Corso di Programmazione

II Accertamento del 11 Giugno 2012

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Astrazione sui dati in Scheme

Il protocollo del dato astratto *"tavola rotonda"* è definito dalle procedure `round-table`, per costruire una tavola rotonda con *n* cavalieri; `last-knight-in?`, per verificare se in tavola è rimasto solo l'ultimo cavaliere; `knight-with-jug-in`, per conoscere l'etichetta del cavaliere con la brocca di sidro; `after-next-exit-from`, per effettuare un passo della conta (il cavaliere con la brocca serve il cavaliere alla sua sinistra, che esce, e passa la brocca al successivo). Rispetto alla versione discussa a lezione, ai cavalieri non vengono assegnate etichette numeriche, bensì stringhe di un solo carattere (lettera, cifra, altro simbolo). I caratteri assegnati a ciascuno degli *n* cavalieri si ricavano dalla stringa passata come argomento al "costruttore" `round-table`, dove il numero e l'ordine dei caratteri della stringa corrisponde al numero e all'ordine orario dei cavalieri attorno al tavolo, a partire da quello che inizialmente ha in mano la brocca di sidro. Di conseguenza, il nuovo protocollo può essere schematizzato così:

```
(new-round-table labels)      :  stringhe → configurazioni
(last-knight-in? table)       :  configurazioni → booleani
(knight-with-jug-in table)    :  configurazioni → etichette [stringhe di un carattere]
(after-next-exit-from table)  :  configurazioni → configurazioni
```

Definisci in Scheme le procedure che realizzano il nuovo protocollo in base alle indicazioni fornite sopra.

```
(define round-table
  (lambda (s)
    s
    ))

(define knight-with-jug-in
  (lambda (tab)
    (substring tab 0 1)
    ))

(define last-knight-in?
  (lambda (tab)
    (= (string-length tab) 1)
    ))

(define after-next-exit-from
  (lambda (tab)
    (string-append (substring tab 2) (substring tab 0 1))
    ))
```

2. Memoization

Considera il seguente metodo statico formalizzato nel linguaggio Java:

```
public static int f( int x, int y ) {  // x, y ≥ 0

  if ( (x == 0) || (y == 0) ) {
    return  0;
  } else if ( x == 1 ) {
    return  y;
  } else if ( y == 1 ) {
    return  x;
  } else {
    int u = (int) ( (x - 1) * Math.random() ) + 1;  // 0 < u < x
    int v = (int) ( (y - 1) * Math.random() ) + 1;  // 0 < v < y
    return  f( u, v ) + f( u, y-v ) + f( x-u, v ) + f( x-u, y-v );
  }
}
```

Trasforma il programma ricorsivo applicando opportunamente la tecnica di *memoization*.

(esercizio 2)

```
public static int fm( int x, int y ) {

  int[][] h = new int[ x+1 ][ y+1 ];

  for ( int i=0; i<=x; i=i+1 ) {
    for ( int j=0; j<=y; j=j+1 ) {
      h[i][j] = UNKNOWN;
  }}
  return  mem( x, y, h );
}

public static int mem( int x, int y, int[][] h ) {

  if ( h[x][y] == UNKNOWN ) {

    if ( (x == 0) || (y == 0) ) {
      h[x][y] = 0;
    } else if ( x == 1 ) {
      h[x][y] = y;
    } else if ( y == 1 ) {
      h[x][y] = x;
    } else {
      int u = (int) ( (x - 1) * Math.random() ) + 1;
      int v = (int) ( (y - 1) * Math.random() ) + 1;
      h[x][y] =  mem( u, v, h ) + mem( u, y-v, h ) + mem( x-u, v, h ) + mem( x-u, y-v, h );
    }
  }
  return  h[x][y];
}

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

3. Programmi in Java

Con riferimento al problema di *Steinhaus*, scrivi un metodo statico `steinhaus` in Java che, data una stringa *s* composta esclusivamente di caratteri `'+'` e `'-'`, restituisce una stringa *t* tale che ad ogni coppia di segni contigui uguali in *s* corrisponda un `'+'` in *t* e ad ogni coppia di segni contigui diversi in *s* corrisponda un `'-'` in *t*, nell'ordine. Per esempio:

`steinhaus( "++--+---++-+-+--" )` → `"+-+--++-+-----+"`

