Corso di Laurea Ingegneria Informatica

Fondamenti di Informatica 1

Dispensa E01

Esercizi su array di array

C. Limongelli

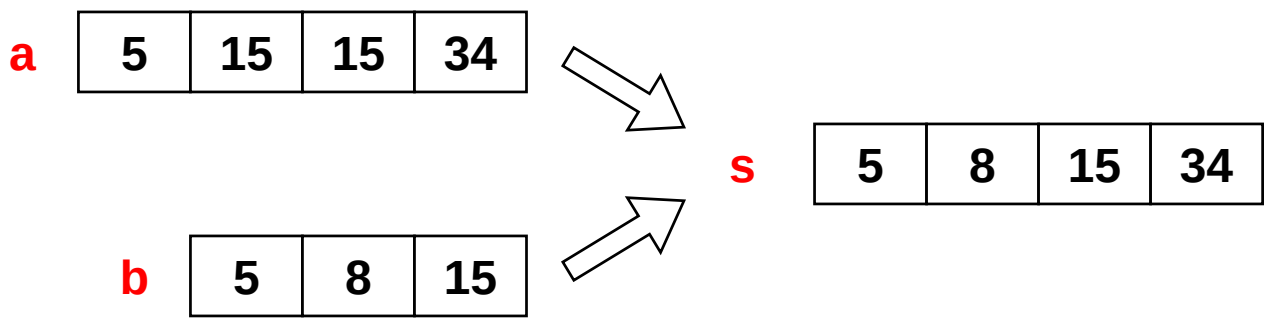
Febbraio 2008

Contenuti

- Fusione di due array ordinati, con eliminazione di duplicati
- Verifica array bidimensionale
- Generazione di un array con numeri disposti a spirale

Esercizio

- **Implementare la fusione con rimozione di duplicati di due array ordinati in modo non decrescente**
  - **ciascuna delle sequenze iniziali può contenere duplicati**
  - **Gli array sono non vuoti (almeno un elemento)**
  - **Esempio:**



I test del problema

```
System.out.println(toString(fusioneSenzaDuplicati
    (new int[] {0, 1, 4, 5}, new int[] {3, 6, 10})));

System.out.println(toString(fusioneSenzaDuplicati
    (new int[] {0, 1, 4, 5}, new int[] {0, 2, 10})));

System.out.println(toString(fusioneSenzaDuplicati
    (new int[] {0, 1, 4, 5}, new int[] {0, 1, 4})));

System.out.println(toString(fusioneSenzaDuplicati
    (new int[] {0, 1, 4}, new int[] {0, 1, 4})));

System.out.println(toString(fusioneSenzaDuplicati
    (new int[] {1, 1, 1}, new int[] {1, 1, 2})));

System.out.println(toString(fusioneSenzaDuplicati
    (new int[] {1, 2, 3}, new int[] {3, 4, 5})));

System.out.println(toString(fusioneSenzaDuplicati
    (new int[] {1}, new int[] {1})));
```

Algoritmo…

- Prima di inserire un elemento (di a o di b) in s, bisogna controllare che lo stesso elemento non compaia già in s
- Sia is l'indice di s in cui verrà memorizzato il prossimo elemento
- Sia a[ia] (o b[ib]) l'elemento che si vorrebbe inserire in s
- Bisogna controllare che l'elemento s[is-1] sia diverso da a[ia] (o b[ib]), altrimenti non si deve inserire, ma comunque si deve incrementare l'indice ia (o ib)
  - La prima volta che inserisco un elemento in posizione s[0], non posso fare il controllo su s[-1], quindi devo considerare a parte il primo inserimento. Come?
  - Controllando che is non sia uguale a 0
- Se l'elemento non viene inserito in s, l'indice relativo, is, non viene incrementato

...Algoritmo...

```
/* supponiamo is = ia = ib = 0 */
/* prima fase della fusione: inserisci da a e b */
while (ia<a.length && ib<b.length) {

/* inserisci in s il più piccolo tra
    a[ia] e b[ib] se non esiste gia' in s */
    if (a[ia]<b[ib]) {
      //provo ad inserire a[ia] in s
      if (is==0 || a[ia] != s[is-1]){
        s[is] = a[ia];
        is++;
      }
      ia++;
    }
}
```

…Algoritmo…

```
/* prima fase della fusione: inserisci da a
e b */
while (ia<a.length && ib<b.length) {
    if {
        ...
    } else { // b[ib] <= a[ia]
        if (is ==0 || b[ib] != s[is-1]){
            s[is] = b[ib];
            is++;
        }
        ib++;
    }
}
```

Perché quando is = 0 l'accessso a s[is-1] non Produce arrayOutOfBoundsException?

