Corso di Laurea Ingegneria Informatica
Fondamenti di Informatica 2

Dispensa E08

Soluzione Esercizi

F. Gasparetti, C. Limongelli

Marzo 2008

Verifica presenza di elementi comuni…

V1 - Date due liste di stringhe scrivere un metodo che verifichi la presenza di elementi comuni ad entrambe

- **La classe per le liste di stringhe:**

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

… Verifica presenza di elementi comuni…

- Bisogna verificare che esiste almeno un elemento nella prima lista che è uguale ad un elemento nella seconda lista
- Verifica esistenziale sulla prima lista
- Verifica esistenziale sulla seconda lista
  - Fissato un elemento della prima lista, verificare che sia presente nella seconda
- Il metodo isMember verifica se una stringa s è presente in una lista l
- Il metodo nodiComuni esegue la verifica sulla prima lista

Il metodo isMember

```
/* data una stringa e una lista verifica che
   la stringa sia presente in qualche nodo
   della lista: verfica esistenziale sulla
   seconda lista*/
public static boolean
            isMember(String s, NodoLista l){
   boolean esiste;
   esiste = false;
   while (l!=null && !esiste){
      if (s.equals(l.info))
            esiste = true;
      l = l.next;
   }
   return esiste;
}
```

… Verifica presenza di elementi comuni…

- Verifica esistenziale sulla prima lista

```
public static boolean
    nodiComuni(NodoLista l1, NodoLista l2){

    boolean comune;
    comune = false;
    while (l1 != null && !comune){
         if (isMember(l1.info,l2))
              comune = true;
         l1 = l1.next;
    }
    return comune;
}
```

Costruzione lista nodi comuni…

B1 - Date due liste di stringhe restituire una nuova lista che contenga gli elementi comuni ad entrambe le liste date

- **Algoritmo:**
  - Costruzione con un nodo generatore:
  - Creo il primo nodo fittizio
  - Finche' ci sono elementi da esaminare nella prima lista
    - **se l'elemento che sto esaminando e' presente nella seconda lista**

      allora creo un nuovo nodo che lo contiene e attacco il nuovo nodo in coda alla lista che sto costruendo
  - Restituisco il secondo nodo della lista costruita

…Costruzione lista nodi comuni…

```
public static NodoLista listaNodiComuni
                    (NodoLista l1, NodoLista l2){
   NodoLista n,app;
   // costruzione con record generatore
   n = new NodoLista(null,null);
   app = n;
   while (l1 !=null){
       if (isMember(l1.info,l2)){
              app.next = new NodoLista(l1.info,null);
              app = app.next;
       }
       l1 = l1.next;
   }
   return n.next;
}
```

Merge liste …

B2 - Date due liste di stringhe, ordinate in modo non decrescente, scrivere un metodo merge che effettui l'operazione di fusione restituendo una nuova lista ordinata, contenente tutti gli elementi delle due liste

- **L'algoritmo è identico all'algoritmo di fusione di due array ordinati, cambia solo la modialita' di scansione delle sequenze di elementi**
- **Il confronto tra le stringhe viene realizzato dal metodo maggiore**

Il metodo per confrontare le stringhe

- Invece di implementare il confronto lessicografico tra stringhe come visto in precedenza, usiamo il metodo compareTo definito nella classe String:

```
public static boolean maggiore(String s1,
String s2){
    boolean res;
    if (s1.compareTo(s2)<=0) // s1<=s2
          res = false;
    else res = true;
    return res;
}
```

… Merge liste …

```
public static NodoLista mergeListe
                    (NodoLista l1, NodoLista l2){
NodoLista n,app;//per la nuova lista

n = new NodoLista(null, null);
app = n;
while(l1 !=null && l2 !=null)
   // se l1.info <= l2.info
   if (maggiore(l2.info,l1.info)){
       //creo un nodo con il contenuto di l1
       app.next = new NodoLista(l1.info,null);
       app = app.next;
       // mi posiziono sul prossimo nodo di l1
       l1 = l1.next;
   }
```

… Merge liste …

```
else{ //l1.info>l2
    // creo un nodo con il contenuto di l2
    app.next = new NodoLista(l2.info,null);
    app = app.next;
    // mi posiziono sul prossimo nodo di l1
    l2 = l2.next;
}
// quando termina il while una delle due
liste e' null
```

