Corso di Programmazione

Esame del 5 Settembre 2017

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmazione in Scheme

Date le stringhe `u`, `v`, la procedura `lcs` calcola una soluzione del problema della *sottosequenza comune più lunga*. Il risultato è rappresentato da una lista di terne, ciascuna delle quali contiene le posizioni in `u` e in `v` di un carattere comune che fa parte della sottosequenza più lunga, numerate a partire da 1, e la stringa costituita dal solo carattere comune. Esempi:

```
(lcs "pino" "pino")    →   ((1 1 "p") (2 2 "i") (3 3 "n") (4 4 "o"))
(lcs "pelo" "peso")    →   ((1 1 "p") (2 2 "e") (4 4 "o"))
(lcs "ala" "palato")   →   ((1 2 "a") (2 3 "l") (3 4 "a"))
(lcs "arto" "atrio")   →   ((1 1 "a") (3 2 "t") (4 5 "o"))
```

In particolare, nell'ultimo esempio `(3 2 "t")` riporta le 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 terne
  (lambda (u v)   ; u, v: stringhe

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

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

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

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

           (cons  ..........................................................................

                 (lcs-rec (+ i 1) (substring u 1) (+ j 1) (substring v 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) false) 1 5)                                    →  ..........

(count (lambda (x) (or (even? x) (= (remainder x 3) 0))) 1 10)    →  ..........

((better (lambda (x) (= (remainder x 3) 0)) (lambda (x) (even? x)) 3 12) 6)

                                                  →  ..........

((better (lambda (x) (= (remainder x 3) 0)) (lambda (x) (even? x)) 3 12) 9)

                                                  →  ..........

(map (livs (lambda (x) (* 2 x)) 1) '(1))     →  ..........

(map (livs (lambda (x) (- 2 x)) 1) '(1))     →  ..........

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

3. Ricorsione e iterazione

Dati un carattere *c* e un *albero di Huffman*, il metodo statico `retrieveCode` restituisce il codice di Huffman associato a *c*. In particolare, la visita dell'albero, finalizzata alla ricerca del carattere e alla costruzione del codice corrispondente (stringa binaria), 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 `retrieveCode`.

```
public static String retrieveCode( char c, Node root ) {

  NodeStringStack stack = ..........;
  stack.push( root, "" );
  do {
    Node n = stack.topNode();

    String code = ..........;

    ..........;

    if ( n.isLeaf() ) {
      if ( n.character() == c ) {  return code;  }
    } else {

      ..........

      ..........

    }
  }  while ( ..........);

  return null;  // c non compare nell'albero di Huffman
}
```

Corso di Programmazione

Esame del 5 Settembre 2017

cognome e nome

4. Programmazione dinamica

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 *bottom-up* di *programmazione dinamica* per realizzarne una versione più efficiente in Java.

5. Classi in Java

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

```
new NodeStringStack()   // costruisce uno stack vuoto
s.empty()               // verifica se lo stack s è vuoto
s.push(n,t)             // aggiunge in cima allo stack s la coppia costituita dal nodo n e dalla stringa t
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.topString()           // restituisce la stringa della coppia in cima allo stack s, senza modificare lo stack
```

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

```
public class NodeStringStack {









  public NodeStringStack() {



  }

  public boolean empty() {


  }

  public void push( Node n, String t ) {



  }

  public void pop() {



  }

  public Node topNode() {


  }

  public String topString() {


  }

}  // class NodeStringStack
```