…Algoritmo…

- Anche nella seconda fase della fusione, quando la scansione di uno dei due array è terminata e bisogna inserire il resto dell'altro array, si devono fare le stesse considerazioni
- poiché gli array **a** e **b** non sono vuoti, ci sarà almeno un elemento in **s**, quindi non bisogna più controllare che l'indice **si** sia **0**
- Se la scansione di **b** è terminata:

```
/* inserisci da a */
while (ia<a.length) {
      if (s[is-1] != a[ia]){
            s[is] = a[ia];
            is++;
      }
      ia++;
}
```

…Algoritmo…

- **Analogamente per l'inserimento da b:**

```
/* inserisci da b */
/* l'array deve essere non vuoto perche'
deve aver    inserito dal ciclo precedente
almeno un elemento in s */

while (ib<b.length) {
   if(s[is-1] != b[ib]){
        s[is] = b[ib];
        is++;
   }
   ib++;
}
```

Considerazioni sulla lunghezza del nuovo array . . .

- L'array s è sicuramente non vuoto
- Avrà lunghezza <= alla somma della lunghezza di a e della lunghezza di b
- La lunghezza sarà uguale alla somma delle lunghezze solo se gli elementi di a e di b sono tutti distinti

. . . Considerazioni sulla lunghezza del nuovo array

- Inizialmente si crea un array di appoggio di lunghezza pari alla somma delle lunghezze di **a** e di **b**
- Dopo l'inserimento degli elementi in **s**, **is** (che rappresenta l'indice del prossimo elemento in cui si può inserire in **s**) è pari al numero di componenti memorizzate fino a quel momento in **s**, cioè è pari alla lunghezza che dovrà avere **s**
- Si crea l'array definitivo con lunghezza pari a **is** e si ricopiano gli **is** elementi dell'array di appoggio in quello definitivo

Gestione della lunghezza del nuovo array

```
int[] s;    // risultato della fusione di a e b
int[] s1;   // risultato definitivo della fusione
            // di a e b

    …
//copiare i valori significativi di s in un nuovo
//array s1;
// si può anche usare System.arraycopy(..);
s1 = new int[is];
for (i=0; i<s1.length;i++)
    s1[i] = s[i];

/* ora la fusione è stata completata */
return s1;
```

Lo schema dell'applicazione

```
public class ArrayFusioneSenzaDuplicati {

   public static int[] fusioneSenzaDuplicati(int[] a, int[] b) {
         …
   }

   public static String toString(int[] array) {
         …
   }

   public static void main(String[] args) {
         …
   }
}
```

Modifica del problema

- Come modificare l'algoritmo in modo che vengano anche accettati array vuoti?
- Basta effettuare un controllo sull'array vuoto e in caso positivo saltare la prima istruzione while?
  - No perché la seconda parte della fusione prevede che l'array contenga almeno un elemento
  - Bisogna controllare, anche nella seconda parte della fusione che is non sia 0.

Esercizio

- **Scrivere un metodo che, ricevendo come parametro un array di array A, verifica se A è bidimensionale**
  - **Ossia che tutti gli elementi di A siano array della stessa lunghezza**
- **Verifica universale**
  - **Schema: la variabile booleana è inizializzata a true**
    - **Si ipotizza che tutti gli array siano della stessa lunghezza**
    - **Ad esempio uguali alla lunghezza del primo array**
  - **L'istruzione while si interrompe non appena un array ha lunghezza diversa dal primo oppure quando è terminato l'esame di tutto l'array**

