Corso di Programmazione

Esame del 28 Luglio 2014

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Correttezza dei programmi ricorsivi

Dati due interi non negativi n, k, il seguente programma in Scheme calcola il relativo coefficiente binomiale. In corrispondenza alla procedura ricorsiva di coda 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 bin
  (lambda (n k)  ; 0 ≤ k ≤ n

    (if  ..........................................................
         1
         (bin-tr n k  ..........................................  1)
         )))                                          ; valore restituito:  n! / (k!·(n−k)!)

(define bin-tr
  (lambda (n k q i)  ; n, k, q, i: interi;  si assume inoltre  1 ≤ i ≤ k ≤ n,  q > 0

    (if  ..........................................................
         (/ (* q n) k)
         (bin-tr n k (/ (* q  ..................  )  ..................  ) (+ i 1))
         )))                                          ; valore restituito:  n·q·(n−i)!·(i−1)! / (k!·(n−k)!)
```

2. Procedure con argomenti e valori procedurali

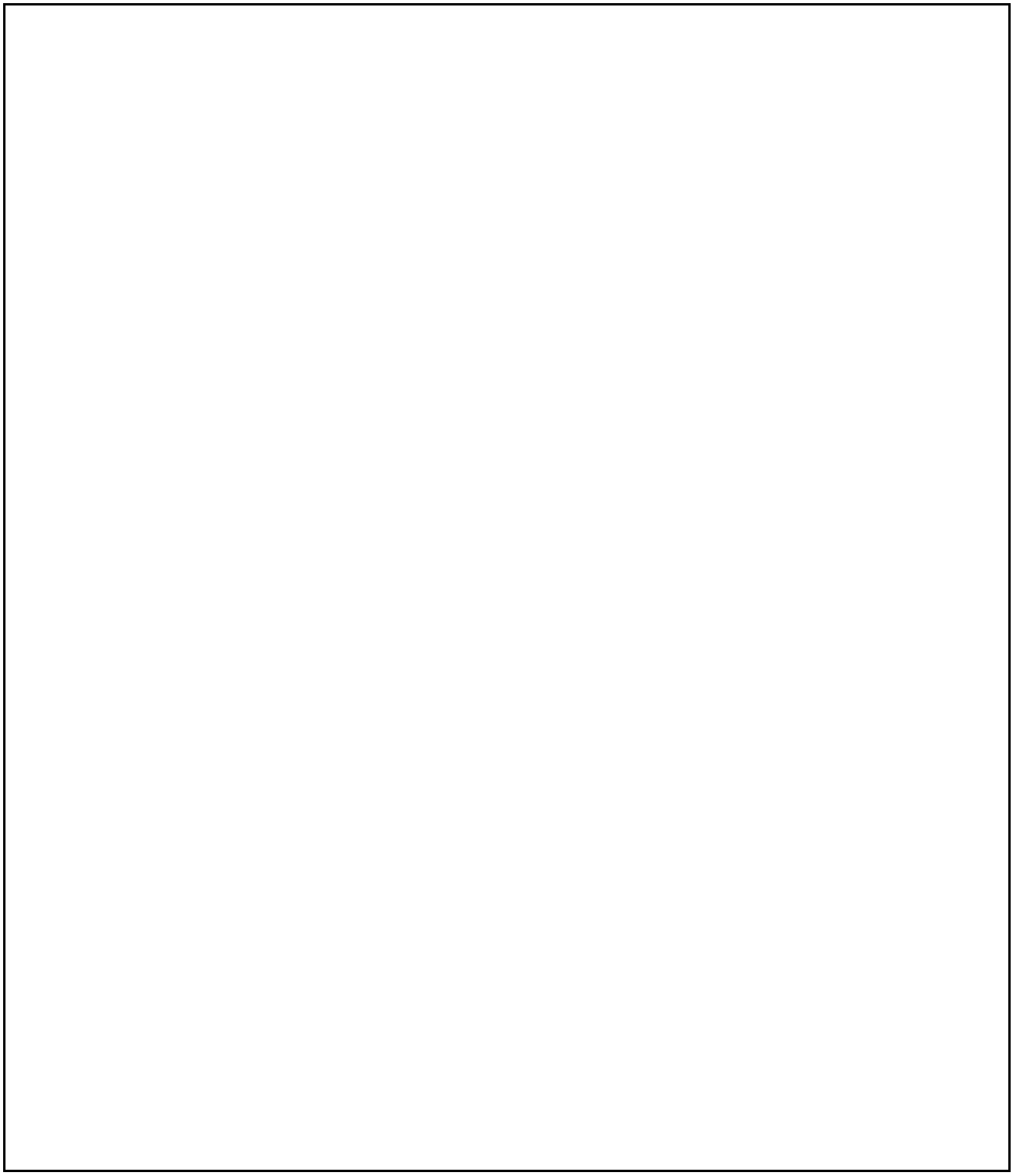
Completa la definizione della procedura `minmax` che, dato un predicato (procedura a valori booleani) per determinare la relazione di precedenza fra elementi del dominio D, restituisce la procedura che calcola la coppia di estremi (lista dei due elementi con precedenza minima e massima) di liste di elementi di D in base alla precedenza stabilita dal predicato. Per esempio, la procedura `(minmax string<=?)` si applica a liste di stringhe per determinare la prima e l'ultima stringa secondo l'ordine alfabetico; `(minmax <=)` consente di determinare i valori estremi di liste numeriche.

