Corso di Programmazione

Prova scritta d'esame del 2 Settembre 2024

cognome e nome

Riporta in modo chiaro negli appositi spazi le soluzioni degli esercizi, oppure precise indicazioni se alcune soluzioni si trovano in un foglio separato. Scrivi inoltre il tuo nome nell'intestazione e su ciascun ulteriore foglio che intendi consegnare.

1. Programmazione in Scheme

Data una funzione $f : D \to D$, un elemento $x \in D$ e un intero $k > 0$, la procedura `iter-fun` restituisce una lista di k elementi di D, ciascuno dei quali risulta da iterazioni dell'applicazione di f con argomento x, iterazioni di ordine via via decrescente a partire dal primo termine della lista in cui f è applicata $k–1$ volte e fino all'iterazione di ordine 0 che assume il valore dell'argomento x stesso, come illustrato dal seguente schema di valutazione:

`(iter-fun` *f x k*`)` → (*f(f(...f(x)...))* . . . . . *f(f(f(x)))* *f(f(x))* *f(x)* *x*)

In altri termini, a parte l'ultimo elemento x, ogni altro elemento si ottiene applicando f a quello seguente. Esempi:

`(iter-fun sqrt 256 4)` → `(2 4 16 256)` [`sqrt` denota la funzione radice quadrata]

`(iter-fun (lambda (x) (string-append x x)) "<>" 3)` → `("<><><><>" "<><>" "<>")`

Scrivi un programma per realizzare la procedura `iter-fun`.

2. Argomenti procedurali in Scheme

Data una lista *ordinata* (crescente) *s* di interi positivi diversi fra loro e dati due interi *x*, *y*, siamo interessati alle combinazioni di elementi di *s*, senza ripetizioni, la cui somma è compresa nell'intervallo [*x, y*], inclusi gli estremi. In particolare, la seguente procedura, `comb-count`, calcola il numero di combinazioni distinte con questa proprietà:

```
(define comb-count  ; val: intero
  (lambda (s x y)   ; s: lista ordinata di interi positivi (diversi fra loro), x, y: interi
    (if (or (null? s) (> (car s) y))
        0
        (let ((e (car s)))
          (let ((n (+ (comb-count (cdr s) (- x e) (- y e))  ; e "candiato" per la somma
                      (comb-count (cdr s) x y)              ; e scartato
                      )))
            (if (and (<= x e) (<= e y)) (+ 1 n) n)
            ))
      )))
```

La procedura `comb` impostata nel riquadro è invece intesa a determinare la lista di tali combinazioni, ciascuna combinazione rappresentata a sua volta da una lista ordinata di interi.

Per esempio:

```
(comb '(1 5 9) 7 8)  →  ()

(comb '(1 2 3 4 5) 4 6)  →  ((1 2 3) (1 3) (1 4) (1 5) (2 3) (2 4) (4) (5))

(comb '(2 3 5 7 11 13) 12 14)  →  ((2 3 7) (2 5 7) (2 11) (3 11) (5 7) (13))
```

Completa il programma in Scheme impostato nel riquadro per realizzare la procedura `comb`.

```
(define comb        ; val: lista di liste ordinate
  (lambda (s x y)   ; s: lista ordinata di interi positivi (diversi fra loro), x, y: interi

    (if (or (null? s) (> (car s) y))

        .....

        (let ((e (car s)))

          (let ((u ( ..... 

                     (map ..... 
                          (comb (cdr s) (- x e) (- y e))
                          )
                     (comb (cdr s) x y)
                     )))

            (if (and (<= x e) (<= e y))

                .....

                ..... )

            ))
      )))
```

3. Memoization

Dati due interi *n*, *k*, tali che $0 \leq k \leq n$, il seguente programma ricorsivo in Java calcola i (moduli dei) *numeri di Stirling di prima specie*:

```
public static long st1( int n, int k ) {    // 0 ≤ k ≤ n

  if ( k == n ) {
    return 1;
  } else if ( k == 0 ) {
    return 0;
  } else {
    return ( (n-1)*st1(n-1,k) + st1(n-1,k-1) );
  }
}
```

Il programma impostato nel riquadro è inteso per applicare una tecnica *top-down* di *memoization* al fine di rendere più efficiente la computazione ricorsiva di `st1`. Completa la definizione dei metodi statici `st1Mem` e `st1Rec`.

```
public static long st1Mem( int n, int k ) {    // 0 ≤ k ≤ n

  .......... mem = .................................................. ;

  ....................................................................

  ....................................................................

  ....................................................................

  ....................................................................

  ....................................................................

  ....................................................................

  return st1Rec( .................................................. );
}

private static long st1Rec( ........................................ ) {

  if ( ............................ == UNKNOWN ) {

    ..................................................................

    ..................................................................

    ..................................................................

    ..................................................................

    ..................................................................

    ..................................................................

    ..................................................................
  }
  return ............................................................ ;
}

private static final int UNKNOWN = .................................. ;
```

4. Ricorsione e iterazione

La procedura ricorsiva `comb-count`, definita nell'esercizio **2**, consente una semplice rielaborazione iterativa tramite stack dopo aver fatto le seguenti osservazioni: (i) dei tre parametri *s*, *x*, *y*, solo i primi due sono indipendenti in quanto la differenza fra *y* e *x* non cambia nel corso delle invocazioni ricorsive; (ii) ogni invocazione di `comb-count` è quindi completamente determinata da una lista di interi (*s*) e da un ulteriore intero (*x*) e i *frame* da introdurre nello stack possono essere convenientemente rappresentati da istanze di `IntSList` semplicemente aggiungendo l'intero *x* in testa alla rappresentazione di *s*; (iii) ogni volta che viene identificata una combinazione utile si aggiunge 1 ai risultati determinati ricorsivamente, per cui la funzionalità di contatore può essere svolta da una variabile "globale" condivisa dalle ricorsioni. Completa la definizione del metodo `combCountIter`, impostato nel riquadro, per trasformare la procedura `comb-count` in un corrispondente programma iterativo in Java che si avvale di uno stack di `IntSList`.

```
public static int combCountIter( IntSList s, int x, int y ) {

  int delta = y - x;   // la differenza y–x è invariante
  int count = 0;       // contatore delle soluzioni

  Stack<IntSList> stack = new Stack<IntSList>();
  IntSList frame = s.cons(x);
  stack.push( frame );

  do {

    frame = ..... ;

    s = ..... ;

    x = ..... ;

    y = ..... ;


    if ( !s.isNull() && ..... ) {
      int e = s.car();
      s = s.cdr();

      stack.push( ..... );

      ..... ;

      if ( ..... ) {
        count = count + 1;
    }}

  } while ( ..... );

  return count;
}
```