Ordinamento

Laboratorio di Programmazione II

Corso di Laurea in Bioinformatica

Dipartimento di Informatica - Università di Verona

Sommario

- Gli array in java
- Problema ordinamento
- Selection sort per array
- Ordinamento di oggetti: il concetto di interfaccia
- Interfaccia Comparable
- Merge Sort
- QuickSort

Array in Java

Array (vettore)

Concetti di base

- Sequenza ordinata di elementi dello stesso tipo
- Ogni elemento e' univocamente identificablile tramite un indice che ne indica la posizione nella sequenza
- $N[i]$ = elemento i-esimo del vettore N

Array in java

Implementazione Array in Java

1. Dichiarazione

```
//array di interi con 10 elementi
int[] x = new int[10];
//array di double con 7 elementi
double[] d = new double[7];
```

2. Accesso

```
//elemento i-esimo
x[i]
```

Array in java

Implementazione Array in Java

1. La dimensione di un array e' invariabile
2. Il campo length mantiene la dimenzione dell'array

```
//array di interi con 10 elementi
int[] x = new int[10];
//assegna 0 a tutti gli elementi dell'array
for (int i=0; i<x.length;i++)
   x[i] = 0;
```

3. Gli indici di un array vanno da 0 a length-1

Esempio uso Array

Example (Lettura e stampa Array)

Lettura da input di un Array

```
private static void leggiArrayInput(int[] res) {
   Scanner sc = new Scanner(System.in);
      for (int i = 0; i < res.length; i++) {
       res[i] = sc.nextInt();
   }
}
```

Stampa di un array

```
private static void stampaArray(int[] x) {
   for (int i = 0; i < x.length; i++) {
      System.out.print(x[i]+" ");
   }
   System.out.println();
}
```

Fibonacci Iterativo

Fibonacci usando gli array

- numero di attivazione di Fibonacci ricorsivo cresce esponenzialmente in n.
- possiamo scrivere una versione iterativa che cresce lineramente in n
- dobbiamo salvare i numeri che calcoliamo nella iterazione in un array.

Fibonacci Iterativo

Pseudocodice

```
FiboIter(n)
    array F di n + 1 elementi
    f[0] = 1;
    f[1] = 1;
    for (i=2,i<=n;i++)
        f[i] = f[i-1] + f[i-2];
```

Esercizi

Array

Implementare i metodi

- calcolaVarianza
  - dato un vettore x di n elementi $var(x) = \sum_{i=0}^{n-1}(x[i] - \mu)^2/n$ dove μ e' la media degli elementi di x.
- fibonaccilterativo

della classe UtilArray.java

Ordinamento

Il problema dell'ordinamento

ordinamento

- Data una sequenza di n elementi $\langle a_0, \cdots, a_{n-1} \rangle$ ed una relazione di ordine $\leq$ tra gli elementi
- Trovare una permutazione degli elementi $\langle a'_0, \cdots, a'_{n-1} \rangle$ tale che $a'_0 \leq a'_1 \leq \cdots \leq a'_{n-1}$
- Esistono molti algoritmi per risolvere questo problema

Example (ordinare un vettore di interi)

Dato un vettore di interi ordinare gli interi in ordine crescente

Ordinamento per selezione del minimo

Selection Sort

Idea generale:

- Passo i: seleziono l'elemento j che occupera' la posizione i-esima nel vettore ordinato
- Elemento selezionato: elemento minore da i in poi.
- Se l'elemento nella posizione i non e' l'elemento corretto (i!=j) scambio gli elementi.

