Corso di Programmazione

Esame del 5 Settembre 2016

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Astrazione sui dati e dati procedurali

Si vuole introdurre in Scheme un dato astratto *"crivello di Eratostene"* per realizzare un modello del crivello utilizzato nell'algoritmo di Eratostene per determinare tutti i numeri primi che non superano un dato limite *n*. Più specificamente, il protocollo del dato astratto è definito dalle seguenti procedure:

```
(new-sieve)                    ; crea un crivello con potezialmente tutti gli interi ≥ 2
(is-in-sieve p sieve)          ; verifica se p è (ancora) presente nel crivello sieve
(remove-multiples-of p sieve)  ; crivello con il contenuto di sieve tranne i multipli propri di p
(sieve-list sieve n)           ; restituisce la lista dei numeri ≤ n contenuti nel crivello sieve
```

Applicando questa struttura, l'algoritmo per generare i numeri primi non superiori a *n* può essere formulato così:

```
(define primes-list
  (lambda (n)  (eratosthenes 2 n (new-sieve)) ))

(define eratosthenes
  (lambda (p n s)
    (cond ((> p n) (sieve-list s n))
          ((is-in-sieve p s) (eratosthenes (+ p 1) n (remove-multiples-of p s)))
          (else (eratosthenes (+ p 1) n s))
          )))
```

Completa la realizzazione del protocollo in modo tale che siano rispettate le specifiche illustrate sopra.

```
(define new-sieve            ; val: crivello di Eratostene (nuovo)
  (lambda ()                 ; "costruttore" senza argomenti
    (lambda (x) (> x 1))     ; rappresentazione interna procedurale
    ))

(define is-in-sieve          ; val: booleano
  (lambda (p sieve)          ; p: intero,  sieve: crivello di Eratostene
    (sieve p)
    ))

(define remove-multiples-of  ; val: crivello di Eratostene
  (lambda (p sieve)          ; p: intero > 1,  sieve: crivello di Eratostene
    (lambda (x)
      (if ( ..... )
          (= x p)
          .....
          ))
    ))

(define sieve-list           ; val: lista di interi
  (lambda (sieve n)          ; sieve: crivello di Eratostene,  n: intero
    (scan-sieve 2 n sieve)
    ))

(define scan-sieve
  (lambda (x y s)
    (cond ((> x y)  ..... )
          .....
          .....
          .....
          )))
```

2. Memoization

Applica la tecnica *top-down* di *memoization* alla seguente procedura, al fine di trasformarla in un programma più efficiente.

```
public static long s( int n, int k ) {   // 1 ≤ k ≤ n
  if ( k == n ) {
    return 1;
  } else {
    long x = (n - 1) * s( n-1, k );
    if ( k == 1 ) {
      return x;
    } else {
      return x + s( n-1, k-1 );
  }}
}
```

3. Correttezza dei programmi ricorsivi

La procedura `lcm-proc` calcola il minimo comune multiplo (*mcm*) di due interi positivi *m,n*. In corrispondenza all'intestazione della procedura ricorsiva di coda `lcm-tr` sono riportate le assunzioni relative agli argomenti e il valore restituito (*mod* denota l'operazione di resto della divisione intera). Completa la definizione del programma in modo che tali specifiche siano soddisfatte.

```
(define lcm-proc      ; val: mcm(m,n)
  (lambda (m n)       ; m, n: interi positivi
    (lcm-tr m n m n)
    ))

(define lcm-tr        ; val: mcm(m,n)
  (lambda (x y m n)   ; x, y, m, n: interi  tali che  0 < x, y ≤ mcm(m,n);  x mod m = y mod n = 0
    (let (
          (s (+ x n -1)) (t (+ y m -1))
          )
      (cond ((< x y) (lcm-tr (- t ..... ) y m n))

            ((> x y) (lcm-tr x (- s ..... ) m n))

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

4. Programmazione in Java

Dato un *array* di numeri (`double`) con almeno due elementi, il metodo statico `closestPair` ne restituisce una coppia la cui differenza in valore assoluto è minima. La coppia è rappresentata da un array ordinato di due elementi. Esempio:

```
closestPair( new double[] {0.3, 0.1, 0.6, 0.8, 0.5, 1.1} ) → {0.5, 0.6}
```

Definisci in Java il metodo statico `closestPair`.

5. Ricorsione e iterazione

Dato un *albero di Huffman*, costruito sulla base dei caratteri che compaiono in un documento di testo, il metodo statico `shortestCodeLength` determina la lunghezza del più corto fra i codici di Huffman associati ai caratteri e il numero di caratteri che hanno codici della lunghezza minima individuata. A tal fine, il valore restituito è un array di due interi (coppia di interi) che rappresentano queste due informazioni nell'ordine in cui sono state appena introdotte, cioè prima la lunghezza e poi la numerosità.

Nel corpo del metodo la visita dell'albero, finalizzata alla determinazione di tali informazioni, è realizzata attraverso uno schema iterativo.

Completa la definizione del metodo `shortestCodeLength` riportata nel riquadro.

```
public static int[] shortestCodeLength( Node root ) {

  int sc = ..... ;

  int k  = ..... ;

  Stack<Node> stack = new Stack<Node>();
  Stack<Integer> depth = new Stack<Integer>();

  stack.push( root );
  depth.push( 0 );

  do {

    Node n = ..... ;

    int  d = ..... ;

    if ( n.isLeaf() ) {
      if ( d < sc ) {
        sc = d;

        k = ..... ;
      } else {

        k = ..... ;
      }

    } else if ( d < ..... ) {
      stack.push( n.right() );

      .....

      .....

      .....

    }
  }  while ( ..... );

  return  new int[] { sc, k };
}
```