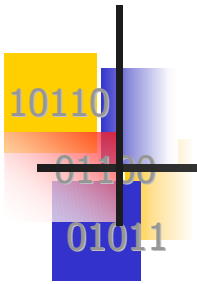


Algoritmi di Ricerca

Esempi di programmi Java

Ricerca in una sequenza di elementi

- Data una sequenza di elementi, occorre verificare se un elemento fa parte della sequenza oppure l'elemento non è presente nella sequenza stessa.

- In generale una sequenza di elementi si può realizzare come un **`array`**. E la scansione avviene usando un indice.

- Se la sequenza non è ordinata a priori occorre eseguire una **ricerca lineare** o **sequenziale**.

- Se la sequenza è ordinata è opportuno eseguire una **ricerca binaria**.

Ricerca lineare

- L'algoritmo di ricerca lineare (o sequenziale) in una sequenza (**`array`**) è basato sulla seguente strategia:
  - Gli elementi dell'array vengono analizzati in sequenza, confrontandoli con l'elemento da ricercare (chiave) per determinare se almeno uno degli elementi è uguale alla chiave.
  - Quando si trova un elemento uguale alla chiave la ricerca termina.

- La ricerca è sequenziale, nel senso che gli elementi dell'**`array`** vengono scanditi uno dopo l'altro sequenzialmente.

- L'algoritmo prevede che al più tutti gli elementi dell'array vengano confrontati con la chiave. Se l'elemento viene trovato prima di raggiungere la fine della sequenza non sarà necessario proseguire la ricerca.

Ricerca lineare in Java

```
/* Questo metodo implementa la ricerca lineare
restituendo l'indice di un elemento di seq uguale
all'elemento cercato (chiave) oppure -1 che indica
che l'elemento non è presente nella sequenza.*/

public static int ricLineare(int[] seq, int chiave)
{
   int i;          // indice per la scansione di seq
   int ind_elem; // indice di un elemento uguale a
                   // alla chiave
   ind_elem = -1;

   for(i=0; ind_elem==-1 && i<seq.length; i++)
   {
    if (seq[i]==chiave)
      ind_elem = i;
   }
   return ind_elem;
}
```

Uso della ricerca lineare per contare le occorrenze

- L'algoritmo di ricerca lineare può essere usato per verificare quante volte un elemento è presente in una sequenza:
  1. Gli elementi dell'array vengono analizzati in sequenza, confrontandoli con l'elemento da ricercare (chiave) per determinare se uno degli elementi è uguale alla chiave.
  2. Quando si trova un elemento uguale alla chiave **si incrementa un contatore** e il passo 1 viene rieseguito fino alla fine della sequenza.

Calcolo delle occorrenze

```
/* Questo metodo conta quante volte l'elemento
cercato (chiave) è presente in seq e ritorna il
valore delle occorrenze (il valore 0 indica che
l'elemento non è presente nella sequenza).*/

public static int Conta(int[] seq, int chiave)
{
   int i;          // indice per la scansione di seq
   int occorre;    // valore delle occorrenze della
                   // chiave in seq
   occorre = 0;

   for(i=0; i<seq.length; i++)
   {
    if (seq[i] == chiave)
      occorre++;
   }
   return occorre;
}
```

Ricerca binaria

- L'algoritmo di ricerca lineare richiede che al più tutti gli elementi dell'array vengano confrontati con la chiave. Questo è necessario perché la sequenza non è ordinata.

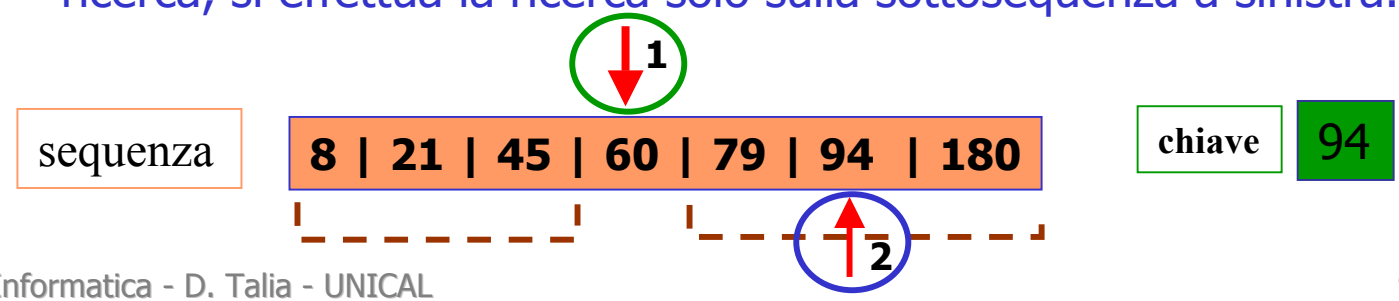
- Se la sequenza su cui occorre effettuare la ricerca è ordinata si può usare un algoritmo di ricerca molto più efficiente che cerca la chiave sfruttando il fatto che gli elementi della sequenza sono già disposti in un dato ordine.

