# Corso di Laurea Ingegneria Informatica

## Fondamenti di Informatica 1

**Dispensa E07**

# Esercizi
# Strutture Collegate e Oggetti

**C. Limongelli**

Marzo 2008

# Contenuti

- **Verifiche universali ed esistenziali su strutture collegate**
  - Verifica dell'esistenza di una stringa, in una struttura collegata, formata tutta da caratteri maiuscoli
  - Ricerca di un doppione in una lista
  - Verifica intersezione insiemi
- **Analisi e creazione di strutture collegate**
  - Costruzione di una lista in modo ordinato

# … Verifica esistenziale: schema risolutivo

```
boolean proprietaSoddisfatta;   // almeno un elemento
                                // soddisfa la proprietà

proprietaSoddisfatta = false;
  ... altre inizializzazioni ...

  finchè non trovo un elemento che soddisfa la
   proprietà
  while (!proprietaSoddisfatta && sequenza non
                                        terminata) {
    if (l'elemento corrente soddisfa la proprietà)
          proprietaSoddisfatta = true;
    ... accedi al prossimo elemento ...
  }
  ... altre elaborazioni ...
```

# … Verifica esistenziale: schema risolutivo

```
boolean proprietaSoddisfatta;   // almeno un elemento
                                // soddisfa la proprietà

proprietaSoddisfatta = false;
  ... altre inizializzazioni ...

  finchè non trovo un elemento che soddisfa la
   proprietà
  while ( !proprietaSoddisfatta && p != null ) {

    if (l'elemento corrente soddisfa la proprietà)
          proprietaSoddisfatta = true;
    ... accedi al prossimo elemento ...
   }
  ... altre elaborazioni ...
```

L'ordine delle condizioni nel while è indifferente?

# Verifica esistenziale: stringa formata tutta da caratteri maiuscoli…

- Verifica dell'esistenza, in una struttura collegata, di una stringa formata tutta da caratteri maiuscoli

- Struttura NodoLista:

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

# …Verifica esistenziale: stringa formata tutta da caratteri maiuscoli…

```
/* verifica esistenziale sugli elementi della lista */
public static boolean stringaTuttaMaiuscola
                                        (NodoLista a){
   NodoLista p;
   boolean trovato;
   p = a;
   trovato = false;
   while (p != null && !trovato){
       if (tutteMaiuscole(p.info))
              trovato = true;
       p = p.next;
   }
   return trovato;
}
```

- Quale valore viene restituito se la lista a è vuota?
- Cosa succede se non si usa la variabile d'appoggio p?

# …Verifica esistenziale: stringa formata tutta da caratteri maiuscoli…

- **Verifiche di correttezza:**
  - (Giulia Alessandra Gabriella Sabrina Paola)
    - false
  - (Giulia ALESSANDRA Gabriella Sabrina Paola)
    - true
  - (GIULIA Alessandra Gabriella Sabrina Paola)
    - true
  - (Giulia Alessandra Gabriella Sabrina PAOLA)
    - true
  - (ALESSANDRA)
    - true
  - ( )
    - false

# Verifica esistenziale: Ricerca di un doppione in una lista…

- Per ogni nodo della lista bisogna controllare se il valore del suo campo info è presente nei nodi successivi della lista
- Se viene trovato un tale valore l'iterazione termina con esito positivo

# …Verifica esistenziale: Ricerca di un doppione in una lista…

- Il metodo EsisteDoppio:

```
public static boolean esisteDoppio(NodoLista a){
    NodoLista p;
    boolean trovato; //trovato il doppione

    p = a;
    trovato = false;
    while (p != null && !trovato){
          if (presente(p.info,p.next))
                trovato = true;
          p = p.next;
    }
    return trovato;
}
```

# …Verifica esistenziale: Ricerca di un doppione in una lista…

- Il metodo presente:

```
public static boolean presente(String s, NodoLista a){
    boolean trovato;
    NodoLista p;

    trovato = false;
    p = a;
    while (p != null && !trovato){
          if (s.equals(p.info))
                trovato = true;
          p = p.next;
    }
    return trovato;
}
```

