Laboratorio di Algoritmi e Strutture Dati

Esercitazioni del 30 Ottobre 2012

Esercizio 3: algoritmi di ordinamento elementari

L'esercizio consiste nell'implementazione di alcuni dei più semplici algoritmi per l'ordinamento di array di interi.

Tutti questi algoritmi hanno complessità quadratica nella lunghezza dell'array, e dunque nessuno di questi algoritmi ha prestazioni ottimali.

Si richiede di implementare le seguenti funzioni:

- `void leggiarray(FILE *fp, int *a, int n)`
  Legge `n` interi dal file di testo associato a `fp`, e li pone nell'array `a` che deve essere un array di `n` interi.
  Si suppone che il file da leggere sia conforme alle specifiche, cioè contenga `n` interi (scritti come stringhe di cifre decimali) separati da caratteri di spaziatura.
  Per la lettura di ognuno di questi interi, usate `fscanf`.

- `void stampaarray(int *a, int n)`
  Stampa gli elementi dell'array `a`, ordinatamente, da `a[0]` a `a[n-1]`.

- `void copiaarray(int *a, int *b, int n)`
  Assumendo che sia `a` che `b` siano array di `n` interi, copia l'array `a` nell'array `b`.

- `void bubblesort(int *a, int n)`
  Ordina l'array `a`, di `n` elementi, mediante l'algoritmo `bubblesort`:
  All'i-esima iterazione del ciclo più esterno (partendo da $i = 0$) pone l'i-esimo elemento più piccolo in posizione `a[i]`.
  Per ottenere questo scopo scandisce l'array dal suo estremo destro: se l'elemento corrente è minore del precedente, allora li scambia.

- `void selectionsort(int *a, int n)`
  Ordina l'array `a`, di `n` elementi, mediante l'algoritmo `selectionsort`:
  All'i-esima iterazione del ciclo più esterno (partendo da $i = 0$) pone l'i-esimo elemento più piccolo in posizione `a[i]`.
  Per ottenere questo scopo cerca l'elemento minimo nella porzione di array ancora da sistemare. Quando lo trova, lo scambia con l'elemento in posizione `a[i]`.

- `void insertionsort(int *a, int n)`
  Ordina l'array `a`, di `n` elementi, mediante l'algoritmo `insertionsort`:
  All'i-esima iterazione del ciclo più esterno (partendo da $i = 0$) sistema l'$i + 1$-esimo elemento in modo tale che il sottoarray `a[0],a[1],...,a[i+1]` sia ordinato.

Introducete la macro:

```
#define DIM   200
```

Il programma deve:

1. Leggere `DIM` interi dal file `numeri.txt` all'array `a`, locale a `main`.

2. Copiare l'array `a` nell'array `b`, anch'esso locale a `main`.
3. Ordinare l'array `b` con `bubblesort` e stampare l'array `b` ordinato.
4. Copiare l'array `a` nell'array `b`.
5. Ordinare l'array `b` con `selectionesort` e stampare l'array `b` ordinato.
6. Copiare l'array `a` nell'array `b`.
7. Ordinare l'array `b` con `insertionsort` e stampare l'array `b` ordinato.

Supplemento

Introducete la seguente definizione del tipo `struct counts`:

```
struct counts {
    int cfrs, swaps;
};
```

Una variabile `struct counts conti` è una collezione di due variabili intere: `conti.cfrs` e `conti.swaps`.

Modificate i prototipi di `bubblesort`, `selectionsort`, `insertionsort` in modo che il tipo di ritorno sia `struct counts`.

Modificate inoltre il codice di queste funzioni in modo che il valore di ritorno di queste funzioni sia tale che:

- Il suo campo `cfrs` contenga il numero di confronti *fra elementi dell'array* effettuati.
- Il suo campo `swaps` contenga il numero di scambi *fra elementi dell'array* effettuati.

Modificate il programma in modo tale che dopo i punti 3,5,7, stampi il messaggio:

algoritmo. `Totale confronti:` $cfrs$. `Totale scambi:` *swaps*

dove *algoritmo* è il nome dell'algoritmo appena usato per l'ordinamento, ed è dunque uno fra `bubblesort, selectionsort, insertionsort`, $cfrs$ è il numero di confronti effettuati dall'algoritmo, *swaps* è il numero di scambi effettuati dall'algoritmo.

Facoltativo

Si noti che ogni scambio richiede tre assegnamenti *fra elementi dell'array e/o con variabili di appoggio* che memorizzano temporaneamente valori dell'array.

Alcuni degli algoritmi proposti (in particolare `insertionsort`) possono essere implementati usando, oltre a un certo numero di scambi, anche semplici assegnamenti fra elementi dell'array (istruzioni del tipo `a[i] = a[j]`).

Modificate `struct counts` aggiungendovi un campo `int assignments`.

Modificate i codici degli algoritmi di ordinamento in modo tale che il valore di ritorno nel suo campo `swaps` contenga precisamente il numero di scambi, mentre nel suo campo `assignments` il numero di assegnamenti semplici che non fanno parte di scambi.

Modificate il messaggio precedente in modo tale che visualizzi anche il `Totale assegnamenti`.

N.B. Non contate fra gli assegnamenti le istruzioni che modificano il valore di indici, o di altre variabili ausiliarie.