Corso di Programmazione

Esame del 18 Luglio 2003

cognome e nome

Risolvi i seguenti esercizi e riporta le soluzioni in modo chiaro su questo foglio, giustificandole sinteticamente.

1. Definizione di procedure in Scheme

Scrivi una procedura *compose* in Scheme che, data una lista $'(f_k, f_{k-1}, ..., f_2, f_1)$ di funzioni $f_i : \mathbf{D} \rightarrow \mathbf{D}$, definite per tutti gli elementi del dominio **D** e a valori in **D**, assuma come valore la funzione composta $g = f_k \circ f_{k-1} \circ ... \circ f_2 \circ f_1$. In altri termini:

$$(compose\ '(f_k, f_{k-1}, ..., f_2, f_1)) \quad \rightarrow \quad g$$

dove $g(x) = f_k(f_{k-1}(...f_2(f_1(x))...))$ per ogni $x \in \mathbf{D}$. (Se la lista passata come argomento è vuota, il valore della procedura è la funzione identità, elemento neutro della composizione.)

2. Dimostrazioni per induzione

Con riferimento all'esercizio precedente, dimostra per induzione che se tutti le funzioni f_i della lista sono rappresentate dalla stessa procedura (*lamda (x) (+ x 1)*) e se *n* è un naturale allora:

$$((compose\ '(f_k, f_{k-1}, ..., f_2, f_1))\ n) \quad \rightarrow \quad n+k$$

In particolare:

- Scrivi formalmente la proprietà che intendi dimostrare per induzione.
- Scrivi formalmente la proprietà che esprime il caso base.
- Scrivi formalmente l'ipotesi induttiva.
- Scrivi formalmente la proprietà che si deve dimostrare come passo induttivo.
- Dimostra formalmente il caso base.
- Dimostra formalmente il passo induttivo.

Corso di Programmazione

Esame del 18 Luglio 2003

cognome e nome

Risolvi il seguente esercizio e riporta la soluzione in modo chiaro su questo foglio, giustificandola sinteticamente.

3. Memoization

Applicando una logica analoga alla soluzione del problema della più lunga sottosequenza comune, la seguente procedura in Scheme valuta il numero di differenze fra due parole *u* e *v*:

```
(define diff
  (lambda (u v)
    (let ((m (string-length u))
          (n (string-length v))
          )
      (cond ((= m 0) n)
            ((= n 0) m)
            ((char=? (string-ref u 0) (string-ref v 0))
             (diff (substring u 1 m) (substring v 1 n)))
            (else
             (+ 1 (min (diff (substring u 1 m) v)
                       (diff u (substring v 1 n))))
             )))
    ))
```

Il valore assunto dalla procedura è dato dal numero di caratteri di *u* e di *v* che non appartengono a una sottosequenza comune di lunghezza massima, e che quindi non possono essere posti in corrispondenza fra le due parole. La valutazione ricorsiva di *(diff u v)* è inefficiente perché gli stessi calcoli vengono ripetuti un numero elevato di volte sugli stessi postfissi delle parole *u* e *v*; pertanto è conveniente applicare la tecnica di *memoization*. Definisci una procedura in Scheme basata sulla tecnica di memoization che, date due parole (string) *u* e *v*, assuma come valore *(diff u v)*. Per rappresentare la "storia", tenendo traccia dei calcoli già svolti, utilizza la struttura matriciale di supporto accessibile attraverso il seguente protocollo:

```
(make-matrix rows cols ini)    ; costruttore
(matrix-ref matrix i j)        ; accesso alla generica componente
(matrix-set! matrix i j val)   ; modifica della generica componente
```

dove il costruttore *make-matrix* assume come valore una matrice bidimensionale con indici di riga e colonna negli intervalli *[0, rows–1]* e *[0, cols–1]*, e le cui componenti sono inizialmente tutte impostate al valore *ini*; la procedura *matrix-ref* permette di accedere alla componente di indici *i, j*; infine *matrix-set!* modifica il valore della componente di indici *i, j*, assegnando il valore *val*. (La realizzazione della matrice non è richiesta.)

Corso di Programmazione

Esame del 18 Luglio 2003

cognome e nome

Risolvi i seguenti esercizi e riporta le soluzioni in modo chiaro su questo foglio, giustificandole sinteticamente.

4. Classi in Java

La classe *Tree* permette di istanziare alberi generici. Il relativo protocollo è così definito:

```
public class Tree {
  public Tree(int v);
  public appendToRoot(Tree subtree);
  public int rootValue();
  public int numberOfSubtrees();
  public Tree subtree(int k);
}
```

dove il costruttore crea un albero di un solo nodo; il metodo *appendToRoot* fa sì che la radice del parametro (albero) *subtree* abbia come padre la radice dell'albero che riceve il messaggio; i metodi *rootValue*, *numberOfSubtrees* e *subtree* restituiscono il valore intero associato alla radice, il numero di sottoalberi (figli della radice) e il *k*-imo sottoalbero, rispettivamente.
Un albero generico è uno *heap* se il valore associato a ciascuno dei nodi è maggiore dei valori associati ai rispettivi discendenti. Definisci un metodo statico a valori booleani in Java che, dato un albero di tipo *Tree*, permetta di verificare se si tratta di uno heap.

5. Classi in Java

Proponi una realizzazione della classe *Tree*, il cui protocollo è stato ntrodotto nell'esercizio precedente.