Gli array

Programmazione – Canale M-Z

LT in Informatica
9-10 Gennaio 2017

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Array (vettori)

Sono collezioni di valori dello stesso tipo a cui viene associato un unico nome simbolico

- i valori appartenenti all'array sono detti elementi dell'array;
- gli elementi dell'array vengono memorizzati in celle di memoria contigue

Motivazione:

Memorizzare un numero finito di elementi dello stesso tipo su cui bisogna operare in maniera uniforme

Esempio

Esempio:

scrivere un programma che legge 20 interi e stampa il valore più vicino alla loro media

Soluzione senza array:

- definire 20 variabili di tipo `int` che memorizzano gli input
- calcolare la media
- stampare la variabile che è più vicina alla media (useremo la funzione `abs` per calcolare la distanza in valore assoluto)

Esempio – soluzione senza array


```
int main() {
    int var1, . . . , var20 ;
    int r;
    float dist, m ;
    cin >> var1 ; . . . ; cin >> var20 ;
    m = (var1 + . . . + var20)/20.0 ;
    r = var1 ;
    dist = abs(m - var1) ;
    if (abs(m-var2) < dist) {
        dist = abs(m-var2) ;
        r = var2 ;
    }
    ...
    if (abs(m-var20)< dist) {
        dist = abs(m-var20) ;
        r = var20 ;
    }
    cout << "Il valore piu' vicino alla media e': " << r <<
    endl;
}
```

Esempio – osservazioni

Osservazione

- sulle 20 variabili ripetiamo sempre le stesse operazioni
- si agisce in modo uniforme

Nuovo problema

Risolvere il problema precedente con 20.000 valori in input:

- bisogna allungare il codice con le nuove variabili, i nuovi input, e i nuovi `if`
- per ovviare a questi problemi si usano gli array

Dichiarazioni di array

Sintassi:

```
tipo nome[dimensione] ;
```

- `tipo` è il tipo di base degli elementi
- `dimensione` è il numero di elementi dell'array

Semantica:

- viene allocato spazio in memoria per contenere l'array il cui identificatore è `nome`, costituito da `dimensione` elementi
- ogni elemento contiene un valore del tipo di base (`int`, `float`, `char`, ...)

Accesso agli elementi di un array

- ogni elemento contiene un valore del tipo di base (`int`, `float`, `char`, ...)
- gli elementi sono numerati da `0` a `dimensione - 1`
- si accede ai singoli elementi mediante i termini `nome[0]`, `nome[1]`, ... `nome[dimensione-1]`
- il contenuto delle parentesi `[.]` è detto indice
- un indice è una espressione di tipo `int`

Esempio: `a[i + 1]`

Semantica: al tempo di esecuzione viene valutato l'indice. Il valore ottenuto determina a quale elemento dell'array ci riferiamo

Assegnare gli elementi di un array

Per assegnare un valore ad un elemento di un array si sua l'operazione di assegnamento:

```
nome[indice] = espressione;
```

Esempio:

```
int a[10], n = 2 ;
a[n+2] = 35 ;
```

assegna all'elemento `a[4]` il valore 35

Assegnamenti e confronti tra array

- Non è possibile fare assegnamenti tra array!

```
int a[5], b[5];
a = b;                  // NON CONSENTITO
```

- I confronti tra array non danno i risultati attesi!

```
char s1[10],s2[10];
if(s1 == s2) {   // IL TEST E' SEMPRE FALSO
     ...
}
```

Per svolgere queste operazioni è necessario operare elemento per elemento

Accesso sequenziale agli array

- è spesso necessario elaborare gli elementi di un array in sequenza, partendo dal primo elemento
- di solito si utilizza un ciclo `for`, la cui variabile di controllo viene usata come indice dell'array

Esempi:

- inizializzazione del contenuto dell'array

```
int a[10];
for (int i = 0; i < 10; i = i+1)
    cin >> a[i] ;
```

il primo indice è `i = 0`

l'ultimo indice è `i = dimensione-1`

- somma degli elementi

```
int sum = 0 ;
for (int i = 0; i < 10; i = i+1)
    sum = sum + a[i] ;
```

Valore più vicino alla media


```cpp
#include <iostream>
#include <cmath>

using namespace std;

// PRE: cin contiene 20 valori interi v_1, ..., v_20
int main() {
    int var[20], r;
    float dist, m = 0;

    for(int i=0 ;i < 20; i=i+1) {
        cin >> var[i];
        m = m + var[i];
    }
    m = m/20.0;
    r = var[0] ;
    dist = abs(m - var[0]);
    for (int i = 1; i < 20 ; i = i+1) {
        if (abs(m-var[i]) < dist){
            dist = abs(m-var[i]);
            r = var[i];
        }
    }
    cout << "Il valore piu' vicino alla media e': " << r << endl;
}
// POST: r contiene il valore v_k piu' vicino alla media
```

Esercizio: cercare un elemento in un array

Esercizio

Scrivere un programma che prende in input un array di 10 interi `a`, un valore intero `x` e determina se `x` si trova in `a` oppure no.