Ordinamento per selezione del minimo

Pseudocodice

Dato array $a = \langle a_0, a_1, \cdots, a_{n-1} \rangle$

```
for i = 0 to i = n-2
   j = indice elemento minore da i ad n-1
   se (i!=j)
      scambia(i,j)
```

Ordinamento per selezione del minimo

Esempio esecuzione

Dato array $a = \langle 73524 \rangle$

```
 7   3   5   2   4
[2]  3   5   7   4

 2   3   5   7   4
[2   3]  5   7   4

 2   3   5   7   4
[2   3   4]  7   5

 2   3   4   7   5
[2   3   4   5]  7
```

Esercizi

SelectionSort

- Implementare i metodi trovaMinimo e scambia della classe SortInteri.java
- Contare il numero di passi ed il numero di scambi eseguiti dall'algoritmo (Come varia il numero di passi ed il numero di scambi al variare dell'input ?)

Selection Sort: discussione

Caratteristiche

- Numero passi: $n * (n-1)/2$, costo: $O(n^2)$
- Costo indipendente dall'input: $O(n^2)$ anche se il vettore e' gia' ordinato
- Numero scambi:
  - caso migliore 0
  - caso peggiore $O(n)$

Ordinamento di oggetti, il meccanismo delle interfacce

Ordinare elementi generici

Example (Ordinare array di double)

- Supponiamo di voler ordinare un array di double utilizzando SelectionSort
- Il codice SortInteri non funziona
  - SelectionSort(int[] a)
- Il codice che dovrei scrivere e' identico a meno della dichiarazione del tipo dell'array
- Si puo' fare di meglio

Ordinare elementi generici II

Idee principali

- Per eseguire l'ordinamento non mi interessa il tipo degli oggetti
- Debbo poter confrontare oggetti tra loro in maniera trasparente rispetto al tipo
- L'utilizzo delle interfacce permette di risolvere questo problema

Il concetto di Interfaccia

Interfaccia

- Definisce un tipo in modo simile ad una classe, keyword: interface
- Puo' contenere costanti:
- Signature di metodi, ma non il corpo
- Rappresenta un contratto
- Esempio: Coordinate.java

Osservazioni

Caratteristiche Interfacce

- Insieme di dichiarazione di metodi senza dati e senza corpo
- Una classe puo' implementare un interfaccia:
  - Se la classe *A* implementa l'intefaccia *I*, *A* puo' essere usata come fosse un oggetto di tipo *I*
- Importante: quando una classe implementa una interfaccia deve implementare tutti i metodi definiti nell'interfaccia
- Non e' possibile istanziare una interfaccia:
  - Coordinate c = new Coordinate(); NO!
- Un interfaccia puo' essere implementata da una classe o estesa da un altra interfaccia

Implementare interfacce

Implementazione delle interfacce

- keyword implements dichiara di implementare l'interfaccia
- Definire ogni metodo dell'interfaccia
- Se *A* implementa *I* si dice che *A* realizza *I*
- Esempio: CoordinateCartesiane.java

Esercizi

Esercizi interfacce

- Implementare la classe CoordinatePolari che realizza l'interfaccia Coordinate (CoordinatePolari.java)
- Implementare i metodi scambia(...) e trovaMinimo(...) della classe SortCoordinate.java
- Implementare il metodo ordinaCoordinatePolari(int n) della classe SortCoordinate.java

Uguaglianza tra oggetti: il metodo Equals

Uguaglianza tra oggetti

Uguaglianza tra oggetti

- L'uguaglianza dovrebbe essere basata sul valore non sul riferimento
- o1 == o2, vero se o1 ed o2 puntano alla stessa locazione di memoria
- voglio poter stabilire se o1 ed o2 rappresentano il medesimo oggetto
- Esempio: uguaglianza tra stringhe:

```
String s1 = new String("Ciao");
String s2 = new String("Ciao");
s1 == s2 // false
s1.equals(s2) // true
```

Uguaglianza tra oggetti

Il metodo Equals

- Metodo della classe Object
- Qualsiasi oggetto in java eredita da Object
- Quindi qualsiasi oggetto puo' ridefinire Equals
- Il metodo equals di default ritorna ==

