Corso di Programmazione

Esame del 13 Luglio 2007

cognome e nome

Risolvi i seguenti esercizi giustificando sinteticamente le risposte.

1. Ricorsione di coda

Trasforma la seguente procedura in un programma Scheme che applica la ricorsione di coda.

```
(define fe
  (lambda (n p)  ;  n, p naturali;  p > 1
    (if (= (remainder n p) 0)
        (+ (fe (quotient n p) p) 1)
        0)
  ))
```

2. Astrazione procedurale

Scrivi una procedura in Scheme che data una funzione *f*, definita per tutti gli argomenti interi e a valori interi, e dati due numeri interi *u* e *v*, con $u \leq v$, restituisce la coppia *(m . M)*, dove *m* e *M* sono rispettivamente il minimo e il massimo valore che *f* assume nell'intervallo di interi *[u, v]*.

3. Astrazione sui dati

Un albero di Huffman è un albero binario non vuoto le cui foglie sono etichettate con simboli dell'alfabeto diversi fra loro e i cui restanti nodi non sono etichettati e hanno sempre esattamente due figli. La codifica di un simbolo *s* dell'alfabeto, basata su un albero di Huffman *H*, è la stringa binaria che si determina scendendo lungo il percorso dalla radice di *H* all'unica sua foglia etichettata con *s*, e giustapponendo la cifra "0" per ogni spostamento a sinistra e la cifra "1" per ogni spostamento a destra. Per esempio, se per raggiungere la foglia etichettata con la lettera "b" a partire dalla radice si scende di tre livelli, spostandosi dapprima al sottoalbero sinistro, quindi per due volte a destra, allora la codifica di "b" (per quell'albero di Huffman) è data dalla stringa binaria "011".

Assumi che sia stata sviluppata una classe *HuffmanTree* in Java per rappresentare questa struttura. Tale classe è utilizzabile attraverso il seguente protocollo:

- un costruttore per creare alberi di un solo nodo, dove il simbolo è un oggetto di tipo *String* passato come argomento;
- un secondo costruttore per creare alberi di più nodi collegando la radice (non etichettata) ai due sottoalberi di Huffman, sinistro e destro, passati come argomento;
- un metodo *symbol()* che restituisce il simbolo rappresentato (*String* di un solo carattere) se l'albero di Huffman ha un solo nodo, *null* altrimenti;
- i metodi *left()* e *right()* che restituiscono i sottoalberi sinistro e destro, rispettivamente, di un albero con più nodi.

Completa il seguente metodo statico in Java che, dati un simbolo dell'alfabeto *s* e un albero di Huffman *H*, restituisce la codifica di *s* basata su *H*, quando esiste una foglia di *H* con etichetta *s*, *null* altrimenti.

```
public static String code( String s, HuffmanTree ht ) {

  if (  .....  ) {
    if ( (ht.symbol()).equals(s) ) {
      return "";
    } else {
      return null;
    }
  } else {
    String c = code(  .....  );
    if ( c == null ) {
      c =  .....  ;
      if ( c == null ) {
        return null;
      } else {
         .....  ;
      }
    } else {
       .....  ;
  }}
}
```

4. Programmazione dinamica

Trasforma la seguente procedura Scheme in un corrispondente programma in Java che applica opportunamente la tecnica di *programmazione dinamica*.

```
(define s
  (lambda (i j)        ; i, j valori naturali
    (cond ((= i 0) 1)
          ((= j 0) 0)
          (else (+ (s i (- j 1)) (* j (s (- i 1) j))))
    )))
```

5. Asserzioni e invarianti

Questo esercizio fa riferimento alla classe *PriorityQueue* discussa a lezione. Il metodo *delMax* è corretto se ogni sua esecuzione conserva l'invariante di classe: $(0 \le n \le M) \wedge (\forall i \in [2,n] \,.\, q[\lfloor i/2 \rfloor] \ge q[i])$, dove n è il numero di elementi presenti nella coda, limitato da M, e q l'array in cui sono rappresentati secondo lo schema dello *heap*. Qui sotto è riportata una versione di *delMax* leggermente semplificata, ma intercambiabile con quella vista a lezione. Immaginando di impostare la verifica della correttezza del codice di *delMax*, riporta precondizioni, postcondizioni, invarianti di ciclo, funzione di terminazione e opportune asserzioni in corrispondenza agli spazi introdotti dalle keyword *require*, *invariant*, *variant*, *check* e *ensure*. (La dimostrazione non è invece richiesta.) A tua scelta, puoi formalizzare le asserzioni nel linguaggio *Jass* oppure utilizzando la consueta notazione logico-matematica.

```
public void delMax() {
  /**  require  ..........................................................
       ..........................................................
  **/
  int x = q[n];
  int i = 1, j = 2 * i;

  while ( j < n )
    /**  invariant  ..........................................................
         ..........................................................
         ..........................................................
    **/
    /**  variant  ..........................................................  **/ {
    if ( (j+1 < n) && (q[j+1] > q[j]) ) {
      j = j + 1;
    }
    if ( q[j] > x ) {
      q[i] = q[j];
      i = j;  j = 2 * i;
    } else {
      break;
    }
  }
  /**  check  ..........................................................
       ..........................................................
  **/
  q[i] = x;
  n = n – 1;
  /**  ensure  ..........................................................
       ..........................................................
  **/
}
```