Corso di Programmazione

I Prova di accertamento del 21 Gennaio 2022 / A

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 stringa *s* e un intero $k > 0$, la procedura `cyclic-pattern` restituisce: (a) la stringa *p* (*pattern*) di lughezza *k*, se *s* è costituita dalla ripetizione ciclica di *p* una o più volte; (b) la stringa vuota altrimenti. In particolare, nel caso (a) le ripetizioni di *p* si intendono consecutive (senza caratteri estranei interposti) e complete (non troncate). Esempi:

```
(cyclic-pattern "" 3)          → ""       (cyclic-pattern "abcabcab" 3)  → ""
(cyclic-pattern "abc" 3)       → "abc"    (cyclic-pattern "abcabcacb" 3) → ""
(cyclic-pattern "abcabcabc" 3) → "abc"    (cyclic-pattern "abcabcabc" 2) → ""
```

Scrivi un programma in Scheme per realizzare la procedura `cyclic-pattern`.

2. Ricorsione ad albero

Le soluzioni dei problemi di tassellazione discussi a lezione possono essere codificate come segue:

```
(define tess-1-2   ; val: intero
  (lambda (n)      ; n: intero non negativo
    (if (< n 2)
        1
        (+ (tess-1-2 (- n 1))
           (tess-1-2 (- n 2)))
        )))
```

```
(define tess-r-b   ; val: intero
  (lambda (n)      ; n: intero non negativo
    (if (< n 2)
        (+ n 1)
        (+ (tess-r-b (- n 1))
           (tess-r-b (- n 2)))
        )))
```

Le due procedure calcolano in quanti modi diversi si può tassellare un cordolo di lunghezza $n \geq 0$, utilizzando piastrelle di larghezza *1* e *2* (`tess-1-2`); oppure utilizzando piastrelle di larghezza *1* ma di colore diverso, rosso e blu, con il vincolo che due piastrelle rosse non possono mai essere adiacenti (`tess-r-b`).

Applicando una impostazione analoga, definisci una procedura `tess-1x-2` per determinare in quanti modi diversi si può tassellare un cordolo di lunghezza $n \geq 0$, utilizzando piastrelle di larghezza *1* e *2*, ma con l'ulteriore vincolo che due piastrelle di lunghezza *1* non possono mai essere adiacenti.

3. Argomenti procedurali

Il seguente programma calcola il numero di percorsi di Manhattan diversi, costituiti da *i* spostamenti verticali e *j* spostamenti orizzontali, per cui si richiede che non si possano effettuare più di *k* spostamenti orizzontali consecutivi:

```
(define manh          ; val: intero
  (lambda (i j k)     ; i, j, k: interi ≥ 0
    (manh-rec i j k k)
    ))
```

```
(define manh-rec
  (lambda (i j k v)
    (cond ((= i 0)
           (if (> j v) 0 1))
          ((= j 0)
           1)
          ((= v 0)
           (manh-rec (- i 1) j k k))
          (else
           (+ (manh-rec (- i 1) j k k)
              (manh-rec i (- j 1) k (- v 1))
              ))
          )))
```

Il programma impostato nel riquadro, ha invece l'obiettivo di restituire *la lista di tali percorsi*, ciascuno codificato da una stringa di `0` e `1`, dove lo zero rappresenta uno spostamento verticale (in basso) e l'uno rappresenta uno spostamento orizzontale (a destra). Esempi:

```
(paths 5 1 2)  →  ("000001" "000010" "000100" "001000" "010000" "100000")
(paths 1 5 2)  →  ()
(paths 2 2 1)  →  ("0101" "1001" "1010")
```

Completa la procedura ricorsiva `paths-rec` inserendo espressioni appropriate negli spazi indicati. (Nota: dati un intero *n* e un carattere *c*, la procedura predefinita `make-string` restituisce la stringa composta da *n* ripetizioni di *c*.)

```
(define paths                ; val:  lista di stringhe
  (lambda (i j k)            ; i, j, k:  interi non negativi
    (paths-rec i j k k)
    ))

(define paths-rec
  (lambda (i j k v)
    (cond ((= i 0)
           .................................................................... )
          ((= j 0)
           ( ...................... (make-string i #\0)))
          ((= v 0)
           (map ...............................................................
                (paths-rec (- i 1) j k k)))
          (else
           ( ......................
            (map ..............................................................
                 (paths-rec (- i 1) j k k))
            (map ..............................................................
                 (paths-rec i (- j 1) k (- v 1)))))
          )
    ))
```

4. Verifica formale della correttezza

Considera la procedura `f` riportata qui a lato, il cui argomento b è una stringa non vuota costituita esclusivamente dalle cifre `0` e `1` e il cui ultimo carattere (quello più a destra) deve essere `1`.

Per ogni stringa t di lunghezza $k \geq 3$ composta da $k–3$ cifre `0` seguite dalla terna `111` si può dimostrare che:

`(f` t`)` $\quad \longrightarrow \quad 3 \cdot 2^{k-2} + 1 \quad$ (*)

```
(define f            ; val: intero
  (lambda (b)        ; b: stringa di 0/1 conclusa da 1
    (cond ((string=? b "1")
            1)
          ((char=? (string-ref b 0) #\0)
            (- (* 2 (f (substring b 1))) 1))
          (else
            (+ (* 2 (f (substring b 1))) 1))
          )
    ))
```

(In altri termini t potrà essere: `"111"`, `"0111"`, `"00111"`, `"000111"`, `"00 … 0111"`.)

Dimostra la proprietà (*) per induzione, attenendoti allo schema delineato qui sotto.