```
(define minmax           ; valore: procedura da liste non vuote di D a coppie di D
  (lambda (before)       ; before: procedura  D x D —> boolean

    (  ..................................................................

      (if (null?  ..................................................................  )
          (cons (car sequence) sequence)

          (let ((pair (  ..................................................  (cdr sequence))))
            (cond ((before (car sequence) (car pair))
                   (list (car sequence) (cadr pair)))

                  (  ..................................................................
                   (list (car pair) (car sequence)))
                  (else

                    ..................................................................  )
                  ))
          ))))
```

3. Programmazione Dinamica

La procedura `manhattan3d` determina il *numero di percorsi di Manhattan* attraverso un reticolo tridimensionale che portano da un vertice del parallelepipedo al piano orizzontale che contiene il vertice diagonalmente opposto (*k* è la distanza dal vertice di partenza a tale piano orizzontale). Trasforma la procedura in un programma non ricorsivo in *Java* che applica opportunamente la tecnica *bottom-up* di *programmazione dinamica*.

```
(define manhattan3d
  (lambda (i j k)
    (cond ((= k 0) 1)
          ((and (= i 0) (= j 0)) 1)
          ((= i 0)
           (+ (manhattan3d 0 (- j 1) k) (manhattan3d 0 j (- k 1))))
          ((= j 0)
           (+ (manhattan3d (- i 1) 0 k) (manhattan3d i 0 (- k 1))))
          (else
           (+ (manhattan3d (- i 1) j k) (manhattan3d i (- j 1) k) (manhattan3d i j (- k 1))))
    )))
```



4. Programmazione in Scheme

Definisci in Scheme una procedura `monotone-subseqs` che, data una lista *L* di interi, restituisce la lista delle sottoliste di elementi successivi non decrescenti di *L*, rispettandone l'ordine. Per esempio:

```
(monotone-subseqs '(1 8 3 5 7 4 4 2 1 9 9 12 20 15 15 14 13 12 18 5 7 8 1 22 5 18 19))
  → '((1 8) (3 5 7) (4 4) (2) (1 9 9 12 20) (15 15) (14) (13) (12 18) (5 7 8) (1 22) (5 18 19))
```

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 join()       // accoppia i due nodi di peso minimo e
                  // li sostituisce nella struttura con la radice
                  // del nuovo albero che ne risulta

Node peekMin()    // 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 ) {
    queue.join();
  }
  return queue.peekMin();
}
```

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 richiesto dal presente esercizio).

(esercizio 5)