```
public static String steinhaus( String s ) {

  String t = "";

  for ( int i=1; i<s.length(); i=i+1 ) {

    t = t + ( (s.charAt(i) == s.charAt(i-1)) ? "+" : "-" );
  }
  return t;
}
```

4. Verifica formale della correttezza

Il seguente metodo statico in *Java* calcola la divisione intera e restituisce la coppia *quoziente* e *resto* attraverso un array di due elementi. Nel programma sono riportate le precondizioni e le postcondizioni. Introduci opportune asserzioni *invarianti* e dimostra formalmente la correttezza parziale (cioè assumendo la terminazione) del programma iterativo. Proponi, inoltre, una funzione di terminazione (senza dimostrazione).

```
public static int[] div( int m, int n ) {
  // Pre: m ≥ 0,  n > 0
  int q = 0, r = m;
  while ( r >= n ) {

    // Inv:    m = qn + r,  r ≥ 0

    // Term:   r

    q = q + 1;  r = r - n;
  }
  return  new int[] { q, r };
  // Post: m = qn + r,  0 ≤ r < n
}
```

(i) $m = 0 \cdot n + m$

$m \geq 0$

(ii) $m = (q+1) \cdot n + (r-n)$ $\Leftarrow$ $m = q \cdot n + r$

$r - n \geq 0$ $\Leftarrow$ $r \geq n$

(iii) $0 \leq r < n$ $\Leftarrow$ $r \geq 0,\ r < n$

5. Oggetti in Java

Considera il modello della scacchiera realizzato dalla classe `Board` per affrontare il rompicapo delle *n* regine. Per le istanze di `Board` è definito il protocollo richiamato qui sotto, a sinistra. Inoltre, come riferimento, è riportato anche il codice discusso a lezione al fine di determinare il numero di soluzioni in funzione della dimensione *n* della scacchiera.

```
Board( int n )  // costruttore

void addQueen( int i, int j )
void removeQueen( int i, int j )

int size()
int queensOn()
boolean underAttack( int i, int j )
String arrangement()
```

```
public static int numberOfArrangements( int n ) {

  return numberOfCompletions( new Board(n) );
}

public static int numberOfCompletions( Board board ) {

  int n = board.size();
  int q = board.queensOn();
  int all;
  if ( q == n ) {
    all = 1;
  } else {
    int i = q + 1;
    all = 0;
    for ( int j=1; j<=n; j=j+1 ) {
      if ( ! board.underAttack(i,j) ) {
        board.addQueen( i, j );
        all = all + numberOfCompletions( board );
        board.removeQueen( i, j );
  }}}
  return all;
}
```

Ora si vuole calcolare il numero di soluzioni del rompicapo, imponendo però l'ulteriore vincolo che una delle *n* regine debba essere collocata nella posizione individuata dalla coppia di coordinate <*i, j*>. Ad esempio, per *n = 4* e *i = j = 1* ci sono *zero* soluzioni; per *n = 4*, *i = 1 e j = 2* c'è *una* soluzione; per *n = 5* e *i = j = 3* ci sono *due* soluzioni.

Utilizzando esclusivamente il protocollo sopra specificato, scrivi un programma in Java che, dati la dimensione *n* della scacchiera e le coordinate *i*, *j*, risolva il problema proposto.

```
public static int numberOfArrangements( int n, int i, int j ) {

  Board b = new Board( n );
  b.addQueen( i, j );

  return numberOfCompletions( b, i );
}


public static int numberOfCompletions( Board board, int u ) {

  int n = board.size();
  int q = board.queensOn();
  int all;

  if ( q == n ) {

    all = 1;

  } else {

    int i = ( u + q - 1 ) % n + 1;
    all = 0;

    for ( int j=1; j<=n; j=j+1 ) {

      if ( ! board.underAttack(i,j) ) {

        board.addQueen( i, j );
        all = all + numberOfCompletions( board, u );
        board.removeQueen( i, j );
  }}}
  return all;
}
```