# …Verifica esistenziale: Ricerca di un doppione in una lista…

- **Verifiche di correttezza:**

- (Giulia Alessandra Paola Sabrina Paola)
  - true
- (Giulia ALESSANDRA Gabriella Sabrina Alessandra)
  - false
- (Paola)
  - false
- ( )
  - false
- (0 1 2 3 4 5 6 6)
  - true

# Verifica universale: schema risolutivo

```
boolean proprietaSoddisfatta;

/* assumo che tutti gli elementi soddisfano
   la proprietà */
   proprietaSoddisfatta = true;

 ... altre inizializzazioni ...

 finchè non trovo un elemento che non soddisfa
 la proprieta
   while (proprietaSoddisfatta &&
                        la sequenza non è finita){
     if (l'elemento corrente non
                 soddisfa la proprietà)
                          proprietàSoddisfatta = false;
     ... accedi al prossimo elemento ...
    }

     ... altre elaborazioni ...
```

# Verifica universale: intersezione di insiemi…

- Date tre liste a, b e c i cui campi informazione rappresentano elementi rispettivamente di tre insiemi A, B e C, verificare che gli elementi di C siano "tutti e soli" gli elementi dell'intersezione di A e B.
- Bisogna verificare che :
  - Ciascun elemento di C è presente sia in A che in B :

    gli elementi di C sono comuni ad A e B e quindi appartengono alla loro intersezione

    *NON BASTA !*

    C *potrebbe essere un sotto-insieme dell'insieme intersezione*)
  - Gli elementi di A che sono anche in B devono necessariamente essere anche in C

# …Verifica universale: intersezione di insiemi…

- Per ogni elemento a di A bisogna verificare che

$$a \in B \Rightarrow a \in C$$

- Quindi tutti gli elementi di A devono soddisfare la suddetta proprietà che si può riscrivere come:

$$!(a \in B) \,||\, (a \in C)$$

- Nella verifica universale l'iterazione si ferma quando la proprietà non è soddisfatta cioè quando

$$!\,(!(a \in B) \,||\, (a \in C)\,) = (a \in B) \;\&\&\; !(a \in C)\,)$$

# Tutti gli elementi di C siano presenti sia in A che in B

- Verifica l'appartenenza degli elementi della lista c sia alla lista a che alla lista b
- Verifica universale sugli elementi di c

```
public static boolean verificaAppartenenza
              (NodoLista a, NodoLista b, NodoLista c){
   NodoLista pc;
   boolean appartengono;
   appartengono = true;
   pc = c;
   while (pc != null && appartengono){
       appartengono = presente(pc.info,a) &&
                                  presente(pc.info,b);
       /* equivalente a
       if (!(presente(pc.info,a) && presente(pc.info,b)))
                    appartengono = false; */
       pc = pc.next; }
       return appartengono;
}
```

## Se esistono elementi di A che sono in B, allora devono anche essere in C.

- Verifica universale sugli elementi di A

  !(!(a ϵ B) ll (a ϵ C) ) = (a ϵ B) && !(a ϵ C) )

```
…
NodoLista pa;
pa = a;
proprieta = true;
while (pa != null  && proprieta){
   if (presente(pa.info, b) && !presente(pa.info, c))
       proprieta = false;
   pa = pa.next;
}
…
```

# Il metodo verificaIntersezione

```
public static boolean verificaIntersezione
            (NodoLista a, NodoLista b, NodoLista c){
  boolean proprieta;
  NodoLista pa;
  pa = a;
  proprieta = true;
  while (pa != null  && proprieta){
      if (presente(pa.info, b) && !presente(pa.info, c))
            proprieta = false;
      pa = pa.next;
  }
  return proprieta && verificaAppartenenza(a,b,c);
}
```

# Intersezione di insiemi - Verifica di correttezza

- a: (A B C D E X Y Z)
- b: (X Y Z)
- c: (X Y)
- gli elementi di c sono tutti sia in a che b? true

- a: (A B C D E X Y Z)
- b: (X Y Z)
- c: (X Y)
- gli elementi di c sono tutti-e-soli gli elementi dell'intersezione di a e b? false

- c: (X Y Z)
- gli elementi di c sono tutti-e-soli gli elementi dell'intersezione di a e b? true

