Corso di Programmazione

III Accertamento del 7 Luglio 2006 / A

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Memoization

Trasforma il seguente metodo statico in un programma corrispondente, formalizzato sempre nel linguaggio *Java*, che applichi la tecnica di *memoization*.

```
public static long count( int d, int k ) {  // 0 <= d < 4, k >= 0
  if ( k == 0 ) {
    if ( d == 0 ) { return 1; } else { return 0; }
  } else {
    return  2*count( d, k-1 ) + count( (d+3)%4, k-1 ) + count( (d+1)%4, k-1 );
}}
```

2. Programmazione dinamica

Trasforma il programma dell'esercizio precedente applicando la tecnica di *programmazione dinamica* e cercando di ridurre per quanto possibile la memoria utilizzata per registrare i valori via via calcolati.

3. Asserzioni e invarianti

Il seguente metodo in Java calcola i valori minimo e massimo di un array di interi. Completa il programma introducento opportune asserzioni, specificamente: precondizioni, postcondizioni e invarianti del comando iterativo; proponi inoltre una funzione di terminazione relativa al ciclo. A tua scelta, puoi formalizzare le asserzioni nel linguaggio *Jass* oppure utilizzando una notazione matematica.

```
public static int[] bounds( int[] v ) {

  /**  require  ..........................................................  **/

  int b = v.length – 1,  k = b, p = k, q = k;
  while ( k > 0 )

    /**  invariant  ..........................................................

    ..........................................................
    **/

    /**  variant  ..........................................................  **/ {

    k = k - 1;
    if      ( v[k] > v[p] ) {  p = k;  }
    else if ( v[k] < v[q] ) {  q = k;  }
  }
  int[] s = new int[2];
  s[0] = v[p];  s[1] = v[q];
  return s;

  /**  ensure  ..........................................................

  ..........................................................
  **/
}
```

4. Classi in Java

Una coda con priorità, *PriorityQueue*, è una collezione di elementi per la quale è definito il seguente protocollo: un costruttore che crea una struttura vuota; il metodo *size()* che consente di determinare il numero di elementi della collezione; il metodo *add(e,p)* che aggiunge alla collezione l'elemento *e*, assegnandogli priorità *p*; il metodo *highest()* che restituisce l'elemento di priorità più elevata, eventualmente scegliendo quello introdotto prima a parità di priorità, e che inoltre rimuove dalla collezione l'elemento restituito.
Formalizza in Java una classe *PriorityQueue* che realizzi le funzionalità descritte sopra, quando gli elementi che vengono inseriti e rimossi dalla struttura sono del tipo predefinito *String*. Per esempio, eseguendo la sequenza di istruzioni riportate qui sotto a sinistra, si vuole ottenere il risultato stampato a destra:

```
PriorityQueue q = new PriorityQueue();
q.add( "alpha", 2 );                                          size: 4
q.add( "beta", 5 );                                           beta
q.add( "gamma", 1 );                                          alpha
q.add( "delta", 2 );                                          delta
System.out.println( "size: " + q.size() );                   gamma
while ( q.size() > 0 ) {                                      size: 0
  System.out.println( q.highest() );
}
System.out.println( "size: " + q.size() );
```

Corso di Programmazione

III Accertamento del 7 Luglio 2006 / B

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Memoization

Trasforma il seguente metodo statico in un programma corrispondente, formalizzato sempre nel linguaggio *Java*, che applichi la tecnica di *memoization*.

```
public static long count( int k, int n ) {  // 0 <= k < 4, n > 0
  if ( n == 1 ) {
    if ( k == 3 ) { return 1; } else { return k; }
  } else {
    return  2*count( k, n-1 ) + count( (k+3)%4, n-1 ) + count( (k+1)%4, n-1 );
}}
```

2. Programmazione dinamica

Trasforma il programma dell'esercizio precedente applicando la tecnica di *programmazione dinamica* e cercando di ridurre per quanto possibile la memoria utilizzata per registrare i valori via via calcolati.

