Corso di Programmazione

III Accertamento del 30 Giugno 2003 / A

cognome e nome

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

1. Memoization

Dato un insieme *S* di *n* elementi, il numero *g(n,k)* di sottoinsiemi di *S* aventi esattamente *k* elementi, con $0 \le k \le n$, è definito dalle relazioni:

$g(n,0) = g(n,n) = 1$ per $n \ge 0$;

$g(n,k) = g(n-1,k-1) + g(n-1,k)$ per $0 < k < n$.

La valutazione ricorsiva di *g(n,k)* è inefficiente perché gli stessi calcoli vengono ripetuti un numero elevato di volte, quindi è conveniente applicare la tecnica di *memoization*. Definisci una procedura in Scheme basata sulla tecnica di memoization che, dati *n*, *k* naturali, assuma come valore *g(n,k)*. Per rappresentare la "storia" (i calcoli già svolti) utilizza la struttura matriciale di supporto *Structure*, accessibile attraverso il seguente protocollo:

`(make-struct` *`imax jmax ini`*`)` : $\mathbf{N} \times \mathbf{N} \times \mathbf{N} \rightarrow$ *Structure*

`(struct-ref` *`str i j`*`)` : *Structure* $\times \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$

`(struct-set!` *`str i j val`*`)` : *Structure* $\times \mathbf{N} \times \mathbf{N} \times \mathbf{N} \rightarrow$ *Structure*

dove il costruttore *make-structure* assume come valore una matrice bidimensionale con indici di riga e colonna negli intervalli *[0, imax]* e *[0, jmax]*, e le cui componenti sono inizialmente tutte impostate al valore *ini*; la procedura *struct-ref* permette di accedere alla componente di indici *i*, *j* (con $j \le i$); infine *struct-set!* modifica il valore della componente di indici *i*, *j* (con $j \le i$), assegnando il valore *val*. (La realizzazione di *Structure* non è richiesta in questo esercizio.)

2. Realizzazione di strutture dati in Scheme

La rappresentazione del dato astratto *Structure*, specificato nell'esercizio precedente, può essere specializzata tenendo conto che i valori *g(i,j)* sono simmetrici, più precisamente:

$g(i,i-j) = g(i,j)$ per $0 \le j \le i$.

Questa osservazione suggerisce di rappresentare esplicitamente solo le componenti della matrice con indice di colonna minore o uguale alla metà dell'indice di riga ($j \le i/2$), cioè le componenti *<0,0>*, *<1,0>*, *<2,0>*, *<2,1>*, *<3,0>*, *<3,1>*, *<4,0>*, *<4,1>*, *<4,2>*, *<5,0>*, e così via.

Realizza il protocollo di *Structure*, così come specificato nell'esercizio 1, utilizzando una rappresentazione "economica" della matrice che sfrutti le simmetrie qui evidenziate. Le procedure proposte devono rimanere compatibili con la soluzione dell'esercizio 1.

Corso di Programmazione

III Accertamento del 30 Giugno 2003 / B

cognome e nome

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

1. Memoization

Dato un insieme *S* non vuoto di *n* elementi, il numero *h(n,k)* di partizioni di *S* in esattamente *k* sottoinsiemi disgiunti e non vuoti, con $1 \le k \le n$, è definito dalle relazioni:

$h(n,1) = h(n,n) = 1$ per $n \ge 1$;

$h(n,k) = h(n-1,k-1) + k \cdot h(n-1,k)$ per $0 < k < n$.

La valutazione ricorsiva di *h(n,k)* è inefficiente perché gli stessi calcoli vengono ripetuti un numero elevato di volte, quindi è conveniente applicare la tecnica di *memoization*. Definisci una procedura in Scheme basata sulla tecnica di memoization che, dati *n*, *k* naturali positivi, assuma come valore *h(n,k)*. Per rappresentare la "storia" (i calcoli già svolti) utilizza la struttura matriciale di supporto *Structure*, accessibile attraverso il seguente protocollo:

`(make-struct` *`imax jmax ini`*`)` : $\mathbf{N}^+ \times \mathbf{N}^+ \times \mathbf{N} \rightarrow$ *Structure*

