Corso di Programmazione

Esame del 15 Luglio 2014

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Correttezza dei programmi ricorsivi

Dati due interi non negativi *i*, *j*, il seguente programma in Scheme risolve il problema dei percorsi di *Manhattan*. In corrispondenza alla procedura ricorsiva sono riportate le assunzioni sui valori degli argomenti e un'espressione che rappresenta il valore restituito. In coerenza con le specifiche indicate, completa il programma introducendo opportune espressioni negli spazi denotati a tratto punteggiato (non è richiesta la dimostrazione formale di correttezza).

```
(define manhattan
  (lambda (i j)

    (manh-rec  ..........................................  1 i j)
    ))          ; valore restituito:  (i+j)! / (i!·j!)

(define manh-rec
  (lambda (p k i j)  ; p, k, i, j: interi;  si assume  p > 0,  1 ≤ k ≤ j+1,  i ≥ 0,  j ≥ 0

    (if  ..........................................

        (manh-rec (+ p  .....................................  ) (+ k 1) i j)

         .....................................

        )))          ; valore restituito:  p·(i+j)!·(k−1)! / (j!·(i+k−1)!)
```

2. Procedure con argomenti e valori procedurali

Completa il seguente programma dove la procedura `sorter` riceve un argomento procedurale, specificamente un predicato che determina la relazione di precedenza fra coppie di elementi di un dominio *D*, e restituisce la procedura per ordinare liste di elementi di *D* sulla base della precedenza stabilita dal predicato. Per esempio, `(sorter string<=?)` ordina liste di stringhe secondo l'ordine alfabetico; `(sorter <=)` ordina liste di numeri.

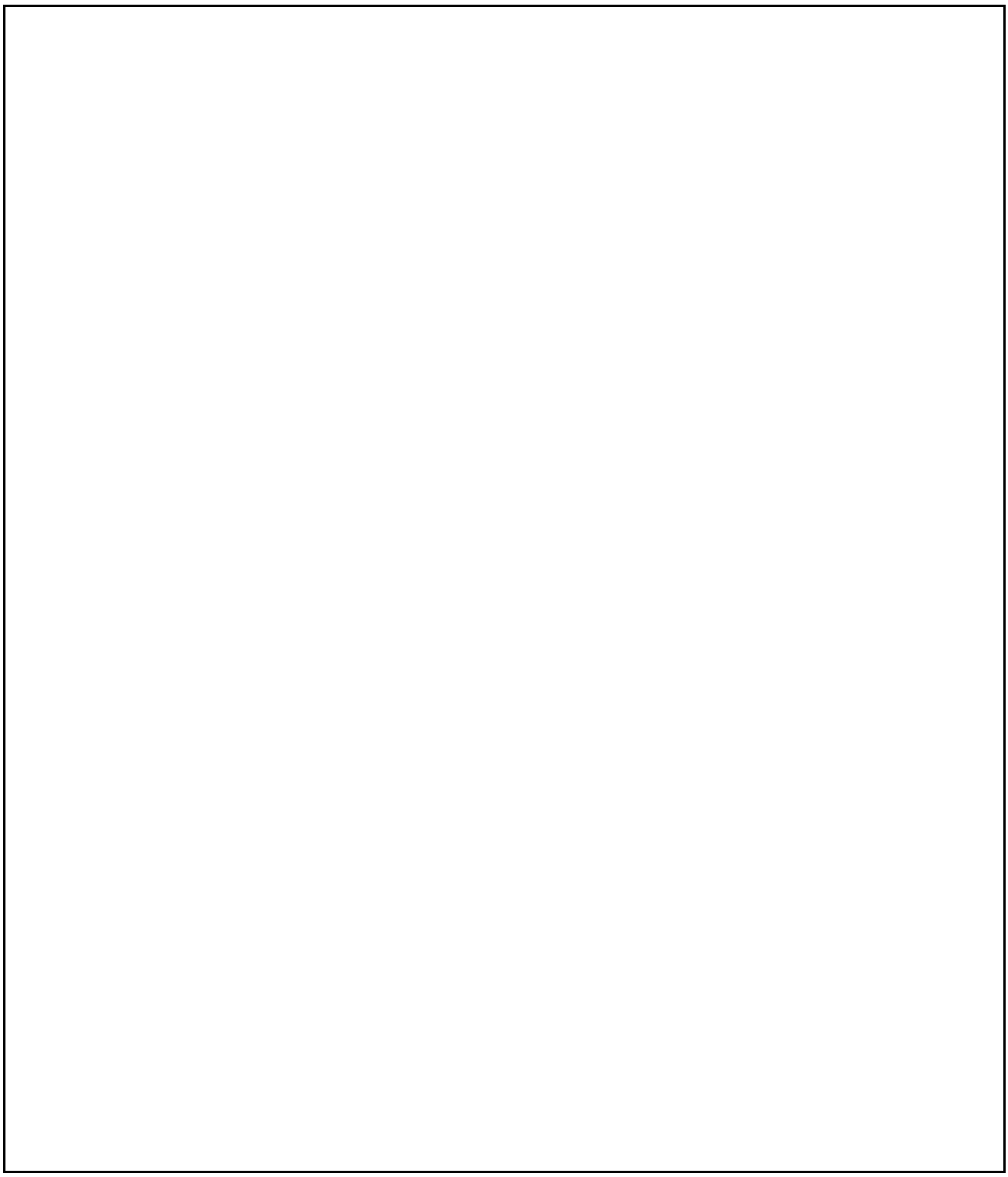
```
(define sorter           ; valore: procedura di ordinamento di liste di D
  (lambda (before)       ; before: procedura  D x D → boolean  (predicato)

    ( ..........................................
      (if (or (null? sequence) (null? (cdr sequence)))

          .....................................

          (sorted-ins before (car sequence)  ..................................... )
          ))))

(define sorted-ins
  (lambda (before item sequence)
    (cond ((null? sequence)

           ..................................... )
          ((before item (car sequence))

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

           (cons (car sequence)  ..................................... ))
          )))
```

