Programmazione avanzata ed elementi di Ingegneria del Software

Ordinamento

BubbleSort

Ordinamento

Ci riferiamo agli elementi di un array di interi, ma gli algoritmi si applicano a casi più generali.

Deve essere definita una **relazione d'ordine** fra gli elementi.

NB:
Questo significa che occorre risolvere esercizi del tipo: "data una struttura contenente nome, cognome, anno di nascita, reddito, … ecc, ed una array di tali strutture, ordinare l'array utilizzando il criterio 'anno di nascita' o 'reddito' a seconda di quanto richiesto dall'utente"

Bubble Sort

Ogni coppia di elementi adiacenti viene comparata e invertita di posizione se sono nell'ordine sbagliato.

Fonte immagine: https://it.wikipedia.org/wiki/Bubble_sort

"Because it is not very good relative to other methods and is the one typically stumbled on by naive and untutored programmers, hackers consider it the canonical example of a **naive algorithm**."
http://www.jargon.net/jargonfile/b/bubblesort.html

Bubble Sort

6 5 3 1 8 7 2 4

Bubble Sort

```
int main() {
  int vett[6]={9,18,7,15,21,11};
  int nEl=6;
  printArray(vett, nEl);
  bubbleSort_1(vett, nEl);
  printArray(vett, nEl);
  getchar();
  return 0;
}
```

Bubble Sort

```c
void bubbleSort_1(int* v, int n){
  int tmp;
  for (int j=0; j<n-1; j++){
    for (int i=0; i<n-1; i++){
      if (v[i]>v[i+1]){
        // SWAP
        tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
      }
```

```
[9 18 7 15 21 11 ]
iter: 0
iter: 1
iter: 2
iter: 3
iter: 4
[7 9 11 15 18 21 ]
```

Quando un intero ciclo interno non ha dato luogo a swap, l'array è ordinato.
Possiamo migliorare l'implementazione.

bubbleSort_1.c

Bubble Sort

Ottimizzazione:
quando non avvengono più swap per un intero ciclo esterno, l'array è ordinato. Posso interrompere la procedura di ordinamento.

Bubble Sort

```
void bubbleSort_2a(int* v, int n){
  int tmp;
  bool continua;
  for (int j=0; j<n-1; j++){
    printf("iter: %d \n",j);
    continua=false;
    for (int i=0; i<n-1; i++){
      if (v[i]>v[i+1]){ // SWAP
        continua=true;
        tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
      }
    }
   if (continua==false) break;
```

```
[9 18 7 15 21 11 ]
iter: 0
iter: 1
iter: 2
iter: 3
[7 9 11 15 18 21 ]
```

bubbleSort_2.c

Bubble Sort

```
void bubbleSort_2b(int* v, int n){
  int tmp,j=0;
  bool continua;
  do {
    printf("iter: %d \n",j++);
    continua=false;
    for (int i=0; i<n-1; i++){
      if (v[i]>v[i+1]){ // SWAP
        continua=true;
        tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
      }
    }
  }
  while (continua);
```

```
[9 18 7 15 21 11 ]
iter: 0
iter: 1
iter: 2
iter: 3
[7 9 11 15 18 21 ]
```

bubbleSort_2.c

Bubble Sort

Ottimizzazione: il ciclo **interno** sposta di volta in volta l'elemento più 'pesante' verso il basso.
Dunque ad ogni iterazione del ciclo **esterno** (j), gli ultimi (j+1) elementi del vettore sono ordinati!

Dopo l'iterazione j=0,
gli ultimi j+1=1 elementi dell'array sono ordinati.

Dopo l'iterazione j=1,
gli ultimi j+1=2 elementi dell'array sono ordinati.

...

Bubble Sort

Ottimizzazione:
ad ogni iterazione del ciclo **esterno** (j), gli ultimi (j+1) elementi del vettore sono ordinati.

Il ciclo **interno** dunque va
da i=0 a i<n-1-j
invece che
da i=0 a i<n-1

Bubble Sort

```
void bubbleSort_3a(int* v, int n){
  int tmp;
  bool continua;
  for (int j=0; j<n-1; j++){
    continua=false;
    for (int i=0; i<n-1-j; i++){
      if (v[i]>v[i+1]){ // SWAP
        continua=true;
        tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
      }
    }
   if (continua==false) break;
```

bubbleSort_3.c

Bubble Sort

```
void bubbleSort_3b(int* v, int n){
  int tmp,j=0;
  bool continua;
  do {
    continua=false;
    for (int i=0; i<n-1; i++){
      if (v[i]>v[i+1]){ // SWAP
        continua=true;
        tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
      } // end IF
    } // end FOR
    n=n-1;
  }
  while (continua && n>1);
```

bubbleSort_3.c

Bubble Sort

Se il **ciclo interno** effettua l'ultimo spostamento fra gli indici *i* e *i+1*, **non è più necessario controllare gli elementi da i in poi.**