# Esercizi . . .

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

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

**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**

# . . . Esercizi

**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 )**

**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**

# Riprendiamo l'oggetto Libro

- è sempre definito allo stesso modo
- Autore (autore)
- Titolo (titolo)
- Numero di pagine (pagine)

```
private String autore;
private String titolo;
private int pagine;

// Costruttore della classe
public Libro (String a, String t, int p){

    this.autore = a;
    this.titolo = t;
    this.pagine = p;
}
```

# … La classe Libro

- I metodi della classe:

```
public String getAutore(){
    return this.autore;
}

public String getTtitolo() {
    return this.titolo;
}

public int getPagine() {
    return this.pagine;
}

public  String toString(){
return    ("AUTORE:    " + this.autore + "\n" +
           "TITOLO:    " + this.titolo + "\n"+
           "N. PAGINE: " + this.pagine + "\n");
}
```

# Esercizi

- **Data la classe Libro per rappresentare oggetti libro con il nome dell'autore, il titolo e il numero di pagine e con i relativi metodi d'istanza**
  - **scrivere un metodo che, ricevendo come parametro un array di oggetti Libro, calcola e restituisce una struttura collegata con gli stessi oggetti Libro nello stesso ordine**
  - **scrivere un metodo che, ricevendo come parametro una struttura collegata di oggetti Libro, e il nome di un autore, verifica se nell'elenco esiste almeno un libro dell'autore dato**

# La lista di oggetti Libro

- La classe che definisce l'oggetto ListaLibri - un elemento della lista di libri:

```
class ListaLibri {
  public Libro info;
  public ListaLibri next;

  public ListaLibri(Libro s, ListaLibri n){
     info = s;
     next = n;
  }
}
```

# Creazione di una lista di oggetti Libro

- **Supponiamo di aver definito un metodo**

  **ListaLibri creaElenco();**

  - **Trasformando un array di oggetti di tipo libro in una lista**
  - **Creando appositamente una lista di oggetti di tipo ListaLibri come segue:**

```
l = new Libro("Esopo","Le storie dell'asino",20);
p.next = new ListaLibri(l,null);
p = p.next;
l = new Libro("Italo Calvino", "Il visconte
                                dimezzato",158);
p.next = new ListaLibri(l,null);
p = p.next;
……
return a.next;
```

# Stampa di una lista di oggetti Libro

```
public static void stampaElenco(ListaLibri elenco) {
   ListaLibri p;
   p = elenco;
   System.out.println("================================");
   System.out.println("======= ELENCO LIBRI =========");
   System.out.println("================================");
   while (p != null){
       System.out.println(p.info.toString());
       p = p.next;
   }
}
```

# Esempi

- Assumendo che esista una classe ProvaLibri che crea e restituisce una struttura collegata di oggetti Libro definire i seguenti metodi per
  - Restituire il libro con il maggior numero di pagine
  - Verificare se nella struttura collegata di oggetti Libro, esiste almeno un libro di un autore (dato in input)
  - Creare e restituire un nuovo elenco con i tutti i libri che hanno un determinato autore (dato in input)
  - Dato un elenco di libri, ordinato secondo il nome dell'autore, inserire un libro in modo che l'elenco rimanga ordinato rispetto all'autore

# Restituisce il libro con il maggior numero di pagine

```
public static Libro maxPagine(ListaLibri a){
      int max;
      Libro libroMax; // libro con max. numero di pag.
      ListaLibri p;
      p = a;

      max = p.info.getPagine();
      libroMax = p.info;
      p = p.next;
      while (p != null){
            if ( p.info.getPagine() >
                                    libroMax.getPagine())
                  libroMax = p.info;
            p = p.next;
      }
      return libroMax;
  }
```

# Verifica se nella lista di oggetti Libro, esiste almeno un libro di un dato autore

- Verifica esistenziale

```
public static boolean esisteAutore(ListaLibri e,
                                   String autore){
    boolean esisteAutore;
    esisteAutore = false;
    while (e != null && !esisteAutore){
          if (e.info.getAutore().equals(autore))
                esisteAutore = true;
          e = e.next;
    }
    return esisteAutore;
}
```