… Merge liste

```
if (l1==null)
  while(l2!=null){
      app.next = new NodoLista(l2.info,null);
      app = app.next;
      l2 = l2.next;
  }
else
  while(l1!=null){
      app.next = new NodoLista(l1.info,null);
      app = app.next;
      l1 = l1.next;
  }
return n.next;
}
```

Il metodo ricercaSequenza…

B3 - Scrivere il metodo ricercaSequenza che, date due liste di stringhe la e lb, verifichi se esiste una sequenza di elementi in la corrispondente agli elementi in lb

Esempio: data la = (A B B C D E) e lb = (B B C) la risposta dovrà essere true, mentre per la = (A B B C D E) e lb = (A B C) la risposta dovrà essere false

… Il metodo ricercaSequenza…

- Considero il generico elemento della lista la come la testa della lista che esamino attualmente.
- Due liste sono uguali se i primi nodi della lista la coincidono con tutti i nodi della lista lb
- Esempio:
  - la = (A B B C D E) e lb = (B B C)
  - Se analizzo la lista la a partire da B osservo che tutti I nodi della lista lb coincidono con i primi nodi di la
- Ricorsione su la

… Il metodo ricercaSequenza…

- **Passo base**
  - la vuota → non ho individuato la sottosequenza
- **Passo ricorsivo**
  - La sottosequenza viene individuata
  - se i primi elementi di la coincidono con tutti gli elementi di lb
  - oppure
  - la sottosequenza viene individuata sulla lista che parte dal nodo successivo ad la

… Il metodo ricercaSequenza…

```
public static boolean
ricercaSequenza(NodoLista l1, NodoLista l2){
/* verifica esistenziale */
   boolean trovata;

   if (l1 == null) trovata = false;
   else
        trovata = uguali(l1,l2) ||
                  ricercaSequenza(l1.next,l2);
   return trovata;
}
```

… Il metodo uguali…

- Date due liste l1 e l2, verifica che i primi elementi di l1 siano uguali a tutti gli elementi di l2
- Ricorsione su entrambe le liste
- Passo base:
  - Se e' terminata la scansione di l2 allora ho esaminato con successo tutti gli elementi true
  - Se e' terminata la scansione di l1 allora l1 non puo' contenere tutti gli elementi di l2 quindi false
  - N.B.: I due passi base vanno necessariamente in questo ordine
- Passo ricorsivo
  - La ricerca ha successo se i primi due elementi delle liste sono uguali e se sono uguali le liste l1.next e l2.next

… Il metodo uguali…

```
public static boolean uguali
                (NodoLista l1, NodoLista l2){
    boolean sonouguali;

    /*passo base */
    if (l2 == null) sonouguali = true;
    else
    if (l1 == null) sonouguali = false;
    else
    /* passo ricorsivo */
       sonouguali = l1.info.equals(l2.info) &&
                              uguali(l1.next,l2.next);
    return sonouguali;
}
```

Eliminazione dei doppioni da una lista

R1 - Scrivere il metodo eliminaDoppioni che, data una lista di stringhe, restituisca una nuova lista ottenuta eliminando dalla lista data i doppioni

Esempio: data elle = (A A A B A B)
eliminaDoppioni(elle) restituisce la lista (A B)

❑ Tre versioni del problema

- **Creazione della nuova lista in modo iterativo modificando quella originale**
- **Creazione della nuova lista in modo iterativo senza modificare quella originale (come richiesto)**
- **Creazione della lista in modo ricorsivo senza modificare quella originale**

Creazione iterativa di una lista che modifica quella originale …

- Per ogni nodo l della lista
  - Si eliminano dal resto della lista (l.next) tutti i nodi che hanno il campo informazione uguale a l.info
- Eliminazione di tutti i nodi il cui campo informazione è uguale ad una data stringa (l.info).
  - Bisogna tenere un puntatore che referenzia il nodo precedente a quello da eliminare
- l.info non è una stringa qualsiasi, bensì la stringa contenuta nel primo nodo della lista
- Il metodo elimina non passa come parametri attuali l.info e l.next, ma l.info e l. In questo modo il primo nodo – che non può essere eliminato - serve per inizializzare il puntatore al nodo precedente

…Creazione iterativa di una lista che modifica quella originale …

```
//elimina tutte le occorrenze di s dalla lista l
// il cui primo nodo contiene proprio s
public static void elimina(String s, NodoLista l){
    NodoLista succ,prec;

    prec = l;
    succ = l.next;
    while (succ != null){
          if (s.equals(succ.info)){
                prec.next = succ.next;
                succ = prec.next;
          }
          else {
                succ = succ.next;
                prec = prec.next;
}  }  }
```

