Corso di Programmazione

Esame del 27 Luglio 2017

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmazione in Scheme

Date due stringhe `u` e `v`, la procedura `lcs` calcola una soluzione del problema della *sottosequenza comune più lunga* (LCS). Il risultato è rappresentato dalla lista delle coppie di posizioni relative ai caratteri comuni a `u` e `v` che fanno parte della sottosequenza più lunga. Esempi:

```
(lcs "pino" "pino")    →   ((0 0) (1 1) (2 2) (3 3))
(lcs "pelo" "peso")    →   ((0 0) (1 1) (3 3))
(lcs "ala" "palato")   →   ((0 1) (1 2) (2 3))
(lcs "arto" "atrio")   →   ((0 0) (2 1) (3 4))
```

In particolare, nell'ultimo esempio `(2 1)` corrisponde alle posizioni di `'t'` rispettivamente in `"arto"` e `"atrio"`.
Completa il programma riportato nel riquadro introducendo opportune espressioni negli appositi spazi.

```
(define lcs       ; valore: lista di coppie
  (lambda (u v)   ; u, v: stringhe

    (lcs-rec u v  ..........................  ..........................  )
    ))

(define lcs-rec
  (lambda (u v i j)
    (cond ((or (= i (string-length u)) (= j (string-length v)))

               ..........................................................  )

          ((char=?  ..........................  ..........................  )

           (cons  ..........................  (lcs-rec u v (+ i 1) (+ j 1))))
          (else
           (better  ..........................................................

                    ..........................................................  ))

          )))
(define better
  (lambda (x y)
    (if (< (length x) (length y))  y  x)
    ))
```

2. Procedure con argomenti e valori procedurali

Nel seguente programma la procedura `livs` risolve una variante del problema della *sottosequenza crescente più lunga*:

```
(define livs
  (lambda (f n)
    (livs-rec f 1 0 n)
    ))

(define livs-rec
  (lambda (f i k n)
    (if (> i n)
        (lambda (x) false)
        (let ((g (livs-rec f (+ i 1) k n)))
          (if (<= (f i) k)
              g
              (let ((h (livs-rec f (+ i 1) (f i) n)))
                (better g (lambda (x) (if (= x i) true (h x))) i n)
                )))
        )))
```

```
(define better
  (lambda (g h i n)
    (if (< (count g i n) (count h i n))  h  g)
    ))

(define count
  (lambda (f i n)
    (cond ((> i n) 0)
          ((f i) (+ 1 (count f (+ i 1) n)))
          (else (count f (+ i 1) n))
          )))
```

Con riferimento al programma riportato sopra, determina il risultato della valutazione di ciascuna delle espressioni:

`(count (lambda (x) (even? x)) 1 5)` → _____

`(count (lambda (x) (not (even? x))) 3 9)` → _____

`(map (livs (lambda (x) 5) 1) '(1))` → _____

`(map (livs (lambda (x) x) 1) '(1))` → _____

`(map (livs (lambda (x) x) 3) '(1 2 3))` → _____

`(map (livs (lambda (x) (- 3 x)) 2) '(1 2))` → _____

`(map (livs (lambda (x) (* (- (* 4 x) 9) (- (* 4 x) 9))) 3) '(1 2 3))`

→ _____

3. Ricorsione e iterazione

Dato un *albero di Huffman*, il metodo statico `shortestCodeLength` determina la lunghezza del più corto fra i codici di Huffman associati ai caratteri che compaiono in un documento di testo. In particolare, la visita dell'albero, finalizzata alla determinazione di tale lunghezza, è realizzata attraverso uno schema iterativo basato su uno stack specializzato il cui protocollo è specificato nell'esercizio **5** (vedi). Completa la definizione del metodo `shortestCodeLength`.

```
public static int shortestCodeLength( Node root ) {
  int sc = 8;
  NodeIntStack stack = new NodeIntStack();
  stack.push( root, 0 );
  do {
    Node n = stack.topNode();

    int  d = _____ ;

    _____ ;

    if ( n.isLeaf() ) {
      sc = Math.min( sc, d );
    } else if ( d+1 < sc ) {

      _____

      _____
    }
  }  while ( _____ );

  return sc;
}
```

Corso di Programmazione

Esame del 27 Luglio 2017

cognome e nome

4. Memoization

Il seguente programma in Scheme calcola la lunghezza della *sottosequenza comune crescente più lunga* di due stringhe, dove l'ordine (crescente) dei caratteri è quello alfabetico:

```
(define llcis                ; valore: intero
  (lambda (u v)              ; u, v: stringhe
    (llcis-rec u v #\space)
    ))

(define llcis-rec
  (lambda (u v c)
    (cond ((or (string=? u "") (string=? v ""))
           0)
          ((char<=? (string-ref u 0) c)
           (llcis-rec (substring u 1) v c))
          ((char<=? (string-ref v 0) c)
           (llcis-rec u (substring v 1) c))
          (else
           (let ((k (max (llcis-rec (substring u 1) v c) (llcis-rec u (substring v 1) c))))
             (if (char=? (string-ref u 0) (string-ref v 0))
                 (max k (+ 1 (llcis-rec (substring u 1) (substring v 1) (string-ref u 0))))
                 k)))
          )))
```

Applica la tecnica *top-down* di *memoization* per realizzare in Java una versione più efficiente del programma.

5. Classi in Java

La classe `NodeIntStack` consente di istanziare *stack* in cui possono essere inserite coppie <*nodo, intero*>. Più specificamente, il protocollo è definito da un costruttore e da cinque metodi così caratterizzati:

```
new NodeIntStack()      // costruisce uno stack vuoto
s.empty()               // verifica se lo stack s è vuoto
s.push(n,k)             // aggiunge in cima allo stack s la coppia costituita dal nodo n e dall'intero k
s.pop()                 // rimuove la coppia in cima allo stack s
s.topNode()             // restituisce il nodo della coppia in cima allo stack s, senza modificare lo stack
s.topInt()              // restituisce l'intero della coppia in cima allo stack s, senza modificare lo stack
```

Completa la definizione della classe `NodeIntStack` introducendo opportune variabili d'istanza (rappresentazione interna nascosta) e realizzando il costruttore e i metodi coerentemente con le scelte implementative fatte.

```
public class NodeIntStack {









  public NodeIntStack() {



  }

  public boolean empty() {


  }

  public void push( Node n, int d ) {



  }

  public void pop() {



  }

  public Node topNode() {


  }

  public int topInt() {


  }

}  // class NodeIntStack
```