Re-implementare metodi

Overriding e Polimorfismo

- Classe implementa Equals: due metodi con la stessa segnatura che fanno cose diverse (Polimorfismo)
- Il metodo eseguito e' sempre quello della classe piu' specifica (Overriding)
- Questo vale per qualsiasi metodo
- Esempio: Metodo toString in CoordinateCartesiane.java

Re-implementare il metodo Equals

Downcasting

- Per fare overriding la signature di un metodo deve essere identica a quello della classe superiore
- Signature di Equals: public boolean Equals(**Object** o)...
- Supponiamo di voler ridefinire il metodo Equals per la classe CoordinateCartesiane.java
- Per poter accedere ai campi di interesse della classe A devo considerare l'oggetto o come una istanza della classe CoordinateCartesiane (downcasting)
- Il downcasting puo' sempre essere fatto come un operazione di casting normale: CoordinateCartesiane cc = (CoordinateCartesiane) o

Downcasting: evitare problemi a run time

parola chiave *instanceof*

- Se sbaglio il downcasting viene generata una ClassCastException ed il programma termina
- Posso controllare se il downcasting e' corretto utilizzando instanceof
- Se il downcasting e' corretto vado avanti altrimenti faccio delle operazioni opportune
- Nel caso di Equals se oggetti di tipo diverso ritorno false (assumo che due oggetti di tipi diversi siano diversi)

Metodo Equals: esempio

metodo Equals per CoordinateCartesiane

```
public boolean equals(Object obj) {
        if (obj instanceof CoordinateCartesiane) {
                CoordinateCartesiane cc =
                  (CoordinateCartesiane) obj;
                return ((x == cc.x) && (y==cc.y));
        }
        return false;
}
```

Esercizi

Esercizi Equals

- implementare il metodo Equals per la classe Studente.java
- implementare il metodo Equals per la classe CoordinatePolari.java

Interfaccia Comparable

Interfaccia Comparable

Principali Caratteristiche

- Interfaccia molto generale che descrive oggetti confrontabili
- Unico metodo da implementare: int compareTo(Object)
- restituisce un valore negativo/positivo/nullo se l'oggetto di invocazione e' minore/maggiore/uguale al parametro

Classi che realizzano Comparable

Principali classi che realizzano Comparable

Class	Natural Ordering
Byte	signed numerical
Character	unsigned numerical
Long	signed numerical
Integer	signed numerical
Short	signed numerical
Double	signed numerical
Float	signed numerical
BigInteger	signed numerical
BigDecimal	signed numerical
File	system-dep. lexicographic on pathname
String	lexicographic

Realizzare l'interfaccia comparable

Realizzazione di Comparable

- Dichiaro che la classe implementa l'interfaccia comparable
  - `public class Studente implements Comparable{...}`
- Implemento il metodo public int compareTo(Studente)
- Esempio: Studente.java
- NOTE: la classe Studente.java genera degli avvertimenti in compilazione (warning), dovuti ad operazioni non type safe. OPZIONALE Per una versione type safe vedere la classe StudenteTypeSafe.java (si capira' meglio dopo l'introduzione dei generics)

Utilizzare l'interfaccia Comparable

Utilizzo di Comparable

Posso chiamare il metodo compareTo su qualsiasi oggetto che realizza Comparable

Example (Massimo tra due oggetti comparable)

```
public static Comparable massimo(Comparable s1,Comparable s2){
   if (s1.compareTo(s2)<0){
      return s2;
   } else {
      return s1;
   }
}
```

NOTA: posso comparare oggetti comparable ma diversi tra loro (e.g., una stringa con uno studente), questo potrebbe essere un problema.

Esercizi

Esercizi su Comparable

- implementare il metodo tovaMinimo(...) della classe Sort.java
  Nota: la classe Sort.java restituisce warning di compilazione per alcune operazioni non *type safe*, OPZIONALE vedere la classe Sort2.java per una versione *type safe*
- Utilizzare la classe TestSort.java per verificare la correttezza del metodo
  (**Nota**: scaricare il file studenti.txt per eseguire il metodo main)

Merge Sort

Ordinamento per Fusione (Merge Sort)

Idee principali

- Dato un array di 1000 elementi, usando Selection Sort avro' un numero di passi nell'ordine di 10^6.
- Posso fare di meglio:
  - divido l'array in due array di 500 elementi ciascuno
  - ordino primo array: 250000 operazioni
  - ordino secondo array: 250000 operazioni
  - combino (fondo) i due array, si puo' fare in O(n): 1000 operazioni
  - Totale: 250000 + 250000 + 1000 = 501000 operazioni vs. 10^6
- posso suddividere fino a quando non raggiungo array di un solo elemento, evito $O(n^2)$

Algoritmo Merge Sort

Idee principali

- Dato un array $A = \{0, \cdots, n\}$
- Suddivido in due parti uguali fino a quando la dimensione e' 1 (divide)
- fondo le sottoparti a due a due ordinando gli elementi fino ad ottenere la dimensione originale (impera)
- L'array finale e' ordinato.