- Esempi di sequenze ordinate: elenco telefonico, agenda, etc.

- In questi casi si usa un algoritmo di **ricerca binaria** che è più efficiente perché riduce lo spazio di ricerca.

Ricerca binaria

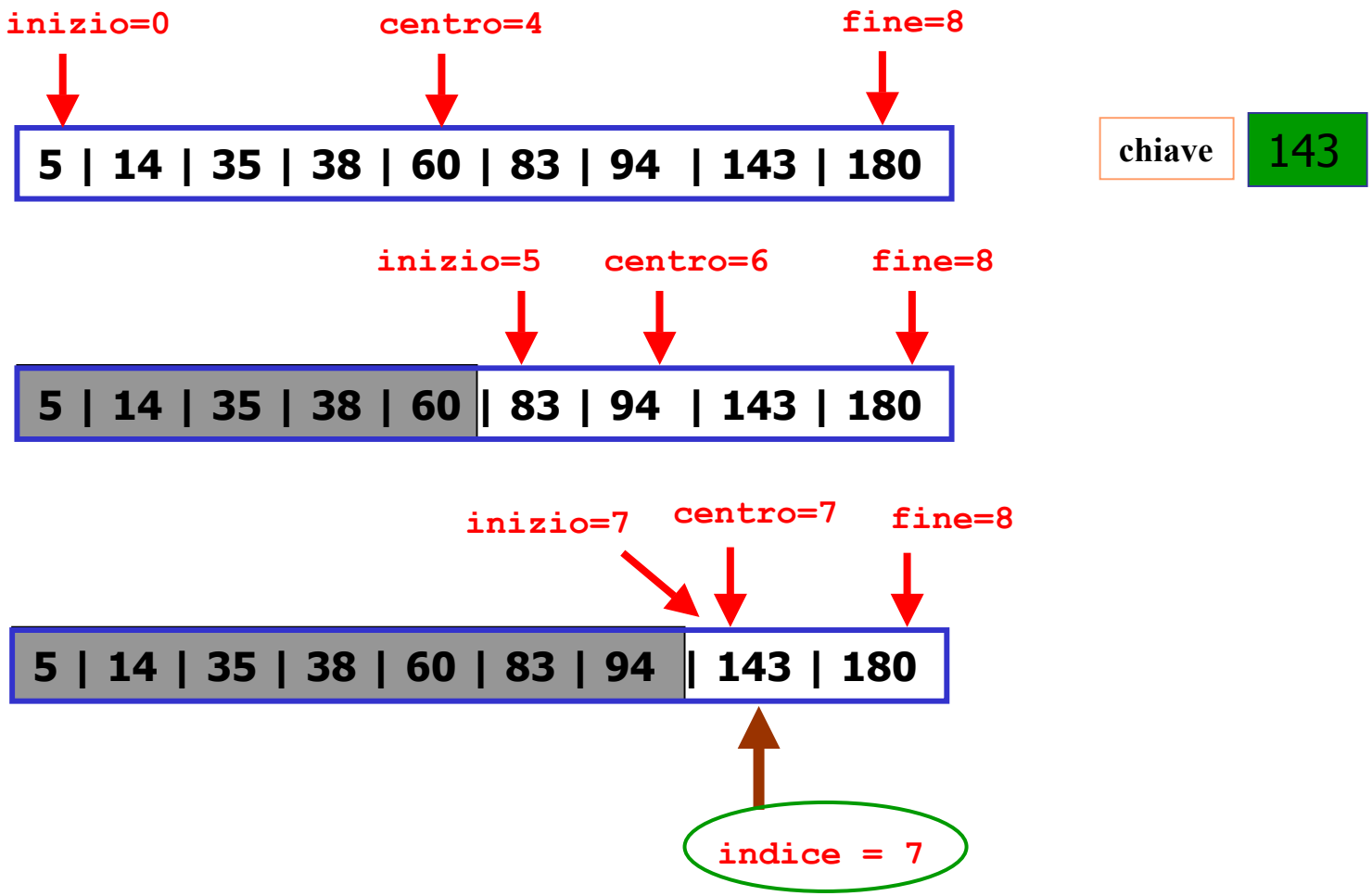
- L'algoritmo di ricerca binaria cerca un elemento in una sequenza ordinata in maniera crescente (o non decrescente) eseguendo i passi seguenti finché l'elemento viene trovato o si è si è completata la ricerca senza trovarlo:
  1. Confronta la chiave con l'elemento centrale della sequenza,
  2. Se la chiave è uguale all'elemento centrale, allora la ricerca termina positivamente,
  3. Se invece la chiave è maggiore dell'elemento centrale si effettua la ricerca solo sulla sottosequenza a destra,
  4. Se invece la chiave è minore dell'elemento centrale dello spazio di ricerca, si effettua la ricerca solo sulla sottosequenza a sinistra.



