Corso di Programmazione

I Accertamento del 21 Gennaio 2013

cognome e nome

Risolvi i seguenti esercizi, riporta le soluzioni in modo chiaro negli appositi spazi e giustifica sinteticamente le risposte. Dovrai poi consegnare queste schede con le soluzioni, avendo cura di scrivere il tuo nome nell'intestazione e su ciascun eventuale foglio aggiuntivo.

1. Definizione di procedure in Scheme

Considera una variante del problema di tassellazione lineare per cui si vuole conoscere in quanti modi diversi sia possibile tassellare un cordolo di lunghezza *n* e altezza 1 utilizzando due tipi di piastrelle: una piastrella quadrata con il lato di lunghezza 1 e una rettangolare con i lati di lunghezza 2 e 1. Rispetto al caso analizzato a lezione, però, si impone come ulteriore vincolo che non sia consentito utilizzare più di *k* piastrelle rettangolari. In altri termini, sono disponibili un numero sostanzialmente illimitato di piastrelle, ma solo *k* piastrelle rettangolari.

Definisci una procedura `xtessellations` che, dati la lunghezza *n* del cordolo e il limite *k* al numero di piastrelle rettangolari, risolve il problema proposto. Esempi:

```
(xtessellations 5 0)    →    1              (xtessellations 5 3)    →    8
(xtessellations 5 1)    →    5              (xtessellations 7 1)    →    7
(xtessellations 5 2)    →    8              (xtessellations 7 2)    →    17
```

```
(define xtessellations
  (lambda (n k)
    (if (or (< n 2) (= k 0))
        1
        (+ (xtessellations (- n 1) k) (xtessellations (- n 2) (- k 1)))
        )))
```

oppure (qual è la logica della seconda soluzione?)

```
(define xtessellations
  (lambda (n k)
    (cond ((= k 0)
           1)
          ((> (* 2 k) n)
           (xtessellations n (- k 1)))
          (else
           (+ (xtessellations n (- k 1)) (disp (- n k) k)))
          )))

(define disp     ; numero di disposizioni  D(n,k)
  (lambda (n k)
    (if (= k 0)
        1
        (/ (* (disp (- n 1) (- k 1)) n) k)
        )))
```

2. Programmi in Scheme

Facendo riferimento alla procedura `tilings` definita nella pagina seguente, determina il risultato della valutazione di ciascuna delle espressioni riportate qui sotto:

```
(tilings 1)   →    2              (tilings 4)   →    13
(tilings 2)   →    4              (tilings 6)   →    44
(tilings 3)   →    7              (tilings 8)   →    149
```

```
(define tilings
  (lambda (n)
    (if (= n 0)
        1
        (+ (tilings (- n 1)) (red-tilings (- n 1)))
        )))

(define red-tilings
  (lambda (n)
    (cond ((= n 0) 1)
          ((= n 1) 2)
          (else (+ (tilings (- n 1)) (tilings (- n 2))))
          )))
```

3. Procedure con valori procedurali

Il *cifrario di Vigenère* (1523–1596) rappresenta una generalizzazione del cifrario di Cesare, in base al quale la rotazione delle lettere dell'alfabeto non è costante, ma varia ciclicamente in funzione di una parola chiave, concordata fra mittente e destinatario, che rappresenta le successive "mappature" della lettera iniziale `A`. Il meccanismo si può illustrare più facilmente ricorrendo a un esempio. Supponiamo che la parola chiave sia `"AKEY"`. Questo significa che la prima lettera del messaggio verrà ruotata di 0 posizioni (`A`→**`A`**, `B`→`B`, `C`→`C`, ...); la seconda lettera del messaggio di 10 posizioni (`A`→**`K`**, `B`→`L`, `C`→`M`, …); la terza di 4 (`A`→**`E`**, `B`→`F`, `C`→`G`, …); la quarta di 24 (`A`→**`Y`**, `B`→`Z`, `C`→`A`, …); la quinta di nuovo di 0 posizioni (`A`→`A`, …) e così via ripetendo rotazioni della stessa entità con periodo 4 (= lunghezza della chiave). Quindi, a partire dal messaggio in chiaro `"ASHORTSAMPLEMESSAGE"` si può procedere secondo questo schema:

`A S H O  R T S A  M P L E  M E S S  A G E`	*messaggio in chiaro*
`A K E Y  A K E Y  A K E Y  A K E Y  A K E`	*parola chiave ripetuta*
`A C L M  R D W Y  M Z P C  M O W Q  A Q I`	*messaggio crittato*

Da cui risulta il testo crittato (terza riga): `"ACLMRDWYMZPCMOWQAQI"`.

Data una parola chiave *key* (stringa di lettere maiuscole), la procedura `vigenere-cipher` restituisce la procedura di crittazione che applica il cifrario di Vigenère con chiave *key* per trasformare un messaggio in chiaro (stringa di lettere maiuscole) nel corrispondente testo crittato (stringa di lettere maiuscole). Per esempio:

```
(define encrypt (vigenere-cipher "AKEY"))

(encrypt "ASHORTSAMPLEMESSAGE")   →   "ACLMRDWYMZPCMOWQAQI"
```