Merge Sort: Pseudocodice

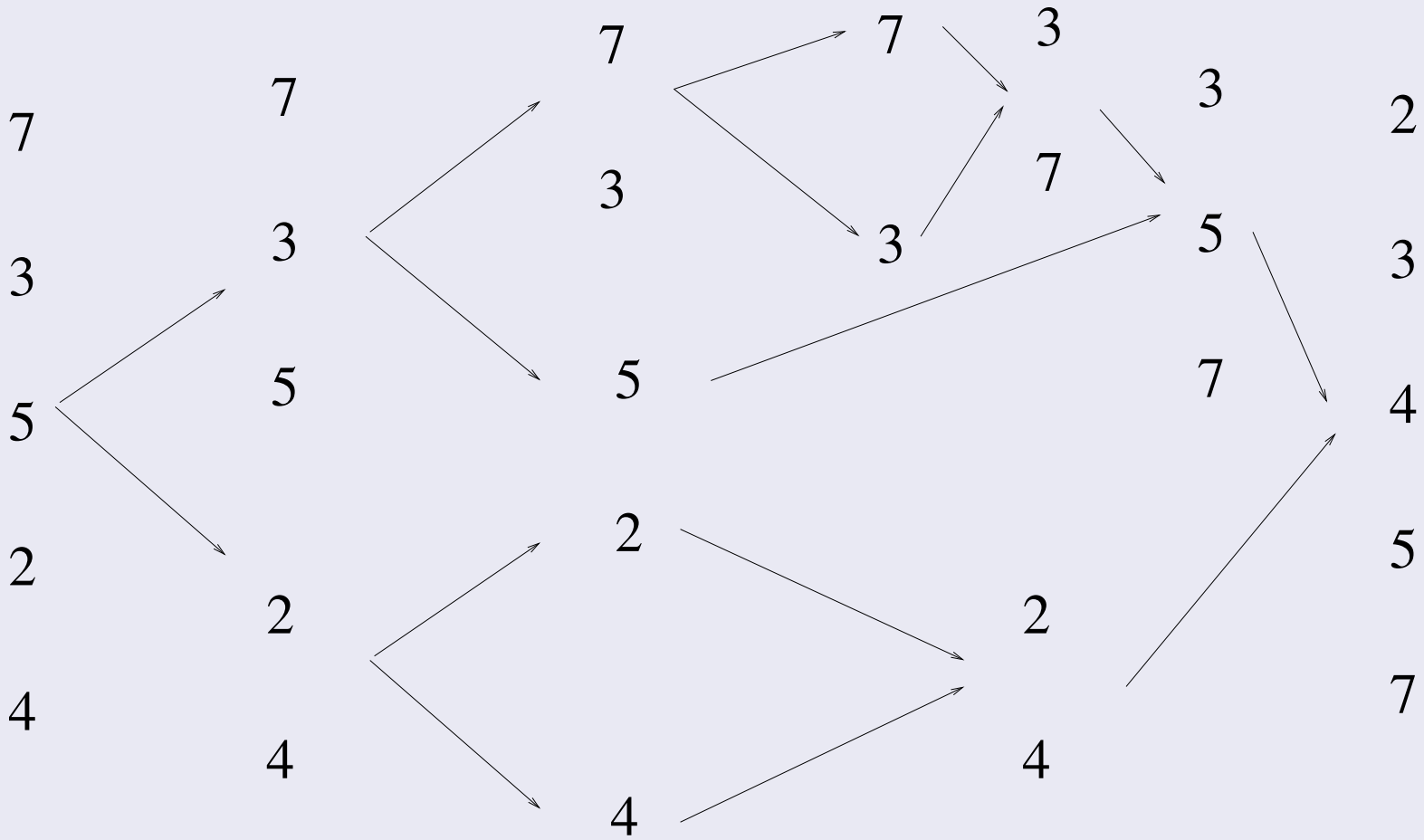
Algoritmo Merge Sort

Ordino per fusione gli elementi da inf a sup del vettore

```
if (inf<sup){
   med = (inf+sup)/2;
   ordina per fusione da inf a med
   ordina per fusione da med+1 a sup
   fondi gli elementi da inf a
    med con gli elementi da med+1 a sup
}
```

Merge Sort: Esempio

Esempio



Implementazione Merge

Merge di due vettori

Utilizzo un vettore di appoggio e due indici per scandire i due sottovettori

- il più piccolo tra i due elementi indicati dai due indici viene copiato nella prossima posizione del vettore di appoggio;
- l'indice corrispondente viene avanzato;
- si continua così fino a quando uno dei due sottovettori non è terminato;
- quando uno dei due sottovettori è terminato si copiano gli elementi rimanenti dell'altro nel vettore di appoggio;
- alla fine si copia il vettore di appoggio nelle posizioni occupate dai due sottovettori.
- Implementazione: vedi metodo merge(..) in Sort.java

Costo Merge Sort

Costo Merge Sort

- Il costo e' $O(nlogn)$.
  - Ottenuto tramite equazioni di ricorrenza.
- Questo e' quanto di meglio si possa fare per l'ordinamento
  - Si deve utilizzare un vettore di appoggio.

Esercizi

Esercizi Merge Sort

- Dato il metodo merge(...) implementare il metodo mergeSort(...) della classe Sort.java
- Modificare la classe CoordinatePolari cosi' da implementare Comparable
- Scrivere un metodo main della classe CoordinatePolari.java che ordini un vettore di Coordinate polari generate in maniera casuale (utilizzare il metodo Math.random()).

Le soluzioni sono in CoordinatePolari.comp.sol e la versione type safe CoordinatePolari2.comp.sol

QuickSort

QuickSort: idee principali

Quick Sort

- Dato un vettore si sceglie un qualsiasi elemento detto pivot (e.g., l'ultimo elemento dell'array)
- Si determina (con una scanzione del vettore) la posizione in cui il pivot si trovera' nel vettore ordinato detta q
- Durante la scansione:
  - portiamo tutti gli elementi minori di pivot a sinistra di q
  - portiamo tutti gli elementi maggiori di pivot a destra di q
- mettiamo pivot nella posizione q. Conclude il partizionamento del vettore.
- Ordiniamo con QuickSort il vettore da 0 a $q-1$
- Ordiniamo con QuickSort il vettore da $q+1$ a n

QuickSort: Esempio

Example

Quick Sort

- Dato v = [2 8 7 1 3 5]
- pivot = 5, $q = 3$
- v' = [2 1 3 5 7 8]
- Ordiniamo con QuickSort [2 1 3]
- Ordiniamo con QuickSort [7 8]