- Indica il valore rispetto al quale intendi impostare la dimostrazione per induzione:

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

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra caso/i base e passo induttivo:

Corso di Programmazione

I Prova di accertamento del 21 Gennaio 2022 / B

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 stringa *s* e un intero $k > 0$, la procedura `cyclic-number` restituisce il numero di ripetizioni in *s* della sua parte iniziale *p* (*pattern*) di lughezza *k*; se la lunghezza della stringa è inferiore a *k* restituisce *0*. In particolare, le ripetizioni di *p* si intendono consecutive (senza caratteri estranei interposti) e complete (non troncate). Esempi:

```
(cyclic-number "" 3)      →  0
(cyclic-number "ab" 3)    →  0
(cyclic-number "abc" 3)   →  1
(cyclic-number "abcabcabc" 3)  →  3
(cyclic-number "abcabcabc" 2)  →  1
(cyclic-number "abcabcacb" 3)  →  2
```

Scrivi un programma in Scheme per realizzare la procedura `cyclic-number`.

2. Ricorsione ad albero

Le soluzioni dei problemi di tassellazione discussi a lezione possono essere codificate come segue:

```
(define tess-1-2   ; val: intero
  (lambda (n)      ; n: intero non negativo
    (if (< n 2)
        1
        (+ (tess-1-2 (- n 1))
           (tess-1-2 (- n 2)))
        )))
```

```
(define tess-r-b   ; val: intero
  (lambda (n)      ; n: intero non negativo
    (if (< n 2)
        (+ n 1)
        (+ (tess-r-b (- n 1))
           (tess-r-b (- n 2)))
        )))
```

Le due procedure calcolano in quanti modi diversi si può tassellare un cordolo di lunghezza $n \geq 0$, utilizzando piastrelle di larghezza *1* e *2* (`tess-1-2`); oppure utilizzando piastrelle di larghezza *1* ma di colore diverso, rosso e blu, con il vincolo che due piastrelle rosse non possono mai essere adiacenti (`tess-r-b`).

Applicando una impostazione analoga, definisci una procedura `tess-1-2x` per determinare in quanti modi diversi si può tassellare un cordolo di lunghezza $n \geq 0$, utilizzando piastrelle di larghezza *1* e *2*, ma con l'ulteriore vincolo che due piastrelle di lunghezza *2* non possono mai essere adiacenti.

3. Argomenti procedurali

Il seguente programma calcola il numero di percorsi di Manhattan diversi, costituiti da *i* spostamenti verticali e *j* spostamenti orizzontali, per cui si richiede che non si possano effettuare più di *k* spostamenti verticali consecutivi:

```
(define manh          ; val: intero
  (lambda (i j k)     ; i, j, k: interi ≥ 0
    (manh-rec i j k k)
    ))
```

```
(define manh-rec
  (lambda (i j k u)
    (cond ((= i 0)
           1)
          ((= j 0)
           (if (> i u) 0 1))
          ((= u 0)
           (manh-rec i (- j 1) k k))
          (else
           (+ (manh-rec (- i 1) j k (- u 1))
              (manh-rec i (- j 1) k k)
              ))
          )))
```

Il programma impostato nel riquadro, ha invece l'obiettivo di restituire *la lista di tali percorsi*, ciascuno codificato da una stringa di `0` e `1`, dove lo zero rappresenta uno spostamento verticale (in basso) e l'uno rappresenta uno spostamento orizzontale (a destra). Esempi:

```
(paths 5 1 2)  →  ()
(paths 1 5 2)  →  ("011111" "101111" "110111" "111011" "111101" "111110")
(paths 2 2 1)  →  ("0101" "0110" "1010")
```

Completa la procedura ricorsiva `paths-rec` inserendo espressioni appropriate negli spazi indicati. (Nota: dati un intero *n* e un carattere *c*, la procedura predefinita `make-string` restituisce la stringa composta da *n* ripetizioni di *c*.)

```
(define paths                 ; val: lista di stringhe
  (lambda (i j k)             ; i, j, k: interi non negativi
    (paths-rec i j k k)
    ))

(define paths-rec
  (lambda (i j k u)
    (cond ((= i 0)
            ( ..................   (make-string j #\1)))
          ((= j 0)
            ...................................................... )
          ((= u 0)
            (map ..................................................
                 (paths-rec i (- j 1) k k)))
          (else
            ( ..................
             (map .................................................
                  (paths-rec (- i 1) j k (- u 1)))
             (map .................................................
                  (paths-rec i (- j 1) k k))))
          )
    ))
```

4. Verifica formale della correttezza

Considera la procedura `f` riportata qui a lato, il cui argomento b è una stringa non vuota costituita esclusivamente dalle cifre `0` e `1` e il cui ultimo carattere (quello più a destra) deve essere `1`.

Per ogni stringa t di lunghezza $k \geq 3$ composta da $k-3$ cifre `1` seguite dalla terna `011` si può dimostrare che:

$$\texttt{(f } t\texttt{)} \quad \rightarrow \quad 3 \cdot 2^{k-2} - 1 \quad (*)$$

```
(define f          ; val: intero
  (lambda (b)      ; b: stringa di 0/1 conclusa da 1
    (cond ((string=? b "1")
            1)
          ((char=? (string-ref b 0) #\0)
            (- (* 2 (f (substring b 1))) 1))
          (else
            (+ (* 2 (f (substring b 1))) 1))
          )
    ))
```

(In altri termini t potrà essere: `"011"`, `"1011"`, `"11011"`, `"111011"`, `"11 … 1011"`.)

Dimostra la proprietà (*) per induzione, attenendoti allo schema delineato qui sotto.

- Indica il valore rispetto al quale intendi impostare la dimostrazione per induzione:

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

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra caso/i base e passo induttivo: