

Algoritmi di ordinamento

Indice

1	Funzioni utili per entrambe le versioni dell'algoritmo	1
2	<i>selection-sort</i>	2
2.1	Esempio con vari spostamenti	3
3	<i>bubble-sort</i>	4
3.1	Versione ottimizzata dell'algoritmo	5
3.2	Esempio con vari spostamenti - <i>non ottimizzato</i>	6

Introduzione

L'ordinamento dei vettori è una questione estremamente dibattuta: esistono moltissimi algoritmi ed è continua la ricerca di nuovi più efficienti e veloci. Il professore ha fatto notare che l'argomento è una delle delle domande più gettonate negli esami di stato per l'iscrizione all'Albo degli ingegneri.

Di seguito parleremo dei due algoritmi introdotti a *Fondamenti di programmazione*.

1 Funzioni utili per entrambe le versioni dell'algoritmo

```
/* "Per stampare il contenuto del vettore prima e dopo aver effettuato l
'ordinamento. Indico vettore in questione e numero di elementi
presenti nel vettore" */
void stampa(const int v[], int n) {
    if (n != 0) {
        cout << "[" << v[0];
        for (int i = 1; i < n; i++) {
            cout << ' ' << v[i];
            cout << ']' << endl;
        }
    }
}

/* "Scambio la posizione di due elementi all'interno del vettore. Indico
vettore in questione, e indici delle due posizioni." */
void scambia(int vettore[], int x, int y) {
    int lavoro = vettore[x]; // "Variabile ausiliaria"
    vettore[x] = vettore[y];
    vettore[y] = lavoro;
}
```

2 *selection-sort*

Nell'algoritmo *selection-sort* si svolge ciclicamente la ricerca del minimo riducendo progressivamente l'area di indagine all'interno del vettore.

1. Con la prima ricerca (che coinvolge tutti gli elementi del vettore) trovo il minimo e lo sposto nella prima cella (se non già lì) scambiandolo con il numero presente.
2. Dopo aver svolto la prima ricerca restringo l'area di indagine ai numeri rimanenti (sono certo che il numero appena spostato sia nel posto giusto, quindi non ho motivo di coinvolgerlo ulteriormente).
3. Svolgo nuovamente la ricerca del minimo tra i numeri rimanenti: lo trovo ed effettuo nuovamente lo scambio con il numero presente nella seconda cella, se necessario.
4. Ripeto gli stessi passaggi finchè l'area di indagine non sarà ridotta a un solo elemento ($i < n - 1$). A quel punto ho la certezza che tutti i numeri siano effettivamente ordinati dal più piccolo al più grande!

La complessità dell'algoritmo ha *ordine* n^2 , con n che consiste nel numero di elementi presenti nel vettore da ordinare.

Controllare le slides 283, 284 e 285 Presente immagine esplicativa dello spostamento.

```
// "Indico vettore da ordinare e num. di elementi del vettore"
void selectionSort(int vettore[], int n) {
    int min; // "Non mi interessa l'ultima casella (n-1)"
    for (int i = 0 ; i < n-1; i++) {
        /* "Ogni volta la posizione in cui spostare il
           valore minimo (indicata dalla variabile i)
           aumenta di uno (i++)" */
        min = i;
        /*"Trovo il minimo. Notare int j = i+1: parto
           sempre dalla cella successiva a quella in cui e
           ' stato collocato poco prima il minimo" */
        for (int j = i + 1; j < n; j++) {
            if (vettore[j] < vettore[min])
                min = j;
        }
        // "Scambio min. trovato con num. in posiz. i"
        scambia(vettore,i,min);
    }
}

int main() { // "Esempio di vettore da ordinare"
    int v[] = {2, 26, 8, 2, 23};
    selectionSort(v, 5);      stampa(v, 5);

    return 0;
}
```

2.1 Esempio con vari spostamenti

```
Inizio cosi':  
[ 560 54 129 63 12 89 56 99 110 230 ]  
  
Il minimo e' 12. Lo spostato da posizione 4 (min) a posizione 0 (i)  
Prima: [ *560* 54 129 63 *12* 89 56 99 110 230 ]  
Dopo:  [ *12* 54 129 63 *560* 89 56 99 110 230 ]  
  
Il minimo e' 54. Lo spostato da posizione 1 (min) a posizione 1 (i)  
Prima: [ 12 *54* 129 63 560 89 56 99 110 230 ]  
Dopo:  [ 12 *54* 129 63 560 89 56 99 110 230 ]  
  
Il minimo e' 56. Lo spostato da posizione 6 (min) a posizione 2 (i)  
Prima: [ 12 54 *129* 63 560 89 *56* 99 110 230 ]  
Dopo:  [ 12 54 *56* 63 560 89 *129* 99 110 230 ]  
  
Il minimo e' 63. Lo spostato da posizione 3 (min) a posizione 3 (i)  
Prima: [ 12 54 56 *63* 560 89 129 99 110 230 ]  
Dopo:  [ 12 54 56 *63* 560 89 129 99 110 230 ]  
  
Il minimo e' 89. Lo spostato da posizione 5 (min) a posizione 4 (i)  
Prima: [ 12 54 56 63 *560* *89* 129 99 110 230 ]  
Dopo:  [ 12 54 56 63 *89* *560* 129 99 110 230 ]  
  
Il minimo e' 99. Lo spostato da posizione 7 (min) a posizione 5 (i)  
Prima: [ 12 54 56 63 89 *560* 129 *99* 110 230 ]  
Dopo:  [ 12 54 56 63 89 *99* 129 *560* 110 230 ]  
  
Il minimo e' 110. Lo spostato da posizione 8 (min) a posizione 6 (i)  
Prima: [ 12 54 56 63 89 99 *129* 560 *110* 230 ]  
Dopo:  [ 12 54 56 63 89 99 *110* 560 *129* 230 ]  
  
Il minimo e' 129. Lo spostato da posizione 8 (min) a posizione 7 (i)  
Prima: [ 12 54 56 63 89 99 110 *560* *129* 230 ]  
Dopo:  [ 12 54 56 63 89 99 110 *129* *560* 230 ]  
  
Il minimo e' 230. Lo spostato da posizione 9 (min) a posizione 8 (i)  
Prima: [ 12 54 56 63 89 99 110 129 *560* *230* ]  
Dopo:  [ 12 54 56 63 89 99 110 129 *230* *560* ]
```

Nota In alcuni casi non avviene spostamento ($min = i$)

3 *bubble-sort*

Nell'algoritmo *bubble-sort* si scorre $n - 1$ volte l'intero array, da destra verso sinistra, analizzando tutte le possibili coppie contigue di numeri: verifico in ciascuna coppia che il numero più a sinistra sia più piccolo del numero più a destra, in caso contrario li scambio con l'apposita funzione. Al termine avrò tutti gli elementi del vettore ordinati dal più piccolo al più grande!

La complessità dell'algoritmo ha *ordine* $\mathbf{n^2}$, con n che consiste nel numero di elementi presenti nel vettore da ordinare.

Controllare le slides 286, 287, 288, e 289 Presenti immagini esplicative dello spostamento.

```
// "Negli argomenti ho il vettore che desidero ordinare, e il
    numero di elementi del vettore"
void bubble(int vettore[], int n) {
    // "Svolgo n-1 scorrimenti dell'array"
    for (int i = 0 ; i < n-1; i++) {
        // "Svolgo lo scorrimento dell'array da destra
        verso sinistra, notare j--"
        for (int j = n-1; j > i; j--) {
            // "Confronto le coppie contigue."
            // "Se l'elemento piu' a dx e' maggiore
            dell'elemento piu' a sx effettuo lo
            scambio"
            if (vettore[j] < vettore[j-1])
                scambia(vettore, j, j-1);
        }
    }
}
```

3.1 Versione ottimizzata dell'algoritmo

Posso ottimizzare il vettore riducendo il numero di controlli: in certe situazioni potrei avere, già con poche mosse, tutti gli elementi ordinati correttamente!

```
void bubble(int vettore[], int n) {
    // "Pongo una variabile booleana"
    bool ordinato = false;

    /* "Aggiungo una condizione nel for: che l'algoritmo si
       blocchi nel caso in cui si abbia ordinato = true" */
    for (int i = 0 ; i < n-1 && !ordinato; i++) {
        /* "Ogni volta che ricomincio lo scorrimento dell'
           array pongo ordinato = true" */
        ordinato = true;

        for (int j = n-1; j >= i+1; j--) {
            if(vettore[j] < vettore[j-1]) {
                scambia(vettore, j, j-1);
                /* "ordinato = false se viene
                   effettuato uno spostamento" */
                ordinato = false;
            }
        }
    }

    // "Se ordinato = true rimane uguale per tutto lo
    // scorrimento dell'array significa che non ho effettuato
    // spostamenti"
    // "Se non ho effettuato spostamenti significa che gli
    // elementi dell'array risultano già ordinati."
    // "Pertanto non ho motivo per effettuare gli scorrimenti
    // rimanenti"
}

int main() { // "Esempio di vettore"
    int v[] = {2, 1, 3, 4, 5};
    bubble(v, 5);      stampa(v, 5);

    /* "Con l'algoritmo non ottimizzato svolgo 4 iterazioni,
       con l'algoritmo ottimizzato ne svolgo 2: nella prima
       effettuo solo lo scambio tra 2 e 1, nella seconda non
       effettuo spostamenti. La variabile ordinato mantiene
       valore true per tutta la seconda iterazione,
       conseguentemente rendo falsa la condizione nel for e
       fermo il ciclo." */
    return 0;
}
```

3.2 Esempio con vari spostamenti - *non ottimizzato*

```
Inizio cosi':
[ 54 129 63 12 89 56 ]

1: 56 (j) e' minore di 89 (j-1)
Dopo:  [ 54 129 63 12 *56* *89*]

2: 56 (j) e' maggiore di 12 (j-1)
Non eseguo spostamenti

3: 12 (j) e' minore di 63 (j-1)
Dopo:  [ 54 129 *12* *63* 56 89 ]

4: 12 (j) e' minore di 129 (j-1)
Dopo:  [ 54 *12* *129* 63 56 89 ]

5: 12 (j) e' minore di 54 (j-1)
Dopo:  [ *12* *54* 129 63 56 89 ]

-----
6: 89 (j) e' maggiore di 56 (j-1)
Non eseguo spostamenti

7: 56 (j) e' minore di 63 (j-1)
Dopo:  [ 12 54 129 *56* *63* 89 ]

8: 56 (j) e' minore di 129 (j-1)
Dopo:  [ 12 54 *56* *129* 63 89 ]

9: 56 (j) e' maggiore di 54 (j-1)
Non eseguo spostamenti

-----
10: 89 (j) e' maggiore di 63 (j-1)
Non eseguo spostamenti

11: 63 (j) e' minore di 129 (j-1)
Dopo:  [ 12 54 56 *63* *129* 89 ]

12: 63 (j) e' maggiore di 56 (j-1)
Non eseguo spostamenti

-----
13: 89 (j) e' minore di 129 (j-1)
Dopo:  [ 12 54 56 63 *89* *129*]

14: 89 (j) e' maggiore di 63 (j-1)
Non eseguo spostamenti

-----
15: 129 (j) e' maggiore di 89 (j-1)
Non eseguo spostamenti

-----
```