I test del problema

```
public static void testBidimensionale(){
    int[][] mat;

    mat = new int[][] {{1}};
    stampa(mat);
    System.out.println("Array bidimensionale? TRUE = "+bidimensionale(mat));

    mat = new int[][] {{1,0},{0,1}};
    stampa(mat);
    System.out.println("Array bidimensionale? TRUE = "+bidimensionale(mat));

    mat = new int[][] {{1,0,0},{0,1}};
    stampa(mat);
    System.out.println("Array bidimensionale? FALSE = "+bidimensionale(mat));

    mat = new int[][] {{2,0,0},{4,0},{0,0,5}};
    stampa(mat);
    System.out.println("Array bidimensionale? FALSE = "+bidimensionale(mat));

    mat = new int[][] {{3,0,0,0},{0,2,0,0},{0,0,5,0},{0,0,0,1}};
    stampa(mat);
    System.out.println("Array bidimensionale? TRUE = "+bidimensionale(mat));

    mat = new int[][] {{3,0,2,0},{0,2,1,0},{0,0,5,0},{}};
    stampa(mat);
    System.out.println("Array bidimensionale? FALSE = "+bidimensionale(mat));
}
```

Il metodo bidimensionale

```
public static boolean bidimensionale(int[][] mat){
   int i,j;
   int r;  //numero di righe dell'array
   int lunghezza; // lunghezza che dovrebbero avere tutti i
                  // sottoarray uguale alla lunghezza
                  // del primo sottoarray

   boolean bidimensionale;

   lunghezza = mat[0].length;
   r= mat.length;
   bidimensionale = true;

   for(i=1; i<r; i++)
       if(mat[i].length != lunghezza)
              bidimensionale = false;

   return bidimensionale;
}
```

Esempio d’esecuzione

```
matrice:
1
Array bidimensionale? TRUE = true
matrice:
1 0
0 1
Array bidimensionale? TRUE = true
matrice:
1 0 0
0 1
Array bidimensionale? FALSE = false
matrice:
2 0 0
4 0
0 0 5
Array bidimensionale? FALSE = false
matrice:
3 0 0 0
0 2 0 0
0 0 5 0
0 0 0 1
Array bidimensionale? TRUE = true
matrice:
3 0 2 0
0 2 1 0
0 0 5 0

Array bidimensionale? FALSE = false
Press any key to continue . . .
```

Esercizio

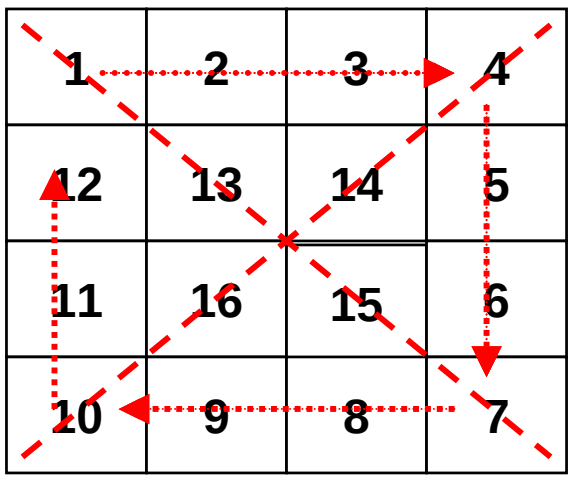
- **Scrivere un metodo che, dato un numero positivo N, crea e restituisce una matrice quadrata NxN di interi che memorizza i numeri da 1 a N^2 disposti "a spirale"**
  - **Esempio: le matrici a spirale di ordine 3 e 4**

1	2	3
8	9	4
7	6	5

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Analisi del problema

- Sia res la matrice risultato che vogliamo costruire
- Bisogna iniziare dalla componente mat[0][0] inserendo 1
- Si scorre verso destra inserendo numeri via via crescenti
- Quando si deve *cambiare direzione*?



