Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa 10

Strutture collegate - 2

A. Miola

Febbraio 2008

Contenuti

- Strutture di array e strutture collegate
- Esempi ed esercizi
- Operazioni tipiche sulle liste
- Implementazioni ricorsive delle operazioni tipiche sulle liste

Array e strutture collegate

- **È possibile a questo punto fare alcune osservazioni di confronto tra le strutture di array (SA) e le strutture collegate (SC)**
  - **allocazione di memoria differente**
    - **una SA è un unico oggetto mentre una SC sono più oggetti tra loro "collegati"**
  - **una SC consente una migliore gestione di operazioni di inserimento e cancellazione**
  - **una SA consente un accesso diretto ai suoi elementi mentre in una SC l'accesso agli elementi è sequenziale**
  - **la scansione di tutti gli elementi è naturalmente eseguita con un'istruzione for nelle SA e con un'istruzione while o con la ricorsione nelle SC**

Costruzione di una struttura collegata

- Si supponga ad esempio di voler realizzare una struttura collegata per rappresentare una sequenza di caratteri – ad esempio A, B, …, Z
  - posso iniziare creando un nodo che rappresenti il primo elemento carattere A e quindi aggiungere, via via, altri nodi in coda ai precedenti per rappresentare gli altri elementi della sequenza
  - in alternativa potrei anche pensare – sbagliando ! – di iniziare creando un nodo che rappresenti il primo elemento carattere A e quindi aggiungere, via via, altri nodi sempre in testa per rappresentare gli altri elementi della sequenza

Costruzione in coda

```
class NodoLista {
   public char info;
   public NodoLista next;
   public NodoLista() {next = null;}
}
public class ProvaLista {
   public static NodoLista creaInCoda() {
        NodoLista a = new NodoLista();
        NodoLista p = a;
        p.info = 'A';
        p.next = new NodoLista();
        p = p.next;
        char c;
        for (c = 'B'; c < 'Z', c++) {
                p.info = c;
                p.next = new NodoLista();
                p = p.next; }
        p.info = 'Z';
        p.next = null;
        return a; }
}
```

Costruzione in testa

```
public static NodoLista creaInTesta() {
      NodoLista a = new NodoLista();
      a.info = 'A';
      a.next = null;
      NodoLista p = a;
      char c;
      for (c = 'B'; c <= 'Z', c++) {
            NodoLista a = new NodoLista();
            a.info = c;
            a.next = p;
            p = a; }
      return p;
}
```

Esercizi

- Scrivere un metodo per visualizzare gli elementi di una data struttura collegata, cioè per la visualizzazione di tutti i valori della variabile info di ciascun nodo, dal primo all'ultimo
- Scrivere i metodi necessari per costruire una struttura collegata per rappresentare una data sequenza di caratteri disponibile come un array di caratteri

Codifica del metodo di stampa

```
public static void stampa(NodoLista p) {
    while (p.next !=null) {
        System.out.println(p.info);
        p = p.next;
    }
    System.out.println(p.info);
}
```

Codifica del metodo di creazione da array

```
public static NodoLista creaDaArray(char[] s) {
   NodoLista a = new NodoLista();
   NodoLista p = a;
   p.info = s[0];
   p.next = new NodoLista();
   p = p.next;
   int i;
   for (i = 1; i < s.length-1, i++) {
             p.info = s[i];
             p.next = new NodoLista();
             p = p.next; }
   p.info = s[s.length-1];
   p.next = null;
   return a;
}
```

Codifica dell'applicazione

```
public class ProvaLista {

  . . .
public static void main(String[] args){
  NodoLista a = creaInCoda();
  NodoLista b = creaInTesta();
  stampa(a);
  stampa(b);
  char[] s = {'A', 'B', 'C', 'D'}
  NodoLista c = creaDaArray(s);
  stampa(c);
}
```

Esercizi

- Scrivere i metodi necessari per costruire una struttura collegata per rappresentare una data sequenza di caratteri disponibile come una stringa

Operazioni comuni sulle liste

- Le operazioni che comunemente vengono eseguite sulle liste sono
  - inserimento
  - ricerca
  - eliminazione
  - accesso
  - modifica
  - lunghezza
  - copia
- Vediamo diverse implementazioni di alcune di queste operazioni
- Le applicazioni che vedremo in seguito avranno bisogno di implementare di volta in volta alcune o tutte le operazioni suddette

