Corso di Programmazione

II Prova di accertamento del 19 Giugno 2024 / A

cognome e nome		

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nell'intestazione e su ciascun ulteriore foglio che intendi consegnare.

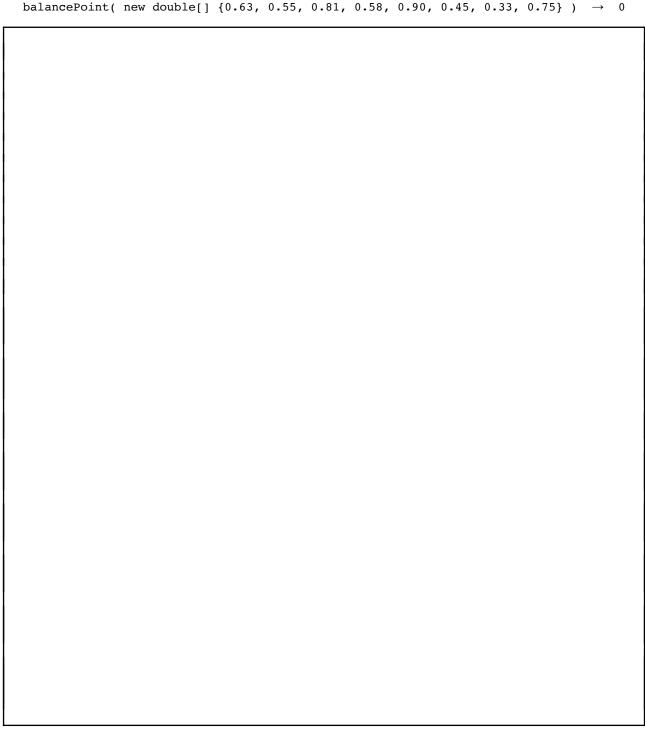
1. Programmazione in Java

Definisci in Java un metodo statico balancePoint che, dato un array s di double con $n \ge 2$ elementi tali che $s[i] \ge 0.2$ per ogni indice i dell'array, identifica un possibile "punto di bilanciamento" b tale che

$$|(s/0) + s/1| + ... + s/b-1| - (s/b) + s/b+1| + ... + s/n-1| | < 0.1$$

e restituisce b, se un tale punto esiste, oppure θ altrimenti. Esempi:

balancePoint(new double[] $\{0.80, 0.85, 0.70, 0.30, 0.36, 0.67, 0.25, 0.84\}$) \rightarrow 3 balancePoint(new double[] $\{0.63, 0.55, 0.81, 0.58, 0.90, 0.45, 0.33, 0.75\}$) \rightarrow 0



2. Programmazione dinamica bottom-up

Il seguente programma ricorsivo determina il coefficiente binomiale di indici $n \in k$ (interi):

$$bin(n,k) = \binom{n}{k} \quad \text{per } 0 \le k \le n$$
 public static long bin(int n, int k) { $\# 0 \le k \le n$ if ($(k == 0) \mid \mid (k == n)$) { return 1; } else { return bin(n-1,k-1) + bin(n-1,k); } }

Il programma impostato nel riquadro applica una tecnica bottom-up di programmazione dinamica per rendere più efficiente la computazione ricorsiva di bin. Ispirandosi ad esempi analoghi discussi a lezione, si potrebbe utilizzare una matrice bidimensionale per registrare i valori delle invocazioni ricorsive di bin(n,k) in corrispondenza agli indici $\langle n,k \rangle$. Tuttavia, si può anche osservare che i valori di una riga di indice n>0 dipendono esclusivamente dai valori della riga "precedente" di indice n-1. Ciò suggerisce di fare economia di memoria, conservando solo i valori di una riga in un array unidimensionale e sostituendo di volta i valori di una riga "nuova" a quelli della riga precedente (avendo cura di prestare attenzione al fatto che la maggior parte dei valori registrati vengono utilizzati due volte per passare alla riga successiva). Il metodo statico nextRow svolge precisamente questa funzione: l'esecuzione di nextRow(i,mem) ha l'effetto di sostituire nell'array mem i valori della riga di indice i a quelli della riga di indice i-1. Completa la definizione del metodo statico nextRow.

<pre>int[] mem = new int[n+1];</pre>	
mem[0] = 1;	// bin(0,0) = 1 è l'unico elemento utile della riga di indice 0
<pre>nextRow(i, mem);</pre>) { // prima dell'i-ma iterazione: $mem[j] = bin(i-1,j)$ per $j=0, 1,,$ // $bin(i-1,) \rightarrow bin(i,)$
<pre>} return mem[k]; }</pre>	<pre>// dopo l'i-ma iterazione: mem[j] = bin(i,j) per j=0, 1,, i // bin(n,k)</pre>
private static nex	tRow() {
private static	, , , , , , , , , , , , , , , , , , ,