- Quando si incontrano le diagonali principale o secondaria
- Oppure quando ci si trova “sotto” la diagonale principale

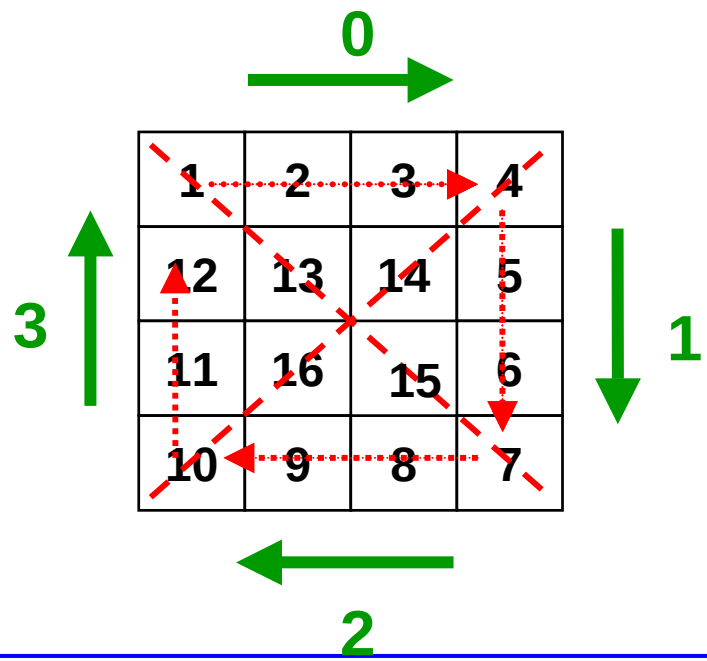
Il problema del cambio di direzione…

- **Convenzioni:**
  - da sinistra verso destra: 0 
  - dall'alto verso il basso: 1 
  - da destra verso sinistra: 2 
  - dal basso verso l'alto: 3 

- Se vado da sinistra verso destra (0) e incontro la diagonale secondaria, devo andare verso il basso (1)
- Se vado dall'alto verso il basso (1) e incontro la diagonale principale devo andare da destra verso sinistra (2)
- Se vado da destra verso sinistra (2) e incontro la diagonale secondaria devo andare dal basso verso l'alto (3)
- Se vado dal basso verso l'alto (3) e mi trovo una riga sotto la diagonale principale devo andare da sinistra verso destra (0)

…Il problema del cambio di direzione

- Quando ci si trova sulla diagonale secondaria i+j = n-1 la direzione diventa 1 o 3
- Quando ci si trova nella metà inferiore della diagonale principale i=j e j+j>=n la direzione diventa 2
- Quando ci si trova una riga sotto la metà superiore della diagonale principale i= j+1 e i+j<n la direzione diventa 0
- Negli altri casi la direzione non varia



Il metodo aggiornaDirezione

```
public static int aggiornaDirezione(int direzione,
                                    int i, int j, int n){
  /* se ci si trova sulla diagonale secondaria
     la direzione diventa 1 e 3 */
  if (i+j == n-1)
      direzione++;
  /* se ci si trova sulla meta' inferiore della
     diagonale principale
     la direzione diventa 2 */
  if (i==j && j+j>=n)
      direzione++;
  /* se ci si trova su una riga sotto la meta'
     superiore della diagonale principale
     la direzione diventa 0 */
  if (i==j+1 && i+j<n)
      direzione++;
  /*negli altri casi la direzione non varia */
  return direzione % 4;
}
```

Come aggiornare gli indici

- Dopo che è stato inserito un numero nella matrice il metodo aggiornaDirezione fornisce la posizione in cui deve essere memorizzato il numero successivo
- se la nuova (o vecchia) direzione è 1 o 3 la colonna non varia ma ci si sposta di riga
  - Se è 1 l'indice di riga viene incrementato
  - Se è 3 l'indice di riga viene decrementato
- Se la nuova (o vecchia) direzione è 0 o 2 la riga non varia ma ci si sposta di colonna
  - Se è 0 l'indice di colonna viene incrementato
  - Se è 2 l'indice di colonna viene decremantato

I metodi aggiornaRiga e aggiornaColonna

```
public static int aggiornaRiga(int i,int direzione){
   int res;
   if (direzione==1) res=i+1; // muove dall'alto verso
                              // il basso
   else res=i-1; // muove dal basso verso l'alto
   return res;
}

public static int aggiornaColonna(int j,int direzione){
   int res;
   if(direzione==0) res=j+1; // muove da sinistra verso
                             // destra
   else res=j-1;  // muove da destra verso sinistra
   return res;
}
```