…Creazione iterativa di una lista che non modifica quella originale …

- Il metodo elimina va applicato ad ogni nodo della lista (che via via viene modificata)

```
public static void
                eliminaTutti(NodoLista l){
   NodoLista app;
   while (l != null){
        /* elimino tutti i doppioni di
           l.info dal resto della lista */
        elimina(l.info,l);
        l = l.next;
   }
}
```

…Creazione iterativa di una nuova lista che modifica quella originale …

- Banalmente si applica il metodo eliminaTutti alla copia della lista data in input:

```
public static NodoLista eliminaDoppioni(NodoLista l){
    NodoLista n,app; //nuova lista con doppioni
    n = new NodoLista(null,null);
    app = n;
    while (l != null){
          app.next = new NodoLista(l.info,null);
          app = app.next;
          l = l.next;
    }
    n = n.next;
    return eliminaTutti(n.next);
}
```

Creazione ricorsiva di una lista che non modifica quella originale …

- **Passo base**
  - lista vuota: tutta la lista e’ stata scandita
- **Passo ricorsivo:**
  - vedo se il primo elemento e' gia' presente nel resto della lista che e’ stata creata eliminando tutti i doppioni.
  - se e' gia’ presente restituisco la nuova lista senza il primo elemento
  - altrimenti restituisco la lista nuova lista a cui e’ stato aggiunto in testa un nuovo nodo che contiene il primo elemento della lista che si sta esaminando

Creazione ricorsiva di una lista che non modifica quella originale …

```
public static NodoLista eliminaRIC(NodoLista l){
  NodoLista app;

  app = null;
  if (l != null){
     app = eliminaRIC(l.next);
     if (!isMember(l.info,app))
           app = new NodoLista(l.info,app);
  }
  return app;
}
```

Tipo di dato astratto Insieme

- **Tipo astratto Insieme = <S, F, C> con**
  - **S = {Ins, V, boolean}**
    - **Ins = dominio d'interesse**
    - **V= dominio di sostegno (elementi degli insiemi)**
  - **C = {Insieme_Vuoto}**
- **E l'insieme delle operazioni F e' costituito da:**
  - **Vuoto: Ins → boolean**
  - **Inserisci: Ins x V → Ins**
  - **Cancella: Ins x V → Ins**
  - **Contiene: Ins x V → boolean**

Rappresentazione del ADT Insieme. . .

- In questo caso la rappresentazione di Ins e' basata sulla classe `Lista`, ovvero una lista collegata dove non ammettiamo duplicazioni
- Generalizziamo `NodoLista` al caso di qualsiasi tipo (non primitivo) Java, defininendo il campo `info` con il tipo `Object`.

```
class NodoLista {
   Object info;
   NodoLista next;
   NodoLista(Object o, NodoLista n) {
      info=o; next=n;
   }
}
```

. . .rappresentazione del ADT Insieme. . .

❑Possiamo definire due realizzazioni:

- **Con side-effect e senza condivisione**:
  I metodi `inserisci` e `cancella` modificano direttamente l'oggetto su cui sono state richiamate
- **Funzionale dove si limita il side-effect nelle operazioni**
  consente un certo grado di condivisione di memoria senza cadere nel problema dell'interferenza

Rappresentazione con side-effect. . .

```
public class InsiemeSS {
 // primo elemento della lista
 private NodoLista inizio;

 public InsiemeSS() {
   inizio = null;
 }
 // realizzazione dell'op.ne Vuoto
 public boolean vuoto() {
   return inizio == null;
 }
 // realizzazione dell'op.ne Contiene
 public boolean contiene(Object e) {
    // uso un metodo ausiliario
   return appartiene(e, inizio);
 }
```

. . .rappresentazione con side-effect. . .

```
// realizzazione dell'op.ne Inserisci
public void inserisci(Object e) {
  if (!appartiene(e, inizio)) {
    inizio = new NodoLista(e,inizio);
  }
}
// realizzazione dell'op.ne Cancella
public void cancella(Object e) {
  // uso un metodo ausiliario cancella
  inizio = elimina(e, inizio);
}
// ritorna il primo elemento inserito
public Object scegli() {
  if (inizio == null) return null;
  else return inizio.info;
}
```

