

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  $n \geq 2$  elementi tali che  $s[i] \geq 0.2$  per ogni indice  $i$  dell'array, identifica un possibile "punto di bilanciamento"  $b$  tale che

$$|(s[0] + s[1] + \dots + s[b-1]) - (s[b] + s[b+1] + \dots + s[n-1])| < 0.1$$

e restituisce  $b$ , se un tale punto esiste, oppure  $0$  altrimenti. Esempi:

```
balancePoint( new double[] {0.80, 0.85, 0.70, 0.30, 0.36, 0.67, 0.25, 0.84} ) → 3
```

```
balancePoint( new double[] {0.63, 0.55, 0.81, 0.58, 0.90, 0.45, 0.33, 0.75} ) → 0
```

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

```
public static long bin( int n, int k ) { // 0 ≤ k ≤ n
    if ( (k == 0) || (k == n) ) {
        return 1;
    } else {
        return bin(n-1,k-1) + bin(n-1,k);
    }
}
```

[illegible]

### 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  $d$ , dove  $s, d \in \{0, 1, 2\}$ . L'oggetto `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 `hanoiIter` impostato nel riquadro per trasformare il programma ricorsivo in un corrispondente programma iterativo che applica uno stack.

```
public static int hanoiIter( int n ) {
    HanoiTowers towers = new HanoiTowers( n );
    Stack<int[]> stack = new Stack<int[]>();
    stack.push( new int[] {0,1,2,n} );
    do {
        int[] h = ..... ;
        if ( ..... == 1 ) {
            .....
        } else if ( ..... ) {
            stack.push( new int[] { ..... } );
            .....
            .....
            .....
        }
    } while ( ..... );
    return towers.count();
}
```

#### 4. Classi in Java

Completa la definizione della classe `HanoiTowers` impostata nel riquadro riportato sotto.

Oltre al costruttore, il protocollo di `HanoiTowers` include i metodi `empty`, `count`, `move`, utilizzati dal programma oggetto dell'esercizio 3, nonché il metodo `topDisk` 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 ) {

        hgt = new int[3];           // altezze delle tre torri
        twr = new int[3][n];       // dischi collocati in ciascuna delle tre torri

        hgt[0] = n;                 // altezze delle torri all'inizio: n, 0, 0
        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 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;
        }
        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  $n \geq 2$  elementi *strettamente* positivi ( $s[i] > 0$  per ogni indice  $i$  dell'array), identifica un possibile “punto di bilanciamento”  $b$  tale che

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

e restituisce  $X$ , se un tale punto esiste, oppure  $0$  altrimenti. Esempi:

```
balancedSum( new int[] {2, 2, 5, 2, 4, 1, 2, 4, 2, 1, 2, 3} ) → 15
```

```
balancedSum( new int[] {3, 1, 4, 5, 5, 4, 4, 1, 3, 4, 2, 1} ) → 0
```

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

```
public static long sti( int n, int k ) { // 1 ≤ k ≤ n
    if ( (k == 1) || (k == n) ) {
        return 1;
    } else {
        return sti(n-1,k-1) + k*sti(n-1,k);
    }
}
```

[illegible]

### 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  $d$ , dove  $s, d \in \{0, 1, 2\}$ . L'oggetto `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( 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.

```
public static int hanoiIter( int n ) {  
    HanoiTowers towers = new HanoiTowers( n );  
    Stack<int[]> stack = new Stack<int[]>();  
    stack.push( new int[] {n,0,1,2} );  
    do {  
        int[] h = ..... ;  
        if ( ..... == 1 ) {  
            .....  
        } else if ( ..... ) {  
            stack.push( new int[] { ..... } );  
            .....  
            .....  
            .....  
        }  
    } while ( ..... );  
    return towers.count();  
}
```

#### 4. Classi in Java

Completa la definizione della classe `HanoiTowers` impostata nel riquadro riportato sotto.

Oltre al costruttore, il protocollo di `HanoiTowers` include i metodi `empty`, `count`, `move`, utilizzati dal programma oggetto dell'esercizio 3, nonché il metodo `topDisk` 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 ) {

        twr = new IntSList[3];        // dischi collocati in ciascuna delle tre torri
        for ( int i=0; i<3; i=i+1 ) {  // creazione delle tre torri inizialmente vuote
            twr[i] = IntSList.NULL_INTLIST;
        }

        for ( 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
        }
        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 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
```