3. Ricorsione e iterazione

Il seguente programma simula il processo risolutivo del rompicapo della *Torre di Hanoi* utilizzando un oggetto towers di tipo HanoiTowers che modella le varie configurazioni attraversate nel corso del gioco. In particolare, l'invocazione del metodo towers.move(s,d) consente di spostare un disco dalla cima della torre in corrispondenza all'asticella s alla cima della torre dell'asticella s, dove s, s, decomposite di spostare un disco dalla cima della torre in corrispondenza all'asticella s alla cima della torre dell'asticella s, dove s, s, decomposite towers registra anche quante volte un disco viene spostato in corrispondenza a un'asticella "vuota", in cui cioè non sono collocati altri dischi, e tale informazione può essere acquisita invocando il metodo towers.count(). Quindi, il metodo statico hanoiCount restituisce proprio questa informazione al termine del gioco.

```
public static int hanoiCount( int n ) {
   HanoiTowers towers = new HanoiTowers( n );
   hanoiRec( towers, 0, 1, 2, n );
   return towers.count();
}

private static void hanoiRec( HanoiTowers towers, int s, int d, int t, int n ) {
   if ( n == 1 ) {
      towers.move( s, d );
   } else if ( n > 1 ) {
      hanoiRec( towers, s, t, d, n-1 );
      hanoiRec( towers, s, d, 0, 1 );
      hanoiRec( towers, t, d, s, n-1 );
   }
}
```

Completa la definizione del metodo hanoilter impostato nel riquadro per trasformare il programma ricorsivo in un corrispondente programma iterativo che applica uno stack.

4. Classi in Java

Completa la definizione della classe Hanoi Towers impostata nel riquadro riportato sotto.

Oltre al costruttore, il protocollo di Hanoi Towers include i metodi empty, count, move, utilizzati dal programma oggetto dell'esercizio 3, nonché il metodo top Disk per conoscere il disco collocato in cima alla torre in corrispondenza all'asticella il cui indice è passato come argomento.

```
public class HanoiTowers {
  private int[] hgt;
  private int[][] twr;
  private int count;
  public HanoiTowers( int n ) {
                                      // altezze delle tre torri
    hgt = new int[3];
                                      // dischi collocati in ciascuna delle tre torri
    twr = new int[3][n];
                                      // altezze delle torri all'inizio: n, 0, 0
    hgt[0] = n;
    hgt[1] = 0;
    hgt[2] = 0;
    for ( int j=0; j<n; j=j+1 ) { // colloca n dischi di diametro n, n-1, ..., 2, 1
     twr[0][j] = n - j;
                                       // in corrispondenza all'asticella 0
    }
    count = 0;
                                        // spostamenti di dischi verso una asticella "vuota"
  public boolean empty( int i ) { // verifica se l'asticella i è "vuota" (senza dischi)
  public int count() {
    return count;
  public void move( int i, int j ) {  // spostamento di un disco dall'asticella i a j
    if ( _____
      count = count + 1;
    hgt[i] = hgt[i] - 1;
} // class HanoiTowers
```

Corso di Programmazione

II Prova di accertamento del 19 Giugno 2024 / B

cognome e nome		

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nell'intestazione e su ciascun ulteriore foglio che intendi consegnare.

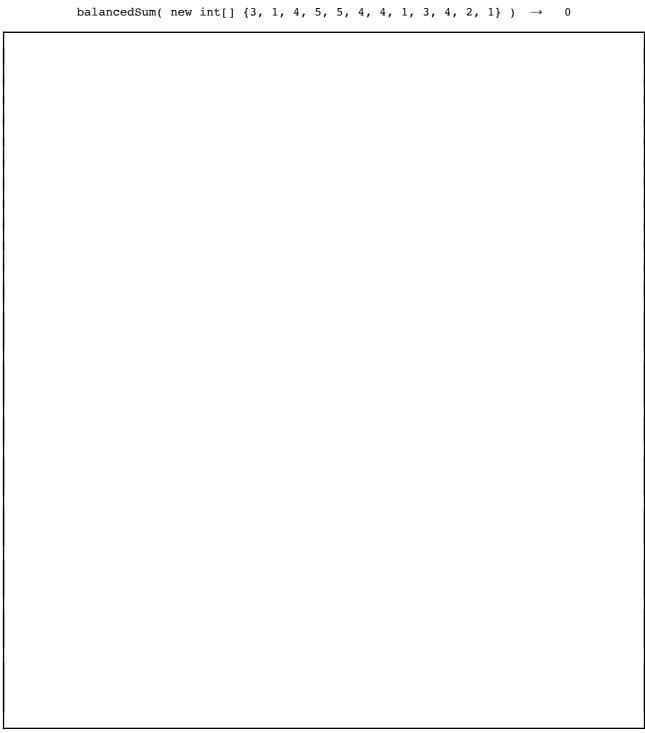
1. Programmazione in Java

Definisci in Java un metodo statico balancedSum che, dato un array s di int con $n \ge 2$ elementi strettamente positivi (s[i] > 0 per ogni indice i dell'array), identifica un possibile "punto di bilanciamento" b tale che