3. Asserzioni e invarianti

Il seguente metodo in Java calcola i valori minimo e massimo di un array di interi. Completa il programma introducento opportune asserzioni, specificamente: precondizioni, postcondizioni e invarianti del comando iterativo; proponi inoltre una funzione di terminazione relativa al ciclo. A tua scelta, puoi formalizzare le asserzioni nel linguaggio *Jass* oppure utilizzando una notazione matematica.

```
public static int[] bounds( int[] v ) {

  /**  require  ..........................................................  **/

  int n = v.length,  p = 0, q = 0, k = 1;
  while ( k < n )

    /**  invariant  ..........................................................

    ..........................................................
    **/

    /**  variant  ..........................................................  **/ {

    if      ( v[k] < v[p] ) {  p = k;  }
    else if ( v[k] > v[q] ) {  q = k;  }
    k = k + 1;
  }
  int[] s = new int[2];
  s[0] = v[p];  s[1] = v[q];
  return s;

  /**  ensure  ..........................................................

  ..........................................................
  **/
}
```

4. Classi in Java

Una coda con priorità, *PriorityQueue*, è una collezione di elementi per la quale è definito il seguente protocollo: un costruttore che crea una struttura vuota; il metodo empty*()* che consente di determinare se la collezione è vuota; il metodo *add(p,x)* che aggiunge alla collezione l'elemento *x*, associandogli un indice di priorità *p*; il metodo *lowest()* che restituisce l'elemento con indice di priorità più basso, eventualmente scegliendo quello introdotto prima a parità di indice, e che inoltre rimuove dalla collezione l'elemento restituito.
Formalizza in Java una classe *PriorityQueue* che realizzi le funzionalità descritte sopra, quando gli elementi che vengono inseriti e rimossi dalla struttura sono del tipo predefinito *String*. Per esempio, eseguendo la sequenza di istruzioni riportate qui sotto a sinistra, si vuole ottenere il risultato stampato a destra:

```
PriorityQueue q = new PriorityQueue();
q.add( 2, "alpha" );
q.add( 5, "beta" );
q.add( 1, "gamma" );
q.add( 2, "delta" );
System.out.println( "empty: " + q.empty() );
while ( !q.empty() ) {
  System.out.println( q.lowest() );
}
System.out.println( "empty: " + q.empty() );
```

```
empty: false
gamma
alpha
delta
beta
empty: true
```

Corso di Programmazione

III Accertamento del 7 Luglio 2006 / A

cognome e nome

1. Memoization

```
public static final long UNKNOWN = -1;

public static long count( int d, int k ) {  // 0 <= d < 4, k >= 0
  long[][] history = new long[4][k+1];
  for ( int i=0; i<4; i=i+1 ) {
    for ( int j=0; j<=k; j=j+1 ) {
      history[i][j] = UNKNOWN;
  }}
  return  count_mem( d, k,  history );
}

public static long count_mem( int d, int k,  long[][] history ) {
  if ( history[d][k] == UNKNOWN ) {
    if ( k == 0 ) {
      if ( d == 0 ) { history[d][k] = 1; } else { history[d][k] = 0; }
    } else {
      history[d][k] = 2*count_mem( d, k-1, history ) +
        count_mem( (d+3)%4, k-1, history ) + count_mem( (d+1)%4, k-1, history );
    }
  }
  return  history[d][k];
}
```

2. Programmazione dinamica

```
public static long count( int d, int k ) {  // 0 <= d < 4, k >= 0
  long[][] history = new long[4][2];
  history[0][0] = 1;
  for ( int i=1; i<4; i=i+1 ) { history[i][0] = 0; }
  for ( int j=1; j<=k; j=j+1 ) {
    for ( int i=0; i<4; i=i+1 ) {
      history[i][1] = 2*history[i][0] + history[(i+3)%4][0] + history[(i+1)%4][0];
    }
    for ( int i=0; i<4; i=i+1 ) { history[i][0] = history[i][1]; }
  }
  return  history[d][0];
}
```

3. Asserzioni e invarianti

```
public static int[] bounds( int[] v ) {

  /**  require   ( v.length > 0 );                                                **/

  int b = v.length – 1,  k = b, p = k, q = k;
  while ( k > 0 )

    /**  invariant     ( 0 <= p ) && ( p <= b ) && ( 0 <= q ) && ( q <= b );

                ( forall i : { k .. b } # ((v[q] <= v[i]) && (v[i] <= v[p])) );
    **/

    /**  variant   ( k )                                                          **/ {

    k = k - 1;
    if      ( v[k] > v[p] ) {  p = k;  }
    else if ( v[k] < v[q] ) {  q = k;  }
  }
  int[] s = new int[2];
  s[0] = v[p];  s[1] = v[q];
  return s;

  /**  ensure      ( exists i : { 0 .. v.length-1 } # (Result[0] == v[i]) );

              ( exists i : { 0 .. v.length-1 } # (Result[1] == v[i]) );

       ( forall i : {0..v.length-1} # ((Result[1]<=v[i]) && (v[i]<=Result[0])) );
  **/
}
```