# Creazione e restituzione di un sottoinsieme di libri

- Crea e restituisce un nuovo elenco con i tutti i libri di un determinato autore (dato in input)

```
public static ListaLibri stessoAutore(ListaLibri e,
                                      String autore){
   ListaLibri nuovo,app;
   //creazione con record generatore
   nuovo = new ListaLibri(null, null);
   app = nuovo;
   while (e != null ){
       if (e.info.getAutore().equals(autore)){
             app.next = new ListaLibri(e.info,null);
             app = app.next; }
       e = e.next;
   }
   return nuovo.next;
}
```

# Inserimento ordinato di libri nell'elenco

- Dato un elenco di libri, ordinato secondo il nome dell'autore (non decrescente), inserisce un nuovo libro in modo che l'elenco rimanga ordinato rispetto all'autore.

# Ordinamento lessicografico di stringhe

- **Esempi:**
  - cassa < forte
  - cassa = cassa
  - cassa < casse
  - cassa > cappa
  - cassa < cassaforte
- **In generale:**
  - Si confrontano i primi caratteri delle due stringhe
    - Se sono uguali si passa ad esaminare i due caratteri successivi
    - Altrimenti si ha un criterio per decidere il carattere minore nell'ordinamento lessicografico

# …Ordinamento lessicografico…

- Siano s1 e s2 due stringhe da confrontare

- Se s1="" e s2 = "" allora s1 = s2
- Se s1="" e s2 <> "" allora s1 < s2
- Se s1<> "" e s2 = "" allora s1 > s2
- Si confrontano i primi due caratteri delle stringhe: c1 e c2
  - Se c1 > c2 allora s1 > s2
  - Se c1 < c2 allora s1 < s2
  - Se c1= c2 allora si esamina l'ordinamento lessicografico delle sottostringhe ottenute da s1 e s2 avendo tolto il primo carattere

# Codifica dell'ordinamento lessicografico

```
public static boolean maggiore(String s1, String s2){
   boolean res;

   if (s1.length() == 0 && s2.length() == 0)res = true;
   else
       if (s1.length() == 0 && s2.length() > 0)res = false;
   else
       if (s1.length() > 0 && s2.length() == 0)res = true;
   else // caso generale:  stringhe entrambe non vuote
       if (s1.charAt(0) == s2.charAt(0) )
              res = maggiore(s1.substring(1),s2.substring(1));
       else
           if (s1.charAt(0) > s2.charAt(0) )
              res = true;
           else
              res = false;
   return res;
}
```

# Alternativa: il metodo delle api di Java

- public int compareTo(String anotherString)
- Compares two strings lexicographically.
- The comparison is based on the Unicode value of each character in the strings.
- The character sequence represented by this String object is compared lexicographically to the character sequence represented by the argument string.
- The result is a negative integer if this String object lexicographically precedes the argument string.
- The result is a positive integer if this String object lexicographically
- follows the argument string.
- The result is zero if the strings are equal;

- s1.compareTo(s2) >0 => s1>s2
- s1.compareTo(s2) <0 => s1<s2
- s1.compareTo(s2) =0 => s1=s2

# Il metodo per l'inserimento ordinato…

```
public static ListaLibri inserisci
                    (ListaLibri elenco, Libro libro){
   ListaLibri app;
   app = elenco;

   if (maggiore(app.info.getAutore(),
                                libro.getAutore()))
   /* devo aggiungere in prima posizione e modificare la
   testa della lista */
       elenco = new ListaLibri(libro, elenco);
   else {
       /* devo aggiungere in mezzo o in coda */
       while (app.next !=null &&
                    !(maggiore(app.next.info.getAutore(),
                                libro.getAutore())))
             app = app.next;
       app.next = new ListaLibri(libro,app.next);
   }
   return elenco;
}
```

# Altri esercizi

- Definizione ricorsiva del metodo `inserisci`
- Eliminazione di un libro dall'elenco
- Ordinamento dei libri rispetto al numero crescente di pagine
- Merge di due liste di libri ordinate secondo l'autore
- Distinguere gli attributi nome, cognome