. . .rappresentazione con side-effect. . .

```
// uguaglianza
public boolean uguali(InsiemeSS ins) {
  boolean ret = true;
  NodoLista l;
  l = inizio;
  // tutti gli elementi di this sono in ins?
  while (ret && (l != null)) {
    if (!appartiene(l.info,ins.inizio))
      ret = false;
    l = l.next;
  }
  l = ins.inizio;
  // tutti gli elementi di ins sono in this?
  . . .
```

. . .rappresentazione con side-effect. . .

```
  while (ret && (l != null)) {
    if (!appartiene(l.info,inizio))
      ret = false;
    l = l.next;
  }
  return ret;
}

// true se l'oggetto e e' contenuto nella lista l
private static boolean appartiene(Object e,
                                  NodoLista l){
  if (l == null)
    return false;
  return(l.info.equals(e) || appartiene(e,l.next));
}
```

. . .rappresentazione con side-effect

```
// cancella l'oggetto e nella lista l
private static NodoLista elimina(Object e,
                                 NodoLista l) {
 NodoLista ret;
 if (l == null)
   ret = null;
 else if (l.info.equals(e))
   ret = l.next;
 else {
   l.next = elimina(e,l.next);
   ret = l;
 }
 return ret;
}
```

Il metodo equals di Object

- Il metodo `equals` richiamato sugli oggetti `info` memorizzati nell'insieme dai metodi `appartiene` e `cancella` non e' quello di Object ma quello ridefinito (*overridden*) nella classe più specifica a cui appartiene l'oggetto denotato da tale riferimento (ad esempio: `String`, `Integer`, etc)
- Questo è dovuto al meccanismo di *late binding* adottato da Java

Rappresentazione funzionale. . .

```
public class InsiemeFC {
  private NodoLista inizio;

  public InsiemeFC() {
    inizio = null;
  }

  public boolean vuoto() {
    return inizio == null;
  }

  public boolean contiene(Object e) {
    return appartiene(e,inizio);
  }
  . . .
```

. . .rappresentazione funzionale. . .

```
public InsiemeFC inserisci(Object e) {
  InsiemeFC ret;
  if (appartiene(e,inizio))
    ret = this;
  else {
    /* si crea un nuovo oggetti InsiemeFC ma gli
  elementi NodoLista esistenti sono condivisi */
    InsiemeFC ins = new InsiemeFC();
    ins.inizio = new NodoLista(e,inizio);
    ret = ins;
  }
  return ret;
}

public InsiemeFC cancella(Object e) {
  InsiemeFC ret; . . .
```

. . .rappresentazione funzionale. . .

```
    if (!appartiene(e,inizio))
      ret = this;
    else {
      InsiemeFC ins = new InsiemeFC();
      // copia parziale
      ins.inizio = elimina(e,inizio);
      ret = ins;
    }
  }
  // ritorna il primo elemento inserito
  public Object scegli() {
    Object ret = null;
    if (inizio != null)
      ret = inizio.info;
    return ret;
  }
```

. . .rappresentazione funzionale. . .

```
public boolean uguali(InsiemeFC ins) {
 // identica al caso con side-effect
}

private static boolean appartiene(...){
  // identica al caso con side-effect
}

 // cancella l'oggetto e nella lista l
 private static NodoLista elimina(Object e,
                                  NodoLista l) {
  NodoLista ret;
  if (l == null)
    ret = null;
  . . .
```

. . .rappresentazione funzionale

```
  . . .
  else if (l.info.equals(e))
    ret = l.next;
  else {
    ret = new NodoLista(l.info,elimina(e,l.next));
  }
  return ret;
}
```

Il metodo elimina di InsiemeFC

- Per evitare il problema dell'interferenza con altre istanze di `InsiemeFC`,il metodo `elimina` duplica la lista fino al nodo contenente l'elemento da cancellare (escluso) e poi crea condivisione con i nodi che seguono l'elemento da cancellare
- Seppure alcuni elementi `NodoLista` possono essere condivisi a piu' insiemi, nel momento in cui si elimina un nodo B e viene aggiornato il campo `next` del nodo A che lo precede, viene creato ed aggiornato un nuovo oggetto distinto da A

Esercizio con InsiemeFC

- Analizzare le strutture collegate che rappresentano internamente gli insiemi `a`, `b`, `c` e `d`, ed in particolare le condivisioni di oggetti durante lo svolgimento del seguente codice.

```
InsiemeFC a, b, c;
Integer i1 = new Integer(1);
Integer i2 = new Integer(2);
a = new InsiemeFC();
a = a.inserisci(i1);
b = a;
c = a.inserisci(i2);
d = a.cancella(i1);
```