4. Classi in Java

```
public class PriorityQueue {

  private static final int MAXSIZE = 256;

  private String[] element;
  private int[] priority;
  private int size;

  public PriorityQueue() {
    element = new String[MAXSIZE];
    priority = new int[MAXSIZE];
    size = 0;
  }

  public int size() {
    return size;
  }

  public void add( String e, int p ) {
    if ( size < MAXSIZE ) {
      int k = 0;
      while ( (k < size) && (priority[k] < p) ) { k = k + 1; }
      for ( int i=size; i>k; i=i-1 ) {
        element[i] = element[i-1];  priority[i] = priority[i-1];
      }
      element[k] = e;  priority[k] = p;
      size = size + 1;
    }
  }

  public String highest() {
    if ( size > 0 ) {
      size = size - 1;
      return element[size];
    } else {
      return null;
    }
  }
}
```

Corso di Programmazione

III Accertamento del 7 Luglio 2006 / B

cognome e nome

1. Memoization

```
public static final long UNKNOWN = -1;

public static long count( int k, int n ) {  // 0 <= k < 4, n > 0
  long[][] history = new long[4][n+1];
  for ( int i=0; i<4; i=i+1 ) {
    for ( int j=0; j<=n; j=j+1 ) {
      history[i][j] = UNKNOWN;
  }}
  return  count_mem( k, n,  history );
}

public static long count_mem( int k, int n,  long[][] history ) {
  if ( history[k][n] == UNKNOWN ) {
    if ( n == 1 ) {
      if ( k == 3 ) { history[k][n] = 1; } else { history[k][n] = k; }
    } else {
      history[k][n] = 2*count_mem( k, n-1, history ) +
        count_mem( (k+3)%4, n-1, history ) + count_mem( (k+1)%4, n-1, history );
    }
  }
  return  history[k][n];
}
```

2. Programmazione dinamica

```
public static long count( int k, int n ) {  // 0 <= k < 4, n > 0
  long[][] history = new long[4][2];
  for ( int i=0; i<3; i=i+1 ) { history[i][0] = i; }
  history[3][0] = 1;
  for ( int j=2; j<=n; j=j+1 ) {
    for ( int i=0; i<4; i=i+1 ) {
      history[i][1] = 2*history[i][0] + history[(i+3)%4][0] + history[(i+1)%4][0];
    }
    for ( int i=0; i<4; i=i+1 ) { history[i][0] = history[i][1]; }
  }
  return  history[k][0];
}
```

3. Asserzioni e invarianti

```
public static int[] bounds( int[] v ) {

  /**  require   ( v.length > 0 );                                                    **/

  int n = v.length,  p = 0, q = 0, k = 1;
  while ( k < n )

    /**  invariant     ( 0 <= p ) && ( p < n ) && ( 0 <= q ) && ( q < n );

         ( forall i : { 0 .. k-1 } # ((v[p] <= v[i]) && (v[i] <= v[q])) );
    **/

    /**  variant   ( n - k )                                                        **/ {

    if      ( v[k] < v[p] ) {  p = k;  }
    else if ( v[k] > v[q] ) {  q = k;  }
    k = k + 1;
  }
  int[] s = new int[2];
  s[0] = v[p];  s[1] = v[q];
  return s;

  /**  ensure     ( exists i : { 0 .. v.length-1 } # (Result[0] == v[i]) );

       ( exists i : { 0 .. v.length-1 } # (Result[1] == v[i]) );

     ( forall i : {0..v.length-1} # ((Result[0]<=v[i]) && (v[i]<=Result[1])) );
  **/
}
```

4. Classi in Java

```
public class PriorityQueue {

  private static final int MAXSIZE = 256;

  private String[] element;
  private int[] priority;
  private int size;

  public PriorityQueue() {
    element = new String[MAXSIZE];  priority = new int[MAXSIZE];
    size = 0;
  }

  public boolean empty() {
    return ( size == 0 );
  }

  public void add( int p, String e ) {
    if ( size < MAXSIZE ) {
      int k = 0;
      while ( (k < size) && (priority[k] > p) ) { k = k + 1; }
      for ( int i=size; i>k; i=i-1 ) {
        element[i] = element[i-1];  priority[i] = priority[i-1];
      }
      element[k] = e;  priority[k] = p;
      size = size + 1;
    }
  }

  public String lowest() {
    if ( size > 0 ) {
      size = size - 1;
      return element[size];
    } else {
      return null;
    }
  }
}
```