Implementazione delle operazioni comuni sulle liste

- Facciamo riferimento alla seguente struttura del nodo di lista di stringhe, con uno specifico nuovo costruttore

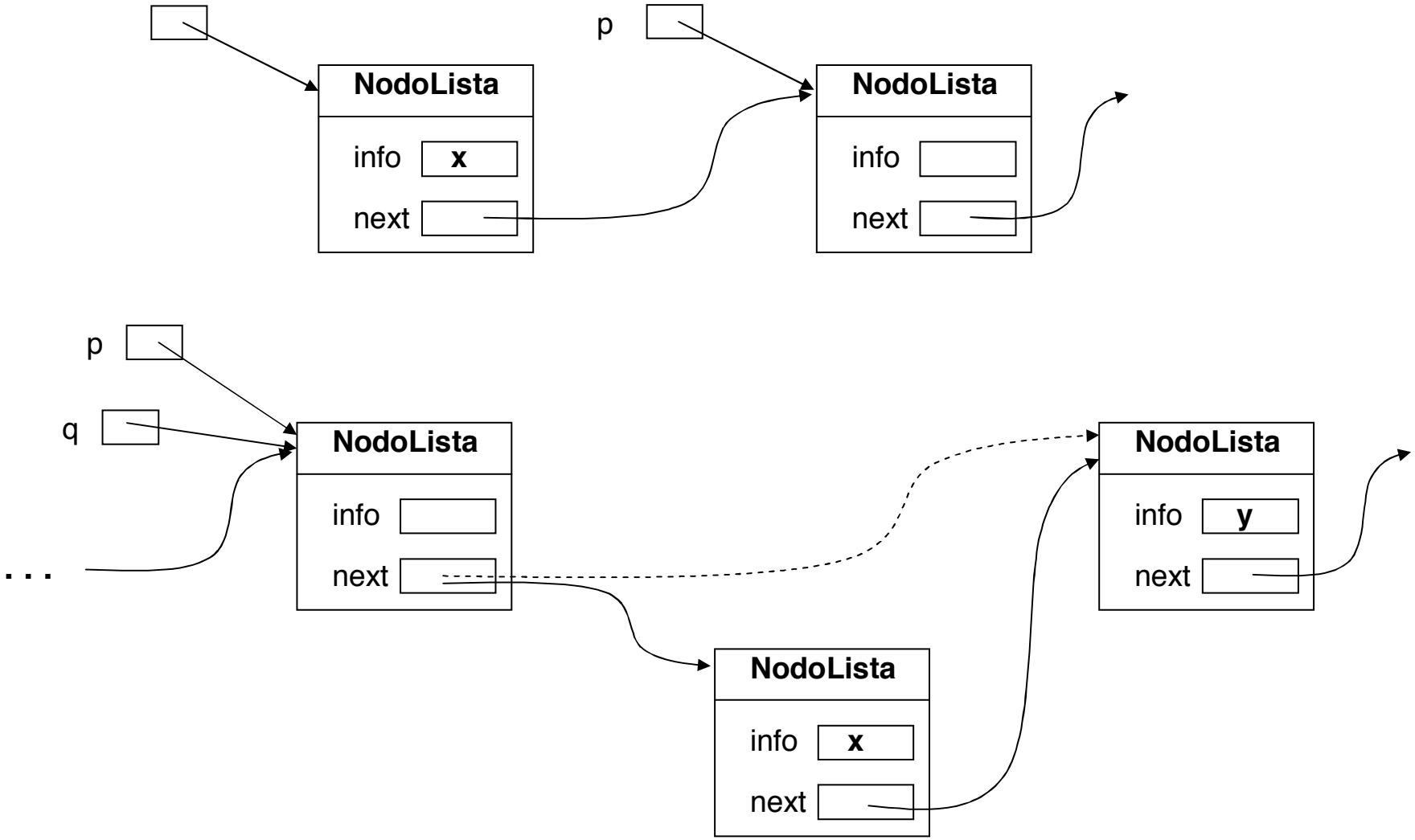
```
class NodoLista {
public String info;
public NodoLista next;
public NodoLista(String i, NodoLista n)
  { info = i; next = n; }
}
```

