Corso di Programmazione

Esame del 11 Luglio 2017

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmazione in Scheme

Scrivi un programma in Scheme basato sulla procedura `sorted-char-list` che, data una stringa, restituisce la lista dei caratteri che vi compaiono, ordinata in ordine alfabetico e senza ripetizioni. Esempi:

```
(sorted-char-list "")        →   ()
(sorted-char-list "abc")     →   (#\a #\b #\c)
(sorted-char-list "cba")     →   (#\a #\b #\c)
(sorted-char-list "list of chars that occur in this text")
             →   (#\space #\a #\c #\e #\f #\h #\i #\l #\n #\o #\r #\s #\t #\u #\x)
```

(Per il confronto alfabetico di caratteri puoi utilizzare le procedure predefinite `char=?`, `char<?`, `char<=?`, ecc.)

2. Programmi in Scheme

```
(define q
  (lambda (n i j)                                ; n > 0
    (q-rec n i j (expt 2 (- n 1)) "A" "B" "C")  ; expt: elevamento a potenza
    ))
```

```
(define q-rec
  (lambda (n i j k x y z)
    (if (= n 1)
        (p i j k x y)
        (let ((u (q-rec (- n 1) i j (/ k 2) x z y))
              (v (q-rec (- n 1) (- i k) (- j k) (/ k 2) z y x))
              )
          (append u (p i j k x y) v)
          ))))

(define p
  (lambda (i j k x y)
    (if (or (< k i) (> k j))
        null
        (list (string-append x y))
        )))
```

Facendo riferimento al programma realizzato dalle procedure `q`, `q-rec` e `p`, determina il risultato della valutazione di ciascuna delle seguenti espressioni:

`(q 1 1 1)` → _____ `(q 3 1 3)` → _____

`(q 2 1 3)` → _____ `(q 3 5 7)` → _____

`(q 2 3 3)` → _____ `(q 4 8 8)` → _____

3. Procedure con argomenti procedurali

Date due funzioni *f, g* con argomenti e valori interi, la procedura `lcvs` calcola una sottosequenza comune più lunga dei valori di *f* e di *g* nell'intervallo *[a, b]*. In altri termini si vuole conoscere una soluzione del problema *LCS* relativamente alle sequenze di interi *f(a), f(a+1), …, f(b)* e *g(a), g(a+1), …, g(b)*. Per esempio:

```
(lcvs (lambda (x) x) (lambda (x) x) 11 20)      → (11 12 13 14 15 16 17 18 19 20)
(lcvs (lambda (x) x) (lambda (x) (- 6 x)) 1 5)  → (5)
(lcvs (lambda (x) x) (lambda (x) (* 2 x)) 1 10) → (2 4 6 8 10)
(lcvs (lambda (x) x) (lambda (x) (* x x)) 1 10) → (1 4 9)
```

Completa il programma riportato nel riquadro introducendo opportune espressioni negli appositi spazi.

```
(define lcvs              ; valore: lista di interi
  (lambda (f g a b)       ; f, g: procedure [intero --> intero];  a, b: interi

    (lcvs-rec f  _____________________________  g  _____________________________  b)
    ))

(define lcvs-rec
  (lambda (f i g j n)
    (cond ((or (> i n) (> j n))

           _____________________________________________________________________ )

          ((=  _____________________________  _____________________________ )

           (cons  _____________________________  (lcvs-rec f (+ i 1) g (+ j 1) n)))
          (else

           (better  _____________________________________________________________

                    (lcvs-rec f i  _____________________  _____________________  n)))
          )))

(define better
  (lambda (u v)
    (if (< (length u) (length v))  v  u)
    ))
```

Corso di Programmazione

Esame del 11 Luglio 2017

cognome e nome

4. Programmazione dinamica

Il seguente programma consente di calcolare la lunghezza della sottosequenza crescente più lunga (*LLIS*) di una sequenza di interi rappresentataa da un'istanza della classe `Vector<Integer>` :

```
public static int llis( Vector<Integer> s ) {

  return llisRec( s, 0, 0 );
}

public static int llisRec( Vector<Integer> s, int t, int k ) {

  if ( k == s.size() ) {
    return 0;
  } else if ( s.get(k) <= t ) {
    return llisRec( s, t, k+1 );
  } else {
    return Math.max( 1+llisRec(s,s.get(k),k+1), llisRec(s,t,k+1) );
  }
}
```

Applica la tecnica *bottom-up* di *programmazione dinamica* per realizzare una versione più efficiente del programma. A tal fine, assumi che le sequenze passate come argomento al metodo statico `llis` soddisfino sempre questa proprietà: una sequenza *s* di lunghezza *k* è costituita da una permutazione degli interi *1, 2, ..., k*. (Tale proprietà limita l'intervallo di valori che possono essere assunti dal parametro `t` del metodo ricorsivo.)

5. Verifica formale della correttezza

Il programma iterativo per `josephus`, che deriva dalla procedura ricorsiva (`ufo`) discussa a lezione, risolve il problema ispirato da un racconto di Giuseppe Flavio quando il numero di commensali (≥ 1) è rappresentato in forma binaria da una stringa x. Tale stringa deve quindi contenere solo le cifre `0`/`1` e deve iniziare con `1` (cifra più significativa). Nel programma sono riportati precondizione, che si riferisce allo stato immediatamente dopo l'introduzione della costante `n`, postcondizione e invariante. Per comodità si è utilizzata la seguente notazione: x_j denota il valore della cifra in posizione j nella stringa x, cioè $x_j = 0$ se `x.charAt(j)` è `'0'` e $x_j = 1$ se `x.charAt(j)` è `'1'`; $\forall\, j$ significa per tutte le posizioni *ammissibili* nella stringa x; $\sum_{j \in [1,i-1]}$ è la sommatoria estesa a tutti i valori interi di j da 1 a $i-1$.

Dimostra formalmente la correttezza *parziale* (cioè tralasciando la terminazione) del programma.

```
public static int josephus( String x ) {
  final int n = x.length();          // Pre: n > 0,  ∀ j . x_j ∈ [0, 1],  x_0 = 1
  int k = 1;
  int i = 1;
  while ( i < n ) {                  // Inv: 0 < i ≤ n,  k = 2 ( Σ_{j∈[1,i-1]} x_{i-j} 2^{j-1} ) + 1
    if ( x.charAt(i) == '0' ) {      // se x_i = 0 …
      k = 2 * k - 1;
    } else {
      k = 2 * k + 1;
    }
    i = i + 1;
  }
  return k;                          // Post: k = 2 ( Σ_{j∈[1,n-1]} x_{n-j} 2^{j-1} ) + 1
}
```