Corso di Programmazione

Esame del 5 Settembre 2014

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Programmi in Scheme

Facendo riferimento alla procedura `f` così definita:

```
(define f
  (lambda (n)
    (f-rec 1 0 1 n)
    ))

(define f-rec
  (lambda (x y z n)
    (cond ((= (+ x y) n)
           z)
          ((< (+ y 1) x)
           (f-rec x (+ y 1) (+ z 2) n))
          (else
           (f-rec (* 2 x) 0 1 n))
          )))
```

determina il risultato della valutazione di ciascuna delle seguenti espressioni:

`(f 1)` → ..... `(f 8)` → .....

`(f 3)` → ..... `(f 25)` → .....

`(f 5)` → ..... `(f 63)` → .....

2. Ricorsione e argomenti procedurali

Completa la definizione della procedura `manhattan` che, dati due interi non negativi *i* e *j*, restituisce la lista dei percorsi di Manhattan all'interno di una griglia composta da *i* x *j* riquadri. Ciascuno dei percorsi è rappresentato da una stringa binaria, dove il simbolo `0` rappresenta uno spostamento unitario verticale e il simbolo `1` ne rappresenta uno orizzontale. Per esempio, i 6 percorsi distinti attraverso una griglia *2* x 2 risultano dalla valutazione dell'espressione:

```
(manhattan 2 2)     →     ("0011" "0101" "0110" "1001" "1010" "1100")
```

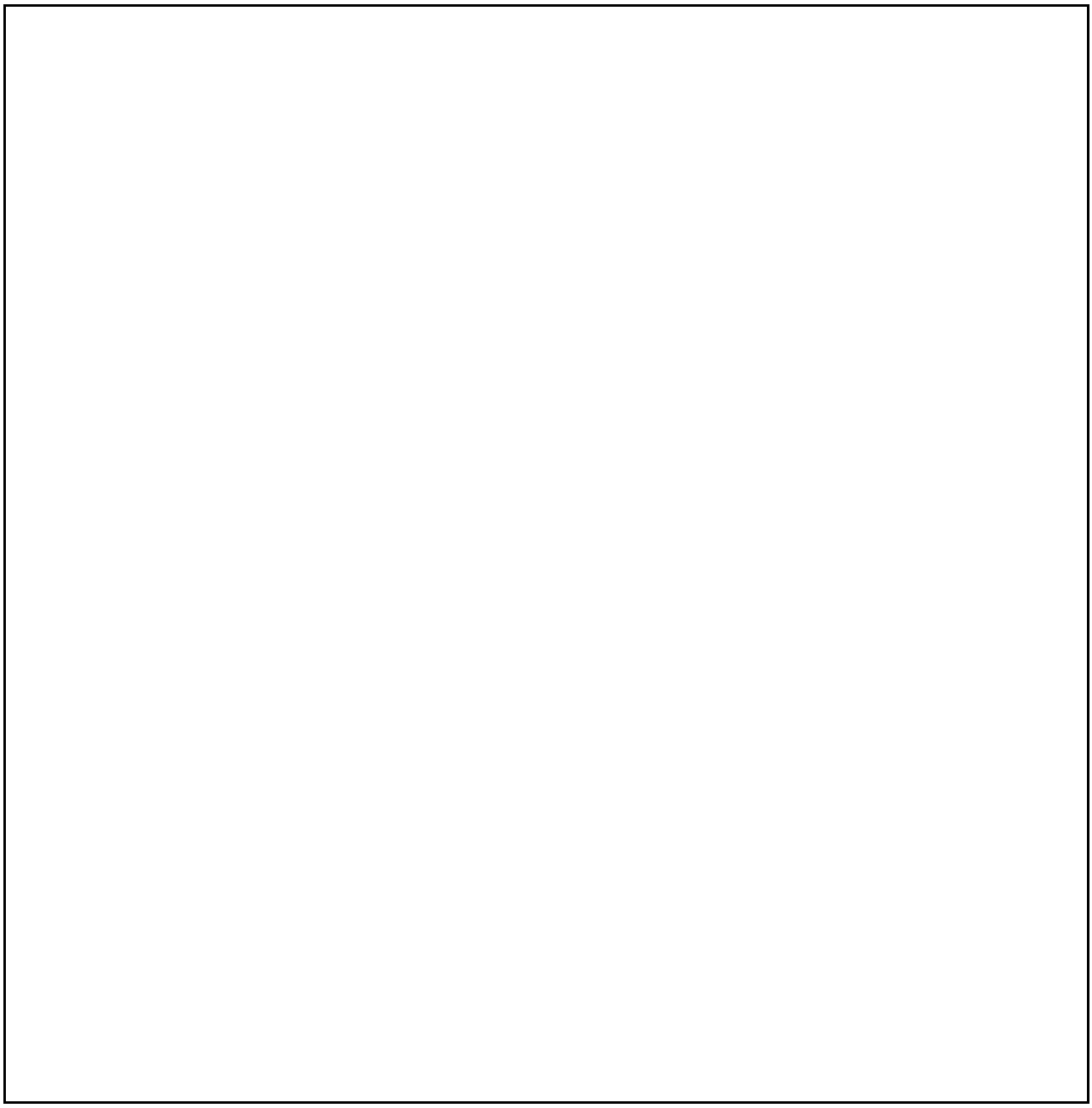
```
(define manhattan  ; valore: lista di stringhe di 0/1
  (lambda (i j)    ; i, j: linteri non negativi
    (cond ((and (= i 0) (= j 0))
           (list ""))
          ((= j 0)
           (list (string-append "0" (car ..... ))))
          ((= i 0)
             ..... )
          (else
           (append
            (map ..... (manhattan (- i 1) j))

            (map ..... )
            ))
          )))
```

3. Memoization

Il seguente metodo statico in Java produce un risultato equivalente a quello della procedura definita nell'esercizio precedente, dove gli oggetti di tipo `Vector<String>` svolgono il ruolo delle liste di stringhe in Scheme.

```
public static Vector<String> manhattan( int i, int j ) {
    Vector<String> u, v = new Vector<String>();
    if ( (i == 0) && (j == 0) ) {
      v.add( "" );
    } else if ( j == 0 ) {
      u = manhattan( i-1, j );  v.add( "0"+u.get(0) );
    } else if ( i == 0 ) {
      u = manhattan( i, j-1 );  v.add( "1"+u.get(0) );
    } else {
      u = manhattan( i-1, j );  for ( String s : u ) { v.add( "0" + s ); }
      u = manhattan( i, j-1 );  for ( String s : u ) { v.add( "1" + s ); }
    }
    return v;
  }
```

Trasforma il metodo `manhattan` in un programma che applica opportunamente la tecnica *top-down* di *memoization*.



4. Programmazione in Java

La *trasposta* M^T di una matrice quadrata `M` si ottiene scambiando gli elementi in posizione simmetrica rispetto alla diagonale principale, cioè $M^T[i][j] = M[j][i]$ per ogni coppia di indici `i`, `j` della matrice. Definisci in Java un metodo statico `transp` che modifica gli elementi della matrice di numeri reali passata come argomento in modo da trasformarla nella relativa trasposta. In altri termini, se inizialmente l'oggetto `a` rappresenta la matrice `M`, dopo l'invocazione di `transp(a)` l'oggetto `a` rappresenta M^T.

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)