Inserimento di un elemento

```
// Inserimento in prima posizione
public static NodoLista aggiungiPrimo(NodoLista p, String x) {
   return new NodoLista(x,p); }

// Inserimento di x prima di y in p
public static NodoLista
       aggiungiPrimaDi(NodoLista p, String x, String y) {
  NodoLista q = p;
  if (p!=null)
     if (p.info.equals(y)) p = new NodoLista(x,p);
     else {
         while (q.next!=null && !q.next.info.equals(y))
             q = q.next;
         if (q.next!=null) q.next = new NodoLista(x,q.next);
      }
  return p;
}
```

Rappresentazione delle operazioni



Esercizio

- Fornire la rappresentazione grafica per ciascuna delle operazioni che seguono

Ricerca di un elemento

```
public static boolean
            cerca(NodoLista p, String x) {
  boolean trovato = false;
  if (p!=null) {
     NodoLista q = p;
     while (q!=null && !trovato) {
          if (q.info.equals(x))
               trovato = true;
          q = q.next;
     }
  }
  return trovato;
}
```

Eliminazione di un elemento

```
public static NodoLista
                  elimina(NodoLista p, String x) {
  if (p != null)
    if (p.info.equals(x))
         p = p.next;
    else {
     NodoLista q = p;
     while (q.next!=null &&
                  !q.next.info.equals(x))
                       q = q.next;
     if (q.next!=null) q.next = q.next.next;
    }
  return p;
}
```

Modifica di un elemento

```
// modifica la prima occorrenza di x in p
public static void
     modifica(NodoLista p, String x, String y){
  if (p!=null) {
     NodoLista q = p;
     while (q!=null && !q.info.equals(x))
                    q = q.next;
     if (q!=null)
          q.info = y;
  }
}
```

Lunghezza di una lista

```
public static int lunghezza(NodoLista p) {
      int n=0;
      NodoLista q = p;
      while (q!=null) {
            n++;
            q = q.next;
      }
      return n;
}
```

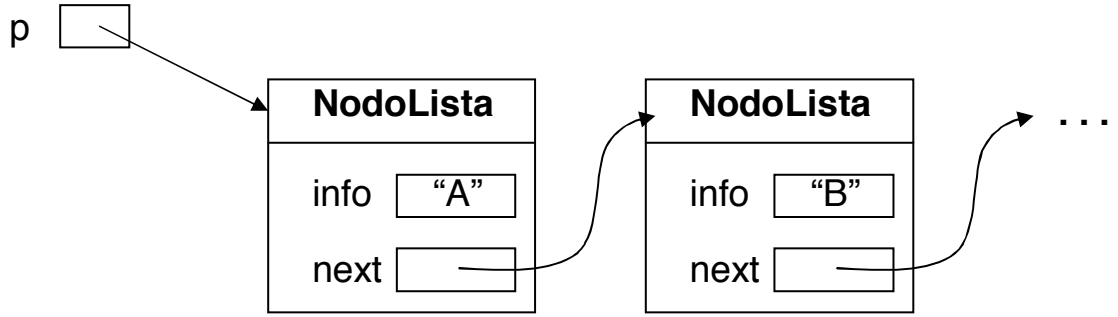
Copia di una lista

```
public static NodoLista copia(NodoLista p) {
      // uso un nodo generatore
   NodoLista r = new NodoLista(null,null);
   Nodolista t = r;
   NodoLista q = p;
   while (q!=null) {
      t.next = new NodoLista(q.info,null);
      q = q.next;
      t = t.next;
   }
   return r.next;
      // elimino il nodo generatore
}
```