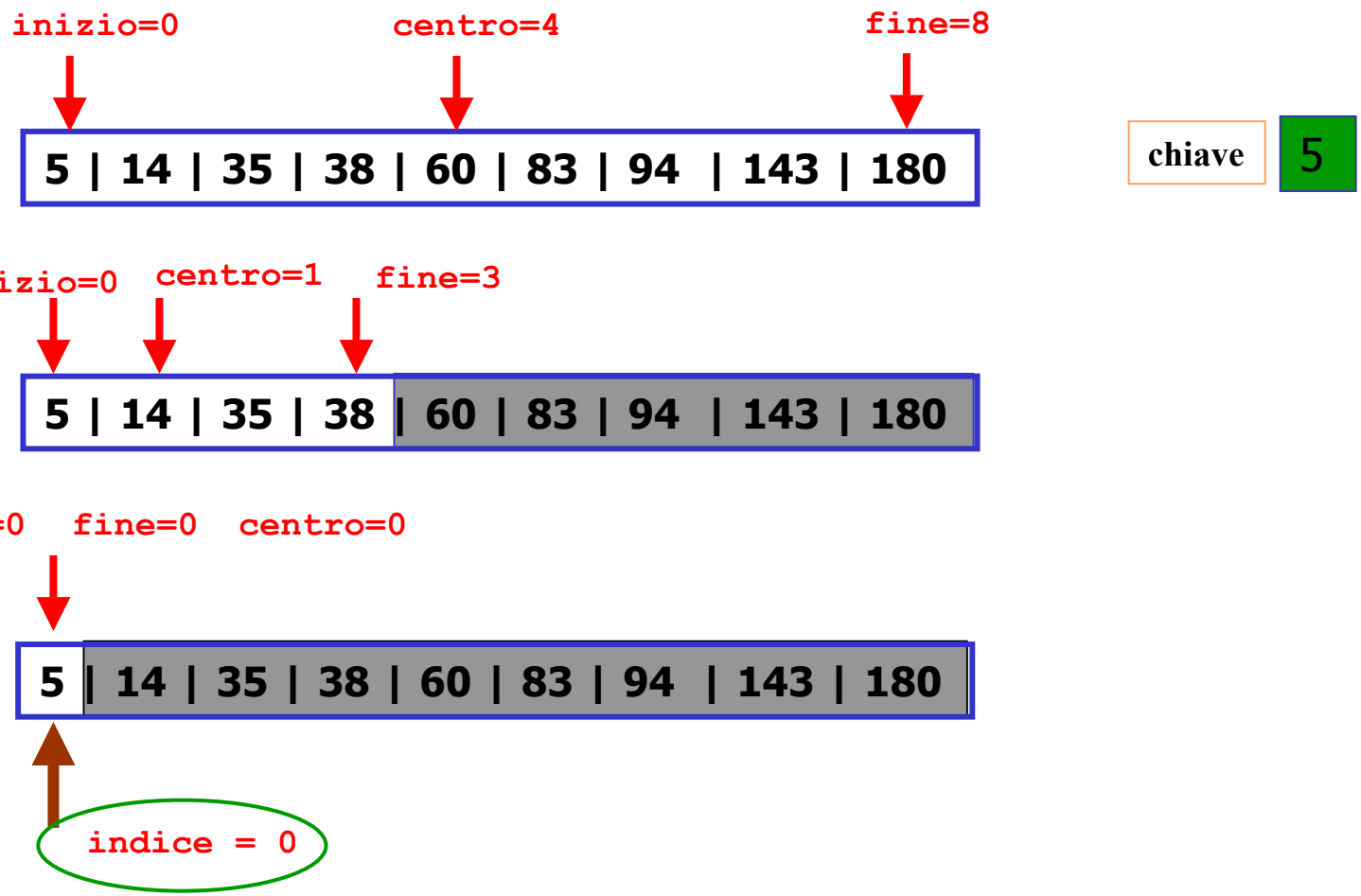
Ricerca binaria in Java

```
/* Questo metodo implementa la ricerca binaria restituendo
l'indice di un elemento di seq uguale all'elemento cercato
(chiave) o -1 che indica che l'elemento non è presente*/
public static int ricBinaria(int[] seq, int chiave)
{  // seq è ordinato in modo non decrescente
 int indice; // indice di un elemento uguale a chiave
 int inizio; int fine; int centro;
 inizio = 0;
 fine = seq.length-1;
 indice = -1;
 while (indice ==-1 && inizio <= fine)
 {
  centro = (inizio+fine)/2;
  if (seq[centro]==chiave)       // trovato
    indice = centro;
  else
   if (chiave > seq[centro])   // continua a destra
      inizio = centro + 1;
   else                          // continua a sinistra
      fine = centro - 1;
 }
 return indice;
}
```