Bubble Sort

Se il **ciclo interno** effettua l'ultimo spostamento fra gli indici *i* e *i+1*, **non è più necessario controllare gli elementi da i in poi.**

i=0; scambio 0 con 1

i=1; scambio 1 con 2

non devo più controllare i=1
i=0 è l'ultimo valore
da controllare al prossimo ciclo

pongo n=i+1 (ultimo i utilizzato)
perché la condizione del for è
i<n-1

i=0; scambio 0 con 1

stop

Bubble Sort

```
void bubbleSort_4(int* v, int n){
    int tmp,j=0, p=n;
    bool continua;
    do {
        continua=false;
        for (int i=0; i<n-1; i++){
            if (v[i]>v[i+1]){ // SWAP
                continua=true;
                p=i+1;
                tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
            } // end IF
        } // end FOR
    n=p;
    }
    while (continua && n>1);
}
```

L'ultimo scambio coinvolge i, i+1
Pongo quindi n= (i+1)
[usando p come appoggio]
in modo che al ciclo successivo
l'ultima coppia confrontata sia (i-1, i)

bubbleSort_4.c

Bubble Sort

Complessità computazionale

Bubble Sort

```
void bubbleSort_1(int* v, int n){
  int tmp;
  for (int j=0; j<n-1; j++){
    for (int i=0; i<n-1; i++){
      if (v[i]>v[i+1]){
        // SWAP
        tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
      }
```

O(n^2)

Bubble Sort – Caso Migliore

```
void bubbleSort_4(int* v, int n){
    int tmp,j=0, p=n;
    bool continua;
    do {
        continua=false;
        for (int i=0; i<n-1; i++){
            if (v[i]>v[i+1]){ // SWAP
                continua=true;
                p=i+1;
                tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
            } // end IF
        } // end FOR
    n=p;
    }
    while (continua && n>1);
}
```

CASO MIGLIORE
VETTORE GIA' ORDINATO

Assegnamento esterno al ciclo: 2

DO: numero di test: 1; iterazioni: 1

FOR: numero di test: n; iterazioni: n-1

Computazioni interne al ciclo: 1 test per ogni iterazione

$f(n) = 2 + 1 + n + 1 \cdot (n - 1) = 2n + 2$

$O(n)$

Bubble Sort – Caso Peggiore

```
void bubbleSort_4(int* v, int n){
    int tmp,j=0, p=n;
    bool continua;
    do {
        continua=false;
        for (int i=0; i<n-1; i++){
            if (v[i]>v[i+1]){ // SWAP
                continua=true;
                p=i+1;
                tmp=v[i]; v[i]=v[i+1]; v[i+1]=tmp;
            } // end IF
        } // end FOR
    n=p;
    }
    while (continua && n>1);
}
```

CASO PEGGIORE
VETTORE IN ORDINE INVERSO

Siamo nel caso 'Algoritmo che cicla su input eliminando un elemento per volta'

$$f(n) = f(n-1) + n = n \cdot \frac{n+1}{2}$$

$O(n^2)$

Bubble Sort – Caso Medio

Il calcolo non è immediato, ma possiamo fare le seguenti considerazioni

- Cosa significa 'caso medio'? Dobbiamo formulare ipotesi circa la distribuzione da cui campioniamo I vettori
- Se i vettori non sono precedentemente ordinati, abbiamo comunque due cicli annidati.
- Le costanti in f(n) sono legate a quanto 'precocemente' l'algoritmo esce dai cicli
- Ma il fatto che ci siano due cicli annidati implica che la complessità sia $O(n^2)$, come nel caso peggiore.

Cenni al calcolo esatto:
https://math.stackexchange.com/questions/453686/what-is-the-expected-number-of-swaps-performed-by-bubblesort

Esercizi

- Illustrate l'operazione di BubbleSort su un array dato
- Calcolare la complessità f(n) utilizzando come input un array dato. Verificare la correttezza di f(n) contando le istruzioni durante l'esecuzione.
- Ipotizzare che lo spostamento di un valore dell'array (dunque `el=v[i], v[j+1]=v[j]`, ecc) abbia costo *c* rispetto alle altre istruzioni elementari. L'ipotesi ha senso perché gli elementi da spostare potrebbero essere strutture. Calcolare f(n).

Esercizi

- Applicate l'algoritmo ad un array DI RECORD
- Applicate l'algoritmo ad un array di PUNTATORI a RECORD

(l'ordinamento ha come oggetto uno dei campi del record)

Contate i passi base per i seguenti casi specifici, con n=8

- vettore ordinato in ordine inverso
- V=[6 1 4 5 3 2 3 7]
- Prima parte del vettore ordinata, seconda parte ordinata in ordine inverso.

Esercizi

- Modificare BubbleSort in modo che disponga gli elementi in ordine non crescente
- Modificare BubbleSort in modo che l'ordinamento possa essere crescente o decrescente. È possibile evitare di introdurre un ulteriore IF all'interno dell'algoritmo?

- Modificare BubbleSort in modo che ordini un vettore di strutture, a seconda di uno specifico campo
- Applicare Strategy pattern per scegliere dinamicamente il criterio di ordinamento e / o il campo
- Stessa cosa, ma usando una lista concatenata al posto di un vettore statico