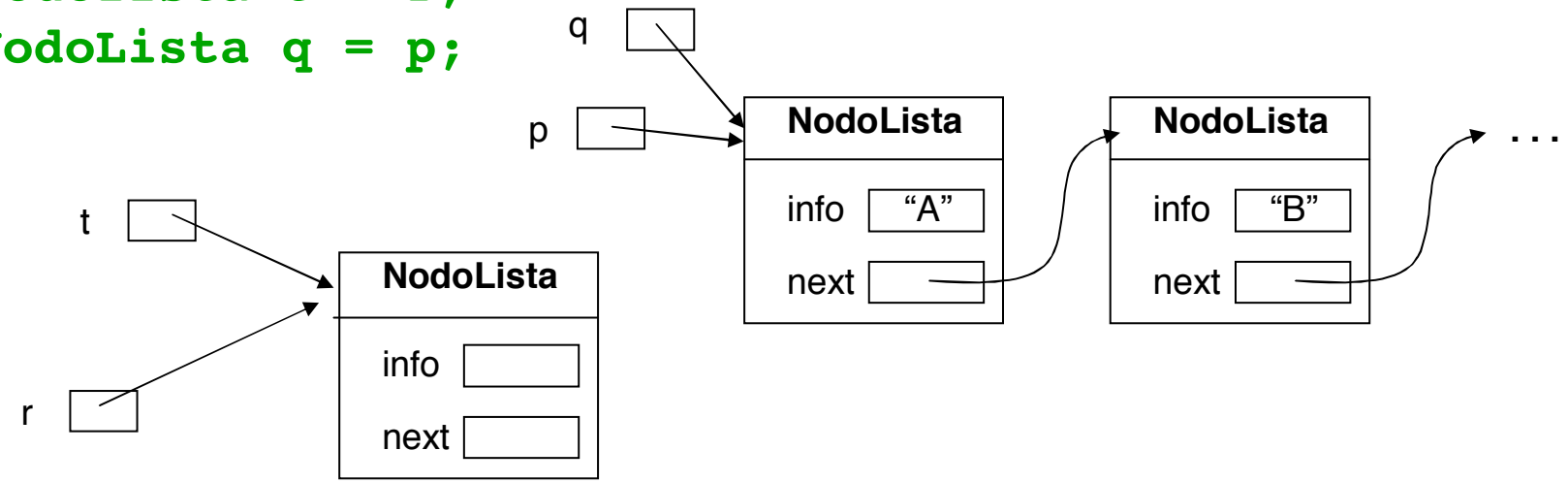
Nota bene

- Il metodo copia usa la tecnica del *nodo generatore*
- Il nodo generatore è un nodo ausiliario che viene anteposto alla lista che vogliamo costruire e da cui partono le operazioni di creazione dei nodi successivi
- Al termine della creazione della lista tale nodo viene rimosso e viene restituita la lista a partire dal nodo successivo – cioè dal primo "effettivo"
- Tale soluzione è usata per trattare uniformemente tutti gli elementi della lista, compreso il primo
- Si osservi infatti che il metodo è corretto anche nel caso in cui la lista su cui viene invocato sia la lista vuota

Creazione e collegamento di nodi . . .

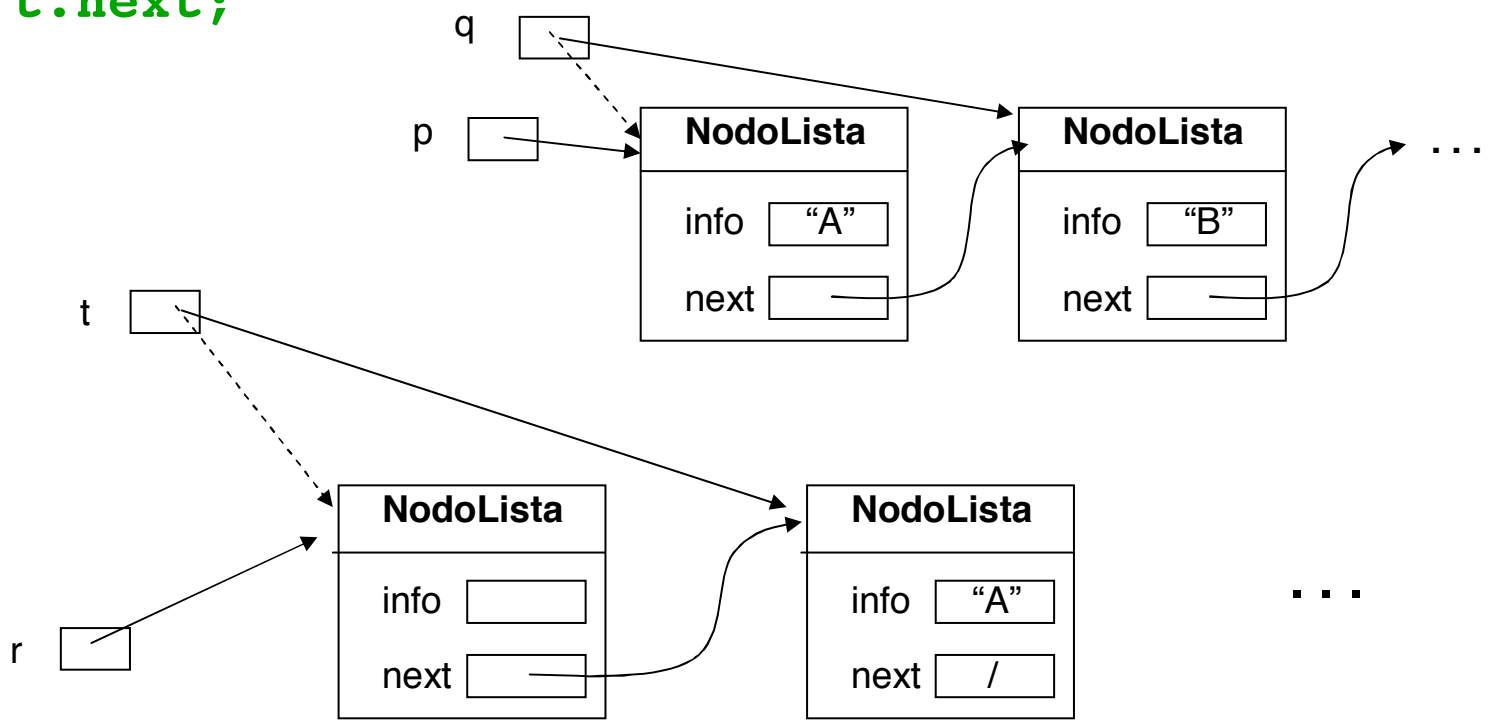



```
NodoLista r = new NodoLista(null,null);
Nodolista t = r;
NodoLista q = p;
```



