

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmi in Scheme

Facendo riferimento alla procedura f così definita:

```
(define f
  (lambda (x y)
    (cond ((= x 0) (+ y 1))
          ((= y 0) (f (- x 1) 1))
          (else (f (- x 1) (f x (- y 1)))))))
```

determina il risultato della valutazione di ciascuna delle espressioni riportate qui sotto:

(f 0 2) →	(f 0 6) →
(f 1 0) →	(f 1 5) →
(f 2 0) →	(f 2 2) →

2. Dati procedurali

Il protocollo del dato “*tavola rotonda*” è definito dalle procedure `new-round-table`, per costruire una tavola 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). Nella realizzazione impostata qui di seguito una configurazione di k cavalieri attorno alla tavola è rappresentata da una procedura, definita nell’intervallo $[0, k]$ e tale che se l’argomento è 0 restituisce k ; se è 1 restituisce l’etichetta numerica del cavaliere con la brocca; per $2, 3, \dots, k$ le etichette dei successivi cavalieri secondo l’ordine orario attorno alla tavola. (I valori restituiti per argomenti al di fuori dell’intervallo $[0, k]$ non hanno invece alcuna rilevanza.)

Completa le definizioni in Scheme delle procedure che realizzano il protocollo in base alle indicazioni fornite sopra.

```
(define new-round-table           ; val: configurazione
  (lambda (n)                    ; n: intero positivo
    (lambda (x) (if (= x 0) n x))
  ))

(define last-knight-in?          ; val: booleano
  (lambda (tab)                  ; tab: configurazione

    (= (tab 0) ..... )
  ))

(define knight-with-jug-in        ; val: etichetta numerica del cavaliere con la brocca
  (lambda (tab)                  ; tab: configurazione
    (tab 1)
  ))

(define after-next-exit-from      ; val: configurazione
  (lambda (tab)                  ; tab: configurazione
    (let ((n (- (tab 0) 1)))
      (lambda (x)

        (cond ((= x 0) ..... )

              ((= x n) ..... )

              (else ..... )
        ))
    )))
```

3. Programmazione Dinamica

Il metodo statico `lcsx` risolve il problema della *sottosequenza comune più lunga* (LCS) restituendo la coppia di stringhe “allineate”, cioè tali da rendere evidente la corrispondenza o meno dei simboli. A tal fine il carattere ‘_’ (underscore) è trattato come simbolo speciale per rappresentare localmente la mancanza di allineamento e perciò non compare nelle stringhe passate come argomento. Le coppie di stringhe sono rappresentate da array di due elementi.

```
public static String[] lcsx( String u, String v ) {
    if ( u.equals("") && v.equals("") ) {
        return new String[] { "", "" };
    } else if ( u.equals("") ) {
        String[] pair = lcsx( u, v.substring(1) );
        return new String[] { '_' + pair[0], v.charAt(0) + pair[1] };
    } else if ( v.equals("") ) {
        String[] pair = lcsx( u.substring(1), v );
        return new String[] { u.charAt(0) + pair[0], '_' + pair[1] };
    } else if ( u.charAt(0) == v.charAt(0) ) {
        String[] pair = lcsx( u.substring(1), v.substring(1) );
        return new String[] { u.charAt(0) + pair[0], v.charAt(0) + pair[1] };
    } else {
        String[] pair1 = lcsx( u.substring(1), v );
        String[] pair2 = lcsx( u, v.substring(1) );
        return better(
            new String[] { u.charAt(0) + pair1[0], '_' + pair1[1] },
            new String[] { '_' + pair2[0], v.charAt(0) + pair2[1] }
        );
    }
}

private static String[] better( String[] pair1, String[] pair2 ) {
    int n1 = 0, n2 = 0;
    for ( int i=0; i<pair1[0].length(); i=i+1 ) {
        if ( pair1[0].charAt(i) == pair1[1].charAt(i) ) { n1 = n1 + 1; }
    }
    for ( int i=0; i<pair2[0].length(); i=i+1 ) {
        if ( pair2[0].charAt(i) == pair2[1].charAt(i) ) { n2 = n2 + 1; }
    }
    if ( n1 < n2 ) {
        return pair2;
    } else if ( n1 > n2 ) {
        return pair1;
    } else if ( Math.random() < 0.5 ) { // scelta causale
        return pair2;
    } else {
        return pair1;
    }
}
```

La valutazione di `lcsx("arto", "atrio")` può restituire, ad esempio, la coppia di stringhe riportata a lato, dove si vede che la sottosequenza comune è costituita dai caratteri a, t, o, nell’ordine.

art__o
a_trio

Trasforma il programma ricorsivo in uno iterativo applicando la tecnica bottom-up di *programmazione dinamica*.

4. Correttezza dei programmi iterativi

Dato un intero positivo n , il seguente metodo statico calcola la soluzione $gf(n)$ del problema ispirato a un racconto di *Giuseppe Flavio*. Nel programma sono riportate preconditione, postcondizione, invariante e funzione di terminazione. Introduci opportune espressioni negli spazi denotati a tratto punteggiato (non è richiesta la dimostrazione).

```
public static int gFlavio( int n ) {    // Pre:     $n > 0$ 

    int q = 1;

    int j = ..... ;

    int p = ..... ;

    while ( q + j < n ) {                // Inv:     $q + j \leq n, \quad \exists k. q = 2^k, \quad 0 \leq j < q, \quad p = 2j + 1$ 
                                        // Term:     $n - q - j$ 

        j = j + 1;

        p = p + ..... ;

        if ( p > q + j ) {
            q = q + j;

            j = ..... ;

            p = ..... ;
        }
    }
    return p;                            // Post:     $p = gf(n)$ 
}
```

5. Oggetti in Java

Considera il metodo `flattenTree` per rappresentare l'*albero di Huffman* con una stringa di caratteri, il cui codice è riportato qui sotto nella versione basata su uno *stack*. A differenza del programma discusso a lezione, lo *stack* è un'istanza della classe `NodeStack` il cui protocollo è costituito dal costruttore `NodeStack()` e dai metodi `boolean empty()`, `void push(Node n)` e `Node pop()`.

```
private static String flattenTree( Node root ) {

    String flat = "";
    NodeStack stack = new NodeStack();    // creazione di uno stack vuoto
    stack.push( root );                  // la radice dell'albero di Huffman è introdotta nello stack
    while ( !stack.empty() ) {           // verifica se lo stack non è vuoto
        Node n = stack.pop();             // estrae e rimuove il nodo in cima allo stack
        if ( n == null ) {                // parentesi chiusa
            flat = flat + ")";
        } else if ( n.isLeaf() ) {        // foglia: codifica del carattere
            char c = n.character();
            if ( (c == '\\') || (c == '(') || (c == ')') ) {
                flat = flat + "\\\" + c;    // caratteri speciali: \, (, )
            } else {
                flat = flat + c;           // altri caratteri
            }
        } else {
            flat = flat + "(";             // (
            stack.push( null );            // )
            stack.push( n.right() );        // la codifica del sottoalbero destro
            stack.push( n.left() );         // è preceduta dalla codifica del sottoalbero sinistro
        }
    }
    return flat;
}
```

Definisci in Java una classe `NodeStack` compatibile con quanto specificato sopra.