`(struct-ref` *`str i j`*`)` : *Structure* $\times \mathbf{N}^+ \times \mathbf{N}^+ \rightarrow \mathbf{N}$

`(struct-set!` *`str i j val`*`)` : *Structure* $\times \mathbf{N}^+ \times \mathbf{N}^+ \times \mathbf{N} \rightarrow$ *Structure*

dove il costruttore *make-structure* assume come valore una matrice bidimensionale con indici di riga e colonna negli intervalli *[1, imax]* e *[1, jmax]*, e le cui componenti sono inizialmente tutte impostate al valore *ini*; la procedura *struct-ref* permette di accedere alla componente di indici *i, j* (con $j \le i$); infine *struct-set!* modifica il valore della componente di indici *i, j* (con $j \le i$), assegnando il valore *val*. (La realizzazione di *Structure* non è richiesta in questo esercizio.)

2. Realizzazione di strutture dati in Scheme

La rappresentazione del dato astratto *Structure*, specificato nell'esercizio precedente, può essere specializzata tenendo conto che, al fine di valutare *h(n,k)*, interessano solo valori *h(i,j)* calcolati per indici *i, j* tali che $1 \le j \le min(i,k)$. Questa osservazione suggerisce di rappresentare esplicitamente solo le componenti della matrice con indice di colonna compreso fra *1* e il minimo fra l'indice di riga e *k*, cioè le componenti *<1,1>*, *<2,1>*, *<2,2>*, *<3,1>*, *<3,2>*, ..., *<i,1>*, *<i,2>*, ..., *<i , min(i,k) >*, e così via.

Realizza il protocollo di *Structure*, così come specificato nell'esercizio 1, utilizzando una rappresentazione "economica" della matrice che sfrutti le informazioni sul campo di variabilità dell'indice di colonna. Le procedure proposte devono rimanere compatibili con la soluzione dell'esercizio 1.

Corso di Programmazione

III Accertamento del 30 Giugno 2003 / A

cognome e nome

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

3. Astrazione sui dati in Java

Definisci in Java una classe *Structure* per rappresentare matrici sfruttando le simmetrie descritte nell'esercizio 2. In altre parole, ti è richiesto di "tradurre in Java" la soluzione dell'esercizio 2, scegliendo gli strumenti del linguaggio più opportuni allo scopo. In particolare, il protocollo deve essere costituito dal costruttore *Structure* e dai metodi *structRef* e *structSet*, con funzioni analoghe alle corrispondenti procedure Scheme specificate nell'esercizio 1.

4. Programmazione in Java

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

```
public class Tree {
  public Tree(int v);
  public void 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 associato alla radice, il numero di sottoalberi (figli della radice) e il *k*-imo sottoalbero, rispettivamente.

Definisci un metodo statico in Java che, dato un albero, restituisca il massimo dei valori associati ai nodi.

Corso di Programmazione

III Accertamento del 30 Giugno 2003 / B

cognome e nome

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

3. Astrazione sui dati in Java

Definisci in Java una classe *Structure* per rappresentare matrici specializzate come descritto nell'esercizio 2. In altre parole, ti è richiesto di "tradurre in Java" la soluzione dell'esercizio 2, scegliendo gli strumenti del linguaggio più opportuni allo scopo. In particolare, il protocollo deve essere costituito dal costruttore *Structure* e dai metodi *structRef* e *structSet*, con funzioni analoghe alle corrispondenti procedure Scheme specificate nell'esercizio 1.

4. Programmazione in Java

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

```
public class Tree {
  public Tree(int v);
  public void 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 associato alla radice, il numero di sottoalberi (figli della radice) e il *k*-imo sottoalbero, rispettivamente.

Definisci un metodo statico in Java che, dato un albero, restituisca il minimo dei valori associati ai nodi.