. . . Creazione e collegamento di nodi

```
t.next = new NodoLista(q.info,null);
q = q.next;
t = t.next;
```



Gestione nodi tramite posizione

```
// Inserimento in posizione data
public static NodoLista
            add(NodoLista p, int k, String x) {
  if (k==0)
    p = new NodoLista(x,p);
  else {
    NodoLista q = p;
    for (int i=0; i<k-1 && q!=null; i++)
                q = q.next;
    if (q!=null)
            q.next = new NodoLista(x,q.next); }
  return p; }
```

Accesso ad un elemento

```
// ricerca di un elemento in posizione data
// restituisce null se non lo trova
public static String cerca(NodoLista p, int k) {
  String s;
  NodoLista q = p;
  for (int i=0; i<k && q!=null; i++)
     q = q.next;
  if (q!=null)
     s = q.info;
  else
     s = null;
  return s;
}
```

Accesso alla posizione di un elemento

```
// ricerca della posizione di un elemento
// (restituisce -1 se non lo trova)
public static int
            posizione(NodoLista p, String x) {
      NodoLista q = p;
      int n = -1;
      int i = 0;
      while (q!=null) {
            if (q.info.equals(x)) n = i;
            q = q.next;
            i++; }
      return n; }
```

Modifica di un elemento

```
// modifica l'elemento in posizione k
// (se esiste !)
public static void
     modifica(NodoLista p, int k, String x) {
  NodoLista q = p;
  for (int i=0; i<k && q!=null; i++)
          q = q.next;
  if (q!=null)
          q.info = x;
}
```

Esercizi

- Scrivere i metodi necessari per costruire una struttura collegata per rappresentare una data sequenza di stringhe
  - disponibile come un array di stringhe
  - mantenendo l'ordine degli elementi dal primo all'ultimo
  - invertendo l'ordine degli elementi dall'ultimo al primo

Implementazione ricorsiva delle operazioni comuni sulle liste

- Ricordiamo la definizione della classe NodoLista

```
class NodoLista {
  public String info;
  public NodoLista next;
  public NodoLista(String i, NodoLista n) {
     info = i; next = n;
  }
}
```

- Questa è una definizione induttiva degli oggetti NodoLista che offre, in maniera del tutto naturale e immediata, la possibilità di definire le operazioni sulle strutture collegate in modo ricorsivo

Inserimento di un elemento . . .

```
// Inserimento tramite posizione
public static NodoLista
            addR(NodoLista p, int k, String x) {
  NodoLista q;
  if (k==0)
     q = new NodoLista(x,p);
  else {
     p.next = addR(p.next,k-1,x);
     q = p;
  }
  return q;
}
```

