Corso di Programmazione

Esame del 24 Gennaio 2011

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. Programmazione Dinamica

Considera il seguente metodo che rappresenta una funzione ricorsiva in *Java*:

```
public static int f( int i, int j ) {  // i, j >= 0

  if ( (i < 2) || (j < 2) ) {
    return i * j + 1;
  } else {
    return f( i-2, j ) + 2 * f( i-1, j-1 ) + f( i, j-2 );
  }
}
```

Applica una tecnica di *programmazione dinamica* per sviluppare una versione *non* ricorsiva del programma.

2. Procedure in Scheme

Con riferimento alla procedura `q` così definita:

```
(define q
  (lambda (x u)
    (let ((k (- (string-length x) 1))
          (f (lambda (z)
               (integer->char (+ (char->integer z) (if (char=? z #\0) u 0) -1))
               )))
      (let ((v (string-ref x k)) (y (substring x 0 k)))
        (string-append
         (cond ((char>? v #\0) y) ((string=? y "1") "") (else (q y u)))
         (string (f v)))
        ))))
```

calcola il risultato della valutazione di ciascuna delle seguenti espressioni Scheme:

`(q "1" 4)` → _____ `(q "11" 4)` → _____

`(q "2" 4)` → _____ `(q "30" 4)` → _____

`(q "3" 4)` → _____ `(q "200" 8)` → _____

`(q "10" 4)` → _____ `(q "1000" 10)` → _____

3. Procedure con valori procedurali

Date due stringhe `plain` ed `encrypted`, la procedura `permutation-rule` restituisce una funzione *f* che rappresenta una regola per crittare un testo sostituendone o permutandone i caratteri in base alle posizioni in cui compaiono nelle stringhe. Più precisamente, `plain` rappresenta l'alfabeto del testo in chiaro e contiene caratteri tutti diversi fra loro; `encrypted` ha la stessa lunghezza di `plain` e ciascuno dei suoi caratteri rappresenta la codifica, nel testo crittato, del carattere che ha la stessa posizione in `plain`. La funzione restituita ha come dominio l'insieme dei caratteri di `plain` e come codominio l'insieme dei caratteri di `encrypted`, entrambi di tipo *char*. Per esempio, sulla base della definizione

```
(define pr (permutation-rule "ABCDEFGHILMNOPQRSTVX" "DEFGHILMNOPQRSTVXABC"))
```

ne risultano le seguenti valutazioni (codice di *Giulio Cesare* applicato all'alfabeto *Latino*):

`(pr #\C)` → `#\F` `(pr #\A)` → `#\D` `(pr #\E)` → `#\H`

`(pr #\S)` → `#\X` `(pr #\V)` → `#\B` `(pr #\R)` → `#\V`

Completa la definizione della procedura `permutation-rule` riportata qui sotto, introducendo il codice Scheme appropriato negli spazi indicati a tratto punteggiato.

```
(define permutation-rule
  (lambda (plain encrypted)

    (  .....  (permutation-rec  .....  plain encrypted))
    ))

(define permutation-rec
  (lambda (x i plain encrypted)

    (if (char=?  ..... )
        (string-ref encrypted i)

        .....

        )))
```

4. Verifica formale della correttezza

```
(define magic          ; valore intero
  (lambda (x)          ; x > 0  intero
    (cond ((= x 1) 1)
          ((even? x)
           (- (* 2 (magic (quotient x 2))) 1))
          (else
           (+ (* 2 (magic (quotient x 2))) 3))
          )))
```

In relazione alla procedura definita sopra è possibile dimostrare che per tutti i valori interi positivi di k:

$$\texttt{(magic } 2^k-1\texttt{)} \quad \rightarrow \quad 2^{k+1}-3$$

Dimostra per induzione questa proprietà; in particolare:

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

- Formalizza l'ipotesi induttiva:

- Formalizza la proprietà da dimostrare come passo induttivo:

- Dimostra il caso / i casi base:

- Dimostra il passo induttivo:

5. Programmazione in Java

Definisci in Java un metodo statico `sort` che, dato un array di stringhe binarie, ne ordina le componenti per valore numerico rappresentato crescente. Si assume che le stringhe rappresentate in ciascuna delle componenti dell'array siano corrette nella notazione binaria: i soli caratteri utilizzati sono `0`, `1`, e la prima cifra è `1` con l'unica eccezione della rappresentazione del numero *zero*. Per esempio, la trasformazione riportata sotto illustra come le componenti dell'array, argomento del metodo `sort`, debbano essere riordinate:

```
{"101", "1011", "10", "111", "10000", "1110", "1", "100", "0", "110", "11" }
   →  { "0", "1", "10", "11", "100", "101", "110", "111", "1011", "1110", "10000" }
```