Il metodo spirale…

- Dichiarazoni e inizializzazioni

```
public static int[][] spirale(int n){
    int[][] res;
    int i,j; // indici della matrice
    int num; // conterrà di volta in volta l'elemento
             // da inserire.
    int direzione;

    /**** Inizializzazione delle variabili ***/

    res = new int[n][n]; // creazione della matrice;
    i = 0;
    j = 0;
    res[i][j] = 1;
    direzione = 0; // inizializzata per default da
                   //sinistra verso destra
```

Il metodo spirale…

```
/* per ogni numero intero da 2 a n*n inserisco nella prossima
   componente, secondo le indicazioni della direzione */
for(num=2; num<=n*n; num++){
   // Calcolo la nuova direzione
   direzione=aggiornaDirezione(direzione,i,j,n);
   // se devo andare da dx verso sinistra o viceversa
   if((direzione==1)||(direzione==3))
       // Aggiorno l'indice delle righe dove necessario
       i=aggiornaRiga(i,direzione);
   else
       // Devo andare dall'alto verso basso o viceversa
       // Aggiorno l'indice di colonna
       j=aggiornaColonna(j,direzione);
   // Inserisco il nuovo elemento nella posizione
   // precedentemente calcolata.
   res[i][j]=num;
}
return res;
}
```

Esempi d’esecuzione

```
Spirale d'ordine 3
1 2 3
8 9 4
7 6 5

Spirale d'ordine 6
1 2 3 4 5 6
20 21 22 23 24 7
19 32 33 34 25 8
18 31 36 35 26 9
17 30 29 28 27 10
16 15 14 13 12 11

Spirale d'ordine 1
1
Press any key to continue . . .
```

Secondo approccio…

- Si puo' pensare di scomporre il problema in piu' step, riempiendo un "anello" della matrice alla volta, dal piu' esterno al piu' interno
- Si noti come i valori di ogni anello sono sequenziali

1	2	3	4	5	6
20	21	22	23	24	7
19	32	33	34	25	8
18	31	36	35	26	9
17	30	29	28	27	10
16	15	14	13	12	11

Il metodo spirale

- **Quanti anelli occorre fare?**
  - **N / 2 se N e' pari**
  - **(N / 2 + 0,5) se N e' dispari**

```
static int[][] spirale(int n) {
  int[][] m = new int[n][n];
  int value = 1;
  /* ceil → the smallest value that is not less
   * than the argument and is equal to a integer */
  int max = Math.ceil(m.length / 2d);
  for (int i = 0; i < max; i++)
    value = ring(m, i, value);

  return m;
}
```

Il metodo ring…

- **Il metodo ring riempie l'anello a distanza i dal primo elemento (offset) con valori a partire da value; il valore restituito e' l'ultimo valore inserito + 1**

ring(m, 1, 21) ⇒ 33

offset (righe o colonne) dal primo elemento = 1

1	2	3	4	5	6
20	21	22	23	24	7
19	32	33	34	25	8
18	31	36	35	26	9
17	30	29	28	27	10
16	15	14	13	12	11

Il metodo offset

- Prima di ring, realizziamo un metodo che data la distanza dal primo elemento della matrice **offset**, riga **i** e colonna **j** relativi al anello, imposti il valore della matrice a **value**

```
static void offset(int m[][], int offset,
                   int i, int j, int value) {
  m[offset + i][offset + j] = value;
}
```

...Il metodo ring...

```
static int ring(int m[][], int iring,
                            int startValue) {
  /*dimensione di un lato dell'anello*/
  int n = m.length - iring*2;
  /* prima riga */
  for (int k = 0; k < n; k++)
    offset(m,iring,0,k,startValue++);
  /* colonna destra */
  for (int k = 1; k < n; k++)
    offset(m,iring,k,n-1,startValue++);
  ...
```

Il metodo ring

```
    ...

    /* ulrima riga */
    for (int k = n - 2; k >= 0; k--)
      offset(m,iring,n-1,k,startValue++);
    /* colonna sinstra */
    for (int k = n - 2; k > 0; k--)
      offset(m,iring,k,0,startValue++);

    return startValue;
}
```