. . . Inserimento di un elemento

```
// Inserimento tramite ricerca
public static NodoLista
      addR(NodoLista p, String y, String x) {
  NodoLista q;
  if (p.info.equals(y))
     q = new NodoLista(x,p);
  else {
     p.next = addR(p.next,y,x);
     q = p;
  }
  return q;
}
```

Eliminazione di un elemento . . .

```
// Eliminazione tramite posizione
public static NodoLista
                  eliminaR(NodoLista p, int k) {
  NodoLista q;
  if (k==0) q = p.next;
  else {
     p.next = eliminaR(p.next,k-1);
     q = p;
  }
  return q;
}
```

. . . Eliminazione di un elemento

```
// Eliminazione tramite ricerca
public static NodoLista
            eliminaR(NodoLista p, String x) {
  NodoLista q;
  if (p.info.equals(x)) q = p.next;
  else {
     p.next = eliminaR(p.next,x);
     q = p;
  }
  return q;
}
```

Ricerca di un elemento

```
public static boolean
            cercaR(NodoLista p, String x) {
  boolean trovato = false;
  if (p != null)
     if (p.info.equals(x))
           trovato = true;
     else trovato = cercaR(p.next,x);
  return trovato;
}
```

Copia di una lista

```
public static NodoLista copiaR(NodoLista p) {
      NodoLista q = null;
      if (p != null){
            q = copiaR(p.next);
            q = new NodoLista(p.info,q);
      }
      return q;
}
```

Stampa di una lista - versione iterativa

```
public static void stampaLista(NodoLista p) {
     while (q != null) {
          System.out.print(q.info + " ");
          q = q.next;
     }
     System.out.println();
}
```

Stampa di una lista - versione ricorsiva

```
public static void stampaListaR(NodoLista p) {
     if (p == null)
          System.out.println();
     else {
          System.out.print(p.info + " ");
          stampaListaR(p.next);
     }
}
```

Concatenazione di due liste – iterativa . . .

```
public static NodoLista
             append(NodoLista p1, NodoLista p2) {
  NodoLista r = new NodoLista(null,null);
  // nodo generatore
  NodoLista t = r;
  NodoLista q = p1;
  while (q != null) {
     t.next = new NodoLista(q.info,null);
     q = q.next;
     t = t.next; }
  . . .
```

. . . Concatenazione di due liste - iterativa

```
. . .
      q = p2;
   while (q != null) {
      t.next = new NodoLista(q.info,null);
      q = q.next;
      t = t.next; }
   return r.next;
}
```

Concatenazione di due liste - ricorsiva

```
public static NodoLista
    appendR(NodoLista p1, NodoLista p2) {
  NodoLista ris
  if (p1 == null) ris = copia(p2);
  else {
     NodoLista ris = new NodoLista();
     ris.info = p1.info;
     ris.next = appendR(p1.next,p2); }
  return ris;
}
```

Inversione di una lista – iterativa . . .

```
public static NodoLista
                  invertiLista(NodoLista p) {
// p riferimento alla lista da invertire
// ris riferimento alla lista invertita
NodoLista ris = null;
while (p != null) {
  // suggerimento: considerare le operazioni
  // nel ciclo come se operassero
  // su due liste p e ris:
  // - estrazione del primo elemento di p
  // - inserimento in testa alla lista ris
. . .
```

. . . Inversione di una lista - iterativa

```
. . .
// suggerimento: considerare le operazioni
// nel ciclo come se operassero
// su due liste p e ris:
  // - estrazione del primo elemento di p
  // - inserimento in testa alla lista ris
     NodoLista aux = p.next;
     p.next = ris;
     ris = p;
     p = aux; }
  return ris;
}
```

Inversione di una lista - ricorsiva

```
public static NodoLista
                  invertiListaR(NodoLista p) {
  NodoLista q;
  if (p==null) q = null;
  else {
     NodoLista aux = invertiListaR(p.next);
     q = aggiungiInCoda(aux,p.info);
  }
  return q;
}
```

Esercizio

- Definire il metodo `aggiungiInCoda` previsto dalla definizione del metodo ricorsivo `invertiListaR` proposto in precedenza