Completa la procedura riportata qui sotto, che realizza `vigenere-cipher`, introducendo il codice Scheme appropriato negli spazi indicati a tratto punteggiato.

```
(define vigenere-cipher
  (lambda (key)

    ( lambda (msg)
      (if (string=? msg "")

          ""
          (let ((r (- (char->integer (string-ref key 0)) cA))
                (c (- (char->integer (string-ref msg 0)) cA))
                (k (string-append (substring key 1) (substring key 0 1)))
                )
            (string-append

              (string (integer->char (+ cA   (remainder (+ c r) 26)  )))

              ( (vigenere-cipher k)   (substring msg 1))
              ))
          ))
    ))

(define cA (char->integer #\A))
```

4. Verifica formale della correttezza

```
(define g
  (lambda (i j)  ; i, j interi non negativi
    (+
     (if (or (= i 0) (= j 0))  0  (- (g (- i 1) (- j 1)) 1))
     i
     j)
    ))
```

In relazione alla procedura definita sopra è possibile dimostrare che per tutte le coppie di valori interi positivi x, y:

$$(g\ x\ y) \quad \rightarrow \quad xy + |x-y|$$

Dimostra per induzione questa proprietà; in particolare:

- Formalizza la proprietà che esprime il caso / i casi base:

  [Dimostrazione per induzione su $k = min(x,y)$]

  $\forall\, x, y \in \mathbb{N}$ t.c. $min(x,y) = 0$. $(g\ x\ y) \rightarrow xy + |x-y|$

- Formalizza l'ipotesi induttiva: preso $k \in \mathbb{N}$

  $\forall\, x, y \in \mathbb{N}$ t.c. $min(x,y) = k$. $(g\ x\ y) \rightarrow xy + |x-y|$

- Formalizza la proprietà da dimostrare come passo induttivo: per k scelto sopra

  $\forall\, x, y \in \mathbb{N}$ t.c. $min(x,y) = k+1$. $(g\ x\ y) \rightarrow xy + |x-y|$

- Dimostra il caso / i casi base:

  $(g\ x\ y) \rightarrow (+\ (\text{if } (\text{or } (=\ x\ 0)\ (=\ y\ 0))\ 0\ (-\ (g\ (-\ x\ 1)\ (-\ y\ 1))\ 1))\ x\ y)$

  $\rightarrow (+\ (\text{if } \textit{true}\ 0\ (-\ (g\ (-\ x\ 1)\ (-\ y\ 1))\ 1))\ x\ y)$; *poiché* $min(x,y) = 0$

  $\rightarrow (+\ 0\ x\ y)$

  $\rightarrow x + y = 0 + |\, max(x,y) - min(x,y)\, | = xy + |x-y|$

- Dimostra il passo induttivo: [$\forall\, x, y \in \mathbb{N}$ t.c. $min(x,y) = k+1$]

  $(g\ x\ y) \rightarrow (+\ (\text{if } (\text{or } (=\ x\ 0)\ (=\ y\ 0))\ 0\ (-\ (g\ (-\ x\ 1)\ (-\ y\ 1))\ 1))\ x\ y)$

  $\rightarrow (+\ (\text{if } \textit{false}\ 0\ (-\ (g\ (-\ x\ 1)\ (-\ y\ 1))\ 1))\ x\ y)$; *poiché* $min(x,y) = k+1 > 0$

  $\rightarrow (+\ (-\ (g\ x-1\ y-1))\ 1)\ x\ y)$

  $\rightarrow (+\ (-\ ((x-1)(y-1) + |\, (x-1) - (y-1)\, |)\ 1)\ x\ y)$; *per l'ipotesi induttiva,*
  ; *poiché* $min(x-1,y-1) = k$

  $\rightarrow (xy - x - y + 1 + |x-y|) - 1 + x + y = xy + |x-y|$

5. Programmi in Scheme

Un numero x compreso nell'intervallo $]–0.5, +0.5[$ (cioè tale che $|x| < ½$) può essere codificato, almeno in forma approssimata, tramite una sequenza finita di cifre del sistema ternario bilanciato (**–**/**.**/**+**), cifre che comparirebbero come parte frazionaria in questa notazione. Definisci una procedura `fract-part` che, data una stringa *btf* di **–**/**.**/**+**, restituisce il valore numerico x di *btf*, interpretata come parte frazionaria nel sistema ternario bilanciato. Esempi:

```
(fract-part "--") → -4/9      (fract-part "-.+") → -8/27     (fract-part ".") → 0
(fract-part ".++") → 4/27     (fract-part "+.-") → 8/27      (fract-part "+") → 1/3
```

```
(define fract-part
  (lambda (btr)
    (if (string=? btr "")
        0
        (/ (+ (btr-val (string-ref btr 0))
              (fract-part (substring btr 1)))
           3)
        )))

(define btr-val
  (lambda (dgt)
    (cond ((char=? dgt #\-) -1)
          ((char=? dgt #\.)  0)
          ((char=? dgt #\+) +1)
          )))
```

oppure, per esempio:

```
(define fract-part
  (lambda (btr)
    (fract-rec btr 0 1)
    ))

(define fract-rec
  (lambda (btr n d)
    (if (string=? btr "")
        (/ n d)
        (fract-rec
         (substring btr 1)
         (+ (* 3 n) (btr-val (string-ref btr 0)))
         (* 3 d)
         ))
    ))
```

6. Astrazione funzionale (ai fini della valutazione, questo problema ha un peso minore degli altri)

Riesci a intuire quale problema risolve (o che funzione calcola) la procedura `tilings` definita nell'esercizio **2**?

Numero di modi diversi in cui si può tassellare un cordolo di lunghezza n con piastrelle quadrate rosse e blu di lato unitario, dato il vincolo di non avere mai raggruppamenti di tre o più piastrelle rosse adiacenti.