QuickSort: pseudocodice

pseudocodice QuickSort

```
quickSort(A,p,r)
   if (p<r)
      q = partition(a,p,r)
      quickSort(a,p,q-1);
      quickSort(a,q+1,r);

partition(a,p,r)
   pivot = a[r];
   i = p-1;
   for j = p to r-1
      if (a[j]<=pivot)
         i = i+1;
         scambia a[i] con a[j]
   scambia a[i+1] con a[r]
   return i+1
```

QuickSort: Esempio di esecuzione

Esempio quick sort

Dato array $a = \langle 2\,8\,7\,1\,3\,5 \rangle$

```
[2, 8, 7, 1, 3, 5]
pivot = 5
partizionato a [2 1 3 5 7 8 ], p = 0 r = 5 q = 3
quicksort su [2 1 3 ] p = 0 r = 2
pivot = 3
partizionato a [2 1 3 ], p = 0 r = 2 q = 2
quicksort su [2 1 ] p = 0 r = 1
pivot = 1
partizionato a [1 2 ], p = 0 r = 1 q = 0
quicksort su [] p =0 q = -1
quicksort su [2 ] p = 1 r = 1
quicksort su [] p = 3 r = 2
quicksort su [7 8 ] p = 4 r = 5
pivot = 8
partizionato a [7 8 ], p = 4 r = 5 q = 5
quicksort su [7 ] p = 4 r = 4
quicksort su [] p = 6 r = 5
[1, 2, 3, 5, 7, 8]
```

QuickSort: Ivarianti di ciclo

Invariante di ciclo

- ogni volta che eseguiamo il ciclo gli elementi del vettore soddisfano delle condizioni
- 3 condizioni:
  1. p <= indice <= i: a[indice] <= pivot
  2. i < indice <= j: a[indice] > pivot
  3. a[r] = pivot
- queste condizioni valgono all'inizio di ogni iterazione del ciclo

QuickSort: Invarianti di ciclo II

Invariante di ciclo II

- Inizializzazione: tre proprieta' banalmente soddisfatte perche' non ci sono elementi tra p ed i e tra i+1 e j-1
- Passo k
  1. a[j] > pivot : sposto j di 1 quindi a[j-1] > pivot. Le altre proprieta' non vengono alterate
  2. a[j] <= pivot : avanzo i di 1, e scambio a[i] con a[j]. Quindi all'inizio del prossimo ciclo a'[i] = a[j] <= pivot e a'[j-1] = a[i] > pivot
- Terminazione: j=r quindi gli elementi dell'array sono stati correttamente partizionati nelle tre regioni di interesse

Quick Sort: discussione

Caratteristiche

- Caso migliore: ad ogni passo dividiamo l'array in due sottoarray con $n/2$ elementi $O(nlogn)$
  - scelta pivot fondamentale
  - elemento in posizione mediana del vettore
  - elemento medio del vettore
- Caso peggiore: vettore ordinato $O(n^2)$
- Costo asintotico peggiore di merge sort, ma se vettore e' *disordinato* si comporta molto bene
- Molto spesso preferito nella pratica, non necessita un vettore di appoggio

Riassunto complessita' metodi di ordinamento

Complessita' metodi di ordinamento

Algoritmo	Confronti	Scambi	Costo totale
SelectionSort	$O(n^2)$	$O(n)$	$O(n^2)$
		0 ordinato	
BubbleSort	$O(n^2)$	$O(n^2)$	$O(n^2)$
	$O(n)$ ordinato	0 ordinato	$O(n)$ ordinato
MergeSort	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$
		usa vettore appoggio	
QuickSort	$O(n^2)$	$O(n^2)$	$O(n^2)$
			$O(nlogn)$ caso medio

Esercizi

Esercizi quicksort

- implementare il metodo qSort(...) della classe Sort.java
- verificare il corretto funzionamento del metodo utilizzando il main della classe Sort.java