Soluzioni / A

1. Memoization

```
(define g
  (lambda (n k)  ; 0 <= k <= n
    (memoization-g n k (make-struct n k 0))
    ))

(define memoization-g
  (lambda (i j history)
    (if (= (struct-ref history i j) 0)
        (struct-set! history i j
                     (if (or (= j 0) (= j i))
                         1
                         (+ (memoization-g (- i 1) (- j 1) history)
                            (memoization-g (- i 1) j history))
                         ))
        )
    (struct-ref history i j)
    ))
```

2. Realizzazione di strutture dati in Scheme

```
(define make-struct
  (lambda (imax jmax ini)
    (let ((str (make-vector (+ imax 1)))
          )
      (struct-rows! str imax jmax ini)
      str
      )
    ))

(define struct-rows!
  (lambda (str i j ini)
    (if (> i 0)
        (struct-rows! str (- i 1) j ini)
        )
    (vector-set! str i (make-vector (ceiling (/ (+ i 1) 2)) ini))
    ))

(define struct-ref
  (lambda (str i j)
    (vector-ref
     (vector-ref str i)
     (if (<= j (quotient i 2))
         j
         (- i j)
         ))
    ))

(define struct-set!
  (lambda (str i j x)
    (vector-set!
     (vector-ref str i)
     (if (<= j (quotient i 2))
         j
         (- i j)
         )
     x)
    ))
```

Soluzioni / B

1. Memoization

```
(define h
  (lambda (n k)  ; 1 <= k <= n
    (memoization-h n k (make-struct n k 0))
    ))

(define memoization-h
  (lambda (i j history)
    (if (= (struct-ref history i j) 0)
        (struct-set! history i j
                     (if (or (= j 1) (= j i))
                         1
                         (+ (memoization-h (- i 1) (- j 1) history)
                            (* j (memoization-h (- i 1) j history))
                            )))
        )
    (struct-ref history i j)
    ))
```

2. Realizzazione di strutture dati in Scheme

```
(define make-struct
  (lambda (imax jmax ini)
    (let ((str (make-vector imax))
          )
      (struct-rows! str imax jmax ini)
      str
      )
    ))

(define struct-rows!
  (lambda (str i j ini)
    (if (> i 0)
        (begin
          (struct-rows! str (- i 1) j ini)
          (vector-set! str (- i 1) (make-vector (min i j) ini))
          ))
    ))


(define struct-ref
  (lambda (str i j)
    (vector-ref (vector-ref str (- i 1)) (- j 1))
    ))


(define struct-set!
  (lambda (str i j x)
    (vector-set! (vector-ref str (- i 1)) (- j 1) x)
    ))
```

Soluzioni / A

3. Astrazione sui dati in Java

```
public class Structure {

  private int[][] str;

  public Structure(int n, int k, int ini) {

    str = new int[n+1][];

    for (int i=0; i<=n; i=i+1) {

      int m = (int) (i/2) + 1;
      str[i] = new int[m];

      for (int j=0; j<m; j=j+1) {
        str[i][j] = ini;
      }
    }
  }

  public int structRef(int i, int j) {

    if (2*j <= i) {
      return ( str[i][j] );
    } else {
      return ( str[i][i-j] );
    }
  }

  public void structSet(int i, int j, int x) {

    if (2*j <= i) {
      str[i][j] = x;
    } else {
      str[i][i-j] = x;
    }
  }
}
```

4. Programmazione in Java

```
public static int maxNode(Tree tree) {

  int m = tree.rootValue();

  for (int k=1; k<=tree.numberOfSubtrees(); k=k+1) {

    int v = maxNode(tree.subtree(k));

    if (m < v) {
      m = v;
    }
  }
  return m;
}
```

Soluzioni / B

3. Astrazione sui dati in Java

```
public class Structure {

  private int[][] str;

  public Structure(int n, int k, int ini) {

    str = new int[n][];

    for (int i=0; i<n; i=i+1) {

      int m = Math.min(i+1,k);
      str[i] = new int[m];

      for (int j=0; j<m; j=j+1) {
        str[i][j] = ini;
      }
    }
  }

  public int structRef(int i, int j) {

    return ( str[i-1][j-1] );
  }

  public void structSet(int i, int j, int x) {

    str[i-1][j-1] = x;
  }
}
```

4. Programmazione in Java

```
public static int minNode(Tree tree) {

  int m = tree.rootValue();

  for (int k=1; k<=tree.numberOfSubtrees(); k=k+1) {

    int v = minNode(tree.subtree(k));

    if (m > v) {
      m = v;
    }
  }
  return m;
}
```