Per risolvere questo esercizio dobbiamo:

- scorrere l'array partendo dalla prima posizione
- appena troviamo `x`, possiamo terminare con successo
- se siamo arrivati alla fine, allora `x` non appartiene ad `a`

Ricerca Lineare Incerta

La tecnica usata per risolvere l'esercizio si chiama **Ricerca Lineare Incerta**:

Ricerca perché dobbiamo cercare il valore `x` all'interno dell'array

Lineare perché scorriamo l'array un elemento per volta dall'inizio alla fine

Incerta perché non è detto che `x` sia presente nell'array

E' una tecnica molto semplice ma allo stesso tempo molto potente:

- molti problemi con gli array si risolvono usando varianti della ricerca lineare incerta

Codice per la Ricerca Lineare Incerta


```
#include <iostream>

using namespace std;
// PRE = cin contiene 10 valori interi a_1, ..., a_10
//       seguiti da un valore intero x
int main() {
    int a[10];
    int x;
    bool trovato = false;

    for(int i = 0; i < 10; i++) {
        cin >> a[i];
    }
    cin >> x;

    for(int i = 0; i < 10 && !trovato; i++) {
        if(a[i] == x) {
            trovato = true;
        }
    }
    if(trovato) {
        cout << x << " e' presente nell'array" << endl;
    } else {
        cout << x << " non e' presente nell'array" << endl;
    }
}
// POST = stampa "x e' presente nell'array" se x e' uguale a uno dei valori a_i,
//        "x non e' presente nell'array" altrimenti
```

Un invariante per la ricerca lineare incerta

Cerchiamo un invariante per il secondo ciclo for

```
R = (i <= 10 && trovato se e solo se esiste j < i tale
 che x == a[j])
```

1. Rispetta la condizione iniziale?
   - ✓ Si: dopo l'inizializzazione `int i = 0` abbiamo che `i = 0 <= 10` e `trovato` è `false`. Siccome gli indici dell'array partono da 0, non esiste nessun `a[j] == x`.
2. Rispetta l'invarianza?
   - ✓ Si: all'inizio di ogni iterazione `i < 10` e `trovato` è `false`, quindi non abbiamo ancora trovato `x`. Se `a[i] == x` allora `trovato` diventa `true`, altrimenti rimane `false`.

Un invariante per la ricerca lineare incerta

Cerchiamo un invariante per il secondo ciclo for

```
R = (i <= 10 && trovato se e solo se esiste j < i tale
 che x == a[j])
```

3. Rispetta la condizione di uscita?
   - ✓ Si. L'invariante ci dice che `i <= 10`. All'uscita dal ciclo ci sono due casi: siamo usciti perché `trovato` è `true` oppure perché `i >= 10` (negazione della guardia). Nel primo caso, dall'invariante sappiamo che esiste `j < 10` tale che `x == a[j]`. Nel secondo caso sempre l'invariante ci dice che non c'è nessun elemento dell'array uguale a `x`.
     L'`if` finale fa stampare il messaggio giusto.

Ordinare un array

Ordinare una lista di valori è una operazione molto comune

- creare una ordinamento di studenti in ordine alfabetico
- ordinare in maniera crescente
- ordinare in maniera decrescente

Ci sono molti modi per ordinare un array:

- alcuni sono semplici da comprendere
- altri sono molto efficienti computazionalmente
- esempi: selection-sort, bubble-sort, quicksort, mergesort, ...

Un modo semplice per ordinare un array

Osservazione:

In un array ordinato abbiamo che

```
a[0] <= a[1] <= . . . <= a[dimensione-1]
```

Ciò porta a un algoritmo molto semplice:

```
for (int i = 0 ; i < dim ; i = i+1) {
    // metti in a[i] il valore piu' piccolo
    // della porzione a[i ... dim-1]
}
```

Selection-sort

Selection-sort è uno degli algoritmi di ordinamento più semplici:

- ricerca il più piccolo elemento nell'array `a`: sia `min` il suo indice
- sostituisci `a[0]` con `a[min]`
- ricerca il più piccolo valore nell'array partendo da `a[1]`: sia `min` il suo indice
- sostituisci `a[1]` con `a[min]`
- ricerca il più piccolo valore nell'array partendo da `a[2]`: sia `min` il suo indice
- sostituisci `a[2]` con `a[min]`
- ...