3. Memoization

La procedura `llcs3` determina la *lunghezza della sottosequenza comune più lunga* (LLCS) di tre stringhe:

```
(define llcs3
  (lambda (t u v)
    (cond ((or (string=? t "") (string=? u "") (string=? v ""))
           0)
          ((char=? (string-ref t 0) (string-ref u 0) (string-ref v 0))
           (+ 1 (llcs3 (substring t 1) (substring u 1) (substring v 1))))
          (else
           (max (llcs3 (substring t 1) u v)
                (llcs3 t (substring u 1) v)
                (llcs3 t u (substring v 1))))
          )))
```

Trasforma la procedura `llcs3` in un programma *Java* che applica opportunamente la tecnica *top-down* di *memoization*.



4. Programmazione in Java

Come è noto dall'algebra lineare, una matrice quadrata Q si dice *simmetrica* se coincide con la propria trasposta, ovvero se $Q_{ij} = Q_{ji}$ per tutte le coppie di indici della matrice. Scrivi un metodo statico in Java per verificare se l'argomento è una matrice simmetrica (assumendo che tale argomento rappresenti una matrice *quadrata*).

Per quanto possibile, struttura il codice in modo tale da ridurre al minimo il numero di confronti effettuati dal programma nei casi in cui la matrice è effettivamente simmetrica.

5. Oggetti in Java

Considera il metodo `huffmanTree`, il cui codice è riportato sotto a destra, per costruire l'*albero di Huffman* direttamente sulla base del documento da comprimere, di nome `src`. In altre parole, questa versione di `huffmanTree` svolge allo stesso tempo i ruoli dei metodi `charHistogram` e `huffmanTree` nel programma discusso a lezione. Essenzialmente l'obiettivo viene perseguito arricchendo e riorganizzando la coda con priorità, ora istanza della classe `HuffmanQueue`, il cui protocollo viene ridefinito come specificato qui di seguito a sinistra.

```
// Coda con priorità "arricchita" di nodi: protocollo

HuffmanQueue()    // costruttore: struttura vuota

int size()        // numero di nodi nella struttura

void addChar(char c)  // se non c'è un nodo associato a c
          // ne crea uno e lo aggiunge alla struttura,
          // altrimenti incrementa il peso del nodo preesistente

void addNode(Node n)  // aggiunge il nodo n alla struttura

Node peekAndRemoveMin()   // rimuove dalla struttura
          // e restituisce il nodo di peso minimo
```

```
public static Node huffmanTree( String src ) {
  InputTextFile in = new InputTextFile( src );
  HuffmanQueue queue = new HuffmanQueue();
  while ( in.textAvailable() ) {
    char c = in.readChar();
    queue.addChar( c );
  }
  in.close();
  while ( queue.size() > 1 ) {
    Node l = queue.peekAndRemoveMin();
    Node r = queue.peekAndRemoveMin();
    Node n = new Node( l, r );
    queue.addNode( n );
  }
  return queue.peekAndRemoveMin();
}
```

Definisci in Java una classe `HuffmanQueue` compatibile con le indicazioni fornite sopra. A tal fine supponi che il protocollo della classe `Node` renda disponibile anche un metodo `incrWeight()` per incrementare di uno il peso di un nodo (anche lo stato interno sarà modificato di conseguenza, ma ciò non ti è richisto dal presente esercizio).

(esercizio 5)