Ricerca binaria



Ricerca binaria



Ricerca binaria

- Inizialmente la ricerca è fatta su **N** elementi dove **N** indica la lunghezza della sequenza (lo spazio di ricerca ha dimensione N).
- Ad ogni iterazione lo spazio della ricerca si riduce di "circa" la metà. Potremmo dire che si passa da N ad N/2 e cosi via.
- Il caso peggiore si ha quando l'elemento cercato non si trova nella sequenza (non esiste un elemento uguale alla chiave).
- Nel caso peggiore, l'iterazione viene eseguita $\log_2 N$ volte.

Esempio di gestione di un elenco di nomi

- L'esempio seguente definisce una classe **`Studenti`** i cui oggetti sono elenchi di *N* nomi e matricole di studenti con metodi per inizializzare, cercare e stampare i dati degli studenti in elenco.

- Ogni elenco è definito come un oggetto composto da due campi che sono array di stringhe e di interi: **`nomi`** e **`matricole`**.

- Ognuno di questi due campi contengono rispettivamente una sequenza di nomi e di matricole in maniera che

  `elenco.nomi[i]` e **`elenco.matricole[i]`**

  avendo lo stesso valore dell'indice contengono rispettivamente il nome ed la matricola di uno studente.

- Nell'esempio esiste un'altra classe **`TestStudenti`** che contiene il **`main`** e usa i metodi della classe **`Studenti`**.

Esempio di gestione di un elenco di nomi

```
class Studenti
{
  String[] nomi    = new String[100];          //VARIABILE D'ISTANZA
  int[] matricole =new int[nomi.length]; //VARIABILE D'ISTANZA

 void leggidati()      // METODO PER LEGGERE NOMI E MATRICOLE
 {
   for (int i=0; i< nomi.length; i++)
   {
    nomi[i]=Console.readString ("Inser. nome " + (i+1) + ": ");
    matricole[i]=Console.readInt("Inser. Matr. " + (i+1) + ": ");
   }
 }

 void stampadati() // METODO PER VISUALIZZARE NOMI E MATRICOLE
 {
   for (int i=0; i<nomi.length; i++)
     System.out.println(nomi[i] + " " + matricole[i]);
 }
// continua ……
```

Esempio di gestione di un elenco di nomi

```
//  ……..  continua
int cercamatricola(int MT)  //METODO CHE CERCA PER MATRICOLA

   {
     int j = 0; int i ;
     boolean trov = false;
     for (i=0; i<matricole.length & trov==false; i++)
       if (MT == matricole[i])
         {  trov = true ;
            j = i;
         }
       if (trov == true)
         return j;
       else
         return -1;
   }
}  // fine della classe Studenti
```

Esempio di gestione di un elenco di nomi

- Nella classe **`TestStudenti`** definiamo il **`main`** che crea un oggetto elenco di **`Studenti`** e usa i metodi della classe per
  - inserire i dati di ogni studentea (nome e matricola) nell'elenco,
  - visualizzare l'elenco degli studenti inseriti (nome e matricola),
  - cercare uno studente di cui si conosce la matricola; la ricerca delle persone continua finchè non viene inserito il valore 0 che indica la fine della ricerca.

Esempio di gestione di un elenco di nomi

```
public class TestStudenti
{
 public static void main(String args[])
 { int matricola;
   int indice ;
   Studenti elenco = new Studenti();

   elenco.leggidati();
   System.out.println("-------------");
   elenco.stampadati();
   do
   {
    matricola=Console.readInt("Inser. la matricola da cercare : ");
    if ( matricola != 0 )
    {
     indice = elenco.cercamatricola(matricola);
     if (indice > -1)
        System.out.println(elenco.nomi[indice]+" " +
                                              elenco.matricole[indice]);
     else
       System.out.println ("Studente non presente nell'elenco");
    }
   } while (matricola != 0 );
 } // fine del main
} // fine della classe TestStudenti
```

Esempio di gestione di un elenco di nomi

- L'esempio si può modificare ad esempio aggiungendo :
  - Un metodo `cercanome` che cerca uno studente anche per nome,
  - Un metodo per leggere i dati di un singolo studente non di tutti.
- Oppure modificando
  - Il metodo `cercanome` per cercare tutte gli studenti con un dato nome non solo il primo che incontra.
- Oppure definendo altre variabili d'istanza per memorizzare dati sugli studenti in elenco, ad esempio: corso di laurea, indirizzo, età, ecc.