Il codice di Selection-sort


```
#include <iostream>
using namespace std;

int main() {
    const int dim = 10;
    int a[dim];
    int min, tmp;

    for(int i = 0; i < dim; i++) {
        cin >> a[i];
    }
    // PRE = a e' un array di dim interi
    for(int i = 0; i < dim - 1; i ++) {
        // R = (a e' permutazione di a originale &&
        //      a[0 ... i-1] e' ordinato e <= a[i ... dim-1])
        min = i;
        for(int j = i + 1; j < dim; j++) {
            if(a[j] < a[min]) {
                min = j;
            }
        }
        tmp = a[i];
        a[i] = a[min];
        a[min] = tmp;
    }
    // POST = a e' un vettore ordinato di dim interi
    for(int i = 0; i < dim; i++) {
        cout << a[i] << "\t";
    }
    cout << endl;
}
```

Un invariante per Selection-sort

Cerchiamo un invariante per il ciclo for principale

```
R = (a permutazione di a originale && a[0 ... i-1]
ordinato e <= a[i ... dim-1]
```

1. Rispetta la condizione iniziale?
   - ✓ Si: dopo l'inizializzazione a non è ancora stato modificato. Siccome gli indici dell'array partono da `0`, `a[0 ... -1]` è un array con zero elementi, quindi ordinato e `<= a[0 ... dim-1]`.
2. Rispetta l'invarianza?
   - ✓ Si: dopo ogni iterazione il minimo della porzione `a[i ... dim-1]` viene scambiato con `a[i]`. Quindi `a` contiene gli stessi valori dell'originale, `a[0 ... i]` diventa ordinato e `<= a[i+1 ... dim-1]`

Un invariante per Selection-sort

Cerchiamo un invariante per il ciclo for principale

```
R = (a permutazione di a originale && a[0 ... i-1]
ordinato e <= a[i ... dim-1]
```

3. Rispetta la condizione di uscita?
   - ✓ Si. La negazione della guardia ci dice che `i >= dim-1`. L'invariante ci dice che la porzione `a[0 ... i-1]` è ordinata e `<= a[i+1 ... dim-1]`, quindi che anche `a[0 ... dim-1]` è ordinato.

Ricerca in un array ordinato

Per cercare un elemento `x` in un array ordinato `a` si usa la ricerca binaria (simile al metodo utilizzato per cercare un numero nell'elenco telefonico):

1. si accede all'elemento memorizzato a metà di `a` e si verifica se è uguale a `x` oppure no
2. se è uguale a `x` il programma termina con successo
3. altrimenti si sceglie la metà appropriata di `a` e si riparte da 1
4. se non posso più dividere a metà l'array allora `x` non è presente

Ricerca binaria

Implementazione della ricerca binaria:

1. si utilizzano due indici `l` (sta per left) e `r` (sta per right) che memorizzano gli estremi della porzione di array su cui si deve cercare la presenza di `x` (`l<=r`)
2. all'inizio `l = 0` e `r = dim-1`
3. si accede all'elemento a metà della porzione di array tra `l` e `r`
   - dove si trova questo elemento? In posizione `(l+r)/2`
4. se `a[(l+r)/2] == x` allora abbiamo trovato l'elemento
5. se `a[(l+r)/2] > x` allora bisogna cercare nella parte sinistra
   - si itera (si ritorna al passo 1) con `r = (l+r)/2 - 1`
6. se `a[(l+r)/2] < x` allora bisogna cercare nella parte destra
   - si itera (si ritorna al passo 1) con `l = (l+r)/2 + 1`

Codice della ricerca binaria


```
#include <iostream>
using namespace std;
// PRE = cin contiene 10 valori interi ordinati a_1 <= ... <= a_10
//       seguiti da un valore intero x
int main() {
    const int dim = 10;
    int a[dim], x;
    for(int i = 0; i < 10; i++) {
        cin >> a[i];
    }
    cin >> x;
    bool trovato = false;
    int l = 0, r = dim-1, m;
    while(!trovato && l <= r) {
        m = (l + r) / 2;
        if(a[m] == x) {
            trovato = true;
        } else if(a[m] > x) {
            r = m - 1;
        } else {
            l = m + 1;
        }
    }
    if(trovato) {
        cout << x << " e' presente nell'array" << endl;
    } else {
        cout << x << " non e' presente nell'array" << endl;
    }
} // POST = stampa "x e' presente nell'array" se x e' uguale a uno degli a_i,
  //        "x non e' presente nell'array" altrimenti
```