$$X = s[0] + s[1] + ... + s[b-1] = s[b] + s[b+1] + ... + s[n-1]$$

e restituisce X, se un tale punto esiste, oppure θ altrimenti. Esempi:

balancedSum(new int[] {2, 2, 5, 2, 4, 1, 2, 4, 2, 1, 2, 3})
$$\rightarrow$$
 15



2. Programmazione dinamica bottom-up

Il seguente programma ricorsivo determina il numero di Stirling di II specie di indici interi $n \in k$:

```
sti(n,k) = \begin{cases} n \\ k \end{cases} \quad \text{per } l \le k \le n \quad \text{("problema dei pasticcini")} public static long sti( int n, int k ) { // 1 \le k \le n if ( (k == 1) \mid \mid (k == n) ) { return 1; } else { return sti(n-1,k-1) + k*sti(n-1,k); } }
```

Il programma impostato nel riquadro applica una tecnica bottom-up di programmazione dinamica per rendere più efficiente la computazione ricorsiva di sti. Ispirandosi ad esempi analoghi discussi a lezione, si potrebbe utilizzare una matrice bidimensionale per registrare i valori delle invocazioni ricorsive di sti(n,k) in corrispondenza agli indici $\langle n,k\rangle$. Tuttavia, si può anche osservare che i valori di una riga di indice n>1 dipendono esclusivamente dai valori della riga "precedente" di indice n-1. Ciò suggerisce di fare economia di memoria, conservando solo i valori di una riga in un array unidimensionale e sostituendo di volta in volta i valori di una riga "nuova" a quelli della riga precedente (avendo cura di prestare attenzione al fatto che la maggior parte dei valori registrati vengono utilizzati due volte per passare alla riga successiva). Il metodo statico nextRow svolge precisamente questa funzione: l'esecuzione di nextRow(i,mem) ha l'effetto di sostituire nell'array mem i valori della riga di indice i a quelli della riga di indice i-1. Completa la definizione del metodo statico nextRow.

<pre>int[] mem = new int[n+1</pre>	1;
mem[1] = 1;	// sti(1,1) = 1 è l'unico elemento utile della riga di indice 1
<pre>for (int i=2; i<=n; i=i+ nextRow(i, mem); } return mem[k];</pre>	1) { // prima dell'iterazione per i: $mem[j] = sti(i-1,j)$ per $j=1, 2,,$ // $sti(i-1,) \rightarrow sti(i,)$ // $dopo l'iterazione relativa a i: mem[j] = sti(i,j) per j=1, 2,, // sti(n,k)$
}	" Su(r), ry
private static n	extRow() {

3. Ricorsione e iterazione

```
public static int hanoiCount( int n ) {
   HanoiTowers towers = new HanoiTowers( n );
   hanoiRec( n, 0, 1, 2, towers );
   return towers.count();
}

private static void hanoiRec( int n, int s, int d, int t, HanoiTowers towers ) {
   if ( n == 1 ) {
      towers.move( s, d );
   } else if ( n > 1 ) {
      hanoiRec( n-1, s, t, d, towers );
      hanoiRec( 1, s, d, 0, towers );
      hanoiRec( n-1, t, d, s, towers );
   }
}
```

Completa la definizione del metodo hanoiIter impostato nel riquadro per trasformare il programma ricorsivo in un corrispondente programma iterativo che applica uno stack.

4. Classi in Java

Completa la definizione della classe Hanoi Towers impostata nel riquadro riportato sotto.

Oltre al costruttore, il protocollo di Hanoi Towers include i metodi empty, count, move, utilizzati dal programma oggetto dell'esercizio 3, nonché il metodo top Disk per conoscere il disco collocato in cima alla torre in corrispondenza all'asticella il cui indice è passato come argomento.

```
public class HanoiTowers {
  private IntSList[] twr;
                                          // torri rappresentate da liste con il disco in cima all'inizio
  private int count;
  public HanoiTowers( int n ) {
                                        // dischi collocati in ciascuna delle tre torri
    twr = new IntSList[3];
    for ( int i=0; i<3; i=i+1 ) { // creazione delle tre torri inizialmente vuote
      twr[i] = IntSList.NULL_INTLIST;
      or ( int j=0; j<n; j=j+1 ) { // colloca n dischi di diametro n, n-1, ..., 2, 1

twr[0] = twr[0].cons(n-j); // in corrispondenza all'asticella 0
    for ( int j=0; j<n; j=j+1 ) {
                                            // spostamenti di dischi verso una asticella "vuota"
    count = 0;
  public boolean empty( int i ) {  // verifica se l'asticella i è "vuota" (senza dischi)
  public int topDisk( int i ) {  // diametro del disco in cima all'asticella i
  public int count() {
    return count;
  public void move( int i, int j ) {  // spostamento di un disco dall'asticella i a j
    if ( ________ ) {
      count = count + 1;
} // class HanoiTowers
```