += numero;
    lunghezza++;
}
/* calcola (e visualizza) la media
media = 1.0*somma/lunghezza;
System.out.println(media);
```


Calcolo del minimo – non c'è elemento neutro

Calcola il minimo di una sequenza non vuota di numeri

```
int numero;    // elemento corrente della sequenza
int minimo;    // elemento di valore minimo della seq.
Scanner in;    // per la lettura dalla tastiera

in = new Scanner ( System.in );

/* leggi una sequenza di numeri interi e
 * calcolane il minimo */
/* leggi il primo elemento,
 * e consideralo il minimo provvisorio */
numero = in.nextInt();
minimo = numero;
/* leggi gli altri elementi della sequenza,
 * e calcola il vero minimo */
while ( in.hasNextInt() ) {
    numero = in.nextInt();
    if (numero<minimo) {
        minimo = numero;
    }
}
System.out.println(minimo);
```

Somma diminuita del massimo

Calcola della somma di una sequenza non vuota di numeri, diminuita del massimo

```
int numero;    // elemento corrente della sequenza
int somma;     // somma degli elementi della sequenza
int massimo;   // elemento di valore massimo della seq.
int sommaDiminuitaDelMassimo;   // somma – massimo
Scanner in;    // per la lettura dalla tastiera

in = new Scanner ( System.in );

... leggi una sequenza di numeri interi e
    calcola la somma e il massimo della sequenza ...

/* calcola la somma diminuita del massimo */
sommaDiminuitaDelMassimo = somma – massimo;
System.out.println(sommaDiminuitaDelMassimo);
```


Somma diminuita del massimo

Calcola della somma di una sequenza non vuota di numeri, diminuita del massimo

```
/* leggi una sequenza di numeri interi e
 * calcola la somma e il massimo della sequenza */
/* leggi ed elabora il primo elemento */
numero = in.nextInt();
massimo = numero;
somma = numero;    // e non: somma = 0 !
/* leggi gli altri elementi della sequenza,
 * e calcola il massimo e la somma */
while ( in.hasNextInt() ) {
    numero = in.nextInt();
    somma += numero;
    if (numero>massimo) {
        massimo = numero;
    }
}
```

Verifica esistenziale

I problemi di **verifica esistenziale** sono un altro caso particolare dei problemi di accumulazione

- bisogna determinare se una sequenza di elementi contiene almeno un elemento che soddisfa una certa proprietà
- come accumulatore viene usata una variabile booleana
  - indica se la sequenza contiene almeno un elemento che soddisfa la proprietà
- inizialmente si assegna alla variabile booleana un valore che indica convenzionalmente che la sequenza non contiene nessun elemento che soddisfa la proprietà (**false**)
- per ogni elemento della sequenza, si verifica se l'elemento corrente soddisfa la proprietà
  - se l'elemento corrente soddisfa la proprietà, allora si assegna alla variabile booleana un valore che indica convenzionalmente che la sequenza contiene almeno un elemento che soddisfa la proprietà (**true**)

Verifica esistenziale

Verifica se una sequenza contiene almeno uno zero

```
int numero;    // elemento corrente della sequenza
boolean contieneZero;    // la sequenza contiene
                         // almeno uno zero
Scanner in;    // per la lettura dalla tastiera

in = new Scanner ( System.in );

/* leggi una sequenza di numeri interi e
 * verifica se contiene almeno uno zero */
contieneZero = false;  // nessuno zero incontrato finora
while ( in.hasNextInt() ) {
    numero = in.nextInt();
    /* verifica se è uno zero */
    if (numero==0) {
        contieneZero = true;
    }
}
/* visualizza il risultato */
if (contieneZero) {
    System.out.println("Almeno uno zero");
} else {
    System.out.println("Nessuno zero");
}
```

Un errore comune

Verifica se una sequenza contiene almeno uno zero

```
/* leggi una sequenza di numeri interi e
 * verifica se contiene almeno uno zero */
contieneZero = false;  // nessuno zero incontrato finora
while ( in.hasNextInt() ) {
    numero = in.nextInt();
    /* verifica se è uno zero */
    if (numero==0) {
        contieneZero = true;
    } else {
        contieneZero = false;
    }
}
```

- che cosa fa questo frammento di codice?

Verifica esistenziale

Schema risolutivo

```
int contieneProprietà;    // almeno un elemento
                          // soddisfa la proprietà

contieneProprietà = false;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi al prossimo elemento ...
    if (l'elemento corrente soddisfa la proprietà) {
        contieneProprietà = true;
    }
}

... altre elaborazioni ...
```

- usare un nome opportuno per l'accumulatore

Verifica universale

Un problema di **verifica universale** consiste nel verificare se tutti gli elementi di una sequenza soddisfano una certa proprietà

- una variante (duale) dei problemi di verifica esistenziale

Un problema di verifica universale può essere sempre ricondotto a un problema di verifica esistenziale

- il problema diventa quello di verificare se **non** esiste nessun elemento della sequenza che **non** soddisfa la proprietà
- inizialmente si assegna alla variabile booleana un valore che indica convenzionalmente che tutti gli elementi della sequenza soddisfano la proprietà (**true**)
- per ogni elemento della sequenza, si verifica se l'elemento corrente **non** soddisfa la proprietà
  - se l'elemento corrente **non** soddisfa la proprietà, allora si assegna alla variabile booleana un valore che indica convenzionalmente che **non** tutti gli elementi della sequenza soddisfano la proprietà (**false**)

Verifica universale

Verifica se tutti gli elementi di una sequenza valgono zero

```
int numero;    // elemento corrente della sequenza
boolean tuttiZeri;    // la sequenza contiene solo zeri
Scanner in;    // per la lettura dalla tastiera

in = new Scanner ( System.in );

/* leggi una sequenza di numeri interi e
 * verifica se contiene solo zeri */
/* nessun elemento diverso da zero incontrato */
tuttiZeri = true;
while ( in.hasNextInt() ) {
    numero = in.nextInt();
    /* verifica se è DIVERSO da zero */
    if (numero!=0) {
        tuttiZeri = false;
    }
}
/* visualizza il risultato */
if (tuttiZeri) {
    System.out.println("Tutti zeri");
} else {
    System.out.println("Non tutti zeri");
}
```

Verifica universale

Schema risolutivo

```
int tuttiProprietà;     // tutti gli elementi
                        // soddisfano la proprietà

tuttiProprietà = true;
... altre inizializzazioni ...

per ciascun elemento della sequenza {
    ... accedi al prossimo elemento ...
    if (l'elemento corrente NON soddisfa la proprietà) {
        tuttiProprietà = false;
    }
}

... altre elaborazioni ...
```

- usare un nome opportuno per l'accumulatore