Corso di Programmazione

II Accertamento del 11 Giugno 2018

cognome e nome

Selezione degli esercizi proposti.

2. Procedure con argomenti e valori procedurali in Scheme

Il *Crivello di Eratostene* è un algoritmo per generare una tabella di numeri primi, da *2* fino a una certa soglia *n*. Nella sua forma più semplice, la tecnica può essere descritta come segue: inizialmente il "crivello" contiene tutti gli interi da *2* a *n*; quindi si considera ciascun intero *k*, procedendo in ordine crescente da *2* a *n*, e se *k* è ancora contenuto nel crivello allora si rimuovono dal crivello i suoi multipli *propri* (cioè diversi da *k* stesso), altrimenti si passa direttamente all'intero successivo. Al termine del processo, raggiunto il valore *n*, il crivello conterrà solo numeri primi.

Il programma impostato nel riquadro è inteso a realizzare l'algoritmo illustrato. In particolare, il terzo argomento della procedura ricorsiva `sieve` rappresenta il crivello attraverso un predicato `p?` (procedura a valori booleani) che, dato un intero *k* compreso fra *2* ed *n*, restituisce `true` se e solo se *k* è contenuto nel crivello associato a `p?`. Quindi, il predicato restituito da `eratosthenes` consente di verificare se un intero nell'intervallo *[2, n]* è primo. Per esempio, la procedura `primes-list` definita qui sotto applica `eratosthenes` per calcolare la lista dei numeri primi compresi fra *2* ed *n*.

```
(define primes-list
  (lambda (n)
    (primes-rec 2 n (eratosthenes n))
    ))
```

```
(define primes-rec
  (lambda (k n prime?)
    (cond ((> k n)
           null)
          ((prime? k)
           (cons k (primes-rec (+ k 1) n prime?)))
          (else
           (primes-rec (+ k 1) n prime?))
          )))
```

Completa le procedure `eratosthenes` e `sieve` introducendo opportune espressioni negli appositi spazi.

```
(define eratosthenes
  (lambda (n)

    (sieve 2 n ........................................ )
    ))

(define sieve
  (lambda (k n p?)
    (cond ((> k n)
           p?)
          ((p? k)
           (sieve (+ k 1) n
                  (lambda (x)

                    (if (and (> x k) .......................... )
                        false

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

           (sieve (+ k 1) n ................................ ))
          )))
```

3. Memoization

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

```
public static int q( int[] s ) {  // s.length > 0
  int n = s.length;
  int[] t = new int[ n ];  t[0] = s[0];
  for ( int k=1; k<n; k=k+1 ) {
    int i=k-1;
    while ( (i >= 0) && (t[i] > s[k]) ) {
      t[i+1] = t[i];  i = i - 1;
    }
    t[i+1] = s[k];
  }
  return qRec( s, t, n, 0, 0 );
}

private static int qRec( int[] s, int[] t, int n, int i, int j ) {
  if ( (i == n) || (j == n) ) {
    return 0;
  } else if ( s[i] == t[j] ) {
    return 1 + qRec( s, t, n, i+1, j+1 );
  } else {
    return Math.max( qRec(s,t,n,i+1,j), qRec(s,t,n,i,j+1) );
  }
}
```

4. Classi in Java

Un'istanza della classe `ProximityStructure` consente di modellare una collezione di misure, rappresentate da valori di tipo `double`, manipolabile attraverso il seguente protocollo:

```
new ProximityStructure()     // costruisce una collezione vuota di misure
s.size()                     // restituisce il numero di misure contenute nella collezione
s.add( x )                   // aggiunge la misura x alla collezione s
s.removeClosestTo( x )       // rimuove da s e restituisce la misura più prossima a x
                             // (la cui distanza da x è più piccola) in s
```

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

```
public class ProximityStructure {



  public ProximityStructure() {


  }

  public int size() {

  }

  public void add( double x ) {



  }

  public double removeClosestTo( double x ) {



  }
}  // class ProximityStructure
```

5. Astrazione funzionale (ai fini della valutazione, questo quesito ha un peso molto minore degli altri)

Riesci ad intuire quale problema risolve o quale funzione calcola la procedura `q` definita nell'esercizio **3**?
Spiega in poche parole le tue intuizioni e le ragioni (il perché) alla base di esse.